Report on Project 1

Zhunxuan Wang, 13300180086
School of Mathematical Sciences

October 31, 2017

1 Fully Connected NN

1.1 Model Description

We decided to apply the simplest fully connected structure (shown below) on `MNIST` dataset

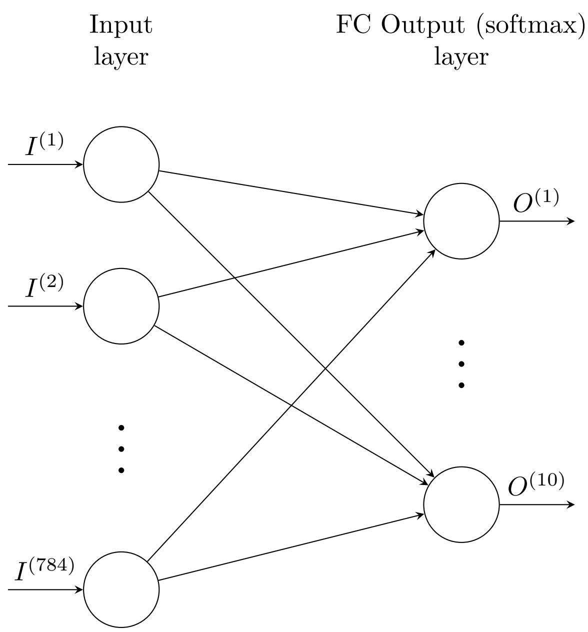


Figure 1: The Fully Connected Neural Network

where the input vector is the vector-form (size: 1×784) of the input image (size: 28×28), and the output vector is the prediction 10-D vector. The edges between input layer and the output layer are the transition weight matrix $\mathbf{W}$ (size: 784×10). The mathematical model of the structure would be

$$\hat{\mathbf{Y}} = \boldsymbol{\sigma}\left(\mathbf{X}\mathbf{W} + \begin{bmatrix} \mathbf{b} \\ \vdots \\ \mathbf{b} \end{bmatrix}\right),$$

and we take the cross-entropy loss function

$$L\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) = -\frac{1}{N} \sum_{n \in N} \sum_{i=1}^{10} Y_{n,i} \log \hat{Y}_{n,i}.$$

Applying the backpropagation [1] based on chain rules we obtained

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{W}} \frac{\partial L}{\partial \hat{\mathbf{Y}}} = \mathbf{X}^{\intercal} \left(\hat{\mathbf{Y}} - \mathbf{Y} \right),$$

and for the bias term

$$\frac{\partial L}{\partial \mathbf{b}} = \sum_{i \in N} \left(\hat{\mathbf{Y}} - \mathbf{Y} \right)_i$$

where $\left(\hat{\mathbf{Y}} - \mathbf{Y} \right)_i$ is the i-th row of $\hat{\mathbf{Y}} - \mathbf{Y}$.

Therefore each training step could be presented as follows

$$\mathbf{W}_{\text{new}} = \mathbf{W}_{\text{prev}} - \eta \frac{\partial L \left(\mathbf{Y}, \hat{\mathbf{Y}} \right)}{\partial \mathbf{W}}, \; \mathbf{b}_{\text{new}} = \mathbf{b}_{\text{prev}} - \eta \frac{\partial L \left(\mathbf{Y}, \hat{\mathbf{Y}} \right)}{\partial \mathbf{b}}$$

where η is the learning rate.

1.2 Implementation by `numpy`

Implementing the model by `numpy`, and setting the max training steps 10000, batch size 100, and learning rate 10^{-4}, we have the iteration running result as the following picture

```
##############################
compute_accuracy: 0.064
cross_entropy: 5.97073653153
##############################
compute_accuracy: 0.53
cross_entropy: 1.04519667206
##############################
compute_accuracy: 0.692
cross_entropy: 0.732215550933
##############################
compute_accuracy: 0.766
cross_entropy: 0.67645556985
##############################
compute_accuracy: 0.802
cross_entropy: 0.766021688845
##############################
compute_accuracy: 0.82
cross_entropy: 0.643111582946
##############################
compute_accuracy: 0.836
cross_entropy: 0.535931146756
##############################
compute_accuracy: 0.84
cross_entropy: 0.589823353555
##############################
compute_accuracy: 0.848
cross_entropy: 0.467187261477
##############################
compute_accuracy: 0.858
cross_entropy: 0.482776806177
##############################
```

```
##############################
compute_accuracy: 0.86
cross_entropy: 0.310175887699
##############################
compute_accuracy: 0.868
cross_entropy: 0.417884012214
##############################
compute_accuracy: 0.87
cross_entropy: 0.356961130137
##############################
compute_accuracy: 0.87
cross_entropy: 0.567378367483
##############################
compute_accuracy: 0.872
cross_entropy: 0.44578480217
##############################
compute_accuracy: 0.878
cross_entropy: 0.374736892225
##############################
compute_accuracy: 0.882
cross_entropy: 0.325100238522
##############################
compute_accuracy: 0.884
cross_entropy: 0.375473098967
##############################
compute_accuracy: 0.886
cross_entropy: 0.327948174319
##############################
compute_accuracy: 0.886
cross_entropy: 0.272131249445
```

Figure 2: The Iteration Result by `numpy`

and we have the accuracy plot and the loss plot against epochs for training and testing

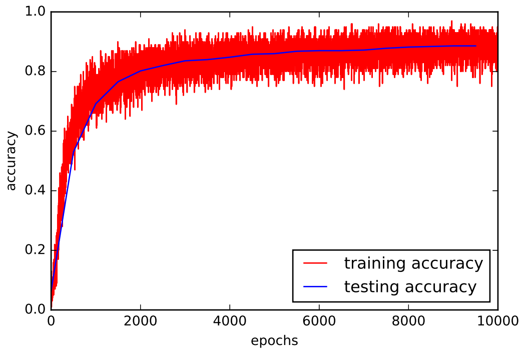


Figure 3: The Accuracy Plot against Epochs

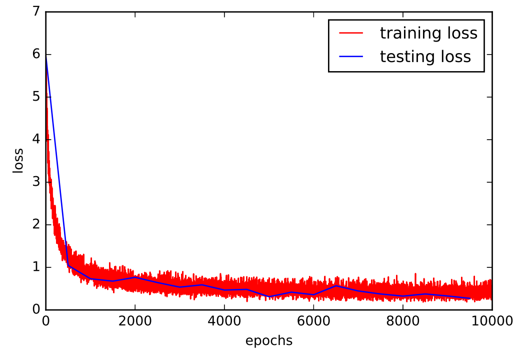
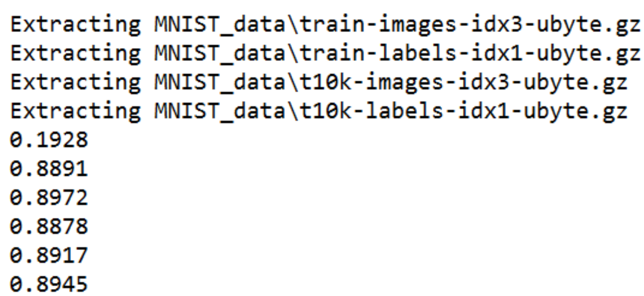


Figure 4: The Loss Plot against Epochs

As we observe from the iteration results and the plots, the accuracies and the losses of both training and testing show a sharp increasing and decreasing trend respectively in the prophase of the training process (the number of epochs is less than around 1500), after which the trends of both slow down. For the training accuracy and loss, they show sharp oscillations during the whole process. The final testing accuracy is 0.886, of which the cross entropy loss is 0.272131.

1.3 Implementation by `tensorflow`

We take the softmax cross entropy as the loss function (the same as the loss function in the `numpy` model), setting the max epochs to 3000, the learning rate to 0.1, and the batch size to 100, using the adam optimizer, we have the running result as the picture shows below



```
Extracting MNIST_data\train-images-idx3-ubyte.gz
Extracting MNIST_data\train-labels-idx1-ubyte.gz
Extracting MNIST_data\t10k-images-idx3-ubyte.gz
Extracting MNIST_data\t10k-labels-idx1-ubyte.gz
0.1928
0.8891
0.8972
0.8878
0.8917
0.8945
```

Figure 5: The Iteration Result by `tensorflow`

and we have the accuracy plot and the loss plot against epochs for training and testing

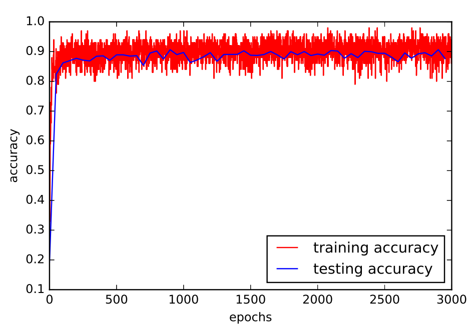


Figure 6: The Accuracy Plot against Epochs

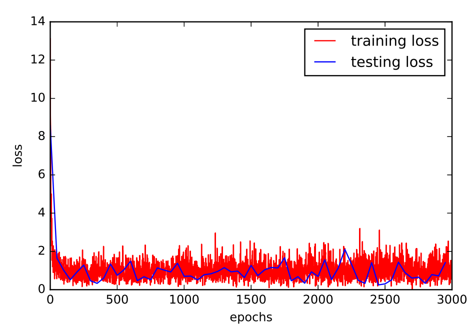
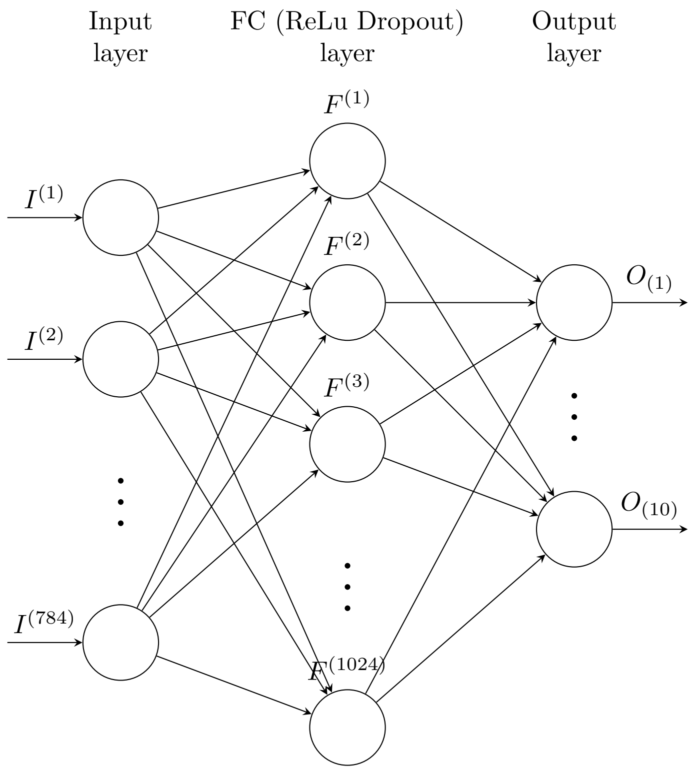


Figure 7: The Loss Plot against Epochs

As we observe from the running result picture and the plots, the convergence speed is much faster than the `numpy` based model (probably because the adam optimizer is much more efficient than normal gradient descent method). This model shows a steady convergence state as the number of epochs reaches 100. The final testing accuracy is 0.8945, of which the cross entropy loss is 0.30369076.

As we observe from the result above, this model's capability has reached its limit as those two methods above showed a convergent state without overfitting. In order to optimize the model, we decide to adjust the structure, inserting a hidden layer (with 1024 neurons) between the appeared two layers



where the hidden layer has activation function `ReLu` and a dropout layer with keep probability 0.9. The training result is better than the previous model. And the final testing accuracy will reach around 0.92, which is a little bit greater than the model without a hidden layer. The running result is shown as follows

```
0.9115
0.8989
0.9049
0.9121
0.9114
0.9099
0.9203
0.9153
0.911
0.9216
```

Figure 8: The Iteration Result with a hidden layer by `tensorflow`

2 Conclusion

Both `numpy` and `tensorflow` based models show convergent trends (accuracy around 0.9) with the training processes pass, and they are both not overfitting. The optimizer of the loss function is the key to the efficiency of the training process.

By adjusting the structure of the network, we can also learn that the complexity of a model is a key point to improve the fitting performance. Sometimes, a network with more complex structure has a stronger capability to fit large-scale data.

References

[1] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.