hw4 5525

Kexin Feng 5165462 fengx463@umn.edu

December 2018

1 Prob 2

The tensorboard output has been stored in directory tb_prob2. You can use the following command to get the url and open it with browser.

```
$tensorboard --logdir=tb\_prob2
```

The test accuracy is 0.8983

The training performance:

```
Average loss epoch 0: 1.2891434678386697
Average loss epoch 1: 0.7327894783798076
Average loss epoch 2: 0.6007488074697259
Average loss epoch 3: 0.5360953472988866
Average loss epoch 4: 0.49808518446130906
Average loss epoch 5: 0.470791361340276
Average loss epoch 6: 0.4516749320191381
Average loss epoch 7: 0.4355761824926852
Average loss epoch 8: 0.42291591550920393
Average loss epoch 9: 0.4135298542765193
Total time: 23.235044956207275 seconds
Optimization Finished!
Accuracy 0.8983
```

2 Prob 3

For this question, instead of GradientDescent I used AdamOptimizer, because GradientDescent doesn't converge fast enough. But if you want to use GradientDescent, you can use the follow command line:

```
$python ConvNetTemplate.py default
```

The convolutional neural network is based on the confolutional product of the 'image' data an the filter.

$$S[i,j] = I * K = \sum_m \sum_n I[m,n]K[i-m,j-n] \tag{1}$$

A CNN layer is very similar to a layer of ordinary Neural Network in a sense that they both, in each channel, capture certain features from the input layer, and that they both can increase or decrease the dimension. The difference is that ordinary neural network doesn't share weight parameters and the parameter space is big; it's less explainable other than seeing it as a regression fitting. The convolutional network, however, share the filters, which has much less degrees of freedom. And when a region of image has similar pattern as filter, the corresponding

The tensorboard output has been stored in directory tb_prob3. You can use the following command to get the url and open it with browser.

```
$tensorboard --logdir=tb_prob3
```

The test accuracy is 0.918

The training performance:

```
Average loss at step 240: 2453.8
```

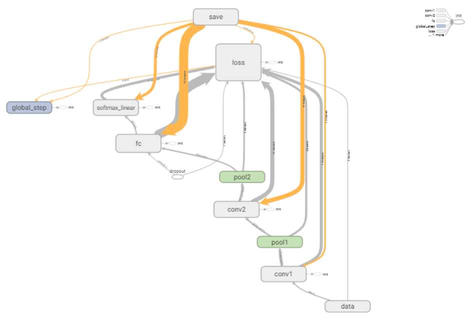


Figure 1: CNN for prob 3

```
Average loss at step 250: 1825.0
Average loss at step 260: 1893.2
Average loss at step 270: 1855.9
Average loss at step 280: 1733.9
Average loss at step 290: 1502.8
Average loss at step 300: 1827.1
Average loss at step 310: 1652.2
Average loss at step 320: 1835.9
Average loss at step 330: 1649.5
Average loss at step 340: 1539.7
Average loss at step 350: 1399.6
Average loss at step 360: 1543.7
Average loss at step 370: 1650.7
Average loss at step 380: 1370.9
Average loss at step 390: 1449.2
Average loss at step 400: 1470.6
Average loss at step 410: 1079.3
Average loss at step 420: 1088.2
Optimization Finished!
Total time: 282.3850169181824 seconds
Accuracy 0.918
```

3 Prob 4

This problem, I did task one. To increase the accuracy, I chose RNN, more specifically, bidirectional LSTM and gru. The reason of choosing this model is because in a digit graph, each digit are not totally independent from each other; they have statistical correlation for a certain hand written number. So using LSTM or gru, we can treat rows as time sequences, so this time correlation information can be captured and the accuracy can be bigger. The bidirectional lstm was chosen because a simple lstm has the problem of forgetting the beginning part. So bidirectional lstm can capture the information of both the beginning and the ending part.

LSTM is gated RNN. One hidden unit in an LSTM cell is shown in the figure 4. It has a forget gate, an input gate and output gate, all of which output a value between 0 and 1, controlling how much information can pass through. It also has two channels, the output channel and memory channel. The memory channel is what used to store the long term memory and constitute part of the output. The output channel takes input at each step and outputs the result to the user and recurrently to the next-step input.

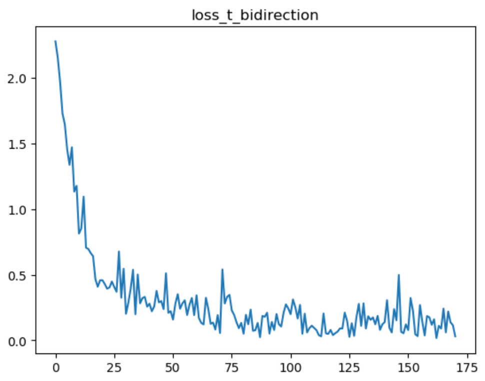
This is a classification problem. Each data is assigned to one of ten distinguished classes, corresponding to ten numbers. The loss function is the cross entropy.

$$L(y(x_i), t_i) = \sum_{n,i} t_{n,i} \log y_{n,i} \tag{2}$$

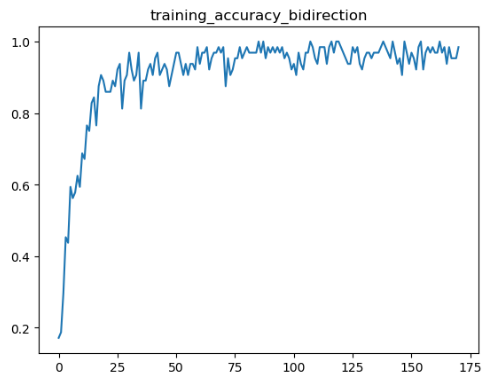
The optimizer is Adam Optimizer.

I tried two models, bidirectional LSTM and GRU. Their results behave similar to each other and both reach 0.97! It's shown in Fig. 2 and Fig.3

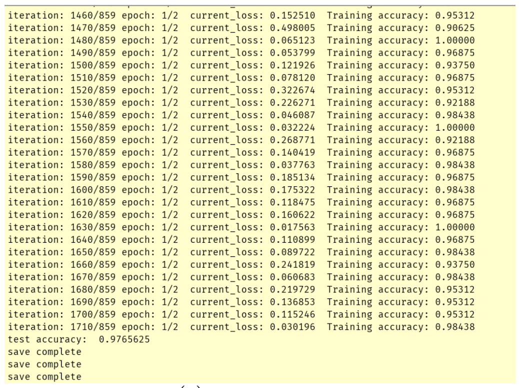
Bidirectional LSTM test accuracy: 0.9765625
GRU test accuracy: 0.9765625



(a) loss

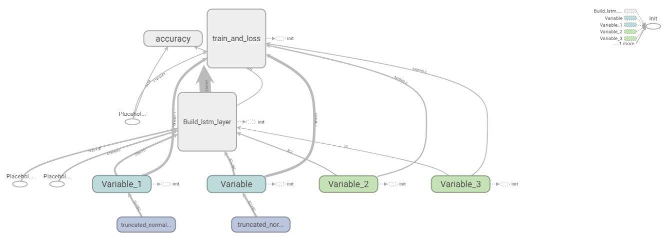


(b) accuracy



```
iteration: 1460/859 epoch: 1/2  current_loss: 0.152510  Training accuracy: 0.95312
iteration: 1470/859 epoch: 1/2  current_loss: 0.498005  Training accuracy: 0.90625
iteration: 1480/859 epoch: 1/2  current_loss: 0.065123  Training accuracy: 1.00000
iteration: 1490/859 epoch: 1/2  current_loss: 0.053799  Training accuracy: 0.96875
iteration: 1500/859 epoch: 1/2  current_loss: 0.121926  Training accuracy: 0.93750
iteration: 1510/859 epoch: 1/2  current_loss: 0.078120  Training accuracy: 0.96875
iteration: 1520/859 epoch: 1/2  current_loss: 0.322674  Training accuracy: 0.95312
iteration: 1530/859 epoch: 1/2  current_loss: 0.226271  Training accuracy: 0.92188
iteration: 1540/859 epoch: 1/2  current_loss: 0.046087  Training accuracy: 0.98438
iteration: 1550/859 epoch: 1/2  current_loss: 0.032224  Training accuracy: 1.00000
iteration: 1560/859 epoch: 1/2  current_loss: 0.268771  Training accuracy: 0.92188
iteration: 1570/859 epoch: 1/2  current_loss: 0.140419  Training accuracy: 0.96875
iteration: 1580/859 epoch: 1/2  current_loss: 0.037763  Training accuracy: 0.98438
iteration: 1590/859 epoch: 1/2  current_loss: 0.185134  Training accuracy: 0.96875
iteration: 1600/859 epoch: 1/2  current_loss: 0.175322  Training accuracy: 0.98438
iteration: 1610/859 epoch: 1/2  current_loss: 0.118475  Training accuracy: 0.96875
iteration: 1620/859 epoch: 1/2  current_loss: 0.160622  Training accuracy: 0.96875
iteration: 1630/859 epoch: 1/2  current_loss: 0.017563  Training accuracy: 1.00000
iteration: 1640/859 epoch: 1/2  current_loss: 0.110899  Training accuracy: 0.96875
iteration: 1650/859 epoch: 1/2  current_loss: 0.089722  Training accuracy: 0.98438
iteration: 1660/859 epoch: 1/2  current_loss: 0.241819  Training accuracy: 0.93750
iteration: 1670/859 epoch: 1/2  current_loss: 0.060683  Training accuracy: 0.98438
iteration: 1680/859 epoch: 1/2  current_loss: 0.219729  Training accuracy: 0.95312
iteration: 1690/859 epoch: 1/2  current_loss: 0.136853  Training accuracy: 0.95312
iteration: 1700/859 epoch: 1/2  current_loss: 0.115246  Training accuracy: 0.95312
iteration: 1710/859 epoch: 1/2  current_loss: 0.030196  Training accuracy: 0.98438
test accuracy:  0.9765625
save complete
save complete
save complete
```

(c) test accuracy



(d) tensorboard

Figure 2: Bidirectional LSTM.

4 Prob 5

In this problem, I treated the digit rows as a time series of vectors of dimension 28. So the number of hidden units of LSTM cell is set to be 28. The model used in this problem, Long Short Term Memory(LSTM), is a kind of recurrent neural network(RNN), which takes sequence as its input, one element at a time.

The detail mechenism and the equations have been described above in problem 4.

Then I treated this as a classification problem. Each data is assigned to one of ten distinguished classes, corresponding to ten numbers. The loss function is the cross entropy.

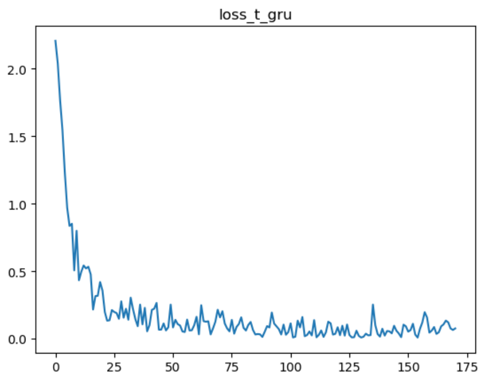
$$L(y(x_i), t_i) = \sum_{n,i} t_{n,i} \log y_{n,i} \tag{3}$$

The optimizer is Adam Optimizer. Test accuracy: 0.9921875

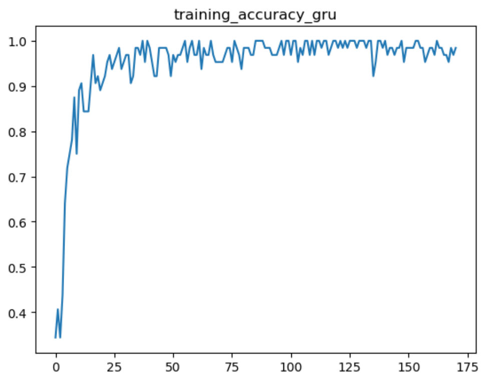
The training loss decrease, accuracy increase and the test accuracy is show below in figure 5

References

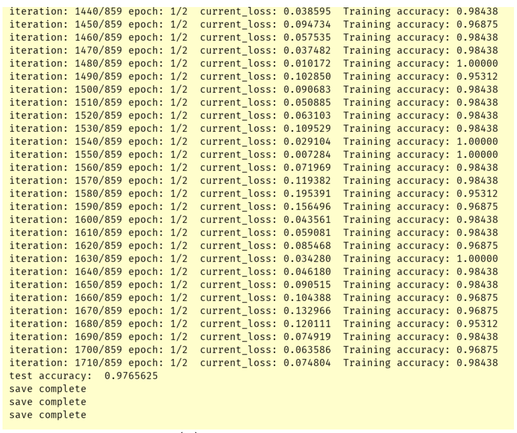
[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs/



(a) loss

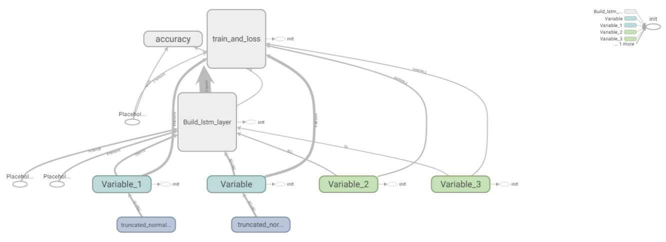


(b) accuracy



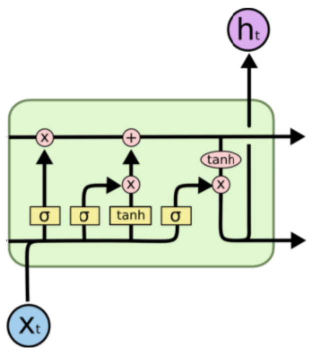
```
iteration: 1440/859 epoch: 1/2  current_loss: 0.038595  Training accuracy: 0.98438
iteration: 1450/859 epoch: 1/2  current_loss: 0.094734  Training accuracy: 0.96875
iteration: 1460/859 epoch: 1/2  current_loss: 0.057535  Training accuracy: 0.98438
iteration: 1470/859 epoch: 1/2  current_loss: 0.037482  Training accuracy: 0.98438
iteration: 1480/859 epoch: 1/2  current_loss: 0.010172  Training accuracy: 1.00000
iteration: 1490/859 epoch: 1/2  current_loss: 0.102850  Training accuracy: 0.95312
iteration: 1500/859 epoch: 1/2  current_loss: 0.090683  Training accuracy: 0.98438
iteration: 1510/859 epoch: 1/2  current_loss: 0.050885  Training accuracy: 0.98438
iteration: 1520/859 epoch: 1/2  current_loss: 0.063103  Training accuracy: 0.98438
iteration: 1530/859 epoch: 1/2  current_loss: 0.109529  Training accuracy: 0.98438
iteration: 1540/859 epoch: 1/2  current_loss: 0.029104  Training accuracy: 1.00000
iteration: 1550/859 epoch: 1/2  current_loss: 0.007284  Training accuracy: 1.00000
iteration: 1560/859 epoch: 1/2  current_loss: 0.071969  Training accuracy: 0.98438
iteration: 1570/859 epoch: 1/2  current_loss: 0.119382  Training accuracy: 0.98438
iteration: 1580/859 epoch: 1/2  current_loss: 0.195391  Training accuracy: 0.95312
iteration: 1590/859 epoch: 1/2  current_loss: 0.156496  Training accuracy: 0.96875
iteration: 1600/859 epoch: 1/2  current_loss: 0.043561  Training accuracy: 0.98438
iteration: 1610/859 epoch: 1/2  current_loss: 0.059081  Training accuracy: 0.98438
iteration: 1620/859 epoch: 1/2  current_loss: 0.085468  Training accuracy: 0.96875
iteration: 1630/859 epoch: 1/2  current_loss: 0.034280  Training accuracy: 1.00000
iteration: 1640/859 epoch: 1/2  current_loss: 0.046180  Training accuracy: 0.98438
iteration: 1650/859 epoch: 1/2  current_loss: 0.090515  Training accuracy: 0.98438
iteration: 1660/859 epoch: 1/2  current_loss: 0.104388  Training accuracy: 0.96875
iteration: 1670/859 epoch: 1/2  current_loss: 0.132966  Training accuracy: 0.96875
iteration: 1680/859 epoch: 1/2  current_loss: 0.120111  Training accuracy: 0.95312
iteration: 1690/859 epoch: 1/2  current_loss: 0.074919  Training accuracy: 0.98438
iteration: 1700/859 epoch: 1/2  current_loss: 0.063586  Training accuracy: 0.96875
iteration: 1710/859 epoch: 1/2  current_loss: 0.074804  Training accuracy: 0.98438
test accuracy:  0.9765625
save complete
save complete
save complete
```

(c) test accuracy



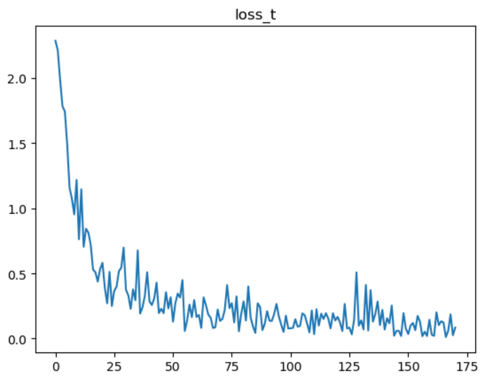
(d) tensorboard

Figure 3: GRU

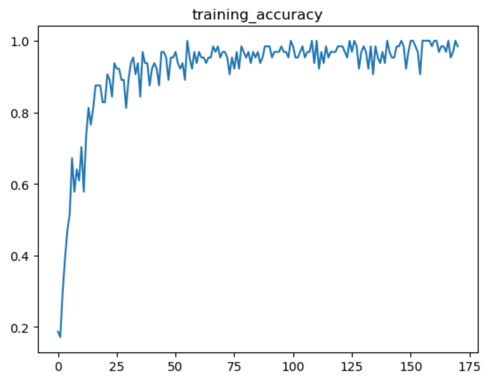


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

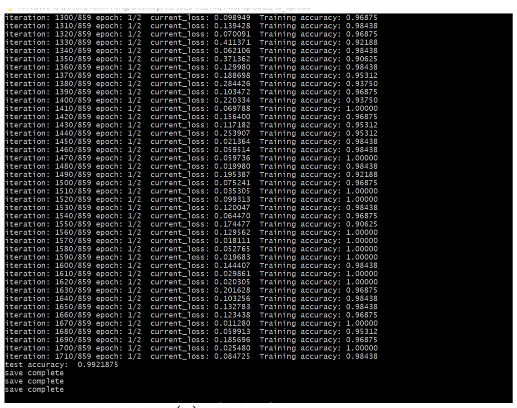
Figure 4: LSTM cell and equation. [1]



(a) loss

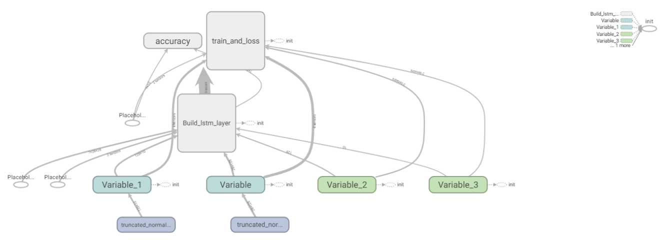


(b) accuracy



```
iteration: 1300/859 epoch: 1/2  current_loss: 0.098949  Training accuracy: 0.96875
iteration: 1310/859 epoch: 1/2  current_loss: 0.139428  Training accuracy: 0.98438
iteration: 1320/859 epoch: 1/2  current_loss: 0.070091  Training accuracy: 0.96875
iteration: 1330/859 epoch: 1/2  current_loss: 0.411371  Training accuracy: 0.92188
iteration: 1340/859 epoch: 1/2  current_loss: 0.062106  Training accuracy: 0.98438
iteration: 1350/859 epoch: 1/2  current_loss: 0.371362  Training accuracy: 0.90625
iteration: 1360/859 epoch: 1/2  current_loss: 0.129980  Training accuracy: 0.98438
iteration: 1370/859 epoch: 1/2  current_loss: 0.188698  Training accuracy: 0.95312
iteration: 1380/859 epoch: 1/2  current_loss: 0.284426  Training accuracy: 0.93750
iteration: 1390/859 epoch: 1/2  current_loss: 0.103472  Training accuracy: 0.96875
iteration: 1400/859 epoch: 1/2  current_loss: 0.220334  Training accuracy: 0.93750
iteration: 1410/859 epoch: 1/2  current_loss: 0.069788  Training accuracy: 1.00000
iteration: 1420/859 epoch: 1/2  current_loss: 0.156400  Training accuracy: 0.96875
iteration: 1430/859 epoch: 1/2  current_loss: 0.117182  Training accuracy: 0.95312
iteration: 1440/859 epoch: 1/2  current_loss: 0.253907  Training accuracy: 0.95312
iteration: 1450/859 epoch: 1/2  current_loss: 0.021364  Training accuracy: 0.98438
iteration: 1460/859 epoch: 1/2  current_loss: 0.059514  Training accuracy: 0.98438
iteration: 1470/859 epoch: 1/2  current_loss: 0.059736  Training accuracy: 1.00000
iteration: 1480/859 epoch: 1/2  current_loss: 0.019980  Training accuracy: 0.98438
iteration: 1490/859 epoch: 1/2  current_loss: 0.195387  Training accuracy: 0.92188
iteration: 1500/859 epoch: 1/2  current_loss: 0.075241  Training accuracy: 0.96875
iteration: 1510/859 epoch: 1/2  current_loss: 0.035305  Training accuracy: 1.00000
iteration: 1520/859 epoch: 1/2  current_loss: 0.099313  Training accuracy: 1.00000
iteration: 1530/859 epoch: 1/2  current_loss: 0.120047  Training accuracy: 0.98438
iteration: 1540/859 epoch: 1/2  current_loss: 0.064470  Training accuracy: 0.96875
iteration: 1550/859 epoch: 1/2  current_loss: 0.174477  Training accuracy: 0.90625
iteration: 1560/859 epoch: 1/2  current_loss: 0.129562  Training accuracy: 1.00000
iteration: 1570/859 epoch: 1/2  current_loss: 0.018111  Training accuracy: 1.00000
iteration: 1580/859 epoch: 1/2  current_loss: 0.052765  Training accuracy: 1.00000
iteration: 1590/859 epoch: 1/2  current_loss: 0.019683  Training accuracy: 1.00000
iteration: 1600/859 epoch: 1/2  current_loss: 0.144407  Training accuracy: 0.98438
iteration: 1610/859 epoch: 1/2  current_loss: 0.029861  Training accuracy: 1.00000
iteration: 1620/859 epoch: 1/2  current_loss: 0.020305  Training accuracy: 1.00000
iteration: 1630/859 epoch: 1/2  current_loss: 0.201628  Training accuracy: 0.96875
iteration: 1640/859 epoch: 1/2  current_loss: 0.103256  Training accuracy: 0.98438
iteration: 1650/859 epoch: 1/2  current_loss: 0.132783  Training accuracy: 0.98438
iteration: 1660/859 epoch: 1/2  current_loss: 0.123438  Training accuracy: 0.96875
iteration: 1670/859 epoch: 1/2  current_loss: 0.011280  Training accuracy: 1.00000
iteration: 1680/859 epoch: 1/2  current_loss: 0.059913  Training accuracy: 0.95312
iteration: 1690/859 epoch: 1/2  current_loss: 0.185696  Training accuracy: 0.96875
iteration: 1700/859 epoch: 1/2  current_loss: 0.025480  Training accuracy: 1.00000
iteration: 1710/859 epoch: 1/2  current_loss: 0.084725  Training accuracy: 0.98438
test accuracy:  0.9921875
save complete
save complete
save complete
```

(c) test accuracy



(d) tensorboard

Figure 5: LSTM result.