COMP41390 Connectionist Computing Project Report

Li Weijing
Student No. 19204246

May 15, 2020

1 MLP Model

In this section, I will introduce the implement details of four functions in MLP model, as well as explain how to use them in the experiment.

1.1 Randomize

This function is used to initialize the weights and weight changes of layers of MLP model. The weights is set to random values between 0 and 1 at the size $(ni \times nh)$ by using numpy.random.uniform, where ni denotes the number of inputs and nh denotes the number of hidden units. The dw(weight changes) array is set to zeros.

1.2 Forward

The forward pass of MLP follows the function:

$$h = g(w \cdot x) \tag{1}$$

h denotes the output of the layer, g indicates the activation function, w is the weight while x is the input of the layer. In this model, it has two different activation functions sigmoid and tanh:

$$sigmoid : g(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

$$tanh : g(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

The function has two input parameters, the input vector and the activation function defaulted as sigmoid.

1.3 Backwards

The backward pass process outputs the weight changes of layers, the equations follows

$$\delta(k) = (y - \hat{y}) \cdot g^{'}(h_k) \tag{4}$$

$$dw_k = \Delta w_k = h_k^T \delta(k) \tag{5}$$

where k denotes the index of layer, y and y_hat indicates the true and predict value of the output individually, as well as $g^{'}$ is the derivative of the activation function.

$$derivative\_of\_sigmoid : g^{'}(x) = \frac{e^{-x}}{(1 + e^{-x})^2} \tag{6}$$

$$derivative\_of\_tanh : g^{'}(x) = 1 - \frac{(e^{x} - e^{-x})^2}{(e^{x} + e^{-x})^2} \tag{7}$$

The input of the function is the input train vectors, target vector and activation function defaulted as derivative of sigmoid. Other than the dw(weight changes) of the layers, the function returns the mean square error of the output.

1.4 UpdateWeights

The function is responsible for the weights update as,

$$w_k = w_k + \eta dw_k \tag{8}$$

where η denotes the learning rate.
This function takes learning rate as an input parameter. After change the w(weights) of the layers, the function also set zeros to dw(weight changes).

1.5 Application method

Firstly, the model need be called, as well as initialized by,

```
model = MLP(ni,nh,no)
model.randomize()
```

In the experiment, this step only needs to be done once. After setting the number of epochs, we can start training by,

```
for i in range(0,epochs):
    model.forward(X_train,activate_func)
    error = model.backwards(X_train,y_train,activate_func)
    model.updateWeights(learning_rate)
```

where X_train is input of the dataset, and y_train is the target of it. What's more, in the testing process, we only need to call forward function, and achieve the predict value by calling *model.o*.

2 Experiment

2.1 XOR function

In this experiment, we only have four input values, so we use the same dataset for training and testing. According to the property of XOR function, the learning rate is set to 1 and the activation function is set to sigmoid.

Input	True value	Predict value
0,0	0	0.01589546
0,1	1	0.9844056
1,0	1	0.98456783
1,1	0	0.01481049

After running 10000 epochs with 4 hidden layers, the predict value is close to the true value. Fig.1 shows the Error vs, epochs, it is clearly that the error decrease rapidly before

approximately 750 epochs, then slowly converge to the true value. The train and test error is 0.0002.

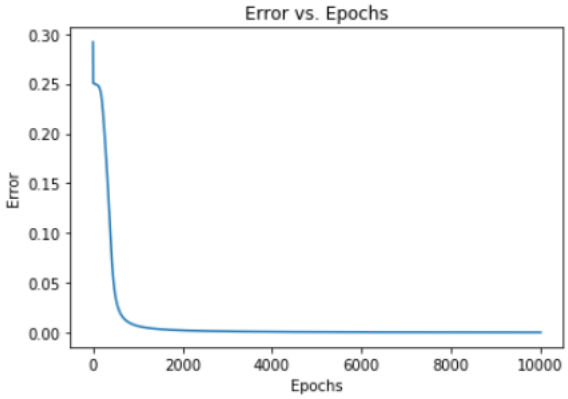

Figure 1: XOR function error vs. epochs

2.2 sin function

The dataset of sin function is created by sin(x1-x2+x3-x4), where x1,x2,x3,x4 is random values between (-1,1), the learning rate is 0.001, and the activation function is tanh.

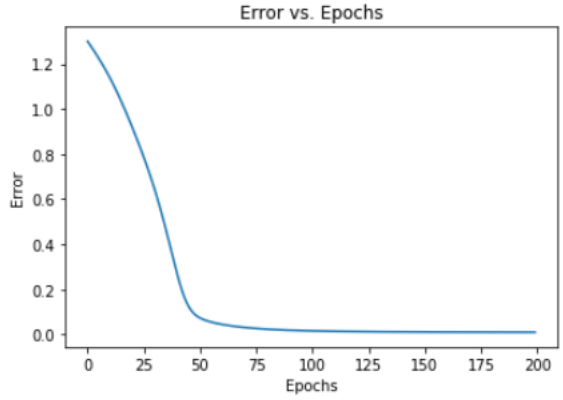

Figure 2: Sin function error vs. epochs

The sin function performs better on the model than XOR, it converges quicker than XOR function, so I chose epochs 200, and achieve the train error 0.014 and test error 0.027.

2.3 letter recognition

By using "letter-recognition.data", we train a model with 1000 epochs, 15 hidden layers and tried two different activation mode.

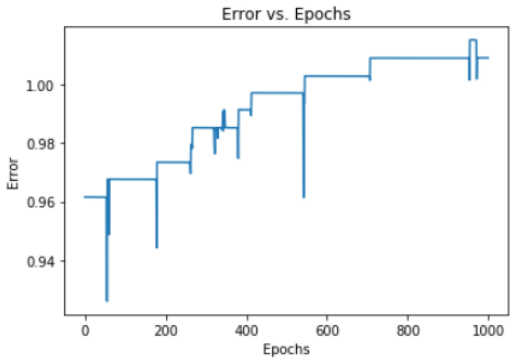

Figure 3: Activation function: tanh

Firstly, I tried tanh as the activation function, but error keeps increase, so I chose sin instead. To make the trend more obvious, I enlarged 1-200 epochs part of the figure.

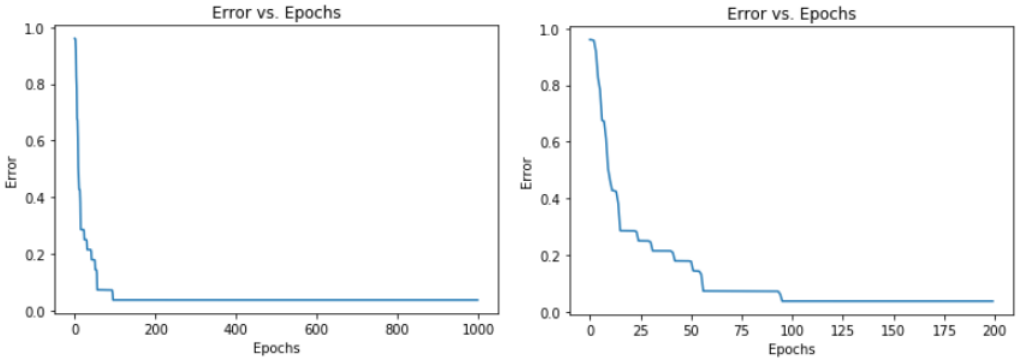

Figure 4: Activation function: sin

It is clearly that the curve is less smooth than the previous two models, because it is a 26-category classification problem.

3 Conclusion

This project is mainly about the basic structure and implementation of MLP model. It gave us a more clear understanding of details of the model. Actually many courses have already used the model many times, but the only thing we need to do is called tensorflow, and it would help us to easily built neural network models. This project give us a chance to think about the specific method of implement models, and remind us of the most basic formulas.

It is a special experience that we can use a model built by ourselves, and test the performance of it on different functions like xor, sin even letter recognition. I have to say that the accuracy of the model is better than I imaged.

Besides, The mathematical derivation included in the course is very important and is not be involved in other courses, so I think this is a great course with an excellent project.