Mnist Project

Yilin Zheng 11510506
Department of Computer Science and Engineering
Southern University of Science and Technology
Email: 11510506@mail.sustc.edu.cn

1. Preliminaries

1.1. Software and Library

This project is written in Python using editor VSCode with python interpreter 3.6.4. The libraries being used includes os, pickle, *numpy*, random, *matplotlib*, collections and decimal.

1.2. Algorithm

This project is to train a feed-forwarding neural network for mnist problem, in which, the algorithm includes two part:

- feed forward
- back propagation

With most of the codes is conduct computation for weighted update.

2. Methodology

2.1. Introduction

Artificial Neural network(usually shorts as neural network) rapidly develops in recent years. However, it is not a new concept since its origin can be dated back to 1943. In 1943, Warren McCulloch and Walter Pitts [1] created a computational model for neural networks based on mathematics and algorithms called threshold logic. This model paved the way for neural network research to split into two approaches. One approach focused on biological processes in the brain while the other focused on the application of neural networks to artificial intelligence which led to work on neural networks and their link to finite automata. [2] Nevertheless, the first functional networks with many layers were published by Ivakhnenko and Lapa in 1965, and it became the Group Method of Data Handling. [3] [4] [5] However, neural network research stagnated after machine learning research by Minsky and Papert (1969), [6] who discovered two key issues with the computational machines that processed neural networks. The first was that basic perceptrons were incapable of processing the exclusive-or circuit. The second was that computers didn't have enough processing power to effectively handle the work required by large neural networks.Until recently, with the development of computations and more efficient algorithms for training neural networks, situation changes quickly and nowadays neural networks are very important fields in artificial intelligence besides machine learning.

And today, neural networks are widely applied for learning problems. There are three types learning datagrams with each has its particular learning task:

- **Supervised learning**: tasks can be approximately divided into two parts: *classification*(pattern recognition) and *regression*.
- **Unsupervised learning**: tasks of general estimation problems including *clustering*, the *estimation of statistical distributions*, *compression* and *filtering*.
- **Reinforcement learning**: tasks includes *control problems*, *games* and other *sequential decision making tasks*.

2.2. Problem

In this project, the target is to use neural networks solving mnist problem. This is a classification problem with the famous dataset **handwritten digits** (http://yann.lecun.com/exdb/mnist/).

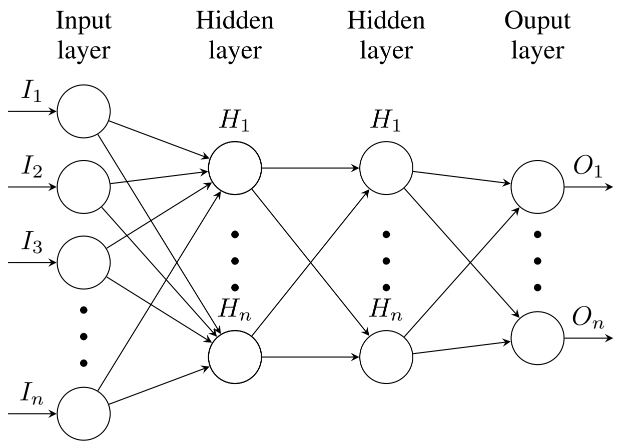
Neural network need to be trained with satisfactory accuracy and acceptable loss on this dataset. The neural network will classifies every input images to match a one hot vector representing a digit integer number in $[0, 9]$.

So, this is a **classification problem** and it is also a *multi-class classification problem*(as well as One-vs-all).

2.3. Model and Algorithm

Model used in this project to target the goal is a simple full connection neural network with exact 1 input layer with 784 neurons, 2 hidden layers with 300 neurons and 100 neurons seperately and a 10 neurons output layer for final output.

The structure can is as following:



2.4. Representation

There are three important representation in this project which should be concerned. One part is the format of dataset which related to how we manage the data, the second is the representation of neural network layers which closely related to how we train the network and the last is the representation of labels. of both training data and test data.

The data of every picture trained and tested are reshaped to a 1×784 dimensional array(also as vector) and every batch of data form the matrix sent to input layer.

The layers are represented by weighted matrices of dimension 784×300, 300×100 and 100×10. Actually, the weighted matrices are not the layer in concept, but the weighted matrices acts as an affine to map the input data of previous layer output (or origin input data) to a high dimensional space and the number of the matrices are usually the same as the number of layers. The actual layer contains two values, state value z which is the value of previous layer output and activation value which are the output value of activation functions.

The labels uses another distinct format to represent its values, called *one-hot* vector and it make only bit to be 1 while rest to be 0 corresponding the value it represents.

Besides, some global variables are set as *hyperparameters* which are some key parameters we can adjust for training a neural network. They are:

- **epoch**: epoch number of training
- **learning_rate**: learning step length for gradient descent
- **n**: iteration numbers for each epoch
- **lambda_value**: regularization parameter for avoiding overfitting
- **batch_size**: batch size which is the number of input data every iteration

Also, kernal size can be modified in this code and list as following:

- **input_size**: the kernal size of input layer
- **kernal_1**: the kernal size of hidden layer 1
- **kernal_2**: the kernal size of hidden layer 2
- **output_size**: the kernal size of output layer

2.5. Architecture

The architecture of the source codes includes fours classes and one part for training control. The fours parts are:

- **Layer**: layer class which stores weights and bias
- **Activation**: activation class which contain several activations
- **Loss**: loss class which provides the loss functions
- **Simple_neural_network**: simple neural network class that are the main class for implementing the neural network, mainly including *feed forward* and *back propagation*.

2.6. Details of Algorithm

This part will talk about the algorithms used in this project with related formulas must known and the method of training with mnist dataset.

The algorithm used in this project is mostly the **feed forward** and **back propagation**. For feed forward, in every layer, calculate its values use following formula:

$$y(x, W, b) = W^T x + b \tag{1}$$

here the weight W is random generated on a uniform distribution, which also call Xavier initialization [7], between:

$$\left[-\frac{\sqrt{6}}{\sqrt{m+n}}, \frac{\sqrt{6}}{\sqrt{m+n}}\right] \tag{2}$$

where m is the row number and n is the column number actually;

after that, calculate its activation values:

$$a = f(y) \tag{3}$$

here, the activation function are often different between hidden layers and output layers. In this projectm we use **ReLU**(Rectified Linear Unit) in hidden layers while **softmax** in output layers.

ReLU:

$$f(y) = max(y, 0) \tag{4}$$

according to (2):

$$f(y) = max(W^T x + b, 0) \tag{5}$$

Softmax, for $j = 1, ..., K$:

$$f(y) = \frac{e^{y_j}}{\sum_k^K e^{y_k}} \tag{6}$$

according to (2):

$$f(y = j|x) = \frac{e^{x^T w_j}}{\sum_k^K e^{x^T w_k}} \tag{7}$$

Algorithm 1 FeedForward

```
Input: training data $x$
Output: predict values $h$
1: $a \leftarrow x$
2: for training in every hidden layer do
3:     $z \leftarrow$ feed forward $a$ by weights and bias
4:     $a \leftarrow$ activate $z$ by specified activation functions
5: end for
6: $h \leftarrow$ final activation values $a$
7: return $h$
```

For back propagation, first calculate the gradientδ of every layer and then update weights and bias of every layer by using δ.

To calculate the δ, we need to firstly calculate the loss of the predict, here in this project, cross entropy loss is used:

$$E(t,h) = -\sum \left(t \ln h^T + (1-t)\cdot \ln(1-h)\right) \tag{8}$$

then calculate the derivative for both weight W and bias b:

$$\Delta W = \frac{\partial E}{\partial W} \tag{9}$$

$$\Delta b = \frac{\partial E}{\partial b} \tag{10}$$

for weight update, t represents the number of iteration, with learning rate η introduced:

$$W^{t+1} = W^t - \eta \Delta W^t \tag{11}$$

for bias update:

$$b^{t+1} = b^t - \eta \Delta b^t \tag{12}$$

Algorithm 2 BackPropagation

```
Input: predict values $h$, training data labels $t$
1: calculate gradient of every layer $\delta$
2: update weight $w$ and bias $b$ backward from output layer
   to input layer using calculated $\delta$
```

The training method is to train the dataset by 100 epochs with each epoch containing a batch of data(batch size 100) conducting feed forward and back propagation for 100 iteration.

3. Empirical Verification

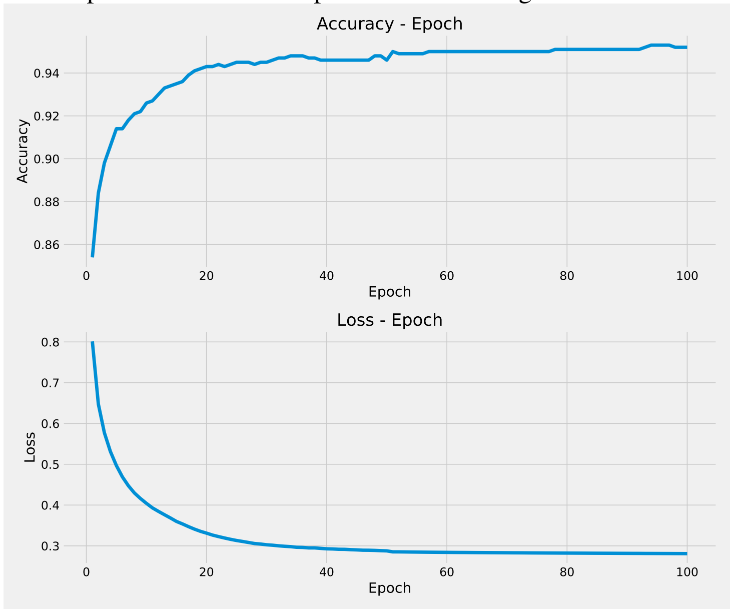
Empirical verification is conducted on mnist dataset with training data size 10000 and test data size 1000 with the following parameters setting:

- **epoch**: 100
- **learning_rate**: 0.1
- **n**: 100
- **lambda_value**: 0.0005
- **batch_size**: 100

TABLE 1. Result of Training

Epoch	1	5	10	20	40	80
Loss	0.8013	0.4969	0.4042	0.3312	0.2927	0.2822
Accuracy	0.854	0.914	0.926	0.943	0.946	**0.951**

The empirical result can be plotted as following:



The figure above shows that the loss goes down quickly at first and then approaches to 0 but decreases at a very slow speed. Likely to the loss change, the accuracy grows rapidly then goes down to a very slow improvement speed which approaches to 0.95.

3.1. Design

The design in this project which must be mentioned is the structure of neural network we used. The structure of neural network here we use is a simple full connection neural network with 2 hidden layers containing 300 and 100 neurons separately. The activation function of hidden layers is ReLU and the activation function of output layer is Softmax. The training method is back propagation which is still the most efficient way for training neural networks. Although in principle, only one hidden layers with enough neurons can approaches arbitrary non-linear functions but empirical results suggest that we prefer use multiply hidden layers so as to lower the difficulty of training a neural network without loss accuracy.

3.2. Data and data structure

The original black and white (bilevel) images from NIST were size normalized to fit in a 20×20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centred in a 28×28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28×28 field. [8]

3.3. Performance

The performance measures in this project is the running time of conducting a complete training. The running time is about 72 seconds so we can know that less 1 second for each epoch.

3.4. Result

The result can be measured by loss and accuracy. The loss is cross entropy loss which approaches to 0 finally and the accuracy is limited to 95%.

3.5. Analysis

This project is to implement a simple artificial neural network with solving classification problem on typical dataset mnist. Through this project we can go deep understanding of ANNs and the principle of feed forward and back propagation. More practice are need to be taken to help us go even further on this field where involves large amount of researches and experiments.

Acknowledgments

I would like to thank to Yao Zhao who reminded me some details which led to solve my final bug on this project. Also, our assistant student provided me help. Finally, I would thank for myself who instant on doing this project even after re-writing all the codes 3 times since for a long time no bug could be found but ANN still wouldn't function normally.

References

[1] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115133. doi:10.1007/BF02478259.

[2] Kleene, S.C. (1956). "Representation of Events in Nerve Nets and Finite Automata". Annals of Mathematics Studies (34). Princeton University Press. pp. 341. Retrieved 2017-06-17.

[3] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85117. arXiv:1404.7828Freely accessible. doi:10.1016/j.neunet.2014.09.003. PMID 25462637.

[4] Ivakhnenko, A. G. (1973). Cybernetic Predicting Devices. CCM Information Corporation.

[5] Ivakhnenko, A. G.; Grigorevich Lapa, Valentin (1967). Cybernetics and forecasting techniques. American Elsevier Pub. Co.

[6] Minsky, Marvin; Papert, Seymour (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press. ISBN 0-262-63022-2.

[7] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 249-256).

[8] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges", Yann.lecun.com. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 27- Dec- 2017].