The Implementation of Genetic Algorithms on Neural Network & Experiment on Marks Prediction Dataset

Utkarsh Marwaha
Research School of Computer Science,
Australian National University,
utkarsh.marwaha@anu.edu.au

Abstract

Artificial neural networks and genetic algorithms are well renowned machine learning concepts that have been intricately modelled on natural biological processes. Many researchers opine that this similarity predisposes them to be effective at solving optimization problems [1]. Notwithstanding their disjoint usage in the past, the combination of genetic algorithms and artificial neural networks continue to widen the spectrum of solvable challenges. In recent years, the duo has found frequent application in the realm of non-linear process optimization wherein an ostensible analytical solution is either completely absent or heavily camouflaged [5]. In this paper, I have chosen a dataset that comprises partial grades from the teaching session of large undergraduate computer science courses at the University of New South Wales. The premise is set under a simple backpropagation neural network which attempts to predict the final examination grades of students based on their current performance of the semester [3]. Initially, I've applied basic preprocessing, domain-oriented outlier detection techniques and k-fold validation to enhance the accuracy of the network. Later on, I've employed the genetic algorithms to train the classical neural network by fine tuning its weights and biases. The ultimate aim is to optimise our model's prediction accuracy.

Keywords: Neural Network, Backpropagation, Outliers, Genetic algorithms.

Introduction

Predicting final exam grades is a highly non-linear process because the data consists of a lot of random noise. This is manifested by missing scores for quite a few assessment items. Heeding these hurdles, the data of such a problem is best suited to be solved by an artificial neural network optimised by genetic algorithms. Unlike other optimisation techniques, genetic algorithms try to locate the optimal solution by applying basic data processing methods like string manipulation and random number simulation. The feasibility of these elementary yet effective methods lessens the computation burden on the overall training process.

Genetic Algorithms

A genetic algorithm can be described as an all-weather random search technique which is frequently employed to solve optimization problems. It offers a unique way to train artificial neural networks since it's not susceptible to being stuck in the local minima [6]. Genetic algorithms were built upon the Darwinian evolution process which advocates the much acclaimed philosophy of "survival of the fittest". The main idea of the algorithm is to operate on a set of features known as chromosomes. A vast multitude of such features collectively constitute what is known as the population. Given the Darwinian origins of these algorithms, defining an apposite criterion of fitness becomes critical for establishing the right parameters for the optimization process.

Listed below are the operations employed to solve the problem mentioned in this work.

Selection

Selection allows us to find out the fittest chromosomes within the current population. The crucial step is to produce a transit generation which comprises chromosomes from the current layer in direct ratio of their fitness score. This ensures that the next generation is a kin to its fittest predecessors. This process of natural selection essays a major role in perpetuating only the fittest chromosomes across generations.

Crossover

The procreation of two offspring from two parents is known as a crossover. It is brought about by the exchange of genetic blocks between the two parents at randomly chosen crossover points.

Mutation

Mutation is used to randomly introduce new alleles in the population. These new alleles are introduced through a random flipping in the bit sequence of a chromosome (string). The role of mutation is to prevent premature convergence in the population by disrupting homogeneity and enhancing diversity. In the broader scheme of things, this guarantees that we find a global minimum (fitness peak) instead of getting stuck in a local minima (substandard fitness). Having mentioned the benefits of mutation, one should be extremely careful at applying it too frequently since that could lead to an unsustainably unrecognizable pair of adjacent generations spelling catastrophe for the survival of the overall population.

Outlier detection and eradication

The accuracy of a backpropagation neural network is greatly influenced by the presence of outliers in the data that is used to train it. Consequently, detection and removal of outliers is believed to considerably improve the generalizability of our model. Over the years, researchers have devised methods like the Absolute Criterion Method, Least Means Squares method and Least Trimmed Squares method among several others to find and eradicate outliers from their dataset [7]. While some of these have certainly proved to be effective on artificially noisy training datasets, the real world anomalies aren't picked up very easily using such techniques [7]. Keeping this in mind, the Bimodal Removal Distribution Technique was composed to tackle real world noisy data sets.

Method

Feature Selection

Due to the limited size of the dataset and the high count of missing values in our data, I decided to include all the numeric columns pertaining to a student's grades as features in my model. There are 10 such features all of which are being factored in by the model before predicting the student's final exam scores.

	lab2	tutass	lab4	h1	h2	lab7	p1	f1	mid	lab10
0	2	3	2.5	19.5	.	2.5	11	.	33	2.5
1	3	5	2	20	17	.	8	.	27	2.4
2	3	3	3	10	.	2	.	.	9.5	2.4
3	2	3	2	20	19.5	2	15.5	.	21	.
4	2	3.5	1.5	19	15.5	2	17.5	.	13	2.5

Fig 1: Features used to inform final exam scores

Preprocessing the data

The initial dataset was in the form of text and it had several missing values. In order to be able to feed it to a neural network, I had to apply a numeric transformation on it. The missing values were replaced by zeroes.

	lab2	tutass	lab4	h1	h2	lab7	p1	f1	mid	lab10
0	2.0	3.0	2.5	19.5	0.0	2.5	11.0	0.0	33.0	2.5
1	3.0	5.0	2.0	20.0	17.0	0.0	8.0	0.0	27.0	2.4
2	3.0	3.0	3.0	10.0	0.0	2.0	0.0	0.0	9.5	2.4
3	2.0	3.0	2.0	20.0	19.5	2.0	15.5	0.0	21.0	0.0
4	2.0	3.5	1.5	19.0	15.5	2.0	17.5	0.0	13.0	2.5

Fig 2: Features after numeric transformation (missing values replaced by 0s)

As can be seen from Fig 2, each column represents the batch's score in a particular assessment. The maximum possible score offered by each assessment is distinct, e.g. the maximum one can score in "tutass (5)" is different from what one can get in some other assessment, say "lab2 (3)". This made it critical for me to normalise the data i.e. transform the scores to a common scale, whilst preserving the difference in the ranges of values. Without normalisation, the scores from "tutass" would have intrinsically influenced the result more than some lowly-valued attributes like "lab3" due to its larger value. A larger influence achieved this way didn't necessarily mandate that "tutass" was a richer predictor of the final exam grades.

Out of the umteen choices one has in normalising numericals such as exam scores, I decided to use the min-max scaler that transforms the given score into a ratio of the marks achieved to the maximum attainable scores between 0 and 1.

$$Zi = \frac{Xi}{max\,(X)}$$

where Xi is the actual score achieved by a student in an assessment piece and max(X) is the maximum attainable score in that assessment.

	lab2	tutass	lab4	h1	h2	lab7	p1	f1	mid
0	0.666667	0.6	0.833333	0.975	0.000	0.833333	0.550	0.0	0.733333
1	1.000000	1.0	0.666667	1.000	0.850	0.000000	0.400	0.0	0.600000
2	1.000000	0.6	1.000000	0.500	0.000	0.666667	0.000	0.0	0.211111
3	0.666667	0.6	0.666667	1.000	0.975	0.666667	0.775	0.0	0.466667
4	0.666667	0.7	0.500000	0.950	0.775	0.666667	0.875	0.0	0.288889

Fig 3: Features after normalisation

Classic Backpropagation

The choice of topology i.e. the number of layers and the number of neurons in each layer is an important one to make whilst designing a neural network. Depending on whether our network is too deep and dense or too shallow and sparse, the adjustment of weights can vary significantly. I decided to produce a four-layered network with 10 neurons in the initial layer, 25 neurons in the first hidden layer, 19 neurons in the second hidden layer and a single output neuron. Apart from the topology, I experimented with the network to find the best learning rate for convergence and optimal training time. Likewise, I tried different activation functions before settling down with (ReLu) because it's computationally superior to most of its alternatives.
Each epoch's performance is computed by finding the discrepancy between the actual and expected values through the Mean Square Loss criterion. Whilst propagating backwards, we reset the gradients to prevent unnecessary accumulation. This is followed by a backward pass to compute loss gradients with regards to every adjustable parameter. Finally the optimiser (Adam in this case) is used to update the parameters accordingly.

Cross Validation

Validating one's model is always a good idea because it assures us about its ability to retain low bias and variance. Initially, I held out a portion of the training data to make predictions. Even though this method is computationally cheap, it didn't give me any control over the variance in the results because it's impossible to predetermine which

data points will be included in the hold-out sample as opposed to the training sample. As a result, the results could be entirely different for different combinations of training and hold-out samples. Apart from the risk of an unknown variance, I also ran the risk of underfitting our model because of its small size. To tackle this problem, I used the K-fold cross validation technique whereby the train-test split method is repeated k times. In each iteration, one of the k subsets is chosen to be the validation set and the remaining k-1 subsets are used for training. This way, we're able to significantly diminish the bias since the entire data is used both for training as well as validation.

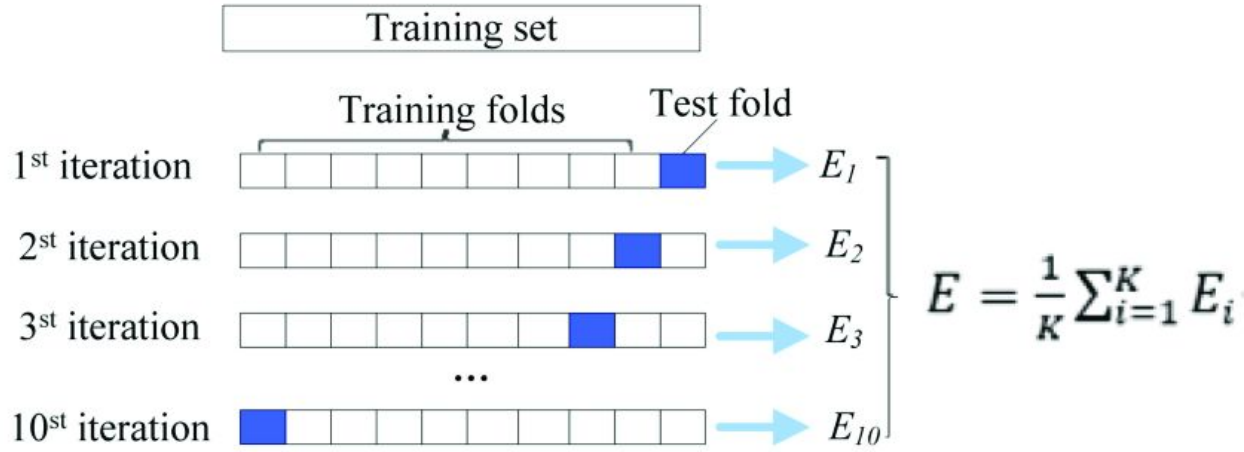

Fig 4: K-fold cross validation [4]

Bimodal Distribution Removal

The first step towards discerning the nature of outliers within our training dataset was to investigate the frequency distributions of errors for all the patterns. In earlier epochs when the model was largely untrained, the error distribution was found to have high variance. Later on, however, I observed that the losses plunged quite sharply. However, as can be seen from the red coloured distribution shown in Fig 5, there exists some patterns with relatively high errors. This occurrence creates a bimodal error distribution with the low error peak containing patterns the network has learnt well, and the high error peak containing the outliers [7].

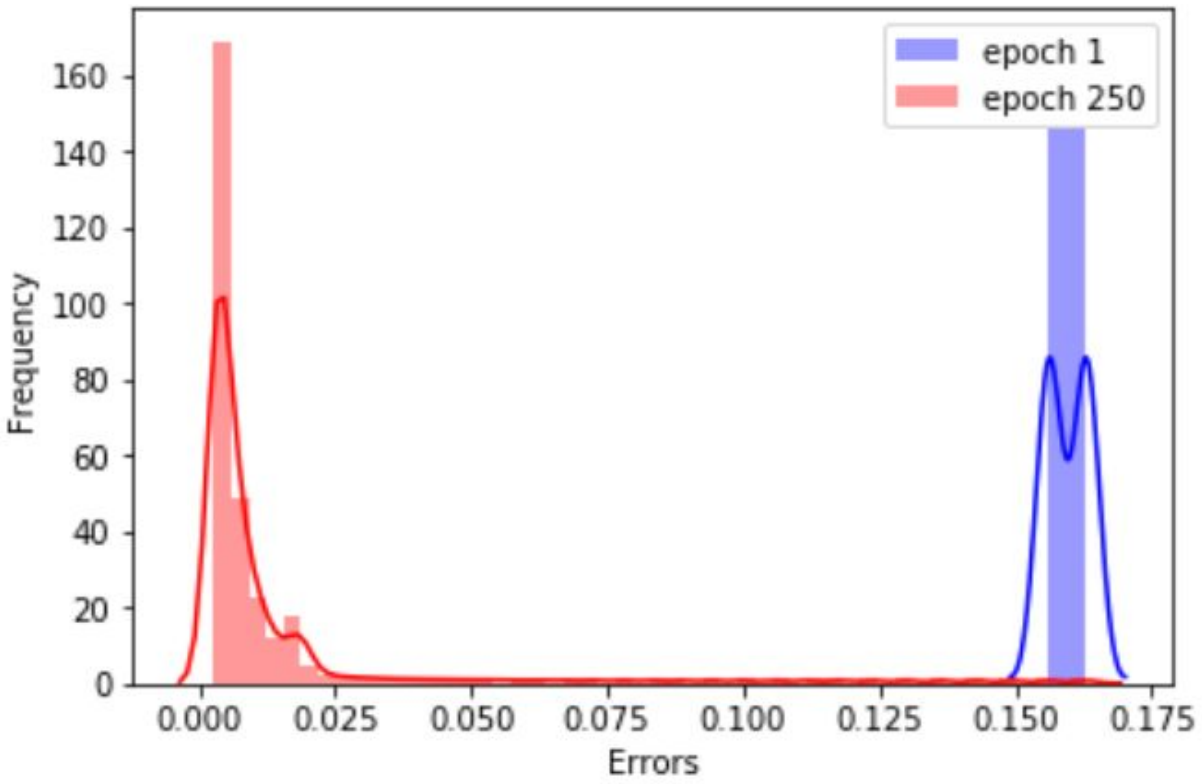

Fig 5: Error distribution at epochs 1 and 250 respectively

In selecting which patterns to remove as outliers, we first need to compute the mean of all errors in the training data set denoted by μ_{ts}. Next up we pool out all those patterns with error greater than μ_{ts}. Then we calculate the mean μ_{ss} and standard deviation σ_{ss} of this pool [7]. Using these two empirical quantities, we decide which outliers are to be permanently eradicated from our training dataset. Those patterns with error greater than or equal to $\mu_{ss} + \alpha \ \sigma_{ss}$, where $0 \leq \alpha \leq 1$ are binned. Every run of BDR would get rid of a few outliers from the high error peak and nudge it rightwards.

Genetic Algorithms

I designed my genetic algorithm to be implemented in the following 6 steps: -

1. Randomly instantiate the population
2. Every chromosome is assigned a fitness score based on the pre decided criteria
3. Choose the fitness chromosomes based on a minimum subsistence threshold.
4. Exchange the features between the selected portion of chromosomes. This process is also called crossover.
5. Choose a few random chromosomes and apply certain random mutations to their features to increase diversity

6. Loop back to Step 2 and carry onto future generations.

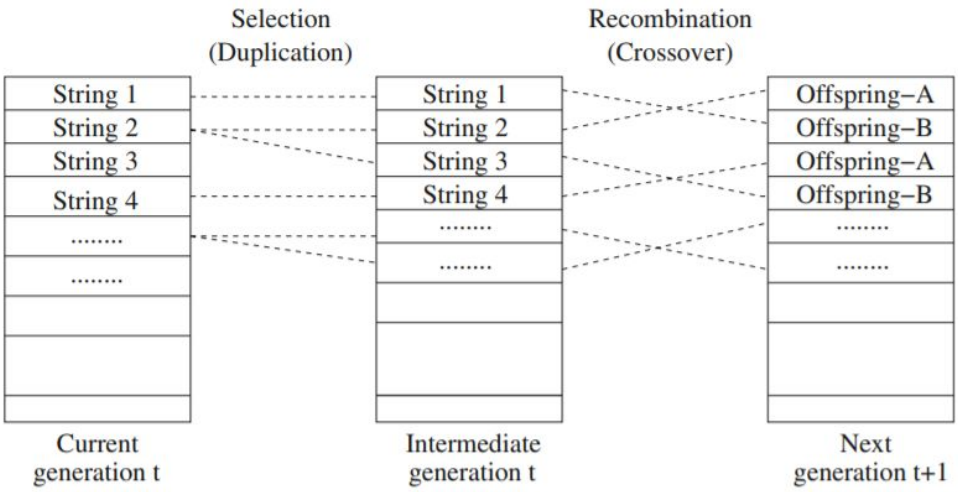

Fig 6: The selection and recombination process carried out at each generation

Genetic algorithms fall under a versatile umbrella of algorithms which can be used to both train as well optimise an artificial neural network. In our case, every single chromosome consists of all the weights of the ANN.

Results and Discussion

Classic Backpropagation

On training the data with backpropagation on a classic feedforward neural network, we get an error value of close to 0.00323 by the end of 500 epochs. As can be observed from the loss comparison shown below, the error plunges quite early on during the training phase and then continues to show incremental improvements before plateauing near the 500 epoch mark. On evaluating the model with unseen data, we get an error value of about 0.00347 which is marginally greater than the eventual training error. This suggests that if we had trained any longer, our model would have likely overfit the given data.

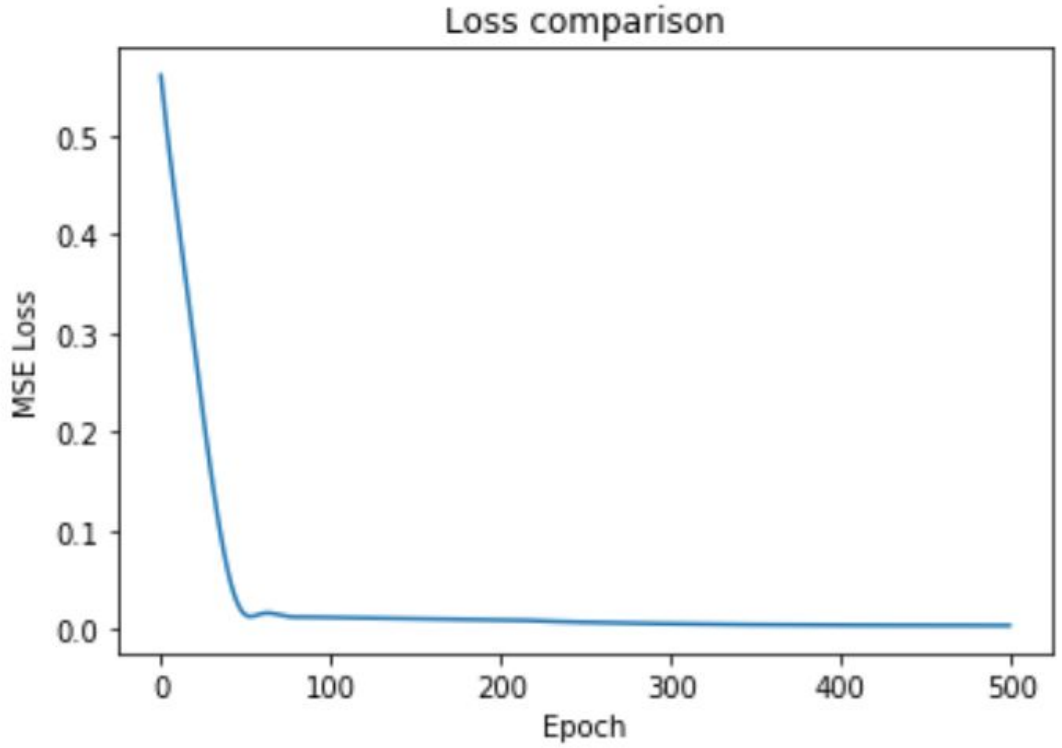

Fig 7: Mean Square Error progression over 500 epochs

Cross Validation

On training the data with K-fold (optimal K value = 3) cross validation on a classic feedforward neural network, we get an error value of close to 0.00567 by the end of 500 epochs. As can be observed from the loss comparison shown below, the error decreases exponentially during the training phase before plateauing near the 500 epoch mark. On evaluating the model with unseen data, we get an error value of about 0.00622 which is marginally greater than the eventual training error. This suggests that if we had trained any longer, our model would have likely overfit the given data.

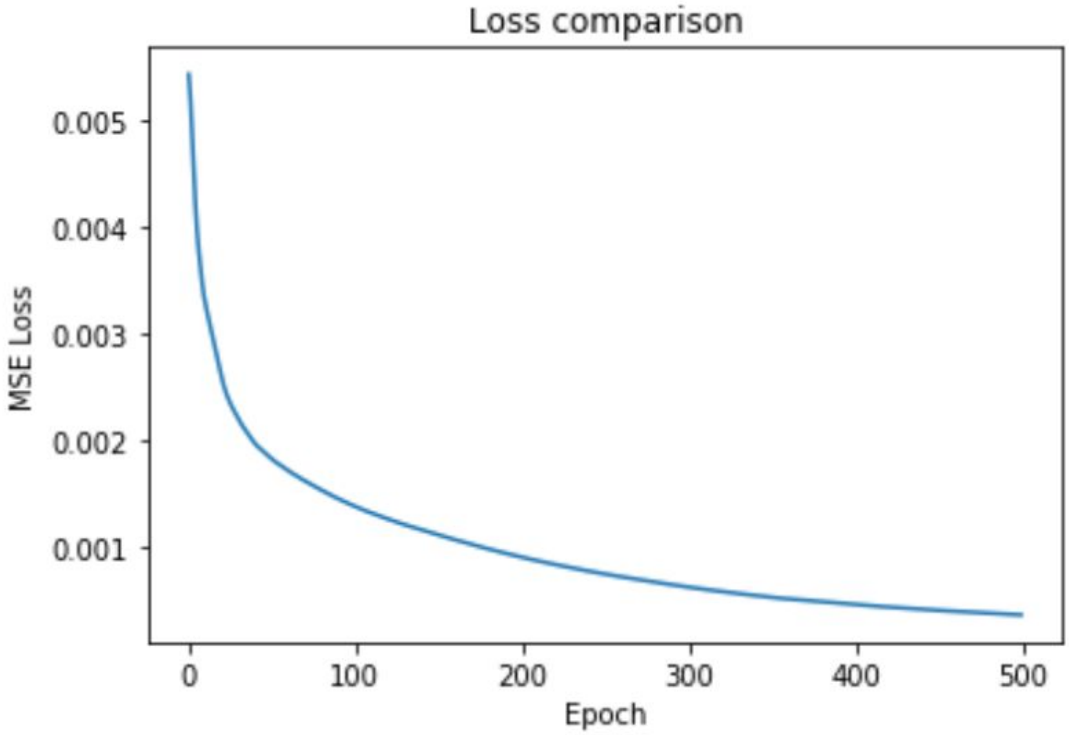

Fig 8: Mean Square Error progression over 500 epochs

Genetic Algorithm

On training the neural network under the genetic algorithm for 100 generations with a crossover rate of 0.8 and a mutation rate of 0.002, we get an error value close to 0.00165 after 500 epochs. On evaluating the model with unseen data, we get an error value of about 0.0010 which is better than the previous attempts. Even though the improvement is marginal, the training time for the network is significantly reduced as compared to other approaches.

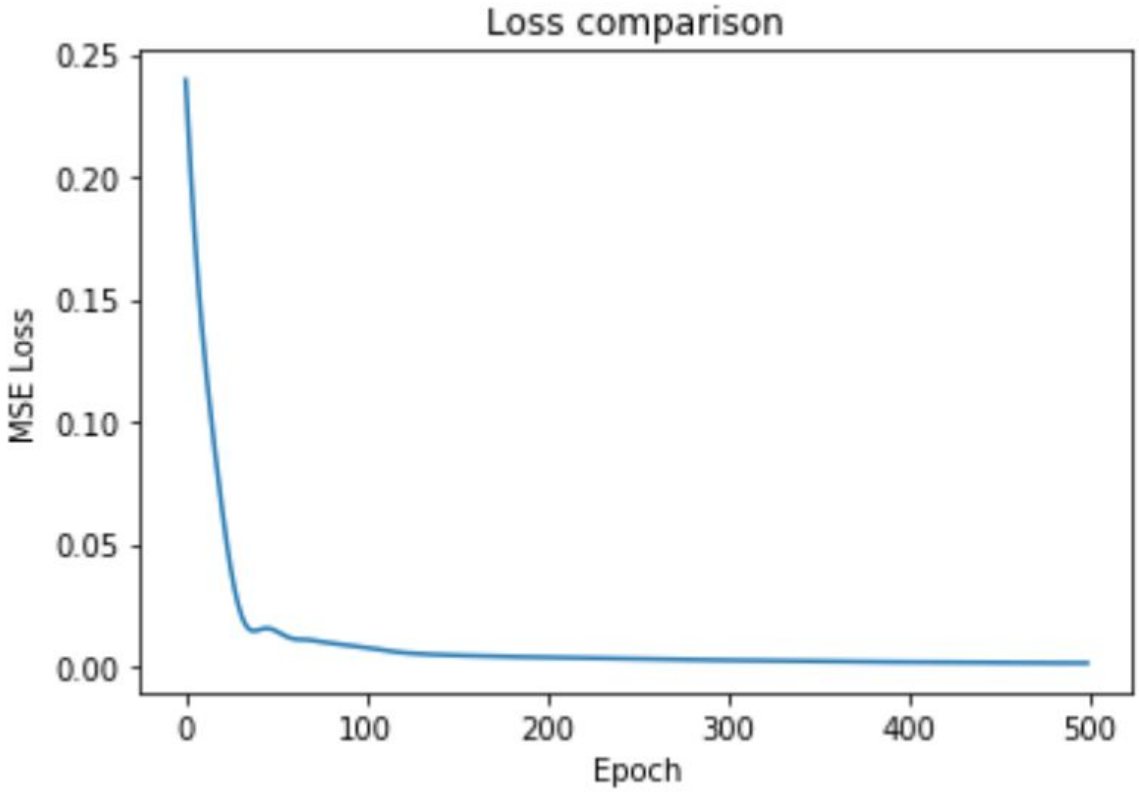

Fig 9: Mean Square Error progression over 500 epochs

Limitation

One of the major limitations of genetic algorithms that was exposed through this piece of implementation was that we need a sizable population to produce accurate results. Without a considerably large initial population, the genetic algorithm will never have enough elbow room to find out the optimal solutions. Before the implementation, python's sensitivity towards tolerating random changes in the representation of chromosomes had fleetingly threatened the accuracy of the final results but it was handled without much trouble in the end. Another major limitation of my algorithm is its heavy reliance on the fitness function to pave the way for subsequent generations of the population. Even the slightest of tinkering with the fitness function could lead to a massive discrepancy between the actual and optimal solution. Besides finding an appropriate fitness criteria and population size, other factors like crossover and mutation rate too can disproportionately influence the accuracy of the final results.

Conclusion and Future Work

Heeding the above mentioned limitations of my work, I would like to test the efficacy of genetic algorithms in optimising the performance of a neural network by applying it to more non-linear as well as purely analytical problems. This would allow me to appreciate its primary tactics in greater depth. Whilst attempting to learn from this comparison, I would also want to test different fitness criteria, crossover and mutation rates so that I can intricately understand the impact that they bear on the overall convergence across several generations.

References

1. Smith K. A., Gupta J. N. D., Neural networks in business: techniques and applications for the operations researcher, Computers & Operations Research 27, 2000, 1023-1044.
2. F. E. Grubbs, “Procedures for detection outlying observations in samples,” Technometrics (1969).
3. Gedeon, T.D. and Turner, S., 1993, October. Explaining student grades predicted by a neural network. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)* (Vol. 1, pp. 609-612). IEEE.
4. Niu, M., Li, Y., Wang, C. and Han, K., 2018. RFAmyloid: a web server for predicting amyloid proteins. *International journal of molecular sciences*, *19*(7).
5. Dokur Z., Segmentation of MR and CT Images Using a Hybrid Neural Network Trained by Genetic Algorithm, Neural Processing Letters 16, 2002.
6. Jiang J., BP Neural Network Algorithm Optimized by Genetic Algorithm and Its Simulation, International Journal of Computer Science Issues 10, 2013.
7. Slade, P. and Gedeon, T.D., 1993, June. “Bimodal distribution removal.” In the International Workshop on Artificial Neural Networks. Springer, Berlin, Heidelberg.