



ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΓΩΓΗΣ, ΑΥΤΟΜΑΤΙΣΜΟΥ & ΡΟΜΠΟΤΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση χειρονομιών με τη χρήση οπτικών δεδομένων

Κατσουλάκης Ν. Εμμανουήλ

Επιβλέπων: Δρ. Κοσμόπουλος Δημήτριος



ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΓΩΓΗΣ, ΑΥΤΟΜΑΤΙΣΜΟΥ & ΡΟΜΠΟΤΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση χειρονομιών με τη χρήση οπτικών δεδομένων

Κατσουλάκης Ν. Εμμανουήλ

Επιβλέπων: Δρ. Κοσμόπουλος Δημήτριος

Εγκρίθηκε από την τριμελή επιτροπή στις /06/2016

Δρ. Κοσμόπουλος Δημήτριος, Καθηγητής
(Υπογραφή)

Δρ. Σφακιωτάκης Μιχαήλ, Επίκουρος Καθηγητής
(Υπογραφή)

Δρ. Φασουλός Ιωάννης, Επίκουρος Καθηγητής
(Υπογραφή)

Ηράκλειο Κρήτης, Ιούνιος 2016

TABLE OF CONTENTS

LIST OF FIGURES.....	6
LIST OF TABLES.....	9
ABSTRACT.....	12
CHAPTER 1. INTRODUCTION	14
1.1 OVERVIEW	14
1.2 RELATED WORK.....	14
1.3 PROPOSED APPROACH.....	16
1.4 CONTRIBUTION OF THE THESIS.....	16
CHAPTER 2. MACHINE LEARNING.....	18
2.1 MACHINE LEARNING	18
2.2 MACHINE LEARNING WORKFLOW	18
CHAPTER 3. NEURAL NETWORKS	19
3.1 NEURAL NETWORKS.....	19
3.1.1 WHAT IS A NEURAL NETWORK?.....	19
3.2 THE BASICS OF NEURAL NETWORKS	19
3.2.1 ACTIVATION FUNCTIONS	23
3.2.2 HOW DO NEURAL NETWORKS DIFFER FROM CONVENTIONAL COMPUTING?	24
3.2.3 WHAT APPLICATIONS SHOULD NEURAL NETWORKS BE USED FOR?	25
3.3 LEARNING TECHNIQUES.....	25
3.3.1 SUPERVISED AND UNSUPERVISED MACHINE LEARNING	25
3.3.2 REINFORCEMENT LEARNING.....	28
CHAPTER 4. RECURRENT NEURAL NETWORKS	30
4.1 OVERVIEW	30
4.2 FORMALISM.....	34
4.2.1 FORMAL DESCRIPTION OF RECURRENT NEURAL NETWORKS.....	34
4.2.2 APPROACHING A MACHINE LEARNING PROBLEM.....	36
4.3 RESERVOIR COMPUTING.....	37

4.3.1	ECHO STATE NETWORK.....	38
4.3.2	LIQUID STATE MACHINE	39
4.3.3	EVOLINO	39
4.3.4	BACKPROPAGATION-DECORRELATION	39
CHAPTER 5. ECHO STATE NETWORK.....		40
5.1	OVERVIEW	40
5.2	BASIC ECHO STATE NETWORK MODEL.....	40
5.3	RESERVOIR READOUTS.....	42
5.3.1	SINGLE-LAYER READOUT.....	43
5.4	ECHO STATE PROPERTY.....	46
5.5	EXAMPLE OF A SMALL TIMER NETWORK	49
5.6	SHORT-TERM MEMORY	51
5.7	TRAINING AN ESN AS A DELAY LINE	52
5.8	MEMORY CAPACITY	54
5.9	GENERIC RESERVOIR “RECIPES”	55
5.9.1	DIFFERENT TOPOLOGIES OF THE RESERVOIR	55
5.9.2	INTERNAL UNITS ACTIVATION METHODS	56
5.10	FORMULATION OF TRAINING PROCEDURE.....	60
5.11	PRODUCING A RESERVOIR	61
5.11.1	Function of the Reservoir.....	61
5.11.2	GLOBAL PARAMETERS OF THE RESERVOIR	62
CHAPTER 6. PRACTICAL APPROACH OF ESN		66
6.1	MAIN PARAMETERS OF THE NETWORK	66
6.2	PARAMETER SELECTION SETUP	66
6.3	MANUAL PARAMETER SELECTION	67
6.4	INITIAL TRANSIENT.....	67
6.5	OUTPUT FEEDBACK AND STABILITY PROBLEMS	67
6.5.1	OUTPUT FEEDBACK.....	67
6.5.2	TEACHER FORCING.....	68
6.5.3	FEEDBACK STABILITY PROBLEMS	69
CHAPTER 7. SUPERVISED TRAINING FOR SEQUENCE LABELING.....		70
7.1	SEQUENCE LABELING OVERVIEW	70
7.2	SEGMENT CLASSIFICATION.	71

CHAPTER 8. TEMPORAL GESTURE RECOGNITION EXPERIMENTS.....	72
8.1 AIM OF THE EXPERIMENT	72
8.2 DATA.....	74
8.2.1 GENERAL DESCRIPTION.....	74
8.2.2 MAIN CHARACTERISTICS OF THE DATASET	74
8.2.3 DATA FORMAT	78
8.3 PREPROCESSING	81
8.3.1 DATA TRANSFORMATION	81
8.3.2 DATA RESCALING AND INPUT FORMAT	83
8.3.3 OUTPUT FORMAT	85
8.4 EXPERIMENTAL SETUP	86
8.5 TRAINING-TEST ECHO STATE NETWORK: ALGORITHM.....	87
8.6 EXPERIMENTS	93
8.6.1 ECHO STATE PROPERTY (A SIMPLE EXPERIMENT)	93
8.6.2 GESTURE CLASSIFICATION EXPERIMENTS	94
8.6.3 RESULTS.....	100
CHAPTER 9. COMPARISONS	110
9.1 ChaLearn 3-TOP RANKED APPROACHES	110
9.2 COMPARISONS WITH OTHER APPROACHES AND VARIATIONS	111
CHAPTER 10. CONCLUSION	113
BIBLIOGRAPHY	114

LIST OF FIGURES

Figure 1. Basic form of Neural Network.....	20
Figure 2. A Single Node Example	21
Figure 3. Delta Rule	22
Figure 4. Some common activation functions used in neural networks ((Verstraeten, 2009-2010)).	24
Figure 5. Supervised training process (Jaeger, 2002b).	26
Figure 6. Overfitting: The model is learning the noise on the data instead of generalizing (learning then statistical properties of the data).	27
Figure 7. Overfitting: the error on the training set keeps decreasing while the error on the (unseen) test set increases. The model is learning the noise on the data instead of generalizing (learning then statistical properties of the data).	27
Figure 8. Typical structure of a feedforward network (left) and a recurrent network (right) (Jaeger, 2002b).	30
Figure 9. Principal pathway in the black-box modeling (Jaeger, 2002b).	32
Figure 10. A. Traditional gradient-descent-based RNN training methods adapt all connection weights (bold arrows), including input-to-RNN, RNN- internal, and RNN-to-output weights. B. In Reservoir Computing, only the RNN-to-output weights are adapted (Lukosevicius, 2012).	33
Figure 11. A topology of RNN models.....	34
Figure 12. Basic architecture of RNN models. Shaded arrows indicate optional connections. Dotted arrows mark connections which are trained in the "echo state network" approach (in other approaches, all connections can be trained), (Jaeger, 2001).	35
Figure 13. Basic echo state network architecture. Dashed arrows indicate connections that are optional (Yildiz, et al., 2012).	41
Figure 14. An echo state network. (Lukosevicius, 2012)	44
Figure 15. Timer network	50
Figure 16. Performance of the network. Dotted line in last graph shows network's output and solid line desired or teacher output.....	51
Figure 17. Short Term Memory of RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations	

of the reservoir, and the network 'forgets' the first inputs (Graves, 2008).	52
Figure 18. Setup of delay learning task.....	53
Figure 19. 20 with a 20-unit DR. Results of training delays $k = 4, 8, 16$. Top row: input weights of size -0.1 or $+0.1$, bottom row: input weights sized -0.001 or $+0.001$	54
Figure 20. Time warped data.....	59
Figure 21. Random scaled internal weights matrix, spectral radius 0.95.	64
Figure 22. An ESN with output feedbacks trained with teacher forcing.....	69
Figure 23. Data set gesture categories.	73
Figure 24. The 20 gestures performed by one person, which consist one sequence.	75
Figure 25. (a, b) Left and right handed instances for gesture "vieni qui", (c, d, e) left and right handed instances and arm position of "vattene" and.	77
Figure 26. Different data modalities of the provided data set. From left to right are the image selected from the RGB video, depth video, user-index video, and skeletal model respectively.	78
Figure 27. Fields of structure Labels for a sample sequence.....	80
Figure 28. Left and right edge: tracked joint types, middle: skeleton joint positions.....	81
Figure 29. Twelve joint pairs data after normalization over sampling period. .	83
Figure 30. Input data rescaled to interval $[0, 1]$. Last row: bias input.	84
Figure 31. A portion of teacher sequence (4 of 13 inputs) and target output vector (4 of 20) and its manual alignment of them. Each input signal contains 20 regions (represented as different color setup) of equally unique gesture instances performed by one individual.	86
Figure 32. Response of an arbitrarily selected internal unit of the network for different spectral radius ρ . The last trace shows the unit impulse input sequence (training signal).	94
Figure 33. Top row: a portion (8 of 20) of labels (red line) and network output (blue line). Last row: traces of some internal units.	96
Figure 34. Top row: a portion (8 of 20) of labels (red line) and network output (blue line). Last row: traces of some internal units without feedback connections.	97
Figure 35. True-false classified gesture.	98
Figure 36. Bar-graph of four different internal units activation error classification rate grouped in each gesture.	103

Figure 37. Error classification rate as a function of spectral radius α . Training parameters (except spectral radius) was the same for all executed simulations.	105
Figure 38. Error classification rate as a function of number of internal units. Training parameters (except number of internal units) was the same for all executed simulations.	106
Figure 39. Shows 4 (of twelve) input channels of two different people performing the same gesture. After normalization 4 input channels of different training sets First person needs 80 frames (4 seconds) to complete 'prendere' gesture and second one only 30 frames(1.5 seconds).	107
Figure 40. Top: TWI_ESN network internal units activations with time constants $\Delta t=0.5$. Bottom: TWI_ESN network internal units activations with time constants $\Delta t=0.2$	109

LIST OF TABLES

Table 1. Calculated distances joint pairs.....	82
Table 2. Network input vector.....	84
Table 3. Network output vector.	85
Table 4. ESN sub-optimal parameter setup and best performance for each type of internal units activation.....	101
Table 5. Error classification rate of each gesture.....	102
Table 6. Team methods and results. HMM: Hidden Markov Models. KNN: Nearest Neighbor. RF: Random Forest. Tree: Decision Trees. ADA: Adaboost variants. SVM: Support Vector Machines. Fisher: Fisher Linear Discriminant Analysis. GMM: Gaussian Mixture Models. NN: Neural Networks. DGM: Deep Boltzmann Machines. LR: Logistic Regression. DP: Dynamic Programming. ELM: Extreme Learning Machines.....	112

Mathematical notations

α	Scalars
\mathbf{a}	Vector
\mathbf{A}	Matrix
\mathbf{A}^t	Matrix transpose
\mathbb{R}^N	A set of real-valued vectors of size N
$\mathbb{R}^{N \times M}$	A set of real-valued vectors of size $N \times M$
$[\cdot; \cdot]$	Vector or matrix vertical concatenation
$\ \cdot\ $	Euclidean distance
n	Discrete time
$\mathbf{u}(n)$	Input signal
\mathbf{W}	Internal weight matrix
\mathbf{W}^{in}	Input weight matrix
\mathbf{W}^{back}	Output weight matrix
\mathbf{W}^{out}	Output weight matrix
$\mathbf{x}(n)$	Internal state signal at time n , reservoir activation
$\mathbf{y}(n)$	Output signal at time n
$\mathbf{y}^{target}(n)$	Target(desired, teacher) output signal at time n
T	Number of time steps
$\rho(\mathbf{W})$	Spectral radius, i.e., largest absolute eigenvalue of \mathbf{W}
s_{scal}^{Rinput}	'Active' inputs scaling vector
s_{shift}^{Rinput}	'Active' inputs shifting vector
s_{scal}^{Binput}	Bias input scaling
s_{shift}^{Binput}	Bias input shifting
$s_{scal}^{teacher}$	Teacher scaling vector

Mathematical notations

s_{scal}^{back}	Teacher shifting vector
s_{scal}^{back}	Feedback scaling vector
v^{res}	Internal units noise vector
v^{tar}	Teacher signal noise vector

ABSTRACT

Gesture recognition is one of the most significant issues of human-machine interconnection via mathematical algorithms. Gestures can originate from body or face motion and most of the time are recorded by a camera. Extracted cameras' data (such as depth map, skeletal model) is provided to specific computational algorithms in order to achieve gesture recognition.

This study is focused on temporal gesture recognition (detection plus classification) from skeletal data by a specific Reservoir Computing type called Echo State Network. The goal is to indicate the real order of gestures in the sequence expressed as a false gestures recognition percentage. The training and test data have been downloaded from the *Chalearn Gesture Recognition Challenge*. They contain a large manually labelled database of 7,820 gestures from a lexicon of 20 Italian gesture categories recorded with a Kinect™ camera.

Recurrent neural networks are a part of Artificial Neural Network architecture that is inspired by brain cyclical connectivity of neurons and uses recurrent function loops to store information. Recurrent neural networks (RNNs) have a great potential for "black box" modeling of nonlinear dynamical systems. Reservoir Computing is a subclass of Recurrent Neural Networks (RNNs). The "echo state" approach is a novel approach of RNNs. *Large* RNNs are interconnected as "reservoirs" of complex, excitable dynamics. Output units "tap" from this reservoir by linearly combining the desired output signal from the rich variety of excited reservoir signals. This idea leads to training algorithms where *only* the network-to-output connection weights have to be trained. This can be solved using ridge regression algorithms. Potential applications of ESN are dynamical systems, *which were difficult to learn with previous methods. They include (long) periodic sequence generators, multistable switches, frequency measurement devices, controllers for nonlinear plants, long short-term memories, dynamical pattern recognizers, and notably, long-term predictors of chaotic attractors. Today ESNs are widely used in dynamical pattern recognition applications, control, and time series prediction applications* (Jaeger, 2001)

We investigate the performance of three different types of ESN (4 implementations) internal activation units: plain_ESN, which generates the internal state of an ESN with standard additive-sigmoid,

leaky1_ESN\Leaky_ESN, which updates internal state using leaky integrator neuron model, and twi_ESN, which updates internal states using a time warping invariant model. The obtained network performance is measured by error classification rate. Error classification rate for each gesture is also provided. Overall approach achieves 28.53% gesture misclassification rate, providing error rate reduction of 3.12% compared to the best single modal approach presented in *Chalearn Gesture Recognition Challenge 2013*.

Keywords: single-modal gesture recognition, ESN, gesture recognition, temporal classification.

CHAPTER 1. INTRODUCTION

1.1 OVERVIEW

Gesture recognition aims to recognize human motions, which are combined movements of body, head, arms and hands. This research field is very popular due to vast human-computer interaction applications.

The most important thing in hand gesture recognition systems is the input feature, and the selection of good features representation. Human-computer interaction takes place by multiple sensors signals. In the beginning of gesture recognition research, the majority of approaches were based on controllers which equip users with wearable hardware devices for recording motion data. Nowadays proposed approaches focus on gesture recognition with vision-based methods for capturing motion data. Plenty of computational methods are used for analyzing cameras' motion data.

Introduction of *Kinecttm* outperformed gesture recognition research. *Kinecttm* is a relatively cheap motion sensing input device which is inbuilt RGB camera, a depth sensor, and a multi-array microphone. Thus, this device provides multi-modal sensing data such as RGB image, depth image, audio, and skeletal. Provided *Kinecttm* features make this device ideal for gesture recognition systems' design.

Reservoir Computing methods have some properties that make them an attractive choice for pattern recognition: learning sequences can represent data of different types (e.g. audio and video data), they can learn to store only the meaningful information of the data and they can learn to identify sequential patterns.

1.2 RELATED WORK

Based on their data capturing method, gesture recognition systems can be classified in two major categories. First one includes controller-based recognition systems: users hold or wear devices during gesture performing. *StrinGlove*, introduced in (Kuroda, et al., 2004), is a low price data glove, which is used for sign language recognition. (Schreiber, et al., 2009) evaluated

the potential of a gesture-based human computer interaction system with a Wii Remote.

Second one is controller-free recognition systems: users do not have to wear or hold any hardware. Various types of sensors can be used for data capturing in these systems. Most used sensors in recent research field is cameras, lasers and infrared. Moreover, camera-based recognition systems can be divided in subsections: single camera, stereo camera systems and so on.

Gestures to be recognized can be divided in two categories, static and dynamic. Static gesture recognition (also known as posture recognition) is expressed as a stable body posture. Body posture can be defined as the static movement, e.g. holding the hand in a specific pose is a posture, a victory sign, for example, pointing. On the other hand, dynamic gesture represents a sequence of dynamic body movements. In (Just, et al., 2006) introduced an approach to hand posture classification and recognition tasks. (Fang, et al., 2007) proposed a robust real-time hand gesture recognition method. Firstly, a specific gesture is required to trigger the hand detection followed by tracking; then the hand is segmented using motion and color cues; finally, in order to break the limitation of aspect ratio encountered in most of learning based hand gesture methods, the scale-space feature detection is integrated into gesture recognition.

Most frequently used approaches for Dynamic recognition and classification of hand gestures is Machine Learning based approaches (Mitra & Acharya, 2007). The most common methods that considered the gesture as a result of some stochastic processes include Hidden Markov Model, Finite State Machine, Kalman Filtering, Artificial Neural Network, and Principal Component Analysis.

In (Yamato, et al., 1992) a Hidden Markov Model method was first applied for gesture recognition. In (Yang & Ahuja, 2001) introduced a method to recognize 40 hand gestures of American Sign Language (ASL) which used Time-Delay Neural Network (TDNN). Pixel-level motion trajectory is obtained across the image sequence by multi-scale motion segmentation and affine transformation. Then, the motion trajectory is matched to a given gesture model with TDNN.

Different data capturing devices provide data which can be combined. These modalities allow us to improve recognition performance. (Bolt & Herranz,

1992) proposed a framework in which both hand gestures and speech signals are used to augment the user's ability to communicate with computers. In their prototype, two handed gestures, both static and dynamic, were designed to input concepts, manipulate items and specify actions to be taken.

1.3 PROPOSED APPROACH

In this study, we focus on the detection and classification of single modally expressed gestures as performed freely by different people. Single modal gesture recognition arises many challenging research issues such as extraction of valuable features of the provided data, appropriate reform of data, and construction of an effective gesture recognition classifier. The database we used was introduced for the needs of *Chalearn Gesture Recognition Challenge*. This database comprises multimodal cultural-anthropological gestures of life realized as both hands-body-arm movements and spoken words, which are intermixed with irrelevant phrases and body movements. This features consist a demanding and challenging database.

We present a single modal temporal gesture recognition framework that exploits skeletal data captured by a *Kinecttm* device. An Echo State Network is used to build the gesture recognition classifier. This classifier is trained to "memorize" significant features of the provided reformed skeletal data and recognize gestures in a novel sequence data during test phase. Test error is measured by misclassification rate.

A detailed report of the results is presented in section 8.6.3. Spectral radius can be tuned in a wide range of values without significant change in network performance and rearranging the input data timescale did not improve performance; is the significant result of experiments.

1.4 CONTRIBUTION OF THE THESIS

The main contribution of this study is to overview and describe an alternative method to classical supervised training of RNNs. This method belongs to the subcategory of RNNs called Reservoir computing. Reservoir computing methods consist a powerful tool for pattern generation.

This study is organized as follows. Firstly, in Chapter 2, we give a brief review of Machine Learning.

In Chapter 3 we refer to Neural Networks and the important section of learning techniques (supervised and unsupervised learning).

In connecting Chapter 4 we represent a description of Artificial Recurrent Neural Networks and an overview of Reservoir Computing Trends.

In Chapter 5 we make a detailed description of Echo State Network and its significant properties. We continue our investigation in Chapter 6 with some practical issues concerning ESN.

Chapter 7 concerns readouts of a network and classification strategies.

Finally, in Chapter 8, we describe the executed experiments and provide our results. Some comparisons to other approaches as represented in (Escalera, et al., 2013b) are demonstrated in Chapter 9.

CHAPTER 2. MACHINE LEARNING

2.1 MACHINE LEARNING

Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that iteratively learn from data. That is software that can develop properties to grow and change when new data is provided.

Most significant feature of this type of Artificial Intelligence is learning. Roughly speaking, machine learning tries to configure how artificial systems can incorporate experience in order to improve their performance.

2.2 MACHINE LEARNING WORKFLOW

A typical workflow model for machine learning is described below:

- Choose a computational model (i.e. Neural Network) which fits the learning task.
- Choose a training algorithm and the associated parameters.
- Collect or artificially generate data to apply to the learning task.
- Train the model to approximate the desired task. For training, data is used. This data contains the 'experience' from which the model learns. Mathematically training is an optimization problem where parameters are adapted to estimate model performance.

CHAPTER 3. NEURAL NETWORKS

3.1 NEURAL NETWORKS

3.1.1 WHAT IS A NEURAL NETWORK?

The definition of a neural network, or otherwise called, 'artificial' neural network (ANN), is given by Dr. Robert Hecht-Nielsen who first introduced neurocomputers. A neural network is defined as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" (Hecht-Nielsen, 1989).

Neural Networks are a part of machine learning and they are inspired by the physical intelligence, brain. This study is mostly concerned with Recurrent Neural Networks and the Reservoir Computing approach.

Trained Neural Network can be assumed as a black-box model since its internal workings are not known enough. The topology and parameters of Neural Networks are manually adapted, based on experience, and trial and error. Training is a looped process and it's time and computation costly. It is hard to analyze mathematically complex and recurrent Neural Networks.

Mathematically, Neural Networks can represent (almost) any function (Cybenko, 1989) or, in the case of Recurrent Neural Networks, dynamical system.

During the last 50 years, computational power is growing exponentially (according to Moore's law). Even so, it still is the major problem of Neural Networks.

3.2 THE BASICS OF NEURAL NETWORKS

Neural networks are usually split into layers. As displayed at the figure below, each layer consists of a number of interconnected 'nodes'. Every node contains an 'activation function'. The 'input layer' is responsible for operating via the

'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' like the figure below (Figure 1).

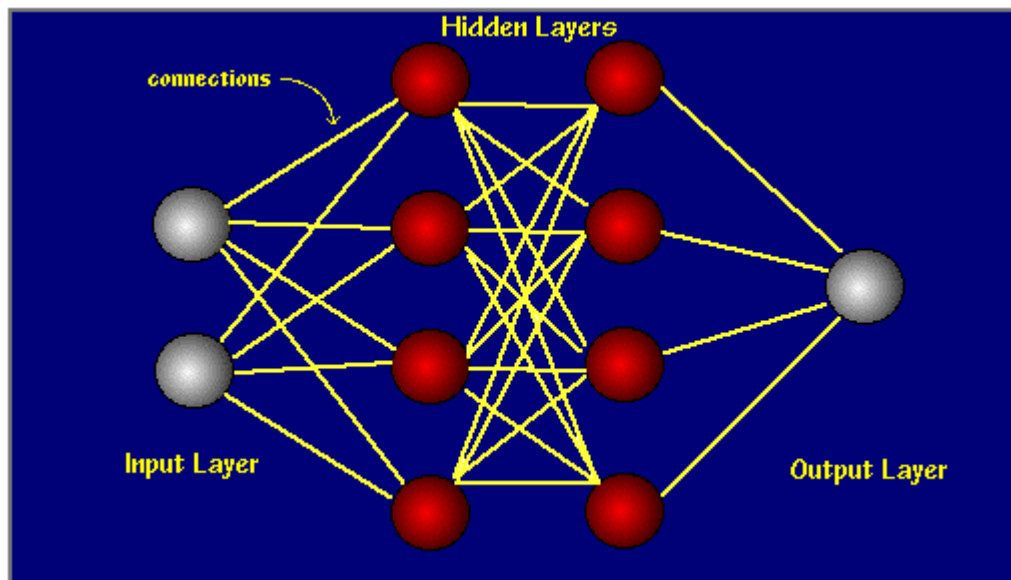


Figure 1. Basic form of Neural Network

The majority of ANNs are based on a 'learning rule' which manipulates the weights of the connections according to the input patterns. In a sense, ANNs learn by example, as do their biological counterparts; a child learns to recognize dogs from examples of dogs.

There is a variety of learning rules that are employed by neural networks. Below an example of the usage of the delta rule is displayed. The delta rule is frequently utilized by the most popular class of ANNs, called 'backpropagational neural networks' (BPNNs). Backpropagation is an abbreviation for the backwards propagation of error.

Learning is a supervised procedure that takes place periodically at each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern) through a forward activation flow of outputs, and the backwards error propagation of weight adjustments. To put it simply, when a neural network is

initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights. More analytically, the process looks something like the Figure 2 below:

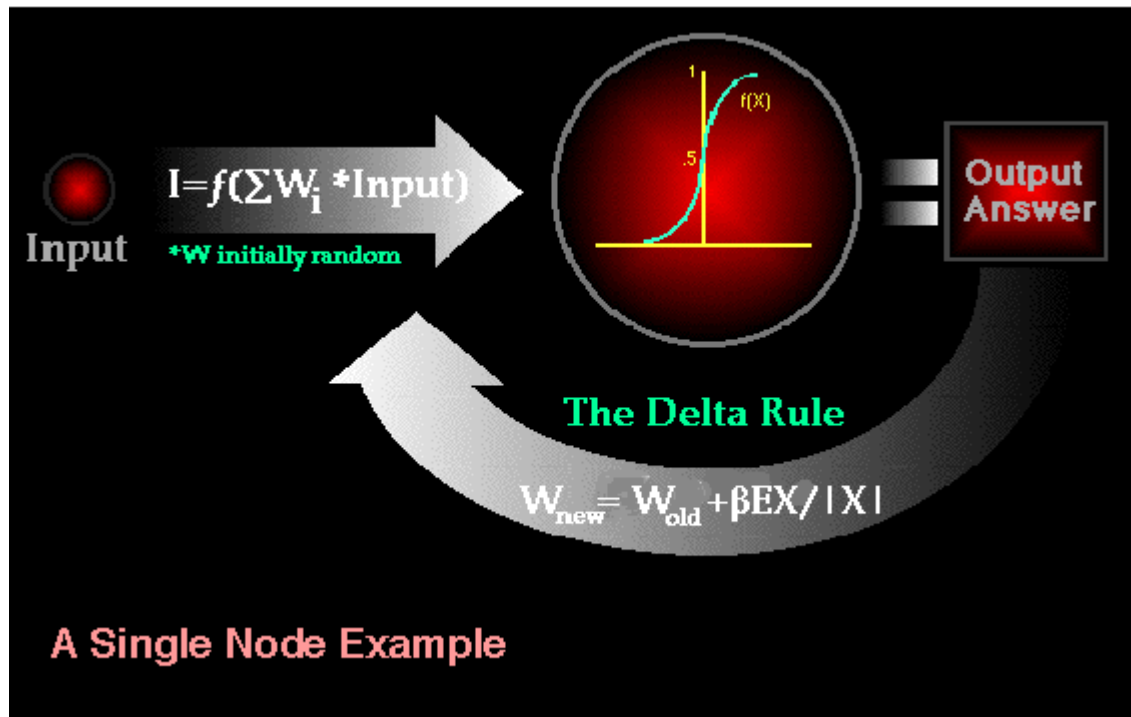


Figure 2. A Single Node Example

Not to forget to mention that inside each hidden layer node there is a sigmoidal activation function which succeeds to stabilize the network through network activity polarization.

Back propagation carries out a gradient descent within the solution's vector space towards a 'global minimum' along the steepest vector of the error surface. The global minimum is the theoretical solution with the lowest possible error. The error surface itself is a hyper paraboloid, but is seldom 'smooth', as is depicted in the graphic below. Indeed, in most problems, the solution space is quite irregular with numerous 'pits' and 'hills' which may cause the network to settle down in a 'local minimum', which is not the best overall solution.

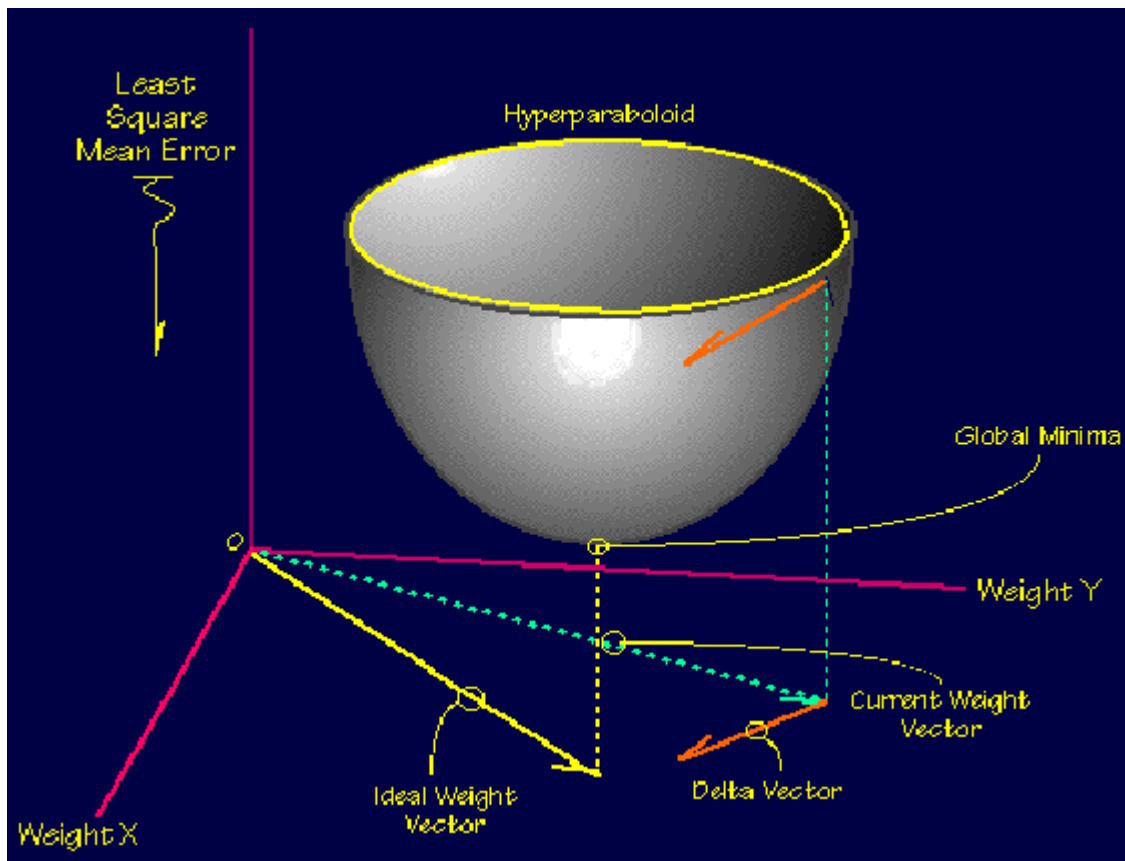


Figure 3. Delta Rule

Due to the fact that the error space cannot be known from the start, neural network analysis is frequently related to a large number of individual executions to find the most appropriate solution. The learning rules used come with built-in mathematical terms to help in this process which control the 'speed' (Beta-coefficient) and the 'momentum' of the learning. The speed of learning is calculated from the rate of convergence between the current solution and the global minimum. Momentum helps the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Upon completion of the neural network 'training', the trained dataset may be used as an analytical tool on other data. In this case no training runs are executed but the network works in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place. However,

at this point the output is retained and no back propagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

The above process sometimes leads to an over trained neural network, which means that the network has been trained exactly to respond to only one type of input; which is much like rote memorization. In real-world applications this situation is not very useful since one would need a separate over trained network for each new kind of input.

3.2.1 ACTIVATION FUNCTIONS

Neuron's transfer function is applied to the weighted sum of its inputs. This function governs the network's behavior. The most common type is sigmoid function. This name comes from its relative similarity with letter S. Different transfer functions are proposed in the literature (Figure 4). The most popular sigmoid functions are *tanh* and *fermi*. The *fermi* activation function equation is:

$$fermi(x) = \frac{1}{1 + \exp(-c)} \quad (3.1)$$

And the related to *fermi tanh* activation function: $tanh(x) = 2ferm(2x) - 1$. Other popular transfer functions are the *piecewise* linear function and the *threshold* function. In most cases transfer function is non-linear, thus called the nonlinearity of the function too.

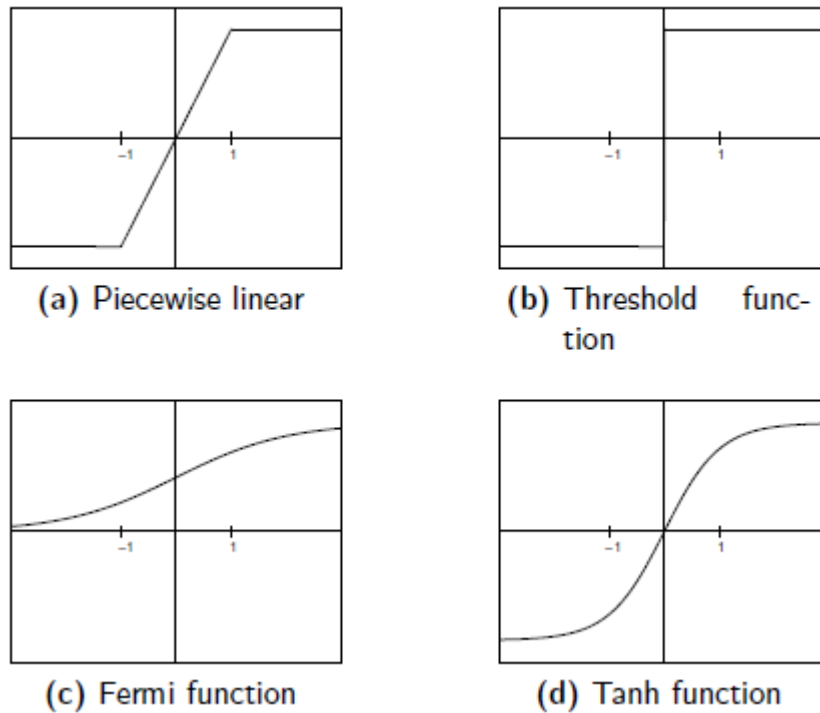


Figure 4. Some common activation functions used in neural networks (Verstraeten, 2009-2010).

A significant characteristic of transfer functions is the squashing effect, which refers to the bounded range values of the neuron. That is, regardless of the magnitude of input signal the magnitude of neuron values remains limited to a specific value range.

Also, the shape of the activation function is crucial for the network's behavior and should be task adapted. Exact requirements and implementation of the task must take into consideration the right choice of activation function.

3.2.2 HOW DO NEURAL NETWORKS DIFFER FROM CONVENTIONAL COMPUTING?

There are some characteristics of artificial neural computing that differ from conventional 'serial' computer and its software process information. For example, a serial computer uses a central processor that has the ability to address an array of memory locations where data and instructions are stored. Computations are made by the processor reading an instruction as well as any

data the instruction requires from memory addresses, then the instruction is executed and the results are saved in a specified memory location as required. In a serial system (and a standard parallel one as well) the computational steps are deterministic, sequential, and logical, and the state of a given variable can be tracked from one operation to another.

On the other hand, ANNs do not necessarily operate sequentially or in a deterministic way like described above. ANNs do not employ complex central processors, rather there are many simple ones which generally do nothing more than take the weighted sum of their inputs from other processors. ANNs do not follow previously programmed instructions; they respond in parallel (either simulated or actual) to the pattern of inputs presented to them. There are also no distinct memory addresses for storing data. Instead, information is contained in the overall activation 'state' of the network. 'Knowledge' is thus represented by the network itself, which is quite literally more than the sum of its individual components.

3.2.3 WHAT APPLICATIONS SHOULD NEURAL NETWORKS BE USED FOR?

Neural networks are considered as universal approximators, and are appropriate for systems that are highly tolerant to error. A neural network is not suitable for balancing one's cheque book. Neural networks are suitable for tracking associations or discovering regularities within a set of patterns, where the volume, number of variables, or diversity of the data is vast and the relationships between variables are complex or the relationships are hard to describe adequately with conventional approaches.

3.3 LEARNING TECHNIQUES

3.3.1 SUPERVISED AND UNSUPERVISED MACHINE LEARNING

In supervised learning, the training data includes training data sets. Each data set is a pair of input vectors and the desired (target) output vectors. This data sets come from empirical observations or are artificially constructed, which represent the desired model behavior.

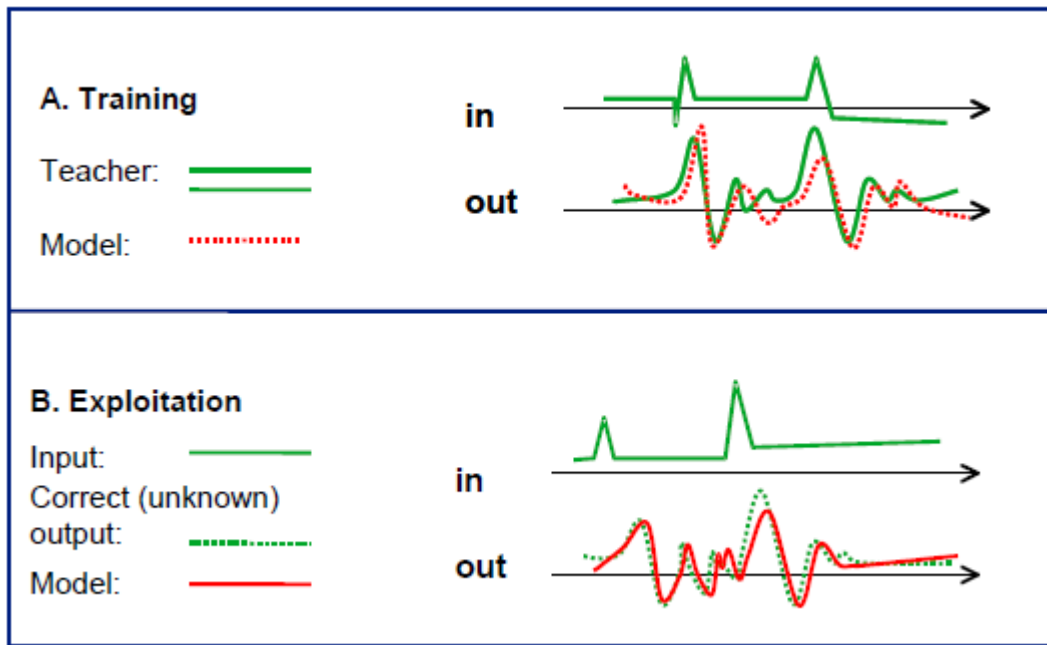


Figure 5. Supervised training process (Jaeger, 2002b).

To train an RNN we use the training (teacher) data for training in order for the output of the network to fit the target output vectors. Then we test the RNN with a novel input which is similar to the training input sequence. We expect the target output to approximate the output of the trained network (Figure 5).

A basic disadvantage of supervised training is overfitting (Figure 6). During the training process network fits the training data too well (extreme case: model duplicates teacher data exactly), but not the underlying function. The result of this process is that the network performs well on the data used during training and poorly with the test data. This is a computational expensive method of learning because we provide the network with vast amounts of training data, in order to improve the learning performance.

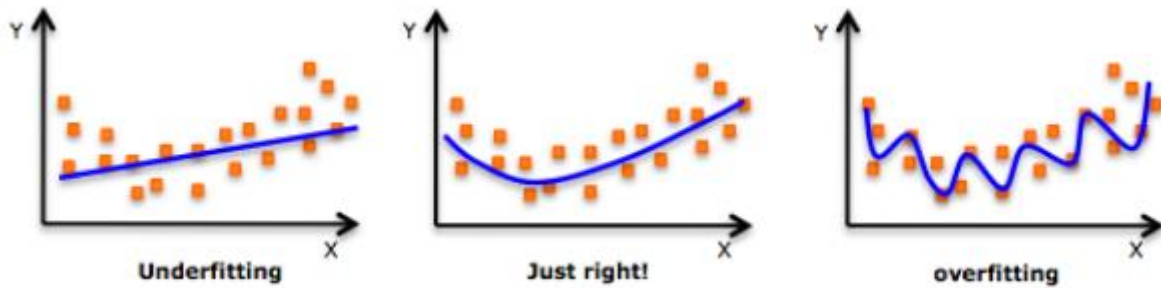


Figure 6. Overfitting: The model is learning the noise on the data instead of generalizing (learning then statistical properties of the data).

Figure 7, below, shows the relationship between model complexity and training and validation errors. The optimal model has the fewest generalization errors, and is marked by a dashed line.

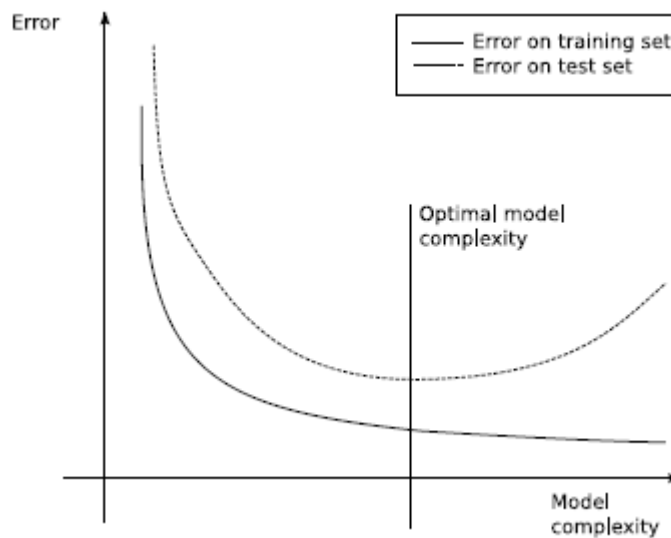


Figure 7. Overfitting: the error on the training set keeps decreasing while the error on the (unseen) test set increases. The model is learning the noise on the data instead of generalizing (learning then statistical properties of the data).

Supervised models are used mainly for:

- Pattern recognition.
- Classification.
- Multi-layer perceptron.

Unsupervised learning on the other hand, is an even less restricted setup. The training data set includes only input sequences, that is, the network is not provided with output data during the training.

An important question is what should the goal of unsupervised learning be? Many authors provide different techniques, which are essentially based on minimizing error or maximizing reward in ML. They include: data compression, clustering, reducing dimensionality while preserving the topology, learning the statistical distribution of the input, minimizing free energy, learning to predict the input, looking for slowly varying (close to invariant) components of the data, sparse representation, maximal information transmission while using minimal energy.

With unsupervised learning it is possible to learn larger and more complex models than with supervised learning. In supervised learning one is trying to find the connection between two sets of observations. The difficulty of the learning task increases exponentially in the number of steps between the two sets and that is why supervised learning cannot, in practice, learn models with deep hierarchies.

3.3.2 REINFORCEMENT LEARNING

Reinforcement learning methods can be expressed as an unbound region of learning. When a model is exposed to an input, model response is evaluated and scored (good response vs. rather bad response), without previous knowledge of correct behavior having been given. The system is provided with clues of desired output behavior, but this information is restricted. In other words, learning process is based on reward-penalty. A physical life example is walking or crawling. These learning algorithms are used in robotics implementations, because it is easier to define a reward signal than a restricted training signal. Complex games are another application example Go (Schraudolph, et al., 1994), because it is almost impossible to define the desired response of the system to every incoming signal.

- It can be used to cluster the input data in classes on the basis of their statistical properties only.
- Cluster significance.
- Labeling.

- The labeling can be carried out even if the labels are only available for a small number of objects representative of the desired classes.
- Kmeans.
- Self-organizing maps.

CHAPTER 4. RECURRENT NEURAL NETWORKS

4.1 OVERVIEW

Artificial recurrent neural networks (RNNs) are a large and varied domain of mathematical models, the architecture of which is trying to simulate biological brain modules. The basic structure element of a neural network is neurons. Neurons are interconnected by synaptic connections (or links) which enable activations to propagate through the network. Basic difference between the more widely used feedforward neural networks (FFNNs) and (RNNs) is that the connection topology includes one or more cycles (Figure 8).

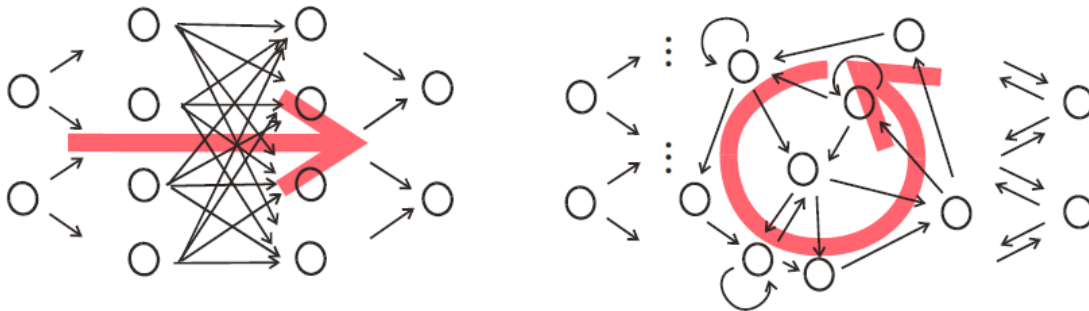


Figure 8. Typical structure of a feedforward network (left) and a recurrent network (right) (Jaeger, 2002b).

The existence of cycles has the following impacts on the network:

- *An RNN may develop self-sustained temporal activation dynamics along with its recurrent connection pathways, even in the absence of input. Mathematically, this renders an RNN a dynamical system, while feedforward networks are functions (Lukosevicius, 2012)*
- *If driven by an input signal, an RNN preserves in its internal state a nonlinear transformation of the input history. In other words, it has a dynamical memory, and is able to process temporal context information (Lukosevicius, 2012).*

The influence of RNN in nonlinear modeling was limited for a long time. The main reason for this stagnancy was that RNN models are to be trained by gradient descent methods, which aim at iteratively reducing the training error.

There is a lot of proposed training algorithms but they have numerous disadvantages:

- It is intrinsically hard to learn dependences requiring long-range memory because the necessary gradient information exponentially dissolves over time (Bengio, et al., 1994)
- Training tasks require advanced algorithms and one must parametrize a lot of global control parameters. Thus, experience is needed for a well-performed network tuning.
- The gradual change of network parameters during learning drives the network dynamics through bifurcations (Doya, 1992).

Some characteristics of feedback networks are:

- Many architectures exist. Activations are fed forward from input to output through "hidden layers" ("Multi-Layer Perceptrons" MLP)
- They are static input-output topologies.
- Backpropagation is the most feedforward supervised training algorithm.
- More than 90% of artificial neural network publications refer to feedforward networks.
- They are good approximators of many practical applications such as nonlinear functions and pattern classifiers.

On the other hand, basic characteristics of Recurrent Neural Networks (RNN) are:

- They have at least one cyclic path of synaptic connections.
- Mathematically consist dynamical systems.
- Many training algorithms have been proposed.
- Theoretical and practical difficulties have prevented practical applications so far.

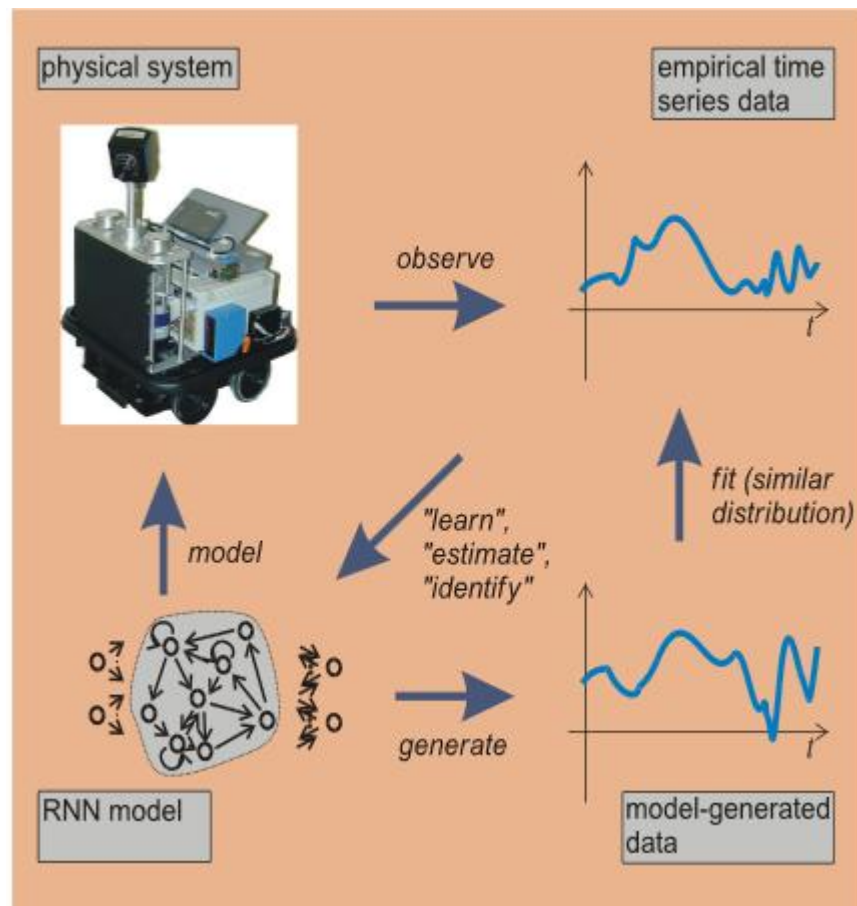


Figure 9. Principal pathway in the black-box modeling (Jaeger, 2002b).

A new approach of RNN design and training was proposed independently by Wolfgang Maass under the name of Liquid State Machines (Maass, et al., 2002) and by Herbert Jaeger under the name of Echo State Networks (Jaeger, 2001). This approach, which had predecessors in computational neuroscience (Dominey, 1995) and subsequent Ramifications in machine learning as the BackPropagation-DeCorrelation (Steil, 2004) learning rule, is now increasingly often referred as Reservoir Computing (RC) (Verstraeten, et al., 2007a). The RC approaches try to overcome the disadvantages of gradient-descent RNN training listed above, by setting up RNNs in the following way (Figure 10):

- The RNN is created randomly and stays unchanged during training. This recurrent neural network is named the reservoir. Reservoir is excited by the input sequence and preserves in its state a highly nonlinear transformation of input history.

- The desired output signal is produced as a linear combination of the neuron's signals from the input-excited reservoir. This linear combination is obtained by linear regression, using the teacher signal as a target.

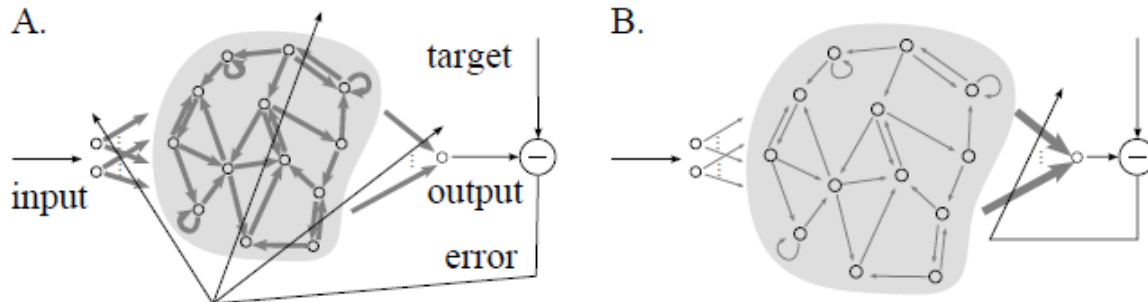


Figure 10. A. Traditional gradient-descent-based RNN training methods adapt all connection weights (bold arrows), including input-to-RNN, RNN-internal, and RNN-to-output weights. B. In Reservoir Computing, only the RNN-to-output weights are adapted (Lukosevicius, 2012).

Reservoir computing methods have become very popular. The main reasons for this development are the following:

- **Modeling accuracy.** Reservoir computing performs better than previous methods of nonlinear identification systems, prediction and classification, for instance, in predicting chaotic dynamics (Jaeger & Haas, 2004), nonlinear wireless channel equalization (two orders of magnitude improvement (Jaeger & Haas, 2004), the Japanese Vowel benchmark (zero test error rate, previous best: 1.8% (Jaeger, et al., 2007a)), financial forecasting (winner of the international forecasting competition NN32), and in isolated spoken digits recognition (improvement of word error rate on benchmark from 0.6% of previous best system to 0.2% (Verstraeten, et al., 2005b) and further to 0% test error in recent unpublished work).
- **Modeling capacity.** RC is used for continuous-time, continuous value real-time systems modeled with bounded resources (including time and value resolution) (Maass, et al., 2003), (Maass, et al., 2006).
- **Biological plausibility.** Numerous connections of RC principles to architectural and dynamical properties of mammalian brains have been established. RC (or closely related models) provides explanations of why biological brains can carry out accurate computations with an

inaccurate" and noisy physical substrate (Buonomano, 1995) (Haeusler & Maass, 2007), especially accurate timing (Karmarkar & Buonomano, 2007).

4.2 FORMALISM

4.2.1 FORMAL DESCRIPTION OF RECURRENT NEURAL NETWORKS

A recurrent neural network consists of neurons (internal units) which are connected by synaptic links whose synaptic strength is coded by weight. These networks usually have input units, internal units and output units. At a given time (n) input units have an activation $u(n)$. Similar activations have the internal $x(n)$ and output units $y(n)$ respectively.

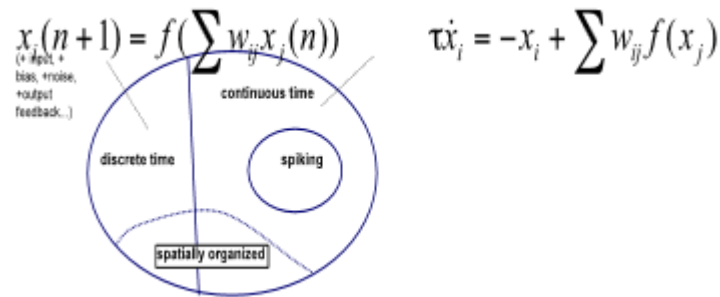


Figure 11. A topology of RNN models

In Figure 11 we can distinguish two major categories of RNN models. One is describing discrete time models over steps $n = 1, 2, 3, \dots$, and the other continuous time models which are defined with differential equations over a continuous time t . Continuous dynamical models which are used for biological modeling usually involve and describe activation signals in the sense of individual action potentials (spikes).

Assume a discrete time RNN model. This model consists of K input units, N internal units and L output units whose activations vectors are respectively:

$$\mathbf{u}(n) = (u_1(n), \dots, u_K(n))^t \quad (4.1)$$

$$\mathbf{x}(n) = (x_1(n), \dots, x_N(n))^t \quad (4.2)$$

$$\mathbf{y}(n) = (y_1(n), \dots, y_L(n))^t \quad (4.3)$$

The input weight matrix \mathbf{w}^{in} size is $N \times K$, internal units \mathbf{w} size is $N \times N$, output weight matrix \mathbf{w}^{out} is $L \times (K \times N)$ and the optional weight matrix \mathbf{w}^{back} of feedback is $N \times L$

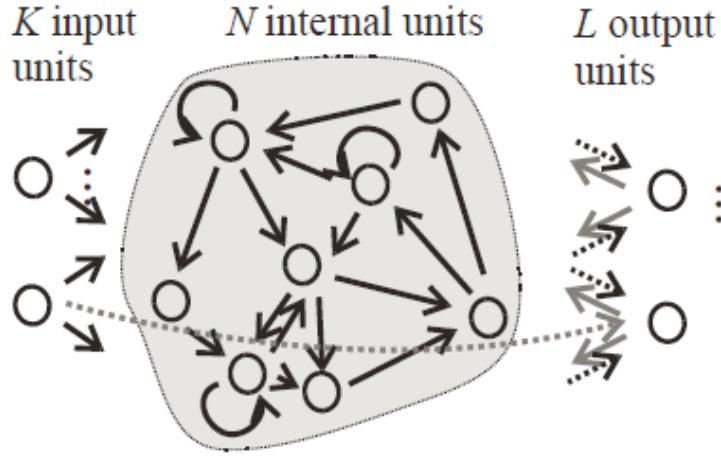


Figure 12. Basic architecture of RNN models. Shaded arrows indicate optional connections. Dotted arrows mark connections which are trained in the "echo state network" approach (in other approaches, all connections can be trained), (Jaeger, 2001).

Output units are allowed to have connections not only to internal units but also (often) to input units and (rarely) to output units.

Updates of internal units activations are calculated according to the equation:

$$\begin{aligned}
x(n+1) &= f(W^{in}u(n+1) + Wx(n) \\
&\quad + W^{back}y(n))
\end{aligned} \tag{4.4}$$

Where f denotes the activation function or unit activation function. Typically, the sigmoid function $f=tanh$ is used. f is applied component-wise.

The output of the network is:

$$y(n) = f^{out}(W^{out}[u(n); x(n); y(n)]) \tag{4.5}$$

Where $[u(n); x(n); y(n)]$ is the concatenation matrix of input, internal and output activation vectors. Output activation function f^{out} is either $f^{out}=tanh$ or $f^{out}=identity$.

4.2.2 APPROACHING A MACHINE LEARNING PROBLEM

A task in machine learning can be defined as a problem of a functional relation between a given input sequence $u(n) \in \mathbb{R}^K$ and a desired (target) $y^{target} \in \mathbb{R}^L$ output sequence, where $n = 1, 2, \dots, T$. T is the number of steps in the data set $\{u(n), y^{target}\}$. When a training set includes both input and target signal it is called supervised machine learning.

If steps included in data set are independent of each other, the goal is to learn a function of the form $y(n) = y(u(n))$, such that the error measure $E(y, y^{target})$ is minimized. Typically, this error is the normalized root-mean-square error (NRMSE)

$$E(\mathbf{y}, \mathbf{y}^{target}) = \sqrt{\frac{\langle \|\mathbf{y}(n) - \mathbf{y}^{target}(n)\|^2 \rangle}{\|\langle \mathbf{y}(n) - \langle \mathbf{y}^{target}(n) \rangle\|^2}} \quad (4.6)$$

Where $\langle \cdot \rangle$ stands for mean.

This is a non-temporal task. Non temporal tasks are memoryless.

On the contrary, a temporal task is to learn a function $\mathbf{y}(n) = (\dots, \mathbf{u}(n-1), \mathbf{u}(n))$ from an input signal $\mathbf{u}(n)$ and a target signal $\mathbf{y}^{target}(n)$ such that $E(\mathbf{y}, \mathbf{y}^{target})$ is minimized. The difference between the temporal and non-temporal task is that the function $y(\cdot)$ we are trying to learn is memoryless in the first case and has memory in the second.

In a temporal task the function to be learned also depends on the input history of the input thus the expansion function has memory: $\mathbf{x}(n) = (\dots, \mathbf{u}(n-1), \mathbf{u}(n))$. Since this function has an unbounded number of parameters, practical implementations often take an alternative, recursive, definition:

$$\mathbf{x}(n) = \mathbf{x}(\mathbf{x}(n-1), \mathbf{u}(n)). \quad (4.7)$$

A well-learned task, or with good precision or accuracy means that the $E(\mathbf{y}, \mathbf{y}^{target})$ is small. Typically, one part of the data points of T is used for training and the rest for testing it.

4.3 RESERVOIR COMPUTING

A basic difference between the traditional design and learning techniques of RNN and Reservoir Computing is that, the second one makes a computational and conceptual separation between a recurrence Dynamic reservoir and a recurrence free (usually linear) readout which produces the desired output from the expansion. The Dynamic Reservoir and the readout serve different purposes, $\mathbf{x}(\cdot)$ expands the input history $\mathbf{u}(n), \mathbf{u}(n-1), \dots$ into a rich enough reservoir state space $\mathbf{x}(n) \in \mathbb{R}^N$, while $y(\cdot)$ combines the internal unit $\mathbf{x}(n)$ into the desired output signal \mathbf{y}^{target} . In the linear readout

case $\mathbf{y}(n) = \mathbf{W}^{out} \mathbf{x}(n) = \mathbf{W}^{out} \mathbf{x}(\mathbf{u}(n))$, for each dimension y_i of \mathbf{y} an output weight vector $(\mathbf{w}^{out})_i$ in the same space \mathbb{R}^N is found such that

$$(\mathbf{w}^{out})_i \mathbf{x}(n) = y_i(n) \approx y_i^{target}(n) \quad (4.8)$$

While the purpose of $\mathbf{x}(n)$ is to contain as much as possible a rich representation of input history (Lukosevicius, 2012). The Reservoir Computing readout is basically a non-temporal function, learning of which is simple.

In RNN training methods, previous to RC, we do not make this separation between reservoir and readout, thus both internal weights and output weights are trained the same manner. Analyses of traditional training algorithms have furthermore revealed that the learning dynamics of internal vs. output weights exhibit systematic and striking differences (Lukosevicius, 2012).

Basic representatives of RC methods are represented below. Each of these methods has its own structure, type of reservoir and specific insights.

4.3.1 ECHO STATE NETWORK

Echo State Network design (Jaeger, 2007b) is based on the principle that if a random RNN possesses certain algebraic properties, training only a linear readout from it is often enough to achieve excellent performance in practical applications. RNN is divided in two parts, the untrained part of ESN is called Dynamic Reservoir which includes the internal units (neurons). These internal units are termed *echoes* of its input history. The readout from the reservoir is the second part which is usually linear.

$$\mathbf{y}(n) = f^{out}(\mathbf{W}^{out}[\mathbf{u}(n); \mathbf{x}(n)]) \quad (4.9)$$

Where $\mathbf{W}^{out} \in \mathbb{R}^{(L) \times (K+N)}$ is the learned output weight matrix, $f^{out}(\cdot)$ is the output neuron activation function (usually the identity) applied component-wise, and $[\cdot; \cdot]$ stands for a vertical concatenation of vectors. The most used batch training method to compute \mathbf{W}^{out} is linear regression.

4.3.2 LIQUID STATE MACHINE

Liquid State Machines (LSMs) (Maass, et al., 2002) developed independently but simultaneously with ESNs. LSMs were developed from a computational neuroscience background, aiming at elucidating the principal computational properties of neural microcircuits (Maass, et al., 2002), (Maass, et al., 2003), (Natschlager, et al., 2002), (Maass, et al., 2004). LSMs use biologically inspired, more sophisticated, models of spiking integration, and fire neurons and dynamic synaptic connection models inside reservoir. In terms referred to LSM design, the reservoir is the liquid, which is the excited states as ripples on the surface of a body of water. LSM input signals usually consist of spike trains. Readouts used are similar to ESNs or multilayer feedforward neural network methods.

4.3.3 EVOLINO

Evolino is a type of RNN with Long Short-Term-Memory ((LSTM)) constructed with units capable of preserving memory for long periods of time. The reservoir weights are trained using evolutionary methods.

4.3.4 BACKPROPAGATION-DECORRELATION

BackPropagation-DeCorrelation (BPDC), was introduced by (Steil, 2004). It approximates and significantly simplifies the APRL method, and only applies it to the output weights \mathbf{W}^{out} , turning it into an online RC method. BPDC uses the reservoir update equation defined in eq. (4.4), where output feedbacks \mathbf{w}^{back} are essential, with the same type of units as ESNs. BPDC learning is claimed to be insensitive to the parameters of fixed internal units (reservoir) weights \mathbf{W} .

CHAPTER 5. ECHO STATE NETWORK

5.1 OVERVIEW

The main idea of an ESN [Jaeger, 2001, 2002b] is (i) to drive a random, large, fixed, recurrent neural network with the input signal thereby inducing in each neuron within this “reservoir” network a nonlinear response signal, and (ii) to create a desired output signal by a trainable linear combination of all of these response signals. The internal weights of the underlying reservoir network are not changed by the learning; only the reservoir-to-output connections are trained.

This means that in order to produce a ‘rich’ set of dynamics inside reservoir, reservoir N must be large with order ranging from ten to thousands, the internal weight matrix \mathbf{W} must be sparse up to 20% connections, and the weights of the connections are chosen from a uniform distribution symmetric around zero.

The optionally feedback weights \mathbf{W}^{back} and input weights \mathbf{w}^{in} are chosen to be either dense or sparse and generated randomly from a uniform distribution. Input scaling and shifting (a constant value added to $\mathbf{u}(n)$ of the input signal must be ‘tuned’ manually. The magnitude of these values depends on how non linearity of the processing unit is needed for the task. If the inputs are far from zero the \tanh internal units tend to drive activations more towards saturation where they exhibit more nonlinearity, while for inputs that are close to zero, \tanh internal units tend to operate with activations close to zero. Input shifting sometimes helps to overcome undesired consequences of the symmetry around zero of the internal units.

5.2 BASIC ECHO STATE NETWORK MODEL

The basic echo state network model is described in this section (Figure 13). We assume a discrete-time neural networks with K input units, N internal network units and L output units. Activation of input units at time step n are $\mathbf{u}(n) = (u_1(n), \dots, u_K(n))$ of internal units are $\mathbf{x}(n) = (x_1(n), \dots, x_N(n))$, and $\mathbf{y}(n) = (y_1(n), \dots, y_L(n))$ of output

units. Real-valued connection weights are collected in a $N \times K$ weight matrix $\mathbf{w}^{in} = (w_{ij}^{in})$ for the input weights, in a $N \times N$ matrix $\mathbf{W} = (W_{ij})$ for the internal connections, in an $L \times (K + N + L)$ matrix $\mathbf{W}^{out} = (w_{ij}^{out})$ for the connections to the output units, and in a $N \times L$ matrix $\mathbf{W}^{back} = (w_{ij}^{back})$ for the connections that project back from the output to the internal units, and in a $N \times L$ matrix $\mathbf{W}^{back} = (w_{ij}^{back})$ for the connections that project back from the output to the internal units. $v(n)$ is the added noise of step n .

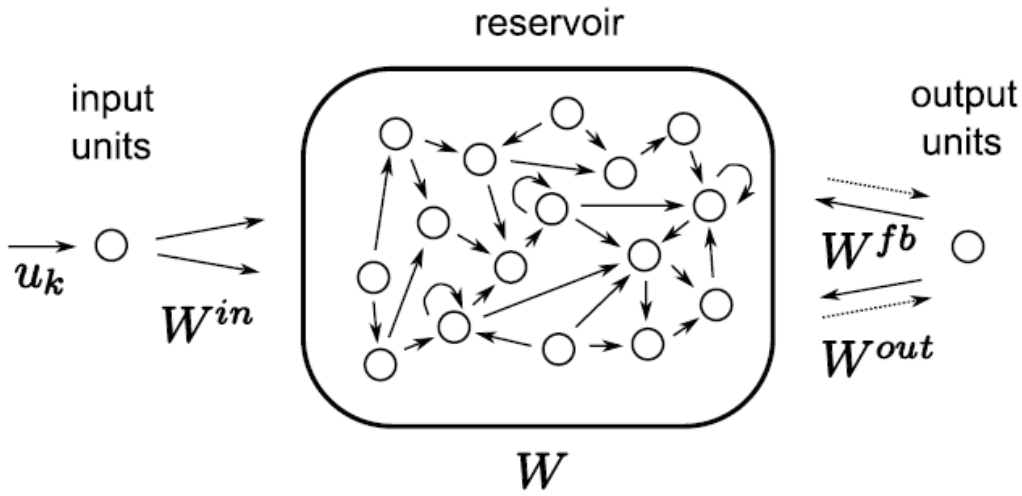


Figure 13. Basic echo state network architecture. Dashed arrows indicate connections that are optional (Yildiz, et al., 2012).

Input to output unit direct connections and connections between outputs are permitted. This basic architecture does not include layered structure of the reservoir.

The activation of internal units is updated according to:

$$\begin{aligned}
x(n+1) &= f(W^{in}u(n+1) + Wx(n) \\
&\quad + W^{back}y(n) + v(n+1))
\end{aligned} \tag{5.1}$$

Where $f = (f_1, \dots, f_N)$ are activation functions of internal units. Usually are sigmoid functions.

The readout from the reservoir is calculated according to:

$$\begin{aligned}
y(n+1) &= f^{out}(W^{out}[u(n+1); x(n) \\
&\quad + 1); y(n)])
\end{aligned} \tag{5.2}$$

Where $f^{out} = f_1^{out}, \dots, f_L^{out}$ are activation functions of internal units. Typically, sigmoid or identity function used and applied element-wise.

The readout can be written as follows without output to output connections

$$y(n+1) \tag{5.3}$$

Where W^{out} is the learned output matrix, $[\cdot; \cdot]$ denotes vertical concatenation of vectors and the column vector of ones is an optional bias input. To compute W^{out} we use linear regression.

5.3 RESERVOIR READOUTS

This is a well-known section in machine learning. Conceptually, we want to “filter” a readout from the reservoir which is a supervised non-temporal task of mapping $x(n)$ to $y^{target}(n)$. Several methods are available, one can use his favorite.

5.3.1 SINGLE-LAYER READOUT

5.3.1.1 LINEAR REGRESSION

The readout of the described network is a single-layer readout. Also, we assume that the output activation function is linear, then the equation can be written in a matrix notation as:

$$\mathbf{Y} = \mathbf{W}^{out} \mathbf{X} \quad (5.4)$$

Where \mathbf{Y} are all $\mathbf{y}(n)$ and \mathbf{X} are all $[1; \mathbf{u}(n); \mathbf{x}(n)]$ produced by presenting the reservoir with $\mathbf{u}(n)$, both are collected into respective matrices over the training period $n = 1, \dots, T$. Due to initial transients the data of the first steps of the training run are discarded. In most cases the task requires to minimize the quadratic error $E(\mathbf{Y}^{target}, \mathbf{X})$. Since our goal is to find the optimal weight \mathbf{W}^{out} that minimizes the squared error between $\mathbf{y}(n)$ and $\mathbf{y}^{target}(n)$ ($\mathbf{y}^{target}(n) \approx \mathbf{y}(n)$) we solve the linear system equations:

$$\mathbf{Y}^{target} = \mathbf{W}^{out} \mathbf{X} \quad (5.5)$$

Where \mathbf{Y}^{target} are all $\mathbf{y}^{target}(n)$. The system is overdetermined because $T > 1 + K + N$.

One direct method to solve this system is calculating the Moore-Penrose pseudoinverse \mathbf{X}^+ of \mathbf{X} and \mathbf{W}^{out} as:

$$\mathbf{W}^{out} = \mathbf{Y}^{target} \mathbf{X}^+ \quad (5.6)$$

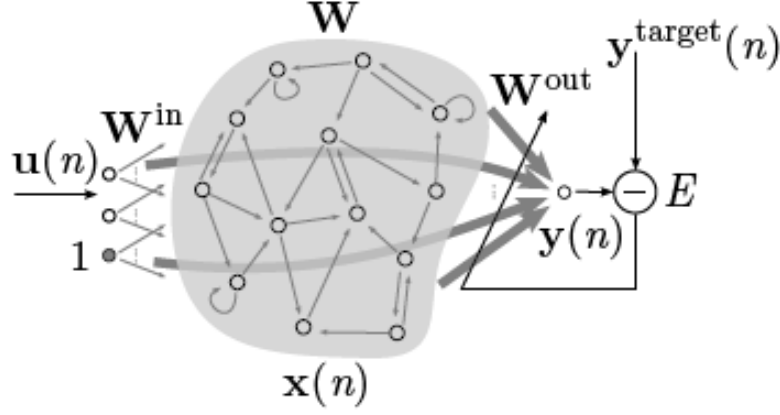


Figure 14. An echo state network. (Lukosevicius, 2012)

Calculations of direct pseudoinverse are memory expensive for large state-collecting matrices \mathbf{X} but is numerical stable.

This issue can be overcome by reforming eq. (5.6) in the normal equations formulation:

$$\mathbf{W}^{out} \mathbf{X} \mathbf{X}^t = \mathbf{Y}^{target} \mathbf{X}^t \quad (5.7)$$

A solution of eq. (5.7) would be:

$$\mathbf{W}^{out} = \mathbf{Y}^{target} \mathbf{X}^t (\mathbf{X} \mathbf{X}^t)^{-1} \quad (5.8)$$

The method eq. (5.8) has lower numerical stability, compared to eq. (5.6). In addition, this method enables one to introduce ridge, or Tikhonov, regularization elegantly:

$$\mathbf{W}^{out} = \mathbf{Y}^{target} \mathbf{X}^t (\mathbf{X} \mathbf{X}^t + \beta \mathbf{I})^{-1} \quad (5.9)$$

Where β is a regularization coefficient explained, and \mathbf{I} is the identity matrix.

Large weight values reveal that \mathbf{W}^{out} amplifies small differences between the dimensions of internal states $\mathbf{x}(n)$ and can be very sensitive to deviations from the exact conditions in which the network has to be trained. When a network topology uses its output as the next input (using feedback connection) this is a big problem. This tiny difference between the output and the expected value becomes bigger in the next steps.

To minimize the effect of feedback instability or if there is a chance of overfitting we use ridge regression. Instead of just minimizing RMSE, ridge regression eq. (5.9) solves:

$$\mathbf{W}^{out} = \underset{\mathbf{W}^{out}}{argmin} \frac{1}{L} \sum_{i=1}^L \left(\sum_{n=1}^T ((y_i)(n) - y_i^{target}(n))^2 + (\beta \|\mathbf{w}_i^{out}\|)^2 \right) \quad (5.10)$$

Where \mathbf{w}_i^{out} is the $i - th$ row of \mathbf{W}^{out} and $\|\cdot\|$ stands for the Euclidean norm. The objective function in eq. (5.10) adds a regularization, or weight decay and the term $(\beta \|\mathbf{w}_i^{out}\|)^2$ limits large sizes of \mathbf{W}^{out} to the square error between $\mathbf{y}(n)$ and $\mathbf{y}^{target}(n)$. This is a sum of two objectives, a compromise between having a small training error and small output weights. The relative "importance" between these two objectives is controlled by the regularization parameter.

It is not necessary to rerun the ESN with training data for every value β , because no variable of the equation eq. (5.9) is affected. Optimal values of β can be found in different ranges and depend on the exact instance of the reservoir and length of training sequence too.

The solution of the equation eq. (5.9) with $\beta = 0$ is

$$\mathbf{W}^{out} = \mathbf{Y}^{target} \mathbf{X}^t (\mathbf{X} \mathbf{X}^t)^{-1} \quad (5.11)$$

and to remove regularization. Equation eq. (5.10) is now equivalent to the RMSE function and the ridge regression is now a regular linear regression.

Setting $\beta = 0$ sometimes leads to numeric instabilities when inverting $\mathbf{X}\mathbf{X}^t$ in eq. (5.9). Numeric instabilities can be avoided by using a pseudoinverse $(\mathbf{X}\mathbf{X}^t)^t$ instead of the real inverse $(\mathbf{X}\mathbf{X}^t)^{-1}$ in eq. (5.9).

In (Jaeger, 2002b) it is recommended to add noise to the internal units $\mathbf{x}(n)$. This practice has a similar effect as Tikhonof regularization. Adding noise to the internal units makes reservoir less sensitive and the output learns to recover from perturbed signals so the network is more stable to feedback loops.

5.3.1.2 WIENER-HOPF SOLUTION

An even faster (but less computationally stable) alternative solution is Wiener-Hopf equations and calculation of $\mathbf{B} = (\mathbf{A}\mathbf{C}^t)(\mathbf{C}\mathbf{C}^t)^{-1}$. Since the $(\mathbf{U}|\mathbf{X})$ matrix has typically many more columns than rows $K + N \ll T$, we get a much smaller autocovariance matrix $[\mathbf{U}|\mathbf{X}][\mathbf{U}|\mathbf{X}]^t \in \mathbf{R}^{(K+N) \times (K+N)}$ whose inverse we need to calculate. In most cases (namely, when the condition number of $\mathbf{C}\mathbf{C}^t$ has a reasonably small size) this method gives similar results to the ones calculated by the QR factorization. When training only a subset $S = \{1, \dots, N_y\}$ of the output (dimensions) $y_s(n)$ at a time, only the corresponding rows of \mathbf{y}^{target} are used and thus only the corresponding rows $\mathbf{w}_s^{out} = (f^{out})^{-1}(Y_s^{target})[\mathbf{U}|\mathbf{X}]^+$ of the output weights are calculated.

5.4 ECHO STATE PROPERTY

To make an ESN network work properly, the reservoir should have the echo state property (Jaeger, 2001). Having echo states (or not having them) is a property of the network prior to training, that is, a property of the weight matrices \mathbf{W}^{in} , \mathbf{W} , and (optionally, if they exist) \mathbf{W}^{back} . We require that the training input vectors $\mathbf{u}(n)$ come from a compact interval U and the training output vectors $\mathbf{y}^{target}(n)$ from a compact interval D . Mathematical definition of echo states is as follows:

Definition (echo states). Assume an untrained network with weights \mathbf{W}^{in} , \mathbf{W} , and \mathbf{W}^{back} is driven by teacher input $\mathbf{u}(n)$ and teacher-forced by teacher output $\mathbf{y}^{target}(n)$ from compact intervals U and D . The network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$ has echo states with regard to U and D , if for every left-infinite input/output sequence $(\mathbf{u}(n), \mathbf{y}^{target}(n-1))$, where $n =$

$\dots, -2, -1, 0$, and for all state sequences $x(n), x'(n)$ compatible with the teacher sequence, i.e. with:

$$\begin{aligned} x(n+1) &= f(W^{in}u(n+1) + Wx(n) \\ &\quad + W^{back}y^{target}(n) + v(n+1)) \end{aligned} \quad (5.12)$$

$$\begin{aligned} x'(n+1) &= f(W^{in}u(n+1) + Wx'(n) \\ &\quad + W^{back}y^{target}(n) + v(n+1)) \end{aligned} \quad (5.13)$$

it holds that $x(n) = x'(n)$ for all $n \leq 0$.

Intuitively, the echo state property says, "if the network has been run for a very long time [from minus infinity time in the definition], the current network state is uniquely determined by the history of the input and the (teacher-forced) output (Jaeger, 2002b)

A condition (forgetting property) equivalent to echo states, determines that the effects on initial network wash out over time (Jaeger, 2002b)

These conditions essentially determine, that the effect of a state $x(n)$ and or the previous $u(n)$ on a future state $x(n+k)$ should 'decay' gradually as time passes and not get amplified or be stable. For most practical purposes, echo state property assures if the largest absolute eigenvalue $|\lambda_{max}|$ of the reservoir weight matrix W is $|\lambda_{max}| < 1$. The largest absolute eigenvalue $|\lambda_{max}|$ is the spectral radius ρ of weight matrix W .

A common computational method to achieve an internal weight matrix W with spectral radius less than one is:

1. Randomly generate an internal weight sparse matrix \mathbf{W}_0 with zero mean value of weights.
2. Normalize \mathbf{W}_0 to a matrix \mathbf{W}_1 with unit spectral radius by putting $\mathbf{W}_1 = 1/|\lambda_{max}|\mathbf{W}_0$, where $|\lambda_{max}|$ is the spectral radius of \mathbf{W}_0 . Scale \mathbf{W}_1 to $\mathbf{W} = \rho \mathbf{W}_1$, where $\rho < 1$, whereby \mathbf{W} obtains a spectral radius of ρ .

The optimal value of the spectral radius is crucial for the performance of the network and should be task adapted. This means that we must set the magnitude of spectral radius considering the profile of memory and nonlinearity the learning task require. The internal timescale of the dynamics of the dynamic reservoir state is connected to ρ [jaeger, 2003]. Spectral radius (Jaeger, 2001) should be close to 1 for learning tasks that require long short-term memory and smaller for the tasks where a too long memory might be harmful. Small ρ means that one has a fast Dynamic Reservoir, on the contrary close to unit ρ has a slow Dynamic Reservoir. For example, if one wishes to train a sine generator, one should use a small ρ for fast sinewaves and a large ρ for slow sinewaves (Jaeger, 2002b). A considerable effect of large spectral radius is that it drives internal units $\mathbf{x}(n)$ into more nonlinear regions of \tanh units similarly to \mathbf{W}^{in} . Thus scalings of both \mathbf{W}^{in} and \mathbf{W} have a similar effect on nonlinearity of the ESN, with a difference that scaling up \mathbf{W} makes reservoir unstable, while their difference determines the effect of current versus past inputs on the current state (Lukosevicius, 2012).

A question that we must answer, is why a Dynamic reservoir must have the echo state property in order to work appropriately.

Engineering speaking, the unknown system's dynamics are ruled by the update equation

$$\begin{aligned} \mathbf{y}^{target} = e(\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{y}^{target}(n-1), \mathbf{y}^{target}(n-2)) \end{aligned} \quad (5.14)$$

Where e is a (possibly highly complex) nonlinear function of the previous inputs and system outputs of deterministic, stationary system.

For modeling a specific task with unknown equation system (black-box) accurately, it relies on how good the approximation of the system function e is. Assume an ESN with output linear function, that its trained output of the network is a linear combination of the internal units of the network.

$$e(\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{y}^{target}(n-1), \mathbf{y}^{target}(n-2), \dots, \mathbf{y}^{target}(n-2)) \approx \mathbf{y}(n) \quad (5.15)$$

$$= \sum w_i^{out} x_i(n) \quad (5.16)$$

$$\sum w_i^{out} e_i(\mathbf{u}(n), \mathbf{u}(n-1), \dots, \mathbf{y}^{target}(n-1), \mathbf{y}^{target}(n-2), \dots, \mathbf{y}^{target}(n-2)) \quad (5.17)$$

Above equation makes clear how the desired approximation of the system function is a linear combination of echo functions e_i . Arguments e_i and e represent the same thing: collections of previous inputs and readouts of the network.

5.5 EXAMPLE OF A SMALL TIMER NETWORK

Assume an input-output network. We want to train this network to act as a timer (Figure 15). For this network we consider two inputs (u_1, u_2) and one output y_1 . First input $u_1(n)$ sometimes jumps to 1 but at most of the time is 0. Second input $u_2(n)$ values range from 0.1 to 1 with specific step 0.1. Each time input $u_1(n)$ jumps to 1 second input $u_2(n)$ takes a new random value of the second input range. The target output is 0.5 for $10 \times u_2(n)$ time steps after $u_1(n)$ was 1, otherwise is zero. This implementation indicates a timer: $u_1(n)$ gives the "start" activation for the timer, $u_2(n)$ gives the desired duration.

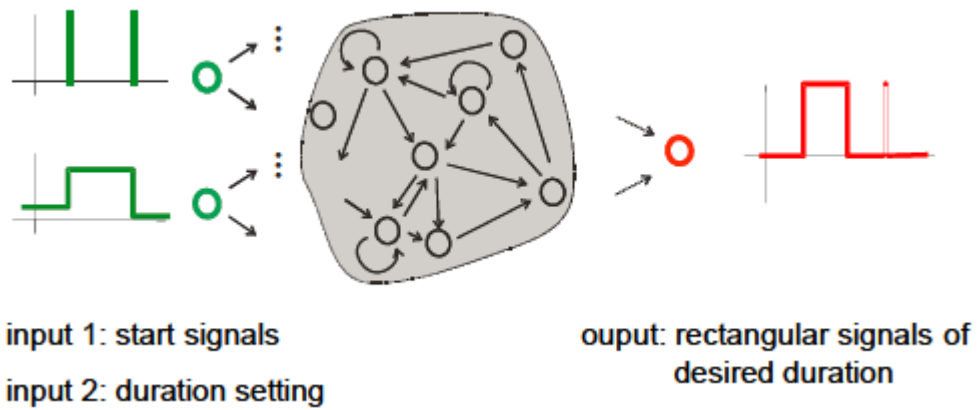


Figure 15. Timer network

Traces of inputs, outputs, teacher output and readout of the network, if represented, are in the following Figure 16:

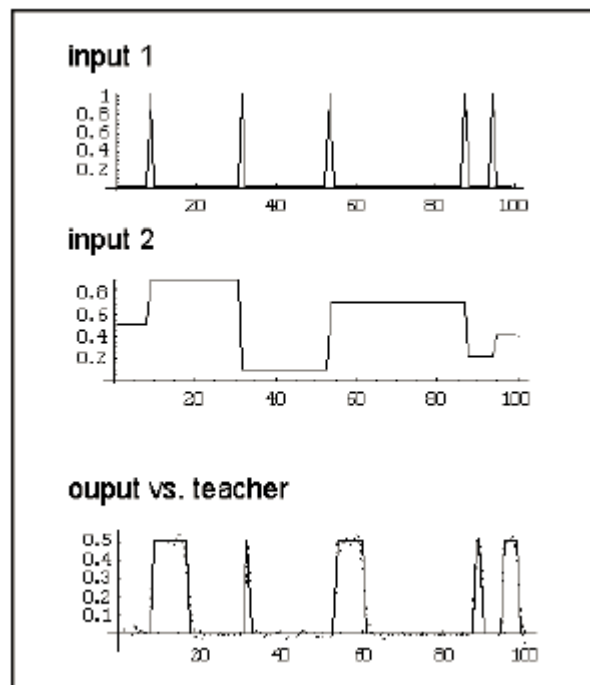


Figure 16. Performance of the network. Dotted line in last graph shows network's output and solid line desired or teacher output.

This learning task example reveals the major property of the network, which must have a kind of memory. Network must act as memory in order to 'store' information about the 'start' signal and the duration of as many time steps as possible. This is possible because of internal units "echoing" the inputs signals.

5.6 SHORT-TERM MEMORY

Most of the tasks in control and furthermore in signal analysis require system models with significant STM spams. By STM we understand memory effects connected with the transient activation dynamics of network (Jaeger, 2002b), or in other words the property of some input- output systems, where the current output $\mathbf{y}(n)$ depends on earlier values $\mathbf{u}(n - k)$ of the input and/or earlier values $\mathbf{y}(n - k)$ of the output itself (Figure 17). RNNs are dynamical systems with high dimensional internal state $\mathbf{x}(n)$. The state $\mathbf{x}(n)$ preserves some kind of information of the input history. Such an engineer task is suppressing *echos* in telephone channels.

The dynamic reservoir activation units $x_i(n)$ is, a sort of, echo functions e_i of the past input-output history to the current state.

$$e_i: (U \times D)^{-\mathbb{Z}} \rightarrow \varepsilon \quad (5.18)$$

$$(\dots, (u(-1), \mathbf{y}^{target}(-2)), (u(0), \mathbf{y}^{target}(-1))) \quad (5.19)$$

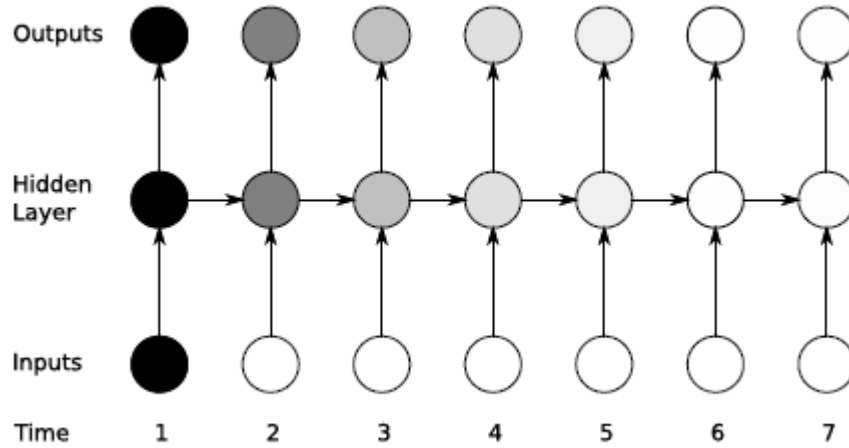


Figure 17. Short Term Memory of RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the reservoir, and the network 'forgets' the first inputs (Graves, 2008).

5.7 TRAINING AN ESN AS A DELAY LINE

A question is revealed, how many of the previous steps ($\mathbf{u}(n - k), \mathbf{y}(n - k - 1)$) are relevant to echo state function, or how long is the short-term memory of an ESN effective.

We train ESN as a pure STM task. For the task we use an ESN with one input and many output units. The input $\mathbf{u}(n)$ is a white noise signal generated by sampling at each time independently from a uniform distribution over $[-0.5, 0.5]$. We consider delays $= 1, 2, \dots$. For each delay k , we train a separate output unit with the training signal $y_k^{target}(n) = \mathbf{u}(n - k)$. Network has no feedback connections, so all output units can be trained simultaneously and independently from each other.

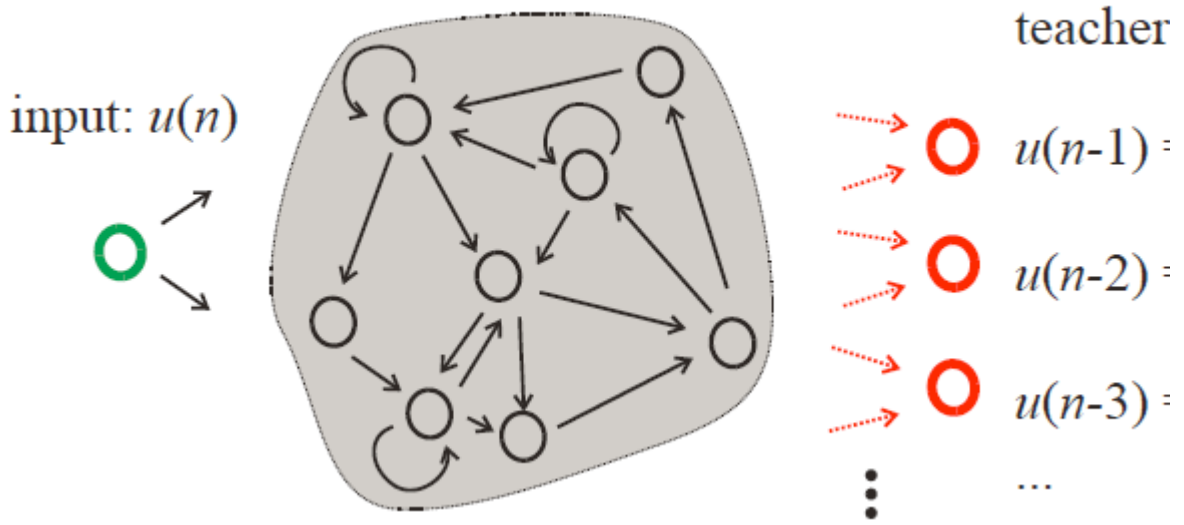


Figure 18. Setup of delay learning task.

Where $\mathbf{u}(n-1) = y_0^{target}(n)$, $\mathbf{u}(n-2) = y_1^{target}(n), \dots$

A Dynamic reservoir of 20 units was used, with connectivity 15%, that is, 15% of the weight matrix are non-null elements and were sampled randomly from a uniform distribution over $[-1,1]$. Spectral radius was $\rho = 0.8$. The input weights were set to values of -0.1 or $+0.1$ with equal probability. We trained 4 output units with delays of $k = 4, 8, 16, 20$. The training was done over 300 time steps, of which the first 100 were discarded to wash out initial transients. On test data, the trained network showed testing mean square errors of $MSE_{test} = 0.0000047, 0.00070, 0.040, 0.12$ for the four trained delays. Figure 19 (upper diagrams) shows an overlay of the correct delayed signals (solid line) with the trained network output.

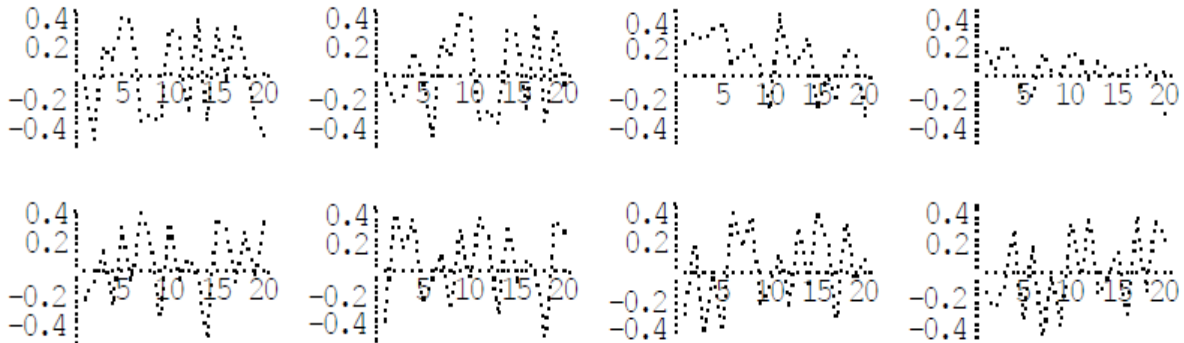


Figure 19. 20 with a 20-unit DR. Results of training delays $k = 4, 8, 16$. Top row: input weights of size -0.1 or $+0.1$, bottom row: input weights sized -0.001 or $+0.001$.

When the same experiment is rerun with the same DR, but with much smaller input weights set to random values of -0.001 or $+0.001$, the performance greatly improves: testing errors $MSE_{test} = 0.000035, 0.000038, 0.000034, 0.0063$ are now obtained.

Three fundamental observations can be gleaned from this simple example:

1. The network can master the delay learning task, which implies that the current network state $\mathbf{x}(n)$ retains extractable information about previous inputs $\mathbf{u}(n - k)$.
2. The longer the delay, the poorer the delay learning performance.
3. The smaller the input weights, the better the performance.

5.8 MEMORY CAPACITY

The most important function of the reservoir is to keep the memory of the previous inputs. Some results concerning the ESNs networks are listed below (Jaeger, 2001):

Theorem 1. In a network whose DR has N nodes, $MC \leq N$. That is, the maximal possible memory capacity is bounded by DR size.

This correlation-based measure of short-term memory capacity, evaluating how well $\mathbf{u}(n)$ can be reconstructed by the reservoir as $\mathbf{y}(n + k)$ after various delays k , was introduced in [Jaeger, 2002a]. We define the memory capacity MC of a network as:

$$MC = \sum_{k=1}^{\infty} r^2(\mathbf{u}(n - k), \mathbf{y}_k(n)) \quad (5.20)$$

Where correlation coefficient $r(\mathbf{u}(n - k), \mathbf{y}_k(n))$ between the correct delayed signal $\mathbf{u}(n - k)$ and the network output $\mathbf{y}_k(n)$ of the unit trained on the delay k . It ranges between -1 and 1 . By squaring it, we obtain a quantity called the determination coefficient $r^2(\mathbf{u}(n - k), \mathbf{y}_k(n))$. It ranges between 0

and 1. A value of 1 indicates perfect correlation between correct signal and network output, a k value of 0 indicates complete loss of correlation. Perfect recall of the k -delayed signal would thus be indicated as $r^2(\mathbf{u}(n-k), \mathbf{y}_k(n)) = 1$, complete failure as $r^2(\mathbf{u}(n-k), \mathbf{y}_k(n)) = 0$.

Theorem 2. In a linear network with N nodes, generically $MC = N$. That is, a linear network will generically reach maximal network capacity. Notes: (i) a linear network is a network whose internal units have a linear transfer function, i.e. $f = id$. (ii) "Generically" means: if we randomly construct such a network, it will have the desired property with probability one.

All of the above memory measurement experiments are accomplished with only one-dimensional $K = 1$ input $\mathbf{u}(n)$. Memory of a multidimensional input in the reservoir was investigated in (Hermans & Schrauwen, 2010). Results show that the shape of the memory curve depends on the spectral radius $|\lambda_{max}|(\mathbf{W})$: reservoirs with small $|\lambda_{max}|(\mathbf{W})$ have precise memory of the recent input which drops sharply with delay k , while those with big $|\lambda_{max}|(\mathbf{W})$ have a more extended memory at the expense of precision. The same limit N of the correlation-based memory capacity applies, which input dimensions have to share. The individual principal components of the input have memory capacity roughly proportional to the square root of their variance, indicating that a lot of memory is spent for non-principal components.

5.9 GENERIC RESERVOIR "RECIPES"

5.9.1 DIFFERENT TOPOLOGIES OF THE RESERVOIR

Many authors have proposed different topologies of the ESN reservoir from sparsely and randomly connected ones. Specifically, small-world (Watts, 1998), scale-free (Barabasi & Albert, 1999), and biologically inspired connection topologies generated by spatial growth (Kaiser & Hilgetag., 2004). Eigenvalue spread of the cross-correlation matrix of the activations $\mathbf{x}(n)$ and the NRMS error were used to evaluate the results of these topologies. The investigation concludes that *'(. . .) none of the investigated network topologies was able to perform significantly better than simple random networks, both in terms of eigenvalue spread as well as testing error'* (Liebald, 2004).

5.9.2 INTERNAL UNITS ACTIVATION METHODS

5.9.2.1 LEAKY INTERGRATOR NEURONS

Standard sigmoid unit networks have the disadvantage that they do not include a time constant. This means that dynamics cannot be “slowed down” like the dynamics of a differential equation. *The units in standard sigmoid networks have no memory; their values at time $n + 1$ depend only fractionally and indirectly on their previous value. Thus, these networks are best suited for modeling intrinsically discrete-time systems with a “computational”, “jumpy” flavor. It is difficult, for instance, to learn slow dynamics like very slow sine waves. For learning slowly and continuously changing systems, it is more adequate to use networks with continuous dynamics. Using a standard sigmoid network it is almost impossible to obtain a very slow dynamics ESN generator (Jaeger, 2001).*

Excluding the basic sigmoid units of ESN, there is another implementation of internal units (Jaeger, 2001) introduced as leaky integrator neurons. This type of internal activation units incorporates a “leaky” integration of its activation from previous time steps.

We assume an Echo State network with K inputs, N reservoir activation units and L output units. $\mathbf{u} = \mathbf{u}(t)$ denotes the K -dimensional input vector, $\mathbf{x} = \mathbf{x}(t)$ the N dimensional reservoir activation state, $\mathbf{y} = \mathbf{y}(t)$ the L -dimensional output vector, \mathbf{W}^{in} , \mathbf{W} , \mathbf{W}^{out} and \mathbf{W}^{back} the input / internal / output / output feedback connection weight matrices of sizes $N \times K$, $N \times N$, $L \times (K + N)$ and $N \times L$, respectively. Then the continuous-time dynamics of a leaky integrator ESN which accumulates (integrates) its inputs, but also exponentially loses (leaks) accumulate excitation over time is given by:

$$\dot{\mathbf{x}} = \frac{1}{c}(-\alpha \mathbf{x} + f(\mathbf{W}^{in} + \mathbf{W}\mathbf{x} + \mathbf{W}^{back}\mathbf{y})) \quad (5.21)$$

$$\mathbf{y} = g(\mathbf{W}^{out}[\mathbf{x}; \mathbf{u}]) \quad (5.22)$$

Where $c > 0$ is a global time constant which rules the speed of dynamics, $a > 0$ is the decay (leakage) rate of internal units (we assume a uniform leaking rate for simplicity), f is a sigmoid function (we will use \tanh), g is the output activation function (usually the identity or a sigmoid) and $[;]$ denotes vector concatenation.

Using Euler (linear) interpolation of above equation with step size δ we obtain the following discrete time internal units updated equation with a given discrete time sample input $\mathbf{u}(n\delta)$:

$$\begin{aligned} \mathbf{x}(n+1) &= \left(1 - \frac{\alpha\delta}{c}\right) \mathbf{x}(n) \\ &+ \frac{\delta}{c} f\left(\mathbf{W}^{in} \mathbf{u}((n+1)\delta) + \mathbf{W} \mathbf{x}(n)\right) \end{aligned} \quad (5.23)$$

$$\mathbf{y}(n) = g(\mathbf{W}^{out}[\mathbf{x}(n); \mathbf{u}(n\delta)]) \quad (5.24)$$

Since we only consider simulations of network here, training data provided in a discrete time domain and the sampling period δ_0 is known. We assume (Jaeger, et al., n.d.) that δ has been suitably fixed. This assumption allow us to write $\mathbf{u}(n)$ instead of $\mathbf{u}(n\delta)$ and indicates that the input sequence is treated now as a discrete time sequence. Also we substitute δ/c with Δt :

$$\begin{aligned}
x(n+1) &= (1 - \alpha\Delta t)x(n) \\
&+ \Delta t f(W^{in}u(n+1) + Wx(n) \\
&+ W^{back}y(n))
\end{aligned} \tag{5.25}$$

$$y(n) = g(W^{out}[x(n); u(n)]) \tag{5.26}$$

where Δt is a compound time gap between two consecutive time steps divided by the time constant of the system and α is the decay (or leakage) rate (Lukosevicius, 2012)

if we set $\alpha = 1$ and redefine Δt in the above equation as the leaking rate α to control the “speed” of dynamics:

$$\begin{aligned}
x(n+1) &= (1 - \alpha)x(n) \\
&+ \alpha f(W^{in}u(n+1) + Wx(n)
\end{aligned} \tag{5.27}$$

Essentially, described equation is an exponential moving average (smoother). This equation has only one additional parameter and the desirable property that neuron activations $x(n)$ never go outside the boundaries denoted by $f(\cdot)$. The basic ESN is a special case of leaky integrator neuron. Small values of α and Δt result in reservoirs that react slowly to the input. A natural constraint of leaking rate is that it never exceeds 1 and specifically $\alpha \in [0 \ 1]$.

From a signal processing point of view, the exponential moving average on the neuron activation eq. (5.25) does a simple low-pass filtering of its activations with the cutoff frequency:

$$f_c = \frac{r}{2\pi(1-r)\Delta t} \quad (5.28)$$

Where Δt is the Euler discretization time step. This makes the neurons average out the frequencies above f_c and enables tuning the reservoirs for particular frequencies.

5.9.2.2 TIME WARPING INVARIANT ECHO STATE NETWORK (TWI ESN)

Dealing with artificial data which are transformed to time series or data who came from human activities a common problem is time warping. Which is input signal including different time scales of process along the time domain or, in other words, sorts of variations in the speed of a process. For discrete time input signals taken by sampling from a continuous time series it can be assumed as variations of sampling rate.

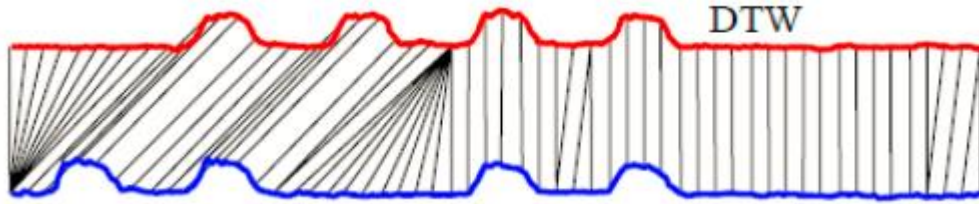


Figure 20. Time warped data.

By definition two signals $s_1(t)$ and $s_2(t)$ are connected by an approximate continuous time warping (τ_1, τ_2) , if τ_1, τ_2 are strictly increasing functions on $[0, T]$, and $s_1(\tau_1(t)) \cong s_2(\tau_2(t))$ for $0 \leq t \leq T$. We can choose one signal, say $s_1(t)$, as a reference and all signals that are connected with it by some time warping (e.g. $s_2(t)$) call (time-) warped versions of $s_1(t)$.

In (Sun, et al., 1993) a time warping invariant neural network was proposed. In this approach, time warping invariance is obtained by normalizing time dependencies of the state variables with respect to the length of trajectory of the input signal in its phase space. In other words, the input signal is considered in a “pseudo-time” domain, where “time span” between two subsequent pseudo time steps is proportional to the metric distance in the input signal between these time steps. As a consequence, input signals will be changing with a constant metric rate in this “pseudo-time” domain.

In discrete time, for a K dimensional input signal $\mathbf{u}(n)$ we make the $\Delta t'$ time-varying by:

$$\Delta t'(n + 1) = b \parallel \mathbf{u}(n + 1) - \mathbf{u}(n) \parallel \quad (5.29)$$

Where $\Delta t'(n + 1)$ is a “pseudo time” gap between time steps (n) and $(n + 1)$, and b is a constant factor. Note that the K -dimensional array of input signals varies with a constant metric value equal to $1/b$ in this “pseudo time” domain.

Substituting Δt in eq. (5.25) with $\Delta t'(n + 1)$ we obtain the update state equation of Time Warp Invariant Echo State Network (TWIESN):

$$\begin{aligned} \mathbf{x}(n + 1) = & \mathbf{x}(n) - b \parallel \mathbf{u}(n + 1) - \mathbf{u}(n) \\ & \parallel (a\mathbf{x}(n) + f[\mathbf{W}^{in}\mathbf{u}(n + 1) \\ & + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)]) \end{aligned} \quad (5.30)$$

5.10 FORMULATION OF TRAINING PROCEDURE

We present a general formulation of the training procedure with output feedback.

Signal processing speaking, are basically a ‘nonlinear moving average’ kind of models. The output of many dynamical systems depends on the input and the output history. Feedback output signals are a kind of input. When the teacher output is written into the output units during the learning task, the learning task becomes teacher forcing.

Task. given :a teacher input output time series $(\mathbf{u}(n), \mathbf{y}^{target}(n))$, $n = 0, 1 \dots T$, where inputs are from a compact set U_{in} and the desired outputs $\mathbf{y}^{teach}(n)$ from a compact set U_{out} .

Wanted: An ESN whose output $\mathbf{y}(n)$ approximates $\mathbf{y}^{target}(n)$

Choose input connection weights. Attach input units to the network. If the original network satisfies the Lipschitz condition (Jaeger, 2001), input

connections \mathbf{W}^{in} can be freely chosen without harming the echo state property. Moreover, the experience accumulated so far indicates that the echo state property remains intact with arbitrarily chosen input connection weights, even if only the weaker condition $|\lambda_{max}| < 1$ was ascertained in the previous step.

Procure an echo-state network. Build a network that has echo state property in a state set A with respect to input sequence $\mathbf{u}(n)$ and a ‘pseudo’-input $\mathbf{y}^{teach}(n)$ (i) re-interpret the $N \times L$ matrix \mathbf{W}^{back} as another input weight matrix and join it with $N \times K$ matrix \mathbf{W}^{in} into a $N \times (K + L)$ -matrix $\mathbf{W}^{in\&back}$, (ii) join the input $\mathbf{u}(n + 1)$ with the output $\mathbf{y}^{teach}(n)$ into a compound pseudo-input $\tilde{\mathbf{u}}(n + 1) = (\mathbf{u}(n + 1), \mathbf{y}^{teach}(n))$, and (iii) make sure that the resulting network has the echo state property in an admissible state set A with respect to input $\tilde{\mathbf{u}}(n + 1)$ from the compact set $\tilde{U} = \times U_{out}$.

Any standard sigmoid neurons whose weight matrix \mathbf{W} has spectral radius less than one satisfies the echo state property. Also this property is independent of the input weight matrix and the output weight matrix and both of them can be freely chosen (Jaeger, 2001).

Run ESN with teacher input and with teacher output forcing, dismiss initial transient. Start with an arbitrary network state $\mathbf{x}(0)$ and update the network with the training input and teacher-forced output for $n = 0, \dots, T$. Discard initial steps due to contamination of initial transient.

Compute output weights which minimize the training error.

5.11 PRODUCING A RESERVOIR

In order to produce a ‘rich’ reservoir it is important to understand what function it is serving.

5.11.1 Function of the Reservoir.

The reservoir acts as (i) a nonlinear expansion and (ii) memory of the input $\mathbf{u}(n)$ at the same time (Lukosevicius, 2012).

Reservoir is acting as a nonlinear high-dimensional expansion of the input sequence $\mathbf{u}(n)$

At the same time it should store feature information, thus providing a temporal context of the input $\mathbf{x}(n)$. This characteristic of the reservoir is crucial for temporal learning tasks.

Combining the two aspects the reservoir must provide a rich space of $\mathbf{x}(n)$ states of the inputs, such that linear combination of them should produce the desired signal $\mathbf{y}^{teach}(n)$.

5.11.2 GLOBAL PARAMETERS OF THE RESERVOIR

According to *ML* literature what we call ‘parameters’ could be called “meta-parameters” or “hyper-parameters” as well, as they are not concrete connection weights but parameters governing their distribution. We call them “global parameter” reflecting their origin.

The global parameters of the reservoir are: the size of the reservoir N , sparsity of non-null elements and their distribution, spectral radius of \mathbf{W} , single or multiple scaling and shifting of \mathbf{W}^{in} and the leaking rate α . Detailed information on the design choices are described below.

5.11.2.1 SIZE OF THE RESERVOIR

A crucial parameter of the ESN model is the number of internal units N of the reservoir. The memory capacity of the network increases as the reservoir increases, also affecting network performance. Hence, training and running an ESN is computationally cheap compared to other RNN approaches; reservoir sizes of order 10^4 are not uncommon [Fabian Triefenbach, 2011]. It is easiest to find a linear combination of the signals to approximate $\mathbf{y}^{teach}(n)$ if the internal units are as many as possible. If the task is trivial or there is not enough data $T < 1 + K + N$ the size of reservoir is smaller.

Taking into account the number of independent real values the reservoir must remember from the input to accomplish the learning task successfully we can set a lower bound of the reservoir N . Memory capacity (maximum number of stored values) of an ESN is equal or less to the size of the reservoir N .

5.11.2.2 SPARCITY OF RESERVOIR

In (Jaeger, 2001) it is recommended to make a sparse connection reservoir, of which most of the entries must be null. Network obtains a slightly better performance with sparse connectivity. Typically, we connect each internal unit to a small fixed number of other internal units on average. And taking advantage of this internal unit sparsity to speed up computation time. This parameter is not crucial to the network's behavior.

5.11.2.3 DISTRIBUTION OF NONZERO ELEMENTS

Nonzero entries of the internal weight matrix can be chosen to be either symmetrically uniformed, discrete bi-valued or normally distributed center around zero. Gaussian distributions also exist. All the described distributions, except bi-valued, have the same performance. The discrete bi-valued provides less rich signal space. The range of distribution is not so important because spectral range rescales the magnitude of the nonzero elements of the matrix.

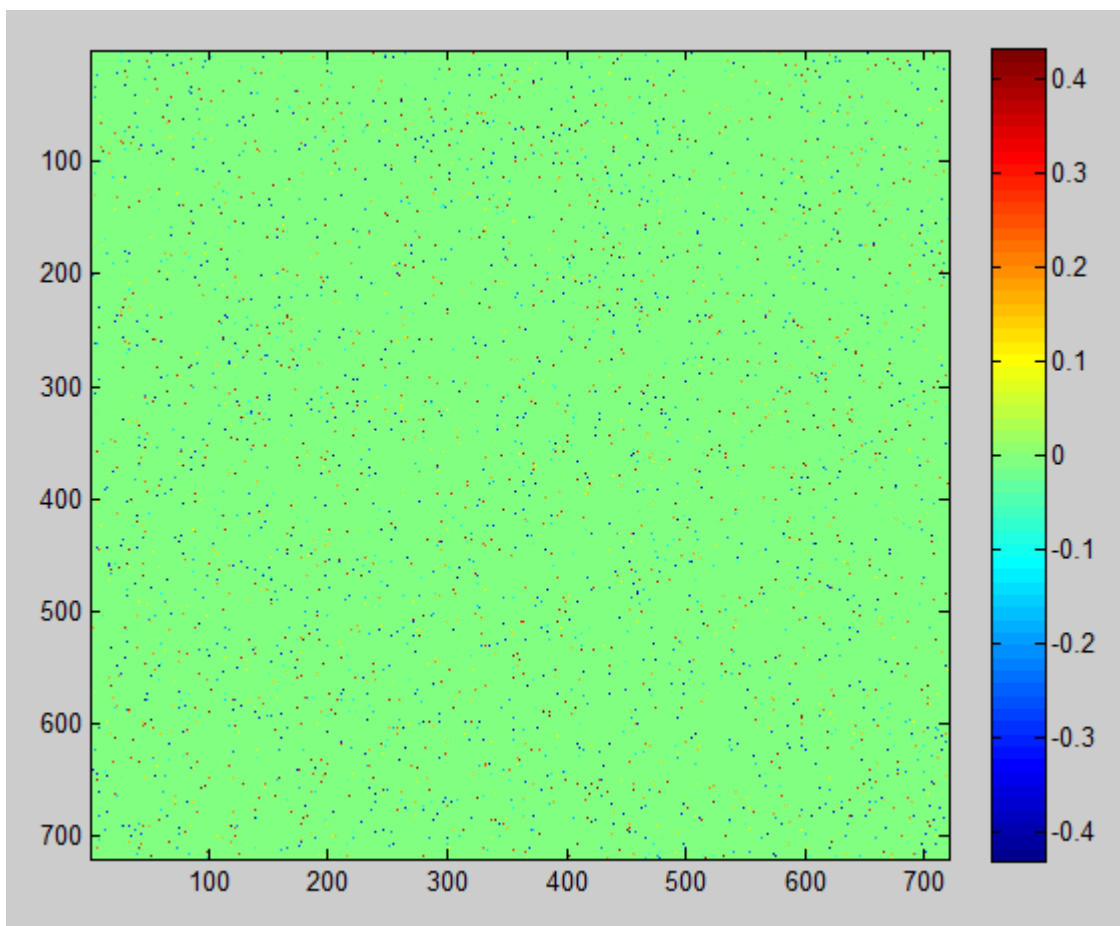


Figure 21. Random scaled internal weights matrix, spectral radius 0.95.

5.11.2.4 SPECTRAL RADIUS

One of the most significant global parameters of the network is the spectral radius. Spectral radius is the maximal absolute eigenvalue $|\lambda_{max}|$ of internal units weight matrix \mathbf{W} . In general, the most popular method to build a matrix \mathbf{W} which maximal absolute value is unit described as: first a random sparsely interconnected matrix \mathbf{W}_0 is generated then the spectral radius $\rho(\mathbf{W})$ is computed. The matrix \mathbf{W} is divided by $\rho(\mathbf{W})$ to get a unit spectral radius matrix this is finally scaled with the desired spectral radius.

An ESN reservoir must satisfy the echo state property: the internal state $\mathbf{x}(n)$ is uniquely defined by the decaying history of the input signals and/or the previous output signal. In other words for a long enough input $\mathbf{u}(n)$, the reservoir state $\mathbf{x}(n)$ should not depend on the initial conditions that existed before the input (Lukosevicius, 2012)

The echo state property, theoretically, can be violated even if the spectral radius is less than unit (Jaeger, 2001) or can be held for values bigger than one for nonzero inputs. This fact can be explained by the strong influence of input signals which are pushing activations of the internal units away from zero, where the $\tanh()$ nonlinearities have a unitary slope to regions where this slope is smaller, thus reducing the gains of the neurons and the effective strength of feedback connections. Intuitively speaking, due to activation-squashing nonlinearities, strong inputs “squeeze out” the autonomous activity from the reservoir activations.

Choosing the appropriate magnitude of spectral radius is not easy and depends of the task. Spectral radius must be greater for tasks that require longer memory of the inputs and smaller for tasks that the current output depends more on the recent history of input signal.

5.11.2.5 INPUT SCALING

The input scaling weight matrix consists another key parameter for the optimization of ESN. Input weight matrix is usually dense and for scaling uniformly distributions or normal distributions are used.

A general practice when a network has many inputs is to scale together using one scale value. This reduces the amount of adjustable parameters. If the network includes bias input (usually first column) it is recommended to scale

separately from the other inputs. If the remaining inputs contribute to the task in very different ways, it is suggested to scale each independently. In (Jaeger, 2001), recommends to scale and shift the input signal. We can achieve the same result by scaling the input weights of the bias input and the rest of the inputs separately.

Many times, input signals come from experimental results or biological observations. This means that data have different range values, noise. In such cases input data must be normalized. This puts each learning task to a bounded region. For example if the input signal distribution is unbounded we should apply $\tanh()$, otherwise outliers can “send” the internal units into strange regions which are not familiar to regions of which global parameters of the network have been adjusted or the outputs learned. This can cause virtual loss of useful memory (may lead to saturations in the activation nonlinearities) or unpredictable output of these points.

Internal activation units are typically $\tanh()$. Input scaling determines how nonlinear internal units responses are. For a linear learning task the scaling of matrix \mathbf{W}^{in} must be small because operating internal units close to zero results in almost linear behavior of the unit. If the scaling is larger the unit’s activation is saturated to values -1 and 1 acting in a more nonlinear, bi-valued switching type. How nonlinear is the task, is not easy to answer.

Scaling of \mathbf{W}^{in} combined with \mathbf{W} denotes the percentage of contribution of the current input $\mathbf{u}(n)$ to the current state $\mathbf{x}(n)$ and the left percentage is on the previous state $\mathbf{x}(n - 1)$. This conclusion must also take into account the size of K and N .

In (Hermans & Schrauwen, 2010) denoted that the representation of different Principle Components of $\mathbf{u}(x)$ in $\mathbf{x}(n)$ is roughly proportional to the square root of their magnitudes in $\mathbf{u}(n)$. *In other words, the reservoir tends to flatten the spectrum of principal components of $\mathbf{u}(n)$ in $\mathbf{x}(n)$, something to keep in mind when choosing the right representation or preprocessing of the data. For example, if smaller principal components carry no useful information it might be helpful to remove them from the data by Principal Component Analysis (PCA) before feeding them to a reservoir, otherwise they will get relatively amplified there* (Lukosevicius, 2012).

CHAPTER 6. PRACTICAL APPROACH OF ESN

In machine learning the most important parameters must be tuned by trial and error method.

6.1 MAIN PARAMETERS OF THE NETWORK

Main parameters to optimize ESN reservoir are: input scaling, spectral radius and leaking rates. The appropriate magnitude in order for the network to reach good performance depends on the task and needs multiple trials.

Some times the performance can be improved if we set different scalings of the columns of input weight matrix W^{in} or separate the bias input scaling from the scaling of the other “active” inputs. This separation takes advantage of the different nature of inputs (if they exist). Instead of using a global leaking rate α , if the task requires modeling of the times series producing dynamical system on multiple time scales, it might be useful to set different leaking rates to different units (making a vector $a \in \mathbb{R}^K$).

The size of reservoir is limited due to finite memory of computational systems.

Lower importance parameters to be tuned is the reservoir sparseness and weight distribution.

6.2 PARAMETER SELECTION SETUP

In ESN training algorithms only the network-to-output connection weights have to be trained. The main advantage of ESN is that learning of outputs is fast. This feature is exploited in evaluating the performance of reservoir by a particular set of parameters.

A method to evaluate the reservoir performance is to train the output and measure the error. Training error is usually used (or validation error can be used).

Due to randomly generated reservoirs, same parameters can cause slightly different network performances. This observation affects more small sized

reservoir, than bigger ones. Random variations inside a big reservoir tend to “average out”.

6.3 MANUAL PARAMETER SELECTION

Machine learning requires some parameters to be manually optimized. In some machine learning approaches some parameters are selected through automated ways. Even in this case it is necessary some of the parameters to be tuned manually. These parameters are typically called meta-parameters.

A typical way of handling these parameters is to change one parameter at a time and record the performance of the network. Repeat this procedure until you get a good performance then change another. Changing more than one parameters at the same time usually has catastrophic effects to network performance. Also it is blurry to tell which parameter contributed what.

6.4 INITIAL TRANSIENT

Usually we set an arbitrary state at time step zero of the internal units of $\mathbf{x}(0) = 0$. This denotes an unknown starting state. The data of $\mathbf{x}(n)$ which are from the beginning of the training run are discarded, so they are not used for learning output weight matrix \mathbf{W}^{out} . The number of time steps to discard depends on the memory of the network. Discarded steps are usually tens or hundreds.

6.5 OUTPUT FEEDBACK AND STABILITY PROBLEMS

6.5.1 OUTPUT FEEDBACK

For some usually complex tasks (e.g. pattern generation, classification), trained readouts are fed back to the reservoir and training process changes its dynamics. This process creates a recurrence between the trained outputs and the reservoir. There are two implementations to make this real. Either by back project (feedback) connections \mathbf{W}^{back} from the output to the reservoir or by looping output $\mathbf{y}(n - 1)$ as an input $\mathbf{u}(n)$ for the next update step n . The second implementation is used to pattern generator. These two options are equivalent, only the notation changes: \mathbf{W}^{in} is equivalent to \mathbf{W}^{back} and $\mathbf{u}(n)$ to $\mathbf{y}(n - 1)$, respectively. There are tasks for which both are used.

Feedback connections make ESN more powerful for hard learning tasks because it is no longer an input driven dynamical system, but the dynamics are adapted to the task. ESN with feedback connections suffers from instability problems.

6.5.2 TEACHER FORCING

As described in the previous section, training ESN with feedback connections changes the dynamics of all the internal units and the outputs, too. In traditional training we feed trained outputs into the reservoir. To break the recurrence relationship between the reservoir and the output readout feedback the desired output $\mathbf{y}^{target}(n-1)$ through the feedback connections \mathbf{W}^{back} instead of real output $\mathbf{y}(n-1)$ while learning. This method is called teacher forcing.

$$\begin{aligned} \mathbf{x}(n+1) \\ = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n)) \end{aligned} \tag{6.1}$$

Teacher forcing means that target values are fed back to the reservoir, as if they were already successfully learned. This enables us to learn outputs in one iteration and is a valid assumption, if in the end the outputs are learned well (i.e. the feedbacks are similar to the one which we assumed while training). If the feedback is not learned well, this assumption is not valid and the distorted feedback may further distort the outputs. (Lukosevicius, 2012)

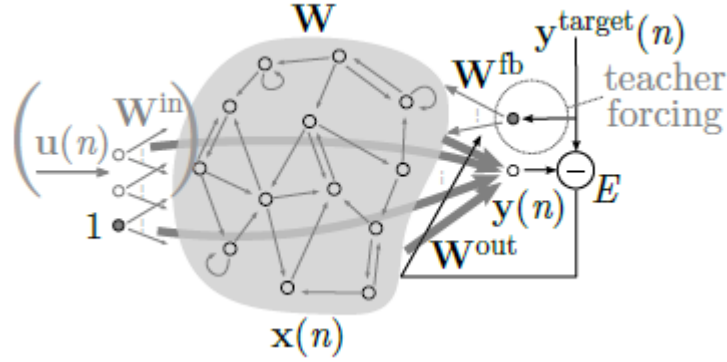


Figure 22. An ESN with output feedbacks trained with teacher forcing.

In other words, this method has very good results if the output can be learned precisely. If this is not feasible, the distorted feedback leads to an even more distorted output of the next time step feedback and so on. This mechanism leads very quickly to a generated output that diverges from the desired output $y^{target}(n)$.

Even with well-learned outputs the dynamical stability of the autonomous running system is often an issue (Lukosevicius, 2012)

6.5.3 FEEDBACK STABILITY PROBLEMS

As described in previous section adding noise to the reservoir states has a similar effect as ridge regression to network stability. *Setting the right amount of noise is a delicate balance between the sharpness of the prediction and stability* (Lukosevicius, 2012).

Another strategy is to add scale noise to teacher forced signal $y^{target}(n)$. This makes the reservoir learn an imperfectly $y^{target}(n)$ and the readout is trained to ignore some inputs and feedback signals.

A recently proposed method is to regularize recurrent connections W . Matrix W is relearned with regularization using ridge regression as proposed for W^{out} . This method reduces recurrent connection strength and makes ESN more stable.

CHAPTER 7. SUPERVISED TRAINING FOR SEQUENCE LABELING

7.1 SEQUENCE LABELING OVERVIEW

As described above, in supervised learning tasks a set of input-target pairs is provided for training. The nature and degree of supervision provided by the targets varies greatly between supervised learning tasks. For example, training a supervised learner to correctly recognize (label) every pixel corresponding to an airplane in an image requires a much more informative target than simply training it to recognize whether or not an airplane exists. To distinguish these extremes, people sometimes refer to weakly and strongly labelled data (Graves, 2008)

Sequence labeling goal is to assign sequences of label, picked from a particular alphabet, to sequences of input data. Well-known examples of sequence labeling are sequences of acoustic data which include spoken words (speech recognition) and video frames time sequences with hand gestures.

For some learning tasks, precise assignment of the labels respectively to the input sequences is determined by the learning algorithm. However, in most of the tasks the alignment is determined with manual or automatic data processing or we only care for the final sequence of label, not the specific time that the label takes place.

Assume that desired sequences have the same length or less than the input sequences. Let S be a dataset of training sets picked independently from a fixed distribution $\mathcal{D}_{\mathbf{u} \times \mathbf{y}^{target}}$. Input space $\mathbf{u} = (\mathbb{R}^K)^*$ is a set of all sequences of K real valued vectors $\mathbf{u}(n) = (u_1(n), \dots, u_K(n))$. Target space $\mathbf{y}^{target} = \Lambda^*$ is a set of all sequences over the finite alphabet Λ of labels. Each element of Λ^* represent a label. On the other hand each element of the dataset S represents a pair of sequences $(\mathbf{u}(n), \mathbf{y}^{target}(n))$. We use S to train a sequence labeling algorithm $h: \mathbf{u} \rightarrow \mathbf{y}^{target}$ to label the sequences in an unknown test (disjoint from S) set S' , as accurately as possible.

7.2 SEGMENT CLASSIFICATION.

When the target sequences consist multiple labels and their assigned location to input sequence is known in advance it is described by the term segment classification. In other words, when the task is to classify separate short time series (which consist input sequences), output of the network $\mathbf{y}(n)$ has one vector (dimension) for each label (class) and the magnitude of the target sequence $\mathbf{y}^{\text{target}}(n)$ is one in the dimension which represents the right class (label) and zero to the other classes. Task examples of segments classification is natural language, human gestures processing and bioinformatics. A disadvantage of sequence classification is that it requires hand segmented data, which is a time-costly procedure.

In sequence labeling input sequences are built from segmented data. Context information of each of this segmented data is the key for a good performance of the training network. Each segmentation of the input sequence is strongly correlated to the corresponding label.

For a pattern classifier $h: \mathbf{u} \rightarrow \mathbf{y}^{\text{target}}$ we can measure misclassifications on the test set S' by classification error rate:

$$\begin{aligned} E^{\text{class}}(h, S') \\ = \frac{1}{|S'|} \sum_{(u, \mathbf{y}^{\text{target}}) \in S'} \begin{cases} 0 & \text{if } h(u) = \mathbf{y}^{\text{target}} \\ 1 & \text{eitherwise} \end{cases} \end{aligned} \tag{7.1}$$

CHAPTER 8. TEMPORAL GESTURE RECOGNITION EXPERIMENTS

8.1 AIM OF THE EXPERIMENT

The focus of the numerical experiments is on “multiple instances, user independent learning” of skeletal data, that is, learning to recognize gestures from several instances for each category performed by different users, picked from a vocabulary of 20 gesture categories (see Figure 23). This vocabulary contains a set of unique gestures, generally related to a particular task. In other words, we try to temporally classify these 20 unique gestures in a time series (sequence) by computing the classification error rate.

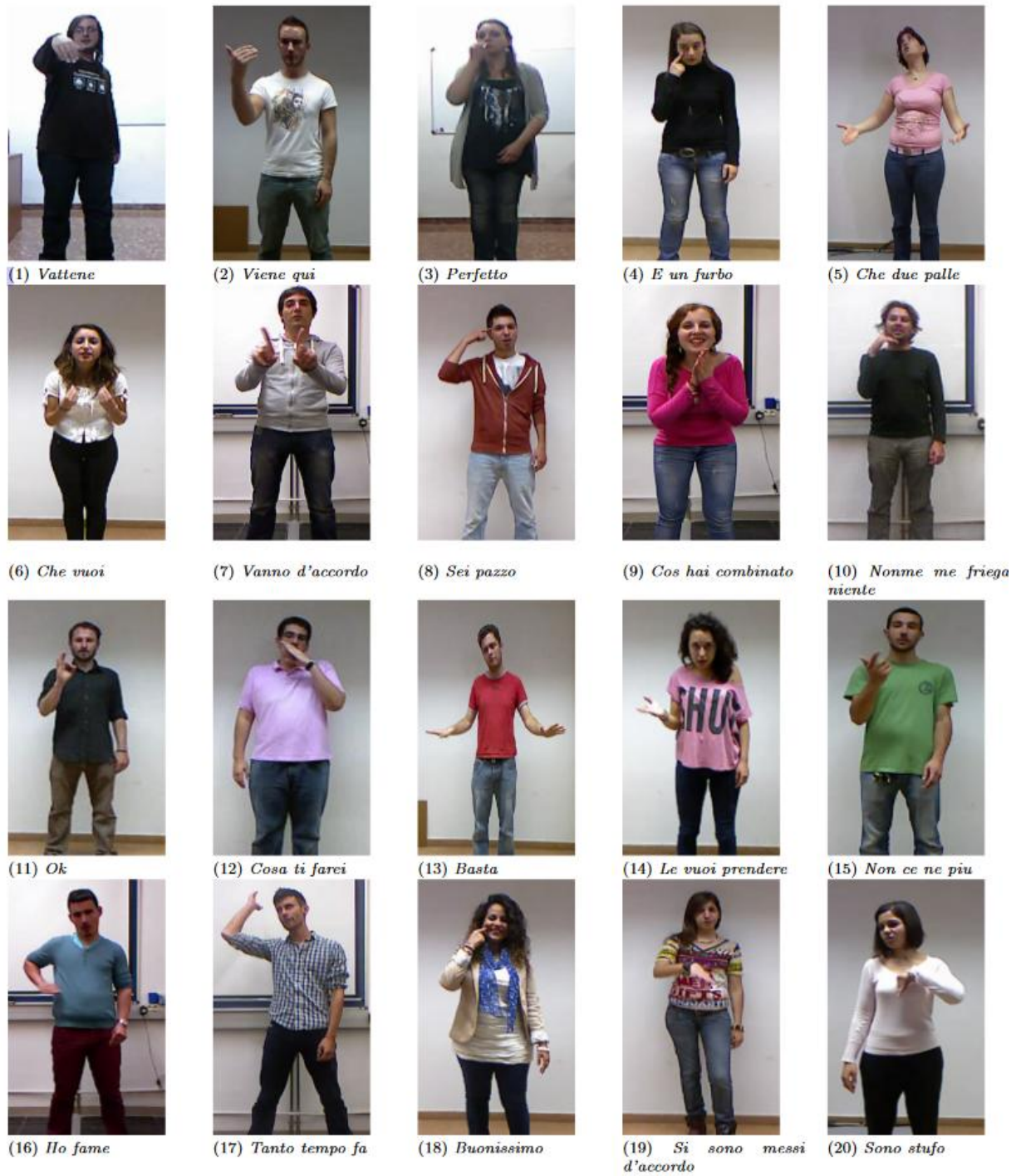


Figure 23. Data set gesture categories.

We used ESN software developed by Jaeger's research team to recognize these gestures. Software executed in Matlab environment.

8.2 DATA

8.2.1 GENERAL DESCRIPTION

We used a large video database (available for competition" **Multi-modal Gesture Recognition Challenge 2013**") from a lexicon of 20 Italian gesture categories recorded with a KinectTM camera. Downloaded from <http://sunai.uoc.edu/chalearn/>. Multi-modal Gesture Recognition Challenge provided 3 datasets: Development, Validation and Final Evaluation, for algorithm development and evaluation. Each dataset consists of hundreds of zip files, and each file contains approximately one-minute-long multi-modal gesture data, including audio, video and skeleton information.

The database contains:

- Training data (RGB+Depth+Audio) and labels for 393 sessions, which correspond to 7.754 Italian gestures.
- Validation data (RGB+Depth+Audio) has the same format as training data, but labels are not provided. There are 287 sessions, which correspond to 3.362 Italian gestures.
- Test data have exactly the same structure as the validation set. It contains 276 files containing a total of 2742 Italian gestures.

8.2.2 MAIN CHARACTERISTICS OF THE DATASET

- The camera is in fixed position
- There are no resting positions and each sequence records one person's gestures (Figure 24).
- A single user is recorded in front of a KinectTM, performing natural communicative gestures and speaking in fluent Italian.



Figure 24. The 20 gestures performed by one person, which consist one sequence.

- 13,858 gesture samples recorded with the Kinect™ camera, including audio, skeletal model, user mask, RGB, and depth images.

- RGB video stream, 8-bit VGA resolution (640X480) with a Bayer color filter, and depth sensing video stream in VGA resolution (640X480) with 11-bit. Both are acquired in 20 Frames per second on average.
- A total number of 27 users appear in the data set.
- The data set contains the following number of sequences: 393 (7,754 gestures), each sequence lasts between 1 and 2 minutes and contains 20 gesture samples, around 1,800 frames. The total number of frames of the data set is 1,720,800.
- All the gesture samples belonging to the 20 main gesture categories from an Italian gesture dictionary are annotated at frame level indicating the gesture label.

There are several aspects that arise from multimodal gesture recognition in this database described by” **Multi-modal Gesture Recognition Challenge 2013**”, such as gesture continuous recording (no resting points between gestures), the presence of distracter gestures, the relatively large number of categories, the length of the gesture sequences varies, and different people performing the same gestures. Furthermore, there is not a specific way to perform the included cultural gestures, e.g., “vieni qui” is performed with repeated movements of the hand towards the user, with a variable number of repetitions (Figure 25). Similarly, gestures are performed using one hand, either the left or right hand. Finally, variations in duration of gestures performed, background, lighting and resolution, occluded some parts of the body, different Italian dialects



(a)



(b)



(c)



(d)



(e)

Figure 25. (a, b) Left and right handed instances for gesture “vieni qui”, (c, d, e) left and right handed instances and arm position of “vattene”.

8.2.3 DATA FORMAT

Each sample sequence (X) contains individual files named: X_audio.wav, X_color.mp4, X_depth.mp4, X_user.mp4 and X_data.mat containing the audio, RGB, depth, user mask and data about videos for a given sequence X. All the sequences are recorded at 20 FPS. Analytically:

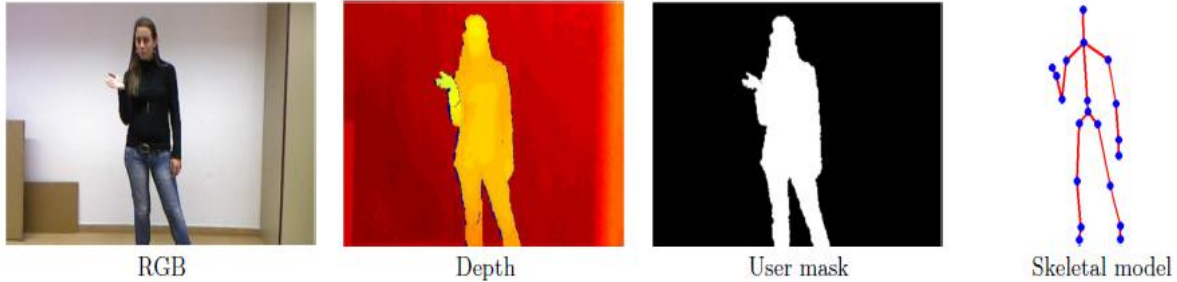


Figure 26. Different data modalities of the provided data set. From left to right are the image selected from the RGB video, depth video, user-index video, and skeletal model respectively.

RGB: This matrix represents the RGB color image, expressed in 8-bit VGA resolution (640x480) with a Bayer color filter.

Depth: The Depth matrix contains the pixel-wise z component, VGA resolution (640x480) represented with 11bits. The value of depth is expressed in millimeters.

UserIndex: The user index matrix represents the player index of each depth pixel. A non-zero pixel value means that a tracked subject occupies the pixel, and a value of 0 denotes that no tracked subject occupies the pixel.

DATA: contains the following structure:

- **Video:** structure that contains above 5 fields (structures and files):
 - **NumFrames:** Total number of frames.
 - **FrameRate:** Frame rate of the video in fps.
 - **Frames:** A column vector (1xnumber_of_frames) structure is contained within a column vector named skeleton.
 - **Skeleton:** Each row of Skeleton is a structure. It contains the joint positions, and bone orientations comprising a skeleton of each frame. The format of a Skeleton structure is:

1. **WorldPosition:** The world coordinates position structure represent the global position of a tracked joint. The format is X, Y, which represents the x, y, and z components of the subject's global position (in millimeters).
2. **PixelPosition:** The pixel coordinates position structure represents the position of a tracked joint. The format of the Position structure is X, Y, Z which represent the x and y components of the joint location over the RGB map (in pixels coordinates).
3. **WorldRotation:** The world rotation structure contains the orientations of skeletal bones in terms of absolute transformations and is formed by a 20x4 matrix, where each row contains the W, X, Y, Z values of the quaternion related to the rotation. The world rotation structure provides the orientation of a bone in the 3D camera space. The orientation of a bone is relative to the child joint and the Hip Center joint still contains the of the player/subject.

Labels: Structure that contains the data about labels contained in the sequence, sorted in order of appearance. The labels considered to the 20 gesture categories as shown in the

- Table 3. The format of a Label structure (Figure 27) is:
 - **Name:** name of the gesture.
 - **Begin:** gesture starting frame.
 - **End:** gesture ending frame.

abc	Name	Begin	End
	'freganiente'	1	99
	'furbo'	100	139
	'daccordo'	140	195
	'basta'	196	274
	'messidacc...	275	339
	'noncenepiu'	341	392
	'combinato'	400	459
	'ok'	460	532
	'tantotempo'	533	594
	'vattene'	595	639
	'seipazzo'	640	679
	'cheduepalle'	680	739
	'buonissimo'	740	799
	'perfetto'	800	879
	'cosatifarei'	880	959
	'fame'	960	1039
	'prendere'	1040	1069
	'vieniqui'	1077	1199
	'sonostufo'	1200	1279
	'chevuoi'	1280	1334

Figure 27. Fields of structure Labels for a sample sequence.

-JointType: **Skeleton joints that make up a tracked skeleton.** The

Figure 28 visualizes these joint types.

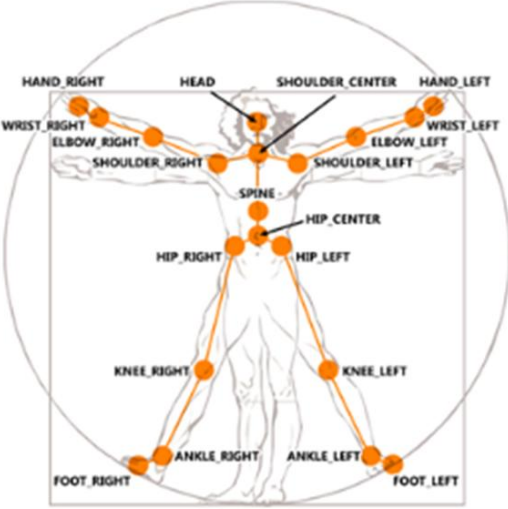
N	Joint type		N	Joint type
1	HipCenter		11	WristRight
2	Spine		12	HandRight
3	ShoulderCenter		13	HipLeft
4	Head		14	KneeLeft
5	ShoulderLeft		15	AnkleLeft
6	ElbowLeft		16	FootLeft
7	WristLeft		17	HipRight
8	HandLeft		18	FootRight
9	ShoulderRight		19	KneeRight
10	ElbowRight		20	FootRight

Figure 28. Left and right edge: tracked joint types, middle: skeleton joint positions.

8.3 PREPROCESSING

8.3.1 DATA TRANSFORMATION

Across all provided video features, skeleton features are a meaningful representation of body posture in each video frame. Thus, we choose skeletal data to consist ESN classifier inputs.

As we describe above, each sequence contains a file named WorldPosition for each frame. This file contains world coordinates position of all 20 tracked joints. We use this data to export three dimensional Euclidean distance between two tracked joints. Three dimensional Euclidean distance computed according to mathematical type:

$$D(\mathbf{p}, \mathbf{q}) \quad (8.1)$$

Where $\mathbf{p} = (x_1, y_1, z_1)$ and $\mathbf{q} = (x_2, y_2, z_2)$ are two points in Euclidean 3D-space

Vectors \mathbf{p} and \mathbf{q} represent two tracked joints.

The 3D distances of the following joint pairs were computed:

Id-row	Joint pairs	3D distance	Id-row	Joint pairs	3D distance
1	'HipCenter' - 'Head'	d	7	'Head' - 'HandRight'	d
2	'HipCenter' - 'ElbowLeft'	d	8	'Head' - 'ElbowLeft'	d
3	'HipCenter' - 'HandLeft'	d	9	'Head' - 'HandLeft'	d
4	'HipCenter' - 'ElbowRight'	d	10	'ShoulderCenter' - 'HandLeft'	d
5	'HipCenter' - 'HandRight'	d	11	'ShoulderCenter' - 'HandRight'	d
6	'Head' - 'ElbowRight'	d	12	'HandLeft' - 'HandRight'	d

Table 1. Calculated distances joint pairs.

Then 3D distances are normalized based on the sum of skeleton based 3D distances of Head-HipCenter joint pair and HandLeft-HandRight joint pair for each element individually.

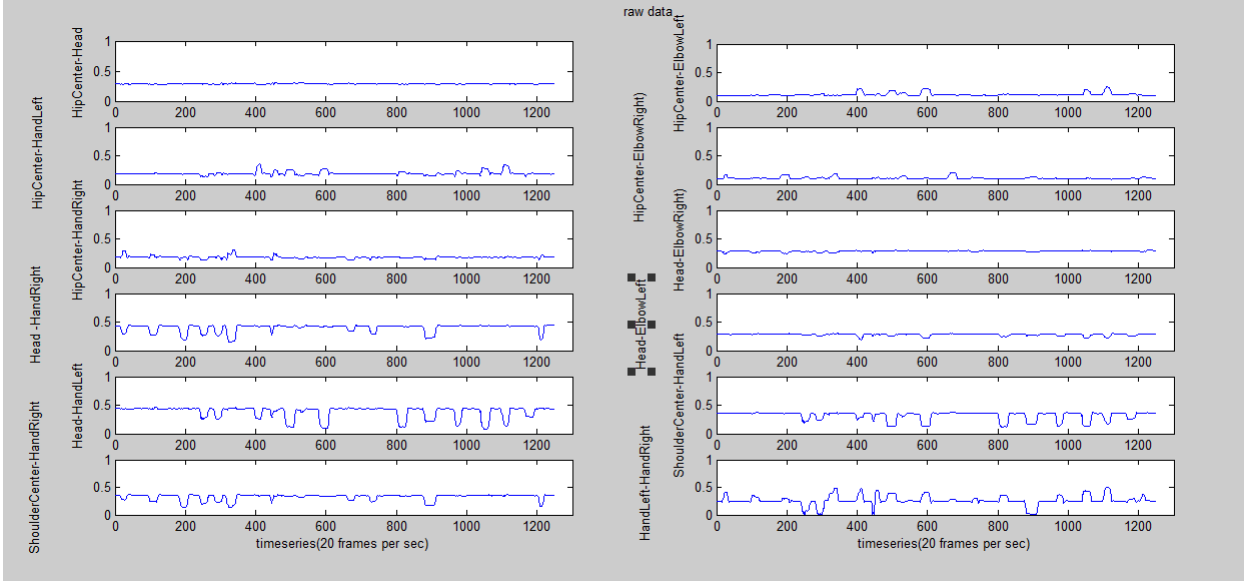


Figure 29. Twelve joint pairs data after normalization over sampling period.

8.3.2 DATA RESCALING AND INPUT FORMAT

Observing raw data (Figure 29), it is obvious that each channel has different value ranges and a considerable offset. Feeding such a signal in its raw version would amount to adding a strong bias constant to the channels with greater mean values, which would effectively shift the sigmoids f of reservoir units away from their centered position toward their saturation range. This leads to systematic wrong behavior of internal units because each input channel contributes differently to the learning task.

We rescale raw data to a desirable range. Considering that output has a switching behavior (0 for non-label existence and 1 for existence) and the fact that in machine learning it is advisable teacher data (input and output sequences) ranged to the same interval, we choose to investigate two different types of normalization. First data normalization bounded to interval $[0\ 1]$, (Figure 30) and the second type is to apply interval $[-0.8\ 0.8]$ to input data. This puts learning task into a more standardized setting.

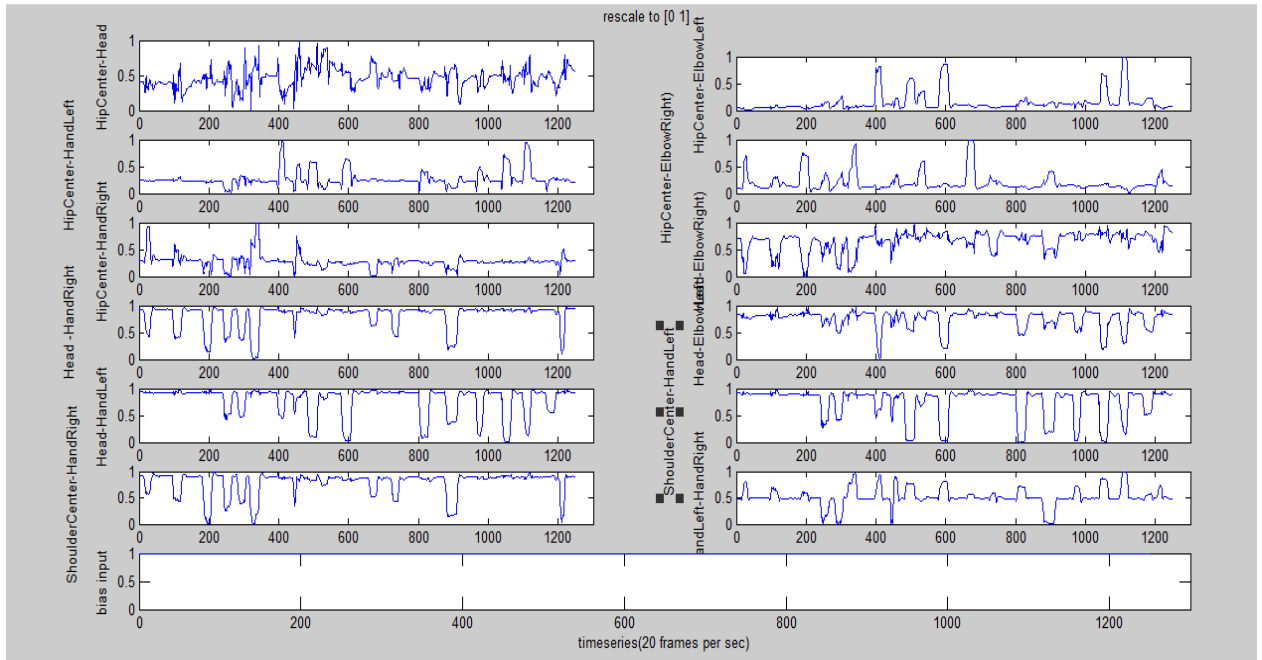


Figure 30. Input data rescaled to interval [0 1]. Last row: bias input.

Input vector (see Figure 30) of ESN network consists of the twelve three dimensional Euclidean distances plus one bias input $\mathbf{u} = u_1 \dots u_{13}$:

Input vector \mathbf{u}			
\mathbf{u}	Joint pairs	\mathbf{u}	Joint pairs
u_1	Bias input	u_8	'Head' - 'HandRight'
u_2	'HipCenter' - 'Head'	u_9	'Head' - 'ElbowLeft'
u_3	'HipCenter' - 'ElbowLeft'	u_{10}	'Head' - 'HandLeft'
u_4	'HipCenter' - 'HandLeft'	u_{11}	'ShoulderCenter' - 'HandLeft'
u_5	'HipCenter' - 'ElbowRight'	u_{12}	'ShoulderCenter' - 'HandRight'
u_6	'HipCenter' - 'HandRight'	u_{13}	'HandLeft' - 'HandRight'
u_7	'Head' - ElbowRight'		

Table 2. Network input vector.

8.3.3 OUTPUT FORMAT

We want to derive assertions of gestures from input histories $\mathbf{u}(n-1), \mathbf{u}(n-2), \dots$. By values 0 and 1 we code the existence/non-existence of a gesture and we want to realize, through learning, $F(\mathbf{u}(n), \mathbf{u}(n-1), \mathbf{u}(n-2), \dots)$ as a 0 – 1 value indicator function that changes its value to 1 as soon as the gesture is present. In this study we consider the 20 gestures which are aligned to 20 subsequents of each training data set as unique outputs $\mathbf{y}^{teach} = y_1^{teach} \dots y_{20}^{teach}$.

Output vector \mathbf{y}					
\mathbf{y}	ID	Gesture name	\mathbf{y}	ID	Gesture name
y_1	1	vattene	y_{11}	11	ok
y_2	2	vieniqui	y_{12}	12	cosatifarei
y_3	3	perfetto	y_{13}	13	basta
y_4	4	furbo	y_{14}	14	prendere
y_5	5	cheduepalle	y_{15}	15	noncenepiu
y_6	6	chevuoi	y_{16}	16	fame
y_7	7	daccordo	y_{17}	17	tantotempo
y_8	8	seipazzo	y_{18}	18	buonissimo
y_9	9	combinato	y_{19}	19	messidaccordo
y_{10}	10	freganiente	y_{20}	20	Sonostuf

Table 3. Network output vector.

Each input sequence represents 20 different gesture instances performed by one person. The person cannot perform the same gesture twice within the same sequence (Figure 27). Essentially, each performed gesture represents a temporal pattern which is hand-coded (assigned) to a class (output). This class holds value 1 over steps that gesture is present and zero to other classes for the same time steps (Figure 31). This indicates two facts: first, the durations of the gesture and second, defines start and stop step. Time steps of input and output signal are the same.

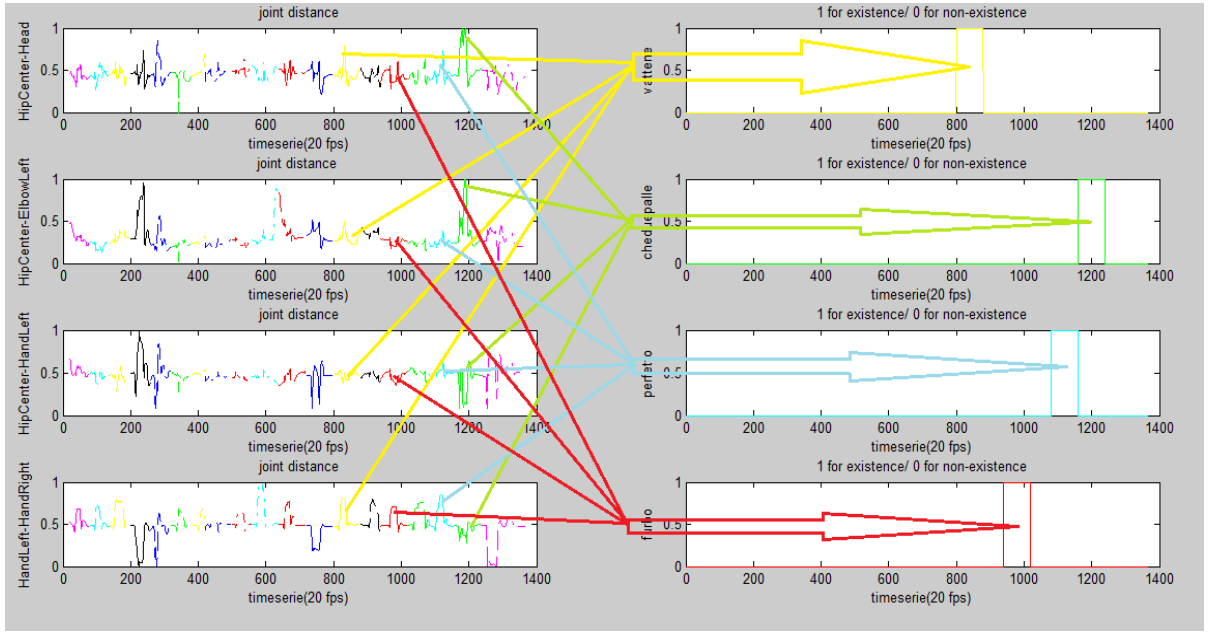


Figure 31. A portion of teacher sequence (4 of 13 inputs) and target output vector (4 of 20) and its manual alignment of them. Each input signal contains 20 regions (represented as different color setup) of equally unique gesture instances performed by one individual.

8.4 EXPERIMENTAL SETUP

The ESN software we used is written in Matlab by Herbert Jaeger and group members. Downloaded from <http://reservoir-computing.org/node/129> .

Teacher signal is composed from the 12 joint pair 3 – D distances which are filtered and normalized plus 1 bias input (input sequence u_1, u_2, \dots, u_{13}) where $u_i \in [0, 1]$. The desired output sequence ($y_1^{target}, y_2^{target}, \dots, y_{20}^{target}$) where $y_i^{target} \in [0, 1]$, that is one output for each label. IF there exists a gesture corresponding to the output, then the value is unit else is zero. We want a trained ESN ($W^{in}, W, W^{back}, W^{out}$) whose output $y(n)$ approximates the teacher output y^{target} , when the ESN is driven by the training input $u(n)$.

A criterion for the actual gesture recognition was the maximum value of each dimension of the output signal ($y(n)$). To avoid outliers we firstly smooth $y(n)$ with a moving average filter. In this setup maximum value of each dimension of output signal corresponds to one recognized gesture instance. A gesture is considered correctly recognized (Figure 35) if the time step $y_i(r)$ matches the

interval of which $y_i^{target}=1$ (only one interval of $y(n)$ can be matched with one interval of y^{target}).

We split the dataset of 393 sequences into two subcategories. The first one, called training data, is used for training the ESN network and the second one for testing the network. Training data is 70% of the dataset and rest of it is for testing.

8.5 TRAINING-TEST ECHO STATE NETWORK: ALGORITHM

The process of ESN software we used is described below and includes the following steps:

STEP 1. PARAMETERS SETUP

We manually select magnitude of the parameters: size dynamic reservoir, spectral radius, input scaling, input shift, output scaling, output shift, feedback scaling, sparsity of nonzero elements of weight matrix W_0 , distribution of input weight matrix W^{in} , distribution of feedback weight matrix W^{back} , noise v . Also select ESN type: *leaky1_ESN*, *leaky_ESN*, *plain_ESN*, *twi_ESN*

Plain _ESN (Plain _ESN.m): generates the internal states of an ESN with standard additive-sigmoid neurons by computing:

$$\begin{aligned} x(n+1) \\ = f(W^{in}u(n+1) + Wx(n)) \end{aligned} \tag{8.2}$$

Or

$$\begin{aligned} x(n+1) \\ = f(W^{in}u(n+1) + Wx(n)) \end{aligned} \tag{8.3}$$

with teacher forcing.

Leaky_ESN (leaky_ESN.m): Updates internal state using the leaky integrator neuron model by computing:

$$\begin{aligned} x(n+1) &= (1 - \alpha \Delta t)x(n) \end{aligned} \quad (8.4)$$

Or

$$\begin{aligned} x(n+1) &= (1 - \alpha \Delta t)x(n) \end{aligned} \quad (8.5)$$

with teacher forcing.

Leaky1_ESN (leaky1_ESN.m): Updates internal state using the leaky integrator neuron model by computing:

$$\begin{aligned} x(n+1) &= (1 - \alpha)x(n) \\ &+ \alpha f(W^{in}u(n+1) + Wx(n)) \end{aligned} \quad (8.6)$$

Or

$$\begin{aligned} x(n+1) &= (1 - \alpha)x(n) \\ &+ \alpha f(W^{in}u(n+1) + Wx(n)) \end{aligned} \quad (8.7)$$

with teacher forcing.

Twf_ESN (twf_ESN.m) updates internal states of an ESN using time warping invariant model:

$$\begin{aligned}
\mathbf{x}(n+1) = & \mathbf{x}(n) - b \parallel \mathbf{u}(n+1) - \mathbf{u}(n) \parallel \\
& (a\mathbf{x}(n) + f[\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \\
& \mathbf{W}^{back}\mathbf{y}^{target}(n)]),
\end{aligned} \tag{8.8}$$

or

$$\begin{aligned}
\mathbf{x}(n+1) = & \mathbf{x}(n) - b \parallel \mathbf{u}(n+1) - \mathbf{u}(n) \parallel \\
& \parallel (a\mathbf{x}(n) + f[\mathbf{W}^{in}\mathbf{u}(n+1) \\
& + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)]))
\end{aligned} \tag{8.9}$$

with teacher forcing.

Some important notes:

- The matrix \mathbf{W}_0 should be sparse, a simple method to encourage a rich variety of dynamics of different internal units. The weights should be roughly zero mean. There are plenty of ways to construct the weight matrix (uniform distribution, Gaussian distribution, or set nonzero weights randomly to -1 or 1).
- The size N of \mathbf{W}_0 should reflect both the length T of training data, and the difficulty of the task. N should not exceed an order of magnitude of $T/10$ to $T/2$ (the more regular-periodic the training data, the closer to $T/2$ can N be chosen). This is a precaution measure against overfitting. Furthermore, more difficult tasks require larger N .
- The setting of ρ is crucial for subsequent model performance. It should be small for fast teacher dynamics and large for slow teacher dynamics, ρ needs to be hand-tuned by trying out several settings.
- This step involves many heuristics and the magnitude of each parameter is crucial for the network behavior.

STEP 2.RUN SYSTEM

2.1 (generate_internal_weights).

Randomly generate an internal weight sparse matrix \mathbf{W}_0 with mean value of weights. Normalize \mathbf{W}_0 to a matrix \mathbf{W}_1 with unit spectral radius by putting $\mathbf{W}_1 = 1/|\lambda_{max}|\mathbf{W}_0$, where $|\lambda_{max}|$ is the spectral radius of \mathbf{W}_0 . Scale \mathbf{W}_1 to $\rho\mathbf{W}_1$, where $\rho < 1$, whereby \mathbf{W} obtains a spectral radius of ρ .

2.2(generate_esn).

- Generate input weights \mathbf{W}^{in} and feedback weights \mathbf{W}^{back} . These weights are chosen from a variety of options such as bi-numeral, Gaussian, normal or bounded distribution. Then, the untrained network ($\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back}$) is (has always been found to be) an echo state network, regardless of how $\mathbf{W}^{in}, \mathbf{W}^{back}$ are chosen.
- Initialize the network state arbitrarily, we assume that at zero state $\mathbf{x}(0) = \mathbf{0}$.
- Drive the network by the training data, for times $n = 0, \dots, T$, by presenting the teacher input $\mathbf{u}(n)$, and by teacher-forcing the teacher output $\mathbf{y}^{target}(n)$, by computing :

$$\begin{aligned} \mathbf{x}(n+1) \\ = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n)) \end{aligned} \tag{8.10}$$

for teacher forcing classic approach of ESN,
or

$$\begin{aligned} \mathbf{x}(n+1) \\ = (1 - \alpha)\mathbf{x}(n) \\ + \alpha f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n)) \end{aligned} \tag{8.11}$$

for teacher forcing leaky integrator neuron model,
or

$$\begin{aligned}
\mathbf{x}(n+1) = & \mathbf{x}(n) - b \parallel \mathbf{u}(n+1) - \mathbf{u}(n) \\
& \parallel (a\mathbf{x}(n) + f[\mathbf{W}^{in}\mathbf{u}(n+1) \\
& + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}^{target}(n)])
\end{aligned} \tag{8.12}$$

for time warping invariant model

- At time $n = 0$, where $\mathbf{y}^{target}(n)$ is not defined, use $\mathbf{y}^{target}(n) = 0$.

2.3(compute_statematrix.m).

- Considering input scaling and shifting recalculate value for each input at time steps $n = 1 \dots T$ according to:

$$u_i(n) = s_{scal(i)}^{Rinput} * u_i(n) + s_{shift(i)}^{Rinput} \tag{8.13}$$

- Considering output scaling and shifting recalculate value for each target output at time steps $n = 1 \dots T$ according to :
-

$$y_i(n) = s_{scal(i)}^{teacher} * y_i(n) + s_{shift(i)}^{teacher} \tag{8.14}$$

- If feedback connections are used, recalculate value for each desired output considering feedback scaling for time steps $n = 1 \dots T$ according to:

$$y_i^{target} = s_{scal(i)}^{back} * y_i^{target} \tag{8.15}$$

- Due to initial transient, for each time larger or equal than an initial washout time $T0$, collect the concatenated input/reservoir/previous-output states ($\mathbf{u}(n) \mathbf{x}(n) \mathbf{y}(n-1)$) as a new row into a state collecting matrix \mathbf{M} . In the end, one has obtained a state collecting matrix of size $(T - T0 + 1) \times (K + N + L)$.

2.4(compute_teacher.m).

Similarly, for each time larger or equal to T_0 , collect the teacher output $\mathbf{y}^{target}(n)$ row-wise into a teacher collection \mathbf{T} of size $(T - T_0 + 1) \times L$.
Where:

$$\mathbf{y}(n + 1) = f^{out}(\mathbf{u}(n + 1), \mathbf{x}(n + 1), \mathbf{y}(n)), \quad (8.16)$$

or without output to output connections

$$\mathbf{y}(n + 1) = f^{out}(\mathbf{u}(n + 1), \mathbf{x}(n + 1)). \quad (8.17)$$

2.5(train_esn).

- Compute output weights. Multiply the pseudoinverse of \mathbf{M} with \mathbf{T} , to obtain a $(K + N + L) \times L$ sized matrix $(\mathbf{W}^{out})^T$ whose $i - th$ column contains the output weights from all network units to the $i - th$ output unit:

$$(\mathbf{W}^{out})^T = \mathbf{M}^{-1}\mathbf{T} \quad (8.18)$$

then

- Transpose $(\mathbf{W}^{out})^T$ to \mathbf{W}^{out} , which is the desired output weight matrix. The learning process has finished.

Step 3. TESTING THE TRAINED NETWORK

3. 1 (test_esn.m).

The matrices \mathbf{W}^{in} , \mathbf{W} , \mathbf{W}^{back} , \mathbf{W}^{out} are now known. Drive the network with a novel sequence $\mathbf{u}(n)$ according to known equations

$$\begin{aligned} \mathbf{x}(n + 1) \\ = f(\mathbf{W}^{in}\mathbf{u}(n + 1) + \mathbf{W}\mathbf{x}(n)) \end{aligned} \quad (8.19)$$

$$\mathbf{y}(n + 1) = f^{out}(\mathbf{u}(n + 1), \mathbf{x}(n + 1)). \quad (8.20)$$

3.2(error_rate_classification.m)

For each testing sequence filter the signal \mathbf{y} . Find the step (r) , $r = 0, 1, \dots, T$ of maximum value of $\mathbf{y}_i(r)$. Make $0.99\mathbf{y}_i(r) = 1$. Suppress values of time steps $[r - 10 \dots r + 10]$ to zero for all \mathbf{y} . Continue to the next output signal. Finally compare it against to ground truth label (\mathbf{y}^{target}). Gestures detected or not. Sum true classified gestures of all test sequences and divided with the whole sum of the gestures that were provided to network during test phase. Multiply by 100. Percentage of true classified gestures exists.

8.6 EXPERIMENTS

8.6.1 ECHO STATE PROPERTY (A SIMPLE EXPERIMENT)

A lot of learning tasks involve some form of short-term memory. We understand this property of some input-output systems, as the current output $\mathbf{y}(n)$ to depend on earlier values $\mathbf{u}(n - k)$ of the input and /or earlier values $\mathbf{y}(n - k)$ of the output itself. Also the dynamic reservoir units activations \mathbf{x}_i can be understood in terms of echo functions \mathbf{e}_i which maps input/output histories to the current state (Jaeger, 2001).

We ran a simple experiment to indicate the significance of spectral radius and how it affects the short-term memory performance of the network. A network with the same parameters design, except the spectral radius, was simulated three times. Spectral radius hand-tuned to 0.3, 0.95 and 1.35 value.

For simulations, an 800-unit sigmoid network was used. Internal weight matrix sparse connectivity was 1%. The input sequence ($\mathbf{u}_1(1), \dots, \mathbf{u}_1(n)$) included 800 steps. At step 200 we provide a unit impulse $\mathbf{u}_1(200) = 1$. All other steps of the input sequence set to zero value. There are no output units assigned to network.

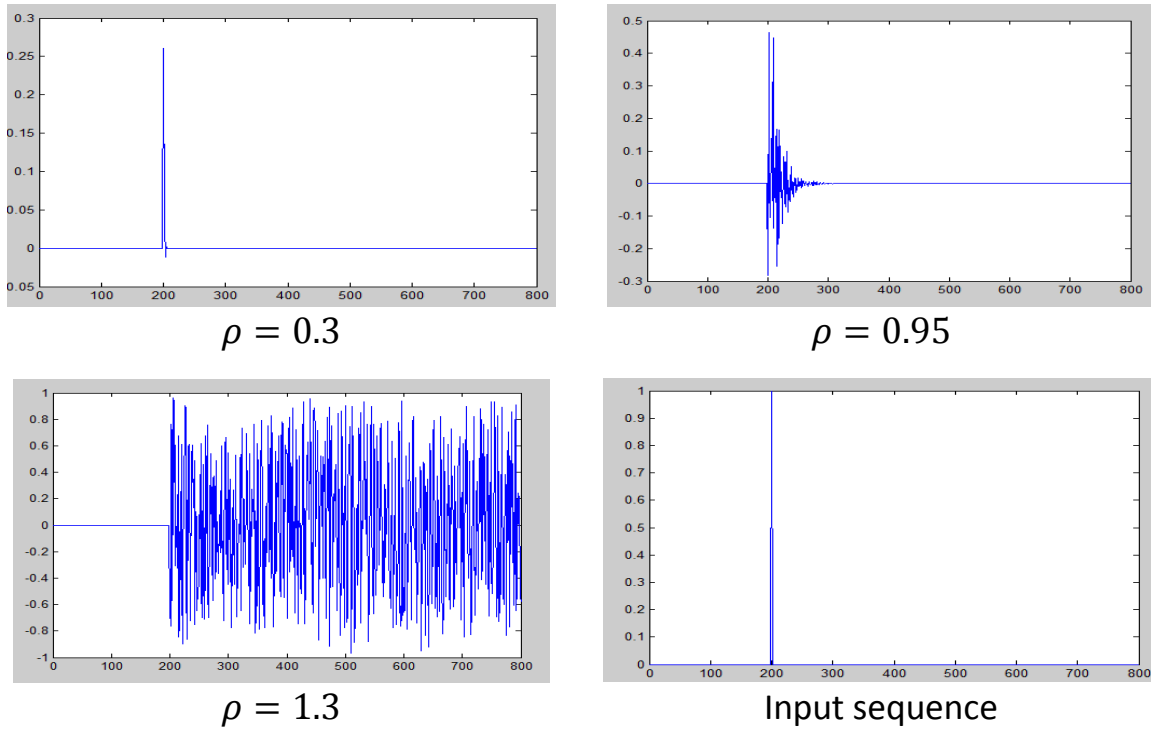


Figure 32. Response of an arbitrarily selected internal unit of the network for different spectral radius ρ . The last trace shows the unit impulse input sequence (training signal).

We observe in Figure 32 that with spectral radius $\rho = 1.3$ the internal units tend to oscillate and network loses its echo state property. If the spectral radius is lower than unit, network retains its echo state property for all $\rho < 1$. For large ρ (close to unit) exhibits a long – lasting response to unit impulse input. The response decays faster for smaller ρ . A relatively long-lasting “echoing” of inputs in the internal network dynamics is a requisite for a sizable short-term memory performance of the network (Jaeger, 2001).

8.6.2 GESTURE CLASSIFICATION EXPERIMENTS

8.6.2.1 EARLY EXPERIMENTS

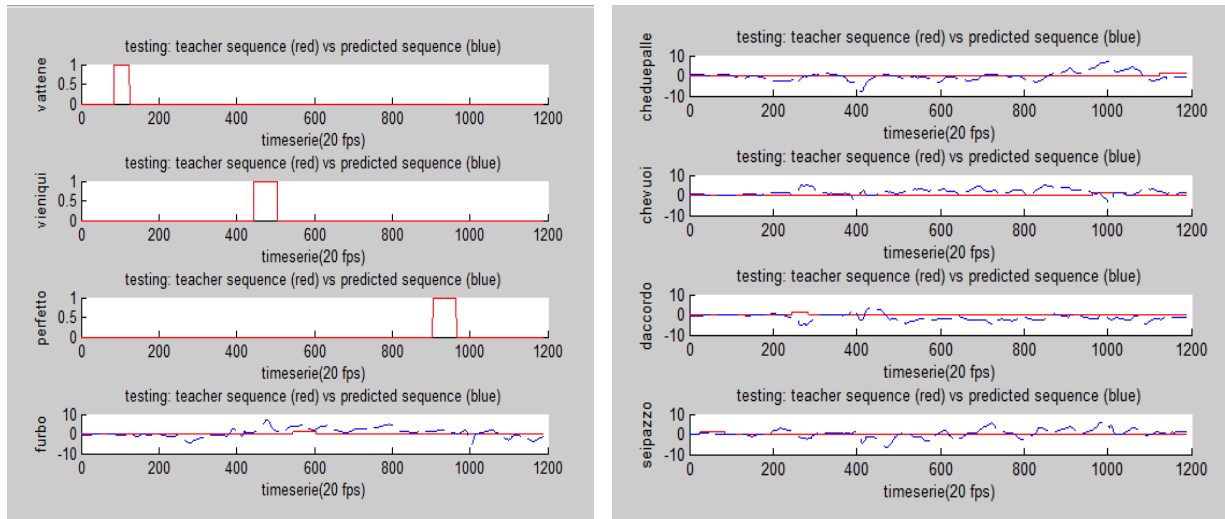
We run a lot of simulations in order to achieve a well-tuned ESN classifier. We try to find out optimal parameters for testing the network. To reach our goal we explore three different types of ESN internal units activations, plain ESN, leaky1 (leaky integrator neurons) and twi_ESN (include time warping function). The modeling task depends primary on the nature of the excited dynamics – it should be adapted to the task at hand. This includes a good judgement on important characteristics of the dynamics inside the dynamic reservoir. These

important characteristics include an appropriate selection of the spectral radius, the visually inspection of internal states and output weights, adding or not an extra constant bias input, the magnitude of input scaling and shifting and adding noise during sampling.

The input connections weights were uniformly distributed to range $[-1 \ 1]$. The network had output feedback connections, which were uniformly distributed to range $[-1 \ 1]$. The other parameters (e.g. input scaling, shifting etc.) of the first simulation of the ESN were chosen close to unity but arbitrarily except spectral radius ($\rho = 0.95$) and the feedback scaling (0.1). The output activation function was linear $f^{out} = id$.

The first 400 steps were discarded and the output weights were computed from the network states collected from $n = 400$ through $n = end$ of training sequence.

Figure 33 shows network output and internal units response.



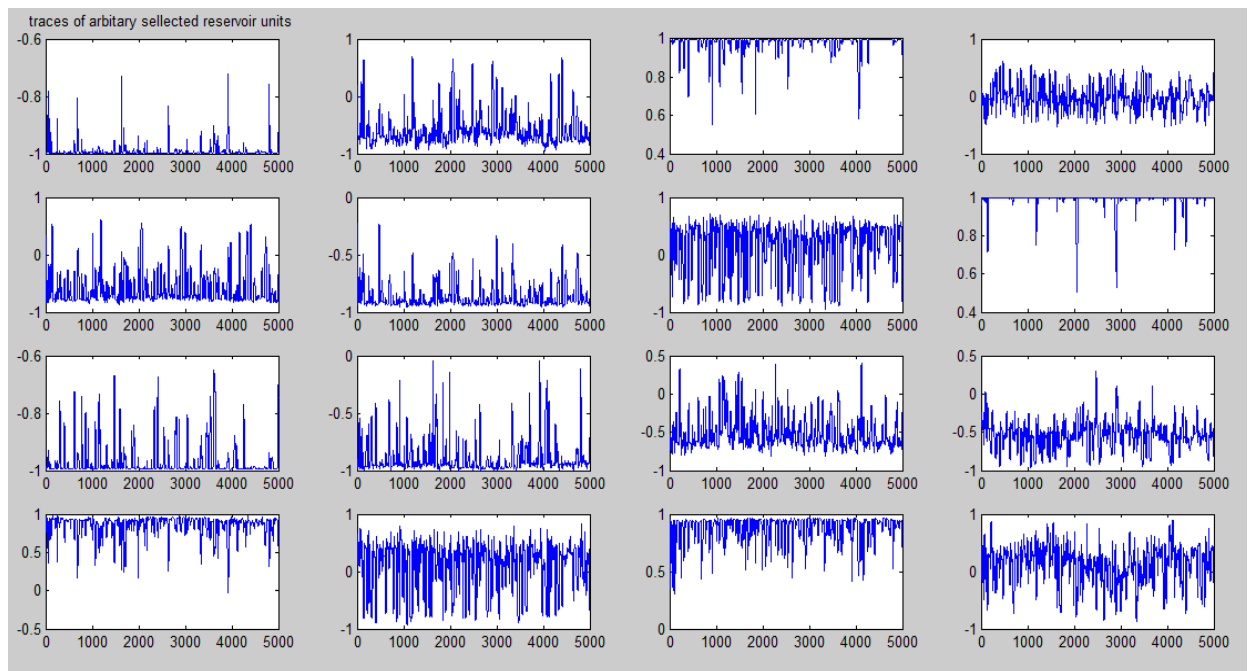
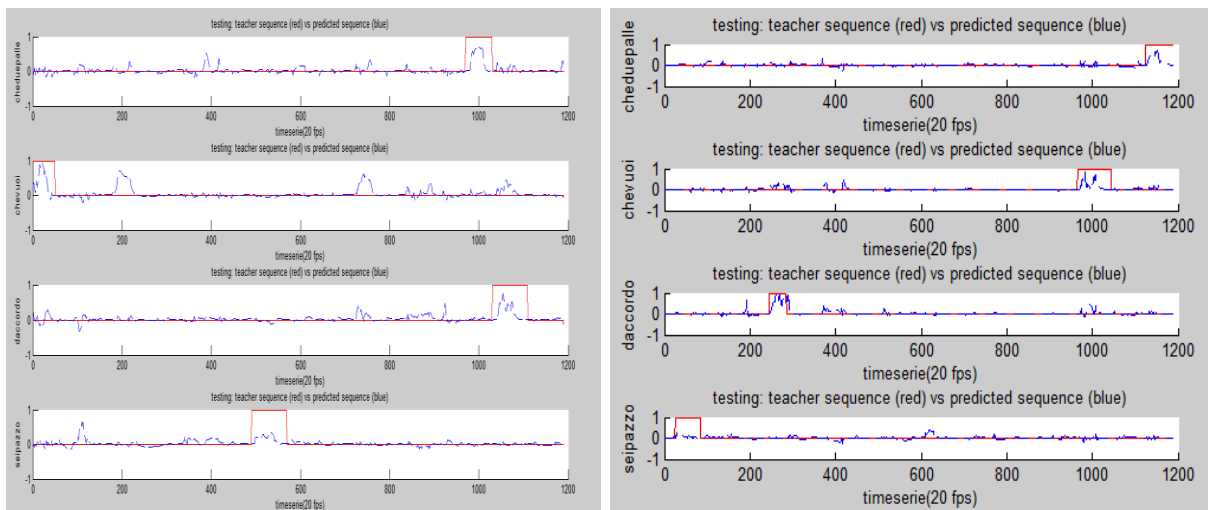


Figure 33. Top row: a portion (8 of 20) of labels (red line) and network output (blue line). Last row: traces of some internal units.

We set the feedback scaling to zero and simulate the network again. Network output and internal units response are represented below:



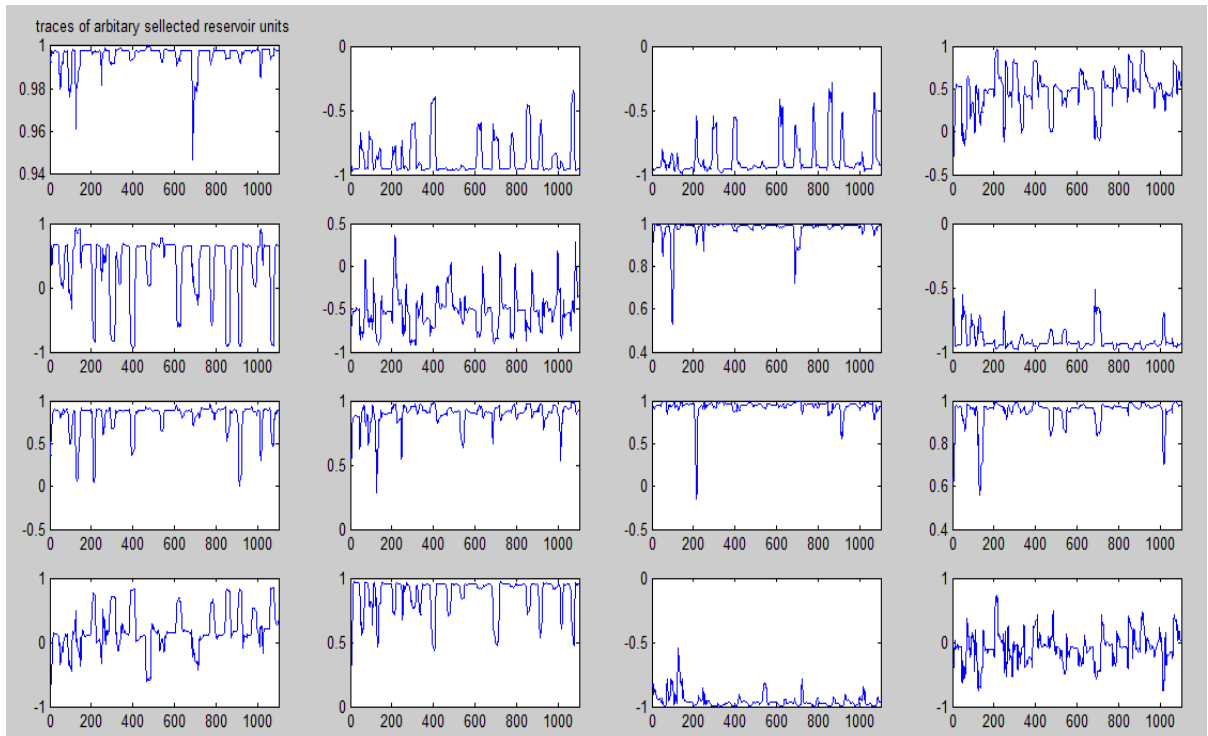


Figure 34. Top row: a portion (8 of 20) of labels (red line) and network output (blue line). Last row: traces of some internal units without feedback connections.

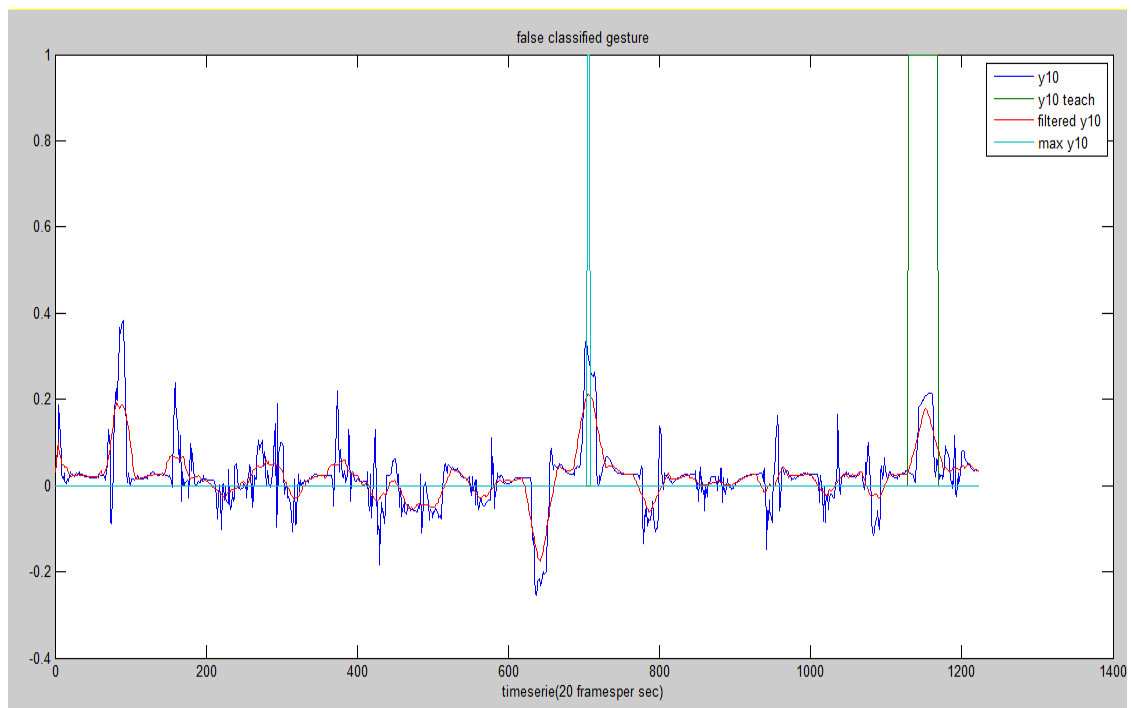
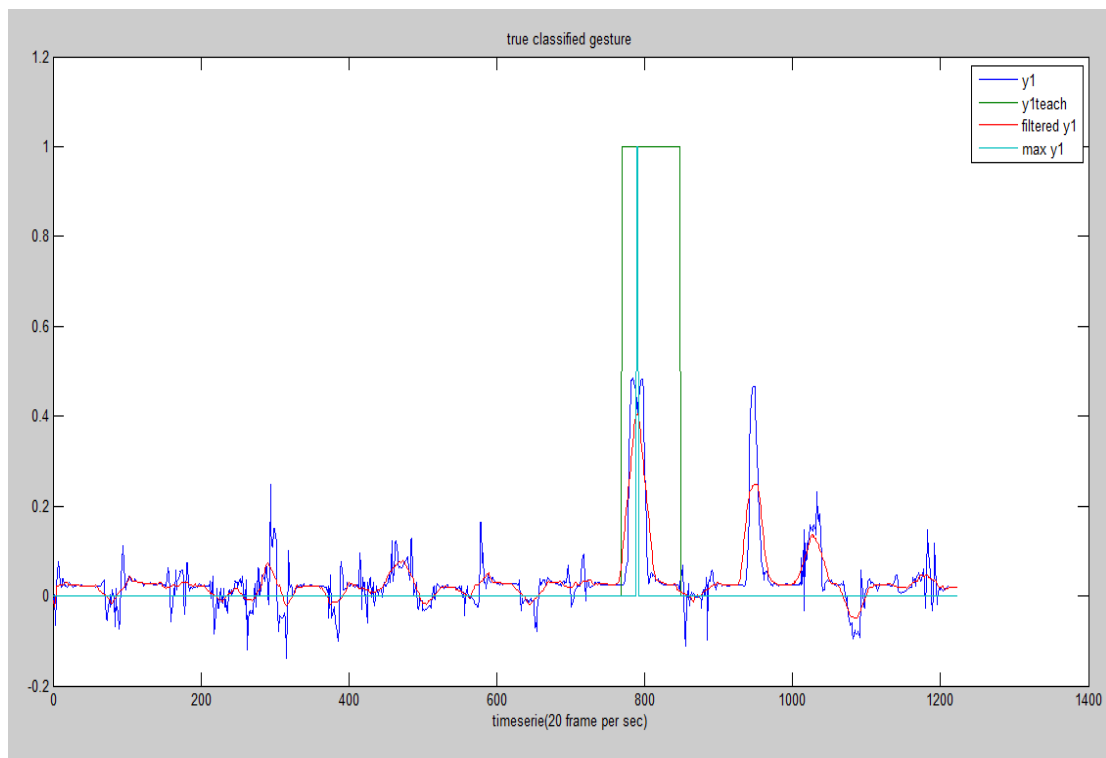


Figure 35. True-false classified gesture.

It is obvious that feedback connections drive the internal units to unstable state (see Figure 33). Because the training readouts are fed back to the

reservoir, they change the dynamics of all the internal units $x(n)$ and thus, of the outputs too. This observation indicates too large values of the output feedback weights. We can also notice that internal units are, in most cases, saturated. This is caused by a large impact of ‘incoming’ output feedback signals to the \tanh internal units. That is, if the inputs are far from the zero the \tanh internal units (neurons) tend to drive to more towards saturation, where they exhibit more nonlinearity. For this kind of learning task (classification pattern recognition) this “switching” type of target dynamics is desirable.

The input signals are a nonlinear sequence. The desired output y^{target} is a linear combination (zero if there is no gesture and unit if there is). This implies a hard learning task so the desired output cannot be learned precisely. The reason for these instabilities issues is that even if the model can’t predict the signal quite accurately, pass through the feedback loop of connections W^{out} and W^{back} small errors get amplified, making $y(n)$ diverge from the intended $y^{target}(n)$.

The magnitude of input weights W^{in} is also important. Input scaling and shifting was selected being 1 and 0 respectively and the signal values are ranged to $[0\ 1]$. In this case, internal units (\tanh) is excited to the linear central part of the sigmoid. In this case (Figure 34) network obtains linear dynamic characteristics. Larger values of W^{in} imply that the network is strongly driven by input and leads internal units closer to the saturation of the sigmoid, which results in a more nonlinear behavior of the resulting model. Even larger values of W^{in} , finally lead internal units into almost pure $-1 / +1$ bi-valued, binary dynamics. Manual adjustment and repeated trial and error processes are required to find the task appropriate scaling setup.

These two trained networks indicate bad tuned learning task. The outputs of the first network oscillate and the second is far away from desired output target.

Both experiments error classification rate is high, measured 92.19% for network with feedback connections and 42.95 % without feedback connections. Further examples showed that (APPENDIX 1) ESN classifier without feedback connections can be slightly improved as opposed to ESN classifier whose performance can be improved dramatically.

8.6.2.2 FURTHER EXPERIMENTS

We continue our experiments to find the appropriate parameters of the network in order to recognize gestures. The experiments reported here were run off-line based on provided training/testing data sets. In every computational experiment setup was the same as the previous except one parameter. This way of experimental setup has the advantage of evaluating the effect of this parameter to the network. There are a lot of parameters that have to be investigated by someone for tuning this network and the effect of each parameter sometimes is not clear.

We evaluate network performance of the three types of internal activation units $x(n)$ over a range of values and designs of the following ESN model parameters:

- Input units weight matrix uniform and Gaussian distribution ranged to interval $[-1 \ 1]$ and bi-valued $[-1 \ 1]$ distribution with probability 0.05 to 1, 0.05 to -1 and 0.9 to 0.
- Feedback units weight matrix uniform distribution range to intervals $[-1 \ 1]$ and $[-0.1 \ 0.1]$
- Spectral radius range $[0.35 \ 0.95]$ with step 0.2.
- Number of internal units range 400 to 750
- Magnitude of leakage range $[0.2 \ 0.9]$ with step 0.1.
- Uniform distribution of time constants vector and manually selected values range $[0 \ 1]$.
- Internal units white noise values 0.001, 0.0001 and 0.000001.
- Target sequence white noise values 0.001, 0.0012, 0.0001 and 0.000001.
- Number of discarding steps due to initial transient: 1350 neurons-500 discarding time steps, 1300 neurons-450 discarding time steps, 1300 neurons-450 discarding time steps, 1250 neurons-400 discarding time steps, 1200 neurons-350 discarding time steps, 1150 neurons-300 discarding time steps, 1100 neurons-200 discarding time steps, 720 neurons-0 discarding time steps.

8.6.3 RESULTS

Here we report ESN sub-optimal parameter setup and best performance for each type of internal units activation:

Internal units activations type	Plain_ESN	Leaky1_ESN	Leaky_ESN	Twl_ESN
Internal unit	<i>id</i>	<i>id</i>	<i>id</i>	<i>id</i>
Input units weight matrix W^{in}	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]
Bias input scaling value s_{scal}^{Binput}	1.0	1.0	1.0	1.0
“active” inputs scaling value s_{scal}^{Rinput}	[1.5]	[2.5]	[1.8]	[1.0]
connectivity	1%	1%	1%	1%
Feedback units weight matrix W^{back}	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]	Uniform distribution range [-1 1]
Feedback scaling value s_{scal}^{back}	-	0.0001	-	0.00017
Number of	720	1000	720	790
Spectral radius ρ	0.95	0.95	0.95	0.45
time constant vector Δt	-	[0.1]	[0.2]	[0.25]
leakage α	-	-	[0.8]	[0.8]
Internal units noise ν	[0]	[0]	[0]	[0]
Target sequence noise ν^{target}	[0]	[0]	[0]	[0]
Moving average filter	15	15	15	15
Error classification rate	30.00%	29.26%	28.53%	31.21%

Table 4. ESN sub-optimal parameter setup and best performance for each type of internal units activation.

And the corresponding error classification rate of each label:

label	Error classification	Plain_ESN	Leaky1_ESN	Leaky_ESN	TwI_ESN
y₁	vattene	36.58%	43.90%	26.82%	36,58%
y₂	vieniqui	39.02%	36.58%	51.21%	36,58%
y₃	perfetto	24.39%	12.19%	9.75%	26.82%
y₄	furbo	29.26%	31.70%	31.70%	39.02%
y₅	cheduepalle	21.95%	24.39%	21.95%	24,39%
y₆	chevuoi	9.75%	12.19%	4.87%	19.51%
y₇	daccordo	7.31%	7.31%	4.87%	2.43%
y₈	seipazzo	43.90%	36.58%	39.02%	24.39%
y₉	combinato	14.63%	9.75%	26.82%	7.31%
y₁₀	freganiente	53.65%	51.21%	41.46%	48.78%
y₁₁	ok	41.46%	48.78%	36.58%	36.58%
y₁₂	cosatifarei	53.65%	48.78%	58.53%	56.09%
y₁₃	basta	12.19%	17.07%	12.19%	19,51%
y₁₄	prendere	46.34%	46.34%	51.21%	46.34%
y₁₅	noncenepiu	63.41%	53.65%	60.97%	63.41%
y₁₆	fame	0.0%	0.0%	0.0%	4.87%
y₁₇	tantotempo	7.31%	2.43%	0.0%	14.63%
y₁₈	buonissimo	29.26%	34.14%	26.82%	36.58%
y₁₉	messidaccordo	53,65%	56.09%	56.09%	56.09%
y₂₀	sonostuf	12.19%	12.19%	9.75%	24.39%

Table 5. Error classification rate of each gesture.

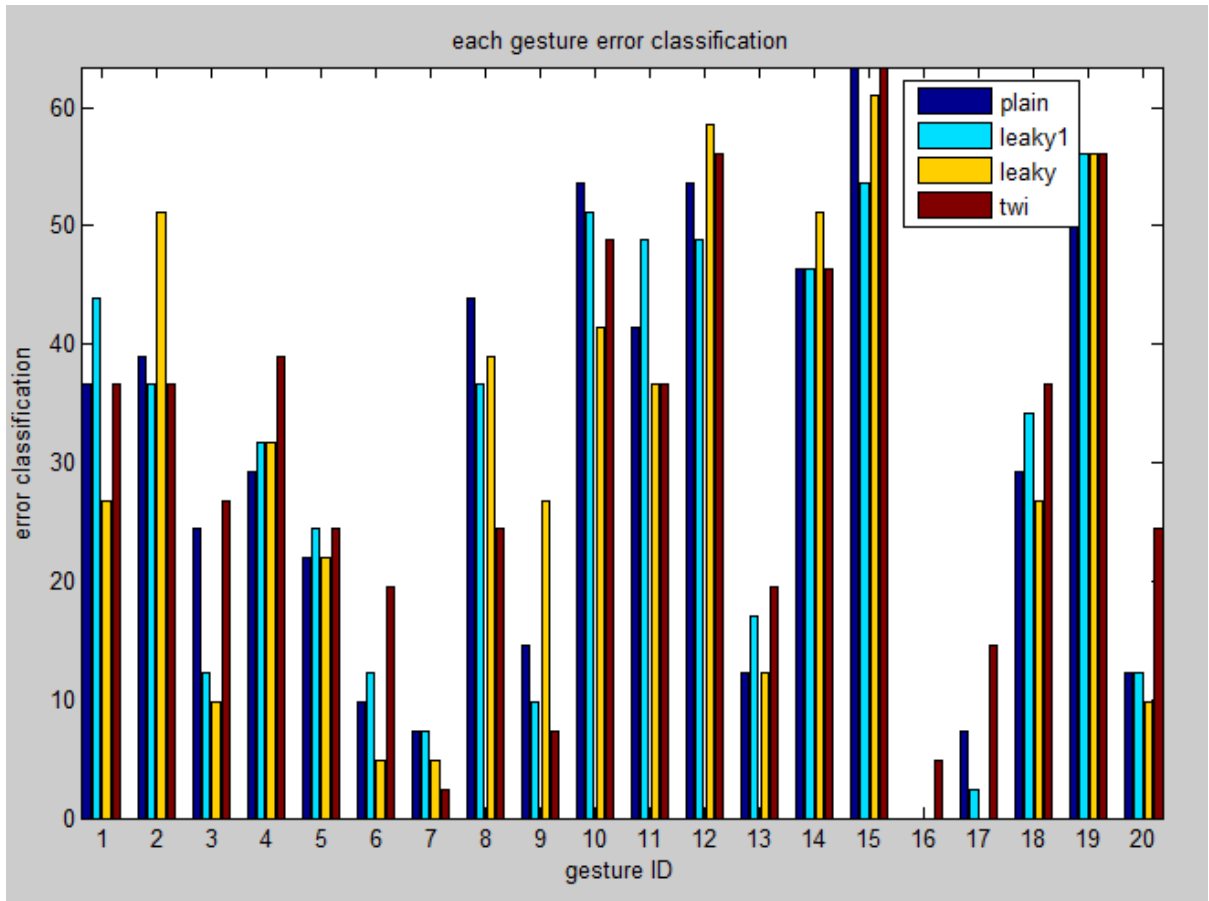


Figure 36. Bar-graph of four different internal units activation error classification rate grouped in each gesture.

Experimental results point out that ESN network provide a reasonable classification hypothesis. That is, after training (using training data) network is capable to recognize (detection plus classification) gestures when exposed to unknown data in testing phase. Finding maximum magnitude of each dimension of the output we manage to recognize 20 different poses which are contained in test input data. Figure 35 shows right and false recognition of an output dimension (class).

Some empirical observations of experimental results which relate to parameters setup are:

- Usage of a bias input (input with constant value) to training and testing input sequences improved network performance 1% roughly (in some cases). This performance was achieved when the value of the bias input was unit. Large bias input value shifted a lot of internal units towards work on a more extreme region of their sigmoids. This fact leads

reservoir to achieve a strongly nonlinear behavior which is preferable for classification learning tasks.

- We treat “active” inputs as a set. That is all so this channel has same scale value. Same method followed for outputs too.
- Network performance slightly changed applying different distributions of input and feedback weight matrices in learning.
- Injecting noise to internal units downscaled network performance (2%-3%). A relative large amount of training data was used for learning. This data comes from experimental measurements, thus contains some random noise components. Adding extra noise led reservoir to “memorize” irrelevant to the task features of the input channels.
- Same parameters setup resimulation, showed tiny variation to network performance (APPENDIX 1. *lines 66,67 , lines 104,98 , lines 99,100 , lines106,107*). This variation is caused by random entries of matrices distributions every time we simulate the network.

This particular learning task focused on classification gestures from segmented data. Experimental results showed that:

- Spectral radius can be scaled in a vast range of values ($0.45 < |\lambda_{max}| < 0.95$) without any significant change in network performance (APPENDIX 1. *lines 16 ... 20, lines 86 – 92 – 93 – 123,124, lines 39,40*). ESN classifier performance seems to be insensitive to duration of Short Term Memory effects.

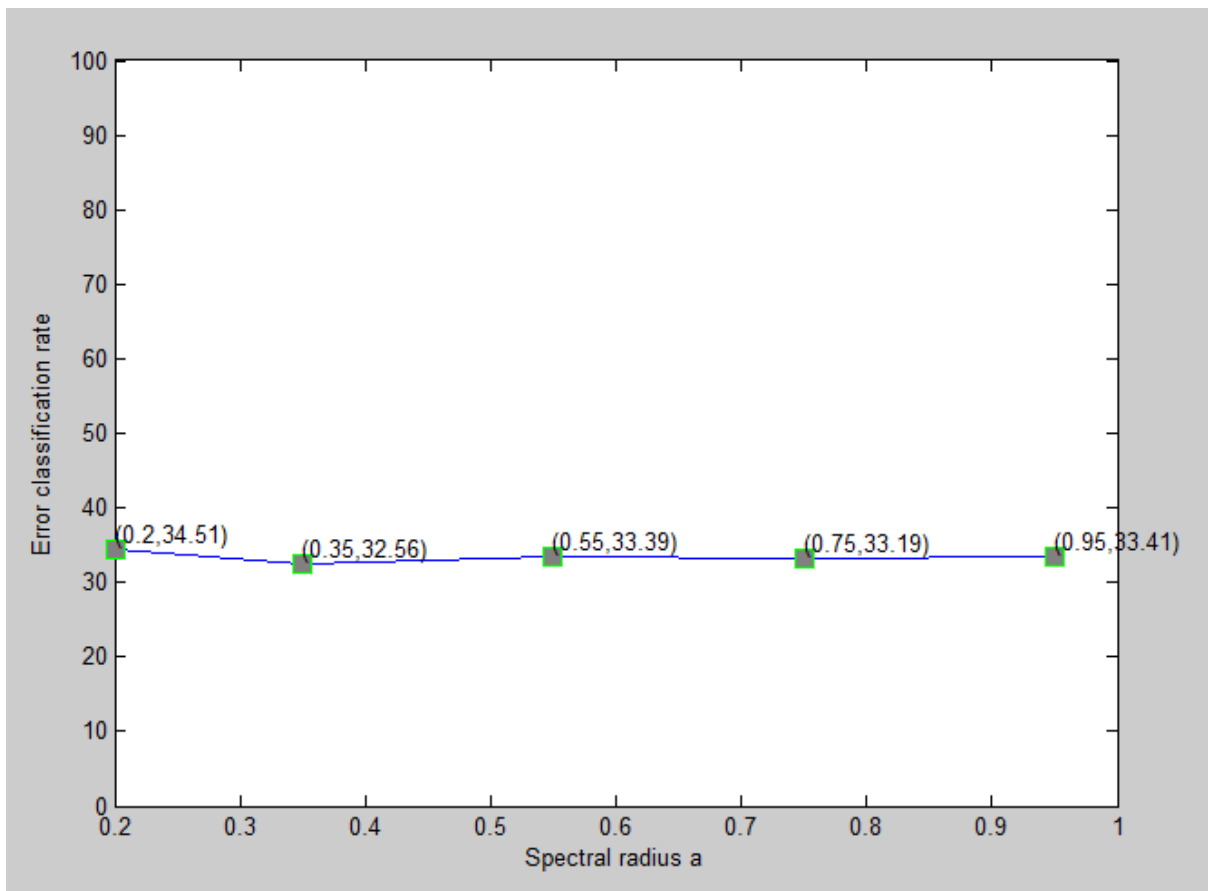


Figure 37. Error classification rate as a function of spectral radius a . Training parameters (except spectral radius) was the same for all executed simulations.

- 720 internal units network reached maximum performance. A network of 400 units performed very poorly (APPENDIX 1. *lines 115 – 122 and 70*). Computations over the number of 720 internal units did not take place due to memory limits of my computer (without discarding initial time steps).

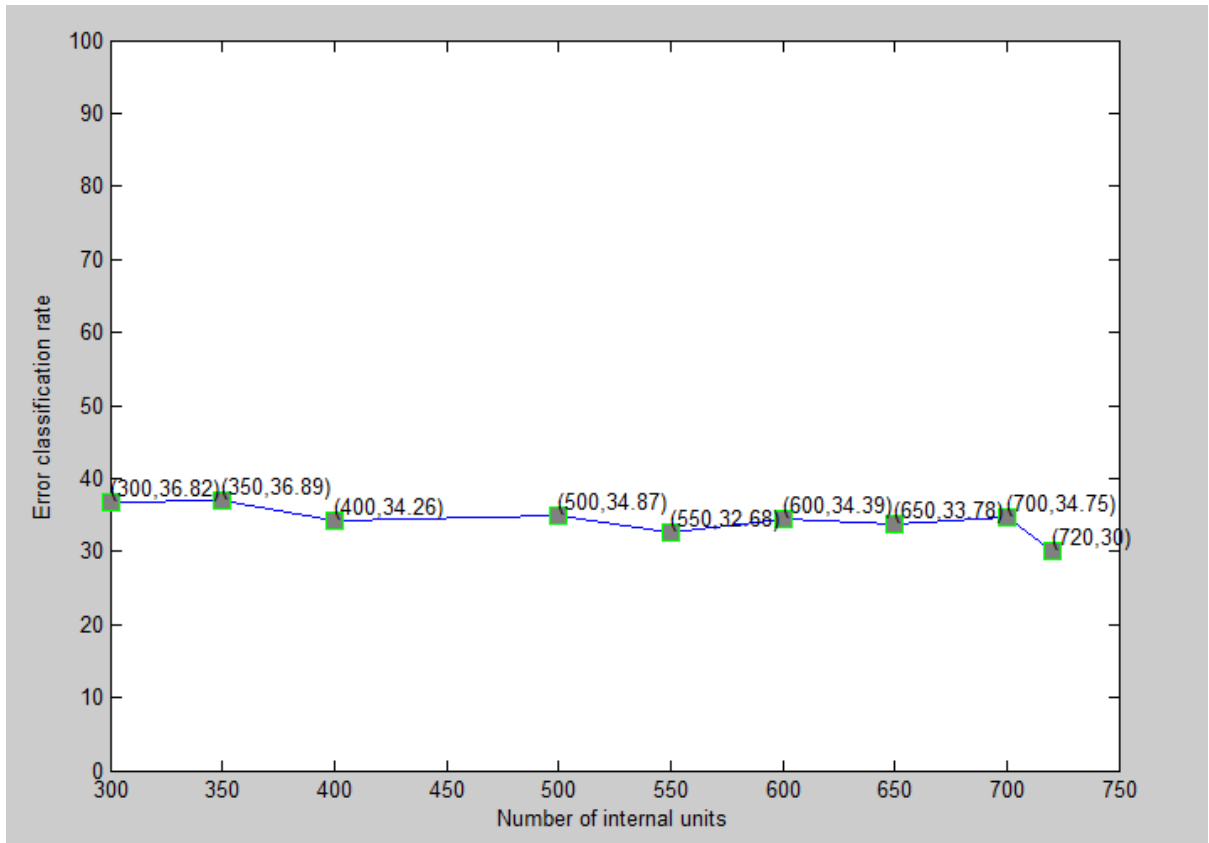


Figure 38. Error classification rate as a function of number of internal units. Training parameters (except number of internal units) was the same for all executed simulations.

- As we describe above data set consists of short separate sequences. Discarding an amount of them due to initial transient leads to loss of precious information. However experiments showed that (APPENDIX 1.lines 9 – 14) network has similar performance with 720 neurons and zero discarding time steps compared to 1000 neurons and 150 initial discarding time steps.
- Between the three different types of internal unit activations there is not a clear winner. They have similar performance value. As we described, input channels consist of 20 subsequences, each of those represents an output class. Every one of the training data sets contains the representations of 20 unique gestures performed by one individual. It is clear that every person has its own way and time scale to perform a gesture.

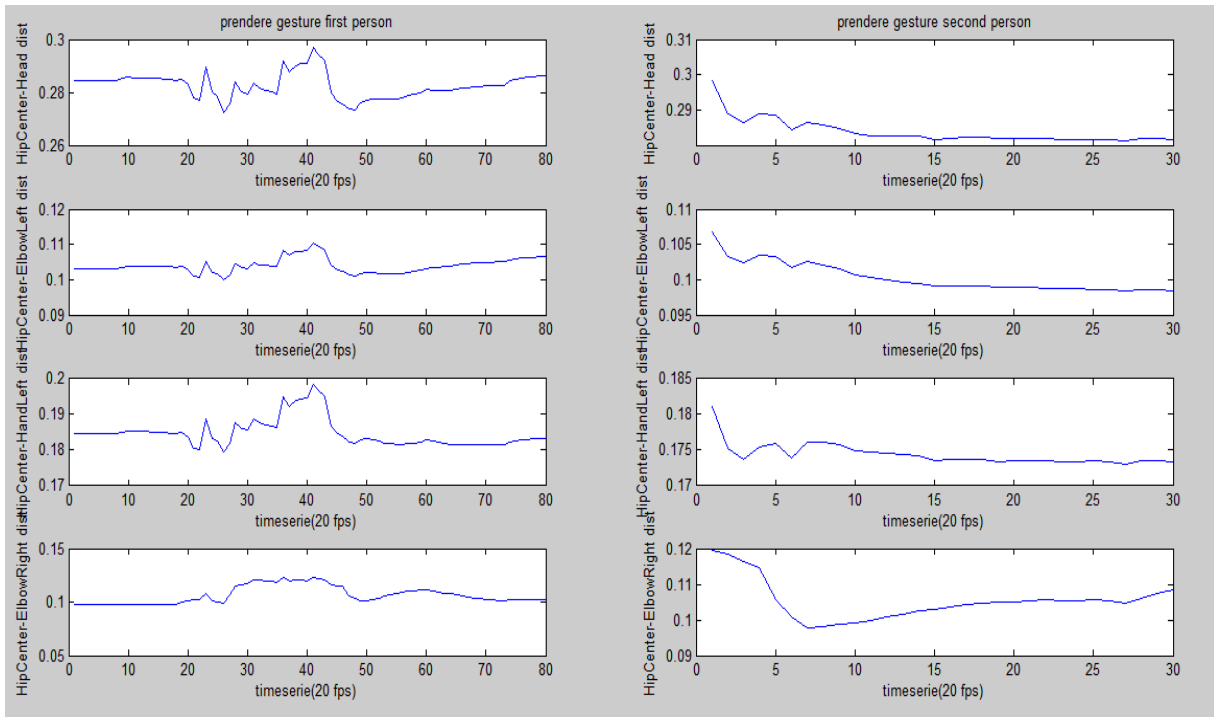


Figure 39. Shows 4 (of twelve) input channels of two different people performing the same gesture. After normalization 4 input channels of different training sets First person needs 80 frames (4 seconds) to complete ‘prendere’ gesture and second one only 30 frames(1.5 seconds).

We understand that is time wrapped training data. ESN Internal units with Time Warping Invariant activations transform wrapped input data to unwrapped. This (we thought) would help the network to perform better to this kind of data, but experimental results do not reveal this conclusion. In (Jaeger, et al., n.d.) it is reported that as the dimensions K of the input sequences increases the benefit of time wrapping is less important. *It is intuitively clear, that the bigger the number of independent input dimensions we have, the less important the role of the actual time axis is, i.e. the loss of temporal information which is intrinsic to TWIESNs becomes less important. In other words, the temporal information can in some sense be deduced from the dynamics of many independent input variables.*

If k is larger than 10 then the time-scaling invariant information of inputs contributes little to the pattern recognition. In other words, the “authentic” temporal information is preferable to the recognizability of patterns than the pseudo-time reform information of the input sequences.

- Providing network with raw data (without rescaling or shifting) or data rescaled to range $[-1 \ 1]$ lead to bad performance training task (30%-40%).
- Optimal input scaling value can be determined into a narrow region without remarkable change of the performance. That is, network obtains same performance with input scaling values 1.2 and 1.4 (e.g. APPENDIX 1. *lines* 100,101,102). Output scaling follows same principle.
- Due to feedback connections network suffers from instability problems. Finding an appropriate value includes many trial-errors. A slight change (0.00002) around the optimal value leads to further instability of the internal units.
- Model performance improved when the internal units tend to saturation. Better performance with saturation internal units can be explained by the binary character (0 or 1) of output vector. In other words, we lead internal units almost always to have values near the saturation point (1 or -1) by applying large input scaling values. This result is desirable in this particular learning task where target dynamics are switching type.
- There are more than one parameter set that network can reach the same performance (e.g. APPENDIX 1 *lines* 70/80).
- Similar network performance is achieved (APPENDIX 1 *lines* 62,81) with and without feedback. But network tuning without feedback is much easier.
- In APPENDIX 1 (*lines* 47 and 48, 98 and 104) we observe that as the “pseudo” time gap Δt increases the performance drops (*Leaky* and *TWI* topologies). This can be explained by the generic problem of linear Euler discretization in eq. (5.21) : if the step size becomes larger, the curvature of the approximate signal trajectories decreases, thus important feature information is lost.

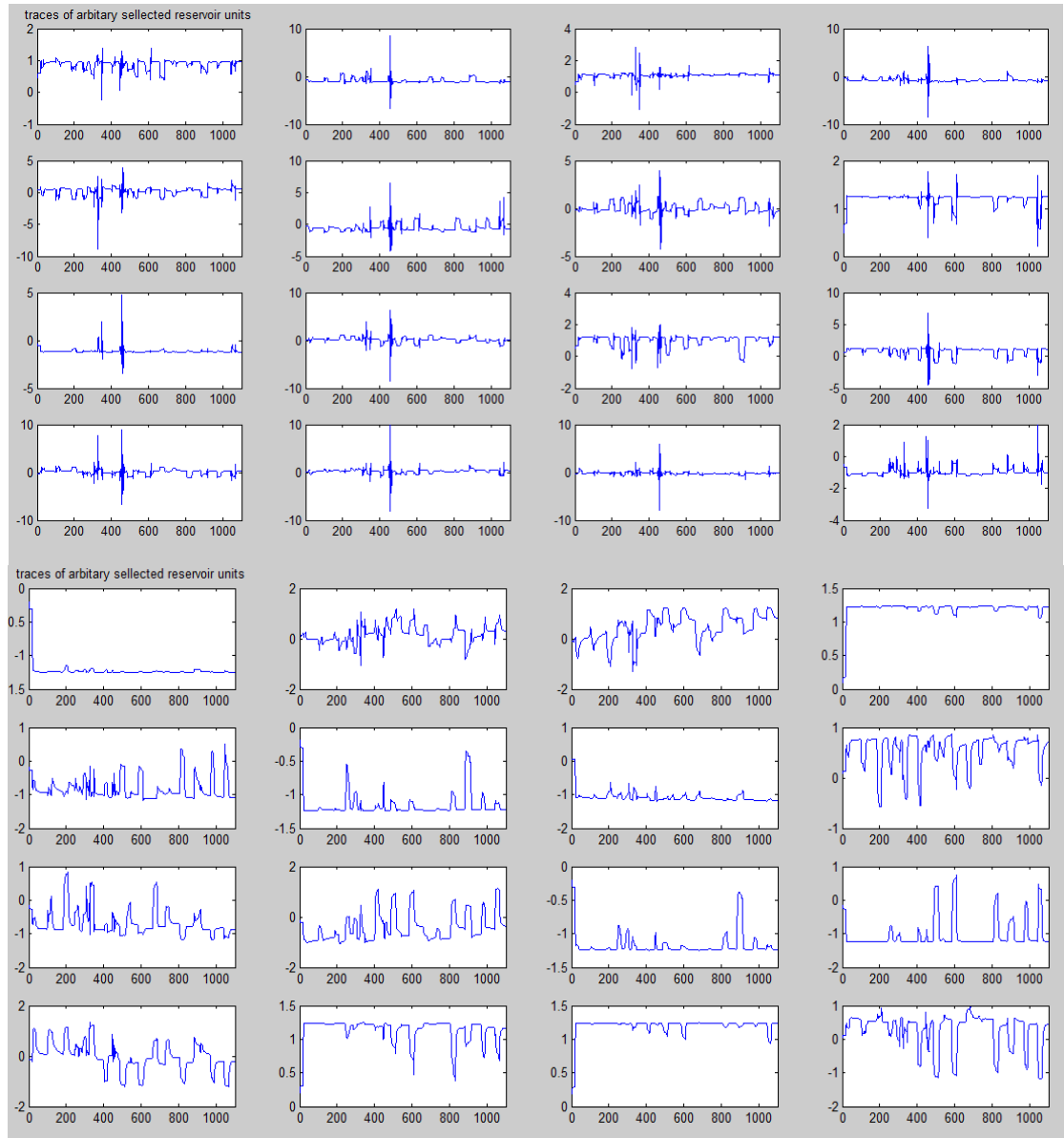


Figure 40. Top: TWI_ESN network internal units activations with time constants $\Delta t=0.5$. Bottom: TWI_ESN network internal units activations with time constants $\Delta t=0.2$.

CHAPTER 9. COMPARISONS

Next, we briefly describe the main components of the top-ranked methods in ChaLearn Multi-modal Gesture Recognition 2013. Then, we compare the error rate result of our proposed single modal framework with other approaches (Escalera, et al., 2013b).

9.1 ChaLearn 3-TOP RANKED APPROACHES

The first-ranked team (IV AMM) used a feature vector based on audio and skeletal information. A simple time-domain end-point detection algorithm based on joint coordinates was applied to segment continuous data sequences into candidate gesture intervals. A HMM was trained with 39-dimension MFCC features and generated confidence scores for each gesture category. A Dynamic Time Warping based skeletal feature classifier was applied to provide complementary information (Escalera, et al., 2013b).

The second-ranked team (WWEIGHT) combined audio and skeletal information, using both joint spatial distribution and joint orientation. They first searched for regions of time with high audio-energy to define time windows that potentially contained a gesture. Feature vectors were defined using a log-spaced audio spectrogram and the joint positions and orientations above the hips (Escalera, et al., 2013b).

The third ranked team (ET) combined the output decisions of two designed approaches. In the first approach, they looked for gesture intervals (unsupervised) using the audio files and extracted features from these intervals (MFCC). Using these features, authors trained a random forest and gradient boosting classifier. The second approach used simple statistics (median, var, min, max) on the first 40 frames of each gesture to build the training samples. The prediction phase used a sliding window. The authors created a weighted average of the output of these two models (Escalera, et al., 2013b).

9.2 COMPARISONS WITH OTHER APPROACHES AND VARIATIONS

Herein we compare the recognition results of our proposed single modal recognition with the final results of ChaLearn Multi-modal Gesture Recognition 2013 as reported in (Escalera, et al., 2013b). At this point, we have to mention that our training data base was limited compared to the complete data base which was used by the participants of the challenge. We performed experiments under a limited RAM setting, which did not allow to use more than 350 data sets (complete data base included 393 data sets).

Table 6 shows the particular strategy for each team. The considered modalities of the three top ranked teams is identical, but the applied classifier differs.

Regarding the considered modalities, none of the participants used only audio data, but the majority used multiple modalities for training. In particular, when no multiple modalities were used for describing the data, skeleton was the most used feature. Final results show that combination of audio plus skeleton data was the best practice. Among single modal approaches our proposed classifier achieved better performance.

TEAM	RANK POSITION	MODALITIES	CLASSIFIER	ERROR
Our	-	Skeleton	NN-ESN	28.53%
IVA MM	1	Audio, Skeleton	HMM, DP, KNN	12.75%
WWEIGHT	2	Audio, Skeleton	RF, KNN	15.38%
ET	3	Audio, Skeleton	Tree, RF, KNN	16.81%
MmM	4	Audio, RGB+ Depth	SVM, Fischer, GMM, KNN	17.21%
PPTK	5	Skeleton, RGB, Depth	GMM, HMM	17.32%
LRS	6	Audio, Skeleton, Depth	NN	17.72%
MMDL	7	Audio, Skeleton	DGM, LR	24.45%
TELEPOINTS	8	Audio, Skeleton, RGB	HMM, SVM	25.84%
CSI MM	9	Audio, Skeleton	HMM	28.91%
SUMO	10	Skeleton	RF	31.65%
GURU	11	Audio, Skeleton, Depth	DP	37.28%
AURINKO	12	Skeleton, RGB	ELM	63.30%
STEVENWUDI	13	Audio, Skeleton	DNN, HMM	74.41%
JACKSPARROW	14	Skeleton	NN	79.31%
JOEWAN	15	Skeleton	KNN	83.77%
MILAN KOVAC	16	Skeleton	NN	87.46%
IAMKHADER	17	Depth	RF	92.02%

Table 6. Team methods and results. HMM: Hidden Markov Models. KNN: Nearest Neighbor. RF: Random Forest. Tree: Decision Trees. ADA: Adaboost variants. SVM: Support Vector Machines. Fisher: Fisher Linear Discriminant Analysis. GMM: Gaussian Mixture Models. NN: Neural Networks. DGM: Deep Boltzmann Machines. LR: Logistic Regression. DP: Dynamic Programming. ELM: Extreme Learning Machines.

CHAPTER 10. CONCLUSION

This study concerns gesture recognition from segmented data. We treated ESN as a gesture classifier. We presented two implementations of internal units update equations, plain_ESN consists a special case of Leaky Integrator units. Overall performance of the network indicates that ESN is capable to detect and classify gestures. ESNs were trained using input segmented data, through a “rich” variety of excitable dynamics, assigned to a specific output, the gesture (represented as 1) and remain zero to all other outputs. Network must combine and remember the important features of input during training in order to obtain a good testing performance. To do so, internal units should realize some kind of dynamic memory. In other words, reservoir must retain in its current dynamic state information about past input features, which can refer back to deep past. This describes an unbounded long lasting dynamic memory.

Detected results as represented in

Table 4 are incomplete in the sense that many aspects of gesture recognition by ESNs are not investigated. Can better performance using either different set of variables or different representation be achieved? Does usage of more input channels improve learning task?

However, the potential of this approach is obvious. This specific learning task is difficult to manipulate completely, because network must detect a gesture and then temporally classify to correct output, from data which are time unwrapped and collections of different individuals.

BIBLIOGRAPHY

Barabasi, A.-L. & Albert, R., 1999. Emergence of scaling in random networks.

Bengio, Y., Simard, P. & Frasconi, P., 1994. s.l., IEEE Transactions on Neural Networks,5(2):157-166.

Bolt, R. A. & Herranz, E., 1992. *Two-handed gesture in multi-modal natural dialog.* , In Proceedings of the 5th annual ACM symposium on User interface software and technology, pages 7–14. ACM.

Buonomano, D. V. M. M. M., 1995. Temporal information transformed into a spatial code by a neural network with realistic properties. In: s.l.:Science,267:1028{1030.

Cybenko, G., 1989. *Approximation by superpositions of a sigmoidal function..* s.l.:Mathematics of Control,Signals and Systems, 2:303-314.

Dominey, P. F., 1995. *Complex sensory-motor sequence learning based on recurrent.* s.l., Biological Cybernetics, 73:265-274.

Doya, K., 1992. *Bifurcations in the learning of recurrent neural networks..* s.l., In Proceed-ings of IEEE International Symposium on Circuits and Systems 1992, volume 6,pages 2777-2780.

Escalera, S. et al., 2013b. *Multi-modal Gesture Recognition Challenge 2013: Dataset and Results..* s.l., in 15th ACM Int'l Conf. on Multimodal Interaction (ICMI), ChaLearn Challenge and Wrksp on Multi-modal Gesture Recognition. ACM,.

Fang, Y., Wang, K., Cheng, J. & Lu, H., 2007. *A real-time hand gesture recognition method.* s.l., In Multimedia and Expo, 2007 IEEE International Conference on, pages 995–998. IEEE.

Graves, A., 2008. *Supervised Sequence Labelling with Recurrent Neural Networks.* s.l., PhD thesis, Technical University Munich, Munich, Germany,.

Haeusler, S. & Maass, W., 2007. *A statistical analysis of information processing properties of lamina-specificcortical microcircuit models.* s.l.:Cerebral Cortex, 17(1):149{162, 2007. ISSN 1047-3211.

- Hecht-Nielsen, R., 1989. *Neural Network Primer:Part I*. s.l.:All expert.
- Hermans, M. & Schrauwen, B., 2010. *Memory in reservoirs for high dimensional input*. s.l., Proceedings of the IEEE International Joint Conference on Neural Networks,pages -{7.
- Jaeger, H., 2001. *The "echo state" approach to analysing and training recurrent*, s.l.: s.n.
- Jaeger, H., 2002b. *A tutorial on training recurrent neural networks, covering BPTT,RTRL, EKF and the "echo state network" approach.*, s.l.: s.n.
- Jaeger, H., 2007b. *Scholarpedia*. [Online] Available at: http://www.scholarpedia.org/article/Echo_state_network.
- Jaeger, H. & Haas, H., 2004. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication..
- Jaeger, H., Lukosevicius, M. & Popovici, D., n.d. *Optimization and Applications of Echo State Networks with Leaky Integrator Neurons Integrator Neurons*, s.l.: International University Bremen,School of Engineering and Science,28759 Bremen, Germany.
- Jaeger, H., Lukosevicius, M., Popovici, D. & Siewert, U., 2007a. *OptiOptimization and applications of echo state networks with leaky-integrator neurons..* s.l.:Neural Networks, 20(3):335-352.
- Just, A., Rodriguez, Y. & Marcel, S., 2006. *Hand posture classification and recognition using the modified census transform..* s.l., In Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on, pages 351–356. IEEE,, p. .
- Kaiser, M. & Hilgetag., C. C., 2004. Spatial growth of real-world networks.
- Karmarkar, U. R. & Buonomano, D. V., 2007. *Timing in the absence of clocks:encoding time in neural network states..* s.l.:Neuron, 53(3):427-438.
- Kuroda, T. et al., 2004. *Consumer price data-glove for sign*. s.l., In Proc. of 5th Intl Conf.Disability, Virtual Reality Assoc. Tech., Oxford, UK,pages 253–258,.
- Liebald, B., 2004. Exploration of effects of different network topologies on theESN signal crosscorrelation matrix spectrum..

Lukosevicius, M., 2012. *A Practical Guide to Applying Echo State Networks*, s.l.: Neural Networks: Tricks of the Trade, Reloaded. G. Montavon, G. B. Orr, and K.-R. Müller, editors, Springer.

Lukosevicius, M., 2012. *Reservoir Computing and Self-Organized Reservoir Computing and Self-Organized*. s.l.: School of Engineering and Science.

Maass, W., Joshi, P. & Sontag, E. D., 2006. Principles of real-time computing with feedback applied to cortical microcircuit models. In: s.l.: Processing Systems 18 (NIPS 2005), pages 835-842. MIT Press, Cambridge, MA.

Maass, W., Natschlager, T. & Markram, H., 2002. *Real-time computing without stable states: a new framework for neural computation based on perturbations*. s.l.: Neural Computation, 14(11):2531-2560, 2002. ISSN 0899-7667.

Maass, W., Natschlager, T. & Markram, H., 2003. A model for real-time computation in generic neural microcircuits.

Maass, W., Natschlager, T. & Markram, H., 2004. *Computational models for generic cortical microcircuits*. s.l.: Computational Neuroscience: A Comprehensive Approach, pages 575-605. Chapman & Hall/CRC.

Mitra, S. & Acharya, T., 2007. *Gesture recognition: A survey*. *Systems, Man, and Cybernetics*. Part C: Applications and Reviews, IEEE Transactions on, 37(3):311-324, s.n.

Natschlager, T., Markram, H. & Maass, W., 2002. Computer models and analysis tools for neural microcircuits. In: s.l.: A Practical Guide to Neuroscience Databases and Associated Tools, chapter 9. Kluwer Academic Publishers (Boston).

Schraudolph, N., Dayan, P. & Sejnowski, T., 1994. *Using TD(λ) to learn an evaluation function of the game of Go*. s.l.: NIPS-6.

Schreiber, M., Wilamowitz-Moellendorff, M. V. & Bruder, R., 2009. New interaction concepts by using the wii remote. In: s.l.: Interaction Methods and Techniques, pages 261-270. Springer.

Steil, J. J., 2004. *Backpropagation-decorrelation: recurrent learning with $O(N)$ complexity*. s.l., Proceedings of the IEEE International Joint Conference on Neural Networks, 2004 (IJCNN 2004), volume 2, pages 84-848.

Sun, G.-Z., Chen, H.-H. & Lee, Y.-C., 1993. *Time warping invariant neural network*, s.l.: Advances in Neural Information Processing Systems 5,[NIPS Conference] (pp. 180–187). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Verstraeten, D., 2009-2010. *Reservoir Computing: computation with dynamical systems*. s.l.:s.n.

Verstraeten, D., Schrauwen, B., D'Haene,, M. & Stroobandt, D., 2007a. *An experimental unification of reservoir computing methods*. s.l., Neural Networks,20(3):391-403.

Verstraeten, D., Schrauwen, B., Stroobandt, D. & Campenhout, J. V., 2005b. *Isolated word recognition with the liquid state machine: a case study*. s.l., Information Processing Letters, 95(6):521-528,.

Watts, D. J. S. S. H., 1998. Collective dynamics of 'small-world networks..

Yamato, J., Ohya, J. & Ishii, K., 1992. *Recognizing human action in time-sequential images using hidden markov model*. s.l., In Computer Vision and Pattern Recognition,1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on, pages 379–385. IEEE.

Yang, M. H. & Ahuja, N., 2001. *Recognizing hand gestures using motion trajectories*. s.l., In Face Detection and Gesture Recognition for Human-Computer Interaction pages 53–81. Springer.

Yildiz, I. B., Jaeger, H. & Kiebel, S. J., 2012. *Re-Visiting the Echo State Property*, s.l.: s.n.

number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	S_{shift}^{teach}	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	W^{in} Distri- bution	W^{back} Distribution	E^{class}
1	P	950	0.95	[1.0]	[0]	1.0	0	[1]	[0]	[0.1]	[0.8]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	92.19%
2	L1	950	0.95	[1.0]	[0]	1.0	0	[1]	[0]	[0]	[0.8]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	42.95%

3	L	720	0.95	[2.5]	[0.2]	1.0	0.2	[1.4]	[0.1]	[0.005]	[0.15]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-0.2 0.2] U	53.68%
4	L	720	0.95	[2.5]	[0.2]	1.0	0.2	[1.4]	[0.1]	[0.004]	[0.15]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-0.2 0.2] U	42.96%
5	L	720	0.95	[2.65]	[0.2]	1.0	0.2	[1.4]	[0.1]	[0.0001]	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	35.12%
6	L	720	0.95	[2.65]	[0.1]	1.0	0.1	[1.4]	[0.1]	[0.0001]	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	35.12%
7	L	1100	0.45	[2.65]	[0.1]	1.0	0.1	[1.4]	[0.1]	[0.0001]	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.51%
8	L	1150	0.45	[2.65]	[0.1]	1.0	0.1	[1.4]	[0]	[0.0001]	[0.8]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.17%
9	L1	1200	0.45	[2.65]	[0.1]	1.0	0.1	[1.4]	[0]	[0.0001]	[0.8]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	35.24%
10	L1	1250	0.95	[2.5]	[0.1]	1.0	0.1	[1.4]	[0]	[0.0001]	[0.8]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.90%
11	L1	1300	0.95	[2.65]	[0.1]	1.0	0.1	[1.4]	[0]	[0.00095]	[0.15]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	37.31%
12	L1	1350	0.95	[2.5]	[-0.2]	1.0	-0.2	[1.4]	0.1	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.56%
13	L1	1000	0.95	[2.5]	[-0.2]	1.0	-0.2	[1.4]	0.1	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	29.26%
14	L1	720	0.95	[2.5]	[-0.2]	1.0	-0.2	[1.4]	0.1	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.24%
15	L1	720	0.95	[2.5]	[-0.2]	1.0	-0.2	[1.4]	0.1	[0.0001]	[0.1]	-	[0]	[0.0012]	<i>id</i>	[-1 1] U	[-1 1] U	38.41%
16	L1	720	0.95	[2.5]	[-0.2]	1	-0.2	[1.4]	[0]	[0.0001]	[0.1]	-	[0]	[0.0001]	<i>id</i>	[-1 1] U	[-1 1] U	33.41%
17	L1	720	0.75	[2.5]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.89%
18	L1	720	0.55	[2.5]	[-0.2]	1.0	-0.2	[1.4]	[0]	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.39%
19	L1	720	0.35	[2.5]	[-0.2]	1.0	-0.2	[1.4]	[0]	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.56%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	\mathbf{W}^{in} Distribution	\mathbf{W}^{back} Distribution	E^{class}
20	L1	720	0.20	[2.5]	[-0.2]	1.0	-0.2	[1.4]	[0]	[0.0001]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.51%
21	L1	720	0.95	[2.4]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.1]	-	[0.000001]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.90%
22	L1	720	0.95	[2.4]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.1]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.02%

23	L1	720	0.45	[2.4]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.1]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.51%
24	L1	720	0.95	[1.3]	[0]	1.0	0	[1.3]	[0]	[0.002]	[0.5 0.7] U	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.87%
25	L1	720	0.45	[1.3]	[0]	1.0	0	[1.3]	[0]	[0.0015]	[0.5 0.7] U	-	[0]	[0]	$tanh$	[-1 1] U	[-1 1] U	37.68%
26	L1	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.0015]	[0.2 0.3] U	-	[0]	[0]	$tanh$	[-1 1] U	[-1 1] U	31.95%
27	L1	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0.3]	[0.0015]	[0.2 0.3] U	-	[0]	[0.01]	$tanh$	[-1 1] U	[-1 1] U	33.41%
28	L1	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0]	[0.0015]	[0.2 0.3] U	-	[0]	[0.001]	$tanh$	[-1 1] U	[-1 1] U	32.23%
29	L1	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0]	[0.0015]	[0.2 0.3] U	-	[0]	[0.0001]	$tanh$	[-1 1] U	[-1 1] U	32.92%
30	L1	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0]	[0.0015]	[0.2 0.3] U	-	[0]	[0.00001]	$tanh$	[-1 1] U	[-1 1] U	34.92%
31	L1	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0]	[0.0015]	[0.2 0.9] U	-	[0]	[0.001]	$tanh$	[-1 1] U	[-1 1] U	34.87%
32	L1	1050	0.95	[2.3]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	[0.0001]	[0.1]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	33.04%
33	L1	1100	0.95	[2.3]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	[0.0001]	[0.9]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.02%
34	L1	1150	0.95	[2.2]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	[0.0002]	[0.8]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	33.17%
35	L1	900	0.95	[2.2]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	[0.0002]	[0.8]	-	[0]	[0]	id	[-1 1] U	[-1 1] U	33.65%
36	T	900	0.95	[2.4]	[0.2]	1.0	0.2	[1.4]	[0.1]	[0.0001]	[0.15]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.87%
37	T	950	0.95	[2.2]	[0.2]	1.0	0.2	[1.4]	[0.1]	[0.00095]	[0.15]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	35.36%
38	T	950	0.95	[2.2]	[0.2]	-	-	[1.4]	[0.1]	[0.00095]	[0.15]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	35.24%
39	T	950	0.95	[2]	[0]	-	-	[1.4]	[0]	[0.00095]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	33.78%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	W^{in} Distribution	W^{back} Distribution	E^{class}
40	T	950	0.45	[2]	[0]	-	-	[1.4]	[0]	[0.00095]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.87%
41	T	950	0.45	[1.8]	[0]	-	-	[1.7]	[0]	[0.00095]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.82%
42	T	950	0.45	[1.5]	[0]	-	-	[1.8]	[0]	[0.00015]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	32.56%

43	T	950	0.45	[1.5]	[0]	1.0	0	[1.8]	[0]	[0.00015]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	33.29%
44	T	800	0.45	[1.5]	[0]	1.0	0	[1.8]	[0]	[0.00015]	[0.25]	[0.8]	[0.0001]	[0]	id	[-1 1] U	[-1 1] U	34.39%
45	T	800	0.45	[1.5]	[0]	1.0	0	[1.8]	[0]	[0.00015]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	33.41%
46	T	790	0.45	[1]	[0]	1.0	0	[1.8]	[0]	[0.00017]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	31.21%
47	T	790	0.95	[1]	[0]	1.0	0	[1.9]	[0]	[0.00017]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	36.46%
48	T	790	0.95	[1]	[0]	1.0	0	[1.9]	[0]	[0.0002]	[0.95]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	95.36%
49	T	790	0.95	[1]	[0]	1.0	0	[0.9]	[0]	[0.0002]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	36.34%
50	T	750	0.95	[0.9]	[0]	1.0	0	[1.9]	[0]	[0.0003]	[0.25]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	36.58%
51	T	720	0.95	[2.4]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.1]	[0.8]	[0.000001]	[0]	id	[-1 1] U	[-1 1] U	36.21%
52	T	720	0.45	[2.4]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.10.03] U	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.26%
53	T	720	0.45	[2.5]	[0]	1.0	0	[1.4]	[0]	[0.00016]	[0.10.03] U	[0.2]	[0]	[0]	id	[-1 1] U	[-1 1] U	40.24%
54	T	720	0.45	[1]	[0]	1.0	0	[2]	[0]	[0.00024]	[0.10.03] U	[0.9]	[0]	[0]	id	[-1 1] U	[-1 1] U	37.19%
55	T	720	0.45	[1]	[0]	1.0	0	[2.2]	[0]	[0.00028]	[0.10.03] U	[0.9]	[0]	[0]	id	[-1 1] U	[-1 1] U	39.26%
56	P	720	0.95	[1.3]	[0.3]	1.0	0.3	[1.8]	[0]	[0.001]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	33.53%
57	P	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.001]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	33.78%
58	P	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	31.70%
59	P	720	0.95	[1.3]	[0]	0.7	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	33.17%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	W^{in} Distribution	W^{back} Distribution	E^{class}
60	P	720	0.95	[1.3]	[0]	0.5	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	33.65%
61	P	720	0.95	[1.3]	[0]	0.2	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	32.21%
62	P	720	0.95	[1.3]	[0]	1.2	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.001]	\tanh	[-1 1] U	[-1 1] U	31.09%

63	P	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.005]	<i>tanh</i>	[-1 1] U	[-1 1] U	34.63%
64	P	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.0012]	<i>tanh</i>	[-1 1] U	[-1 1] U	32.92%
65	P	720	0.95	[1.3]	[0]	1.0	0	[1.8]	[0]	[0.005]	-	-	[0]	[0.0012]	<i>tanh</i>	[-1 1] U	[-1 1] U	33.24%
66	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.31%
67	P	720	0.95	[2.0]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.04%
68	P	720	0.95	[2.5]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.65%
69	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.8]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.56%
70	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.0]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.00%
71	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.2]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.31%
72	P	720	0.35	[1.5]	[-0.2]	1.0	-0.2	[2.0]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.29%
73	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.3]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.29%
74	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.0]	[0.1]	-	-	-	[0]	[0.0001]	<i>id</i>	[-1 1] U	[-1 1] U	32.56%
75	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.4]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.53%
76	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.8]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.53%
77	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.0]	[0.1]	-	-	-	[0]	[0]	<i>tanh</i>	[-1 1] U	[-1 1] U	33.17%
78	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.4]	[0.1]	-	-	-	[0]	[0]	<i>tanh</i>	[-1 1] U	[-1 1] U	34.02%
79	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[2.8]	[0.1]	-	-	-	[0]	[0]	<i>tanh</i>	[-1 1] U	[-1 1] U	31.34%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	$S_{scal}^{Bininput}$	$S_{shift}^{Bininput}$	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	v^{res} Uniform distribution	v^{tar} Uniform distribution	f^{out}	W^{in} Distribution	W^{back} Distribution	E^{class}
80	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[3.2]	[0.1]	-	-	-	[0]	[0]	<i>tanh</i>	[-1 1] U	[-1 1] U	30.12%
81	P	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[3.6]	[0.1]	-	-	-	[0]	[0]	<i>tanh</i>	[-1 1] U	[-1 1] U	30.73%
82	L	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.6]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.73%

83	L	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.3]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.85%
84	L	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.36%
85	L	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	31.70%
85	L	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[0.5]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.39%
86	L	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	31.46%
87	L	720	0.95	[2.2]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	31.58%
88	L	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	28.53%
89	L	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.3]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	31.09%
90	L	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.4]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	29.87%
91	L	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.5]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	31.09%
92	L	720	0.75	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.00%
93	L	720	0.55	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	28.41%
94	L1	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	30.12%
95	T	720	0.95	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.41%
96	T	720	0.95	[1.5]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.51%
97	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.78%
98	T	720	0.95	[1.0]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	33.17%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	\mathbf{W}^{in} Distribution	\mathbf{W}^{back} Distribution	E^{class}
99	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.02%
100	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.09%
101	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.3]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.02%

102	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.5]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.39%
103	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.4]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	0%
104	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.1]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.63%
105	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.05]	[0.8]	[0]	[0]	id	[-1 1] U	[-1 1] U	35.97%
105	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.9]	[0]	[0]	id	[-1 1] U	[-1 1] U	31.09%
106	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[1.0]	[0]	[0]	id	[-1 1] U	[-1 1] U	32.92%
107	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[1.0]	[0]	[0]	id	[-1 1] U	[-1 1] U	34.63%
108	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[1.3]	[0]	[0]	id	[-1 1] U	[-1 1] U	38.04%
109	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.7]	[0]	[0]	id	[-1 1] U	[-1 1] U	33.04%
110	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.6]	[0]	[0]	id	[-1 1] U	[-1 1] U	35.73%
111	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.5]	[0]	[0]	id	[-1 1] U	[-1 1] U	40.36%
112	T	720	0.95	[1.2]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.4]	[0]	[0]	id	[-1 1] U	[-1 1] U	47.43%
113	T	720	0.95	[1.1]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.9]	[0]	[0.0001]	id	[-1 1] U	[-1 1] U	33.78%
114	T	720	0.95	[1.1]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.9]	[0]	[0.00001]	id	[-1 1] U	[-1 1] U	33.65%
115	T	720	0.95	[1.1]	[-0.2]	1.0	-0.2	[1.1]	[0.1]	-	[0.2]	[0.9]	[0]	[0.01]	id	[-1 1] U	[-1 1] U	32.80%
115	P	700	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.75%
116	P	650	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	id	[-1 1] U	[-1 1] U	33.78%
number	type	N	ρ	S_{scal}^{Rinput}	S_{shift}^{Rinput}	S_{scal}^{Binput}	S_{shift}^{Binput}	$S_{scal}^{teacher}$	$S_{shift}^{teacher}$	S_{scal}^{back}	Δt	a	ν^{res} Uniform distribution	ν^{tar} Uniform distribution	f^{out}	W^{in} Distribution	W^{back} Distribution	E^{class}
117	P	600	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.39%
118	P	550	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	id	[-1 1] U	[-1 1] U	32.68%
119	P	500	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	id	[-1 1] U	[-1 1] U	34.87%

120	P	400	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.26%
121	P	350	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	36.89%
122	P	300	0.95	[1.5]	[-0.2]	1.0	-0.2	[2]	[0.1]	-	-	-	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	36.82%
123	L	720	0.35	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	32.41%
124	L	720	0.2	[1.8]	[-0.2]	1.0	-0.2	[0.8]	[0.1]	-	[0.2]	[0.8]	[0]	[0]	<i>id</i>	[-1 1] U	[-1 1] U	34.86%

APPENDIX 1. Plain_ESN, leaky1_ESN,leaky_ESN,Twi_ESN error classification rate and the associated parameters setup.