The Optimization of NN Classification: Based on Feature Selection with Genetic Algorithm & Hidden Neuron Pruning

Le Yang

Research School of Computer Science, Australian National University
U6023269@anu.edu.au

Abstract. Feed-forward neuron networks of several layers trained by back-propagation has been used to solve plenty of problems for research from the end of last century. Researchers requires more hidden neurons for some research. However, these excessive neurons which apply no efficacy in final products are redundant after learning. Similarly, researchers could specially select effective combination of features to input neuron instead of inputting all features. Thus, some researchers suggested that excess neurons could be removed, but older methods still have some disadvantages in efficiency. In this report, author tried to implement both neuron network pruning and feature selection based on Genetic Algorithm to delete redundant neurons automatically. This report would solve dataset classification problem. As the result, superfluous neurons would be pruned with little negative effects to accuracy (about 90%). Since this report took simple method, the result performs a little worse than some papers with more advanced method.

Keywords: Network Reduction, Genetic Algorithm, Feature Selection, Neuron Pruning, Classification, Mushrooms

1 Introduction

In this report, a simple feed-forward network with input, output layer and one hidden layer would be assumed. To simplify the discussion, we have assumed this problem is a binary classification problem. A set of inputs with desired outputs (i.e. target) would be used to train a network which applied back-propagation of error methods [1]. Sigmoid function has been chosen in this report as activation function in implementation for simplicity.

$$\text{Sigmoid\_function}(x) = (1 + e^{-x})^{-1}$$

1.1 Background

1.1.1 Introduction of Back-Propagation Algorithm(BP)

As the simplicity in theory, back-propagation algorithm is widely used from the last century to current in building Neuron Network. This algorithm and its derivative methods could be usually found in various area like “Handwritten Digit Recognition” [2] or “Automatic Control System Design” [3].

Although Back-propagation algorithm is diffusely used, the disadvantage of it also should be overcome. The major drawback of it is that back-propagation algorithm probably cost a big number of time to training network. In addition, although we can determine the input and output neuron with the architecture of dataset (attribute number for input and kind of classification for output), we are still unable to decide the number of hidden neuron easily.

Therefore, reduction of networks is focused by many researchers. As they indicated, using network reduction technology can avoid these two major disadvantages we mentioned before.

It can improve the efficiency of data testing, and more important, estimate the minimum number of hidden neurons to prepare the problems with similar size we would solve in future [4].

As the frontiers, Sietsma and Dow tried to prune trained networks in 1988. Their method, in brief, is removing neurons whose input is always near to zero and binding output of others as 0 or 1 in first step, then examining neurons output again for redundancy about the separation of classes within the given input space in second step [5].

1.1.2 Introduction of Genetic Algorithm(GA)

In brief, Genetic Algorithms are probabilistic search procedures which are designed to solve problems with "strings" states in large spaces [6]. As a part of Evolutionary computing, this bio-inspired heuristic algorithm, which could simulate an ecosystem with population and reproduction, is inspired by Theory of Evolution. Compared with some other heuristic rules, Genetic Algorithms could estimate a plenty of solutions.

Since the process of information in neural network(NN) is difficult to understand, it is important to generate solutions automatically rather than only engineering by human brain. Thus, people could design a better neural network with assistance of GA.

Crossover (i.e. mate) and mutate are both significant traits of Genetic Algorithm. Crossover could provide combination of fit genes from parents to child which usually better than individuals in generations before and mutate usually generate child which might be different with all the individual in population in last generation, which might lead to better solution [7].

To implement GA, we require a set of random numbers as “chromosome (also called DNA)” – usually built with binary numbers – to mask each subset. Only the feature whose counter point is “1” in chromosome could be kept to next step. Also, evaluation function is used to evaluate the fitness of chromosomes. A chromosome with better

fitness (higher or lower in this own evaluation system) could be easier to be continued as parent, and then reproduce the child with better fitness.

1.2 Preparation of raw dataset

This report requires a dataset for classification. To keep the complexity of dataset, it is necessary to choose a dataset with at least 300 instances and a minimum 15 attributes. Since larger size of instance will increase the training efficiency of network, the author chooses “Mushroom” dataset on UCI with 8124 instances and 22 attributes. This dataset has shown the classification result on the first column before 22 attributes vectors as "e"(edible) and "p"(poisonous), so that a binary classification network could be built in this report.

The problem we would solve is training a neuron network which accuracy is as high as possible for testing whether a mushroom has poison. The method to determine the performance of this network and prediction is estimating and comparing the training and testing accuracy. The higher the accuracy rate of testing is, the more accurate this classification network is. For training accuracy, this report would present a plot with the number of iteration epoch on x-axis and loss value (i.e. error) on y-axis. The function curve of this plot should rapidly decrease at the beginning, then the decreasing rate will be weakened as the epoch increasing to approach a stable loss value.

To show the result of classification, the confusion matrix could be used in the estimation step.

2 Method

2.1 Load and process dataset

The “Mushroom” dataset should be pre-processed before using as input.

Firstly, transform all the data from characters to integers.

Secondly, since only normalized value can be input as a dataset in this Linear Neural Network, normalizing these integer values to float values from 0 to 1.

Thirdly, shuffling these data by instances (randomly change the order of row) to avoid using similar instance in some order.

Next, randomly selecting parts of data as three different sets:

- Training set (70% of data);
- Testing set (15% of data);
- Validation set (15% of data);

Then, divide these three sets as 2 parts respectively:

- input data (including 22 columns of attribute value);
- target data (including the target value: edible or not).

The last step before building is transform training/testing input and training/testing target as Tensor type. Otherwise, the network data structure in torch cannot input these data.

We can save these results as a .csv file so that it can be loaded then without transforming step.

2.2 Preparing to build simple Neural Networks

To build NN, "learning rate (=0.28, which is related to gradient descent)" and "Epoch (=2000, should be large enough, but should not too big to decrease efficiency of implementation)" should be defined by researchers.

After setting parameters, the number of neurons in each layer should be defined initially. In this network, input neuron number should be 22(as the number of feature before implementing GA) and output should be 2("0" as poisonous and "1" as edible). As the report noted before, number of hidden units is difficult to define before learning with network. Thus, assuming number of hidden neuron as 50, the Linear network could be built and initialized with Sigmoid activation function.

Learning rate requires multiple times to test, so we can set it as a small number to observe the shape of function curve (x-axis: epoch times; y-axis: loss value), then increase or decrease it and observe again and again to select a fit value.

2.3 Iterating to build temporary NN and process GA

In this step, before the beginning of Genetic Algorithm procession, set all the important parameters like "Size of population (=15)", "Crossover rate (=0.8)", "Mutation rate (=0.01)" and "Number of generation (=80)" manually.

Before the starting, randomly set some binary words respectively in a list (which length is equal to the number of features = 22) and then duplicate it with several times (equal to "Size of population").

For each individual chromosome in each generation, train a temporary neural network with "Epoch" times iteration independently. After masking training input set with chromosome, input the rest feature into temporary NN as patterns. After enough iterations, to avoid the overfitting or coincidence of dataset, we should test each neural network with validation set instead of using testing set (i.e. using test set to process this step is same as "cheating") and then extract the "fitness" – which is usually decided by what kind of loss function (MSE or Cross Entropy) the neural network applied or only the accuracy.(This report applies accuracy as the result of fitness.)

Holding the collected list of fitness until the iteration of population finished, we can process the chromosome population in genetic algorithm:

- Selection: select the member of next population from the population before with their fitness. We choose proportional selection in this report.

For everyone in selected population:

- Crossover: produce offspring by recombining genes from parent individuals:
- Mutation: probably exchange some of numbers on chromosome for someone to another binary number;
- Replace: replace parents with offspring.

Then finish this time of generation and start the next time until finish all the generations.

After all the operation in this step, we can get a selected feature subset which is the best solution of the problems we solved (may not be optimal).

2.4 Pruning training neural network

2.4.1 Pruning Input neurons

With the best chromosome masking we collected in 2.3, we could delete the redundant input neuron before building the final neural network. We should both reduce input for training input set and testing input set.

2.4.2 Pruning Hidden neurons

According to the method from the paper "Network Reduction Technology" [4] which was written by Gedeon and Harris in 1991, this report implements that pruning hidden neuron by comparing inner angles of each vector. As this paper presented, the distinctiveness of hidden neurons is determined from their output vector which processed by activation function after processing input data. In other words, we should get the output of first Linear layer (from input to hidden) processed by activation function and divide them by column as many vectors (100 vectors in our network) which size is [approximate 70% * instances number (8124)] as 1-dimention array.

These vectors are used to calculate the inner angle of each pair of them.

As the paper mentioned, neurons with short activation vectors in input space should be removed because they are recognized as insignificant. Since all the value of vectors are between 0 and 1, we should minus 0.5 for all of them to broaden the range of possible value to get angles from 0° to 180° instead of from 0° to 90°. Then all the inner angle of each pair of vectors can be calculate and stored into lists in Python. As the paper shown, all the pairs of neurons whose inner angle of vectors are larger than 165° should be both removed. Then if there are some vector pairs whose inner angle are less than 15°, one of a them should be removed, and its value should be added to the other. We should note the pairs of neurons for both condition as two list.

After selecting neurons, we should replace the weight of new neuron network which requires no more training. The step flow is following:

- Calculate the number of hidden neuron in the new network.
- Build a new linear network with same parameters of the original one except hidden neuron number.
- Extracting the weight of Input-Hidden layer and Hidden-Output layer of original trained network.
- Removing all the neurons whose angles are bigger than 165° and merging neurons whose angles are smaller than 15° for both weight data.
- Replace the both weights of new network and we can test it with pruned network.

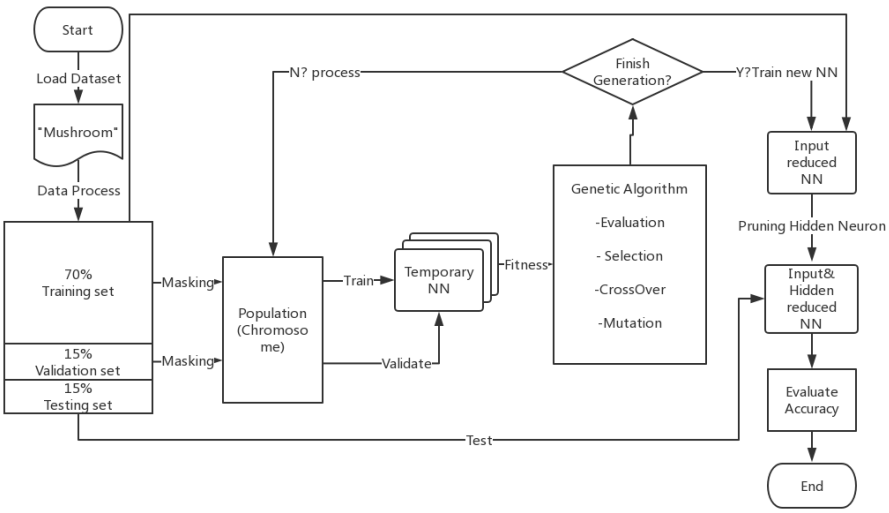

Fig. 1. General Flow

2.5 Network Testing

Using test set to evaluate both the final NN and the pruned final NN. We can get the result and discuss it then.

3 Evaluation, Results and Discussion

3.1 Evaluation of Neuron Network

An efficient neuron network requires a suitable learning rate value. As the report mentioned in stage 2, to get an appropriate learning rate, we should test different value again and again.

If a too small value was chosen to build a neuron network, the accuracy might be lower than the best situation of network; If the value is too big, although the training accuracy would be presented as a big value, the testing accuracy would still worse than the best situation (i.e. it would be overfitting).

The learning rate, as a hyperparameter which is hard to be determined, is a coefficient of gradient descent. Thus, the change of learning rate could adjust the gradient descent speed into a suitable range. The loss function cannot easily converge with too small value – difficult to descent to the bottom of gradient model – and it also could shake on y-axis violently because in gradient descent aspect, too big learning rate could let the object go through the bottom of gradient model.

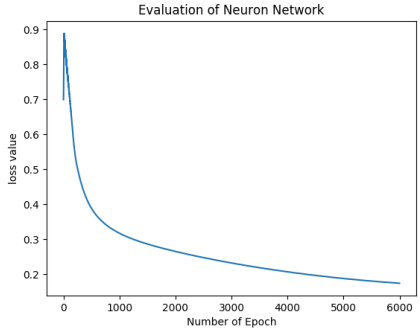

Fig. 2. A standard learning rate: learning rate = 0.18

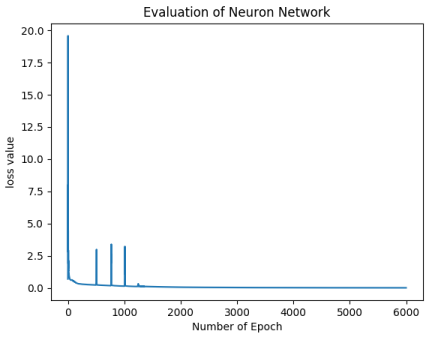

Fig. 3. Too large learning rate: learning rate = 1.8

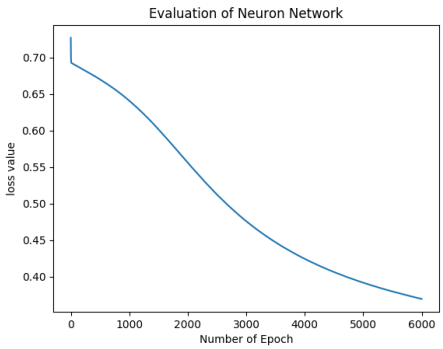

Fig. 4. Too small learning rate: learning rate = 0.018

3.2 Result of Pruning

To compare the result of pruning, we collect 10 sets of result of testing accuracy of comparing between original network and pruned network:

	Accuracy of training set	Accuracy of testing set	Accuracy of pruned testing set	Number of pruned hidden neurons
1	93.96%	93.26%	93.38%	93
2	93.60%	94.25%	92.68%	89
3	93.74%	92.70%	92.45%	91
4	93.90%	93.61%	92.08%	92
5	93.43%	93.68%	92.20%	90
6	93.71%	93.57%	91.58%	92
7	93.62%	94.03%	92.50%	96
8	93.54%	93.35%	93.11%	91
9	93.82%	93.95%	93.52%	96
10	93.70%	92.88%	92.94%	90

Table 1. The comparation result of 10 set with 100 hidden neurons before pruning with learning rate = 0.18 and Epoch = 6000

According to the result of Table 1, we can observe that all the accuracy here are over 90%. Accuracy of training set are stable around 93.5%; Accuracy of testing set are similar as training with a little waving; Accuracy of pruned training set are usually a little lower than original testing result but sometimes bigger; The number of pruned hidden neurons are usually 5%~10% lower than original number.

These result shows that the approprate pruning is useful to reduce the size of hidden layer. Although the accuracy could be still gently decreased, we pruned network with the lowest cost. Thus, this pruning method can be implemented for our dataset.

Pruning hidden neuron with this method can really help us to find the minimal size of hidden neurons. Since the hidden neuron number is reduced, it seems also a good way for larger testing set in future.

More importantly, this method can reduce the hazard of pruning instead of randomly deleting hidden neuron. For instance, we can reduce 10% of neuron with only 2% decrease of accuracy with this function, but if we try to reduce same number of neurons randomly, the accuracy might be declined by over 50% and the network would be no use.

3.3 Result of Genetic Algorithm:

	Acc of train set with GA[1]	Acc of train set without GA	Acc of test set with GA	Acc of test set without GA	Acc of pruned test set with GA	Acc of pruned test set without GA
1	90.21%	91.05%	90.99%	90.94%	90.41%	89.95%
2	90.18%	91.09%	92.00%	90.62%	87.33%	85.51%
3	86.95%	90.83%	87.28%	92.27%	80.96%	90.61%
4	90.67%	91.45%	90.48%	89.82%	87.97%	85.51%

[1] The control group without GA has same hyperparameters with other groups with GA.

5	91.40%	91.50%	91.35%	91.59%	91.67%	91.78%

Genetic Algorithm in this report has high time complexity. Therefore, we have to decline the Epoch from 6000 to 2000 (with learning rate = 0.28) to evaluate the performance of these instance.

The 5 sets of result without genetic algorithm looks like similar which could fit to the conclusion we have got in 3.2. The other 5 sets experimental groups still shows the similar result, so it seems that pruning inputs with suitable feature selection cannot reduce the accuracy in most of times – it seems a little more efficient than the control groups without GA which means that fit feature combination could improve the efficient of neural networks.

3.4 Comparing with other researchers

Since the results in this report were made by a simple linear network (and pruned by a simple method which was provided in 1991[4]), we have enough reason to believe that there are many papers which have higher accuracy as result with the same dataset "Mushroom".

This report choose the result from the paper "Extraction of crisp logical rules using constrained backpropagation networks" [8] which has published in 1996. It provides four different results which collected by four different logical rules of selecting input attributes. The result of them are respective as 98.52%, 99.41%, 99.90% and 100.00% which are much bigger than the result we collected before as 93.26%.

However, since the result showed that GA could select a better combination of features automatically, if we have enough time to iterate the generation, we might find the same combination like the manual selection from the paper written by Duch et al[8].

The other reason of this phenomenon might be the difference of network architecture. In this paper, researchers use MLP2LN [2] and Structural Learning with Forgetting (SLF) method [9] to build a more complex network than the simple linear back-propagation network we built.

As this paper provided, the major reason might be the selection of input set. In our implementation, all the 22 attributes were input into the simple network. Although the more attribute can usually support a better accuracy result, if a good combination of input was found for some dataset, the accuracy would become almost perfect. However, not all the attributes are obvious enough like distinguish poisonous mushroom (like the 100% accuracy combination is habitat and cap-color).

4 Conclusion

Although the result of pruning is representative enough without influence from other factors, the accuracy is also shown dissatisfactory since the neuron network in this report is too simple. Fortunately, the pruning is useful for reduction of hidden neuron with a little cost of accuracy. This method is too old, but reduction of network is still

[2] a smooth transition from Multi-Layer Perceptron to Logical Network

researched by people at present. For instance, Zeng and Yeung provided a paper as pruning hidden neuron with a quantified sensitive measure [10]. Pruning could be always a useful topic in neuron network, because people should process enormous datasets in future which require much more efficiency.

Genetic algorithm is potential to automatically generate solutions. Although it could cost plenty of time to select features initially, researchers could get much more accurate result with less input of features in future works – eliminate redundant data could make the programs easier. In addition, genetic algorithm cannot only be used to select feature, but it can also select hyperparameters (hard to predict) without the planning of people. In other words, we can also prune hidden neurons with GA-ANN[7].

5 Reference

1. Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning internal representations by error propagation (No. ICS-8506). California Univ San Diego La Jolla Inst for Cognitive Science.
2. LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Advances in neural information processing systems (pp. 396-404).4. Gedeon, T. D., & Harris, D. (1991). Network reduction techniques. In Proceedings International Conference on Neural Networks Methodologies and Applications (Vol. 1, pp. 119-126).
3. Mehedi, I. M. (2017). Time Varying Back Propagating Algorithm for MIMO Adaptive Inverse Controller. INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 8(2), 370-377.
4. Gedeon, T. D., & Harris, D. (1991). Network reduction techniques. In Proceedings International Conference on Neural Networks Methodologies and Applications (Vol. 1, pp. 119-126).
5. Sietsma, J., & Dow, R. J. (1988, July). Neural net pruning-why and how. In IEEE international conference on neural networks (Vol. 1, pp. 325-333). IEEE San Diego.
6. Kermani, B. G., White, M. W., & Nagle, H. T. (1995, September). Feature extraction by genetic algorithms for neural networks in breast cancer classification. In Engineering in Medicine and Biology Society, 1995., IEEE 17th Annual Conference (Vol. 1, pp. 831-832). IEEE.
7. Ahmad, F., Mat-Isa, N. A., Hussain, Z., Boudville, R., & Osman, M. K. (2010, July). Genetic Algorithm-Artificial Neural Network (GA-ANN) hybrid intelligence for cancer diagnosis. In Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on (pp. 78-83). IEEE.
8. Duch, W., Adamczak, R., & Grabczewski, K. (1997). Extraction of crisp logical rules using constrained backpropagation networks.
9. Ishikawa, M. (1996). Structural learning with forgetting. Neural networks, 9(3), 509-521.
10. Zeng, X., & Yeung, D. S. (2006). Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure. Neurocomputing, 69(7-9), 825-837.