5th Semester

PROJECT REPORT

Multi Objective Neuro-Evolution with Back Propagation Assisted Local Search

Mentor: Dr. Rahul Kala

Kunal Gupta (IIT2015007)
Swapnil Sharma (IIT2015085)
Atul Kumar Sinha (IIT2015087)
Aditya Pimparkar (IIT2015012)
Raghav Khandelwal (LIT2015002)

Abstract

Neural network as a technology has grown immensely and its multi-layered feed-forward properties make networks particularly suited to complex pattern classification problems. However, their application to real world problems show lack of a training algorithm to find optimal weights. Genetic algorithms are a class of optimization procedures that are good at finding global optimum for a given problem. Hence genetic algorithm can be used to train neural networks and find optimal values. In this report we use a common data set PIMA and find better ways of training neural networks using evolutionary algorithms and back-propagation assisted local search.

Keywords: Neural Networks, Genetic Algorithms, Back Propagation, Multi-Objective Evolution

1 INTRODUCTION

Neural network and genetic algorithms have evolved a lot since the 1980s. Both have grown independently and have been used in various fields like economics, stock market, health, pattern classification etc. Both have received a great acclaim in computer science research community, each having its own strength and weakness. Recently the world has witnessed the merger of strengths of the two technologies. Davis (1988) demonstrated that any neural network could be transformed or written as a type of genetic algorithm (classifier system) and vice-a-versa. Many other researchers have tried to combine the two technologies to train feed-forward neural networks. It has been witnessed that genetic algorithm can be used to evolve a neural network by achieving optimal value of hyper-parameters [1] in very less time as compared to general (brute-force) approach. In this paper we aim to find ways of optimizing neural network using various techniques of evolutionary algorithms and back propagation assisted local search. We used a hybrid model of neural network that uses both genetic algorithm and back propagation to obtain optimal results.

2 PROBLEM DEFINITION

2.1 INPUT

The higher order system gives the neural network a set of weights to feed forward, along with the number of hidden nodes, encoded in a single chromosome. The artificial neural network takes this chromosome as input, along with values from the Pima Indians Diabetes Data Set. The dataset requires 8 input nodes and gives a binary output on 1 node.

2.2 OUTPUT

Applying feed forward on this network allow us to calculate the accuracy of the network using the supplied weights on the input dataset, which is returned back to our global search as output of Neural Network. Genetic Algorithm uses this as a parameter to evaluate fitness, and continues this cycle to finally return the configuration of network which is best in terms of accuracy, i.e. the set of weights and number of hidden nodes in that network.

2.3 CONSTRAINTS

Since creating and training a very large population of networks repeatedly, can take up a huge chunk of memory, take a very long time and can sometimes just be redundant, we put some constraints on our operations and problem instance sizes.

In generating variable length chromosomes, the number of hidden nodes is set from 0 to 20, and 200 of such chromosomes are generated. This population pool is wide enough to incorporate all the variations, and make our search space truly global, as well as it allows us to reach an optimal solution in an appropriate time frame. Also, the number of hidden layers is set to 1 in all such genomes, as increasing hidden layers increases complexity significantly. The initial weights of network lie in the range [-1,1] which allows search algorithm to reach to optimal set of weights faster.

The probability of mutation is a very significant factor that generates all of the variety, required to reach the optimal point in search space and can also cause the algorithm to wander around it, never actually reaching the global solution. To address both of these, the probability of mutation is set to 0.2 in first 200

generations to allow greater variety and then reduced to 0.02 to let it converge on a single optimal result.

Similarly, the probability of crossover is set to 0.8 which allows some partners to not generate child chromosomes, and reproduce themselves into the next generation of population. This distinctively incorporates the ideology of elitism into search strategy.

Finally, a run lasts for 500 generations, ingeniously allowing the members of our genome to mature enough and produce a result that can get as close as it can, to the globally optimum result.

3 Literature Review

3.1 Genetic Algorithms [2]

Genetic algorithms are Search and Optimization algorithms driven by natural evolution principles. Genetic algorithms were first introduced by John Holland in 1970. Genetic algorithms implement optimization by carrying out evolution of strings (Chromosomes in Biology). They combine survival of the fittest among string structures with a structured but randomized information exchange to form a search algorithm with some innovative flair of human search. The central theme of genetic algorithms has been robustness. If somehow artificial systems can be made more robust, costly redesigns can be reduced.

Earlier search and optimization methods couldn't be used to find global minima or global maxima of functions having more than one minimum or maximum. Methods based on calculus can get stuck on a local optimum peak rather than reaching the global optimum peak. Direct search methods seek local optima by moving in a direction related to a local gradient or hill.

There was a need to Optimize functions in a better way i.e. our search methods should be able to reach highest and lowest peaks and not just get stuck on a local minima or maxima.

Genetic algorithms have achieved increasing popularity as researchers have recognized shortcomings of calculus based schemes.

Genetic algorithms were first introduced by John Holland and his students in 1970. De Garis used a sequence of fitness functions to implement Genetic algorithms. A randomly initialized population is evolved by genetic algorithms. Fitness values are used for making new populations in each iteration. John Grefenstette used genetic algorithms to find optimal parameters of genetic algorithm. Sivaraj R. came up with an idea to selectively initialize the first population. The results of genetic algorithm are highly dependent on initial population and population size. Thus, if a genetic algorithm is initialized with selective values rather than random values it can produce better results in lesser time. This approached was followed for quite some time but the problem with this approach is restricted randomness. Randomness is the essence of genetic algorithms. Bramletter proposed a general approach to improve the initialization of GA in 1991. He proposed that initial population should be built by taking best of n randomly chosen chromosomes. The success of these results however is dependent on expertise of user and the nature of problem.

Genetic algorithms start with an initial population filled with random chromosomes. The dimension of each chromosome depends on function. Fitness of each chromosome is evaluated using the function to be optimized. Chromosomes to be picked for reproduction are decided using these fitness values. Individuals are selected from current population based on their fitness values. More fitter individuals have a higher chance of being picked than the individuals with comparatively lower fitness values. Different methods can be used for this selection process. Some of them are Roulette Wheel selection, Tournament selection, Rank based selection etc. Selected individuals are then mated using one of the various crossover techniques like one-point crossover, multi-point crossover, whole arithmetic recombination etc.

Children generated after crossover are then mutated so as to maintain a randomness in the population so that the whole population is not dominated by some particular individuals.

The previous population is then replaced with this new population and this whole process is repeated again and again until the optimum value is reached or until some fixed number of generations.

Various problems which are tough to solve by using traditional calculus-based methods or enumeration

techniques can be easily solved using genetic algorithms. Consider the Rastrigin function as an example which has several local minima. It has several minima but locations of the minima are regularly distributed.

Rastrigin function:

$$10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i)\,;\ -5.12 \le x_i \le\ 5.12$$

Global minimum for Rastrigin function:

$$F(x) = 0, x_i = 0, i = 1\ to\ n$$

One can easily reach values in the range of 10^{-6} (which is nearly optimal) using genetic algorithms.

Genetic algorithms are based on evolutionary ideas of natural selection and reproduction. All evolutionary algorithms including genetic algorithms can find near optimum solutions. A set of unimodal and multimodal functions are usually used as benchmarks for optimization by genetic algorithms. In this paper author has discussed about the optimization of Rastrigin function which is a highly multimodal function.

3.2 Variable chromosome lengths [3]

Crossover in genetic algorithms derives its base from Darwin's theory of evolution. Although chromosomes in human DNA composed of Guanine(G), Cytosine(C), Adenine(A) and Thymine(T), the chromosomes used in genetic algorithms are either made of binary alleles group together as strings or real valued strings which act as weights of neural networks. Chromosomes in human DNA are of same size whereas chromosomes used in genetic algorithms can be of variable length.

Chromosomes of variable length in genetic algorithms are a result of different number of nodes in hidden layer of a neural network. In typical GA, length of chromosomes is known beforehand. It is during the encryption of genotype from corresponding phenotype. Hence, length of chromosomes does not change during crossover or mutation. Concept of variable length chromosomes come into play when traditional genetic algorithms with fixed length chromosomes does not solve problems fairly good with huge numbers of architectural variables. This typical method is somewhat limiting the ideal fitness value to be reached by restraining due to size of chromosome.

In graphs, it is observed in genetic maximization or minimization algorithm, the tangent to infinity to curve is a result of conditions and architectural variables.

We propose an algorithm which processes variable length GA in which the algorithm starts with a less number of hidden neurons in the hidden layer as a result we get a chromosome of short length and then find an about best. Then approximate best solutions are then transferred to further levels with chromosomes of longer length while maintaining diversity among the population.

In combination we use any local refinements such as back propagation to minimize values in local area. Genetic algorithm in combination with Back Propagation (memetic algorithm) optimizes search in such a way that genetic algorithm searches for global optima and hence search for randomly spread values whatsoever but in contrast back propagation searches in locality only. Hence, we get best result when both are used hand in hand.

3.3 Neural Networks [4]

An artificial neural network represents a computer system that is modelled on the human brain and the nervous system. Neural network used here represents supervised learning i.e. neural networks learns with the help of known desired output.

Neural network as a technology grew in the late 1960s but the first artificial neuron was produced in 1943 by neurophysiologist Warren McCulloch and Walter Pits. Neural network grew because of its remarkable ability – "Adaptive learning" i.e. its ability to train and learn in accordance with the data.

Neural network model generally consists of:

- Input nodes
- Output node(s)
- Hidden layer node(s)
- Weights joining these nodes
- An activation function: That decides whether neuron fires for current inputs
- A state variable is also associated to each node.

Neural network processes information in a similar way the human brain does.

It has two basic architectures, feedforward and feedback networks. Feed-forward allows signal to only travel in one way i.e. from input to outputs and does not include feedback loops. It is extensively used for pattern recognition. Feedback allows signals to travel in both direction and uses back propagation for learning.

A neural network is basically made up of many layers. It can be considered a model having at-least two layers- Input layer and the output layer. It can variable number of hidden layers to increase the efficiency.

In a simple neural network model, the input nodes and the output nodes are fixed. Not considering the hidden layers, the network changes its weights in accordance with the output and thus learns by changing its weights through back propagation. Backpropagation generally uses least-squares optimality.

The neural network model described above has hidden layer as a factor. A basic model without hidden layer is called a perceptron that uses back propagation on only single layer weights (connecting input layer to output to nodes). Backpropagation basically uses the idea of calculating error with respect to weights for a given input and change weights by propagating error backwards through the network.

A learning rate η is used a parameter to control rate of change of weights. A high value results in reaching a good result quickly while a low rate reaches an optimal result though it takes time. To obtain better results hidden layers are used which uses back propagation through multiple layers. This model is a multi-layer perceptron.

Back propagation has some drawbacks. It works for simple data set and gives high performance but performance degrades for complex dataset (not linearly separable) as optimal value may miss certain local minima or may be stuck at some local minima or it may keep oscillating near the global minima but may not reach it. Also, backpropagation cannot handle discontinuous optimality criteria or discontinuous node transfer functions which restricts its area of operation to simple optimality criteria.

3.4 Combining Genetic Algorithms and Neural Networks: The Encoding Problem [5]

Philipp Koehn of The University of Tennessee, Knoxville published the paper in 1994 highlighting the importance of Genetic Neural Networks as both Genetic Algorithms and Neural Networks received great acclaim in computer science research community, and how they combined together demonstrate powerful problem-solving ability.

Neural networks combined with feed forward validation along with backpropagation show a great new computation model inspired by nature, the power of self-organization of interdependent units. This all happens with an all different approach from classical computer programming, the non-declarative programming paradigm, feeding large amount of training patterns which are instances of the problem to a training framework. These systems are more fault tolerant and adaptable to new data, whereas traditional programs are not.

Genetic Algorithms being based on principals of selection, crossover and mutation start from a global and truly random search space and evolve that set into a global optimal solution taking into account all the variation. Genetic Algorithms are a good object for parallelization as the crossover, mutation and evaluation of individuals are independent of each other, which will allow high population sizes without increase in time.

The synergy of GANN greatly improves the learning time for networks, results in greatly boosted accuracy and overcomes the shortcomings of backpropagation, where initial parameters greatly determine the success of training process. By combining these two, the genetic algorithm finds initial parameters and applies a natural algorithm which is very successful, as we humans have evolved in a similar way on earth.

Combining Genetic Algorithms and Neural Networks involves encoding the information about the neural network in the genome of the genetic algorithm. Starting with generating a number of random individuals in the beginning, the parameter strings are evaluated to return a fitness value, which determines their rank in the pool. Encoding the Neural Network is

the main focus of the paper and various strategies are discussed. Incorporating the network size in evaluation is significant to finding efficient networks. The Problem of Overfitting is another such issue that results after excessive training, decreasing generalization. Using the combination of search strategies of GA and NN, we get the best of both worlds. GA performs a more global search than NN with Back Propagation, but Back Propagation reaches an optimal solution more precisely.

4 METHODOLOGY

4.1 NEURAL NETWORKS

Neural network was used to evaluate few datasets.

4.1.1 IRIS

Iris is the most common dataset which is used in pattern recognition literature. Fisher's Iris dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant. Classes are linearly separable.

Neural network was implemented using NumPy and Theano and also using Keras with TensorFlow as backend.

We divide data into 60% training, 30% testing and 30% validation. Input layer has 4 neurons. Output layer has 3 neurons. A hidden layer having 3 neurons was used.

Results:

- Training accuracy: 98.6%
- Testing accuracy: 96.77%
- Validation accuracy: 96.77%

4.1.2 PIMA

Pima dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. Objective of this dataset is to predict based on diagnostic measurements whether a patient has diabetes.

Neural network was implemented using Keras with TensorFlow as backend. Dataset was divided into 67% testing and 33% training and some samples of testing set were used for validation of the dataset. Input layer has 8 neurons. Output layer has 1 neuron. A hidden layer having 500 neurons was used.

Results:

- Training accuracy: 100%
- Testing accuracy: 71.03%
- Validation accuracy: 70.67%

4.1.3 MNIST

MNIST is a simple computer vision dataset which consists of images of handwritten digits.

Dataset was divided into 70% training and 30% testing dataset and was implemented using NumPy with Theano as backend. Input layer has 727 neurons. Output layer has 10 neurons. A hidden layer having 500 neurons was used.

Results:

- Training accuracy: 99%
- Testing accuracy: 98%
- Validation accuracy: 98.5%

4.1.4 CARDS

This dataset is used to aid the detection of credit card fraud with the help of neural network.

Dataset is divided into 70% training and 30% testing data and was implemented using NumPy with Theano as backend. Input layer has 23 neurons. Output layer has 2 neurons. A hidden layer having 500 neurons was used.

Results:

- Training accuracy: 94%
- Testing accuracy: 93.2%
- Validation accuracy: 92%

4.2 BINARY CHROMOSOMES

In the way of getting to know genetic algorithms from a closer view, we implemented few functions using binary based chromosomes.

In High Conditioned Elliptic Function, we found the minimum value in approximately 96 generations for a Population size of 200 and 10 dimensions.

Rosenbrock function reached its minimum value in about 100 iterations on average, for D = 5 and Population size of 200.

Ackley's Function reached minimum value of $4.4408920985 \times 10^{-16}$ for D = 10.

Using AND & OR after one-point crossover and not using the traditional method of swapping the strings after one point and between two points, better results were obtained, i.e. the run time of the algorithm decreased drastically.

After doing lot of translations and rotations over the functions and performing the crossover operations, this approach showed better results, but it hasn't been tested it for the pairwise-t test yet.

So, for binary chromosomes, three types of crossover operations were done:

1. Simple one-point crossover or two-point crossover.
2. One-point crossover using AND & OR.
3. Translation and rotation of the functions and then one-point cross over with AND & OR.

First, we generated random binary strings and formed the population array with those random binary strings. Then for natural selection we used different techniques:

a) Fitness proportionate selection - Roulette wheel selection
b) Tournament selection
c) Rank based selection

In Darwinian theory of evolution, nature tends towards the 'survival of the fittest'.

In the first selection procedure we used the concept of mating pool, in such a way that the fitter individual occurs more number of times than less fit individual, and hence the probability of selection of fitter individual is more which is in analogy with natural selection in living beings.

In tournament selection, we at first took two individuals and found the fitter among them to get one parent and then repeated the process to get the other parent. This can even work for negative fitness values.

In rank based selection we assigned ranks to the individuals on the basis of their fitness. And then used cumulative sum and for selection we used binary search. This technique does not need mating pool, so there is no extra space used.

And for crossover we applied one point and two-point crossover. And in the modified version, we applied AND & OR after finding a random number between 0 and length of chromosome (both exclusive).

And mutation is something that is generally seen in nature, humans evolved from apes (losing tails and stuff), in analogy mutation is done over those binary chromosomes.

Crossover and mutation both have different rates, mutation is generally rare so we made 2% mutation rate and crossover rate to be 80%.

4.3 GENETIC ALGORITHM IMPLEMENTATION FOR POPULATION HAVING CHROMOSOMES MADE UP OF REAL VALUES

Genetic Algorithms were explored by trying to optimize some benchmark functions. Results for the same are presented below.

4.3.1 Initialization

The initial population was created by filling each chromosome with 'D' random values lying in the range $[-100,100]$. Here 'D' denotes the dimension for which the function was analysed. A population size of 150 was chosen.

4.3.2 Fitness evaluation

Each individual's value was calculated by putting them in the function under observation. These values were stored in a 'values' array.

4.3.3 Selection

The 'Values' array was sorted in decreasing order and the corresponding indices were used as rank for individuals. The array was sorted in decreasing order since we had to find the global optimum value. So, the lower values were given a higher rank in this way. A combination of rank based selection and tournament selection was used to pick parents for reproduction. Two random parents were picked based on their ranks and the one which gave a smaller value was chosen as parent A. This process was repeated again to find parent B.

4.3.4 Crossover

Two parents picked in step 3 were then crossed by using Whole-Arithmetic recombination method. In this method a random real value 'x' in the range [0,1] is chosen and two children are found as follows:

$$childA_i = x \cdot parentA_i + (1 - x) \cdot parentB_i$$

$$childB_i = (1 - x) \cdot parentA_i + (x) \cdot parentB_i$$

$$1 \leq i \leq D$$

4.3.5 Mutation

The children generated were then altered by a small amount at a rate of 20% for first 5000 generations and 2% for the rest generations. Mutation was kept high for first 5000 generations so as to approach the optimum value quickly. After 5000 generations mutation rate was lowered so as to approach the optimum value with greater precision. A tradeoff with speed was made.

4.3.6 Replacement

The previous population was then replaced with these new off-springs and the whole process was executed for 40000 generations

Any variations in number of generations and/or mutation rates are specifically shown below.

4.3.7 Results for some CEC 2015 Benchmark functions [6]

4.3.7.1 Rastrigin function:

$$f(x) = \sum_{i=1}^{D} (x_i^2 - 10\cos(2\pi x_i) + 10)$$

$$-100.0 \leq x_i \leq 100.0$$

Global minimum for this function is $f(x) = 0$ and it is obtained when each gene of chromosome is '0'. After running 40000 generations, the minima obtained was 1.32×10^{-8} which is nearly optimal.

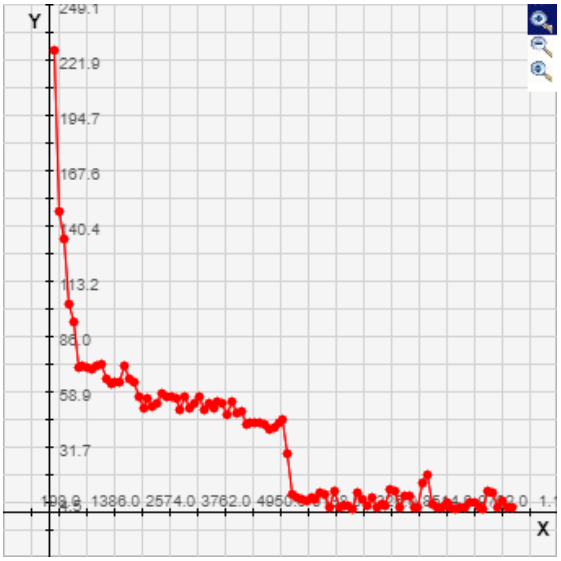

Figure 1. Graph showing a plot of Number of generations (X) vs Average function value (Y)

4.3.7.2 Weierstrass Function:

$$F(x) = \sum_{i=1}^{D} \left(\sum_{k=0}^{kmax} \left[a^k . \cos\left(2\pi b^k (x_i + 0.5)\right) \right] \right) - D \sum_{k=0}^{kmax} \left[a^k . \cos(2\pi b^k \cdot 0.5) \right]$$

$$a = 0.5, \qquad b = 3, \qquad kmax = 20$$

Global minimum for this function is 0 and it is obtained when each gene of chromosome is '0'. After running 5000 generations, the minima obtained was 3×10^{-3} which is nearly optimal.

4.3.7.3 High Conditioned Elliptic Function:

$$F(x) = \sum_{i=1}^{D} (10^6)^{\frac{i-1}{D-1}} \cdot x_i^2$$

Global minimum for this function is 0 and it is obtained when each gene of chromosome is '0'. After running 40000 generations, the minima obtained was 2.90×10^{-7} which is nearly optimal.

4.3.7.4 Katsuura function:

Global minimum for this function is 0 and it is obtained when each gene of chromosome is '0'. After running 10000 generations, the minima obtained was 3.08×10^{-7} which is nearly optimal.

4.4 Selection and Crossover of Variable Length Chromosomes

Genetic Algorithms work on one basic principle of searching a truly global search space and optimizing the results in each subsequent generation to reach a global optimal value. In our problem, we are using this fundamental concept related to GA to train our neural networks faster and better. Only one hidden layer is taken in account in the model for simplicity and efficiency. Encoding the chromosomes of a genome with Neural Networks incorporates number of hidden nodes in the one hidden layer, which then determines the number of weights in the network (also the length of the chromosome). This seems fitting, as then only we are using a global search space. But the problem arises during crossover; as networks with different number of hidden units can't be crossed over with each other.

To solve this problem, different selection techniques are incorporated. One of them finding both parents that satisfy the condition of having equal number of hidden units, resulting in no anomaly during crossover. This requires alteration of the used selection procedure.

Selection gives two parents, a father and a mother. The father is selected using the normalized rank based selection from the population. The ranks evaluated according to fitness returned by the artificial neural network, i.e. the accuracy of the feed forward training of said network. The number of hidden nodes in the first parent, father is used to find suitable match in a subset of the original population which satisfy

$$\{x \in population \mid no\_of\_hidden\_nodes(x) = no\_of\_hidden\_nodes(father)\}$$

If the population contains elements having number of hidden nodes varying from 1 to n, total number of such subsets come out to be n, labeled $A_1, A_2, \dots, A_n$, which are mutually exclusive and exhaustive. If $i = no\_of\_hidden\_nodes(father)$, then we apply our normalized rank roulette wheel selection procedure on the set A_i. And the result is selected as the second parent, the mother. This allows us to apply crossover on both parents with ease.

4.5 Structural Mutation

A different kind of mutation called structural mutation is also applied to the population so as to increase its randomness. Structural mutation changes the number of hidden neurons corresponding to a particular chromosome. Thus, structural mutation will change the number of hidden neurons and thus the number of weights in the length of the chromosome. If the number of hidden layer or the length of chromosome is to be decreased, then we can arbitrarily remove some weights from the chromosomes. If the number of hidden neurons are to be increased the new random weights can be generated between specified values (-1,1) and can be added to the present chromosome.

Structural mutation prevents early convergence and randomizes the data so that all possible local optimum values can be explored to find global optimum value.

4.6 Regularization

A common problem faced in training neural networks is overfitting, where the model becomes overly complex by learning too well the details from training data, but doesn't generalize and hence performs poorly for the testing data. Since, the objective is to make good predictions for all kinds of data, seen or unseen, this has to be dealt with, and regularization is the answer.

Some common techniques [7] of regularization include:

- Dataset augmentation
- Early stopping
- Dropout layer
- Weight penalty L1 and L2

We have used the standard method, weight penalty which tries to reduce the weights by imposing a penalty on the square value of weight (L2 norm) or absolute value of weight (L1 norm), hence reducing the model size.

4.7 Clustering Algorithm (K-Means)

K-means is a simple unsupervised learning algorithm that solves clustering problems. This is used when we have unlabeled data (i.e., data without defined groups). The aim of this algorithm is to form groups in the data, where the number of groups (K) is specified by the user. The algorithm follows a simple and easy way to compute cluster centers [8]. It is generally run till the clusters reach saturation i.e., there is no change in the cluster centers or it can be made to

run for a fixed number of iterations. It is performed in following steps:

1. Take k random data points and mark them as k clusters for first iteration.
2. For each subsequent iteration, take the mean of each cluster to find new cluster centers.
3. Repeat the steps for next iterations till the clusters reach saturation or for run for a specified number of times.

The K-means clustering algorithm uses iterative refinement to produce a final result [9].

4.8 Multi-objective optimization

Multi-objective optimization means achieving an optimal solution for a problem involving more than one objectives to be optimized [10]. In Multi-objective optimization optimal decisions are taken by doing trade-offs between various objectives that are conflicting in nature.

For non-trivial multi-objective optimization problems all of the objectives cannot be optimized simultaneously. Here the objectives are said to be conflicting and some trade-offs are made.

A solution for multi-objective optimization problem is called pareto-optimal if it is better than all other feasible solutions. The pareto-optimal is the best-known solution with respect to all constraints and objective functions and cannot be improved in one objective without degrading some other objectives.

In the domain of evolutionary algorithms there are many test functions that can be used to study multi-objective optimization [11]. Some of them are Zitzler-Deb-Thiele's test functions (ZDT1, ZDT2, ZDT3, ZDT4, ZDT16), Srinivas and Deb test function and Tanaka test function.

What we have done so far is optimized our network on the basis of a single objective and that is error. We have only tried to reduce the error so far and hence this problem of optimization was single objective. In Multi-objective optimization there is a need to optimize various objectives like error, Number of hidden neurons in each hidden layer, Number of hidden layers, Population size, Activation function and various other input and output parameters. Results given by multi-objective are certainly better than single objective optimization. A pareto set is generated which contains each of the above values after considering all the conflicting objectives and then doing the optimal trade-off for them.

5 Initial Results

Performance of neural networks was enhanced by using Genetic algorithms. Applying back-propagation on neural networks to train the PIMA dataset yielded the following results:

- Testing accuracy – 75.03%
- Validation accuracy – 77.67%

Much Better results were obtained for the network by using Genetic algorithms instead of back-propagation to train the PIMA dataset. Through use of genetic algorithm, the model was able to outperform our simple model which used a hidden layer having 500 neurons. A modest population size of 600 was used. Even better results were obtained for smaller population sizes. One hidden layer with 20 neurons was used to obtain the results:

- Testing accuracy – 76.53%
- Validation accuracy –77 %

Similar results were obtained by changing the number of neurons in the hidden layer which generated variable length chromosomes.

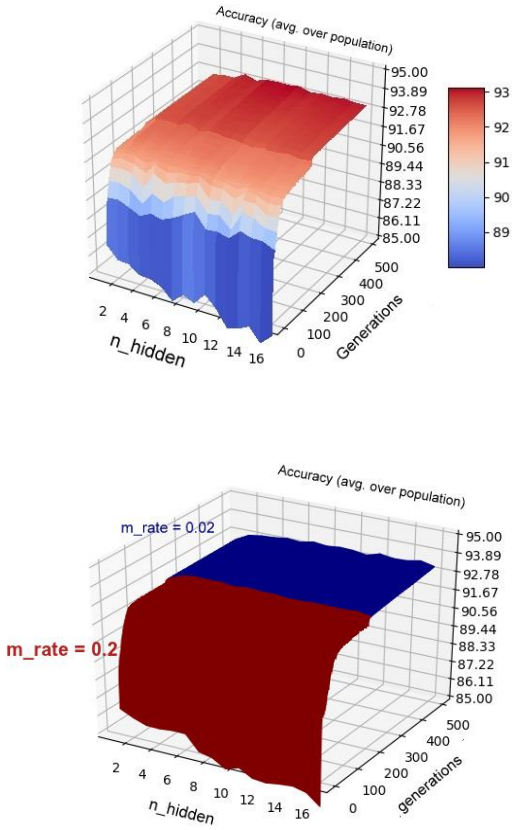

Figure 2. Heat map and effect of change in mutation rate on Cost (MSE) for hybrid evolution

6 Conclusion and Future Work

It was observed that on the dataset (PIMA), the conventional methods (back propagation) of neural networks gave results that were outperformed by a model using both the technologies of neural network and genetic algorithm.

Use of evolutionary algorithms for training data increased the testing accuracy tremendously even the use of number of neurons in the hidden layer were reduced from 500 to 20.

Validation accuracy was also increased. Thus, it was shown that adding domain-specific knowledge into genetic algorithm can enhance the performance of neural networks.

The model constructed using both the technologies has the added advantage of being able to work on nodes with discontinuous transfer functions and increases the efficiency when applied to real world applications which are complex.

The work described here only shows the strength of using genetic algorithm over back propagation. Results can be further increased by using a combination of genetic algorithm and back propagation thus leading to the formation of a hybrid model. The report will be further extended to optimize multiple hyperparameters including number of hidden neurons, activation function, error rate as well as various input and output parameters thus turning the problem from single-objective to multi-objective.

Optimizing many hyperparameters will help in achieving better results and further enhance the model.

7 REFERENCES

[R. Kemp, "An introduction to genetic algorithms for neural networks," [Online]. Available: https://www.phase-trans.msm.cam.ac.uk/2003/MP9/MP9-5.pdf.

[R. Garg and S. Mittal, "Optimization by Genetic Algorithm," April 2014. [Online]. Available: https://www.ijarcsse.com/docs/papers/Volume_4/4_April2014/V4I4-0471.pdf.

[I. Kim and O. de Weck, "Variable chromosome length genetic algorithm for progressive," 7 January 2005. [Online]. Available: http://strategic.mit.edu/docs/2_7_SMO_VCLGA.pdf.

[D. J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms," [Online]. Available: http://www.ijcai.org/Proceedings/89-1/Papers/122.pdf.

[P. Koehn, "Combining Genetic Algorithms and Neural Networks: The Encoding Problem," December 1994. [Online]. Available: http://homepages.inf.ed.ac.uk/pkoehn/publications/gann94.pdf.

[J. J. Liang, B. Y. Qu, P. N. Suganthan and Q. Chen, "CEC 2015 Competition on Learning-based Real-Parameter Single Objective Optimization," Zhengzhou University, November 2014. [Online]. Available: http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared%20Documents/CEC-2015/Learning-Based%20Single%20Objective%20Optimization/Definitions%20of%20%20CEC2015%20benchmark%20learning-based%2020141228.pdf.

[C. Scheau, "Regularization in deep learning," Medium.com, 16 November 2016. [Online]. Available: https://chatbotslife.com/regularization-in-deep-learning-f649a45d6e0.

[A. Mordvintsev and A. K, "Understanding K-Means Clustering," 2013. [Online]. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_kmeans/py_kmeans_understanding/py_kmeans_understanding.html.

[A. Trevino, "Introduction to K-means Clustering," DataScience.com, 6 12 16. [Online]. Available: https://www.datascience.com/blog/introduction-to-k-means-clustering-algorithm-learn-data-science-tutorials.

["Multi-objective optimization," [Online]. Available: https://en.wikipedia.org/wiki/Multi-objective_optimization.

[K. Amouzgar, "Multi-Objective Optimization using Genetic Algorithms," 30 5 2012. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:570751/FULLTEXT01.pdf.