A Time-Delayed Neural Network Approach to the Prediction of the Hot Metal Temperature in a Blast Furnace

by

Mike Leonida

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 22, 2000

[June 2000]

Copyright 1999 M.I.T. All rights reserved.

The author hereby grants M.I.T. permission to reproduce and distribute publicly paper and electronic copies of this thesis and to grant others the right to do so.

Author_____

Department of Electrical Engineering and Computer Science
May 22, 2000

Certified by_____

Dr. Amar Gupta
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

ENG

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUL 27 2000

LIBRARIES

A Time-Delayed Neural Network Approach to the Prediction of the Hot Metal

Temperature in a Blast Furnace

by
Mike Leonida

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 2000

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The research in this document is motivated by a problem which arises in the steel industry. The problem consists of predicting the temperature of a steel furnace based on the values of several inputs taken one through seven hours in advance (seven different sets of data). Two different time-delayed neural network (TDNN) implementations were used. The data was provided by a large steel plant located outside the United States. This work extends analysis already done by the group on this data using a multi-layer perceptron (MLP).

This paper examines the architectures used in detail and then presents the results obtained. A survey of the data mining field related to TDNNs is also included. This survey consists of the theoretical background necessary to understand this kind of neural network, as well as recent progress and innovations involving TDNNs. Issues involved with running computationally intensive neural networks and the optimizations that have led to progress in this domain are also discussed.

ACKNOWLEDGEMENTS

I would like to thank Dr. Amar Gupta, Brian Purville, Ravindra Sarma and all the members of the data mining group for their support. Thanks also go to Dr. Eric Wan from the University of Oregon who wrote some of the Matlab software used.

Many thanks to Prof. Paul Gray whose continued advice during the past five years has helped me out immensely.

Most of all, I would like to thank my family for always being there for me.

ABSTRACT

The research in this document is motivated by a problem which arises in the steel industry. The problem consists of predicting the temperature of a steel furnace based on the values of several inputs taken one through seven hours in advance (seven different sets of data). Two different time-delayed neural network (TDNN) implementations were used. The data was provided by a large steel plant located outside the United States. This work extends analysis already done by the group on this data using a multi-layer perceptron (MLP).

This paper examines the architectures used in detail and then presents the results obtained. A survey of the data mining field related to TDNNs is also included. This survey consists of the theoretical background necessary to understand this kind of neural network, as well as recent progress and innovations involving TDNNs. Issues involved with running computationally intensive neural networks and the optimizations that have led to progress in this domain are also discussed.

TABLE OF CONTENTS

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

1 INTRODUCTION

2 PROBLEM SPECIFICS AND DATA COLLECTION

2.1 History

2.2 Data collection

3 ONGOING RESEARCH IN THE DATA MINING GROUP ON THIS PROBLEM

3.1 Genetic Algorithms

3.2 Decision Trees

3.3 Inversion Problem

4 ARCHITECTURES USED: TDNN, MLP with FIR

4.1 Neural Networks and Data Mining

4.2 Neural Network Architecture

4.3 Multi-Layer Perceptron (MLP) Architecture

4.3.1 Feed-forward Stage

4.3.2 Backpropagation Stage

4.4 Time-Delayed Neural Network (TDNN) Architecture

4.4.1 Historical Development

4.4.2 Architecture

4.4.3 TDNN <=> MLP with FIR

4.5 Backpropagation Methods

4.5.1 Vanilla/Online Backpropagation

4.5.2 Enhanced Backpropagation

4.5.3 Batch Backpropagation

4.5.4 Weight Decay Backpropagation

4.5.5 Temporal Backpropagation

5 TDNN THEORY AND COMPUTATION

5.1 TDNN Theory

5.1.1 General Neural Network theory

5.1.1.1 Single Perceptron

5.1.1.2 Multiple-node networks

5.1.2 TDNN Theory

5.2 TDNN Applications Examples

5.2.1 Speech Processing

5.2.2 Stock Market

5.2.3 Student Admissions

5.2.4 Other Applications

5.3 Modified TDNN Architectures

5.3.1 Two-level TDNNs (TLTDNNs)

5.3.2 Adaptive Time-Delay Neural Networks (ATNN)

5.3.3 Memory Neural Networks (MNN)

5.3.4 Multi-State TDNNs

5.3.5 Cascaded TDNNs

5.3.6 Hybrid systems using TDNNs

5.4 TDNN Optimizations

5.4.1 Input Saliency

5.4.2 Optimized Running Time

5.4.3 Gaussian filters

5.4.4 Alternative Backpropagation Methods

5.4.5 Parameter Sensitivity Alleviation

5.4.6 One-shot learning during testing

6 RESULTS

7 CONCLUSIONS

8 REFERENCES

Appendix A - TDNN Inputs and Outputs

Appendix B - Trained TDNN from SNNS

Appendix C - Results

1 INTRODUCTION

This paper's primary goal is to derive an effective neural network architecture to use for predicting the hot metal temperature in a steel furnace. Neural network architectures are developed and tested through a training/validation process. This process uses data collected in real-world environments, the same environments where the neural network developed is used.

The data necessary for running the models was provided by a foreign customer named SteelCorp. Predicting and controlling the manufactured steel temperature is extremely important because it directly affects the quality of the plant's product. The neural network in this paper has the manufactured steel hot metal temperature (HMT) as an output. Effectively, this means the neural network attempts to predict the hot metal temperature.

The datamining group at MIT (under the supervision of Dr. Amar Gupta) has previously conducted research on this data. The results were obtained with a multi-layer perceptron (MLP) architecture.

Due to the nature of the problem, it was believed that a time-delayed neural network (TDNN) would perform better at predicting the hot metal temperature than the standard MLP. A priori, a conjecture that the data would exhibit temporal locality was made. The basis for this conjecture was that the input/output pairs were collected during a continuous period of time at set intervals. Continuous time period data often contains

data patterns which can be modeled efficiently with TDNNs because of their temporal nature.

TDNNs are extended MLPs which store a fixed (constant) number of delayed values from the network. (These delayed values are generally used to better predict patterns with a temporal component.) TDNNs retain the feed-forward and backpropagation stages which characterize a multi-layer perceptron. During both these stages, the delayed values stored in the network are used in calculating the updated values/weights in the networks.

The inputs to the time-delayed neural network (TDNN) are 18 parameters thought to affect the future hot metal temperature. A complete list of these parameters can be found in Appendix A. The observed relative importance of inputs to the output will be discussed in the results section. This section will also include a methodology part, which examines the range of approaches tried and why some were more successful than others.

In order to understand the overall impact of TDNN technology in the data mining field, this paper includes a comprehensive survey of the field. This survey includes a neural networks theoretical computation background section and an in-depth look at the TDNN and its architecture. The survey concludes with a detailed look at the latest technological advances involving TDNN_s, from internal optimizations such as recurrent backpropagation techniques to speech processing hybrids such as HMM/TDNN hybrids.

Section 2 will give a few more details regarding the collection of the data. Other ongoing research within the datamining group regarding this problem will follow, along with a

general background for the techniques used in these areas. Section 4 presents a detailed examination of both the MLP and TDNN architectures.

The results will be detailed in Section 6, after an extensive theoretical background section on recent TDNN developments in Section 5. Section 7 will present the conclusions, as well as future anticipated work in this domain. (For a more detailed overview, please refer to the outline on page 3.)

2 PROBLEM SPECIFICS AND DATA COLLECTION

This section will first present a history of the problem, in order to provide the motivation for the research. Data collection and make-up will be discussed in the second part.

2.1 History

Iron working has been an occupation mastered and perfected by humans generation after generation since the very beginning of time. Some of the earliest known traces of iron work date as far back as 1700 B.C. [54]. According to Dr. Gupta, the leader of the data mining group at MIT, a massive iron pillar exists in Delhi, India which may date even farther back. This pillar is not at all rusted today and is believed to be 4000 years old. For a very long period of time, the production of iron was extremely important to producing better weapons, and thus extremely important to a nation_s ability to fight a war [2]. It is only natural then that the process of manufacturing this important material has been the object of relentless scrutiny and improvement for millenia.

During the Middle Ages, this process consisted of heating alternating layers of ore and wood to obtain hot ore. This hot ore was then cleansed of its impurities by hammering it to obtain the raw iron. The raw iron was then molded in a forge and prepared in a hearth [2].

The passage of time also brought improvements to the quality of the product. Steel (or iron with carbon added) started to be manufactured in small quantities as its hardness was better than that of iron.

Iron and steel manufacturing today have become extremely complex from a technological standpoint. Their production involves a multitude of mechanical, thermodynamic and chemical reactions which are very hard to model mathematically in an efficient manner. In the problem described by this paper, the generation of pig iron (the output of the production process) involves melting down ore and coke. Several chemical reductions occur next, with the final product being pig iron. The pig iron produced here is later molded and cooled down to produce steel.

Some of the impurities present in the ores are separated off during this process forming a substance called slag [54, 55]. For a more detailed description of this process also see [1, 58].

There are three parameters which are normally used to judge the quality of the ore produced by the furnace. They are: silicon content of the sample, hot metal temperature and the condition of the slag.

Silicon is an impurity introduced both through the coke ash and the iron ore, and should be kept at a minimum. Thus, the lower the silicon concentration, the higher the quality of the pig iron produced.

The condition of the slag is also a good indicator of how good the resulting pig iron will be. This occurs because the slag is separated from the pig iron relatively late in the process.

The third and final parameter used to judge the quality of the furnace output is its temperature. This varies mostly between 1450 and 1500 degrees Celsius [57].

Fortunately, all these variables have been found to have high correlation with one another. Being able to predict one of them allows one to make a reasonable estimate for the actual quality of the pig iron produced without knowing the other two.

In this work the hot metal temperature (HMT) has been chosen as output because it is the simplest of the three variables to measure. The primary motivation in this project is to help the steel plant increase its productivity by allowing it to more accurately predict the quality of the pig iron produced. A reasonably accurate prediction of the HMT as proposed would serve this purpose adequately, as shown in this section.

2.2 Data collection

The steel data used for the experiments was collected at "SteelCorp", a pseudoname for a major steel company located outside the United States. Each input/output pair is composed of 16 inputs and one output. For a detailed list of the inputs, please refer to Appendix A. Note that the current HMT in the furnace is used as an input. For reference, much better results were obtained last year when the current hot metal temperature was used as an input [2]. The only problem with using this as an input is that the HMT at the current time may closely trail the predicted HMT because of the very temporal locality that the TDNN architecture exploits. As explained in the Results and Conclusions sections, however, this correlation is desired, useful and available for the calculation of the output HMT.

However, the analysis was done with extreme caution so as to keep the HMT values used for training separate from the HMT values used for testing. This was ensured by taking the validation set to be the last (approximately) 20% of the values in the data supplied.

There were seven data sets collected. In the first, the output HMT was taken exactly one hour after the measurement of the 16 other parameters. In the second, the output HMT was taken exactly two hours after the initial measurements. This pattern was maintained up until the 7th set, where the output HMT was taken exactly seven hours afer the recording of the 16 inputs.

3 ONGOING RESEARCH IN THE DATA MINING GROUP ON THIS PROBLEM

This section will present an overview of the other ongoing approaches to solving this problem. These approaches include genetic algorithms, decision trees and the inversion problem. A brief synopsis of these data mining methods and their relevance to the task at hand is included [53, 2].

3.1 Genetic Algorithms

Genetic Algorithms (GAs) are data mining techniques which apply principles from biology such as genetics and natural selection to searching for an optimal set of classifier parameters [58]. These are often used as aides to training neural networks.

Genetic algorithms, like many other data mining tools, find only local optima without guaranteeing global optimality. Although the use of mutation is often employed to counterbalance this drawback, its observed efficiency has been limited.

The data mining group will attempt to use the principles of genetic algorithms in order to accomplish better hot metal temperature (HMT) predictions in the future.

3.2 Decision Trees

The concept of decision trees attempts to combine ideas from artificial intelligence and statistics. The most prevalent algorithms are CHAID, CART and C4.5 [58].

The advantage of decision trees is that they produce understandable rules independent of the complexity of the input. Although not very well suited to temporal data analysis or prediction, decision trees will also be used in analyzing the problem of hot metal temperature prediction. The main effort in doing so will probably consist of preprocessing of the data so that temporal trends are easily seen. This can most easily be accomplished through TDNN preprocessing of the data.

3.3 Inversion Problem

A relatively new data mining approach, the inversion problem involves analysis similar to neural networks. Although this analysis will be a lot more computationally intensive than any neural networks approach, it is anticipated that it will yield great results when perfected.

In addition, this method answers a question that the steel company wants answered. This question is: what can be done to raise the hot metal temperature for optimal steel production?

The inversion problem involves using derivative analysis and complex input sensitivity parameters in order to model the direct input/output causality interaction. For example, the desired output is taken as given and becomes the input to the network. The inversion problem architecture must now answer what modifications in the inputs to the furnace will yield this desired temperature.

The inversion problem derives its name because in essence it attempts to invert the causality of the input/output relationship in a real life application such as modeling the hot metal temperature in a blast furnace.

4 ARCHITECTURES USED: TDNN, MLP with FIR

This section includes a detailed look at the architectures employed in this research. First, neural networks are examined from a data mining tool perspective and compared to some of the methods seen briefly in the section before. The section will then examine the architecture of neural nets in general and present a few interesting theoretical results related to training.

The third subsection will take a look at the two stages for a multi-layer perceptron (MLP): feed-forward and backpropagation. The fourth subsection will examine the time-delayed neural network (TDNN) architecture closely and the context of its historical development. This section will also include an explanation for why TDNN_s are equivalent to MLP_s with finite impulse response filters. The fifth and final subsection will present the choices available for weight updating during the backpropagation stage, along with the choices in this matter.

4.1 Neural Networks and Data Mining

Neural networks are the most common data mining technique, as well as the oldest. They are very flexible instruments which can handle arbitrarily complex problems, so long as they are adaptable to the training/testing model [58]. Despite the obvious advantages of neural networks, they also have a few disadvantages. The two most notable and common disadvantages are examined here:

- no explicit representation of how the solution was obtained, and often no connection to real life phoenomena/parameters. Although one can analyze the derived post-training weights in a neural network and attempt to find a pattern for how the net accomplishes the given task, this process is often highly speculative and rewardless.
- neural networks most often do not and cannot guarantee an optimal solution. This is a result of the fact that the various procedures employed for training the neural net perform functions like gradient descent, which can sometimes settle on local minima for the cost function. It is very hard to prove that these solutions are optimal, although Hessian matrix techniques have been employed in this domain.

Despite these disadvantages, neural networks have possessed and maintained a high usage rate as data mining techniques over time. The main two reasons for this have been their consistent results and the unique ability to cope with complexity which has yet to be understood through any other methods.

4.2 Neural Network Architecture

Neural networks are made up of neurons, which get their name because of similarities they have with the cells that make up the human nervous system.

Human neurons are made up of a cell body and an axon, both of which have dendrites on the end. These neurons communicate with one another axon-to-cell-body through dendrite connections called synapses. Human neurons are all-or-none devices, meaning they either fire or do nothing.

Human neurons fire based on the influence (positive or negative) coming in at the cell body dendrites. If the sum of these influences exceeds a certain threshold (which characterizes the neuron), then the neuron fires, or in translation, its axon exerts its influence on the next connected neuron.

Neural networks neurons are similar to real life neurons in that they contain incoming synapses whose "influence" (or value) adds up. The simulated neuron may or may not be all-or-nothing. It "fires" (e.g. outputs a value) based on its activation function. If this function is a one-step function the neuron becomes all-or-nothing. It is most common, however, for the neuron to fire according to a sigmoid function that approximates a step function, mainly because output values of 0 are sometimes undesirable.

Collections of such simulated neurons organized in layers are called neural networks. Normally, the first (input) layer influences the second, which in turn influences the third, and so on until the output layer. In more complicated neural networks this hierarchy can be modified to include both backward links and links that skip over one or more layers.

The usefulness of neural networks lies in their ability to learn patterns and characteristics before they are fully understood by even humans. Thus, the two steps in using a neural network are:

1) learning: the network is presented a set of valid input/output pairs over and over. This continues until the neural net is able to develop an internal representation which includes the "learned" characteristics of the training set.

2) testing/use: the existing network is now used to predict the output from new inputs which did not belong to the training set.

Sometimes very intricate and difficult to analyze, the biggest strength of neural networks lies in their ability to effectively model complex non-linear decision surfaces. Neural networks are based on a fundamentally simple idea. The neural network paradigm is easy to express: all that is needed for neural network operation are the training input/output pairs. The complexity that makes neural networks effective lies in their highly adaptive internal structure.

Neural networks are very flexible compared to other data mining tools. Not much data processing has to go into deploying a neural networks approach to most problems. This is the reason why neural nets have remained the oldest and most commonly used data mining tool, along with the neural net's ability to deal with complexity [58].

4.3 Multi-Layer Perceptron (MLP) Architecture

A multi-layer perceptron (MLP) is a neural network which is characterized by two passes during training: a feed-forward stage and a backpropagation stage.

4.3.1 Feed-forward Stage

The feed-forward stage in a neural network is the stage where the inputs are propagated using the weights/state already existent in the network. Note that in order to get the network up and running, it must be initialized with a set of weights/state. These weights are normally chosen to be random numbers that are small in magnitude.

Weights are a measure of how much one neuron affects another along a particular connection.

The output from the first neuron's activation function is multiplied by the weight along the connection and is added to the other values affecting the second neuron. This sum is computed and fed into the activation function, which determines the output value of the second neuron.

For reference see Figure 1 below.

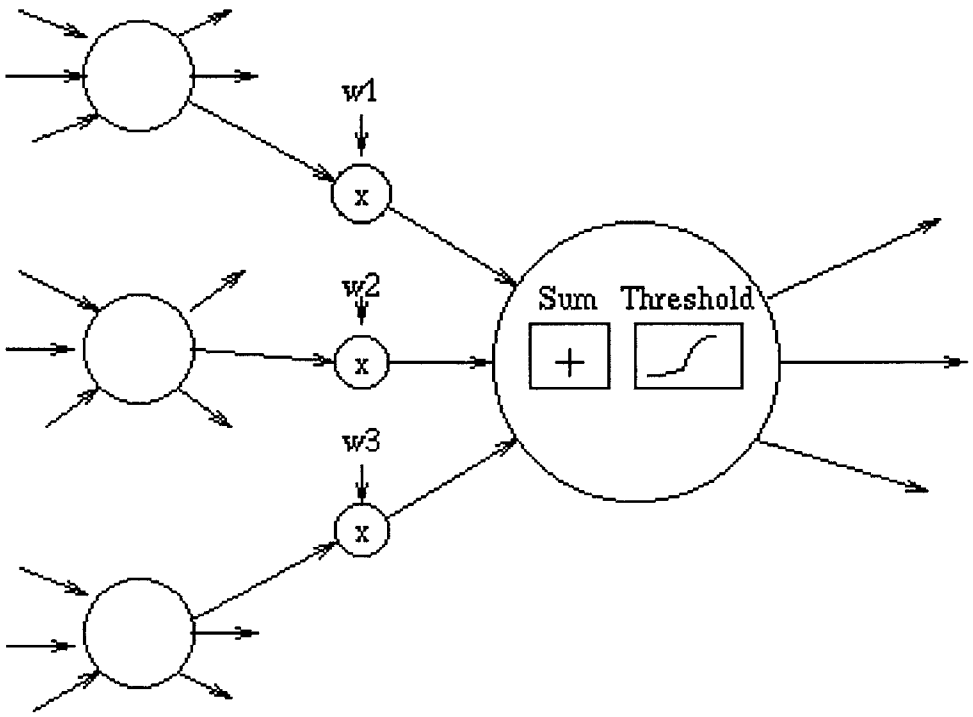


Fig. 1: Neuron level view

Inputs from other neurons are multiplied by weights, then the sum is passed through a threshold filter.

4.3.2 Backpropagation Stage

The backpropagation stage involves taking the output(s) values and traversing the network in the opposite direction of the arrows in Figure 1.

In the process of doing so, the weights are updated proportional to the improvement they could make in the evaluation of the given output (or set of outputs). The weights are also updated proportional to a learning parameter which is preset. Depending on the type of application the neural network is designed for, the learning parameter is usually chosen between .02 and .4. Increasing the learning parameter, of course, presents a tradeoff between quicker learning (a positive) and posible overshooting in the adjustment (the drawback).

As seen in [8], the (derived) formulae to be used during backpropagation for weight updates are the following:

(r is the learning parameter, $w_{i->j}$ is the weight from neuron i to neuron j, d_z = desired output at z, o_z = actual output at z)

$$\Delta w_{i->j} = r\, o_i\, o_j\, (1 - o_j)\, \beta_j$$

(Note: the reason for the $o_j(1-o_j)$ term is the use of the sigmoid activation function. Its derivative is just $o_j(1-o_j)$)

Here

$$\beta_j = \Sigma_k\, w_{j->k} o_k (1 - o_k)\, \beta_k,$$

for the nodes in the hidden (middle) layers and

$$\beta_j = d_z - o_z,$$

for nodes in the output layer.

For a detailed derivation of these formulae please see [8], pgs. 451-458.

It is thus possible to update the weights in a feed-forward network using these backpropagation formulas. In order to be consistent with the rules of gradient ascent/descent, these calculated deltas/changes would have to be summed and taken into

account at the same time for all training patterns. Had this been done individually, the order of the training patterns may cause the learning to deviate from the standard gradient ascent/descent which are being followed. However, there are several different flavors of weight updating methods, and the most important will be examined in section 4.5.

4.4 Time-Delayed Neural Network (TDNN) Architecture

Time-delayed neural networks are in essence MLPs augmented to contain past values in the input and hidden layer(s). The number of extra such values kept around in the network is called the number of delays of the respective layer.

What is the motivation for using such a structure to augment the more traditional multi-layer perceptron? TDNNs were first used primarily in speech recognition, where the dynamic nature of signal forced the neural network to account for the temporal evolution of the amplitude.

Additionally, TDNNs are more adept at representing temporal relationships between events. This ensures that longer and/or shorter pauses in speech would not interfere with the quality of the neural network. In the case of the MLP, the neural network would try to account for these slight variations in the pauses by making the recognized signal more flexible. This would naturally result in a degradation of the

performance for the neural net. TDNNs' main advantage over MLPs is that they offer relatively good recognition of slight temporal misalignment in patterns. This means that the time delays built within the TDNN allow the neural network to better learn the slight variations in, for example, the pauses between various logical subunits of the input. A good example of such variations that would need to be recognized consists of the pauses between words in speech. These pauses differ from person to person.

4.4.1 Historical Development

TDNNs became a popular between 1986 and 1991, when they were used for several speech processing application. McClelland and Elman, Elman and Zipser, Tank and Hopfield and Waibel et. all are just a few of the first major contributors to this field [10, 11, 12, 4].

The most widely recognized and cited contribution to this field is Waibel et. all's "Phoneme Recognition Using Time-Delay Neural Networks" (1989) [4]. This paper presents both a strong argument in favor of using TDNN_s for speech processing, as well as a clear and detailed description of TDNN architecture.

With time, TDNNs have become increasingly popular. Their widespread use today ranges from the more traditional speech processing application to the newer image processing, tool wear prediction, inventory management and access pattern prediction applications.

4.4.2 Architecture

Waibel et. all [4] outline the five criteria that a TDNN must meet in order to be effective for speech processing, as well as any other application which requires spatio-temporal alignment.

1) The neural network must be complex enough to have the ability to learn the problem at hand. Thus, enough layers and enough connections between these layers of the neural net are needed.

2) At some level, the network must be able to represent temporal relationships.

3) Translation in time should not be able to affect the patterns learned by the network. This criterion is a result of the variation in pauses between the observed features.

4) Training the neural network must not be dependent on the precise temporal alignment of the features being extracted.

5) The training set must be large enough compared to the number of network nodes, so the network is forced to learn the "correct" representation of the patterns studied. Indeed, if the training set were too small compared to the internal representation, it may be the case that several network configurations could encode the training set equally well, or even better, than the optimal/desired configuration. The odds of obtaining this optimal configuration would be therefore greatly decreased, and the results would be at the mercy of the initial conditions fairy.

As shown in Figure 2, the basic TDNN unit is somewhat alike the basic MLP unit in that it computes the weighted sum of its inputs, U1 through Un. In addition, the TDNN

factors in the delayed values it stores in its nodes. These values are represented by the register outputs from the delay blocks D1 through Dm. These delayed values are also multiplied by their respective (but probably smaller) learned weights and added to determine the input to block F, the activation function.

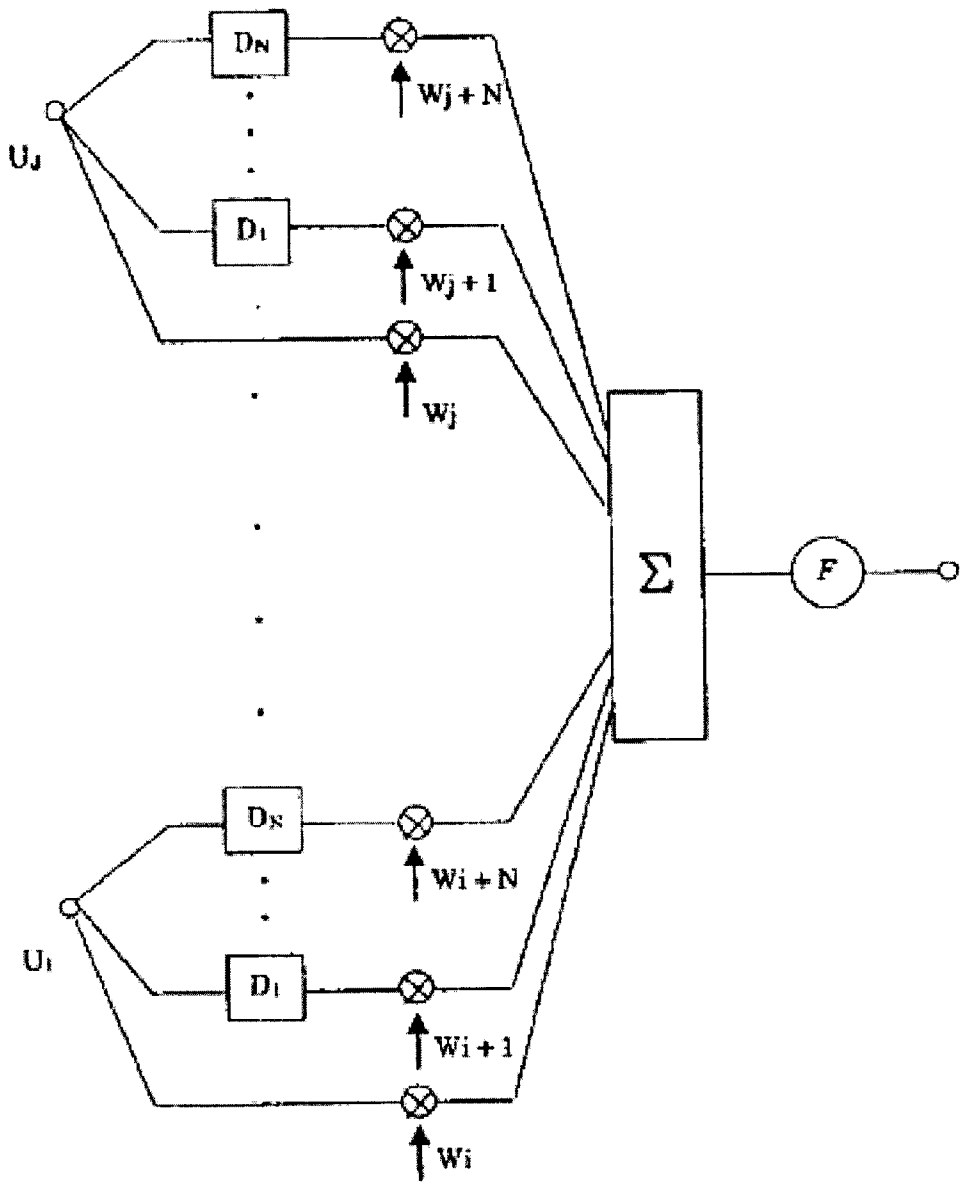


Fig. 1. A Time-Delay Neural Network (TDNN) unit.

While Figure 2 presented the layer-to-layer connectionist view of a TDNN, this section will now shift focus to the multi-layered view of TDNNs. As seen in Figure 3, the input layer is an i x d_i grid of units, where i is the original number of inputs and d_i is the number of delays employed by the input layer.

There are two hidden layers in this example, both with h_j x $(h_{j-1}-d_{hj}+1)$ units. Similarly, h_j is the number of units specified for the j-th hidden layer and d_{hj} the number of corresponding delays for hidden layer j.

Finally, the output layer in the example has only one unit. The output layer may generally have several units, but must always have a delay of just 1. This is because the delays are only used for performance, not prediction purposes.

For an example of a TDNN which has more than one output, see [4]. Here for efficiency there were three output units used, one for each of the three phonemes examined. It is generally a good strategy to split the output into several units in order to have a higher margin of error during testing. Indeed, for a three-output neural net, the margin of error is approximately 2^2 = 4 times greater than that of a one-input net because there are two extra units that either fire or do not fire.

A single output (un-initialized) TDNN is shown below in Figure 3. This network obtained very good results in the simulation. Note that like all TDNNs, it is fully connected between layers.

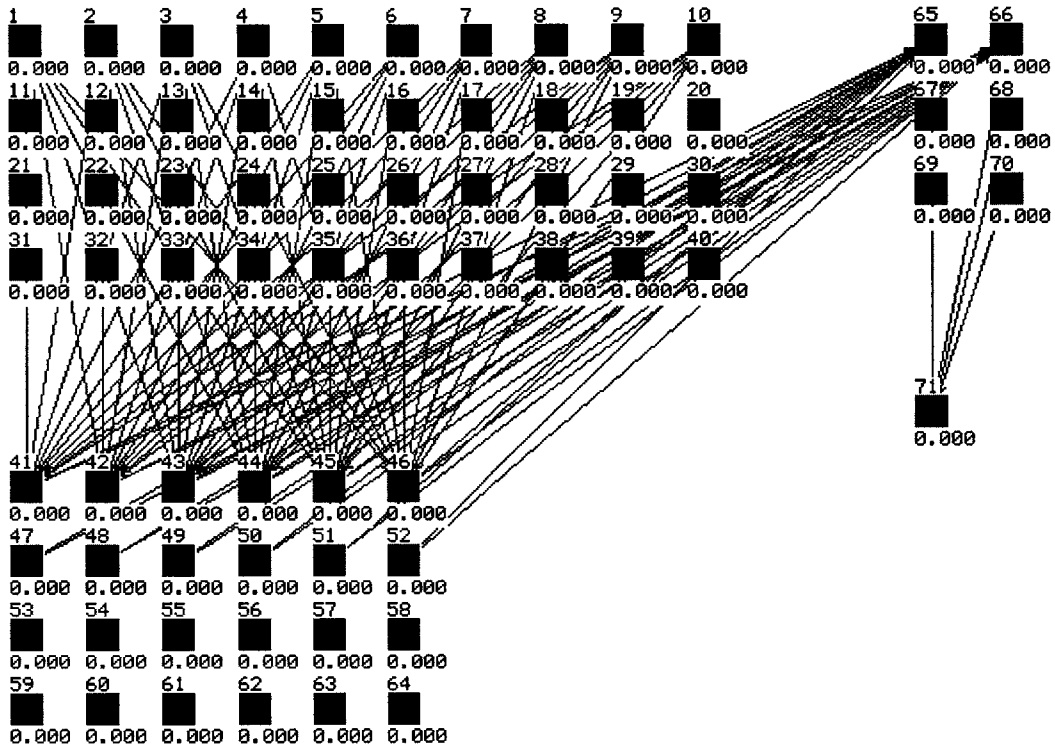


Figure 3: A TDNN architecture with 16 (x 4 delays) input units, two hidden layers with 6 (delay 4) and 2 (delay 3) nodes, and a single output. This network was among the best performing on the data.

4.4.3 TDNN <=> MLP with FIR

While the term time-delayed neural network has been widely adopted in the field, there is an alternate architecture which is very similar to a TDNN. This architecture is an MLP with finite impulse response (FIR) filters. (see Figure 4)

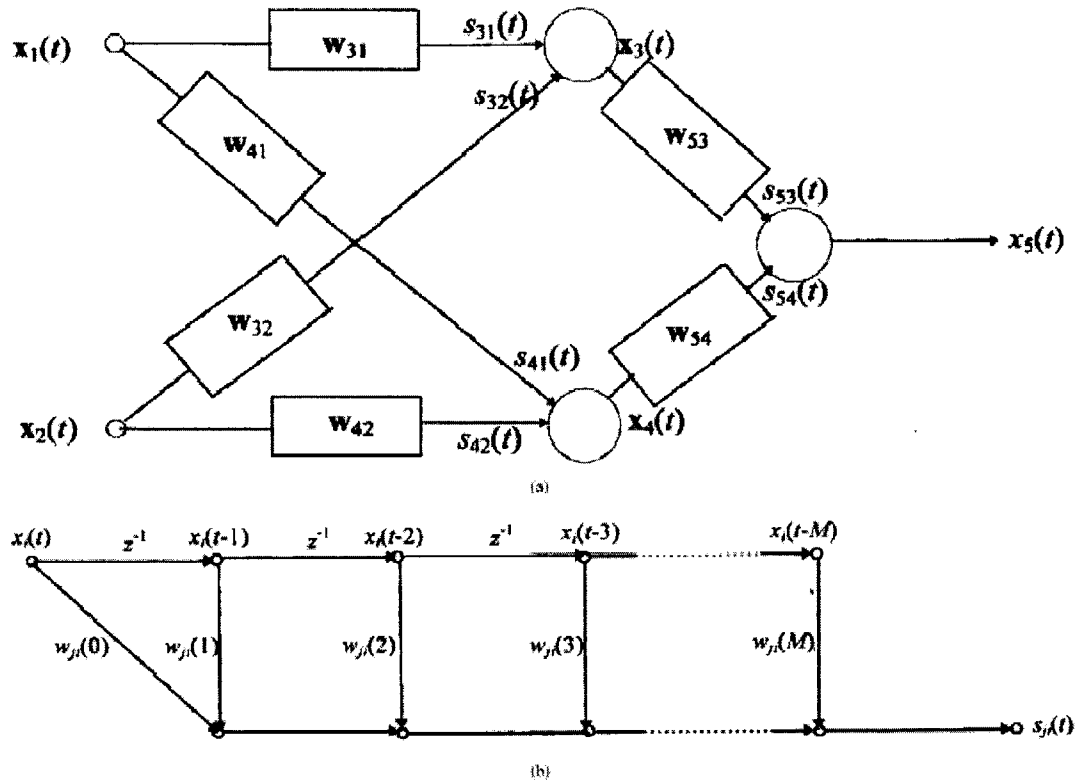


Fig. 1. (a) Three-neuron TDNN with FIR filters (w_{ji}) as synaptic connections. (b) Expanded view of FIR synaptic connections of TDNN. FIR filters build internal memory into the network.

In place of the normal synaptic weights, the MLP with FIR has weight vectors wij, where i represents the index of the from-node in the network and j represents the index of the to-node:

$$w_{ij} = [w_{ij}(0), w_{ij}(1), \ldots w_{ij}(d)].$$

Here d is the number of delays used by the respective network layer.

The neuron output $n_i(t)$ is also a vector of size d+1:

$$x_i(t) = [x_i(t), x_i(t-1), \ldots x_i(t-d)]$$

The ouput s_{ij} of the weight filter w_{ij} is thus the inner product of w_{ij} and $x_i(t)$:

$$s_{ij}(t) = w^T_{ij} * x_i(t)$$

The activation potential is then computed as the sum of the FIR weight outputs:

$$\text{active potential} = a_{pi}(t) = \Sigma^{d}_{i=0}\, s_{ij}(t) \quad [13]$$

This architecture is now compared with the one presented in section 4.4.2. Both architectures keep track of a number of extra states at each level. In both architectures, each extra state has its own individual weight. In both cases the sum of the products becomes the activation potential, or the input to the activation function. The rest of the parameters are all identical.

It has been shown that the difference between a TDNN and an FIR MLP is purely notational. The TDNN expands its links out for a more explicit description, while the FIR MLP concentrates its delayed values and weights in a concise vector/matrix form. The two models are therefore equivalent and the rest of the paper will refer to them interchangeably.

4.5 Backpropagation Methods

This section examines some of the more common backpropagation methods used in neural networks. These methods describe the policy used to update the weights in a neural network. Their description has been obtained mostly from [14].

4.5.1 Vanilla/Online Backpropagation

This is the standard backpropagation learning algorithm. Its definition is similar to the one presented in section 4.3.2. It consists of updating weights after each pattern (hence the name online backpropagation)

Online backpropagation equations: [14]

$$\Delta w_{i->j} = r\ \delta_j\ o_i$$

where:

$$\delta_j = f'_j(net_j)\ (d_j - o_j)$$

for the nodes in the output layer and

$$\delta_j = f'_j(net_j)\ \Sigma_k\ \delta_k w_{jk}$$

for the nodes in the hidden layers.

4.5.2 Enhanced Backpropagation

This method extends the online backpropagation algorithm to contain a momentum parameter. The result of using a momentum parameter is that "flatter" regions of the non-linear surface are traversed more rapidly. As a result, the training time is improved [14].

Here is the only changed formula for enhanced backpropagation:

$$\Delta w_{i->j}(t+1) = r\ \delta_j\ o_i + \alpha\ \Delta w_{ij}(t)$$

4.5.3 Batch Backpropagation

This backpropagation method is almost identical to the online/vanilla method. The only difference lies in the timing of the weight updates - they are all done after examination of all the training patterns. While this method saves on communication costs, it sometimes proves ineffective because it may not "learn" quite quickly enough.

Here are the batch backpropagation formulas: [14]

$$\Delta w_{i\text{->}j} = r\, \delta_j\, o_i$$

where:

$$\delta_j = f'_j(net_j)\,(d_j - o_j)$$

for the nodes in the output layer and

$$\delta_j = f'_j(net_j)\, \Sigma_k\, \delta_k w_{jk}$$

for the nodes in the hidden layers.

4.5.4 Weight Decay Backpropagation

This method also augments the online backpropagation method. Its only modification is the subtraction of a weight decay term during the weight change update.

The weight decay term ensures that continuously unupdated weights are driven to 0 over time because of their relative lack of importance. This method can be shown to be

a more "lenient" version of the all-or-nothing input sensitivity analysis presented in section 6.

Here are the modified weight decay backpropagation formulas: [14]

$$\Delta w_{i->j}(t+1) = r\, \delta_j\, o_i - d\, w_{ij}(t),$$

where d is the decay rate used.

4.5.5 Temporal Backpropagation

This backpropagation method is the backpropagation method which is actually used for TDNNs. It necessarily must update all the weights in the TDNN. The function this updating method performs is (approximately) equivalent to the gradient descent/ascent performed by online backpropagation for the regular multi-layer perceptron.

Along with the additional nodes and weights in a TDNN, this backpropagation method constitutes the difference between a regular MLP and an FIR MLP. For efficiency, both the non-causal and causal formulas for temporal backpropagation will not be presented in this section.

Rather, they will be reproduced, with a summary derivation and explanation of the need to approximate gradient descent with respect to the error/cost function, in Section 6.1.2 (For reference purposes, these formulas were taken from [5, pp 651-660], where they are derived for distributed Time Lagged Feedforward Networks (TLFNs). Distributed TLFNs are in actuality MLPs with FIR filters, or simply TDNNs)

5 TDNN THEORY AND COMPUTATION

This section will present a more in-depth look at the data mining instrument used for the analysis, the time-delayed neural network (TDNN). It will start with a theoretical view, which will present some general results about both neural networks in general and time-delayed neural networks in particular.

The second subsection will take a look at the applications that TDNNs have generally been used for. Next, the focus will shift to a few optimizations that have improved or will have an impact on TDNN computation. Modified TDNN architectures have also evloved over time. These, together with their related applications will be presented in the fourth and final subsection.

5.1 TDNN Theory

Since training is the key to neural network success, this section is dedicated to exploring the training of neural networks in general and TDNNs in particular. A careful description of trainability will be attempted. During this process the focus will be mainly on the following two questions:

1) Is it possible to learn training sets perfectly?

2) If not, how well can training sets that cannot be learned perfectly be approximated?

5.1.1 General Neural Network theory

5.1.1.1 Single Perceptron

Neural networks are complicated tools that attempt to solve very difficult problems, with multiple and sometimes complex constraints. The section will start with very simple examples of neural network problems and then generalize the results to cover the examples that are relevant to this paper. Thus, the first neural network classification problem will consist of having a single perceptron attempt to learn a binary valued output o.

In order to examine the ability of a single perceptron to learn a training set effectively, the concepts of half-spaces and linear separability must be introduced first. Half-spaces are useful geometrical abstractions in n-dimensional geometry. They are partitions of n-dimensional space according to an(n-1)-dimensional boundary. This (n-1)-dimensional boundary is sometimes called a hyperplane [15].

For example, in two-dimensional space, half-spaces are a result of partitioning the plane with a one-dimensional line. (See figure 5 below) In three-dimensional space, half-spaces are determined by space being partitioned by a two-dimensional plane, and so on. The resulting two half-spaces are denoted H^+ and H^-, based on whether their exceed the threshold defined by the hyperplane or not.

Linear separability of two partitions of the set S, namely S^+ and S^-, is now defined to be whether or not there exist complementary half-spaces H^+ and H^- such that:

$$S^+ \subset H^+ \text{ and } S^- \subset H^-.$$

Additionally, strict linear separability requires that no points in S^+ and S^- lie on the separating hyperplane H.

The first question posed in the beginning of this section for a single perceptron is now ready to be answered. Since a single perceptron compares a function of the inputs to a threshold, it essentially divides the input space into two half-planes. In order for it to perfectly learn a training set, the perceptron must be able to discern between all inputs with just one hyperplane. This is equivalent to the partition of the inputs having to obey linear separability.

In order for such a dividing hyperplane to exist, the Hahn-Banach theorem states the following is a necessary and sufficient condition: the convex hulls of S^+ and S^- do not intersect. (A convex hull is the shape defined by the "outermost" points in a set and it encloses all the other points. For a more detailed discussion of convex hulls please refer to [15] pgs. 20-25)

Assuming at least one dividing hyperplane exists, an algorithm has been developed for actually finding one such hyperplane. This algorithm, equivalently, trains the single perceptron adequately. The algorithm uses the industry standard simplex algorithm in order to find a feasible point (starting point which the hyperplane will contain or come close to). It then proceeds by an iterative process

which is equivalent to online backpropagation described in section 4.5.1 [15]. This algorithm is called PTA (Perceptron Training Algorithm).

In the case of non-separable inputs, the optimal solution is most often a hyperplane with the fewest number of classification errors (although minimizing the distance to the hyperplane could also be used). This optimal solution must, of course, classify at least half of the points correctly (otherwise its converse would be better). An algorithm for finding this optimal solution, however, has not yet been found.

It has been shown that finding this optimal solution is an NP-complete problem [16]. For those unfamiliar with NP-complete problems, this is a class of problems whose solutions can be checked, but cannot yet be found in polynomial time. Additionally, the problems included in this class have all been shown to be reducible to one another. Thus, if a polynomial time solution is ever found to one of them, all the problems in this class would be solvable in polynomial time. Conversely, if one problem is proven not to be solvable in polynomial time (which seems to be the general consensus at this time), all the others are also unsolvable in polynomial time.

Blum and Rivest [17] evaluate the difficulty of NP-complete problems:

"While NP-completeness does not render a problem totally unapproachable in practice, it usually implies that only small instances of the problem can be solved exactly, and that large instances can at best only be solved approximately, even with large amounts of computer time."

The algorithms that give sub-optimal polynomial time solutions to NP-complete problems are categorized as approximation algorithms.

For the problem, the best known approximation algorithm is an online revision of the PTA Algorithm called a pocket algorithm [15, pp 37-38].

5.1.1.2 Multiple-node networks

Blum and Rivest also show in [17] that training a 2-layer, 3-node, n-input neural network is NP-complete. As is pointed out in [15] pp. 176, this means that no best overall strategy for training multiple-node networks is likely to exist. Indeed, the 2-layer, 3-node n-input network is by far one of the simplest used. (Neural networks are generally used because of overwhelming complexity of the problem)

However, in the case of real-input binary-output (RB) network nodes, it is guaranteed that there will always be a net which recognizes every training set correctly. The Sandwich Construction Theorem gives a way to make a net with $2 * \lceil n/2d \rceil$ linear layer 1 units and one linear layer 2 unit which recognizes any partition S^+/S^- of a set S of binary values [15, pgs. 53-66].

RB network nodes are generally used for pattern classification, so this approach may seem useful in many applications presented in section 6.2. Unfortunately, the Sandwich Construct does not employ an optimal training set size to internal nodes structure ratio (too low a ratio). This shortcoming leaves the resulting network lacking in accuracy for classification, and does so because of its overly general approach. This is the phenomenon known as overfitting.

In conclusion, training a multiple-node neural net has been proven to be an art. Care must be taken to ensure several conditions are met. Among these, the training set size must be sufficiently large compared to the number of internal nodes in order to avoid overfitting. In the end, an optimal balance must be struck between the number of training set errors and unnecessary complications of the internal representation of the neural network.

5.1.2 TDNN Theory

In many applications, time dependencies dictate lags in the causality relations implicitly derived within the neural network. Building time into the workings of a neural network has been attempted through two main neural network techniques.

Time-delayed neural networks (TDNNs) are the simpler approach, they involve simply using shift registers (or the equivalent of such a structure) in order to preserve, or keep around earlier values found within the network. The lagged values may be either input values, or hidden layer values from within the network.

Recurrent neural networks (RNNs) build temporal information by adding a carefully chosen set of feedback connections to the feedforward network. These feedback connections are used to remember important cues about the past. A more extensive overview of recurrent neural networks is beyond the scope of this paper, however. The

reader may use [5, chs. 14 & 15] and [9] as a starting point for a theoretical background to RNNs.

Building time into a neural network architecture essentially means allowing it to have memory. In practice, there are two kinds of memory: short-term and long-term. How are both these kinds of memory built into the architecture of a TDNN?

Short-term memory in a TDNN is simply built by virtue of the shift registers introduced. Thus, as many past values of the input as necessary can be reproduced by ensuring enough lags at the particular level are kept around. This is also true at any of the hidden layers. The decision on how many of these lags are kept around at each level of the network is part of the art that constitutes hand-tuning a neural network in general.

As presented in section 5.3.1, attempts have been made to have the network determine how many lags should be kept around in a dynamic fashion. As far as short-term memory is concerned, however, it is easy to see that it almost solely resides in the first hidden layer in any TDNN. This is because all the units in the first hidden layer are connected to all the lagged data that the network stores in short-term memory.

Long-term memory is actually the objective of building a time-delayed neural network in the first place. The network must recognize patterns in a time-invariant way, and apply this knowledge to the data to be classified. The hierarchy of layers within a time-delayed neural network allows the short-term memory patterns available at a lower layer to be propagated and transformed into long-term memory at a higher level.

This evolution of short-term memory patterns and how they affect long-term memory patterns is a direct result of learning. The features of long-term memory thus clearly must reside close to and into the output layer.

A close study of the additional architecture that transforms an MLP into a TDNN has been presented. As mentioned in section 4.5.5, because of this architectural difference, a new backpropagation method used must be modified to update all the weights.

Notice that simply unfolding the network to contain several parallel lagged nodes at each level, and then using any of the first four backpropagation methods in sections 4.5.1 - 4.5.4 will not be efficient. Modeling of the shift registers at all of the hidden layers would first have to be accomplished. When updating the weights corresponding to any of the delayed values, the change would actually have to be calculated with respect to the weights at the time the delayed value was determined! (since all the weight changes afterwards should not be taken into account, they happenned after the value was determined) This would force a much greater space usage for the man more weights that would need to be stored.

This instantaneous gradient approach suffers from several drawbacks: [5, pp. 653]

- loss of symmetry between the feed-forward propagation of states and backpropagation of terms required for calcuating the error derivatives/gradients
- no recursive formula for propagating error terms
- need for additional bookkeeping to keep track of the correspondence between the static unfolded weights and the original weights in the TDNN

Because of these drawbacks, the following alternative suggested by Wan. [5, pp. 653-657] is considered.

Using

$$\delta E_{total}/\delta w_{ji} = \Sigma_n \, \delta E_{total}/\delta v_j(n) * \delta v_j(n)/\delta w_{ji}$$

weight gradient descent can be applied to obtain:

$$w_{ji}(n+1) = w_{ji}(n) - \eta \, \delta E_{total}/\delta v_j(n) * \delta v_j(n)/\delta w_{ji}$$

$$\Leftrightarrow$$

$$\Delta w_{ji} = r \, \delta_j(n) \, x_i(n),$$

with δ_j defined as before.

After further derivations [5, pp. 653-660] similar to those for online backpropagation but slightly more complex, the formulas for $\delta_j(n)$ are obtained. They depend on whether or not neuron j is an output or hidden layer neuron. Note that these are the same equations revisited from section 4.5.5.

$$w_{ji}(n+1) = w_{ji}(n) + r \, \delta_j(n) \, x_i(n)$$

with:

$$\delta_j(n) = e_j(n) \, \varphi'(v_j(n))$$

for the nodes j in the output layer and

$$\delta_j = \varphi'(v_j(n)) \, \Sigma_{r \in A} (\Delta^T{}_r(n) * w_{rj})$$

for the nodes in the hidden layers. Here φ' is the derivative of the activation function.

Figure 5 below details how this backpropagation algorithm finds the delta gradient term by using all neurons k in the next layer which are affected by neuron j in a feed-forward fashion. Notice that not much additional bookkeeping is required at all, and symmetry is preserved between the feed-forward and backpropagation stages. Also, there

is no loss of efficiency like in the original method, because in the temporal back-propagation algorithm each unique weight is used exactly once for calculating the deltas.

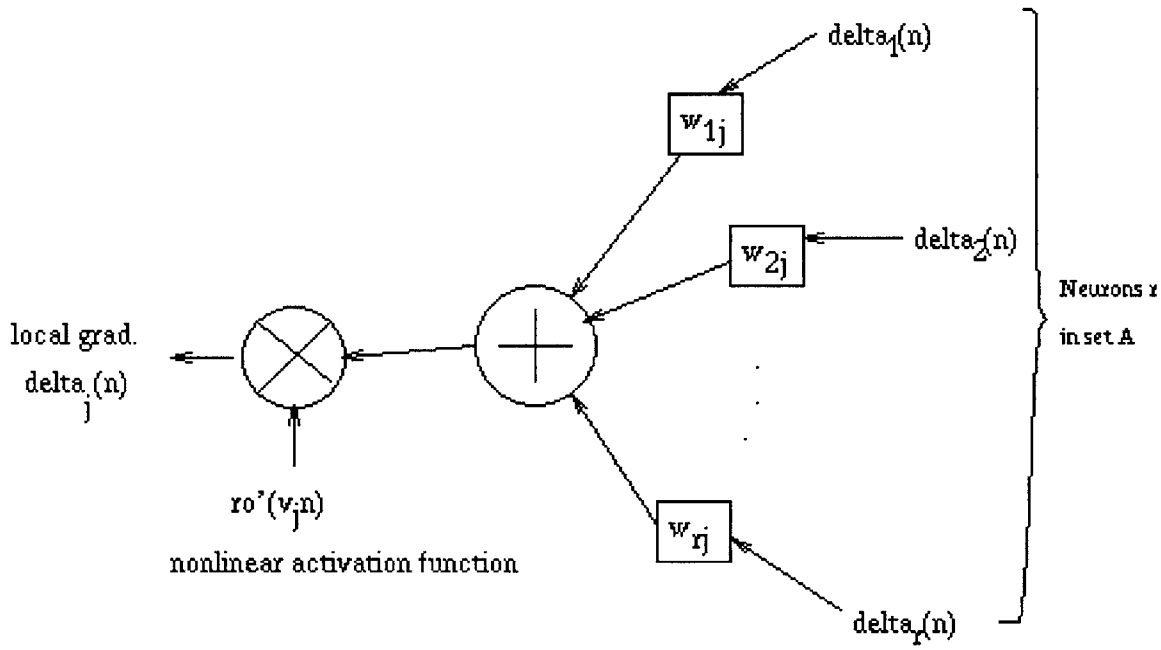


Fig. 5: Temporal Backpropagation, neuron level view.

Again, to make the temporal backpropagation equations causal, a change of index for the delta term calculated must be introduced[5, pp 658]. Replacing delta-j(n) with delta-j(n-p) as an index in 13.46 and propagating through this change, the following temporal backpropagation algorithm rules are obtained:

Temporal Backprop Alg.

1. Apply the feedforward algorithm and calculate the error for the neuron o in the output layer e-o(n). This error is simple the absolute value of the difference between the desired output and its value resulting from the feedforward stage.

2. Compute the following for the output layer neuron:

$$\delta_j(n) = e_j(n)\ \varphi'(v_j(n))$$

$$w_{ji}(n+1) = w_{ji}(n) + r\ \delta_j(n)\ x_i(n)$$

(as seen above)

3. For neuron j in the hidden layer, calculate:

$$\delta_j(n\text{-}lp) = \varphi'(v_j(n\text{-}lp))\ \Sigma_{r\in A}\ (\Delta^T{}_r(n\text{-}lp) * w_{rj})$$

$$w_{ji}(n+1) = w_{ji}(n) + r\ \delta_j(n\text{-}lp)\ x_i(n\text{-}lp)$$

The only difference between the causal and non-causal versions of the temporal backpropagation algorithm is that like the instantaneous backpropagation algorithm, the causal TBA propagates deltas without added delays. This forces the internal values to be shifted in time.

As a result, the states must be stored in delayed form in order to accomodate changes to the weight vectors. This constitutes the only drawback, as the deltas do not have to be delayed in time and the symmetry between the feed-forward and backpropagation stages is preserved. Similar to the instantaneous gradient approach, the number of computations is linear in the number of synapses [5].

It is important to understand that in theory, regular TDNNs are equivalent to TDNNs which only contain lags in their input layer. This latter subset of time-delay neural networks has sometimes been called input delay neural networks (IDNNs).

Indeed, IDNNs are capable of recognizing all languages representable by a definite memory machine. (DMM) It is also well-known that general TDNNs represent the same set of languages that DMMs recognize [23].

What are DMMs? They are finite-state machines (FSMs) with an associated order d whose present state can always be determined uniquely from the knowledge of the last d inputs. Thus, a DMM of order d is an FSM whose accepting/rejecting behavior is only influenced by the last d inputs. For a great overview and introduction of FSMs and computability, the author suggests [24]. This book is currently being used as the textbook for the Theory of Computation course offered at MIT.

Neural networks have evolved as an instrument of coping with complexity rather than an interesting mathematical or computational abstraction. As a result, it is more important what neural networks can actually learn, rather than what neural networks can represent. It would be almost useless if neural networks had the ability to represent a certain language if the language could never be learned.

Thus, the question of finding what problems the two kinds of TDNNs are actually adept at learning (rather than just representing) is the next important question that must be answered. In order to answer this question, it will be helpful to define the terms narrow and repeated in order to characterize a function.

A function is repeated if its underlying logical pattern is observed to repeat across time. The term narrow is defined to represent the ability of a function's logical pattern to be completely defined within the number of delays which characterizes the TDNN. Thus, a narrow pattern "fits" into the delays which are used for the network and must at some instant be completely present within the taps which are connected to a node in the hidden layer.

As shown by the experiments performed in [23], it turns out that TDNNs with delays at the output of a hidden layer (commonly referred to as HDNNs in this paper, HDNNs = TDNNs - IDNNs) have a decided advantage over IDNNs in learning problems which are narrow and repeated in nature. On the other hand, IDNNs do better at learning problems which are not narrow and repeated.

Although widely accepted as a computational tool, not many additional theoretical results have been derived specifically for time-delayed neural networks. This is a direct consequence of the fact that although dynamically different from feedforward networks, TDNNs are essentially just "bigger" MLPs. In the future one must be able to prove results which more precisely differentiate between the learning that occurs in TDNNs and the learning done by simple MLPs. There clearly is a difference between the learning which takes place in TDNNs and the learning corresponding to MLPs because of the dynamic structure and more complicated backpropagation of the latter.

Understanding how the dynamic structure of the network affects the learning process, however, is still an important open question which has only been answered partially. At the same time, neural networks have yet to be fully understood. For the

most part, they are still regarded as complex black boxes whose purpose is to deal with problems that are hard to understand.

Theoretical computer science's goal is to be able to precisely quantify and explain what kind of training sets each network can learn and in what fashion, as well as how this will affect the testing results. If researchers had able to prove several theorems and results about these objectives, many of the tasks performed by neural networks would be regarded as the outputs of a structured algorithm. Instead, neural networks have retained some of the mystique which truly makes them part of artificial intelligence.

5.2 TDNN Applications Examples

This subsection includes several examples of TDNN real-world applications. Although efficiency today dictates that computational tools be combined in order to achieve optimal results, most of the applications in this subsection are purely TDNN results. Sections 6.4 and 6.3 have been dedicated to computational optimizations to TDNNs, through both modifications to the original TDNN architecture proposed by Waibel [4] and hybridization with other computational techniques, respectively. This section will therefore be kept as a survey of purely TDNN-based applications.

5.2.1 Speech Processing

Time-delayed neural networks were initially designed as a tool for speech processing. NETtalk can be considered as one of the most famous early neural network applications for speech processing. A giant system designed and deployed by Sejnowski and Rosenberg in 1987, NETtalk was the first example of a largely distributed network that converts English speech to phonemes (or basic linguistic units) [5, pg 640] [9, pg 177].

The NETtalk neural net was composed of an input layer with 203 sensory nodes, a hidden layer with 80 units and an output layer with 26 outputs (one for each letter of the alphabet). In addition, NETtalk's brand of artificial "intelligence" seemed rather similar to human performance on the same task:

- the more words the network learned, the better it would generalize and pronounce the new words
- neural net performance was only affected very little as its synaptic connections were damaged
- learning after suffering damaged connections occured a lot faster than before
- training followed a power law

Another famous early TDNN result which proved the viability of time delayed neural networks in this domain was Waibel's average of 98.5 recognition of phonemes B, D and G. This recognition rate was much better than the recognition rate for the control,

the Hidden Markov Model technique. HMMs were the widely accepted norm in the field at the time. Their recognition rate of 93.7% in Waibel's experiments.

Despite several advances and TDNN refinement, HMMs have remained the preferred method for speech processing. This is a direct consequence of a basic assumption that using TDNNs for speech processing makes, namely that speech can easily be segmented into its constituent phonemes. In practice, this is a very hard problem because of the variations in pronounciation introduced by several factors such as length of sentence and individual speaker bias [5].

Pure TDNN applications in speech processing are still widespread, as the speech segmentation problem could easily be solved at any time. [25], for example, extends Waibel's early results to classification of English stop consonants b, d, g, p, t and k.

The recognition rate achieved was 88%. The authors also detail the hand-tuning which went on in finding the optimal TDNN for their model, as the result was an unusual 9-256-6-6 architecture [9 input nodes, 256(!) nodes in the first hidden layer, 6 nodes in the second hidden layer and 6 output nodes, each corresponding to one of the classified letters]. While the very high number of nodes in the second layer is rather unusual, reading this particular paper inspired me to try a second hidden layer with a number of nodes closer to the number of nodes in the output layer. This strategy eventually resulted in the most successful observed TDNN, which is covered in the Results section.

As shown in sections 6.3 and 6.4, there are several ways in which one can supplement TDNNs in order to improve their performance. A pure TDNN approach which supplements TDNN speech recognition is lip-reading [26, 27]. Lip reading has also been approached through TDNNs since its data presents the same temporal locality as

speech processing. With the help of lip-reading cues, it is hoped TDNN performance on speech recognition will be improved.

The basic approach to lip-reading is simple. Three different stages make up the algorithm for lip-reading, while only the last stage is processed using a TDNN approach. These stages are: human mouth detection and extraction, mouth feature detection and neural network learning. A very interesting human mouth detection and extraction algorithm is also presented [27].

5.2.2 Stock Market

Another common application of TDNNs is stock market prediction. Because of its natural time patterns, past movements/trends in stock price are very important when trying to predict future developments. This time dependency lends itself well to use of TDNNs, which were designed to take into account exactly such patterns.

Similar to the field of speech processing, smaller problems have been the focus of stock market TDNN applications. The work done in this field so far has been focused on stock trend prediction, rather than full-fledged stock price prediction. The particular stock trend predicted in [20], for example, is a binary output value indicating whether or not the stock in question experienced a 2% increase within a 22-day period. Accurate prediction of these kinds of trends is useful in financial applications such as options trading. For these applications, false alarms generally incur great costs, resulting in a necessity for accuracy.

Both [19] and [20] examine such stock trends for well-established stocks like Apple, IBM, Motorola, Microsoft, American Express, Wells Fargo, Walt Disney Co. and McDonald's with the help of several prediction tools, including TDNNs. The TDNN results were optimized by including a higher penalty factor for mispredictions than reward for correct predictions to model the real life scenario of option trading.

Overall, the TDNN approach proved fairly effective. Its results were close to averages from the other neural networks used, with false alarm ratios of 0%, 8.1% and 13.8% for Apple, IBM and Motorola stocks [20]. These results are especially encouraging given the volatile nature of technology stocks in the market today. Note also that the overall upward movement of the stock market and the nature of option training made predicting upward stock trends preferable over downward trends.

With advances in neural networks in general and TDNNs in particular, there may be increasingly more difficult stock market problems tackled through data mining techniques. Some of these advances and optimizations will be examined in detail in Section 6.4.

5.2.3 Student Admissions

A fairly new application of neural networks, student admissions prediction could end up relying heavily on TDNNs. Colleges like Johns Hopkins University currently employ neural network architectures to predict whether or not students will accept

admission offers based on the extensive student data they have available [18]. Using these results, the colleges can better approximate the number of students to offer admission to in order to satisfy their admissions quota. The neural network method obtained an error rate of 3 percent, while the techniques used previously faltered 6 percent of the time.

TDNNs are usually successful in fields where definite temporal patterns exist. Historically, colleges have experienced definite such trends. Their admissions numbers have fluctuated based on temporal factors such as industry growth or innovation in a certain featured university field, previous admission of famous students or availability of new funding.

As many as 50 percent of colleges nationwide are expected to employ neural networks for student acceptance prediction by the year 2010. TDNNs can be expected to become widely used in this new and expanding field.

5.2.4 Other Applications

There are many other applications for time-delayed neural networks. An exhaustive coverage of these would be impossible, as TDNNs have become a norm for analysis of time dependency in the neural network field.

One of the more interesting applications using TDNNs is a neural net designed to measure raindrop sizes and velocities based on photodiode sensors [21]. The photodiode output current changes as the drops pass through its light beam. The architecture was designed to take adavantage of the redundancy of two beam slots instead of only one.

Thus, in spite of lower individual unit accuracy, the higher order module exploits the redundancy of the two lower order modules to give 100% efficiency for droplet detection.

Other newer pure TDNN applications include but are not limited to image processing, human signature verification, tool wear prediction based on individual component wear and online multiprocessor memory access pattern prediction [9, 21, 28, 29]. Although TDNNs have proved effective in several domains, it is clear that the future will see more successful hybrid TDNN application rather than pure TDNN applications. This is a result of each computational tool being best at a particular type of problem. Optimal hybridization, where possible, will allow two such computational tools to complement each other nicely, as the next section will show.

5.3 Modified TDNN Architectures

During the past decade, TDNNs have become widely accepted as neural network computation tools. As a result, their architecture has come under close scrutiny, and several improvements have been attempted. In this subsection, the paper will take a closer look at several of these improvements which involve combining the basic (unmodified) TDNN unit with other computational units and/or abstractions.

The subsection will begin by highlighting the features that constitute the underlying difference between the new system and the regular TDNN architecture. Afterwards, focus will be shifted to an attempt to explain how these modifications adapted the TDNN

architecture to the particular problem, and why the innovation resulted in a better solution to the particular task at hand.

5.3.1 Two-level TDNNs (TLTDNNs)

This particular modification is probably the most trivial change of all the examples examined in this subsection (6.3). A two-level TDNN was used because of the complexity introduced by the size of the problem. In order to reduce this complexity, two TDNNs with relatively orthogonal tasks were employed in parallel to work on the same input. Their results were then simply combined with the help of a simple table to identify the final output of the combined system unit [30]. A summary diagram of the architecture is given in Figure 6 below.

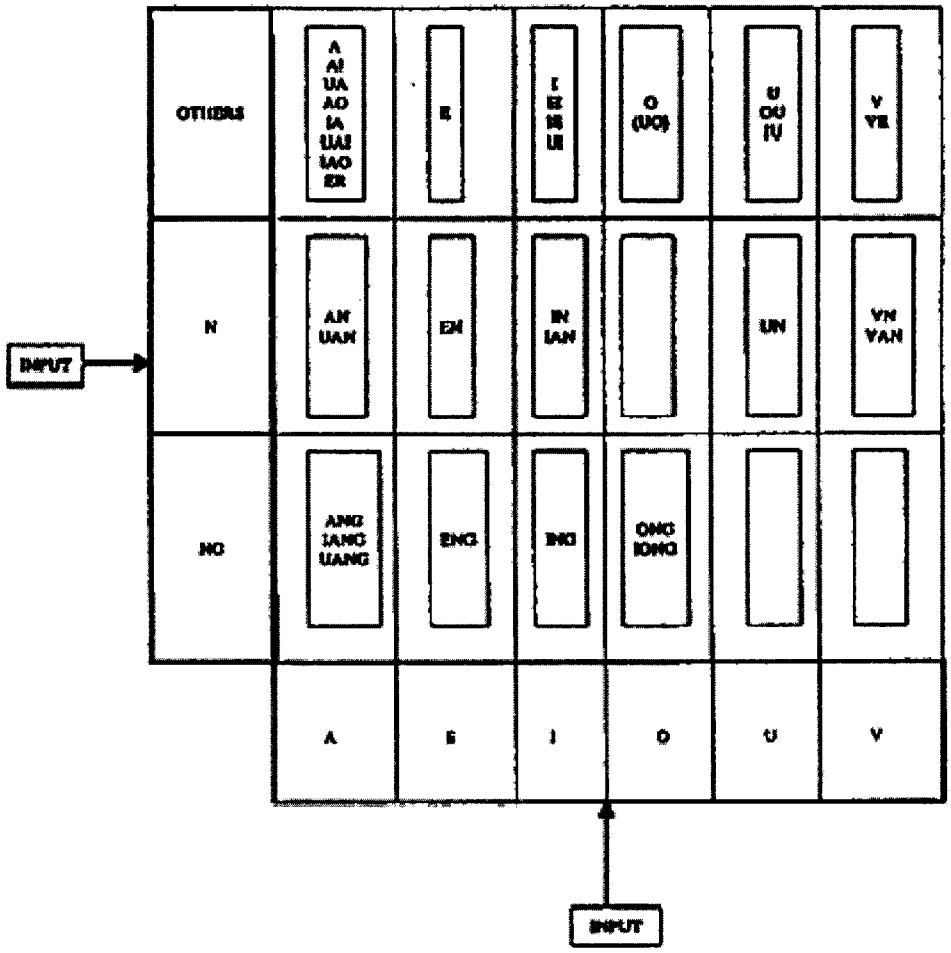


Figure 1 The TLTDNN architecture

Mandarin Finals of the entire Chinese syllables recognition was the particular task that prompted use of the given modified architecture. The nature of the problem lent itself well to this approach because the problem of recognizing the input vowel-group (a, e, i, o, u, v) was essentially different from the problem of identifying the nasal ending of the input (-n, -ng and others).

As a result, the orthogonal problems approach was used to greatly reduce complexity in a case where many more nodes would have had to be trained otherwise. Training two reasonably-sized TDNNs in parallel (or even one after another) is much

easier than training one TDNN which has more than two or three times as many nodes. This approach proved successful, with the average recognition rate for the Mandarin Finals around 93 - 94%.

5.3.2 Adaptive Time-Delay Neural Networks (ATNN)

Adaptive Time-Delay Neural Networks (ATNNs) are a generalization of TDNNs. In addition to training the feedforward weights appropriately like a regular TDNN, they also attempt to dynamically adapt the number of delays in the network so as to provide the optimal internal architecture for the solution.

One of the intrinsic limitations of TDNN learning is the number of delays which is input at the onset of training and remains fixed throughout. ATNNs attempt to adapt both weights and time delays during training, for a one-size-fits-all solution to any problem solvable by TDNNs.

The first contributions to the ATNN field belonged to Ulrich Bodenhausen, who authored and co-authored several papers in this domain [31]. His systems, the Tempo and Tempo 2, both implemented adaptive time-delays, but they applied only to a particular recirculation network [32]. Bodenhausen's results, however, were remarcable. In the experiment described in [31], namely B/D/G phoneme recognition -- the benchmark of all TDNN experiments, his ATNN obtained recognition rates of 98% when trained for all of weights, delays and layer widths. It is important to note, however, that even for this

specialized ATNN result, the classification rates were lower than those obtained by Waibel in 1989. Waibel had obtained recognition rates of 98.5%.

More general additional works followed in the field of ATNNs. The theoretical base for ATNNs, along with a derivation of the learning/backpropagation algorithm was done [32]. From this work, the sample ATNN unit is reproduced below. Notice the extra index on the weight connections between layers due to delay learning.

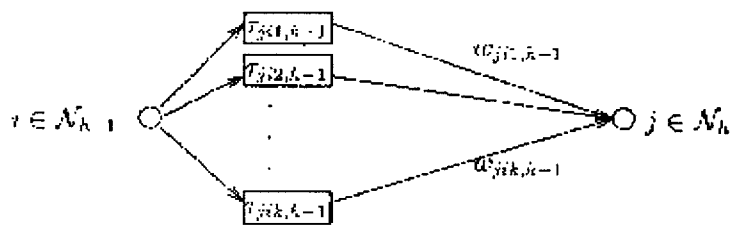


Figure 4: Basic time-delay connections between two neurons (node i of layer $h-1$ and node j of layer h) in ATNN

As a result of this architecture, the corresponding backpropagation algorithm includes formulas not only for weight learning but also for time-delay adaptation:

Weight learning:

$$\Delta w_{jik,h-1} = r\, \delta_{j,h}(t_n)\, a_{j,h-1}(t_n - \tau_{jik,h-1})$$

$$\delta_{j,h}(t_n) = (d_j(t_n) - a_{j,h}(t_n)) * f'(S_{j,h}(t_n)),$$

for j in the output layer and

$$\delta_{j,h}(t_n) = (\Sigma_{p \in N(h+1)} \Sigma_{q \in Tpj,h}\, \delta_{p,h+1}(t_n) w_{pjq,h}(t_n)) * f'(S_{j,h}(t_n))$$

for j in the hidden layer.

Time decay learning:

$$\Delta\tau_{jik,h-1} = \eta_1\ \rho_{j,h}(t_n)\ w_{jik,h-1}\ a'_{j,h-1}(t_n - \tau_{jik,h-1}),\ \text{where}$$

$$\rho_{j,h}(t_n) = -(d_j(t_n) - a_{j,h}(t_n)) * f'(S_{j,h}(t_n)),$$

for j in the output layer and

$$\rho_{j,h}(t_n) = -(\Sigma_{p\in N(h+1)} \Sigma_{q\in Tpj,h}\ \rho_{p,h+1}(t_n) w_{pjq,h}(t_n)) * f'(S_{j,h}(t_n))$$

for j in the hidden layer.

A derivation of these results is available in [32].

Despite the strong theoretical appeal and developed foundation, ATNNs failed to dramatically improve results generated by TDNNs. A number of factors contributed to this. The most significant of these factors was probably that better exhaustive results could be obtained by further refining the ATNNs resulting network with a hand-tuned TDNN which "borrowed" certain key features from the ATNN.

But how could this happen when all the parameters were already being optimized for?

A hint of the answer comes in [31], where the author admits that the three learning parameters for weights, delays and layer widths could not be chosen independently! Choosing one of them to be much larger than the others would result in "egotistical" training of that parameter over the others, as Bodenhausen himself admits [31].

Evaluating the net change in the problem, the approach has gone from having to fine-tune three parameters in TDNNs (weight learning rate, time delays and number of hidden layer nodes) to still having to fine-tune three parameters, namely the three learning rates.

For lack of a better source, the best guess the author of this document is able to offer is that optimizing the TDNNs three parameters proved more effective because they were more directly related to the solution of the problem. Fine-tuning the weight and number of node learning rate in the ATNN influenced the learning of parameters which were manually exhaustively optimized in TDNNs.

Although the two techniques may seem to lead to a common answer, practice proved that one should never give up the bird in hand for the one in the bush. When applied to neural networks, this lesson means that just because a network has the ability to represent something, it doesn't necessarily mean it will end up learning it. In addition, the extra learning done in ATNNs results in orders of magnitude more training time as seen in the previous section, since extra features need to be trained for.

5.3.3 Memory Neural Networks (MNN)

Memory Neural Networks (MNNs) are part of recurrent neural networks. They will be examined because of the striking similarities they share with TDNNs. During this analyis, it must be kept in mind that the added feedback connections have two effects on the model. The positive effect is that temporal sequences, and especially distorted temporal sequences, are more likely to be recognized by the architecture. The price paid for this ability is an increase in the already high training time of the network (the second effect of the recurrent connectionist architecture).

As shown in [49], MNNs differ very little from TDNNs. Like TDNNs, MNNs layer units are made up of "main" network neurons (which are not delayed) and memory neurons (the neurons which correspond to the delayed values in a TDNN).

The equations describing this network (MNN) are reproduced here:

$$x_i(t) = \Sigma_{j \in N(i)} \Sigma^{K}_{k=0} w_{ijk}\, u_{jk}(t)$$

$$u_{i0}(t) = g_i(x_i(t))$$

$$u_{ik}(t) = (1-\mu_i)u_{ik}(t-1) + \mu_i u_{i,k-1}(t-1)$$

As can be seen from the last equation, the parameter μ is the memory coefficient which determines the activation of the memory neuron. Setting this parameter to 1 (μ=1) yields a TDNN architecture because the memory neuron just becomes the delayed version of the neuron above it:

$$u_{ik}(t) = u_{i,k-1}(t-1) = u_{i0}(t-k)$$

Also, setting this parameter to 0 yields a static MLP architecture as the memory neurons do not affect the calculation of the output. (Since the memory neurons serve no purpose and take up valuable training time, an MNN with μ=0 should never be run. Rather, one should just settle for an MLP model).

The application the MNNs were used for in [49] was the classic phoneme classification problem for B, D and G. The error rates obtained were less that 5%, which is comparable with Waibel's results. In addition, the authors of [49] further suggest that hybridization of MNNs will lead to better results. The now-popular technique of

computational hybridization of TDNNs will be examined in section 5.3.6, although not necessarily in the context of using memory neural networks (MNNs).

5.3.4 Multi-State TDNNs

Multi-state TDNNs (MS-TDNNs) are the first of the hybrid TDNN architectures examined. MS-TDNNs combine the basic TDNN unit proposed and perfected by Waibel [4] with a nonlinear time alignment procedure (Dynamic Time Warping, or DTW) into a word classifier for the telephone spelled name recognition problem [33].

The ability of TDNNs to outperform other classification means in the phoneme problem is revisted and used here. This feature, along with a good performance of the DTW unit and an efficient probabilistic tree-search procedure allow for rather lofty classification accuracy figures. Most of the recognition performed obtained rates of 90% or much better for this difficult problem [33].

5.3.5 Cascaded TDNNs

Another of the many improvements in TDNN architecture design, cascading neural nets has been proven effective in several applications. This technique is somewhat similar to the one seen in section 6.3.1. Two TDNN neural nets are used in parallel and their

outputs are allowed to interact in a certain fashion that is useful to the particular application being explored.

Below is a figure detailing the TDNN - TDNN cascade interaction employed in a magnetic resonance acoustic noise cancellation application. Note that both the regular and delayed output of TDNN unit number 1 are allowed to affect the second TDNN unit. The result is an Active Noise Control (ANC) system which reduces magnetic resonance noise [34].

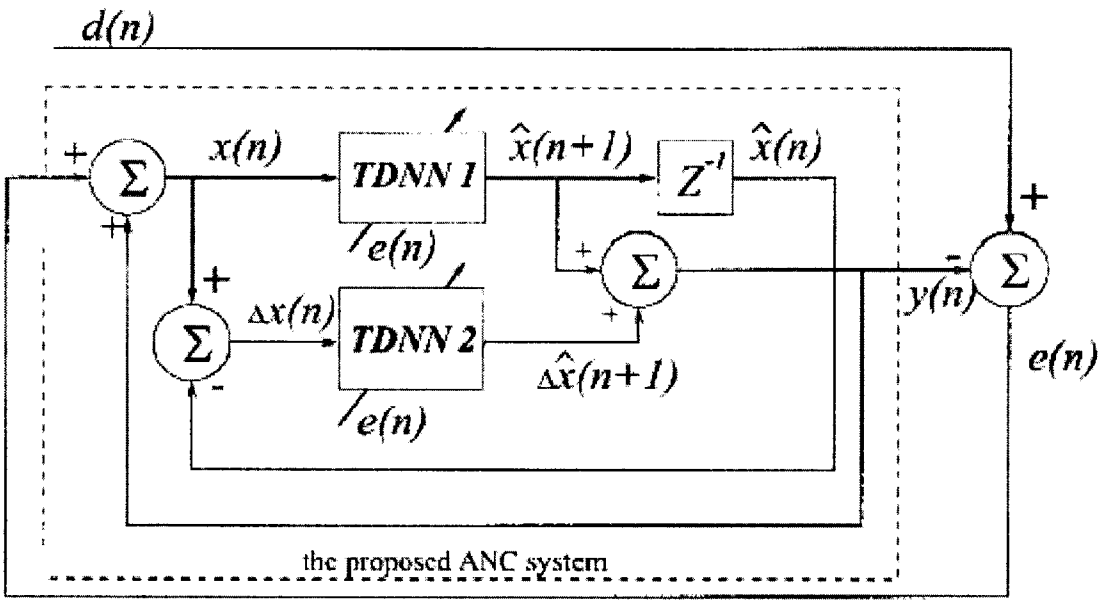


Fig. 8. Two TDNN's are cascaded to implement the proposed ANC system.

Although the unit has only been tested succintly, encouraging results were obtained as the broadcast signal tested on was received cleanly and free of noise.

5.3.6 Hybrid systems using TDNNs

This paper has introduced to the idea of hybridization of a TDNN unit along with other computational resources which complement its weaknesses before. In this subsection a look will be taken at a couple of examples of these architectures.

One of the simpler such hybrids is the one tested for a small word vocabulary problem in [35]. Here the TDNN is augmented with a simple dynamic programming (DP) algorithm which allows it to perform better at the higher stages. The figure below (Figure 9) details the architecture that increased recognition rates for the given problem from 94.5% and 96.5% for the TDNN by itself to 98.8% for the hybrid TDNN-DP architecture.

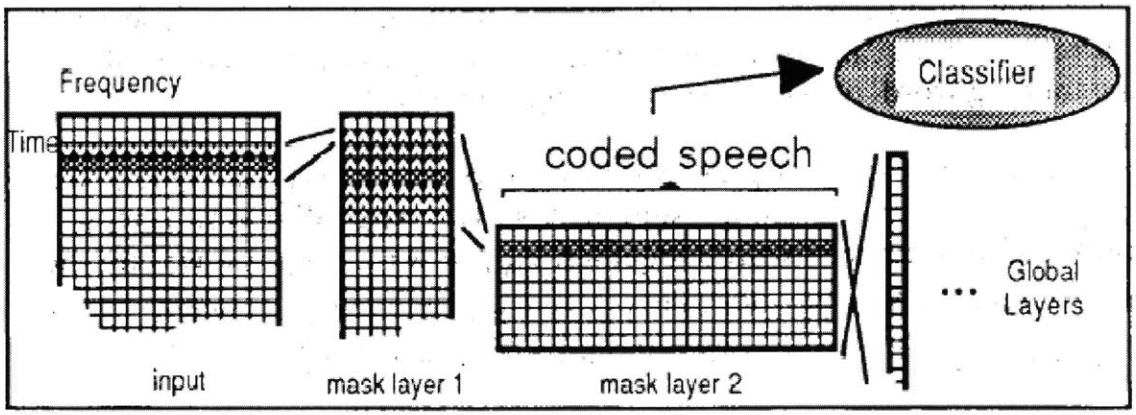


Figure 9: TDNN unit used as the first part of a hybrid to recognize the short-term temporal patterns. Note that the long-term pattern detection is mostly left up to the Classifier.

Note that once again, the TDNN's strength of encoding basic phoneme information is used at a very low level. The higher level decision-making is left up to the dynamic programming unit, which was simply trained with the examples from the training set [35].

Another example of TDNN hybridization into an architecture can be seen in yet another phoneme classification problem [36]. This time the TDNN is combined with a self-organizing clustering procedure, which approximates a dynamic nearest neighbor algorithm. The recognition rates obtain are in the mid-80%'s, which is good considering the fact that the problem solved is speaker-independent phoneme recognition.

The most commonly encountered TDNN hybrid, however, is the TDNN/HMM hybrid used ubiquitously in speech recognition. While the TDNN is normally used for lower level phoneme identification, the HMM then takes over higher order patterns with its probabilistic analysis. Several speech recognition applications have been developed using this model, and only a few are included here.

These applications outperform both HMM and TDNN speech processors because each computational unit (i.e. TDNN and HMM) is applied to the part of speech recognition which it excels on [50, 51, 52].

As more advanced techniques are developed that employ TDNNs, it is expected hybrids will play an important role in obtaining better artificial intelligence computational models. Doing so will require an increasingly better understanding of the problem. This superior understanding of the problem will then hopefully result into being able to divide a problem up into two or three orthogonal smaller problems. All that is then left is to use

the particular tools which perform best at solving these smaller problems. Once these results are obtained, one can then combine the individual answers to satisfy the initial problem. The true new measure of how easily a problem will be solved by such techniques will be just its ability to be broken down into smaller tasks which can be readily be solved efficiently.

5.4 TDNN Optimizations

This subsection will take a closer look at TDNN optimizations which have attempted to internally modify the architecture proposed by Waibel in [4]. While these are not the only attempts to date, this paper has presented a selection of the more successful. Note that some of these techniques, such as input saliency and optimizing of the running time could and already were applied to neural networks in general.

In this section, the author has tried not to present optimizations which were touched upon in previous sections, such as the momentum backpropagation technique from section 4.5. Rather, the focus has been shifted to new techniques that optimize not only the neural network architecture and perceived performance, but other parameters such as the training time. These optimizations all have the potential to cause more widespread acceptance and use of neural networks technology.

5.4.1 Input Saliency

One common optimization applicable to all neural networks is input sensitivity analysis. The main idea behind this concept is that extra nodes in the network not only require additional training, but end up creating noise which may influence the accuracy in a negative fashion. The biggest causes of extra network nodes which end up affecting performance negatively is the inclusion of one or more inputs which are not in reality correlated to the output. The basic problem leading to this is that one can sometimes be unsure of exactly which parameters should be included in the neural net and which should not.

The algorithm for alleviating this problem is simple. Several versions have been proposed [37, 38, 39]. Only the simplest are presented here, but the reader must keep in mind that alternative cost/reward functions may be used instead of the regular error functions used.

Here is the algorithm reproduced from [37]:

1) Train network with full input set

2) Calculate all input/output sensitivity functions according to

$$S = O'WY'V$$

, where W and V are the output and hidden layer weight matrices in a three-layer network (result is easily generalizable to more layers [38]) and $O' = \text{diag}(o1', ..., oK')$, $Y' = \text{diag}(y1', ..., yK')$, with oi' and yj' being the derivatives of the output and hidden layer activation functions.

3) Calculate the overall sensitivity matrix using one of a few alternative choices, for example maximum sensitivity mtx = max {Ski(p)} [37]

4) Calculate each input significance over all outputs as the maximum between all output sensitivities

5) Prune all inputs which score significantly lower than the highest echelon (e.g. look for first or last significant break in the input significance rankings)

6) Retrain network with the fewer inputs.

7) Repeat steps 1 to 6 until no irrelevant inputs are left.

Input saliency is very important to obtaining a healthy and trainable neural network and should always be used. Application of this to TDNNs presents some optimization issues since training time is already very high in time delayed neural networks relative to smaller networks without time delays.

5.4.2 Optimized Running Time

As mentioned in the last section, one of the major obstacles against using TDNNs is their prohibitive training time. Especially true when TDNNs first appeared [4], this constraint has been relaxed by the high rate of processor speed improvement and parallel architectures. Relative to other methods, however, TDNNs will always be somewhat slower in training. Despite the obvious benefits of this architecture (deploying the TDNN once it has been created is very efficient), it would be beneficial to propose some ways in which one can improve this aspect of traning.

There are two basic ways which have been employed to speed up TDNN training: software and hardware.

Most of the software techniques resemble these basic ones presented in [40]:

- the backpropagation algorithm initially trains the weights of the network;
- the single-extreme output is replaced by a double-extreme output (the more output units the better, as there is more room for differentiation);
- backpropagation is changed to update all layers in a similar fashion instead of just all the weights of the same time delay;

Hardware acceleration for TDNNs, although rare, has been accomplished through use of a grid-based SIMD computer [41]. This is a direct result of the parallel capabilities of this processor and yields great improvements in training times.

All these proven performance optimizations for TDNNs are very important as they cut into the time costs associated with running traning for this application.

5.4.3 Gaussian filters

Another problem discussed previously with TDNNs lies in the discrete nature of the delays. While this is not a problem if care is taken to precisely sample the input so the corresponding pattern's temporal alignment matches up with that of the network, it is often the case that there is not enough information to realize this synchronization.

Initially suggested as part of the ATNN architectures, Gaussian-shaped filter signals may replace discrete time sampling in order to alleviate this problem [9].

The two diagrams below show how Gaussian-shaped filters on the input window can help cope with imprecise temporal alignment of the input. Especially interesting is the interaction between two or more Gaussian-shaped filters and how they interact in smoothing out the input samples over time [31].

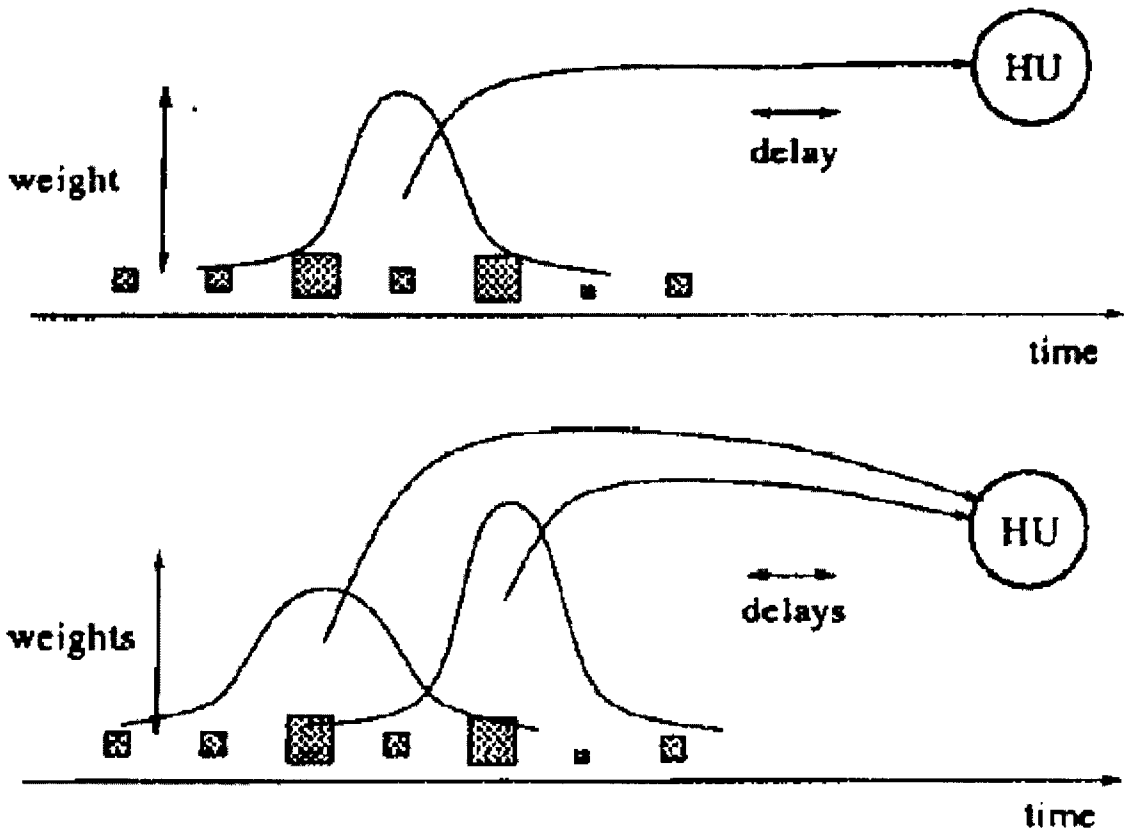


Figure 10: (a) Gaussian shaped input filter "smoothes" input.

(b) Interaction of overlapping Gaussian input filters.

Notice that although useful, Gaussian filters yield a further approximation of the input. Their usage may alter some of the temporal properties of the input and thus degrade the performance of the network.

5.4.4 Alternative Backpropagation Methods

Many alternative backpropagation methods have been proposed to temporal backpropagation. This paper will discuss two particular ones in this section, namely Truncated Recursive Backpropagation (TRBP) and autoregressive backpropagation (AR-TDNN).

TRBP is an online algorithm derived from Back-Propagation Through Time (BPTT). BPTT was initially an improvement over temporal backpropagation because of its recursive nature. The recursive nature of this algorithm is interesting enough to discuss here because it only manifests itself locally in the backpropagation stage. Because of its few feedback connections, BPTT is able to learn more complex temporal relationships than the standard temporal backpropagation algorithm presented earlier.

BPTTs drawback, however, is that it only works in batch mode. (and hence is not an online algorithm) TRBP solves this problem and results in results in good performance [47].

AR-TDNNs approach to dealing with temporal distortion also involves feedback weights. These weights allow it to discern specific temporal patterns based on common context, with good performance results [48].

Although locally recursive in nature, these backpropagation methods were examined because they improve temporal recognition performance in TDNNs. They could, alternatively, have been placed under hybrid TDNN systems, although this choice is somewhat arbitrary.

5.4.5 Parameter Sensitivity Alleviation

A problem which has received little publicity in spite of the wide usage of TDNNs is its susceptibility to parameter sensitivity problems. Parameter sensitivity problems occur consist of large perturbations in the neural network behavior resulting from very small input conditions changes. These characteristics are obvioiusly undesirable in a neural net such as a TDNN, which is both computationally expensive and is usually initially trained with small random weights.

The sensitivity of the TDNN-associated polynomial resulting from the existence of the FIR filter (or, equivalently, the existence of the time delays) is derived in [43] to be:

$$S_{ij} = d\lambda_i/da_j = \lambda_i^{n-j} / \Pi_{k\neq i}(\lambda_i - \lambda_k),$$

where λi is the i-th root of the polynomial equation

$$A(q) = 1 + a_1^{-1} + \ldots + a_n^{-n}$$

associated with the FIR synapse.

From here, it is easy to see that the two conditions which lead to instability are:

1) Two roots are very close to one anohter ($\lambda_i - \lambda_k \sim 0$)

2) λ_i has a large magnitude (close to 1)

These situations correspond to fast sampling or lightly damped low frequency (LDLF) conditions.

The proposed solutions to this problem in [43] are either to subsample (and thus intrinsically build more information in each sample) or to low pass filter the input.

While subsampling is trivial to implement, [43] goes on and provides alternative discrete time operators which can alleviate the ADTO problem if used in place of the regular shift operator $q\ x(t) = x(t-1)$. Testing with these filters yields the result that the delta, rho and pi operators can all improve performance over the shift operator. In addition to these benefits, the gamma operator also brings in architectural advantages as described by de Vries and Principe [44, 45].

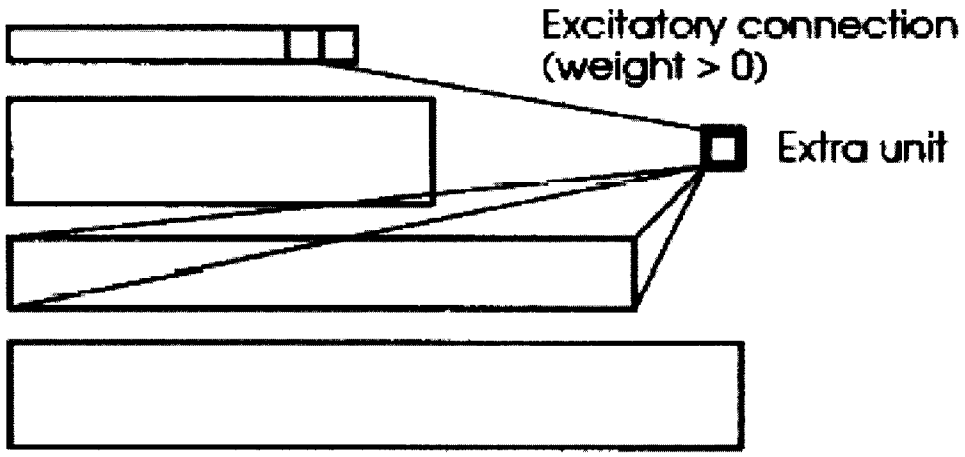
An alternative approach to the shift operator problem consists of a cascade form TDNN architecture, CTDNN [46]. (Note that this is different from the cascaded TDNNs seen in section 6.3.5) In this context, either cascade, parallel or cascade-parallel filters are proven more effective from a stability point of view than the shift operator.

5.4.6 One-shot learning during testing

A very interesting optimization of TDNNs was proposed in [42]. The change consists of one-step learning attempted by the already trained network during training. This learning attempts to deal with misconfiguration of the training set, where not enough training samples were supplied in order to cover the whole testing set, and "surprises" can thus occur.

There are two proposed ways to achieve this incremental learning which occurs only on missed samples in the testing set. One is to fine-tune the already existing units. The second way is employed only when the first fails. It consists of adding a dedicated extra inhibitory or excitatory (Figure 11) unit to the hidden layer right below the output. This unit's effect is to stimulate proper recognition upon reoccurence of a similar pattern to the one which was just misclassified [42].

FIGURE 2: Addition of an Extra Hidden Unit to Excite Output 8



In the given example, new digit recognition in the test set jumped up from 0% to 99% with the changes outlined above. While not a conclusive test in itself, this finding opens up new avenues of improving TDNN and neural network performance in general.

6 RESULTS

This chapter presents the neural network simulation results and attempts to interpret them. The neural networks trained solve the problem of predicting the hot metal temperature in a blast steel furnace one through seven hours in advance. This paper focuses mainly on mean-squared error (MSE) results obtained from the time-delayed neural network (TDNN) approach. The author of this paper worked

closely with both Brian Purville and Ravindra Sarma. These authors' papers present the results of the extended TDNN analysis (normalized mean squared error, average error, Pearson coefficient analysis, [59]) and those results obtained from the MLP (or feed-forward neural network, FFNN) approaches to the same problem [60].

This paper focuses mainly on MSE analysis of the results because this is a widespread and reliable means of analysis. The formula for mean squared error is:

$$MSE = \frac{1}{2} \Sigma_j (d - o)^2$$

The chapter will be divided into five sections. The first section will present a detailed look at the methodology employed in each of the approaches. The second section will describe the work with data pre-processed using the sliding window technique.

As this section's results will show, the usage of this sliding window technique gave rise to the anomaly of the testing data performing better than the training data! Indeed, in neural networks approaches, it is impossible with a sufficiently large data set to achieve

better prediction rates for the testing than the training. The paper will examine what caused this phenomenon with the data analysis in the third section.

The fourth section of this chapter will present a similar TDNN analysis using SNNS, this time done on data which was simply averaged over 60 minutes. As a result, the findings presented here accomplish one hour through seven hours ahead prediction of the hourly average hot metal temperature. The anomaly of the training data performing better than the testing data is not present in these results.

The fifth and final section of this chapter will compare this paper's results with those of the alternate approaches tried by the data mining group. These approaches include MLP/FFNN and genetic algorithm architectures, as well as a more mathematically comprehensive TDNN result analysis.

6.1 Methodology

Two different TDNN implementations were used to analyze the given steel data. The first was a matlab implementation and was developed by Prof. Eric Wan. The second TDNN implementation used is called the Stuttgart Neural Networks Simulator (SNNS) and is available as shareware at www.snns.com.

In using both these implementations, the team was attempting to refine the neural network model using an MLP architecture developed previously by the data mining group.

The best result obtained for this problem with the MLP architecture was error prediction 1 hour in advance with a mean squared error (MSE) of just .00400.

The first implementation is due to Prof. Eric Wan, from the university of Oregon [7]. The author thanks him for granting permission to use the software.

The second implementation contains various additional neural networks models and is distributed as shareware. The best results were obtained with SNNS because of the versatility of the software.

Both implementations use temporal backpropagation as the backpropagation algorithm. In the future there may be attempts to use momentum backpropagation and backpropagation with weight decay to solve the same problem. At the present time, however, it was determined that the expected performance gain from these more complicated algorithms would be minimal while the overhead in terms of training time would be significant.

The next section presents the first attempts to train a TDNN for HMT prediction on both these implementations. Despite the overwhelming success of these attempts, it later turned out that the data had been incorrectly processed using the sliding window technique. These results were naturally reworked in the later attempts.

The next section's data was processed in the following way: the available minute-interval HMT data was averaged in groups of 20 to provide 20-minute average HMT data. Next, the sliding window technique was then used to attempt to turn these data points into hourly HMT data. This means that the first data point is the average of 20-minute average entries 1, 2 and 3, the second data point is the average of 20-minute

average entries 2, 3 and 4, and so forth. The validity of this processing is scrutinized in section 5.3.

6.2 First Attempts

The best results between the two TDNN implementations were obtained with the SNNS software. This was a result of much higher flexibility and a larger development effort deployed in the construction of SNNS. Several options for backpropagation, for example, were available in SNNS, while in the Matlab version it was difficult if not impossible to find the tools necessary for integrating any code that may be written to provide alternatives. The best result obtained with the Matlab software was an MSE of .00251 for the HMT one hour in advance.

After several trials with numerous network configurations, it was possible to reduce the testing data to a miniscule mean squared error (MSE) of .00106 for one hour ahead prediction. The result details are available in appendix D. The best MSE_s for the rest of the data were are included in Table 1 below.

Output HMT number of hours ahead	Best MSE
1 hour	.00106
2 hours	.00482
3 hours	.01248
4 hours	.01900
5 hours	.02909
6 hours	.02112
7 hours	.03396

Table 1: Best MSE results found.

These results were at first obtained with a one hidden layer architecture. The specific number of nodes in each layer was 16 for the input, 25 for the first hidden layer and 1 for the output. As the study moved towards 2 through 7 hours ahead prediction, it was found that a two hidden layer architecture worked best, specifically 16-6-2-1 and 16-4-2-1 architectures. (input-hidden-hidden-output, respectively)

As it will be shown in the next section, however, these results were invalid, mainly because of the sliding window technique used. The author was tipped off to this problem by an anomaly observed with the collected data. This anomaly consisted of the training data performing worse than the testing data.

6.3 Review of First Attempts

The sliding window concept involves averaging entries according to a "sliding" window of desired fixed size n in order to generate more data points. Thus, the first entry in the resulting data will be the average of entries 1 through n, the second will be the average of entries 2 through (n+1), and so forth.

It was believed at the time of the experiments that using a sliding window of n = 3 entries would allow generation of three times as much hourly data as would normally be possible.

However, because of the averaging involved, the data resulting from the sliding window processing is merely "smoothed out" 20 minute average data. This is because, on the average, the difference between consecutive entries is 20 minutes and not one hour.

For this reason, all the results mentioned in section 5.1 which claim to accomplish "x hour ahead hmt prediction" are actually "x/3 hour ahead average hmt prediction", with two caveats. Because of the sliding window technique, the resulting smoothed out data was more redundant than the 20 minute average data. Indeed, the sliding window technique only "smoothes out" the highs and lows of the 20 minute average data by averaging these extremes with their neighboring, less extreme, values. Put another way,

each point in the 20 minute graph is averaged in with the neighboring 2 points and scaled down to be closer to these two neighboring values.

As a result, the claim to have predicted average hmt in section 5.2 is a fallacy. The resulting network topology merely predicted the slided-average data with a window of n=3. It is unclear how this data relates to the desired actual or average hot metal temperature. To determine this relationship, further numerical analysis would be required. Even assuming this analysis is possible with good results, the solution to the problem would become unnecessarily complex and therefore harder to use by SteelCorp.

In addition, the sliding window technique is an invalid processing technique. By averaging 2, 3 or more values into one, the study is essentially processing the current hot metal temperature by using information from both the past and, more importantly, the future. Since what the author is trying to accomplish is future hmt prediction, this approach is clearly flawed because it indirectly uses what the author is trying to predict in order to predict it. During the trials this resulted in the training data performing worse than the testing data for 1 and 2 "hour" ahead prediction. (in reality "hour" = 20 mins)

Because this phenomenon was NOT present for 3-7 "hour" ahead prediction, one can directly point to the slided window processing of the data as the cause of the anomaly. The slided window of 3 results in successive entries (1 "hour" ahead prediction) having 67% (2 out of 3) of their data in common. This technique also results in entries separated by another entry (2 "hour" ahead prediction) having 33% (1 out of 3) of their data in common.

As can be seen from the appendix runs, the anomaly was very pronounced for 1 "hour" ahead prediction and only sometimes present for 2 "hour" ahead prediction. Once prediction moves on to 3 "hour" ahead forecasting, there is no more "sharing" of values to be averaged in and the problem disappears. (this can easily be checked, again, by examining the appendix runs for section 5.1)

Exactly what gives rise to this anomaly is a question still left to be answered, however. While the anomaly was clearly to some degree caused by the incorrect processing of the data, finding exactly how is a matter of speculation to some degree. Additional factors such as an unfavorable split of the training/testing sets should also be considered.

One possible cause could be the fact that less variation occurred in the hot metal temperature towards the end. (i.e. the 20% of the data used as the testing set) This factor may have made the hmt easier to predict in the testing set when compared to the training set, where the temperature is in the beginning driven to the desired levels. To test this cause, the data was processed into an even 50-50% split and the study alternated between using the first set as the training and the testing sets.

The results of this processing are shown in Table 2

Output HMT number of hours ahead	Best MSE (training first)	Best MSE (testing first)
1 hour	.00423	.00274
2 hours	.01406	.02056

Table 2: Best MSE results found.

As one can see from the runs in the appendix, these results presented no training data performing worse than testing data anomaly. Do these results let the sliding window technique "off the hook"? To a certain extent, yes. It would now seem like the data split was mainly to blame for the observed anomaly.

However, the sliding window technique, as seen previously, still remains a questionable practice in processing the data. This happens because of both the underlying uncertainty of the relevance of the results produced and the involvement of future values in prediction.

The observed anomaly is most probably due to the relative "unpredictability" of some data in the sets. With the initial 80/20 split, this data may have happened to be in the 80% set, thereby driving up this set_s MSE. With the subsequent 50/50 split, the data was split more evenly and there was no observed anomaly. The lesson learned here for the future is that a 50/50 or 60/40 split may be safer as far as the validity of results is concerned.

6.4 Final Results

Having gained a better perspective on the problem from the attempts described in section 5.2 and the subsequent analysis, it was proceeded to again predict average hourly hmt using the SNNS software. New data was obtained from SteelCorp upon request, since there was not enough old data for the intended purpose of this study. The larger size of the new data ensured the TDNN topology developed met the fifth criterion for effective spatio-temporal pattern prediction discussed by Waibel [4]. Indeed, to revisit this point, the training set needs to be large enough compared to the size of the network. This forces the network to learn the representation that is the best at predicting the spatio-temporal pattern studied.

The new data was pre-processed by averaging 12 5-minute apart data points into one point. As can easily be seen this yielded hourly hmt averages. Again, an 80% / 20% split was employed to divide the data into a training and a testing set.

This pre-processing allows the TDNN architecture developed to approximate most of the requirements mentioned by Waibel in [4].

The best mean-squared error results are presented in Table 3 below. Note that the study tested with both using the current HMT and leaving it out. Better results were obtained using the current HMT.

Output HMT number of hours ahead	Best MSE (with HMT)	Best MSE (w/o HMT)
1 hour	.01603	.02322
2 hours	.02635	.03023
3 hours	.02779	.03328
4 hours	.03405	.04149
5 hours	.03978	.04278
6 hours	.04194	.04451
7 hours	.04202	.04643

Table 3: Best MSE results.

As it can easily be seen, these values do not compare favorably with those obtained using the sliding window technique. One must keep in mind, however, that the values predicted by the TDNN using lagged data were also lagged values.

The trade-off between the two alternative approaches seems clear: using the same architecture, one can either more accurately predict less relevant information (lagged hourly HMT values) or can less accurately predict more relevant information (average hourly HMT values). Because of the difficulty of interpreting the lagged values, one can arguably conclude that this section's results are the more valid results of the study.

An issue encountered was whether or not the current HMT should be used in predicting the HMT x hours ahead. A more detailed discussion of this choice is done in [2]. It was decided to go ahead and use the current HMT, because prediction without this

parameter would be rather difficult. Indeed, it could make all the difference in the world to know the current temperature in certain situations. For example, one needs the TDNN that solves the problem to differentiate between the following two scenarios:

Scenario 1: Heating up the furnace is in progress. Temperature could be anywhere between 0 and 1450 degrees Celsius.

Scenario 2: The furnace has been operating for a long period of time. Temperature is in range 1450-1500 degrees Celsius.

In order to do so, one clearly needs to know the current HMT as none of the other 17 input parameters will help. Aside from further trying to perfect the layer structure used, there is one other improvement which could be tried in the future. Input sensitivity detection could be important to this application. This method is detailed in Section 5.4 and its results analyzed in Section 6.5.

6.5 Other Results in the Data Mining Group

During the course of this study, other techniques were also employed to solve the same problem. These techniques included another neural network effort involving MLPs/FFNNs, a genetic algorithm and an inversion problem attempt [60, 61, 62]. A more extensive analysis of the TDNN results was also performed [59]. This analysis includes

data for normalized mean squared error (NMSE), average error (AE) and Pearson coefficient correlation between current and future HMT.

This section will not attempt to reestablish the analysis involving these statistics. A lengthy discussion can be found in [59]. To summarize, the NMSE and AE numbers were somewhat inconclusive. The Pearson coefficient correlation numbers supported the conjecture that as the HMT prediction interval is increased, there is less dependency between current and predicted HMT [59].

In [60], Sarma details another neural network effort on the same problem using MLPs (referred to as FFNNs in the paper). The numbers for one through eight hours ahead prediction are reproduced in Table 4 below.

Output HMT number of hours ahead	**Best MSE (with TDNN)**	**Best MSE (with MLP)**
1 hour	.01603	.0138
2 hours	.02635	.0147
3 hours	.02779	.0180
4 hours	.03405	.0148
5 hours	.03978	.0172
6 hours	.04194	.0145
7 hours	.04202	.0188

Table 4: Best MSE results – TDNN vs. MLP

These results are somewhat better than the MSE numbers obtained for average HMT prediction using TDNNs in section 5.4. It is likely that the TDNN approach used to obtain the results in section 5.4 can be further refined to match these results, since TDNNs initially outperformed the MLPs on time-delayed data.

While the genetic algorithm for Lai [61] did not converge, this paper contains a very thorough analysis of steel work in general. Also included is a good study of what constitutes a good artificial intelligence technique to be applied to the problem at hand. [62] details an attempt at applying the inversion problem technique to the SteelCorp problem. In this paper, Dua lays a theoretical foundation for the still developing field of inversion as it applies to SteelCorp's problem.

The data mining group at MIT, under the supervision of Dr. Amar Gupta, has performed extensive work on the problem presented by SteelCorp. Several approaches have been employed and the solutions obtained more than satisfy the sponsor's need. Future work in this domain may involve applying more exotic neural network architectures such as recursive neural networks (RNNs) and adaptive time-delayed neural networks (ATNNs). An overview of the latter is presented in the previous section, where the former is also summarily described.

7 CONCLUSIONS

As seen in the results section, the performance of the neural network developed with an MLP architecture for the sliding window data prediction was surpassed. However, that processing done with slided data invalidates the corresponding results as shown in chapter 5. Despite the drawback, this series of experiments reinforces the fact that in general TDNN architectures have a much greater affinity for temporal patterns, both short- and long-term.

The results obtained were great for slided window one hour ahead prediction and reasonable for two hour ahead prediction. The failure of obtaining a good model for three or more hours ahead prediction can be attributed to one of three factors. This study's conclusion is that either not enough data was available, the given data did not exhibit the correlation required for prediction or, finally, more complicated neural network models (RNNs or ATNNs) should be run in order to exploit the existing correlation.

The average HMT prediction results detailed in section 5.4 are somewhat less remarcable. The best MSE obtained was .0160 for one hour ahead prediction. This value comes close to .0130 -- the best MSE value obtained in [60] with MLPs. Further work will be needed in this domain to refine the TDNN model described in this paper.

This paper also presents a survey of the TDNN computational field. This includes a theoretical background and new computational advances in this field, advances that were both intrinsic to TDNNs and using TDNNs as computational black boxes.

The neural network received some very positive results. In the future, it is hoped to perfect these results with both TDNNs and other data mining tools. Among the latter, the inversion problem tools appear very promising. Solving the inversion problem for the hot metal temperature in the furnace would allow one not only to predict what the temperature will be in advance, but also would also tell what inputs one needs to change and by how much in order to obtain the desired output temperature.

8 REFERENCES

[1] Sarma, Ravindra, "Using Neural Networks to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel", Advanced Undergraduate Research Project, MIT, scanner-group.mit.edu, May 1999.

[2] Ge, Angela, "A Neural Network Approach to the Modeling of Blast Furnace", Masters of Engineering Thesis, MIT, scanner-group.mit.edu, 1997.

[3] Bansal, Vadhavakar, Gupta, "Neural Networks Based Data Mining Applications for Medical Inventory Problems", MIT, 1992.

[4] Waibel et. all "Phoneme Recognition Using Time-Delay Neural Networks", IEEE Trans, Accoust., Speech, Signal Processing, vol. 37, no 3, pp.328-339, 1989

[5] S. Haykin, "Neural Networks: A Comprehensive Foundation", Englewood Cliffs, NJ: Prentice-Hall, 1998

[6] Zell A., et. al. Stuttgart Neural Netowrk Simulator User Manual, Version 4.1, report No. 6, 1995.

[7] Prof. Eric Wan's MLP with FIR software, Univ. of Oregon.

[8] Patrick Winston, "Artificial Intelligence" 3rd edition, Addison-Wesley Publishing Company, Reading, MA, 1991.

[9] J. Hertz, A. Krogh, R. Palmer - "Introduction to the Theory of Neural Computation", Addison-Wesley Publishing Company, Redwood City, CA, 1991.

[10] McClelland and Elman "Interactive Processes in Speech Perception: The TRACE Model", Parallel Distributed Processing, vol. 2, chap. 15.

[11] Elman and Zipser, "Learning the Hidden Structure of Speech", Journal of the Acoustical Society of America 83, 1988, pp. 1615-1626.

[12] Tank and Hopfield, "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", IEEE Transactions on Circuits and Systems 33, 1986, pg. 533-541.

[13] E. Saad, D. Prokhorov, D. Wunsch - "Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks", IEEE Transactions on Neural Networks, vol. 9, november 1998.

[14] SNNS website - http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html

[15] Fine, Terrence "Feedforward neural Network Methodology", Springer, New York, 1999.

[16] Amaldi, E. [1994], "From Finding Maximum Feasible Subsystems of Linear Systems to Feedforward Neural Network Design, Ph.D. disseration, Dept. of Mathematics, Ecole Polytechnique Federale de lausanne, Lausanne, Switzerland.

[17] Blum, A., R. Rivest [1989], "Training a 3-node neural network is NP-complete", Advances in Neural Information Processing Systems 1, Morgan Kaufmann Pub., 494-501.

[18] EDUPAGE@LISTSERV.EDUCAUSE.EDU, Mar 27th 2000 Newsletter.

[19] H. Tan, D. Prokhorov, and D. Wunsch, "Probabilistic and time-delay neural-network techniques for conservative short-term stock trend predition," in Proceedings of the World Congress on Neural Netowrks, San Diego, CA, June 1994.

[20] E. Saad, D. Prokhorov, D. Wunsch - "Comparative Study of Stock Trend Prediction Using Time Delay, Recurrent and Probabilistic Neural Networks", IEEE Transactions on Neural Networks, vol. 9, november 1998.

[21] Prospectives in Neural Computing, Proceedings of the 8th International Conference on Artificial Neural Networks, Skovde, Sweden, sept 1998, Vols I and II.

[22] [editor] Simpson, Patrick K., "Neural Networks Applications", IEEE Technology Series, IEEE Inc., New York, NY, 1996.

[23] Clouse, Daniel S., Giles, Lee C. "Time-Delay Neural Networks: Representation and Induction of Finite-State Machines", IEEE Home, www.ieee.com/ielonline, 1994.

[24] Sipser, Mike "Introduction to the Theory of Computation, PWS Publishing Company, New York, NY 1999.

[25] A. Esposito, C. E. Ezin, M. Ceccarelli, "Preprocessing and Neural Classification of English Stop Consonants" International Institute for Advanced Scientific Studies, 1995.

[26] Lavagetto, Fabio, "Time-Delay Neural Networks for Estimating Lip Movements From Speech Analysis: A Useful Tool in Audio-Video Synchronization.", IEEE Transactions on Circuits and Systems for Video Technology, Vol 7, No. 5, October 1997.

[27] Liang, Shin-Hwei, Lin, Chin-Teng, "Feature-Based Visual Speech Recognition Using Time-Delay Neural Network", IEEE Home, www.ieee.com/ielonline, 1995.

[28] Lee, Luan Ling, "Neural Approaches for Human Signature Verification", IEEE home, www.ieee.com/ielonline, published 1995.

[29] Sakr, Majd F. et. all, "Performance of On-Line Learning Methods in Predicting Multiprocessor Memory Access Patterns", Technical Report, UMIACS-TR-96-59 and CS-TR-3676, Institute for Advanced Computer Studies, University of Maryland, 1996.

[30] Poo, Gee-Swee, "A Two-Level TDNN (TLTDNN) Technique for Large Vocabulary Mandarin FINAL Recognition", IEEE home, www.ieee.com/ielonline, 1994.

[31] Bodenhausen, Ulrich, Waibel, Alex, "Learning the Architecture of Neural Networks for Speech Recognition", IEEE home, www.ieee.com/ielonline, 1991.

[32] Lin, D-T., Dayhoff, J.E., Ligomenides, P.A. "A Learning Algorithm for Adaptive Time-Delays in a Temporal Neural Network", Technical Report, University of Maryland, 1992.

[33] Hild, Hermann; Waibel, Alex "Recognition of Spelled Names Over The Telephone", IEEE Home, www.ieee.com/ielonline, 1991.

[34] Chen, Casper K.; Chiueh, Tzi-Dar; Chen, Hyh-Horng "Active Cancellation System of Acoustic Noise in MR Imaging", IEEE Home, www.ieee.com/ielonline, 1999.

[35] Driancourt, Xavier; Bottou, Leon; Gallinari, Patrick "Learning Vector Quantization, Multi Layer Perceptron and Dynamic Programming: Comparison and Cooperation", IEEE home, www.ieee.com/ielonline, 1991.

[36] Islam, Rafiqul et. all, "Phoneme Recognition Using Modified TDNN and a Self-Organizing Clustering Network", IEEE Home, www.ieee.com/ielonline, 1995.

[37] Engelbrecht, A.P.; Zurada, J.M.; Cloete, I. "Determining the Significance of Input Parameters Using Sensitivity Analysis", University of Stellenbosch, South Africa, 1994.

[38] Sung, A.H., "Ranking Importance of Input Parameters of Neural Networks", Elsevier Science Ltd., 1998.

[39] Nath, Ravinder; Rajagopalan, Balaji; Ryker, Randy; "Determining the Saliency of Input Variables in Neural Network Classifiers", Elsevier Science Ltd., 1997.

[40] Minghu, Jiang et. all "Fast Learning Algorithms for Time-Delay Neural Networks Phoneme Recognition", Proceedings of ICSP 1998.

[41] Vogt, M.; Schumacher, W.; "How the Problem Size Influences the Performance of an SIMD TDNN Implementation", IEEE Home, www.ieee.com/ielonline, 1994.

[42] Vo, Minh Tue, "Incremental Learning Using the Time Delay Neural Network", IEEE Home, www.ieee.com/ielonline, 1994.

[43] Back, Andrew; Tsoi, Ah Chung; Horne, Bill; Giles, C.Lee; "Alternative Discrete-Time Operators and Their Application to Nonlinear Models", Technical Report, University of Maryland, 1997.

[44] J.C. Principe, B. de Vries, and P. Guedes de Oliveira "The Gamma filter - a new class of adaptive IIR filters with restricted feedback", IEEE Transactions on Signal Processing, 41:649-656, 1993.

[45] J.C. Principe et. All, "Modeling applications with the focused gamma net." In J.E. Moody, S.J. Hanson, and R.P. Lippman, editors, Advances in Neural Information Processing Systems, vol. 4, pg. 143-150, San Mateo, CA, 1992, Morgan Kaufmann.

[46] Back, Andrew et. all, "Low Sensitivity Time Delay Neural Networks with Cascade Form Structure", IEEE Home, www.ieee.com/ielonline, 1997.

[47] Campolucci, Paolo et all., "A New IIR-MLP Learning Algorithm for On-Line Signal Processing", IEEE Home, www.ieee.com/ielonline, 1997.

[48] Selouani, S.A. and Caelen, Jean "Arabic Phonetic Features Recognition using Modular Connectionist Architectures", IEEE Home, www.ieee.com/ielonline, 1998.

[49] Poddar, P. and Rao, P.V.S. "Temporal Sequence Classification by Memory Neuraon Networks", Neural Networks Applications, IEEE Technological Update Series, New York, NY 1996.

[50] Dugast, C.; Devillers, L., "Incorporating acoustic-phonetic knowledge in hybrid TDNN/HMM frameworks", Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on, Volume: 1 , 1992 , Page(s): 421 -424 vol.1

[51] Ma, W.; Van Compernolle, D., "TDNN labeling for a HMM recognizer", Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990, International Conference on , 1990 , Page(s): 421 -423 vol.1

[52] Dugast, C.; Devillers, L.; Aubert, X., "Combining TDNN and HMM in a hybrid system for improved continuous-speech recognition", Speech and Audio Processing, IEEE Transactions on, Volume: 2 1 2 , Jan. 1994 , Page(s): 217 -223

[53] http://scanner-group.mit.edu/, Knowledge Acquisition, Knowledge Discovery and Knowledge Dissemination Research at PROFIT web site, the web site of Dr. Gupta's datamining group.

[54] Mallick S.K. Basu, "Automation in Blast Furnace: Scope and Recent Trends". Tata Search 1998. Pp 89-96.

[55] Singh, H., Sridhar, N.V., and Deo, B. "Artificial neural nets for prediction of silicon content of blast furnace hot metal". Steel Research 67 (1996) No. 12. 117-120.

[56] SNNS website - http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html

[57] Biswas A.K., "Principles of Blast Furnace Ironmaking", SBA Publications, Calcutta, 1984, pp. 309.

[58] Michael J. A. Berry, Gordon Linoff. "Data Mining Techniques", Wiley Computer Publishing. 1997

[59] Purville, Brian, "Using TDNNs to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel", Advanced Undergraduate Research Project, MIT, scanner-group.mit.edu, May 2000.

[60] Sarma, Ravindra, "Using Neural Networks to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel", Masters of Engineering Thesis, MIT, scanner-group.mit.edu, May 2000.

[61] Lai, Danny, "Using Neural Networks to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel", Advanced Undergraduate Research Project, MIT, scanner-group.mit.edu, May 2000.

[62] Dua, Ashima, "Using Inversion to Predict the Hot Metal Temperature of the "G" Blast Furnace at Tata Steel", Advanced Undergraduate Research Project, MIT, scanner-group.mit.edu, May 2000.

APPENDIX A : TDNN INPUTS AND OUTPUTS

The 16 inputs used in the network:

Input 1 : Cast

Input 2 : Cast1op2

Input 3 : Cast2op2

Input 4 : Chargetime

Input 5 : Coalinjection

Input 6 : G2heatflux

Input 7 : Grp1HeatFlux

Input 8 : HotBlastTempPV

Input 9 : Orebycok

Input 10: Oxygenflow

Input 11: Steamflow

Input 12: TopgasCO

Input 13: TopgasCO2

Input 14: TopgasH2

Input 15: WindFlow

Input 16: current hot metal temperature

Output:

Future hot metal temperature

APPENDIX B : BEST 1 HOUR TDNN NET

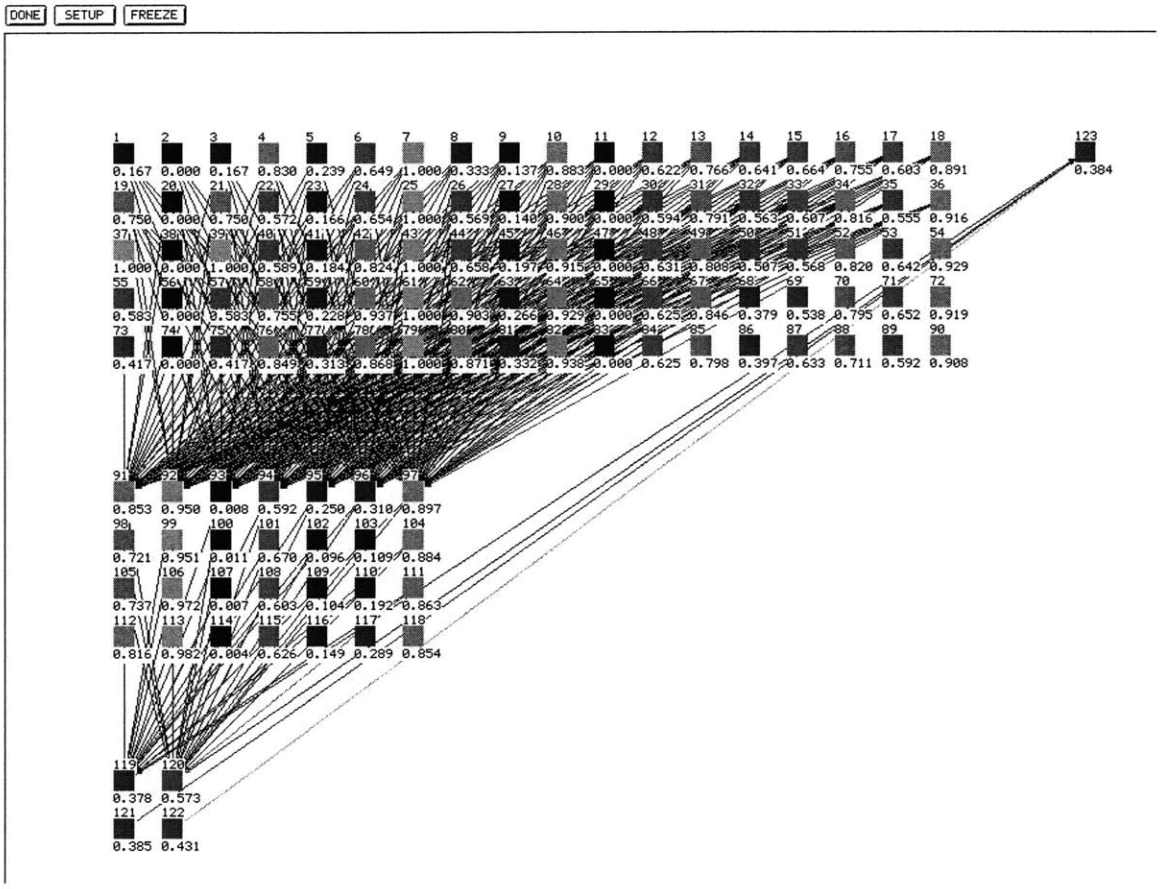


Figure 12: Trained 4-layer TDNN. The numbers below each node represent the respective weights.