Network Reduction on an Optimized Neural Network Based on Distinctiveness

Yadu Krishna Choyi
Australian National University
Acton, ACT, 2601

Abstract. The paper explores and compares the effect of network reduction technique based on distinctiveness, on models optimized using genetic algorithm. In particular, the paper focuses on studying the variation in accuracy of a model that has been trained on a dataset using genetic algorithm before and after applying network reduction technique. The results show that an optimized network can be reduced by a significant factor with a small trade off in performance with respect to its accuracy.

Keywords: Distinctiveness, mutation, cross-over, pattern vector, similar pairs, complementary pairs.

1 Introduction

The wide scale application of neural networks in many fields of science has proved itself to be almost invaluable. Even though the popularity of neural networks and the research in the field has been progressing at a steady rate, one of the key drawbacks that neural networks suffer from is that the number of parameters used for training is huge, which means that the user has to train the model several times to find optimum hyper-parameters. Often the best model that can solve a real world problem is complex in terms of its network size even if the model has low accuracy. Neural network models are often termed as black box models as it is almost impossible to interpret how a network actually performs a given task. Neural networks are evaluated on the basis of evaluation metrics such as accuracy, r2 score etc. These metrics provide the user with an efficient way of assessing a model on its performance in terms of its output. A good network not only has good performance in terms of evaluation metrics, but also has optimal number of neurons(Network size should be related to the complexity of the task in hand). The number of neurons in a network effectively defines its performance in terms of execution times. Depending on the use case a network that can perform a task in 0.01 second with an accuracy of 90% may be more preferable than a model that can perform the same task in 1 second with an accuracy of 95%. These models are hard to optimize due to the presence of a number of hyper-parameters. These hyper-parameters define how a network is trained, and therefore it is crucial to set these parameters optimally to achieve a model with good performance. This is achieved by implementing a genetic algorithm that mimics human population. Instead of trying to optimize the hyper-parameters on a single model, the genetic algorithm creates a number of model each with different traits(hyper-parameters) such that only the best models present in the current generation are able to 'reproduce' and pass on their 'child' to the next generation. Iterating through each generation, we expect the model to get better and better until the performance eventually plateaus. This results in a model that has been perfected through evolution, mutation and cross-over. The paper focuses on the application of network reduction technique on such models.

A neural network has a fixed number of hidden neurons in a specific hidden layer. These hidden neurons act as neurons in the human brain and a specific input to these neurons produce a specific output. By analyzing and comparing outputs of these neurons, we can determine the existence of neuron pairs that are similar or complementary to each other. If the output of two neurons in a specific hidden layer are similar, this implies that these two neurons are performing the same task. Such neuron pairs can be merged into one to reduce the effective size of the network. On the other hand if the output of two neurons are opposite/complementary to each other, then such neuron pairs can be effectively removed. This idea is termed as distinctiveness by Gedeon and TD, Harris[1]. The output from each hidden layer neuron is termed as pattern vectors. For one specific input to the network, each hidden neuron produces a single output. The outputs of a hidden layer neuron for all the data in the training set results in a pattern vector, where each element in the vector is the output of a hidden layer neuron for a specific input data in the training dataset. Therefore for a given neural network model, we have n pattern vector, where n is the number of hidden layer neurons. Each pattern vector is compared with other patter vectors in the same hidden layer. The angle between these pattern vectors are used as metrics to grade a being pair as similar or complementary. With this knowledge, all redundant neurons in a specific hidden layer can be removed in order to produce an optimized model.

The above mentioned technique is demonstrated on the faces emotion dataset based on anger detection[2]. The dataset is generated form the pupilary response of 22 participants. Each participant was shown 20 different clips displayed in 20 different sequences. These clips consisted of people expressing genuine and fake anger. The subjects pupilary response was then captured and sampled at every 1/60th of a second. This results in a time series data for 20 participants per video clip. The response from participants were used to train a neural network model. The task of the model is to identify whether the people in the clip expressed genuine or fake anger.

The version 1 of this paper describes the effect of network reduction on a network model that has not been optimized in terms of its hyper-parameters. For the previous paper, we trained a simple feed forward neural network for binary classification by using the summarized version of the anger dataset[2]. We did not focus on improving the accuracy of the trained model by tuning its hyper-parameters, instead we focused on the change in accuracy of the model after applying the network reduction technique. The aim of the previous paper was to compare and contrast two models that performs the same basic task, but one of the model was a reduced model that did not require training, as it was essentially derived from the original model. The reduced model was then evaluated for performance by comparing it with the original model. The evaluation was based on the test set accuracy of the trained model and the reduced model. The results showed that there was a decrease in accuracy of the reduced model by an average of 2%, and on average the network reduction technique[1] pruned 45% of hidden layer neuron. Using these results as a baseline, here we try to evaluate a reduced network model that has been optimized using genetic algorithm by training the model on the complete version of the dataset.

2 Method

2.1 Data Preprocessing

For a particular video clip shown to a participants, we have two datasets that contains the left eye pupilary response and the right eye pupilary response of all 22 participants. There are 40 such dataset(20 for left eye and 20 for right eye) corresponding to each of the 20 video clips. The response of the left eye and right eye for each participants was averaged, which resulted in 20 dataset. All 20 dataset was then stacked in order to generate a single dataset with 2644 records. The dataset contains a large number of zero values which implies that some of the participants blinked when their response was captured. Assuming that the dataset does not contain too many outliers the zero values was replaced with the mean value. The output label was encoded as 1 for responses from participants when shown genuine anger and 0 for responses from participants when shown fake anger. Using the min_max scaler[5], the data was normalized. The dataset was split into train and test data in the ratio 90/10. The same ratio of train/test split is used throughout the paper.

2.2 Genetic/evolutionary algorithm

Genetic algorithm[6] is based on the idea of natural selection where the aim is to optimize a constrained or unconstrained problem. The genetic algorithm is implemented by defining an initial population of size n. The initial population consists of n number of network models, where each model has randomly assigned hyper-parameter. The fitness of each individual(model) is then calculated for the current generation. The fitness of an individual is defined by the test set accuracy of a specific model. The higher the test set accuracy of a model, the greater is the fitness. Two parents are then selected form the population to perform cross-over and mutation. The cross-over operator takes properties(hyper-parameters) from two parents and then combines them to generate an intermediate on which the mutation operator is performed. The mutation operator performs mutation by randomly flipping bits in the binary representation of the intermediate. The resulting mutated representation is converted back to decimal and is then used as hyper-parameters to train a child network model, which is then passed on to the next generation. The process continues for m number of generation, where members in each generation is expected to evolve so as to find the optimal solution.

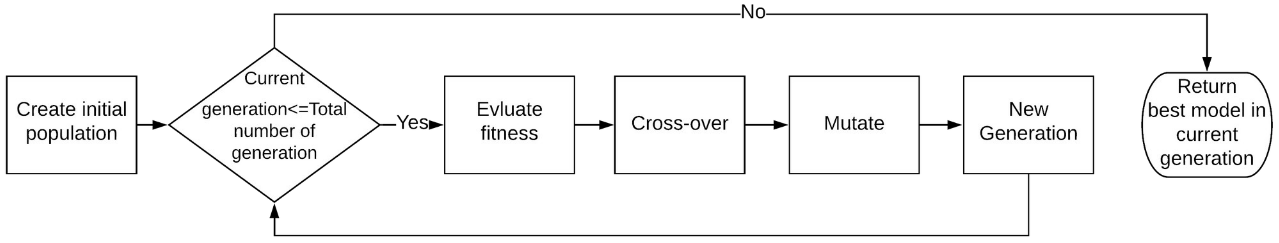
Create initial population
Current generation<=Total number of generation
Yes
No
Evluate fitness
Cross-over
Mutate
New Generation
Return best model in current generation

Figure 1: Genetic algorithm flowchart

For our case, the task of the genetic algorithm was to find the optimal value for the following hyper-parameters; 1) Learning rate: This parameter determines the extent to which the model weights are updated in response to the estimated error, 2) Number of hidden layer neurons, 3) Number of epochs for learning: This parameter determines the number of times the training data is used to update the weights of the network. These parameters are learned over time through the use of genetic algorithm. For the purpose of this paper, the hyper-parameters of the genetic algorithm such as the population size, mutation rate and the number of generation was set to 100, 0.1 and 100 respectively.

The genetic algorithm was manually implemented in python. The main components of the implementation includes the following:

- A method to initialize the initial generation with random hyper-parameter values. The hyper-parameters such as the learning rate, number of hidden layer neurons and epochs for each individual in the first generation was selected randomly. The random selection of these hyper-parameters was restricted within the limit mentioned below.

  Learning rate = (0.01 – 0.9)

  Number of hidden layer neurons = (10 – 300)

  Number of epochs = (10 – 500)

- A method to determine the fitness of all individuals in the current generation. The fitness of an individual is equal to its test set accuracy. The chances of an individual being selected as parents in a specific generation is directly proportional to its fitness score, therefore individuals that perform best has a higher chance of being selected as a parent. The aim of the evolution process is to maximize the fitness score.

- A method to implement the cross-over operator. This allows us to combine the hyper-parameter values of two parents to generate a child. The cross-over operator selects two individuals from the current population as parents. The hyper-parameters of the parents are then represented in their binary form. The binary representation facilitates cross-over. The cross-over is implemented manually by mixing up the binary representation of parameter values of both the parents. The resulting binary number is then converted back to its decimal representation and is then used as hyper-parameters to train the child model.

- A method to implement mutator operator. The mutator operator is used to maintain genetic diversity throughout the evolution process. This operator changes the properties of the hyper-parameters of the child class by a small amount based on a random event. The hyper parameters resulting from the cross-over operator is modified slightly by adding a small value(a random value not greater than 10% of the max limit mentioned above) to it This small change ensures that the child class is not a direct copy of its parent class.

2.3 Network Architecture

For the purpose of demonstrating the effect of applying the network reduction technique[1], a simple feed forward network with 20 input neurons corresponding to the input features and 2 output neuron was created. The number of hidden layer neuron is selected by the genetic algorithm. The number off hidden layers was fixed to a bare minimum of 1. Other variables, such as criterion was set to cross entropy loss[3] and the optimizer was set to Adam[4] with sigmoid[7] activation function for the hidden layer neurons. This architecture can be considered as the foundation of individuals in a population with respect to the genetic algorithm. Each individual in the population has the same architecture except for varying learning rate, number of hidden layer neurons and number of epochs.

2.3 Network reduction computation

Reducing the above mentioned basic network requires us to find the angels between output activation pattern vectors. Each vector consists of outputs form one single neuron. For the case in hand, we have a variable number of hidden neurons for individuals in each generation, therefore as an example, let us consider the case of a model with 5 hidden layer neurons. Such a model will have 5 pattern vectors, each having n number of elements, where n is the size of the training set. Given below in table 1 is the pattern vector representation for the example model.

Training data index	Pattern vector 1	Pattern vector 2	Pattern vector 3	Pattern vector 4	Pattern vector 5
	1	2	3	4	5
1	0.14	1.03	0.15	-0.67	-0.41
2	0.14	-0.93	0.15	-0.67	-0.41
3	0.14	0.37	0.15	-0.67	-0.41
.	.	.	.	.	.
.	.	.	.	.	.
n-2	0.14	0.71	0.15	-0.67	-0.41
n-1	0.14	1.64	0.15	-0.17	-0.41
n	0.14	-0.4	0.15	-0.67	-0.41

Table 1: Pattern vector representation

With these pattern vector representation, we calculate the angles between all pairs of hidden neurons, but before the angles can be calculated, we normalize the values between the range -0.5 to 0.5 [1], this is done so that when the angle between pattern vectors are calculated, the resulting angles are in the range of 0-180 degrees and not 0-90 degrees. The angles calculated provides the criteria for us to distinguish between similar pairs and complementary pairs. According to Gedeon, TD, Harris[1], if the angle between pairs is less than 15 degrees, we can consider them as being similar, and if the angle is greater than 165 degrees, they are considered to be complementary(the threshold angles are set with a small degree of freedom because in the real world, two neurons are rarely exactly similar or complementary). The threshold parameter in our case has been set to 10 degrees for classifying similar pairs and 180 degrees for classifying complementary pairs. Continuing with the previous example of a simple neural network with 5 hidden neurons, the vector angles calculated for each pairs from the pattern vectors are given below in the table 2. In this case we have a similar pair (0,2) and no complementary pairs. The two neurons in the similar pairs are replaced by a single neuron (either 0 or 2 in this case) after adding the weight from the neuron to be deleted to the neuron that will be replacing the pair itself(we can add weights of both neuron 0 and neuron 2 and replace either neuron 0 or neuron 2 with the updated weights[1]). With respect to complementary pairs, both the neuron can be deleted(Note: Neurons that are classified to be deleted are not removed form the network, instead the weights of the corresponding neuron are set to zero to mimic neuron deletion).

Hidden Neuron Pairs	Angle	Category
(0,1)	131.51	-
(0,2)	1.87	Similar
(0,3)	22.83	-
(0,4)	25.01	-
(1,2)	131.51	-
(1,3)	130.59	-
(1,4)	146.05	-
(2,3)	22.83	-
(2,4)	25.01	-
(3,4)	30.56	-

Table 2: Classifying neuron pairs as similar or complementary

3 Results

The genetic algorithm was run with its parameters such as population size, mutation rate and number of generations set to 100, 0.1 and 100 respectively. After every ten generations, the best performing individual in the current generation is evaluated for test set accuracy. The same model is then reduced using the network reduction technique[1]. The new reduced model does not require training as the trained weights are just modified to mimic neuron deletion. The reduced model is then run on the testing set and is evaluated. Looking at figure 2, we can see that the accuracy of the best performing model for each generation is increasing over time. The genetic algorithm successfully modifies the hyper-parameter such that the resulting best model in the final generation has much higher accuracy when compared to the best model in the first generation. The accuracy of the reduced model is lower than the model before applying the network reduction technique by an average of 5%. Upon testing for improvement in accuracy by increasing the number of generations, it was seen that the accuracy of the best performing model did not go beyond 83.80%, and the accuracy of the reduced model peaked at 81.29%.

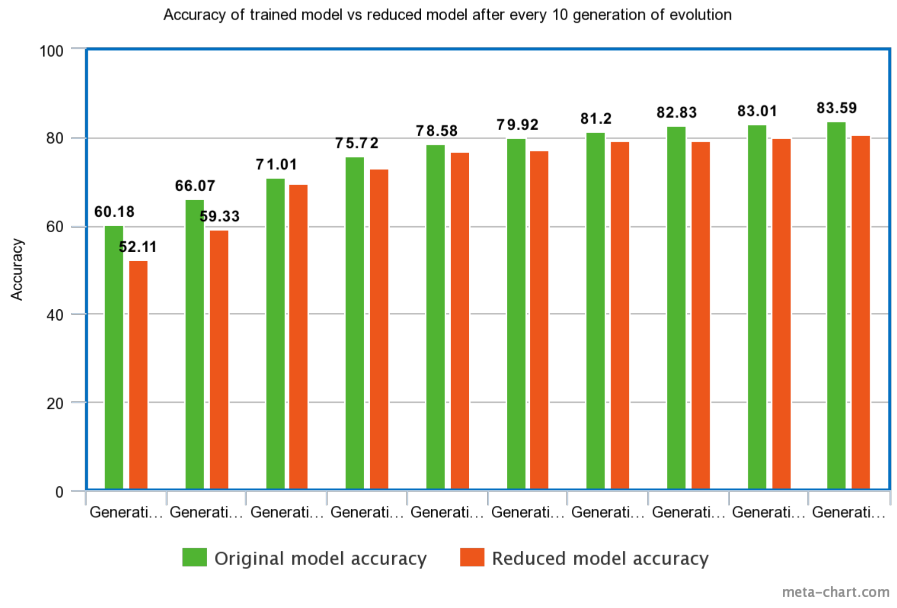

Figure 2: Comparison of accuracy of best performing model after 10 generation before and after network reduction

Figure 3 visualizes the number of hidden layer neuron of the best performing model for every 10 generations, and the number of neurons pruned using the network reduction technique. Form figure 3, we can see that the network reduction technique usually prunes 45% of the hidden layer neurons, which implies that for each model, there exists a large portion of redundant neurons that do not contribute much its performance. More over, we can also see that the models in the first 30 generations have a large number of hidden layer neurons(Figure 3), and low accuracy(Figure 2), and it is the opposite for the last 30 generations. Specifically, for generation 100, in which the best accuracy was achieved, the size of the hidden layer in the network was reduced from 156 to 85 (156-71). Even though more than 50% of the hidden layer neurons were removed, the model was able to perform with an accuracy of 81.23%.

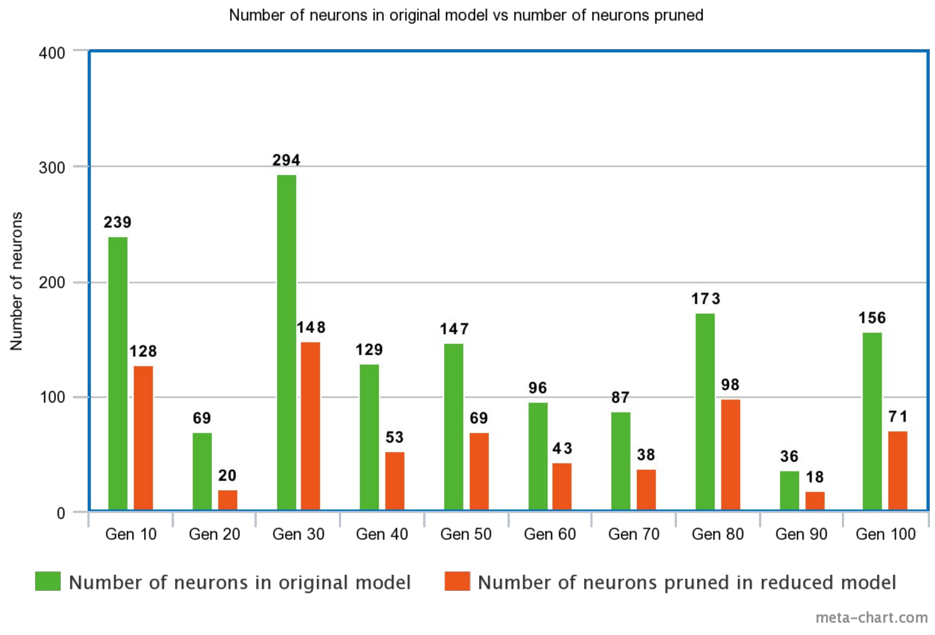

Figure 3: Number of hidden layer neurons of best model vs the number of pruned neurons for the same model after every 10 generations

Figure 4 and 5 shows the evolution of other hyper-parameters such as the learning rate and number of epochs respectively, which is used for training the model that performs best after every 10 generations. Both figure 4 and 5 follows a similar trend. We can see that both the values dip and then increase at the end of the evolution process. It is observed that the values of the mentioned hyper-parameters vary by a significant amount every 10 generations, but eventually reaches optimal value at the end of the evolution process. The average time for running 100 generations was 6.8 hours. The code implementation was run on Google colab notebook with 12.7 gb ram, and GPU acceleration for training individuals in each generations. In comparison, an exhaustive search method would not have been able to find any reliable combination of hyper-parameters.

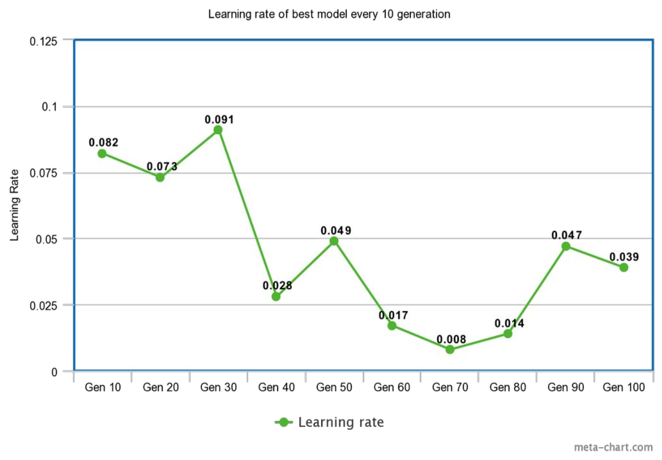

Figure 4: Evolution of the learning rate parameter for the best performing model after every 10 generations

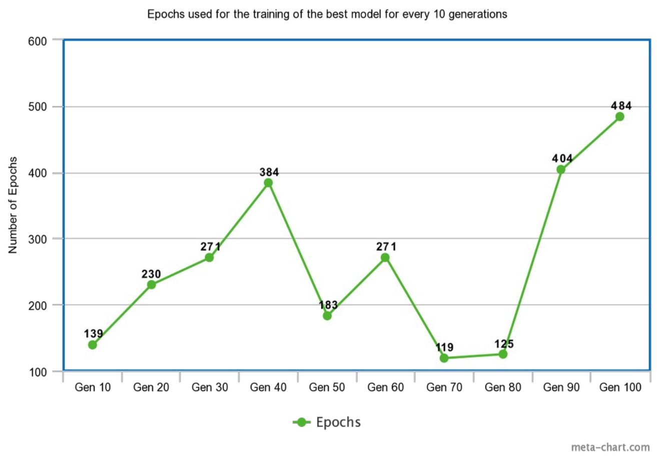

Figure 5: Evolution of the number of epochs parameter for the best performing model after every 10 generations

4. Conclusion and Future work

In this paper we discuss the effect of network reduction technique[1] using a simple neural network model optimized using genetic algorithm with one output on the anger dataset[2]. The accuracy of the original and the reduced model vary within an average of 5%. The reduced model seems to produce acceptable results, even though in most cases almost 45% of the hidden neurons are removed. This implies that in most neural network models, there exists a large number of units that are redundant and removing these units can improve computational performance by trading off a small percentage of evaluation metrics. One thing to keep in mind is that the reduced model performance is dependent on the original model. If the original model has high accuracy on the test set, the reduced model can be assumed to have less, but similar test set accuracy/performance.

In summary, the results generated from applying the network reduction technique[1] on the anger dataset[2] shows us that there is a huge margin for us to improve upon when it comes to network pruning. The results are compelling and can be used in real world application, but more research needs to be done. Research can be done on evaluating this technique on models with multiple hidden layers and multiple outputs. Models with multiple hidden layers may behave differently and using this technique may or may not yield ideal results. The threshold parameter for selecting similar and complementary pairs are based on the network reduction paper[1]. This threshold can be varied and its effects on the reduced model can be studied.

5. References

[1] Gedeon, TD, Harris, D, “Network Reduction Techniques,” Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, vol. 1, pp. 119-126, San Diego, 1991a. Proc. 9th Ann. Cog. Sci. Soc. Conf., Seattle, pp. X-y, 1987

[2] Chen, L., Gedeon, T., Hossain, M. Z., & Caldwell, S. (2017, November). Are you really angry?: detecting emotion veracity as a proposed tool for interaction. In Proceedings of the 29th Australian Conference on Computer-Human

Interaction(pp.412-416).ACM.

[3] Zhang, Z. and Sabuncu, M. Generalized cross entropy loss for training deep neural networks with noisy labels. NeurIPS, 2018.

[4] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[5] T. Jayalakshmi, A. Santhakumaran, "Statistical Normalization and Back Propagation for Classification," *International Journal of Computer Theory and Engineering* vol. 3, no. 1, pp. 89-93, 2011.

[6] Goldberg, David E. "Genetic and evolutionary algorithms come of age." *Communications of the ACM*, vol. 37, no. 3, Mar. 1994, p. 113+. Accessed 30 May 2020

[7] Karlik, Bekir, and A. Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks." *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011): 111-122.