TIME SERIES CLASSIFICATION USING DEEP LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

POYRAZ UMUT HATIPOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

AUGUST 2016

Approval of the thesis:

TIME SERIES CLASSIFICATION USING DEEP LEARNING

submitted by **POYRAZ UMUT HATIPOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Murat Köksalan _____
Head of Department, **Industrial Engineering**

Assoc. Prof. Dr. Cem İyigün _____
Supervisor, **Industrial Engineering Department, METU**

Examining Committee Members:

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Cem İyigün
Industrial Engineering Department, METU _____

Assoc. Prof. Dr. Serhan Duran
Industrial Engineering Department, METU _____

Assist. Prof. Dr. Bahar Çavdar
Industrial Engineering Department, METU _____

Assist. Prof. Dr. Öznur Taştan Okan
Computer Engineering Department, Bilkent University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: POYRAZ UMUT HATIPOĞLU

Signature :

ABSTRACT

TIME SERIES CLASSIFICATION USING DEEP LEARNING

Hatipoğlu, Poyraz Umut
M.S., Department of Industrial Engineering
Supervisor : Assoc. Prof. Dr. Cem İyigün

August 2016, 115 pages

Deep learning is a fast-growing and interesting field due to the need to represent statistical data in a more complex and abstract way. Development in the processors and graphics processing unit technology effects undeniably that the deep networks get that popularity.

The main purpose of this work is to develop robust and full functional time series classification method. To achieve this intent a deep learning based novel methods are proposed. Because time series data can have complex and variable structure, it may be more suitable to use algorithms that can handle the nonlinear sophisticated operations rather than shallow-structured methods. While shallow structured methods need handcrafted features and expert knowledge about data, deep learning based algorithms are capable of working with raw features. Both deep belief network and stacked autoencoders based architectures are constructed and trained for the dataset gathered from different researches areas. In time series classification, even though dynamic time warping and nearest neighbor based methods are hard to beat, many classification methods have been studied recently. To examine the performance of proposed method comparative analysis is conducted with popular benchmark methods. Despite higher accuracy in the results, the deep learning based methods cannot outperform superiorly.

Keywords: Deep Learning, Time Series, Deep Belief Networks, Stacked Autoencoders

ÖZ

ZAMAN SERİLERİNİN DERİN ÖĞRENME İLE SINIFLANDIRILMASI

Hatipoğlu, Poyraz Umut
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Cem İyigün

Ağustos 2016 , 115 sayfa

İstatistiksel verilerin daha karmaşık ve soyut bir şekilde temsil edilme ihtiyacıyla birlikte derin öğrenme tekniği hızla gelişen ve ilgilenilen bir alan konumuna gelmiştir. İşlemci ve grafik işleme ünitesi alanlarındaki gelişmelerin derin öğrenmenin rağbet kazanmasına katkısı yadsınamaz. Bu çalışmanın asıl amacı gürbüz ve tam işlevsel zaman serileri sınıflandırma metodu geliştirmektir. Bu amaca ulaşmak için derin öğrenme tabanlı özgün metotlar önerilmiştir. Zaman serisi verileri karışık ve değişken yapıya sahip olduğundan basit yapıya sahip olan yöntemlerden ziyade doğrusal olmayan karışık işlem yeteneklerine sahip metotlar ile çalışmak daha anlamlı ve uygun olabilmektedir. Basit yapıya sahip yöntemler el ile ayarlanmış özniteliklere ve uzman bilgi birikimine ihtiyaç duyarken derin öğrenme tabanlı algoritmalar ham öznitelikler ile çalışabilme yeteneğine sahiptir. Çalışma kapsamında birçok araştırma alanından elde edilmiş veriler için hem derin anlama ağları hem de yığılı oto-kodlayıcı tabanlı mimariler kurulup eğitildi. Zaman serileri sınıflandırma çalışmalarında dinamik zaman bükme ve en yakın komşu temelli yöntemleri performans açısından yenmek zor olmasına rağmen farklı sınıflandırma yöntemleri ile yakın geçmişte çalışmalar yapılmıştır. Önerilen yöntemin performansını ölçümlemek için rağbet gören kıyaslama yöntemleri ile karşılaştırmalı analizler yapıldı. Daha yüksek başarım oranı elde edilmesine rağmen derin öğrenme tabanlı metot sonuçlarında çok üstün bir performans aşımı gözlemlenememiştir.

Anahtar Kelimeler: Derin Öğrenme, Zaman Serileri, Derin Anlama Ağları, Yığılı Oto-Kodlayıcı

To my family...

ACKNOWLEDGMENTS

Firstly, I wish to express my sincere gratitude to my thesis advisor Assoc. Dr. Cem İyigün for his consistent and kind support, guidance and encouragement. I am absolutely sure that without his guidance, continued help, and insights this this thesis couldn't have been completed.

I would also like to extend my appreciation to members of METU Industrial Engineering Department for all of their support and endless trust within the last three years.

My special thanks are extended to my mentors and colleagues in my professional life in both METU Electrical and Electronics Engineering Department and HAVELSAN INC for their guidance and accompaniment.

Last but not least, I would like to thank my dear friends for their social accompaniment, and for supporting me in every aspect on my life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

SVM	Support Vector Machine
PCA	Principal Component Analysis
NN	Neural Networks
AI	Artificial Intelligence
GMM	Gaussian Mixture Model
MLP	Multi-Layer Perceptron
DL	Deep Learning
DNN	Deep Neural Networks
GPU	Graphical Processing Units
ANN	Artificial Neural Networks
DBN	Deep Belief Networks
DBM	Deep Boltzmann Machines
LSTM	Long Short Time Memory
AE	Autoencoder
SAE	Stacked Autoencoder
RBM	Restricted Boltzmann Machines
KPCA	Kernel Principal Component Analysis
DTW	Dynamic Time Warping
DDTW	Derivative Dynamic Time Warping
WDTW	Weighted Dynamic Time Warping
KNN	k-Nearest Neighbors
ED	Euclidean Distance
FNN	Feed-Forward Neural Networks
RNN	Recurrent Neural Networks
LMS	Least Mean Square
SRN	Simple Recurrent Network
MKM	Multilayer Kernel Machines
KKT	Karush Kuhn Tucker

PSD	Positive Semi-Definite
RBF	Radial Basis Function
SDK	Software Development Kit
MEX	MATLAB Executable
LIBSVM	Library for Support Vector Machines

CHAPTER 1

INTRODUCTION

1.1 Motivation, Scope and Goal

Statistical data analysis and machine learning are popularly studied topics in the literature and have strong connections with many fields. The main goal of machine learning is detecting, sensing and learning phenomena as human do or better than the human in some aspects. While scientist and researchers have been trying to accomplish to this purpose, they have used simple and complex decision models. Less human interaction and higher performance are the key points in machine learning studies. For basic and simpler algorithms (or models), a good and simpler representation of data can be a necessity. Unfortunately without the expert knowledge of data and the model this simplified representation can be turned into a bigger and challenging problem. In such cases, it is more appropriate to design a model which is able to learn the characteristics of data either by extracting abstract features (or attributes) or transforming the data to more compact form. First, the neural networks (NN) then the deep learning have been trying to respond to that need. The simplest form of deep architecture, perceptron, was defined in 1950s. Then the multilayer structures with limited learning capabilities ware proposed. For over 40 years, these architectures have attempted to be evolved on specific demands. Not only the need for the representation of data in more abstract and complex way, but also the advances in information technology, machine learning and signal processing increased the importance of deep networks.

Time series having unique properties that have been studied for a long time. The

time interval of point of a time series should be continuous and have at most one observation in time. During the data gathering procedure time intervals between the observations should be fixed and constant. These special properties and time dependency issues specialize the scope of the classifications problem. Our aim in this study to classify the datasets with higher performance than the benchmark methods. Moreover, for possible future reverse engineering works, the observed weight will be an opportunity.

1.2 Contribution

In this study, novel time series classification methods (or classifiers) are proposed and analyzed in a detailed way. Except for a few studies, time series classification using deep architectures has not been the area of focus in the statistical data analysis. Instead of extracting hand-crafted features, deep learning models are designed to be capable of working with raw representation of time series. Training and test procedures of the proposed method are completed without making any prior assumptions.

While conducting our analysis for four different dataset gathered from different domains are examined. These publicly available datasets have been selected in the UCR database [16]. To complete Dynamic Time Warping (DTW) tests, we need pre-calculated parameters due to time and resource constraints. While selecting the datasets, we should also be careful whether the datasets have sufficient data points to be trained by deep learning algorithms. With respect to our criteria we have narrowed our search down and got these four datasets.

In our study we investigated different aspects of this problem, including the dimensionality and the size of the data and the limitations introduced by these aspects. By monitoring and visualizing the weights between the layers of deep learning methods compact representation of unsupervised learning stage can be apprehended. Unlike most of the previous classification applications using same datasets, imbalanced dataset effects are also examined which is another contribution of our work.

1.3 Outline

This thesis document, which focuses on the time series classification using deep learning, is divided into 6 main chapters:

- Introduction
- Time Series and Learning
- Benchmark Methods Used in Time Series Classification
- Proposed Method
- Experimental Analysis and Results of the Proposed Method
- Conclusion

The first chapter is designed to reveal the dissertation topic and purpose of the specified study. In Chapter 2 time series data and learning methods in the literature are conferred. Theoretical explanations and mathematical derivations of the commonly used benchmark methods for time series classification are presented in chapter 3. Fourth chapter explains the proposed method from the very basic introduction to the final complex structure. This chapter includes also the literature survey for proposed method in a detailed way. In chapter 5, experimental procedures and comparative results of proposed method and benchmark methods are presented. Detailed discussions about the study can also be monitored in this part of thesis. Lastly, in chapter 6 works conducted during the dissertation research is concluded and feature research directions are presented.

CHAPTER 2

TIME SERIES AND LEARNING

2.1 Time Series

Time series is a special kind of data which has ordered sequence. Ordering property is crucial since it effects dependency of data points and meaning of data. Data points forming the time series have some more pre-defined properties. One of the properties is that data points need to be obtained through repeated measurements over time at equally spaced intervals. The time interval of data points should be continuous and each time unit observations should have at most one data point.

Time series are used in various areas, such as statistics, economics, pattern recognition, control engineering, signal processing, astronomy, meteorology, entertainment and so on. Time series analysis has been developing and trending research areas for decades. Despite the progress, there are a lot of open topics about time series.

Noise in time series is one of the main challenging problems and causes inaccurate prediction [64]. Noisy, high dimensional and complex time series data cannot be modelled with traditional shallow methods which have limited non-linear operation ability. So as to reduce the dimension of data and remove the noise, various dimensionality reduction techniques (i.e. principle component analysis (PCA), independent component analysis) can be applied to data. However, these procedures often require expert knowledge and can lose the informative parts of the data.

Another difficulty or tradeoff is related to the number of data points in the time series. More data from the past increase the precision and the chance of detailed analysis.

However while increasing the precision property of the data, we also increase the risk that the model cannot handle the data processing.

When analyzing the time series previously mentioned unique properties should be taken in consideration. In order to reduce challenge in analyzing data, features may need to be transformed into invariant feature space. [41]

Finally, obtaining large amounts of labelled time series training data may be expensive, resourceful and difficult. Whereas, huge amount of unlabelled data can be easily obtained in various areas. Hence, current shallow-structured methods needing large amount of labelled training data cannot be used for most of the time series data. Deep learning networks using unsupervised learning have gotten highly successful results. However, so as to get more accurate results the architecture of the model should be adjusted or modified respecting the characteristics of time-series. Sample time series observations taken from different datasets are shown in Figure 2.1 .

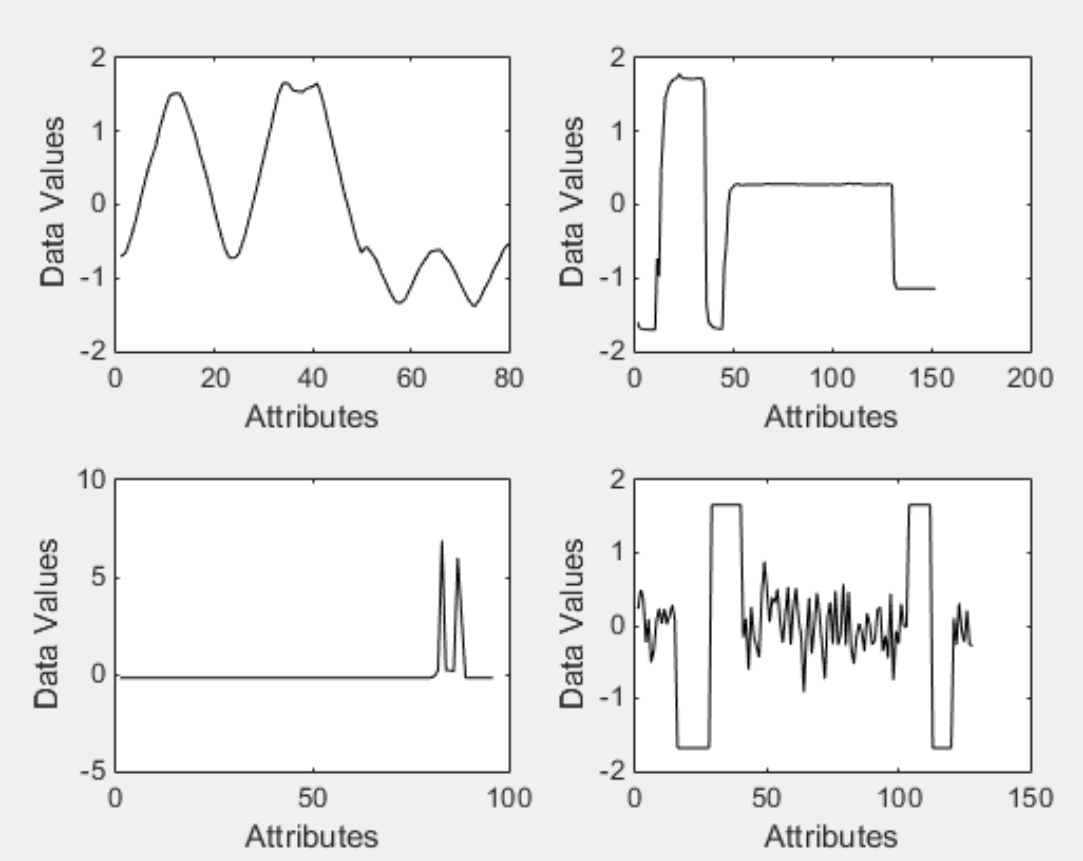


Figure 2.1: Sample time series observations taken from different datasets

2.2 Learning

Any approaches or methods that take information from training samples to design a classifier can be called learning. Supervised, unsupervised and reinforced learning are the general forms of machine learning. Before explaining the learning types, the data types should be first distinguished clearly. Learning can be seen as a special technique to reduce the error on a training set.

Labelled and unlabelled data are treated differently with respect to the scope of applications. While labelled data are commonly used to predict or estimate the target attributes (or class labels) of new observations (or samples, or data points), unlabelled data are useful for the clustering or investigating associations in data. Attributes of the data excluding the label information are often called features (or attributes) in machine learning.

2.2.1 Supervised Learning

In supervised learning, data with prior target information are provided for training, and there is an effort to reduce the sum of cost of prediction or the number of errors. Supervised learning algorithms evaluate the training data for developing a rule, produce a model to map new data samples. Once the learning model is trained, it can then classify or estimate the target attribute (or class label) of test data (or samples). If the estimated target attributes are labels, the supervised learning task is called classification. However, if the predicted attribute is a continuous numerical value, the algorithm is named as regression or prediction. In both classification and regression problems, the algorithms search an appropriate function to minimize cost of prediction.

2.2.2 Unsupervised Learning

If class labels of training data are not available, the learning process is called unsupervised learning. Main purpose of unsupervised learning is finding out or underlying similarities, grouping the training samples or detecting the association rules between

data points. The grouping procedure is often known as clustering task.

In the clustering task, one of the critical issues is to define similarities between two data points. The other one is defining distance measures. Last but not least important one is finding the number of clusters and/or determining boundaries of clusters.

2.2.3 Type of Variables and Data Preparation

Input variables define the measurement types used in the learning methods; some of the measurement types are suitable for qualitative input variables whereas some of them are favorable for quantitative inputs. Not only the measurement types but also the model itself is defined for the type of the input attributes. In order to use models that are invented for qualitative variables, grouping or binning the quantitative inputs is mandatory.

Variables, which are also known as features or attributes, are categorized into four main types as nominal, ordinal, interval, and ratio variables.

Nominal variables and ordinal variables are called categorical variables. Nominal variables provide descriptive information labels to distinguish one object from another, e.g. red, green, blue. Ordinal variables are similar to the nominal variables, however ordinal variables have a meaningful order and can be arranged in that order, e.g. low, medium, high. Ranges between the ordinal variables may not be equally spaced in ordinal variables.

Interval variables take numerical values which are equally spaced, such as temperatures scale or calendar dates. But the interval variables are not suitable for the proportional calculation. The ratios between the interval variables are not meaningful. Ratio variables are similar to interval variables except that the ratio calculations are meaningful and the origin or zero value represents the absence of measured characteristic.

During the data gathering process some faults can be observed and this failure can negatively affect the learning process. Hence obvious outlier or missing values should be discarded or readjusted before the learning operations to increase performance

or reduce the effort. In addition to that, features within different dynamic ranges should be normalized or standardized on demand to equalize the influences in the cost function.

2.2.4 Training-Validation-Test Set Separation

Random division the whole dataset into training, validation and testing set is very common in statistically data analysis. First, the training data is used to fit the supervised learning model. Then prediction errors are estimated by using the validation set to choose the convenient model. After that, the test set is used to check the generalization of error for the selected and tuned final model.

There is no predefined and general rule for the way of partitioning the dataset into training, validation and test parts, but training sample size is the most determining factor for supervised learning.

2.2.5 Curse of Dimensionality

In machine learning, we have to deal with high dimensional feature space which means many input attributes for each observation (or sample). This phenomenon can reveal some serious complications and affect negatively the design of the learning task.

Little or nothing in the way of data reduction is provided, which leads to severe requirements for computation time and storage. With sufficient observation samples, some of the dimensionality problems can be overcome, but not all of them. Besides, the number of samples required may be very large. The demand for samples can increase exponentially or power law growth with the high-dimensional feature space. This serious difficulty is often called the curse of dimensionality [6].

One of the key reasons for the curse of dimensionality is that high-dimensional calculations have more tendency to the potential computational and storage problems than low-dimensional ones. The other one is that it creates noise and hides the real patterns and makes the classification problem more intractable.

2.2.6 Generalization and Overfitting

The ability of classification of test data, which have not been seen by the model yet, is called generalization. Generalization with small error rate is a main aim of all kind of learning methods. Generalization property of a learning method can be checked by validating the model with different and separate test sets.

While trying to reduce training set classification/regression errors, complexity of the classifier should be adjusted by taking generalization issue into consideration.

Over trained complex models may perform perfect classification on the training set during learning process, but not on the new data. This phenomenon is known as overfitting. There is quite important tradeoff between highly complex structure that tends to overfitting and simple structure producing poor classification result on novel observation samples. Therefore this problem is one of the important research challenge in machine learning and pattern recognition.

2.2.7 Dimension Reduction

As mentioned earlier, one of the major problem is curse of dimensionality in machine learning and pattern recognition. Moreover, limited resource and computational complexity are the major concern in these areas. Hence, dimensionality reduction also known as feature reduction methods are proposed. To design and tune a successful classifier with better generalization ability, reducing the number of dimensions is also critical, especially when limited number of samples (or observation, data points) exist for training process. Besides, noise elimination and outlier detection are other positive capabilities of the feature reduction methods.

Most of the dimensionality reduction techniques provide a functional mapping. For example principle component analysis, which is a popular dimensionality reduction method, is used due to linear mapping ability. Principle component analysis tries to find a lower independent dimensional representation of original data with respect to the variance values of the feature space.

CHAPTER 3

BENCHMARK METHODS USED IN TIME SERIES CLASSIFICATION

In this chapter, theoretical explanations and mathematical derivations of the commonly used benchmark methods for time series classification are presented. Nearest Neighbor algorithm using Dynamic Time Warping distance measures and Multi-Class Support Vector Machines are examined in the following sections.

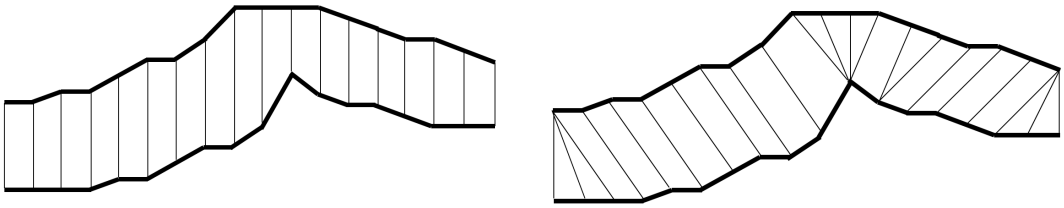
3.1 Dynamic Time Warping with Nearest Neighbor Classifier

3.1.1 Dynamic Time Warping

Dynamic Time Warping (DTW) [49] is a commonly used distance measure for time series clustering and classification applications. The working principle of this method is finding the minimum cost path between two time series by providing nonlinear alignments using dynamic programming. In other words, DTW tries to align time series by warping them in a non-linear way so that pairwise distance is minimized. Lock-step distance measures like Euclidean distance compare same sequenced points of time series. However elastic distance measures like DTW allow comparison of one data point of a time series to many data points of the other time series. Lock-step distance measure and elastic distance measure illustrations can be seen from Figure 3.1 .

DTW was invented to increase the performance of automatic speech recognition originally but researchers developed and adapted the technique and its variants to their

research areas such as manufacturing, medicine, robotics and gesture recognition, see [26, 14, 51, 25], for more details.



(a) Lock-step distance measure (b) Elastic distance measure

Figure 3.1: Distance measures type illustration

Warping window is a parameter used to optimize the dynamic time warping classifier. Warping window, shown as the percentage of the time series length, defines the maximum allowable phase difference between a reference point and a testing point. While restricting the path, it also helps to reduce the computational load of DTW. Dynamic Time Warping with warping window size, Derivative Dynamic Time Warping (DDTW) [38] and weighted dynamic time warping (WDTW) [34] are some of the popular modified versions of DTW.

More complex DTW-based similarity measures like LCSS [58], Swale [44], SpADe [17] have been developed by research groups for specific purposes. However [60] demonstrates that these complex similarity measures can not show any significant improvement in datasets collected from various application domains.

DTW is an optimization problem with several constraints. Let's say we have two time series Q and C whose lengths are m and n.

$$Q = q_1, q_2, ..., q_i, ..., q_n \tag{3.1}$$

$$C = c_1, c_2, ..., c_j, ..., c_m \tag{3.2}$$

m-by-n matrix is constructed so as to align the Q and C in DTW and (i^{th}, j^{th}) element of the matrix defines the distance between q_i and c_j, $d(q_i, c_j)$,. The warping path of the series can be represented as W and an element of W can be shown as $W_k = (i, j)_k$.

$$W = w_1, w_2, ..., w_k, ..., w_K \tag{3.3}$$

where $max(m,n) \leq K < m+n-1$

Warping path constraints can be analyzed in there main categories which are namely, *boundary conditions*, *continuity* and *monotonicity*.

With respect to the boundary conditions warping path should start with point $w_1 = (1,1)$ and finish at point $w_K = (m,n)$.

In consecutive adjacent path cells, $w_k = (a,b)$ $w_{k-1} = (\bar{a},\bar{b})$, $a-\bar{a}$ and $b-\bar{b}$ should be less than or equal to 1. These rules reflect the continuity property of DTW.

Monotonicity constraint forces the points in the path to be monotonically spaced. $w_k = (a,b)$, $w_{k-1} = (\bar{a},\bar{b})$, $a-\bar{a}$ and $b-\bar{b}$ should be bigger or equal to 0. The path minimizing the warping cost is

$$DTW(Q,C) = min\left\{\frac{\sqrt{\sum_{k=1}^{K} w_k}}{K}\right\}. \tag{3.4}$$

Equation (3.4) can also be expressed as a recursive cumulative formula. This representation is suitable to solve the problem using dynamic programming,

$$\gamma(i,j) = d(q_i,c_j) + min\{\gamma(i-1,j-1),\gamma(i-1,j),\gamma(i,j-1)\}, \tag{3.5}$$

where $\gamma(i,j)$ represents the minimum cumulative distance at arbitrary alignment point (i,j).

If L_p distance is used in DTW calculations. For a real number $p \geq 1$, the L_p-distance or Lp-norm of x is defined as

$$|| \ x \ ||_p = (\mid x_1 \mid^p + \mid x_2 \mid^p + \ldots + \mid x_n \mid^p)^{\frac{1}{p}}. \tag{3.6}$$

Therefore Equation (3.5) turns into

$$\gamma(i,j) = \mid q_i, c_j \mid^p + min\{\gamma(i-1,j-1),\gamma(i-1,j),\gamma(i,j-1)\}. \tag{3.7}$$

Minimum of the cumulative distance shown in Equation (3.7) can be evaluated effectively by using dynamic programming.Trajectory demonstrates the warping path as $w_1, w_2, \ldots, w_k$, and the dashed lines define an envelope which the trajectory must lie in for an arbitrary warping parameter in Figure 3.2 .

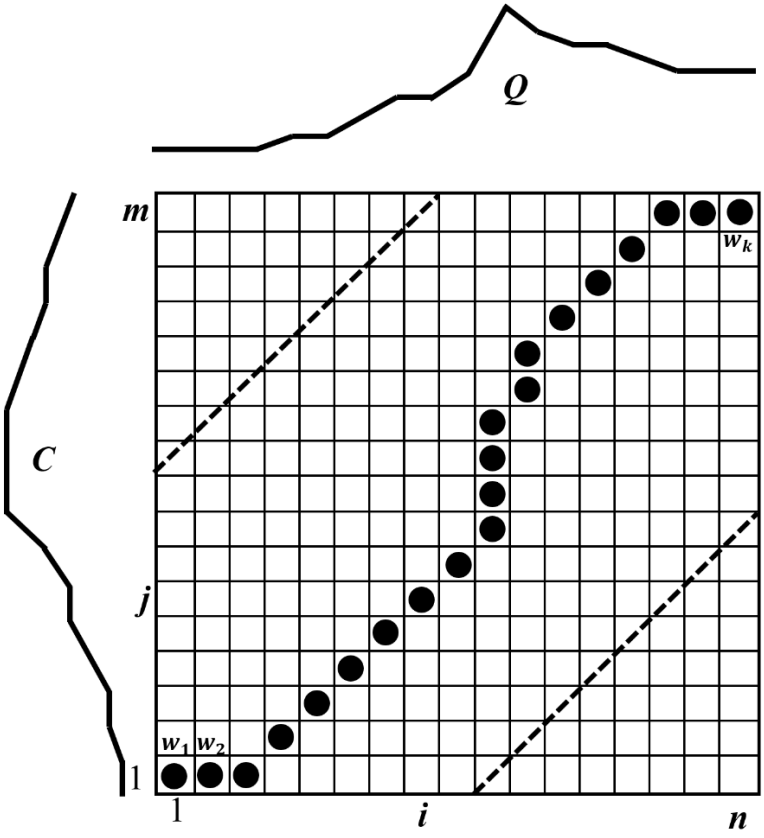


Figure 3.2: DTW warping path illustration

WDTW and DTW with limited warping window algorithms try to reduce the warping ratio between two time series.

If the series have fluctuations not only in X-axis, but also in Y-axis, it is possible that DTW cannot be successful. DTW is sensible to global differences such as offset translation or amplitude scaling effects [38]. To avoid the negative effects of these cases DDTW algorithm has been devised. The main difference between DTW and DDTW is that we should take discrete derivatives of data points in DDTW. In DDTW time sequences transformed with respect to the following formula,

$$\begin{aligned} D[Q] &= \frac{(q_i - q_{i-1}) + ((q_{i+1} - qi - 1)/2)}{2} \qquad , 1 < i < m \\ D[Q_1] &= D[Q_2], \\ D[Q_m] &= D[Q_{m-1}], \end{aligned} \tag{3.8}$$

where $D[Q]$ refers to the data points of series Q and

$$\begin{aligned} D[C] &= \frac{(c_i - c_{i-1}) + ((c_{i+1} - ci - 1)/2)}{2} \qquad , 1 < i < n \\ D[C_1] &= D[C_2], \\ D[C_n] &= D[C_{n-1}], \end{aligned} \tag{3.9}$$

where $D[C]$ refers to the data points of series C.

3.1.2 K-Nearest Neighbor Classifier

Nearest neighbor classification is one of the most basic and nonparametric classification methods in statistical data analysis. The main idea is finding the instances in the training set that are the closest to new instance to be labelled. Then taking the most commonly seen target class label as a classification result. k value defined in k-Nearest Neighbour (KNN) algorithm [23] stands for the number of closest instances that should be examined before deciding the class label.

The basic and classical k-Nearest Neighbor classifier uses L_p norm distance as a distance measures. Euclidean distance with nearest neighbor classifier model is one of the most common classification methods in literature. However it is not specialized for a time series classification purposes.

3.1.3 DTW with KNN Classification Method

Due to unsuitability of the classical KNN approach to the time series data, DTW with KNN classifier has been developed in this study. Actually nearest neighbor classifier with DTW is very common and a state-of-art method in time series classification and clustering due to non-linear mapping capability [37, 11]. DTW is the distance measure of k-nearest neighbor classifier instead of L_p norm distances. Researches demonstrate that 1-NN with DTW has been found very effective and successful on classification of time series [63]. The number used as k value (e.g. 1-NN, 4-NN) indicates how many closest neighbor need to be checked before deciding the target value.

3.2 Multi-Class Support Vector Machines

3.2.1 Support Vector Classifier

Although simple K-NN with DTW distance classifier has produced better results than the other classification methods, many studies have worked on other well-known models tried to adopt them for time series analysis [63]. Support vector classifiers is one of the most trending topics in last decades and often surpasses the K-NN method on classification problems. Therefore to classify time series, Support Vector Machines (SVM)-based algorithms have been studied intensively in diverse application domains, see [62, 21, 5, 66, 48, 45, 46] for more details.

In this section, firstly two class linearly separable SVM [19] method, then the general version, multi class version and kernel version of SVM will be briefly explained. In SVM, the main goal is to find a hyperplane that divides the training set into the classes with the largest margin, distance from the decision line to closest data points of each class, and classify them with minimum error.

$$w.x_i + b \geq 1, \quad y_i = 1. \tag{3.10}$$

$$w.x_i + b \leq -1, \quad y_i = -1. \tag{3.11}$$

where y_i is the class indicator, x_i is the features of each observation and w describes the normal vector to the hyperplane that separates the data points.

The hyperplane described in Equations (3.10), (3.11) can be combined as (3.12).

$$y_i(w.x_i + b) \geq 1, \forall x. \tag{3.12}$$

Since we know that generalization issue has crucial importance in designing stage of the classifiers, we should choose the hyperplane classifier having the maximum margin space separating the classes. The hyperplane which provides the same amount margin space to the closest samples of each class should be defined. To find the best direction giving the maximum margin space mathematically we should use Equation (3.13) that gives the distance of hyperplane to the nearest points of each class.

$$\frac{b}{|| \, w \, ||} \tag{3.13}$$

where $|| w ||$ is L_2 norm of w. Figure 3.3 illustrates this phenomenon clearly.

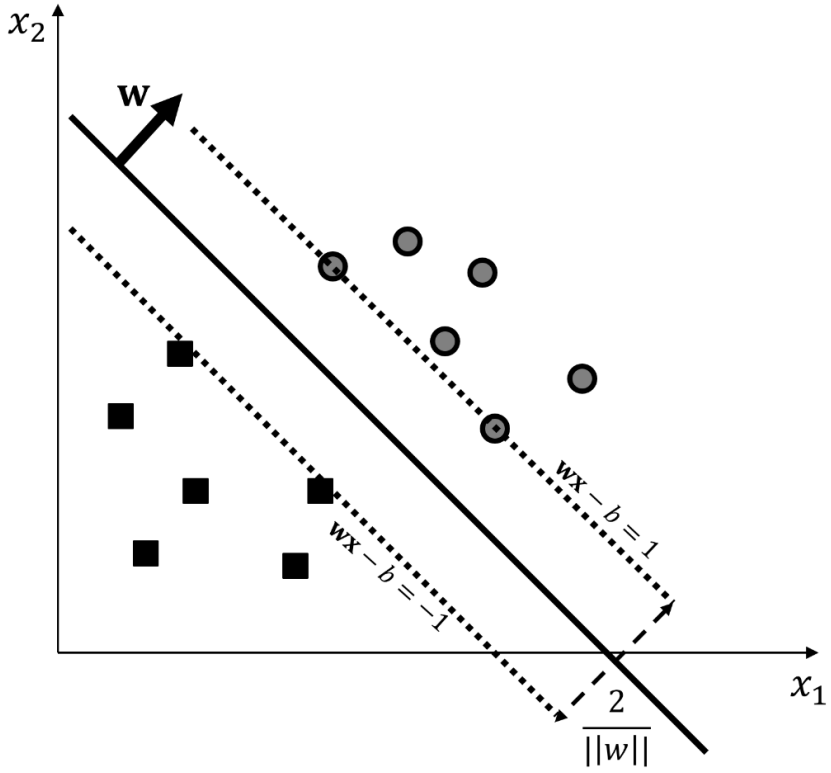


Figure 3.3: Hyperplane with margin length of two-class SVM

Let's scale the distance b to 1 for y_1 and to -1 for y_2. Then try to search the direction giving the maximum margin length.

Then margin length

$$\frac{1}{|| w ||} + \frac{1}{|| w ||} = \frac{2}{|| w ||} \tag{3.14}$$

requiring that

$$\begin{aligned} w.x_i + b &\geq 1, \quad \forall x \in y_1 \\ w.x_i + b &\leq -1, \quad \forall x \in y_2 \end{aligned} \tag{3.15}$$

If class indicator y_i takes +1 for w_1 and -1 for w_2, the problem can be transformed to the minimization problem such that

$$\text{minimize} \qquad J(w) = \frac{1}{2} || w ||^2 \tag{3.16}$$

$$\text{subject to} \qquad y_i(x_i w + b) \geq 1 \qquad i = 1, 2, ..., N \tag{3.17}$$

where N is the number of samples.

By applying the Karush Kuhn Tucker (KKT) conditions to Equation(3.16)-(3.17)

$$\frac{\partial}{\partial w}\mathcal{L}(w, b, \lambda) = 0, \tag{3.18}$$

$$\frac{\partial}{\partial w_0}\mathcal{L}(w, b, \lambda) = 0, \tag{3.19}$$

$$\lambda_i \geq 0 \qquad i = 1, 2, ..., N, \tag{3.20}$$

$$\lambda_i[y_i(x_i w + b) - 1] = 0, \qquad i = 1, 2, ..., N. \tag{3.21}$$

where λ_i is Lagrangian multiplier. Then the Lagrangian function is

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2}w^T w - \sum_{i=1}^{N} \lambda_i[y_i(x_i w + b) - 1]. \tag{3.22}$$

Using Equation (3.18), (3.19) and (3.22) we can get the hyperplane which divides the classes optimally

$$w = \sum_{i=1}^{N} \lambda_i y_i x_i, \tag{3.23}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0. \tag{3.24}$$

So as to compute Lagrangian multipliers we can use Lagrangian duality as

$$\text{maximize} \qquad \mathcal{L}(w, b, \lambda) \tag{3.25}$$

$$\text{subject to} \qquad w = \sum_{i=1}^{N} \lambda_i y_i x_i \tag{3.26}$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0 \tag{3.27}$$

$$\lambda \geq 0. \tag{3.28}$$

Substituting Equation (3.26) and (3.27) into (3.25) we get

$$\max_{\lambda}\left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j x_i^T x_j \right) \tag{3.29}$$

$$\text{subject to} \qquad w = \sum_{i=1}^{N} \lambda_i y_i x_i \tag{3.30}$$

$$\lambda \geq 0. \tag{3.31}$$

After finding the optimum Lagrangian multipliers (λ_i) from Equation (3.29), by substituting Lagrangian multipliers into Equation (3.26) optimal hyperplane can be calculated.

In general nonseparable cases the hyperplanes cannot separate the classes without false classification. There are three kinds of data points in nonseperable case as shown in Figure 3.4 .

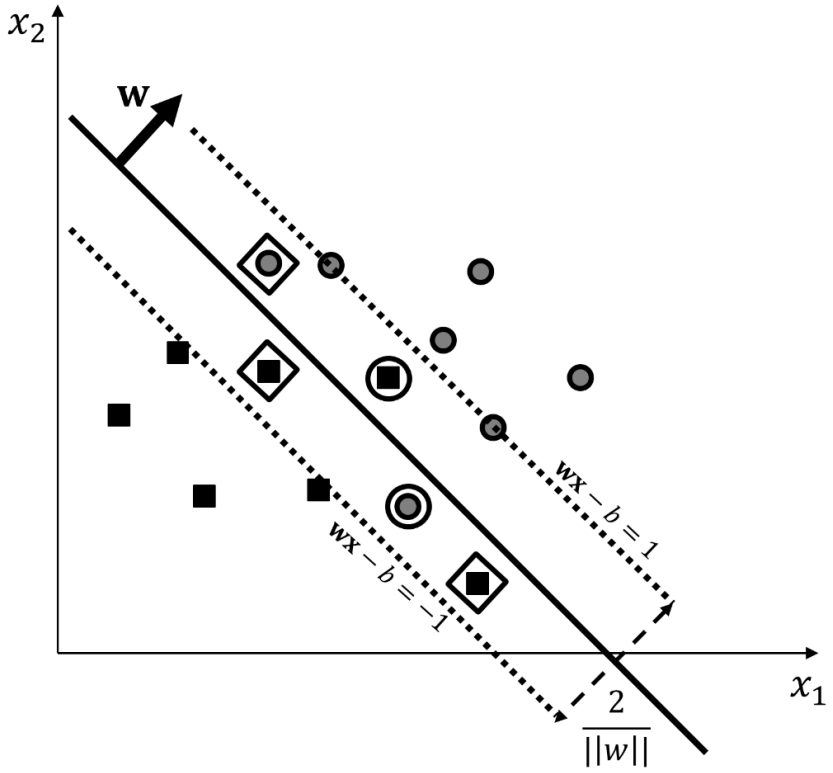


Figure 3.4: General non-separable hyperplane illustration

In first category data points are located outside of the hyperplane channel with correct labels. In second category, data points fall inside hyperplane channel, are correctly classified and marked with rectangle. In the last category circle-marked data points are classified falsely.

In nonseparable case Objective Function (3.16) and Inequality (3.17) should be replaced with

$$\text{minimize} \qquad J(w,b,\xi) = \frac{1}{2} \mid\mid w \mid\mid^2 + C \sum_{i=1}^{N} \xi_i \tag{3.32}$$

$$\text{subject to} \qquad y_i(x_i w + b) \geq 1 - \xi_i, \qquad i = 1, 2, ..., N \tag{3.33}$$

$$\xi_i \geq 0 \qquad i = 1, 2, ..., N. \tag{3.34}$$

where xi_i is the positive slack variable

For the first category of data points $\xi = 0$ as explained in separable case, for the second category $0 > \xi \geq 1$ and for the last category $\xi > 1$. C is a positive constant parameter and balances the sub-cost functions in Equation (3.32).The cost parameter, C, must be determined by the user. The best cost parameter can be obtained using parameter search algorithms such as grid search. $C \sum_{i=1}^{N} \xi_i$ is the penalized part of the objective function. The solution and further derivation of Objective Function (3.32) with Inequalities (3.33),(3.34) can be examined from [20, 55, 24].

The kernel function has essential importance in SVM. In fact the kernel function is a similarity measure for the data points. Non-linear transformation (or mapping) of the inputs to the high-dimensional feature space helps us work in non-separable cases. After the operation by a function, Φ, the linear separation problem of the inputs can be solved in the high-dimensional space. The duty of kernel functions is to take inner product of mapped data points.

$$k(x, y) = \Phi(x) \cdot \Phi(y). \tag{3.35}$$

There is no need to transform all the data points with Φ due to computational load and unnecessity. Using the kernel function corresponding to the Equation (3.35), dot product, we can reduce the computational time.

High dimensional transformation, the linear separation process and its reflection to the low dimensional space example can be monitored in Figure 3.5 .

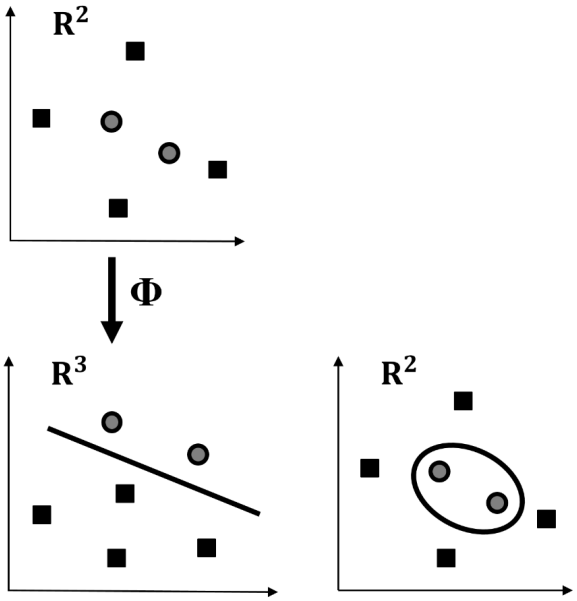


Figure 3.5: Basic non-linear kernel SVM operation

We should be careful about not violating the convexity properties of the SVM when using kernels. Hence, to achieve proper transformation, we should use positive semi-definite (PSD) definite kernel functions [27, 53]. As mentioned before, kernel functions increases the chances of more accurate separation of classes. However choosing the proper kernel function requires expert knowledge about data and kernel model. In our study we have used trial and error method while selecting the kernel function. All of the parameters used in kernel functions are searched by grid search technique.Some of the most common kernels used in non-linear SVM operations are presented below.

Polynomial kernel classifiers

$$k(x, x_i) = \Big\langle x, x_i \Big\rangle^d, \tag{3.36}$$

where d is degree of polynomial.

Radial Basis Function(RBF) classifiers

$$k(x, x_i) = exp(- \mid\mid x - x_i \mid\mid^2 / c) \qquad \text{where } c > 0. \tag{3.37}$$

Sigmoid Function classifiers

$$k(x, x_i) = tanh\Big(\kappa\langle x, x_i\rangle + \Theta\Big), \tag{3.38}$$

where $\kappa > 0, \quad \Theta \in \mathbb{R}, \quad \langle x, x_i \rangle$ denotes inner product.

In the literature, there are different kinds of approach for multi-class SVM classification. One of these approaches is classifying one class in each iteration. The name of this procedure is one-against-rest method and this approach has been used widely in SVM researches [12, 50]. Another approach is called the one-against-one method. This method defining $\left(\frac{k(k-1)}{2}\right)$ hyperplanes where each one is trained to separate two classes (k is the number of class). The other and the most systematic approach is direct multi-classification by modifying the objective function. By applying this direct method to the classification tasks all classes can be separated at once in a simultaneous way [53]. Related derivations and explanations can be examined from [61] and [53].

CHAPTER 4

NEURAL NETWORKS, DEEP LEARNING & FRAMEWORK

In this section firstly the neural network architectures and working principles have been introduced for the background information. Then the training mechanism and specialized versions of artificial neural networks have been demonstrated. Afterwards the deep learning concepts derived from neural network has been presented in a detailed way. We have proposed two deep learning based classifiers which were constructed using Stacked-Autoencoder (SAE) and Deep Belief Networks (DBN).

4.1 Neural Networks

Neural networks are seen as a class of mathematically structured models inspired by biological neural networks in living complex organisms. Although relation between artificial neural network and real neural system is weak, the working principles of real neural system are tried to be emulated by artificial neural networks in simple way.

The artificial neural networks have been widely used to solve problems such as time series prediction, image processing, classification, regression analysis, data processing, pattern recognition, decision making, clustering in areas like fraud detection, astronomy, process control, cognitive sciences etc. See [29, 3, 42, 2] for more details.

4.1.1 Artificial Neurons

Basic block of artificial neural network is an artificial neuron that shows some similarities with a biological neuron. A biological neuron collects signals from other neurons

via their dendrites; then these signals are collected and the response is generated by a cell. Afterwards the response is distributed by axons to the other neurons.

Similarly, in artificial neuron case, the individually weighted inputs are transmitted to the body of artificial neurons. The body sums the weighted inputs and bias. Then the summation is processed with respect to the defined transfer function.

First formal definition of a basic computing neuron model is made and formulated by McCulloch and Pitts [43] in 1943. In these definitions the output function was a step function. It means when the specific threshold value is met, the output takes a value; in the other case output result value is zero. McCulloch and Pitts model can be examined in Figure 4.1 .

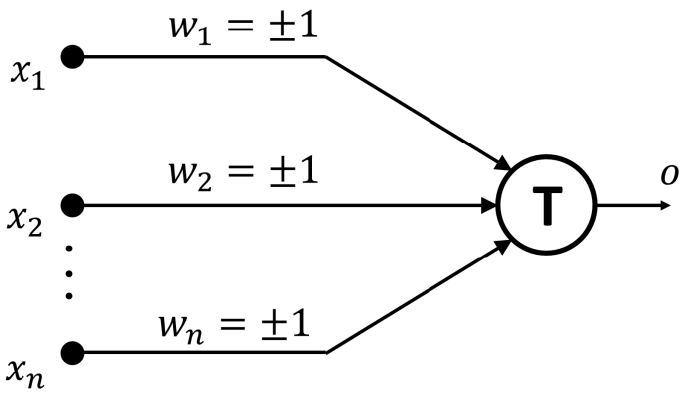


Figure 4.1: McCulloch and Pitts model: weights are represented as w_i, input are x_i ,output is o and the transfer function is τ.

Formulation of McCulloch and Pitts model:

$$o^{k+1} = \left\{ \begin{array}{ll} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{if} \quad \sum_{i=1}^{n} w_i x_i^k < T \end{array} \right\}. \tag{4.1}$$

The positive weight values reflect inhibitory connections while the negative ones designate excitatory. The developed version of the general model is not using a basic thresholding stage but a transfer function block instead. Common expression used for output is y, o or f(net) in the literature. Also a bias can be added to the input of transfer functions in the general model.

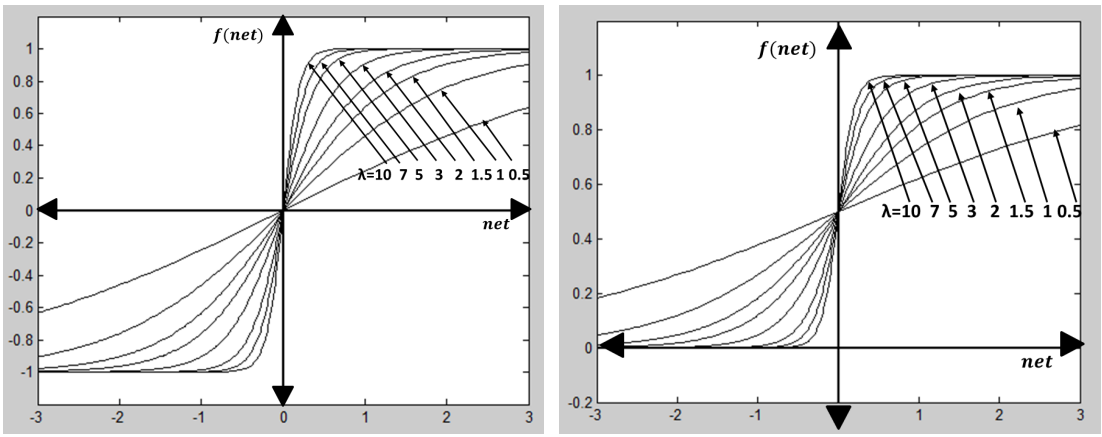
Transfer function processes the weighted input signals and biases. Hence the general

mathematical model is descried as

$$y(k) = \mathrm{F}\Big(\sum_{i=1}^{n} w_i(k)x_i(k) + b\Big). \tag{4.2}$$

A linear combination of multiple linear functions is still a linear functions. Therefore, if each of the neurons in a network uses a linear function, then the final output of the neural network will be the linear function combinations of inputs. Hence, generally the neurons perform the nonlinear operation in their activation functions.

Typically used application functions are either bipolar continuous / binary functions or unipolar continuous / binary functions. The difference between unipolar and bipolar functions are that bipolar functions can produce both negative and positive responses while unipolar ones produce only non-negative responses.



(a) Bipolar continuous function

(b) Unipolar continuous function

Figure 4.2: Continuous Sigmoid Activation Functions

Bipolar continuous activation function is

$$f(net) = \frac{2}{1 + exp(-\lambda net)} - 1. \tag{4.3}$$

Unipolar continuous activation function is

$$f(net) = \frac{1}{1 + exp(-\lambda net)}. \tag{4.4}$$

Unipolar and bipolar continuous functions shown in Figure 4.2 are often called sigmoidal characteristics or sigmoid functions. Bipolar structures are more common than the unipolar functions in artificial networks and the bipolar sigmoid functions

are the most common ones. Moreover, artificial neurons that have binary and continuous transfer function are called binary and continuous artificial neuron perceptron respectively. Binary perceptron can be most commonly used in the output layer of the artificial neural networks for classification problems.

Bipolar binary activation function can be given as

$$f(net) = sgn(net) = \left\{ \begin{array}{ll} 1 & \text{if} \quad net > 0 \\ -1 & \text{if} \quad net < 0 \end{array} \right\}. \tag{4.5}$$

Unipolar binary activation function is

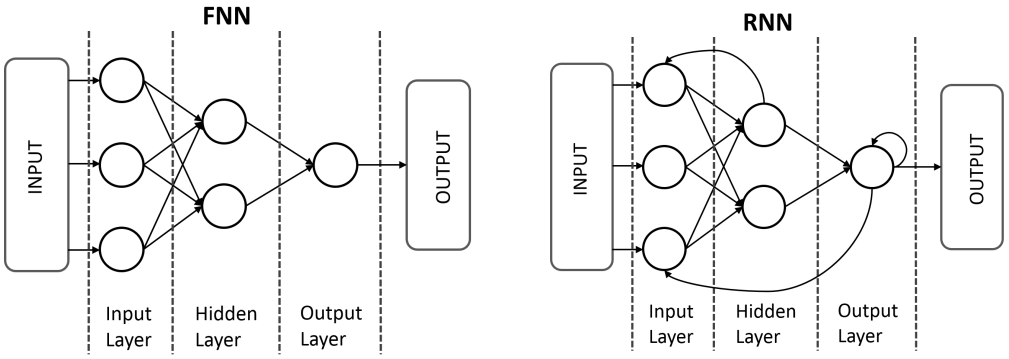
$$f(net) = sgn(net) = \left\{ \begin{array}{ll} 1 & \text{if} \quad net > 0 \\ 0 & \text{if} \quad net < 0 \end{array} \right\}. \tag{4.6}$$

4.1.2 Models of Articially Neural Networks

Single artificial neuron cannot solve real life problems at all, however combining two or more artificial neurons are capable of solving complex real life problems.

The neural network can be defined as an interconnection of neurons. Related neuron outputs and inputs are connected, through weights. Delay block can be placed between neurons if needed.

Neurons of an artificial neural network are not randomly interconnected. There are standardized topologies of neural networks. These topologies are fixed and predefined so as to solve the problems in an efficient and easy way. These topologies can be examined in two basic classes which are *feed-forward* and *recurrent topologies*. In feed-forward case information flows in only one direction from the input to the output. Therefore feed-forward neural network (FNN) topology is also called *acyclic graph*. On the contrary, in simple recurrent neural network (RNN) topology the information does not flow only from input to the output direction, but also flows from output to the input direction. Hence these kind of topologies can be referred to as *semi-cyclic graphs* [39].



(a) FNN topology (b) RNN topology

Figure 4.3: Feed-forward (FNN) and recurrent (RNN) topology

Neural networks can be described layer by layer which refers to a group of neurons in same level as shown on Figure 4.3 . From input to the output direction, the first layer is called input layer and the last one is named output layer. All remaining layers placed between the input and output layer are labeled as hidden layers.

Hidden layers are where neural networks stores abstract and internal representation of the training samples.

A single hidden layer network which has a finite number of units can be trained to express any random function with an acceptable error ratio with respect to the universal approximation theorem. Although single hidden layer network is sufficient to learn any function, multi hidden layer networks can give better results.

4.1.3 Feedforward Networks

Feed-forward artificial network is basically artificial neural network with feed-forward topology. This topology has one main condition that the information should flow only from the input to the output direction. There is no limitation or restriction on type of transfer functions, number of connections between neurons or number of layers such as deep networks.

Simple feed-forward artificial network with one hidden layer is shown on Figure 4.4 .

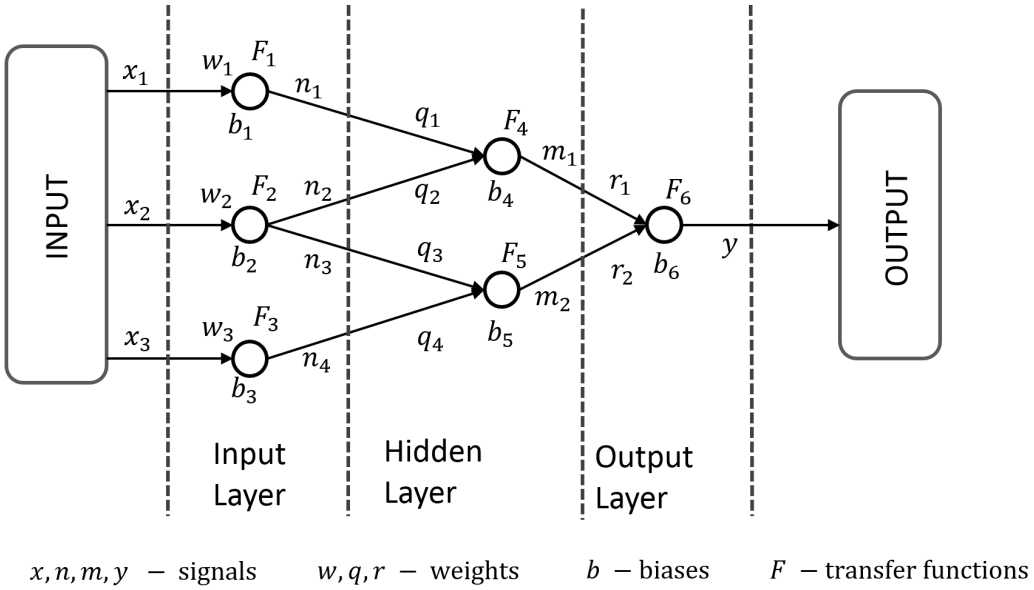


Figure 4.4: Feed-forward artificial neural network

Detailed mathematical representation of feed-forward NN can be given as follows in below. In the Equations (4.7) to (4.14) the transfer functions are represented as F_i. In fact F_1, F_2, F_3 are input layer transfer functions fed by input signals(x_i). w_i, q_i and r_i are weights of input hidden and output layers respectively. The output of input layers that feeds the hidden layers are n_i. The output signals produced by hidden layer transfer functions(F_3,F_4) are shown as m_i in the Equations (4.11) ,(4.12) and (4.13). The biases are represented as b_i in all of the following equations. Lastly the output of feedforward network that assigns the class labels is y.

$$n1 = F_1(w_1x_1 + b_1). \tag{4.7}$$

$$n2 = F_2(w_2x_2 + b_2). \tag{4.8}$$

$$n3 = F_2(w_2x_2 + b_2). \tag{4.9}$$

$$n4 = F_3(w_3x_3 + b_3). \tag{4.10}$$

$$m1 = F_4(q_1n_1 + q_2n_2 + b_4). \tag{4.11}$$

$$m2 = F_5(q_3 n_3 + q_4 n_4 + b_5). \tag{4.12}$$

$$y = F_6(r_1 m_1 + r_2 m_2 + b_6). \tag{4.13}$$

$$\begin{aligned} y = F_6\Big[& r_1(F_4[q_1 F_1[w_1 x_1 + b_1] + q_2 F_2[w_2 x_2 + b_2]] + b_4) + ... \\ & ... + r_2(F_5[q_3 F_2[w_2 x_2 + b_2] + q_4 F_3[w_3 x_3 + b_3] + b_5]) + b_6\Big]. \end{aligned} \tag{4.14}$$

Hand calculation of the parameters of artificial neural network is impractical hence computers and specialized software are used to build and optimize the neural network for all type topologies.

4.1.4 Backpropagation

The most common training method for neural network is backpropagation algorithm. Gradient descent method is used in least mean square error of target value to feed the network and updating the parameters. Backpropagation is a very popular method because it is useful, powerful and simple.

Setting the weights and biases of neural networks is one of the essential problems. Backpropagation method tries to find optimal weights based on training samples and desired target values. Difference between the desired and actual output values is used in least mean square (LMS) error calculation that is the main input of backpropagation algorithm. This LMS training error on a training set is the sum over output units of the squared difference between the actual and the desired output. [20]

LMS Training error is formulated as:

$$J(w) = \frac{1}{2} \sum_{k=1}^{c} (t_k - z_k)^2, \tag{4.15}$$

where t is desired and z is actual output vectors of length c, and w is weights of the network.

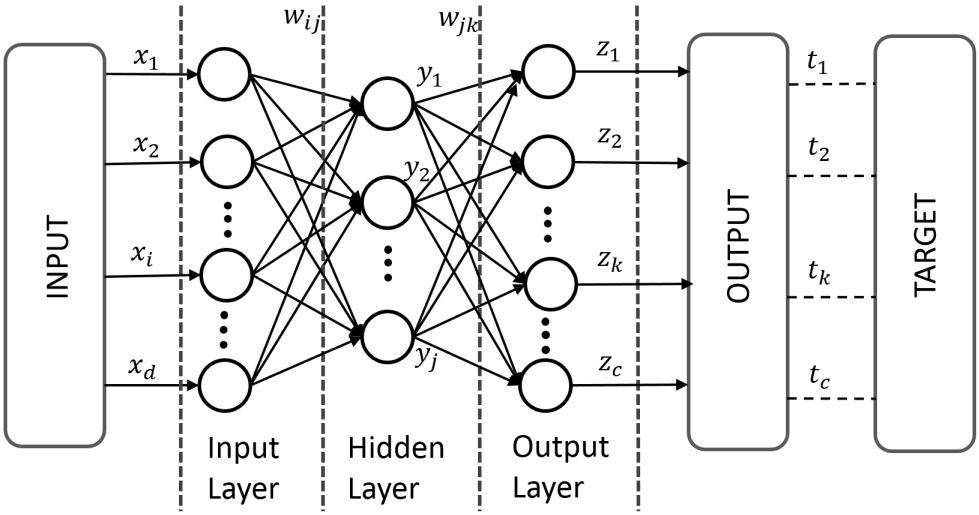


Figure 4.5: Backpropagation diagram

Randomly initialized weight values are changed with respect to the gradient descent method to reduce the error. Learning rate defines the amount of the change in weights. Learning rate is represented as η.

$$\Delta w_{mn} = -\eta \frac{\partial J}{\partial w_{mn}}. \tag{4.16}$$

While updating the weights of hidden layers and input layers the dependency of weights should be taken in consideration. If the error is not explicitly dependent on a hidden-output weight, chain rule differentiation expansion should be used.[20] Such as for a one-input-layer, one-hidden-layer network the learning rule for the hidden-to-output weights as follows. By applying the chain rule to find partial derivatives of the LMS error($J(w)$)

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}, \tag{4.17}$$

where w_{jk} represents the hidden to output layer weights and δ_k is sensitivity value of outputs.

$$\delta_k = -\frac{\partial J}{\partial net_k}, \tag{4.18}$$

where net_k represents the input of output layer. Then we differentiate Equation

(4.16) we get

$$-\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k)f'(net_k). \tag{4.19}$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j. \tag{4.20}$$

After taken the derivative in Equation (4.20), we get the weight update (Δw_{kj})

$$\Delta w_{kj} = \eta\delta_k y_j = \eta(t_k - z_k)f'(net_k)y_j. \tag{4.21}$$

For the input-to-hidden weights we applied similar derivations. By applying chain rule one more time we obtain

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}. \tag{4.22}$$

$$\begin{aligned}
\frac{\partial J}{\partial y_j} =& \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
=& -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
=& -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} \\
=& -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{jk}.
\end{aligned} \tag{4.23}$$

The Equation (4.23) expresses the effect of output signal to the LMS error value.

$$\delta_j = f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k. \tag{4.24}$$

Finally the weight update value between the hidden and input nodes can be calculated as

$$\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k, \tag{4.25}$$

where δ_j and δ_k are hidden layer and output layer sensitivity values.

Hidden unit sensitivities are proportional to the sensitivities at the output units as shown in Equation 4.25 and Figure 4.6 .

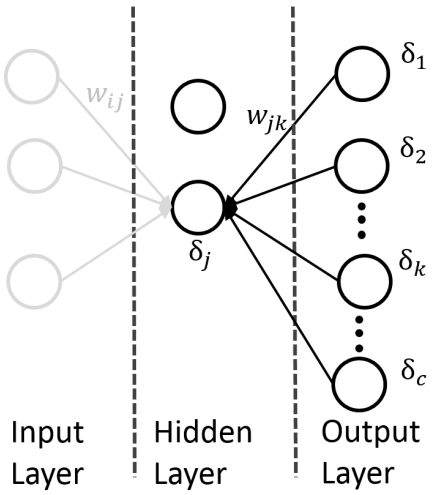


Figure 4.6: Hidden units and output units sensitivity relation (reproduced from [20])

The most common and useful neural network training optimization methods are *batch training* and *stochastic training*. In stochastic training patterns are selected from training samples randomly. Then with the help of gradient descent algorithm weights are updated gradually. However, in batch training all of the training samples are used in learning process. Epoch can be defined as the number of training set presentation to the network. Lastly, to stop the training process automatically, the stopping criterion should be defined.

4.1.5 Recurrent Artificial Neural Networks

Recurrent artificial network is basically artificial neural network with recurrent topology. In this topology there is no restriction on information flow direction. The information can flow in backward direction and also in between same level neurons. Backward flow ability enables recurrent networks to use their internal memory to examine sequences of inputs.

The most general topology of recurrent network is fully recurrent one where each basic network block is connected to the other ones directly in all direction. For different applications, recurrent networks are specialized such as Hopfield [33], Elman [22], Jordan [35] recurrent networks etc. Fully recurrent artificial neural network can be

seen in Figure 4.7 .

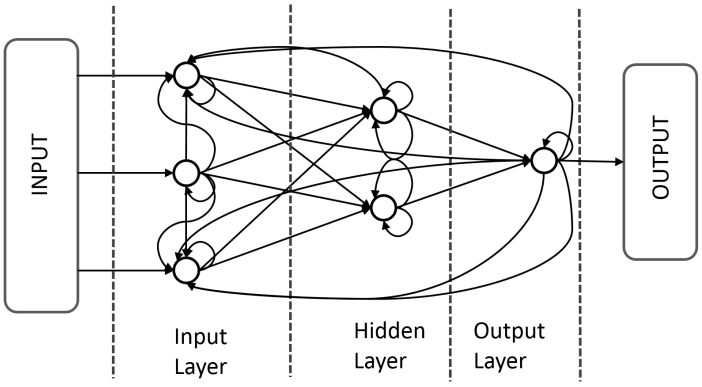


Figure 4.7: Fully recurrent artificial neural network

4.1.5.1 Hopfield Artificial Neural Network

Hopfield artificial neural network is a specialized recurrent artificial neural network used to get stable target information. Hopfield network is an asynchronous and fully connected graph. These recurrent architectures take two different values either -1,+1 or 0,+1 for it states. Hopfield artificial neural networks have two critical restrictions. One of them is that there should not be self-loops in network. The other one is that weights between nodes connections should be symmetric. Symmetric weights assure that the energy decreases monotonically while following the activation rules. Non-symmetric weights can cause some chaotic or periodic behaviour. [39, 67]

4.1.5.2 Elman and Jordan Artificial Neural Networks

Elman artificial neural network, also called simple recurrent network (SRN), is a special recurrent network with tree layer. In these topologies there is a back loop from hidden layer to input layer consisting of unit time delays. Hidden layer transfer functions of this topology are sigmoid and it has linear output layer. Elman artificial neural network has a memory allowing to detect time-varying pattern information and generate that pattern. Spatial and temporal pattern can be detected by Elman artificial neural network. [28]

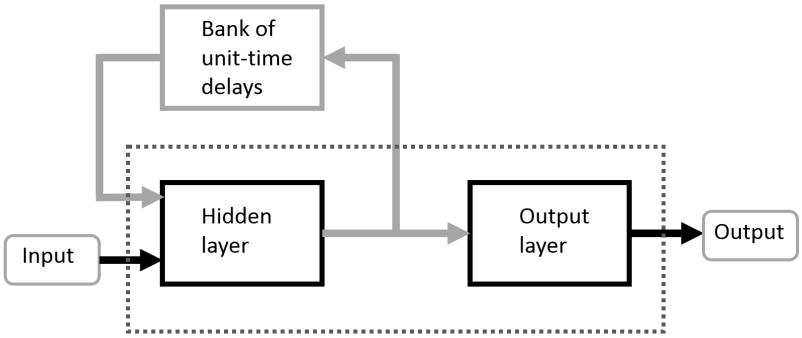


Figure 4.8: Elman artificial neural network (reproduced from [28])

The only difference between Jordan and Elman neural network is that Jordan network context unit is connected to the output layer, not to the hidden layer.

4.1.5.3 Long Short Term Memory

Long Short Term Memory is another topology of recurrent artificial neural nets. It can classify and predict time series with both short and long lag times between important events. Long Short Term Memory is much more capable of remembering information than the basic recurrent artificial networks. Transfer function of input layer of Long Short Term Memory is sigmoid.

Architecture of Long Short Term Memory can be seen in Figure 4.9 .

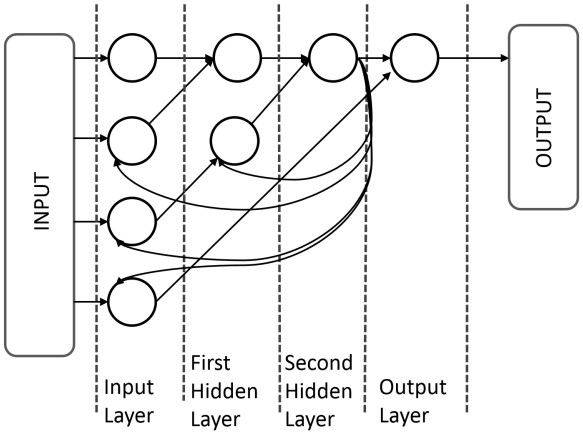


Figure 4.9: Simple Long Short Term Memory artificial neural network

The most critical neurons in the input layer are third and fourth neurons. Third neuron determines how long the memory unit will remember the information and bottom neuron decides when value from memory should be passed to output [39].

4.1.5.4 Bi-directional Artificial Neural Networks

Bi-directional Artificial Neural Networks are specialized to predict complex time series like Long Short Term Memory. Bi-directional Artificial Neural Networks can give better results than Long Short Term Memory in time series prediction problems [59].

Bi-directional are named direct and inverse directional neural networks. These two individual artificial sub-networks are interconnected through two dynamic artificial neurons. These neurons can remember their internal states. This structure also allows artificial network to predict not only future but also past values. After one of the artificial neural sub-network learns how to predict future values in first phase, the second sub-network learns predicting past values in the second phase.

Architecture of Bi-directional Artificial Neural Networks can be monitored in Figure 4.10 .

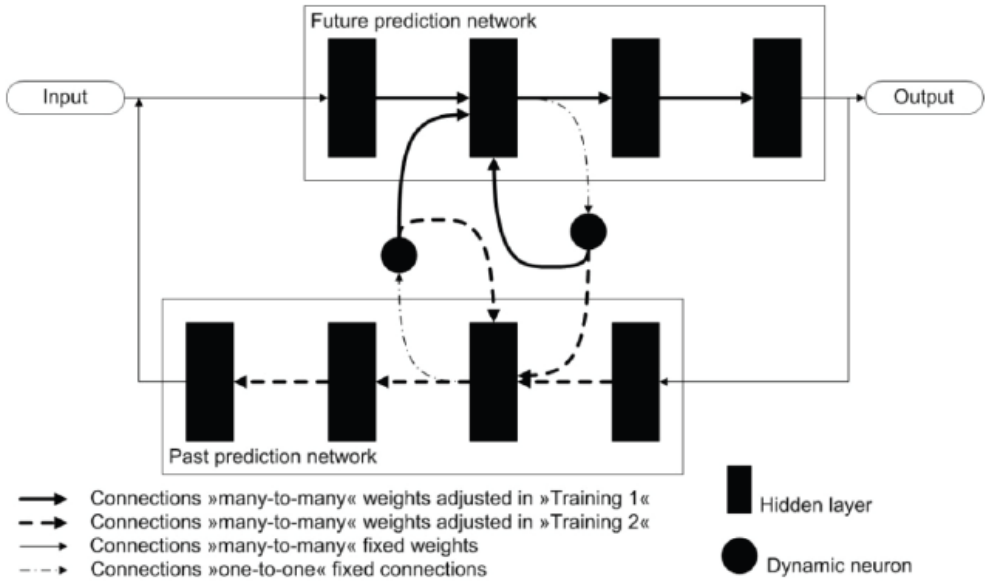


Figure 4.10: Bi-directional Artificial Neural Network (reproduced from [39])

4.1.5.5 Stochastic Artificial Neural Network

Stochastic artificial neural network can be constructed either by giving stochastic weight to the artificial network inputs or by arranging the transfer functions of network as a stochastic transfer function. Random fluctuation can help us to avoid local optimal solutions, so stochastic artificial neural network is useful for optimization areas.

A special form of stochastic recurrent artificial neural network is Boltzmann machines whose transfer functions are stochastic. Boltzmann machines can be seen as the stochastic, generative counterpart of Hopfield nets [1].

4.2 Deep Learning

Deep learning concept have been inspired by artificial neural network and uses the structures of neural networks. We have proposed two methods in this study which are Autoencoder (AE) and Restricted Boltzmann Machine (RBM) based models using the NN training principles. Such as the Autoencoder is a special type of feedforward and the Restricted Boltzmann Machine is a generative stochastic neural network model.

In the last decade, deep learning has been seen as an emerging branch of machine learning research [7, 32]. It takes the name of deep due to its deep and hierarchical structures. Until recently, shallow structured architectures such as Gaussian mixture models (GMM), support vector machines (SVM), multi-layer perceptrons (MLP) which has a single hidden layer (including extreme learning machines) were used in signal processing and machine learning areas. Some real world applications except for well-constrained and simple problems cannot be solved with shallow architectures.

Deep learning studies have been impacting lately a wide range of signal and information processes. It has also developed the scope of problem definitions and solutions especially in machine learning areas.

Deep learning (DL) can be described in many ways depending on the purpose of

using it. Basically deep learning is an improving branch in machine learning that has many complicated layers to process information in nonlinear way. Main intended purposes of deep learning are both supervised and unsupervised feature selection, data transformation and classification. Higher level and more complex features can be extracted with this hierarchical technique. Developing deep learning concepts move machine learning areas closer to the AI (artificial intelligence) which is primarily purpose of machine learning.

There is a growing interest in deep learning in the past couple of years. Also there are some reasons for the popularity of deep learning in recent years. First of all, developments in the processing chip technology, especially in the graphical processing units (GPU) area, allow us inspect the bigger amount of data and parallel processing in very limited time. GPUs are highly suited for the matrix/vector math operations involved in machine learning. GPUs have been shown to speed up training algorithms significantly. Hence increased size of training data demand would not be a problem anymore. The advances in information technology, machine learning and signal processing research also increases the importance of complex and non-linear approaches to learn feature spaces. Besides it has one more positive aspect which is deep learning can be used for both unlabeled and labeled data.

One of the main advantage of deep learning is that it replaces handcrafted features with effective methods for semi-supervised and unsupervised feature learning and hierarchical feature extraction. [54] Also one essential principle of deep learning is that it uses raw features for classification problems. By way of explanation, deep learning transfers the data to more compact level in each consecutive layer, and eliminates the redundancies and outliers, and does not try to learn manually extracted and specially designed features [65]. Deep learning helps to disentangle abstract concepts being learned from the lower level ones and pick out which features are useful for learning [8]. However when shallow structured learning method is applied to the linearly non-separable data, highly expert knowledge and huge computational analysis are essential to define the kernel functions and cost variables. Moreover, selecting domain specific features is also time-consuming.

The deep learning also overcomes the curse of dimensionality [9] issue also men-

tioned in chapter 2.2.5 which is one of the major problems in machine learning and pattern recognition while the shallow-structured learning method does not. Learning methods based on local generalization can suffer from highly varying functions. Deep learning can overcome this curse of dimensionality problem with the use of distributed representations and different training protocols.

Deep learning concept is inspired by artificial neural network (ANN) research (new generation NN). Layers that have been used in deep learning include hidden layers of an artificial neural network and sets of complicated propositional formulas [7]. In deep generative models, layer wise organized latent variables may also be included. Deep Belief Networks (DBN) and Deep Boltzmann Machines (DBM) can be given as successful examples of these deep generative models.

More hidden layers with many neurons in a DNN can improve the power of models and get closer to the optimal scenario. However using more deep and wide networks increase the computational time in training process. Resource requirement also increases with complexity.

Although the larger deep networks learn better and improve the power of models, adding more hidden layers does not always give better results. More hidden layers makes the backpropagation less effective for the first layers. Hence the gradient descent algorithm may fail to find optimal global solution. This phenomenon is named as vanishing gradient problem in machine learning. The other negative aspect of larger networks is overfitting problem.

So as to solve the vanishing gradient and overfitting problems some unsupervised pre-training methods were introduced and have been being developed. [32, 10] Unsupervised greedy layers-wise training should be done separately and successively. Training begins from the first hidden layer and continues to the last layer of deep network. After that, fine tuning-process to whole network should be performed in supervised way. Training procedure are shown on Figure 4.11 in simple way.

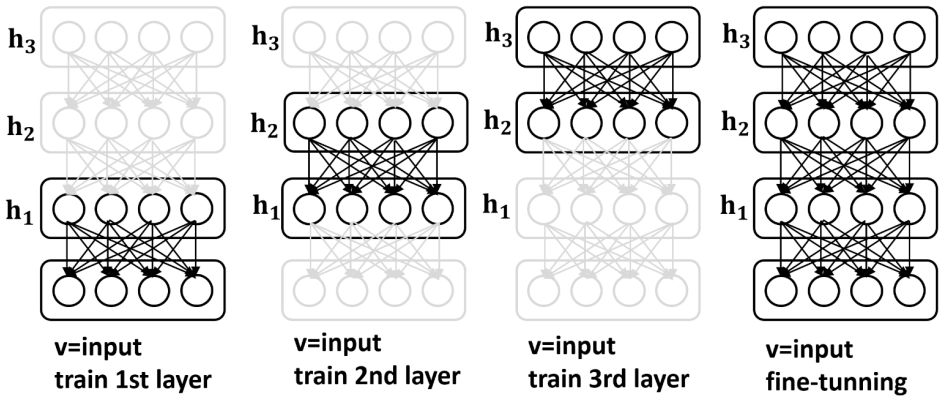


Figure 4.11: A greedy unsupervised layer-wise pretraining stage followed by a supervised fine-tuning stage affecting all layers (reproduced from [4])

Deep learning is much related to, signal processing, artificial intelligence, pattern recognitions, optimization and graphically-represented modelling main fields.

Various deep learning architectures have been implemented to fields such as computer vision, automatic speech recognition, image and speech feature coding, natural language understanding, audio processing and recognition, hand-writing recognition, bioinformatics, medical areas. The real impact of deep learning in industry started in large-scale speech recognition around 2010.

4.2.1 Applications of Deep Learning

Recognition of speech signals automatically is the most widely used and successful case among the all deep learning applications. Almost all of the currently used big commercial speech signal recognitions system uses deep learning based algorithms. (Apple Siri, Skype Translator, Microsoft Cortana . . .)[65].

Sequence division and discrimination, feature processing, multi-tasking, convolutional neural nets and complex version of LSTM recurrent networks processes are conducted by deep learning based models in automatic speech recognition.

Very recently, deep learning methods outperformed the shallow-structured state of art methods in computer vision area especially in image and object recognition branch of

computer vision. Nowadays major image processing and object recognitions projects based on deep learning uses pre-trained ImageNet database[40]. With the pre-trained model and successful fine tuning stage, challenging tasks of automatic object detection problems can be solved and performances are improved.[57]

Deep learning techniques are currently used for inventing or developing drugs and toxicology experiments lately. CRM automation system also prefers deep reinforcement learning in direct marketing settings. In linguistics, especially in sentiment studies, word embedding and machine translation areas deep learning gives promising results. Deep learning based approaches are getting more and more common in Bioinformatics.

4.3 Autoencoders

An autoencoder is a special type of feedforward neural network model designed to learn compressed representation of given dataset. Actually, the autoencoder was first proposed as a feature reduction model. The linear autoencoder can give same independent representation of dataset as Principle Component Analysis. An autoencoder has two layers the first of which is compression layer commonly named as encoding layer, the other one is decoding layer. The number of decoding layer units is equal to the number of input variables tried to be compressed. Autoencoders are trained so as to recover an input sample as much as possible i.e., the output and the input data are nearly the same.

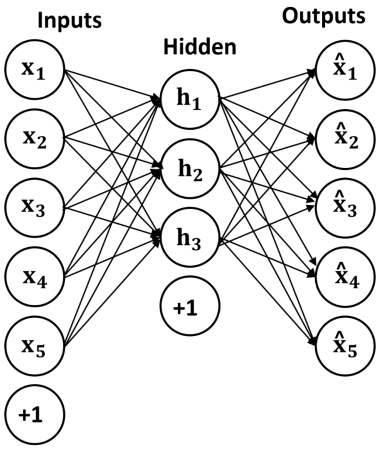


Figure 4.12: Basic Autoencoder architecture

In Figure 4.12 , hidden layer encodes the inputs, and the outputs layer tries to decode the encoded input. The basic formulation of this process is

$$f_{dec}(f_{enc}(x)) = \hat{x} \simeq x. \tag{4.26}$$

The number of hidden units should be smaller than the number of output layer units to force the network to learn compressed, compact and important features. Both dimensionality reduction and outlier detection operations would be achieved in encoding procedure. Therefore by favoring simpler representations, overfitting related problems would be avoided during feature detection procedure.

The training of autoencoders can be achieved by applying back-propagation of output layer error. Cross-entropy and LMS errors are two commonly used error types for backpropagation.

$$L_{LMS}(x, \hat{x}) = \frac{1}{2}\sum_{i=1}^{c}(\hat{x}_i - x_i)^2. \tag{4.27}$$

$$L_{CE}(x, \hat{x}) = \sum_{i=1}^{c}[(x_i log\hat{x}_i + (1 - x_i)log(1 - \hat{x}_i)]. \tag{4.28}$$

To avoid unwanted negative effects of training process like encoding identity function, $f_{enc}(x) = x$, tied weights constraint can be added.

Working principle of an autoencoder based deep learning algorithm can be demonstrated with a basic running example. In the illustration, the main objective is to detect sick patients who have flu symptoms. Let's suppose

- There are six binary criteria (features) for all of the patients.
- The first three criteria are symptoms related to flu illness. For instance, 1 0 0 | 0 0 0 reveals that the patient is coughing, and 0 1 0 | 0 0 0 reveals that the patient has a high temperature. The combination of high temperature and coughing is shown as 1 1 0 | 0 0 0, so on
- The last three binary attributes (criteria) are counter symptoms. If the patients have one or more than one "1" values at these features, it's more likely that they are healthy. For example, 0 0 0 | 0 1 0 indicates that the patient has a flu vaccine. Hence, 0 1 0 | 0 1 0 indicates that the patient has a flu vaccine but also he has a high temperature, and so forth.

In this illustration we will consider the patient to be sick when the sum of the last three attributes are less than the sum of the first three. Ties breaking in favor of the healthy patients. e.g.:

110 | 010, 111 | 001, 101 | 100, 011 | 010, 001 | 000, ... = sick patients

111 | 111, 010 | 011, 100 | 110, 000 | 111, 011 | 101, ... = healthy patients

We can construct and train an autoencoder with six input, two hidden and six output units. After hundreds iterations, we can observe that when any of the sick patients is presented to the AE based network, one of the two the hidden units always exhibits a higher activation value than the other one. On the contrary, when the data of the healthy patient is presented, the other hidden unit has a higher activation value. Hence we have a model to classify status of new patients. Also the more compact representation of inputs is obtained in the hidden layer of the network.

4.4 Stacked Autoencoders

Stacked autoencoders (SAE) is a deep network which is composed of consecutive autoencoders. A greedy unsupervised learning method is proposed for stacked autoencoders in [13, 30]. The hidden layer of the first autoencoder acts as an input of the second encoder. The hidden layer of the second autoencoder also cats as an input of a third autoencoder, and so on.

During the training process of first autoencoder, we are interested in only the first hidden layer and network inputs. Weights of the first autoencoders are updated by using backpropagation algorithm. After updating the weights, the temporary output layer, essential for the training stage of the autoencoder, is removed from the network.

For the second autoencoder, the outputs of the first hidden layer are the inputs of the second hidden layer. Hence after the training procedure of the first autoencoder, we are no longer interested in the network raw inputs.

This procedure is repeated for all hidden layers. After the pre-training of the hidden layers, the network is trained in a supervised way which is called fine-tuning stage.

In fine-tuning stage the output layer is also needed.

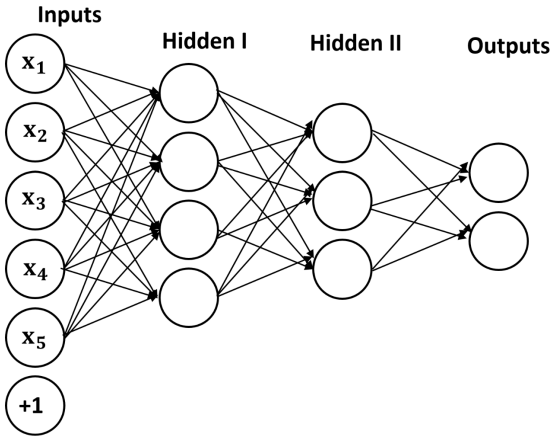


Figure 4.13: Stack autoencoder network diagram

4.4.1 Restricted Boltzmann Machines

The Restricted Boltzmann Machine (RBM) is a generative stochastic neural network model between input units (visible units) and hidden units (latent units). The connection between the hidden units and input units are undirected and there is no connection among the visible units and hidden units.

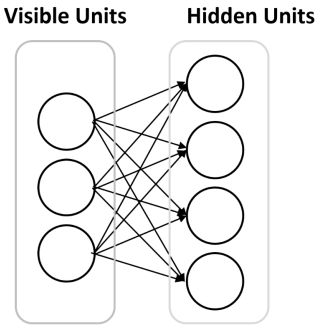


Figure 4.14: Restricted Boltzmann network diagram

A Restricted Boltzmann Machine probability distribution for input vector is

$$p(v) = \sum_{h} \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}. \tag{4.29}$$

An RBM defines a joint probability on visible and hidden units as shown in Figure 4.15.

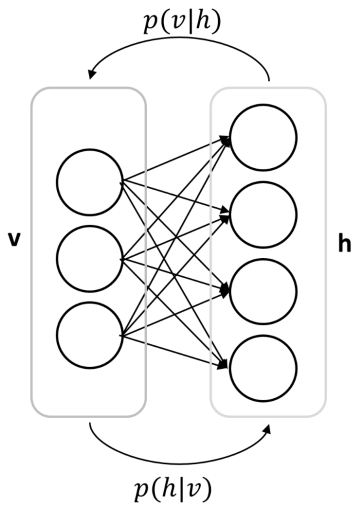


Figure 4.15: Joint probability on visibile and hidden units

Then the energy function defined over visible and hidden vectors are

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} w_{ij} v_i h_j \tag{4.30}$$

where a_i and b_j are visible and hidden unit biases.

According to the neural network propagation rule the conditional probabilities are

$$\begin{aligned} p(v \mid h) &= \prod_i p(v_i \mid h) \quad \text{and} \quad p(v_i = 1 \mid h) = sigm\Big(a_j + \sum_j h_j w_{ij}\Big), \\ p(h \mid v) &= \prod_j p(h_j \mid v) \quad \text{and} \quad p(h_j = 1 \mid v) = sigm\Big(b_j + \sum_i v_i w_{ij}\Big). \end{aligned} \tag{4.31}$$

To define the Gaussian-Bernoulli RBM energy function can be approximately modified as

$$E(v,h) = \sum_i \frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_j b_j h_j - \sum_{i,j} w_{ij} \frac{v_i}{\sigma_i} h_j. \tag{4.32}$$

Then the conditional probability $p(h \mid v)$ is unchanged while $p(v \mid h)$ becomes Gaussian function whose mean is

$$a_i + \sigma_i \sum_j w_{ij} h_j. \tag{4.33}$$

and diagonal covariance matrix is

$$\begin{aligned} p(v_i = x \mid h) &= \frac{1}{\sigma_i \sqrt{2\pi}} . e^{-\frac{\left(x - a_i - \sigma_i \sum_j w_{ij} h_j\right)^2}{2\sigma_i^2}}, \\ p(h_j = 1 \mid v) &= sigm\Big(b_j + \sum_i \frac{v_i}{\sigma_i} w_{ij}\Big). \end{aligned} \tag{4.34}$$

4.4.2 Contrastive Divergence

In order to train RBMs as a probabilistic model, contrastive divergence algorithms is proposed [31]. Contrastive divergence has two phases which are the positive and negative phase. In positive phase the input vector is propagated from visible layer to the hidden layer. On the contrary in the negative stage the hidden vector is propagated back to the visible unit. Then the weights are updated

$$w(t+1) = w(t) + a(vh^T - \acute{v}\acute{h}^T) \text{ where } a \text{ is the learning rate.} \tag{4.35}$$

The purpose of using contrastive divergence is reducing the reconstruction error while trying to recreate the input data by using internal generated data. If reproduction of model is not close enough to the input data, model makes an adjustment and tries to recreate it with less error.

4.4.3 Deep Belief Networks

Like the stacked autoencoders, in the deep belief network layers are trained unsupervisedly. Deep belief networks need to be trained greedily, just one layer at a time. This structure helps to overcome the overfitting and vanishing gradient problems as we mentioned in the stacked autoencoders. The visible layer of each Restricted Boltzmann Machine is set to the hidden layer of the previous one. Contrastive divergence training is repeated for all the layers. Then similar to the stacked autoencoders, after the pre-training process, the model should be trained a supervised manner by using backpropagation algorithm. During fine tuning stage the connection between the layers should be arranged depending on the application.

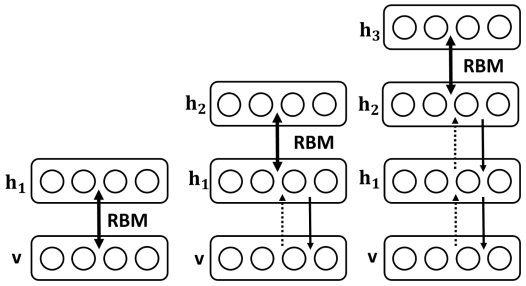


Figure 4.16: Deep Belief Network diagram (reproduced from [56])

If we want to summarize the training procedure:

- Construct an RBM with one input and one hidden layer.
- Train the constructed RBM in an unsupervised way.
- Add another hidden layer on top of the current RBM and construct a new RBM
- Fix the weights of the first RBM, take a sample from the hidden layer of the first RBM using the probability distribution and use this sample as an input to the new RBM
- Continue to adding new hidden layers on top of the network and train
- Lastly, perform the supervised fine-tuning to the formed neural network.

4.4.4 Deep Kernel Machines and Deep Convolutional Networks

Multilayer Kernel Machines (MKM) [18] can model non-linear function with the iterative procedure of basic kernel methods. Greedily unsupervised pre-training performed with Kernel Principle Component Analysis (KPCA) [52]. For supervised fine tuning stage it is common to use Kernel Partial Least Squares method to simplify or eliminate the cross validation stage.

Deep convolution network is a special kind of deep network designed for image processing tasks especially for visual detection and object recognition applications. Main idea of this network is combining local computations and pooling. Performing unsupervised pre-training can reduce the number of labelled samples needed in supervised stage. [36]

4.5 Proposed Framework

In Deep Network based algorithms we should first determine the feature length size because the number of nodes in the input layer should be equal to the attribute size of dataset. Then we have tried to construct the first network which have one hidden layer. To find optimal amount of nodes for the first network we have checked average

reconstruction errors for different number of nodes in the first hidden layer. The loss function used for the average reconstruction error in the unsupervised training procedure of both RBMs and Autoencoders was decided as LMS error defined in Equation 4.15. This equation is one of the common loss functions used in back-propagation and contrastive divergence. $\frac{1}{2}$ is included to the equation so as to simplify the gradient value. Reconstruction error checks the ability of reconstruction of input values of the previous layer after transforming in a compact form in the latter hidden layer.

While searching the optimal amount of nodes we had a limitation. The number of nodes in hidden layer should be equal or less than the number of nodes in input layer, because we have tried to represent inputs in more compact way at the first hidden layer. After finding the optimal number of nodes in the hidden layer and training the first network, the first network was ready to supervised fine tuning stage.

The second network has two hidden layers, the third one has three and so on. For the second network we should find the optimal number of nodes in the second hidden layer. The amount of nodes in the first hidden layer of the second network and the weights were already fixed in the unsupervised stage of the first network. The number of nodes in the second hidden layer should be less than or equal to the number of nodes in the first hidden layer, due to the same compactness reason. After fixing and training the second network using backpropagation or contrastive divergence, the second network was also ready to fine tuning stage. Illustrative diagram of proposed methods for the first two networks can be seen in Figure 4.17.

For each dataset we have different network options which have different depth. The number of nodes has decreased gradually in subsequent hidden layers of later networks as we can figure out. Since the number of nodes decreases through successive hidden layers gradually, we can get finite number of different architectures for each datasets.

The network that has the minimum validation error value was selected as the best winner architecture and the number of epochs giving the best validation error was decided as a stopping criterion parameter for fine tuning stage of testing set. Finally, the final tests with testing set were conducted by the model fixed on.

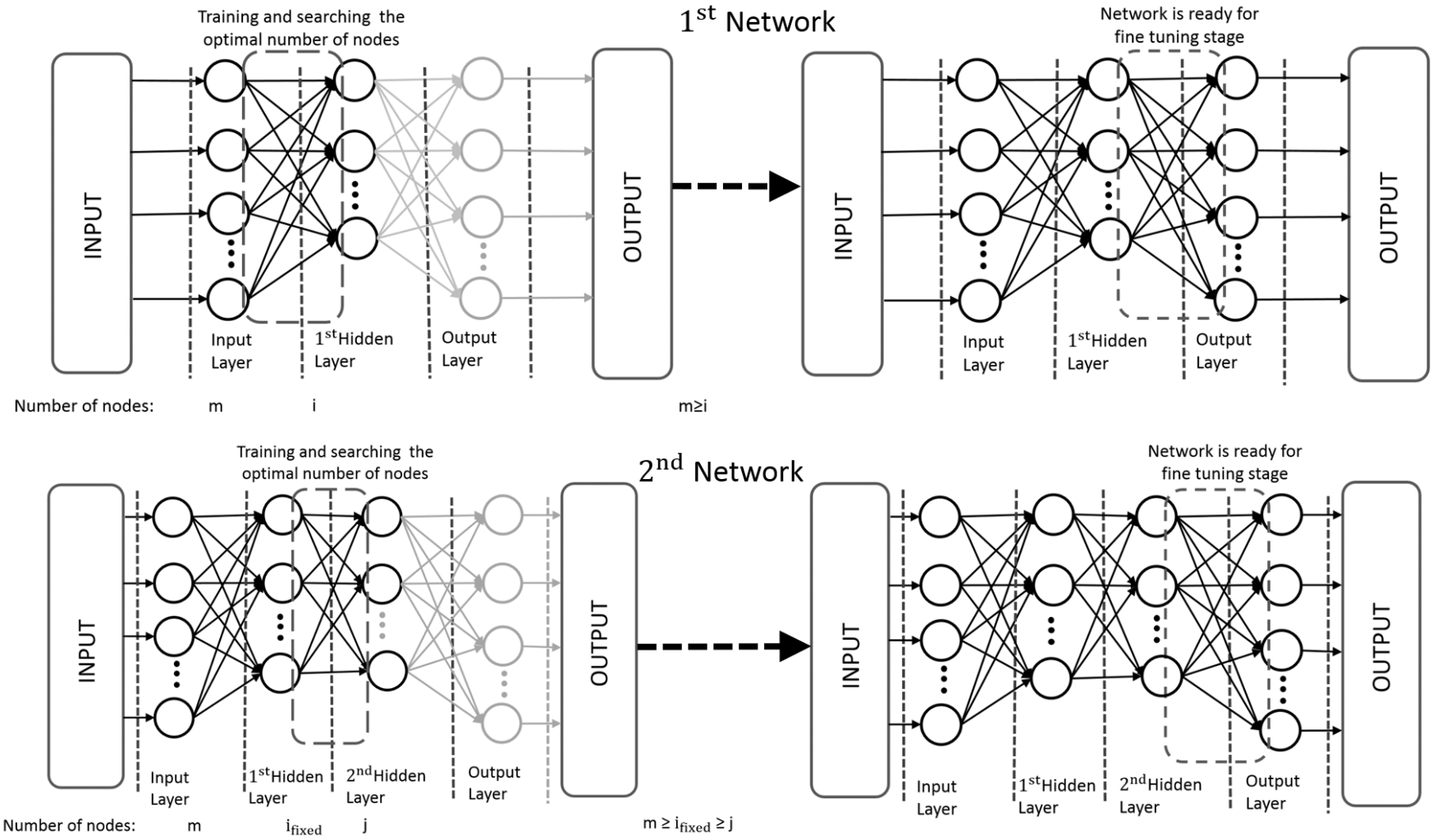


Figure 4.17: Illustrative diagram of proposed methods for the first two networks

If we want to summarize the unsupervised training procedure:

- Step 1-> Construct an RBM/AE with one input and one hidden layer.
- Step 2-> Add nodes to the hidden layer and train the constructed RBM/AE (up to the number of nodes in the previous layer).
- Step 3-> Measure reconstruction errors for each number of nodes in the hidden layer of RBM/AE.
- Step 4-> Find the network with minimum reconstruction error and fix the weights of the network.
- Step 5-> For the next network add another hidden layer on top of the current RBM/AE and construct a new RBM/AE. Then continue from Step 2.

If we want to summarize the unsupervised training procedure:

- After constructing and fixing the weights of networks with different depth, perform the supervised fine-tuning to formed neural networks.
- Lastly, pick the network with minimum validation set error as the best network.

CHAPTER 5

EXPERIMENTS AND RESULTS

5.1 Datasets and The System Used in Experiments

We have performed the comparative experiments on 4 different time series datasets taken from the UCR Time Series [16] dataset collection. All of the datasets in UCR Time Series repository are publicly available and labelled. From over 80 datasets we chose 4 different dataset gathered from various study fields. These datasets have different attribute length and different number of class quantities. Predefined sizes of both training and testing sets are also different. Selected datasets are commonly used to evaluate methods which are developed by researchers. As mentioned in the introduction chapter, while selecting these datasets, we have filtered the whole dataset list with respect to their warping window ratio and samples sizes. One of our filtering criteria is that sample size of datasets should be adequate for training stages of deep learning. Moreover warping window parameter has direct relevancy with the process time of DTW during constructing the pairwise distance matrix.

The proposed method and the benchmark methods are coded and implemented in Intel (R) Core ™ i7 3.60 GHz PC with 16.00 GB RAM using MATLAB R2014b 64-bit. For multi-class SVM algorithm Library for Support Vector Machines (LIBSVM) [15], which is C++ based software library, is used. To use this package in MATLAB, MATLAB executable (MEX) files are generated using MATLAB software development kit (SDK) compilers. During implementation of the deep learning based algorithms some source codes used in [47] are utilized.

The properties of datasets used in the experimental study are listed in Table 5.1

Table 5.1: Properties of Datasets Used in Experiments

Name	Number of classes	Size of training set	Size of testing set	Time series length
ElectricDevices	7	8926	7711	96
Wafer	2	1000	6174	152
Two Patterns	4	1000	4000	128
ProximalPhalanxOutlineCorrect	2	600	291	80

5.2 Data Preparation

Before starting the experimental study of the algorithms, the data in four different dataset must be prepared for the tests. All of the datasets consist of training and testing parts.

DTW with 1-NN classifier does not need validation set to optimize parameters. On the other hand multi-class SVM based and deep learning based algorithms have various parameters that needs to be tuned. In order to make use of the computed and reported results of datasets in [16], the training sets are left as they are, however the testing set is divided into two parts which are namely validation set and testing set. While dividing the testing sets into two parts we should be careful about some properties of the datasets. Firstly, the proportions of each class observations must be the same as much as possible in testing and validation sets after the division operation. The second key point is that the selection of samples for separation operation should be decided randomly. To produce comparative results datasets separated into validation and testing parts are used in all classification algorithms.

5.3 Measurements and Performance Metrics

The total classification performance can be calculated by comparing expected target labels with predicted target class labels. The detailed results can be observed from the confusion matrices whose each row represents the samples in an expected class and each column represents the samples in predicted classes. To express classification performances of each class in dataset, the precision and recall criteria will be examined for proposed methods. Precision can be defined as the fraction of positive results

that are correctly identified among the positive predictions while recall is fraction of positive results that are correctly identified among the real positive values.

$$Precision = \frac{TP}{TP + FP} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

where TP and FP stand for correctly labelled positive samples, and wrongly labelled positive samples respectively. FN symbolizes wrongly labelled negative observations.

Besides the classification performances of the proposed and benchmark methods, the computational time costs will be analyzed in this thesis.

5.4 Experimental Procedures and Performances of Benchmark Methods

Let's say we have m observations in a training set, n observations in testing set and the number of attributes equals to p. In that case we should solve the shortest path dynamic programming problems $(m * n)$ times for DTW with 1-NN classifier. We know that the number of attributes are the same in both training and testing sets, therefore the pxp sized square dynamic programming matrix is constructed for each of the $(m * n)$ distance comparison. By solving the shortest-pairwise distance problem with dynamic time warping, the distance map whose dimension is mxn is constructed. 1-NN classifier is the decision maker for assigning target values to every n test observations. Each testing sample gets the class label of a training sample whose pairwise distance is the smallest. Illustrative example can be seen in Figure 5.1 .

As mentioned in Chapter 3.1.1 , limited warping window parameters can be used to reduce the phase difference between the samples. While reducing the phase difference the total time consumption of DTW algorithm is also decreased, because there is no need to construct full pxp matrix any more. In limited warping window case, the minimum cost path and minimum distance may change. Therefore, the input of 1-NN classifier which is mxn distance matrix may change accordingly.

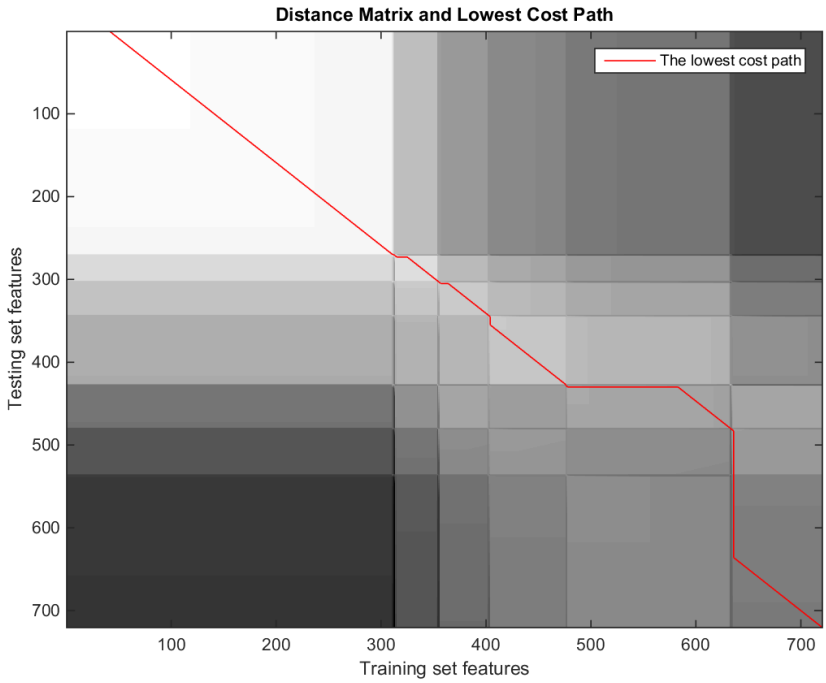


Figure 5.1: A dynamic programming solution (p=720 & Darker regions imply bigger distances)

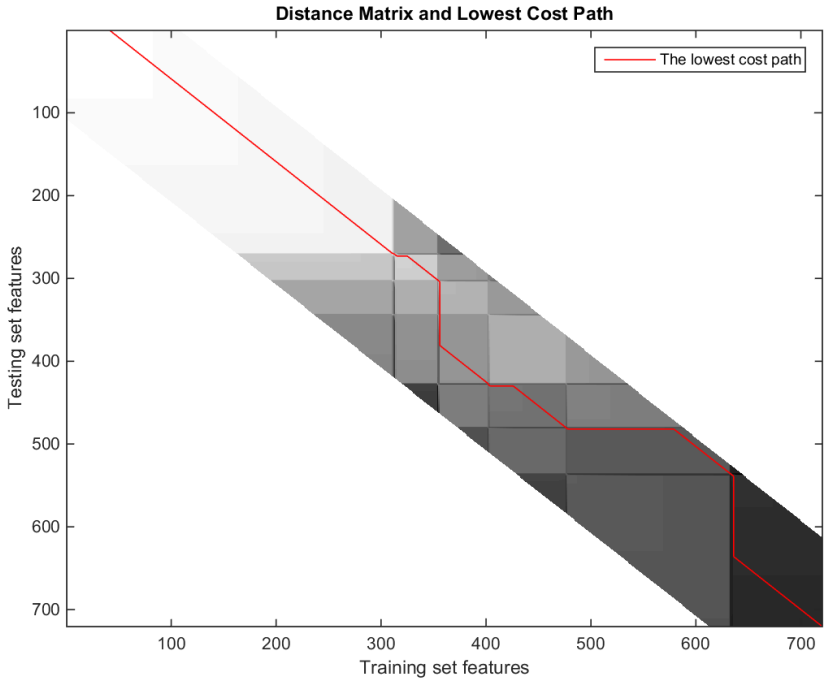


Figure 5.2: A dynamic programming solution (p=720 & Warping Window Parameter 15%)

In [16] warping window parameters were studied, from 0% to 100% , in 1% increments, searching for the warping window sizes giving the highest training performances. Then the warping window parameters which give the best classification performances were reported. In order to be able to compare computational times and classification performances we produced DTW with 1-NN and Euclidean Distance(ED) with 1-NN results of datasets. Due to time constraints and limited resources, we did not search the warping window parameters, but used the reported ones [16]. ED with 1-NN is one of the most basic and widely used classification methods, not just for time series but all kinds of data. While demonstrating the DTW with 1-NN classification accuracies, we also demonstrated the ED with 1-NN performances to see the effect of *elastic distance* measurement operation.

The warping window parameters, classification errors and time spent at the the DTW-1NN and ED-1NN methods are listed in Table 5.2 . Moreover, confusion matrices and individual class performances are given in Appendix A.2 and A.1 respectively.

SVM functions are supplied by well-designed LIBSVM library as black-box functions. For given datasets, and input parameters MEX files produce a multi-class SVM classifier model and optimizes the model parameters. Surprisingly, independent from the variations of the input parameters, very similar models were produced by the LIBSVM files. Hence the classification accuracies of the models on validation sets remain steady in spite of the change in the input parameters.

During the training of SVM classifier, the model is optimized internally without any external interaction. Due to proprietary properties of LIBSVM, workspace variables and intermediate level outputs cannot be analyzed.

Because we could not tune the parameters with respect to the change in validation set error, it is unnecessary and pointless to give the elapsed duration at parameter tuning phase.

Optimum model training time and elapsed time at final classification testing were reported with the validation set and error set classification errors for each dataset in Table 5.3 . Moreover the number of support vectors was specified for all datasets in the same table. Detailed confusion matrices, precision and recall values of each target

Table 5.2: Best Classification Results and Warping Window Parameters of DTW

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
Warping Window Parameters	14 %	1%	4%	1%
DTW with 1-NN error	37.6%	0.5%	2.5%	20.0%
Time Spent(sec)	$1.387x10^6$	$3.571x10^4$	$1.728x10^4$	176.012
ED with 1-NN error	45.8%	2.6%	6.9%	18.6%
Time Spent(sec)	3.72	0.465	0.236	0.011

class of dataset are provided in Appendix A.4

5.5 Experimental Procedures and Performances of Deep Learning Approaches

Before trying to train the first RBM of Deep Belief Network and the first Autoencoder of SAE, the batch sizes for training process should be decided. Batch training concept makes it possible to obtain more robust models since it allows training with a bunch of samples in each iteration. In batch training with respect to the average batch error the weights and bias values are adjusted. The effect of the batch size which is one of the hyper-parameters on the training performance is not a focus of this work, thus not extensively studied. The batch sizes for each dataset were chosen intuitively by examining the training set size of the datasets. We know that the training size of the dataset should be a multiple of the mini-batch size, a few training observations were discarded from the set randomly.

In supervised learning the number of epochs is determined with respect to the point which the validation error starts to increase firmly. Thus we can prevent the over-fitting and over usage of resources. Although in unsupervised training process the target values are not defined and the classification error does not exist, we have the average reconstruction error. As mentioned in Chapter 4.2 , we have tried to represent the input weights in a more compact way in the hidden layers. Hence the minimum reconstruction error is the only criteria for the stopping criteria of epochs during unsupervised learning. When the average reconstruction error of an iteration reaches to saturated limit values, the stopping criteria for the number of iterations is obtained.

The remaining hyper-parameters used in unsupervised learning which are the learning rate and momentum were found by grid search method. Grid search tries every single value can be taken by the hyper-parameter over a specified range. The search was applied to each of the remaining parameters individually.

While sampling and assigning observations to the mini-batches and updating some weighs and biases of networks random number generator is used. Therefore the random seeds have slight effect on the classification results. To compare performances of different network architectures, we initialize the random seed in training processes.

Table 5.3: Parameters, Properties and Results of SVM Classifier Model

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
The number of support vectors	6593	96	921	394
Time elapsed at training of the SVM classifier(sec)	53.020	0.155	1.143	0.153
Time elapsed at testing stage(sec)	3.679	0.053	0.256	0.001
Validation set error	0.393	0.010	0.148	0.192
Testing set classification error	**0.404**	**0.004**	**0.060**	**0.152**

As mentioned in Sections 4.4 and 4.4.3, both deep belief networks and stack autoencoders need to be trained greedily with just one layer at a time. The hidden layer of an autoencoder or a RBM acts as an input layer of the later autoencoder or RBM. To overcome the overfitting phenomena and represent the inputs in more compact way, the number of nodes in each hidden layer should be less than or equal to the number of nodes in input layer. As a matter of fact, if the number of nodes equals in hidden and input layer, we should be careful that the transfer function does not become an identity function.

The number of nodes in hidden layers is found by inspecting the minimum reconstruction errors in unsupervised training stage. During the training of the first RBM, we construct networks so that the number of nodes in the hidden layer varies from 1 to the number of nodes in input layer of the first RBM. Then these networks are trained in an unsupervised way and the one that has minimum average reconstruction error was chosen as a candidate for the best network for fine tuning stage. After fixing the number of nodes in the first RBM, we tried to get the second candidate network that has two hidden layers with same procedure. The quantities of nodes in the input and hidden layers are shown in Tables 5.4 to 5.11 . The first network has one hidden layer to be trained in supervised tuning phase, the second network has two hidden layers to be trained, and so on. The times elapsed at the unsupervised learning procedure is also demonstrated in Tables 5.4 to 5.11 .

Table 5.4: Number of Nodes in Each Layer and Elapsed Time at the DBN Based Unsupervised Learning Phase /for ElectricDevices dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	96	80	$0.110\text{x}10^5$
$2^{nd}Network$	96	80 42	$0.302\text{x}10^5$
$3^{rd}Network$	96	80 42 35	$0.436\text{x}10^5$
$4^{th}Network$	96	80 42 35 35	$0.576\text{x}10^5$
$5^{th}Network$	96	80 42 35 35 18	$0.744\text{x}10^5$
$6^{th}Network$	96	80 42 35 35 18 16	$0.840\text{x}10^5$
$7^{th}Network$	96	80 42 35 35 18 16 8	$0.929\text{x}10^5$
$8^{th}Network$	96	80 42 35 35 18 16 8 8	$0.978\text{x}10^5$
$9^{th}Network$	96	80 42 35 35 18 16 8 8 3	$1.028\text{x}10^5$
$10^{th}Network$	96	80 42 35 35 18 16 8 8 3 2	$1.051\text{x}10^5$
$11^{th}Network$	96	80 42 35 35 18 16 8 8 3 2 1	$1.069\text{x}10^5$

Table 5.5: Number of Nodes in Each Layer and Elapsed Time at the DBN Based Unsupervised Learning Phase /for Wafer dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	150	120	0.349x10^3
$2^{nd}Network$	150	120 31	0.908x10^3
$3^{rd}Network$	150	120 31 23	1.077x10^3
$4^{th}Network$	150	120 31 23 22	1.224x10^3
$5^{th}Network$	150	120 31 23 22 19	1.378x10^3
$6^{th}Network$	150	120 31 23 22 19 16	1.528x10^3
$7^{th}Network$	150	120 31 23 22 19 16 7	1.662x10^3

Table 5.6: Number of Nodes in Each Layer and Elapsed Time at the DBN Based Unsupervised Learning Phase /for Two Patterns dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	128	98	0.642x10^3
$2^{nd}Network$	128	98 86	1.622x10^3
$3^{rd}Network$	128	98 86 75	2.839x10^3
$4^{th}Network$	128	98 86 75 53	4.213x10^3
$5^{th}Network$	128	98 86 75 53 44	5.359x10^3
$6^{th}Network$	128	98 86 75 53 44 13	6.433x10^3
$7^{th}Network$	128	98 86 75 53 44 13 8	6.789x10^3
$8^{th}Network$	128	98 86 75 53 44 13 8 1	7.091x10^3

Table 5.7: Number of Nodes in Each Layer and Elapsed Time at the DBN Based Unsupervised Learning Phase /for ProximalPhalanxOutlineCorrect dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	80	16	0.210x10^3
$2^{nd}Network$	80	16 9	0.26x10^3
$3^{rd}Network$	80	16 9 1	0.298x10^3

Table 5.8: Number of Nodes in Each Layer and Elapsed Time at the SAE Based Unsupervised Learning Phase /for ElectricDevices dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	96	96	0.101x10^3
$2^{nd}Network$	96	96 92	0.317x10^3
$3^{rd}Network$	96	96 92 80	0.623x10^3
$4^{th}Network$	96	96 92 80 65	0.962x10^3
$5^{th}Network$	96	96 92 80 65 60	1.293x10^3
$6^{th}Network$	96	96 92 80 65 60 60	1.644x10^3
$7^{th}Network$	96	96 92 80 65 60 60 59	2.045x10^3

Table 5.9: Number of Nodes in Each Layer and Elapsed Time at the SAE Based Unsupervised Learning Phase /for Wafer dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	152	44	0.480x10^4
$2^{nd}Network$	152	44 39	0.656x10^4
$3^{rd}Network$	152	44 39 11	0.864x10^4
$4^{th}Network$	152	44 39 11 9	0.932x10^4
$5^{th}Network$	152	44 39 11 9 8	0.996x10^4
$6^{th}Network$	152	44 39 11 9 8 5	1.059x10^4
$7^{th}Network$	152	44 39 11 9 8 5 2	1.104x10^4

Table 5.10: Number of Nodes in Each Layer and Elapsed Time at the SAE Based Unsupervised Learning Phase /for Two Patterns dataset/

	Input Layer	Hidden Layers	Elapsed Time(sec)
$1^{st}Network$	128	128	0.320x10^4
$2^{nd}Network$	128	128 39	1.037x10^4
$3^{rd}Network$	128	128 39 34	1.493x10^4
$4^{th}Network$	128	128 39 34 32	1.948x10^4
$5^{th}Network$	128	128 39 34 32 21	2.067x10^4
$6^{th}Network$	128	128 39 34 32 21 21	2.148x10^4
$7^{th}Network$	128	128 39 34 32 21 21 21	2.196x10^4

Table 5.11: Number of Nodes in Each Layer and Elapsed Time at the SAE Based Unsupervised Learning Phase /for ProximalPhalanxOutlineCorrect dataset/

	Input Layer	Hidden Layers			Elapsed Time(sec)
$1^{st}Network$	80	73	9		$1.095\text{x}10^3$
$2^{nd}Network$	80	73	9		$3.092\text{x}10^3$
$3^{rd}Network$	80	73	9	4	$3.372\text{x}10^3$

Even though larger networks can learn and model the data better, the larger networks do not always give better results due to vanishing gradient problem. In those cases the supervised training procedures of the deeper networks were cancelled due to resource and time limitations. Therefore the properties of untrainable networks have not been demonstrated after a certain level in the Tables 5.4 to 5.11 .

Among all valid unsupervisedly-trained networks with different depths the best networks were chosen by using validation set. After the initialization step of the weights and biases the supervised fine tuning process was applied via the *back-propagation algorithm*. During the fine tuning phase, the number of epochs was determined by monitoring the validation set error instead of training set error. Related error vs the number of epochs diagrams can be examined in Appendix A . The best networks for both DBN and SAE based classifiers for all datasets were marked in Tables 5.4 to 5.11 with darker color.

The network that has the minimum validation error value was selected as the best winner architecture and the number of epochs giving the best validation error was decided as a stopping criterion parameter for fine tuning stage of testing set. Finally, the final tests with testing set were conducted by the model fixed on.

The properties of winner networks and supervised tuned parameters were listed below in Table 5.13 and Table 5.12 . Cumulative times required to specify the best model and parameters at supervised stage and the elapsed time during the final tests with fixed models are listed in Table 5.13 and Table 5.12 .

Visual illustration of some of the correctly-classified and all misclassified samples of Wafer dataset are shown in Figure 5.3 and Figure 5.4 respectively. We observed that misclassified samples of the 2^{nd} class of the Wafer dataset have nearly same

Table 5.12: Parameters and Properties of SAE Based Networks at Fine Tuning Phase

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
Network Order	2^{nd}	3^{rd}	6^{th}	1^{st}
Minimum validation set error	0.339	0.014	0.149	0.130
Epochs when validation error reaches minimum value	39	196	585	963
Time elapsed at model and parameter search(sec)	$1.144\text{x}10^5$	$1.555\text{x}10^3$	$3.566\text{x}10^4$	$2.680\text{x}10^4$
Time elapsed at fine tuning of the winner network(sec)	4.960	2.196	12.142	5.086
Time elapsed at final testing stage(sec)	0.019	0.011	0.019	0.006

Table 5.13: Parameters and Properties of DBN Based Networks at Fine Tuning Phase

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
Network Order	3^{rd}	1^{st}	2^{nd}	1^{st}
Minimum validation set error	0.324	0.026	0.194	0.164
Epochs when validation error reaches minimum value	47	205	316	597
Time elapsed at model and parameter search(sec)	$1.476\text{x}10^5$	$3.451\text{x}10^3$	$4.532\text{x}10^3$	$6.313\text{x}10^3$
Time elapsed at fine tuning of the winner network(sec)	6.011	2.134	4.301	2.238
Time elapsed at final testing stage(sec)	0.015	0.012	0.012	0.008

characteristics.

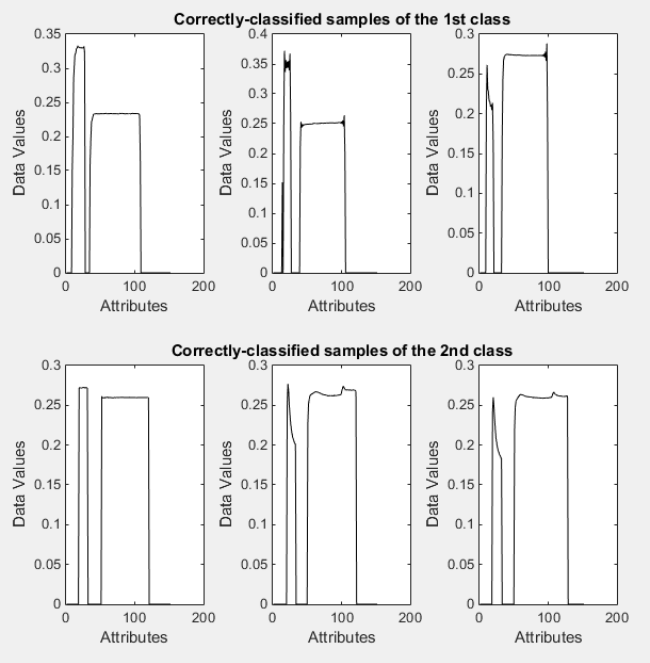


Figure 5.3: Some of the correctly-classified samples of Wafer dataset

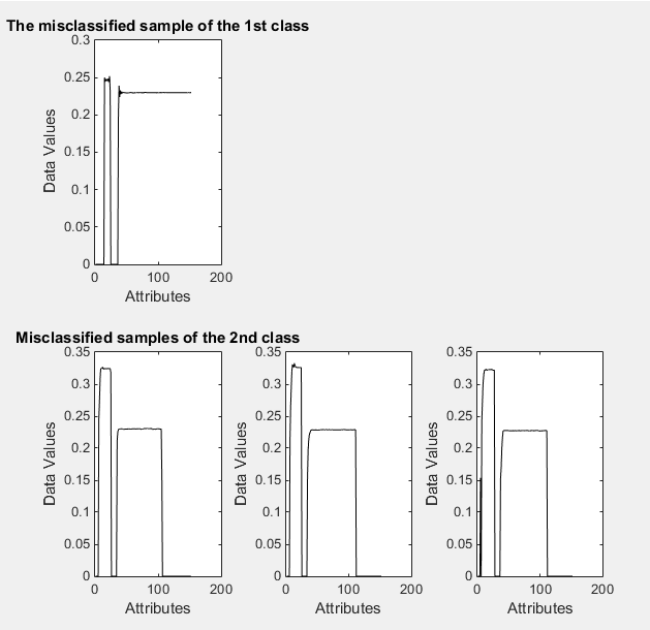


Figure 5.4: Misclassified samples of Wafer dataset

We constructed the table which shows the comparative classification performances of proposed and benchmark methods. The minimum error values in Table 5.14 have been marked as bold for all the datasets. Confusion matrices and individual class

performances can be seen from Appendix A.4 and A.5 for SAE based proposed method and DBN based proposed method respectively.

Lastly we constructed the table that demonstrates total computational times of proposed deep learning methods and DTW with 1-NN in Table 5.15 .

While analyzing the Table 5.15 a critical issue we should be aware of that we have used pre-measured parameters for DTW- with 1-NN algorithm. If we tried to measure optimal warping window parameters of DTW, the durations would increase up to hundreds of times of the current durations.

5.6 Discussions

SAE or DBN based deep learning methods have given the best performances among all five classification methods. SAE based proposed method outperformed the other algorithms in ElectricDevices and ProximalPhalanxOutlineCorrect dataset and DBN based algorithm surpassed the rest of the methods in Wafer and Two Pattern datasets. In the classification task of Wafer dataset, the proposed models have shown drastic improvement but for the rest of datasets deep network based algorithms have produced slight progress in error rates.

Due to the fact that in time series classification tasks the imbalanced class distributions cases are very common, we should not evaluate the methods only with the total classification error rates. Imbalanced data means sets do not have equal number of observations in each class. To understand the capability of the models we have decided to use the precision and recall metrics for imbalanced-class cases. Basically the precision is an accuracy and the recall is a sensitivity related metrics. The precision gives us the ratio of how many predicted items are relevant to the expected class and the recall gives us the ratio of how many relevant items are labelled as target value of predicted class.

The 6^{th} class of the ElectricDevices and $4th^{th}$ class of Two Pattern datasets have gotten the highest precision and recall values when classified by DTW-1NN model. So if we want to classify or identify these classes particularly, in spite of the higher total

Table 5.14: Classification Performances of Proposed and Benchmark Methods

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
ED with 1-NN	45.8%	2.6%	6.9%	18.6%
DTW with 1-NN	37.6%	0.5%	2.5%	20.0%
SVM	40.4%	0.4%	6.0%	15.2%
SAE based Proposed Method	**34.1**%	3.0%	2.8%	**13.0**%
DBN based Proposed Method	36.1%	**0.1**%	**2.3**%	15.2%

Table 5.15: Total Time Comparison of DTW 1-NN and Proposed Methods

	ElectricDevices	Wafer	Two Patterns	ProximalPhalanxOutlineCorrect
DTW with 1-NN	$1.387x10^6$	$3.571x10^4$	$1.728x10^4$	176.012
SAE based Proposed Method	$1.164x10^5$	$1.260x10^4$	$5.762x10^4$	$3.017x10^4$
DBN based Proposed Method	$2.545x10^5$	$5.113x10^3$	$1.162x10^4$	$6.661x10^3$

classification error, DTW-1NN model should be considered as the best option. Furthermore, the sensitivity or accuracy of the one class should be taking into account while choosing the model. Such as if we do not want to miss even a single observation for a class, we should analyze the recall metrics of that class. 2^{nd} class of the ProximalPhalanxOutlineCorrect dataset with DBN based method has the highest recall performance of any algorithm can produce ever. Hence regardless the importance of any false alarms, if we do not want to miss a single entry of the 2^{nd} class of ProximalPhalanxOutlineCorrect dataset we should choose DBN based network as the classifier. Appendix A contains the detailed precision and recall results of each methods and each class.

To inspect every aspect of the dataset properties, four disparate datasets have been selected from the UCR database set. As mentioned earlier, one of our criteria while selecting these datasets was that we need smaller warping window parameters to complete the DTW-1NN test physically. To demonstrate the workload of DTW and the importance of warping window parameter ElectricalDevices dataset with 14% warping window ratio was selected. Even for 14% warping window ratio which means constructing about one fourth of the pairwise DTW distance matrix, the test has taken more than 16 days. In all of the DTW computations parallel CPU programming techniques were used. Without parallel programing and smaller warping windows parameters, tests would not be completed in months. Moreover, we did not search for the best warping window parameters due to the resource and time constraints. Obviously the most time consuming classification method is DTW-1NN and the least time consuming one is ED-1NN due to simplicity of the algorithm. For one of the dataset, ProximalPhalanxOutlineCorrect, ED-1NN is more successful than the DTW-1NN model, however for the rest of the datasets DTW outperform due to elasticity property of the DTW metric. While giving the better performance, DTW consumes much more time than ED based classifier obviously.

Produced DTW-1NN performances show very close similarities with the reported performances in the [16] even the testing set divided into validation and new testing groups for parameter search stage of classifiers. This fact cannot prove but strongly support the idea that the random seeds have insignificant effect to the classification performances.

SVM based classifiers are the least sensible method to the model parameters. In fact the models produced by the algorithms have slight change hence the classification performances have not varied with the different input parameters. Because the parameter tuning stage is not valid for SVM based algorithms we can compare only the optimal model training and final test durations with the proposed methods. Training time of the proposed methods' models with fixed parameters are less than the training time of the SVM based model while final testing times of proposed and SVM based algorithms are quite similar.

Computational times of SAE and DBN based algorithms shows similarity in supervised phases. However the unsupervised phase of SAE trainings takes more time than DBN. This duration gap is caused by the working principle of contrastive diverge and backpropagation. Our study is not suitable to give exact mathematical comparison between the contrastive diverge and backpropogation algorithms, but intuitively we can say contrastive divergence method trains the model faster with mini-batches. For both of the proposed algorithm deeper networks takes more time in both unsupervised and supervised phases. In fact, in fine tuning stage the deeper network could not be trained due to vanishing gradient phenomenon. We can examine these fact from the supervised training phase diagrams in Appendix B . For instance, 5th and deeper networks of the SAE and DBN based approaches begin failing on the supervised training stage of the ElectricalDevices dataset. The overfitting issue has also been observed during the supervised classification processes. The second network of SAE based approaches and the third network of DBN based architectures of ElecrticalDevices dataset shows the overfitting issue clearly. After a certain point the validation error starts increasing due to outlier or anomalous samples while the classification error of the training dataset continues decreasing. To prevent models from overtraining we should examine the validation set errors after the training process of each epochs.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Summary

In this work, we explored the applicability of deep learning idea to time series datasets and we tried to propose novel methods for classification task of time series. Our main goal is to outperform the benchmark methods and classical approaches. While producing deep learning based classification methods, we have also analyzed the feasibility of applying deep architectures to the differential time series datasets. Prior information was not used in any step of the procedure of proposed algorithms. We have constructed various depth sized networks for both stacked autoencoder and deep belief network by inspecting the average total reconstruction errors automatically in unsupervised stage. Then by monitoring validation set supervised training errors, we compared the networks in terms of accuracy and elapsed time in training for each dataset. From beginning to the end all phases of proposed and benchmark methods have been executed without any external interaction except one condition. When the vanishing gradient phenomenon prevented the improvement of the classification process, the training procedures of larger networks were interrupted.

The performances indicate that it is possible to get a reasonable accuracy even with only one-hidden-layer network for some datasets. However depending on the complexity of the dataset the optimum number of hidden layer can vary for both of the proposed methods and every datasets. One of the important implications from the results of proposed methods is that models have not failed even for highly imbalanced dataset.

Although the main decision criteria is total classification error in this study, we should observe the related precision and recall ratios that should have an impact on classifier selection. We have figured out that if we want to classify a specific class with higher accuracy or sensitivity expectation, the classifier can be chosen with respect to the class-wise performance. In fact we can design a hybrid classifier which fuses the proposed and benchmark methods on specified recall and precision demands.

Due to the size of the datasets and complexity of DTW algorithm, the whole dynamic programming computations have parallelized. However, due to the architecture of source codes taken by [47], the proposed methods could not be optimized or parallelized in this work. After reimplementation of the source codes, drastic reduction in computational times may be observed.

6.2 Future Work

This study can be considered as the first phase of an optimized robust time series classifier system based on deep learning architectures. If we want to develop an optimized universal solution to the time series classification, the proposed model should be tested using different datasets with varying properties in terms of the distribution of samples and the intrinsic characteristics of the data itself. Unity tests should be applied to the proposed algorithms to ensure that the bug-free library has been produced in this work. Moreover some source codes will be implemented again to make the algorithms become suitable for parallel computing techniques.

To eliminate slight effects of the random seed on hyper-parameters selection a set of random seed should be used instead of one. However due to the time and resource restriction, this step is postponed until the reimplementation of source codes used in study. The optimization of hyperparameters has very crucial effect on the performance of the proposed methods. Hence more systematic and detailed parameter search techniques will be integrated to the developed models so as to increase the classification ability. Besides the short term improvements, the applicability of deep architecture to LSTM networks can be investigated in long term.

While implementing DTW with 1-NN, we have also implemented the DDTW and

WDTW classifiers and tested them in sample time series. However there is no a reported study for warping window parameters for DDTW and WDTW. Moreover we do not have enough computational resources to search the parameters of these algorithms. Hence we could not have worked with these classifiers in our study. If we have opportunity of using computers having more computational ability, we can compare our proposed methods with these additional classification methods.

REFERENCES

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[2] H.-M. Adorf. Connectionism and neural networks. In *Knowledge-Based Systems in Astronomy*, pages 213–245. Springer, 1989.

[3] E. Aleskerov, B. Freisleben, and B. Rao. Cardwatch: A neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 220–226. IEEE, 1997.

[4] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy. An introduction to deep learning. In *ESANN*, 2011.

[5] C. Bahlmann, B. Haasdonk, and H. Burkhardt. Online handwriting recognition with support vector machines-a kernel approach. In *Frontiers in handwriting recognition, 2002. proceedings. eighth international workshop on*, pages 49–54. IEEE, 2002.

[6] R. E. Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 2015.

[7] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[8] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[9] Y. Bengio, O. Delalleau, and N. L. Roux. The curse of highly variable functions for local kernel machines. In *Advances in neural information processing systems*, pages 107–114, 2005.

[10] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[11] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[12] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3d models. In *International Conference on Artificial Neural Networks*, pages 251–256. Springer, 1996.

[13] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

[14] E. Caiani, A. Porta, G. Baselli, M. Turiel, S. Muzzupappa, F. Pieruzzi, C. Crema, A. Malliani, and S. Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *Computers in Cardiology 1998*, pages 73–76. IEEE, 1998.

[15] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[16] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The ucr time series classification archive, July 2015.

[17] Y. Chen, M. A. Nascimento, B. C. Ooi, and A. K. Tung. Spade: On shape-based pattern detection in streaming time series. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 786–795. IEEE, 2007.

[18] Y. Cho and L. K. Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[19] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[20] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. John Wiley & Sons, 2012.

[21] D. R. Eads, D. Hill, S. Davis, S. J. Perkins, J. Ma, R. B. Porter, and J. P. Theiler. Genetic algorithms and support vector machines for time series classification. In *International Symposium on Optical Science and Technology*, pages 74–85. International Society for Optics and Photonics, 2002.

[22] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[23] E. Fix and J. L. Hodges Jr. Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, DTIC Document, 1951.

[24] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[25] D. Gavrila, L. Davis, et al. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *International workshop on automatic face-and gesture-recognition*, pages 272–277. Citeseer, 1995.

[26] K. Gollmer and C. Posten. Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. *On-line fault detection and supervision in chemical process industries*, 1995.

[27] S. Gudmundsson, T. P. Runarsson, and S. Sigurdsson. Support vector machines and dynamic time warping for time series. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2772–2776. IEEE, 2008.

[28] S. Haykin and N. Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[29] G. F. Hepner. Artificial neural network classification using a minimal training set. comparison to conventional supervised classification. *Photogrammetric Engineering and Remote Sensing*, 56(4):469–473, 1990.

[30] G. E. Hinton. Connectionist learning procedures. artificial intelligence, 40 1-3: 185 234, 1989. reprinted in j. carbonell, editor,". *Machine Learning: Paradigms and Methods", MIT Press*, 1990.

[31] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[32] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[33] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[34] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9):2231–2240, 2011.

[35] M. I. Jordan. Attractor dynamics and parallellism in a connectionist sequential machine. 1986.

[36] K. Kavukcuoglu, R. Fergus, Y. LeCun, et al. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1605–1612. IEEE, 2009.

[37] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[38] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *Sdm*, volume 1, pages 5–7. SIAM, 2001.

[39] A. Krenker, A. Kos, and J. Bešter. *Introduction to the artificial neural networks*. INTECH Open Access Publisher, 2011.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[41] M. Längkvist, L. Karlsson, and A. Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[42] M. McCloskey. Networks and theories: The place of connectionism in cognitive science. *Psychological Science*, 2(6):387–395, 1991.

[43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[44] M. D. Morse and J. M. Patel. An efficient and accurate method for evaluating time series similarity. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 569–580. ACM, 2007.

[45] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pages 511–520. IEEE, 1997.

[46] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *International Conference on Artificial Neural Networks*, pages 999–1004. Springer, 1997.

[47] R. B. Palm. Prediction as a candidate for learning deep hierarchical models of data, 2012.

[48] S. Rüping. Svm kernels for time series analysis. Technical report, Universitätsbibliothek Dortmund, 2001.

[49] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[50] P. Schiilkop, C. Burgest, and V. Vapnik. Extracting support data for a given task. In *Proceedings of the 1st International Conference on Knowledge Discovery & Data Mining*, pages 252–257, 1995.

[51] M. D. Schmill, T. Oates, and P. R. Cohen. Learned models for continuous planning. In *AISTATS*, 1999.

[52] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[53] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[54] H. A. Song and S.-Y. Lee. Hierarchical representation using nmf. In *International Conference on Neural Information Processing*, pages 466–473. Springer, 2013.

[55] S. Theodoridis and K. Koutroumbas. Pattern recognition., 1999.

[56] I. Vasiliyev. A deep learning tutorial: From perceptrons to deep networks. https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks, 2016. [Online; accessed 08-July-2016].

[57] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

[58] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE, 2002.

[59] H. Wakuya and K. Shida. Bi-directionalization of neural computing architecture for time series prediction. iii. application to laser intensity time record "data set a". In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 2098–2103. IEEE, 2001.

[60] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[61] J. Weston and C. Watkins. Multi-class support vector machines. Technical report, Citeseer, 1998.

[62] Y. Wu and E. Y. Chang. Distance-function design and fusion for sequence data. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 324–333. ACM, 2004.

[63] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning*, pages 1033–1040. ACM, 2006.

[64] Q. Yang and X. Wu. 10 challenging problems in data mining research. *International Journal of Information Technology & Decision Making*, 5(04):597–604, 2006.

[65] D. Yu, L. Deng, and D. Yu. Deep learning methods and applications. *Foundations and Trends in Signal Processing*, 2014.

[66] D. Zhang, W. Zuo, D. Zhang, and H. Zhang. Time series classification using support vector machine with gaussian elastic metric kernel. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 29–32. IEEE, 2010.

[67] J. M. Zurada. *Introduction to artificial neural systems*, volume 8. West St. Paul, 1992.

APPENDIX A

CONFUSION MATRICES AND DETAILED PERFORMANCES

A.1 Confusion Matrices & Precision and Recall Values for ED with 1NN Classifier

Table A.1: Confusion Matrix of ElectricDevices Dataset

		Predicted Classes						
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Actual Classes	1^{st} Class	49	2	7	61	133	73	8
	2^{nd} Class	1	661	7	112	189	7	1
	3^{rd} Class	24	0	200	44	20	41	48
	4^{th} Class	32	0	52	334	99	53	12
	5^{th} Class	25	97	12	80	667	20	33
	6^{th} Class	49	0	50	80	41	124	27
	7^{th} Class	5	1	10	46	159	4	53

Table A.2: Precision and Recall Values of ElectricDevices Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Precision	0.265	0.869	0.592	0.441	0.510	0.385	0.291
Recall	0.147	0.676	0.531	0.574	0.714	0.334	0.191

Table A.3: Confusion Matrix of Wafer Dataset

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	302	0
	2^{nd} Class	81	2698

Table A.4: Precision and Recall Values of Wafer Dataset

	1^{st} Class	2^{nd} Class
Precision	0.789	1.000
Recall	1.000	0.971

Table A.5: Confusion Matrix of Two Patterns Dataset

		Predicted Classes			
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Actual Classes	1^{st} Class	488	14	12	3
	2^{nd} Class	18	468	0	19
	3^{rd} Class	14	0	468	15
	4^{th} Class	1	27	15	436

Table A.6: Precision and Recall Values of Two Patterns Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Precision	0.937	0.919	0.946	0.922
Recall	0.944	0.927	0.942	0.910

Table A.7: Confusion Matrix of ProximalPhalanxOutlineCorrect

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	28	18
	2^{nd} Class	9	90

Table A.8: Precision and Recall Values of ProximalPhalanxOutlineCorrect

	1^{st} Class	2^{nd} Class
Precision	0.765	0.820
Recall	0.565	0.919

A.2 Confusion Matrices & Precision and Recall Values for DTW with 1NN Classifier

Table A.9: Confusion Matrix of ElectricDevices Dataset

		Predicted Classes						
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Actual Classes	1^{st} Class	80	0	17	80	104	41	11
	2^{nd} Class	0	685	0	24	268	0	1
	3^{rd} Class	5	0	264	9	6	8	85
	4^{th} Class	33	0	74	332	105	23	15
	5^{th} Class	7	58	7	78	761	2	21
	6^{th} Class	42	0	8	60	51	204	5
	7^{th} Class	2	4	4	11	170	2	85

Table A.10: Precision and Recall Values of ElectricDevices Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Precision	0.473	0.917	0.704	0.559	0.516	0.729	0.381
Recall	0.240	0.700	0.700	0.561	0.815	0.550	0.306

Table A.11: Confusion Matrix of Wafer Dataset

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	302	0
	2^{nd} Class	14	2715

Table A.12: Precision and Recall Values of Wafer Dataset

	1^{st} Class	2^{nd} Class
Precision	0.956	1.000
Recall	1.000	0.995

Table A.13: Confusion Matrix of Two Patterns Dataset

		Predicted Classes			
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Actual Classes	1^{st} Class	506	10	0	0
	2^{nd} Class	13	492	0	0
	3^{rd} Class	5	21	471	0
	4^{th} Class	0	0	0	479

Table A.14: Precision and Recall Values of Two Patterns Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Precision	0.966	0.941	1.000	1.000
Recall	0.981	0.9743	0.948	1.000

Table A.15: Confusion Matrix of ProximalPhalanxOutlineCorrect

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	26	20
	2^{nd} Class	9	90

Table A.16: Precision and Recall Values of ProximalPhalanxOutlineCorrect

	1^{st} Class	2^{nd} Class
Precision	0.743	0.818
Recall	0.565	0.909

A.3 Confusion Matrices & Precision and Recall Values for Multi-Class SVM Classifier

Table A.17: Confusion Matrix of ElectricDevices Dataset

		Predicted Classes						
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Actual Classes	1^{st} Class	35	11	1	84	154	38	10
	2^{nd} Class	2	709	9	21	228	1	8
	3^{rd} Class	10	9	238	35	9	12	64
	4^{th} Class	25	12	35	424	59	20	7
	5^{th} Class	11	86	4	57	745	6	25
	6^{th} Class	36	30	44	106	40	87	28
	7^{th} Class	3	11	19	69	114	4	58

Table A.18: Precision and Recall Values of ElectricDevices Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Precision	0.287	0.817	0.680	0.533	0.552	0.518	0.290
Recall	0.105	0.725	0.631	0.729	0.798	0.235	0.209

Table A.19: Confusion Matrix of Wafer Dataset

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	302	0
	2^{nd} Class	13	2766

Table A.20: Precision and Recall Values of Wafer Dataset

	1^{st} Class	2^{nd} Class
Precision	0.959	1.000
Recall	1.000	0.995

Table A.21: Confusion Matrix of Two Patterns Dataset

		Predicted Classes			
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Actual Classes	1^{st} Class	487	14	16	0
	2^{nd} Class	9	478	2	16
	3^{rd} Class	11	1	462	23
	4^{th} Class	0	11	17	451

Table A.22: Precision and Recall Values of Two Patterns Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Precision	0.961	0.948	0.930	0.920
Recall	0.942	0.947	0.930	0.942

Table A.23: Confusion Matrix of ProximalPhalanxOutlineCorrect

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	29	17
	2^{nd} Class	5	94

Table A.24: Precision and Recall Values of ProximalPhalanxOutlineCorrect

	1^{st} Class	2^{nd} Class
Precision	0.853	0.847
Recall	0.630	0.950

A.4 Confusion Matrices & Precision and Recall Values for SAE Based Classifier

Table A.25: Confusion Matrix of ElectricDevices Dataset

		Predicted Classes						
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Actual Classes	1^{st} Class	84	14	17	57	92	66	3
	2^{nd} Class	0	856	0	0	122	0	0
	3^{rd} Class	1	0	271	15	1	18	71
	4^{th} Class	25	1	98	381	41	25	10
	5^{th} Class	29	91	9	37	726	20	22
	6^{th} Class	33	4	48	60	65	140	21
	7^{th} Class	2	80	3	8	95	8	82

Table A.26: Precision and Recall Values of ElectricDevices Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Precision	0.483	0.818	0.608	0.683	0.636	0.505	0.392
Recall	0.252	0.875	0.719	0.656	0.777	0.377	0.295

Table A.27: Confusion Matrix of Wafer Dataset

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	212	90
	2^{nd} Class	3	2776

Table A.28: Precision and Recall Values of Wafer Dataset

	1^{st} Class	2^{nd} Class
Precision	0.986	0.969
Recall	0.702	0.999

Table A.29: Confusion Matrix of Two Patterns Dataset

		Predicted Classes			
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Actual Classes	1^{st} Class	506	5	6	0
	2^{nd} Class	6	490	2	7
	3^{rd} Class	2	2	481	12
	4^{th} Class	0	4	10	465

Table A.30: Precision and Recall Values of Two Patterns Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Precision	0.984	0.978	0.964	0.961
Recall	0.989	0.970	0.968	0.971

Table A.31: Confusion Matrix of ProximalPhalanxOutlineCorrect

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	36	10
	2^{nd} Class	9	90

Table A.32: Precision and Recall Values of ProximalPhalanxOutlineCorrect

	1^{st} Class	2^{nd} Class
Precision	0.800	0.900
Recall	0.783	0.909

A.5 Confusion Matrices & Precision and Recall Values for DBN Based Classifier

Table A.33: Confusion Matrix of ElectricDevices Dataset

		Predicted Classes						
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Actual Classes	1^{st} Class	71	77	32	128	19	5	1
	2^{nd} Class	0	912	0	1	65	0	0
	3^{rd} Class	2	0	334	14	1	0	26
	4^{th} Class	46	0	97	397	36	4	1
	5^{th} Class	21	131	15	55	689	2	10
	6^{th} Class	22	0	105	156	59	9	20
	7^{th} Class	2	39	14	45	126	2	50

Table A.34: Precision and Recall Values of ElectricDevices Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class	5^{th} Class	6^{th} Class	7^{th} Class
Precision	0.4056	0.787	0.560	0.499	0.693	0.409	0.463
Recall	0.213	0.933	0.886	0.683	0.738	0.024	0.180

Table A.35: Confusion Matrix of Wafer Dataset

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	301	1
	2^{nd} Class	3	2776

Table A.36: Precision and Recall Values of Wafer Dataset

	1^{st} Class	2^{nd} Class
Precision	0.9901	0.9996
Recall	0.9967	0.9989

Table A.37: Confusion Matrix of Two Patterns Dataset

		Predicted Classes			
		1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Actual Classes	1^{st} Class	512	3	2	0
	2^{nd} Class	16	482	1	6
	3^{rd} Class	1	2	484	10
	4^{th} Class	0	3	3	473

Table A.38: Precision and Recall Values of Two Patterns Dataset

	1^{st} Class	2^{nd} Class	3^{rd} Class	4^{th} Class
Precision	0.968	0.984	0.988	0.967
Recall	0.990	0.955	0.974	0.988

Table A.39: Confusion Matrix of ProximalPhalanxOutlineCorrect

		Predicted Classes	
		1^{st} Class	2^{nd} Class
Actual Classes	1^{st} Class	24	22
	2^{nd} Class	0	99

Table A.40: Precision and Recall Values of ProximalPhalanxOutlineCorrect

	1^{st} Class	2^{nd} Class
Precision	1.000	0.818
Recall	0.522	1.000

APPENDIX B

TRAINING SET ERROR AND VALIDATION SET ERROR DIAGRAMS

Green lines in Appendix B diagrams indicate the minimum values that is reached by the validation error curves.

B.1 Training Set Error and Validation Set Error Diagrams for SAE Based Classifier

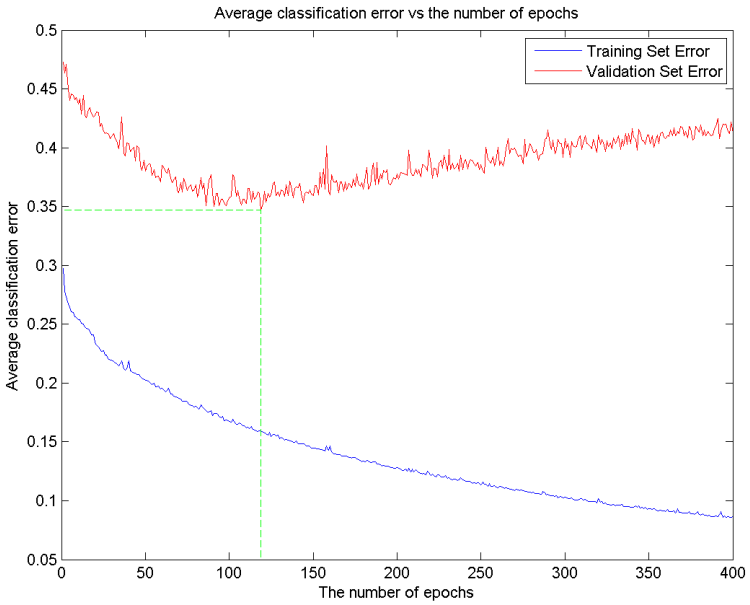


Figure B.1: Training and Validation Set Errors of ElectricalDevices Dataset for 1-Hidden-Layer-SAE architecture

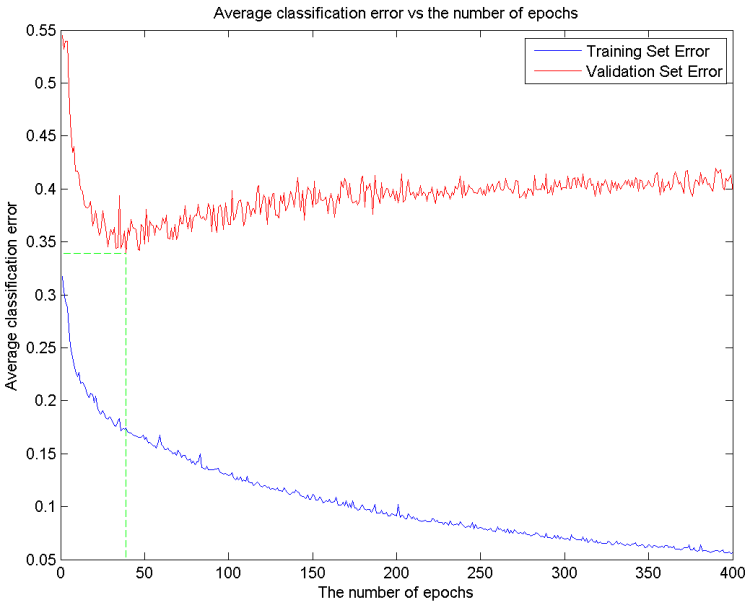


Figure B.2: Training and Validation Set Errors of ElectricalDevices Dataset for 2-Hidden-Layers-SAE architecture

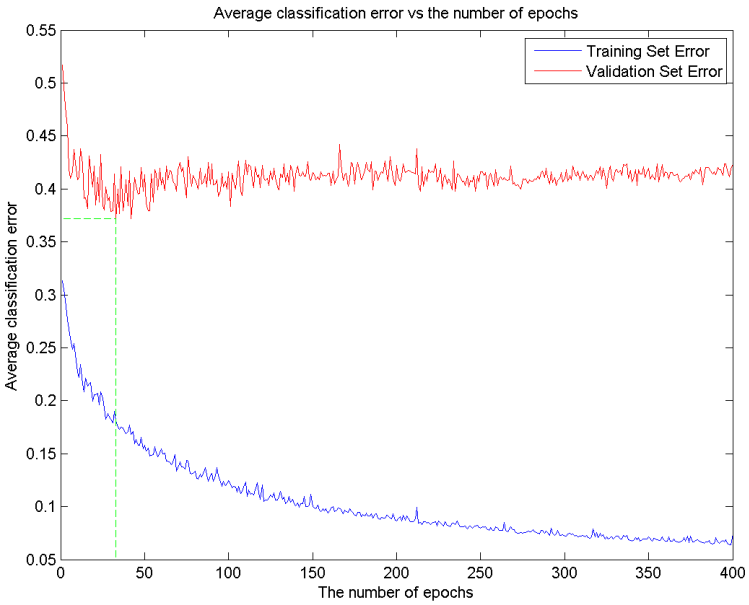


Figure B.3: Training and Validation Set Errors of ElectricalDevices Dataset for 3-Hidden-Layers-SAE architecture

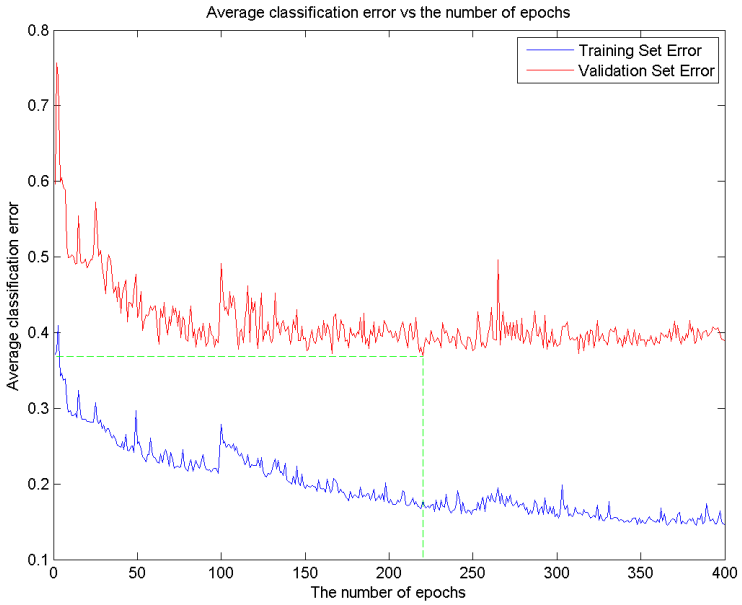


Figure B.4: Training and Validation Set Errors of ElectricalDevices Dataset for 4-Hidden-Layers-SAE architecture

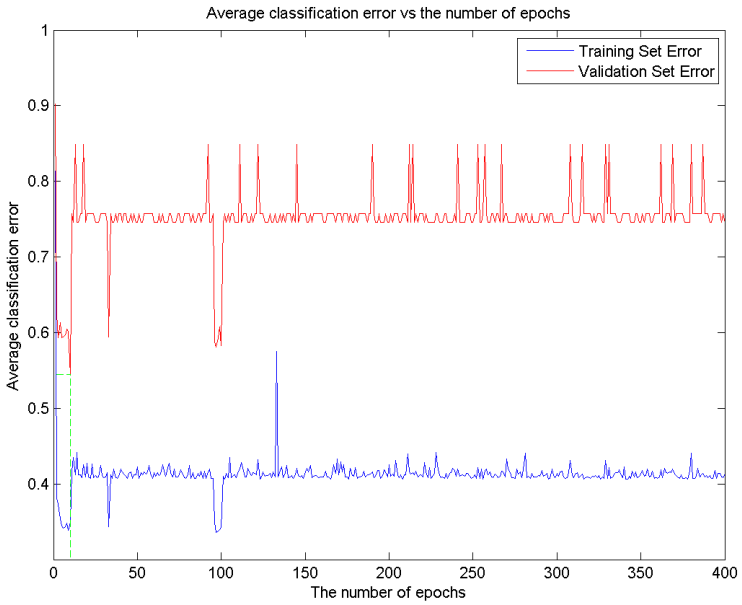


Figure B.5: Training and Validation Set Errors of ElectricalDevices Dataset for 5-Hidden-Layers-SAE architecture

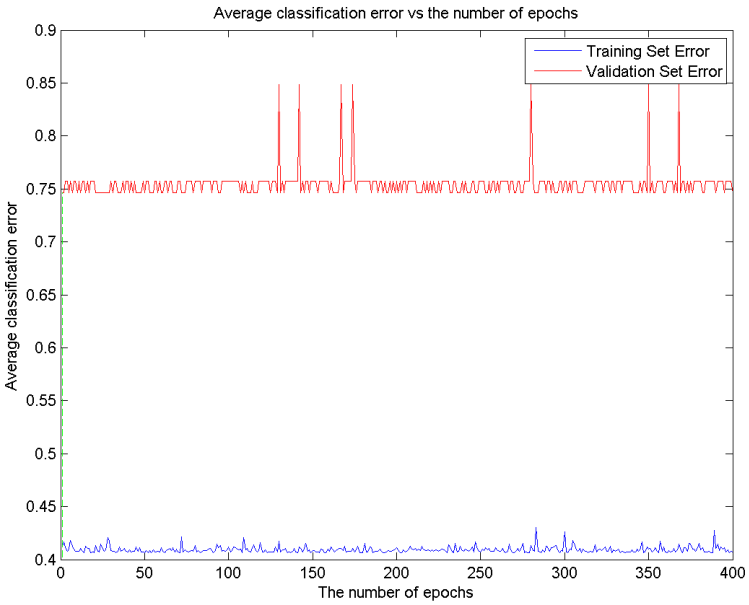


Figure B.6: Training and Validation Set Errors of ElectricalDevices Dataset for 6-Hidden-Layers-SAE architecture

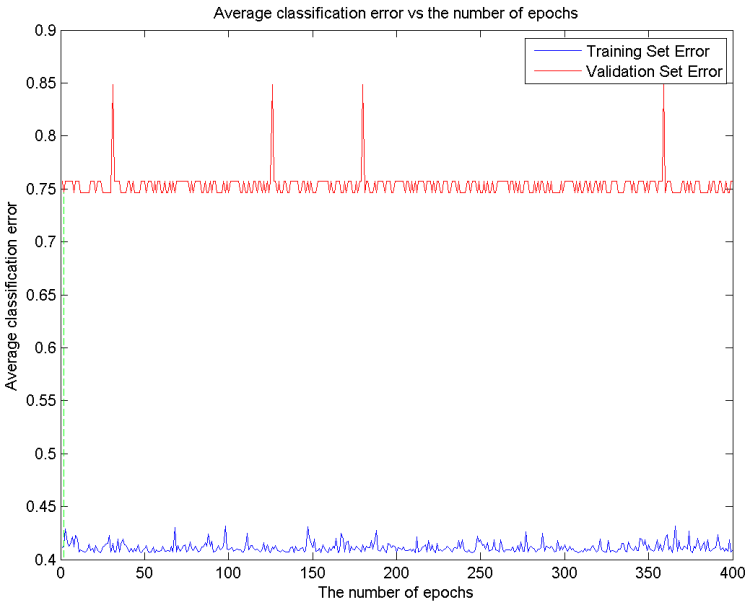


Figure B.7: Training and Validation Set Errors of ElectricalDevices Dataset for 7-Hidden-Layers-SAE architecture

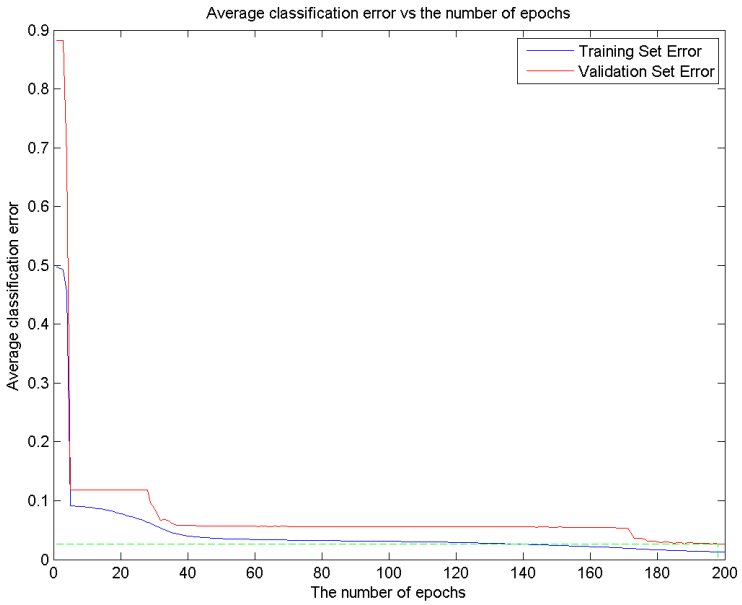


Figure B.8: Training and Validation Set Errors of Wafer Dataset for 1-Hidden-Layer-SAE architecture

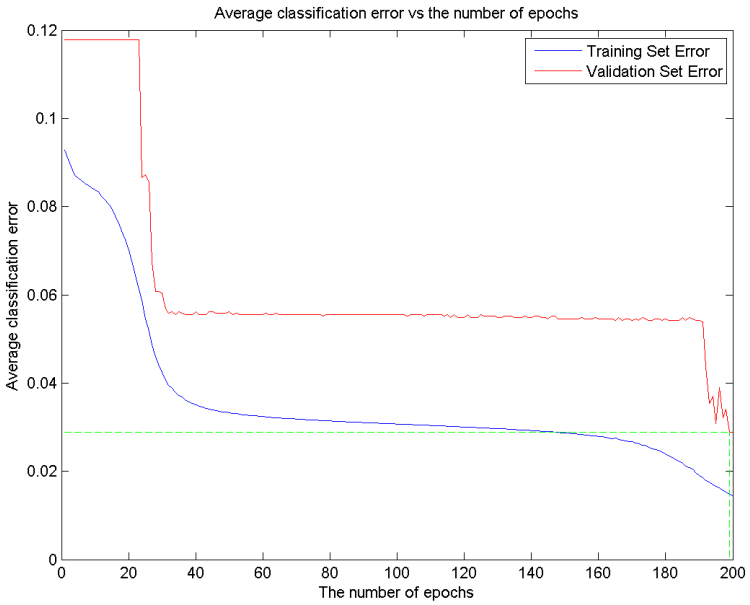


Figure B.9: Training and Validation Set Errors of Wafer Dataset for 2-Hidden-Layers-SAE architecture

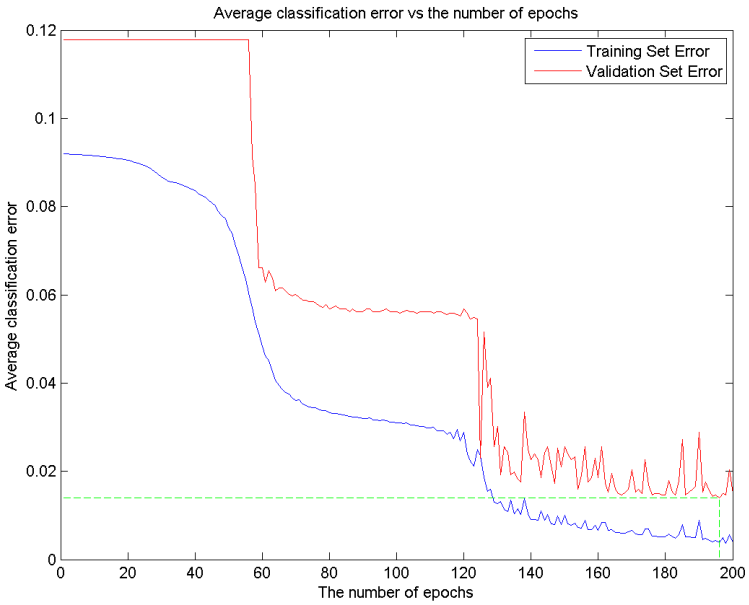


Figure B.10: Training and Validation Set Errors of Wafer Dataset for 3-Hidden-Layers-SAE architecture

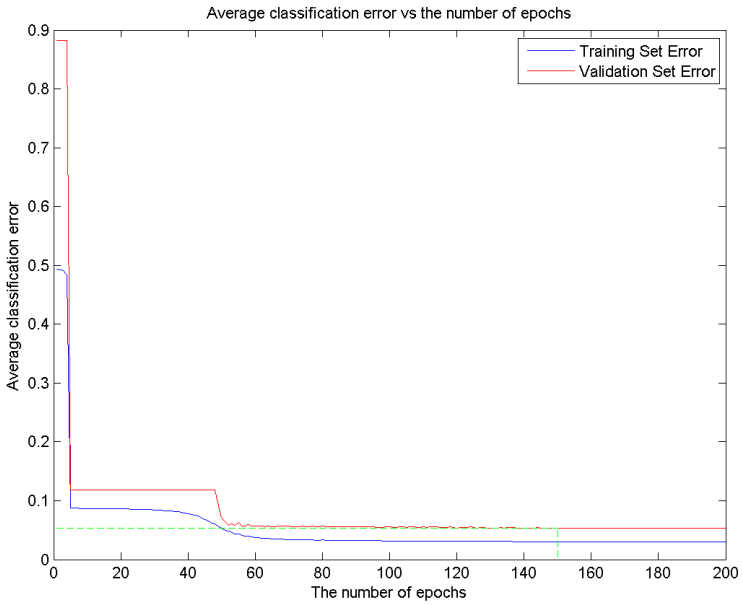


Figure B.11: Training and Validation Set Errors of Wafer Dataset for 4-Hidden-Layers-SAE architecture

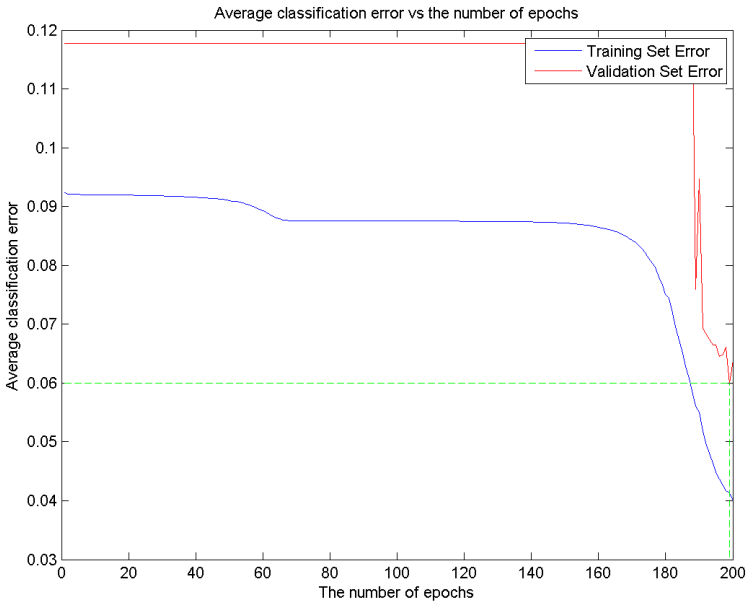


Figure B.12: Training and Validation Set Errors of Wafer Dataset for 5-Hidden-Layers-SAE architecture

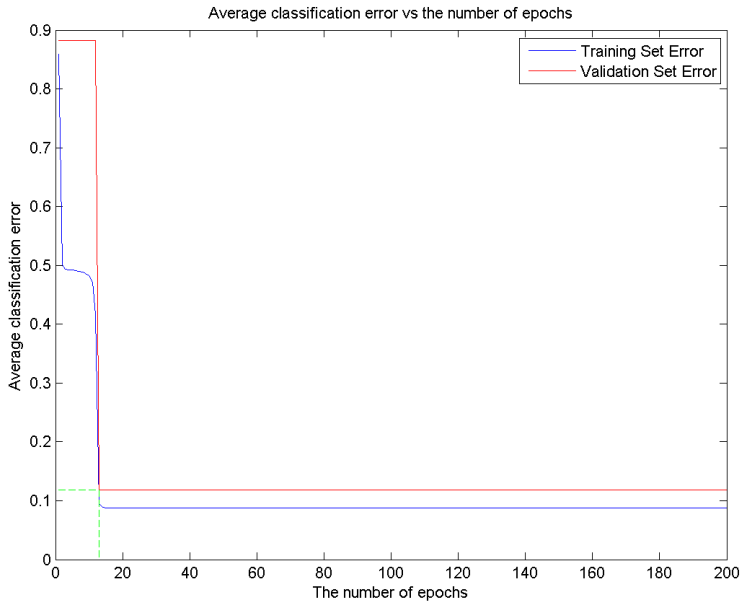


Figure B.13: Training and Validation Set Errors of Wafer Dataset for 6-Hidden-Layers-SAE architecture

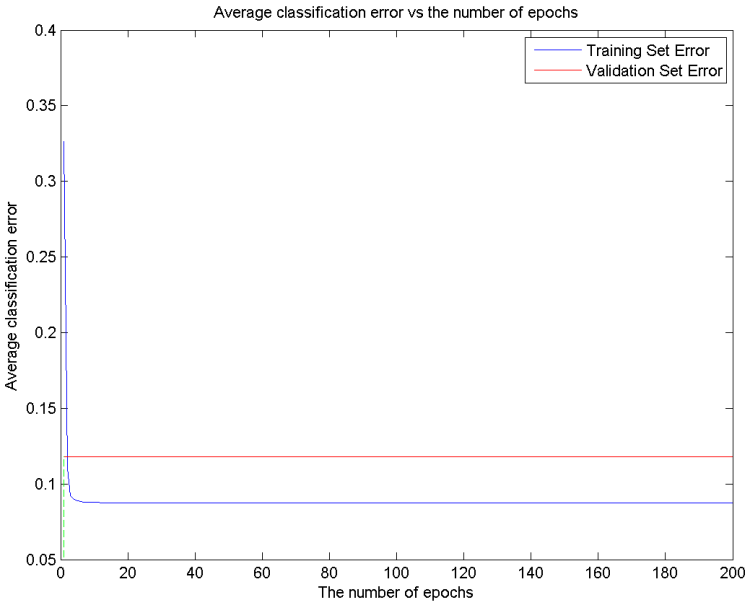


Figure B.14: Training and Validation Set Errors of Wafer Dataset for 7-Hidden-Layers-SAE architecture

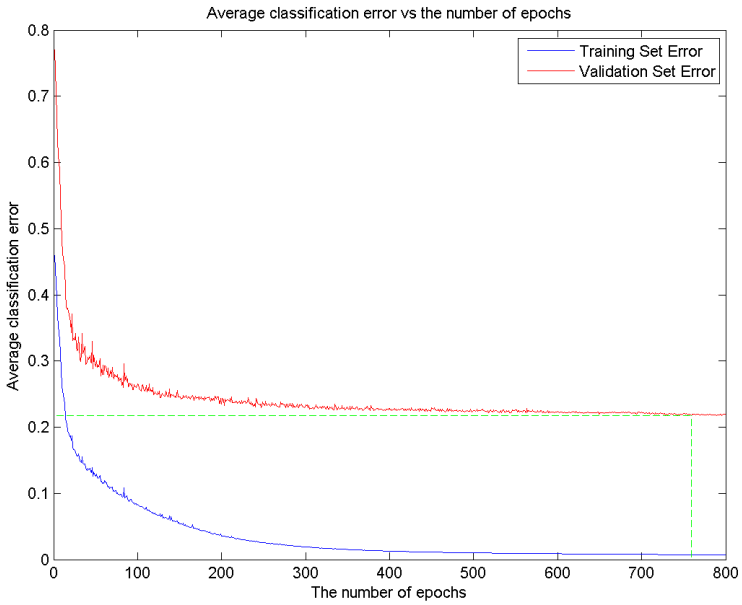


Figure B.15: Training and Validation Set Errors of Two Patterns Dataset for 1-Hidden-Layer-SAE architecture

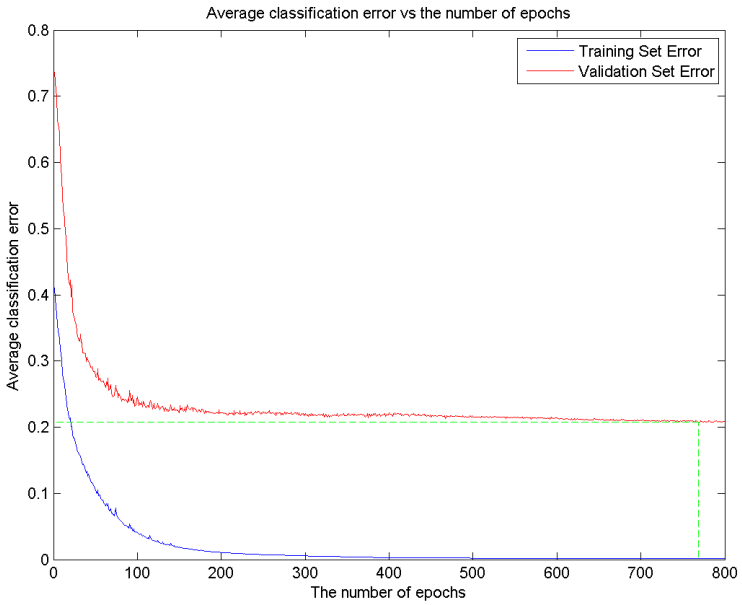


Figure B.16: Training and Validation Set Errors of Two Patterns Dataset for 2-Hidden-Layers-SAE architecture

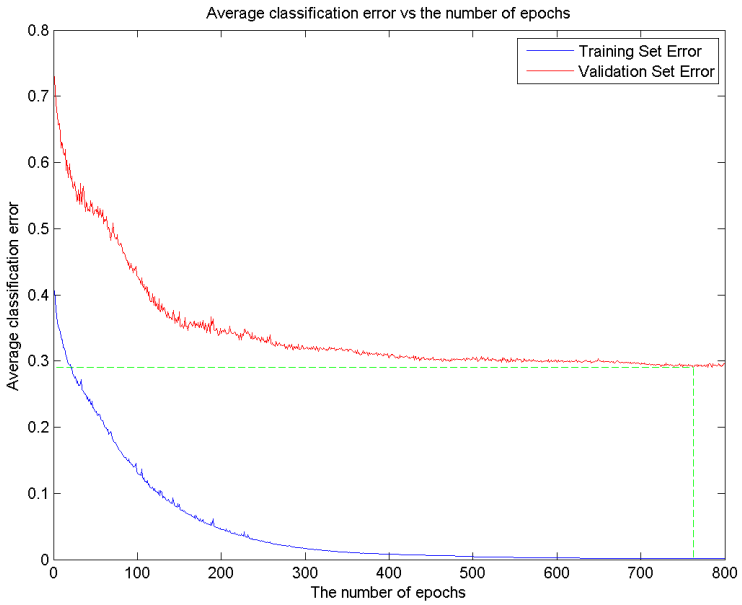


Figure B.17: Training and Validation Set Errors of Two Patterns Dataset for 3-Hidden-Layers-SAE architecture

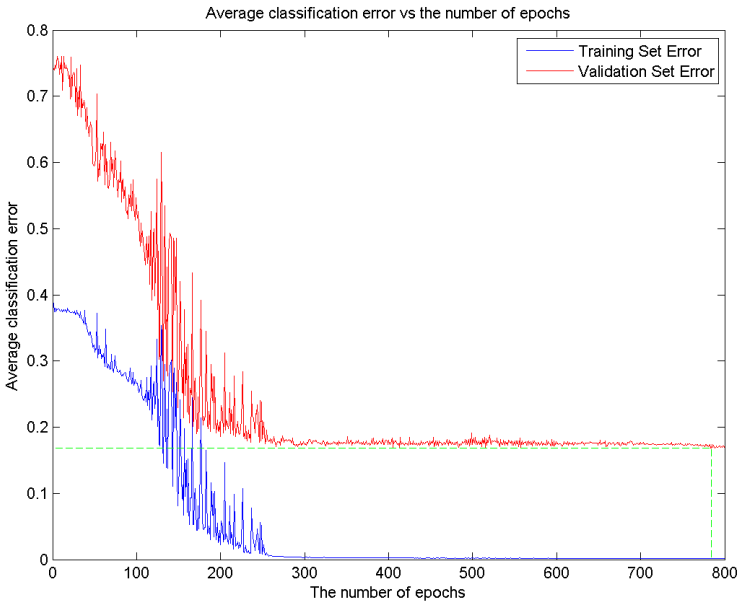


Figure B.18: Training and Validation Set Errors of Two Patterns Dataset for 4-Hidden-Layers-SAE architecture

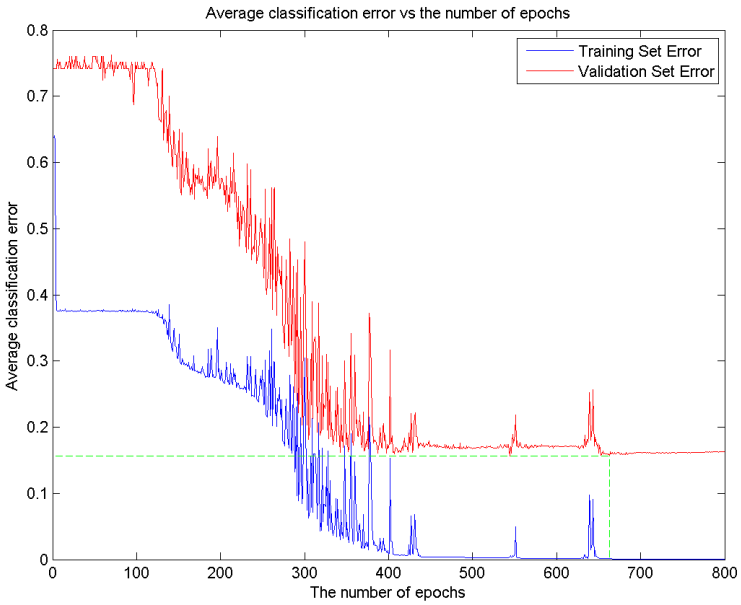


Figure B.19: Training and Validation Set Errors of Two Patterns Dataset for 5-Hidden-Layers-SAE architecture

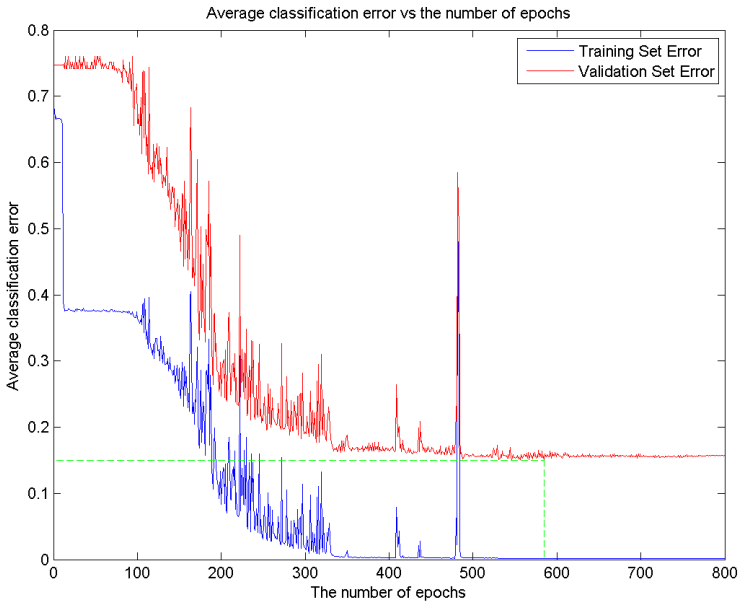


Figure B.20: Training and Validation Set Errors of Two Patterns Dataset for 6-Hidden-Layers-SAE architecture

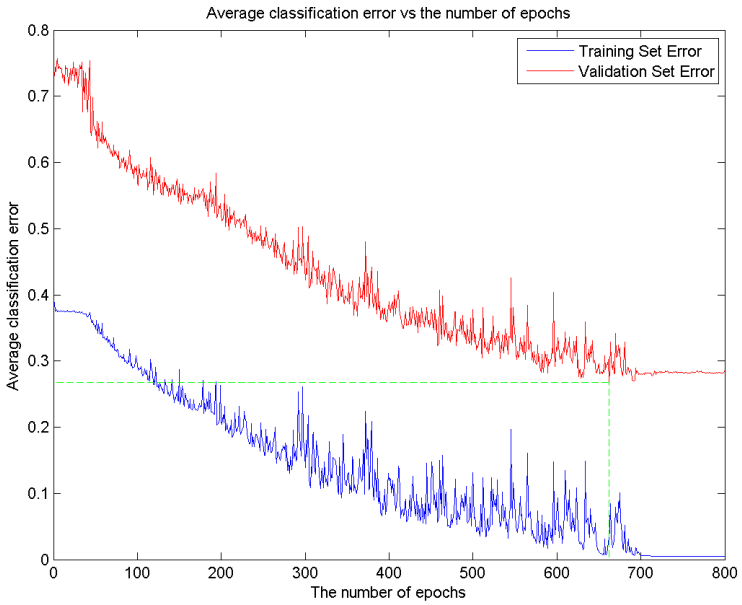


Figure B.21: Training and Validation Set Errors of Two Patterns Dataset for 7-Hidden-Layers-SAE architecture

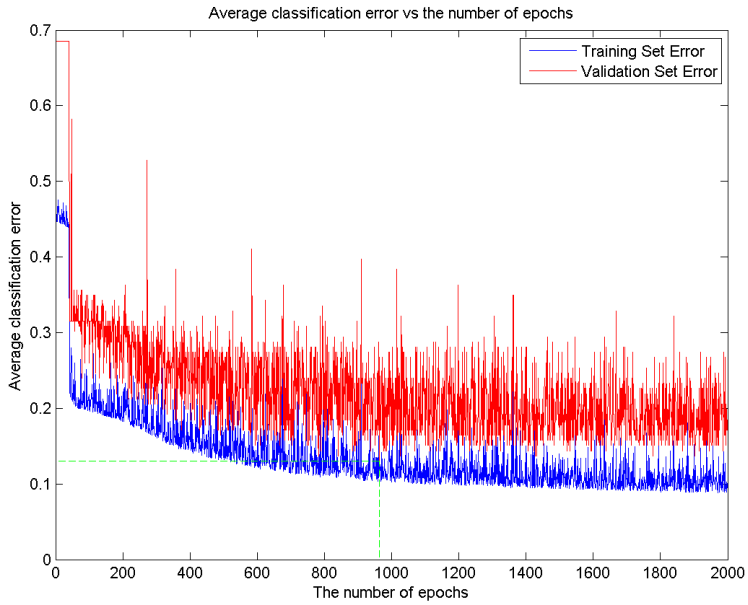


Figure B.22: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 1-Hidden-Layer-SAE architecture

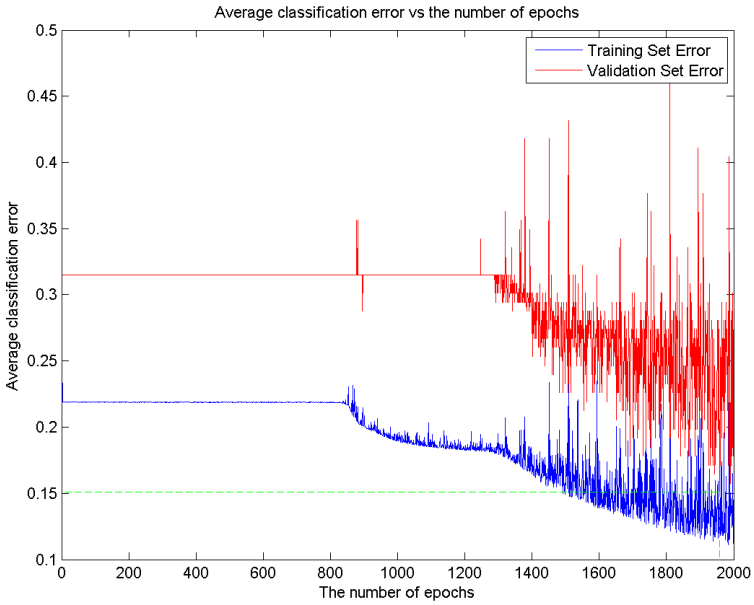


Figure B.23: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 2-Hidden-Layers-SAE architecture

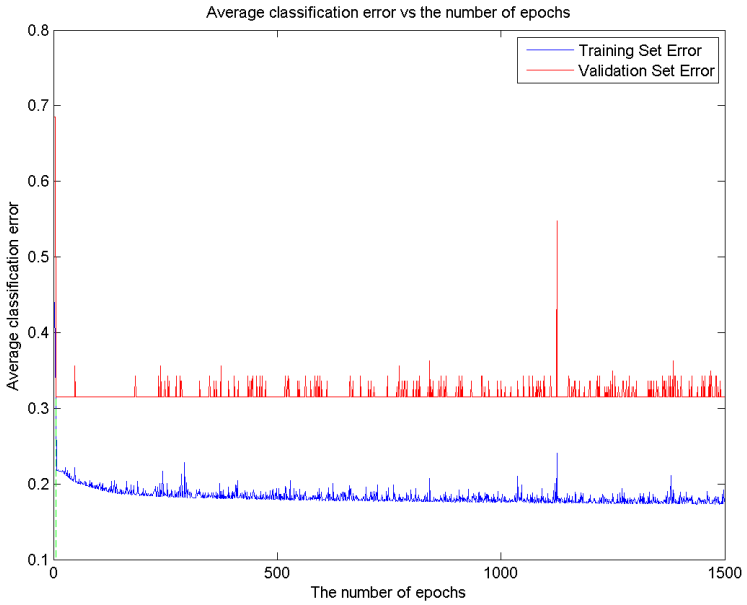


Figure B.24: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 3-Hidden-Layers-SAE architecture

B.2 Training Set Error and Validation Set Error Diagrams for DBN Based Classifier

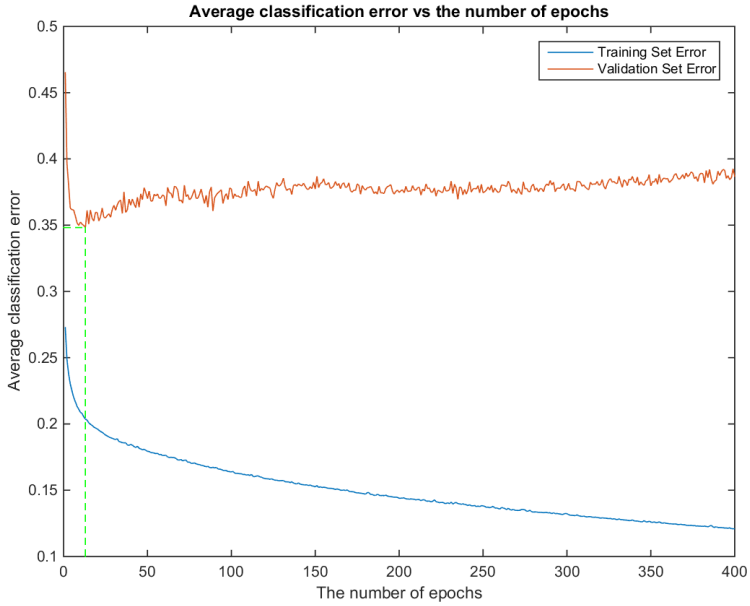


Figure B.25: Training and Validation Set Errors of ElectricalDevices Dataset for 1-Hidden-Layer-DBN architecture

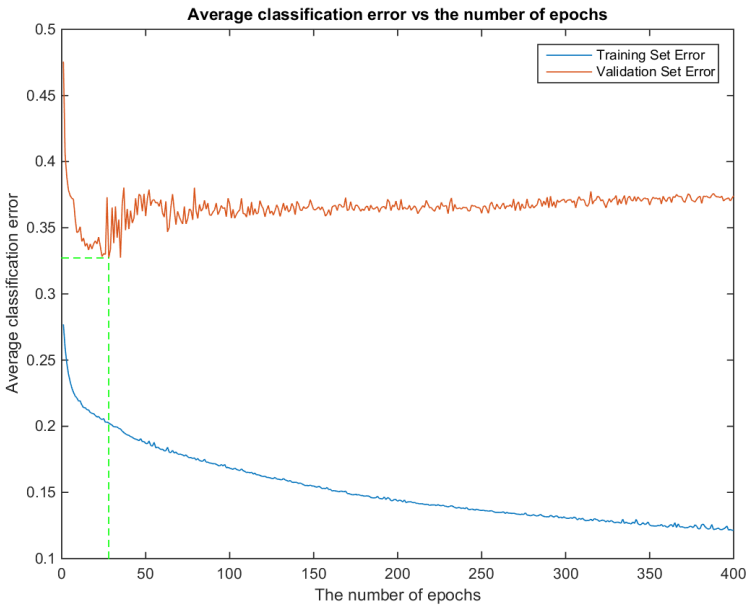


Figure B.26: Training and Validation Set Errors of ElectricalDevices Dataset for 2-Hidden-Layers-DBN architecture

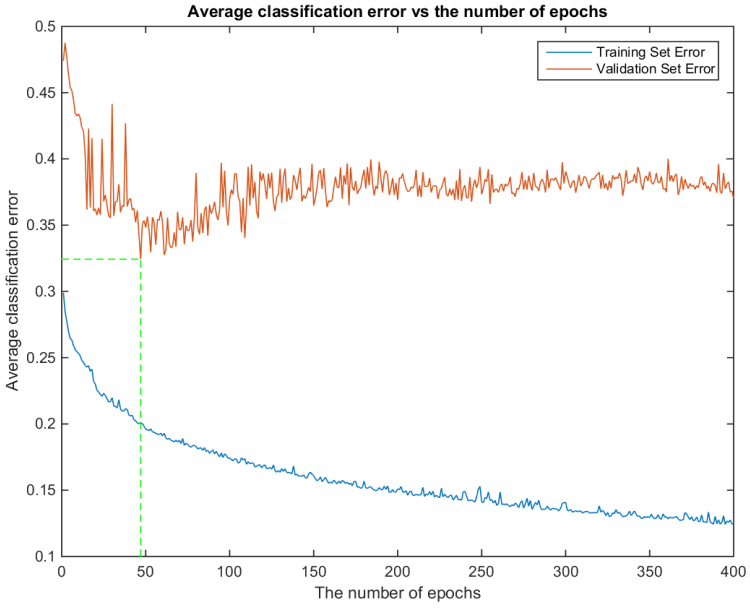


Figure B.27: Training and Validation Set Errors of ElectricalDevices Dataset for 3-Hidden-Layers-DBN architecture

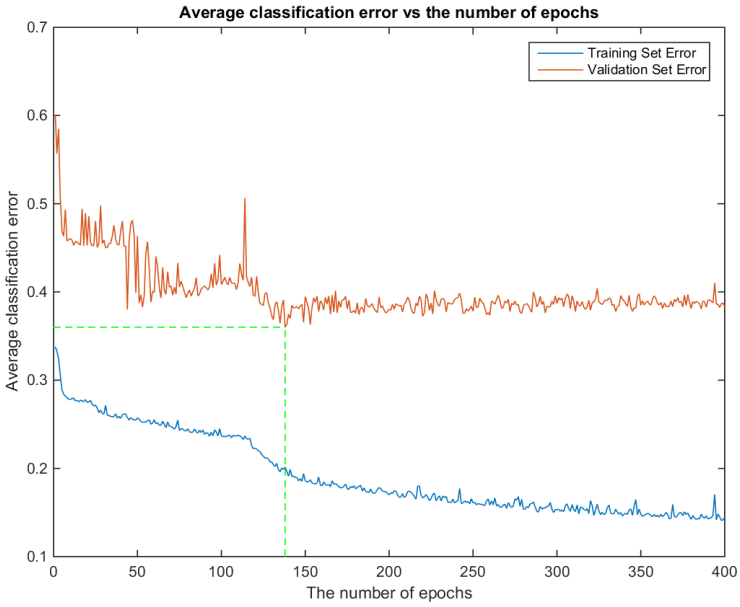


Figure B.28: Training and Validation Set Errors of ElectricalDevices Dataset for 4-Hidden-Layers-DBN architecture

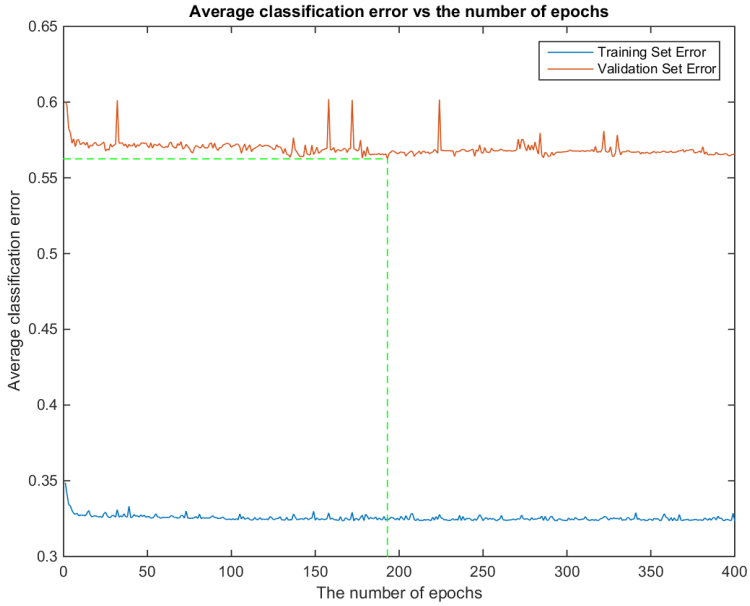


Figure B.29: Training and Validation Set Errors of ElectricalDevices Dataset for 5-Hidden-Layers-DBN architecture

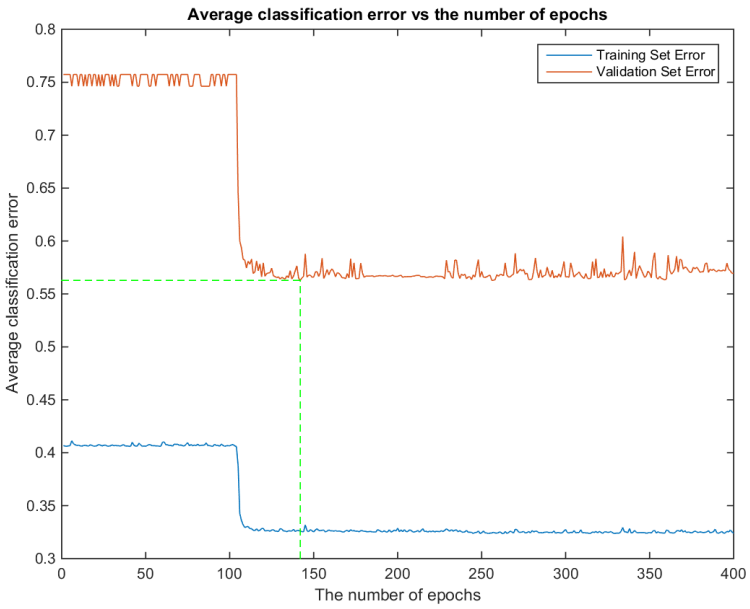


Figure B.30: Training and Validation Set Errors of ElectricalDevices Dataset for 6-Hidden-Layers-DBN architecture

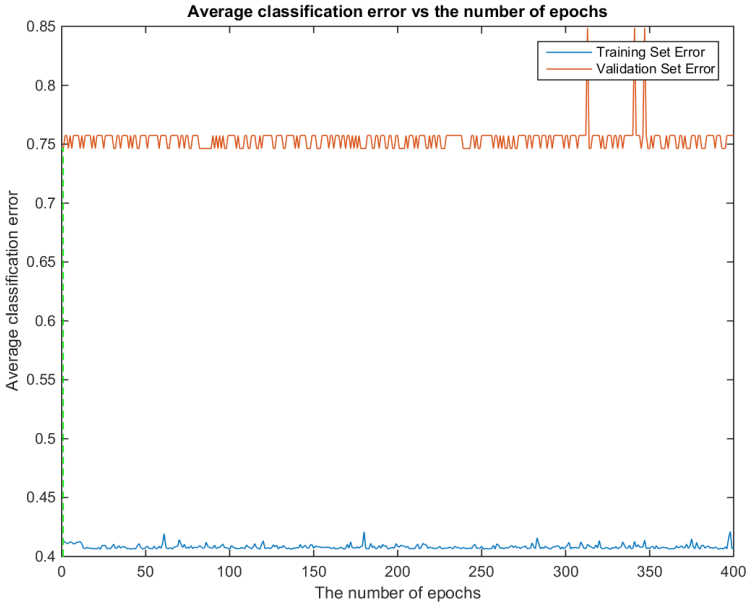


Figure B.31: Training and Validation Set Errors of ElectricalDevices Dataset for 7-Hidden-Layers-DBN architecture

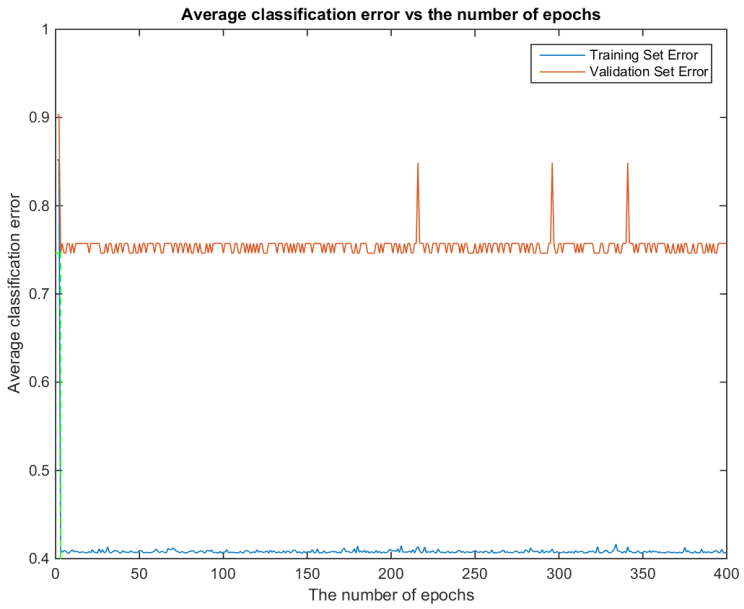


Figure B.32: Training and Validation Set Errors of ElectricalDevices Dataset for 8-Hidden-Layers-DBN architecture

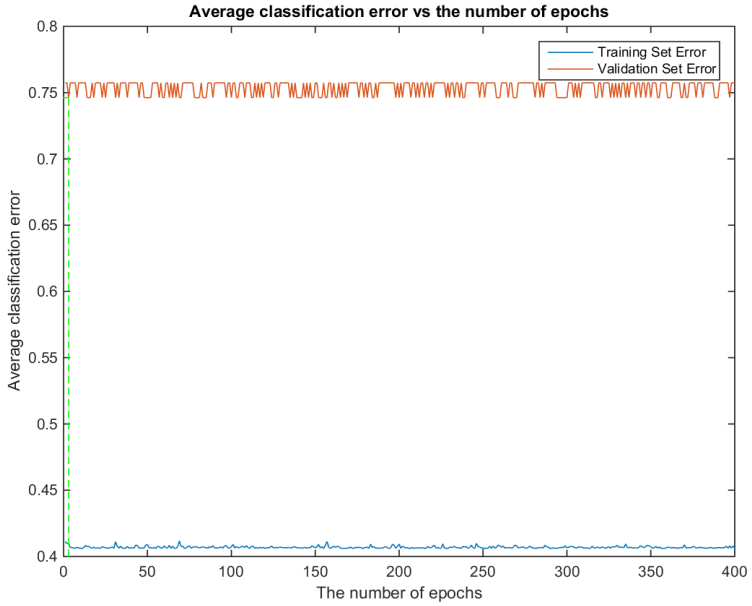


Figure B.33: Training and Validation Set Errors of ElectricalDevices Dataset for 9-Hidden-Layers-DBN architecture

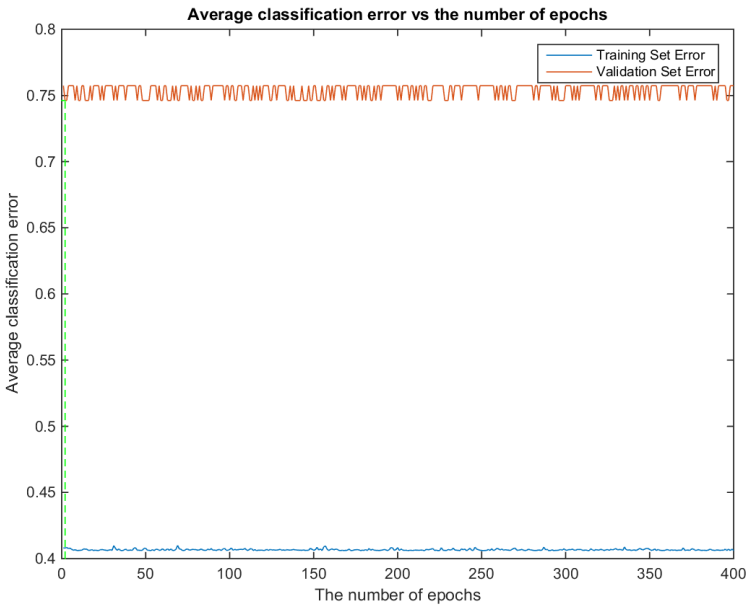


Figure B.34: Training and Validation Set Errors of ElectricalDevices Dataset for 10-Hidden-Layers-DBN architecture

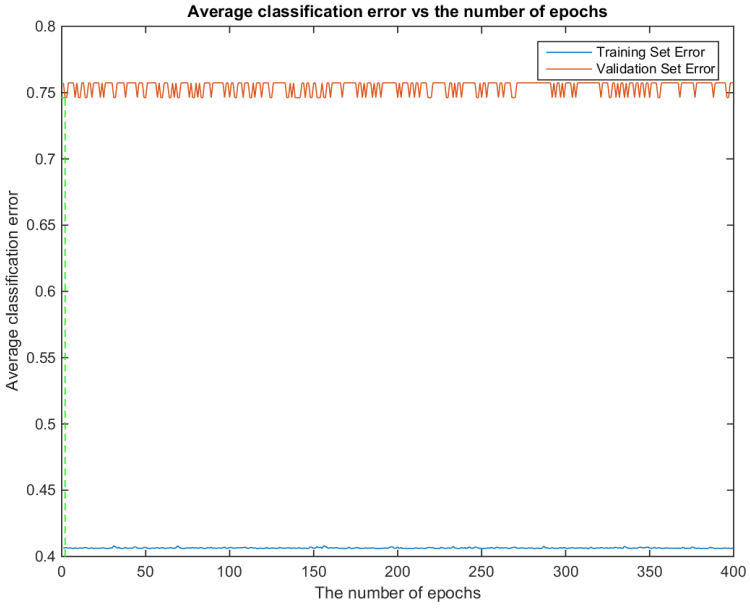


Figure B.35: Training and Validation Set Errors of ElectricalDevices Dataset for 11-Hidden-Layers-DBN architecture

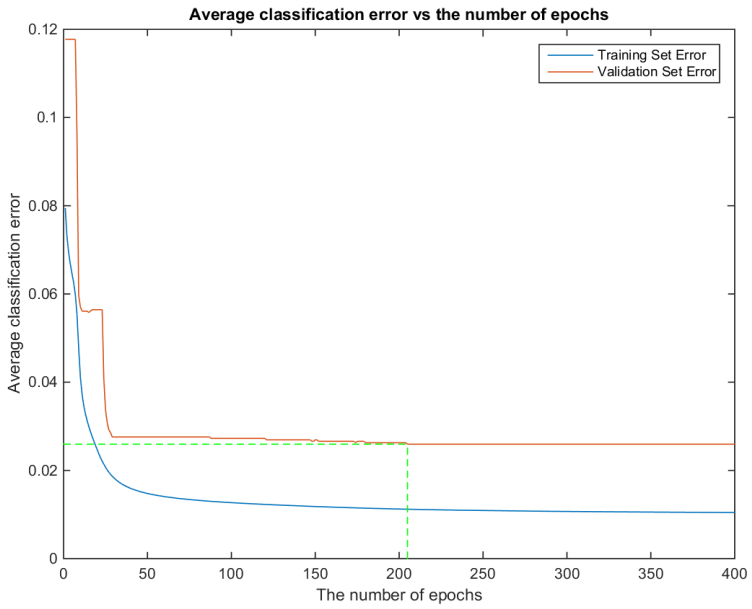


Figure B.36: Training and Validation Set Errors of Wafer Dataset for 1-Hidden-Layer-DBN architecture

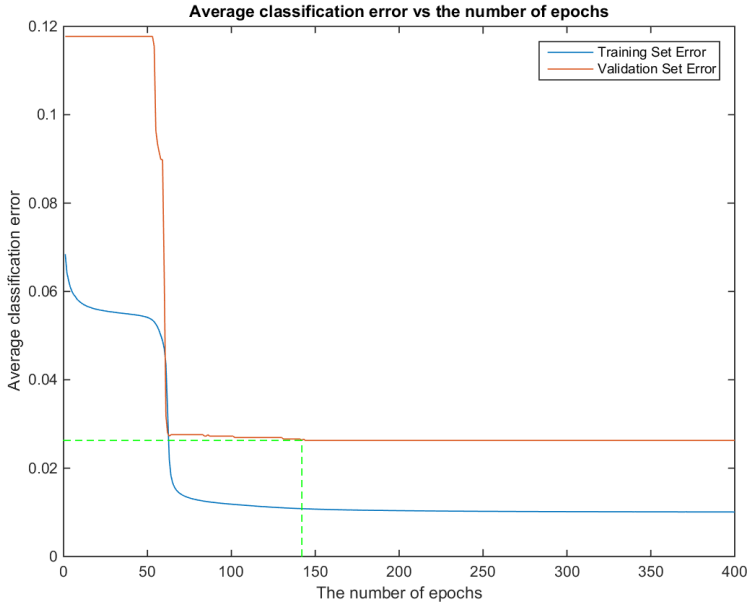


Figure B.37: Training and Validation Set Errors of Wafer Dataset for 2-Hidden-Layers-DBN architecture

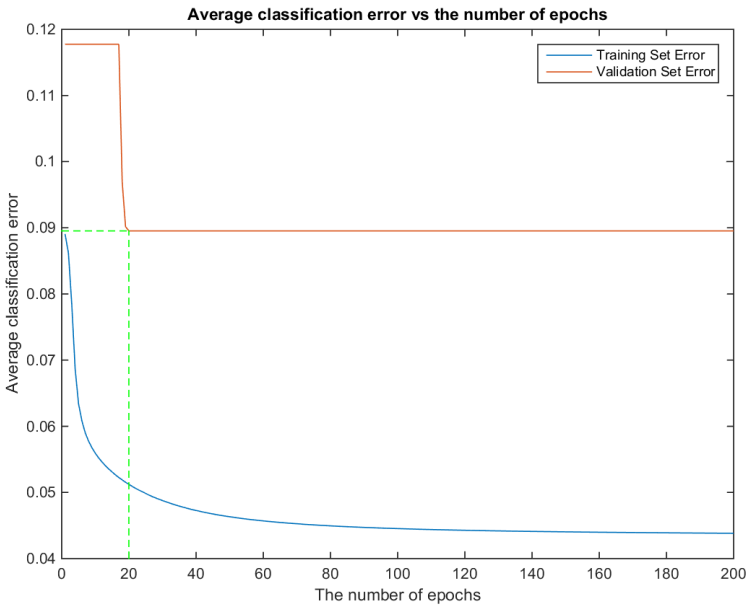


Figure B.38: Training and Validation Set Errors of Wafer Dataset for 3-Hidden-Layers-DBN architecture

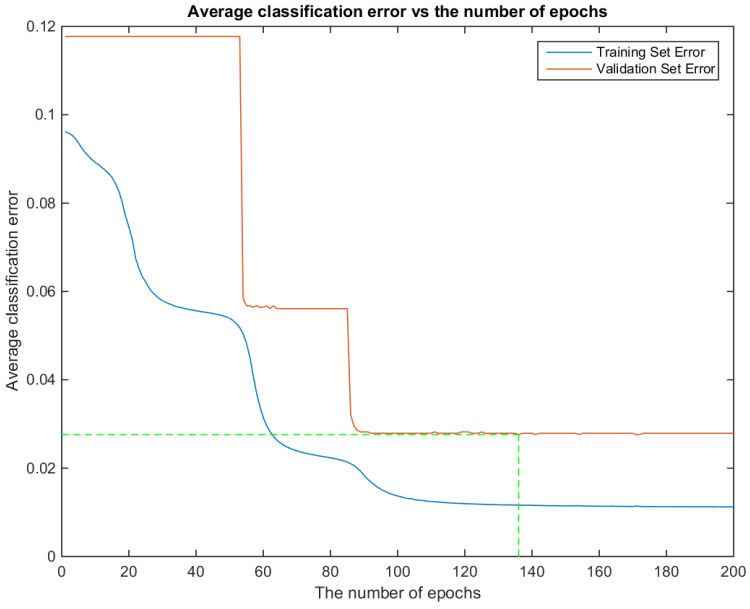


Figure B.39: Training and Validation Set Errors of Wafer Dataset for 4-Hidden-Layers-DBN architecture

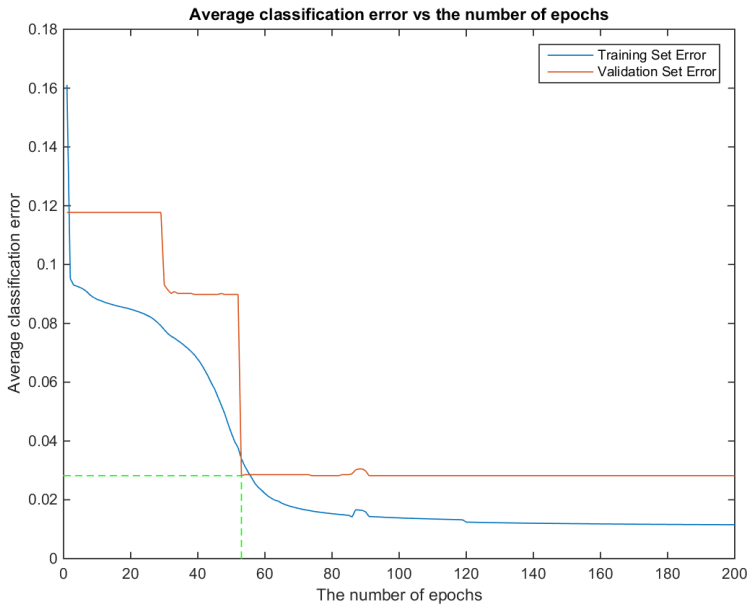


Figure B.40: Training and Validation Set Errors of Wafer Dataset for 5-Hidden-Layers-DBN architecture

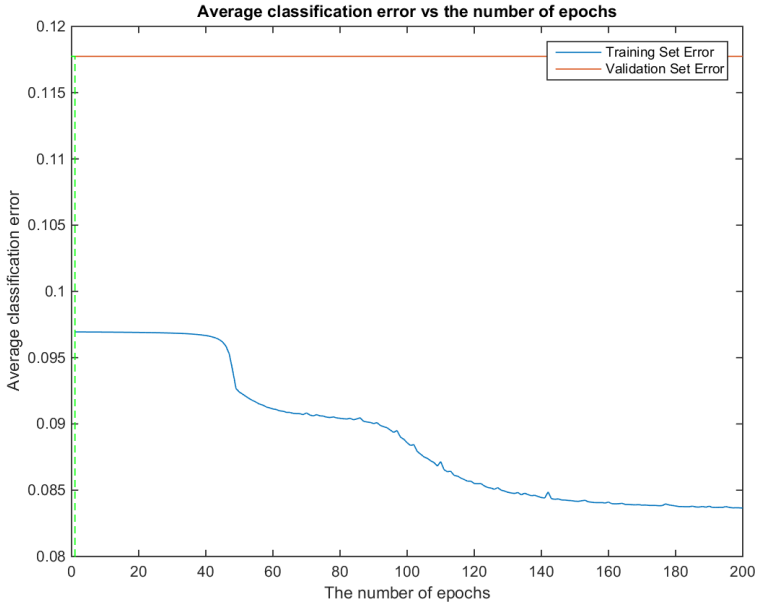


Figure B.41: Training and Validation Set Errors of Wafer Dataset for 6-Hidden-Layers-DBN architecture

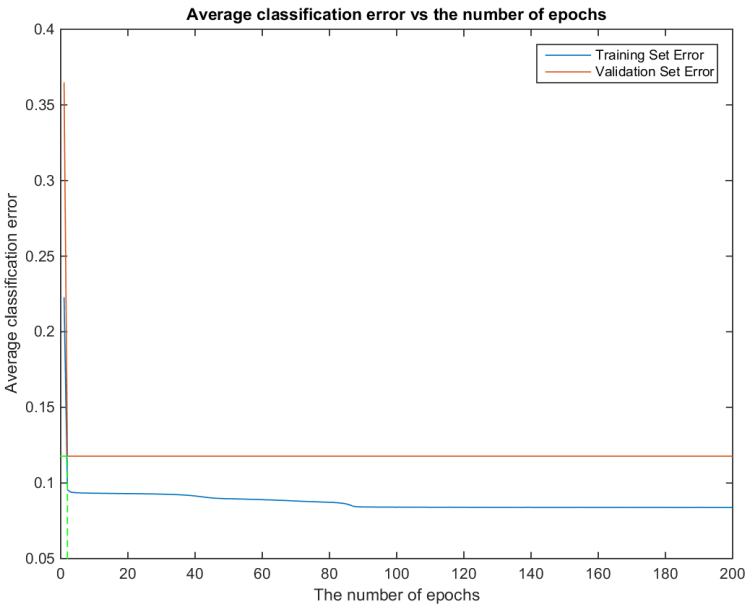


Figure B.42: Training and Validation Set Errors of Wafer Dataset for 7-Hidden-Layers-DBN architecture

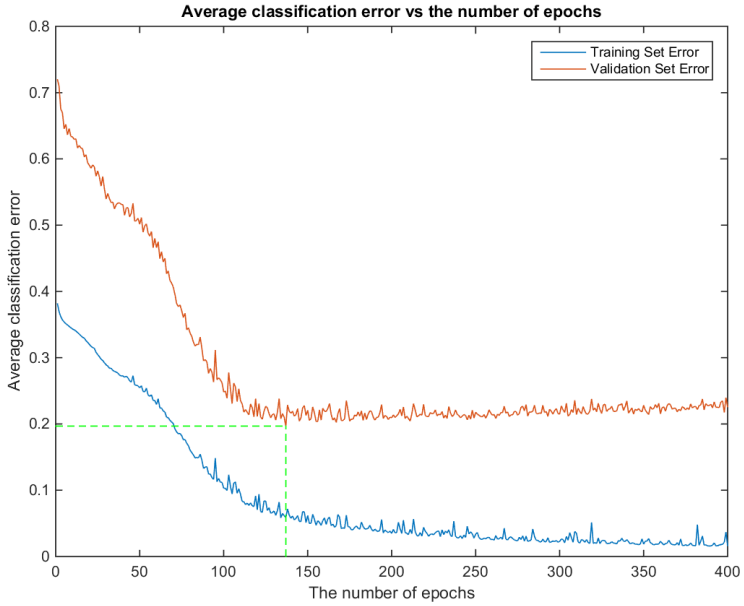


Figure B.43: Training and Validation Set Errors of Two Patterns Dataset for 1-Hidden-Layer-DBN architecture

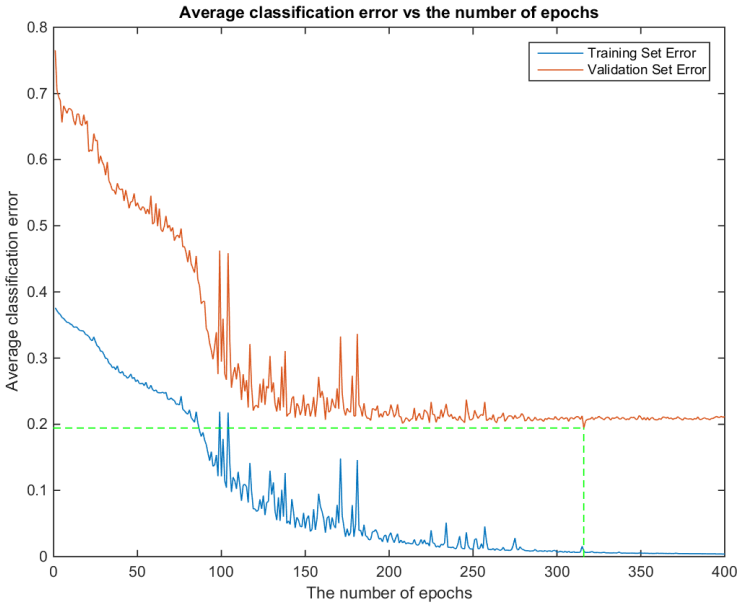


Figure B.44: Training and Validation Set Errors of Two Patterns Dataset for 2-Hidden-Layers-DBN architecture

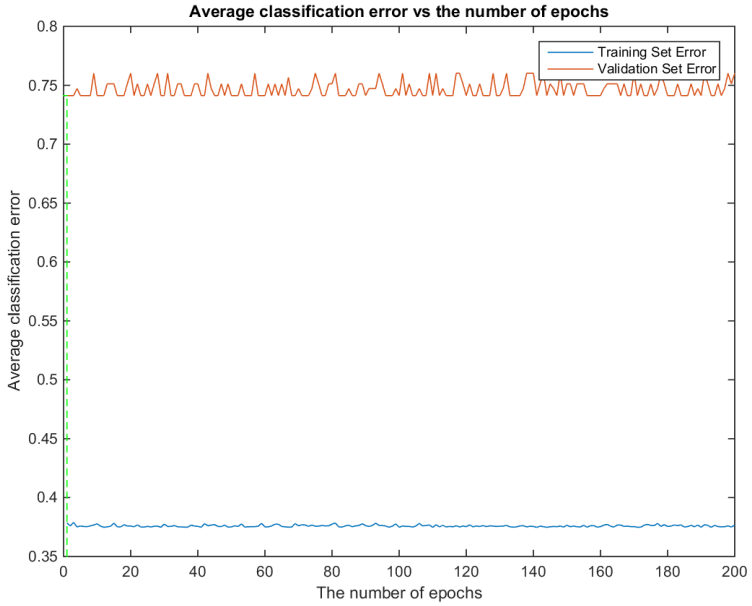


Figure B.45: Training and Validation Set Errors of Two Patterns Dataset for 3-Hidden-Layers-DBN architecture

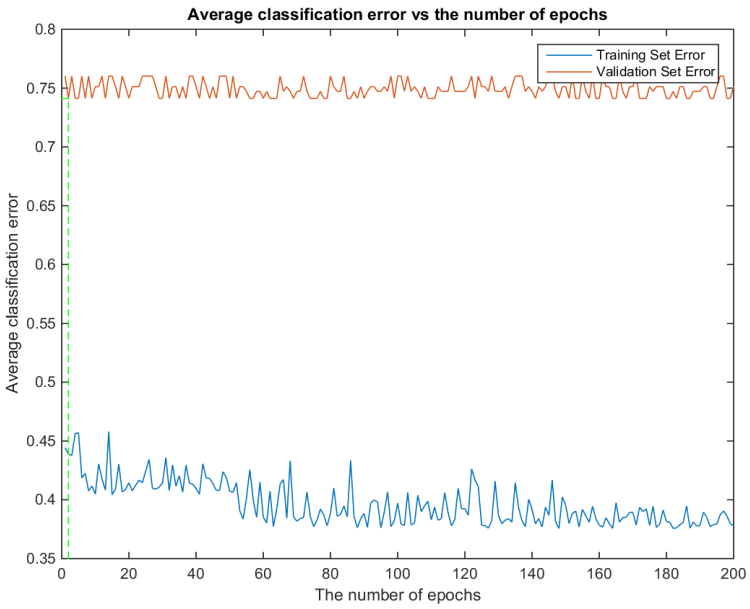


Figure B.46: Training and Validation Set Errors of Two Patterns Dataset for 4-Hidden-Layers-DBN architecture

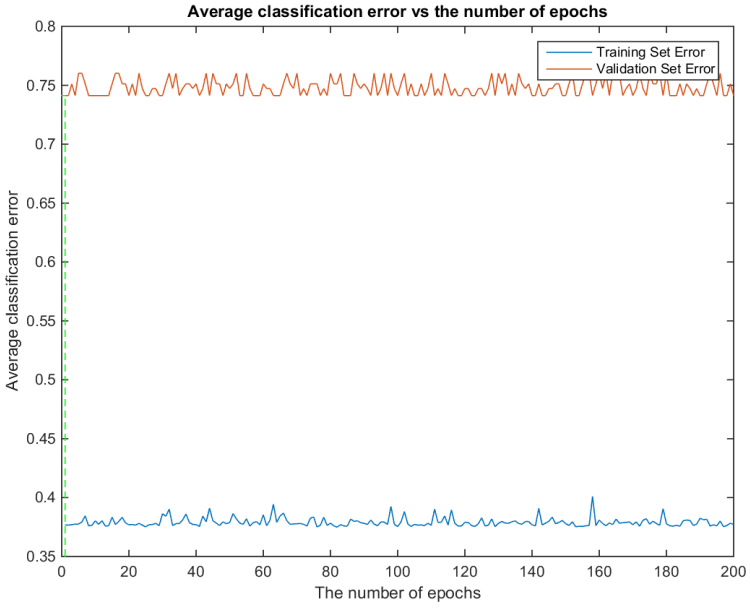


Figure B.47: Training and Validation Set Errors of Two Patterns Dataset for 5-Hidden-Layers-DBN architecture

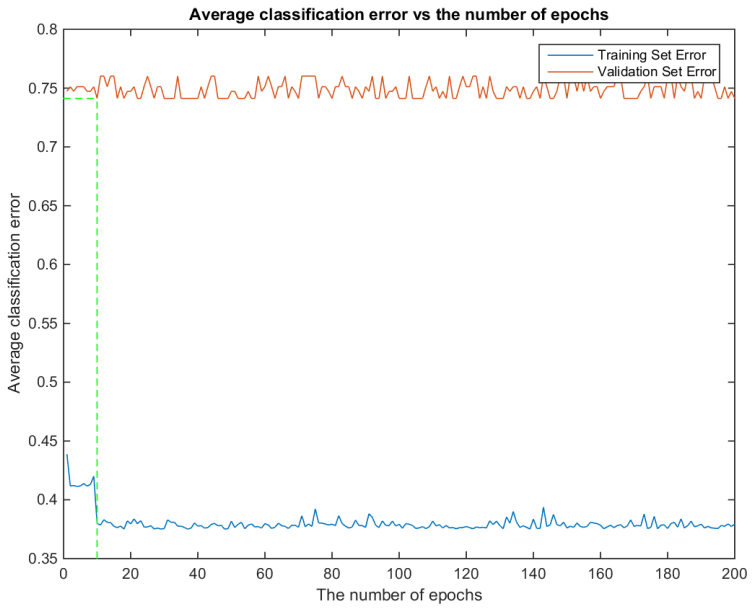


Figure B.48: Training and Validation Set Errors of Two Patterns Dataset for 6-Hidden-Layers-DBN architecture

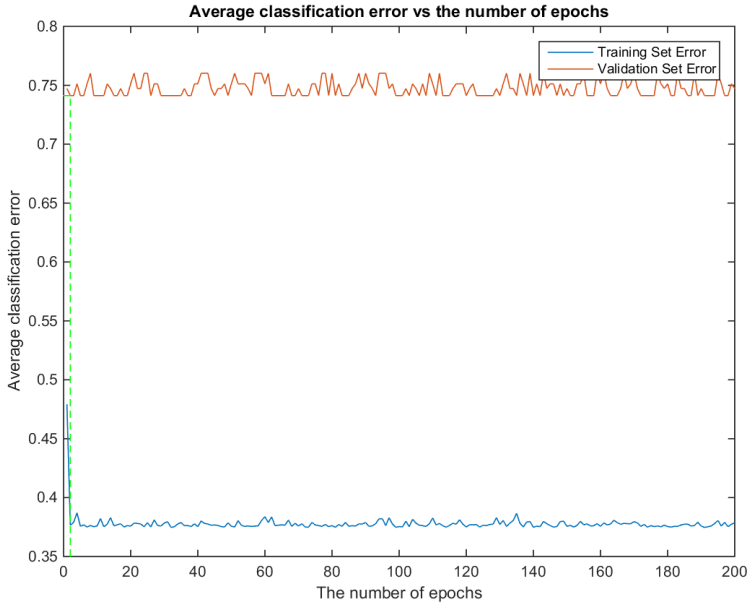


Figure B.49: Training and Validation Set Errors of Two Patterns Dataset for 7-Hidden-Layers-DBN architecture

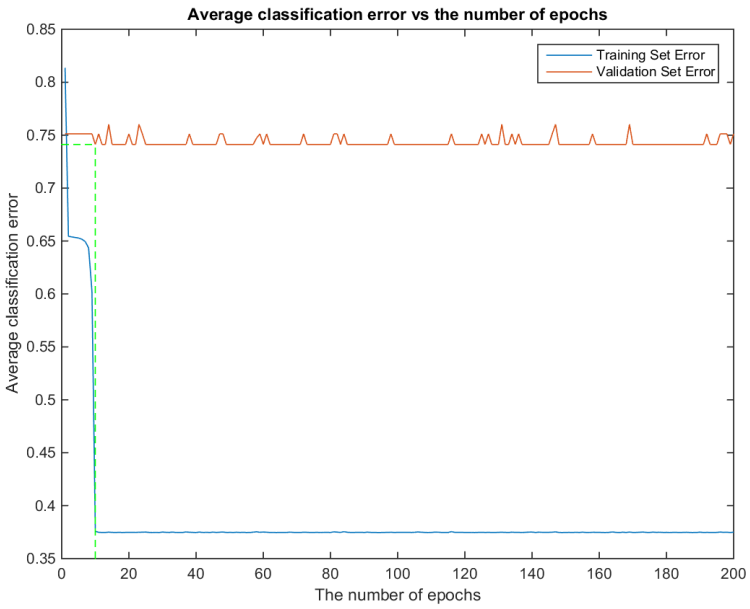


Figure B.50: Training and Validation Set Errors of Two Patterns Dataset for 8-Hidden-Layers-DBN architecture

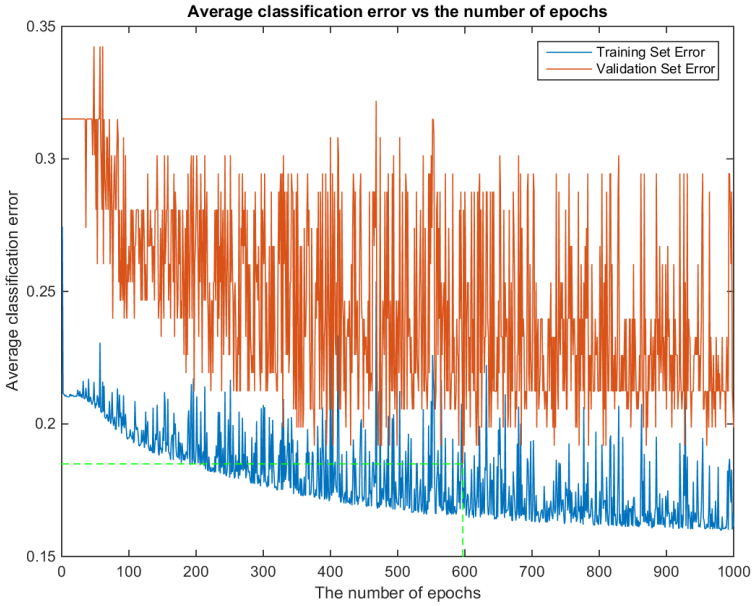


Figure B.51: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 1-Hidden-Layer-DBN architecture

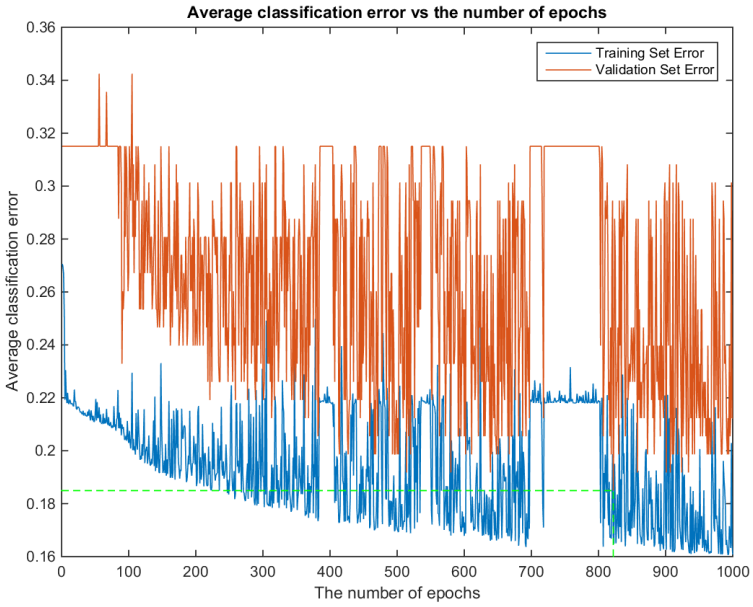


Figure B.52: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 2-Hidden-Layers-DBN architecture

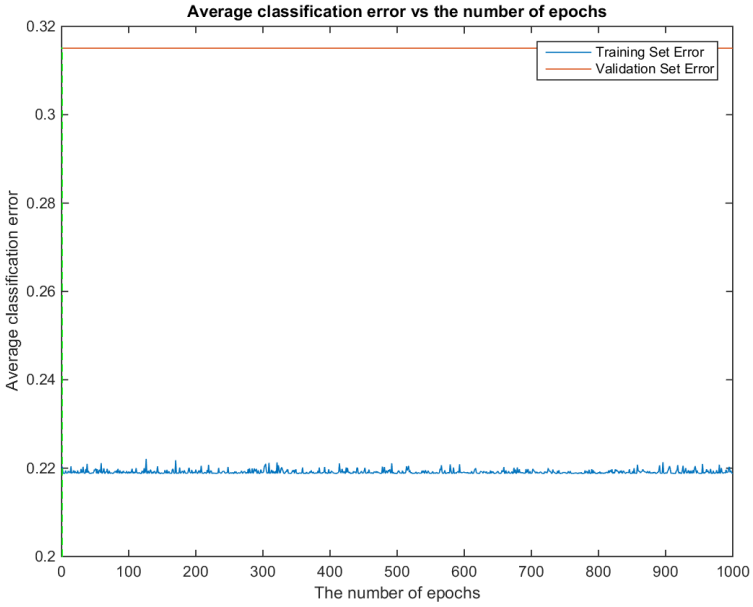


Figure B.53: Training and Validation Set Errors of ProximalPhalanxOutlineCorrect Dataset for 3-Hidden-Layers-DBN architecture