Assignment2: Classification on EEG information: Neural Network & Genetic Algorithm

Songjiang Xie[1]

[1] Research School of Computer Science, Australian National University
[2] u7102674@anu.adu.au

Abstract. EEG is closely related to the state of human mind. About the former experiment[1], we have know that which parameters in the data have bigger effect on humans mind. However, the whole neural network output is not always relatively high when executing. One of the main reason is that the original parameter is not very well at the first time. It may cause the whole neural network into a situation where gradient disappear or gradient boom happens.
This paper uses genetic algorithm in order to find the most suitable parameters for neural network training. It also want to know whether this method can improve this network.
The experiment shows that genetic algorithm can not improve this neural network very well. After analysis, maybe the GA running time is too few for improving the result or the dataset is too simple for this question. The results of this study can improve the efficiency of the related equipment which analysis people's psychological status. The result also can help to take the most reasonable solution to the symptoms.

Keywords: EEG · neural network · genetic algorithm(GA) · mood classification

1 Introduction

Emotions can often affect a person's behavior and mental state. Having a good mood can make people solve the problems around them better. Some studies have found that brain waves have an inseparable relationships with people's mental state[2], but it is not possible to accurately find out which aspect of EEG affects people's mental state. After the former research[1], we have found that which side can make a people more possible to produce the calm or stressful mood. But the result of neural network is not stable. We want to try genetic algorithm as the improvement to modify the original parameters to check whether the whole network can get a stable answers when running the code. We want to analysis the result of with and without GA. This study improves the efficiency of emotion recognition, and gives a common method of improving neural network.

2 The Data preparation and feature engineering

We randomly took EEG data from 24 participants. The data covered 15 channels of EEG of 24 participants. Each channels came from different locations in the brain and was named AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8

We get 15 different dimensions of data, including basic information and special information.

1. Basic information: average value (mean), maximum value (max), minimum value (min), standard deviation (std), variance (var), the difference between the third quartile and the first quartile (iqr), skewness (skw)

2. Other special information can be seen in paper [2]

3 The Method

3.1 Neural Network

The neural network we use is full connected layer.

Experiment
The optimizer function is fixed on Adam. About activation function, we will use relu function in our paper, their formula are defined below:

$$relu : f(z) = max(0, x) \tag{1}$$

3.2 Genetic Algorithm

The genetic algorithm always used to find the maximum or minimum value of the nonlinear multivariate equations, we can see the parameters in the function is solution, the input dataset is the coefficient of this function. So the target is that we need find a parameters, so that the training loss is the minimum.

Definition
The genetic algorithm imitate the natural selection and biological evolution on the dataset to find the most suitable solution. The whole algorithm can be divided into five part:

1) Initialization: Assign Random values for the function as the original population.

2) Evaluate: Calculate the output of the function as this population performance which called fitness value. ($fitness\_func$ function in the code)

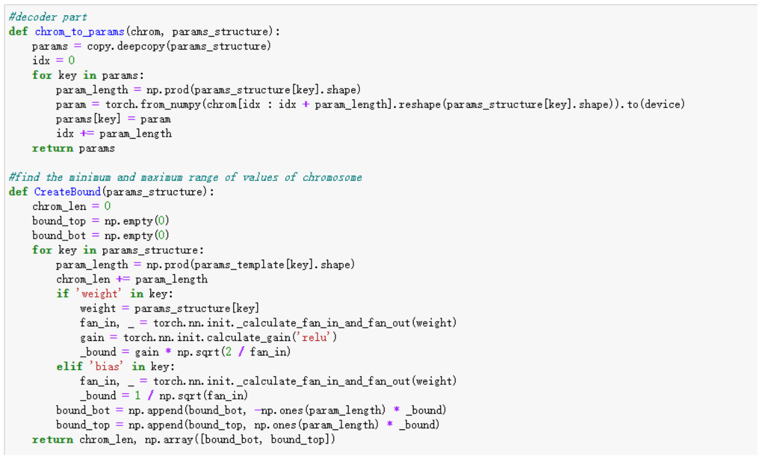
3) Selection: Choose some population in all possible solutions based on fitness value as the next generations. ($select$ function in the code)

4) Crossover: Swap or combine some values of solutions between some generations to create new solutions. ($cross$ function in the code)

5) Mutation: Change some values of solutions to create new solutions. ($mutate$ function in the code)
After too many generations, we can get the closet solution of the answers.

Experiment
In order to finish the whole code, the code idea comes from the github[3]. Next is the detail of the idea in the code:

1) Initialization:



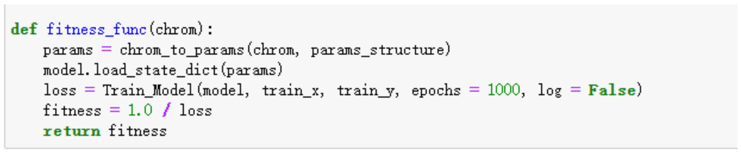
```python
#decoder part
def chrom_to_params(chrom, params_structure):
    params = copy.deepcopy(params_structure)
    idx = 0
    for key in params:
        param_length = np.prod(params_structure[key].shape)
        param = torch.from_numpy(chrom[idx : idx + param_length].reshape(params_structure[key].shape)).to(device)
        params[key] = param
        idx += param_length
    return params

#find the minimum and maximum range of values of chromosome
def CreateBound(params_structure):
    chrom_len = 0
    bound_top = np.empty(0)
    bound_bot = np.empty(0)
    for key in params_structure:
        param_length = np.prod(params_template[key].shape)
        chrom_len += param_length
        if 'weight' in key:
            weight = params_structure[key]
            fan_in, _ = torch.nn.init._calculate_fan_in_and_fan_out(weight)
            gain = torch.nn.init.calculate_gain('relu')
            _bound = gain * np.sqrt(2 / fan_in)
        elif 'bias' in key:
            fan_in, _ = torch.nn.init._calculate_fan_in_and_fan_out(weight)
            _bound = 1 / np.sqrt(fan_in)
        bound_bot = np.append(bound_bot, -np.ones(param_length) * _bound)
        bound_top = np.append(bound_top, np.ones(param_length) * _bound)
    return chrom_len, np.array([bound_bot, bound_top])
```

Using deep copy to get the unchanged structure record of the neural network
The Create Bound function can get the range of every parameters, the idea comes from initialization of normal distribution of parameters with normal strategy. The parameters function range $(U, -U)$ is shown below:

$$U = gain \times \sqrt{\frac{2}{fan_{in} + fan_{out}}} \tag{2}$$

Gain is the activation function gain. It is decided for the function itself. For Relu function, the gain is $\sqrt{2}$
fan_{in} and fan_{out} is the structure for every layer input node and output node.

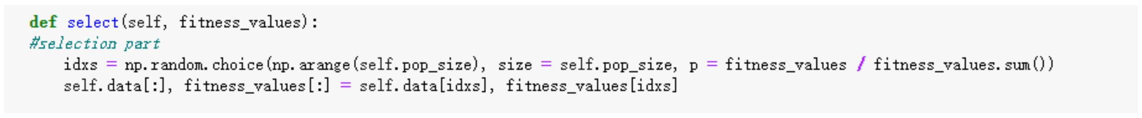
2) Evaluate:



```python
def fitness_func(chrom):
    params = chrom_to_params(chrom, params_structure)
    model.load_state_dict(params)
    loss = Train_Model(model, train_x, train_y, epochs = 1000, log = False)
    fitness = 1.0 / loss
    return fitness
```

Because we want to find the minimum value of loss. So we let $\frac{1}{loss}$ as the fitness value, when loss value is bigger, the fitness value is smaller. It is means that this have worse performance.

3) Selection:

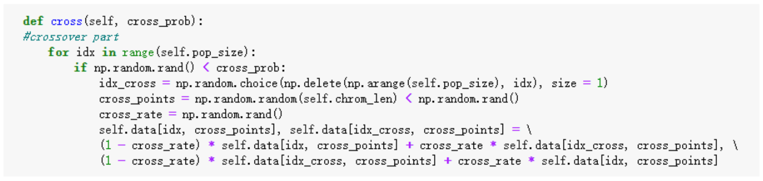


```python
def select(self, fitness_values):
#selection part
    idxs = np.random.choice(np.arange(self.pop_size), size = self.pop_size, p = fitness_values / fitness_values.sum())
    self.data[:], fitness_values[:] = self.data[idxs], fitness_values[idxs]
```

We use roulette wheel selection as our selection method. When the performance for one solution is better, it is more likely to be choose as the next generation.

Function *np.random.choice* can have repeat value in the sequence, so the value which is not exist in the sequence means they are dropped for the next generation.

4) Crossover:



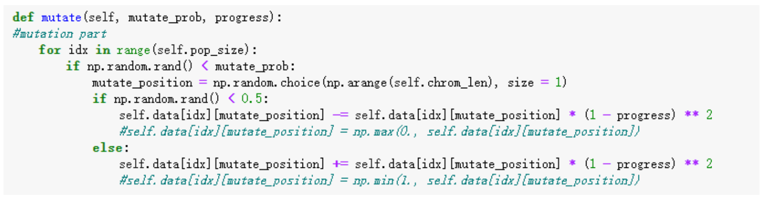
```python
def cross(self, cross_prob):
#crossover part
    for idx in range(self.pop_size):
        if np.random.rand() < cross_prob:
            idx_cross = np.random.choice(np.delete(np.arange(self.pop_size), idx), size = 1)
            cross_points = np.random.random(self.chrom_len) < np.random.rand()
            cross_rate = np.random.rand()
            self.data[idx, cross_points], self.data[idx_cross, cross_points] = \
            (1 - cross_rate) * self.data[idx, cross_points] + cross_rate * self.data[idx_cross, cross_points], \
            (1 - cross_rate) * self.data[idx_cross, cross_points] + cross_rate * self.data[idx, cross_points]
```

The crossover idea is multiple cross over idea, we choose a lot points as the crossover point and use the method below to generate the new solution:

$$\begin{aligned} a_{kj} &= a_{kj} \times (1-b) + a_{lj} \times b \\ a_{lj} &= a_{lj} \times (1-b) + a_{kj} \times b \end{aligned} \tag{3}$$

The function above means crossover gene k and gene l at j position, b is the random value in $[0, 1]$

5) Mutation:



```python
def mutate(self, mutate_prob, progress):
#mutation part
    for idx in range(self.pop_size):
        if np.random.rand() < mutate_prob:
            mutate_position = np.random.choice(np.arange(self.chrom_len), size = 1)
            if np.random.rand() < 0.5:
                self.data[idx][mutate_position] -= self.data[idx][mutate_position] * (1 - progress) ** 2
                #self.data[idx][mutate_position] = np.max(0., self.data[idx][mutate_position])
            else:
                self.data[idx][mutate_position] += self.data[idx][mutate_position] * (1 - progress) ** 2
                #self.data[idx][mutate_position] = np.min(1., self.data[idx][mutate_position])
```

The mutation idea is shown below:

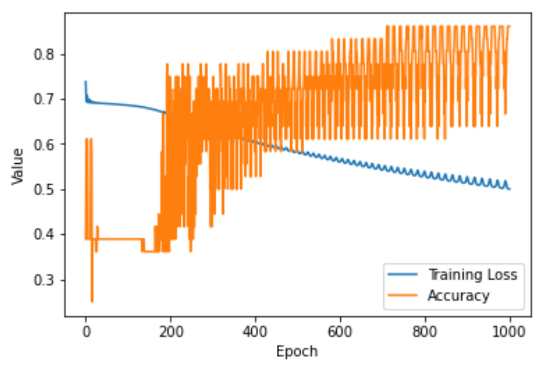
$$
\begin{aligned}
a_{kj} &= \begin{cases} a_{kj} - f(g) * a_{kj} & 0 \le r < 0.5 \\ a_{kj} + f(g) * a_{kj} & 0.5 \le r \le 1 \end{cases} \\
f(g) &= (1 - \frac{e}{epochs})^2
\end{aligned}
\tag{4}
$$

e is the time of the genetic algorithm, epochs is the whole times of the genetic algorithm, r is the random value in $[0, 1]$

4 Result & Discussion

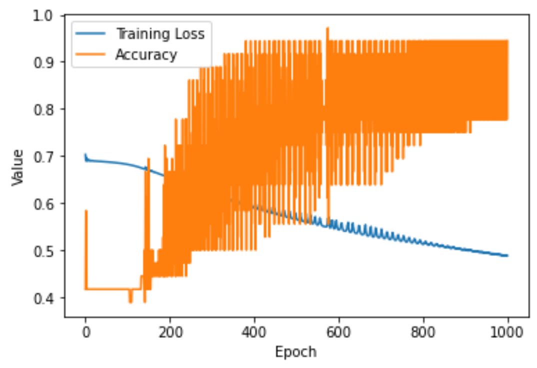
In this paper, we try many times about running the code with and without GA. The training loss and accuracy are shown in image below:

```
0.8611111
```



Information about training without GA

```
0.9444444
```



Information about training with GA

We can find that they both have very high value in accuracy. The minimum accuracy without GA is less than that is with GA. The beginning training loss with GA is lower than that is without GA.

After genetic algorithm, the neural network is faster close to the final value of training, but GA consume a huge mount of time about find the best parameter.

Because the whole data are only $144 * 15$, which is very small for training, maybe it is the one reason why they both have high accuracy.

The other reason may be the limit of the computer, so that the whole code only have 50 population and 15 generations in my code. Maybe more generations can have better result.

5 Conclusion

In this paper, we propose a neural network training model with and without genetic algorithm based on the difference of EEG information between calm and nervous state. We find that the performance of two algorithm have no clear difference. The problem maybe caused by dataset or computation limit.

6 Future Work

However, there is still a lot of work to be improved further. We can analysis more complex dataset to find whether the result of algorithm with GA is better. Such as thermal-eeg_v2 dataset. We can also use fuzzy logic to create a fuzzy neural network, compare the performance with other algorithm.

When these aspects can have deeper research, it is very likely to find a one-to-one relationship between the human brain and emotion. We can also have a deeper understanding of people's health, which may lead to more abundant solutions to people's psychological problems.

7 Reference

[1] Songjiang Xie. Classification on EEG information Neural Network & Decision Tree, pp.1-4.

[2] Rahman, J.S., Gedeon, T., Caldwell, S., Jones, R. and Jin, Z., 2021. Towards Effective Music Therapy for Mental Health Care Using Machine Learning Tools: Human Affective Reasoning and Music Genres. Journal of Artificial Intelligence and Soft Computing Research, 11(1), pp.5-20.

[3] https://github.com/cattidea/bp-ga-pytorch.

[4] Milne, L. K., Gedeon, T. D., & Skidmore, A. K. (1995). Classifying Dry Sclerophyll Forest from Augmented Satellite Data: Comparing Neural Network, Decision Tree & Maximum Likelihood. training, 109(81), 0.

[5] Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., & Brown, S. D. (2004). An introduction to decision tree modeling. Journal of Chemometrics: A Journal of the Chemometrics Society, 18(6), 275-285.

[6] Bromiley, P. A., Thacker, N. A., & Bouhova-Thacker, E. (2004). Shannon entropy, Renyi entropy, and information. Statistics and Inf. Series (2004-004).

[7] Jayalakshmi, T., & Santhakumaran, A. (2011). Statistical normalization and back propagation for classification. International Journal of Computer Theory and Engineering, 3(1), 1793-8201.

[8] Ma, W., & Lu, J. (2017). An equivalence of fully connected layer and convolutional layer. arXiv preprint arXiv:1712.01252.

[9] Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.

[10] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.