Deep Learning HW2

1st 楊淳先
Institute of Data Science
National Cheng Kung University
Tainan, Taiwan
re61110404@gs.ncku.edu.tw

***Abstract*—Under the previous mention of only using numpy, a two-layer neural network model and LeNet5 model were implemented, and LeNet5 was improved and compared with the original model.**

***Index Terms*—Fully Connect Layer, Backpropagation, LeNet5**

I. Introduction

This assignment needs to use non-CNN-related packages such as numpy and SKLearn to implement a two-layer neural network model and LeNet-5 without using mainstream neural network packages such as Tensorflow and PyTorch.The model should calculate backward parameters, in order to get it's gradient . In the part of LeNet5, you can refer to the github URL provided by the teacher and add your own annotations.All code can be found at https://github.com/OhhhYaaa/-Deep-Learning-HW2https://github.com/OhhhYaaa/-Deep-Learning-HW2.

II. Implementation

A. Computational Graph

The first question is to implement a two-layer fully connected layer, which needs to include an input layer, a hidden layer, and an output layer. As shown in the figure 1, that is to say, assuming that X is the input value, $\hat{\mathbf{y}}$ is the predicted value, W_1 and W_2 are respectively in the two layers The linear transformation matrix that needs to go through, b_1 and b_2 are the bias that need to be added in the two layers, $Sigmoid$ and $Softmax$ is the activation function used in the two-layer fully connected layer I made.

$$\hat{y} = Softmax(W_2(Sigmoid(W_1X + b_1)) + b_2) \quad (1)$$

In addition, during the training, the efficiency will be very poor if the gradient is updated once only one sample is viewed at a time. I hope to update multiple samples at a time, so I hope to add dataloader to help me get batch samples in each training , but since the dataloader provided in python is in PyTorch, it cannot be used, so I wrote a dataloader myself to help me iterate the samples. In addition, when calculating the loss, I use CrossEntropy for calculation. He needs to make the input matrix dimensions the same to perform the calculation, so

Identify applicable funding agency here. If none, delete this.

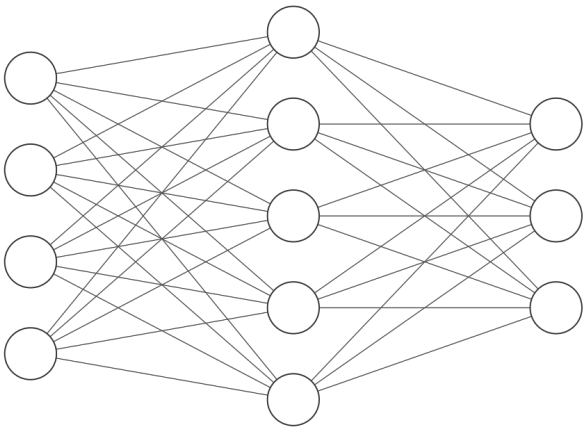


Fig. 1. TwoLayerNN

I also wrote a onehot function to transform the label into onehot encoding. Special mention here that I designed a activation function by myself, trying to imporve the predict accuracy.The function are shown in equation 2.

$$0.5 \cdot (1 + \tanh(0.5 \cdot x)) \quad (2)$$

The followings are the step that I calculate the gradient.First , we set some variable to represent the temporary storage used in the middle.

$$\begin{aligned} z_1 &= W_1 * X + b_1 \\ a_1 &= Softmax(z_1) \\ z_2 &= W_2 * a_1 + b_2 \\ a_2 &= Softmax(z_2) \end{aligned} \quad (3)$$

And now let's start the backpropagation of our model:

$$\begin{aligned} \mathrm{d}y &= \hat{\mathbf{y}} - label \\ \mathrm{d}w_2 &= \mathrm{d}a_1 * \mathrm{d}y \\ \mathrm{d}b_2 &= \mathrm{d}y \\ \mathrm{d}s &= \mathrm{d}w_2 * \mathrm{d}y * a_1 * (1 - a_1) \\ \mathrm{d}w_1 &= X * \mathrm{d}s \\ \mathrm{d}b_1 &= \mathrm{d}s \end{aligned} \quad (4)$$

B. LeNet-5

The second question is to let us implement LeNet-5. LeNet-5 is a model proposed by Professor Yann LeCun in

the paper Gradient-based learning applied to document recognition in 1998. It is the first convolution successfully applied to digital recognition problems.On the MNIST dataset, the LeNet-5 model can achieve an accuracy rate of about 99.2%. First, let's briefly introduce the LeNet-5 model, the model has seven layer :

1) C1 : Convolutional Layer :
   - input size : 32*32*1
   - kernel size: 5
   - depth : 6
   - stride : 1
   - padding : 0
   - output size : 28*28*6
2) S2 : Pooling Layer
   - input size : 28*28*6
   - output size : 14*14*6
3) C3 : Convolutional Layer
   - input size : 14*14*6
   - kernel size: 5
   - depth : 16
   - stride : 1
   - padding : 0
   - output size : 10*10*16
4) S4 : Pooling Layer
   - input size : 10*10*16
   - output size : 5*5*16
5) FC5 : Fully Connected Layer
   - input size : 5*5*16
   - output size : 1*120
6) FC6 : Fully Connected Layer
   - input size : 1*120
   - output size : 1*84
7) FC7 : Fully Connected Layer
   - input size : 1*84
   - output size : 1*10

For MNIST dataset, All pictures are after gray scale, so the input channel would be 1, but for our dataset, the input channel would be 3. And for the final output of the model, because there are 50 different labels in our dataset, instead of 10 for MNIST.So we need to modify the model to let the model train on our data.

First, we have to change the input size of C1, which should be changed from 32*32*1 to 32*32*3, and for the final output size, it should become 1*50. The following are the changes in the dimention of the input data. $32*32*3 \rightarrow 28*28*6 \rightarrow 14*14*6 \rightarrow 16*10*10 \rightarrow 16*5*5 \rightarrow 1*120 \rightarrow 1*84 \rightarrow 1*50$

By the way, the activation function in the Source code is ReLU, But in the paper, it used Sigmoid function.So, I used Sigmoid instead of ReLU in my implementation.But I find out that the backward function in Sigmoid function the source code gave is wrong.it says that the backpropagation is

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= x*(1-x) \end{aligned} \tag{5}$$

But after my calculation, I think it's backward should be

$$\begin{aligned} f(x) &= \frac{1}{1+e^{-x}} \\ f'(x) &= \frac{1}{1+e^{-x}} * (1 - \frac{1}{1+e^{-x}}) \end{aligned} \tag{6}$$

C. Improved LeNet-5

In Improved LeNet-5 model, first we have to change activation function to below

$$f(x) = x * \frac{1}{1+e^{-x}} \tag{7}$$

And the backpropagation should be

$$f'(x) = \frac{1}{1+e^{-x}} + x * \frac{1}{1+e^{-x}} * (1 - \frac{1}{1+e^{x}}) \tag{8}$$

And if we let $\sigma(x) = x * \frac{1}{1+e^{-x}}$, then we can simplified above as

$$f'(x) = \sigma(x) * f(x) * \sigma(x) \tag{9}$$

Second, we have to change the kernel size of convolutional layer from 5 to 3, and add one more convolutional layer to the model.But because it would take too much for training on 256*256 figure, I reshaped the data into 32*32.Because the figure is too small, so I take away one of the pooling layer away. So the model structure would be like

$$\begin{aligned} C1 &: 32*32*3 \rightarrow 30*30*6 \\ S2 &: 30*30*6 \rightarrow 15*15*6 \\ C3 &: 15*15*6 \rightarrow 13*13*16 \\ padding &: 13*13*16 \rightarrow 14*14*16 \\ S4 &: 14*14*16 \rightarrow 7*7*16 \\ padding &: 7*7*16 \rightarrow 8*8*16 \\ C5 &: 8*8*16 \rightarrow 6*6*6 \\ FC6 &: 6*6*6 \rightarrow 1*120 \\ FC7 &: 1*120 \rightarrow 1*84 \\ FC8 &: 1*84 \rightarrow 1*50 \end{aligned} \tag{10}$$

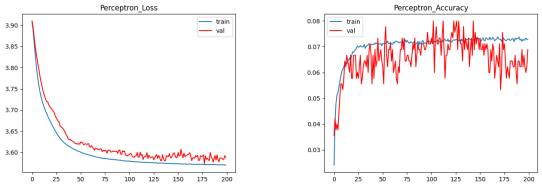
Originally, for LeNet5 and Improved LeNet5, it takes more than 2 hours to train one epoch, So I changed the forward and backward calculation of Conv and MaxPool, trying to accelarate the speed of training.Finally, it only take sixteen and twenty two minutes for one epoch, seperately. And I discovered that the backward calculation of Conv in the source code didn't record the gradient of W and B so I plus the gradient in order to improve the result.

III. RESULTS

A. Computational Graph

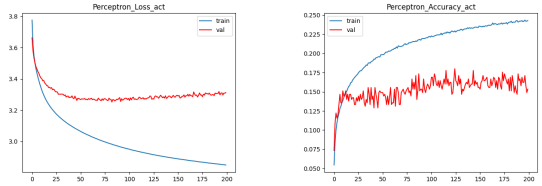
Below are the results of comparing my two layer perceptron with one layer perceptron in SKLearn.For SKLearn perceptron, I trained for 2000 epochs, but for my own two layer perceptron , I only trained for 200 epochs. And figure 2 represents the loss and accuracy while training my own two layer perceptron.3 represents the loss and accuracy while train two layer perceptron with my own activation function. According to the result,the curve have become gentle , so I think it can converege well and give not bad prediction.

	Train	Val	Test
Mine With Sigmoid	0.074	0.0689	0.0644
Mine With Own Act	0.2487	0.1533	0.1622
SKLearn	0.0501	0.0511	0.0444



(a) perceptron_loss (b) perceptron_acc

Figure 2. Perceptron_Sigmoid



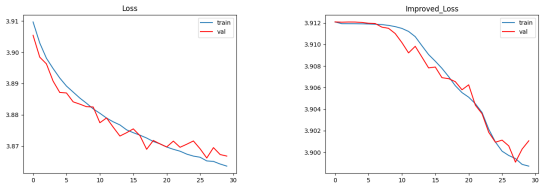
(a) perceptron_loss_act (b) perceptron_acc_act

Figure 3. Perceptron_Self_Act

B. LeNet-5

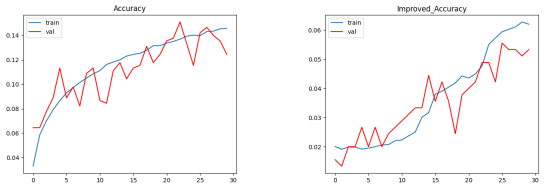
Below are the table comparing the original LeNet5 and Improved LeNet5.By the way I trainall of the model for 30 epochs.Figure 5(a) and 4(a) represent the accuracy and loss of original LeNet5 seperately.Figure 5(b) and 4(b) represent the accuracy and loss of Improved LeNet5 seperately.And for figure 6(a) and 7(a) represents the loss and accuracy of adding gradient of W and B.

	Val	Test
LeNet5	0.1244	0.1178
LeNet5grad	0.1644	0.1689
Improved	0.0689	0.0711
Improvedgrad	0.1644	0.1689



(a) lenet5_loss (b) Improved_lenet5_loss

Figure 4. LeNet5_Loss

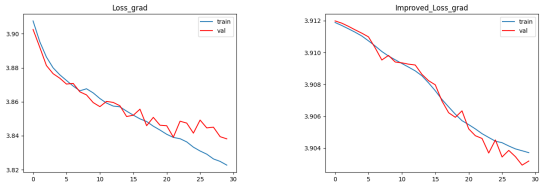


(a) lenet5_acc (b) Improved_lenet5_acc

Figure 5. LeNet5_acc

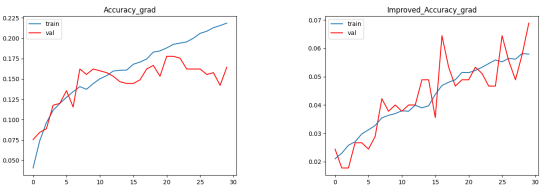
IV. CONCLUSION

It shows that my own activation would help the two layer perceptron performed better, And for LeNet5, after adding gradient of the model, the loss of the model can drop down faster. In my implementation, two layer perceptron performs the best , it can reach accuracy 24.87%, and the worst is Improved LeNet5, which can only get 6.9%, I guess the reason that Improved LeNet5 performs more worse than LeNet5 is because I didn;t set the depth of convolution layer well, which result in bad performance.



(a) lenet5_loss (b) Improved_lenet5_loss

Figure 6. LeNet5_Loss_grad



(a) lenet5_acc (b) Improved_lenet5_acc

Figure 7. LeNet5_acc_grad

References

[1] https://github.com/toxtli/lenet-5-mnist-from-scratch-numpy
[2] https://chih-sheng-huang821.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92-weight-initialization%E5%92%8Cbatch-normalization-f264c4be37f5
[3] https://betterprogramming.pub/how-to-build-2-layer-neural-network-from-scratch-in-python-4dd44a13ebba
[4] https://blog.csdn.net/Sophia_11/article/details/85043685