Handwritten Digit Classification

Cheng Jin

jin.ch@husky.neu.edu

ABSTRACT

In this project, we applied the multi-layer perceptron models to train the Mnist image data. The dataset is labeled by 0-9, and according to different digit showed in the image. Then used the trained models to predict the digit in the image. In the results and conclusion part, we made a compare of different models and tried to explain the prediction results.

Introduction:

Image recognition is important for many advanced technologies today, most smart phones today also come with recognition app that convert handwriting into typed words. In our project, we will investigate the multi-layers perceptron model, this model has 3 or more layers (1 input layer, one or more hidden layers and 1 output layer) structure. We will be using the famous MNIST (Mixed National Institute of Standards and Technology) dataset.

Dataset description:

MNIST is a popular handwritten digit database used extensively in machine learning, computer vision communities.

We had 10000 handwritten digits images, together with the lable digit they represent. The resolution of the gray scale images is 28\*28(which indicates the input layer size is 28\*28), the output layer size is 10.

We got the 10000 images dataset in this website: <http://yann.lecun.com/exdb/mnist/>

Data processing:

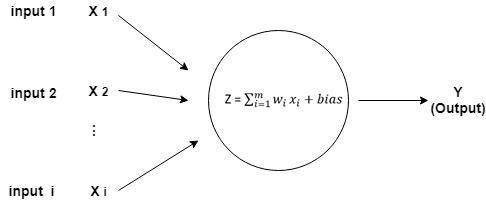
We had *MnistReader.java*, and *MnistFile.java* to process the dataset.

The neural network:

Single perceptron:

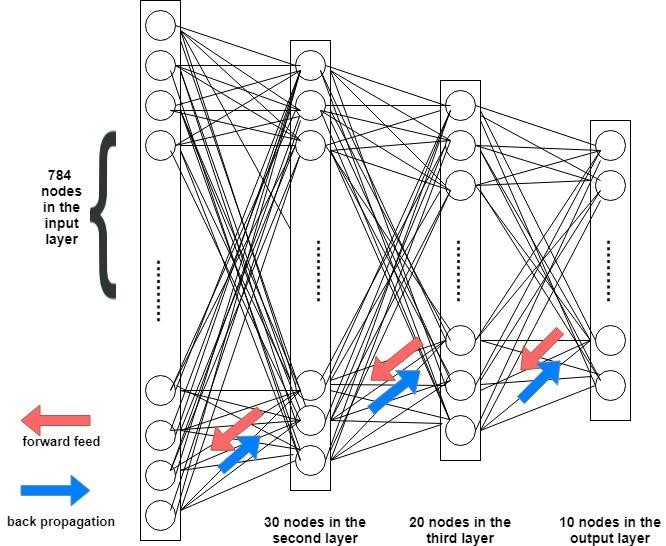
The way we calculate the input data X:𝑥1, 𝑥2, …, 𝑥𝑚 in the perceptron, we had the weight set W: 𝑤1, 𝑤2, …, 𝑤𝑚 , note that the weights of each layer’s neuron will be adjusted when deploying the back propagation. Here is how we get the output result Z:

# Z = 𝑤𝑖 𝑥𝑖 + 𝑏𝑖𝑎𝑠



Multi-Layer perceptron:

In this project the Multi-Layer Perceptron is a four-layer architect, the first layer perceptron has 784 nodes/neurons, the second layer perceptron has 30 nodes and the third layer has 20 nodes and the last layer has 10 nodes.



*Forward Feeding method*: the individual nodes output is calculated as sigmoid function. Output = weights\*input\_previous\_layer + node\_bias, the calculation tools are defined in the Help.java file. In the end of calculation, we need to convert the output of the final layer back into a single dimensional array.

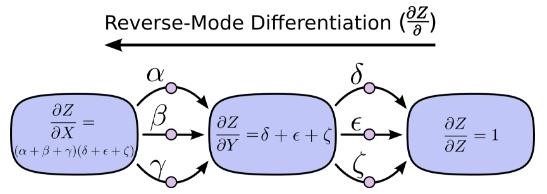
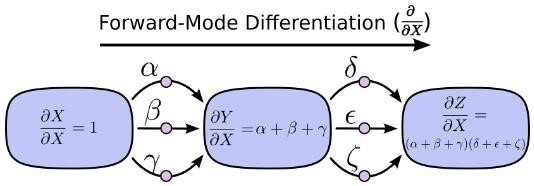
*Back propagation method*: There are two ways of computing the differentiation, the forward-mode differentiation, and the reverse-mode differentiation. The forward-mode differentiation is very similar to what we learned in the calculus class. The forward-mode differentiation is to track how the input affects every

∂ node, it applies to every node. But the reverse-mode differentiation is to track

## ∂X

∂Z how every node affects one output, it applies to every node.

∂



*Activation method*: We use the sigmoid function as the activation function. three Sigmoid functions with different input (integer, one dimensional array, two- dimensional array)

*Gradient class*: define a two-dimensional array biases, and three-dimensional array weights.

SGD (stochastic gradient descent) *method*: We wrote two SGD functions, the first SGD method is to calculate the correct accuracy after every epoch on training data, we use the evaluate method to calculate the number of the right predict. The second SGD method is to decide to use the training dataset or the testing dataset. The parameters in the SGD means defines how many epochs, size of batch, learning rate.

*Evaluate method*: calculate the accuracy of the testing/training dataset

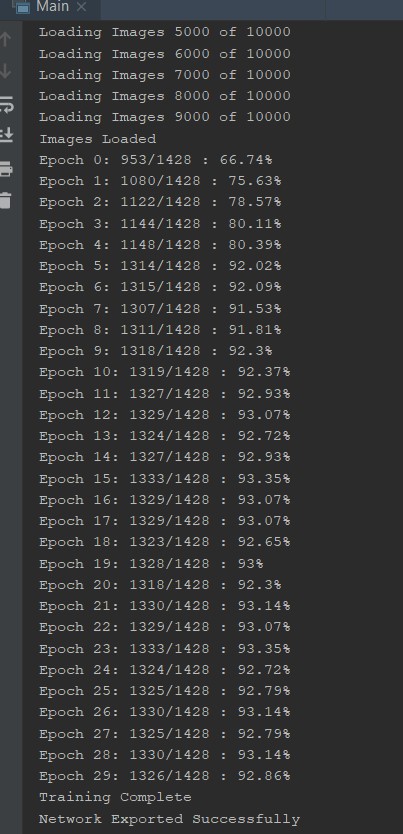
*Biases class:* a two-dimensional matrix representing the bias of each node in our network:

*Weights class*: weights is a 3-d matrix representing the weights between nodes in two different layers, for example weights[1] stores the weights connecting the second and third layers(since layer 0 is the first layer). Let weights[1] be matrix w. w[j][k] stores the weight between the kth neuron in the, second layer and the j-th neuron in the third layer.

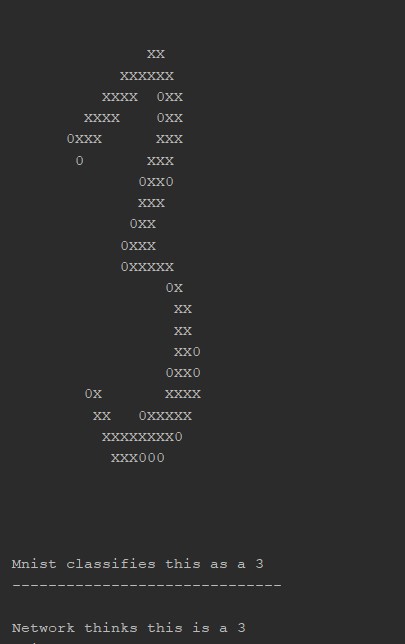
*Neuron class*: it is the corresponding input and output data of each layer of network.

*MLP class*: the input of the MLP class is an int array, the int array defines the number of nodes in each layer. The length of the int array is the number of layers in the multi-layer perceptron. We define a global Boolean variable *test*, the *test* default value is false, which means we are deploying the forward-feed/backpropagate to the training dataset. If we set the *test* into True Output description:

First, we load data from the lable and image dataset, each time we fetch 1/10 of the total amount of data, then train the data through the network. Program print out the precision after each epoch like this:



We did some visualization work to show the result of the trained model, we use *printImage* *method* in MnistReader.java. If the value after the process of the activation function is greater than 0.8, it prints out “x”. If the value after the process of the activation function, it prints out “0”. Each time we fetch one single image from the dataset, print out the actual lable and the guess of the neural network, here is the result of one single image:

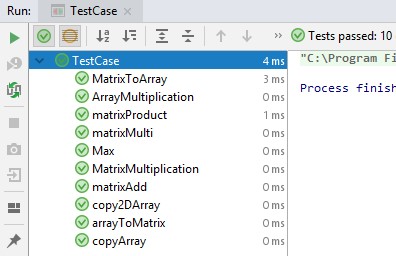


Model analysis:

Aimed at getting higher precision for the model, we changed the architecture of the model, batch size of input to each layer, size of the input dataset, epochs of training, activation function in MLP, learning rate etc.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Model No.* | *Nodes in hidden layer* | *Num of hidden layer* | *Learning rate* | *Activation function* | *Epochs* | *Accuracy* *On training set* |
| *1* | 784,30,10 | 1 | 3.0 | sigmoid | 30 | 86.2% |
| *2*  *3* | 784,30,20,10 | 2 | 3.0 | sigmoid | 30 | 85.92% |
| 784,40,30,20,10 | 3 | 3.0 | sigmoid | 30 | 85.43% |
| *4*  *5* | 784,30,10 | 1 | 3.0 | sigmoid | 100 | 91.25% |
| 784,30,10 | 1 | 9.0 | sigmoid | 30 | 89.99% |
| *6*  *7* | 784,30,10 | 1 | 3.0 | ReLU | 30 | 10.22% |
| 784,30,10 | 1 | 3.0 | Tanh | 30 | 30.5% |

Test Result：



Conclusion and Limitations:

From the experiments above, we found the rules below:

1. The number of hidden layers doesn’t affect much to the accuracy.
2. Sigmoid is the best activation function out of the ReLU, tanh function.
3. The epochs and learning rate have a positive-linear dependent to the accuracy.
4. The more dataset you trained, the higher accuracy you will get.

The limitations of the model are:

1. The activation function of each layer is the same.
2. The best accuracy is not good as our expectation, it is about 92%.

Reference:

1. Mnist reader: <https://github.com/jeffgriffith/mnist-reader>
2. Mnist file processing: <https://github.com/turkdogan/mnist-data-reader>
3. Calculus on Computational Graphs ---- Backpropagation:

<http://colah.github.io/posts/2015-08-Backprop/>