Training Artificial Neural Networks to Pronounce Arabic Text

٩٨/١١/٥
٤/١
٤/١

by
Nizar Radi Mabroukeh

Supervisor

Dr. Khalil el Hindi

Co-Supervisor

Dr. Abdulraouf Al-Hallaq

[illegible]
عميد كلية الدراسات العليا

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Faculty of Graduate Studies
University of Jordan

June 1998

This thesis was successfully defended and approved on 27 / 6 / 1998

Examination Committee	Signature
Dr. Khalil el Hindi, Chairman **Asst. Prof. of Artificial Intelligence**	[illegible]
Dr. Abdulraouf Al-Hallaq, Member **Asst. Prof. of Interconnection Networks**	[illegible]
Dr. Yahia Halabi, Member **Prof. of Numerical Simulation**	[illegible]
Dr. Ahmad Sharieh, Member **Asst. Prof. of Parallel Processing**	[illegible]
Dr. Walid Salameh, Member **Asst. Prof. of Neural Networks**	[illegible]

To the memory of the great Arab poet.....

....Nizar Qabbani.

Acknowledgement

I am very grateful for the guidance and help that my supervisor Dr. Khalil el Hindi and Co-supervisor Dr. Abdulraouf Al-Hallaq provided.

I am in debt to Dr. Mohammad Al-Anani, Chairman of the Phonetics Research Center at the University of Jordan and all the persons working there for their help and enlightenment, they made me realize how great our language is.

I would also like to thank Dr. Walid Salameh at PSCUT, the staff members at the Computer Science Department at JU and my friends and colleagues in the graduate program. Also, thanks to Haitham Ibrahim for taking the time and putting the effort in reviewing and checking the flow of thoughts in this thesis, and Muhannad Tayyem for helping me learning C language.

Thank you my mother for making me the man I am, and my father for his belief and trust in me, my sisters, Lucy and Suzan for their help and for being there when I felt down.

Thank you Amjad Hudaib, Issa Qunbor, Aseel Al-Anani, Khalid Waleed, Ziad Al Masri and "Ansar Magdalena" Khoury.

The followings have also lent a helping hand: Dr. Martin Riedmiller (University of Karlsruhe, Germany) who helped me understand Rprop and provided me with the latest modifications, Lars Kindermann (University of Erlangen, Germany) who allowed me to test-run his commercial neural network F.A.S.T., Dr. Terry Sejnowski (Salt Lake Institute and UCSD, USA) for explaining more about how NETtalk learns, and finally Dr. Jeff Elman for providing me with information about Recurrent nets.

Contents

Dedication iii

Acknowledgement iv

List of Contents v

List of Appendices vii

List of Tables viii

List of Figures x

Abstract xi

Chapter 1: Introduction 1

1.1 Brief Introduction and History of Artificial Neural Networks 1

1.2 Anthropomorphism, The Biological Metaphor 5

1.2.1 Biological Background 5

1.2.2 Mapping Biological Neural Networks onto Artificial Neural Networks 7

1.3 Importance and Applications of Artificial Neural Networks 9

1.4 Structure of Thesis 12

Chapter 2: Background Information on Artificial Neural Networks 13

2.1 Outline and Architecture of Artificial Neural Networks 13

2.1.1 Input and Output Patterns 15

2.1.2 Connections 16

2.1.3 Processing Elements (PEs) 16

2.2 Learning 21

2.3 Error Backpropagation and Gradient Descent Supervised Learning 22

2.3.1 BP Algorithm 27

2.3.2 Problems and Pitfalls of Backpropagation (BP) 28

2.4 Resilient Backpropagation 31

2.4.1 Rprop Algorithm 34

Chapter 3: Literature Review 36

3.1 Research in Speech Recognition and Text Synthesis 36

3.2 A Neural Network that Learns to Pronounce English Text: NETtalk 42

3.2.1 NETtalk's Topology 42

3.2.2 Representation in NETtalk 43

3.2.3 Learning in NETtalk 45

3.2.4 NETtalk Performance 47

3.2.5 Conclusions about NETtalk 49

Chapter 4: A NETtalk-like Neural Network to Pronounce Arabic Text 51

4.1 Duplicating NETtalk 51

4.2 The Topology of NETtalk2 51

4.3 Representation of Input and Output in NETtalk2 53

4.4 How NETtalk2 works 53

4.5 Learning in NETtalk2 54

4.6 Training NETtalk2 on English Text 55

4.6.1 Duplicating Sejnowski and Rosenberg's Experiments 62

4.7 Training NETtalk2 on Arabic Text 63

4.7.1 Measuring NETtalk2's performance on Arabic text 77

4.7.2 Using Multiple Hidden Layers in NETtalk2 with Arabic Text.................. 82

4.7.3 Configuring NETtalk2 as a Recurrent Neural Network for Pronouncing Arabic Text.. 83

Chapter 5: Analysis of NETtalk2 Performance on Arabic Text................ 85

5.1 Analysis of the Input.. 85

5.2 Output Analysis and Thorough Investigation of NETtalk2......................... 87

5.2.1 Non-convergence in NETtalk2.. 94

5.3 Using Different Numbers of Input Groups in NETtalk2.............................. 100

Chapter 6: Conclusion and Future Work.. 103

6.1 Summary and Conclusion.. 103

6.2 Future Work.. 108

References.. 109

Appendices

Appendix A: Set of Phonemes used by NETtalk2.................................... 119

Appendix B: Format of the .wav Sound Files.. 126

Appendix C: Code for BP and Rprop algorithms used with NETtalk2............. 129

Appendix D: Training and Generalization Sets used with NETtalk2.............. 147

Abstract in Arabic... 162

List of Tables

Table 4.1: Training NETtalk2 on the English alphabet 56

Table 4.2: Generalization test on NETtalk2 after training on 40 words 57

Table 4.3: Training NETtalk2 using BP with different sizes of the input window. 59

Table 4.4: Generalization results on a small English text 60

Table 4.5: Generalization results of training NETtalk2 on a simple English story. 61

Table 4.6: Training NETtalk2 on the Arabic alphabet 65

Table 4.7: Generalization results 67

Table 4.8: Generalization results on the final extended text of Example A2 68

Table 4.9: Learning Rate vs. Convergence Speed 68

Table 4.10: Results of generalization tests on a small Arabic story 70

Table 4.11a: Generalization results of NETtalk2 on a children Arabic story using Rprop 71

Table 4.11b: Generalization results of NETtalk2 on a children Arabic story using BP 71

Table 4.12a: Generalization tests after training NETtalk2 on continuous text using size-7 window 73

Table 4.12b: Generalization tests after training NETtalk2 on continuous text using different window sizes 73

Table 4.13: Generalization tests on a continuous text 74

Table 4.14: Generalization test on a continuous text of 300 Arabic words 75

Table 4.15a: NETtalk2's performance on Arabic text using different input window sizes with BP 76

Table 4.15a: NETtalk2's performance on Arabic text using different input window sizes with Rprop 77

Table 4.16: Training NETtalk2 on the Arabic alphabet using BP and Rprop...... 81

Table 5.1: Identifying "n" 89

Table 5.2: Identifying "ع" 90

Table 5.3: Generalization tests when NETtalk2 was trained on example A7 with all consonant letters 98

Table 5.4: Generalization tests when NETtalk2 was trained on text with correct pronunciation 98

Table 5.5: Generalization tests on example A8 with all consonant letters 99

Table 5.6: Generalization tests on example A8 with correct pronunciation.......... 100

List of Figures

Figure 1.1: A juxtaposition of a neuron grown on a microprocessor................ 2

Figure 1.2: A brief history of neural networks.. 5

Figure 1.3: Cell body of a neuron.. 6

Figure 2.1: A typical neural network.. 13

Figure 2.2: Sample transfer functions.. 19

Figure 2.3: Gaussian (Bell-shaped) function... 20

Figure 2.4: Adjusting the weights by Gradient Descent.............................. 25

Figure 2.5: Problem of Gradient Descent.. 29

Figure 2.6 Pseudocode for error BackPropagation....................................... 30

Figure 2.7: pseudocode for Resilient backpropagation................................. 35

Figure 3.1: Schematic drawing of NETtalk architecture............................... 43

Figure 4.1 Diagram of NETtalk2 system.. 54

Figure 4.2: Training epochs vs. Window size... 59

Figure 4.3: Training epochs vs. Momentum... 66

Figure 4.4: Convergence speed versus Learning Rate.................................. 69

Figure 4.5: Jordan Recurrent Neural Network model.................................... 83

Abstract

Training Artificial Neural Networks to Pronounce Arabic text

by
Nizar Radi Mabroukeh

Supervisor

Asst. Prof. Khalil el Hindi

Co-Supervisor

Asst. Prof. Abdulraouf Al-Hallaq

Humans have always wondered if it is possible to build the Artificial Brain. Artificial Neural Networks have been known to us for more than half a century by now, and with the rising of the computing age, it made realizing these human brain models possible to an extent. Ever since the 80s, Artificial Neural Networks were used to mimic humanly processes in different ways. These models are significant in their ability to learn, memorize and adapt to changes on their own.

In this research we take a closer look into NETtalk, the neural network that pronounces English text aloud, we build a matching model (NETtalk2) and try to train it to pronounce Arabic text, using "Error Backpropagation" and "Resilient Backpropagation".

Results reveal that, a model like NETtalk is unable to pronounce Arabic text, as it fails either in learning or in generalization; due to reasons that are sole characteristics of Arabic language and grammar that cannot be ignored. We point out these reasons, discuss them thoroughly and present examples on each one. We also present examples on training NETtalk2 using two hidden layers in order to increase feature representation in the net. In this case generalization performance was improved by 8% using two hidden layers of 80 units each instead of one, reaching 72% successful generalization on the average.

When recurrent learning was experimented with (using the Jordan model), NETtalk2 showed good results and generalization performance improved by 12%, reaching 82% successful generalization on the average.

Chapter 1

Introduction

1.1 Brief Introduction and History of Artificial Neural Networks:

The term *neural network*, originally referred to a network of interconnected neurons in the human brain. The average human brain consists of 15×10^{10} neurons.
To try to understand the brain function, various models have been proposed. Probably the best known was the work of Hodgkin and Huxley on the modelling of the giant squid axon. They methodically collected empirical data, postulated the mechanism of ion transport channels, formulated mathematical and circuit models, and then developed the voltage clamped technique to validate their models. Certainly, modelling work done by many others, such as Von Bekesy's cochlear microphonics work, can be cited as contributing to the understanding of the brain function. For example, the discovery of special feature detecting cells in the visual cortex by Hubel and Wiesel has had a profound impact on the understanding of information processing in the visual system, and on the field of pattern recognition (Lau, 1992).
We are now dealing with the ultimate question: How does the brain work? Neurobiologists have taken the bottom-up approach by studying the stimulus-response characteristics of single neurons and networks of neurons. On the other hand, psychologists have taken the top-down approach by studying brain function from the cognitive and behavioural level. They are gradually and incrementally getting a better idea of how the brain works, both at

the single neuron level and at the behavioural level. However, it may take another fifty years before we have a solid, complete microscopic, intermediate, and macroscopic view of how the brain works. We seek solutions to problems that are difficult with today's digital computing technology; problems that are easily solved by people and animals. We try to build more brainlike computers out of neuronlike parts.

Figure 1.1 shows a neuron (the basic processing element in the human brain) compared with a Motorola microprocessor.

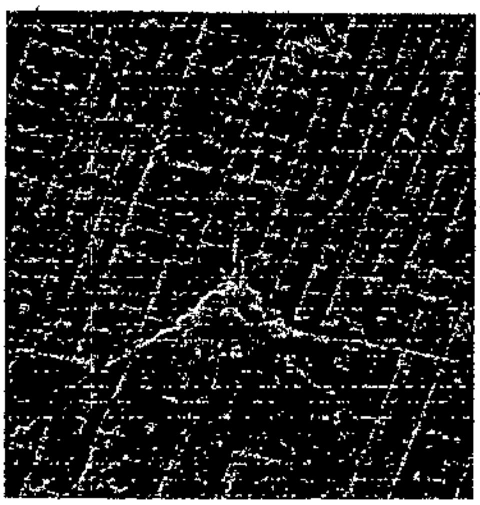
Figure 1.1: This micrograph shows an unusual juxtaposition of the components from which living computers and human-made computers are made. A single nerve cell (neuron) was taken out of the nervous system and allowed to grow on the surface of a Motorola 68000 microprocessor (Campbell ,1993).

It was this view that led to the development of many of the earlier models of neurons and artificial neural networks. McCullogh and Pitts in the 1940s (McCulloch *et al*, 1943) showed that the neuron can be modelled as a simple threshold device to perform logic functions. In the same time frame, relationships among engineering principles, feedback, and brain function were expounded by Wiener as the principle of Cybernetics. By the late 1950s and early 1960s, neuron models were further refined into Rosenblatt's Perceptron (Rosenblatt, 1959), Widrow and Hoff's ADALINE (ADAptive LINear Element) (Widrow *et al*, 1960), and Steinbuch's Learning Matrix. The Perceptron received considerable excitement when it was first introduced because of its conceptual simplicity. However, that excitement was short lived when Minsky and Papert proved mathematically that the Perceptron cannot be used for complex logic functions (Minsky *et al*, 1969); because it cannot work with problems that are not linearly separable, like the famous XOR problem. On the other hand, the fate of Adaline was quite different. The Adaline is a weighted sum of the inputs, together with a Least Mean Square (LMS) algorithm to adjust the weights to minimize the difference between the output and the desired signal. Because of its linear and adaptive nature, this technique has developed into a powerful tool for adaptive signal processing, which is used in equalization, echo and noise cancellation, adaptive beam forming, and adaptive control. The reason is primarily due to rigorous, mathematical foundation of LMS algorithm. Because of that, it has stood the test of time.

Today the term artificial neural networks has come to mean any computing architecture that consists of massively parallel interconnections of simple "neural" processors; that is why it is also called PDP (Parallel Distributed Processing).

The present impetus in artificial neural networks research is due in part to the paper John Hopfield published in 1982 in the *Proceedings of the National Academy of Sciences* (Hopfield, 1982). Hopfield gave a new and powerful kickstart in artificial neural networks research and development. In his paper, he presented a model of neural computation that is based on the complete interaction of neurons. The model consisted of a set of first order (nonlinear) differential equations that minimize a certain "energy" function. He agreed that there are emergent computational capabilities at the network level that are nonexistent at the single neuron level. This kind of artificial neural network is now known as Hopfield Net.

Certainly Hopfield was not the first to recognize the neuron's spatial and temporal integration properties, which were known in the very early days of neurophysiology. Furthermore, the idea that neurons organize themselves to perform the necessary functions was promoted by a number of researchers. During the 1970s, when no one else was working on artificial neural networks, Steven Grossberg at Boston University and Teuvo Kohonen at Helsinki University were making significant contributions. Grossberg, together with Gail Carpenter, have developed an artificial neural network architecture they call *Adaptive Resonance Theory* (ART), based on the ideas that the brain spontaneously organizes itself into recognition codes (Grossberg *et al*, 1992). The dynamics of the network were also modelled by first-order differential equations. Meanwhile, Kohonen was developing his ideas on self-organizing maps, based on the idea that neurons organize themselves to tune to various and specific algorithms (Kohonen, 1990). In the early 1970s, Paul Werbos discovered the mathematical principles of the backpropagation algorithm while studying problems in the social sciences. In the mid 1980s, David Rumelhart and his colleagues published their landmark books on Parallel Distributed Processing (PDP), which established the backpropagation algorithm and feedforward layered networks as the major paradigm of the field (Rumelhart *et al*, 1986). This and earlier works have finally galvanized a large segment of the scientific community into thinking in terms of collective neural computation rather than single neurons.

Figure 1.2 show a brief history of ANN.

Present	Late 80s to now	Interest explodes with conferences, articles, simulations, new companies and government funded research.
Late Infancy	1982	Hopfield at National Academy of Sciences.
		Some research continues.
Stunted Growth	1969	Minsky & Papert's critique, *Perceptrons*.
		Excessive hype.
Early Infancy	late 50s, 60s	Research efforts expand.
		AI and Neural Computing fields launched.
Birth	1956	Darmouth Summer Research Project.
Gestation	1950s	Age of computer simulation.
	1949	Hebb, *The Organization of Behaviour*.
	1943	McCulloch & Pitts paper on neurons.
	1936	Turing uses brain as computing paradigm.
Conception	1890	James, *Psychology (Briefer Course)*.

Figure 1.2: A brief history of neural networks (Nelson *et al*, 1993).

1.2. Anthropomorphism, The Biological Metaphor:

1.2.1 Biological Background:

Artificial Neural Networks are based on the biological nervous system of animals and are made to mimic human neural processes.

Neurons in animals are cells specialized for transmitting signals from one location in the human body to another, and are the functional units of the nervous system.

Neurons occur in a variety of sizes and shapes; nevertheless, most of them contain four parts as shown in Figure 1.3: (1) the cell body; (2) the dendrites; (3) the axon, and (4) the axon terminals.

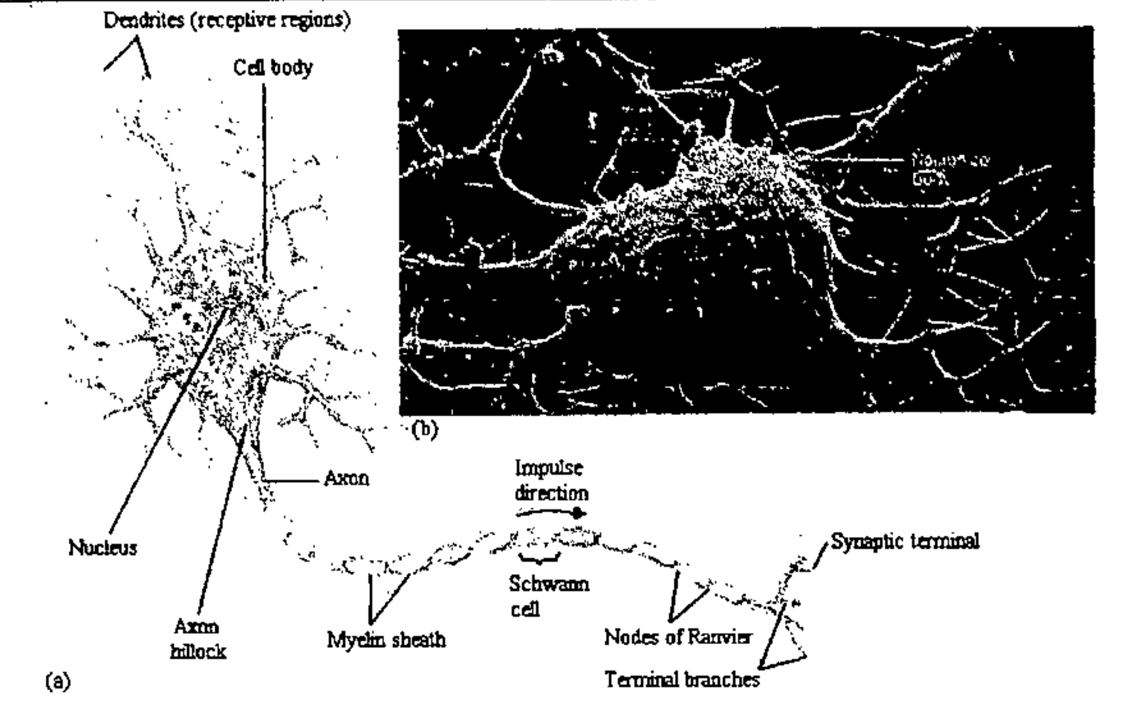

Figure 1.3: **(a)** The cell body of the neuron. **(b)** Scanning electron micrograph of a neuron (Campbell,1993).

Dendrites convey signals toward the cell body and are short, numerous and extensively branched (indeed, the name is derived from the Greek *dendron*, "tree") where the cell is most likely stimulated. Whereas axons conduct messages away from the cell body. The portion of the axon closest to the cell body plus the part of the cell body where the axon is joined is known as the *initial segment (axon hillock)*, where electric signals are generated and then propagated away from the cell body along the axon to other near cells through axon terminals *(Telodendria)*, that ends with *synaptic knobs* which release chemicals called *neurotransmitters*, that help to relay nervous signals to other cells. These chemical messages defuse across the *synapse*, a narrow gap between the synaptic knob and the dendrites of another neuron (Campbell, 1993) (Vander *et al*, 1994).

Electric signals are generated and conveyed through the axon to other neighbour cells only if the result input of signals at the dendrites exceeds a certain threshold. In (Vander *et al*, 1994), threshold is defined as: "Membrane potential to which excitable membrane must be depolarized to initiate an action."

1.2.2 Mapping Biological Neural Networks onto Artificial Neural Networks:

Starting from the biological model in the previous section, we can develop a metaphor, despite the fact that anthropomorphism can lead to misunderstanding when the metaphor is carried too far. This explains why some researchers are deliberately calling their efforts "Connectionism" (Nelson *et al*, 1993).

Now let us try to map biological neural networks over the model of artificial neural networks.

We call our neuron a *Processing Element (PE)*, or a *node*. These artificial neurons bear only a modest resemblance to the real things, since it is easier to observe and measure electrical activity than it is to understand the chemical properties. We know that there are at least 150 processes performed by neurons in the human brain. PEs model approximately three of these processes as follows (Nelson *et al*, 1993):

- Evaluate the input signals, determining the strength of each one.
- Calculate the total for the combined input signals and compare that total to some threshold level.
- Determine what the output should be.

In the human brain, signals come into the synapses. These are the inputs, they are "weighted", that is, some signals are stronger than others. Some signals excite (are positive and have positive weights), and others inhibit (are negative and have negative weights). The effects of all weighted inputs are summed, which also happens in artificial neural networks. If the sum is equal to or greater than the threshold for the neuron, then the neuron fires (gives output). This is an "all-or-nothing" situation; either a neuron fires or does not fire.

The ease of transmission of signals is altered by activity in the nervous system. Synapses are susceptible to fatigue, oxygen deficiency, and agents such as anaesthetics. These events create a resistance to the passage of impulses. Other events may increase the rate of firing. This ability to adjust signals is a mechanism for learning. Threshold functions in artificial neural networks integrate the energy of incoming signals over space and time. In artificial neural networks synapses are represented as weights on the connections among PEs, events that affect the transfer of signals between synapses are represented as *bias* inputs to PEs. This bias term or *forcing term* could also be a forgetting term, when a system needs to unlearn something.

As mentioned, weights are like the varying synaptic strengths, as some inputs are more important than others in the way they combine to produce an impulse. Weights are **adaptive** coefficients within the network that determine the intensity of the input signal. Think of them as a measure of the connection strength (see Figure 2.1). The initial weight of a PE can be modified in response to various inputs and according to the network's own rules for modifications. It can also be drawn from a probability density function (Salameh, 1996).

1.3 Importance and Applications of Artificial Neural Networks:

Artificial Neural Networks (ANNs) occur in many fields and have various applications.

Some of the operations that artificial neural networks perform include (Simpson, 1992):

- *Classification* - an input pattern is passed to the network, and the network produces a representative class as output.
- *Pattern matching* (Heterassociative)- an input pattern is passed to the network, and the network produces the corresponding output pattern.
- *Pattern completion* (Autoassociative)- an incomplete pattern is passed to the network, and the network produces an output pattern that has the missing portions of the input pattern filled in.
- *Noise removal* - a noise-corrupted input pattern is presented to the network, and the network removes some (or all) of the noise and produces a cleaner version of the input pattern as output.
- *Optimization* - an input pattern representing the initial values for a specific optimization problem is presented to the network, and the network produces a set of variables that represents an optimal solution to the problem.
- *Control* - an input pattern represents the current state of a controller and the desired response for the controller, and the output is the proper command sequence that will create the desired response.

These operations fall into any of the following three primary situations where neural networks are advantageous (Simpson, 1992):

1. Situations where only a few decisions are required from a massive amount of data (e.g., speech and image processing).
2. Situations where nonlinear mappings must be automatically acquired (e.g., loan evaluations and robotic control).
3. Situations where near-optimal solution to a combinatorial optimization problem is required quickly (e.g., airline scheduling and telecommunication message routing).

Some commercial products relevant to our study that use ANNs are (Nelson *et al*, 1993):

1. Intel's word recognizer: Intel™ built a system that capitalizes on the expressiveness of human speech. By limiting the system vocabulary to a single speaker at a time and limiting the vocabulary to around a hundred isolated words or phrases, the speech recognition system provides better than 99% accuracy for the speaker who trains the system. This voice-controlled data entry system has been used in various manufacturing applications since 1983.
2. NestorWriter: A handwriting recognizer that runs on an IBM PC AT. Input is in the form of handwriting on a digitized pad. After being trained on a set of typical handwriting samples, NestorWriter can interpret handwriting which it has not seen previously, and this can be accomplished in spite of changes in scale, shifts in position, distortions, and idiosyncrasies in style.

As any engineering technique, artificial neural networks have their own set of advantages and disadvantages. The natural question is: What can artificial neural networks do that traditional signal processing techniques cannot do? Certainly speed of computation is a factor. In traditional single processor Von Neumann computers, the speed is limited by the propagation delay of the transistors. Artificial neural networks, on the other hand, because of their massively parallel nature, can perform computations at a much higher rate. Because of their adaptive nature, artificial neural networks can adapt to changes in the data and learn the characteristics of input signals. Furthermore, because of their nonlinear nature, ANNs can perform functional approximation and signal-filtering operations, which are beyond optimal linear techniques. ANNs can be used in pattern classification by defining nonlinear regions in the feature space.

Artificial neural networks require a different set of skills from that required by conventional programming. Information is not stored in a single memory location, but is distributed throughout the system. This feature provides robustness for artificial neural networks, you could lose a percentage of the PEs and still not lose the information stored there. The power is in the collective computational abilities. The result is a new information-processing paradigm, namely, Associative Memory. (Nelson *et al*, 1993).

Still though, artificial neural networks have their own limitations. The nonlinear sigmoidal functions used in layered networks cause multiple minima to appear during learning, and one is never sure whether the system has reached its global minimum. Stochastic techniques such as "simulated annealing," can help the situation but often require excessive computation time. As in adaptive signal processing, there is always trade-off between the speed and the stability of convergence. Digital computer simulations of neural nets are still too slow for practical use in large-scale problems (Lau, 1992).

1.4 Structure of the Thesis:

Scientific research in ANNs is divided into two fields. Some scientists study neural networks to mimic the human brain, while others try to model the human brain to be able to understand more how it works, to help people who suffer from brain related illness and disease. In either case, we believe that before indulging into Artificial Neural Networks, one has to understand the biology behind it.

Now, after a biological model was presented, we move on to talk about the structure of a typical neural network and how it learns and adapts to changes in Chapter 2. We will also present the new Riedmiller's learning algrothim called 'Resilient Backpropagatoin.'

In Chapter 3, we will review the literature in the fields of speech recognition and text synthesis, using Expert Systems and ANNs. We will also introduce NETtalk, the famous text synthesis ANN by Sejnowski and Rosenberg. The core subject will be discussed in Chapter 4, in which a NETtalk-like ANN will be constructed to pronounce Arabic text, tested, performance-evaluated and discussed. Then its results will be analized and investigated further in Chapter 5. Finally, results will be summerized and recommendations put forward in Chapter 6.

Chapter 2

Background Information on Artificial Neural Networks

2.1 Outline and Architecture of Artificial Neural Networks:

Artificial neural networks (ANNs) can be thought of as a "Black Box", devices that accept input and produce output. Inside this black box, an ANN consists of processing elements and weighted connections. Figure 2.1 illustrates a typical neural network.

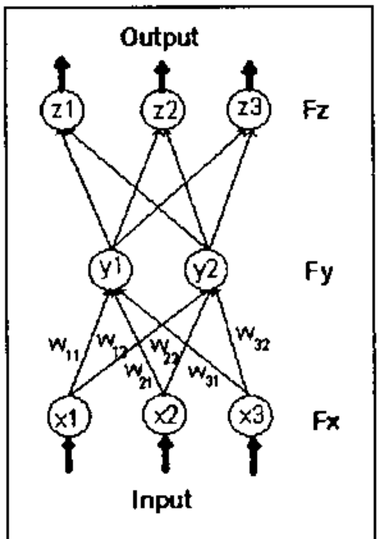

Figure 2.1: A typical neural network (Simpson, 1992).

Each layer in a neural network consists of a collection of *Processing Elements PEs*. Each PE collects the values from all of its input connections, performs a predefined mathematical operation, and produces a single output value. The neural network in Figure 2.1 has three layers: F_x, which consists of the PEs $\{x_1, x_2, x_3\}$; F_y, which consists of the PEs $\{y_1, y_2\}$; and F_z, which consists of the PEs $\{z_1, z_2, z_3\}$, from bottom to top, respectively. The PEs are connected with weighted connection from every F_x PE to every F_y PE, and there is a weighted connection from every F_y PE to every F_z PE not shown in the figure for clarity. Each weighted connection (often synonymously referred to as either a connection or a weight) acts as both a label and a value. As an example, in Figure 2.1 the connection from the PE x_1 to the PE y_2 is the connection weight w_{12} (from x_1 to y_2). The connection weights store information. The values of the connection weights is often determined by a neural network learning procedure, by adjustment of these weights, and thus the neural network is able to learn. On the other hand, by performing the update operations for each of the PEs, the neural network is able to recall information (Simpson, 1992).

There are several important features illustrated by the neural network shown in Figure 2.1 that apply to all neural networks:

- Each PE acts independently of all others - each PE's output relies only on its constantly available inputs from the abutting connections.
- Each PE relies only on local information - the information that is provided by the adjoining connections is all a PE needs to process; it does not need to know the state of any of the other PEs where it does not have an explicit connection.
- The large number of connections provides a large amount of redundancy and facilitates a distributed representation.

The first two features allow neural networks to operate efficiently in parallel. The last feature provides neural networks with inherent fault-tolerance and generalization qualities that are very difficult to obtain from typical computing systems. In addition to these features, through proper arrangement of neural networks, introduction of nonlinearity in the processing elements, and use of the appropriate learning rules, neural networks are able to learn arbitrary nonlinear mappings. This is a powerful attribute.

Neural networks consist of three principle elements (Simpson, 1992):

1. *Topology:* How a neural network is organized into layers and how those layers are connected.

2. *Learning and memory:* How information is stored in the network.

3. *Recall:* How stored information is retrieved from the network.

An artificial neural network is formed mainly of three components, described in the following three sub-sections.

2.1.1 Input and Output Patterns:

Some artificial neural networks require only single patterns of data, and others require pattern pairs. The dimensionality of the input pattern is not necessarily the same as the output pattern. When a network works only with single patterns it is an autoassociative network. When it works with pattern pairs, it is heteroassociative (Simpson, 1992). In this case, the pattern pair is called "the training vector." One of the patterns is used for input and the other is used to check the correctness of the output for error computation and backpropagation in the learning phase.

2.1.2 Connections:

A neural network is equivalent to a digraph, with edges (connections) between nodes (PEs) that allow information to flow only in one direction. Information flows through the digraph along the edges and is collected at the nodes. Neural Networks include a weight with each connection that modulates the amount of output signal passed from one PE down the connection to the adjacent PE. So, a connection both defines the information flow through the network and modulates the amount of information passing to PEs.

Connection weights that are positive-valued are *excitatory* connections, those with negative values are *inhibitory* connections. A connection weight that has a zero value is the same as not having a connection present (Simpson, 1990).

2.1.3 Processing Elements (PEs):

The PE is the portion of the neural network where all the computing is performed. Each PE collects the information that has been sent down its abutting connections and produces a single output value. There are two important qualities that a PE must possess (Simpson, 1990):

1. PEs require only local information. All the information necessary for a PE to produce an output value is present at the inputs and resides within the PE. No other information about other values in the network is required.

2. PEs produce only one output value. This single output value is propagated down the connections from the emitting PE to other receiving PEs, or it will serve as an output from the network when the PEs are at the output layer.

A PE is made up of different components:

1. Input Weights: A neuron (PE) usually receives many simultaneous inputs. Each input has its own relative weight which gives the input the impact that it needs on the processing element's summation function. These weights perform the same job as the varying synaptic strengths of biological neurons. In both cases, some inputs are made more important than others so that they have a greater effect on the processing element as they combine to produce a neural response.

Weights are adaptive coefficients within the network that determine the intensity of the input signal as registered by the artificial neuron. You can consider them as a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or through its learning rules.

2. Summation Function: The first step in a processing element's operation is to compute the weighted sum of all of the inputs. Mathematically, the inputs and the corresponding weights are vectors which can be represented as $(i_1, i_2 \ldots i_n)$ and $(w_1, w_2 \ldots w_m)$. The total input signal is the dot, or inner, product of these two vectors. This simplistic summation function is found by multiplying each component of the i vector by the corresponding component of the w vector and then adding up all the products (Section 2.1). The result is a single number, not a multi-element vector.

The summation function can be more complex than just the simple input and weighted sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function. In addition to a simple product summing, the summation function can select the minimum, maximum, majority, product, or several normalizing algorithms.

3. <u>Transfer Function</u>: The result of the summation function, is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal, if it is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant.

The threshold, or transfer function, is generally non-linear. Linear (straight-line) functions are limited because the output is simply proportional to the input. Linear functions are not very useful. That was part of the problem in the earliest network models as noted in Minsky and Papert's book *Perceptrons* (Minsky *et al*, 1969).

The transfer function could be something as simple as depending upon whether the result of the summation function is positive or negative. The network could output zero, one and minus one, or other numeric combinations. The transfer function would then be a "hard limiter" or step function. See Figure 2.2 for a sample of transfer functions.

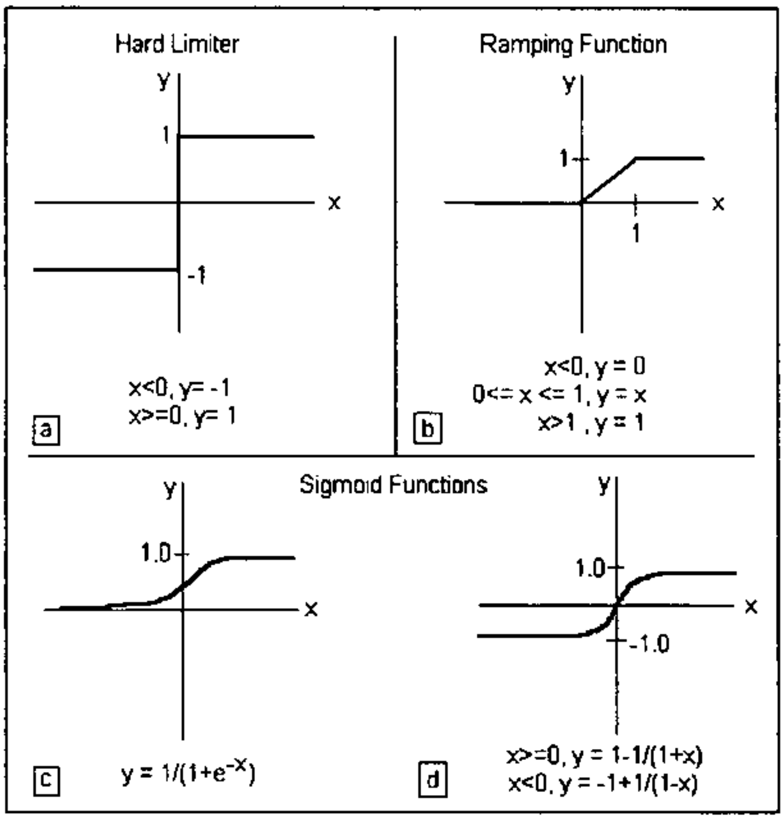

Figure 2.2: Sample transfer functions (Nelson *et al*, 1993).

Another type of transfer functions, the threshold or ramping function, could mirror the input within a given range and still act as a hard limiter outside that range. It is a linear function that has been clipped to minimum and maximum values, making it non-linear. Yet another option would be a sigmoid or S-shaped curve. That curve approaches a minimum and maximum value at the asymptotes. It is common for this curve to be called a sigmoid when it ranges between 0 and 1, and a hyperbolic tangent (tanh) when it ranges between -1 and 1. Mathematically, the exciting feature of these curves is that both the function and its derivative are continuous. This option works fairly well and is often the transfer function of choice.

Prior to applying the transfer function, uniformly distributed random noise may be added. The source and amount of this noise is determined by the learning mode of a given network paradigm. This noise is normally referred to as "temperature" of the artificial neurons. The name, temperature, is derived from the physical phenomenon that as people become too hot or cold their ability to think is affected. This process is simulated electronically by adding noise. By adding different levels of noise to the summation result, more brain-like transfer functions are realized. To more closely mimic nature's characteristics, some experimenters are using a gaussian noise source. Gaussian noise is similar to uniformly distributed noise except that the distribution of random numbers within the temperature range is along a bell curve (Figure 2.3).

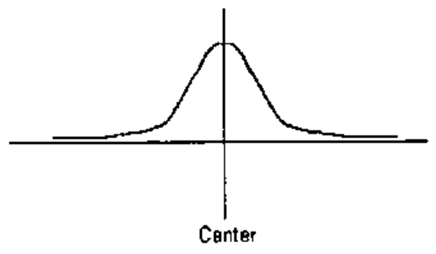

Figure 2.3 Gaussian (Bell shaped) Function.

4. Scaling and Limiting: After the processing element's transfer function, the result *can* pass through additional processes to perform scaling and limiting. This scaling simply multiplies a scale factor times the transfer value, and then adds an offset. Limiting is the mechanism which insures that the scaled result does not exceed an upper or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

This type of scaling and limiting is mainly used in topologies to test biological neuron models, such as James Anderson's brain-state-in-a-box (Nelson *et al*, 1993).

5. PE's Output: Each processing element is allowed one output signal which it may output to hundreds of other neurons. This is just like the biological neuron, where there are many inputs and only one output action. Normally, the output is directly equivalent to the transfer function's result.

Those previous subsections are the three main components that constitute a NN Topology. In addition to the fact that PEs can be arranged into layers, the model described here is missing a vital component, *learning*.

2.2 Learning:

Another important feature of a neural network (NN) is the *Learning Function,* which is a property of the NN as a whole, not of a single PE.

The purpose of the learning function is to modify the variable connection weights on the inputs of each processing element according to some algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaptation function, as well as the learning mode. There are two types of learning: supervised and unsupervised. Supervised learning requires a teacher. The teacher may be a training set of data or an observer who grades the performance of the network results. Either way, having a teacher is learning by reinforcement. When there is no external teacher, the system must organize itself by some internal criteria designed into the network. This is unsupervised learning (learning by doing).

Part of the learning process is the error backpropagation. In most artificial neural networks the difference between the current output and the desired output is calculated. This raw error is then transformed by the learning process to match a particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The PE's error is then typically propagated into the learning function of another PE in the previous layer.

This back-propagated value can be either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied against each of the incoming connection weights to modify them before the next learning cycle, in order to produce an output closer to the one desired each time.

2.3 Error Backpropagation and Gradient Descent in Supervised Learning:

The most widely used learning algorithm with supervised learning, is the Error Backpropagation algorithm (BP) (Rumelhart *et al*, 1986). It appeared as a solution to multi-layered networks's learning after Minsky and Papert have proven the inefficiency of single-layered networks to solve the XOR problem, in their famous book *Perceptrons* (Minsky *et al*, 1969).

A unit (PE), call it i, in a backpropagation (BP) network computes its activation s_i (in equation (2)) with respect to its incoming excitation, which is, the net input net_i (in equation (1)).

$$net_i = \sum_{j \in pred(i)} s_j w_{ij} - \theta_i \qquad (1)$$

where *pred(i)* denotes the predecessors of unit i , w_{ij} denotes the connection weight from unit j to unit, s_j is the output of the predecessor unit, and θ_i is the unit's bias value. For the sake of a homogenous representation, θ_i is usually substituted by a weight to a *"bias unit"*, which is treated as any other neuron but with a constant output of 1 (Riedmiller, 1994 a).

BP requires a slightly different activation function from the perceptron. The neuron produces a real value between 0 and 1 as output (Rich *et al*, 1991), which is computed by passing the net input through a non-linear activation function, usually the sigmoid logistic (S-shaped) function as in equation (2).

$$s_i = f(net_i) = \frac{1}{1 + e^{-net_i}} \qquad (2)$$

Notice that if the sum is 0, the output is 0.5 (in contrast to the perceptron, where it must be either 0 or 1). As the sum gets larger the output approaches 1. As the sum gets smaller, on the other hand, the output approaches 0.

A nice property of this function is being continuous and having an easily computable derivative, shown in equation (3).

$$\frac{\partial s_i}{\partial net_i} = f'(net_i) = s_i * (1 - s_i) \qquad (3)$$

Like a perceptron, a BP network typically starts out with a random set of weights. The network adjusts its weights each time it sees an input-output pair. Each pair requires two stages: a forward pass and a backward pass. The forward pass involves presenting a sample input to the network and letting activations flow until they reach the output layer. During the backward pass, the network's actual output (from the forward pass) is compared with the target (desired) output and error estimates are computed for the output units. The weights connected to the output units can be adjusted in order to reduce those errors. Then the error estimates of the output units can be used to derive error estimates for the units in the hidden layers. Finally, errors are propagated back to the connections stemming from the input units (Rich *et al*, 1991). This process is repeated until the network "learns" to give the desired output. This is supervised learning.

In technical terms, consider the pattern set P (a pattern set is a set of pattern pairs, containing input vectors given to the network for learning and desired output vectors, respectively). Each pattern pair p of the pattern set consists of an input activation vector x^p and its target activation vector t^p. After training the weights, when an input activation x^p is presented, the resulting output vector s^p of the net should equal the target vector t^p (or has a small error margin ε, as desired). The distance between the target and actual output vector, in other words the fitness of the weights, is measured by the energy or cost function E (Riedmiller, 1994 a)(Reynolds *et al*, 1995) as in equation (4).

$$E_{p\in P} = \frac{1}{2}\sum_k (t_k^p - s_k^p)^2 \qquad \forall k: k = 1, \cdots, n \qquad (4)$$

Where n is the number of units in the output layer. Fulfilling the learning goal now is equivalent to finding a global minimum of E. For this, the weights in the network are changed step by step by adding or subtracting a difference *Δw(t)* to each and everyone, and thus are changed along a search direction *d(t)* (Figure 2.4), driving the weights in the direction of the estimated minimum:

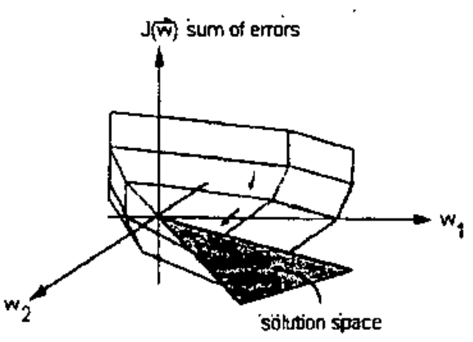

Figure 2.4: Adjusting the weights by Gradient Descent, minimizing $J(\vec{w})$ (Rich *et al*, 1991).

$$\Delta w(t) = \eta * d(t)$$
$$w(t+1) = w(t) + \Delta w(t)$$

Where the learning parameter η scales the size of the weight-step, called the *Learning Rate*. To determine the search direction *d(t)*, first order derivative information, the gradient $\nabla E = \frac{\partial E}{\partial w}$ is commonly used (Riedmiller, 1994 a). The gradient is a vector that tells us the direction to move in the weight space in order to reduce the error (Figure 2.4). To find a solution space (a solution weight vector), we simply change the weights in the direction of the gradient (this technique is known as "gradient descent" (Rich *et al*, 1991)). Backpropagation performs successive computations of ∇E by propagating the error back from the output layer towards the input layer.

The basic idea, used to compute the partial derivative $\frac{\partial E}{\partial w_{ij}}$ for each weight in the network, is to repeatedly apply the chain rule in equation (5).

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial s_i}\frac{\partial s_i}{\partial w_{ij}} \qquad (5)$$

where

$$\frac{\partial s_i}{\partial w_{ij}} = \frac{\partial s_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ij}} = f'(net_i)s_j \qquad (6)$$

To compute $\frac{\partial E}{\partial s_i}$, or the influence of the output s_i of the unit i on the global error E, the following two cases are distinguished:

- If i is an output unit, then

$$\frac{\partial E}{\partial s_i} = \frac{1}{2}\frac{\partial (t_i - s_i)^2}{\partial s_i} = -(t_i - s_i) \qquad (7)$$

derived from equation (4)

- If i is not an output unit (a hidden unit), then the computation of $\frac{\partial E}{\partial s_i}$ is a little more complicated. Again, the chain rule is applied:

$$\begin{aligned}\frac{\partial E}{\partial s_i} &= \sum_{k\in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial s_i}\\ &= \sum_{k\in succ(i)} \frac{\partial E}{\partial s_k}\frac{\partial s_k}{\partial net_k}\frac{\partial net_k}{\partial s_i} \qquad (8)\\ &= \sum_{k\in succ(i)} \frac{\partial E}{\partial s_k} f'(net_k)w_{ki}\end{aligned}$$

Where *succ(i)* denotes the set of all units *k* in successive layers (successive means closer to the output layer) to which unit *i* has a non-zero weighted connection w_{ki}.

Equation (8) assumes knowledge of the values $\frac{\partial E}{\partial s_k}$ for the units in successive layers to which unit *i* is connected. This can be provided by starting the computation (7) at the output layer and then successively computing the derivatives for units in preceding layers, applying (8). This way, the gradient information is successively moved from the output layer back towards the input layer. Hence the name 'backpropagation algorithm' (Riedmiller, 1994 a).

2.3.1 BP Algorithm:

From the above computation, one can write a pseudo-code representation of the BP algorithm. Assume the following:

We have a neural net with three layers:

n, input layer with A PEs,

h, hidden layer with B PEs,

and *o*, output layer with C PEs.

A function, *random*(a,b), generates floating point random numbers between a and b.

The NN described in this algorithm (Figure 2.6), has two weight layers:

w1, between *n* and *h*,

and w2, between *h* and *o*.

This algorithm is also tuned to increase the learning speed. The speed of learning in BP is increased by modifying the weight update steps, and adding a Momentum term α (Rich *et al*, 1991), as in Figure 2.6.

2.3.2 Problems and Pitfalls of BackPropagation (BP):

A BP network may slide down the error surface into a set of weights that does not solve the problem it is being trained on. If that set of weights is at a local minimum, the network will never reach the optimal set of weights. This requires re-running the algorithm with new random initial weights and might require also playing around with the learning rate. Although BP is simple, it is often a difficult task and requires a lot of experimenting to choose a good learning rate. A good choice depends on the shape of the error function, which obviously changes with the learning task itself. A small learning rate will result in long convergence time on a flat error function, whereas a large learning rate will possibly lead to oscillations, preventing the error to fall below a certain value, and be stuck in a local minimum. Although the algorithm can be guaranteed for convergence to a local minimum under certain circumstances, there is no guarantee that it finds a global minimum of the error function.

Another problem of gradient descent is the influence of the partial derivative on the size of the weight step. If the error function is shallow, the derivative is quite small, resulting in a small weight step. On other hand, large derivatives lead to large weight steps, possibly taking the algorithm to a completely different region of the weight space (see Figure 2.5), especially when the error function has steep ravines (Riedmiller, 1994 a).

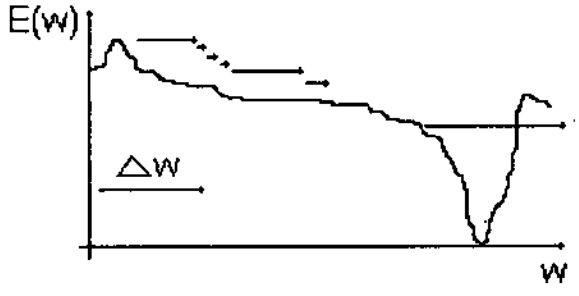

Figure 2.5: Problem of gradient descent: The weight-step is dependent on both the learning parameter and the size of the partial derivative $\frac{\partial E}{\partial w_{ij}}$ (Riedmiller, 1994 a)

One of the solutions was the idea of a momentum term presented in section 2.3.1 in the BP algorithm. It was introduced earlier in the literature to make learning more stable. This parameter α scales the influence of the previous weight step on the current one. Usually, when using gradient descent with momentum, the learning rate should be decreased to avoid unstable learning (Riedmiller, 1994 a).

$w1_{ij} = random(-0.1,0.1) \quad \forall i,j: i = 0,\cdots,A \quad ,j = 1,\cdots,B$

$\Delta w1_{ij}(t) = random(-0.1,0.1) \quad \forall i,j: i = 0,\cdots,A \quad ,j = 1,\cdots,B \quad at \quad t = 0$

$w2_{ij} = random(-0.1,0.1) \quad \forall i,j: i = 0,\cdots,B \quad ,j = 1,\cdots,C$

$\Delta w2_{ij}(t) = random(-0.1,0.1) \quad \forall i,j: i = 0,\cdots,B \quad ,j = 1,\cdots,C \quad at \quad t = 0$

$\left.\begin{array}{l} x_0 = 1.0 \\ h_0 = 1.0 \end{array}\right\}$ These are the thresholding (bias) (section 2.3) units, they have fixed output, 1.

η=0.35 Optimal value for the learning rate (Rich *et al*, 1991).
α=0.9 Optimal value for the momentum (Rich *et al*, 1991).

Suppose the training vector contains training pairs (x_i, y_i) where x_i is the input vector and y_i is the target output vector.

```
Repeat
    Repeat
        Choose training pair (x_i, y_i)
        $\forall j: \quad h_j = \frac{1}{1 + e^{-\sum_{i=0}^{A} w1_{ij} \cdot x_i}} \quad ,j = 1,\cdots,B$   where $w1_{0j}$ is the weight for $x_0$
        $\forall j: \quad o_j = \frac{1}{1 + e^{-\sum_{i=0}^{B} w2_{ij} \cdot h_i}} \quad ,j = 1,\cdots,C$   where $w2_{0j}$ is the weight for $h_0$
        {compute the error at units in the output layer}
        $\forall j: \quad \delta 2_j = o_j(1 - o_j)(y_j - o_j) \quad ,j = 1,\cdots,C$
        {compute error at units in the hidden layer}
        $\forall j: \quad \delta 1_j = h_j(1 - h_j) \sum_{i=1}^{C} \delta 2_i \cdot w2_{ji} \quad ,j = 1,\cdots,B$
        {now adjust weights}
        $\forall i,j: \quad \Delta w2_{ij}(t+1) = \eta . \delta 2_j . h_i + \alpha \Delta w2_{ij}(t) \quad ,i = 0,\cdots,B \quad ,j = 1,\cdots,C$
            $w2_{ij} = w2_{ij} + \Delta w2_{ij}(t+1)$
        $\forall i,j: \quad \Delta w1_{ij}(t+1) = \eta . \delta 1_j . x_i + \alpha \Delta w1_{ij}(t) \quad ,i = 0,\cdots,A \quad ,j = 1,\cdots,B$
            $w1_{ij} = w1_{ij} + \Delta w1_{ij}(t+1)$
        increment t
    Until all training pairs have been presented
Until $\forall i: \ (y_i - o_i) \le \varepsilon$, where ε is a small error margin
```

Figure 2.6: A neural network learning algorithm using error backpropagation.

2.4 Resilient Backpropagation:

To solve the problems of BP many algorithms were devised which are, in fact, variants of BP, like the Delta-Bar-Delta algorithm (Jacobs, 1988), Quickprop (Fahlman, 1988). Later, we used a very efficient and faster algorithm than BP, it is called "Resilient backpropagation" or Rprop, devised by Martin Riedmiller (Riedmiller *et al*,1993). It also passed through many variations by the author himself, the latest (to our knowledge) being in (Riedmiller, 1994 b).

Rprop is a local adaptive learning scheme (local adaptation strategies are based on weight-specific information, only, as the temporal behavior of the partial derivative of this weight. The local approach is more closely related to the neural network concept of distributed processing in which computations can be made in parallel) performing supervised batch learning (or learning by epoch). Though Rprop is also based on gradient descent, its basic principle is to eliminate the harmful influence of the <u>size</u> of the partial derivative on the weight step, *only* the <u>sign</u> of the derivative is considered to indicate the direction of the weight update. The algorithm is still that of gradient descent, *same as BP, but the weight update procedure is different; it depends on the derivative sign instead of its value.*

The size of the weight change is exclusively determined by a weight-specific, so-called *"update-value"* $\Delta_{ij}^{(t)}$:

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \quad if \quad \dfrac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & , \quad if \quad \dfrac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & , \quad else \end{cases} \qquad (9)$$

Where $\frac{\partial E}{\partial w_{ij}}^{(t)}$ denotes the summed gradient information over all patterns of the pattern set (Batch learning, or learning by epoch). This also means that the weight update and adaptation are performed after the gradient information of the whole pattern set is computed. By replacing the $\Delta_{ij}^{(t)}$ by a constant update-value Δ, equation (9) yields the so-called 'Manhattan'-update rule.

The second step is to determine the new update-values $\Delta_{ij}^{(t)}$. This is based on a sign-dependent adaptation process.

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^{+} * \Delta_{ij}^{(t-1)} & , \quad if \quad \dfrac{\partial E}{\partial w_{ij}}^{(t-1)} * \dfrac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ \eta^{-} * \Delta_{ij}^{(t-1)} & , \quad if \quad \dfrac{\partial E}{\partial w_{ij}}^{(t-1)} * \dfrac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ \Delta_{ij}^{(t-1)} & , \quad else \end{cases} \qquad (10)$$

where $0 < \eta^{-} < 1 < \eta^{+}$

In words, the adaptation rule works as follows: every time the partial derivative of the corresponding weight w_{ij} changes sign, which indicates that the last update was too big and the algorithm has jumped over a local minimum, the update-value $\Delta_{ij}^{(t)}$ is decreased by the factor η^{-}. If the derivative retains its sign, the update-value is slightly increased in order to accelerate convergence in shallow regions. Additionally, in case of a change in sign, there should be no adaptation in the succeeding learning step (Riedmiller, 1994 b). In practice, this can be achieved by setting $\frac{\partial E}{\partial w_{ij}}^{(t-1)} = 0$ in the above adaptation rule.

In order to reduce the number of freely adjustable parameters, often leading to a tedious search in parameter space, the increase and decrease factors are set to fixed values. The choice of the decrease factor η^{-} was lead by the following considerations: if a jump over a minimum occurred, the previous update-value was too large. For it cannot be derived from gradient information how much the minimum was missed, the correct value had to be estimated. On average it would be a good guess to halve the update-value (maximum likelihood estimator), so it was chosen as $\eta^{-} = 0.5$. The increase factor η^{+} on the one hand, has to be large enough to allow fast growth of the update-value in shallow regions of the error function, but on the other hand the learning process can be considerably disturbed if a too large increase factor leads to persistent changes of the direction of the weight step. In several experiments, it was found that the choice of $\eta^{+} = 1.2$ gives very good results, independent of the examined problem. Slight variations of this value did neither improve nor deteriorate convergence time. So, in order to get parameter choice more simple, it was decided to constantly fix the increase parameter to $\eta^{+} = 1.2$.

All this allows Rprop to try to adapt its learning process to the topology of the error function (Riedmiller, 1994 b) (Riedmiller, 1994 a).

2.4.1 Rprop Algorithm:

The following pseudocode in Figure 2.7 describes how Rprop updates the weights. The rest of the algorithm is the same as BP in Figure 2.5 (you can simply insert this code instead of the code in the weight adjustment part of Figure 2.5). The code put here is only for one weight layer, but it applies on any other weight layer, only the computation of the Gradient $\frac{\partial E^{(t)}}{\partial w}$ changes according to equations (5), (7) and (8) in section 2.3.

The **minimum (maximum)** operator delivers the minimum (maximum) of two numbers; the **sign** operator returns +1, if the argument is positive, -1, if the argument is negative, and 0 otherwise (Riedmiller, 1994 b).

```
$\forall i, j: \Delta_{ij}(t) = \Delta_0$
$\forall i, j: \frac{\partial E}{\partial w_{ij}}(t-1) = 0$

{Please read about these initial values in section 2.4. Values for $\Delta_0$, can be set by the Random function
described in section 2.3.1}

Repeat
        Compute Gradient $\frac{\partial E}{\partial w}(t)$
        For all weights and biases {
                If $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0)$ then {
                    $\Delta_{ij}(t) = \min imum(\Delta_{ij}(t-1) * \eta^{+}, \Delta_{\max})$
                    $\Delta w_{ij}(t) = -sign(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$
                    $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$
                    $\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$
                }
                else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0)$ then {
                    $\Delta_{ij}(t) = \max imum(\Delta_{ij}(t-1) * \eta^{-}, \Delta_{\min})$
                    $\frac{\partial E}{\partial w_{ij}}(t-1) = 0$
                }
                else if $(\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0)$ then {
                    $\Delta w_{ij}(t) = -sign(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$
                    $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$
                    $\frac{\partial E}{\partial w_{ij}}(t-1) = \frac{\partial E}{\partial w_{ij}}(t)$
                }
        }
Until (converged)
```

Figure 2.7: Weight update in Resilient backpropagation algorithm.

Chapter 3

Literature Review

Researchers have been trying to build many computational models to mimic different processes in the human brain, ever since computers appeared and even before that (Figure 1.2). One might say, we are on the quest trying to build the artificial eye or the artificial ear, in an overall model of the artificial brain. Two distinct fields have been identified:

- Speech Recognition, in which the computer is used to recognize speech and convert it into text.
- Text Synthesis, in which written text is synthesized by a computer into speech or phonological representation.

3.1 Research in Speech Recognition and Text Synthesis:

We will talk first about old attempts in speech recognition and text synthesis, then we will talk about using ANNs for speech recognition and then for text synthesis.

As one reads through the literature, one will find different attempts, some are from a computational point of view, and others from a linguistic point of view. It was in the late eighties when the two started working together in this interdisciplinary field. During those years, case analysis was made from the cases provided by the two points of view, then solutions started to appear (Barry, 1985).

There has been little interaction between speech recognition and linguistic phonology in the early eighties, due to the different aims of the two fields. Hoequist (Hoequist, 1987) from the Linguistics Department at the University of Cambridge, calls for interaction between the two fields as he proposes that information about phonological processes can be of use in Automatic Speech Recognition (ASR). He compared two phonological rule components in a system having limited dialect normalization, one illustrating context-free rules and one making use of context sensitivity. In either components, the system utilizes the interaction between speech recognition and linguistic phonology, in the sense that speech is recognized into phonological strings, which are then used by text synthesizers to generate text from these phonemes. It is argued that a context-free set of phonological rules is inadequate to deal with phonological processes in natural languages (Hoequist, 1987).

In his paper, Hoequist, also claims that techniques used for isolated-word recognition suffer badly degraded performance when confronted with continuous speech (Hoequist, 1987). He also describes a context-free phonological parser presented by Church (Church, 1983). This context-free system, does not ignore context, but encodes it in such a way that a context-sensitive formalism is claimed to be unnecessary. Hoequist analizes Church's system and points out its difficulties and describes a case in which the system failed to parse. Then he presents an example on context-sensitive parsing. It is the 'two-level' parser proposed first by Koskenniemi in 1983 and then developed for phonological rule implementation at Cambridge (Ritchie *et al*, 1986). The core of two-level parsing is its encoding of rules into nondeterministic, finite-state automata, which simulates context sensitive rules. Since context sensitive rules are formally more powerful than finite-state machines, it would be possible to write rules to generate strings not correctly analysable by any finite-state device (Hoequist, 1987). However, phonological and phonetic processes in a natural language do not ever seem to produce such outputs. These automata can be envisioned as moving simultaneously along two 'tapes' (hence the name 'two-level'), one representing the input and one representing a graph through a lexicon having a tree structure. The rules check whether the current input character can be matched to the current lexical character. If all the rules allow the current pairing, the next character is taken from the input and checked for any allowable matches in the lexicon, and the process is repeated. This continues until the end of the input is reached (a successful parse) or no pairing of lexical and input characters is possible at some point (Hoequist, 1987).

While a rule component allowing context sensitivity is superior in this respect, it cannot be said to be optimal. The rules' formal power enables them to undo the effects of processes which defeat context-free rules and also overgenerates mappings between the input and the lexicon. The obvious next step is to find ways to prune these mappings. Two methods are already in use. One is that of automatically checking the lexicon to see whether it contains the segment sequences the rules produce, the other is the reliance on an early identification of the speaker's dialect to determine which rules will be applied to an input string (Hoequist, 1987).

Although Hoequist did not mention any expert or rule-based systems, one might notice that what he described can be implemented as an Expert System (ES).

In Morocco, researchers did not ignore the fact that an ES can be constructed for Arabic language. Actually Rajouani and his colleagues (Rajouani *et al*, 1987), were able to do text-to-speech synthesis-by-rule of Arabic language to some point of accuracy. The input to the system could be any typed Arabic text. The orthographic phonetic translation is designed as a set of arborescent (i.e. tree-like) tests relative to the right and left contexts of an analysis window that slides along the sentence.

The system that Rajouani and his colleagues devised, works in two steps, the first step consisted of analysis of Arabic speech using linear prediction techniques. The second step was the development of a text-to-speech conversion system based on the concatenation of diphones. The system allows the transformation of any orthographic text entered through a terminal to a corresponding vocal string. It also utilizes a component (a linguistic processor) that first starts by transcribing the letters, symbols and punctuation of the text into a sequence of phonemes and stress markers.

(Rajouani *et al*, 1987) describes the synthesis strategy as follow: The parametrization of the Arabic sounds has been done by a trial and error procedure. Each phoneme-segment is stored such as a unique characterization including phonetic information, inherent values of temporal parameters and synthesizer control parameters. The stops are stored as the segment corresponding to the burst. For the consonants sounds, the targets of the first three formants are computed by suitable linear equations depending on the rounded/unrounded feature of the following vowel. A set of complicated rules computes the control parameters to be sent to the synthesizer every 10ms. Formant target values for two consecutive segments are connected by means of ½ cosinus interpolation taking into account the transition durations. Amplitudes of the excitation sources and formant resonators ones are computed by adequate transition rules which are function of the phonetic classes of the two consecutive segments and the position of the segment on the sequence (initial, medial, final, intervocal) (Rajouani *et al*, 1987). Three states of problems are defined by Rajouani's system, (1) the letter-to-phoneme translation, (2) the linguistic processing rules and (3) the parametrization of the Arabic sounds according to the used synthesizer. We will come back to these problems and discuss them in Chapter 5, as they are part of the problems we faced in our system.

The generation of the rules in Rajouani's system, is based on an intensive manual investigation and compilation of a wide corpus of natural speech, made by him and his colleagues. The evaluation and validation of the rules is done by a hard "trial and error" procedure. The best justification for the rules is that they produce acceptable synthetic speech (Rajouani *et al*, 1987).

As Rajouani and his colleagues in Morocco tried to use rule-based synthesizers, others noticed the emergent use of ANNs in pattern recognition, and started studying the possibility of such a neural synthesizer using English language.

One of the early attempts was that of (Watrous *et al*, 1987), who used a feedforward NN with recurrent links and BP learning to develop an English speech recognition NN. In the paper referenced here, Watrous described simple experiments to discriminate the consonants [b,d,g] in the context of [i,a,u] using CV (Consonant-Vowel) words. The results show that connectionist networks can be designed and trained to successfully discriminate similar word pairs by learning context-dependent acoustic-phonetic features. The speech data used for these experiments consisted of isolated CV utterances for a single speaker, consisting of the stop consonants [b,d,g] in combination with the vowels [i,a,u]. Five repetitions of each CV word for a total of 45 utterances were spoken into a commercial speech recognition device (Siemens CSE 1200), where it was passed through a 16-channel filter bank, full-wave rectified, log compressed and sampled every 2.5 millisecond. The data files were segmented by hand to extract the transition portion of the CV word. The segmentation was done to decrease the computational load on the optimization algorithm, which is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm (Fletcher, 1980). What this algorithm does, is that it combines a linear search along a minimizing vector with an approximation of the second derivative of the object function. In this way, knowledge about the structure of the error surface is used to select optimal search directions and achieve much more rapid convergence, especially in the neighbourhood of the function minima (see Figure 2.4)(Watrous *et al*, 1987). The transfer function at the output units is the Gaussian function, with a variable center point and sharpness parameter.

A randomly initialized network with 16 hidden units was optimized for consonant discrimination. The squared error decreased from 2934 to 121 after approximately 500 iterations. A similarly initialized network, with 10 hidden units, was optimized for vowel discrimination. The squared error decreased from 2995 to 38.2 after approximately 140 iterations.

The analysis of the hidden unit activation in response to the training data showed little or no context dependence. The features responded similarly to the appropriate vowel across consonant contexts (Watrous *et al*, 1987). In conclusion, it has been shown that a NN with a temporal data flow architecture with recurrent links, can infer directly from real speech data a mechanism for acoustic-phonetic discrimination.

As more research progresses, some researchers combine the movements of the speaker's mouth and facial expressions (i.e. visual data), with speech recognition; to correctly identify vowels (Yuhas *et al*, 1992). Yuhas defines the unit of visual data as the *viseme*.

Other research, includes Kohonen's Neural Phonetic Typewriter (Kohonen, 1988), which transcribes dictation (speech recognition) using an unlimited vocabulary, and Hinton's NN for isolated word recognition from an input speech (Hinton *et al*, 1990).

The typical model for a text-synthesis neural network is the work of Rosenberg and Sejnowski, which they call NETtalk (Sejnowski *et al*, 1987). NETtalk is considered by the literature as the benchmark and the standard for every research in text-to-speech by neural networks.

3.2 A Neural Network that Learns to Pronounce English Text: NETtalk:

The idea of our work was inspired by Sejnowski and Rosenberg (Sejnowski *et al*, 1987). They constructed a neural network called NETtalk that learns to convert English text to speech. It demonstrates that a relatively small network can capture most of the significant regularities in English pronunciation as well as absorb many of the irregularities, by converting strings of letters to strings of phonemes[1] (Sejnowski *et al*, 1992).

3.2.1 NETtalk's Topology:

NETtalk is a feedforward neural network with two layers of weights (i.e. Three layers of PEs). There are seven groups of units in the input layer, each group encodes one letter of the input at any one time as shown in Figure 3.1. The desired output of the network is the correct phoneme, associated with the center, or fourth, letter of this seven letter "window". The other six letters (three on either side of the centre letter) provide a partial context. The text is stepped through the window letter-by-letter. At each step, the network computes a phoneme, and after each word the weights are adjusted according to how closely the computed pronunciation matched the correct one (Sejnowski *et al*, 1992).

[1] A phoneme is the basic unit of speech, while a grapheme is the basic unit of writing, *the letter in an alphabet*.

The activation function used here is the sigmoidal activation function (in Figure 2.2 and described by equations (1) and (2) in section 2.3), it is zero if the input is very negative, then increases monotonically, approaching the value one for large positive inputs.
This form regularly approximates the firing rate of a neuron as a function of its integrated input, if the input is below threshold there is no output; the firing rate increases with input, and saturates at a maximum firing rate. The behaviour of the network does not depend critically on the details of the sigmoid function (Sejnowski *et al*, 1992).

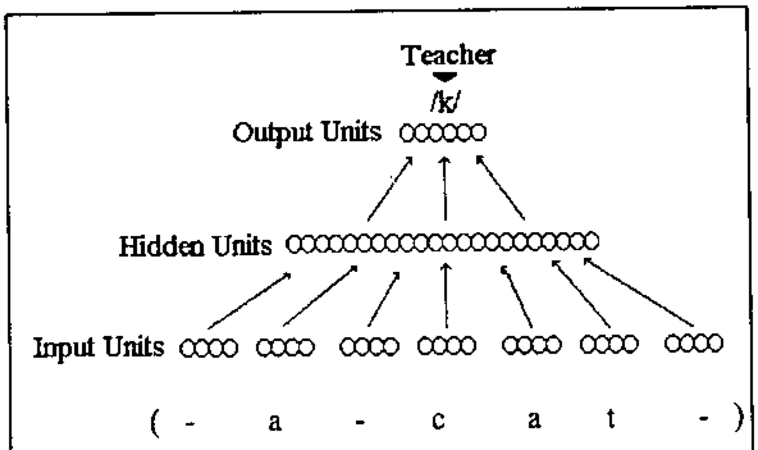

Figure 3.1: Schematic drawing of the NETtalk network architecture. A window of letters in an English text is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 hidden units to produce patterns of activity in 26 output units (Sejnowski *et al*, 1992).

3.2.2 Representation in NETtalk:

The letters of English text are represented locally within each input group by 29 dedicated units, one for each letter of the English alphabet, plus an additional three units to encode punctuation and word boundaries. Only one unit in each output group is active for a given input (Sejnowski *et al*, 1992).

Given the number of letters (graphemes) in the English alphabet, which is 26, we can compute the number of PEs at the input layer as (26+3) X 7 = 203. The phonemes, in contrast, are represented in terms of 21 articulatory features, such as point of articulation, voicing, vowel height, and so on. Five additional units encoded stress and syllable boundaries, making a total of 26 output units in the output layer (for a listing of these articulation features and phonemes used in NETtalk, please refer to (Sejnowski *et al*, 1992)).

The goal of the learning algorithm is to search effectively the space of all possible weights, for a network that performs the mapping of graphemes at the input layer to phonemes at the output layer.

Representation at the output layer is considered distributed representation; since each output unit participates in the encoding of several phonemes (Sejnowski *et al*, 1992)(Rich *et al*,1991). The hidden layer has 80 hidden PEs. This results in data compression; because hidden PEs are used by the network to form internal representations that are appropriate for solving the mapping problem of letters to phonemes. Later after analysis of the hidden layer, it was found that this data compression is possible due to the redundancy in English pronunciation. Also, it was found that, on the average, about 20% of the hidden units were highly activated for any given input, and most of the remaining hidden units had little or no activation. Thus, the coding scheme could be described neither as a local representation, which would have activated only a few units, or a "holographic" representation, in which all of the hidden units would have participated to some extent. It was apparent, even without using statistical techniques, that many hidden units were highly activated only for certain letters, or sounds, or letter-to-sound correspondences. A few of the hidden units could be assigned unequivocal characterization, such as one unit that responded only to vowels, but most of the units participated in more than one regularity (Sejnowski *et al*, 1992).

3.2.3 Learning in NETtalk:

NETtalk uses a variation of supervised BP learning. The text is stepped through the window letter-by-letter. At each step, the network computes a phoneme, and after each word the weights are adjusted according to how closely the computed pronunciation matched the correct one. The error signal is back-propagated only when the error margin is greater than 0.1 (Sejnowski *et al*, 1992).

In addition to weights between layers that connect the PEs, each PE also has a threshold which can vary. To make notation uniform, the threshold was implemented as an ordinary weight from a special PE, that always had an output value of 1. This fixed bias acts like a threshold whose value is the negative of the weight.

As the overall goal of the learning algorithm is to search effectively the space of all possible weights, for a network that performs the mapping. Technically, the goal of the learning procedure is to minimize the average squared error between the values of the output units $s_i^{(N)}$ and the correct pattern, s_i^*, provided by a teacher:

$$Error = \sum_{i=1}^{J}(s_i^* - s_i^{(N)})^2 \qquad (11)$$

where J is the number of units in the output layer N. This is accomplished by first computing the error gradient on the output layer:

$$\delta_i^{(N)} = (s_i^* - s_i^{(N)}) f'(net_i^{(N)}) \qquad (12)$$

and then propagating it backwards through the network, layer by layer:

$$\delta_i^{(n)} = \sum_j \delta_j^{(n+1)} w_{ji}^{(n)} f'(net_i^{(n)}) \qquad (13)$$

where $f'(net_i)$ is the first derivative of the function $f(net_i)$ (refer to equations (2) and (3) in section 2.3), and n is any layer (N being the output layer).

These gradients (also found in the BP algorithm in Figure 2.6), are the directions that each weight should be altered to reduce the error for a particular item. To reduce the average error for all the input patterns, these gradients must be averaged over all the training patterns before updating the weights, this is batch learning by epoch. Another method, that was actually used by (Sejnowski *et al*, 1992), is to compute a running average of the gradient with an exponentially decaying filter:

$$\Delta w_{ij}^{(n)}(u) = \alpha \Delta w_{ij}^{(n)}(u-1) + (1-\alpha)\delta_i^{(n+1)} s_j^{(n)} \qquad (14)$$

Where α is the momentum (typically 0.9), u is the number of input patterns already presented up to the point this equation is to be calculated (you can consider it as a parameter of time), so $\Delta w_{ij}^{(n)}(u-1)$ is the change the weight experienced during the previous forward-backward pass, and n is the number of layers (N being the output layer). The smoothed weight gradients $\Delta w_{ij}^{(n)}(u)$ can then be used to update the weights:

$$w_{ij}^{(n)}(t+1) = w_{ij}^{(n)}(t) + \eta \Delta w_{ij}^{(n)}(u) \qquad (15)$$

where t is the number of weight updates (also a parameter of time) and η is the learning rate. This ensured that the network did not overlearn on inputs that it was already getting correct (Sejnowski *et al*, 1992).

As for the training set, two texts were used to train the network : phonetic transcriptions from a corpus of informal continuous speech, and 1000 most commonly used English words from the 20,012 words in the *Miriam Webster's Pocket Dictionary*.

Two procedures were used to move the text through the window of 7 input groups. For the corpus of informal continuous speech the text was moved through in order of word boundary between the words. Several words or word fragments could be within the window at the same time. For the dictionary, the words were placed in random order and were moved through the window individually.

Each weight in the network was adjusted after every word to minimize its contribution to the total mean squared error between the desired and actual outputs (Sejnowski *et al*, 1992).

3.2.4 NETtalk Performance:

A simulator was written in the C programming language for configuring a network with arbitrary connectivity (NETtalk), training it on a corpus and collecting statistics on its performance. A network of 10,000 weights had a throughput during learning of about 2 letters/sec on a VAX 11/780 FPA (Sejnowski *et al*, 1992).

On the performance of NETtalk on the ***continuous informal text***, (Sejnowski *et* al, 1992) is quoted saying: *"This was a particularly difficult training corpus because the same word was often pronounced several different ways."*

The percentage of correct phonemes rose rapidly at first and continued to rise at slower rate throughout the learning. When learning was first started, NETtalk sounded like an incomprehensible babbling sound, like a baby when it first learns to talk. Then, word boundaries were recognised and the sound started making sense, until after about 50 passes when words began to clear up.

When the network made an error it often substituted phonemes that sounded similar to each other. For example, a common confusion was between the "th" sounds in "thesis" and "these" which differ only in voicing. Also, few errors in a well-trained network were confusions between vowels and consonants.

As a test of whether the network memorizes the training words or just captures the regular features of pronunciation (test of generalization), a network was trained on the 1024 word corpus of informal speech was tested without training on a 439 words continuation from the same speaker. The performance indicated 78% correctness, which means that much of the learning was transferred to novel words even after a small sample of English words was learned.

Is the network resistant to damage? After making random changes of varying size to the weights, the performance was examined: random perturbations of the weights uniformly distributed on the interval [-0.5, 0.5] had little effect on the performance of the network, and degradation was gradual with increasing damage. This damage caused the magnitude of each weight to change on average by 0.25; this is the roundoff error that can be tolerated before the performance of the network begins to deteriorate and it can be used to estimate the accuracy with which each weight must be specified. It was also noticed that, if the damage is not too severe, relearning was much faster than the original learning starting from the same level of performance.

As for the ***Miriam Webster's Pocket Dictionary*** training. The 1000 most commonly occurring English words that were selected are also amongst the most irregular, so it was also a test of the capacity of the network to absorb exceptions.

The ability to generalize was tested on a large dictionary. Using words from a network with hidden units trained on the 1000 words, the average performance of the network on the dictionary of 20,012 words was 77%.

About how generalization is made in NETtalk, (Sejnowski *et al*, 1992) mentions that all performance figures refer to the percentage of correct output phonemes chosen by the network in the output string after a generalization test.

The performance was also assayed by "playing" the output string of phonemes and stresses through DECtalk, bypassing the parts of the machine that converts letters to phonemes (Sejnowski *et al*, 1992).

In other experiments, the number of input groups was varied from three to eleven. Both the speed of learning and the asymptotic level of performance improved with the size of the window. The performance with 11 input groups and 80 hidden units was about 7% higher than a network with 7 input groups and 80 hidden units (Sejnowski *et al*, 1992).

3.2.5 Conclusions about NETtalk:

Despite the similarities between NETtalk and human learning, NETtalk is too simple to serve as a good model for the acquisition of reading skills in humans. The network attempts to accomplish in one stage what occurs in two stages of human development. Children learn to talk first, and only after representations for words and their meanings are well developed, do they learn to read (Sejnowski *et al*, 1992).

NETtalk can be used to study the importance of particular phonological rules in the context of a particular corpus by presenting the network with nonsense words that are constructed to critically test a proposed rule. The performance of the network can also be studied following damage to the network by either removing units or by disrupting weights which can be compared with reading errors observed in humans suffering from acquired Dyslexia. **NETtalk is clearly limited to handle ambiguities that require syntactic and semantic levels of analysis** (Sejnowski *et al*, 1987).

Chapter 4

A NETtalk-like Neural Network to Pronounce Arabic Text

In this chapter we will present an Arabic text synthesis model that we constructed, and the experiments we carried out on it.

4.1 Duplicating NETtalk:

Since NETtalk is considered a benchmark for any neural-based text synthesis research, we constructed a NETtalk-like ANN in an attempt to duplicate it; we call our neural net, NETtalk2. Many experiments to pronounce Arabic text were made on NETtalk2. Here we will present the structure of NETtalk2, then we will discuss the experiments we ran on Arabic and English text. In the last section of this chapter we will present the results and then discuss the performance of NETtalk2. Investigation and analysis of results will follow in Chapter 5.

4.2 The Topology of NETtalk2:

The architecture of NETtalk2 is mostly the same as NETtalk (in Figure 3.1) because it mimics it. There are two weight layers, i.e., three layers of Processing Elements (PEs).

The Input layer consists of seven input groups in the form of a sliding window, each input group contains 33 PEs, corresponding to 28 letters in the Arabic alphabet plus word boundaries, Full-stop, question mark, exclamation mark and gemination mark[2], so we have 33 X 7 = 231 PEs at the input layer. The hidden layer consists of 80 PEs, like NETtalk, and at the output layer there are 8 PEs, that encode more than 240 phonemes. The set of phonemes used by NETtalk2 was extended to handle phonemes that are capable of representing Arabic and English text (see Appendix A).

The window of size seven letters in NETtalk was chosen for two reasons (Sejnowski *et al*, 1992). First, A significant amount of information needed to correctly pronounce a letter is contributed by nearby letters. Secondly, (Sejnowski *et al*, 1992) were limited by the computational resources to exploring small networks. These same reasons also apply on our research. The limited size of the window also meant that some important nonlocal information about pronunciation and stress could not be properly taken into account by the model. Later we will show how we experimented with larger window sizes.

The transfer function of a PE in NETtalk2 is the sigmoidal function,

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

NETtalk2 also uses the same BP algorithm of NETtalk (equations: 11, 12 ,13, 14 and 15 from section 3.2.3 with the algorithm in Figure 2.6. See Appendix C for the code of the algorithm in ANSI C programming language).

[2] Gemination mark is the "chadda" (ّ).

4.3 Representation of Input and Output in NETtalk2:

The input to NETtalk2, could be any text typed by a Standard 101/102-Key English/Arabic Keyboard. The text could be Arabic or English. The user first types the required text, stores it in a text file, then a certain program (a pre-processor), reads the text file and encodes every character, including Space (which is considered as word-boundary) into a 33-bit binary pattern (from 0 to 32). The gemination mark is also considered as one of the input characters, in which case the pre-processor will simply duplicate the letter (character) over which the gemination mark appears.

Output is represented by an 8-bit binary number that corresponds to the appropriate phonological representation of the input. This is distributed representation, like in NETtalk, in the sense that 8-bit binary numbers in the output file encode more than 240 phonemes, because more than one of the 8 PEs at the output layer can take the value 1 at the same time, generating a binary number between 0 and 255, used later to fetch the corresponding phoneme and sound file.

4.4 How NETtalk2 works:

NETtalk2 (Figure 4.1) reads the binary input to the input layer as groups of 7 at each time, a forward pass is made and output is generated as an 8-bit binary number stored in another text file, and so on, until all the input binary file ends. Another program (a post-processor) scans through the output text file (which by now includes all binary 8-bit numbers that represent the output phoneme for every binary pattern read by NETtalk2) and converts every number to a corresponding phoneme. To generate the sound, each output number, is associated with the binary numbers are then combined by the post-processor -as described in Appendix B- into one long sound file, which can then be played by any commercial software that reads sound files of the .wav format.

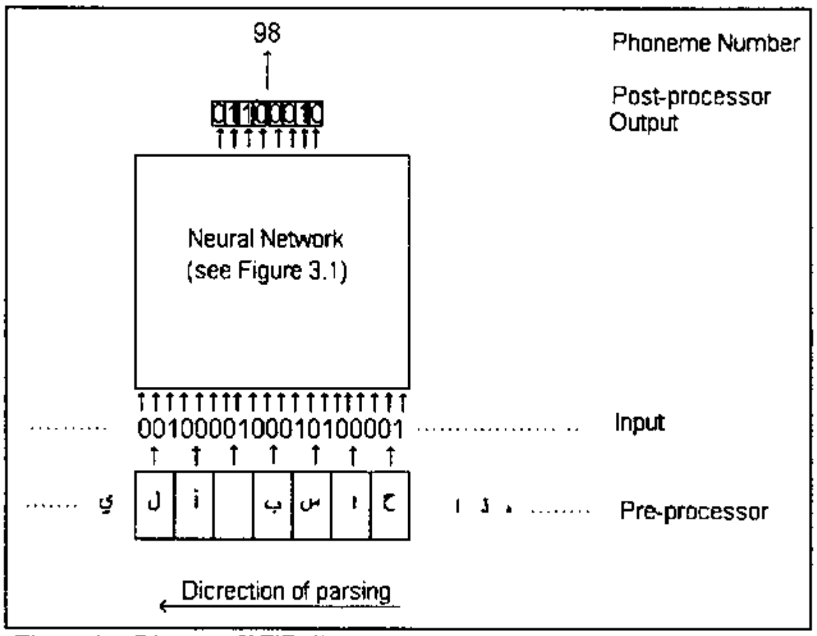

Figure 4.1: Diagram of NETtalk2 system.

4.5 Learning in NETtalk2:

Two learning algorithms were used in NETtalk2. First, the BP algorithm as mentioned in sections 4.2 and 3.2.3 was used to mimic the training and learning of NETtalk, then experiments were made using the Rprop learning algorithm (described in Chapter 2. Code in Appendix C). Results of training using **R**prop are in, most cases, better and faster than BP as we will see later[3].

NETtalk2 was first tested on English using two texts, (1) an informal continuous corpus of text, and (2) the 1000 most commonly used English words (that were used by (Sejnowski *et al*, 1992), see Appendix D).

[3] Benchmark experiments were also ran using Rprop, and compared to be found very close to benchmark results reported in (Riedmiller, 1994 a). Benchmarks include the XOR problem, 3-bit and 6-bit parity, 10-5-10 and 12-2-12 decoders and the two-spirals problem, for more information on these benchmarks, please refer to (Riedmiller, 1994 a).

Also, the two cases were used to train NETtalk2 on Arabic, (1) a continuous corpus of text, and (2) a set of 1024 commonly used Arabic words from everyday use.

During learning, the binary text generated by the pre-processor is stepped through the sliding window, seven letters at a time. At each step, NETtalk2 computes a binary representation of a phoneme corresponding to the middle letter in the window, after each word the weights are adjusted according to how closely the computed pronunciation matched the correct one in the training pair. The error signal is backpropagated only when the error margin is greater than 0.1, using momentum value of $\alpha = 0.9$. As described by NETtalk learning in section 3.2.3.

Two procedures were used to move the text through the window of 7 input groups. For the corpus of continuous text, the text was stepped-through in order of word boundaries, signified by the space between words in the text. Several words or word fragments could be within the window at the same time. Whereas, for the corpus of commonly used words, the words were placed in random order and moved through the window individually, such that, only one word or fragments of only one word could be within the window at the same time.

4.6 Training NETtalk2 on English Text:

To make sure that NETtalk2 can really simulate NETtalk, we ran some experiments with NETtalk2 on English text. These include experiments that are similar to the ones made by (Sejnowski *et al*, 1992).

Some experiments were carried out by using a window of 7 input groups (this is the default unless mentioned otherwise and from now on we will refer to it as size-7 window), other experiments were conducted with windows of 3, 5 and 11 input groups.

We will mention here some short but yet important examples from these experiments. Duplicates of (Sejnowski *et al*, 1992) experiments can be found in subsection 4.6.1

<u>Example E1:</u> First we experimented with the English alphabet, we ran NETtalk2 on the English alphabet in three cases, first case with the alphabet as a set of separate letters, second case with the alphabets as a corpus of continuous letters using a size-7 window and the third case is the same as the second but with size-3 window. See results in Table 4.1. Percentage figures refer to the percentage of correct output phonemes. Those were calculated by the following equation:

$$\textit{Percentage of corrrect output phonemes} = \frac{\textit{Number of correct output phonemes}}{\textit{Total number of output phonemes}} \times 100\%$$

Table 4.1: Training NETtalk2 on the English alphabet. Figures represent percentage of correct output phonemes in the output string of phonemes.

	alphabets as a set of separate letters, using size-7 window	**one-line continuous representation of the alphabets, using size-7 window**	**one-line continuous representation of the alphabets, using size-3 window**
Number of Epochs[4] until 0% error convergence	35 (Rprop) 105 (BP)	15 (Rprop) 50 (BP)	2000 (Rprop & BP)
recalling[5] test using: m x u a s j k	100%	100%	20%
generalization test using: man and dog	10%	54%	10%
generalization test using: man and dog	2%	80%	30%

Example E2: In this experiment, the following set of 40 words (part of the 1000 most commonly used English words) was stepped through NETtalk2 for learning as a continuous text:

[4] An epoch is a full pass (iteration) of the neural net over the training set before updating the weights, and is a characteristic of batch learning.
[5] Recalling is the ability of a neural network to remember what it was trained on when it is presented with the testing set (i.e. using the training set for testing after learning).

and or no not right wrong I am he she it is they we are them their there you her him his when what where how who clue glue whoever however never nor neither either with between among come go

Learning took only 79 epochs using the Rprop algorithm, while it took about 200 epochs using the BP learning algorithm, which was used by NETtalk. After NETtalk2 has learned this corpus of words using BP, strings in Table 4.2 were passed through NETtalk2 as a test of generalization. Table 4.2 also shows different results when different input window sizes were used during training.

Table 4.2: Generalization test on NETtalk2 after training on 40 words. Notice that generalization power increases with incrasing input groups and saturates at size 11.

Input String	**Percentage of correct phonemes in the generated output using different input window sizes**					
	size-5	**size-7**	**size-9**	**size-11**	**size-13**	**size-15**
who is he	80%	90%	92%	92.5%	92%	89%
weed	60%	83%	85%	85%	85.1%	80%
ever	73.4%	100%	100%	100%	100%	91%
nizar radi	68%	86%	89%	90%	90%	77%
not yet	62%	77%	77%	79%	77%	62%

Figures in Table 4.2 reveal that the ability of NETtalk2 to generalize on novel words, after training, increases as the size of the input window (used during learning) increases, but then saturates at size 11, i.e. generalization performance is forzen at certain values or increases in small fractions when sizes larger than 11 are used, but then starts decreasing after size 15.

As we will see later, also increasing the input window size increases the leanring speed (by decreasing the number of epochs needed to reach converegence), but of course each epoch takes more time to finish than with a smaller input window; due to the increased number of PEs, connections and thus calculations made by NETtalk2.

Example E3: As another experiment on the effects of having an input window of a different size than 7 letters, the following continuous corpus of 10 English words:

```
and or no not right wrong I am he
```

was stepped through NETtalk2 with an input window of size 5, in this case, NETtalk2 finished training in 28 epochs using Rprop as the learning algorithm, while it took around 100 epochs using BP. When the size of the input window was changed to 11 input groups (11 letters), training took only 24 epochs (Rprop) and 92 epochs (BP), but, when an input window of size 3 (only one letter at each side of the letter to be pronounced) was used, the learning algorithm did not converge (i.e., NETtalk2 could not learn using either of the learning algorithms).

When the same text was used on the standard size-7 window, it finished training in 27 epochs using Rprop and in 97 using BP (Figure 4.1).

After NETtalk2 was trained on the above text, it was put on a test of generalization with the strings in Table 4.3. Results in Table 4.3 show that generalization also improved with increasing window size. As the window size was increased from the default of 7 to 11, performance of NETtalk2 increased by 6%, as also reported on NETtalk (Sejnowski *et al*, 1992).

Table 4.3: Training NETtalk2 using BP with different sizes of the input window.

Input Text	Correctness percentage in output phonemes using size-5 window	Correctness percentage in output phonemes using size-7 window	Correctness percentage in output phonemes using size-11 window
`hero`	77%	80%	86%
`rite`	76%	79%	83%
`not and wrong`	95%	100%	100%
`I am not wrong`	96%	100%	100%
`yes or no`	70%	73%	77%

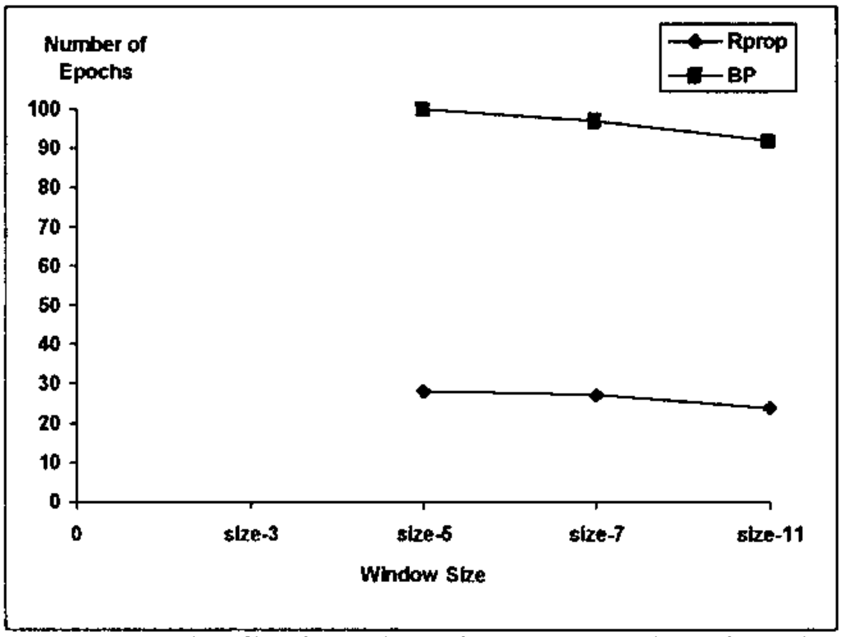

Figure 4.2: Number of learning epochs (y-axis) decreases as the input window size (x-axis) increases. When the window size is 3, number of epochs is infinite, as described in example E3.

Example E4: When NETtalk2 was trained on the following small corpus of continuous text using size-11 window:

```
The  Computer Science department at the University
of  Jordan  announces  the starting of its Masters
program,  from the point that the current number of
eligible students is adequate
```

it finished training in around 200 epochs using BP, as a test of generalization, strings in Table 4.4 were passed through the net. Results of generalization are reported in the Table.

Table 4.4: Generalization results on a small English text.

Input string	Correctness percentage in phonemic output string
`scientist`	95%
`announcement`	90%
`current students of the university of jordan`	93%
`enough number of students led to starting the program`	75%
`programmer`	97%

Example E5: Later on, we tried NETtalk2 on extracts from children stories due to the simplicity in their text. Consider the following first lines from the children story *"The Golden Goose*[6] *"*:

[6] "The Golden Goose" is by Betty Evans, copyrights for LADYBIRD BOOKS LTD MCMLXXXI, Loughborough, UK.

```
Once  upon a time, there was a man who had a wife and
three sons.
They all lived in a cottage on the edge of a forest.
The  youngest son was called Simpleton, and everyone
laughed  at  him  because he wasn't as clever as his
brothers.
```

NETtalk2 needed 120 training epochs when Rprop was used to learn this text with the default input window size, while it needed 200 epochs when BP was used. When a window of size 11 was used, NETtalk2 needed only 70 epochs using Rprop and 107 using BP. Generalization tests results are shown in Table 4.5.

Table 4.5: Generalization results of training NETtalk2 on a simple English story using different learning algorithms with different window sizes.

Input test string	**Percentage of correct output phonemes using input window of size 7**		**Percentage of correct output phonemes using input window of size 11**	
	BP	**Rprop**	**BP**	**Rprop**
everybody laughed at simpleton	86%	90%	92%	97%
he was dump	64%	70%	70%	73%
the man with his wife and three sons lived on the edge of a forest	79%	87%	86%	96%
one day the oldest son had to go into the forest to cut firewood	53%	53%	57%	63%

4.6.1 Duplicating Sejnowski and Rosenberg's Experiments:

When NETtalk2 was trained on the same 1000 corpus of words used by (Sejnowski *et al*, 1992) using BP it took around 7000 epochs, while with Rprop it took around 4000 epochs. For a listing of the 1000 words corpus, see Appendix D.

In the following experiments, we will concentrate on NETtalk2's performance using BP with the modifications that (Sejnowski *et al*, 1992) introduced in section 3.2.3.

<u>NETtalk2 on a corpus of separate words:</u> After NETtalk2 has been trained on the 1000 corpus of separate words, it was tried on a corpus of 10,000 separate words (a continuum from the *Miriam Webster's Pocket Dictionary*) without further training. The correct output was generated in 75% of the cases (compared with the result of 77% reported in (Sejnowski *et al*, 1992)). When the larger corpus (the 10,000 words) was passed through NETtalk2 for learning, generalization performance jumped to 90% after 100 epochs.

The same experiment was tried with an input window of size 11. When the corpus of 10,000 words was passed through the pre-trained net, it generated a correct output in 77% of the cases. After 5 passes through the larger corpus, performance improved to 89%.

Different sizes of input windows were tried and varied from three to eleven. Both the speed of learning and the neural net's generalization performance improved as the size of the window increased, also as reported by (Sejnowski *et al*, 1992) (refer to Example E3 and Figure 4.1).

<u>NETtalk2 on a continuous text:</u> To test NETtalk2 on continuous text, first it was successfully trained on a corpus of 1024 continuous words, then passed over a continuation

of 500 new words and was able to generalize successfully on 76% of the cases (NETtalk was reported to have succeeded on 78%). **Which indicates that much of the learning was transferred to novel words even after a small sample of English words was presented (i.e., NETtalk2 was able to generalize on many new words even after few words were presented during leanring).**

If the previously-mentioned results on NETtalk2 are compared with results reported by (Sejnowski *et al*, 1992) on NETtalk, a very close match will be found, which asserts the validity of NETtalk2 as a model of NETtalk.

4.7 Training NETtalk2 on Arabic text:

The goal of this research is to train a NN to pronounce Arabic text. We chose the way most researchers follow in this field, which is NETtalk, that has became a benchmark in training NNs on text-to-speech problems. As described, a NETtalk-like algorithm was devised and called NETtalk2. We ran NETtalk2 on English text and received very good and close results to those of NETtalk (Sejnowski *et al*, 1992).

We will now present examples on experiments we conducted using NETtalk2 on Arabic text, then we will compare the results with those in the previous section where NETtalk2 was trained on English text. In the experiments here, the value of the learning rate η used in training is 0.35 and the value of the margin error ε between required output and actual output is 0.4 (i.e. if . $\varepsilon \leq 0.4$ then the output is considered correct).

Finally, we will show the overall performance of NETtalk2 on Arabic text. Analysis and further discussion of the results will be presented in Chapter 5.

Example A1: On experimenting with the Arabic alphabet, all the alphabet was presented as a continuous set of letters, NETtalk2 was then required to learn it to be pronounced as all letters being Front Low Unrounded(7) .

NETtalk2 learned the alphabet in around 38 epochs using BP and 27 epochs using Rprop. While using BP, different values of the momentum α were tried every time (see the graph in Figure 4.2). In the graph, the learning processes was stopped every 10 epochs and the percentage of *incorrect* output phonemes calculated. It was found that, as the value of α increases, the percentage of incorrect phonemes starts with a relatively big number, but then decreases rapidly and NETtalk2 converges faster than if with a smaller α. Since the learning speed depends on the convergence of the algorithm; it also increases with increasing α, but starts to deteriorate with $\alpha \geq 0.4$ due to over-training(8) .

Table 4.6 shows generalization and recalling tests after NETtalk2 has learned the front low unrounded alphabets. Figures refer to percentage of correct output phonemes in the phonemic output string.

(7) Front Low Unrounded: جميع الأحرف مفتوحه .

(8) Over-training is the case when leaning is going fast due to a large momentum or learning rate, and it jumps in big steps so it misses the global minima and diverges away from it. The symptoms of over-training is when the average squared error reaches a low value but then suddenly increases rapidly with an increasing number of training epochs that might continue infinitly.

Table 4.6: Training NETtalk2 on the Arabic alphabet (numbers represent percentage of correct output phonemes after using BP for training).

	alphabets as a set of separate letters, using size-7 window	**one-line continuous representation of the alphabets, using size-7 window**	**one-line continuous representation of the alphabets using size-3 window**
Number of Epochs until 0% error convergence	49 (Rprop) 200 (BP)	27 (Rprop) 60 (BP)	>2000 (Rprop & BP) (diverges)
recalling test using: ب ث ق ص غ ا هـ	100%	50%	0%
generalization test using: نحو أمس	50%	0%	0%
generalization test using: نحو أمس	10%	50%	0%

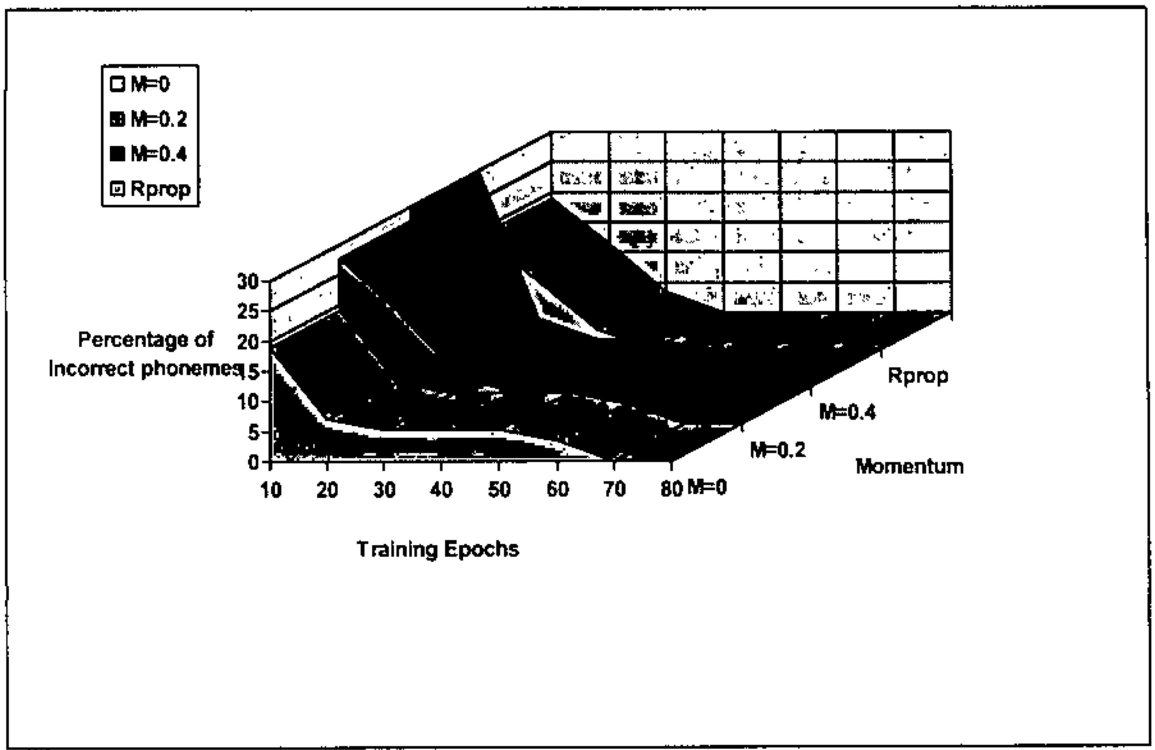

Figure 4.3: Training NETtalk2 on the Arabic alphabets with BP using different Momentum values (M) on the z-axis and with Rprop (last category on the z-axis). Values at the x-axis represent number of training epochs, while values at the y-axis represent percentage of *incorrect* phonemes in the output string measured every 10 epochs during training.

<u>Example A2</u>: In this experiment, NETtalk2 was trained on a very small corpus of continuous text using window of size 7, and then tested for generalization on different input orderings of the same text.

The text is:

أنا الموقّع أدناه

NETtalk2 took 29 epochs using BP to learn this phrase. Table 4.7 shows results of the generalization tests.

Afterwards, the phrase was extended to:

أنا الموقّع أدناه، نزار راضي مبروكه

The phrase "نزار راضي مبروكه" alone took only 31 training epochs using BP, but when it was combined as above, the whole sentence was used for training in two ways:

- When NETtalk2 was trained on it over the old weights from the first phrase (i.e., training continued), it converged in an average of 40 epochs.
- When it was trained on it from initial random weights, it surprisingly converged in an average of 37 epochs.

In these two cases, the average numbers presented were calculated over five trials of the same experiment.

Then the text was extended again to:

أنا الموقّع أدناه، نزار راضي مبروكه، أشهد بأن المعلومات الوارده أدناه صحيحه و خاليه من الأخطاء

Which took NETtalk2 65 epochs of BP learning from initial random weights, generalization tests were also carried as can be seen in Table 4.8.

Table 4.7: Generalization results of training NETtalk2 on different input patterns of the same Arabic text. Notice the different cases of the Arabic letter "Alef ," it appears with "Hamza" in some input patterns and without "Hamza" in others.

Input Pattern	**Percentage of correct output phonemes**
أنا أدناه	71%
أدناه أنا الموقّع	40%
أدناه الموقّع أنا	40%
أنا الموقّع أدناه	**100%**
أنا المُوقّع أدناه	67%
أنا المُوقّع ادناه	67%
دنى	0%
وَقَعْ	25%
قمع	0%

Table 4.8: Generalization results on the final extended text of Example A2.

Input String	Correctness percentage in output
صح	0%
صحه	0%
رزان	0%
علوم	0%
شهادة الموقع نزار	36.4%
ا ب ت ث ن ز ر م ع	22%
ابتثزرمع	33%

The phrase: "أنا الموقع أدناه" was also used to test NETtalk2 convergence speed with different values of the learning rate η , Table 4.9 shows different values of the learning rate with corresponding number of epochs that NETtalk2 took to learn the phrase. In Figure 4.3 we can see that convergence speed increases (i.e. number of epochs to learn decreases) with increasing the learning rate, but then starts to shoot off due to over-training after 0.55, the value of 0.35 has the minimum number of training epochs, and can be considered a very good value to implement in a learning algorithm, as recommended by (Rich *et al*, 1991).

Table 4.9: Learning Rate η vs. convergence speed. Each number was calculated as the average over five trials.

Value of the Leanring Rate η	Number of training epochs NETtalk2 took to successfully learn the phrase in Example A2
0.1	208
0.2	158
0.35	**105**
0.45	140
0.55	114
0.6	183

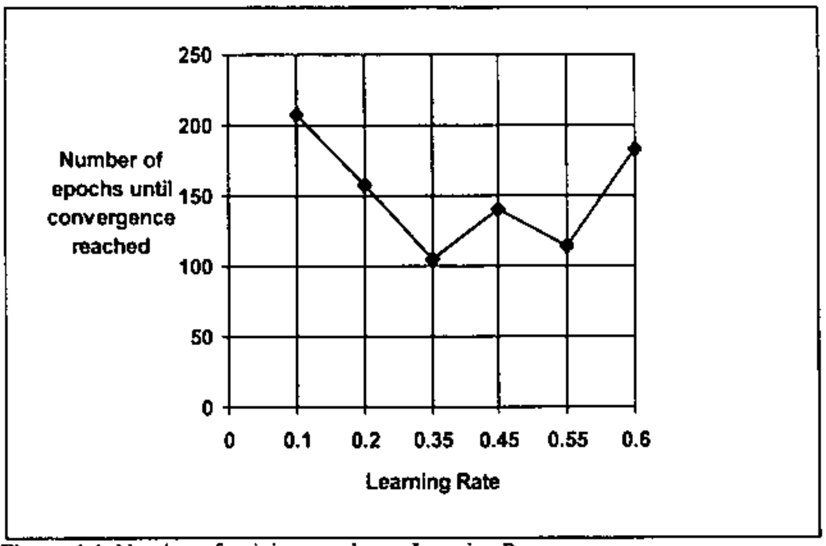

Figure 4.4: Number of training epochs vs. Learning Rate.

Example A3: NETtalk2 was also tested on small Arabic stories using the default window of size 7, an example is this story by the famous Arab writer, Gibran Khalil Gibran:

دخل رجل في ليلة ظلماء الى حديقة جاره، فسرق أكبر بطّيخة وصلت اليها يده و حملها و جاء بها الى بيته.

و عندما كسرها وجد أنها عجراء لم تبلغ بعد نموها، فتحرك ضميره في داخله اذ ذاك، و أوسعه تونيبا، فندم علـــى أنـــه سرق البطّيخه.

After 500 training epochs over this text using BP, NETtalk2 still did not learn the text. It reported a 1.4% error in the output phonemes. After around 3000 epochs the error increased to 2.2%, although the momentum α was 0. This is a case where the net did not converge (learn), but instead diverged away from the minima. In such cases, if the error in output is measured during traning, one will notice that the neural net reaches a local minima, then it either gets stuck in this minima or diverges fast in a way that it misses the global minima. Nevertheless, when Rprop was used, NETtalk2 did converge after 230 learning epochs, but the resulting net failed the generalization tests, as shown in Table 4.10

Table 4.10: Results of generalization tests on a small Arabic story using Rprop as the learning algorithm.

Input string for generalization test	Percentage of correct output phonemes
ما كسرها	37.5%
وسع	0%
رجل سارق	0%
عندما فتح البطيّخه و أكلها كانت لذيذه	3%
جاء رجل فسرق البطيخه في حديقة جاره	20%

Example A4: NETtalk2 was also trained using simple Arabic text extracted from a children story, The following text is a small part from the Arabic version of the story of *"Little Red Ridinghood*[9] *":*

عملت أم رباب بعض الحلوى و نادت ابنتها و قالت لهـــا: رباب خذي هذه الحلوى الى جدّتك.

فرحت رباب لأنها كانت تحب جدّها و تحب أن تأخذ لها الحلوى.

حملت رباب سلة الحلوى و ودّعت أمها. قالت لها أمها: انتبهي يا ابنتي من الذئب الشرير.

توقفت رباب في الغابه و قطفت أزهارا لجدّتها. كانت جدّتها تحب الأزهار كثيرا.

NETtalk2 was able to learn this text in 1600 epoch using Rprop. When BP was used, NETtalk2 did not converge, and continued iterating for more than 5000 epochs. When the

(9) The Arabic version is "رباب في الغابه" by Albir Mutlak, Library of Lebanon, Beirut, 1980.

input window size was increased to 9 using BP, NETtalk2 was able to converge after 3040 epochs. Convergence speed increased with increasing the window size to 11, 13 and 15, but generalization performance started decreasing after size 11 (Table 4.11b). A test of generalization after using Rprop was carried out as shown in Table 4.11a. Table 4.11b shows the results of the same generalization tests, but after using BP for learning with different input window sizes.

Table 4.11a: Generalization results after training NETtalk2 on a children Arabic story using Rprop with the default window size.

Test number	Input string for generalization test	Percentage of correct output phonemes
test1	نادت رباب أمها	46%
test2	قطفت الأزهار و حملت السله ثم ذهبت الى الغابه	37%
test3	رأى الذئب رباب ذات الثوب الأحمر	17%
test4	رباب تحب جدتّها و تحب الأزهار	77%

Table 4.11b: Generalization results after training NETtalk2 on a children Arabic story using BP with different input window sizes.

[redacted]	Input window sizes				
[redacted]	7	9	11	13	15
Number of epochs needed until convergence	No convergence	3040	3000	2952	2830
Generalization Performance					
Test Number	Correctness percentage of output phonemes				
test1	[redacted]	35%	40%	30%	18%
test2	[redacted]	18%	26%	17%	0%
test3	[redacted]	5%	8%	0%	0%
test4	[redacted]	56%	62%	30%	12%

Example A5: The following set of 40 words is part of the 1000 commonly used Arabic words on which NETtalk2 was trained:

من الى عن على في يمشي يركب يشتري وقف يدفع دفع نقود سيّاره طائره نزل اكل يأكل طعام شراب ماء حمّـــام أيـــن

كيف كم لم لن لا نعم صحيح خطأ لو سمحت كان الملك بسم الله الرحمن الرحيم شكراً عفواً

NETtalk2 was trained on this text using BP in two cases:

1. As a corpus of separate words: After 2000 learning epochs, NETtalk2 still did not reach successful training (i.e. 0% error on the training set), but was able to successfully learn 140 phonemes out of the 142 it was trained on and supposed to learn (so the error of running NETtalk2 on *the training set* is 2/142 = 1.4%). After the 2000 epochs, NETtalk2 diverged and could not learn at all. When Rprop was used, NETtalk2 was able to reach successful training after 159 learning epochs.

2. As a corpus of continuous text: After 2000 learning epochs, NETtalk2 still did not reach successful training, after the first 1000 epochs, it was able to reach 98.4% success *on the training set* (that is, 1.6% error), after that it diverged and could not learn (took more than 5000 epochs and still did not learn). But when Rprop was used as the learning algorithm in NETtalk2, it could learn the whole continuous text of 40 words in only 124 epochs. A generalization test was carried on using the input strings in Table 4.12a.

Table 4.12b shows the same test but after different input window sizes were used during training.

Table 4.12a: Generalization tests after training NETtalk2 on a continuous text of the first 40 words from the 1000 corpus of commonly used Arabic words, using Rprop and size-7 input window.

Test Number	Input String	Correctness percentage of output phonemes after training using size-7 window
test1	شكرا لله	33%
test2	بسم الله الرحمن الرحيم	87%
test3	جواب صحيح	39%
test4	ركب السياره و دفع النقود	27%
test5	أين الطعام	35%
test6	شكر	90%
test7	كرش	0%
test8	ذهب من هنا و أكل الطعام ثم شرب الشراب و نام في الغرفة على السرير	53%

Table 4.12b: Generalization tests after training NETtalk2 on a continuous text of the first 40 words from the 1000 corpus of commonly used Arabic words, using Rprop with different input window size during training. At size-3, NETtalk2 was not able to converge. Test numbers refer to tests in Table 4.12a.

[redacted]	Input window sizes									
	5	7	9	11	13	15	17	19	21	25
Number of epochs needed until convergence on the 40 training words	137	124	119	111	111	98	97	87	80	71
Generalization Performance										
Test Number	Correctness percentage of output phonemes									
test1	5%	33%	56%	75%	50%	20%	0%	12%	0%	0%
test2	80%	87%	95%	97%	85%	60%	40%	35%	10%	3%
test3	11%	39%	62%	80%	71%	50%	22%	0%	0%	0%
test4	14%	27%	47%	60%	40%	21%	17%	15%	5%	0%
test5	30%	35%	56%	73%	50%	30%	0%	0%	0%	0%
test6	85%	90%	97%	99%	50%	35%	33%	33%	0%	0%
test7	0%	0%	33%	33%	0%	0%	33%	33%	0%	0%
test8	40%	53%	73%	83%	20%	14%	14%	0%	0%	0%

Another 60 words were added to the text (now we have the first 100 words of the 1000 commonly used Arabic words):

من الى عن على في يمشي يركب يشتري وقف يدفع دفع نقود سيّاره طائره نزل أكل يأكل طعام شراب ماء حمّـــام أيـــن

كيف كم لم لن لا نعم صحيح خطأ لو سمحت كان الملك بسم الله الرحمن الرحيم شكرًا عفوًا مرحبًا وداعًا أريد أذهب

أن ذهب يذهب سيّد سيّده ولد بنت قهوه شاي كهرباء محارم ورقه قلم ميراه ممحاه مسطره كأس فنجان أستاذ طـــــالب

دكتور مدرسه جامعه مسجد جامع كرد باص تلفزيون تلفون ملعقه شوكه سكينه محفظه غاز فرن سيجاره أحمد محمـــــد

خليل شجره تفاح برتقال أحمر شفاه أخضر أصفر أبيض أسود الشمس القمر الليل النهار الظهر الظهيره المساء

After 2000 epochs using Rprop, NETtalk2 reached 4% error on the training set. Using BP, it did not converge.

As a test of generalization, the input strings in Table 4.13a were stepped through NETtalk2, the results are still not much convincing for a net to pronounce Arabic text.

Table 4.13a: Generalization test on a continuous text of 100 Arabic words, after training NETtalk2 using Rprop with size-7 input window.

Input String	Percentage of correct output phonemes
كرد ذهب	0%
مَدْرَسَه مدارس مُدَرِّسَه	20%
ذهب الرجل الى المدرسه	5%
ذهب الرجل الى المدرسه	7%
هذه كلمات جديده	0%

Other tests of generalization were also carried out on the first 300 commnoly used Arabic words, when different input window sizes were used during training, as shown in Table 4.14. For a complete listing of the 1000 Arabic words used to train NETtalk2, refer to Appendix D.

Table 4.14: Generalization test on a continuous text of 300 Arabic words, after training NETtalk2 using Rprop with different input window sizes.

	Input window sizes								
	5	7	9	11	13	15	17	19	21
Number of epochs needed until convergence on the 300 training words	3100	3000	2930	2893	2820	2750	2710	2650	2600
Generalization Performance									
Input test string	**Correctness percentage of output phonemes**								
ذهب الرجل الى المدرسه	2%	14.3%	50%	89%	70%	55%	32%	14%	0%
هذه كلمات جديده	0%	7%	20%	33.3%	33.3%	9%	0%	0%	0%
تطلع الشمس في النهار و يطلع القمر في الليل	7.3%	15%	30%	47%	41%	28%	0%	0%	0%
أريد أن أذهب الى الجامعه	13%	24%	72%	80%	70%	53%	33%	17%	0%
يذهب أحمد في المساء الى مسجد الجامعه	5%	18%	23%	56%	43%	20%	7%	0%	0%
خليل طالب في المدرسه، يركب خليل الباص في الظهر و يمشي في المساء. لدى خليل قلم و محاة و مرآه و هو يحب شرب القهوه و الشاي	10%	30%	47%	65%	45%	27%	14%	0%	0%

The first 10 words from the set of 40 words mentioned previously, are:

من الى عن على في يمشي يركب يشتري وقف يدفع

This set of 10 words was used to test NETtalk2's performance with different input window sizes during training using BP and Rprop. The window size was varied from 3 input groups to 41. Tables 4.15a and b describe convergence speed and generalization performance of NETtalk2 using these different window sizes with BP and Rprop as learning algorithms. Notice that as the window size increases so does convergence speed (i.e. number of training epochs decreases, although NETtalk2 takes more time to finish one epoch) and the number of correct output phonemes, which results in better generalization. When the input window was of size 3, NETtalk2 did not converge and was not able to train using either of the two learning algorithms.

After observing the results in Tables 4.15a-b, we notice that as the window size starts approaching 11, generalization performance saturates and then starts decreasing; due to the large number of configurations and activations that the PEs must be arranged into. We will discuss this thoroughly in the next chapter.

Table 4.15a: NETtalk2's performance on Arabic text using different input window sizes when BP was used as learning algorithm (Blocked cells indicate divergence).

■	Input window sizes											
	5	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**25**	**31**	**41**
Number of epochs needed until convergence on the 10 training words	■	510	501	457	437	413	398	352	301	241	154	103
Generalization Performance												
Input test string	**Correctness percentage of output phonemes**											
وقف يشتري	■	11%	61%	78%	53%	27.2%	0%	0%	0%	0%	0%	0%
مشى ثم وقف	■	0%	9.2%	27.2%	10%	0%	0%	0%	0%	0%	0%	0%
منفي	■	0%	37.5%	50%	12.5%	0%	0%	0%	0%	0%	0%	0%
ركب ثم وقف فاشترى و دفع	■	21%	29%	33.3%	20%	2.1%	0%	0%	0%	0%	0%	0%
يدفع وقف يشتري يركب في على عن الى من	■	46.3%	62.3%	63.4%	35%	28%	5%	0%	0%	0%	0%	0%
مركب مشى و وقف	■	7.1%	15%	39.3%	5%	0%	0%	0%	0%	0%	0%	0%

Table 4.15b: NETtalk2's performance on Arabic text using different input window sizes when Rprop was used as learning algorithm. Blocked cells indicate divergence.

■	Input window sizes											
	5	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**25**	**31**	**41**
Number of epochs needed until convergence on the 10 training words	37	28	29	30	32	26	28	28	27	27	25	26
Generalization Performance												
Input test string	**Correctness percentage of output phonemes**											
وقف يشتري	0%	33.3%	89%	89%	72.2%	0%	39%	33.3%	0%	0%	0%	0%
مشى ثم وقف	9%	45%	36.3%	54.5%	20%	0%	0%	0%	0%	0%	0%	0%
منفي	0%	25%	37.5%	62.5%	25%	0%	0%	0%	0%	0%	0%	0%
ركب ثم وقف فاشترى و دفع	0%	17%	37%	52%	30%	0%	0%	0%	0%	0%	0%	0%
يدفع وقف يشتري يركب في على عن الى من	47%	67%	88%	78%	58.8%	45%	0%	0%	0%	0%	0%	0%
مركب مشى و وقف	29%	39%	32%	43%	21.4%	21%	0%	36%	20%	0%	0%	0%

Large versions of the experiments mentioned in this section were used as a measaure of the performance of NETtalk2 on Arabic text as opposed to its performance on English.

4.7.1 Measuring NETtalk2's performance on Arabic text:

All running and testing of NETtalk2 was made on an IBM RISC RS/6000 machine. NETtalk2 was trained on two forms of English text (section 4.6.1), and on two forms of Arabic text: using a corpus of continuous text, and a corpus of 1024-commonly-used separate Arabic words.

Numbers in the following paragraphs refer to results using the default size-7 window during training, unless mentioned otherwise.

<u>NETtalk2 on a corpus of separate Arabic words:</u> A corpus of 1024 commonly used Arabic words was formed (Appendix D). An attempt was made to train NETtalk2 on this corpus, using BP. Upon tracing the average squared error value as it changes during training, it was found that, usually after about 3000 epochs, the learning process would fall into a local minima and gets stuck, so it does not converge to the global minima needed (this happens technically when the average squared error (equation (11) in Chapter 3) reaches a certain value and sticks there). In some cases when the momentum α was increased (but never more than 0.4; because this led to over-training), NETtalk2 was able to climb slowly out of this minimum (the average squared error starts changing slowly), very slow that it would take a large number of learning epochs compared with NETtalk, in such cases NETtalk2 was stopped after 25000 epochs and *the* average squared error found not less than 0.55 (when ε=0.4).

When Rprop was used, NETtalk2 was able to jump over some local minima to the right direction, or climb out of it fast and smoothly (also observed by tracing the average squared error during training) in around 7100 epochs. Nevertheless, generalization results were not promising and generally did not surpass more than 60% of correct phonological output (like in examples A4 and A5 discussed previously). One can deduce that, even if NETtalk2 was able to converge and learn the given corpus in some cases, it does not generalize to a reasonable percentage of correctness of the phonemes in an output string, corresponding to an input string of Arabic text.

Analysis of this behavior and reasons behind it will be discussed in the next chapter, where more about NETtalk2 performance will also be discussed. In Chapter 5, we will see that one of the most important reasons why NETtalk2 did not converge was the existence of some words more than once in the same training set but requiring different output pronunciation each time.

(Sejnowski *et al*, 1992) also faced the same problem in some cases with the English language as mentioned previously in section 3.2.4 (quoted in Italics).

<u>NETtalk2 on a continuous corpus of Arabic words:</u> Training attempts were also tried on different pieces of Arabic text and also on the same previous 1024 corpus, but in the form of continuous text, using BP. In this case NETtalk2 did not converge.

What continuous text possesses that separate word corpora lack, is the semantics and the rules that govern Arabic grammar and make a sound and correct Arabic statement (refer to Chapter 5).

As (Sejnowski *et* al. 1992) did with NETtalk on English text, input groups of sizes 3 to 41 were tried with NETtalk2 on Arabic text, as we saw in the previous examples. These experiments were also carried out on the 1024 commonly used words. When the sliding input window was of size 3, NETtalk2 could not learn and did not converge even if Rprop was used. At size 5, some cases were not able to learn, but others were able to learn and converge. As the window size was increased from 7 to 41 (7, 9, 11, 13, 15, 17, 19, 21, 25, 31, 41) the speed of learning and generalization performance increased. Generalization performance saturated at size 11 with about 70% correctness on average, then started to decrease. The reason is that NETtalk2 in this case needs more training cases and input configurations to store, and be able to generalize using this large amount of input, presented through the large window.

NETtalk2 was also tried on a continuous Arabic text of around two pages (an Arabic story by Gibran Khalil Gibran), where it unfortunately got stuck and did not converge.

On a small piece of text (2 paragraphs) from the daily newspaper, it did converge after 1700 epochs using BP. Then a continuation (a third paragraph) was passed through NETtalk2 as a generalization test, to result in only 30% correctness in most output cases. When the third paragraph was combined with the training set, and then passed through NETtalk2 for another 500 epochs, performance jumped to 70%, before learning was complete (100%) after 3500 epochs. For a listing of all the training sets, see Appendix D

In an attempt to avoid complexity of the input text and redundancy of input patterns with various pronunciations, attempts were made to train NETtalk2 on small simple children Arabic stories, as in Example A4. In another experiment NETtalk2 was trained on the Arabic version of the children story of *"Little Red Ridinghood"* (except the last 2 paragraphs in it) and did converge in 2000 epochs, using Rprop. The last two paragraphs were then passed through the NN as a test of generalization, to result in around 53% correct output. This result increased to 66% when an input window of size 11 was used, but still not enough to state that we have a NN that can at least pronounce Arabic children stories.

Most training cases that did not converge when the window size was 7 or 11, using either BP or Rprop as learning algorithm, were able to converge after increasing the window size to 25 input groups and more, but generalization results were very bad (not more than 20% generally).

After experimenting with BP and Rprop, one might wonder if a NN -especially NETtalk2- produces better generalization results after using Rprop as a learning algorithm compared with BP. To answer this consider the following example:

After training NETtalk2 on the Arabic alphabet (Example A1), generalization tests (mentioned in Table 4.16) were made, first after NETtalk2 has learned the alphabet using BP and then after it learned it using Rprop. It was found that NETtalk2 generalizes better and is able to recall better after being trained using Rprop, this is also obvious in Tables 4.15a and b.

Table 4.16: Generalization and recall results after training NETtalk2 on the Arabic alphabets using BP and Rprop with the default window size.

Testing Cases	Correctness percentage in phonological output generated after BP learning	Correctness percentage in phonological output generated after Rprop learning
ب ن ق ص ع أ هـ	50%	61.5%
نحو أمس	0%	0%
نحو أمس	0%	0%
أ ب ت ج ح خ م ن ك	66.7%	83%
صباح الخير	0%	10%

Most of the experiments mentioned in this chapter, were also ran using a commercial NN.[10] Leanring and generalizaton results were compared with those of NETtalk2 and found to be close as much as 3% in generalization cases with a difference of only 5 or 10 epochs less during training (NETtalk2 even reported faster training and better generalization results in a small number of cases). This commecrial neural net (called F.A.S.T.) also did not converge on cases were NETtalk2 did not. This assured that NETtalk2 algorithm is working fine.

(10) F.A.S.T., developed by Lars Kindermann (kinderma@forwiss.uni-erlangen.de) at the University of Erlangen, Germany.

4.7.2 Using Multiple Hidden Layers in NETtalk2 with Arabic Text:

In other experiments we increased the number of hidden layers in NETtalk2.

In the first experiment, two hidden layers were used of sizes 40 (i.e. NETtalk2's topology was 231-40-40-8). In this case NETtalk2's performance was the same when it had one hidden layer of size 80, except that it took more processing time to finish one epoch and also took more epochs to learn. Generalization performance was almost the same.

In the second experiment, two hidden layers of sizes 80 were used (topology: 231-80-80-8). Compared with having only one hidden layer of 80 PEs, this network was somewhat better in generalization but about the same in absolute performance; in the sense that generalization performance increased by 5%, but learning took too much time to complete. For example, learning the 1024 corpus of continuous words took around 31500 epochs to complete, and generalization tests showed 72% correctness on novel words.

In the third Experiment, three hidden layers were used, each consisting of 80 PEs (topology: 231-80-80-80-8). Compared with having one hidden layer of size 80, this topology lead to very law successul rates in generalization and increased training epochs. For example, learning the 1024 corpus of continuous words took around 71200 epochs, and generalization tests showed 40% correctness on novel words.

In the fourth experiment, two hidden layers of sizes 120 units each, were utilized (topology: 231-120-120-8). In this experiment, generalization performance was 8% better than having one hidden layer of 80 PEs (units), but learning took around 50000 epochs on the average. For example, learning the 1024 corpus of continuous words took around 49200 epochs to complete, and generalization tests showed 76% correctness on novel words.

4.7.3 Configuring NETtalk2 as a Recurrent Neural Network for Pronouncing Arabic Text:

As NETtalk2 steps through the input text, seven graphemes at a time, a mechanism may be needed to allow it to remember the previous context (i.e. the previous seven characters it has already stepped through)

In an attempt to enable NETtalk2 to remember previous context when capturing the features of the current input during training, we changed NETtalk2's topology to be recurrent. We used the Jordan model (Rich *et al*, 1991).

In the Jordan model, output from the output layer is linked back to PEs in the hidden layer through a layer of *"Current Units"* as seen in Figure 4.4.

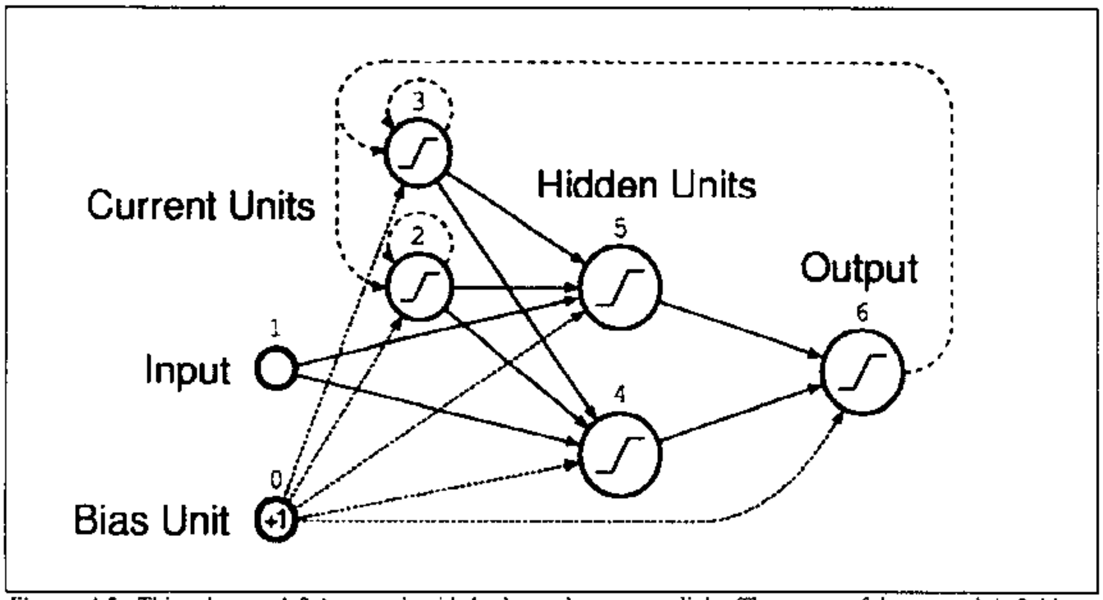

Figure 4.5: This shows a 1-2-1 network with Jordan style recurrent links. The output of the network is fed into the (two) current units directly without any weights (the dashed lines in the figure). The current units output's are fed into the hidden units and back into themselves. All current units are autorecurrent. Recurrent neural networks are characterized by their ability to remember, as the Current Units represent previous output of the net (Arras *et al*, 1996).

In its recurrent form, NETtalk2 has the same old topology (i.e. 231-80-8) plus a layer of 80 current units. The learning algorithm used is BP (sometimes we tried Rprop).

We trained this network on the continuous corpus of 1024 Arabic words. Using Rprop and a size-11 input window, it was able to learn the whole set in 30500 epochs. After that, a continuum of 300 words was presented for generalization, and NETtalk2 was able to generalize successfully on 77% of the input. We also trained NETtalk2 on the Arabic version of the children story *"Little Red Ridinghood"* (see section 4.7.1), it was able to learn it in 2130 epochs. The last two paragraphs of the story, which were not presented during training, were presented as a test of generalization. NETtalk2 was able to generalizae successfully on 78% of the input.

We can claim that, using recurrent topology with Rprop and an input window of 11 input groups, generalization power of NETtalk2 increases by about 12% than its default topology (section 4.2), although learning speed does not increase.

Chapter 5

Analysis of NETtalk2 Performance on Arabic Text

In this chapter we will talk about the performance of NETtalk2, and analyze it from input to output. We will also shed light on the reasons why results of NETtalk2 generalization on Arabic language are not very successful.

5.1 Analysis of the Input:

In the previous chapter we presented cases of NETtalk2 where different input window sizes were experimented with. For the sake of analysis and comparison with NETtalk, we will discuss NETtalk2 here using a 7-character-wide sliding window to capture the input. The analysis of this window below can be extended on other window sizes without modification.

Using this 7-character-wide sliding window, NETtalk2 utilizes the three characters at the left of the middle one, and the other three to its right as a surrounding environment to detect the appropriate pronunciation (phonological representation) of the middle character.

We agreed earlier to refer to this 7-character-wide window as size-7 window. We will represent it here as an input pattern that looks like:

- , - , - , # , - , - , -

Where # denotes the middle letter in the window, which is to be pronounced and assigned an appropriate phoneme as output; depending on its surrounding 6 letters.

Let us set # as a constant for it is the letter that will be pronounced, say the letter "ع".

Every other surrounding letter may be any of the 28 Arabic letters (left aside word boundaries, spaces, gemination and punctuation marks), there is also a possibility that any of the surrounding letters (especially the first one next to # on the left) is a space, denoted by $:

- , - , $, # , - , - , -

This represents the case when the middle letter in the window (the # here), occurs as the last letter at the end of an Arabic word, in a continuous text being stepped through the window.

Now, we have around 5 places (denoted by -) where each one can take any of the 28 Arabic letters, this yields a number of 28^5 = 17,210,368 different input patterns for the letter #. This is also the case in NETtalk, but in Arabic language a letter can also take on any of three shapes: at the beginning, middle or ending of a word. This adds more input patterns to the number above, about 28^5 x 3.

To overcome the problem of an input letter having more than one shape, the pre-processor accompanied with NETtalk2 (refer to section 4.3), will read the input Arabic text and turn all the different shapes of every one letter to one unique binary representation, to be input to NETtalk2. For example, take the letter "ع", in the following text:

عَلِمْتُ العِلْمَ يَنْفَعُ

it has three different shapes according to its place in a word (frontal "عـ"), (midial "ـعـ") and (final "ـع"). The pre-processor will read the text and convert all the "ع"'s that might come through into the pattern 00000000000000000100000000000000 which is used as input to NETtalk2 denoting the letter "ع" in any case. Same applies on all other letters of the alphabet.

As NETtalk2 learns, it passes through a learning set with different input patterns, requiring different output patterns, depending on different reasons discussed next.

5.2 Output Analysis and Thorough Investigation of NETtalk2:

After presenting NETtalk2 performance and generalization results in section 4.7, we noticed the low convergence speed of the learning algorithm (or non-convergence in some cases), and the low success percentage in generalization tests.

After analysing the results, we believe that the following features of Arabic text make it difficult for NETtalk2 to converge or to generalize. Examples and Experiments mentioned with each feature were conducted to support these beliefs:

1) A single letter can be represented to NETtalk2 in a large number of different input patterns (i.e. too many patterns for each letter): As shown in the previous section, a single letter in the middle of the input window may be surrounded by six different

letters in different contexts, leading to more than 28^5 (may be 28^6) number of input cases for each of the 28 Arabic letters, and more when using larger input windows.

This same case appears also in English text, but two points contribute to the fact that NETtalk2 (and so NETtalk) performance on English text is better than its performance on Arabic, the two points being:

i. The English alphabet contains only 26 letters, which makes the number of input patterns approximately 26^6, instead of the 28^6 patterns in Arabic, which has 28 letters in its alphabet. This less number of input patterns (cases) in English, turned out to be quiet enough for training and generalization as the experiments and results show in (Sejnowski *et al*,1992), who say: *"Learning was transferred to novel words even after a small sample of English words was presented."*

ii. A considerable number of English letters usually have the same pronunciation wherever they appear. Take for example the letter "n", in most training and test cases, it was found that this letter (and many others like it) has the same phoneme \n\(11) every time it appears. This makes it easier for NETtalk (and so NETtalk2) to easily capture this feature and successfully generalize on texts that have the letter "n" (and many others like it), because it always has the same output phoneme, which does not require that NETtalk\NETtalk2 be trained on a lot of different input patterns that "n" might appear in (see the material about how NETtalk generalizes in section 3.2.4).

(11) Phonetic representation in NETtalk2 follows that of Webster's Ninth New Collegiate Dictionary, MERRIAM-WEBSTER Inc., Publishers, USA, 1985.

In Arabic, a single letter has *at least* three different pronunciations: Front Low Unrounded, Front High Unrounded and Back High Rounded[12] . These pronunciations, which should be output by NETtalk2, depend on the context in which the letter appears. This context does not only include the nearby letters but could include the whole sentence in which the letter appears.

Consider this experiment: NETtalk2 was trained on a text where the letter "n" appears in different cases, and then a generalization test was conducted using words that contain the letter "n", but did not appear in the training text. Here is the text that NETtalk2 was trained on:

```
Nigel is nearly not into this nonsense
```

Table 5.1 shows generalization test strings presented to NETtalk2 after it was trained on the above text, and clarifies whether NETtalk2 was able to identify "n" as /n/ or not. Notice that "n" appears in different places in the strings used for the test.

Table 5.1: Identifying "n" as the phoneme \n\ during a generalization test on NETtalk2.

Input for generalization test	**Did NETtalk2 identify "n" as /n/ or not?**
Night	YES
bend	YES
burn	YES
sense	YES

[12] Front Low Unrounded مفتوح (َ), Front High Unrounded مكسور (ِ), Back High Rounded مضموم (ُ).

The same experiment was carried on the following Arabic text:

عَلِمْتُ العِلْمَ يَنْفَعُ

In the above text, the letter "ع" appears in three shapes (frontal "عـ"), (midial "ـعـ") and (final "ـع"). After NETtalk2 has learned this text, a generalization test was made (see Table 5.2) and it was supposed to identify "ع" as either \ʕa\ (front low unrounded), \?ʔu\ (back high rounded), or \ʕi\ (front high unrounded) according to its place in the input test string and from what it has learned. Unfortunately, In most cases it did not, even with an input window of size 11.

Table 5.2: Identifying "ع" correctly during a generalization test on NETtalk2.

Input for generalization test	**Did NETtalk2 identify "ع" correctly?**
عَشاء	NO
العِيد	NO
ينعَقد	NO
دافعْ	YES

2) <u>A single input pattern occurs more than once requiring a different pronunciation each time,</u> depending on the Arabic grammatical semantics resulting from its syntactic position in a statement.

This occurs in some cases in English, an example in English is the pronunciation of the letter "a", in some contexts it is pronounced as \a\, and in others (usually when there is an "e" at the end of a word) it is pronounced as \ā\.

A single Arabic letter might be: Front Low Unrounded, Front High Unrounded, Back High rounded, or Consonant[13]. Sometimes, in handwriting, these features are signified with unique marks typed above every letter (mentioned in Footnotes 12 and 13), so that the human reader can tell the correct pronunciation of a letter. But in our model, text is entered into the computer using a keyboard (the Standard 101/102-Key keyboard), which does not contain these marks, so there is no indication that an input letter is in any of these four features. This forces NETtalk2 to take in the same input pattern different times as it occurs, but each time, the middle letter in the window might have any of the four marks mentioned above, requiring different phonological representation (output) each time. This results in the following two problems:

i. To be able to capture the surrounding context of a letter, in order to assign to it the appropriate representation out of its four possible ones, a very large window size is needed. For a NN (here NETtalk2) to be able to pronounce the letter correctly, it should be trained with many different input patterns in which a letter (#) might appear, and their mappings with the appropriate output phonemes. This requires: (1) A large window to include surrounding *words* of the letter # -not only surrounding letters- which in its turn requires a big NN structure to capture the different mappings and save them as configurations (weights) in the hidden connections, (2) A very large training set to capture most of the cases, or at least the most frequent ones.

(13) Consonant ساكن (ْ).

A big structure and a large training set will most probably take a lot of working memory and processing time. In section 5.3 we will talk about using larger window sizes in training NETtalk2.

ii. A neural network might not learn Arabic text, that is; the learning algorithm might not converge, which actually happened during many of the learning attempts. We will come to this later and discuss it in more details in section 5.2.1

3) The pronunciation of one input letter in a pattern is usually represented by more than one output phoneme: The phonological representation of an Arabic letter, most of the time, involves the original phonological representation of the letter in its consonant case plus a vowel phoneme (a (َ), u (ُ) or i (ِ)), or a double vowel phoneme (an (ً), un (ٌ) or in (ٍ)). For example, the letter "م" in the word "عِلْمٌ" is pronounced as \mun\.

This case of more than one phoneme representing one grapheme, occurs in English with letters like "x", whose phonetical representation involves two phonemes, \ks\.

It is known that a neural network takes only one input at a time at the input layer and gives only one corresponding output (one-to-one mapping). This problem of one input requiring a combination of output phonemes, can be overcame by expressing each of these phonemic combinations as one new single phoneme. For example, the letter "ب" can have any of the following phonemes:

\b\, \bu\, \ba\, \bi\, \ban\, \bun\, \bin\, ... and so for other letters of the Arabic alphabet....

Each one of these is considered as a single phoneme that is represented by a unique binary pattern at the output layer and in training pairs, they are part of the set of phonemes used by NETtalk2 and are implemented as single phonemes (so the word "عِلْمٌ" is represented as \ ʕilmun\). As we can see, this increases the number of phonemes considered by NETtalk2, which requires enlarging the training set adding more overhead to the training process in NETtalk2.

4) <u>Some input patterns are not pronounced:</u> This happens when there is a pattern at the input layer, but NETtalk2 is not required to yield an output. The most commonly occurring input pattern is "ال" as in "الشَّمْسُ" (ال-الشمسيه), where the letter "ل" should not be pronounced. Another example, the word "الدَّرب", its pronunciation is \ ʔaddarb\, the \d\ appears twice, because "د" is stressed, but the "ل" did not appear at all. To overcome this problem, we insert a dummy character at the input string (in the training vector) to cover for the absence of a phoneme that should correspond to the input "ل", so \ʔaddarb\ will be \ʔa▪ddarb\. Also notice in this particular example, that the second \d\ corresponds to the gemination mark "ّ", which can be typed with the input text by any standard Arabic keyboard. The dummy character (▪) affects the training process, in the sense that NETtalk2 trains on it like it trains on spaces between words, but it does not have a sound effect when the output sound file is generated, it is just skipped.

This case occurs in English text too, with combinations like, "ch, sh, th, tion,...etc", where a combination of letters is pronounced by only one phoneme, or a number of phonemes less than the number of input letters. In such cases, we say that a phoneme corresponds to a cluster of letters.

In NETtalk, when a phoneme corresponds to a cluster of letters, the phoneme is most closely associated with the first letter of the cluster, and is, accordingly, placed in the same position as the first letter. Hyphens (instead of the dummy characters we use with NETtalk2) are placed in the remaining positions. In some cases there is a letter common to many or all of the letter clusters associated with a particular phoneme, and the letter does not come first in some of the clusters. In these cases the phoneme is placed in a position corresponding to that letter, and the other letter positions are filled with hyphens. For example, the sound \n\ is associated with the letters "nd." This is coded as \n-\, with the \n\ in the same position as the letter "n," and the hyphen in the position of the "d." The \n\ sound is also written as "gn," "sn" and "kn". Here, the \n\ phoneme is associated with the common "n," not the differing initial letters, so all of these graphemes are coded as \-n\, where the \n\ is placed opposite the second letter "n" instead of the letter that varies.(14)

5) <u>There is no input, but the NN is required to yield a phoneme</u>, usually an elongation phoneme. Some examples are the words " هٰذه، هٰذا، الرحمٰن، لٰكن." Notice the elongation mark (ٰ), It can't be typed into an Arabic text by the standard keyboard, but is pronounced as \a:\ and a phoneme should be given as output by NETtalk2. In this case, we cannot help but write the text as is, and ignore the elongation. It will not be pronounced because there is no indication of its existence, which makes NETtalk2

(14) Reference from The Internet, http://herens.idiap.ch/~miguel/dbases/nettalk/nettalk.info

present wrong pronunciation of the whole word, contributing more to bad generalization results.

We looked for this case in English language, and were able to find a close case (not a very similar one). From the Internet[14], Scott Fahlman says about NETtalk:

"Some words used phonemes that were not matched by any letters in the word. The word 'eighth' is an example. 'Eighth' is phonemically coded by Webster's as /etT/, but there is no clear letter for the /t/. The dictionary uses /e---T-/, dropping the /t/." (by "Dictionary" here he means the set of phonemes used by DECtalk and NETtalk)

5.2.1 Non-convergenve in NETtalk2:

The five points presented in the previous section, are the main and most important reasons, we believe cause bad generalization results, and in some cases non-convergence in NETtalk2.

For an example of the case where an input pattern occurs more than once in different contexts, and requires different pronunciation (output phoneme) each time. Consider the letter "ب" in the word "كتب" in the following example:

a) كَتَبَ ابن خلدون الكثير

b) كُتُبُ ابن خلدون كثيرة

When the letter "ب" is the middle letter of the size-7 input window, the window will look like this:

a) $, كَ , تَ , بَ , $, اِ , بْ

b) $, كُ , تُ , بُ , $, اِ , بْ

In this example, there is only one input pattern. This pattern occurred twice, (a) and (b), but in (a) the output should be the phoneme /ba/, while in (b) it should be the phoneme /bu/. The Arabic language contains many cases like this. To mention some:

سَلَطَهُ، سُلْطَهْ

دَفَعَ، دَفْعٌ

أَدْفَعُ، إِدْفَعْ

نَظَرَ، نَظَرٌ

As we all know, we can train a typical NN to yield 1 as output for an input of value 1, or 0 as output, but not the two outputs for the same input. When this occurs in a training set, it is obvious that a learning algorithm will not converge, this is exactly what the previous example is all about. It also occurs in Arabic in cases like this example:

a) ضَرَبَ الولدَ

b) ضُرِبَ الولدُ

In either pattern the input to NETtalk2 is the same when the letter "ر" is the one to be pronounced:

أ , $, ب , ر , ض , $, $

But the phonological output should be /ra/ in case (a) and /ri/ in case (b). Same also applies for the letter "ض" in the same example, and for the letter "ع" in this exmaple:

a) صَعْبٌ الدرسُ

b) صَعُبَ الدرسُ

Another example, notice the pronunciation of the letter "ع" in the word "بديع" in these patterns:

1) نَظَرَ بديعٌ من النافذه

2) فحصتُ نَظَرَ بديعٍ من النافذه

3) نظرتُ بديعًا من النافذه

The pronunciation of "ع" was different every time depending on the grammatical semantics resulting from the syntactic position of the word "بديع", which was subject in the first pattern, and object in the third one. Such grammatical and semantical information cannot be explicitly encoded in the input to NETtalk2, nor can it be captured by the size-7 window, as in the second pattern, the pronunciation of "ع" depended on its 9th right-most neighboring letter. A larger window size reduces the effect of this problem; because it reduces the number of similar input patterns introduced to NETtalk2, requiring different outputs each time, and thus leads to convergence.

One might think of ignoring the fact that an Arabic letter has four different pronunciations, and thus reducing the effect of this problem on the learning process, by trying to train NETtalk2 on a text and requiring it to learn on all the letters in this text being only consonants.

Below, are examples of experiments that were conducted in which NETtalk2 was trained to pronounce a given Arabic text in all consonants letters, generalization performance was measured, and then NETtalk2 was trained on the same text but with the correct pronunciation, also generalization performance was measured then and the two cases compared. Comparison revealed that, NETtalk2 takes less training epochs on the same text when it is in consonants form (about 50% less training epochs), and makes about 30% better generalization results.

Example A7: In this experiment the following text was presented to NETtalk2 twice, once trained for an output of all consonant letters, and the next time with correct pronunciation:

محمدٌ طالبٌ مجتهدٌ. يقوم محمدُ كل يوم في الصباح الباكر، لكن البارحه جاءت أم محمدٍ الى غرفة محمد متـــــأخره، و

حينما قام محمدٌ من النوم كان غاضباً جداً

Table 5.3 summarizes generalization results when the net was trained on the text with all consonant letters:

Table 5.3 Generalization tests when NETtalk2 was trained on the text in example A7 as all consonant letters.

Test string	Correctness percentage in output phonemes	Correctness percentage in representing the phoneme for "د"
طالب مجتهد	87%	-
طالب مجتهد	87%	-
يقوم محمد كل يوم	80%	100%
يقوم محمد الى الصلاه	50%	100%
قام محمد من النوم	97%	100%
قام محمد من السرير	60%	100%

While Table 5.4 shows the results with the same generalization test, but when NETtalk2 was trained on the text with correct pronunciation.

Table 5.4 Generalization tests when NETtalk2 was trained on the text with correct pronunciation.

Test String	Correctness percentage in output phonemes	Correctness percentage in representing the phoneme for "د"
طَالِبٌ مُجتَهِدٌ	50%	-
طَالَبَ مُجْتَهَدٍ	0%	-
يقوم محمدٌ كل يوم	57%	100%
يقوم محمدُ الى الصلاه	28.6%	0%
قام محمدُ من النوم	64.7%	100%
قام محمدٌ من السرير	50%	100%

Example A8: We repeated the experiment in example A7 with the following text:

نَظَرَ بديعٌ من النافذه عندما كنت قادماً، ألحت بيدي لبديعٍ كي يفتح البابَ، سلمت على بديعٍ. عندما جلسنا أعطيت بديعَ كتابه و شكرته.

Table 5.5 summarizes generalization results when the net was trained on this text as all consonant letters, while Table 5.6 shows the results with the same generalization test, but NETtalk2 was trained on the text with the correct pronunciation.

Table 5.5 Generalization tests when NETtalk2 was trained on the text in example A8 as all consonant letters.

Input String	Correctness percentage in output phonemes	Correctness percentage in representing the phoneme for "ع"
فحصت نظر بديع من النافذه	70%	100%
نظرت بديع من النافذه	95%	100%
قدم بديع ملوحا	33%	100%
لبديع سياره جميله	40%	100%
منظر جميل	90%	-

Table 5.6 Generalization tests when NETtalk2 was trained on the text in example A8 with the correct pronunciation.

Input String	Correctness percentage in output phonemes	Correctness percentage in representing the phoneme for "ع"
فحصتْ نَظَرَ بديعٍ من النافذه	48%	0%
نظرتٌ بديعٌ من النافذه	20%	0%
قَدِمَ بديعٌ ملوحاً	0%	0%
لبديعٍ سيارة جميله	0%	0%
منظرٌ جميل"	3%	-

5.3 Using Different Numbers of Input Groups in NETtalk2:

We saw in Chapter 4 how different input window sizes were used for training NETtalk2 on Arabic text. Results in Chapter 4 revealed that generalization performance increases with increasing the window size and saturates at size 11 (and 13 sometimes), then it starts to decrease dramatically. Nevertheless, learning speed increases linearly as the window size increases.

When the input window size increases, NETtalk2 is able to capture more features of the current context, helping it to identify different graphemes during generalization. This is why generalization performance becomes better with increasing window size. But, when the window size becomes larger and larger (15 and more) too much context is presented in a small number of smaples, and features representations are scattered in different places all over the hidden connections among PEs, in the *now* large structure. This weakens the classification power of the net and thus its ability to identify different and unique graphemes (input patterns), which results in bad generalization performance. This problem also starts to appear when NETtalk2 has more than two hidden layers (see section 4.7.2).

One good advantage of large window sizes, is that, occurance of the same input pattern more than once, requiring different output each time, no longer gets NETtalk2 stuck in some local minima (and thus not able to learn). Simply, because this input pattern is no longer the same in each occurance; due to the fact that a large window captures also more surrounding features which are different in each occurance, canceling the uniqueness of this input pattern. Consider the following example:

a) ضَرَبَ الولدُ

b) لقد ضُرِبَ الولدُ

Using an input window of size 11:

a) و . ل , ا , $, ب , ر , ض , $, $, $, $

b) و , ل , ا , $, ب , ر , ض , $, د , ق , ل

The input pattern in (a) is different than the one in (b). In this case, NETtalk2 will be able to converge but will take too much time climbing out of the small local mimina, which is the result of the overwhelming resemblance between the two patterns, for they are different in only three input groups out of eleven.

Another example is in the patterns:

a) نظر بديعٌ من النافذه

b) فحصت نظر بديعٍ من النافذه

To correctly identify each one as a unique pattern to pronounce the letter "ع", the input window should be able to capture the "ت" in the second pattern, which is the 9th right-most letter to "غ", and this requires the input window to be of size 19 at least. But, as we described before, unfortunately, with such large sizes, the generalization performance degrades dramatically.

Chapter 6

Conclusion and Future Work

6.1 Summary and Conclusion:

In this thesis, we constructed a NETtalk-like neural network, called NETtalk2. We trained NETtalk2 on pronouncing English text, in order to compare its performance with that of NETtalk as reported in (Sejnowski *et al*, 1992), then we ran NETtalk2 on Arabic text and compared the results with those of English. We analysed the input, the output and representation in NETtalk2, and were able to identify reasons behind its failure to generalize reasonably and sometimes even to learn to pronounce Arabic text.

NETtalk2 can be described as 231-80-8 MLP (MultiLayer Perceptron) artificial neural network, that uses supervised error-backpropagation learning (BP). Like NETtalk, NETtalk2 utilizes a sliding input window that can read seven groups of graphemes (in other experiments the number was varied from 3 to 41) at one time, then it generates an appropriate phoneme and voice corresponding to the middle grapheme in the input window, utilizing the other six graphemes as context.

NETtalk2 was trained on English text in two forms (as was NETtalk):

1. A corpus of words: Each word was presented separately to NETtalk2. It was able to learn the 1000 most commonly used English words in around 7000 training epochs. NETtalk2 was then passed on a continuum of 10,000 words, and the correct output phoneme was generated in 75% of the cases. When the 10,000 words were added to

training, generalization performance jumped to 90% after the first 100 epochs of training.

2. A corpus of continuous text: NETtalk2 was trained on a continuous text of 1024 words, then passed over a continuation of 500 words and was able to generalize successfully on 76% of the cases.

In either of the two forms, other tests were made with a different input window size each time. The size varied from 3 groups to 11. Usually generalization performance increased by no more than 7% with increasing window size, accompanied of course by increase in the learning speed, but generalization starts to deteriorate after using 13 input groups and more.

The above two forms and other experiments with NETtalk2 on English text, can be found in details in section 4.6.

Training NETtalk2 on Arabic text was mainly also in two forms:

1. A corpus of words: A set of 1024 separate Arabic words was chosen from everyday use. NETtalk2 was not able to learn on the corpus and was stuck after 3000 epochs, with the average squared error around 0.55, momentum value of 0 and learning rate 0.35, using the BP learning algorithm.
2. A corpus of continuous text:
   - NETtalk2 was trained on the same set of 1024 commonly used words from point 1 above, but as a continuous text and was not able to converge, even when changing the learning rate and the momentum.
   - An Arabic novel by Gibran Kahlil Gibran was passed through NETtalk2 for learning. NETtalk2 did not learn it and got stuck in a local minima.

- Two paragraphs from an essay in the daily newspaper were passed for learning, NETtalk2 was able to learn them in 1700 epochs, the third paragraph was then used to test generalization performance of the net, which did not exceed 30%, but when this paragraph was added to the training set, performance jumped to 70% after another 500 epochs.

For details on these experiments, refer to section 4.7.1.

Other experiments of NETtalk2 on Arabic text involve:

1- Training NETtalk2 on the Arabic alphabet using BP with learning rate of 0.35 and varying the momentum α from 0 to 0.4. The value of 0.4 for the momentum was found to be a very good choice for fast learning. Increasing this value to more than 0.4 would lead to over-training (refer to example A1 and Figure 4.2).

2- NETtalk2 was also trained using different window sizes, ranging from 3 to 41 input groups. Experiments in section 4.7, show small increase in generalization performace and training speed with increasing the input window size. When the window size was of 3 input groups, NETtalk2 could not converge. As this window size increased so did the leanring speed. Generalization performance also increased as the window size increased from 5 to 11, becuase NETtalk2 was able to capture more surrounding context with more input groups, which lead to better identification and generalization power. However, this power starts to deteriorate when increasing the window size more than 13 input groups; due to the large amount of features presented to NETtalk2 that require classification compared to the small number of samples (input patterns) in which these features were presented.

3- Experiments with different learning rates were also conducted, revealing that a value of 0.35 is a good choice for fast learning, as NETtalk2 starts to shoot off into divergence after a value of 0.55 (refer to Table 4.9 and Figure 4.3).

4- Training NETtalk2 on simple Arabic children stories also did not work using BP, as it did not converge (refer to Example A4 in Chapter 4). When the input window size was increased to 9 input groups, convergence took place but generalization power was weak (about 30%).

5- Other attempts were made on training NETtalk2 on a text of all consonant letters, very good results appeared, but we cannot consider this as a neural net that can pronounce Arabic text (refer to section 5.2.1).

6- Training NETtalk2 using multiple hidden layers. We tried:

- Two hidden layers of 40 units each: Performance was the same as with one hidden layer of 80 units.
- Two hidden layers of 80 units each: 5% better generalization, but about the same in absolute performance; because learning time increased.
- Three hidden layers of 80 units each: Worst than the previous one; bad generalization performance and very low learning speed.
- Two hidden layers of 120 units each: 8% better generalization, but very slow learning.

7- Using recurrent topology in training NETtalk2 (Jordan style) with Rprop as the training algorithm. In this case, generalization power of NETtalk2 improved by 12%, although the learning speed stayed about the same and did not imporve.

Details on these experiments and others more, can be found in Chapters 4 and 5. These experiments were also conducted using Rprop as a learning algorithm with NETtalk2, which showed slightly better generalization results and learning speed, as discussed in the previous chapters.

After further investigation and analysis, we were able to conclude and identify the following crucial problems, that we believe make a NETtalk-like artificial neural network unable to pronounce Arabic text (for either it can not learn it, or generalization tests fail if learning did happen):

1- A single Arabic letter can appear in a large number of different input patterns. The net has to learn to identify this letter in whichever input pattern it might appear in. The following fact contributes more to making this a difficult process: A single Arabic grapheme is, most of the time, pronounced by a consonant phoneme that corresponds to it, combined with either of three vowels (a,u,i) and thus has at least three phonological representations as output, which adds more to the training patterns that must be presented to the net, and so requiring more training time and a larger structure to be able to capture the different features and save them as configurations in the hidden connections of the net.

2- A single input pattern can occur more than once in the same text, requiring a different output for each occurrence. In this case, NETtalk2 cannot learn, but diverges in an infinite number of loops. In many cases, very similar input patterns require totally different output patterns. Here, NETtalk2 falls into a local minima and takes a lot of learning time (and iterations) to climb out of the ravine. This problem disappears as larger input window sizes are used during training.

3- Some graphemes do not require a pronunciation in some cases, while they might in others. This is a case where there is an input to NETtalk2, but it should not give any output.

4- There exist cases where there is no input to the net, but an output is required, usually a phoneme representing elongation in voicing. Such cases cannot be captured by NETtalk2, thus leaving it to generate wrong output, corresponding to input patterns in which this case appears.

6.2 Future Work:

It has been shown in this study that a NETtalk-like neural network is unable to pronounce Arabic text to a regardable amount of accuracy.

Starting from this, one might think of using other learning algorithms than the one used by NETtalk. A learning algorithm called "Resilient Backpropagation" (Rprop) was presented earlier in the text, experiments with Rprop showed slightly better learning speed and generalization results than BP, but did not yet yield a NETtalk-like neural network that can pronounce Arabic text successfully.

Another good solution is to combine an Expert System (ES) or a knowledge-based system with the neural network. This can be implemented when we have explicit rules that govern the pronunciation of certain graphemes according to some rules of syntax. In this case, the ES can take a considerable amount of load off the neural net, by solving pronunciation cases that are governed by these rules, cases that are still not clear can be left for the neural network component to solve. This will reduce the number of patterns and cases given to the neural network for training. This Expert System (or Front

Processor) component is a very good and easy solution to languages, whose most of its grammar and pronunciation can be governed by rules of syntax.

References

(**Arras** ***et al*****, 1996)** Arras, M. K. and Mohraz, K. 1996. *FAST v2.2 - FORWISS Artificial Neural Network Simulation Toolbox, 1st edition.* FORWISS, Erlangen, Germany.

(Barry, 1985) Barry, W. J. 1985. Automatic Identification of Regional Accent: Theory and Practice. *Cambridge Papers in Phonetics and Experimental Linguistics 4*, Cambridge. UK.

(Campbell, 1993) Campbell, N. A. 1993. *Biology, 3rd edition.* The Benjamin/Cummings Publishing Company, Inc., Canada.

(Church, 1983) Church, K. W. 1983. *Phrase-Structure Parsing: A Method for Taking Advantage of Allophonic Constraints.* PhD Dissertation, Distibuted by IULC, USA.

(Fahlman, 1988) Fahlman, S. E. 1988. An Empirical Study of Learning Speed in Back-propagation Networks. *Technical Report, CMU-CS-88-162*, Carngie-Mellon University.

(Fletcher, 1980) Fletcher, R. 1980. *Practical Methods of Optimization, 1st edition.* John Wiley, NY.

(Grossberg ***et al*****, 1992)** Grossberg, S. and Carpenter, G. 1992. A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. In: Lau, C. (editor), *Neural Networks - Theoritical Foundations and Analysis.* IEEE Press, New York.

(Hinton *et al*, 1990) Hinton, G. E. Lang, K. J. and Waibel, A. H. 1990. A Time-Delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks*, Vol.3. Pergamon Press plc.

(Hoequist, 1987) Hoequist, C. Jr. 1987. Phonological Rules and Speech Recognition. In: Laver, J. and Jack, M. A. (editors), *European Conference on Speech Technology*, Vol.1, Edinburgh, pp. 480-483.

(Hopfield, 1982) Hopfield, J. J. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proc. Natl. Academic Society, Vol. 79, pp. 2554-2558, USA.

(Jacobs, 1988) Jacobs, R. 1988. Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks*, Vol. 1.

(Kohonen, 1988) Kohonen, T. 1988. The "Neural" Phonetic Typewriter. *IEEE Computer Magazine*, Mar., pp. 11-12.

(Kohonen, 1990) Kohonen, T. 1990. The Self-Organizing Map. *Proc. IEEE*, 78 (9): 1464-1480.

(Lau, 1992) Lau, C. 1992. *Neural Networks - Theoritical Foundations and Analysis, 1st edition*. IEEE Press, New York.

(McCulloch *et al*, 1943) McCulloch, W. S. and Pitts, W. 1943. A Logical Calculus of the Ideas Imminent in Nervous Activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-113.

(Minsky *et al*, 1969) Minsky, M. and Papert, S. 1969. *Perceptrons, 1st edition*. MIT Press, Cambridge, MA.

(Nelson *et al*, 1993) Nelson, M. M. and Illingworth, W. T. 1993. *A Practical Guide to Neural Nets, 5th edition*. Addison-Wesley Publishing Company, Inc., USA.

(Rajouani *et al*, 1987) Rajouani, A. Najim, M. Chiadmi, D. and Zyoute, M. 1987. Synthesis-By-Rule of Arabic Language. In: Laver, J. and Jack, M. A. (editors), *European Conference on Speech Technology*. Vol. 1, Edinbrugh, pp. 29-32.

(Reynolds *et al*, 1995) Reynolds, S. B. Mellichamp, J. M. and Smith, R. E. 1995. Box-Jenkins Forecast Model Identification. *AI Expert*,10 (6):15-28.

(Rich *et al*, 1991) Rich, E. and Knight, K. 1991. *Artificial Intelligence, 2nd edition*. McGraw-Hill Inc. USA.

(Riedmiller *et al*, 1993) Riedmiller, M. and Braun, H. 1993. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In: Ruspini, H. (editor), *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*,pp. 586-591, USA.

(Riedmiller, 1994 a) Riedmiller, M. 1994. Advanced Supervised Learning in Multilayer Perceptrons - From Backpropagation to Adaptive Learning Algorithms. *Int. Journal of Computer Standards and Interfaces, Special Issue on Neural Networks*, 16: 265-278.

(Riedmiller, 1994 b) Riedmiller, M. 1994. Rprop - Description and Imlementation Details. *Technical Report, W-76128 Karlsruhe FRG*, University of Karlsruhe.

(Ritchie *et al*, 1986) Ritchie, G. D. Black, A. W. Pulman, S. J. and Russel, G. J.1986. *The Edinburgh/Cambridge Morophological Analyser and Dictionary System (Prototype: Version 2.2) User Manual.* Cambridge University Computer Laboratory, UK.

(Rosenblatt, 1959) Rosenblatt, R. 1959. *Principles of Neurodynamics, 1st edition.* Spartan Books, New York.

(Rumelhart *et al*, 1986) Rumelhart, D. E. Hinton, G. E. and Williams, R. J. 1986. Learning Internal Representations by Error Propagation. In: Rumalehart, D. E. and McClelland, J. L. (editors), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, 1st edition*, Vol.1: Foundations. MIT Press, Cambridge, MA.

(Sejnowski *et al*, 1987) Sejnowski, T. J., and Rosenberg, C. R. 1987. NETtalk: A Parallel Network that Learns to Read Aloud. *Technical Report, JHU/EECS-86/01*. The John Hopkins University of Electrical Engineering and Computer Science.

(Sejnowski *et al*, 1992) Sejnowski, T. J., and Rosenberg, C. R. 1992. Parallel Networks that Learn to Pronounce English Text. In: Lau, C. and Sánchez-Sinencio, E. (editors), *Artificial Neural Networks - Paradigms, Applications and Hardware Implementations*. IEEE Press, New York.

(Simpson, 1992) Simpson, P. K. 1992. Foundation of Neural Networks. In: Lau, C. and Sánchez-Sinencio, E. (editors), *Artificial Neural Networks - Paradigms, Applications and Hardware Implementations*. IEEE Press, New York.

(Vander *et al*, 1994) Vander, A. J. Sherman, J. H. and Luciano, D. S. 1994. *Human Physiology. The Mechanism Of Body Function, 6th. edition*. McGraw-Hill, Inc., USA.

(Watrous *et al*, 1987) Watrous, R. L. Shastri, L. and Waibel, A. H. 1987. Learned Phonetic Discrimination Using Connectionist Networks. In: Laver, J. and Jack, M. A. (editors), *European Conference on Speech Technology,* Vol.1, Edinburgh, pp. 377-381.

(Widrow *et al*, 1960) Widrow, B. and Hoff, M. E. 1960. Adaptive Switching Circuits. *IRE WESCON Conv. Record*, Part 4, pp. 96-104.

(Yuhas *et al*, 1992) Yuhas, B. P. Goldstein, M. H. Jr. Sejnowski, T. J. and Jenkins, R. E. 1992. Neural Network Models of Sensory Integration for Improved Vowel Recognition. In: Lau, C. (editor), *Neural Networsk, Theoritical Foundations and Analysis*. IEEE Press, NY, pp. 311-320.

Appendix A

Set of Phonemes used by NETtalk2

Decimal Representation	Phonological Representation	Example
0	/?a/	أ
1	/?u/	ؤ
2	/?i/	ئ
3	/?/	ء
4	/a:/	ا
5	/ba/	بَ
6	/bu/	بُ
7	/bi/	بِ
8	/b/	بْ
9	/ta/	تَ
10	/tu/	تُ
11	/ti/	تِ
12	/t/	تْ
13	/tha/	ثَ
14	/thu/	ثُ
15	/thi/	ثِ
16	/th/	ثْ
17	/3a/	جَ
18	/3u/	جُ
19	/3i/	جِ
20	/3/	جْ
21	/hha/	حَ
22	/hhu/	حُ
23	/hhi/	حِ
24	/hh/	حْ
25	/xa/	خَ
26	/xu/	خُ
27	/xi/	خِ
28	/x/	خْ
29	/da/	دَ
30	/du/	دُ
31	/di/	دِ
32	/d/	دْ
33	/?a/	ذَ
34	/?u/	ذُ
35	/?i/	ذِ
36	/?/	ذْ
37	/ra/	رَ

38	/ru/	رُ
39	/ri/	رِ
40	/r/	رْ
41	/za/	زَ
42	/zu/	زُ
43	/zi/	زِ
44	/z/	زْ
45	/sa/	سَ
46	/su/	سُ
47	/si/	سِ
48	/s/	سْ
49	/$a/	شَ
50	/$u/	شُ
51	/$i/	شِ
52	/$/	شْ
53	/Sa/	صَ
54	/Su/	صُ
55	/Si/	صِ
56	/S/	صْ
57	/Da/	ضَ
58	/Du/	ضُ
59	/Di/	ضِ
60	/D/	ضْ
61	/Ta/	طَ
62	/Tu/	طُ
63	/Ti/	طِ
64	/T/	طْ
65	/?da/	ظَ
66	/?du/	ظُ
67	/?di/	ظِ
68	/?d/	ظْ
69	/¿a/	عَ
70	/¿u/	عُ
71	/¿i/	عِ
72	/¿/	عْ
73	/µa/	غَ
74	/µu/	غُ
75	/µi/	غِ
76	/µ/	غْ
77	/fa/	فَ
78	/fu/	فُ
79	/fi/	فِ
80	/f/	فْ
81	/qa/	قَ
82	/qu/	قُ
83	/qi/	قِ

84	/q/	قْ
85	/ka/	كَ
86	/ku/	كُ
87	/ki/	كِ
88	/k/	كْ
89	/la/	لَ
90	/lu/	لُ
91	/li/	لِ
92	/l/	لْ
93	/ma/	مَ
94	/mu/	مُ
95	/mi/	مِ
96	/m/	مْ
97	/na/	نَ
98	/nu/	نُ
99	/ni/	نِ
100	/n/	نْ
101	/ha/	هَـ
102	/hu/	هُـ
103	/hi/	هِـ
104	/h/	هْ
105	/wa/	وَ
106	/wu/	وُ
107	/wi/	وِ
108	/w/	وْ
109	/ja/	يَ
110	/ju/	يُ
111	/ji/	يِ
112	/j/	يْ
113	/ /	SPACE
114	/,/	COMMA
115	/?un/	ءٌ
116	/?in/	ءٍ
117	/?an/	ءً
118	/bun/	بٌ
119	/bin/	بٍ
120	/ban/	بً
121	/tun/	تٌ
122	/tin/	تٍ
123	/tan/	تً
124	/thun/	ثٌ
125	/thin/	ثٍ
126	/than/	ثً
127	/3un/	جٌ
128	/3in/	جٍ
129	/3an/	جً
130	/hhun/	حٌ

131	/hhin/	حٍ
132	/hhan/	حً
133	/xun/	خٌ
134	/xin/	خٍ
135	/xan/	خً
136	/dun/	دٌ
137	/din/	دٍ
138	/dan/	دً
139	/?un/	ذٌ
140	/?in/	ذٍ
141	/?an/	ذً
142	/run/	رٌ
143	/rin/	رٍ
144	/ran/	رً
145	/zun/	زٌ
146	/zin/	زٍ
147	/zan/	زً
148	/sun/	سٌ
149	/sin/	سٍ
150	/san/	سً
151	/$un/	شٌ
152	/$in/	شٍ
153	/$an/	شً
154	/Sun/	صٌ
155	/Sin/	صٍ
156	/San/	صً
157	/Dun/	ضٌ
158	/Din/	ضٍ
159	/Dan/	ضً
160	/Tun/	طٌ
161	/Tin/	طٍ
162	/Tan/	طً
163	/?dun/	ظٌ
164	/?din/	ظٍ
165	/?dan/	ظً
166	/¿un/	عٌ
167	/¿in/	عٍ
168	/¿an/	عً
169	/µun/	غٌ
170	/µin/	غٍ
171	/µan/	غً
172	/fun/	فٌ
173	/fin/	فٍ
174	/fan/	فً
175	/qun/	قٌ
176	/qin/	قٍ

178	/kun/	كٌ
179	/kin/	كٍ
180	/kan/	كً
181	/lun/	لٌ
182	/lin/	لٍ
183	/lan/	لً
184	/mun/	مٌ
185	/min/	مٍ
186	/man/	مً
187	/nun/	نٌ
188	/nin/	نٍ
189	/nan/	نً
190	/hun/	هـٌ
191	/hin/	هـٍ
192	/han/	هـً
193	/wun/	وٌ
194	/win/	وٍ
195	/wan/	وً
196	/jun/	يٌ
197	/jin/	يٍ
198	/jan/	يً
199	/■/	Dummy character, will not be spelled
200	/:/	Repeats previous character. Represents gemenation mark.
201	/E/	banana
202	/Er/	bird
203	/a/	map
204	/a:/	day, fade
205	/au/	now, out
206	/b/	rib
207	/ch/	chin
208	/d/	did
209	/e/	bed
210	/ē/	beat, bleed
211	/f/	fast
212	/g/	go
213	/h/	ahead
214	/i/	tip
215	/ī/	side
216	/j/	job
217	/k/	cook

222	/o:/	bone, know
223	/o/	saw, all
224	/oi/	coin
225	/p/	lip
226	/r/	rat
227	/s/	source
228	/sh/	shy, mission, special
229	/t/	late
230	/th/	thin
231	/TH/	this
232	/u/	rule, youth
233	/v/	vivd
234	/w/	we, away
235	/y/	yard, young, few
236	/z/	zone, raise
237	/zh/	vision
238	/ks/	excess, axe
239	/U/	pull, book, wood
240	/yu/	union

Appendix B

Format of the 8-bit mono .wav Sound Files

In this appendix, we will talk about our personal experience in trying to detect the format of 8-bit mono .wav sound file (which we will refer to as "wave files" throughout the text).

Wave files have different formats ranging from 8-bit mono to 16-bit stereo and PCM. We were able to figure out the format of the 8-bit mono files, which is quiet good enough for our study. This happened when we were trying to copy waves and paste them in new files using the WaveStudio by Creative Technologies. It was found that when trying to concatenate two wave files (16-bit stereo) into one file, if you open a PCM or 16-bit stereo wave file, select the part you want from it (or even all of it), copy it, and paste it in a new file; it just won't work, it'll appear as sound distractions and the voice will sound very weird.

Nevertheless, it worked with 8-Bit Mono files. So, we made the following small trick.

First, to find out how long the header of a wave file is and if it has a trailer or not. We made a small PASCAL program that will read a wave file as text file (character by character) as ASCII, finds the ordinal of every character (it's decimal ASCII number), and then passes it to the Sound() function, it worked fine. So, it seems that the sound frequencies (in integer decimal values) are transformed during recording of a sound into their respective "decimal-to-ASCII" characters, and then written to the file after a specific header, after all this forms a wave file.

So, we know now that voice is represented as frequencies in character values (notice that each character is one byte). So, how can we extract the header?

After different trials of the above-mentioed program on different ready-made wave files (all of the same type, 8-bit mono), we noticed that they all begin with the same wave form. When we checked their wave files (opened them as text files), it was noticed that they all share the same first 41 characters, which obviously is the header.

An important notice we would like to mention here, is that the spaces (7-8,17-19,22,24,27-28,31-32,34,36 characters in the header) are not regular spaces (made by the space bar), but are the ASCII code of the decimal number 0.

Now, the next 4 characters (bytes or fields) after the header are very important, they represent the length (amount) of data in the file.

In our thesis we need to be able to concatenate many wave file into one large file, so what length should we put in the length field (after the header). What we did next is a program that will take each wave file, reads the length field of its header (ignore all the rest of the header), transforms it to its decimal value, adds it to an accumulator, and appends the data part of the file to a big "Master file"(this will be the result wave file at the end), and so on. At the end, the value in the accumulator will be the sum of lengths of all the wave files, and then the usual single header will be given to this Master file, followed by the ASCII value of the accumulator.

During this process we noticed that, the wave files that we deal with have roughly the same size, they are all letters of the Arabic alphabet. If the NUMBER of the wave files is more than 13, another field after the length field is used with the ASCII value of the decimal number 1.

If the number of wave files is greater than 23 then it is filled with the ASCII value of the decimal 2, and so on.

The following PASCAL code represnts this idea. Notice that this code comes after the header has been written to the Master wave file (MASTERWAVFILE) and the length field has been specified and written:

```
if WAVE_COUNTER>13 then
  if (WAVE_COUNTER mod 10)>0.3 then
    write(MASTERWAVFILE,chr(trunc(WAVE_COUNTER/10)));
   else
    write(MASTERWAVFILE,chr(trunc(WAVE_COUNTER/10)-1)); {Here it is in the
 else                                                    previous phase}
  write(MASTERWAVFILE,chr(0));
```

Where the trunc() function returns the integer part of a real number and the chr() function returns the ASCII code of its argument. We call these blocks in the code *"Phases"*.

The same process happens for bigger wave files, i.e. the next 2 fields after this phase field are treated much the same.

For more information, we advise that you refer to the documentation in any BC++ compiler.

Appendix C

Code for BP and Rprop learning algorithms used with NETtalk2

C.1 Error Backpropagation (BP):

```
/* First the header file. This is code of the header file BP.H which will be used next in BP.C    */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define FALSE  0;
#define TRUE   1;

typedef int                Input_Unit_type;
typedef float              Input_Threshold_type;
typedef Input_Unit_type    Input_Layer_type;
typedef float              First_Weight_Layer_type;
typedef float              Hidden_Layer_type;
typedef float              Second_Weight_Layer_type;
typedef float              Output_Layer_type;
typedef int                Training_Vector_type;
typedef float              Error_at_output_layer_elements_type;
typedef float              Error_at_hidden_layer_elements_type;
typedef float              Learning_Rate_type;
typedef float              Minimum_Error_Wanted_type;

Input_Threshold_type        In_Threshold;          /*Bias unit in the input layer */
Input_Layer_type            In_Layer[7][31];       /* The input layer */
First_Weight_Layer_type     First_WL[218][80];     /* First weight layer, including bias unit
                                                       connection */
float                       DeltaFWL[218][80];     /* Will be used with the monentum term later */
Hidden_Layer_type           HLayer[81];            /* Hidden layer of 80 PEs, and a bias unit */
Second_Weight_Layer_type    Second_WL[81][8];      /* Second weight layer, between hidden and
                                                       output */
float                       DeltaSWL[81][8];       /* Will be used with the monentum term later */
Output_Layer_type           Out_Layer[8];          /* Output layer */
Training_Vector_type        TVector[8];            /* Required output values */
Error_at_output_layer_elements_type      Out_Error[8];         /* Error derivtive at the output layer
*/
Error_at_hidden_layer_elements_type      Hidden_Error[80];     /* Error derivative at the hidden layer

                                                                */

Learning_Rate_type                       Eta;
```

```
Minimum_Error_Wanted_type             Epsilon;              /* Error tolerance */

FILE  *BFile,*TFile,*Out_File;
char  File_Name[15],Out_File_Name[15];
int   Error_Adjustment_Cycles;
int   File_Pointer;
int   File_Size;
float  Momentum;



/* now this is the code for BP.C */

#include "bp.h"

/* Following are the prototypes of the functions used, code and documentation is after the main program
*/

void Get_File_Size(void);

void Read_Input_Characters(Input_Layer_type In_Layer[7][31]);

void Read_Input(char File_Name[15]);

void Initialize_Weights_and_Thresholds(Input_Threshold_type *In_Threshold,
                                        First_Weight_Layer_type First_WL[218][80],
                                        Hidden_Layer_type HLayer[81],
                                        Second_Weight_Layer_type Second_WL[81][8],float
                                           DeltaSWL[81][8],float DeltaFWL[218][80]);

void Read_Training_Characters(Training_Vector_type TVector[8]);

void Read_Training_Input_to_Compare(char File_Name[15]);


float First_Sum(int PE_NUM,Input_Threshold_type In_Threshold,Input_Layer_type
                In_Layer[7][31],First_Weight_Layer_type First_WL[218][80]);

float Second_Sum(int PE_NUM,Hidden_Layer_type HLayer[81],
                 Second_Weight_Layer_type Second_WL[81][8]);

void Run_Neural_Network(Input_Threshold_type In_Threshold,
                        Input_Layer_type In_Layer[7][31],
                        First_Weight_Layer_type First_WL[218][80],
                         Hidden_Layer_type HLayer[81],
                        Second_Weight_Layer_type Second_WL[81][8],
                        Output_Layer_type Out_Layer[8]);

void Compute_Error_at_Output_Layer(Training_Vector_type TVector[8],
                                   Output_Layer_type Out_Layer[8],
                                   Error_at_output_layer_elements_type Out_Error[8]);

void Compute_Error_at_Hidden_Layer(Hidden_Layer_type HLayer[81],
                                   Second_Weight_Layer_type Second_WL[81][8],
                                   Error_at_output_layer_elements_type Out_Error[8],
                                   Error_at_hidden_layer_elements_type Hidden_Error[80]);
```

```
void Adjust_Weights_Hidden_to_Output(Learning_Rate_type Eta,
                                      Hidden_Layer_type HLayer[81],
                                      Error_at_output_layer_elements_type Out_Error[8],
                                      Second_Weight_Layer_type Second_WL[81][8],
                                      float DeltaSWL[81][8]);

void Adjust_Weights_Input_to_Hidden(Learning_Rate_type Eta,
                                     Input_Layer_type In_Layer[7][31],
                                     First_Weight_Layer_type First_WL[218][80],
                                     Input_Threshold_type In_Threshold,
                                     float DeltaFWL[218][80]);

float abs_val(float i);

int Satisfied(Minimum_Error_Wanted_type Epsilon);

void Print_Weights(First_Weight_Layer_type First_WL[218][80],
                   Second_Weight_Layer_type Second_WL[81][8],char File_Name[15]);


/********************************** Main Program *****************************/
void main(void)
{
 /* Specify the input file, momentum and input file size */
 Read_Input(File_Name);
 printf("\nEnter Momentum: ");
 scanf("%f",&Momentum);
 Get_File_Size();
 /* Initialize weights with random numbers and bias units with the value 1 */

Initialize_Weights_and_Thresholds(&In_Threshold,First_WL,HLayer,Second_WL,DeltaSWL,DeltaFW
L);
 Eta=0.35;
 Epsilon=0.4;
 /* The file pointer will point to the current binary number being read */
 File_Pointer=0;
 /* Specify the file that contains training vectors */
 Read_Training_Input_to_Compare(File_Name);
 /* Initialize epochs counter */
 Error_Adjustment_Cycles=0;
 strcpy(Out_File_Name,File_Name);
 strcat(Out_File_Name,".out");
 do /* This is the start of an epoch */
  {Error_Adjustment_Cycles++; /* Increment epochs counter */
   Out_File=fopen(Out_File_Name,"w");
   /* Now starts the BP alg. for one epoch */
   while ((!feof(TFile)) && (!feof(BFile)))
         {/* 1. Read values at the input window into the input layer */
          Read_Input_Characters(In_Layer);
          /* 2. Read desired output values into the training vector */
          Read_Training_Characters(TVector);
          /* 3. Feedforward step from input layer to output layer */
          Run_Neural_Network(In_Threshold,In_Layer,First_WL,HLayer,Second_WL,Out_Layer);
          /* 4. Computer error derivative at the output layer */
          Compute_Error_at_Output_Layer(TVector,Out_Layer,Out_Error);
          /* 5. Computer error derivative at the hidden layer */
```

```
            Compute_Error_at_Hidden_Layer(HLayer,Second_WL,Out_Error,Hidden_Error);
            /* 6. Backpropagate error value and adjust weights at the second weight layer */
            Adjust_Weights_Hidden_to_Output(Eta,HLayer,Out_Error,Second_WL,DeltaSWL);
            /* 7. Backpropagate more and adjust weights at the first weight layer */
            Adjust_Weights_Input_to_Hidden(Eta,In_Layer,First_WL,In_Threshold,DeltaFWL);
            }
     printf("\n end of epoch %d and working...\n",Error_Adjustment_Cycles);
     rewind(BFile);rewind(TFile);File_Pointer=0;
    } /* End of one epoch */
    /* Stop only if the average squared error is less than Epsilon */
  while (!Satisfied(Epsilon));

  printf("Number of error adjustment cycles over the files is: %d",Error_Adjustment_Cycles);
  /* Save the weights in a file to be frozen and hardwired while running the neural net
     next time, for generalization */
  Print_Weights(First_WL,Second_WL,File_Name);
  fclose(BFile);fclose(TFile);fclose(Out_File);
} /************************** End of main() **********************************/

/****************************** Functions code *******************************/

void Read_Input(char File_Name[15])
/* This function specifies the name of the input file */
{  char s[15];
   int i;

   printf("Enter the Input Binary File name (but without extension): \n");
   gets(s);
   strcpy(File_Name,s);
   strcat(s,".in");
   if ((BFile=fopen(s,"r"))==NULL)
     {puts("error opening file!..Aborting...");
      exit(1);
     }
}

void Get_File_Size()
/* This function gets the size of the binary input file, measured in number of digits */
{ int i;
  char c;

  rewind(BFile);
  i=0;
  while (!feof(BFile))
       { fscanf(BFile,"%c",&c);
         i++;
       }
  rewind(BFile);
  File_Size=i-1;
}
```

```
void Read_Input_Characters(Input_Layer_type In_Layer[7][31])
/* This function reads the 7 input groups into the input layer and slides the window 7 groups further */
{  char c;
   int i,j;

   if (File_Pointer!=0) /* The sliding window is not at the start */
     { rewind(BFile);
       i=0;
       while(i<(File_Pointer-186))  /* move pointer from the start of the file */
          {fscanf(BFile,"%c",&c); /* to the current new position.          */
           i++;
          }
      File_Pointer=File_Pointer-186;
     }

  for(i=0;i<=6;i++)
    for(j=0;j<=30;j++)
      {
       fscanf(BFile,"%c",&c);
       File_Pointer++;
       if (c=='0')
         In_Layer[i][j]=0;
       else
         In_Layer[i][j]=1;
      }
}

void Read_Training_Input_to_Compare(char File_Name[15])
/* This function specifies the name of the file that contains the desired output values for training. */
{char s[15];

 strcpy(s,File_Name);
 strcat(s,".trn");
 if ((TFile=fopen(s,"r"))==NULL)
   { puts("Error opening training file...Aborting!!");
     exit(1);
   }
}

void Initialize_Weights_and_Thresholds(Input_Threshold_type *In_Threshold,
                                       First_Weight_Layer_type First_WL[218][80],
                                       Hidden_Layer_type HLayer[81],
                                       Second_Weight_Layer_type Second_WL[81][8],
                                       float DeltaSWL[81][8],float DeltaFWL[218][80])
/* This function initializes the connection weights to random numbers between -0.1 and 0.1 and sets the
   bias units to value 1 and initializes the differences (Deltas) to zero */
{int i,j;
 time_t t;

 *In_Threshold=1;             /* Bias unit at the input layer */
 HLayer[0]=1;                 /* Bias unit at the hidden layer */
 srand((unsigned) time(&t));  /* initialize the random number generator to a random number depending
                                 on the system time. */
 /* Initialize weight connections at the first weight layer (between input and hidden layer */
 for(i=0;i<=217;i++)
   for(j=0;j<=79;j++)
```

```
      if ((rand()%10)>4)
              First_WL[i][j]=(rand()%10)/10.0;
            else
              First_WL[i][j]=-((rand()%10)/10.0);

 /* Initialize weight connections at the second weight layer (between hidden and hidden layer */
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     if ((rand()%10)>5)
              Second_WL[i][j]=(rand()%10)/10.0;
            else
              Second_WL[i][j]=-((rand()%10)/10.0);

 for(i=0;i<=217;i++)
   for(j=0;j<=79;j++)
        DeltaFWL[i][j]=0;
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
        DeltaSWL[i][j]=0;
}

void Read_Training_Characters(Training_Vector_type TVector[8])
/* This function reads the desired output in the training vector */
{  int i;

  for (i=0;i<=7;i++)
    fscanf(TFile,"%d",&TVector[i]);
}

float First_Sum(int PE_NUM,Input_Threshold_type In_Threshold,
                          Input_Layer_type In_Layer[7][31],
                          First_Weight_Layer_type First_WL[218][80])
/* This function is called from Run_Neural_Network(), it computes the sum of all the conection
    comming into a PE in the hidden layer multiplied by their weights                    */
{float temp;
 int i,j,k;

 temp=In_Threshold*First_WL[0][PE_NUM-1];      /* first the connection from the */
 k=0;                                           /* bias unit at the input layer */
 /* then the other PE at the input layer */
 for (i=0;i<=6;i++)
    for (j=0;j<=30;j++)
           {  k++;
             temp=temp+(In_Layer[i][j]*First_WL[k][PE_NUM-1]);
           }
 return temp;
}

float Second_Sum(int PE_NUM,Hidden_Layer_type HLayer[81],
                          Second_Weight_Layer_type Second_WL[81][8])
/* This function is called from Run_Neural_Network(), it computes the sum of all the conection
    comming into a PE in the output layer multiplied by their weights                    */
{ float temp;
  int j;

 temp=HLayer[0]*Second_WL[0][PE_NUM]; /* first from the bias unit in the hidden layer */
 /* then the other PE at the hidden layer */
 for (j=1;j<=80;j++)
```

```
   temp=temp+(HLayer[j]*Second_WL[j][PE_NUM]);
 return temp;
}

void Run_Neural_Network(Input_Threshold_type In_Threshold,
                              Input_Layer_type In_Layer[7][31],
                              First_Weight_Layer_type First_WL[218][80],
                              Hidden_Layer_type HLayer[81],
                              Second_Weight_Layer_type Second_WL[81][8],
                              Output_Layer_type Out_Layer[8])
/* This function represents the feedforward process, writes the output of the net to an output file */
{ int i;

  /* 1. first step in the feedforward, from the input layer to the hidden layer */
  for (i=1;i<=80;i++)
     HLayer[i]=1/(1+exp(-First_Sum(i,In_Threshold,In_Layer,First_WL)));
  /* 2. second step in the feedforward, from the hidden layer to the output layer */
  for (i=0;i<=7;i++)
     { Out_Layer[i]=1/(1+exp(-Second_Sum(i,HLayer,Second_WL)));
       fprintf(Out_File,"%f ",Out_Layer[i]);
     }
}

void Compute_Error_at_Output_Layer(Training_Vector_type TVector[8],
                                        Output_Layer_type Out_Layer[8],
                                        Error_at_output_layer_elements_type Out_Error[8])
/* This function computes the error derivative at the output layer */
{ int j;

  for (j=0;j<=7;j++)
     Out_Error[j]=Out_Layer[j]*(1-Out_Layer[j])*(TVector[j]-Out_Layer[j]);
}

void Compute_Error_at_Hidden_Layer(Hidden_Layer_type HLayer[81],
                                        Second_Weight_Layer_type Second_WL[81][8],
                                        Error_at_output_layer_elements_type Out_Error[8],
                                        Error_at_hidden_layer_elements_type Hidden_Error[80])
/* This function computes the erro drivative at the hidden layer */
{ int i,j;
  float temp;

  for(j=1;j<=80;j++)
     { temp=0;
       for(i=0;i<=7;i++);
             temp=temp+Out_Error[i]*Second_WL[j][i];
       Hidden_Error[j-1]=HLayer[j]*(1-HLayer[j])*temp;
     }
 }

 void Adjust_Weights_Hidden_to_Output(Learning_Rate_type Eta,
                                          Hidden_Layer_type HLayer[81],
                                          Error_at_output_layer_elements_type Out_Error[8],
                                          Second_Weight_Layer_type Second_WL[81][8],
                                          float DeltaSWL[81][8])
 /* This function updates\modifies the weights on the connections between the hidden layer and the
     output layer (i.e. in the second weight layer)                                          */
 { int i,j;
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     {DeltaSWL[i][j]=(Eta*Out_Error[j]*HLayer[i])+(Momentum*DeltaSWL[i][j]);
      /* DeltaSWL holds the previous value of the weight, and is used with the  Momentum term, as
seen
          in the above equation.   */
      Second_WL[i][j]=Second_WL[i][j]+DeltaSWL[i][j];
     }
}

void Adjust_Weights_Input_to_Hidden(Learning_Rate_type Eta,
                                    Input_Layer_type In_Layer[7][31],
                                    First_Weight_Layer_type First_WL[218][80],
                                    Input_Threshold_type In_Threshold,
                                    float DeltaFWL[218][80])
/* This function updates\modifies the weights on the connections between the input layer and the hidden
   layer (i.e. in the first weight layer)                                                            */
{ int i,j,k;
  int temp[217];

 for (j=0;j<=79;j++)
   {DeltaFWL[0][j]=(Eta*Hidden_Error[j]*In_Threshold)+(Momentum*DeltaFWL[0][j]);
    /* DeltaSWL holds the previous value of the weight, and is used with the Momentum term, as seen
        in the above equation. */
    First_WL[0][j]=First_WL[0][j]+DeltaFWL[0][j];
    }

 k=0;
 for(i=0;i<=6;i++)
   for(j=0;j<=30;j++)
     {temp[k]=In_Layer[i][j];
          k++;
         }
 k=0;
 for(i=1;i<=217;i++)
   {for(j=0;j<=79;j++)
      {DeltaFWL[i][j]=(Eta*Hidden_Error[j]*temp[k])+(Momentum*DeltaFWL[i][j]);
       First_WL[i][j]=First_WL[i][j]+DeltaFWL[i][j];
      }
    k++;
    }
}

float sqr(float i)
/* This function returns the squared value of its argument, to be used in the next function */
{ return ((i)*(i));}

int Satisfied(Minimum_Error_Wanted_type Epsilon)
/* This function decides if the net needs more training epochs, depending on the average squared error,
   which is half the sum of the squared difference between the actual output and the desired output over
   all the training pair.
*/
{  int Go_On;
   float TVal,BVal,Sum;

   rewind(TFile);
   fclose(Out_File);
   Out_File=fopen(Out_File_Name,"r");
  Go_On=TRUE;Sum=0;
  do
   {fscanf(TFile,"%f ",&TVal);
    fscanf(Out_File,"%f ",&BVal);
    Sum +=sqr(TVal-BVal);                /* This is the equation of the average squared error */
   }
  while (!feof(TFile) && !feof(Out_File));
  Sum /=2;
  if(Sum>Segma_Error) Go_On=FALSE;      /* If the error is still less than the required tolerance, we
                                            need more training */
  printf("Error Index: %f\n",Sum);
  rewind(TFile);fclose(Out_File);
  return Go_On;                          /* TRUE=no further training required we can stop BP.
                                            FALSE=need more training and weights adjustment */
}

void Print_Weights(First_Weight_Layer_type First_WL[218][80],
                   Second_Weight_Layer_type Second_WL[81][8],char File_Name[15])
/* This function writes the final weights after training into a file, to be used later as frozen weights in
the
   net for generalization tests and regular running.                                                    */
{ int i,j;
 char Weight_File[15];
 FILE *WFile;

 strcpy(Weight_File,File_Name);
 strcat(Weight_File,".wgh");
 WFile=fopen(Weight_File,"w");
 for(i=0;i<=217;i++)
   for(j=0;j<=79;j++)
     fprintf(WFile,"%f \n",First_WL[i][j]);
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     fprintf(WFile,"%f \n",Second_WL[i][j]);
 fclose(WFile);
}
```

C.1 Resilient Backpropagation (Rprop):

```
/* First the header file. This file contains definitions used in Rprop.C    */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define FALSE          0;
#define TRUE           1;
#define eta_minus      0.5;
#define eta_plus       1.2;
#define Delta_max      50.0;
#define Delta_min      0.0000006;
#define Delta0         0.1;
```

```
#define First_Derivative0  0.1;
#define Min(i,j)            ((i)<(j)?(i):((j)<(i)?(j):(i)));
#define Max(i,j)            ((i)<(j)?(j):((j)<(i)?(i):(i)));

typedef int           Input_Unit_type;
typedef float         Input_Threshold_type;
typedef Input_Unit_type  Input_Layer_type;
typedef float         First_Weight_Layer_type;
typedef float         Hidden_Layer_type;
typedef float         Second_Weight_Layer_type;
typedef float         Output_Layer_type;
typedef int           Training_Vector_type;
typedef float         Error_partial_derivative;
typedef float         Minimum_Error_Wanted_type;
typedef float         First_WL_Delta_type;
typedef float         Second_WL_Delta_type;

Input_Threshold_type          In_Threshold;        /*Bias unit in the input layer */
Input_Layer_type              In_Layer[7][31];     /* The input layer */
First_Weight_Layer_type       First_WL[218][80];  /* First weight layer, including bias unit
                                                       connection */
First_WL_Delta_type           FDelta[218][80];     /* delta[i][j](t) and also delta[i][j](t-1) at the
first
                                                       weight layer */
First_WL_Delta_type           FDeltaW[218][80];   /* deltaw[i][j](t) and also deltaw[i][j](t-1) will be
                                                       used later */
Hidden_Layer_type             HLayer[81];          /* Hidden layer of 80 PEs, and a bias unit */
Second_Weight_Layer_type      Second_WL[81][8];   /* Second weight layer, between hidden and
                                                       output */
Second_WL_Delta_type          SDelta[81][8];       /* delta[i][j](t) and also delta[i][j](t-1) at the
first
                                                       weight layer */
Second_WL_Delta_type          SDeltaW[81][8];      /* deltaw[i][j](t) and also deltaw[i][j](t-1) will
be
                                                       used later */
Output_Layer_type             Out_Layer[8];        /* Output layer */
Training_Vector_type          TVector[8];          /* Required output values */
Minimum_Error_Wanted_type     Epsilon;             /* Error tolerance */


FILE  *BFile,*TFile,*Out_File;
int   Num_of_epochs,i,j;
int   File_Pointer,File_Size;
float  Sum_dEdw_HLayer[218][80],Sum_dEdw_Out_Layer[81][8];    /* Error derivatives that will be
    summed over training pairs and used at the end of an epoch to determine the direction on the error
    surface */


/* now this is the code for Rprop.C */

#include "Rprop.h"

/* Following are the prototypes of the functions used, code and documentation is after the main program
*/

void Get_File_Size(void);

void Read_Input_Characters(Input_Layer_type In_Layer[7][31]);
void Read_Input(char File_Name[15]);

void Initialize_Weights_and_Thresholds(Input_Threshold_type *In_Threshold,
                                        First_Weight_Layer_type First_WL[43][80],
                                        Hidden_Layer_type HLayer[81],
                                         Second_Weight_Layer_type Second_WL[81][8],
                                         First_WL_Delta_type FDelta[43][80],
                                          First_WL_Delta_type FDeltaW[43][80],
                                          Second_WL_Delta_type SDelta[81][8],
                                          Second_WL_Delta_type SDeltaW[81][8]);

void Read_Training_Characters(Training_Vector_type TVector[8]);

void Read_Training_Input_to_Compare(char File_Name[15]);


float First_Sum(int PE_NUM,Input_Threshold_type In_Threshold,Input_Layer_type
                In_Layer[7][31],First_Weight_Layer_type First_WL[218][80]);

float Second_Sum(int PE_NUM,Hidden_Layer_type HLayer[81],
                 Second_Weight_Layer_type Second_WL[81][8]);

void Run_Neural_Network(Input_Threshold_type In_Threshold,
                        Input_Layer_type In_Layer[7][31],
                        First_Weight_Layer_type First_WL[218][80],
                         Hidden_Layer_type HLayer[81],
                        Second_Weight_Layer_type Second_WL[81][8],
                        Output_Layer_type Out_Layer[8]);

void Compute_Error_Derivative(Output_Layer_type Out_Layer[8],  Training_Vector_type TVector[8],
                              Hidden_Layer_type HLayer[81],
                              Second_Weight_Layer_type Second_WL[81][8],
                              Input_Threshold_type In_Threshold,
                              Input_Layer_type In_Layer[7][31],
                              float Sum_dEdw_HLayer[218][80],
                              float Sum_dEdw_Out_Layer[81][8]);

int Sign(float i);

void Adjust_Weights(First_Weight_Layer_type First_WL[218][80],
                     Second_Weight_Layer_type Second_WL[81][8],
                    First_WL_Delta_type FDelta[218][80],
                    First_WL_Delta_type FDeltaW[218][80],
                    Second_WL_Delta_type SDelta[81][8],
                    Second_WL_Delta_type SDeltaW[81][8],
                    float Sum_dEdw_HLayer[218][80],
                    float Sum_dEdw_Out_Layer[81][8]);

float abs_val(float i);

int Satisfied(Minimum_Error_Wanted_type Epsilon);

void Print_Weights(First_Weight_Layer_type First_WL[218][80],
                   Second_Weight_Layer_type Second_WL[81][8],char File_Name[15]);


/********************************** Main Program *****************************/
```

```
void main(void)
{
 /* Specify the input file, momentum and input file size */
 Read_Input(File_Name);
 printf("\nEnter Momentum: ");
 scanf("%f",&Momentum);
 Get_File_Size();
 /* Initialize weights with random numbers and bias units with the value 1 */
Initialize_Weights_and_Thresholds(&In_Threshold,First_WL, HLayer,Second_WL,FDelta, FDeltaW,
                                        SDelta, SDeltaW);
Eta=0.35;
Epsilon=0.4;
/* The file pointer will point to the current binary number being read */
File_Pointer=0;
/* Specify the file that contains training vectors */
Read_Training_Input_to_Compare(File_Name);
/* Initialize epochs counter */
Error_Adjustment_Cycles=0;
strcpy(Out_File_Name,File_Name);
strcat(Out_File_Name,".out");
do /* This is the start of an epoch */
   {Error_Adjustment_Cycles++; /* Increment epochs counter */
    Out_File=fopen(Out_File_Name,"w");
    /* Now starts the BP alg. for one epoch */
    while ((!feof(TFile)) && (!feof(BFile)))
          {/* 1. Read values at the input window into the input layer */
           Read_Input_Characters(In_Layer);
           /* 2. Read desired output values into the training vector */
           Read_Training_Characters(TVector);
           /* 3. Feedforward step from input layer to output layer */
           Run_Neural_Network(In_Threshold,In_Layer,First_WL,HLayer,Second_WL,Out_Layer);
           /* 4. Computer error derivative at the output layer */
           Compute_Error_Derivative(Out_Layer,TVector,HLayer,Second_WL,In_Threshold,
                                  In_Layer,Sum_dEdw_HLayer,Sum_dEdw_Out_Layer);
          }
   /* 5. Adjust weights after stepping thru the whole training set */
   Adjust_Weights(First_WL,Second_WL,FDelta,FDeltaW,SDelta,SDeltaW,Sum_dEdw_HLayer,
                 Sum_dEdw_Out_Layer);
   printf("\n end of epoch %d and working...\n",Error_Adjustment_Cycles);
   rewind(BFile);rewind(TFile);File_Pointer=0;
  } /* End of one epoch */
  /* Stop only if the average squared error is less than Epsilon */
  while (!Satisfied(Epsilon));

  printf("Number of error adjustment cycles over the files is: %d",Error_Adjustment_Cycles);
  /* Save the weights in a file to be frozen and hardwired while running the neural net
    next time, for generalization */
  Print_Weights(First_WL,Second_WL,File_Name);
  fclose(BFile);fclose(TFile);fclose(Out_File);
 } /************************** End of main() ***********************************/

/******************************* Functions code ********************************/

void Read_Input(char File_Name[15])
/* This function specifies the name of the input file */
{ char s[15];
  int i;
  printf("Enter the Input Binary File name (but without extension): \n");
  gets(s);
  strcpy(File_Name,s);
  strcat(s,".in");
  if ((BFile=fopen(s,"r"))==NULL)
    {puts("error opening file!..Aborting...");
     exit(1);
    }
}

void Get_File_Size()
/* This function gets the size of the binary input file, measured in number of digits */
{ int i;
  char c;

  rewind(BFile);
  i=0;
  while (!feof(BFile))
       { fscanf(BFile,"%c",&c);
         i++;
       }
  rewind(BFile);
  File_Size=i-1;
}

void Read_Input_Characters(Input_Layer_type In_Layer[7][31])
/* This function reads the 7 input groups into the input layer and slides the window 7 groups further */
{   char c;
    int i,j;

    if (File_Pointer!=0) /* The sliding window is not at the start */
      { rewind(BFile);
        i=0;
        while(i<(File_Pointer-186))  /* move pointer from the start of the file */
           {fscanf(BFile,"%c",&c); /* to the current new position.          */
            i++;
           }
       File_Pointer=File_Pointer-186;
      }

   for(i=0;i<=6;i++)
     for(j=0;j<=30;j++)
        {
         fscanf(BFile,"%c",&c);
         File_Pointer++;
         if (c=='0')
            In_Layer[i][j]=0;
          else
            In_Layer[i][j]=1;
        }
}

void Read_Training_Input_to_Compare(char File_Name[15])
/* This function specifies the name of the file that contains the desired output values for training. */
{char s[15];

  strcpy(s,File_Name);
  strcat(s,".trn");
```

```
 if ((TFile=fopen(s,"r"))==NULL)
   { puts("Error opening training file...Aborting!!");
     exit(1);
   }
}



void Initialize_Weights_and_Thresholds(Input_Threshold_type *In_Threshold,
                                       First_Weight_Layer_type First_WL[43][80],
                                        Hidden_Layer_type HLayer[81],
                                       Second_Weight_Layer_type Second_WL[81][8],
                                       First_WL_Delta_type FDelta[43][80],
                                       First_WL_Delta_type FDeltaW[43][80],
                                      Second_WL_Delta_type SDelta[81][8],
                                      Second_WL_Delta_type SDeltaW[81][8])
/* This function initializes the connection weights to random numbers between -0.1 and 0.1 */
/* and sets the bias units to value 1 and initializes the differences (Deltas) to zero */
{int i,j;
 time_t t;

 *In_Threshold=1;  /* Bias unit at the input layer */
 HLayer[0]=1;      /* Bias unit at the hidden layer */
 srand((unsigned) time(&t));  /* initialize the random number generator to a random number */
                              /* depending on the system time.                      */
 /* Initialize weight connections at the first weight layer (between input and
   hidden layer */
 for(i=0;i<=217;i++)
   for(j=0;j<=79;j++)
     if ((rand()%10)>4)
          First_WL[i][j]=(rand()%10)/10.0;
         else
          First_WL[i][j]=-((rand()%10)/10.0);

 /* Initialize weight connections at the second weight layer (between hidden and
   hidden layer */
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     if ((rand()%10)>5)
          Second_WL[i][j]=(rand()%10)/10.0;
         else
          Second_WL[i][j]=-((rand()%10)/10.0);
 /* Initialize Deltas */
 for(i=0;i<=217;i++)
   for(j=0;j<=79;j++)
     { FDelta[i][j]=Delta0;
          FDeltaW[i][j]=Delta0;
     }
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     { SDelta[i][j]=Delta0;
          SDeltaW[i][j]=Delta0;
     }
 }

 void Read_Training_Characters(Training_Vector_type TVector[8])
 /* This function reads the desired output in the training vector */
 {  int i;
```

```
  for (i=0;i<=7;i++)
    fscanf(TFile,"%d",&TVector[i]);
}




float First_Sum(int PE_NUM,Input_Threshold_type In_Threshold,
                    Input_Layer_type In_Layer[7][31],
                    First_Weight_Layer_type First_WL[218][80])
/* This function is called from Run_Neural_Network(), it computes the sum of all the conection
   comming into a PE in the hidden layer multiplied by their weights                        */
{float temp;
 int i,j,k;

 temp=In_Threshold*First_WL[0][PE_NUM-1];     /* first the connection from the */
 k=0;                                          /* bias unit at the input layer */
 /* then the other PE at the input layer */
 for (i=0;i<=6;i++)
    for (j=0;j<=30;j++)
         {  k++;
            temp=temp+(In_Layer[i][j]*First_WL[k][PE_NUM-1]);
         }
 return temp;
}

float Second_Sum(int PE_NUM,Hidden_Layer_type HLayer[81],
                    Second_Weight_Layer_type Second_WL[81][8])
/* This function is called from Run_Neural_Network(), it computes the sum of all the conection
   comming into a PE in the output layer multiplied by their weights                        */
{ float temp;
  int j;

 temp=HLayer[0]*Second_WL[0][PE_NUM]; /* first from the bias unit in the hidden layer */
 /* then the other PE at the hidden layer */
 for (j=1;j<=80;j++)
    temp=temp+(HLayer[j]*Second_WL[j][PE_NUM]);
 return temp;
}

void Run_Neural_Network(Input_Threshold_type In_Threshold,
                              Input_Layer_type In_Layer[7][31],
                              First_Weight_Layer_type First_WL[218][80],
                              Hidden_Layer_type HLayer[81],
                              Second_Weight_Layer_type Second_WL[81][8],
                              Output_Layer_type Out_Layer[8])
/* This function represents the feedforward process, writes the output of the net to an output file */
{ int i;

  /* 1. first step in the feedforward, from the input layer to the hidden layer */
  for (i=1;i<=80;i++)
     HLayer[i]=1/(1+exp(-First_Sum(i,In_Threshold,In_Layer,First_WL)));
  /* 2. second step in the feedforward, from the hidden layer to the output layer */
  for (i=0;i<=7;i++)
     { Out_Layer[i]=1/(1+exp(-Second_Sum(i,HLayer,Second_WL)));
       fprintf(Out_File,"%f ",Out_Layer[i]);
     }
```

```
}

void Compute_Error_Derivative(Output_Layer_type Out_Layer[8],
                              Training_Vector_type TVector[8],
                              Hidden_Layer_type HLayer[81],
                              Second_Weight_Layer_type Second_WL[81][8],
                               Input_Threshold_type In_Threshold,
                               Input_Layer_type In_Layer[7][31],
                               float Sum_dEdw_HLayer[218][80],
                               float Sum_dEdw_Out_Layer[81][8])
/* This function sums up the values of the error derivatives at the PEs in the output layer
  and the hidden layer over training pairs                                        */
{ int i,j,k;
 float temp,temp2,temp3[8];
 Input_Layer_type In_Layer_flat[31];

 /* First, compute the average squared error derivative at the output layer, and add it to
   the accumulator. Eventually, the value in the accumulator is the sum over all the training pairs */
 for(i=0;i<=7;i++)
   temp3[i]=Out_Layer[i]*(1-Out_Layer[i])*(Out_Layer[i]-TVector[i]);
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
         Sum_dEdw_Out_Layer[i][j] +=temp3[j]*HLayer[i];

 /* Now we will make a flat temporary copy of In_Layer for reasons to come ahead*/
 In_Layer_flat[0]=In_Threshold;
 k=1;
 for(i=0;i<=6;i++)
   for(j=0;j<=30;j++)
         {In_Layer_flat[k]=In_Layer[i][j];
          k++;
         }

 /* Now, compute the error derivative at the hidden layer PEs, and add it to
   the accumulator. Eventually, the value in the accumulator is the sum over all the training pairs */
 for(i=0;i<=217;i++)
    for(j=1;j<=80;j++)
          {temp=0;
           for(k=0;k<=7;k++)
             temp +=temp3[k]*Second_WL[j][k];
           Sum_dEdw_HLayer[i][j-1] +=HLayer[j]*(1-HLayer[j])*temp*In_Layer_flat[i];
          }
}

int Sign(float i)
/* This function returns 1 if its argument is +ve, -1 if it is -ve and 0 otherwise */
{ return((i)>0?1:((i)<0?-1:0));}

void Adjust_Weights(First_Weight_Layer_type First_WL[218][80],
                    Second_Weight_Layer_type Second_WL[81][8],
                    First_WL_Delta_type FDelta[218][80],
                    First_WL_Delta_type FDeltaW[218][80],
                    Second_WL_Delta_type SDelta[81][8],
                    Second_WL_Delta_type SDeltaW[81][8],
                    float Sum_dEdw_HLayer[218][80],
                    float Sum_dEdw_Out_Layer[81][8])
/* This function will compute the weights updates, depending on the direction of the gradient on the
error surface */
```

```
{ int i,j;
 float temp1,temp2,direction;

 /* For weights in the second weight layer, between the hidden layer and the output layer */
 for(i=0;i<=80;i++)
   for(j=0;j<=7;j++)
     {direction=SDeltaW[i][j]*Sum_dEdw_Out_Layer[i][j];
          if (direction<0)
            { temp1=SDelta[i][j]*eta_plus;
             temp2=Delta_max;
             SDelta[i][j]=Min(temp1,temp2);
             SDeltaW[i][j]=-Sign(Sum_dEdw_Out_Layer[i][j])*SDelta[i][j];
            }
          else if (direction>0)
                    { SDeltaW[i][j]=0;
                     temp1=SDelta[i][j]*eta_minus;
                     temp2=Delta_min;
                     SDelta[i][j]=Max(temp1,temp2);
                    }
              else
                    SDeltaW[i][j]=-Sign(Sum_dEdw_Out_Layer[i][j])*SDelta[i][j];
          Second_WL[i][j] +=SDeltaW[i][j];
          Sum_dEdw_Out_Layer[i][j]=0;
     }

 /* For weights in the first weight layer, between the input layer and the hidden layer */
 for(i=0;i<=42;i++)
   for(j=0;j<=79;j++)
     {direction=FDeltaW[i][j]*Sum_dEdw_HLayer[i][j];
          if (direction<0)
            { temp1=FDelta[i][j]*eta_plus;
             temp2=Delta_max;
             FDelta[i][j]=Min(temp1,temp2);
             FDeltaW[i][j]=-Sign(Sum_dEdw_HLayer[i][j])*FDelta[i][j];
            }
          else if (direction>0)
                    { FDeltaW[i][j]=0;
                     temp1=FDelta[i][j]*eta_minus;
                     temp2=Delta_min;
                     FDelta[i][j]=Max(temp1,temp2);
                    }
              else
                    FDeltaW[i][j]=-Sign(Sum_dEdw_HLayer[i][j])*FDelta[i][j];
          First_WL[i][j] +=FDeltaW[i][j];
          Sum_dEdw_HLayer[i][j]=0;
     }
}


 float sqr(float i)
 /* This function returns the squared value of its argument, to be used in the next function */
 { return ((i)*(i));}

 int Satisfied(Minimum_Error_Wanted_type Epsilon)
 /* This function decides if the net needs more training epochs, depending on the
    average squared error, which is half the sum of the squared difference between the
    actual output and the desired output over all the training pair         */
 {  int Go_On;
```

```
  float TVal,BVal,Sum;

  rewind(TFile);
  fclose(Out_File);
  Out_File=fopen(Out_File_Name,"r");
  Go_On=TRUE;Sum=0;
  do
    {fscanf(TFile,"%f ",&TVal);
     fscanf(Out_File,"%f ",&BVal);
     Sum +=sqr(TVal-BVal);  /* This is the equation of the average squared error */
    }
  while (!feof(TFile) && !feof(Out_File));
  Sum /=2;
  if(Sum>Segma_Error) Go_On=FALSE;  /* If the error is still less than the required
  printf("Error Index: %f\n",Sum);  /* tolerance, we need more training */
  rewind(TFile);fclose(Out_File);
  return Go_On;  /* TRUE=no further training required we can stop Rprop.
                    FALSE=need more training and weights adjustment */
}

void Print_Weights(First_Weight_Layer_type First_WL[218][80],
                   Second_Weight_Layer_type Second_WL[81][8],char File_Name[15])
/* This function writes the final weights after training into a file, to be used
   later as frozen weights in the net for generalization tests and regular running */
{ int i,j;
  char Weight_File[15];
  FILE *WFile;

  strcpy(Weight_File,File_Name);
  strcat(Weight_File,".wgh");
  WFile=fopen(Weight_File,"w");
  for(i=0;i<=217;i++)
    for(j=0;j<=79;j++)
        fprintf(WFile,"%f \n",First_WL[i][j]);
  for(i=0;i<=80;i++)
    for(j=0;j<=7;j++)
        fprintf(WFile,"%f \n",Second_WL[i][j]);
  fclose(WFile);
}
```

Note: This code is written in ANSI C, to be run on an IBM RISC 6000 machine, using AIX OS.

Appendix D

Training and Generalization Sets used with NETtalk2

D.1 English Text Sets:

D.1.1 Set of the 1000 most commonly used English words, used by Rosenberg and Sejnowski:

The following list was taken from The Internet site:

http://herens.idiap.ch/~miguel/dbases/nettalk/nettalk.info

Copyright (C) 1988 by Terrence J. Sejnowski. Permission is hereby given to use the included data for non-commercial research purposes. Contact The Johns Hopkins University, Cognitive Science Center, Baltimore MD, USA for information on commercial use.

MAINTAINER: neural-bench@cs.cmu.edu

Note (by Scott Fahlman):

The Nettalk paper by Sejnowski and Rosenberg reports a number of experiments in which the net was trained on "a list of the 1000 most common English words". Several people have asked for this list so that they can attempt to duplicate those experiments. Unfortunately, the original list was not saved. However, Terry Sejnowski states that this list was created by scanning the list of most common words in the Brown corpus and selecting the first 1000 of these that also appear in the nettalk dictionary. He also provided me with a portion of this list.

After a bit of hacking with editor macros and Lisp, I have prepared the following list of words that should closely approximate the 1000-word list used by Sejnowski and Rosenberg.

(THE OF AND TO IN THAT IS WAS HE FOR IT WITH AS HIS ON BE AT BY I THIS HAD NOT ARE BUT FROM OR HAVE AN THEY WHICH ONE YOU WERE HER ALL SHE THERE WOULD THEIR WE HIM BEEN HAS WHEN WHO WILL MORE NO IF OUT SO SAID WHAT UP ITS ABOUT INTO THAN THEM CAN ONLY OTHER NEW SOME TIME COULD THESE TWO MAY THEN DO FIRST ANY MY NOW SUCH LIKE OUR OVER MAN ME EVEN MOST MADE AFTER ALSO DID MANY BEFORE MUST THROUGH BACK WHERE MUCH YOUR WAY WELL DOWN SHOULD BECAUSE EACH JUST THOSE PEOPLE HOW TOO LITTLE STATE GOOD VERY MAKE WORLD STILL OWN SEE MEN WORK LONG HERE GET BOTH BETWEEN LIFE BEING UNDER NEVER SAME DAY ANOTHER KNOW WHILE LAST MIGHT US GREAT OLD YEAR OFF COME SINCE GO AGAINST CAME RIGHT TAKE THREE HIMSELF FEW HOUSE USE DURING WITHOUT AGAIN PLACE AMERICAN AROUND HOWEVER HOME SMALL FOUND THOUGHT WENT SAY PART ONCE HIGH GENERAL UPON SCHOOL EVERY DOES GOT UNITED LEFT NUMBER COURSE WAR UNTIL ALWAYS SOMETHING FACT THOUGH WATER LESS PUBLIC PUT THINK ALMOST HAND ENOUGH FAR TOOK HEAD YET GOVERNMENT SYSTEM SET BETTER TOLD NOTHING NIGHT END WHY FIND LOOK GOING POINT KNEW NEXT CITY BUSINESS GIVE GROUP YOUNG LET ROOM PRESIDENT SIDE SOCIAL SEVERAL GIVEN PRESENT ORDER NATIONAL RATHER POSSIBLE SECOND FACE PER AMONG FORM OFTEN EARLY WHITE CASE LARGE BECOME NEED BIG FOUR WITHIN FELT ALONG CHILDREN SAW BEST CHURCH EVER LEAST POWER THING LIGHT FAMILY INTEREST WANT MIND COUNTRY AREA DONE OPEN GOD SERVICE CERTAIN KIND PROBLEM THUS DOOR HELP SENSE WHOLE MATTER PERHAPS ITSELF TIMES HUMAN LAW LINE ABOVE NAME EXAMPLE ACTION COMPANY LOCAL SHOW WHETHER FIVE HISTORY GAVE EITHER TODAY FEET ACT ACROSS TAKEN PAST QUITE HAVING SEEN DEATH BODY EXPERIENCE REALLY HALF WEEK WORD FIELD CAR ALREADY THEMSELVES INFORMATION TELL TOGETHER SHALL COLLEGE PERIOD MONEY SURE HELD KEEP PROBABLY REAL FREE CANNOT MISS POLITICAL QUESTION AIR OFFICE BROUGHT WHOSE SPECIAL HEARD MAJOR AGO MOMENT STUDY FEDERAL KNOWN AVAILABLE STREET RESULT ECONOMIC BOY REASON POSITION CHANGE SOUTH BOARD INDIVIDUAL JOB SOCIETY WEST CLOSE TURN LOVE TRUE COMMUNITY FULL FORCE COURT SEEM COST AM WIFE FUTURE AGE VOICE CENTER WOMAN COMMON CONTROL NECESSARY POLICY FRONT SIX GIRL CLEAR FURTHER LAND ABLE FEEL PARTY MUSIC PROVIDE MOTHER UNIVERSITY EDUCATION EFFECT LEVEL CHILD SHORT RUN STOOD TOWN MILITARY MORNING TOTAL OUTSIDE FIGURE RATE ART CENTURY

CLASS NORTH LEAVE THEREFORE PLAN TOP SOUND EVIDENCE MILLION BLACK HARD
STRONG VARIOUS BELIEVE PLAY TYPE SURFACE VALUE SOON MEAN NEAR MODERN
TABLE PEACE RED ROAD TAX SITUATION PERSONAL PROCESS ALONE GONE NOR IDEA
WOMEN ENGLISH INCREASE LIVING LONGER BOOK CUT FINALLY NATURE PRIVATE
SECRETARY THIRD SECTION CALL FIRE KEPT GROUND VIEW DARK PRESSURE BASIS
SPACE FATHER EAST SPIRIT UNION EXCEPT COMPLETE WROTE RETURN SUPPORT
ATTENTION LATE PARTICULAR RECENT HOPE LIVE ELSE BROWN BEYOND PERSON
COMING DEAD INSIDE REPORT LOW STAGE MATERIAL INSTEAD READ HEART LOST
DATA AMOUNT PAY SINGLE COLD MOVE HUNDRED RESEARCH BASIC INDUSTRY TRIED
HOLD COMMITTEE ISLAND EQUIPMENT DEFENSE ACTUALLY SON SHOWN TEN RIVER
RELIGIOUS SORT CENTRAL DOING REST INDEED CARE PICTURE DIFFICULT SIMPLE FINE
SUBJECT RANGE WALL MEETING FLOOR BRING FOREIGN CENT PAPER SIMILAR FINAL
NATURAL PROPERTY COUNTY MARKET POLICE GROWTH INTERNATIONAL START TALK
WRITTEN STORY HEAR ANSWER NEEDS HALL ISSUE CONGRESS WORKING LIKELY
EARTH SAT PURPOSE LABOR STAND MEET DIFFERENCE HAIR PRODUCTION FOOD FALL
STOCK WHOM SENT LETTER PAID CLUB KNOWLEDGE HOUR YES CHRISTIAN SQUARE
READY BLUE BILL TRADE INDUSTRIAL DEAL BAD MORAL DUE ADDITION METHOD
NEITHER THROUGHOUT COLOR TRY ANYONE READING LAY NATION FRENCH
REMEMBER SIZE PHYSICAL UNDERSTAND RECORD WESTERN MEMBER SOUTHERN
NORMAL STRENGTH POPULATION VOLUME DISTRICT TEMPERATURE TROUBLE
SUMMER MAYBE RAN TRIAL LIST FRIEND EVENING LITERATURE LED MET ARMY
ASSOCIATION INFLUENCE CHANCE HUSBAND STEP FORMER SCIENCE STUDENT CAUSE
MONTH HOT AVERAGE SERIES AID DIRECT WRONG LEAD PIECE MYSELF THEORY
SOVIET ASK FREEDOM BEAUTIFUL MEANING FEAR NOTE LOT SPRING CONSIDER BED
PRESS ORGANIZATION TRUTH HOTEL EASY WIDE DEGREE HERSELF RESPECT FARM
PLANT MANNER REACTION APPROACH RUNNING LOWER GAME FEED COUPLE CHARGE
EYE DAILY PERFORMANCE BLOOD RADIO STOP TECHNICAL PROGRESS ADDITIONAL
MARCH MAIN CHIEF WINDOW DECISION RELIGION TEST IMAGE CHARACTER MIDDLE
APPEAR BRITISH RESPONSIBILITY GUN LEARNED HORSE ACCOUNT WRITING SERIOUS
LENGTH GREEN ACTIVITY FISCAL CORNER FORWARD HIT AUDIENCE SPECIFIC
NUCLEAR DOUBT STRAIGHT LATTER QUALITY JUSTICE DESIGN PLANE SEVEN STAY
POOR BORN CHOICE OPERATION PATTERN STAFF FUNCTION INCLUDE WHATEVER SUN
SHOT FAITH POOL WISH LACK SPEAK HEAVY MASS HOSPITAL BALL STANDARD AHEAD
VISIT DEEP LANGUAGE FIRM PRINCIPLE CORPS INCOME DEMOCRATIC NONE EXPECT
DISTANCE IMPORTANCE PRICE ANALYSIS SERVE PRETTY ATTITUDE CONTINUE
DETERMINE EXISTENCE DIVISION STRESS HARDLY WRITE SCENE REACH LIMITED
APPLIED AFTERNOON DRIVE PROFESSIONAL STATION HEALTH ATTACK SEASON SPENT
EIGHT ROLE CURRENT NEGRO ORIGINAL BUILT DATE MOUTH RACE UNIT TEETH
MACHINE COUNCIL COMMISSION NEWS SUPPLY RISE DEMAND UNLESS BIT SUNDAY

OFFICER MEANT WALK DOCTOR ACTUAL CLAY GLASS POET JAZZ CAUGHT HAPPY FIGHT POPULAR CONCERN SHARE STYLE BRIDGE GAS CLAIM FOLLOW THOUSAND SUPPOSE HEAT STATUS CHRIST CATTLE RADIATION USUAL FILM OPINION PRIMARY BEHAVIOR CONFERENCE SEA PROPER ATTEMPT MARRIAGE SIR HELL CONSTRUCTION WORTH PRACTICE SIGN SOURCE WAIT ARM PARK TRADITION REMAIN PROJECT AUTHORITY LORD ANNUAL JUNE OIL OBVIOUS THIN FELL PRINCIPAL JACK CONDITION DINNER BASE STRUCTURE MEASURE WEIGHT OBJECTIVE CIVIL COMPLEX MANAGEMENT MIKE EQUAL NOTED KITCHEN DANCE BALANCE CORPORATION PASS FAMOUS REGARD DEVELOP FAILURE CLOTHES COVER BREAK CARRY MOREOVER KEY KING ADD ACTIVE CHECK BOTTOM PAIN MANAGER ENEMY POETRY TOUCH FIXED POSSIBILITY SPOKE BRIGHT BATTLE PRODUCT BUILD SIGHT ROSE LOSS PREVIOUS FINANCIAL PHILOSOPHY REQUIRE SCIENTIFIC SHAPE MARKED MUSICAL VARIETY GERMAN CAPITAL CAPTAIN CONCEPT DISTRIBUTION IMPOSSIBLE LEARN BEGIN AWARE BROAD STRANGE SEX POST CATHOLIC REGULAR OPENING WINTER CAPACITY SHIP SPREAD HOUSES PREVENT MARK SPEED YESTERDAY TEAM BANK GOVERNOR INSTANCE TRAIN YOUTH PRODUCE FRESH CRISIS BAR DRINK IMMEDIATE ROUND WATCH LIVES ESSENTIAL TRIP NINE EVENT APARTMENT CAMPAIGN FILE OPPOSITE NECK INDEX TWENTY OFFER GRAY LADY FULLY INDICATE SESSION RUSSIAN PROVIDENCE STUDIED SEPARATE ATMOSPHERE PROCEDURE TERM EXPRESSION REALITY MAXIMUM ECONOMY SECRET MISSION FAST FAVOR EDGE TONE ENTER LITERARY COFFEE SOLID LAID FAIR PERMIT RESPONSE TITLE JUDGE ADDRESS MODEL ELECTION ANODE)

D.1.2 Another list of 1024 English words that we also used for training:

(one two three four five six seven eight nine ten zero first second thrid fourth fifth sixth seventh eighth ninth tenth eleven twelve twenty thirty fourty fifty sixty seventy eighty ninty hundred thousand million and or no not yes right wrong I am he she it is they we are them their there you her him his when what where how who clue glue whoever however never nor neither either with between among come go stay stand sit set up down left north south west east easy wet dry cloth cloths help find reach straight the this had been have man woman men women boy girl mister send male female mail sex type write read book news listen radio receive office room door window glass chair couch board desk disk computer key table breakfast lunch dinner light dark darkness turn on off of that pen pencil rubber money paper wallet coin iron copper dictionary speak talk speaker walk pavement street car bus airplane plan air thin thick wide narrow coat wear get take leave fly bed recorder sheets mat hell hill heaven good god bad well will nice fine friend friendship mate wife husband son daughter brother sister father mother uncle ankle under over round around circle circular rectangle triangle square dog cat animal pet bet bite byte hi bye buy by way preserve keep english united states america britain british language asia euorope latin africa australia pole earth sun star planet sea dust day night moon lake river fish boat machine saturday sunday monday tuesday wednesday thursday friday week weak month year happy birthday decade century age old young youth lady king queen prince princess bishop rabbi kitchen rule ruler a b c d e f g h i j k l m n o p q r s t u v w x y z equal enter close open delete kill back front in out beside behind middle lock look print home house end beginning begin stop start numbers eat drink food frozen hot cold water rain snow hail drag drop lift move copy push pull kick shoot save rescue dead death life live cave wave waive remove put without inside outside upside cool breeze warm boiling boling ball foot head hair cut make do spray gel color colour ears eyes nose forehead eyebrows lashes mouth cheek neck nick name teeth tooth tongue smell sniff shave smear beard shoulders chest breast billy stomach ache button rear beam leg thigh knee foot feet toe shoes trousers scissors pair pant plant wear skirt short long sleeves shirt blouse wool cotton nylon plastic real fake ring rang hand thumb finger nail polish hip muscles bone skeleton cord brain mind your own business digest smoke cigarette heart love hurt liver lever leave depart arrive departure arrival rival enemy drive peek give draw amount sum subtract add multi multiply divide conquer win lose loser winner lost found miss mess mistake missing park away far near here hear see touch taste lick suck blow wind sky dive swim swimmer sub super script marine logy science scientist college university wake morning evening afternoon dawn dusk sleep gold golden goose once twice triple tuple upon was were be time clock forest desert dessert every all none at for youngest eldest big biggest small smallest tiny full empty clever stupid genius laugh ed because why much many too lot again also into fire wood would like gave took taken given cake bake bottle cup spread lips kiss poke meal meet meat chicken beef pork sandwich cheese tea coffee mid last late early came met little older thirsty hungry say said spoke spoken eaten rub listen disturb distribute attribute fight war battle missile space ship piece please me mine certain clear dusty answer question any shall should nothing anything something some more enough began chop clouds tree leaf loaf axe cow bull duck black white yellow red blue purpule green gray dirty clean nizar radi mabroukeh training artificial neural networks to pronounce arabic text test exam speech simulator thesis master doctor patient degree jordan amman watch soon sooner later earlier slip arm bandage gone went hint just as appear disappear ask self himself herself itself themselves selves myself less bother slow quick speed wheels distance same different than then if else punish mean meaning word world report country letter sentence paragraph compose let hold work job money free both using use trouble terrible can could sure bread together gather whether weather reward shared point pin bring brought thing think tought thought teach luck lucky pick special ordinary saw fell fall fallen roots beautiful ugly pure careful instead inn hotel flat floor flour toast before after bed safe danger dangerous endangered barn bar soup plate fork knife know now spoon afterwords remind forward backward landlord landlady lord able ability unable try tried fail succeed fallen success depress depressed outgoing other each stick stuck fast spend next previous current seems notice note still bird follow along tell told tall become became firm loose through lend borrow village city capital amaze amazing all until dig digging field farm ground play us shout loud deep shallow drop dropped rush hurry people nation national international united art artist faculty engineer lawer mathematician phoneme phonetic voice sound file physician dentist nurse ill process procede precedence carry usual usually unusual kept across town wonder wander band stare great top bottom course knew serious kid kidding joke joking so whole part sad worry about from declare hide hidden sight view picture imagine imagination marry merry heard lead led lid tin place palace very high low law maximum minimum daze althuogh though break broke magic magical via magician spell held bride groom gloom want wine beer spot matter stranger known acknowledge famous barrel drain claim simple complex complicated mountain gun apple banana orange system tomato potato further map mat bat blast fade rib chin did does done beat blood bleed even ahead hat tip active awaken awake site side cook pool sing song bereau caught catch destroy build source shy shame attack peace tie latter union yard few zone raise increase decrease lower vision mission mark mouse garbage net ocean rotate)

D.1.3 An English Children Story:

The following is the story of the Golden Goose:

(Once upon a time, there was a man who had a wife and three sons.

They all lived in a cottage on the edge of a forest.

The youngest son was called Simpleton, and everyone laughed at him because he wasn't clever as his brothers.

One day the oldest son had to go into the forest to cut firewood. It would take a long time, so his mother gave him a cake and a bottle of wine for his mid-day meal.

When he came to the forest, the oldest son met a little grey man.

" I am so hungry and thirsty," said the little grey man. "Please will you give me a little piece of your cake, and a sip of your wine?"

"Certainly not," answered the oldest son. "if I give you any, I shall have nothing left for myself. Go away."

He began to chop at a big tree with his axe.

Soon the axe slipped and cut his arm, and he had to go home to have it bandaged.

So the second son went into the forest to fetch the wood. His monther gave him a cake and a bottle of wine for his mid-day meal, just as she had given his brother.

Once again the little grey man appeared, and asked for a piece of cake and a sip of wine.

The second son was as selfish as the oldest son.

"If I give you any, I shall have less for myself," he said. "Go away and don't bother me."

In the same way as his brother, he was quickly punshied for being so mean.

For, as soon as he began to chop, the axe slipped and cut his leg. He had to limp home without any wood.

"Father," said Simpleton, "why not let me go and cut the wood?"

"Oh no," said his father. "You know nothing about using an axe or working in the forest. Both your older brothers have hurt themselves. It would be asking for trouble to let you go as well."

"Please, father, let me go." Said Simpleton again. "I'm sure I could do it."

At last his father said the he could go, and Simpleton set off. All he could take with him was some stale bread and a bottle of sour beer. There was no cake or wine left in the house.

As soon as Simpleton came to the forest, the little grey man met him.

"I am so hungry and thirsty," he said. "Please will you give me a piece of your cake, and a sip of your wine?"

"I'm sorry," said Simpleton, "there is only stale bread and sour beer. You may share those with me if you wish."

When they sat down to eat together, Simpleton found that the stale bread had turned into rich cake, and the beer had changed into wine.

After they had eaten, the little grey man said, "As you have shared your meal with me so willingly, I will reward you."

He pointed to one of the trees. "Chop down that tree there," he said, "and you will find something you that will bring you good luck."

Simpleton picked up his axe and set to work to chopp down the special tree.

When it fell he sw, sitting amongst the roots, a beautiful Golden Goose. It had feathers of purest gold.

Simpleton picked the goose up very carefully. Then instead of going home, he st out for a nearby inn to stay for the night. Before he went to bed himself, he put the goose safely to bed in a barn.

The landlord of the inn had three daughters. When the saw the goose, each of them longed for one of the golden feathers.

The oldest went to the barn first and tried to pull out a feather. Then she found that she was unable to let go!

When the other two sisters came they tried to help. But as soon as they touched their sister, they stuck fast to her. All three has to spend the night in the barn, stuck to the goose and to each other.

The next morning Simpleton came in, tucked the goose under his arm, and set out. He didn't seem to notice the three girls who were still unable to let go of the bird or each other. They had to follow him.

They were still tripping and stumbling along after Simpleton when they met a priest. He told them it was naughty to run after Simpleton like that and tried to stop them, But he too became firmly fixed and had to go with them.

As they went through the village, the sexton was amazed to see the priest following simpleton and the three girls.

He called out to the priest, "Don't forget you have a christening this afternoon," and caught at the sleeve of the priest's coat. Then he too stuck fast and had to follow the others, willy nilly.

They all went on together, following Simpleton and the golden goose, until they saw two peasants diggin in a field.

"Help us," shouted the priest and the sexton together.

The two men dropped their shovels and rushedto help. They tried to pull the others away, but they too stuck fast.

There were now seven people in the little procession, all firmly stuck fast to the golen goose. Simpleton went happily on his way carrying the goose. He didn't seem to notice anything unusual.

Simpleton wasn't sure where he was going. He just kept on waling, his lucky goose tucked safely under his arm.

Over hill and dale, across fields and moors, he went with his goose. Through towns and villages they went, where people stared in wonder at them and their little band of followers.

At last, near the end of the day, they saw a great city on top of a hill.

Simpleton decided to go to the city, and the little procession, of course, had to go with him.

Now in this city reigned a king with one daughter. This princess was so serious that she never laughed and, because of this, the whole city was sad and gloomy.

The king was very worried about hisdaughter. He declared that whoever made her laugh could marry her and become a prince.

A simpleton came near to the city, he heard the king's promise. So he led his little procession straight to the palace.

The princess, who looked very sad, sat gazing down from a window.

No sooner did the princess see Simpleton, the goose, and his seven weary followers than shee began to laugh. Indeed, she laughed and laughed and laughed as though she would never stop.

The laughter of the princess broke the magic spell that had held the followers fast to the golden goose. They at once set out on their way home.

Simpleton, still clutching the golden goose, went straight to the king and asked for his reward, the princess as his bride.

The king was very happy to se his daughter laugh, but he didn't want her to marry a ragged woodcutter like Simpleton.

"Not so fast." said he king. "First you must bring me a man who can drink all the wine in my cellar."

Siimpleton thought at once of the little grey man and set out for the forest. There, on the very spot where he had found his golden goose, he saw a stranger, looking very sad.

" What's the matter?" asked Simpleton

"I'm so very, very thirsty," said the stranger.

"I think I can help you," said Simpleton. "Come with me and you shall have a whole cellarful of wine to drink."

They went to the king's palace, and the stranger sat down and began to drink and drink and drink.

Before the sun set that day, every barrel in the king's cellar had been drained dry. Once more Simpleton went to the king to claim his bride.)

D.2 Arabic Text Sets:

D.2.1 A set of 1024 commonly used Arabic Words:

The following set was formed from everyday use of commonly used Arabic words:

(من الى عن على في يمشي يركب يشتري وقف يدفع دفع نقود سياره طائره نزل اكل ياكل طعام شـراب مـاء حمـام
اين كيف كم لم لن لا نعم صحيح خطأ لو سمحت كان الملك بسم الله الرحمن الرحيم شكرا عفوا مرحبـا وداعـا اريـد
اذهب ان ذهب يذهب سيد سيده ولد بنت قهوه شاي كهرباء محارم ورقه قلم مبراه ممحاه مسطره كاس فنجان اسـتاذ
طالب دكتور مدرسه جامعه مسجد جامع كره باص تلفزيون تلفون ملعقه شـوكه سكينه محفظـه غـاز فـرن سـيجاره
احمد محمد خليل نزار راضي مبروكه أحمر شفاه أخضـر اصفـر ابيـض اسود الشـمس القمـر الليـل النـهار الظـهر
الظهيره المساء شجره تفاح برتقال فواكه خضراوات سرير وساده كرسي طاوله خزانــه صـوره كتـاب يكتـب كتابـه
مكتوب رساله رصاص حبر يستحم مرآه امراه مراه رجل مشط قميص بنطـال تنوره حذاء عـالي مرتفـع منخفـض
متوسط فوق تحت شمال يمن أعلى اسفل الوسط قصير طويل قطار شارع رصيف نائم تنام يصـحو يفيق وطن ارض
شعب امه دستور علم عماره عالم الأرضيه شقه باب نافذه ضوء اضاءه صـوت يتحـدث بلاغـه يقص نظـاره شـعر
راس دماغ يعقل عقلاني مجتهد موعدب عين حاجب رموش بؤبؤ اذن اذنان وجه أنف فم سن اسنان لسان فك يتعانق
رقبه كتف يد ايدي كف اصبع أظفر ابهام كوع ساعه حائط اسواره خاتم صدر بطـن عصـر حـزام ارجـل قـدم ركبـه
كعب أطفال اب ام ابي امي أختي أخي امي السبت الأحد الاثنين الثلاثاء العدل الانسان الأربعاء الخميس الجمعه يـوم
اسبوع أشهر سنوات سنين سنه كبير صغير نحيل نحيف طقس جـو سـماء جيش صنـدوق بنـدوره خيـار انجليـزي
عربي اجنبي يهودي اردني مالح سكر يسمع جارور يفتح يغلق مفتاح دجاجه ديك سمكه حمام بقره خارووف الوقـت
الزمان مسلسل برنامج نجوم زياره زائر بحر نهر بحيره محيط نبـع زكـام صخـره رمـل تـراب نبـات بنـات حجـاره
أطفال دافع بارد ساخن مدفأه يصبح حرام صافره يصفر ينفخ يضـغط دم اسعاف معتم كمبيوتـر يحسـب سـريع بطيـخ
سجاده هاتف يسمع يرى راى رؤيه غرفه سكان حلو مر حامض كنافه حوامه زجاج قنينه علبـه متنزه حديقـه غابـه
بستان مطار خشب حديد عنب بطيخ حليب عجل كهرباني مغناطيسي عامل نظافه صلاه مهملات آله ماكنه مـهندس
مبرمج سائق حبيب حب يحب قبله يصلي جبين متوضأ وضوء ينظف يسـوق مكتـب مجمـع بنـك مقالـة واحـد اثنـان
ثلاثة اربعة خمسة ستة سبعة ثمانية تسعة عشرة احـدى عشـر الاول الثـاني الثـالث الرابـع الخـامس السـادس السـابع
الثامن التاسع العاشر شارع معبد يفتح يغلق آه عصا شيخ عجوز شعاع لكن كلمة امل حياة يا انا انت نحن انتـم انتـن
انتما هو هي جل جلاله عليه سلام الاردن عمان العاصمة العدو دبابة السلاح حـرب معركـة اسـد المملكـة الهاشـمية
جمهوريه جمهورية أمريكا شركة مصنع غذاء انتاج حلوى املاح كم مما بائع دموع ينسى ذاكرة اتمنى امنيـه هـواء
عذاب يشتاق مع يرجع السلام عليكم جلس مجلس خوف رعب قالت حزن عليك مات جنازة قبر قيامة نار جنة عـدن
فداء شهيد مقتول مجرم مجنون مهوس حمار ثور كلب بغل معزاه نمر هره قطه طباشير دفاع غرفة بيض ديناصور
عهد عمر قديم جديد لام غار لماذا العشق الروح دار دوار شاشة مار شاطر ممتاز حسن مقبول ناجح راسـب خـائب
فاشل فاشلة مصعد يصعد ينزل شـهاده درج سلم خطبـه عرس حفلـه راقصـه غطـاء مفـاتيح ابتسـم هـادئ اعصـاب

شبكات عصبيه اصطناعيه أخذت صلح دلال هبه هديه معدود تعال اذهب هذه هذا ساحه يمسح جمال جاذبيه جوهره
فضه مزارع سلك حبل حامل راهب رهيب فتاه ميزان حوت ملابس داخليه خارجيه وزاره وزير اميره مؤلف حداد
نجار حلاق عبد الرؤوف يتناسب مناسبه عزاء غدا بعد قبل خلال عبر سماعه سكوت سكتت سر سراء ضراء
مقهى كفتيريا عمادة عميد قانورات أوساخ شحن زر سحاب معطف مطر ثلج شتاء صيف خريف ربيع زهور ورد
تكاثر ولاده ملهى القرآن التوراه الانجيل مسلم مسيحي يهودي كافر شيطان ملاك مزعج اعماق عميق ضحل ثعلب
ماكر خبيث بقره مشروب غناء مغنيه طرب سباق هدف هداف مباراه ألعاب سباحه مسبح جيران أقرباء زبائن
زبون شغل يشتغل يتخرج دفء امتحان اختبار برد مثلج نظيف قليل كثير غسق فجر نور رئيس نائب نشرح كتيب
استلقت يتردد يتذبذب موجه راديو مذياع رؤيه حلق بلعوم مريء أورده معده أمعاء غليظ دقيق متأخر مبكر غير
مستقيم أعوج عمود البحرين اسرائيل فلسطين الريف الباديه بدوي المدينه القريه السعوديه مصر القاهره الاهرامات
آثار قائم جالس مائل جلوس قيام اذاعة محطه كاز بنزين سولار زيت اشعال مشتعل تدخين ممنوع مسموح سام
صحي مستشفى اغماء وجع علم محرك قرص تسجيل ادعاء نقود دينار نصف قرش قروش فلس فلوس دولار ليره
سوريه تركيه العراق الكويت ليبيا الجزائر المغرب تونس المشرق غروب شروق البحر ابحرت قارب سفينه مياه
جبال سهول وديان وادي مراعي حقول ألغام متفجرات درب دروب طريق ممر مدخل تذكره هندسه كليه مجتمع
اجتماع علوم احياء كمياء رياضيات فيزياء جيولوجيا رياضه زراعه طب صيدله مجمع تربيه شريعه ديانه الاسلام
دوار مدار برج مكتبه دخله بوابه مبنى آداب مركز لغات استشارات أبحاث صوتيه مرئيه بصريه طابق طوابق
استراتجيه تكتيك تقنيه تكنولوجيه تجاره اعمال سياسه حقوق الانسان ادم حواء ابليس شيطان الخير الشر خير شر
صالح طالح يطلع خلفي أمامي جانبي جنوب شرق غرب رياح ريح أعصير دوامه طاقه شاويش تيغزل رقيق
محاضرات رجال سارق يحرق غيره يعيش يموت مرض ترقيه صافره رقصه ترقص يلعب رقصت حاد قوي
ضعيف حافي بلى هم سينميا سهره هيا بنا بكم بهم بهن بهما معي معك معهما معه معهم معهن شوكولاته كاكاو
حليب بوظه سماح خصام شقاق شقيق أخو ابن بن اخ حم فو ذو ذويه أخت أخيها امك أباك يبكي بكاء تماسيح صياح
أغنيه منه ممانه ألف ألفين آلآف سلطه سلعه عبد حر مستعبد مظلوم ظالم حاكم ابتدائي اعدادي ثانوي توجيهي يعد
يحضر يحضر بدايه ابتداء يطبع طباعه بابا ماما أشرطه مسجل يتابع يفقد مفقوده أداه ادوات مبرآه اطار خارجي
داخلي امتصاص ابتزاز اختطاف خطاف حرامي بيض الخارج الداخل الموجود المفقود عرس عريس عروس
موسيقى كانون شباط آذار خالد نيسان أيار حزيران تموز آب أيلول تشرين شبكات مواسير الشبكات العصبيه
الاصطناعيه تدريب نطق لفظ اللغه العربيه الانجليزيه الفرنسيه يناير فبراير مارس ابريل برميل مايو يونيو يوليو
اغسطس عطست سبتمبير اكتوبر نوفمبر ديسمبر جماد جمادى شوال رمضضان محرم صفر سفر ربيع رجب
شعبان شبعان ذو القعده الحجه دميه يرن استعمار مستعمره مستوطنه عصير موجه دليل سياحه لين متين مكسور
ملصق لاصق طبيب ممرضه غرق نبيذ مريضه متزوجه أعزب عزباء طفل شاب عجوز شيخ امام عادل مراهقه
متهور متسلط عربه فأر ظريف الكواكب أفلاك ذره جزيء ماده تركيب عصبي خليه روضه بستان حضانه حصانه
يتحصن يتسلح يخلع يفك مفك شريكة العمر ضغط انفراج نبوءه الصبح سكاكر خيط الفرقان دراسات شرطي معه)

D.2.2 A Corpus of Continuous Arabic Text:

The following is an Arabic story by the famous Arab writer Gibran Khalil Gibran:

طفلان

وقف الأمير على شرفة القصر و نادى الجمزع المزدحمه في تلك الحديقه و قال: "أبشـركم و أهنئ البلاد. فالأميرة قد وضعت غلاما يحي شرف عائلتي المجيده و يكون لكم فخرا و مـلاذا و وريثا لما أبقته أجدادي العظام. افرحوا و تهللوا فمستقبلكم صار مناطا بسليل المعالي".

فصاحت الجموع و ملأت الفضاء بأهازيجالفرح متأهلة بمن سوف يربى علـى مـهد الترف و يشب من منصة الاعزاز و يصير بعد ذلك حاكما مطلقا برقاب العباد، ضابطا بقونه أعنة الضعفـاء، حـرا باستخدام أجسادهم و اتلاف أرواحهم. من أجـل ذلك كـانوا يفرحـون و يغنـون الأناشـيد و يعـاقرون كاسات السرور.

و بينما سكان تلك المدينه يمجدون القوي و يحتقرون ذواتهم و يتغنون باسم المستبد و الملائكه تبكي على صغرهم كان في بيت حقـير مـهجور امـراه مطروحـه علـى سرير السقام تضم الـى صدرهـا الملتهب طفلا ملتفا بأقمطه باليه.

صبيه كتبت لها الأيام فقرا و الفقر شقاء فأهملت من بني الأنسان. زوجة أمات رفيقها الضعيف ظلـم الأمير القوي وحيدة بعثت اليها الآلهه في تلك الليله رفيقا صغيرا يكبل يديها دون العمل و الارتزاق.

و لما سكنت جلبه الناس في الشوارع و ضعت تلك المسكينه طفلها على حضنها و نظرت في عينيه اللامعتين و بكت بكـاء مـرا، كأنـها تريـد أن تعمـده بـالدموع السخيه، و قـالت بصـوت تتصـدع لـه الصخور: "لماذا جئـت يـا فلذه كبـدي مـن عـالم الأرواح أطمعـا بمشـاطرتي الحيـاة المرة؟ أرحمـة بضعفي؟ لماذا تركت الملائكه و الفضـاء و الوسيع و أتيت الى هذه الحيـاة الضيقـه المملـوءه شـقاء و مذله؟ ليس عندي يا وحيـدي الا الدمـوع، فـهل تتغـذى بـها بـدلا مـن الحليـب؟ و هـل تلبـس ذراعـي

العاريتين عوضا عن النسيج؟ صغار الحيوان ترعى الأعشاب و تبيت في أوكارها آمنة، و صغار الطير تلتقط البذور و تنام بين الأغصان مغبوطة و أنت يا ولدي ليس لك الا تنهداتي و ضعفي.

حينئذ ضمت الطفل الى صدرها بشده كأنها تريد أن تجعل الجسدين جسدا واحدا و رفعت عينيها نحو العلاء و صرخت (أرفق بنا يا رب).

و لما انقشعت الغيوم عن وجه القمر دخلت أشعته اللطيفه من نافذه ذلك البيت الحقير و انسكبت على جسدين هامدين.

D.2.3 An Arabic Children Story:

The following is the Arabic version of the story of "Little Red Ridinghood":

(عملت أم رباب بعض الحلوى و نادت ابنتها و قالت لها: رباب خذي هذه الحلوى الى جدتك.

فرحت رباب لأنها كانت تحب جدتها و تحب أن تأخذ لها الحلوى.

حملت رباب سلة الحلوى و ودعت أمها. قالت لها أمها: انتبهي يا ابنتي من الذئب الشرير.

توقفت رباب في الغابه و قطفت أزهارا لجدتها. كانت جدتها تحب الأزهار كثيرا.

رأى الذئب رباب ذات الثوب الأحمر و قال لها: تعالي نلعب معا.

رفضت رباب و قالت: سأزور جدتي المريضه و أحمل لها سلة الحلوى و أعطيها الأزهار.

ركض الذئب الشرير الى بيت جدة رباب.

قال بصوت ناعم: أنا رباب يا جدتي، افتحي لي الباب.

فرحت الجده و قالت: أدخلي يا رباب، أنا مريضه، و الباب مفتوح.

و لما رأت الجده الذئب الشرير أمامها قفزت من السرير بخوف شديد و اختبأت فوق خزانة الثياب.

نام الذئب الشرير في سرير الجده و لف نفسه بشالها و وضع على راسه طاقيتها.

وصلت رباب ذات الثوب الأحمر الى بيت جدتها تحمل معها سلة الحلوى و باقة الأزهار.

نادت رباب جدتها و قالت: جدتي، أنا رباب، افتحي الباب، من فضلك.

أجاب الذئب الشرير بصوت خشن: ادخلي يا رباب، انا مشتاقة اليك كثيرا.

دخلت رباب الى البيت و قالت: جئتك يا جدتي بالحلوى اللذيذه و الأزهار الجميله.

نظرت رباب الى الذئب الشرير و هو ينام في سرير جدتها و يلبس ثيابها.

استغربت رباب شكل جدتها و قالت: عيناك كبيرتان يا جدتي.

أجاب الذئب: نعم، عيناي كبيرتان لأراك جيدا يا حبيبتي.

قالت رباب: أذناك كبيرتان يا جدتي.

أجاب الذئب: نعم، أذناي كبيرتان لأسمعك جيدا يا حبيبتي.

قالت رباب: و أسنانك كبيره يا جدتي.

كشف الذئب عن وجهه و قال: نعم، أسناني كبيره لآكلك بسرعه.

قفز الذئب الشرير من السرير.

خافت رباب خوفا شديدا و راحت تركض و تصيح: خلصوني، خلصوني.

من بعيد رأت رباب رجلا قويا.

كان ذلك الرجل أباها.

تعلقت رباب بأبيها و قالت: خلصني يا أبي. الذئب الشرير يريد أن يأكلني.

حملها أباها و وضعها على شجره و قال لها: لا تخافي، سأقتل الذئب الشرير.

هجم والد رباب على الذئب و قتله بضربة واحده.

ارتاحت رباب و نزلت عن الشجره و قالت: أنا خائفه على جدتي يا أبي تعال نبحث عنها.

ذهب الأب و رباب الى بيت الجده كانت الجده تصيح أنا فوق الخزانه، خلصوني، خلصوني، أرجوكم خلصوني.

أسرع والد رباب و أنزل الجده من فوق الخزانه و ساعدها، هو و رباب، لتنام في سريرها.

جلست رباب ذات الذوب الأحمر قب جدتها و قالت: يا جدتي جئتك بحلوى لذيذه و أزهار جميله.

فرحت الجده كثيرا لأنها كانت تحب الحلوى و الأزهار، و كانت تحب رباب أكثر من كل ما في الدنيا.

ودعت رباب جدتها و قالت لها: سأزورك دائما يا جدتي.

قالت لها جدتها: ما ألطفك يا رباب! خذي هذه التفاحه. سلمي على أمك. مع السلامه يا رباب.)

اجباري للحفاظ على البقاء و هذا هدف فطري غريزي دائم عند الانسان، و من هنا نجد أن ممارسة الأنشطه الحركيه و اللياقه البدنيه نشأت منذ نشأة الخلق و بتتبع مفهوم اللياقه البدنيه عبر العصور نجد أن هنالك حضارات مختلفه متتابعه اهتمت بالاعداد البدني للأطفال من خلال ألوان متعدده من الأنشطه البدنيه و غلب على هذه الأنشطه الطابع العسكري و طابع القوه و المغامره و التحدي و لقد كانت الأجسام السائده في ذلك العصر اجساما قويه.

ففي حضارة اسبارطه كان هنالك الاهتمام باللياقه البدنيه موجها لخدمة الأغراض العسكريه و اعداد مواطن قوي و شجاع للدفاع عن الوطن ضد الغزو الخارجي كما و اهتموا باعداد المرأه للايمانهم بأن المرأه القويه ستنجب الأطفال الأقوياء.

أما في أثينا فقد كان المجتمع ديناميكي متطور آمن بالحريه و ركز على الألعاب الرياضيه و وضعوا أساسا لتطور الألعاب الأولمبيه و كانت ممارستهم للتربيه البدنيه وسيله لتحقيق أهداف عسكريه و ترويحيه و اجتماعيه.)

D.2.4 A Corpus of Continuous Arabic Text from the Daily Newspaper:

Following is a three-paragraphs piece of text from a the newspaper, used to train and test NETtalk2:

(تعلن اليوم الجمعه نتائج امتحان الثانويه العامه للعام الحالي.

و يعقد الدكتور منذر المصري وزير التربيه و التعليم و التعليم العالي عند الساعه التاسعه و النصف صباحا مؤتمرا صحفيا يعلن فيه النتائج النهائيه للامتحان و أسماء العشره الأوائل في مختلف الفروع.

و كانت لجنة الامتحانات العامه في الوزاره عقدت اجتماعا أمس الأول برئاسة الدكتور عزت جرادات أمين عام الوزاره نظرت خلاله في الحالات الانسانيه و الفرديه و اتخذت القرارات المناسبه بشأنها.)

D.2.5 Other Arabic Corpora:

Many other corpora of continuous Arabic text were used, some are:

1- A small story by Gibran Khalil Gibran:

(كان رجل يحفر في حقله. و فيما هو يحفر عثر على تمثال بديع من المرمر الجميل. فأخذه و مضى به الى رجل كان شديد الولع بالآثار و عرضه عليه. فاشتراه منه بأبهظ الأثمان. و مضى كل منهما في سبيله.

و بينما كان البائع راجعا الى بيته أخذ يفكر في ذاته قائلا: "ما أكثر ما في هذا المال من القوة و الحياة! انه بالحقيقة ليدهشني كيف أن رجلا ينفق مالا هذا مقداره لقاء صخر أصم فاقد الحركة، كان مدفونا في الأرض منذ ألف سنه و لم يحلم به أحد."

و في الساعة عينها كان المشتري يتأمل التمثال مفكرا و قائلا في ذاته: "تبارك ما فيك من الجمال! تبارك ما فيك من الحياة! حلم أية نفس علوية أنت؟ هذه بالحقيقه نضارة أعطيتها من نوم ألف سنة في سكينة الأرض! انني و الله لا أفهم كيف يمكن الانسان أن يبيع مثل هذه الطرفه النادره بمال جامد زائل.")

2-

(لقد لازمت الحركه الانسان منذ أقدم العصور حيث فرضت الطبيعه عليه العنايه بجسمه و اكسابه القوه و اللياقه حتى يستطيع الدفاع عن نفسه، ففي العصر الحجري القديم عاش الانسان حياة مليئة بالخوف و الترحال و استخدم الحجار هو أوراق الشجر للدفاع عن نفسه، فلقد مارس الانسان في هذا العصر الأنشطه الحركيه المختلفه بشكل

ملخص

تدريب الشبكات العصبيه الاصطناعيه على لفظ النص العربي

اعداد

نزار راضي مبروكه

المشرف

الدكتور خليل الهندي

المشرف المشارك

الدكتور عبد الرؤوف الحلاق

لقد تساءل الانسان منذ القدم عن امكانية بناء الدماغ الاصطناعي. ان الشبكات العصبية الاصطناعية معروفه لدينا منذ أكثر من نصف قرن، و مع ظهور ثورة الحاسبات أصبح بالامكان انشاء نموذج للدماغ الاصطناعي الى حد ما، كما و استخدمت الشبكات العصبية الاصطناعية منذ الثمانينات لمحاكاة وظائف الدماغ الانساني بطرق مختلفة. ان هذه النماذج تتميز بقدرتها على التعلم، التذكر و مسايرة التغير بنفسها.

في دراستنا هذه، سوف نبحث في "NETtalk"، الشبكة العصبية التي تنطق النص الانجليزي، ثم سنبني نموذجا مشابها (NETtalk2) و نحاول تدريبه على نطق النص العربي باستخدام الانتشار الخلفي للخطأ (Error Backpropagation) و الانتشار الخلفي الرجوع (Resilient Backpropagation).

لقد أظهرت النتائج أن نموذجا مثل "NETtalk" غير قادر على نطق النص العربي، من خلال عدم قدرته على تعلمه أو عدم قدرته على انشاء التعميم ان حصل التعلم؛ لأسباب تعتبر من أهم خصائص اللغة العربية و قواعدها و لا يمكن اهمالها. في نهاية البحث، سوف نشير الى هذه الأسباب و نناقشها و نوضح كل منها بأمثلة. كما نقدم أيضا أمثلة على

تدريب (NETtalk2) على نطق النص العربي باستخدام طبقتين مخفيتين (Hidden Layers). في هذه الحالة يتحسن التعميم و يزداد بنسبة ٨% عند استخدام طبقتين مخفيتين في كل منهما ٨٠ وحده معالجه.

أما عند استخدام التدريب المتواتر (على طريقة جوردن)، فلقد أظهرت NETtalk2 نتائج جيده و تحسنت قدرتها في التعميم بنسبة ١٢%.