Department of Mathematics and Computer Science,
Data Mining Research Group

Efficient Learning in Deep Learning through Evolutionary Computations

2IMM00-Final Report

Sai Krishna Kalluri (1035631)

Supervisor:
dr. D.C. (Decebal) Mocanu

version 1.0

Eindhoven, January 2017

Contents

Chapter 1

Problem Statement

The word 'Intelligence' has a wide range of definitions spanning over many fields. However, for simplicity sake let us define intelligence as the model that maps a problem to it's solution. While many might disagree on this simple definition of intelligence, most would agree that intelligence is the key factor in the fitness function of survival. So crucial that we are not satisfied with the amount of natural intelligence we possess. Artificial Intelligence, a branch of computer science aims at making machines intelligent. However, it is far from simple to make machines intelligent. After all, it took evolution billions of years to produce human-level intelligence. In this project, we study two key subtopics of Artificial Intelligence namely Machine Learning and Evolutionary Computations. Later, we discuss the problems associated with training artificial neural networks. Finally, some research questions are formulated to mitigate these problems and advance in the direction of improving state of the art training mechanisms.

1.1 Machine Learning

Machine Learning aims at making machines learn appropriate functions from data. It is broadly divided into following three topics.

1. Supervised Learning [18], the objective of which is to build an approximate function that maps the known input and output data (labeled data) and use this knowledge to predict the outputs of unknown data. Supervised learning is broadly divided further into classification and regression problems.

2. Unsupervised Learning [18], the objective of which is to extract useful information and patterns from the data on its own. Some examples of the learning problems are clustering, density estimation, and dimensionality reduction.

3. Reinforcement Learning [17], the objective of which is to build an agent that takes optimal actions by studying the environment around it. Learning is obtained by navigating the search space through a series of actions and processing the feedback (reward) from the environment at each state.

1.2 Artificial Neural Networks (ANN's)

The first step in creating artificial intelligence is a mechanism or model to create and process knowledge from data. Artificial neural networks are mathematical models inspired by biological neural networks. ANN's have great success in all the three types of learning paradigms associated with machine learning described above, particularly in the field of deep learning [20] ranging from Computer Vision [20] to playing complex games like Chess and Go [5]. In the following sections, we explore different training mechanisms for training Artificial Neural Networks.

1.3 Back Propagation with Stochastic Gradient Descent

Back Propagation with Stochastic Gradient [6] is perhaps the most commonly used training mechanism for training neural networks. The applications in supervised learning particularly in pattern recognition have demonstrated its power for training Multi-layered Networks (MLN's). Despite its capabilities for training large-scale networks, two main criticisms have been raised commonly against Back Propagation. The first criticism is that it is computationally expensive to train networks using back-propagation. For example, it could take days to train state of the art deep learning models on ordinary workstations. The other criticism is the lack of guarantee on the reachability of global minimum. This is because back propagation internally uses gradient descent to optimize the weights, and gradient descent suffers from local minima problems [10]. Despite above criticisms, backpropagation is widely used in many applications, thanks to the advances in VLSI technology providing cheap and fast computational power, and the results of local minima in most cases are satisfactory enough. There are also concerns associated with the requirement of a large amount of training data, in order to build high-quality models.

1.4 Evolutionary Algorithms for training ANN

Evolutionary algorithms, a subset of evolutionary computations [7] is a population-based approach inspired by mechanics of nature, and are used widely for solving various optimization problems. Unlike gradient descent, the evolutionary algorithms don't suffer from the problem of getting stuck in the local minima. The downside is that similar to the process of evolution, the algorithms are slow and reaching global optimum can take a lot of time depending on the nature of the problem. However, these algorithms contain a lot of parallel components and thus the speed can be drastically improved using Graphical Processing Units (GPU's). Training ANN's can be thought of as an optimization problem where the weights and biases are the random variables, and the Mean Square Error (MSE) between expected and predicted output being the fitness function to be minimized. Depending on the nature of the problem a certain evolutionary algorithm works better than the rest. For example, ant colony optimization [1] works better for finding shortest routes, while algorithms like genetic algorithm [12] and particle swarm optimization algorithm [9] perform better in searching the search space for an absolute minimum or maximum. NeuroEvolution [16] techniques have proved successful in the field of reinforcement learning. However, less amount of research has been conducted in the areas of supervised and unsupervised learning. Neuroevolution techniques solve the problem of choosing a right number of hidden nodes. They also can optimize the number of links which overcomes the problem of sparsity in feedforward networks. Also, evolutionary algorithms have been proved successful in solving multi-objective optimization problems which in this case would help the neural network to evolve with respect to another kind of objectives and constraints like memory, adaptiveness to new data, constraints with respect to structural topology, e.t.c.

From the context discussed in above sections, the problem statement can be summarized into following broad research questions.

1.5 Research Questions

1. *Exploring training Artificial Neural Networks using evolutionary algorithms in-order to avoid local minima. This also concerns with assessing the performance comparisons with Back Propagation with Stochastic Gradient Descent (BPSGD) for varying amount of training data.*

2. *Improving Convergence time of training using evolutionary algorithms.*

3. *Exploring hybrid training mechanisms, a mixture of back-propagation and evolutionary algorithms to optimize the benefits of each in accordance with the nature of the problem.*

4. *Extending the analysis to optimize multi-objective functions, with the objectives being fitness score and memory consumption.*

Also, many more research questions can be formulated over the course of the thesis based on the experiments and accumulated knowledge.

1.6 Contributions

The research is aimed to improve the state of the art training mechanisms in Artificial Intelligence and is also expected to contribute to a top-tier machine learning journal article, perhaps even to a highly reputed multidisciplinary journal.

Chapter 2

Background

In this chapter, a brief background of Artificial Neural Networks, Back Propagation with Stochastic Gradient Descent and Genetic Algorithm are explained for the benefit of non-specialist readers.

2.1 Artificial Neural networks

Artificial Neural Networks are a network of artificial neurons inspired by neural connections in the brain. Figure 2.1 shows the general architecture of a feed-forward neural network. Inputs $[x_1, x_2, ..., x_k]$ denotes the input vector and $[y_1, y_2, ..., y_k]$ denotes the output vector. The objective of the neural network is to model the hidden function that maps the input data to the output data. Most of the times these functions are complex and therefore can't be represented by direct connection from inputs to outputs. Thus to incorporate more complex functions a series of one or more hidden layers are used between input and output layers. As shown in the figure 2.1 the outputs of each layer is connected to the inputs of next layer by links with weights (inspired by synaptic connections between neurons in the brain). Every neuron also has its own bias. The output of each neuron is an activation function applied to the weighted sum of inputs and its bias. This can be thought of as voting based on the opinions of self and all the neurons connected to it. The activation function is typically a logistic function that outputs a value between 0 and 1. The logistic activation function is shown in figure 2.2. Figure 2.3 shows various other activation functions used in neural networks depending on application. Now that we have established how a typical feedforward neural network looks like, in the next section we study how backpropagation with stochastic gradient can be used to train these networks.

2.2 Back Propagation with Stochastic Gradient Descent (BPSGD)

Before diving into how we can train neural networks using back propagation let's look closer into the mechanics of the network shown in figure 2.1. From the workings of the neural network described in the above section, we can compute the following outputs for a hidden node h_i as follows, where function f represents the activation function of the node. Thus the output of node h_i is computed as follows.

$$h_i = f(\sum_{i=1}^{i=K} w_{ki} \cdot x_k)$$

In similar way the final output y_i is computed as:

$$y_j = f(\sum_{j=1}^{j=N} w_{ij} \cdot h_i)$$

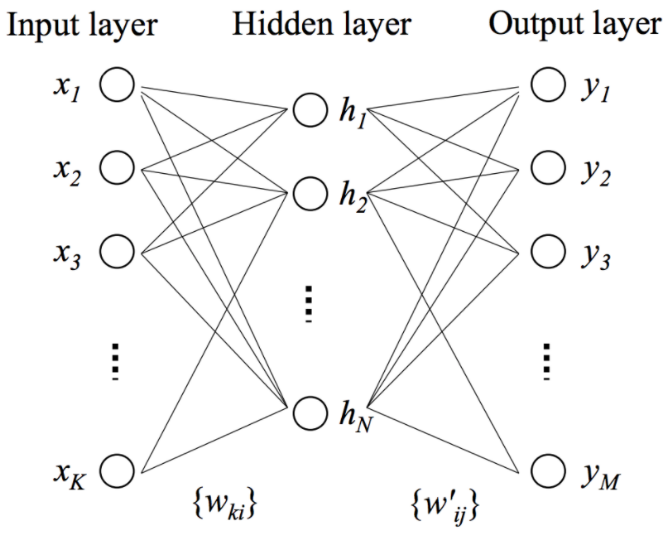


Figure 2.1: Neural Network Architecture of a typical feed forward network.

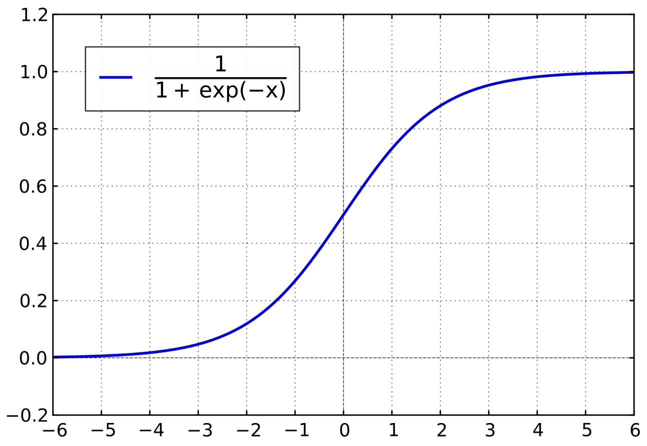


Figure 2.2: Logistic Function.

This way of computing the outputs for a given input and weight configuration is called forward propagation. In order to train the network, we have to ensure that the outputs of the network match the target outputs. This can be achieved by minimizing the Mean Square Error (MSE) between the outputs of the network and the target data. This can be done by propagating this error backwards. This part of propagating the error backward and adjusting the weights accordingly is called back-propagation.

Back Propagation with Stochastic Gradient Descent (BPSGD) is the process of minimizing the error by adjusting the weights such that they move in the direction of the lowest gradient. Therefore for every weight w_k first the back propagation error is calculated by computing the partial derivative of the total error with respect to the weight. The weight is then updated in each iteration by subtracting the error multiplied by a suitable learning rate (η). The above process can be represented in a mathematical form as follows.

$$w_k^{(+)} = w_k - \eta \cdot \frac{\partial E_{total}}{\partial w_k}$$

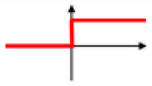
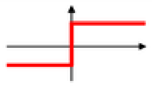
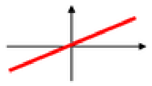
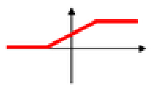
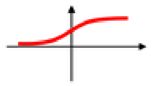
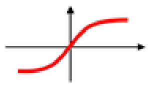
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Figure 2.3: Various activation functions used in neural networks.

The partial derivatives are propagated in a chain like fashion from outputs to all the way to inputs modifying every weight in each iteration. The training process saturates when the gradient approaches zero. Thus in this process, the network is trained when the back propagation error becomes zero. However, this does not always ensure that the weights at the end is a global minimum. We can clearly see from the above approach, how BPSGD can get stuck in local minima. Also notice that since the errors are propagated in a chain like fashion, as the error propagates, it becomes less and less.Thus this error can vanish in large networks as it reaches the layers near inputs, making the training process slow. This problem is called vanishing gradient problem. Also in case of small training data (small errors propagating backward, hence fewer adjustments near inputs), the above problem causes the network to get stuck in local minima (depending on the initial random state).

2.3 Evolutionary Algorithms

Evolutionary algorithms are algorithms inspired from nature and are commonly used for solving NP-hard optimization problems. There are many evolutionary algorithms inspired by nature. However, in this section, only a discussion regarding genetic algorithm and particle swarm optimization algorithm are discussed.

2.3.1 Genetic Algorithm

Genetic Algorithm is an evolutionary algorithm inspired by natural selection process of 'survival of the fittest'. Figure 2.4 shows the general state flow of a genetic algorithm. The process starts with a pool of population evaluated against a fitness function. Based on the results, a selection operation is performed on the population. This selection typically consists of selecting the elite population that continues to be part of the next generation and also selecting the population and parents that undergo crossover to create next generation children. Crossover operation takes care of how the crossover is performed between a given parents. Mutation operator ensures that there is a diversity of the population as the generations progress, and typically concerns with aspects like what percent of population undergo a mutant transformation, and to what degree. Based on the above operations new population is created and the process repeats as long as the termination condition is reached. This condition can be from reaching a satisfactory fitness value to reaching maximum generations. Configuration parameters play an important role in exploring the search space particularly when the search space is huge. For example, high crossover rate makes the algorithm stuck in local minima, while high mutation rate causes the algorithm take a large time to converge. Population size also plays a huge role in determining the quality of the solution. To counter these problems various dynamic parameter tuning and parameter control integrated into the traditional genetic algorithm.

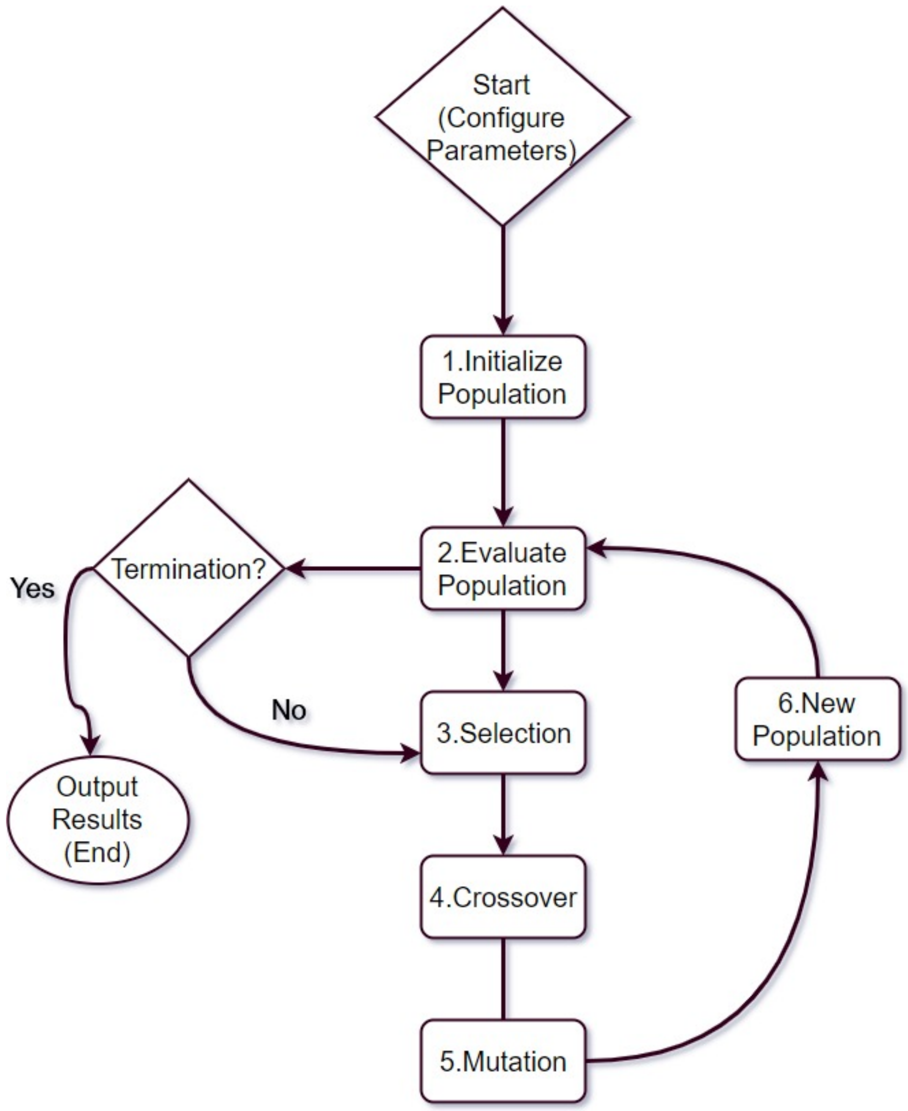


Figure 2.4: General state flow of a genetic algorithm

2.3.2 Particle Swarm Optimization Algorithm

Particle Swarm optimization Algorithm similar to the genetic algorithm is a nature-inspired population-based optimization method commonly used for solving NP-hard optimization problems. This algorithm is inspired by the social behavior of swarms in the process of reaching their goal. The state flow diagram of a general PSO is shown in the figure 2.5. The process starts with the creation of a population of particles with random positions and random velocities. The fitness of a particle is determined as a function of its current position. In each iteration, the velocity of each particle is updated as a weighted sum of the differences between its personal best position, global best position of the swarm to its current position. The new position is then computed from its new velocity and current position. In other words the at every iteration the particle moves to a new position based on its personal exploration results and exploiting the collective intelligence of the swarm. The process is repeated until a termination condition is reached.

2.4 Neuro Evolution of Augmented Topologies (NEAT)

Due to inadequate time the background of NEAT is left for self studying of the reader. The required background information can be found at [16].

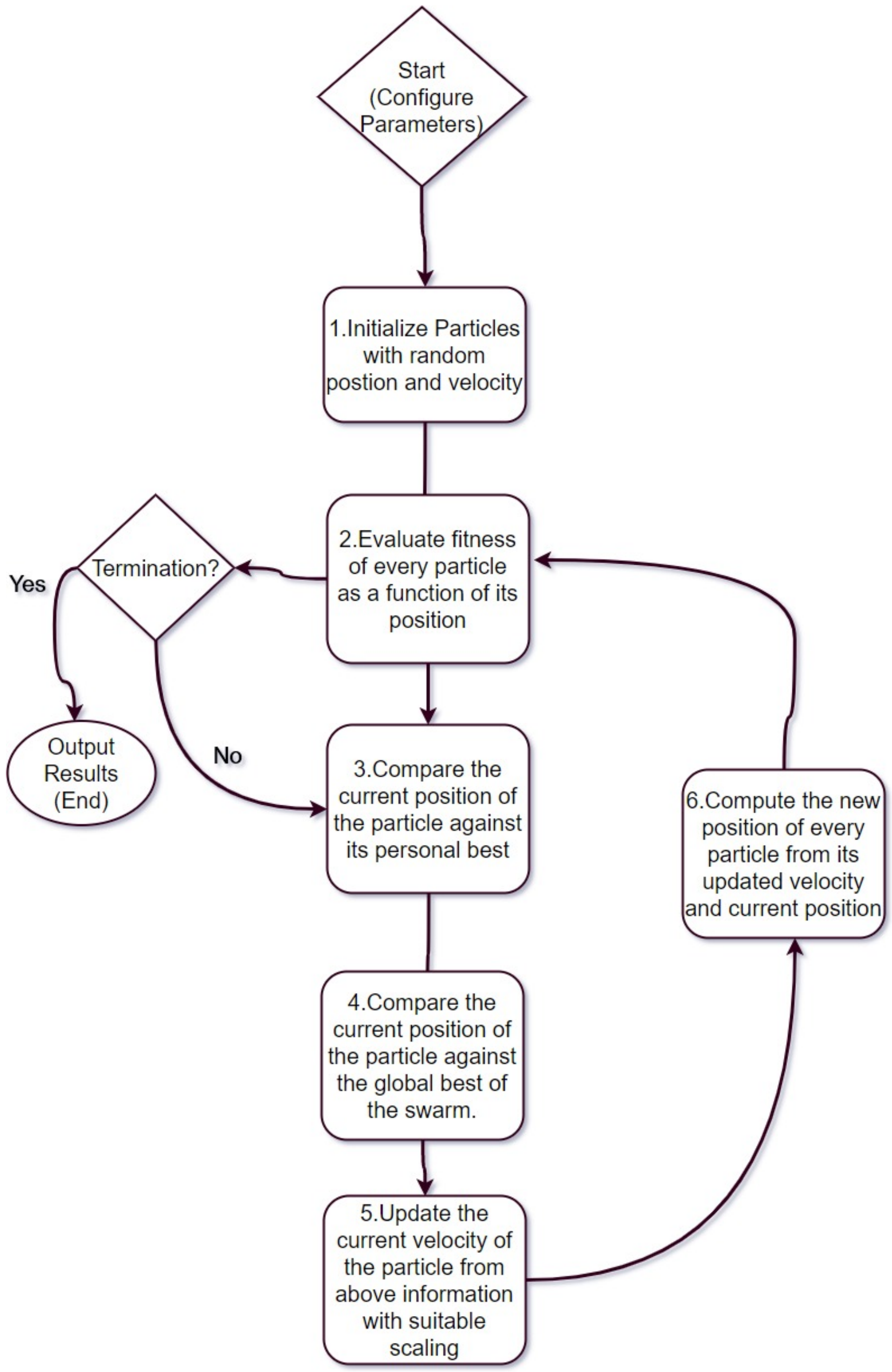


Figure 2.5: General state flow of a Particle Swarm Optimization algorithm

Chapter 3

Literature Study

The following literature study is performed to explore the solutions to the research questions mentioned in section 1.5. Prior to exploring the strategies of training that avoid local minima, we need to first explore the cases where BPSGD can get stuck in local minima. A general analysis has been shown in [4] concerning the local minima in MLN's. However, the analysis is performed under the hypothesis that the neurons are linear. This hypothesis cannot be generalized to architectures consisting of nonlinear neurons. An interesting analysis of the conditions ensuring optimal learning along with some examples of local minima for architectures involving non-linear neurons is presented in [10]. It has been shown, how for a simple case of XOR training, BPSGD can get stuck in local minima. In this particular case, the gradient approaches zero while the error is not null as shown in Figure 4 in [10]. It has also been proven that a better solution (close to null) exists. The experiment is replicated and the results are discussed in section 5.1. Now that we have established a case where back propagation fails we need explore if training using an evolutionary algorithm can solve this problem. Genetic Algorithm has been [12] proved as a successful technique in optimizing many problems. In [11] genetic algorithm has been used to train a neural network to forecast stock prices. The results proved to shown high precision compared to the statistical forecasting model Auto-Regressive Moving Average(ARMA) model. In [3] genetic algorithm proved to be better in the diagnosis of breast cancer compared to back-propagation. In [13] a general method for training feedforward networks is described. Based on the above study genetic algorithm is used to train the feedforward network for XOR data. Unlike BPSGD training with GA achieved optimal results and the results are discussed in section 5.2.

Feed-forward networks also suffer from the problem of multiple sparse connections between layers. Also finding the right number of hidden nodes has a great impact on the performance of the network. A large number of hidden layers even though might produce less training error, would suffer from the problem of generalizing to new data. Thus optimizing the topology of the network would result in great benefits both with respect to cost (memory requirements) as well as performance. Topology and Weight Evolving Artificial Neural Networks (TWEANNs) solve this problem by finding the optimum topology and weights. In [16] a neuroevolution method called NeuroEvolution of Augmenting Topologies (NEAT) is proposed which solves this problem efficiently. The speciation process described in section 2.3 in [16] ensures that the population consists of multiple species. This information can also be exploited in formulating a multi-objective optimization problem where we can optimize the memory requirements in addition to performance. This is particularly useful when we are planning to embed the network in a small device sacrificing some performance.

Hybrid models, in general, offer great advantages in exploiting the advantages and minimizing the disadvantages of models that are mixed. In [16] only genetic algorithm is used to optimize the structure of the topology and weights. This makes the algorithm slow since a lot of variables need to be optimized. A possible research direction would be to integrate back propagation between generations to improve the local search between generations. This is feasible since the topologies are directed and acyclic, and the derivatives of activation functions can be computed. In [15] similar approach is used to evolve deep neural networks and the results produced a 5 percent improvement which is difficult to discover by hand engineered structures as described in section 4.2 of the article. Choosing evolutionary algorithms for training comes with a lot of tunable parameters and poor parameterization can hinder the quality of the results. These parameters also need to be modifed dyanamically to result in better training. Adaptive praemter control is a very famous approach for controlling the paramters run time. In [2] various strategies for evaluating, estimating and generating new parameters are discussed. Many of those principles in combination with network analysis can aid in the training of artificial neural networks.

3.1 Strategies

Following strategies are being considered in order to explore and solve the research questions. The strategies are ordered in accordance with the research questions mentioned in section 1.5.

1. The first step in solving this problem would be to identify and study the problems where training with BPSGD gets stuck in local minima. The next step would then be to evaluate the results when trained with evolutionary algorithms. Benchmarking the results on standard datasets.

2. While the above research question concerns with the quality of the solution, this research question concerns with the computation time involved in finding the solution. The approach is to benchmark the results on various datasets and come up with an analysis on how computation time varies with the nature of the problem.

3. The objective of this research question is to explore the possibilities of combining both training mechanisms to train the network efficiently. The approach is to extract features from data and user in order to direct the flow of training efficiently.

4. Evolutionary Algorithms can offer a wide variety of solutions for the same problem. The objective is to improve the algorithm NEAT(NeuroEvolution of Augmented Topologies) introduced in [16] with the analysis performed in the above research questions to obtain a Pareto plot of performance vs memory. The results of speciation described in section 2.3 in [16] can be exploited to create these plots.

3.2 Road Map

A broad high-level roadmap of various tasks associated with the project is shown in table 3.1. This might change in future in order to accommodate new ideas explored during the course of the project.

Task	**Time Frame**	**Start Date**	**End Date**
Literature Survey	2 months	12/01/17	01/31/18
Basic Implementation (Seminar Report Submission)	2 months	12/01/17	01/31/18
Deeper Exploration and Implementation (from the objectives of seminar)	4 months	02/01/18	05/01/18
Evaluation of Results and Conclusions	4 months	02/01/18	05/01/18
Thesis Report	1 month	05/01/18	06/01/18
Extra spare month for Improvements and backlogs	1 month	06/01/18	07/01/18

Table 3.1: High Level Road Map for completing the project

Chapter 4

Methodology

Before jumping into the exact implementation methods, it is important to understand the characteristics of a good model. A good model should be as small as possible besides having high-quality metrics. Consider an example of simple multiplication, which takes an input and multiplier and generates an output. Training the data of this function on a large network can generate good accuracy, however, this network can suffer from generalization problems, particularly if the training data is small. Also, this could easily generate many sparse weights which only is a wastage of memory. Besides, the training time can be large and the probability of getting stuck in local minima is high. Due to the above reasons, it is important to have the model as small as possible. In other words, it is important that the model should have an optimal topology in addition to optimal quality metrics.

With the above objective in mind, let us move on to review the training process used in this project. Figure 4.1 shows the high-level overview of a training process used in this project.

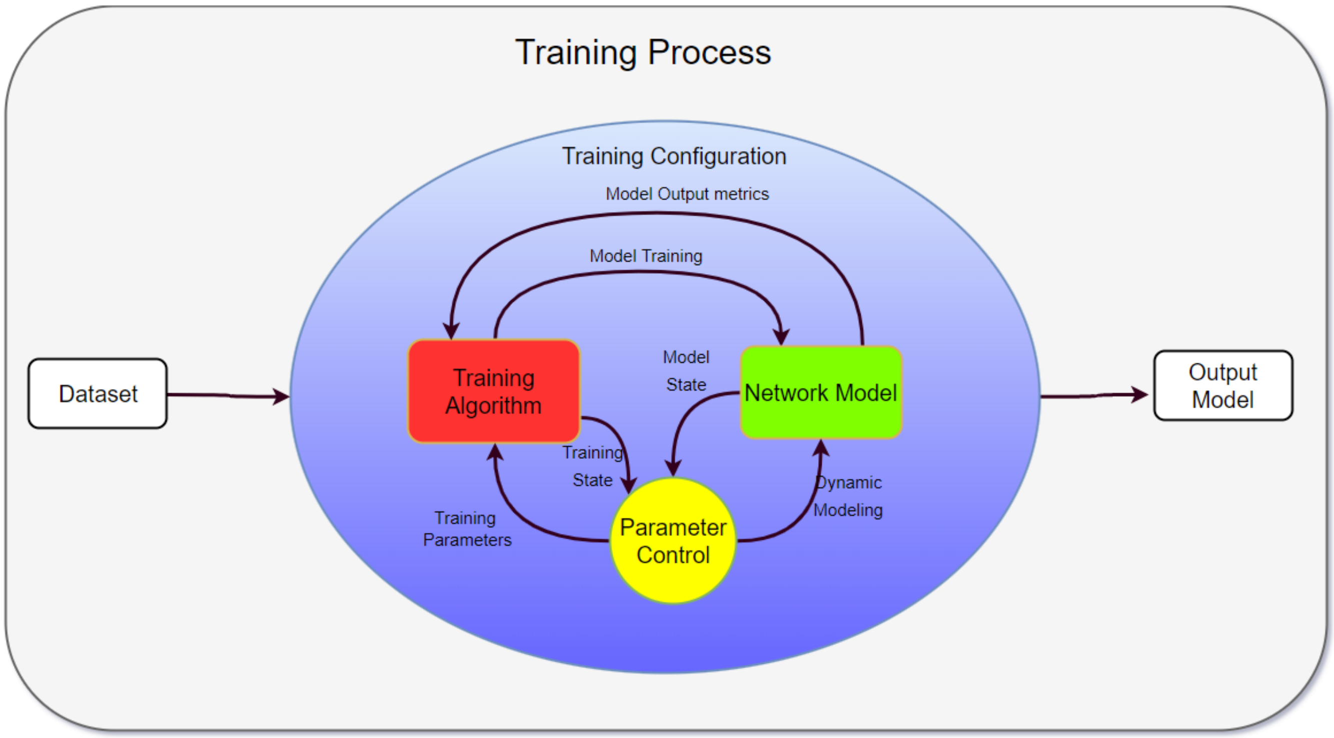


Figure 4.1: Training Process Overview

The process at high level consists of 3 components. The dataset, the training configuration and the output model. Dataset component contains the data to be modeled, while the output model is the final output model that is generated by the training process. The key component obviously is the training configuration. The training configuration is again broadly divided into 3 components. The first component is the training algorithm, the second is the network model and the last component is the parameter control component. The arrows between the components show the interactions between them. These interactions control the training process and output the final model when the termination conditions are satisfied. Following sections describe a brief overview of each component and their interaction with other components.

4.1 Training Algorithm

As the name indicates, this component consists the training algorithm that is responsible for training the network model. Some of the key algorithms that are used so far in this project are Back Propagation with Stochastic Gradient Descent (BPSGD), Particle Swarm Optimization Algorithm (PSO) and Genetic Algorithm (GA). The other algorithm that is also used in this project is NEAT. However, NEAT is more like a configuration than a training algorithm since it inherently takes care of evolving the topology in addition to the weights. Other potential algorithms will be investigated and incorporated in course of time. The key interactions of this component, as shown in the figure 4.1 are training the model and collecting the quality metrics in each iteration. Communicating the state information to the parameter control component and updating the training parameters in return. These interactions with parameter control component can vary from changing the parameters like crossover rate or mutation rate in case of Genetic Algorithm to completely changing the training Algorithm to a new one (Hybrid Training).

4.2 Network Model

A model necessarily doesn't have to be a neural network. However, the scope of this project is limited to training neural networks. Hence, the name of the component 'Network Model' follows. Now, that we determined that the model is a neural network, this component consists of various kinds of network topologies. Following are some of the initial ideas on the topologies. However, in course of time, more topologies will be considered.

4.3 Topologies

Topology plays can play an important role in determining the quality of learning of a neural network. Following topologies are used in the experiments.

4.3.1 Feed Forward Network Topology

Feed Forward Network Topology is perhaps the most common topology used for training an Artifical Neural Network. In this topology, the neurons are organized in layers. Every neuron in a layer is completely connected to the neurons in the previous and next layer. The drawback, however of this topology is that it suffers from many sparse connections particularly in case of large networks. Also, the topology is fixed before training starts resulting in converging in local minima.

4.3.2 Modified Feed Forward Network

This topology is inspired by topology and weight evolving artificial neural networks (TWEANN). In this project, two different cases of topologies are considered.

Simple Case

The first one is a simple case in the sense, every possible edge between a source and destination exists as long as the destination node has equal (bias) or higher number than the source node. Thus the graph is directed (data flows from input to output) and acyclic (no feedback connections). A sample figure of complete connections of a 5 node graph is shown in figure 4.5. The nodes are divided into 3 different classes namely, Input Nodes, hidden nodes and Output Nodes. Input Nodes take one input at a time and pass it to all the connected nodes without applying any activation function. The output nodes do not apply any activation function and provide raw output. The function of nodes in hidden layers are similar to the ones in the feedforward network. The objective is to optimize the weights of all these connections. Clearly, we can notice in this case we have more number of connections than in the case of feedforward networks. Thus the number of sparse connections will be even more than in the case of feedforward connections. However, the advantage of having all possible connections, in some cases can result in better performance. Also, since there are all possible connections, the number of hidden nodes to model the data is considerably less than the number required by feedforward network effectively reducing the total number of links in the network.

Complex Case

The second case is little complicated in the sense, here we allow the algorithm to not only optimize weights between nodes but also optimize the presence of nodes and links as well. Thus the problem turns into a mixed integer optimization problem. This topology reduces the number of sparse connections by optimizing the presence or absence of a link. However, the disadvantage is that now the number of parameters to optimize are increased and this might result in more training time. Also, being a mixed integer optimization problem, the probability of getting stuck in local minima is high compared to the simple case.

4.3.3 Probabilistic Networks

Inspired from the probabilistic existence of a particle in quantum mechanics, the idea of probabilistic networks is to modify the structure of the topology in a continuous fashion based on a probability parameter. These topologies aim at mitigating the problems of Simple and Complex topologies discussed above. While complex case optimizes the presence or absence of a link or a node in a discrete fashion, since being a mixed integer problem, the combinatorial optimization is more close to brute force search, particularly for larger networks. This discrete presence or absence of a link or anode can be converted into continuous domain by adding an additional transfer function with a probability parameter. This existence value of a link for a given probability p is calculated as follows :

$$ExistanceValue = \frac{1}{(1 + e^{-A*(p-0.5)})}$$

where A controls the smoothness of the distribution. Figure 4.2 shows the influence of smoothness factor. High values of A (red line) force the network to behave like a Complex case while low values (blue line) of A are similar to a simple case.

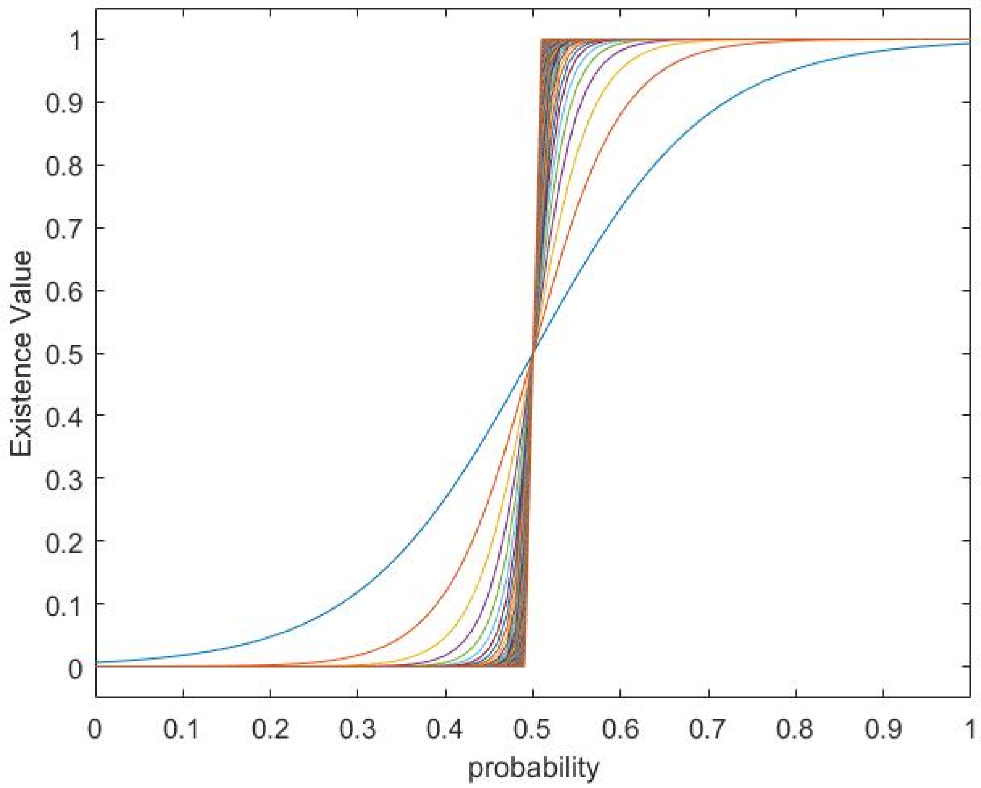


Figure 4.2: Existance Value vs probability for varying smoothness parameter A

The disadvantage of this method is however, it introduces additional non-linearity in addition to the activation function applied on output. This might result in more training time. However, with appropriate parameter control and dynamic bounds on the probability values, these networks are expected to model the data better without getting stuck in local minima.

Activation function also plays a key role in determining the quality of the training. Evolving activation function in addition to the topology is also being explored.

4.4 Parameter Control

As we can observe from the study of above components, the training process consists of a lot of tunable parameters like crossover rate, Population size, weight bounds, link and node probabilities e.t.c. A poor parameterization can hinder the quality of training. Parameter Control component tries to overcome this problem by dynamically re-configuring these parameters run-time based on the training state. Stateflow of the high level functioning of parameter control component is shown in the figure 4.3. This Stateflow is inspired by the work done on adaptive parameter control by Aldeidi Aleti as part of his P.h.d thesis [2]. The first step in adaptive parameter control is to collect the feedback from the other two components. This feedback typically consists of training quality metrics, population diversity, graph statistics of the network e.t.c. Following the feedback, the next operation is to determine the effect of parameter values. This can involve observing the change in the state of the system for existing parameter values. These can range from fitness improvement of the best parent to change in overall mean fitness of the population. The third component is to attribute the quality to the parameter values. This is done from the observations of the assessment values of the previous component over a period of time. The quality can be attributed by observing the overall positive or negative change in the system. The next step is to decide whether to exploit or explore the parameter space. Exploration involves generating uncharted parameter values and obtaining quality attributes over subsequent generations. Exploitation involves choosing the parameter values from the current knowledge to improve the training performance. Finally, the new set of parameter values are generated based on the above decisions and corresponding components are addressed to accommodate the change. The final generation can be done using simple fuzzy logic rules to complex reinforcement learning strategies like Dynamic Multi-arm Bandits. Currently, parameter control is under investigation for determining effective strategies and metrics for each function. While a great amount of work is done in [2] on exploring effective strategies for above components, it is still to be explored, if they indeed suited to the existing conditions, and if not what modifications are necessary.

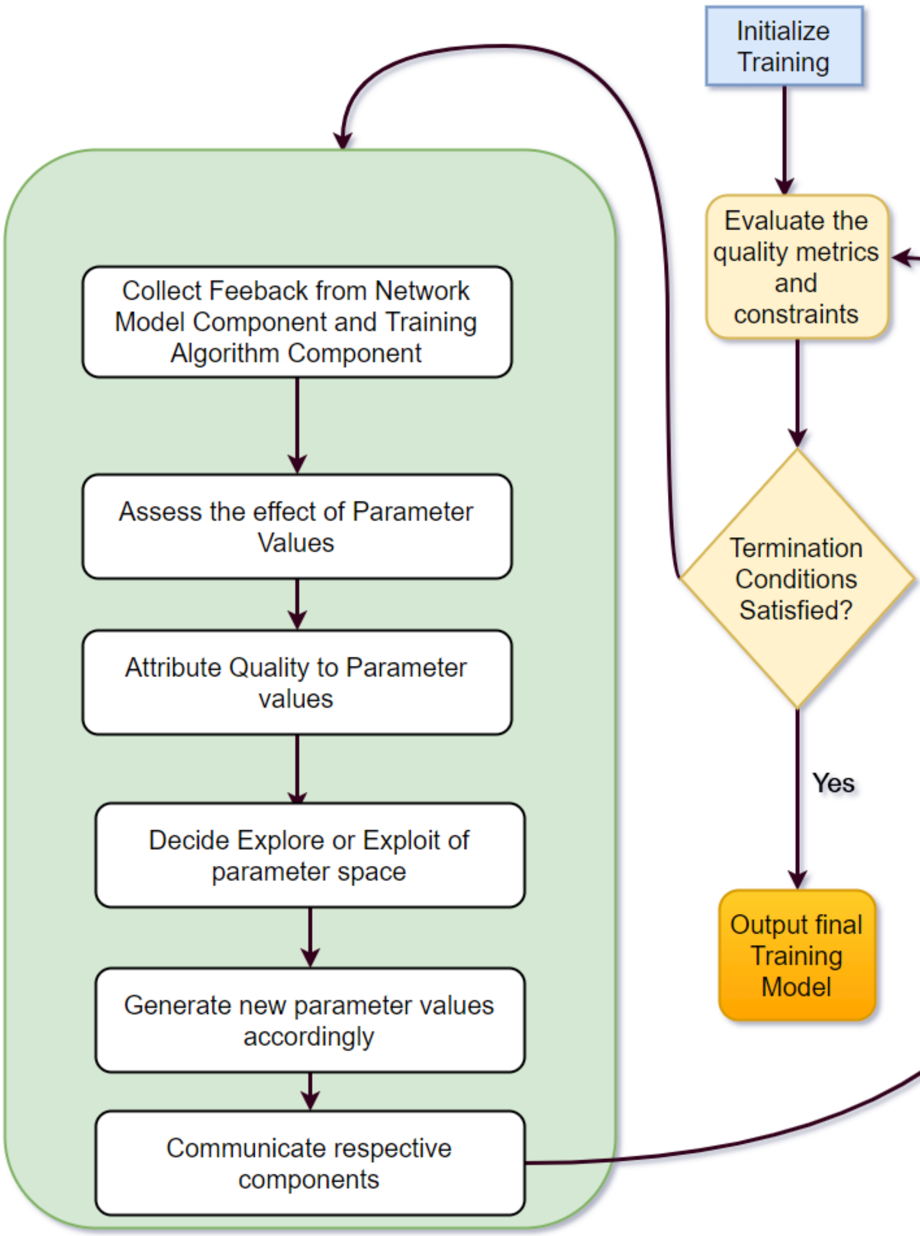

Initialize Training
Evaluate the quality metrics and constraints
Termination Conditions Satisfied?
Yes
Output final Training Model
Collect Feeback from Network Model Component and Training Algorithm Component
Assess the effect of Parameter Values
Attribute Quality to Parameter values
Decide Explore or Exploit of parameter space
Generate new parameter values accordingly
Communicate respective components

Figure 4.3: Stateflow of high level functioning of parameter control component

Chapter 5

Preliminary Evaluation and Results

In order to understand and demonstrate the strengths and limitations of training using back propagation and evolutionary algorithms (Genetic Algorithm in this case), following preliminary studies have been made. XOR data [10] is used to train ANN with the architecture [10] shown in Figure 5.1. The data is shown in Table 5.1

5.1 Back Propagation with Stochastic Gradient Descent

The results for Mean Squared Error (MSE) and gradient are as shown in Figure 5.2 and Figure 5.3 respectively. We can observe from the figures while the gradient becomes zero, the error is still at 1.2. We demonstrate in the next section that for the given network configuration this is indeed a local minima and a better solution can be found.

5.2 Genetic Algorithm

Following the experiments, with back-propagation, the same network is trained using a genetic algorithm. The results for Mean Squared Error over generations is shown in figure 5.4. We can observe from the figure training with genetic algorithm managed to find the optimal solution for the problem. However, as described in Section 1.4 evolutionary algorithms consume a lot of time in finding optimal solutions. To demonstrate this drawback of evolutionary algorithms further experiments are made.

5.2.1 Modified Feed Forward Networks

In these experiments, the behavior of two different cases of modified feedforward networks is studied.

Input1	**Input2**	**Output**
1	0	1
0	0	0
1	1	0
0	1	1
0.5	0.5	0

Table 5.1: XOR training data

Simple Case

The first one is a simple case in the sense, every possible edge between a source and destinations exists as long as the destination node has equal (bias) or higher number than the source node. Thus the graph is directed (data flows from input to output) and acyclic (no feedback connections) with the exception of biases. A sample figure of complete connections of a 5 node graph is shown in figure 5.5. However, the nodes are divided into 3 different classes namely, Input Nodes, Hidden Nodes and Output Nodes. Input Nodes take one input at a time and pass it to all the connected nodes without applying any activation function. The output nodes also don't apply any activation function and provide raw output. The function of nodes in hidden layers are similar to the ones in the feedforward network. The objective is to optimize the weights of all these connections. The results for 4 and 5 nodes are shown in the figure 5.6 and 5.8 respectively. The corresponding graphs are shown in figure 5.7 and 5.9 respectively. We can observe the optimal solution (best fitness in the graph) is found very fast (less than 5 generations) in both cases.

Complex case

The second case is little complicated in the sense, here we allow the algorithm to not only optimize weights between nodes but also optimize the presence of nodes and links as well. Thus the problem turns into a mixed integer optimization problem. In this case, the algorithm couldn't be able to find an optimal solution in 50 generations. While the optimal solution can be eventually found with high population count and more generations, the search is more close to a greedy search due to the nature of the problem. However, still, the solution is better compared to the backpropagation. The result is shown in the figure 5.10.

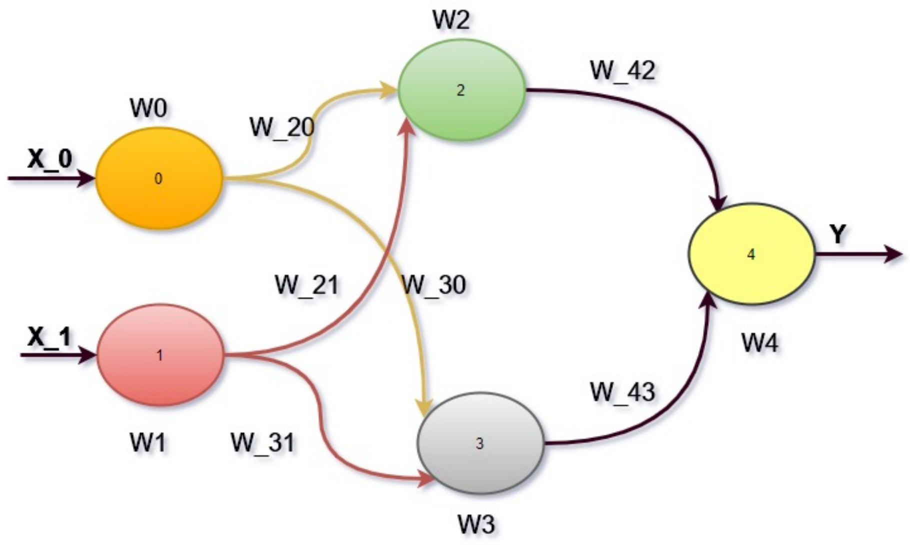


Figure 5.1: Neural Network Architecture used for training XOR data

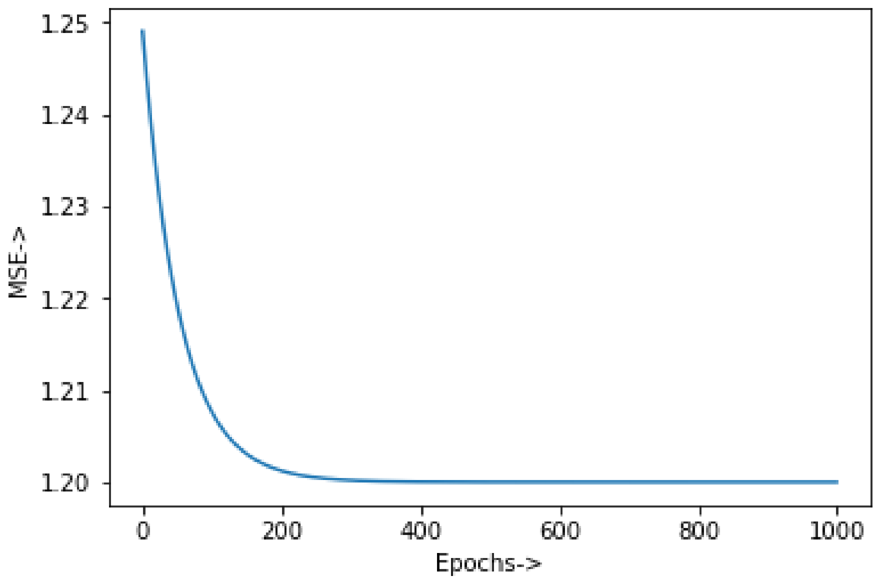


Figure 5.2: Mean Squared Error plot for training with BPSGD

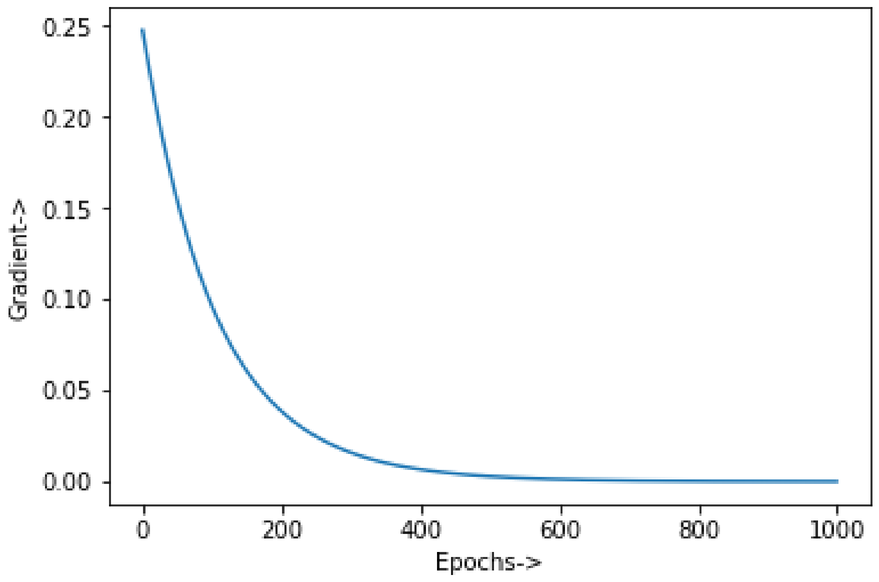


Figure 5.3: Gradient for training with BPSGD

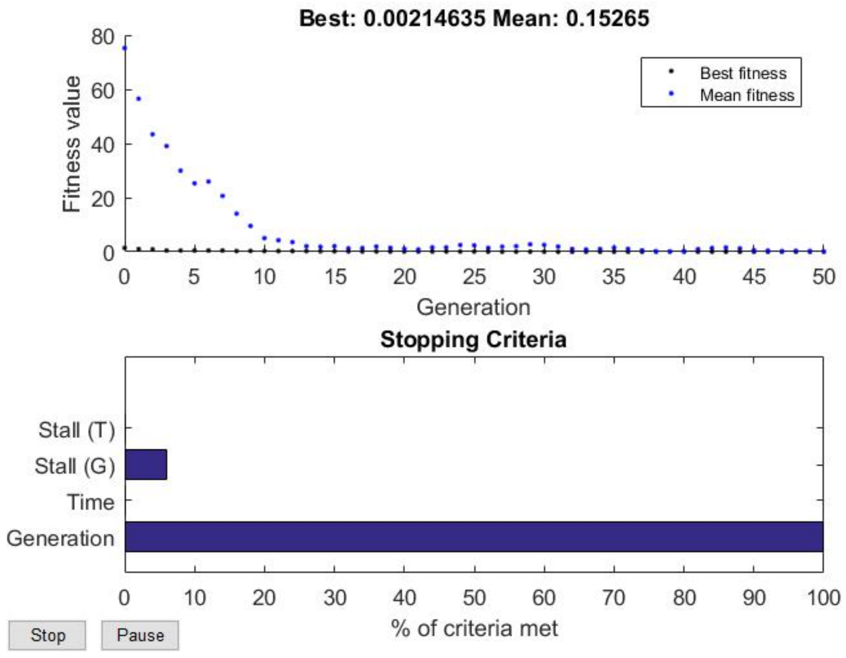


Figure 5.4: Fitness Plot for training with Genetic Algorithm (feed forward network)

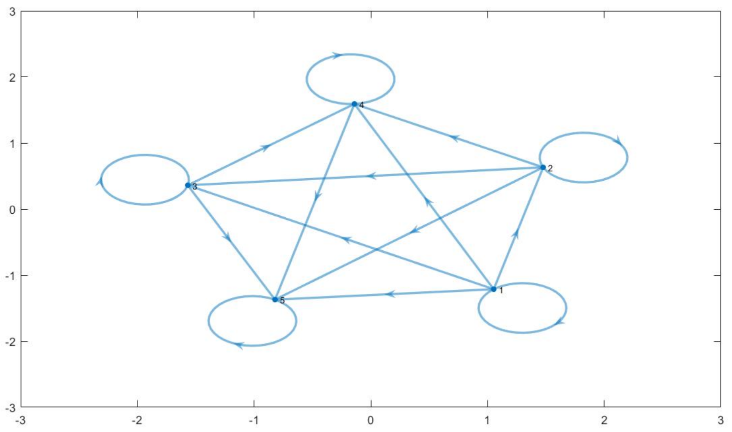


Figure 5.5: Fully Connected directed 5 node graph

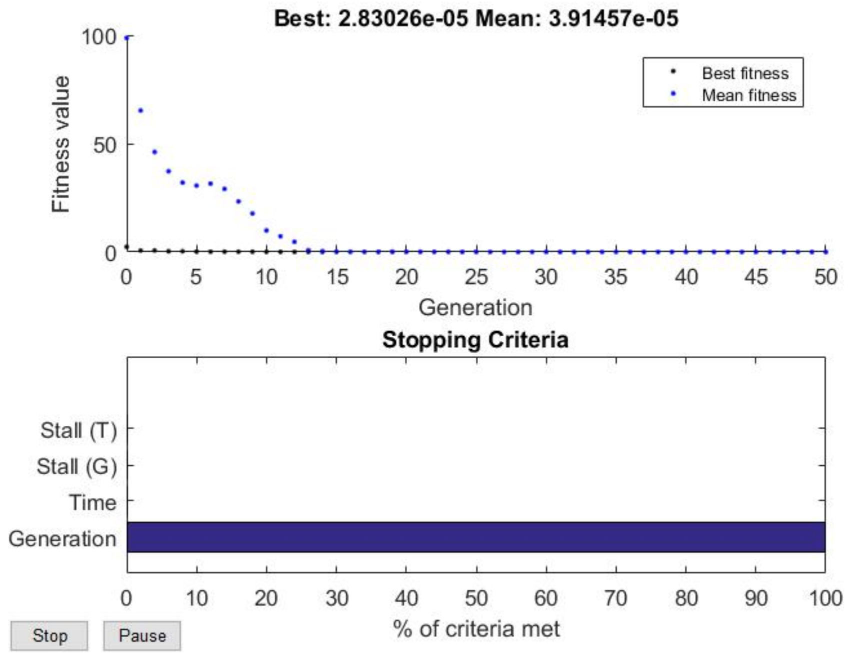


Figure 5.6: Fitness Plot for training with Genetic Algorithm for 4 nodes(simple case)

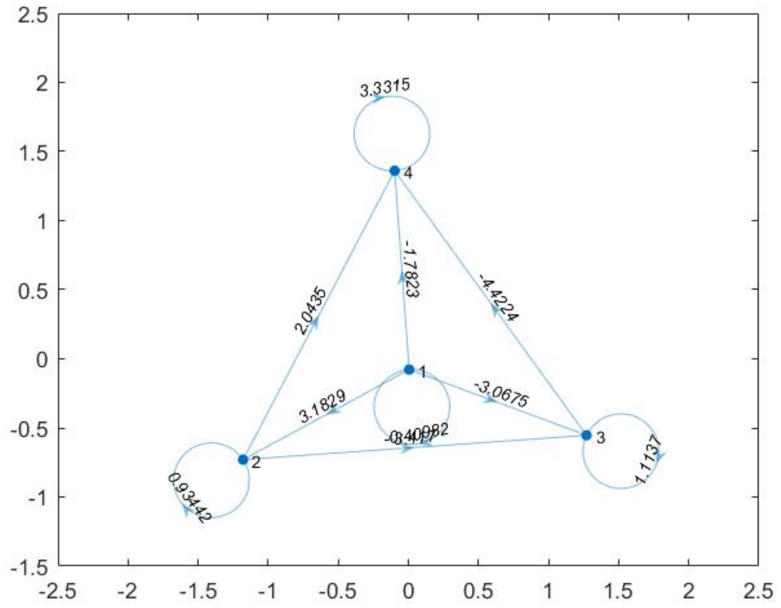


Figure 5.7: Network Graph for training with Genetic Algorithm for 4 nodes(simple case)

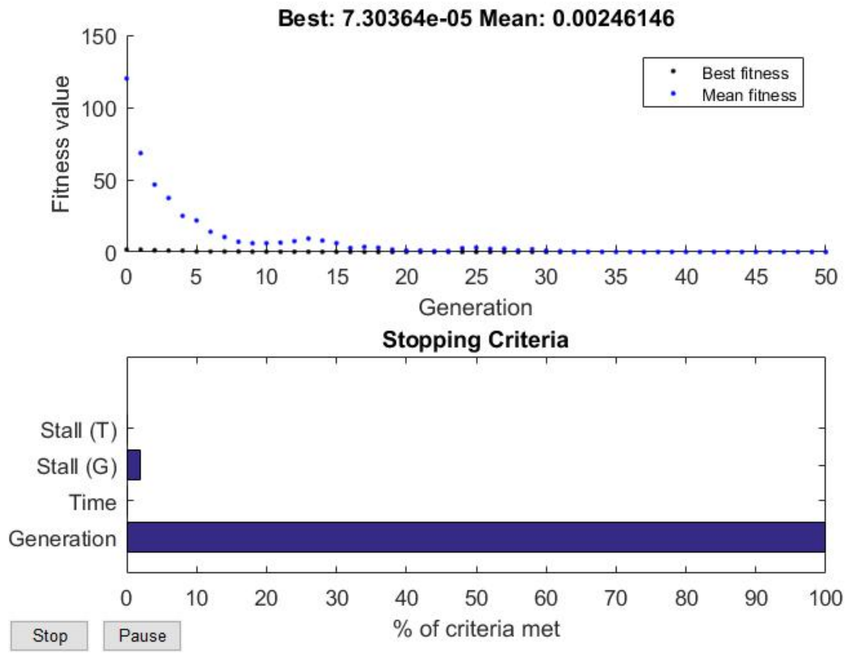


Figure 5.8: Fitness Plot for training with Genetic Algorithm for 5 nodes(simple case)

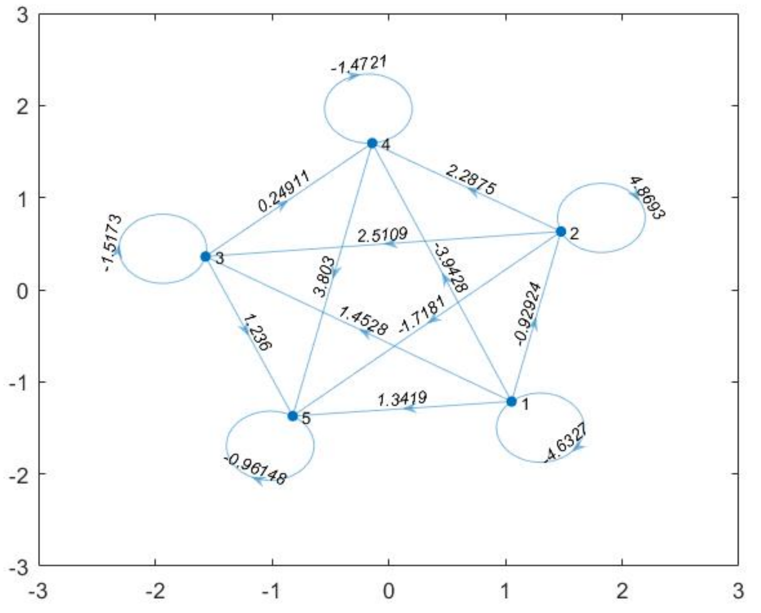


Figure 5.9: Network Graph for training with Genetic Algorithm for 5 nodes(simple case)

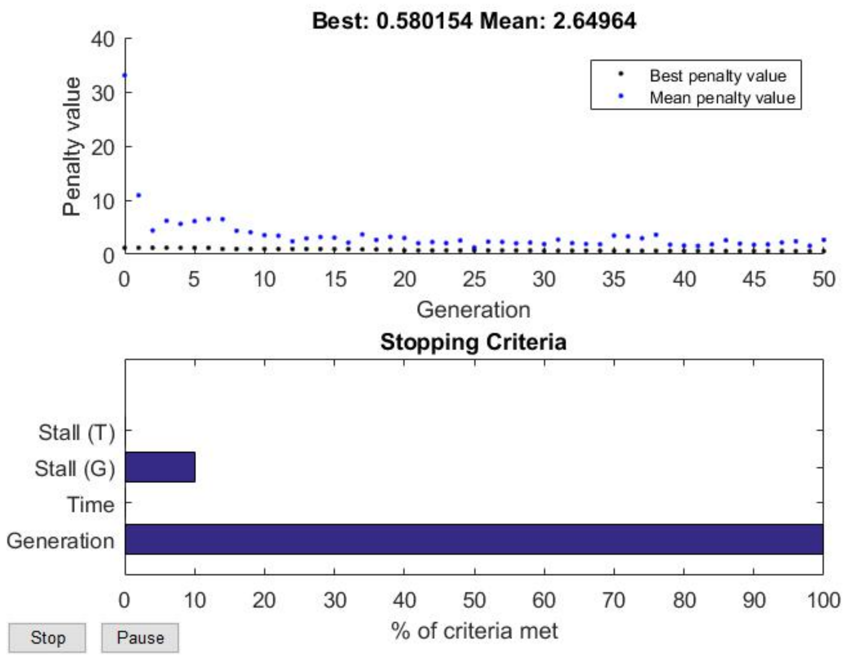


Figure 5.10: Fitness Plot for training with Genetic Algorithm for 4 nodes(complex case)

5.3 Particle Swarm optimization

Figure 5.11 shows fitness plot for training using particle swarm optimization algorithm on 5 nodes. Clearly, compared to genetic algorithm PSO performed better. PSO has been known to perform better than GA in finding solutions. However, this result cannot be sufficient to conclude that PSO performs better than GA for trainig neural networks. After all the results are close. However, this result provides an opportunity to explore training using PSO.

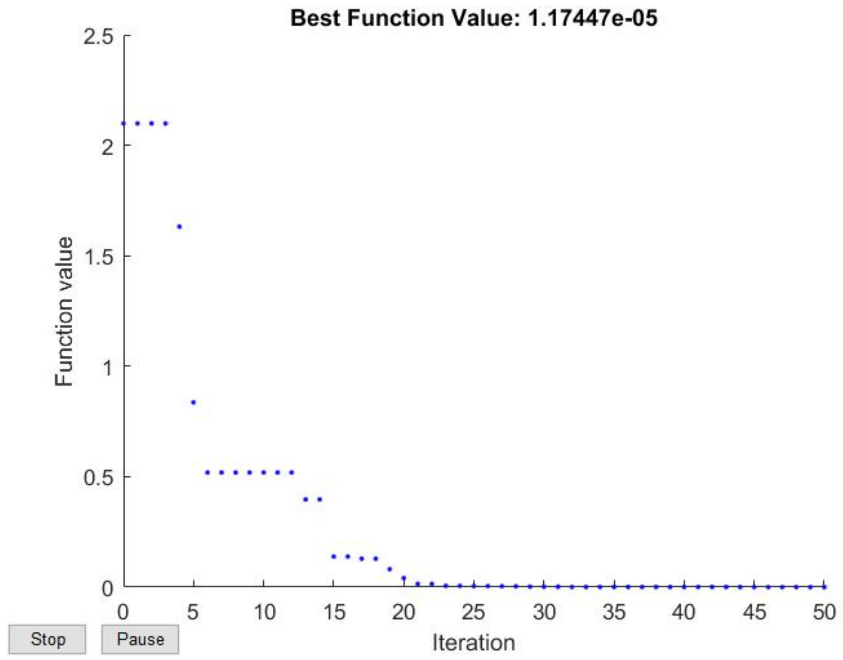


Figure 5.11: Fitness Plot for training with Particle Swarm Optimization Algorithm for 5 nodes(simple case)

5.4 Computation time analysis

From the results above we can observe BPSGD achieved the local minima in around 200 epochs, while global optimum has been achieved by the genetic algorithm in less than 5 generations for all the cases except the complex one. One should notice that while the computation time when running on a single CPU might be high in case of GA, the enormous parallel part of the algorithm can help in porting the application to GPU's offering high computation speeds. In this context, we can assume the time taken for an epoch and generation to be of equal order. Thus, from the results above, in this case, we can state that genetic algorithm performs a lot faster than BPSGD.This speed can further be improved by increasing the population count. However, a more detailed analysis should be made before generalizing the statement to different large-scale datasets, particularly when the structure is also being optimized in addition to weights.

5.5 Summary

From above results we can conclude that evolutionary algorithms can indeed find better solutions compared to BPSGD and hence opens the doors for exploring the training of neural networks. In the next chapter we discuss training on standard datasets using a statistical approach to better compare the performances of various configurations.

Chapter 6

Results

In this chapter the experimental setup and results for various configurations described in chapter 4 are discussed. Due to insufficient time, only 5 configurations are considered. The datasets belong to two different domains. The first one is a regression while the later is a classification problem. In the experiments below the effects due to variation in training sample size is also captured. In order to have results comparable to different configurations and different training sizes, same validation data is used in every experiment. The validation data is the complete dataset. the training data, however, varies over multiple repetitions of same sample size but constant over different configurations. Having complete data for validation seems like overfitting on the validation dataset. However, the objective of these experiments is to compare the performance of multiple configurations and under identical conditions. More details on the exact approach are mentioned in the following sections.

6.1 Power Plant dataset

The dataset is taken from UCI machine learning repository. The data is available at [19]. The dataset consists of data points collected from Combined Cycle Power Plant over a period of 6 years. The data has 5 features namely Temperature, Ambient Pressure, Relative Humidity, Exhaust Vaccum and Net hourly electrical energy output. A regression problem is formulated to predict the net hourly electrical energy output from the first four features.

Before training the data is normalized for all the features such that the value of all the features lies in the range of 0 and 1. This is done so that the final output is not affected by bound constraints of link weights. Following this step experiments are performed to evaluate configurations mentioned in chapter 4 for varying size training data. The following sample length vector is used during the experiments.

$$SampleLengthVector = [3, 6, 9, 12, 15, 30, 60, 120, 150]$$

A brief Stateflow of training data generation is shown in figure 6.1. For each sample size, 10 experiments are performed. This is done to avoid bias on a particular training data. The results for each configuration is discussed in following subsections and finally, the summary section discusses the results as a whole for this dataset.

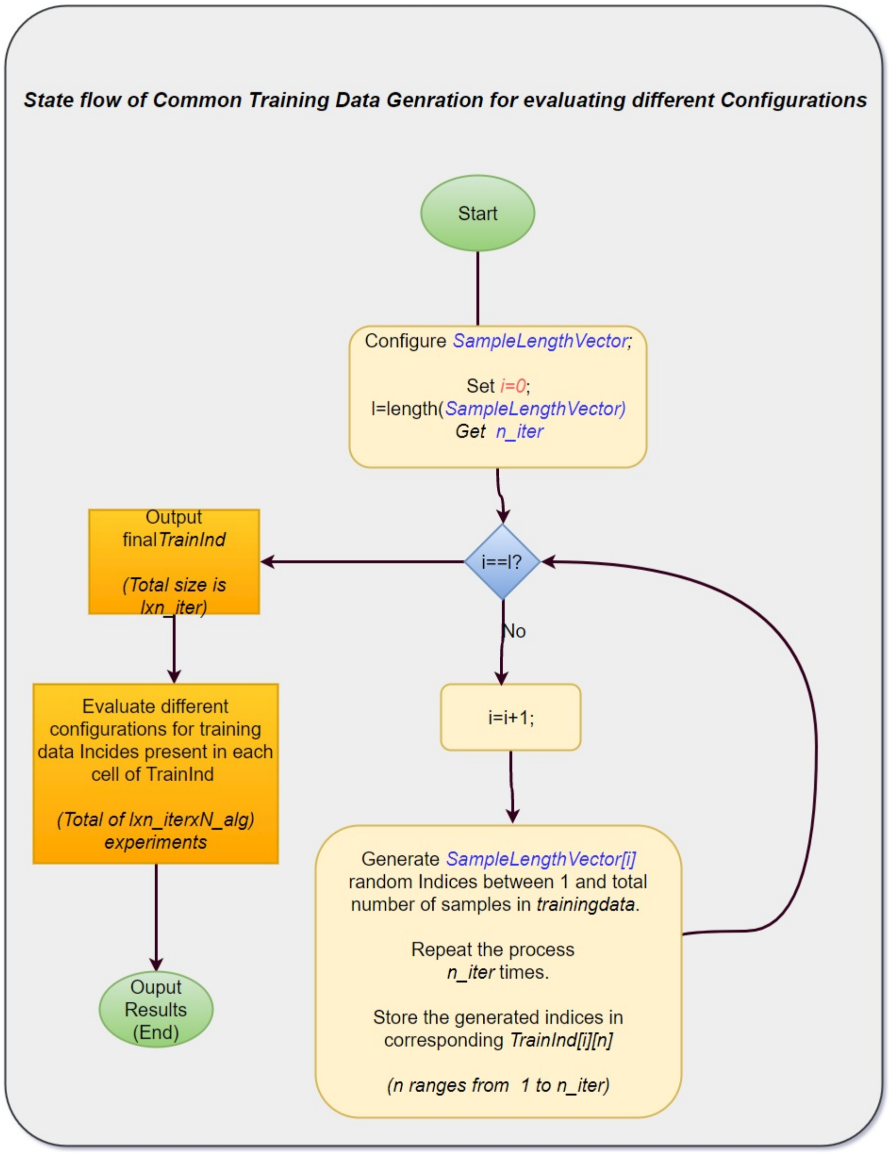

State flow of Common Training Data Genration for evaluating different Configurations
Start
Configure SampleLengthVector;
Set i=0;
l=length(SampleLengthVector)
Get n_iter
i==l?
No
i=i+1;
Generate SampleLengthVector[i] random Indices between 1 and total number of samples in trainingdata.
Repeat the process n_iter times.
Store the generated indices in corresponding TrainInd[i][n]
(n ranges from 1 to n_iter)
Output finalTrainInd
(Total size is lxn_iter)
Evaluate different configurations for training data Incides present in each cell of TrainInd
(Total of lxn_iterxN_alg) experiments
Ouput Results (End)

Figure 6.1: Stateflow of common Training data generation for evaluating different algorithms

6.1.1 BPSGD(tansig)

This configuration uses back propagation algorithm with the nodes in the hidden layer having a hyperbolic tangent sigmoid function as their activation function. Figure 6.2 shows the network architecture used to train the model. As it can be observed from the figure the network has two hidden layers with 10 nodes each and the output has linear activation function. Mean Squared Error (MSE) is used as the metric to evaluate the performance of the training. The mean and standard deviation of MSE's for this configuration is shown in Figure 6.3. It can be observed from the plots as the training sample size increases, the mean error is reduced. This trend can be expected, as the training size increases, the quality of fit increases. Similar trends can be explained by the standard deviation plot. The more the sample size, more is the generalization of the data and hence less deviation with respect to the differences in the training data.

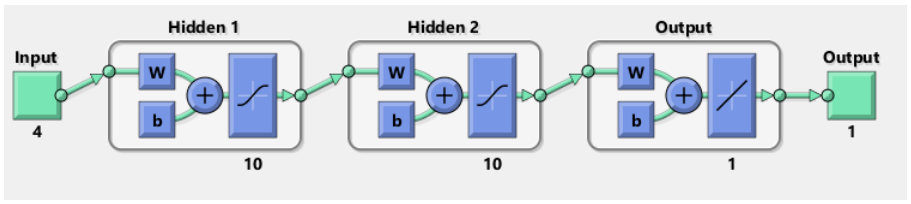


Figure 6.2: Feed forward Network Architecture used in regression of power dataset

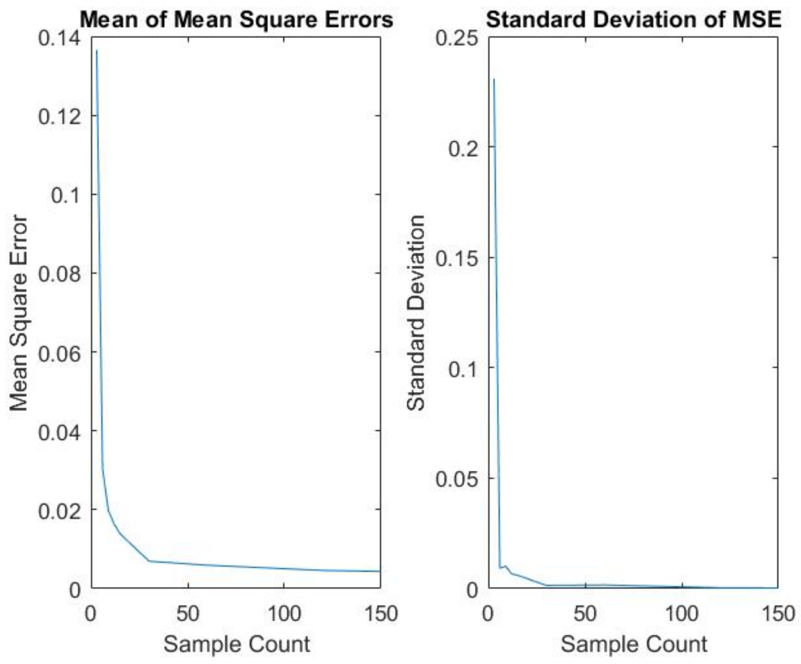


Figure 6.3: Mean and standard deviation of Mean Square Errors of power dataset for BPSGD (tansig)

6.1.2 BPSGD(logsig)

This configuration uses back propagation algorithm with the nodes in the hidden layer has Log-sigmoid function as their activation function. Figure 6.4 shows the network architecture used to train the model. As it can be observed from the figure similar to the above configuration the network has two hidden layers with 10 nodes each, and the output has linear activation function. Mean Squared Error (MSE) is used as the metric to evaluate the performance of the training. The mean and standard deviation of MSE's for this configuration is shown in Figure 6.5. The trends can be explained with same reasons mentioned in the previous configuration.

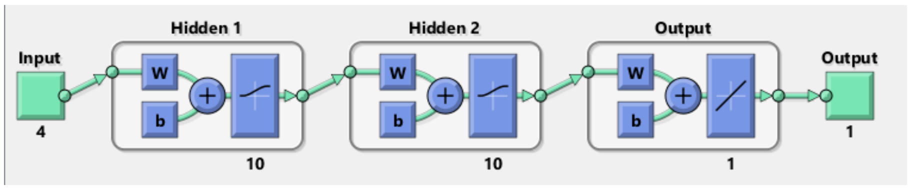


Figure 6.4: Feed forward Network Architecture used in regression of Power dataset

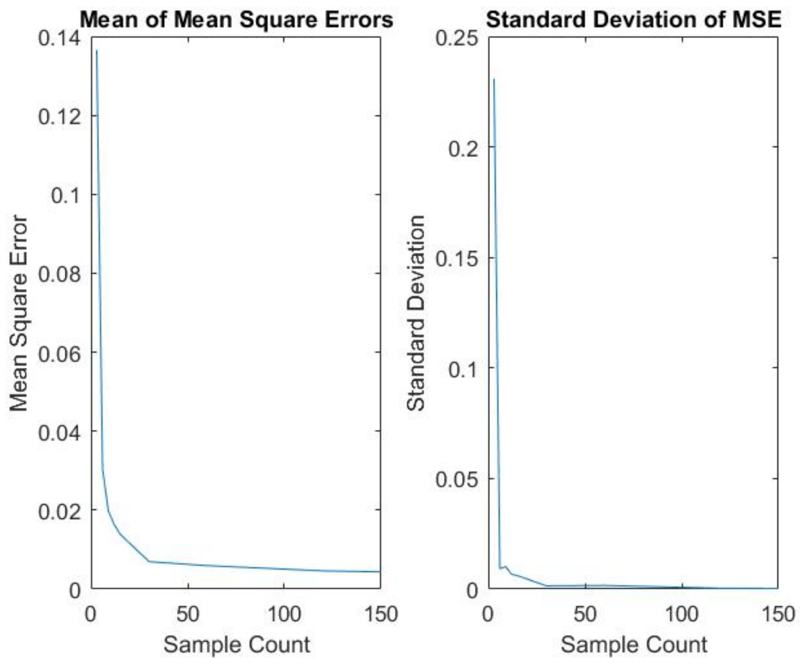


Figure 6.5: Mean and standard deviation of Mean Square Errors of power dataset for BPSGD (logsig)

6.1.3 Genetic Algorithm (Simple)

This configuration uses Genetic Algorithm for training on a Simple network topology. Figure 6.6 shows the network architecture used in the training process. All the hidden nodes use Log-Sigmoid activation function. The input and output nodes use Linear Activation Function. Similar to the backpropagation training, Mean Squared Error (MSE) is used as the metric to evaluate the performance of the training. The mean and standard deviation of MSE's for this configuration is shown in Figure 6.7. Similar to the backpropagation the trends of mean and standard deviation plots decreases with increase in training sample size.

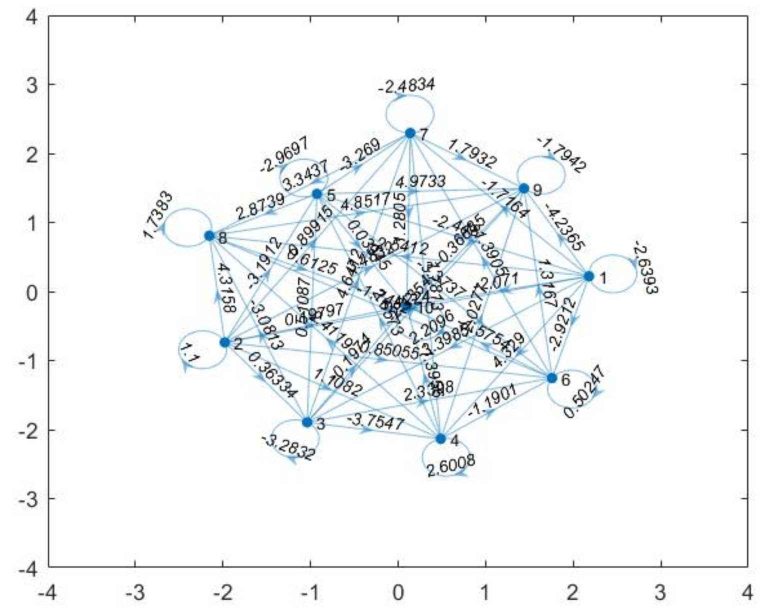


Figure 6.6: Network Architecture of 10 nodes (Simple topology)

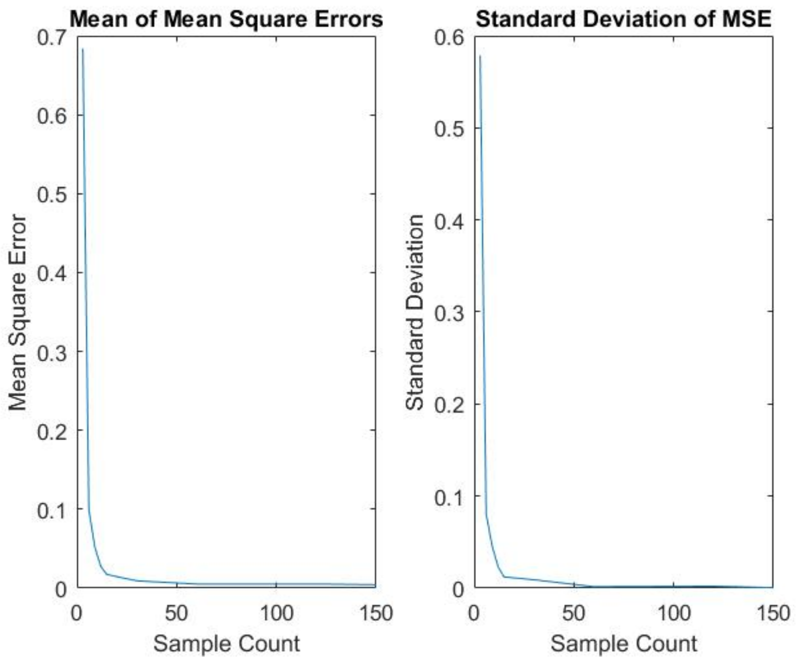


Figure 6.7: Mean and standard deviation of Mean Square Errors of power dataset for GA (simple)

6.1.4 Particle Swarm Optimization Algorithm (Simple)

This configuration uses Particle Swarm Optimization Algorithm for training on a Simple network topology. The same network topology shown in Figure 6.6 for training GA+simple configuration is used. All the hidden nodes use Log-Sigmoid activation function. The input and output nodes use Linear Activation Function. The results for this configuration are shown in figure 6.8 and the trends are consistent with the concepts discussed in above configurations.

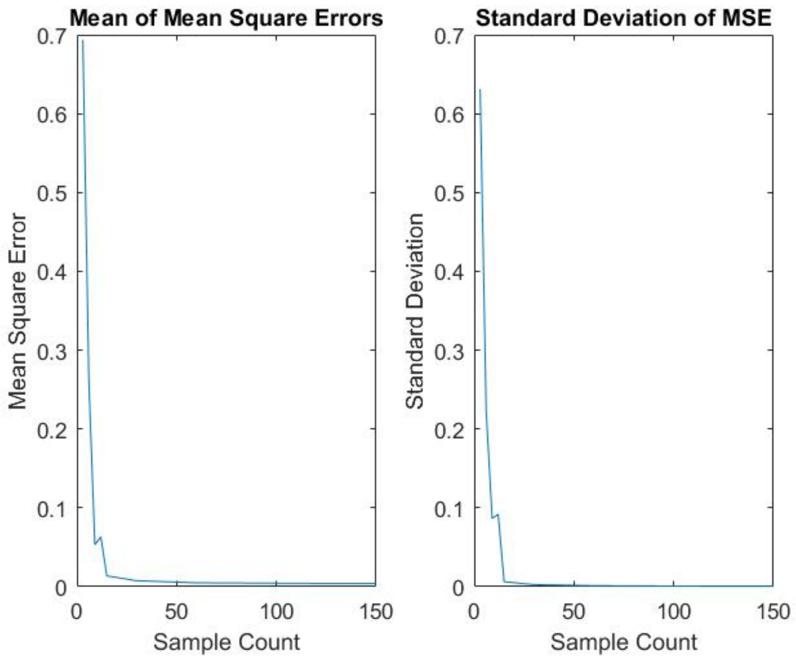


Figure 6.8: Mean and standard deviation of Mean Square Errors of power dataset for PSO(simple)

6.1.5 Neuro Evolution of Augmented Topologies (NEAT)

NEAT optimizes topology in addition to weights, hence the topology is different in each experiment. The results of training using this configuration are shown in figure 6.9. The trends are consistent with increasing quality and generalization as the training sample size increases.

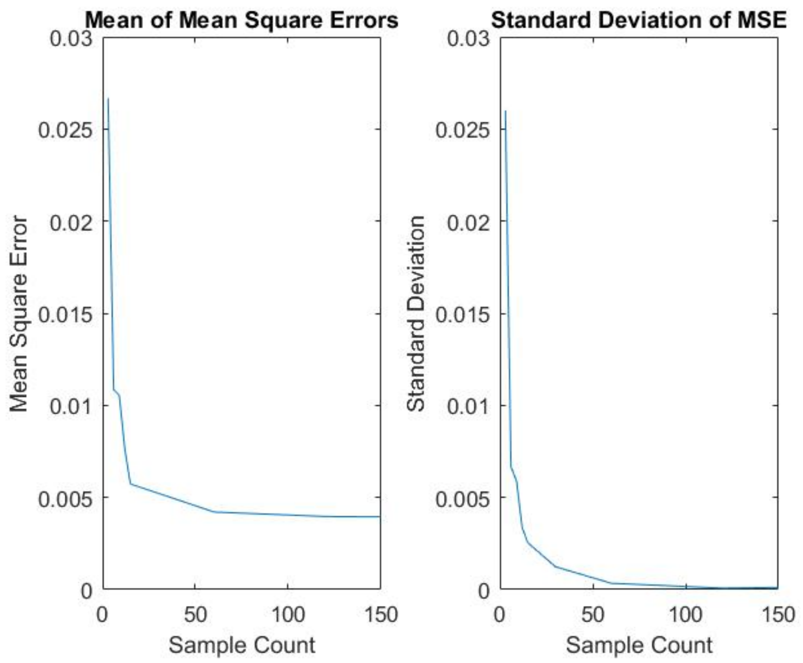


Figure 6.9: Mean Accuracy and standard deviation plot of iris dataset for NEAT

6.1.6 Summary

Quality

Figure 6.10 shows a summary of the performance of different configurations discussed in above subsections.The exact values can be found in the table 6.1. Out of all the configurations, NEAT performed well compared to the other configurations, particularly at a lower number of training samples. As we can observe with the increase in training sample size the performance is almost equal for all the configurations.

	GA+Simple	**BPSGD+TanSig**	**BPSGD+LogSig**	**PSO+Simple**	**NEAT**
3	0.68376+/-0.57863	0.13652+/-0.2309	0.042217+/-0.032135	0.69338+/-0.63102	0.026669+/-0.026017
6	0.099286+/-0.080311	0.030504+/-0.0091773	0.024808+/-0.013103	0.26052+/-0.22338	0.010851+/-0.006653
9	0.051988+/-0.045794	0.01981+/-0.010141	0.020496+/-0.0080686	0.053197+/-0.086474	0.010555+/-0.0058879
12	0.027222+/-0.02298	0.016373+/-0.0067009	0.014701+/-0.0049024	0.06327+/-0.09194	0.0076345+/-0.0033662
15	0.017272+/-0.012104	0.013969+/-0.0060821	0.013892+/-0.0053191	0.013653+/-0.0062327	0.0057329+/-0.0025489
30	0.0092183+/-0.0091009	0.0069448+/-0.0014804	0.010571+/-0.0044046	0.0075936+/-0.0026406	0.0052297+/-0.0012361
60	0.0052015+/-0.0015815	0.0059386+/-0.0016506	0.012318+/-0.0048853	0.0048906+/-0.0012055	0.0042098+/-0.0003430
120	0.0052162+/-0.0021051	0.0046127+/-0.00037465	0.0093997+/-0.0027696	0.004153+/-0.00065688	0.0039678+/-7.9022e-05
150	0.0043407+/-0.00050876	0.0043615+/-0.00031327	0.0086614+/-0.0028556	0.0040843+/-0.00051284	0.0039468+/-0.0001173

Table 6.1: Summary results table of training of various configurations on Power dataset

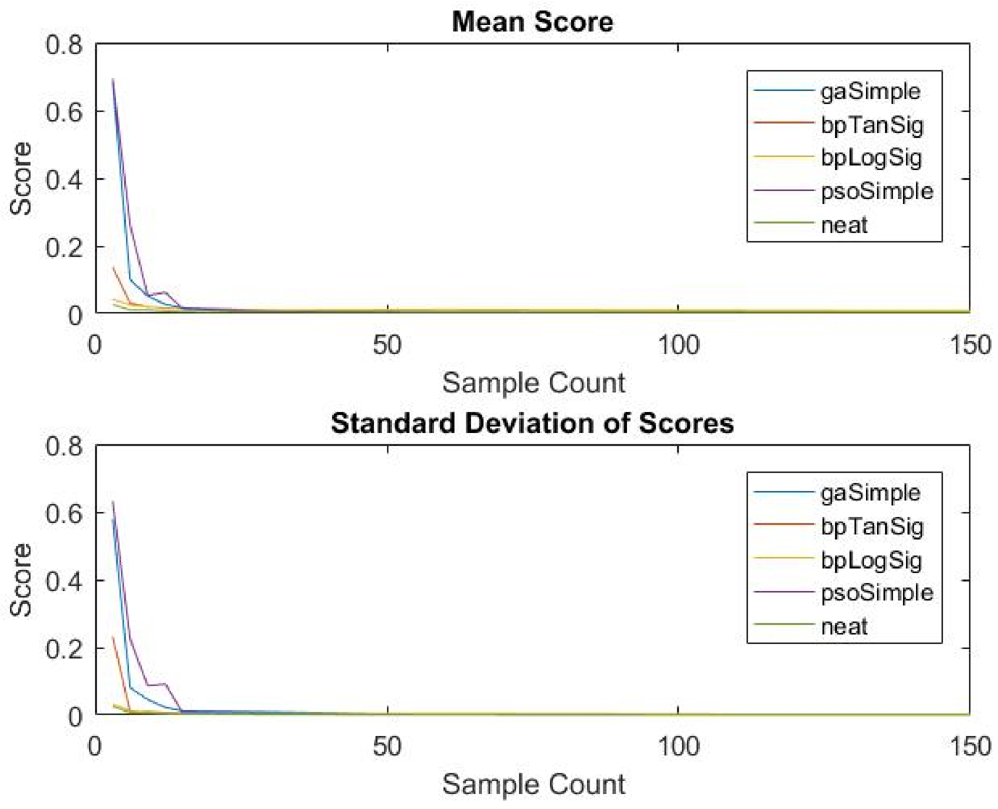


Figure 6.10: Summary plot of Mean and standard deviation of MSE for various configurations on Power Plant dataset

Computation Time

For training involving back propagation maximum epochs of 50,000 is allowed. This is done in order to let the training process using backpropagation to converge if possible. In other words, a gradient close to zero whenever possible is important to determine if the convergence is around local minima. On the other hand, the training using evolutionary algorithms GA and PSO are allowed a maximum of 200 generations and NEAT is allowed for 500 generations. This choice of number is purely based on practical time constraints. It would be interesting to see how the quality of solutions improve if more number of generations are allowed. From the above results, we can observe that the results are comparable to the results with back propagation configurations. However, from the perspective of computation time, the evolutionary algorithms are a lot faster when parallel processing capabilities are considered (very slow if only one CPU is used depending on the population size). For more detailed insight into how backpropagation would perform if the maximum number of epochs is comparable to that of maximum generations in evolutionary algorithms, following experiments are performed. Under the assumption inspired from [14], that computation time of one epoch is approximately equal to one generation, the modified summary plot allowing BPSGD for a maximum of 200 epochs is shown in figure 6.11. With little more leniency and allowing BPSGD to run 2.5 epochs for 1 generation, the modified summary plot for allowing BPSGD for a maximum of 500 epochs is shown in figure 6.12. It can be observed from the results the performance both in terms of quality and generalization is a lot better in case of training using evolutionary algorithms compared to BPSGD. Also, the performance improvement from 200 to 500 epochs is not that high.

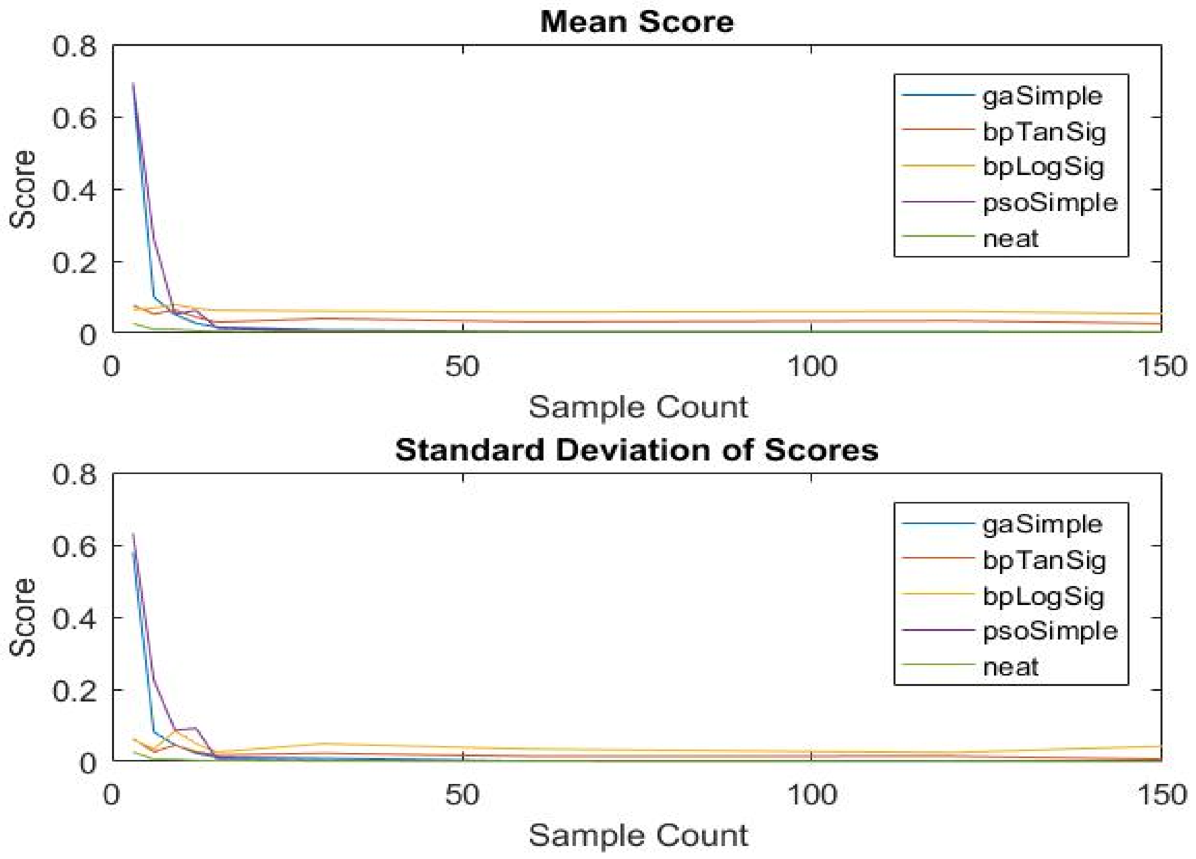


Figure 6.11: Summary plot of Mean and standard deviation of MSE for various configurations on Power Plant dataset(with Back Propagation configurations having max epochs of 200)

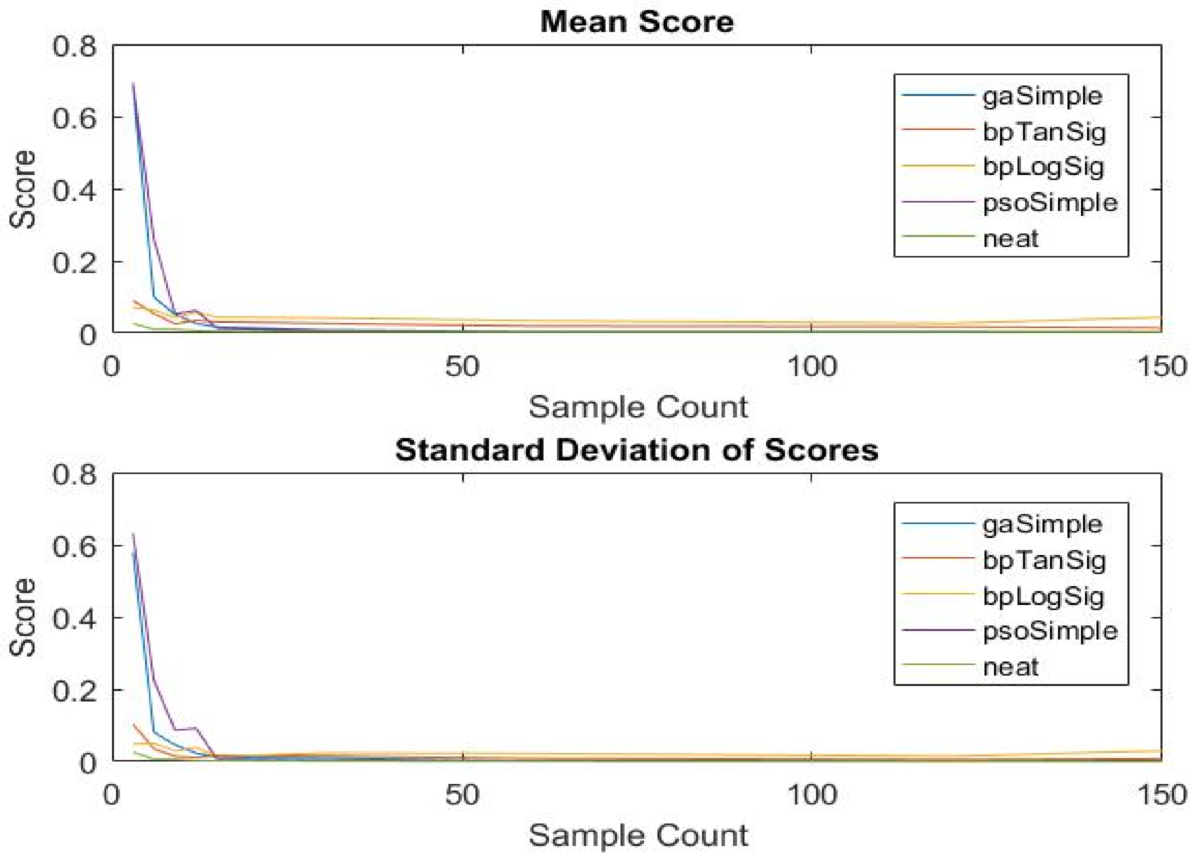


Figure 6.12: Summary plot of Mean and standard deviation of MSE for various configurations on Power Plant dataset(with Back Propagation configurations having max epochs of 500)

6.2 IRIS dataset

Iris dataset [8], created by R.A. Fischer, is a famous benchmarking dataset used in many pattern recognition studies to assess the classification performance of an algorithm. The data consists of four attributes of an iris plant and one target class. The attributes are namely sepal length, sepal width, petal length and petal width. The possible target classes are Setosa, Versicolor, and Virginia. The objective of the algorithm is to predict the correct class of the plant for a given attribute information. As mentioned in section 4 one-shot learning is used to assess the performance of various configurations described in section 4. The results of each configuration are as follows. A similar process used in creating the training data for Power Plant dataset is used. The Stateflow is also the same as shown in the figure 6.1. However, in this case instead of total sample count, as this is a classification problem, the training data should contain at least one sample per class.The following sample length vector (samples per class) is used during the experiments.

$$SampleLengthVector = [1, 2, 3, 4, 5, 10, 20, 40, 50]$$

As there are three classes in total, this translates to the following sample count vector.

$$SampleCountVector = [3, 6, 9, 12, 15, 30, 60, 120, 150]$$

A total of 10 experiments are performed for each sample size to avoid bias on the training data. The results are as follows.

6.2.1 BPSGD(tansig)

This configuration uses back propagation algorithm with the nodes in the hidden layer having a hyperbolic tangent sigmoid function as their activation function. Figure 6.13 shows the network architecture used to train the model. As it can be observed from the figure the network has two hidden layers with 10 nodes each, and 3 output nodes.The outputs from the output nodes are passed through a softmax layer. The output of this layer indicates the probability of input belonging to a particular class. The objective function is to minimize the cross-entropy between the expected distribution and the predicted distribution. The mean and standard deviation of accuracy for this configuration is shown in Figure 6.14. It can be observed from the plots as the training sample size increases, the mean accuracy increases. This trend can be expected, as the training size increases, the quality of fit increases. Similar trends can be explained by the standard deviation plot. The more the sample size, more is the generalization of the data and hence less deviation with respect to the differences in the training data.

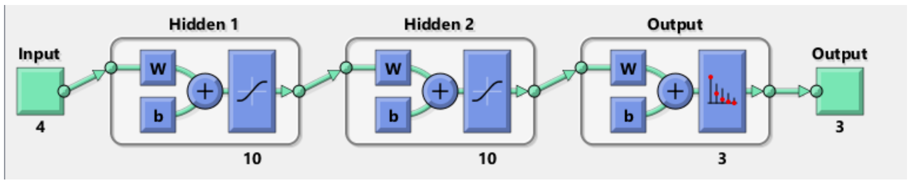


Figure 6.13: Feed forward Network Architecture used in Classsifcation of IRIS dataset

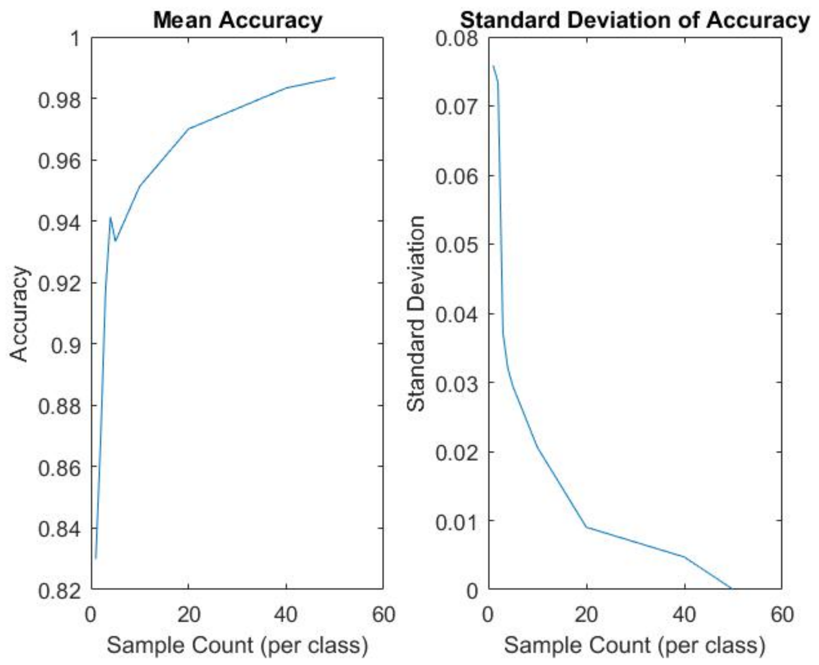


Figure 6.14: Mean Accuracy and standard deviation plot of iris dataset for BPSGD (tansig)

6.2.2 BPSGD(logsig)

This configuration uses back propagation algorithm with the nodes in the hidden layer having Log sigmoid function as their activation function. Figure 6.15 shows the network architecture used to train the model. The architecture is identical to the above case, except for the activation function of the hidden nodes. The mean and standard deviation of accuracy for this configuration are shown in Figure 6.16. The plots indicate the quality and generalization of the model increases with increase in the size of training data similar to the above case.

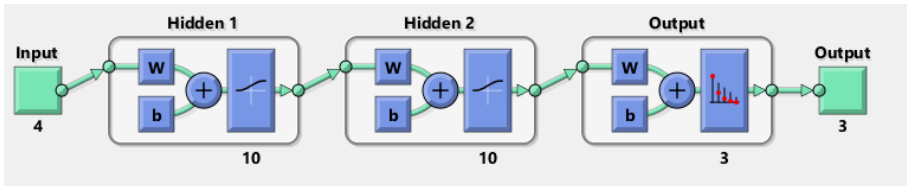


Figure 6.15: Feed forward Network Architecture used in Classsifcation of IRIS dataset

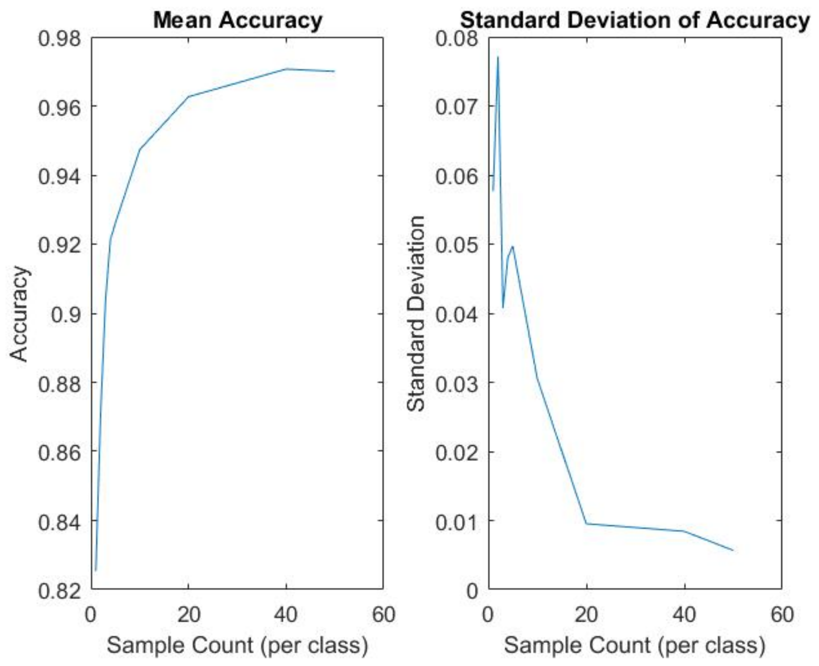


Figure 6.16: Mean Accuracy and standard deviation plot of iris dataset for BPSGD (logsig)

6.2.3 Genetic Algorithm (Simple)

This configuration uses Genetic Algorithm for training on a Simple network topology.The network architecture used in same as the one used during the training of power dataset shown in figure 6.6. All the hidden nodes use Log-Sigmoid activation function. The input and output nodes use Linear Activation Function. Similar to the backpropagation training, accuracy is used as the metric to evaluate the performance of the training (the objective function is, however, cross entropy between expected and predicted probability distributions). The mean and standard deviation of accuracy for this configuration is shown in Figure 6.17. While the overall trend of accuracy increases with increase in training size, the trend is not as smooth as in the case of backpropagation. This is because of the data used in the training process and not due to the genetic algorithm. This can be observed from the fact that accuracy value compared to backpropagation. GA managed to model the data better than BPSGD at lower sample sizes and the fluctuations in the accuracy are solely because of data presented at that particular data samples.

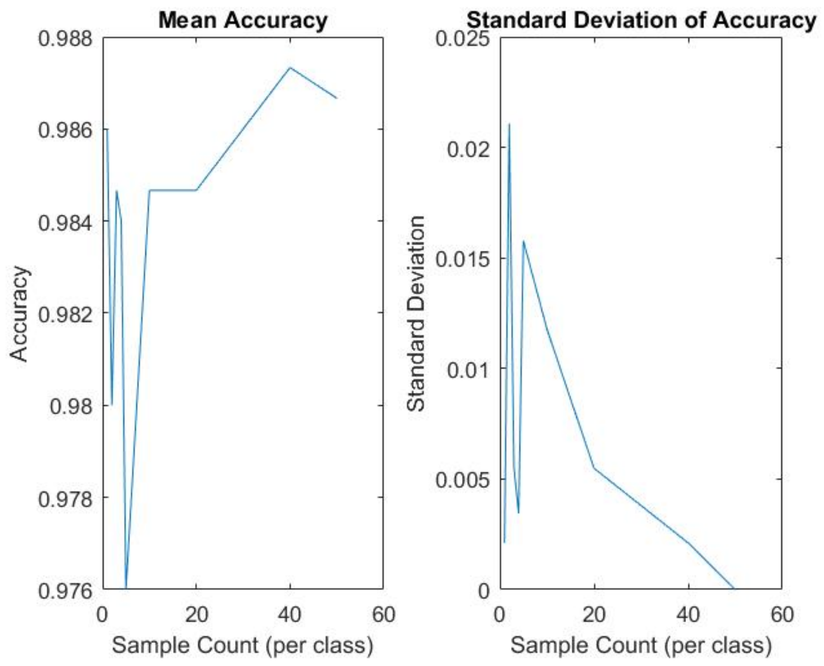


Figure 6.17: Mean Accuracy and standard deviation plot of iris dataset for GA (simple)

6.2.4 Particle Swarm Optimization Algorithm (Simple)

This configuration uses PSO for training on a simple network topology. The network architecture, evaluating metrics and experimental conditions are similar to the above mentioned GA+Simple configuration. The mean and standard deviation of accuracy for this configuration is shown in Figure 6.18. The trends are similar to the process of training with GA. PSO managed to model the data better than both BPSGD and GA in this case.

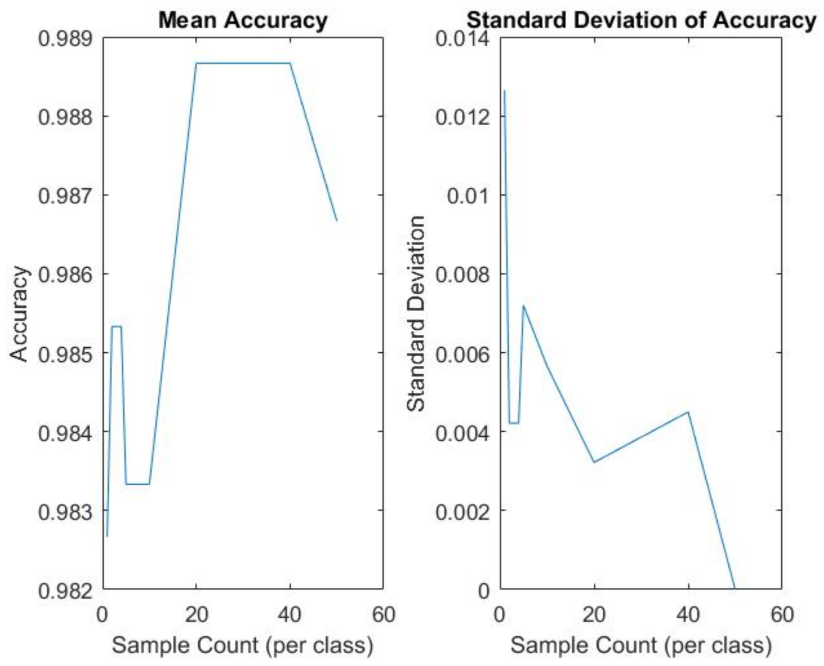


Figure 6.18: Mean Accuracy and standard deviation plot of iris dataset for PSO (simple)

6.2.5 Neuro Evolution of Augmented Topologies (NEAT)

NEAT optimizes topology in addition to weights, hence the topology is different in each experiment. The results of training using this configuration are shown in figure 6.9. The trends are not reliable compared to the other evolutionary algorithms. This is particularly evident from the fact of high standard deviation at higher sample size. This could be because of the fact, that NEAT evolves topology in addition to the optimizing the objective function. Because of this reason, it is expected that NEAT requires more generations to model the data better.However, compared with the results in case of regression, this is still a puzzling result. More on this is described in the overall results summary section 6.5

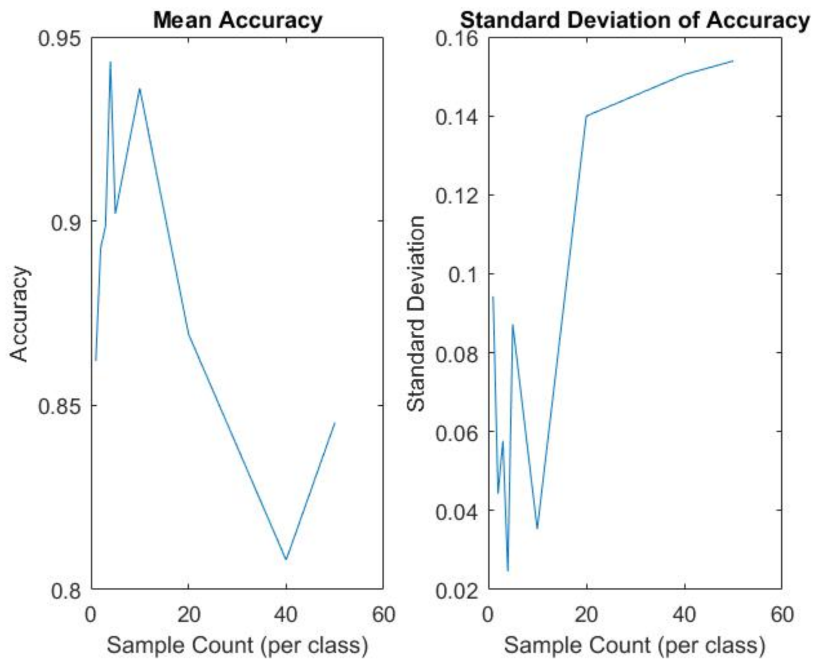


Figure 6.19: Mean Accuracy and standard deviation plot of iris dataset for NEAT

6.2.6 Summary

6.3 Quality

Figure 6.10 shows a summary of the performance of different configurations discussed in above subsections. The exact values can be found in the table 6.2. Out of all the configurations, PSO+Simple performed well compared to the other configurations, particularly at a lower number of training samples. As we can observe with the increase in training sample size the performance is almost equal for all the configurations.

Sample Count (Per Class)	**GA+Simple**	**BPSGD+TanSig**	**BPSGD+LogSig**	**PSO+Simple**	**NEAT**
1	0.986+/-0.0021082	0.83+/-0.075898	0.82533+/-0.057675	0.98267+/-0.012649	0.862+/-0.094284
2	0.98+/-0.021082	0.86867+/-0.07337	0.87133+/-0.077175	0.98533+/-0.0042164	0.89267+/-0.044272
3	0.98467+/-0.0054885	0.91733+/-0.037211	0.904+/-0.040758	0.98533+/-0.0042164	0.89867+/-0.057675
4	0.984+/-0.0034427	0.94133+/-0.032019	0.92133+/-0.048053	0.98533+/-0.0042164	0.94333+/-0.024595
5	0.976+/-0.015776	0.93333+/-0.029481	0.926+/-0.049735	0.98333+/-0.0072008	0.902+/-0.087209
10	0.98467+/-0.01178	0.95133+/-0.02062	0.94733+/-0.030542	0.98333+/-0.0056656	0.936+/-0.035305
20	0.98467+/-0.0054885	0.97+/-0.0090267	0.96267+/-0.0095323	0.98867+/-0.0032203	0.86933+/-0.13992
40	0.98733+/-0.0021082	0.98333+/-0.004714	0.97067+/-0.0084327	0.98867+/-0.0044997	0.808+/-0.15045
50	0.98667+/-1.1703e-16	0.98667+/-1.1703e-16	0.97+/-0.0056656	0.98667+/-1.1703e-16	0.84533+/-0.15389

Table 6.2: Summary results table of training of various configurations on Iris dataset

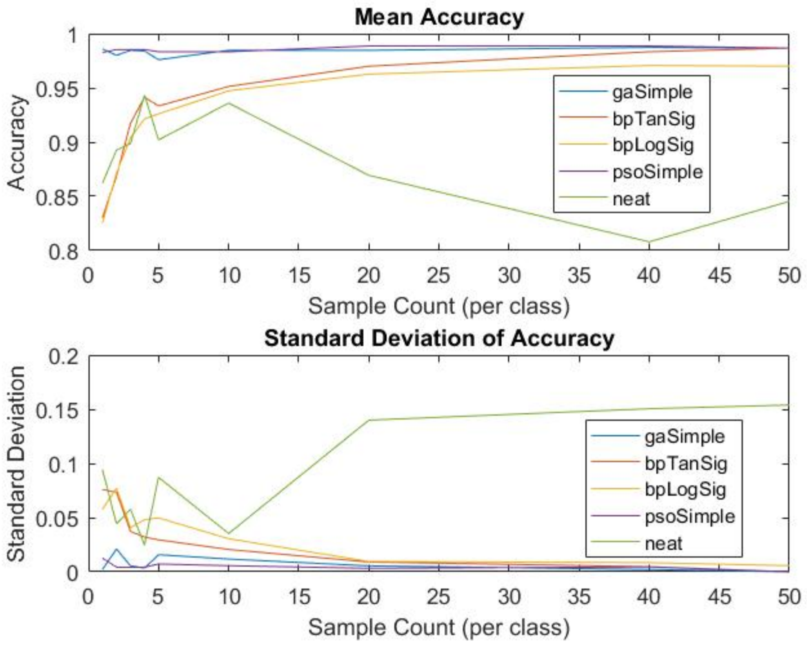


Figure 6.20: Mean and standard deviation of iris dataset

6.4 Computation Time

For training involving back propagation maximum epochs of 50,000 is allowed. This is done in order to let the training process using backpropagation to converge if possible. In other words, a gradient close to zero whenever possible is important to determine if the convergence is around local minima. On the other hand, the training using evolutionary algorithms GA and PSO are allowed a maximum of 200 generations and NEAT is allowed for 500 generations. This choice of number is purely based on practical time constraints. It would be interesting to see how the quality of solutions improve if more number of generations are allowed. From the above results, we can observe that the results are comparable to the results with back propagation configurations. However, from the perspective of computation time, the evolutionary algorithms are a lot faster when parallel processing capabilities are considered (very slow if only one CPU is used depending on the population size).

For more detailed insight into how backpropagation would perform if the maximum number of epochs is comparable to that of maximum generations in evolutionary algorithms, following experiments are performed. Under the assumption inspired from [14], that computation time of one epoch is approximately equal to one generation, the modified summary plot allowing BPSGD for a maximum of 200 epochs is shown in figure 6.21. With little more leniency and allowing BPSGD to run 2.5 epochs for 1 generation, the modified summary plot for allowing BPSGD for a maximum of 500 epochs is shown in figure 6.22. It can be observed from the results the performance both in terms of quality and generalization is a lot better in case of training using evolutionary algorithms compared to BPSGD. Also, the performance improvement from 200 to 500 epochs is not that high.

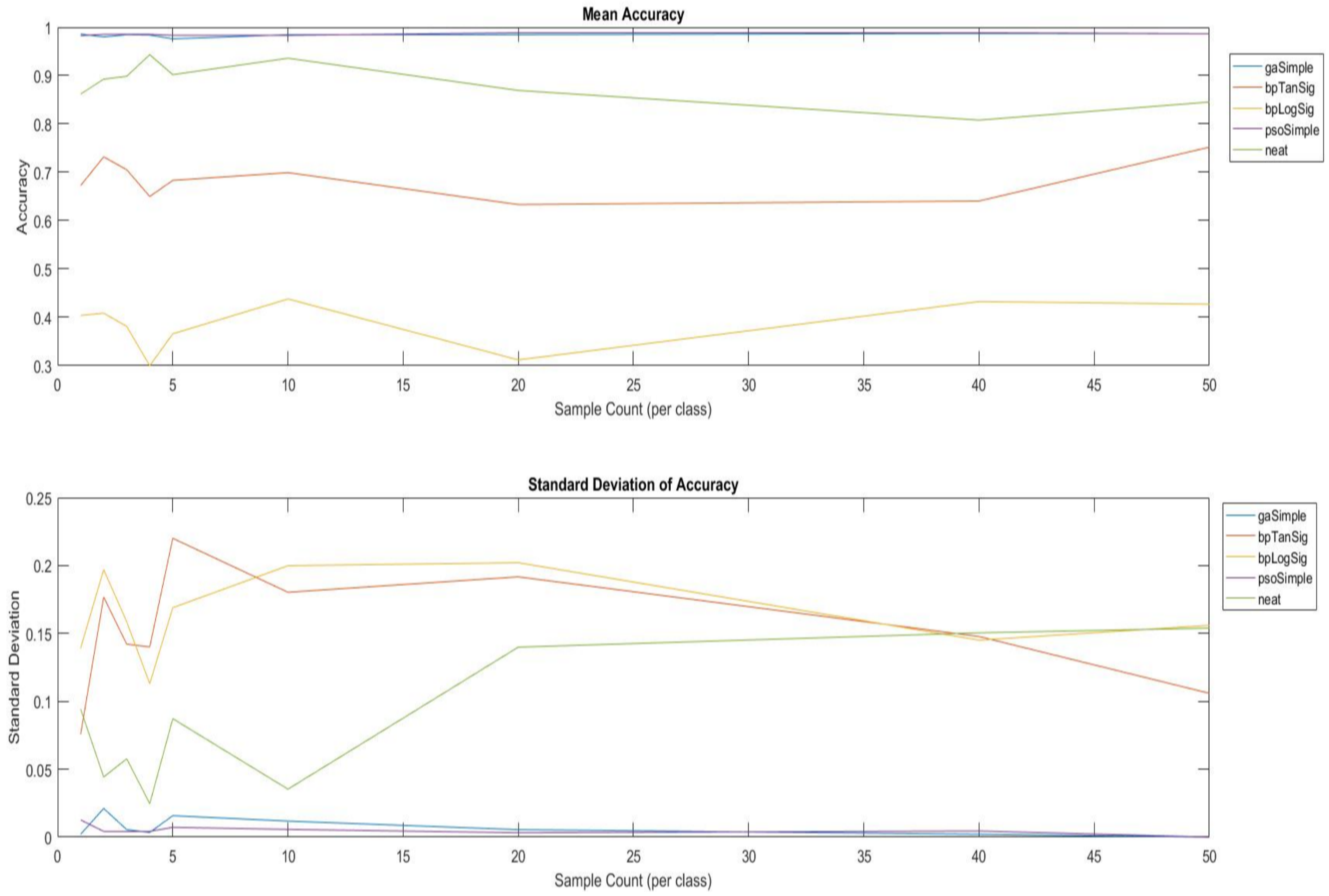


Figure 6.21: Summary plot of Mean and standard deviation of accuracy for various configurations on Iris dataset (restricting Back Propagation configurations to max epochs of 200)

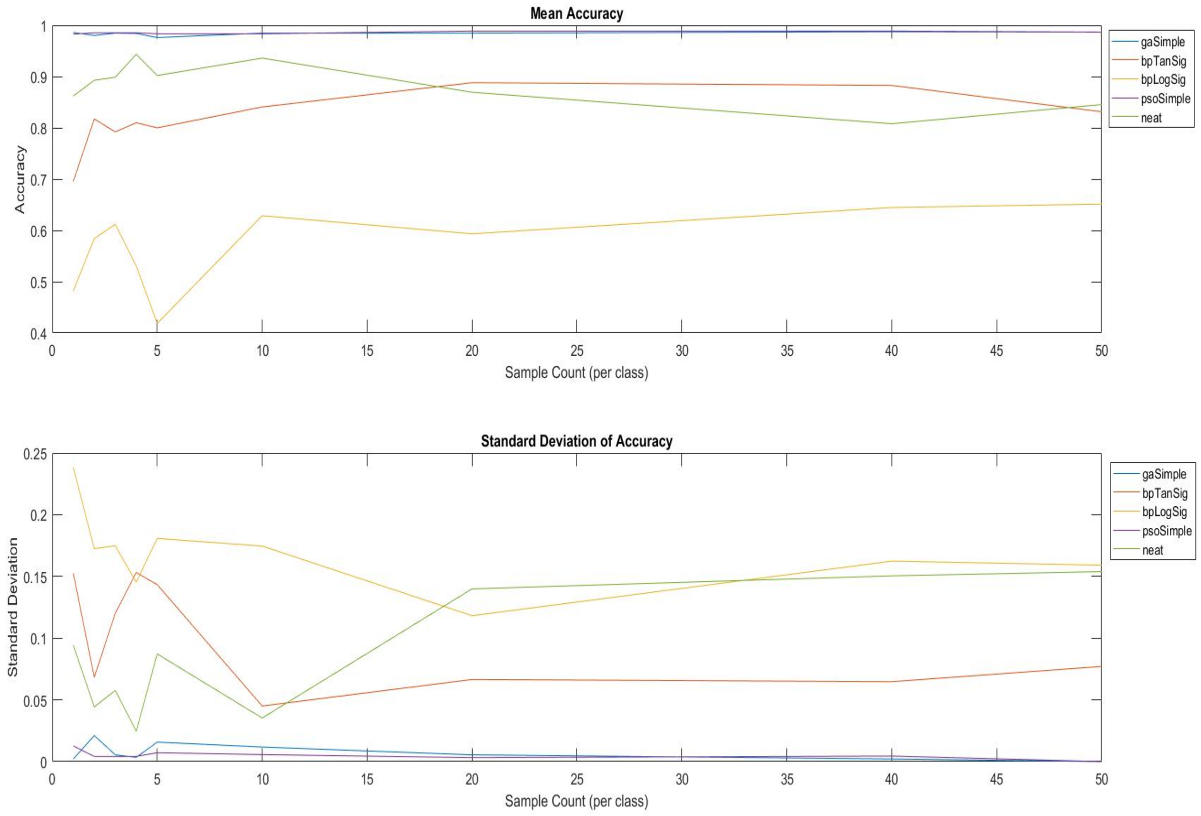


Figure 6.22: Summary plot of Mean and standard deviation of accuracy for various configurations on Iris dataset (restricting Back Propagation configurations to max epochs of 500)

6.5 Overall Results Summary

From the training of power and iris datasets, it can be observed overall that performance of evolutionary algorithms is comparable to backpropagation. The results are a lot better if parallel processing capabilities are included. There are, however, a couple of anomalies that need further attention. The first one is the performance of GA, PSO at very low sample size. This anomaly can be explained as follows. The simple topology has multiple connections coming to the output node. At low training data due to high range of weight bounds, the overfitting of less training data might result in such large fluctuations. While the validation metrics generally ensure no such overfitting occurs, this case of extremely very low sample training (3/9568) can become an exception. The other algorithms look better solely because of their lack of generating a wide range of output (less number of connections- less error combined). This argument can be reinforced by the fact that the performance becomes comparable immediately once the sample size is increased a bit further.

The other anomaly is the performance of NEAT in the classification of IRIS dataset. This anomaly is still little puzzling. The probable reason would be since NEAT evolves both topology and weights it might need further time to converge. However, this is just a speculation. A more detailed analysis will be provided in future.

Chapter 7

Conclusions and Further work

From the training results of power and iris datasets, it can be observed overall that performance of evolutionary algorithms is comparable to backpropagation. The results are a lot better if parallel processing capabilities are included. While it is too early to conclude that evolutionary algorithms perform better than BPSGD for most cases, the extra capabilities like including multi-objective optimization, satisfying structural constraints, huge support for parallel processing makes training using evolutionary algorithms an interesting study area in the future. With the advent of increasing popularity of GPU's, it would only be fair to say from the above results that the training using evolutionary algorithms become more and more prominent with time. The further work will be concentrated on scaling up the dimensionality of data-sets (like MNIST), spreading to other domains (time series, unsupervised learning), and exploring other configurations discussed in methodology section including dynamic network topology and adaptive parameter control.

Chapter 8

Acknowledgements

I would sincerely like to thank my supervisor **Dr. Decebel.C.Mocanu** for his continuous support and guidance throughout the phase of this seminar. I would also like to thank **Anil Yaman** for sharing his knowledge on adaptive parameter control techniques used in Evolutionary Algorithms.

Bibliography

[1] Marco Dorigo Alberto Colorni and Vittorio Maniezzo. “Distributed Optimization by Ant Colonies”. In: *European Conference on Artificial Life* (1991), pp. 134–142.

[2] Aldeida Aleti. *An Adaptive Approach to Controlling Parameters of Evolutionary Algorithms*. URL: `http://users.monash.edu.au/~aldeidaa/papers/thesis.pdf`.

[3] H. AttyaLafta, N. KdhimAyoob, and A. A. Hussein. “Breast cancer diagnosis using genetic algorithm for training feed forward back propagation”. In: *2017 Annual Conference on New Trends in Information Communications Technology Applications (NTICT)*. Mar. 2017, pp. 144–149. DOI: `10.1109/NTICT.2017.7976101`.

[4] P. Baldi and K. Hornik. “Neural Networks and Principal component Analysis”. In: *Neural Networks* 2 (1989), pp. 53–58.

[5] Chris J Maddison et al. David Silver Aja Huang. “Mastering the game of go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[6] G.E. Hintont D.E. Rumelhart and R.J. Williams. “Learning representations by backpropagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536.

[7] Agoston E. Eiben and J. E. Smith. *Introduction to evolutionary computing*. SpringerVerlag, 2003.

[8] R.A. Fischer. *Iris Dataset*. URL: `http://archive.ics.uci.edu/ml/datasets/Iris`.

[9] V. Genovese et al. “Self organizing behavior and swarm intelligence in a cellular robotic system”. In: *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*. Aug. 1992, pp. 243–248. DOI: `10.1109/ISIC.1992.225098`.

[10] Marco Gori and Alberto Tesi. “On the Problem of Local Minima in Back Propagation”. In: *IEEE transactions on Pattern Analysis and Machine Intelligence* 14.1 (1992), pp. 76–86.

[11] Fu Kai and Xu Wenhua. “Training neural network with genetic algorithms for forecasting the stock price index”. In: *1997 IEEE International Conference on Intelligent Processing Systems (Cat. No.97TH8335)*. Vol. 1. Oct. 1997, 401–403 vol.1. DOI: `10.1109/ICIPS.1997.672809`.

[12] Melanie Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[13] David J Montana and Lawrence Davis. *Training Feedforward Neural Network Using Genetic Algorithms*. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.77.3838&rep=rep1&type=pdf`.

[14] Gregory Morse and Kenneth O. Stanley. “Simple Evolutionary Optimization Can Rival Stochastic Gradient Descent in Neural Networks”. In: *GECCO 2016* (2016).

[15] et.al Risto Miikkulainen Jason Liang. *Evolving Deep Neural Networks*. URL: `https://arxiv.org/pdf/1703.00548.pdf`.

[16] Kenneth O. Stanley and Risto Miikkulanien. “Efficient Reinforcement Learning through Evolving Neural Network Topologies”. In: *Proceedings of geetic and evoluionary computation conference* (2002).

[17] Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning,1st ed.* MIT Press, Cambridge, MA, USA, 1998.

[18] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The elements of statistical learning.* Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[19] Pınar Tüfekci. *Combined Cycle Power Plant Data Set.* URL: `http://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant`.

[20] Yoshua Bengio Yann LeCun and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444.