Meta-heuristic Based Optimization of Deep Neural Networks

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Puneet Kumar
(Roll No. 801632039)

Under the supervision of:
Dr. Shalini Batra
Associate Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

June 2018

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*Meta-heuristic based Optimization of Deep Neural Networks*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala is an authentic record of my own work carried out under the supervision of *Dr. Shalini Batra* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for the award of any other degree of this or any other University.

(Puneet Kumar)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Shalini Batra)
Assoc. Professor, CSED

ACKNOWLEDGEMENT

First of all, I would like to express my sincere gratitude towards my supervisor ***Dr. Shalini Batra*** for her valuable guidance and support throughout the thesis work. She has been helping me since the very beginning of my M.E. to explore my research interests in an organized manner. From providing the important machine learning data, teaching the technical writing skills to lightning fast responses to the mails, she has always been there to help me. The brain-storming sessions in her cabin are among the most worthwhile experiences I have had during my Master's studies.

I appreciate the Computer Science and Engineering Department of Thapar Institute of Engineering and Technology for providing the necessary research facilities. I am also thankful to Dr. Maninder Singh (HOD) and Dr. Ashutosh Mishra (PG Coordinator) and all the respected faculty members of the department for their teaching and guidance. I would also want to extend my obligation towards Nava Nalanda Central Library for providing the access to the prominent research journals.

I would like to thank my classmates for the research discussions and meaningful life experiences I got to have with them in last two years. Last but not the least I would like to thank my mother for supporting me throughout writing this thesis and in life in general. She has made me, me.

ABSTRACT

Deep Learning (DL) has emerged out as the most important sub-area of machine learning (ML). It deals with the design and application of deep neural networks (DNN) which are multi-layered adaptations of artificial neural networks (ANN). A machine learning model is typically a formula that learns its parameters from the data but there are some higher level parameters, known as the 'hyper-parameters' that cannot be learnt from the data. DNNs involve various hyper-parameters such as - number of layers and nodes, activation function, optimizer, regularization rate, loss function, *etc*. DNNs are architecturally complex and need to be trained on large data. There are enormous choices for their hyper-parameters and it is challenging to pick the best of them. Discovery of the suitable hyper-parameters is especially important for the DNNs implemented to recognize complex multimedia data that is being generated by various devices at a very high speed.

In this research work, traditional and meta-heuristic optimization approaches have been analyzed for DNN optimization. Convolutional and recurrent variants of DNNs have been implemented to recognize image objects and predict streaming data of the Indian stock market. Four experimental cases have been designed and Genetic Algorithm (GA) based approach is used to find the optimal hyper-parameter combination for DNN design. The proposed optimization process includes two phases. The first phase quickly returns the optimal set of hyper-parameters to design a DNN. It is applied to both Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Compared to the traditional grid search based methods it has provided an average speed-up of 8 times for CNN and 6.5 times for RNN. The second phase has been applied only to RNNs deployed to process streaming data. It finds an appropriate subset of the training-data for near optimal prediction performance. The optimized RNN version has been experimentally observed to be 74.34% faster than single layered Long Short Term Memory (LSTM) architecture and 75.86% faster than the deep LSTM model. The decline in accuracy is 7.17% and 10.78% respectively.

Table of Contents

CERTIFICATE ..... **I**

ACKNOWLEDGEMENT ..... **I**

ABSTRACT ..... **II**

LIST OF FIGURES ..... **VI**

LIST OF TABLES ..... **VII**

LIST OF EQUATIONS ..... **VIII**

LIST OF ABBREVIATIONS ..... **VIII**

1. INTRODUCTION **1**

1.1 DEEP NEURAL NETWORK (DNN) 1

1.1.1 Convolution Neural Network (CNN) 1

1.1.2 Recurrent Neural Network (RNN) 2

1.1.2.1 Long Short Term Memory (LSTM) 3

1.1.2.2 GRU (Gated Recurrent Network) 4

1.2 HYPER-PARAMETERS 5

1.2.1 Number of Hidden Layers 5

1.2.2 Number of Neurons per Layer 5

1.2.3 Activation Function 6

1.2.3.1 Linear Function 6

1.2.3.2 Sigmoid Function 6

1.2.3.3 tanh Function 7

1.2.3.4 ReLU Function 7

1.2.3.5 ELU Function 7

1.2.3.6 SELU Function 8

1.2.4 Network Optimizer 8

1.2.4.1 SGD 8

1.2.4.2 AdaGrad 9

1.2.4.3 AdaDelta 9

1.2.4.4 Adam 9

1.2.4.5 Nadam 9

1.2.4.6 AdaMax 9

1.2.5 Regularizer 10
1.2.5.1 L1 Regularization 10
1.2.5.2 L2 Regularization 10
1.2.6 Loss Function 10
1.3 META-HEURISTIC TECHNIQUES 11
1.3.1 Genetic Algorithm (GA) 12
1.4 META-HEURISTIC FORMULATION OF DNN OPTIMIZATION 13
1.5 THESIS ORGANIZATION 14
2. LITERATURE SURVEY 15
2.1 CHALLENGES OF DNNS 15
2.1.1 Vanishing Gradient Issue 15
2.1.2 Difficulty in Model Training 15
2.1.3 Hyper-parameter Tuning 15
2.2 DNN OPTIMIZATION APPROACHES 16
2.2.1 Manual Search 16
2.2.2 Grid Search 16
2.2.3 Random Search 16
2.2.4 Bayesian Search 17
2.2.5 Gradient-based Search 17
2.2.6 Meta-heuristic Search 17
2.3 SURVEY OF NEURAL NETWORK OPTIMIZATION ATTEMPTS 17
2.4 SCOPE OF USING META-HEURISTICS FOR DNN OPTIMIZATION 19
3. PROBLEM STATEMETNT 21
3.1 INTRODUCTION 21
3.2 PROBLEM FORMULATION 21
3.3 OBJECTIVES 22
4. PROPOSED METHODOLOGY 23
5. IMPLEMENTATION AND RESULTS 25
5.1 IMPLEMENTATION PLATFORMS 25
5.2 ALGORITHM 26
5.2.1 Phase 1: Hyper-parameter Tuning 26
5.2.2 Phase 2: Look-back Window Selection 27

5.3 EXPERIMENT USE-CASES 29
5.3.1 Use-case 1: MNIST digit recognition using CNN 29
5.3.2 Use-case 2: CIFAR image recognition using CNN 30
5.3.3 Use-case 3: Sample streaming data prediction using RNN 30
5.3.4 Use-case 4: Stock market data prediction using RNN 32
5.4 RESULTS 33
5.4.1 USE-CASE 1: MNIST DIGITS RECOGNITION USING CNN 34
5.4.1.1 Hyper-parameter Discovery 34
5.4.1.2 Performance Evaluation 35
5.4.2 USE-CASE 2: CIFAR IMAGES RECOGNITION USING CNN 37
5.4.2.1 Hyper-parameter Discovery 37
5.4.2.2 Performance Evaluation 38
5.4.3 USE-CASE 3: SAMPLE STREAMING DATA PREDICTION USING RNN 40
5.4.3.1 Hyper-parameter Discovery 41
5.4.3.2 Performance Evaluation 41
5.4.3.3 Prediction Plots 42
5.4.4 USE-CASE 4: STOCK MARKET DATA PREDICTION USING RNN 45
5.4.4.1 Hyper-parameter Discovery 45
5.4.4.2 Performance Evaluation 46
5.4.4.3 Prediction Plots 47
6. CONCLUSION AND FUTURE WORK 49
6.1 THESIS CONTRIBUTIONS 49
6.2 CONCLUSIONS 49
6.3 FUTURE SCOPE 50
REFERENCES 52
LIST OF PUBLICATIONS 58
VIDEO PRESENTATION 59
ORIGINALALITY REPORT 60

List of Figures

Figure 1.1: Schematic Architecture of CNN [29] 2
Figure 1.2: Schematic Architecture of RNN [31] 3
Figure 1.3: Schematic Architecture of LSTM [31] 4
Figure 1.4: Linear function 6
Figure 1.5: Sigmoid function 6
Figure 1.6: tanh function 7
Figure 1.7: ReLU function 7
Figure 1.8: ELU function 8
Figure 1.9: SELU function 8
Figure 1.10: Classifications of meta-heuristic techniques [22] 12
Figure 1.11: Overview of the Genetic Algorithm [52] 13
Figure 1.12: Spectrum of meta-heuristic design of DNN [23] 14
Figure 5.1: Sample images of MNIST dataset 29
Figure 5.2: Sample images of CIFAR dataset 30
Figure 5.3: a - rectangular, b - sinusoidal and c - sinc data pulse [70] 31
Figure 5.4: a – BSE Sensex, b – BSE SmallCap and c – BSE Bank stock data [71] 32
Figure 5.5: Use-case 1 accuracy plot 36
Figure 5.6: Use-case 1 loss plot 36
Figure 5.7: Use-case 1 confusion matrix 37
Figure 5.8: Use-case 2 accuracy plot 39
Figure 5.9: Use-case 2 loss plot 39
Figure 5.10: Use-case 2 confusion matrix 40
Figure 5.11: Prediction plots for rectangular pulse data 43
Figure 5.12: Prediction plots for sinusoidal pulse data 43
Figure 5.13: Prediction plots for sinc pulse data 44
Figure 5.14: Prediction plots for BSE Sensex30 data 47
Figure 5.15: Prediction plots for BSE SmallCap data 48
Figure 5.16: Prediction plots for BSE Bank data 48

List of Tables

Table 1: Hyper-parameter choices [42] 5
Table 2: Summary of the literature survey 18
Table 3: Summary of the use-cases 33
Table 4: Experiment summary for use-case 1 34
Table 5: Optimal hyper-parameters for use-case 1 35
Table 6: Results for MNIST digit recognition 35
Table 7: Experiment summary for use-case 2 38
Table 8: Optimal hyper-parameters for use-case 2 38
Table 9: Results for CIFAR digit recognition 39
Table 10: Experiment summary for use-case 3 41
Table 11: Optimal hyper-parameters for use-case 3 41
Table 12: Results for rectangular pulse data prediction 42
Table 13: Results for sinusoidal pulse data prediction 42
Table 14: Results for sinc pulse data prediction 42
Table 15: Experiment summary for use-case 4 45
Table 16: Optimal hyper-parameters for use-case 4 45
Table 17: Results for BSE Sensex30 data prediction 46
Table 18: Results for BSE SmallCap data prediction 46
Table 19: Results for BSE Bank data prediction 46

List of Equations

Equation 1: Cost Function 22
Equation 2: Grid Search.....23
Equation 3: Optimized Search 24

List of Abbreviations

Abbreviation	Explanation
DL	**D**eep **L**earning
ML	**M**achine **L**earning
DNN	**D**eep **N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
FFNN	**F**eed **F**orward **N**eural **N**etwork
GD	**G**radient **D**escent
BP	**B**ack **P**ropagation
BPTT	**B**ack **P**ropagation **T**hrough **T**ime
CNN	**C**onvolution **N**eural **N**etwork
RNN	**R**ecurrent **N**eural Network
LSTM	**L**ong **S**hort **T**erm **M**emory
GRU	**G**ated **R**ecurrent **U**nits
SVM	**S**upport **V**ector **M**achine
GA	**G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimization
ABC	**A**nt **B**ee **C**olony Optimization
GPU	**G**raphical **P**rocessing **U**nit
CPU	**C**entral **P**rocessing **U**nit
ReLU	**R**ectified **L**inear **U**nits
ELU	**E**xponential **L**inear **U**nits
SELU	**S**caled **E**xponential **L**inear **U**nits
SGD	**S**tochastic **G**radient **D**escent
Adagrad	**Ada**ptive **Grad**ient
Adam	**Ada**ptive **M**omentum
NAG	**N**esterov **A**ccelerated **G**radient
RMSE	**R**oot **M**ean **S**quare **E**rror
LASSO	**L**east **A**bsolute **S**hrinkage and **S**election **O**perator
MNIST	**M**odified **N**ational **I**nst. of **S**tandards & **T**echnology
CIFAR	**C**anadian **I**nstitute **F**or **A**dvanced **R**esearch
BSE	**B**ombay **S**tock **E**xchange
NSE	**N**ational **S**tock **E**xchange

1. INTRODUCTION

1.1 Deep Neural Network (DNN)

Developed by G. Hinton *et al.* [1], Deep Learning (DL) is a collective concept of a series of algorithms that attempt to model high-level abstractions in data by using multiple processing layers with complex structures [2]. Fundamental DL architectures are called Deep Neural Networks (DNNs). They are special types of artificial neural networks (ANNs) that have one or more hidden layers between the input layer and the output layer [3]. Advances in the design of DNN combined with effective approaches to train them have opened the opportunities to use the DNNs for a wide range of applications. With data being collected from a wide variety of devices and sensors, it has rekindled the interest in using DL as a method to uncover hidden structures in these ever-growing datasets. One of the greatest challenges with these datasets is to recognize their complex and high-dimensional data. DL approaches have achieved significant importance in this direction, as a way of constructing hierarchical representations from the unlabeled data [43]. Presence of multiple layers enables a DNN to learn the nonlinear functions, and automatically perceive the complex features of huge datasets more efficiently [44].

1.1.1 Convolution Neural Network (CNN)

A convolutional neural network is a type of feed-forward neural network (FFNN) in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. They are designed to process the grid data such as – images and video frames. Traditional FFNNs with all fully connected processing layers find it too complex to process the large amount of pixels present in image grids. CNN attempts to improve on this issue by incorporating special properties to deep FFNNs such as – Convolution layers, Pooling or Subsampling layers and Normalization layer with ReLU activation to extract the high-level features to be fed to the last layer which is fully connected [4]. A typical CNN network consists of Convolution, Subsampling and ReLU layers stacked over each other. Fig. 1.1 represents a simple CNN architecture with 2 convolution layers.

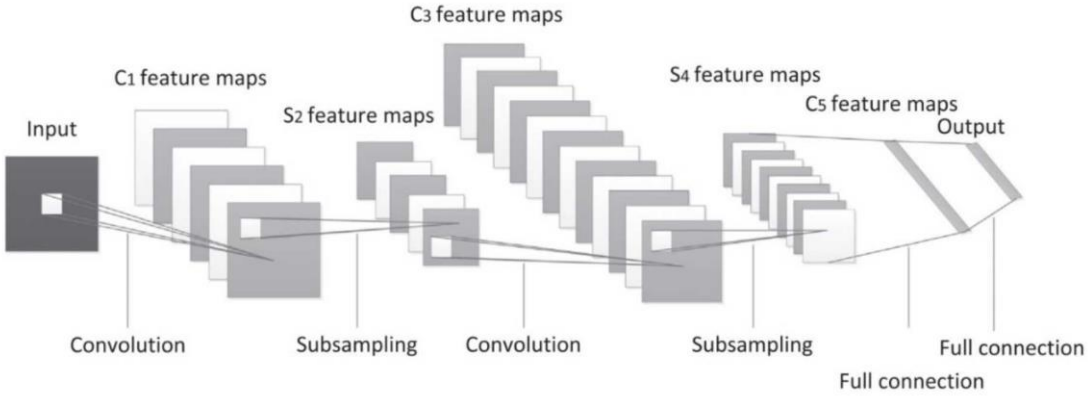


Figure 1.1: Schematic Architecture of CNN [29]

1.1.2 Recurrent Neural Network (RNN)

A recurrent network is created by applying the same set of weights recursively to produce a structured prediction over variable-size input structures. RNNs have recurrent connections between the time-steps to memorize what has been calculated so far in the network. RNNs have been especially successful for the sequential input and output instances, for example - in natural language processing, image captioning, sentence generation, video frame classification, *etc.* Simple FFNNs are designed to process fixed sized input and outputs and they cannot model the memory element to capture time-series data or sequences. RNNs are specially designed to overcome these shortcomings of FFNNs. They can process variable sized data and they are capable of remembering the old state information recursively. However, this makes the implementation and training of RNNs very complex [30]. RNNs work on this recursive formula:

$$S_t = F_w(S_{t-1}, X_t)$$
$$S_t = tanh(W_s S_{t-1} + W_X X_t)$$
$$Y_t = W_y S_t$$

Where F_w, X_t, W, S_t, S_{t-1} denote the recursive function, input at time step t, weight vector, the state at time step t and the state at time t-1. Same sets of weights are used through-out the network, i.e. W_x, W_y and W_s remain same for all the time-steps. RNNs learn through back propagation through time (BPTT). Fig. 1.2 shows the modeling process of a simple RNN network unrolled for 2 time-steps.

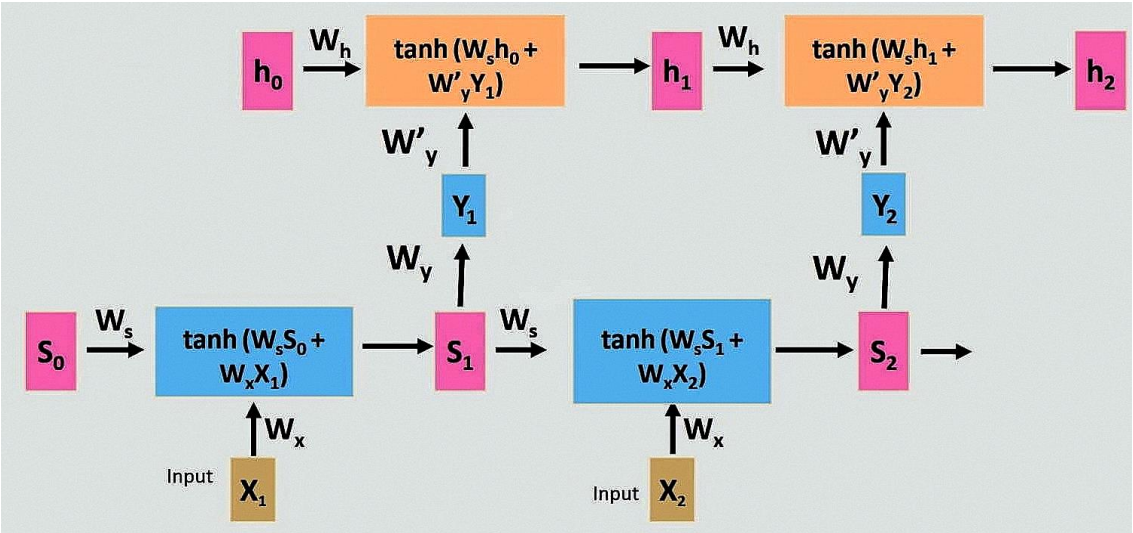


Figure 1.2: Schematic Architecture of RNN [31]

Vanishing Gradient problem [24] is a major hurdle in the application of simple RNNs as it increases the training cost for the network. LSTM and GRU are two of the most commonly known solutions to control absorbing or forgetting the information in recurrent networks [25][26].

1.1.2.1 Long Short Term Memory (LSTM)

A RNN with LSTM units, commonly known as LSTM network is composed of LSTM units as the building blocks of the network [25]. LSTM is the result of the interactions that are added to RNN in terms of memory gates, to resolve the vanishing gradient problem. They are made up of three gates i.e. input, output and forget gate and a memory cell state. An LSTM is trained with BPTT and it tends to store the state values using the gates, hence the gradients do not vanish faster. Following is the mathematical description of the gates and state element.

$$I_t = \sigma(W_i S_{t-1} + W_i X_t + b_i)$$
$$O_t = \sigma(W_o S_{t-1} + W_o X_t + b_o)$$
$$F_t = \sigma(W_f S_{t-1} + W_f X_t + b_f)$$
$$C_t = (I_t * tanh(W_c S_{t-1} + W_c X_t) + F_t * C_{t-1})$$
$$S_t = O_t * tanh(C_t)$$

Where I_t, O_t, F_t, C_t, S_t, X_t, W, b, σ *and* $*$ denote input gate, output gate, forget gate, cell state at time-step t, network state at time-step t, inputs, weight matrix, bias vector, activation function and element-wise multiplication operator respectively. Fig. 1.3 shows a simple visualization of the LSTM.

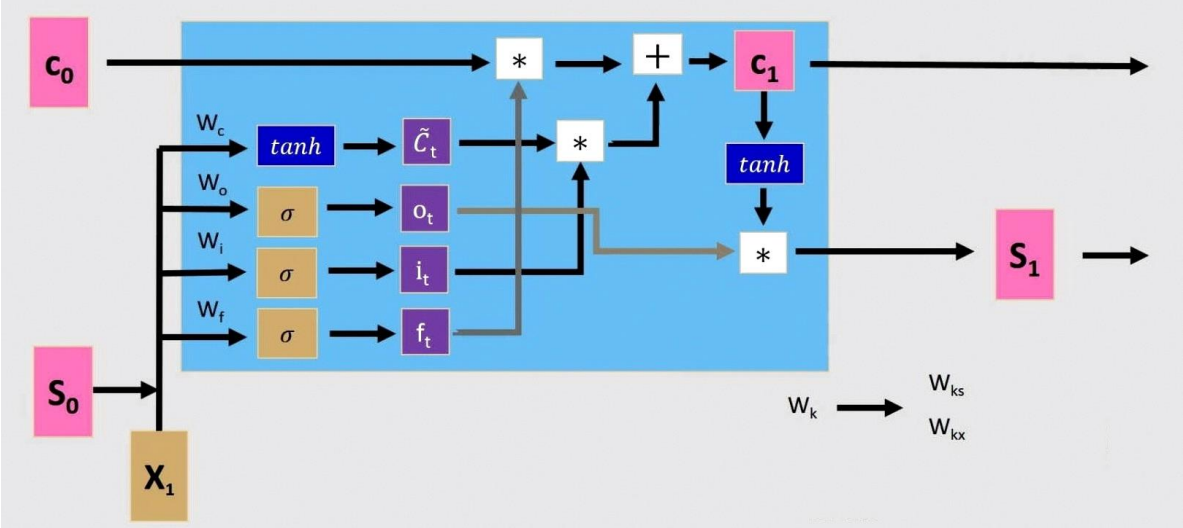


Figure 1.3: Schematic Architecture of LSTM [31]

C_O is the previous cell state; X_1 and S_O are the inputs. W_s and W_x are separate weight vectors for S_O and X_1. They are represented together in the Fig. 1.3 to make the visualization easier. Input X_1 and previous state S_O are passed through activation function σ to calculate the gates I_t, O_t and F_t. Then the intermediate cell state $\bar{C}_t$ is found by passing these two values through *tanh* function. Element wise multiplication is performed for previous state C_O with F_t, intermediate state $\bar{C}_t$ with F_t; and the quantities obtained are added to get the new cell state C_t which is used to find the network state S_1.

1.1.2.2 GRU (Gated Recurrent Network)

GRU based RNN is also capable of modeling the information flow and memory states, but it is simpler than LSTM. It is made up of 2 gates - update gate and reset gate [26]. Compared to LSTM, forget and input gates are combined into a single update gate, and the cell state and hidden state is merged together. So a GRU network can be considered as a simpler form of the LSTM network Mathematical formulation of a GRU based RNN is represented below.

$$z_t = \sigma(W_z S_{t-1} + W_z X_t + b_z)$$
$$r_t = \sigma(W_r S_{t-1} + W_r X_t + b_r)$$
$$S_t = (1 - z_t) * S_{t-1} + z_t * tanh(W\ (r_t * S_{t-1}) + W\ * X_t)$$

Where z_t, r_t, S_t, X_t, W, b, σ *and* $*$ denote update gate, reset gate, forget gate, network state at time-step t, inputs, weight matrix, bias vector, activation function and element-wise multiplication operator respectively.

1.2 Hyper-parameters

A machine learning model is a formula that learns a number of parameters from the data such as – connection weights of a Neural Network, slope and intercept for Linear Regression, support vectors in SVM, *etc.* However, there are some higher level parameters known as 'hyper-parameters' whose values cannot be learnt from the data [5][6]. Some examples of the hyper-parameters are - the number of trees in a Random Forest, the number of hidden layers in a Neural Network, the learning rate for Logistic Regression, kernel in SVM, *etc.* Some of the most important hyper-parameters for the DNNs and their commonly used values are listed in the Table 1.

Table 1: Hyper-parameter choices [42]

Hyper-parameter	**Hyper-parameter values**
No. of hidden layers	1, 2, 3, 4, 5, 6
No. of neurons per layer	64, 128, 256, 512, 768, 1024
Activation Function	linear, sigmoid, tanh, relu, elu, selu
Network Optimizer	sgd, adagrad, adadelta, adam, nadam, adamax
Regularizer	l1(0.01), l2(0.01)
Loss Function	mean squared error, mean absolute error, categorical crossentropy, squared hinge

1.2.1 Number of Hidden Layers

Number of hidden processing layers of a network is the most important hyper-parameter. This is the factor that qualifies a network to be deep or shallow. A deep network is the one with one or more hidden layers. Network complexity as well as the prediction accuracy increases proportionate to the number of layers. It is important to find the right number of layers to get sufficiently good accuracy in reasonable training time. Learning based and automated search techniques such as grid search, meta-heuristic search could be helpful to initialize this hyper-parameter [32].

1.2.2 Number of Neurons per Layer

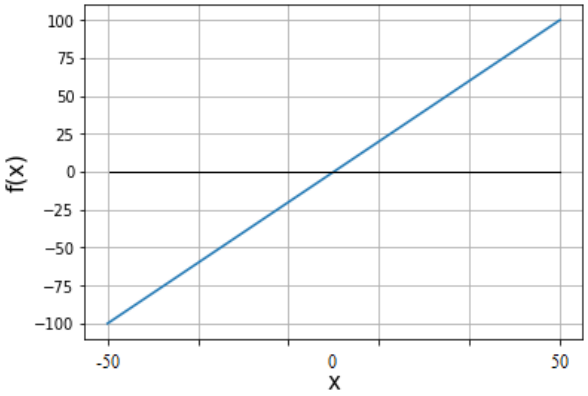
To minimize the error and make the model generalize well, it is important to select the optimal number of nodes for the network's layers [33]. Selection of the number of nodes has mostly been carried out by manual simulations and trials. Automated meta-heuristic based search could be useful for their optimization as well.

1.2.3 Activation Function

An activation function defines the output of a particular node of the network for a given input or set of inputs [34]. Details of various activation functions considered for the current research are as follows.

1.2.3.1 Linear Function

Linear functions are the simplest forms of the network activators. They map the outputs proportional to the inputs. An example linear function is shown in Fig. 1.4. Neural networks are supposed to be universal function approximators with the capability of learning non-linear complex functions as well. This cannot be achieved with linear activation functions.



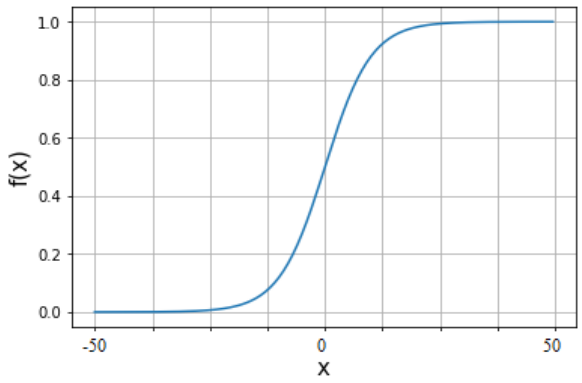
$$f(x) = a.x$$

Where:
x: input,
f(x): output,
a: constant.

Figure 1.4: Linear function

1.2.3.2 Sigmoid Function

Represented in Fig. 1.5, sigmoid function or the logistic function is the most commonly used neural network activation function. It is non-linear, not zero centered and suffers from vanishing gradient problem of DNNs [24]



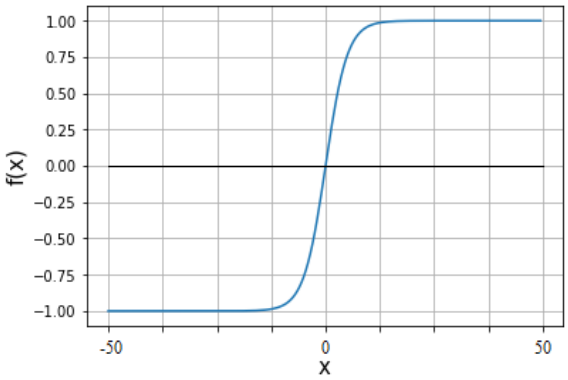
$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

Where:
x: input,
f(x): output,
e: 2.71828,
β: constant.

Figure 1.5: Sigmoid function

1.2.3.3 tanh Function

The hyperbolic tangent function or simply tanh function can be understood as a mathematically shifted version of the logistic function. As shown in Fig. 1.6, its output is zero centered; however, it also suffers from the vanishing gradient issue.



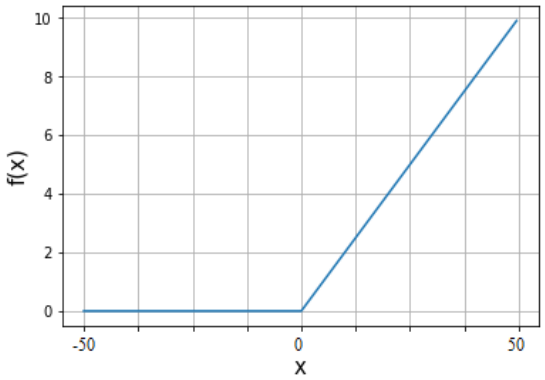
$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Where:
x: input,
f(x): output,
e: 2.71828.

Figure 1.6: tanh function

1.2.3.4 ReLU (Rectified Linear Units) Function

It is a fast learning function with no complex operations such as computing the gradients. It does not suffer from the vanishing gradient issue. It is used to learn the non-linear functions and is mostly used for the hidden layers.



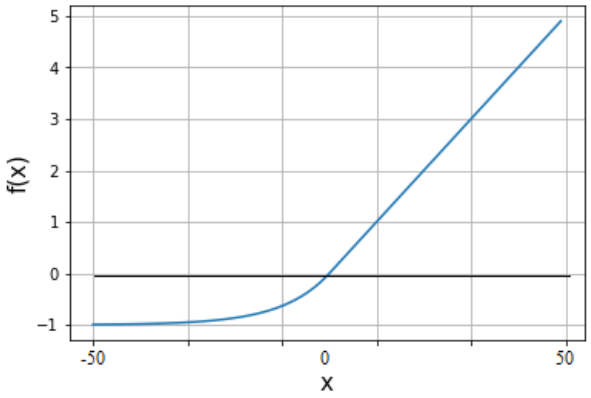
$$f(x) = \max(x, 0)$$

Where:
x: input,
f(x): output.

Figure 1.7: ReLU function

1.2.3.5 ELU (Exponential Linear Units) Function

ELU is an improved version of ReLU [35]. As shown in Fig. 1.8, it outputs negative values as well, which allows shifting the mean towards zero. It reduces the bias shift effect and speeds up the learning.



$$f(x) = \begin{matrix} e^{2x} - 1 & for\ (x < 0) \\ x & for\ (x \geq 0) \end{matrix}$$

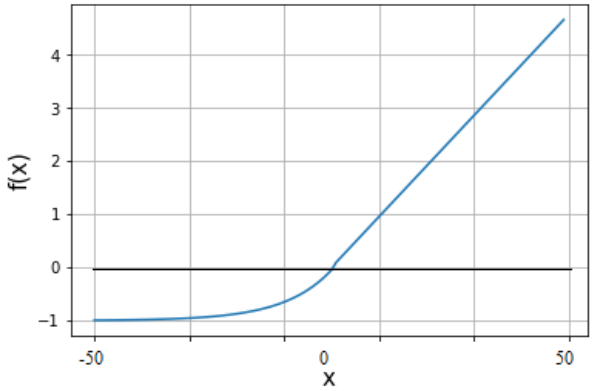
Where:

x: input,
f(x): output,
e: 2.71828.

Figure 1.8: ELU function

1.2.3.6 SELU (Scaled Exponential Linear Units) Function

SeLU implies scaling to the function output, which induces self-normalizing behavior. It aims to keep the mean and variance of the network activations to 0 and 1 respectively [36]. An example SELU function is plotted in Fig. 1.9.



$$f(x) = \begin{matrix} \lambda(e^{2x} - 1) & for\ (x < 0) \\ \lambda x & for\ (x \geq 0) \end{matrix}$$

Where:

x: input,
f(x): output,
e: 2.71828,
λ: scaling constant.

Figure 1.9: SELU function

1.2.4 Network Optimizer

An optimizer is an iterative strategy to optimize the objective function of the network. This is what actually decides how the network will learn. Optimizers examined for this work are detailed below.

1.2.4.1 SGD (Stochastic Gradient Descent)

SGD computes the gradients of the loss function with respect to the parameter for each training example. The parameters are then updated for a given number of epochs. The frequent parameter updates with respect to each data point cause the

objective function to fluctuate more intensely. It helps to discover new and possibly better local optima, but it complicates the convergence to the exact optimum [37].

1.2.4.2 AdaGrad (Adaptive Gradient)

It performs gradient descent based optimization allowing dynamic learning to the learning rate. It uses a different learning rate for every parameter at a given time step based on the past gradients that were computed for that parameter. Larger updates are made for the infrequent parameters and small updates are made for the frequent ones [37].

1.2.4.3 AdaDelta

It is an improved version of AdaGrad optimizer which prevents learning rate decay. It defines the sum of gradients as a decaying average of all past squared gradients. Running average at a time-step depends only on the previous average and current gradient.

1.2.4.4 Adam (Adaptive Momentum)

Apart from calculating the learning rate for each parameter, adam calculates and stores the momentum for each parameter separately. Thus it allows adapting the momentum in gradient calculations. The momentum increases for the dimensions whose gradients keep pointing in the same direction, and it decreases otherwise. It helps in faster convergence and reduced oscillations of the gradients.

1.2.4.5 Nadam

Nadam is the combination of Adam and NAG (Nesterov Accelerated Gradient) optimizations [38]. It starts by adaptively calculating the momentum of each parameter. Then, unlike the basic Momentum optimizer, it updates the gradients before calculating the momentum.

1.2.4.6 AdaMax

AdaMax is another variant of Adam optimizer, which is much more stable in its updates because of the use of infinite order norm [39]. The norm of a vector denotes its length. An infinite order norm can process the vectors of any length, hence

providing more flexibility to durably accommodate the updates of multiple parameters.

1.2.5 Regularizer

Regularizers are the methods that apply penalties on layer parameters during the learning process. These penalties use the loss functions being optimized by the network. There are following two methods to carry out the regularization.

1.2.5.1 L1 Regularization

L1 Regularization, also known as the LASSO (Least Absolute Shrinkage and Selection Operator) Regression uses the absolute value of the magnitude of the coefficient as a penalty which is added to the loss function [40].

$$\text{cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

Where:

N: Number of training examples, i.e. inputs,

M: Number of weights,

x^i: i^{th} input,

W_j: weight vector for j^{th} layer

λ: Regularization Rate

1.2.5.2 L2 Regularization

L2 Regularization, also known as the Ridge Regression uses the squared magnitude of the coefficient as the penalty which is added to the loss function [41].

$$cost = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} {W_j}^2$$

1.2.6 Loss Function

A loss function or the cost function maps the parameter values of a network on a real number scale and computes the errors made by the learning model by finding

the numeric differences between the expected and predicted values. Commonly used loss functions for the DNNs are as follows:

- mean squared error function,
- mean absolute error function,
- mean absolute percentage error function,
- mean squared logarithmic error function,
- categorical cross-entropy function,
- hinge function,
- squared hinge function,
- log-cosh function, *etc*.

1.3 Meta-heuristic techniques

A meta-heuristic is a problem independent search process involving exploration and exploitation of the search space. It assists a subordinate heuristic to quickly find near optimal solutions. Meta-heuristic algorithms are frequently used as approximation techniques to find a sufficiently good solution to complex optimization problems. It is a collective concept of a series of algorithms including evolutionary algorithm such as Genetic Algorithm (GA) [48], naturally inspired algorithm such as Particle Swarm Optimization (PSO) [49], trajectory algorithm such as Tabu search [50], and so on. They don't guarantee the optimality, but they are well-known to provide a near-optimal approximated solution very quickly [46]. When it comes to optimizing multiple DNN parameters with number of choices for each parameter, meta-heuristics emerge out as an intuitive potential choice as alternate methods to train DNNs. Abounding number of meta-heuristic approaches is available in the literature [45][46]. Fig. 1.10 shows the classification of the well-known meta-heuristics. As per No Free Lunch theorem, single meta-heuristic method will not work for all optimization problems [47]. Different approaches may be suitable for different DL architectures, for various application domains.

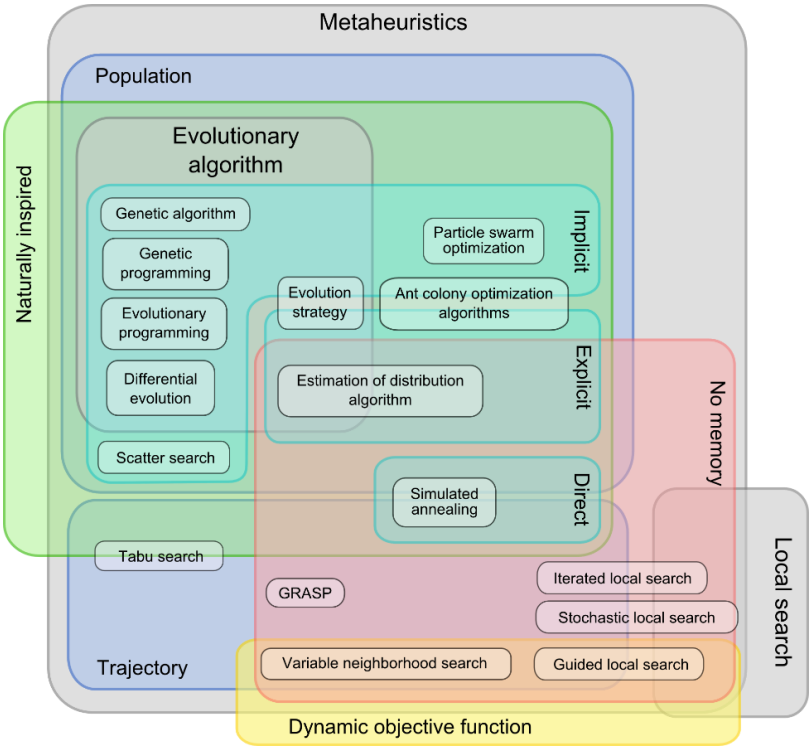


Figure 1.10: Classifications of meta-heuristic techniques [22]

1.3.1 Genetic Algorithm (GA)

GA is an optimization and search approach inspired by the natural selection process [51]. It involves the process of evolving the species over the generations. GA uses following three basic operations to generate next generation for the given population:

- **Selection**: This operation selects the individuals that would contribute to the population at next generation. They are called the ‘parents’.
- **Cross-over**: The parents are combined to form the children for the next generation by this operation.
- **Mutation**: This operation aims to diversify the population by applying random changes in the children evolved by the cross-over operation.

Fig. 1.11 shows general working of GA. The initial population is randomly generated and evaluated against the fitness parameters. Then Cross-over and Mutation is performed and next generation is evolved. This whole process is repeated for the defined number of generations and the final solutions with highest fitness score are computed.

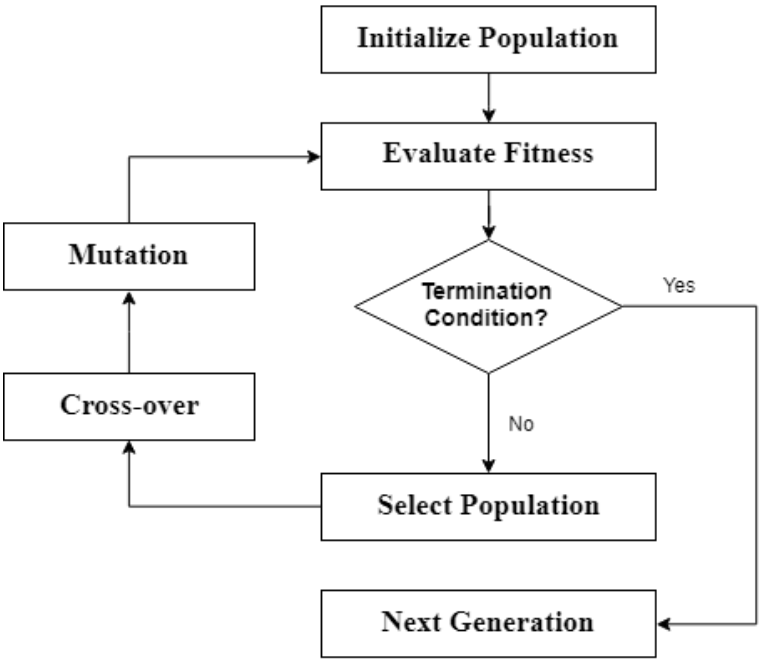


Figure 1.11: Overview of the Genetic Algorithm [52]

1.4 Meta-heuristic Formulation of DNN Optimization

The conventional strategy that has mostly been used to train an FFNN is Gradient Descent (GD) algorithm. GD uses back-propagation (BP) to modify the weights of the networks, which is an iterative approach, and is known to be very time-consuming [24]. DNNs are based on FFNN structures and they may have millions of parameters while being trained for a large dataset. The limitations of GD in terms of the requirement of an extensive amount of time to train the DNNs, have triggered the need to consider alternative methods of learning the right weights the network that could fetch us near-optimal accuracy with lesser computational efforts.

Various new ways are gradually emerging to make the DNN more efficient, for example-pre-training the network, transfer learning, carrying on the training process layer by layer, *etc.* [7][8] but there isn't any known method to control one or more components of the network in combination. Meta-heuristics have also started finding their place in parameter tuning [9]. Though they have mostly been explored to tune the connection weights, their usability doesn't solely depend on finding the correct weights, but on finding the correct values of other parameters and hyper-parameters such as - learning rate, architecture, number of nodes and layers, activation function, *etc.* as well. They have the potential to allow optimizing one or

more components in combination. This is the major motivation to use them to optimize DNNs as a tool to have more control over the optimization process.

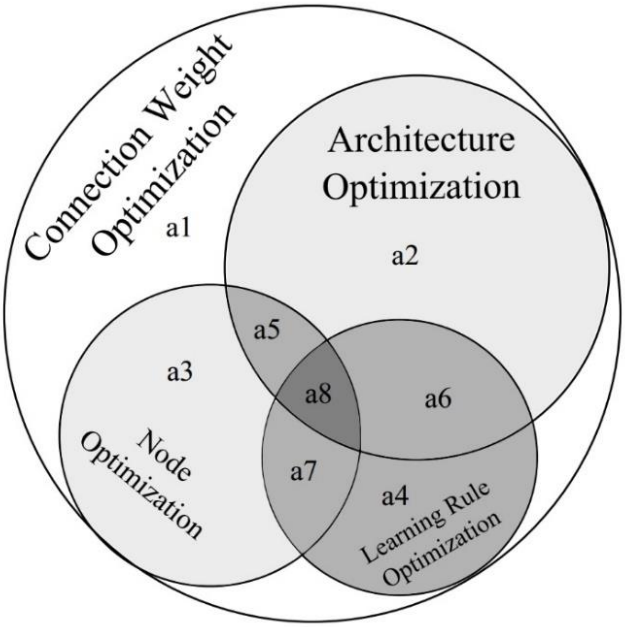


Figure 1.12: Spectrum of meta-heuristic design of DNN [23]

Fig 1.12 discusses various avenues for DNN optimization. Here regions *a1* to *a4* denote the optimization of two components while keeping other components' values, fixed; *a5* to *a7* denote the optimization considering three components; and, *a8* represents the optimization process considering all four components.

1.5 Thesis Organization

The key concepts of Deep Neural Networks and Meta-heuristic optimization techniques have been introduced in chapter 1. Chapter 2 verbalizes the challenges and approaches for DNN optimization and research attempts made in this direction. The problem statement of optimizing DNN architectures in terms of hyper-parameter tuning has been formulated in chapter 3. A GA based solutions to find the hyper-parameter combination most suited for CNNs and RNNs for different use-cases of object detection and streaming data prediction have been proposed in chapter 4 and implemented in chapter 5. Chapter 5 also discusses the results of the implementation and chapter 6 concludes this report and lists the future research scope.

2. LITERATURE SURVEY

This chapter starts by briefing the general challenges faced by DNNs and the methods to optimize their architecture in order to resolve the same. Then a summary of the research attempts on DNN optimization has been presented and research gaps have been identified.

2.1 Challenges of DNNs

DNNs are associated with the following challenges:

2.1.1 Vanishing Gradient Issue

For deep networks, even the small differences in the error gradients get over-amplified. Depending on their values being less than or more than one, the gradients either vanish or explode as they pass through the layers and get multiplied over and again. This is known as the vanishing gradient problem [24].

2.1.2 Difficulty in Model Training

Model training is among the most difficult of all the optimization problems involved with DNNs [11]. For the CNNs deployed to recognize images and videos, even a single instance of neural network training problem could take days or even weeks to train [12]. Training of RNNs is also difficult because of their complex architecture and vanishing gradient problem [13].

2.1.3 Hyper-parameter Tuning

Hyper-parameters are the values of a DNN model that are set before training on any data. Setting the right parameters is essential to unleash the maximum potential of a DNN. The hyper-parameters cannot be learnt from the data; hence it is effortful to select their appropriate values. Some of the challenges involved with hyper-parameter tuning are – selecting the appropriate number of parameters, using suitable evaluation metric, avoiding initializing the parameters with biased or default values, *etc.* [14].

2.2 DNN Optimization Approaches

In simple words, optimization means ‘to make the best of something’. DNN optimization deals with choosing the best parameters for the neural network’s architecture. There are various parameters and hyper-parameters involved in designing a deep network, for example – number of layers, number of neurons per layer, weights of the layers, activation function, optimization method, number of training epochs, evaluation method, *etc.* The best choice for the parameters of a DNN can be learnt by observing more training examples but hyper-parameters cannot be learnt from the data. Following are the general search methods to select the suitable values for the hyper-parameters.

2.2.1 Manual Search

In manual search, the hyper-parameters are adjusted by a human as per his or her knowledge about the effect of the parameters on the model’s performance. With limited computational power, it was not possible to have a lot of trials to train the deep networks and evaluate their performance against various parameter combinations. Hence, the selection of the hyper-parameters has typically been a human job [6]. Manual selection of the hyper-parameters depends on the modeler's knowledge. It is prone to biases and mistakes, suffers from ordering and scheduling problems [14].

2.2.2 Grid Search

Grid-search is a brute force approach that specifies the possible values of the hyper-parameters upfront, and each combination is iteratively evaluated. It does return the absolute best parameter combination but suffers from the curse of dimensionality. As the number of parameters increases, the computational time to evaluate those increases exponentially [58].

2.2.3 Random Search

It is a black-box search approach that aims to minimize the cost function for the network by picking and evaluating the hyper-parameter combinations randomly [17]. The random search approach is known to suffer from cost function fluctuations and slow convergence rates [18].

2.2.4 Bayesian Search

Bayesian optimization maps the hyper-parameter values to an objective function. Then it iteratively evaluates the function on a validation set and updates the model accordingly [58]. The bayesian search is slow and does not always perform well for the dynamic problems such as real time data prediction [19].

2.2.5 Gradient-based Search

This method uses gradient descent method to optimize the hyper-parameters after finding network's gradients with respect to them [60]. Gradient-based optimization sometimes suffers from local minima trapping issue, *i.e.* it might get stuck in a local minimum and might never converge to the global minima [20].

2.2.6 Meta-heuristic Search

Meta-heuristics are multi-purpose search techniques that search the hyper-parameter space independent of the application. They don't guarantee the best solution but they are known to fetch near-best solution very quickly. A category of meta-heuristics, the evolutionary algorithm has been applied for the initialization of neural network's weights [28]. They have shown good potential in the context of Deep Learning [21] but they have not been explored much for the optimization of the deep networks [54]. A class of meta-heuristic search techniques, GA is found to be practical in learning hierarchal rules in large datasets [53].

2.3 Survey of Neural Network Optimization Attempts

In recent years, the researchers have made a number of attempts to optimize the training process of ANN using various search algorithms. Bergstra *et al.* [57] have implemented various search techniques to figure out the relevant values of a particular hyper-parameter given the values of other parameters. Grid search has been used as an automated way to scan all the combinations of the hyper-parameters. Stewart *et al.* [59] have used Bayesian Optimization to automate neural network training. Maclaurin *et al.* [59] have worked to fix the issue of unavailable gradients while evaluating the parameter values of a model. During the training process, the available gradients values have been modified in the backward direction to find the value of the unavailable gradients. After evaluating all the gradients with respect to the hyper-

parameters, the amount and direction of the change in their values have been determined and used to find their optimal values. Bergstra *et al.* [57] have implemented random search to configure the architecture of a DBN (Deep Belief Network). Empirical comparison over a 32-dimensional search space demonstrated the superiority of the random search over manual and grid search.

In the context of using meta-heuristic based approaches, Leung *et al.* [62] worked on improving the optimization parameters of the network. He presented a method to tune the structure and parameters of a neural network using an improved GA. Improved GA was shown to perform better than the standard GA based on some benchmark test functions. Bullinaria *et al.* [63] used Artificial Bee Colony (ABC) optimization method to optimize the connection weights of an FFNN classifier. Gudise and Venayagamoorthy *et al.* [61] compared feed forward with PSO and feed forward with BP, and the results showed that the feed forward algorithm performed better with PSO than with BP for nonlinear function. Juang *et al.* [66] tried optimizing the recurrent version of FFNN. He proposed a new evolutionary learning algorithm based on a hybrid of GA and PSO, called HGAPSO. In each epoch, the upper-half of the GA population were defined as elites and the rest were discarded. Enhanced by PSO and GA, the elites formed the next generation of GA. This hybrid method outperformed PSO and GA alone, for recurrent and fuzzy neural networks. Table 2 presents a survey summary about the attempts to optimize various types of artificial neural networks using different search techniques.

Table 2: Summary of the literature survey

Author	**Architecture**	**Optimization Approach**	**Outcome**
Bergstra *et al.* [57]	ANN	Grid Search	Automated hyper-parameter search to learn XOR function.
Bergstra and Bengio [58]	DBN	Random Search	Efficient parameter search over a 32-dimensional search space.
Stewart *et al.* [59]	ANN	Bayesian Optimization	Higher prediction accuracy in learning non-linear functions.
Maclaurin *et al.* [60]	FFNN	Gradient Based Optimization	Calculated unavailable gradients to decide the amount and direction of change in hyper-parameter values.
Venayagamoorthy *et al.* [61]	FFNN	Network training with PSO.	Training with PSO was competitive to training with GD.

Leung et al. [62]	FFNN	Network training with GA.	Better parameter discovery and prediction performance.
Bullinaria and Alyahya [63]	FFNN	Optimization of connection weights.	Network with optimized weights showed better classification performance.
Pasa *et al.* [64]	CNN	Re-parameterization of weights.	Network got better conditioned and showed faster convergence.
Rasdi *et al.* [65]	CNN	Weight optimization of the network.	Network showed better accuracy for MNIST and CIFAR datasets.
Juang *et al.* [66]	RNN	Weight Initialization with GA-PSO hybrid.	Better performance than simple recurrent neural network.

2.4 Scope of using Meta-heuristics for DNN optimization

Meta-heuristics have found their place for the optimization of simpler classes of neural networks but only a few relevant publications were found on their applications for complex deep networks. DNNs show higher prediction accuracy when trained with a huge amount of data. Training them with large data is involved with tuning large number of parameters. Optimizing the parameters in a reasonable amount of computational time may be difficult. Traditional approaches are not suitable for designing deep neural network architectures because they usually optimize the number and connectivity of low-level neurons only. When CNN attempts to re-configure all the parameters even when small noises are introduced, it causes over-fitting and slowness [67]. RNNs have exhibited great effectiveness in processing sequential data without the shortcomings of the traditional approaches such as - slow learning, inability to handle dynamic updates and concept drift, *etc.* [68]. However, Deep RNNs train slowly due to vanishing gradient problem [24]. Techniques like LSTM, GRU, *etc.* have shown efficient results in solving such type of problems [25][26]. Since the data is being generated at a very high speed in real-time applications, it is almost impractical to store and process all of the data. Need is to develop approximation techniques for collecting and processing valuable information from data streams. To perform such fast processing, various meta-heuristic approaches have been tested for the adaptive preprocessing of the image and sequence data [27].

The aim of the current research work is to come up with a method to optimize various hyper-parameters of a general class of deep neural network models for their predominant application areas. Meta-heuristic optimization techniques are capable to handle the optimization of multiple quantities simultaneously. They have also shown good potential to learn the hierarchal rules, which could be used for neural network's training as they are also composed of hierarchal structures [53]. With that inspiration, Genetic Algorithm based meta-heuristic approach has been selected for the optimization of the DNNs. Hyper-parameters of deep neural networks are typically optimized using manual search, grid search, random search, gradient-based methods and bayesian optimization. These techniques do not guarantee to find the best hyper-parameter combination, and they are not fast enough for real time applications [6]. Grid search guarantees to find the best solution but it gets miserably slow as the number of hyper-parameters and their choices increase as they require checking each and every combination of the hyper-parameters. This inefficiency can be conquered by using meta-heuristic approaches, which are expected to find a near-optimal combination of hyper-parameters very quickly. Most of the optimization attempts require the measurements of the performance on the entire training set [28]. Meta-heuristics can be used further to decide a subset of the training data to train the model quickly, without losing too much on the prediction performance. This approach can be very useful for RNNs to process real-time data.

3. PROBLEM STATEMETNT

3.1 Introduction

The training performance of DNNs heavily depends on the hyper-parameter choices for their design [27]. It's comparatively easy to train the DNN models but it is very difficult to decide the right hyper-parameters for their training. The hyper-parameters cannot be learnt from the data, hence it is very important to initialize their optimal values at the first place [5]. There are multiple hyper-parameters for DNN architectures such as – the number of layers, number of nodes per layer, learning rate, regularization rate, loss function, activation function, optimization method, evaluation method, *etc.* [69]. Each of these parameters could have a few to thousands of choices. Deciding the best choice for each of these hyper-parameters all at once is chaotically complex. Manual attempts to initialize the optimal hyper-parameters don't guarantee an adequate performance; hence they are not suitable for complex recognition and prediction tasks of today's performance-oriented era. Grid search based brute force methods are very time-consuming; hence they are also inadequate for time-bound predictions. It's needed to compose a method to quickly find a near-optimal combination of the hyper-parameter values.

3.2 Problem Formulation

Given a cost function $f(x)$ to train a neural network model m, consider an n dimensional search space P denoted as $P = \{p_1, p_2, p_3, \dots, p_n\}$. Each point in the search space P denotes a solution tuple n values for n hyper-parameters, i.e., $p_i = \{v_1, v_2, v_3, \dots, v_n\}$. The problem is to select the solution point p_i to satisfy the following constraints:

- Time to search optimal hyper-parameter tuple (t_s) should be minimum.
- The value of the cost function $f(x)$ should be minimum.
- Accuracy (acc) to train the model m with t_s should be more than the threshold accuracy (acc_{th}).
- The value of the loss metric ($loss$) while training m with t_s should be less than threshold loss value ($loss_{th}$)

Mathematically, above stated constraints can be modeled as follows:

$$\text{Cost Function: } f(c) = \sum_{i=0}^{K} (y_i - h(x^i))^2 \quad (1)$$

Where:

$$h(x^i) = \sum_{j=0}^{M} x_j w_j$$

Goal: minimize $f(c)$ for the value of p_i

Where:

K: Number of training examples, i.e. inputs,

M: Total number of weights,

x^i: ith input,

$h(x^i)$: Predicted output for x^i,

y_i: Actual output for x^i.

3.3 Objectives

The current research work focuses on optimizing the general architecture of DNNs by determining the optimal values of their hyper-parameters. Following are the core objective of the thesis:

- To study the architecture of general purpose DNNs, CNNs for object recognition and RNNs for sequential predictions; understand the most important DNN hyper-parameters and their choices and survey various approaches for hyper-parameter tuning.
- To evaluate the most suitable approach for DNN optimization and implement the same; analyze various hyper-parameter combinations and evaluate the most suitable one.
- To propose a method to optimally train various application specific DNNs. The aim is to quickly find a near-optimal hyper-parameter combination for their training.
- To evaluate the performance of optimized DNNs against the benchmark performances known in the literature. For object recognition, train CNN models and for sequential data prediction, train RNN models.

4. PROPOSED METHODOLOGY

This chapter proposes a solution to tune DNN hyper-parameters using GA based meta-heuristic search. Six important hyper-parameters i.e. number of hidden layers, number of processing nodes in each layer, activation function, network optimizer, regularization function and loss function are tuned for a suitable combination of their values. The search space can be considered as a 6 dimensional space, and a hyper-parameter combination can be thought of as the coordinate in this space. The problem is to quickly find such coordinate for which the cost function defined in the Eq. 1 is minimized. Proposed optimal search method for the hyper-parameter discovery is mathematically described in Eq. 3. It is compared to the grid search based brute force method described in Eq. 2.

Grid Search

$$Goal: \quad min \sum_{}^{V_1} \sum_{}^{V_2} \sum_{}^{V_3} \sum_{}^{V_4} \sum_{}^{V_5} \sum_{}^{V_6} f(c) \tag{2}$$

Where:

f(c): cost function as defined in Eq. 1,

V_1: Total choices for Number of Layers,

V_2: Total choices for Number of Nodes,

V_3: Total choices for Activation Functions,

V_4: Total choices for Network Optimizers,

V_5: Total choices for Regularizers,

V_6: Total choices for Loss Functions.

The grid search exhaustively evaluates the value of f(x) for each combination of the hyper-parameters and finally reports the combination causing the minimum value of f(x).

$$Meta - heuristic\ Search$$
$$Goal: \quad min \sum^{gen} f(c)^p \tag{3}$$

Where:

f(c): cost function as defined in Eq. 1,

gen: Number of generations to evolve the networks,

p ∈ P, P is 6 dimensional hyper-parameter space.

The meta-heuristic search randomly initializes the hyper-parameter set p and evolves it by exchanging and mutating the parameters for a specified number of generations. The cost function is evaluated for the evolved hyper-parameters and the performance is recorded. The number of generations is aimed to be selected in such a way to achieve the accuracy above acc_{th} and keep the loss below $loss_{th}$. The detailed implementation of the Genetic Algorithm and the design of DNNs are discussed in chapter 5 and the experimental results are described in chapter 6.

5. IMPLEMENTATION AND RESULTS

This chapter discusses the implementation details of hyper-parameter optimization for various DNN architectures. The implementation has been carried out considering four use-cases. CNN has been optimized in the first two cases for the recognition of image objects while the last two cases deal with the prediction of sequential data streams. Simple RNN, LSTM, GRU and Optimized RNN are implemented for the prediction of the next value given the values of previous 100 time-steps.

5.1 Implementation Platforms

Various DNN architectures are implemented using deep learning libraries. Following platforms have been utilized for the same.

- **Hardware Platforms**:
    - **GPU**: The experiments to discover the best hyper-parameters have been performed on Nvidia Tesla K80 GPU machine with 24GB RAM and 4992 CUDA cores.
    - **CPU**: Recognition and prediction tasks have been performed on Intel(R) Core(TM) i5-7200U, 2.70 GHz, 8GB RAM CPU machine with 64 bit Windows 10 OS machine. CNN and RNN are trained using the hyper-parameters discovered by meta-heuristic search.
- **Software Environment**: Python Anaconda 3.6 has been used as python package manager and environment manager. The implementation has been carried out with Jupyter notebook 5.4.0 and Python Spyder IDE 3.2.7.
- **Machine Learning Libraries**: Following libraries have been used for basic programming operations - numpy, matplotlib, panda, sklearn, urllib and seaborn.
- **Deep Learning Libraries**: The libraries used for implementing deep networks are - keras with TensorFlow and theano as backend.

- **Meta-heuristic Toolbox**: The toolbox provided by DEAP python package has been utilized to carry out some of the Genetic Algorithm's operations.
- **Data Repositories**: The stock market data has been fetched from Google Finance, Yahoo Finance and Quandl.com. Quandl.com is a repository of financial, economic and alternative datasets.

5.2 Algorithm

The Genetic Algorithm introduced in section 1.3.1 has been implemented with the goal of finding the best parameters to train DNNs for image classification and sequential data prediction tasks. The implementation is inspired from Will Larson's blog-post on GA [72]. The optimization process is carried out in two phases. The first phase quickly selects an optimal combination of the network hyper-parameters to train the network. The second phase deals with finding a subset of the training data with which the network could be trained near optimally.

5.2.1 Phase 1: Hyper-parameter Tuning

Hyper-parameters are the network values that are set before performing the training on any dataset. These are the higher level properties that cannot be learnt from the data. This phase uses a GA based method to collectively tune a number of network hyper-parameters such as - no. of layers, no. of neurons per layer, activation function, optimizer, regularization rate, loss function, *etc.* Following are the steps involved in this phase:

- **Selection**: A population of randomly initialized networks is defined. The prediction accuracy is defined as the fitness measure to judge the fitness of a network. A more accurate network is considered fit.
- **Cross-over** or **Breeding**: It is the process of two members of a population generating children. Top 20% fit networks, along with 5% random unfit networks for genetic diversity are selected and their hyper-parameter values are randomly exchanged.
- **Mutation**: A parameter is randomly selected and assigned with a new value from the defined range. Mutation probability is set to 1%

GA operations are collaborated together in the evolve method. This method is called once per evolution. After executing the code for defined number of generations, optimal hyper-parameter combination is discovered.

5.2.2 Phase 2: Look-back Window Selection

This phase is implemented for RNNs only. It implements GA separately using DEAP package to select a part of the training data such that the network could be trained near optimally with as less training data as possible. The sub-set is known as the Look-back window. Following are the steps to select an optimal look-back window of the training data using GA:

- Initialize GA parameters such as - number of generations, population size, *etc.*
- Run GA using DEAP package to obtain the window size. Divide the training data into training and validation sets accordingly.
- Train Simple RNN using the training set; Use the error found on validation set as the fitness score.
- Calculate best window size for the above step with the least fitness score.

After performing above two phases, optimal network settings as well as the look-back window for RNNs are found. CNN models are designed with the suggested hyper-parameter settings and deployed to recognize the image objects. For RNN variants, a subset of the given dataset is sliced for the training according to the best look-back window size. LSTM based RNN model is trained with the sliced training set and tested with the sliced testing set. Training time, error and accuracy is calculated and compared against normal RNN implementation.

The code for the GA implementation is constructed in *optimizer.py* module. The module *train.py* utilizes it to design DNNs using *TensorFlow* and *Keras* libraries. *Train.py* is called from the grid search and optimized search functions and the network details as well as the hyper-parameter suggestions are recorded in a log file. The proposed solution with computational complexity *O(n)* is expected to bring drastic performance improvements compared to the grid search approach with complexity $O(n^6)$. Experimental results are analyzed in chapter 6. The algorithm to use the genetic algorithm for DNNs optimization is outlined in Algorithm 1.

Algorithm 1: Genetic Algorithm based Optimized DNN

Define $P = \{p_1, p_2, p_3, \dots, p_n\}$ - parameter search space.
Define $p_i = \{v_1, v_2, v_3, \dots, v_n\}$ - *network* as a set of n hyper-parameters.
Define *population_size* - denoting all the solutions
Define *num_generations* - denoting the number of iterations for GA
Define *fitness_score* – the accuracy value of the solution
Define *mutation_prob* – the probability of random parameter exchange
Define *best_network* – denoting the solution with best fitness score
Define *data; target* - known and expected data
Define *X; Y* – data subsets, sliced according to best window size
Define window_size - look-back window for near optimal training
Define best_window_size - the data window with minimum training error

Function: TrainDNNwithGA()
/* Phase 1: Hyper-parameter Tuning */
for *i* in *num_generations* do
    initialize *population_size* number of DNNs with random hyper-parameters
    find *fitness_score* of each network
    select top 20% fit and 5% random networks
    mutate the networks according to *mutation_prob*
end for
return the *best_network*

/* Phase 2: Look-back Window Selection */
for RNN implementation
    Initialize *best_window_size*, *best_RMSE, best_individuals*
    for i in *num_generations* do
        find *best_individuals*
        window_size ← size(*best_individuals*)
        X; Y subset of data; target sliced according to window_size
        train and test RNN with *X* and *Y*
        calculate *RMSE*
        if (*RMSE* < *best_RMSE)* then
            best_RMSE ← *RMSE*
            best_window_size ← *window_size*
        end if
    end for
end for
return *best_window_size*

if (*network* == CNN)
    X; *Y* - same as data, target
else if (network == RNN)
    X; *Y* – subset of data; target sliced according to *best_window_size*
end if

train and test *best_network* with X and Y
calculate RMSE; Accuracy and Training Time

5.3 Experiment Use-cases

Hyper-parameter tuning and DNN optimization has been demonstrated for the following use-cases:

5.3.1 Use-case 1: MNIST digit recognition using CNN

The demand of quick and accurate object detection systems has constantly been increasing with the rise of smart video surveillance, autonomous vehicles, face recognition apps, *etc.* CNNs provide one of the most successful object recognition approaches. With the idea to make them even more quick and accurate, their hyper-parameter tuning has been considered in the first two use-cases. The first use-case deals with recognizing simpler images of handwritten digits while the second use-case attempts to recognize the pictures of various objects.

Figure 5.1: Sample images of MNIST dataset [65]

Use-case 1 attempts to recognize MNIST images using CNN. MNIST (Modified National Institute of Standards and Technology) is a dataset of 70,000 gray scale images of hand-written digits. It consists of 60,000 training samples and 10,000 testing samples. It is a modified subset of a larger dataset available from NIST (National Institute of Standards and Technology). Fig. 5.1 shows sample MNIST images which have been size-normalized to fit into a 28x28 pixel size.

5.3.2 Use-case 2: CIFAR image recognition using CNN

Use-case 2 implements a CNN to recognize the objects of a relatively challenging CIFAR-10 (The Canadian Institute for Advanced Research) dataset. The original CIFAR dataset contains 80 million labeled images. The CIFAR-10 is a subset with 60000 images of size 32x32 pixels, with 50000 samples for training and 10000 samples for testing. As shown in Fig. 5.2, CIFAR images are arranged into 10 classes, with 6000 images in every class.

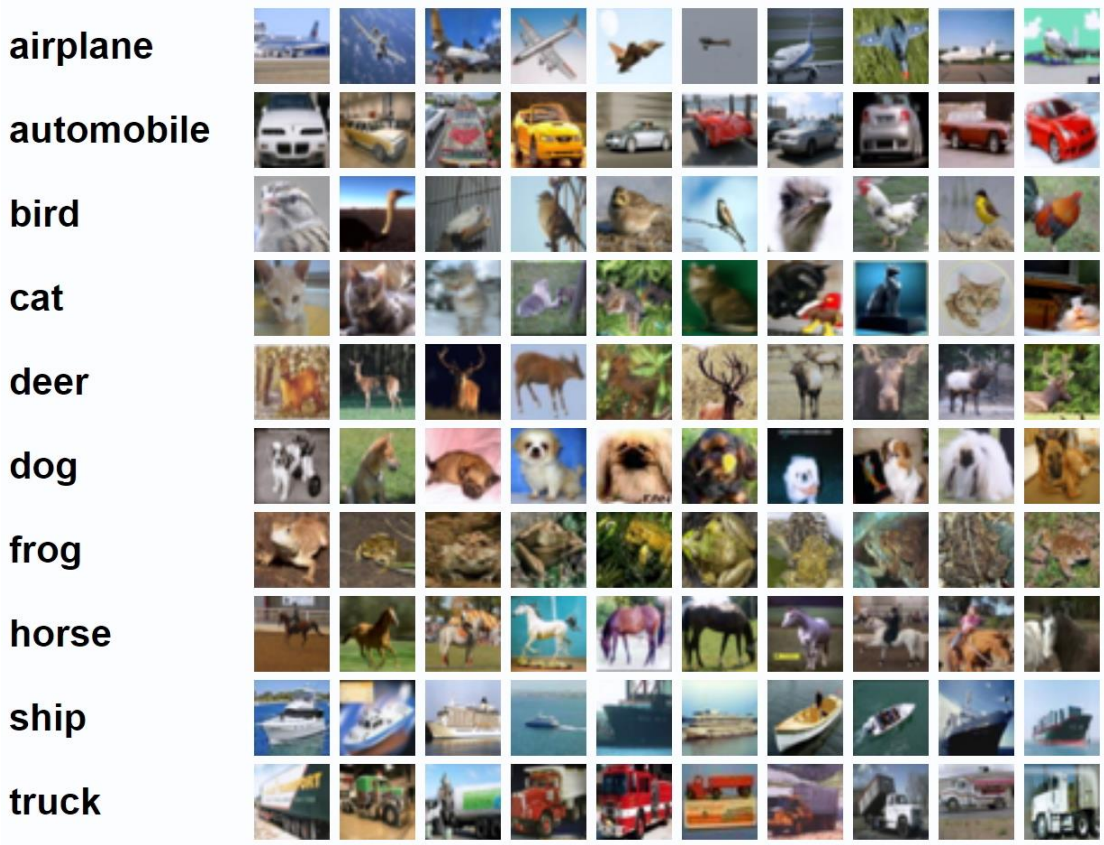


Figure 5.2: Sample images of CIFAR dataset [65]

5.3.3 Use-case 3: Sample streaming data prediction using RNN

The data is growing at an unimaginable rate. Since a majority of our daily life activities are being carried out using technical devices, by predicting the future data which will be generated by such devices, the trends of our activities can be mapped and better decisions can be made. These devices generate data over the time which needs to be captured and analyzed on the go. It is becoming increasingly important to build real-time data processing architectures that provide the flexibility to choose between speed and accuracy of the data prediction [43]. With that as an inspiration,

use-case 3 and 4 are designed to predict the next value for given data-streams. The streaming data is rapidly generating sequence of data that continuously arrives for processing at a high speed from varied sources [55]. Examples of such data consist of sensor data, credit card transactions, e-commerce records, real-time surveillance information, telecommunication data, weather reports, stock market data, *etc.*

This use-case considers following three types of sequential data-streams. A data stream is general flow of data which could be of different kinds such as - sequential data, temporal data and time series data. Fig. 5.3 shows sample examples of such data signals.

- **Rectangular Pulse data**: It denotes periodic data of the non-sinusoidal form. Electronic devices, connected networks, *etc.* produce this form of data showing their on and off states.
- **Sinusoidal Pulse data**: It denotes periodic data of infinite duration. Periodically increasing and decreasing phenomenon like traffic flow, network flow, weather patterns, *etc.* produce data in this type of format.
- **Sinc Pulse data**: The cardinal sine pulse or in-short sinc pulse, denotes the finite number of un-normalized data samples. Examples of such data involve the data signals carried out by coaxial cables, oscillatory response signals from sensors and controllers, multiplexed biochemical signals, *etc.*

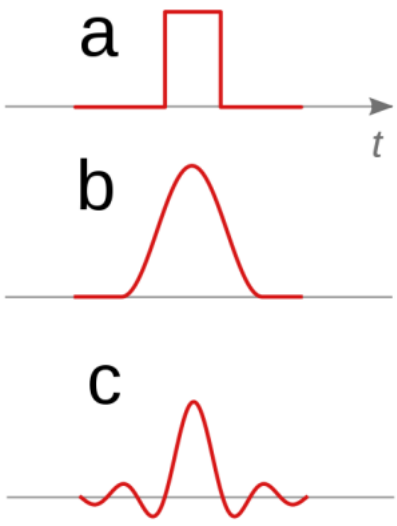


Figure 5.3: a - rectangular, b - sinusoidal and c - sinc data pulse [70]

5.3.4 Use-case 4: Stock market data prediction using RNN

Recent data from Indian stock market has been dynamically captured and stored in unrestricted tables. It is then fed into deep recurrent networks to make continuous predictions. Python library '*urllib*' is utilized to fetch the data from Google Finance, Yahoo Finance and Quandl.com. The data from 15 Feb to 25 May of the following benchmark indices have been used for the analysis. Fig. 5.4 shows the values of the benchmarks for the tenure considered.

- **BSE Sensex 30:** Symbolically denoted as *'INDEXBOM:* SENSEX', BSE Sensex 30 is a measure indicating the relative pricing of the stocks of Bombay Stock Exchange (BSE). Sensex 30 is an additive value of 30 shares from various sectors and domains. It represents an overall picture of Indian share market.

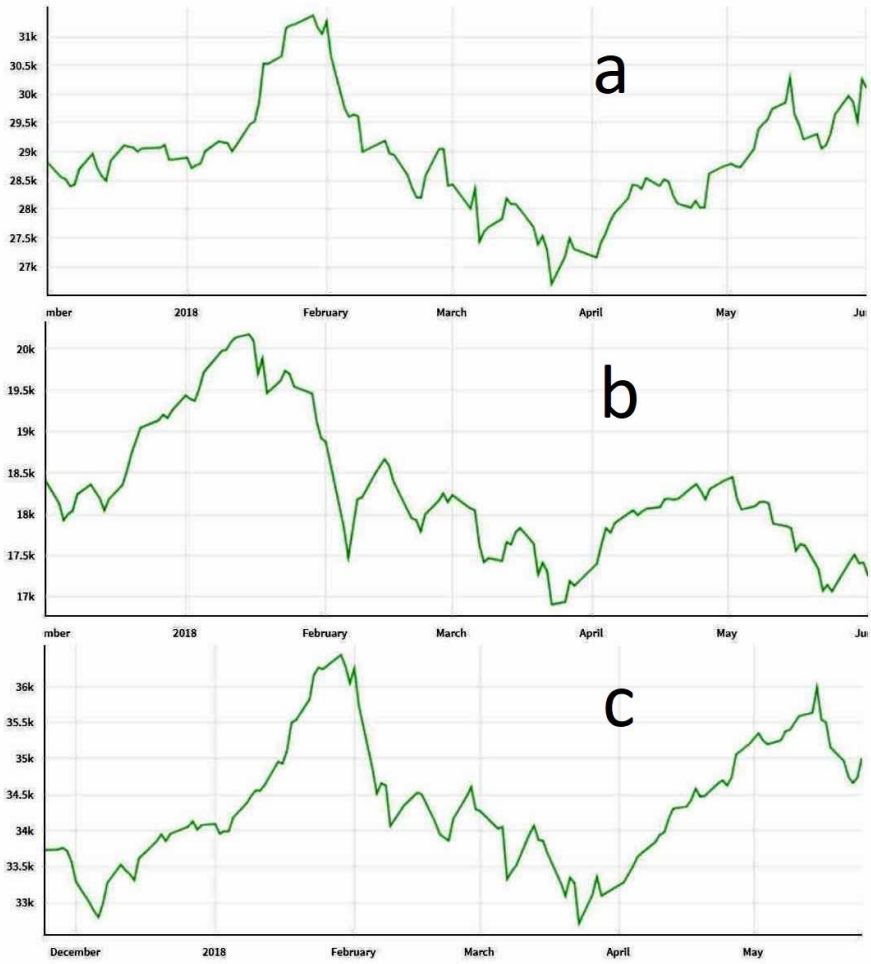


Figure 5.4: a – BSE Sensex, b – BSE SmallCap and c – BSE Bank stock data [71]

- **BSE Small Cap**: It denotes the relative situation of the companies with small capitalization, for example start-ups. It is a volatile benchmark with high-risk and

high-return prepositions as the smaller companies involve both the possibilities of growing rapidly or depreciating away. It's denoted as '*INDEXBOM: BSE SmallCap'*.

- **BSE Bank**: Denoted as '*INDEXBOM: BSE BANKEX'* represents a relative measure of Bank sector stocks. As banking impacts other sectors and the economy of the country, this index also approximates an overall progress of the whole market.

Table 3 presents the summary of the use-cases described above.

Table 3: Summary of the use-cases

Use-case	DNN Architecture	Dataset	Evaluation Metric	
			Statistical Evaluation	Prediction Plots
1	CNN	MNIST	Confusion Matrix	Accuracy & Loss Plots
2	CNN	CIFAR10	Confusion Matrix	Accuracy & Loss Plots
3	RNN	Sample Streams	Time & RMSE Tables	Prediction Comparisons
4	RNN	Stock Market Data	Time & RMSE Tables	Prediction Comparisons

5.4 Experimental Results

The optimal hyper-parameters are first discovered using the grid search for the four use-cases formulated in chapter 5. It has adopted a brute-force based strategy to check each and every combination of the parameters by putting 6 nested loops in the code. The network was built with the hyper-parameters being considered and evaluated for its accuracy. The combinations associated with top 50 most accurate networks were recorded. The proposed method to optimize DNN hyper-parameters using GA is then evaluated for all the use-cases. For each use-case a population of 20 randomly selected networks is considered and search is performed for 10 generations. For every generation, the hyper-parameters of 5 most accurate networks are recorded in a log file. 50 such combinations of 6 hyper-parameters are obtained. Frequency of the occurrence of each value for every hyper-parameter is summarized a table. Finally the most frequently occurring value for each hyper-parameter is considered and the network is trained and evaluated in detail, in comparison to the grid search results, as well as to the benchmark performance measurements. A second layer of optimization to find the best look-back window for the sequential data is also implemented for use-case 3 and 4. Hyper-parameter selection part is implemented on Nvidia Tesla K10

GPU. Model evaluation with the best hyper-parameter combination is performed on Intel i5, 2.7 GHz CPU. Comparative evaluations for accuracy and training time for every use-case are presented in the following sections.

5.4.1 Use-case 1: MNIST digits recognition using CNN

CNN is implemented to recognize MNIST hand-written digits. It has first been evaluated for various the hyper-parameter choices; and then it is implemented for the most suitable choice thus emerged out.

5.4.1.1 Hyper-parameter Discovery

Table 4 summarizes the frequency of various hyper-parameter values for the top 50 combinations discovered by grid search and optimized search. The most suitable hyper-parameter values for this use-case are listed in Table 5. Both grid search and optimized search returned almost the same parameters. The grid search took **47 minutes** on Nvidia Tesla K10 GPU for evaluating the CNN model for 100 epochs, while the optimized search took **5 minutes and 48 seconds** which is approximately **8 times faster** than the grid searh.

Table 4: Experiment summary for use-case 1

# Layers			**# Neurons**			**Act Fun**			**Optimizer**			**Regularizer**			**Loss Fun**		
	gs	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**
$\mathbf{v_{11}}$	2	3	$\mathbf{v_{21}}$	3	1	$\mathbf{v_{31}}$	1	2	$\mathbf{v_{41}}$	4	3	$\mathbf{v_{51}}$	18	20	v_{61}	16	21
$\mathbf{v_{12}}$	20	17	$\mathbf{v_{22}}$	8	7	$\mathbf{v_{32}}$	5	5	$\mathbf{v_{42}}$	8	6	$\mathbf{v_{52}}$	32	30	v_{62}	12	9
$\mathbf{v_{13}}$	9	11	$\mathbf{v_{23}}$	10	11	$\mathbf{v_{33}}$	7	8	$\mathbf{v_{43}}$	7	8	-	-	-	v_{63}	12	15
$\mathbf{v_{14}}$	12	9	$\mathbf{v_{24}}$	13	15	$\mathbf{v_{34}}$	12	13	$\mathbf{v_{44}}$	9	10	-	-	-	v_{64}	8	5
$\mathbf{v_{15}}$	4	8	$\mathbf{v_{25}}$	14	14	$\mathbf{v_{35}}$	10	12	$\mathbf{v_{45}}$	11	12	-	-	-	-	-	-
$\mathbf{v_{16}}$	3	2	$\mathbf{v_{26}}$	2	2	$\mathbf{v_{36}}$	15	10	$\mathbf{v_{46}}$	11	11	-	-	-	-	-	-

gs: Grid Search; **os**: GA based Optimized Search; $\mathbf{v_{11}}$: 1; $\mathbf{v_{12}}$: 2; $\mathbf{v_{13}}$: 3; $\mathbf{v_{14}}$: 4; $\mathbf{v_{15}}$: 5; $\mathbf{v_{16}}$: 6; $\mathbf{v_{21}}$: 64; $\mathbf{v_{22}}$: 128; $\mathbf{v_{23}}$: 256; $\mathbf{v_{24}}$: 512; $\mathbf{v_{25}}$: 768; $\mathbf{v_{26}}$: 1024; $\mathbf{v_{31}}$: Linear; $\mathbf{v_{32}}$: Sigmoid; $\mathbf{v_{33}}$: tanh; $\mathbf{v_{34}}$: ReLU; $\mathbf{v_{35}}$: ELU; $\mathbf{v_{36}}$: SELU; $\mathbf{v_{41}}$: SGD; $\mathbf{v_{42}}$: AdaGrad; $\mathbf{v_{43}}$: AdaDelta; $\mathbf{v_{44}}$: Adam; $\mathbf{v_{45}}$: Nadam; $\mathbf{v_{46}}$: AdaMax; $\mathbf{v_{51}}$: l1(0.01); $\mathbf{v_{52}}$: l2(0.01); $\mathbf{v_{61}}$: mean squared error; $\mathbf{v_{62}}$: mean absolute error; $\mathbf{v_{63}}$: categorical crossentropy; $\mathbf{v_{64}}$: squared hinge.

Table 5: Optimal hyper-parameters for use-case 1

Hyper-parameter	Grid Search	Optimized Search
No. of layers	2	2
Neurons per layer	512	512
Activation Function	SELU	ReLU
Network optimizer	Nadam/Adamax	AdaMax
Regularizer	L2(0.01)	L2(0.01)
Loss Function	Mean squared error	Mean squared error

5.4.1.2 Performance Evaluation

CNN has been implemented with the optimal hyper-parameter combination that was discovered by the optimized search, i.e. *{'nb_layers': 2, 'nb_neurons': 512, 'activation': 'relu', 'optimizer': 'adamax', 'regularizer': 'l2(0.01)', 'loss': 'mean_squared_error'}*. Table 6 shows the accuracy and loss values on the training and validation datasets for 100 epochs. Fig. 5.5 and Fig. 5.6 demonstrate the convergence of accuracy and loss values over the epochs. Epoch denotes one pass of network training over all the data. Loss is a scalar value that is aimed to be minimized during the training.

Table 6: Results for MNIST digit recognition

Epoch#	ETA	train_acc	val_acc	train_loss	Val_loss
1	13s	91.75 %	96.09 %	0.2724	0.1237
10	7s	98.55 %	98.12 %	0.0465	0.0605
20	10s	99.22 %	98.39 %	0.0258	0.0552
30	10s	99.47 %	98.45 %	0.0186	0.0560
40	10s	99.54 %	98.53 %	0.0129	0.0557
50	10s	99.59 %	98.58 %	0.0113	0.0574
60	10s	99.64 %	98.64 %	0.0093	0.0596
70	10s	99.69 %	98.70 %	0.0076	0.0595
80	10s	99.73 %	98.75 %	0.0066	0.0595
90	10s	99.73 %	98.83 %	0.0061	0.0609
100	10s	99.72 %	98.82 %	0.0056	0.0609

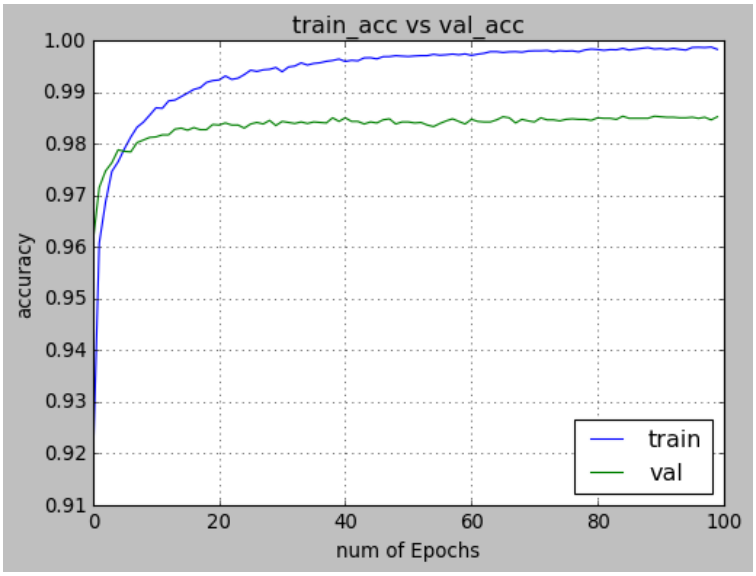


Figure 5.5: Use-case 1 accuracy plot

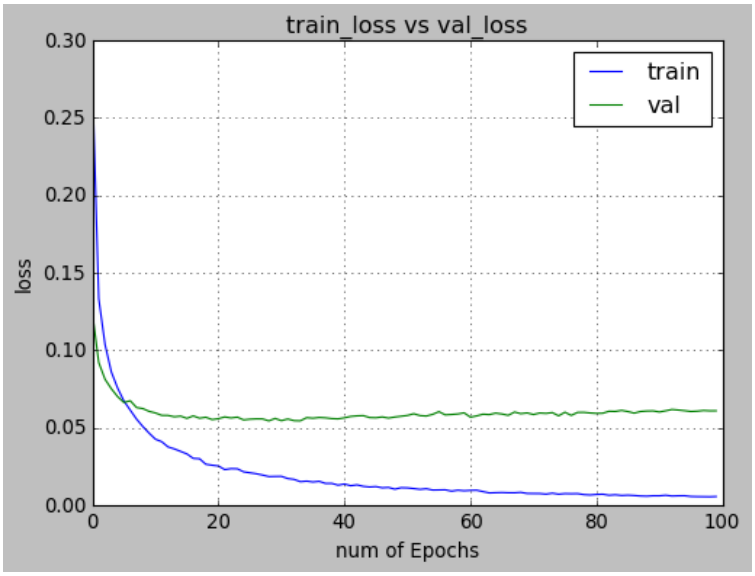


Figure 5.6: Use-case 1 loss plot

This use-case resulted in 98.82% prediction accuracy which is numerically presented in Fig. 5.7. It is as per the known benchmark performance for MNIST dataset [16].

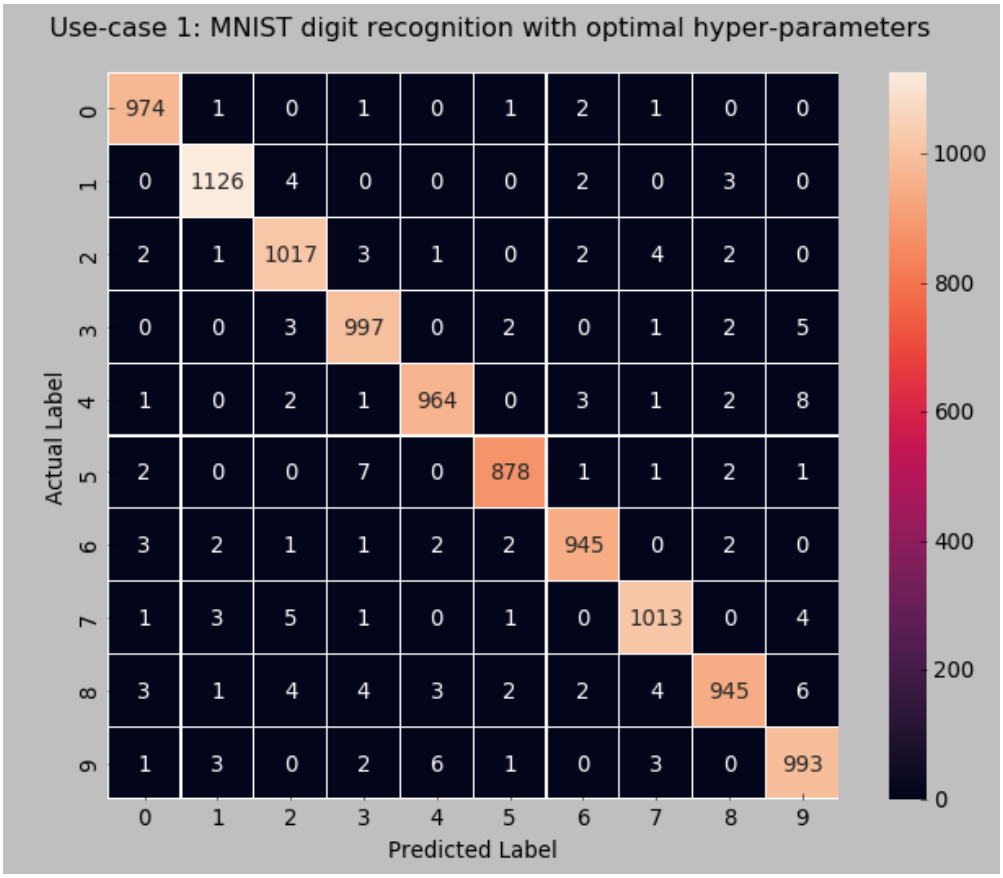


Figure 5.7: Use-case 1 confusion matrix

5.4.2 Use-case 2: CIFAR images recognition using CNN

CNN is implemented to recognize the images of CIFAR10 dataset. It is a relatively complex data-set as compared to the MNIST dataset. It has first been evaluated for various the hyper-parameter choices; and then it is implemented for the most suitable choice fetched by the optimized search.

5.4.2.1 Hyper-parameter Discovery

Table 7 summarizes the frequency of various hyper-parameter values for top 50 combinations discovered by grid search and optimized search. The most suitable hyper-parameter values for this use-case are listed in Table 8.

Table 7: Experiment summary for use-case 2

# Layers			# Neurons			Act Fun			Optimizer			Regularizer			Loss Fun		
	gs	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**
$\mathbf{v_{11}}$	3	7	$\mathbf{v_{21}}$	3	4	$\mathbf{v_{31}}$	1	2	$\mathbf{v_{41}}$	3	3	$\mathbf{v_{51}}$	21	25	v_{61}	11	14
$\mathbf{v_{12}}$	9	9	$\mathbf{v_{22}}$	7	7	$\mathbf{v_{32}}$	6	7	$\mathbf{v_{42}}$	8	6	$\mathbf{v_{52}}$	29	25	v_{62}	14	10
$\mathbf{v_{13}}$	11	9	$\mathbf{v_{23}}$	8	9	$\mathbf{v_{33}}$	6	6	$\mathbf{v_{43}}$	9	7	-	-	-	v_{63}	17	20
$\mathbf{v_{14}}$	12	15	$\mathbf{v_{24}}$	12	13	$\mathbf{v_{34}}$	17	14	$\mathbf{v_{44}}$	8	11	-	-	-	v_{64}	8	6
$\mathbf{v_{15}}$	9	8	$\mathbf{v_{25}}$	13	14	$\mathbf{v_{35}}$	10	11	$\mathbf{v_{45}}$	10	10	-	-	-	-	-	-
$\mathbf{v_{16}}$	6	2	$\mathbf{v_{26}}$	7	3	$\mathbf{v_{36}}$	9	10	$\mathbf{v_{46}}$	12	13	-	-	-	-	-	-

gs: Grid Search; **os**: GA based Optimized Search; $\mathbf{v_{11}}$: 1; $\mathbf{v_{12}}$: 2; $\mathbf{v_{13}}$: 3; $\mathbf{v_{14}}$: 4; $\mathbf{v_{15}}$: 5; $\mathbf{v_{16}}$: 6; $\mathbf{v_{21}}$: 64; $\mathbf{v_{22}}$: 128; $\mathbf{v_{23}}$: 256; $\mathbf{v_{24}}$: 512; $\mathbf{v_{25}}$: 768; $\mathbf{v_{26}}$: 1024; $\mathbf{v_{31}}$: Linear; $\mathbf{v_{32}}$: Sigmoid; $\mathbf{v_{33}}$: tanh; $\mathbf{v_{34}}$: ReLU; $\mathbf{v_{35}}$: ELU; $\mathbf{v_{36}}$: SELU; $\mathbf{v_{41}}$: SGD; $\mathbf{v_{42}}$: AdaGrad; $\mathbf{v_{43}}$: AdaDelta; $\mathbf{v_{44}}$: Adam; $\mathbf{v_{45}}$: Nadam; $\mathbf{v_{46}}$: AdaMax; $\mathbf{v_{51}}$: l1(0.01); $\mathbf{v_{52}}$: l2(0.01); $\mathbf{v_{61}}$: mean squared error; $\mathbf{v_{62}}$: mean absolute error; $\mathbf{v_{63}}$: categorical crossentropy; $\mathbf{v_{64}}$: squared hinge.

Both grid search and optimized search returned the almost same parameters. The grid search took **50 minutes** on Nvidia Tesla K10 GPU for evaluating the CNN model for 100 epochs, while the optimized search took **6.5 minutes** showing a **7.5 times speed-up**.

Table 8: Optimal hyper-parameters for use-case 2

Hyper-parameter	Grid Search	Optimized Search
No. of layers	4	4
Neurons per layer	768	768
Activation Function	ReLU	ReLU
Network optimizer	AdaMax	AdaMax
Regularizer	L2	L1/L2
Loss Function	Mean squared error	Categorical crossentropy

5.4.2.2 Performance Evaluation

CNN has been implemented with the optimal hyper-parameter combination of the optimized search, i.e. *{'nb_layers': 4, 'nb_neurons': 768, 'activation': 'relu', 'optimizer': 'adamax', 'regularizer': 'l2(0.01)', 'loss': 'categorical_crossentropy'}*. Table 9 shows the accuracy and loss values on the training and validation sets for 100 epochs. Fig. 5.8 and Fig. 5.9 demonstrate the convergence of accuracy and loss values over the epochs.

Table 9: Results for CIFAR digit recognition

Epoch#	ETA	train_acc	val_acc	train_loss	Val_loss
1	59s	28.94 %	36.70 %	1.9597	1.7551
10	47s	44.68 %	47.40 %	1.5345	1.4980
20	47s	48.92 %	49.56 %	1.4251	1.4265
30	47s	51.40 %	50.40 %	1.3540	1.3876
40	47s	53.50 %	51.79 %	1.3016	1.3724
50	46s	54.58 %	52.48 %	1.2607	1.3289
60	47s	56.02 %	52.81 %	1.2300	1.3653
70	47s	56.94 %	53.82 %	1.2038	1.3380
80	47s	58.26 %	53.76 %	1.1764	1.3380
90	64s	58.45 %	54.88 %	1.1552	1.3399
100	65s	59.40 %	54.95 %	1.1343	1.3314

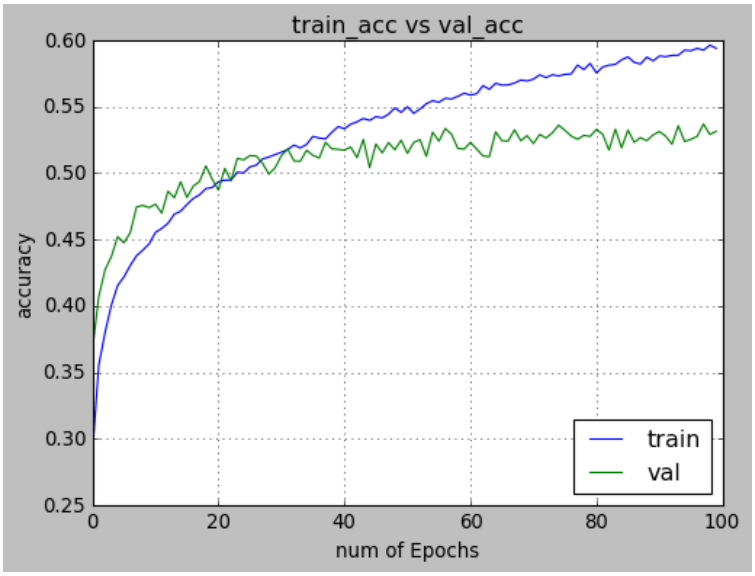


Figure 5.8: Use-case 2 accuracy plot

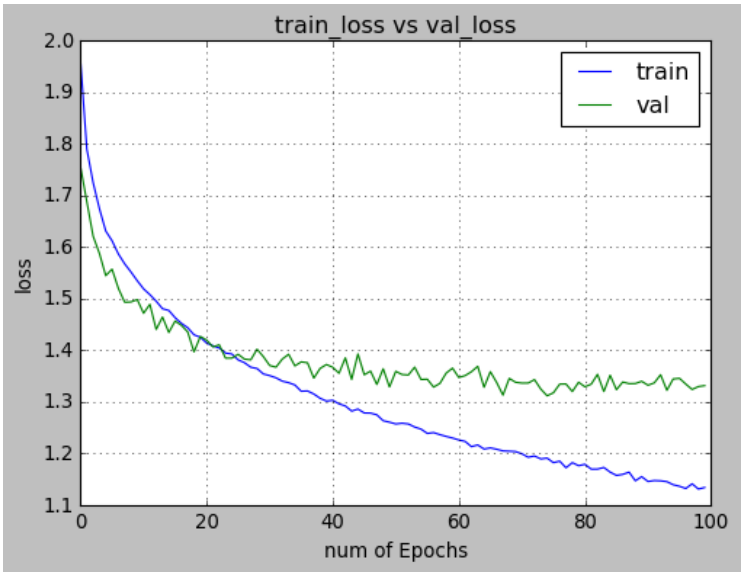


Figure 5.9: Use-case 2 loss plot

This use case resulted in 54.95% prediction accuracy, which is as per the standard DNN implementations without batch normalization [15]. The performance is numerically presented in Fig. 5.10.

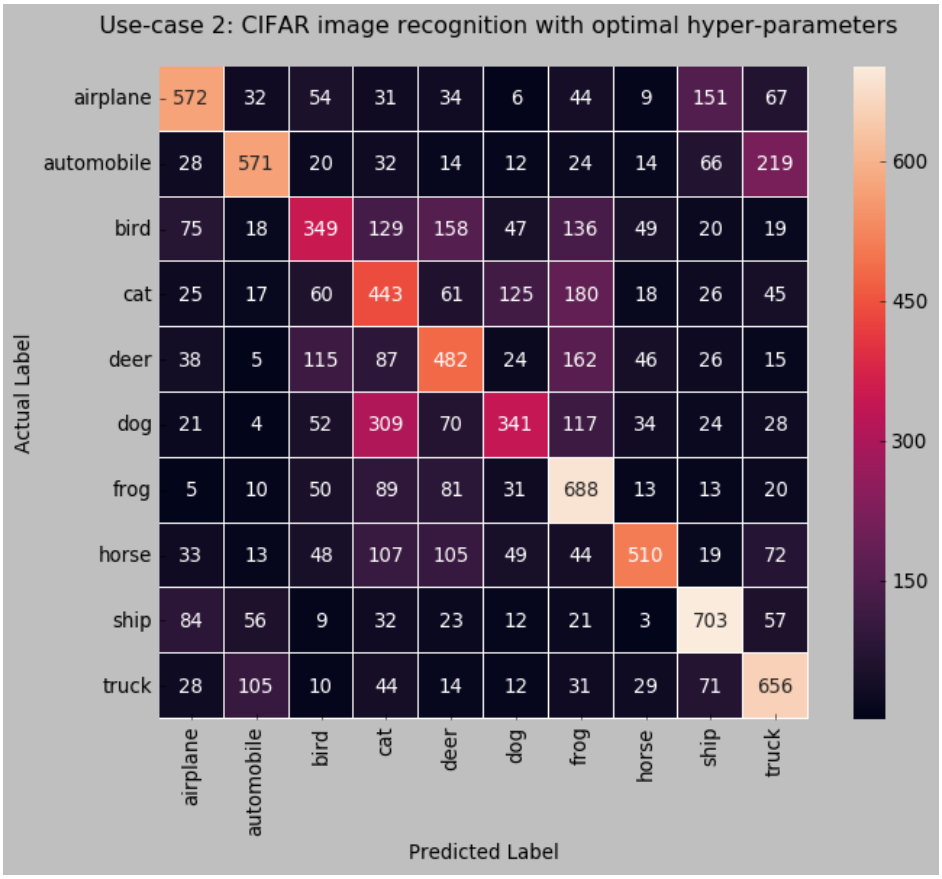


Figure 5.10: Use-case 2 confusion matrix

5.4.3 Use-case 3: Sample streaming data prediction using RNN

A RNN has been implemented to predict the next value for a given data stream. The implementation has been carried out in two phases. The first phase attempts to discover the best architectural parameters to design the network. The second phase determines a suitable look-back window to train the model optimally with minimum data. *OptRNN* is a deep LSTM trained with the look-back data. The additional speed-up achieved by the second optimization phase is depicted in Table 12, 13, and 14.

5.4.3.1 Hyper-parameter Discovery

Table 10 summarizes the frequency of various hyper-parameter values for their top 50 combinations discovered by grid search and optimized search. Grid search approach approximately took **16 min, 19 min and 18.5 min** for rectangular, sinusoidal and sinc pulse. GA based optimized search took around **2.5 min, 2 min 40 seconds and 3 min** respectively. That's a **6 times speed-up**. The most suitable hyper-parameter values for this use-case are listed in Table 11.

Table 10: Experiment summary for use-case 3

# Layers			**# Neurons**			**Act Fun**			**Optimizer**			**Regularizer**			**Loss Fun**		
	gs	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**
$\mathbf{v_{11}}$	3	7	$\mathbf{v_{21}}$	3	1	$\mathbf{v_{31}}$	1	3	$\mathbf{v_{41}}$	3	3	$\mathbf{v_{51}}$	23	29	v_{61}	20	21
$\mathbf{v_{12}}$	9	9	$\mathbf{v_{22}}$	8	7	$\mathbf{v_{32}}$	6	7	$\mathbf{v_{42}}$	8	6	$\mathbf{v_{52}}$	27	21	v_{62}	12	11
$\mathbf{v_{13}}$	11	9	$\mathbf{v_{23}}$	10	11	$\mathbf{v_{33}}$	7	5	$\mathbf{v_{43}}$	9	7	-	-	-	v_{63}	13	14
$\mathbf{v_{14}}$	12	15	$\mathbf{v_{24}}$	13	15	$\mathbf{v_{34}}$	13	13	$\mathbf{v_{44}}$	8	11	-	-	-	v_{64}	5	4
$\mathbf{v_{15}}$	9	8	$\mathbf{v_{25}}$	14	14	$\mathbf{v_{35}}$	13	12	$\mathbf{v_{45}}$	10	10	-	-	-	-	-	-
$\mathbf{v_{16}}$	6	2	$\mathbf{v_{26}}$	2	2	$\mathbf{v_{36}}$	9	10	$\mathbf{v_{46}}$	12	13	-	-	-	-	-	-

gs: Grid Search; **os**: GA based Optimized Search; $\mathbf{v_{11}}$: 1; $\mathbf{v_{12}}$: 2; $\mathbf{v_{13}}$: 3; $\mathbf{v_{14}}$: 4; $\mathbf{v_{15}}$: 5; $\mathbf{v_{16}}$: 6; $\mathbf{v_{21}}$: 64; $\mathbf{v_{22}}$: 128; $\mathbf{v_{23}}$: 256; $\mathbf{v_{24}}$: 512; $\mathbf{v_{25}}$: 768; $\mathbf{v_{26}}$: 1024; $\mathbf{v_{31}}$: Linear; $\mathbf{v_{32}}$: Sigmoid; $\mathbf{v_{33}}$: tanh; $\mathbf{v_{34}}$: ReLU; $\mathbf{v_{35}}$: ELU; $\mathbf{v_{36}}$: SELU; $\mathbf{v_{41}}$: SGD; $\mathbf{v_{42}}$: AdaGrad; $\mathbf{v_{43}}$: AdaDelta; $\mathbf{v_{44}}$: Adam; $\mathbf{v_{45}}$: Nadam; $\mathbf{v_{46}}$: AdaMax; $\mathbf{v_{51}}$: l1(0.01); $\mathbf{v_{52}}$: l2(0.01); $\mathbf{v_{61}}$: mean squared error; $\mathbf{v_{62}}$: mean absolute error; $\mathbf{v_{63}}$: categorical crossentropy; $\mathbf{v_{64}}$: squared hinge.

Table 11: Optimal hyper-parameters for use-case 3

Hyper-parameter	Grid Search	Optimized Search
No. of layers	4	4
Neurons per layer	768	512
Activation Function	ReLU/ELU	ReLU
Network optimizer	AdaMax	AdaMax
Regularizer	L2	L1
Loss Function	Mean squared error	Mean squared error
Lookback window size		47%

5.4.3.2 Performance Evaluation

Simple RNN, LSTM, GRU and Optimized RNN architectures with single and multiple hidden layers are trained for the prediction of the next value of a sequential data stream. The implementation is carried out with the most suitable hyper-parameter discovered by the optimized search, i.e. *{'nb_layers': 4, 'nb_neurons': 512,*

'activation': 'relu', 'optimizer': 'adamax', 'regularizer': 'l1(0.01)', 'loss': 'mean_squared_error'}. Prediction performance is shown in Table 12, Table 13 and Table 14 for rectangular, sinusoidal and sinc data pulses respectively. Accuracy is calculated by allowing 20% error margin in the predictions.

Table 12: Results for rectangular pulse data prediction

Architecture	**1 Layered Model**			**4 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	65.00 %	0.2066	1.31s	67.00 %	0.1856	6.34s
LSTM	60.00 %	0.2598	2.18s	72.00 %	0.1625	7.82s
GRU	59.00 %	0.2305	2.30s	70.00 %	0.1832	4.32s
OptRNN	42.00 %	0.3317	1.46s	57.00 %	0.2681	4.16s

Table 13: Results for sinusoidal pulse data prediction

Architecture	**1 Layered Model**			**4 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	59.00 %	0.4058	1.56s	60.00 %	0.2479	5.65s
LSTM	40.00 %	0.4207	1.96s	58.00 %	0.2702	6.69s
GRU	36.00 %	0.3883	1.63s	52.00 %	0.3165	5.13s
OptRNN	45.00 %	0.3580	0.96s	49.00 %	0.3580	2.76s

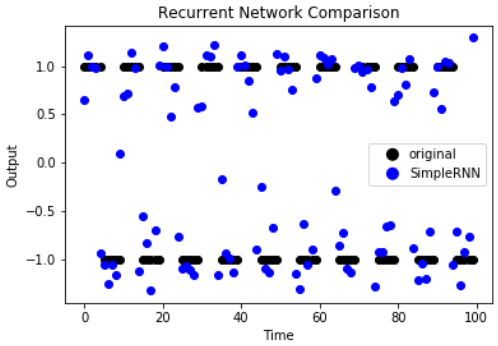
Table 14: Results for sinc pulse data prediction

Architecture	**1 Layered Model**			**4 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	81.00 %	0.2523	1.81s	84.00 %	0.2367	6.18s
LSTM	87.00 %	0.1412	2.70s	88.00 %	0.1372	7.16s
GRU	89.00 %	0.1249	2.92s	91.00 %	0.1029	8.82s
OptRNN	76.00 %	0.3180	1.43s	74.00 %	0.3230	4.43s

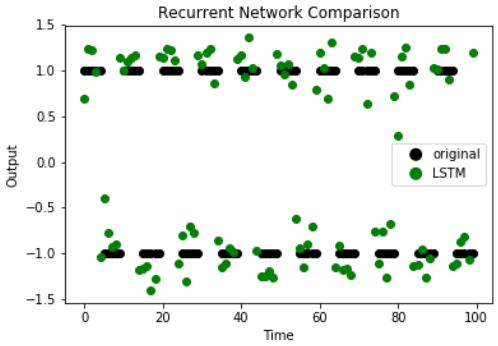
Comparing single layered and 4 layered models, the training time for LSTM and optRNN increased by a factor of 3.17 and 2.94 respectively. A lower increase in the training time as the architecture got deeper indicates the effectiveness of the meta-heuristic techniques to optimize the deeper RNN models.

5.4.3.3 Prediction Plots

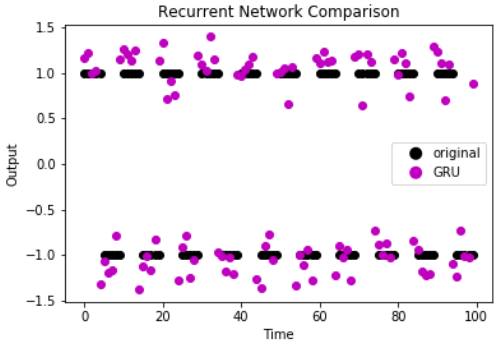
Fig. 5.11, Fig. 5.12 and Fig. 5.13 visually demonstrate the prediction performance of RNN models for rectangular, sinusoidal and sinc data pulses respectively.



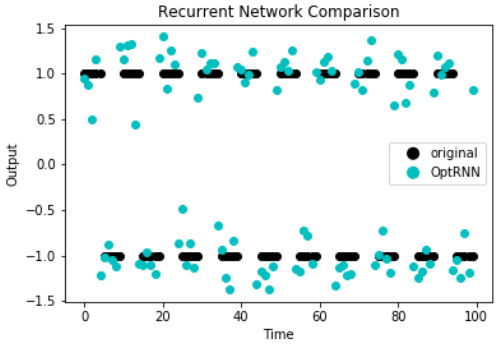
(a) SimpleRNN



(a) LSTM

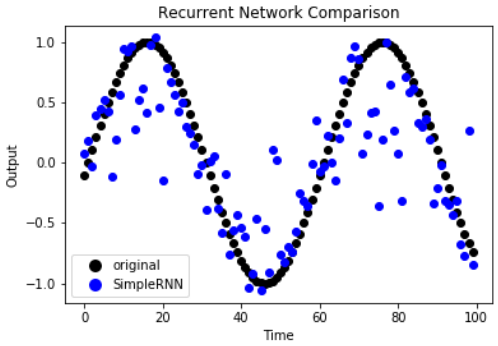


(a) GRU

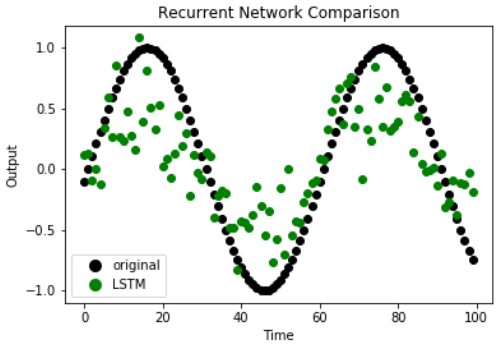


(a) OptRNN

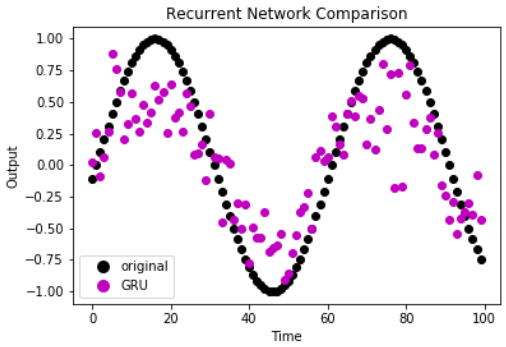
Figure 5.11: Prediction plots for rectangular pulse data



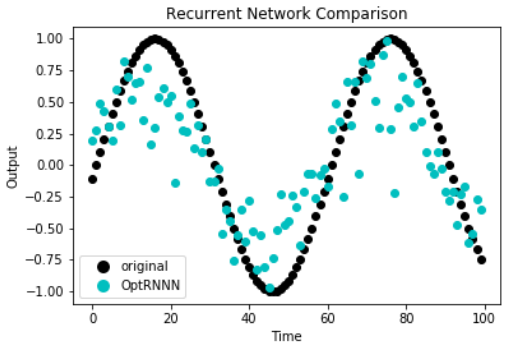
(a) SimpleRNN



(a) LSTM

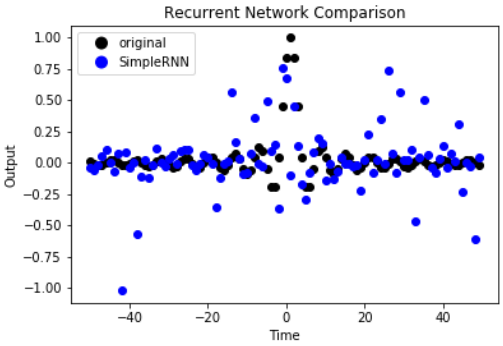


(a) GRU

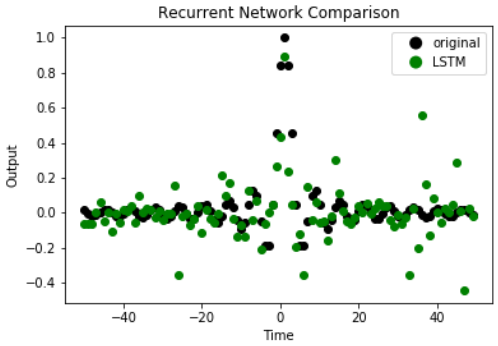


(a) OptRNN

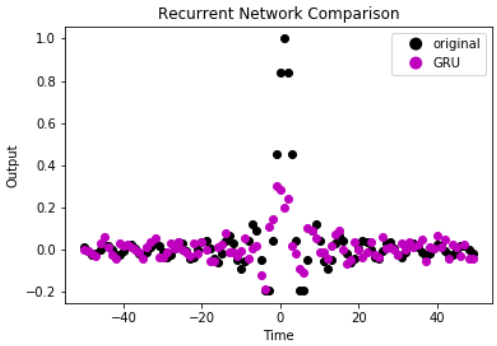
Figure 5.12: Prediction plots for sinusoidal pulse data



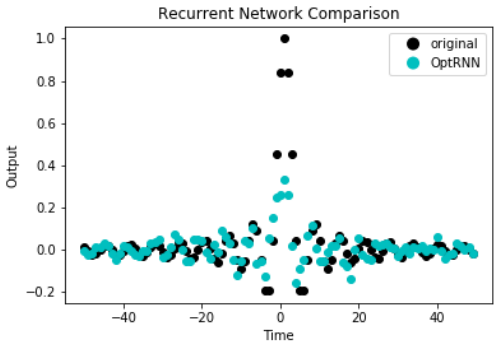
(a) SimpleRNN



(a) LSTM



(a) GRU



(a) OptRNN

Figure 5.13: Prediction plots for sinc pulse data

Fig. 5.11, Fig. 5.12 and Fig. 5.13 show the expected and predicted sequences of the data for rectangular, sinusoidal and sinc data pulses respectively. As the number of processing layers increased, the accuracy improved for all the considered architectures, i.e., Simple RNN, LSTM, GRU and OptRNN. However, LSTM has demonstrated better performance than Simple RNN and GRU with an equal number of layers and OptRNN showed comparable performance to LSTM.

The goal of this research work is to demonstrate the effectiveness of meta-heuristic based recurrent network, in producing near optimal results significantly faster. Since simple data streams are considered in use-case 3, it was easy for all the considered RNN architectures to shoot up to 100% prediction accuracy for a suitable number of training epochs. The number of training epochs is kept considerably low (of the order of 10 to 20) so that the effectiveness of optimal RNN could be depicted clearly. Use-case 4 deals with relatively complex data of the share market where 100 or more epochs are used for the network training.

5.4.4 Use-case 4: Stock market data prediction using RNN

A RNN is implemented to predict the values of Indian stocks. Three benchmark indices have been considered – Bombay Stock Exchange (BSE) Sensex30, BSE Small Capitalization shares and BSE Bank sector shares.

5.4.4.1 Hyper-parameter Discovery

Table 15 summarizes the frequency of various hyper-parameter values, for top 50 combinations discovered by grid search and optimized search. Total hyper-parameter search time for BSE Sensex 30, BSE Small Cap and BSE Bank data was **57 minutes 13 seconds** for the grid search and **9 minutes** for the optimized search. The optimized search came out to be **6.5 times faster** than the grid search, yet it returned similar hyper-parameters. Table 16 lists the most suitable hyper-parameters for this use-case.

Table 15: Experiment summary for use-case 4

# Layers			**# Neurons**			**Act Fun**			**Optimizer**			**Regularizer**			**Loss Fun**		
	gs	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**		**gs**	**os**
$\mathbf{v_{11}}$	3	5	$\mathbf{v_{21}}$	3	1	$\mathbf{v_{31}}$	2	3	$\mathbf{v_{41}}$	3	3	$\mathbf{v_{51}}$	23	23	v_{61}	17	18
$\mathbf{v_{12}}$	8	9	$\mathbf{v_{22}}$	8	7	$\mathbf{v_{32}}$	5	6	$\mathbf{v_{42}}$	8	9	$\mathbf{v_{52}}$	27	27	v_{62}	14	15
$\mathbf{v_{13}}$	6	6	$\mathbf{v_{23}}$	10	14	$\mathbf{v_{33}}$	7	6	$\mathbf{v_{43}}$	9	8	-	-	-	v_{63}	13	10
$\mathbf{v_{14}}$	9	6	$\mathbf{v_{24}}$	14	13	$\mathbf{v_{34}}$	14	13	$\mathbf{v_{44}}$	9	10	-	-	-	v_{64}	6	7
$\mathbf{v_{15}}$	12	9	$\mathbf{v_{25}}$	13	12	$\mathbf{v_{35}}$	12	11	$\mathbf{v_{45}}$	11	9	-	-	-	-	-	-
$\mathbf{v_{16}}$	12	15	$\mathbf{v_{26}}$	2	3	$\mathbf{v_{36}}$	10	11	$\mathbf{v_{46}}$	10	11	-	-	-	-	-	-

gs: Grid Search; **os**: GA based Optimized Search; $\mathbf{v_{11}}$: 1; $\mathbf{v_{12}}$: 2; $\mathbf{v_{13}}$: 3; $\mathbf{v_{14}}$: 4; $\mathbf{v_{15}}$: 5; $\mathbf{v_{16}}$: 6; $\mathbf{v_{21}}$: 64; $\mathbf{v_{22}}$: 128; $\mathbf{v_{23}}$: 256; $\mathbf{v_{24}}$: 512; $\mathbf{v_{25}}$: 768; $\mathbf{v_{26}}$: 1024; $\mathbf{v_{31}}$: Linear; $\mathbf{v_{32}}$: Sigmoid; $\mathbf{v_{33}}$: tanh; $\mathbf{v_{34}}$: ReLU; $\mathbf{v_{35}}$: ELU; $\mathbf{v_{36}}$: SELU; $\mathbf{v_{41}}$: SGD; $\mathbf{v_{42}}$: AdaGrad; $\mathbf{v_{43}}$: AdaDelta; $\mathbf{v_{44}}$: Adam; $\mathbf{v_{45}}$: Nadam; $\mathbf{v_{46}}$: AdaMax; $\mathbf{v_{51}}$: l1(0.01); $\mathbf{v_{52}}$: l2(0.01); $\mathbf{v_{61}}$: mean squared error; $\mathbf{v_{62}}$: mean absolute error; $\mathbf{v_{63}}$: categorical crossentropy; $\mathbf{v_{64}}$: squared hinge.

Table 16: Optimal hyper-parameters for use-case 4

Hyper-parameter	**Grid Search**	**Optimized Search**
No. of layers	5/6	6
Neurons per layer	256	256
Activation Function	ReLU	ReLU
Network optimizer	Nadam	AdaMax
Regularizer	L2	L2
Loss Function	Mean squared error	Mean squared error
Lookback window size		78%

5.4.4.2 Performance Evaluation

Simple RNN, LSTM, GRU and Optimized RNN architectures with single and multiple hidden layers are trained for the prediction of the next value of stock market's data-streams. The implementation is carried out with the most suitable hyper-parameter of the optimized search, i.e. *{'nb_layers': 6, 'nb_neurons': 256, 'activation': 'relu', 'optimizer': 'adamax', 'regularizer': 'l2(0.01)', 'loss': 'mean_squared_error'}*. Prediction performance is shown in Table 17, Table 18 and Table 19 for Sensex, BSE Small Cap and BSE Bank indices' data.

Table 17: Results for BSE Sensex30 data prediction

Architecture	**1 Layered Model**			**6 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	56.67 %	770.4316	5.23s	59.00 %	730.1892	17.81s
LSTM	55.00 %	778.8973	6.72s	62.33 %	713.7909	19.16s
GRU	56.33 %	772.3498	6.98s	60.67 %	724.8885	19.72s
OptRNN	49.00 %	823.9525	3.80s	54.00 %	766.7492	9.43s

Table 18: Results for BSE SmallCap data prediction

Architecture	**1 Layered Model**			**6 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	50.00 %	576.2458	4.56s	55.50 %	542.3925	12.92s
LSTM	51.67 %	523.0897	4.98s	57.00 %	536.5054	11.80s
GRU	48.33 %	577.3274	5.13s	52.00 %	519.7510	13.41s
OptRNN	46.50 %	603.9981	3.15s	47.33 %	581.3954	8.85s

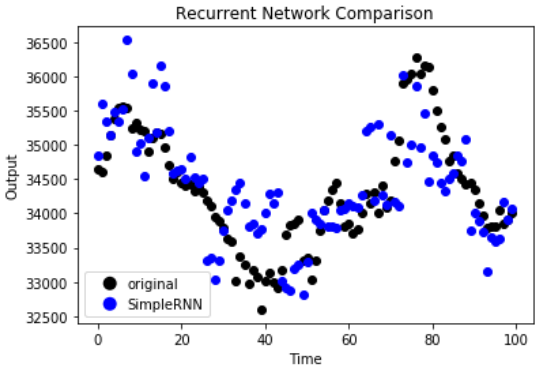
Table 19: Results for BSE Bank data prediction

Architecture	**1 Layered Model**			**6 Layered Model**		
	Accuracy	**RMSE**	**Time**	**Accuracy**	**RMSE**	**Time**
Simple RNN	51.00 %	816.6372	6.12s	59.00 %	734.2531	17.74s
LSTM	52.33 %	810.2784	6.67s	60.00 %	694.4727	18.03s
GRU	52.67 %	808.9842	7.89s	58.67 %	758.7057	18.47s
OptRNN	47.00 %	902.0924	3.82s	51.33 %	815.5708	10.73s

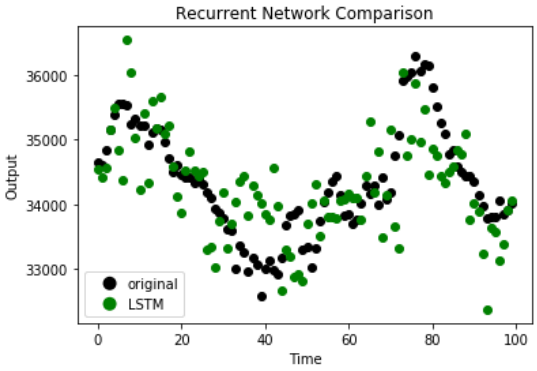
Recent values of Sensex, Small Cap and Bank index are approximately 35000, 20000 and 30,000 respectively. 20% margin is allowed for the accuracy during the predictions, which is equivalent to error measure of 700, 400 and 600 respectively. Table 17, Table 18 and Table 19 have the RMSE in line with the allowed margin.

5.4.4.3 Prediction Plots

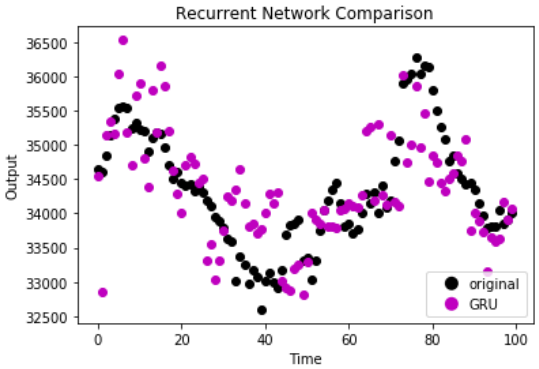
The prediction performance of the RNN models has been shown in Fig. 5.14, Fig. 5.15 and Fig. 5.16. The RNN implementation has shown the accuracy of the order of 55% to predict the stocks. It is comparable to the current benchmark performances [56].



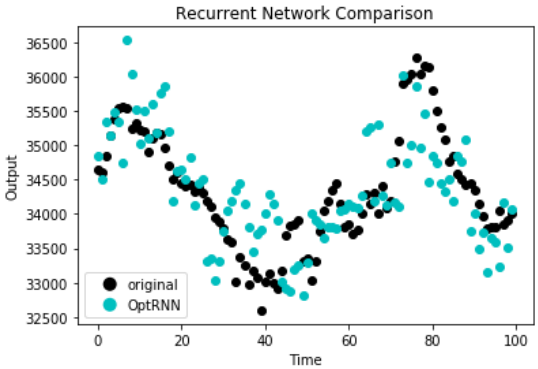
(a) SimpleRNN



(a) LSTM

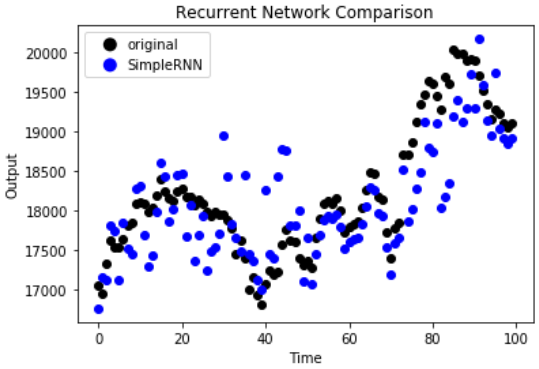


(a) GRU

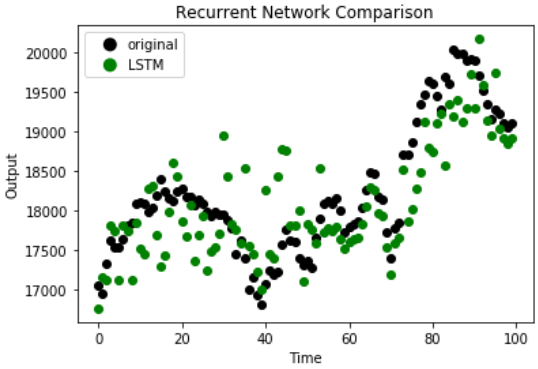


(a) OptRNN

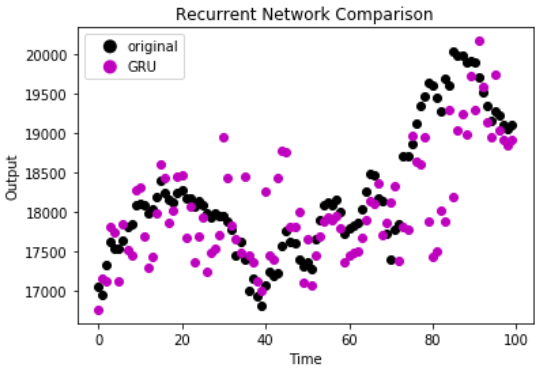
Figure 5.14: Prediction plots for BSE Sensex30 data



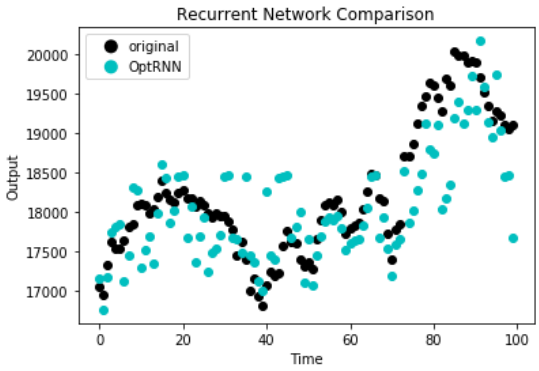
(a) SimpleRNN



(a) LSTM

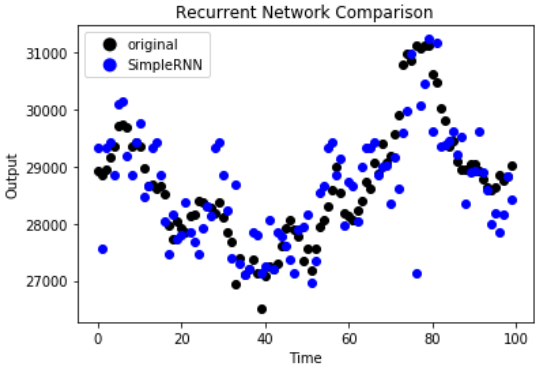


(a) GRU

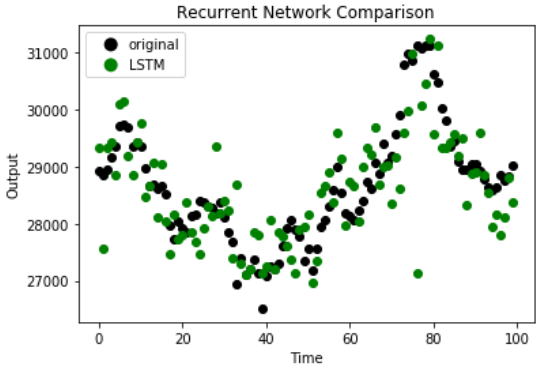


(a) OptRNN

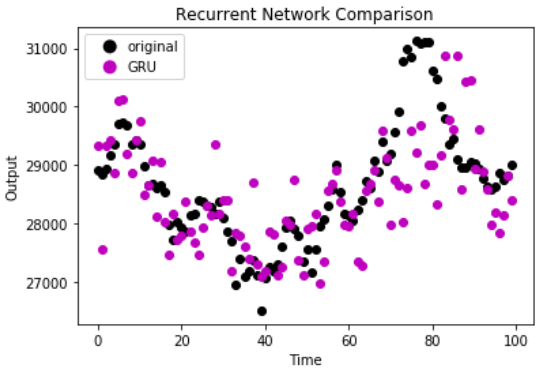
Figure 5.15: Prediction plots for BSE SmallCap data



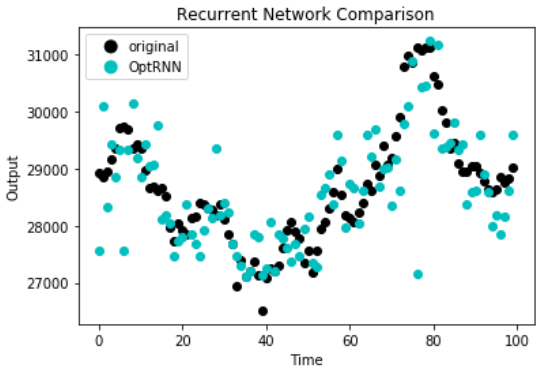
(a) SimpleRNN



(a) LSTM



(a) GRU



(a) OptRNN

Figure 5.16: Prediction plots for BSE Bank data

Fig. 5.14, Fig. 5.15 and Fig. 5.16 show the expected and predicted sequences of the stock market's data-streams. Deep variants of RNN depicted better performance than single layered implementations. OptRNN showed comparable performance to LSTM and GRU though it is 70.56% and 68.96% faster for single and multi-layered implementations.

6. CONCLUSION AND FUTURE WORK

A GA based meta-heuristic optimization strategy has been implemented to tune the hyper-parameters of CNN and RNN to recognize images and predict streaming data. The contributions, conclusions and the directions for future research have been mentioned in the following sections.

6.1 Thesis Contributions

This thesis work has furnished the following contributions:

- A meta-heuristic based search method has been proposed that quickly returns the optimal hyper-parameter combinations to train DNN variants such as – CNN and RNN for their specific applications.
- Meta-heuristic based attempts to optimize neural networks have already been made but most of them focused on tuning a single class of parameters in isolation, for example, the connection weights [14]. A method to tune multiple parameters in combination has been implemented. It helped understanding the internal working of DNNs which have mostly been assumed to be black-box learners.
- The proposed method is compared against the grid search method that exhaustively analyzes all the hyper-parameter combinations. The computation speed-up and performance has been analyzed.
- Apart from hyper-parameter tuning, another phase of optimization is implemented for RNNs that finds the subset of the data for quicker but near-optimal model training.
- CNN is deployed to recognize various image objects and RNN is deployed to predict the next value of stock market's data streams. Prediction performance and computation time has been analyzed.

6.2 Conclusions

Following observations have been drawn out of the implementation:

- The proposed method resulted equivalent combinations of hyper-parameters as compared to the grid search but it is 7 to 8 times faster.
- The prediction accuracies for the model trained with the hyper-parameters suggested by the proposed method are found to be comparable to the benchmark performance measures.
- The optimized version of RNN demonstrated significant improvements in training time with a slight reduction in accuracy. It is 74.34% faster than single layered LSTM architecture and 75.86% faster than the deep LSTM model. The decline in accuracy was 7.17% and 10.78% respectively.
- Optimal hyper-parameter suggestions for use-case 2 and use-case 4 involved more number of layers and processing nodes as compared to use-case 1 and use-case 3. It makes sense as the data used to model use-case 2 is more complex than the date used to model use-case 1, and the data for use-case 4 is more complex than the one for use-case 3.
- For use-case 4, the recurrent network was able to reflect upon the small details of various data-streams of Indian stock market. It was able to replicate the fluctuating nature of small capitalization stocks as they are more volatile compared to the Sensex or Bank benchmark index.

6.3 Future Scope

The future research plans include the following ideas:

- GA has been implemented to optimize DNN because of its superiority with hierarchal rule learning [53]. Other meta-heuristic techniques will also be explored for their applicability to optimize DNNs. The application specific parameters for example – price to earnings (PE) ratio, portfolio size, asset under management (AUM), net asset value (NAV), *etc.* for the stock market data-streams, will also be examined to further improve upon the optimization process.
- In the present implementation, prediction accuracy on the validation set is taken as the fitness score for Genetic Algorithm's operations. It is also planned to design better fitness measures for the GA implementation.

- Most of the hyper-parameters could be analyzed appropriately but the best choice for regularizer could not be identified very clearly. It has only two choices - L1 and L2 normalization. Ensemble of L1 and L2 could be tried out for better normalization performance of DNN models.
- Meta-heuristic search showed significant improvements in the rate of finding the optimal hyper-parameter combinations but during the hyper-parameter evaluation phase, CNN models have to be trained repeatedly with the entire dataset. Training on sliced dataset hampered the performance drastically. There is a need to find a balance of accuracy and training size reduction for CNN training.

REFERENCES

[1] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," Neural Computation, vol. 18, no. 7, pp. 1527-1554, 2006.

[2] I. Arel, C. Rose, and T. Karnowski, "Deep Machine Learning - A new frontier in artificial intelligence," IEEE Computational Intelligence Magazine, vol. 5, pp. 13-18, 2010.

[3] Y. Bengio, "Learning deep architectures for AI," In Foundation and Trends in Machine Learning, ACM, vol. 2, no. 1, pp.1-127, 2009.

[4] C. Szegedy, W. Liu, Y. Jia1, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[5] G. I. Diaz, A. Fokoue and G. Nannicini, "An effective algorithm for hyperparameter optimization of neural networks," IBM Journal of Research and Development, vol. 61, issue 4, 2017.

[6] J. Bergstra, R. Bardenet, Y. Bengio and B. Kegl, "Algorithms for hyper-parameter optimization," Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS), pp.2546-2554, 2011.

[7] L. Pasa and A. Sperduti, "Pre-training of Recurrent Neural Networks via Linear Autoencoders," Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS), 2014.

[8] K Weiss, T. M. Khoshgoftaar and D.D. Wang, "A survey of transfer learning," Journal of Big Data, Springer, 2016.

[9] I. Loshchilov and F. Hutter, "CMA-ES for hyperparameter optimization of deep neural networks," arXiv preprint arXiv:1604.07269, 2016.

[10] V. K. Ojha, A. Abraham and V. Snel, "Metaheuristic design of feedforward neural networks," Engineering Applications of Artificial Intelligence, vol. 60, pp. 97-116, 2017.

[11] Y. LeCun, Y. Bengio and G Hinton, "Review: Deep Learning," Nature Journal, 2015.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[13] R. Pascanu, T. Mikolov and Y. Mikolov, "On the difficulty of training recurrent neural networks," arXiv preprint arXiv:1211.5063, 2012.

[14] A. Johnson, "Common Problems in Hyperparameter Optimization," blog-post, March 2017. [Accessed on 5 June, 2018]

[15] S. Ioffe and C Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv preprint arXiv:1502.03167, 2015.

[16] Y. Lecun, "MNIST Data Repository: www.yann.lecun.com/exdb/mnist/" [Accessed on 5 June, 2018]

[17] L.A. Rastrigin, "The convergence of the random search method in the extremal control of a many parameter system". Automation and Remote Control. 24 (10): 1337–1342.

[18] M. A. Schumer and K. Steiglitz, "Adaptive step size random search," IEEE Transactions on Automatic Control. 13 (3): 270–276. doi:10.1109/tac.1968.1098903.

[19] J. Bergstra, R. Bardenet, Y. Bengio, Yoshua and B. Kegl, "Algorithms for hyper-parameter optimization," Advances in Neural Information Processing Systems, 2011.

[20] E. Polak, "Optimization : Algorithms and Consistent Approximations. Springer-Verlag. ISBN 0-387-94971-2.

[21] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal and B. Hodjat, "Evolving Deep Neural Networks," arXiv preprint arXiv:1703.00548, 2017.

[22] J. Dreo, J. P. Aumasson, W. Tfaili and P. Siarry, "Adaptive Learning Search, a new tool to help comprehending metaheuristics," International Journal on Artificial Intelligence Tools, vol. 16, no. 3, 2007.

[23] V Ojha, A Abraham and V Snasel, "Metaheuristic Design of Feedforward Neural Networks: A Review of Two Decades of Research," arXiv preprint arxiv:1705.05584, 2017.

[24] Y. Bengio, P Simard, and P Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, 5(2), 157–166, 1994.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] Y. Bengio, J. Chung and C. K., "Gated feedback recurrent neural networks," arXiv preprint arXiv:1502.02367, 2015.

[27] F. Dobslaw, "A Parameter-Tuning Framework for Metaheuristics Based on Design of Experiments and Artificial Neural Networks," Engineering and Technology International Journal of Computer and Information Engineering, vol.4, no.4, 2010.

[28] J. Schmidhuber, "Deep learning in neural networks: An overview". Neural Networks. no.61, pp.85–117, doi:10.1016/j.neunet.2014.09.003, 2015.

[29] M. Nielsen, "Neural Networks and Deep Learning," blog-post, March 2017. [Accessed on 5 June, 2018]

[30] A. Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1502.02367, 2015.

[31] YouTube Channel 'TheSemicolon' www.youtube.com/TheSemicolon [Accessed on 12 May, 2018]

[32] U. Anders and O. Korn, "Model selection in neural networks,"Elsevier, vol.12, Issue 2,pp.309-323, doi-10.1016/S0893-6080(98)00117-8, March 1999.

[33] Blog-post: Generalization - How many hidden units should I use? www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html [Accessed on 5 June, 2018]

[34] Blog-post: Activation Functions in Deep Learning www.delanover.com/activation-functions-in-deep-learning-sigmoid-relu-lrelu-prelu-rrelu-elu-softmax [Accessed on 30 May, 2018]

[35] A. Clevert, T. Unterthiner and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," arXiv preprint arXiv:1511.07289, 2015.

[36] G. Klambauer, T Unterthiner, A. Mayr, Andreas and S. Hochreiter, "Self-Normalizing Neural Networks," arXiv preprint arXiv:1706.02515, 2017.

[37] G. Papamakarios, "Comparison of Modern Stochastic Optimization Algorithms," Edinbuegh University Press, 2014.

[38] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arxiv:1609.04747, 2016.

[39] D. Kingma and L. B. Jimmy, "ADAM: A method for stochastic optimization," arXiv preprint arXiv:1412.6980v9, 2017.

[40] Blog-post: Difference between L1 and L2 regularization http://laid.delanover.com/difference-between-l1-and-l2-regularization-implementation-and-visualization-in-tensorflow/ [Accessed on 30 May, 2018]

[41] Blog-post: Differences between L1 and L2 as Loss Function and Regularization www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/ [Accessed on 30 May, 2018]

[42] Keras Documentation www.keras.io [Accessed on 25 May, 2018]

[43] Y. Kai, J. Lei, C. Yuqiang and X. Wei, "Deep learning: yesterday, today, and tomorrow," Journal of Computer Research and Development," vol. 50, no. 9, pp. 1799-1804, 2013.

[44] J. Schmidhuber, "Deep learning in neural networks: An overview. Neural Networks," vol. 61, pp. 85-117, 2015.

[45] I. H. Osman and G. Laporte, "Metaheuristics: a bibliography," Annals of Operations Research, vol. 63, no. 5, pp. 511-623, 1996.

[46] Beheshti, Z. and Shamsuddin, "A review of population-based meta-heuristic algorithms," International Journal of Advances in Soft Computing & Its Applications, vol. 5, no. 1, pp. 1-35, 2013.

[47] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67-82, 1997.

[48] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," Machine Learning, vol. 3, no. 2, pp. 95-99, 1988.

[49] J. Kennedy, "Particle Swarm Optimization," Springer, USA, pp. 760-766, 2010.

[50] F. Glover, "Tabu search-part I," ORSA Journal on Computing, vol. 1, no. 3, pp. 190-206, 1989.

[51] K. F. Man and K. S., Tang, "Genetic algorithms: Concepts and applications," IEEE Transactions on Industrial Electronics, vol. 43, no. 5, 1996.

[52] T. M. Mitchell, "Genetic Algorithms of Machine Learning," Chapter 9.

[53] T. M. Aguilar and J.S. Riquelme, "Evolutionary learning of hierarchical decision rules," IEEE Transaction on Systems, Man and Cybernetics B, vol. 32, no. 2, pp. 324–331, 2003.

[54] V Madhavan, E Conti, J Lehman, K Stanley and J Clune J, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training

Deep Neural Networks for Reinforcement Learning," arXiv preprint arXiv:1712.06567, 2017.

[55] S. K. Chaudhry and A. M, "Stream data management," Advances in Database system, vol. 30, 2005.

[56] B. Wamkaya, "ANN Model to Predict Stock Prices at Stock Exchange Markets," arXiv preprint arXiv:1511.07289, 2015.

[57] J. Bergstra, R. Bardenet, Y. Bengio and B. Kegl, "Algorithms for hyper-parameter optimization," Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS), pp.2546-2554, 2011.

[58] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization" (PDF). Journal of Machine Learning Research. 13: 281–305, 2012.

[59] L. Stewart and M. Stalzer, "Bayesian Optimization for Parameter Tuning of the XOR Neural Network," arXiv preprint arxiv:1709.07842, 2017.

[60] D. Maclaurin, D Duvenaud and R.P. Adams, "Gradient-based Hyperparameter Optimization through Reversible Learning," arXiv preprint arxiv:1709.07842, 2015.

[61] V. G. Gudise and G.K. Venayagamoorthy, "Comparison of particle swarm optimization and back propagation as training algorithms for neural networks," Proceedings of In Swarm Intelligence Symposium (SIS); pp. 110-117, 2006.

[62] H. F. Leung, H. K. Lam, S. H. Ling and P. K. Tam, "Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm," IEEE Transactions on Neural Networks, vol. 14, pp 79-88, 2003.

[63] J. Bullinaria and K. Alyahya, "Artificial bee colony training of neural networks," Nature Inspired Cooperative Strategies for Optimization (NICSO), pp. 191-201, 2013.

[64] L. Pasa and A. Sperduti, "Pre-training of Recurrent Neural Networks via Linear Autoencoders," Conference on Neural Information Processing Systems (NIPS), 2014.

[65] L. M. Rasdi, M.I. Fanany and A. M. Arymurthy, "Metaheuristic Algorithms for Convolution Neural Network," Computational Intelligence and Neuroscience, 2016.

[66] C. F. Juang, "A Hybrid of Genetic Algorithm and Particle Swarm Optimization for Recurrent Network Design," Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions vol. 34, no. 2, pp. 997-1006, 2004.

[67] M. Liu, J. Shi, C. Li and S. Liu, "Towards Better Analysis of Deep Convolutional Neural Networks," arXiv preprint arxiv:1709.07842, 2016.

[68] L. Golab and M. Oszu, "Issues in data stream management," ACM SIGMOD Record, vol. 32, no. 2, pp. 5–14, 2003.

[69] K. Li and J. Malik, "Learning to Optimize Neural Nets," Computing Research Repository (CoRR), 2017.

[70] Web-page: Data stream www.en.wikipedia.org/wiki/Data_stream [Accessed on 24 May, 2018]

[71] Web-page: Financial, Economic and Alternate Data www.quandl.com [Accessed on 28 May, 2018]

[72] Blog-post: Genetic algorithms: cool name & damn simple https://lethain.com/genetic-algorithms-cool-name-damn-simple/. [Accessed on 13 May, 2018]

LIST OF PUBLICATIONS

[1] Puneet Kumar and Shalini Batra, "Meta-heuristic Based Optimized Deep Neural Network for Streaming Data Prediction", in proceedings of IEEE International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Galgotia University, Noida, India, October 12-13 2018. **[Accepted]**

VIDEO PRESENTATION

The video presentation for the thesis has been uploaded to the link embedded below:
M.E. Thesis Presentation | Puneet Kumar (801632039) | ME CSE | TIET Patiala
(https://www.youtube.com/c/PuneetKumar1)