Diabetic Retinopathy Automatic Screening- a Neural Network Approach with Pruning and Genetic Algorithm Hyper-Parameter Search

Areij Javed

Research School of Computer Science, Australian National University
Email: U6270870@anu.edu.au

Abstract. In this paper, the UCI diabetic retinopathy dataset has been classified using a neural network. A feed-forward single-layered neural network model is used. Furthermore, a neuron pruning technique known commonly as badness has also been employed in order for the neural network to converge faster. For both of these network models, optimal hyper-parameters are discovered using a Genetic Algorithm. The pruned version of the neural network converges faster initially but slows down towards the last few epochs. The neural network has been evaluated using 10-fold cross validation, and its performance is then compared with that of a previously published work on the same dataset. The results yielded by the two versions of the neural networks (although better than some of the classifiers used in the published work), are worse than numerous other classifiers, including a neural network implemented in the published paper as well. The disparity in performance is most likely a result of inadequate feature preprocessing.

Keywords: Diabetic Retinopathy, neural network, badness, pruning, Genetic Algorithm

1 Introduction

Diabetic Retinopathy(DR)- is an eye disease and a consequence of diabetes, that causes damage to the blood vessels in the retina, thereby gradually resulting in loss of vision. If the disease is diagnosed in its early stages, immediate treatment can hinder any further degradation of the retina. The diagnosis is carried out by investigating retinal images, specifically detecting the presence of microaneurysms, hemorrhages, neovascularization, and exudates [1]. In order to diagnose effectively, a highly specialized ophthalmologist is required to screen the images very carefully, meaning that the process is highly susceptible to human error. As a result, a lot of focus has been put on developing automatic screening systems for efficiently and effectively detecting DR [2]. Numerous automatic methods have been used in order to detect the presence of DR, such as SVM, adaptive boosting, neural nets, Decision trees etc. [3].

In this paper, we will employ neural networks in order to automatically diagnose Diabetic Retinopathy. Specifically, we will employ a feed-forward neural network with a single hidden layer in order to solve this classification problem. Neural networks have many parameters that need to be determined, and since these parameters are essential to yielding good performance, we will use a Genetic Algorithm in order to find the best parameters for the task.

Since the neural network can be slow to converge, we investigate a network pruning technique called badness [7], that will allow us to obtain a neural network that is fast to train and test, without degrading its overall performance.

In this paper, we also compare the results of our neural network with those published in A comparative analysis of classification algorithms in diabetic retinopathy screening [3].

2 Method

2.1 Data Set and Input pre-processing

The UCI Diabetic Retinopathy Debrecen dataset is used in this paper [4]. The dataset consists of a total of 19 features extracted from Messidor images which indicate if the image shows signs of DR or not. Binary attribute f_0 corresponds to the image quality, binary feature f_1 is a pre-screening and indicates if there is severe retinal abnormality or not, features f_2-f_7 state the number of microaneurysms found at different confidence levels (α=0.5-1, respectively), attributes f_8-f_{15} represent the number of exudates for varying confidence levels, f_{16} consists of the Euclidean distance between the macula

and the center of the optic disc, f_{17} represents the diameter of the optic disc, and f_{18} is the binary result of AM/FM based classification. The dataset also includes a class label for each image- image shows signs of DR (denoted with a 1) or not (0). The dataset has a total of 1151 instance (one instance per Messidor image) and is meant for solving a classification problem.

Based on the dataset, it was observed that Attribute f_0 had a negligible variation in its values- it was 1 in all but a few countable cases, meaning that its existence or not would not affect the classification and thus was removed. Features f_2-f_7 had a strong correlation amongst themselves [6]. f_2 was always the largest for each individual example, and f_3-f_7 became progressively smaller, with f_7 being the smallest. Upon further investigation it was noted that the magnitude of the difference in values of f_2 and f_7 were strong indicators of the presence of DR or its absence. Since features f_6 and f_7 were approximately the same, feature f_7 was replaced by the magnitude of the difference in values of f_2 and f_7; intended to provide more information about the image. Features f_{16} and f_{17} had really small values with seemingly insignificant differences between examples (ranged 0.4-0.59 and 0.08-0.15 respectively) and thus were normalized to a range of 0-1 in order to represent the data with a more noticeable spread. All of the other non-binary features were not normalized to 0-1 because there was a correlation amongst features which would have been undermined had they been normalized.

2.2 Model Design

The inputs for the neural network consist of 18 features (from the original 19 features one had been removed)- in their processed form, as stated in the previous section. This is a binary classification problem; hence the output consists of two neurons, one that is meant to classify the image as showing signs of DR, and the other node which indicates the absence of DR. The neural network implemented is a two-layered neural network, meaning that it has one hidden layer.

Earlier on in the experiments the optimal hyper parameters were determined to be 140 neurons, with a learning rate of 0.001, and 2000 epochs. However, not all possible combinations of parameters could be tested, hence we need to look at other options that could help us find better parameters in a feasible amount of time. In order to find better hyper parameters for our model, we will make use of Genetic Algorithms. The details of the algorithm will be mentioned, in detail, in a forthcoming section.

The performance of the neural network is compared over the following activation functions: sigmoid, hyperbolic tangent, and rectified linear unit. Based on the investigation conducted, rectified linear unit gives significantly better results than the other two. Two different optimization techniques have also been used: stochastic gradient descent(SGD), and Resilient Backpropagation(RProp). SGD yields better results, while RProp results in overfitting, meaning that the training error is very low but the testing error is very high.

Given that the model takes time to converge, further technique/s need to be employed which would result in it converging faster without compromising performance.

2.3 Hyper-Parameter Search using Genetic Algorithm

Finding the optimal hyper-parameters can be a time-consuming task, and if done manually, can take very long. In order to solve this problem, we use a Genetic Algorithm that can determine good parameters over the course of a number of iterations. Since Genetic Algorithms are non-deterministic, and have random initializations, the parameters that they yield are not the same each time the Algorithm is run, however, its resulting parameters are still optimal, or very close to the optimal solution.

The number of neurons, learning rate, and number of epochs for the neural network have been determined using a Genetic Algorithm. The chromosome, used for representing one individual of a population, consists of three genes: the first gene represents the number of neurons, and its alleles are integers ranging from 90-140, the second gene is the learning rate, a float that ranges from 0.0005-0.002, and the last gene represents the number of epochs- its values are all integers between 800-2000. The ranges of all of the genes are a result of experiments conducted, where it had been determined that values belonging to these ranges would result in a neural network that would yield good results.

In order to run the Genetic Algorithm to find optimal hyper-parameters, the number of individuals in a generation are kept as 20, and the algorithm terminates after 5 generations have been produced. The reason there are more individuals than the number of generations, is to allow for more diversity. We could have had more generations, and less individuals, and introduced diversity using mutation, but in that case guaranteeing optimal outcomes is more tedious. The reproduction of new individuals is done using crossover with a probability of 0.9, and mutation with a probability of 0.1. The crossover approach used is a one-point crossover, and for mutation a new random number from the ranges specified, is selected and the old one is replaced. In order to select individuals for the next generation, the 10 individuals with the highest fitness value are chosen.

The fitness of an individual is determined by the testing accuracy that a neural network defined by the individual's genes, yields. Since each individual represents one neural network and given that we work with a total of 100 individuals, we

split our data using a hold-out method with an 80/20 split, as it gets too computationally expensive if a K-Fold Cross Validation approach is used. In order to allow for consistent evaluation of individuals, the split of the training and test sets is kept the same throughout the time that the Genetic Algorithm is run. Since we are basing our evaluation on the accuracy on a test set, this eliminates a chance of over-fitting on the training set. Our aim is to find the optimal parameters, and they are ones that yield a good accuracy, and since parameters with bad accuracy cannot be expected to work well, we have used a selection criterion for the next generation that is biased towards individuals with good fitness values i.e. the best individuals are selected to represent the next generation.

2.4 Evaluation Method

10-fold cross validation is used in order to evaluate the performance of the model. In this method the data is split into 10 partitions, and 10 different trainings are carried out, with each partition partaking the role of the test set exactly once. The purpose of this method is to prevent overfitting by exposing the model to different test sets, and thus yielding a model that gives good predictions on new data. This method also uses all of the data for training and thus utilizes all data for both testing and training, making it less susceptible to generalizing. Since the dataset used only has 1151 instances, splitting the data into a train and test set will leave very few instances for training, thus 10-fold cross validation will also ensure that there is sufficient data to train on, while also having enough test examples to effectively estimate the model accuracy. The measure used for evaluating performance is accuracy and is computed as:

$$\text{Accuracy} = \frac{TP+TN}{P+N}$$

Note: TP and TN represent true-positive and true-negative respectively, and P+N is the total number of test instances

2.5 Network Pruning using Badness

As mentioned previously, the initial neural network model yields good results, however it takes a long time to converge, and thus is not the most efficient. In order for the network to converge faster a technique called ‘badness’ has been employed which removes neurons from the network gradually [7]. This method works by detecting the ‘worst neuron’, which is essentially one that most significantly prevents the network from converging. For each neuron the badness factor is calculated-which is the sum of backpropagated errors over all input patterns. The neuron with the largest badness factor is one which has the most learning to do and therefore has the greatest cumulative error and is hence removed from the network so that it may converge faster. The inherent characteristics of the neural network are that it learns the most simplistic features first and takes it time to learn the more complex ones. Thus, if there exists a neuron with a high initial error, it could be the case that it is just learning a more complex feature and simply needs more time. In order to allow for more complex features to be learnt, the neurons are only pruned when the loss does not change drastically between a specific number of epochs. This way, when a lot of learning occurs, the overall loss will also differ drastically over iterations, and thus no pruning will occur. A lower bound also needs to be defined for how small a change in the loss accounts for pruning, because when the network is approaching convergence, there are minute variations only, and without a lower bound a lot of neurons will be pruned towards the end, even though these neurons have learnt features that help with the task.

Analysis of the threshold for change in loss over epochs lead to the conclusion that a percentage change in the total backpropagation error of 1-6% would result in the neurons getting pruned optimally. If the minimum is left to be lower than 1%, neuron/s are pruned despite having a very low error thus signally that it is learning a feature adequately- the removal of such a neuron would only result in a loss of learning and hence worsen performance. If the threshold is kept greater than 6% the performance depletes again because now neurons could be removed at earlier stages, meaning neurons taking time to learn complex features could possibly be removed. The number of epochs after which a neuron could be pruned were ranged between 5-100 epochs during experiments, with pruning every 35 epochs yielding the best results.

3 Results and Discussion

3.1 Results of neural network implementation

The first part of our implementation deals with finding the best hyper-parameters for our neural network model, using a Genetic Algorithm. The Genetic algorithm is non-deterministic, meaning that it yields different results each time it is run. On top of that, the neural networks are themselves non-deterministic, and since the fitness function is based on the output of a neural network, the outcome of the Genetic algorithm is likely to be different each time it is run. However, since the Genetic Algorithm aims to find the global maximum, the mean average accuracy of its final generation tends to be roughly the same, as witnessed during our experiments.

One interesting pattern observed during our experiments is that for the Standard Neural Network the Genetic Algorithm favors higher epochs, hence the best individual generally has number of epochs greater than 1600. Also, we found, that this is not the case for our Pruned Neural Network, for which the Genetic Algorithm does not necessarily yield the best

individual that has an epoch greater than 1600. This proves, that pruning the neural network does lead to faster convergence, and that is exactly why the Genetic Algorithm behaves the way it does.

For the purpose of our experiments, we use the individual from the final generation that yields the highest testing accuracy. Table 1 shows the optimal hyper-parameters found using the Genetic Algorithm.

Table 1. Optimal hyper-parameters

Type of Network	Number of processing neurons	Learning Rate	Number of epochs
Standard Network	137	0.001678	1747
Pruned Network	140	0.00123	1590

Based on the best hyper-parameters we found for the Standard Neural Network, we conducted experiments to see which optimizer is more suitable to solve this problem. As mentioned previously, SGD gives far superior results than RProp primarily because RProp has a tendency to overfit the model. Table 2 illustrates the difference in performance between SGD and RProp. From the results presented in Table 2, it is very evident that RProp leads to overfitting as the training accuracy is very high while the testing accuracy is low.

Table 2. SGD vs RProp, using Standard Neural Network

Optimizer	Mean Training Accuracy	Mean Testing Accuracy
Stochastic Gradient Descent	0.76	0.75
Resilient Propagation	0.93	0.70

The performance of different activation functions was also investigated, with Rectifier linear unit performaing significantly better than sigmod and hyperbolic tangent. Table 3. Illustrates the performance of each of these activation funcitons.

Table 3. Accuracy measured on the standard neural network with the optimal hyper-parameters

Activation Function	Performance Parameter (Accuracy)
Rectified Linear Unit (RELU)	0.75
Sigmoid	0.67
Hyperbolic Tangent (tanh)	0.70

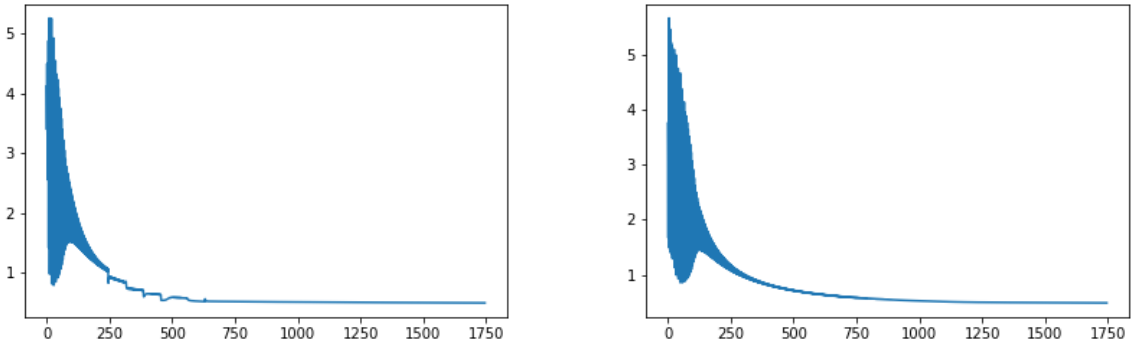
Table 4 summarises the results of the results obtained when using best hyper-parameters found by the Genetic Algorithm. In our earlier experiments we made use of 140 neurons, a learning rate of 0.001, and an epoch number of 2000 (all manually computed). Using these manually computed parameters, and our basic neural network we had found our best accuracy to be 0.75. The results we obtained using parameters found by the Genetic Algorithm (see Table 1) also yield the same accuracy, albeit with a fewer number of epochs needed in this case. This showcases the effectiveness of using Genetic Algorithms, whereby it was possible for us to explore a large search space in a feasible amount of time, and find a solution that is just as accurate as the older one, but computationally faster.

A method from literature called badness has been implemented on top of the standard neural network, in the hopes that it would yield a neural network that converges faster without compromisng on results. The results yielded by both the neural network with badness and the standard neural network have been presented in Table 4. These results have been obtained based on the best paramter settings for each mentioned earlier.

Table 4. Comparison of performance of networks based on respective optimal hyper-parameters

Type of Network	Testing Accuracy
Standard Network	0.75 ± 0.05
Pruned Network	0.745 ± 0.01

In order to compare the convergence rate of the two different neural networks consistently, we use the same hyper-parameters for both networks (number of neurons = 140, learning rate = 0.0016718, number of epochs = 1747). Based on this experiment it has been deduced that the pruned neural network starts to converge faster than the standard neural network. However, it reaches its best accuracy at roughly the same time as does the basic neural network. This is primarily because the pruning of neurons has been limited to a specific percentage change in total backpopagated errors (1-6 %), thus most pruning occurs in the earlier stages and the results start to improve earlier, but when pruning stops, the acceleration is also halted.



(a) (b)

Fig. 1. Total backpropagated error vs number of epochs *(a)* neural nework with badness *(b)* standard neural network

Figure 1 illustrates how the total backpropagated error varies with the number of epochs for both types of neural networks that have been implemented. This plot is taken from one iteration of the 10-fold cross validation, and the split of data is exactly the same in both cases, and both of the neural networks yielded a very similar mean test accuracy.The initial weights are different however. The plot for the basic neural network (b) is very inconsistent in the initial few epochs, but adjusts around the 200th epoch and then decreases consistently, with very minute reduction after 900 iterations approximately. Plot (a) initially showcases an irregular fluctutation of the loss, and starts to consistenly decrease around the 200th epoch. The jagged lines occurcing between epochs 200 and 550 occur as a consequence of pruning of neurons. The loss only insignificantly changes after the 600th iteration. Both of these plots make it evident that when badness is implemented the back propagated error decreases faster, but slows down once the pruning is halted, and as a result becomes optimal at roughly the same time as the original neural network. Both of the networks yield an approximate accuracy of 0.75 in this case.

3.2 Comparison of results with published work

In this section, the results of our neural networks are compared with a paper published on the same dataset- A Comparative Analysis of Classification Algorithms in Diabetic Retinopathy Screening [3]. This paper uses many different classifiers and reports the performance of each of the classifiers Table 5. Presents the performance of different classifiers presented in the paper. The data is evaluated using the 70/30 rule, with 70% of the data used for training and the remaining 30% used for testing.

Table 5. Performance of classifiers in comparison paper [3]

Classifier	Performance Parameter (Accuracy)
Multi-Layer Perceptron (MLP) neural network	0.8125
Adaptive boosting	0.8319
Gaussian Process	0.8707
Random Forest	0.8028
SVM	0.8449

The most relevant comparison seems to be between the neural networks proposed in this paper, and the MLP neural network used in the published paper. The neural network used in the published paper yields the best accuracy of 0.8125, while the proposed neural network yields an accuracy of 0.755 at best. Although the optimal parameters stated in the published paper were also tested out, it still did not yield the same performance, in fact it performed even worse than the model specified in the earlier sections of this paper. One reason for this could be that the features have been preprocessed in such a way that the published paper's neural network ends up giving a far superior performance. This would also justify why the two neural nets differ in performance even if the model design is completely replicated. Another reason for the disparity in performance could be because of the variation in the split of data. Perhaps owing to the 70/30 rule, the split of data in the published paper might have been ideal, resulting in the neural network making good predictions. Given the results for the 10 different test accuracies obtained using 10-fold cross validation, it is observed that the dataset displays high variance, with different splits yielding very different accuracies. Thus, perhaps the split of data used in the published work might simply be a lucky one.

For the rest of the classifiers, it seems that they are perhaps more suitable for classifying this particular problem. Given that nearly all of them- with the exception of random forests, performed better than the MLP neural network, it seems very evident that the classifiers will outperform neural networks on this problem. Gaussian Process [9] and SVM [10] give drastically better results than our model, and the primary reason for this is that they work better with smaller datasets,

whereas Neural Networks require a lot of data in order to yield better results. Since our dataset is relatively small, this deduction seems reasonable.

4 Conclusion and Future Work

In this paper, the UCI diabetic retinopathy dataset has been classified using a neural network. The neural network's hyper-parameters have been determined using a Genetic Algorithm. A neuron pruning technique called badness is also implemented in order to speed up convergence, and the results of the two variations of the neural network are compared with classifiers from a published paper. It has been found that the pruned network starts to initially converge faster, but once the neurons are no longer pruned, its speed also decreases, resulting in both models reaching optimal performance at similar times. The hyper-parameters found for both models by the Genetic Algorithm are different in the sense that the Genetic Algorithm determines that the pruned network can give optimal performance at fewer epochs than the standard neural network. From the comparisons, it is determined that the preprocessing of data and the neural network model can be modified further as better performance results for this dataset are possible.

Since the comparisons allowed it to be determined that the neural network could in fact yield better results, this leaves room for future improvements. One improvement could be that the data is preprocessed further. Perhaps more correlations are determined, or indistinct features are removed, or outliers are detected and removed so that the neural network may learn more distinct features from the dataset without overfitting, which will allow it to give better performance. Another future improvement could be to perhaps add a neuron intermittently- after a number of neurons have been pruned, as mentioned in a previously published paper [8], this could accelerate performance even further, and thus might solve the problem of the neural network with badness implemented slowing down towards the end of epoch iterations. The hyper-parameter search could also be made more extensive. If computation power exists, perhaps each generation can have more individuals, and the network might be run over more generations. This would ensure diversity, and as the search space is large, and may yield even better parameters than the ones already found.

References

[1] University of Iowa Health care, Ophthalmology and Visual Sciences, https://webeye.ophth.uiowa.edu/eyeforum/tutorials/diabetic-retinopathy-med-students/Classification.htm

[2] Wikipedia, https://en.wikipedia.org/wiki/Diabetic_retinopathy

[3] Mohammadian, S., Karsaz, A., Roshan, Y.M.: A comparative analysis of classification algorithms in diabetic retinopathy screening. 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE). (2017).

[4] UCI Machine Learning Repository, https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

[5] Antal, B., Hajdu, A.: An ensemble-based system for automatic screening of diabetic retinopathy. Knowledge-Based Systems. 60, 20–27 (2014).

[6] Machine Learning on the Diabetic Retinopathy Debrecen Data Set Data Set, https://rpubs.com/tiagotaveira/debrecen

[7] Gedeon, T.D., Harris D.: Network Pruning Techniques. Department of Computer Science, Brunel University.

[8] Hagiwara, M.: Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection

[9] Networks, G. (2018). Gaussian process vs Neural Networks. Retrieved from https://stats.stackexchange.com/questions/242602/gaussian-process-vs-neural-networks

[10] Which is more accurate, Neural Networks (NNs)or support vector machine(SVM)?. (2018). Retrieved from https://www.researchgate.net/post/Which_is_more_accurate_Neural_Networks_NNsor_support_vector_machineSVM