ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ/ ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ ΤΟΥ

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΦΟΙΤΗΤΗ ΑΝΤΩΝΙΟΥ ΜΙΧ ΜΑΡΑΓΚΟΥΔΑΚΗ

ΑΡ. ΜΗΤΡΩΟΥ: mtt13

ΘΕΜΑ: Review of Neural Networks and Deep Learning and Design and Implementation of Deep Learning Neural Network Backpropagation in CUDA C

ΕΠΙΒΛΕΠΟΝ ΚΑΘΗΓΗΤΗΣ: ΔΡ. ΑΝΤΩΝΙΟΣ ΚΩΝΣΤΑΝΤΑΡΑΣ

ΘΕΜΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

(ΕΛΛΗΝΙΚΑ) ΑΝΑΣΚΟΠΗΣΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΚΑΙ ΒΑΘΙΑΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΚΑΙ ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ ΒΑΘΙΑΣ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ ΜΕ ΧΡΗΣΗ CUDA C

(ΑΓΓΛΙΚΑ) REVIEW OF NEURAL NETWORKS AND DEEP LEARNING AND DESIGN AND IMPLEMENTATION OF A DEEP LEARNING NEURAL NETWORK USING CUDA C

Η εργασία αυτή είναι πρωτότυπη και εκπονήθηκε αποκλειστικά και μόνο για την απόκτηση του συγκεκριμένου μεταπτυχιακού.

ΠΕΡΙΛΗΨΗ ΘΕΜΑΤΟΣ ΕΡΓΑΣΙΑΣ

Η Μετάβαση απο τον σειριακό κώδικα σε παράλληλο σε συνδιασμό με εφαρμογές με μεγάλα δεδομένα γέννησε τον ετερογενή παράλληλο προγραμματισμό. Αυτός αξιοποιεί τόσο το CPU όσο και την κάρτα γραφικών αναθέτοντας στην τελευταία παράλληλες διεργασίες ενώ στο CPU τις σειριακές. Η τεχνολογία CUDA δηλαδή η παράλληλη αρχιτεκτονική των καρτών γραφικών επιτρέπει την εξομοίωση νευρωνικών δικτύων τα οποία επίσης χαρακτηρίζονται απο παράλληλη αρχιτεκτονική . Η παρούσα διπλωματική εργασία με χρήση CUDA C υλοποιεί την αρχιτεκτονική ενός σύνθετου νευρωνικού δικτύου βαθιάς μηχανικής μάθησης.

The transition from the serial code in parallel combined with applications with large data gave birth to the heterogeneous parallel programming. He leverages both the CPU and graphics card instructing the last parallel processes while the CPU serial. CUDA technology i.e. the parallel master-Masonic graphics card allows the simulation of neural networks which also featured from parallel architecture. The present thesis using CUDA C implements the architecture of a complex neural network of deep learning engineering.

ΕΙΣΑΓΩΓΗ

ΜΕΡΟΣ Α ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Το θέμα της Μετα -Πτυχιακής εργασίας εχει να κάνει με την Σχεδιάση και Υλοποίηση ενός Νευρωνικού Δικτύου Βαθιάς Μηχανικής Μάθησης χρησιμοποιώντας το λογισμικό CUDA C της NVIDIA. Συγκεκριμένα χρησιμοποιήθηκε το περιβάλλον Visual Studio 2013 professional μαζί με την εκδοση της CUDA C και συγκεκριμένα την cuda_8.0.61 windows

Η NVIDIA κατασκευάζει κάρτες γραφικών οι οποίες χρησιμοποιούνε πολλούς παράλληλους επεξεργαστές (graphics processing units –GPU) αριθμητικών υπολογισμών που προγραμματίζονται σε γλώσσα C ,Python με τις επεκτάσεις τους.

Στην αρχή η εταιρία κατασκέυαζε τις κάρτες γραφικών με το σκεπτικό να επεξεργάζονται τα δεδομένα γραφικών, αργότερα λόγω των υψηλών απαιτήσεων για υπολογιστική ισχύ την οποία δεν μπορούσαν οι συμβατικοί (Central Processing Units –CPU) και πόσο μάλλον για μαζική παράλληλη επεξεργασία μετατράπηκαν απο απλές παιχνιδομηχανές σε συστήματα μαζικά παράλληλης επεξεργασίας (massively parallel systems).

Το επόμενο βήμα ήταν το project CUDA (Compute Unified Device Architecture – Αρχιτεκτονική Υπολογιστών Ενοποιημένων Συσκευών) και αργότερα βγήκε και το αντίστοιχο λογισμικό που ήταν προσαρμοσμένο στην κάρτα γραφικών της NVIDIA. Το λογισμικό περιελάμβανε τους κατάλληλους οδηγούς drivers και τον προσαρμοσμένο μεταγλωττιστή nvcc compiler το οποίο ήταν σε θέση να μεταφράσει το πρόγραμμα που ήταν γραμμένο απο την γλώσσα προγραμματισμού C (C for CUDA) σε γλώσσα μηχανής PTX των επεξεργαστών της κάρτας γραφικών.

Το μοντέλο CUDA παρέχει υποστήριξη για πράξεις με αριθμούς κινητής υποδιαστολής διπλής ακρίβειας ,επίσης διαθέτει εργαλεία αποσφαλμάτωσης CUDA debugger και ανάλυσης και εκτέλεσης CUDA profiler [11] . Υποστηρίζει ασύγχρονη μεταφορά δεδομένων μεταξύ της μνήμης της κάρτας γραφικών (μνήμη συσκευής – device memory) και της κύριας μνήμης (host memory), καθώς και συμβατότητα της εκτέλεσης ενός τμήματος κώδικα τόσο στην CPU όσο και στην GPU [11], οπότε δεν χρειάζεται ο προγραμματιστής να γράφει δυο φορές τον κώδικα του προβλήματος για την περίπτωση που το πρόγραμμα θα εκτελεστεί και σε υπολογιστές που δεν έχουν συμβατή κάρτα γραφικών.

Η εργασία χωρίζεται σε τρεια μέρη:

ΜΕΡΟΣ Α ΘΕΩΡΙΑ ΠΑΝΩ ΣΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

ΜΕΡΟΣ Β ΘΕΩΡΙΑ ΠΑΝΩ ΣΕ ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

ΜΕΡΟΣ Γ ΚΩΔΙΚΑΣ ΣΕ CUDA C ΚΑΙ ΕΠΕΞΗΓΗΣΗ ΑΥΤΟΥ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ ΜΕΡΟΥΣ Α ΘΕΩΡΙΑ ΠΑΝΩ ΣΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ ΣΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ Σελίδα 6

1.1 Ο νευρώνας Σελίδα 7

1.2 Πλεονεκτήματα Τεχνητών Νευρωνικών Δικτύων Σελίδα 9

1.3 Μοντέλο του τεχνητού νευρώνα Σελίδα 11

1.4 Αρχιτεκτονικές Νευρωνικών Δικτύων Σελίδα 17

1.5 Εκπαίδευση Τεχνητών Νευρωνικών Δικτύων Σελίδα 25

1.6 Τρόποι Νευρωνικής Μάθησης Σελίδα 31

ΚΕΦΑΛΑΙΟ 2 ΠΕΡΙΓΡΑΦΗ ΑΠΛΟΥ ΜΟΝΤΕΛΟΥ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗ Σελίδα 33

2.1 Το μοντέλο του αισθητήρα ή αλλιώς Perceptron του Rosenblatt Σελίδα 33

2.2 Εκπαίδευση και μειονεκτήματα του δικτύου Perceptron Σελίδα 37

ΚΕΦΑΛΑΙΟ 3 ΜΕΘΟΔΟΙ ΚΑΙ ΤΡΟΠΟΙ ΕΚΠΑΙΔΕΥΣΗΣ Σελίδα 41

3.1 Το Perceptron πολλών επιπέδων Σελίδα 41

3.2 Πολυεπίπεδο δίκτυο Perceptron – MLP Σελίδα 41

3.3 Ο αλγόριθμος της ανάδρομης διάδοσης σφάλματος ή backpropagation Σελίδα 43

3.4 Η παράμετρος του βήματος εκπαίδευσης β ή ρυθμός μάθησης Σελίδα 49

3.5 Μειονεκτήματα του αλγόριθμου Back Propagation – Αρχικοποίηση των βαρών *Σελίδα 51*

3.6 Το πρόβλημα της XOR *Σελίδα 52*

3.7 Τρόποι εκπαίδευσης ενός δικτύου MLP *Σελίδα 56*

3.8 Βελτίωση της απόδοσης του αλγόριθμου BP με ευρετικές μεθόδους *Σελίδα 58*

ΚΕΦΑΛΑΙΟ 4 *ΣΥΝΕΛΙΚΤΙΚΑ ΔΙΚΤΥΑ* Σελίδα 67

4.1 Συνελικτικά Δίκτυα (Convolutional Networks) *Σελίδα 67*

ΑΝΑΦΟΡΕΣ Σελίδα 98

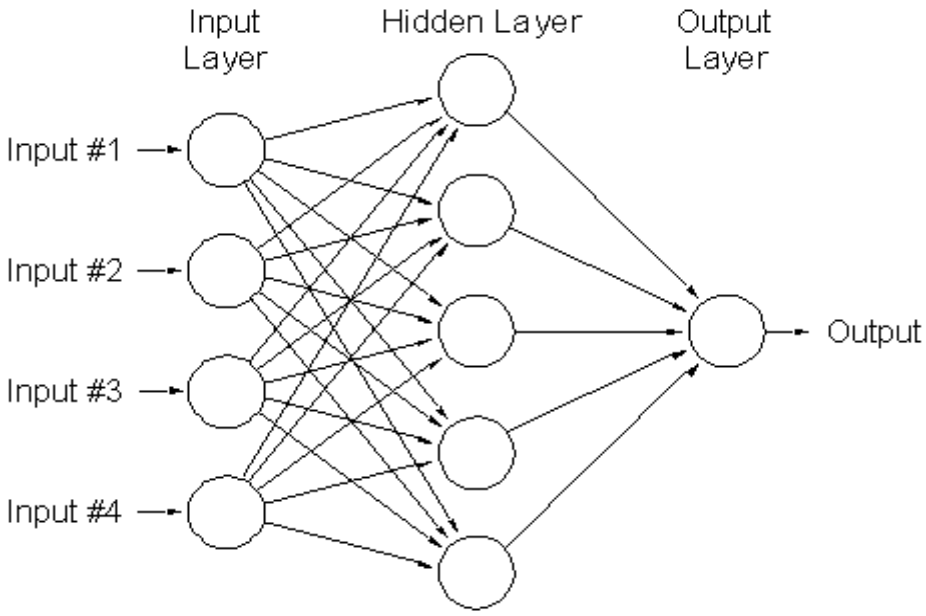
ΣΗΜΕΙΩΣΕΙΣ - ΠΑΡΑΠΟΜΠΕΣ Σελίδα 99

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ ΣΕ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Τα νευρωνικά δίκτυα βασίζονται λίγο πολύ στην δομή και λειτουργία του ανθρώπινου εγκεφάλου. Βασικό δομικό στοιχείο του εγκεφάλου είναι οι νευρώνες, λέγονται και βιολογικοί νευρώνες ενώ στο νευρωνικό δίκτυο έχουμε τους τεχνητούς νευρώνες που είναι και η δομική μονάδα. Νευρώνες λοιπόν είναι τα νευρικά κύτταρα τα οποία δημιουργούν ενα πυκνό δίκτυο επικοινωνίας μεταξύ τους [1]. Κάθε νευρώνας δέχεται μια/την πληροφορία , την επεξεργάζεται και δίνει μια τιμή εξόδου [2]. Οι εισόδου του είναι είτε οι εξόδοι άλλων νευρώνων , είτε το πρωταρχικό σήμα εισόδου του δικτύου [2]. Βασικά το κίνητρο για την μελέτη του νευρώνα και των νευρωνικών δικτύων είναι η ελπίδα ανακάλυψης ενός νέου υπολογιστικού μοντέλου βασισμένου σε μια δικτυακή δομή παρόμοια με αυτή του ανθρώπινου εγκεφάλου. Αυτή η πλατφόρμα λέγεται connectional model που θα είναι κατάλληλη για ανάπτυξη ευφυών αλγορίθμων και διαδικασιών που θα σχετίζονται με την νοημοσύνη την μάθηση την μνήμη την γενίκευση την ομαδοποίηση προτύπων κτλ [1]. Τα τεχνητά νευρωνικά δίκτυα χρησιμοποιηούν πιο απλά μοντέλα νευρώνων έτσι ώστε να διατηρούν τα πολύ απλά χαρακτηριστικά των μοντέλων που χρησιμοποιηούνται στην νευρολογία [1]. Οι νευρώνες έχουν δυο βασικά χαρακτηριστικά:

(α) θα πρέπει να έχουν ρυθμιζόμενες παραμέτρους ώστε να διευκολύνεται η διαδικασία της μάθησης δηλαδή να έχουν μια πλαστικότητα οι νευρώνες [1].

(β) ενα δίκτυο που αποτελείται απο ενα μεγάλο πλήθος νευρώνων πετυχαίνει παραλληλισμό της επεξεργασίας και κατανομή της πληροφορίας [1].



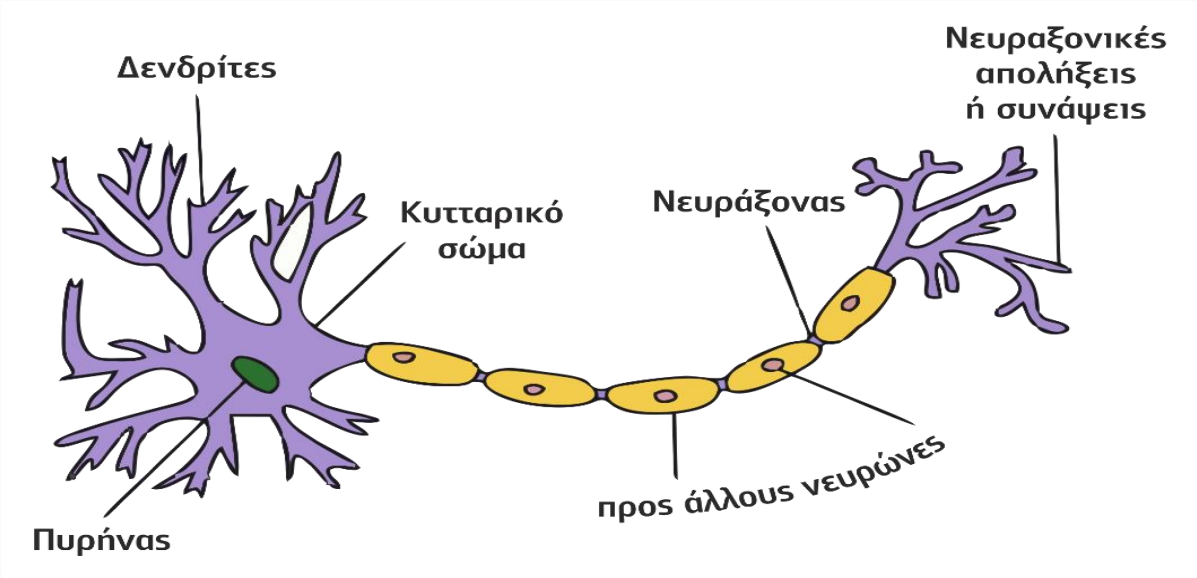
Σχήμα 1.1

πηγή

http://www.astroml.org/book_figures/appendix/fig_neural_network.html

1.1 Ο νευρώνας

Είναι το βασικό δομικό στοιχείο του εγκεφάλου στους ανθρώπους αλλά και στα ζώα. Ο νευρώνας αποτελείται απο τα εξής τμήματα: (α) το σώμα, (β) τους δενδρίτες, (γ)τον άξονα, και (δ) τις συνάψεις που συνδέουν τις διακλαδώσεις του άξονα με τους δενδρίτες άλλων νευρώνων δημιουργώντας ένα νευρωνικό δίκτυο [1].



Σχήμα 1.1.1

πηγή

http://repfiles.kallipos.gr/html_books/93/04a-main.html

Τα τμήματα του νευρώνα παίζουν διαφορετικούς ρόλους:

- Οι δενδρίτες είναι οι πύλες εισόδου του νευρώνα. Δέχονται ηλεκτρικά σήματα απο άλλους νευρώνες [1].
- Ο αξονας είναι η πύλη εξόδου του νευρώνα , στέλνει σήματα προς άλλους νευρώνες με μορφή ηλεκτρικών παλμών σταθερού πλάτους αλλά μεταβλητής συχνότητας [1].
- Οι συνάψεις ή απολήξεις είναι τα σημεία ένωσης μεταξύ των διακλαδώσεων του άξονα ενός νευρώνα και των δενδριτών απο άλλους νευρώνες [1]. Το πλάτος της σύναψης, η απόσταση της απο τον δενδρίτη και η πυκνότητα του ηλεκτροχημικού υλικού που έχουν οι συνάψεις επηρεάζουν την ευκολία με την οποία η ηλεκτρική δραστηριότητα μεταδίδεται απο τον άξονα στο δενδρίτη [1]. Μπορούμε να πούμε ότι οι συνάψεις είναι κύστες με ηλεκτροχημικό υλικό ιόντα , κυρίως Νάτριο και Κάλιο [3]. Το υλικό αυτό μεταδίδει την ηλεκτρική δραστηριότητα του αξονα αποστολέα στους δενδρίτες παραλήπτες. Το ποσοστό της ηλεκτρικής δραστηριότητας που μεταδίδεται τελικά στον δενδρίτη λέγεται συναπτικό βάρος [1][3]. Οι συνάψεις χωρίζονται σε ενισχυτικές και ανασταλτικές ανάλογα με το αν το φορτίο που εκλύεται απο την σύναψη ερεθίζει τον νευρώνα για να παράγει παλμούς με μεγαλύτερη συχνότητα ή αν τον καταστέλλει εμποδίζοντάς τον να παράγει παλμούς [1]. Το πιο βασικό χαρακτηριστικό του νευρώνα είναι η ευαισθησία του, δηλαδή η ικανότητά να αντιδρά σε διάφορα εξωτερικά ερεθίσματα

(ηλεκτρομαγνητικά ,χημικά, θερμικά, μηχανικά κτλ.) [3]. Αποτέλεσμα αυτού η παραγωγή παλμών μικρής διάρκειας.
Οι παλμοί αυτοί είναι οι φορείς πληροφορίας στα Βιολογικά Νευρωνικά Δίκτυα, που ταξιδεύουν στον άξονα κάθε νευρώνα και μέσω των συνάψεων διαδίδονται στους δενδρίτες των παραληπτών νευρώνων [1]. Ο κάθε νευρώνας συλλέγει όλο το ηλεκτρικό φορτίο που δέχεται απο κάθε σύναψη στους δενδρίτες του. Όσο πιο ισχυρή είναι μια συναπτική ζεύξη τόσο πιο πολύ έντονα συμμετέχει το συγκεκριμένο φορτίο εισόδου στο συνολικό άθροισμα φορτίου [3]. Αν το άθροισμα του φορτίου ξεπερνάει κάποιο κατώφλι τότε ο άξονας του νευρώνα αρχίζει να παράγει ηλεκτρικούς παλμούς με μεγάλη συχνότητα οπότε λέμε οτι ο νευρώνας πυροβολεί [3] [1]. Αν το φορτίο δεν περνάει το συγκεκριμένο αυτό όριο τότε ο νευρώνας παράγει πολύ αραιά παλμούς σε τυχαίες στιγμές οπότε λέμε ότι ο νευρώνας είναι αδρανής [1]. Μπορούμε να πούμε ότι ο νευρώνας είναι δυαδικό στοιχείο [3].

Κάθε παλμός έχει συγκεκριμένο χρονικό πλάτος tp και μετά απο κάθε παλμό ο νευρώνας χρειάζεται ένα ελάχιστο χρόνο ανάπαυσης tr. Ετσι ο μέγιστος ρυθμός των παλμών δεν ξεπερνάει το όριο [1].

Firing frequency < 1 (tp + tr)

Τελικά οι παλμοί που παράγονται ταξιδεύουν κατά μήκος του άξονα και τροφοδοτούν τους άλλους νευρώνες με τους οποίους συνδέεται ο αρχικός νευρώνας.

Οι αριθμοί των μονάδων των νευρώνων και οι συνδέσεις τους στον ανθρώπινο εγκέφαλο ξεπερνάει το 10^11 . Ενας νευρώνας λαμβάνει χαρακτηριστικά μεταξύ 5000 και 15000 εισαγωγών απο άλλους νευρώνες [4]. Αυτοι οι αριθμοί είναι κατά πολύ μεγαλύτεροι απο τους αριθμούς που μπορεί να χειριστεί έυκολα ένα υπολογιστής [3]. Και αυτό διότι ο ανθρώπινος εγκέφαλος είναι τόσο πολύπλοκος όσο και οι διεργασίες που επιτελεί. Ενα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) μπορεί να έχει μερικές εκατοντάδες ή χιλιάδες νευρώνες αλλά όχι της τάξης μεγέθους που έχει ο ανθρώπινος εγκέφαλος [3]. Απο την άλλη τα ΤΝΔ μπορούν να λύσουν δύσκολα μαθηματικά προβλήματα, όπως η αναγνώριση συστήματος, η πρόβλεψη κτλ., στα οποία ο ανθρώπινος εγκέφαλος δεν τα καταφέρνει και ιδιαίτερα καλά [3]. Ο ανθρώπινος εγκέφαλος έχει πιο πολλές συνάψεις αλλα και η συνδεσμολογία τους είναι πιο περίπλοκη απο τα ΤΝΔ. Η ταχύτητα στους υπολογιστές είναι χιλιάδες φορές μεγαλύτερη απο την ταχύτητα διάδοσης του σήματος στα Βιολογικά Νευρωνικά Δίκτυα [3]. Η διαφορά στην ταχύτητα δεν επαρκεί για να καλύψει την διαφορά στην πολυπλοκότητα . Ο ανθρώπινος εγκέφαλος μαθαίνει και καταλαβαίνει πολύ γρήγορα , η μάθηση στα ΤΝΔ παίρνει πολύ χρόνο , ακόμα και στον πιο γρήγορο υπολογιστή [3]. Επίσης ο εγκέφαλος μπορεί να κάνει σύγχρονη ή ασύγχρονη ενημέρωση των μονάδων του , ενώ ένα ΤΝΔ κάνει μόνο σύγχρονη ενημέρωση [3].

1.2 Πλεονεκτήματα Τεχνητών Νευρωνικών Δικτύων

Την υπολογιστική ισχύ ενα ΤΝΔ την οφείλει στην παράλληλη ,κατανεμημένη δομή του και στην ικανότητά του να μαθαίνει και να γενικεύει. Τι ονομάζουμε όμως γενίκευση; Γενίκευση λοιπόν λέμε ότι είναι η παραγωγή για το ΤΝΔ λογικών εξόδων για εισόδους τις οποίες δεν έχει συναντήσει κατά την διάρκεια της εκπαίδευσής του [3]. Οι δυνατότητες αυτές βοηθάνε τα ΤΝΔ να βρίσκουνε καλές προσεγγιστικές λύσεις σε δύσκολα και πολύπλοκα προβλήματα, τα οποία ίσως να μην καταλήγουν σε λύσεις. Τα Νευρωνικά Δίκτυα έχουν τις ακόλουθες χρήσιμες ιδιότητες [3].

1. Μη γραμμικότητα. Ενας τεχνητός νευρώνας μπορεί να είναι είτε γραμμικός, είτε μη γραμμικός. Ενα ΤΝΔ που εχει μη γραμμικούς νευρώνες διασυνδεδεμένους είναι μη γραμμικό. Σαν ιδιότητα είναι πολύ σημαντικό διότι για παράδειγμα ο μηχανισμός που παράγει το σήμα εισόδου (ομιλία) είναι μη γραμμικός [3].

2. Αντιστοίχιση Εισόδου-Εξόδου. Παράδειγμα μάθησης, η μάθηση με εκπαιδευτή ή επιβλεπόμενη μάθηση, οδηγεί στην τροποποίηση των συναπτικών βαρών ενός Νευρωνικού Δικτύου με την βοήθεια παραδειγμάτων εκπαίδευσης [3]. Εστω για παράδειγμα σε ενα δίκτυο έχουμε ένα μοναδικό σήμα εισόδου και μια αντίστοιχη επιθυμητή έξοδο/απόκριση (στόχο) [3]. Τα συναπτικά βάρη τροποποιούνται έτσι ώστε να ελαχιστοποιηθεί η διαφορά μεταξύ της επιθυμητής και της πραγματικής απόκρισης του δικτύου [3]. Κάποια στιγμή το δίκτυο φτάνει σε μια κατάσταση όπου δεν υπάρχουν άλλες αλλαγές βαρών.

3. Παράλληλος Τρόπος Λειτουργίας. Τα Νευρωνικά Δίκτυα λειτουργούν με παράλληλο τρόπο διότι μια εργασία μοιρά-ζεται στα διάφορα τμήματα του δικτύου ,στους επιμέρους νευρώνες [3]. Αρα μπορούμε να πούμε ότι τα Νευρωνικά Δίκτυα είναι συστήματα παράλληλων κατανεμημένων διεργασιών ,τα οποία έχουνε υψηλές ταχύτητες σαν να έχουμε ταυτόχρονα πολλούς επεξεργαστές [3].

4. Προσαρμοστικότητα . Τα Νευρωνικά Δίκτυα έχουν την δυνατότητα να προσαρμόζουνε τα συναπτικά τους βάρη ανάλογα με τις μεταβολές που γίνονται στο περιβάλλον τους [3]. Ενα ΤΝΔ που έχει εκπαιδευτεί σε ενα συγκεκριμένο περιβάλλον μπορεί να επανεκπαιδευτεί έτσι ώστε να μπορεί να χειρίζεται τις μεταβολές στο νέο πλέον περιβάλλον λειτουργίας του [3].

5. Ανοχή σε βλάβες. Ενα Νευρωνικό Δίκτυο που είναι υλοποιημένο σε hardware μπορεί να είναι ανεκτικό σε βλάβες και η απόδοσή του να μειώνεται βαθμιαία σε αντίξοες συνθήκες λειτουργίας [3]. Σε περίπτωση που ένας νευρώνας ή σύνδεση καταστραφούν η ποιότητα της εξόδου μειώνεται [3]. Επειδή η πληροφορία που αποθηκεύεται στο δίκτυο είναι κατανεμημένη σε όλη τη δομή του δικτύου ,για να μειωθεί αρκετά η συνολική απόκριση του δικτύου θα πρέπει να βλάβη να πάρει μεγάλη έκταση.

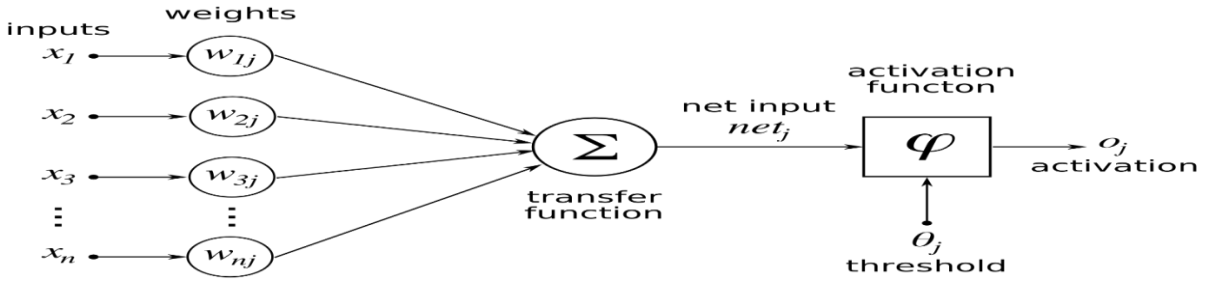
6. Ομοιομορφία ανάλυσης και σχεδίασης. Τα Νευρωνικά Δίκτυα χρησιμοποιηούνται πάρα πολύ σαν επεξεργαστές πληροφοριών [3]. Οι νευρώνες ουσιαστικά είναι ενα

κοινό συστατικό σε όλα τα Νευρωνικά Δίκτυα και αυτό μας επιτρέπει να εφαρμόσουμε τις ίδιες θεωρίες και αλγόριθμους μάθησης σε διαφορετικές εφαρμογές των Νευρωνικών Δικτύων.

7. Αναλογία με τη νευροφυσιολογία του εγκεφάλου. Για την σχεδίαση ενός Νευρωνικού Δικτύου παίρνουμε στοιχεία απο την λειτουργία του ανθρώπινου εγκεφάλου όπως η παράλληλη επεξεργασία που είναι γρήγορη και ισχυρή [3]. Τα Τεχνητά Νευρωνικά Δίκτυα όπως και τα βιολογικά έχουνε μεγάλη ανοχή σε δομικά σφάλματα [3]. Οπότε εάν καταστραφεί ένας νευρώνας ή κάποιος σύνδεσμος δεν θα δημιουργήσει ιδιαίτερο πρόβλημα στο δίκτυο διότι η πληροφορία που έχει είναι κατανεμημένη σε όλο το δίκτυο. Το μέγεθος του σφάλματος είναι ανάλογο του ποσοστού των κατεστραμμένων συνδέσεων [3].

8. Η δυνατότητα θεώρησής τους ως κατανεμημένη μνήμη και ως μνήμη συσχέτισης. Το ότι τα ΤΝΔ χαρακτηρίζονται ως κατανεμημένα έγκειτε στο γεγονός ότι η κωδικοποίηση που δημιουργούν είναι κατανεμημένη σε όλα τα βάρη της συνδεσμολογίας τους [3]. Το ίδιο μπορούμε να πούμε και για την μνήμη συσχέτισης. Η πληροφορία που αποθηκεύεται απο την μνήμη συσχέτισης γίνεται μέσω κατάλληλων συσχετίσεων που δημιουργεί απο τα δεδομένα εκπαίδευσης [3]. Τώρα όσον αφορά για την ανάκληση της πληροφορίας γίνεται με βάση το περιεχόμενο και όχι την διεύθυνσή της όπως γίνεται με τον ανθρώπινο εγκέφαλο. Αυτή η οργάνωση κάνει ορισμένα ΤΝΔ πολύ ανεκτικά σε μικρές αλλαγές στα σήματα εισόδου [3]. Αρα παράγουν σωστή έξοδο ακόμα και αν τα δεδομένα εισόδου είναι λίγο διαφορετικά για παράδειγμα λόγω θορύβου [3].

9. Η ικανότητά τους για αναγνώριση προτύπων. Τα ΤΝΔ έχουν εξαιρετική ικανότητα αναγνώρισης προτύπων καθώς δεν επηρεάζεται απο ελλιπή η και με θόρυβο δεδομένα[3]. Οταν ένα ΤΝΔ εκπαιδευτεί για να αναγνωρίζει συνθήκες και καταστάσεις , θέλει ενα κύκλο λειτουργίας για να προσδιορίσουν μια συγκεκριμένη κατάσταση. Αυτή η ιδιότητα το κάνει ιδανικό για αυτοματισμούς που μπορούν να λειτουργήσουν σε αντίξοες συνθήκες όπως σε πεδία μάχης ,σε χώρους με ραδιενέργεια [3].



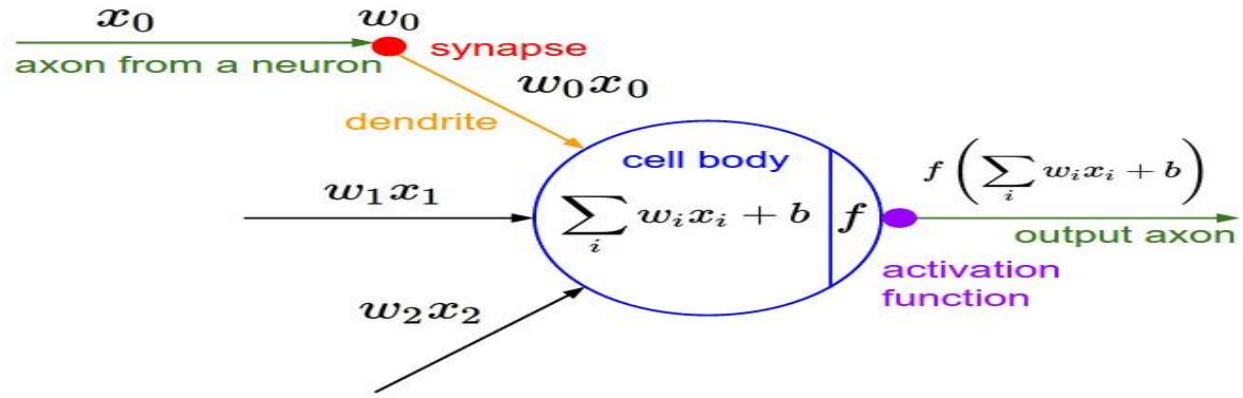
Σχήμα 1.2

πηγή

https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html

1.3 Μοντέλο του τεχνητού νευρώνα

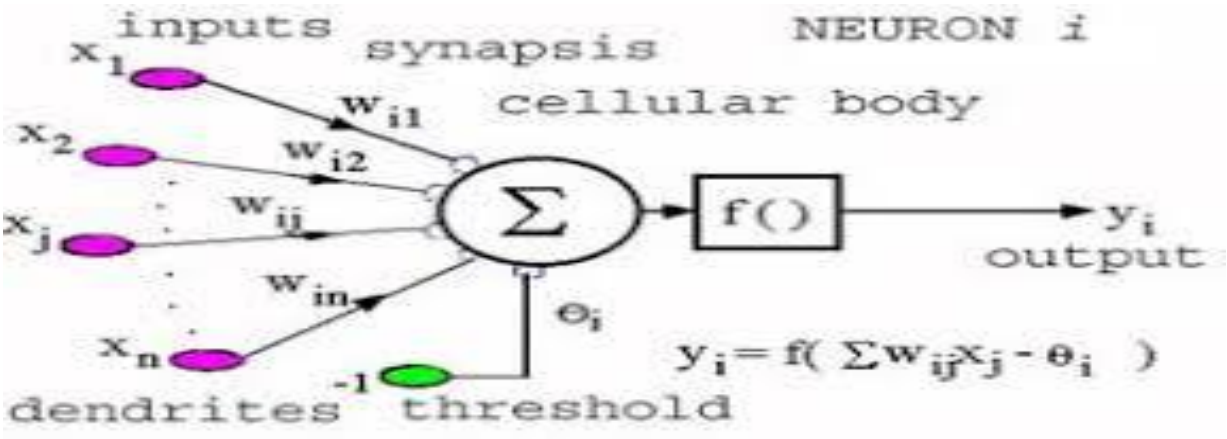
Ο Τεχνητός νευρώνας είναι ενα υπολογιστικό μοντέλο του οποίου τα μέρη μπορεί να αντιστοιχούν με εκείνα του βιολογικού νευρώνα [3]. Ο τεχνητός νευρώνας λοιπόν δέχεται σαν σήματα εισόδου τα x1, x2, x3, ...,xn . καθένα απο αυτά τα σήματα εισόδου μεταβάλλονται απο μια τιμή βάρους wi (weight) που ο ρόλος τους είναι αντίστοιχος του ρόλου του βιολογικού νευρώνα [3]. Η τιμή του βάρους μπορεί να είναι είτε θετική είτε αρνητική δηλαδή επιταχυντική ή επιβραδυντική αντίστοιχα λειτουργία της σύναψης [3]. Παρακάτω έχουμε ενα μοντέλο τεχνητού νευρώνα. Το σώμα του τεχνητού νευρώνα μπορούμε να το χωρίσουμε σε δυο μέρη, τον αθροιστή (sum), ο οποίος προσθέτει τα πολλαπλασιασμένα απο τα βάρη σήματα εισόδου παίρνοντας την ποσότητα S και την συνάρτηση ενεργοποίησης ή δραστηριοποίησης. Η συνάρτηση αυτή είναι ένα είδος φίλτρου το οποίο διαμορφώνει την τελική τιμή του σήματος εξόδου y, σε συνάρτηση με την ποσότητα S και την τιμή κατωφλίου της συνάρτησης ενεργοποίησης [3].



Σχήμα 1.3.1

πηγή

https://www.imagenesmi.com/im%C3%A1genes/neural-model-78.html



Σχήμα 1.3.2

πηγή

users.civil.ntua.gr

Ενα μοντέλο τεχνητού νευρώνα αποτελείται απο τα εξής στοιχεία:

1. Ενα σύνολο συνάψεων ,κάθε μια απο τις οποίες χαρακτηρίζεται απο τα δικά της βάρη. Πιο αναλυτικά έχουμε ένα σήμα xj στην είσοδο της σύναψης j που συνδέεται με τον νευρώνα k πολλαπλασιάζεται με το συναπτικό βάρος wkj [3]. Ο δείκτης k αναφέρεται στον νευρώνα , ενώ ο δείκτης j αναφέρεται στην είσοδο της σύναψης [3]. Οταν είναι ενισχυτική σύναψη τότε είναι θετικός αριθμός , ενώ όταν είναι ανασταλτική σύναψη τότε είναι αρνητικός αριθμός.
2. Το επόμενο είναι ο αθροιστής για την άθροιση των σημάτων εισόδου , πολλαπλασιασμένων με τα αντίστοιχα συναπτικά βάρη του νευρώνα. Γραμμική άθροιση.
3. Και τέλος μια συνάρτηση ενεργοποίησης η οποία είναι ένας μη γραμμικός μετασχηματιστής για τον περιορισμό του πλάτους της εξόδου του νευρώνα σε κάποια πεπερασμένη τιμή [3]. Συνήθως το έυρος τιμών πλάτους της εξόδου ενός νευρώνα είναι το διάστημα [0,1]ή [-1,1].

Μην ξεχάσουμε να αναφέρουμε ότι το μοντέλο του νευρώνα περιλαμβάνει και μια εξωτερικά εφαρμοζόμενη πόλωση (bias) bk. Αυτή η πόλωση έχει ως αποτέλεσμα την αύξηση ή την μείωση της δικτυακής διέγερσης της συνάρτησης ενεργοποίησης ανάλογα με το αν είναι η συνάρτηση ενεργοποίησης θετική ή αρνητική αντίστοιχα [3]. Αυτή η πόλωση bk είναι ενας εξωγενής παράμετρος του νευρώνα και ισούται με το συναπτικό βάρος wk0 της σταθερής εισόδου x0=1 [3]. Ποιες είναι οι μαθηματικές εξισώσεις που περιγράφουν τον νευρώνα; Είναι οι κάτωθι:

u = $\sum_j^m wkj\ xj$ και yk = f(uk) (Σχέση 1.3.1)

Η συνάρτηση ενεργοποίησης συμβολίζεται με f(uk) και είναι η έξοδος ενός νευρώνα του τοπικού πεδίου u. Εάν το άθροισμα u είναι μεγαλύτερο απο το κατώφλι (threshold) θ τότε ο νευρώνας πυροβολεί ,αλλιώς παραμένει αδρανής.

Τώρα η σχέση παίρνει την παρακάτω μορφή:

u = $\sum_j^m wkj\ xj$ – θ (Σχέση 1.3.2)

επίσης η σχέση: yk = f(uk) γίνεται: y = f(u – θ)

όπου η f(.) είναι η λεγόμενη βηματική συνάρτηση.

Υπάρχουν πολλά είδη συναρτήσεων αλλά οι πιο βασικές είναι:

1. *Βηματική συνάρτηση 0/1 (step function) :*

$$f(u) = \begin{cases} 0, & u \leq 0 \\ 1, & u > 0 \end{cases}$$

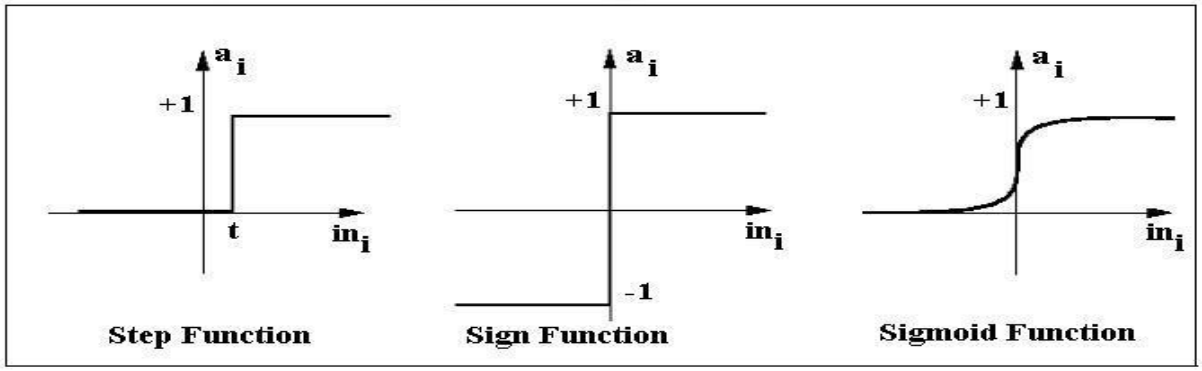
Αυτό το μοντέλο παριστάνεται απο ένα αθροιστή ακολουθούμενο απο ένα μη γραμμικό μετασχηματιστή f.

Το κατώφλι θ που ορίσαμε πιο πάνω είναι ένας πραγματικός (θετικός η αρνητικός αριθμός) όπως και τα συναπτικά βάρη.

Μπορούμε να θεωρήσουμε το κατώφλι θ σαν ένα συναπτικό βάρος συνδεδεμένο σε μια είσοδο x0 και να έχει μια τιμή -1 ,οπότε και η σχέση μπορεί να γίνει [1]:

u = $\sum_{j=1}^{m} wkj\ xj$ – θ = $\sum_{j=1}^{m} wkj\ xj$ (Σχέση 1.3.3)

όπου w0=θ και x0 = -1.



Σχήμα 1.3.3

πηγή

https://stackoverflow.com/questions/33582251/can-an-ann-2-2-1-layers-be-implemented-to-learn-xor-using-linear-activation-fu

2. *. Βηματική συνάρτηση -1/1 (step function or sign function):*

$$f(u) = \begin{cases} -1, & u \leq 0 \\ 1, & u > 0 \end{cases}$$

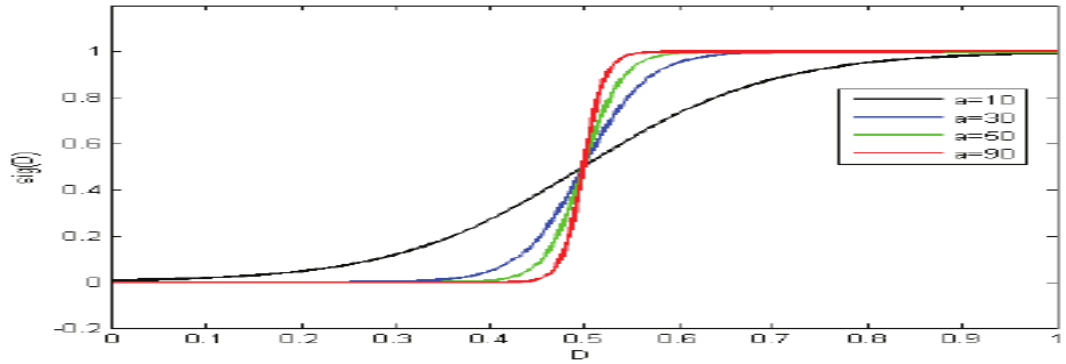
3. Σιγμοειδής συνάρτηση (Sigmoid function)

Η σιγμοειδής συνάρτηση είναι πιο γνωστή συνάρτηση για την κατασκευή ΤΝΔ. Είναι μια γνησίως αύξουσα συνάρτηση που είναι ομαλή και ασυμπτωτική [5]. Η παρακάτω συνάρτηση είναι ένα παράδειγμα αυτού.

f(u) = 1 /(1+e^-αu) (Σχέση 1.3.4)

οπου u είναι η έξοδος του νευρώνα η αλλιώς διέγερση

όπου α είναι η παράμετρος κλίσης της σιγμοειδής συνάρτησης. Εάν μεταβληθεί το α μεταβάλλεται η κλίση της σιγμοειδούς όπως φαίνεται και στο σχήμα.



Σχήμα 1.3.4

πηγή

https://www.researchgate.net/figure/Sigmoid-function-with-different-values-of-a-and_fig2_284063119

Παρατηρούμε για διάφορες τιμές του α με διαφορετικό χρώμα την κλίση της σιγμοειδούς συνάρτησης απο την αρχή των αξόνων μέχρι το απειρο.

Μια άλλη συνάρτηση που χρησιμοποιείται εναλλακτικά και μοιάζει με την βηματική συνάρτηση -1/1 είναι η υπερβολική εφαπτόμενη (hyperbolic tangent):

$f(u) = \tanh(u) = \frac{1-e^{-u}}{1+e^{-u}}$ (Σχέση 1.3.5)

Hyperbolic sine $\sinh x = \frac{e^x - e^{-x}}{2}$

Hyperbolic cosine $\cosh x = \frac{e^x + e^{-x}}{2}$

Hyperbolic tangent $\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$\tanh(u) = (e^{u} - e^{-u})/(e^{u} + e^{-u})$ (Σχέση 1.3.6)

Η υπερβολική εφαπτόμενη είναι επίσης παραγωγίσιμη και έχει απλή σχέση με τη σιγμοειδή συνάρτηση:

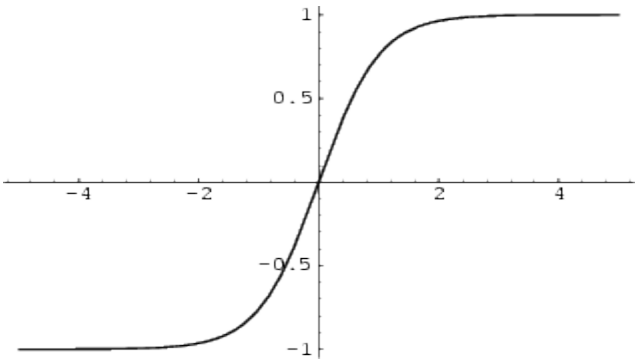
$$\tanh(u) = \frac{2f}{2u} - 1 \text{ (Σχέση 1.3.7)}$$

Ο γενικός τύπος της σιγμοειδούς συνάρτησης είναι ο παρακάτω:

f(u) = 1 /(1+e^-u) (Σχέση 1.3.8)

4. Υπερβολική εφαπτόμενη (hyperbolic tangent)

f(u) = tanh(u) = (1 – e^-u) (1 + e^-u) (Σχέση 1.3.9)



Σχήμα 1.3.5

πηγή

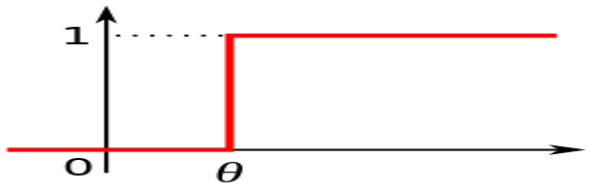
https://archive.lib.msu.edu/crcmath/math/math/h/h431.htm

5. Συνάρτηση κατωφλίου (threshold function)

f(u) = 0 αν u<= 0

f(u) = u αν 0<u<1

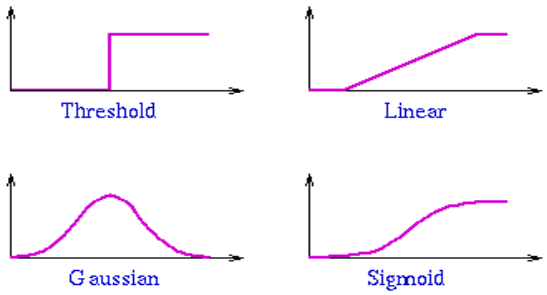
f(u) = 1 αν u>=1



Σχήμα 1.3.6

πηγή

https://en.wikipedia.org/wiki/File:Threshold_function.svg



Σχήμα 1.3.7

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Σχήμα 1.3.8

πηγή

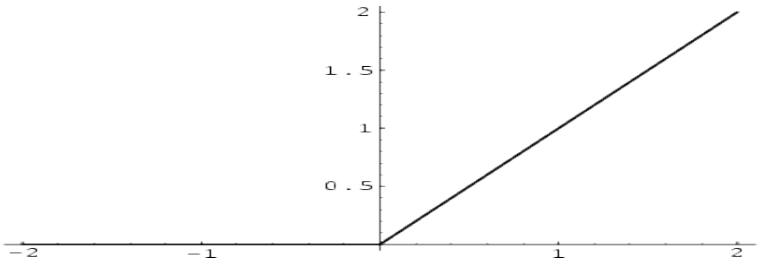
http://staff.itee.uq.edu.au/janetw/cmc/chapters/Introduction/

https://www.quora.com/What-is-signum-function-what-is-its-uses-in-machine-learning-neural-networks

6. Συνάρτηση ράμπας (ramp function)

f(u) = 0 αν u<= 0

f(u) = u αν u>0



Σχήμα 1.3.9

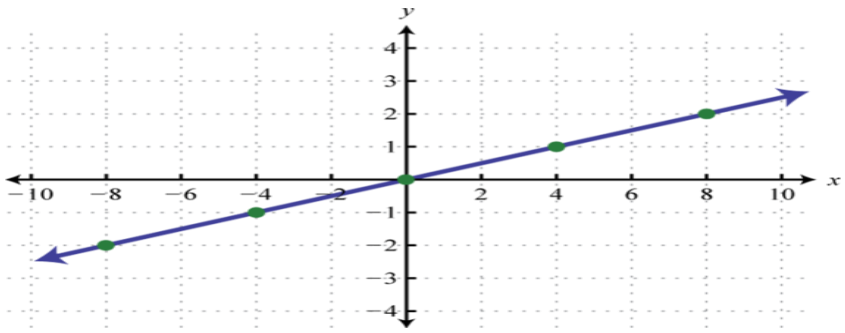
πηγή

https://archive.lib.msu.edu/crcmath/math/math/r/r064.htm

7. Γραμμική (linear function)

Κυρίως χρησιμοποιείται σε νευρώνες που προορίζονται για γραμμική προσέγγιση στα γραμμικά φίλτρα.

f(u) = u



Σχήμα 1.3.10

πηγή

https://saylordotorg.github.io/text_intermediate-algebra/s05-02-linear-functions-and-their-gra.html

1.4 Αρχιτεκτονικές Νευρωνικών Δικτύων

Η τοπολογία ενός ΤΝΔ Τεχνητού Νευρωνικού Δικτύου αναφέρεται στην διάταξη σύμφωνα με την οποία διασυνδέονται οι νευρώνες του δικτύου. Οι δυο βασικές ιδιότητες που καθορίζουν την αρχιτεκτονική ενος ΤΝΔ είναι το πλήθος των επιπέδων (layers) και οι συνδέσεις μεταξύ των νευρώνων [5]. Ενα άλλο χαρακτηριστικό το οποίο σχετίζεται με

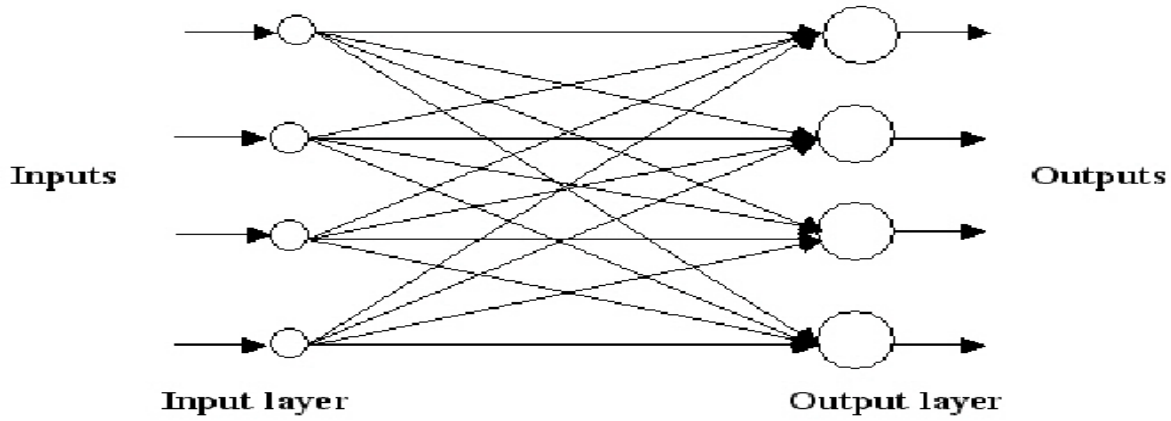
τον τρόπο κατα τον οποίο είναι δομημένοι οι νευρώνες , είναι ο αλγόριθμος μάθησης που χρησιμοποιείται για την εκπαίδευση του δικτύου.

Υπάρχουν τρεις διαφορετικές κλάσεις δικτυακών αρχιτεκτονικών [5]:

- Δίκτυα εμπρόσθιας τροφοδότησης ενός επιπέδου (Single Layer Feedforward networks)
- Δίκτυα εμπρόσθιας τροφοδότησης πολλαπλών επιπέδων (Multilayer Feedforward Networks)
- Αναδρομικά Τεχνητά Νευρωνικά Δίκτυα (Recurrent Neural Networks)

Δίκτυα εμπρόσθιας τροφοδότησης ενός επιπέδου (Single Layer Feedforward networks)

Σε ένα νευρωνικό δίκτυο οι νευρώνες οργανώνονται σε μορφή επιπέδων ,στην πιο απλή μορφή έχουμε ενα α επίπεδο με κόμβους εισόδου που μας οδηγεί σε ένα επίπεδο εξόδου με νευρώνες (υπολογίσημοι κόμβοι) ,ενώ το αντίθετο δεν ισχύει, δηλαδή απο το τέλος στην αρχή δεν πάμε [5]. Είναι εμπρόσθιας τροφοδότησης δίκτυο.



Σχήμα 1.4.1

πηγή

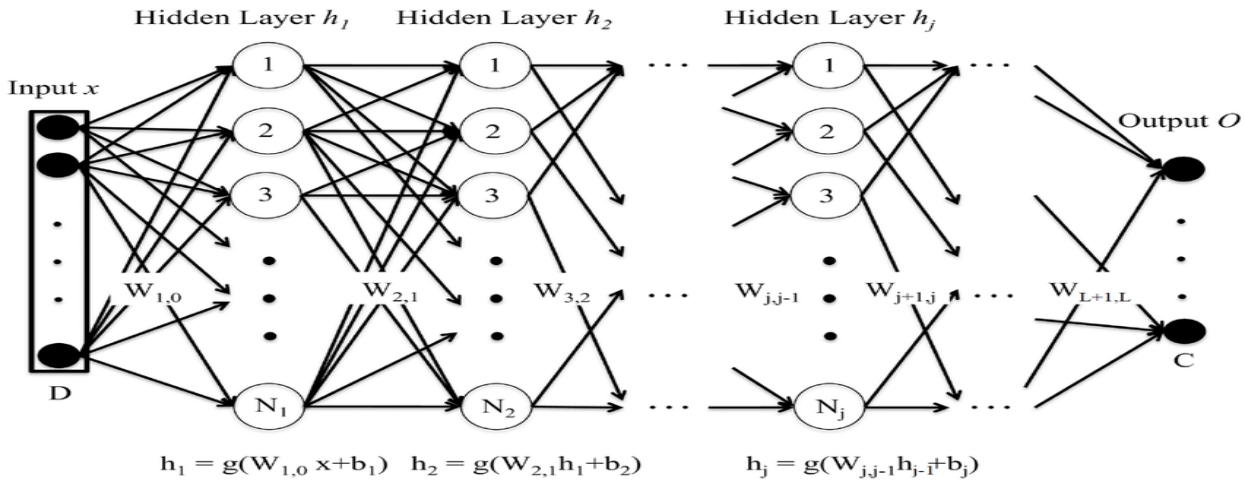
https://www.researchgate.net/figure/A-single-layer-feed-forward-neural-network_fig1_228394623

Για την περίπτωση τεσσάρων κόμβων όπως φαίνεται και στο σχήμα απο πάνω το δίκτυο καλείται ενός επιπέδου δίκτυο , με την σχεδίαση ενός layer επίπέδου εννοούμε το επίπεδο εξόδου των υπολογίσημων κόμβων (νευρώνες) . Δεν λαμβάνεται υπόψιν στον υπολογισμό το επίπεδο εισόδου με τους κόμβους εισόδου επειδή δεν γίνονται υπολογισμοί.

Δίκτυα εμπρόσθιας τροφοδότησης πολλαπλών επιπέδων (Multilayer Feedforward Networks)

Αυτή η αρχιτεκτονική διαφέρει απο την πρώτη στην παρουσία ενός ή περισσοτέρων κρυφών επιπέδων,των οποίων οι νευρώνες καλούνται κρυφοί νευρώνες [5]. Κρυφοί νευρώνες σημαίνει οτι δεν φαίνονται απευθείας είτε απο την είσοδο είτε απο την έξοδο του δικτύου. Η λειτουργία των κρυφών νευρώνων είναι οτι παρεμβάλλονται μεταξύ της εξόδου του κόμβου εισόδου και των νευρώνων εξόδου του δικτύου. Προσθέτοντας ενα η περισσότερα κρυμένα επίπεδα ,το δίκτυο έχει την δυνατότητα να προσεγγίσει συναρτήσεις μεγαλύτερης πολυπλοκότητας [5].

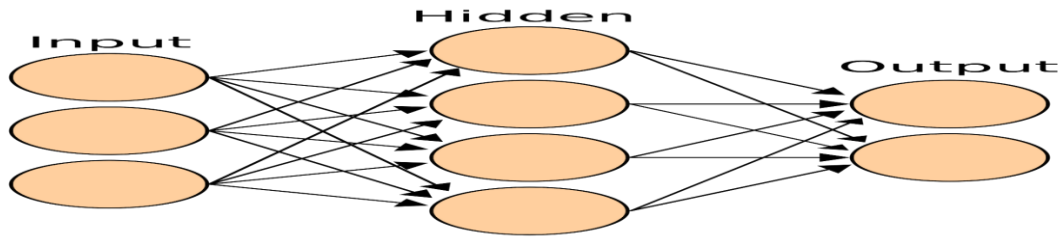
Οι κόμβοι εισόδου, στο επίπεδο εισόδου του δικτύου ,παρέχουν τα σεβαστά στοιχεία των προτύπων εισόδων , με την μορφή διανυσμάτων ,που γίνονται είσοδοι στους νευρώνες του δεύτερου επιπέδου δηλαδή του πρώτου κρυφού επιπέδου [5]. Τα σήματα εξόδου των νευρώνων του δεύτερου κρυφού επιπέδου , γίνονται σήματα εισόδου για το τρίτο επίπεδο και συνεχίζεται με αυτόν τον τρόπο η ροή των σημάτων μεταξύ των επιπέ-δων του δικτύου μέχρι να φτάσουν στο επίπεδο εξόδου [5]. Τυπικά , οι νευρώνες σε κάθε επίπεδο του δικτύου παίρνουν σαν είσοδο τα σήματα εξόδου απο τους νευρώνες του προηγούμενου επιπέδου μόνο. Το σύνολο των σημάτων εξόδου των νευρώνων στο επίπεδο εξόδου (τελευταίο επίπεδο του δικτύου) συνεισφέρει στην συνολική απάντηση του δικτύου για τα δεδομένα που εισάγονται στους κόμβους εισόδου [5].



Σχήμα 1.4.2

πηγή

https://www.researchgate.net/figure/Deep-Neural-Network-DNN-This-is-a-graphical-representation-of-a-standard-feedforward_fig4_328213334

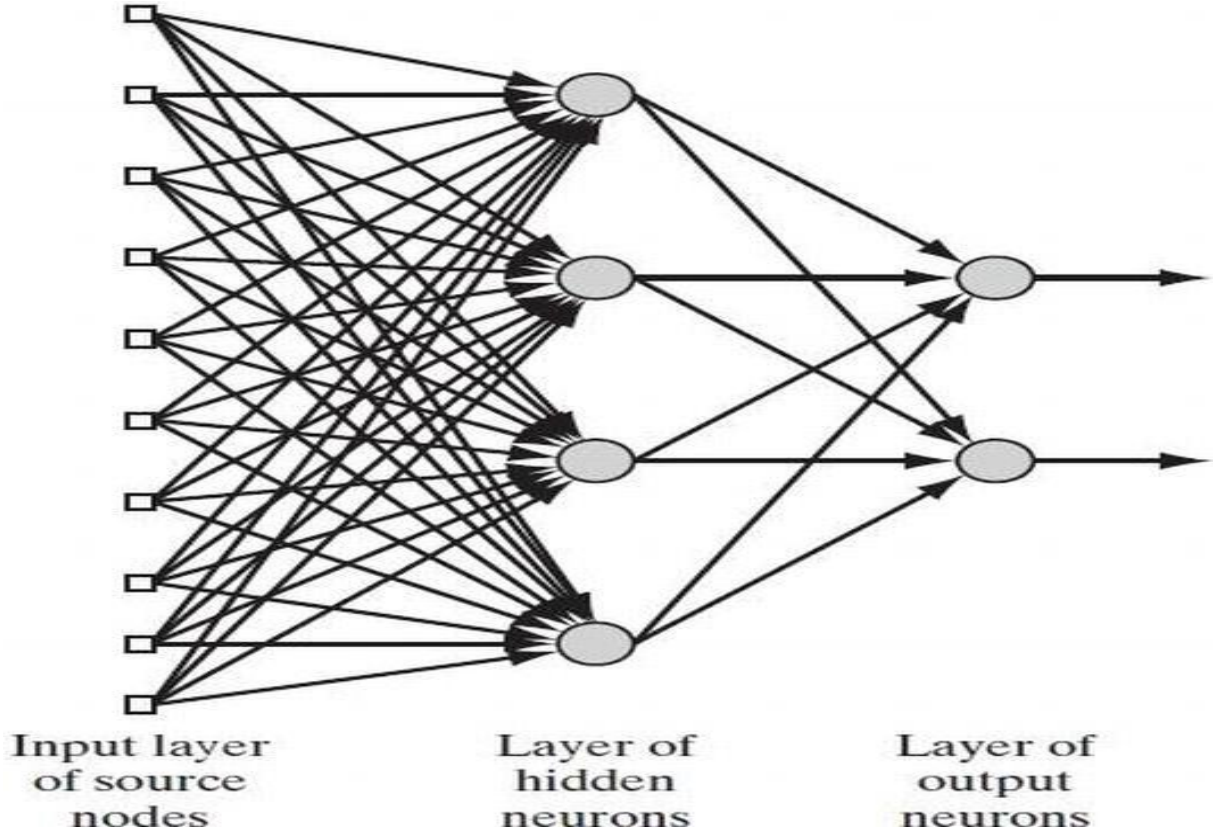


Σχήμα 1.4.3

πηγή

https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Print_Version

Στο παραπάνω σχήμα έχουμε ενα ΤΝΔ εμπρόσθιας τροφοδότησης με ένα κρυφό επίπεδο τεσσάρων κόμβων (h=4) ,τρεις κόμβους πηγής στο επίπεδο εισόδου (p=3) και δύο κόμβους (q=2) στο επίπεδο εξόδου.



Σχήμα 1.4.4

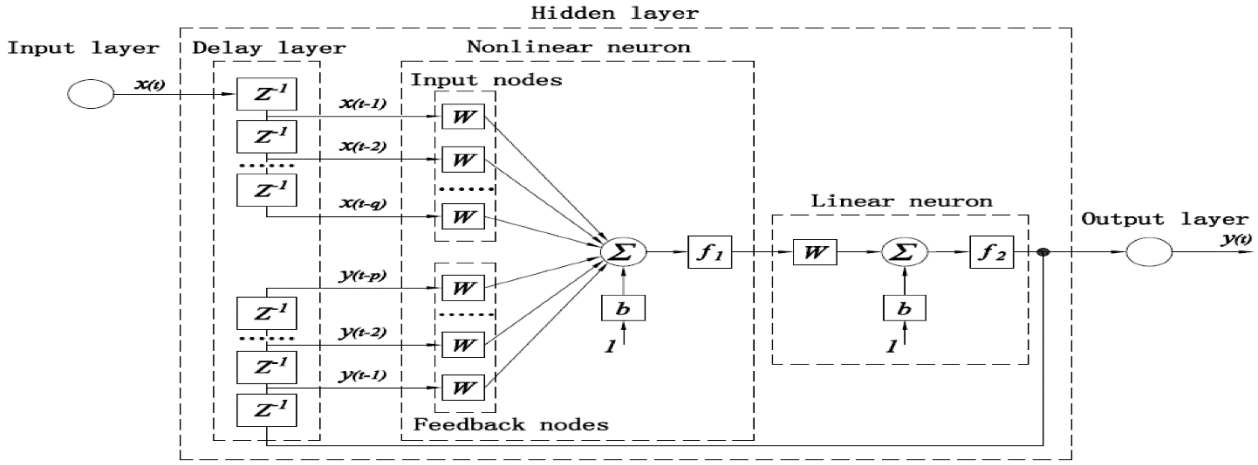
πηγή

https://www.intechopen.com/books/digital-systems/neural-network-principles-and-applications

Στο σχήμα αυτό έχουμε 10 κόμβους εισόδου , ενα κρυφό επίπεδο τεσσάρων κόμβων , 2 κόμβους στο επίπεδο εξόδου. Αρα είναι 10 – 4 – 2 δίκτυο. Ονομάζεται και πλήρως διασυνδεμένο νευρωνικό δίκτυο

Αναδρομικά Τεχνητά Νευρωνικά Δίκτυα (Recurrent Neural Networks)

Ενα Αναδρομικό Τεχνητό Νευρωνικό Δίκτυο διαφέρει απο το εμπρόσθιας τροφοδό-τησης στο γεγονός οτι έχει τουλάχιστον ένα βρόγχο ανατροφοδότησης. Αυτό σημαίνει ότι σε έναν τουλάχιστον νευρώνα , το σήμα εξόδου του επηρεάζει το σήμα που έρχεται στην είσοδο του νευρώνα [5]. Εστω λοιπόν ενα Αναδρομικό Τεχνητό Νευρωνικό Δίκτυο μπορεί να αποτελείται απο ένα μόνο επίπεδο νευρώνων όπου κάθε νευρώνας επιστρέφει το σήμα εξόδου του σαν σήμα εισόδου σε όλους τους άλλους νευρώνες του επιπέδου. Στα παρακάτω σχήματα βλέπουμε διάφορες μορφές δικτύων όπου σε άλλα υπάρχουν κρυφοί νευρώνες και σε άλλα δεν υπάρχουν κρυφοί νευρώνες.

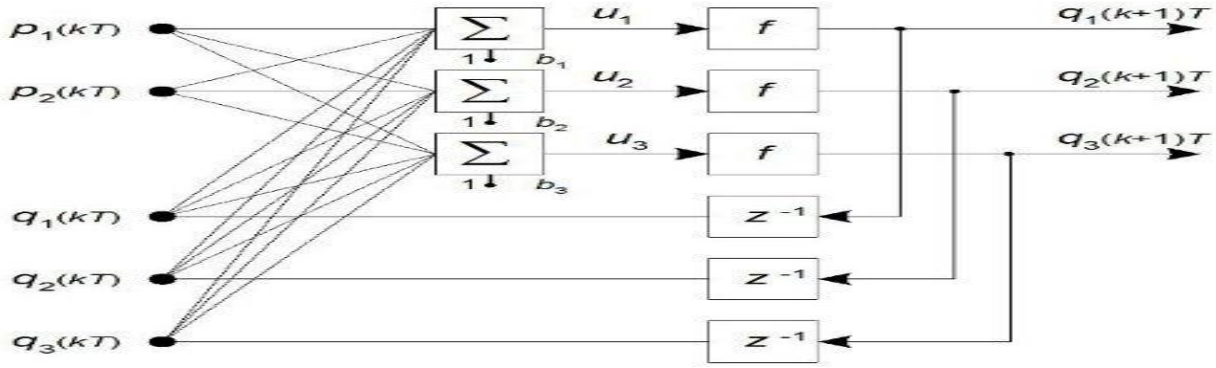


Σχήμα 1.4.5

πηγή

https://www.mdpi.com/2076-3417/8/10/1916

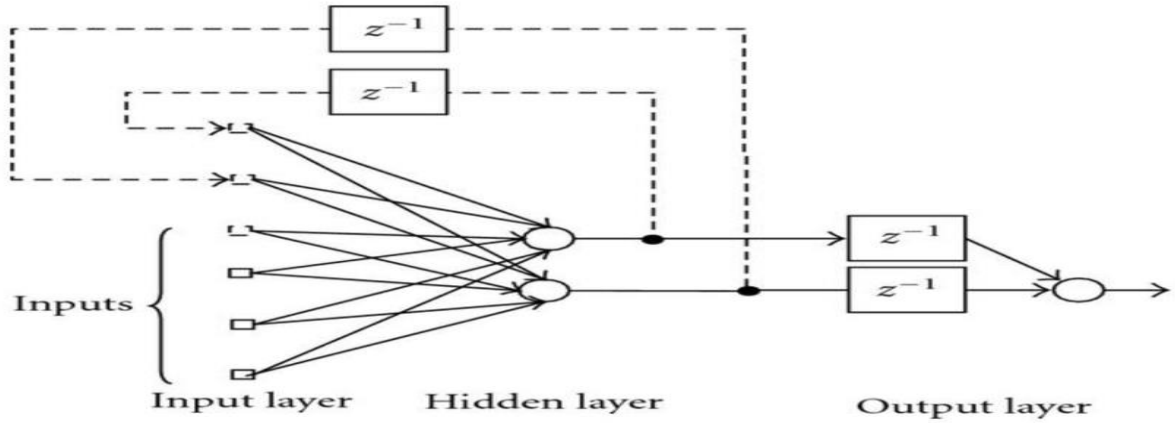
Σε αυτό το σχήμα έχουμε ενα Αναδρομικό ΤΝΔ με κρυφούς νευρώνες



Σχήμα 1.4.6

πηγή

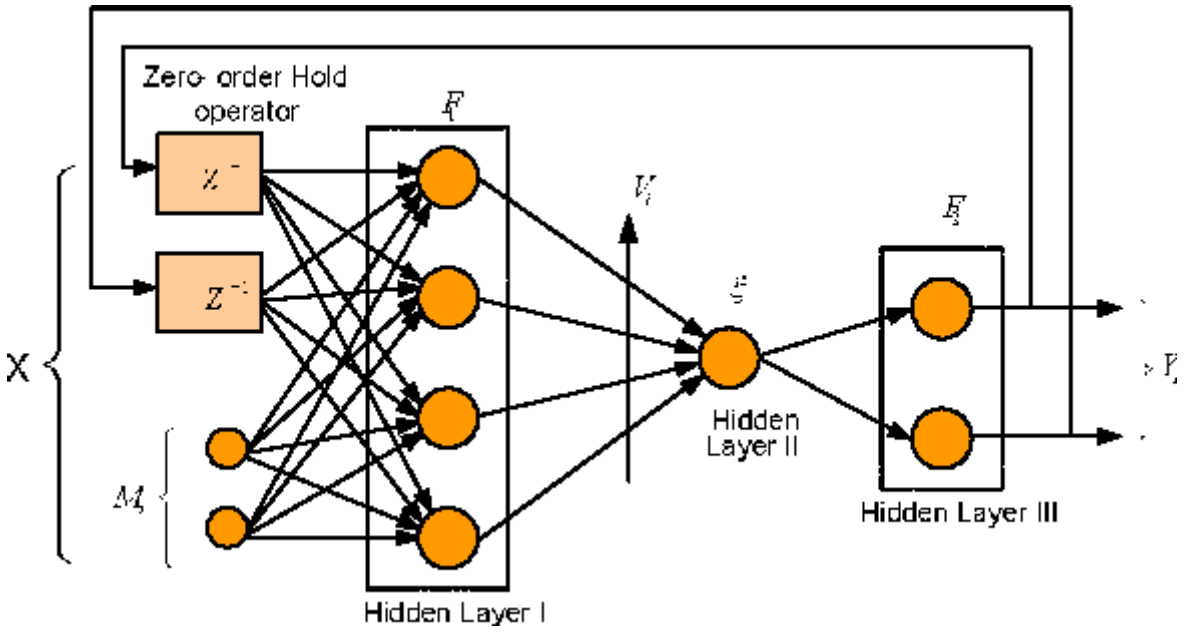
https://www.intechopen.com/books/parallel_manipulators_new_developments/application_of_neural_networks_to_modeling_and_control_of_parallel_manipulators



Σχήμα 1.4.7

πηγή

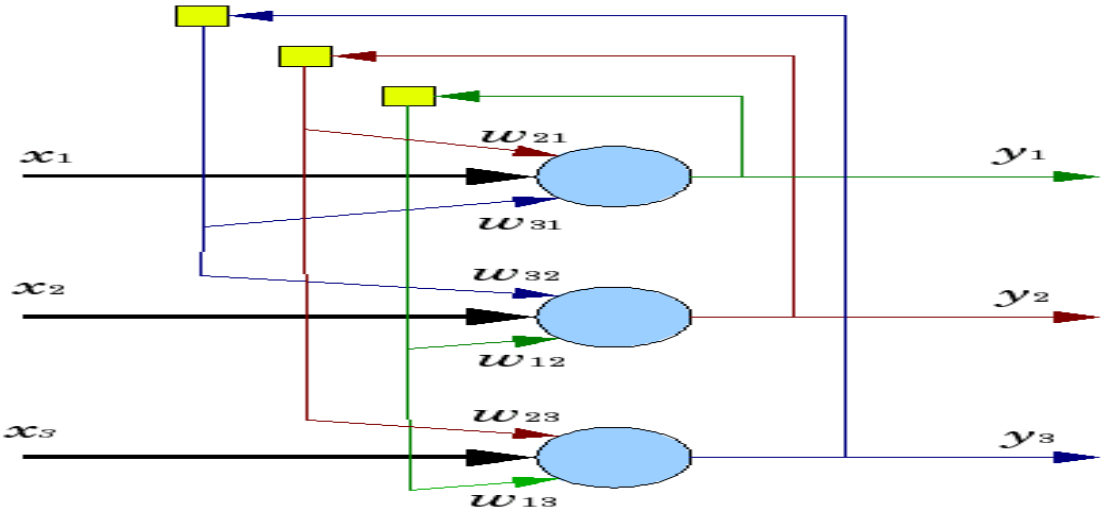
https://www.researchgate.net/figure/A-fully-connected-recurrent-neural-network-with-4-inputs-a-hidden-layer-with-2-neurons_fig5_220588437



Σχήμα 1.4.8

πηγή

https://www.semanticscholar.org/paper/Symmetric-Cipher-Design-Using-Recurrent-Neural-Arvandi-Wu/54dbb06662aa8d9398ecc1eb85eca2aa5ba737ca/figure/0

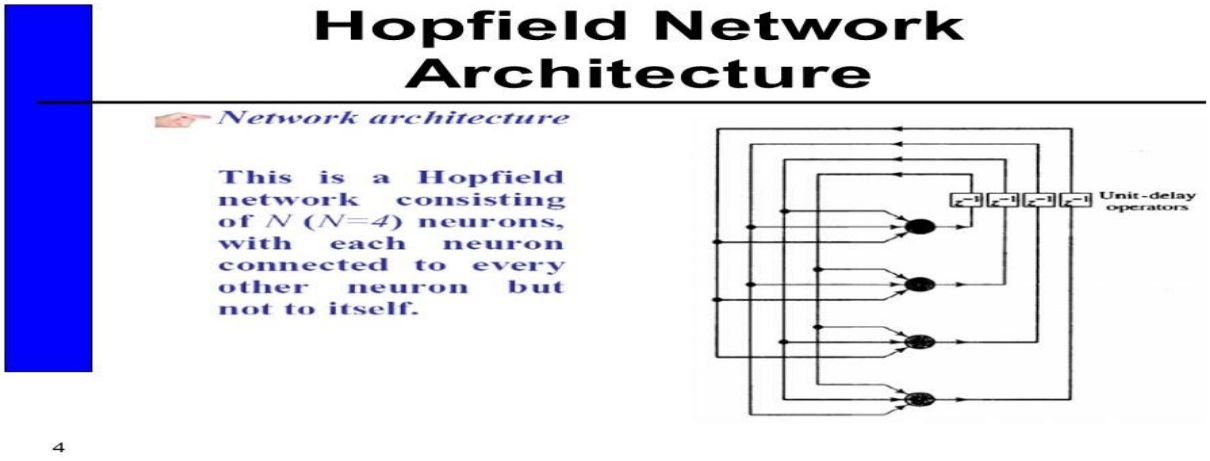


Σχήμα 1.4.9

πηγή

http://ikee.lib.auth.gr/record/131564/files/GRI-2013-10277.pdf

Στο παραπάνω σχήμα έχουμε ενα Αναδρομικό ΤΝΔ με κρυφούς νευρώνες . Τα κουτιά με το κίτρινο χρώμα είναι τα στοιχεία καθυστέρησης.

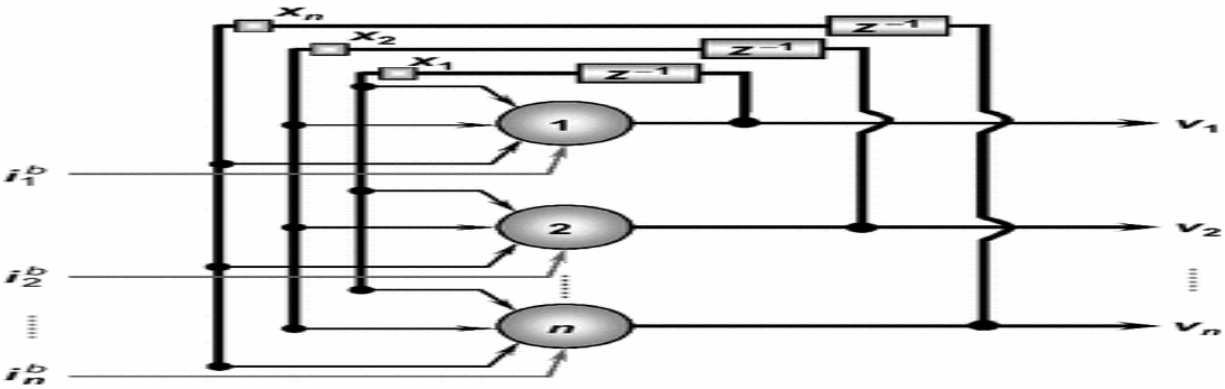


Σχήμα 1.4.10

πηγή

https://slideplayer.com/slide/11629444/

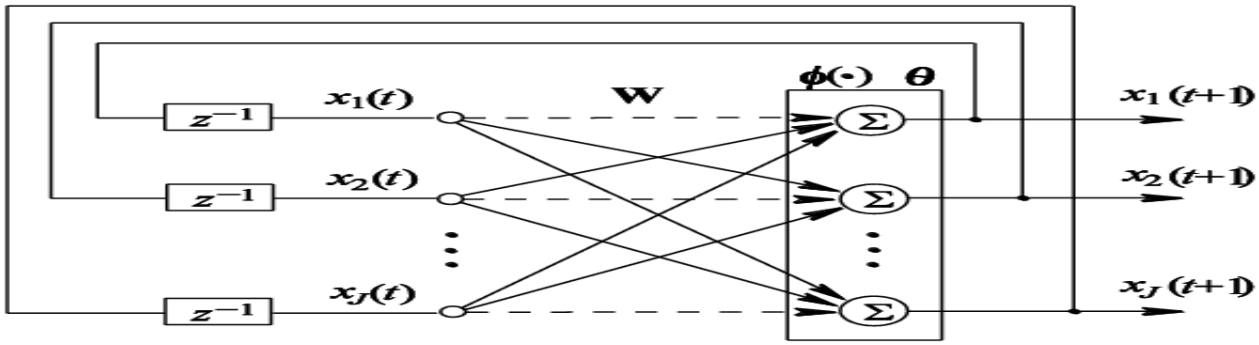
Στο παραπάνω σχήμα έχουμε ενα Hopfield Δίκτυο με 4 νευρώνες που ο κάθε νευρώνας συνδέεται με οποιαδήποτε άλλο εκτός απο τον εαυτόν του.



Σχήμα 1.4.11

πηγή

https://link.springer.com/chapter/10.1007/978-3-319-43162-8_7



Σχήμα 1.4.12

πηγή

https://www.researchgate.net/figure/Architecture-of-the-Hopfield-network-Note-that-i-i-i-ii-i-0-is-represented-by_fig1_254052169

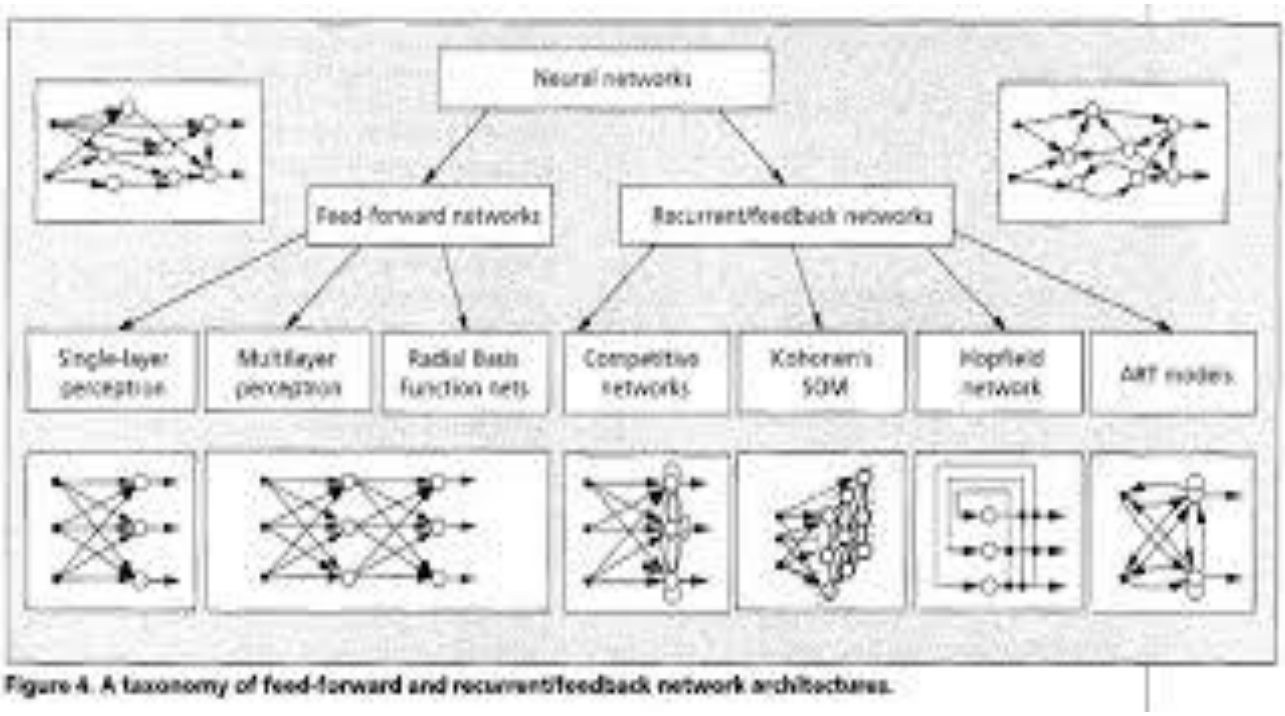


Figure 4. A taxonomy of feed-forward and recurrent/feedback network architectures.

Σχήμα 1.4.13

πηγή

http://recentscientific.com/sites/default/files/7984-A-2017_0.pdf

Τυπικά δίκτυα απο τις δύο κατηγορίες νευρωνικών δικτύων

1.5 Εκπαίδευση Τεχνητών Νευρωνικών Δικτύων

Οπως έχουμε ξαναπεί ο ανθρώπινος εγκέφαλος αποτελείται απο περίπου 100 δισεκατομμύρια νευρώνες και σε κάθε νευρώνα αντιστοιχούν κατά μέσο όρο περίπου 1000 συνάψεις , αρα έχουμε ένα σύνολο 100 τρισεκατομμύρια συνάψεις [1].

Με το να είναι τόσο πολύπλοκος ο ανθρώπινος εγκέφαλος του επιτρέπει να επιτελεί τόσες λειτουργίες και αυτό ονομάζεται νοημοσύνη. Οι λειτουργίες αυτές είναι [1]:

- Η αναγνώριση εικόνων
- Η μνήμη
- Η αναγνώριση φωνής
- Η αυτόνομη πλοήγηση στο χώρο
- Η λήψη αποφάσεων
- Η κατάστρωση στρατηγικής και η επιλογή της καλύτερης με βάση διάφορα κριτήρια κόστους
- Η λογική, η ανάπτυξη επιχειρημάτων , η συνεπαγωγή
- Η μάθηση και η αναπροσαρμογή σε καινούργιο περιβάλλον

Η μάθηση τώρα είναι απο τα πιο σημαντικά χαρακτηριστικά του εγκεφάλου και γενικά των βιολογικών νευρωνικών δικτύων. Οι υπόλοιπες λειτουργίες μαθαίνονται στα πρώιμα χρόνια ενώ η μάθηση είναι απο κει και έπειτα που περιλαμβάνει την εκμάθηση της μητρικής γλώσσας εκμάθησης ξένης γλώσσας, τα οικεία πρόσωπα και πολλά άλλα [1]. Τώρα στη Τεχνητή Νοημοσύνη δεν έχουμε βιολογικούς νευρώνες που έχει ο εγκέφαλος αλλά ενα υλικό που αποτελείται απο ημιαγωγά στοιχεία και τρανζίστορ. Και δέυτερον έχουμε τους αλγόριθμους που ουσιαστικά μιμούνται τις ανθρώπινες λειτουργίες όπως αναγνώριση φυσικής γλώσσας ,αναγνώριση φυσικού περιβάλλοντος, αυτόματη πλοήγηση ,θα έχουν μνήμη και πολλά άλλα.

Τα τεχνητά νευρωνικά δίκτυα είναι μοντέλα που μιμούνται την λειτουργία των βιολογικών νευρώνων.

Το αντικείμενο των τεχνητών νευρωνικών δικτύων είναι η ανάπτυξη και η μελέτη μαθηματικών αλγορίθμων που μιμούνται και την αρχιτεκτονική αλλά και το πρότυπο των βιολογικών νευρωνικών δικτύων.

Πάμε τώρα στην μάθηση των τεχνητών νευρωνικών δικτύων που αναφέρεται σαν την διεργασία επίτευξης μιας επιθυμητής συμπεριφοράς μέσω ανανέωση τις τιμής των συναπτικών βαρών.

Υπάρχουν πολλοί αλγόριθμοι μάθησης ΤΝΔ με πλεονεκτήματα και μειονεκτήματα. Κάθε ένας απο αυτούς προσφέρει διαφορετικό τρόπο προσαρμογής (επιλογής/ανανέωσης) των α συναπτικών βαρών [5].

Ενα άλλο θέμα που θα μας απασχολήσει είναι το περιβάλλον στο εργάζεται το ΤΝΔ [5]. Αρα διαφορετικά μοντέλα περιβάλλοντος οδηγεί σε διαφορετικούς αλγόριθμους μάθησης. Οι αλγοριθμοι μπορούν να χωριστούν σε τρεις μεγάλες κατηγορίες [5].

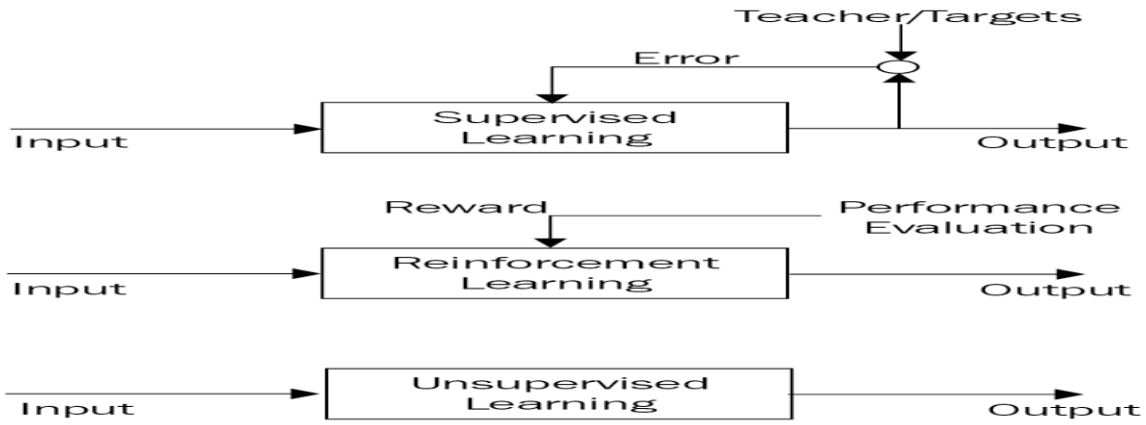
- Μάθηση με επίβλεψη
- Μάθηση χωρίς επίβλεψη
- Ενισχυτική Μάθηση

Υπάρχουν τέσσεριις τρόποι/κανόνες νευρωνικής μάθησης [5]

- Μάθηση διόρθωσης σφάλματος
- Μάθηση Hebb
- Κανόνες του Hopfield
- Κανόνες Δέλτα
- Κανόνες Εκμάθησης Kohonen

<u>Μαθηση με επίβλεψη (supervised learning)</u>

Σε αυτό το μοντέλο έχουμε δυο συνιστώσες , το δάσκαλο και το σύστημα μάθησης



Σχήμα 1.5.1

πηγή

https://www.oreilly.com/library/view/neural-networks-with/9781788397872/0bc8f2bc-bc57-4478-aa3c-4bf72c7bc912.xhtml

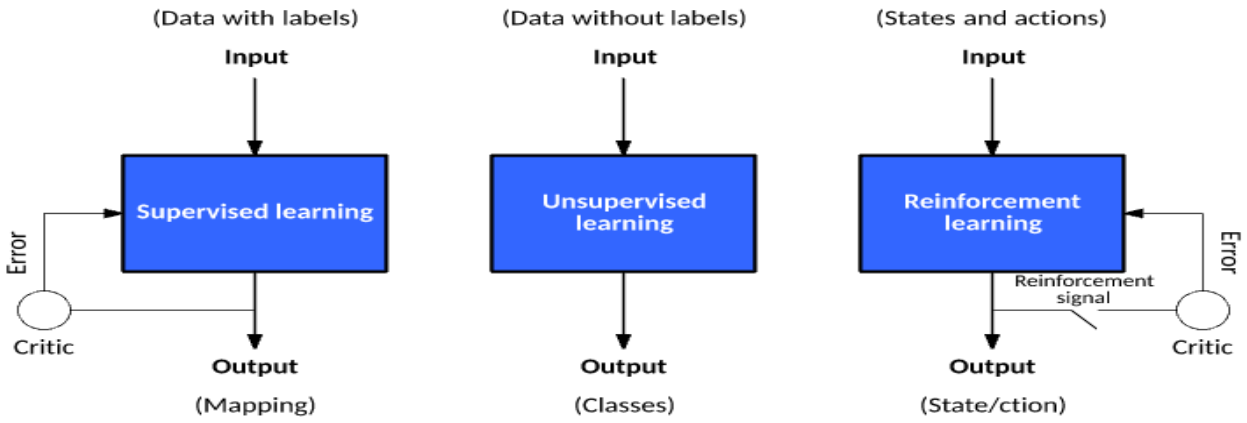
Το κύριο χαρακτηριστικό της μάθησης με επίβλεψη είναι η ύπαρξη του εξωτερικού δάσκαλου ο οποίος μπορεί να διδάξει στο σύστημα μάθησης (ΤΝΔ) τις επιθυμητές εξόδους (output) για ένα σύνολο εισόδων εκπαίδευσης [5]. Ουσιαστικά ο δάσκαλος για ενα διάνυσμα εισόδου εκπαίδευσης θα δώσει μια επιθυμητή έξοδο η οποία θα δείχνει τη καλύτερη ενέργεια που θα πρέπει να κάνει το ΝΔ . Αρα μπορούμε να πούμε οτι έχουμε ζεύγη πραγματικών διανυσμάτων εισόδου – εξόδου , και το δίκτυο υπολογίζει το διάνυσμα εξόδου βάση των τρέχουσων τιμών των συναπτικών βαρών. Μετά έχουμε το σήμα σφάλματος που υπολογίζεται μεταξύ της εξόδου του δικτύου και της επιθυμητής εξόδου. Αυτό το μοντέλο αποτελείται απο ένα σύστημα ανάδρασης κλειστού βρόγχου οπως φαίνεται στο παραπάνω σχήμα. Σύμφωνα με αυτό η εξοδος συγκρίνεται με την επιθυμητή έξοδο και ανάλογα με τη διαφορά τους τροποποιείται το σύστημα, δηλαδή τροποποιούνται τα βάρη των συνάψεων [5]. Αυτό θα γίνει για όλα τα ζεύγη εισόδων – εξόδων, οπότε σε κάθε επανάληψη το σύστημα θα διορθώνει τα λάθη του. Σκοπός μας είναι να μπορέσουμε να περιορίσουμε το σφάλμα και αυτό έχει να κάνει με την απόδοση του συστήματος. Ως μέτρο της απόδοσης του συστήματος ορίζεται το μέσο τετραγωνικό σφάλμα ή άθροισμα των τετραγώνων των σφαλμάτων πάνω στο διάνυσμα εξόδου (πραγματική και επιθυμητή έξοδος).

Αλγόριθμοι για μάθηση με επίβλεψη είναι δυο ,πρώτον ο αλγόριθμος ελαχίστου μέσου τετραγώνου (Least Mean Square) και δεύτερον είναι ο αλγόριθμος της οπισθεν διάδοσης σφάλματος (Back Propagation) [5].

Μάθηση χωρίς επίβλεψη (unsupervised or adaptive learning)

Σε αυτόν το τύπο μάθησης δεν έχουμε δάσκαλο για να επιβλέψει την εκπάιδευση του ΝΔ. Εχει εισόδους, αλλά όχι και τα επιθυμητά αποτελέσματα. Εδώ το σύστημα θα πρέπει να αποφασίσει ποια χαρακτηριστικά θα χρησιμοποιήσει για να ομαδοποιήσει τα δεδομένα εισόδου [6]. Δηλαδή κάνει αυτο – οργάνωση ή προσαρμογή. Η συγκεκριμένη οργάνωση

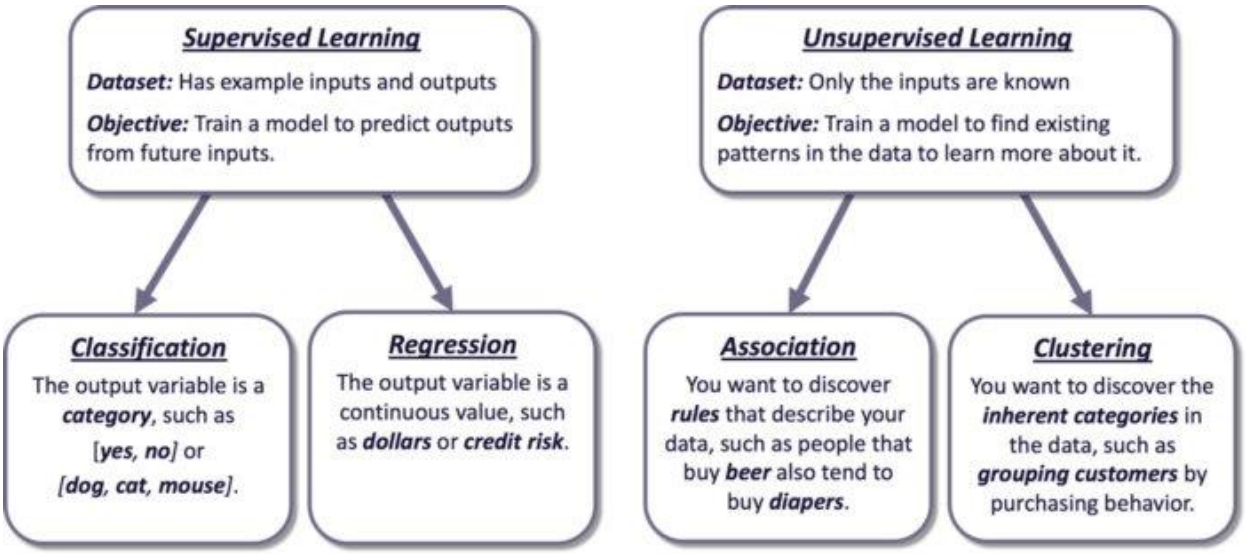
γίνεται με τέτοιον τρόπο ώστε για κάποιο συγκεκριμένο αριθμό εισόδων να ενεργοποιείται ένας συγκεκριμένος νευρώνας [6]. Και επειδή δεν έχουμε διανύσματα εξόδου που να αντιστοιχούν σε διανύσματα εισόδου , θα πρέπει το δίκτυο να χρησιμοποιήσει κάποιον εσωτερικό έλεγχο που να εντοπίζει κανονικότητα στα διανύσματα εισόδου έτσι ώστε τα διανύσματα εξόδου να έχουν τα ίδια χαρακτηριστικά με αυτά των εισόδων [6]. Οπότε με αυτόν το τρόπο το σύστημα ελέγχει τον εαυτόν του και διορθώνει τα σφάλματα στις εκτιμήσεις των διανυσμάτων εξόδου με ενα μηχανισμό ανάδρασης. Για να εκτελεστεί εκπαίδευση χωρίς επίβλεψη θα πρέπει να χρησιμοποιηθεί ένα ανταγωνιστικός κανόνας εκπαίδευσης [6].



Σχήμα 1.5.2

πηγή

https://developer.ibm.com/articles/cc-models-machine-learning/



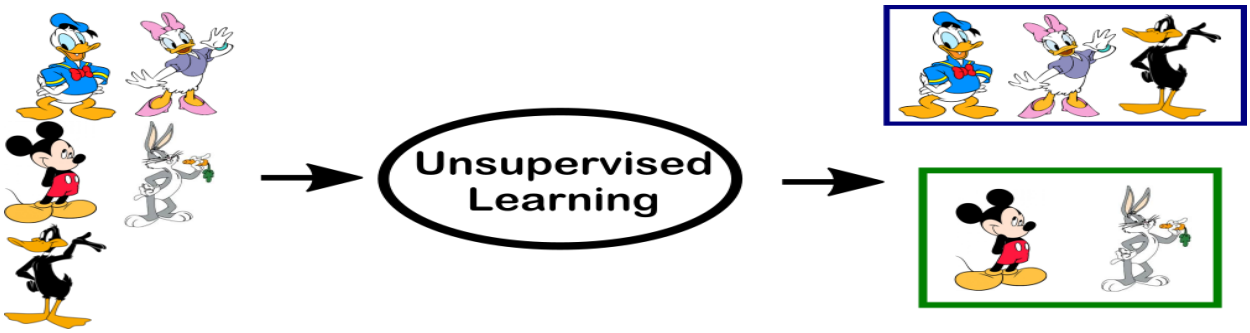
Σχήμα 1.5.3

πηγή

https://www.infoworld.com/article/3259512/machine-learning-what-developers-and-business-analysts-need-to-know.html

Η μάθηση χωρίς επίβλεψη μπορεί να συνδυασθεί με την μάθηση με επίβλεψη σε ένα πολυεπίπεδο δίκτυο εμπρόσθιας τροφοδότησης εκπαιδευόμενο με τον αλγόριθμο ανάστροφης διάδοσης για να επιταχύνει την διαδικασία μάθησης [5].

Ο αλγόριθμος της όπισθεν διάδοσης σφάλματος (Back Propagation) που είναι ένας αλγόριθμος μάθησης με επίβλεψης περιλαμβάνει μια φάση προς τα εμπρός όπου τα σήματα εισόδου διανύουν το ΤΝΔ επίπεδο με επίπεδο απο την είσοδο προς την έξοδο βγάζοντας μια απόκριση και μετά μια φάση προς τα πίσω κατά την οποία τα σήματα σφάλματος (διαφορά πραγματικών και επιθυμητών αποκρίσεων) διαδίδονται ανάστροφα (απο την έξοδο προς τη είσοδο) [5]. Στην φάση προς τα πίσω οι ελεύθεροι παράμετροι του ΤΝΔ προσαρμόζονται (επιλέγονται) έτσι ώστε να ελαχιστοποιηθεί το άθροισμα των τετραγώνων των σφαλμάτων [5].



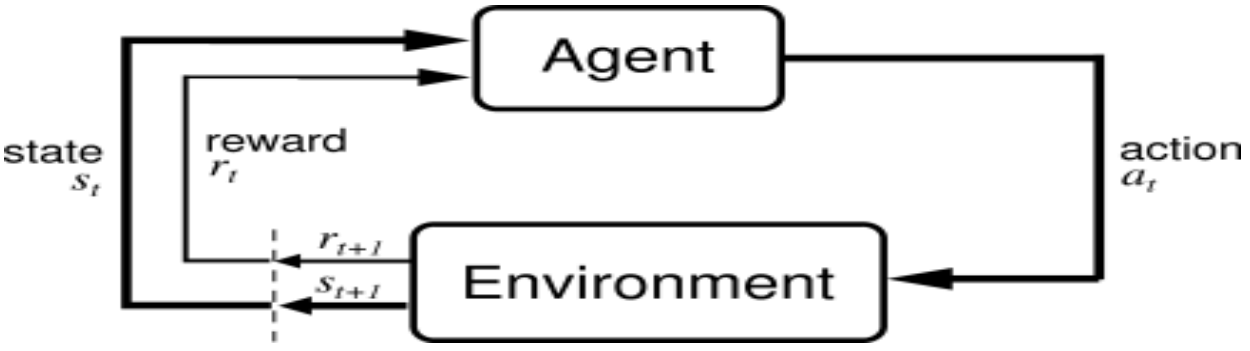
Σχήμα 1.5.4

πηγή

https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

Ενισχυτική Μάθηση (Reinforcement Learning)

Σε αυτόν τον τύπο μάθησης , το ΤΝΔ τροφοδοτείται με δειγματικά πρότυπα εισόδου αλλά δεν τροφοδοτείται με τις επιθυμητές αποκρίσεις σε αυτές τις εισόδους [5].



Σχήμα 1.5.5

πηγή

https://skymind.ai/wiki/deep-reinforcement-learning

Ενισχυτική μάθηση λοιπόν μπορούμε να πούμε ότι είναι το πρόβλημα που αντιμετω-πίζει ενας πράκτορας (agent) ο οποίος διαμορφώνει τη συμπεριφορά του εντός ενός δυναμικού περιβάλλοντος (environment) διαμέσου αλληλεπιδράσεων του τύπου προσπάθεια και λάθος (trial and error) .

Σύμφωνα λοιπόν με το παραπάνω σχήμα το μοντέλο αλληλεπίδρασης που μελετάται έχει ως εξής: σε κάθε διακριτό σημείο στο χρόνο ο πράκτορας μελετά την τρέχουσα κατάσταση (state) του περιβάλλοντος (environment) και επιλέγει την πραγματοποίηση μιας ενέργειας (action). Αυτό έχει σαν αποτέλεσμα ,το περιβάλλον να μεταβαίνει σε μια νέα κατάσταση και ο πράκτορας να λαμβάνει μια τιμή ανταμοιβής (reward) [7]. Αυτό το σήμα αποτελεί ένα μέτρο της ποιότητας των ενεργειών του πράκτορα, όπως καθορίζονται απο το περιβάλλον.

Οι τεχνικές μάθησης που βασίζονται σε ένα τέτοιο μοντέλο είναι πολύ ελκυστικές. Αυτό συμβαίνει διότι αν οι πράκτορες υπόκειται σε ενισχυτική μάθηση, οι σχεδιαστές θα πρέπει να παράγουν μόνο την τιμή της ανταμοιβής.

Η ενισχυτική μάθηση διακρίνεται σε δυο κατηγορίες σε συσχετιστική και σε μη συσχετι-στική μάθηση [5]. Στην μεν πρώτη , το περιβάλλον τροφοδοτεί ,πέρα απο το ενισχυτικό σήμα ,άλλες μορφές πληροφορίας απο τις οποίες το ΤΝΔ πρέπει να μάθει μια απεικόνι-ση με τη μορφή μιας συσχέτισης αιτίου – αποτελέσματος (stimulus –action) [5] .Στη δεύτερη , η μοναδική πληροφορία (είσοδος) που δίνεται απο το περιβάλλον είναι το ενισχυτικό σήμα, ενώ ο προορισμός του συστήματος μάθησης είναι να επιλέξει την μοναδική βέλτιστη ενέργεια αντί να συσχετίσει διάφορες ενέργειες με διαφορετικά ερεθίσματα (εισόδους) [5]. Μπορούμε να το περιγράψουμε διαφορετικά το πώς λειτουργεί η ενισχυτική μάθηση. Καταρχάς το ΤΝΔ υπολογίζει τις εξόδους οι οποίες θα παραχθούν απο την τρέχουσα κατάσταση είσοδο με τις παρούσες τιμές βαρών. Μετά το σύστημα θα αξιολογήσει την έξοδο και το ενισχυτικό σήμα θα τροφοδοτήσει το δίκτυο. Τα βάρη ανα-νεώνονται με βάση το ενισχυτικό σήμα αυξάνοντας τις τιμές των βαρών που συνέ-βαλαν σε καλή συμπεριφορά ή μειώνοντας τις τιμές αυτών που προκάλεσαν την κακή συμπεριφορά [5]. Σε όλες τις περιπτώσεις όταν το δίκτυο σταματάει να αλλάζει τις τιμές των βαρών, τότε λέμε οτι η εκπαίδευση έχει επιτεχθεί. Και αυτό διότι επειδή το σφάλμα στην έξοδο γίνεται μηδέν ή τείνει στο μηδέν [5]. Σε κάποιες περιπτώσεις μπορεί η εκπαίδευση να μην επιτεχθεί. Και αυτό μπορεί να συμβεί διοτι ίσως τα δεδομένα εισόδου να μην περιείχαν την σωστή πληροφορία απο την οποία θα προέκυπταν οι επιθυμητοί στόχοι [5].

Μπορούμε να πούμε ότι με τη μάθηση με επίβλεψη το δίκτυο πρέπει να εκπαιδευτεί πριν χρησιμοποιηθεί. Επίσης τα δεδομένα εκπαίδευσης που θα χρησιμοποιηθούν θα πρέπει να είναι αρκετά για να μας δώσουνε όλες τις απαράιτητες πληροφορίες ,έτσι ώστε το δίκτυο να μάθει απο τα χαρακτηριστικά γνωρίσματα και τις σχέσεις. Μόλις εκπαιδευτεί ένα δίκτυο με τα εκάστονται δεδομένα –στοιχεία εκπαίδευσης θα μπορούμε να δούμε τι μπορεί να κάνει με τα στοιχεία που δεν έχει δει πριν. Στην περίπτωση που δεν παίρνουμε λογικά αποτελέσματα για το εκάστονται σύνολο δοκιμής εννοείται οτι η εκπαίδευση δεν έχει τελειώσει. Γιαυτό είναι σημαντικό για ένα δίκτυο όχι μόνο να έχει απομνημονεύσει ένα δεδομένο σύνολο στοιχείων αλλά να έχει μάθει τα γενικά πρότυπα που περιλαμβάνονται σε μια εφαρμογή.

Τώρα πως γίνεται η χρήση των δεδομένων εκπαίδευσης , γίνεται λοπόν σε κύκλους εκπαίδευσης. Στη διάρκεια κάθε κύκλου το δίκτυο δέχεται σαν είσοδο, ενα ένα ,όλα τα διανύσματα εκπαίδευσης, αθροίζει την μεταβολή στην τιμή των βαρών που προκύπτει απο κάθε διάνυσμα και αναπροσαρμόζει τα βάρη στο τέλος κάθε κύκλου, χρησιμοποιώντας την συσσωρευμένη μεταβολή. Σε αυτήν την περίπτωση έχουμε εκπαίδευση ανά ομάδα προτύπων εισόδου (batch training) [5]. Μια αλλη περίπτωση είναι η προσαρμογή των βαρών να γίνεται μετά την χρήση ενός απο τα διανύσματα εκπαίδευσης οπότε λέμε για εκπαίδευση ανά πρότυπο εισόδου (online training) [5]. Και τέλος έχουμε και τους κύκλους εκπαίδευσης τους λεγόμενους εποχές (epoch), που έχουνε σχέση πιο πολύ με τη μάθηση δέσμης με γρηγορότερα αποτελέσματα και με μεγαλύτερες απαιτή-σεις μνήμης [5].

1.6 Τρόποι Νευρωνικής Μάθησης

Μάθηση διόρθωσης σφάλματος

Σε αυτόν τον τύπο μάθησης η απαιτούμενη ανανέωση (αλλαγή, προσαρμογή) των συναπτικών βαρών υπολογίζονται παρουσιάζοντας στο ΤΝΔ τα διανύσματα εισόδου και συγκρίνοντας τις πραγματικές/προκύπτουσες αποκρίσεις με τις επιθυμητές αποκρί-σεις και ακολούθως αλλάζοντας τα συναπτικά βάρη προς την κατεύθυνση μείωσης του σφάλματος [5].

Μάθηση Hebb

Η Μάθηση αυτού του τύπου βασίζεται στην εξής λογική: Οταν ένας άξονας ενός κυττάρου Α είναι τόσο κοντά ώστε να ερεθίσει ενα κύτταρο Β επίμονα τότε εχει μέρος ευθύνης στην πυροδότησή του, οπότε έχουμε μια διεργασία ανάπτυξης αλλαγών σε ένα η και τα δυο κύτταρα, με αποτέλεσμα η αποδοτικότητα του Α που πυροδοτεί το Β να αυξάνει [5].

Η υπόθεση αυτή μπορεί να επαναδιατυπωθεί σε δυο μέρη [5]:

(α) Το βάρος μιας σύναψης των νευρώνων της οποίας σε οποιαδήπότε πλευρά της, ενεργοποιούνται συγχρόνως, αυξάνεται επιλεκτικά.

(β) Το βάρος μιας σύναψης των νευρώνων της οποίας σε οποιαδήπότε πλευρά της, ενεργοποιούνται ασύγχρονα, μειώνεται επιλεκτικά

Μια τέτοια μάθηση ονομάζεται σύναψη τύπου Hebb

Κανόνας Hopfield

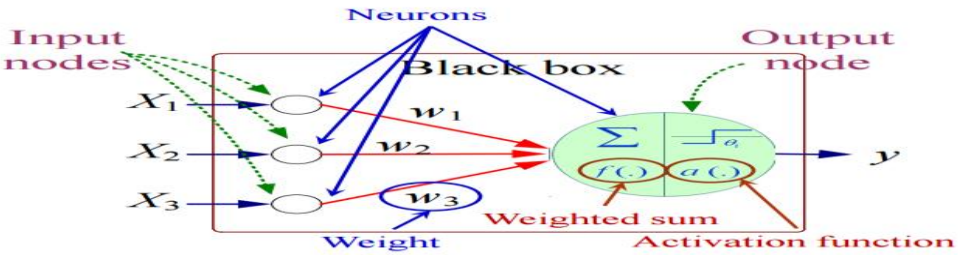
Μοιάζει με το κανόνα μάθησης του Hebb με μόνη διαφορά ότι διευκρινίζει το μέγεθος της ενίσχυσης ή της αποδυνάμωσης [5]. Βάση αυτού του κανόνα εάν η επιθυμητή έξοδος και είσοδος του νευρώνα είναι και οι δυο ενεργές ή και οι δυο ανενεργές τότε αυξάνεται το συναπτικό βάρος σύμφωνα με το ποσοστό εκμάθησης, διαφορετικά μειώνετια το βάρος σύμφωνα με το ποσοστό εκμάθησης [5].

Κανόνες Δέλτα

Ο κανόνας Δέλτα είναι μια παραλλαγή του κανόνα Hebb . Αυτός ο κανόνας βασίζεται στην ιδέα της συνεχούς τροποποίησης των βαρών του δικτύου έτσι ώστε να μειωθεί η διαφορά (Δέλτα) μεταξύ της επιθυμητής εξόδου και της πραγματικής [5]. Αυτός ο κανόνας αλλάζει τα συναπτικά βάρη με τέτοιον τρόπο ώστε να ελχαιστοποιεί το μέσο τετραγωνικό σφάλμα του δικτύου. Λέγεται και κανόνας εκμάθησης Widrow – Hoff και ο αλγόριθμος που τον υλοποιεί ονομάζεται και LMS [5].

Κανόνας εκμάθησης Kohonen

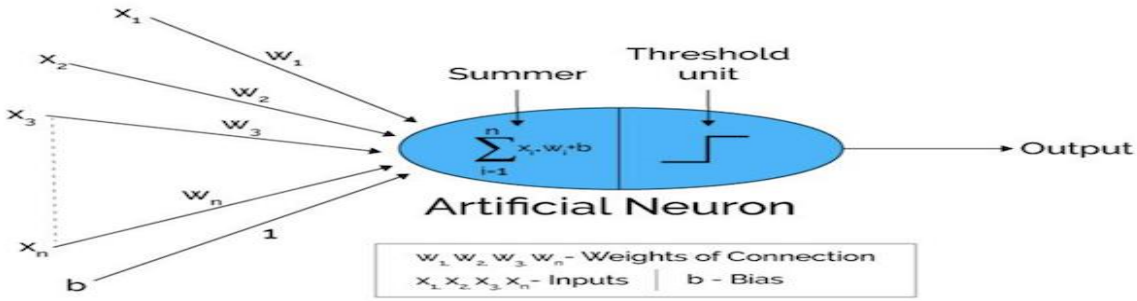
Αναπτύχθηκε απο τον Teuvo Kohonen εμπνευσμένος απο την μάθηση στα βιολογικά συστήματα [5]. Εδώ τα στοιχεία επεξεργασίας του δικτύου ανταγωνίζονται για να μάθουν ή να ενημερώσουνε τα βάρη τους [5]. Αυτό που θα έχει την μεγαλύτερη έξοδο είναι ο νικη-τής και έχει την ικανότητα να παρεμποδίζει τους ανταγωνιστές του όπως επίσης και να διεγείρει τους γειτονές του. Μόνο ο νικητής βγάζει έξοδο και μόνο αυτός και οι γείτονές του επιτρέπεται για να ρυθμίσουν τα βάρη σύνδεσής τους [5].



Σχήμα 1.6.1

Πηγή

https://github.com/chaitjo/Perceptron



Σχήμα 1.6.2

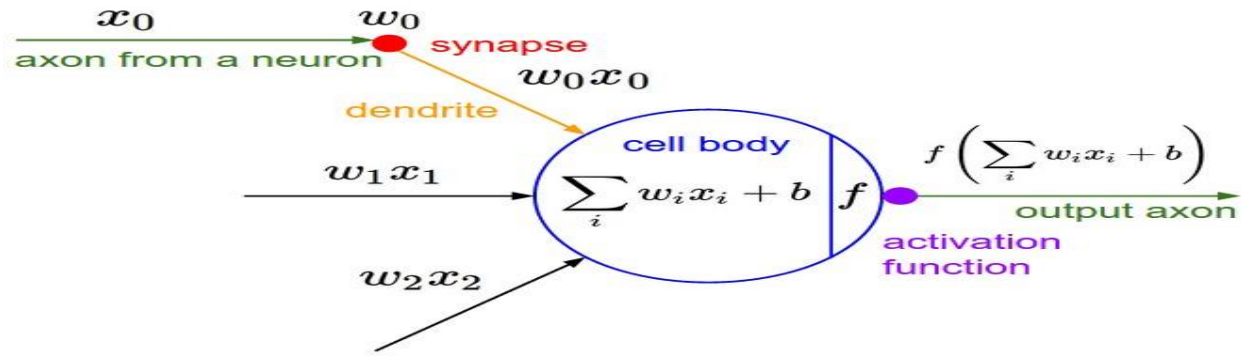
πηγή

https://becominghuman.ai/from-perceptron-to-deep-neural-nets-504b8ff616e

ΚΕΦΑΛΑΙΟ 2 ΠΕΡΙΓΡΑΦΗ ΑΠΛΟΥ ΜΟΝΤΕΛΟΥ ΚΑΙ ΕΚΠΑΙΔΕΥΣΗ

2.1 Το μοντέλο του αισθητήρα ή αλλιώς Perceptron του Rosenblatt

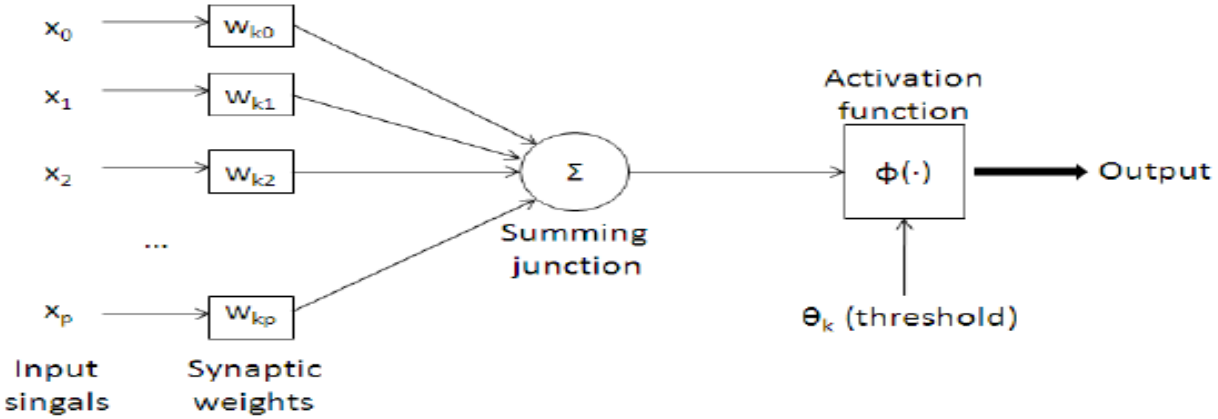
Το μοντέλο του αισθητήρα ή αλλιώς Perceptron του Rosenblatt είναι το πιο απλό Νευρωνικό Δίκτυο που μπορεί να σχεδιαστεί διότι αποτελείται μόνο απο ένα νευρώνα [3]. Είναι απο τα πρώτα μοντέλα Νευρωνικών Δικτύων που αναπτύχθηκαν και πάνω σε αυτό βασίζονται και τα πιο περίπλοκα δίκτυα που αναπτύχθηκαν μετέπειτα. Μπορεί να χαρακτηριστεί σαν ενα είδος ενός εμπρός τροφοδοτούμενου νευρωνικού δικτύου (feed – forward) . Αποτελείται απο ορισμένο αριθμό εισόδων ,αλλά παράγει μόνο μια έξοδο οπως φαίνεται και στο παρακάτω σχήμα.



Σχήμα 2.1.1

πηγή

http://euler.stat.yale.edu/~tba3/stat665/lectures/lec12/img/



Σχήμα 2.1.2

πηγή

https://www.researchgate.net/figure/Mathematical-model-of-an-artificial-neural-network-ANN_fig2_284516167

Εχουμε σύνδεση εισόδων x1, x2 ….xn και του νευρώνα . Κάθε σύνδεση του νευρώνα με το εισερχόμενο σήμα xi έχει και το αντίστοιχο συναπτικό βάρος wi που δείχνει την

επίδραση του εισερχόμενου σήματος στο νευρώνα. Μετά παίρνουμε το γινόμενο xi wi . Ο νευρώνας ακολουθεί το μοντέλο McCulloch – Pitts σύμφωνα με τις παρακάτω εξισώσεις [3]:

(Σχέση 2.1.1)

$$u = \sum_{i=1}^{n} WiXi - \theta$$

Η εξοδος του νευρώνα είναι η y=f(u) που μπορεί επίσης να γραφτεί και ώς εξής:

$$y = f(\sum_{i=1}^{n} WiXi - \theta)$$ (Σχέση 2.1.2)

Όπου u είναι η διέγερση του νευρώνα δηλαδή το συνολικό σήμα που αυτός λαμβάνει. Η συνάρτηση μεταφοράς (ενεργοποίησης) f απεικονίζει το διάνυσμα εισόδου x = [x1,x2,…xn]^T στην έξοδο y. Η παράμετρος θ ονομάζεται κατώφλι ενεργοποίησης και είναι το όριο πάνω απο το οποίο ενεργοποιείται ο νευρώνας [3]. Εχουμε τρεις περιπτώσεις:

$$u > 0 \ \alpha\nu \sum_{i=1}^{n} WiXi > \theta$$

$$u = 0 \ \alpha\nu \sum_{i=1}^{n} WiXi = \theta$$

$$u < 0 \ \alpha\nu \sum_{i=1}^{n} WiXi < \theta$$

Τώρα λοιπόν η συνάρτηση ενεργοποίησης f τροφοδοτείται απο τη διέγερση u και δίνει την έξοδο του νευρώνα [3]. Είναι μη γραμμική και για το Perceptron παίρνει μια απο τις παρακάτω μορφές της συνάρτησης κατωφλίου:

$$f(u)=\begin{cases} 1, \alpha\nu \ u > 0 \\ 0, \alpha\nu \ u \leq 0 \end{cases}$$

$$f(u)=\begin{cases} 1, \alpha\nu \ u > 0 \\ -1, \alpha\nu \ u \leq 0 \end{cases}$$

Απο τον πρώτο τύπο η έξοδος y = f(u) είναι ένας δυαδικός αριθμός που παίρνει την

τιμή y =1

$$\alpha v \sum_{i=1}^{n} WiXi > \theta$$

ο αισθητήρας ενεργοποιείται και πυροδοτεί ή την y = 0

$$\alpha v \sum_{i=1}^{n} WiXi < \theta$$

Τότε το u μηδενίζεται και ο αισθητήρας παραμένει αδρανής .Το πόσο ενεργός θα είναι ο αισητήρας εξαρτάται απο τρεις παραμέτρους: τα βάρη των συνδέσεων, τις τιμές των εισόδων και την τιμή του κατωφλίου [3]. Πολλές φορές θεωρούμε το κατώφλι θ σαν ένα επι-πλέον συναπτικό βάρος w0 = 0, το οποίο αντιστοιχεί στην είσοδο x0 = -1 και ξεχωρίζει απο τα άλλα συναπτικά βάρη αλλά δρά με τον ίδιο τρόπο. Οπότε το u γίνεται:

(Σχέση 2.1.3)

$$u = \sum_{i=1}^{n} WiXi - \theta = \sum_{i=1}^{n} WiXi + W0X0 = \sum_{i=1}^{n} WiXi$$

Τελικά προκύπτει οτι είναι το εσωτερικό γινόμενο των διανυσμάτων w και x (u=w^T x).

Θα προσθέσουμε και την επιπλέον είσοδο x0 = -1 οπότε γίνεται το διάνυσμα x = [x0,x1,…,xn]^T και το διάνυσμα των βαρών w = [w0,w1,…,wn]^T . Το βάρος w0 = -0 και αντιστοιχεί στην είσοδο x0 = +1 και ονομάζετια πόλωση (bias). Η πόλωση είναι εξωτερικό ερέθισμα το οποίο προστίθεται στο υπόλοιπο άθροισμα για να δώσει το σωστό δυναμικό ενεργοποίησης του νευρώνα [3]. Η εξοδος του δικτύου θα είναι ίση με 1 ή 0 ανάλογα με την είσοδο. Το δίκτυο μπορεί να χρησιμοποιηθεί για κατηγοριοποίηση των εισόδων σε δυο κατηγορίες. Ο διαχωρισμός μεταξύ των δυο κλάσεων σε αυτήν την περίπτωση είναι μια ευθεία γραμμή ή οποία δίνεται απο τις τιμές της εισόδου για τις οποίες η **εξοδος** του δικτύου είναι ίση με 0. Δηλαδή.

Y = W1X1 + W2X2 + bias (Σχέση 2.1.4)

Το δίκτυο αυτό χρησιμοποιείται για την λύση γραμμικών διαχωρίσιμων προβλημάτων.

Για να γίνει η ταξινόμηση σε μια απο τις δυο κλάσεις A1 ή A2 για το σύνολο των εισόδων ο κανόνας απόφασης για την ταξινόμηση είναι να αναθέτει το σημείο που αναπαριστούν οι εισόδοι x1,x2,…,xn στην κλάση A1 αν η έξοδος y = +1 ή στην κλάση A2 αν η έξοδος y = -1. Πιο αναλυτικά σχεδιάζουμε τις περιοχές απόφασης στον ηδιάστατο χώρο σημάτων που καταλαμβάνουν οι n μεταβλητές εισόδου x1,x2,…xn. Στην απλή μορφή έχουμε δυο περιοχές απόφασης που διαχωρίζονται απο ένα υπερεπίπεδο, το οποίο ορίζεται ως εξής [3]:

(Σχέση 2.1.5)

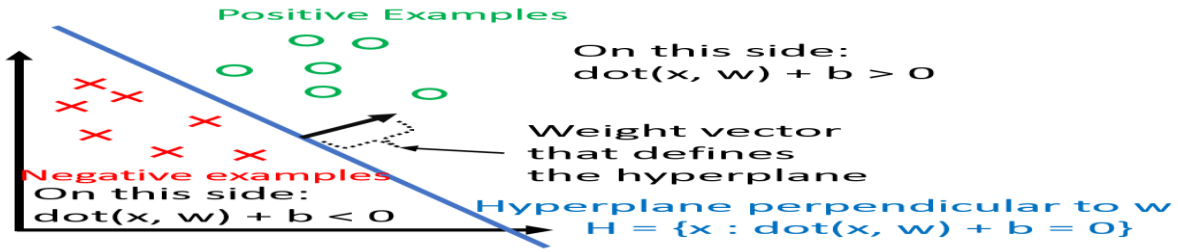
$$u = \sum_{i=1}^{n} WiXi - \theta = 0$$

(Σχέση 2.1.6)

$$\sum_{i=1}^{n} WiXi + b = 0$$

Εαν έχουμε δυο μεταβλητές εισόδων x1 και x2 το υπερεπίπεδο είναι μια ευθεία γραμμή που χρησιμοποιείται ώς όριο απόσφασης μεταξύ των κλάσεων και δίνεται απο την εξίσωση:

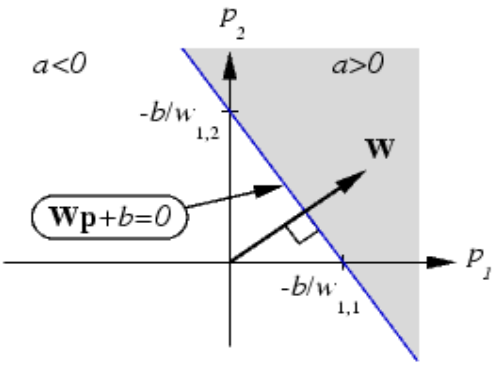
W1X1 + W2X2 + bias = 0

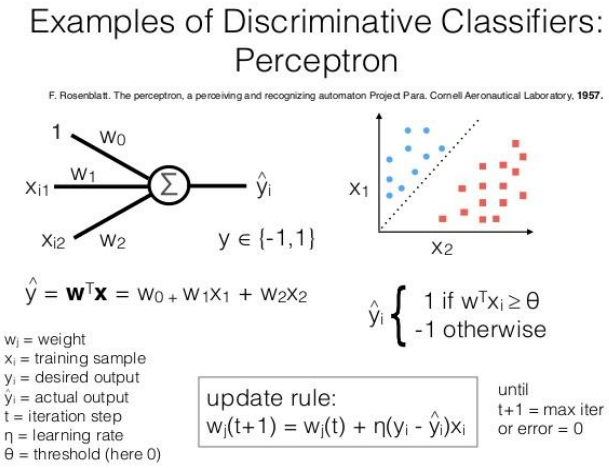


Σχήμα 2.1.3

πηγή

http://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote03.html





Σχήμα 2.1.4 Σχήμα 2.1.5

Πηγή

https://imagenesmi.com/im%C3%A1genes/perceptron-examples-ac.html

Τα σημεία (x1,x2) που αντιστοιχχούν σε τιμή u>0 βρίσκονται στην περιοχή πάνω απο την ευθεία γραμμή και ανήκουν στην κλάση A1 ή A γιατί η έξοδος του νευρώνα είναι y = +1. Τα σημεία (x1,x2) που αντιστοιχούν σε τιμή u<0 βρίσκονται στην περιοχή κάτω απο

την ευθεία γραμμή και ανήκουν στην κλάση Α2 ή Β γιατί η έξοδος του νευρώνα είναι y = -1 [3]. Τα σημεία (x1,x2) που αντιστοιχούν σε τιμή u = 0 βρίσκονται ακριβώς πάνω στην ευθεία. Τώρα όσον αφορά για την πόλωση bias ανάλογα με την τιμή που έχει μετατοπίζει το όριο απόφασης μακριά απο την αρχή των αξόνων.

2.2 Εκπαίδευση και μειονεκτήματα του δικτύου Perceptron

Ο σκοπός μας σε ένα νευρωνικό δίκτυο όπως το Perceptron είναι η αυτόματη εκμάθη-ση των παραμέτρων του συστήματος για να επιτεχθεί ο επιθυμητός στόχος , δηλαδή η έυρεση της διαχωριστικής γραμμής μεταξύ των δυο κλάσεων Α1 ή Α και Α2 ή Β. Για την κλάση Α1 ή Α θα πρέπει να ικανοποιείται η εξής ανισότητα: w^T x >0 για κάθε πρότυπο εισόδου και για την Α2 ή Βθα πρέπει να ικανοποιείται η εξής ανισότητα w^T x <0 για κάθε πρότυπο εισόδου. Το δίκτυο εκπαιδεύεται με επίβλεψη ,δηλαδή υπάρχει ένας δάσκαλος ο οποίος θα δίνει την επιθυμητή έξοδο d για κάθε πρότυπο εκπάιδευσης εισόδου x χρησιμοποιώντας κάποιον επαναληπτικό αλγόριθμο [3]. Εδώ έχουμε τον κλασικό κανόνα εκπαί-δευσης Perceptron γνωστός και σαν κανόνας σταθερής αύξησης (fixed increment rule) [3] . Είναι επαναληπτικός δηλαδή τα πρότυπα παρουσιάζονται στο δίκτυο με κυκλική σειρά και όταν τελειώσουν επαναλαμβάνονται απο την αρχή. Ενας πλήρης κύκλος χρήσης όλων των προτύπων καλείται εποχή (epoch) . Ο κανόνας αυτός μεταβάλει το διάνυσμα των συναπτικών βαρών w μόνο όταν υπάρχει σφάλμα ταξινόμησης, δηλαδή όταν ο στόχος d για το συγκεκριμένο πρότυπο εισόδου κάθε φορά διαφέρει απο την έξοδο του δικτύου [3]. Ο αλγόριθμος σύγκλισης για ένα Perceptron n εισόδων με ενα υπολογιστικό νευρώνα φαίνεται παρακάτω .Το διάνυσμα εισόδου είναι το x(n) = [+1, x1(n), x2(n), … .,xn(n)]^T και το διάνυσμα των βαρών είναι w(n) = [b, w1(n),w2(n),….,wn(n)]^T.

Εχουμε τα εξής βήματα :

1) Αρχικοπόίηση , μηδενισμός του διανύσματος των βαρών, δηλαδή w(0) = 0 και n =1 [3]
2) Ενεργοποίηση στο χρονικό βήμα n , ενεργοποιείται το Perceptron εφαρμόζοντας το διάνυσμα ειδόδων x(n) [3].
3) Υπολογισμός πραγματικής απόκρισης δηλαδή υπολογισμός της πραγματικής εξόδου του Perceptron με τους ακόλουθους τύπους [3]:

$$u(n) = w^T * x(n) \quad \text{(Σχέση 2.2.1)}$$

$$y(n) = \mathrm{sgn}(u(n)) = \mathrm{sgn}[w^T(n)*x(n)] \quad \text{(Σχέση 2.2.2)}$$

όπου η συνάρτηση προσήμου $sgn(u)=\begin{cases} 1, \alpha\nu\ u > 0 \\ -1, \alpha\nu\ u \leq 0 \end{cases}$

4) Προσαρμογή διανύσματος βαρών δηλαδή μεταβολή (προσαρμογή) των βαρών του Perceptron σύμφωνα με τον παρακάτω κανόνα [3]

$$W(n+1) = w(n) + n * [d(n) – y(n)] * x(n) \quad (\Sigma\chi\acute{\epsilon}\sigma\eta\ 2.2.3)$$

Όπου n είναι η παράμετρος μάθησης (βήμα ή ρυθμός εκπαίδευσης) που ρυθμίζει το μέγεθος της διόρθωσης και παίρνει τιμές στο διάστημα [0 1] και έχουμε επίσης και το d

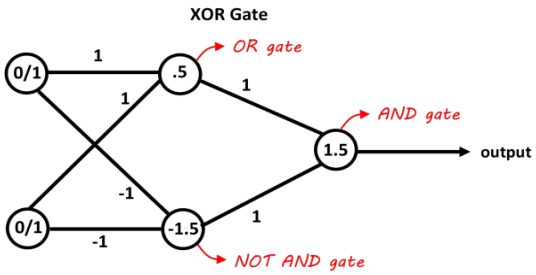
$$d(n)=\begin{cases} 1, \alpha\nu\ X(n)\ \alpha\nu\acute{\eta}\kappa\epsilon\iota\ \sigma\tau\eta\nu\ \kappa\lambda\acute{\alpha}\sigma\eta\ A1 \\ -1, \alpha\nu\ X(n)\ \alpha\nu\acute{\eta}\kappa\epsilon\iota\ \sigma\tau\eta\nu\ \kappa\lambda\acute{\alpha}\sigma\eta\ A2 \end{cases}$$

5) Συνέχιση δηλαδή αύξηση του χρονικού βήματος n κατά ένα και επιστροφή στο βήμα 2 [3]

Η εκπαίδευση του w γίνεται έτσι ώστε το πρότυπο που ταξινομήθηκε τώρα λάθος , την επόμενη φορά είτε θα ταξινομηθεί σωστά είτε θα πλησιάζει πιο πολύ ώστε να ταξινομηθεί σωστά [3].

Το μειονέκτημα του μοντέλου Perceptron είναι ότι δεν μπορεί να λύσει προβλήματα τα οποία είναι μη γραμμικά διαχωρίσιμα . Ενα απο αυτά είναι η συνάρτηση XOR , η οποία δεν μπορεί να παρασταθεί με ένα δίκτυο ενός μόνο νευρώνα [3]. Παρακάτω έχουμε τον πίνακα αληθείας της XOR .

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B



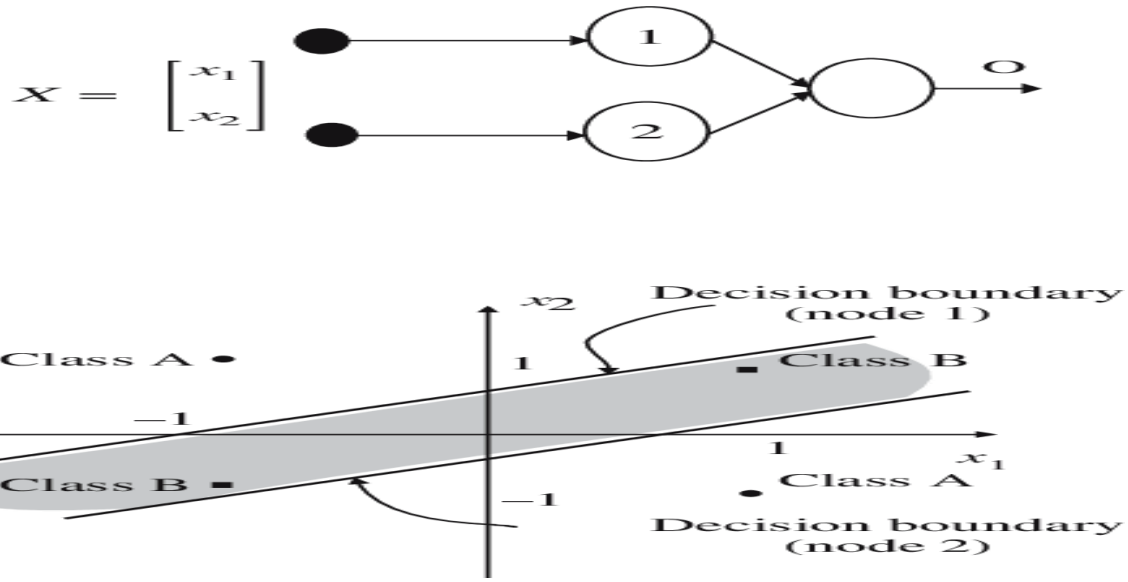
Σχήμα 2.2.1 *Σχήμα 2.2.2*

Πηγή

http://cs.joensuu.fi/pages/oili/PR/?a=Some__Material&b=Linear__And__Nonlinear__Classifiers

https://becominghuman.ai/neural-network-xor-application-and-fundamentals-6b1d539941ed

Εχουμε λοιπόν δυο εισόδους και μια έξοδο



Σχήμα 2.2.3

πηγή

https://www.researchgate.net/figure/a-Two-layers-perceptron-able-to-solve-the-Xor-problem-implementing-a-boundary-as_fig6_287165592

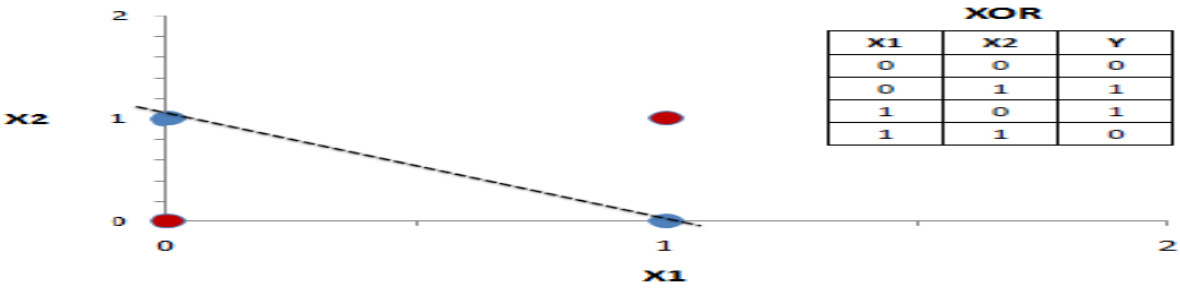
Η έξοδος του αισθητήρα δίνεται απο την σχέση

$$Y = \begin{cases} 0, & \alpha\nu\ w1x1 + w2x2 \le \theta \\ 1, & \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

Όπου w1 και w2 τα συναπτικά βάρη και θ το κατώφλι . Εμείς θέλουμε να βρεθούν οι τιμές για τα βάρη , θα πρέπει να είναι τέτοιες ώστε το αποτέλεσμα των εισόδων να συνάγει με την λογική πράξη XOR .

Τα τέσσερα σετ εισόδων του αισθητήρα αντιστοιχούν σε τέσσερα σημεία στο επίπεδο (x1,x2) [2]. Σύμφωνα με τον πίνακα αληθείας τα σημεία (0,0) και (1,1) ανήκουν στην κλάση 0 ,y = 0, ενώ τα (0,1) και (1,0) στην κλάση 1 ,y = 1. Η εξίσωση θ = w1x1 + w2x2 είναι μια ευθεία που χωρίζει το επίπεδο (x1,x2) σε δυο ημιεπίπεδα . Πρέπει να βρεθεί ένα ζεύγος τιμών για τα w1 και w2 ,τέτοια ώστε να χωρίζουν τις δυο αυτές κλάσεις [2] . Στο παρακάτω σχήμα

βλέπουμε ότι δεν είναι εφικτό . Δεν υπάρχει καμία ευθεία τέτοια ώστε τα σημεία των δυο κλάσεων να ανήκουν σε διαφορετικά ημιεπίπεδα. Για να το πετύχουμε αυτό χρειαζόμαστε δύο ευθείες. Αρα τελικά δεν είναι γραμμικά διαχωρίσιμο και δεν είναι επιλύσιμο απο τον αισθητήρα [2].



XOR

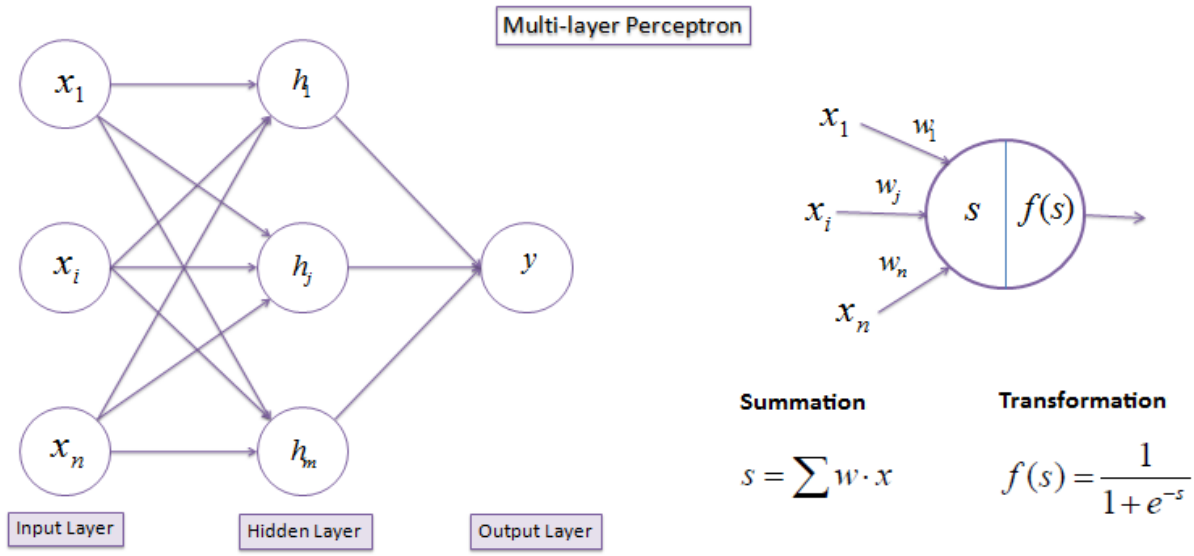
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Σχήμα 2.2.4

πηγή

http://www.saedsayad.com/artificial_neural_network_bkp.htm

Τίθεται λοιπόν το ερώτημα για το αν θα μπορούσε το αρχικό μοντέλο του αισθητήρα να τροποποιηθεί έτσι ώστε να επιλύει μη γραμμικά διαχωρίσιμα προβλήματα . Αυτό γίνετια χρησιμοποιώντας μοντέλο Perceptron πολλών επιπέδων μιας που είναι πιο περίπλοκο καθώς διαθέτει περισσότερους νευρώνες και μπορεί πλέον να επιλύσει και μη γραμμικά διαχωρίσιμα προβλήματα.



Σχήμα 2.2.5

Πηγή

http://www.saedsayad.com/artificial_neural_network_bkp.htm

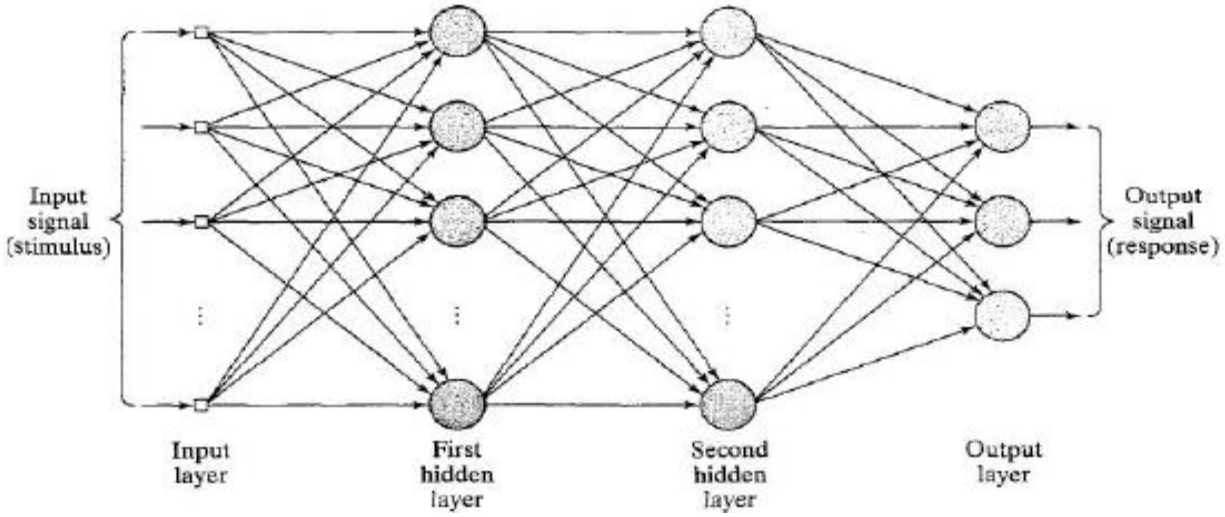
ΚΕΦΑΛΑΙΟ 3 ΜΕΘΟΔΟΙ ΚΑΙ ΤΡΟΠΟΙ ΕΚΠΑΙΔΕΥΣΗΣ

3.1 Το Perceptron πολλών επιπέδων

Η πιο απλή περίπτωση ΤΝΔ και πιο χρησιμοποιούμενη είναι τα δίκτυα πρόσθιας τροφοδότησης (feedforward networks) [3]. Σε αυτά τα δίκτυα δεν έχουμε ανατροφοδότηση της εξόδου ενός νευρώνα προς τους νευρώνες απο τους οποίους επηρεάζεται άμεσα ή έμμεσα [3]. Λέγονται πρόσθιας τροφοδότησης , γιατί με δεδομένη κάποια είσοδο x , για να υπολογίσουμε την εξοδο απαιτούνται υπολογισμοί, απο την είσοδο προς την έξοδο και όχι αντίστροφα.

3.2 Πολυεπίπεδο δίκτυο Perceptron – MLP

Ενα δίκτυο MLP αποτελείται απο έναν σύνολο κόμβων εισόδου που είναι οι αισθητήρες και αποτελούν το επίπεδο εισόδου, ένα ή περισσότερα κρυφά επίπεδα που αποτελούνται απο νευρώνες που εκτελούν υπολογισμούς και ένα επίπεδο εξόδου που αποτελείται επίσης απο νευρώνες που εκτελούν υπολογισμούς. Το βασικό χαρακτηριστικό του δικτύου είναι ότι οι νευρώνες του οποιαδήποτε στρώματος l τροφοδοτούν τους νευρώνες του επόμενου στρώματος l+1 και τροφοδοτούνται μόνο απο τους νευρώνες του προηγούμενου στρώμτος l-1 [3] . Παρακάτω έχουμε ενα MLP δίκτυο τριών επιπέδων με δυο κρυφά επίπεδα.



Σχήμα 3.2.1

πηγή

https://www.researchgate.net/figure/Architectural-graph-of-a-multilayer-perceptron-with-two-hidden-layers_fig4_228731551

Η ροή της πληροφορίας όπως είπαμε γίνεται απο αριστερά προς τα δεξιά και απο επίπεδο σε επίπεδο, το δίκτυο αυτό είναι πλήρως διασυνδεδεμένο , δηλαδή ένας

νευρώνας σε κάθε επίπεδο είναι συνδεδεμένος με όλους τους νευρώνες του προηγούμενου επιπέδου. Ενα MLP έχει τα παρακάτω χαρακτηριστικά:

1. Το μοντέλο κάθε νευρώνα στο δίκτυο περιλαμβάνει μια μη γραμμική συνάρτηση ενεργοποίησης στην έξοδο του , η οποία είναι παντού παραγωγίσιμη. Η πιο συνηθισμένη μορφή που χρησιμοποιείται είναι η σιγμοειδής συνάρτηση και ορίζεται απο την σχέση:

   (Σχέση 3.2.1)

   $$f(u) = \frac{1}{1+e^{-u}}$$

   Η συνάρτηση αυτή έχει συνεχείς τιμές σε σχέση με την βηματική που έχει βηματικές τιμές και δεν μπορεί να παραγογισθεί [3]. Η συνάρτηση ενεργοποίησης πρέπει να παραγωγίζεται διότι οι περισσότεροι κανόνες εκπαίδευσης βασίζονται σε μεθόδους βελτιστοποίησης που χρησιμοποιούν παραγώγους [3]. Επίσης η μη γραμμικότητα είναι σημαντική διότι σε διαφορετική περίπτωση η σχέση εισόδου – εξόδου του δικτύου θα μειωνότανε σε αυτήν ενός απλού Perceptron.

2. Το δίκτυο περιλαμβάνει ένα ή περισσότερα επίπεδα με κρυφούς νευρώνες που παραμένουν κρυφά για τους κόμβους των επιπέδων εισόδου και εξόδου [3]. Οι κρυφοί νευρώνες επιτρέπουν στο δίκτυο να μαθαίνει και να εκτελεί πολύπλοκες εργασίες εξάγοντας τα πιο σημαντικά χαρακτηριστικά απο τα διανύσματα εισόδου [3].
3. Το δίκτυο επιδεικνύει υψηλό βαθμό διασύνδεσης που καθορίζεται απο τα συναπτικά βάρη του δικτύου [3]. Μια αλλαγή στην συνδεσιμότητα του δικτύου απαιτεί μια αλλαγή στον πληθυσμό των συνδέσεων των συνάψεων ή στα βάρη τους.

Στο δίκτυο μεταδίδονται δύο ειδών σήματα:

1. Σήματα συναρτήσεων ή λειτουργικά σήματα (function signals) , τα οποία είναι σήματα εισόδου που φτάνουν στην είσοδο του δικτύου , μετά διαδίδονται προς τα εμπρός απο νευρώνα σε νευρώνα και τελικά καταλήγουν στους νευρώνες εξόδου του δικτύου ως σήματα εξόδου [3] . Σε κάθε νευρώνα του δικτύου απο όπου περνάει το σήμα υπολογίζεται σαν συνάρτηση όλων των εισερχομένων σημάτων και των συναπτικών βαρών που καταλήγουν στον συγκεκριμένο νευρώνα.
2. Σήματα σφάλματος (error signals). Ενα σήμα σφάλματος ξεκινάει απο τους νευρώνες εξόδου του δικτύου και διαδίδεται προς τα πίσω απο επίπεδο σε επίπεδο [3]. Σε κάθε νευρώνα το σήμα αυτό υπολογίζεται απο μια συνάρτηση που εξαρτάται απο το σφάλμα.

Κάθε κρυφός νευρώνας και κάθε νευρώνας εξόδου ενός MLP έχει σχεδιαστεί για να εκτελεί δύο υπολογισμούς:

1. Υπολογίζει το σήμα συνάρτησης ή λειτουργικό σήμα που εμφανίζεται στην έξοδο του νευρώνα και εκφράζεται σαν μια συνεχής μη γραμμική συνάρτηση των ση-

μάτων που εισέρχονται στον νευρώνα και των αντίστοιχων βαρών των συνάψεων που σχετίζεται με τον νευρώνα.
2. Επίσης μας δίνει μια εκτίμηση του διανύσματος κλίσης (δηλαδή τις κλίσεις της επιφάνειας σφάλματος ως προς τα βάρη που σχετίζονται με τις συνάψεις που εισέρχονται στον νευρώνα) που χρειάζεται για την πίσω διάδοση του σήματος μέσω του δικτύου.

Τα δίκτυα Perceptron πολών επιπέδων μπορούν να υλοποιήσουν συναρτήσεις που δεν μπορεί να υλοποιήσει ένα απλό δίκτυο Perceptron . Τα δίκτυα MLP που ενσωματώνουν την σιγμοειδή συνάρτηση μπορούν να προσεγγίσουν οποιαδήποτε συνεχή συνάρτηση όσο κοντά θέλουμε (2 επίπεδα φτάνουν) και τα ονομάζουνε καθολικούς προσεγγιστές (universal approximators) [3].

3.3 Ο αλγόριθμος της ανάδρομης διάδοσης σφάλματος ή backpropagation

Για να μπορέσουμε να εκπαιδέυσουμε νευρωνικά δίκτυα που να μπορούν να αναπαρισθούν πιο περίπλοκες διεργασίες έχουμε την μέθοδο της εκπαίδευσης της όπισθεν διάδοσης σφάλματος και εφαρμόζεται σε αρκετά περίπλοκα νευρωνικά δίκτυα, όπως το δίκτυο πολλαπλών επιπέδων. Το κάθε επίπεδο έχει πολλούς νευρώνες. Οι νευρώνες μέσα στο ίδιο επίπεδο δεν συνδέονται μεταξύ τους, αλλά οι νευρώνες που ανήκουν σε διαφορετικά επίπεδα συνδέονται ως συνήθως με τις γνωστές συνάψεις.

Ποια είναι λοιπόν η διαδικασία; Η λειτουργία και δομή τέτοιων δικτύων είναι ώς εξής: Καταρχάς το δίκτυο ξεκινά την διαδικασία μάθησης απο τυχαίες τιμές βαρών. Εάν δώσει λάθος απάντηση, τότε τα βάρη διορθώνονται μέχρις το σφάλμα να γίνει μικρότερο. Αυτό επαναλαμβάνεται πολλές φορές έτσι ώστε σταδιακά το σφάλμα ελαττώνεται μέχρις να γίνει πολύ μικρό και ανεκτό. Οπότε λέμε μετά ότι δίκτυο έχει μάθει αυτά που τους διδάξαμε με την ακρίβεια που θέλαμε να μάθει.

Η διαδικασία εκπαίδευσης κατα τον αλγόριθμο της όπισθεν διάδοσης σφάλματος λαμβάνει χώρα με την παρουσίαση και εφαρμογή στο δίκτυο ενός συνόλου παραδειγμάτων εκπαίδευσης [5]. Η παρουσίαση όλων των προτύπων εκπαίδευσης μια φορά το καθένα στο δίκτυο λέγεται εποχή (epoch). Οσο γίνεται η εκπαίδευση εκτελούνται επαναλήψεις των εποχών, μέχρι να σταθεροποιηθούν τα βάρη σε συγκεκριμένες τιμές που θα προκαλούν σύγκλιση της μέσης τιμής των σφαλμάτων, για όλα τα πρότυπα εκπαίδευσης, στην μικρότερη τιμή της. Μην ξεχάσουμε να αναφέρουμε ότι η εκπαίδευση γίνεται με επίβλεψη.

Η διαδικασία αποτελείται απο δυο περάσματα διαμέσου των επιπέδων του δικτύου, το ένα πέρασμα γίνεται προς τα εμπρός και το άλλο πέρασμα γίνεται προς τα πίσω. Στο εμπρός πέρασμα έχουμε το διάνυσμα εισόδου που θα εφαρμοστεί στους νευρώνες εισόδου και θα προχωρήσει μέσα στο δίκτυο απο επίπεδο σε επίπεδο μέχρι να φτάσει στο τέλος όπου έχουμε τους εξόδους που είναι και η πραγματική απόκριση του δικτύου. Στο πέρασμα αυτό τα βάρη είναι σταθερά [3]. Μετά έχουμε το πίσω πέρασμα όπου τα βάρη μεταβάλλονται σύμφωνα με τον κανόνα διόρθωσης του λάθους. Πιο συγκεκριμένα η πραγματική απόκριση (εξοδος) του δικτύου αφαιρείται απο την επιθυμητή έξοδο για να δημιουργηθεί ένα σήμα λάθους το οποίο και διαδίδεται προς τα πίσω στο δίκτυο. Μετά

τα συναπτικά βάρη προσαρμόζονται έτσι ώστε η πραγματική απόκριση (εξοδος) να πλησιάσει την επιθυμητή έξοδο.

Όπως έχουμε πει στα πολυεπίπεδα δίκτυα ή έξοδος ενός επιπέδου γίνεται είσοδος στο επόμενο επίπεδο. Εαν υποθέσουμε ότι έχουμε n εισόδους x1,x2,….,xn .Το πρώτο επίπεδο (κρυφό επίπεδο) αποτελείται απο p μη γραμμικούς νευρώνες που χρησιμοποιούν τη σιγμοειδή συνάρτηση ενεργοποίησης f . Η έξοδος ενός νευρώνα i του πρώτου επιπέδου είναι [3]:

(Σχέση 3.3.1)

$$yi = f(\sum_{j=1}^{n} wij\ xj + \theta i)$$

Το δεύτερο επίπεδο αποτελείται απο έναν μη γραμμικό νευρώνα που προσθέτει τις ενεργοποιήσεις του πρώτου επιπέδου χρησιμοποιώντας τα συναπτικά βάρη ω1,ω2,....ωp .Οπότε έχουμε:

(Σχέση 3.3.2)

$$g(x1, x2, \dots, xn) = \sum_{i=1}^{p} \omega i\ yi - \theta$$

Θα μπορούσε να περιγραφεί η διαδικασία με την εξίσωση:

(Σχέση 3.3.3)

$$y^{m+1} = f^{m+1}\ (W^{m+1} y^{m} + b^{m+1})\ \gamma\iota\alpha\ m = 0{,}1\ \dots . M - 1,$$

Όπου M είναι ο αριθμός των στρωμάτων του δικτύου. Οι νευρώνες στο πρώτο επίπεδο δέχονται εξωτερικές εισόδου y^0 = p.

Οι εξόδοι των νευρώνων του τελευταίου επιπέδου αποτελούν τις εξόδους του νευρωνικού δικτύου.

Εαν υποθέσουμε ότι x είναι η είσοδος, y είναι η έξοδος (πραγματική), d είναι ο στόχος (επιθυμητο) . Το ιδανικό θα ήταν να ταυτιστούν οι έξοδοι με τους στόχους για κάθε πρότυπο εισόδου , δηλαδή να γινόταν y^p = d^p για p =1,2,…,P, (P = διανύσματα εισόδου), το οποίο δεν μπορεί να γίνει ,και γιαυτό αναζητούμε την βέλτιστη προσέγγιση της επιθυμητής εξόδου, χρησιμοποιώντας ενα κριτήριο κόστους και αυτό είναι το μέσο τετραγωνικό σφάλμα (mse) ή J (Mean Square Error) .

(Σχέση 3.3.4)

$$\text{J ή mse} = \frac{1}{P}\sum_{P=1}^{P} |d^p - y^p|\text{^}2 = \frac{1}{P}\ \sum_{P=1}^{P}\sum_{i=1}^{m} \quad |d^p - y^p|\text{^}2$$

Η στιγμιαία τιμή της συνάρτησης ενέργειας σφάλματος ή στιγμιαίας τιμής τετραγωνικού σφάλματος δίνεται απο την σχέση:

(Σχέση 3.3.5)

$$E(p) = \frac{1}{2}\sum_{j=1}^{p} |d^p - y^P|^2$$

Όπως είχαμε αναφέρει και πιο πρίν είναι ότι στο πισω πέρασμα τα συναπτικά βάρη να προσαρμόζονται ,σε κάθε βήμα της επανάληψης , έτσι ώστε η συνάρτηση κόστους να μειώνεται. Η προσαρμογή των βαρών γίνεται σύμφωνα με τα σφάλματα που υπολογίζονται σε κάθε βήμα της διαδικασίας. Με άλλα λόγια ξεκινάμε απο μια τιμή συναπτικών βαρών wji(0) και εφαρμόζοντας μια διόρθωση παίρνουμε ένα νέο σέτ βαρών wji(1), και μετά άλλη μια διόρθωση και παίρνουμε νέο σετ βαρών wji(2) κ.ο.κ., έτσι ώστε σε κάθε επανάληψη να ισχύει:

$$E(W(n+1) \leq E(W(n)) \qquad \text{(Σχέση 3.3.6)}$$

Οι εισόδοι x^P και οι στόχοι d^P είναι δεδομένοι και σταθεροί. Σύμφωνα με την μέθοδο κατάβασης δυναμικού (steepest descent) ,η μεταβολή του wij ως προς το χρόνο t γίνεται χρησιμοποιώντας την παράγωγο του mse ή J ως προς wij [3].

(Σχέση 3.3.7)

$$\frac{dwij}{dt} = -\frac{\partial J}{\partial wij}$$

Και ονομάζεται κλίση του J ως προς το wij . Για να το εξομοιώσεις στον Η/Υ δεν είναι εφικτό λόγω ότι χρησιμοποιεί τον συνεχή χρόνο t , οπότε η κατάβαση δυναμικού εξομειώνεται με την αντίστοιχη εξίσωση στον διακριτό χρόνο k

(Σχέση 3.3.8)

$$wij(l, k+1) - wij(l, k) = -\beta \frac{\partial J}{\partial wij(l,k)}$$

Εδώ έχουμε μια παράμετρο β ,η οποία είναι ένας θετικός αριθμός και είναι συνάρτηση του k [1] [3] .

Η έξοδος του νευρώνα i του στρώματος l δίνεται απο την σχέση:

$$yi^k(l) = f(ui^k(l)) \qquad \text{(Σχέση 3.3.9)}$$

οπου

(Σχέση 3.3.10)

$$ui^k(l) = \sum_{j=1}^{N} wij(l,k)yj^k(l-1) + wi0(l,k)$$

$ui^k(l)$ Είναι η λεγόμενη δικτυακή διέγερση (net – input) του νευρώνα και είναι το άθροισμα των διεγέρσεων των νευρώνων του προηγούμενου στρώματος συνδυασμένων με τα συναπτικά βάρη wij(l,k) [1].

Για τον υπολογισμό της κλίσης του J ως προς το συναπτικό βάρος wij(l,k) θα πρέπει να οριστεί η τοπική κλίση (η σφάλμα) του νευρώνα I την χρονική στιγμή k σύμφωνα με την σχέση [3]:

(Σχέση 3.3.11)

$$\delta i^k(l) = -\frac{\partial J}{\partial ui^k(l)}$$

Απο το γράμμα δέλτα που χρησιμοποιείται για τον συμβολισμό αυτής της παράστασης, ο κανόνας αυτός παίρνει και το όνομα <<Γενικευμένος κανόνας Δέλτα>>. Η παράμετρος δi σχετίζεται με το σφάλμα του νευρώνα i. Αρα χρησιμοποιώντας αυτόν τον συμβολισμό μπορούμε να γράψουμε [1]:

(Σχέση 3.3.12)

$$\frac{\partial J}{\partial wij(l,k)} = \frac{\partial J}{\partial ui^k(l)}\frac{\partial ui^k(l)}{\partial wij(l,k)} = -\delta i^k(l)\frac{\partial ui^k(l)}{\partial wij(l,k)}$$

οπου $ui^k(l)$ είναι η δικτυακή διέγερση (ή τοπικό πεδίο του νευρώνα i)

ισχύει $\frac{\partial ui^k(l)}{\partial wij(l,k)} = yj^k(l-1)$ *(Σχέση 3.3.13)*

οπότε

(Σχέση 3.3.14)

$$\frac{\partial J}{\partial wij(l,k)} = -\delta^\kappa(l) yj^k(l-1)$$

Για j = 0,1,2,…..,N και l =l,2,…..,L

Για την έυρεση των τοπικών κλίσεων $\delta i^k(l)$ για όλους του νευρώνες i, ξεκινάμε απο το επίπεδο εξόδου και με κατεύθυνση προς τα πίσω φτάνουμε στο πρώτο επίπεδο και πιο συγκεκριμένα έχουμε [3]:

1. *Για το στρώμα εξόδου L*

Η παράγωγος του κόστους J ως προς την διέγερση $ui^k(L)$ είναι [1] :

(Σχέση 3.3.15)

$$\delta i^k(L) = -\frac{\partial J}{\partial ui^k(L)} = -\frac{\partial J}{\partial yi^k(L)}\frac{\partial yi^k(L)}{\partial ui^k(L)}$$

Απο τον υπολογισμό του πρώτου κλάσματος, ισχύει [3]:

(Σχέση 3.3.16)

$$\frac{\partial J}{\partial yi^k(L)} = -(di^k - yi^k)$$

Αρα αντικαθιστώντας την μερική παράγωγο του J στον παραπάνω τύπο της $\delta i^k(L)$ οπότε προκύπτει ότι

(Σχέση 3.3.17)

$$\delta i^k(L) = (di^k - yi^k)\frac{\partial fui^k(L)}{\partial ui^k(L)} => \delta i^k(L) = (di^k - yi^k)f'(ui^k(L))$$

Η τοπική κλίση ή τιμή του σφάλματος $\delta i^k(L)$ που παίρνουμε είναι η διαφορά της εξόδου του νευρώνα i απο την αντίστοιχη επιθυμητή έξοδο, πολλαπλασιασμένη με την παράγωγο της συνάρτησης ενεργοποίησης f [1]. Ανάλογα με την συγκεκριμένη συνάρτηση ενεργοποίησης που χρησιμοποιούμε παίρνουμε και διαφορετικό τύπο για την παράγωγο f' . Οι πιο συχνά χρησιμοποιούμενες συναρτήσεις είναι η σιγμοειδής , η υπερβολική και η γραμμική [1]. Οπότε για αυτές έχουμε την παράγωγο:

(α) Για την σιγμοειδή συνάρτηση: $f(u) = \frac{1}{1+e^{-u}}$, η παράγωγος είναι f'(u) = f(u)(1-f(u)) [3]

(b) Για την υπερβολική εφαπτόμενη: $f(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$ η παράγωγος f'(u) =(1+ f(u))(1-f(u)) [3]

(c) Για την γραμμική συνάρτηση f(u) = u , η παράγωγος f'(u) = 1 [3]

2. *Για οποιαδήποτε στρώμα l =1,…..,L-1 [1] εχουμε το εξής σφάλμα:*

(Σχέση 3.3.18)

$$\delta i^k(l) = -\frac{\partial J}{\partial ui^k(l)} = -\sum_{\mu=1}^{N(l+1)} \frac{\partial J}{\partial u\mu^k(l+1)} \frac{\partial u\mu^k(l+1)}{\partial yi^k(l)} \frac{\partial yi^k(l)}{\partial ui^k(l)}$$

Και οι πράξεις μας δίνουν [3]

(Σχέση 3.3.19)

$$\delta i^k(l) = \sum_{\mu=1}^{N(l+1)} \delta\mu^K(l+1) w\mu i^k(l+1) f'(ui^k(l))$$

Οπου $\delta\mu^K(l+1)$ είναι το σφάλμα ενός νευρώνα μ που βρίσκεται στο επόμενο επίπεδο l+1 [3]. Απο τον παραπάνω τύπο φαίνεται ότι το σφάλμα δ σε οποιαδήποτε νευρώνα του

στρώματος l είναι συνάρτηση των σφαλμάτων δ του στρώματος l+1 [3]. Δηλαδή τα σφάλματα μοιάζουν να πηγάζουν απο το τελευταίο στρώμα και να προωθούνται προς τα πίσω δηλαδή προς το πρώτο στρώμα. Θα Πρέπει να αναφέρουμε ότι τα σφάλματα που προωθούνται προς τα πίσω χρησιμοποιώντας τα ίδια βάρη του δικτύου που χρησιμοποιούνται και στην ανάκληση. Ο αλγόριθμος ονομάζεται Back – propagation (BP).

Κάνοντας κάποιες αντικαταστάσεις παίρνουμε την τελική μορφή του τύπου

$wij(l,k+1) - wij(l,k) = \beta\delta\iota^K(l)yj^k(l-1)$ *(Σχέση 3.3.20)*

Τα βήματα του επαναληπτικού αλγόριθμου Back – propagation είναι:

Εισόδοι: P ζεύγη διανυσμάτων εισόδων – στόχων $\{x^p, d^p\}$ *[1]*

Εισόδοι: Τα εκπαιδευμένα βάρη wij(l) [1]

Αρχικοποίηση των βαρών wij(l) σε μικρές τυχαίες τιμέςγια κάθε στρώμα l. Το βάρος wi0(l) αντιστοιχεί στο κατώφλι του νευρώνα i του στρώματος l [1].

Θέσε εποχή

Επανέλαβε για κάθε εποχή = 1,2,…..,MAXepoch

{

Για κάθε πρότυπο p= 1,…,P

{

/ Φάση ανάκλησης = forward phase*/*

Υπολόγισε τις εξόδους yi(1) του στρώματος 1 σύμφωνα με τον τύπο [3]:

(Σχέση 3.3.21)

$$yi(l) = f(ui(l)) = f\Big(\sum_{j=1}^{N(l-1)} wij(l)yj(l-1) + wi0(l)\Big)$$

l =1,2,….,L

Υπολόγισε τις εξόδους Yi(L) του στρώματος L.

/ Φάση υπολογισμού δ = backward phase */*

Υπολόγισε τα σφάλματα δi(L) του στρώματος L σύμφωνα με τον τύπο [3]:

(Σχέση 3.3.22)

$$\delta i^k(L) = (di^k - yi^k)f'(ui^k(L))$$

Υπολόγισε τα σφάλματα δi(L-1) του στρώματος L-1 σύμφωνα με τον τύπο [3]:

(Σχέση 3.3.23)

$$\delta i^k(l) = f'(ui^k(l)) \sum_{\mu=1}^{N(l+1)} \delta\mu \quad (l+1) w\mu i \quad (l+1)$$

l =1,2,…..L-1

Υπολόγισε τα σφάλματα δi(l) του στρώματος 1

*/*φάση ενημέρωσης βαρών = Update phase*/*

Ενημέρωσε τα βάρη όλων των στρωμάτων σύμφωνα με τον τύπο [3]:

(Σχέση 3.3.24)

$$wij(l, k+1) = wij(l, k) + \beta\delta i^K(l) yj^k(l-1)$$

Για j = 0,1,…...,N(l-1) και l =1,…...,L

}

Μεχρι το συνολικό σφάλμα J σε μια εποχή να είναι μικρότερο απο κάποιο κατώφλι ε ή θ

}

*3.4 **Η παράμετρος του βήματος εκπαίδευσης β ή ρυθμός μάθησης***

Το βήμα εκπαίδευσης β επηρεάζει την ταχύτητα σύγκλισης του αλγόριθμου [3]. Αν επιλέξουμε β πολυ μικρό τότε ο αλγόριθμος θα συγκλίνει στο κοντινότερο τοπικό ελάχιστο, όπως ακριβώς θα συνέβαινε με την πραγματική κατάβαση δυναμικού, αλλά η σύγκλιση μας θα είναι πιο αργή [1]. Αν όμως επιλέξουμε β μεγαλύτερο τότε πετυχαίνουμε ταχύτερη σύγκλιση. Απο κάποιο σημείο και μετά ο κανόνας BP θα αποτελεί πλέον φτωχή προσέγγιση της κατάβασης δυναμικού ,με αποτέλεσμα να έχουμε κακή σύγκλιση(ταλαντώσεις) ή ακόμα χειρότερα απόκλιση του αλγορίθμου, δηλαδή οι τιμές των βαρών θα οδηγούνται στο ±∞(απειρο). Για την σωστή επιλογή του β χρησιμο-ποιείται η μέθοδος δοκιμής και σφάλματος και οι τιμές που χρησιμοποιούνται είναι μικρότερες έως πολύ μικρότερες της μονάδας. Αν το δίκτυο έχει πολλές εισόδους ή οι είσοδοι έχουν μεγάλες τιμές τότε θα πρέπει το βήμα εκπαίδευσης β να έχει μικρή τιμή [1]. Αν όμως έχει λίγες εισόδους ή εισόδοι με μικρές τιμές τότε το β μπορεί να έχει μεγαλύτερη τιμή. Ακόμα μπορεί να επιτεχθεί ταχύτερη σύγκλιση αν χρησιμοποιούνται διαφορετικά βήματα εκπαίδευσης για κάθε στρώμα του δικτύου ή ακόμα και για κάθε συναπτικό βάρος ξεχωριστά [1]. Συνήθως τα τελευταία στρώματα (που είναι κοντά στην έξοδο) έχουν μεγαλύτερη παράγωγο $\frac{\partial J}{\partial wij}$ *σε σχέση με τα πρώτα στρώματα [1]. Οπότε αν χρησιμοποιηθεί το ίδιο βήμα εκπαίδευσης β σε όλα τα στρώματα , τότε η διόρθωση των βαρών είναι μεγαλύτερη στα τελευταία στρώματα απο ότι στα πρώτα. Για να εξισο-ρροπηθεί η ταχύτητα εκπαίδευσης σε όλα τα στρώματα μπορούμε να χρησιμοποιήσουμε μεγαλύτερο βήμα εκπαίδευσης β στα πρώτα στρώματα και μικρότερο β στα τελευταία στρώματα [1].*

$$\Delta wij(k) = -\beta \frac{\partial J}{\partial wij(k)} \quad \textit{(Σχέση 3.4.1)}$$

Αν χρησιμοποιηθεί διαφορετικό βήμα εκπαίδευσης βij για κάθε βάρος wij ξεχωριστά,τότε ο κανόνας BP γίνεται [3] [1]:

$$\Delta wij(k) = -\beta ij \frac{\partial J}{\partial wij(k)} \quad \textit{(Σχέση 3.4.2)}$$

Η λογική είναι ότι το κάθε βάρος είναι ξεχωριστό και μπορεί να χρειάζεται πιο γρήγορους ή πιο αργούς ρυθμούς εκπαίδευσης απο τα άλλα βάρη [3]. Σκεπτόμενοι λοιπόν μπορούμε να πούμε ότι τα βήματα μπορούν να αλλάζουνε απο επανάληψη σε επανάληψη ή απο εποχή σε εποχή. Το θέμα είναι ότι δεν υπάρχουν ορισμένα κριτήρια που να σταματούν την λειτουργία του αλγόριθμου BP ,αλλά υπάρχουνε κάποια λογικά κριτήρια που μπορούν να χρησιμοποιηθούν για να τερματίσουν τις ρυθμίσεις των βαρών. Τα κριτήρια αυτά περιλαμβάνουν το κόστος J ή τα συναπτικά βάρη wij [3]. Δηλαδή:

Α. Με βάση το κόστος J έχουμε:

a. Το κόστος J(n) για κάποια εποχή εκπαίδευσης n είναι μικρότερο απο κάποιο κατώφλι ε ή θ που ορίζεται απο τον χρήστη [3]. Για να μπορέσουμε να το υπολογίσουμε το J(n) θα πρέπει να κρατήσουμε σταθερά τα βάρη και να εισάγουμε όλα τα πρότυπα εισόδου υπολογίζοντας το σφάλμα [1] . Το J(n) δίνεται απο τον τύπο:

(Σχέση 3.4.3)

$$J = \frac{1}{P} \sum_{P=1}^{P} \sum_{i=1}^{m} \quad |di^p - yi^p|\text{^}2$$

b. Το σφάλμα σε δύο διαδοχικές εποχές n-1, n δεν μειώνεται σημαντικά , δηλαδή ισχύει ότι J(n-1) – J(n) < ε οπου ε είναι το κατώφλι που ορίζει ο χρήστης [3].

Β. Με βάση τα συναπτικά βάρη έχουμε:

Κατα την εποχή n τα συναπτικά βάρη αλλάζουν πολύ λίγο και αυτό μπορεί να μετρηθεί αν προσθέουμε τις μεταβολές των βαρών υψωμένες στο τετράγωνο ή σε απόλυτο τιμή και συγκριθούν με κάποιο κατώφλι ε που ορίζει ο χρήστης [1] [3]:

(Σχέση 3.4.4)

$$\sum_{i,j} (\Delta wij)^2 < \varepsilon$$

είτε

(Σχέσση 3.4.5)

$$\sum_{i,j} (\Delta wij) < \varepsilon$$

3.5 Μειονεκτήματα του αλγόριθμου Back Propagation – Αρχικοποίηση των βαρών

Μέχρι στιγμής αναφέραμε τα θετικά του αλγόριθμου οπισθο διάδοσης ,σε κάποιες περιπτώσεις έχει και αποτυχίες ο αλγόριθμος . Σε αυτές τις περιπτώσεις θα πρέπει να αλλαχθούν οι τιμές των παραμέτρων ή οι αρχικές συνθήκες μέχρι να διορθωθεί το πρόβλημα [3]. Κάποιες φορές η διάρκεια εκπαίδευσης είναι πολυ μεγάλη. Δηλαδή παίρνει χιλιάδες η εκατομμύρια κύκλους διόρθωσης για να συγκλίνει το σύστημα ή μπορεί και να μην συγκλίνει ποτέ [3]. Οπότε τότε θα πρέπει να αλλάξει το μέγεθος του βήματος εκπαίδευσης β. Καμιά φορά χρειάζονται λίγες επαναλήψεις που πραγματοποιούνται σε λίγα λεπτά ακόμα και για μεγάλο όγκο δεδομένων.

Επίσης άλλο μειονέκτημα είναι ότι μπορεί ο αλγόριθμος να συγκλίνει σε κάποιο τοπικό ελάχιστο της συνάρτησης κόστους αντί στο ολικό [3]. Το τοπικά ελάχιστα λέμε ότι είναι ανεπιθύμητα διότι είναι υποδιαίστερες λύσεις του προβλήματος σε σχέση με το ολικό ελάχιστο [3]. Λόγο αυτού του γεγονότος, η διαδικασία μάθησης μπορεί να σταματήσει πρόωρα, δηλαδή στην αρχή του αλγόριθμου οπότε δεν θα έχουμε τα αναμενόμενα αποτελέσματα.

*Για την αντιμετώπιση αυτού του προβλήματος θα πρέπει απο την αρχή τα βάρη να αρχικοποιούνται σε μικρές τιμές [3]. Εαν βάλουμε μεγάλες τιμές βαρών θα πάμε σε **κορεσμό** κατά τον οποίο η μεταβολή των συναπτικών βαρών θα είναι πολύ μικρή όχι όμως διότι το δίκτυο θα έχει φτάσει σε καλό σημείο, αλλά διότι η έξοδος των νευρώνων είναι πολύ κοντά στο όριο της συνάρτησης ενεργοποίησης f [1]. Θα πρέπει να αναφερθεί ότι στον κανόνα διόρθωσης του βάρους wij εμπλέκεται το σφάλμα δi το οποίο με την σειρά του περιέχει την παράγωγο της συνάρτησης ενεργοποίησης f'(ui). Αν χρησιμοποιήσουμε την σιγμοειδή συνάρτηση ενεργοποίησης, της οποίας η παράγωγος της f είναι:*

(Σχέση 3.5.1)

$$f'(ui) = f(ui)\,[1 - f(ui)]$$

Αν υπάρχουν μεγάλα βάρη τότε το ui παίρνει μεγάλες θετικές ή αρνητικές τιμές και με την σιγμοειδή συνάρτηση ενεργοποίησης η έξοδος του νευρώνα f(ui) πάει κοντά στα όρια 0 και 1. Τότε η παράγωγος f' είναι σχεδόν μηδέν και η διόρθωση του συναπτικού βάρους είναι πρακτικά αμελητέα [1].

Αυτό αποδεικνύεται απο τον τύπο της δικτυακής διέγερσης του νευρώνα i που είναι:

(Σχέση 3.5.2)

$$ui = \sum_j wij\ yj + wi0$$

η οποία σπρώχνεται σε μεγάλες θετικές ή αρνητικές τιμές και η έξοδος f(ui) του νευρώνα πάει κοντά σε οριακές τιμές 0 και 1 [1]. Οπότε το αντίστοιχο θα συμβεί αν επιλέξουμε την συνάρτηση ενεργοποίησης της υπερβολικής συνάρτησης [1]:

(Σχέση 3.5.3)

$$f(u) = \tanh(u)$$

η παράγωγος είναι:

(Σχέση 3.5..4)

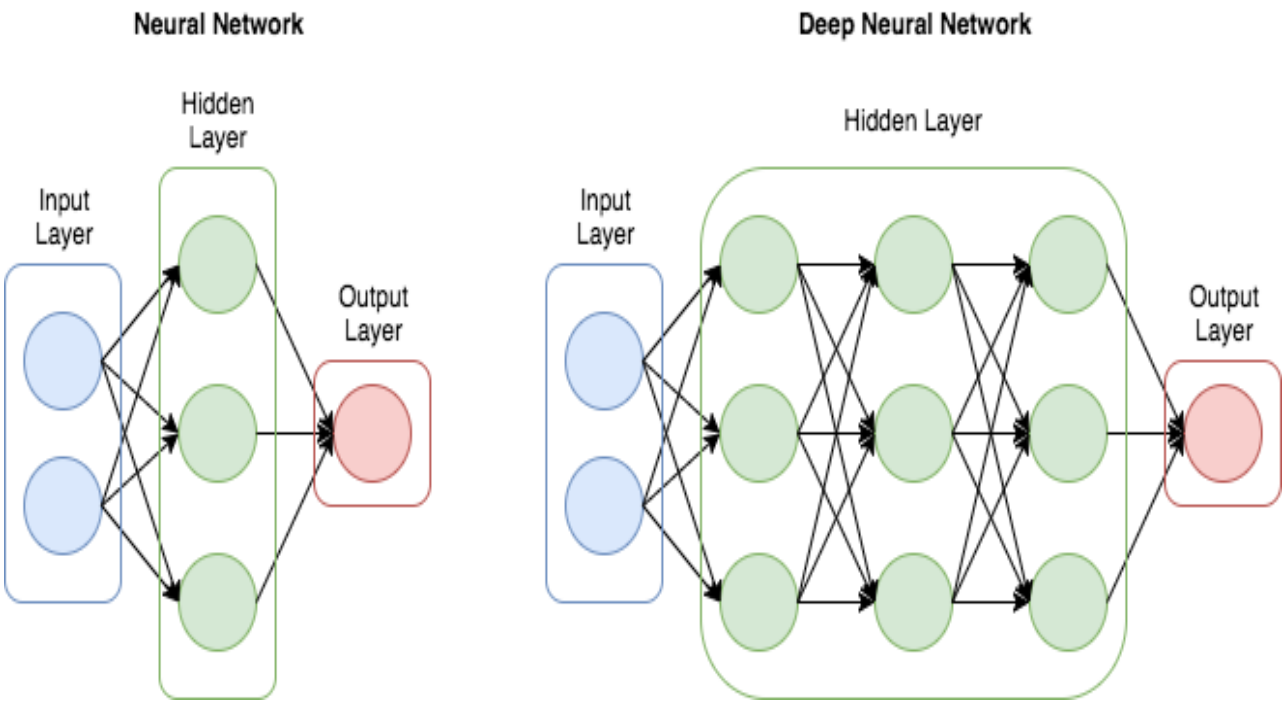
$$f'(ui) = [1 + f(ui)]\,[1 - f(ui)]$$

αλλά τα όρια του f είναι -1 και +1.

3.6 Το πρόβλημα της XOR

'Οπως είχαμε δει στο κεφάλαιο 2 όσον αφορά για ένα απλό αισθητήρα που δεν μπορούσε να δώσει σωστά αποτελέσματα σε μη γραμμικά διαχωρίσιμα προβλήματα, καθώς δεν μπορεί να χωρίσει το επίπεδο σε περισσότερες απο δυο περιοχές. Ψάχνουμε τρόπο να συνδυάσουμε τα αποτελέσματα δύο διαφορετικών αισθητήρων ,τότε θα είναι δυνατόν να χωρίσουμε το επίπεδο σε περισσότερες περιοχές και να έχουμε το επιθυμητό αποτέλεσμα.

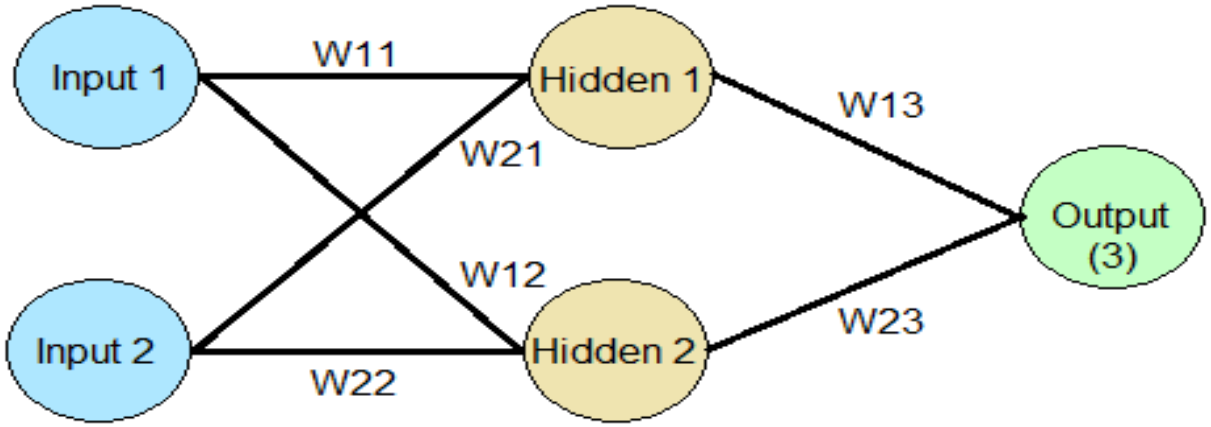
Τώρα πλέον μπορούμε να ξαναδούμε το παράδειγμα της XOR . Θεωρούμε και πάλι ένα νευρωνικό δίκτυο με δυο εισόδους και μια έξοδο , θα παρεμβάλλουμε και ένα κρυφό επίπεδο με δύο νευρώνες μεταξύ τους , όπως στο σχήμα παρακάτω.



Σχήμα 3.6.1

πηγή

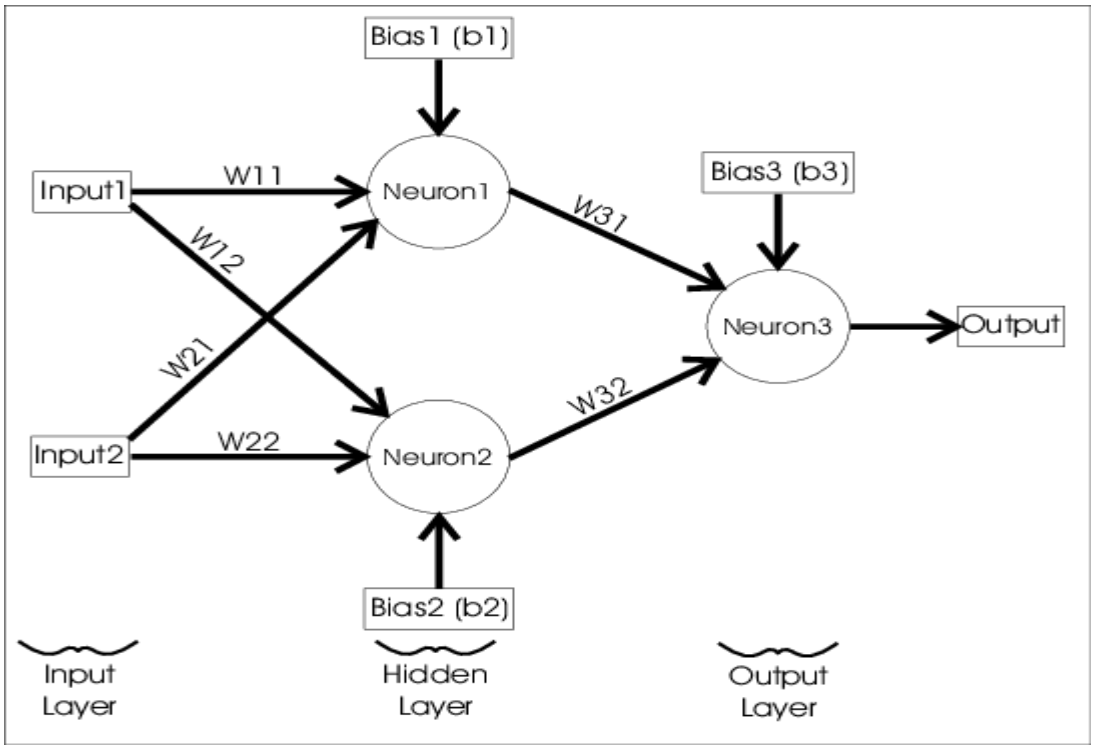
https://www.imagenesmy.com/imagenes/dnn-network-8c.html



Σχήμα 3.6.2

πηγή

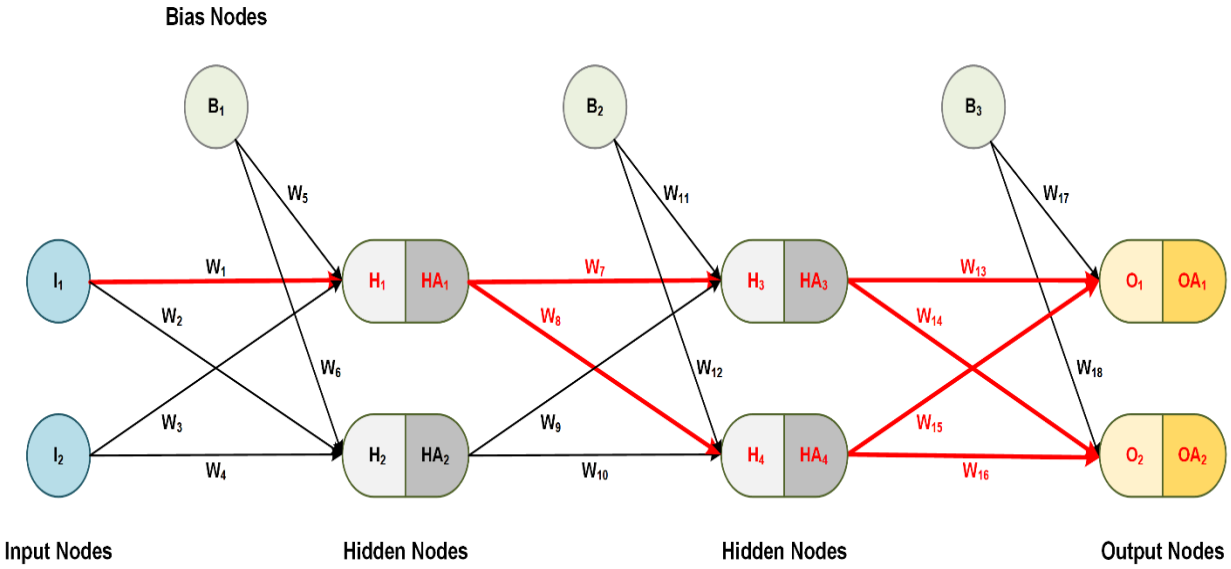
https://www.pintherest.eu/backpropagation-in-neural-networks.html



Σχήμα 3.6.3

πηγή

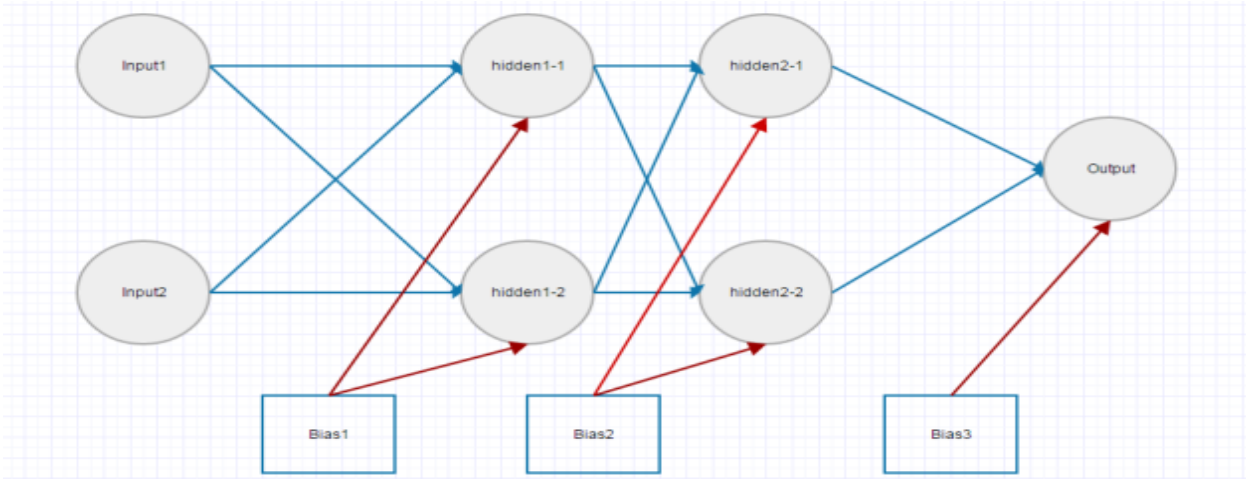
https://www.researchgate.net/figure/Topology-of-ANN-used-to-solve-logic-XOR-problem_fig1_228939274



Σχήμα 3.6.4

πηγή

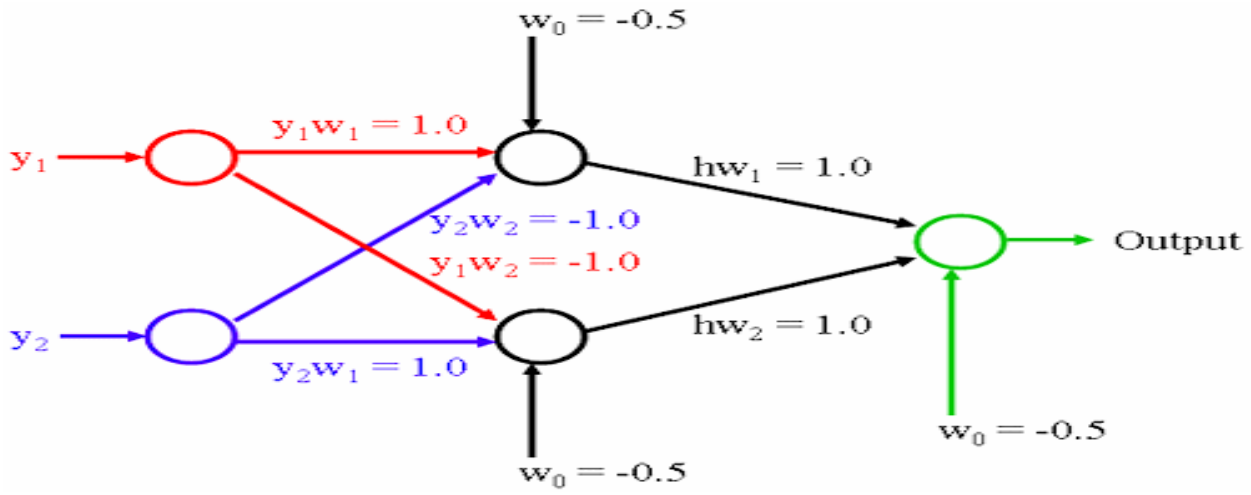
https://pintherest.eu/pdf-convolutional-neural-networks-using-supervised-pre-training-for.html



Σχήμα 3.6.5

πηγή

https://gmgolem.wordpress.com/tag/xor-problem/



Σχήμα 3.6.6

πηγή

http://sunkidesign.info/xor-problem-in-neural-network-13.html

Κάνουμε τις ακόλουθες υποθέσεις:

(α) Κάθε μοντέλο αναπαρίστανται απο ένα μοντέλο McCulloch-Pitts, το οποίο χρησιμοποιεί μια συνάρτηση κατωφλίου σαν συνάρτηση ενεργοποίησης του [8].

(β) Τα bits 0 και 1 αναπαρίστανται απο τις στάθμες 0 και +1, αντίστοιχα [8].

Ο πάνω νευρώνας (Νευρώνας 1 στο κρυφό επίπεδο), χαρακτηρίζεται ως εξής [8]:

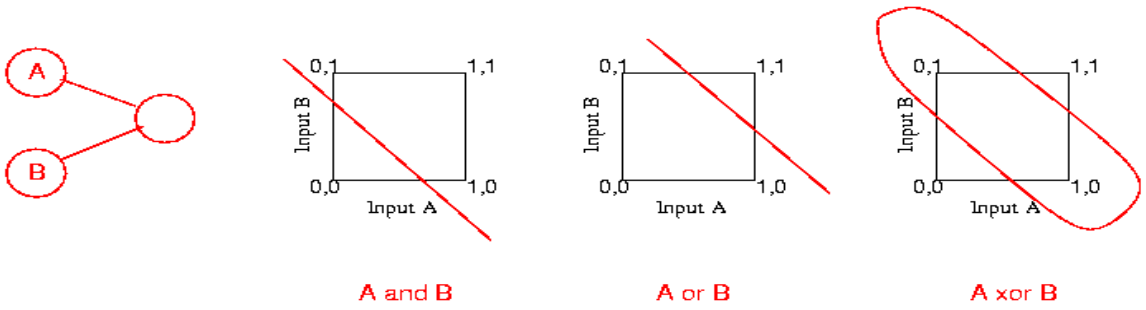
$$w11 = w12 = +1 \quad b1 = -\frac{3}{2}$$

Η κλίση του ορίου απόφασης που δημιουργείται απο αυτό τον κρυφό νευρώνα ισούται με -1 [8]

Ο κάτω νευρώνας (Νευρώνας 2 στο κρυφό επίπεδο), χαρακτηρίζεται ως εξής [8]:

$$w21 = w22 = +1 \quad b2 = -\frac{1}{2}$$

Η λειτουργία του νευρώνα εξόδου είναι να δημιουργεί ένα γραμμικό συνδυασμό των ορίων απόφασης που δημιουργούν οι δυο κρυφοί νευρώνες [8]. Το αποτέλεσμα αυτού του υπολογισμού φαίνεται στο παρακάτω σχήμα.



Σχήμα 3.6.7

πηγή

http://web.science.mq.edu.au/~cassidy/comp449/html/ch13.html

Ο κάτω κρυφός νευρώνας έχει διεγερτική (θετική) σύνδεση με τον νευρώνα εξόδου , ενώ ο πάνω κρυφός νευρώνας έχει ανασταλτική (αρνητική) σύνδεση με τον νευρώνα εξόδου [8]. Οταν και οι δυο κρυφοί νευρώνες είναι ανενεργοί , που σημαίνει ότι αυτό που συμβαίνει όταν το πρότυπο εισόδου είναι (0,0) , ο νευρώνας εξόδου παραμένει ανενεργός [8]. Ενώ όταν και οι δυο κρυφοί νευρώνες είναι ενεργοί ,πράγμα το οποίο συμβαίνει όταν το πρότυπο εισόδου είναι (1,1) , ο νευρώνας εξόδου απενεργοποιείται ξανά επειδή η ανασταλτική επίδραση του μεγαλύτερου αρνητικού βάρους που συνδέεται με τον πάνω κρυφό νευρώνα υπερισχύει έναντι της διεγερτικής επίδρασης του θετικού βάρους που συνδέεται με τον κάτω κρυφό νευρώνα [8]. Οταν ο πάνω κρυφός νευρώνας είναι ανενεργός και ο κάτω κρυφός νευρώνας είναι ενεργός, αυτό γίνεται όταν το πρότυπο εισόδου είναι (0,1) ή (1,0) , ο νευρώνας εξόδου ενεργοποιείται λόγω της διεγερτικής επίδρασης του θετικού βάρους που συνδέεται με τον κάτω κρυφό νευρώνα [8].

Τελικά ο νευρώνας του επιπέδου εξόδου συνδυάζει τα αποτελέσματα των δυο προηγούμενων (δηλαδή των δυο νευρώνων του κρυφού επιπέδου) και δίνει το επιθυμητό αποτέλεσμα. Τοποθετεί επιτυχώς τα σετ (0,0) και (1,1) στην κλάση 0 και τα (0,1) και (1,0) στην κλάση 1 [2]. Μπορούμε να πούμε ότι λύνει το πρόβλημα της XOR.

3.7 Τρόποι εκπαίδευσης ενός δικτύου MLP

Γενικά στον αλγόριθμο πίσω διάδοσης , η μάθηση επιτυγχάνεται εφαρμόζοντας ένα σύνολο απο διανύσματα εκπαίδευσης στην είσοδο του πολυεπίπεδου Perceptron. Η πλήρης παρουσίαση όλων των διανυσμάτων εκπαίδευσης στο δίκτυο ονομάζεται κύ-κλος ή εποχή εκπαίδευσης (epoch) [3]. Η διαδικασία μάθησης πάει απο κύκλο σε κύκλο μέχρι να σταθεροποιηθούν τα βάρη και τα κατώφλια του δικτύου και το μέσο τετραγω-νικό σφάλμα όλων των διανυσμάτων εκπαίδευσης μέχρι να φτάσει σε κάποιο ελάχιστο. Έχουμε δυο τρόπους για όσον αφορά την παρουσίαση των διανυσμάτων εκπαίδευσης:

(α) Έρχονται τα δείγματα εκπαίδευσης το ένα μετά το άλλο στην σειρά μέχρι να εμφανιστούν όλα και να έχουμε μια εποχή εκπαίδευσης . Αυτό θα γίνει σε όλες τις εποχές.

(β) Ερχονται τα δείγμα εκπαίδευσης σε τυχαία σειρά, μέχρι σε κάθε κύκλο εκπαίδευσης να παρουσιαστούν όλα τα δείγματα απο μια και μόνο μια φορά το καθένα. Λέμε επίσης ότι η τυχαία αυτή σειρά διαφέρει σε κάθε κύκλο εκπαίδευσης.

Παρακάτω παρουσιάζονται οι τρόποι για την εκπαίδευση του δικτύου:

(α) Τρόπος Προτύπων (Pattern Mode) ή On Line Μάθηση

Με αυτήν την μέθοδο επιβλεπόμενης μάθησης , οι προσαρμογές στα συναπτικά βάρη του Perceptron πολλών επιπέδων εκτελούνται σε βάση παράδειγμα προς παράδειγμα, με την παρουσίαση κάθε διανύσματος εκπαίδευσης [3]. Η συνάρτηση κόστους που θα ελαχιστοποιηθεί είναι:

(Σχέση 3.7.1)

$$J = \frac{1}{2}\sum_{i=1}^{m} |di^{p} - yi^{P}|^{2}$$

Θεωρούμε δεδομένη μια εποχή με N παραδείγματα εκπαίδευσης , οργανωμένα σε σειρά {x(1),d(1)}, {x(2),d(2),…….,{x(N),d(N)}. Το πρώτο ζεύγος παραδείγματος {x(1),d(1)} σε

αυτήν την εποχή παρουσιάζεται στο δίκτυο και οι προσαρμογές των βαρών εκτελούνται χρησιμοποιώντας την μέθοδο βαθμωτής κατάβασης (gradient descent). Μετά έρχεται το δεύτερο παράδειγμα {x(2),d(2)} το οποίο οδηγεί σε περεταίρω προσαρμογές των βαρών του δικτύου [3]. Αυτό θα συνεχιστεί μέχρι το τελευταίο παράδειγμα {x(N),d(N)}. Δυστυχώς μια τέτοια διαδικασία δεν επιτρέπει την παράλληλη υλοποίηση της on-line μάθησης.

(β) Μαζική μάθηση (Batch mode)

Στην μαζική μέθοδο επιβλεπόμενης μάθησης , οι προσαρμογές στα συναπτικά βάρη του Perceptron πολλών επιπέδων εκτελούνται μετά απο την παρουσίαση του συνόλου των N παραδειγμάτων του δείγματος εκπαίδευσης τα οποία αποτελούν μια εποχή εκπαίδευ-σης [3]. Η συνάρτηση κόστους για την μαζική μάθηση προς ελαχιστοποίηση είναι:

(Σχέση 3.7.2)

$$J = \frac{1}{2N}\sum_{n=1}^{N} \sum_{i=1}^{m} [di^{n} - yi^{n}]^{2}$$

Οι διορθώσεις των βαρών γίνονται εποχή προς εποχή [3].

Αν συγκρίνουμε τις δυο μεθόδους παρατηρούμε ότι η on-line μάθηση προτιμάται καλύ-τερα απο την μαζική μάθηση διότι απαιτεί πολύ λιγότερο χώρο αποθήκευσης για κάθε σύναψη απο την μαζική μάθηση. Επίσης μπορεί να παρακολουθεί τις μικρές μεταβολές στα δεδομένα εκπαίδευσης [3]. Ενα ακόμα είναι ότι τα παραδείγματα εκπαίδευσης παρουσιάζονται στο δίκτυο με τυχαία σειρά η χρήση της on-line μάθησης καθιστά την αναζήτηση στον πολυδιάστατο χώρο βαρών στοχαστική διαδικασία η οποία μειώνει τις πιθανότητες η διαδικασία μάθησης να παγιδευτεί σε ένα τοπικό ελάχιστο ,που είναι ένα σαφώς πλεονέκτημα της on-line μάθησης σε σχέση με την μαζική μάθηση.

Μια άλλη χρήσιμη ιδιότητα της on-line μάθησης είναι η δυνατότητα της να παρακολουθεί τις μικρές μεταβολές στα δεδομένα εκπαίδευσης, ιδιαίτερα όταν το περιβάλλον που είναι υπεύθυνη για τη παραγωγή των δεδομένων είναι μη-στατικό [8].

Γενικά η on-line μάθηση είναι πολύ δημοφιλής για την επίλυση προβλημάτων ταξινόμησης προτύπων για δυο σημντικούς λόγους [8]:

(α) Η on-line μάθηση είναι καλή στην υλοποίηση [8].

(β) Παρέχει αποτελεσματικές λύσεις σε μεγάλης κλίμακας και αυξημένης δυσκολίας προβλήματα ταξινόμησης προτύπων [8].

3.8 Βελτίωση της απόδοσης του αλγόριθμου BP με ευρετικές μεθόδους

Υπάρχουν λοιπόν κάποιοι μεθόδου οι οποίοι μπορούν να βελτιώσουν σημαντικά την απόδοση του αλγόριθμου Back Propagation (BP) ,αυτοί είναι οι παρακάτω:

1. *Στοχαστική έναντι μαζικής ενημέρωσης*

Η στοχαστική (ακολουθιακή) διαδικασία μάθησης με οπισθοδιάδοση σφάλματος (που απαιτεί ενημέρωση πρότυπο προς πρότυπο) είναι πιο γρήγορη στον υπολογισμό απο την διαδικασία μαζικής μάθησης [8]. Αυτό το βλέπουμε πιο πολύ όταν το δείγμα δεδομένων εκπαίδευσης είναι μεγάλο και επιδεικνύει υψηλό πλεονασμό. Ο υψηλός πλεονασμός στα δεδομένα ορίζει υπολογιστικά προβλήματα όσον αφορά τον υπολογισμό του ιακωβιανου πίνακα για την μαζική ενημέρωση [8].

2. *Μεγιστοποίηση του περιεχομένου της πληροφορίας*

Μπορούμε να πούμε ότι κάθε παράδειγμα που παρουσιάζεται στον αλγόριθμο BP θα πρέπει να επιλέγεται με την λογική ότι το πληροφοριακό του περιεχόμενο είναι το μεγαλύτερο δυνατό για την τρέχουσα εργασία [8]. Για την επίτευξη αυτής της επιλογής εχουμε δυο τρόπους:

(α) να χρησιμοποιήσουμε ενα παράδειγμα το οποίο θα έχει σαν αποτέλεσμα το μεγαλύτερο σφάλμα εκπαίδευσης.

(β) να χρησιμοποιήουμε ένα παράδειγμα το οποίο θα είναι εντελώς διαφορετικό απο όσα χρησιμοποιήθηκαν προηγουμένως.

Με αυτές τις δυο ευρετικές οδηγίες ξεκινάει την επιθυμία αναζήτησης σε όσο το δυνατόν μεγαλύτερο μέρος του χώρου βαρών.

3. *Συνάρτηση ενεργοποίησης*

Όσον αφορά την ταχύτητα μάθησης , η προτιμώμενη επιλογή είνια η χρήση μιας σιγμοειδούς συνάρτησης ενεργοποίησης η οποία θα είναι περιττή συνάρτηση του ορίσματός της [8]

$$f(-u) = -f(u) \quad \textit{(Σχέση 3.8.1)}$$

Αυτή η συνθήκη ικανοποιείται απο την υπερβολική συνάρτηση

$$f(u) = a\, tannh(bu) \ \textit{(Σχέση 3.8.2)}$$

όπως παρουσιάζεται στα παρακάτω σχήματα αλλά όχι απο την λογιστική συνάρτηση. Κατάλληλες τιμές για τους περιορισμούς a και b στον τύπο της f(u) είναι οι ακόλουθες [LenCun] ,[8] .

$$a = 1.7159 \quad \textit{και} \quad b = \frac{2}{3}$$

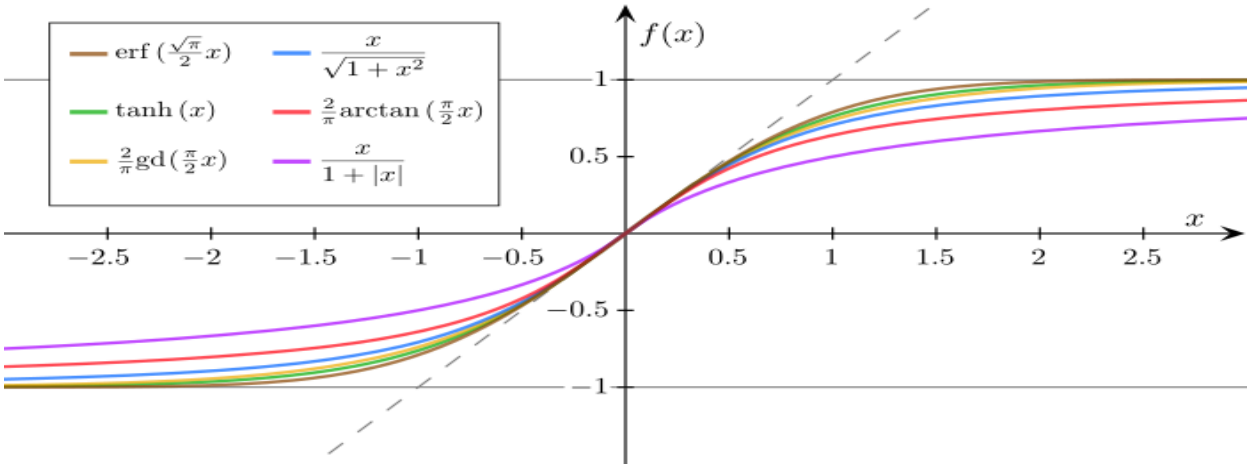
Η συνάρτηση υπερβολικής εφαπτόμενης f(u) έχει τις ακόλουθες χρήσιμες ιδιότητες [8]

$$f(1) = 1 \quad \textit{και} \quad f(-1) = -1$$

Στο σημείο της αρχής των αξόνων , η κλίση της συνάρτησης ενεργοποίησης είναι κοντά στην μονάδα , όπως υποδεικνύει η [8]

$$f(0) = a\, b = 1.7159 \left(\frac{2}{3}\right) = 1.1424 \ \textit{(Σχέση 3.8.3)}$$

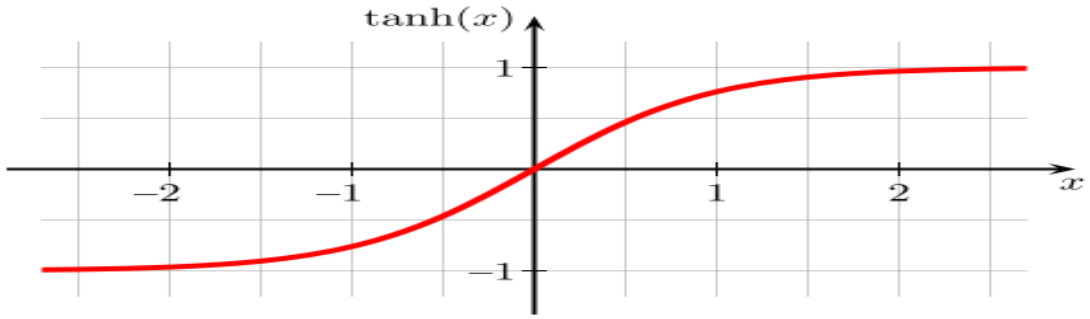
Η δεύτερη παράγωγος της f(u) αποκτά την μέγιστη τιμή της στο u = 1 [8].



Σχήμα 3.8.1

πηγή

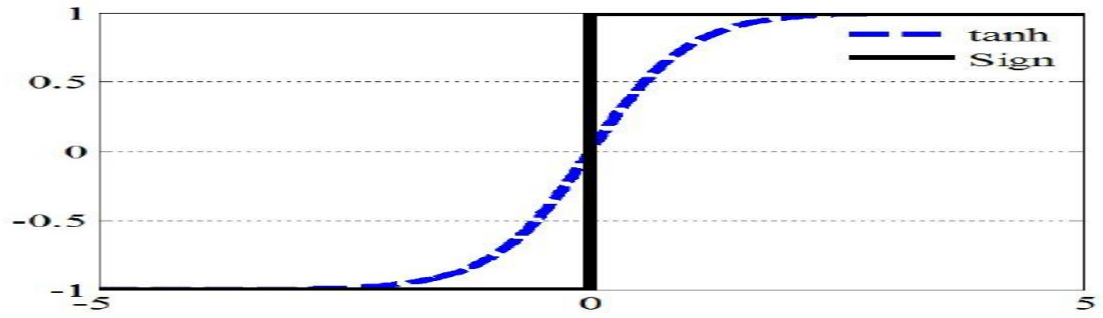
http://www.gallamine.com/2013/01/a-sigmoid-function-without-exponential_31.html



Σχήμα 3.8.2

πηγή

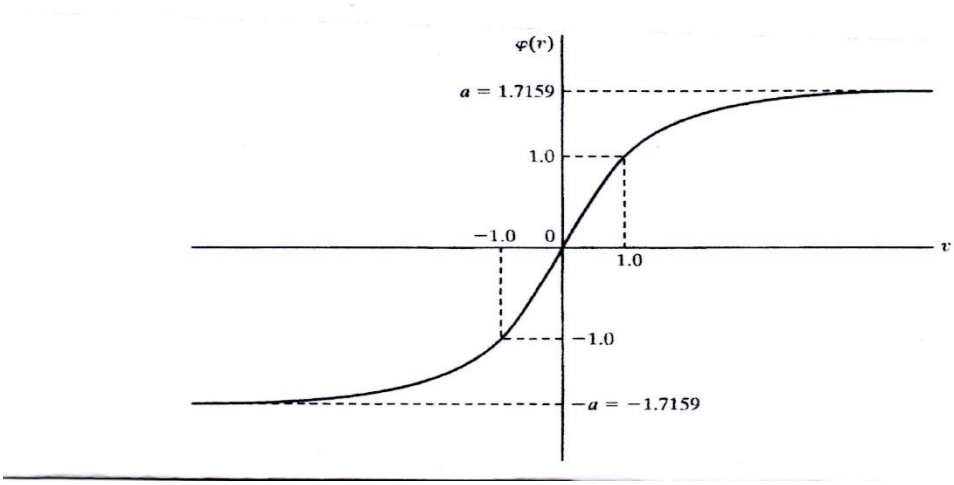
https://commons.wikimedia.org/wiki/File:Hyperbolic_Tangent.svg



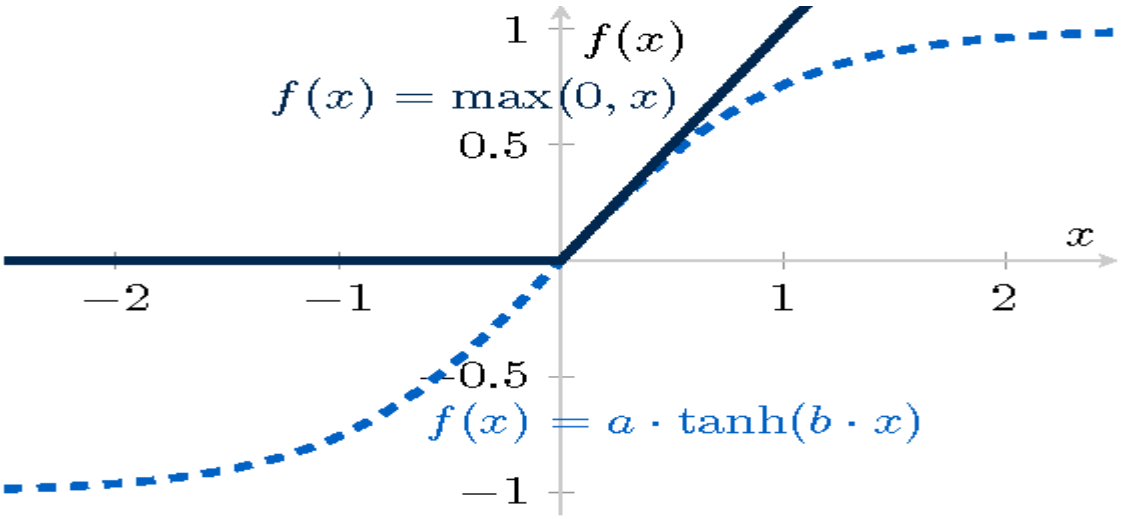
Σχήμα 3.8.3

πηγή

https://www.researchgate.net/figure/Graphical-representation-of-sign-versus-hyperbolic-tangent_fig3_282701377



Πηγή: *[8] Νευρωνικά Δίκτυα και Μηχανική Μάθηση, Simon Haykin ,Εκδόσεις Παπασωτηρίου*



x 5: This figure plots the activation functions hyperbolic tangent (3)

Σχήμα ,όπου a =1.7159 και b = 2/3. Οι συνιστώμενες τιμές στόχου είναι +1 και -1.

Σχήμα 3.8.4

4. *Τιμές στόχου*

Θα πρέπει να τονίσουμε ότι οι τιμές στόχου (επιθυμητές/η τιμές/απόκριση) να επιλέγονται μέσα στο εύρος τιμών της σιγμοειδής συνάρτησης ενεργοποίησης. Και πιο συγκεκριμένα , η επιθυμητή απόκριση dj για τον νευρώνα j στο επίπεδο εξόδου του perceptron πολλών επιπέδων θα πρέπει να μετατοπίζονται (offset) κατά κάποιο ποσό ε μακριά απο την περιοριστική τιμή της σιγμοειδούς συνάρτησης ενεργοποίησης, ανάλογα με το αν η περιοριστική τιμή είναι θετική ή αρνητική [8]. Αλλιώς , ο αλγόριθμος BP θα οδηγήσει τις ελεύθερες παραμέτρους του δικτύου στο άπειρο οπότε μετά θα επιβραδύνει την διαδικασία μάθησης, οδηγώντας τους κρυφούς νευρώνες στον κορεσμό [8] . Μπορούμε να δούμε τη συνάρτηση υπερβολικής εφαπτόμενης του παρακάτω σχήματος. Για την περιοριστική τιμή +α, θέτουμε:

$$dj = a - \varepsilon \quad \textit{(Σχέση 3.8.4)}$$

Και για την περιοριστική τιμή -α , θέτουμε:

$$dj = -a + \varepsilon \quad \textit{(Σχέση 3.8.5)}$$

Όπου ε είναι μια κατάλληλη θετική σταθερά. Για την επιλογή α = 1.7159 που χρησιμοποιείται στο παραπάνω σχήμα , μπορούμε να θέσουμε ε = 0.7159 περίπτωση στην οποία η τιμή στόχου (επιθυμητή απόκριση) dj μπορεί να επιλεχτεί στο ±1, όπως φαίνεται στο σχήμα [8].

5. *Κανονικοποίηση των εισόδων*

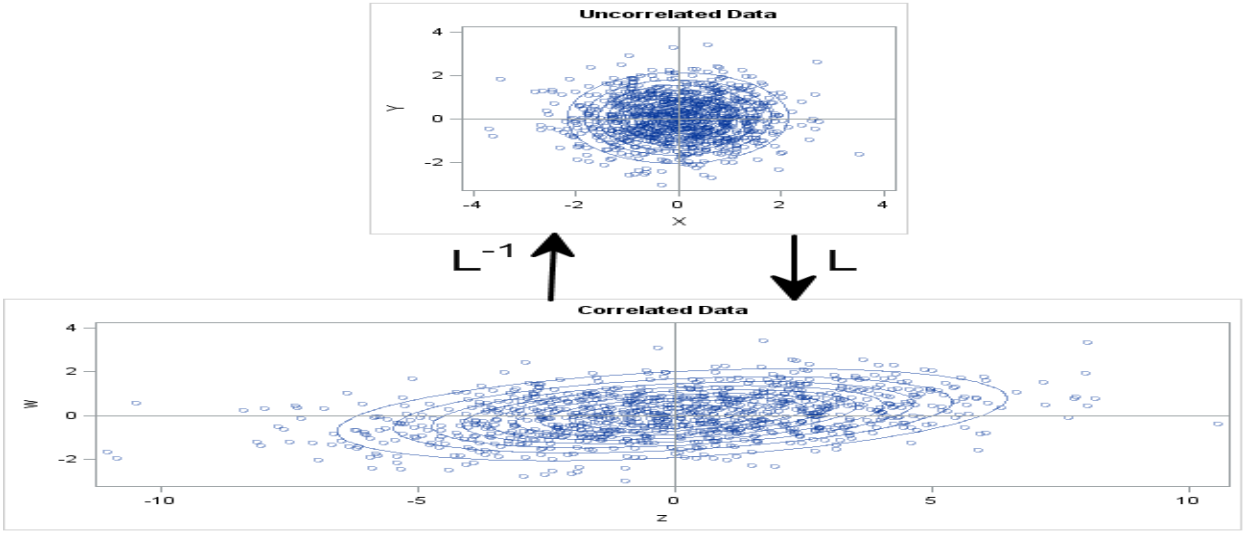
Οποιαδήποτε μεταβλητή εισόδου θα πρέπει να περνά απο την διαδικασία της προεπεξεργασίας έτσι ώστε η μέση τιμή τιμής της , υπολογισμένη για ολόκληρο το δείγμα εκπαίδευσης , να είναι κοντά στο μηδέν αλλιώς , θα είναι μικρή σε σχέση με την τυπική από-

κλιση [8], (Le Chun 1993). Ας εξετάσουμε μια ακραία περίπτωση όπου οι μεταβλητές εισόδου είναι συνεχώς θετικές [8]. Εδώ λοιπόν έχουμε , τα συναπτικά βάρη ενός νευρώνα στο πρώτο κρυφό επίπεδο μπορούν μόνο να αυξάνονται μαζί, ή να μειώνονται μαζί.

Οπότε , εάν το διάνυσμα βαρών αυτού του νευρώνα πάει να αλλάξει κατεύθυνση , θα μπορεί να το κάνει ακολουθώντας μια διαδρομή zig – zag πάνω στην επιφάνεια σφάλματος, που σημαίνει ότι συνήθως είναι αργό και θα πρέπει να αποφεύγεται [8]. Για να επιταχυνθεί η διαδικασία μάθησης με οπισθοδίαδοση σφάλματος, η κανονικοποίηση των εισόδων θα πρέπει επίσης να περιλαμβάνει δύο άλλα μέτρα [8], (Le Chun, 1993):

- *Οι μεταβλητές εισόδου που περιέχονται στο σύνολο εκπαίδευσης θα πρέπει να είναι αποσυσχετισμένες αυτό μπορεί να γίνει χρησιμοποιώντας ανάλυση κύριων συνιστωσών [8] .*
- *Οι αποσυσχετισμένες μεταβλητές εισόδου θα πρέπει να κλιμακώνονται έτσι ώστε οι συνδιακυμάνσεις τους να είναι περίπου ίσες, διασφαλίζοντας έτσι ότα τα διάφορα συναπτικά βάρη του δικτύου να μαθαίνουν με περίπου την ίδια ταχύτητα [8].*

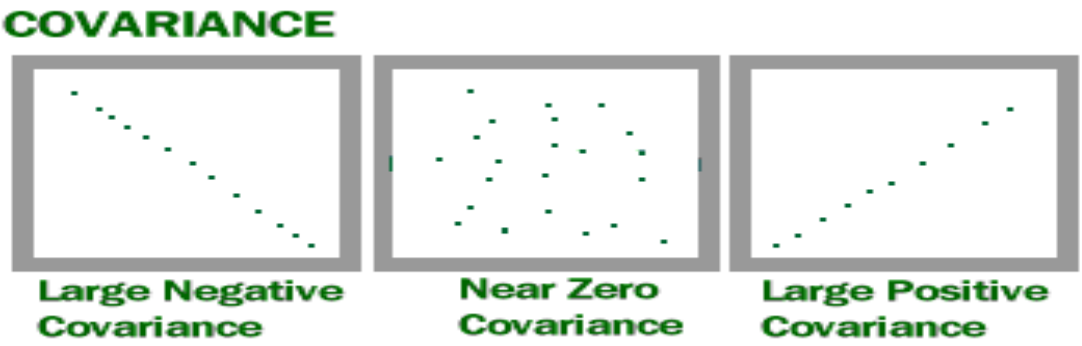
Στα σχήματα παρακάτω βλέπουμε τα αποτελέσματα των τριών βημάτων κανονικοποίησης: απομάκρυνση μέσων τιμών (mean removal), αποσυσχέτιση (decorrelation) και εξίσωση συνδιακυμάνσεων (covariance equalization), τα οποία εφαρμόζονται με αυτήν την σειρά [8]. Ακόμα Θα πρέπει να επισημάνουμε ότι όταν οι εισόδοι μετασχηματίζονται με τον τρόπο που απεικονίζονται στο σχήμα και χρησιμοποιούνται σε συνδυασμό με την συνάρτηση υπερβολικής εφαπτόμενης του σχήματος παραπάνω, η διακύμανση των μεμονωμένων εξόδων των νευρώνων στο perceptron πολλών επιπέδων θα είναι κοντά στην μονάδα [8] , (Ott και Muller, 1998). Τα πραγματικό κέρδος που δίνει η σιγμοειδής συνάρτηση για την ωφέλιμη περιοχή λειτουργίας της είναι περίπου μονάδα.



Σχήμα 3.8.5

πηγή

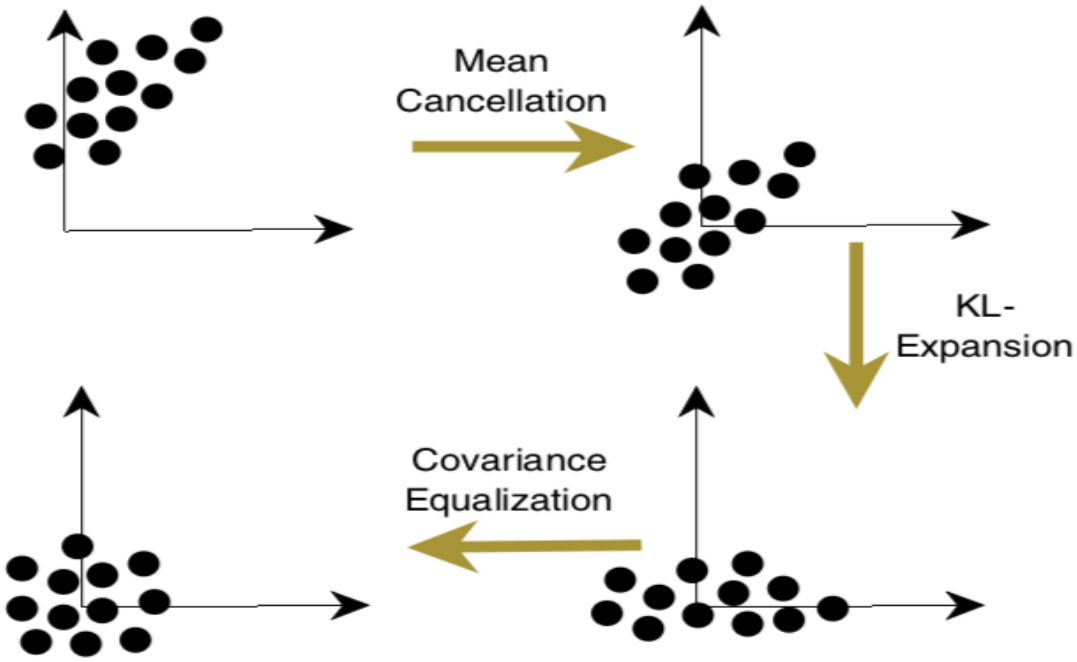
https://blogs.sas.com/content/iml/2012/02/08/use-the-cholesky-transformation-to-correlate-and-uncorrelate-variables.html



Σχήμα 3.8.6

πηγή

https://www.statisticshowto.datasciencecentral.com/covariance/



Σχήμα 3.8.7

πηγή

https://machinelearningmastery.com/best-advice-for-configuring-backpropagation-for-deep-learning-neural-networks/

6. *Αρχικοποίηση*

Επιλέγοντας κατάλληλα αρχικές τιμές συναπτικών βαρών και των κατωφλίων του δικτύου μπορούμε να πετύχουμε σωστή σχεδίαση ενός δικτύου. Πιο αναλυτικά, εάν στα συναπτικά βάρη βάλουμε μεγάλες αρχικές τιμές , είναι πιθανό οι νευρώνες του δικτύου να

οδηγηθούν σε κορεσμό [8]. Οπότε εαν συμβεί αυτό τότε οι τοπικές κλίσεις στον αλγόριθμο ΒΚ θα έχουν μικρές τιμές που με την σειρά τους, αναγκάζουν την διαδικασία μάθησης να επιβραδυνθεί [8]. Τώρα εάν στα συναπτικά βάρη μπούνε μικρές αρχικές τιμές , τότε ο αλγόριθμος ΒΚ μπορεί να λειτουργεί σε μια πολύ επίπεδη περιοχή γύρω απο το σημείο αρχής της επιφάνειας σφάλματος και πιο πολύ ισχύει στην περίπτωση των σιγμοειδών συναρτήσεων όπως η συνάρτηση υπερβολικής εφαπτόμενης [8]. Το μειονέκτημα είναι ότι το σημείο αρχής είναι ένα στατικό σημείο, όπου η καμπύλη της επιφά-νειας σφάλματος είναι αρνητική προς την μια κατεύθυνση και θετική προς την άλλη [8] .

Αρα λοιπόν η χρήση μεγάλων και μικρών τιμών για την αρχικοποίηση των συναπτικών βαρών θα πρέπει να αποφεύγεται. Η σωστή επιλογή των τιμών αρχικοποίησης βρίσκεται κάπου ανάμεσα σε αυτές τις δυο ακραίες περιπτώσεις [8] .

Αν εξετάσουμε την περίπτωση ενός perceptron πολλών επιπέδων που χρησιμοποιεί τη συνάρτηση υπερβολικής εφαπτομένης για τις συναρτήσεις ενεργοποίησης του [8] . Εαν η πόλωση (bias) που εφαρμόζεται σε κάθε νευρώνα του δικτύου είναι μηδέν. Τότε , μπορούμε να εκφράσουμε το τοπικό πεδίο του νευρώνα j ως [8] :

(Σχέση 3.8.6)

$$u_j = \sum_{i=1}^{m} w_{ji} \quad y_i$$

Αν υποθέσουμε ότι οι εισόδοι που εφαρμόζονται σε κάθε νευρώνα του δικτύου έχουν μηδενικό μέσο και μοναδιαία διακύμανση [8] .

(Σχέση 3.8.7)

$$\mu_y = E[y_i\ ] = \ 0 \quad \text{για όλα τα } i$$

και

(Σχέση 3.8.8)

$$\sigma_y^2 = E\ [((y_i) - (\mu_i))\text{^}2]\ = E[y_i^2\ ] = \ 1 \qquad \text{για όλα τα } i$$

Εστω ότι οι εισόδοι είναι αποσυσχετισμένες [8] :

$$E[y_i\ y_k] = \begin{cases} 1\ \gamma\iota\alpha\ k = i \\ 0\ \gamma\iota\alpha\ k\ \neq i \end{cases}$$

Και τα συναπτικά βάρη αντλούνται απο ένα ομοιόμορφο κατανεμημένο σύνολο αριθμών με μηδενικό μέσο – δηλαδή,

(Σχέση 3.8.9)

$$\mu_w = E\left[w_{ji}\ \right] = \ 0 \quad \text{για όλα τα ζεύγη } (j\ ,\ i)$$

και διακύμανση

(Σχέση 3.8.10)

$$\sigma_w^2 = E\ [((w_{ji}) - (\mu_w))\text{^}2]\ = E[w_{ji}^2\ ]$$ *για όλα τα ζεύγη (j, i)*

Οπότε μπορούμε να εκφράσουμε τη μέση τιμή και τη διακύμανση του τοπικού πεδίου u_j *ως [8] :*

(Σχέση 3.8.11)

$$\mu_v\ \ =\ \ E[\ u_j\ \ ] = E\left[\sum_{i=1}^{m} w_{ji}\ \ y_i\ \ \right] = \sum_{i=1}^{m} E[w_{ji}\ \ ]E[\ y_i\ \ ] = 0$$

και

(Σχέση 3.8.12)

$$\sigma_v^2 = E\ [((v_i) - (\mu_v))\text{^}2]\ = E[v_i^2\ \ ]$$

$$= E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji}\ \ y_i\ \ w_{jk}\ \ y_k\ \ \right]$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{m}\ E[w_{ji}\ \ w_{jk}\ ]E[\ y_i\ \ y_k\ ]$$

$$= \sum_{i=1}^{m} E[w_{ji}^2\ \ ]$$

$$=\ m\sigma_w^2$$

όπου m είναι ο αριθμός των συναπτικών συνδέσεων ενός νευρώνα [8] . Ετσι λοιπόν μπορούμε να περιγράψουμε μια στρατηγική για την αρχικοποίηση των βαρών με τέτοιον τρόπο ώστε η τυπική απόκλιση του τοπικού πεδίου ενός νευρώνα να βρίσκεται στο όριο μεταξύ των μερών γραμμικής και κορεσμένης λειτουργίας της σιγμοειδούς συνάρτησης ενεργοποίησής του [8]. Για την περίπτωση μιας συνάρτησης υπερβολικής εφαπτόμενης με τις παραμέτρους a και b του σχήματος παραπάνω , οστόχος αυτός ικανοποιείται θέ-

τοντας $\sigma_v = 1$ *στην παραπάνω εξίσωση, όπου παίρνουμε το ακόλουθο [8] ,(LeCun, 1993):*

(Σχέση 3.8.13)

$$\sigma_w = m^{-1/2}$$

Οπότε , είναι επιθυμητό η ομοιόμορφη κατανομή απο την οποία επιλέγονται τα συναπτικά βάρη να έχει μέση τιμή μηδέν και διακύμανση ίση με το μαιθηματικό αντίστροφο του αριθμού συναπτικών συνδέσεων ενός νευρώνα [8].

7. Μάθηση από υπαινικτικά στοιχεία

Η μάθηση που βασίζεται σε ενα δείγμα παραδειγμάτων εκπαίδευσης έχει να κάνει με μια αγνωστη συνάρτηση αντιστοίχησης εισόδου –εξόδου f(). Βασικά , η διαδικασία μάθησης αξιοποιεί την πληροφορία που περιέχεται στα παραδείγματα αυτα σχετικά με την συνάρτηση f() για να συμπεράνει μια προσεγγιστική υλοποίησή της [8]. Η διαδικασία της μάθησης μπορεί μέσα απο παραδείγματα να γενικευτεί έτσι ώστε να περιλαμβάνει επίσης και τη μάθηση απο υπαινικτικά στοιχεία (hints), η οποία επιτυγχάνεται επιτρέ-ποντας τη συμπερίληψη στην διαδικασία μάθησης οποιαδήποτε μέχρι τότε πληροφορίας που μπορεί να έχουμε σχετικά με την f() (Abu – Mostafa) [8]. Σε αυτές τις πληροφορίες μπορεί να περιλαμβάνονται αναλλοίωτες ιδιότητες, συμμετρίες ή οποιαδήποτε άλλη γνώση σχετική με την συνάρτηση f() , η οποία μπορεί να χρησιμοποιηθεί για την επιτάχυνση της αναζήτησης μιας προσεγγιστικής υλοποίησης της και πιο σημαντικό τη βελτίωση της ποιότητας της τελικής εκτίμησης. [8]

8. Ρυθμοί μάθησης

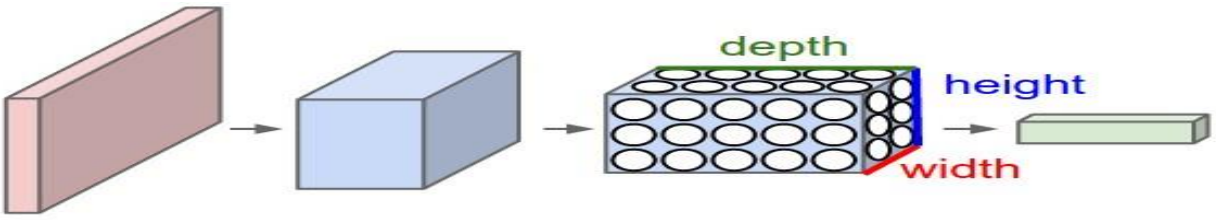
Στην ιδανική περίπτωση , όλοι οι νευρώνες στο perceptron πολλών επιπέδων θα πρέπει να μαθαινουν με τον ίδιο ρυθμό . Στα τελευταία επίπεδα έχουμε συνήθως μεγαλύτερες τοπικές κλίσεις απο ότι στα επίπεδα που βρίσκονται στην αρχή του δικτύου. [8] Η παρά-μετρος ρυθμού μάθησης n θα πρέπει να λαμβάνει μικρότερη τιμή στα τελευταία επίπεδα απο ότι στα αρχικά επίπεδα του perceptron. [8] Οι νευρώνες με πολλές εισόδους θα πρέπει να έχουν μικρότερη παράμετρο ρυθμού μάθησης απο τους νευρώνες με λίγες εισόδους, έτσι ώστε να διατηρείται ο ίδιος χρόνος μάθησης για όλους τους νευρώνες στο δίκτυο. [8] Στο LeCun (1993) , προτείνεται για ενα δεδομένο νευρώνα , ο ρυθμός μάθησης να είναι αντιστρόφως ανάλογος της τετραγωνικής ρίζας των συναπτικών συν-δέσεων που βρίσκονται στον νευρώνα. [8]

ΚΕΦΑΛΑΙΟ 4 *ΣΥΝΕΛΙΚΤΙΚΑ ΔΙΚΤΥΑ*

4.1 Συνελικτικά Δίκτυα (Convolutional Networks)

Στην ενότητα αυτήν θα δώσουμε βάρος στην δομικήν διάταξη του ίδιου του perceptron πολλών επιπέδων. Δηλαδή θα περιγράψουμε μια ειδική κατηγορία perceptron πολλών επιπέδων , τα οποία είναι γνωστά σαν συνελικτικά (Convolutional) δίκτυα ,τα οποία αποδεικνύονται ιδιαίτερα κατάλληλα για την ταξινόμηση προτύπων. [8] Το σκεπτικό στο οποίο βασίζεται η ανάπτυξη αυτών των δικτύων είναι δανεισμένο απο τα βιολογικά νευρωνικά δίκτυα πάνω στους τοπικούς ευαίσθητους και επιλεκτικούς ως προς τον προσανατολισμό νευρώνες του οπτικού φλοιού της γάτας. [8] Ενα συνελικτικό δίκτυο είναι ένα perceptron πολλών επιπέδων ειδικά σχεδιασμένο ώστε να αναγνωρίζει δισδιάστατα σχήματα με υψηλό βαθμό μη ευαισθησίας (ιδιότητα του αναλοίωτου) στη μετατόπιση, την κλιμάκωση , την στρέβλωση και άλλες μορφές παραμόρφωσης. [8] Τα συνελικτικά Νευρωνικά Δίκτυα χωρίζονται σε δυο μεγάλες κατηγορίες τα Αβαθή Νευρωνικά Δίκτυα και τα Βαθιά Νευρωνικά Δίκτυα. Ενα συνελικτικό επίπεδο αποτελείται απο νευρώνες οι οποίοι κάνουν συνέλιξη των φίλτρων που έχουν προκαθοριστεί με την εικόνα – διάνυσμα που δέχονται στην είσοδο.[10] Η διαφορά τους με τα Νευρωνικά Δίκτυα είναι οτι τα Συνελικτικά Δίκτυα υποθέτουνε ότι οι εισόδοι τους έχουνε την μορφή και χαρακτηριστικά εικόνων , όπως η τοπικότητα και η γειτνίαση.[9] Γι'αυτό η διάταξη και η οργάνωσή τους στο χώρο έχουν μεγάλη σημασία.[9] Χρησιμοποιούν ένα πλήθος επιπέδων που έχουν ως σκοπό την μείωση των διαστάσεων των χαρακτηριστικών, την κανονικοποίηση και άλλες σχετικές λειτουργίες. [9] Αρα τα συνελικτικά δίκτυα περιέχουν πολύ μικρότερο αριθμό παραμέτρων προς εκμάθηση σε σχέση με τα κλασσικά Νευρωνικά Δίκτυα και είναι εφοδιασμένα με επιπλέον ιδιότητες.[9] Τα κλασσικά ΝΔ δουλεύουν για συγκεκριμένο μέγεθος εικόνων εισόδου ,ενώ στα Συνελικτικά Δίκτυα που βασίζονται στις ιδιότητες της συνέλιξης, δεν περιορίζονται στο μέγεθος της εισόδου αφού η διαδικασία μπορεί να εφαρμοστεί επαναληπτικά σε κάθε περιοχή της εισόδου. [9]

Η παρακάτω εικόνα μας δείχνει ενα συνελικτικό δίκτυο και τον τρόπο με τον οποίο οι νευρώνες του οργανώνονται στον 3 – διάστατο χώρο. Οπότε κάθε στρώμα του συνελικτικού δικτύου, μετασχηματίζει έναν όγκο εισόδου σε έναν όγκο εξόδου, μειώνοντας με αυτόν τον τρόπο τις προς μάθηση παραμέτρους.[9]



πηγή

http://cs231n.github.io/convolutional-networks/

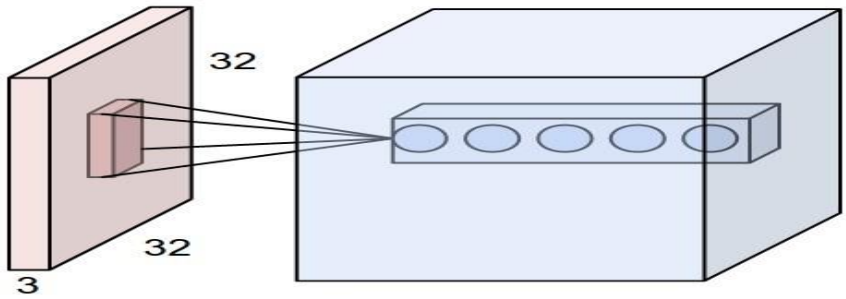
Εικόνα 4.1.1. Δομή των νευρώνων ενός Συνελικτικού Δικτύου στις τρεις διαστάσεις (πλάτος , ύψος, βάθος)

Το Συνελικτικό Επίπεδο (Convolutional Layer) *είναι ένα δομικό στοιχείο του Συνελικτικού Δικτύου. Οι παράμετροι του συνελικτικού επιπέδου αποτελούνται απο ένα πλήθος εκπαιδευόμενων φίλτρων. [9] Κάθε φίλτρο συνήθως είναι χωρικά μικρό (όσον αφορά τις διαστάσεις του, δηλαδή το ύψος του και το πλάτος του), αλλά επεκτείνεται μέχρι το συνολικό βάθος του όγκου εισόδου. [9] Κάθε φίλτρο μπορεί να περιέχει και μια πόλωση (bias).[9] Οταν έχουμε την εμπρός σάρωση (forward pass), κάθε φίλτρο σαρώνει την είσοδο κατά ύψος και κατά πλά-τος και μετά υπολογίζοντας το στοιχείο προς στοιχείο γινόμενο, μεταξύ του φίλτρου και των δεδομένων που βρίσκονται στο δεικτικό πεδίο του φίλτρου. [9] Οπως κινείται το φίλτρο πάνω απο το πλάτος και το ύψος του όγκου εισόδου θα παράγει έναν διδιάστατο (2) χάρτη χαρακτηριστικών (feature maps) της εισόδου που δίνει τις αποκρίσεις του εν λόγω φίλτρου σε κάθε χωρική θέση. [9] Οπότε το δίκτυο εκπαιδεύει τα φίλτρα ώστε να ενεργοποιούνται όταν βλέπουν κάποιο συγκεκριμένο τύπο χαρακτηριστικό σε κάποια χωρική θέση στην είσοδο. [9]*

Οπότε έχουμε ένα σύνολο φίλτρων σε κάθε στρώμα του Συνελικτικού επιπέδου , που το καθένα απο αυτά παράγει ένα ξεχωριστό χάρτη χαρακτηριστικών 2 διαστάσεων. Τώρα εάν φτιάξουμε σε μια στοίβα όλους αυτούς του χάρτες χαρακτηριστικών ως προς της διάσταση του βάθους θα δημιουργήσουμε τον όγκο εξόδου τους στρώματος.

Αξίζει να σημειωθεί ότι συνδέουμε κάθε νευρώνα σε μία μόνο μικρή, τοπική περιοχή του όγκου εισόδου (κατά πλάτος και κατά ύψος).[9] Η χωρική αυτή έκταση της σύνδεσης είναι μια υπέρ – παράμετρος που ονομάζεται το δεικτικό πεδίο του νευρώνα (receptive field) και ισοδύναμα αυτό είναι και το μέγεθος του φίλτρου. [9] Η έκταση της σύνδεσης αυτής κατά μήκος του άξονα βάθους είναι πάντα ίση με το βάθος του όγκου εισόδου.[9] Οπότε δημιουργείται μια χωρική τοπικότητα σύνδεση με την είσοδο (Local Connectivity). [9]

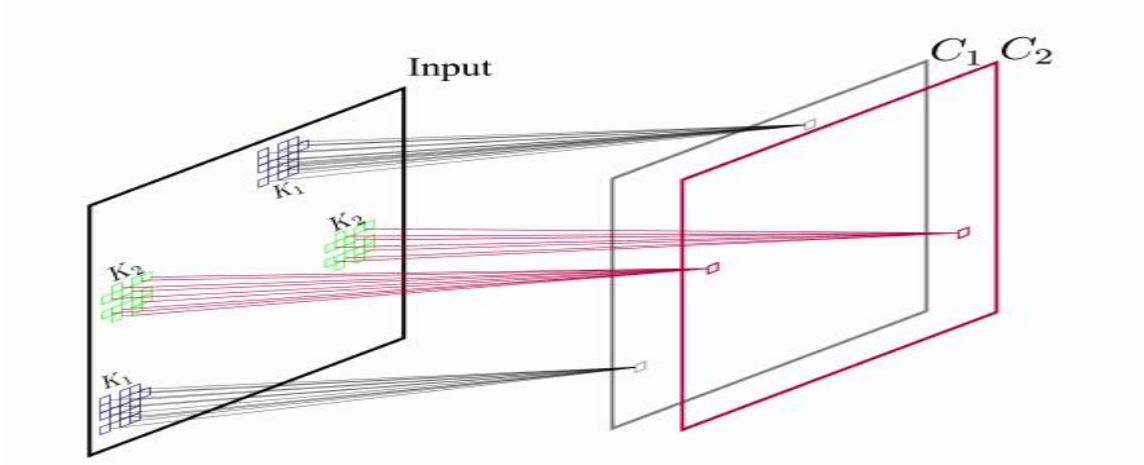
Για παράδειγμα έχουμε ενα όγκο εισόδου με μέγεθος διαστάσεων [32 χ 32 χ 3] ,(μια RGB CIFAR -10 εικόνα) και έχουμε ενα πλήθος όγκου νευρώνων στο πρώτο συνελικτικό στρώμα. Ο κάθε νευρώνας στο συνελικτικό στρώμα συνδέεται μόνο σε μια τοπική περιοχή του όγκου εισόδου στο κατακόρυφο επίπεδο αλλά συνδέεται κατά μήκος όλου του βάθους του (δηλαδή όλα τα κάναλια χρώματος).[9] Υπάρχουν πολλαπλοί νευρώνες (5 σε αυτό το παράδειγμα) κατά μήκος του βάθους και όλοι αναζητούν την ίδια περιοχή στην είσοδο.[9]



Εικόνα 4.1.2 Στο κόκκινο ένα παράδειγμα όγκου εισόδου και ακολουθεί ένα παράδειγμα όγκου νευρώνων του πρώτου συνελικτικού στρώματος

πηγή

http://cs231n.github.io/convolutional-networks/



Εικόνα 4.1.3

Αφού δείξαμε τον τρόπο με τον οποίο συνδέεται κάθε νευρώνας στον όγκο εισόδου, θα δείξουμε τώρα πως είναι τοποθετημένοι στον όγκο εξόδου και τους παράμετρους που καθορίζουν το μέγεθος του όγκου εξόδου και είναι οι:

- *Οσον αφορά για το βάθος (depth) του όγκου εξόδου γίνεται αντιστοίχιση με τον αριθμό των φίλτρων που θα θέλαμε εμείς να χρησιμοποιήσουμε οπότε το καθένα μαθαίνει να ψάχνει κάτι διαφορετικό*
- *Μετά θα πρέπει να καθορίσουμε το Stride με το οποίο θα κινείται/γλιστράει το φίλτρο. Το Stride καθορίζει κατά πόσες χωρικές μονάδες (π.χ. pixel) θα μετακινείται το δεικτικό πεδίο του φίλτρου οριζοντίως και καθέτως. [9] Οταν γίνεται μετακίνηση του φίλτρου κατά 1 pixel τη φορά θα μας οδηγήσει σε αλληλοεπικάλυψη των δεικτικών πεδίων μεταξύ των στηλών και παράγονται μεγάλοι όγκοι εξόδου. Αν όμως γίνει μεγαλύτερη μετακίνηση τότε θα έχουμε μικρότερη επικάληψη και ο όγκος εξόδου θα έχει μικρότερες διαστάσεις στο χώρο.*
- *Μετά έχουμε και την παράμετρο του Zero padding. Μας βολεύει να βάλουμε μηδενικά γύρω απο το περίγραμμα του όγκου εισόδου. Οπότε με το Zero padding καταφέρνουμε να διατηρήσουμε το πραγματικό χωρικό μέγεθος του όγκου εισόδου.*

*Και αφού ορίσαμε τις παραμέτρους ενός συνελικτικού στρώματος μετά μπορούμε να υπολογίσουμε τις χωρικές διαστάσεις του όγκου εξόδου. Με το **O** συμβολίζουμε τον όγκο εξόδου. Με το **W** συμβολίζουμε τις διαστάσεις τις εισόδου. Οι διαστάσιες των συνελικτικών πυρήνων συμβολίζονται με το **F** ,το Stride με το **S** ,το μέγεθος του Zero Padding με το **P** . Μια διευκρίνηση ,δεν είναι απαραίτητο ο όγκος εισόδου να έχει το ίδιο ύψος με πλάτος ,το ίδιο μπορούμε να πούμε και για τα φίλτρα, δηλαδή δεν είναι απαραί-τητο να είναι τετράγωνα. Παρακάτω έχουμε την σχέση , με τους δείκτες w, h να δηλώνουνε αν η διάσταση που υπολογίζουμε είναι πλάτος ή το ύψος.*

$$O_{h,w} = \frac{W_{h,w} - F_{h,w} + 2P_{h,w}}{S_{h,w}} + 1$$

(Σχέση 4.1.1)

Θα πρέπει να διευκρινήσουμε ότι θέλουμε ακέραιο αριθμό σαν αποτέλεσμα αλλιώς δεν ταιρίαζουν σωστά οι νευρώνες και συμμετρικά κατά μήκος της εισόδου. Οπότε μετά δεν μας κάνει.

Ενα παράδειγμα για να καταλάβουμε την χρήση του παραπάνω τύπου απο ενα παράδειγμα του πραγματικού κόσμου ,αρχιτεκτονική Krizhevsky που δέχεται εικόνες μεγέ-θους [227 χ 227 χ 3] . Στο πρώτο layer της συνέλιξης χρησιμοποιήθηκαν νευρώνες με δεικτικό πεδίο μεγέθους F =11 και Stride S =4 και καθόλου zero padding P =0 . Οπότε έχουμε:

$$O_{h,w} = \frac{W_{h,w} - F_{h,w} + 2P_{h,w}}{S_{h,w}} + 1 =$$

$$O_{h,w} = \frac{227_{h,w} - 11_{h,w} + 0_{h,w}}{4} + 1 = 55$$

Και αφού το συνελικτικό στρώμα είχε βάθος K = 96 , ο όγκος της εξόδου του συνελι-κτικού είχε μέγεθος: [55 χ 55 χ 96]. Κάθε απο τους νευρώνες 55 χ 55 χ 3 στον παρα-πάνω όγκο , συνδέθηκε σε μια περιοχή μεγέθους: [11 χ 11 χ 3] του όγκου εισόδου.[9]

Η συνέλιξη σήματος δίνεται απο τον παρακάτω τύπο:

$$z_j = \sum_i w_{j,i} * a_i$$

(Σχέση 4.1.2)

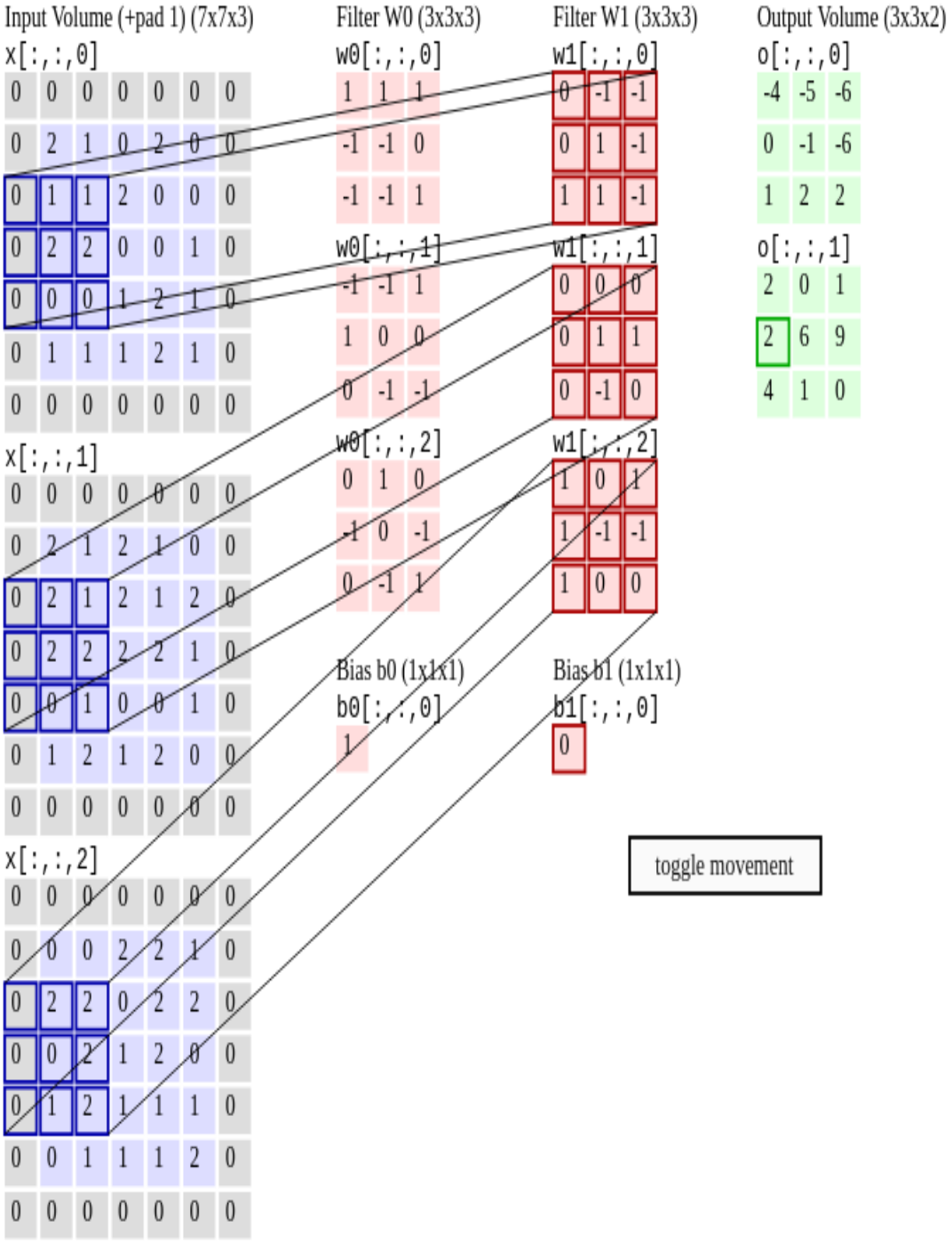
Αν τώρα η είσοδος a_i και τα φίλτρα του στρώματος $w_{j,i}$ είναι δυσδιάστατα μεγέθη αντί για μονοδιάστατα τότε η παραπάνω εξίσωση για το εμπρός πέρασμα γράφεται ως [9]:

$$z_j(p,q) = \sum_{i,u,v} w_{j,i}(u,v) * a_i(p-u, q-u) = \sum_{i,u,v} w_{j,i}(p-u, q-u) * a_i(u,v)$$

(Σχέση 4.1.3)

*Οπου το i παίρνει τιμές απο το 1 εως το k , όπου k είναι ο μέγιστος αριθμός των φίλ-τρων του στρώματος.[9] Τα u και v παίρνουν τιμές απο το 1 εως το **W** και **H** ,όπου **W** και **H** είναι οι διαστάσεις της κατά μήκος και κατά πλάτος της εισόδου α.[9]*

Στο παρακάτω Σχήμα παρουσιάζεται η συνέλιξη δισδιάστατων σημάτων για 2 φίλτρα W1 και W2, τριών χρωματικών καναλιών , στα οποία περιλαμβάνονται και πολώσεις



Εικόνα 4.1.4 Αποτέλσμα συνέλιξης μιας εικόνας RGB με δυο δυσδιάστατα φίλτρα W1 και W2

πηγή

https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

Για να συνοψίσουμε όσον αφορά το συνελικτικό επίπεδο:

a) *δέχεται ως είσοδο έναν όγκο με διαστάσεις:* $\boldsymbol{W_1\ x\ H_1\ x\ D_1}$ *[9]*
b) *Απαιτούνται 4 υπέρ – παράμετροι [9]:*
   - *Το πλήθος των φίλτρων/πυρήνων* ***K***
   - *Το χωρικό μέγεθος των πυρήνων σε κάθε διάσταση* ***F***
   - *Το βήμα μετατόπισης του δεικτικού πεδίου των φίλτρων (stride)* ***S***
   - *Το πόσο του zero padding κατά ύψος και πλάτος του όγκου εισόδου* ***P***
c) *Τα βάρη του στρώματος οργανώνονται σε μια διάταξη μεγέθους [9]:* $\boldsymbol{K\ x\ C\ x\ F\ x\ F}$ *, με* ***C*** *να είναι ο αριθμός των χρωματικών καναλιών των φίλτρων. Επίσης σε κάθε ένα απο τα* ***K*** *φίλτρα αντιστοιχεί και μια πόλωση* ***b.*** *[9]*
d) *Στην έξοδο του στρώματος, παράγεται ένας όγκος με διαστάσεις:* $\boldsymbol{W_2\ x\ H_2\ x\ D_2}$ *όπου:*

   - $W_2 = \frac{W_1 - F\ +2P}{S} + 1$
   - $H_2 = \frac{H_1 - F\ +2P}{S} + 1$
   - $D_2 = K$

e) *Μέσω τις ιδιότητας διαμοιρασμού των παραμέτρων, κάθε φίλτρο αποτελείται απο* $\boldsymbol{F * F * D_1}$ *βάρη και συνολικά οι προς μάθηση παράμετροι είναι [9]:* $\boldsymbol{(F * F * D_1) * K}$ *βάρη και* $\boldsymbol{K}$ *πολώσεις.[9]*
f) *Στον όγκο εξόδου, μια d-οστή βαθμίδα (μεγέθους* $\boldsymbol{W_2\ x\ H_2}$ *) αποτελεί τον χάρτη χαρακτηριστικών (**feature map**) , ο οποίος είναι αποτέλεσμα εκτέλεσης μιας έγκυρης συνέλιξης του d-οστού φίλτρου στον όγκο εισόδου.[9] Μετά προστίθεται η d-οστή πόλωση.*
g) *Κάθε στήλη του όγκου εξόδου, κατά την διάσταση του βάθους περιέχει το αποτέλεσμα της συνέλιξης μιας τοπικής χωρικής υπο – περιοχής, για κάθε ένα απο τα φίλτρα.[9]*

Επίσης θα πρέπει να αναφέρουμε ότι τα συνελικτικά φίλτρα μπορεί να μην είναι τετράγωνα και μπορεί να έχουν την εξής διάσταση ***F1 X F2****. Μπορεί μια διάσταση να είναι μοναδιαία. Οπότε μπορεί να συμβεί συνέλιξη μονοδιάστατου σήματος με μονοδιάστατα φίλτρα. Τα συνελικτικά δίκτυα χρησιμοποιούνται κατά βάση στην επεξεργασία εικόνων. [9]*

Το συνελικτικό Δίκτυο Διδάσκεται με επιβλεπόμενο τρόπο, μέσω ενός δικτύου του οποίου η δομή περιλαμβάνει τις ακόλουθες μορφές περιορισμών (LeCun and Bengio 2003):. [8]

a. *Εξαγωγή χαρακτηριστικών: Κάθε νευρώνας παίρνει τις συναπτικές εισόδους του απο ένα τοπικό δεικτικό πεδίο του προηγούμενου επιπέδου , οπότε το αναγκάζει να εξάγει τοπικά χαρακτηριστικά.[8] Μόλις εξαχθεί ενα χαρακτηριστικό , η ακριβής θέση του γίνεται λιγότερο σημαντική με την προυπόθεση να διατηρείται η σχετική του θέ-ση ως προς άλλα χαρακτηριστικά.*
b. *Αντιστοίχιση χαρακτηριστικών: Κάθε υπολογιστικό επίπεδο του δικτύου απαρτίζεται απο πολλαπλούς χάρτες χαρακτηριστικών (feature maps) , με κάθε χάρτη χαρακτηριστικών να είναι στην μορφή ενός επιπέδου μέσα στο οποίο οι μεμονωμένοι νευρώ-*

νες ελέγχονται (constrained) ώστε να μοιράζονται το ίδιο σύνολο συναπτικών βα-ρών. [8] Αυτή η δεύτερη μορφή δομικού περιορισμού έχει τα ακόλουθα ευεργετικά επακόλουθα:

- *Μη – ευαισθησία (αναλλοίωτο) ως προς την μετατόπιση (shift invariance),η οποία επιβάλλεται στη λειτουργία ενός χάρτη χαρακτηριστικών μέσω της χρήσης μιας συνέλιξης με εναν πυρήνα (kernel) μικρού μεγέθους, η οποία παρακολουθείται απο την εφαρμογή μιας σιγμοειδούς συνάρτησης.[8]*
- *Μείωση του αριθμού των ελεύθερων παραμέτρων, η οποία επιτυγχάνεται μέσω διαμοιρασμού βαρών.[8]*

c. *Υποδειγματοληψία: Κάθε συνελικτικό επίπεδο ακολουθείται απο ένα υπολογιστικό επίπεδο το οποίο εκτελεί τοπικό υπολογισμό μέσων όρων και υποδειγματοληψία ,των οποίων η ανάλυση του χάρτη χαρακτηριστικών μειώνεται. [8] Και αυτό έχει σαν αποτέλεσμα την μείωση της ευαισθησίας της εξόδου του χάρτη χαρακτηριστικών στις μετατοπίσεις και άλλες μορφές παραμόρφωσης.*

Οσον αφορά την υποδειγματοληψία εχουμε 4 τύπους υποδειγματοληψίας (pooling):

- *Στρώμα χωρικής Υποδειγματοληψίας (Pooling layer)*
- *Υποδειγματοληψία εξαγωγής τοπικού μέγιστου (Max pooling)*
- *Στοχαστική Υποδειγματοληψία (Stochastic pooling)*
- *Υποδειγματοληψία μέσου όρου (Average pooling)*

Οσον αφορά το στρώμα χωρικής Υποδειγματοληψίας ,αυτό εισάγεται μεταξύ δυο διαδοχικών συνελικτικών στρωμάτων. Σκοπός του είναι να μειώσει σιγά σιγά το χωρικό μέγεθος του όγκου εισόδου και τελικά να μειώσει τον αριθμό των απαιτούμενων παραμέτρων προς υπολογισμό του επόμενου στρώματος. Επίσης βοηθάει στο να αποφύγουμε την υπέρ – εκπαίδευση.

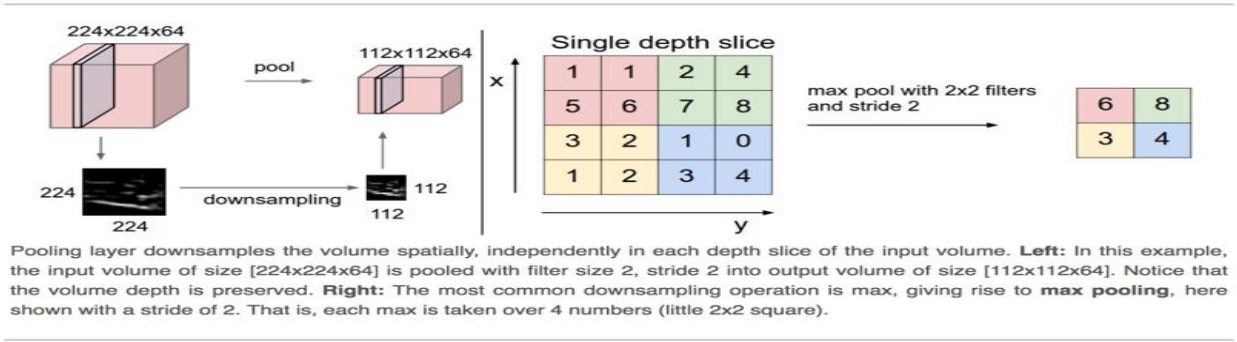
Η πιο συνηθισμένη μορφή ενός στρώματος χωρικής υποδειγματοληψίας περιλαμβάνει φίλτρα μεγέθους 2 χ 2 με stride 2.[9] Η μορφή αυτή μειώνει στο μισό, τις χωρικές διαστάσεις ύψους και πλάτους του όγκου εισόδου,απορρίπτοντας το 75 % των ενεργοποιήσεων.[9]

Εάν υποθέσουμε ότι ο όγκος εισόδου του στρώματος χωρικής υποδειγματοληψίας περιγράφεται ως:

- $\boldsymbol{W_1\ x\ H_1\ x\ D_1}$
- *Το στρώμα αυτό προυποθέτει την δήλωση δυο υπέρ – παραμέτρων:*
- *Το μέγεθος του πυρήνα κάθε φίλτρου **F***
- *Το Stride **S***
- *Τελικά παράγεται ένας όγκος μεγέθους $\boldsymbol{W_2\ x\ H_2\ x\ D_2}$,όπου:*
  - $W_2 = \frac{W_1 - F}{S} + 1$
  - $H_2 = \frac{H_1 - F}{S} + 1$
  - $D_2 = D_1$
- *Δεν εισάγονται παράμετροι προς υπολογισμό αφού υλοποιείται απλά μια συνάρτηση μετατροπής της εισόδου.[9]*
- *Δεν χρησιμοποιούμε zero padding*

Οι δυο πιο κοινές παραλλαγές του max pooling στρώματος στην πράξη είναι [9]:

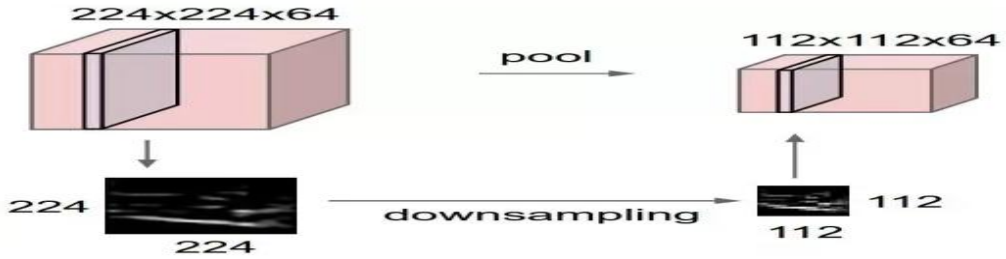
pooling στρώμα με ***F = 3 , S =2*** *(overlapping pooling) και πιο συχνά* ***F = 2 , S =2.*** *Οταν το δεικτικό πεδίο των φίλτρων είναι πολύ μεγάλο τότε εχουμε καταστροφή πληροφορίας [9]. Εάν ενα pooling στρώμα με* ***F = 2 , S =2*** *και η χωρική επίδραση που έχει σε έναν όγκο εισόδου [224 χ 224 χ 64] φαίνεται στο παρακάτω σχήμα.*



Εικόνα 4.1.5 Επίδραση χωρικής υπο-δειγματοληψίας σε έναν όγκο δεδομένων

πηγή

https://github.com/mratsim/Arraymancer/issues/174



πηγή

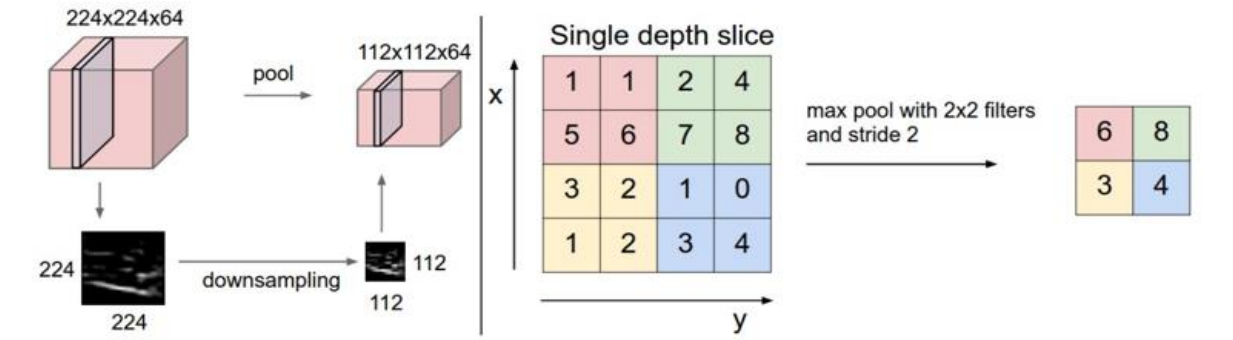
http://cs231n.github.io/convolutional-networks/

Εικόνα 4.1.6 Επίδραση χωρικής υπο-δειγματοληψίας σε έναν όγκο δεδομένων

Εικόνα 4.1.7 Λειτουργία χωρικής υποδειγματοληψίας με την χρήση του τελεστή max για stride = 2 και μέγεθος εφαρμογής 2 χ 2, προκαλεί μέγιστη συγκέντρωση

πηγή

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Εικόνα 4.1.8 χωρική υποδειγματοληψία

πηγή

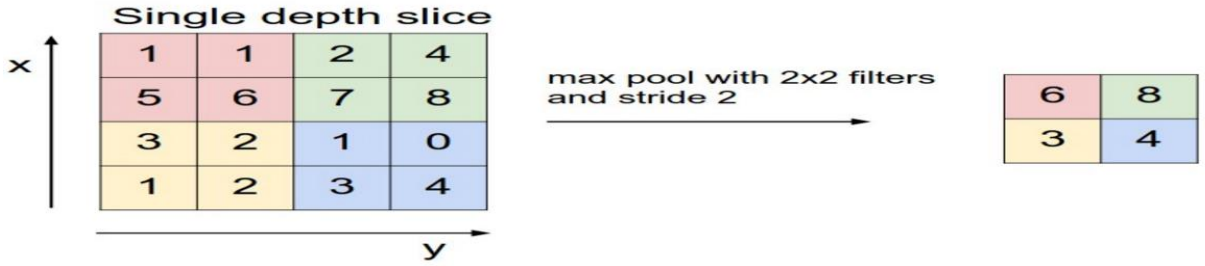
https://github.com/mratsim/Arraymancer/issues/174

*Πάμε τώρα στην Υποδειγματοληψία εξαγωγής τοπικού μεγίστου θεωρείται σαν μια μη – γραμμική συνάρτηση που χρησιμοποιεί τον **τελεστή μεγίστου** **max – pooling**. Η εικόνα εισόδου χωρίζεται σε ένα σύνολο μη – επικαλυπτόμενων ορθογωνίων και για τέτοια υπό – περιοχή , εξάγεται το μέγιστο.[9] Εάν βρεθεί ένα χαρακτηριστικό , η ακριβή θέση του δεν μας ενδιαφέρει τόσο όσο η σχετική θέση του σε σχέση με άλλα χαρακτηριστικά. Εάν ας πούμε ότι μια d – οστή βαθμίδα του όγκου εισόδου στο max pooling στρώμα, τότε η συνάρτηση που περιγράφει την προς τα εμπρός διάδοση δίνεται απο την σχέση[9]:*

$$y(i,j) = max_{m,n} \in \quad [o,p] \ \ (x(i*s+m,\ j*s+n))$$

(Σχέση 4.1.4)

Η απο πάνω εξίσωση παρουσιάζεται γραφικά απο την παρακάτω εικόνα , για stride S=2, και δεικτικό πεδίο p=2



Εικόνα 4.1.9 Λειτουργία χωρικής υποδειγματοληψίας με την χρήση του τελεστή max για stride = 2 και μέγεθος εφαρμογής 2 χ 2, προκαλεί μέγιστη συγκέντρωση

πηγή

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Το τοπικό μέγιστο είναι η καλύτερη περίπτωση pooling . Αλλά τα μοντέλα που έχουν τέτοια στρώματα δεν λαμβάνουν υπόψιν τα χαρακτηριστικά που έχουν μεγαλύτερη ισχύ. Οπότε εν τέλει γίνεται απόρριψη των pooling στρωμάτων και καταλήγουν σε αρχιτεκτονικές που αποτελούνται μόνο απο συνελικτικά στρώματα.

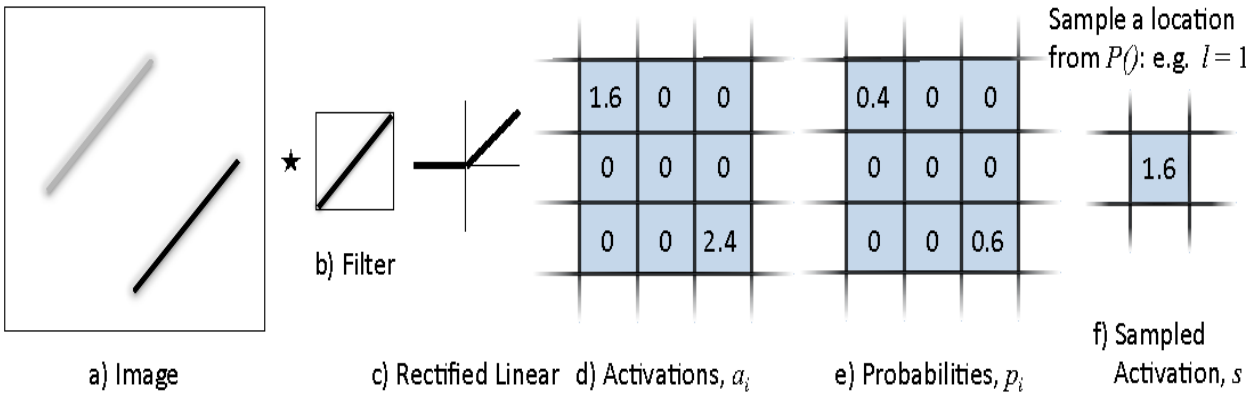
Επόμενη διαδικασία Υπο – δειγματοληψίας είναι αυτή της ***στοχαστικής υποδειγματοληψίας (Stochastic pooling)*** *,μπορούμε να πούμε ότι πολλές φορές χρησιμοποιείται για κανονικοποίηση (regularization) του δικτύου. Ουσιαστικά δημιουργεί μια πιθανοτική κατανομή* ***P*** *, οπότε με βάση τις τιμές των ενεργοποιήσεων που περικλείονται σε κάθε j περιοχή , μεγέθους* ***F*** *στον συνολικό χάρτη ενεργοποίησης ενός χαρακτηριστικού [9].*

$$P_i = \frac{\alpha_i}{\sum k \ \in R_J \alpha_K}$$

(Σχέση 4.1.5)

Θα πρέπει να αναφερθεί ότι οι τιμές των α_i πρέπει να είναι θετικές μιας και μιλάμε για πιθανοτική κατανομή.

Ως έξοδος μιας περιοχής θα επιλεχθεί η θέση που προκύπτει απο την πολυωνυμική κατανομή $P(p_1 \ldots, p_{R_j})$ με πιθανότητες αυτές που υπολογίστηκαν απο την κανονικοποίηση των χαρακτηριστικών [9]. Παρακάτω φαίνεται αυτη η διαδικασία.



Εικόνα 4.1.10 Αποτέλεσμα εφαρμογής στοχαστικής υποδειγματοληψίας

πηγή

https://www.semanticscholar.org/paper/Stochastic-Pooling-for-Regularization-of-Deep-Zeiler-Fergus/0abb49fe138e8fb7332c26b148a48d0db39724fc

Ας πούμε ότι η παραπάνω εικόνα έχει δυο ακμές που συνελίσσονται με έναν φίλτρο οπότε σαν αποτέλεσμα απο την εφαρμογή της Rectified Linear προκύπτει ο χάρτης χαρακτηριστικών (d) [9]. Μετά σε μια περιοχή μεγέθους ***F =3*** *του συνολικού χάρτη χαρακτηριστικών, οι τιμές κανονικοποιούνται με βάση τον τύπο που αναφέρθηκε νωρίτερα. Σαν αποτέλεσμα αυτής της διαδικασίας είναι το χαρακτηριστικό του οποίου η θέση προ-κύπτει απο την πολυωνυμική κατανομή $P(p_1 \ldots, p_9)$.*

Στο απο πάνω παράδειγμα επιλέχθηκε η θέση l = 1 της 3 x 3 περιοχής χαρακτηριστικών [9]. Κατά την οπισθοδρόμηση μέσω του δικτύου, χρησιμοποιείται αυτή η επιλεγμένη θέση l για να κατευθύνει την κλίση προς τα πίσω μέσω του επιπέδου υποδειγματοληψίας [9].

Υποδειγματοληψία μέσου όρου (average pooling) *,η συνάρτηση χωρικής Υπο – δειγματοληψίας δίνεται απο την παρακάτω σχέση:*

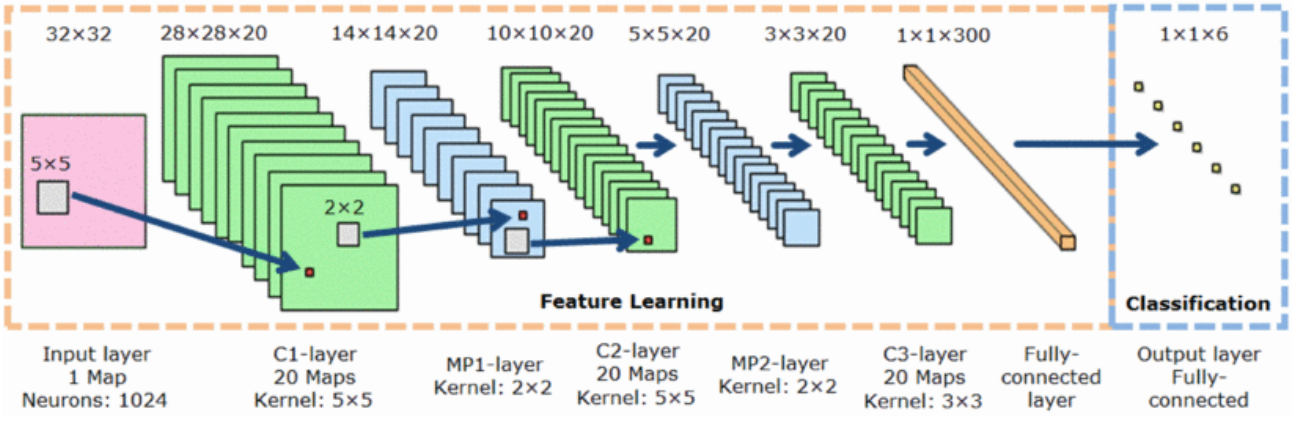
$$y(i,j) = average_{m,n} \in \quad [o,p] \ (x(i*s+m,\ j*s+n))$$

(Σχέση 4.1.6)

Έχουμε και εδώ μια d – οστή βαθμίδα του όγκου εισόδου. Αυτή η συνάρτηση average λέγεται και μέσου όρου, και εφαρμόζεται σε μια περιοχή μεγέθους ***F*** *του συνολικού χάρτη χαρακτηριστικών [9]. Αυτή η μέθοδος έχει και μειονεκτήματα και δεν πολυχρησιμοποιείται στην πράξη.*

Μην ξεχάσουμε να αναφέρουμε ότι όλα τα βάρη σε όλα τα επίπεδα ενός συνελικτικού δικτύου μαθαίνονται μέσω εκπαίδευσης. Το δίκτυο μαθαίνει να εξάγει τα δικά του χαρακτηριστικά απο μόνο του.

Το παρακάτω σχήμα μας δείχνει την αρχιτεκτονική διάταξη ενός συνελικτικού δικτύου το οποίο αποτελείται απο το επίπεδο εισόδου, τέσσερα κρυφά επίπεδα και το επίπεδο εξόδου. Είναι σχεδιασμένο έτσι ώστε να εκτελεί επεξεργασία εικόνας (Αναγνώριση χειρόγραφων χαρακτήρων).



Εικόνα 4.1.11

πηγή

http://pubs.sciepub.com/ajme/2/7/9/

Το επίπεδο εισόδου αποτελείται απο 28 χ 28 αισθητηριακούς κόμβους , λαμβάνει τις εικόνες διαφορετικών χαρακτήρων οι οποίοι έχουν περίπου κεντραριστεί και υποστεί κανονικοποίηση μεγέθους. [8]

Απο αυτό το σημείο και μετά τα υπολογιστικά επίπεδα του δικτύου εναλλάσσονται μεταξύ συνέλιξης και υποδειγματοληψίας:

1. *Το πρώτο κρυφό επίπεδο εκτελεί συνέλιξη και αποτελείται απο τέσσερις χάρτες χαρακτηριστικών (feature map) , με κάθε χάρτη να αποτελείται απο 24 χ 24 νευρώνες. Ο Κάθε νευρώνας έχει δεικτικό πεδίο μεγέθους 5 χ 5.*
2. *Το αμέσως επόμενο κρυφό επίπεδο κάνει υποδειγματοληψία και τοπικό υπολογισμό μέσω όρων: αποτελείται απο τέσσερις χάρτες χαρακτηριστικών, που ο καθένας απο αυτούς αποτελείται απο 12 χ 12 νευρώνες. Ο κάθε νευρώνας έχει δεικτικό πεδίο μεγέθους 2 χ 2 , εκπαιδεύσιμο συντελεστή, εκπαιδεύσιμη πόλωση και μια σιγμοειδή συνάρτηση ενεργοποίησης . [8] Ο εκπαιδεύσιμος συντελεστής και η πόλωση ελέγχουν το σημείο λειτουργίας του νευρώνα για παράδειγμα ,εάν ο συντελεστής είναι μικρός ,ο νευρώνας λειτουργεί σε ψευδογραμμική κατάσταση. [8]*
3. *Το τρίτο κρυφό επίπεδο εκτελεί μια δεύτερη συνέλιξη. Αποτελείται απο 12 χάρτες χαρακτηριστικών, με κάθε χάρτη χαρακτηριστικών να απαρτίζεται απο 8 χ 8 νευρώνες. Κάθε νευρώνας σε αυτό το κρυφό επίπεδο μπορεί να έχει συναπτικές συνδέσεις απο αρκετούς χάρτες χαρακτηριστικών του προηγούμενου κρυφού επιπέδου. [8] Λειτουργεί σαν το πρώτο συνελικτικό επίπεδο.*
4. *Στο τέταρτο κρυφό επίπεδο εχουμε μια δεύτερη υποδειγματοληψία και τοπικό υπολογισμό μέσων όρων. Και αποτελείται απο 12 χάρτες χαρακτηριστικών που καθένας απο αυτούς αποτελείται απο 4 χ 4 νευρώνες.*
5. *Και φθάνουμε στο επίπεδο εξόδου που γίνεται και εδώ η τελική συνέλιξη. Αποτελείται απο 26 νευρώνες με κάθε νευρώνα να αντιστοιχίζεται σε έναν απο τους 26 πιθανούς χαρακτήρες.[8] Κάθε νευρώνας έχει δεικτικό πεδίο μεγέθους 4 χ 4 .[8]*

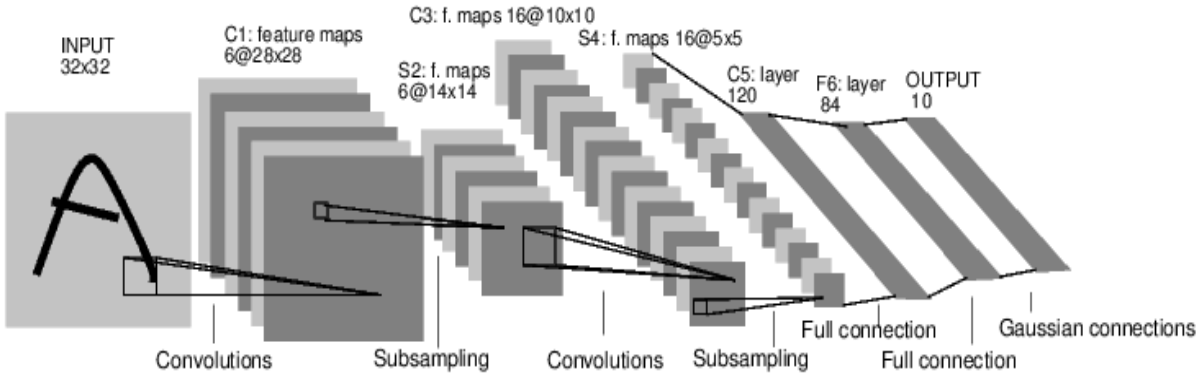
Με τα διαδοχικά υπολογιστικά επίπεδα του δικτύου να εναλλάσονται μεταξύ συνέλιξης και υποδειγματοληψίας, καταλήγουμε στο εξής αποτέλεσμα ότι σε κάθε συνελικτικό επίπεδο ή επίπεδο υποδειγματοληψίας να έχουμε αύξηση του αριθμου των χαρτών χαρακτηριστικών, ενώ η χωρική ανάλυση να μειώνεται σε σχέση με το προηγούμενο επίπεδο.[8]

Το perceptron πολλών επιπέδων που περιγράφηκε προηγουμένως περιέχει κατα προσέγγιση 100.000 συναπτικές συνδέσεις, αλλά εχει περίπου 2.600 ελεύθερες παράμετρους. Η μειώση στον αριθμό των ελεύθερων παραμέτρων πετυγχαίνεται μέσω του διαμοιρασμού βαρών. Οπότε καταφέρνουμε να μειώσουμε την χωρητικότητα της μηχανής μάθησης, πράγμα το οποίο με την σειρά του, βελτιώνει τη δυνατότητα γενίκευσης της μηχανής. [8] Επίσης να αναφέρουμε ότι κάνοντας προσαρμογές στις ελεύθερες παραμέτρους του δικτύου γίνονται κάνοντας χρήση της στοχαστικής κατάστασης λειτουργίας της μάθησης με οπισθοδιάδοση σφάλματος. Με το να γίνεται διαμοιρασμός βαρών μπορεί να γίνει υλοποίηση του συνελικτικού δικτύου σε παράλληλη μορφή. Αυτό είναι θετικό απο την μεριά του συνελικτικού δικτύου έναντι ενός πλήρως συνδεδεμένου perceptron πολλών επιπέδων. Αυτό που μάθαμε απο το

συνελικτικό δίκτυο του παραπάνω σχήματος είναι: πρώτον ,ένα perceptron πολλών επιπέδων κάποιου διαχειρίσιμου μεγέθους μπορεί να μαθαίνει μια πολύπλοκη , μεγάλων διαστάσεων ,μη γραμμική αντιστοίχιση ελέγχοντας (με περιορισμούς) την σχεδίασή του μέσω της ενσωμάτωσης προηγούμενης γνώσης όσον αφορά την εργασία που πρέπει να επιτελεί.

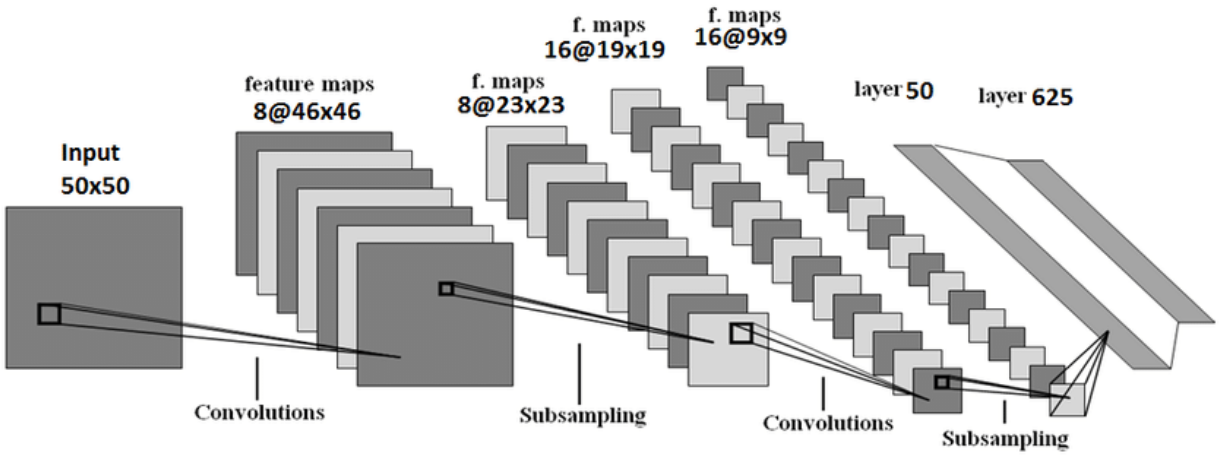
Και δεύτερον ,τα συναπτικά βάρη και οι στάθμες πόλωσης διδάσκονται κάνοντας εφαρμογή του απλού αλγόριθμου BK διαδοχικά σε όλο το δείγμα εκπαίδευσης.

Παραδείγματα – Σχήματα με Convolutional Neural Networks (Συνελικτικά Δίκτυα)



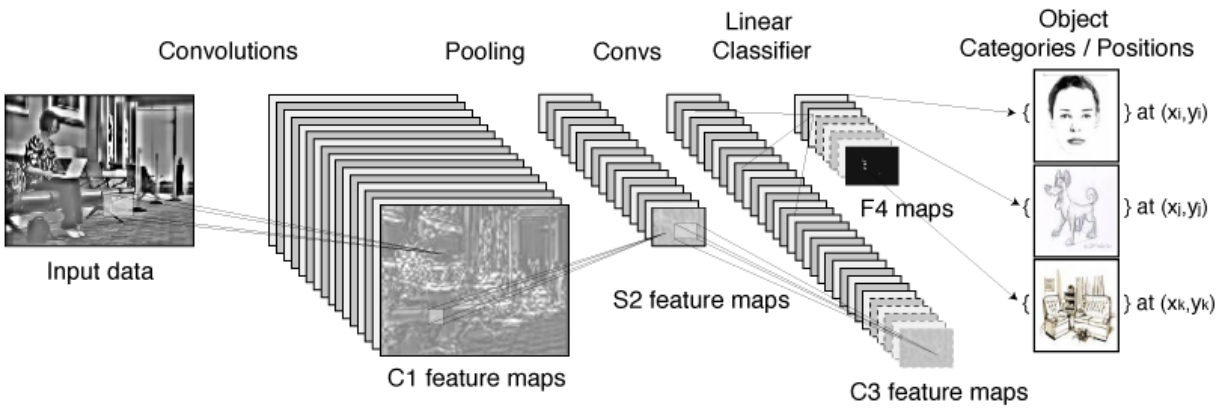
πηγή

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html



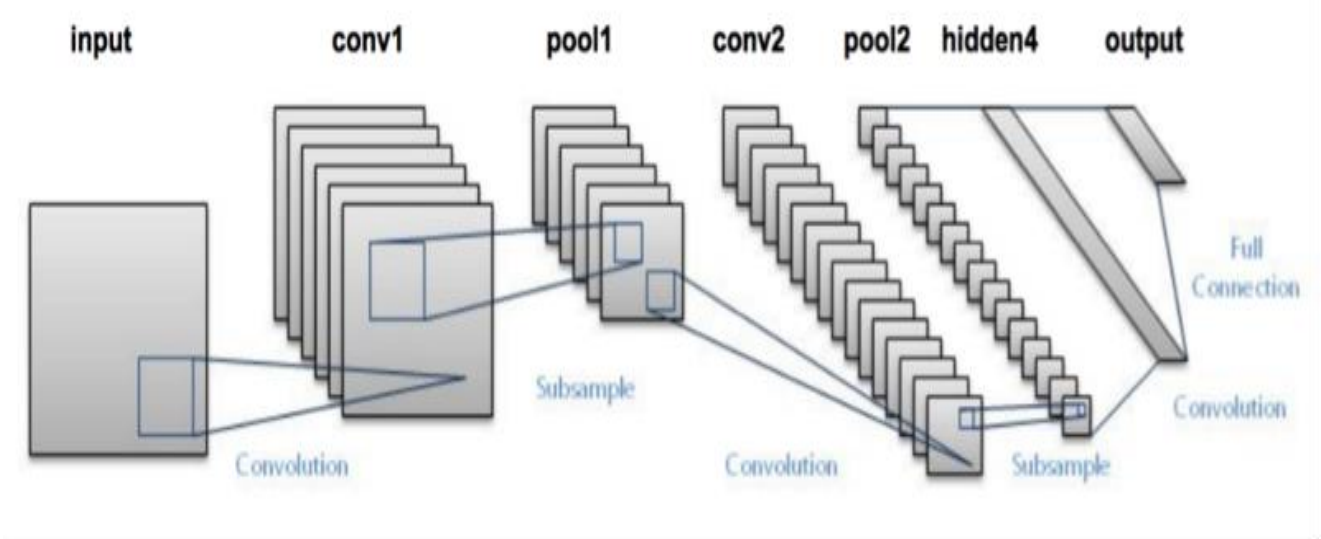
Πηγή

https://www.researchgate.net/figure/An-example-of-the-architecture-of-the-convolutional-neural-network-used-in-this-work_fig3_309487032



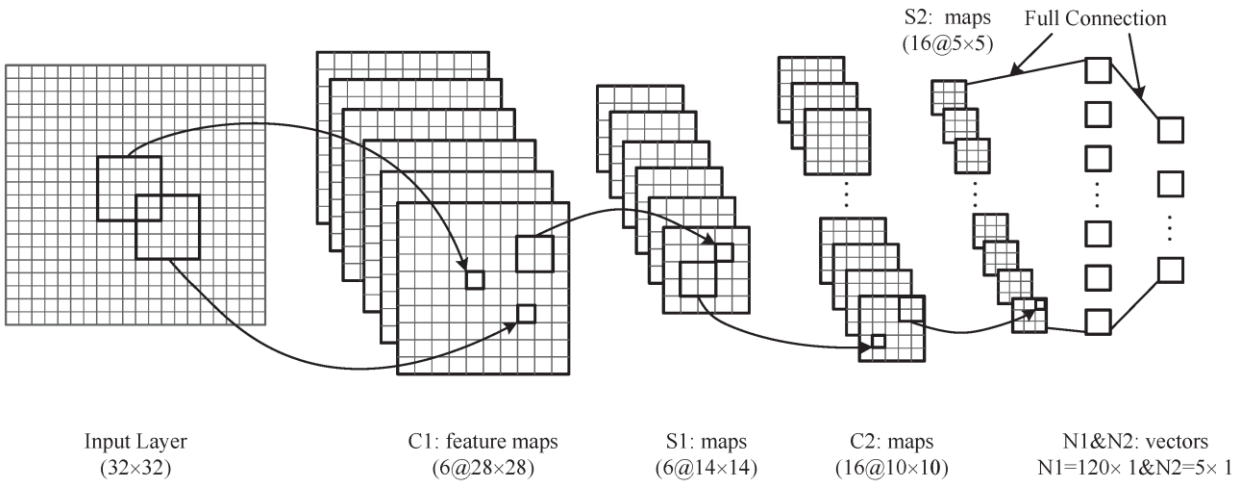
Πηγή

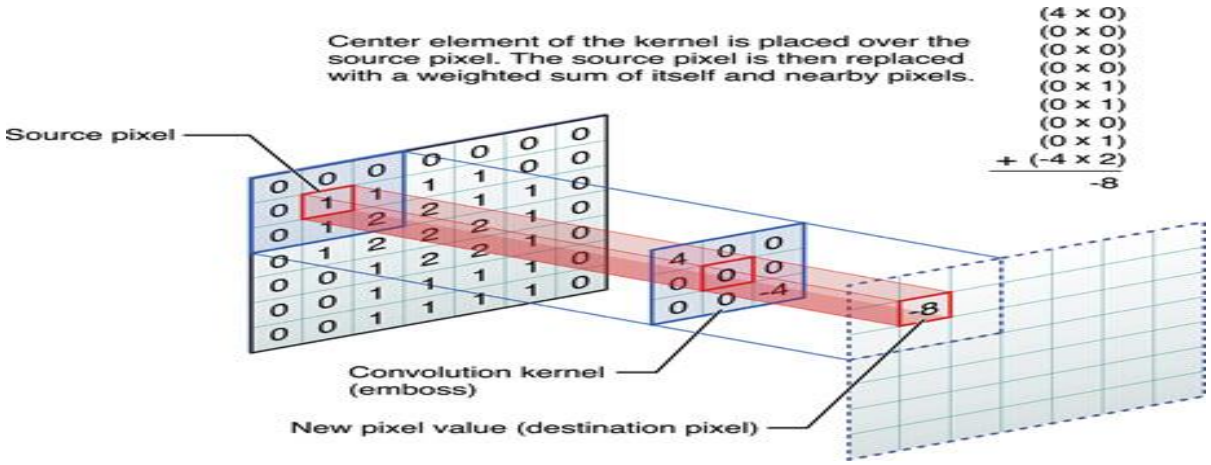
https://datascience.stackexchange.com/questions/14899/how-to-draw-deep-learning-network-architecture-diagrams



πηγή

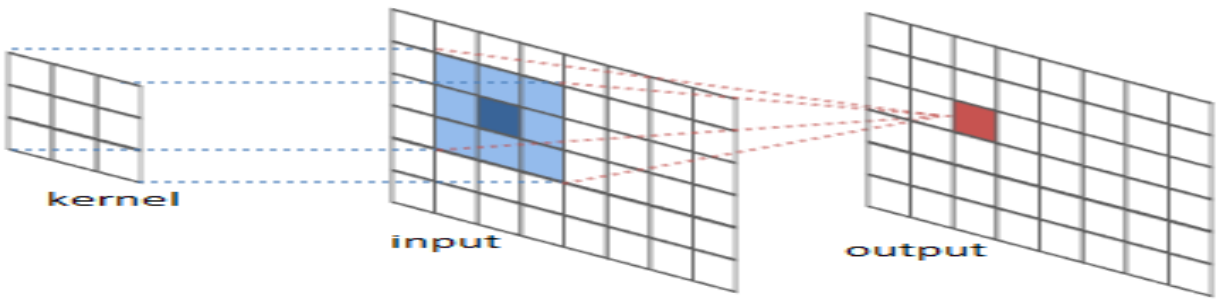
https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59





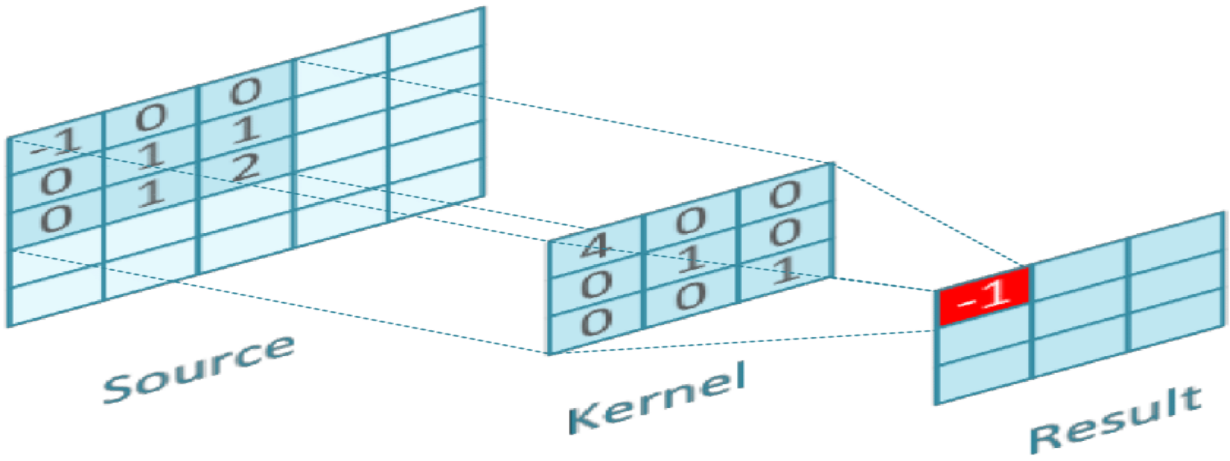
Πηγή

https://stats.stackexchange.com/questions/114385/what-is-the-difference-between-convolutional-neural-networks-restricted-boltzma



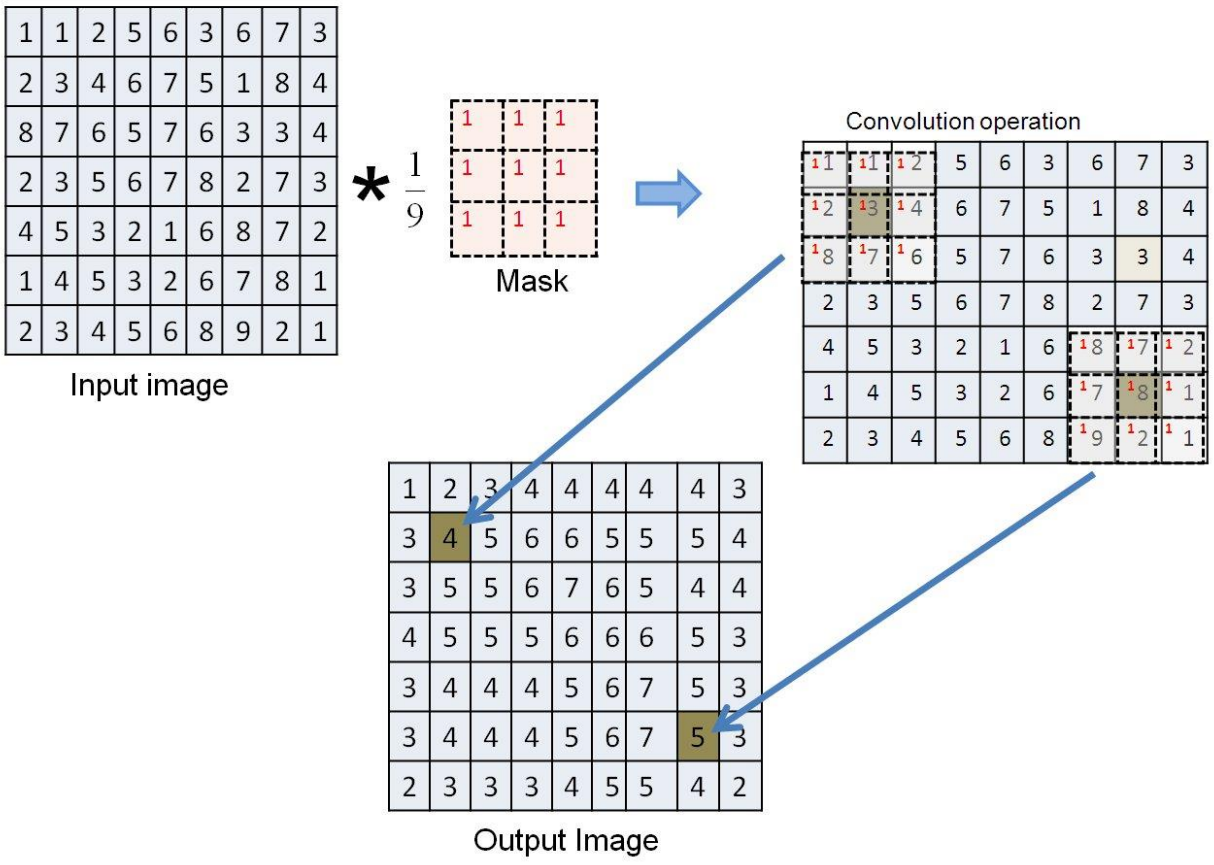
Πηγή

https://www.raywenderlich.com/188-beginning-machine-learning-with-keras-core-ml



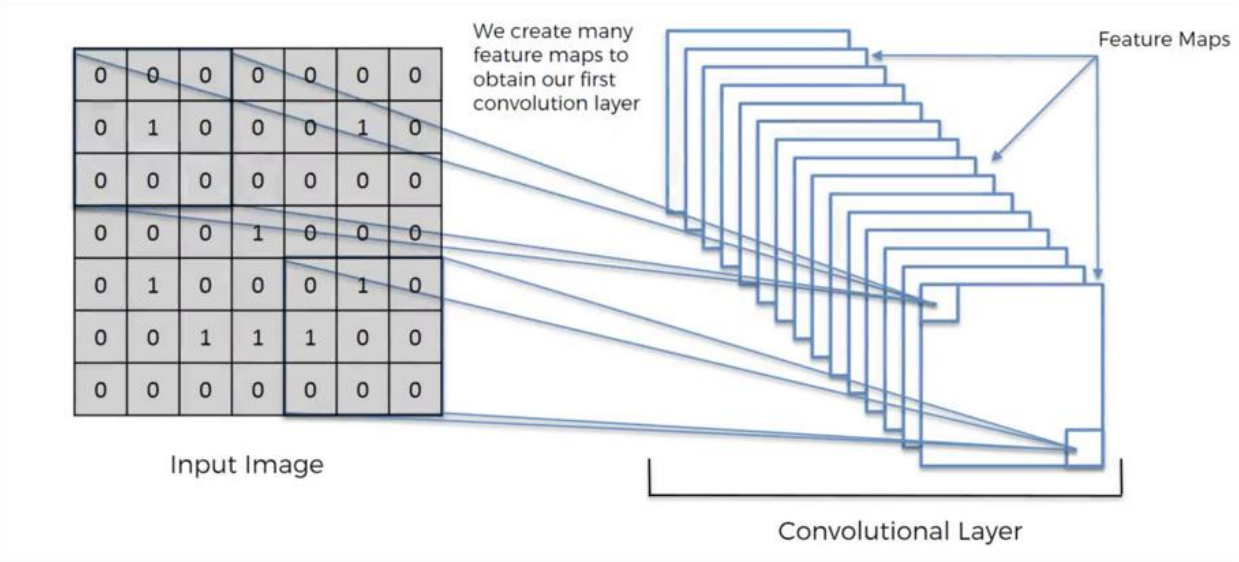
Πηγή

https://mejoresfotos.eu/3x3-5x5-www-picswe-com.html



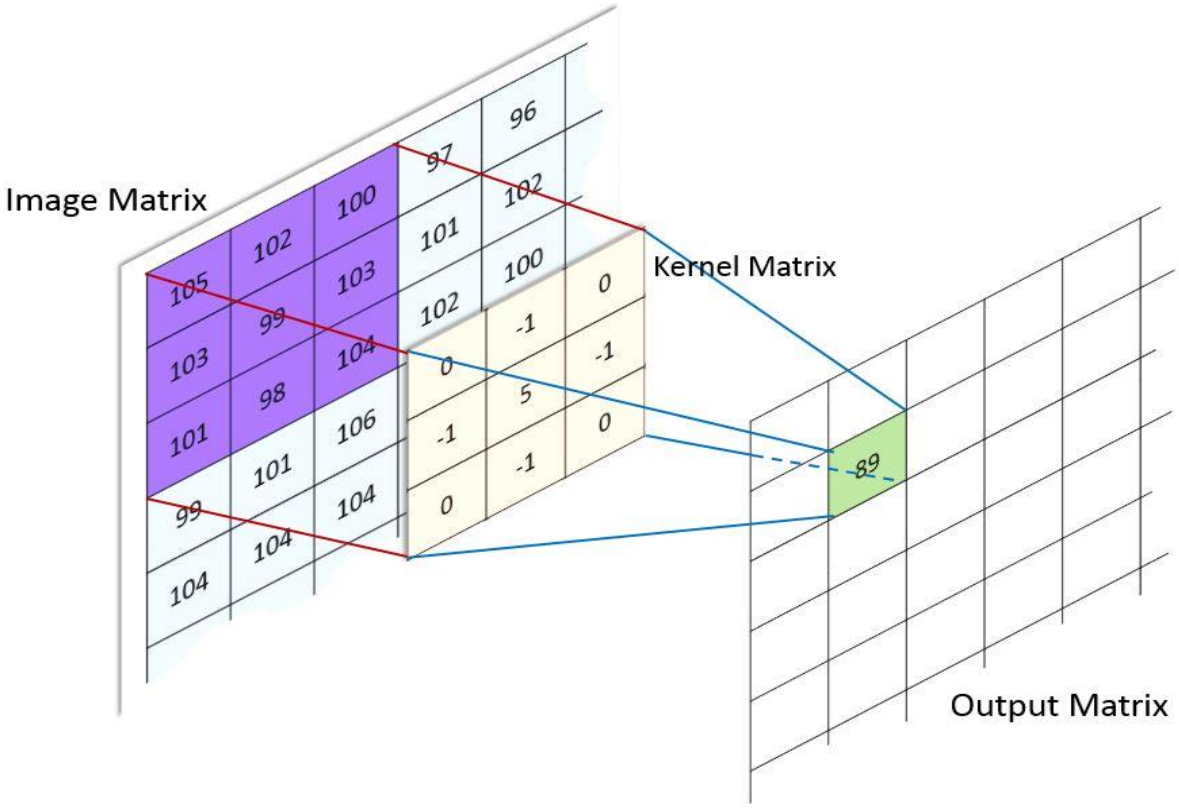
πηγή

https://cse19-iiith.vlabs.ac.in/theory.php?exp=neigh



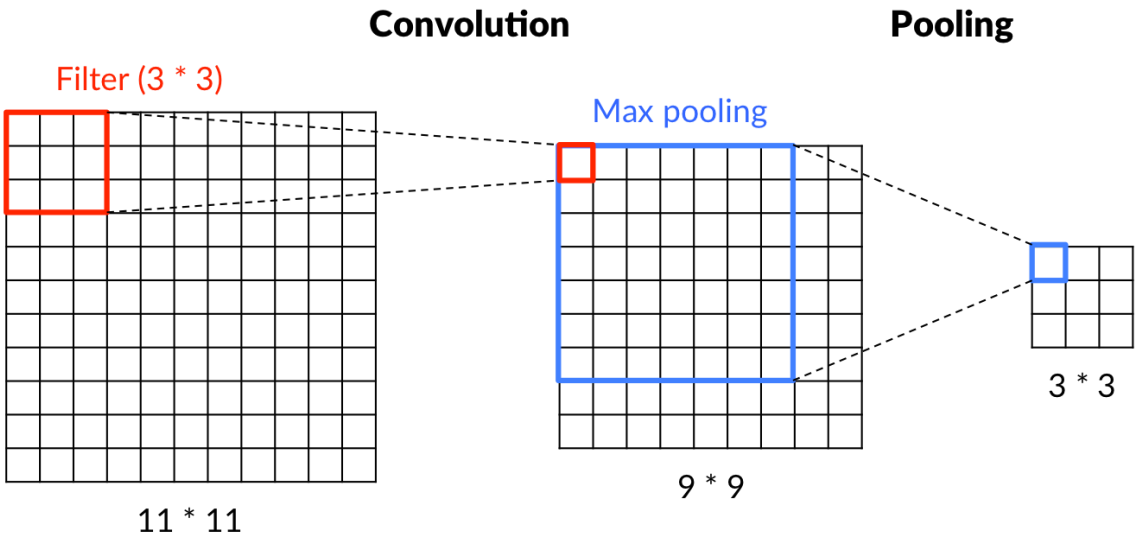
πηγή

https://www.superdatascience.com/the-ultimate-guide-to-convolutional-neural-networks-cnn/



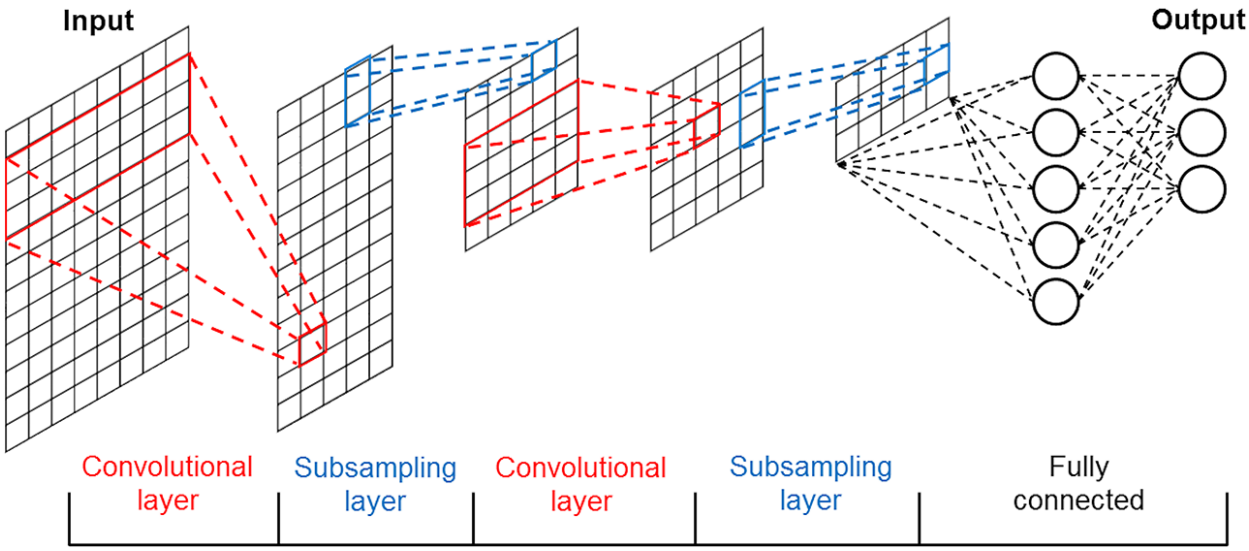
πηγή

https://brilliant.org/wiki/convolutional-neural-network/



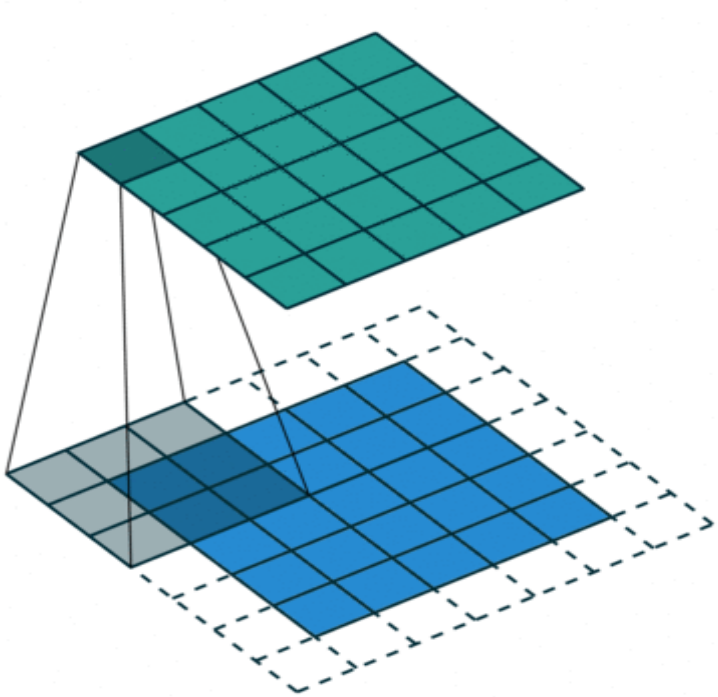
πηγή

https://saitoxu.io/2017/01/01/convolution-and-pooling.html



πηγή

https://www.spiedigitallibrary.org/journals/neurophotonics/volume-5/issue-01/011008/Convolutional-neural-network-for-high-accuracy-functional-near-infrared-spectroscopy/10.1117/1.NPh.5.1.011008.full?SSO=1



πηγή

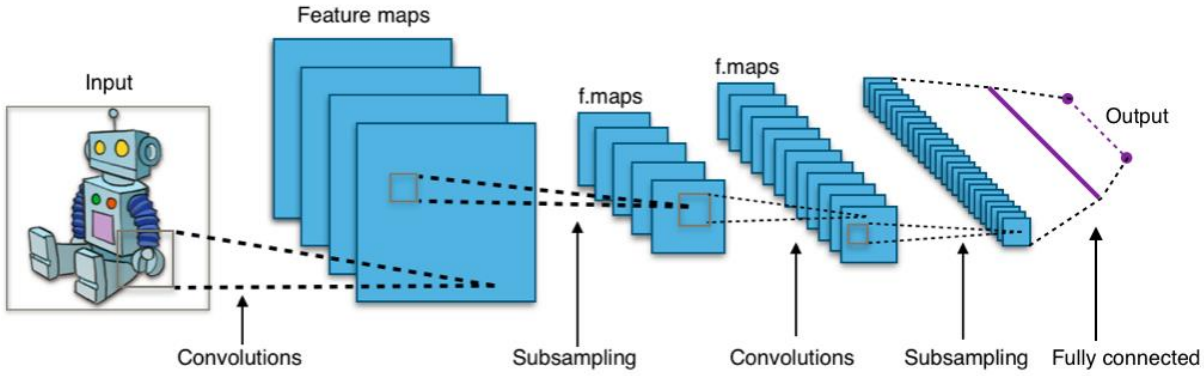
https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

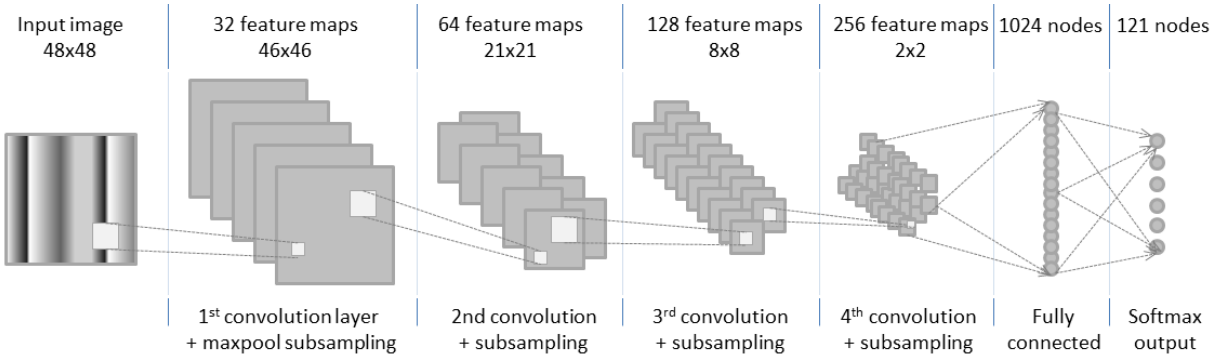
πηγή

https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050



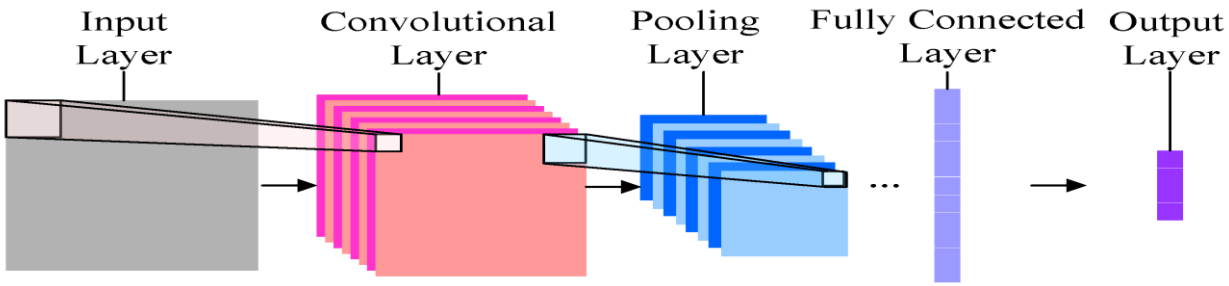
Πηγή

http://www.jessicayung.com/explaining-tensorflow-code-for-a-convolutional-neural-network/



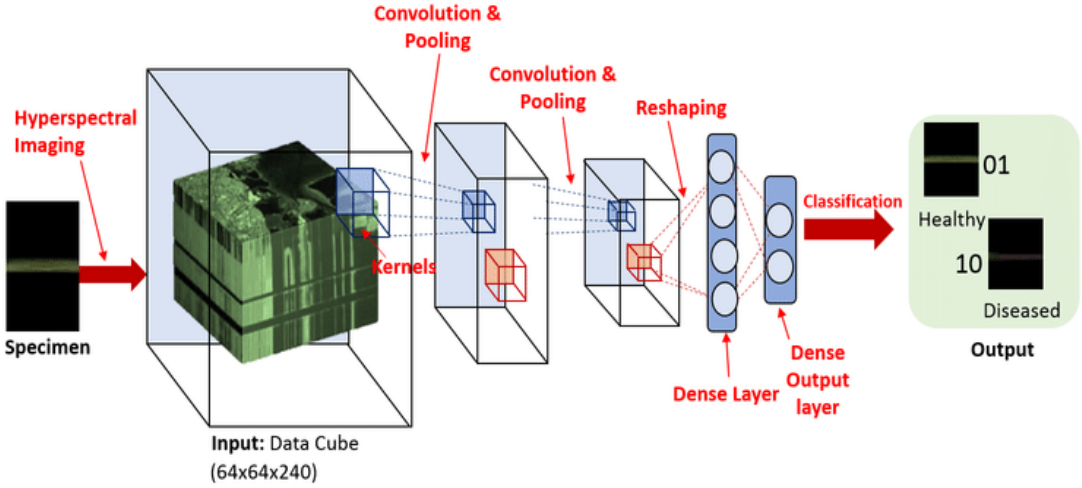
Πηγή

https://flashiontrash.com/convolutional-neural-network/10141752531225251504



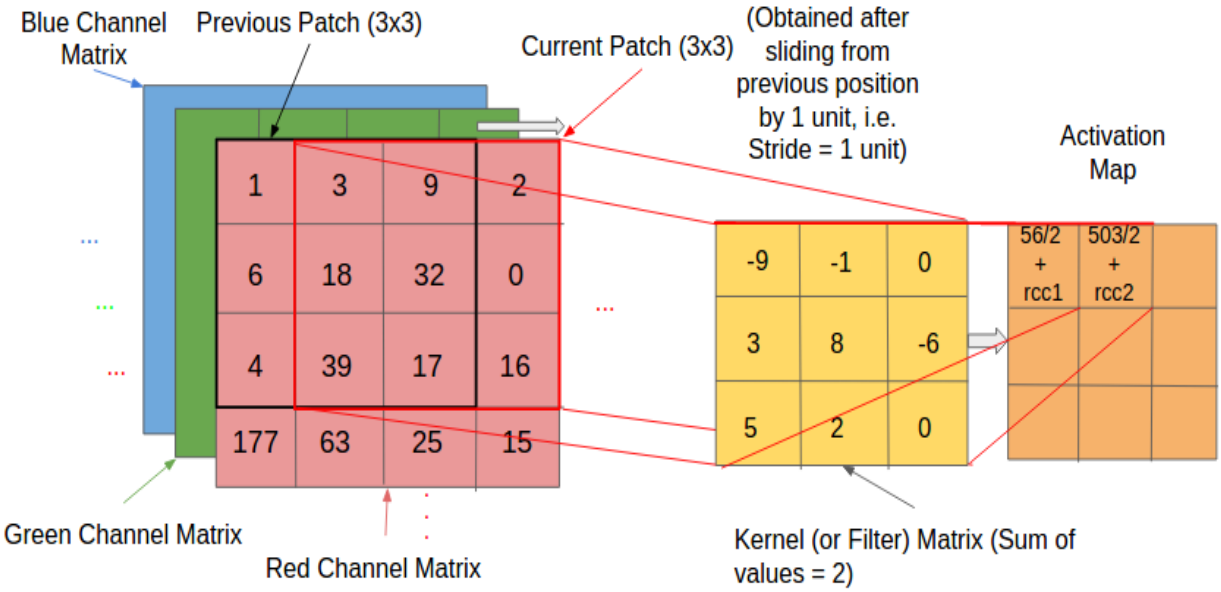
πηγή

https://www.mdpi.com/2078-2489/7/4/61



πηγή

https://www.researchgate.net/figure/3D-Convolutional-Neural-Network-Architecture-for-Classification_fig2_324744613



πηγή

https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/

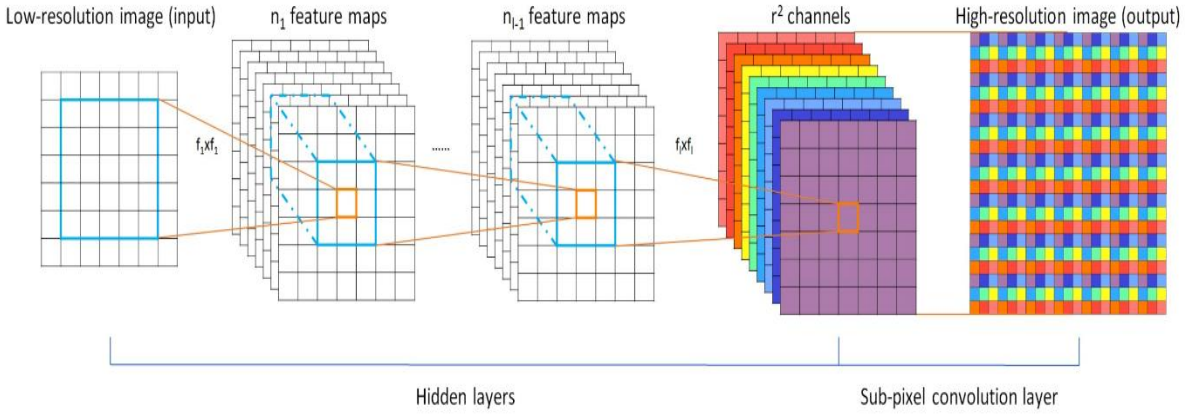
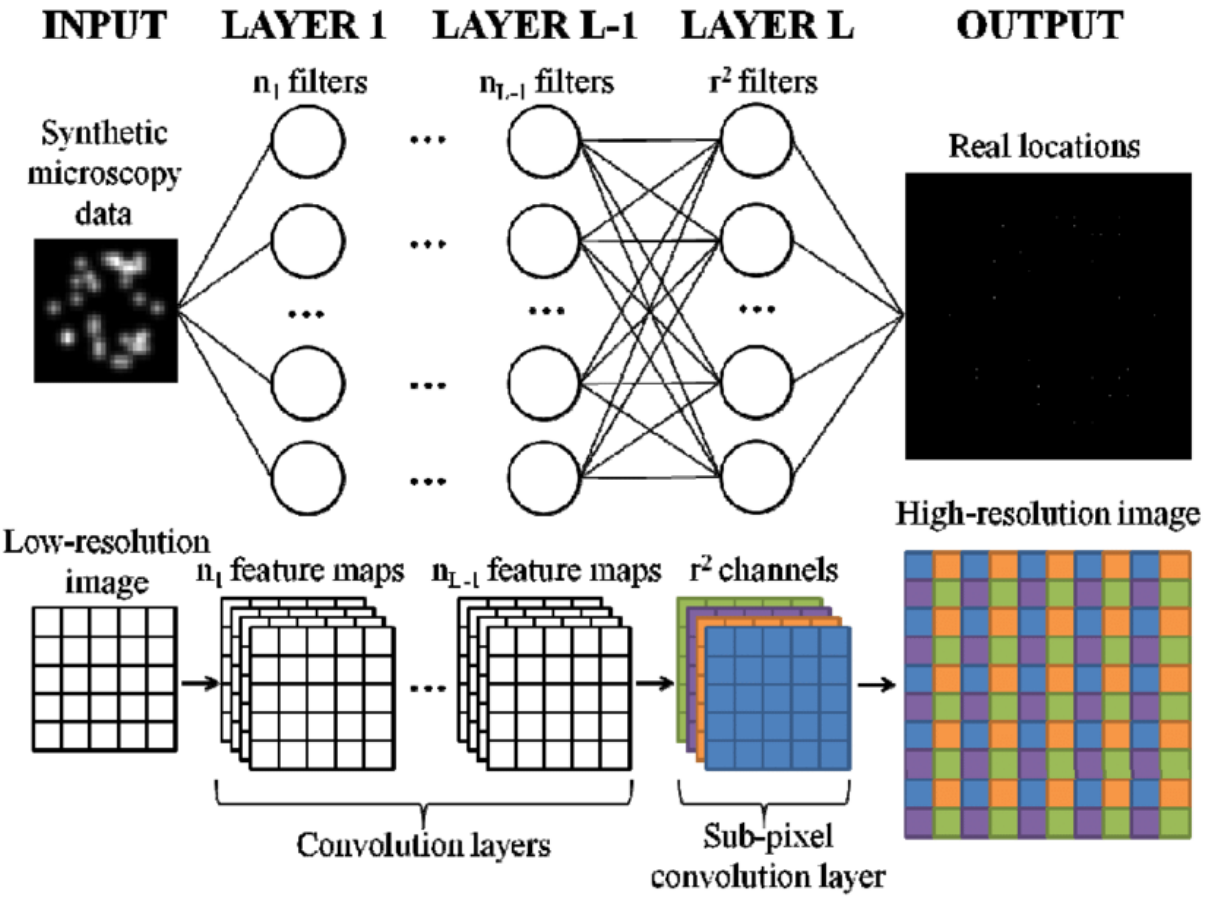


Figure 1. The proposed efficient sub-pixel convolutional neural network (ESPCN), with two convolution layers for feature maps extraction, and a sub-pixel convolution layer that aggregates the feature maps from LR space and builds the SR image in a single step.

πηγή

https://github.com/kweisamx/TensorFlow-ESPCN

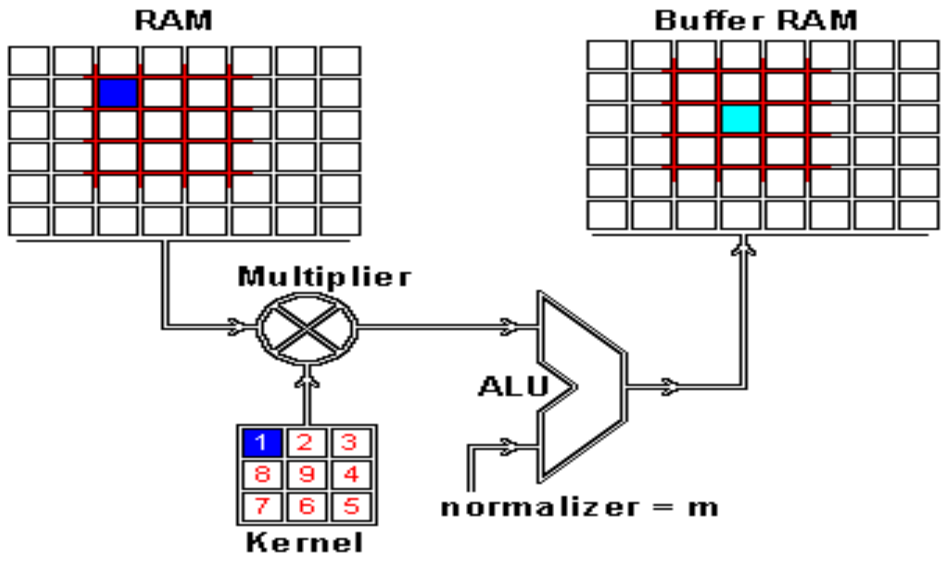


πηγή

https://www.researchgate.net/figure/Schematic-depicting-how-a-convolutional-neural-network-trained-on-a-low-resolution-image_fig1_328469215

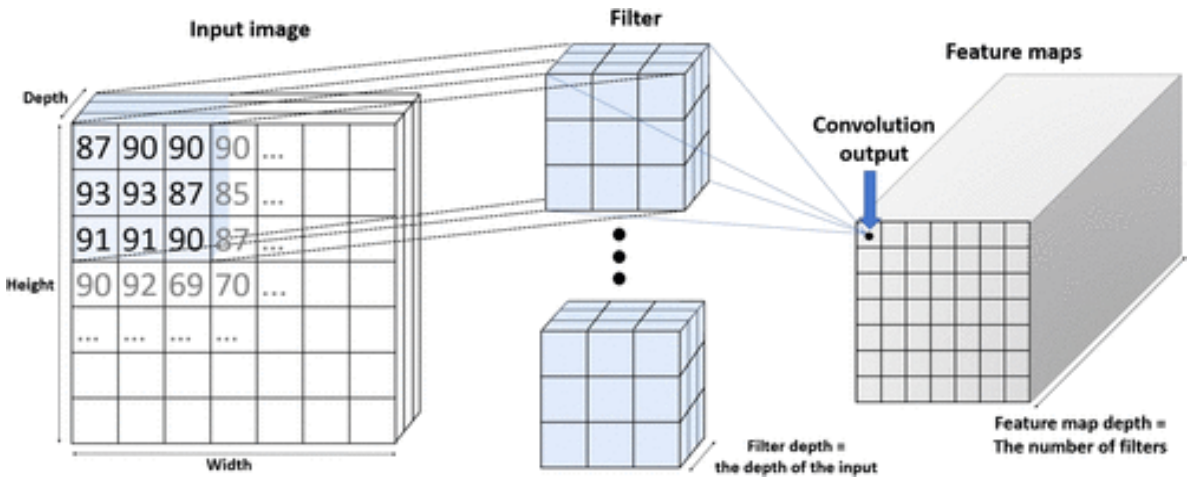
πηγή

https://www.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187



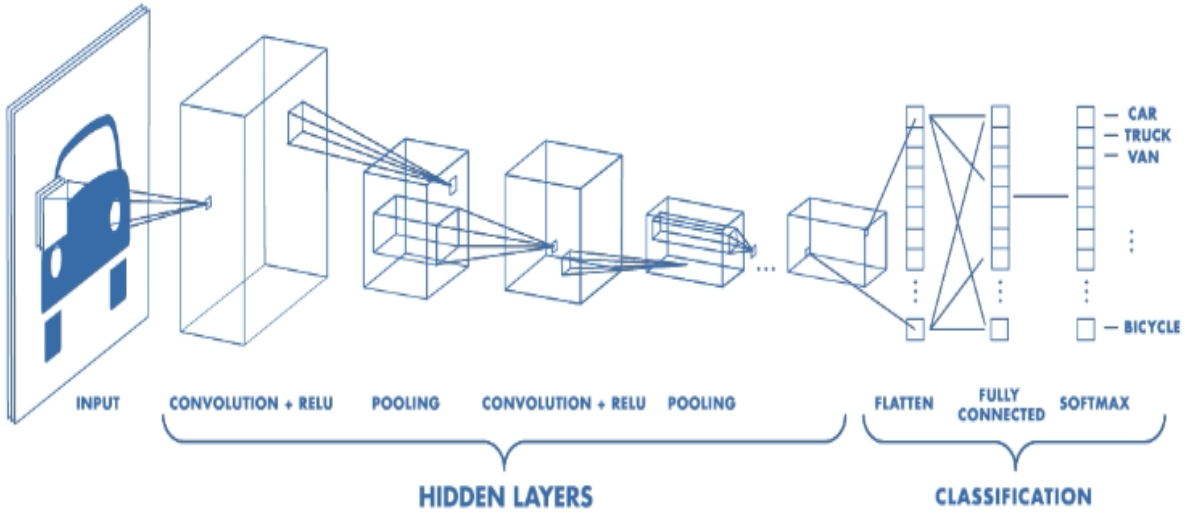
πηγή

https://gifer.com/en/FqOh



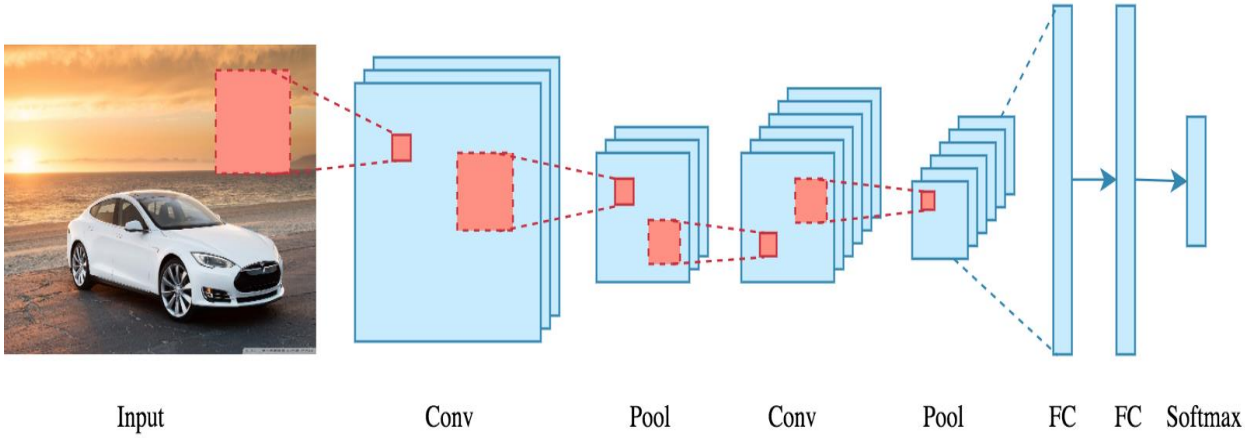
πηγή

https://ascelibrary.org/doi/10.1061/%28ASCE%29CP.1943-5487.0000731



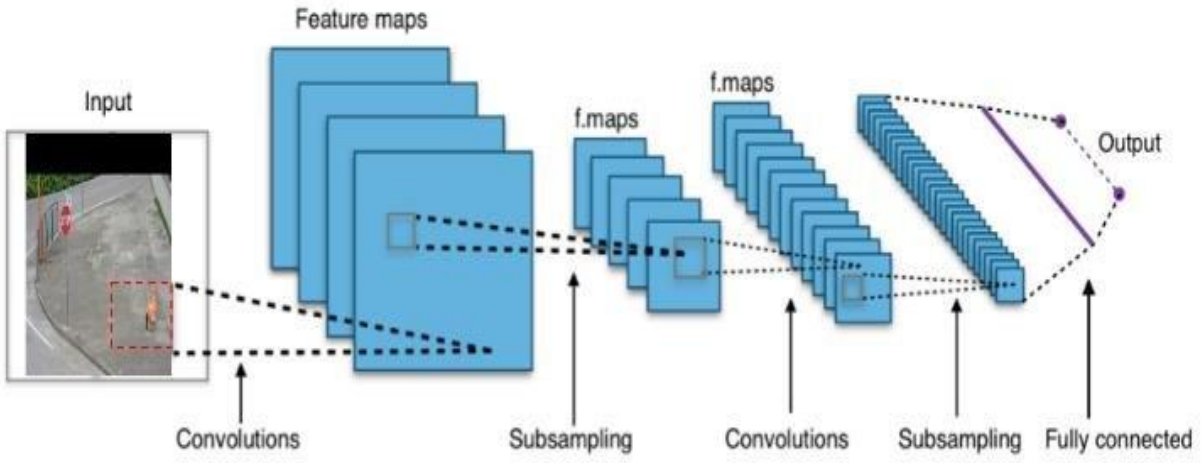
πηγή

https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050



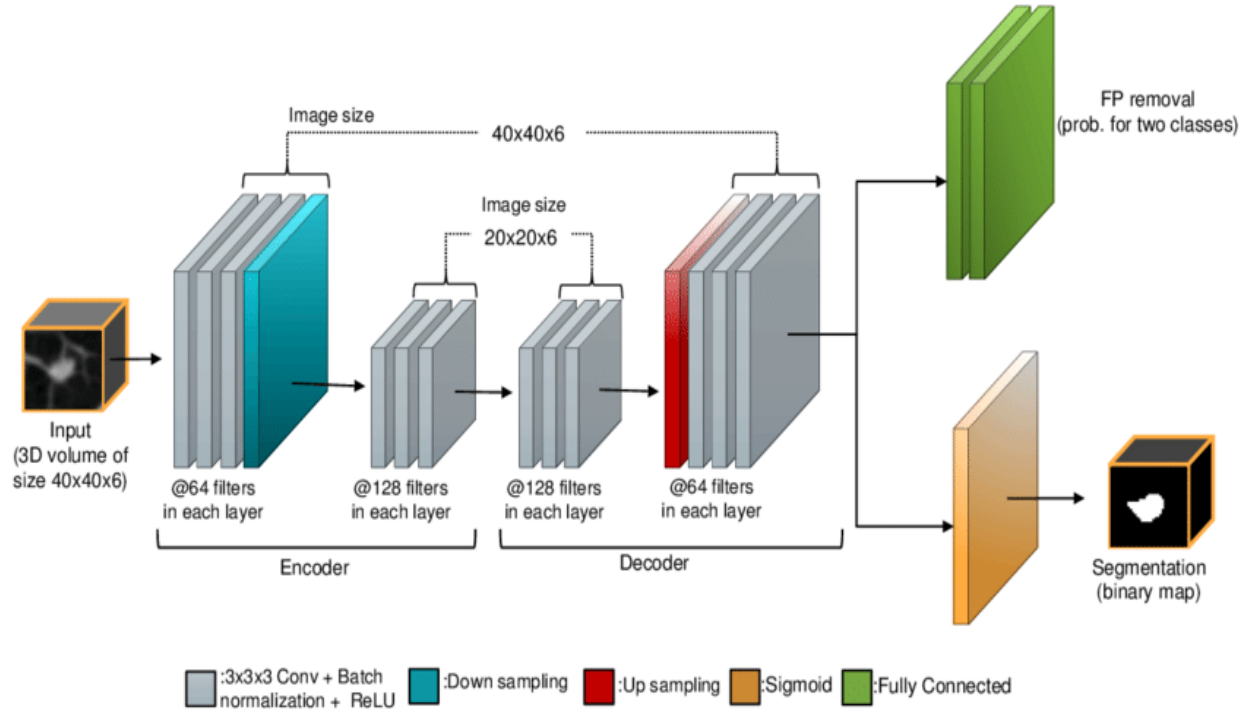
πηγή

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2



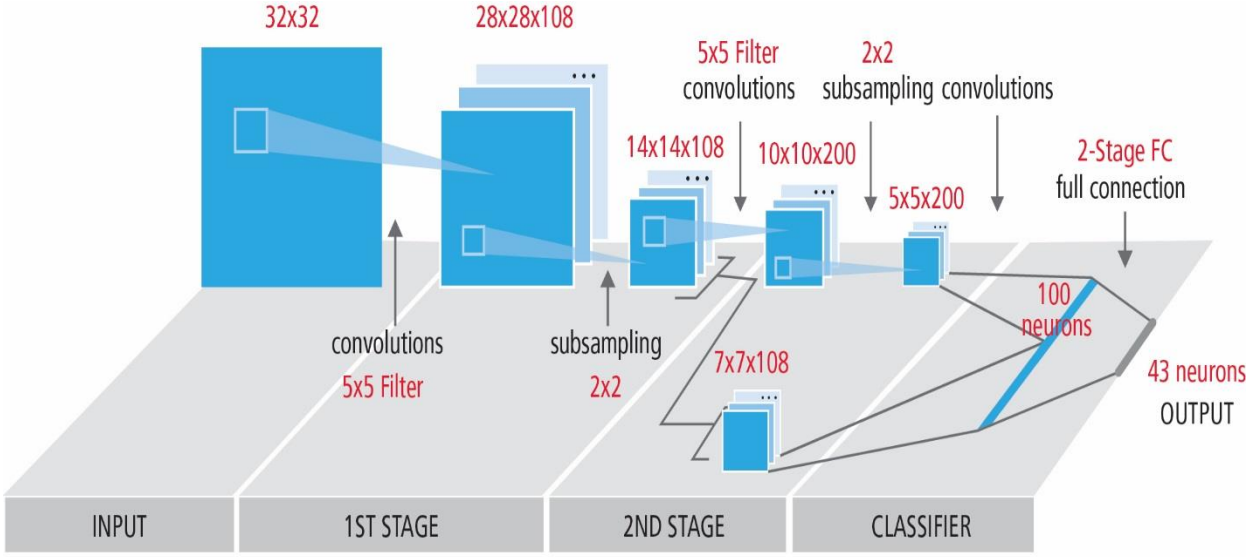
πηγή

https://www.researchgate.net/figure/Main-operations-of-a-typical-CNN-architecture_fig2_323589806



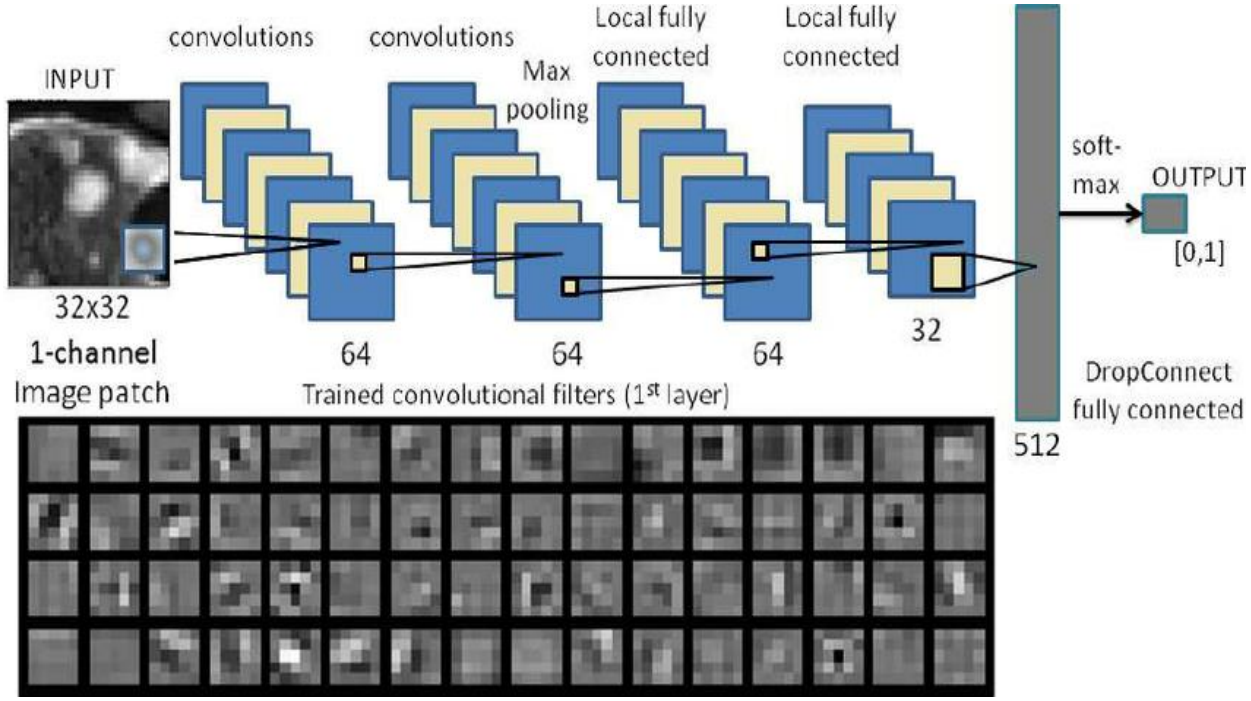
πηγή

https://www.researchgate.net/figure/The-3D-deep-multi-task-CNN-architecture-The-size-of-all-convolutions-are-333-with-a_fig3_323276486



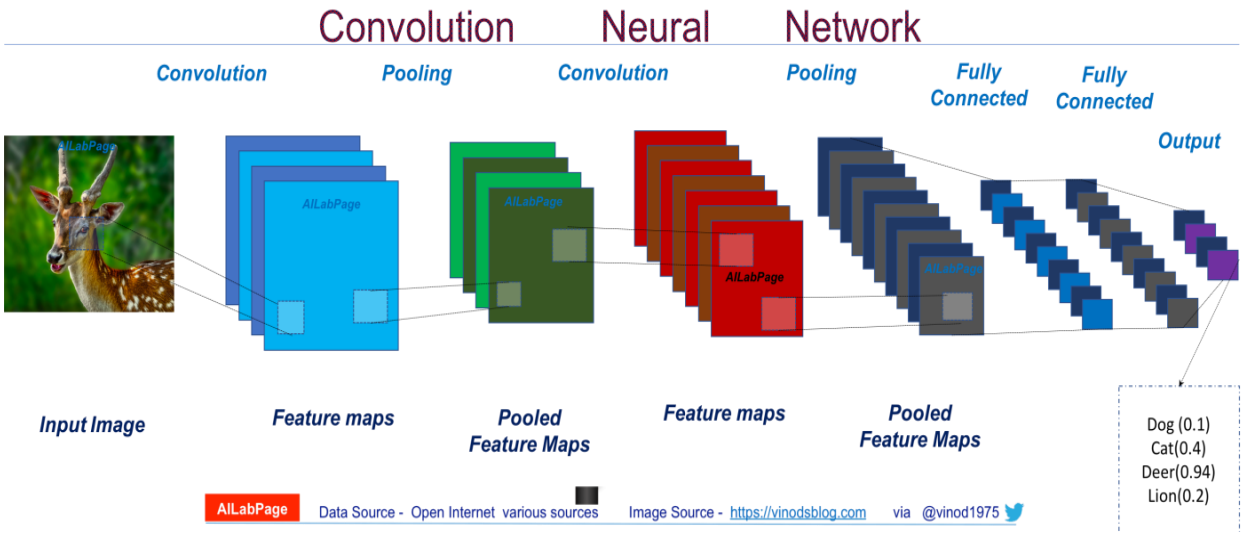
πηγή

https://community.cadence.com/cadence_blogs_8/b/breakfast-bytes/posts/enns



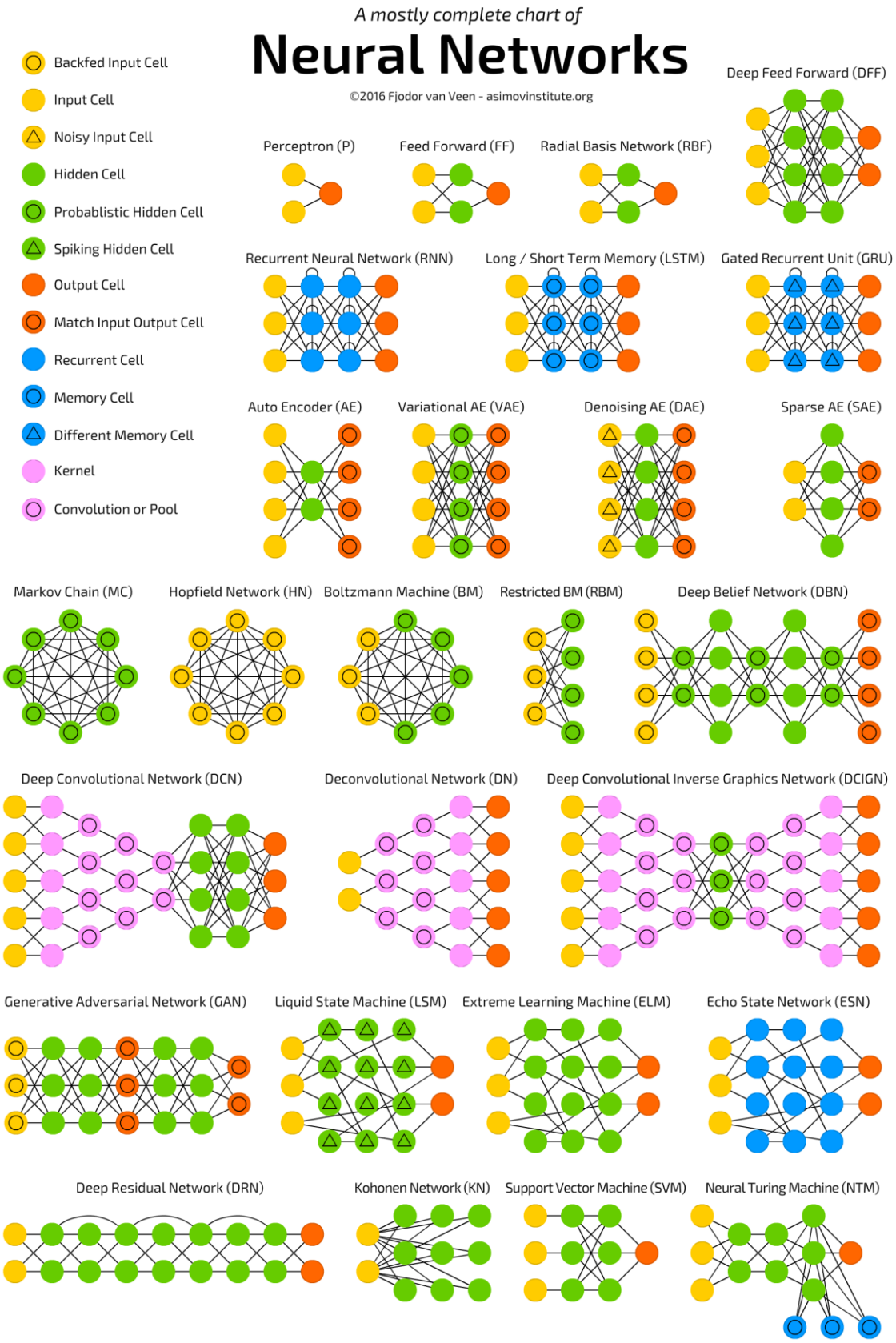
πηγή

https://www.researchgate.net/figure/The-proposed-convolution-neural-network-consists-of-two-convolutional-layers-max-pooling_fig3_264160750



πηγή

https://www.datasciencecentral.com/profiles/blogs/everything-you-need-to-know-about-convolutional-neural-networks

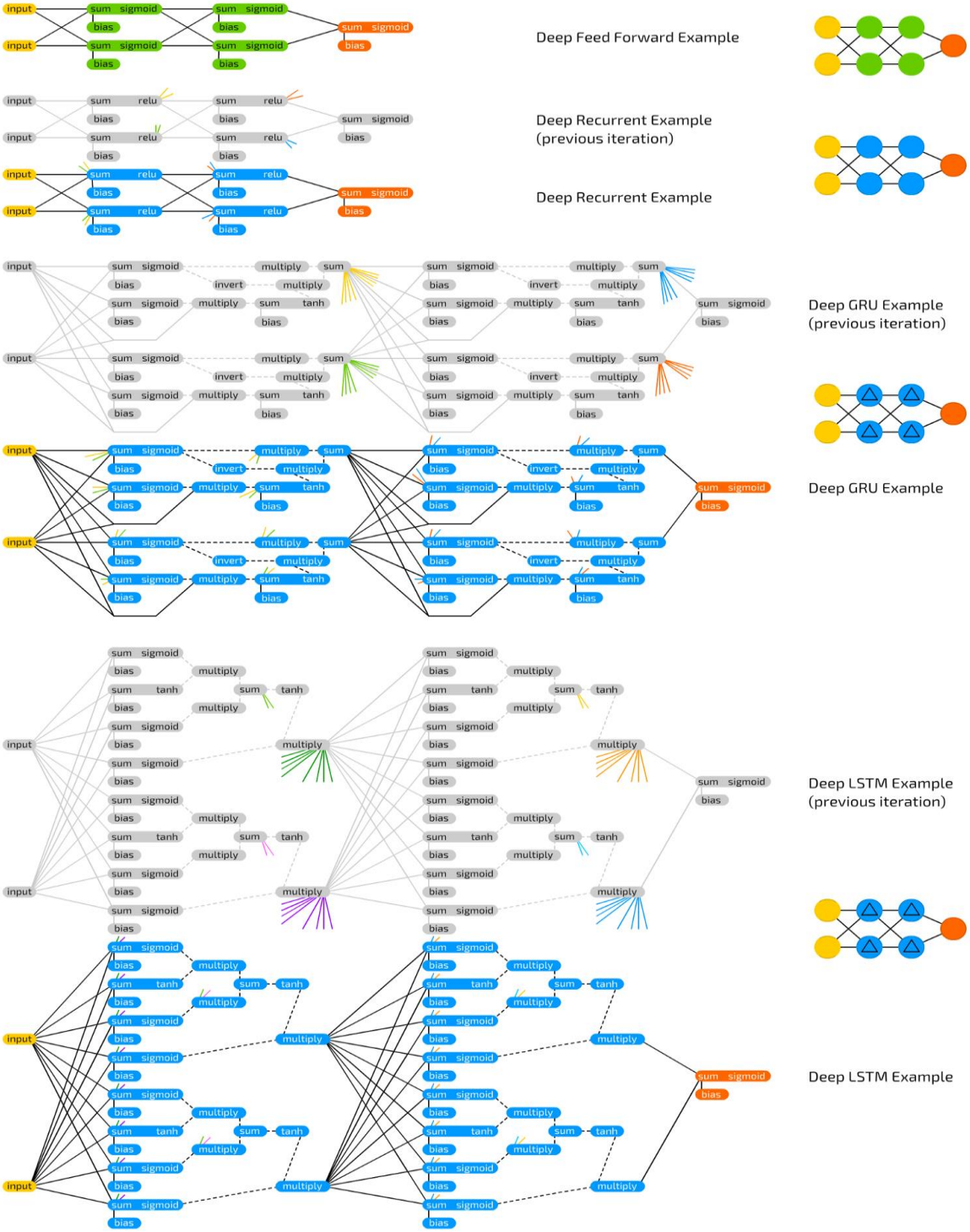


https://ai.stackexchange.com/questions/5546/what-is-the-difference-between-a-convolutional-neural-network-and-a-regular-neur

An informative chart to build

Neural Network Graphs

©2016 Fjodor van Veen - asimovinstitute.org



https://www.pinterest.nz/pin/510384570268313769/

Linear Vector Spaces:

Definition: A linear vector space, X is a set of elements (vectors) defined over a scalar field, F, that satisfies the following conditions:

1) if $x \in X$ and $y \in X$ then $x+y \in X$. **2)** $x+y=y+x$ **3)** $(x+y)+z=x+(y+z)$
4) There is a unique vector $0 \in X$, such that $x+0=x$ for all $x \in X$.
5) For each vector $x \in X$ there is a unique vector in X, to be called $(-x)$, such that $x+(-x)=0$. **6)** multiplication, for all scalars $a \in F$, and all vectors $x \in X$,
7) For any $x \in X$, $1x=x$ (for scalar 1).
8) For any two scalars $a \in F$ and $b \in F$ and any $x \in X$, $a(bx)=(ab)x$.
9) $(a+b)x=ax+bx$. **10)** $a(x+y)=ax+ay$.

Linear Independence:

Consider n vectors $\{x_1, x_2, .., x_n\}$. If there exists n scalars $a_1, a_2, \ldots, a_n$, at least one of which is nonzero, such that $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$, then the $\{x_i\}$ are linearly dependent.

Spanning a Space:

Let X be a linear vector space and let $\{u_1, u_2, \ldots, u_n\}$ be a subset of vectors in X. This subset spans X if and only if for every vector $x \in X$ there exist scalars $x_1, x_2, \ldots, x_n$ such that $x = x_1u_1 + x_2u_2 + \ldots + x_nu_n$.

Inner Product:

(x, y) for any scalar function of x and y.
1. $(x,y) = (y,x)$ 2. $(x, ay_1 + by_2) = a(x,y_1) + b(x,y_2)$
3. $(x,x) \geq 0$, where equality holds iff x is the zero vector.

Norm:

Ascalar function $\|x\|$ is called a norm if it satisfies:
1. $\|x\| \geq 0$ 2. $\|x\| = 0$ if and only if $x = 0$.
3. $\|ax\| = |a|\|x\|$ 4. $\|x + y\| \leq \|x\| + \|y\|$

Angle:

The angle θ bet. 2 vectors x and y is defined by $\cos\theta = \frac{(x,y)}{\|x\|\,\|y\|}$

Orthogonality:

2 vectors $x, y \in X$ are said to be orthogonal if $(x,y) = 0$.

Gram Schmidt Orthogonalization:

Assume that we have n independent vectors $y_1, y_2, \ldots, y_n$. From these vectors we will obtain n orthogonal vectors $v_1, v_2, \ldots, v_n$.

$$v_1 = y_1, \quad v_k = y_k - \sum_{i=1}^{k-1} \frac{(v_i, y_k)}{(v_i, v_i)} v_i,$$

$$where\ \frac{(v_i, y_k)}{(v_i, v_i)} v_i\ is\ the\ projection\ of\ y_k\ on\ v_i$$

Vector Expansions:

$$x = \sum_{i=1}^{n} x_i v_i = x_1 v_1 + x_2 v_2 + \cdots + x_n v_n,$$

$$for\ orthogonal\ vectors, x_j = \frac{(v_j, x)}{(v_j, v_j)}$$

Reciprocal Basis Vectors:

$$(r_i, v_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}, \quad x_j = (r_j, x)$$

To compute the reciprocal basis vectors: set $\mathbf{B} = [v_1\ v_2\ \ldots\ v_n]$,

$\mathbf{R} = [r_1\ r_2\ \ldots\ r_n]$, $\mathbf{R}^T = \mathbf{B}^{-1}$ In matrix form: $\mathbf{x}^v = \mathbf{B}^{-1}\mathbf{x}^s$

Transformations:

A *transformation* consists of three parts:
domain: $X = \{x_i\}$, range: $Y = \{y_i\}$, and a rule relating each $x_i \in X$ to an element $y_i \in Y$.

Linear Transformations: transformation A is *linear* if:
1. for all $x_1, x_2 \in X$, $A(x_1+x_2) = A(x_1) + A(x_2)$
2. for all $x \in X, a \in R$, $A(ax) = aA(x)$

Matrix Representations:

Let $\{v_1, v_2, \ldots, v_n\}$ be a basis for vector space X, and let $\{u_1, u_2, \ldots, u_n\}$ be a basis for vector space Y. Let A be a linear transformation with domain X and range Y: $A(x) = y$

The coefficients of the matrix representation are obtained from

$$A(v_j) = \sum_{i=1}^{m} a_{ij} u_i$$

Change of Basis: $\mathbf{B}_t = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots \mathbf{t}_n]$, $\mathbf{B}_w = [\mathbf{w}_1\ \mathbf{w}_2\ \ldots \mathbf{w}_n]$

$$\mathbf{A}' = [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t]$$

Eigenvalues & Eigenvectors: $\mathbf{Az} = \lambda\mathbf{z}$, $|[\mathbf{A} - \lambda\mathbf{I}]| = 0$

Diagonalization: $\mathbf{B} = [\mathbf{z}_1\ \mathbf{z}_2\ \ldots \mathbf{z}_n]$,

where $\{\mathbf{z}, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ are the eigenvectors of a square matrix A,

$$[\mathbf{B}^{-1}\mathbf{A}\mathbf{B}] = \mathrm{diag}([\lambda_1\ \lambda_2\ \ldots \lambda_n])$$

Perceptron Architecture:

$$\mathbf{a} = hardlim(\mathbf{Wp} + \mathbf{b}),\ \mathbf{W} = [\ {}_1\mathbf{w}^T\ {}_2\mathbf{w}^T\ \ldots\ {}_S\mathbf{w}^T]^T,$$

$$a_i = hardlim(n_i) = hardlim(\ {}_i\mathbf{w}^T\ \mathbf{p} + b_i)$$

Decision Boundary: ${}_i\mathbf{w}^T\ \mathbf{p} + b_i = 0$

The decision boundary is always orthogonal to the weight vector.
Single-layer perceptrons can only classify linearly separable vectors.

Perceptron Learning Rule

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{ep}^T,\ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e},$$

$$where\quad \mathbf{e} = \mathbf{t} - \mathbf{a}$$

Hebb's Postulate: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Linear Associator: $\mathbf{a} = purelin(\mathbf{Wp})$

The Hebb Rule: *Supervised Form:* $w_{ij}^{new} = w_{ij}^{old} + t_{qi}P_{qi}$

$$\mathbf{W} = \mathbf{t}_1\mathbf{P}_1^T + \mathbf{t}_2\mathbf{P}_2^T + \cdots + \mathbf{t}_Q\mathbf{P}_Q^T$$

$$\mathbf{W} = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots \mathbf{t}_Q]\begin{bmatrix}\mathbf{P}_1^T \\ \mathbf{P}_2^T \\ \vdots \\ \mathbf{P}_Q^T\end{bmatrix} = \mathbf{TP}^T$$

Pseudoinverse Rule: $\mathbf{W} = \mathbf{TP}^+$

When the number, R, of rows of $\mathbf{P}$ is greater than the num ber of columns, Q, of $\mathbf{P}$ and the columns of $\mathbf{P}$ are independent, then the pseudoinverse can be computed by $\mathbf{P}^+ = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T$

Variations of Hebbian Learning:

Filtered Learning (Ch.14): $\mathbf{W}^{new} = (1-\gamma)\mathbf{W}^{old} + \alpha\mathbf{t}_q\mathbf{p}_q^T$

Delta Rule (Ch.10): $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha(\mathbf{t}_q - \mathbf{a}_q)\mathbf{p}_q^T$

Unsupervised Hebb (Ch.13): $\mathbf{W}^{new} = \mathbf{W}^{old} + \alpha\mathbf{a}_q\mathbf{p}_q^T$

Taylor: $F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^*)\nabla^2 F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*) + \cdots$

Grad $\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1}F(\mathbf{x})\ \ \frac{\partial}{\partial x_2}F(\mathbf{x})\ \ \ldots\ \frac{\partial}{\partial x_n}F(\mathbf{x})\right]^T$

Hessian: $\nabla^2 F(\mathbf{x}) =$

$$\begin{bmatrix} \frac{\partial}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial}{\partial x_1 \partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_1 \partial x_n}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2 \partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_2^2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_2 \partial x_n}F(\mathbf{x}) \\ \vdots & \vdots & \vdots \\ \frac{\partial}{\partial x_n \partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_n \partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_n^2}F(\mathbf{x}) \end{bmatrix}$$

Directional Derivatives:

1st Dir.Der.: $\frac{\mathbf{p}^T\nabla F(\mathbf{x})}{\|\mathbf{p}\|}$, **2nd Dir.Der.:** $\frac{\mathbf{p}^T\nabla^2 F(\mathbf{x})\mathbf{p}}{\|\mathbf{p}\|^2}$

Minima:

Strong Minimum: if a scalar $\delta > 0$exists, such that $F(x) < F(x+\Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.
Global Minimum: if $F(x) < F(x+\Delta x)$ for all $\Delta x \neq 0$
Weak Minimum: if it is not a strong minimum, and a scalar $\delta > 0$exists, such that $F(x) \leq F(x+\Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.

Necessary Conditions for Optimality:

1st-Order Condition: $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} = 0$ (Stationary Points)
2nd-Order Condition: $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} \geq 0$ (Positive Semi-definite Hessian Matrix).

Quadratic fn.: $F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\,\mathbf{x} + \mathbf{d}^T\mathbf{x} + c$

$$\nabla F(\mathbf{x}) = \mathbf{Ax} + \mathbf{d},\ \nabla^2 F(\mathbf{x}) = \mathbf{A},\ \lambda_{min} \leq \frac{\mathbf{p}^T\mathbf{A}\,\mathbf{p}}{\|\mathbf{p}\|^2} \leq \lambda_{max}$$

https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

General Minimization Algorithm:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ *or* $\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$

Steepest Descent Algorithm:

$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$ where, $\mathbf{g}_k = \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$

Stable Learning Rate: ($\alpha_k = \alpha$, constant) $\alpha < \frac{2}{\lambda_{max}}$

$\{\lambda_1\ \lambda_2, \ldots, \lambda_n\}$ Eigenvalues of Hessian matrix A

Learning Rate to Minimize Along the Line:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \overset{is}{\Rightarrow} \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (For quadratic fn.)

After Minimization Along the Line:

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$

ADALINE: $\mathbf{a} = purelin(\mathbf{W}\mathbf{p} + \mathbf{b})$

Mean Square Error: *(for ADALINE it is a quadratic fn.)*

$F(\mathbf{x}) = E[e^2] = E[(t - a)^2] = E[(t - \mathbf{x}^T\mathbf{z})^2]$

$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\,\mathbf{x}$,

$c = E[t^2]$, $\mathbf{h} = E[t\mathbf{z}]$ *and* $\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow \mathbf{A} = 2\mathbf{R}, \mathbf{d} = -2\mathbf{h}$

Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$,

where $\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$

LMS Algorithm: $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\,\mathbf{e}(k)\,\mathbf{p}^T(k)$

$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\,\mathbf{e}(k)$

Convergence Point: $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$

Stable Learning Rate: $0 < \alpha < 1/\lambda_{max}$ where λ_{max} is the maximum eigenvalue of R

Adaptive Filter ADALINE:

$$a(k) = purelin(\mathbf{W}\mathbf{p}(k) + b) = \sum_{i=1}^{R} \mathbf{w}_{1,i} y(k - i + 1) + b$$

Backpropagation Algorithm:

Performance Index:

Mean Square error: $F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t} - \mathbf{a})^T(\mathbf{t} - \mathbf{a})]$

Approximate Performance Index: (single sample)

$\hat{F}(x) = \mathbf{e}^T(k)\mathbf{e}(k) = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k))$

Sensitivity: $\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left[\frac{\partial \hat{F}}{\partial n_1^m} \quad \frac{\partial \hat{F}}{\partial n_2^m} \quad \cdots \quad \frac{\partial \hat{F}}{\partial n_{s^m}^m}\right]^T$

Forward Propagation: $\mathbf{a}^0 = \mathbf{p}$,

$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1})$ *for* $m = 0, 1, \ldots, M-1$

$\mathbf{a} = \mathbf{a}^M$

Backward Propagation: $\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$,

$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}$ *for* $m = M-1, \ldots, 2, 1$, where

$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \text{diag}([\dot{f}^m(n_1^m)\ \ \dot{f}^m(n_2^m)\ \ \ldots\ \ \dot{f}^m(n_{s^m}^m)])$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

Weight Update (Approximate Steepest Descent):

$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T$

$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$

***Heuristic Variations of Backpropagation:**

Batching: The parameters are updated only after the entire training set has been presented. The gradients calculated for each training example are averaged together to produce a more accurate estimate of the gradient.(If the training set is complete, i.e., covers all possible input/output pairs, then the gradient estimate will be exact.)

Backpropagation with Momentum (MOBP):

$\Delta\mathbf{W}^m(k) = \gamma\Delta\mathbf{W}^m(k-1) - (1-\gamma)\alpha\,\mathbf{s}^m(\mathbf{a}^{m-1})^T$

$\Delta\mathbf{b}^m(k) = \gamma\Delta\mathbf{b}^m(k-1) - (1-\gamma)\alpha\,\mathbf{s}^m$

Variable Learning Rate Backpropagation (VLBP)

1. If the squared error (over the entire training set) increases by more than some set percentage ζ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $\rho < 1$, and the momentum coefficient γ (if it is used) is set to zero.
2. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If γ has been previously set to zero, it is reset to its original value.
3. If the squared error increases by less than ζ, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged.

Association: $\mathbf{a} = hardlim(\mathbf{W}^0\mathbf{P}^0 + \mathbf{W}\mathbf{p} + b)$

An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

Associative Learning Rules:

Unsupervised Hebb Rule:

$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\,\mathbf{a}(q)\mathbf{p}^T(q)$

Hebb with Decay:

$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha\,\mathbf{a}(q)\mathbf{p}^T(q)$

Instar: $\mathbf{a} = hardlim(\mathbf{W}\mathbf{p} + b)$, $\mathbf{a} = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$

The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\| cos\theta \geq -b$

where θ is the angle between $\mathbf{p}$ and ${}_1\mathbf{w}$.

Instar Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$

${}_i\mathbf{w}(q) = (1-\alpha)\,{}_i\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$, *if* $(a_i(q) = 1)$

Kohonen Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$ *for* $i \in X(q)$

Outstar Rule: $\mathbf{a} = satlins(\mathbf{W}p)$

$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha\,\left(\mathbf{a}(q) - \mathbf{w}_j(q-1)\right)p_j(q)$

Competitive Layer: $\mathbf{a} = compet(\mathbf{W}\mathbf{p}) = compet(\mathbf{n})$

Competitive Learning with the Kohonen Rule:

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha\,\left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)\right)$

$= (1-\alpha)\,{}_{i^*}\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1)$, $i \neq i^*$ where i^* is the winning neuron.

Self-Organizing with the Kohonen Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,\left(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)\right)$

$= (\mathbf{1}-\alpha)\,{}_i\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$, $i \in N_{i^*}(d)$

$N_i(d) = \{j, d_{i,j} \leq d\}$

LVQ Network: $(w_{k,i}^2 = 1) \Rightarrow$ subclass i is a part of class k

$n_i^1 = -\|{}_i\mathbf{w}^1 - \mathbf{p}\|$, $\mathbf{a}^1 = compet(\mathbf{n}^1)$, $\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$

LVQ Network Learning with the Kohonen Rule:

${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha\,\left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)\right)$,

if $a_{k^*}^2 = t_{k^*} = 1$

${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha\,\left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)\right)$,

if $a_{k^*}^2 = 1 \neq t_{k^*} = 0$

hardlim: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$, ***hardlims***: $a = \begin{cases} -1 & n < 0 \\ +1 & n \geq 0 \end{cases}$, ***purelin***: $a = n$, ***Logsig***: $a = \frac{1}{1+e^{-n}}$, ***tansig***: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, ***poslin***: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$,

compet: $a = \begin{cases} 1 & \text{neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$, ***satlin***: $a = \begin{cases} 0 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$, ***satlins***: $a = \begin{cases} -1 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$,

Delay: $a(t) = u(t-1)$, ***Integrator***: $a(t) = \int_0^t u(\tau)d\tau + a(0)$

****HINT:**

$diag([1\ 2\ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

ΑΝΑΦΟΡΕΣ – REFERENCES-BIBLIOGRAPHY- ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Τεχνητά Νευρωνικά Δίκτυα, Κωνσταντίνος Διαμαντάρας Εκδόσεις Κλειδάριθμος

[2] Νευρωνικά Δίκτυα και Εφαρμογές , Τσουχνικά Μαρία , Επιβλέπων: Δρ Γ. Βουγιατζής

[3] Τεχνητά Νευρωνικά Δίκτυα και Εφαρμογές στα Συστήματα Αυτομάτου Ελέγχου, Διπλωματική Εργασία Θεοδώση – Κοκκινου Λαουρα, Επιβλέπων: Πέτρος Γρουμπός

[4] Νευρωνικά Δίκτυα , Σημειώσεις Θεωρίας Μαθήματος Δρ. Αντώνιο Ι. Κωνσταντάρα

[5] Σύγκριση Μεθόδων Εκπαίδευσης Τεχνητών Νευρωνικών Δικτύων, Σταθοπούλου Δήμητρα , Επιβλέπων: Πλαγιανάκος Βασίλειος

[6] Βελτιστοποίηση Βραχυπρόθεσμης Πρόβλεψης Ζήτησης Φορτίου με χρήση Συστάδας Νευρωνικών Δικτύων, Α. Ν. Αδαμάκος, Επιβλέπων: Μ. Τίτσιας

[7] Τεχνικές Ενισχυτικής Μάθησης σε Πολυπρακτορικά Συστήματα, Γεωργιος Ι. Μπουλούγαρης

[8] Νευρωνικά Δίκτυα και Μηχανική Μάθηση, Simon Haykin ,Εκδόσεις Παπασωτηρίου

[9] Εκπαίδευση Συνελικτικών Νευρωνικών Δικτύων με προκαθωρισμένα φίλτρα Gabor, Διπλωματική Εργασία Δαιο Δημήτριο , Επιβλέπων: Θεοχάρης Ι , Ζήκος Ν.

[10] Συνελικτικά Νευρωνικά Δίκτυα Στην Υπολογιστική Οραση, Διπλωματική Εργασία Παπαδόπουλος Αθανάσιος Πανεπιστήμιο Πατρών

[11] Προγραμματισμός και Αρχιτεκτονική Συστημάτων Παράλληλης Επεξεργασίας, Στυλιανός Παπαδάκης, Κωνσταντίνος Διαμαντάρας Εκδόσεις Κλειδάριθμος

[12] Προγραμματισμός Μαζικά Παράλληλων Επεξεργαστών ,David B. Kirk, Wen-mei W. Hwu ,Εκδόσεις Κλειδάριθμος

[16] CHALMERS ,Implementation of Deep Feedforward Neural Network with CUDA backend for Efficient and Accurate Natural Image Classification, Convolutional Neural Network Applications , Master's thesis in Complex Adaptive Systems,AUGUST VON HACHT, Department of Physics , CHALMERS UNIVERSITY OF TECHNOLOGY , GOTHENBURG SWEDEN 2017

[17] Neural Network for Recognition of Handwritten Digits -CodeProject

[18] THE MNIST DATABASE OF HANDWRITTEN DIGITS ,Yann LeCun Courant institute, NYU, Corinna Cortes Google Labs, New York , Christopher J.C. Burges Microsoft Research Redmond.

[19] Neural Networks Matlab examples. Deep Learning with Matlab and Multiple GPUs. Practical Deep Learning Examples with Matlab.

[20] Yann LeCun , Leon Botton , Yoshua Bengio , and Patrick Haffner .Gradient – based Learning applied to document recognition .Proceedings of the IEEE, 86 (11) :2278 – 2324, 1998. Various Papers of Yann Le Cun about Handwritten Recognition with Neural Convolutional Networks.

ΣΗΜΕΙΩΣΕΙΣ ΚΑΙ ΠΑΡΑΠΟΜΠΕΣ

1. *Ο Muller κ.α. (1998) έχει μελετήσει την εφαρμογή του βασιζόμενου στην μέθοδο της ανόπτησης on – line αλγόριθμου μάθησης σε ενα πρόβλημα τυφλού διαχωρισμού σημάτων απο μια πηγή, γεγονός το οποίο επιδεικνύει την ευρεία εφαρμοστικότητα των αλγόριθμων προσαρμοστικού ελέγχου του ρυθμού μάθησης όπως παρουσιάζονται απο τον Murata (1998), [8].*
2. *Η πρώτη τεκμηριωμένη περιγραφή της χρήσης της μεθόδου οπισθοδιάδοσης για τον αποτελεσματικό υπολογισμό κλίσεων αποδίδεται στον Werbos (1974) [8].*
3. *Οι σιγμοειδείς συναρτήσεις δίνουν γραφήματα σχήματος <<S>> Ο Mennon κ.α. (1996) παρουσιάζει μια λεπτομερή μελέτη δυο κατηγοριών σιγμοειδών συναρτήσεων [8]:*
    - *Απλές σιγμοειδείς , οι οποίες ορίζονται ως περιττές, ασυμπτωτικά φραγμένες και απολύτως μονοτονικές συναρτήσεις μιας μεταβλητής*
    - *Υπερβολικές σιγμοειδείς, οι οποίες αντιπροσωπεύουν ενα γνήσιο υποσύνολο των απλών σιγμοειδών και μια φυσιολογική γενίκευσης της συνάρτησης υπερβολικής εφαπτόμενης.*

ΜΕΡΟΣ Β ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ ΜΕΡΟΥΣ Β ΘΕΩΡΙΑ ΠΑΝΩ ΣΕ ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

ΚΕΦΑΛΑΙΟ 1 ΓΕΝΙΚΑ ΓΙΑ ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ Σελίδα 101

1.1 Κατάταξη υπολογιστών αρχιτεκτονικών Σελίδα 102

1.2 Ταξινόμηση με βάση την κατανομή της μνήμης, πολυεπεξεργαστές και πολυ-υπολογιστές Σελίδα 108

1.3 Πολυνημάτωση Σελίδα 114

1.4 GPU Σελίδα 115

1.5 CUDA Σελίδα 131

ΑΝΑΦΟΡΕΣ Σελίδα 153

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ ΓΕΝΙΚΑ ΓΙΑ ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Απο την αρχή της δημιουργίας των Ηλεκτρονικών Υπολογιστών οι μηχανικοί σχεδίασης υπολογιστών κοιτούσαν για την δημιουργία πανίσχυρων υπολογιστών με την σύνδεση πολλών μικροτέρων μεταξύ τους. Αυτοί είναι οι πολυεπεξεργαστές, όπου το λογισμικό αυτών πρέπει να σχεδιάζεται ώστε να δουλεύει με μεταβλητό αριθμό επεξεργαστών. Το κυρίαρχο θέμα είναι η ηλεκτρική ισχύς . Εαν αντικαταστήσουμε τους μεγάλους και μη αποδοτικούς επεξεργαστές με πολλούς και μικρότερους και αποδοτικούς τότε μπορούμε να έχουμε καλύτερη απόδοση ,βέβαια αυτό θα εξαρτηθεί και απο το λογισμικό που θα τους χρησιμοποιήσει αποδοτικά.

Υψηλή απόδοση μπορεί να σημαίνει υψηλή ικανότητα διεκπαιραίωσης για ανεξάρτητες εργασίες που ονομάζονται παραλληλία επιπέδου εργασίας (job level parallelism) ή παραλληλία επιπέδου διεργασίας (process level parallelism) [13]. Οι παράλληλες εργασίες είναι ανεξάρτητες εφαρμογές και χρησιμοποιούνται απο παράλληλους υπολογιστές. Εδώ θα συναντήσουμε και τον όρο πρόγραμμα παράλληλης επεξεργασίας (parallel processing program) το οποιο και αναφέρεται σε ένα μοναδικό πρόγραμμα που εκτελείται ταυτόχρονα απο πολλαπλούς επεξεργαστές [13].

Μην ξεχάσουμε να αναφέρουμε και την (clusters) συστοιχία όπου και αποτελείται απο ένα σύνολο μικροεπεξεργαστών οι οποίοι φιλοξενούνται σε πολλούς και ανεξάρτητους διακομιστές [13]. Μπορούν αυτές οι συστοιχίες να τρέξουν πέρα απο επιστημονικές εφαρμογές ,μηχανές αναζήτησης, βασεις δεδομένων,διακομιστές ιστου.

Επειδή το πρόβλημα της ηλεκτρικής ισχύς το βρίσκουμε μπροστά μας μιας και οι αυξήσεις σε απόδοση προέρχονται απο περισσότερους επεξεργαστές ανά τσιπ και όχι απο τους υψηλότερους ρυθμούς ρολογιού. Αυτοί οι επεξεργαστές ονομάζονται πολυπύρηνοι μικροεπεξεργαστές . Αυτοί οι επεξεργαστές για να μπορούνε να δουλέψουν αποδοτικά θέλουν προγράμματα παράλληλης επεξεργασίας και όχι σειριακά η ακολουθιακά που είναι πολύ αργά στο τρέξιμο.

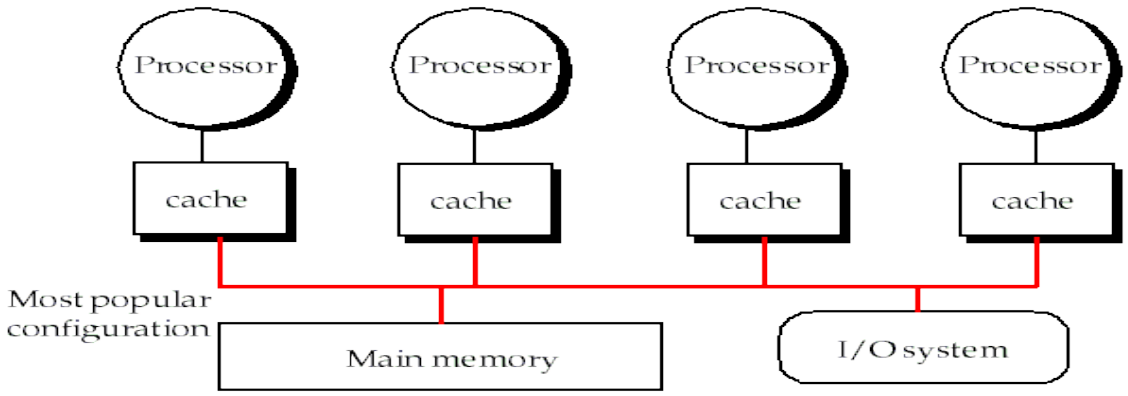
Για τους σχεδιαστές λογισμικού υπήρχε το εξής πρόβλημα ,αν θα μπορούσαν να ανανεώσουν τα παλιά προγράμματα και να τρέξουν καλά στο παράλληλο υλικό. Βρέθηκε μια λύση και αυτή είναι , να παρέχεται ενα μοναδικός χώρος φυσικών διευθύνσεων που να μπορούν να μοιραστούν όλοι οι επεξεργαστές, οπότε με αυτό τον τρόπο τα προγράμματα δεν χρειάζονται να απασχολούνται με το πού εκτελούνται αλλά να εκτελούνται παράλληλα [13] . Με αυτόν τον τρόπο πετυχαίνουμε όλες οι μεταβλητές ενός προγράμματος να έιναι διαθέσιμες οποιαδήποτε στιγμή σε οποιαδήποτε επεξεργαστή.

Μια αλλη προσέγγιση είναι να υπάρχει ενας ξεχωριστός χώρος διευθύνσεων ανά επεξεργαστή [13].

Στην περίπτωση που ο χώρος φυσικών διευθύνσεων είναι κοινός όπως συμβαίνει στα πολυπύρηνα τσιπ ,το υλικό παρέχει συνοχή κρυφής μνήμης (cache coherence) για να δώσει μια συνεπή εικόνα της κοινόχρηστης μνήμης [13].

1.1 Κατάταξη υπολογιστών αρχιτεκτονικών

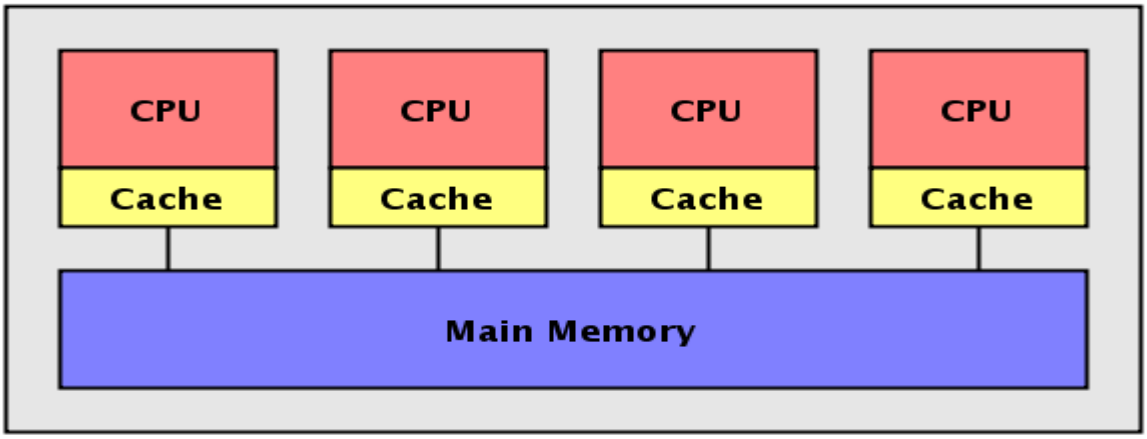
Το πώς θα διασυνδέσουμε πολλές μονάδες επεξεργασίας για να λύσουμε ενα πρόβλημα, μπορεί να γίνει με πολλούς τρόπους επιλέγοντας διάφορες παράλληλες αρχιτεκτονικές. Οι εντολές που εκτελούνται ταυτόχρονα μπορεί να είτε ίδιες είτε διαφορετικές [11]. Επίσης η μνήμη μπορεί να είναι είτε κοινόχρηστη ,όπου έχουμε ομοιόμορφη πρόσβαση στα δεδομένα ανεξάρτητα απο την διεύθυνση, είτε κατανεμημένη οπότε δεν έχουμε ομοιόμορφη πρόσβαση [11], ένας πολυεπεξεργαστής κοινόχρηστης μνήμης (shared memory multiprocessor – SMP) δίνει στον προγραμματιστή ένα μοναδικό χώρο φυσικών διευθύνσεων για όλους τους επεξεργαστές [13]. Επίσης τέτοια συστήματα μπορούν να εχουν δικούς τους χώρους διευθύνσεων για να εκτελούν δικές τους ανεξάρτητες εργασίες. Σύμφωνα με τις παρακάτω εικόνες ο κάθε επεξεργαστής έχει προσπέλαση στην δικιά του κρυφή μνήμη και στην κοινόχρηστη έχουν προσπέλαση όλοι οι επεξεργαστές.



Σχήμα 1.1.1

πηγή

http://ece-research.unm.edu/jimp/611/slides/chap8_1.html



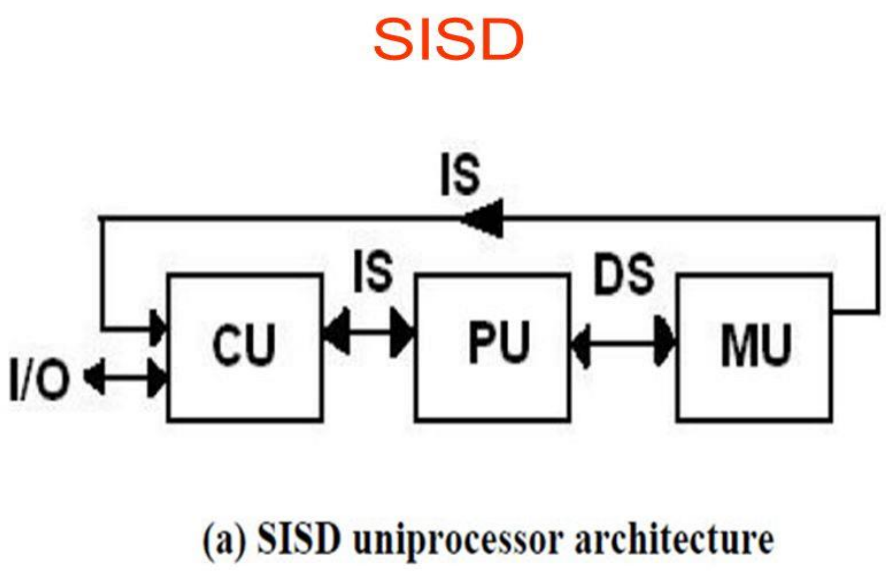
Σχήμα 1.1.2 Ταξινόμηση με βάση τις ακολουθίες εντολών και δεδομένων

πηγή

https://www.cs.rit.edu/~ark/lectures/pj04/notes.shtml

Η επόμενη ταξινόμηση των υπολογιστικών συστημάτων γίνεται με βάση το πόσα διαφορετικά ρεύματα εντολών (instruction streams) εκτελούνται ταυτόχρονα σε πόσα ρεύματα δεδομένων (data streams) [11]. Οι υπολογιστές ανήκουν σε μια απο τις παρακάτω κατηγορίες:

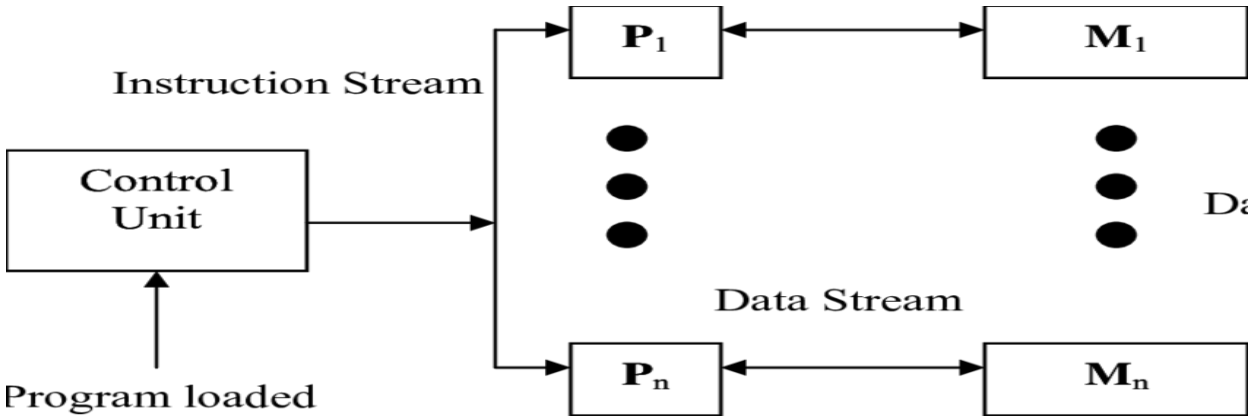
1. SISD (Single Instruction stream, Single Data stream – Ενα ρεύμα εντολών , ένα ρεύμα δεδομένων). Αυτού του τύπου ο υπολογιστής εκτελεί μια μόνο εντολή κάθε χρονική στιγμή, σε ένα δεδομένο [11]. Είναι μονοπύρηνος επεξεργαστής. Υπάρχει μια μονάδα ελέγχου (Control Unit -CU) που δέχεται εντολές και εξυπη-ρετεί την είσοδο και την έξοδο. Μετά η εντολή που βγαίνει απο την μονάδα ελέγ-χου πάει στην μονάδα επεξεργασίας (Processing Unit – PU) που θα εκτελεστεί. Η μονάδα επεξεργασίας χρησιμοποιεί δεδομένα απο την μονάδα μνήμης (Me-mory Unit – MU) η οποία επίσης μεταβιβάζει εντολές στην μονάδα ελέγχου [11].
2. SIMD (Single Instructions stream, Multiple Data stream – Ενα ρεύμα εντολών , πολλά ρεύματα δεδομένων). Σε αυτόν τον τύπο υπάρχουν πολλές μονάδες επεξεργασίας,που οι οποίες εκτελούν την ίδια εντολή. Η εντολή προέρχεται απο την μοναδική μονάδα ελέγχου (Control Unit) που υπάρχει στο σύστημα [11]. Κάθε μονάδα επεξεργασίας έχει κάποια τοπική μνήμη (Local Memory – LM) όπου απο την οποία αντλεί τα δεδομένα στα οποία θα εκτελεστεί η συγκεκριμένη εντολή [11].



Σχήμα 1.1.3

πηγή

https://slideplayer.com/slide/8353953/

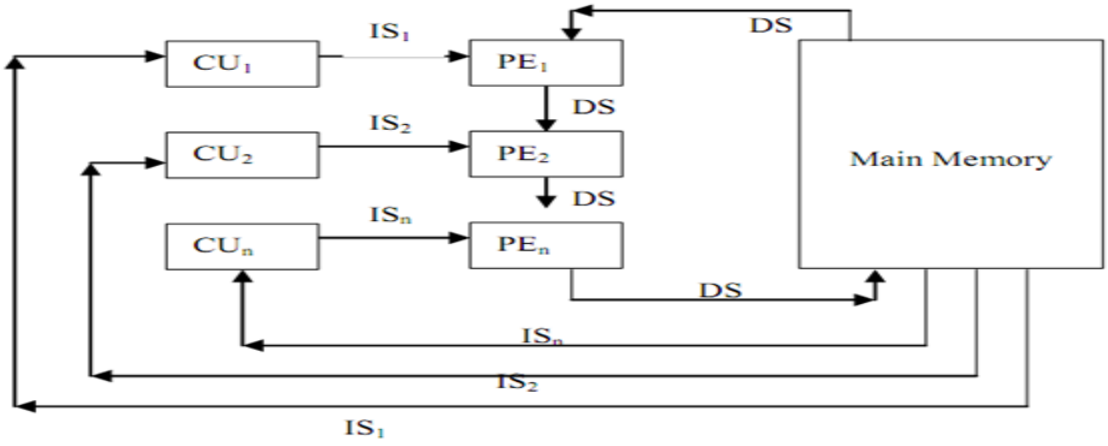


(b) SIMD uniprocessor architecture

Σχήμα 1.1.4

πηγή
https://www.researchgate.net/figure/SIMD-architecture_fig8_268041882

3. MISD (Multiple Instruction streams, Single Data stream – πολλές εντολές, ένα ρεύμα δεδομένων). Το μοντέλο αυτό προσομοιώνει την λειτουργία της διοχέτευσης , εδώ έχουμε μια σειρά από μονάδες επεξεργαστών όπου η κάθε μια έχει τον δικό της ελεγκτή που εκτελεί διαφορετική εντολή απο τις άλλες. Συγκρίνοντας το MISD με το MIMD που θα περιγραφτεί πιο μετά .προκύπτει ότι στο μοντέλο που περιγράφουμε τώρα το ρεύμα δεδομένων είναι ένα. Τα δεδομένα πηγάζουν απο την κύρια μνήμη, όπου τα επεξεργάζεται η πρώτη μονάδα επεξεργασίας, που το αποτέλεσμα της πράξης πάει στην μονάδα 2 και αυτή με την σειρά της το πάει στην 3 και πάει λέγοντας [11].



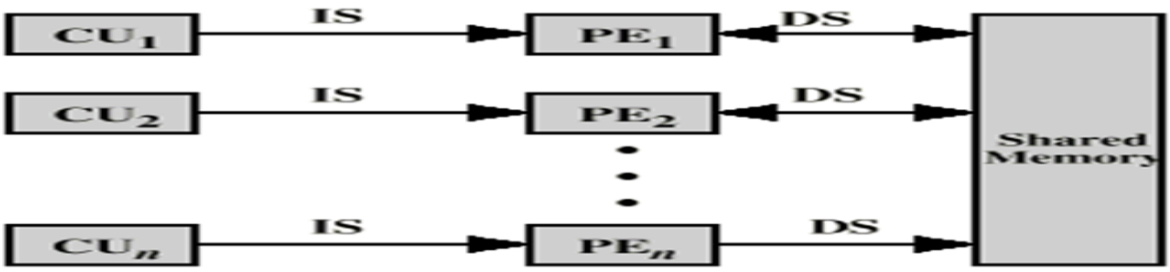
(c) MISD uniprocessor architecture

Σχήμα 1.1.5

πηγή

http://www.expertsmind.com/questions/multiple-instruction-and-single-data-stream-misd-30138149.aspx

4. MIMD (Multiple Instruction streams, Multiple Data streams – Πολλές εντολές, πολλά ρεύματα δεδομένων). Εδώ έχουμε πολλές μονάδες ελέγχου όπου ξεκινούν διαφορετικές εντολές ,επίσης έχουμε πολλές μονάδες επεξεργασίας ,μια για κάθε μονάδα ελέγχου. Οι μονάδες επεξεργασίας μπορούν να επικοινωνούν είτε με κοινόχρηστη μνήμη είτε να έχουν τοπικές μνήμες μέσω κάποιας διασύνδεσης. Είναι σαν έχουμε πολλούς υπολογιστές που λειτουργούν νε μεν ανεξάρτητα μεταξύ τους αλλά απο την άλλη μπορούν και να συνεργάζονται μεταξύ τους εφόσον προκύπτει κάποιο δύσκολο πρόβλημα.

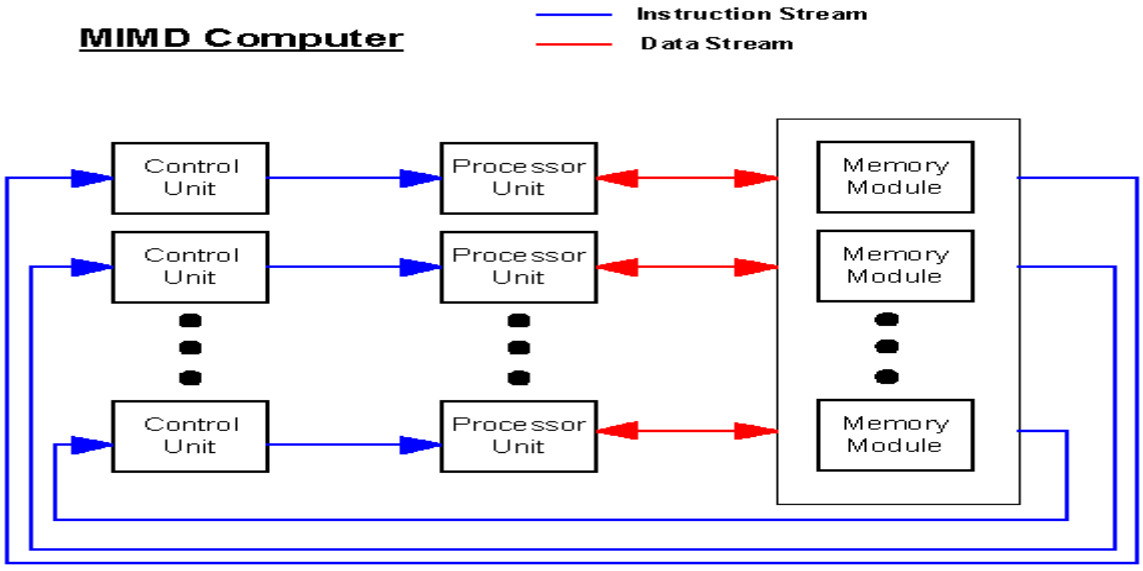


(d) MIMD uniprocessor architecture with shared memory

Σχήμα 1.1.6

πηγή

https://jessamaecasanoba.wordpress.com/2014/10/12/parallel-processing/



(d) MIMD uniprocessor architecture with shared or local memory

Σχήμα 1.1.7

πηγή

http://sistemasdistribuidos-ja7.blogspot.com/2016/03/metodos-de-informacion.html

Οι μηχανές SIMD χρησιμοποιούνται συνήθως ως επιταχυντές ειδικά για εκτέλεση κάποιων ειδικών λειτουργιών όπου αυτές απαιτούνται απο τις εφαρμογές. Για παράδειγμα σε multimedia εφαρμογές όπου έχουμε εικόνα ,ήχο,γραφικά ,επαναλαμβάνεται η ίδια εντολή σε πολλά δεδομένα.

Σήμερα οι υπολογιστές τύπου SIMD βρίσκονται:

(α) προσαρτημένοι σε ένα κύριο υπολογιστή σε μορφή κάρτας επιταχυντή (accelarator card) για εκτέλεση είτε συγκεκριμένων πράξεων (πρόσθεση,αφαίρεση, πολ/σμό ,μετακίνηση κ.λ.π.) είτε για εκτέλεση αλγορίθμων με την μορφή συστολικής επεξεργασίας. Ο κεντρικός υπολογιστής μπορεί να στείλει εντολές για να εκτελεστούν στην κάρτα SIMD όταν απαιτείται η εκτέλεση αυτών των συγκεκριμένων πράξεων ή αλγορίθμων [11].

(β) ενσωματωμένοι σε έναν επεξεργαστή για την εκτέλεση συγκεκριμένων ομάδων εντολών ,όπως εντολές που αφορού πολυμεσική επεξεργασία [11].

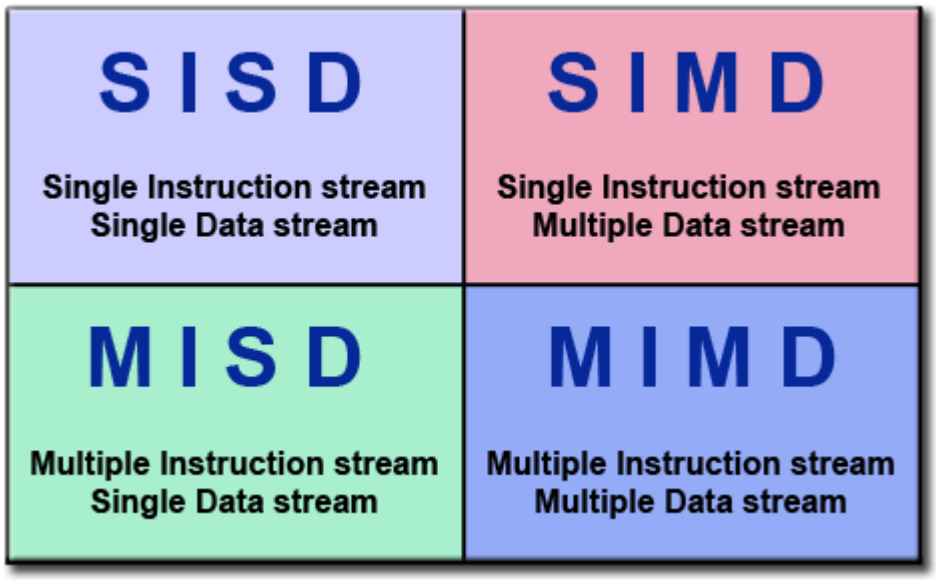
Το μοντέλο MISD έχει την λογική της διοχέτευσης (pipeline) . Στην διοχέτευση μια εργασία διαιρείται σε βήματα , οι επεξεργαστές μετά που ειδικοί σε αυτό, εκτελούν ένα συγκεκριμένο βήμα , για παράδειγμα ο επεξεργαστής E1 θα εκτελεί το βήμα B1 και πάει λέγοντας. Μόλις ο επεξεργαστής E1 τελειώσει το βήμα B1 της εργασίας T1 μετά θα έρθει η σειρά του επεξεργαστή E2 να ξεκινήσει το βήμα B2 της εργασίας T1 ,ενώ ο E1 ασχολείται με το βήμα B1 της εργασίας T2. Απο κάποια στιγμή και μετά όλοι οι επεξεργαστές είναι απασχολημένοι σες διαφορετικές εργασίες και έτσι επιτυγχάνεται παράλληλη και ταχύτερη εκτέλεση των εργασιών [11].

Στην πράξη αυτή η αρχιτεκτονική χρησιμοποιείται σε συγκεκριμένες υπομονάδες μέσα στους υπολογιστές και είναι βασική αρχιτεκτονική στο εσωτερικό του επεξεργαστή. Οι επεξεργαστές MIMD είναι γενικού σκοπού διότι μπορούν να χρησιμοποιηθούν στην εκτέλεση οποιαδήποτε παράλληλης εφαρμογής [11].. Εδώ μπαίνουν και οι συστοιχίες (clusters) διότι αποτελούνται απο πολλές μονάδες τύπου SISD που συνδεδεμένες σε δίκτυο.

Το μοντέλο MIMD όσον αφορά την εξέλιξή του οφείλεται σε δυο λόγους:

(α) Οι μηχανές MIMD προσφέρουν ευελιξία αυτό διότι μπορούνε είτε να χρησιμοποιηθούνε σαν υπολογιστές ενός χρήστη με δυνατότητα επιτάχυνσης των εφαρμογών, είτε μπορούν να χρησιμοποιηθούνε σαν πολυπρογραμματιζόμενες μηχανές που θα μπορούν να εκτελούν πολλές διεργασίες ταυτόχρονα.

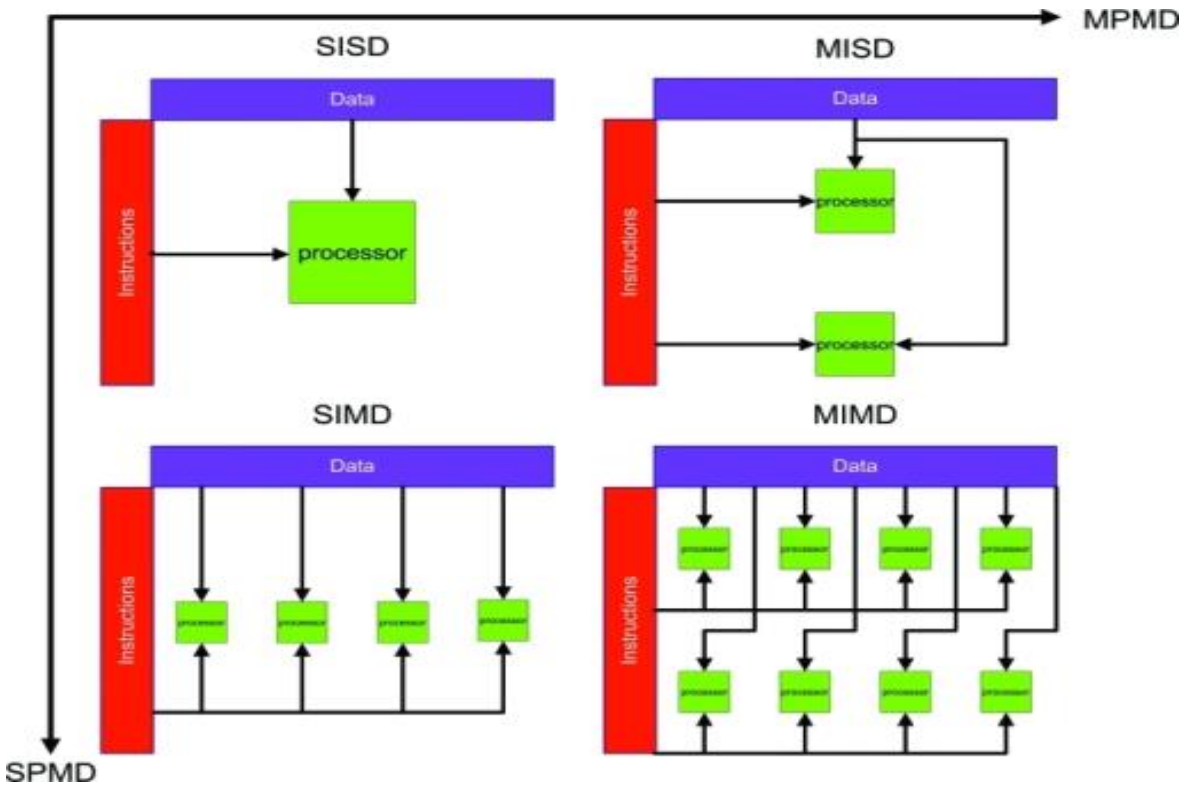
(β) Οι μηχανές τύπου MIMD μπορούν να κατασκευαστούν απο κοινούς επεξεργαστές του εμπορίου [11]. Οι περισσότεροι παράλληλοι υπολογιστές τύπου MIMD που υπάρχουν στην αγορά σήμερα χρησιμοποιούν τέτοιου τύπου επεξεργαστές [11].



Σχήμα 1.1.8

πηγή

https://www.oreilly.com/library/view/build-supercomputers-with/9781787282582/ch01s02.html



Σχήμα 1.1.9

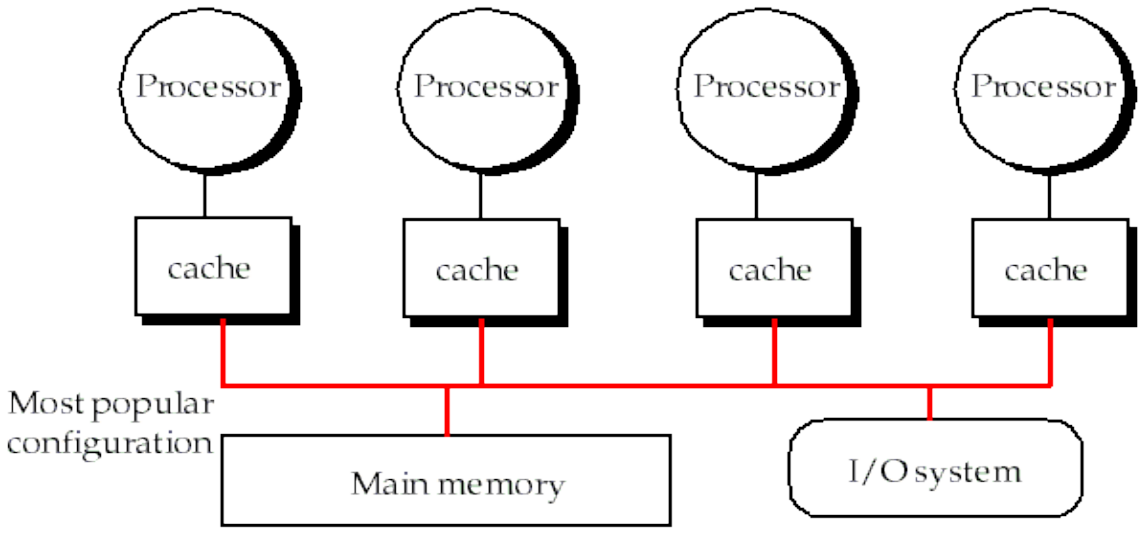
πηγή

https://www.picswe.com/pics/simd-sisd-83.html

1.2 Ταξινόμηση με βάση την κατανομή της μνήμης, πολυεπεξεργαστές και πολυ-υπολογιστές

Οι μηχανές MIMD με βάση την οργάνωση της μνήμης ταξινομούνται σε δυο αρχιτεκτονικές: (α) αυτούς με την κοινόχρηστη μνήμη (shared memory) και (β) αυτούς με την κατανεμημένη μνήμη (distributed memory). Οι μεν πρώτοι ονομάζονται πολυεπεξεργαστές (multiprocessors) και οι δεύτεροι ονομάζονται πολυ-υπολογιστές (multicomputers).

Ο πολυεπεξεργαστής αποτελείται απο λίγους επεξεργαστές συνήθως 16 το αριθμό και είναι απο τα πιο διαδεδομένα μοντέλα στην αγορά. Αυτοί οι επεξεργαστές συνδέονται με την κεντρική (κοινόχρηστη) μνήμη και μεταξύ τους με ένα δίκτυο τύπου διάυλου (bus) [11]. Υπάρχει και ένας άλλος τύπος, αυτός όπου είναι πολλοί δίαυλοι ιεραρχημένοι ή δίκτυο πλέγματος (crossbar) όπου έχουμε σύνδεση κάθε επεξεργαστή με ξεχωριστή γραμμή με καθέναν απο τους υπόλοιπους. Και επειδή όλοι οι επεξεργαστές έχουν ομοιόμορφη πρόσβαση σε όλες τις διευθύνσεις της μνήμης ,αυτό το μοντέλο ονομάζεται Ομοιόμορφη Προσπέλαση Μνήμης (Uniform Memory Access – UMA) [11]. Για οποιαδήποτε διεύθυνση η διαδικασία προσπέλασης της μνήμης είναι η ίδια: δηλαδή ο επεξεργαστής υποβάλλει αίτημα στον δίαυλο για εγγραφή ή για ανάγνωση και περιμένει την ολοκλήρωση της εργασίας απο την μνήμη [11]. Ο χρόνος απόκρισης είναι ο ίδιος ανεξάρτητα απο την ακριβή διεύθυνση των δεδομένων [11].

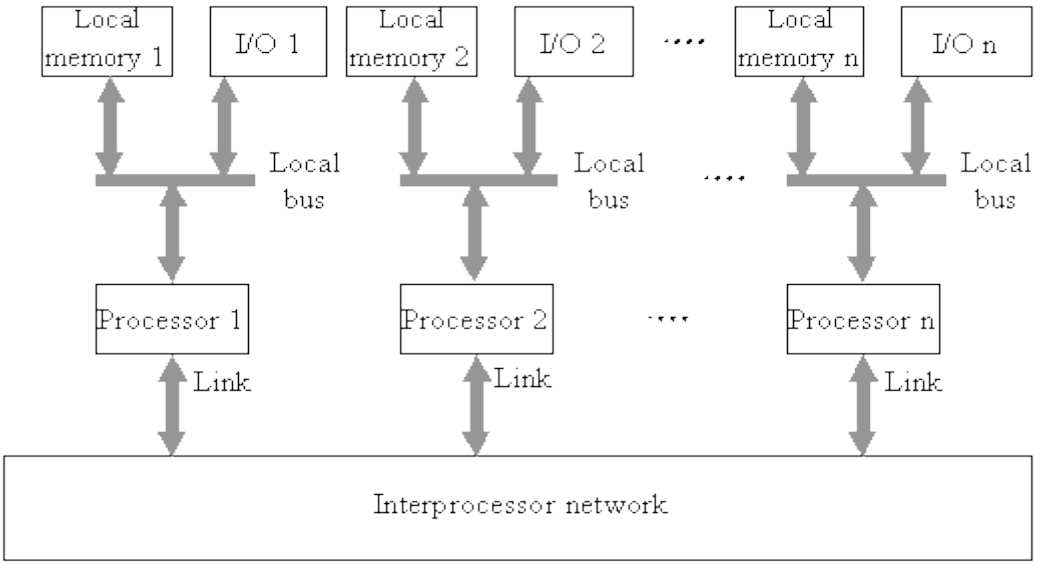


Σχήμα 1.2.1 Η δομή ενός πολυεπεξεργαστή με κοινόχρηστη κεντρική μνήμη

πηγή

http://ece-research.unm.edu/jimp/611/slides/chap8_1.html

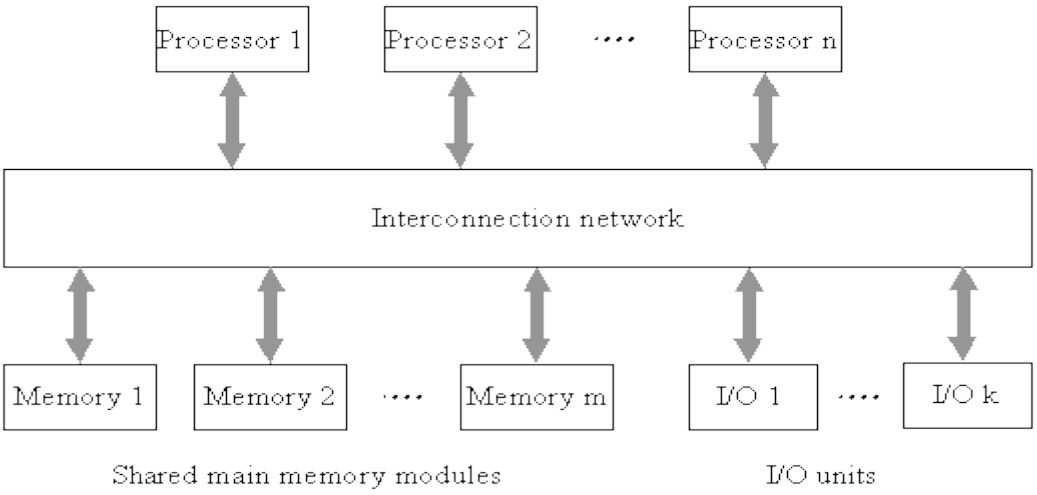
Το βασικό μοντέλο ενός πολυ – υπολογιστή αποτελείται απο έναν αριθμό επεξεργαστών όπου ο καθένας απο αυτούς έχει την δική του τοπική μνήμη και επίσης διαθέτει την δική του είσοδο και έξοδο. Οι επεξεργαστές συνδέονται μεταξύ τους με ένα δίκτυο διασύνδεσης (interconnection network), η τοπολογία του οποίου είναι απο τα κεντρικά αντικείμενα έρευνας στην τεχνολογία των παράλληλων αρχιτεκτονικών [11].



Σχήμα 1.2.2 Η δομή ενός πολύ – υπολογιστή κατανεμημένης μνήμης

Πηγή

https://edux.pjwstk.edu.pl/mat/264/lec/main119.html



Σχήμα 1.2.3

Πηγή

https://edux.pjwstk.edu.pl/mat/264/lec/main119.html

Η χωρική κατανομή της μνήμης στους επεξεργαστές έχει δύο βασικά πλεονεκτήματα:[11].

(α) Αυξάνει τον όγκο των δεδομένων που μπορούν να προσπελαστούν σε ένα συγκεκριμένο χρονικό διάστημα [11]. Και αυτός ο όγκος λέγεται εύρος ζώνης μνήμης (memory bandwidth) και μετράται σε bits/sec. Εάν για παράδειγμα έχουμε μια κεντρική μνήμη με χρόνο προ-σπέλασης 4 ns και δίαυλο δεδομένων (data bus) των 32 bit , οπότε η μνήμη αυτή μπορεί να αποδώσει 32 bit δεδομένων ανά 4 ns , δηλαδή έχει εύρος ζώνης 32 / (4 * 10^-9) = 8 Gbit/sec [11]. Αν αυτή η μνήμη εξυπηρετεί Ν επεξεργαστές, όλοι οι επεξεργαστές μαζί εξυπηρετούνται απο το εύρος ζώνης των 8 Gbit/sec [11]. Σε περίπτωση που η μνήμη κοπεί σε κομμάτια και κάθε επεξεργαστής έχει το δικό του τοπικό κομμάτι, τότε ο καθένας απο αυτούς μπορεί να εξυπηρετηθεί με 8 Gbit/sec και μετά να πολλαπλασιαστεί κατά Ν φορές συνολικά ο όγκος των προσπελάσιμων πληροφοριών [11].

(β) Εχουμε μείωση του χρόνου προσπέλασης της μνήμης και δεν χρειάζεται πλέον διαιτησία για να χειριστούμε την περίπτωση που δυο επεξεργαστές θέλουν να προσπελάσουν ταυτόχρονα την ίδια μνήμη, που συμβαίνει στο μοντέλο της κοινόχρηστης μνήμης.

Το μειονέκτημα του μοντέλου αυτού είναι ότι επιβαρύνεται η επικοινωνία και αυτό διότι δεν μπορούν να ανταλλάξουν δεδομένα μέσα απο την κοινόχρηστη μνήμη. Οπότε και αναγκάζονται οι επεξεργαστές να ανταλλάσουν μηνύματα μέσω του δικτύου διασύνδεσης. Επίσης το δίκτυο πέρα απο την πολυπλοκότητα που έχει προσθέτει και καθυστέρηση καθώς το μήνυμα θέλει κάποιο χρόνο για να πάει απο τον αποστολέα στον παραλήπτη. Επίσης μια άλλη καθυστέρηση που εισάγεται είναι λόγω της κυκλοφοριακής συμφόρησης του δικτύου.

Η χωρική κατανομή της μνήμης δε σημαίνει ότι υπάρχει και λογική κατανομή της [11]. Η μνήμη λέγεται λογικά κατανεμημένη εάν κάθε επεξεργαστής μπορεί μόνο να εκδώσει διευθύνσεις που προσπελάζουν την δική του τοπική μνήμη [11]. Σε περίπτωση που όλες μαζί οι τοπικές μνήμες δημιουργούν έναν ενιαίο χώρο διευθύνσεων,που είναι προσπελάσιμος απο όλους τους επεξεργαστές , τότε θα έχουμε μια λογικά κοινόχρηστη μνήμη και ας είναι οι τοπικές μνήμες χωρικά κατανεμημένες. Οπότε μετά απο αυτό μπορούμε να πούμε για το μοντέλο της λογικά κατανεμημένης μνήμης και το μοντέλο της λογικά κοινόχρηστης μνήμης.

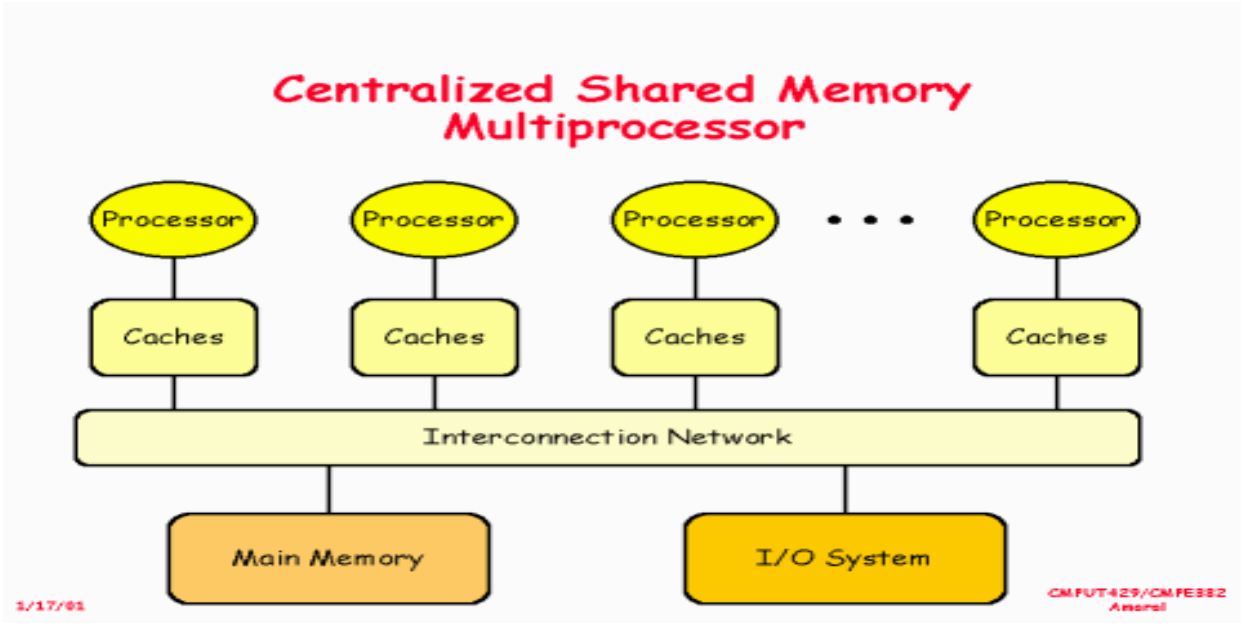
Λογικά κατανεμημένη μνήμη. Κάθε τοπική μνήμη έχει τον δικό της χώρο διευθύνσεων ο οποίος δεν είναι προσπελάσιμος απο τους άλλους επεξεργαστές [11]. Οπότε ο κάθε επεξεργαστής φαίνεται να είναι ο πυρήνας ενός ξεχωριστού υπολογιστή γι αυτό και ονομάζεται πολυ – υπολογιστής. Οι μνήμες που αντιστοιχούν σε κάθε επεξεργαστή έχουν ξεχωριστούς χώρους διευθύνσεων, οπότε η επικοινωνία μεταξύ των επεξεργαστών γίνεται με μεταβίβαση μηνυμάτων [11].

Κοινόχρηστη κατανεμημένη μνήμη. (Distributed Shared Memory – DSM). Οσον αφορά το μοντέλο της κοινόχρηστης μνήμης,νε μεν οι μνήμες αυτές είναι χωρικά κατανεμημέ-

νες αλλά αποτελούν έναν ενιαίο χώρο διευθύνσεων [11]. Μπορεί να ζητήσει μια διεύθυνση ένας απομακρυμένος υπολογιστής για μια τοπική μνήμη ενός άλλου επεξεργαστή. Οι πολυ –υπολογιστές DSM λέγονται και μηχανές Μη Ομοιόμορφης Προσπέλασης Μνήμης (Non Uniform Memory Access – NUMA) .Παρόλο που ο χώρος διευθύνσεων είναι ενιαίος , έχει διαφορετικό χρόνο προσπέλασης για μια κλήση που θα εξυπηρετηθεί απο την τοπική μνήμη και διαφορετικό χρόνο προσπέλασης για μια κλήση η οποία θα εξυπηρετηθεί απο μια απομακρυσμένη μνήμη. Οταν ένας παράλληλος επεξεργαστής έχει κοινό χώρο μνήμης για όλους τους επεξεργαστές, δεν θα διαφέρει προγραμματιστικά απο έναν πολυεπεξεργαστή με κοινόχρηστη μνήμη [11]. Η διαφορά είναι στην αρχιτεκτονική , στον πολυεπεξεργαστή η μνήμη είναι χωρικά συγκεντρωμένη ενώ στον πολυ – υπολογιστή DSM είναι κατανεμημένη χωρικά σε τμήματα που τοποθετούνται κοντά σε κάθε επεξεργαστή [11].

Οι πολυπύρηνοι επεξεργαστές είναι οι επεξεργαστές των οποίων η κεντρική μονάδα επεξεργασίας αποτελείται απο πολλούς πυρήνες (cores) . Ο κάθε πυρήνας είναι ουσιαστικά ένας επεξεργαστής με δική του μονάδα ελέγχου και αποκωδικοποίησης εντολών καθώς και δική του αριθμητική λογική μονάδα (ALU) [11]. Αυτά τα πολυπύρηνα συστήματα δεν ανήκουν στην κατηγορία των συστημάτων SISD αλλά στα παράλληλα συστήματα. Στο επίπεδο της αρχιτεκτονικής οι επιλογές μοιάζουν πολύ με αυτών των παράλληλων συστημάτων, μπορεί να επιλέξει το είδος του δικτύου που θα συνδέει τους πυρήνες διάυλου ή πλέγματος – crossbar ,μπορεί να αποφασίσει τι μοντέλο θα ακολουθήσει SIMD , MIMD , ίσως MIMD αν θα υπάρχει κοινή κρυφή μνήμη (cache) ή ξεχωριστές για κάθε πυρήνα , αν οι πυρήνες θα μπορούν να επικοινωνούν με ανταλλαγή μηνυμάτων.

Ενας πολυεπεξεργαστής κοινόχρηστης μνήμης (shared memory multiprocessor –SMP) παρέχει σε πολλαπλούς επεξεργαστές να επικοινωνούν με κοινή μνήμη , θα μπορούσε κάλιστα να ονομαστεί πολυεπεξεργαστής κοινόχρηστης διεύθυνσης (shared addresss multiprocessor). Τέτοια συστήματα μπορούν να εκτελούν ανεξάρτητες εργασίες στους δικούς τους χώρους εικονικών διευθύνσεων, ακόμη και αν όλοι μοιράζονται ένα χώρο φυσικών διευθύνσεων. [13] Οι επεξεργαστές επικοινωνούν μέσω κοινόχρηστων μεταβλητών στη μνήμη, και είναι όλοι ικανοί να προσπελάσουν οποιαδήποτε θέση μνήμης με την χρήση φορτώσεων (loads) και αποθηκεύσεων (stores).

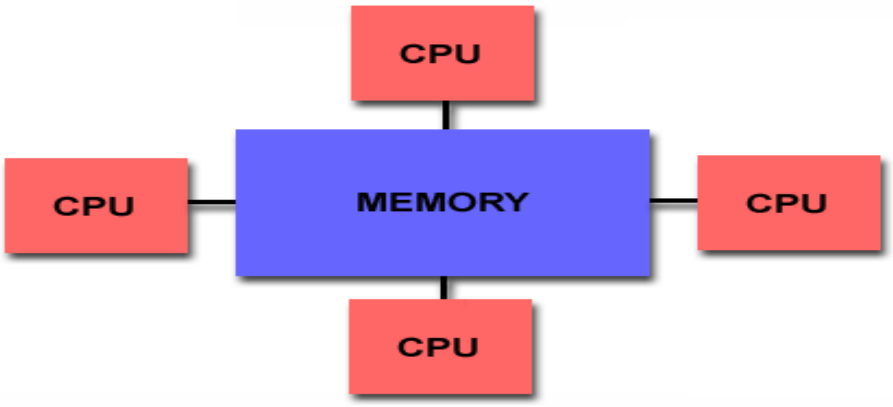


Σχήμα 1.2.4

πηγή

https://webdocs.cs.ualberta.ca/~amaral/courses/429/webslides/TopicB-MultipleProcessors/sld002.htm

Οι πολυεπεξεργαστές κοινόχρηστης μνήμης χωρίζονται σε δυο κατηγορίες , η μια ομοιόμορφης προσπέλασης μνήμης (uniform memory access- UMA). Και η δέυτερη ανομοιόμορφης προσπέλασης μνήμης (nonuniform memory access -NUMA). Πιο εύκολα προγραμματίζεις εναν πολυεπεξεργαστή UMA , αλλά οι NUMA μπορούν να επεκταθούν σε μεγαλύτερα μεγέθη και μπορού να έχουν μικρότερο λανθάνουσα χρόνο προς την γειτονική μνήμη. [13].

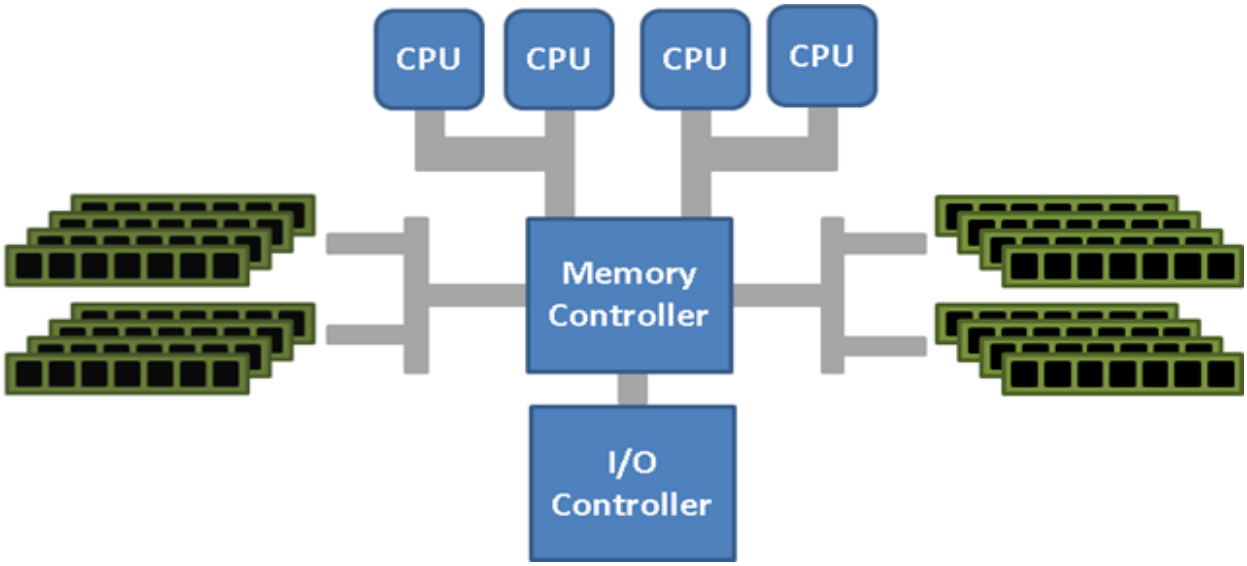


ομοιόμορφης προσπέλασης μνήμης (uniform memory access- UMA)

Σχήμα 1.2.5

πηγή

https://www.researchgate.net/figure/Shared-Memory-Architecture_fig3_259181530



ανομοιόμορφης προσπέλασης μνήμης (nonuniform memory access –NUMA)

Σχήμα 1.2.6

πηγή

https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/

Επειδή οι επεξεργαστές που λειτουργούν παράλληλα μοιράζονται δεδομένα σε κανονικές συνθήκες θα πρέπει και να συντονίζονται όταν χειρίζονται κοινά δεδομένα αλλιώς, ο ένας επεξεργαστής θα μπορούσε να αρχίσε να επεξεργάζεται τα δεδομένα πριν κάποιος άλλος ολοκληρώσει την επεξεργασία τους. [13] Οπότε ο συντονισμός αυτός λέγεται συγχρονισμός. Εφόσον η κοινή χρήση υποστηρίζεται απο ένα μοναδικό χώρο διευθύνσεων , πρέπει να υπάρχει ξεχωριστός μηχανισμός συγχρονισμού. [13] Μια προσέγγιση είναι το κλείδωμα για μια κοινόχρηστη μεταβλητή : μόνο ένας επεξεργαστής κάθε φορά μπορεί να αποκτήσει το κλείδωμα και οι άλλοι επεξεργαστές που ενδιαφέρονται για τα κοινόχρηστα δεδομένα πρέπει να περιμένουν μέχρι ο πρώτος επεξεργαστής να ξεκλειδώσει τη μεταβλητή. [13]

Μια άλλη μορφή υπολογιστικής μεγάλης κλίμακας (large scale computing) είναι η υπολογιστική πλέγματος (grid computing), στην οποία οι υπολογιστές είναι διάσπαρτοι σε μεγάλες περιοχές, και στην συνέχεια τα προγράμματα που εκτελούνται σε αυτούς πρέπει να επικοινωνούν μέσω δικτύων μεγάλης απόστασης (long haul networks). [13] Η πιο γνωστή μορφή υπολογιστικής πλέγματος ήταν αυτή που παρουσιάστηκε πρωτοποριακά στο έργο SETI@home . Με εκατομμύρια υπολογιστές αδρανείς διάσπαρτοι χωρίς να κάνουν κάτι ιδιαίτερο σε οποιαδήποτε χρονική στιγμή, θα μπορούσαν να δωθούν κάποιες εργασίες αν υπήρχε λογισμικό που θα μπορούσε να

εκτελεστεί σε αυτούς του υπολογιστές και να έδινε σε κάθε υπολογιστή ένα τμήμα του προβλήματος για να δουλέψει. Ενα παράδειγμα ήταν η αναζήτηση για εξωγήινη Νοημοσύνη (Search for Extra Terrestrial Intelligence –SETI). [13] Σε αυτό το project συνεργάστηκαν πάνω απο 5 εκατομμύρια χρήστες και συνέσφεραν πάνω απο 19 δισεκατομμύρια ώρες χρόνου επεξεργασίας. [13] Μέχρι το τέλος του 2006 το πλέγμα SETI@home λειτουργούσε στα 257 TeraFLOPS. [13]

1.3 Πολυνημάτωση

Η πολυνημάτωση υλικού (Hardware Multithreading) επιτρέπει σε πολλά νήματα να μοιράζονται τις λειτουργικές μονάδες ενός μοναδικού επεξεργαστή με επικαλυπτόμενο τρόπο [13]. Για να επιτραπεί και για να επιτευχθεί αυτή η κοινή χρήση , ο επεξεργαστής θα πρέπει να αναπαράγει την ανεξάρτητη κατάσταση κάθε νήματος (δηλαδή να δημιουργηθεί αντίγραφό της) [13]. Εάν για παράδειγμα το κάθε νήμα χρειάζεται ένα ξεχωριστό αντίγραφο ενός αρχείου ή του αρχείου καταχωρητών και του μετρητή προγράμματος (Program Counter- PC) [13].Η ίδια η μνήμη μπορεί να μοιράζεται μέσω των μηχανισμών εικονικής μνήμης, οι οποίοι ήδη υποστηρίζουν τον πολυπρογραμματισμό (multiprogramming) [13]. Επίσης το υλικό θα πρέπει να μπορεί να αλλάζει σε διαφορετικό νήμα κάθε φορά αρκετά γρήγορα. Δηλαδή μια εναλλαγή νήματος θα πρέπει να είναι πιο αποδοτική απο μια εναλλαγή διεργασίας που τυπικά θέλει εκατοντάδες ή χιλιάδες κύκλους του επεξεργαστή ενώ μια εναλλαγή νήματος μπορεί να είναι ακαριαία.

Όσον αφορά για την πολυνημάτωση υλικού έχουμε δύο προσεγγίσεις, η μια είναι η λεπτή πολυνημάτωση και η δεύτερη η χονδρή πολυνημάτωση.

Η λεπτή πολυνημάτωση (fine – grained multithreading) αλλάζει νήμα σε κάθε εντολή, που αυτό μας οδηγεί σε μια πλεγμένη (interleaved) εκτέλεση πολλών νημάτων. Αυτό μπορεί να γίνεται με μια μέθοδο εκ περιτροπής (round robin) , αγνοώντας τα σταματημένα νήματα εκείνη την χρονική στιγμή. Επίσης για να είναι λειτουργική η λεπτή πολυνημάτωση θα πρέπει ο επεξεργαστής να αλλάζει νήμα σε κάθε κύκλο του ρολογιού. Το θετικό της λεπτής πολυνημάτωσης είναι ότι μπορεί να κρύψει ενδεχόμενες απώλειες διεκπεραιωτικής ικανότητας που μπορεί να προέρχονται απο καθυστερήσεις (stalls) που μπορεί να είναι είτε σύντομες είτε μεγάλης διάρκειας, διότι όταν ένα νήμα είναι σταματημένο μπορούν να εκτελεστούν εντολές απο άλλα νήματα [13]. Και το αρνητικό με την λεπτή πολυνημάτωση είναι ότι επιβραδύνει την εκτέλεση των ανεξάρτητων νημά-των, επειδή ένα νήμα που είναι έτοιμο να εκτελεστεί χωρίς καθυστερήσεις θα καθυστε-ρήσει απο εντολές άλλων νημάτων [13].

Η χονδρή πολυνημάτωση (coarse – grained multithreading) είναι η αλλη επιλογή της πολυνημάτωσης. Εδώ η αλλαγή νημάτων γίνεται μόνο σε μεγάλου κόστους καθυστερήσεις, όπως να έχουμε αστοχίες στην κρυφή μνήμη δεύτερου επιπέδου. Οπότε μετριάζει αυτή η αλλαγή το να έχουμε εναλλαγή νημάτων χωρίς κόστος ,οπότε το πιο πιθανό είναι να έχουμε επιβράδυνση στην εκτέλεση ενός ανεξάρτητου νήματος και αυτό διότι θα έχουμε εκτέλεση εντολών απο άλλα νήματα μόνο όταν ένα νήμα έχει μεγάλου κόστους καθυστέρηση. Ενα βασικό μειονέκτημα της χονδρής πολυνημάτωσης είναι η περιορισμένη ικανότητα ανάκαμψης απο απώλειες διεκπαιραιωτικής ικανότητας, ιδίως για μικρές καθυστερήσεις [13]. Αυτός ο περιορισμός προκύπτει απο το κόστος εκκίνησης της διοχέτευσης στην χονδρή πολυνημάτωση [13]. Μια CPU με χονδρή πολυνημάτωση ξεκινάει εντολές απο ένα μόνο νήμα ,όταν συμβαίνει μια καθυστέρηση οπότε η διοχέτευση

θα πρέπει να εκκενωθεί ή να παγώσει [13].Οπότε το νέο όμως νήμα που αρχίζει να εκτελείται μετά την καθυστέρηση θα πρέπει να γεμίσει την διοχέτευση ούτως ώστε να μπορούν να ολοκληρωθούν οι εντολές [13]. Τελικά η χονδρή πολυνημάτωση είναι πιο χρήσιμη για την μείωση της ποινής των καθυστερήσεων υψηλού κόστους , όπου η επαναπλήρωση της διοχέτευσης είναι αμελητέα σε σχέση με το χρόνο της καθυστέρησης [13].

Μια άλλη πολυνημάτωση είναι η ταυτόχρονη πολυνημάτωση (simultaneous multithreading – SMT) είναι μια παραλλαγή της πολυνημάτωσης που χρησιμοποιεί τους πόρους ενός δυναμικά χρονοπρογραμματιζόμενου επεξεργαστή εκκίνησης πολλών εντολών, για να αξιοποιήσει την παραλληλία σε επίπεδο νήματος ταυτόχρονα με την αξιοποιήσή της σε επίπεδο εντολής. [13] Η ιδέα της ταυτόχρονης πολυνημάτωσης είναι ότι οι επεξεργαστές εκκίνησης πολλών εντολών έχουν συχνά διαθέσιμη περισσότερη παραλληλία λειτουργικών μονάδων απο όση μπορεί να χρησιμοποιήσει αποδοτικά ένα μοναδικό νήμα.[13] Επίσης με τη μετονομασία καταχωρητών (register renaming) και το δυναμικό χρονοπρογραμματισμό, μπορούν να ξεκινήσουν πολλές εντολές απο ανεξάρτητα νήματα χωρίς να παίζουν ρόλο οι εξαρτήσεις μεταξύ τους η επίλυση των εξαρτήσεων μπορεί να αντιμετωπιστεί με την δυνατότητα δυναμικού χρονοπρογραμματισμού.[13] Επίσης η ταυτόχρονη πολυνημάτωση δεν εναλλάσει τη χρήση των πόρων σε κάθε κύκλο. [13]

1.4 GPU

Μονάδες GPU για γραφικά και υπολογιστική. Η GPU = Graphics processing unit = μονάδα επεξεργασίας γραφικών η οποία βρίσκεται σε κάθε Η/Υ. Στην πιο απλή μορφή της παράγει γραφικά εικόνες και βίντεο δυο και τριών διαστάσεων επιτρέποντας την ύπαρξη τους στα λειτουργικά συστήματα που βασίζονται σε παράθυρα , σε γραφικά ,σε παιχνίδια ,σε εφαρμογές επεξεργασίας εικόνας και βίντεο. [13]

Η σύγχρονη GPU είναι ένας πολυεπεξεργαστής υψηλής παραλληλίας και υψηλής πολυνημάτωσης βελτιστοποιημένος για οπτική υπολογιστική (visual computing). Για να παρέχει οπτική αλληλεπίδραση με υπολογιζόμενα αντικείμενα σε real time χρόνο μέσω γραφικών , εικόνων και βίντεο , η GPU διαθέτει μια ενοποιημένη αρχιτεκτονική για γραφικά και υπολογιστική που εξυπηρετεί και ως προγραμματίσιμος επεξεργαστής γραφικών και ως επεκτάσιμη πλατφόρμα παράλληλης υπολογιστικής.[13] Οι προσωπικοί υπολογιστές συνδυάζουν GPU με CPU οπότε έχουμε ετερογενή συστήματα (heterogeneous systems).[13]

Στην πρώιμη μορφή τους οι κάρτες γραφικών είχαν έναν ελεγκτή VGA (Video Graphics Array) , ήταν ελεγκτής μνήμης , υπήρχε και μια γεννήτρια εικόνων που συνδέονται με μια μνήμη DRAM. [13] Αργότερα προσθέθηκαν πιο πολλές λειτουργίες . Λίγο αργότερα ενσωματώθηκαν λειτουργίες επιτάχυνσης τρισδιάστατων γραφικών και υλικό για την διευθέτυνση τριγώνων και ραστεροποίηση – διαίρεση των τριγώνων σε μεμονωμένα πίξελ , και τη χαρτογράφηση υφής και την σκίαση shading δηλαδή την εφαρμογή χαλκομανιών ή μοτίβων σε πίξελ και την ανάμιξη χρωμάτων. [13]

Αργότερα ενσωμάτωσε σε ένα τσιπ κάθε λεπτομέρεια της διοχέτευσης γραφικών (pipeline) υψηλών επιδόσεων και λόγω αυτού άλλαξε όνομα απο VGA σε GPU.[13]

Με την πάροδο του χρόνου η GPU έγιναν πιο προγραμματίσιμες αντικαθιστώντας την λογική σταθερών λειτουργιών κρατώντας την 3D graphics pipeline.[13]

Μετά για να είναι πιο ακριβείς σε υπολογισμούς απο την αριθμητική με αριθμοδείκτες, μπήκανε ακέραιοι σταθερής υποδιαστολής , κινητή υποδιαστολή διπλής ακριβείας. Οπότε περάσαμε πλέον σε GPU με μαζικά παράλληλους προγραμματίσιμους επεξεργαστές με εκατοντάδες πυρήνες και χιλιάδες νήματα.[13]

Αργότερα προσθέθηκε και υλικό μνήμης για την υποστήριξη γλωσσών προγραμματισμού γενικού σκοπού και δημιουργήθηκε και ένα προγραμματιστικό περιβάλλον για να είναι δυνατός ο προγραμματισμός των GPU με χρήση γλωσσών όπως C και C++ .[13] Οπότε η GPU γίνεται ένας πλήρως γενικού σκοπού προγραμματίσιμος επεξεργαστής πολλών πυρήνων (manycore processor) με πλεονεκτήματα και περιορισμούς. [13]

Οι GPU και οι οδηγοί τους (drivers) υλοποιούν τα μοντέλα επεξεργασίας γραφικών OpenGL και DirectX . Το OpenGL είναι ένα ανοικτό πρότυπο για το προγραμματισμό τρισδιάστατων γραφικών που είναι διαθέσιμο στους Η/Υ .[13] Το DirectX είναι μια σειρά προγραμματιστικών διασυνδέσεων πολυμέσων της Microsoft που περιλαμβάνει και το Direct3D για τρισδιάστατα γραφικά. Απο την στιγμή που αυτές οι διασυνδέσεις προγραμματισμού εφαρμογών (Application programming interface – API) έχουν καλή συμπεριφορά , πετυχαίνεται αποδοτική επιτάχυνση μέσω υλικού των συναρτήσεων επεξεργασίας γραφικών που ορίζονται στο API.[13] Πέρα απο την αύξηση των τρανζίστορ στις GPU έχουμε και διπλασιασμό της απόδοσης και εμφάνιση νέων εφαρμογών που δεν υπήρχαν πριν. Οπότε έχουμε επεξεργασία γραφικών και παράλληλη υπολογιστική που μας οδηγεί στην οπτική υπολογιστική (visual computing). Γίνεται αντικατάσταση του παραδοσιακού μοντέλου διοχέτευσης γραφικών με ακολουθιακό υλικό (sequential hardware graphics pipeline) με προγραμματίσιμα στοιχεία για προγράμματα χειρισμού γεωμετρίας , κορυφών και πίξελ, όλα αυτά μας οδηγούν σε νέους αλγόριθμους γραφικών και έχουμε νέες εφαρμογές παράλληλης επεξεργασίας στις GPU υψηλών επιδόσεων.[13]

Η υπολογιστική με GPU (GPU computing) είναι η χρήση της GPU για υπολογισμό μέσω μιας γλώσσας και ενός API παράλληλων προγραμματισμού , χωρίς την χρήση του παραδοσιακού API γραφικών και του μοντέλου διοχέτευσης γραφικών.[13]

Η Αρχιτεκτονική Συσκευών Ενοποιημένων Υπολογισμών (Compute Unified Device Architecture –CUDA) είναι ένα επεκτάσιμο μοντέλο παράλληλου προγραμματισμού και πλατφόρμα λογισμικού για την GPU και άλλους παράλληλους επεξεργαστές , το οποίο επιτρέπει στον προγραμματιστή να παρακάμψει το API και τις διασυνδέσεις γραφικών της GPU και να προγραμματίζει απλώς σε C και C++ [13]. Το προγραμματιστικό μοντέλο CUDA έχει ένα στυλ λογισμικού SPMD (Single – Program Multiple Data - Ενα πρόγραμμα , πολλά δεδομένα), όπου σύμφωνα με αυτό ο προγραμματιστής γράφει για μια παρουσία νήματος, ένα προγραμμα που εκτελείται , απο πολλά νήματα , νήματα παράλληλα, στους πολλούς επεξεργαστές της GPU . Στην πράξη , η CUDA παρέχει επίσης την δυνατότητα προγραμματισμού και των άλλων πυρήνων CPU , οπότε είναι ένα περιβάλλον γραφής παράλληλων προγραμμάτων για ολόκληρο το ετερογενές υπολογιστικό σύστημα.[13] Οι ενοποιημένες αρχιτεκτονικές GPU βασίζονται σε μια παράλληλη διάταξη πολλών προγραμματίσιμων επεξεργαστών. Ενοποιούν στους ίδιους επεξεργαστές την επεξεργασία σκίασης κορυφών, γεωμετρικών σχημάτων, και πίξελ με την παράλληλη υπολογιστική, σε αντίθεση με προηγούμενες GPU που είχαν ξεχωριστούς επεξεργαστές αφιερωμένους σε κάθε τύπο επεξεργασίας. [13] Η διάταξη των προγραμματίσιμων επεξεργαστών είναι πολύ κοντά τοποθετημένη σε επεξεργαστές , σταθερών συναρτήσεων

για φιλτράρισμα υφής (texture filtering), ραστεροποίηση (rasterization), εξομάλυνση ορίων (anti – aliasing), συμπίεση , αποσυμπίεση , εμφάνιση στην οθόνη , αποκωδικοποίηση βίντεο , και επεξεργασία βίντεο υψηλής ευκρίνειας .[13] Θα εστιάσουμε στους προγραμματίσιμους επεξεργαστές παρόλο που οι επεξεργαστές σταθερών συναρτήσεων είναι πιο αποδοτικές.

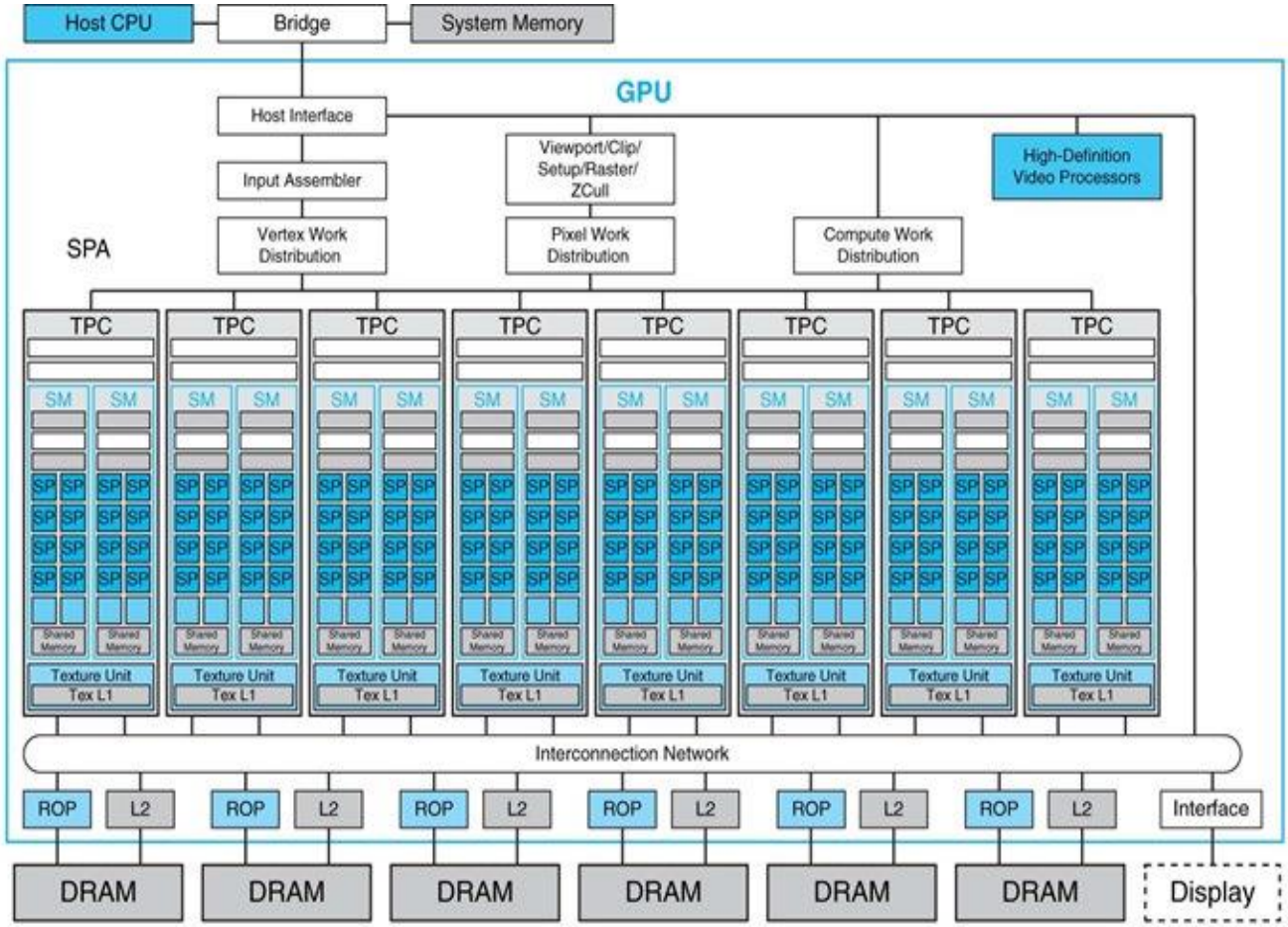
Οι GPU με μεγάλο αριθμό πυρήνων έχουν άλλη αρχιτεκτονική σχεδίαση που επιτρέπει αποδοτική εκτέλεση πολλών παράλληλων νημάτων σε πολλούς πυρήνες επεξεργαστών.[13] Με χρήση πιο απλών πυρήνων έχουμε καλύτερη παραλληλία δεδομένων μεταξύ ομάδων νημάτων, και το μεγαλύτερο μερος των τρανζίστορ του κάθε τσιπ είναι ειδικά για υπολογισμούς και μικρότερο για κρυφές μνήμες στο τσιπ και την επιβάρυνση.[13]

Μια διάταξη ενοποιημένης GPU περιέχει πολλούς πυρήνες επεξεργαστών, που τυπικά είναι οργανωμένοι ως πολυνηματικοί πολυεπεξεργαστές.[13]

Ενα παράδειγμα είναι μια GPU με μια διάταξη 112 πυρήνων επεξεργαστών συνεχούς ροής (streaming processors – SP), οι οποίοι είναι οργανωμένοι ως 14 πολυνηματικοί πολυεπεξεργαστές συνεχούς ροής (streaming multiprocessors – SM). [13] Κάθε πυρήνας SP έχει υψηλή πολυνημάτωση , και χειρίζεται 96 ταυτόχρονα νήματα και την κατάσταση τους στο υλικό. Οι επεξεργαστές συνδέονται με τέσσερα διαμερίσματα DRAM εύρους 64 bit μέσω ενός δικτύου διασύνδεσης . Κάθε SM έχει οκτώ (8) πυρήνες SP δυο μονάδες ειδικών συναρτήσεων (special function units – SFU), κρυφές μνήμες εντολών και σταθερών (instruction and constant caches), μια πολυνηματική μονάδα εντολών, και μια κοινόχρηστη μνήμη. [13] Αυτή είναι η βασική αρχιτεκτονική tesla που υλοποιείται στην GeForce 8800 της Nvidia . Εχει ενοποιημένη αρχιτεκτονική στην οποία τα παραδοσιακά προγράμματα γραφικών για σκίαση κορυφών, γεωμετρικών σχημάτων , και πίξελ εκτελούνται στους ενοποιημένους SM και τους πυρήνες SP τους, και τα προγράμματα υπολογισμών εκτελούνται στους ίδιους επεξεργαστές.[13]

Η αρχιτεκτονική της διάταξης των επεξεργαστών μπορεί να προσαρμοστεί σε μικρότερες και μεγαλύτερες διαμορφώσεις GPU με την προσαρμογή του αριθμού των πολυεπεξεργαστών και των διαμερισμάτων μνήμης.[13]

Η παρακάτω εικόνα δείχνει οκτώ συστοιχίες όπου δυο SM μοιράζονται μια μονάδα επεξεργασίας υφής (texture unit) και μια κρυγή μνήμη L1 για την επεξεργασία υφής. Η μονάδα υφής παραδίδει φιλτραρισμένα τα αποτελέσματα στον SM με δεδομένο ένα σύνολο συντενταγμένων ενός χάρτη υφής (texture map).[13] Και επειδή οι περιοχές που υποστηρίζει το φιλτράρισμα συχνά επικαλύπτονται για διαδοχικές αιτήσεις υφής, μια μικρή κρυφή μνήμη L1 συνεχούς ροής για επεξεργασία υφής (streaming L1 texture cache) είναι αρκετή για να μειώσει τον αριθμό των αιτήσεων προς το σύστημα μνήμης.[13] Η διάταξη των επεξεργαστών συνδέεται με επεξεργαστές λειτουργιών ράστερ (raster operation processors – ROP), κρυφές μνήμες L2 για επεξεργασία υφής (L2 texture caches), εξωτερικές μνήμες DRAM , και μνήμη συστήματος μέσω ενός δικτύου διασύνδεσης σε όλη την έκταση της GPU. [13] Ο αριθμός των επεξεργαστών και ο αριθμός των μνημών μπορούν να προσαρμοστούν ώστε να σχεδιάζονται ισορροπημένα συστήματα GPU για διαφορετική απόδοση και διαφορετικά τμήματα της αγοράς.[13]



Graphisme unifié NVIDIA Tesla . Ce GeForce 8800 a 128 processeur en stream (SP) dans 16 multiprocesseurs en stream (SM), arrangé dans huit groupes de texture/processeur (TPC). Les processeurs se connectent avec six divisions de DRAM de 64 bit de largeur via un réseau d'intercom-munication.Graphe fourni par NVIDIA.

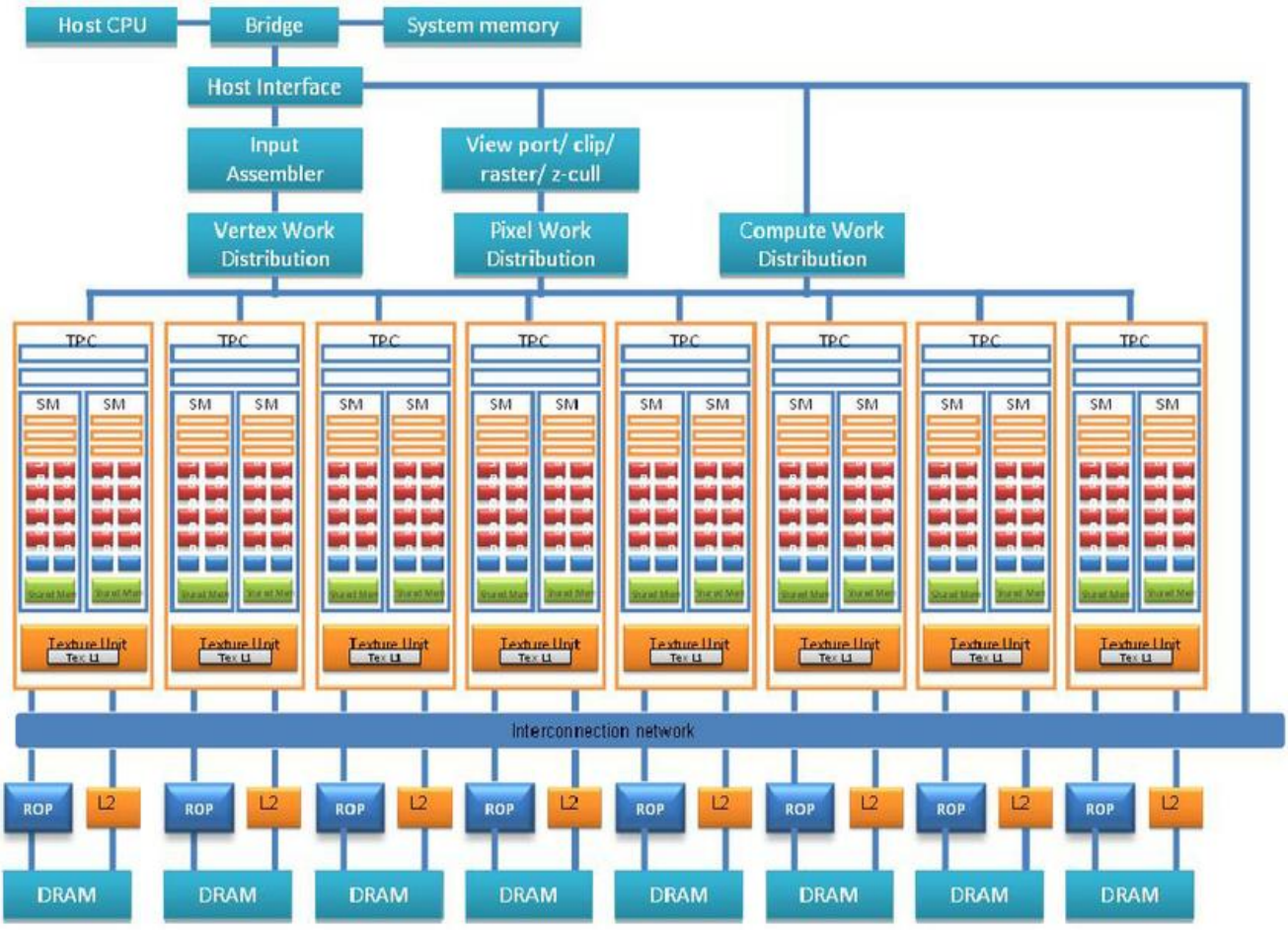
Σχήμα 1.4.1

Βασικη Αρχιτεκτονικη Ενοποιημένης GPU. GPU με 128 πυρήνες επεξεργαστών συνεχούς ροής (streaming processor – SP) οργανωμένους σε 16 πολυεπεξεργαστές συνεχούς ροής (streaming multiprocessors – SM) οι πυρήνες έχουν υψηλή πολυνημά-τωση. [13] Εχει τη βασική αρχιτεκτονική Tesla μιας GeForce 8800 της Nvidia. Οι επε-ξεργαστές συνδέπνται με τέσσερα διαμερίσματα DRAM εύρους 64 bit μέσω ενός δικτύ-ου διασύνδεσης. [13] Κάθε SM έχει οκτώ πυρήνες SP , δύο μονάδες ειδικών συναρ-ήσεων (special function units – SPU), κρυφές μνήμες εντολών (I-Cache) και σταθερών (C-Cache), μια πολυνηματική μονάδα εκκίνησης εντολών (MT Issue) , και μια κοινόχρ-στη μνήμη (Viewport = Παράθυρο παρατήρησης, Clip = Περικοπή, Setup = Διαμόρ-φωση, Raster = Ραστερ , ZCull = Απαλοιφή αόρατων στοιχείων). [13]

πηγή

https://www.researchgate.net/figure/Graphisme-unifie-NVIDIA-Tesla-Ce-GeForce-8800-a-128-processeur-en-stream-SP-dans-16_fig1_265165403

Παρακάτω βλέπουμε διάφορες αρχιτεκτονικές απο διάφορες κάρτες γραφικών της Nvidia

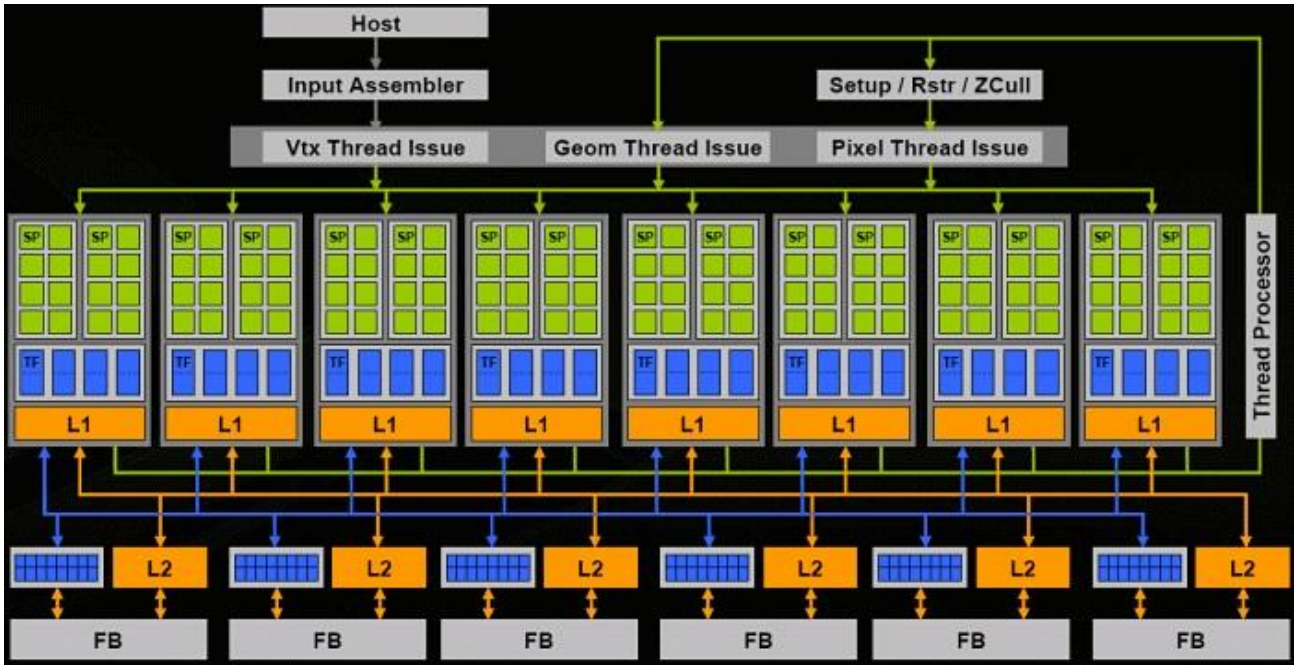


Βασικη Αρχιτεκτονικη Ενοποιημένης GPU

Σχήμα 1.4.2

πηγή

https://www.researchgate.net/figure/Block-diagram-of-NVIDIA-Tesla-Architecture-for-GeForce-8800-Courtesy-1_fig8_267847833



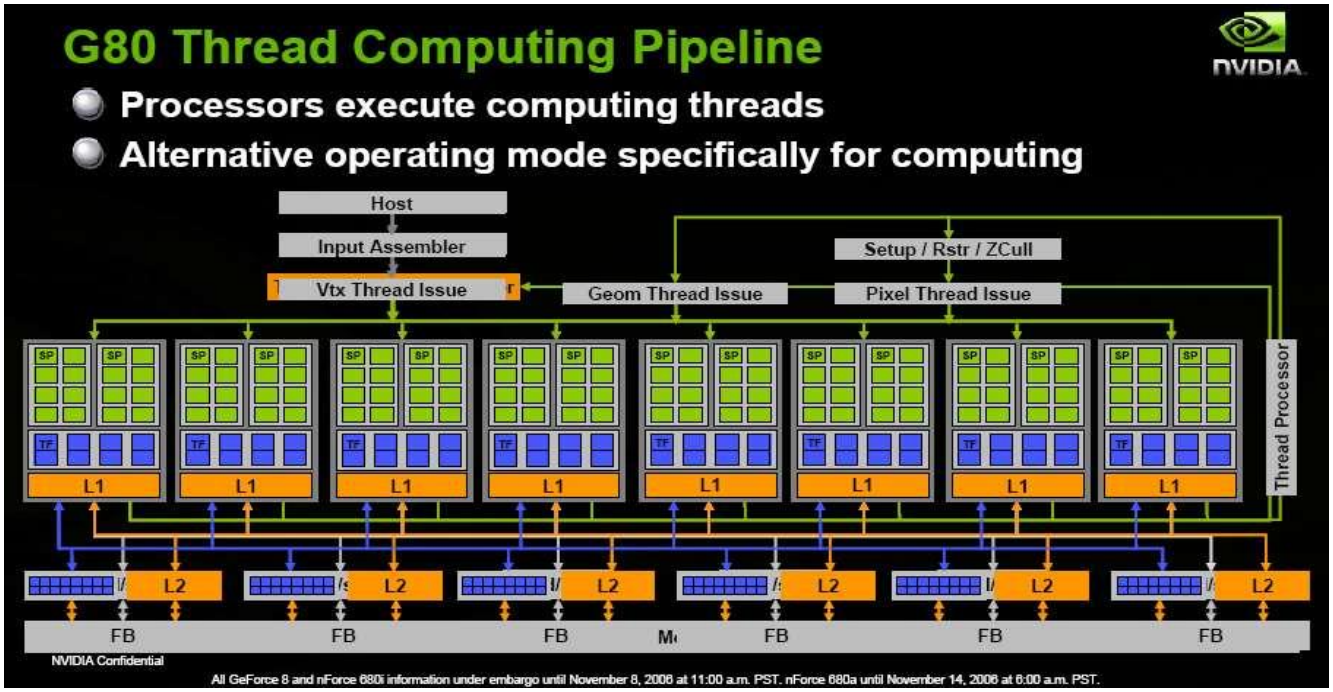
Nvidia's GTX 8800 family were the first consumer graphics cards to swap dedicated pixel and vertex shaders for a wide array of simpler stream processors (SPs, later referred to as CUDA cores). While previous GPUs were vector processors that could operate concurrently on the red, green, blue, and alpha color components of a single pixel, Nvidiadesigned the G80 as a scalar processor, in which each streaming processor handled one color component. At a high level, Nvidia had switched from a GPU architecture with dedicated hardware for specific types of shader programs to an array of relatively simple cores that could be programmed to perform whatever types of shader calculations the application required at that particular moment.

The simpler cores could also be clocked much faster. The GeForce 7900 GTX was built on a 90nm process and hit 650MHz, while the GeForce 8800 GTX was built on an 80nm "half node" process and ran its shader cores at 1.35GHz. But as with any brand-new architecture, there were significant risks involved.

Σχήμα 1.4.3

πηγή

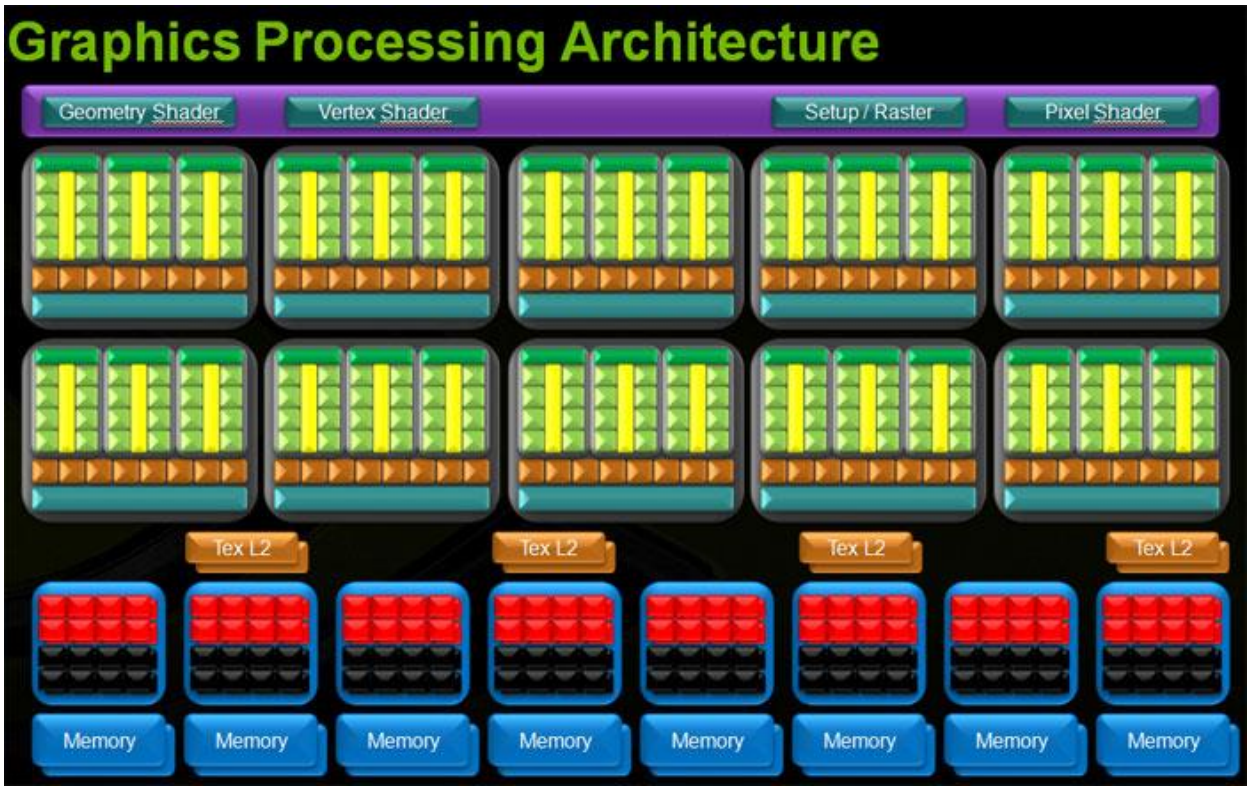
https://www.extremetech.com/gaming/239078-ten-years-ago-today-nvidia-launched-g80-powered-geforce-8800-changed-pc-gaming-computing-forever



Σχήμα 1.4.4

πηγή

http://spmarchitecture.com/cuda-architecture/nvidia-cuda-compute-unified-device-architecture-214808/



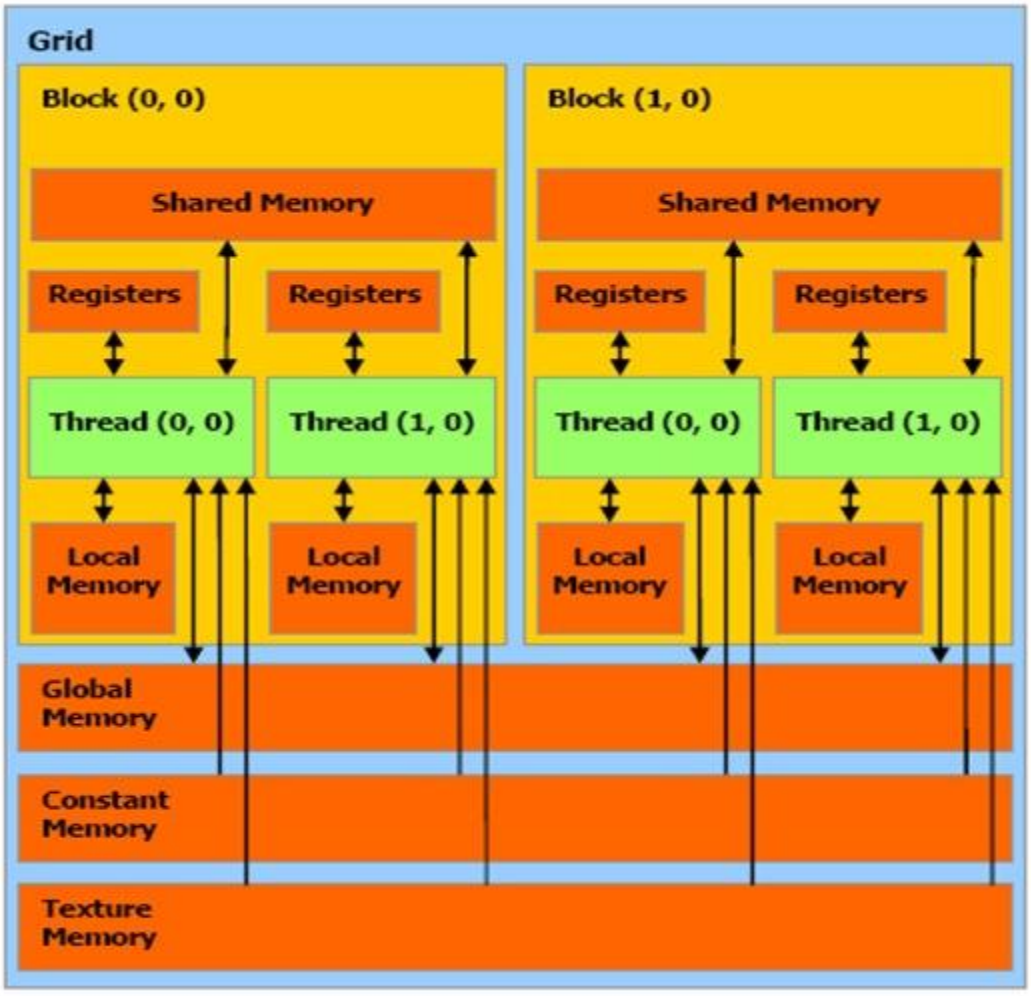
- 1.4 billion transistors
- 993 GigaFLOP processing power
- 240 processing (shader) cores (GTX 280)

- 192 processing (shader) cores (GTX 260)
- DirectX 10
- New power management enhancements
- CUDA parallel processing
- GeForce PhysX

Σχήμα 1.4.5

πηγή

https://www.guru3d.com/articles-pages/geforce-gtx-280-review-test,3.html

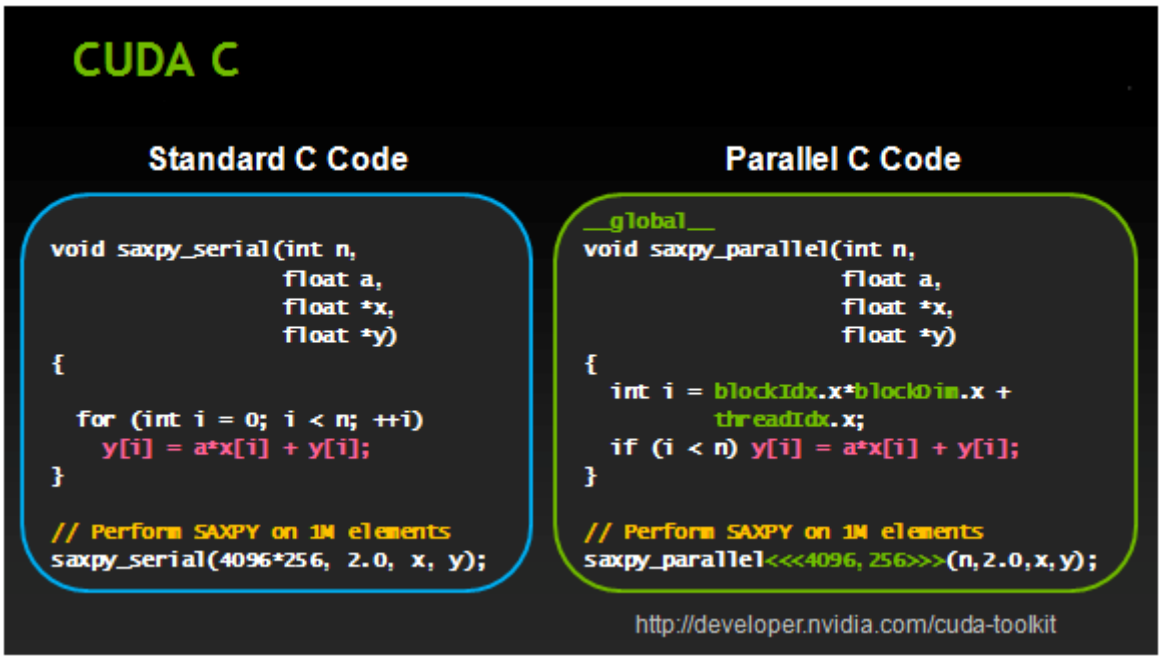


Πλέγμα

Σχήμα 1.4.6

πηγή

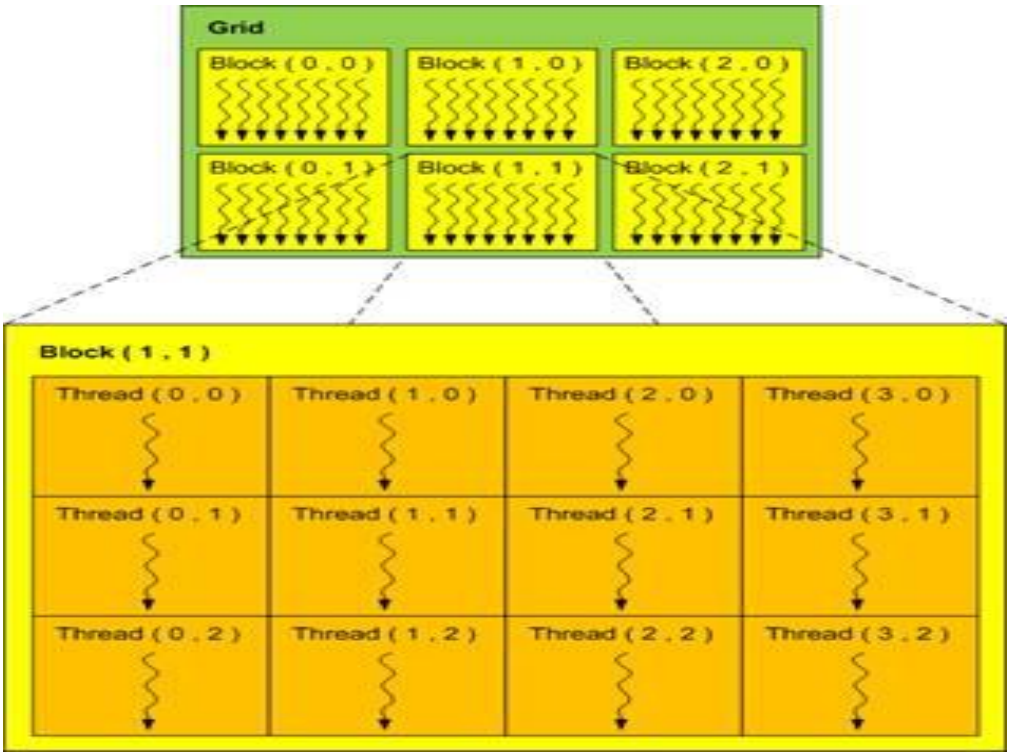
https://www.researchgate.net/figure/Le-modele-memoire-du-CUDA_fig5_265165403



Σχήμα 1.4.7

πηγή

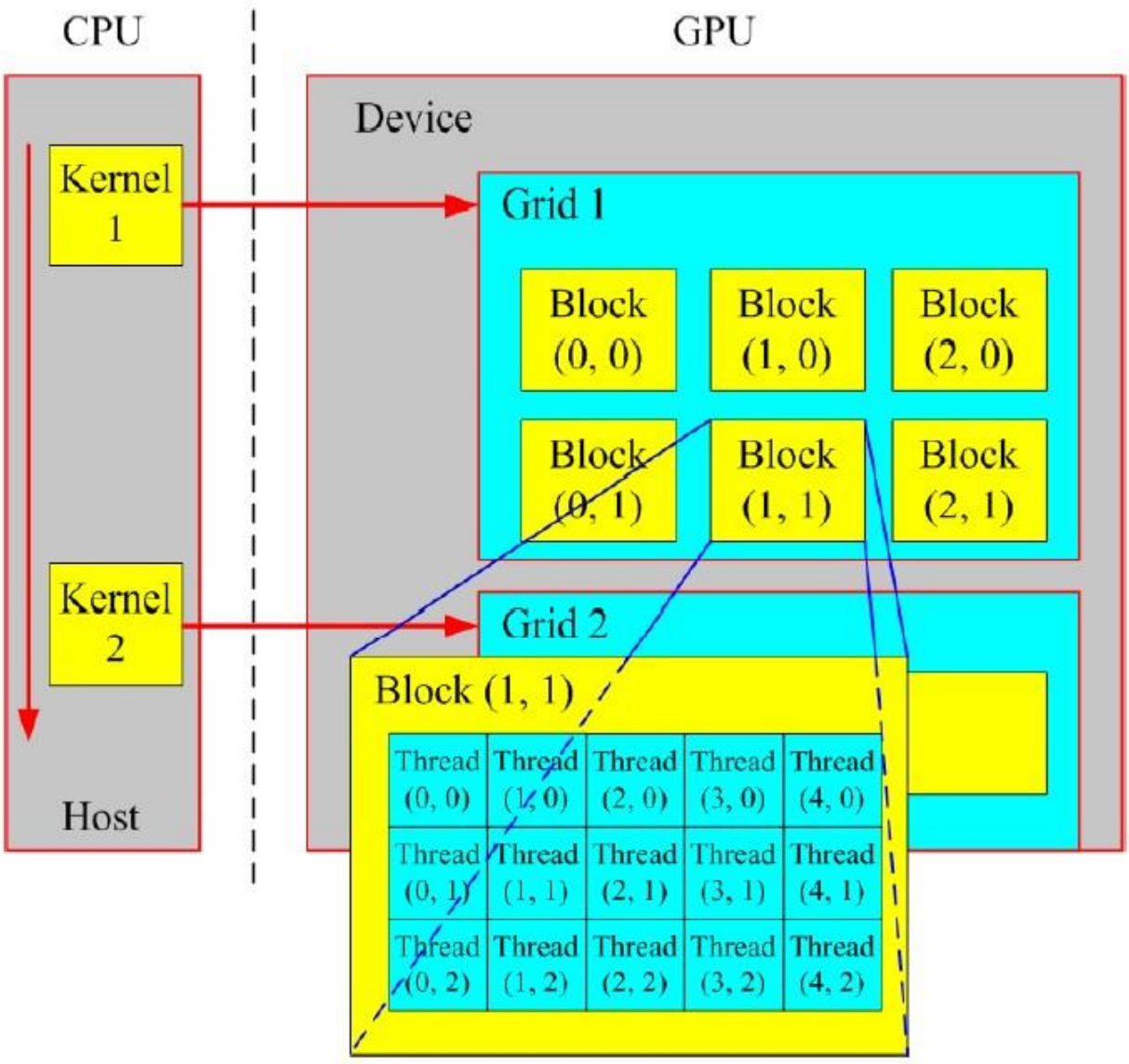
http://www.drdobbs.com/tools/nvidia-cuda-55-reaches-arm/240159621



Σχήμα 1.4.8

πηγή

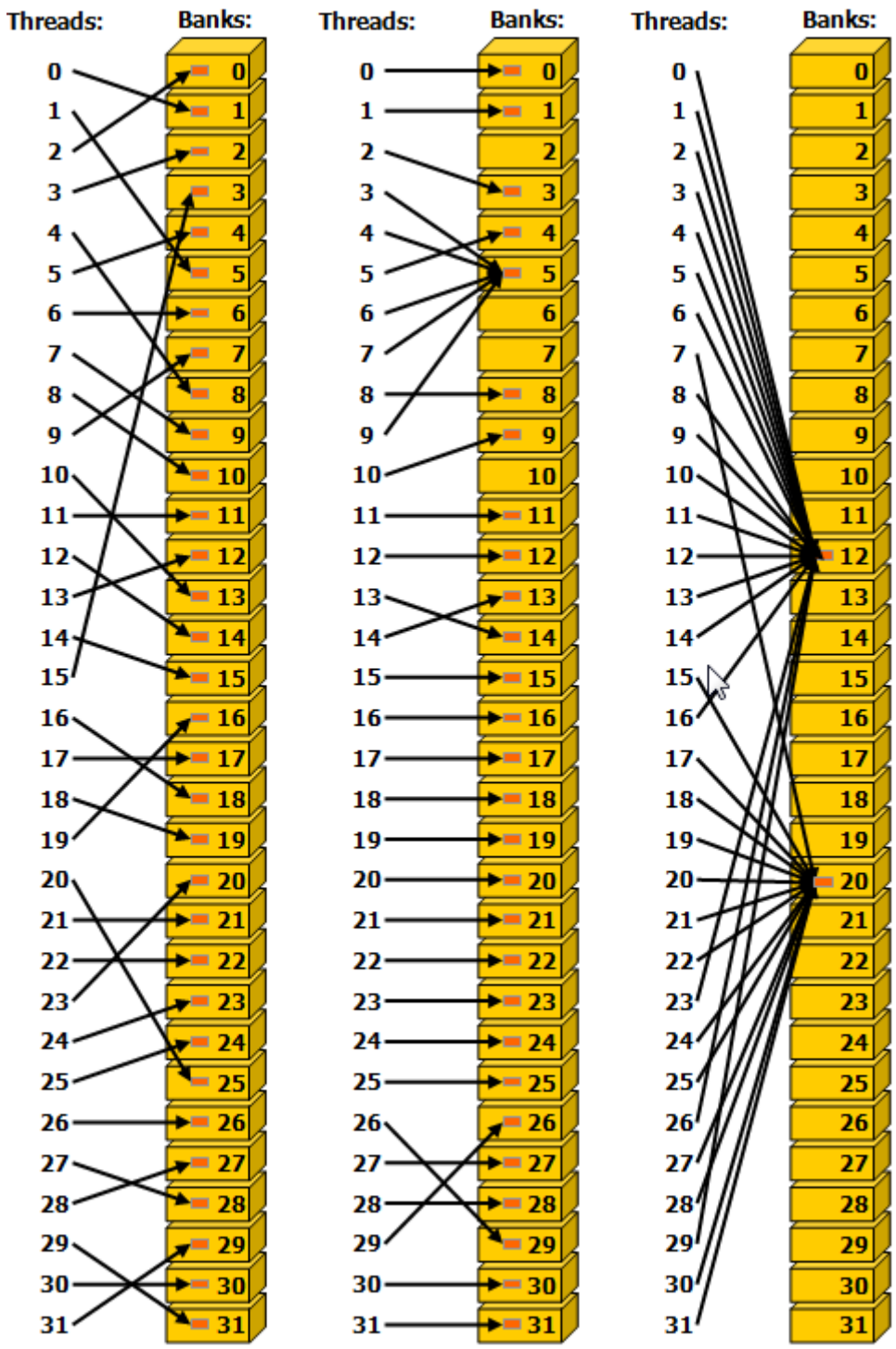
http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm



Σχήμα 1.4.9

πηγή

https://theimmagine.eu/mejores-135-im%C3%A1genes-de-ratones-y-molde-navide%C3%B1os-en.html



Σχήμα 1.4.10

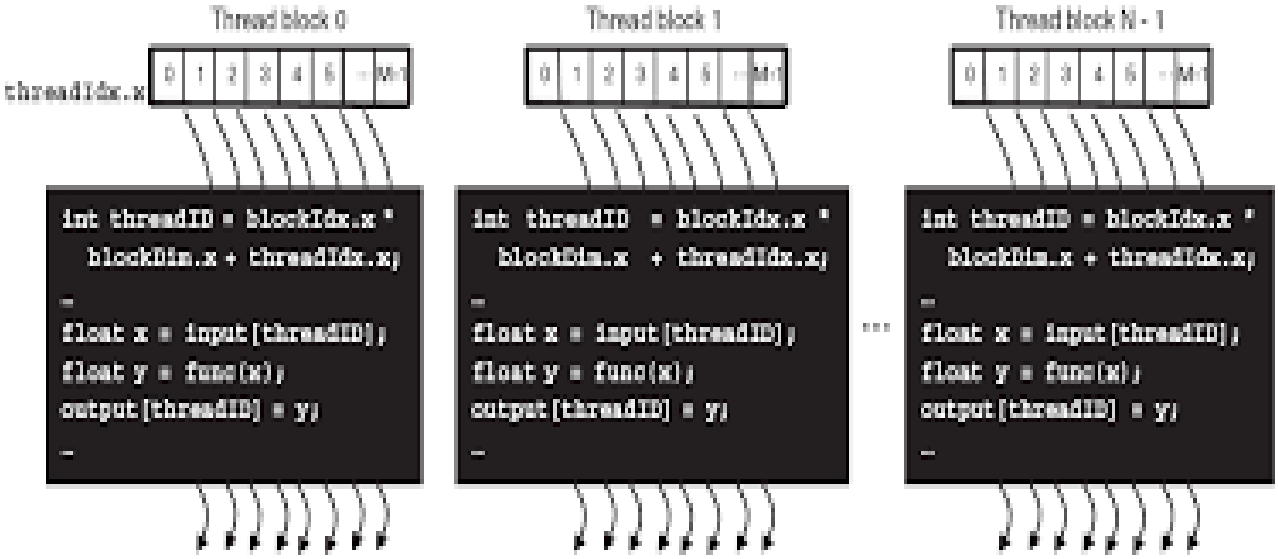
πηγή

https://www.theimagen.eu/instand-download-1-x-2-domino-vorlage-festlegen.html

Σχήμα 1.4.11

πηγή

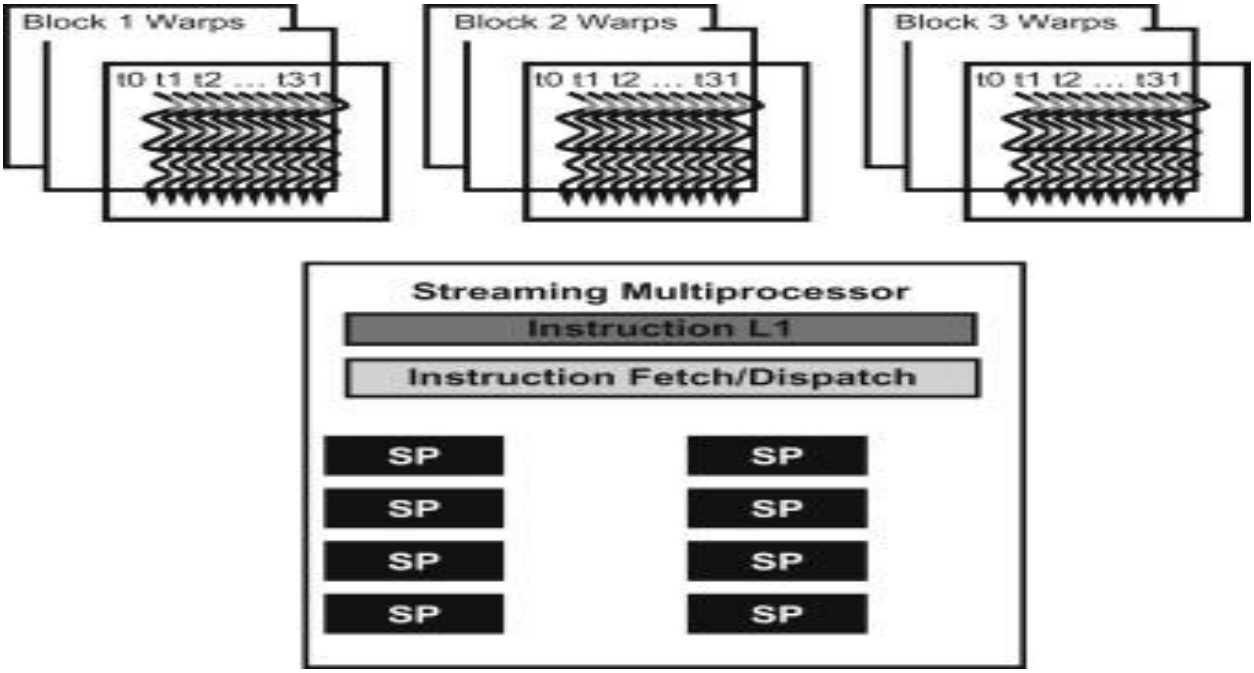
https://www.youtube.com/watch?v=x-N6pjBbyY0



Σχήμα 1.4.12

πηγή

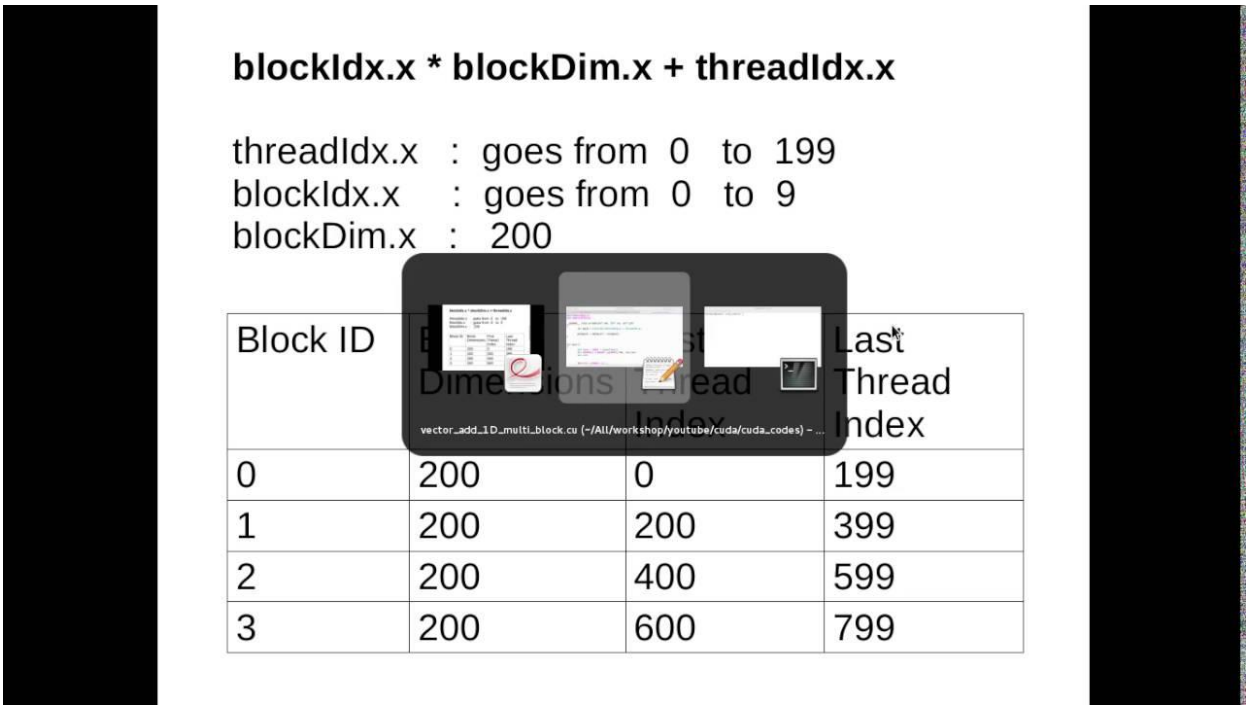
https://cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/lecture5.pdf



Σχήμα 1.4.13

πηγή

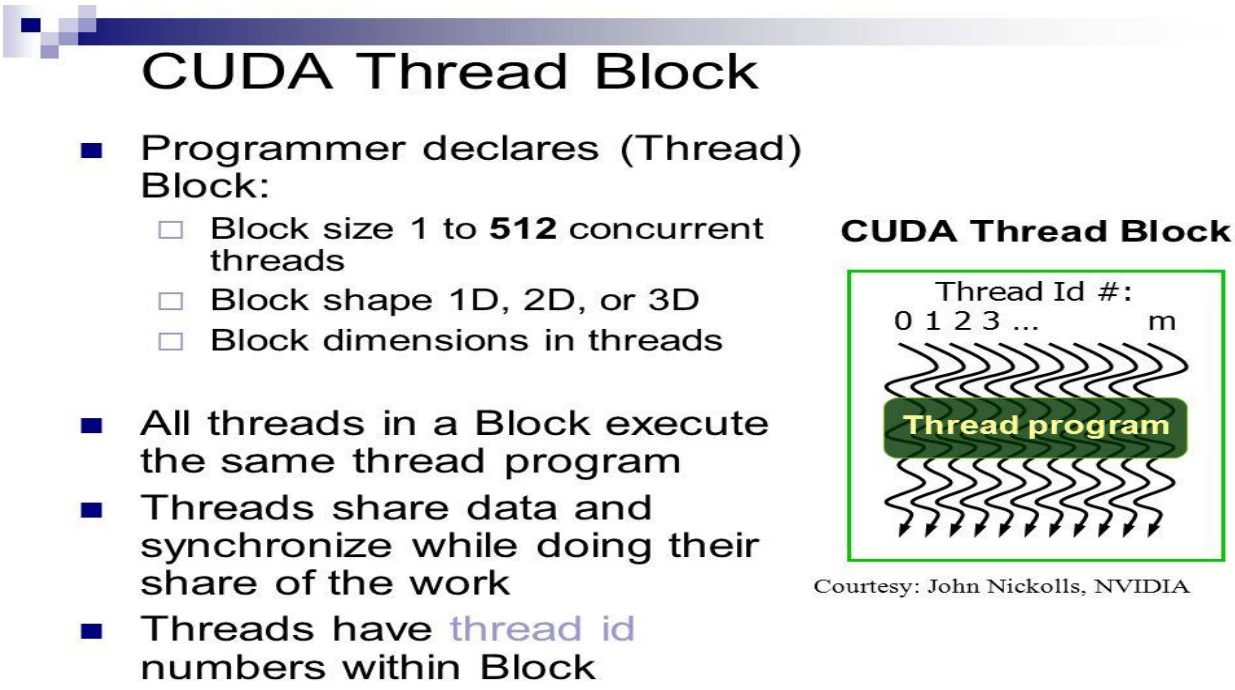
https://www.sciencedirect.com/topics/computer-science/thread-scheduling



Σχήμα 1.4.14

πηγή

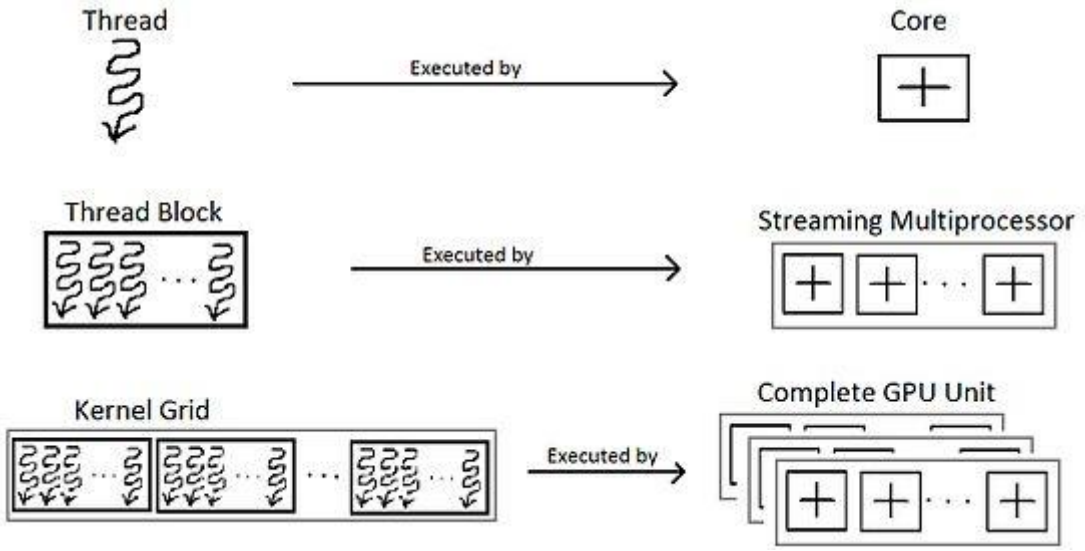
https://www.youtube.com/watch?v=Ma5a149XJoc



Σχήμα 1.4.15

πηγή

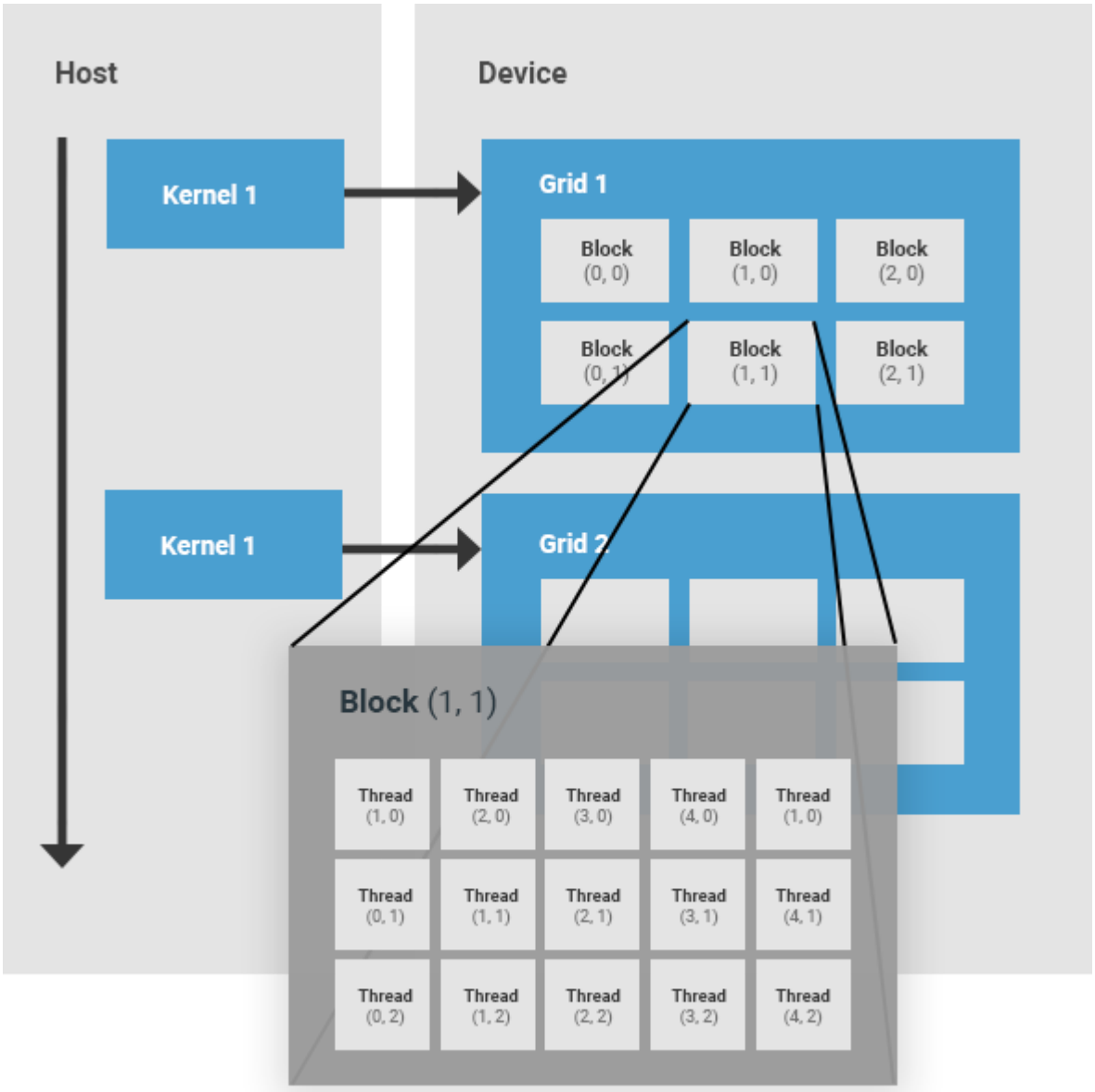
https://slideplayer.com/slide/5355102/



Σχήμα 1.4.16

πηγή

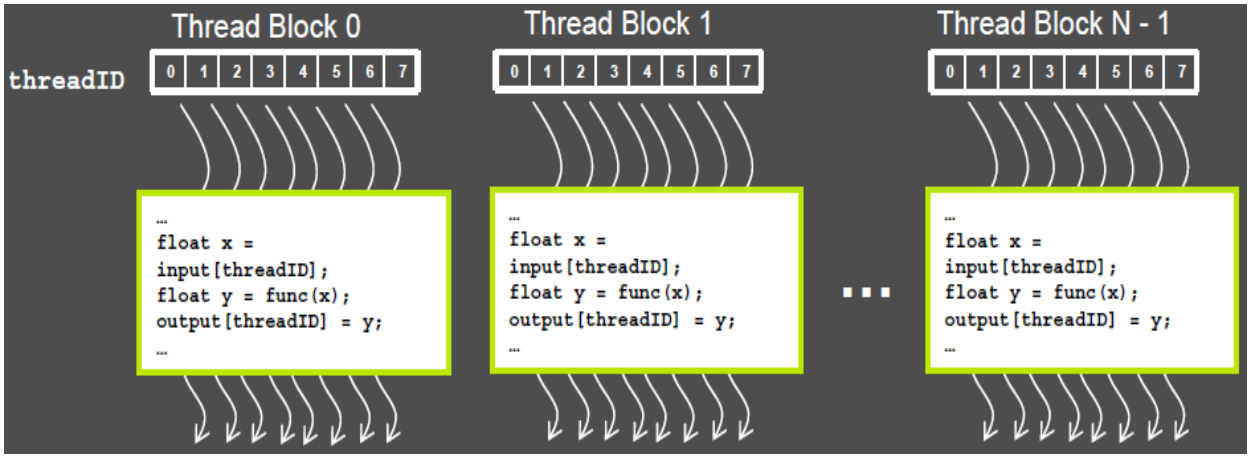
https://www.picswe.com/pics/grid-block-0e.html



Σχήμα 1.4.17

πηγή

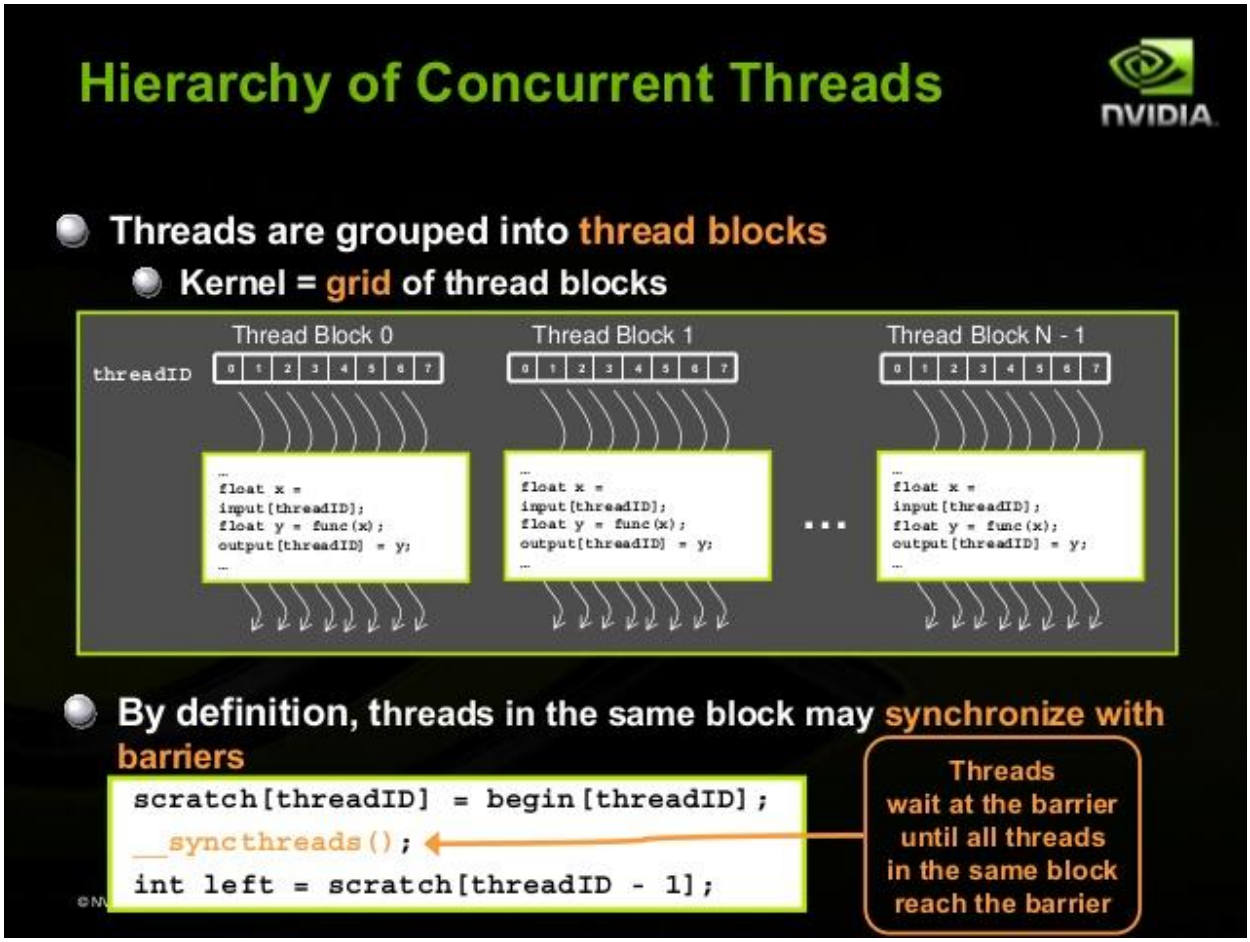
https://developer.codeplay.com/computecppce/latest/sycl-for-cuda-developers-execution-model



Σχήμα 1.4.18

πηγή

https://people.duke.edu/~ccc14/sta-663/CUDAPython.html



Σχήμα 1.4.19

πηγή

https://www.slideshare.net/angelamm2012/nvidia-cuda-tutorialnondaapr08

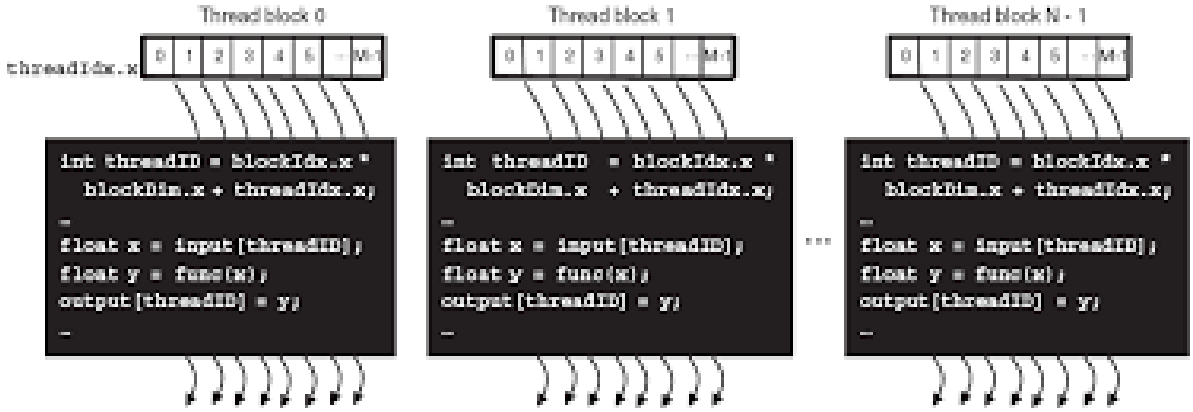
1.5 CUDA

Η CUDA είναι μια επέκταση των γλωσσών προγραμματισμού C ,C++ . Ο προγραμματιστής γράφει ένα σειριακό πρόγραμμα που καλεί παράλληλους πυρήνες (kernel), οι οποίοι μπορεί να είναι απλές συναρτήσεις μέχρι πλήρη προγράμματα. [13] Ενας πυρήνας εκτελείται παράλληλα απο ένα σύνολο παράλληλων νημάτων. Ο προγραμματστής οργανώνει αυτά τα νήματα σε μια ιεραρχία μπλοκ νημάτων και πλέγματα μπλοκ νημάτων.[13] Ενα μπλοκ νημάτων (thread block) είναι ένα σύνολο από ταυτόχρονα νήματα που μπορούν να συνεργαστούν μέσω συγχρονισμού φράγματος (barrier synchronization) και μέσω κοινής πρόσβασης σε ένα χώρο μνήμης ιδιωτικό για το μπλοκ. Ενα πλέγμα (grid) είναι ένα σύνολο μπλοκ νημάτων που καθένα μπορεί να εκτελείται ανεξάρτητα και, για το λόγο αυτόν , μπορούν να εκτελούνται παράλληλα.[13]

Οταν καλεί έναν πυρήνα , ο προγραμματιστής καθορίζει τον αριθμό των νημάτων ανά μπλοκ και τον αριθμό των μπλοκ που αποτελούν το πλέγμα . [13]

Κάθε νήμα λαμβάνει ενα μοναδικό αριθμό ταυτότητας νήματος (thread ID) threadIdx μέσα στο μπλοκ νημάτων που ανήκει , με αρίθμηση 0,1,2, ... blockDim-1, και κάθε μπλόκ νημάτων λαμβάνει ενα μοναδικό αριθμό ταυτότητας μπλοκ (block ID) blockIdx μέσα στο πλέγμα που ανήκει. [13] Η CUDA υποστηρίζει μπλοκ νημάτων που περιέχουν έως και 512 νήματα . Για ευκολία , τα μπλοκ νημάτων και τα πλέγματα μπορούν να έχουν 1,2 ή 3 διαστάσεις που προσπελάζονται μέσω των πεδίων αριθμοδείκτη (index fields). x,.y. και .z.

Στην παρακάτω εικόνα εχουμε το πως είναι οργανωμένα τα νήματα της CUDA .Το πλέγμα αποτελείται απο N μπλοκ νημάτων, καθένα έχει μια τιμή blockIdx.x με εύρος τιμών απο 0 έως N – 1 . Το κάθε μπλοκ τώρα με την σειρά του , αποτελείται απο M νήματα , που το καθένα εχει μια τιμή threadIdx.x με εύρος τιμών απο 0 εως M – 1. Λέμε επίσης ότι όλα τα μπλοκ σε επίπεδο πλέγματος τα εχουμε οργανώσει με την μορφή μονοδιάστατου πίνακα και όλα τα νήματα μέσα στο μπλοκ είναι επίσης οργανωμένα με την μορφή ενός μονοδιάστατου πίνακα. [12] Κάθε πλέγμα διαθέτει ένα σύνολο N * M νημάτων.[12]



Σχήμα 1.5.1

πηγή

https://cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/lecture5.pdf

Το μάυρο πλαίσιο κάθε μπλοκ νημάτων στην παραπάνω εικόνα δείχνει ένα απόσπα-σμα του κώδικα του πυρήνα. [12] Το απόσπασμα του κώδικα χρησιμοποιεί την threadID = blockIdx.x * blockDim.x + threadIdx για να προσδιορίσει το τμήμα των δεδομένων εισόδου από όπου θα διαβάσει, και το τμήμα των δεδομένων εξόδου στα οποία θα γράψει. [12] Το νήμα 3 του μπλοκ 0 έχει τιμή threadID ίση με 0 * M + 3 = 3. Το νήμα 3 του μπλοκ 5 έχει τιμή threadID ίση με 5 * M + 3.[12]

Εαν για παράδειγμα ένα πλέγμα έχει 128 μπλοκ (N = 128) και κάθε μπλοκ έχει 32 νήματα (M = 32). [12] Αρα λοιπόν στο παράδειγμα αυτό , η προσπέλαση στην μεταβλητή blockDim στον πυρήνα δίνει 32. Ετσι λοιπόν στο πλέγμα υπάρχουνε συνολικά 128 * 32 = 4096 νήματα. [12]

Το νήμα 3 του μπλοκ 0 έχει τιμή νημάτωσης (threaded value) 0 * 32 + 3 = 3. Το νήμα 3 του μπλοκ 5 έχει τιμή νημάτωσης (threaded value) 5 * 32 + 3 = 163. [12] Το νήμα 15 του μπλοκ 102 έχει τιμή νημάτωσης (threaded value) 102 * 32 + 15 = 3279 . [12] Μπορούμε να πούμε ότι καθένα απο τα 4096 νήματα έχει την δική του μοναδική τιμή νημάτωσης. Στην παραπάνω εικόνα ,ο κώδικας του πυρήνα χρησιμοποιεί την μεταβλητή threadID για την αριθμοδεικτοδότηση των πινάκων input [ ] και output [ ] . Αν πούμε ότι και οι δυο πίνακες δηλώνονται με 4096 στοιχεία, τότε κάθε νήμα θα παίρνει ένα απο τα στοιχεία εισόδου και θα παράγει ένα απο τα στοιχεία εξόδου. [12]

Γενικά , ένα πλέγμα είναι οργανωμένο ώς ένας δισδιάστατος πίνακας μπλοκ . [12] Κάθε μπλοκ είναι οργανωμένο ώς ένας τρισδιάστατος πίνακας νημάτων. [12] Η ακριβής οργά-νωση του πλέγματος καθορίζεται απο τις παραμέτρους της εκτέλεσης που παρέχονται κατά την εκκίνηση του πυρήνα. Η πρώτη παράμετρος της εκτέλεσης καθορίζει τις δια-στάσεις του πλέγματος συναρτήσει του αριθμού των μπλοκ. [12] Η δεύτερη καθορίζει τις διαστάσεις κάθε μπλοκ συναρτήσει του αριθμού των νημάτων .[12] Κάθε τέτοια παράμε-τρος είναι τύπου dim3, που είναι μια δομή struct της **C** με τρια πεδία απρόσημων ακε-ραίων: x, y και z. Επειδή τα πλέγματα είναι δισδιάστατοι πίνακες των διαστάσεων των μπλοκ, το τρίτο πεδίο της παραμέτρου διάστασης του πλέγματος αγνοείται για λόγους σαφήνειας πρέπει να τεθεί ίσο με με 1. [12] Ο επόμενος κώδικας στην CPU μπορεί να χρησιμοποιηθεί για την εκκίνηση του πυρήνα του οποίου η οργάνωση φαίνεται στο παραπάνω σχήμα. [12]

dim3 dimGrid(128, 1, 1);

dim3 dimBlock(32, 1, 1);

KernelFunction <<<dimGrid, dimBlock>>> (. . .);

Οι πρώτες δυο εντολές δίνουν αρχικές τιμές στις παραμέτρους εκκίνησης.[12] Λόγο του ότι το πλέγμα και το μπλοκ είναι μονοδιάστατοι πίνακες , χρησιμοποιείται μόνον η πρώτη διάσταση των dimBlock και dimGrid. Οι άλλες διαστάσεις παίρνουν την τιμή 1. Η τρίτη εντολή είναι η πραγματική εκκίνηση του πυρήνα. [12] Οι παράμετροι εκτέλεσης τοπο-θετούνται ανάμεσα στα σύμβολα <<< και >>> .[12] Μπορούμε επίσης να χρησιμο-ποιήσομε και βαθμωτές τιμές για τις παράμετρους εκτέλεσης αν ένα πλέγμα ή μπλοκ έχει μόνο μια διάσταση για παράδειγμα , το ίδιο πλέγμα μπορεί να ξεκινήσει με μια μόνο εντολή:

KernelFunction<<<128, 32>>> (. . .);

Οι τιμές των gridDim.x και gridDim.y μπορούν να κυμαίνονται απο 1 έως 65535. [12] Οι τιμές των gridDim.x και gridDim.y μπορούν να υπολογιστούν με βάση άλλες μεταβλητές κατά την εκκίνηση του πυρήνα.[12] Μετά απο την εκκίνηση ενός πυρήνα , οι διαστάσεις δεν είναι δυνατόν να αλλάξουν. Ολα τα νήματα ενός μπλοκ μοιράζονται την ίδια τιμή blockIdx. Η τιμή blockIdx.x κυμαίνονται μεταξύ 0 και gridDim.x – 1 , και η τιμή blockIdx.y μεταξύ 0 και gridDim.y – 1. [12]

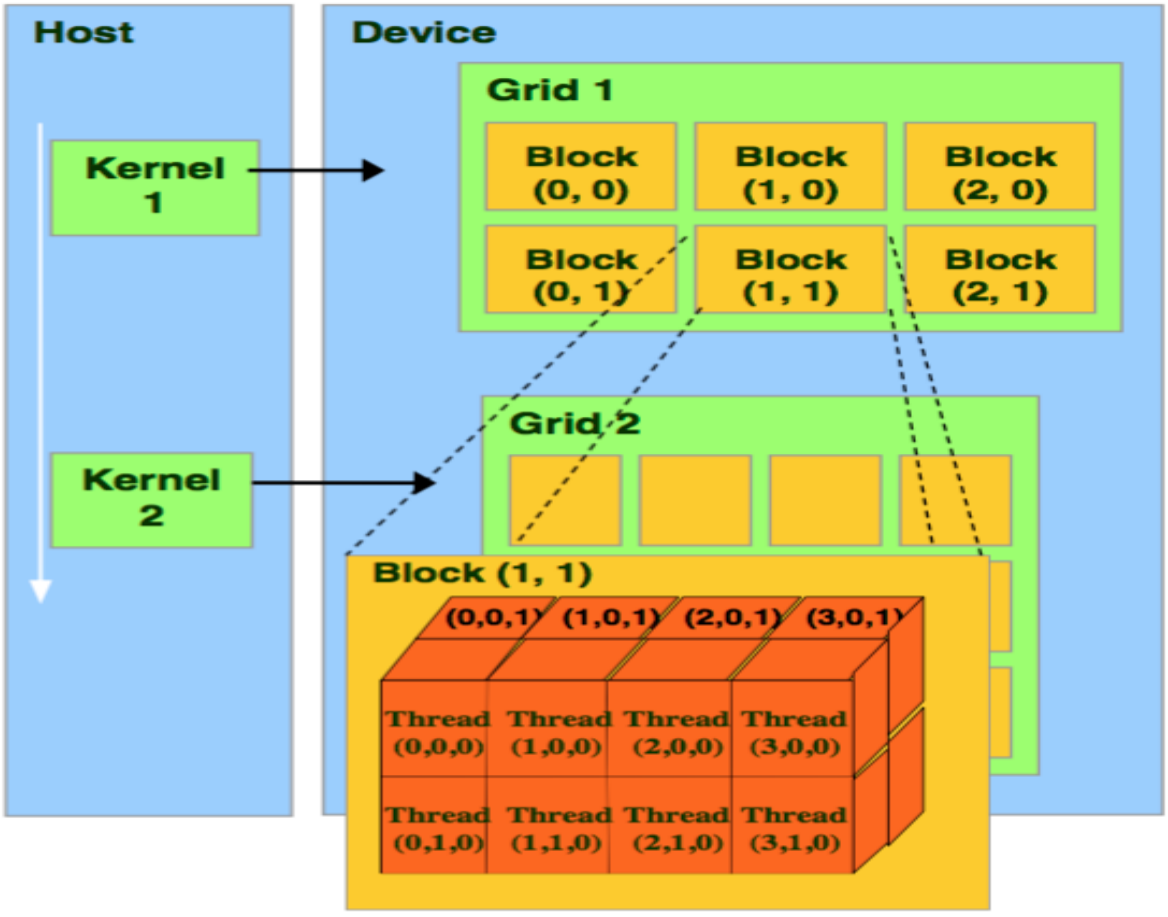
Παρακάτω βλέπουμε ένα μικρό πλέγμα δυο διαστάσεων που ξεκινάει με τον παρακάτω κώδικα CPU :

dim3 dimGrid (2, 2, 1);

dim3 dimBlock (4, 2, 2);

KernelFunction <<< dimGrid, dimBlock >>> (. . .);

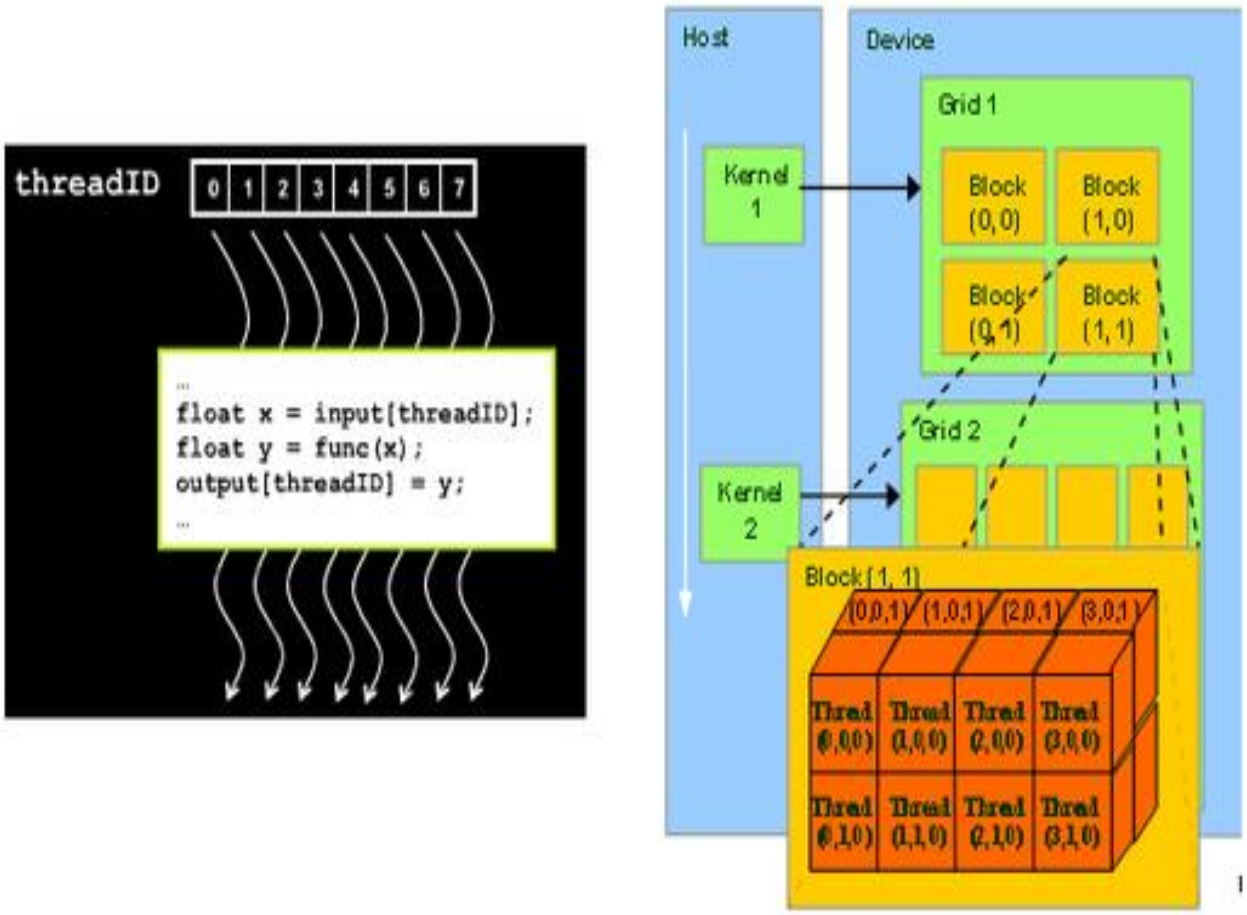
Το πλέγμα αποτελείται απο τέσσερα μπλοκ οργανωμένα σε έναν πίνακα 2 χ 2. [12] Κάθε μπλοκ είναι γραμμένο με (blockIdx.x, blockIdx.y), έστω για παράδειγμα πού-με ότι το μπλοκ (1, 0) έχει τιμές blockIdx.x = 1 και blockIdx.y = 0.



Σχήμα 1.5.2

πηγή

https://nyu-cds.github.io/python-gpu/02-cuda/



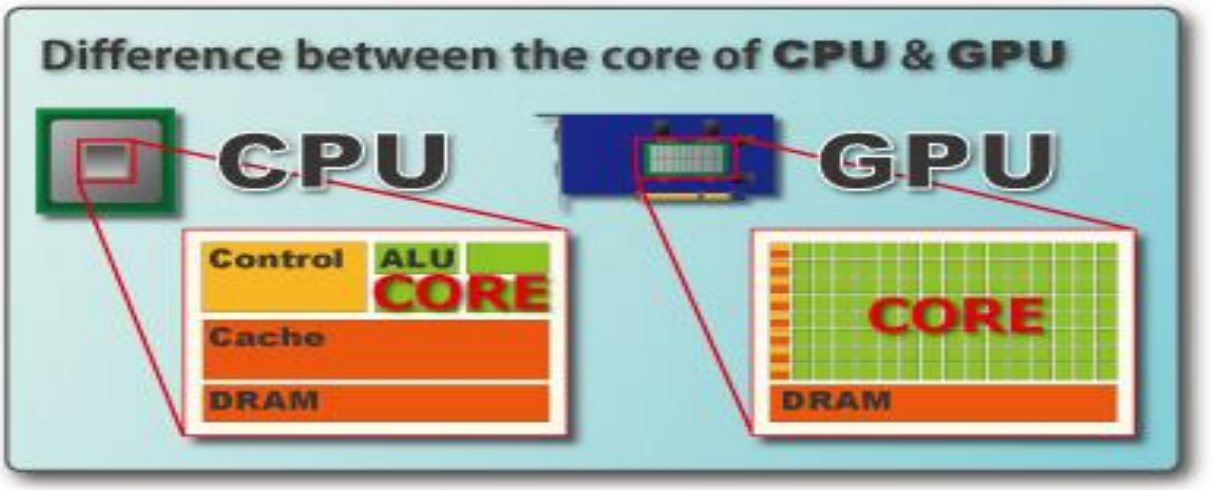
Σχήμα 1.5.3

πηγή

https://portal.futuresystems.org/manual/gpu/running-programs-single-gpu

Οπως είπαμε λοιπόν οτι τα μπλοκ είναι οργανωμένα ως τρισδιάστατοι πίνακες νημάτων. Οπότε και τα μπλοκ ενός πλέγματος έχουν τις ίδιες διαστάσεις.[12] Κάθε **threadIdx** αποτελείται απο τρεις συνιστώσες: τη συντεταγμένη **xthreadIdx.x** τη συντεταγμένη **ythreadIdx.y** τη συντεταγμένη **zthreadIdx.z.** [12]. Ο αριθμός των νημάτων σε κάθε διάσταση ενός μπλοκ καθορίζεται απο την δεύτερη παράμετρο εκτέλεσης που δίνεται κατά την εκκίνηση του πυρήνα.[12]. Με τον πυρήνα , αυτή η παράμετρος εκτέλεσης μπορεί να προσπελαστεί ως εκ των προτέρων ορισμένη μεταβλητή **blockDim** τύπου struct. [12]. Το συνολικό μέγεθος ενός μπλοκ πάει στα 512 νήματα, με ελευθερία στην κατανομή των στοιχείων αυτών στις τρεις διαστάσεις εφόσον ο συνολικός αριθμός των νημάτων δεν υπερβαίνει τα 512. [12]. Εάν οι (512,1,1) = 512, (8,16,4) = 512 , και (16,16,2) = 512 είναι όλες αποδεχτές τιμές της **blockDim,** αλλά η (32,32,1) δεν έιναι αποδεχτό διότι έχουμε πάνω απο 512 νήματα ,πάει στα 1024 νήματα. **Δίοτι έχουμε 32 χ 32 = 1024 νήματα.** Επίσης στη παραπάνω εικόνα – σχήμα βλέπουμε την οργάνωση των νημάτων μέσα σε ενα μπλοκ . Στο παράδειγμα αυτό , **κάθε μπλοκ είναι οργανωμένο σε πίνακες νημάτων 4 χ 2 χ 2 ,κατεύθυνση κατά x = 4 ,κατά y = 2 ,και κατά z = 2**. [12]. Και επειδή όλα τα μπλοκ ενός πλέγματος έχουν τις ίδιες διαστάσεις , πρέπει να δείξουμε ένα απο αυτά.

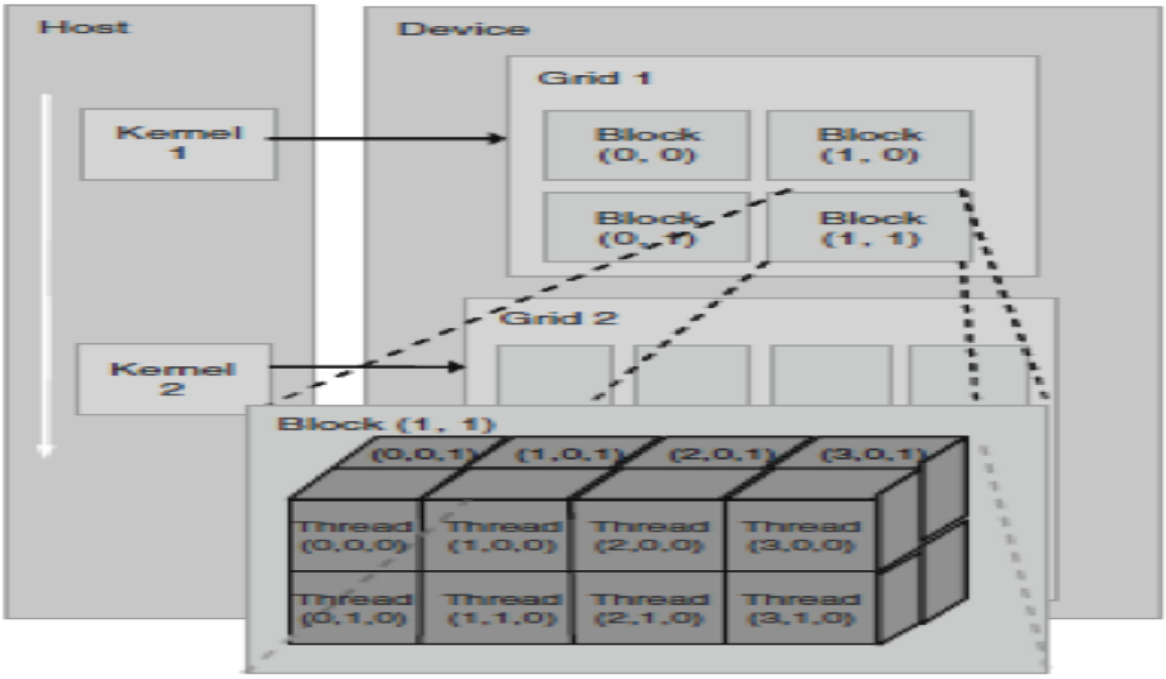
Το **μπλοκ (1,1)** έχει αναπτυχθεί ,όπως φαίνεται στο παραπάνω σχήμα , για να φανούνε τα 16 νήματα που περιέχει το παράδειγμα, **το νήμα (2,1,0**) έχει **threadIdx.x = 2, threadIdx.y = 1,** και **threadIdx.z = 0.** [12]. Εχουμε 4 μπλοκ με 16 νήματα το καθένα , άρα 64 νήματα στο πλέγμα.



Σχήμα 1.5.4

πηγή

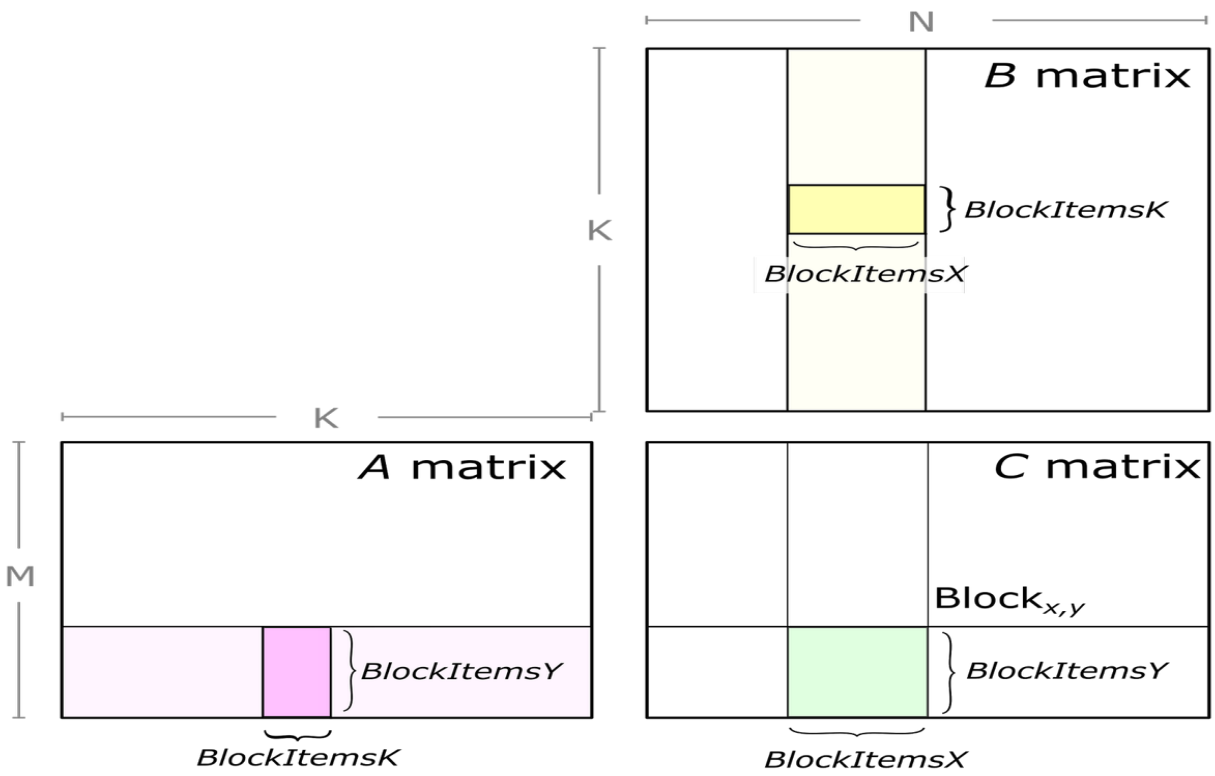
https://loilo.tv/us/product/5/desc/82



Σχήμα 1.5.5

πηγή

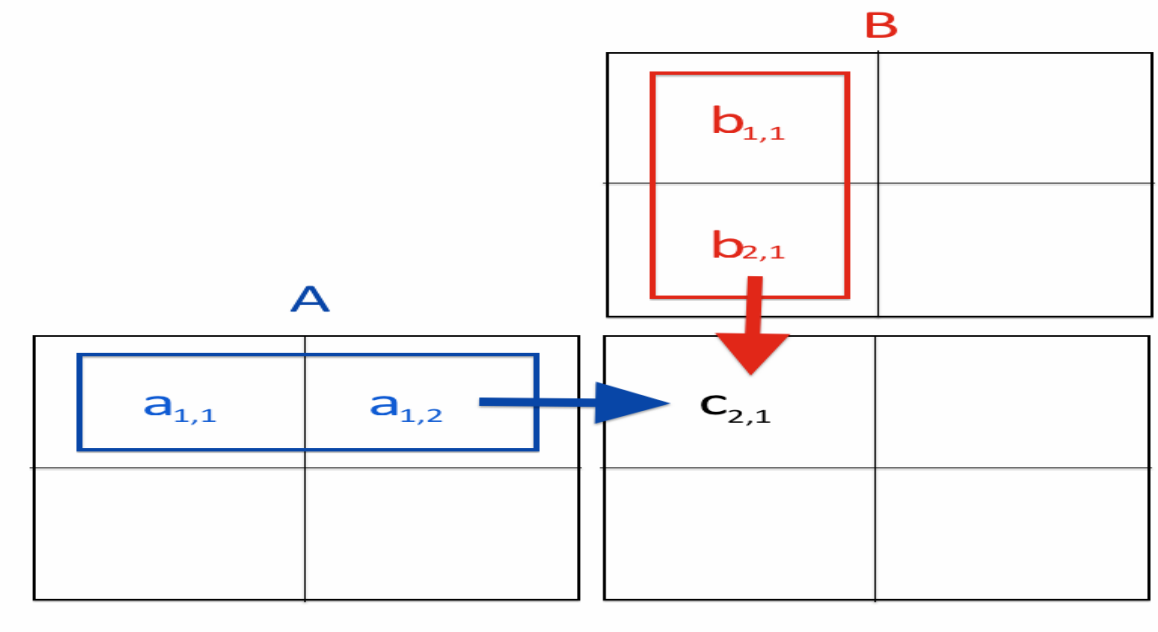
http://onceuponanengineer.blogspot.com/2013/09/a-multidimensional-example-of-cuda-grid.html



Σχήμα 1.5.6

πηγή

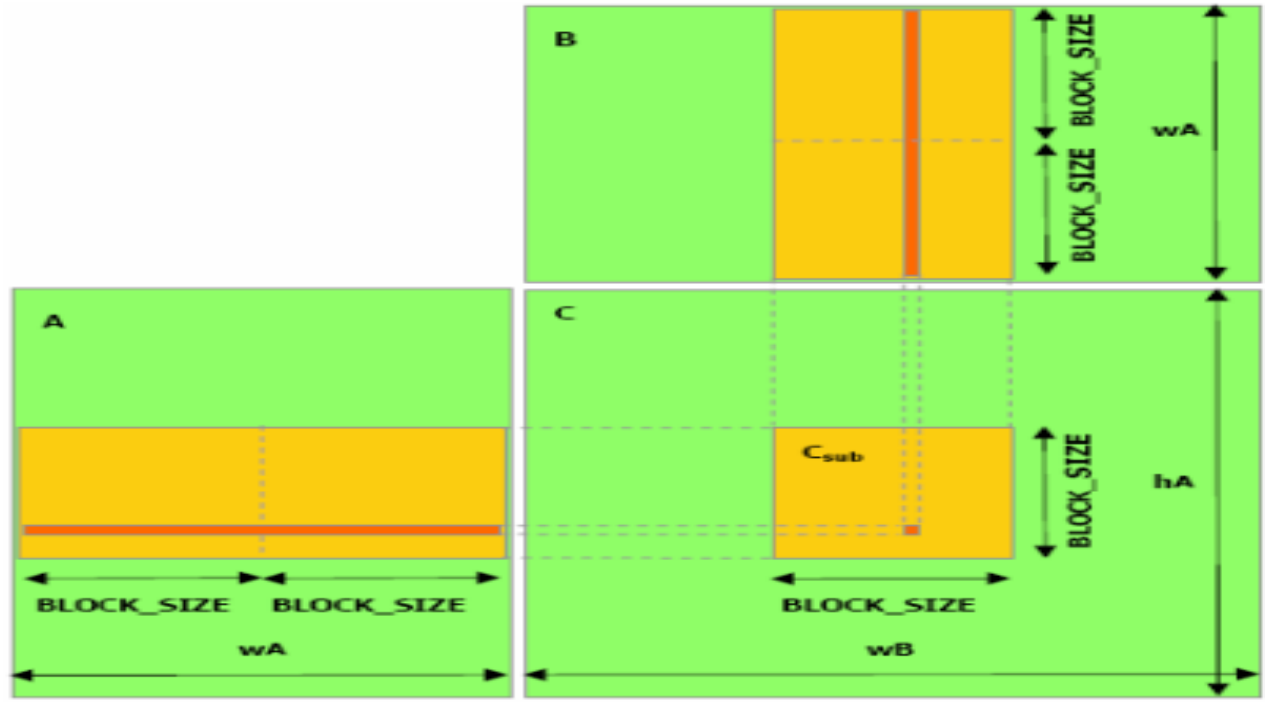
https://twitter.com/harrism/status/938468567657443330



Σχήμα 1.5.7

πηγή

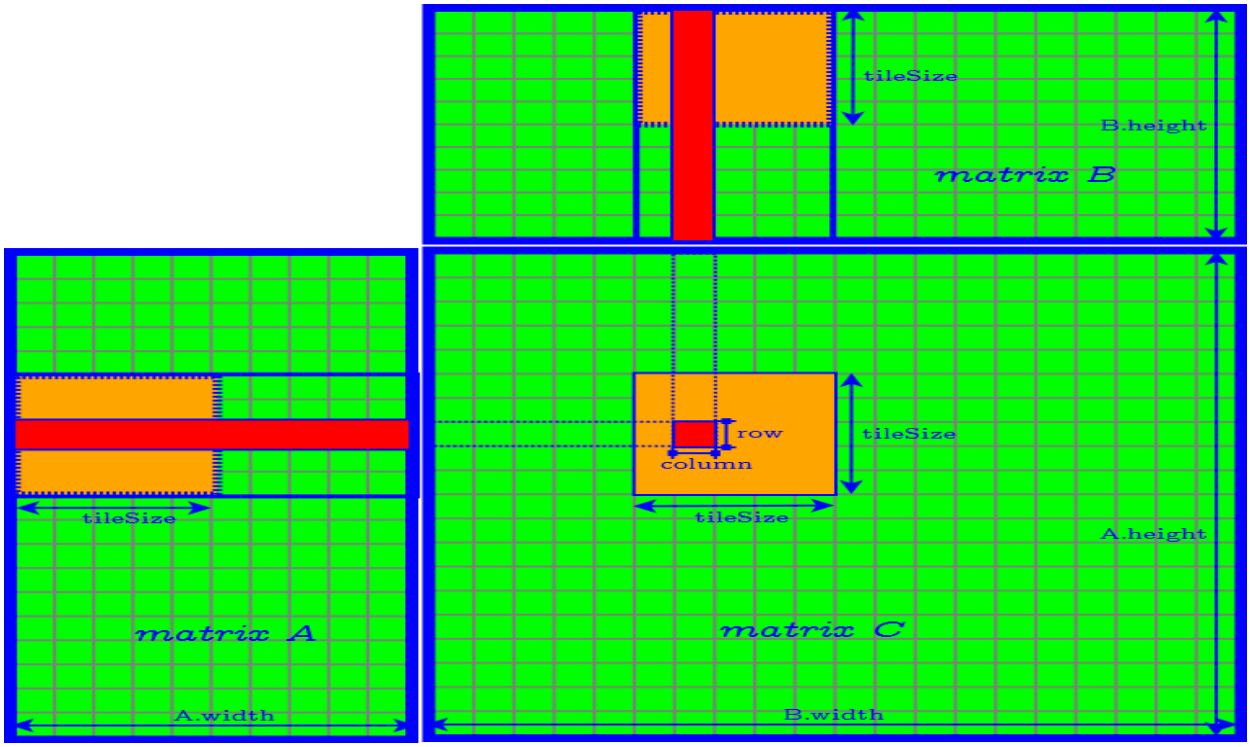
https://www.quantstart.com/articles/Matrix-Matrix-Multiplication-on-the-GPU-with-Nvidia-CUDA



Σχήμα 1.5.8

πηγή

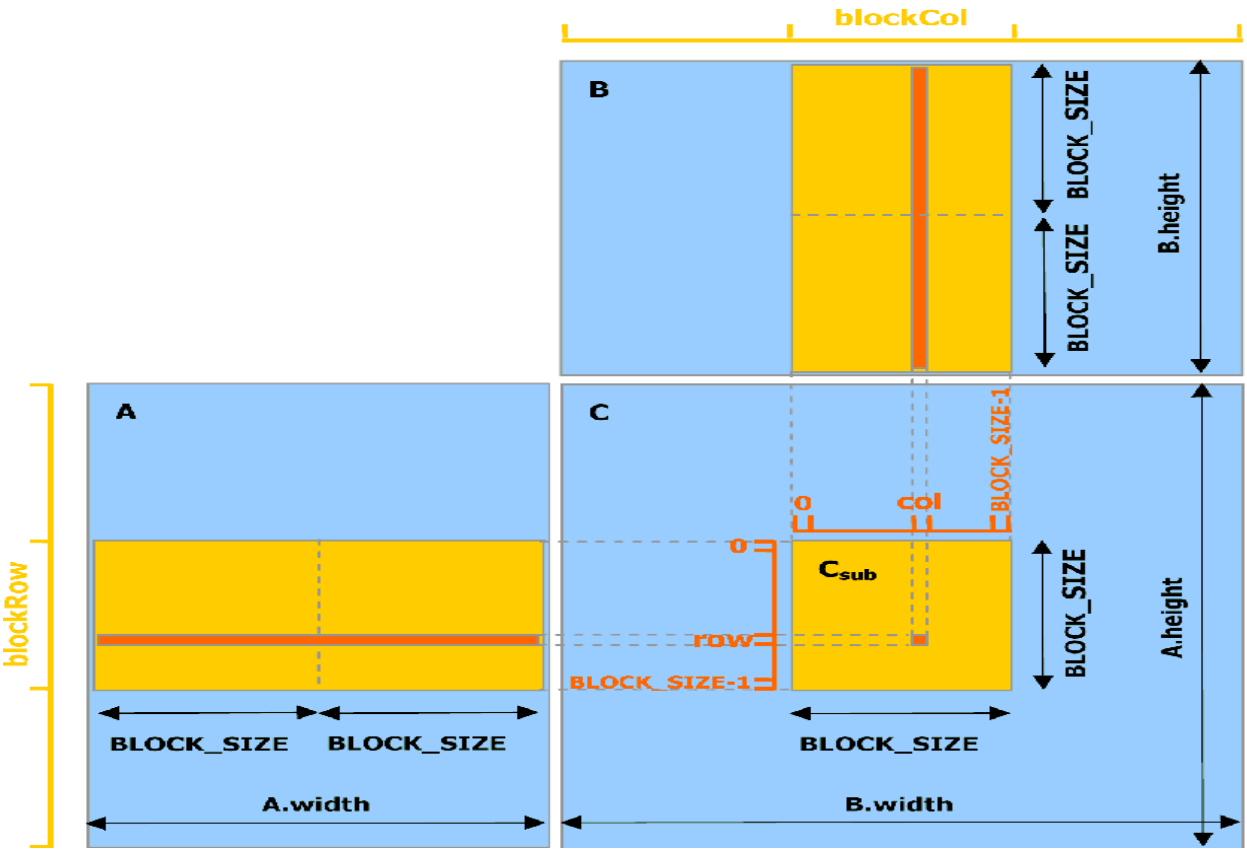
https://www.beyond3d.com/content/articles/12/5



Σχήμα 1.5.9

πηγή

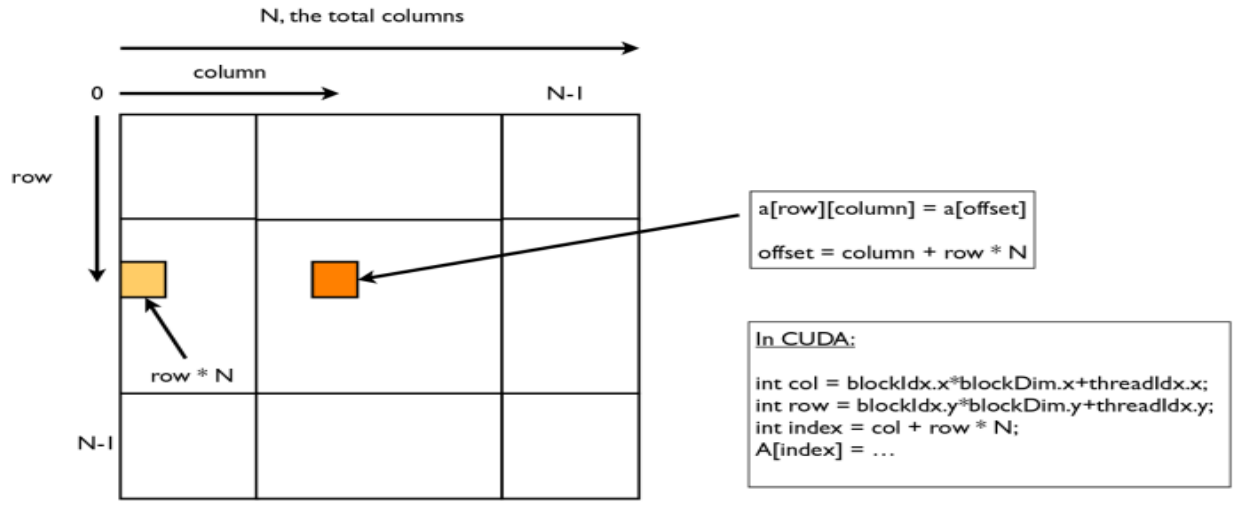
http://yszheda.github.io/GPU-RSCode/#/



Σχήμα 1.5.10

πηγή

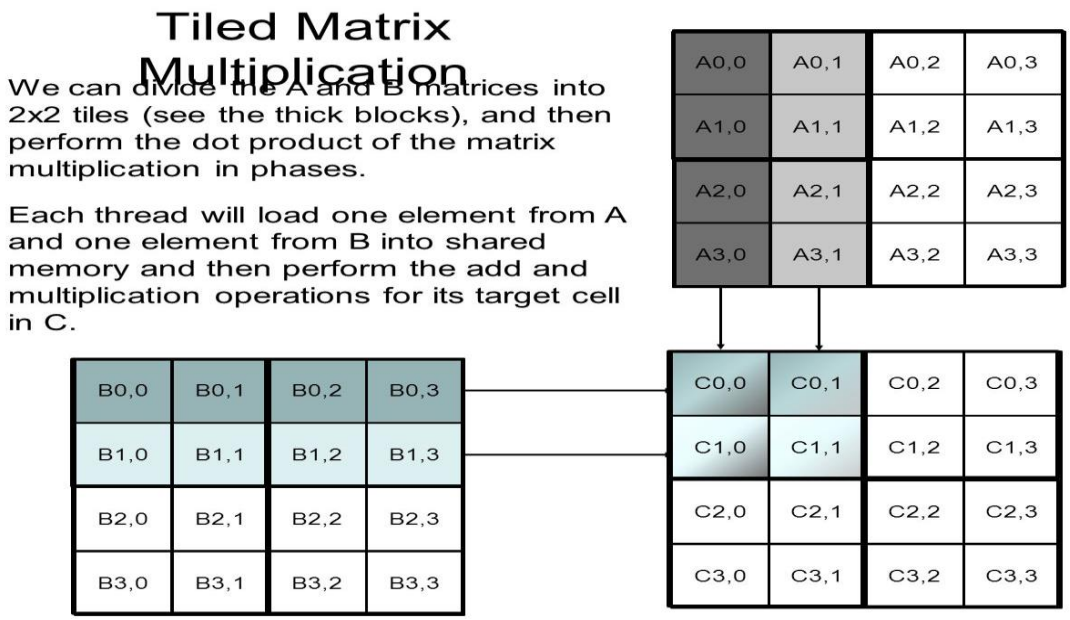
https://www.semanticscholar.org/paper/General-Purpose-Computation-on-Graphics-Processing-Zibula/33f7154128da586d4726e809ede877448c6a8d2e/figure/9



Σχήμα 1.5.11

πηγή

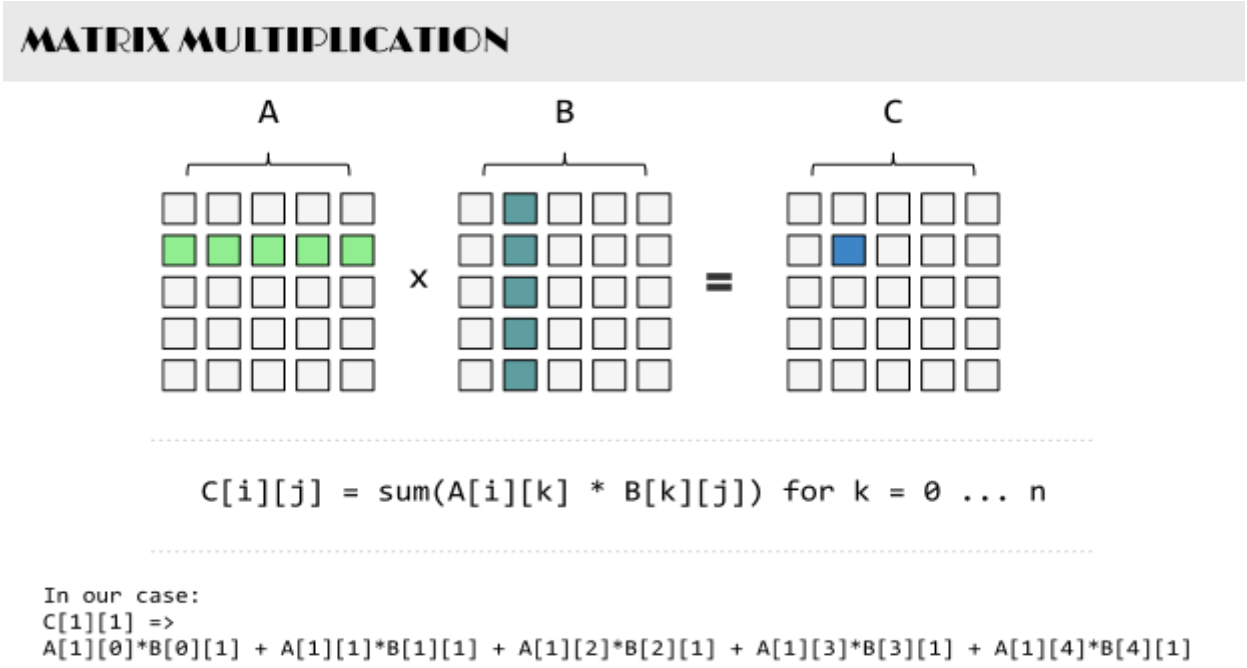
https://eilianyu.wordpress.com/2016/12/04/pycuda-and-gpu-computing-experiments/



Σχήμα 1.5.12

πηγή

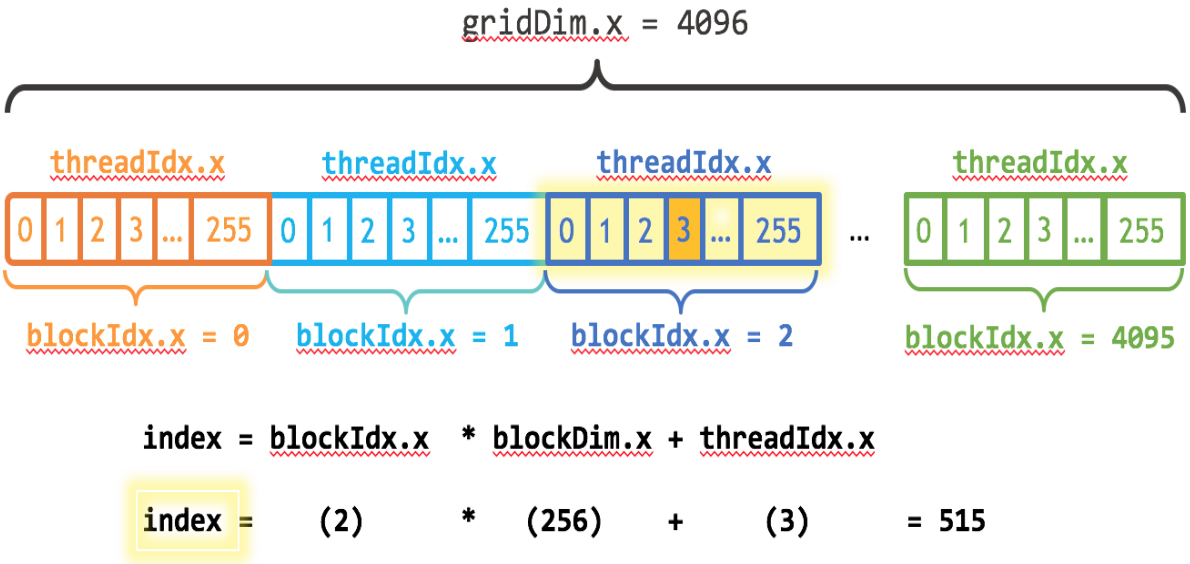
https://slideplayer.com/slide/9398454/



Σχήμα 1.5.13

Πηγή

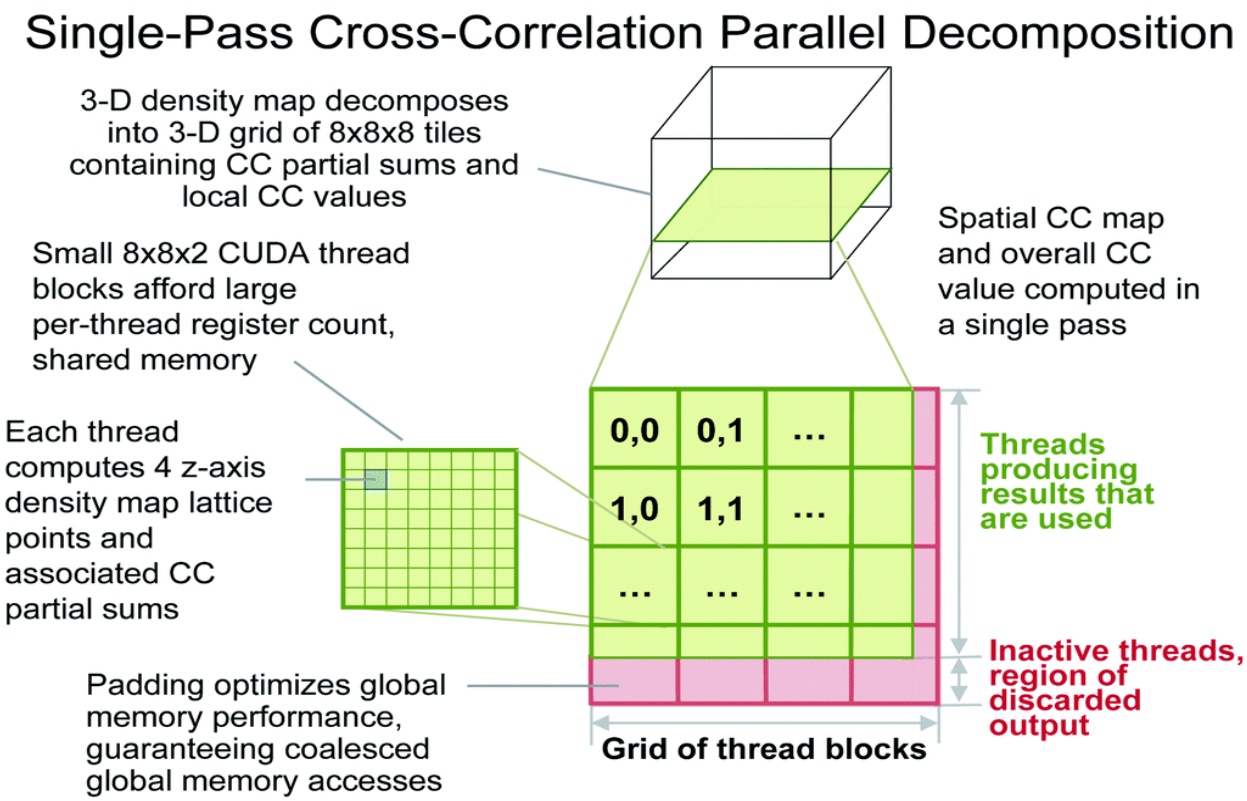
https://insidehpc.com/2016/01/heterogeneous-streams/



Σχήμα 1.5.14

πηγή

https://devblogs.nvidia.com/even-easier-introduction-cuda/cuda_indexing/

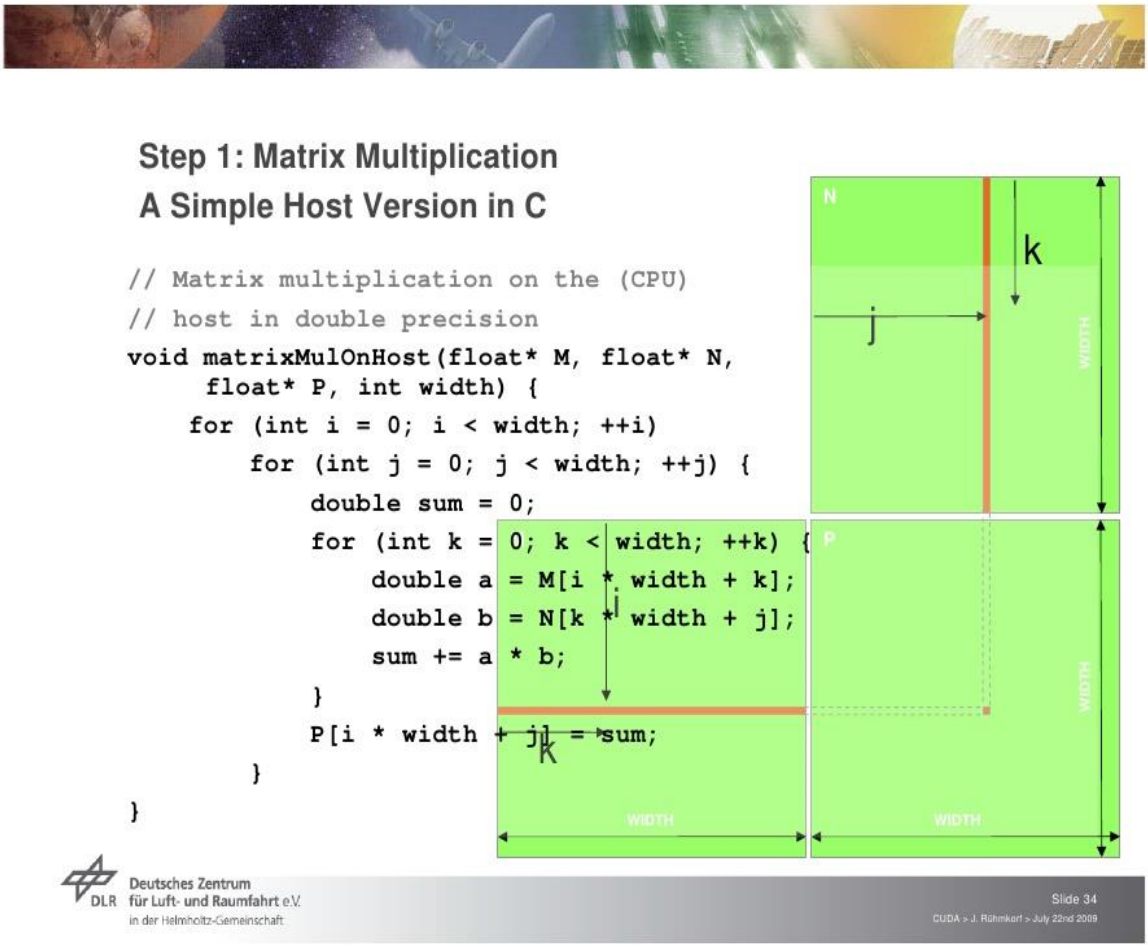


Σχήμα 1.5.15

πηγή

https://pubs.rsc.org/en/content/articlelanding/2014/fd/c4fd00005f/unauth#!divAbstract

Μια συνήθης χρήση των **threadIdx** και **blockIdx** είναι για τον καθορισμό του τμήματος δεδομένων που πρόκειται να επεξεργαστεί κάθε νήμα. [12] Το βλέπουμε στο κώδικα του παρακάτω σχήματος όπου έχουμε πολλαπλασιασμό μήτρας. Οπου το **i** είναι το **threadIdx.y** και το **j** είναι το **threadIdx.x .** Το **M ειναι η γραμμή και η N η στήλη.** Περιορίζεται μόνο στα 16 νήματα διότι δεν εχει το **blockIdx,** αρα τυπικά είναι ενα μπλοκ νημάτων. Θα πρέπει να γνωρίζουμε οτι ενα μπλοκ έχει 512 νήματα. Κάθε νήμα υπολογίζει ένα στοιχείο της **P,** μπορούμε να υπολογίσουμε μέχρι 512 στοιχεία του **P** με τον κώδικα. Για τετράγωνες μήτρες περιοριζόμαστε σε 16 χ 16 επειδή οι διαστάσεις 32 χ 32 θέλουνε πάνω απο 512 νήματα ανά μπλοκ. [12].



Σχήμα 1.5.16

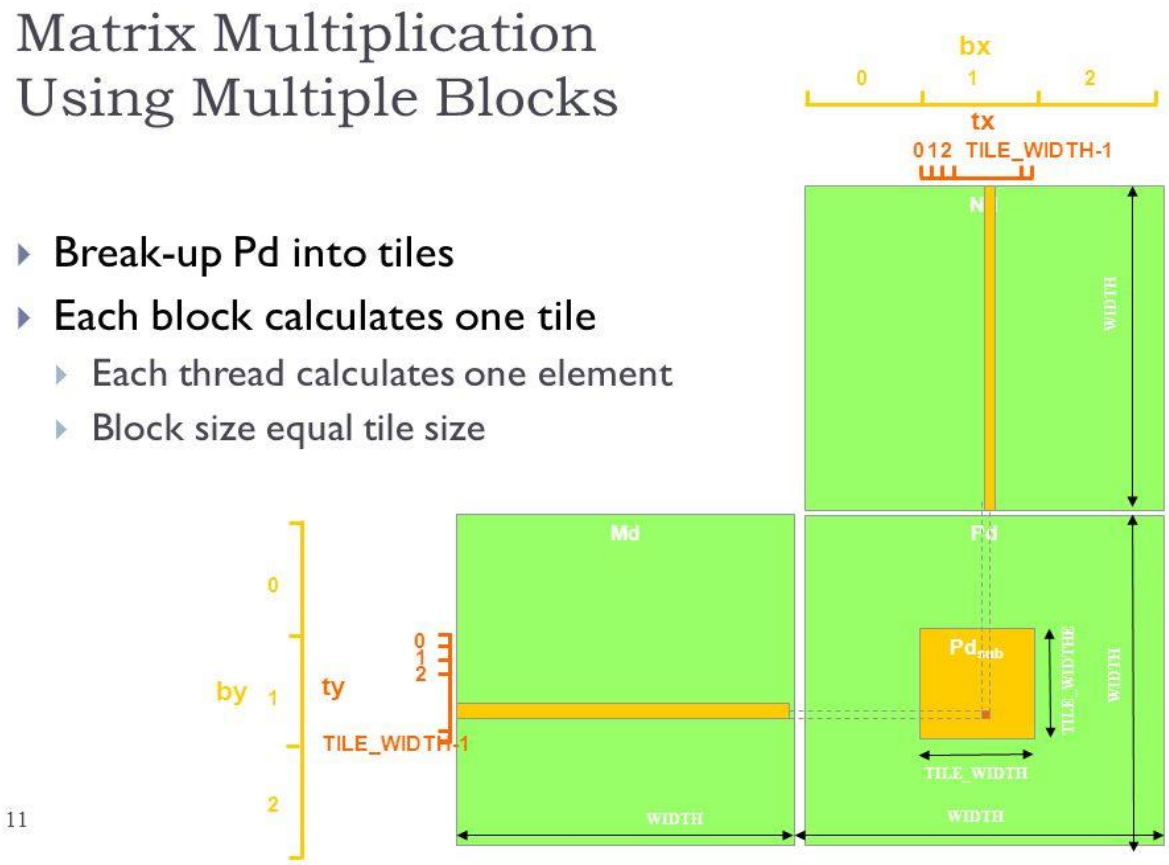
πηγή

https://www.slideshare.net/ruehmkorf/tech-talk-nvidia-cuda

Εάν θέλουμε να καλύψουμε μεγαλύτερες μήτρες θα πρέπει να χρησιμοποιήσουμε πολλά μπλοκ νημάτων. Στα παραπάνω σχήματα φαίνεται πως υλοποιείται. Θα πρέπει να **διαιρέσουμε το P σε πλακίδια (tiles).** Όλα τα στοιχεία του P που ανήκουν σε ένα

πλακίδιο υπολογίζονται απο ένα μπλοκ νημάτων.[12]. Με μικρές τις διαστάσεις αυτών των πλακιδίων του P ,πετυχαίνουμε τον συνολικό αριθμό των νηματων σε κάθε μπλοκ κάτω απο 512.

Κάθε νήμα υπολογίζει και πάλι ένα στοιχείο του **P** . Με την μόνη διαφορά ότι πρέπει να χρησιμοποιήσει τις τιμές **blockIdx** του στοιχείου για να προσδιορίσει το πλακίδιο που περιέχει το στοιχείο, πριν χρησιμοποιήσειτις τιμές **threadIdx** για να προσδιορίσει το στοιχείο μέσα στο πλακίδιο.[12]. Δηλαδή , κάθε νήμα θα χρησιμοποιεί και την τιμή **threadIdx** και την τιμή **blockIdx** για να προσδιορίσει το στοιχείο της **P** που θα επεξεργαστεί. [12]. Το βλέπουμε στην παρακάτω εικόνα, **όπου οι τιμές bx, by, tx, και ty νημάτων που επεξεργάζονται τα στοιχεία του P σημειώνονται και στις δύο διαστάσεις x και y.** [12]. Ολα τα νήματα που υπολογίζουνε τα στοιχεία του P μέσα σε ένα πλακίδιο έχουνε τις ίδιες τιμές **blockIdx**.[12].



Σχήμα 1.5.17

πηγή

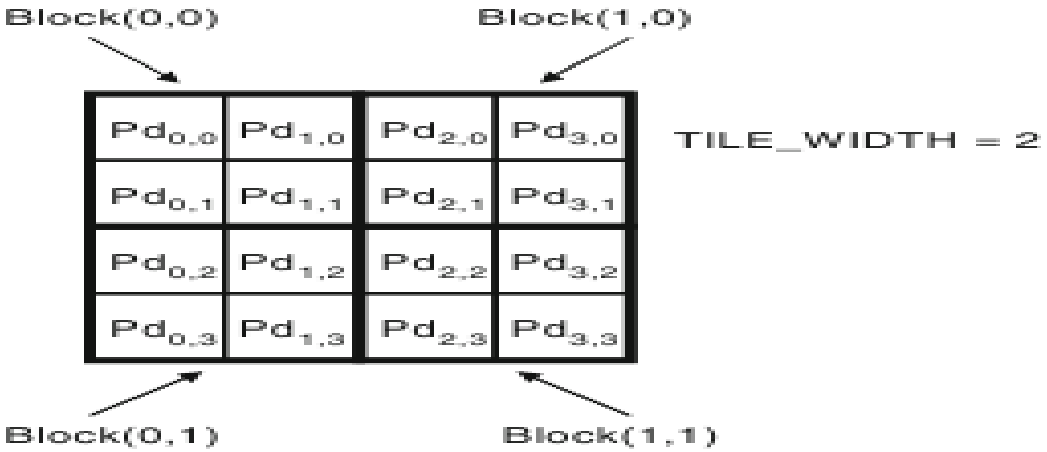
https://slideplayer.com/slide/8412195/

Εάν υποθέσουμε ότι έχουμε τετράγωνες διαστάσεις ενός μπλοκ που καθορίζονται απο την μεταβλητή **TILE_WIDTH (πλάτος πλακιδίου)** . Κάθε διάσταση του P διαιρείται τώρα σε τμήματα που το καθένα έχει **TILE_WIDTH** στοιχεία ,όπως το βλέπουμε στα παραπάνω σχήματα. Κάθε μπλοκ χειρίζεται ένα τέτοιο τμήμα. [12]. Ετσι , ένα νήμα μπορεί να βρει τον **αριθμοδείκτη x** του στοιχείου του στο **P** ως **(bx * TILE_WIDTH + tx)** και τον

αριθμοδείκτη y ως **(by * TILE_WIDTH + ty)**.[12]. Δηλαδή , **το νήμα (tx, ty)** στο μπλοκ **(bx, by)** θα χρησιμοποιήσει την γραμμή **(by * TILE_WIDTH + ty)** της **M** και την στήλη **(bx * TILE_WIDTH + tx)** της **N** για να υπολογίσει το στοιχείο του **P** που βρίσκεται στην στήλη **(bx * TILE_WIDTH + tx)** και την γραμμή **(by * TILE_WIDTH + ty).** [12].

Παρακάτω στην εικόνα έχουμε ενα παράδειγμα χρήσης πολλών μπλοκ για τον υπολογισμό του P . Εδώ η μήτρα P διαιρείται τώρα σε 4 πλακίδια . Κάθε διάσταση του P διαιρείται σε τμήματα των 2 στοιχείων. [12]. Κάθε μπλοκ πρέπει να υπολογίζει 4 στοιχεία του P .[12]. Αυτό μπορούμε να το κάνουμε δημιουργώντας μπλοκ οργανωμένα σε πίνακες 2 χ 2, όπου κάθε νήμα υπολογίζει ένα στοιχείο του P .[12]. Εδώ για παράδειγμα , το νήμα (0,0) του μπλοκ (0,0) υπολογίζει το στοιχείο P 0,0 , ενώ το νήμα (0,0) του μπλοκ (1,0) υπολογίζει το στοιχείο P 2,0 . Μπορούμε να το δούμε και με τον παρακάτω τύπο:

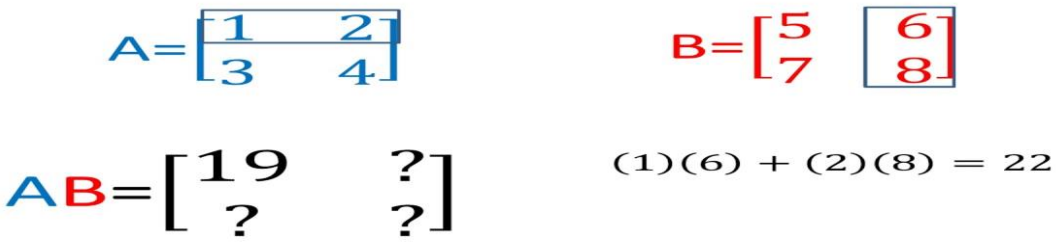
P [bx * TILE_WIDTH + tx] [by * TILE_WIDTH + ty] = P [1 * 2 + 0] [0 * 2 + 0] = P [2] [0]



Σχήμα 1.5.18

πηγή

http://marks-cs.blogspot.com/2010/11/



Σχήμα 1.5.19

πηγή

https://www.youtube.com/watch?v=yaXnkWMHAIo

Οταν προσδιορίζουμε τους αριθμοδείκτες του στοιχείου του **P** που θα υπολογίσει ενα νήμα, θα έχουμε επίσης προσδιορίσει τον αριθμοδείκτη της γραμμής (y) **Md** και τον αριθμοδείκτη της στήλης (x) **Nd** για τις τιμές εισόδου. Οπως φαίνεται στην παραπάνω εικόνα (Σχήμα 1.5.17), ο αριθμοδείκτης γραμμής της **Md** που χρησιμοποιεί το νήμα **(tx, ty)** του μπλοκ **(bx, by)** είναι **(by * TILE_WIDTH + ty)**. [12]. Ο αριθμοδείκτης στήλης της **Nd** που χρησιμοποιεί το ίδιο νήμα είναι **(bx * TILE_WIDTH + tx)** .[12].

Σύμφωνα με το Σχήμα 1.5.12 βλέπουμε τις ενέργειες του πολλαπλασιασμού σε κάθε μπλοκ νημάτων.[12]. Για τον πολλαπλασιασμό των μικρών μητρών, τα νήματα του **μπλοκ (0,0)** παράγουν τέσσερα εσωτερικά γινόμενα : **το νήμα (0,0)** παράγει το **Pd0,0** υπολογίζοντας το εσωτερικό γινόμενο της **γραμμής 0** της **Md** και της **στήλης 0** της **Nd.** [12]. **Το νήμα (1,0**) παράγει το **Pd1,0** υπολογίζοντας το εσωτερικό γινόμενο της **γραμμής 0** της **Md** και της **στήλης 1** της **Nd**. [12].

Όπως είναι γνωστό, η αρχιτεκτονική του μοντέλου CUDA βασίζεται στους Streaming Multiprocessors (SMs) οι οποίοι μπορούν να εκτελούν εκατοντάδες νήματα ταυτόχρονα. [15]. Οι επεξεργαστές αυτοί υλοποιούν μια συγκεκριμένη αρχιτεκτονική χάρη στην οποία έχουν τη δυνατότητα να εκτελούν ένα μεγάλο αριθμό παράλληλων νημάτων. [15]

Αυτή η αρχιτεκτονική ονομάζεται SIMT (Single Instruction, Multiple Thread) χρησιμοποιεί πολυνημάτωση και διοχετεύσεις εντολών, οι οποίες, αντίθετα με τις διοχετεύσεις τις CPU, εκτελούνται με αυστηρή σειρά χωρίς υποθετική εκτέλεση. [15]

Ένας SM δημιουργεί, χειρίζεται και προγραμματίζει την εκτέλεση των νημάτων, αφού πρώτα τα οργανώσει σε ομάδες των 32 νημάτων που ονομάζονται warp (στημόνια-σμήνη). [15] Ο ξενόγλωσσος όρος warp έναντι του αντίστοιχου ελληνικού όρου (σμήνος) είναι πιο δόκιμος και επομένως θα χρησιμοποιείται και στη συνέχεια υποκεφαλαίου. [15] Τα νήματα που αποτελούν ένα warp ξεκινούν από την ίδια διεύθυνση προγράμματος, αλλά το καθένα από αυτά διαθέτει τον δικό του καταχωρητή και μετρητή εντολής, έτσι ώστε να μπορεί να ακολουθεί διαφορετικό μονοπάτι εκτέλεσης σε σχέση με τα υπόλοιπα νήματα. [15] Ως δόκιμοι όροι χρησιμοποιούνται επίσης και το half-warp (το μισό warp) και το quarter-warp (τέταρτο του warp). [15]

Η διαδικασία ανάθεσης νημάτων στους SM (Streaming Multiprocessors)είναι απλή
1. Όταν ένας πυρήνας καλείται, δημιουργεί ένα πλέγμα (grid). [15]

2. Το πλέγμα οργανώνεται σε μπλοκ νημάτων και τα μπλοκ αυτά στη συνέχεια διανέμονται στους SM, ανάλογα με την διαθεσιμότητα του κάθε SM και τις δυνατότητες να φιλοξενήσουν ένα ή περισσότερα μπλοκ νημάτων. [15]
3. Στη συνέχεια οι SM οργανώνουν τα μπλοκ νημάτων σε warps, ξεκινώντας με το πρώτο warp (στημόνι) που αρχίζει με το νήμα 0 και συνεχίζοντας με τα υπόλοιπα νήματα κατά αύξοντα αριθμό. Τα warp που σχηματίζονται προγραμματίζονται για εκτέλεση από έναν warp-scheduler. [15]

Τα 32 νήματα τα οποία συμμετέχουν σε ένα warp εκτελούν την ίδια εντολή και έτσι η μέγιστη παραλληλία παρατηρείται στις περιπτώσεις όπου όλα τα νήματα ακολουθούν το ίδιο μονοπάτι εκτέλεσης.[15] Σε διαφορετική περίπτωση, η πορεία εκτέλεσης που ακολουθεί κάθε νήμα εκτελείται σειριακά αναιρώντας τις παράλληλες δυνατότητες του επεξεργαστή. [15] Κάθε warp (στημόνι) εκτελείται τελείως παράλληλα και ανεξάρτητα σε σχέση με τα υπόλοιπα warp, ασχέτως αν εκτελούν τον ίδιο ή διαφορετικά κομμάτια κώδικα. [15]
Το περιεχόμενο εκτέλεσης (καταχωρητές, μετρητές προγράμματος κλπ) για κάθε warp που εκτελείται σε έναν SM αποθηκεύεται στην μνήμη του SM (στους καταχωρητές και την κοινόχρηστη μνήμη) και παραμένει εκεί μέχρι ότου ολοκληρωθεί η εκτέλεση του συγκεκριμένου warp. [15]. Με αυτόν τον τρόπο είναι εύκολη η εναλλαγή από το ένα warp στο άλλο προκειμένου να μην υπάρχει μεγάλο διάστημα κατά το οποίο ένας SM παραμένει αδρανής. Λόγω των περιορισμένων καταχωρητών [15]

Οπως έχουμε αναφέρει και προτίτερα η CUDA χρησιμοποιεί νήματα, η νηματοσυνάρτηση που θα εκτελεστεί στους πυρήνες δεν μπορεί να δεσμεύσει δυναμικά μνήμη κατά τον χρόνο εκτέλεσης (run time) και , επίσης θα πρέπει να λεχθεί ότι δεν υπάρχει απευθείας πρόσβαση στη μνήμη της κύριας διεργασίας απο τους πυρήνες λογισμικού (kernels) και αντίστροφα.[11] Για παράδειγμα εάν θέλουμε να προσθέσουμε 2 πίνακες ,τους A και B και το αποτέλεσμα τις άθροισης θα πάει στον C. Αρα λοιπόν θέλουμε τρεις πίνακες A,B,C με διάσταση N. Καταρχάς υπάρχουν κάποιοι περιορισμοί.

- *Η κεντρική μονάδα επεξεργασίας δεν έχει απευθείας πρόσβαση στην μνήμη της κάρτας γραφικών και αντίστροφα , οι επεξεργαστές της κάρτας γραφικών (GPU) δεν έχουν απευθείας πρόσβαση στην μνήμη του κεντρικού υπολογιστή.[11]*
- *Τα δεδομένα του προβλήματος βρίσκονται αποθηκευμένα στην εικονική μνήμη (virtual memory) της κυρίας διεργασίας (host process) που εκτελείται στη CPU.[11]*
- *Στην νηματοσυνάρτηση που θα εκτελεστεί στους πυρήνες (kernel function) δεν επιτρέπεται δυναμική δέσμευση μνήμης.[11]*
- *Στη νηματοσυνάρτηση που θα εκτελεστεί στους πυρήνες δεν επειτρέπεται η αναδρομή.[11]*
- *Για κάθε GPU υπάρχουν κατασκευαστικοί περιορισμοί ως προς το μέγιστο αριθμό διαστάσεων του πλέγματος και του μπλοκ , ως προς το μέγεθος της μνήμης , και ως προς τις δυνατότητες τις CUDA που υποστηρίζει.[11]*

Τα βήματα που απαιτούνται για την επεξεργασία είναι:

- *Προσδιορισμός αρχικών συνθηκών για το API της CUDA. Εδώ ελέγχουμε τον αριθμό των GPU που διαθέτει το σύστημα και τα κατασκευαστικά χαρακτηριστικά κάθε μιας απο αυτές.[11]*
- Δέσμευση μνήμης της κύριας διεργασίας ,CPU memory, και προετοιμασία των δεδομένων . Η δέσμευση της μνήμης πραγματοποιείται με τις συναρτήσεις δέσμευσης μνήμης malloc ,calloc, new ,το αντίστοιχο στην CUDA είναι το CudaMalloc .[11]
- Δέσμευση τώρα στην κάρτα γραφικών GPU . Πραγματοποιείται απο την κυρίως διεργασία με κλήση της συνάρτησης API της CUDA και είναι η εντολή CudaMalloc () . Η δέσμευση αυτή αφορά τη μνήμη της GPU που θα δεχθεί τα δεδομένα εισόδου και τη μνήμη στην οποία θα γράψει η GPU τα αποτελέσματα της επεξεργασίας. [11].
- Μετά γίνεται αντιγραφή των προς επεξεργασία δεδομένων εισόδου , απο την μνήμη της κύριας διεργασίας CPU (host) στην μνήμη της GPU (device). Η μεταφορά γίνεται με την συνάρτηση API (Application programming interface) της CUDA ,CudaMemcpy ().[11]
- Καθορισμός της διάταξης των νημάτων και εντολή εκτέλεσης της νηματοσυνάρτησης στους πυρήνες με την εντολή API της CUDA.[11] ,Kernel_function <<< GRID DIM, BLOCK DIM >>> (argument list); [11]
- Επιβολή φράγματος (barrier) συγχρονισμού για να εξασφαλιστεί ότι όλα τα νήματα έχουνε ολοκληρώσει την εκτέλεσή τους , με κλήση της συνάρτησης API της CUDA ,CudaThreadSynchronize (). [11]

..

Ερχόμαστε πάλι στο παράδειγμα με την πρόσθεση των 2 πινάκων A και B και το αποτέλσμα πάει στον C πίνακα,
στην αρχή θα δεσμέυσουμε μνήμη στην CPU με τον κλασικό τρόπο. [11]

```
int N  = 100000;
float *hA, *hB, *hC;
hA = (float*) malloc (N*sizeof(float));
hB = (float*) malloc (N*sizeof(float));
hC = (float*) malloc (N*sizeof(float));
```

Η malloc δεσμεύει ενα μπλοκ N*sizeof (float) συνεχόμενου byte μνήμης και επιστρέφει την αρχική διέυθυνση
του μπλοκ , την οποία αποθηκεύουμε στον αντίστοιχο δείκτη (pointer).[11]
Μετά δεσμέυουμε μνήμη στην GPU χρησιμοποιώντας τη συνάρτηση CudaMalloc () του API της CUDA.[11] Ο
αντίστοιχος κώδικας είναι:

```
int N = 100000;
float *dA,*dB, *dC;
dA = CudaMalloc ((void **)&dA, N*sizeof(float));
dB = CudaMalloc ((void **)&dB, N*sizeof(float));
dC = CudaMalloc ((void **)&dC, N*sizeof(float));
```

Μετα θα βάλουμε τυχαία δεδομένα στους πίνακες:

```
for (i =0; I < N; i++)
{
   hA[i] = (float)rand();
   hB[i] = (float)rand();
}
```

Μετά θα πρέπει να γίνει μεταφορά στην μνήμη της GPU για να είναι προσπελάσιμα απο τους πυρήνες για να
το επεξεργαστούν.[11] Η μεταφορά γίνεται με την συνάρτηση CudaMemcpy () του API της CUDA.[11]Οπότε με την
χρήση της συνάρτησης αυτής, γίνεται μεταφορά των δεδομένων των 2 πινάκων Α και Β απο την μνημη CPU στην
μνήμη GPU.[11]. Κώδικας παρακάτω:

```
CudaMemcpy (dA, hA, N*sizeof(float), cudaMemcpyHostToDevice);
CudaMemcpy (dB, hB, N*sizeof(float), cudaMemcpyHostToDevice);
```

Μεταφορά των αποτελεσμάτων απο μνήμη GPU στην μνήμη CPU

```
CudaMemcpy (hC, dC, N*sizeof(float), cudaMemcpyDeviceToHost);
```

Και αποδέσμευση της μνήμης GPU που δεσμεύθηκε για την επεξεργασία γίνεται απο κλήση της συνάρτησης
CudaFree (); του API της CUDA.[11]. Κώδικας παρακάτω:

```
CudaFree (dA);
CudaFree (dB);
CudaFree (dC);
```

Συνάρτηση πυρήνα Kernel function

```
__global__ void add1(float* A, float* B, float* C, long N){
C[threadidx.x] = A[threadidx.x] + B[threadidx.x];
}
__global__void add2(float* A, float* B, float* C, long N){
C[blockIdx.x] = A[blockIdx.x] + B[blockIdx.x];
```

```
}

__global__void add3(float* A, float* B, float* C, long N){
long gid = blockDim.x * blockIdx.x + threadIdx.x;
C[gid] = A[gid] + B[gid];
}
```

εκτέλεση συνάρτησης πυρήνα

```
Kernel_name <<< GRID DIM, BLOCK DIM >>>  (argument list);
add3 <<<3,5>>> (dA,dB,dC,N);
```

malloc = Δεσμευση μνήμης στην CPU (HOST)

CudaMalloc = Δέσμευση μνήμης στην GPU (DEVICE)

CudaMemcpy = Αντιγραφή /Μεταφορά Δεδομένων απο την CPU στην GPU ή απο την GPU στην CPU

CudaFree = Αποδέσμευση μνήμης GPU που είχε δεσμευθεί

gridDim (τύπου dim3 που περιέχει τις διαστάσεις ενός πλέγματος)
blockIdx (τύπου unint3 που προσδιορίζει κάθε μπλοκ μέσα σε ένα πλέγμα) [15]
blockDim (τύπου dim3 και προσδιορίζει τις διαστάσεις ενός μπλοκ) [15]
threadIdx (τύπου unint3 που προσδιορίζει μονοσήμαντα ένα νήμα μέσα στο πλέγμα) [15]
warpSize (τύπου int που προσδιορίζει το μέγεθος ενός σμήνους-στημόνι) [15]

Ο αριθμός των νημάτων που μπορεί να περιέχει ένα μπλοκ είναι περιορισμένος κυρίως λόγω του ότι όλα τα νήματα στο ίδιο μπλοκ κάνουν χρήση της κοινόχρηστης μνήμης.[15] Η ταυτότητα κάθε νήματος είναι προσδιορίσιμη άμεσα, με τη χρήση του διανύσματος threadIdx που περιέχει τις 3 συντεταγμένες ενός νήματος, ανάλογα αν το νήμα αυτό ανήκει σε ένα μονοδιάστατο / δισδιάστατο / τρισδιάστατο μπλοκ.[15] Για παράδειγμα, για να πάρουμε τις συντεταγμένες ενός νήματος δύο διαστάσεων αναφερόμαστε σε αυτό ως εξής [15]:

int x = threadIdx.x;
int y = threadIdx.y;

Φυσικά, στην περίπτωση που θέλουμε να προσδιορίσουμε μονοσήμαντα ένα νήμα, αλλά έχουμε περισσότερα του ενός μπλοκ και έχοντας υπόψη ότι κάθε μπλοκ μπορεί να διαθέτει μέχρι 3 διαστάσεις, θα χρησιμοποιήσουμε τον παρακάτω τύπο [15]:

int x= threadIdx.x + blockIdx.x * blockDim.x;

Γενικά, ο αριθμός των μπλοκ που μπορούμε να χρησιμοποιήσουμε διαφέρει ανάλογα με το πλήθος των δεδομένων που διαχειριζόμαστε, αλλά και από το πλήθος των διαθέσιμων επεξεργαστών.[15] Για να προσδιορίσουμε τη θέση ενός συγκεκριμένου μπλοκ μέσα σε μια ομάδα από μπλοκ (πλέγμα) δύο διαστάσεων, για παράδειγμα, χρησιμοποιούμε τις εξής μεταβλητές [15] :

int x = blockIdx.x; int y = blockIdx.y;

Αντιστοίχως, για να προσδιορίσουμε τις διαστάσεις ενός μπλοκ, αναφερόμαστε σε αυτές ως εξής [15]:
int x = blockDim.x; int y = blockDim.y;

Ένα μπλοκ νημάτων μπορεί να είναι μονοδιάστατο, δισδιάστατο ή τρισδιάστατο [15]

Ένα πλέγμα μπορεί να έχει δύο διαστάσεις το πολύ και τις διαστάσεις του τις περνάμε ως παραμέτρους στη συνάρτηση πυρήνα με τρόπο που θα εξεταστεί παρακάτω.[15] Κάθε πυρήνας εκτέλεσης δημιουργεί ένα πλέγμα εκτέλεσης, του οποίου οι διαστάσεις του προσδιορίζονται με παρόμοιο τρόπο, όπως με τα μπλοκ και τα νήματα με τις μεταβλητές του τύπου dim3 [15]:
gridDim.x
gridDim.y
Ένα πλέγμα μπορεί να είναι είτε μονοδιάστατο,η είτε δισδιάστατο [15]

Προσδιορισμός νημάτων ανά warp

Γίνεται μέσω της μεταβλητής warpSize η οποία είναι τύπου int και καθορίζει από πόσα νήματα αποτελείται ένα σμήνος-στημόνια [15].
int warpSize;

Για να είναι εφικτή η παράλληλη εκτέλεση των διεργασιών σε μια συσκευή, με υποστήριξη CUDA, θα πρέπει αυτές να οργανώνονται με ένα συγκεκριμένο τρόπο.[15] Βάσει του μοντέλου CUDA εισάγονται ορισμένες νέες έννοιες που αφορούν την ομαδοποίηση των διεργασιών και την κατάλληλη διαχείρισή τους. [15]

Οι πυρήνες (kernel) είναι συναρτήσεις γραμμένες στη γλώσσα C και καθορίζουν τα τμήματα κώδικα, που θα εκτελεστούν στη device και δημιουργούν νήματα τα οποία εκτελούνται παράλληλα [15] όσες φορές καθορίσουμε εμείς μέσα από τις παραμέτρους που τους δίνουμε. [15]

Η δυνατότητα παράλληλης εκτέλεσης των νημάτων δίνεται μέσα από την οργάνωσή τους σε ομάδες δηλαδή τα μπλοκ (block) εννοώντας ότι τα νήματα τα οποία βρίσκονται μέσα στα μπλοκ εκτελούνται παράλληλα. [15] Σε κάθε περίπτωση, το κάθε μπλοκ μπορεί να φιλοξενήσει μέχρι ένα συγκεκριμένο αριθμό νημάτων, ο οποίος καθορίζεται πάντα από τις δυνατότητες της εκάστοτε κάρτας εκτέλεσης. [15] Συγκεκριμένα, πρόκειται για την ιδιότητα maxThreadsPerBlock, που ορίζει το μέγιστο αριθμό νημάτων που μπορεί να περιέχει ένα μπλοκ και η οποία στα τελευταία μοντέλα είναι ίση με 512 νήματα. [15] Τα μπλοκ επίσης, οργανώνονται σε συστοιχίες για να αποτελέσουν ένα ή περισσότερα πλέγματα, ο αριθμός των οποίων εξαρτάται από το πλήθος των δεδομένων μας και τον αριθμών των διαθέσιμων επεξεργαστών στην κάρτα γραφικών. [15]

Για τον προσδιορισμό όλων των εννοιών που χρησιμοποιούνται για τον προγραμματισμό σε CUDA, δηλαδή των νημάτων, των μπλοκ και των πλεγμάτων μέσα σε ένα περιβάλλον εκτέλεσης, χρησιμοποιούνται ορισμένες ενσωματωμένες μεταβλητές.[15] Με τη βοήθεια αυτών των μεταβλητών ο προγραμματιστής μπορεί να καθορίσει επακριβώς τις διαστάσεις ενός μπλοκ ή ενός πλέγματος και να έχει πρόσβαση στους δείκτες που προσδιορίζουν μοναδικά κάθε μπλοκ ή κάθε νήμα μέσα σε ένα πλέγμα. [15]

1 πλέγμα Grid μπορεί να έχει 128 μπλοκ
1 μπλοκ μπορεί να έχει 512 νήματα ή 256 νήματα
Αρα εχουμε 128 μπλοκ χ 512 νήματα = 65536 νήματα
Εάν εχουμε 128 πυρήνες (cores) που μπορεί να έχει μια κάρτα γραφικών SP = 128 μπλοκ

Στην κάρτα γραφικών GeForce 8800 ή οποία έχει 16 πολυεπεξεργαστές συνεχούς ροής (SM) Streaming Multiprocessors. Σε κάθε SM μπορούν να ανατεθούν μέχρι 8 μπλοκ / πυρήνες (SP) Streaming processors αρκεί να υπάρχουν αρκετοί πόροι ώστε να ικανοποιούν τις ανάγκες όλων των μπλοκ. Με 16 SM στον επεξεργαστή της GeForce 8800 μπορούν να ανατεθούν μέχρι 128 μπλοκ ταυτόχρονα σε αυτούς.
Επίσης ενα πλέγμα μπορεί να έχει 128 μπλοκ ή 240 μπλοκ ή 192 μπλοκ ή 384 μπλοκ
Εαν για παράδειγμα είναι δυνατή η ανάθεση 1024 νημάτων τότε θα μπορούσε να γίνει με την μορφή 4 μπλοκ (SP) των 256 νημάτων το καθένα εαν το κάθε μπλοκ έχει 256 νήματα, μπορεί να έχει 512 νήματα το μπλοκ . Επίσης μπορεί να γίνει με 8 μπλοκ των 128 νημάτων = 1024 νήματα.

Η διαδικασία ανάθεσης νημάτων στους **SM (Streaming Multiprocessors)** είναι απλή [15]

1. Όταν ένας πυρήνας καλείται, δημιουργεί ένα πλέγμα (grid). [15]
2. Το πλέγμα οργανώνεται σε μπλοκ νημάτων και τα μπλοκ αυτά στη συνέχεια διανέμονται στους SM, ανάλογα με την διαθεσιμότητα του κάθε SM και τις δυνατότητες να φιλοξενήσουν ένα ή περισσότερα μπλοκ νημάτων.[15]
3. Στη συνέχεια **οι SM οργανώνουν τα μπλοκ νημάτων σε warps (στημόνια)**, ξεκινώντας με το πρώτο warp που αρχίζει με το νήμα 0 και συνεχίζοντας με τα υπόλοιπα νήματα κατά αύξοντα αριθμό. [15]**Τα warp που σχηματίζονται προγραμματίζονται για εκτέλεση από έναν warp-scheduler. [15]**

Στις υλοποιήσεις μιας κάρτας γραφικών για παράδειγμα η GT200 και οποιαδήποτε άλλης μετά από την ανάθεση ενός μπλοκ σε έναν πολυεπεξεργαστή συνεχούς ροής (SM) , αυτό υποδιαιρείται σε μονάδες των 32 νημάτων που λέγονται στημόνια (warps). [12] Το μέγεθος των στημονιών εξαρτάται από την υλοποίηση. Το στημόνι είναι η μονάδα χρονοπρογραμματισμού νημάτων στους SM. [12] Κάθε στημόνι αποτελείται από 32 νήματα με διαδοχικές τιμές threadIdx; Τα νήματα 0 έως 31 σχηματίζοντας το πρώτο στημόνι , τα νήματα 32 έως 63 το δεύτερο , κτλ.[12]

Μπορούμε λοιπόν να υπολογίσουμε το πλήθος των στημονιών που περιέχονται σε έναν SM για δεδομένο μέγεθος μπλοκ και δεδομένο αριθμό μπλοκ που έχουν ανατεθεί σε έναν SM. [12] Εάν για παράδειγμα κάθε μπλοκ διαθέτει 256 νήματα, μπορούμε να προσδιορίσουμε ότι κάθε μπλοκ διαθέτει 256/32 ή 8 στημόνια .[12] Με 3 μπλοκ σε κάθε SM , έχουμε 8 χ 3 = 24 στημόνια σε κάθε SM. Τίθεται το ερώτημα γιατι να έχουμε τόσα πολλά στημόνια σε έναν SM αν υπάρχουν μόνον 8 SP σε έναν SM . [12]

Ο λόγος είναι ότι τρόπος που οι επεξεργαστές της CUDA εκτελούν αποδοτικά λειτουργίες με μεγάλο λανθάνουσα χρόνο (latency) όπως οι προσπελάσεις της καθολικής μνήμης.[12] Όταν μια εντολή που εκτελείται από τα νήματα ενός στημονιού πρέπει να περιμένει για το αποτέλεσμα της λειτουργίας με μεγάλο λανθάνουσα χρόνο που ξεκίνησε προηγουμένως, το στημόνι δεν επιλέγεται για εκτέλεση. [12] Αντι αυτού επιλέγεται κάποιο άλλο στημόνι που δεν περιμένει πλέον για αποτέλεσμα . [12] Αν είναι ετοιμα προς εκτέλεση περισσότερα από έναν στημόνια , για την επιλογή κάποιου από αυτά χρησιμοποιείται ενας μηχανισμός προτεραιοτήτων.[12]

Εύρος ζώνης επικοινωνίας και εύρος ζώνης μνήμης. [15] Το εύρος ζώνης επικοινωνίας μιας κάρτας γραφικών αφορά το διαθέσιμο εύρος για ανταλλαγή δεδομένων μεταξύ της κάρτας και του κεντρικού συστήματος επεξεργασίας.[15] Το εύρος ζώνης μνήμης αποτελεί το διαθέσιμο εύρος για ανταλλαγή δεδομένων μεταξύ της καθολικής μνήμης και των υπόλοιπων στοιχείων μέσα στα όρια της κάρτας γραφικών.[15] Το εύρος ζώνης επικοινωνίας είναι πολύ μικρότερο από το εύρος ζώνης μνήμης.[15] Για παράδειγμα στην κάρτα γραφικών G80 που εισήγαγε την αρχιτεκτονική CUDA το εύρος ζώνης επικοινωνίας με την CPU ήταν 8 GB/s ενώ το εύρος ζώνης μνήμης 86,4 GB/s αλλά αναμένεται να αυξηθεί στο μέλλον. [15]

Ιεραρχία μνήμης.[15] Η αρχιτεκτονική CUDA ορίζει οτι όλα τα νήματα τα οποία καλούνται από τον κώδικα που εκτελείται στο κεντρικό σύστημα, αλλά εκτελούνται σε μια ξεχωριστή κάρτα γραφικών, που διατηρεί τα δικά της ανεξάρτητα κυκλώματα μνήμης.[15] Τα πιο γρήγορα στοιχεία μνήμης αποτελούν οι καταχωρητές και η κοινόχρηστη μνήμη που βρίσκονται τοποθετημένα στα κυκλώματα των SM και διαθέτουν πολύ μικρή χωρητικότητα, της τάξης των μερικών KB. [15] Τη μεγαλύτερη χωρητικότητα (της τάξης των GB) αλλά ταυτόχρονα και τους μεγαλύτερους χρόνους προσπέλασης διαθέτει η καθολική μνήμη, η οποία βρίσκεται πάνω στο κύκλωμα της κάρτας γραφικών, αλλά εκτός ορίων των SM. [15]

Χρονισμός

clock_t clock();

Η κλήση της παραπάνω συνάρτησης επιστρέφει την τιμή του χρονιστή για ένα συγκεκριμένο SM. [15] Η τιμή του κυκλώματος χρονιστή αυξάνεται κατά ένα σε κάθε κύκλο ρολογιού. [15] Μετρώντας την τιμή αυτή στην αρχή της εκτέλεσης κάθε πυρήνα και στο τέλος του, αφαιρώντας την τελική τιμή από την αρχική και κάνοντας το ίδιο για κάθε νήμα μπορούμε να προσδιορίσουμε το συνολικό χρόνο που χρειάστηκε η κάρτα γραφικών για να ολοκληρώσει το συγκεκριμένο νήμα εντολών.[15] Πάντως, ο μετρητής αυτός δεν προσδιορίζει τον ακριβή χρόνο που χρειάστηκε η κάρτα γραφικών αποκλειστικά για την εκτέλεση του νήματος των εντολών. [15] Ο συνολικός χρόνος είναι κατά πολύ μεγαλύτερος σε σχέση με τον χρόνο εκτέλεσης του νήματος, καθώς ο συνολικός χρόνος μετράται ανά νήμα και τα νήματα μπορεί να απέχουν μεταξύ τους χρονικά. [15]

Συναρτήσεις συγχρονισμού :

void __syncthreads(): Περιμένει μέχρις ότου όλα τα νήματα του ίδιου μπλοκ φτάσουν στο σημείο του κώδικα, όπου έχει οριστεί η syncthreads(). [15] Επίσης, θα πρέπει όλες οι προσπελάσεις στην καθολική μνήμη αλλά και την κοινόχρηστη μνήμη που έγιναν από τα νήματα του μπλοκ μέχρι εκείνο το σημείο να είναι ορατές σε όλα τα νήματα του μπλοκ. [15]

ΑΝΑΦΟΡΕΣ – REFERENCES ΜΕΡΟΥΣ Β ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

[11] Προγραμματισμός και Αρχιτεκτονική Συστημάτων Παράλληλης Επεξεργασίας, Στυλιανός Παπαδάκης, Κωνσταντίνος Διαμαντάρας Εκδόσεις Κλειδάριθμος

[12] Προγραμματισμός Μαζικά Παράλληλων Επεξεργαστών ,David B. Kirk, Wen-mei W. Hwu ,Εκδόσεις Κλειδάριθμος

[13] Οργάνωση και Σχεδίαση Υπολογιστών Η Διασύνδεση Υλικού και Λογισμικού, David A.Patterson, John L. Hennessy , Εκδόσεις Κλειδάριθμος

[14] http://repfiles.kallipos.gr/html_books/93/04a-main.html

[15] ΜΕΛΕΤΗ ΤΗΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ CUDA ΚΑΙ ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΚΑΡΤΩΝ GPU ΤΗΣ NVIDIA, Πτυχιακη Εργασία Κελαιδη Αιμιλια, Επιβλέπων Καθηγητής *Κωνσταντίνος Διαμαντάρας*

[21] CUDA C PROGRAMMING GUIDE

ΜΕΡΟΣ Γ ΚΩΔΙΚΑΣ CUDA C ΚΑΙ ΕΠΕΞΗΓΗΣΗ ΑΥΤΟΥ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΤΟΥ ΑΝΤΩΝΙΟΥ ΜΙΧΑΗΛ ΜΑΡΑΓΚΟΥΔΑΚΗ

ΜΕ ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ ΜΕ ΧΡΗΣΗ ΝΗΜΑΤΩΝ ΜΕ ΧΡΗΣΗ ΕΠΕΞΕΡΓΑΣΤΗ GPU ΚΑΡΤΑΣ ΓΡΑΦΙΚΩΝ NVIDIA

ΚΑΙ ΧΡΗΣΗ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ ΜΕ CONVOLUTION METHOD ΚΑΙ BACK PROPAGATION METHOD

ΥΠΟΛΟΓΙΖΕΙ ΜΕΣΟ ΤΕΤΡΑΓΩΝΙΚΟ ΣΦΑΛΜΑ , ΔΕΛΤΑ ΣΦΑΛΜΑ ΚΑΙ ΥΠΟΛΟΓΙΖΕΙ ΝΕΟ ΒΑΡΟΣ LAYER NEYRONS

ΓΡΑΜΜΕΝΟ ΣΕ CUDA C ΜΕ ΧΡΗΣΗ MICROSOFT VISUAL STUDIO 2013 PROFESSIONAL

Ο ΣΥΓΚΕΚΡΙΜΜΕΝΟΣ ΠΑΡΑΚΑΤΩ ΚΩΔΙΚΑΣ ΥΠΟΛΟΓΙΖΕΙ:

ΤΟ ΜΕΣΟ ΤΕΤΡΑΓΩΝΙΚΟ ΣΦΑΛΜΑ ΚΑΙ ΓΙΑ ΤΑ 4 LAYERS NEURONS

ΥΠΟΛΟΓΙΖΕΙ ΔΕΛΤΑ ΣΦΑΛΜΑ ΓΙΑ ΤΟΥΣ LAYERS NEURONS 1 ΚΑΙ 2 ΚΑΙ 3 ΚΑΙ 4

ΥΠΟΛΟΓΙΖΕΙ ΝΕΟ ΒΑΡΟΣ ΓΙΑ ΤΟΥΣ LAYERS NEYRONS 1 ΚΑΙ 2 ΚΑΙ 3 ΚΑΙ 4

ΥΠΟΛΟΓΙΖΕΙ ΔΕΛΤΑ ΣΦΑΛΜΑ ΜΕ ΣΩΣΤΟ ΤΡΟΠΟ ΓΙΑ ΤΟΥΣ LAYERS NEURONS 1 ΚΑΙ 2 ΚΑΙ 3 ΚΑΙ 4

ΥΠΟΛΟΓΙΖΕΙ ΝΕΟ ΒΑΡΟΣ ΜΕ ΣΩΣΤΟ ΤΡΟΠΟ ΓΙΑ ΤΟΥΣ LAYERS NEYRONS 1 ΚΑΙ 2 ΚΑΙ 3 ΚΑΙ 4

ΕΚΠΑΙΔΕΥΕΤΑΙ ΠΑΝΩ ΣΕ XOR 2 ΕΙΣΟΔΩΝ ΚΑΙ 1 ΕΞΟΔΟΥ
ΧΡΗΣΙΜΟΠΟΙΕΙ ΤΟΥΣ LAYERS 1 ΚΑΙ 2 ΓΙΑ ΤΙΣ 2 ΕΙΣΟΔΟΥΣ ΚΑΙ ΤΟΝ ΤΡΙΤΟ LAYER ΓΙΑ ΤΗΝ ΕΞΟΔΟ

ΜΙΑ ΝΕΑ ΠΡΟΣΕΓΓΙΣΗ

ΑΝΤΩΝΙΟΣ ΜΙΧΑΗΛ ΜΑΡΑΓΚΟΥΔΑΚΗΣ

ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ ΦΟΙΤΗΤΗ: mtt13

ΠΑΡΑΤΗΡΗΣΕΙΣ: Ο ΚΩΔΙΚΑΣ ΒΡΙΣΚΕΤΑΙ ΣΤΟΝ ΦΑΚΕΛΛΟ: ERGASIA ΜΕΤΑΡΤΥΧΙΑΚΟU10B DEVELOPMENT ΒΕΛΤΙΩΜΕΝΗ ΕΚΔΟΣΗ

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <assert.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <malloc.h>
#include <iostream>
#include <direct.h>
#include "device_launch_parameters.h"

float *Layer_neurons_CPU;

int neuron;



float *Layer1_Neurons_GPU;
//float Layer1_Neurons_GPU[29*29];
float Layer1_Weights_CPU[156];
float *Layer1_Weights_GPU;
float *mean_square_error_Layer1_Neurons_CPU[841];

float *mean_square_error_Layer2_Neurons_CPU[1014];

float *mean_square_error_Layer3_Neurons_CPU[1250];

float *mean_square_error_Layer4_Neurons_CPU[100];

float *mean_square_error_Layer5_Neurons_CPU[10];

float *mean_square_error_Layer6_Neurons_CPU[10];

float pinakas_Layer1_Neurons_CPU[841];

float pinakas_Layer2_Neurons_CPU[1014];

float *pinakas_Layer3_Neurons_CPU[1250];

float *XOR_X_Layer1_Neurons_CPU[841];

float *XOR_Y_Layer2_Neurons_CPU[1014];

float *XOR_Z_Layer3_Neurons_CPU[1250];



float Layer2_Weights_CPU[7800];
float *Layer2_Weights_GPU;
float *Layer2_Neurons_GPU;

float Layer3_Weights_CPU[125100];
float *Layer3_Weights_GPU;
float *Layer3_Neurons_GPU;

float New_Layer4_Weights_CPU[1010];
float New_Layer3_Weights_CPU[125100];
float New_Layer2_Weights_CPU[7800];
float New_Layer1_Weights_CPU[841];
```

```
float Layer4_Weights_CPU[1010];
float *Layer4_Weights_GPU;

float *New_Layer4_weights_GPU;
float *New_Layer3_weights_GPU;
float *New_Layer2_weights_GPU;
float *New_Layer1_weights_GPU;


float Layer5_Neurons_CPU[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
float *Layer5_Neurons_GPU;

float result;

float delta_Layer5_Neurons_CPU[10];
float delta_Layer3_Neurons_CPU[1250];
float delta_Layer4_Neurons_CPU[100];
float delta_Layer2_Neurons_CPU[1014];
float delta_Layer1_Neurons_CPU[841];

float delta_Layer5_Neurons_GPU;
float delta_Layer4_Neurons_GPU;
float delta_Layer3_Neurons_GPU;
float delta_Layer2_Neurons_GPU;
//float delta_Layer1_Neurons_CPU;
float delta_Layer1_Neurons_GPU;


float mean_square_error_Layer1_Neurons_GPU;
//float *mean_square_error_Layer1_Neurons_GPU;

float mean_square_error_Layer2_Neurons_GPU;
//float *mean_square_error_Layer2_Neurons_GPU;
//float mean_square_error_Layer3_Neurons_GPU;
float mean_square_error_Layer4_Neurons_GPU;
float mean_square_error_Layer5_Neurons_GPU;
float mean_square_error_Layer6_Neurons_GPU;

float pinakas_Layer1_Neurons_GPU;

float pinakas_Layer2_Neurons_GPU;

float pinakas_Layer3_Neurons_GPU;

float XOR_X_Layer1_Neurons_GPU;

float XOR_Y_Layer2_Neurons_GPU;

float XOR_Z_Layer3_Neurons_GPU;


//float mean_square_error_Layer1_Neurons_GPU[mse1] = 1 / 2 * (target - result);

double *outputLayer;
void readIn(float *Layer1_Neurons_CPU);
void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU);



__global__ void executeFirstLayer(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU,
float *Layer2_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
```

```
*delta_Layer1_Neurons_GPU, float *delta_Layer2_Neurons_GPU, float
*New_Layer1_Weights_GPU, float *delta_Layer3_Neurons_GPU, float *Layer2_Weights_GPU,
float *pinakas_Layer1_Neurons_GPU, float *XOR_X_Layer1_Neurons_GPU)
{
	int blockID = blockIdx.x;
	int pixelX = threadIdx.x;
	int pixelY = threadIdx.y;
	//	float mean_square_error_Layer1_Neurons_GPU;
	float target;

	int XOR1;
	int PIN1;
	float DLTA2B;
	float DLTA1B = 0;
	float DLTA2;
//	float DLTA2B = 0;
	float DLTA3B;
	float NL1W = 0;
	float NL1WB = 0;
	int a = 0.5; //learning rate//
	int dlta2 = blockID * 26;
	int dlta2b = blockID * 26;
//	int dlta3b = blockID * 26 * 6;
	int dlta1b = blockID * 26;

	int kernelTemplate[25] = {
		0, 1, 2, 3, 4,
		29, 30, 31, 32, 33,
		58, 59, 60, 61, 62,
		87, 88, 89, 90, 91,
		116, 117, 118, 119, 120 };


	int weightBegin = blockID * 26;
	int windowX = pixelX * 2;
	int windowY = pixelY * 2;
	int mse1 = blockID * 26;
	int pinak1 = blockID * 26;
	int xor1 = blockID * 26;
	//int pinak1;
	//int xor1;


	float result = 0;
	float MSE1 = 0;
	float resulta;
	result += Layer1_Weights_GPU[weightBegin];
	//++mse1;
	++weightBegin;
	dlta2++;
	dlta2b++;
	++pinak1;
	++xor1;

	PIN1 = pinakas_Layer1_Neurons_GPU[pinak1];
	XOR1 += XOR_X_Layer1_Neurons_GPU[xor1];

	DLTA2 = delta_Layer2_Neurons_GPU[dlta2];
	NL1W = New_Layer1_Weights_GPU[weightBegin];
	DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
	//DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
	DLTA1B = delta_Layer1_Neurons_GPU[dlta1b];
```

```
        NL1WB = New_Layer1_Weights_GPU[weightBegin];

        //PIN1[1*4] = { 0, 0, 1, 1 };
        // pinakas_Layer1_Neurons_GPU[4*1] = { 0, 0, 1, 1 };

        //PIN1 =   { (0, 0, 0,
              //                        0, 1, 1,
        //                                1, 0, 1,
        //                                1, 1, 0) };

        //PIN1 = {
        //     { ( 0, 0 ), (0) },
        //     { ( 0, 1 ), (1) },
        //     { ( 1, 0 ), (1) },
        //     { ( 1, 1 ), (0) }
        //};
    PIN = {(00, 01, 10, 11)};
        for (int i = 0; i<25; ++i)
        {
              result += Layer1_Neurons_GPU[windowY * 29 + windowX + kernelTemplate[i]] *
Layer1_Weights_GPU[weightBegin + i];
              printf("apotelesma: %d\n", result);
        //     XOR1 = PIN1 * result;
              printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
              printf("pinakas_Layer1_Neurons_GPU: %d\n", pinakas_Layer1_Neurons_GPU);
//            system("PAUSE");


        }
        printf("Layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);
        result = (1.7159*tanhf(0.66666667*result));
        int i;
        DLTA1B = -((result * (1 - result) * DLTA2B * Layer1_Weights_GPU[weightBegin +
i]));
        printf("dlta1b delta_Layer1_Neurons_GPU: %d\n", DLTA1B);

        XOR1 = PIN1 * result;
        printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//      system("PAUSE");


        resulta = result;
        printf("Layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);
        printf("apotelesma: %d\n", resulta);
        //Layer2_Neurons_GPU[13 * 13 * blockID + pixelY * 13 + pixelX] = result;

        printf("layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);

        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

        printf("layer1_weights_GPU: %f\n", Layer1_Weights_GPU);




        target = 1;
        MSE1 = mean_square_error_Layer1_Neurons_GPU[mse1];

        MSE1 = (1 / 2 * (target - result));
        //mean_square_error_Layer1_Neurons_GPU = (1 / 2 * (target - result));
```

```
	//DLTA2 = -((target - result) * result * (1 - result));
	NL1W = (Layer1_Weights_GPU[weightBegin] - a * DLTA2); //new weight layer2
//	int i;
	//DLTA2B = -((result * (1 - result) * DLTA3B * Layer2_Weights_GPU[weightBegin +
i]));
	NL1WB = (Layer1_Weights_GPU[weightBegin] - a * DLTA2B); //new weight layer2

	printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
	printf("mean_square_error_Layer1_Neurons_GPU: %f\n", MSE1);
	printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
	printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
	printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
	printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
	printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
	printf(" New_Layer1_Weights_GPU: %d\n", NL1WB);
	printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
	printf("nl1wb New_Layer1_Weights_GPU: %d\n", NL1WB);


	Layer2_Neurons_GPU[13 * 13 * blockID + pixelY * 13 + pixelX] = result;


//	XOR1 = PIN1 * result;
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//	system("PAUSE");

}

__global__ void executeSecondLayer(float *Layer2_Neurons_GPU, float *Layer2_Weights_GPU,
float *Layer3_Neurons_GPU, float *mean_square_error_Layer2_Neurons_GPU, float
*delta_Layer2_Neurons_GPU, float *delta_Layer3_Neurons_GPU, float
*New_Layer2_Weights_GPU, float *delta_Layer4_Neurons_GPU, float *Layer3_Weights_GPU,
float *pinakas_Layer2_Neurons_GPU, float *XOR_Y_Layer2_Neurons_GPU)
{
	int blockID = blockIdx.x;
	int pixelX = threadIdx.x;
	int pixelY = threadIdx.y;
	//	float mean_square_error_Layer2_Neurons_GPU;
	float target;

//	float DLTA4B;
	float DLTA3;
	float NL2W = 0;
	float DLTA3B;
	float DLTA2B = 0;
	//float DLTA1B = 0;
	float NL2WB = 0;
	int a = 0.5; //learning rate//
	int dlta3 = blockID * 26 * 6;
	int dlta2b = blockID * 26 * 6;
	//int dlta4b = blockID * 1251;
	int dlta3b = blockID * 26 * 6;
	//int dlta1b = blockID * 26;
	int XOR2;
	int PIN2;

	int kernelTemplate2[25] = {
		0, 1, 2, 3, 4,
		13, 14, 15, 16, 17,
		26, 27, 28, 29, 30,
		39, 40, 41, 42, 43,
```

```
		52, 53, 54, 55, 56 };

	int weightBegin = blockID * 26 * 6;
	int windowX = pixelX * 2;
	int windowY = pixelY * 2;
	int mse2 = blockID * 26;
	int pinak2 = blockID * 26 * 6;
	int xor2 = blockID * 26 * 6;
	//int pinak2;
	//int xor2;


	float result = 0;
	float MSE2 = 0;
	//float MSE3 = 0;
	//result += Layer2_Weights_GPU[weightBegin];


	result += Layer2_Weights_GPU[weightBegin];
	//PIN2[4*1] = { 0, 1, 0, 1 };
//	float pinakas_Layer2_Neurons_GPU[4 * 1] = { 0, 1, 0, 1 };

  // PIN2 =   {( 0, 0, 0,
	//		0, 1, 1,
	//		1, 0, 1,
	//		1, 1, 0) };
	//PIN2 = {
	//	{ ( 0, 0 ), ( 0 ) },
	//	{ ( 0, 1 ), ( 1 ) },
	//	{ ( 1, 0 ), ( 1 ) },
	//	{ ( 1, 1 ), ( 0 ) }
	//};

	PIN2 = { (00, 01, 10, 11)};


	if (blockID == 1 && pixelX == 0 && pixelY == 0)
	{
		result += 0;
	}

	++weightBegin;
	//++mse2;
	dlta3++;
	dlta3b++;
	++pinak2;
	++xor2;

	PIN2 = pinakas_Layer2_Neurons_GPU[pinak2];
	XOR2 += XOR_Y_Layer2_Neurons_GPU[xor2];

	DLTA3 = delta_Layer3_Neurons_GPU[dlta3];
	NL2W = New_Layer2_Weights_GPU[weightBegin];
//	DLTA4B = delta_Layer4_Neurons_GPU[dlta4b];
	DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
	DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
//	DLTA1B = delta_Layer1_Neurons_GPU[dlta1b];
	NL2WB = New_Layer2_Weights_GPU[weightBegin];

	for (int i = 0; i<25; ++i)
	{
```

```
            result += Layer2_Neurons_GPU[windowX + 13 * windowY + kernelTemplate2[i]] *
Layer2_Weights_GPU[weightBegin + i * 6];
            result += Layer2_Neurons_GPU[169 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 1];
            result += Layer2_Neurons_GPU[338 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 2];
            result += Layer2_Neurons_GPU[507 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 3];
            result += Layer2_Neurons_GPU[676 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 4];
            result += Layer2_Neurons_GPU[845 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 5];
            //XOR2 = PIN2 * result;
            printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
//            system("PAUSE");

      }

      result = (1.7159*tanhf(0.66666667*result));
      int i;
      DLTA2B = -((result * (1 - result) * DLTA3B * Layer2_Weights_GPU[weightBegin +
i]));
      printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);

      XOR2 = PIN2 * result;
      printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
      printf("pinakas_Layer2_Neurons_GPU: %d\n", pinakas_Layer2_Neurons_GPU);
//      system("PAUSE");



      //Layer3_Neurons_GPU[5 * 5 * blockID + pixelY * 5 + pixelX] = result;

      printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

      printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

      printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);







      target = 1;

      MSE2 = mean_square_error_Layer2_Neurons_GPU[mse2];

      MSE2 = (1 / 2 * (target - result));
      //DLTA3 = -((target - result) * result * (1 - result));
      NL2W = (Layer2_Weights_GPU[weightBegin] - a * DLTA3); //new weight layer2
//      int i;
      //DLTA3B = -((result * (1 - result) * DLTA4B * Layer3_Weights_GPU[weightBegin +
i]));
      NL2WB = (Layer2_Weights_GPU[weightBegin] - a * DLTA3B); //new weight layer3


      //mean_square_error_Layer2_Neurons_GPU[mse2] = 1 / 2 * (target - result);
      //      mean_square_error_Layer2_Neurons_GPU = 1 / 2 * (target - result);
      printf("mean_square_error_Layer2_Neurons_GPU: %d\n",
mean_square_error_Layer2_Neurons_GPU);
```

```
	printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);
	printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
	printf(" New_Layer2_Weights_GPU: %d\n", NL2W);
	printf("delta_Layer3_Neurons_GPU: %d\n", delta_Layer3_Neurons_GPU[dlta3]);

	printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
	printf(" New_Layer2_Weights_GPU: %d\n", New_Layer2_Weights_GPU[weightBegin]);
	printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
	printf(" New_Layer2_Weights_GPU: %d\n", NL2WB);
	printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
	printf("nl2wb New_Layer2_Weights_GPU: %d\n", NL2WB);

	Layer3_Neurons_GPU[5 * 5 * blockID + pixelY * 5 + pixelX] = result;


	//XOR2 = PIN2 * result;
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
//	system("PAUSE");



}

__global__ void executeThirdLayer(float *Layer3_Neurons_GPU, float *Layer3_Weights_GPU,
float *Layer4_Neurons_GPU, float *mean_square_error_Layer3_Neurons_GPU, float
*delta_Layer3_Neurons_GPU, float *delta_Layer4_Neurons_GPU, float
*New_Layer3_Weights_GPU, float *Layer4_Weights_GPU, float *delta_Layer5_Neurons_GPU,
float *pinakas_Layer3_Neurons_GPU, float *XOR_Z_Layer3_Neurons_GPU, float
*XOR_Y_Layer2_Neurons_GPU, float *XOR_X_Layer1_Neurons_GPU)
{
	int blockID = blockIdx.x;
	int pixelY = threadIdx.y;
	//float mean_square_error_Layer3_Neurons_GPU;
	float target;

	int XOR2;
	int PIN3;
	int XOR3;
	int XOR1;

	float DLTA4;
	float DLTA5;
	float NL3W = 0;
	float DLTA4B;
	float DLTA3B = 0;
	float NL3WB = 0;
	int a = 0.5; //learning rate//
	int dlta4 = blockID * 1251;
	int dlta4b = blockID * 1251;
	int dlta3b = blockID * 1251;
	int dlta5 = blockID * 10;
	//int pinak3 = blockID * 1251;
	int xor3 = blockID * 1251;
	int pinak3;
//	int xor3;


	//int xor2 = blockID * 26 * 6;
	int xor2;
	//int xor1 = blockID * 26;
	int xor1;
```

```
	int weightBegin = blockID * 1251;
	int mse3 = blockID * 1251;
	float result = 0;
	float MSE3 = 0;
	result += Layer3_Weights_GPU[weightBegin];
	++xor3;
	//++xor2;
	//++xor1;
	//++pinak3;
	++weightBegin;
	++mse3;
	dlta4++;
	dlta4b++;

	PIN3 = pinakas_Layer3_Neurons_GPU[pinak3];
	XOR3 += XOR_Z_Layer3_Neurons_GPU[xor3];
	XOR2 = XOR_Y_Layer2_Neurons_GPU[xor2];
	XOR1 = XOR_X_Layer1_Neurons_GPU[xor1];

	DLTA4 = delta_Layer4_Neurons_GPU[dlta4];
	NL3W = New_Layer3_Weights_GPU[weightBegin];
	DLTA4B = delta_Layer4_Neurons_GPU[dlta4b];
	DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
	NL3WB = New_Layer3_Weights_GPU[weightBegin];
	DLTA5 = delta_Layer5_Neurons_GPU[dlta5];

	for (int i = 0; i<1250; ++i)
	{
		result += Layer3_Neurons_GPU[i] * Layer3_Weights_GPU[weightBegin + i];
		//XOR3 = XOR1 + XOR2;
		//printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
//		system("PAUSE");

	}

	result = (1.7159*tanhf(0.66666667*result));
	XOR3 = XOR1 + XOR2;
	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
//	system("PAUSE");
	int i;
	DLTA3B = -((result * (1 - result) * DLTA4B * Layer3_Weights_GPU[weightBegin +
i]));

	printf("dlta4b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);

	target = 1;
	MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3];
	MSE3 = (1 / 2 * (target - result));
	//Layer4_Neurons_GPU[blockID] = result;

	printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

	printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

	printf("layer3_weights_GPU: %f\n", Layer3_Weights_GPU);



	//target = 1;
	////MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3];
	//MSE3 = (1 / 2 * (target - result));
	//DLTA4 = -((target - result) * result * (1 - result));
```

```
	NL3W = (Layer3_Weights_GPU[weightBegin] - a * DLTA4); //new weight layer3
//	int i;
	//DLTA4B = -((result * (1 - result) * DLTA5 * Layer4_Weights_GPU[weightBegin +
i]));
	NL3WB = (Layer3_Weights_GPU[weightBegin] - a * DLTA4B); //new weight layer3



	printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %d\n", MSE3);
	printf("delta_Layer4_Neurons_GPU: %d\n", DLTA4);
	printf(" New_Layer3_Weights_GPU: %d\n", NL3W);
	printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", DLTA4B);
	printf("nl3wb New_Layer3_Weights_GPU: %d\n", NL3WB);


	Layer4_Neurons_GPU[blockID] = result;
	//XOR3 = XOR1 + XOR2;
	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//	system("PAUSE");

	//printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
	//printf(" New_Layer2_Weights_GPU: %d\n", NL2W);


}

__global__ void executeFourthLayer(float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU,
float *Layer5_Neurons_GPU, float *mean_square_error_Layer4_Neurons_GPU, float
*mean_square_error_Layer6_Neurons_GPU, float *mean_square_error_Layer5_Neurons_GPU, float
*mean_square_error_Layer3_Neurons_GPU, float *mean_square_error_Layer2_Neurons_GPU, float
*mean_square_error_Layer1_Neurons_GPU, float *delta_Layer5_Neurons_GPU, float
*New_Layer4_Weights_GPU, float *delta_Layer3_Neurons_GPU,float *delta_Layer4_Neurons_GPU
,float *New_Layer2_Weights_GPU, float *delta_Layer2_Neurons_GPU, float
*New_Layer1_Weights_GPU, float *Layer3_Neurons_GPU, float *Layer3_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer1_Neurons_GPU, float
*Layer1_Weights_GPU, float *XOR_X_Layer1_Neurons_GPU, float *XOR_Y_Layer2_Neurons_GPU,
float *XOR_Z_Layer3_Neurons_GPU)
{
	int blockID = blockIdx.x;
	int pixelY = threadIdx.y;


	float target;

	int XOR2;
	//int PIN3;
	int XOR3;
	int XOR1;


	//int xor3 = blockID * 1251;
	int xor3;
	//int xor2 = blockID * 26 * 6;
	int xor2;
//	int xor1 = blockID * 26;
	int xor1;
```

```
int weightBegin = blockID * 101;
int mse4 = blockID * 101;
int mse5 = blockID * 101;
int mse6 = blockID * 101;
int dlta5 = blockID * 101;
int dlta4b = blockID * 101;

float MSE4=0;
float result = 0;
float MSE5=0;
float MSE6 = 0;
result += Layer4_Weights_GPU[weightBegin];



float MSE1;
float MSE2;
float MSE3;
float DLTA5 = 0;
float NL4W = 0;
float DLTA3;
float DLTA3B;
float DLTA4B = 0;
float NL2W;
float NL2WB;

float DLTA2;
float DLTA2B;
float NL1W;
float NL1WB;

int a = 0.5; //learning rate//
//float MSE4 = 0;
//float MSE5 = 0;
//int dlta3 = blockID * 26 * 6;
int dlta3b = blockID * 26 * 6;

int dlta2b = blockID * 26;
//int dlta2 = blockID * 26;
//int mse1 = blockID * 26;
int mse1;
//int mse2 = blockID * 26 * 6;
int mse2;

//int mse3 = blockID * 1251;
int mse3;
//int mse4 = blockID * 101;
//int mse5 = blockID * 10;
XOR3 = XOR_Z_Layer3_Neurons_GPU[xor3];
XOR2 = XOR_Y_Layer2_Neurons_GPU[xor2];
XOR1 = XOR_X_Layer1_Neurons_GPU[xor1];

DLTA4B = delta_Layer4_Neurons_GPU[dlta4b];


DLTA5 = delta_Layer5_Neurons_GPU[dlta5];
NL4W = New_Layer4_Weights_GPU[weightBegin];
//DLTA3 = delta_Layer3_Neurons_GPU[dlta3];
DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
```

```
        NL2W = New_Layer2_Weights_GPU[weightBegin];
        NL2WB = New_Layer2_Weights_GPU[weightBegin];

        //DLTA2 = delta_Layer2_Neurons_GPU[dlta2];
        DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
        //NL1W = New_Layer1_Weights_GPU[weightBegin];
        NL1WB = New_Layer1_Weights_GPU[weightBegin];

        ++weightBegin;
        ++mse4;
        //++xor3;
        ////++xor2;
        //++xor1;
//      ++mse3;
//      ++mse2;
//      ++mse1;
        dlta5++;
        ++mse5;
        ++mse6;
        for (int i = 0; i<100; ++i)
        {
                result += Layer4_Neurons_GPU[i] * Layer4_Weights_GPU[weightBegin + i];
                //printf("apotelesma result layer4_Neurons_GPU: %f\n", result);
                XOR3 = XOR1 + XOR2;
                printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
//              system("PAUSE");
        }
        //printf("apotelesma result layer4_Neurons_GPU: %f\n", result);

        result = (1.7159*tanhf(0.66666667*result));
        XOR3 = XOR1 + XOR2;
        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        //system("PAUSE");

        //printf("apotelesma uperboliko efaptomeni  layer4_Neurons_GPU: %f\n", result);
        //Layer5_Neurons_GPU[blockID] = result;

        XOR3 = XOR1 + XOR2;
        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
        printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//      system("PAUSE");

        printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

        printf("layer5_Neurons_GPU: %f\n", Layer5_Neurons_GPU);

        printf("layer4_weights_GPU: %f\n", Layer4_Weights_GPU);

        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

        printf("layer3_weights_GPU: %f\n", Layer3_Weights_GPU);

        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

        printf("layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);
```

```
        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

        printf("layer1_weights_GPU: %f\n", Layer1_Weights_GPU);

        printf("layer5_Neurons_GPU: %f\n", Layer5_Neurons_GPU);


        target = 1;

        MSE4 = mean_square_error_Layer4_Neurons_GPU[mse4];
        MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3];
        MSE2 = mean_square_error_Layer2_Neurons_GPU[mse2];
        MSE1 = mean_square_error_Layer1_Neurons_GPU[mse1];
        MSE5 = mean_square_error_Layer5_Neurons_GPU[mse5];
        MSE6 = mean_square_error_Layer6_Neurons_GPU[mse6];

        MSE4 = (1 / 2 * (target - result));
        int i;
        DLTA4B = -((result * (1 - result) * DLTA5 * Layer4_Weights_GPU[weightBegin + i]));
//      DLTA5 = -((target - result) * result * (1 - result));
        NL4W = (Layer4_Weights_GPU[weightBegin] - a * DLTA5); //new weight layer4

        printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", DLTA4B);
        printf("nl4w New_Layer4_Weights_GPU: %d\n", NL4W);

        printf("mean_square_error_Layer4_Neurons_GPU: %f\n",
mean_square_error_Layer4_Neurons_GPU);
        printf("mean_square_error_Layer4_Neurons_GPU: %d\n", MSE4);
        printf("mean_square_error_Layer1_Neurons_GPU: %d\n", MSE1);
        printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);

        Layer5_Neurons_GPU[blockID] = result;

        target = 1;
        MSE5 = (1 / 2 * (target - result));
        DLTA5 = -((target - result) * result * (1 - result));

        MSE6 = MSE5 + MSE4 + MSE3 + MSE2 + MSE1;
        mean_square_error_Layer6_Neurons_GPU[mse6]=
mean_square_error_Layer5_Neurons_GPU[mse5] + mean_square_error_Layer4_Neurons_GPU[mse4] +
mean_square_error_Layer3_Neurons_GPU[mse3] + mean_square_error_Layer2_Neurons_GPU[mse2] +
mean_square_error_Layer1_Neurons_GPU[mse1];
             printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU);
        printf("mean_square_error_Layer5_Neurons_GPU: %f\n",
mean_square_error_Layer5_Neurons_GPU[mse5]);
        //MSE5 = mean_square_error_Layer5_Neurons_GPU[mse5];
//      MSE5 += MSE4 + MSE3 + MSE2 + MSE1;
        printf("mean_square_error_Layer2_Neurons_GPU: %f\n",
mean_square_error_Layer2_Neurons_GPU);
        printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
        printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
        printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU[mse6]);

        printf("mean_square_error_Layer5_Neurons_GPU: %d\n", MSE5);
        //printf("mean_square_error_Layer4_Neurons_GPU: %d\n", MSE4);
```

```
	//printf("mean_square_error_Layer1_Neurons_GPU: %d\n", MSE1);
	//printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);
	printf("dlta5 delta_Layer5_Neurons_GPU: %d\n", DLTA5);
	//printf("nl4w New_Layer4_Weights_GPU: %d\n", NL4W);

	printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
	printf(" New_Layer2_Weights_GPU: %d\n", NL2W);

	printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
	printf("nl2wb New_Layer2_Weights_GPU: %d\n", NL2WB);


	printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
	printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
	printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
	printf("nl1wb New_Layer1_Weights_GPU: %d\n", NL1WB);

	XOR3 = XOR1 + XOR2;
	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//	system("PAUSE");



}

////////////////////////////////////////////////////////////////////////////
//

//
void NeuralNetwork();
//void readIn(float *Layer1_Neurons_CPU);
//void readIn(float *Layer1);
////////////////////////////////////////////////////////////////////////////
//
////////////////////////////////////////////////////////////////////////////
int main(int argc, char** argv)
{
	//readIn(float *Layer1);
void readIn(float *Layer1_Neurons_CPU);
	void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU);
	//InitHostMem();
	NeuralNetwork();
void output(double *Layer5_Neurons_CPU);
	printf("test:");
}


/*void InitGPUMem(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer3_Neurons_GPU, float
*Layer3_Weights_GPU, float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU, float
*Layer5_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*mean_square_error_Layer2_Neurons_GPU, float *mean_square_error_Layer3_Neurons_GPU, float
*mean_square_error_Layer4_Neurons_GPU, float *mean_square_error_Layer5_Neurons_GPU, float
*delta_Layer5_Neurons_GPU, float *New_Layer4_Weights_GPU, float
*delta_Layer4_Neurons_GPU, float *New_Layer3_Weights_GPU, float
*delta_Layer3_Neurons_GPU, float *New_Layer2_Weights_GPU, float
*delta_Layer2_Neurons_GPU, float *New_Layer1_Weights_GPU)
{
cudaMalloc((void**)&Layer1_Neurons_GPU, sizeof(float) * 29 * 29);
```

```
cudaMalloc((void**)&Layer1_Weights_GPU, sizeof(float) * 156);
cudaMalloc((void**)&mean_square_error_Layer1_Neurons_GPU, sizeof(float) * 29 * 29);

cudaMalloc((void**)&Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
cudaMalloc((void**)&Layer2_Weights_GPU, sizeof(float) * 7800);
cudaMalloc((void**)&mean_square_error_Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);

cudaMalloc((void**)&Layer3_Neurons_GPU, sizeof(float) * 1250);
cudaMalloc((void**)&Layer3_Weights_GPU, sizeof(float) * 125100);

cudaMalloc((void**)&mean_square_error_Layer3_Neurons_GPU, sizeof(float) * 1250);

cudaMalloc((void**)&Layer4_Neurons_GPU, sizeof(float) * 100);
cudaMalloc((void**)&Layer4_Weights_GPU, sizeof(float) * 1010);

cudaMalloc((void**)&New_Layer4_Weights_GPU, sizeof(float) * 1010);
cudaMalloc((void**)&New_Layer3_Weights_GPU, sizeof(float) * 125100);
cudaMalloc((void**)&New_Layer2_Weights_GPU, sizeof(float) * 7800);
cudaMalloc((void**)&New_Layer1_Weights_GPU, sizeof(float) * 156);

cudaMalloc((void**)&mean_square_error_Layer4_Neurons_GPU, sizeof(float) * 100);

cudaMalloc((void**)&Layer5_Neurons_GPU, sizeof(float) * 10);

cudaMalloc((void**)&delta_Layer5_Neurons_GPU, sizeof(float) * 10);
cudaMalloc((void**)&delta_Layer4_Neurons_GPU, sizeof(float) * 100);
cudaMalloc((void**)&delta_Layer3_Neurons_GPU, sizeof(float) * 1250);
cudaMalloc((void**)&delta_Layer2_Neurons_GPU, sizeof(float) * 1014);


cudaMalloc((void**)&mean_square_error_Layer5_Neurons_GPU, sizeof(float) * 10);

outputLayer = (double*)malloc(sizeof(double) * 10);
}*/
void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU)
{
	// initial layer 1 weight
	FILE *fp1;
	fp1 = fopen("lw1.bin", "rb");
	if (fp1 != NULL)
	{
		for (int i = 0; i < 156; ++i)
			fread(&(Layer1_Weights_CPU[i]), sizeof(float), 156, fp1);
		//		fread(Layer1_Weights_CPU, sizeof(float), 1, pFile1);
		fclose(fp1);
	}
	//	for (int i = 0; i < 156; ++i)

	//	printf("layer1_WEIGHTS_CPU: %f\n", (Layer1_Weights_CPU[i]));
	//	system("PAUSE");


	// initial layer 2 weight
	FILE *fp2;
	fp2 = fopen("lw2.bin", "rb");
	if (fp2 != NULL)
	{
		//for (int i = 0; i<7800; ++i)
		fread(Layer2_Weights_CPU, sizeof(float), 7800, fp2);
		fclose(fp2);
```

```
		for (int i = 0; i < 7800; ++i)
			printf("layer2_WEIGHTS_CPU  arxh hhhhhhhhhhhhhhhhhhh: %f\n",
(Layer2_Weights_CPU[i]));
		system("PAUSE");
	}

	// initial layer 3 weight
	FILE *fp3;
	fp3 = fopen("lw3.bin", "rb");
	if (fp3 != NULL)
	{
		for (int i = 0; i < 125100; ++i)
			fread(&(Layer3_Weights_CPU[i]), sizeof(float), 125100, fp3);
		fclose(fp3);
	}
	// initial layer 4 weight
	FILE *fp4;
	fp4 = fopen("lw4.bin", "rb");
	if (fp4 != NULL)
	{
		//	for (int i = 0; i < 1010; ++i)
		fread(Layer4_Weights_CPU, sizeof(float), 1010, fp4);
		fclose(fp4);
	}
}

void readIn(float *layer1_Neurons_CPU)
{
	FILE *fp6 = 0;
	fp6 = fopen("neuron.bin", "rb");
	if (fp6 != NULL)
	{

//for (int i = 0; i < 841; ++i)

//printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

		fread(&Layer1_Neurons_CPU, sizeof(int), 29 * 29, fp6);

		fread(&neuron, sizeof(int), 29 * 29, fp6);


	float* Layer_neurons_CPU = (float*)  malloc(neuron * sizeof(float) * 29 * 29);

		//fclose(fp6);

		for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer_neurons_CPU[i]));

		for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

		free(malloc);

		fclose(fp6);

		for (int i = 0; i < 841; ++i)
```

```
printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

	}

for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

}


void output(double *Layer5_Neurons_CPU)
{
	FILE *fp5 = 0;
	fp5 = fopen("neuronout.bin", "wb");
	if (fp5)
	{
		fwrite(Layer5_Neurons_CPU, sizeof(double), 10, fp5);
		fclose(fp5);
for (int i = 0; i < 10; ++i)

printf("layer5_Neurons_CPU apotelesma: %f\n", (Layer5_Neurons_CPU[i]));


	}
}
void NeuralNetwork()
{
	//CUT_DEVICE_INIT();
	float Layer1_Neurons_CPU[29 * 29] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
	float Layer4_Weights_CPU[1010] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
	//exei perpou
	for (int i = 0; i<1010; ++i)
		printf("layer4_weights_CPU: %f\n", (Layer4_Weights_CPU[i]));
	system("PAUSE");


	float Layer3_Weights_CPU[125100] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
	// exei peripou 125.000 weights
	for (int i = 0; i<125; ++i)
		printf("layer3_weights_CPU: %f\n", (Layer3_Weights_CPU[i]));
	system("PAUSE");


	float Layer2_Weights_CPU[7800] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1 };
	//exei peripou 8.000 weights
	for (int i = 0; i < 780; ++i)
		printf("layer2_weights_CPU: %f\n", (Layer2_Weights_CPU[i]));
	system("PAUSE");

	float pinakas_Layer1_Neurons_CPU[841] = { 0, 0, 1, 1 };
	for (int i = 0; i<10; ++i)
		printf("pinakas_layer1_Neurons_CPU: %f\n",
(pinakas_Layer1_Neurons_CPU[i]));
	system("PAUSE");



	float pinakas_Layer2_Neurons_CPU[1014] = { 0, 1, 0, 1 };
	for (int i = 0; i<10; ++i)
		printf("pinakas_layer2_Neurons_CPU: %f\n",
(pinakas_Layer2_Neurons_CPU[i]));
	system("PAUSE");



	//float Layer1_Neurons_CPU[29 * 29];

	//void readIn(float *Layer1_Neurons_CPU);
	//void readln(layer1_Neurons_CPU);

	void readln(float *Layer1_Neurons_CPU);

	float *Layer1_Neurons_GPU;
	//	float Layer1_Weights_CPU[156];
	float *Layer1_Weights_GPU;

	float *mean_square_error_Layer1_Neurons_CPU[841];
	float *mean_square_error_Layer1_Neurons_GPU;

	//	float Layer2_Weights_CPU[7800];
	float *Layer2_Weights_GPU;
	float *Layer2_Neurons_GPU;
	float *mean_square_error_Layer2_Neurons_GPU;
	float *mean_square_error_Layer2_Neurons_CPU[1014];

	//	float Layer3_Weights_CPU[125100];
	float *Layer3_Weights_GPU;
	float *Layer3_Neurons_GPU;

	float *mean_square_error_Layer3_Neurons_CPU[1250];

	float *mean_square_error_Layer4_Neurons_CPU[100];

	float *mean_square_error_Layer5_Neurons_CPU[10];
	float *mean_square_error_Layer6_Neurons_CPU[10];

	float *mean_square_error_Layer3_Neurons_GPU;



	//float Layer4_Weights_CPU[1010];
```

```
        float *New_Layer4_Weights_GPU;
        float *New_Layer3_Weights_GPU;
        float *New_Layer2_Weights_GPU;
        float *New_Layer1_Weights_GPU;

        float *Layer4_Neurons_GPU;

        float *mean_square_error_Layer4_Neurons_GPU;

        float Layer5_Neurons_CPU[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        float *Layer5_Neurons_GPU;

        float *mean_square_error_Layer5_Neurons_GPU;
        float *mean_square_error_Layer6_Neurons_GPU;

        float *delta_Layer5_Neurons_GPU;
        float *delta_Layer5_Neurons_CPU[10];

        float *delta_Layer4_Neurons_GPU;
        float *delta_Layer4_Neurons_CPU[100];

        float *delta_Layer3_Neurons_GPU;
        float *delta_Layer3_Neurons_CPU[1250];

        float *delta_Layer2_Neurons_GPU;
        float *delta_Layer2_Neurons_CPU[1014];

        float *delta_Layer1_Neurons_GPU;
        float *delta_Layer1_Neurons_CPU[841];

//      float *pinakas_Layer1_Neurons_CPU[841];

//      float *pinakas_Layer2_Neurons_CPU[1014];

        float *pinakas_Layer3_Neurons_CPU[1250];

        float *XOR_X_Layer1_Neurons_CPU[841];

        float *XOR_Y_Layer2_Neurons_CPU[1014];

        float *XOR_Z_Layer3_Neurons_CPU[1250];

        float *pinakas_Layer1_Neurons_GPU;

        float *pinakas_Layer2_Neurons_GPU;

        float *pinakas_Layer3_Neurons_GPU;

        float *XOR_X_Layer1_Neurons_GPU;

        float *XOR_Y_Layer2_Neurons_GPU;

        float *XOR_Z_Layer3_Neurons_GPU;




        double *outputLayer;
```

```
	void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU);
	InitHostMem;


	//FILE *pFile3 = 0;
	//pFile3 = fopen("lw3.wei", "rb");
	//if (pFile3 != NULL)
	//{
	//	for (int i = 0; i < 125100; ++i)
	//		fread(Layer3_Weights_CPU, sizeof(float), 125100, pFile3);
	//	fclose(pFile3);
	for (int i = 0; i < 125; ++i)
		printf("layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer3_Weights_CPU[i]));
	system("PAUSE");


	//}
	// initial layer 4 weight
	//	FILE *pFile4;
	//	pFile4 = fopen("lw4.wei", "rb");
	//	if (pFile4 != NULL)
	//	{
	//	for (int i = 0; i < 1010; ++i)
	//	fread(Layer4_Weights_CPU, sizeof(float), 1010, pFile4);
	//	fclose(pFile4);
	for (int i = 0; i < 1010; ++i)
		printf("layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer4_Weights_CPU[i]));
	system("PAUSE");


	//}

	//	FILE *pFile2;
	////	pFile2 = fopen("lw2.wei", "rb");
	//	if (pFile2 != NULL)
	//	{
	//	for (int i = 0; i < 7800; ++i)
	//	fread(Layer2_Weights_CPU, sizeof(float), 7800, pFile2);
	//	fclose(pFile2);

	for (int i = 0; i < 78; ++i)
		printf("layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer2_Weights_CPU[i]));
	system("PAUSE");
	//}


	//allocate momory on Device
	//	void InitGPUMem(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer3_Neurons_GPU, float
*Layer3_Weights_GPU, float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU, float
*Layer5_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*mean_square_error_Layer2_Neurons_GPU, float *mean_square_error_Layer3_Neurons_GPU, float
*mean_square_error_Layer4_Neurons_GPU, float *mean_square_error_Layer5_Neurons_GPU, float
*delta_Layer5_Neurons_GPU, float *New_Layer4_Weights_GPU, float
*delta_Layer4_Neurons_GPU, float *New_Layer3_Weights_GPU, float
*delta_Layer3_Neurons_GPU, float *New_Layer2_Weights_GPU, float
*delta_Layer2_Neurons_GPU, float *New_Layer1_Weights_GPU);
```

```
	//void InitGPUMem(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer3_Neurons_GPU, float
*Layer3_Weights_GPU, float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU, float
*Layer5_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*mean_square_error_Layer2_Neurons_GPU);



	cudaMalloc((void**)&Layer1_Neurons_GPU, sizeof(float) * 29 * 29);
	cudaMalloc((void**)&Layer1_Weights_GPU, sizeof(float) * 156);
	cudaMalloc((void**)&mean_square_error_Layer1_Neurons_GPU, sizeof(float) * 29 *
29);
	cudaMalloc((void**)&pinakas_Layer1_Neurons_GPU, sizeof(float) * 29 * 29);
	cudaMalloc((void**)&XOR_X_Layer1_Neurons_GPU, sizeof(float) * 29 * 29);

	cudaMalloc((void**)&Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
	cudaMalloc((void**)&Layer2_Weights_GPU, sizeof(float) * 7800);
	cudaMalloc((void**)&mean_square_error_Layer2_Neurons_GPU, sizeof(float) * 13 * 13
* 6);

	cudaMalloc((void**)&pinakas_Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
	cudaMalloc((void**)&XOR_Y_Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);

	cudaMalloc((void**)&Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&Layer3_Weights_GPU, sizeof(float) * 125100);

	cudaMalloc((void**)&mean_square_error_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&pinakas_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&XOR_Z_Layer3_Neurons_GPU, sizeof(float) * 1250);

	cudaMalloc((void**)&Layer4_Neurons_GPU, sizeof(float) * 100);
	cudaMalloc((void**)&Layer4_Weights_GPU, sizeof(float) * 1010);

	cudaMalloc((void**)&New_Layer4_Weights_GPU, sizeof(float) * 1010);
	cudaMalloc((void**)&New_Layer3_Weights_GPU, sizeof(float) * 125100);
	cudaMalloc((void**)&New_Layer2_Weights_GPU, sizeof(float) * 7800);
	cudaMalloc((void**)&New_Layer1_Weights_GPU, sizeof(float) * 156);

	cudaMalloc((void**)&mean_square_error_Layer4_Neurons_GPU, sizeof(float) * 100);

	cudaMalloc((void**)&Layer5_Neurons_GPU, sizeof(float) * 10);

	cudaMalloc((void**)&delta_Layer5_Neurons_GPU, sizeof(float) * 10);
	cudaMalloc((void**)&delta_Layer4_Neurons_GPU, sizeof(float) * 100);
	cudaMalloc((void**)&delta_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&delta_Layer2_Neurons_GPU, sizeof(float) * 1014);
	cudaMalloc((void**)&delta_Layer1_Neurons_GPU, sizeof(float) * 841);


	cudaMalloc((void**)&mean_square_error_Layer5_Neurons_GPU, sizeof(float) * 10);
	cudaMalloc((void**)&mean_square_error_Layer6_Neurons_GPU, sizeof(float) * 10);

	outputLayer = (double*)malloc(sizeof(double) * 10);

	cudaEvent_t Start, Stop;
	cudaEventCreate(&Start);
	cudaEventCreate(&Stop);


	//copy from CPU to GPU
```

```
	cudaMemcpy(Layer1_Neurons_GPU, Layer1_Neurons_CPU, sizeof(float) * 29 * 29,
cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer1_Neurons_GPU,
mean_square_error_Layer1_Neurons_CPU, sizeof(float) * 29 * 29, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer2_Neurons_GPU,
mean_square_error_Layer2_Neurons_CPU, sizeof(float) * 13 * 13 * 6,
cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer3_Neurons_GPU,
mean_square_error_Layer3_Neurons_CPU, sizeof(float) * 1250, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer4_Neurons_GPU,
mean_square_error_Layer4_Neurons_CPU, sizeof(float) * 100, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer5_Neurons_GPU,
mean_square_error_Layer5_Neurons_CPU, sizeof(float) * 10, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer6_Neurons_GPU,
mean_square_error_Layer6_Neurons_CPU, sizeof(float) * 10, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer1_Neurons_GPU, pinakas_Layer1_Neurons_CPU, sizeof(float) *
29 * 29, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer2_Neurons_GPU, pinakas_Layer2_Neurons_CPU, sizeof(float) *
13 * 13 * 6, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer3_Neurons_GPU, pinakas_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_X_Layer1_Neurons_GPU, XOR_X_Layer1_Neurons_CPU, sizeof(float) * 29
* 29, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_Y_Layer2_Neurons_GPU, XOR_Y_Layer2_Neurons_CPU, sizeof(float) * 13
* 13 * 6, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_Z_Layer3_Neurons_GPU, XOR_Z_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);




	cudaMemcpy(Layer1_Weights_GPU, Layer1_Weights_CPU, sizeof(float) * 156,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer2_Weights_GPU, Layer2_Weights_CPU, sizeof(float) * 7800,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer3_Weights_GPU, Layer3_Weights_CPU, sizeof(float) * 125100,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer4_Weights_GPU, Layer4_Weights_CPU, sizeof(float) * 1010,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer5_Neurons_GPU, Layer5_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);

	cudaMemcpy(delta_Layer5_Neurons_GPU, delta_Layer5_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer4_Neurons_GPU, delta_Layer4_Neurons_CPU, sizeof(float) *
100, cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer3_Neurons_GPU, delta_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);
```

```
	cudaMemcpy(delta_Layer2_Neurons_GPU, delta_Layer2_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer1_Neurons_GPU, delta_Layer1_Neurons_CPU, sizeof(float) *
841, cudaMemcpyHostToDevice);


	cudaMemcpy(New_Layer4_Weights_GPU, New_Layer4_Weights_CPU, sizeof(float) * 1010,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer3_Weights_GPU, New_Layer3_Weights_CPU, sizeof(float) * 125100,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer2_Weights_GPU, New_Layer2_Weights_CPU, sizeof(float) * 7800,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer1_Weights_GPU, New_Layer1_Weights_CPU, sizeof(float) * 156,
cudaMemcpyHostToDevice);



	cudaEventRecord(Start);
	dim3 Layer1_Block(6, 1);
	dim3 Layer1_Thread(13, 13);
	executeFirstLayer << <Layer1_Block, Layer1_Thread >> >(Layer1_Neurons_GPU,
Layer1_Weights_GPU, Layer2_Neurons_GPU, mean_square_error_Layer1_Neurons_GPU,
delta_Layer1_Neurons_GPU, delta_Layer2_Neurons_GPU, New_Layer1_Weights_GPU,
delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU, pinakas_Layer1_Neurons_GPU,
XOR_X_Layer1_Neurons_GPU);

	dim3 DimLayer2_Block(50, 1);
	dim3 DimLayer2_Thread(5, 5);
	executeSecondLayer << <DimLayer2_Block, DimLayer2_Thread >> >(Layer2_Neurons_GPU,
Layer2_Weights_GPU, Layer3_Neurons_GPU, mean_square_error_Layer2_Neurons_GPU,
delta_Layer2_Neurons_GPU, delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU,
delta_Layer4_Neurons_GPU, New_Layer3_Weights_GPU, pinakas_Layer2_Neurons_GPU,
XOR_Y_Layer2_Neurons_GPU);

	dim3 DimLayer3_Block(100, 1);
	dim3 DimLayer3_Thread(1, 1);
	executeThirdLayer << <DimLayer3_Block, DimLayer3_Thread >> >(Layer3_Neurons_GPU,
Layer3_Weights_GPU, Layer4_Neurons_GPU, mean_square_error_Layer3_Neurons_GPU,
delta_Layer3_Neurons_GPU, delta_Layer4_Neurons_GPU, New_Layer3_Weights_GPU,
Layer4_Weights_GPU, delta_Layer5_Neurons_GPU, pinakas_Layer3_Neurons_GPU,
XOR_Z_Layer3_Neurons_GPU, XOR_Y_Layer2_Neurons_GPU, XOR_X_Layer1_Neurons_GPU);

	dim3 DimLayer4_Block(10, 1);
	dim3 DimLayer4_Thread(1, 1);
	executeFourthLayer << <DimLayer4_Block, DimLayer4_Thread >> >(Layer4_Neurons_GPU,
Layer4_Weights_GPU, Layer5_Neurons_GPU, mean_square_error_Layer1_Neurons_GPU,
mean_square_error_Layer2_Neurons_GPU, mean_square_error_Layer3_Neurons_GPU,
mean_square_error_Layer4_Neurons_GPU, mean_square_error_Layer5_Neurons_GPU,
mean_square_error_Layer6_Neurons_GPU, delta_Layer4_Neurons_GPU, delta_Layer5_Neurons_GPU,
New_Layer4_Weights_GPU, delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU,
delta_Layer2_Neurons_GPU, New_Layer1_Weights_GPU, Layer3_Neurons_GPU, Layer3_Weights_GPU,
Layer2_Neurons_GPU, Layer2_Weights_GPU, Layer1_Neurons_GPU, Layer1_Weights_GPU,
XOR_Z_Layer3_Neurons_GPU, XOR_Y_Layer2_Neurons_GPU, XOR_X_Layer1_Neurons_GPU);



	cudaThreadSynchronize();
	cudaDeviceSynchronize();
	outputLayer = (double*)malloc(sizeof(double) * 10);
	//copy from GPU to CPU
	cudaMemcpy(Layer5_Neurons_CPU, Layer5_Neurons_GPU, sizeof(float) * 10,
cudaMemcpyDeviceToHost);
```

```
	cudaEventSynchronize(Stop);
	float milliseconds = 0;

	cudaEventElapsedTime(&milliseconds, Start, Stop);

	for (int i = 0; i < 10; ++i)
		printf("output layer [%d]=%d\n", outputLayer[i]);
	printf("xronos: deyterolepta %d\n start %d\n stop %d\n", milliseconds, Start,
Stop);




	for (int a = 0; a<10; a++)
	{
		outputLayer[a] = (double)Layer5_Neurons_CPU[a];
	}
	output(outputLayer);

	//float resulta;
	//printf("resulta apotelesma: %f\n", resulta);

	float Layer4_Neurons_CPU[100];
	cudaMemcpy(Layer4_Neurons_CPU, Layer4_Neurons_GPU, sizeof(float) * 100,
cudaMemcpyDeviceToHost);
	FILE *fp44 = fopen("layer_4.bin", "wb");
	//	for (int i = 0; i < 100; ++i)
	fwrite(Layer4_Neurons_CPU, sizeof(float), 100, fp44);
	fclose(fp44);
	for (int i = 0; i<100; ++i)
printf("layer4_Neurons_CPU NEOI NEYRONES apotelesma: %f\n", (Layer4_Neurons_CPU[i]));
	system("PAUSE");


	//float delta_Layer4_Neurons_CPU[100];
	cudaMemcpy(delta_Layer4_Neurons_CPU, delta_Layer4_Neurons_GPU, sizeof(float) *
100, cudaMemcpyDeviceToHost);
	FILE *fp4d = fopen("layer_4d.bin", "wb");
	//for (int i = 0; i < 100; ++i)
	fwrite(delta_Layer4_Neurons_CPU, sizeof(float), 100, fp4d);
	fclose(fp4d);




	float Layer3_Neurons_CPU[50 * 5 * 5];
	cudaMemcpy(Layer3_Neurons_CPU, Layer3_Neurons_GPU, sizeof(float) * 50 * 5 * 5,
cudaMemcpyDeviceToHost);
	FILE *fp33;
	fp33 = fopen("layer_3.bin", "wb");
	fwrite(Layer3_Neurons_CPU, sizeof(float), 50 * 5 * 5, fp33);
	fclose(fp33);
	for (int i = 0; i<1250; ++i)
printf("layer3_Neurons_CPU NEOI NEYRONES apotelesma: %f\n", (Layer3_Neurons_CPU[i]));
	system("PAUSE");


	float Layer2_Neurons_CPU[13 * 13 * 6];
```

```
	cudaMemcpy(Layer2_Neurons_CPU, Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6,
cudaMemcpyDeviceToHost);
	FILE *fp22 = fopen("layer_2.bin", "wb");
	//	for (int i = 0; i < 1014; ++i)
	fwrite(Layer2_Neurons_CPU, sizeof(float), 13 * 13 * 6, fp22);
	fclose(fp22);
	for (int i = 0; i<1014; ++i)
printf("layer2_Neurons_CPU NEOI NEYRONES apotelesma: %f\n", (Layer2_Neurons_CPU[i]));
	system("PAUSE");


	//float Layer1_Neurons_CPU[29 * 29];
	cudaMemcpy(Layer1_Neurons_CPU, Layer1_Neurons_GPU, sizeof(float) * 29 * 29,
cudaMemcpyDeviceToHost);
	FILE *fp11;
	fp11 = fopen("layer_1.bin", "wb");
	//for (int i = 0; i < 841; ++i)
	fwrite(Layer1_Neurons_CPU, sizeof(float), 29 * 29, fp11);
	fclose(fp11);
	for (int i = 0; i<841; ++i)
printf("layer1_Neurons_CPU NEOI NEYRONES apotelesma: %f\n", (Layer1_Neurons_CPU[i]));

	system("PAUSE");

	printf("mean_square_error_Layer2_Neurons_GPU: %f\n",
mean_square_error_Layer2_Neurons_GPU);
	printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer4_Neurons_GPU: %f\n",
mean_square_error_Layer4_Neurons_GPU);
	printf("mean_square_error_Layer5_Neurons_GPU: %f\n",
mean_square_error_Layer5_Neurons_GPU);
	printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU);
	system("PAUSE");

	printf("mean_square_error_Layer2_Neurons_GPU: %d\n",
mean_square_error_Layer2_Neurons_GPU);
	printf("mean_square_error_Layer1_Neurons_GPU: %d\n",
mean_square_error_Layer1_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %d\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer4_Neurons_GPU: %d\n",
mean_square_error_Layer4_Neurons_GPU);
	printf("mean_square_error_Layer5_Neurons_GPU: %d\n",
mean_square_error_Layer5_Neurons_GPU);
	printf("mean_square_error_Layer6_Neurons_GPU: %d\n",
mean_square_error_Layer6_Neurons_GPU);
//	printf("mean_square_error_Layer5_Neurons_GPU: %d =
mean_square_error_Layer1_Neurons_GPU %d + mean_square_error_Layer2_Neurons_GPU %d +
mean_square_error_Layer3_Neurons_GPU %d + mean_square_error_Layer4_Neurons_GPU %d\n",
mean_square_error_Layer5_Neurons_GPU);

	printf("mean_square_error_Layer2_Neurons_CPU: %d\n",
mean_square_error_Layer2_Neurons_CPU);
	printf("mean_square_error_Layer1_Neurons_CPU: %d\n",
mean_square_error_Layer1_Neurons_CPU);
	printf("mean_square_error_Layer3_Neurons_CPU: %d\n",
mean_square_error_Layer3_Neurons_CPU);
```

```
	printf("mean_square_error_Layer4_Neurons_CPU: %d\n",
mean_square_error_Layer4_Neurons_CPU);
	printf("mean_square_error_Layer5_Neurons_CPU: %d\n",
mean_square_error_Layer5_Neurons_CPU);
	printf("mean_square_error_Layer6_Neurons_CPU: %d\n",
mean_square_error_Layer6_Neurons_CPU);

	system("PAUSE");


	printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", delta_Layer2_Neurons_GPU);
	printf("nl1wb New_Layer1_Weights_GPU: %d\n", New_Layer1_Weights_GPU);

	printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", delta_Layer3_Neurons_GPU);
	printf("nl2wb New_Layer2_Weights_GPU: %d\n", New_Layer2_Weights_GPU);

	printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", delta_Layer4_Neurons_GPU);
	printf("nl3wb New_Layer3_Weights_GPU: %d\n", New_Layer3_Weights_GPU);

	printf("dlta5 delta_Layer5_Neurons_GPU: %d\n", delta_Layer5_Neurons_GPU);
	printf("nl4w New_Layer4_Weights_GPU: %d\n", New_Layer4_Weights_GPU);

	printf("dlta2b delta_Layer2_Neurons_CPU: %d\n", delta_Layer2_Neurons_CPU);
	printf("nl1wb New_Layer1_Weights_CPU: %d\n", New_Layer1_Weights_CPU);

	printf("dlta3b delta_Layer3_Neurons_CPU: %d\n", delta_Layer3_Neurons_CPU);
	printf("nl2wb New_Layer2_Weights_CPU: %d\n", New_Layer2_Weights_CPU);

	printf("dlta4b delta_Layer4_Neurons_CPU: %d\n", delta_Layer4_Neurons_CPU);
	printf("nl3wb New_Layer3_Weights_CPU: %d\n", New_Layer3_Weights_CPU);

	printf("dlta5 delta_Layer5_Neurons_CPU: %d\n", delta_Layer5_Neurons_CPU);
	printf("nl4w New_Layer4_Weights_CPU: %d\n", New_Layer4_Weights_CPU);


	system("PAUSE");

	printf("pinakas_Layer1_Neurons_GPU: %d\n", pinakas_Layer1_Neurons_GPU);
	printf("pinakas_Layer2_Neurons_GPU: %d\n", pinakas_Layer2_Neurons_GPU);
	printf("pinakas_Layer3_Neurons_GPU: %d\n", pinakas_Layer3_Neurons_GPU);

	for (int i = 0; i<10; ++i)
		printf("pinakas_layer1_Neurons_CPU: %f\n",
(pinakas_Layer1_Neurons_CPU[i]));
	system("PAUSE");


	for (int i = 0; i<10; ++i)
		printf("pinakas_layer2_Neurons_CPU: %f\n",
(pinakas_Layer2_Neurons_CPU[i]));
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("pinakas_layer3_Neurons_CPU: %f\n",
(pinakas_Layer3_Neurons_CPU[i]));
	system("PAUSE");



	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
```

```
	//printf("XOR_Z_Layer3_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU);
	//printf("XOR_Y_Layer2_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU);

	for (int i = 0; i<10; ++i)
		printf("XOR_X_Layer1_Neurons_CPU: %f\n", XOR_X_Layer1_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Y_Layer1_Neurons_CPU: %f\n", XOR_Y_Layer2_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Z_Layer1_Neurons_CPU: %f\n", XOR_Z_Layer3_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_X_Layer1_Neurons_CPU: %d\n", XOR_X_Layer1_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Y_Layer1_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Z_Layer1_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU[i]);
	system("PAUSE");


	printf("XOR_X_Layer1_Neurons_CPU: %d\n", XOR_X_Layer1_Neurons_CPU);
	printf("XOR_Y_Layer2_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU);
	printf("XOR_Z_Layer3_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU);


	system("PAUSE");

	printf("XOR_Z_Layer3_Neurons_GPU: %f\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %f\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %f\n", XOR_X_Layer1_Neurons_GPU);


	exit(0);
}
```

ΚΑΡΤΑ ΓΡΑΦΙΚΩΝ GT710 ΤΗΣ NVIDIA ΣΤΟΝ ΣΤΑΘΕΡΟ ΥΠΟΛΟΓΙΣΤΗ ΠΟΥ ΕΓΙΝΑΝ ΟΙ ΜΕΤΡΗΣΕΙΣ

Η Palit, παρουσιάζει τη σειρά καρτών γραφικών GT710 της NVIDIA! Με αυτόνομη μνήμη video 1GB DDR3 και 192 επεξεργαστές Cuda αποδίδει γραφικά υψηλής ανάλυσης, για gaming και ομαλή αναπαραγωγή βίντεο. Με passive σύστημα ψύξης εξασφαλίζει αθόρυβη λειτουργία και χαμηλές θερμοκρασίες λειτουργίας, ενώ υποστηρίζει τα τελευταία πρότυπα απεικόνισης DirectX 12! . Με Thunder Master Overclocking Tool.

- **Chipset:** NVidia.
- **Σύνδεση**PCI Express 8x.
- **Engine Clock** 954 MHz.
- **Direct X υποστήριξη:** Direct X 10.
- **Memory Interface:** 64 bit.
- **Μέγεθος Μνήμης:** 1 GB.
- **Ταχύτητα Μνήμης:** 800 MHz.
- **Τύπος Μνήμης:** DDR3.
- **Έξοδοι:** 1x D-sub out / 1x DVI / 1x HDMI.
- **HDCP υποστήριξη:** Ναι.
- **HDTV ready:** Ναι.
- **RAMDAC:** 400 MHz.

Στον φορητό υπολογιστή που τρέξαμε το πρόγραμμα είχε μια κάρτα γραφικών της NVIDIA GeForce MX110 με αυτόνομη μνήμη video 2 GB και 384 πυρήνες

The GeForce MX110 is a graphics card by NVIDIA, launched in November 2017. Built on the 28 nm process, and based on the GM108 graphics processor, in its N16V-GMR1-A1 variant, the card supports DirectX 12.0. It features 384 shading units, 24 texture mapping units and 8 ROPs. NVIDIA

has placed 2,048 MB GDDR5 memory on the card, which are connected using a 64-bit memory interface. The GPU is operating at a frequency of 965 MHz, which can be boosted up to 993 MHz, memory is running at 1253 MHz.
Being a igp card, the NVIDIA GeForce MX110 does not require any additional power connector, its power draw is rated at 30 W maximum. This device has no display connectivity, as it is not designed to have monitors connected to it. GeForce MX110 is connected to the rest of the system using a PCI-Express 3.0 x16 interface.

NVIDIA GeForce MX110

GM108
GRAPHICS PROCESSOR

384
CORES

24
TMUS

8
ROPS

2 GB
MEMORY SIZE

GDDR5
MEMORY TYPE

64 bit
BUS WIDTH

Memory

Memory Size	2 GB
Memory Type	GDDR5
Memory Bus	64 bit
Bandwidth	40.10 GB/s

Graphics Features

DirectX	12.0 (11_0)
OpenGL	4.6
OpenCL	1.2
Vulkan	1.1.92
CUDA	5.0

Shader Model	
5.1	

Graphics Card

Release Date	
Nov 17th, 2017	
Generation	
GeForce MX	
Production	
Active	
Bus Interface	
PCIe 3.0 x16	

Clock Speeds

GPU Clock	
965 MHz	
Boost Clock	
993 MHz	
Memory Clock	
1253 MHz 5012 MHz effective	

Render Config

Shading Units	
384	
TMUs	
24	
ROPs	
8	
SMM Count	
3	

Theoretical Performance

Pixel Rate	
7.944 GPixel/s	
Texture Rate	
23.83 GTexel/s	
FP32 (float) performance	
762.6 GFLOPS	
FP64 (double) performance	
23.83 GFLOPS (1:32	

Board Design

Slot Width	
IGP	
TDP	
30 W	
Outputs	
No outputs	
Power Connectors	
None	

Graphics Processor

GPU Name	
GM108	
GPU Variant	
N16V-GMR1-A1	
Architecture	
Maxwell	
Foundry	
TSMC	

Process Size	
28 nm	
Transistors	
unknown	
Die Size	
unknown	

https://www.techpowerup.com/gpu-specs/geforce-mx110.c3044

οι πληροφοριες όσον αφορά την κάρτα γραφικών πάρθηκαν απο την ιστοσελίδα techpowerup.com

Ο κώδικας που έφτιαξα πατάει πάνω σε ένα προηγούμενο κώδικα που πήρα απο το **web site codeproject.com** με δημιουργούς τους εξής: Kavinguy ,Phd Student of University of California, Davis και billconan ,Phd Student of University of California, Davis. Ο κώδικας στην αρχική του μορφή ήταν σε 2 τμήματα ,το ένα ήταν το GUI source code και το άλλο ήταν το The Neural Network core. Και τα 2 σκέλη του προγράμματος ήταν γραμμένα σε CUDA C. Παλιότερη έκδοση αυτού του κώδικα είχε αναπτυχθεί απο τον προγραμματιστή Mike O'Neill γραμμένο σε γλώσσα C/C++ πάλι κάνοντας χρήση των Νευρωνικών Δικτύων .Ο κώδικας του Mike O'Neill δημιουργήθηκε με σκοπό την Αναγνώριση των χειρόγραφων ψηφίων (Handwritten Digits) κάνοντας χρήση των Νευρωνικών Δικτύων.

Ο κώδικας που πήρα είναι γραμμένος σε CUDA C απο τους Kavinguy ,Phd Student of University of California, Davis και billconan ,Phd Student of University of California, Davis και είναι μετεξέλιξη του πρότερου κώδικα του Mike O'Neill πάλι για την αναγνώριση των χειρόγραφων ψηφίων με χρήση των Νευρωνικών Δικτύων.

Ο δικός μου κώδικας λοιπόν τροποποιήθηκε αρκετά ,τα 2 τμήματα του κώδικα έγιναν 1 ,κάποιες εντολές που υπήρχαν και στα 2 προγράμματα αφαιρέθηκαν , έγιναν και κάποιες άλλες αλλαγές , μπήκαν νέες εντολές , κάνει και άλλους υπολογισμούς όπως δέλτα σφάλμα, υπολογίζει Νέο Βάρος , Το Μέσο τετραγωνικό σφάλμα.

Το νέο λοιπόν πρόγραμμα έχει πάρει την μορφή που έχει τώρα και εκπαιδέυεται πάνω σε μια XOR 2 εισόδων και 1 εξόδου με τις όποιες αστοχίες που μπορεί να έχει σαν κώδικας.

Ο κώδικας έτρεξε και σε σταθερό υπολογιστή και σε φορητό υπολογιστή. Ο σταθερός είχε κάρτα γραφικών NVIDIA GT710 με 1GB DDR3 και 192 επεξεργαστές CUDA.

Ο φορητός είχε κάρτα γραφικών της NVIDIA MX110 με 2GB και 384 επεξεργαστές CUDA

Λίγα Λόγια για το Νευρωνικό Δίκτυο που έχουμε Υλοποιήσει

Το Νευρωνικό Δίκτυο που υλοποιούμε έχει 5 επίπεδα {layers) δικτύου που ονομάζεται συνελικτικό δίκτυο (Convolutional Neural Network). Το συνελικτικό δίκτυο έχει αποδειχθεί ότι είναι κατάλληλο για αναγνώριση χαρακτήρων (handwritten digits).

Τα πρώτα 3 επίπεδα του δικτύου μας αποτελούνται απο πολλά feature maps (χαρακτηριστικούς χάρτες) . Ο κάθε ένας απο αυτούς είναι συνέχεια απο τον προηγούμενο επίπεδο (layer) . Η είσοδός μας είναι μια εικόνα ανάλυσης 29 χ 29 ψηφίων . Οπότε έχουμε 29 χ 29 = 841 ψηφία/νευρώνες στο πρώτο επίπεδο (layer). Το δεύτερο επίπεδο (layer) είναι ενα συνελικτικό επίπεδο (convolutional layer) με έξι χαρακτηριστικούς χάρτες (feature maps). Ο κάθε χαρακτηριστικός χάρτης (feature map) είναι μια 13 χ 13 εικόνα που έχει παρθεί απο το πρώτο επίπεδο (layer). Το κάθε pixel/neuron στον χαρακτηριστικό χάρτη (feature maps) είναι ενας συνελικτικός πυρήνας (convolutional kernel) διαστάσεων 5 χ 5 στο επίπεδο εισόδου (input layer). Οπότε έχουμε 13 χ 13 χ 6 = 1014 κομβοι/νευρώνες (nodes/neurons) σε αυτό το επίπεδο και (5 χ 5 + 1 (bias) node)) x 6 = 156 Βάρη (Weights). Μετά έχουμε 1014 χ (5 χ 5 +1) = 26364 συνδέσεις που συνδέονται στο πρώτο επίπεδο.

Το επίπεδο 3 (layer 3) είναι και αυτό ένα συνελικτικό επίπεδο (convolutional layer) αλλά αυτό έχει 50 μικρότερους χαρακτηριστικούς χάρτες (feature maps). Ο κάθε χαρακτηριστικός χάρτης (feature maps) είναι 5 χ 5 στο μέγεθος και το κάθε pixel στον κάθε χαρακτηριστικό χάρτη (feature map) είναι ένας συνελικτικός πυρήνας (convolutional kernel) των αντιπροσωπευτικών περιοχών όλων των 6 χαρακτηριστικών χαρτών (feature maps) του προηγούμενου επιπέδου. Οπότε εδώ έχουμε 5 χ 5 χ 50 = 1250 Νευρώνες (neurons) σε αυτό το επίπεδο (layer) , (5 χ 5 + 1) χ 6 χ 50 = 7800 Βάρη (Weights) και 1250 χ 26 = 32500 συνδέσεις.

Το επίπεδο 4 (layer 4) είναι ενα πλήρως συνδεδεμένο επίπεδο με 100 νευρώνες. Και αφού μιλάμε για πλήρως συνδεδεμένο δίκτυο/επίπεδο κάθε ένας απο τους 100 νευρώνες του επιπέδου συνδέεται με όλους τους 1250 νευρώνες του προηγούμενου επιπέδου. Οπότε έχουμε 100 νευρώνες μέσα σε αυτό άρα λοιπόν έχουμε 100 χ (1250 + 1(bias)) = 125100 Βάρη (Weights) και 100 χ 1251 = 125100 συνδέσεις.

Το επίπεδο 5 (layer 5) είναι το τελικό επίπεδο ,επίπεδο εξόδου, Είναι και αυτό πλήρως συνδεδεμένο επίπεδο/δίκτυο με 10 μονάδες . Ο κάθε ένας απο του 10 νευρώνες σε αυτό το επίπεδο είναι συνδεδεμένο με όλους τους 100 ευρώνες του προηγούμενου επιπέδου. Υπάρχουν 10 νευρώνες στο επίπεδο 5,οπότε έχουμε 10 χ (100 + 1) 1010 Βάρη (Weights) και 10 χ 101 = 1010 συνδέσεις.

Παρατηρούμε λοιπόν ότι έχουμε να κάνουμε με μια μεγάλη βάση .

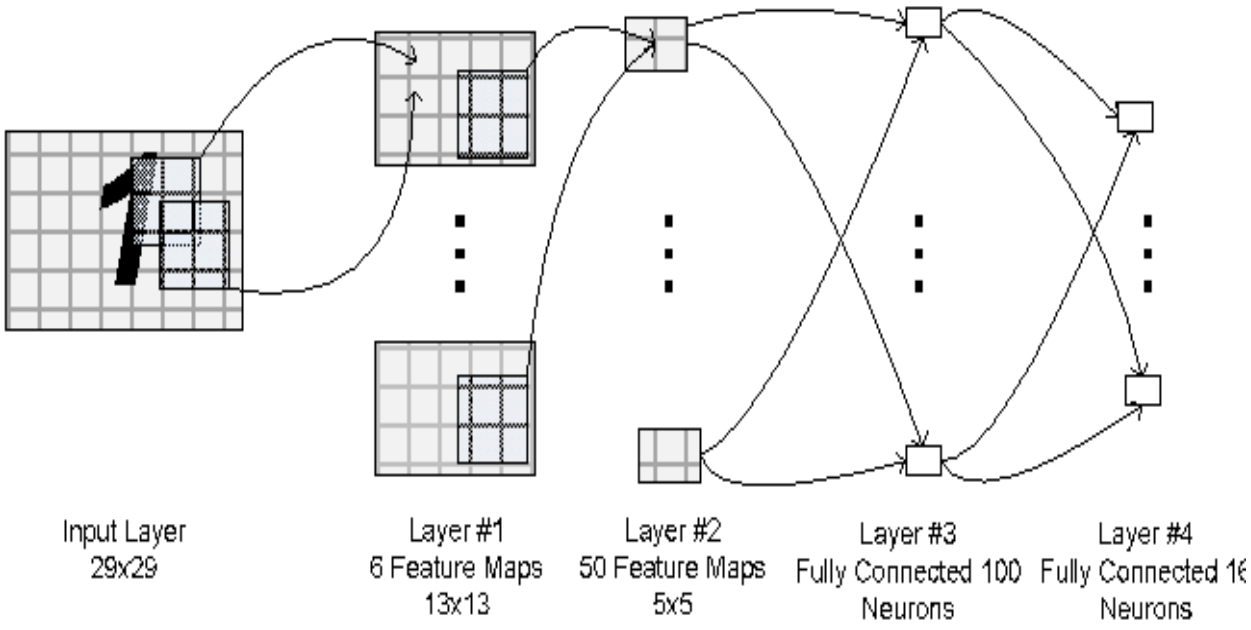
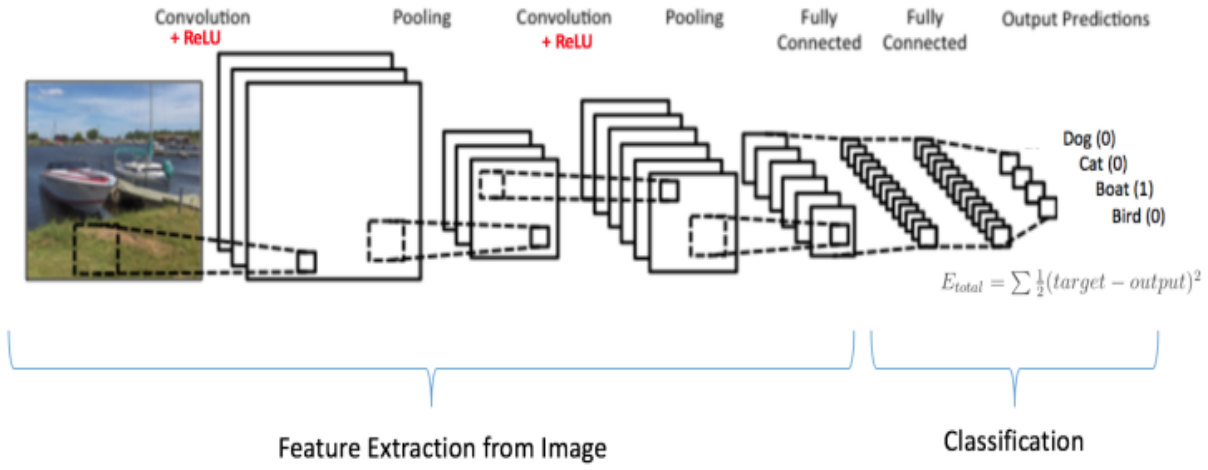


Fig. 3. Convolutional neural network

https://www.semanticscholar.org/paper/Improved-neural-network-OCR-based-on-preprocessed-Fedorovici-Voisan/2e9fa1e9717f80f8bfea4f14b30568dd6c99c9f4



https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Επεξήγηση επιμέρους τμημάτων του κώδικα

Η τιμή του κάθε νευρώνα του δευτέρου επιπέδου είναι ζυγισμένη αθροιστικά με 25 νευρώνες (5 χ 5) του πρώτου επιπέδου και μια πόλωση (bias).

Το δεύτερο επίπεδο νευρώνων αποτελείται απο 6 χαρακτηριστικούς χάρτες (feature maps) που ο κάθε ένας έχει μέγεθος 13 χ 13 pixels που δειγματοληπτείται απο το πρώτο επίπεδο.

Ορίζουμε **blockID** για κάθε **χαρακτηριστικό χάρτη (feature map)**

Ορίζουμε **threadID** για κάθε **νευρώνα του χαρακτηριστικού χάρτη (feature map)**

Ο κάθε χαρακτηριστικός χάρτης (feature map) χειρίζεται απο ένα block και κάθε pixel μέσα σε αυτό διαχειρίζεται απο ένα thread

Η συνάρτηση μακροεντολή __global__ σημαίνει ότι η συνάρτηση που δηλώνεται είναι μια συνάρτηση πυρήνα της Cuda. Θα εκτελεστεί στην συσκευή device και καλείται απο την CPU για να δημιουργήσει ένα πλέγμα νημάτων σε μια συσκευή.

Ο παρακάτω κώδικας τυπικά υπολογίζει το δεύτερο επίπεδο

```
__global__ void executeFirstLayer(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU,
float *Layer2_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*delta_Layer1_Neurons_GPU, float *delta_Layer2_Neurons_GPU, float
*New_Layer1_Weights_GPU, float *delta_Layer3_Neurons_GPU, float *Layer2_Weights_GPU,
float *pinakas_Layer1_Neurons_GPU, float *XOR_X_Layer1_Neurons_GPU)
{
	int blockID = blockIdx.x; //ορισμος του block με τιμη απο 0 εως N – 1 , κάθε μπλοκ
αποτελείται απο M νήματα καθένα απο μια τιμή threadIdx.x που έχει έυρος τιμών
 απο 0 έως M -1. Ολα τα μπλοκ σε επίπεδο μπλέγματος είναι οργανωμένα με τη μορφή
μονοδίαστατου πίνακα ,όλα τα νήματα μέσα στο μπλοκ είναι επίσης οργανωμένα με την μορφή
ενός μονοδιάστατου πίνακα . Καθε πλέγμα διαθέτει ένα σύνολο N x M νημάτων.
	int pixelX = threadIdx.x; //νημα χ = πιχελ χ
	int pixelY = threadIdx.y; //νημα y = πιχελ y
	//	float mean_square_error_Layer1_Neurons_GPU;
	float target; //Στοχος – Επιθυμητο αποτελεσμα

	int XOR1;
	int PIN1;
	float DLTA2B;
	float DLTA1B = 0; //Αρχικοποιηση
	float DLTA2;
//	float DLTA2B = 0;
	float DLTA3B;
	float NL1W = 0; //Αρχικοποιηση
	float NL1WB = 0; //Αρχικοποιηση
	int a = 0.5; //learning rate//
 //ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ blockID
	int dlta2 = blockID * 26;
	int dlta2b = blockID * 26;
//	int dlta3b = blockID * 26 * 6;
	int dlta1b = blockID * 26;
				//convolutional kernel,ΤΙΜΕΣ ΤΟΥ ΠΥΡΗΝΑ ΣΑΝ ΠΙΝΑΚΑΣ
	int kernelTemplate[25] = { //κερνελ (πυρήνας) 5 χ 5 = 25
		0, 1, 2, 3, 4,  //βασικά είναι οι τιμές που θα παρθούν – σαρωθούν
		29, 30, 31, 32, 33, //απο την αρχική εικόνα απο το δισδιάστατο εκπαιδευόμενο
		58, 59, 60, 61, 62, //φίλτρο/kernel
		87, 88, 89, 90, 91,
		116, 117, 118, 119, 120 };
```

```
	int weightBegin = blockID * 26; // ((5 χ 5) + 1 (bias)) //ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ
blockID
	int windowX = pixelX * 2; //διαστασεις του καθε πιχελ χ 2  ισο με το windowX της
εικόνας (29 χ 29) ,είναι τυπικά τα τρισδιάστατα Νήματα (threads) κατά Χ κατεύθυνση
	int windowY = pixelY * 2; //διαστασεις του καθε πιχελ χ 2  ισο με το windowY της
εικόνας (29 χ 29) ,είναι τυπικά τα τρισδιάστατα Νήματα (threads) κατά Υ κατεύθυνση
 στο input Layer ή Layer1

	    //ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ blockID

	int mse1 = blockID * 26;
	int pinak1 = blockID * 26;
	int xor1 = blockID * 26;
	//int pinak1;
	//int xor1;


	float result = 0; //Αρχικοποιηση result - πραγματικο αποτελεσμα
	float MSE1 = 0;
	float resulta;

	//ΒΑΡΟΣ ΣΤΟ ΕΠΙΠΕΔΟ 1 (LAYER 1)
	result += Layer1_Weights_GPU[weightBegin];
	//++mse1;
	++weightBegin;
	dlta2++;
	dlta2b++;
	++pinak1;
	++xor1;

	PIN1 = pinakas_Layer1_Neurons_GPU[pinak1]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
	XOR1 += XOR_X_Layer1_Neurons_GPU[xor1];   //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ

	DLTA2 = delta_Layer2_Neurons_GPU[dlta2];
	NL1W = New_Layer1_Weights_GPU[weightBegin];
	DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
	//DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
	DLTA1B = delta_Layer1_Neurons_GPU[dlta1b];
	NL1WB = New_Layer1_Weights_GPU[weightBegin];

	PIN1 = { (00, 01, 10, 11)}; //ΤΙΜΕΣ ΓΙΑ ΤΗΝ XOP1

	for (int i = 0; i<25; ++i)
	{
		result += Layer1_Neurons_GPU[windowY * 29 + windowX + kernelTemplate[i]] *
Layer1_Weights_GPU[weightBegin + i]; //το κάθε νευρώνιο που αντιστοιχεί σε κάθε pixel για
την εικονα 29 χ 29 = 841 νευρωνες + πυρήνα 5 χ 5 πολ/μενο με το βάρος κάθε νευρώνα του
πρωτου επιπέδου μας δίνει το αποτέλεσμα.  Βασικά εδώ γίνεται το εξής,οι Νευρώνες που τυπικά
είναι τα pixels της εικόνας 29 Χ 29 πολλαπλιασιάζονται με τα Βάρη  των εκπαιδευόμενων φίλτρων
,στην προκειμένη περίπτωση είναι τα βάρη (νήματα) του φίλτρου /kernel διαστάσεων 5 Χ 5 = 25 +
1 (bias/πόλωση) = 26


		printf("apotelesma: %d\n", result);
	//	XOR1 = PIN1 * result;
		printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
		printf("pinakas_Layer1_Neurons_GPU: %d\n", pinakas_Layer1_Neurons_GPU);
//		system("PAUSE");

	}
```

```
       printf("Layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);

   // Σιγμοειδής Συνάρτηση ενεργοποίησης f
  // Συναρτηση υπερβολικης εφαπτόμενης με περιορισμούς για α = 1.7159 και β = 0.67
       result = (1.7159*tanhf(0.66666667*result));
       int i;



  //ΥΠΟΛΟΓΙΣΜΟΣ ΔΕΛΤΑ ΣΦΑΛΜΑ 1

       DLTA1B = -((result * (1 - result) * DLTA2B * Layer1_Weights_GPU[weightBegin +
i]));
       printf("dlta1b delta_Layer1_Neurons_GPU: %d\n", DLTA1B);

 //Η XOR1 ΕΙΝΑΙ ΤΟ PIN1 ΜΕ ΤΙΣ ΤΙΜΕΣ ΤΗΣ XOR1 ΠΟΛ/ΝΟ ΜΕ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΤΗΣ ΣΥΓΜΟΕΙΔΟΥΣ
ΣΥΝΑΡΤΗΣΗΣ ΥΠΕΡΒΟΛΙΚΗΣ ΕΦΑΠΤΟΜΕΝΗΣ
       XOR1 = PIN1 * result;

       printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//     system("PAUSE");


       resulta = result;
       printf("Layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);
       printf("apotelesma: %d\n", resulta);
       //Layer2_Neurons_GPU[13 * 13 * blockID + pixelY * 13 + pixelX] = result;

       printf("layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);

       printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

       printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

       printf("layer1_weights_GPU: %f\n", Layer1_Weights_GPU);



       target = 1;
       MSE1 = mean_square_error_Layer1_Neurons_GPU[mse1];

     //Μεσο τετραγωνικο σφαλμα
       MSE1 = (1 / 2 * (target - result));
       //mean_square_error_Layer1_Neurons_GPU = (1 / 2 * (target - result));
       //DLTA2 = -((target - result) * result * (1 - result));
       NL1W = (Layer1_Weights_GPU[weightBegin] - a * DLTA2); //new weight layer2
//     int i;
       //DLTA2B = -((result * (1 - result) * DLTA3B * Layer2_Weights_GPU[weightBegin +
i]));

      //ΥΠΟΛΟΓΙΣΜΟΣ ΝΕΟΥ ΒΑΡΟΥΣ 1
NL1WB = (Layer1_Weights_GPU[weightBegin] - a * DLTA2B * Layer1_Neurons_GPU[windowY * 29 +
windowX + kernelTemplate[i]); //new weight layer1

       printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
       printf("mean_square_error_Layer1_Neurons_GPU: %f\n", MSE1);
       printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
       printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
       printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
```

```
        printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
        printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
        printf(" New_Layer1_Weights_GPU: %d\n", NL1WB);
        printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
        printf("nl1wb New_Layer1_Weights_GPU: %d\n", NL1WB);

         //Δηλωση εικονας διαστασεων 13 χ 13 pixels/Νευρώνων για το δευτερο επίπεδο
        Layer2_Neurons_GPU[13 * 13 * blockID + pixelY * 13 + pixelX] = result;


//      XOR1 = PIN1 * result;
        printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//      system("PAUSE");

}
```

Το layer 2 εχει 6 feature maps, το κάθε ένα map είναι 13 χ 13 pixel/neuron/image. Τα 6 feature maps τα δεσμεύει απο τα 6 block του layer_1 (6,1)
Τα νήματα /pixels/neurons σύνολο 169 νήματα τα παίρνει απο το thread layer_1 (13,13) . Ορίζουμε block_ID για κάθε feature map και thread_ID για κάθε νευρώνα.

Ο παρακάτω κώδικας τυπικά υπολογίζει το τρίτο επίπεδο

```
__global__ void executeSecondLayer(float *Layer2_Neurons_GPU, float *Layer2_Weights_GPU,
float *Layer3_Neurons_GPU, float *mean_square_error_Layer2_Neurons_GPU, float
*delta_Layer2_Neurons_GPU, float *delta_Layer3_Neurons_GPU, float
*New_Layer2_Weights_GPU, float *delta_Layer4_Neurons_GPU, float *Layer3_Weights_GPU,
float *pinakas_Layer2_Neurons_GPU, float *XOR_Y_Layer2_Neurons_GPU)
{
        int blockID = blockIdx.x; //ορισμος του block με τιμη απο 0 εως N – 1 , κάθε μπλοκ
αποτελείται απο M νήματα καθένα απο μια τιμή threadIdx.x που έχει έυρος τιμών
 απο 0 έως M -1. Ολα τα μπλοκ σε επίπεδο μπλέγματος είναι οργανωμένα με τη μορφή
μονοδίαστατου πίνακα ,όλα τα νήματα μέσα στο μπλοκ είναι επίσης οργανωμένα με την μορφή
ενός μονοδιάστατου πίνακα . Καθε πλέγμα διαθέτει ένα σύνολο N x M νημάτων.

        int pixelX = threadIdx.x; //νημα χ = πιχελ χ

        int pixelY = threadIdx.y; //νημα y = πιχελ y
        //      float mean_square_error_Layer2_Neurons_GPU;
        float target; //Στοχος – Επιθυμητο αποτελεσμα

//      float DLTA4B;
        float DLTA3;
        float NL2W = 0; //Αρχικοποιηση
        float DLTA3B;
        float DLTA2B = 0; //Αρχικοποιηση
        //float DLTA1B = 0;
        float NL2WB = 0; //Αρχικοποιηση
        int a = 0.5; //learning rate//
        int dlta3 = blockID * 26 * 6;
        int dlta2b = blockID * 26 * 6;
        //int dlta4b = blockID * 1251;
        int dlta3b = blockID * 26 * 6;
        // (5 x 5) + 1 (bias)
        //int dlta1b = blockID * 26;
        int XOR2;
        int PIN2;
     //kernel πυρηνας (5 χ 5) = 25
     //convolutional kernel, ΤΙΜΕΣ ΤΟΥ ΠΥΡΗΝΑ ΣΑΝ ΠΙΝΑΚΑΣ
```

```
	int kernelTemplate2[25] = {
		0, 1, 2, 3, 4,
		13, 14, 15, 16, 17,
		26, 27, 28, 29, 30,
		39, 40, 41, 42, 43,
		52, 53, 54, 55, 56 };

//ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ blockID

	int weightBegin = blockID * 26 * 6;
   // 6 feature maps
	int windowX = pixelX * 2; //διαστασεις του καθε πιχελ χ 2  ισο με το windowX της εικόνας (29 χ 29)

	int windowY = pixelY * 2; //διαστασεις του καθε πιχελ χ 2  ισο με το windowY της εικόνας (29 χ 29)

//ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ blockID

	int mse2 = blockID * 26;
	int pinak2 = blockID * 26 * 6;
	int xor2 = blockID * 26 * 6;
	//int pinak2;
	//int xor2;


	float result = 0;
	float MSE2 = 0;
	//float MSE3 = 0;
	//result += Layer2_Weights_GPU[weightBegin];


	result += Layer2_Weights_GPU[weightBegin];
	//PIN2[4*1] = { 0, 1, 0, 1 };
//	float pinakas_Layer2_Neurons_GPU[4 * 1] = { 0, 1, 0, 1 };

  // PIN2 =   {( 0, 0, 0,
	//		0, 1, 1,
	//		1, 0, 1,
	//		1, 1, 0) };
	//PIN2 = {
	//	{ ( 0, 0 ), ( 0 ) },
	//	{ ( 0, 1 ), ( 1 ) },
	//	{ ( 1, 0 ), ( 1 ) },
	//	{ ( 1, 1 ), ( 0 ) }
	//};

     PIN2 = { (00, 01, 10, 11)}; //ΤΙΜΕΣ ΓΙΑ ΤΗΝ XOP2


	if (blockID == 1 && pixelX == 0 && pixelY == 0)
	{
		result += 0;
	}

	++weightBegin;
	//++mse2;
	dlta3++;
	dlta3b++;
	++pinak2;
	++xor2;
```

```
        PIN2 = pinakas_Layer2_Neurons_GPU[pinak2];
        XOR2 += XOR_Y_Layer2_Neurons_GPU[xor2];

        DLTA3 = delta_Layer3_Neurons_GPU[dlta3];
        NL2W = New_Layer2_Weights_GPU[weightBegin];
//      DLTA4B = delta_Layer4_Neurons_GPU[dlta4b];
        DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];
        DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
//      DLTA1B = delta_Layer1_Neurons_GPU[dlta1b];
        //New_Layer2_weights - Νεο Βαρος στο Layer 2
        NL2WB = New_Layer2_Weights_GPU[weightBegin];

        for (int i = 0; i<25; ++i)
        {
//ΥΠΟΛΟΓΙΣΜΟΣ RESULT ΑΠΟ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟ ΝΕΥΡΩΝΩΝ/PIXEL ΑΠΟ ΤΟΝ FEATURE MAP ΠΟΥ ΕΙΝΑΙ ΜΙΑ
ΕΙΚΟΝΑ 13 X 13 ΠΟΥ ΕΧΕΙ ΠΑΡΘΕΙ ΑΠΟ ΤΟ ΠΡΩΤΟ ΕΠΙΠΕΔΟ (LAYER1) ,ΤΟ ΚΑΘΕ PIXEL/NEURON ΣΤΟ
FEATURE MAP ΕΙΝΑΙ ΕΝΑΣ CONVOLUTIONAL KERNEL (ΣΥΝΕΛΙΚΤΙΚΟΣ ΠΥΡΗΝΑΣ) ΔΙΑΣΤΑΣΕΩΝ 5 X 5 ΣΤΟ
ΕΠΙΠΕΔΟ ΕΙΣΟΔΟΥ. Ο ΠΟΛ/ΣΜΟΣ ΓΙΝΕΤΑΙ ΜΕΤΑΞΥ ΝΕΥΡΩΝΩΝ ΤΟΥ ΕΠΙΠΕΔΟΥ 2 ΜΕ ΒΑΡΗ ΤΟΥ ΕΠΙΠΕΔΟΥ 2

//ΕΔΩ Ο ΥΠΟΛΟΓΙΣΜΟΣ ΓΙΝΕΤΑΙ ΜΕ ΤΟΝ ΠΡΩΤΟ FEATURE MAP ΚΑΙ ΤΟΝ ΑΝΤΙΣΤΟΙΧΟ ΤΟΥ (KERNEL)
ΠΥΡΗΝΑ
                result += Layer2_Neurons_GPU[windowX + 13 * windowY + kernelTemplate2[i]] *
Layer2_Weights_GPU[weightBegin + i * 6];

// ΕΔΩ Ο ΥΠΟΛΟΓΙΣΜΟΣ ΓΙΝΕΤΑΙ ΜΕ ΤΟΝ ΔΕΥΤΕΡΟ FEATURE MAP ,ΑΠΟ ΤΟ PIXEL 169 ΚΑΙ ΜΕΤΑ,
ΒΑΣΙΚΑ ΕΙΝΑΙ ΟΤΙ ΤΟ ΚΑΘΕ PIXEL ΤΟΥ FEATURE MAP ΕΙΝΑΙ ΕΝΑΣ (KERNEL) ΠΥΡΗΝΑΣ 5 X 5, ΑΡΑ
 ΤΟ 169 ΤΟΥ ΔΕΥΤΕΡΟΥ FEATURE MAP ΑΝΤΙΣΤΟΙΧΕΙ ΣΕ ΑΥΤΟΝ ΤΟΝ (KERNEL) ΠΥΡΗΝΑ
                result += Layer2_Neurons_GPU[169 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 1];

// ΕΔΩ Ο ΥΠΟΛΟΓΙΣΜΟΣ ΓΙΝΕΤΑΙ ΜΕ ΤΟΝ ΤΡΙΤΟ FEATURE MAP ,ΑΠΟ ΤΟ PIXEL 338 ΚΑΙ ΜΕΤΑ

                result += Layer2_Neurons_GPU[338 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 2];
                result += Layer2_Neurons_GPU[507 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 3];
                result += Layer2_Neurons_GPU[676 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 4];

// ΕΔΩ Ο ΥΠΟΛΟΓΙΣΜΟΣ ΓΙΝΕΤΑΙ ΜΕ ΤΟΝ ΕΚΤΟ FEATURE MAP ,ΑΠΟ ΤΟ PIXEL 845 ΚΑΙ ΜΕΤΑ,
ΠΑΡΟΛΟ ΠΟΥ Η ΕΙΚΟΝΑ ΣΤΟ ΕΠΙΠΕΔΟ 1 (LAYER1 ) ΕΙΝΑΙ 29 X 29 = 841 PIXELS / NEURONS

                result += Layer2_Neurons_GPU[845 + windowX + 13 * windowY +
kernelTemplate2[i]] * Layer2_Weights_GPU[weightBegin + i * 6 + 5];
                //XOR2 = PIN2 * result;
                printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
//              system("PAUSE");

        }
   // Σιγμοειδής Συνάρτηση ενεργοποίησης f
  // Συναρτηση υπερβολικης εφαπτόμενης με περιορισμούς για α = 1.7159 και β = 0.67

        result = (1.7159*tanhf(0.66666667*result));
        int i;

  //ΥΠΟΛΟΓΙΣΜΟΣ ΔΕΛΤΑ ΣΦΑΛΜΑ 2
        DLTA2B = -((result * (1 - result) * DLTA3B * Layer2_Weights_GPU[weightBegin +
i]));
        printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
```

```
//Η XOR2 ΕΙΝΑΙ ΤΟ PIN2 ΜΕ ΤΙΣ ΤΙΜΕΣ ΤΗΣ XOR2 ΠΟΛ/ΝΟ ΜΕ ΤΟ ΑΠΟΤΕΛΕΣΜΑ ΤΗΣ ΣΥΓΜΟΕΙΔΟΥΣ
ΣΥΝΑΡΤΗΣΗΣ ΥΠΕΡΒΟΛΙΚΗΣ ΕΦΑΠΤΟΜΕΝΗΣ
        XOR2 = PIN2 * result;

        printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
        printf("pinakas_Layer2_Neurons_GPU: %d\n", pinakas_Layer2_Neurons_GPU);
//      system("PAUSE");



        //Layer3_Neurons_GPU[5 * 5 * blockID + pixelY * 5 + pixelX] = result;

        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);



        target = 1; //ΔΗΛΩΣΗ ΤΙΜΗΣ ΣΤΟ ΕΠΙΘΥΜΗΤΟ ΑΠΟΤΕΛΕΣΜΑ - TARGET = 1;

        MSE2 = mean_square_error_Layer2_Neurons_GPU[mse2];

       //ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΣΟΥ ΤΕΤΡΑΓΩΝΙΚΟΥ ΣΦΑΛΜΑΤΟΣ 2
        MSE2 = (1 / 2 * (target - result));
        //DLTA3 = -((target - result) * result * (1 - result));

       //ΥΠΟΛΟΓΙΣΜΟΣ ΝΕΟΥ ΒΑΡΟΥΣ 2

        NL2W = (Layer2_Weights_GPU[weightBegin] - a * DLTA3); //new weight layer2
//      int i;
        //DLTA3B = -((result * (1 - result) * DLTA4B * Layer3_Weights_GPU[weightBegin +
i]));
       //ΥΠΟΛΟΓΙΣΜΟΣ ΝΕΟΥ ΒΑΡΟΥΣ 2

        NL2WB = (Layer2_Weights_GPU[weightBegin] - a * DLTA3B * Layer2_Neurons_GPU[13 * 13
* blockID + pixelY * 13 + pixelX]); //new weight layer2


        //mean_square_error_Layer2_Neurons_GPU[mse2] = 1 / 2 * (target - result);
        //      mean_square_error_Layer2_Neurons_GPU = 1 / 2 * (target - result);
        printf("mean_square_error_Layer2_Neurons_GPU: %d\n",
mean_square_error_Layer2_Neurons_GPU);
        printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);
        printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
        printf(" New_Layer2_Weights_GPU: %d\n", NL2W);
        printf("delta_Layer3_Neurons_GPU: %d\n", delta_Layer3_Neurons_GPU[dlta3]);

        printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
        printf(" New_Layer2_Weights_GPU: %d\n", New_Layer2_Weights_GPU[weightBegin]);
        printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
        printf(" New_Layer2_Weights_GPU: %d\n", NL2WB);
        printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
        printf("nl2wb New_Layer2_Weights_GPU: %d\n", NL2WB);

        Layer3_Neurons_GPU[5 * 5 * blockID + pixelY * 5 + pixelX] = result;


        //XOR2 = PIN2 * result;
        printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
```

```
//	system("PAUSE");



}


__global__ void executeThirdLayer(float *Layer3_Neurons_GPU, float *Layer3_Weights_GPU,
float *Layer4_Neurons_GPU, float *mean_square_error_Layer3_Neurons_GPU, float
*delta_Layer3_Neurons_GPU, float *delta_Layer4_Neurons_GPU, float
*New_Layer3_Weights_GPU, float *Layer4_Weights_GPU, float *delta_Layer5_Neurons_GPU,
float *pinakas_Layer3_Neurons_GPU, float *XOR_Z_Layer3_Neurons_GPU, float
*XOR_Y_Layer2_Neurons_GPU, float *XOR_X_Layer1_Neurons_GPU)
{
	int blockID = blockIdx.x; //ορισμος του block με τιμη απο 0 εως N – 1 , κάθε μπλοκ
αποτελείται απο M νήματα καθένα απο μια τιμή threadIdx.x που έχει έυρος τιμών
 απο 0 έως M -1. Ολα τα μπλοκ σε επίπεδο μπλέγματος είναι οργανωμένα με τη μορφή
μονοδίαστατου πίνακα ,όλα τα νήματα μέσα στο μπλοκ είναι επίσης οργανωμένα με την μορφή
ενός μονοδιάστατου πίνακα . Καθε πλέγμα διαθέτει ένα σύνολο N x M νημάτων.


	int pixelY = threadIdx.y; //νημα y = πιχελ y
	int pixelX = threadIdx.x;
	//float mean_square_error_Layer3_Neurons_GPU;
	float target;

	int XOR2;
	int PIN3;
	int XOR3;
	int XOR1;

	float DLTA4;
	float DLTA5;
	float NL3W = 0; //Αρχικοποιηση
	float DLTA4B;
	float DLTA3B = 0; //Αρχικοποιηση
	float NL3WB = 0; //Αρχικοποιηση
	int a = 0.5; //learning rate//

	   //ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΤΥΠΟΥ blockID

	int dlta4 = blockID * 1251;
	int dlta4b = blockID * 1251;
	int dlta3b = blockID * 1251;
	int dlta5 = blockID * 10;
	//int pinak3 = blockID * 1251;
	int xor3 = blockID * 1251;
	int pinak3;
//	int xor3;

	//int xor2 = blockID * 26 * 6;
	int xor2;
	//int xor1 = blockID * 26;
	int xor1;
	int weightBegin = blockID * 1251;
	int mse3 = blockID * 1251;
	float result = 0;
	float MSE3 = 0;
	result += Layer3_Weights_GPU[weightBegin];
	++xor3;
	//++xor2;
```

```
        //++xor1;
        //++pinak3;
        ++weightBegin;
        ++mse3;
        dlta4++;
        dlta4b++;

        PIN3 = pinakas_Layer3_Neurons_GPU[pinak3]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        XOR3 += XOR_Z_Layer3_Neurons_GPU[xor3]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        XOR2 = XOR_Y_Layer2_Neurons_GPU[xor2]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        XOR1 = XOR_X_Layer1_Neurons_GPU[xor1]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ

        DLTA4 = delta_Layer4_Neurons_GPU[dlta4]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        NL3W = New_Layer3_Weights_GPU[weightBegin]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        DLTA4B = delta_Layer4_Neurons_GPU[dlta4b]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        DLTA3B = delta_Layer3_Neurons_GPU[dlta3b]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        NL3WB = New_Layer3_Weights_GPU[weightBegin]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ
        DLTA5 = delta_Layer5_Neurons_GPU[dlta5]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ

        for (int i = 0; i<1250; ++i)
        {
                result += Layer3_Neurons_GPU[i] * Layer3_Weights_GPU[weightBegin + i];
                //XOR3 = XOR1 + XOR2;
                //printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        //      system("PAUSE");

        }

// Σιγμοειδής Συνάρτηση ενεργοποίησης f
  // Συναρτηση υπερβολικης εφαπτόμενης με περιορισμούς για α = 1.7159 και β = 0.67
        result = (1.7159*tanhf(0.66666667*result));
        XOR3 = XOR1 + XOR2; //ΑΠΟΤΕΛΕΣΜΑ ΑΠΟ ΤΙΣ 2 XOR
        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
//      system("PAUSE");
        int i;

     //ΥΠΟΛΟΓΙΣΜΟΣ ΔΕΛΤΑ 3
        DLTA3B = -((result * (1 - result) * DLTA4B * Layer3_Weights_GPU[weightBegin +
i]));

        printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);

        target = 1;
        MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3]; //ΟΡΙΣΜΟΣ ΠΙΝΑΚΑ

       //ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΣΟΥ ΤΕΤΡΑΓΩΝΙΚΟΥ ΣΦΑΛΜΑΤΟΣ 3
        MSE3 = (1 / 2 * (target - result));
        //Layer4_Neurons_GPU[blockID] = result;

        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

        printf("layer3_weights_GPU: %f\n", Layer3_Weights_GPU);



        //target = 1;
        ////MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3];
        //MSE3 = (1 / 2 * (target - result));
        //DLTA4 = -((target - result) * result * (1 - result));
//ΥΠΟΛΟΓΙΣΜΟΣ ΝΕΟΥ ΒΑΡΟΥΣ 3
```

```
	NL3W = (Layer3_Weights_GPU[weightBegin] - a * DLTA4); //new weight layer3
//	int i;
	//DLTA4B = -((result * (1 - result) * DLTA5 * Layer4_Weights_GPU[weightBegin +
i]));
//ΥΠΟΛΟΓΙΣΜΟΣ ΝΕΟΥ ΒΑΡΟΥΣ 3
	NL3WB = (Layer3_Weights_GPU[weightBegin] - a * DLTA4B * Layer3_Neurons_GPU[5 * 5 *
blockID + pixelY * 5 + pixelX]); //new weight layer3



	printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %d\n", MSE3);
	printf("delta_Layer4_Neurons_GPU: %d\n", DLTA4);
	printf(" New_Layer3_Weights_GPU: %d\n", NL3W);
	printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", DLTA4B);
	printf("nl3wb New_Layer3_Weights_GPU: %d\n", NL3WB);


	Layer4_Neurons_GPU[blockID] = result;
	//XOR3 = XOR1 + XOR2;
	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//	system("PAUSE");

	//printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
	//printf(" New_Layer2_Weights_GPU: %d\n", NL2W);


}

__global__ void executeFourthLayer(float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU,
float *Layer5_Neurons_GPU, float *mean_square_error_Layer4_Neurons_GPU, float
*mean_square_error_Layer6_Neurons_GPU, float *mean_square_error_Layer5_Neurons_GPU, float
*mean_square_error_Layer3_Neurons_GPU, float *mean_square_error_Layer2_Neurons_GPU, float
*mean_square_error_Layer1_Neurons_GPU, float *delta_Layer5_Neurons_GPU, float
*New_Layer4_Weights_GPU, float *delta_Layer3_Neurons_GPU,float *delta_Layer4_Neurons_GPU
,float *New_Layer2_Weights_GPU, float *delta_Layer2_Neurons_GPU, float
*New_Layer1_Weights_GPU, float *Layer3_Neurons_GPU, float *Layer3_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer1_Neurons_GPU, float
*Layer1_Weights_GPU, float *XOR_X_Layer1_Neurons_GPU, float *XOR_Y_Layer2_Neurons_GPU,
float *XOR_Z_Layer3_Neurons_GPU)
{
	int blockID = blockIdx.x; //ορισμος του block με τιμη απο 0 εως N – 1 , κάθε μπλοκ
αποτελείται απο M νήματα καθένα απο μια τιμή threadIdx.x που έχει έυρος τιμών
 απο 0 έως M -1. Ολα τα μπλοκ σε επίπεδο μπλέγματος είναι οργανωμένα με τη μορφή
μονοδίαστατου πίνακα ,όλα τα νήματα μέσα στο μπλοκ είναι επίσης οργανωμένα με την μορφή
ενός μονοδιάστατου πίνακα . Καθε πλέγμα διαθέτει ένα σύνολο N x M νημάτων.


	int pixelY = threadIdx.y; //νημα y = πιχελ y



	float target;

	int XOR2;
	//int PIN3;
	int XOR3;
	int XOR1;
```

```
        //int xor3 = blockID * 1251;
        int xor3;
        //int xor2 = blockID * 26 * 6;
        int xor2;
//      int xor1 = blockID * 26;
        int xor1;



        int weightBegin = blockID * 101;
        int mse4 = blockID * 101;
        int mse5 = blockID * 101;
        int mse6 = blockID * 101;
        int dlta5 = blockID * 101;
        int dlta4b = blockID * 101;

        float MSE4=0; //Αρχικοποιηση
        float result = 0; //Αρχικοποιηση
        float MSE5=0; //Αρχικοποιηση
        float MSE6 = 0; //Αρχικοποιηση
        result += Layer4_Weights_GPU[weightBegin];



        float MSE1;
        float MSE2;
        float MSE3;
        float DLTA5 = 0; //Αρχικοποιηση
        float NL4W = 0; //Αρχικοποιηση
        float DLTA3;
        float DLTA3B;
        float DLTA4B = 0; //Αρχικοποιηση
        float NL2W;
        float NL2WB;

        float DLTA2;
        float DLTA2B;
        float NL1W;
        float NL1WB;

        int a = 0.5; //learning rate//
        //float MSE4 = 0;
        //float MSE5 = 0;
        //int dlta3 = blockID * 26 * 6;
        int dlta3b = blockID * 26 * 6;

        int dlta2b = blockID * 26;
        //int dlta2 = blockID * 26;
        //int mse1 = blockID * 26;
        int mse1;
        //int mse2 = blockID * 26 * 6;
        int mse2;

        //int mse3 = blockID * 1251;
        int mse3;
        //int mse4 = blockID * 101;
        //int mse5 = blockID * 10;
        XOR3 = XOR_Z_Layer3_Neurons_GPU[xor3];
        XOR2 = XOR_Y_Layer2_Neurons_GPU[xor2];
        XOR1 = XOR_X_Layer1_Neurons_GPU[xor1];
```

```
        DLTA4B = delta_Layer4_Neurons_GPU[dlta4b];


        DLTA5 = delta_Layer5_Neurons_GPU[dlta5];
        NL4W = New_Layer4_Weights_GPU[weightBegin];
        //DLTA3 = delta_Layer3_Neurons_GPU[dlta3];
        DLTA3B = delta_Layer3_Neurons_GPU[dlta3b];

        NL2W = New_Layer2_Weights_GPU[weightBegin];
        NL2WB = New_Layer2_Weights_GPU[weightBegin];

        //DLTA2 = delta_Layer2_Neurons_GPU[dlta2];
        DLTA2B = delta_Layer2_Neurons_GPU[dlta2b];
        //NL1W = New_Layer1_Weights_GPU[weightBegin];
        NL1WB = New_Layer1_Weights_GPU[weightBegin];

        ++weightBegin;
        ++mse4;
        //++xor3;
        ////++xor2;
        //++xor1;
//      ++mse3;
//      ++mse2;
//      ++mse1;
        dlta5++;
        ++mse5;
        ++mse6;
        for (int i = 0; i<100; ++i)
        {
                result += Layer4_Neurons_GPU[i] * Layer4_Weights_GPU[weightBegin + i];
                //printf("apotelesma result layer4_Neurons_GPU: %f\n", result);
                XOR3 = XOR1 + XOR2;
                printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
//              system("PAUSE");
        }
        //printf("apotelesma result layer4_Neurons_GPU: %f\n", result);

// Σιγμοειδής Συνάρτηση ενεργοποίησης f
  // Συναρτηση υπερβολικης εφαπτόμενης με περιορισμούς για α = 1.7159 και β = 0.67

        result = (1.7159*tanhf(0.66666667*result));
        XOR3 = XOR1 + XOR2;
        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        //system("PAUSE");

        //printf("apotelesma uperboliko efaptomeni  layer4_Neurons_GPU: %f\n", result);
        //Layer5_Neurons_GPU[blockID] = result;

        XOR3 = XOR1 + XOR2; //ΑΘΡΟΙΣΤΙΚΗ XOR3

        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
        printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//      system("PAUSE");

        printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

        printf("layer5_Neurons_GPU: %f\n", Layer5_Neurons_GPU);

        printf("layer4_weights_GPU: %f\n", Layer4_Weights_GPU);
```

```
        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer4_Neurons_GPU: %f\n", Layer4_Neurons_GPU);

        printf("layer3_weights_GPU: %f\n", Layer3_Weights_GPU);

        printf("layer3_Neurons_GPU: %f\n", Layer3_Neurons_GPU);

        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

        printf("layer1_Neurons_GPU: %f\n", Layer1_Neurons_GPU);

        printf("layer2_Neurons_GPU: %f\n", Layer2_Neurons_GPU);

        printf("layer2_weights_GPU: %f\n", Layer2_Weights_GPU);

        printf("layer1_weights_GPU: %f\n", Layer1_Weights_GPU);

        printf("layer5_Neurons_GPU: %f\n", Layer5_Neurons_GPU);


        target = 1;

        MSE4 = mean_square_error_Layer4_Neurons_GPU[mse4];
        MSE3 = mean_square_error_Layer3_Neurons_GPU[mse3];
        MSE2 = mean_square_error_Layer2_Neurons_GPU[mse2];
        MSE1 = mean_square_error_Layer1_Neurons_GPU[mse1];
        MSE5 = mean_square_error_Layer5_Neurons_GPU[mse5];
        MSE6 = mean_square_error_Layer6_Neurons_GPU[mse6];

   //ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΣΟΥ ΤΕΤΡΑΓΩΝΙΚΟΥ ΣΦΑΛΜΑΤΟΣ 4
        MSE4 = (1 / 2 * (target - result));
int i;
//ΥΠΟΛΟΓΙΣΜΟΣ ΔΕΛΤΑ ΣΦΑΛΜΑ 5
DLTA5 = -((target - result) * result * (1 - result));
////ΥΠΟΛΟΓΙΣΜΟΣ ΔΕΛΤΑ 4
        DLTA4B = -((result * (1 - result) * DLTA5 * Layer4_Weights_GPU[weightBegin + i]));
 NL4W = (Layer4_Weights_GPU[weightBegin] - a * DLTA5 * Layer4_Neurons_GPU[blockID]);
////new weight layer4

//      printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", DLTA4B);
//      printf("nl4w New_Layer4_Weights_GPU: %d\n", NL4W);
        printf("dlta5 delta_Layer5_Neurons_GPU: %d\n", DLTA5);
        printf("mean_square_error_Layer4_Neurons_GPU: %f\n",
mean_square_error_Layer4_Neurons_GPU);
        printf("mean_square_error_Layer4_Neurons_GPU: %d\n", MSE4);
        printf("mean_square_error_Layer1_Neurons_GPU: %d\n", MSE1);
        printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);

        Layer5_Neurons_GPU[blockID] = result;

        target = 1;
  //ΥΠΟΛΟΓΙΣΜΟΣ ΜΕΣΟΥ ΤΕΤΡΑΓΩΝΙΚΟΥ ΣΦΑΛΜΑΤΟΣ 5
        MSE5 = (1 / 2 * (target - result));


//NL4W = (Layer4_Weights_GPU[weightBegin] - a * DLTA5 * Layer4_Neurons_GPU[blockID]);
//new weight layer4

printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", DLTA4B);
```

```
        printf("nl4w New_Layer4_Weights_GPU: %d\n", NL4W);



        MSE6 = MSE5 + MSE4 + MSE3 + MSE2 + MSE1;
        mean_square_error_Layer6_Neurons_GPU[mse6]=
mean_square_error_Layer5_Neurons_GPU[mse5] + mean_square_error_Layer4_Neurons_GPU[mse4] +
mean_square_error_Layer3_Neurons_GPU[mse3] + mean_square_error_Layer2_Neurons_GPU[mse2] +
mean_square_error_Layer1_Neurons_GPU[mse1];
              printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU);
        printf("mean_square_error_Layer5_Neurons_GPU: %f\n",
mean_square_error_Layer5_Neurons_GPU[mse5]);
        //MSE5 = mean_square_error_Layer5_Neurons_GPU[mse5];
//      MSE5 += MSE4 + MSE3 + MSE2 + MSE1;
        printf("mean_square_error_Layer2_Neurons_GPU: %f\n",
mean_square_error_Layer2_Neurons_GPU);
        printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
        printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
        printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU[mse6]);

        printf("mean_square_error_Layer5_Neurons_GPU: %d\n", MSE5);
        //printf("mean_square_error_Layer4_Neurons_GPU: %d\n", MSE4);
        //printf("mean_square_error_Layer1_Neurons_GPU: %d\n", MSE1);
        //printf("mean_square_error_Layer2_Neurons_GPU: %d\n", MSE2);
        printf("dlta5 delta_Layer5_Neurons_GPU: %d\n", DLTA5);
        //printf("nl4w New_Layer4_Weights_GPU: %d\n", NL4W);

        printf("delta_Layer3_Neurons_GPU: %d\n", DLTA3);
        printf(" New_Layer2_Weights_GPU: %d\n", NL2W);

        printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", DLTA3B);
        printf("nl2wb New_Layer2_Weights_GPU: %d\n", NL2WB);


        printf("delta_Layer2_Neurons_GPU: %d\n", DLTA2);
        printf(" New_Layer1_Weights_GPU: %d\n", NL1W);
        printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", DLTA2B);
        printf("nl1wb New_Layer1_Weights_GPU: %d\n", NL1WB);

        XOR3 = XOR1 + XOR2; //ΑΘΡΟΙΣΤΙΚΗ XOR3
        printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
        printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
        printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);
//      system("PAUSE");


}




int main(int argc, char** argv)
{     //ΚΥΡΙΩΣ ΠΡΟΓΡΑΜΜΑ
       //readIn(float *Layer1);
```

```
void readIn(float *Layer1_Neurons_CPU);
	void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU);
	//InitHostMem();
	NeuralNetwork();
void readIn(float *layer1_Neurons_CPU);

	printf("test:");
}


void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU)
{
	// initial layer 1 weight
	FILE *fp1;
//ΑΝΟΙΓΜΑ ΑΡΧΕΙΟΥ ΓΙΑ ΔΙΑΒΑΣΜΑ ΒΑΡΟΥΣ ΓΙΑ ΤΟ ΕΠΙΠΕΔΟ 1 (Layer 1)
	fp1 = fopen("lw1.bin", "rb");
	if (fp1 != NULL)
	{
		for (int i = 0; i < 156; ++i)
			fread(&(Layer1_Weights_CPU[i]), sizeof(float), 156, fp1);
		//		fread(Layer1_Weights_CPU, sizeof(float), 1, pFile1);
		fclose(fp1);
	}
	//	for (int i = 0; i < 156; ++i)

	//	printf("layer1_WEIGHTS_CPU: %f\n", (Layer1_Weights_CPU[i]));
	//	system("PAUSE");


	// initial layer 2 weight
	FILE *fp2;
//ΑΝΟΙΓΜΑ ΑΡΧΕΙΟΥ ΓΙΑ ΔΙΑΒΑΣΜΑ ΒΑΡΟΥΣ ΓΙΑ ΤΟ ΕΠΙΠΕΔΟ 2 (Layer 2)
	fp2 = fopen("lw2.bin", "rb");
	if (fp2 != NULL)
	{
		//for (int i = 0; i<7800; ++i)
		fread(Layer2_Weights_CPU, sizeof(float), 7800, fp2);
		fclose(fp2);

		for (int i = 0; i < 7800; ++i)
			printf("layer2_WEIGHTS_CPU  arxh hhhhhhhhhhhhhhhhhhh: %f\n",
(Layer2_Weights_CPU[i]));
		system("PAUSE");
	}

	// initial layer 3 weight
	FILE *fp3;
//ΑΝΟΙΓΜΑ ΑΡΧΕΙΟΥ ΓΙΑ ΔΙΑΒΑΣΜΑ ΒΑΡΟΥΣ ΓΙΑ ΤΟ ΕΠΙΠΕΔΟ 3 (Layer 3)
	fp3 = fopen("lw3.bin", "rb");
	if (fp3 != NULL)
	{
		for (int i = 0; i < 125100; ++i)
			fread(&(Layer3_Weights_CPU[i]), sizeof(float), 125100, fp3);
		fclose(fp3);
	}
	// initial layer 4 weight
	FILE *fp4;
//ΑΝΟΙΓΜΑ ΑΡΧΕΙΟΥ ΓΙΑ ΔΙΑΒΑΣΜΑ ΒΑΡΟΥΣ ΓΙΑ ΤΟ ΕΠΙΠΕΔΟ 4 (Layer 4)

	fp4 = fopen("lw4.bin", "rb");
```

```
	if (fp4 != NULL)
	{
		//	for (int i = 0; i < 1010; ++i)
		fread(Layer4_Weights_CPU, sizeof(float), 1010, fp4);
		fclose(fp4);
	}
}

void readIn(float *layer1_Neurons_CPU)
{
	FILE *fp6 = 0;
	fp6 = fopen("neuron.bin", "rb");
	if (fp6 != NULL)
	{
		//for (int i = 0; i < 841; ++i)

		//	printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou
arxeiou: %f\n", (Layer1_Neurons_CPU[i]));

		fread(&Layer1_Neurons_CPU, sizeof(int), 29 * 29, fp6);

		fread(&neuron, sizeof(int), 29 * 29, fp6);


	float* Layer_neurons_CPU = (float*)  malloc(neuron * sizeof(float) * 29 * 29);

		//fclose(fp6);

		for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer_neurons_CPU[i]));

		for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

		free(malloc);

		fclose(fp6);

		for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

	}

for (int i = 0; i < 841; ++i)

printf("layer1_Neurons_CPU apotelesma meta APO to diabasma tou arxeiou: %f\n",
(Layer1_Neurons_CPU[i]));

}




void output(double *Layer5_Neurons_CPU)
{
	FILE *fp5 = 0;
//εξοδος νευρωνα
```

```
	fp5 = fopen("neuronout.bin", "wb");
	if (fp5)
	{
		fwrite(Layer5_Neurons_CPU, sizeof(double), 10, fp5);
		fclose(fp5);
for (int i = 0; i < 10; ++i)

printf("layer5_Neurons_CPU apotelesma: %f\n", (Layer5_Neurons_CPU[i]));


	}
}




void NeuralNetwork()
{
	 //CUT_DEVICE_INIT();
     //EIKONA 29 X 29 PIXELS/NEURONS
	 float Layer1_Neurons_CPU[29 * 29] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
```

```
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
```

```
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1,
```

```
//ΒΑΡΗ ΓΙΑ ΤΟ ΕΠΙΠΕΔΟ 2

float Layer2_Weights_CPU[7800] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
            1, 1, 1, 1,
```

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

```
		1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
		1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1,
```

```
for (int i = 0; i < 780; ++i)
		printf("layer2_weights_CPU: %f\n", (Layer2_Weights_CPU[i]));
	system("PAUSE");

	float pinakas_Layer1_Neurons_CPU[841] = { 0, 0, 1, 1 };
	for (int i = 0; i<10; ++i)
		printf("pinakas_layer1_Neurons_CPU: %f\n",
(pinakas_Layer1_Neurons_CPU[i]));
	system("PAUSE");



	float pinakas_Layer2_Neurons_CPU[1014] = { 0, 1, 0, 1 };
	for (int i = 0; i<10; ++i)
		printf("pinakas_layer2_Neurons_CPU: %f\n",
(pinakas_Layer2_Neurons_CPU[i]));
	system("PAUSE");
```

```
	//float Layer1_Neurons_CPU[29 * 29];

	//void readIn(float *Layer1_Neurons_CPU);
	//void readln(layer1_Neurons_CPU);

	void readln(float *Layer1_Neurons_CPU);
//ΔΗΛΩΣΕΙΣ – ΟΡΙΣΜΑΤΑ - POINTERS
	float *Layer1_Neurons_GPU;
	//     float Layer1_Weights_CPU[156];
	float *Layer1_Weights_GPU;

	float *mean_square_error_Layer1_Neurons_CPU[841];
	float *mean_square_error_Layer1_Neurons_GPU;

	//     float Layer2_Weights_CPU[7800];
	float *Layer2_Weights_GPU;
	float *Layer2_Neurons_GPU;
	float *mean_square_error_Layer2_Neurons_GPU;
	float *mean_square_error_Layer2_Neurons_CPU[1014];

	//     float Layer3_Weights_CPU[125100];
	float *Layer3_Weights_GPU;
	float *Layer3_Neurons_GPU;

	float *mean_square_error_Layer3_Neurons_CPU[1250];

	float *mean_square_error_Layer4_Neurons_CPU[100];

	float *mean_square_error_Layer5_Neurons_CPU[10];
	float *mean_square_error_Layer6_Neurons_CPU[10];

	float *mean_square_error_Layer3_Neurons_GPU;



	//float Layer4_Weights_CPU[1010];

	float *New_Layer4_Weights_GPU;
	float *New_Layer3_Weights_GPU;
	float *New_Layer2_Weights_GPU;
	float *New_Layer1_Weights_GPU;

	float *Layer4_Neurons_GPU;

	float *mean_square_error_Layer4_Neurons_GPU;

	float Layer5_Neurons_CPU[10] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
	float *Layer5_Neurons_GPU;

	float *mean_square_error_Layer5_Neurons_GPU;
	float *mean_square_error_Layer6_Neurons_GPU;

	float *delta_Layer5_Neurons_GPU;
	float *delta_Layer5_Neurons_CPU[10];

	float *delta_Layer4_Neurons_GPU;
	float *delta_Layer4_Neurons_CPU[100];

	float *delta_Layer3_Neurons_GPU;
	float *delta_Layer3_Neurons_CPU[1250];
```

```
	float *delta_Layer2_Neurons_GPU;
	float *delta_Layer2_Neurons_CPU[1014];

	float *delta_Layer1_Neurons_GPU;
	float *delta_Layer1_Neurons_CPU[841];

//	float *pinakas_Layer1_Neurons_CPU[841];

//	float *pinakas_Layer2_Neurons_CPU[1014];

	float *pinakas_Layer3_Neurons_CPU[1250];

	float *XOR_X_Layer1_Neurons_CPU[841];

	float *XOR_Y_Layer2_Neurons_CPU[1014];

	float *XOR_Z_Layer3_Neurons_CPU[1250];

	float *pinakas_Layer1_Neurons_GPU;

	float *pinakas_Layer2_Neurons_GPU;

	float *pinakas_Layer3_Neurons_GPU;

	float *XOR_X_Layer1_Neurons_GPU;

	float *XOR_Y_Layer2_Neurons_GPU;

	float *XOR_Z_Layer3_Neurons_GPU;







	double *outputLayer;

	void InitHostMem(float *Layer1_Weights_CPU, float *Layer2_Weights_CPU, float
*Layer3_Weights_CPU, float *Layer4_Weights_CPU);
	InitHostMem;


	//FILE *pFile3 = 0;
	//pFile3 = fopen("lw3.wei", "rb");
	//if (pFile3 != NULL)
	//{
	//	for (int i = 0; i < 125100; ++i)
	//		fread(Layer3_Weights_CPU, sizeof(float), 125100, pFile3);
	//	fclose(pFile3);
	for (int i = 0; i < 125; ++i)
		printf("layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer3_Weights_CPU[i]));
	system("PAUSE");


	//}
	// initial layer 4 weight
	//	FILE *pFile4;
	//	pFile4 = fopen("lw4.wei", "rb");
	//	if (pFile4 != NULL)
	//	{
```

```
	//	for (int i = 0; i < 1010; ++i)
	//	fread(Layer4_Weights_CPU, sizeof(float), 1010, pFile4);
	//	fclose(pFile4);
	for (int i = 0; i < 1010; ++i)
		printf("layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer4_Weights_CPU[i]));
	system("PAUSE");


	//}

	//	FILE *pFile2;
	////	pFile2 = fopen("lw2.wei", "rb");
	//	if (pFile2 != NULL)
	//	{
	//	for (int i = 0; i < 7800; ++i)
	//	fread(Layer2_Weights_CPU, sizeof(float), 7800, pFile2);
	//	fclose(pFile2);

	for (int i = 0; i < 78; ++i)
		printf("layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: %f\n",
(Layer2_Weights_CPU[i]));
	system("PAUSE");
	//}


	//allocate momory on Device
	//	void InitGPUMem(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer3_Neurons_GPU, float
*Layer3_Weights_GPU, float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU, float
*Layer5_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*mean_square_error_Layer2_Neurons_GPU, float *mean_square_error_Layer3_Neurons_GPU, float
*mean_square_error_Layer4_Neurons_GPU, float *mean_square_error_Layer5_Neurons_GPU, float
*delta_Layer5_Neurons_GPU, float *New_Layer4_Weights_GPU, float
*delta_Layer4_Neurons_GPU, float *New_Layer3_Weights_GPU, float
*delta_Layer3_Neurons_GPU, float *New_Layer2_Weights_GPU, float
*delta_Layer2_Neurons_GPU, float *New_Layer1_Weights_GPU);
	//void InitGPUMem(float *Layer1_Neurons_GPU, float *Layer1_Weights_GPU, float
*Layer2_Neurons_GPU, float *Layer2_Weights_GPU, float *Layer3_Neurons_GPU, float
*Layer3_Weights_GPU, float *Layer4_Neurons_GPU, float *Layer4_Weights_GPU, float
*Layer5_Neurons_GPU, float *mean_square_error_Layer1_Neurons_GPU, float
*mean_square_error_Layer2_Neurons_GPU);


//ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ ΓΙΑ CUDA ΣΤΗΝ GPU ΓΙΑ ΤΙΣ ΠΑΡΑΜΕΤΡΟΥΣ – ΜΕΤΑΒΛΗΤΕΣ ΠΟΥ ΘΑ
ΧΡΗΣΙΜΟΠΟΙΗΘΟΥΝΕ

	cudaMalloc((void**)&Layer1_Neurons_GPU, sizeof(float) * 29 * 29);
	cudaMalloc((void**)&Layer1_Weights_GPU, sizeof(float) * 156);
	cudaMalloc((void**)&mean_square_error_Layer1_Neurons_GPU, sizeof(float) * 29 *
29);
	cudaMalloc((void**)&pinakas_Layer1_Neurons_GPU, sizeof(float) * 29 * 29);
	cudaMalloc((void**)&XOR_X_Layer1_Neurons_GPU, sizeof(float) * 29 * 29);

	cudaMalloc((void**)&Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
	cudaMalloc((void**)&Layer2_Weights_GPU, sizeof(float) * 7800);
	cudaMalloc((void**)&mean_square_error_Layer2_Neurons_GPU, sizeof(float) * 13 * 13
* 6);

	cudaMalloc((void**)&pinakas_Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
	cudaMalloc((void**)&XOR_Y_Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6);
```

```
	cudaMalloc((void**)&Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&Layer3_Weights_GPU, sizeof(float) * 125100);

	cudaMalloc((void**)&mean_square_error_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&pinakas_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&XOR_Z_Layer3_Neurons_GPU, sizeof(float) * 1250);

	cudaMalloc((void**)&Layer4_Neurons_GPU, sizeof(float) * 100);
	cudaMalloc((void**)&Layer4_Weights_GPU, sizeof(float) * 1010);

	cudaMalloc((void**)&New_Layer4_Weights_GPU, sizeof(float) * 1010);
	cudaMalloc((void**)&New_Layer3_Weights_GPU, sizeof(float) * 125100);
	cudaMalloc((void**)&New_Layer2_Weights_GPU, sizeof(float) * 7800);
	cudaMalloc((void**)&New_Layer1_Weights_GPU, sizeof(float) * 156);

	cudaMalloc((void**)&mean_square_error_Layer4_Neurons_GPU, sizeof(float) * 100);

	cudaMalloc((void**)&Layer5_Neurons_GPU, sizeof(float) * 10);

	cudaMalloc((void**)&delta_Layer5_Neurons_GPU, sizeof(float) * 10);
	cudaMalloc((void**)&delta_Layer4_Neurons_GPU, sizeof(float) * 100);
	cudaMalloc((void**)&delta_Layer3_Neurons_GPU, sizeof(float) * 1250);
	cudaMalloc((void**)&delta_Layer2_Neurons_GPU, sizeof(float) * 1014);
	cudaMalloc((void**)&delta_Layer1_Neurons_GPU, sizeof(float) * 841);


	cudaMalloc((void**)&mean_square_error_Layer5_Neurons_GPU, sizeof(float) * 10);
	cudaMalloc((void**)&mean_square_error_Layer6_Neurons_GPU, sizeof(float) * 10);

	outputLayer = (double*)malloc(sizeof(double) * 10);

	cudaEvent_t Start, Stop;
	cudaEventCreate(&Start);
	cudaEventCreate(&Stop);









	//copy from CPU to GPU

 //ΑΝΤΙΓΡΑΦΗ ΑΠΟ CPU (HOST)  ΣΤΗΝ GPU (DEVICE)


	cudaMemcpy(Layer1_Neurons_GPU, Layer1_Neurons_CPU, sizeof(float) * 29 * 29,
cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer1_Neurons_GPU,
mean_square_error_Layer1_Neurons_CPU, sizeof(float) * 29 * 29, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer2_Neurons_GPU,
mean_square_error_Layer2_Neurons_CPU, sizeof(float) * 13 * 13 * 6,
cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer3_Neurons_GPU,
mean_square_error_Layer3_Neurons_CPU, sizeof(float) * 1250, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer4_Neurons_GPU,
mean_square_error_Layer4_Neurons_CPU, sizeof(float) * 100, cudaMemcpyHostToDevice);
```

```
	cudaMemcpy(mean_square_error_Layer5_Neurons_GPU,
mean_square_error_Layer5_Neurons_CPU, sizeof(float) * 10, cudaMemcpyHostToDevice);

	cudaMemcpy(mean_square_error_Layer6_Neurons_GPU,
mean_square_error_Layer6_Neurons_CPU, sizeof(float) * 10, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer1_Neurons_GPU, pinakas_Layer1_Neurons_CPU, sizeof(float) *
29 * 29, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer2_Neurons_GPU, pinakas_Layer2_Neurons_CPU, sizeof(float) *
13 * 13 * 6, cudaMemcpyHostToDevice);

	cudaMemcpy(pinakas_Layer3_Neurons_GPU, pinakas_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_X_Layer1_Neurons_GPU, XOR_X_Layer1_Neurons_CPU, sizeof(float) * 29
* 29, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_Y_Layer2_Neurons_GPU, XOR_Y_Layer2_Neurons_CPU, sizeof(float) * 13
* 13 * 6, cudaMemcpyHostToDevice);

	cudaMemcpy(XOR_Z_Layer3_Neurons_GPU, XOR_Z_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);




	cudaMemcpy(Layer1_Weights_GPU, Layer1_Weights_CPU, sizeof(float) * 156,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer2_Weights_GPU, Layer2_Weights_CPU, sizeof(float) * 7800,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer3_Weights_GPU, Layer3_Weights_CPU, sizeof(float) * 125100,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer4_Weights_GPU, Layer4_Weights_CPU, sizeof(float) * 1010,
cudaMemcpyHostToDevice);
	cudaMemcpy(Layer5_Neurons_GPU, Layer5_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);

	cudaMemcpy(delta_Layer5_Neurons_GPU, delta_Layer5_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer4_Neurons_GPU, delta_Layer4_Neurons_CPU, sizeof(float) *
100, cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer3_Neurons_GPU, delta_Layer3_Neurons_CPU, sizeof(float) *
1250, cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer2_Neurons_GPU, delta_Layer2_Neurons_CPU, sizeof(float) * 10,
cudaMemcpyHostToDevice);
	cudaMemcpy(delta_Layer1_Neurons_GPU, delta_Layer1_Neurons_CPU, sizeof(float) *
841, cudaMemcpyHostToDevice);


	cudaMemcpy(New_Layer4_Weights_GPU, New_Layer4_Weights_CPU, sizeof(float) * 1010,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer3_Weights_GPU, New_Layer3_Weights_CPU, sizeof(float) * 125100,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer2_Weights_GPU, New_Layer2_Weights_CPU, sizeof(float) * 7800,
cudaMemcpyHostToDevice);
	cudaMemcpy(New_Layer1_Weights_GPU, New_Layer1_Weights_CPU, sizeof(float) * 156,
cudaMemcpyHostToDevice);
```

```
	cudaEventRecord(Start);

// INITIALIZE THE GRID AND BLOCK DIMENSIONS HERE
	dim3 Layer1_Block(6, 1);
	dim3 Layer1_Thread(13, 13);

//LAUNCH THE GPU KERNEL HERE
	executeFirstLayer << <Layer1_Block, Layer1_Thread >> >(Layer1_Neurons_GPU,
Layer1_Weights_GPU, Layer2_Neurons_GPU, mean_square_error_Layer1_Neurons_GPU,
delta_Layer1_Neurons_GPU, delta_Layer2_Neurons_GPU, New_Layer1_Weights_GPU,
delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU, pinakas_Layer1_Neurons_GPU,
XOR_X_Layer1_Neurons_GPU);

// INITIALIZE THE GRID AND BLOCK DIMENSIONS HERE

	dim3 DimLayer2_Block(50, 1);
	dim3 DimLayer2_Thread(5, 5);

//LAUNCH THE GPU KERNEL HERE
	executeSecondLayer << <DimLayer2_Block, DimLayer2_Thread >> >(Layer2_Neurons_GPU,
Layer2_Weights_GPU, Layer3_Neurons_GPU, mean_square_error_Layer2_Neurons_GPU,
delta_Layer2_Neurons_GPU, delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU,
delta_Layer4_Neurons_GPU, New_Layer3_Weights_GPU, pinakas_Layer2_Neurons_GPU,
XOR_Y_Layer2_Neurons_GPU);

// INITIALIZE THE GRID AND BLOCK DIMENSIONS HERE

	dim3 DimLayer3_Block(100, 1);
	dim3 DimLayer3_Thread(1, 1);

//LAUNCH THE GPU KERNEL HERE
	executeThirdLayer << <DimLayer3_Block, DimLayer3_Thread >> >(Layer3_Neurons_GPU,
Layer3_Weights_GPU, Layer4_Neurons_GPU, mean_square_error_Layer3_Neurons_GPU,
delta_Layer3_Neurons_GPU, delta_Layer4_Neurons_GPU, New_Layer3_Weights_GPU,
Layer4_Weights_GPU, delta_Layer5_Neurons_GPU, pinakas_Layer3_Neurons_GPU,
XOR_Z_Layer3_Neurons_GPU, XOR_Y_Layer2_Neurons_GPU, XOR_X_Layer1_Neurons_GPU);

// INITIALIZE THE GRID AND BLOCK DIMENSIONS HERE
	dim3 DimLayer4_Block(10, 1);
	dim3 DimLayer4_Thread(1, 1);

//LAUNCH THE GPU KERNEL HERE
	executeFourthLayer << <DimLayer4_Block, DimLayer4_Thread >> >(Layer4_Neurons_GPU,
Layer4_Weights_GPU, Layer5_Neurons_GPU, mean_square_error_Layer1_Neurons_GPU,
mean_square_error_Layer2_Neurons_GPU, mean_square_error_Layer3_Neurons_GPU,
mean_square_error_Layer4_Neurons_GPU, mean_square_error_Layer5_Neurons_GPU,
mean_square_error_Layer6_Neurons_GPU, delta_Layer4_Neurons_GPU, delta_Layer5_Neurons_GPU,
New_Layer4_Weights_GPU, delta_Layer3_Neurons_GPU, New_Layer2_Weights_GPU,
delta_Layer2_Neurons_GPU, New_Layer1_Weights_GPU, Layer3_Neurons_GPU, Layer3_Weights_GPU,
Layer2_Neurons_GPU, Layer2_Weights_GPU, Layer1_Neurons_GPU, Layer1_Weights_GPU,
XOR_Z_Layer3_Neurons_GPU, XOR_Y_Layer2_Neurons_GPU, XOR_X_Layer1_Neurons_GPU);



	cudaThreadSynchronize();
	cudaDeviceSynchronize();
	outputLayer = (double*)malloc(sizeof(double) * 10);

	//copy from GPU to CPU
//ΑΝΤΙΓΡΑΦΗ ΑΠΟ ΤΗΝ GPU (DEVICE) ΣΤΗΝ CPU (HOST)
```

```
	cudaMemcpy(Layer5_Neurons_CPU, Layer5_Neurons_GPU, sizeof(float) * 10,
cudaMemcpyDeviceToHost);



	cudaEventSynchronize(Stop);
	float milliseconds = 0;

	cudaEventElapsedTime(&milliseconds, Start, Stop);

	for (int i = 0; i < 10; ++i)
		printf("output layer [%d]=%d\n", outputLayer[i]);
	printf("xronos: deyterolepta %d\n start %d\n stop %d\n", milliseconds, Start,
Stop);




	for (int a = 0; a<10; a++)
	{
		outputLayer[a] = (double)Layer5_Neurons_CPU[a];
	}
	output(outputLayer);

	//float resulta;
	//printf("resulta apotelesma: %f\n", resulta);

	float Layer4_Neurons_CPU[100];

//ΑΝΤΙΓΡΑΦΗ ΑΠΟ ΤΗΝ GPU (DEVICE) ΣΤΗΝ CPU (HOST)

	cudaMemcpy(Layer4_Neurons_CPU, Layer4_Neurons_GPU, sizeof(float) * 100,
cudaMemcpyDeviceToHost);

//ΕΓΓΡΑΦΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΑΡΧΕΙΟ

	FILE *fp44 = fopen("layer_4.bin", "wb");
	//	for (int i = 0; i < 100; ++i)
	fwrite(Layer4_Neurons_CPU, sizeof(float), 100, fp44);
	fclose(fp44);
	for (int i = 0; i<100; ++i)
		printf("layer4_Neurons_CPU NEOI NEYRONES apotelesma: %f\n",
(Layer4_Neurons_CPU[i]));
	system("PAUSE");


	//float delta_Layer4_Neurons_CPU[100];
	cudaMemcpy(delta_Layer4_Neurons_CPU, delta_Layer4_Neurons_GPU, sizeof(float) *
100, cudaMemcpyDeviceToHost);


//ΕΓΓΡΑΦΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΑΡΧΕΙΟ
	FILE *fp4d = fopen("layer_4d.bin", "wb");
	//for (int i = 0; i < 100; ++i)
	fwrite(delta_Layer4_Neurons_CPU, sizeof(float), 100, fp4d);
	fclose(fp4d);



	float Layer3_Neurons_CPU[50 * 5 * 5];
```

```
//ΑΝΤΙΓΡΑΦΗ ΑΠΟ ΤΗΝ GPU (DEVICE) ΣΤΗΝ CPU (HOST)

	cudaMemcpy(Layer3_Neurons_CPU, Layer3_Neurons_GPU, sizeof(float) * 50 * 5 * 5,
cudaMemcpyDeviceToHost);

	FILE *fp33;
//ΕΓΓΡΑΦΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΑΡΧΕΙΟ
	fp33 = fopen("layer_3.bin", "wb");
	fwrite(Layer3_Neurons_CPU, sizeof(float), 50 * 5 * 5, fp33);
	fclose(fp33);
	for (int i = 0; i<1250; ++i)
		printf("layer3_Neurons_CPU  NEOI NEYRONES apotelesma: %f\n",
(Layer3_Neurons_CPU[i]));
	system("PAUSE");


	float Layer2_Neurons_CPU[13 * 13 * 6];

//ΑΝΤΙΓΡΑΦΗ ΑΠΟ ΤΗΝ GPU (DEVICE) ΣΤΗΝ CPU (HOST)

	cudaMemcpy(Layer2_Neurons_CPU, Layer2_Neurons_GPU, sizeof(float) * 13 * 13 * 6,
cudaMemcpyDeviceToHost);

//ΕΓΓΡΑΦΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΑΡΧΕΙΟ
	FILE *fp22 = fopen("layer_2.bin", "wb");
	//	for (int i = 0; i < 1014; ++i)
	fwrite(Layer2_Neurons_CPU, sizeof(float), 13 * 13 * 6, fp22);
	fclose(fp22);
	for (int i = 0; i<1014; ++i)
		printf("layer2_Neurons_CPU NEOI NEYRONES apotelesma: %f\n",
(Layer2_Neurons_CPU[i]));
	system("PAUSE");


	//float Layer1_Neurons_CPU[29 * 29];

//ΑΝΤΙΓΡΑΦΗ ΑΠΟ ΤΗΝ GPU (DEVICE) ΣΤΗΝ CPU (HOST)

	cudaMemcpy(Layer1_Neurons_CPU, Layer1_Neurons_GPU, sizeof(float) * 29 * 29,
cudaMemcpyDeviceToHost);

//ΕΓΓΡΑΦΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΣΕ ΑΡΧΕΙΟ
	FILE *fp11;
	fp11 = fopen("layer_1.bin", "wb");
	//for (int i = 0; i < 841; ++i)
	fwrite(Layer1_Neurons_CPU, sizeof(float), 29 * 29, fp11);
	fclose(fp11);
	for (int i = 0; i<841; ++i)
		printf("layer1_Neurons_CPU NEOI NEYRONES apotelesma: %f\n",
(Layer1_Neurons_CPU[i]));

	system("PAUSE");
//ΕΚΤΥΠΩΣΗ
	printf("mean_square_error_Layer2_Neurons_GPU: %f\n",
mean_square_error_Layer2_Neurons_GPU);
	printf("mean_square_error_Layer1_Neurons_GPU: %f\n",
mean_square_error_Layer1_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %f\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer4_Neurons_GPU: %f\n",
mean_square_error_Layer4_Neurons_GPU);
```

```
	printf("mean_square_error_Layer5_Neurons_GPU: %f\n",
mean_square_error_Layer5_Neurons_GPU);
	printf("mean_square_error_Layer6_Neurons_GPU: %f\n",
mean_square_error_Layer6_Neurons_GPU);
	system("PAUSE");

	printf("mean_square_error_Layer2_Neurons_GPU: %d\n",
mean_square_error_Layer2_Neurons_GPU);
	printf("mean_square_error_Layer1_Neurons_GPU: %d\n",
mean_square_error_Layer1_Neurons_GPU);
	printf("mean_square_error_Layer3_Neurons_GPU: %d\n",
mean_square_error_Layer3_Neurons_GPU);
	printf("mean_square_error_Layer4_Neurons_GPU: %d\n",
mean_square_error_Layer4_Neurons_GPU);
	printf("mean_square_error_Layer5_Neurons_GPU: %d\n",
mean_square_error_Layer5_Neurons_GPU);
	printf("mean_square_error_Layer6_Neurons_GPU: %d\n",
mean_square_error_Layer6_Neurons_GPU);
//	printf("mean_square_error_Layer5_Neurons_GPU: %d =
mean_square_error_Layer1_Neurons_GPU %d + mean_square_error_Layer2_Neurons_GPU %d +
mean_square_error_Layer3_Neurons_GPU %d + mean_square_error_Layer4_Neurons_GPU %d\n",
mean_square_error_Layer5_Neurons_GPU);

	printf("mean_square_error_Layer2_Neurons_CPU: %d\n",
mean_square_error_Layer2_Neurons_CPU);
	printf("mean_square_error_Layer1_Neurons_CPU: %d\n",
mean_square_error_Layer1_Neurons_CPU);
	printf("mean_square_error_Layer3_Neurons_CPU: %d\n",
mean_square_error_Layer3_Neurons_CPU);
	printf("mean_square_error_Layer4_Neurons_CPU: %d\n",
mean_square_error_Layer4_Neurons_CPU);
	printf("mean_square_error_Layer5_Neurons_CPU: %d\n",
mean_square_error_Layer5_Neurons_CPU);
	printf("mean_square_error_Layer6_Neurons_CPU: %d\n",
mean_square_error_Layer6_Neurons_CPU);

	system("PAUSE");
//ΕΚΤΥΠΩΣΗ

	printf("dlta2b delta_Layer2_Neurons_GPU: %d\n", delta_Layer2_Neurons_GPU);
	printf("nl1wb New_Layer1_Weights_GPU: %d\n", New_Layer1_Weights_GPU);

	printf("dlta3b delta_Layer3_Neurons_GPU: %d\n", delta_Layer3_Neurons_GPU);
	printf("nl2wb New_Layer2_Weights_GPU: %d\n", New_Layer2_Weights_GPU);

	printf("dlta4b delta_Layer4_Neurons_GPU: %d\n", delta_Layer4_Neurons_GPU);
	printf("nl3wb New_Layer3_Weights_GPU: %d\n", New_Layer3_Weights_GPU);

	printf("dlta5 delta_Layer5_Neurons_GPU: %d\n", delta_Layer5_Neurons_GPU);
	printf("nl4w New_Layer4_Weights_GPU: %d\n", New_Layer4_Weights_GPU);

	printf("dlta2b delta_Layer2_Neurons_CPU: %d\n", delta_Layer2_Neurons_CPU);
	printf("nl1wb New_Layer1_Weights_CPU: %d\n", New_Layer1_Weights_CPU);

	printf("dlta3b delta_Layer3_Neurons_CPU: %d\n", delta_Layer3_Neurons_CPU);
	printf("nl2wb New_Layer2_Weights_CPU: %d\n", New_Layer2_Weights_CPU);

	printf("dlta4b delta_Layer4_Neurons_CPU: %d\n", delta_Layer4_Neurons_CPU);
	printf("nl3wb New_Layer3_Weights_CPU: %d\n", New_Layer3_Weights_CPU);

	printf("dlta5 delta_Layer5_Neurons_CPU: %d\n", delta_Layer5_Neurons_CPU);
	printf("nl4w New_Layer4_Weights_CPU: %d\n", New_Layer4_Weights_CPU);
```

```
	system("PAUSE");

	printf("pinakas_Layer1_Neurons_GPU: %d\n", pinakas_Layer1_Neurons_GPU);
	printf("pinakas_Layer2_Neurons_GPU: %d\n", pinakas_Layer2_Neurons_GPU);
	printf("pinakas_Layer3_Neurons_GPU: %d\n", pinakas_Layer3_Neurons_GPU);



	for (int i = 0; i<10; ++i)
		printf("pinakas_layer1_Neurons_CPU: %f\n",
(pinakas_Layer1_Neurons_CPU[i]));
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("pinakas_layer2_Neurons_CPU: %f\n",
(pinakas_Layer2_Neurons_CPU[i]));
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("pinakas_layer3_Neurons_CPU: %f\n",
(pinakas_Layer3_Neurons_CPU[i]));
	system("PAUSE");

//ΕΚΤΥΠΩΣΗ
	printf("XOR_Z_Layer3_Neurons_GPU: %d\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %d\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %d\n", XOR_X_Layer1_Neurons_GPU);

	//printf("XOR_Z_Layer3_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU);
	//printf("XOR_Y_Layer2_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU);

	for (int i = 0; i<10; ++i)
		printf("XOR_X_Layer1_Neurons_CPU: %f\n", XOR_X_Layer1_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Y_Layer1_Neurons_CPU: %f\n", XOR_Y_Layer2_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Z_Layer1_Neurons_CPU: %f\n", XOR_Z_Layer3_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_X_Layer1_Neurons_CPU: %d\n", XOR_X_Layer1_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Y_Layer1_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU[i]);
	system("PAUSE");

	for (int i = 0; i<10; ++i)
		printf("XOR_Z_Layer1_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU[i]);
	system("PAUSE");
	//ΕΚΤΥΠΩΣΗ

	printf("XOR_X_Layer1_Neurons_CPU: %d\n", XOR_X_Layer1_Neurons_CPU);
	printf("XOR_Y_Layer2_Neurons_CPU: %d\n", XOR_Y_Layer2_Neurons_CPU);
	printf("XOR_Z_Layer3_Neurons_CPU: %d\n", XOR_Z_Layer3_Neurons_CPU);
```

```
	system("PAUSE");

	printf("XOR_Z_Layer3_Neurons_GPU: %f\n", XOR_Z_Layer3_Neurons_GPU);
	printf("XOR_Y_Layer2_Neurons_GPU: %f\n", XOR_Y_Layer2_Neurons_GPU);
	printf("XOR_X_Layer1_Neurons_GPU: %f\n", XOR_X_Layer1_Neurons_GPU);


	exit(0);
}
```

Κάποιες επιπλέον επεξηγήσεις πάνω στον κώδικα

```
// result += Layer1_Weights_GPU[weightBegin];
// Layer1_Weights_GPU[weightBegin + i];
// Layer1_Weights_CPU[156] //Layer1_Weights_GPU[156]

result += Layer1_Neurons_GPU[windowY * 29 + windowX +
kernelTemplate[i]] * Layer1_Weights_GPU[weightBegin + i]; //το κάθε
νευρώνιο που αντιστοιχεί σε κάθε pixel για την εικονα 29 χ 29 = 841
```

```
νευρωνες + πυρήνα 5 χ 5 πολ/μενο με το βάρος κάθε νευρώνα του πρωτου
επιπέδου μας δίνει το αποτέλεσμα. Βασικά εδώ γίνεται το εξής,οι
Νευρώνες που τυπικά είναι τα pixels της εικόνας 29 X 29
πολλαπλιασιάζονται με τα Βάρη  των εκπαιδευόμενων φίλτρων ,στην
προκειμένη περίπτωση είναι τα βάρη (νήματα) του φίλτρου /kernel
διαστάσεων 5 X 5 = 25 + 1 (bias/πόλωση) = 26
```

//int weightBegin =blockID * 26; //To weightBegin παίρνει σαν αρχή 26 θέσεις για Βάρη μέσα στο Block ,όπου θα φτιαχτεί στην αρχή ο πρώτος χαρακτηριστικός χάρτης (τυπικά είναι το φίλτρο) με ταυτότητα blockID. Αρα το Block στην αρχή έχει 26 θέσεις [(5 X 5) + 1(bias/πόλωση)] = 26 θέσεις για τιμές Βαρών δηλαδή όσο είναι οι διαστάσεις του φίλτρου/kernel/ πυρήνα = (5 X 5) +1 (bias/πόλωση)

/ / result += Layer1_Weights_GPU[weightBegin]; // εδώ νε μεν το weightBegin είχε στην αρχή 26 θέσεις δεσμευμένες στο Block,BlockID. Μετά όμως θα πάρει τις τιμές που έχει το Layer1_Weights_GPU[156], δηλαδή 156 θέσεις για Βάρη ,που είναι τυπικά 6 φίλτρα /πυρήνες /kernels διαστάσεων (5 X 5) + 1 (bias/πόλωση) τα οποία θα χρησιμοποιηθούν στο δεύτερο Layer Layer2 ,και αυτό διότι το Layer1_Weights_GPU[weightBegin] έχει μέσα στην παρέθενση το weightBegin που στην αρχή είχε 26 θέσεις ,μετά όμως που θα γίνει η νέα δέσμευση στο δεύτερο Layer ,Layer2 οι θέσεις θα αυξηθούν και απο 26 θέσεις θα γίνουν 156 θέσεις για τα Βάρη στο σύνολο.

Και το result+ σημαίνει ότι το αποτέλεσμα επιπρόσθετα για να υπάρχει μια συνέχεια στο ήδη υπάρχον αποτέλεσμα result που προέκυψε απο την σάρωση /πολλαπλασιασμό Νευρώνων με τα Βάρη για να φτιαχτεί ο 1^{ος} χαρακτηριστικός χάρτης απο την αρχική εικόνα διαστάσεων 29 X 29 (input Layer ή Layer1) οπότε επειδή οι συναρτήσεις είναι global που σημαίνει ότι είναι καθολικές σε όλο το πρόγραμμα και επίσης θα εκτελεστεί στο device (GPU/Κάρτα Γραφικών) . Η συνέχεια που υπάρχει στο ήδη υπάρχον αποτέλεσμα result , σημαίνει ότι το result + δηλώνει στην προκειμένη περίπτωση για το result + = Layer1_Weights_GPU[weightBegin]; ,ότι τα Βάρη που τώρα απο 26 θέσεις για Βάρη θα είναι τώρα 156 θέσεις για Βάρη στο υπάρχον αποτέλεσμα result που τώρα θα έχει νε μεν το αποτέλεσμα αλλά θα πάει στους Νευρώνες του δεύτερου επιπέδου (Layer) , Layer2_Neurons_GPU[13 * 13 * blockID + pixelY * 13 + pixelX] = result, βασικά το αποτέλεσμα result θα μεταφερθεί στο Layer2_Neurons_GPU που τυπικά είναι στο δέυτερο επίπεδο (Layer2) και το result+ θα έχει τώρα απο τις 26 θέσεις για Βάρη 156 θέσεις για Βάρη που θα τις αντλήσει απο το Layer1_Weights_GPU[156] όπου 156 το νέο weightBegin που θα χρησιμοποιηθεί για το δεύτερο επίπεδο (Layer) ,Layer2, για τα φίλτρα /kernels που είναι τώρα 6 στο αριθμό για 6 χαρακτηριστικούς χάρτες που βρίσκονται στο δεύτερο επίπεδο (Layer2) ή Layer1 εαν το input Layer το πούμε Layer1. Οι διαστάσεις των εκπαιδευόμενων φίλτρων θα είναι [(5 χ 5) +1 (πόλωση)] = 26 , άρα έχουμε 26 X 6 = 156 θέσεις για Βάρη.
//int weightBegin = blockID * 26 * 6; // To weightBegin παίρνει σαν αρχή 26 * 6 =156 θέσεις για Βάρη μέσα στο Block ,όπου θα φτιαχτούν 6 χαρακτηριστικοί χάρτες (τυπικά είναι 6 φίλτρα διαστάσεων) [(5 X 5) + 1 (πόλωση)] το καθένα , άρα έχουμε σύνολο [(5 X 5) + 1] X 6 = 156 θέσεις για Βάρη

Τα αρχεία εξόδου είναι με κατάληξη bin δηλαδή δυαδικά ,τα ανοίγω με το ultra edit 64 bit φαίνεται η δεκαεξαδική τους μορφή αλλά και η δυαδική τους.

Αρχεία Εισόδου: lw1.bin ,lw2.bin, lw3.bin ,lw4.bin ,τα αρχεία αυτά περιέχουν τις τιμές των βαρών για τους αντίστοιχους Νευρώνες που είναι στα αντίστοιχα επίπεδα : Layer1_Weights_CPU , Layer2_Weights_CPU, Layer3_Weights_CPU , Layer4_Weights_CPU

Το αρχείο neuron.bin περιέχει τις τιμές των Νευρώνων που βρίσκονται στο πρώτο επίπεδο (Layer1_Neurons_CPU)

Αρχεία Εξόδου: neuronout.bin αυτό περιέχει την έξοδο του πέμπτου επιπέδου (Layer5_Neurons_CPU)

Υπόλοιπα Αρχεία Εξόδου: Layer_1.bin , Layer_2.bin, Layer_3.bin, Layer_4.bin για τα αντίστοιχα επίπεδα (Layers), Layer1, Layer2 , Layer3 , Layer4. Σε αυτά τα αρχεία γράφονται οι νεες τιμές των Νευρώνων στα αντίστοιχα επίπεδα

Αποτελέσματα απο κώδικα απο τον Σταθερό Υπολογιστή με την κάρτα γραφικών Nvidia GT710 με 1GB και 192 πυρήνες

```
pinakas_Layer1_Neurons_GPU: 1082792960
pinakas_Layer2_Neurons_GPU: 1082808320
pinakas_Layer3_Neurons_GPU: 1085941760
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 1.000000
pinakas_layer1_Neurons_CPU: 1.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
Press any key to continue . . .
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 1.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 1.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
Press any key to continue . . .
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
pinakas_layer3_Neurons_CPU: 0.000000
Press any key to continue . . .
XOR_Z_Layer3_Neurons_GPU: 1085946880
XOR_Y_Layer2_Neurons_GPU: 1082812416
XOR_X_Layer1_Neurons_GPU: 1082796544
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
XOR_X_Layer1_Neurons_CPU: 0.000000
```

mean_square_error_Layer2_Neurons_GPU: 1082804224
mean_square_error_Layer1_Neurons_GPU: 1082789376
mean_square_error_Layer3_Neurons_GPU: 1085936640
mean_square_error_Layer4_Neurons_GPU: 1083836928
mean_square_error_Layer5_Neurons_GPU: 1083838976
mean_square_error_Layer6_Neurons_GPU: 1083839488
mean_square_error_Layer2_Neurons_CPU: 3769112
mean_square_error_Layer1_Neurons_CPU: 3773224
mean_square_error_Layer3_Neurons_CPU: 3764080
mean_square_error_Layer4_Neurons_CPU: 3763672
mean_square_error_Layer5_Neurons_CPU: 3763624
mean_square_error_Layer6_Neurons_CPU: 3763576
Press any key to continue . . .
dlta2b delta_Layer2_Neurons_GPU: 1082824704
nl1wb New_Layer1_Weights_GPU: 1083835904
dlta3b delta_Layer3_Neurons_GPU: 1085952000
nl2wb New_Layer2_Weights_GPU: 1084914176
dlta4b delta_Layer4_Neurons_GPU: 1083838464
nl3wb New_Layer3_Weights_GPU: 1087480832
dlta5 delta_Layer5_Neurons_GPU: 1083837952
nl4w New_Layer4_Weights_GPU: 1082820608
dlta2b delta_Layer2_Neurons_CPU: 3753832
nl1wb New_Layer1_Weights_CPU: 19595776
dlta3b delta_Layer3_Neurons_CPU: 3757908
nl2wb New_Layer2_Weights_CPU: 19564576
dlta4b delta_Layer4_Neurons_CPU: 3762928
nl3wb New_Layer3_Weights_CPU: 19064176
dlta5 delta_Layer5_Neurons_CPU: 3763348
nl4w New_Layer4_Weights_CPU: 19060136

layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000
layer1_Neurons_CPU: 1.000000

layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000
layer1_WEIGHTS_CPU: 1.000000

layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000
layer4_weights_CPU: 1.000000

layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000
layer3_weights_CPU: 1.000000

layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000

layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000
layer2_weights_CPU: 1.000000

pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 1.000000
pinakas_layer1_Neurons_CPU: 1.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000
pinakas_layer1_Neurons_CPU: 0.000000

pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 1.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 1.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000
pinakas_layer2_Neurons_CPU: 0.000000

layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer3_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000

layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer4_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000

layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000

layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000
layer2_WEIGHTS_CPU ekteleitai mesa apo to Neural Network: 1.000000

output layer [-842150451]=-842150451
output layer [-842150451]=-842150451
output layer [-842150451]=-842150451
output layer [-842150451]=-842150451
output layer [-842150451]=-842150451
output layer [-842150451]=-842150451

layer4_Neurons_CPU apotelesma: 1.715900
layer4_Neurons_CPU apotelesma: 1.715900
layer4_Neurons_CPU apotelesma: 1.715900

layer3_Neurons_CPU apotelesma: 0.000000
layer3_Neurons_CPU apotelesma: 0.000000
layer3_Neurons_CPU apotelesma: 0.000000

layer2_Neurons_CPU apotelesma: 1.715900
layer2_Neurons_CPU apotelesma: 1.715900
layer2_Neurons_CPU apotelesma: 1.715900
layer2_Neurons_CPU apotelesma: 1.715900

layer1_Neurons_CPU apotelesma: 1.000000
layer1_Neurons_CPU apotelesma: 1.000000
layer1_Neurons_CPU apotelesma: 1.000000

xronos: deyterolepta 0
start 0
stop 9293392

mean_square_error_Layer2_Neurons_GPU: 1082804224
mean_square_error_Layer1_Neurons_GPU: 1082789376
mean_square_error_Layer3_Neurons_GPU: 1085936640
mean_square_error_Layer4_Neurons_GPU: 1083836928
mean_square_error_Layer5_Neurons_GPU: 1083838976
mean_square_error_Layer6_Neurons_GPU: 1083839488
mean_square_error_Layer2_Neurons_CPU: 8356424
mean_square_error_Layer1_Neurons_CPU: 8360536
mean_square_error_Layer3_Neurons_CPU: 8351392
mean_square_error_Layer4_Neurons_CPU: 8350984
mean_square_error_Layer5_Neurons_CPU: 8350936
mean_square_error_Layer6_Neurons_CPU: 8350888

ΕΠΙΛΟΓΟΣ

Η εκπαίδευση πάνω στην XOR γίνεται μόνο κατά την εμπρόσθια διάδοση. Επίσης οι νευρώνες αλλά και τα βάρη των νευρώνων στην εργασία/κώδικας μας σε όλα τα επίπεδα/στρώματα παίρνουνε τιμές 0 και 1 ,θα μπορούσε κάλιστα να πάρει και άλλες τιμές πέρα απο τις δυαδικές τιμές δηλαδή απο το δεκαδικό σύστημα. Επίσης όπως αναφέραμε και προτίτερα ο κώδικας που έχουμε υλοποιήσει είναι ενα συνελικτικό δίκτυο ,που ουσιαστικά είναι μια ειδική κατηγορία perceptron πολλών επιπέδων, 5 επιπέδων .Αυτά τα δίκτυα είναι κατάλληλα για ταξινόμηση προτύπων.[8]. Επίσης το συνελικτικό δίκτυο είναι όπως αναφέραμε ένα perceptron πολλών στρωμάτων και είναι ειδικά σχεδιασμένο έτσι ώστε να αναγνωρίζει δισδιάστατα σχήματα με μεγάλο βαθμό ευαισθησίας στην μετατόπιση και άλλες μορφές παραμόρφωσης.[8]. Ο κώδικας που έχουμε αναπτύξει είναι και πρόσθιας τροφοδότησης (feedforward network) που ουσιαστικά είναι η ροή της πληροφορίας που πάει απο αριστερά προς τα δεξιά και απο επίπεδο σε επίπεδο. Οι νευρώνες του προηγούμενου στρώματος τροφοδοτούν τους νευρώνες του επόμενους στρώματος.[3]. Και επίσης είναι και η ανάδρομη διάδοσης σφάλματος ή backpropagation και ο λόγος είναι ότι με την ανάδρομη μπορούμε να αναπαραστήσουμε και πιο περίπλοκες διεργασίες και είναι σε περίπλοκα νευρωνικά δίκτυα. Με την ανάδρομη διάδοση σφάλματος αφού ξεκινήσει η διαδικασία εκμάθησης με τιμές βαρών που θα έχει ο κάθε νευρώνας μόλις δωθεί λάθος απάντηση ,τότε τα βάρη διορθώνονται μέχρι το σφάλμα να γίνει μικρότερο και αυτό θα επαναληφθεί πολλές φορές μέχρις το σφάλμα να γίνει πάρα πολύ μικρό. Οπότε τότε θα λέμε ότι το δίκτυο έχει μάθει.

Η διαδικασία εκπαίδευσης κατα τον αλγόριθμο της όπισθεν διάδοσης σφάλματος λαμβάνει χώρα με την παρουσίαση και εφαρμογή στο δίκτυο ενός συνόλου παραδειγμάτων εκπαίδευσης [5]. Η παρουσίαση όλων των προτύπων εκπαίδευσης μια φορά το καθένα στο δίκτυο λέγεται εποχή (epoch). Οσο γίνεται η εκπαίδευση εκτελούνται επαναλήψεις των εποχών, μέχρι να σταθεροποιηθούν τα βάρη σε συγκεκριμένες τιμές που θα προκαλούν σύγκλιση της μέσης τιμής των σφαλμάτων, για όλα τα πρότυπα εκπαίδευσης, στην μικρότερη τιμή της. Μην ξεχάσουμε να αναφέρουμε ότι η εκπαίδευση γίνεται με επίβλεψη.

Η διαδικασία αποτελείται απο δυο περάσματα διαμέσου των επιπέδων του δικτύου, το ένα πέρασμα γίνεται προς τα εμπρός και το άλλο πέρασμα γίνεται προς τα πίσω. Στο εμπρός πέρασμα έχουμε το διάνυσμα εισόδου που θα εφαρμοστεί στους νευρώνες εισόδου και θα προχωρήσει μέσα στο δίκτυο απο επίπεδο σε επίπεδο μέχρι να φτάσει στο τέλος όπου έχουμε τους εξόδους που είναι και η πραγματική απόκριση του δικτύου. Στο πέρασμα αυτό τα βάρη είναι σταθερά [3]. Μετά έχουμε το πίσω πέρασμα όπου τα βάρη μεταβάλλονται σύμφωνα με τον κανόνα διόρθωσης του λάθους. Πιο συγκεκριμένα η πραγματική απόκριση (εξοδος) του δικτύου αφαιρείται απο την επιθυμητή έξοδο για να δημιουργηθεί ένα σήμα λάθους το οποίο και διαδίδεται προς τα πίσω στο δίκτυο. Μετά τα συναπτικά βάρη προσαρμόζονται έτσι ώστε η πραγματική απόκριση (εξοδος) να πλησιάσει την επιθυμητή έξοδο.

Επίσης και η nvidia που παράγει τις κάρτες γραφικών με τους πολλούς παράλληλους επεξεργαστές κοινόχρηστης μνήμης που ο καθένας απο τους οποίους διαθέτει έναν αριθμό πυρήνων.[11]. Η εκτέλεση των προγραμμάτων σε μια κάρτα γραφικών

πραγματοποιείται απο τους πυρήνες της.[11] Τα προγράμματα ,που είναι γραμμένα σε μια γλώσσα προγραμματισμού ,όπως είναι η C ή C++ που υποστηρίζεται απο το CUDA project μπορούν να χειριστούν μόνο λογικές μονάδες επεξεργασίας , δηλαδή νήματα εκτέλεσης (execution threads) που φυσικά ανατίθονται απο το υλικό στους πυρήνες προς εκτέλεση. Και τα φυσικά τα νήματα είναι διατεταγμένα και οργανωμένα σε πλέγματα (grids).[11]

Και φυσικά οι διαστάσεις του πλέγματος και του μπλοκ καθορίζονται κατά τον χρόνο εκτέλεσης του προγράμματος.[11].

Η nvidia λοιπόν όσον αφορά για τα νευρωνικά δίκτυα και την βαθιά μηχανική μάθηση, που είναι η ικανότητα του υπολογιστή να δημιουργεί μοντέλα ή πρότυπα απο ένα σύνολο δεδομένων,[14] υποστηρίζει όσον αφορά τον προγραμματισμών των καρτών γραφικών την εμπρόσθια διάδοση σφάλματος αλλά και την ανάδρομη διάδοση σφάλματος.

Οπως ο κλάδος των ΤΝΔ, η Μηχανική Μάθηση η οποία ασχολείται με τη μελέτη αλγορίθμων που θα βελτιώνουν τη συμπεριφορά τους σε κάποια εργασία που τους έχει ανατεθεί χρησιμοποιώντας την εμπειρία τους.[14]
Όσον αφορά τη σχεδίαση των συστημάτων Μηχανικής Μάθησης, για τα συστήματα που ανήκουν στη συμβολική ΤΝΔ, η δυνατότητα μάθησης προσδιορίζεται ως η ικανότητα απόκτησης επιπλέον γνώσης, που θα επιφέρει κάποιες μεταβολές στην ήδη υπάρχουσα καταχωρημένη γνώση είτε αλλάζοντας χαρακτηριστικά της είτε με αυξομείωσή της.[14] Στην περίπτωση των συστημάτων ΤΝΔ που ανήκουν στη Μη Συμβολική ΤΝ (όπως η περίπτωση των Τεχνητών Νευρωνικών Δικτύων), ως μάθηση προσδιορίζεται η δυνατότητα που διαθέτουν τα συστήματα στο να μετασχηματίζουν την εσωτερική τους δομή, παρά στο να μεταβάλλουν κατάλληλα τη γνώση που έχει καταχωρηθεί μέσα σε αυτά κατά το σχεδιασμό τους.[14]

Στην εργασία μας η εκτέλεση του προγράμματος γίνεται παράλληλα απο τα νήματα που ανατίθονται απο το υλικό στους πυρήνες. Ο χρόνος εκτέλσης φυσικά είναι πάρα πολύ μικρός όταν εκτελείται παράλληλα απο ότι εάν εκτελούνταν σειρακά. Το μόνα τμήματα που μας καθυστερεί λίγο στον χρόνο εκτέλεσης είναι το επίπεδο 5 και το επίπεδο 4 που πλήρως διασυνδεδεμένα, είναι σαν να μαζέυονται όλα τα πλακίδια ή τα κομμάτια απο τα προηγούμενα επίπεδα/στρώματα και να πηγαίνουνε στο προτελευταίο και στο τελευταίο στρώμα/επίπεδο γιαυτό και η καθυστέρηση στο χρόνο εκτέλσης. Τυπικά το προτελευταίο και το τελευταίο επίπεδο/στρώμα συνδέονται με όλους τους νευρώνες των προηγουμένων επιπέδων/στρωμάτων.

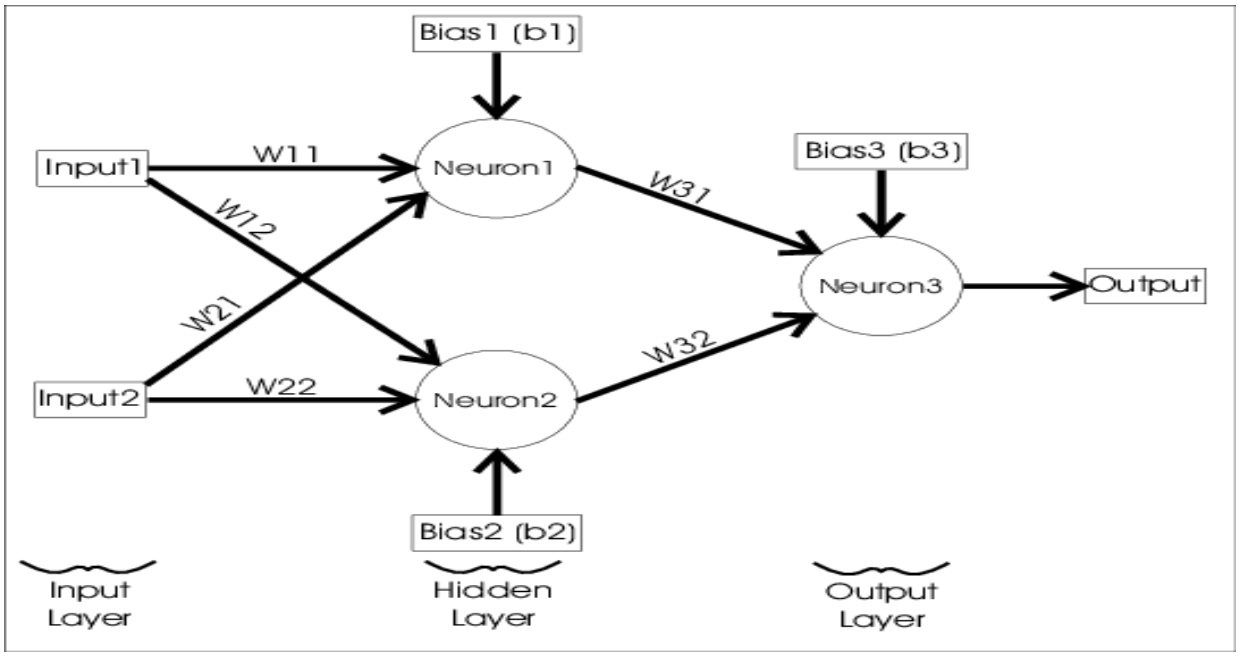
Επίσης η εργασία μας είναι ενα συνελικτικό δίκτυο με εναλλαγές μεταξύ συνέλιξης και υποδειγματοληψίας. Η συνέλιξη γίνεται απο τους νευρώνες ,οι οποίοι κάνουν την συνέλιξη των φίλτρων που σχετίζονται με την εικόνα που δέχονται στην είσοδο. Η διαφορά των συνελικτικών με τα νευρωνικά δίκτυα είναι ότι στα συνελικτικά δίκτυα στις εισόδους τους έχουμε μια εικόνα ή ένα σχήμα. Αρα το θέμα είναι χωρικό. Γιαυτό μετά το συνελικτικό χρησιμοποιεί ένα πλήθως επιπέδων για να μειώσει τις διαστάσεις των χαρακτηριστικών ,την κανονικοποίηση και άλλες σχετικές λειτουργίες.[9]. Αρα τα συνελικτικά περιέχουν πολύ μικρότερο αριθμό παραμέτρων προς εκμάθηση σε σχέση με τα νευρωνικά δίκτυα.[9].

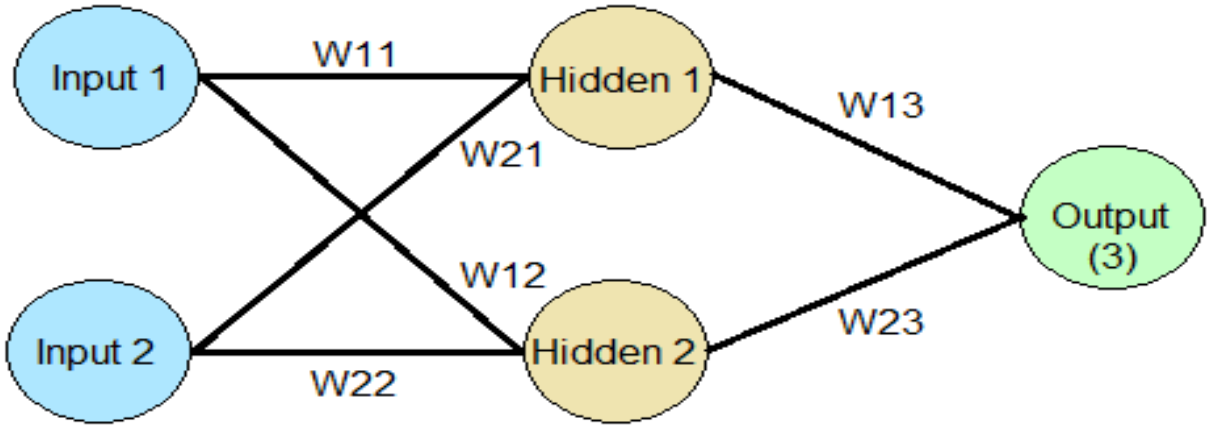
Με τα διαδοχικά υπολογιστικά επίπεδα του δικτύου να εναλλάσονται μεταξύ συνέλιξης και υποδειγματοληψίας, καταλήγουμε στο εξής αποτέλεσμα ότι σε κάθε συνελικτικό επίπεδο ή επίπεδο υποδειγματοληψίας να έχουμε αύξηση του αριθμου των χαρτών χαρακτηριστικών, ενώ η χωρική ανάλυση να μειώνεται σε σχέση με το προηγούμενο επίπεδο.[8]

Επίσης με την CUDA μας δίνεται η δυνατότητα να σχεδιάσουμε και να δημιουργήσομε κώδικες όχι μόνο με χρήση νευρωνικών δικτύων αλλά για άλλου είδους δίκτυα και επίσης ανοίγει ένα εύρος εφαρμογών όπως για σεισμικά μοντέλα ,πρόβλεψη σεισμών με χρήση νευρωνικών δικτύων με χρήση CUDA , μετεωρολογικά δεδομένα, επίσης μπορούν να χρησιμοποιηθούν με ενσωματωμένα συστήματα ή με συστήματα απομακρυσμένου ελέγχου του τύπου IoT , επίσης άλλη χρήση θα ήτανε στις τηλεπικοινωνίες που κάνουν χρήση στα στοχαστικά μοντέλα και άλλα πολλά. Επίσης εχουμε CUDA με Python με Matlab όλοι αυτοί οι συνδυασμοί. Επίσης υπάρχει και το προγραμματάκι/patch που υπολογίζει επίσης νευρωνικά δίκτυα TensorFlow το οποίο εγκαθίστανται πάνω στο Microsoft visual όπως και η CUDA και συνεργάζονται όλα μεταξύ τους . TensorFlow supports CUDA 10.0.

Τέλος θα ήθελα να ευχαριστήσω τον Κ. Κωνσταντάρα που με εμπιστεύτηκε για την υλοποιήση την εργασίας , καθώς και τους υπόλοιπους καθηγητές σε όλην την διάρκεια των μεταπτυχαικών μαθημάτων. Επίσης θα ήθελα να αφιερώσω την εργασία μου στον Πατέρα μου απο κει που με βλέπει στην γυναίκα μου και τα παιδιά μου.

ΤΥΠΟΛΟΓΙΟ





$$E = MSE = \frac{1}{2} * \sum (target - result)$$

MSE = Μεσο Τετραγωνικο Σφαλμα

target = επιθυμητη αποκριση

result = πραγματικη αποκριση

ΑΝΑΔΡΟΜΟ ΠΕΡΑΣΜΑ

$$\boldsymbol{\delta oj = (Rj - Fj) * [Fj * (1 - Fj)]}$$

δoj = δελτα σφαλμα εξωτερικου νευρωνα

R = επιθυμητη εξοδος

F = πραγματικη εξοδος

$$\boldsymbol{Woj\ (new) = Woji(old) + \alpha * \delta oj * H}$$

H =Neuron = Νευρωνας

Woj = Weight = Βαρος εξωτερικου νευρωνα

α = ρυθμός εκμάθησης

H1 = f(a) = f [x1 * w1 + x2 * w2+ Bias * wb]

Bias = WojBias(new) =Woji(old) + α * δoj * Bias

$$\boldsymbol{H1 = \frac{1}{1 + \exp -(x1 * w1 + x2 * w2 + bias * wb)}}$$

H1 = sigmoid function = σιγμοειδης συνάρτηση Νευρωνα 1

$$\boldsymbol{H2 = \frac{1}{1 + \exp -(x1 * w1 + x2 * w2 + bias * wb)}}$$

H2 = sigmoid function = σιγμοειδης συνάρτηση Νευρωνα 2

WH1 = Woj1(new)

WH2 = Woj2(new)

WH1 = Woj1(new) = Woj1(old) + α * δoj * H1

WH2 = Woj2(new) = Woj2(old) + α * δoj * H2

ΠΙΟ ΠΙΣΩ ΑΝΑΔΡΟΜΟ ΠΕΡΑΣΜΑ

WhjB1: $\boldsymbol{\delta hj = [H1 * (1 - Hj)]\, \Sigma \delta ok * Wokj}$

$$\boldsymbol{\delta hj1 = [H1 * (1 - H1)]\, \Sigma \delta ok * Wok1}$$

$$\boldsymbol{WhjB1\ (new) = WhiB1(old) + \alpha * \delta hj * B}$$

$$\boldsymbol{WhjB1\ (new) = WhiB1(old) + \alpha * \delta hj1 * B}$$

WhjB2: $\boldsymbol{\delta hj2 = [H2 * (1 - H2)]\, \Sigma \delta ok * Wok2}$

$$\boldsymbol{WhjB2\ (new) = WhiB2(old) + \alpha * \delta hj2 * B}$$

Wh1x1: $\boldsymbol{\delta hj = [H1 * (1 - Hj)]\, \Sigma \delta ok * Wokj}$

$$\boldsymbol{\delta h1x1 = [H1 * (1 - H1)]\, \Sigma \delta ok * Wok1}$$

$$\boldsymbol{Wh1x1\ (new) = Whix1(old) + \alpha * \delta h1x1 * X1}$$

Wh1x2: $\boldsymbol{\delta hj = [H1 * (1 - Hj)]\, \Sigma \delta ok * Wokj}$

$$\boldsymbol{\delta h1x2 = [H1 * (1 - H1)]\, \Sigma \delta ok1 * Wok1}$$

$$\boldsymbol{Wh1x2\ (new) = Wh1x2(old) + \alpha * \delta h1x2 * X2}$$

Wh2x2: $\boldsymbol{\delta hj = [H1 * (1 - Hj)]\, \Sigma \delta ok * Wokj}$

$$\boldsymbol{\delta h2x2 = [H2 * (1 - H2)]\, \Sigma \delta ok2 * Wok2}$$

$$\boldsymbol{Wh1x2\ (new) = Wh2x2(old) + \alpha * \delta h2x2 * X2}$$

Wh2x1: $\boldsymbol{\delta hj = [H1 * (1 - Hj)]\, \Sigma \delta ok * Wokj}$

$$\boldsymbol{\delta h2x1 = [H2 * (1 - H2)]\, \Sigma \delta ok2 * Wok2}$$

$$\boldsymbol{Wh2x1\ (new) = Wh2x1(old) + \alpha * \delta h2x1 * X1}$$

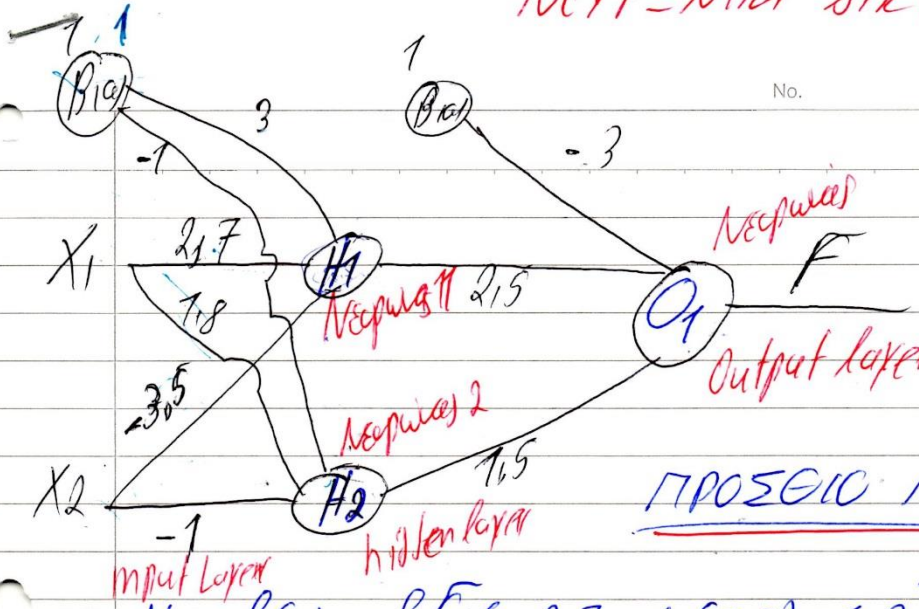
Appendix:

Χειρόγραφο Κείμενο

Παράδειγμα υπολογισμού ενός απλού Νευρωνικού δικτύου 3 επιπέδων

Η διαδικασία αποτελείται απο δυο περάσματα διαμέσου των επιπέδων του δικτύου, το ένα πέρασμα γίνεται προς τα εμπρός και το άλλο πέρασμα γίνεται προς τα πίσω. Στο εμπρός πέρασμα έχουμε το διάνυσμα εισόδου που θα εφαρμοστεί στους νευρώνες εισόδου και θα προχωρήσει μέσα στο δίκτυο απο επίπεδο σε επίπεδο μέχρι να φτάσει στο τέλος όπου έχουμε τους εξόδους που είναι και η πραγματική απόκριση του δικτύου. Στο πέρασμα αυτό τα βάρη είναι σταθερά . Μετά έχουμε το πίσω πέρασμα όπου τα βάρη μεταβάλλονται σύμφωνα με τον κανόνα διόρθωσης του λάθους. Πιο συγκεκριμένα η πραγματική απόκριση (εξοδος) του δικτύου αφαιρείται απο την επιθυμητή έξοδο για να δημιουργηθεί ένα σήμα λάθους το οποίο και διαδίδεται προς τα πίσω στο δίκτυο

ΝΕΥΡΩΝΙΚΗ ΣΙΚΤΥΑ



ΠΡΟΣΘΙΟ ΠΕΡΑΣΜΑ

$$H1 = f(a) = f[X_1 \cdot 2,7 + X_2(-3,5) + 1 \cdot 3]$$

$$H1 = \frac{1}{1+e^{-(X_1 \cdot 2,7 + X_2(-3,5) + 1 \cdot 3)}}$$

$$H_2 = f(a) = f[X_1 \cdot 1,8 + X_2(-1) + 1 \cdot (-1)]$$

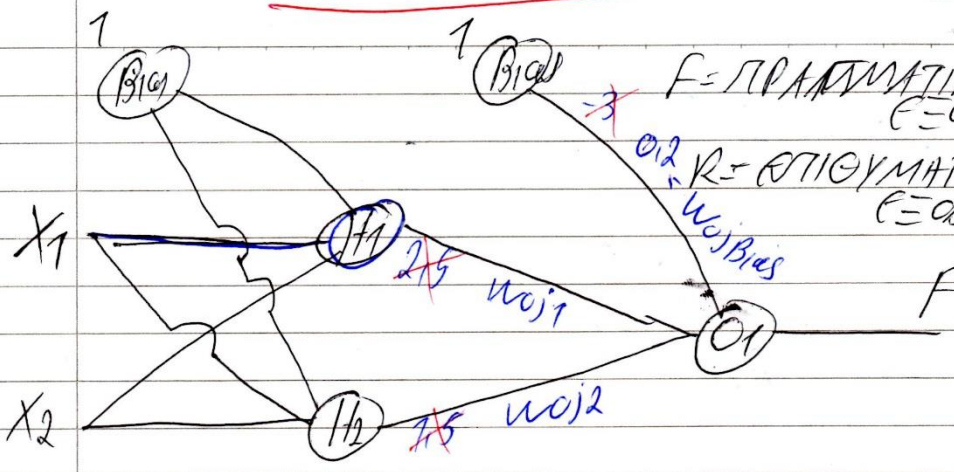
$$H_2 = \frac{1}{1+e^{-[X_1 \cdot 1,8 + X_2(-1) + 1(-1)]}} =$$

$$F = f(a) = f[H_1 \cdot 2,5 + H_2 \cdot 1,5 + 1(-3)]$$

$$F = \frac{1}{1+e^{-[H_1 \cdot 2,5 + H_2 \cdot 1,5 + 1(-3)]}} =$$

http://www.tancenghong.com

ΔΙΑΔΟΣΗ ΠΡΟΣΩ



$$\delta oj = (Rj - Fj) \cdot [Fj(1-Fj)]$$

Για παράδειγμα εάν $F = 0,5$ και $R = 1$

$$\delta oj = (1-0,5) \cdot [0,5(1-0,5)] = 0,125$$

περνάμε διαίρεση τιμή για το $a = 1,5$

$$Woj(new) = Woji(old) + a \cdot \delta oj \cdot H$$

a = ρυθμός εκμάθησης

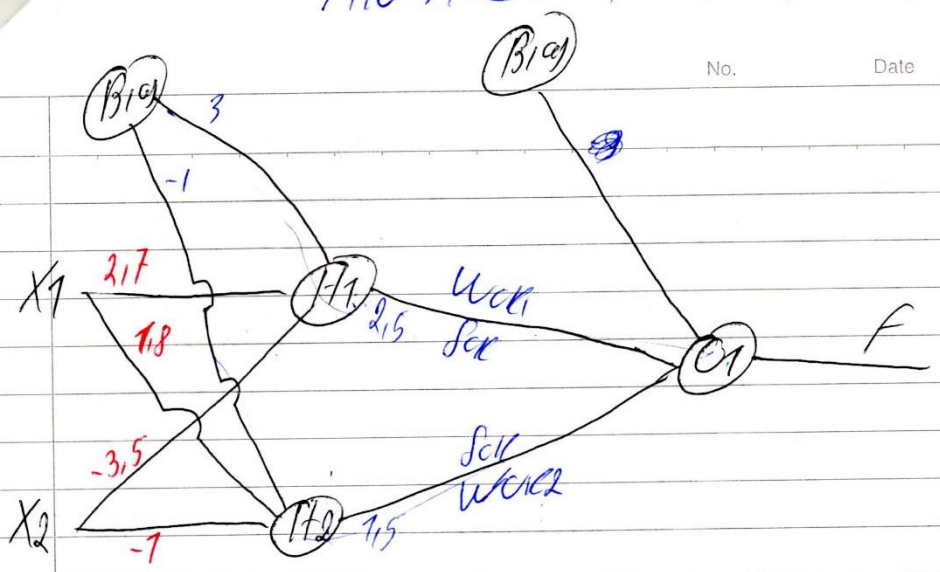
$$Bias = Woj\,Bias\,(new) = -3 + 1,5 \cdot \delta oj \cdot Bias =$$
$$= -3 + 1,5 \cdot \delta oj \cdot 1 \simeq 0,2 \text{ ΝΕΑ ΤΙΜΗ}$$

$$W_{H1} = Woj\,1\,(new) = 2,5 + 1,5 \cdot \delta oj \cdot H1 =$$

$$W_{H2} = Woj\,2\,(new) = 1,5 + 1,5 \cdot \delta oj \cdot H2 =$$

http://www.tanenghong.com

ΠΙΟ ΠΙΣΩ ΑΝΑΔΡΟΜΟ ΠΕΡΑΣΜΑ



W_{hjB1}: $\delta_{hj} = [H_j \cdot (1 - H_j)] \sum \delta_{oi} \cdot W_{oikj}$

$\delta_{hj} = [H_1 \cdot (1 - H_1)] \cdot 0,125 \cdot 2,5 =$

$W_{hjB1}(new) = W_{hjB1}(old) + a \cdot \delta_{hj} \cdot B$

$W_{hjB1} = 3 + 1,5 \cdot \delta_{hj} \cdot 1$

W_{hjB2} : $\delta_{h2} = [H_2 (1 - H_2)] \cdot \sum \delta_{oi} \cdot W_{oik2}$

$\delta_{h2} = [H_2 \cdot (1 - H_2)] \cdot 0,125 \cdot 1,5 =$

$W_{hjB2}(new) = W_{hjB2}(old) + a \delta_{h2} \cdot B$

$W_{hjB2} = -1 + 1,5 \cdot \delta_{h2} \cdot 1$

http://www.tancnghong.com

-3-

No. Date

w_{h1x1}: $\delta_{h1x1} = [H1 \cdot (1-H1)] \sum \delta_{ok} \cdot w_{ok1} =$

$\delta_{h1x1} = [H1 \cdot (1-H1)] 0,125 \cdot 2,5$

$w_{h1x1}(new) = w_{h1x1}(old) + \alpha \delta_{h1x1} \cdot x1$

$w_{h1x1} = 2,7 + 7,5 \cdot \delta_{h1x1} \cdot x1$

---

w_{h1x2}: $\delta_{h1x2} = [H1 \cdot (1-H1)] \sum \delta_{ok} w_{ok1}$

$\delta_{h1x2} = [H1 \cdot (1-H1)] \sum \delta_{ok} w_{ok}$

$= [H1 \cdot (1-H1)] 0,125 \cdot 2,5$

$w_{h2x2}(new) = w_{h2x2}(old) + \alpha \delta_{h2x2} \cdot x2$

$w_{h2x2} = -3,5 + 7,5 \delta_{h2x2} \cdot x2$

http://www.tanenghong.com

No. Date

$w_{h2x2}: \delta_{h2x2} = [H2 \cdot (1-H2)] \sum \delta_{ok2} w_{ok2}$

$\delta_{h2x2} = [H2 \cdot (1-H2)] \delta_{ok2} w_{ok2}$

$= [H2 \cdot (1-H2)] \cdot 0,125 \cdot 1,5$

$w_{h2x2}(new) = w_{h2x2}(old) + a \delta_{h2x2} \cdot x_2$

$w_{h2x2} = -1 + 1,5 \cdot \delta_{h2x2} \cdot x_2$

$w_{h2x1}: \delta_{h2x1} = [H2(1-H2)] \sum \delta_{ok2} w_{ok2}$

$\delta_{h2x1} = [H2(1-H2)] \delta_{ok2} w_{ok2}$

$= [H2(1-H2)] 0,125 \cdot ~~w_{ok2}~~ 1,5$

$w_{h2x1}(new) = w_{h2x1}(old) + a \delta_{h2x1} \cdot x_1$

$w_{h2x1} = 1,8 + 1,5 \cdot \delta_{h2x1} \cdot x_1$

http://www.tanenghong.com

No. Date

Εαν παιρνει με XOR τοτε λυση διαφορη

X_1	X_2	R
0	0	0
0	1	1
1	0	1
1	1	0

~~$f = X_1 \oplus X_2$~~

$R = X_1 \oplus X_2$

Για το πρώτο πέρασμα εαν

$$H1 = \frac{1}{1+e^{-(X_1 \cdot 2,7 + X_2 \cdot (-3,5) + 7,3}} =$$

εαν $X_1 = 0 = X_2$ τοτε

$$H1 = \frac{1}{1+e^{-[0 \cdot 2,7 + 0 \cdot (-3,5) + 7,3]}} = \frac{1}{1+e^{-7,3}} =$$

$$\frac{1}{1+0,2725} = \frac{1}{1,2725} = 0,78585$$

$$H2 = \frac{1}{1+e^{-(X_1 \cdot 7,8 + X_2 (-7,6) + 7,33}} = 0,78585$$

http://www.tanenghong.com

No. Date

Εαν $X_1 = 0$

$X_2 = 1$ τοτε

$$H_1 = \frac{1}{1+e^{-[X_1 \cdot 2,7 + X_2(-3,5) + 1,3]}} =$$

$$\frac{1}{1+e^{-[0 \cdot 2,7 + 1 \cdot (-3,5) + 1,3]}} = \frac{1}{1+e^{-[0-3,5+1,3]}}$$

$$\frac{1}{1+e^{2,2}} = \frac{1}{1+9,0250} = \frac{1}{10,0250} = 0,0997$$

$$H_2 = \frac{1}{1+e^{-[X_1 \cdot 1,8 + X_2 \cdot (-1) + 1(-1)]}} = \frac{1}{1+e^{-[0+0-1-1]}}$$

$$\frac{1}{1+e^{-[-2]}} = \frac{1}{1+e^{+2}} = \frac{1}{1+7,389} = 0,1192$$

Εαν $X_1 = 1$ και $X_2 = 0$ τοτε

$$H_1 = \frac{1}{1+e^{-[X_1 \cdot 2,7 + X_2(-3,5) + 1,3]}} = \frac{1}{1+e^{-[2,7+0+1,3]}} =$$

$$= \frac{1}{1+e^{-4}} = \frac{1}{1+0,018} = 0,9823$$

http://www.tanenghong.com

-8-

No. Date

$$H_2 = \frac{1}{1+e^{-[X_1 \cdot 1{,}8 + X_2(-1) + 1(-1)]}} = \frac{1}{1+e^{-[2{,}7+0-1]}} =$$

$$= \frac{1}{1+e^{-1{,}7}} = \frac{1}{1+0{,}1826} = 0{,}8455$$

Kal $X_1 = 1$ dan $X_2 = 1$ maka

$$H_1 = \frac{1}{1+e^{-[X_1 \cdot 2{,}7 + X_2(-3{,}5) + 1{,}3]}} = \frac{1}{1+e^{-[2{,}7 - 3{,}5 + 1{,}3]}}$$

$$\frac{1}{1+e^{-0{,}5}} = \frac{1}{1+0{,}606} = 0{,}62206$$

$$H_2 = \frac{1}{1+e^{-[X_1 \cdot 1{,}8 + X_2(-1) + 1 \cdot (-1)]}} = \frac{1}{1+e^{-[1{,}8 - 1 - 1]}} =$$

$$= \frac{1}{1+e^{+0{,}2}} = \frac{1}{1+1{,}22} = \frac{1}{2{,}22} = 0{,}4504$$

http://www.tanenghong.com

No. Date

για $X1=0$ $X2=0$

τότε

$$F = f(H1 \cdot 2{,}5 + H2(1{,}5) + 1(-3)$$
$$(0{,}78585 \cdot 2{,}5 + 0{,}78585 \cdot 1{,}5 - 3)$$
$$(1{,}964625 + 1{,}178775 - 3) = 0{,}1434$$

$$F = \frac{1}{1+e^{-(H1 \cdot 2{,}5 + H2 \cdot 1{,}5 + (-3))}} =$$

$$f = \frac{1}{1+e^{-(0{,}78585 \cdot 2{,}5 + 0{,}78585 \cdot 1{,}5 - 3)}} =$$

$$f = \frac{1}{1+e^{-0{,}1434}} = \frac{1}{1+0{,}866} = 0{,}535905068$$

Στο δεύτερο πέρασμα για

$X1=0$ και $X2=0$ έχουμε

για $R=0$ πραγματικό

$$\delta_{oj} = (R_j - f_j)[f_j(1-f_j)]$$
$$= (0-0{,}5)[0{,}5(1-0{,}5)] =$$
$$= -0{,}5[0{,}5 - 0{,}25] = -0{,}25 + 0{,}125$$
$$= +0{,}125$$
$$= -0{,}125$$

http://www.tanenghong.com

No. Date

Пр X1=1 кг X2=1 R=0

$F = F[W_1 \cdot 2{,}5 + W_2(7{,}5) + 1(-3)]$

$F[0{,}0266 \cdot 2{,}5 + 0{,}4504 \cdot 7{,}5 - 3]$

$F[1{,}56665 + 0{,}6756 - 3]$ =

$= -0{,}76775$

На данном этапе

X1=1 кг X2=1 R=0

$S(c) = (R_j - F_j)[F_j(1 - F_j)]$

$(0 - (-0{,}76775))[(-0{,}76775)(1 - 0{,}76775)] =$

$0{,}76775[-0{,}76775 + 0{,}589944]$

$0{,}76775(-0{,}17831) = -0{,}136897$

http://www.tanenghong.com

Για $X1=0$ και $X2=1$

$F = f(H_1 \cdot 2,5 + H_2 \cdot 1,5 + 1(-3)$

$(0,0997 \cdot 2,5 + 0,1192 \cdot 1,5 + 1(-3)$

$(0,24925 + 0,17881 - 3) = -2,57795$

Στο πλαίσιο πρέπει να

$X1=0$ $X2=1$ έχεις $R=1$

$S(j) = (R_j - F_j)[F_j(1-F_j)] =$

$= (1-(-2,57795)[(-2,57795)][1-(-2,57795)] =$

$= (3,57795)[-2,57795 - [0,6149 \quad]] =$

$= 3,57795 \cdot (-9,19686) = -32,8980$

http://www.tanenghong.com

No. Date

Για $X_1 = 1$ και $X_2 = 0$ $R = 1$

$F = f([H_1 \cdot 2,5 + H_2 \cdot 7,5 + 1(-3)]) =$

$= f[0,9823 \cdot 2,5 + 0,8455 \cdot 7,5 - 3] =$

$= f[2,45575 + 7,26825 - 3] = 0,724$

Για άνισους πίθανες τιμές

$X_1 = 1$ και $X_2 = 0$ $R = 1$

$S(i) = (R) - F_i)[F_i (1 - F_i)]$

$= (1 - 0,724)[0,724(1 - 0,724)] =$

$= (0,276)[0,724 - 0,524176]$

$(0,276)(0,199824) = 0,05515$

http://www.tanenghong.com

παίρνουμε αυτούσια τιμή για το α=1,5
για λύσουμε πρόβλημα

~~W0j(new)~~ k=0 για x1=0 x2=0 s=0,125

$W_{0j}(new) = W_{0j}(old) + a \cdot \delta_{0j} \cdot H$

$W_{cj}(new) = W_{0j}(old) + a \cdot \delta_{cj} \cdot Bias$

$Bias = W_{0j}Bias(new) = -3 + 1,5 \cdot (-0,125) \cdot 1$

$= -3 - 0,1875 = -3,1875$

$W_{H1} = W_{0H1}(new) = 2,5 + 1,5 \cdot (-0,125) \cdot H1 =$

$2,5 + 1,5(-0,125) \cdot 0,78585$

$2,5 - 0,1474 = 2,3526$

$W_{H2} = W_{0H2}(new) = 1,5 + 1,5 \cdot (-0,125) \cdot (0,78585) =$

$= 1,5 + (-0,1474) = 1,3526$

http://www.tanenghong.com

Πε πΣ= ΑΝΑΔΟΜΟ ΠΕΡΑΣ
ΓΙΑ X1=0 X2=0 Z=0

No. Date

← B1 ως 1

$w_{hj}B1$: $S_{hj} = [H_1(1-H_1)] \Sigma S_{oc} \cdot w_{oc}$

$S_{hj} = [0,78585(1-0,78585)] \Sigma(-0,125) \cdot 2,5 =$

$S_{hj} = [0,78585 - 0,61756] (-0,3125)$

$= 0,18829 \cdot (-0,3125) = -0,0526$

$w_{hjB1(new)} = w_{hjB1(old)} + \alpha \cdot S_{hj} \cdot B \Rightarrow$

$w_{hjB1} = 3 + 1,5 \cdot (-0,0526) \cdot 1$

$w_{hjB1} = 3 - 0,0789 = 2,9211$

$w_{hj}B2$: $S_{hj2} = [H_2(1-H_2)] S_{oc} \cdot w_{oc2}$

$S_{hj2} = [0,78585(1-0,78585)](-0,125) \cdot 1,5$

$= [0,78585 - 0,61756] (0,1875) =$

$= 0,16829 (-0,1875) = -0,03155$

$w_{hjB2(new)} = w_{hjB2(old)} + \alpha S_{hj2} \cdot B$

$w_{hjB2} = -1 + 1,5 \cdot (-0,03155) \cdot 1$

http://www.tanenghong.com

No. Date

$w_{h1}B2 = -1 - 0{,}047325 = -1{,}047325$

$w_{h1}B2 = -1{,}047325$

$w_{h1}x1 : \delta_{h1}x1 = [H1 \cdot (1-H1)] \Sigma \delta_{ok} \cdot w_{ok1} =$

$\delta_{h1}x1 = [0{,}78585(1-0{,}78585)](-0{,}125) \cdot 2{,}5$

$= [0{,}78585 - 0{,}61756](-0{,}3125) \cdot$

$= -0{,}0526$

$w_{h1}x1(new) = w_{h1}x1(old) + a \cdot \delta_{h1}x1 \cdot x1$

$w_{h1}x1 = 2{,}7 + 1{,}5 \cdot (-0{,}0526) \cdot 0$

$w_{h1}x1 = 2{,}7$

http://www.tanenghong.com

No. Date

~~w11~~

w_{h1x2}: $\delta_{h1x2} = [H1 \cdot (1-H1)] \delta_{o1} w_{o1o1}$

$= [0,78585(1-0,78585)][(-0,125) \cdot 2,5$

$= [0,78585 - 0,614756] \cdot (-0,3125)$

$= -0,0526$

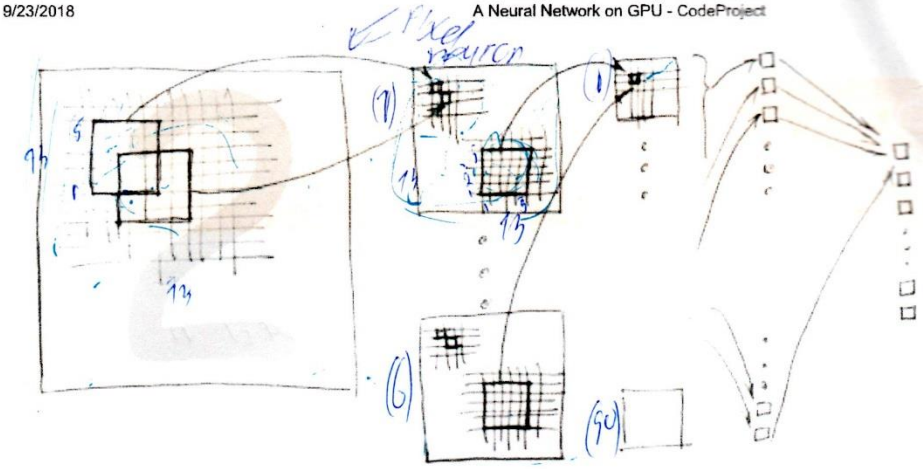
$w_{h1x2(new)} = w_{h1x2(old)} + \alpha \delta_{h1x2} \cdot x_2$

$= -3,5 + 1,5 \cdot (-0,0526) \cdot 0$

$= -3,5$

-18

http://www.tanenghong.com



The neural network we implemented was a 5 layer network called convolutional neural network. This kind of network is proven to be suitable for recognizing handwritten digits. For more theoretical details, please check out Mike's article and the references he has listed.

The first three layers of our neural network consist of several feature maps. Each of them is shrunken from the previous layer. Our input is a 29*29 image of a digit. Therefore, we have 29*29=841 neurons in the first layer. The second layer is a convolutional layer with 6 feature maps. Each feature map which is a 13*13 image is sampled from the first layer. Each pixel/neuron in a feature map is a 5*5 convolutional kernel of the input layer. So, there are 13*13*6 = 1014 nodes/neurons in this layer, and (5*5+1(bias node))*6 = 156 weights, 1014*(5*5+1) = 26364 connections linking to the first layer.

Layer 3 is also a convolutional layer, but with 50 smaller feature maps. Each feature map is 5*5 in size, and each pixel in these feature maps is a 5*5 convolutional kernel of corresponding areas of all 6 feature maps of the previous layer. There are thus 5*5*50 = 1250 neurons in this layer, (5*5+1)*6*50 = 7800 weights, and 1250*26 = 32500 connections.

The fourth layer is a fully-connected layer with 100 neurons. Since it is fully-connected, each of the 100 neurons in the layer is connected to all 1250 neurons in the previous layer. There are therefore 100 neurons in it, 100*(1250+1) = 125100 weights and 100x1251 = 125100 connections.

Layer 5 is the final output layer. This layer is also a fully-connected layer with 10 units. Each of the 10 neurons in this layer is connected to all 100 neurons of the previous layer. There are 10 neurons in Layer 5, 10*(100+1) = 1010 weights and 10x101 = 1010 connections.

As you can see, although structurally simple, this Neural Network is a huge data structure.

Previous GPU Implementation

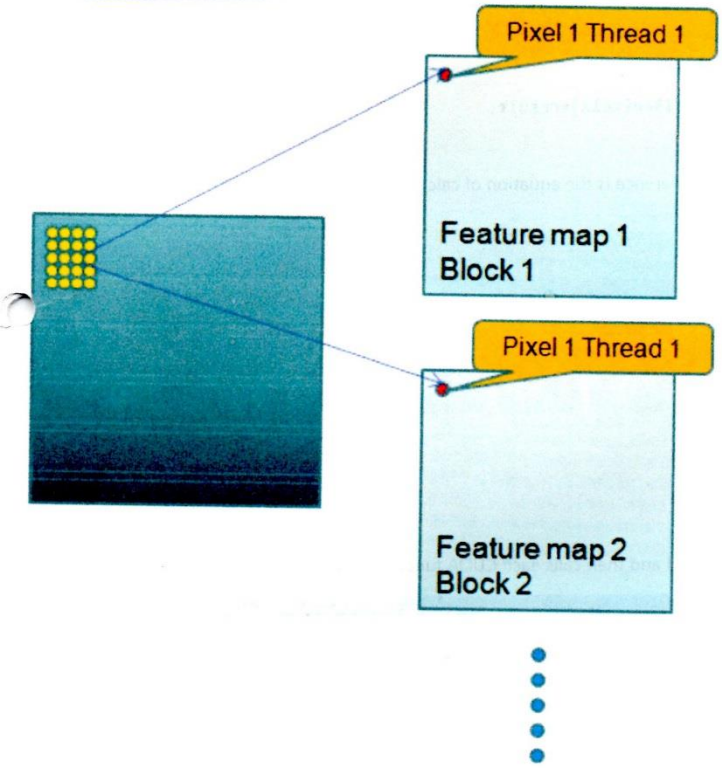
Fast Neural Network Library (FANN) has a very simple implementation of Neural Network on GPU with GLSL. Each neural is represented by a single color channel of a texture pixel. This network is very specific; neurons are ranging from 0 to 1 and have an accuracy of only 8 bits. This implementation takes the advantage of hardware accelerated dot product function to calculate neurons. Both neurons and weights are carried on texture maps.

This implementation is straightforward and easy, however limited. First, in our neural network, we require 32-bit float accuracy for each neuron. Since our network has five layers, accuracy lost at the first level could be accumulated and alter the final results. And because it is important that a handwriting recognition system should be sensitive enough to detect slight differences between different inputs, using only 8 bits to represent a neuron is unacceptable. Secondly, normal Neural Networks map neuron values to the range from 0 to 1. However, in our program, the Neural Network which is specifically designed for handwriting recognition has a special activation function mapping each neuron value to the range from -1 to 1. Therefore, if the neuron is represented by a single color value as in FANN library, our neurons will lose accuracy further. Finally, the FANN method uses a dot product to compute neurons, which is suitable for full connected Neural Networks. In our implementation, the Neural Network is partially connected. Computations performed on our Neural Network involve dot products of large vectors.

Our Implementation

Due to all the inconvenience about GLSL mentioned above, we finally choose CUDA. The reason that the Neural Network is suitable for GPU is that the training and execution of a Neural Network are two separate processes. Once properly trained, no writing access is required while using a Neural Network. Therefore there is no synchronization issue that needs to be addressed. Moreover, neurons on a same network level are completely isolated, such that neuron value computations can achieve highly parallelization.

In our code, weights for the first layer are stored as an array, and those inputs are copied to device. For each network level, there is a CUDA function handling the computation of neuron values of that level, since parallelism can only be achieved within one level and the connections are different between levels. The connections of the Neural Network are implicitly defined in CUDA functions with the equations of next level neuron computation. No explicit connection data structure exists in our code. This is one main difference between our code and the CPU version by Mike.



For example, each neuron value of the second level is a weighted sum of 25 neurons of the first level and one bias. The second neuron level is composed of 6 feature maps; each has a size of 13*13. We assign a `blockID` for each feature map and a `threadID` for each neuron on a feature map. Every feature map is handled by a block and each pixel on it is dealt with by a thread.

This is the CUDA function that computes the second network layer:

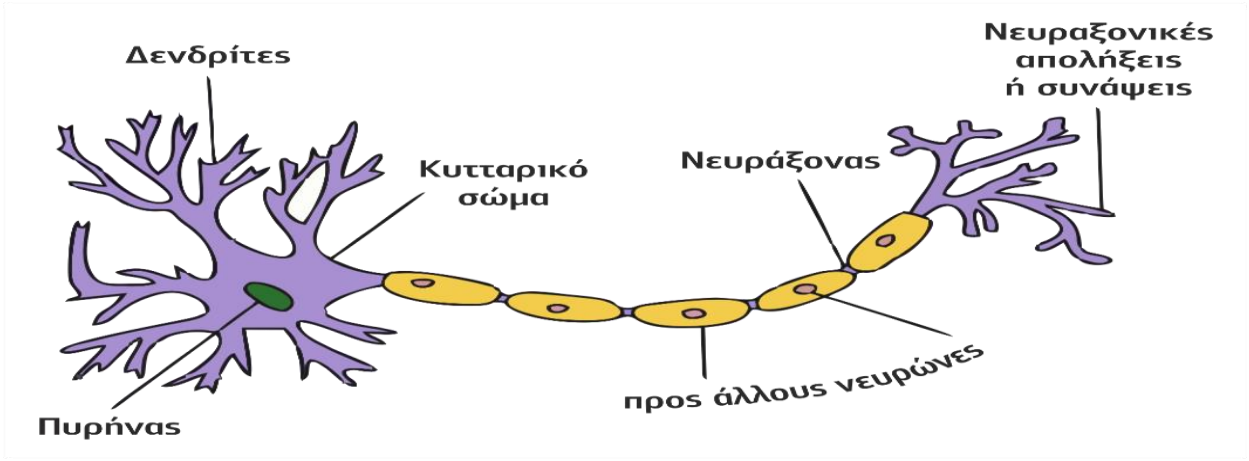
Hide Shrink Copy Code

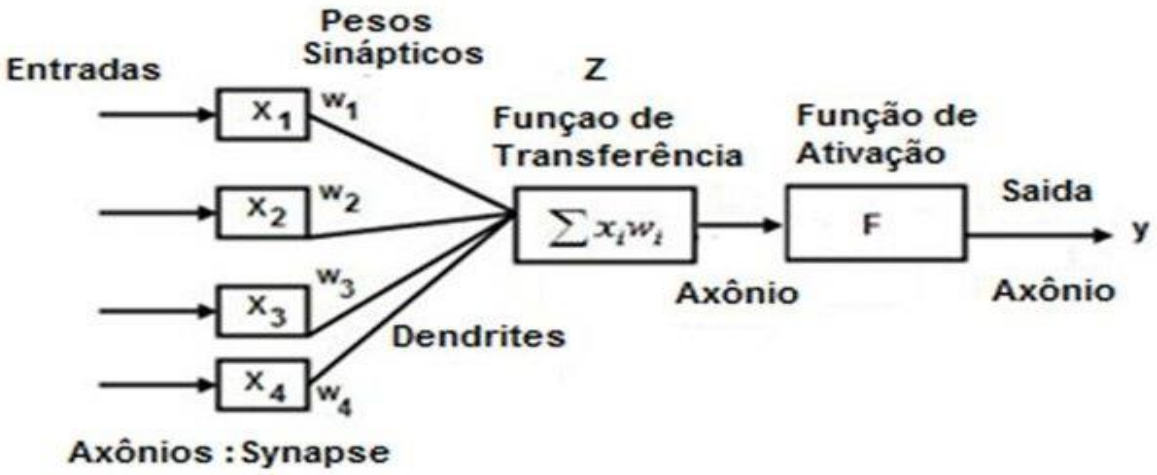
```
__global__ void executeFirstLayer
    (float *Layer1_Neurons_GPU,float *Layer1_Weights_GPU,float *Layer2_Neurons_GPU)
{
    int blockID=blockIdx.x;
    int pixelX=threadIdx.x;
    int pixelY=threadIdx.y;

    int kernelTemplate[25] = {
        0,  1,  2,  3,  4,
        29, 30, 31, 32, 33,
        58, 59, 60, 61, 62,
        87, 88, 89, 90, 91,
        116,117,118,119,120 };

    int weightBegin=blockID*26;
    int windowX=pixelX*2;
```

ΕΥΧΑΡΙΣΤΙΕΣ

Πατερα ήθελα να σουνα εδω να με καμαρώνεις
να με βλεπες αποφοιτο σχολης και να μη θυμωνεις
Να μπορούσα τον θάνατο να τονε σταματήσω
Κοντά σου αλλη μια ζωή θα ηθελα να ζήσω
Κόπο πολύ εδωσα για τουτο να το πτυχίο
μα στο τέλος πια εγινε ιστορία σαν βιβλιο
Συνάδελφους πολλούς εγνώρισα σε τουτη δα την σχολή
Ολοι τους καλά παιδιά και ολοι τους μεσα την λογική
Δάσκαλοι πολλοί και όλοι τους καλοι και μορφωμένοι
βοήθαγαν ολους μας και εμεναν ευχαριστημένοι
Μα μη ξεχάσω να ειπω και για την γραμματεία
που δεν επαψε ποτέ να δίνει πολλα αριστεια
Να μην ξεχασω ν' αναφερθω και στον Κ. Κωνσταντάρα
που με εμπιστέυτηκε πολύ και ήταν όλα μέλι γάλα

Τέλος θα ήθελα να ευχαριστήσω τον Κ. Κωνσταντάρα που με εμπιστεύτηκε για την υλοποιήση την εργασίας , καθώς και τους υπόλοιπους καθηγητές σε όλην την διάρκεια των μεταπτυχαικών μαθημάτων. θα ήθελα να ευχαριστήσω προσωπικά τον Κ. Λιοδάκη τον Κ. Κόκκινο τον Κ. Πετράκη τον Κ. Αντωνιδάκη τον Κ. Καλιακάτσο τον Κ. Βαρδιάμπαση τον Κ. Καπετανάκη τον Κ. Πετρίδη τον Κ. Δοιτσίδη τον Κ. Φουσκιτάκη τον Κ. Μπαρμπουνάκη επίσης την Κυρία Ελένη Ζυμβραγού την Κυρία Γκατζουνη Πόπη την Κ.Κατσαμάκη Τέση .Επίσης να ευχαριστήσω τον Κ. Παπαδογιάννη Νεκτάριο για την εμπιστοσύνη που μου εδειξε. Θα ήθελα επίσης να ευχαριστήσω την γραμματεία της Σχολής για όλη την βοήθεια που μου παρείχε. Επίσης θα ήθελα να αφιερώσω την εργασία μου στον Πατέρα μου απο κει που με βλέπει, στην γυναίκα μου και τα παιδιά μου.





ΠΗΓΗ:

https://www.researchgate.net/publication/285420286_Aplicacoes_das_Redes_Neurais_na_Logistica_Interna_Estudo_de_Casos_nas_Empresas_do_Polo_Industrial_de_Manaus/figures?lo=1

ΠΑΡΑΡΤΗΜΑ – ΠΑΡΑΛΕΙΨΕΙΣ

Κεφάλαιο 3. Μεθοδοι και Τρόποι Εκπαίδευσης

3.7 Τροποι εκπαίδευσης ενός δικτύου MLP

γ) incremental learning (επαυξητική μάθηση) είναι μια μέθοδος στην οποία τα δεδομένα εισόδου συνεχώς χρησιμοποιούνται για να επεκτείνουνε την υπάρχουσα γνώση του μοντέλου για να εκπαιδεύσει επιπλέον το μοντέλο. Αναπαριστά μια δυναμική τεχνι-κή επιβλεπόμενης μάθησης και μη επιβλεπόμενης μάθησης και μπορεί να εφαρμοστεί όταν τα εκπαιδευόμενα δεδομένα γίνονται βαθμιαία διαθέσιμα στον χρόνο ή στο μέγεθος εκτός ορίων του συστήματος. Σκοπός του είναι να προσαρμοστεί στα καινούργια δεδομένα χωρίς να ξεχάσει την υπάρχουσα γνώση , δεν ξανα-εκπαιδεύει το δίκτυο. Αυτή η τεχνική συνήθως εφαρμόζεται σε μεγάλες ροές δεδομένων ,θέματα διευθυνσιοδότησης σε διαθεσιμότητα δεδομένων

Κεφάλαιο 1

1.2 Πλεονεκτήματα Τεχνητών Νευρωνικών Δικτύων . Σελίδα 9

Επίσης τα νευρωνικά δίκτυα δεν λειτουργούν σειριακά αλλά πιο πολύ με παράλληλο τρόπο και αυτό διότι μια εργασία μοιράζεται στα διάφορα τμήματα του δικτύου και σε όλους τους επιμέρους νευρώνες .[5] Θα μπορούσαμε να πούμε ότι τα νευρωνικά δίκτυα είναι συστήματα παράλληλων κατανεμημένων διεργασιών.[5] Αρα έχουμε μεγάλες ταχύτητες και έχουμε ταυτόχρονα πολλούς επεξεργαστές στην διάθεσή μας.[5]

Ομως η αρχιτεκτονική των νευρωνικών δικτύων διαφέρει απο αυτήν των παράλληλων επεξεργαστών στο ότι οι απλοί επεξεργαστές των νευρωνικών δικτύων έχουν μεγάλο αριθμό διασυνδέσεων και ο οποίος είναι αρκετά

μεγαλύτερος απο τον αριθμό των νευρώνων και αυτό διότι ο κάθε νευρώνας έχει πολλές συνδέσεις .[5] Απο άλλη όμως οι παράλληλοι επεξεργαστές είναι συνήθως περισσότεροι απο τις διασυνδέσεις μεταξύ τους και βέβαια ως πρός την πολυπλοκότητα τους ακολουθούν το πρότυπο δηλαδή τηνμηχανή Von Neumann. [5]

Τα νευρωνικά ξέρουν μόνο να αθροίζουν τα σήματα εισόδου και να τροποποιούν τα βάρη των διασυνδέσεων. Επίσης οι νευρώνες λειτουργούν ανεξάρτητα ο ένας απο το άλλον και δεν χρειάζονται συγχρονισμό.[5]

Οπότε έχουν την ευρωστία και ανοχή σε σφάλματα.[5]

ΠΑΡΑΡΤΗΜΑ – ΠΑΡΑΛΕΙΨΕΙΣ

THEORY FROM CHALMERS Implementation of Deep Feedforward Neural Network with CUDA backend for Efficient and Accurate Natural Image Classification Convolutional Neural Network Applications Master's thesis in Complex Adaptive Systems

AUGUST VON HACHT Department of Physics CHALMERS UNIVERSITY OF TECHNOLOGY Gothenburg, Sweden 2017

Convolutional layer

A convolutional layer is also parametrized by weights and a bias but introduces additional parameters through weight sharing. The weight sharing principle is shown in a one dimensional case using a color scheme to show case the same weight occurrences. [16]

layer l−1 layer l

Unlike the fully connected layer where each neuron connects to the next layer via a unique weight, the convolution layer use a set of weights called filters F which is applied multiple times onto various spatial locations of the input I. [16] As a result, the number of weights in the one dimensional case remains constantly equal to the filter size instead of O(nm), where n is the number of neurons in layer l and m is the number of neuron is layer l−1, as in the fully connected case. [16] The weight sharing introduces additional parameters called stride and padding. The stride determines the step size by which the filters moves over the input, e.g. in the diagram the stride equals to one. [16] The padding is used to fill out the input's spatial domain such that the convolution is defined at the boarders for a certain filter size andstride. These parameters are not learnable and instead used to define the spatial output domain which is related to the spatial input domain through

where i denote the domain by which the filters acts, e.g. width or height of an image. Note that the output domain before the floor function is preferably integer valued. It is also preferred to have nl i = nl−1 i ∀i since this makes the filters occur at the same spatial locations in both the backward and forward pass.[16]

The two operations performed in a forward and backward pass of a convolutional layer are the discrete cross-correlation [16]

Ccross jk =X j0 X k0 Ij+j0,k+k0Fj0k0 (4)

and discrete convolution

Cconv jk =X j0 X k0 Ij−j0,k−k0Fj0k0 (5)

which are related through a flipping operation applied onto the filters. [16] During a forward pass through the layer each output activation is formed by

aljk = fwljk ∗al−1 jk +bljk(6)

= f□ □X j0 X k0 wl j0k0al−1 j−j0,k−k0 +bljk□ □ (7)

The input to the activation function f is again an affine transformation denoted by

zljk =X j0 X k0 wl j0k0al−1 j−j0,k−k0 +bljk (8)

Now given an error ∆zljk this is propagated backwards according to

δl−1 jk = δljk ∗rot180◦(wljk)f0(zljk) (9)

Looking back at the one dimensional case in the diagram, the color scheme shows that when forward propagating a value in layer l−1 to layer l this requires traversing the weight connections according to red, green and blue order. [16] When backward propagating an error in layer l to layer l −1 this requires traversing the weights according to blue, green and red order. This is the reason for rotating the filters in eq. (9). [16] Given an error at each neuron in a convolutional layer the weights and bias gradients are

∂L ∂wljk = δljk ∗f(rot180◦(zl−1 jk)) (10)

∂L ∂bljk = δljk (11)

Maxpooling layer

A maxpooling layer summarizes the output from a convolutional layer by grouping output regions into a single representative candidate. [16] This layer therefore does not require any trainable parameters but has to describe the choice of the region used

by the max operator.[16] A region is a two dimensional rectangular neighborhood with side lengths rx and ry and when applied to the input space encloses some subset of activation values.[16] Multiple regions are spread either jointly or disjointly across the input space which is decided by a striding and padding parameter. If the stride sizealong a spatial domain is larger than the rectangular sidelengthin thatdomain then the regions won't overlap otherwise they will. [16] The output's spatial domain is obtained through eq. (2.23), but with fi → ri. Using a one dimensional case with stride equals to two and padding zero results in the following diagram

a1

a2

a3

a4

max{a1,a2} max{a3,a4}

layer l−1 layer l

The particular form of weight sharing used in the convolutional layer has the property of being equivariant to translation. [16] Equivariance refers to the property of changinginputresultsintheoutputchangingsimilarly. [16] This means that first translating and then convolving is equal to first convolving and then translating,assuming the translation keeps the object within the input domain,

$$[[L_t F] * I](x) = \sum_y F(y-t) I(x-y) \quad (12)$$

$$= \sum_{y_0} F(y_0) I(x-(y_0+t)) \quad (13)$$

$$= \sum_{y_0} F(y_0) I(x-y_0-t) \quad (14)$$

$$= [L_t [F * I]](x) \quad (15)$$

By summarizing the output from a convolutional layer using a maxpooling layer the resulting representation of the activations is made approximately invariant to small uniform translations in the input. [16] If a small uniform translation in the input occurs then this is canceled out by the max operator since the winning activations maintain their relative positions. For image

classification tasks this makes such a layer combination invariant to where in the image an object of the same class is positioned spatially.[16]

The forward pass through a maxpooling layer is

aljk =maxnal−1 jsy−py,ksx−px,al−1 jsy−py,ksx+1−px,...,al−1 jsy+ry−py,ksx+rx−pxo(16)

where sx,sy are the striding parameters and px,py are the padding parameters of respective domains.[16]

Since there are no trainable parameters the errors propagated backwards from the next layer is simply back propagated to the previous layer as is. However, for each region only the winning activation obtains the error since the other activations did not participate in the computation of the output mapping [16]

δl−1 jk =(δljk if al−1 jk was max 0 otherwise (17)

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)



Bitwise XOR Operation of 12 and 25

  00001100

^ 00011001

  ________

  00010101  = 21 (In decimal)
```

Example #3: Bitwise XOR

```c
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
    printf("Output = %d", a^b);
    return 0;
}
```

Output

```
Output = 21
```

```
clear;
clc;
%Getting weights and threshold value
disp('Enter weights');
w11=input('Weight w11=');
w12=input('weight w12=');
w21=input('Weight w21=');
w22=input('weight w22=');
v1=input('weight v1=');
v2=input('weight v2=');
disp('Enter Threshold Value');
theta=input('theta=');
x1=[0 0 1 1];
x2=[0 1 0 1];
z=[0 1 1 0];
con=1;
while con
zin1=x1*w11+x2*w21;
zin2=x1*w21+x2*w22;
for i=1:4
if zin1(i)>=theta
y1(i)=1;
else
y1(i)=0;
end
if zin2(i)>=theta
y2(i)=1;
else
y2(i)=0;
end
end
yin=y1*v1+y2*v2;
for i=1:4
if yin(i)>=theta;
y(i)=1;
else
y(i)=0;
end
```

```
end
disp('Output of Net');
disp(y);
if y==z
con=0;
else
disp('Net is not learning enter another set of weights and
Threshold value');
w11=input('Weight w11=');
w12=input('weight w12=');
w21=input('Weight w21=');
w22=input('weight w22=');
v1=input('weight v1=');
v2=input('weight v2=');
theta=input('theta=');
end
end
disp('McCulloch-Pitts Net for XOR function');
disp('Weights of Neuron Z1');
disp(w11);
disp(w21);
disp('weights of Neuron Z2');
disp(w12);
disp(w22);
disp('weights of Neuron Y');
disp(v1);
disp(v2);
disp('Threshold value');
disp(theta);
```

```
#include <iostream>
#include <ctime>
#include <stdlib.h>
#include <valarray>
#define POPSIZE 100
using namespace std;
float randfloat();
float Sigmoid(float A);
class Perceptron{
public:
   Perceptron()
   {
      weights[0] = randfloat();
      weights[1] = randfloat();
      weights[2] = randfloat();
      weights[3] = randfloat();
      weights[4] = randfloat();
      weights[5] = randfloat();
      weights[6] = randfloat();
      weights[7] = randfloat();
      weights[8] = randfloat();
      CalculateError();
   }
   float FeedForward(int A, int B)
   {
      float Sum1 = A*weights[0] + B*weights[1] + bias*weights[4];
      float Sum2 = A*weights[2] + B*weights[3] + bias*weights[5];
      float Sum3 = Sigmoid(Sum1)*weights[6]
                 + Sigmoid(Sum2)*weights[7]
                 + bias*weights[8];
      return Sigmoid(Sum3);

   }
   void PrintStats()
   {
      cout << "weights =" << weights[0] << " " << weights[1] << " " << weights[2];
      cout << "\nerror= " << normalisedError;
   }
   void CalculateError()
   {
      error = abs(0-FeedForward(1,1)) +
      abs(1-FeedForward(1,0)) +
      abs(1-FeedForward(0,1)) +
      abs(0-FeedForward(0,0));
      normalisedError = error*10;
   }
   float GetNormalisedError()
   {
      return normalisedError;
   }
   void SetFitnessScore(int i)
   {
   fitnessScore = i;
   }
   int GetFitnessScore()
   {
      return fitnessScore;
   }
friend void CrossOver(Perceptron A, Perceptron B, Perceptron &C);
friend void TransferGenes(Perceptron A, Perceptron &B);
```

```
friend int ChooseFittestSpecies(Perceptron pop[POPSIZE]);
private:
    float weights[9];
    int bias = 1;
    float error =0;
    int normalisedError;
    int fitnessScore;
};

int ChooseParent(Perceptron pop[POPSIZE]);
int ChooseFittestSpecies(Perceptron pop[POPSIZE]);
int ChooseWorstSpecies(Perceptron pop[POPSIZE]);
void Reproduce(Perceptron pop[POPSIZE], Perceptron C[POPSIZE]);



int main()
{
    srand(time(0));
    Perceptron pop[POPSIZE];

    Perceptron Children[POPSIZE];

for(int x =0 ;  x < 100 ; x++){
    Reproduce(pop, Children);
    for(int i = 0 ; i < POPSIZE ; i ++)
    {
        TransferGenes(Children[i], pop[i]);
        pop[i].CalculateError();
    }
}

int alphaSpecies = ChooseFittestSpecies(pop);

cout << "Most fit species is: " << alphaSpecies << endl;
    cout << "inputs:" << endl;
    cout << "1 1 : " << pop[alphaSpecies].FeedForward(1,1) << endl;
    cout << "0 1 : " << pop[alphaSpecies].FeedForward(0,1) << endl;
    cout << "1 0 : " << pop[alphaSpecies].FeedForward(1,0) << endl;
    cout << "0 0 : " << pop[alphaSpecies].FeedForward(0,0) << endl;

    return 0;
}

void TransferGenes(Perceptron A, Perceptron &B)
{
    for(int i = 0 ; i < 9 ; i++)
    {
        B.weights[i] = A.weights[i];
    }
}

void Reproduce(Perceptron pop[POPSIZE], Perceptron C[POPSIZE])
{
    int ParentA, ParentB;
    for(int i = 0; i < POPSIZE ;  i++)
    {
        ParentA = ChooseParent(pop);
        ParentB = ChooseParent(pop);
        CrossOver(pop[ParentA], pop[ParentB], C[i]);
```

```
    }
}

void CrossOver(Perceptron A, Perceptron B, Perceptron &C)
{
    int random;
    for(int i = 0 ; i < 9 ; i ++)
    {
        random = rand()%2;
        if(random)
        {
            C.weights[i] = A.weights[i];
        }else{
            C.weights[i] = B.weights[i];
        }
    }
}

int ChooseParent(Perceptron pop[POPSIZE])
{
    int GreatestError = pop[0].GetNormalisedError();
    for(int i = 1; i < POPSIZE ; i++)
    {
        if(pop[i].GetNormalisedError() > GreatestError)
        {
            GreatestError = pop[i].GetNormalisedError();
        }
    }

    int totalFitness=0;

    for(int i =0 ; i  < POPSIZE ;  i++)
    {
        pop[i].SetFitnessScore(GreatestError  + 1 - pop[i].GetNormalisedError());
        totalFitness = totalFitness + pop[i].GetFitnessScore();
    }

    int random = rand()%totalFitness +1;

    int ParentSelector = 0;
    int ParentIndex;
    for(int i=0;ParentSelector<random;i++)
    {
        ParentSelector = ParentSelector + pop[i].GetFitnessScore();
        ParentIndex = i;
    }

    return ParentIndex;
}

float randfloat()
{
    float f = static_cast <float> (rand()) / (static_cast <float> (RAND_MAX/10));
    int Pos_or_Neg = rand()%2;
    if(Pos_or_Neg)
    {
        return f;
    }else{
    return -f;
    }
```

```
}
float Sigmoid(float A)
{
   return 1/(1+exp(-2*A));
}
int ChooseFittestSpecies(Perceptron pop[POPSIZE])
{
   int MostFitParent = 0;

   for(int i = 0;  i  < POPSIZE ;  i++)
   {
      if(pop[i].error < pop[MostFitParent].error)
      {
         MostFitParent = i;
      }
   }
   return MostFitParent;
}
```

Convolutional Networks and Applications in Vision

Yann LeCun, Koray Kavukcuoglu and Cl´ement Farabet Computer Science Department, Courant Institute of Mathematical Sciences, New York University {yann,koray,cfarabet}@cs.nyu.edu

Abstract—Intelligent tasks, such as visual perception, auditory perception, and language understanding require the construction of good internal representations of the world (or ”features”), which must be invariant to irrelevant variations of the input while, preserving relevant information. A major question for Machine Learning is how to learn such good features automatically. Convolutional Networks (ConvNets) are a biologicallyinspired trainable architecture that can learn invariant features. Each stage in a ConvNets is composed of a filter bank, some non-linearities, and feature pooling layers. With multiple stages, a ConvNet can learn multi-level hierarchies of features. While ConvNets have been successfully deployed in many commercial applications from OCR to video surveillance, they require large amounts of labeled training samples. We describe new unsupervised learning algorithms, and new non-linear stages that allow ConvNets to be trained with very few labeled samples. Applications to visual object recognition and vision navigation for off-road mobile robots are described. I. LEARNING INTERNAL REPRESENTATIONS One of the key questions of Vision Science (natural and artificial) is how to produce good internal representations of the visual world. What sort of internal representation would allow an artificial vision system to detect and classify objects into categories, independently of pose, scale, illumination, conformation, and clutter? More interestingly, how could an artificial vision system learn appropriate internal representations automatically, the way animals and human seem to learn by simply looking at the world? In the time-honored approach to computer vision (and to pattern recognition in general), the question is avoided: internal representations are produced by a hand-crafted feature extractor, whose output is fed to a trainable classifier. While the issue of learning features has been a topic of interest for many years, considerable progress has been achieved in the last few years with the development of so-called deep learning methods. Good internal representations are hierarchical. In vision, pixels are assembled into edglets, edglets into motifs, motifs into parts, parts into objects, and objects into scenes. This suggests that recognition architectures for vision (and for other modalities such as audio and natural language) should have multiple trainable stages stacked on top of each other, one for each level in the feature hierarchy. This raises two new questions: what to put in each stage? and how to train such deep, multi-stage architectures? Convolutional Networks (ConvNets) are an answer to the first question. Until recently, the answer to the second question was to use gradient-based supervised learning, but recent research in deep learning has produced a number of unsupervised methods which greatly reduce the need for labeled samples. Convolutional Networks Convolutional Networks [1], [2] are trainable multistage architectures composed of multiple stages. The input and output of each stage are sets of arrays called feature maps. For example, if the input is a color image, each feature map would be a 2D array containing a color channel of the input image (for an audio input each feature map would be a 1D array, and for a video or volumetric image, it would be a 3D array). At the output, each feature map represents a particular feature

Fig. 1. A typical ConvNet architecture with two feature stages extracted at all locations on the input. Each stage is composed of three layers: a filter bank layer, a non-linearity layer, and a feature pooling layer. A typical ConvNet is composed of one, two or three such 3-layer stages, followed by a classification module. Each layer type is now described for the case of image recognition. Filter Bank Layer - F: the input is a 3D array with n1 2D feature maps of size n2×n3. Each component is denoted xijk, and each feature map is denoted xi. The output is also a 3D array, y composed of m1 feature maps of size m2 × m3. A trainable filter (kernel) kij in the filter bank has size l1 × l2 and connects input feature map xi to output feature map yj. The module computes yj = bj + Pi kij ∗ xi where ∗ is the 2D discrete convolution operator and bj is a trainable bias parameter. Each filter detects a particular feature at every location on the input. Hence spatially translating the input of a feature detection layer will translate the output but leave it otherwise unchanged. Non-Linearity Layer: In traditional ConvNets this simply consists in a pointwise tanh() sigmoid function applied to each site (ijk). However, recent implementations have used more sophisticated non-linearities. A useful one for natural image recognition is the rectified sigmoid Rabs: abs(gi.tanh()) where gi is a trainable gain parameter. The rectified sigmoid is sometimes followed by a subtractive and divisive

local normalization N, which enforces local competition between adjacent features in a feature map, and between features at the same spatial location. The subtractive normalization operation for a given site xijk computes: vijk = xijk −Pipq wpq.xi,j+p,k+q, where wpq is a normalized truncated Gaussian weighting window (typically of size 9x9). The divisive normalization computes yijk = vijk/max(mean(σjk),σjk) where σjk = (Pipq wpq.v2 i,j+p,k+q)1/2. The local contrast normalization layer is inspired by visual neuroscience models [3], [4]. Feature Pooling Layer: This layer treats each feature map separately. In its simplest instance, called PA, it computes the average values over a neighborhood in each feature map. The neighborhoods are stepped by a stride larger than 1 (but smaller than or equal the pooling neighborhood). This results in a reduced-resolution output feature map which is robust to small variations in the location of features in the previous layer. The average operation is sometimes replaced by a max PM. Traditional ConvNets use a pointwise tanh() after the pooling layer, but more recent models do not. Some ConvNets dispense with the separate pooling layer entirely, but use strides larger than one in the filter bank layer to reduce

Fig. 2. An example of feature extraction stage of the type F −Rabs−N−PA. An input image (or a feature map) is passed through a filter bank, followed by abs(gi.tanh()), local subtractive and divisive contrast normalization, and spatial pooling/sub-sampling.

the resolution [5], [6]. In some recent versions of ConvNets, the pooling also pools similar feature at the same location, in addition to the same feature at nearby locations [7]. Supervised training is performed using a form of stochastic gradient descent to minimize the discrepancy between the desired output and the actual output of the network. All the coefficient of all the filters in all the layers are updated simultaneously by the learning procedure. The gradients are computed with the back-propagation method. Details of the procedure are given in [2], and methods for efficient training are detailed in [8].

History and Applications ConvNets can be seen as a representatives of a wide class of models that we will call Multi-Stage Hubel-Wiesel Architectures. The idea is rooted in Hubel and Wiesel's classic 1962 work on the cat's primary visual cortex. It identified orientation-selective simple cells with local receptive fields, whose role is similar to the ConvNets filter bank layers, and complex cells, whose role is similar to the pooling layers. The first such model to be simulated on a computer was Fukushima's Neocognitron [9], which used a layer-wise, unsupervised competitive learning algorithm for the filter banks, and a separately-trained supervised linear classifier for the output layer. The innovation in [5], [1] was to simplify the architecture and to use the back-propagation algorithm to train the entire system in a supervised fashion. The approach was very successful for such tasks as OCR and handwriting recognition. An operational bank check reading system built around ConvNets was developed at AT&T in the early 1990's [2]. It was first deployed commercially in 1993, running on a DSP board in check-reading ATM machines in Europe and the US, and was deployed in large bank check reading machines in 1996. By the late 90's it was reading over 10% of all the checks in the US. This motivated Microsoft to deploy ConvNets in a number of OCR and handwriting recognition systems [6], [10], [11] including for Arabic [12] and Chinese characters [13]. Supervised ConvNets have also been used for object detection in images, including faces with record accuracy and real-time performance [14], [15], [16], [17], Google recently deployed a ConvNet to detect faces and license plate in StreetView images so as to protect privacy [18]. NEC has deployed ConvNet-based system in Japan for tracking customers in supermarket and recognizing their gender and age. Vident Technologies has developed a ConvNet-based video surveillance system deployed in several airports in the US. France T´ el´ecom has deployed ConvNetbased face detection systems for video-conference and other systems [15]. Other experimental detection applications include hands/gesture [19], logos and text [20]. A big advantage of ConvNets for detection is their computational efficiency: even though the system is trained on small windows, it suffices to extend the convolutions to the size of the input image and replicate the output layer to compute detections at every location. Supervised ConvNets have also been used for visionbased obstacle avoidance for off-road mobile robots [21]. Two

participants in the recent DARPA-sponsored LAGR program on vision-based navigation for off-road robots used ConvNets for long-range obstacle detection [22], [23]. In [22], the system is pre-trained off-line using a combination of unsupervised learning (as described in section II) and supervised learning. It is then adapted on-line, as the robot runs, using labels provided by a short-range stereovision system (see videos at http://www.cs.nyu.edu/ yann/research/lagr). Interesting new applications include image restoration [24] and image segmentation, particularly for biological images [25]. The big advantage over MRFs is the ability

to take a large context window into account. Stunning results were obtained at MIT for reconstructing neuronal circuits from an stack of brain slice images a few nanometer thick. [26]. Over the years, other instances of the Multi-Stage HubelWiesel Architecture have appeared that are in the tradition of the Neocognitron: unlike supervised ConvNets, they use a combination of hand-crafting, and simple unsupervised methods to design the filter banks. Notable examples include Mozer's visual models [27], and the so-called HMAX family of models from T. Poggio's lab at MIT [28], [29], which uses hard-wired Gabor filters in the first stage, and a simple unsupervised random template selection algorithm for the second stage. All stages use point-wise non-linearities and max pooling. From the same institute, Pinto et al. [4] have identified the most appropriate non-linearities and normalizations by running systematic experiments with a a single-stage architecture using GPU-based parallel hardware. II. UNSUPERVISED LEARNING OF CONVNETS Training deep, multi-stage architectures using supervised gradient back propagation requires many labeled samples. However in many problems labeled data is scarce whereas unlabeled data is abundant. Recent research in deep learning [30], [31], [32] has shown that unsupervised learning can be used to train each stage one after the other using only unlabeled data, reducing the requirement for labeled samples significantly. In [33], using abs and normalization non-linearities, unsupervised pre-training, and supervised global refinement has been shown to yield excellent performance on the Caltech101 dataset with only 30 training samples per category (more on this below). In [34], good accuracy was obtained on the same set using a very different unsupervised method based on sparse Restricted Boltzmann Machines. Several works at NEC have also shown that using auxiliary tasks [35], [36] helps regularizing the system and produces excellent performance.

Unsupervised Training with Predictive Sparse Decomposition The unsupervised method we propose, to learn the filter coefficients in the filter bank layers, is called Predictive Sparse Decomposition (PSD) [37]. Similar to the well-known sparse coding algorithms [38], inputs are approximated as a sparse linear combination of dictionary elements. In conventional sparse coding for any given input X, one needs to run an expensive optimization algorithm to find $Z*$ (the "basis pursuit" problem). PSD trains a feed-forward regressor (or encoder) C(X,K) to quickly approximate the sparse solution $Z*$. During training, the feature vector $Z*$ is obtained by minimizing: $E(Z,W,K) = kX - WZk2\ 2 + \lambda kZk1 + kZ - C(X,K)k2\ 2$ where W is the matrix whose columns are the dictionnary elements and K are the filters. For each training sample X, one first finds $Z*$ that minimizes E, then W and K are

TABLE I AVERAGE RECOGNITION RATES ON CALTECH-101. Rabs − N − PA Rabs − PA N − PM PA U+ 65.5% 60.5% 61.0% 32.0% R+ 64.7% 59.5% 60.0% 29.7% U 63.7% 46.7% 56.0% 9.1% R 62.9% 33.7% 37.6% 8.8%

adjusted by stochastic gradient descent to lower E. Once training is complete, the feature vector for a given input is simply obtained with $Z* = C(X,K)$, hence the process is extremely fast (feed-forward). Results on Object Recognition In this section, various architectures and training procedures are compared to determine which non-linearities are preferable, and which training protocol makes a difference. Generic Object Recognition using Caltech 101 Dataset: We use a two-stage system where, the first stage is composed of an F layer with 64 filters of size 9×9, followed by different combinations of non-linearities and pooling. The second-stage feature extractor is fed with the output of the first stage and extracts 256 output features maps, each of which combines a random subset of 16 feature maps from the previous stage using 9 × 9 kernels. Hence the total number of convolution kernels is 256 × 16 = 4096. Table I summarizes the results for the experiments, where U and R denotes unsupervised pre-training and random initialization respectively, and + denotes supervised finetuning of the whole system. 1. Excellent accuracy of 65.5% is obtained using unsupervised pre-training and supervised refinement with abs and normalization non-linearities. The result is on par with the popular model based on SIFT and pyramid match kernel SVM [39]. It is clear that abs and normalization are cruciala for achieving good performance. This is an extremely important fact for users of convolutional networks, which traditionally only use tanh(). 2. Astonishingly, random filters without any filter learning whatsoever achieve decent performance (62.9% for R), as long as abs and normalization are present (Rabs − N − PA). A more detailed study on this particular case can be found in [33]. 3. Comparing experiments from rows R vs R+, U vs U+, we see that supervised fine tuning consistently improves the performance, particularly with weak non-linearities. 4. It seems that unsupervised pre-training (U, U+) is crucial when newly proposed non-linearities are not in place.

Handwritten Digit Classification using MNIST Dataset: Using the evidence gathered in previous experiments, we used a two-stage system with a two-layer fully-connected classifier. The two convolutional stages were pre-trained unsupervised, and refined supervised. An error rate of 0.53% was achieved ont he test set. To our knowledge, this is the lowest error rate ever reported on the original MNIST dataset, without distortions or preprocessing. The best previously reported error rate was 0.60% [32]. Connection with Other Approaches in Object Recognition Many recent successful object recognition systems can also be seen as single or multi-layer feature extraction systems followed by a classifier. Most common feature extraction systems like SIFT [40], HoG [41] are composed of filterbanks (oriented edge detectors at multiple scales) followed by non-linearities (winner take all) and pooling (histogramming). A Pyramid

Match Kernel (PMK) SVM [39] classifer can also be seen as another layer of feature extraction since it performs a K-means based feature extraction followed by local histogramming. III. HARDWARE AND SOFTWARE IMPLEMENTATIONS Implementing ConvNets in software is best achieved using the modular, object-oriented approach suggested in [2]. Each basic module (convolution, pooling, etc) is implemented as a class with three member functions module.fprop(input,output), which computes the output from the input, module.bprop(input,output), which back-propagates gradients from the outputs back to the inputs and the internal trainable parameters, and optionally module.bbprop(input,output), which may back-propagate second diagonal derivatives for the implementation of second-order optimization algorithms [8]. Several software implementations of ConvNets are built around this concept, and have four basic capabilities: 1. a flexible multi-dimensional array library with basic operations such as dot products, and convolutions, 2. a class hierarchy of basic learning machine building blocs (e.g. multiple convolutions non-linear transforms, cost functions, ...), 3. a set of classes for energy-based inference [42], gradient-based optimization, and performance measurement. Three available ConvNet implementations use this concept. The first one is part of the Lush system, a Lisp dialect with an interpreter and compiler with an easy interface to C [43] . The second one is EBlearn, a C++ machine learning library with class hierarchy to the Lush implementation [44]. Third is Torch-5 [45] a C library with an interpreter front end based on Lua. All three systems come with facilities to manipulate large datasets, images, and videos. The first hardware implementations of ConvNets date back to the early 90's with Bell Labs' ANNA chip, a mixed analogdigital processor that could compute 64 simultaneous 8 × 8 convolutions at a peak rate of 4.109 multiply-accumulate operations per second [46], [47], with 4 bit resolution on the states and 6 bits on the weights. More recently, a group from the Canon corporation developed a prototype ConvNet chip for low-power intelligent cameras [48]. Some current approaches rely on Addressed-Event Representation (AER) convolvers, which present the advantage of not requiring multipliers to compute the convolutions. CAVIAR is the leading such project, with a performance of 12G connections/sec [49]. FPGA implementations of ConvNets appeared in the mid-90's with [50], which used low-accuracy arithmetic to avoid implementing full-fledged multipliers. Fortunately, recent DSP-oriented FPGAs include large numbers of hardwired MAC units, which allow extremely fast and low power implementations of ConvNets. The CNP developed in our group [51] achieves 10GOPS for 7x7 kernels, with an architecture that implements entire ConvNets, including pre/postprocessing, and is entirely programmable. An actual face detection application was demonstrated on this system, achieving 10fps on VGA images [52]. IV. CONCLUSION The Convolutional Network architecture is a remarkably versatile, yet conceptually simple paradigm that can be applied to a wide spectrum of perceptual tasks. While traditional ConvNet trained with supervised learning are very effective, training them require a large number of labeled training samples. We have shown that using simple architectural tricks such as rectification and contrast normalization, and using

unsupervised pre-training of each filter bank, the need for labeled samples is considerably reduced. Because of their applicability to a wide range of tasks, and because of their relatively uniform architecture, ConvNets are perfect candidates for hardware implementations, and embedded applications, as demonstrated by the increasing amount of work in this area. We expect to see many new embedded vision systems based on ConvNets in the next few years. Despite the recent progress in deep learning, one of the major challenges of computer vision, machine learning, and AI in general in the next decade will be to devise methods that can automatically learn good features hierarchies from unlabeled and labeled data in an integrated fashion. Current and future research will focus on performing unsupervised learning on multiple stages simultaneously, on the integration of unsupervised and unsupervised learning, and on using the feed-back path implemented by the decoders to perform visual inference, such as pattern completion and disambiguation.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in NIPS’89. [2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, 1998. [3] S. Lyu and E. P. Simoncelli, “Nonlinear image representation using divisive normalization,” in CVPR, 2008. [4] N. Pinto, D. D. Cox, and J. J. DiCarlo, “Why is real-world visual object recognition hard?” PLoS Comput Biol, vol. 4, no. 1, p. e27, 01 2008. [5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” Neural Computation, 1989. [6] Y. Simard, Patrice, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in ICDAR’03. [7] K. Kavukcuoglu, M. Ranzato, R. Fergus, and Y. LeCun, “Learning invariant features through topographic filter maps,” in CVPR’09. [8] Y. LeCun, L. Bottou, G. Orr, and K. Muller, “Efficient backprop,” in Neural Networks: Tricks of the trade, 1998. [9] K. Fukushima and S. Miyake, “Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position,” Pattern Recognition, vol. 15, no. 6, pp. 455–469, 1982. [10] K. Chellapilla, M. Shilman, and P. Simard, “Optimally combining a cascade of classifiers,” in Proc. of Document Recognition and Retrieval 13, Electronic Imaging, 6067, 2006. [11] K. Chellapilla, S. Puri, and P. Simard, “High performance convolutional neural networks for document processing,” in IWFHR’06. [12] A. Abdulkader, “A two-tier approach for arabic offline handwriting recognition,” in IWFHR’06. [13] K. Chellapilla and P. Simard, “A new radical based approach to offline handwritten east-asian character recognition,” in IWFHR’06. [14] R. Vaillant, C. Monrocq, and Y. LeCun, “Original approach for the localisation of objects in images,” IEE Proc on Vision, Image, and Signal Processing, vol. 141, no. 4, pp. 245–250, August 1994. [15] C. Garcia and M. Delakis, “Convolutional face finder: A neural architecture for fast and robust face detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004. [16] M. Osadchy, Y. LeCun, and M. Miller, “Synergistic face detection and pose estimation with energy-based models,” Journal of Machine Learning Research, vol. 8, pp. 1197–1215, May 2007. [17] F. Nasse, C. Thurau, and G. A. Fink, “Face detection using gpu-based convolutional neural networks,” pp. 83–90, 2009. [18] A. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, and L. Vincent, “Large-scale privacy protection in street-level imagery,” in ICCV’09. [19] S. Nowlan and J. Platt, “A convolutional neural network hand tracker.” San Mateo, CA: Morgan Kaufmann, 1995, pp. 901–908. [20] M. Delakis and C. Garcia, “Text detection with convolutional neural networks,” in International Conference on Computer Vision Theory and Applications (VISAPP 2008), 2008. [21] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, “Off-road obstacle avoidance through end-to-end learning,” in Advances in Neural Information Processing Systems (NIPS 2005). MIT Press, 2005. [22] R. Hadsell, P. Sermanet, M. Scoffier, A. Erkan, K. Kavackuoglu, U. Muller, and Y. LeCun, “Learning long-range vision for autonomous off-road driving,” Journal of Field Robotics, vol. 26, no. 2, pp. 120–144, February 2009. [23] M. Happold and M. Ollis, “Using learned features from 3d data for robot navigation,” 2007.

[24] V. Jain and H. S. Seung, “Natural image denoising with convolutional networks,” in Advances in Neural Information Processing Systems 21 (NIPS 2008). MIT Press, 2008. [25] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. Barbano, “Toward automatic phenotyping of developing embryos from videos,” IEEE Transactions on Image Processing, 2005, special issue on Molecular and Cellular Bioimaging. [26] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. Briggman, M. Helmstaedter, W. Denk, and H. S. Seung, “Supervised learning of image restoration with convolutional networks.” in ICCV’07. [27] M. Mozer, The Perception of Multiple Objects, A Connectionist Approach. MIT Press, 1991. [28] T. Serre, L. Wolf, and T. Poggio, “Object recognition with features inspired by visual cortex,” in CVPR, 2005. [29] J. Mutch and D. G. Lowe, “Multiclass object recognition with sparse, localized features,” in CVPR, 2006. [30] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks.” Science, 2006. [31] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layerwise training of deep networks,” in NIPS, 2007. [32] M. Ranzato, Y. Boureau, and Y. LeCun, “Sparse feature learning for deep belief networks,” in NIPS’07, 2007. [33] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?” in Proc. International Conference on Computer Vision (ICCV’09). IEEE, 2009. [34] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng., “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in ICML, 2009. [35] A. Ahmed, K. Yu, W. Xu, Y. Gong, and E. Xing, “Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks,” in ECCV. Springer-Verlag, 2008. [36] J. Weston, F. Rattle, and R. Collobert, “Deep learning via semisupervised embedding,” in ICML, 2008. [37] K. Kavukcuoglu, M.

Ranzato, and Y. LeCun, “Fast inference in sparse coding algorithms with applications to object recognition,” Tech. Rep., 2008, tech Report CBLL-TR-2008-12-01. [38] B. A. Olshausen and D. J. Field, “Sparse coding with an overcomplete basis set: a strategy employed by v1?” Vision Research, 1997. [39] S. Lazebnik, C. Schmid, and J. Ponce, “Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories,” in CVPR, 2006. [40] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, 2004. [41] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in CVPR, 2005. [42] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, “A tutorial on energy-based learning,” in Predicting Structured Data, G. Bakir, T. Hofman, B. Sch¨olkopf, A. Smola, and B. Taskar, Eds. MIT Press, 2006. [43] Y. LeCun and L. Bottou, “Lush reference manual,” Tech. Rep., 2002, code available at http://lush.sourceforge.net. [Online]. Available: http://lush.sourceforge.net [44] P. Sermanet, K. Kavukcuoglu, and Y. LeCun, “Eblearn: Open-source energy-based learning in c++,” in Proc. International Conference on Tools with Artificial Intelligence (ICTAI’09). IEEE, 2009. [45] R. Collobert, “Torch,” presented at the Workshop on Machine Learning Open Source Software, NIPS, 2008. [46] B. Boser, E. Sackinger, J. Bromley, Y. LeCun, and L. Jackel, “An analog neural network processor with programmable topology,” IEEE Journal of Solid-State Circuits, vol. 26, no. 12, pp. 2017–2025, December 1991. [47] E. S¨ackinger, B. Boser, J. Bromley, Y. LeCun, and L. D. Jackel, “Application of the ANNA neural network chip to high-speed character recognition,” IEEE Transaction on Neural Networks, 1992. [48] O. Nomura and T. Morie, “Projection-field-type vlsi convolutional neural networks using merged/mixed analog-digital approach,” in ICONIP’07. [49] R. Serrano-Gotarredona, M. Oster, P. Lichtsteiner, A. Linares-Barranco, R. Paz-Vicente, F. G´omez-Rodr´ıguez, L. Camu nas-Mesa, R. Berner, M. Rivas-P´erez, T. Delbr¨uck, S.-C. Liu, R. Douglas, P. H¨afliger, G. Jim´enez-Moreno, A. C. Ballcels, T. Serrano-Gotarredona, A. J. Acosta-Jim´enez, and B. Linares-Barranco, “Caviar: a 45k neuron, 5m synapse, 12g connects/s aer hardware sensory-processing-learningactuating system for high-speed visual object recognition and tracking,” Trans. Neur. Netw., vol. 20, no. 9, pp. 1417–1438, 2009. [50] J. Cloutier, E. Cosatto, S. Pigeon, F. Boyer, and P. Y. Simard, “Vip: An fpga-based processor for image processing and neural networks,” in MicroNeuro, 1996. [51] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, “Cnp: An fpga-based processor for convolutional networks,” in International Conference on Field Programmable Logic and Applications, 2009. [52] C. Farabet, C. Poulet, and Y. LeCun, “An fpga-based stream processor for embedded real-time vision with convolutional networks,” in Fifth IEEE Workshop on Embedded Computer Vision (ECV’09). IEEE, 2009

Large-Scale FPGA-based Convolutional Networks

Cl´ement Farabet1, Yann LeCun1, Koray Kavukcuoglu1, Eugenio Culurciello2, Berin Martini2, Polina Akselrod2, Selcuk Talay2 1. The Courant Institute of Mathematical Sciences, New York University, New York, USA 2: Electrical Engineering Department, Yale University, New Haven, USA

Chapter in Machine Learning on Very Large Data Sets, edited by Ron Bekkerman, Mikhail Bilenko, and John Langford, Cambridge University Press, 2011.

May 2, 2011

International Conference on Computational Modeling and Security (CMS 2016)

Proposing Solution to XOR problem using minimum configuration MLP

Vaibhav Kant Singh*

Assistant Professor, Department of Computer Science & Engineering, Institute ot Technology, Guru Ghasidas Vishwavidyalaya, Central University, Bilaspur, Chhattisgarh, Pincode 495001, India

StochasticPoolingforRegularizationof DeepConvolutionalNeuralNetworks

MatthewD.Zeiler Department of Computer Science Courant Institute, New York University zeiler@cs.nyu.edu

RobFergus Department of Computer Science Courant Institute, New York University fergus@cs.nyu.edu

ISSN: 2319-5967 ISO 9001:2008 Certified International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 6, Issue 1, January 2017

GPU Parallelization of Back-Propagation Neural Network Snehal M. Wagh1, Prof. D. R. Pawar2 Dept. of Computer Engg., Sinhgad College of Engg., Pune, India1

Training Many Neural Networks in Parallel via Back-Propagation

Javier A. Cruz-L´opez, Vincent Boyer Graduate Program in Systems Engineering Universidad Aut´onoma de Nuevo Le´on 66451, Monterrey, Mexico javier.cruz.003@gmail.com, vincent.boyer@uanl.edu.mx,

Didier El-Baz LAAS-CNRS, Universit´e de Toulouse, CNRS Toulouse, France elbaz@laas.fr

ParallelTrainingofaBack-PropagationNeuralNetworkusingCUDA

Xavier Sierra-Canto Divisi´on Industrial Universidad Tecnol´ogica Metropolitana M´erida, M´exico xavier.sierra@utmetropolitana.edu.mx

Francisco Madera-Ram´ırez Facultad de Matem´aticas Universidad Aut´onoma de Yucat´an M´erida, M´exico mramirez@uady.mx

V´ıctor Uc-Cetina Facultad de Matem´aticas Universidad Aut´onoma de Yucat´an M´erida, M´exico
uccetina@uady.mx

Charles University in Prague Faculty of Mathematics and Physics

MASTER THESIS

Bc. Ján Vojt

Deep neural networks and their implementation

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: doc. RNDr. Iveta Mrázová CSc. Study programme: Informatics Specialization: Software Systems

Prague 2016

Characterization of Neural Network Backpropagation on Chiplet-based GPU Architectures Colin A. Weinshenker College of William and Mary

THE MNIST DATABASE

of handwritten digits
Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

Deep Learning Made Easier by Linear Transformations in Perceptrons

Tapani Raiko Harri Valpola Yann LeCun Aalto University Aalto University New York University

5.1 MNIST Handwritten Digit Classification

The MNIST data set [9] consists of 28 by 28 pixel gray-scale images of handwritten digits with examples depicted in Figure 3. There are 60000 training samples and 10000 test samples. The mean activation of each pixel was subtracted from the data, and the dimensionality was dropped from 28×28 = 784 to 200 using PCA followed by a random rotation (see Figure 3). A classification network with layer sizes 200–200– 200–10 was learned with a normal MLP model (original), one with shortcut mappings included (shortcuts), and the proposed model with shortcut mappings and transformations in the nonlinearities (transformations) (see Figure 2). We also ran a simple 200–10 network (linear) for comparison. Training and test errors were tracked during learning and their computation was not

Figure 3: Left: Top row shows the PCA filters corresponding to the largest eigenvalues (1–5), second row to the smallest eigenvalues (196–200). The bottom rows show 10 filters after the random rotation in the principal subspace. Middle: Top row shows 5 examples from the MNIST handwritten digit data set. The second row shows reconstructions from the 200 component principal subspace. The third and forth row show reconstructions when including the added noise to the training data. Two instantiations of the noise is shown to remind that the noise is resampled for each epoch. Right: The corresponding images (16 examples) for the CIFAR-10 data set with 500 components.

linear original shortcuts transformations literature

MNIST classification

training error 8.99 0.063 0.058 0.068 test error 8.58 1.15 1.22 1.10 1.64 learning rate - 1.0 0.5 1.0 # of iterations 30k 4717 3498 2674

CIFAR-10 classification

training error 58.07 23.21 22.46 4.56 test error 59.09 44.42 44.99 43.70 48.47 # of iterations 32k 12k 8k 8k

MNIST autoencoder

training error 8.11 2.37 2.11 1.94 1.75 test error 7.85 2.76 2.61 2.44 2.55 # of iterations 92k 49k 38k 37k

Table 1: Results of the proposed method (transformations) compared against other methods run with the same settings and against results in the literature [4, 6, 11]. The number of iterations in the allocated time is reported to compare the computational complexities.

included in learning time. Learning time was restricted to 15 minutes, weight decay parameter was $\lambda = 0.0001$ and the regularization noise was used with standard deviation 0.4 for each input (see Figure 3).

The results are shown in Table 1 and Figure 4. With a proper learning rate (1.0), the proposed method gets the test error below 1.5% in six minutes and to 1.1% in fifteen minutes. Thus, the transformations make a simple gradient descent algorithm competitive in speed with complex state-of-the-art learning methods such as TONGA [8], which reaches test error 1.7% in around 30 minutes. Deep networks learned with backpropagation have reached 1.64% error in [4]. Deep belief network [5] gives 1.20% error.

The experiments were run with an 8-core desktop computer and Matlab implementation. The learning time

was measured with Matlab function cputime, which counts time spent by each core, so wall-clock learning times were smaller. For example one 900 cpu-second run took 233 seconds including computations of the training and test errors for result graphs. Table 1 reports the number of iterations reached in the allocated learning time.

One can note that the errors drop fast in the latter half of learning when the learning rate is decreased (See middle of Figure 4). This is mostly due to filtering out the noise caused by the stochastic gradient. It might seem that the comparison methods are even catching up at the end. However, when the experiment is repeated with a longer learning time (not shown here), the curves look qualitatively similar with comparison methods almost catching up at the end.

10 −0.9

10 −0.3

10 0.3

0

0.5

1

1.5

2

original

shortcuts

transformations

0 200 400 600 800

0

1

2

3

4

5

6

original shortcuts transformations

0 2000 4000 6000 8000 10000

0

10

20

30

40

50

60

70

original

shortcuts

transformations

Figure 4: Left: MNIST problem. Classification error rate in percentage as a function of learning rate after 15 minutes of learning. Lower curves in each figure are the training error rates and higher curves are test error rates. The vertical dashed line shows the point at which the learning rate starts to be decreased. Middle: MNIST problem. Error rates against learning time for the best learning rates for each method. Right: CIFAR-10 problem. Error rates against learning time.

−0.9 −0.8 −0.7 −0.6 −0.5 0

5

10

15

20

25

30

−0.2 −0.1 0 0.1 0.2 0

5

10

15

20

−2 −1 0 1 2 −0.5

0

0.5

Figure 5: MNIST classification problem. Histograms of the transformation parameters αi (left) and βi (middle) for the first hidden layer after learning. Right: Twenty examples of corresponding nonlinearities fi(·).

Let us also study some of the properties of the transformations. Figure 5 shows what the transformed nonlinearities look like in practice. How do they aff ect the Fisher information matrix? Equation (9) measures the covariance of the signals fi(·). The ratio of mean square nondiagonal element of the covariance to mean square diagonal element drops from 0.051 to 0.007 in the first hidden layer and from 0.080 to 0.009 in the second hidden layer, when comparing models learned traditionally or with transformation. The transformations also decrease the norm of the gradient with respect to weights of adjacent layers. The decrease is about 2 to 3-fold in the initial phase of learning. This might also explain why the proposed model performed worse than the others with a too small learning rate (See left part of Figure 4). With a small norm of the gradient and a small learning rate, the optimization simply does not finish in the allocated time.

To study which regularization method was important, the runs were repeated with several variants. The

table below shows test errors evaluated by including regularization methods one by one, using the learning rate γ = 1.0. The final run was repeated with ten times the learning time and γ = 0.5. regularization original shortcuts transform. none 1.87 2.02 1.63 weight decay 1.85 1.77 1.65 PCA 1.62 1.59 1.56 rotation 1.63 1.60 1.48 input noise 1.15 1.23 1.10 long run 1.03 1.17 1.02 Adding noise to the inputs turned out to be the most important regularization method, followed by dimensionality reduction by PCA. Using the transformations improved all variants.

CIFAR-10 Classification

The CIFAR-10 data set [6] consists of 32 by 32 pixel color images classified to 10 different classes, with examples depicted in Figure 3. There are 50000 training

samples and 10000 test samples. Each channel of each pixel was normalized to zero mean unit variance, and the dimensionality was dropped from 32×32×3 = 3072 to 500 using PCA followed by a random rotation. A classification network with layer sizes 500–500–500–10 was used with the same structure and variants as in MNIST classification. Learning time was restricted to 10000 seconds, the base learning rate was set by hand to $\gamma = 0.3$, weight decay parameter was $\lambda = 0.001$ and the regularization noise was used with standard deviation 0.4 for each input (see Figure 3).

The results are shown in Table 1 and Figure 4. Earlier test errors with MLP networks include 48.47% in [6] and 52.92% in [4]. The test error of 44.42% obtained here with the original MLP is already much better and it is further improved to 43.70% by using the proposed transformations. It should be noted that even the best back-propagation results are far behind from results obtained with for instance unsupervised pretraining or convolutional networks.

Role of Regularization

The role of regularization was studied by rerunning the CIFAR-10 classification experiment without any regularization. The network size was thus 3072–500– 500–10, and we dropped the learning rate to $\gamma = 0.03$ to better compare to the initialization. All three methods reached the training error 0.0%, but the test errors increased to 50.7%, 49.1%, and 46.8%. This overfitting was expected since the number of weights in the network is much larger than the number of labels in the training set, which makes the system underdetermined.

To further study the found solutions, we compute the angles between the 3072-dimensional incoming weight vectors of the neurons in the first hidden layer against the vectors that they were initialized to. The median angle over the 500 units was 39.6◦, 29.8◦, and 22.0◦ in the three models. Firstly, the angles are surprisingly small taking into account that high-dimensional vectors easily become rather orthogonal, which indicates that the found solution still retains much of the randomness of the initialization.2 Secondly, shortcut weights seem to help against overfitting.

MNIST Autoencoder

The third problem uses the same MNIST handwritten digit data set [9], but in this case both the inputs and outputs of the network are the images. The network topology is 784–500–250–30–250–500–784 with tanhnonlinearity at each layer except the bottleneck layer

2This also partly explains why initializing with unsupervised pretraining works so well in large networks.

in the middle. The output is scaled from -1 to 1 to match the tanh nonlinearity. The goal in this problem is to find a good 30-dimensional representation from which the image can be reconstructed. Naturally the shortcut connections that skip the bottleneck are not used (see left part of Figure 6). The labels are not used in this experiment at all. Learning time was restricted to 60000 seconds, the base learning rate was $\gamma = 0.05$, weight decay parameter was $\lambda = 0.001$ and the regularization noise was used with standard deviation 0.1 for each input. To avoid early divergence in learning, the learning rate was increased from one hundredth to the full rate exponentially during the first one percent of learning time.

The performance is measured by the average sum of squared reconstruction errors on the test data when it is scaled back to the range from 0 to 1. The final reconstruction error is 2.44 and some reconstructions are visualized in the middle of Figure 6. As a comparison, a linear 784–30–784 autoencoder gives an error 7.85. State-of-the-art comparison results are presented by Martens [11]: Hessian-free optimization gives 2.55 with a larger network. The results in [11] were further improved to 2.28 by initializing the network with layerwise pretraining, which could be used here, too.

Typical applications of dimensionality reduction methods include visualization, data denoising, or missing value imputation, but we will show another simple demonstration by comparing them as preprocessing methods for the k nearest neighbor (kNN) classifier. By using the raw pixel data, kNN gives 2.95% test error. Using a linear network to reduce dimensionality to 30 as a preprocessing gives 2.39% error, while using the proposed model gives 1.95% error. It can be concluded that the bottleneck layer has found a representation that also better separates the clusters formed by the different classes, despite the fact that the labels were not used in learning.

Discussion

We proposed transformations to nonlinearities that make learning MLP networks much easier. The motivation is to make the nonlinear mapping as separate as possible from the linear mapping which is modelled using shortcut weights. A basic stochastic gradient optimization became faster than state-of-the-art learning algorithms. Those algorithms could also be tested with transformations for further improvements. The theory of the speed-up is based on making the standard gradient more similar to the natural gradient by having the nondiagonal terms of the Fisher information matrix closer to zero. As a side effect, the transformed nonlinearities might also help in avoiding plateaus [10]. The

Deep Learning Made Easier by Linear Transformations in Perceptrons

30

layers

layer

500 784784 500 250 250

input hidden hidden layersbottleneck (hidden)

30 500 784784 500 250 250

output

0 1 2 3 4 5 6

x 10 4

1.5

2

2.5

3

3.5

4

4.5

5

5.5

6

original shortcuts transformations

Figure 6: MNIST autoencoder. Left: Traditional autoencoder network above, shortcut connections included below. Note that the bottleneck layer does not have a nonlinearity or transformations. Middle: Each triplet shows an example digit from test data, its reconstruction with a deep autoencoder, and a reconstruction with a linear autoencoder as a comparison. Right: Error rate in average sum of squared reconstruction errors plotted against learning time in seconds. Higher curves are test errors, lower curves are training errors. The vertical dashed line shows the point at which the learning rate starts to be decreased.

experiments showed that these simple transformations helps learning deep structures at least up to 5 hidden layers using good-old back-propagation learning.

The transformations also seemed to help generalization when no regularization was used. We think that this is because the transformations help to separate the simpler and more complex parts of each mapping. Let us think of a network with just one hidden layer. The linear part of the problem (the shortcut connections C) do not suffer much from the overfitting and can be learned more reliably even without regularization. Overfitting the more complex parts A and B might not hurt the performance as much when they do not influence the linear part of the whole mapping. It might be possible to test this hypothesis in the future.

Another theoretical study that could be done in the future, is to measure the angle between the traditional gradient and the natural gradient with and without transformations. It would require a small network such that computing the natural gradient would be feasible.

The effect of the transformations could also be studied in other contexts. For variational Bayesian (VB) learning, the proposed transformations make both the signals in the network and the network weights less dependent a posteriori. Often they are assumed to be independent in the posterior approximation of VB anyway. Thus, the transformation makes the effect of the assumption smaller and the approximation more accurate. This should help in avoiding the problem of underfitting (or output weights of too many hidden neurons going to zero). One could also use MCMC methods for sampling network weights. The effective

ness of the sampling process depends heavily on how large jumps can be made in the parameter space. We can make longer jumps for the matrices A and B if we use the proposed transformations to ensure that even large changes in them do not affect the linear part of the input-output mapping. These new contexts would highlight the difference between transforming nonlinearities compared to transforming gradient factors [14, 13].

The term deep learning refers either to networks with many layers (as in this work) but sometimes it is used for unsupervised pretraining which allows for wellperforming deeper networks in practice. The proposed transformations could also be applied to initializations based on unsupervised pretraining. We could also study a multi-task learning problem combining classification and auto-encoding. This simple alternative to layerwise pretraining might provide some useful insights about combining unsupervised and supervised learning.

A “poor man’s variant” of the proposed framework would be to use shortcut connections and fixed nonlinearities such as f(x) = tanh(x)−0.5x. Dispite its simplicity, this variant might still provide most of the discussed benefits in practice.

Another future direction is to introduce a third transformation. While currently we aim at making the Fisher information matrix diagonal, we could also make it closer to the unit matrix. This could be done by using a multiplicative transformation in order to normalize the scale of the output and the slope of each nonlinearity.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

please note that your browser may uncompress these files without telling you. If the files you downloaded have a larger size than the above, they have been

uncompressed by your browser. Simply rename them to remove the .gz extension. Some people have asked me "my application can't open your image files". These files are not in any standard image format. You have to write your own (very simple) program to read them. The file format is described at the bottom of this page.

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

With some classification methods (particuarly template-based methods, such as SVM and K-nearest neighbors), the error rate improves when the digits are centered by bounding box rather than center of mass. If you do this kind of pre-processing, you should report it in your publications.

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets.

The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. We made sure that the sets of writers of the training set and test set were disjoint.

SD-1 contains 58,527 digit images written by 500 different writers. In contrast to SD-3, where blocks of data from each writer appeared in sequence, the data in SD-1 is scrambled. Writer identities for SD-1 is available and we used this information to unscramble the writers. We then split SD-1 in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available.

Many methods have been tested with this training set and test set. Here are a few examples. Details about the methods are given in an upcoming paper. Some of those experiments used a version of the database where the input images where deskewed (by computing the principal axis of the shape that is closest to the vertical, and shifting the lines so as to make it vertical). In some other experiments, the training set was augmented with artificially distorted versions of the original training samples. The distortions are random combinations of shifts, scaling, skewing, and compression.

FILE FORMATS FOR THE MNIST DATABASE

The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. General info on this format is given at the end of this page, but you don't need to read that to use the data files.

All the integers in the files are stored in the MSB first (high endian) format used by most non-Intel processors. Users of Intel processors and other low-endian machines must flip the bytes of the header.

There are 4 files:

```
train-images-idx3-ubyte: training set images
train-labels-idx1-ubyte: training set labels
t10k-images-idx3-ubyte:  test set images
t10k-labels-idx1-ubyte:  test set labels
```

The training set contains 60000 examples, and the test set 10000 examples.

The first 5000 examples of the test set are taken from the original NIST training set. The last 5000 are taken from the original NIST test set. The first 5000 are cleaner and easier than the last 5000.

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label

The labels values are 0 to 9.
```

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

TEST SET LABEL FILE (t10k-labels-idx1-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  10000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label
```

The labels values are 0 to 9.

TEST SET IMAGE FILE (t10k-images-idx3-ubyte):

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  10000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

THE IDX FILE FORMAT

the IDX file format is a simple format for vectors and multidimensional matrices of various numerical types.

The basic format is

```
magic number
size in dimension 0
size in dimension 1
```

```
size in dimension 2
.....
size in dimension N
data
```

The magic number is an integer (MSB first). The first 2 bytes are always 0.

The third byte codes the type of the data:
0x08: unsigned byte
0x09: signed byte
0x0B: short (2 bytes)
0x0C: int (4 bytes)
0x0D: float (4 bytes)
0x0E: double (8 bytes)

The 4-th byte codes the number of dimensions of the vector/matrix: 1 for vectors, 2 for matrices....

The sizes in each dimension are 4-byte integers (MSB first, high endian, like in most non-Intel processors).

The data is stored like in a C array, i.e. the index in the last dimension changes the fastest.

The digit images in the MNIST set were originally selected and experimented with by Chris Burges and Corinna Cortes using bounding-box normalization and centering. Yann LeCun's version which is provided on this page uses centering by center of mass within in a larger window.

Yann LeCun, Professor
The Courant Institute of Mathematical Sciences
New York University
yann@cs.nyu.edu

Corinna Cortes, Research Scientist
Google Labs, New York
corinna at google dot com

The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.[1][2] The database is also widely used for training and testing in the field of machine learning.[3][4] It was created by "re-mixing" the samples from NIST's original datasets. The creators felt that since NIST's training dataset was taken from American Census Bureau employees, while the testing dataset was taken from American high school students, it was not well-suited for machine learning experiments.[5] Furthermore, the black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced grayscale levels.[5]

Sample images from MNIST test dataset

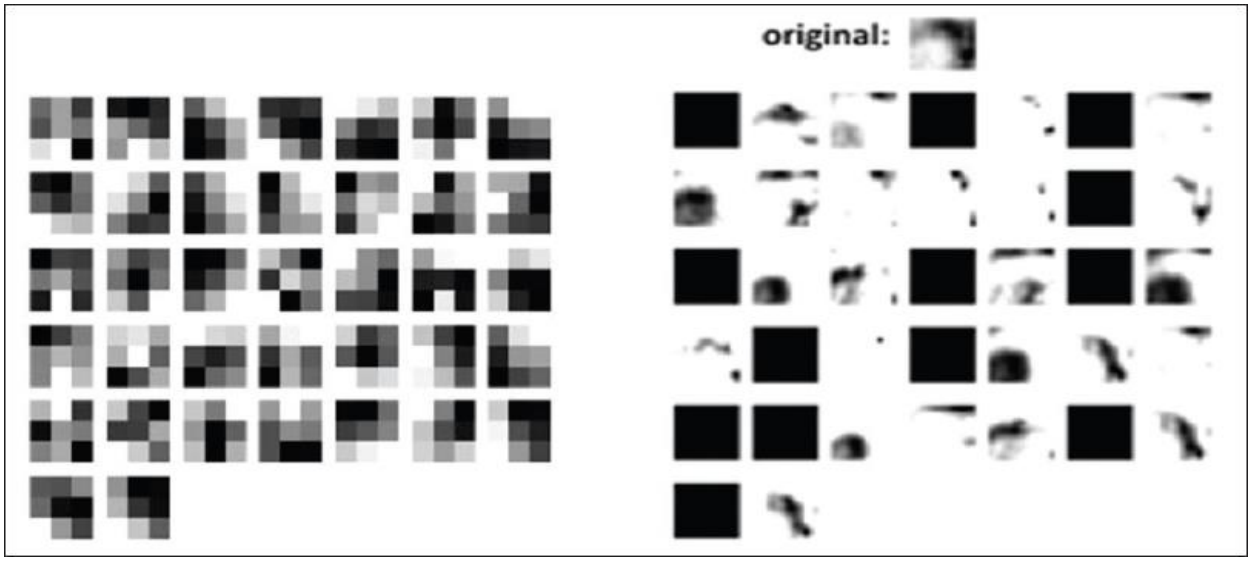
Each example is a 28×28 image of black and white pixels, where white is foreground of value 0 and black is background of value 255. Each image contains a digit, which is centered using the center of mass of the pixels. The data in the set is stratified with approximately equally many occurrences of each class

The MNIST database contains 60,000 training images and 10,000 testing images.[6] Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.[7] There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23%.[8] The original creators of the database keep a list of some of the methods tested on it.[5] In their original paper, they use a support-vector machine to get an error rate of 0.8%.[9] An extended dataset similar to MNIST called EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.[10]

Dataset

The set of images in the MNIST database is a combination of two of NIST's databases: Special Database 1 and Special Database 3. Special Database 1 and Special Database 3 consist of digits written by high school students and employees of the United States Census Bureau, respectively.[5]

MNIST classification The training (optimization) consists of using the entire dataset but splitting 75% into a training set and the rest into a validation set. For each training batch a corresponding validation batch is used to estimate the ability for the neural network to generalization to unseen data. The validation batches are chosen with a random starting point in the validation set. Thereafter, the number required by the batch size is selected as each example following this randomly chosen one. The reason is to maintain stratified data of approximately equal many classes.
Running the trained neural network architecture on the test data yields and accuracy of 97.31%. The 32 filters used by the first convolution layer are applied directly onto the input image. These gives a good intuition about what the neural network tries to use for classification and are shown in figure



The first 32 filters which the neural network learned to use when classifying handwritten digits.

Performance

Some researchers have achieved "near-human performance" on the MNIST database, using a committee of neural networks; in the same paper, the authors achieve performance double that of humans on other recognition tasks.[8] The highest error rate listed[5] on the original website of the database is 12 percent, which is achieved using a simple linear classifier with no preprocessing.[9]

In 2004, a best-case error rate of 0.42 percent was achieved on the database by researchers using a new classifier called the LIRA, which is a neural classifier with three neuron layers based on Rosenblatt's perceptron principles.[11]

Some researchers have tested artificial intelligence systems using the database put under random distortions. The systems in these cases are usually neural networks and the distortions used tend to be either affine distortions or elastic distortions.[5] Sometimes, these systems can be very successful; one such system achieved an error rate on the database of 0.39 percent.[12]

In 2011, an error rate of 0.27 percent, improving on the previous best result, was reported by researchers using a similar system of neural networks.[13] In 2013, an approach based on regularization of neural networks using DropConnect has been claimed to achieve a 0.21 percent error rate.[14] Recently,[*when?*] the single convolutional neural network best performance was 0.31 percent error rate.[15] As of August 2018, the best performance of a single convolutional neural network trained on MNIST training data using realtime data augmentation is 0.26 percent error rate.[16] Also, the Parallel Computing Center (Khmelnitskiy, Ukraine) obtained an ensemble of only 5 convolutional neural networks which performs on MNIST at 0.21 percent error rate.[17][18] Incorrect labeling of the testing dataset may prevent reaching test error rates of 0%.[16]

Classifiers

This is a table of some of the machine learning methods used on the database and their error rates, by type of classifier:

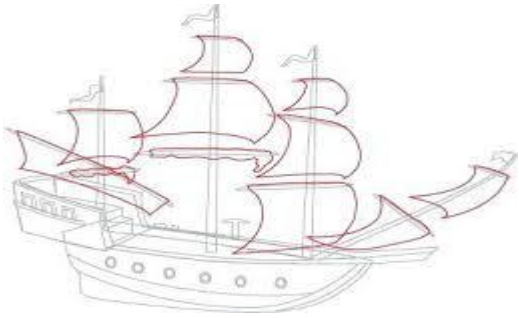
Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52[19]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87[20]
Non-linear classifier	40 PCA + quadratic classifier	None	None	3.3[9]
Support-vector machine (SVM)	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56[21]
Deep neural network (DNN)	2-layer 784-800-10	None	None	1.6[22]
Deep neural network	2-layer 784-800-10	Elastic distortions	None	0.7[22]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	Elastic distortions	None	0.35[23]
Convolutional neural network (CNN)	6-layer 784-40-80-500-1000-2000-10	None	Expansion of the training data	0.31[15]
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.27[24]

Type	Classifier	Distortion	Preprocessing	Error rate (%)
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	Elastic distortions	Width normalizations	0.23[8]
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.21[17][18]

ΠΗΓΗ: https://en.wikipedia.org/wiki/MNIST_database

ΠΗΓΗ: THEORY FROM CHALMERS Implementation of Deep Feedforward Neural Network with CUDA backend for Efficient and Accurate Natural Image Classification Convolutional Neural Network Applications Master's thesis in Complex Adaptive Systems

AUGUST VON HACHT Department of Physics CHALMERS UNIVERSITY OF TECHNOLOGY Gothenburg, Sweden 2017



ΑΝΤΩΝΙΟΣ ΜΙΧ. ΜΑΡΑΓΚΟΥΔΑΚΗΣ