# Constructing optimized Neural Networks using Genetic Algorithms and Distinctiveness

Divya Mary

Research School of Computer Science,
Australian National University,
Canberra, ACT – 2601, Australia
u6479594@anu.edu.au

**Abstract.** A back-propagation neural network was trained on the Steel Plate Faults dataset and the resulting model was observed to have a better classification accuracy than other models constructed using the same dataset. In order to acquire this improved accuracy, the neural network had to be constructed with an estimated guess of the hyper-parameters based on trial and error. An evolutionary algorithm was implemented to obtain an optimized set of hyper-parameter values. However, the results of the evolutionary algorithm suggested that the generated model required a large number of hidden neurons. To further optimize the classification model, a technique called distinctiveness, which is a process that will identify hidden neurons that do not contribute any significant functionality to a neural network model was implemented. This paper will discuss how neural networks can be optimized by implementing genetic algorithms and pruning using distinctiveness without compromising on the classification accuracy.

**Keywords:** Genetic Algorithms, Distinctiveness, Back-propagation neural networks, optimal hyper-parameters

## 1 Introduction

Back-propagation neural networks are used extensively for various classification tasks because they can be implemented easily. However, one of the key design issues while implementing a back-propagation neural network is to find optimal values for parameters that would correctly classify a given input without overfitting the model on training data. These parameters primarily include number of hidden neurons, number of epochs and learning rate for training [3,4]. Usually values for these hyper-parameters are decided based on trial and error with the data that the classification task is working on. Although, most optimization algorithms update learning rates during training [5], an optimal learning rate helps to find the right balance between training time and convergence time of a neural network [15]. Number of epochs specify the amount of training required for the model. This must be selected carefully to avoid underfitting and overfitting. Overfitting can be avoided by defining an early stopping condition based on validation results [6]. As a result, the model will be trained until it fails the stopping condition on the validation set. This ensures that finding an optimal value for the number of epochs does not produce any significant impact to the learning of the network. Evolutionary algorithms can be used to find an optimal setting for the remaining hyper-parameters of a neural network on a given dataset. Consequently, a genetic algorithm was applied to obtain the best values for hyper-parameters such as the learning rate and the number of hidden neurons. Moreover, this paper will implement a classification model for the Steel Plate Faults dataset and discuss distinctiveness as a supplementary method to reduce the number of hidden neurons of the trained classification model. This technique is expected to further generalize the trained model and produce the least complicated neural network for classifying the data.

### 1.1 Steel Plate Faults Dataset

The steel plate faults dataset consists of 1941 instances of data that are each classified into one of 7 types of faults that can occur on steel surfaces [2]. These defects are namely, Pastry, Dirtiness, Scratch, Stains, K-Scatch, Bumps and Other. The dataset has 27 features that are used by the network to identify steel plate faults. A classification model that can classify defects in steel plates can be put to significant use in the manufacturing industry. This would reduce the amount of inspection required by humans that takes place in such companies. With a model that can detect defects on steel plate surfaces, a lot of human labor involved with the process can be reduced.

### 1.1.1 Preprocessing Raw Data

Correlation helps to identify the relation between two features in a data space [8]. On computing correlation of all input features of the dataset, it was identified that a pair of input features, namely Y_Minimum and Y_Maximum, were positively correlated with each other with correlation value of 0.999997, which could be rounded to 1. This means that these two input features were almost completely identical to each other (Figure 1) and removing one of these features would not significantly impact the quality of the dataset. Moreover, input features TypeOfSteel_A300 and TypeOfSteel_A400 were found to be perfectly negatively correlated with each other, that is, these features have a correlation value of -1.0 (indicated in Figure 1). This indicates that the presence of one feature directly accounts for the absence of the other. On further inspection, it was clear that the 2 features represented the type of steel and since there were only 2 types of steel, removing one of the features would make no difference to the dataset.

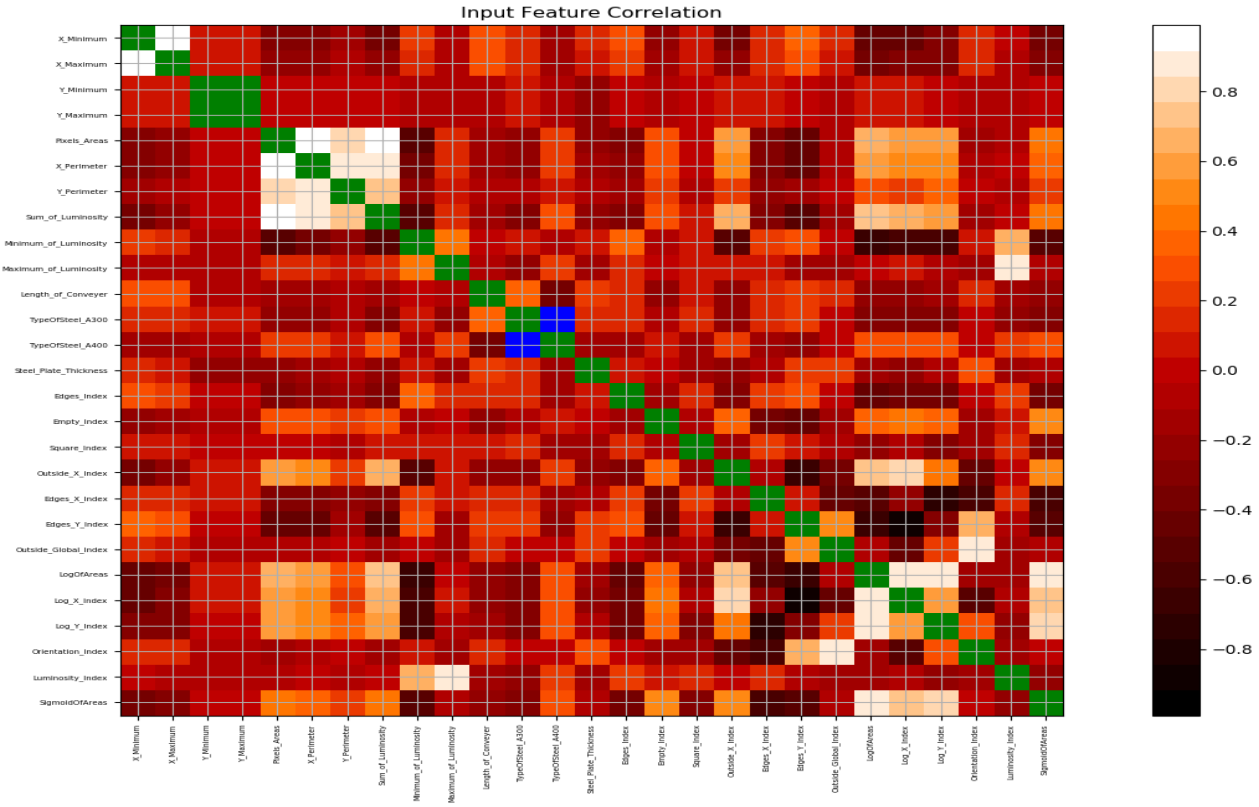


*Figure 1: Correlation of input features of Steel Plates Faults dataset (Green indicates a complete positive correlation and Blue indicates a perfectly negative correlation)*

Furthermore, since all of the data points had varying ranges of values for each input feature, an essential preprocessing step for the dataset was to normalize these values to a scale of 0 to 1. This was implemented by linearly squashing all the input feature values of the data space. In theory, this would ensure that all input features would be given equal importance during the classification process [9].

The dataset represented its output classes with 7 columns where each column corresponds to one of the 7 faults and every instance of the dataset had a 1 in exactly one of the 7 columns. However, this results in a very sparse output matrix since only a fraction of the input instances corresponds to a single output class. To rectify this and to handle multi-class classification, these columns were abstracted into a single column with class labels mapped from 0 to 6.

### 1.2 Back Propagation Neural Network

After preprocessing the data, there were 25 input features and 7 output classes. As a result, a standard two-layer back-propagation neural network was implemented with 25 input neurons and 7 output neurons. The model was trained separately with both normalized and unnormalized data and as expected, feeding normalized input data resulted in a better classification accuracy. Since the data corresponds to a multi-class classification task, better results were observed on applying SoftMax to the output of the neural network before computing the cross-entropy loss of the output neuron. This technique works best for the task because SoftMax computes values (ranging from 0 to 1) for each of the 7 output classes and assigns the maximum value to the predicted class [10]. Cross entropy loss will then detect the loss generated due to incorrect classifications on the training set.

The training data was split into 4 batches and the model was trained on each of these batches iteratively. This is equivalent to a mini-batch training process since the model's error and weights were computed and updated only after each batch was presented to the network [11]. It was observed that the model trained faster in this manner as opposed to the pace of training when the entire dataset was provided as a whole.

The resilient back-propagation (Rprop) optimizer was observed to over-fit the training data and as a result fails in predicting testing data accurately. This may be since the algorithm sets individual learning rates for each weight in the network without momentum and this may cause the network to learn the training data extremely well. The model was also trained using a stochastic gradient descent optimizer and it was observed that this caused the model to reach convergence at an extremely slow pace. If training data is fed into the network in meaningful order, stochastic gradient descent (SGD) is known to take time to converge because the gradients would be biased [12, 13]. However, the data was shuffled before training and hence this could not have slowed the convergence pace. The Adam optimizer was eventually used since this optimizer gave the best results in terms of accuracy. Adam, short for Adaptive Moment Estimation, computes adaptive learning rates for each parameter along with momentum [5]. Thus, it was preferred since it did not noticeably over-fit on the training data and brought the model to convergence at a faster pace than SGD. A sigmoid activation function was applied on output of the hidden layer. Optimum values for hyper-parameters such as learning rate and number of hidden neurons were discovered using genetic algorithms, which will be discussed in Section 2.

#### 1.2.1 Early Stopping and Evaluation

The dataset was shuffled randomly before splitting into train and test sets. The train set was constructed by extracting 80% of the data and the rest was allocated for the test set. An early stopping condition was implemented by additionally splitting the training data into train and validation sets. Thus, the data was split into train, validation and test sets, in the form of 60/20/20. After training for 300 epochs, validation accuracy was computed. For every subsequent epoch, if validation accuracy reduced consistently for 80 consecutive epochs, the training was stopped and the network with the highest validation accuracy was selected. On an average, the model seemed to train well with 1000 epochs set as the maximum. A lower number of epochs (values less than 300) would cause the model to under-fit on the training data. Early stopping further ensured that an optimal value for the number of epochs was not a requirement for effective training of the neural network.

Having implemented a two-layer back propagation network, it was observed that the testing accuracy kept fluctuating on a wide range for various splits of the dataset. This occurred because the train-test splits were random and the model was trained on only one split of train and test data. Moreover, since the number of instances in the dataset were relatively small, a "bad" split of the dataset could force the neural network to learn poorly. A bad split, in the context of this dataset, means that the training set could have noisy data, the presence of outliers or could also be an extremely optimistic representation of the test set. It also could mean that the training data may not have a good representation of a particular class since most of the class's instances have been partitioned into the test set. To obtain a more reliable estimate of the general accuracy of the final model, 10-fold stratified cross validation was implemented. 10-fold stratified cross validation splits the entire data into 10 subsets with each subset having an almost equal representation of all output classes [14]. These subsets are then used to construct 10 separate models in which each model would be trained with one of the ten subsets as testing data and the remaining as training data. As a result, each of the ten subsets would be used for testing exactly one of the models. The average of the testing accuracies obtained from the ten different models was reported as the final prediction accuracy. This result provided a better generalization of the model's accuracy on new and unseen data. Moreover, the entire dataset was used for both training and testing. However, as this was a computationally

expensive process, the number of folds were reduced to 3 while constructing the models to evaluate fitness in the genetic algorithm, which is discussed in the next section.

## 2 Genetic Algorithm for hyper-parameter optimization

Genetic algorithms are used extensively to solve hard optimization problems since they are guaranteed to converge eventually to the global minima [16]. As discussed earlier, finding a perfect optimal value for hyper-parameters such as learning rate and number of hidden neurons requires a brute-force trial and error approach. This was simplified by implementing a genetic algorithm to compute these values for the model. The fitness values for the proposed genetic algorithm was the 3-fold validation accuracy obtained on each individual model. The population size of the genetic algorithm was set to 10 and the size of each DNA was two. This was because the objective of the genetic algorithm was to find optimal values for learning rate and number of hidden neurons. Also since learning rate is a floating point value and the number of hidden neurons is an integer, the DNA was a mixed type representation consisting of a float and an integer value. The initial population was randomly selected for values of learning rate ranging from 0.01 to 0.002, and number of hidden neurons ranging from 50 to 1000. Prior experimentations on the model trained with the dataset, indicated that values out of the ranges had a degrading effect to the training performance of the neural network.

Since genetic algorithms are not proven to have an upper-bound on runtime, the number of generations was limited to 15. Moreover, the algorithm was implemented to terminate if the best value does not change for two consecutive generations. This form of an early stopping technique will guarantee that the model will stop training after it has discovered a good solution. Proportional selection (roulette wheel selection technique), which selects individuals based on a probability distribution proportional to fitness was implemented. This selection technique was used with replacement to ensure that better individuals had a higher chance of being selected from the current population. Furthermore, an elitist selection approach was adopted to force the genetic algorithm to converge to a good solution faster. This means that the best candidate of each generation was exempted from the selection process and used in the new population for generating new offspring. This also confirms that the best candidate will be bypassed to the next generation's population. A uniform crossover technique was implemented with a crossover rate of 0.8. A crossover rate indicates a probability of the proportion of couples that would be picked for mating. This rate was set to 0.8 to ensure that a good proportion of individuals in the next generation is obtained by combining features of individuals of the current generation. In genetic algorithms, mutation introduces genetic diversity to the population. Random mutation, which changes genes randomly in chromosome, was implemented as the mutation technique for the genetic algorithm discussed in this paper. Mutation rate indicates the proportion of random genetic information that will introduced to the new population. A larger mutation rate will allow the population to explore the search space well and this is preferred during the start of the genetic algorithm. However, as the algorithm proceeds and begins to find good solutions, a lower mutation rate would ensure that these solutions are preserved and exploited further. As a result, a mutation rate that decreases with respect to the number of generations that have been completed was implemented.

## 3 Distinctiveness

### 3.1 Inspiration to implement pruning

The results from the genetic algorithm indicated large values of hidden neurons for good results on the training. However, this value can be further reduced to obtain a better generalization of the trained model. A good generalization will indicate that the network uses the ideal number of interconnections, processing units and memory required to represent weights. Lowering the number of hidden neurons will also ensure that the network does not over fit on the training data. As a result, the next goal was to find the minimal number of hidden neurons required to build a model that is well suited for the classification task. In order to obtain that, hidden neurons that do not provide any significant help in classification must be pruned out of the network. In other words, it was necessary to implement a method that identifies hidden neurons that are replicas of other hidden neurons and removes them from the network. The method must also identify neurons that dissimilar from other hidden neurons and eliminate such neurons as well. There are many pruning techniques to reduce the number of hidden neurons, namely badness, sensitivity, distinctiveness and so on [7]. However, distinctiveness aligns more closely with the requirements of this model and hence, the main focus of this paper will be to employ distinctiveness with an aim to find the optimal number of hidden neurons for this classification task.

### 3.2 Distinctiveness

Distinctiveness is the process of identifying indistinct neurons in a neural network model [7]. Distinctiveness can be determined by computing angles between every pair of hidden neuron vectors in the data space [7]. If the angle of a pair of hidden neurons is below a certain threshold, both the hidden neurons are concluded to be similar to each other in terms of functionality. Moreover, if the angle is more than another given threshold, the hidden neurons are contradictory to each other, which means that one neuron complements the other neuron's functionality. In both cases, removing at least one of the hidden neurons from the pair of hidden neurons will ensure that the network only has distinct neurons. As a result, this process helps to reduce the computational complexity of the network. By implementing this approach, the network will also classify results faster because it will have lesser neurons to process in each forward pass [7]. This will, in effect, ensure that each hidden neuron in the network will be performing a unique function and thus maximizes its utility.

### 3.3 Implementation

Since the neural network model was implemented with a large number of hidden neurons, it was necessary to identify weak and indistinct hidden neurons and remove them. In theory, this would ensure that the overall accuracy remains the same while the complexity of the model reduces. In order to implement distinctiveness, the output activations of the hidden layer were scaled from -0.5 to 0.5. This ensures that the computed angles lie between 0 degrees to 180 degrees [7]. Angles were computed for all pairs of hidden neurons in the network. All angles less than 15 degrees and more that 165 degrees were identified as similar or dissimilar hidden neuron pairs, respectively [7]. Neurons with lower index values in pairs that had angles less that 15 degrees were added to a list to be pruned from the network. Alternatively, both neurons in pairs having an angle greater than 165 degrees, were also added to the same list. It was observed that removing pairs of neurons with angles in between 15 and 165 degrees would results in a degraded performance of the neural network model. This indicates that such neurons provide significant functionality to the neural network for the classification task discussed. Further experiments with pruning showed that removing the indistinct neurons after training resulted in some loss of functionality of the trained model. This could suggest that while neurons that are pruned out of the model were not distinct to the network, they still provided some assistance to the classification task. As a result, some training must be done on the neural network after pruning to allow other neurons in the network to learn functions that might have been eliminated as a consequence of pruning. This also meant that pruning had to be implemented towards the end of training to ensure that irrelevant neurons were identified accurately. If pruning were implemented on a partially learned network, the accuracy of the results obtained from distinctiveness would suffer.

## 4 Results and Discussion

The evolutionary algorithm was successful in identifying optimal values for learning rates and numbers of hidden neurons. It was observed that the algorithm would select learning rates equivalent to 0.5 (+/-0.1). Similarly, the number of hidden neurons selected by the algorithm would always be more than 300, which is a significantly large value. Since this approach was computationally expensive, the models of each individual chromosome was trained with 3-fold cross validation. However, after obtaining results from the genetic algorithm, the final neural network model was trained with a 10-fold cross validation, along with pruning implemented towards the end of the training process. The genetic algorithm was also observed to generally converge to a good solution in a generation between the $6^{th}$ and $10^{th}$ generations.

The optimized neural network model had an aggregated testing accuracy of approximately 75%. As discussed earlier, pruning the model based on distinctiveness after the training was completed resulted in a lower testing accuracy of approximately 74%. This may have occurred because the hidden neurons that were removed might have been providing some useful information to the model. The network was later pruned once at the $300^{th}$ epoch, to allow the model to learn characteristic patterns that might have been eliminated as a result of pruning. The $300^{th}$ epoch was selected for pruning since the network was observed to have completed at least 60% of learning by this epoch. Figure 2 compares the historical loss during the training process for the two cases that pruning was implemented. First, after training and second, at the $300^{th}$ epoch. As expected, it can be seen that there was a slight increase in loss after pruning the network in both cases.

However, if the model is pruned at a certain point during the training process, it then recovers from this loss quickly with further training. Pruning the model in this manner resulted in a generalized testing accuracy of 76%, which is similar to the accuracy of the model with 600 neurons.

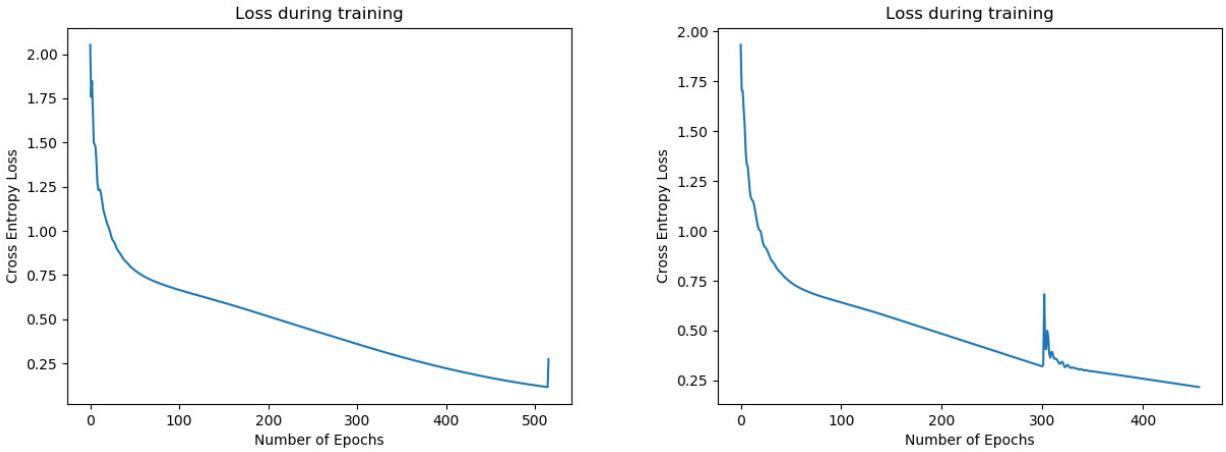


*Figure 2: Training loss plotted when (a) pruning was implemented at the end of training and; (b) pruning was implemented after the 300th epoch*

Since the Steel Plate Faults dataset has 1941 instances and the model was constructed with a significantly large number of hidden neurons, computing angles between every hidden unit output activation vector slowed down the training process since each hidden unit vector had 1941 values. As a result, this could be an inefficient approach to identify indistinct hidden neurons for larger datasets.

For each fold in the 10-fold validation, a minimum of 10 neurons were identified as insignificant to the model for most values of hidden neurons selected by the genetic algorithm. This number of indistinct hidden neurons were almost the same for both cases; that is, when pruning was implemented at the 300$^{th}$ epoch and at the end of the training. Thus, neurons that would add to the complexity of the model could be safely removed without hampering the efficiency of the model.

Researchers, Massimo Buscema, Stefano Terzi and William Tastle had implemented a back-propagation neural network using the steel plate faults dataset for their research and concluded that the neural network model had a weighted mean of 70.53% [1]. This value is significantly lower than the generalized testing accuracy of approximately 75% of the proposed model. This difference in accuracy may be due to the improvements in technology since 2010 and the use of sophisticated approaches for the implementation of my model. One such approach would be the utilization of the Adam optimizer, an optimization algorithm that was introduced years after their paper was published. As their paper has not covered details of their back-propagation neural network implementation, it is difficult to identify any other reason for the lower accuracy that they have reported. The researchers also compared the results of classification with a model they proposed and some other classification models [1]. Some of those classification model results for the dataset are reported in Table 1 along with the results for model discussed here (Back-Propagation – Distinctiveness).

| **Model** | **Weighted Aggregate (%)** |
|---|---|
| Back-Propagation – Distinctiveness | 75.27 |
| Back-Propagation (Buscema et al.) | 70.53 |
| SVM (Buscema et al.) | 74.04 |
| KNN (Buscema et al.) | 70.94 |
| Decision Tree (Buscema et al.) | 73.11 |
| Meta-Consensus (Buscema et al.) | 76.47 |
| Bayesian Linear Classifier (Buscema et al.) | 64.97 |

*Table 1: Classification results of various classifiers on the Steel Plate Faults dataset [1]*

## 5 Conclusion and Future Work

Overall, it was observed that a generalized neural network model could be constructed with the help of genetic algorithms for hyper-parameter optimization. Furthermore, distinctiveness could successfully identify hidden neurons in the network that do not provide any significant quality to the trained model. This makes it possible to begin training the network with a relatively high number of hidden neurons since distinctiveness can be employed to successfully prune weaker neurons out of the model. Pruning the network based on distinctiveness after the entire training was completed resulted in a slightly lower classification accuracy. It was observed that pruning towards the end of training gave the model an opportunity to retrain on the distinct hidden neurons for an enhanced testing accuracy.

Applying a genetic algorithm for parameter optimization and calculating distinctiveness for all pairs of hidden unit output activation vectors of a model are both computationally expensive processes and will decelerate the training process. While a combination of these methods was a good start in terms of reducing network complexity and obtaining an optimized network, there are many more alternative approaches that can possibly produce the same (or even better) results without drastically affecting the training time of the neural network. Techniques such as memetic algorithms that are known to find good solutions faster could be adopted to optimize the neural network model. Furthermore, other approaches to prune the network such as badness or sensitivity must be investigated and implemented to analyze which pruning method works best for the classification task discussed in this paper.

In future, the classification problem on the Steel plate faults dataset could also be solved by implementing various other machine learning classification algorithms, such as Support Vector Machine, for instance. Support vector machines are known to require fewer hyper-parameters, when compared to neural networks and are also guaranteed to find the global optimum for the classification problem at hand [17]. As a result, research can be extended to investigate the performance of SVMs on the dataset.

# References

[1] M. Buscema and W. Tastle, "A new meta-classifier", 2010 Annual Meeting of the North American Fuzzy Information Processing Society, 2010.

[2] "UCI Machine Learning Repository: Steel Plates Faults Data Set", Archive.ics.uci.edu, 2010. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/steel+plates+faults. [Accessed: 29- Apr- 2018].

[3] Y. Hirose, K. Yamashita and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units", Neural Networks, vol. 4, no. 1, pp. 61-66, 1991.

[4] S. Fahlman, An Empirical Study of Learning Speed in Back-Propagation Networks. CMU, 1988.

[5] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", 2017.

[6] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria", Neural Networks, vol. 11, no. 4, pp. 761-767, 1998.

[7] T. Gedeon and D. Harris, "Network Reduction Techniques."

[8] M. Hall, "Correlation-based feature selection for machine learning", 1999. [Online]. Available: https://www.cs.waikato.ac.nz/~mhall/thesis.pdf. [Accessed: 29- Apr- 2018].

[9] "Data Normalization and Standardization for Neural Networks Output Classification", AH's Blog. [Online]. Available: https://ahmedhanibrahim.wordpress.com/2014/10/10/data-normalization-and-standardization-for-neural-networks-output-classification/. [Accessed: 29- Apr- 2018].

[10] P. Golik, P. Doetsch and H. Ney, "Cross-Entropy vs. Squared Error Training: a Theoretical and Experimental Comparison."

[11] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size", Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/. [Accessed: 29- Apr- 2018].

[12] "Inefficiency of Stochastic Gradient Descent with Larger Mini-Batches", Openreview.net, 2018. [Online]. Available: https://openreview.net/pdf?id=Bk_zTU5eg. [Accessed: 29- Apr- 2018].

[13] "Unsupervised Feature Learning and Deep Learning Tutorial", Ufldl.stanford.edu. [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/. [Accessed: 29- Apr- 2018].

[14] P. Refaeilzadeh, L. Tang and H. Liu, "Cross-Validation", Encyclopedia of Database Systems, pp. 532-538, 2009.

[15] https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0

[16] A. van Soest and L. Casius, "The Merits of a Parallel Genetic Algorithm in Solving Hard Optimization Problems", Journal of Biomechanical Engineering, vol. 125, no. 1, p. 141, 2003.

[17] Z. Huang, H. Chen, C. Hsu, W. Chen and S. Wu, "Credit rating analysis with support vector machines and neural networks: a market comparative study", Decision Support Systems, vol. 37, no. 4, pp. 543-558, 2004.