TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 759 • Publication 759

Enikö Beatrice Bilcu

Text-To-Phoneme Mapping Using Neural Networks

Tampere 2008

Tampereen teknillinen yliopisto. Julkaisu 759
Tampere University of Technology. Publication 759

Enikö Beatrice Bilcu

Text-To-Phoneme Mapping Using Neural Networks

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB224, at Tampere University of Technology, on the 22nd of October 2008, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2008

ISBN 978-952-15-2045-7 (printed)
ISBN 978-952-15-2046-4 (PDF)
ISSN 1459-2045

Tampereen teknillinen yliopisto. Julkaisu ...
Tampere University of Technology. Publication ...

Enikő Beatrice Bilcu

Text-To-Phoneme Mapping Using Neural Networks

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB224 at Tampere University of Technology on the 22nd of October 2008, at 12:00 o'clock noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2008

- To my son Tomi Eduard -

Abstract

Text-to-phoneme (TTP) mapping, also called grapheme-to-phoneme (GTP) conversion, defines the process of transforming a written text into its corresponding phonetic transcription. Text-to-phoneme mapping is a necessary step in any state-of-the-art automatic speech recognition (ASR) and text-to-speech (TTS) system, where the textual information changes dynamically (i.e., new contact entries for name dialing, or new short messages or e-mails to be read out by a device). There are significant differences between the implementation requirements of a text-to-phoneme mapping module embedded into the automatic speech recognition and into the text-to-speech systems: in automatic speech recognition systems the errors of the text-to-phoneme mapping module are tolerated better (leading to occasional recognition errors) than in the text-to-speech applications, where the effect is immediately and in all cases audible. Automatic speech recognition systems typically use text-to-phoneme mapping to lower the footprint (to avoid storing the lexicon), while maintaining quality. The use of text-to-phoneme mapping in the text-to-speech systems is different. In addition to the phonetic information, the text-to-speech systems also need prosodic information to be able to produce high quality speech, which cannot be predicted by text-to-phoneme mapping. Most state-of-the-art text-to-speech systems use explicit pronunciation lexicon, which is aimed at providing the widest possible coverage, in the order of $100K$ words, with high quality pronunciation information. Because of this reason, text-to-phoneme mapping is typically used as a fall-back strategy, when the system encounters very rare or non-native words and the quality of a text-to-speech system is indirectly affected by the quality of the grapheme-to-phoneme conversion. Another important issue is the question of training the text-to-phoneme mapping module. The problem of grapheme-to-phoneme conversion is a static one and such a system is trained off-line. The correspondence between the written and spoken form of a language is usually unchanged in the lifetime of an application. So the complexity/speed of the model training is of secondary importance compared to e.g., the speed of convergence or model size.

In this thesis, the problem of text-to-phoneme mapping using neural networks is stud-

ied. One of the main goals of the thesis is to provide a comprehensive analysis of different neural network structures which can be implemented to convert a written text into its corresponding phonetic transcription. Another important target, of this work, is to provide new solutions that improve the performance of the existing algorithms, in terms of convergence speed and phoneme accuracy. Three main neural network classes are studied in this thesis: the multilayer perceptron (MLP) neural network, the recurrent neural network (RNN) and the bidirectional recurrent neural network (BRNN).

Due to their ability of self adaptation, neural networks have been shown to be a viable solution in applications that require modeling abilities. Such an application is the text-to-phoneme mapping where the correspondence between letters of a written text and their corresponding phonetic transcription must be modeled.

One of the main concerns in all practical implementations, where neural networks are used, is to develop algorithms which provide fast convergence of the synaptic weights and in the same time good mapping performances. When a neural network is trained for text-to-phoneme mapping, at every iteration, a letter-phoneme pair is presented to the network such that, the number of letters and the number of training iterations are equal. As a result, fast convergence of the neural network means smaller size of the training dictionary since fast convergence is in fact similar to less necessary training letters[1]. A fast convergence speed is important in applications where only a small linguistic database is available. Of course, one solution could be to use a small dictionary (with very few words) which is presented at the input of the neural network many times until the convergence of the synaptic weights is reached. In this case the time of training becomes more important[2]. Taking into account these two sides of the convergence speed (the size of the training dictionary and the processing time during training) one can understand the importance of having algorithms that ensure fast convergence of the neural network.

It is well known that the error back-propagation algorithm which is used to train the MLP neural network, possess sometimes a quite slow convergence (a very large number of iterations required to reach the stability point). In order to increase the convergence speed two novel alternative solutions are proposed in this thesis: one using an adaptive learning rate in the training process and another which is a transform domain implementation of the multilayer perceptron neural network. The computational complexity of the two proposed training algorithms is slightly higher than the computational complexity of the

[1]Actually, as we will see during this thesis, at the input of the neural network we have "patterns" of 3, 5, 7, etc letters. In any case, fast convergence means less input letters in the training phase.

[2]Time of training equals the product between the number of iterations and the time needed to complete one iteration.

error back-propagation algorithm but the number of training iterations is highly reduced. Due to this fact, although the three algorithms might have the same training time, the novel algorithms necessitate smaller training dictionary.

Due to the limitations of the processing power that usually are encountered in real devices, another very important requirement for a text-to-phoneme mapping system is to have low computational and memory costs. In the case of text-to-phoneme mapping systems based in neural networks, the computational complexity is mainly linked to the mathematical complexity of the training algorithm as well as to the number of the synaptic weights of the neural network. Memory load is due to the number of synaptic weights of the neural network which must be stored.

Taking into account all these limitations and implementation requirements, in this thesis, several neural network structures with different number of synaptic weights and trained with various training algorithms, are studied. The modeling capability of the neural networks is addressed, which is translated in the text-to-phoneme mapping case into the phoneme accuracy. Different neural network structures, training algorithms and network complexities are analyzed also from this point of view. As a remark here, we mention that input letter encoding plays a very important role in the phoneme accuracy of the grapheme-to-phoneme conversion system. This is why special attention has been paid to the comparative analysis of the performances (in terms of phoneme accuracy) obtained with several orthogonal and non-orthogonal encoding of the input letters.

The thesis is structured into four main parts. Chapter 1 brings the reader into the world of text-to-phoneme mapping. In Chapter 2 several different neural network structures and their corresponding training algorithms are described and two new training algorithms are introduced and analyzed. In Chapter 3 the experimental results, for the problem of monolingual text-to-phoneme mapping, obtained with the neural networks described in Chapter 2 are shown. Chapter 4 is dedicated to the problem of bilingual grapheme-to-phoneme conversion and Chapter 5 concludes the thesis.

Acknowledgements

The research presented in this thesis has been carried out at the Department of Signal Processing, from Tampere University of Technology, in Tampere, Finland.

First and foremost, my sincerest gratitude goes to my supervisor, Prof. Jaakko Astola, for his endless patience, for continuous guidance, support and for giving me a privilege to work in the great atmosphere at the Department of Signal Processing, as well as for his unfailing interest to my research, accompanied by excellent ideas and comments.

I am particularly indebted to DrTech. Jukka Saarinen from Nokia Research Center, Tampere, Finland, who has given me the great chance to work within his research team in the Department of Computer Systems, Tampere University of Technology, during 2001-2003, and also for numerous fruitful discussions and constructive recommendations.

Distinguished thanks are due to DrTech. Petri Salmela for his great help and for fruitful technical discussions during my work within the Department of Computer Systems.

My special thanks are addressed to DrTech. Imre Kiss from Nokia Research Center and DrTech. Marco Carli from the University of Roma TRE, the reviewers of this thesis. Their excellent comments have helped me improve this dissertation, I thank them for their time and effort.

I appreciate the generosity of Tampere Graduate School in Information Science and Technology (TISE) which partially funded my studies. In particular, I thank Prof. Markku Renfors, the Director of TISE, who granted part of my salary and funded the scientific travels during this period. I send many warm thanks to DrTech. Pertti Koivisto, Coordinator of TISE for the consistent help and for organizing enjoyable and interesting courses and seminars. This thesis was also supported by Academy of Finland, Nokia Research Center, Nokia Foundation and by Jenni and Antti Wihuri Foundation, which are appreciatively thanked.

The people from the Department of Signal Processing have greatly contributed to the wonderful atmosphere and working conditions of which I took benefit. I thank them all.

Warm thanks are due to Pirkko Ruotsalainen, Ulla Siltaloppi, Kirsi Järnström, Virve Larmila and Elina Orava for their always kind help with practical matters. They have

always been ready to provide friendly support and kind advices. I am also thankful to Ulla and Seppo Siltaloppi for their friendship and the very nice time spent together.

Special thanks go to Prof. DrTech. Corneliu Rusu and Prof. DrTech Lucia Rusu for theirs valuable advices and encouragements. I am profoundly grateful to all my Romanian and international friends here in Finland, for having such a great time together in unforgettable parties and for other free time activities. Special thanks go to Maria, Ioan, Daniela and Marian Crisan for being my family here in Tampere. I wish to thank my friends Anca, Mircea and their families and to Lala and Micu for their fun company and for being my friends. I would also like to thank to Reija and Sebastian Sjöblad for their warm friendship.

I express my deep gratitude to my parents Ibolya and Endre for every moment of my life for encouraging me in my studies and giving me the freedom and support to become the person I am. I will always be grateful to my grandparents for the love, affection and help during theirs life.

Last but not least I am truly grateful to my husbands family, Eugenia, Mircea, Elena, Alin, Ionut for their endless love, continuous support and for being part of my family.

Finally and most, I am deeply and forever indebted to my dear husband Radu for his love, support and never ending patience during the intensive stages of putting this work together. Without you Radu I could never achieved this important goal of my life. I wish to thank also our beloved son Tomi Eduard for bringing all the joy and happiness and for showing me what really matters in life.

Tampere, June 2008 Enikő Beatrice Bilcu

Contents

List of Figures

List of Tables

Abbreviations

ASR	Automatic Speech Recognition
BPTT	Back-Propagation Through Time
BRNN	Bidirectional Recurrent Neural Network
CMU	Carnegie Mellon University
DCT	Discrete Cosine Transform
DT	Decision Trees
FFT	Fast Fourier Transform
GTP	Grapheme-To-Phoneme
HMM	Hidden Markov Model
MLP	Multilayer Perceptron
MLPALR	Multilayer Perceptron with Adaptive Learning Rate
MLPVLR	Multilayer Perceptron with Variable Learning Rate
MSE	Mean Squared Error
NN	Neural Network
NOBC	Non-Orthogonal Binary Code
NOC	Non-Orthogonal Code
OBC	Orthogonal Binary Code
OCR	Optical Character Recognition
RC	Random Code
RNN	Recurrent Neural Network
RTRL	Real Time Recurrent Learning
TDMLP	Transform Domain Multilayer Perceptron
TTP	Text-To-Phoneme
TTS	Text-To-Speech
UI	User Interface

Notations

y	a constant scalar
$h(x)$	a function
$h'(x)$	the derivative of $h(x)$
$\mathbf{y}$	a vector with constant elements
$\mathbf{y}_L$	a vector of length L
$\mathbf{y}(n)$	a vector with time-varying elements
$\mathbf{y}^t$	the transpose of the vector $\mathbf{y}$
$\Delta\mathbf{y}(n)$	the difference of two instances of the vector $\mathbf{y}(n)$
$\mathbf{y}_N$	a vector of length N
$\mathbf{W}$	a matrix with constant elements
$\mathbf{W}(n)$	a matrix with time-varying elements
$\mathbf{W}_{N\times M}$	a matrix with dimensions $N \times M$
$\mathbf{W}^t$	the transposed of matrix $\mathbf{W}$
$W_{1_{ij}}(n)$	the element of matrix $\mathbf{W}_1$ situated on the i^{th} row and j^{th} column
λ	a fixed learning rate
$\lambda(n)$	a time-varying learning rate
$\nabla J(n)$	the gradient of $J(n)$
$E\{A\}$	the expected value of A
$sign(x)$	the sign of x.

Chapter 1

Introduction

During the past decades intensive research efforts have been done in the area of speech recognition and synthesis to keep up with the growing demands set by the speech processing applications. In particular the speech synthesis from text has gained an increased interest due to its wide range of practical applications.

A text-to-speech (TTS) synthesizer can be defined as a system capable to read any written text aloud [26]. Systems able to pronounce words or sentences are already commercially available such as the ones used for announcements in the train stations. However, these systems only produce speech by concatenation of few isolated words usually selected from a very limited dictionary. It is not practical to record and store all the words of a language as speech signals in order to be able to generate any sentence from that language[1]. A more feasible solution is to implement automatic systems able to generate speech from the phonetic transcription of an input text.

The text-to-phoneme (TTP) mapping, also called grapheme-to-phoneme (GTP) conversion, is a preliminary step in the text-to-speech synthesis and it highly affects the degree of naturalness and understanding of synthetic speech. With the text-to-phoneme mapping, the words are converted into their phonetic transcriptions and synthetic speech is generated from the corresponding phonemes. Attempts to directly transform the letters of a written text into the corresponding speech signal, without the grapheme-to-phoneme conversion have been also proposed in the literature. Such a system was proposed in [18] for languages of the minority communities. For these languages there is no, or only limited, phonetic knowledge available such that the implementation of text-to-phoneme mapping system would be almost impossible. However, these attempts do not reduce the

[1] A given word might have for instance several different intonations which must be kept into the device's memory. Storing a large dictionary in audio format might require large available memory.

importance of the text-to-phoneme mapping. Such a direct text-to-sound system would not be capable, for instance, to generate written word pronunciations needed in printed language learning books.

There is a large number of potential applications of the text-to-speech systems such as: telecommunications services, language learning, aid to impaired persons, human machine interaction, etc. In the telecommunication industry, for instance, the text-to-speech technology can be used to implement systems able to access text information over the telephone line. Different text messages, such as e-mail, facsimile or other large databases can be remotely accessed. It is more economical to store large information databases in text format than as digitized speech. Having a telephone conversation with a speech or hearing impaired person is another interesting application of the text-to-speech systems. The text written by such person with a keyboard can be converted into speech signal and transmitted over the telephone line.

A text-to-speech synthesizer can be a very helpful tool in learning new languages. A language learning book can be scanned and transformed into an electronic text through an optical character recognition (OCR) system. The text can be then used as the input into the text-to-speech system which can provide spoken language lessons. Of course, in such a case the text-to-speech synthesizer should be able to pronounce words in two languages and the problem of multilingual text-to-phoneme mapping arises.

Through text-to-speech synthesis newspapers and books can be transformed into speech signals such that they can be available also to blind people.

Human machine interaction is another interesting application of the text-to-speech synthesizers. For instance multimodal user interfaces (UI) for mobile devices represent an important research topic nowadays. In such user interfaces the automatic speech recognition (ASR) can be combined with the text-to-speech systems to implement voice command capabilities and feedback to the user.

Text-to-speech technology has a large number of possible applications but there are still some limitations that prevent the successful implementation of some of these applications. One limitation is the naturalness of the generated speech. While in some applications, such as remotely accessing text messages, the naturalness of the speech signal is not of main importance, in other applications (for instance learning new languages), naturalness is one of the most important aspects. In building a successful TTS system, capable to generate speech that sounds natural, many aspects must be addressed. For example pitch and pause are two very important properties of a speech signal and the generation of a synthesized utterance with correct pitch and pause sequences is one of the goals in the design of a text-to-speech system. Stress and intonation play a very important role in the

naturalness of the generated speech signal. Even if the other modules of the text-to-speech system are implemented with high accuracy the wrong stress and intonation could decrease the naturalness of the generated speech. An illustrative example was given in [29] for the two words *Connally* and *Connelli*. While both words have almost identical phonetic transcription the first have initial stress and the other have penultimate stress. Speech naturalness and understanding are also linked to the performance of the text-to-phoneme mapping process. Due to this fact the phoneme accuracy of the grapheme-to-phoneme conversion plays an important role in most of the applications. If the transcription of a written text into its corresponding phoneme string contains many errors the synthesized speech signal does not sound natural.

Another limitation imposed on the text-to-speech synthesizers is the low memory and computational complexity. Most of the devices that include text-to-speech technology are portable and suffer from limited memory and processing power. It is then natural to search for text-to-speech solutions that necessitate low hardware resources.

The amount of language resources[2] is another aspect that must be taken into account in the design and implementation of a text-to-speech synthesizer. While for some well studied languages, such as English and French, there are many language resources available, such as large training dictionaries, for other languages there is still a limited amount of such data available. Text-to-speech systems able to learn the characteristics of a certain language from very limited resources are therefore of great interest.

1.1 Overview of the thesis

The thesis consists in three main parts. In Chapter 2 the neural network structures used to build our text-to-phoneme mapping system are described in detail. The chapter starts with an overview of neural networks providing background information necessary to understand our reason to use neural networks for grapheme-to-phoneme conversion. We continue the chapter with the description of several existing neural network structures emphasizing their advantages and possible limitations in the context of the TTP mapping problem. Two new fast training algorithms are introduced and analyzed in detail.

Chapter 3 is dedicated to the problem of monolingual text-to-phoneme mapping. First, we give an overview of several text-to-phoneme mapping systems previously published in the open literature. The advantages and disadvantages of these systems are discussed and

[2]By the term "language resources" we define here all the necessary information, about a certain language, that are needed to design and implement a text-to-phoneme mapping system, such as: the number of available words with known phonetic transcription, punctuation and prosodic information.

their performances are outlined. The chapter continues with a comparison of the results obtained with the different neural networks introduced in Chapter 2. A study of the effect of different letter encoding schemes, to the phoneme accuracy, is presented next. The results of this study are also used in the next chapter for building a multilingual grapheme-to-phoneme conversion system.

In Chapter 4 the problem of bilingual text-to-phoneme mapping is addressed. The chapter begins with a short introduction in the field of multilingual text-to-phoneme mapping pointing out several different practical applications. A number of different approaches to the problem of language identification from text as well as some multilingual grapheme-to-phoneme conversion systems, which have been previously published, are described. The chapter continues with the description of our bilingual text-to-phoneme mapping system. We describe the pre-processing of the training database, the letter and phoneme encoding schemes as well as the structure of our bilingual grapheme-to-phoneme module. A new hybrid module for language identification from text is proposed and the implementation details are given. At the end of the chapter we describe the experiments done with our proposed grapheme-to-phoneme conversion system for English and French languages. The accuracy of the language identification from text as well as the phoneme accuracy of the proposed bilingual grapheme-to-phoneme system are presented.

1.2 Author's contribution

The author's contribution in the field of neural networks and text-to-phoneme mapping is mainly in Chapter 2, Chapter 3 and Chapter 4. The thesis introduced two new training algorithms that increase the convergence speed of the multilayer perceptron neural network. The multilayer perceptron with variable learning rate (MLPVLR) uses an adaptive learning rate in the training phase which reduces the required number of training iterations. In the transform domain multilayer perceptron neural network, several groups of adjacent inputs are transformed through the Digital Cosine Transform (DCT). Due to this input transformation and the use of a time-varying learning rate the transform domain multilayer perceptron have also higher convergence speed compared to the multilayer perceptron neural network. Increased convergence speed can be useful in several different applications where a fast adaptive system must be implemented. In the context of grapheme-to-phoneme conversion, a short convergence time gives the possibility to use smaller training dictionaries. This can be useful for languages for which only small linguistic resources are available.

A study of different letter encoding schemes is given in the thesis. In this study it

was shown that by random encoding of the input letters the memory load of a text-to-phoneme mapping system based on neural networks is decreased without reducing too much the phoneme accuracy.

A novel hybrid system for language recognition from text has been also introduced in the thesis. This language identification module was used to build a bilingual grapheme-to-phoneme conversion system. The advantage of our bilingual grapheme-to-phoneme conversion system consists in its modularity and the possibility to include new languages without retraining the entire system.

The author's publications, most related to the topic of this thesis, are:

1. E. B. Bilcu, J. Astola - A Hybrid Neural Network for Language Identification from Text, in Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2006, [7].

2. E. B. Bilcu, J. Astola - Neural Networks with Random Letter Codes for Text-To-Phoneme Mapping and Small Training Dictionary, in Proceedings of the 14th European Signal Processing Conference, EUSIPCO 2006, [8]

3. E. B. Bilcu, J. Astola - Improved Hybrid Approach for Bilingual Language Recognition from Text, in Proceedings of the the Fifth IEEE International Symposium on Image and Signal Processing and Analysis, ISPA 2007, [9].

4. E. B. Bilcu, J. Astola - A Hybrid Approach to Bilingual Text-To-Phoneme Mapping,- in Facta Electronics and Energetics journal 2008, [10].

5. E. B. Bilcu, J. Astola, J. Saarinen - A Hybrid Neural Network Rule/Based System for Bilingual Text-To-Phoneme Mapping, in Proceedings of the 14th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2004, [11].

6. E. B. Bilcu, J. Astola, J. Saarinen - Comparative Study of Letter Encoding for Text-To-Phoneme Mapping, in Proceedings of the 13th European Signal Processing Conference, EUSIPCO 2005, [12].

7. E. B. Bilcu, J. Suontausta, J. Saarinen - A New Transform Domain Neural Network for Text-To-Phoneme Mapping, in Proceedings of the 6th WSEAS Multiconference on Circuits, Systems, Communications and Computers, WSEAS-CSCC 2002, [13].

8. E. B. Bilcu, J. Suontausta, J. Saarinen - Application of Neural Networks for Text-To-Phoneme Mapping, in Proceedings of the XI European Signal and Image Processing Conference, EUSIPCO 2002, [14].

9. E. B. Bilcu, J. Suontausta, J. Saarinen - A Study on Different Neural Network Architectures Applied to Text-To-Phoneme Mapping, in Proceedings of the 3rd IEEE International Symposium on Image and Signal Processing and Analysis, ISPA 2003, [15]

10. E. B. Bilcu, J. Suontausta, J. Saarinen - Text-To-Phoneme Mapping Using a Fast Neural Network with Adaptive Learning Rate in Proceedings of the 7th WSEAS Multiconference on Circuits, Systems, Communications and Computers, WSEAS-CSCC 2003, [16].

The author has done the basic derivations, the experimental work and most of the writing work in all these publications. The author fulfilled the publication task with the supervisor and the co-authors of the papers.

Chapter 2

An Overview of Neural Networks in Speech Processing

Neural networks belong to the general class of adaptive systems designed to solve various problems in pattern recognition, prediction, optimization and control.

In the case of pattern classification, the task is to associate a certain input pattern to one class from a set of several predefined classes. The input pattern can be for instance a set of features extracted from a speech signal, a vector containing the numerical representation of several adjacent letters of a text or other predefined patterns depending on the application at hand. Usually in the case of pattern classification, a set of training data, which contains a number of input-class pairs, is available. The neural network is first trained on this available data set and after that it can be used for classification.

In the clustering or categorization problem, there is no available training data which contains input-class pairs. In this case the neural network is trained to learn the similarity between the input patterns. Similar input patterns are grouped into the same class.

Function approximation is another important application of neural networks and it is required in many engineering and scientific tasks. In this application the neural networks are trained to model the dependence between two non-independent sets of numerical values (scalars or vectors).

Many problems from a large number of different signal processing fields can be formulated as optimization problems. In the optimization task the goal is to minimize or maximize an objective function subject to one or several constraints.

The research work in the field of neural networks is motivated by their several advantages (see [33] for more details) such as:

1. *Nonlinearity:* the basic structure of a neural network, the neuron, can model linear

and also nonlinear functions[1]. As a consequence a neural network can be used to model linear and nonlinear systems.

2. *Modeling capability:* during training the parameters of a neural network (the so called synaptic weights) are modified such that it can model the dependence between two sets of patterns, the input and the output set.

3. *Adaptivity:* probably, the most important property of a neural network is the adaptivity known also as the learning ability. This property comes from the fact that the connections between the neurons can change their value according to the training algorithm. Neural networks have also the ability to adapt themselves to the changes in the external environment making them useful tools in control applications for instance.

4. *Context:* due to the interconnections between the neurons of a neural network, one particular neuron can be seen as being connected with all other neurons. As a consequence, the context information is naturally included into the structure of the neural network.

In order to design a system based on neural networks, the environment in which the neural network will operate must be modeled. That is, all the information available to the neural network must be known (the type of inputs and the type of desired outputs, if available). This model of the environment in which a particular neural network will operate is known as the learning paradigm. Another important issue in designing a system based on neural networks is to define the learning rules (and the learning algorithm) used to update the synaptic weights of the neural network. There are three main learning paradigms (learning with a teacher, learning without a teacher and hybrid learning) and five basic learning rules (error-correction rules, memory-based rules, Hebbian rules, Boltzmann rules and competitive learning rules) [33].

Learning paradigms

The block diagram of a system trained using the learning with a teacher paradigm is illustrated in Fig. 2.1. The *teacher* provides the desired response that is the optimum response of the neural network to a particular input. The error signal is computed as the difference between the desired optimal output and the actual output of the network and it is used to modify the synaptic connections of the neural network.

While in the learning with a teacher paradigm the adaptation of the neural network is done knowing the optimum output provided by a teacher, in the learning without a

[1]This is influenced by the activation functions used at the output of the neurons.

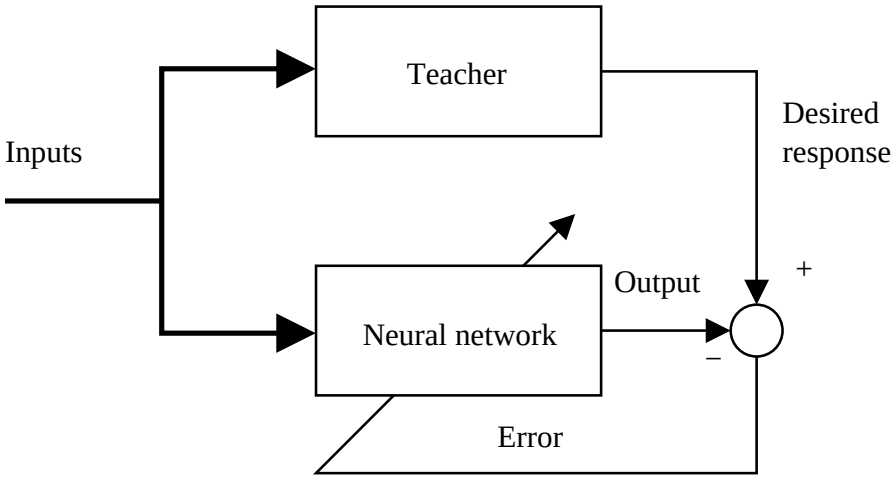

Figure 2.1: The block diagram illustrating the principle of learning with a teacher.

teacher paradigm the learning process is not supervised by a teacher. There are two subdivisions of the learning without a teacher paradigm: the reinforcement learning and the unsupervised learning. In the reinforcement learning[2] the learning system contains a *critic* that observes the environment and controls the learning of the neural network through a heuristic reinforcement signal. As one can see, from the block diagram of Fig. 2.2 the *critic* observes the changes in the environment, done by the neural network, and produces the heuristic reinforcement. In the unsupervised learning (also called self-organizing learning) there is no teacher or *critic* to supervise the training of the neural network. The training of the neural network is made rather based on some quality measure that is task independent.

The third learning paradigm is the hybrid learning which is a combination of the unsupervised and the supervised learning.

Learning rules

As we have mentioned above, the learning rules specify the mechanism to update the neural network weights.

Error-correction learning rules: are applied to neural networks which are trained to estimate a desired output signal from an input sequence[3]. A simplified block diagram of a neural network, trained using an error-correction rule, is depicted in Fig. 2.3.

The input pattern $\mathbf{x}(n)$ is applied to the neural network that responds with the output $\mathbf{y}(n)$. The error between a desired pattern $\mathbf{d}(n)$ and the output pattern is computed and

[2]Also called neurodynamic programming.

[3]The input and the output signals can be sequences of scalars or vectors.

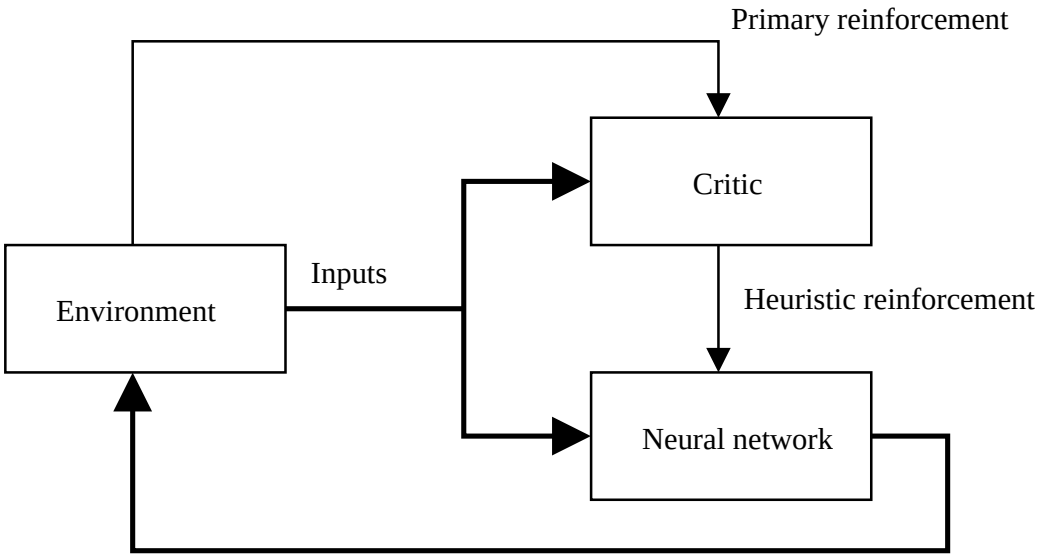

Figure 2.2: The block diagram illustrating the principle of reinforcement learning.

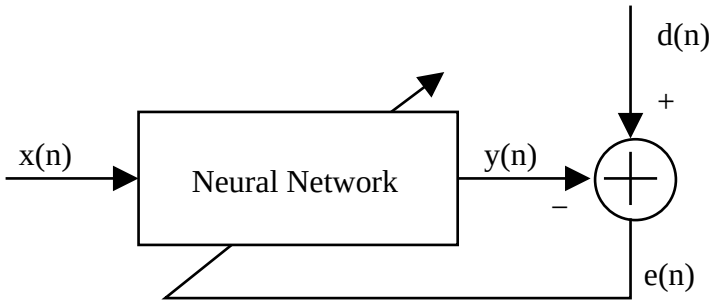

Figure 2.3: The block diagram of a neural network trained using the error-correction rule. The input sequence of the neural network is $\mathbf{x}(n)$, the response of the neural network is $\mathbf{y}(n)$, the desired response is $\mathbf{d}(n)$ and the output error is $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}(n)$.

used to adapt the free parameters of the neural network. In this manner, the neural network is trained to estimate as close as possible the sequence of desired patterns from the sequence of input patterns.

Memory-based learning: the whole training sequence together with the corresponding desired sequence are kept into the memory as pairs $\{\mathbf{x}(n), \mathbf{d}(n)\}$ (n being the time instant or the index of the input-output pair[4]). A test input pattern that does not belong to the training set is classified according to the training pairs that are in its neighborhood.

[4]In some cases it appears as subscript $\{\mathbf{x}_n, \mathbf{d}_n\}$.

Lets consider the simple case of classification of an input pattern $\mathbf{x}_{test}$ from the test set. When the test input pattern $\mathbf{x}_{test}$ is to be classified the minimum distance between $\mathbf{x}_{test}$ and all input patterns from the training set is calculated [33]:

$$\min_i D\left(\mathbf{x}(i), \mathbf{x}_{test}\right), \quad i = 1, \ldots, n$$

with $D\left(\mathbf{x}(i), \mathbf{x}_{test}\right)$ being the distance between $\mathbf{x}(i)$ and $\mathbf{x}_{test}$.

The pattern $\mathbf{x}(i)$ that gives the smallest distance to $\mathbf{x}_{test}$ is selected as its neighbor and the test pattern is classified into the same class as $\mathbf{x}(i)$. Another variant of this method is the method of *k-nearest neighbor*. In this method a number of k patterns from the training set, that are the closest to $\mathbf{x}_{test}$, are selected as neighbors of $\mathbf{x}_{test}$. The k neighbors usually belong to several different classes. The input pattern $\mathbf{x}_{test}$ is classified to the class that have more members in the k selected training patterns.

Hebbian learning: has its roots in the Hebb's *postulate of learning*. This postulate was modified and changed into the following two rules (see [33] and the references therein):

1. If two neurons on both sides of a synaptic connection are simultaneously activated then the value of that synaptic connection is increased.

2. If the two neurons situated at the extremities of a synaptic connection are activated asynchronously, then the value of that synaptic connection is decreased or the synaptic connection is eliminated.

There are several variations of Hebbian learning given by the different implementations of the function used to modify the synaptic weights.

Competitive learning rule: the output neurons of the neural network compete between each other in order to become active. This means that, at a certain time instant, just one output neuron of the neural network is active as opposed to other learning rules where several output neurons can be active in the same time.

A simplified block diagram of such neural network implementing competitive learning is shown in Fig. 2.4. The simple structure shown in Fig. 2.4 contains just one input layer and an output layer and no hidden neurons[5]. Notice the feed-forward connections from the inputs to the output neurons as well as the lateral connections between the output neurons. The output that is active at a certain moment is decided by the implemented competing rule. Various, competing rules and methods for synaptic weight adaptation can be implemented (see [33] for a more detailed description of this method).

[5]Networks with hidden layers can be trained using competitive learning as well.

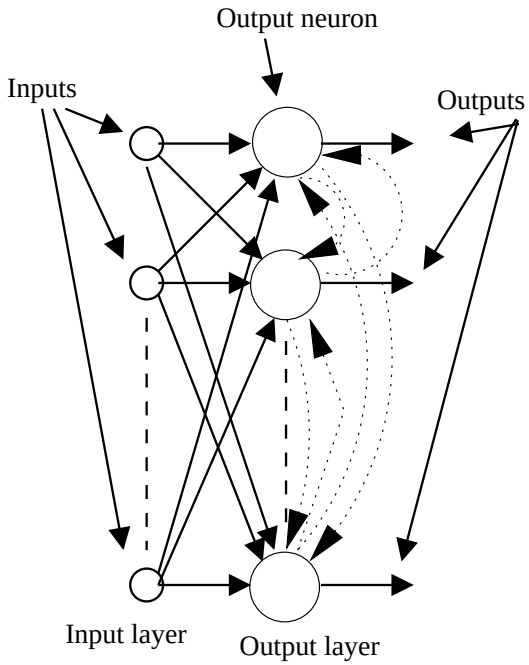

Figure 2.4: A neural network architecture implementing the competitive learning. The connections between the output neurons are depicted by dotted lines. The number of inputs and outputs are generally different.

Boltzmann learning rule: is a stochastic learning algorithm which has its roots in statistical mechanics [33]. A Boltzmann machine, a neural network trained by Boltzmann learning rule is a recurrent structure in which the neurons are either in the "on" or in the "off" state. The main feature of this kind of neural network is the energy function that is computed as the sum of the energy functions of each neurons. The training of the Boltzmann machine is done by flipping the state of each neuron until the network reach the steady-state.

2.1 Neural networks for speech processing

Speech processing is one example of application where neural networks have been implemented with success. This is due to the fact that many problems in speech processing (such as TTP mapping) require modeling non-linear functions, between a set of input-output patterns. In this chapter of the thesis an overview of the most common types of neural networks (NN) used in the speech processing applications is presented. The structures and the training algorithms for each type of neural network implemented for the

problem of text-to-phoneme mapping are described in detail. The discussion is focused on the following three neural network structures: the multilayer perceptron (MLP) neural network, the recurrent neural network (RNN) and the bidirectional recurrent neural network (BRNN). The theoretical aspects of neural networks are described here, in a general level, and the implementation of these models to the specific problems of monolingual and bilingual TTP mapping are presented in Chapter 3 and Chapter 4.

For the MLP neural network, besides the standard well known approach that uses a constant learning rate in the training process [33], two new approaches are introduced in this section. The first one, called the transform domain multilayer perceptron (TDMLP) neural network uses modified inputs compared to the MLP case. A similar method was introduced in [51] but differs from our approach due to the fact that the orthogonal transform is applied to the entire input vector while in our approach the orthogonal transform is applied separately on groups of inputs. The second novel implementation discussed here is the multilayer perceptron neural network with adaptive learning rate that is introduced in order to increase the convergence speed. Also, this kind of approach was previously addressed in the open literature [38]. Our implementation possess the advantage of lower complexity and simplicity of implementation compared to the existing ones.

2.2 The multilayer perceptron neural network

In this section the multilayer perceptron neural network, which is widely used to approximate nonlinear functions, is described. The structure of the neural network and the well known training algorithm called error back-propagation are detailed here. The theoretical considerations presented in this section are later used in the next chapters where the practical problem of text-to-phoneme mapping is addressed.

The block diagram of the MLP neural network is depicted in Fig. 2.5. The neural network has one input layer, several hidden layers and one output layer. The input layer provides the inputs, shown as small circles in Fig. 2.5, to the neural network while the neurons from the output layer generate the outputs of the neural network. The neurons from the intermediate layers receive their inputs from the neurons situated in the previous layer and send their outputs to the neurons situated on the next layer. Since these neurons (depicted by large dashed circles in Fig. 2.5) do not have any connections outside the neural network they are usually called hidden neurons. If each neuron of each layer (input, hidden or output) of the neural network is connected to all neurons of the next and previous layers, the neural network is called a fully connected neural network. All

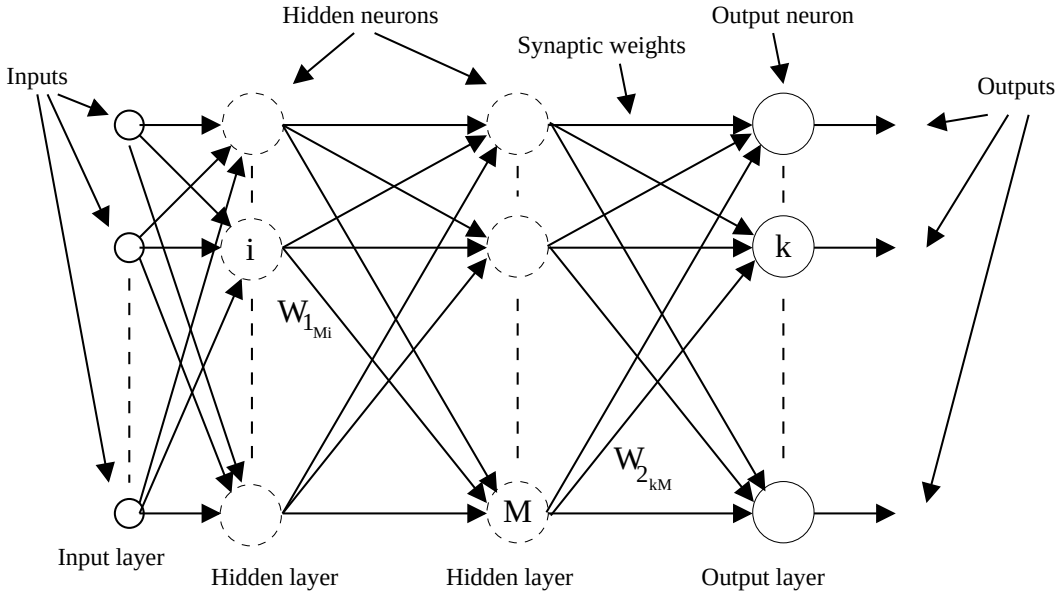

Figure 2.5: The block diagram of the multilayer perceptron neural network showing the basic elements such as: inputs, neurons, synaptic weights and outputs (the bias terms are not shown).

the implementations and experiments presented in this thesis are done with such fully connected neural networks.

Before we proceed with the description of the training algorithm, we first have a look in more detail at the neural network structure. To this end, in Fig. 2.6, the structure of a neuron showing all its elements in detail is presented. The connection between neurons situated on different layers is ensured by the synaptic weight denoted with $w_{i,j}(n)$, where i and j represent the fact that the weight connects the neuron i to the neuron j and n is the time instant (also known as iteration number). The induced local field of the neuron, denoted as $y_j(n)$ in Fig. 2.6, is computed as the weighted sum:

$$y_j(n) = \sum_{i=1}^{N} w_{j,i}(n) x_i(n)$$

where $x_i(n)$ can be an input of the neural network or an output of another neuron (depending if the current neuron belongs to a hidden layer or to the output layer).

The output $y_i^{(a)}(n)$ of the neuron[6] is obtained from its induced local field through the activation function $h(\cdot)$. The nonlinear behavior of the neurons and thus of the neural

[6]The exponent (a) denotes the fact that $y_i^{(a)}(n)$ is obtained from the induced local field $y_i(n)$ after applying the activation function.

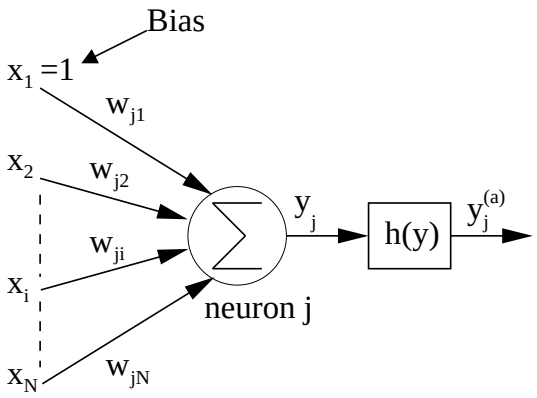

Figure 2.6: The structure of a neuron showing its basic elements.

network is obtained due to the activation function $h(\cdot)$ at the output of the neuron. If these activation functions were linear (for instance $h(x) = ax + b$ with a and b constants) for all neurons of the neural network, the function between the inputs and the outputs of the neural network would also be linear. As there is limited interest in multilayer perceptron neural networks realizing linear function, nonlinear activation functions are used.

The following types of activation functions are common in practice (see also Fig. 2.7):

1. The *threshold activation function* for which the relation between the induced local field $y_j(n)$ of the neuron and the output of the neuron $y_j^{(a)}(n)$ is given by:

$$y_j^{(a)}(n) = h\left(y_j(n)\right) = \begin{cases} 0, & \text{if } \ y_j(n) < 0 \\ 1, & \text{if } \ y_j(n) \geq 0 \end{cases} \tag{2.1}$$

   or by the symmetrical functional:

$$y_j^{(a)}(n) = h\left(y_j(n)\right) = \begin{cases} -1, & \text{if } \ y_j(n) < 0 \\ 1, & \text{if } \ y_j(n) \geq 0 \end{cases} \tag{2.2}$$

2. The class of the *sigmoid activation functions* includes the logistic sigmoid activation function and the hyperbolic tangent activation function.

   The logistic sigmoid activation function is expressed by the following equation:

$$y_j^{(a)}(n) = h\left(y_j(n)\right) = \frac{1}{1 + \exp(-Ay_j(n))} \tag{2.3}$$

   where A is a constant.

The hyperbolic tangent activation function is given by:

$$y_j^{(a)}(n) = h\left(y_j(n)\right) = \frac{1 - \exp(-Ay_j(n))}{1 + \exp(-Ay_j(n))} \tag{2.4}$$

where $y_j(n)$ is the induced local field of the neuron j and $y_j^{(a)}(n)$ is the output of the neuron j (see Fig. 2.6).

3. The *softmax activation function* expressed by:

$$y_j^{(a)}(n) = \frac{\exp(-Ay_j(n))}{\sum_{i=1}^{N} \exp(-Ay_i(n))} \tag{2.5}$$

where A is a constant, N is the number of the neurons on the current layer, $y_i(n)$ is the induced local field of the i^{th} neuron from the current layer and $y_j^{(a)}(n)$ is the output of the current neuron.

The softmax activation function gives a good approximation of the class posterior probabilities [17, 14, 33]. Due to this fact, it is often used in the output neurons for classification problems.

The above mentioned activation functions are not the only ones that can be implemented in a MLP neural network in order to obtain a nonlinear model. These functions are the most known due to the fact that they are easily implementable in software.

2.2.1 Training the multilayer perceptron neural network

The training algorithm used in this thesis, for the multilayer perceptron neural network, is the error back-propagation with momentum which belongs to the class of gradient-based techniques [14], [33]. The weights of the adaptive model are modified such that the position of the system on the error surface moves in the opposite direction of the cost function gradient. The movement is controlled by the so-called learning rate that sets the speed of convergence and also controls the stability and the modeling accuracy of the MLP neural network.

The training algorithm is described here for a fully connected neural network which has one input layer, one hidden layer and one output layer. The block diagram illustrating the error back-propagation algorithm is depicted in Fig. 2.8 and the algorithm is derived from the minimization of the following cost function:

$$J(n) = \sum_{i=1}^{P} e_i(n).$$

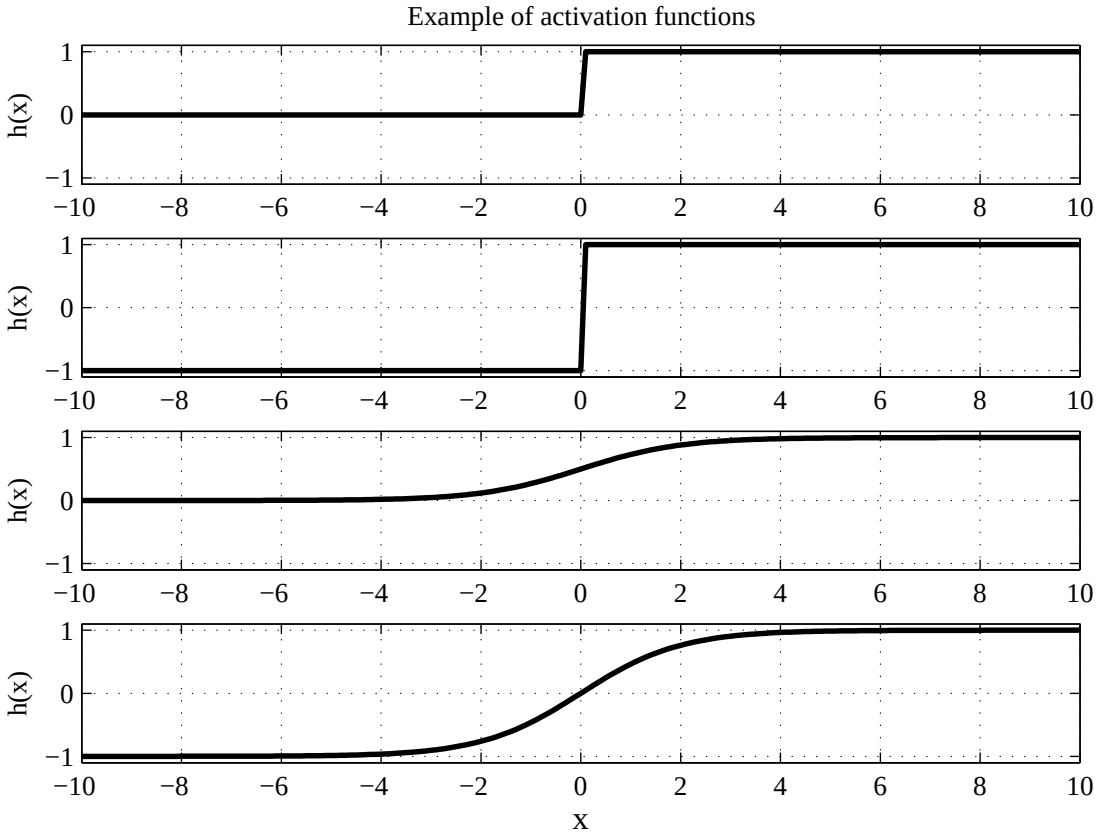

Figure 2.7: Examples of some commonly used activation functions. From top to bottom: the threshold activation function, the symmetrical threshold activation function, the logistic sigmoid activation function and the tangent activation function.

where $e_i(n)$ is the error of the i^{th} output neuron at time instant n.

The values of the synaptic weights are changed such that $J(n)$ is minimized at each iteration[7]. In order to obtain the equations to update the weights, the derivative of $J(n)$ with respect to the synaptic weights are computed. The error back-propagation algorithm is described by the following steps:

1. **Forward processing:** the inputs are forward propagated from the input layer to the output layer of the neural network.

   - Compute the induced local fields of the hidden neurons:

$$\mathbf{y}_1(n) = \mathbf{W}_1(n)\mathbf{x}(n), \tag{2.6}$$

[7]At every iteration one input pattern is presented to the neural network such that one iteration corresponds to one input pattern.

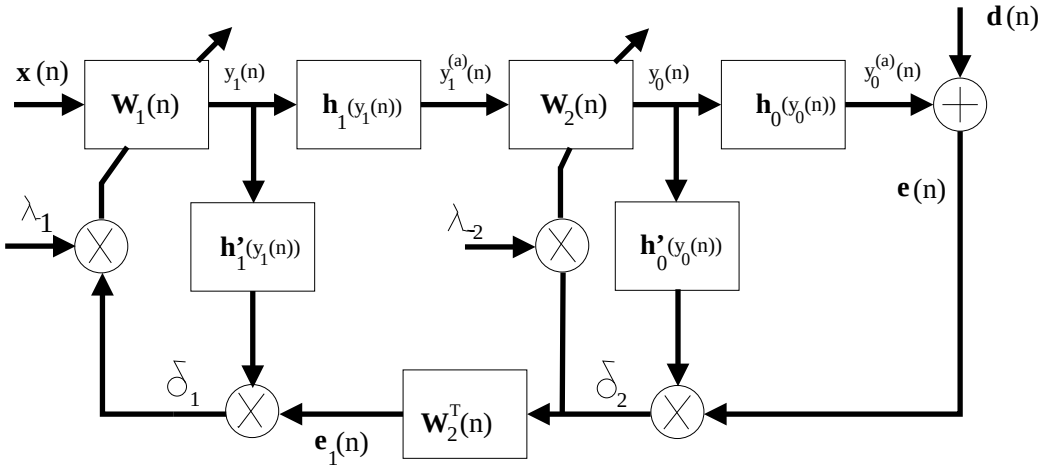

Figure 2.8: The block diagram illustrating the error back-propagation algorithm for a three layered multilayer perceptron neural network.

where $\mathbf{W}_1(n)$ is the $M \times N$ matrix that contains the synaptic connections from the input layer to the hidden layer and $\mathbf{x}(n) = [x_1(n), x_2(n), \ldots, x_N(n)]^t$ is the input vector with N elements[8]. The exponent t represents the transposition operator. The number of neurons on the hidden layer is M and also the length of the column vector $\mathbf{y}_1$ is M.

- Compute the outputs of the hidden layer:

$$\mathbf{y}_1^{(a)}(n) = h_1\left(\mathbf{y}_1(n)\right), \tag{2.7}$$

where $h_1(\cdot)$ is the activation function used in the hidden layer (see equations (2.1) to (2.5)) and $\mathbf{y}_1^{(a)}(n)$ is a length M vector containing the outputs of the hidden neurons.

- The induced local fields of the output neurons are given by the following equation:

$$\mathbf{y}_0(n) = \mathbf{W}_2(n)\mathbf{y}_1^{(a)}(n), \tag{2.8}$$

where $\mathbf{W}_2(n)$ is the $M \times P$ matrix containing the synaptic connections from the hidden layer to the output layer, $\mathbf{y}_0(n)$ is the length P vector that contains the induced local fields of the output neurons and P is the number of output

[8] A bias term is usually included in $\mathbf{x}(n)$. It is not explicitly shown here for simplicity of the presentation.

neurons[9].

- The output of the network is given by the following transformation:

$$\mathbf{y}_0^{(a)}(n) = h_0\left(\mathbf{y}_0(n)\right), \tag{2.9}$$

where $h_0(\cdot)$ is the activation function used in the output layer.

- Compute the output error vector:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}_0^{(a)}(n), \tag{2.10}$$

where $\mathbf{d}(n)$ is the length P vector of the desired outputs of the neural network.

2. **Error back-propagation:** the error is back-propagated through the neural network from the output to the input. During this processing procedure the updates are computed for both weight matrices $\mathbf{W}_1(n)$ and $\mathbf{W}_2(n)$ according to:

- For the output layer the weights changes are computed as follows:

$$\Delta W_{2_{ij}}(n) = \alpha \Delta W_{2_{ij}}(n-1) + \lambda_2 \delta_{2_i}(n) y_{1_j}^{(a)}(n) \tag{2.11}$$

where $\Delta W_{2_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_2(n)$, α is the momentum constant, λ_2 is the learning rate used to update the synaptic weights $\mathbf{W}_2(n)$, $y_{1_j}^{(a)}(n)$ is the j^{th} element of the vector $\mathbf{y}_1^{(a)}(n)$ and $\delta_{2_i}(n)$ is computed by:

$$\delta_{2_i}(n) = e_i(n) h_0'(y_{0_i}(n)), \tag{2.12}$$

with $h_0'(\cdot)$ being the derivative of the activation function $h_0(\cdot)$, $e_i(n)$ being the i^{th} element of the error vector $\mathbf{e}(n)$ and $y_{0_i}(n)$ is the i^{th} element of $\mathbf{y}_0(n)$.

The synaptic weights $\mathbf{W}_2(n)$ are then updated using the following equation:

$$\mathbf{W}_2(n+1) = \mathbf{W}_2(n) + \Delta \mathbf{W}_2(n), \tag{2.13}$$

and the elements of the matrix $\mathbf{W}_2(n)$ are computed in (2.11).

- For the hidden layer the weight changes are computed as follows:

$$\Delta W_{1_{ij}}(n) = \alpha \Delta W_{1_{ij}}(n-1) + \lambda_1 \delta_{1_i}(n) x_j(n) \tag{2.14}$$

where $\Delta W_{1_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_1(n)$, α is the momentum constant, λ_1 is the learning rate used to update the synaptic

[9]The vector $\mathbf{y}_1^{(a)}(n)$ contains also a bias term which was not explicitly shown in order to keep the presentation simpler.

weights $\mathbf{W}_1(n)$, $x_j(n)$ is the j^{th} element of the input vector $\mathbf{x}(n)$ and $\delta_{1_i}(n)$ is computed as follows:

$$\delta_{1_i}(n) = e_{1_i}(n)h'_1(y_{1_i}(n)), \tag{2.15}$$

where $h'_1(\cdot)$ is the derivative of the activation function $h_1(\cdot)$, $e_{1_i}(n)$ is the i^{th} element of the error vector $\mathbf{e}_1(n)$ in the hidden layer and $y_{1_i}(n)$ is the i^{th} element of the vector $\mathbf{y}_1(n)$ computed in (2.6).

We have to emphasize here that in (2.11) and (2.14) different learning rates, λ_2 and respectively λ_1, are used. These learning rates can be equals or different depending on the application of the neural network.

The errors of the hidden units (that are used in (2.15)) are computed by back-propagation of the output error. The following equation implements the computation of the hidden layer error:

$$\mathbf{e}_1(n) = \mathbf{W}_2^t(n)\delta_2(n), \tag{2.16}$$

where the elements of the vector $\delta_2(n)$ are computed in (2.12).

- Finally the synaptic weights $\mathbf{W}_1(n)$ are updated using the following equation:

$$\mathbf{W}_1(n+1) = \mathbf{W}_1(n) + \Delta\mathbf{W}_1(n), \tag{2.17}$$

The above training algorithm is iteratively applied to modify the value of the synaptic weights of the MLP neural network. There are two main methods used to update the synaptic weights. In the online training, the weight changes are computed at every iteration and the weights are immediately updated[10].

The main disadvantage of the back-propagation with momentum training algorithm is its slow convergence. Several methods have been proposed (in the open literature) to increase the convergence speed of the back-propagation with momentum algorithm. Two such methods, a transform domain implementation and an approach using variable learning rate, are introduced in the next sections of this chapter.

2.2.2 Transform domain multilayer perceptron neural network

In this section, the transform domain multilayer perceptron neural network is presented. The idea to implement the multilayer perceptron in transform domain comes from the observation that faster training of a neural network can be achieved if the eigenvalue spread of

[10] Another possibility is to update the weights in the, so called, batch mode. In the batch mode the weight changes are computed at each iteration and they are accumulated for several consecutive iterations (called the batch interval). The synaptic weights are updated at the end of the batch interval.

the input autocorrelation matrix is reduced[11] [33, 47, 51]. To reduce the eigenvalue spread, orthogonal inputs with equal power should be used [47]. One way to do orthogonalization of the inputs is to use orthogonal codes (as we will see in more details in the next chapter). Another way is to use non-orthogonal codes and some orthogonalization mechanism such as the DCT transform. The new structure possess the advantage of compactness and fast training of the synaptic weights. In Chapter 3, the new transform domain multilayer perceptron neural network is applied for the problem of text-to-phoneme mapping and it shows better speed of convergence during training than the multilayer perceptron neural network with comparable phoneme accuracy. Another similar TDMLP neural network structure was proposed in [51]. The method proposed here differs from the one in [51] in the way the input vector is transformed. While the model from [51] was introduced for applications where the input vector is shifted by one element, from one iteration to another, our method is better suited for the problem of text-to-phoneme mapping where the inputs are shifted with more steps between two adjacent iterations.

We start from the multilayer perceptron neural network described in the previous subsection and we introduce the transformation of the input vector resulting in the transform domain multilayer perceptron neural network. By using this transformation of the input vector, the complexity of the neural network is reduced[12], and the convergence speed is increased as compared to the multilayer perceptron neural network.

The block diagram of the proposed TDMLP neural network architecture is depicted in Fig. 2.9, where the blocks denoted by $l_i(n)$, $i = -2, \ldots, 2$ represents the vectors containing groups of inputs. For instance in Fig. 2.9 the inputs of the neural network are partitioned into 5 groups. The number of input groups is not limited to five as in Fig. 2.9 but one can use different number of groups depending on the application at hand[13]. The blocks denoted by DCT represent the *Discrete Cosine Transform* used to orthogonalize each input vector (it contains the DCT matrix). The lines labeled with H represent the connections of length H (for instance the vector corresponding to the current input letter can be written as $l_0(n) = [l_{0_1}(n), l_{0_2}(n), \ldots, l_{0_H}(n)]$), the three-layered neural network is fully connected and have $5H + 1$ inputs, N hidden neurons and P output neurons. One should note that, unlike the approach in [51], in this implementation the vectors containing groups of inputs are independently orthogonalized and not the entire input

[11] The fastest convergence is obtained in the ideal case of unity eigenvalue spread.

[12] This will become more clear in Chapter 3 where the problem of text-to-phoneme mapping is described in more detail.

[13] The reason to split the inputs into smaller groups will become more clear in the next chapter. Here we just mention that each group corresponds to an input letter in the text-to-phoneme mapping mapping application.

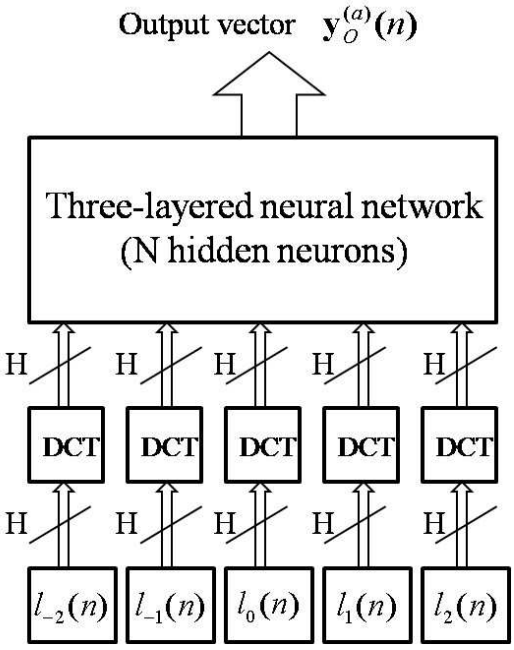

Figure 2.9: The block diagram of our proposed transform domain multilayer perceptron neural network. The neural network is depicted as a single block and the inputs and their orthogonal transformation are detailed. The fact that the inputs are split into several groups is clearly shown. The blocks denoted as $l_{-2}(n)$, $l_{-1}(n)$, $l_0(n)$, $l_1(n)$ and $l_2(n)$ represents the code vectors of the corresponding letters.

vector (see for comparison Fig. 2.9 and Fig. 2.11). Each of the DCT matrices has the same $H \times H$ dimension and contains the same elements. The elements of these matrices can be computed before starting the training of the neural network and they can be saved, in order to be used at the testing step.

The scheme that uses partial orthogonalization of the input vector (by separately transforming the input letters using the DCT) has the advantage of smaller memory load. This is due to the fact that the DCT matrix to be stored in the implementation shown in Fig. 2.9 has $H \times H$ elements and for the architecture depicted in Fig. 2.11 one must store a $5H \times 5H$ DCT matrix.

Since the inputs of the neural network, in the case of the text-to-phoneme mapping application, are letters, one alternative, that also reduces the computational complexity, is to store the transform vectors of the letters (the outputs of the DCT blocks in Fig. 2.9). For English language for example, there are 27 letters that can be encoded with binary vectors of length 5. As a consequence, if this method is chosen, one must store

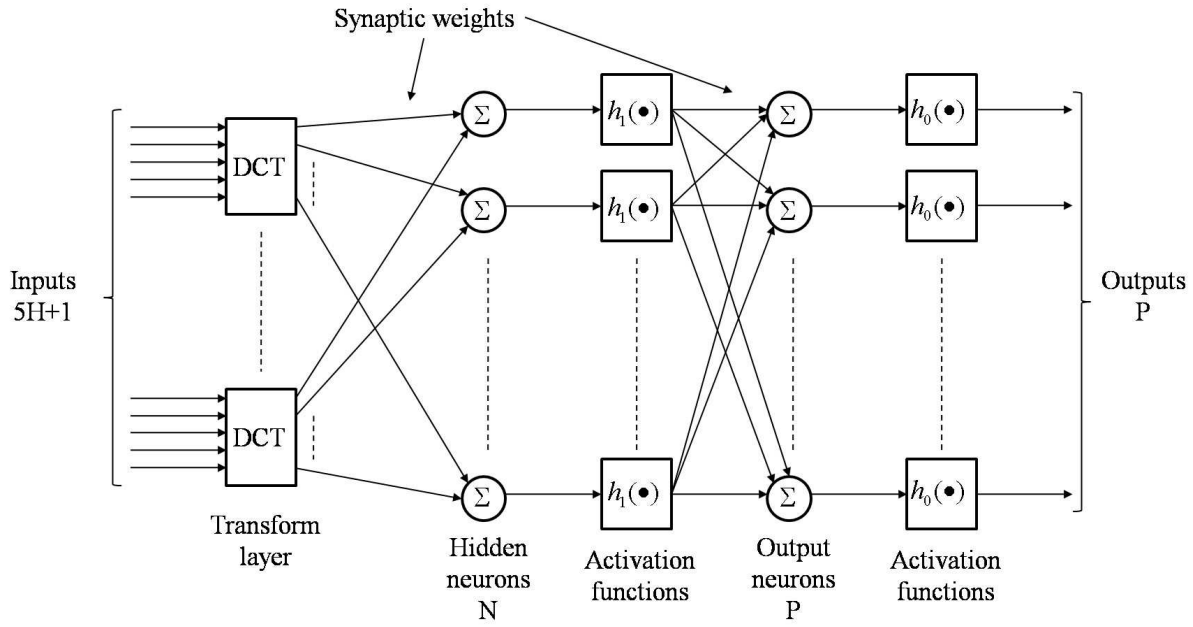

Figure 2.10: The detailed structure of the transform domain multilayer perceptron neural network.

27×5 real numbers (27 vectors of length 5). On the other hand, if this scheme is used in conjunction to Fig. 2.11, the length of the input vector is 5×5 and the number of different possible combinations of 5 input letters is 27^5. This leads to an increase in the memory load compared to our approach.

Training the transform domain multilayer perceptron neural network

The transform domain multilayer perceptron neural network can be trained with a similar algorithm as the standard MLP neural network. However, there are several differences between the two algorithms in time and transform domain. The differences arise from the use of the DCT transform. Due to this fact, the complete modified back-propagation algorithm in transform domain emphasizing the differences with its time domain counterpart is detailed here (see also Fig. 2.10):

1. **Forward processing**

   - Take the current input vector and split it into 5 equal parts $l_{-2}(n), \ldots, l_2(n)$ which are transformed in the DCT domain:

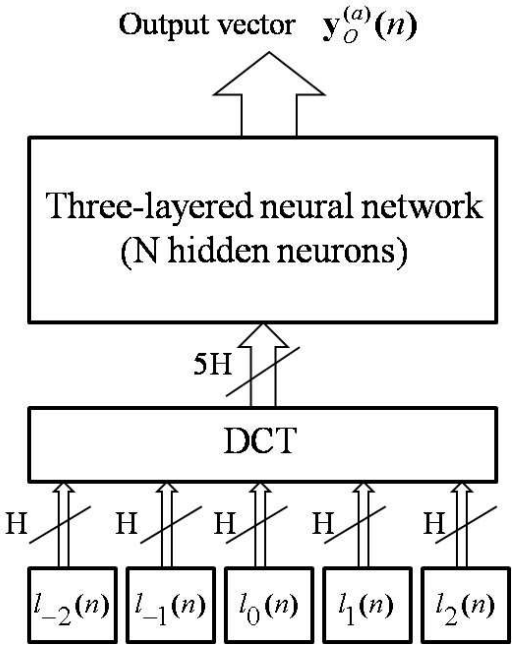

Figure 2.11: The block diagram of the transform domain multilayer perceptron neural network proposed in [51]. The neural network is depicted as a single block and the input vector composed of 5 adjacent letters is transformed through the block denoted as DCT.

$$\mathbf{t}_i(n) = \mathbf{D} * \mathbf{l}_i(n), \quad i = -2:2 \tag{2.18}$$

where $\mathbf{l}_i(n)$ represents the vector corresponding to the i^{th} part of the input vector, $\mathbf{D}$ is the $H \times H$ matrix of the DCT transform and $\mathbf{t}_i(n)$ is the transform domain vector of the i^{th} part of the input vector (see Fig. 2.9).

This first processing step does not appear in the time domain implementation. Moreover, it is also different from the algorithm proposed in [51] since here the DCT is applied to groups of inputs and in [51] the DCT is applied to all inputs together[14].

- Compute the induced local fields of the hidden neurons:

$$\mathbf{y}_1(n) = \mathbf{W}_1(n)\mathbf{x}(n), \tag{2.19}$$

where $\mathbf{W}_1(n)$ is the matrix that contains the synaptic connections from the input layer to the hidden layer and $\mathbf{x}(n) = \left[1, \mathbf{t}_{-2}^t(n), \ldots, \mathbf{t}_2^t(n)\right]^t$ is the $(5H+1)\times$

[14]Fourier or other transforms can be used as well.

1 input vector obtained by concatenation of all five transform domain vectors corresponding to the parts of the input vector plus the bias. The exponent t represents the transposition operator.

- Compute the outputs of the hidden layer:

$$\mathbf{y}_1^{(a)}(n) = h_1\left(\mathbf{y}_1(n)\right), \tag{2.20}$$

where $h_1(\cdot)$ is the activation function of the hidden layer.

- The induced local fields of the output neurons are given by the following equation:

$$\mathbf{y}_0(n) = \mathbf{W}_2(n)\mathbf{y}_1^{(a)}(n), \tag{2.21}$$

where $\mathbf{W}_2(n)$ is the matrix containing the synaptic connections from the hidden layer to the output layer, and $\mathbf{y}_1^{(a)}(n)$ is the output of the hidden layer computed in (2.20).

- The output of the network is given by:

$$\mathbf{y}_0^{(a)}(n) = h_0\left(\mathbf{y}_0(n)\right), \tag{2.22}$$

where $h_0(\cdot)$ is the output activation function.

- Compute the output error vector:

$$\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{y}_0^{(a)}(n), \tag{2.23}$$

with $\mathbf{d}(n)$ being the desired vector at time instant n.

As one can see from (2.18)-(2.23) orthogonalization is introduced only at the input of the neural network. In [51] it was shown that introducing orthogonalization also in the hidden layer slightly increases the convergence speed. However, the slight performance improvement does not justify the important increase of the computational complexity. For this reason, we limited our discussion to the approach that uses orthogonalization just in the input layer.

2. **Error back-propagation** Back-propagate the output error through the neural network from the output to the input and compute the weight changes.

- For the output layer the weights changes are computed as follows:

$$\Delta W_{2_{ij}}(n) = \alpha \Delta W_{2_{ij}}(n-1) + \lambda_2 \delta_{2_i}(n) y_{1_j}^{(a)}(n) \tag{2.24}$$

where $\Delta W_{2_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_2(n)$, α is the momentum constant, λ_2 is the learning rate used to update the synaptic weights $\mathbf{W}_2(n)$, $y_{1_j}^{(a)}(n)$ is the j^{th} element of the vector $\mathbf{y}_1^{(a)}(n)$ and $\delta_{2_i}(n)$ is computed as follows:

$$\delta_{2_i}(n) = e_i(n)h_0'(y_{0_i}(n)), \tag{2.25}$$

where $h_0'(\cdot)$ is the derivative of the activation function $h_0(\cdot)$, $e_i(n)$ is the i^{th} element of the error vector $\mathbf{e}(n)$ and $y_{0_i}(n)$ is the i^{th} element of $\mathbf{y}_0(n)$.

The synaptic weights $\mathbf{W}_2(n)$ are updated using the following equation:

$$\mathbf{W}_2(n+1) = \mathbf{W}_2(n) + \Delta\mathbf{W}_2(n), \tag{2.26}$$

- For the input layer the weights changes are computed as follows:

$$\Delta W_{1_{ij}}(n) = \alpha\Delta W_{1_{ij}}(n-1) + \lambda_{1_j}(n)\delta_{1_i}(n)x_j(n) \tag{2.27}$$

where $\Delta W_{1_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_1(n)$, α is the momentum constant, $\lambda_{1_j}(n)$ is the variable learning rate used to update the synaptic weights $\mathbf{W}_1(n)$, $x_j(n)$ is the j^{th} element of the vector $\mathbf{x}(n)$ (see (2.19)) and $\delta_{1_i}(n)$ is computed as follows:

$$\delta_{1_i}(n) = e_{h_i}(n)h_1'(y_{1_i}(n)), \tag{2.28}$$

where $h_1'(\cdot)$ is the derivative of the activation function $h_1(\cdot)$, $e_{h_i}(n)$ is the i^{th} element of the error vector $\mathbf{e}_h(n)$ in the hidden layer and $y_{1_i}(n)$ is the i^{th} element of the vector $\mathbf{y}_1(n)$ from (2.19).

The errors of the hidden units are computed as in the standard back-propagation algorithm and are given by the following equation:

$$\mathbf{e}_h(n) = \mathbf{W}_2^t(n)\delta_2(n), \tag{2.29}$$

where the elements of the vector $\delta_2(n)$ are computed in (2.26). The learning rate $\lambda_{1_j}(n)$ in (2.28) is given by:

$$\lambda_{1_j}(n) = \frac{\lambda_1}{P_{x_j}(n)}, \tag{2.30}$$

where $P_{x_j}(n)$ is the power estimate of the j^{th} element of the vector $\mathbf{x}(n)$.

The normalization of the learning rate is justified by the fact that an orthonormal transform is achieved by both the orthogonal transform and the normalization of the learning rate[15] [33].

[15]It is possible also to include normalization into the input layer of the neural network by normalization of each of its inputs.

The following formula for the power estimates computation was used in our implementation:

$$P_{x_j}(n+1) = \beta P_{x_j}(n) + (1-\beta)x_j^2(n), \tag{2.31}$$

with β being a constant in the interval $(0, 1)$.

The synaptic weights $\mathbf{W}_1(n)$ are updated as follows:

$$\mathbf{W}_1(n+1) = \mathbf{W}_1(n) + \Delta\mathbf{W}_1(n), \tag{2.32}$$

As we can see, from the above description of the training algorithm, the extra computations that are introduced, when the transform domain multileyer perceptron is used, are the DCT transformation (see (2.18)), the normalization with the power estimates (see (2.30)) and computation of $P_{x_j}(n)$ in (2.31). Therefore, the computational complexity of the new TDMLP structure is slightly higher compared to the computational complexity of the standard multilayer perceptron neural network.

Also by using the new structure, the DCT matrices have to be saved and used for mapping. Since all 5 DCT matrices contain the same elements, it is necessary to store just the elements of one DCT matrix of size $H \times H$.

Moreover, in text-to-phoneme mapping application, it is possible to make even more reduction in the computational complexity and memory load of the TDMLP neural network. These reductions are based on the fact that the inputs of the neural network, in the text-to-phoneme mapping application, cannot take any value since they represent letters from a certain alphabet (several inputs are grouped in order to encode a letter). The way how to reduce further the complexity of the transform domain multilayer perceptron model will be explained in more detail in Chapter 3.

It should be noted that in the approach proposed here the transform layer appears just at the input of the neural network whereas in one of the approaches presented in [51], the DCT transform is used also in the hidden layer. For the application described in this thesis, we have found that such complicated structure does not give important improvement in terms of convergence speed or phoneme accuracy.

2.2.3 Training the multilayer perceptron using an adaptive learning rate

In order to use neural networks, in real time applications, the requirement is to implement fast training algorithms and models which use a small amount of memory and can provide accurate modeling. For instance in the text-to-phoneme mapping application a fast

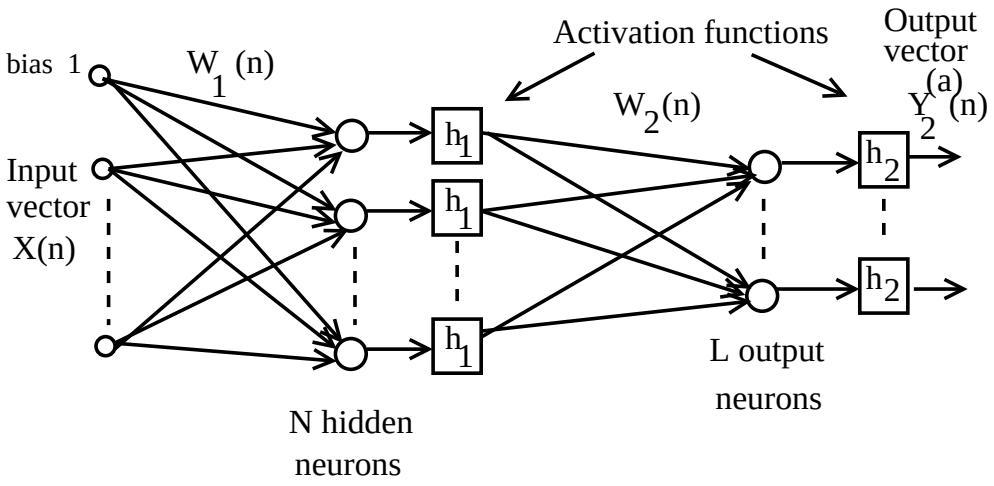

Figure 2.12: The detailed diagram of a three layered multilayer perceptron neural network.

convergence of the neural network means a small training dictionary or smaller number of epochs. More specifically, fast convergence means fewer number of iterations required by the neural network to go to the stability point. When the neural network is trained in on-line mode the number of iterations equals the number of letters in the training dictionary and a fast convergence is similar with smaller training dictionary.

The error back-propagation algorithm is one of the most popular method for training the multilayer perceptron neural networks [17], [33]. Although the effectiveness of the back-propagation algorithm is a proven fact, many researchers from this field often find its convergence rate to be too slow. The convergence speed depends most of all on the choices of the parameters in the algorithm. The learning rate determines the size of the weight updates made at each iteration therefore influences the rate of convergence. But the optimum value of the learning rate depends on the problem to be solved, therefore, it is not easy to choose an appropriate value of the learning rate which ensures fast convergence. Most of the time the learning rate is chosen based on trial and error.

The development of fast learning algorithms has attracted the attention of many researchers and, as a result, several training algorithms with adaptive learning rates have been introduced [6, 16, 38, 77]. In this section, a new algorithm for learning rate adaptation of the multilayer perceptron neural network is described. The new algorithm has the advantage of small complexity and simplicity of implementation while providing improved performance of the neural network.

A detailed block diagram of a three layered fully connected MLP neural network which contains a number of M inputs, N hidden neurons and L outputs is depicted in Fig. 2.12.

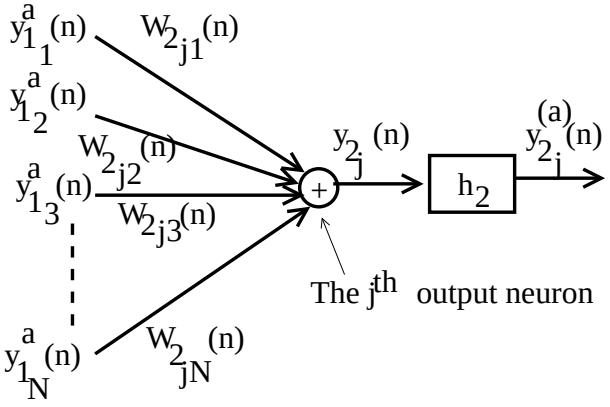

Figure 2.13: The detailed diagram of an output neuron showing all the connections with other neurons and the notations used in the text to describe the training algorithm.

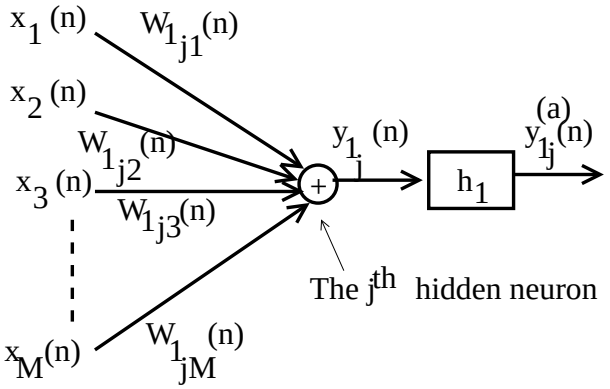

Figure 2.14: The detailed diagram of a hidden neuron showing all the connections with other neurons and the notations used in the text to describe the training algorithm.

The $M \times 1$ input vector is denoted by $\mathbf{x}(n)$, the $L \times 1$ output vector is denoted by $\mathbf{y}_2^{(a)}(n)$ and the matrices of the synaptic connections are denoted by $\mathbf{W}_1(n)$ and $\mathbf{W}_2(n)$ for the hidden neurons and for the output neurons respectively.

The following training algorithm, with adaptive learning rate, is introduced here which possess a low computational complexity and improved convergence speed. In this section of the thesis the theoretical aspects of the new algorithm are described and the experimental results are presented in the next chapter.

1. **Forward processing:**

- Compute the induced local fields of the hidden neurons as follows:

$$\mathbf{y}_1(n) = \mathbf{W}_1(n) \begin{bmatrix} 1 \\ \mathbf{x}(n) \end{bmatrix}. \tag{2.33}$$

  where 1 represents the input bias term.

- Compute the outputs of the hidden neurons:

$$\mathbf{y}_1^{(a)}(n) = h_1\left(\mathbf{y}_1(n)\right). \tag{2.34}$$

  where $h_1(\cdot)$ is the activation function used in the hidden layer.

- The induced local fields of the output neurons are obtained using the formula:

$$\mathbf{y}_2(n) = \mathbf{W}_2(n) \begin{bmatrix} 1 \\ \mathbf{y}_1^{(a)}(n) \end{bmatrix}. \tag{2.35}$$

  where 1 represents the bias term of the hidden layer.

- The outputs of the network are then given by:

$$\mathbf{y}_2^{(a)}(n) = h_2\left(\mathbf{y}_2(n)\right). \tag{2.36}$$

  where $h_2(\cdot)$ is the output activation function.

- The final step in the forward processing is the computation of the output error vector:

$$\mathbf{e}_2(n) = \mathbf{d}(n) - \mathbf{y}_2^{(a)}(n). \tag{2.37}$$

  and $\mathbf{d}(n)$ is the desired vector.

The forward procedure is identical with the one used in the standard error back-propagation algorithm with fixed learning rate. The difference between the algorithm that uses a fixed learning rate and the one that uses an adaptive learning rate is in the backward processing part.

2. **Backward processing:** During this process the output error computed in (2.37) is back-propagated through the neural network in order to compute the weight changes.

   - For the output layer the changes of the synaptic weights are computed as follows:

$$\Delta W_{2_{ij}}(n) = \alpha \Delta W_{2_{ij}}(n-1) + \lambda_{2_i}(n)\delta_{2_i}(n)y_{1_j}^{(a)}(n) \tag{2.38}$$

where $\Delta W_{2_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_2(n)$, α is the momentum constant, $\lambda_{2_i}(n)$ is the time-varying learning rate, $y_{1_j}^{(a)}(n)$ is the j^{th} element of the vector $\mathbf{y}_1^{(a)}(n)$ and $\delta_{2_i}(n)$ is computed as:

$$\delta_{2_i}(n) = e_{2_i}(n)h_2'(y_{2_i}(n)), \tag{2.39}$$

with $h_2'(\cdot)$ being the derivative of the activation function $h_2(\cdot)$, $e_{2_i}(n)$ the i^{th} element of the error vector $\mathbf{e}_2(n)$ and $y_{2_i}(n)$ the i^{th} element of $\mathbf{y}_2(n)$.

We emphasize here that the synaptic weights are updated in (2.38) using a time-varying learning rate instead of a constant learning rate as in (2.24).

The synaptic weights $\mathbf{W}_2(n)$ are updated as:

$$\mathbf{W}_2(n+1) = \mathbf{W}_2(n) + \Delta\mathbf{W}_2(n), \tag{2.40}$$

- For the input layer the weights changes are computed as follows:

$$\Delta W_{1_{ij}}(n) = \alpha\Delta W_{1_{ij}}(n-1) + \lambda_{1_i}(n)\delta_{1_i}(n)x_j(n) \tag{2.41}$$

where $\Delta W_{1_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_1(n)$, α is the momentum constant, $\lambda_{1_i}(n)$ is the time-varying learning rate, $x_j(n)$ is the j^{th} element of the input vector $\mathbf{x}(n)$ and $\delta_{1_i}(n)$ is computed by the following formula:

$$\delta_{1_i}(n) = e_{1_i}(n)h_1'(y_{1_i}(n)), \tag{2.42}$$

where $h_1'(\cdot)$ is the derivative of the activation function $h_1(\cdot)$, $e_{1_i}(n)$ is the i^{th} element of the error vector $\mathbf{e}_1(n)$ in the hidden layer and $y_{1_i}(n)$ is the i^{th} element of the vector $\mathbf{y}_1(n)$ from (2.33).

The errors of the hidden units are given by:

$$\mathbf{e}_1(n) = \mathbf{W}_2^t(n)\delta_2(n), \tag{2.43}$$

where the elements of the vector $\delta_2(n)$ are computed in (2.39).

The synaptic weights $\mathbf{W}_1(n)$ are updated as:

$$\mathbf{W}_1(n+1) = \mathbf{W}_1(n) + \Delta\mathbf{W}_1(n), \tag{2.44}$$

The error back-propagation algorithm presented in Section 2.2.1 for training the MLP uses the same constant learning rate to adapt all the synaptic weights of the network. It is well known, that better results in terms of convergence speed are obtained when different

synaptic weights are trained with different time-varying learning rates [38, 6]. The error function of the multilayer perceptron represents a nonlinear surface in the weights space and the trajectory of the algorithm, on this error surface, can be decomposed along each of the weight axes. If the same learning rate is used to update each of the neural network weights, at a certain iteration, the trajectory would not point directly to the error surface minima. This leads to a longer path through the error surface which increases the training time. Moreover a fixed and small learning rate would require many iterations steps to reach the error surface minima while a too large learning rate would cause oscillations around the error surface minima. Due to this fact, in the algorithm described by (2.33) to (2.44), the learning rates are time-varying and different in the hidden and in the output layer.

The main disadvantage when each synaptic weight is trained with its own learning rate is the increased computational complexity and memory load (the value of each learning rate must be computed and stored during the adaptation and there is one such learning rate for each synaptic weight). In order to reduce this disadvantage, a new algorithm is introduced in order to adaptively adjust the learning rate without increasing too much the memory load and the computational complexity. To this end, we analyze the detailed diagram of an output neuron depicted in Fig. 2.13 and the detailed diagram of a hidden neuron depicted in Fig. 2.14. From Fig. 2.13 we can see that the output of the j^{th} output neuron is influenced by all the outputs $y_{1_i}^{(a)}(n)$ of the hidden neurons and the synaptic connections $W_{2_{j1}}(n)$ ... $W_{2_{jN}}(n)$ (where $W_{2_{j1}}(n)$ ... $W_{2_{jN}}(n)$ are the elements of the j^{th} row of the matrix $\mathbf{W}_2(n)$). Here, we propose to use not different learning rates for each synaptic weight but different learning rates for every neurons in the network (hidden neurons and output neurons). For instance, in Fig. 2.13 all the synaptic weights $W_{2_{j1}}(n)$ ... $W_{2_{jN}}(n)$ are updated using the same time-variable learning rate denoted by $\lambda_{2_j}(n)$. Since in the three layered multilayer perceptron neural network, depicted in Fig. 2.12, there are N hidden neurons and L output neurons the number of different learning rates is $N + L$ which does not increase too much the complexity of the algorithm. For the adaptation of each of the learning rates we propose the following scheme:

For the output neurons: At each iteration we compute the squared error at the output of each neuron using the following time-averaging formula:

$$E_{2_i}(n) = 0.9E_{2_i}(n-1) + 0.1e_{2_i}^2(n), \quad i = 1, \ldots, L \tag{2.45}$$

where $e_{2_i}(n)$ is the i^{th} element of the output error computed in (2.37) (the constants 0.9 and 0.1 have been selected by trial and error in order to obtain the best compromise between speed of convergence and modeling capabilities).

The learning rates corresponding to each output neuron are modified as follows:

$$\lambda_{2_i}(n) = \beta\lambda_{2_i}(n-1) - sign(S_{2_i}(n))\gamma e^{-S_{2_i}(n)}, \quad i = 1, \ldots, L \tag{2.46}$$

where $S_{2_i}(n) = \frac{E_{2_i}(n)-E_{2_i}(n-1)}{E_{2_i}(n)}$, $E_{2_i}(n-1)$ is the square error at the previous iteration computed using (2.45) and $sign(x)$ is the signum function. The constant parameters β and γ can be used to tune the convergence speed of the neural network.

For the hidden neurons: To update the learning rates corresponding to each of the hidden neurons we use the same algorithm as for the output neurons except that the errors are those from the hidden layer:

$$E_{1_i}(n) = 0.9E_{1_i}(n-1) + 0.1e_{1_i}^2(n), \quad i = 1, \ldots, N \tag{2.47}$$

where $e_{1_i}(n)$ is the i^{th} element of the error in the hidden layer computed in (2.43).

The learning rates corresponding to each hidden neuron are updated as follows:

$$\lambda_{1_i}(n) = \beta\lambda_{1_i}(n-1) - sign(S_{1_i}(n))\gamma e^{-S_{1_i}(n)}, \quad i = 1, \ldots, N \tag{2.48}$$

where $S_{1_i} = \frac{E_{1_i}(n)-E_{1_i}(n-1)}{E_{1_i}(n)}$ and $E_{1_i}(n-1)$ is the square error at the previous iteration computed using (2.47).

As one can see from (2.45)-(2.48) the learning rate corresponding to a neuron is modified in the following manner: the mean squared error at the output of the neuron is estimated at each iteration by (2.45) for the output neurons and by (2.48) for the hidden neurons. If the estimated mean squared error of the neuron decreases, it means that the neuron goes toward its steady state. In this case, it is possible to use a larger learning rate that will increase its convergence speed. On the contrary, when the value of $S_{2_i}(n)$ in (2.46) and $S_{1_i}(n)$ in (2.48) are positive (the corresponding output mean squared error increases), the neuron does not converge. In this case, the learning rate must be decreased in order to drive the neuron to its stability region. The quantity by which the learning rate is changed at one iteration is also proportional to the increase or decrease in the corresponding output mean squared error.

We emphasize here that, in [38] four heuristical rules for learning rate adaptation of the neural networks, that will ensure faster training, have been defined. One of these rules specifies that every weight should have its own individual learning rate. In our proposed algorithm this rule is partially satisfied in the sense that every neuron has its own learning rate. All the weights that enter on the same neuron have equal learning rates. This implementation was chosen for complexity reasons (fewer different learning rates necessitate less memory and computational effort).

The above described training algorithm with adaptive learning rate has the advantage of increased convergence speed with a computational complexity closer to the complexity of the standard multilayer perceptron neural network. The results obtained using this training algorithm for the problem of text-to-phoneme mapping are shown in the next chapter.

2.3 The recurrent neural network

Recurrent neural networks incorporate one or more feedback loops in their structure. These feedback loops can be global (for instance one or more outputs of the neural network are returned at the input of the neural network) or can be local when several outputs from a certain layer are returned as inputs to the same layer. Moreover, if one starts from the structure of the multilayer perceptron described in the above sections (see for example Fig. 2.5), it is clear that there is a very large variety of possible combinations of feedback loops (there are many possible combinations of feedback connections between the neurons and layers.).

Due to their feedback connections, the recurrent neural networks incorporate not only the direct mapping of the inputs to the output values, but also they incorporate a temporal dependence between the inputs and the outputs of the neural network. For instance, in the case of a two layer RNN with feedback loops from the output to the input, as the one depicted in Fig. 2.15, there are some delay units incorporated into the feedback loops. Due to these delays, the outputs of the neural network at time instant n are returned to the input of the neural network at time instant $n + 1$. This fact can make the recurrent neural networks more suitable for a wide range of applications such as: prediction, channel equalization, etc [33]. Elman introduced in 1990 the use of the recurrent neural network to discover word boundaries in a continuous stream of phonemes. The input to the neural network represents the current phoneme. The output represents the network's best guess as to what the next phoneme is in the sequence. The role of the context units is to provide the network with "dynamic memory" so as to encode the information contained in the sequence of phonemes which is relevant to the prediction.

In some applications, in order to obtain better modeling results, a large number of inputs must be used into the neural network[16]. As a consequence, in order to maintain the same number of synaptic weights, when more input letters are considered, the number

[16]Text-to-phoneme mapping is one such application where several adjacent letters are used as inputs to the neural network model instead of the current letter. The accuracy of the text-to-phoneme transcription increases in this case.

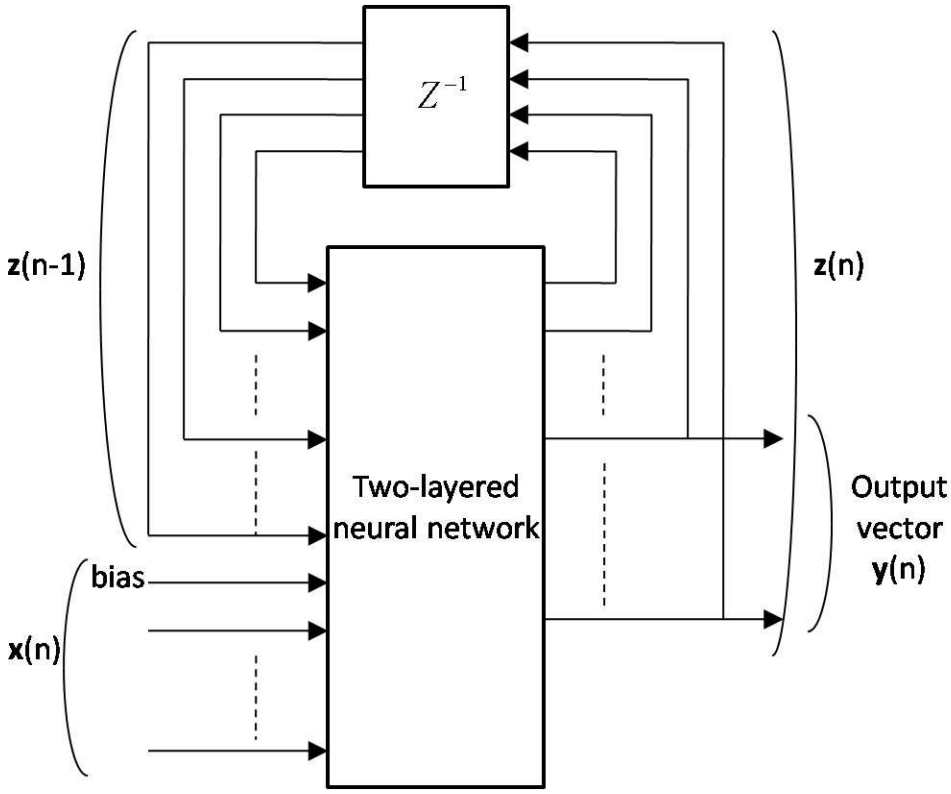

Figure 2.15: The diagram of a fully connected recurrent neural network.

of hidden neurons has to be decreased. To test the possibility to use a small number of inputs, we study the performance of other network architectures such as recurrent neural networks that contains feedback loops which introduce a certain degree of context information.

The structure of the recurrent neural network considered in this thesis is that of a fully connected recurrent neural network as depicted in Fig. 2.15. When the RNN is applied to the problem of text-to-phoneme mapping in Chapter 3, letter context is not explicitly introduced into the input vector. The goal is to study the context dependence incorporated into the feedback loop of the recurrent neural network (see Fig. 2.15). The inputs $\mathbf{x}(n)$ are presented to the network along with the state vector $\mathbf{z}(n-1)$. These two vectors are passed through a standard single layer feed-forward network to give the new state vector $\mathbf{z}(n)$. The outputs of the neural network are obtained from the state vector $\mathbf{z}(n)$[17].

[17]They represent the corresponding phonemes in the TTP mapping application of Chapter 3.

2.3.1 Training the recurrent neural network

There are two methods to train the recurrent neural networks: epoch-wise training and continuous training. In the epoch-wise training of the recurrent neural network, the meaning of the term "epoch" is different than in the case of the multilayer perceptron. In the case of multilayer perceptron one epoch is composed of several time instants while in the case of recurrent neural network one epoch is the interval between two states of the neural network[18]. For instance in the case of MLP one epoch can include all the training iterations such that several training epochs are equivalent to passing the entire dictionary through the neural network many times. In the case of RNN one epoch represents the time interval from the presentation of the input pattern to the neural network until the output is computed and the weights are updated. This is in connection with the training algorithm and the structure of the neural network. For instance, if the back-propagation through time (BPTT) training algorithm with truncation depth of 5 is used, one epoch contains all the computations and iterations that are performed until all the weights are updated (see Fig. 2.16 and the details below) and in fact is similar to 5 consecutive iterations performed in the multilayer perceptron neural network. In the case of continuous training, the synaptic weights of the neural network are changed at each time instant.

A well known algorithm to train recurrent neural networks is the error back-propagation through time algorithm [15, 17, 33, 57, 71, 72, 73]. When the synaptic weights are adapted using the BPTT algorithm it is necessary to back-propagate the error to the past, which means that the past states of the neural network have to be stored. Usually, the number of past states used to train the recurrent neural network is limited to a finite value, for example to 3, 5, or 7 states. The main reason to use the truncation of the past states is to avoid large memory storage. Another very important reason to use the finite truncation of the past states in this thesis, is that we are dealing with the phonetic transcription of isolated words, and therefore an infinite number of past states is not allowed. This is due to the fact that the context dependence between the first letter of the current word and the last letters of the previous word must be avoided (see Chapter 3 for more details). When the first letter of a certain word is the current input into the RNN, the influence of the letters from the previous word has to be removed from the past states of recurrent neural network. Resetting the state vectors at the beginning of each word eliminates the influence of previous words to the current word.

[18] One epoch of the recurrent neural network corresponds with an input pattern of a multilayer perceptron neural network see [33]

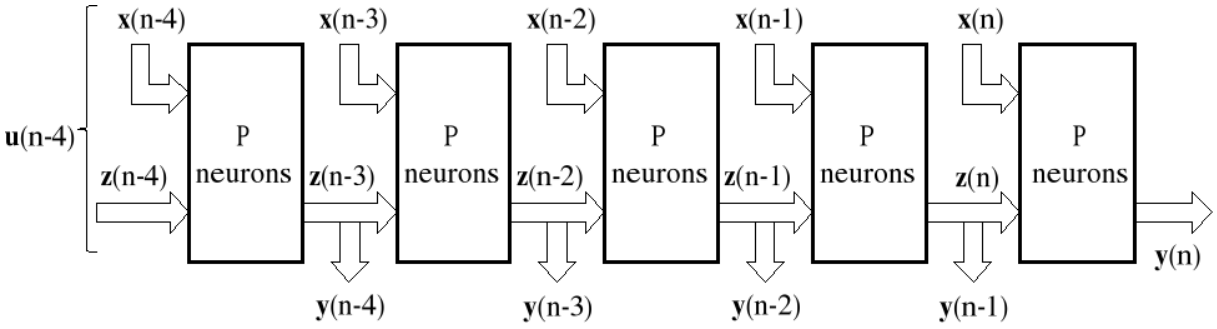

Figure 2.16: The fully connected recurrent neural network unfolded in time. The truncation depth, which is the number of past steps used in the training algorithm, is 5.

For the recurrent neural network there might be not necessary to introduce context dependence into the input vector, as we do in the case of multilayer perceptron neural network when we perform text-to-phoneme mapping mapping, since the output state is feedback to the input. The context dependence is incorporated into the feedback. The structure of the fully connected recurrent neural network used in our experiments is depicted in Fig. 2.15. The input vector $\mathbf{u}(n)$ is formed by the concatenation of the inputs $\mathbf{x}(n)$ and the states $\mathbf{z}(n)$. All outputs are feedback to the input and they represent the network state. The first P states are taken as the outputs of the network and represents the corresponding phoneme.

The training of the recurrent neural networks in this thesis is based on the truncated back-propagation through time algorithm[19]. The training procedure may be derived by unfolding the temporal operation of the network into a layered feed-forward network. In order to maintain a feasible computational complexity, the relevant history of input data and network states are saved for a fixed number of time steps. The number of time steps, for which the data is stored, is called the truncation depth [71, 72, 73]. In this thesis, the experiments were done with recurrent neural networks trained by the BPTT algorithm with a truncation depth of 3 and 5 time steps and the unfolded neural networks with 5 time steps is shown in Fig. 2.16.

The back-propagation through time training algorithm can be described by the following steps[20] (see also Fig. 2.16 for better understanding):

[19]Another training algorithm is the real-time recurrent learning (RTRL) algorithm which is not addressed here.

[20]These steps describe the truncated BPTT algorithm. Other variant of this algorithm is the epochwise BPTT.

1. **Forward propagate the inputs and the states:** The inputs and the states of the recurrent neural network are forward propagated through the unfolded structure shown in Fig. 2.16. For instance in Fig. 2.16 the input into the last block is $\mathbf{x}(n)$ which is the input vector at time instant n while the input into the second last block is $\mathbf{x}(n-1)$ which is the past input vector.

   - The induced local field of the neuron i at time instant l is computed as follows:

   $$v_i(l) = \sum_{k=1}^{N} w_{ki}(l) u_k(l) \tag{2.49}$$

   where $l = \{n-T+1, n-T+2, \ldots, n\}$, $T = 5$ in the case of the NN shown in Fig. 2.16 with truncation depth of 5 and $u_k(n)$ is the k^{th} element of the vector $\mathbf{u}(n)$:

   $$\mathbf{u}(n) = \begin{bmatrix} 1 \\ \mathbf{z}(n) \\ \mathbf{x}(n) \end{bmatrix}. \tag{2.50}$$

   - The output of the neurons are computed applying the activation function to the corresponding induced local fields:

   $$y_i(l) = h\left(v_i(l)\right) \quad i = 1, \ldots, P \quad \text{and} \quad l = n-T+1, \ldots, n \ . \tag{2.51}$$

   where $h(\cdot)$ is the activation function.

   - Compute the output error using the following formula:

   $$e_i(n) = d_i(n) - y_i(n), \quad i = 1, \ldots, P. \tag{2.52}$$

   with $d_i(n)$ being the i^{th} element of the desired vector.

   As we can see from (2.52) the output error is computed just for the last block from Fig. 2.16 that corresponds to the time instant n. Due to this fact, the computations (blocks) for time instants $n-T+1$ to $n-1$ are treated similar to the hidden layers in the multilayer perceptron neural network [71, 72, 73].

2. **Back-propagate the error:** In this processing phase the error computed at time instant n is back-propagated in time until the time instant $n-T+1$. This is similar to back-propagation of the error from the last block until the first processing block in Fig. 2.16.

- Compute the local gradient for neuron j:

$$\delta_j(l) = -\frac{\partial J(n)}{\partial v_j(l)} \quad \forall j \quad \text{and} \quad n - T < l \leq n \ , \tag{2.53}$$

where $v_j(l)$ is the induced local field of the neuron j, T is the truncation depth ($T = 5$ in Fig. 2.16) and $J(n)$ is the cost function minimized by this algorithm. The cost function $J(n)$ is defined as the sum of the output squared errors:

$$J(n) = \frac{1}{2} \sum_{i=1}^{P} e_i^2(n). \tag{2.54}$$

After some mathematical manipulations (2.53) can be written as follows:

$$\delta_j(l) = \begin{cases} h'\left(v_j(l)\right) e_j(l) & \text{for} \quad l = n \\ h'\left(v_j(l)\right) \sum\limits_{k=1}^{P} w_{kj}(l)\delta_k(l+1) & \text{for} \quad n - T < l < n \end{cases} \tag{2.55}$$

- After the error was back-propagated from the time instant n to the time instant $n - T + 1$ the correction that is applied to the synaptic weights of the neuron j can be computed as follows:

$$\Delta w_{ji}(n) = \lambda \sum_{l=n-T+1}^{n} \delta_j(n) u_i(l-1). \tag{2.56}$$

where λ is the learning rate and $u_i(l-1)$ is the input applied to the i^{th} synaptic weight of the neuron at time instant $l - 1$.

As we can see from the above steps, the BPTT algorithm is very similar to the error back-propagation algorithm implemented to train the multilayer perceptron neural network. The unfolded structure of the recurrent neural network with truncation depth of T corresponds to a multilayer perceptron with T hidden layers. The input is forward propagated and the error is back-propagated through the RNN in a similar manner as in the MLP. There are differences between the two neural networks. In the multilayer perceptron neural network all the inputs of a hidden layer are obtained from the outputs of the previous layer. In the recurrent neural network case, the inputs in one processing block, that corresponds to a certain time instant, are composed of outputs from the previous processing block and the current inputs of the neural network. The multilayer perceptron neural networks uses different activation functions for neurons situated on different layers. In the recurrent neural network case since there is basically only one layer of neurons just one type of activation function is used.

2.4 The bidirectional recurrent neural network

As we have seen in the previous section, the recurrent neural network incorporates a feedback loop in its structure. Due to this fact, there is a backward temporal dependence between the output of the network and its inputs at several different time instants. This might be beneficial in some practical applications where the current output not only depends on the actual input but also on several past inputs. For the problem of text-to-phoneme mapping this means that the current translated phoneme depends on the actual input letter and on several adjacent input letters situated on its left-hand side. For such applications as text-to-phoneme mapping it would be beneficial to include into the model also the letters that are situated on the right-hand side of the current letter. In other words, in such applications, there will be beneficial to incorporate a feed-forward loop into the neural network structure.

The main drawback of a regular recurrent neural network is that it incorporates just a weak left side context dependence due to the feedback loops[21]. In order to overcome this limitation, in 1997 a regular recurrent neural network has been extended, by Schuster et all., to a bidirectional recurrent neural network [59]. Splitting the state neurons into two parts will transform a recurrent neural network into a bidirectional recurrent neural network (BRNN). The first part, i.e. the forward states, is responsible for the positive time direction whereas the second part, i.e. the backward states, is responsible for the negative time direction. Backward states are used to introduce context dependence to the right-hand side of the current letter and the forward states introduce the context information from the left-hand side of the current letter.

2.4.1 Training the bidirectional recurrent neural network

The block diagram of an unfolded bidirectional recurrent neural network is depicted in Fig. 2.17. We notice the fact that the forward states are not connected to the backward states. Since the structure of the bidirectional recurrent neural network can be viewed as a connection of two regular recurrent neural networks, the first one for the positive direction and the second one for the negative direction, the synaptic weights of the forward and backward states can be trained using the BPTT algorithm as in the case of recurrent neural networks. However, some differences between the training algorithms for the recurrent neural network and bidirectional recurrent neural network exist due to the

[21] The current recognized phoneme is obtained from the input letter and from the previous recognized phoneme which it is obtained from the corresponding input letter and so on.

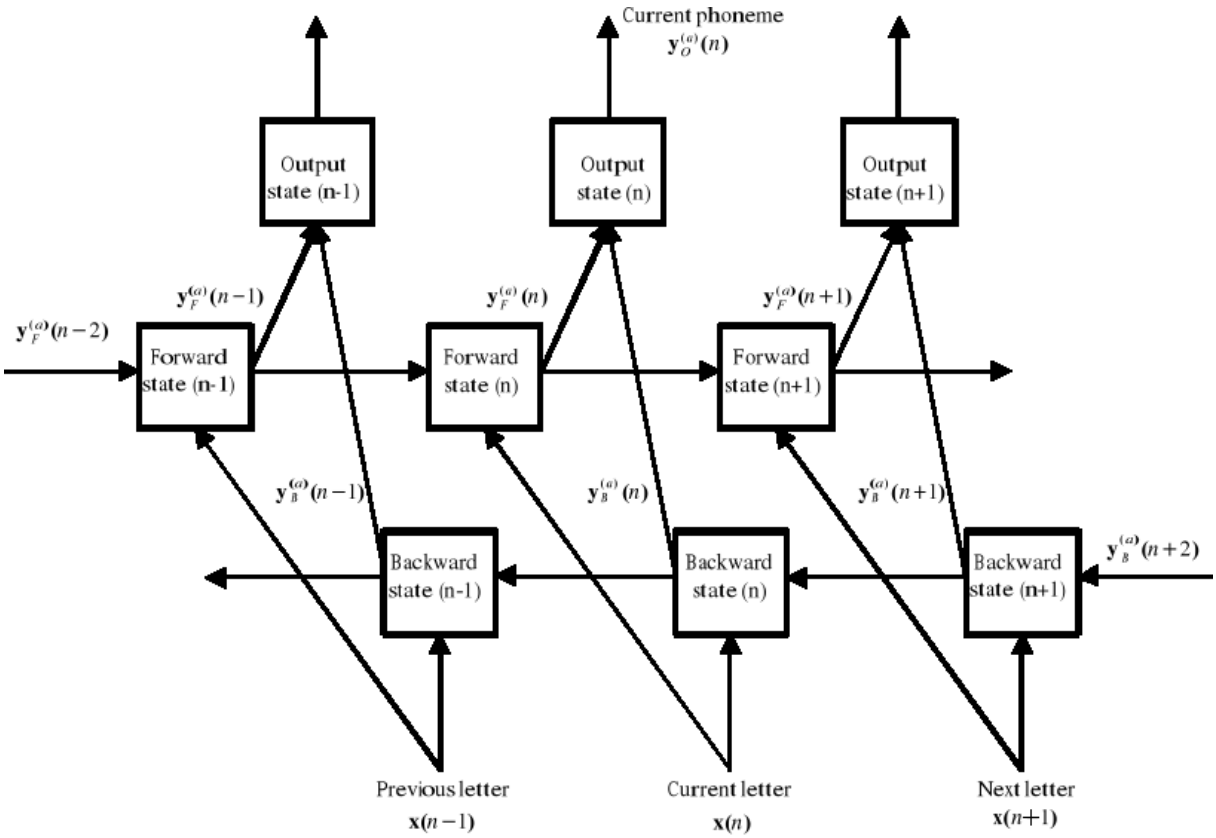

Figure 2.17: The block diagram of a bidirectional recurrent neural network applied for text-to-phoneme mapping. Here the current letter corresponds to $\mathbf{x}(n)$, the previous letter to $\mathbf{x}(n-1)$ and the next letter to $\mathbf{x}(n+1)$.

fact that the output neurons and the state neurons of the BRNN cannot be updated in the same time. Moreover, special attention has to be paid when the synaptic weights from the output states are trained. When a bidirectional recurrent neural network is trained for the text-to-phoneme mapping problem, a finite number of forward and backward states are considered. This is because we apply the bidirectional recurrent neural networks to the problem of finding phonetic transcriptions of isolated words where a letter from a certain word does not depend on letters from adjacent words[22]. In addition, the small number of states avoids the problem of increasing too much the memory load.

The following algorithm can be implemented for training the bidirectional recurrent neural network structure from Fig. 2.17 (see also [59]):

1. **Forward processing (output computation):** the input data $\mathbf{x}(n-T+1), \ldots \mathbf{x}(n+$

[22]The same strategy as in the case of recurrent neural network.

T) from the time instant $n-T+1$ to the future time instant $n+T$ is passed through the neural network.

- The forward step is done for the forward state neurons (see Fig. 2.17) and the induced local field of the forward neurons are computed as follows:

$$\mathbf{y}_F(i) = \mathbf{W}_F(i)\mathbf{x}_F(i), \qquad i = n-T+1, \ldots, n+T \tag{2.57}$$

  where $\mathbf{y}_F(i)$ is the vector containing the induced local fields of the forward state neurons at time instant i, $\mathbf{x}_F(i)$ is the vector containing the inputs of the forward state neurons at time instant i and $\mathbf{W}_F(i)$ is the matrix containing the synaptic weights of the forward neurons.

  The input vector $\mathbf{x}_F(i)$ is obtained by concatenation of the input vector of the neural network $\mathbf{x}(i)$ at time instant i and the output of the previous state neurons $\mathbf{y}_F^{(a)}(i-1)$:

$$\mathbf{x}_F(i) = \begin{bmatrix} 1 \\ \mathbf{x}(i) \\ \mathbf{y}_F^{(a)}(i-1) \end{bmatrix} \tag{2.58}$$

  The forward states are passed in the positive direction from time instant $n-T+1$ to $n+T$.

  The outputs of the forward neurons are obtained applying the activation functions to $\mathbf{y}_F(i)$:

$$\mathbf{y}_F^{(a)}(i) = \mathbf{h}_F\left(\mathbf{y}_F(i)\right), \tag{2.59}$$

  with $\mathbf{h}_F(\cdot)$ being the activation function implemented in the forward neurons.

- The forward step is done for the backward state neurons (see Fig. 2.17) and the induced local fields of the backward state neurons are computed as follows:

$$\mathbf{y}_B(i) = \mathbf{W}_B(i)\mathbf{x}_B(i), \qquad i = n+T, \ldots, n-T+1 \tag{2.60}$$

  where $\mathbf{y}_B(i)$ is the vector containing the induced local fields of the backward state neurons at time instant i, $\mathbf{x}_B(i)$ is the vector containing the inputs of the backward state neurons at time instant i and $\mathbf{W}_B(i)$ is the matrix containing the synaptic weights of the backward neurons.

  Also in this case, the input vector $\mathbf{x}_B(i)$ is obtained by concatenation of the input vector of the neural network $\mathbf{x}(i)$ at time instant i and the output of the next backward state neurons $\mathbf{y}_B^{(a)}(i+1)$. Due to this fact, the backward states

are passed in the negative direction from time instant $n+T$ to $n-T+1$ as we can notice also from (2.60).

The outputs of the backward neurons are obtained applying the activation functions to $\mathbf{y}_B(i)$:

$$\mathbf{y}_B^{(a)}(i) = \mathbf{h}_B\left(\mathbf{y}_B(i)\right), \tag{2.61}$$

with $\mathbf{h}_B(\cdot)$ being the activation function implemented in the backward neurons.

- After the forward processing is done for the forward and for the backward state neurons, this step is repeated for the output neurons as follows:

$$\mathbf{y}_O(i) = \mathbf{W}_O(i)\mathbf{x}_O(i), \qquad i = n-T+1, \ldots, n+T \tag{2.62}$$

with $\mathbf{y}_O(i)$ being the vector containing the induced local fields of the output state neurons at time instant i, $\mathbf{x}_O(i)$ is the vector containing the inputs of the output state neurons at time instant i and $\mathbf{W}_O(i)$ is the matrix containing the synaptic weights of the output neurons.

The input vectors $\mathbf{x}_O(i)$ into the output state neurons are obtained by concatenation of the corresponding forward and backward output vectors $\mathbf{y}_F^{(a)}(i)$ and $\mathbf{y}_B^{(a)}(i)$ respectively.

The outputs of the neural network are obtained applying the activation functions to $\mathbf{y}_O(i)$:

$$\mathbf{y}_O^{(a)}(i) = \mathbf{h}_O\left(\mathbf{y}_O(i)\right), \tag{2.63}$$

with $\mathbf{h}_O(\cdot)$ being the activation function implemented in the output neurons.

2. **Backward processing (weight update):** the error at the output of the neural network is computed and the synaptic weight changes for all three kinds of neurons (forward, backward and output) are calculated.

   - Compute the error vectors at the output of the neural networks for all time instants from $n-T+1$ to $n+T$ as follows:

$$\mathbf{e}_O(i) = \mathbf{d}(i) - \mathbf{y}_O(i), \qquad i = n-T+1, \ldots, n+T, \tag{2.64}$$

   - Perform the backward processing step for the output neurons and compute the synaptic weight changes for the output neurons as follows:

$$\Delta W_{O_{ij}}(n) = \alpha \Delta W_{O_{ij}}(n-1) + \lambda \delta_{O_i}(n) y_{O_j}^{(a)}(n) \tag{2.65}$$

where $\Delta W_{O_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_O(n)$, α is the momentum constant, λ is the learning rate, $y^{(a)}_{O_j}(n)$ is the j^{th} element of the vector $\mathbf{y}^{(a)}_O(n)$ and $\delta_{O_i}(n)$ is computed as:

$$\delta_{O_i}(n) = e_{O_i}(n)h'_O(y_{O_i}(n)), \tag{2.66}$$

with $h'_O(\cdot)$ being the derivative of the activation function $h_O(\cdot)$, $e_{O_i}(n)$ the i^{th} element of the error vector $\mathbf{e}_O(n)$ and $y_{O_i}(n)$ the i^{th} element of $\mathbf{y}_O(n)$.

The synaptic weights $\mathbf{W}_O(n)$ are updated as:

$$\mathbf{W}_O(n+1) = \mathbf{W}_O(n) + \Delta\mathbf{W}_O(n), \tag{2.67}$$

- Perform the backward processing pass for the forward state neurons and compute the changes applied to their synaptic weights as follows:

$$\Delta W_{F_{ij}}(n) = \alpha\Delta W_{F_{ij}}(n-1) + \lambda\delta_{F_i}(n)x_{F_j}(n) \tag{2.68}$$

where $\Delta W_{F_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_F(n)$, α is the momentum constant, λ is the learning rate, $x_{F_j}(n)$ is the j^{th} element of the input vector $\mathbf{x}_F(n)$ and $\delta_{F_i}(n)$ is computed by the following formula:

$$\delta_{F_i}(n) = e_{F_i}(n)h'_F(y_{F_i}(n)), \tag{2.69}$$

where $h'_F(\cdot)$ is the derivative of the activation function $h_F(\cdot)$, $e_{F_i}(n)$ is the i^{th} element of the error vector $\mathbf{e}_F(n)$ in the hidden layer and $y_{F_i}(n)$ is the i^{th} element of the vector $\mathbf{y}_F(n)$.

The errors of the forward state neurons are given by:

$$\begin{aligned} \mathbf{e}(n) &= \mathbf{W}^t_O(n)\delta_O(n), \\ \mathbf{e}_F(n) &= [e_1(n), \ldots, e_N(n)] \end{aligned} \tag{2.70}$$

where the elements of the vector $\delta_O(n)$ are computed in (2.66) and N is the number of forward neurons.

The synaptic weights $\mathbf{W}_F(n)$ are updated as:

$$\mathbf{W}_F(n+1) = \mathbf{W}_F(n) + \Delta\mathbf{W}_F(n), \tag{2.71}$$

- Do the backward processing for the backward state neurons and update their synaptic weights:

$$\Delta W_{B_{ij}}(n) = \alpha \Delta W_{B_{ij}}(n-1) + \lambda \delta_{B_i}(n) x_{B_j}(n) \tag{2.72}$$

where $\Delta W_{B_{ij}}(n)$ is the correction applied to the ij^{th} element of $\mathbf{W}_B(n)$, α is the momentum constant, λ is the learning rate, $x_{B_j}(n)$ is the j^{th} element of the input vector $\mathbf{x}_B(n)$ and $\delta_{B_i}(n)$ is computed by the following formula:

$$\delta_{B_i}(n) = e_{B_i}(n) h'_B(y_{B_i}(n)), \tag{2.73}$$

where $h'_B(\cdot)$ is the derivative of the activation function $h_B(\cdot)$, $e_{B_i}(n)$ is the i^{th} element of the error vector $\mathbf{e}_B(n)$ in the hidden layer and $y_{B_i}(n)$ is the i^{th} element of the vector $\mathbf{y}_B(n)$.

The errors of the backward state neurons are given by:

$$\begin{aligned} \mathbf{e}(n) &= \mathbf{W}_O^t(n) \delta_O(n), \\ \mathbf{e}_B(n) &= [e_{N+1}(n), \ldots, e_{N+M}(n)] \end{aligned} \tag{2.74}$$

where M is the number of the backward state neurons.

The synaptic weights of the backward state neurons are updated using the following formula:

$$\mathbf{W}_B(n+1) = \mathbf{W}_B(n) + \Delta \mathbf{W}_B(n), \tag{2.75}$$

After the above mentioned training algorithm is applied for all input-output pattern pairs the output, the forward and the backward synaptic weights ($\mathbf{W}_O(n)$, $\mathbf{W}_F(n)$ and $\mathbf{W}_B(n)$) are saved. These weights will be used in the testing procedure.

2.5 Conclusions

In this chapter some theoretical aspects of neural networks have been reviewed. The multilayer perceptron, the recurrent and the bidirectional recurrent neural networks together with their most known training algorithms have been described. Also a new structure, introduced to speed up the convergence, namely a new transform domain neural network and a new algorithm for learning rate adaptation have been detailed. The new proposed algorithm and network structure possess the advantage of increased convergence speed and relatively low computational complexity. In the context of the text-to-phoneme mapping, the fast convergence will have a positive impact in reducing the length of the

training dictionary. As a consequence, the design of a text-to-phoneme mapping system, implemented using the proposed approaches, would necessitate lower language resources.

The performances of these neural network structures and training algorithms will be analyzed and the results for the problem of monolingual text-to-phoneme mapping will be presented in Chapter 3. Some of the neural networks structures described in this chapter have been also implemented for the bilingual text-to-phoneme mapping and the results are discussed in Chapter 4.

Chapter 3

Monolingual Text-To-Phoneme Mapping Using Neural Networks

This chapter is dedicated to the problem of monolingual text-to-phoneme mapping and our goal is to describe and analyze the performances of the neural networks described in Chapter 2, for the problem of grapheme-to-phoneme conversion. As we have seen in Chapter 1, the text-to-phoneme mapping is a preliminary step in any text-to-speech application and its goal is to transform a written text into the corresponding phonetic transcription. The synthetic speech can be then generated from this phonetic transcription.

3.1 Classification of text-to-phoneme mapping systems

Based on the specific application, the text-to-phoneme mapping systems can be classified into two main classes. One class deals with the isolated word transcription while the other class deals with continuous grapheme-to-phoneme conversion. The difference, between the two classes of text-to-phoneme mapping, consists in the manner how the letters of a word are treated. In isolated word text-to-phoneme mapping the pronunciation of the letters of a word do not depend on the neighboring words. Due to this fact, when a text-to-phoneme mapping system is trained for isolated word transcription, the dependence between adjacent words must be eliminated. This fact might decrease the accuracy of the phonetic transcriptions in some cases. On the contrary, systems designed for continuous grapheme-to-phoneme conversion do not contain this limitation. In such systems, the phonetic transcription of the letters of a word depends on the previous and on the next word. In some cases, this might be beneficial for the system performance. Fig. 3.1

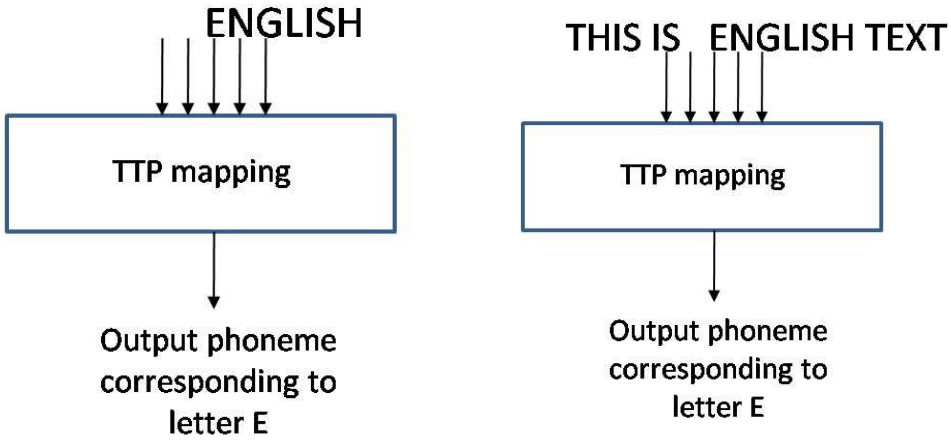

Figure 3.1: Illustration of the main idea of the isolated word text-to-phoneme mapping (left) and of the continuous text text-to-phoneme mapping (right).

illustrates the main idea of the isolated word and of the continuous text text-to-phoneme mapping.

In the example from Fig. 3.1, when the letter "E" of the word "ENGLISH" must be mapped to its corresponding phoneme, several of its adjacent letters are input into the text-to-phoneme mapping module. In the continuous grapheme-to-phoneme conversion the input of the system is "S _ E N G" while in the isolated word text-to-phoneme mapping the input consists of the graphemes "_ _ E N G" (where the symbol "_" represents the space between words). Clearly in the continuous grapheme-to-phoneme conversion the letter "S" of the previous word influences the mapping process of the letter "E" of the current word "ENGLISH". In the isolated word text-to-phoneme mapping only letters of the current word must be taken into consideration as inputs such that the letter "S", from the previous word would be replaced by one blank "_".

Another classification of the text-to-phoneme mapping systems can be made based on the method used to perform the grapheme-to-phoneme conversion. Although there are many different solutions to this problem, they can be classified into two main classes: the rule-based approaches and the data-driven implementations. In the case of rule-based systems a set of context-dependent phonological rules, written by an expert, is used to generate the phonetic transcription of a written text. The definition of the set of rules is a time consuming task and necessitates an expert knowledge of the specific language. Moreover, for many languages, such as English and French, it is extremely difficult to

find a good set of rules that covers all possible letter-phoneme correspondences[1]. The data-driven approaches represent a more powerful alternative to the problem of text-to-phoneme mapping. Such methods are capable of learning the letter-phoneme correspondences from a training dictionary which contains words with known pronunciation. Decision trees (DT), neural networks and Hidden Markkov Models (HMM) are some examples of data-driven methods used to convert a written text into its phonetic transcription. In Table 3.1 we have summarized the main advantages and disadvantages of several rule-based and data-driven methods implemented for grapheme-to-phoneme conversion.

The remaining of this chapter is organized as follows: in the next section we review several different solutions to the text-to-phoneme mapping problem. After that, the pre-processing steps needed to be done before the implementation of a text-to-phoneme mapping system and the database used in our experiments are presented in detail. The performance of the neural network structures, described in Chapter 2, is illustrated for this specific application and the discussion continues with a study of the influence of different letter encoding schemes on the phoneme accuracy. The chapter ends with several conclusions about the feasibility of a text-to-phoneme mapping system based on neural networks.

3.2 Existing approaches

Although our main focus is on the specific problem of text-to-phoneme mapping, we also describe here some implementations of the complete text-to-speech systems. This will help to appreciate the importance and the role of the text-to-phoneme mapping module in a larger context. A speech generation system, able to produce speech in Slovenian from written text, has been proposed in [60]. We have chosen this example here (although we do not consider Slovenian in this thesis) for two main reasons. Firstly, it is a modular approach and each system module can be re-implemented by different means and also adapted to different languages. The second reason is that it uses a combination of rules and decision trees to find the phonetic transcription of the input words which is an alternative approach to the neural networks-based solution. The block diagram of the text-to-speech system proposed in [60] is depicted in Fig. 3.2. The block denoted as "Text Normalization" performs a pre-processing of the input text. For instance, it detects the boundaries of the sentences, detects the abbreviations and special formats (numbers, dates, hours, etc) and transforms them into text. The block denoted as "Grapheme to

[1]Japanese and Finnish are languages that have regular phonological rules which can be implemented in a rule-based grapheme-to-phoneme conversion system.

Rule-based methods	Data-driven methods
Advantages: • Good performance for languages with simple phonemical rules. • Good performance for small lexicons. Disadvantages: • Very difficult to build a good set of rules for many languages with complex phonemical rules. • Necessitate expert knowledge for the specific language addressed.	Advantages (DT): • Accurate modelling. Disadvantages (DT) • Memory extensive (their size depend on the lexicon). • Limited ability to generalize. • Sensitive to the training material. Advantages (NN) • Ability to generalize for unseen words. • Compact representation (the size does not depend on the training lexicon). • Not sensitive to the type of training material (domain specific words can be used for training). Disadvantages (NN) • Inferior performance compared to DT. • Sensitive to over-fitting. Advantages (HMM) • Some approaches does not necessitate an aligned training dictionary. Disadvantages (HMM): • More complicated training compared to NNs.

Table 3.1: Main advantages and disadvantages of several methods used for TTP mapping.

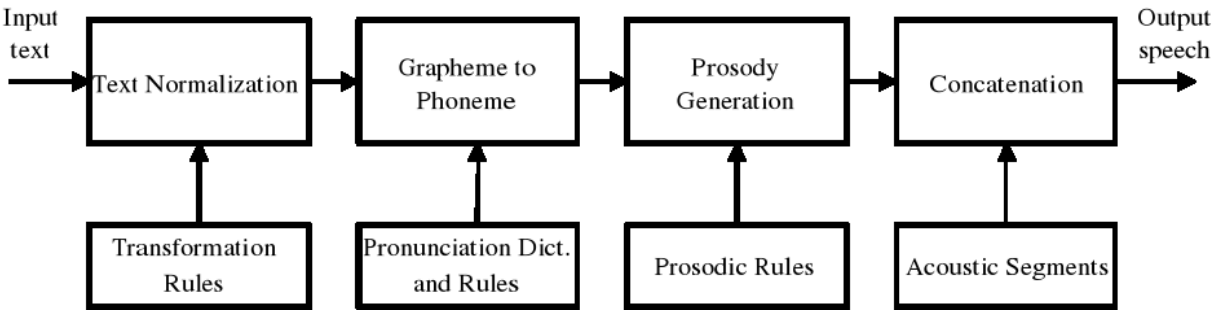

Figure 3.2: The block diagram of the text-to-speech approach from [60].

Phoneme" performs the translation of the written text into the corresponding phoneme string. Prosody is generated in the module "Prosody Generation" and the acoustic speech is produced by the "Concatenation" module[2].

In their approach the authors of [60] included the stress information in the text-to-phoneme module while in our implementations the stress was not taken into account. The results, in terms of word accuracy, for the text-to-phoneme mapping module was presented in [60]. For Slovenian language, the decision trees approach with a context dependence of 4 letters produced 87.80 % accuracy while the rule-based text-to-phoneme approach produced 70.0 % accuracy.

A hybrid grapheme-to-phoneme conversion system, implemented as a combination of rules and decision trees, was also introduced in [50]. The aim of the work presented in this publication was to obtain a text-to-phoneme mapping system with reduced memory load. The core of that system is implemented by means of decision trees (DT) but in addition some rules were added to specify the structure of the initial trees. Another modification, compared to the standard DT approach, is the fact that the leafs of the decision trees generate two informations[3]: an output sequence of phonemes and the length of the corresponding grapheme block. As a consequence, the output of the decision tree, in this approach, generates a phoneme sequence corresponding to multiple input letters instead of a single phoneme for a single letter. The hybrid system from [50] contains also a dictionary of exceptions in which words that cannot be correctly mapped by the decision trees and their transcriptions are stored. The dictionary containing the exceptions is

[2]This is a concatenative approach to speech generation in which the speech segments corresponding to different phonemes are concatenated to obtain the speech signal of the words or sentences.

[3]In text-to-phoneme mapping, usually the leafs of the decision trees generate the phoneme corresponding to the input letter.

used to increase the grapheme-to-phoneme mapping accuracy[4]. The memory load of the system is decreased by reducing the size of the exceptions dictionary. Instead of recording the whole exception words and their transcriptions only the corrections of such words were included in the exception dictionary. The baseline system in which the entire words and their transcriptions are stored in the exception dictionary requires on average 28.8 bits per transcription while the system proposed in [50] necessitates only 3.9 bits per transcription. This compression method of the exceptions dictionary can be used in any grapheme-to-phoneme conversion system not only on those based on decision trees.

In [65] a low memory text-to-phoneme mapping method, based on decision trees, suitable for mobile implementations was proposed. Although their very good performances, in terms of phoneme accuracy, decision trees suffers from high memory footprint [65]. This makes them inappropriate to be implemented for instance in mobile devices which suffers from limited hardware resources [65]. In order to reduce the memory requirements of a grapheme-to-phoneme conversion method, based on decision trees in [65] three methods have been proposed. The first method addresses the issue of building the training dictionary. The second method concentrates on the memory minimization of each DT corresponding to the letters of the vocabulary. The third method addresses the problem of Huffman coding of the DT variables to further reduce the memory requirements. The low memory text-to-phoneme mapping was tested in the automatic speech recognition framework and shows good recognition accuracy with reduced memory load compared to the standard DT approach [64]. From memory point of view, it was shown in [65], that for English language, the memory consumption can be reduced from 262 kB for the standard approach [64] to 167 kB for the low memory approach [65]. In terms of the phoneme accuracy, the low memory DT-based approach achieved 98.99 % phoneme accuracy compared with 98.84 % accuracy of the original approach [64]. However these percentages were obtained when the decision trees-based methods have been tested on the training set. A drop of the phoneme accuracy can be expected when the system is tested on words that have not been present in the training dictionary.

In [3] a comparison between two approaches to the text-to-phoneme mapping problem is presented. A grapheme-to-phoneme conversion system based on binary decision trees and a text-to-phoneme mapping system based on a Trie approach are compared for English and French languages. A Trie is similar with a decision tree but its leafs contains statistical information about a letter in its context. For instance for a given grapheme the leafs contain the number of occurrences of each possible corresponding phoneme, in

[4]In our approaches, presented in this thesis, we do not make use of such exceptions dictionary. However, it can be easily included to improve the text-to-phoneme mapping accuracy.

the specific context, from the entire training dictionary. In terms of phoneme accuracy, both approaches (the DT-based approach and the Trie-based approach) have a phoneme accuracy around 90 % on the Carnegie Melon University database [75].

The rule-based text-to-phoneme mapping systems have been also implemented with success for some languages that have phonological regularity such as: European Portuguese [19], Brazilian Portuguese [63] and Korean [45] to mention a few. In these approaches the grapheme-to-phoneme module is implemented by means of a set of rules that specify the correspondence between the letters and phonemes. For the Brazilian Portuguese a 97.44 % in the phoneme accuracy was reported in [63] while 98.43 % phoneme accuracy was obtained with the rule-based system in [19] for the European Portuguese language. The main difficulty in the implementation of the rule-based text-to-phoneme mapping systems is the definition of a good set of rules that would cover all possible letter-to-phoneme correspondences. Usually the rules are specified by an expert but approaches that use automatically generated sets of rules have been also published. Such a method was proposed in [24] and gave around 95 % phoneme accuracy for Dutch language. However, these results were obtained by training the grapheme-to-phoneme system on the most common 7.000 Dutch words. A larger training dictionary would have a much larger variability which could decrease the phoneme accuracy.

Hidden Markov Models (HMM's) have been used to perform the grapheme-to-phoneme mapping [40, 67]. Usually, the text-to-phoneme mapping problem is addressed assuming that there is a one-to-one correspondence between the letters of a word and the phonemes of its phonetic transcription. In many cases two letters are actually mapped in one phoneme and also one letter can produce two phonemes. Inserting null phonemes, when two letters map to a single phoneme, and using combined phonemes, when a letter generates two phonemes, is the usual way to address this problem. In [40] a HMM-based approach to the problem of grapheme-to-phoneme conversion was introduced and the novelty of this approach consists in the training procedure which assumes many-to-many alignment between the graphemes and the corresponding phoneme string. In such an approach, the number of letters of an input text is not restricted to be equal to the number of phonemes. Firstly, a candidate list of one or more phonemes is generated for each letter of a word and a method based on hidden Markov models is then applied to find the best phoneme sequence corresponding to the input word. The tests done on the Carnegie Mellon University database for English language yield a word accuracy of around 65 % for this approach.

In a large number of publications the text-to-phoneme mapping have been implemented using various neural network architectures [4, 27, 31, 39, 41, 54, 56, 61, 76]. A

neural network approach with self-organizing letter code-book was implemented in [39]. This approach uses a similar architecture as in the NETTalk [61] but the letters are encoded differently. In the NETTalk approach each letter is encoded using some binary codes[5] while in [39] a second smaller neural network is used to adaptively compute the optimal letter codes. The approach using self-adaptive encoding of the input letters has been shown to increase the phoneme accuracy by approximately 1 % compared to the NETTalk implementation. For the American English language, the tests, done on the Carnegie Mellon University dictionary, shows a phoneme accuracy around 90 % when the self-organizing code-book approach was used.

An interesting approach using staged neural networks was proposed in [4]. In this approach three neural networks are used to perform the grapheme-to-phoneme task. The first neural network is used to distinguish between single and dual phoneme cases (in the single phoneme case one letter is mapped to one phoneme while in the dual phoneme case one letter is mapped to two phonemes). The output of this neural network is then used to select one of the two networks from the second stage. The networks from the second stage are trained separately for the single and for the dual case respectively. The experimental results, done on a dictionary containing the most common words in American English, show that by this implementation one could obtain between 82 % to 97 % phoneme accuracy (depending on the size of the training and testing dictionaries). The approach using staged neural networks and the implementation using the staged self-organizing maps were compared in [27] for American English. It was shown that staged neural networks gives better phoneme accuracy than the implementation using self-organizing maps. In both publications the experiments were done on a small dictionary containing the most common words from the American English. It was also shown, that when the size of dictionary increases the performance of both approaches decreases.

A recurrent neural network approach was proposed and tested on a small dictionary of 150 words in [56]. The recurrent neural network has one input layer, one hidden layer and one output layer. The outputs of the neural networks have been used as context inputs through a feedback loop. The difference between this approach and the approach we have tested relies in the fact that the graphemes and their corresponding phonemes does not need to be aligned. Moreover, when the phonetic transcription of one word is generated all its letters are presented at the input of the neural network and kept until the transcription of the entire word is generated. The only inputs which are allowed to change from one phoneme to the other are the context inputs. Such an approach would necessitate a large number of neural network inputs to be able to transcribe long words

[5]This letter encoding scheme will be detailed later in the thesis.

such that, in [56], only words shorter than 6 letters have been used. Nevertheless, a very high phoneme accuracy of 99.55 % was obtained with this implementation, on a very small dictionary of 150 words.

In order to improve the performance of the text-to-phoneme mapping systems based on neural networks, hybrid systems, implemented as combinations of neural networks and decision rules, have been also proposed in [31], [54]. However, in the above mentioned publications, the performances of such systems are not evaluated in terms of phoneme accuracy. Another grapheme-to-phoneme mapping system, based on neural networks, was presented in [76] and tested on the CMU database for the American English language. In this approach the words does not need to be aligned [6]. Instead of aligning by introducing null phonemes the input words are segmented into the so called graphemes. The graphemes, in [76] consists of one or more letters (as opposed to the standard approach in which one letter corresponds to one grapheme). The mapping between these graphemes and the corresponding phonemes is learned by a multilayer perceptron neural network similar to the one used in NETTalk. Using this approach a 78.33 % word accuracy was reported in [76] for the American English language.

In this section, several existing solutions to the problem of grapheme-to-phoneme conversion have been discussed and their performances, in terms of phoneme or word accuracy, have been presented. In the remaining of this chapter we are going to introduce our neural networks-based approach to this problem. However, the reader should understand that a direct comparison of the performances of all solutions presented in this chapter is not straightforward. All these solutions have been implemented in different scenarios (different sizes of the training dictionaries and different selection of the words used for training) or for different languages which make the direct comparison difficult.

3.3 Database pre-processing

Since we address the problem of text-to-phoneme mapping implemented by means of neural networks, the letters and the phonemes must be translated into numerical values. In this section we detail the pre-processing of the database, such as numerical encoding of the letters and phonemes and word alignment, which must be done prior to training and testing the neural networks-based text-to-phoneme mapping system. The dictionary used for training and testing the neural networks in our experiments was the American English Carnegie Mellon University pronunciation dictionary. In order to implement a

[6] Word alignment ensures that the number of letters and their corresponding phonemes must be equals for a given word.

text-to-phoneme mapping system using neural networks, the data from the dictionary was pre-processed in the following manner:

- The words and their phoneme transcriptions were aligned such that one-to-one correspondence was obtained between the letters of each word and their phoneme symbols. Since we are implementing an automatic system for grapheme-to-phoneme conversion, this system will take as input a letter from a current word and will give at the output the corresponding phoneme. Such a system will provide an output for all letters of the current word no matter if that letter is actually pronounced or not. Also in the training phase, it is necessary to have a desired output for each input of the system. These were the reasons why the alignment of the entries into the dictionary is a necessary pre-processing step.

- In order to eliminate the ambiguity that can occur for multiple pronunciations of the same word, only one phonetic transcription was chosen from each entry into the dictionary. This is due to the fact that our text-to-phoneme mapping system was designed to produce only one output for a given input such that, for each word from the dictionary, it is necessary to choose just one pronunciation. This can be in fact a limitation that we have enforced for simplicity reasons. Dealing with multiple pronunciations is left beyond the scope of this thesis.

- Since neural networks are learning systems that necessitate training before utilization, the whole dictionary was split into two parts: the training set and testing set. For the first part we have randomly chosen 80% from the whole CMU dictionary (each word with a single phonetic transcription) and for the second part we have selected the rest of 20% words from the Carnegie Mellon University dictionary. These two sets are used for training (the first part) and respectively for testing (the second part) of the neural network based text-to-phoneme mapping system. The training dictionary used in our simulations contains the phonetic transcriptions of 86821 words and the testing dictionary contains 22015 words. The set used for training the neural networks, and the set used for testing the neural networks did not contain words in common.

- Once the training and testing sets have been obtained, the order of the words in both sets have been randomized.

Since neural networks are systems that works with numerical values, each letter in a word is encoded using numerical vectors, such as the binary orthogonal codes shown in

Table 3.2 (other codes can be implemented as well and some of them are studied later in this chapter). In this table, \0 is introduced to represent the *graphemic null* which is the space between two adjacent words. The number of letters in the English dictionary is 26, and together with a *graphemic null* we have 27 letters. Therefore, each vector representing a letter in a word or space between words, has 27 elements in this encoding scheme.

Letters	Corresponding binary vector
a	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
b	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
c	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
d	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
e	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
f	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
g	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
h	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
i	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
j	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
k	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
l	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
m	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
n	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
o	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
p	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
q	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
r	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
s	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
t	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
u	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
v	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
w	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
x	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
y	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
z	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
\0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

Table 3.2: Orthogonal letter codes. Each vector has 27 elements of which only one is set to unity.

Phonemes	Corresponding binary vector
_	1 0 0 0 . . . 0 0 0
aa	0 1 0 0 . . . 0 0 0
⋮	⋮
zh	0 0 0 0 . . . 0 0 1

Table 3.3: Example of phoneme encoding by orthogonal binary codes. Each vector has 47 elements of which only one is set to unity.

Similar encoding scheme can also be applied for the phonemes. Since sounds from English language can be represented with 47 phonemes including the *null phoneme* (the missing phoneme or the phoneme that is introduced for a letter which is not pronounced) and *pseudo phonemes* (they are obtained by concatenation of several phonemes), the dimension of the orthogonal binary vectors that encode the phonemes is 47, as illustrated in Table 3.3. The above mentioned letter and phoneme encoding schemes are mainly used through this thesis. For the input letters, however, we have done a number of experiments in which several other orthogonal and non-orthogonal codes have been used. The results obtained with such letter encoding schemes are presented at the end of this Chapter.

3.4 The multilayer perceptron neural network for text-to-phoneme mapping

For the multilayer perceptron, in order to take into account the grapheme context, a number of letters on each side of the current letter have been also used as input to the network. It is well known that, for English language, the pronunciation of a certain letter depends on the context it appears (the previous and the following letters). Better transcription accuracy is obtained when, in the mapping of the current letter, several letters from the left and from the right are taken into consideration. However, when the input window of the neural network contains more than 7 letters (current letter, three letters at left and three letters at right) the gain in the transcription accuracy does not justify the increase in the computational complexity and memory load [27]. We have tested the multilayer perceptron neural network having as input three adjacent letters, five adjacent letters and seven adjacent letters with the middle one being the letter to be transcribed. In this section, the phoneme accuracy obtained with multilayer perceptrons that used the letters just from the left context is also presented. These results are included

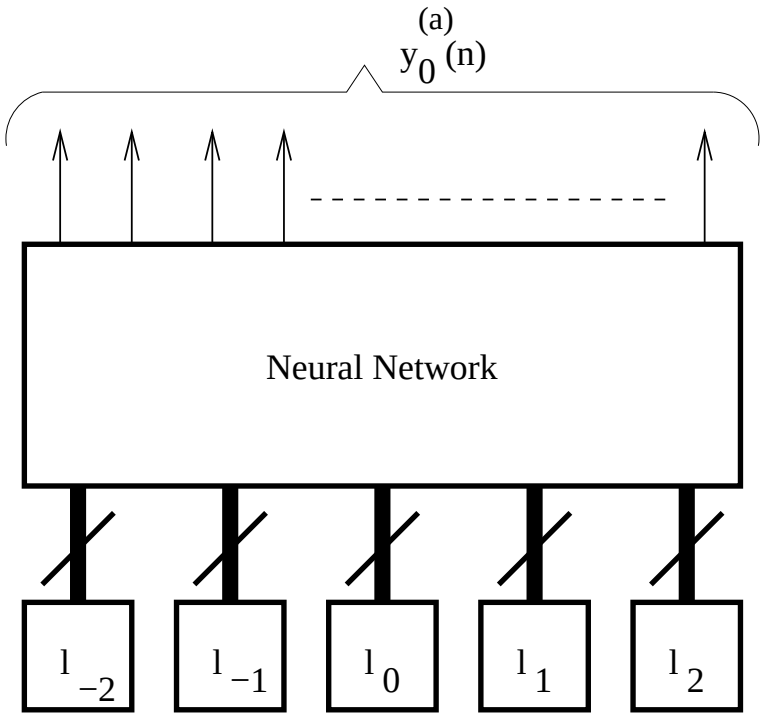

Figure 3.3: The multilayer perceptron with 5 input letters encoded by binary vectors shown in Table 3.2.

here just for benchmark purposes.

In Fig. 3.3 the block diagram of the multilayer perceptron neural network used in the simulations is shown. The input vector $u(n)$ consists of five adjacent letters: two letters at left of the current letter (l_{-2} and l_{-1}), two letters at right of the current letter (l_1 and l_2) and the current letter (l_0). In the experiments the systems where the input vector is obtained by concatenation of three and seven adjacent letters following the same block diagram have also been implemented and compared.

All tested multilayer perceptrons neural networks have one input layer, one hidden layer and one output layer and all of these networks are fully connected. This means that each neuron in the hidden layer receives inputs from each unit in the input layer. Furthermore, each hidden neuron sends its output to all the neurons of the output layer. The number of inputs of the neural networks is different due to the different number of input letters. The number of hidden neurons is denoted by N and the number of outputs is 47 for all implementations. As a consequence, the number of synaptic weights for the compared multilayer perceptron neural networks is computed as follows:

$$S_1 = (2 \cdot 27 + 1)\, N + (N + 1)\, 47 \quad \text{for MLP1}$$

$$S_2 = (3 \cdot 27 + 1)\, N + (N + 1)\, 47 \quad \text{for MLP2}$$

$$S_3 = (3 \cdot 27 + 1)\, N + (N + 1)\, 47 \quad \text{for MLP3}$$

$$S_5 = (5 \cdot 27 + 1)\, N + (N + 1)\, 47 \quad \text{for MLP5}$$

$$S_7 = (7 \cdot 27 + 1)\, N + (N + 1)\, 47 \quad \text{for MLP7} \tag{3.1}$$

where MLP1, MLP2, MLP3, MLP5 and MLP7 represents the multilayer perceptrons with 2, 3, 5 and 7 input letters[7].

Training of the multilayer perceptron neural network is done in on-line mode, such that, the update of the synaptic weights is done for each input of the neural network. The training algorithm used to update the synaptic weights was the error back-propagation with momentum described in the previous chapter.

The phonemes have been encoded using the binary codes from Table 3.3 that have a unity value in the position corresponding to the index of the phoneme (for instance the second phoneme *aa* has the code $[010\ldots0]$). Therefore, in the testing procedure, when one pattern (a group of letters) is presented at the input of the network, at the output we obtain a vector $\mathbf{O}$ with 47 elements that has a maximum value in the position corresponding to the index of the recognized phoneme. The recognized phoneme is then selected using the following criterion:

$$C_c = \text{argmax}\,(\mathbf{O}) \tag{3.2}$$

where C_c is the index of the recognized phoneme from the list of total phonemes (see Table 3.3).

We note that the list of phonemes was alphabetically ordered prior to encoding. In the case of text-to-phoneme mapping, the multilayer perceptron neural network shows very good performance in the sense of phoneme accuracy and simplicity of training [33]. However, the multilayer perceptron can have sometimes a very high complexity (a very large number of synaptic weights), when the input vector has a large dimension.

[7]both MLP2 and MLP3 have 3 input letters. The difference is that MLP2 have 2 letters at the left of the current letter while MLP3 have one letter at the left and one letter at the right of the current letter.

	Input Length	Output Length	Hidden Neurons	Syn. Weights
MLP1	54	47	60	6167
MLP2	81	47	60	7787
MLP3	81	47	60	7787
MLP5	135	47	60	11027
MLP7	190	47	60	14247

Table 3.4: The size of the compared multilayer perceptron neural networks. MLP1 denotes the multilayer perceptron with two input letters (the current letter and the letter at left), MLP2 denotes the multilayer perceptron with three input letters (the current letter and two letters at the left of the current letter), MLP3 denotes the multilayer perceptron with three input letters (current letter and the two adjacent letters from the left and the right), MLP5 denotes the multilayer perceptron with five adjacent letters with the middle one being the current letter and MLP7 denotes the multilayer perceptron with seven adjacent letters with the middle one being the current letter.

The complexities of the compared neural networks are given in Table 3.4, where by MLP1 is denoted the multilayer perceptron having just one left-side context dependence (one letter at the left of the current letter), MLP2 is the multilayer perceptron with two left context dependence letters, MLP3 is the multilayer perceptron with one letter on both sides of the current letter, MLP5 is the multilayer perceptron with two letters on both sides of the current letter and MLP7 is the multilayer perceptron with three letters on both sides of the current letter.

In order to test the recognition performance of each neural network a large number of simulations have been performed and during the experiments different parameters settings have been tested. Among the tested values of each of the parameters we have selected the ones that ensure the best modeling capability of the neural network, for the results shown in this section. The synaptic weights have been initialized with small values chosen from a uniform random distribution with samples in the interval $[-0.02, 0.02]$. All the networks have been trained using a constant learning rate $\lambda = 0.01$. The constant A for the activation functions in the hidden layer and in the output layer (see (2.3)-(2.5)) have been chosen equal to 0.5. The number of hidden neurons in the multilayer perceptron neural networks was $N = 60$ neurons and this leads to different complexities of the neural networks. Later in this chapter, also the comparative results for equal network sizes are shown.

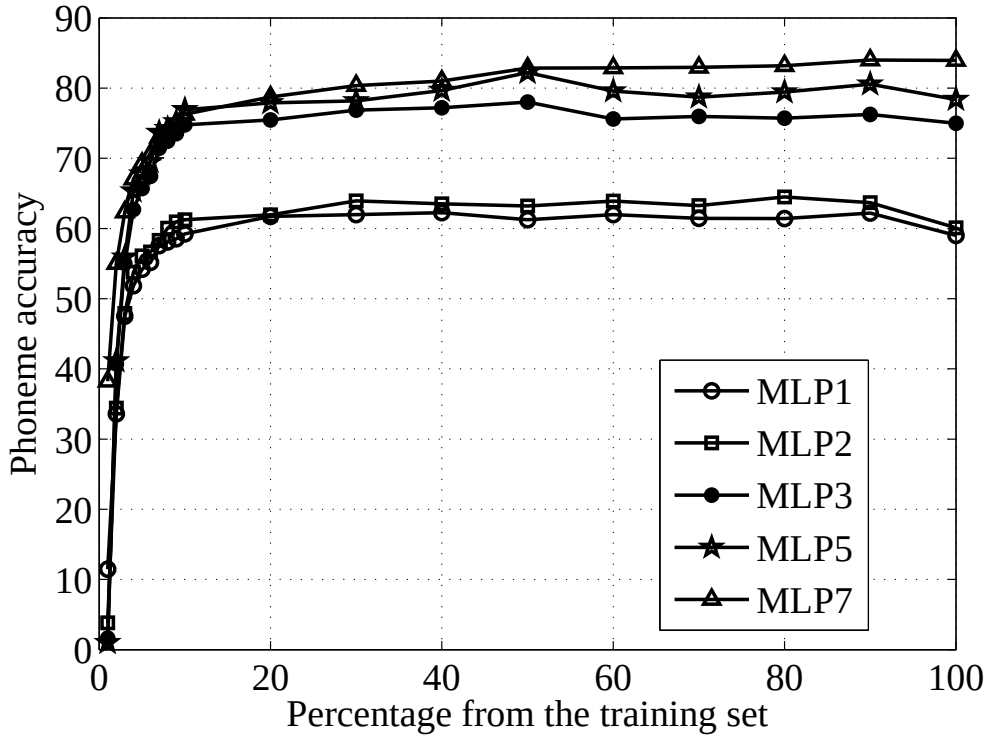

Figure 3.4: Phoneme accuracy for the tested neural networks.

During the training, some intermediate values of the synaptic weights were saved and the testing procedure is performed using these values. The synaptic weights were saved at 1%, 2%,..., 100% from the whole training set. In Fig. 3.4 the phoneme accuracy (that is the percentage of the correct translated letters from the test dictionary) is plotted as a function of this percentage.

From the learning curves shown in Fig. 3.4 one can see that higher phoneme accuracy is obtained with MLP5 and MLP7. This was expected since the MLP5 and MLP7 incorporate large context dependencies from both sides of the current letter. Lower phoneme accuracy was obtained with MLP1 and MLP2. This is because all these networks incorporate context dependence just on the left side of the current letter. From Fig. 3.4 we can see that, as the contextual information increases from MLP3 to MLP7 the phoneme accuracy increases. Of course, the increased phoneme accuracy of the MLP3, MLP5 and MLP7 is also due to their larger number of synaptic weights. Later, in this chapter, we will see that also in the case of equal numbers of synaptic weights this observation is still valid.

3.5 The transform domain multilayer perceptron neural network for text-to-phoneme mapping

The transform domain multilayer perceptron neural network tested in this section was described in detail in Chapter 2 and we show here that it possess a faster convergence than the multilayer perceptron neural network. In the context of grapheme-to-phoneme conversion faster training means smaller size of the training dictionary and reduced training time of the neural network. In this section, the comparative results obtained with the new transform domain multilayer perceptron and with the multilayer perceptron neural network for equal numbers of synaptic weights are shown.

In the case of multilayer perceptron neural network, each letter is encoded using a codebook of length 27 and the corresponding output phoneme is represented by a vector of length 47. The letters and the phonemes are encoded using orthogonal vectors (see Table 3.2 and Table 3.3 respectively).

The proposed transform domain multilayer perceptron neural network was implemented for the problem of text-to-phoneme mapping and the input letters have been encoded differently. Since the number of letters in the English dictionary is 26, the length of the input vectors was chosen to be 5, which is enough to encode 26 letters plus the space between words. The binary vectors used to encode the letters and the *graphemic null* are given in Table 3.5. The binary codes from Table 3.5 are obtained by first ordering the letters from the English dictionary in alphabetical order and after that the index of each letter is translated in binary format. The binary format of the index represents the code of the corresponding letter. For instance, the letter 'c' has the index 3 (it is the third letter in the alphabet) and 11000 is the binary vector used to encode the letter 'c'. Compared to the orthogonal letter codes, used also in NETtalk [49] and presented in Table 3.2, we can see that by using the new approach, the number of inputs of the neural network is highly decreased, which gives us the possibility to use a larger number of hidden neurons for the same neural network complexity.

The NETtalk network is a three layered multilayer perceptron neural network trained using the back-propagation algorithm and the neural network does not contain feedback connections. In NETtalk there are 203 inputs corresponding to 7 consecutive letters, 80 hidden neurons and 26 outputs. The task of the network is to map the input letters to a single phoneme corresponding to the fourth input character. NETtalk uses the same binary codes from Tab. 3.2 to encode the input letters. For the phonemes the codes used in NETtalk were different than the codes used in this thesis.

The block diagram of the transform domain multilayer perceptron neural network is

Letter	Binary Code (length 5)	Letter	Binary Code (length 5)
a	1 0 0 0 0	o	1 1 1 1 0
b	0 1 0 0 0	p	0 0 0 0 1
c	1 1 0 0 0	q	1 0 0 0 1
d	0 0 1 0 0	r	0 1 0 0 1
e	1 0 1 0 0	s	1 1 0 0 1
f	0 1 1 0 0	t	0 0 1 0 1
g	1 1 1 0 0	u	1 0 1 0 1
h	0 0 0 1 0	v	0 1 1 0 1
i	1 0 0 1 0	w	1 1 1 0 1
j	0 1 0 1 0	x	0 0 0 1 1
k	1 1 0 1 0	y	1 0 0 1 1
l	0 0 1 1 0	z	0 1 0 1 1
m	1 0 1 1 0	\0	1 1 0 1 1
n	0 1 1 1 0		

Table 3.5: Non-orthogonal codes used to encode the input letters in the transform domain multilayer perceptron neural network.

shown in Fig. 2.9 for the case of 5 input letters. Since the vector of each letter is orthogonalized by the Discrete Cosine Transform, there is no need to use orthogonal vectors for letter encoding, as in NETtalk and other approaches. Both neural networks compared here have been trained with their corresponding algorithms described in Chapter 2. Because the training dictionary contained a large amount of data, both neural networks have been trained in on-line mode, where the update of the synaptic weights is done after the presentation of each input letter. Since the multilayer perceptron and the transform domain multilayer perceptron neural networks have different number of inputs, 136 and 26 inputs respectively[8], the number of hidden neurons were also different such that the number of synaptic weights is approximately the same. The parameters (learning rates, momentum constant, etc) used in the training procedure have been chosen to obtain the best phoneme accuracy for both neural networks. The hyperbolic tangent activation function was implemented in the hidden layer and the softmax activation function was implemented in the output layer. We emphasize here, that the output phonemes have been encoded in the same manner in both implementations using the binary codes shown in Table 3.3.

[8]We compare here the performances of the multilayer perceptron and of the transform domain multilayer perceptron with 5 input letters.

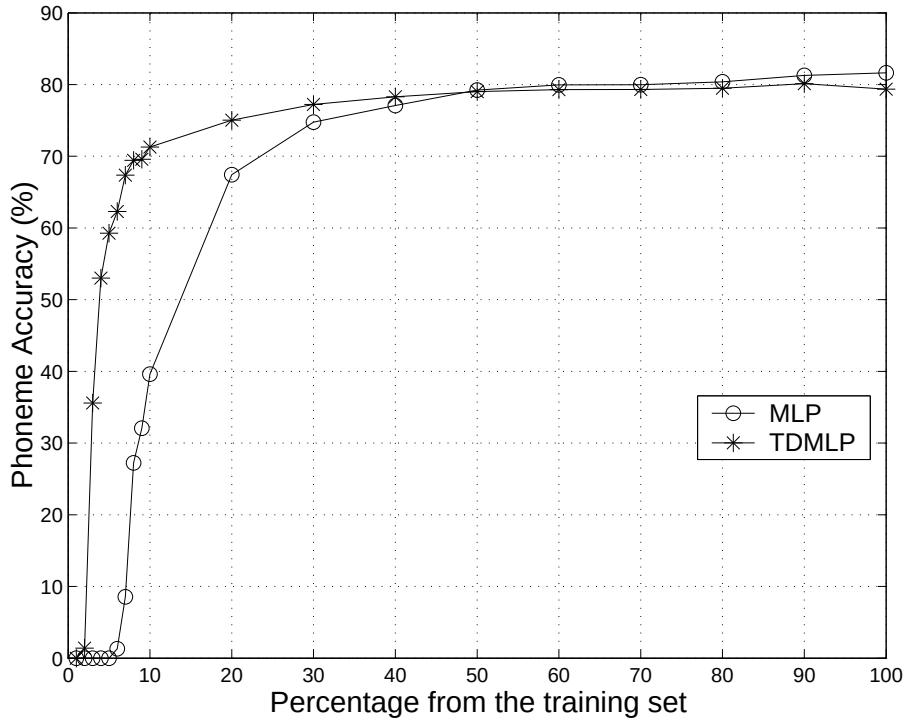

Figure 3.5: Phoneme accuracy obtained with the multilayer perceptron neural network and with the transform domain multilayer perceptron neural network.

To compare the convergence speed of both neural networks, the synaptic weights have been saved during training at 1%, 2%, ... and 100% from the training dictionary. The testing was performed on the whole testing dictionary using the saved synaptic weights. The results showing the dependence between the number of training iterations (training letters) and phoneme accuracy is shown in Fig. 3.5. From this figure it can be seen that the transform domain multilayer perceptron neural network has much faster convergence which means that the necessary size of the training dictionary is much smaller. Actually, for the transform domain multilayer perceptron we can use a training set that is $\approx 30\%$ from the training dictionary and we obtain good results.

3.6 The multilayer perceptron with adaptive learning rate for text-to-phoneme mapping

In this section, the experimental results obtained with the multilayer perceptron neural network that uses an adaptive learning rate in the training process are presented. The

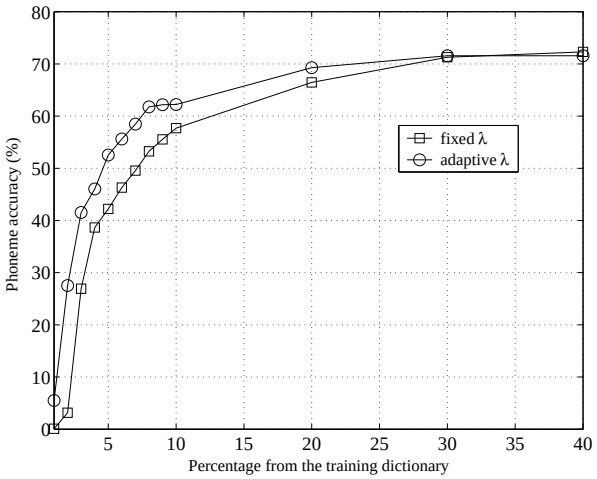

Figure 3.6: Phoneme accuracy, during training, obtained with the MLPALR and the multilayer perceptron neural networks, with 3 input letters, for 10^3 synaptic weights.

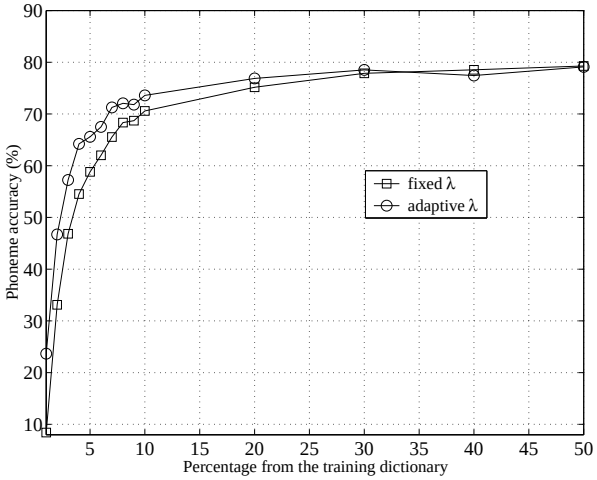

Figure 3.7: Phoneme accuracy, during training, obtained with the MLPALR and the multilayer perceptron neural networks, with 3 input letters, for 3×10^3 synaptic weights.

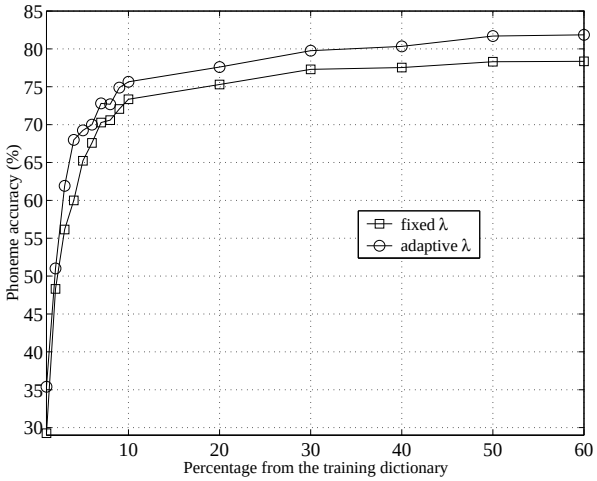

Figure 3.8: Phoneme accuracy, during training, obtained with the MLPALR and the multilayer perceptron neural networks, with 3 input letters, for 5×10^3 synaptic weights.

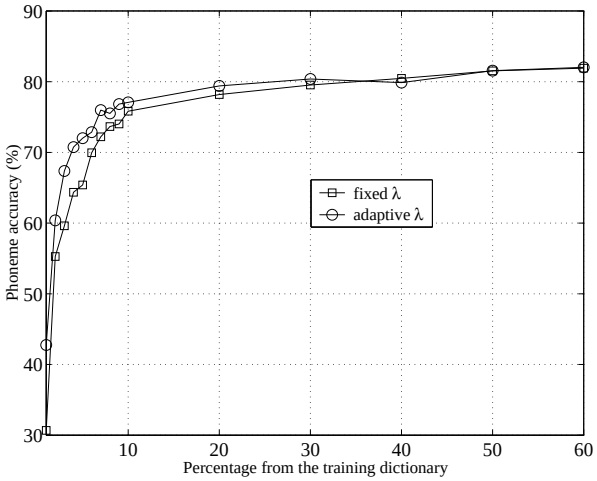

Figure 3.9: Phoneme accuracy, during training, obtained with the MLPALR and the multilayer perceptron neural networks, with 3 input letters, for 10^4 synaptic weights.

aim of using adaptive learning rate is to increase the convergence speed of the neural network and to reduce the number of required training iterations [6, 22, 38, 77].

For benchmark purposes we illustrate also the phoneme accuracy obtained with a

multilayer perceptron neural network trained with a constant learning rate. The compared neural networks were trained in on-line mode. Moreover, in the training process, of the MLPALR neural network, adaptation of the learning rate was done in on-line mode also. This means that the learning rate was changed at each training iteration. The training algorithm of the multilayer perceptron with adaptive learning rate was presented in detail in Section 2.2.3. Both compared networks used sigmoid tangential activation functions in the hidden layer and the softmax activation functions for the output neurons. The results, presented here, are obtained with the multilayer perceptron and with the multilayer perceptron with adaptive learning rate having 3 input letters (one letter on both sides of the current letter) and equal number of synaptic weights.

Experimental results have been done for different numbers of synaptic weights: 10^3, 3×10^3, 5×10^3 and 10^4 and the phoneme accuracies are shown in Fig. 3.6, Fig. 3.7, Fig. 3.8 and Fig. 3.9 respectively. During training the synaptic weights have been saved at 1%, 2%, ... and 100% from the training dictionary such that the phoneme accuracy during training was plotted.

From the learning curves shown in Fig. 3.6 to Fig. 3.9 one can see that the convergence speed of the multilayer perceptron trained with adaptive learning rate is higher than the convergence speed of the multilayer perceptron neural networks trained with a constant learning rate. This is more evident especially at low neural network complexities (small number of synaptic weights). For instance from Fig. 3.6 we can see that the MLPALR neural network necessitates only 20% from the training dictionary (approximately 17364 words) to obtain a phoneme accuracy of around 70%, while in the case of a fixed learning rate the network needed 30% (approximately 26046 words) to obtain the same phoneme accuracy. An interesting result can be observed in Fig. 3.8 where the multilayer perceptron with adaptive learning rate consistently gave better phoneme accuracy compared with the multilayer perceptron trained with a fixed learning rate.

3.7 The recurrent and bidirectional recurrent neural networks for text-to-phoneme mapping

When the multilayer perceptron neural network is applied to the grapheme-to-phoneme conversion, the context information is included into the system by taking several adjacent letters at the input of the neural network. In many applications, the requirement is to obtain fast networks that use a small amount of memory and also can provide accurate recognition rates. Usually, each letter and each phoneme are represented as binary vec-

tors for the neural network. If these vectors have large dimensions, also the input-output layers of the neural network scales correspondingly, which easily results in large number of weights especially if context vectors are used. Although the context dependence has been shown to increase the phoneme recognition accuracy, in some cases the complexity (number of synaptic weights) of the multilayer perceptron is too large for practical implementations. Based on these observations we study the recognition results obtained by neural networks with different structures, such as recurrent neural network and bidirectional recurrent neural network and we compare the results with the context dependent multilayer perceptron approaches.

For this purpose, we have tested different parameter settings and different network complexities for the multilayer perceptron, the recurrent neural network and the bidirectional recurrent neural network. The phoneme accuracy obtained on both training set and testing set with each of the tested network architectures, having different number of synaptic weights, is given in Table 3.6.

The compared networks have been the multilayer perceptron with 3, 5 and 7 adjacent letters at the input, the recurrent neural network with a truncation depth of 3 and the bidirectional recurrent neural network with 3 and 5 states. We didn't include the results of recurrent neural network with larger truncation depth since the results were close to RNN3. In all neural networks the activation function of the output layer was the softmax activation function. In the hidden layer of multilayer perceptrons and in the forward and backward states in bidirectional recurrent neural networks we have used the hyperbolic tangent activation function.

The learning rates used for training all the considered neural networks have been chosen, such that, the network has the best phoneme accuracy.

From the results presented in Table 3.6 we can conclude the following:

- As the contextual information increases from MLP3 to MLP7 the phoneme accuracy also increases. However, the MLP7 does not give a significant increase in the phoneme accuracy compared to the results obtained with the MLP5 neural network[9].

- The recurrent neural network with left context dependence, due to the feedback loops, gives much smaller phoneme accuracy than the multilayer perceptron neural network. If the recurrent neural network is transformed into a bidirectional recurrent neural network the phoneme accuracy increases.

[9]This result was observed also in Section 3.4 but there the compared multilayer perceptrons had equal numbers of hidden neurons and not equal complexities.

NNs	Tested dict	Number of weights 7500	10000	15000	20000
MLP3	Train	79.11	78.99	78.78	78.60
	Test	79.26	79.12	78.88	78.71
MLP5	Train	83.46	83.27	83.28	83.00
	Test	83.37	83.17	83.55	83.32
MLP7	Train	83.37	83.04	84.00	83.68
	Test	84.25	83.86	84.38	84.04
RNN3	Train	61.10	60.90	60.20	60.17
	Test	60.44	60.45	60.83	60.85
BRNN3	Train	76.04	75.88	76.35	76.24
	Test	76.45	76.35	75.64	75.50
BRNN5	Train	72.99	72.81	74.45	74.27
	Test	74.61	74.36	72.66	72.48

Table 3.6: Phoneme accuracies in percents obtained with the multilayer perceptron neural network, the recurrent neural network and the bidirectional recurrent neural network for different network complexities (number of weights).

- When the performance obtained with BRNN3 and BRNN5 are compared, we can see that BRNN5 performs worse than BRNN3. Also, both BRNNs perform worse than the MLPs. For our task of finding the phonetic transcriptions of isolated words, we can conclude that among the compared architectures, the multilayer perceptron neural network is the best choice in terms of network complexity, simplicity of training and phoneme accuracy. It should be noted that the input context information due to adjacent letters is not considered in the recurrent neural network presented here. To be more specific, only a weak context dependence exists due to the feed-back and feed-forward loops in the recurrent neural network and bidirectional recurrent neural network. These feed-back and feed-forward loops are inserting the past and respectively the future outputs back to the input of the neural network. As a consequence, modeling of the current inputs depend on the modeling of the past and future inputs which, for text-to-phoneme mapping, means context dependence between current letter and left and right adjacent letters. However, the dependence established by the feed-forward and feed-back loops is weaker compared to the one obtained in the multilayer perceptron neural network with several input letters.

3.8 The effect of orthogonal and non-orthogonal letter codes to the phoneme accuracy

In this section the performance, in terms of phoneme accuracy, of a multilayer perceptron-based text-to-phoneme mapping system when the input letters are encoded in several different ways is studied. More specifically, the following vector codes have been used for the input letters:

- **Orthogonal binary codes (OBC)** as shown in Table 3.2. The length of a vector corresponding to a single letter has 27 elements (there are 26 letters in the English alphabet and the space between the words). Usually, orthogonal vectors are used in order to increase the phoneme accuracy and to speed up the training of the neural network [39]. A simple straightforward approach is to use the codes from Table 3.2.

- **Non-orthogonal binary codes (NOBC)** as shown in Table 3.5. The vector to encode a single letter has length 5 (5 bits are enough to encode 27 characters). These codes, although non-orthogonal, have the advantage of having a much shorter length than the previous ones. However, the inputs of the neural network are correlated in this case, and we should expect the convergence speed of the multilayer perceptron neural network to decrease.

- **Non-orthogonal codes of -1 and $+1$ (NOC)** as shown in Table 3.7. These codes are obtained from the ones in Table 3.5 replacing the zero bits with -1. The non-orthogonal binary codes from Table 3.5 have non-negative values. Changing the zero bits with -1 we increase the dynamic range of the inputs.

- **Random real valued codes (RC)** shown in Tab. 3.8. In this experiment, the codes of the input letters are obtained from a random Gaussian-distributed sequence of real numbers. The length of the codes was 5 and the random sequence has zero mean and unity variance. The aim of using these codes is to study the phoneme accuracy when the inputs of the neural network are un-correlated and non-binary. Moreover, the elements of each code have positive and negative values in order to expand the dynamic range of the neural network inputs. It should be emphasized here that the random codes are generated just once at the beginning of the training procedure and the same codes are used for testing the neural network.

- **DCT codes (DCT)** In this case, the letters are encoded as shown in Table 3.5 and transformed using the Discrete Cosine Transform prior to application to the

Letter	Binary Code (length 5)	Letter	Binary Code (length 5)
a	1 -1 -1 -1 -1	o	1 1 1 1 -1
b	-1 1 -1 -1 -1	p	-1 -1 -1 -1 1
c	1 1 -1 -1 -1	q	1 -1 -1 -1 1
d	-1 -1 1 -1 -1	r	-1 1 -1 -1 1
e	1 -1 1 -1 -1	s	1 1 -1 -1 1
f	-1 1 1 -1 -1	t	-1 -1 1 -1 1
g	1 1 1 -1 -1	u	1 -1 1 -1 1
h	-1 -1 -1 1 -1	v	-1 1 1 -1 1
i	1 -1 -1 1 -1	w	1 1 1 -1 1
j	-1 1 -1 1 -1	x	-1 -1 -1 1 1
k	1 1 -1 1 -1	y	1 -1 -1 1 1
l	-1 -1 1 1 -1	z	-1 1 -1 1 1
m	1 -1 1 1 -1	\0	1 1 -1 1 1
n	-1 1 1 1 -1		

Table 3.7: Non-orthogonal letter codes. Each vector has 5 elements of -1 and $+1$.

Letters	Corresponding binary vectors (length 5)				
a	-0.4326	1.1909	-0.1867	0.1139	0.2944
b	-1.6656	1.1892	0.7258	1.0668	-1.3362
c	0.1253	-0.0376	-0.5883	0.0593	0.7143
...			...		
\0	0.2877	0.3273	2.1832	-0.0956	1.6236

Table 3.8: Random real valued letter codes of length 5. The elements of each vector have been randomly chosen from a zero mean Gaussian-distributed random sequence with unity variance.

neural network. Moreover, the training algorithm of the neural network changes due to this fact. A block diagram of the transform domain multilayer perceptron is depicted in Fig. 2.9 and a detailed description of the training algorithm was given in this chapter and can be found also in [13]. By using this encoding of the input letters we wanted to compare the phoneme accuracy obtained with the DCT codes and the performance of the neural network that uses the above mentioned letter codes.

To implement our text-to-phoneme mapping system we have chosen to use the multilayer perceptron neural network due to its simplicity of implementation. In our experiments, we have used three layered neural networks with one input layer, one hidden layer and one output layer. To increase the phoneme accuracy, 5 letters have been considered at the input of the network with the middle one being the letter to be transcribed[10]. A block diagram of the neural network that was used in these experiments is depicted in Fig. 3.3.

The multilayer perceptron neural network was trained using the back-propagation with momentum algorithm. The hidden neurons have hyperbolic tangent activation functions and the output neurons have softmax activation function.

The number of synaptic connections between the neurons of the neural network can be computed as follows:

$$S_5 = (5L + 1)\, N + (N + 1)\, P \tag{3.3}$$

where L is the length of the vector used to encode a single input letter, N is the number of neurons in the hidden layer, and P is the number of output neurons ($P = 47$ in this case).

The same formula (3.3) can be applied for both the multilayer perceptron and the transform domain multilayer perceptron neural networks to compute the number of synaptic neurons. The difference is the length of letter codes ($L = 27$ for the binary orthogonal codes from Table 3.8 and $L = 5$ for the other codes). The term $5L + 1$ appears due to the five input letters plus the input bias while the term $N + 1$ appears due to N hidden neurons and the bias term in the hidden layer.

From (3.3) we can see that if one uses larger vectors to encode the input letters, the number of synaptic connections S_5 will be larger if the number of hidden neurons N is kept constant. This increases the memory and computational load of a system that implements grapheme-to-phoneme conversion. Low memory and computational load are extremely important, for instance, in mobile implementations. Although available memory and computational power of the mobile devices continuously increases there will be more and more applications to run in parallel. As a consequence, applications with low complexity will still be of large interest. One alternative to decrease the complexity of the above mentioned text-to-phoneme mapping system is to reduce the number N of hidden neurons. However, this cannot be reduced too much without decreasing the performance of the system. Another way to reduce the memory load and computational complexity is to decrease the number of inputs of the neural network. This can be done

[10]This setup is similar with the one in the pervious experiments.

either by reducing the number of input letters or by reducing the length of the vectors used to encode the letters. In text-to-phoneme mapping the phoneme accuracy is highly influenced by the number of input letters (see [17, 33, 39]) therefore, the former solution is not of interest. The best way to reduce the complexity is to shorten the letter codes and this is the reason why we studied the above mentioned codes[11].

All compared codes have advantages and disadvantages: the OBC, used also in many other implementations [14, 49], increases the memory load of the neural network. In order to decrease the memory load of a text-to-phoneme mapping system, different types of codes with shorter length can be implemented. When the DCT codes are used to perform the grapheme-to-phoneme conversion the neural network training is different due to the transform layer. Normalization of the synaptic weight corrections[12] must be introduced into the training algorithm, which makes the training more complicated (see Chapter 2 for more details). However, once trained, the neural network using DCT input codes is used exactly as the standard multilayer perceptron neural network and has the same computational and memory load.

It must be emphasized that, in the implementations described here the problem of isolated word transcription is addressed. As a consequence, the dependence between adjacent words is not taken into account. Therefore, when the first letter of a word is transcribed, the input vector of the neural network is formed by concatenation of five letter codes: two codes for \0, the code of the current letter and the codes corresponding to the second and the third letters from the current word. A similar approach is done when the last letter of a word must be transcribed. In this case, the first elements of the input vector correspond to the last three letters of the word (the code of the current letter being in the middle of the input vector) and the last elements corresponds to two spaces.

The phoneme accuracy obtained with the five input letter codes are shown in Table 3.9 for different neural network complexities. The complexities of the neural networks listed in Table 3.9 represents the number of the synaptic weights. Analyzing the results from this table we see that for very small number of synaptic weights (between 400 and 500), the shorter codes (NOBC, RC, NOC and DCT) provide much higher accuracy (the difference is around 20%) compared to the OBC case. For moderate number of synaptic weights

[11] Actually one could decrease the number of neural network outputs, but this only decreases the number of synaptic weights in the output layer. Another alternative to reduce the computational complexity is to use non-fully connected neural networks. These two alternatives are beyond the scope of this thesis.

[12] Without this normalization the convergence speed does not improve. See for instance [33] for more details.

Complexity (number of synaptic weights)	NOBC	RC	NOC	DCT	Complexity	OBC
485	62.41%	63.83%	62.78%	62.85%	413	40.75%
923	71.32%	70.87%	71.50%	72.03%	962	72.24%
1361	74.63%	74.70%	74.93%	73.94%	1328	75.11%
1799	75.95%	75.42%	76.28%	75.66%	1877	77.50%
2383	76.90%	76.84%	78.44%	77.02%	2426	79.97%
3551	77.62%	77.69%	79.07%	78.73%	3524	81.37%
4427	77.75%	77.14%	78.83%	78.55%	4439	82.68%
5303	77.65%	78.05%	78.79%	78.70%	5354	82.41%
6325	77.83%	77.64%	79.06%	77.93%	6269	82.74%
7347	77.91%	77.65%	78.95%	77.68%	7367	82.81%
8807	75.95%	76.50%	77.56%	76.96%	8831	83.02%

Table 3.9: Phoneme accuracy obtained with five different input codes: non-orthogonal binary codes (NOBC), random real valued codes (RC), non-orthogonal codes of $\{-1, +1\}$ (NOC), DCT domain codes (DCT) and orthogonal binary codes (OBC).

(around 1000 and 1300) all codes provide slightly the same performances. When the number of synaptic weights is increased, the use of the orthogonal binary codes (OBC) increases the phoneme accuracy of the neural network (the difference is between 4 to 5 percent). We have to emphasize here that increasing too much the number of the synaptic weights does not bring a significant increase in the phoneme accuracy. We can see also from the results shown in Table 3.9, for the OBC implementation, increasing the complexity above 4439 synaptic weights the phoneme accuracy is limited around 82 %. This is linked to the well known issue of "overfitting" that is characteristic to many neural network-based systems [33].

From these results we can conclude that in applications which require a very low memory load the orthogonal binary codes does not provide good enough performance. In these cases, shorter codes, as the ones presented here, provide higher phoneme accuracy. However, in applications where the complexity of the neural network is not a limiting factor, the OBC codes applied to larger neural networks provide an additional 4 to 5 percents increase in the phoneme accuracy. The above mentioned encoding schemes can be also implemented in more sophisticated text-to-phoneme mapping systems such as the multilingual case which is treated in the next chapter of this thesis.

Letters	Corresponding random real valued vectors of length 27							
a	−0.43	− 1.59	0.59	0.79	...	− 0.94	1.47	0.03
b	−1.66	− 1.44	− 0.64	0.94	...	− 0.37	1.13	− 0.62
⋮				⋮				
z	−1.14	0.69	− 0.01	0.23	...	1.47	− 0.07	− 0.20
\0	0.14	0.28	2.09	− 0.13	...	0.70	− 0.83	− 1.08

Table 3.10: Random real valued letter codes of length 27. The elements of each vector have been randomly chosen from a zero mean Gaussian-distributed random sequence with unity variance.

3.8.1 Neural networks with random letter codes for text-to-phoneme mapping and small training dictionary

We continue the study of the convergence speed of the multilayer perceptron neural network that uses random codes for the input letters and we show that, when the input letters are encoded using vectors with equals lengths, real random codes can provide faster convergence compared to the orthogonal binary codes that are usually implemented. We compare the results obtained with the multilayer perceptron using binary and random valued codes with transform domain multilayer perceptron and the multilayer perceptron with adaptive learning rate. The main goal of this study is to verify the performance of the approach based on neural networks when only a small training dictionary is used.

During the training of a grapheme-to-phoneme conversion system based on neural networks, at each iteration, a group of letters are presented at the input of the neural network. The output of the neural network is the phoneme that corresponds to the middle input letter. Due to this fact the number of iterations during the training process is equal to the total number of letters in the training dictionary. In order to ensure an increased level of phoneme accuracy, usually a large training dictionary is used. In a large dictionary, the number of repetitions of a certain group of input letters is large enough to be properly learned by the neural network. If the available training dictionary is large enough, then the multilayer perceptron neural network can be trained with good performance using a constant learning rate [33]. However, it is not a trivial task to build a large training dictionary and it can be very time consuming. As a consequence, it would be of practical interest to have some training methodology that ensures convergence in fewer number of iterations such that a smaller training dictionary can be used[13]. The total size of the

[13]This can be useful for languages for which large training dictionaries are not available.

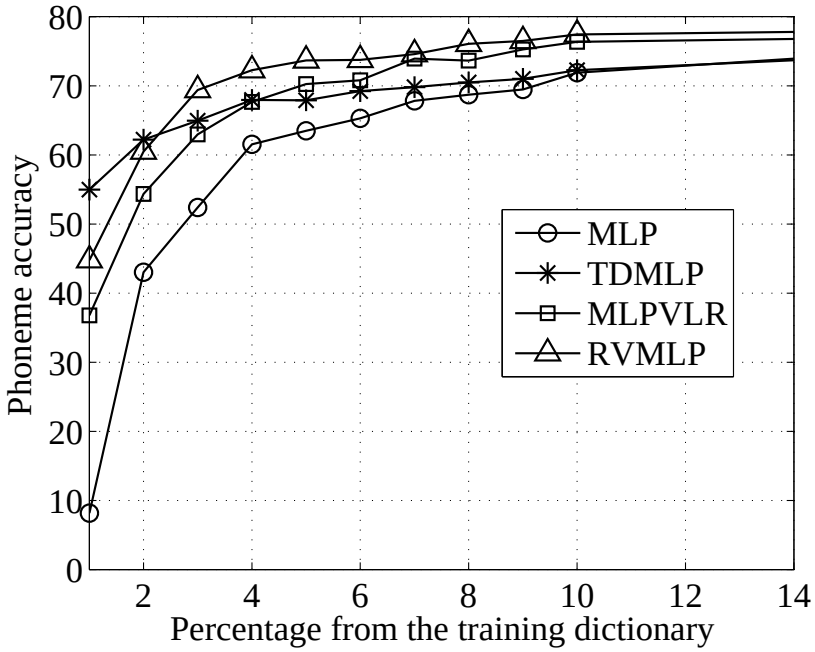

Figure 3.10: Phoneme accuracy obtained with the different neural networks as function of the size of the training dictionary (MLP denotes the multilayer perceptron neural network trained with a fixed learning rate, TDMLP is the transform domain multilayer perceptron neural network, MLPVLR is the multilayer perceptron trained with an adaptive learning rate and RVMLP is the multilayer perceptron trained with a constant learning rate but using random encoding of the input letters).

training dictionary that we have used here is very large (around 8×10^4 words) but we will see later in this section, that only a small fraction from this dictionary is needed in some circumstances.

In the experiments shown here, we have used three types of letter codes: the binary vectors shown in Tab. 3.7, the random real codes depicted in Tab. 3.10 and the binary non-orthogonal codes of length 5 from Tab. 3.8. The random real codes have length 27 and their elements were chosen from a sequence of real numbers with zero mean random Gaussian distribution and unity variance. A binary encoding scheme was applied for the phonemes (see Tab. 3.3).

We have trained the neural networks in online mode where the synaptic weights are updated at each training iteration [33]. After the neural networks have been trained, the performance in terms of phoneme accuracy is evaluated on the test dictionary.

We study the convergence speed of two neural network structures such as multilayer perceptron neural network and the transform domain multilayer perceptron neural network. Also three types of encoding vectors for the input letters are analyzed and two training algorithms: the error back-propagation with momentum and fixed learning rate and the error back-propagation with momentum and time-varying learning rate.

In our experiments the multilayer perceptron neural networks have one input layer, one hidden layer of neurons and one output layer. The number of outputs is 47 which is equal to the length of the phoneme codes and the multilayer perceptron takes five adjacent letters at the input. The transform domain multilayer perceptron neural network has also one input layer, one hidden layer of neurons and one output layer. A number of 47 outputs and a number of 26 inputs (25 inputs due to the 5 input letters and one input bias term) are used in the transform domain multilayer perceptron neural network.

The activation function used in the hidden layer, for both multilayer perceptron and transform domain multilayer perceptron neural networks was the hyperbolic tangent activation function. At the output both multilayer perceptron and transform domain multilayer perceptron have softmax activation function.

The training algorithm for the TDMLP neural networks is derived from the standard error back-propagation with momentum in which the orthogonal transformation of the neural network inputs is included. The multilayer perceptron neural network was trained with both fixed learning rate and time-varying learning rate and the algorithms have been detailed in Chapter 2. All compared neural networks were fully connected and had equals numbers of synaptic weights.

We emphasize that our main goal here is to study the influence of the training algorithm and input letter encoding on the convergence speed of the neural networks. In order to see the phoneme accuracy obtained when using a short training dictionary, the synaptic weights of all the compared systems were saved at 1%, 2%, ..., 100% from the training dictionary. The tests have been done for all these synaptic weights and the results are shown in Fig. 3.10. For instance 5% of the training dictionary represents roughly 4000 words. As we can see from Fig. 3.10, for very small training dictionaries (up to approximately 2000 words) the transform domain multilayer perceptron offers better phoneme accuracy compared with the other neural networks. For higher sizes of the training dictionary, the MLP that uses random real valued letter codes provides the best mapping accuracy. For very large size of the training dictionary, the performances of the four neural networks tends to stabilize around the same level of the phoneme accuracy.

3.9 Conclusions

In this chapter of the thesis, the practical problem of monolingual text-to-phoneme mapping for isolated words was studied for the American English. We started with an overview of the monolingual text-to-phoneme mapping problem and we emphasized the differences between the two main problems: the isolated word grapheme-to-phoneme conversion (which was the focus of our work) and continuous text-to-phoneme mapping. We continued with the description of a number of existing approaches to the problem of text-to-phoneme mapping, published in the open literature, which helps the reader to get a more deep insight into this field. The database pre-processing, which is a pre-requisite in the implementation of a grapheme-to-phoneme conversion system, was detailed next.

In the following section the results of several experiments done with a multilayer perceptron neural network with different number of input letters are presented and analyzed. As we have expected the accuracy of the grapheme-to-phoneme conversion increases when the number of input letters increases.

In the text-to-phoneme mapping application there is a direct link between the number of training iterations and the size of the training dictionary. Specifically, the number of training iterations equals the number of letters in the training dictionary. For some languages there are no large dictionaries available to train the grapheme-to-phoneme conversion systems. In these cases, the text-to-phoneme mapping systems must be able to learn the letter-to-phoneme correspondences from small available resources. For neural networks-based text-to-phoneme mapping systems this is equivalent to fast training which necessitates only a reduced number of training iterations. The convergence speed of the multilayer perceptron neural network can be improved in at least two ways. First the MLP can be implemented in transform domain and second it can be trained with an adaptive learning rate. These two implementations lead to the TDMLP neural network and MLPALR neural network that have also been discussed in this chapter.

The results obtained using the multilayer perceptron, the recurrent and the bidirectional recurrent neural network architectures have been compared in order to verify the influence of the different types of context dependence introduced by these neural network architectures. Multilayer perceptron neural networks utilized contextual information based on the orthography of a word[14]. Letter context was not utilized in the recurrent and bidirectional recurrent neural networks. Instead, contextual information due to the previously transcribed phonemes was introduced by the feedback loop of the recurrent

[14]Several adjacent letters are input together into the neural network in order to find the correct phoneme corresponding to the current letter.

neural network and bidirectional recurrent neural networks. From the simulation results we can conclude that among the compared network architectures, a multilayer perceptron neural network represents a feasible choice in terms of phoneme accuracy and model complexity. An interesting result can be observed comparing the phoneme accuracy for RNN3 and RNN5. Since the phoneme accuracies have been the same for RNN3 and RNN5, we can conclude that in the application addressed here there is no need to train a recurrent neural network model with a truncation depth of more than 3 time steps. Therefore, using a RNN with a small truncation depth we can obtain the same phoneme accuracy but with considerably less computational effort.

The effect of orthogonal and non-orthogonal vectors, used to encode the input letters, have been also studied in this chapter. We have shown that, shorter letter codes provide better phoneme accuracy for very small neural network sizes. Among the different letter encoding schemes, the random vector codes have two main advantages over the traditional binary vector codes. They provide better phoneme accuracy for very small neural network sizes and also faster convergence of the text-to-phoneme mapping system. Due to this fact, using random letter codes, could address two important aspects of practical implementation of the grapheme-to-phoneme conversion: the request of a small memory footprint and the good performance when trained with small dictionaries.

In the next chapter of this thesis the problem of bilingual grapheme-to-phoneme conversion is addressed where we make use of some of the observations and results obtained here.

Chapter 4

Bilingual Text-To-Phoneme Mapping

In Chapter 3 the monolingual text-to-phoneme mapping problem was addressed. In that framework the language of the input text is known a priori and only the phonemes corresponding to the input string of letters must be found during the text-to-phoneme mapping process. This chapter of the thesis is dedicated to the problem of multilingual (specifically bilingual) text-to-phoneme mapping. The difference between this framework and the monolingual grapheme-to-phoneme conversion relies in the fact that the language of the input text is unknown. Due to this fact, the bilingual text-to-phoneme mapping task is more challenging and difficult to solve compared to the monolingual case. We can observe this from a simple example of name pronunciation task. For instance, the same name *Johnny*, is pronounced differently in different languages although it is written in the same manner. However, it only sounds natural when is pronounced in the original language (English) rather than, for instance, the French pronunciation. Moreover, the words which are specific to a certain language (for instance the French word "garçon") might not have any meaning in the context of other languages for instance English. Such words, if they are pronounced independently and a wrong language is assumed (for instance English instead of French), the generated speech is totally meaningless for a human.

There has been a lot of research done in the field of multilingual speech processing systems during the last decades. Thanks to the wide spread of the internet, the users have now access to a huge amount of documents from all around the world. Many times such a document is not written in only one language but it can contain words from several languages [55]. Such documents might be used in various applications, such as document classification based on the languages they contain, automatic translation of the documents from their original language into a new language or text reading. These applications should address both cases of input texts, the monolingual and the multilingual ones. In

either cases the language of the input text must be identified and sometimes, depending on the application, the phonetic transcription of the text must be generated.

Human machine interaction is another important aspect that has been addressed by the researchers from the speech community. Voice enabled web services and email, call center services or voice command and dictation systems are few examples of such applications [1], [23], [44]. Electronic texts, such as e-mails and messages, very often contain words in several languages. If a monolingual text-to-speech system is used to generate speech from such texts, the generated speech sounds unnatural. However, one option would be to use separate TTS systems for each language. This approach also does not give satisfactory results since it does not address, for instance, the problem of intonation. As a consequence, multilingual text-to-speech systems able to produce speech signals from mixed texts, written in more than one language, are needed. These systems must provide a smooth speech signal that must sound like it is read by a multilingual person. On the other hand, voice command systems must deal with the problem of isolated word recognition. In such a case, single words are spoken by a human and the machine must understand the input word and perform according to the spoken command. Since many users, having different mother tongues, can access one such machine, its speech processing interface must be multilingual. For instance route information systems situated in airports must be able to understand queries and provide voice informations in many different languages [37].

In the mobile devices market, the multilinguality is one of the most important requirement of speech processing applications. Since such devices are sold worldwide, speech processing applications, such as voice command and text reading, must cope with a wide range of possible languages. For example, voice command applications can be implemented based on speaker-trained technologies. However, the users are very seldom willing to train such a system therefore, speaker independent versions are of larger interest. Production costs are also a very important aspect of such mass-produced devices and on top of that their computational power and available memory are limited [34], [69]. As a consequence, speech processing applications based on isolated words might be better suited for implementation into mobile devices [69].

Language learning and text translation represents another important application of multilingual speech processing systems [46], [70]. Such systems must provide language learning lessons based on electronic books or just the translation of a text or speech signal. To translate a speech signal it can be first transformed into a text, representing the message of the speech, which is then translated in a different language and a corresponding audio signal is generated. One interesting application of the multilingual TTS systems

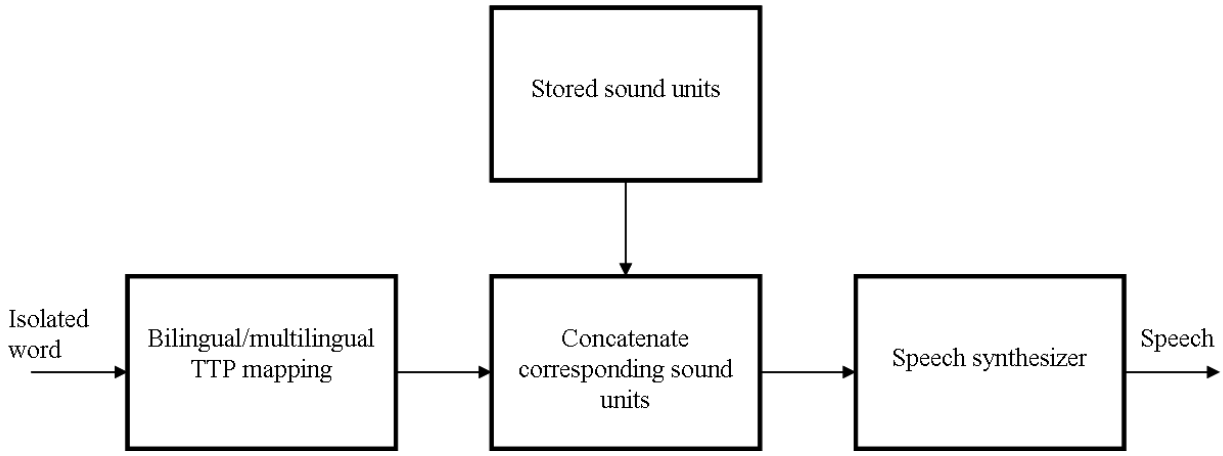

Figure 4.1: A simplified block diagram of a text-to-speech synthesis system.

is to provide language learning lessons based on some text books. For this purpose, the book is first scanned and transformed into an electronic version. The next step would be to generate the speech signal for each lesson from the book. Obviously, it is necessary to perform continous text as well as isolated word text-to-speech and text-to-phoneme mapping processing since such a book would contain portions of text written in two languages as well as explanatory translation dictionaries.

From the above brief review of several applications one could understand the importance of the multilingual text-to-speech processing and in particular of the multilingual text-to-phoneme mapping. A simplified block diagram of a concatenative bilingual/multilingual text-to-speech synthesizer is depicted in Fig. 4.1. The block denoted as "Bilingual/multilingual TTP mapping" is responsible for transcription of the input word into the corresponding phoneme string. It contains a sub-system for language identification as well as several processing blocks that perform translation of the written text into the corresponding phonemes according to the identified language. For each phoneme, of the current word, the block "Concatenate corresponding sound units" assigns a sound unit from a database called "Stored sound units". The sound units from this database are organized according to their language such that, for a given phoneme its sound unit is retrieved from the database corresponding to the language of the input word. The speech is then generated in the "Speech synthesizer" block.

For comparison purposes, in Fig. 4.2 the simplified block diagrams of a monolingual and a bilingual (particularly English and French) text-to-phoneme mapping systems are

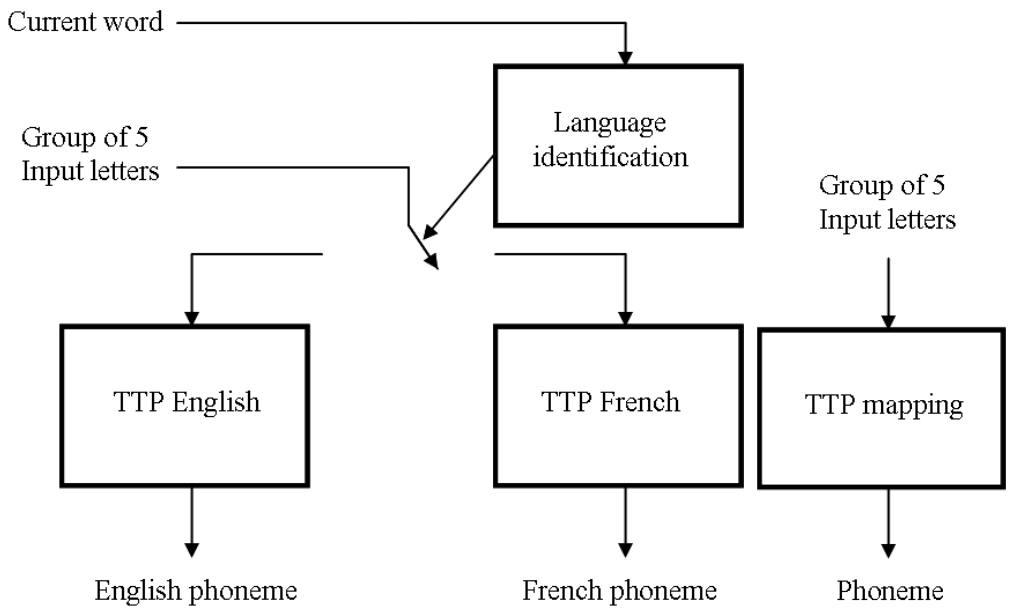

Figure 4.2: The block diagrams of the bilingual (left) and monolingual (right) text-to-phoneme mapping systems.

shown[1]. The bilingual system is composed of a language detection block and two monolingual text-to-phoneme mapping systems. In the bilingual approach, when the translation of an isolated word must be generated, the first step is to identify the language to which the input word belongs. After that, the letters are introduced into the text-to-phoneme mapping sub-system corresponding to the identified language which generates the output phoneme string.

As in the monolingual case, the bilingual grapheme-to-phoneme conversion can also be classified as text-to-phoneme mapping for isolated words and text-to-phoneme mapping for continuous text. In the continuous text case, a sequence of words are mapped and the pronunciation of a word depends on its context (the previous and next words). In the isolated word text-to-phoneme mapping the words are spoken independently and their pronunciation does not depend on the adjacent words. Language identification is a more difficult task when it is applied in the context of multilingual isolated word TTP mapping compared to the case when it is applied to the continuous framework. For instance a simple decision about the language of the input text can be taken when some language specific letters or groups of letters exist. In the case of the French word "garçon", for example,

[1]For the bilingual case other implementations have also been considered in the literature. In this thesis we develop our multilingual system around this block diagram since it ensures a good modularity.

the letter "ç" belongs to the French alphabet and does not belong to the English one. Due to this fact, in a bilingual (English/French) grapheme-to-phoneme conversion system the French language can be assigned with probability 1 to this word. As the number of letters (or words), from which the language is identified, increases the probability to find such special characters (or groups of characters) also increases. As a consequence, when the language is identified based on more than one input word, the accuracy of this module is higher.

One can ask what would be the benefit of multilingual TTP mapping for isolated words. One immediate application is to develop an automatic system for human-machine interaction, as briefly mentioned above. In such a system, for instance, an input word is entered from a keyboard and the machine automatically generates the speech signal according to its pronunciation. Usually, in such systems the language must be known for the machine in order to generate the correct speech signal but a more complex system that automatically can identify the input language can also be of interest. Another benefit to study and develop multilingual text-to-phoneme mapping systems for isolated words is to obtain good methods and algorithms for both language and phoneme transcription tasks. These algorithms can be extended and adapted for the continuous text framework.

This chapter focuses on the bilingual (English/French) text-to-phoneme mapping task for isolated words. We start with by presenting some existing approaches for the multilingual text-to-phoneme mapping task and we continue by introducing a simple system implemented as a combination of neural networks and decision rules. We study next the impact of several factors (such as letter encoding, decision rules and neural network size) in the performance of the initial system. Based on this study we refine our system and finally we propose a bilingual text-to-phoneme mapping procedure that can be easily extended to more languages. Although the multilingual grapheme-to-phoneme conversion for continuous text is left beyond the scope of this thesis, we believe that it can be and interesting and natural extension of the work presented here.

4.1 Existing approaches

In this section we overview some of the previously published multilingual text-to-phoneme mapping approaches and we point out the differences and similarities with our approach. There are many publications dealing with the problem of multilingual TTP mapping. Some of them address mainly the problem of language identification from text [20, 30, 52, 55] while others propose solutions for both language identification from text and for the entire problem of multilingual pronunciation generation [32, 43, 68, 69].

Language identification from written text has been implemented using vector-space based categorizers and N-grams [20, 30, 55, 52], scalable neural networks [68] and decision trees [32] to mention a few. In [55] the "Linguini" system is presented, which was developed to identify the language of portions of text. This system was implemented in the context of continuous text processing and not for the isolated word case, which is addressed in this thesis. However, we mention it here due to its excellent performances.

The main idea of such a system is to construct feature vectors $\mathbf{F}_i$, $i = 1, \ldots, N$ for each of the N known languages during the training phase. In the testing phase a feature vector $\mathbf{D}$ is calculated from the input text and it is compared with all the stored language feature vectors $\mathbf{F}_i$. The language K ($1 \leq K \leq N$) is assigned to the input text if the angle between vector $\mathbf{D}$ and vector $\mathbf{F}_K$ is minimum. Usually, the similarity measure between $\mathbf{D}$ and $\mathbf{F}_i$ is computed as the scalar product of the two vectors. The value of this measure ranges from 0 (corresponding to orthogonality of $\mathbf{D}$ and $\mathbf{F}_i$) to 1 (corresponding to identity or the perfect match of $\mathbf{D}$ and $\mathbf{F}_i$). In [55] the elements of the feature vectors are N-grams (sequences of N consecutive letters) and words. Specifically, in the training phase several portions of text are required for every language. For each of the training texts (languages) the number of occurrences of different N-grams have been counted and the vectors $\mathbf{F}_i$ are constructed. For instance, lets assume that the 3-gram "ary" is selected as the j^{th} feature of $\mathbf{F}_i$. If this 3-gram occurred at least one time in M_j languages $1 \leq M_j \leq N$ its *inverse document frequency* (*idf*) is $1/M_j$. If the "ary" 3-gram occurred $L_j^{(i)}$ times in the training set of a particular language i, its language frequency is $L_j^{(i)}$. Finally, the feature vector $\mathbf{F}_i$ for language i is constructed as follows:

$$\mathbf{F}_i = \left[\frac{L_1^{(i)}}{M_1}, \frac{L_2^{(i)}}{M_2}, \ldots, \frac{L_K^{(i)}}{M_K} \right]. \tag{4.1}$$

where K is the number of features (N-grams and complete words selected as features) and i is the index of a given language.

In (4.1) if a certain feature (N-gram or word) occurs in many languages, at least one time, its *idf* is large and the value of the corresponding element of $\mathbf{F}_i$ is smaller. By this mechanism, N-grams and words that are common to many languages have less importance in the classification process. Moreover, an even stronger selection of features was used in [55]. If some elements of $\mathbf{F}_i$ are smaller than a threshold T (typically between 0.3 and 0.5) the corresponding features are eliminated and not used in the classification process. In this manner N-grams that occur only few times in the training set of a language but they are common to several languages are not used for classification.

In the testing procedure, the feature vector $\mathbf{D}$ corresponding to an input text is com-

Length of the input text	English	French
≈ 20 letters	92.7%	94.6%
≈ 50 letters	99.6%	99.2%
≈ 100 letters	100%	99.8%
≈ 200 letters	100%	100%
≈ 500 letters	100%	100%
≈ 1000 letters	100%	100%

Table 4.1: The language identification performance reported in [55] for English and French and different sizes of the test input texts.

puted in a similar manner by counting the number of occurrences of each feature in the input text. Only N-grams and words that have been selected to construct the language feature vectors $\mathbf{F}_i$ are used to build the vector $\mathbf{D}$. The scalar products between $\mathbf{D}$ and all $\mathbf{F}_i$'s, corresponding to all allowed languages, are computed and the language of the input text is selected the one that have the closest feature vector to $\mathbf{D}$.

In [55] the authors selected 2-grams, 3-grams, 4-grams and 5-grams as well as short words (maximum 4 letters) as features in their implementation. As one can see, this language identification approach is better suited for continuous text processing in which the language of a written text, composed of several words, must be identified. This is due to the fact that the feature vector $\mathbf{D}$ is computed from some statistics gathered from the input text. A single word is not capable to correctly resemble these statistics such that this method might not give satisfactory results in the isolated word case. Moreover, in some cases it might be possible that the input word do not contain any of the selected features such that the vector $\mathbf{D}$ will be the null vector and the language identification would be impossible. However, for input texts composed from several words, this language identification provides excellent performances. In Table 4.1 we show the language identification results for English and French languages presented in [55]. The system was tested with 13 allowed languages for different lengths of input texts and different selected features. The results shown in Table 4.1 are obtained with 4-grams and short words selected as features.

As it was expected, the Linguini system has better performance (100% language identification) for long chunks of texts but its language identification score decreases with the decrease of the input text length.

A neural network-based approach for language identification from text was proposed in [68]. The aim of the research described in that paper was to propose a language iden-

tification solution with a very low memory load suitable for implementation in mobile devices. The authors started from the observation that the number of synaptic weights, in a neural network-based language identification, is proportional to the length of the alphabet. Indeed, since neural networks necessitate numerical values as inputs, the letters must be numerically encoded using orthogonal or non-orthogonal vectors. The length of these vectors increases with the size of the alphabet. Since different languages have different alphabets, a larger set of letters must be used in order to cover all possible letters that might occur. For instance the English alphabet contains 27 letters but a 40 letters alphabet is needed to cover the words from both English and French languages. An alphabet with even larger size would be needed when the number of covered languages increases (in [68] a number of 25 languages were covered by the language identification system which increased the size of the alphabet to 133 letters). The main idea of the approach from [68] is to convert the input alphabet of size 133 into a smaller one called standard set. The standard set could be for instance composed of letters $a, \ldots, z$. Obviously, mapping letters from a 133 alphabet to a 27 letters alphabet will introduce ambiguity.

The language of a given word is then identified using the following equation [68]:

$$\begin{aligned} lang &= \underset{i}{\operatorname{argmax}} \left(P\left(word|lang_i\right)\right) = \underset{i}{\operatorname{argmax}} \left(P\left(word_s, alphabet|lang_i\right)\right) \\ &= \underset{i}{\operatorname{argmax}} \left(P\left(word_s|lang_i\right) \cdot P\left(alphabet|lang_i\right)\right). \end{aligned} \tag{4.2}$$

where $word_s$ is the current word written in the standard alphabet and $word$ is the current word written in the input alphabet.

The first probability from the right-hand side of (4.2) is estimated by the neural network. The second probability $P\left(alphabet|lang_i\right)$, which represents the probability of the alphabet set of $word$ given the language $lang_i$, is estimated as follows:

$$P\left(alphabet|lang_i\right) = \begin{cases} 1, & \text{if } F\left(alphabet|lang_i\right) = 1, \\ \gamma F\left(alphabet|lang_i\right) & \text{otherwise.} \end{cases} \tag{4.3}$$

where γ is a parameter (set at value 0.05 in [68]) and $F\left(alphabet|lang_i\right)$ is the frequency of occurrence of letters from the current word into the language $lang_i$ and is estimated as:

$$F\left(alphabet|lang_i\right) = \frac{\text{number of matched letters}}{\text{number of letters in } word} \tag{4.4}$$

In [68] the following illustrative example on how to compute the probability $P\left(alphabet|lang_i\right)$ for the Finnish name *Häkkinen* is given. Assuming 4 supported languages (English, Finnish, Swedish and Russian) the letter frequency occurrence can be

computed, for each language, using (4.4):

$$\begin{aligned} F\left(alphabet|English\right) &= \frac{7}{8} = 0.875 \\ F\left(alphabet|Finnish\right) &= \frac{8}{8} = 1 \\ F\left(alphabet|Swedish\right) &= \frac{8}{8} = 1 \\ F\left(alphabet|Russian\right) &= \frac{0}{8} = 0 \end{aligned} \tag{4.5}$$

Using $\alpha = 0.05$ in (4.3) the probability $P\left(alphabet|lang_i\right)$ is computed as:

$$\begin{aligned} P\left(alphabet|English\right) &= \frac{7}{8} = 0.04375 \\ P\left(alphabet|Finnish\right) &= \frac{8}{8} = 1 \\ P\left(alphabet|Swedish\right) &= \frac{8}{8} = 1 \\ P\left(alphabet|Russian\right) &= \frac{0}{8} = 0 \end{aligned} \tag{4.6}$$

At the output of the language identification system the n-best decision is used to select the language of the current word. For example in [68] the system was trained on sets of 10^4 common words for every language. The tests were done with two sizes of standard alphabet sets: an alphabet of 27 letters and an alphabet of 30 letters. The proposed system was compared also with the direct implementation in which all the letters from the 25 languages have been included in the alphabet and Table 4.2 summarizes the results.

As a conclusion, decreasing the length of the multilingual alphabet decreases the memory load of the overall language identification system while the performance in terms of language accuracy is similar to the case of a large alphabet. In our work, we have also addressed this problem of decreasing the memory load of our neural network-based system but we followed a different path. Instead of reducing the length of the alphabet, in order to reduce the length of the code vectors, we have used different non-orthogonal and random vector codes with smaller lengths to encode the input letters (these letter codes have been discussed in Chapter 3 of this thesis).

The scalable neural network approach for language identification was also tested for the multilingual text-to-phoneme mapping application in [68]. The results, averaged over all 25 languages are summarized in Table 4.3. From this table two aspects can be concluded. First, the scalable neural network with less than half memory load can ensure similar recognition rate as the neural network-based approach using the whole alphabet.

System setup	1-st best decision	4-th best decision
40 hidden neurons all letters	67.81%	89.93%
30 hidden neurons all letters	65.25%	88.49%
40 hidden neurons 27 letters	57.36%	87.77%
80 hidden neurons 27 letters	65.59%	90.44%
40 hidden neurons 30 letters	64.16%	88.78%
80 hidden neurons 30 letters	71.01%	91.73%

Table 4.2: The language identification accuracy obtained with the approach from [68] based on neural networks.

System setup	1-st best decision	4-th best decision
133 letters	86.69%	93.49%
30 letters	86.79%	93.35%

Table 4.3: The text-to-phoneme mapping accuracy (average over 25 languages) obtained with the neural network approach from [68].

Secondly, using 4-best decision at the output of the neural network highly increases the accuracy of the overall system. The results presented in [68] and summarized in Table 4.3 are the average of the results obtained for all 25 languages. Some of the languages, such as Finnish, have more regular pronunciation and the text-to-phoneme mapping scores are higher for such languages.

Methods based on decision tree have been also utilized for language identification from text as well as for multilingual text-to-phoneme mapping. Such an example was presented in [32] where decision trees have been trained to identify the language of written names and compared with an N-gram based approach. It is well known that decision trees are able to cope very well the contextual information of a given letter and provide good recognition accuracy for words that have been used in their training process. However, they have limited generalization capabilities, such that, their recognition accuracy for new unseen words might drop. This can be seen also from the results presented in [32] where

Language	Enhanced bigram	Decision tree
English	63.00%	87.20%
Finnish	87.50%	94.90%
Spanish	83.60%	88.50%
German	70.40%	93.10%

Table 4.4: The language identification accuracy obtained with the N-gram and decision trees approaches from [32]. Both systems were trained and tested on the training set.

Language	Enhanced bigram	Decision tree
English	66.30%	63.20%
Finnish	84.50%	71.30%
Spanish	71.40%	54.70%
German	65.00%	75.40%

Table 4.5: The language identification accuracy obtained with the N-gram and decision trees approaches from [32]. Both systems were trained on the training set and tested on the testing set.

the N-gram and decision tree based approaches have been tested on the train and also on the test dictionaries for the problem of language identification from text. The system was designed to identify written words belonging to either one of the four languages: English, French, Spanish and German. In Table 4.4 the recognition accuracy of the methods based on N-gram and decision trees are summarized. These results were obtained by training and testing both methods on the training set. In Table 4.5 the results obtained using both methods, trained on the training set but tested on a different test set, are shown.

From these two tables we can see that the accuracy of the method based on decision trees drops when the training and testing sets are different. However, for very short words, the decision tree approach is expected to perform better than the N-gram method that needs a fairly long input text in order to gather enough statistical information.

The performance of both language identification methods have been also tested, in [32], in the context of multilingual TTP mapping. The recognition rates were 92.99% for the decision trees approach and 92.68% for the N-gram approach. We should mention that these recognition rates of the overall text-to-phoneme mapping system are given as average over four languages: English, Finnish, Spanish and German. Since for Finnish a much better recognition rate should be expected, the recognition rates of the other languages might be lower in practice.

Language	Known language	Auto-detected language and phonetic transcription
English	93.30%	85.00%
Finnish	98.10%	95.50%
Spanish	94.50%	91.00%
German	93.10%	89.50%

Table 4.6: The recognition accuracy, of the method from [69], obtained in two scenarios: when the language and the phonetic transcription are known and when the language and the phonetic transcriptions of each tested word are automatically detected.

The problem of multilingual automatic speech recognition, for mobile implementations, was addressed in [69]. The multilingual ASR architecture contains 3 main processing units: the language identification from text module, the text-to-phoneme mapping module and the acoustic modeling module. The input into the system is represented as text and the output consists in the acoustic model of the input word. In this publication the language identification from text was implemented using a similar decision trees approach as in [32]. The grapheme-to-phoneme conversion was implemented using decision trees (for English, German and Spanish) and rule-based systems (for Finnish). The acoustic modeling (generating the acoustic models of the phoneme strings) was done using HMM's. The testing dictionary consisted of 120 isolated commands from 4 languages: English, Finnish, Spanish and German. Their proposed approach was tested in two scenarios in order to verify the influence of language identification and text-to-phoneme mapping modules. In the first testing scenario the language of each tested word and its phonetic transcription were known (assigned by an expert) while in the second testing scenario the language of each tested word and its phonetic transcription were automatically identified. The recognition accuracies, obtained in both scenarios, are illustrated in Table 4.6.

We can see, from this table, that the multilingual automatic speech recognition system performs well when the language and the phonetic transcription of each tested word are known. Not surprisingly, when the language and the phonetic transcription of the tested words have to be estimated the recognition accuracy of the system drops. Another observation is that, the accuracy of the English words drops more significantly when the automatic language identification is included. This is somehow in line with the fact that, the language of the English words is more difficult to be identified. This can also be observed from the results presented in other papers mentioned above. In [69] it was concluded that in an ASR system the errors of the automatic language identification module seems to degrade the overall system performance more than the errors of the

text-to-phoneme mapping part.

4.2 The proposed bilingual text-to-phoneme mapping system

In this section, we describe our approach for the problem of multilingual text-to-phoneme mapping. Our bilingual grapheme-to-phoneme conversion system is composed of three main processing units and its block diagram is shown in Fig. 4.2. In this figure, the block denoted as *Language identification* identifies the language of the input words. The block *TTP English* generates the phonemes of the English words while the module *TTP French* performs the phonetic transcription of the French words. The language identification module, which is detailed more in this section, has been implemented as a hybrid combination of n-gram based decision rules and a multilayer perceptron neural network. The two text-to-phoneme mapping modules have been implemented using multilayer perceptron neural networks. These two processing blocks are monolingual and we do not give here too many details about their implementation since the monolingual text-to-phoneme mapping was treated in Chapter 3 for English language[2]. When a word is presented at the input of the bilingual system, the first processing step is to identify its language. After the language of the entire input word was identified, the word is transferred to the corresponding grapheme-to-phoneme conversion module and its phonetic transcription is generated.

Although our system shares some features in common with the existing approaches presented in the previous section, our research work was focused in a slightly different direction. Similarly to [68], we have implemented our language identification module and the GTP conversion part using neural networks. We are also interested to obtain a low complexity system that has a small memory footprint and low computational cost while keeping as high as possible the recognition accuracy.

In order to decrease the computational complexity and the memory load of our system we also try to decrease the number of inputs in the neural network, as the authors of [68] did. However, we propose a different solution. We implement different letter encoding schemes that generates shorter input vectors. Similarly to [20], [30] and [55] we use N-grams to improve the performance of the NN-based language identification module. In our approach N-grams are used taking into account the specific problem we address: the isolated word text-to-phoneme mapping.

[2]The *TTP French* module is implemented in a similar manner.

More than that, in our work we have been interested mainly in improving as much as possible the performances of the baseline system, therefore we did not implemented any correction method at the output of the system. Of course n-best decision methods, such as the one used in [68], would increase the accuracy of our system. We believe that the combination of our method with any of the methods described in the previous section could improve the recognition accuracy.

In the rest of this chapter we give a detailed description of our proposed text-to-phoneme mapping system. We begin with the description and the pre-processing of the database used. We continue with the details of the different encoding schemes used for letter encoding. The language identification module is presented next and we show the language identification results obtained for a bilingual (English and French) database. Finally, we introduce the complete bilingual grapheme-to-phoneme conversion system and we illustrate its performance in terms of phoneme accuracy.

4.2.1 The pre-processing of the bilingual database

When monolingual or multilingual text-to-phoneme mapping systems are implemented, they must be trained first on some set of words for which the phonetic transcription and the language are known. Usually, the databases available for training the grapheme-to-phoneme conversion systems are builded in a form of a list of words and their phonetic transcriptions. Some of the words in the database can have multiple pronunciations due to their context. Moreover, other linguistic information, such as accents and stress, might be included in the original database. According to the specific application addressed, several information from the database can be eliminated. For instance, the accents and the stress information have been eliminated from the database in our experiments and only one phonetic transcription of each word was retained. As a consequence, prior to implementation and training of a text-to-phoneme mapping system, the available database must be pre-processed. In this section, we review the steps for database pre-processing and, in the next section, we describe the encoding method for letters, phonemes and language tags. The dictionaries used for training and testing the neural networks in our experiments were the Carnegie Mellon University pronunciation dictionary [75] for the English words and the Brulex dictionary [28] for the French words. We denote these dictionaries as *cmu.dic* and *brul.dic* respectively. The *cmu.dic* contains 108080 English words and the *brul.dic* contains 32245 French words. Since here we deal with the problem of bilingual text-to-phoneme mapping, the database is pre-processed in a slightly different manner compared to the monolingual case described in Chapter 3. For the bilingual case

both dictionaries have been pre-processed as follows:

1. In order to eliminate the ambiguity that can occur for multiple pronunciations of the same word and due to the fact that we address the problem of isolated word TTP mapping, only one phonetic transcription was chosen for each input in the dictionary.

2. The words and their phonetic transcriptions were aligned, such that there is a one-to-one correspondence between letters of each word and its phoneme symbols [25]. Corresponding to the letters that have no pronunciation a so-called *null phoneme* (_) is introduced in the phonetic transcription in order to have equal numbers of letters and phonemes for a given word. In the case when the phonetic transcription of a single letter consists of two or more phonemes they are combined together in a compound phoneme. For instance the word *ox* have the phonetic transcription *A: c s*. In this case the phonemes 'c' and 's' are combined together in the compound phoneme 'cs' that corresponds to the letter 'x'. After the alignment, the number of the letters and the number of phonemes are equal for each entry in the dictionary. This alignment is important since, in the training process, for each input letter the text-to-phoneme mapping system requires a so called desired phoneme. To exemplify the alignment procedure, few words from the aligned training dictionary are shown in Tab. 4.7.

3. Both *cmu.dic* and *brul.dic* were split into two parts. From the *cmu.dic* we have randomly chosen 80% of the words for training (each word with a single phonetic transcription). The obtained training dictionary was denoted as *train_en.dic* and contained 86464 words. The remaining 20% (21616 words) from the *cmu.dic* formed the testing dictionary *test_en.dic*. In the same manner *brul.dic* was split into *train_fr.dic* (25796 words) and *test_fr.dic* (6449 words). In addition to these files we have obtained the file *train_en_fr.dic* by concatenation of *train_en.dic* and *train_fr.dic*. The files *train_en.dic*, *test_en.dic*, *train_fr.dic* and *test_fr.dic* were used for training and testing the neural networks responsible for text-to-phoneme mapping of a single language (English or French). The *train_en_fr.dic* was used to train the neural network for language recognition.

4. The order of the words in the training and in the testing dictionaries was randomized. This was done in order to increase the modeling capability of the neural network modules [33]. After that, each letter in a word was encoded using orthogonal vector

Word	Corresponding transcription after alignment
agglomeration	ah _ g l aa m _ er ey _ sh ah n
aggrandizement	ae _ g r ah n d ay z _ m ah n t
aggravate	ae _ g r ah v ey t _
aggravated	ae _ g r ah v ey t ah d
aggravates	ae _ g r ah v ey t _ s
aggravating	ae _ g r ah v ey t ih _ ng
aggravation	ae _ g r ah v ey _ sh ah n
aggregated	ae _ g r ah g ey t ah d
aggregates	ae _ g r ah g ih t _ s
aggression	ah _ g r eh _ _ sh ah n
aggressions	ah _ g r eh _ _ sh ah n z
aggressive	ah _ g r eh _ s ih v _
aggressively	ah _ g r eh _ s ih v _ l iy
aggressiveness	ah _ g r eh _ s ih v _ n _ ah s
aggressor	ah _ g r eh _ s _ er
aggressors	ah _ g r eh _ s _ er z
aggrieved	ah _ g r iy _ v _ d

Table 4.7: Words from the aligned training dictionary.

codes (see Table 4.8) or random codes. Letter encoding together with phoneme and language encoding are further described in the next sub-section.

4.2.2 Letter, phoneme and language encoding

In our approach, the bilingual system produces the transcription of English and French words only. As a consequence, the letter encoding is based only on the English and on the French alphabets[3]. The English language contains 26 letters and the French language contains 39 letters and we note that the French alphabet contains all the English letters. As a consequence, it is enough to have an encoding for the French alphabet. We have used in the experiments of this chapter two letter encoding schemes: binary orthogonal codes and randomly generated codes. Other codes have been studied for instance in [11], [13] and Chapter 3 for the monolingual case. Since there are 39 letters in the French alphabet the binary vectors must have length 40 to encode 39 letters plus the *graphemic null* (denoted as \0). The *graphemic null* is used to represent the spaces between words.

[3]If more languages must be introduced the letter encoding must be changed. Alternatively, an approach using a reduced alphabet, similar to [68] can be used.

Letters	Corresponding binary codes of length 40
\ 0	1 0 0...0
a	0 1 0...0
b	0 0 1...0
⋮	...
ü	0 0 0...1

Table 4.8: Binary codes used to encode the input letters. The elements of the vectors are zeros except one which equals unity and is placed on the position corresponding to the letter index.

Phonemes	Corresponding binary codes of length 62
_	1 0 0...0
A	0 1 0...0
A:	0 0 1...0
⋮	...
ä	0 0 0...1

Table 4.9: Binary codes used to encode the phonemes. The elements of the vectors are zeros except one which equals unity and is placed on the position corresponding to the phoneme index.

Examples of the binary letter codes are shown in Table 4.8. Several studies have concluded that orthogonal codes provide better phoneme accuracy than non-orthogonal codes when neural networks are used for text-to-phoneme mapping (see [14], [39]). This is why we have selected these orthogonal encoding schemes in our approach.

The letter codes shown in Table 4.8 are not the only orthogonal codes that can be implemented. It is possible to use also some orthogonalization methods as the one described in Chapter 3, in [11] and in the references therein. Another possibility for letter encoding is to randomly generate the input letter codes. In this case, for each of the 39 letters and the *graphemic null*, we have selected the elements of the vectors, representing the letter codes, from a random zero-mean Gaussian-distributed sequence with unity variance. The length of the vector codes was 40. Also in this case, it is possible to select the codes in several manners, for instance by using different distributions and different parameters. However, we have limited our research to only the Gaussian case and we have left the other alternatives for future work.

In our system we have used neural networks for both text-to-phoneme transcription

and language identification and a similar phoneme encoding approach as described in Chapter 3. The phonemes have been encoded using binary vectors with elements 0's except one unity element which corresponds to the phoneme index (there are 62 English and French phonemes together). Examples of phoneme encoding are shown in Table 4.9 where in the first line the vector of the *null phoneme* is shown. For language encoding we have used the binary codes from Tab. 4.10.

Languages	Corresponding binary codes of length 2
English	0 1
French	1 0

Table 4.10: Binary codes used to encode the two languages (English and French).

The reason to use this encoding scheme comes from the implementation of the neural networks. At the output of the neural networks we have used the softmax activation function. In this case, one of the neural network outputs will have the maximum value among others and corresponds to the index of the recognized phoneme or language tag. As a consequence, these binary codes are well suited for encoding the phonemes and language tags in this context.

4.2.3 The language identification module

In this section of the thesis the language identification module implemented in our bilingual TTP mapping system is described. We emphasize here that the allowed languages are English and French, such that, the system can assign words to only either one of these languages. We start with a simplified system composed of a combination of multilayer perceptron neural network and a set of simple decision rules. Further, we add some new rules to the system that allows better language identification accuracy. Finally, we provide experimental results showing the performance, in terms of language identification accuracy, and the improvement obtained by adding more rules into the system.

The first problem that arise in the case of bilingual/multilingual text-to-phoneme mapping is the fact that the language of each input word is unknown. If a very simple system that does not take into account the language of the input words is implemented, the achieved phoneme accuracy is very poor [15]. A much higher phoneme accuracy is obtained when the language identification (or some prior information about the input language) is included into the system. Due to this fact, the usual way to address the multilingual approach is to first identify the language of the current word and the

text-to-phoneme mapping is done accordingly to the identified language. An example of a simplified block diagram showing the main components of a bilingual (English and French) grapheme-to-phoneme conversion system is shown in Fig. 4.2. Throughout this chapter we rely on this block diagram to implement our bilingual text-to-phoneme mapping system. A similar system architecture was used also in [66] for multilingual optical character recognition. In our text-to-phoneme mapping system five adjacent letters are taken as inputs to the system. The central (third input letter) is the current letter that must be mapped to the corresponding phoneme, the first 2 input letters represents the left context dependency and the last 2 input letters form the right context dependency. As a first step, the language recognition module identifies the language of the current word and selects the corresponding sub-system for text-to-phoneme mapping. Once the language of the whole input word was identified, the input letters are transmitted to the corresponding grapheme-to-phoneme conversion sub-system (either TTP mapping English or TTP mapping French) and the corresponding phonemes are found. We should make an important remark here: the language identification module first scans the entire input word and identifies its language. After that, the word is scanned again by the corresponding text-to-phoneme mapping sub-system that assigns a phoneme to each of its letters. The functionality of the entire bilingual text-to-phoneme mapping system will become more clear during this chapter.

In our approach the implementation of the language recognition part is done by means of a hybrid system composed of a multilayer perceptron neural network and a decision rule block as depicted in Fig. 4.3. The input of the system consists on 5 adjacent letters whose language must be recognized by the system. The language identification is done by means of a neural network (denoted as MLP5 in Fig. 4.3) and decision rule sub-system (the decision block in Fig. 4.3). Usually, the decision block contains some simple "if' and "else" statements (rules) that are based on some known observations and differences between the possible languages.

We start from the following simple procedure applied to all letters of a word in order to identify the language they belong:

1. Take 5 adjacent letters (the current tested letter and 2 letters on both sides of it), from the current word, and check if the middle letter belongs to the specific French letters. If the middle letter is French specific, all the letters of the current word are labeled as French. This rule is based on the fact that, French dictionary contains 13 letters that are specific to the French language (they belong to the French dictionary and are not contained in the English dictionary). If one of these specific letters are

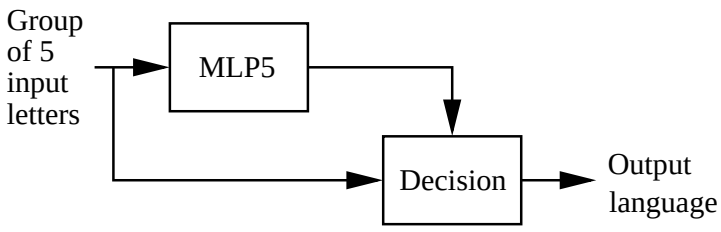

Figure 4.3: The block diagram of the hybrid system implemented for language identification from text.

found in one word, the entire word is labeled as French.

2. If the current letter was not a French specific letter the first 4 input letters are selected. If these letters have a unique language correspondence, all letters of the current word are labeled with this language. This decision rule is based on the following observation: in [15] it has been performed a simple test on a large training dictionary to verify how many combinations of 4 adjacent letters belong to only one language (English or French). It was observed that about 74% of the combinations of 4 adjacent letters have a unique language correspondence. This language correspondence is saved in a look-up table. Due to memory limitations only combinations of 4 letters and their corresponding unique language labels have been stored [15].

3. If the language was not identified in the previous 2 steps, a neural network is used for language assignment. The 5 input letters are input to a multilayer perceptron neural network and its output will be used to select language corresponding to the middle input letter.

   The input vector $\mathbf{I}(i)$ and the output vector $\mathbf{O}(i)$ of the neural network are given by:

$$\begin{aligned} \mathbf{I}(i) &= [I_1(i), I_2(i), I_3(i), I_4(i), I_5(i)]\,, \\ \mathbf{O}(i) &= [O_1(i)\ O_2(i)]\,. \end{aligned} \tag{4.7}$$

   where $I_3(i)$ is the binary code corresponding to the current letter and i is its position in the current word. The vectors $I_1(i)$ and $I_2(i)$ are the binary vectors corresponding to the two letters situated on the left of the current letter and $I_4(i)$ and $I_5(i)$ are the vectors corresponding to the two letters situated on the right hand side of the current letter.

   At the output of the neural network the softmax activation function is used, such

that, $O_1(i)$ can be viewed as the probability of the input to belong to English language and $O_2(i)$ as the probability of the input to be French text.

The language corresponding to the current input letter is then decided as follows:

$$L(i) = \begin{cases} \text{English} & \text{if} \quad O_1(i) > O_2(i) \\ \text{French} & \text{if} \quad O_1(i) < O_2(i) \\ \text{Undecided} & \text{if} \quad O_1(i) = O_2(i) \end{cases} \tag{4.8}$$

We note that, at the beginning of the word, the vectors $I_1(1)$ and $I_2(1)$ correspond to the so called *graphemic null* ($\backslash 0$) since there are no letters on the left side of the first letter. Similarly at the end of the word $I_4(K)$ and $I_5(K)$ corresponds to the *graphemic null* (K being the number of letters in the current word).

At the end of the word the number of letters that have been labeled as English and the number of letters that have been labeled as French are counted. If there are more English letters than French the whole current word is labeled as English and it is labeled as French otherwise.

There are two problems that can decrease the language accuracy of the above mentioned approach. Firstly, in (4.8) if the outputs of the neural network are equals for a certain letter the language of that letter cannot be decided. The second problem may arise when the neural network identify the same numbers of English letters and French letters in one word. In this case, the language of the entire word cannot be identified. A modification that address these two problems is proposed in the sequel.

We continue by introducing a modification of the language identification algorithm described above which improves the language recognition. Our new method follows the same steps described above. More specifically, we use the same rules to assign the language to words that contain French specific letters or groups of letters with unique language correspondence. The difference relies on the language selection for words that were entirely processed by the neural network. When the multilayer perceptron neural network is used to assign the language for all letters of a word (the word did not contain French specific letters nor combinations of letters with unique language correspondence) at the end of the word the number of letters labeled as English and the number of letters labeled as French are counted. If there were more English letters the entire word is labeled as English and if there were more French letters the entire word is labeled as French.

In the case of equals numbers of letters, classified as English and French in the current word, the probability p_{en} that all letters belong to English and the probability p_{fr} that

all letters belong to French are computed as follows:

$$p_{en} = \prod_{i=1,\ldots,K} O_1(i), \quad p_{fr} = \prod_{i=1,\ldots,K} O_2(i). \tag{4.9}$$

where K stands for the total number of letters of the current word.

In this case of equal number of letters classified as English and French the language of the current word is assigned as:

$$L_W = \begin{cases} \text{English} & \text{if} \quad p_{en} > p_{fr} \\ \text{French} & \text{if} \quad p_{en} < p_{fr} \\ \text{Undecided} & \text{if} \quad p_{en} = p_{fr} \end{cases} \tag{4.10}$$

At least theoretically, in (4.10) it is possible that a certain word cannot be classified. This would be the case where the number of letters classified as English and the number of letters classified as French are equals and also the two probabilities, p_{en} and p_{fr} are equals. One could argue that the condition imposed by (4.9) is to tight and a more relaxed condition could be used instead. For instance we can impose that the probability of most of letters to be English in order to classify the word as English. From our experiments, done on a very large database, we have observed that at least in the case of the bilingual (English+French) dictionary, there are no words that could not be classified by our method. This does not suggest that the language identification module have 100% accuracy but it means that there were no words that have language label *Undecided*. A block diagram showing the processing steps done for language identification is shown in Fig. 4.4.

4.3 Experiments and results

In this section, we present the experimental results obtained with the proposed bilingual text-to-phoneme mapping system equipped with both language identification modules described above. The training parameters and the sizes of the neural networks in all experiments were equals. In the training process the synaptic weights of all neural networks have been initialized with random values uniformly distributed in the range $[-1,\ 1]$ and the training algorithm was the error back-propagation with momentum described in Chapter 2.

We present first the experiments done in order to verify the performance of the language identification module. We continue with a study on the influence of the letter encoding and neural network size into the phoneme accuracy of the bilingual grapheme-to-phoneme conversion system. Using the results of this study, we implement our final

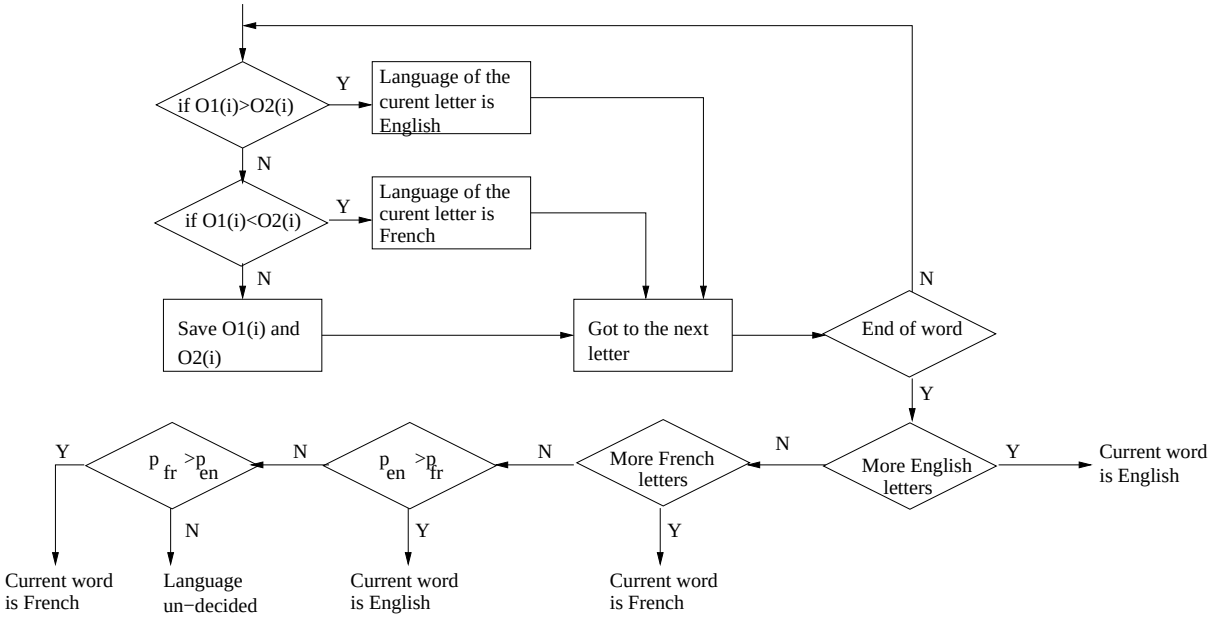

Figure 4.4: The block diagram of the method implemented at the output of the neural network for language identification.

system and we present its performances, in terms of phoneme accuracy, for both English and French languages.

4.3.1 Experiments and results for the language identification module

We show here experimental results comparing the two language identification approaches described above. A three layered multilayer perceptron neural network, with 3049 synaptic weights, was implemented in both approaches and it was trained with the error back-propagation with momentum algorithm. The synaptic weights have been updated at each iteration using a fixed learning rate $\lambda = 0.1$ and a momentum constant $\alpha = 0.9$.

In both implementations we have used, in the training of the multilayer perceptron, only the letters that were not French specific and do not have a unique corresponding language. This is due to the fact that those letters or groups of letters represent outliers for the neural network and can decrease the training accuracy. This issue has been observed and explained also in [15].

The multilayer perceptron neural network has 5 adjacent letters as inputs each of which are encoded by orthogonal binary vectors of length 40. Of course, one can use a

larger number of input letters in order to increase the language identification accuracy but this highly increases the number of synaptic weights when the number of hidden neurons is kept constant. In our experiments, we have chosen to keep a relatively large number of hidden neurons and a low memory load for our system. Taking into account these imposed restrictions we have been forced to use an input window of only 5 adjacent letters in the neural network. The MLP neural network has 2 outputs that encode the recognized language (English or French). The neurons from the hidden layer have been equipped with hyperbolic tangent activation functions whereas in the output layer we have implemented the softmax activation function for both approaches. A detailed description of the training algorithm was given in Chapter 2.

In our experiments we have used the Carnegie Mellon University pronunciation dictionary which contains 108080 English words and the Brulex dictionary that contains 32245 French words. We note that the Brulex dictionary has less number of words. In order to improve the language recognition for French words the entries of the Brulex dictionary were repeated 3 times and then the result was concatenated with the CMU dictionary to obtain a bilingual set of words[4]. The training and the testing sets as well as the pre-processing of the database was performed as described in Section 4.2.1. The letter, phoneme and language encoding was done according to the procedure outlined in Section 4.2.2. For letter encoding, we have used the binary codes shown in Tab. 4.8 in which every letter code is a vector of length 40 and has null elements except the one situated in the position of the letter index. Similar binary codes were used to encode the two languages as shown in Tab. 4.10. These binary language codes model the probability of the corresponding letter to be either English or French (if the language code has a unity element on the first position, the probability of the corresponding letter to be English is 1). As a consequence, the use of the softmax activation function at the output of the neural network, motivates the language encoding used here.

During training the synaptic weights of the neural networks have been saved at 1%, 2%, ..., 100% from the training dictionary. The saved synaptic weights have been used then to test the language recognition performance. In this manner, we have obtained two plots of the language recognition during training which are shown in Fig. 4.5. By this kind of training we wanted to simulate the so called training with validation where the training of the neural network is stopped from time to time and the performance of the system is evaluated on a validation set. In the text-to-phoneme mapping application (whether it is for a single language or for multiple languages) the training is done usually

[4]However, repeating some words in the training dictionary introduces some redundancy. A better way would be to train the system on a larger database if this is available.

in off-line mode. This means that the synaptic weights of the neural networks are trained first on some training set and after that their final values are used. Saving the synaptic weights at intermediate steps (1%, 2%, ..., 100% from the test set) makes possible the selection of the best synaptic weights.

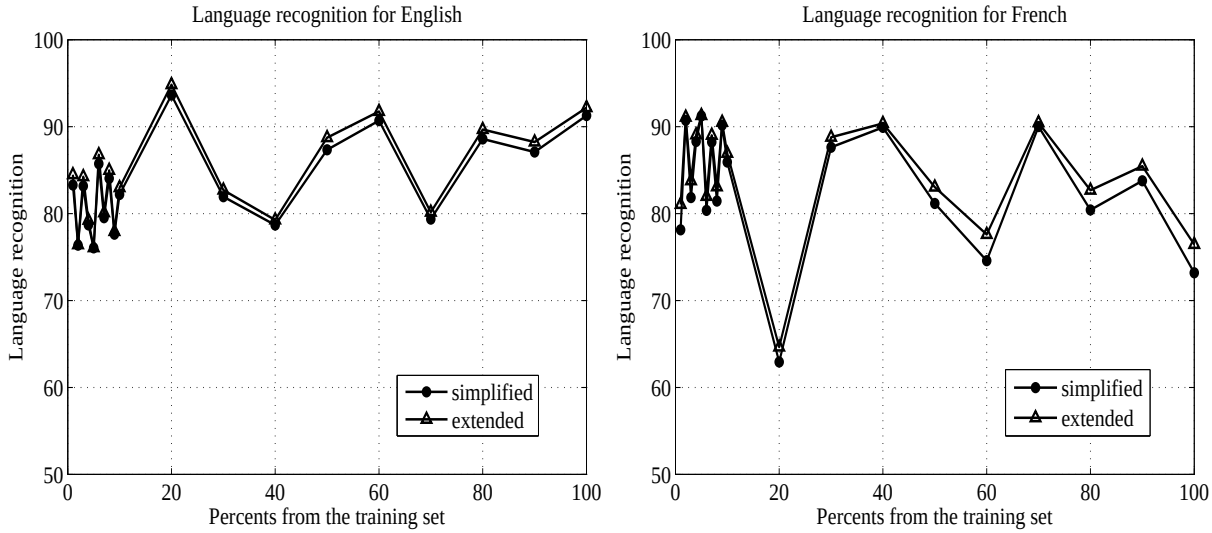

Figure 4.5: The language recognition for: English (left) and French (right).

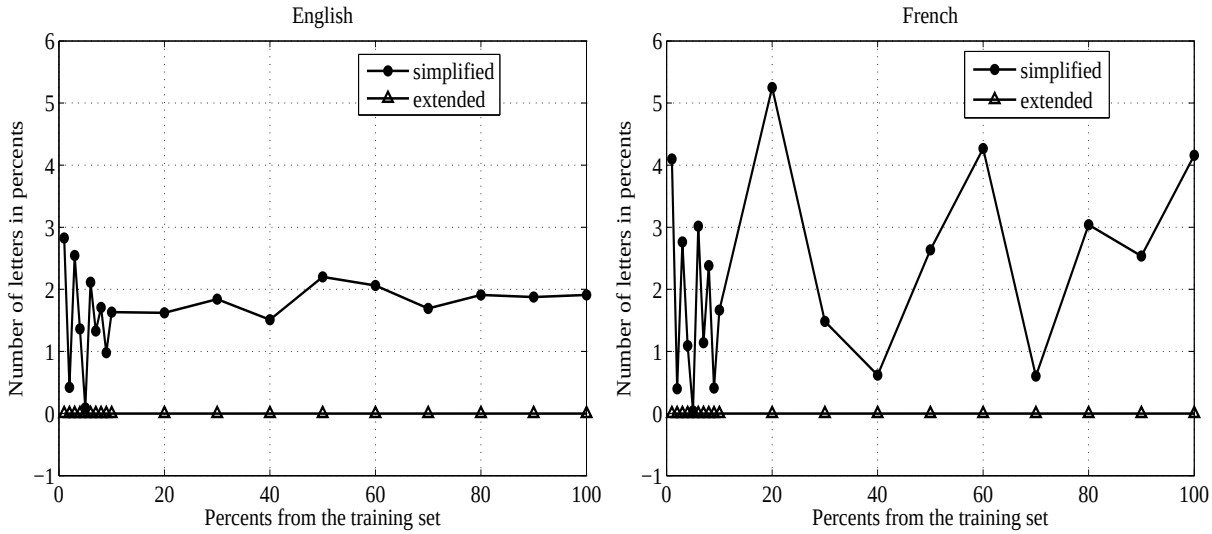

Figure 4.6: The number of letters, in percentage from the whole testing set, that belong to words for which the language could not be decided: English (left) and French (right).

In Fig. 4.5 the English and French language recognition accuracy are shown for both implementations. From these results we can see that, the language recognition for English

words and French words is improved when the language assignment of (4.9) and (4.10) is added into the system. In this figure we have shown the percentage of the correctly labeled letters.

In order to clearly see the improvements obtained with (4.9) and (4.10), in Fig. 4.6 we show the percentage of letters that belong to words whose language could not been decided. In the case of the more complex system there are no such letters whereas in the simplified approach there are on average 1% English and 2% French letters belonging to words that could not be classified.

4.3.2 The influence of the letter encoding and neural networks size into the phoneme accuracy of the bilingual text-to-phoneme mapping system

In these experiments we have tested our bilingual text-to-phoneme mapping system with different sizes and different encoding of the letters. First, we did a set of experiments in which we have varied the size of the neural networks responsible for language recognition and text-to-phoneme mapping for the English and French languages. The aim of these experiments was to study the performances of the system for several neural network sizes.

The results, in terms of phoneme accuracy, for both English and French language are shown in Table 4.11 for the case when the input letters were encoded using random codes and in Table 4.12 for the case of binary encoding of the input letters. Comparing the results shown in Table 4.11 and Table 4.12 we can see that better phoneme accuracy for French language is obtained with random codes for sizes larger than 10500 synaptic weights while English words are transcribed more accurate when binary codes are used. We note also that the phoneme accuracy for French words is about 5% larger when random codes have been used to encode the input letters compared to the case of binary codes, while the phoneme accuracy for the English words only drops by 2% (see for example the values obtained for 10500 synaptic weights). One reason can be found in the language recognition step which provides better results in the case of random codes. This is also demonstrated by the results reported in [11], [13] and Chapter 3 where it was shown that random codes can improve the phoneme accuracy of small sized neural networks for monolingual TTP mapping (in our experiments the smaller neural network was used in the language recognition sub-system). Moreover, we have seen in Chapter 3 that random codes are more suited for training the multilayer perceptron neural network with small training dictionaries (they ensure increased convergence speed of the grapheme-to-phoneme conversion system). Since in our experiments we have had only a small French

Number of synaptic weights	English	French
2100	49.42%	44.23%
10500	77.64%	79.84%
16000	78.68%	80.62%
22000	78.57%	81.05%

Table 4.11: Phoneme accuracy, for different number of synaptic weights, obtained with the proposed hybrid approach. The input letters were encoded using random codes.

Number of synaptic weights	English	French
2100	65.92%	65.03%
10500	79.44%	74.81%
16000	80.51%	75.32%
22000	80.53%	79.09%

Table 4.12: Phoneme accuracy, for different number of synaptic weights, obtained with the proposed hybrid approach. The input letters were encoded using binary codes.

dictionary available, random codes are more suited for the TTP French module. This leads us to the conclusion that different input letter encoding schemes can be used for the 3 neural networks involved.

4.3.3 The final implementation

Based on the above conclusions, we have implemented the final bilingual text-to-phoneme mapping system in a slightly different manner. The architecture of the system is similar to the one depicted in Fig. 4.2 with the main difference that we have used random vectors to encode the inputs of the language recognition and French TTP mapping modules. In the grapheme-to-phoneme conversion module responsible for the translation of the English words we have used binary encoding of the input letters. As a consequence, the final bilingual system must contain a module that translates the binary codes into random codes. We have implemented this module as a look-up table.

In Table 4.13 we show the comparative results obtained with the proposed approaches when the input letters into all 3 neural networks were encoded using binary and also random codes. In the second, third and forth columns of the table the phoneme accuracies obtained with the text-to-phoneme mapping system equipped with the more complex language identification module are shown for situations where different letter encoding

Language	Orthogonal encoding for letters and languages (extended set of rules)	Random encoding for letters and languages (extended set of rules)	Orthogonal letter codes for English NN. Random letter codes for French NN. Random letter codes for language identification NN (extended set of rules).	Orthogonal encoding for letters and languages (reduced set of rules)
English	80.53%	78.57%	80.05%	80.04%
French	79.09%	81.05%	81.21%	73.86%

Table 4.13: Comparison between the proposed bilingual hybrid system and the bilingual system from [11].

schemes have been used. In the fifth column the simplified set of rules have been used in the language identification module and the letters have been encoded using binary orthogonal codes. We can see from these results that improved performance is obtained with the proposed hybrid approach that uses an extended set of decision rules, especially for the French language. From the four compared implementations the one that uses different input codes for the 3 neural networks gives improved phoneme accuracy.

The results shown in Table 4.13 might look on the low side since, for instance, in [27] phoneme accuracy levels around 90% was reported for English language. However, the results reported in [27] have been obtained in a different framework. In that publication the single language approach was studied while in this paper we address the problem of bilingual TTP mapping. As one can expect, the phoneme accuracy in our approach, for both English and French words, drops due to the imperfect language identification. There are also differences in the topology of the neural networks implemented and in the selection of the training dictionary. For instance in [27] non-symmetric windows were used to include context dependence between adjacent letters while in our approach we have used symmetric ones.

At the beginning of this chapter, several solutions for the text-to-phoneme mapping, published in the open literature, have been presented. Some of them show better phoneme accuracy than our proposed bilingual system. However, the reader should keep in mind that our experiments have been done in a different framework and no other corrections have been added to our system. For instance, n-best decision can be used at the output of our system to improve the phoneme accuracy as it was implemented in [68].

4.4 Conclusions

This section of the thesis addressed the problem of bilingual text-to-phoneme mapping. We started with an introduction in the field of multilingual text-to-phoneme mapping reviewing several different aspects that must be addressed. We presented various applications that make use of a multilingual TTP mapping showing its increasing importance. We continued our discussion by briefly presenting several different approaches used to implement multilingual grapheme-to-phoneme conversion systems. The results of the existing approaches have been also presented together with the framework in which they have been tested.

The section continued with the detailed description of our approach. First, we briefly discussed about the similarities and the differences between our method and the ones published already. During this discussion we emphasized the main goal of our research work and the specific problem we have addressed. We continued by describing the databases used and their pre-processing and then we gradually introduced, block by block, our text-to-phoneme mapping system. Finally, we provided experimental results showing the performances of the proposed implementation mentioning also several direction for further improvements.

Chapter 5

Conclusions

This thesis provides a study about the application of different neural network architectures and letter codes for the problem of text-to-phoneme mapping. It also introduces two new algorithms for fast training of the neural networks and its contributions have both theoretical and practical importance. The new algorithms proposed here are derived from the well known error back-propagation with momentum algorithm, which is widely used for training multilayer perceptron neural networks due to its simplicity. However, despite its simplicity, the multilayer perceptron neural network trained with the error back-propagation with momentum algorithm can have some drawbacks which are discussed in the thesis. The goal was to address and study each of these drawbacks and to provide solutions to improve the performances in terms of convergence speed, phoneme accuracy and memory load of a neural network based text-to-phoneme mapping system.

In the context of grapheme-to-phoneme conversion, the multilayer perceptron neural network shows very good performance in the sense of phoneme accuracy and simplicity of implementation and training [14, 15, 33]. However, the MLP can have sometimes a very high complexity (a very large number of synaptic weights are required to obtain good phoneme accuracy results), when the input vector has a large dimension. This, for instance, can be due to the use of long input codes. Another drawback of the multilayer perceptron neural network is its slow convergence (the large number of iterations necessary for the neural network to attain its stability point). The slow convergence, however, has two sides when the TTP mapping is considered. On one side, the convergence speed measures the time necessary to finalize the training of the neural network and on the other side, it is inversely proportional to the size of the training dictionary. As a consequence, simple neural network structures and fast training algorithms are of great interest for practical purposes.

In the thesis the multilayer perceptron, recurrent and bidirectional recurrent neural network architectures have been implemented and tested for the problem of text-to-phoneme mapping. The main reason to use these three network architectures is to study the suitability of different neural network architectures to the text-to-phoneme mapping task.

In the case of multilayer perceptron, if just one letter was used as the input to the network no contextual information would be taken into consideration. Therefore, in the experiments several number of letters have been considered at the input of the MLP (for instance 3, 5 and 7 letters). The Recurrent Neural Network architecture considered in our study contains feed-back loops from each output neuron to the input neurons. Due to the presence of these feed-back loops the transcription of the current phoneme depends on the previously recognized phoneme and the current letter. The previously recognized phoneme depends on the previous letter and on the phoneme before it. In this manner a recurrent dependence between the current recognized phoneme and all letters from the beginning of the input sequence is constructed. Therefore, in the considered recurrent neural network architecture the left context dependence is included into the feed-back loop. The study of introducing the letter context at the input of the recurrent neural networks is left beyond the scope of this thesis. In the bidirectional recurrent neural network implemented in the experimental work shown in this thesis, the contextual information from both sides of a letter is included due to the network structure (by using the feedback and feed-forward loops).

From the experiments presented here, it can be concluded that the recurrent neural network provides smaller phoneme accuracy than the multilayer perceptron neural network with context dependencies from both sides. However, when multilayer perceptron has only the left context, its phoneme accuracy and and the phoneme accuracy of the recurrent neural network are approximately the same. Furthermore, the memory needed for storing the weights is much smaller in the case of recurrent neural networks. Also, it was shown that the training of the recurrent neural network can be performed by using a small truncation depth without loss in the phoneme recognition accuracy.

Convergence speed is another issue that is addressed in this thesis. To this end, two new training algorithms for the multilayer perceptron neural network have been derived. Both algorithms provide high convergence speed while maintaining a high phoneme accuracy. One of the proposed algorithms uses a time-varying learning rate in the training process of the multilayer perceptron while the second algorithm is the transform domain implementation of the MLP neural network. Both approaches are different that the ones existing in the open literature and have smaller computational complexity and memory

load.

Further, it has been shown that the codes used to encode the input letters have a large impact on the performance of the multilayer perceptron neural network. Due to this fact this thesis provides a study of the performance obtained with different orthogonal and non-orthogonal letter codes. This study is of interest in practical implementations because it reveals the link between the size of the neural network and the most suited type of the letter code.

The monolingual and the bilingual text-to-phoneme mapping problems have been addressed in our work. For the bilingual case the proposed text-to-phoneme mapping system consists of three main modules: the language identification from text, the grapheme-to-phoneme conversion module for English words and the text-to-phoneme mapping module for French words. For the language identification module, a hybrid approach have been implemented which combines a neural network and a set of decision rules. Its performance for the bilingual isolated text-to-phoneme mapping have been presented and compared to other existing methods.

As a final conclusion, we can state that this thesis succeeded to introduce several solutions to improve the convergence speed of the multilayer perceptron neural network. It is also a comprehensive study about the performances of different neural network structures to the problem of text-to-phoneme mapping. The thesis provides some guidelines for practical implementations of monolingual and bilingual TTP mapping systems using neural network. Finally, a bilingual text-to-phoneme mapping system is introduced which contains a new hybrid language identification from text module.

5.1 Future work

The thesis also opens new research directions in the field of text-to-phoneme mapping. Based on the developments and the experimental work, presented here, several ways to improve the proposed methods can be identified.

For instance hybrid systems, implemented as a combination of decision trees or N-grams and neural networks, could also be implemented for text-to-phoneme mapping in the monolingual case. Decision trees could also be used in combination with N-grams and decision rules to improve the performance of the language identification module in the multilingual grapheme-to-phoneme conversion scenario.

In the text-to-phoneme mapping systems, proposed in this thesis, only binary decisions have been considered at the output of the neural networks. For instance, in the case of language identification module, the identified language of a certain word could only be

English or French. Soft decisions can be implemented at the output of the language identification module, to cope with words common to several languages.

In this thesis, the prosodic information has not been utilized in the proposed text-to-phoneme mapping modules. This can be included in order to further improve the phoneme accuracy.

Another issue, which was left beyond the scope of this thesis, was the selection of the training dictionary. In many practical applications only a limited number of words are needed such that selection of the lexicon could provide improved phoneme accuracy. Moreover, by proper selection of the entries into the training set, the redundancy, into the training dictionary, can be reduced which would help to improve the phoneme accuracy.

Bibliography

[1] U. Ackermann, B. Angelini, F. Brugnara, M. Federico, D. Giuliani, R. Gretter, G. Lazzari, and H. Niemann, "SpeeData: Multilingual Spoken Data Entry," in *Proceedings of the International Conference on Spoken Language Processing*, (Philadelphia, USA), pp. 2211–2214, Oct. 1996.

[2] M. Adamson and R. Damper, "A Recurrent Network that Learns to Pronounce English Text," in *Proceedings of the International Conference on Spoken Language Processing*, (Philadelphia, USA), pp. 1704–1707, Oct. 1996.

[3] O. Andersen, R. Kuhn, A. Lazarides, P. Dalsgaard, J. Haas, and E. Noth, "Comparison of Two Tree-Structured Approaches for Grapheme-to-Phoneme Conversion," in *Proceedings of the International Conference on Spoken Language Processing*, (Philadelphia, USA), pp. 1700–1703, 1996.

[4] F. Arciniegas and M. J. Embrechts, "Phoneme Recognition with Staged Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks*, (Como, Italy), pp. 259–264, 2000.

[5] F. Beaufays, "Transform-Domain Adaptive Filters: an Analytical Approach," *IEEE Trans. Signal Process*, vol. 43, pp. 422 – 431, 1995.

[6] U. Bhattacharya and S. K. Parui, "Self-Adaptive Learning Rates in Backpropagation Algorithm Improve its Function Approximation Performance," in *Proceedings of the International Conference on Neural Networks*, pp. 2784–2788, 1995.

[7] E. B. Bilcu and J. Astola, "A Hybrid Neural Network for Language Identification from Text," in *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, (Maynooth, Ireland), 2006.

[8] E. B. Bilcu and J. Astola, "Neural Networks with Random Letter Codes for Text-To-Phoneme Mapping and Small Training Dictionary," in *Proceedings of the 14th European Signal Processing Conference*, (Florence, Italy), 2006.

[9] E. B. Bilcu and J. Astola, "Improved Hybrid Approach for Bilingual Language Recognition from Text," in *Proceedings of the the Fifth IEEE International Symposium on Image and Signal Processing and Analysis*, (Istanbul, Turkey), 2007.

[10] E. B. Bilcu and J. Astola, "A Hybrid Approach to Bilingual Text-To-Phoneme Mapping," *Facta Electronics and Energetics*, vol. 21, pp. 91 – 105, April 2008.

[11] E. B. Bilcu, J. Astola, and J. Saarinen, "A Hybrid Neural Network Rule/Based System for Bilingual Text-To-Phoneme Mapping," in *Proceedings of the 14th IEEE International Workshop on Machine Learning for Signal Processing*, (Sao Luis, Brazil), 2004.

[12] E. B. Bilcu, J. Astola, and J. Saarinen, "Comparative Study of Letter Encoding for Text-To-Phoneme Mapping," in *Proceedings of the 13th European Signal Processing Conference*, (Antalya, Turkey), 2005.

[13] E. B. Bilcu, J. Suontausta, and J. Saarinen, "A New Transform Domain Neural Network for Text-To-Phoneme Mapping," in *Proceedings of the 6th WSEAS Multiconference on Circuits, Systems, Communications and Computers*, (Crete, Greece), pp. 97–100, 2002.

[14] E. B. Bilcu, J. Suontausta, and J. Saarinen, "Application of Neural Networks for Text-to-Phoneme Mapping," in *Proceedings of the XI European Signal and Image Processing Conference*, (Toulouse, France), pp. 97–100, Sept. 2002.

[15] E. B. Bilcu, J. Suontausta, and J. Saarinen, "A Study on Different Neural Network Architectures Applied to Text-to-Phoneme Mapping," in *Proceedings of the 3rd IEEE International Symposium on Image and Signal Processing and Analysis*, (Rome, Italy), pp. 4591–4596, Sept. 2003.

[16] E. B. Bilcu, J. Suontausta, and J. Saarinen, "Text-To-Phoneme Mapping Using a Fast Neural Network with Adaptive Learning Rate," in *Proceedings of the 7th WSEAS Multiconference on Circuits, Systems, Communications and Computers*, (Corfu, Greece), 2003.

[17] C. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

[18] A. W. Black and A. F. Llitjos, "Unit Selection Without A Phoneme Set," in *Proceedings of 2002 IEEE Workshop on Speech Synthesis*, (Santa Monica, USA), pp. 207 – 210, 2002.

[19] D. Braga, L. Coelho, R. Vianna, and G. Fernando, "A Rule-Based Grapheme-to-Phone Converter for TTS Systems in European Portuguese," in *Proceedings of the IEEE International Telecommunications Symposium*, (Fortaleza, Ceara, Brazil), pp. 328–333, 2006.

[20] W. B. Cavnar and J. M. Trenkle, "N-Gram-Based Text Categorization," in *Proceedings of the International Symposium on Document Analysis and Information Retrieval*, (Las Vegas, USA), pp. 161 – 174, 1994.

[21] J. A. Chambers, W. Sherliker, and D. P. Mandic, "A Normalized Gradient Algorithm for an Adaptive Recurrent Perceptron," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Istanbul, Turkey), pp. 396 – 399, May 2000.

[22] B. Y. Chen, M. W. Mao, and J. B. Kuo, "Coded Block Neural Network VLSI System Using an Adaptive Learning-Rate Technique to Train Chinese Character Patterns," *IEEE Electronics Letters*, vol. 28, pp. 1941 – 1942, 1992.

[23] M. Chu, H. Peng, Y. Zhao, Z. Niu, and E. Chang, "Microsoft Mulan - A Bilingual TTS System," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Hong Kong), pp. 264 – 267, Apr. 2003.

[24] B. V. Coile, "Inductive Learning of Pronunciation Rules With the Depes System," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Toronto, Canada), pp. 745 – 748, 1991.

[25] R. I. Damper, Y. Marchand, J. D. Marsters, and A. I. Bazin, "Aligning Text and Phonemes for Speech Technology Applications Using an EM-Like Algorithm," *International Journal of Speech Technology*, vol. 8, pp. 149 – 162, 2005.

[26] T. Dutoit, *An Introduction to Text-To-Speech Synthesis*. Dordrecht: Kluwer Academic Publishers, 1997.

[27] M. Embrechts and F. Arciniegas, "Neural Networks for Text-to-Speech Phoneme Recognition," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, (Nashville-Tennessee, USA), pp. 3582–3587, 2000.

[28] P. M. F. Content and M. Radeau, "BRULEX: Une Base de Donnees Lexicales Informatisee pour le Francais Ecrit et Parle.," pp. 551 – 566, 1990.

[29] J. Goldsmith, "Dealing With Prosody in a Text to Speech System," *International Journal of Speech Technologies (to appear)*, 2008.

[30] G. Grefenstette, "Comparing Two Language Identification Schemes," in *Proceedings of the 3rd International Conference on Statistical Analysis of Textual Data*, (Rome, Italy), pp. 1–6, Dec. 1995.

[31] P. R. Gubbins, K. M. Curtis, and J. D. Burniston, "A Hybrid Neural Network/Rule Based Architecture Used as a Text to Phoneme Transcriber," in *Proceedings of the IEEE International Symposium on Speech, Image Processing and Neural Networks*, (Hong Kong), pp. 113–116, Apr. 1994.

[32] J. Hakkinen and J. Tian, "N-gram and Decision Tree Based Language Identification for Written Words," in *Proceedings of the IEEE Workshop of Automatic Speech Recognition and Understanding*, (Trento, Italy), pp. 335–338, Dec. 2001.

[33] S. Haykin, *Neural Networks - A Comprehensive Foundation.* New York: Prentice-Hall, 1999.

[34] R. Hoffmann, O. Jokisch, D. Hirschfeld, G. Strecha, H. Kruschke, U. Kordon, and U. Koloska, "A Multilingual TTS System with Less than 1 Mbyte Foothprint for Embeded Applications," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (Hong Kong), pp. 532–535, Apr. 2003.

[35] H.-C. Hsin, C.-C. Li, M. Sun, and R. J. Sclabassi, "An Adaptive Training Algorithm for Back-Propagation Neural Networks," *IEEE Transactions on Syatems, Man., and Cybernetics*, vol. 25, pp. 512 – 514, 1995.

[36] Z. Huang and A. Kuh, "A Combined Self-Organizing Feature Map and Multilayer Perceptron for Isolated Word Recognition," *IEEE Transactions on Signal Processing*, vol. 40, pp. 2652 – 2657, 1992.

[37] I. Ipsic, N. Pavesic, F. Mihelic, and E. Noth, "Multilingual Spoken Dialog System," in *Proceedings of the IEEE International Symposium on Industrial Electronics*, (Bled, Slovenia), pp. 183–187, July 1999.

[38] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295 – 307, 1988.

[39] K. Jensen and S. Riis, "Self-Organizing Letter Code-Book for Text-to-Phoneme Neural Network Model," in *Proceedings of the International Conference on Spoken Language Processing*, (Beijing, China), Oct. 2000.

[40] S. Jiampojamarn, G. Kondrak, and T. Sherif, "Applying Many-to-Many Alignments and Hidden Markov Models to Letter-to-Phoneme Conversion," in *Proceedings of the Main Conference on Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*, (Rochester, New York, USA), pp. 372–379, 2007.

[41] O. Karaali, G. Corrigan, N. Massey, C. Miller, O. Schurr, and A. Mackie, "A High Quality Text-to-Speech System Composed of Multiple Neural Networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Seattle, USA), pp. 1237–1240, May 1998.

[42] S. Kolias and D. Anastassiou, "An Adaptive Least Squares Algorithm for the Efficient Training of Artificial Neural Networks," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 1092 – 1101, 1989.

[43] F. Korkmazskiy, "Statistical Learning of Language Pronounciation Structure," in *Proceedings of the IEEE workshop on Automatic Speech Recognition and Understanding*, (Trento, Italy), pp. 339–342, Dec. 2001.

[44] L. F. Lamel, M. A. Decker, J. L. Gauvain, and G. Adda, "Spoken Language Processing in a Multilingual Context," in *Proceedings of International Conference on Speech and Language Processing*, (Philadelphia, USA), pp. 2203–2206, Oct. 1996.

[45] K. Y. Lee, A. Sode-Yome, and J. D. Park, "Adaptive Hopfield Neural Networks for Economic Load Dispatch," *IEEE Transactions on Power Systems*, vol. 13, pp. 519 – 526, 1998.

[46] M.-S. Liang, R.-C. Yang, Y.-C. Chiang, D.-C. Lyu, and R.-Y. Lyu, "A Taiwanese Text-to-Speech System with Applications to Language Learning," in *Proceedings*

of the IEEE International Conference on Advance Learning Technologies, (Joensu, Finland), Aug. 2004.

[47] D. F. Marshall, W. K. Jenkins, and J. J. Murphy, "The Use of Orthogonal Transforms for Improving Performance of Adaptive Filters," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 474 – 484, 1989.

[48] S. Matsunaga, A. Ogawa, Y. Yamaguchi, and A. Imamura, "Non-Native English Speech Recognition Using Bilingual English Lexicon and Acoustic Models," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (Hong Kong), pp. 340–343, Apr. 2003.

[49] N. McCulloch, M. Bedworth, and J. Bridle, "NetSpeak a Re-implementation of NetTalk," in *Computer Speech and Language*, pp. 289–301, 1987.

[50] J. Meron and P. Veprek, "Compression of Exception Lexicons for Small Footprint Grapheme-to-Phoneme Conversion," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (Philadelphia, USA), pp. I293–I296, 2005.

[51] I. Nakanishi, Y. Itoh, and Y. Fukui, "Transform Domain Neural Filters," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 579–582, 1999.

[52] T. Olvecky, "N-Gram Based Statistics Aimed at Language Identification," in *Proceedings of the Informatics and Information Technologies Student Research Conference*, (Bratislava, Slovakia), pp. 1–7, Apr. 2005.

[53] V. Pagel, K. Lenzo, and A. Black, "Letter to Sound Rules for Accented Lexicon Compression," in *Proceedings of the International Conference on Spoken Language Processing*, (Sydney, Australia), pp. 2015–2018, 1998.

[54] I. T. Podolak and S.-W. Lee, "A Hybrid Neural System for Ponematic Transformation," in *Proceedings of the 15th International Conference on Pattern Recognition*, (Barcelona, Spain), pp. 44–47, Sept. 2000.

[55] J. M. Prager, "Linguini: Language Identification for Multilingual Documents," in *Proceedings of the 32nd Hawaii International Conference on System Sciences*, (Hawaii, USA), pp. 1–11, 1999.

[56] M. J. Radio, J. A. Reggia, and R. S. Berndt, "Learning Word Pronunciations Using a Recurrent Neural Network," in *Proceedings of the IEEE International Joint Conference on Neural Networks, 2001. Proceedings*, (Washington, DC, USA), pp. 11–15, 2001.

[57] A. J. Robinson, "An Application of Recurrent Nets to Phone Probability Estimation," *in IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 298–305, 1994.

[58] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation.* Cambridge, MA:Bradford Books/MIT Press, 1986.

[59] M. Schuster and K. Paliwal, "Bidirectional Recurrent Neural Networks," *in IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, Nov. 1997.

[60] T. Sef, "Slovenian Text-to-Speech System," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, (Geneva, Switzerland), pp. V41–V44, May 2000.

[61] T. Sejnowski and C. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *in Complex systems*, vol. 1, pp. 145–168, 1987.

[62] H. Shah-Hosseini and R. Safabakhsh, "TASOM: A New Time Adaptive Self-Organizing Map," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 33, pp. 271 – 282, 2003.

[63] D. C. Silva, A. A. de Lima, R. Maia, and D. Braga, "A Rule-Based Grapheme-to-Phone Converter and Stress Determination for Brazilian Portuguese Natural Language Processing," in *Proceedings of the IEEE International Telecommunications Symposium*, (Fortaleza, Ceara, Brazil), pp. 550–554, 2006.

[64] J. Suontausta and J. Hakkinen, "Decision Tree Based Text-To-Phoneme Mapping for Speech Recognition," in *Proceedings of the International Conference on Spoken Language Processing*, (Beijing, China), Oct. 2000.

[65] J. Suontausta and J. Tian, "Low Memory Decision Tree Method for Text-to-Phoneme Mapping," in *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, (St Thomas, Virgin Islands, USA), pp. 135–140, Dec. 2003.

[66] T. N. Tan, "Written Language Recognition Based on Texture Analysis," in *Proc. of the IEEE International Conference on Image Processing (ICIP'1996)*, pp. 185–188, 1996.

[67] P. Taylor, "Grapheme-to-Phoneme Conversion Using Hidden Markov Models," in *Proceedings of the INTERSPEECH*, 2005.

[68] J. Tian and J. Suontausta, "Scalable Neural Network Based Language Identification from Written Text," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (Hong Kong), pp. 48–51, Apr. 2003.

[69] O. Viikki, I. Kiss, and J. Tian, "Speaker- and Language-Independent Speech Recognition in Mobile Communication Systems," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, (Salt Lake City, USA), pp. 5–8, May 2001.

[70] Z. Wang, U. Topkara, T. Schultz, and A. Waibei, "Towards Universal Speech Recognition," in *Proceedings of the IEEE International Conference on Multimodal Interfaces*, (Pittsburgh, USA), pp. 247–252, Oct. 2002.

[71] P. J. Werbos, "Backpropagation Through Time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[72] R. J. Williams and J. Peng, "An Efficient Gradient-based Algorithm for on-line Training of Recurrent Network Trajectories," *in Neural computation*, vol. 2, pp. 490–501, 1990.

[73] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks," *in Neural Computation*, vol. 1, no. 2, pp. 270–280, 1998.

[74] E. Wong and S. Sridharan, "Three Approaches to Multilingual Phone Recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, (Hong Kong), pp. 44–47, Apr. 2003.

[75] www.speech.cs.edu, "The Carnegie Mellon University,"

[76] D.-M. Xiong and M. Yao, "A High Accuracy Approach for Word-Phoneme Translation Using Neural Networks," in *Proceedings of the IEEE International Conference on Neural Networks and Brain*, (Beijing, China), pp. 1029–1031, 2005.

[77] C.-C. Yu and B.-D. Liu, "A Backpropagation Algorithm with Adaptive Learning Rate and Momentum Coefficient," in *Proceedings of the International Joint Conference on Neural Networks*, pp. 1218–1223, 2002.

[78] J. Zibert, S. Martinich-Ipsic, I. Ipsic, and F. Mihelic, "Bilingual Speech Recognition of Slovenian and Croatian Weather Forecasts," in *Proceedings of the EURASIP Conference focused on Video/Image Processing and Multimedia Communications*, (Zagreb, Croatia), pp. 957–960, July 2000.