Homework 1 Report
Zhaoheng Zheng
12430279
EECS-542

Problem Statement

Implement layers of Convolutional Neural Network and train a network on MNIST dataset.

Part One

In this part, layers required by this assignment were implemented and their outputs are verified.

Additionally, in order to accelerate the training process, momentum was introduced. It is fairly an excellent feature for training. It considerably speed up the convergence and improve the performance. To be more specific momentum can keep the network converging in an approximately correct direction and adjust it slightly with the gradient descent from batches. It is useful for training, especially for the SGD training policy. This policy samples a batch in one flow and update the parameters. A small batch can hardly reflect the whole dataset. Thus, a gradient descent from batch shouldnt influence the model greatly.

Part Two

In this part, we trained a Convolutional Neural Network on MNIST dataset. And we conducted experiments upon leaky-ReLU and batch normalization.

For baseline model, we visited the page included in the requirement file, where we found that LeNet-5 with no distortions can reach a desirable error rate at 0.95%. The network architecture is shown below.

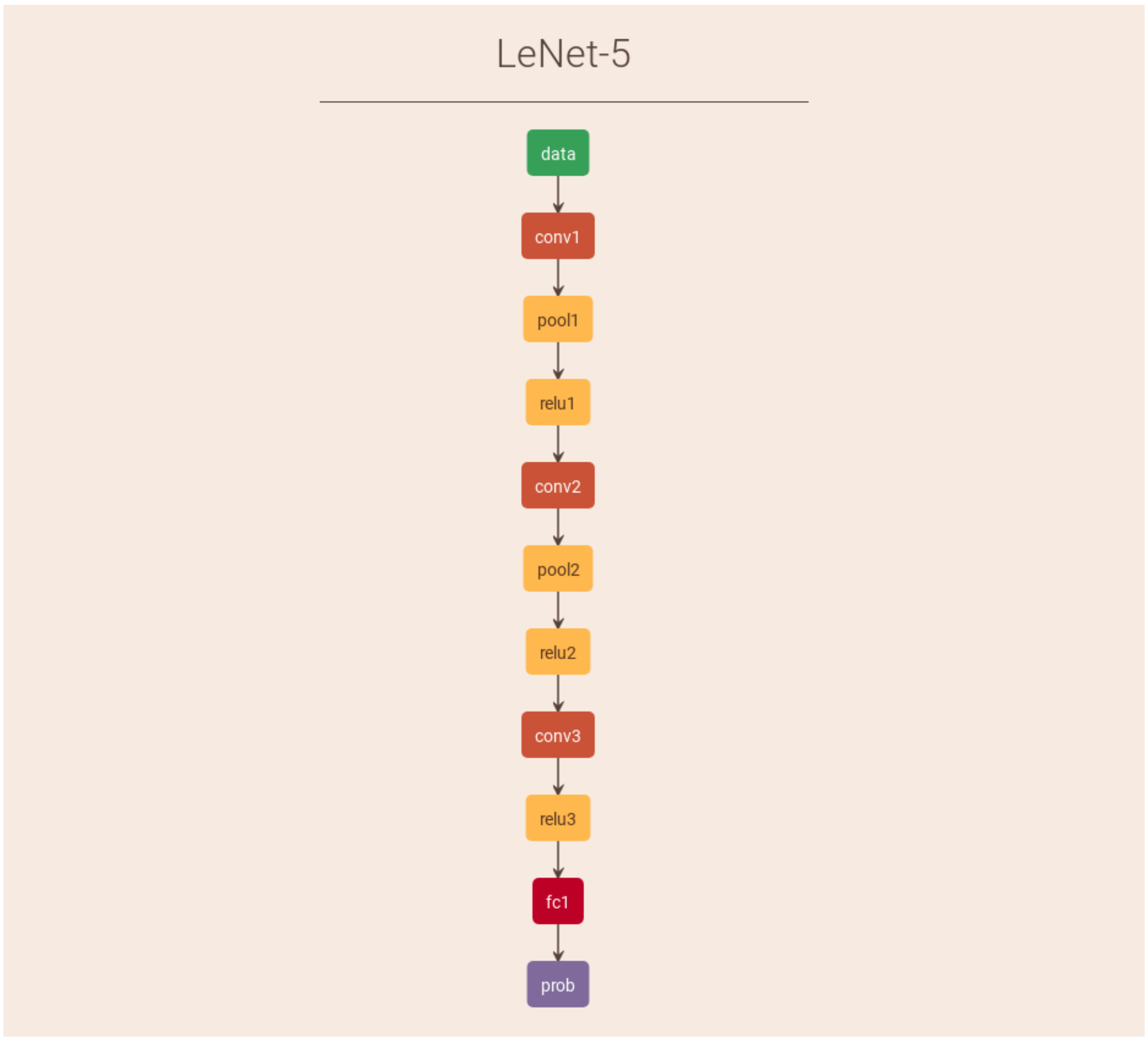

Figure 1: LeNet-5 model for baseline

In our experiment, the network shown above reached a error rate at 1.46%, which is far more lower than the 4% error rate that assignment instruction required. The performance convinced us to take it as base line module.

To be more specific, the kernel size of the first two convolution layers is 5. The 5*5 kernel can summarize more local information from input and produces rich feature map. Even it might be better to adopt two convolution layers with 3*3 kernel rather than a single convolution layer with a kernel of 5, the later one is faster given that we conduct our experiments upon CPUs rather than GPUs. The last convolution layer has a 3*3 kernel because the size of input image is only 4*4. Furthermore, according to LeCun's paper, the first convolution layer produces a feature map with 6 channels, while the second contains 16 and the last feature map comes with 120 channels.

As for hyper-parameters, we conducted our experiment at the learning rate at 0.03, with 0.0005 weight decay and 1000 images per batch. We ever chose a batch size of 100, but it takes too long to run an epoch. So we fed 1000 images to the network per batch and obtained desirable performance. In addition, we initially set momentum rate as 0.5 within the first epoch and then raise it to 0.9 in the rest epochs. It worked well and greatly accelerate the training progress. With this set of hyper-parameters, we achieved an accuracy at 98.54% after 50 epochs, with 60 iterations each epoch. The chart containing training and test loss is shown below.

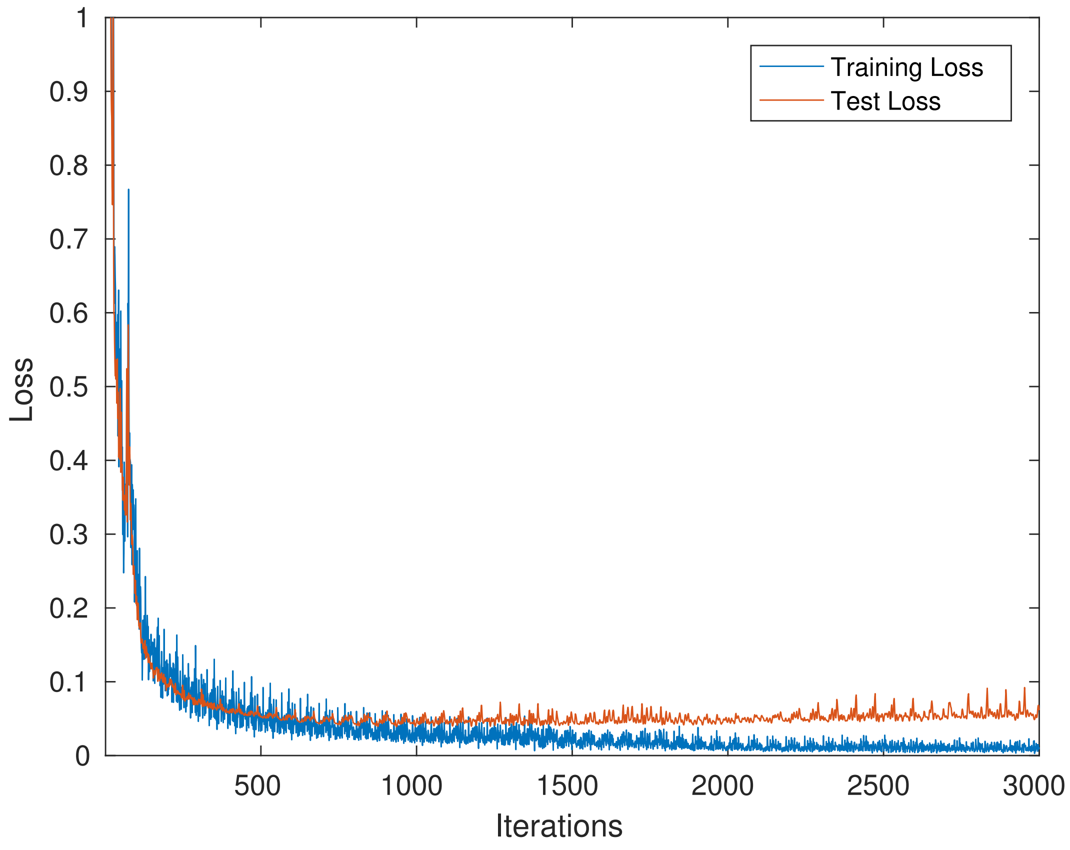

Figure 2: Baseline loss curve

According to the curves shown above, we can find that both training and test loss decreased dramatically in first 500 iterations. After 500 iterations, test loss remains fluctuating while training loss continue decreasing, indicating that the network is overfitting.

With the same hyper-parameters and epochs, we conducted experiment with leaky-ReLU and batch normalization layers. For leaky-ReLU, we simply replaced all ReLU layers with leaky-ReLU layers. In this experiment, we achieved an accuracy at 98.62%. The training and test loss are plotted below.

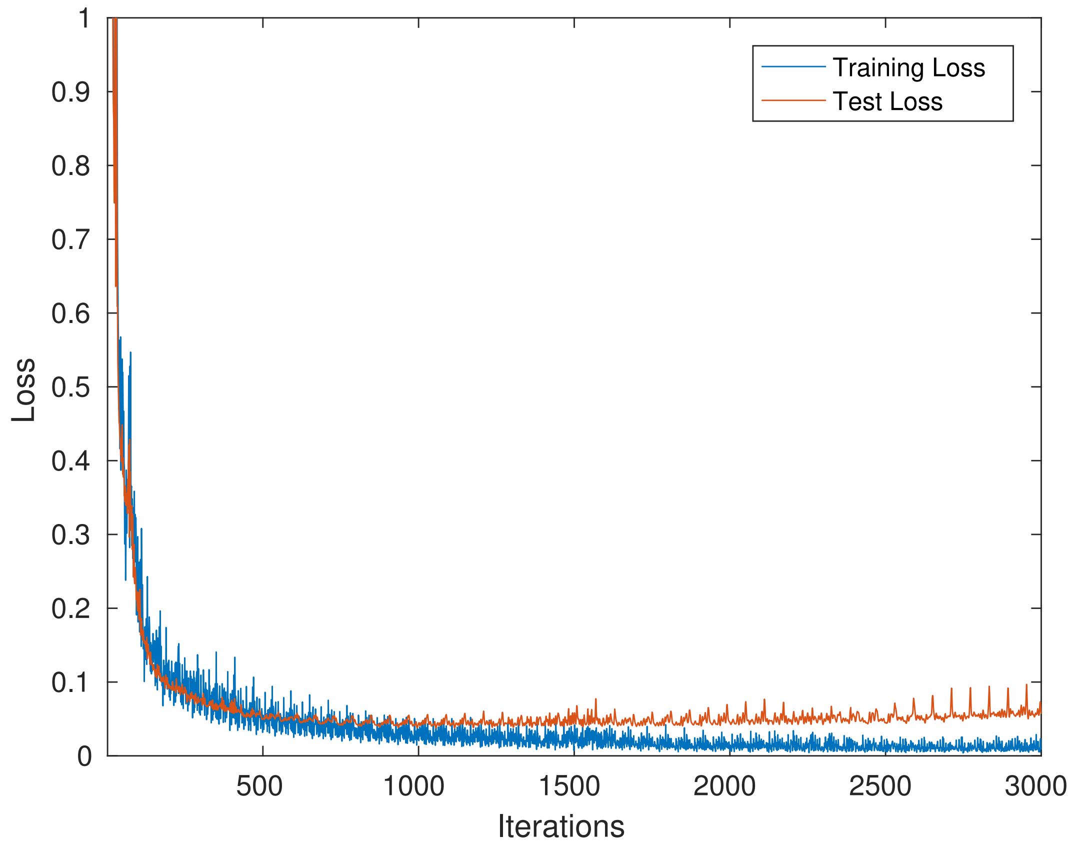

Figure 3: Leaky-ReLU loss curve

This figure indicates that the network is still overfitting after several epochs. So leaky-ReLU layers only slightly improve the performance but remain overfitting unsolved.

As a comparative experiment, we added batch normalization layers in front of nonlinearity layers of our baseline. The new architecture is shown below.

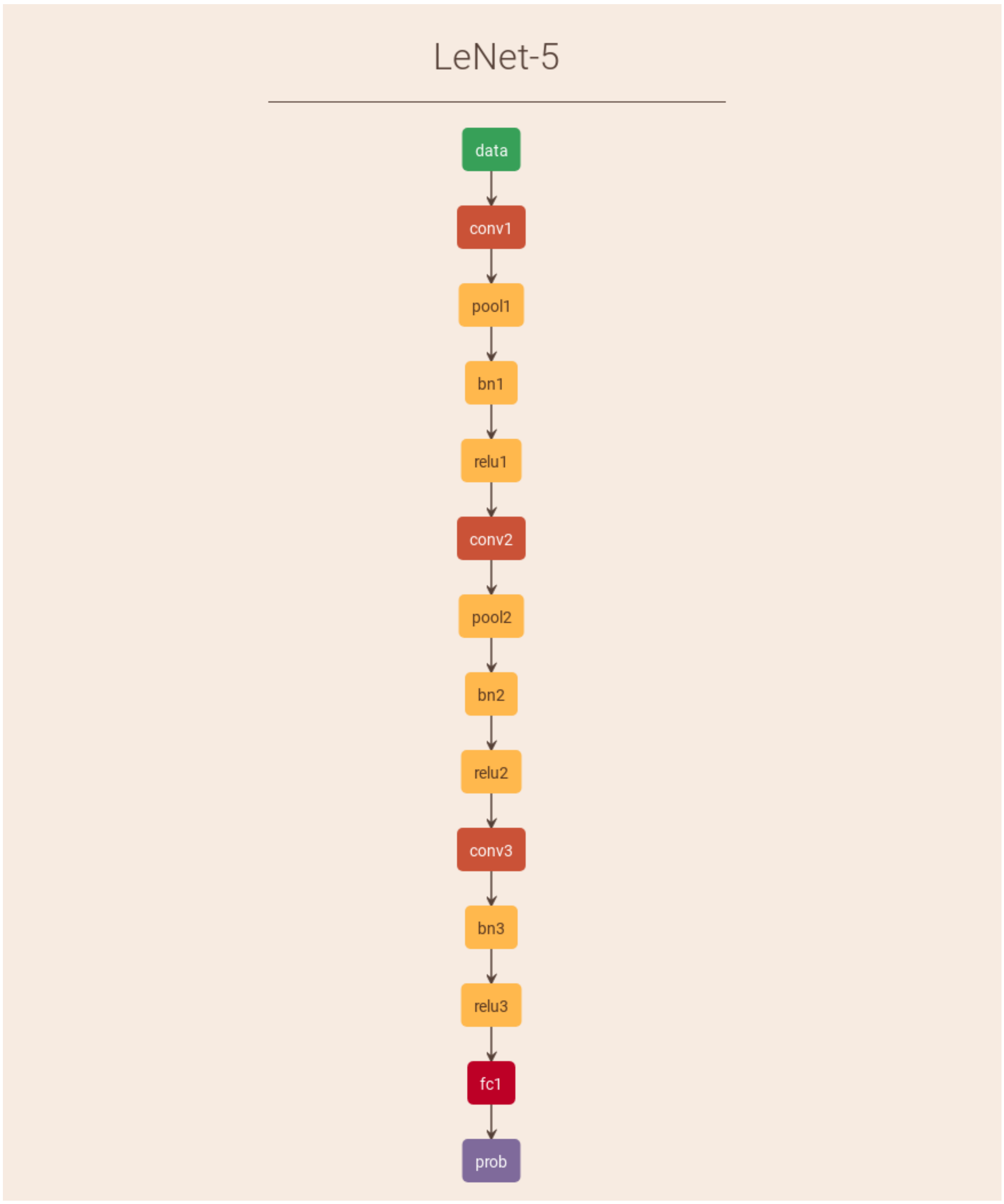

Figure 4: LeNet-5 with batch normalization

With same hyper-parameters and training epochs, we achieved 98.58%, slightly better than baseline. Loss curve is shown below.

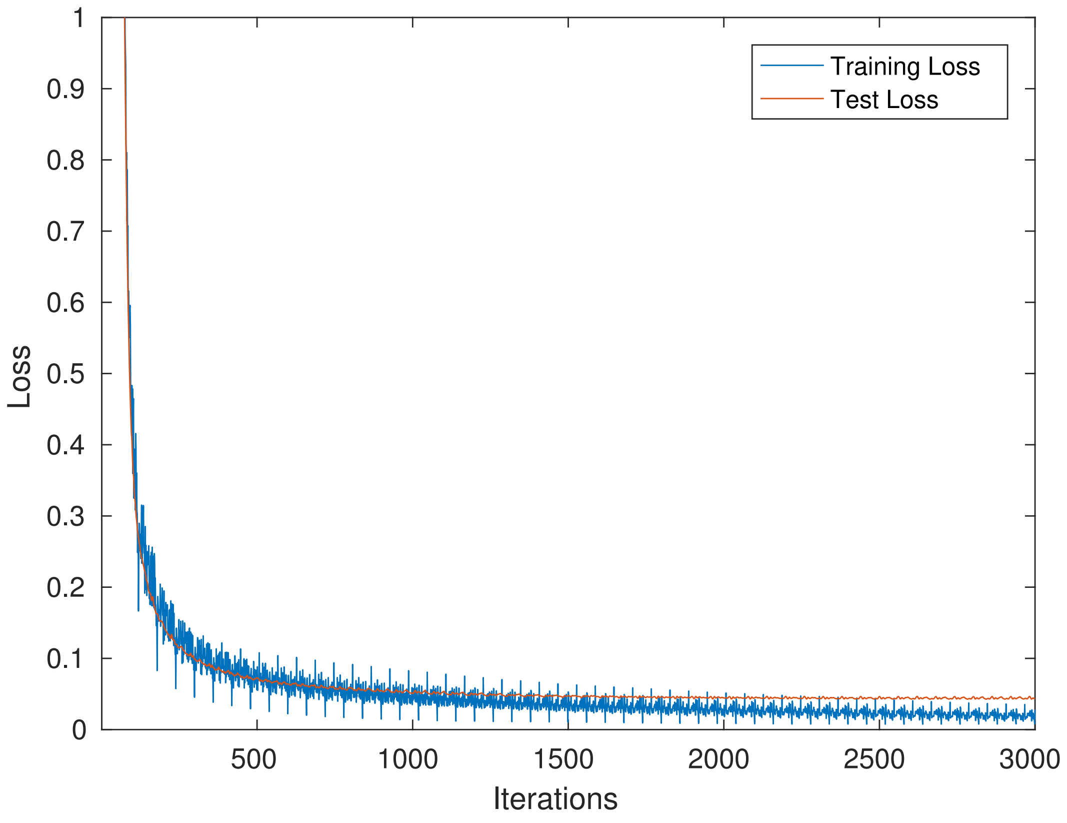

Figure 5: Batch normalization loss curve

According to this curve, we can figure out that the gap between training and test loss is smaller than those two networks before, indicating that batch normalization is effective on preventing the network from overfitting.

Discussions on baseline, leaky-ReLU and batch normalization

To start with, we plotted the loss curve of all the 3 architectures with training and test set respectively. And we also listed the final training and test loss after 50 epochs in the chart below.

	Baseline	Leaky-ReLU	Batch normalization
Training loss	0.0111	0.0083	0.0282
Test loss	0.0615	0.0569	0.0433

Table 1: Loss after 50 epochs

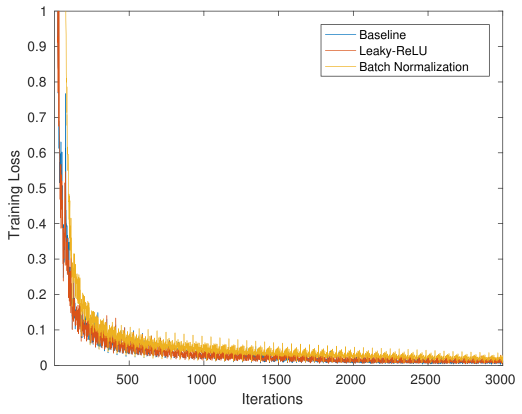

Figure 6: Training loss

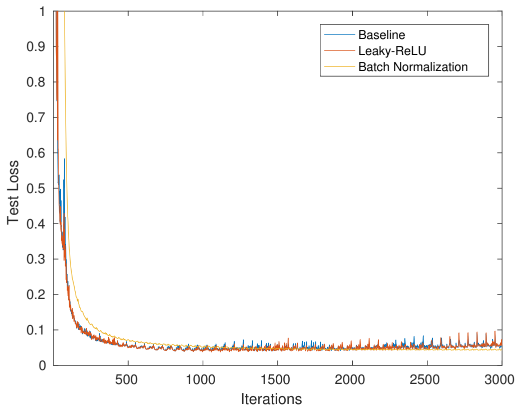

Figure 7: Test loss

The chart and two figures above indicate that both baseline and leaky-ReLU model are overfitting, producing lower training loss and higher test loss than batch normalization. In contrast, model with batch normalization achieved a lower test loss, even though its training loss is a little bit higher. As for exact performance on test set, the final accuracy after 50 epochs is shown below.

	Baseline	Leaky-ReLU	Batch normalization
Accuracy	98.54%	98.62%	98.58%

Table 2: Accuracy after 50 epochs

We can find that there is no significant difference on accuracy between these models. And there is a hypothesis that the test set is similar with training set so that overfitting doesn't exert impart on final performance.

In conclusion, all the 3 models can achieve desirable accuracy on MNIST dataset, while baseline and leaky-ReLU model are overfitting after several epochs. But the model with batch normalization is free from overfitting, achieve a lower test loss. Although it doesn't improve the performance on MNIST dataset, it will be a powerful tool on other datasets, which are larger and more complicated.