Distinctiveness and Evolutionary Algorithm on Neural Network Reduction

Xinyu Hou

College of Engineering & Computer Science, Australian National University

{u5916376}@anu.edu.au

Abstract. In this paper, I constructed a feed-forward neural network with a non-specialized architecture to solve classification problem. Later I reduced the neural network by applying the distinctiveness analyze method and the evolutionary algorithm pruning method. Both pruned networks still guarantee a consistent quality without need of further training. As the pruning rate increases, the distinctiveness analyze method's classification accuracy slightly decreases, while the evolutionary algorithm pruning method still keep a high performance on classification accuracy. Comparing the performance with other paper which used the same dataset. The Linear Discriminant Analysis classifier has the best performance on classification accuracy than other classification methods, followed by my evolutionary algorithm pruned neural network's performance, while the K-Nearest Neighbors (3) classifier has the worst performance, and has the highest classification error.

Keywords: Neural Network, Reduction, Distinctiveness, Evolutionary Algorithm, Pruning.

1 Introduction

In this report, I solve a classification problem by modeling and training a neural network. After the basic classification functionality be implemented, I prune the network by removing the hidden units that with redundant functionality after training. The neural network automatically locates and removes the redundant hidden units by implementing a *distinctiveness* automatable process which is mentioned by Gedeon and Harris [1], and the neural network automatically adjust networks weights, so the pruning trained network still guarantees a consistent level of functionality of hidden units and without the need for further training. In order to evaluate how the performance of *distinctiveness* method different with the performance of other neural network pruning method, I implement a further *evolutionary algorithm* pruning method on the same basic classification network and on the same data set as the comparison.

The neural network is a feed-forward network of three layers of processing units. All the connections are from units in one layer to each unit in subsequent layer, with a simple non-specialized architecture. The neural network uses stochastic gradient descent as the optimizer, sigmoid as activation function, and use the backpropagation to adjust the weight of neurons by calculating the gradient of the loss function.

The neural network use the *Leaf* dataset [2] as the training and testing dataset, which has 16 attributes, 340 instances, and with no missing values. This dataset is suitable for my classification neural network, which can classify different leaf species by training the leaf species and leaf features attributes. I also measure the test dataset results on both original network and pruning trained networks, and compare the difference between their classification accuracy, confusion matrix, hidden layer size, pruning rate and consumed time. Performance of the pruning trained networks will be analyzed and evaluated base on those measured results.

Getting a faster or a smaller neural network without impacting performance is important for training a bigger and deeper neural network on a device with limited hardware resources and power constraints [6]. The aim of this report is to implement the *distinctiveness* analyze method and *Evolutionary Algorithm* method of pruning a neural network, and to measure and evaluate the performances between original neural network and pruning trained neural networks.

2 Method

2.1 Data Set

The dataset that I chose for the classification problem is the *Leaf* dataset [2] which is provided by UCI. This dataset has 16 attributes, 340 instances, no missing values, and all the attributes in this dataset are in real number type. For the 16 attributes, the first attribute represents the *Leaf Species* and consists the number from 1 to 36, each number represents one *Leaf Species* (for example, number 1 represents 'Quercus suber', number 36 represents 'Geranium sp.', and so on). Each attribute from 3 to 16 represents a *Leaf Feature*, in more detail, attributes 3 to 9 represents different leaf shapes, and attributes 10 to 16 represent different leaf textures [3]. The second attribute represent the *Specimen Number*, which is irrelative to the leaf classification problem, so this attribute was dropped from both the training dataset and the testing dataset before I train the network.

When I train the neural network, the *Leaf Species* in first attribute are to be treated as the leaf classification neural network's outputs, and the 14 *Leaf Feature* attributes are to be treated as the leaf classification neural network's inputs. Besides, 80% of the instances are used to train the neural network, and 20% of the instances are used to test the trained neural network.

The reasons to choose this *Leaf* dataset are:

- This *Leaf* dataset has enough instances and attributes for training and testing a neural network
- This *Leaf* dataset has enough leaf's features as inputs and leaf's classes as outputs for a classification problem.
- This report compares the original neural network's performance with the pruning trained neural network's performance by calculating the accuracy of classification, so a 36 classes dataset is better than a binary classes dataset. That is because, for the binary classes dataset, there is around fifty percentage to classify an instance correctly even if the network works not correctly. More output classes could reduce the possibility of irrelative factors impact the classification accuracy.
- Because this report is mainly focus on the performance after a neural network's size is pruned, a dataset only in real number type and with no missing values is enough for this neural network's training and testing. The attributes' type of the dataset will not greatly impact the pruning method that this report will use, and the pruned network's performance.

2.2 Basic Classification Neural Network's Model Design

For my basic classification neural network, I construct a three-layer neural network which comprises one input layer with 14 neurons, one hidden layer with 100 neurons, and one output layer with 36 neurons. The 14 input neurons are for inputting the 14 different leaf's features, and each of the 36 output units is for one leaf specie. The hidden layer's size is initialized to 100 units, and for controlling the variables and for mainly focusing on the different performance between the original network's size and reduced network's size later, the bias is not used in the neural network's structure.

For pre-processing the dataset, I dropped the second irrelative attribute and update the data index, then randomly split the dataset into training set which uses 80% instances of the original dataset and testing set which uses 20% instances of the original dataset, and the same testing set is for testing both original network's performance and pruned network's performance later. The raw dataset is transformed to torch dataset, and the network will use Dataloader to read the data during the training process.

The feed-forward neural network uses sigmoid function as the activation function, and uses stochastic gradient descent as the optimizer. The network use the backpropagation to adjust the weights by calculating the gradient of the defined loss function. The training process trains the network with 100 epochs and with the batch size 5. For every 50 epochs, the processing of the last epoch will be print. The training results will be present by using the measure results method which I will mention later.

2.3 Pruning Neural Network Methods

The investigation's purpose is to find ways to reduce neural network's hidden layer size and will not affect the quality of the network's performance greatly. Small size network can save both training time and processing time, and more training iterations could be done in the same time for the smaller networks [4]. In this report, I implement two pruning methods to reduce the network size. The first method is to construct a neural network which can automatically detect the redundant units, the redundant units are measured and decided by the *distinctiveness* automatable process. The network then automatically remove those duplicative units and adjust the network's weights, to minimize the negative effect of the pruned network's performance. The second method is to implement an evolutionary algorithm method to prune the network. The network

initializes a fixed size population of different hidden layers' states, then does the evolutionary process to evolve the population until the limited number of generation reached. The neural network automatically replaces the current worst performance individual in the population by a better performance new child individual without further training.

Distinctiveness analyze method

Some hidden units have the similar functionality with some other hidden units in the same layer, units with redundant functionality after training can be removed. For the first method, I will focus on the *distinctiveness* property of hidden units, and will use this property to remove the redundant units [1]. The *distinctiveness* is determined from the hidden units output activations vector over the epochs, that is, for each hidden unit, I construct an output activations vector of the same dimensionality as the number of total epochs, to record one output activation for every epoch. And finally, I construct an *activation_matrix* to store all the output activations vectors of all hidden units after training the network, with the matrix size of *number_of_hidden_units * number_of_epochs*.

However, in each epoch, the neural network will output training dataset's size of activations for each unit, but I only want to get one activation from all the activations of each epoch, and to store this chosen activation into the final *activation_matrix* for the further operations. I randomly choose one activation from all the activations of each epoch. Because the instances' order of the training dataset is set to random, the activation which will be chosen is the first instance's hidden output activation of each epoch's last batch. By this method, in each epoch, the network randomly get only one activation for each hidden unit to store into the final *activation_matrix*.

After the neural network training, the network fulfills the *activation_matrix*. I got one epochs' size of activations vector for each hidden unit. To recognize the similarity of pairs of hidden units, I made a method to calculate the angle between pairs of units' activations vectors. If angular separation is below 15°, then the two units are considered too similar and one of them is removed, and the weights vector of the unit which is removed is added to the weights vector of the unit which remains. If the angular separation is over 165°, then the two units are considered complementary, and both are removed [1].

This pruning method does not need any further training, so I stored the pruned weights of each layer and the pruned hidden layer size. Then I initialized a new neural network which is same to the original network, but with the stored pruned weights and stored pruned hidden layer size, as the pruning trained neural network.

Evolutionary algorithm pruning method

For the second method, I do not remove the similar functionality hidden units anymore, I use the *evolutionary algorithm* pruning method to initialize a fixed size population of random hidden layers' states, calculate the fitness score for each individual, generate varied children from the population, and optimize the population to a higher fitness score trend. To compare with the *distinctiveness* pruning method, the *evolutionary algorithm* pruning method does not add the removed unit' weights to the remained unit' weights anymore, but reduce the hidden units' size with a more free and straightforward way, that is, the method may have more possibilities to test more varied hidden layers' states and the method just keeps or removes the hidden units without any further weights' operations.

To represent and store each individual's hidden layer's state and fitness score, I made an *individual_score_matrix* with *population_size* rows and *hidden_size + 1* columns, each row represents one individual, and there are *hidden_size + 1* elements for each individual to store the hidden layer's state and fitness score. The first *hidden_size* elements represent the state of *hidden_size* hidden units, that is, each element in the first *hidden_size* elements represents each hidden unit's state, and the element's value 1 means to keep this corresponding unit, element's value 0 means to remove this corresponding unit (For example, 1, 1, 0, 0, 0, …, 0, 0, 0 means this individual only keep the first two hidden units and remove other hidden units). The first *hidden_size* elements are used for pruned the network later. The last element of each row is to store the fitness score of each individual, and the fitness score is the accuracy of the corresponding pruned network.

The fitness function is to calculate the classification accuracy of each individual. The first step of the fitness function is to prune the network based on individual's hidden layer's state, after removed the 0 value hidden units and their weights, I initialized a new neural network which is same to the original network, but with the pruned weights and pruned hidden layer size, as the pruning trained neural network. The second step is to return the classification accuracy for this pruned network by calculating the accuracy on the test set dataset.

For the whole *evolutionary algorithm* process, I initialize the population of candidate by randomly assigning a binary value which ranges from 1 to 2^(*hidden_size*) – 1 (i.e. at least keeps 1 hidden unit to 111…111, keeps all hidden units) to each individual, and store this hidden layer's state into the *individual_score_matrix*. Uniform random initialization is used to ensure that the initial population is a uniform representation of the entire search space. Then I calculate the fitness score of each individual through the fitness function, and store the score into the *individual_score_matrix* as well. Two Parents are selected from the population through the *Roulette Wheel Selection* method. *One-Point Crossover* method is used to generate two children from the selected parents, and the *Uniform Mutation* method is to mutate each child thus adds diversity to the genetic characteristics of the population. Then the mutated child's fitness score is calculated through the fitness function, and a *Replace worst* replacement strategy is used to optimize the population to a higher fitness score trend, where the children replaces the worst individual of the current population. Finally, the evolutionary process will stop after *limited_generations* generations are reached.

2.4 Measure Results Method

For the training results, I calculate the training accuracy by dividing the number of correctly classified instances by the number of total classified instances. Training set accuracy can be used to find if the training process has the high possibility of overfit, for example, the training set accuracy is always 100% may shows the network has already overfitted. I also calculate the confusion matrix, which can be thought as a table with two dimensions 'actual' and 'predicted', and has identical sets of 'classes' in both dimensions. The confusion matrix allows a visualization of the performance of the classification.

For the testing results, I test both the original size network and the pruned networks to compare the performance between them. I calculate the testing set accuracy, and the confusion matrix for both networks. Furthermore, I measure the consumed time of getting the classification outputs by testing the corresponding network. And I also measured the pruning rate by dividing the pruned network hidden layer's size by the original hidden layer's size.

3 Result and Discussion

3.1 Analyze and Evaluate the Testing Set Results by using the *distinctiveness* analyze method

- Relationship between pruning rate and saved testing time

To analyze and evaluate the relationship between pruning rate and saved testing time, I make a data chart by randomly testing 15 times by using the *distinctiveness* analyze method, to get 15 sets of data results, and draw the results on the graph to have a visualization of the performance. The pruning rate is provided by the measure results method, and the saved time is calculated by subtracting pruned network testing consumed time from original network testing consumed time. Set original networks' hidden layer size to 100.

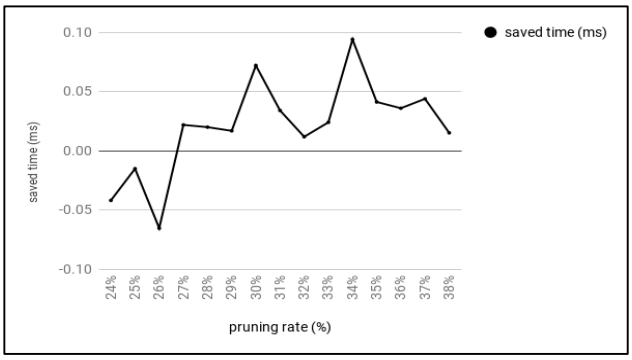

Figure 1. Relationship between pruning rate and saved testing time

This line chart (Figure 1) shows that, with the increment of the pruning rate, more testing time is saved by getting the outputs from the pruned neural network. This result proves the previous statement that Lawrence, Lee Giles and Chung Tsoi mentioned [4]: a smaller neural network could save more processing time.

- Relationship between pruning rate and test set accuracies on original network and *distinctiveness* pruned network

To analyze and evaluate the relationship between pruning rate and test accuracies of both original networks and *distinctiveness* pruned network, I make a data chart by randomly testing 15 times by using the *distinctiveness* analyze method, to get 15 sets of data results, and draw the results on the graph to have a visualization of the performance. The pruning rate and the test accuracies are provided by the measure results method. Set original networks' hidden layer size to 100.

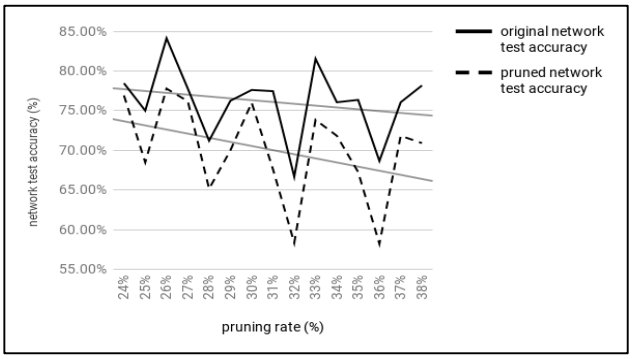

Figure 2. Relationship between pruning rate and test accuracies on original network and *distinctiveness* pruned network

In this line chart (Figure 2), we can find that, the *distinctiveness* pruned network test accuracy's change is closely following the change of original network test accuracy. To get more information from this chart, we can compare the trend lines between the original network's test accuracy and pruned network's test accuracy. The two relative trend lines show the pruned network test accuracy is down slightly with the increment of the pruning rate. Because the two trend lines are not parallel, instead, the distance between two trend lines is increased with the increment of the pruning rate.

3.2 Analyze and Evaluate the Testing Set Results by using the *Evolutionary Algorithm* Pruning Method

- Relationship between pruning rate and test set accuracies on original and *Evolutionary Algorithm* pruned network

To analyze and evaluate the relationship between pruning rate and test set accuracies of both original networks and *Evolutionary Algorithm* pruned network, I make a data chart by randomly testing 15 times by using the *Evolutionary Algorithm* method, to get 15 sets of data results, and draw the results on the graph to have a visualization of the performance. I use the best accuracy in the population to represent the accuracy of *Evolutionary Algorithm* pruned network. The pruning rate and the test accuracies are provided by the measure results method. Set original networks' hidden layer size to 100.

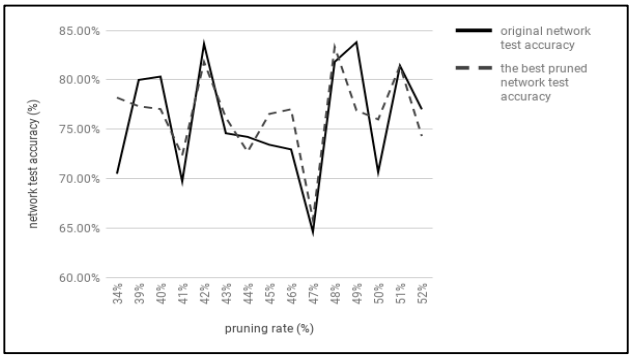

Figure 3. Relationship between pruning rate and test accuracies on original network and *Evolutionary Algorithm* pruned network

In this line chart (Figure 3), we can find that, the *Evolutionary Algorithm* pruned network's best test accuracy is similar to the original network test accuracy, and sometimes the pruned network's accuracy even exceeds the original test set accuracy. For example, when the pruning rate reached 50%, the pruned network's best test accuracy (76%) is higher than the original test set accuracy (70.67%), that is, I get a better performance after I reduced the neural network, which achieves the previous expect: the pruned network still guarantee a consistent quality.

3.3 Compare the Results between the Two Pruning Methods

- Compare the pruned networks' accuracies with the original network' accuracy

To compare the results between the two pruning methods, one important feature is that, the pruned network's accuracies in the *Evolutionary Algorithm* pruning method can exceed the original network's accuracy (Figure 3), while the pruned network's accuracies in the *distinctiveness* pruning method are always slightly below the original network's accuracy (Figure 2). The *Evolutionary Algorithm* method shows that there are some redundant hidden units in the original network, after removed these redundant hidden units, the pruned neural network can even get a better performance than the original size network's performance.

- Compare the pruning rate between two pruning methods

By comparing the results from the two pruning methods, we can find that, the *Evolutionary Algorithm* pruning method can prune to a smaller neural network (with the average pruning rate of 45%) without impacting performance than the *distinctiveness* pruning method (with the average pruning rate of 31%). The *Evolutionary Algorithm* pruning method also got a higher pruning performance (pruning rate range from 34% to 52%) than the *distinctiveness* pruning method (pruning rate range from 24% to 38%).

- Compare the pruned networks' accuracy between two pruning methods

Comparing the accuracy between two methods, the *Evolutionary Algorithm* pruning method's accuracies can exceed the original network's accuracies while the *distinctiveness* pruning method cannot exceed the original network's accuracies. The *Evolutionary Algorithm* pruning method's average accuracy (76.48%) is also higher than the *distinctiveness* pruning method's average accuracy (70.02%).

3.4 Compare and Evaluate the Results with Other Paper that Use Same Dataset

Pedro Silva also used same dataset with mine for classifying the leaf species [5]. The classifiers that he used are the Linear Discriminant Analysis (LDA) and the K-Nearest Neighbors (KNN). In fact, LDA is the simplest parametric classification technique and KNN is the simplest non-parametric classification technique [5].

To comparing the results with Pedro Silva's result [5] (Table 1.), I calculated my 'Test Set Classification Error' by get the average testing accuracy of both original network and pruned networks from the previous fifteen accuracy data (Figure 2 & Figure 3), and simply use one minus each average testing accuracy to get the 'Test Set Classification Error' for each network, at the final, the results of my networks' performance are showed in Table 2.

Table 1. Shape and texture features: Summary of classification results.

Method	Cross-validation Error	Classification Error
LDA	22.30%	15.70%
KNN (1)	31.90%	28.10%
KNN (3)	30.70%	32.60%
KNN (5)	33.80%	27.00%

Table 2. Classification Error of different networks

Pruning Method	Original Test Set Classification Error	Pruned Test Set Classification Error
Distinctiveness	23.90%	29.98%
Evolutionary Algorithm	24.08%	23.52%

To compare the performance between different methods, the classification errors of each method are evaluated. Comparing with the above two result tables, the parametric classification technique LDA (in Table 1.) has the best performance with the lowest classification error 15.7%, and then the second-best method is the *evolutionary algorithm* pruned neural network (in Table 2.) which has a 23.52% classification error. On the other hand, the method with the worst performance is KNN (3) (in Table 1.), which has a highest classification Error 32.60%. All the methods worked well with this *Leaf* dataset [2] with an average classification accuracy at least higher than 65%, and the best classification result was using linear discriminant analysis as classifier.

4 Conclusion and Future Work

I have constructed a feed-forward neural network to solve a classification problem, and used the *distinctiveness* of hidden units to find the units with similar functionality, and pruned the network by removing the redundant functionality units. With the pruned network weights being adjusted appropriately, the pruned network can still guarantee a consistent quality. The classification accuracy will slightly decrease alone with the increment of the pruning rate. I also used the *evolutionary algorithm* pruning method to reduce the same basic network, and got a high performance on both classification accuracy and pruning rate. In this method, I got a better average classification accuracy than the *distinctiveness* pruning method's average classification accuracy. Both the pruning methods need no more further training. However, the *distinctiveness* pruning method still has limitation, while the *evolutionary algorithm* pruning method's classification accuracy can exceed the original network's classification accuracy but the *distinctiveness* pruning method's classification accuracy is always less than the original network's classification accuracy.

The current *distinctiveness* pruning trained net still has limitation, for the next step, I will widen the current *distinctiveness* method, to analyze the similar functionality of groups of three or more units which together have no effect, or two or more units with a constant effect, to improve the performance of more complex pruning situations.

5 References

1. Gedeon, T., Harris, D.: Progressive image compression. In Neural Networks, 1992. IJCNN., International Joint Conference. IEEE. Vol. 4, pp. 403-407, (1992).
2. Dua, D., Karra Taniskidou, E.: UCI Machine Learning Repository, http://archive.ics.uci.edu/ml.
3. Silva, P., Marcal, A., Rubim Almeida da Silva: "leaf" dataset. (2014).
4. Lawrence, S., Ah Chung Tsoi, Lee Giles, C.: What size neural network gives optimal generalization? convergence properties of backpropagation, http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.6019.
5. Silva, P.: Development of a System for Automatic Plant Species Recognition, http://repositorio-aberto.up.pt/handle/10216/67734.
6. Nguyen, N.: Pruning AI networks without impacting performance, https://www.ibm.com/blogs/research/2017/12/pruning-ai-networks/.

6 Appendix: Experimental Data for Result and Discussion Section

Table 3. Experimental data for Figure 1

No.	pruning rate (%)	saved time (ms)

1	24%	-0.04172325
2	25%	-0.01502037
3	26%	-0.06509792
4	27%	0.02193451
5	28%	0.02002717
6	29%	0.01692772
7	30%	0.07200241
8	31%	0.03409386
9	32%	0.01192093
10	33%	0.02408027
11	34%	0.09393692
12	35%	0.04124641
13	36%	0.03600121
14	37%	0.04386902
15	38%	0.01525879

Table 4. Experimental data for Figure 2

No.	pruning rate(%)	original network test accuracy (%)	pruned network test accuracy (%)
1	24%	78.46%	76.92%
2	25%	75%	68.48%
3	26%	84.13%	77.78%
4	27%	77.78%	76.19%
5	28%	71.21%	65.15%
6	29%	76.25%	70.00%
7	30%	77.61%	76.00%
8	31%	77.46%	67.61%
9	32%	66.67%	58.33%
10	33%	81.54%	73.85%
11	34%	76.06%	71.83%
12	35%	76.36%	67.27%
13	36%	68.66%	58.21%
14	37%	76.06%	71.83%
15	38%	78.18%	70.91%
average value	31%	76.10%	70.02%
classification error		23.90%	29.98%

Table 5. Experimental data for Figure 3

No.	pruning rate(%)	original network test accuracy (%)	the best pruned network test accuracy (%)
1	34%	70.51%	78.21%
2	39%	80.00%	77.33%
3	40%	80.33%	77%
4	41%	69.74%	72.37%
5	42%	83.64%	81.82%
6	43%	74.60%	76.19%
7	44%	74.24%	72.73%
8	45%	73.44%	76.56%
9	46%	72.97%	77.03%
10	47%	64.63%	65.85%
11	48%	81.82%	83.33%
12	49%	84%	76.92%
13	50%	70.67%	76%
14	51%	81.43%	81.43%
15	52%	77.03%	74.32%
average value	45%	75.92%	76.48%
classification error		24.08%	23.52%