PA3 MLP-CNN Report

计 72　刘光哲　2017011329

2018.10.26

目录

1 Environment

Operating System: macOS 10.14 Mojave
Python: Python 3.6.6
tensorflow: tensorflow 1.11.0
CPU: Intel Core i7-8750H, Memory: 16 GB

2 Code Review

The code of PA3 is much more easier to analyse than PA1 and PA2 since tensorflow is allowed to use.

2.1 basic architecture

The code for MLP and CNN share the same architecture, so I'll only analyse MLP for instance.

main.py *main.py* is the main controller of the code, in which we manage the training of the model. During training process, it first load data from dataset, and then run *train_epoch* to train the model (forward and backward), and run *valid_epoch* to test the model.

load_data.py *load_data.py* just gives a method to read the data from the dataset.

model.py In *model.py* we define a class $Model$. In the construction function of the class, we assign some parameters to build the model, and then train the model. In $foward$ method, we construct a neural network layer by layer and calculate the value with tensorflow default settings, and the Optimizer we assigned will then be used to do the *backward* process by default. Tensorflow allows us only to consider the forward pass of the model, and deal with the backward pass itself. This apparently improved the efficiency of realizing a model.

2.2 arguments of forward function

There are two arguments for $forward$ method: is_train and $reuse$.

If $reuse$ is $True$, tensorflow would use the parameters defined in the namescope before, otherwise it would construct some new parameters for training; if $reuse$ is $tf.AUTO\_REUSE$, $get_variable$ will decide whether to construct new parameters automatically. In this case, I assign the $reuse$ in the first $forward$ to be $tf.AUTO\_REUSE$, and the one in the second to be $True$, so that we can construct parameters at first, and then continue training and testing on the same group of parameters.

If is_train is $True$, the forward pass will do the normal forward pass. If is_train is $False$, the forward pass won't do *batch_normalization* and *dropout_layer*, cause these two tricks are used to improved training quality, and shouldn't be used when testing.

```
1 self.loss, self.pred, self.acc = self.forward(is_Train, tf.AUTO_REUSE)
2 self.loss_val, self.pred_val, self.acc_val = self.forward(is_Train, True)
```

2.3 model construction

In this part I'll explain the model construction part in my model.

2.3.1 batch_normalization

With *tf.layers*, I found many well-encapsuled layer realizations. As a result, I used *tf.layers.batch_normalization* to implement this method:

```
if is_train:
    return tf.layers.batch_normalization(incoming, momentum=0.99, epsilon=1e-5,
        training=True)
else:
    return tf.layers.batch_normalization(incoming, training=False)
```

2.3.2 dropout_layer

Similar to *batch_normalization*, I used *tf.layers.dropout* to implement this method:

```
if is_train:
    return tf.layers.dropout(incoming, rate=drop_rate)
else:
    return incoming
```

2.3.3 MLP layers

```
w_fc1 = tf.get_variable('w_fc1', shape=[28*28, 256])
b_fc1 = tf.get_variable('b_fc1', shape=[256])
h_fc1 = tf.nn.relu(batch_normalization_layer(tf.matmul(self.x_, w_fc1) + b_fc1,
     is_train=is_train) )
ho_fc1 = dropout_layer(h_fc1, 0.3, is_train)
w_fc2 = tf.get_variable('w_fc2', shape=[256, 10])
b_fc2 = tf.get_variable('b_fc2', shape=[10])
logits = tf.matmul(ho_fc1, w_fc2) + b_fc2
```

2.3.4 CNN layers

Actually there is layer for convolution and maxpooling in *tf.layers*, but I didn't use them in order to implement *batch_normalization* and *dropout_layer*.

```
k_conv1 = tf.get_variable(name='k_conv1', shape=[3, 3, 1, 4])
b_conv1 = tf.get_variable(name='b_conv1', shape=[4])
h_conv1 = tf.nn.conv2d(self.x_,k_conv1,padding='SAME',strides=[1,1,1,1])+b_conv1
hr_conv1=dropout_layer(tf.nn.relu(batch_normalization_layer(h_conv1)),0.3,is_train)
p_pool1 = tf.nn.max_pool(hr_conv1,ksize=[1,2,2,1],strides=[1,2,2,1],padding="SAME")
k_conv2 = tf.get_variable(name='k_conv2', shape=[3, 3, 4, 4])
b_conv2 = tf.get_variable(name='b_conv2', shape=[4])
h_conv2= tf.nn.conv2d(p_pool1,k_conv2,padding='SAME',strides=[1,1,1,1])+b_conv2
hr_conv2=dropout_layer(tf.nn.relu(batch_normalization_layer(h_conv2)),0.3,is_train)
p_pool2 = tf.nn.max_pool(hr_conv2,ksize=[1,2,2,1],strides=[1,2,2,1],padding="SAME")
flt = tf.reshape(p_pool2, shape=[-1, 196])
```

```
w_fc3 = tf.get_variable(name='w_fc3', shape=[196, 10])
b_fc3 = tf.get_variable(name='b_fc3', shape=[10])
logits = tf.matmul(flt, w_fc3) + b_fc3
```

3 Experiments and Answers

3.1 Test with MLP and CNN

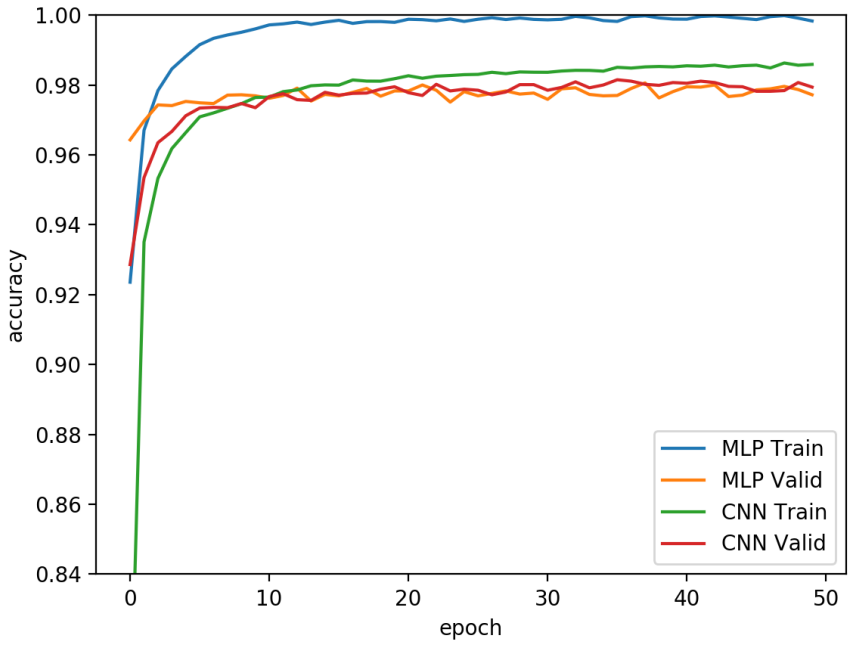
I succeed training with these parameters:

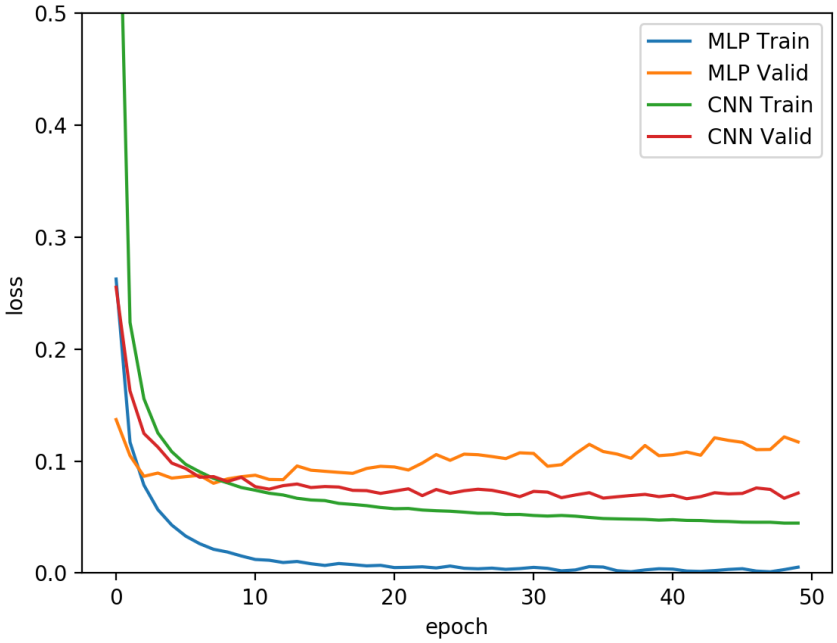
neural network	learning rate	momentum	epsilon	batchsize
MLP	0.001	0.99	1e-5	100
CNN	0.001	0.99	1e-5	100

The training lasts 50 epochs, end up like this:

neural network	time	test accuracy
MLP	1 min	0.9777
CNN	10 min	0.9817

The training process is like this:





We can see from the graph that:

- MLP converges much more faster than CNN
- MLP has a higher training accuracy, but has a lower validation accuracy; MLP has a lower loss value, but has a higher validation value
- obviously CNN will converge at a higher accuracy
- CNN needs a much longer training time than MLP because of convolution calculation
- during the most time of the training, training loss is lower than validation loss, because the model fits the training data better than validation data
- When testing CNN, validation loss is lower than training loss at first, this shows CNN is less troubled by over-fitting

At the same time, when comparing these results to those in PA1 and PA2, I found that the MLP of tensorflow is little slower than mine, but the CNN of tensorflow is much more faster than mine.

3.2 Test on batch normalization

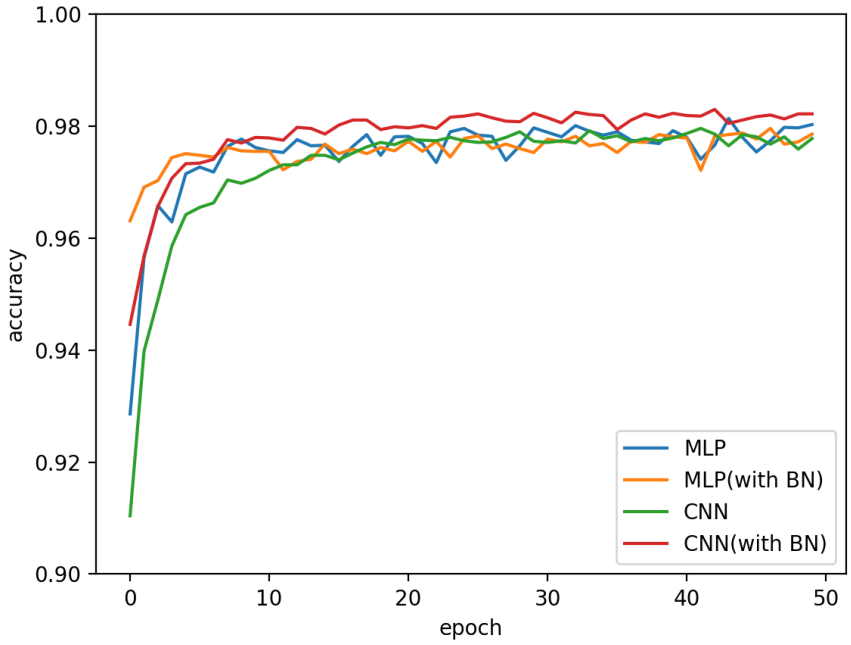
I did these tests with CNN and MLP:

neural network	learning rate	momentum	epsilon	batchsize
MLP	0.001	-	-	100
MLP(with BN)	0.001	0.99	1e-5	100
CNN	0.001	-	-	100
CNN(with BN)	0.001	0.99	1e-5	100

After 50 epochs of training, the results are:

neural network	time	test accuracy
MLP	45 s	0.9811
MLP(with BN)	65 s	0.9779
CNN	6 min	0.9771
CNN(with BN)	11 min	0.9788

The validation accuracy during the training process is liek this:



This shows the following conclusions:

- batch normalization is of little help on MLP, there's no difference between MLP and MLP with BN
- batch normalization obviously improved the training quality: CNN with BN covnergers a little faster and converges to a higher limitation
- batch normalization apparently effects training time: models with BN takes a half more time in training

3.3 Test on drop rate

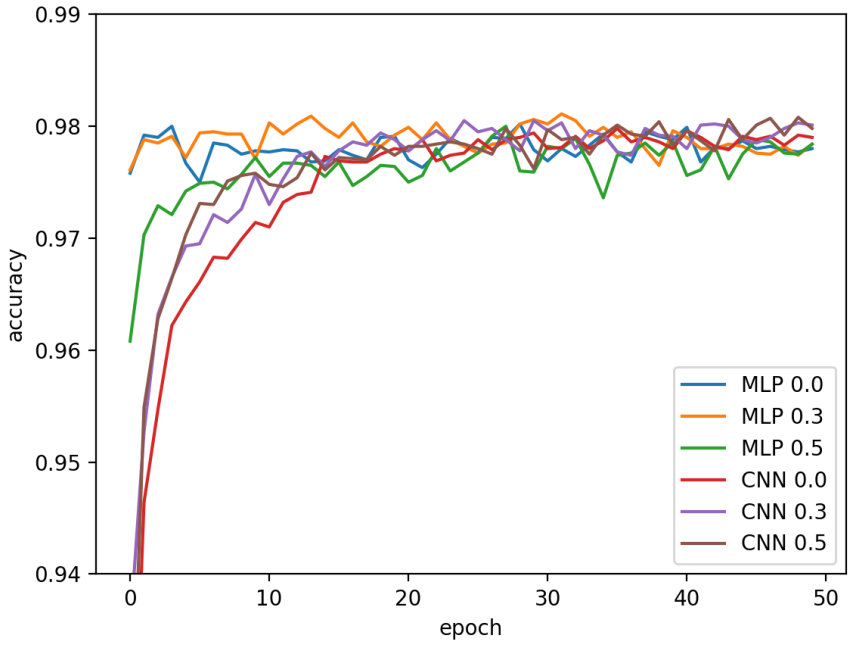
I did these tests with CNN and MLP:

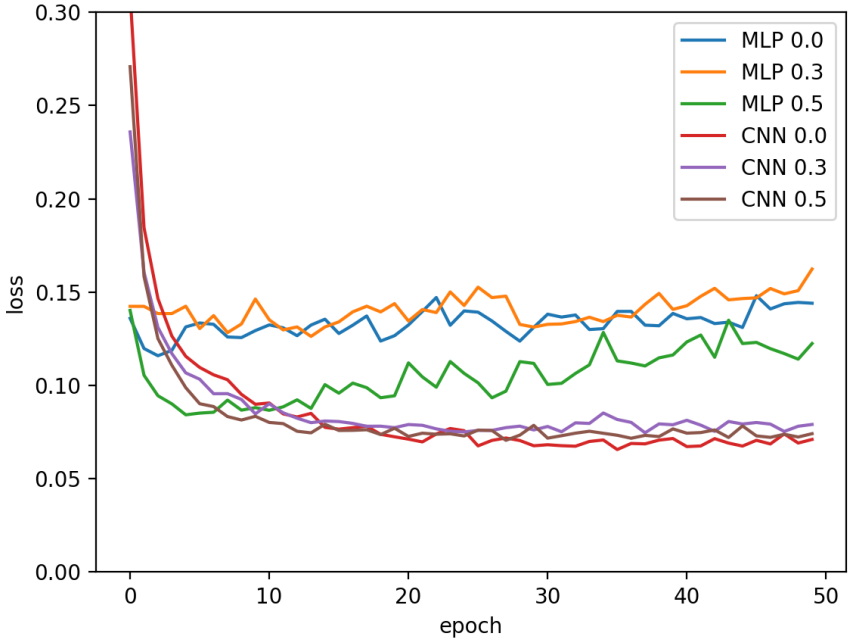
neural network	learning rate	momentum	epsilon	batchsize	drop rate
MLP	0.001	0.99	1e-5	100	0.0
MLP	0.001	0.99	1e-5	100	0.3
MLP	0.001	0.99	1e-5	100	0.5
CNN	0.001	0.99	1e-5	100	0.0
CNN	0.001	0.99	1e-5	100	0.3
CNN	0.001	0.99	1e-5	100	0.5

After 50 epochs of training, the results are:

neural network	drop rate	test accuracy
MLP	0.0	0.9773
MLP	0.3	0.9791
MLP	0.5	0.9776
CNN	0.0	0.9796
CNN	0.3	0.9780
MLP	0.5	0.9811

The training process is like this:





From the test result we can see that:

- In general, CNN has better accuracy and loss value than MLP

- MLP has bigger variance than CNN, it may be because it has less layers and parameters
- 0.3 seems to be the best drop rate for MLP, and 0.5 is the best drop rate for CNN

3.4 Explanation about training loss and validation loss

Training loss is different from validation loss because validation dataset is not the same as training dataset. Models will fit the training data better after enough epochs of training, but the model would not adjust its parameters to satisfy the validation dataset.

4 Summary

In this experiments, I implemented MLP and CNN with tensorflow, and understood the advantage of using tensorflow. I learned these from this PA:

- the use of Tensorflow namescope
- the difference between nn, layers and contrib
- the meaning of batch normalization and drop out
- the importance of Google and official documents