MLP Report

Name: Xiangyun Ding (丁相允)

Student ID: 2016011361

Overview

In this homework, I implemented multilayer perceptron with the original code structure. All codes are in directory `codes`.

In addition to the normal parts to complete, I split Softmax and CrossEntropyLoss and wrote an extra Softmax layer. I will explain it in the following part.

Details

Changes to files:

- `layers.py`: Add details to class Relu, Sigmoid and Linear. Add class Softmax layer.
- `loss.py`: Add details to EuclideanLoss. Change SoftmaxCrossEntropyLoss to CrossEntropyLoss.
- `utils.py`: Add functions to record run time.
- `run_mlp.py`: Change the network to test more frequently. Add codes to draw loss-acc figures.

Since the Euclidean loss function is: $E_k(n) = \frac{1}{2}(t_k(n) - y_k(n))^2$, and the $t_k(n)$ we have is either 0 or 1, it should be a good idea to normalize all $y_k(n)$ to $(0, 1)$. I noticed that the Softmax function happens to do this normalization, so I decided to add a Softmax layer before the EuclideanLoss or CrossEntropyLoss layer.

The SoftmaxCrossEntropyLoss function has a gradient with a simple format. Since I split out the Softmax layer, we have to re-calculate its gradient. In the forward propagation, given input $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, the output of Softmax layer is $\mathbf{y} = [y_1, y_2, \ldots, y_n]$, where we have:

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

In the backward propagation, we've known $\frac{\partial E}{\partial y_i}$, and we want to calculate $\frac{\partial E}{\partial x_i}$. So we should use the chain rule:

$$\frac{\partial E}{\partial x_i} = \sum_{j=1}^n \frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial x_i}$$

If $j \neq i$, then we have:

$$\frac{\partial y_j}{\partial x_i} = \frac{\partial \frac{e^{x_j}}{\sum_{j=1}^n e^{x_j}}}{\partial x_i} = e^{x_i} \times -\frac{e^{x^j}}{(\sum_{j=1}^n e^{x_j})^2} = -y_i \times y_j$$

If $j = i$, then we have:

$$\frac{\partial y_j}{\partial x_i} = \frac{\partial \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}}{\partial x_i} = \frac{(\sum_{j=1}^n e^{x_j}) \times e^{x_i} - e^{x_i} \times e^{x_i}}{(\sum_{j=1}^n e^{x_j})^2} = y_i \times (1 - y_i)$$

So we can simply use the result of $j \neq i$, then add y_i to the answer. Finally we have:

$$\frac{\partial E}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial E}{\partial y_j} \times -y_i \times y_j + \frac{\partial E}{\partial y_i} \times y_i$$

This is the forward and backward propagation equation for Softmax layer.

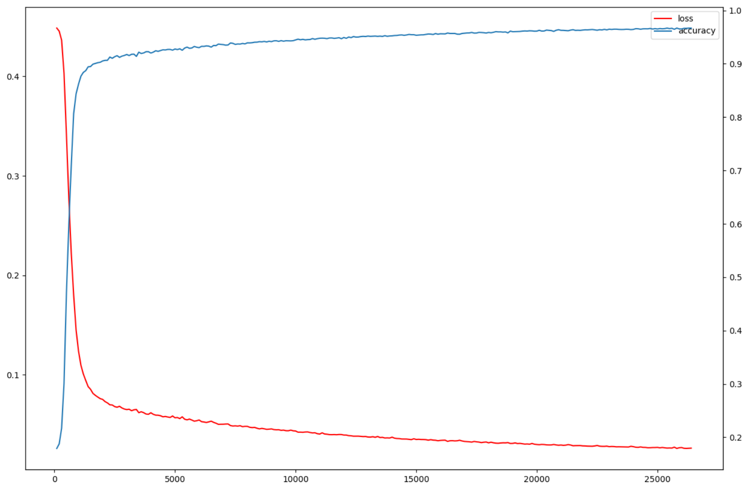
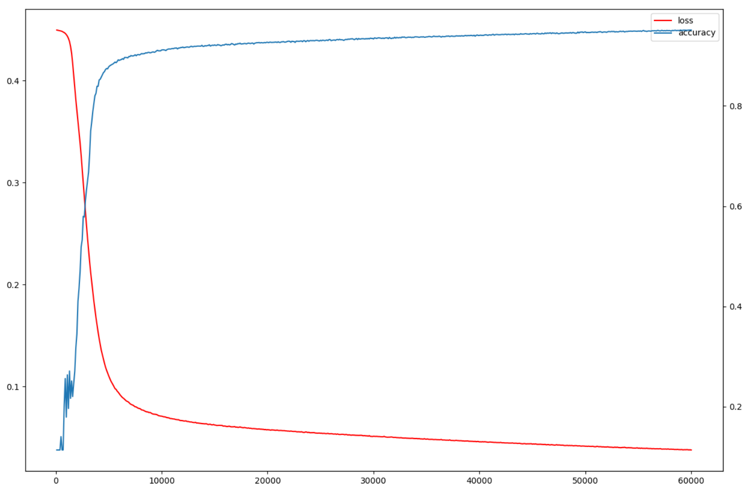
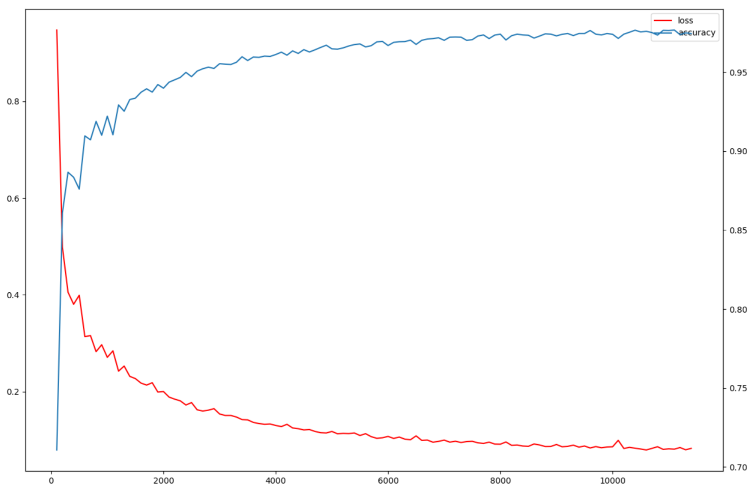
Experiments

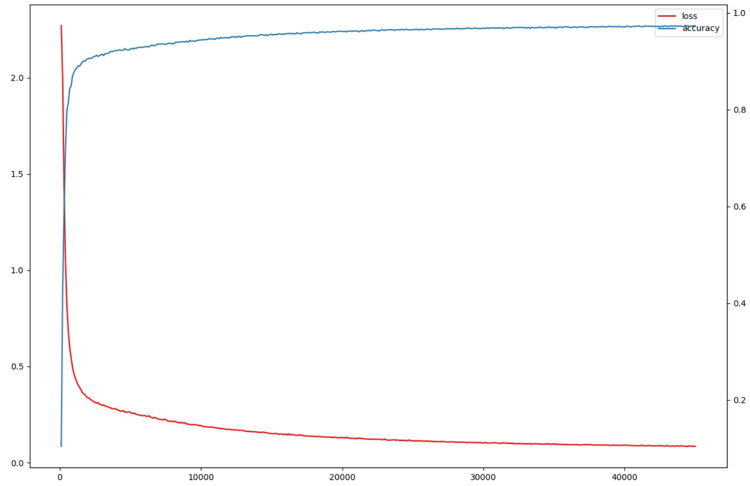
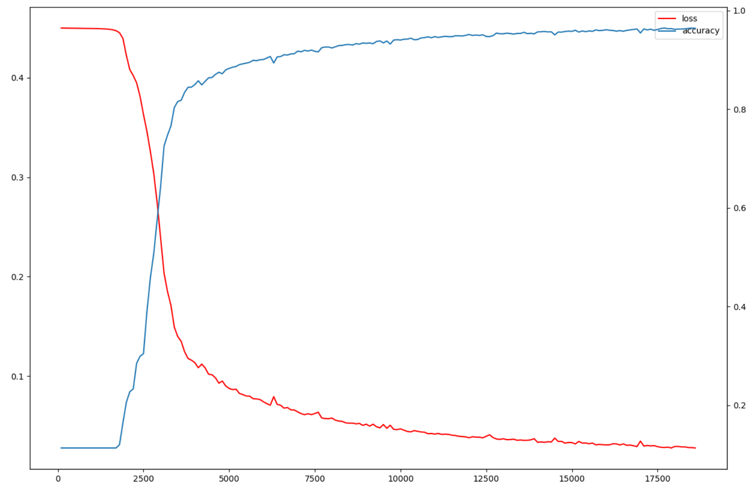
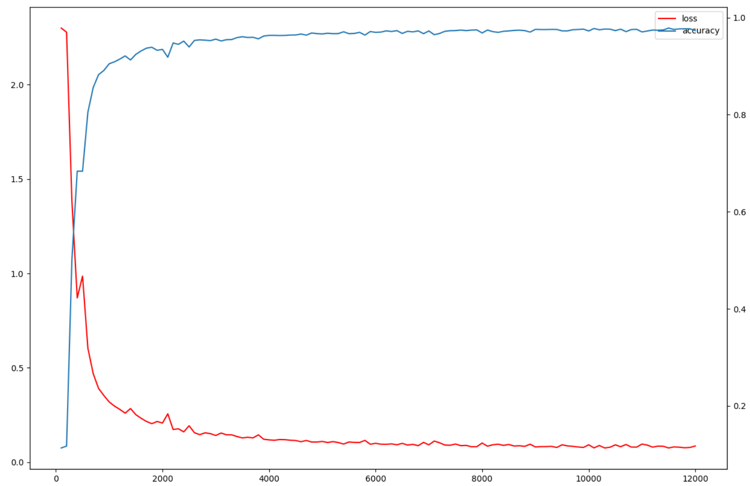
Basic Results:

ID	Network Structure	Loss Function	Learning Rate	Train Time(s)	Best Accuracy(%)
0	Linear(784×128)->Relu->Linear(128×10)->Softmax	EuclideanLoss	0.1	179	96.7
1	Linear(784×128)->Sigmoid->Linear(128×10)->Softmax	EuclideanLoss	0.1	431	95.1
2	Linear(784×128)->Relu->Linear(128×10)->Softmax	CrossEntropyLoss	0.1	84.3	97.6
3	Linear(784×128)->Sigmoid->Linear(128×10)->Softmax	CrossEntropyLoss	0.1	326	97.4
4	Linear(784×128)->Relu->Linear(128×128)->Relu->Linear(128×10)->Softmax	EuclideanLoss	0.1	162	96.5
5	Linear(784×128)->Relu->Linear(128×128)->Relu->Linear(128×10)->Softmax	CrossEntropyLoss	0.1	101	98.1

Loss-Accuracy Figures:

ID	Figure

ID	Figure
0	
1	
2	

ID	Figure
3	
4	
5	

For higher resolution images, please visit directory `codes/images`.