Building Up the Proper Neural Network: Evolutionary Algorithm and Hidden Layer Compression on Classifying Genuine vs Posed Anger

Fan Wu
ResearchSchool of Computer Science,
Australian National University
Canberra Australia
u7155005@anu.edu.au

Abstract. Building a neural network to learn with perceiver's pupillary response patterns and the anger they saw is genuine or posed, we can get a relatively high accuracy, which means these two factors are deeply connected. We also conduct some basic parameter tuning stretagies for the the model, and make the network performence better. Futhermore, we implement method to optimize and compress the model structure, analyzing the similarity of hidden units and decided which of them should be removed. And we achieve the goal that without significantly performance decline, the network's redundant neurons are removed and the speed of training is rasied. Finally, we use evolutionary algorithm to retrain our network. After comparing the results of two training methods, we find the similar results of two optimizing methods can indicate that with the structure of 3-layer simple neural network, the model has probably reached its maximum accuracy of around 86%.

Keywords: Neural Network, Anger Classification, Pruning Network, Evolutionary Algorithm

1 Introduction

Anger expression can be genuine or posed, and sometimes people deliberately act anger to manipulate other's emotion for some purpose [1]. So whether people are able to discriminate the two kind of anger is the main problem that we want to address in this paper. We will use a set of data from [1] containing information about perceiver's pupillary response and the anger the saw is genuine or posed and build a neural network to train on it. If the model has a good performance, we can say that perceivers discriminate the two kind of anger even unconsciously. Also, we want to implement some network optimization and reduction methods according to [2], to improve our neural network as good as possible to prove the relationship between the two factors.

Evolutionary algorithm uses mechanisms inspired by biological evolution. The procedure in natural world like reproduction and gene mutation are imitated in evolutionary algorithm. The possible solutions of question to be addressed are coded into strings. We can evaluate these strings by calculating their fitness to the problem. Then we manipulate these strings to imitate natural procedures like mutation and select good offspring every generation. With the generation grows, the solutions of the problem (population) will become better and better. And we achieve the goal of optimize the solutions to the problem. One of the advantages of evolutionary algorithm is that it is easy to realize and not restricted to particular optimization problem. In this paper we will use evolutionary algorithm as a substitutional method to train our neural network. And try to find out the reason of the performance limit of our model.

After we trained the model, we also want to optimize its structure. Many formats of data we daily meet are of big size especially pictures and videos. However, what we want from these pictures and videos usually takes a small percentage of the file, which means there are many redundancies in the files [3]. For example, most of the video's content is coherent, which means the adjacent frames in the video do not change much. Then if the video just stores information for every frame, there could be a huge waste of data space. Reducing the redundancies is beneficial but this can be difficult by traditional algorithms. One implemented strategy in [2] is to let a neural network to learn how to store the information of a picture more efficiently, using same input and output to train the network. The result in the hidden layer can be a substitutional expression of the original data. Finally, using technique of modifying the hidden layer, we can get the wanted size of the picture stored in a compression form. The techniques of compression will be used in this paper's following part.

2 Method

The basic outline of the technique implemented on the dataset in this paper can be divided into three main parts. The first part is to construct a neural network to make predictions of the anger observed by perceivers is genuine or posed by physiological signals which is pupillary responses. The network is implemented with three layers of neurons, and basic techniques including back-propagation and crossEntropy loss function. The second part is using evolutionary algorithm

as a substitutional method to optimize the model. And the third part is building methods according to [2] [5]'s implementation of neural network reduction and image compression to prune to network built in part one.

2.1 Classification of Genuine or Posed Anger

Classification Problem Setup

The data in Anger dataset provided by [1], according to the description of the dataset, has 400 inputs and label with 9 columns and the first two dimensions are experiment number and video number, following by 6 dimensions of pupillary responses and one label (genuine or posed). Thus, the classification problem decided is to predict the label of genuine or posed with 6-dimension data.

Data Inspection

Before we start to design a specific method to accomplish the classification task, we need to have a detailed inspection of the dataset given by [1]. In order to figure out the relationship between humans' unconscious physiological characteristics and the anger they saw is genuine or posed. The experiment where the data comes from asks 20 volunteers to watch 20 videos. For the 20 videos, there are 10 videos with genuine anger and 10 videos with posed anger. All the videos come from YouTube and the experiment designer made the videos of genuine anger from documentaries and news, while the posed anger videos are made from movies. All other factors that may potentially influence the volunteers' judgement are removed as much as possible, for example, the format resolution of the video, and the actors in the posed anger movies are not famous so that the possibility of volunteers knowing the actor and perceiving the acted anger in the movie is reduced. The nine columns of the dataset contain the following information.

O: the id of volunteer who observes the videos of posed or genuine video. 20 observers correspond O1-O20.
Video: id of videos been watched by volunteers, with posed of genuine anger. 20 videos correspond T1-T10, F1-F20.
Mean: the mean of observer's pupillary response.
Std: the standard deviation of the pupillary response.
Diff1: the change of observer's left pupil size
Diff2: the change of observer's right pupil size
PCAd1 and PCAd2: orthogonal transformations
Label: indicates the video is genuine or posed.

Firstly, we will not use the first two dimensions of the data as input of classifier, because they are directly connected with the label result. Then, we can find some clues in the density plots for different input dimensions of data with Genuine and Posed labels. In figure 2.1.1, all the six input dimensions that would be given to the classifier are showed with their density plots. In this figure we can learn that regarding different dimension of the data, how much the difference is between the data with different label. And the figure shows most of the six dimensions (except PCAd1) seem to have similar density plots of Genuine and Posed label, which means they may contribute less to the classification. To solve this kind of classification problem, there are some common choices including k-nearest neighbor (KNN), support vector machine (SVM) and neural network (NN) according to [4]. And for this dataset, the difference showed in PCAd1 may not support the KNN and SVM have a good performance to classify the data. [4] also mentioned that the final accuracy result on its smile dataset with neural network is the best. Thus we decided to build a neural network for this anger classification problem.

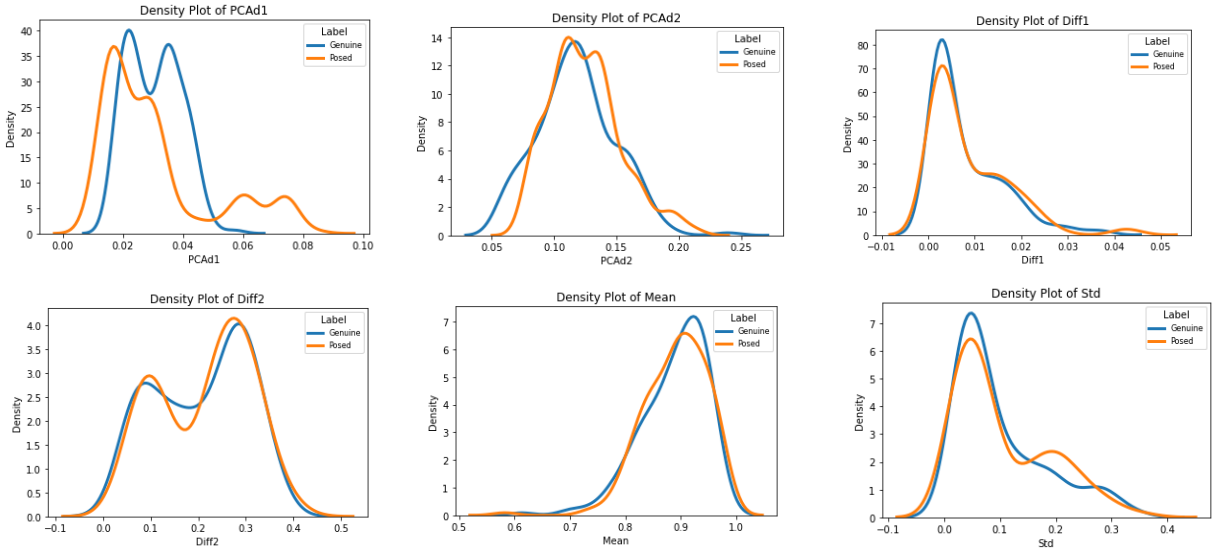

Figure2.1.1 Density plots for different input dimensions of Genuine and Posed label

Data preprocessing

The 6-dimension input patten in the dataset needs preprocessing. As we can see in Table 1, the average of Mean, is much larger than other dimensions. We are also aware of that, from the Data Inspection part, the values of Mean do not contribute much to the classifier. Thus it is important to conduct data preprocessing. Or else the learning process will be significantly slowed down because Mean dimension can have big weights when the training is in early phase.

Table 1. The average and standard deviation of six dimensions of input data

Dimension	Average	Standard Deviation
Mean	0.88909015	0.04603393
Std	0.10246244	0.06934124
Diff1	0.00842139	0.0065422
Diff2	0.20957463	0.08669128
PCAd1	0.03070341	0.01101771
PCAd2	0.12138183	0.0235699

Suitable data preprocessing can improve the performance of data-driven models [6]. We conduct data preprocessing with (1). For every column, every value in the column is subtracted with their mean and divided by the variance. And finally, we change the text labels to integers for calculating. The genuine and posed labels are changed to 0 and 1 accordingly.

$$x = \frac{x - x_{mean}}{\sigma_x^2} \tag{1}$$

One of the benefits of data preprocessing is speeding up the training process. On the Anger data set, with same neural network model, optimizer, and loss function, it takes around 20000 epochs to converge without data preprocessing. While it only takes 5000 epochs to converge after above preprocessing. One interesting thing is that although the learning rate is same, for the model with data without preprocessing, the reduction of the loss of the training set very little in the first 5000 epochs. When after 5000 epochs of training, the reduction comes to a relative normal speed. This implies the model may consume much time in adapting to the unbalanced data.

Basic Neural Network Model

Except for input neurons, the network built has 3 layers of neurons with 6, 6 and 1 neurons like showed in Figure 2.1.2. At first the number of hidden neurons was set to 10, and it was reduced because of the following tests of performance.

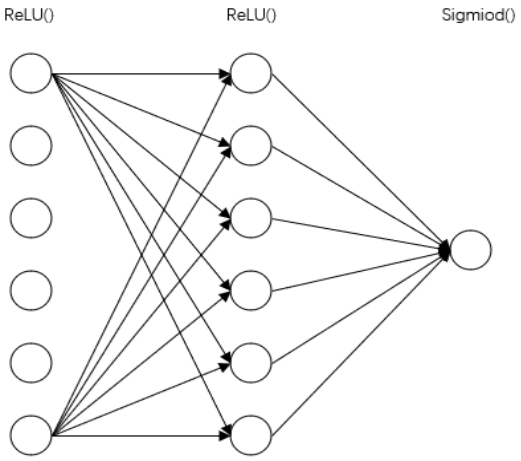

Figure2.1.2 network structure (some weights not drawn)

The hidden layer 1 and hidden layer 2 use activation function ReLU (), and the last layer has activation function Sigmoid () for classification purpose. The loss function is Binary Cross Entropy which is fit for binary classification problems along with Sigmoid (). The optimize method is SGD.

2.2 Evolutionary Algorithm on optimizing neural network

Except using normal back-propagation method to train the neural network, we also implement an evolutionary algorithm to optimize the network. The basic idea of using EA is the traditional method gives us a model that do not reach a very high test accuracy. We want to find that the performance problem is whether attributed to the learning being constrained to a local minimum, or the best performance of neural network model on this dataset have been reached. Because [10] shows that evolutionary algorithm, as a general method, can have a good effect on multiple-minimum problems. Thus we decided to implement EA.

Although evolutionary algorithm cannot be used on a classification problem directly, for a neural network which has a fully-known structure, we can generally regard the collection of all weights and bias in the network as an individual, which means every individual in the population corresponds to one neural network with that same structure. And we can conduct evolutionary algorithm to optimize the individual (neural network).

Coding of Individuals

The target neural network, as mentioned above, has 4 layers with input dimension 6, two 6 units hidden layer and a 1 unit output layer. To fully store weight and bias in the network, individual needs to have a length of (2):

$$\begin{aligned} L = {} & Units_{input} * Units_{hidden1} + Units_{hidden1} \\ & + Units_{hidden1} * Units_{hidden2} + Units_{hidden2} \\ & + Units_{hidden2} * Units_{output} + Units_{output} \end{aligned} \tag{2}$$

Because the network is fully connected, so there are 6 * 6 weights between the input layer and first hidden layer, 6 * 6 weights between second and third layer, 6 * 1 weights between third and last layer. And we have 6 + 6 + 1 biases to store. The length of individual code should be 91. Individuals should have a form of (3):

$$X_i^t = [x_{i1}^t, \cdots, x_{ik}^t, \cdots, x_{in}^t] \quad i = 1, 2, \cdots, N \tag{3}$$

where,

x_{ik}^t = real numbers of weights or bias
t = generation of the individual
i = number of individuals
n = number of genes, which is 91
N = number of populations

Evaluation function (decoding of individuals)

To evaluate an individual in the population, we extract the first 36 real numbers from the individual and cast them to the type and shape of first weight matrix in the neural network, then assign their values to the matrix. Similarly, number 37-42 genes are responsible for first part of biases, number 43-78 genes are responsible for second part of weights, number 79-84 are responsible for second part of biases, number 85-90 genes are responsible for third part of weights and number 91 gene is responsible for the last bias.

After assigns the values in the individual to the network parameter, we have the new network with weights and biases optimized by the evolutionary algorithm. Then we use the method $evaluate\_accuracy$ which is used in the back propagation method to get the accuracy on the whole dataset and take it as the fitness of this individual. From here we can also know that we need to maximize the fitness function during the process of evolutionary algorithm.

Population Initialization

Every individual is initialized by a random generator that generates 91 real numbers with standard normal distribution. Then we generate 100 individuals as the initial population.

Selection

We use tournament selection to select the individuals in the population, tournsize is set to 3. So every time we randomly pick up three individuals in the population, and choose the individual with best fitness to the pool of making offspring. The procedure will repeat until needed number of individuals have been selected.

Crossover

The method used for crossover is blend crossover, the main steps of this method can be summarized as:

1. Select two parents X^t, Y^t
2. For gene i, calculate $d_i = |x_i^t - y_i^t|$
3. In interval $[min(x_i^t, y_i^t) - \alpha d_i, \ max(x_i^t, y_i^t) + \alpha d_i]$ randomly choose μ_i
4. μ_i becomes the value of child gene, $x_i^{t+1} = \mu_i$

α is a parameter, and for any $\alpha > 0$, blend crossover will expand the search space. In the implementation, we set $\alpha = 0.5$.

Mutation

For every gene (real number) in the mutated individual, there will be an independent probability that this gene will change to a random number selected according to the Gaussian distribution with mean value unchanged. The variance of the Gaussian distribution is set to 0.1. And the independent probability is 0.4.

General Settings

For the evolutionary algorithm, we initialize the population as 100 individuals. Crossover probability is 0.8. Mutation probability is 0.4, and we set the algorithm will run for 800 generations.

2.3 Compression on the Hidden Layer

After implementing a neural network classifying genuine and posed anger and using evolutionary algorithm to optimize it, we want to address the problem that if the 6, 6, 1 layer structure of the network have some redundancies. So we conduct following analysis and techniques to reduce the hidden layer units of the trained neural network.

Distinctiveness analysis

This analysis is towards the activation matrix, which is the output matrix of a particular layer of neurons when all training input enters the model. Every column in the matrix is a vector which corresponds to a neuron. Thus, we can analyze the distinctiveness between units according to these vectors. The steps implementing this analysis are showed below:

1. Calculating the activation matrix for the hidden units in the layer to be analyzed, which is the second hidden layer. The activation matrix is given by:

$$A_{hidden1} = max(0, input * M_{weight1} + M_{bias1}) \quad (4)$$

$$A_{hidden2} = max(0, A_{hidden1} * M_{weight2} + M_{bias2}) \quad (5)$$

where,

$A_{hidden1}$ is the activation matrix of the first hidden layer

$M_{weight1}$, M_{bias1}, $M_{weight2}$, M_{bias2} are the weights and biases matrices of the first and second layer.

$A_{hidden2}$ is the activation matrix of the second hidden layer, the target matrix.

2. We know that every column in the activation matrix corresponds to one hidden unit's output. So by analyzing the similarities between these columns we can get the distinctiveness of hidden units. We use cosine similarity to measure the distinctiveness.

$$cos\_similarity = \frac{V_i \cdot V_j}{||V_i|| \cdot ||V_j||} \quad (6)$$

where V_i and V_j is the $i\,th$ and $j\,th$ column vector of the activation matrix.

3. For every two hidden units in the second layer, analyze their similarities. Then record the most similar two units' index i and j

Pruning the Network
After finding the two similar hidden units, we decide to delete the first unit, and add its weights and bias to the second unit. The steps implementing the pruning are showed below:

1. Get the indexes of two most similar hidden units i and j.
2. Add the weight of hidden unit i to hidden unit j by add the corresponding rows in the weight matrix.
3. Delete the $i\,th$ row of weight matrix.
4. Add the bias of hidden unit i to hidden unit j by add the corresponding element in the bias vector.
5. Delete the $i\,th$ element of bias vector.
6. We also need to change the weight matrix of output layer similarly, but no need for output layer bias.

Every time when we prune the network, we delete one of the most similar two hidden units in the second hidden layer. Then we fine tune the network with 1000 epochs of back propagation training.

3 Result and Discussion

3.1 Neural Network and its basic Optimization

At first the model was trained with not preprocessed data. After training many times with different learning rate and finally at 0.05 and momentum 0.8, the best test accuracy is no higher than 55% as showed in Figure 3.1.1.

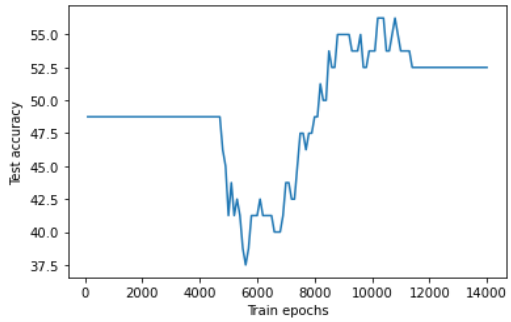

Figure3.1.1 Test accuracy with data not preprocessed

First useful optimization is to change the activation function from ReLU() to Tanh(). After using Tanh() as the first two layer's activation function, test accuracy can reach around 58% as Figure 3.1.2 showes. Also, the speed of learning is much higher, which is mentioned in the preprocessing part.

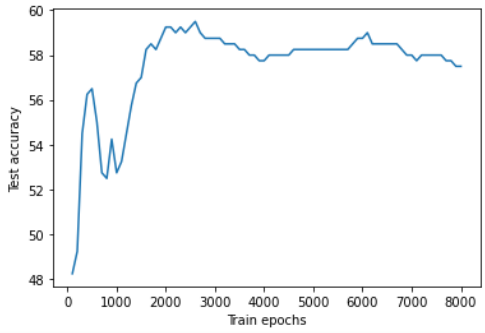

Figure3.1.2 Test accuracy with data not preprocessed and Tanh ()

Then the data is preprocessed, and after using different activation function, ReLU() retrive the position of best function. With data preprocessed and the ReLU () function and learning rate 0.001, the model can achieve best test accuracy around 75% (Figure 3.1.3). However, Tanh () does not have a good performance.

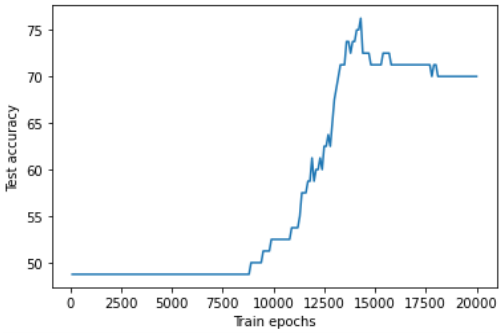

Figure3.1.3 Test accuracy with data preprocessed and ReLU ()

After preprocessing the data, one of the most important factors of the model is learning rate, and the final progress is made by raising the learning rate. When learning rate is higher than around 0.01(0.017 in the Figure 2.1.6) and with a momentum 0.9. The best test accuracy can reach 85%, which means that the higher learning rate allows the model to get over some saddle points constraint the accuracy to 75% (Figure 3.1.4).

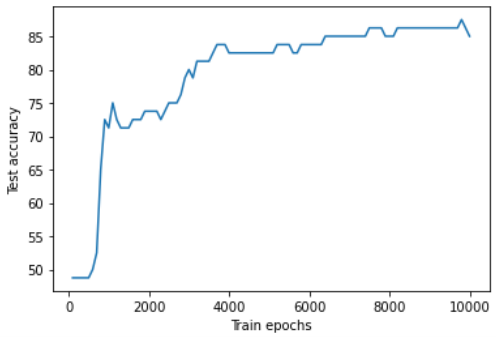

Figure3.1.4 Test accuracy with data preprocessed, ReLU () and higher learning rate

3.2 Evolutionary Algorithm for Optimization

The result of evolutionary algorithm design in the Method section for optimizing the neural network is showed in Figure 3.2.1. Because we randomly initialize all weights and biases of the network, at first, the mean accuracy of individuals is 50%. Then with the progress of evolution, the best accuracy raises to around 86%. For this optimizing method, we can use the best individual in the last generation of population as the final weights and biases for our neural network. So we can conclude that evolutionary algorithm has trained the model to an accuracy of 86%.

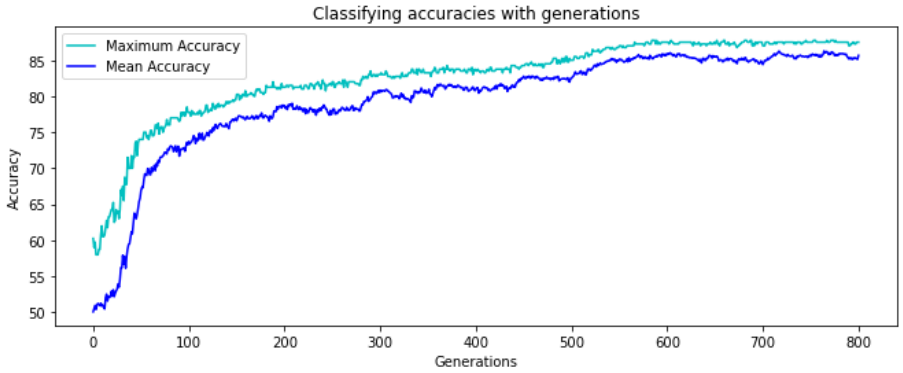

Figure3.2.1 Accuracy of the neural network model optimized by EA

3.3 Neural Network Pruning

After implementing pruning the hidden layers, we want to find the influence of pruning hidden units on the performance of the model. In Figure 3.3.1, there are 5 subfigures to show the pruning process. In the first subfigure, the accuracy is given by the unpruned neural network after 10000 epochs of back propagation training. Then every time we prune one hidden layer unit in the network and train another 1000 epochs. Subfigure 2-5 shows the results of accuracy of network after pruning 1, 2, 3, and 4 hidden units.

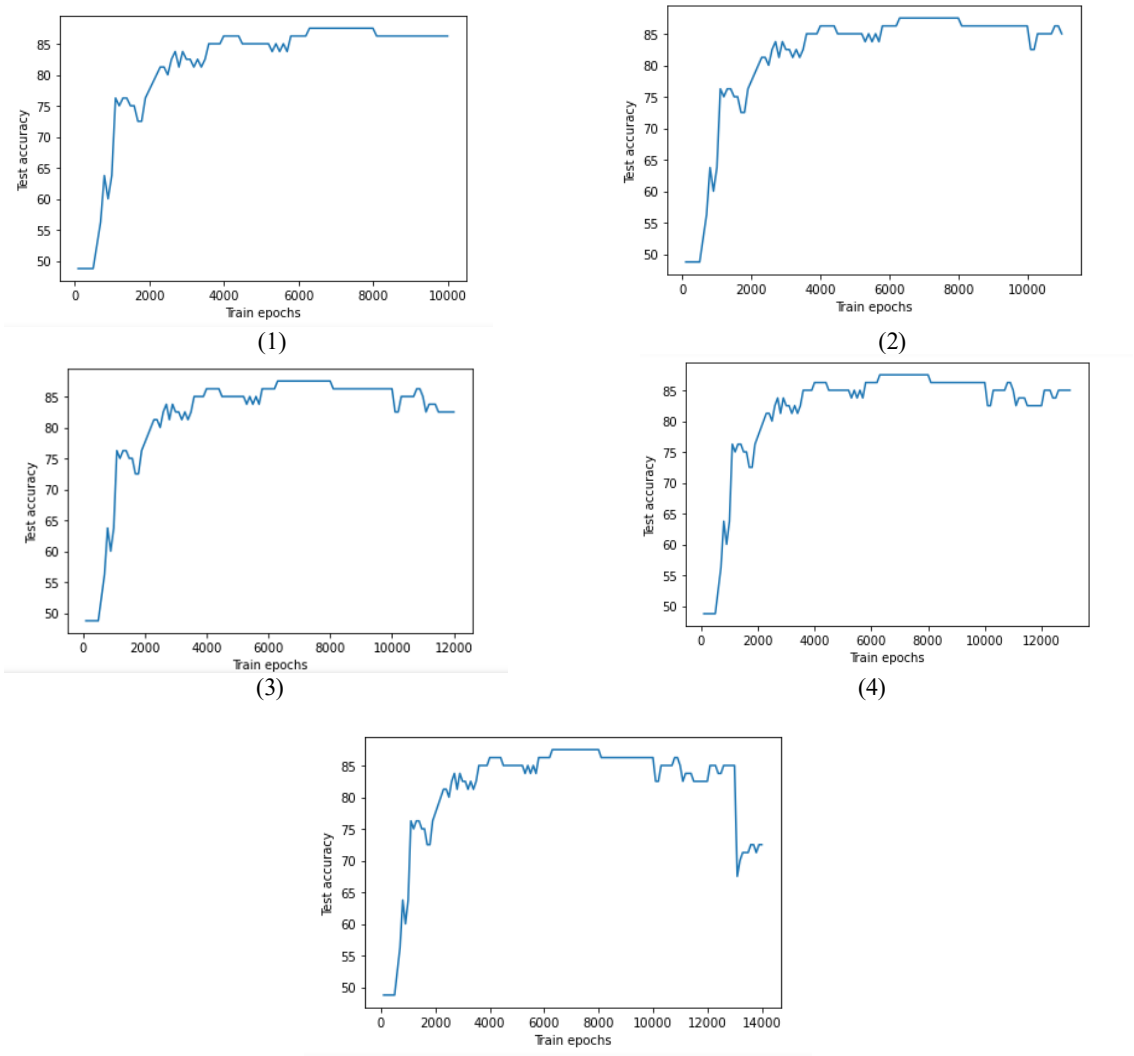

Figure3.3.1 Accuracies with the pruning process on hidden layer

According to the result of pruning, although the network with 6 hidden units can learn well, there are still space to optimize to model (some hidden units are redundant). When reduce the number of hidden units to 3, the performance of the model remains around 90% of its peak. When delete the fourth unit, the performance declined significantly. And we can draw a conclusion that this model may need at least 3 hidden units in the hidden layers to learning from data. This analysis is stick to part of the conclusion in [5] that when we want to compress the image with a big scaler, the quality will decline because the units remained in the hidden layer are not enough to be able to store the majority information of the image.

3 Comparison and Limitations

Firstly, we can compare the results of back propagation training method and evolutionary algorithm on the neural network. After tuning many hyper parameters of the back propagation method, the model after 10000 epochs of training can reach an accuracy of 85%. And with 800 generations of evolution, evolutionary algorithm achieves an accuracy of 86%. These two similar accuracies can address the problem we mentioned in the introduction part. Can the 85% accuracy of back propagation method due to being constrained by some local minimum? We cannot conclude with complete certainty,

but the answer can be no because evolutionary algorithm usually has a good performance on finding the global best solution. And its best accuracy is still around 85%.

Then we compared our model with the method implemented in [1], their model can reach an accuracy of 99.6% on the Anger dataset. While model in this paper can only reach around 85%. One of the big differences between the two models is the node number in the hidden layer, in [4]'s implementation, they achieve around 95% accuracy with about 13 hidden nodes. However, due to the lack of experience, the maximum number of hidden units I ever set is 10, which can be a factor of the performance gap. Another difference is the training function, Levenberg-Marquardt training function [7] was used in the implementation, the adjustable step size may contribute to the better result. On the other hand, the model developed in this paper after tuning the hyper-parameter and hidden layer compression, still has a relatively good performance with simple structure. This can be the only advantage of it.

4 Conclusion and Future Work

In conclusion, implementing a simple structured neural network for classification the anger perceivers saw is either genuine or posed is achievable. And with some reasonable adjustment during testing the network, we can get a model with around 86% or higher accuracy of classification. This proved that perceiver's pupillary response patterns can reflect the anger they saw is genuine or posed even the precievers are unconscious.

For future work, after using NN to building relationship between facial emotion and perceiver's physiological signals, speech emotion detection neural network has also been build up [8]. We may design and optimize a network that use people's speech emotion to predict whether their facial emotion is genuine, which could be interesting. And we can also implement advanced network pruning technique including using thresholds and structured filter level pruning [9].

References

1. 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.
2. Gedeon, TD, Harris, D, "Network Reduction Techniques," Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34, 1991.
3. Stefan Siersdorfer, Jose San Pedro, and Mark Sanderson. 2009. Automatic video tagging using content redundancy. In Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval (SIGIR '09). Association for Computing Machinery, New York, NY, USA, 395–402. DOI:https://doi.org/10.1145/1571941.15720103.
4. Md Zakir Hossain and Tom Gedeon. 2017. Classifying Posed and Real Smile from Observers' Peripheral Physiology. 11th International Conference Pervasive Computing Technologies for Healthcare (PervasiveHealth '17), EAI Conference Series, Barcelona, Spain, 1-4. Retrieved from http://bit.ly/2gkllCZ
5. T. D. Gedeon and D. Harris, "Progressive image compression," [Proceedings 1992] IJCNN International Joint Conference on Neural Networks, 1992, pp. 403-407 vol.4, doi: 10.1109/IJCNN.1992.227311.
6. S. Tang, S. Yuan and Y. Zhu, "Data Preprocessing Techniques in Convolutional Neural Network Based on Fault Diagnosis Towards Rotating Machinery," in *IEEE Access*, vol. 8, pp. 149487-149496, 2020, doi: 10.1109/ACCESS.2020.3012182.
7. Moré J.J. (1978) The Levenberg-Marquardt algorithm: Implementation and theory. In: Watson G.A. (eds) Numerical Analysis. Lecture Notes in Mathematics, vol 630. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0067700
8. Han, Kun / Yu, Dong / Tashev, Ivan (2014): "Speech emotion recognition using deep neural network and extreme learning machine", In INTERSPEECH-2014, 223-227.
9. Jian-Hao Luo, Jianxin Wu, Weiyao Lin; ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression, Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 5058-5066
10. Korolev, L.N. Evolutionary computations and neuronet and genetic algorithms — formal statements. J Math Sci 168, 80–88 (2010). https://doi-org.virtual.anu.edu.au/10.1007/s10958-010-9976-z