Implementation of A Fully connected Neural Network Using Numpy

Haocong Wang

mw814@scarletmail.rutgers.edu

April 8, 2021

Abstract

In this project, we design and implement a fully connected neural network using numpy. The network we finished is a four-layer network. All the codes are finished on PyCharm with Python 3.7.8. We choose the MINST dataset to train and test our model. Finally, the result shows that our model can reach the accuracy above 90%, which fulfills the requirement.

I. Introduction

A neural network is a network or circuit of neurons, or in a modern sense, an artificial neural network, composed of artificial neurons or nodes, as shown in figure 1. Thus a neural network is either a biological neural network, made up of real biological neurons, or an artificial neural network, for solving artificial intelligence (AI) problems [4]. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output.

Fully connected neural networks (FCNNs) are a type of artificial neural network where the architecture is such that all the nodes, or neurons, in one layer are connected to the neurons in the next layer [2], as shown in figure 2.

FCNNs have great potential in image recognition. In image recognition, digits recognition is one of the basic applications. So in this project, we design and implement a $784 \times 200 \times 10 \times 1$ network using the MINST dataset, which contains a large number of handwrit-

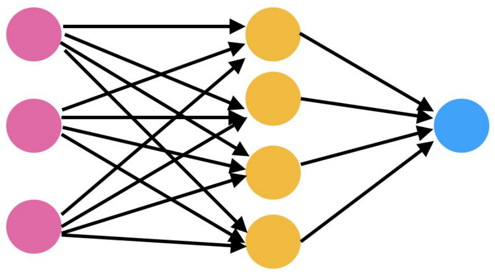


Figure 1: *A simple neural network.*

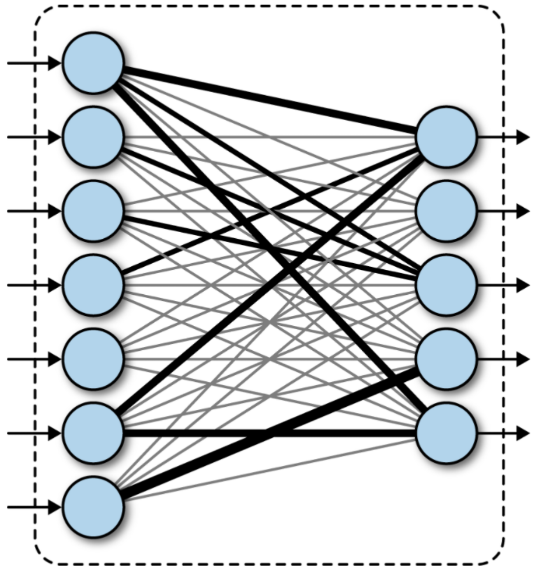


Figure 2: *A fully connected neural network.*

ten digits from 0-9 that are commonly used for training various image processing systems. The result shows that the accuracy can reach above 90% after the training process is completed.

II. Related Work

i. Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function [3]. The idea is to take repeated steps in the opposite direction of the gradient (or approximate gradient) of the function at the current point, because this is the direction of steepest descent, as shown in figure 3.

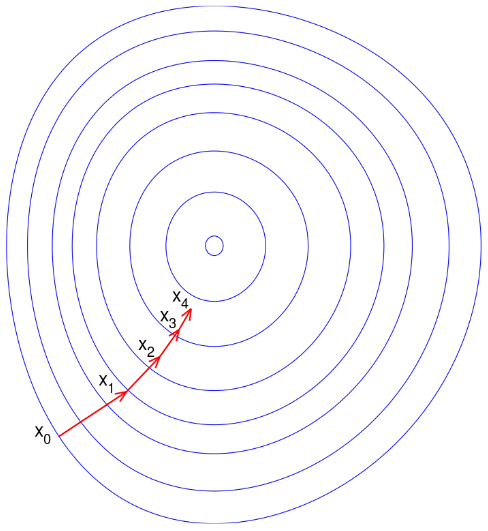


Figure 3: *Gradient Descent.*

ii. Backpropagation

In machine learning, backpropagation (BP) is a widely used algorithm for training feed-forward neural networks [9]. In fitting a neural network, backpropagation computes the gradient of the loss function with respect to the weights of the network for a single inputoutput example, and does so efficiently, unlike a naive direct computation of the gradient with respect to each weight individually.

III. Description of Dataset, Method and Model

In this section, we will discuss the details of the dataset, method and model used in our project.

i. Dataset

The dataset we are using is MNIST [8]. This is a large dataset of handwritten digits from 0-9 that are commonly used for training various image processing systems. All these numbers

Figure 4: *An image from the MNIST dataset.*

are from 250 different people, 50% of who are middle school students, and another 50% of who are faculty of the Census Bureau. The dataset itself is divided into a training set that has 60000 samples and a testing set with 10000 samples. Each image in this dataset has 28×28 pixels. So in the training process, we flat each image into a 784 dimension vector. The examples are shown in figure 4.

ii. Method

In this section, we provide detailed introduction for all the mathematical methods used in our model, which includes ReLu Function, Softmax Function and Loss Function.

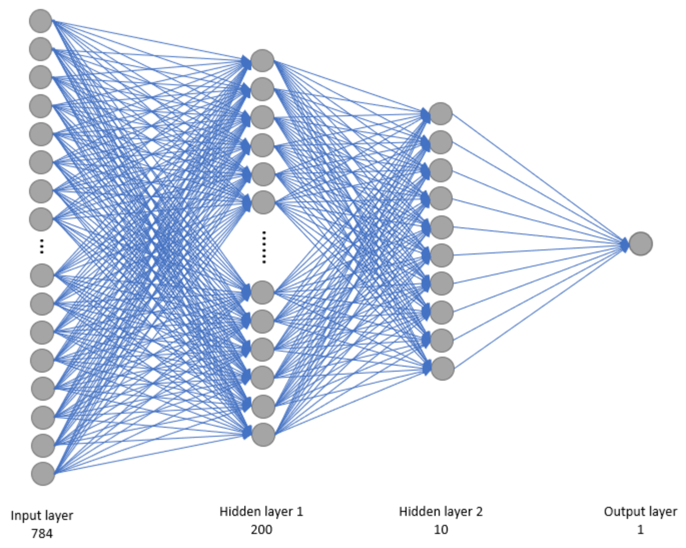


Figure 5: *The review of our network.*

ii.1 ReLu Function

For the ReLU Function, we decide use the Ramp function, which is a widely used rectifier in the machine learning field [5]. As its name showed, the graph of ramp function looks like a ramp. And can be described mathematically as follow:

$$R(x) := max(0, x) \tag{1}$$

The backward computation of this function can be seen as:

$$\frac{dr_i}{dx_i} = \begin{cases} 0, x_i \leq 0 \\ 1, x_i > 0 \end{cases} \tag{2}$$

ii.2 Softmax Function

The Softmax function [6] takes input as a vector of k real numbers and normalizes it into a probability distribution consisting of K probabilities proportional to the exponential of the input numbers. And it can be mathematically defined as follow:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=i}^{K} e^{z_i}} \tag{3}$$

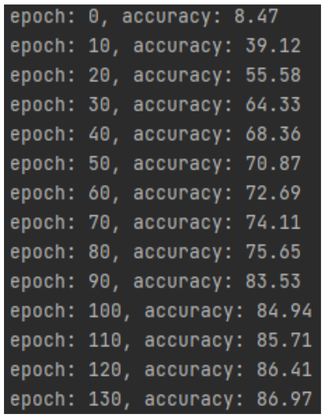
ii.3 Loss Function

We decide to use cross-entropy as our loss function. The cross-entropy [1] between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p.

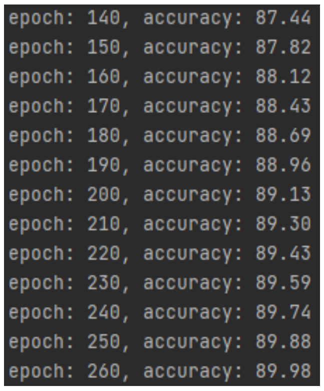
$$H(p, q) = -\sum_{i=1}^{n} p(x_i) log(q(x_i))$$

iii. Model

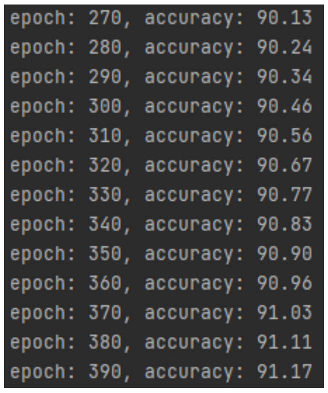
As mentioned in Section I, we build a four-layer fully connected neural network, as shown in figure 5. Considering the image data that will be taken as input, the first layer has 784 neurons and will output 200 neurons. The two hidden layers have 200 and 10 dimensions, respectively.



```
epoch: 0, accuracy: 8.47
epoch: 10, accuracy: 39.12
epoch: 20, accuracy: 55.58
epoch: 30, accuracy: 64.33
epoch: 40, accuracy: 68.36
epoch: 50, accuracy: 70.87
epoch: 60, accuracy: 72.69
epoch: 70, accuracy: 74.11
epoch: 80, accuracy: 75.65
epoch: 90, accuracy: 83.53
epoch: 100, accuracy: 84.94
epoch: 110, accuracy: 85.71
epoch: 120, accuracy: 86.41
epoch: 130, accuracy: 86.97
```



```
epoch: 140, accuracy: 87.44
epoch: 150, accuracy: 87.82
epoch: 160, accuracy: 88.12
epoch: 170, accuracy: 88.43
epoch: 180, accuracy: 88.69
epoch: 190, accuracy: 88.96
epoch: 200, accuracy: 89.13
epoch: 210, accuracy: 89.30
epoch: 220, accuracy: 89.43
epoch: 230, accuracy: 89.59
epoch: 240, accuracy: 89.74
epoch: 250, accuracy: 89.88
epoch: 260, accuracy: 89.98
```



```
epoch: 270, accuracy: 90.13
epoch: 280, accuracy: 90.24
epoch: 290, accuracy: 90.34
epoch: 300, accuracy: 90.46
epoch: 310, accuracy: 90.56
epoch: 320, accuracy: 90.67
epoch: 330, accuracy: 90.77
epoch: 340, accuracy: 90.83
epoch: 350, accuracy: 90.90
epoch: 360, accuracy: 90.96
epoch: 370, accuracy: 91.03
epoch: 380, accuracy: 91.11
epoch: 390, accuracy: 91.17
```

Figure 6: *Epochs of training.*

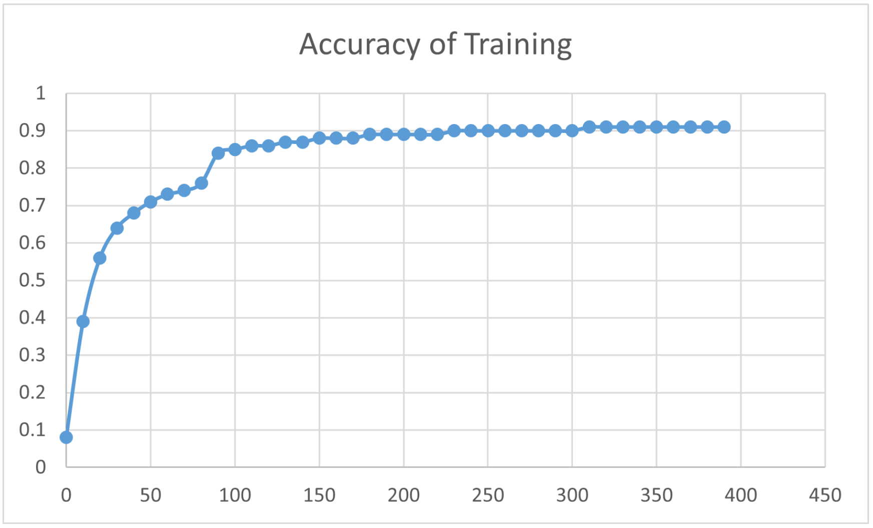
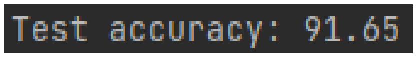


Figure 7: *The trend of accuracy.*

IV. Experimental Procedure and Results

We use numpy to build the model on PyCharm. We first set the epochs of training to be 1000, but then we found that after the accuracy reaches 80%, the speed of increasing accuracy starts to slow down. Therefore, we finally choose 400 as the epochs of training, as shown in figure 6.

After 400 times of training, we test the trained model on the testing dataset extracted from the MNIST dataset. The accuracy is finally stabilized at around 90%, as shown in figure 8. We also show the trend of the accuracy during training in figure 7. As can be seen from the figure, the accuracy reach above 90% at around 200 epochs. There is another point in our project. Since we initiate the weights and bias randomly, the process might be much slower if we are unfortunate to initiate a series of bad weights.



```
Test accuracy: 91.65
```

Figure 8: *The final accuracy.*

V. Conclusion

In this project, we design and implement a four-layer fully connected neural network with numpy. We choose MNIST as our dataset and finish 400 epochs of training. The results show that the accuracy can reach above 90%.

References

[1] Cross-entropy function, 2021. https://en.wikipedia.org/wiki/Cross_entropy.

[2] Fully connected neural network, 2021. https://radiopaedia.org/articles/fully-connected-neural-network.

[3] Gradient descent, 2021. https://en.wikipedia.org/wiki/Gradient_descent.

[4] Neural network, 2021. https://en.wikipedia.org/wiki/Neural_network.

[5] Relu function, 2021. https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[6] Softmax function, 2021. https://en.wikipedia.org/wiki/Softmax_function.

[7] E. Deng, Y. Zhang, W. Kang, B. Dieny, J.-O. Klein, G. Prenat, and W. Zhao. Synchronous 8-bit non-volatile full-adder based on spin transfer torque magnetic tunnel junction. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(7):1757–1765, 2015.

[8] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[9] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28, 1988.