Undergraduate Research Project
Report

A Study of Neural Network Architectures and their Applications

Submitted in partial fulfillment of
the requirements for

Summer Internship
on
Artificial Neural Networks

By

G V S Akash Bharadwaj

15311A03C1

Under the guidance of

Dr. C P Vyasarayani

Department of Mechanical and Aerospace Engineering
Indian Institute of Technology Hyderabad
Kandi, Sangareddy, Telangana, India, 502 285

May - July 2018

Department of Mechanical and Aerospace Engineering

INDIAN INSTITUTE OF TECHNOLOGY HYDERABAD

Certificate

This is to certify that this is a bonafide record of the project completed by G V S Akash Bharadwaj (15311A03C1), during the months of May - July 2018 in partial fulfillment of the requirements of the summer internship on the topic Artificial Neural Networks.

Dr. C P Vyasarayani
(Associate Professor)

Date: 12/07/2018

Contents

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my guide Dr. Chandrika Prakash Vyasarayani, for providing an opportunity to work under him and for his excellent guidance. His kindness, friendly accessibility and attention to details has been a great inspiration. My heartiest thanks for the support, motivation and the patience he has shown during this project. His suggestions and advice have been always constructive. I hope that I will be able to pass on the research values and knowledge that he has imparted to me.

I am very much thankful to Dr. Sobhan Babu for his continued support and guidance. I am grateful for his instrumental suggestions for continuous improvement and for further research. I would also like to thank the PhD scholars and Research fellows under Dr. Vyasarayani, for their guidance and inputs during various stages of the project. Their friendly behavior and kind cooperation made the environment productive.

Finally, I would also like to thank Dr. G S Reddy, the Project Coordinator from Sreenidhi Institute of Science and Technology, for enabling me to take up this internship and for his support throughout the duration of this program.

Chapter 1

Abstract

In the process of the development of intelligent systems, the artificial neural network plays an important role as a paradigm especially for pattern recognition, pattern association, optimization, prediction, and decision making problems. With a general definition of artificial neural networks, a multi neuron perceptron model for obtaining weight and bias by convergence to the perceptron learning rule has been developed. This was done after studying several activation functions and discussion of network topologies.

The second section introduces supervised network training, briefly about concepts of Linear Associator, Hebb rule and Pseudo-inverse rule. An Auto-associative Network for Pattern extraction and Digit Recognition was designed, and further trained for recovery of occluded and noisy patterns from the image datasets.

In addition, the concepts of Generalization and Hidden-layer Neural Nets are elucidated in the third section. Initially tested with a XOR layout network, for function approximation and then a network model was created and trained for convergence & steepest descent; studying changes for each of network parameters. ADALINE (hidden-layer) neural networks were implemented by leveraging backpropagation techniques, to facilitate in creating a Time Series Analysis model for the project submission.

Chapter 2

Introduction

2.1 Motivation

All the architectural efforts behind the modern advancements in intelligent computing and autonomous systems harness an artificial neural network at its core, to essentially accomplish a simple task, *learning*. Training the network to perform the tasks without being explicitly programmed. Theoretically, the human brain has a very low rate of operations per second when compared to a state of the art high-end GPU computer, it out-ravels the computer considering this important factor.

This project is a culmination of studying earlier efforts in this domain, *to enable design intelligent computing models to address particular problem statements assigned*; and the exploration done in regard to my research proposal to use neural network architecture for interdisciplinary applications, especially vibration analysis of machine tools to predict tool life,defects and improve the fabrication process. Recent developments illustrate applications to medical, engineering and economic fields to provide innovative solutions by use of reinforcement and recurrent networks. Modern computation uses a plethora of tools at its disposal, from open-source libraries and pre-trained models for the most complex of problems.
Deep Learning models were created and trained for three particular use-cases during the project undertaking, thereby implementing a set of concepts pertaining to each task. Denoising of patterns extracted and recovery of such data was done during the second task. An autoassociative memory model was employed for this purpose. Auto-associative neural networks are feed-forward nets trained to produce an approximation of the identity mapping between network inputs and outputs using backpropagation or similar learn-

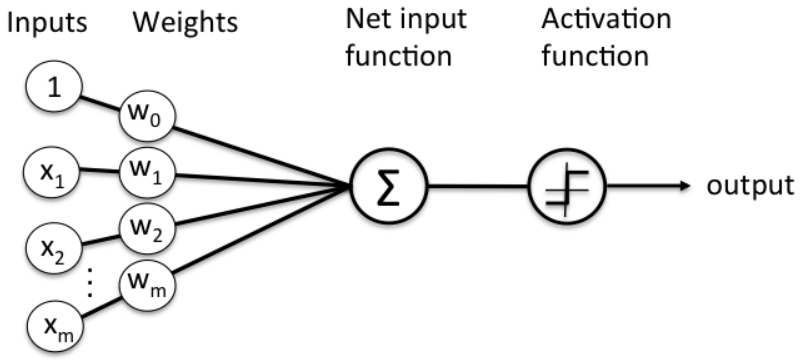
ing procedures. The key feature of an autoassociative network is a dimensional bottleneck between input and output. Some motivating applications of the networks designed in this project include domains of statistics, financial analysis, prediction systems, image and character recognition, Autonomous driving to name a few.

2.2 Background and Literature review

2.2.1 Perceptron Learning Rule

In 1943 McCulloch and Pitts introduced the first mathematical models concerning networks of neurons we call artificial neural networks. But this first step did not include any results on network training. The first work on how to train similar networks was Rosenblatt's perceptron in 1958 which is discussed below in the section.

The neurons in this network were similar to those of McCulloch and Pitts. Rosenblatt's key contribution was the introduction of a learning rule for training perceptron networks to solve pattern recognition problems. He proved that his learning rule will always converge to correct network weights, if weights that solve the problem exist.



Schematic of Rosenblatt's perceptron.

$$w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{j=0}^{m} x_j w_j$$

As illustrated above, with the expression for Rosenblatt's model the basic idea behind the perceptron was to mimic how a single neuron in the brain works: It either fires or not(if considering an extreme case of a reductionist approach). However, the perceptron network is inherently limited; as illustrated by Marvin Minsky and Seymour Papert. But, even today it remains a fast and reliable network for the class of problems that it can solve.
The Learning rule modifies the weights and biases of a network, and trains to perform some task. These generally fall into three broad categories: supervised, unsupervised and reinforcement(or graded) learning.

For supervised learning, as inputs are applied to the network the outputs are compared to the predefined target values. The rule is then used to adjust the weights and biases in order to move outputs closer to targets.
Reinforcement learning is similar, except that, instead of of being provided with correct output, the algorithm is only given a grade. The grade (or score) is a measure of the network performance over some sequence of inputs. This type of learning is currently much less common than supervised learning, mostly for control system applications.

2.2.2 Supervised Hebbian Learning

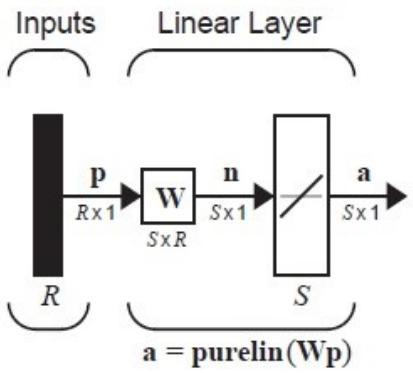
In 1949 Hebb summarized his two decades of research in *The Organization of Behavior*. The main premise of this book was that behavior could be explained by the action of neurons.This was in marked contrast to the behaviorist school of psychology (with proponents such as B. F. Skinner), which emphasized the correlation between stimulus and response and discouraged the use of any physiological hypotheses. It was a confrontation between a top-down philosophy and a bottom-up philosophy.

Hebb stated his approach: The method then calls for learning as much as one can about what the parts of the brain do, and relating the behavior as far as possible. Then seeing what further information is to be had about how the total brain works, from the discrepancy between actual behavior and the behavior that would be predicted from adding up what is known about the action of the various parts.

Postulate: *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."*

Linear Associator & Associative Memory:

Hebb's learning law can be used in combination with a variety of neural-net architectures. Consider, the linear associator network(introduced independently by *James Anderson* and *Teuvo Kohonen*):



The linear associator is an example of a type of neural network called an associative memory. The task of an associative memory is to learn Q pairs of prototype input/output vectors: $(p_1, t_1), (p_2, t_2), ..., (p_Q, t_Q)$.

$$a = Wp,$$

or

$$a_i = \sum_{j=1}^{R} w_{ij} p_j.$$

Mathematically, the Hebb rule(for training model used):

$$w_{ij}^{new} = w_{ij}^{old} + \alpha f_i(a_{iq}) g_j(p_{iq}),$$

This equation says that the change in the weight is proportional to a product of functions of the activities on either side of the synapse,

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{iq}.$$

For the supervised Hebb rule we substitute the target output for the actual output. In this way, we are telling the algorithm what the network *should* do, rather than what it is currently doing. The resulting equation is

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq}p_{jq},$$

In vector notation: $W^{new} = W^{old} + t_q p_q^T$.

If we assume that the weight matrix is initialized to zero and then each of Q the input/output pairs are applied once to the equation. This can be represented in matrix form:

$$\mathbf{W} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & \cdots & \mathbf{t}_Q \end{bmatrix} \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_Q^T \end{bmatrix} = \mathbf{T}\mathbf{P}^T,$$

Thereafter, we implement performance analysis of the network utilizing the learning rules and in case of non-orthogonal input patterns, the Psuedoinverse rule, $W = TP^+$; where P^+ is the Moore-Penrose psuedoinverse.

The pseudoinverse of a real matrix $\mathbf{P}$ is the unique matrix that satisfies

$$\mathbf{P}\mathbf{P}^+\mathbf{P} = \mathbf{P},$$

$$\mathbf{P}^+\mathbf{P}\mathbf{P}^+ = \mathbf{P}^+,$$

$$\mathbf{P}^+\mathbf{P} = (\mathbf{P}^+\mathbf{P})^T,$$

$$\mathbf{P}\mathbf{P}^+ = (\mathbf{P}\mathbf{P}^+)^T.$$

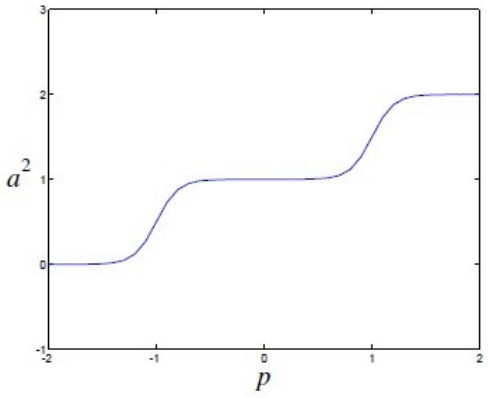
When the number, R, of rows of P is greater than the number of columns, Q, of P, and the columns of P are independent, then the pseudoinverse can be computed by $P^+ = (P^T P)^{-1} P^T$.

2.2.3 Backpropagation

The perceptron learning rule of Frank Rosenblatt and the LMS algorithm of Bernard Widrow and Marcian Hoff were designed to train single-layer perceptron-like networks. However, these single-layer networks suffer from

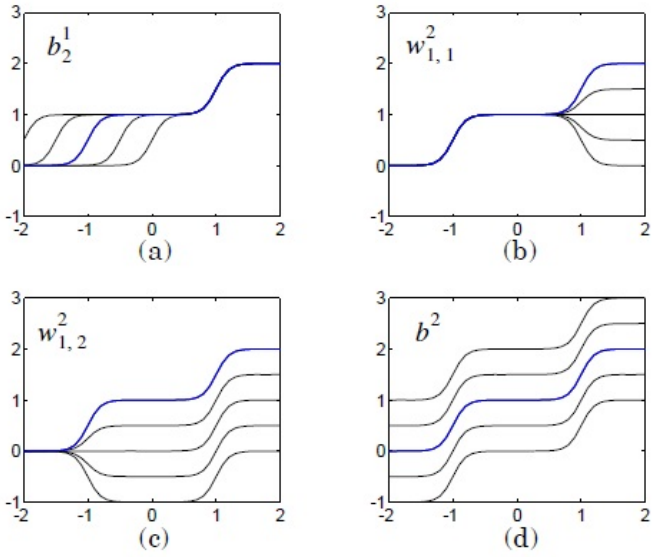
the disadvantage that they are only able to solve linearly separable classification problems. Both Rosenblatt and Widrow were aware of these limitations and proposed multilayer networks that could overcome them, but they were not able to generalize their algorithms to train these more powerful networks.

Apparently the first description of an algorithm to train multilayer networks was contained in the thesis of Paul Werbos in 1974.This presented the algorithm in the context of general networks, with neural networks as a special case, and was not disseminated in the neural network community. It was not until the mid 1980s that the backpropagation algorithm was rediscovered and widely publicized. It was rediscovered independently by David Rumelhart, Geoffrey Hinton and Ronald Williams, David Parker, and Yann Le Cun.



Nominal response of Two-layer XOR Network(from code).

The multilayer perceptron, trained by the backpropagation algorithm, is currently the most widely used neural network. The implementation explored during the third task of this project initially explores XOR problem with a two-layer network and explored function approximation. Based on the nominal response feedback, it illustrated the effects of parameter changes on the network.



Effect of parameter changes on network response.

Backpropagation Algorithm:

As we know, for multilayer networks the output of one layer becomes the input to the following layer. This can be represented as $a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1})$ for $m = 0, 1, ..., M-1$; where M is number of layers in the network. The neurons in first layer receive external inputs, as the starting point. This propagates along the network and output of neurons in last layer is considered network outputs.

$$a = a^M.$$

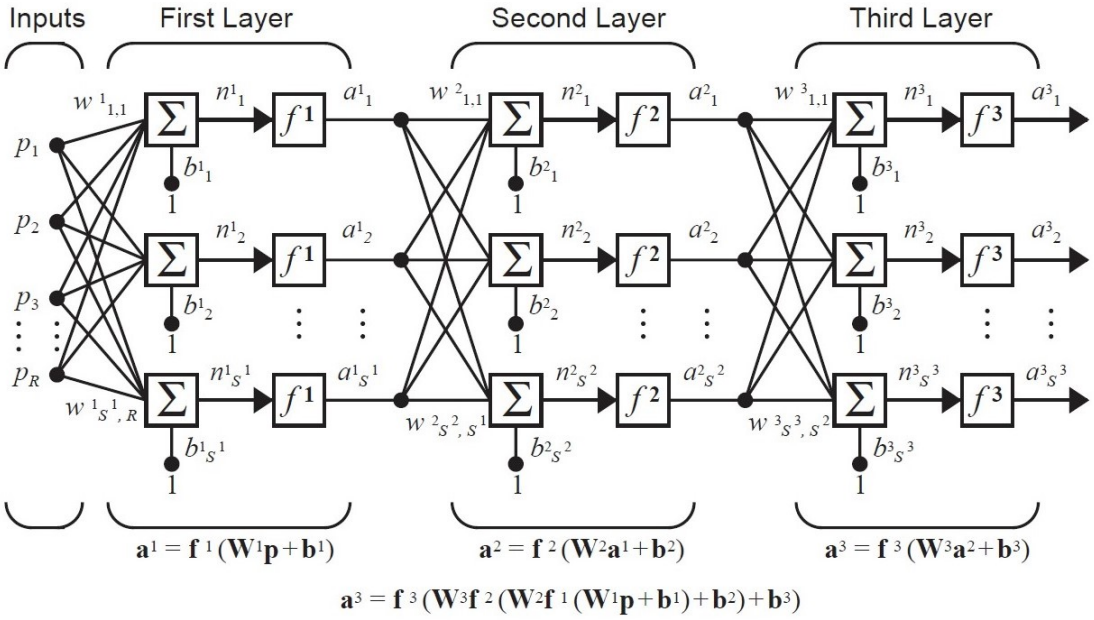


Fig. Three-layer network, Abbreviated Notation.

This algorithm is a generalization of the LMS algorithm, and both use the same performance index: *mean square error*.

$$F(x) = E[e^T e] = E[(t-a)^T(t-a)].$$

By employing steepest descent to generate approximate mean square error, we later address the computation of the partial derivatives by Chain rule. The third task of this project explores ADALINE networks and concepts like sensitivity, batch and incremental training. This is later applied to develop a convergence model for time series analysis of a particular problem assigned.

2.3 Recent Research

TensorFlow, Pytorch, pandas, AWS Deep Learning, CoreML(iOS).

Few of the aforementioned terms might be obscure to anyone dabbling in the field of intelligent computing. Deep learning is all around us. It‘s used to determine which online advertisements are to display in real time, identify and tags friends in photographs, translate your voice to text, translate text into different languages on a Web page(now images too), and drive autonomous vehicles.
Deep learning is also found in less visible places. Credit card companies use deep learning for fraud detection; businesses use it to predict whether or not you will cancel a subscription and provide personalized customer recommendations; banks use it to predict bankruptcy and loan risk; hospitals use it for detection, diagnosis, and treatment of diseases. Google, Amazon, Netflix and other monolithic online platforms use it to deliver semantic results based on algorithms that analyze a user's search, purchase and viewing history to predict what is it they're looking for or more likely to want. The data they have at their disposal is massive.

The range of applications is almost limitless. Other options include text analysis, image captioning, image colorization, x-ray analysis, weather forecasts, finance predictions, and more. Deep learning is already being widely used to automate processes, improve performance, detect patterns, and solve problems. All this can be termed under *The Exponential explosion of Available Data*. Without this data, training ANNs containing millions of connections and thousands of nodes could not happen. For an ANN to recognize a face, detect credit fraud, or translate a voice to text in a noisy room it takes more than just a few bits of test data for consistent, accurate predictions. This is why *ANNs* flourish in the age of big data.

The rise of GPU is another important factor for this. Making a neural network run fast is difficult. Hundreds or thousands of neurons must interact with each other in parallel. Depending on the task, it could take weeks for traditional CPUs to generate a prediction from an ANN. With GPUs, the same task that took weeks may only take days or hours.
To illustrate the power of GPUs, Andrew Ng replicated the Google X project with a network with 11 billion connections running on 16 computers powered by just 64 GPUs the previous project used 1,000 computers with 16,000 CPUs. The new project did not run too much faster or perform better, but Ng made his point.

Sixty-four GPUs could handle the same amount of work as 16,000 CPUs in roughly the same amount of time. Pair this with the inception of the current Advanced Algorithms like CapsuleNets, and we can see that heightened processing power going hand-in-hand with them to achieve and solve even more complex problems.

Chapter 3

Networks and Implementation

3.1 Perceptron architecture:

Hamming and Hopfield networks

While determining the unified learning rule and activation function for the model, these two networks were studied in detail. Feedforward and recurrent layers were trained with numerous input trails and *poslin* function was employed in the recurrent layer. The observation for this implementation was effect of the layer is to zero out all neuron outputs, except the one with largest initial value.

As a contrast to the Hamming network, Hopfield network actually produces selected prototype patterns as its output. After the program trails of these networks, the following architecture was designed using *hardlim* as training algorithm.

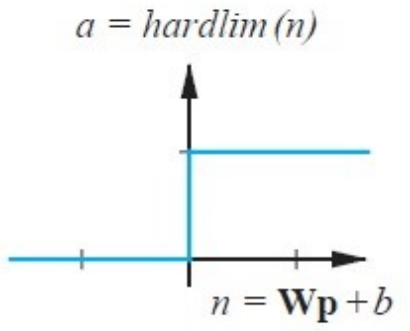
$$a = hardlim(Wp + b)$$

Consider, the network weight matrix:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}.$$

After defining a vector composed of the elements of ith row of W, we partition the weight matrix. This allows us to write the ith element of the network

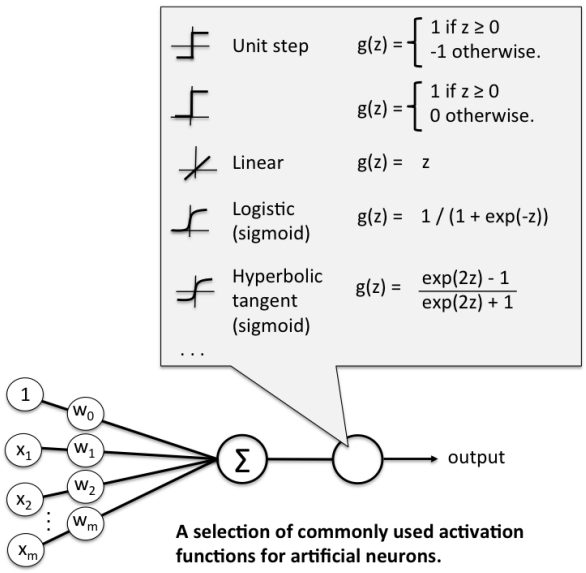
output vector as: $a_i = hardlim(n_i) = hardlim({}_i w^T p + b_i)$



This architecture has been implemented for the first task during the project, and decision boundary for a two-input perceptron network is obtained based on the learning rule.

3.1.1 Log sigmoid & Rosenblatt's model

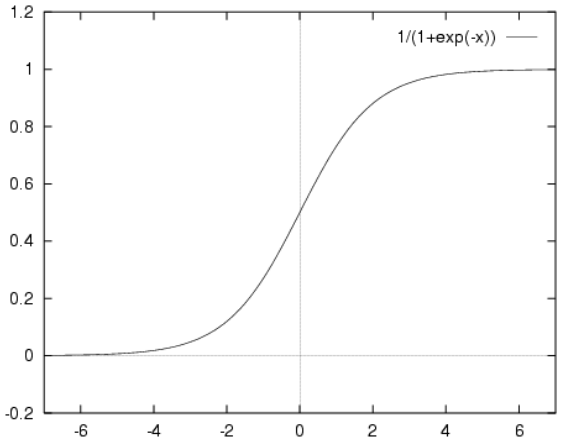
Among the activation functions utilized for recurrent networks, the current model was based on the log-sigmoid function and implemented as an extension of Rosenblatt model architecture.



A selection of commonly used activation functions for artificial neurons.

Considering the Rosenblatt's perceptron (introduced in 1958), the activation function was used to explore how output was generated. This helped in determining that '*hardlims*' was required for the desired model. The softmax function of the implementation used was

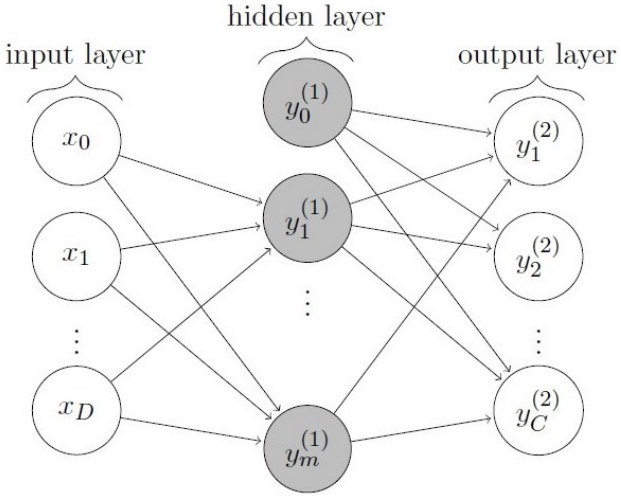
$$\sigma(z, i) = \frac{\exp(z_i)}{\sum_{k=1}^{C} \exp(z_k)}$$



An illustration of the log-sigmoid function.

The model required a network to analyze weight and bias inputs. Based on the above function, several layers were deployed with each having its own weight matrix $\mathbf{W}$, own bias vector $\mathbf{b}$, net.input vector n and net.output vector a.

3.1.2 Multi-layer perceptron



An illustration of multi-layer architecture in a perceptron.

By implementing multiple layers of neurons across the network, the learning rule was studied until arriving upto a unified rule. Training begins by assigning some initial values for the network parameters. By presenting them to the network, we obtain p_1. This value is provided as input to next layer and verified with the target values. In case of error or changes, we define the new variable, the perceptron error e: $e == t - a$.
Hence, three rules for various conditions of error obtained can be unified as a single expression: ${}_1w^{new} =_1 w^{old} + ep =_1 w^{old} + (t-a)p$.
Perceptron learning rule for matrix notation,

$$W^{new} = W^{old} + ep^T,$$

and

$$b^{new} = b^{old} + e.$$

The key characteristic of single layer perceptron is that it creates linear decision boundaries to separate categories of input vector(s). A network with two inputs and single neuron was created and decision boundary was plotted accordingly. Upon its successful implementation, the final model had four inputs as weights W, and bias b to train and obtain desired target values.
With sample problems, a code was implemented in Matlab to generate weights and bias for each iteration until convergence is achieved by cycling through the input vectors. Consider the example:

p1	p2	p3	p4
2	1	-2	-1
2	-2	2	1

Taking initial value as null for weight and bias, perceptron learning rule was applied and weights and bias were generated. $W(0) = [00]^T, b(0) = 0$.

Target values
0
1
0
1

Weight values: $w = [a\, b]^T$.

-2	-2	-2	-3	-3	-2
-2	-2	-2	-1	-1	-3

and, bias values:

-1	-1	-1	0	0	1

This code was tested with several problems and decision boundaries for inputs in each case have been accurate. However, if we consider larger number of inputs and scattered plots, the issue of linear inseparability arises.

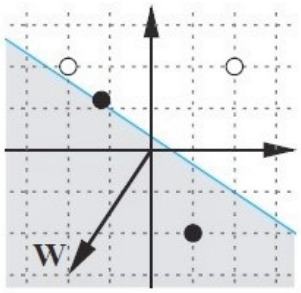
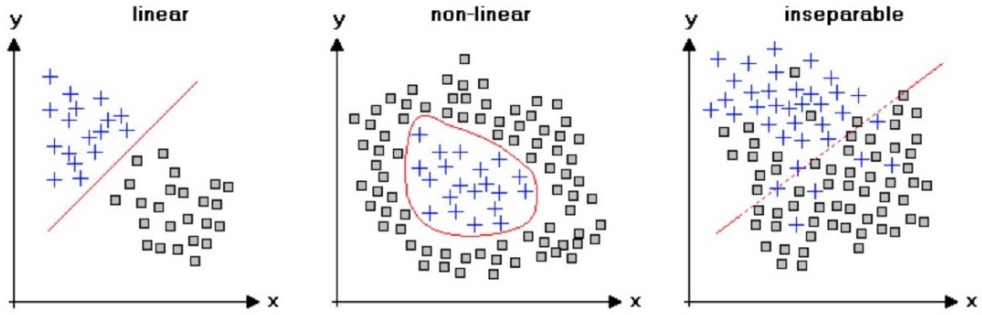


Fig. Decision boundary for the problem(from code).

Linear inseparability(limitation):



The perceptron learning rule is guaranteed to converge a solution in a finite number of steps, so long as a solution exists. The boundary between regions(for network implemented in this project) is defined by:

$$_iw^T p + b = 0.$$

This is a linear boundary (hyperplane). The perceptron can be used to classify input vectors that can be separated by a linear boundary.
Such vectors are referred as linearly separable.It was the inability of basic perceptron to solve such simple problems that led, in part, to a reduction in interest in neural network research during the 1970s. Rosenblatt had investigated more complex networks, which he felt would overcome the limitations of basic perceptron, but he was never able to effectively extend the perceptron rule to such networks.

3.2 Autoassociative network:

We implement the pattern extraction and recognition problem by using an autoassociative memory, where the desired output vector is equal to the input vector(i.e., $t_q = p_q$).

$$W = p_1 p_1^T + p_2 p_2^T + p_3 p_3^T.$$

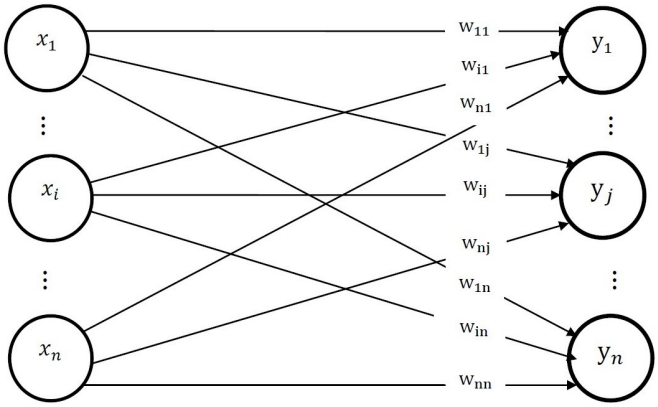


Fig. Illustration of autoassociative architecture in neural network.

The autoassociative network is employed for recovery and extraction of occluded(50% and 67%) and noisy patterns from image scans, as further discussed in the next chapter. Apart from the application to the problem in this project, in general, there have been a number of variations on the basic Hebb rule. For instance, the delta rule[1](Widrow-Hoff algorithm) to minimize the sum of squares of errors, filtered learning[2] and also unsupervised Hebb rule.

3.2.1 Pattern Extraction

An autoassociative memory model was used to store a set of patterns and then to recall these patterns, even when corrupted patterns are provided as input. Training rule for the code employed functions like'crossentropy' and 'trainscg'[3] to extract numerical code of (-1, 1) as vectors to squares across the grid for a digit.

[1] Delta rule: $W^{new} = W^{old} + \alpha(t_q - a_q)p_q^T$.

[2] Filtered Learning rule: $W^{new} = (1-\gamma)W^{old} + \alpha t_q p_q$.

[3] Scaled Conjugate Generalization(backpropagation).

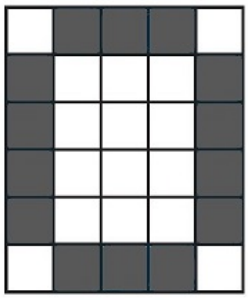


Illustration of grid for digit scan, assigned to 1, -1 values.

The above figure is initialized as 6x5 grid, one column at a time. The prototype pattern of vectors assigned will be an array of [-1 1 .... 1]^T assigned to p_1. This corresponds to the digit 0, p_2 to digit 1, and so on. Using Hebb rule, the weight matrix is computed $W = p_1 p_1^T + p_2 p_2^T + p_3 p_3^T$.

The given memory model was utilized and following results were obtained for the code.

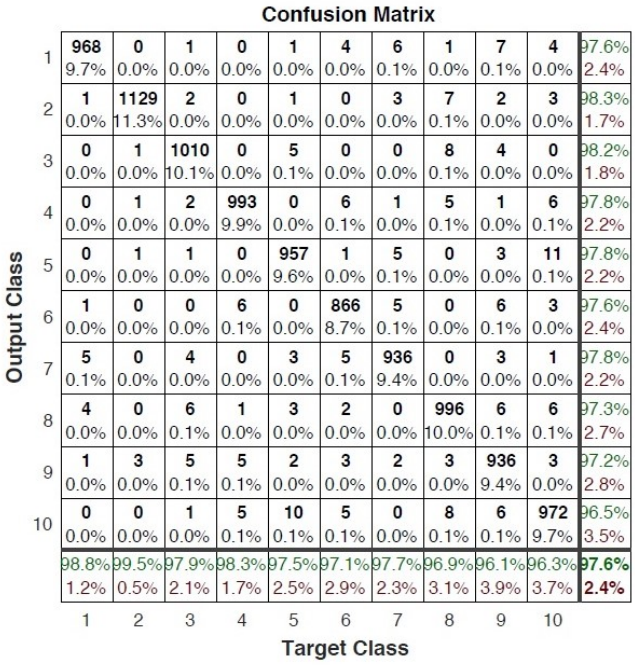


Fig. Confusion matrix obtained from extracted patterns in MNIST.

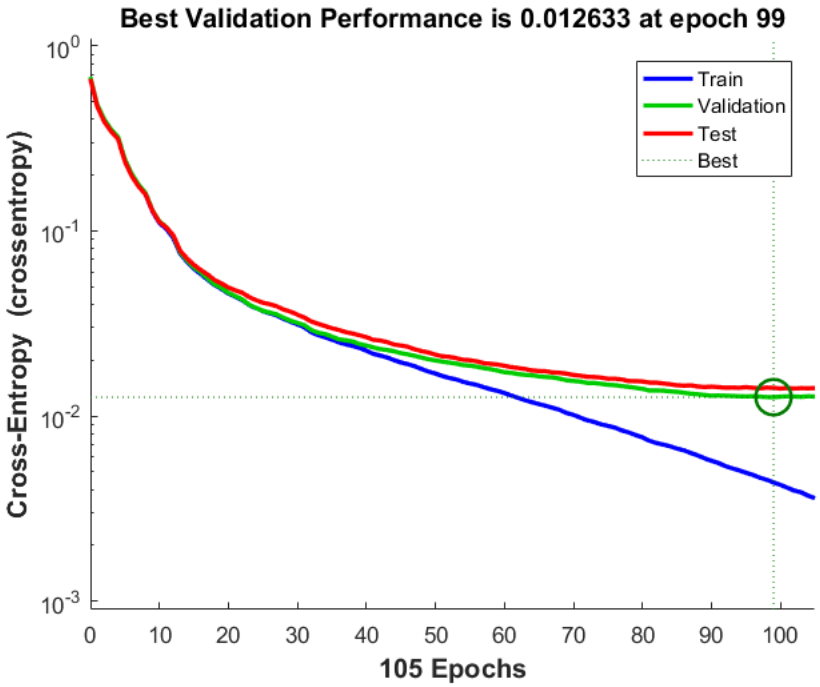


Fig. Performance plot of the code upon Matlab.

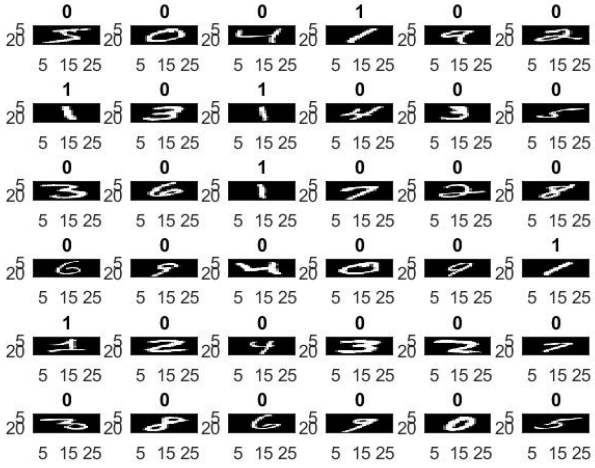


Fig. Number plot for recovery of patterns.

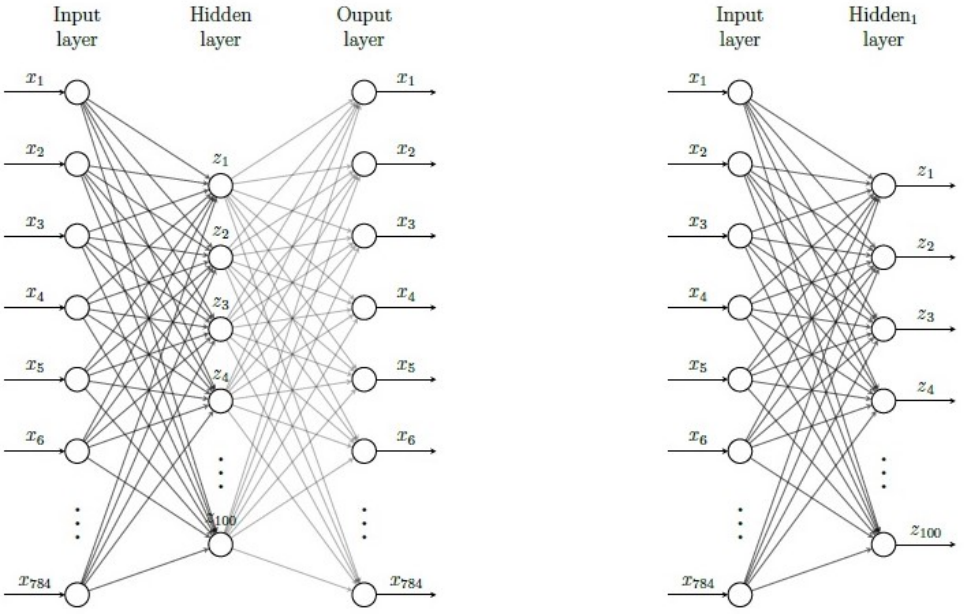
3.2.2 Recovery and denoising:

The code was tested to recover occluded patterns from provided images in a dataset. The final illustration in the above page shows how values of digit can be estimated or determined to a certain accuracy based on the vector assignment of grid input scans. The code implemented also specifies a probability of each digit for the occluded pattern.
De-noising of images is not particular to thee networks or the application, but is harnessed in a plethora of domains. Eigen vectors and values are assigned for higher order analysis in case of image scans of structural elements. The project submission was extended for similar use in assessing and denoising structural images. An illustration of denoising digits below sums up the work done during the second task for the project.

These networks have observed further changes in terms of application, as modern architectures like autoencoders are also used to significantly improve the prediction and recovery of such patterns, from even larger datasets.

3.3 Hidden layer Neural networks

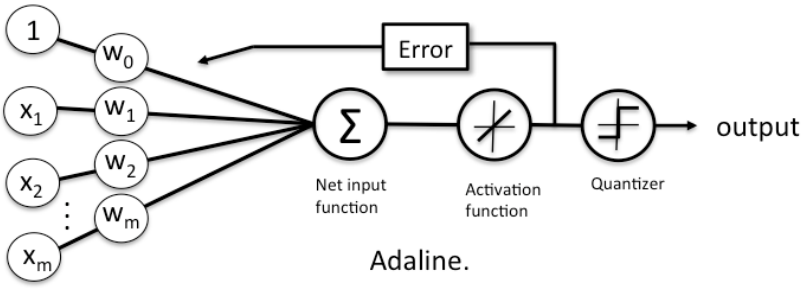
Extending upon the pattern classification , a XOR layout was implemented to explore the shortcomings of the model(single layer perceptron). Individual decision boundaries were obtained and later function approximation, adaptive filtering for the network, exploring the nominal response. It would appear that we could use such networks to approximate almost any function, if we had a sufficient number of neurons in the hidden layer.



(a) Full architecture (b) Removed output layer

In fact, it has been shown that two-layer networks, with sigmoid transfer functions in the hidden layer and linear transfer functions in the output layer, can approximate virtually any function of interest to any degree of accuracy, provided sufficiently many hidden units are available.

3.3.1 ADALINE network



Adaline.

An ADALINE neural network model was created and implemented based on the LMS algorithm for the convergence and response observations, thereby studying stochastic gradient descent algorithm with the code.

It is instructive to view network here as a function approximator in this regard, to develop the multi-layer(with hidden layers as well) model for backpropagation. Unlike single-layer linear network, the error is not an explicit function of weights in the hidden layers for a multilayer network, thereby partial derivatives are not computed easily.

3.3.2 Backpropagation

After training the model based on performance index, sensitivity and generation of a chain rule to generalize the expressions and the algorithm. The process of backpropagating the sensitivities, that gives us the term *backpropagation*, because it describes a recurrence relationship in which the sensitivity at layer m is computed from the sensitivity at layer $m+1$.

The recurrence relationship for sensitivities was derived using:

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}.$$

Next we want to find an expression for this matrix. Consider the i, j element of the matrix:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial\left(\sum_{l=1}^{s^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right)}{\partial n_j^m} = w_{i,j}^{m+1}\frac{\partial a_j^m}{\partial n_j^m}$$

$$= w_{i,j}^{m+1}\frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1}\dot{f}^m(n_j^m),$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}.$$

The sensitivities are propagated backward through the network from the last layer to the first layer: $s^M \rightarrow s^{M-1} \rightarrow ... \rightarrow s^2 \rightarrow s^1$.

In contrast to LMS algorithm, backpropagation has the only complication is that in order to compute the gradient we need to first backpropagate the sensitivities. The beauty of backpropagation is that we have a very efficient implementation of the chain rule.

Further implementations were used to compare batch vs. incremental training for the function approximation programs, and this was the basis for the final assignment.

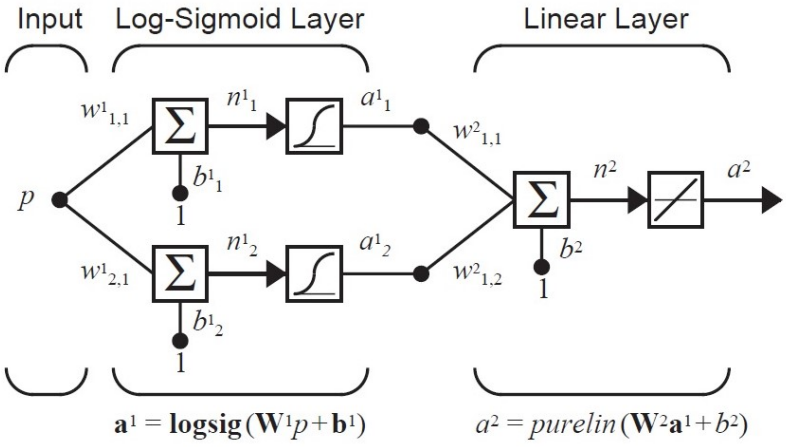
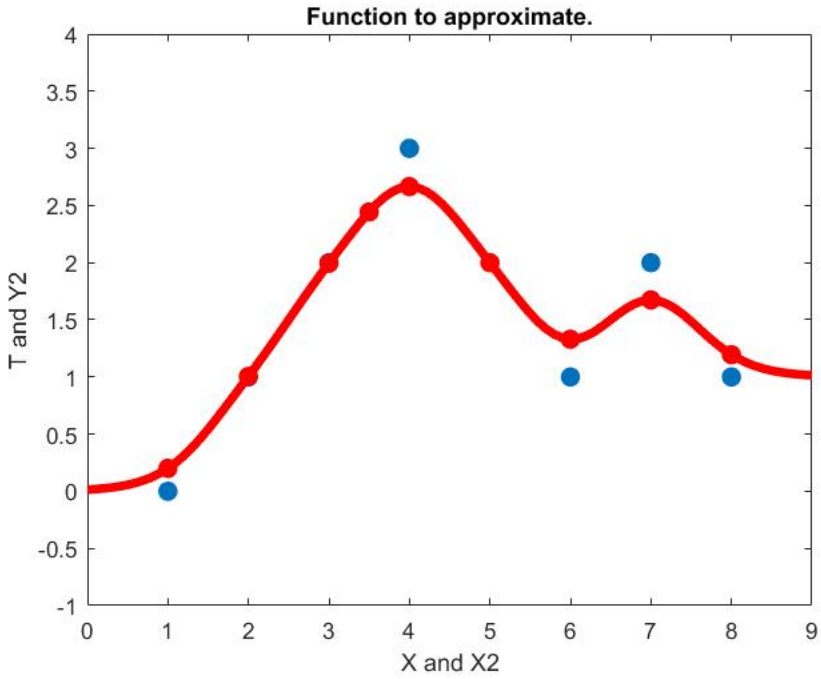


Fig. Example Function Approximation Network

Time Series Analysis



An extension for the function approximation was to be implemented for each trial of the algorithm, thereby it needed to remap the performance and network parameters for each iteration. The dataset in use was to be updated regularly, based on findings and results from the experiments. This model was built keeping in perspective that requirement, as Time series analysis comprises methods for analyzing the data in order to extract meaningful statistics and other characteristics of the data.

Applications The usage of time series models is twofold:
1. Obtain an understanding of the underlying forces and structure that produced the observed data
2. Fit a model and proceed to forecasting, monitoring or even feedback and feedforward control.
Time series are widely used for non-stationary data, like economic, weather, stock price, and retail sales in this post. However, for the current submission, a retail sales data was used to design the code and exploratory analysis, curve fitting techniques were employed. The prediction and forecasting logic was implemented for a furniture sales dataset and then extended to the structural image dataset for stochastic simulation. The complete project was implemented without any dependencies and toolboxes, but this required

automated statistical software and was hence deployed upon the pandas library(Python).

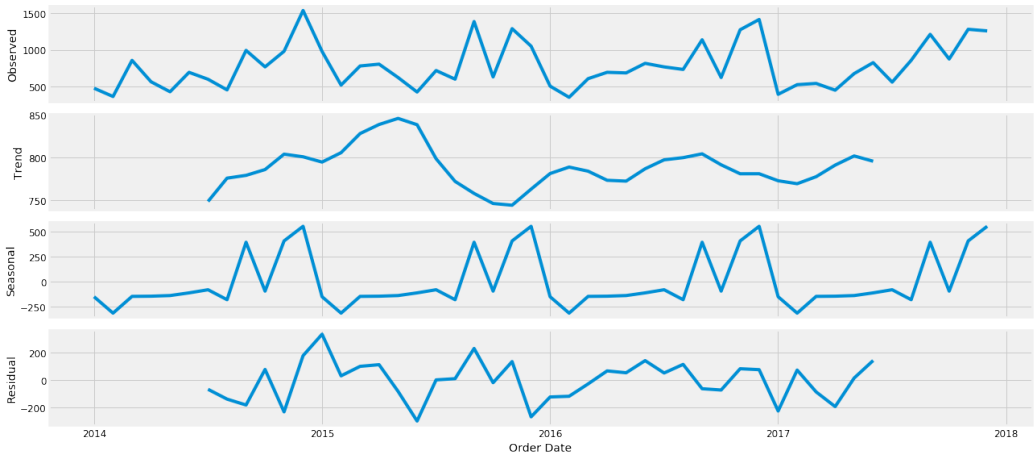


Fig. Decomposition plot for each parameter in time series on the sample sales data.

ARIMA method was used for the forecasting , as the predictions studied seasonality, trend and noise in the sample sales data. Further elucidation is provided in the code(6.2, in Appendices). The objective was to use a grid search to find the optimal set of parameters that yields the best performance for our model. This training model was later used for the data samples and successfully implemented.

Extension of concept: The multivariate form of the Box-Jenkins univariate models, is sometimes called the ARMAV model, for AutoRegressive Moving Average Vector or simply vector ARMA process. An architecture similar to this was deployed on Matlab with the neural-net toolbox for further extension, and this proposal has been used on the experiment data after submission.

Chapter 4

Future Work

This project was also enabling me to demystify neural networks and the principles of operation to these complex architectures, but this project is unfortunately too short to cover the expanse of research in the domain of image recognition or backpropagation or even perceptrons, in general. The recent advancements include using techniques like deep reinforcement learning, a type of neural-net that earns by interacting with the environment through observations, actions, and rewards. It has even been used to learn gaming strategies like Atari and AlphaGo.

Perhaps the most interesting and impressive architecture has been developed since the advent Generative Adversarial Networks(GANs), which pair neural-nets to spur learning and lighten the processing load. GANs open up deep learning to a larger range of unsupervised tasks in which labeled data does not exist or is too expensive to obtain. They also reduce the load required for a deep neural network because the two networks share the burden. Expect to see more business applications, such as cyber detection, employ GANs.
Probabilistic programming due to the above improvements have the ability to accommodate the uncertain and incomplete information that is so common in the business domain. Autoencoders are being implemented with other techniques for Hybrid architecture (*eg.* Bayesian conditional GANs), in recent times, and leading to automated machine learning models(AutoML).

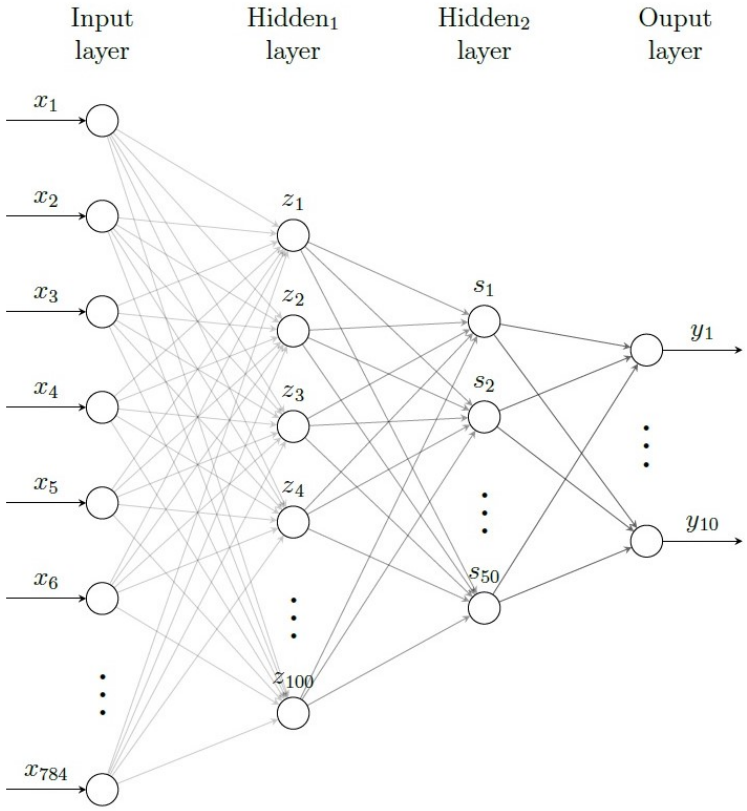


Fig. An illustration of deep neural network.

As an extension of this project, I intend to explore further into Fuzzy systems and Explainable Artificial Intelligence, as the prediction systems from data have been developing a staggering amount of precision and accuracy by employing automated model creation without programming.
AI that is explainable, provable, and transparent will be critical to establishing trust in the technology and will encourage wider adoption of machine learning techniques. Enterprises will adopt explainable AI as a requirement or best practice before embarking on widespread deployment of AI, while governments may make explainable AI a regulatory requirement in the future.

Chapter 5

Conclusion

Neural network is an application of modern intelligent computing to learn and try to replicate a model based on the *human brain*. In neural nets the computation of components are performed in a parallel form. In this project, the training rules, algorithms and applications of neural networks were studied in detail. I have explored various topics and use-cases of Neural Nets, with emphasis to types of Network Architecture and Learning Algorithms. A performance improvement of a neural network was achieved through learning process, which has been used to develop the multi-neuron perceptron model(s).
Later we explored Supervised Learning, pseudo-inverse rule before developing the Auto-associative Network models. The application of Hebb Rule enables computing the optimal weight matrix in feed-forward neural nets comprising of two layers: input and the target output layer.
With 'crossentropy' and 'trainscg' functions to train, and thereby utilizing Scaled conjugate backpropagation for updating the weight values. As an application, digit recognition based on the pattern extraction algorithm was trained on the MNIST dataset using a two-layer perceptron. This algorithm was further extended to recover occluded and noisy digit patterns as well.

Finally, a model was deployed for time series forecast analysis, based on backpropagation algorithm. The model achieved convergence to a local minimum, with LMS algorithm as the principle for the architecture. It had been trained with hidden neuron layers(ADALINE) for further extension, to improve the network approximation while studying generalization[1].

[1] For a network to generalize, it should have fewer parameters than there are data points in the training set.

In conclusion, artificial neural networks are a powerful tool applicable to a wide range of problems not just in the domain of data analysis, but across inter-disciplinary fields for various solutions[2].

[2] During the course of this project, network models which were designed and implemented; in Matlab, was without use of any toolboxes or dependencies.

Chapter 6

Appendices

6.1 Codes Implemented in Matlab

6.1.1 Logistic Sigmoid

```
function y = logisticSigmoid(x)
% simpleLogisticSigmoid Logistic sigmoid activation function
%
% INPUT:
% x     : Input vector.
%
% OUTPUT:
% y     : Output vector where the logistic sigmoid was applied element by
% element.
%

    y = 1./(1 + exp(-x));
end
```

```
function y = dLogisticSigmoid(x)
% dLogisticSigmoid Derivative of the logistic sigmoid.
%
% INPUT:
% x     : Input vector.
%
% OUTPUT:
% y     : Output vector where the derivative of the logistic sigmoid was
% applied element by element.
%
    y = logisticSigmoid(x).*(1 - logisticSigmoid(x));
end
```

6.1.2 Perceptron Learning demo on 2D data

```
% Normal Dataset
%X =
    [0,0;0,1;0,2;0,3;0,4;1,0;1,1;1,2;1,3;1,4;2,0;2,1;2,2;2,3;2,4;3,0;3,1;3,2;3,3;3,4;
%    4,0;4,1;4,2;4,3;4,4];
% y =
    [-1;-1;-1;-1;-1;+1;-1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;+1;-1;-1;+1;+1;+1;+1;-1];
% Some other y
% y =
    [+1;+1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;-1;-1;-1;];
% Some -1's are gone
%X = [0,0;0,1;0,2;1,0;1,1;1,2;1,3;2,0;2,1;2,2;2,3;2,4;3,0;3,1;3,2;3,3;3,4;
%    4,0;4,1;4,2;4,3;4,4];
%y = [-1;-1;-1;+1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;+1;-1;-1;+1;+1;+1;+1;-1];

% With added mass
X =
    [0,0;0,1;0,2;0,3;0,4;1,0;1,1;1,2;1,3;1,4;2,0;2,1;2,2;2,3;2,4;3,0;3,1;3,2;3,3;3,4;
4,0;4,1;4,2;4,3;4,4;0,-6;1,-6;2,-6;3,-6;4,-6;5,-6;6,-6];
y =
    [-1;-1;-1;-1;-1;+1;-1;-1;-1;-1;+1;+1;-1;-1;-1;+1;+1;+1;-1;-1;+1;+1;+1;+1;-1;
+1;+1;+1;+1;+1;+1;+1];

% Adding column of 1's in X for y-intercept of the line
X = [X,ones(length(y),1)];
% Least Square
%w = X\y;

w = [0,0,0];
clf
while(sum(sign(w*X')~=y')>0)
for i = 1:length(y)
scatter(X(:,1),X(:,2),[],y,'filled')
hold on
% Perceptron
if(sign(w*X(i,:)')~=y(i))
w = w + X(i,:)*y(i);
end
strEq = sprintf('%f*x + %f*y + %f = 0',w(1),w(2),w(3));
xx = -2:0.01:6;
yy = -(w(1)/w(2))*xx - (w(3)/w(2));
scatter(xx,yy,'.');
hold off
title(strEq);
drawnow
pause(0.1)
end
end
```

6.1.3 Practice model of Rosenblatt perceptron

```
clear all;
r=10;
w=6;
d=-1;
rand('state',0);
rho=r+(rand(1000,1)-1/2)*w;
rand('state',4);
theta1=pi*rand(1000,1);
D=zeros(2000,2);
D(1:1000,:)=[rho.*cos(theta1),rho.*sin(theta1)]; %the upper moon

theta2=pi+pi*rand(1000,1);
D(1001:2000,:)=[r+rho.*cos(theta2),d+rho.*sin(theta2)];
m=3;
mu=.1;
itermax=50; %the maximum of iterative
x(1,:)=ones(1,2000);
x(2,:)=D(:,1)';
x(3,:)=D(:,2);
w=zeros(m,1);
goal(1:1000,1)=ones(1000,1); %target
goal(1001:2000,1)=-1*ones(1000,1);
y=ones(2000,1);% output if choose y=-ones(2000,1), this means if w'*x<=0,
% x belongs to the first category
k=1;
while k<=itermax
for i=1:2000
w=w+mu*(goal(i)-y(i))*x(:,i);
%        w=w+x(:,i);
y(i)=sign(w'*x(:,i));
end
error(k)=length(find(y~=goal))/length(goal);
if y==goal
break;
end
k=k+1;
end
figure(1);
plot(D(:,1),D(:,2),'.');
f=@(x) -w(1)/w(3)-w(2)/w(3).*x;
s=-15:0.1:25;
hold on; plot(s,f(s),'k');
figure(2);
plot(error)
```

6.1.4 Task 1: Multi-neuron perceptron model

```
clear all; clc;
t=[0;1;0;1];
p1=[2;2]; p2=[1;-2]; p3=[-2;2]; p4=[-1;1];
p=[p1 p2 p3 p4]
W(:,1)=[0;0]; %let initial values be null for weight and bias
b(1)=0
w = W; bias = b;
count = 0;
count1 = 0;
j = 1;
for i=1:50
    a(j)=hardlim(w'*p(:,j)+bias);
    if a(j)==t(j)
        W(:,i+1)=W(:,i);
        b(i+1)=b(i);
    else
        e=t(j)-a(j);
    W(:,i+1)=W(:,i)+e*(p(:,j));
    b(i+1)=b(i)+e;
    end
    w = W(:,i+1);
    bias = b(i+1);
    count = count + 1;
    j = j + 1;
    if count == length(p)
        count = 0;
        for k = 1:length(p)
            if a(k) == t(k);
                count1 = count1 + 1;
            end
            j = 1;
        end
    end
    if count1 == length(p)
        break
    else
        count1 = 0;
    end
end
```

6.1.5 Task 2: Autoassociative network model for Digit Recognition

```
clc;
clear all;
images = loadMNISTImages('train-images.idx3-ubyte'); % initialize figure
labels = loadMNISTLabels('train-labels.idx1-ubyte'); % initialize figure
labels = labels';                                    % transpose
labels(labels==0)=10;                                % dummyvar function
    doesnt take zeroes
labels=dummyvar(labels);                             %

% initialize figure
figure
colormap(gray)

%trying for 6 x 6 samples from mnist.mat dataset
for i = 1:36
subplot(6,6,i)
digit = reshape(images(:, i), [28,28]);
imagesc(digit)
title(num2str(labels(i)))              % to show label
end

x = images;
t = labels';
trainFcn = 'trainscg';
% use scaled conjugate gradient for training

hiddenLayerSize = 100;
net = patternnet(hiddenLayerSize);
% to create Pattern Recognition Network

% In order to assess the performance,
% you have to define test and validation sets and a performance metric.
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

net.performFcn = 'crossentropy'; % using Cross-Entropy to test and
    validate the function.

[net,tr] = train(net,x,t);
% net saves info about network

%tr gives information about the training record
```

6.1.6 Task 3: Function approximation model with hidden neuron layers

```
% code to generate the figure

function createfigure(X1, YMatrix1, X2, Y1, X3, Y2)
%CREATEFIGURE(X1, YMATRIX1, X2, Y1, X3, Y2)
%  X1:  vector of x data
%  YMATRIX1: matrix of y data
%  X2:  vector of x data
%  Y1:  vector of y data
%  X3:  vector of x data
%  Y2:  vector of y data

%  Auto-generated by MATLAB on 12-Jul-2018 12:07:55

% Create figure
figure1 = figure;

% Create axes
axes1 = axes('Parent',figure1);
hold(axes1,'on');

% Create multiple lines using matrix input to plot
plot1 = plot(X1,YMatrix1,'MarkerSize',30,'Marker','.','LineStyle','none');
set(plot1(2),'Color',[1 0 0]);

% Create plot
plot(X2,Y1,'MarkerSize',30,'Marker','.','LineStyle','none','Color',[1 0
    0]);

% Create plot
plot(X3,Y2,'LineWidth',4,'Color',[1 0 0]);

% Create xlabel
xlabel('X and X2','FontSize',11);

% Create title
title('Function to approximate.','FontSize',11);

% Create ylabel
ylabel('T and Y2','FontSize',11);

% Uncomment the following line to preserve the X-limits of the axes
% xlim(axes1,[0 9]);
% Uncomment the following line to preserve the Y-limits of the axes
% ylim(axes1,[-1 4]);
box(axes1,'on');
```

6.1.7 Function approximation model : Final code

```
X = [1 2 3 4 5 6 7 8];
T = [0 1 2 3 2 1 2 1];

plot(X,T,'.','markersize',30)
axis([0 9 -1 4])
title('Function to approximate.')
xlabel('X')
ylabel('T')

spread = 0.7;
net = newgrnn(X,T,spread);
A = net(X);

hold on
outputline = plot(X,A,'.','markersize',30,'color',[1 0 0]);
title('Create and test y network.')
xlabel('X')
ylabel('T and A')

x = 3.5;
y = net(x);
plot(x,y,'.','markersize',30,'color',[1 0 0]);
title('New input value.')
xlabel('X and x')
ylabel('T and y')

X2 = 0:.1:9;
Y2 = net(X2);
plot(X2,Y2,'linewidth',4,'color',[1 0 0])
title('Function to approximate.')
xlabel('X and X2')
ylabel('T and Y2')
```

6.2 Time series analysis model(Python)

```
import warnings
import itertools
import numpy as np
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import pandas as pd
import statsmodels.api as sm
import matplotlib

matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'

df = pd.read_excel("Superstore.xls")
furniture = df.loc[df['Category'] == 'Furniture']

furniture['Order Date'].min()

furniture['Order Date'].max()

cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID',
    'Customer Name', 'Segment', 'Country', 'City', 'State', 'Postal
    Code', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product
    Name', 'Quantity', 'Discount', 'Profit']
furniture.drop(cols, axis=1, inplace=True)
furniture = furniture.sort_values('Order Date')

furniture.isnull().sum()

furniture = furniture.groupby('Order Date')['Sales'].sum().reset_index()

furniture.head()

furniture = furniture.set_index('Order Date')
furniture.index

y = furniture['Sales'].resample('MS').mean()
y['2017':]
y.plot(figsize=(15, 6))
plt.show()

from pylab import rcParams
rcParams['figure.figsize'] = 18, 8

```

```
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
plt.show()

##Using ARIMA for time series forecasting##
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p,
    d, q))]

print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))

for param in pdq:
for param_seasonal in seasonal_pdq:
try:
mod = sm.tsa.statespace.SARIMAX(y,
order=param,
seasonal_order=param_seasonal,
enforce_stationarity=False,
enforce_invertibility=False)

results = mod.fit()

print("'ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal,
    results.aic)")
except:
continue

mod = sm.tsa.statespace.SARIMAX(y,
order=(1, 1, 1),
seasonal_order=(1, 1, 0, 12),
enforce_stationarity=False,
enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])

results.plot_diagnostics(figsize=(16, 8))
plt.show()

##Validating forecasts##
pred = results.get_prediction(start=pd.to_datetime('2017-01-01'),
    dynamic=False)
pred_ci = pred.conf_int()
```

```
90
91  ax = y['2014':].plot(label='observed')
92  pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast',
        alpha=.7, figsize=(14, 7))
93
94  ax.fill_between(pred_ci.index,
95  pred_ci.iloc[:, 0],
96  pred_ci.iloc[:, 1], color='k', alpha=.2)
97
98  ax.set_xlabel('Date')
99  ax.set_ylabel('Furniture Sales')
100 plt.legend()
101
102 plt.show()
103
104 y_forecasted = pred.predicted_mean
105 y_truth = y['2017-01-01':]
106
107 # Compute the mean square error
108 mse = ((y_forecasted - y_truth) ** 2).mean()
109 print('The Mean Squared Error of our forecasts is {}'.format(round(mse,
        2)))
110 print('The Root Mean Squared Error of our forecasts is
        {}'.format(round(np.sqrt(mse), 2)))
111
112 pred_uc = results.get_forecast(steps=100)
113 pred_ci = pred_uc.conf_int()
114
115 ax = y.plot(label='observed', figsize=(14, 7))
116 pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
117 ax.fill_between(pred_ci.index,
118 pred_ci.iloc[:, 0],
119 pred_ci.iloc[:, 1], color='k', alpha=.25)
120 ax.set_xlabel('Date')
121 ax.set_ylabel('Furniture Sales')
122
123 plt.legend()
124 plt.show()
125
126 ##Data Preprocessing##
127 furniture = df.loc[df['Category'] == 'Furniture']
128 office = df.loc[df['Category'] == 'Office Supplies']
129 furniture.shape, office.shape
130
131 cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID',
        'Customer Name', 'Segment', 'Country', 'City', 'State', 'Postal
        Code', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product
        Name', 'Quantity', 'Discount', 'Profit']
132 furniture.drop(cols, axis=1, inplace=True)
```

```
133 office.drop(cols, axis=1, inplace=True)
134
135 furniture = furniture.sort_values('Order Date')
136 office = office.sort_values('Order Date')
137
138 furniture = furniture.groupby('Order Date')['Sales'].sum().reset_index()
139 office = office.groupby('Order Date')['Sales'].sum().reset_index()
140 office.head()
141 furniture.head()
142
143 furniture = furniture.set_index('Order Date')
144 office = office.set_index('Order Date')
145
146 y_furniture = furniture['Sales'].resample('MS').mean()
147 y_office = office['Sales'].resample('MS').mean()
148
149 furniture = pd.DataFrame({'Order Date':y_furniture.index,
        'Sales':y_furniture.values})
150 office = pd.DataFrame({'Order Date': y_office.index, 'Sales':
        y_office.values})
151
152 store = furniture.merge(office, how='inner', on='Order Date')
153 store.rename(columns={'Sales_x': 'furniture_sales', 'Sales_y':
        'office_sales'}, inplace=True)
154 store.head()
155
156 plt.figure(figsize=(20, 8))
157 plt.plot(store['Order Date'], store['furniture_sales'], 'b-', label =
        'furniture')
158 plt.plot(store['Order Date'], store['office_sales'], 'r-', label =
        'office supplies')
159 plt.xlabel('Date'); plt.ylabel('Sales'); plt.title('Sales of Furniture
        and Office Supplies')
160 plt.legend();
161
162 first_date = store.ix[np.min(list(np.where(store['office_sales'] >
        store['furniture_sales'])[0])), 'Order Date']
163
164 print("Office supplies first time produced higher sales than furniture is
        {}.".format(first_date.date()))
```

References

[1] Demuth, H.B., Beale, M.H., De Jess, O. and Hagan, M.T., 2014. *Neural network design*. Martin Hagan.

[2] Sue Becker and Yann LeCun. *Improving the convergence of back-propagation learning with second order methods.* Technical report, University of Toronto, Toronto, 1988.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer Verlag, New York, 2006.

[4] Geoffrey E. Hinton. Learning distributed representations of concepts, 1986.

[5] Yann LeCun. A theoretical framework for back-propagation. 1988.

[6] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience Publication, New York, 2001.

[7] Frank Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review, 65, 1958.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning representations by back-propagating errors.* Nature, 323, 1986.

[9] D. B. Parker, Learning-logic: Casting the cortex of human brain in silicon, Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA, 1985.

[10] Colin Molter, Utku Salihoglu and Hugues Bersini. (2005). Introduction of a Hebbian unsupervised learning algorithm to boost the encoding capacity of Hopfield networks. *IEEE International Joint Conference on Neural Networks, IJCNN '05. Proceedings, Volume 3*, pp. 1552-1557.

[11] Geoffrey E. Hinton (2003). The Ups and Downs of Hebb Synapses. *Canadian Psychology, Vol. 44, No. 1,* pp. 10-13.

[12] James Ting-Ho Lo. (2010). Unsupervised Hebbian learning by recurrent multilayer neural networks for temporal hierarchical pattern recognition. *44th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1-6.

[13] Bart Kosko. (1988). Biderectional associative memories. *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 18, No. 1*, pp. 49-60.

[14] Dunham, M. H., Meng, Y., Huang, J. (2004, November).Extensible markov model. *In Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on* (pp. 371-374).IEEE.

[15] Genevieve Gorrell and Brandyn Webb. (2005). Generalized Hebbian Algorithm for Incremental Latent Semantic Analysis. *INTERSPEECH, ISCA,* pp.1325-1328.