Project 1 MLP CNN and SNN

GIX 李心成 2016211089

Abstract

In this project, I learned Tensorflow framework, and implemented MLP, CNN and SNN with it on the MNIST data set. Finally, MLP with sigmoid activation function has the accuracy of 97.21%, while ReLU reach the 97.87%. CNN with traditional gradient descent algorithm has the accuracy of 99.33%, while Adam algorithm one reach the 99.41%. SNN has the accuracy of 90.03%, lower than the original MLP ReLU model.

1. Introduction

Project1 use the MNIST[1] handwriting digits data set, to train and evaluate three neuron network models MLP, CNN and SNN separately. MNIST is a classical data set for machine learning, which contains 60,000 train images and 10,000 test images of digital handwriting. Each one is 28×28 grey image with 10 unique labels from number 0 to 9.

MLP come from the early stage of neuron network. It simplified the neuron model to a liner aggregation of previous layer input and then a activation function is used. And the BP algorithm is used to learning the weight. CNN gain a huge success these days and have a massive amount of used in a huge area of tasks in both academic and industry. It use convolution and pooling instead of the liner method in MLP, which is inspired by the mechanism in visual area of human brain. SNN is thought as the third generation of neuron network, with more inspiration of the spiking model of neurons from human brain. SNN can be super energy efficient implemented on neuromorphic hardware, with a low accuracy cost.

In the following sections, I will introduce my method, result and discussion of three models I implemented.

2. Method

After some search on the Internet, I choose Tensorflow [2] framework to do this project, for the reasons listed below:

- Huge community. First I choose the language python other than the proprietary Matlab and the new comer Julia, mainly for its huge community and maturity of all the tools (Though I think Julia will be the next killer language in data Science and machine learning). And Tensorflow is the most popular among them nowadays.
- Supported by Google, and it targets the both side of academic and industry.

After some struggle of picking up python3 again and learning Tensorflow, all the method discussed below will be implemented in Tensorflow r0.11 version in python 3.5.1 environment (notice that some Tensorflow API have been changed in the newer version).

In the project folder, /summary contains the variable logs that can be visualized by Tensorboard[3], use command

```
tensorboard --logdir="./summary"
```

and you can see the interactive graph in your browser.

2.1. Preprocessing

At the preprocessing stage, the input images is one channel 28*28, so the read method reshape the image to a 1D vector at the shape [784], and a batch of images with the shape [BatchSize, 784]. And the imgaes will be normolized to float type at the interval $[0.0, 1.0]$. The labels was origin a scalar denote the label 0 to 9 of the image, read method reshape the label to one hot form with the shape [10], and a batch of labels with the shape [BatchSize, 10].

2.2. MLP

Project1 appointed the structrue of MLP should used, 784-64-128-10 network and trained 30 epochs.

The weight variables are initialized with normal distribution (stddev depends on the earch layer's units number), and bias variables are initialized with a constant slightly bigger than zero, named 0.1 in this case, for don't get two much zero gradient while training.

Two kinds of activation function have been implemented and compared, sigmoid and ReLU.

[1]MNIST `http://yann.lecun.com/exdb/mnist/`

[2]Tensorflow `https://www.tensorflow.org/`

[3]Tensorboard `https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/tensorboard/README.md`

For sigmoid function, learning rate is determined at 0.3, while ReLU is 0.1.

2.3. CNN

For CNN, I used ReLU activation function, trained 50 epochs at the batch size of 100. The variables are initialized almost the same as MLP. For convolution layer, I use the same convolution with stride 1 at each direction. For pooling layer, I use 2x2 average pooling. And 50% dropout are added after each pooling.

I tried two different structures, The first one is a complex one, with structure 28x28-32c5-2s-64c5-2s-1024-10o, two hidden layer with 32 and 64 feature maps, and two full connected layer with 1024 and 10 units. The second one is a simple one, with structure 28x28-12c5-2s-64c5-2s-10o.

I use traditional gradient descent algorithm for the first complex structure, but tried two different optimize algorithm for the second simple structure, gradient descent algorithm and Adam algorithm[4].

2.4. SNN

As for the SNN, I tune the MLP model with the ReLU activation function for training. I directly use the weights from the MLP, change the biases to zero for both MLP and SNN.

SNN share the similary structure with MLP, but with some additional part. At the begining after images input, there is a spike generator. And after each layer's ReLU activation function, there is a integrate-and-fire activation function, transferring the input membrane voltage to spike generation. And at the end of softmax layer, there is a spike counter.

Spike generator use the Poisson distributed generator. If

$$I > \lambda * UniformRandom() \quad (1)$$

which I denote for image input normalized to interval $[0.0, 1.0]$. Then a spike will be generated. I use 1.0 for parameter λ.

In integrate-and-fire activation function,

```
V(t) = V(t - 1) + L + X(t)
If V(t) >= V_th , spike and reset V(t) = 0
If V(t) < V min, reset V(t) = V_min
```

and I set 0.0 for `V_min`, 0.5 for `V_th` of each layer, 0.0 for `L`.

As for spike counter, it is used to count the spike in each output class for a period of time, named 100ms as I set, to determine the final classification of a picture.

[4] Adam algorithm `https://arxiv.org/abs/1412.6980`

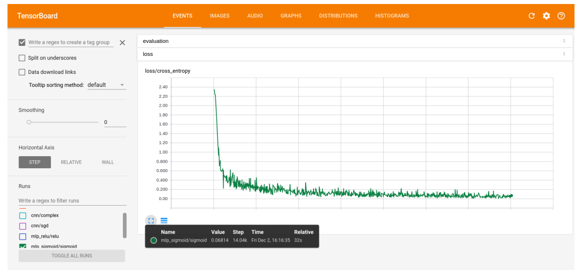


Figure 1. Train loss of MLP sigmoid

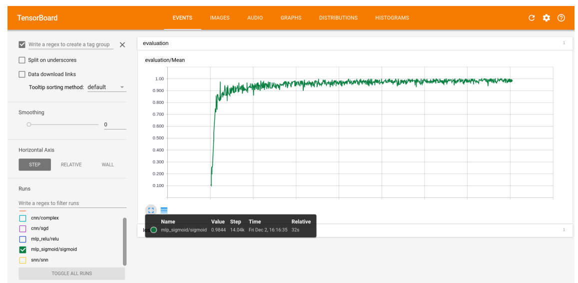
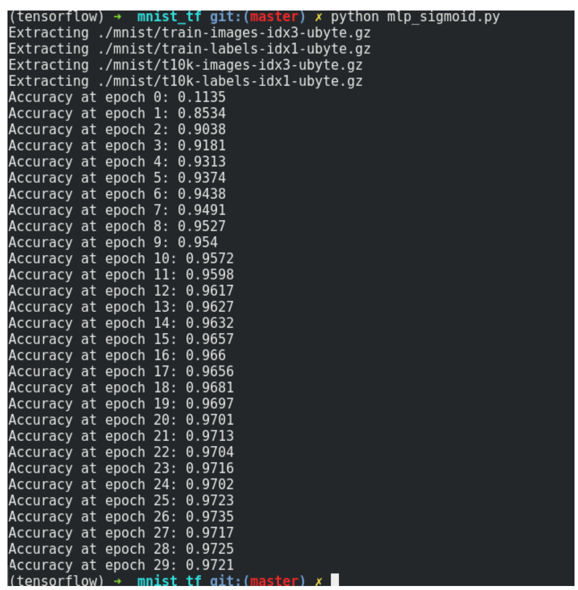


Figure 2. Train accuracy of MLP sigmoid



```
(tensorflow) → mnist_tf git:(master) ✗ python mlp_sigmoid.py
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
Accuracy at epoch 0: 0.1135
Accuracy at epoch 1: 0.8534
Accuracy at epoch 2: 0.9038
Accuracy at epoch 3: 0.9181
Accuracy at epoch 4: 0.9313
Accuracy at epoch 5: 0.9374
Accuracy at epoch 6: 0.9438
Accuracy at epoch 7: 0.9491
Accuracy at epoch 8: 0.9527
Accuracy at epoch 9: 0.954
Accuracy at epoch 10: 0.9572
Accuracy at epoch 11: 0.9598
Accuracy at epoch 12: 0.9617
Accuracy at epoch 13: 0.9627
Accuracy at epoch 14: 0.9632
Accuracy at epoch 15: 0.9657
Accuracy at epoch 16: 0.966
Accuracy at epoch 17: 0.9656
Accuracy at epoch 18: 0.9681
Accuracy at epoch 19: 0.9697
Accuracy at epoch 20: 0.9701
Accuracy at epoch 21: 0.9713
Accuracy at epoch 22: 0.9704
Accuracy at epoch 23: 0.9716
Accuracy at epoch 24: 0.9702
Accuracy at epoch 25: 0.9723
Accuracy at epoch 26: 0.9735
Accuracy at epoch 27: 0.9717
Accuracy at epoch 28: 0.9725
Accuracy at epoch 29: 0.9721
(tensorflow) → mnist_tf git:(master) ✗
```

Figure 3. Test accuracy of MLP sigmoid

3. Result

3.1. MLP

MLP with sigmoid activation. Figure 1 shows the train loss, figure 2 shows the train accuracy, figure 3 shows the test accuracy, with finally test accuracy 97.21%.

MLP with relu activation. Figure 4 shows the train loss, figure 5 shows the train accuracy, figure 6 shows the test ac-

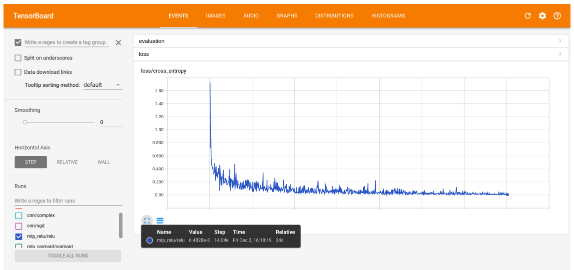


Figure 4. Train loss of MLP relu

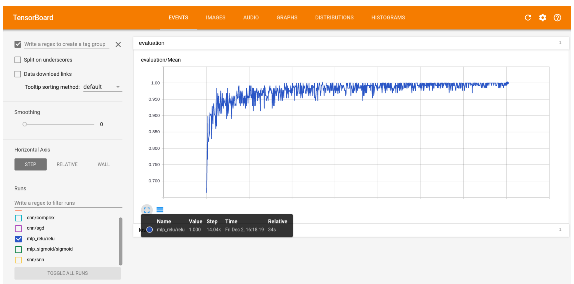
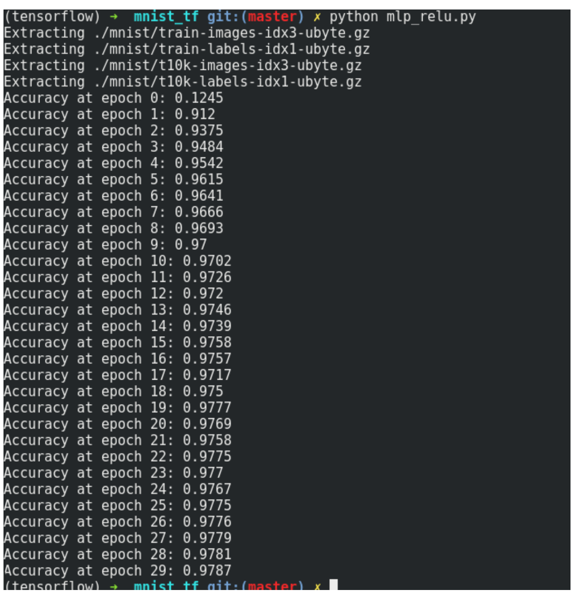


Figure 5. Train accuracy of MLP relu



```
(tensorflow) → mnist_tf git:(master) ✗ python mlp_relu.py
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
Accuracy at epoch 0: 0.1245
Accuracy at epoch 1: 0.912
Accuracy at epoch 2: 0.9375
Accuracy at epoch 3: 0.9484
Accuracy at epoch 4: 0.9542
Accuracy at epoch 5: 0.9615
Accuracy at epoch 6: 0.9641
Accuracy at epoch 7: 0.9666
Accuracy at epoch 8: 0.9693
Accuracy at epoch 9: 0.97
Accuracy at epoch 10: 0.9702
Accuracy at epoch 11: 0.9726
Accuracy at epoch 12: 0.972
Accuracy at epoch 13: 0.9746
Accuracy at epoch 14: 0.9739
Accuracy at epoch 15: 0.9758
Accuracy at epoch 16: 0.9757
Accuracy at epoch 17: 0.9717
Accuracy at epoch 18: 0.975
Accuracy at epoch 19: 0.9777
Accuracy at epoch 20: 0.9769
Accuracy at epoch 21: 0.9758
Accuracy at epoch 22: 0.9775
Accuracy at epoch 23: 0.977
Accuracy at epoch 24: 0.9767
Accuracy at epoch 25: 0.9775
Accuracy at epoch 26: 0.9776
Accuracy at epoch 27: 0.9779
Accuracy at epoch 28: 0.9781
Accuracy at epoch 29: 0.9787
(tensorflow) → mnist_tf git:(master) ✗
```

Figure 6. Test accuracy of MLP relu

curacy, with finally test accuracy 97.87%, higher than sigmoid with lesser training time.

3.2. CNN

CNN with complex structure. Figure 7 shows the train loss, figure 8 shows the train accuracy. Just notice the training time, it is too long with less accuracy than the simple structure, so I give it up at early stage.

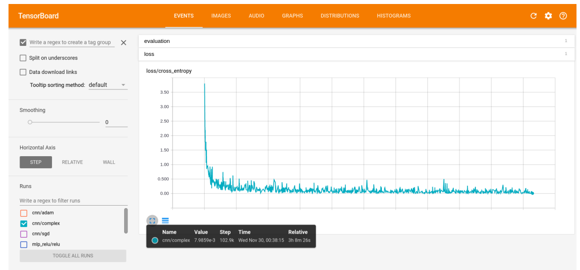


Figure 7. Train loss of CNN complex

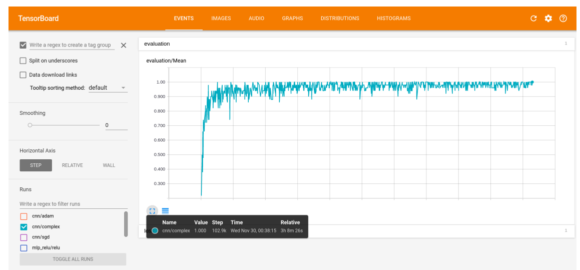


Figure 8. Train accuracy of CNN complex

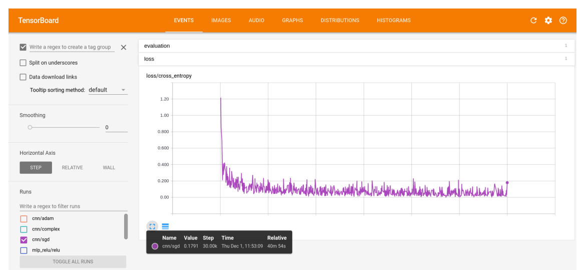


Figure 9. Train loss of CNN gradient descent algorithm

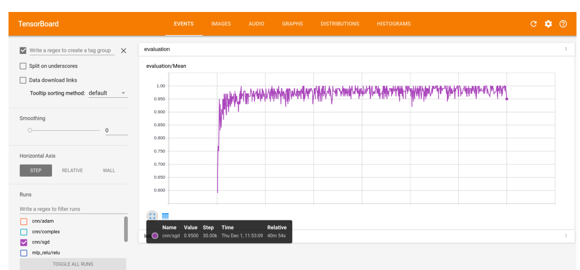


Figure 10. Train accuracy of CNN gradient descent algorithm

CNN with simple structure and gradient descent algorithm. Figure 9 shows the train loss, figure 10 shows the train accuracy, figure 11 shows the test accuracy, with finally test accuracy 99.33%. Notice the training time is only 40 minutes compared to the 3 hours of the complex one.

CNN with simple structure and gradient descent algorithm. Figure 12 shows the train loss, figure 13 shows the

```
(tensorflow) → mnist_tf git:(master) ✗ python cnn.py
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
Accuracy at epoch 0: 0.1582
Accuracy at epoch 1: 0.9579
Accuracy at epoch 2: 0.9712
Accuracy at epoch 3: 0.9747
Accuracy at epoch 4: 0.9794
Accuracy at epoch 5: 0.9814
Accuracy at epoch 6: 0.9819
Accuracy at epoch 7: 0.9842
Accuracy at epoch 8: 0.9848
Accuracy at epoch 9: 0.9851
Accuracy at epoch 10: 0.9865
Accuracy at epoch 11: 0.9874
Accuracy at epoch 12: 0.9875
Accuracy at epoch 13: 0.9882
Accuracy at epoch 14: 0.9885
Accuracy at epoch 15: 0.9894
Accuracy at epoch 16: 0.9891
Accuracy at epoch 17: 0.9899
Accuracy at epoch 18: 0.9904
Accuracy at epoch 19: 0.9898
Accuracy at epoch 20: 0.9902
Accuracy at epoch 21: 0.9915
Accuracy at epoch 22: 0.9909
Accuracy at epoch 23: 0.9915
Accuracy at epoch 24: 0.9916
Accuracy at epoch 25: 0.9923
Accuracy at epoch 26: 0.9923
Accuracy at epoch 27: 0.9922
Accuracy at epoch 28: 0.9929
Accuracy at epoch 29: 0.9918
Accuracy at epoch 30: 0.9922
Accuracy at epoch 31: 0.9916
Accuracy at epoch 32: 0.9927
Accuracy at epoch 33: 0.9925
Accuracy at epoch 34: 0.9918
Accuracy at epoch 35: 0.9926
Accuracy at epoch 36: 0.9923
Accuracy at epoch 37: 0.9925
Accuracy at epoch 38: 0.9928
Accuracy at epoch 39: 0.9923
Accuracy at epoch 40: 0.9927
Accuracy at epoch 41: 0.9924
Accuracy at epoch 42: 0.9933
Accuracy at epoch 43: 0.9933
Accuracy at epoch 44: 0.9929
Accuracy at epoch 45: 0.993
Accuracy at epoch 46: 0.9924
Accuracy at epoch 47: 0.9925
Accuracy at epoch 48: 0.9924
Accuracy at epoch 49: 0.9933
```

Figure 11. Test accuracy of CNN gradient descent algorithm

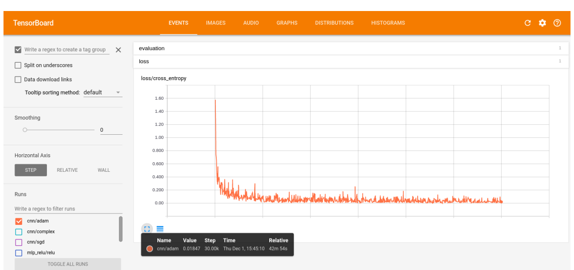


Figure 12. Train loss of CNN Adam algorithm

train accuracy, figure 14 shows the test accuracy, with finally test accuracy 99.41%, a little higher than traditional gradient descent algorithm.

3.3. SNN

SNN figures. Figure 15 shows the train loss, figure 16 shows the train accuracy, almost the same as MLP with ReLU activation function. Figure 17 shows the test accuracy, with finally test accuracy 90.03%, lower than original MLP solution.

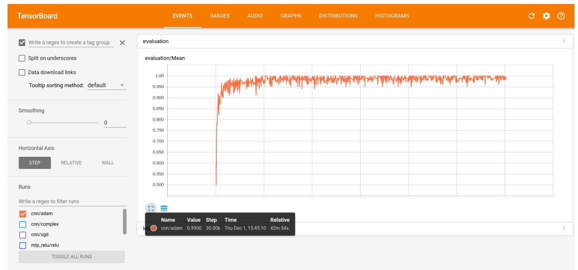


Figure 13. Train accuracy of CNN Adam algorithm

```
(tensorflow) → mnist_tf git:(master) ✗ python cnn.py
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
Accuracy at epoch 0: 0.0993
Accuracy at epoch 1: 0.9578
Accuracy at epoch 2: 0.9761
Accuracy at epoch 3: 0.9797
Accuracy at epoch 4: 0.9839
Accuracy at epoch 5: 0.9865
Accuracy at epoch 6: 0.9884
Accuracy at epoch 7: 0.9892
Accuracy at epoch 8: 0.9898
Accuracy at epoch 9: 0.99
Accuracy at epoch 10: 0.9905
Accuracy at epoch 11: 0.9905
Accuracy at epoch 12: 0.9913
Accuracy at epoch 13: 0.9911
Accuracy at epoch 14: 0.9916
Accuracy at epoch 15: 0.9923
Accuracy at epoch 16: 0.9931
Accuracy at epoch 17: 0.9922
Accuracy at epoch 18: 0.992
Accuracy at epoch 19: 0.9924
Accuracy at epoch 20: 0.9921
Accuracy at epoch 21: 0.9928
Accuracy at epoch 22: 0.9931
Accuracy at epoch 23: 0.9935
Accuracy at epoch 24: 0.993
Accuracy at epoch 25: 0.9932
Accuracy at epoch 26: 0.993
Accuracy at epoch 27: 0.9939
Accuracy at epoch 28: 0.994
Accuracy at epoch 29: 0.9942
Accuracy at epoch 30: 0.9925
Accuracy at epoch 31: 0.9936
Accuracy at epoch 32: 0.9939
Accuracy at epoch 33: 0.994
Accuracy at epoch 34: 0.9935
Accuracy at epoch 35: 0.9935
Accuracy at epoch 36: 0.994
Accuracy at epoch 37: 0.9936
Accuracy at epoch 38: 0.9929
Accuracy at epoch 39: 0.9936
Accuracy at epoch 40: 0.9939
Accuracy at epoch 41: 0.9941
Accuracy at epoch 42: 0.9933
Accuracy at epoch 43: 0.9944
Accuracy at epoch 44: 0.9936
Accuracy at epoch 45: 0.9937
Accuracy at epoch 46: 0.9935
Accuracy at epoch 47: 0.9943
Accuracy at epoch 48: 0.9941
Accuracy at epoch 49: 0.9941
(tensorflow) → mnist_tf git:(master) ✗
```

Figure 14. Test accuracy of CNN Adam algorithm

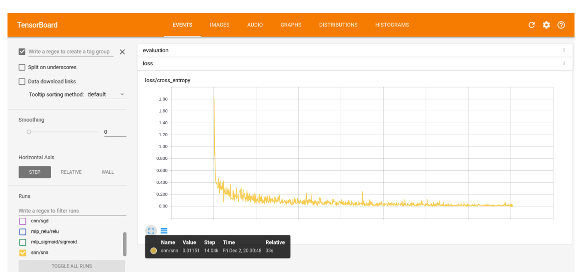


Figure 15. Train loss of SNN

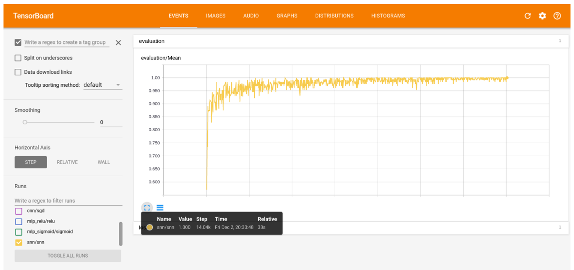
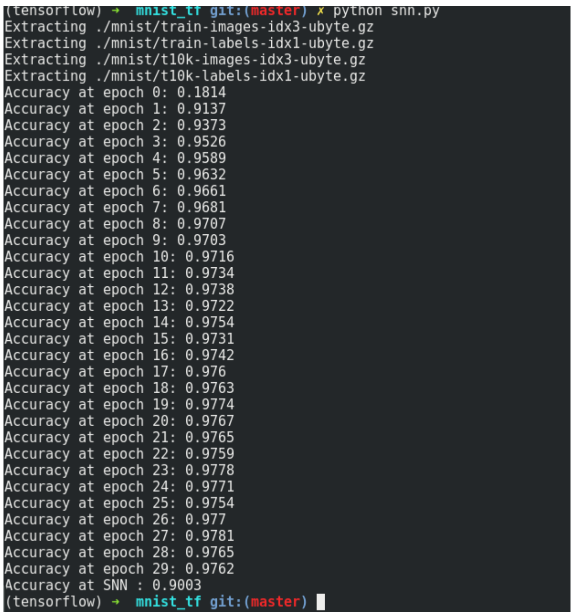


Figure 16. Train accuracy of SNN



```
(tensorflow) → mnist_tf git:(master) ✗ python snn.py
Extracting ./mnist/train-images-idx3-ubyte.gz
Extracting ./mnist/train-labels-idx1-ubyte.gz
Extracting ./mnist/t10k-images-idx3-ubyte.gz
Extracting ./mnist/t10k-labels-idx1-ubyte.gz
Accuracy at epoch 0: 0.1814
Accuracy at epoch 1: 0.9137
Accuracy at epoch 2: 0.9373
Accuracy at epoch 3: 0.9526
Accuracy at epoch 4: 0.9589
Accuracy at epoch 5: 0.9632
Accuracy at epoch 6: 0.9661
Accuracy at epoch 7: 0.9681
Accuracy at epoch 8: 0.9707
Accuracy at epoch 9: 0.9703
Accuracy at epoch 10: 0.9716
Accuracy at epoch 11: 0.9734
Accuracy at epoch 12: 0.9738
Accuracy at epoch 13: 0.9722
Accuracy at epoch 14: 0.9754
Accuracy at epoch 15: 0.9731
Accuracy at epoch 16: 0.9742
Accuracy at epoch 17: 0.976
Accuracy at epoch 18: 0.9763
Accuracy at epoch 19: 0.9774
Accuracy at epoch 20: 0.9767
Accuracy at epoch 21: 0.9765
Accuracy at epoch 22: 0.9759
Accuracy at epoch 23: 0.9778
Accuracy at epoch 24: 0.9771
Accuracy at epoch 25: 0.9754
Accuracy at epoch 26: 0.977
Accuracy at epoch 27: 0.9781
Accuracy at epoch 28: 0.9765
Accuracy at epoch 29: 0.9762
Accuracy at SNN : 0.9003
(tensorflow) → mnist_tf git:(master)
```

Figure 17. Test accuracy of SNN

4. Discussion

4.1. MLP

The weight variables are initialized with normal distribution and bias variables are initialized with a constant slightly bigger than zero is effective for not get two much zero gradient while training.

After some trial, learning rate is determined at 0.3 for sigmoid, while ReLU is 0.1. If learning rate is larger than that, such as 1.0 I've tried, then the loss will go down very fast at first, but will easily overfit after several epochs, which cause a low accuracy finally. And if learning rate is smaller than that, loss will go down too slow and not sufficiently learned. Because of the ReLU will converge faster than sigmoid, so the learning rate is little smaller.

4.2. CNN

Add 50% dropout after each pooling layer lower the training time while decrease the overfit chances tremendously.

As the result show, the complex structure have a much logner training time, but with limited gain. As I am considered, it is mainly caused by the simplicity of the MNIST data set. Digital handwriting image has relatively less features than normal colorful daily images, so simple structure is enough for the task.

Adam algorithm add some advanced features to traditional gradient descent algorithm, such as stochastic descent, adaptive estimates of lower-order momentum. Which means it have a bigger step at the beginning and lower step at the end. It converges faster than traditional gradient descent but have higher accuracy finally.

4.3. SNN

To achieve the minimal accuracy loss, there are some tricks.

Use the ReLU as the activation function, because ReLU can prevent the merge of negative number.

Set the biases to zero so that the spike can be controlled.

Choose the right threshold voltage of each layer. This is the most critical part to have a decent loss of accuracy. When threshold is too high, too little spikes are generated. When threshold is too low, too many spikes are generated. Both will cause SNN have no ability to distinguish the images. Or otherwise you can do the weight normalization, which may gain a better result than manually trial.