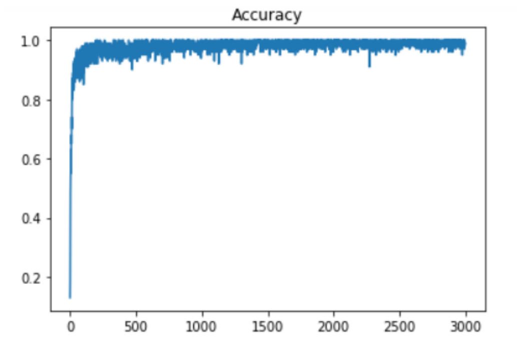
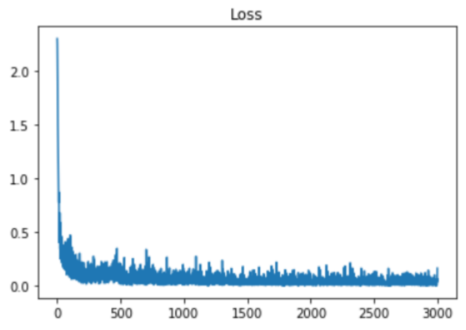
MNIST Classification with PyTorch

By Fedor Ivachev 费杰 2019280373

Training Accuracy and Loss:





Testing accuracy: 0.9916

Model structure:

layer1	Convolution(1,32,5,1,2)	ReLU()	Max Pooling (2, 2)
layer2	Convolution(32,64,5,1,2)	ReLU()	Max Pooling (2, 2)
Dropout			
Fully Connected Layer 1 (7 * 7 * 64, 1000), ReLU () 1			
Fully Connected Layer 2 (1000, 100), ReLU () 2			
Fully Connected Layer 3 (100, 10)			

Number of epochs = 5

Learning rate = 0.001

Loss criterion = CrossEntropyLoss() (No SoftMax for the last layer because CrossEntropyLoss combines it with cross entropy function)

Optimizer = Adam()

Using this structure and parameters the best result has been achieved.

Compare it with other modifications:

Modification	Testing accuracy
No normalizing of the data	0.9730
Fully Connected Layer 1 (7 * 7 * 64, 1000), ReLU Fully Connected Layer 2 (1000, 10), ReLU (no middle FC)	0.9915
Change kernel_size to 6	0.9823
Learning rate = 0.01	0.9903

Observations:

- Normalizing the data is very important. The model trains better when data is between 0 and 1.
- Adding another fully connected layer increased the accuracy, but for such a small value that it can't be considered as a major improvement.
- Changing the size of windows on each of the convolution layers to a bigger size decreased the accuracy. I guess it is probably connected to the thickness of lines on the pictures.
- Using ReLU activation can save us from vanishing gradient, when number of layers is high.
- Increasing learning rate decreased the accuracy, although not drammaticaly.
- The number of epochs may be increased to achieve a better accuracy.