Deep Learning Final Project Report

Name: Dazhi Li
RUID: 197007456

1. Abstract

This is a type A project discussing on how batch normalization layer in convolution layers and dropout in fully connection layer influence the model performance. Performance are evaluated from total time consuming, last epoch accuracy and accuracy since all the epoch.

2. Introduction

Since we all know batch normalization and dropout will influence our training process, we are going to dig deeper into how they influence our model performance. We know batch normalization will increase the gradient flow in back propagation but will it increase our test accuracy or really accelerate our training in time axis? And what are the exact advantages of dropout? I take LeNet5 with MNIST hand written data sets as my example.

3. Related work

I did a related work in my homework 4 which is a convolution network based on LeNet5 to do classification. The training set and testing set are CIFAR-10 data sets. I have tried the simplest way of LeNet5 without adding anything to it and calculate the accuracy and running time. So does the LeNet5 with one dropout layer and LeNet5 with one batch normalization layer. But I did not compare them to get any results on batch normalization and dropout' s influence from that project. That is where the current experiment differs from that. Also, data sets are changed to MNIST hand written datasets.

4. Data description

The data could be discussed in many ways. First of all, LeNet5 requires the raw image size to be 32 x 32. However, the data from MINIST hand written is 28 x 28 size. To enable our LeNet5 model, we need to pad the image size to 32 x 32. Secondly, I would like to mention that there are 50000 training data while there are 10000 testing data.

5. Method description/Experiment procedure

The experiment method is not so hard. I firstly set a list of different kind of models in it. Then I call a loop to run every model one by one. I set another two lists which is empty. One list is total running time list which will record the total time for one specific model training and testing. Another list is total accuracy model, which will record the accuracy of the last testing period in the last epoch. Moreover, I set a dictionary for every model to store their running time for one epoch and accuracy in one epoch. I could directly watch how does the training epoch influence the model in the iteration. Then I use matplotlib library to draw six intuitive maps for me to better view the result. By comparing the data shown in the figure we can deduce our results.

6. Model description

Before I tell what exactly the model is, I should mention some important hyperparameters in

this experiment. The batch size was set to 128 while the learning rate is 0.01. The optimizer I used in this experiment is SGD optimizer without momentum or anything. Loss function is cross entropy loss and epoch is set to be 10 which is sufficient for MNIST data sets.
There are totally four kinds of models. The first one is a classical LeNet5 model without any batch normalization layer or dropout layer. Code on model is in **box 1**.

```
class LeNet(nn.Module):
   def __init__(self):
      self.name='LeNet'
      super(LeNet, self).__init__()
      self.convnet = nn.Sequential(OrderedDict([
         ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5),padding=2)),
         ('relu1', nn.ReLU()),
         ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
         ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
         ('relu3', nn.ReLU()),
         ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
         ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
         ('relu5', nn.ReLU())
      ]))
      self.fc = nn.Sequential(OrderedDict([
         ('f6', nn.Linear(120, 84)),
         ('relu6', nn.ReLU()),
         ('f7', nn.Linear(84, 10)),
         ('sig7', nn.LogSoftmax(dim=-1))
      ]))
   def forward(self, x):
      x = self.convnet(x)
      x = x.view(x.size()[0], -1)
      out = self.fc(x)
      return out
```

Box 1 LeNet5 model code

Attribute name is specified for storing data into dictionary by name.

The second model is LeNet5 model with three batch normalization layer in convolution layer part. Those batch normalization layers are inserted between the convolution layer and activation function. Details could be viewed in **box 2**.

```
class LeNet_BN_noDropout(nn.Module):
   def __init__(self):
      self.name='LeNet_BN_noDropout'
      super(LeNet_BN_noDropout, self).__init__()
      self.convnet = nn.Sequential(OrderedDict([
         ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5),padding=2)),
         ('bn1', nn.BatchNorm2d(6)),
         ('relu1', nn.ReLU()),
```

```
            ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
            ('bn3', nn.BatchNorm2d(16)),
            ('relu3', nn.ReLU()),
            ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
            ('bn5', nn.BatchNorm2d(120)),
            ('relu5', nn.ReLU())
        ]))
        self.fc = nn.Sequential(OrderedDict([
            ('f6', nn.Linear(120, 84)),
            ('relu6', nn.ReLU()),
            ('f7', nn.Linear(84, 10)),
            ('sig7', nn.LogSoftmax(dim=-1))
        ]))
    def forward(self, x):
        x = self.convnet(x)
        x = x.view(x.size()[0], -1)
        out = self.fc(x)
        return out
```

Box 2 LeNet5 with batch normalization without dropout

The third kind of model is a LeNet5 with two dropout layers in the fully connection part while the convolution part keeps the same as classical LeNet5 model. The hyperparameter was set to be 0.5 for each drop out layer. Details could be viewed in **box 3**.

```
class LeNet_noBN_Dropout(nn.Module):
    def __init__(self):
        self.name='LeNet_noBN_Dropout'
        super(LeNet_noBN_Dropout, self).__init__()
        self.convnet = nn.Sequential(OrderedDict([
            ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5),padding=2)),
            ('relu1', nn.ReLU()),
            ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
            ('relu3', nn.ReLU()),
            ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
            ('relu5', nn.ReLU())
        ]))
        self.fc = nn.Sequential(OrderedDict([
            ('f6', nn.Linear(120, 84)),
            ('drop6', nn.Dropout(0.5)),
            ('relu6', nn.ReLU()),
            ('f7', nn.Linear(84, 10)),
```

```
            ('drop7', nn.Dropout(0.5)),
            ('sig7', nn.LogSoftmax(dim=-1))
        ]))
    def forward(self, x):
        x = self.convnet(x)
        x = x.view(x.size()[0], -1)
        out = self.fc(x)
        return out
```

Box 3 LeNet5 with dropout without batch normalization

The last model is LeNet5 with both batch normalization layers and dropout layers. The dropout rate is still 0.5 to maintain the same with the third model. Details are in **box 4**.

```
class LeNet_BN_Dropout(nn.Module):
    def __init__(self):
        self.name='LeNet_BN_Dropout'
        super(LeNet_BN_Dropout, self).__init__()
        self.convnet = nn.Sequential(OrderedDict([
            ('c1', nn.Conv2d(1, 6, kernel_size=(5, 5),padding=2)),
            ('bn1', nn.BatchNorm2d(6)),
            ('relu1', nn.ReLU()),
            ('s2', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c3', nn.Conv2d(6, 16, kernel_size=(5, 5))),
            ('bn3', nn.BatchNorm2d(16)),
            ('relu3', nn.ReLU()),
            ('s4', nn.MaxPool2d(kernel_size=(2, 2), stride=2)),
            ('c5', nn.Conv2d(16, 120, kernel_size=(5, 5))),
            ('bn5', nn.BatchNorm2d(120)),
            ('relu5', nn.ReLU())
        ]))
        self.fc = nn.Sequential(OrderedDict([
            ('f6', nn.Linear(120, 84)),
            ('drop6', nn.Dropout(0.5)),
            ('relu6', nn.ReLU()),
            ('f7', nn.Linear(84, 10)),
            ('drop7', nn.Dropout(0.5)),
            ('sig7', nn.LogSoftmax(dim=-1))
        ]))
    def forward(self, x):
        x = self.convnet(x)
        x = x.view(x.size()[0], -1)
        out = self.fc(x)
        return out
```

Box 4 LeNet5 with batch normalization and dropout

7. Results

From running the code I provide in the zip file we could directly get the following figure 1. One thing I need to mention is this code is run on a CUDA supported machine. If not, please manually modify the device to CPU.

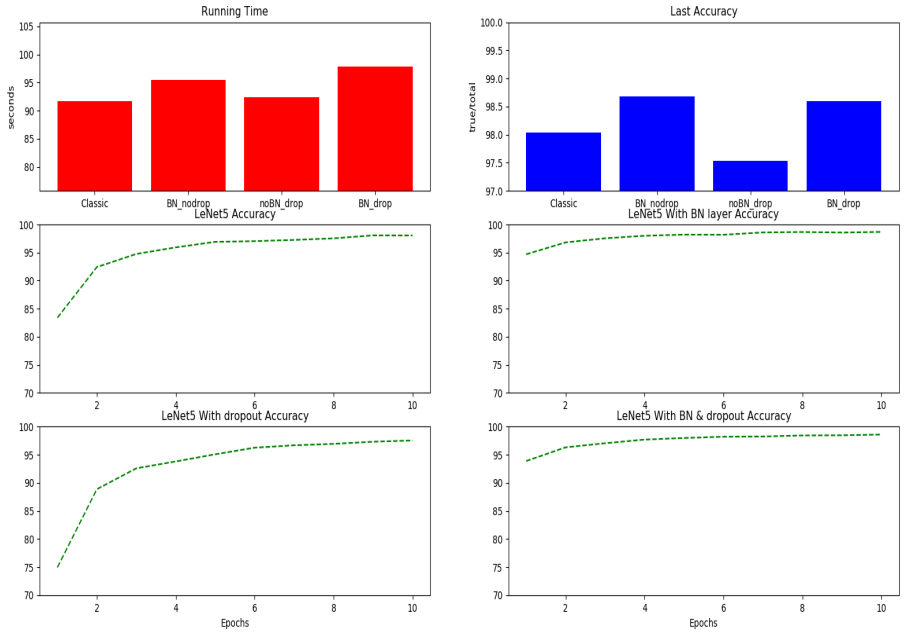

Figure 1 Running results

From the figure above, we could find that the total running time is different from each models. The running time means the total time to run a model for 10 epochs. I also stored the running time for each model in each epoch but I found that the time is almost the same for every epoch, so we just analyze the total running time difference between each models. We could find that there is not much difference since our model is small. However, comparing both batch normalization and dropout to the classical one, we will find that the classical one is consuming less time. In other points of view, less energy consuming.

Then comparing the accuracy in the second bar chart, we will find that dropout did not increase the accuracy like what we have learnt in our classes. I am wondering if I make too much neurons to be zero, so I did another experiment on my laptop which has different models than those I mentioned above. My laptop only have CPUs so it takes a lot of time to do it. But time is not we are expecting something different from our previous model. The biggest change is I decrease the number of dropout layers for both no batch normalization with dropout model and batch normalization with dropout model. I manually delete one raw of code where the 'drop7' layer exists. The new result will be shown in **figure 2**.

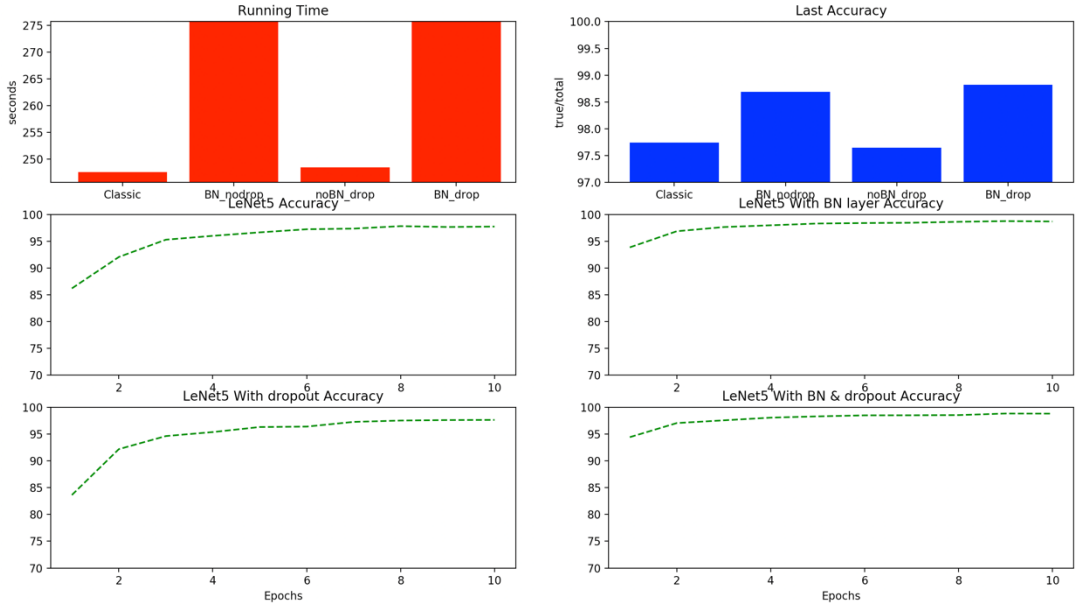

Figure 2 Only one dropout layer

The results here show that simply adding dropout layer will still not increase the accuracy. However, if we train the model too well with batch normalization, the situation is different. We can observe there is little improvement for batch normalization with dropout model.

Then, let us compare those leaning status of each model. We will easily find that no matter in what kind of situation, batch normalization will increase the accuracy at the begging of the training. As a consequence, the final accuracy is higher than those without batch normalization layers. Simply adding dropout layer will increase the learning speed from the figures above, I think that is simply because the first time accuracy for dropout is too bad.

8. Conclusion

From those results above we can deduce the following conclusions:

(1) Batch normalization and dropout will increase the total execution time for training and testing.

(2) Batch normalization will increase the accuracy a lot, especially for the first epoch.

(3) Dropout too much will decrease the training accuracy. Neural network should find a right dropout rate to make its accuracy increase.

(4) Dropout will prevent model from training too well, this will make the test accuracy better.

(5) Simply adding dropout layer will have a very bad performance for its first epoch. However, this will result a better learning speed that other methods.