Research and Implementation of Handwritten Numbers Recognition System Based on Neural Network and Tensorflow Framework

Jing Bai * and Jie Fei

College of Information Technology, Shanghai Jian Qiao University, Shanghai 201306, China

* E-mail: 15008@gench.edu.cn

Abstract. It is difficult for a computer to recognize handwritten numbers with a high accuracy because of many factors such as shape, size and others. This paper applies a neural network based on Tensorflow framework to implement the handwritten numbers recognition system. The system is compiled and run in vim under Mac operating system with an 8g RAM. This system first builds a neural network, then trains the model on the MNIST dataset to output the accuracy of handwritten numbers recognition. We can input the handwritten picture and get the prediction result through this system. Then we optimize this system on neural network, activation function, loss function, learning rate and generalization to improve the accuracy. Finally, we prove that this system has a higher accuracy closed to 98% for recognizing.

1. Introduction
For many years, the problem of recognizing handwritten numbers has been the focus of researches. The technology refers to recognize handwritten 0 to 9 Arabic numbers by a computer. Number recognition technology can be applied to the fields of postal code recognition, bank bill code recognition, license plate recognition, etc. [1]. But it is difficult for a computer to correctly recognize handwritten numbers because that the shape, size, depth, clarity, position and personal writing style of handwritten numbers are often different [2].

Artificial intelligence has proved the coming of the intelligent era, from AlphaGo's victory over Korean player Li Shishi in the Chess competitions in 2016 to the victory of Master over the world's number one player Ke Jie in 2017. In recent research, artificial intelligence and neural networks have been used in character recognition, speech recognition and target recognition [3]. In the field of numbers recognition, the methods of artificial intelligence that used frequently include optical flow method [4], classification method [5], KNN [6], SVM [7], etc. In this paper, the system can recognize handwritten numbers 0-9 with a higher accuracy, and we improve the recognition accuracy through some optimization by building a neural network and training the model.

2. Neural network and Tensorflow framework
A neural network is a mathematical model of an algorithm that imitates the characteristics of animal’s neural networks and performs distributed parallel information processing. This kind of network depends on the complexity of the system and adjusts the interconnection relationship between a large number of
internal nodes to achieve the purpose of processing information. The neural network consists of multiple neurons. A single neuron is shown in Figure 1:

![Figure 1. Neuron model](image)

This neuron is an arithmetic unit with \(x_1, x_2, x_3\) and intercept +1 as input values. The output is \(h_{W,b}(x) = f(W^T x) = f(\sum_i W_i x_i + b)\), where the function \(f(\cdot)\) is called "activation function" which generally select sigmoid function as the activation function \(f(\cdot)\). As shown in formula (1):

\[
f(z) = \frac{1}{1 + e^{-z}}
\]  

(1)

A neural network is a combination of many single neurons, so that the output of one neuron can be the input of another. Figure 2 shows a simple neural network:

![Figure 2. A neural network](image)

We use \(a_i^{(l)}\) to denote the output of the \(i^{th}\) unit of the \(l^{th}\) layer. When \(l = 1\), \(a_i^{(l)} = x_i\), which is the \(i^{th}\) input. For a given parameter set \(W\) and \(b\), the neural network can calculate the output according to the function \(h_{W,b}(x)\).

Tensorflow is the second-generation artificial intelligence learning system introduced by Google [8], which has higher computing efficiency, better flexibility and better portability. At present, Tensorflow has been widely used in many machine learning and deep learning fields such as text processing, speech recognition and image recognition.

The neural network based on Tensorflow uses tensor to represent the data, builds the neural network with calculation graph, executes the calculation graph with drawing, optimizes the weights (parameters) on the line, and obtains the model.

2.1. Parameters of the neural network

The parameters of the neural network refer to the weight \(w\) on the line between neurons, which is expressed by variables. Generally, these parameters are generated randomly at first. The way to generate parameters is to make \(w\) equal to \(tf\cdot variable\), and write the way to generate in parentheses.

Functions commonly used in neural networks to generate random numbers or arrays are:

- \(tf\cdot random\_normal()\): Generates a normal distributed random number.
- \(tf\cdot truncated\_normal()\): Generates a normal distributed random number which removes excessively large deviation points.
- \(tf\cdot random\_uniform()\): Generates a uniformly distributed random number.
- \(tf\cdot zeros\): Generates an all 0 array.
- \(tf\cdot ones\): Generates an all 1 array.
- \(tf\cdot fill\): Generates a full fixed value array.
• \textit{tf.constant}: Generates an array of directly given values.

This system uses \( w = \text{tf.Variable}(\text{tf.random_normal}([2,3], \text{stddev} = 2, \text{mean} = 0, \text{seed} = 1)) \), which means that a random number with normal distribution is generated, the shape is two rows and three columns, and the standard deviation is 2. The mean is 0 and the random seed is 1.

2.2. Implementation of neural networks

The implementation process of the neural network:

• Prepare a data set, extract features, and feed them to a neural network as input.
• Build the neural network structure from input to output (set up the calculation graph first, and then execute it with a session). Neural network forward propagation algorithm is used to calculate the output.
• Feed a large amount of feature data to neural network, and iteratively optimize neural network parameters. Back propagation algorithm of neural network is used to optimize the parameter training model.
• Forecast and classify the result using a trained model.

It is mainly divided into two processes, namely the training process and the use process. The training process is a cyclic iteration of the first step, the second step, and the third step. The use process is the fourth step. Once the parameter optimization is completed, these parameters can be fixed to achieve a specific application.

2.2.1. Forward propagation. Forward propagation is the computational process of building a model, so that the model is capable of reasoning and can give a corresponding output for a set of inputs. The schematic diagram of the neural network is shown in Figure 3.

![Figure 3. An example of a forward propagation neural network](image)

• \textbf{The first layer}: The input is represented by \( x \), which is a matrix of 1 row and 2 columns. It means that a set of features is input at one time. This set of features contains two elements of volume and weight. \( W_{\text{layer}} \) is the parameter to be optimized. Represented as shown in formula (2):

\[
W^{(1)} = \begin{bmatrix} W^{(1)}_{1,1} & W^{(1)}_{1,2} & W^{(1)}_{1,3} \\ W^{(1)}_{2,1} & W^{(1)}_{2,2} & W^{(1)}_{2,3} \end{bmatrix}
\]  

(2)

The input is not the calculation layer. So \( \alpha \) is a matrix with one row and three columns which is the first layer of network. As is shown in formula (3):

\[
\alpha_1 = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \end{bmatrix} = XW^{(1)}
\]  

(3)

• \textbf{The second layer}: The parameters must fit the three nodes before and the next node, so \( W^{(2)} \) is a matrix with three rows and one column. As shown in formula (4) and (5):
\[ W^{(2)} = \begin{bmatrix} W_{1,1}^{(2)} \\ W_{2,1}^{(2)} \\ W_{3,1}^{(2)} \end{bmatrix} \]  

\( y = a_{11}W_{1,1}^{(2)} + a_{12}W_{2,1}^{(2)} + a_{13}W_{3,1}^{(2)} \)  

Multiply the input of each layer by the weight \( w \), so that the output \( y \) can be calculated by matrix multiplication.

2.2.2. Back propagation. The training model parameters of back propagation minimize the loss function of the neural network model on the training data, using gradient descent on all parameters.

**Loss:** We use the loss function to represent the difference between the predicted value \( y \) which is obtained through the calculations and the known answer \( y_\sim \). There are many methods for calculating the loss function, and the mean square error \( MSE \) is one of the more commonly used methods. The mean square error \( MSE \) is the average of the squared difference between the forward propagation calculation result and the known answer. As shown in formula (6):

\[ MSE(y_\sim, y) = \frac{\sum_{i=1}^{n} (y - y_\sim)^2}{n} \]  

Back propagation training methods: To optimize the goal by reducing the loss, there are optimization methods such as gradient descent, momentum optimizer, and adam optimizer. These three optimization methods can be expressed as functions of tensorflow:

- **gradient descent**: The random gradient descent algorithm is used to move the parameters in the opposite direction of the gradient, that is, the direction in which the total loss is reduced, so as to update the parameters. As shown in Figure 4:

![Figure 4. Gradient descent optimizer](image)

The parameter update formula is shown in formula (7):

\[ \theta_{n+1} = \theta_n - \alpha \frac{\partial J(\theta_n)}{\partial \theta_n} \]  

Among them, \( J(\theta_n) \) is loss, \( \theta \) is parameter, and \( \alpha \) is learning rate.

- **momentum optimizer**: When updating the parameters, the hyperparameters are used, and the parameter update formulas are shown in formula (8) and (9):

\[ d_i = \beta d_{i-1} + g(\theta_{i-1}) \]  

\[ \theta_i = \theta_{i-1} - \alpha d_i \]  

Among them, \( \alpha \) is learning rate, \( \beta \) are hyperparameters, \( \theta \) is a parameter, and \( g(\theta_{i-1}) \) is a gradient of a loss function.

- **adam optimizer**: It is an optimization algorithm using an adaptive learning rate. The adam algorithm is different from the random gradient descent algorithm. The random gradient descent algorithm maintains a single learning rate to update all parameters, and the learning rate does not change during the training process. The adam algorithm designs independent adaptive learning rates for different parameters by calculating the first and second moment estimates of the gradient.

**Learning rate**: Determines the magnitude of each parameter update. The optimizer requires a parameter called the learning rate. When used, if the learning rate is selected too large, there will be
oscillations and convergence. If the learning rate is too small, the convergence rate will be slow. We can choose a relatively small value, such as 0.01, 0.001.

**Parameter update:** $z^l$ represents the input value of the hidden layer and the output layer of the $l$ layer. $a^l$ represents the output value of the hidden layer and the output layer of the $l$ layer. $f(z)$ represents the activation function. The final output layer is the $L$ layer.

Layer $L$ output in forward propagation:

$$a^l = f(z^l) = f(w^l a^{l-1} + b^l), \text{ loss: } J(w, b, x, y) = \frac{1}{2} ||a^l - y||^2_2.$$ 

For the $L$ layer of the output layer, there are:

$$a^l = f(z^l) = f(w^l a^{l-1} + b^l) \quad (10)$$

Find partial derivatives, then:

$$\frac{\partial J(w, b, x, y)}{\partial w^L} = \frac{\partial J(w, b, x, y)}{\partial z^L} \frac{\partial z^L}{\partial w^L} = (f(z^L) - y) f'(z^L) a^{l-1} = (a^l - y)(a^{l-1})^T \odot f'(z^L) \quad (11)$$

$$\frac{\partial J(w, b, x, y)}{\partial b^L} = \frac{\partial J(w, b, x, y)}{\partial z^L} \frac{\partial z^L}{\partial b^L} = (f(z^L) - y) f'(z^L) = (a^l - y) \odot f'(z^L) \quad (12)$$

Update the parameters $w^L$ and $b^L$ of the last $L$ layer, then:

$$w^L = w^L - \frac{\partial J(w, b, x, y)}{\partial w^L} \quad (13)$$

$$b^L = b^L - \frac{\partial J(w, b, x, y)}{\partial b^L} \quad (14)$$

Let $\delta^l = \frac{\partial J(w, b, x, y)}{\partial z^L} = \frac{\partial J(w, b, x, y)}{\partial z^{l-1}} \frac{\partial z^{l-1}}{\partial z^L} \ldots \frac{\partial z^{l+1}}{\partial z^L} \frac{\partial z^L}{\partial w^L} = \delta^l (a^{l-1})^T \quad (15)$

$$\frac{\partial J(w, b, x, y)}{\partial b^l} = \frac{\partial J(w, b, x, y)}{\partial z^L} \frac{\partial z^L}{\partial b^L} = \delta^l \quad (16)$$

$\delta^l$ can be obtained by induction:

$$\delta^l = \frac{\partial J(w, b, x, y)}{\partial z^L}$$

Because: $z^{l+1} = w^{l+1} a^l + b^{l+1}$, then:

$$\delta^l = \frac{\partial J(w, b, x, y)}{\partial z^L}$$

The parameters and update formula of the $L$ layer are:

$$w^l = w^l - \frac{\partial J(w, b, x, y)}{\partial w^l} = w^l - \delta^l (a^{l-1})^T = w^l - \delta^{l+1} (w^{l+1})^T \odot f'(z^L) \quad (17)$$

$$b^l = b^l - \frac{\partial J(w, b, x, y)}{\partial b^l} = b^l - \delta^l = b^l - \delta^{l+1} (w^{l+1})^T \odot f'(z^L) \quad (18)$$

2.3. Optimization

The optimization process starts from the following perspectives: neural network, activation function, loss function, learning rate, generalization.

2.3.1. Neural Networks. The neural network is composed of neurons as basic units. The neuron model is expressed by a mathematical formula: $f(\sum_i x_i w_i + b)$, where $f$ is the activation function. The optimization of the neural network model can be adjusted from the complexity and parameters of the neural network. The complexity of a neural network can be represented by the number of layers of the neural network and the number of parameters to be optimized in the neural network. The neural network generally does not count into the input layer, the number of layers = $n$ hidden layers + 1 output layer. The parameters to be optimized in the neural network refer to the number of all parameters $w$ + the number of all parameters $b$ in the neural network.

2.3.2. Activation function. The nonlinear activation factor is introduced to improve the expressive power of the model. Commonly used activation functions are *relu, sigmoid, tanh*, etc. This system uses *sigmod*, as shown in formula (1) and Figure 5.
2.3.3. Loss. The loss function is used to represent the difference between the predicted value \( y \) and the known answer \( y_\_ \). When training a neural network, the loss function is continuously reduced by continuously changing all parameters in the neural network. Then we can get a neural network model with higher accuracy. Common loss functions include mean square error, customization, and cross entropy.

- **Mean square error mse**: The sum of the squares of the difference between the predicted value \( y \) and the known answer \( y_\_ \) of the \( n \) samples. Then get the average. As shown in formula 6.
- **Custom loss function**: According to the actual situation of the problem, customize a reasonable loss function.
- **Cross Entropy**: Represent the distance between two probability distributions. The larger the cross-entropy, the farther the two probability distributions are, the more different the two probability distributions are. The smaller the cross-entropy, the closer the two probability distributions are, and the more similar the two probability distributions are. Cross entropy calculation formula:

\[
H(y_\_, y) = - \sum y_\_ \log y 
\]  

2.3.4. Softmax. Change the \( n \) outputs \((y_1, y_2 \ldots y_n)\) of \( n \) classification into functions that satisfy the requirements of the probability distribution of the formula: \( \forall x \ P(X = x) \in [0,1] \) and \( \sum P(X = x) = 1 \). The softmax function is expressed as:

\[
s_{oftmax}(y_i) = \frac{e^{yi}}{\sum_{j=1}^{n} e^{yj}} \]  

In the TensorFlow framework, the output of the model is generally passed through the softmax function to obtain the probability distribution of the output classification, and then compared with the standard answer to find the cross-entropy to obtain the loss function.

2.3.5. Learning rate. Indicates the magnitude of each parameter update. If the learning rate is too large, the parameters to be optimized will fluctuate around the minimum value which will make the result not converge. If the learning rate is too small, the parameters to be optimized will converge slowly. During the training process, the parameters are updated in the direction of the gradient of the loss function. The parameter update formula is:

\[
w_{n+1} = w_n - \text{learning rate} \nabla \]  

Exponential decay learning rate: The learning rate is dynamically updated as the number of training rounds. The learning rate calculation formula is as follows:

\[
\text{learning rate} = \text{LEARNING RATE BASE} \times \text{LEARNING RATE DECAY} \times \frac{\text{global} \text{ LEARNING RATE BATCH SIZE}}{\text{LEARNING RATE BATCH SIZE}} \]  

2.3.6. Generalization. Moving average: The average values of all parameters \( w \) and \( b \) in the model are recorded over a period of time. Using the moving average can enhance the generalization ability of the model.

- **Overfitting**: The accuracy rate of the neural network model on the training dataset is high, and the accuracy rate is low when new data is used for prediction or classification, indicating that the model has poor generalization ability.
• Regularization: To suppress model noise and reduce overfitting, we add weight to each parameter \( w \) in the loss function and introduce model complexity indicators.

3. System implementation
The system is compiled and run in vim under Mac operating system with an 8g memory. First build a neural network, then train the model on the Mnist dataset to output the accuracy of handwritten numbers recognition. Implement the breakpoint continuous training function, then input the handwritten picture, finally output the prediction result.

3.1. Mnist dataset
Mnist dataset contains 70,000 pictures of handwritten numbers with black and white characters, of which 55,000 are in training sets, 5000 are in validation sets, and 10,000 are in test sets. The size of each picture is 28 * 28 pixels. The pure black pixel value in the picture is 0, and the pure white pixel value is 1. The label of the dataset is a one-dimensional array of length 10. The index of each element in the array represents the probability of the corresponding number.

The file is shown in Figure 6.

![Mnist dataset file](image)

**Figure 6.** Mnist dataset file

3.2. Forward propagation process
The forward propagation process completes the construction of the neural network. In the forward propagation process, it is necessary to define the parameters \( w \) and offset \( b \) in the neural network, and to define the network structure from input to output. Set the parameter \( w \) by defining the function `get_weight()`, including the shape of the parameter \( w \) and the flag of whether it is regularized. Similarly, the setting of the offset \( b \) is achieved by defining the function `get_bias()`.

In the forward propagation process, when the regularization parameter is set to 1, it indicates that when optimizing the model parameters in back propagation process, a regularization term needs to be added to the loss function.

3.3. Backward propagation process
The backward propagation process completes the training of the network parameters. When training the model, the exponential decay learning rate can make the model quickly converge close to the better solution in the early stages of training, and also ensure that the model does not fluctuate too much in the later stages of training. The introduction of moving averages during model training can make the model more robust on test data.

3.4. Test accuracy.
After the neural network model is trained, it can be used to test the dataset to verify the performance of the neural network.

3.5. Recognizing handwritten numbers.
Network input: one-dimensional array (784 pixels).

Pixels: float numbers between 0-1 (black near 0 and white near 1).

Network output: one-dimensional array (ten possibilities). The index corresponding to the largest element in the array is the prediction result.
Detailed process:
- The model requires white text on a black background, but the input image is black text on a white background. The value of each pixel needs to be changed to the value making 255 minus the original value.
- Binarize the picture (so as to filter out noise and adjust the threshold appropriately during debugging).
- Pull the image shape into 1 row and 784 columns. Then change the value to float number (because the pixel is required to be a float number between 0-1).
- Change the existing RGB image from a number between 0-255 to a float number between 0-1.
- Return to main function after running.
- The output index $y$ is calculated. The index of the list corresponding to the maximum value of $y$ is the prediction result.

4. Test

4.1. Neural network training (back propagation).
As the neural network undergoes multiple rounds of training, it can be seen from figure 7 that the model loss rate gradually decreases.

![](image)

Figure 7. The loss rate gradually decreases during training.

4.2. Neural Network Accuracy Test.
As the number of training rounds increases, it can be seen from figure 8 that the accuracy rate has gradually improved. When the accuracy rate reaches about 98%, handwritten numbers recognition can be performed.

![](image)

Figure 8. The accuracy of the model gradually improves during training.
4.3. Handwritten numbers recognition.
Write down 0 to 9 as standard as possible on a clean piece of white paper, and input the model for recognition. The recognition result is shown in Figure 9.

![Figure 9. Handwritten numbers recognition test results.](pic)

5. Conclusion
Handwritten numbers recognition technology is widely used, but the recognition effect and efficiency are very different. In many cases, there is no way to meet the ideal requirements. For example, factors such as picture brightness, sharpness, colour and noise will affect the recognition effect, so it is necessary to have a good noise-reduction function in the system. The quality of the noise reduction function will directly affect the recognition effect.

References
[1] Kui Z 2012 Recognition method of handwritten numbers based on genetic-BP neural network *Xi'an University of Science and Technology*.
[2] Zhuang M 2007 Research on Handwritten Numbers Recognition Model Based on DFS and Its Application *Suzhou University*.
[3] Nanning Z and Long W 1992 Improvement of BP Neural Network and Its Application to Handwritten Numbers Recognition *Journal of Xi'an Jiaotong University* 1 1-12.
[4] Jing B and Lili H 2018 Research on LK Optical Flow Algorithm with Gaussian Pyramid Model Based on OpenCV for Single Target Tracking (Shanghai: Proceedings of 2018 2nd International Conference on Artificial Intelligence Applications and Technologies) p443-448.
[5] Jiamin Z 2013 Research and Application of Handwritten Numbers Recognition Based on Bayesian Network *Science & Technology Horizon* 8 27-27.
[6] Weidong Z, Yonghong L and Tao Y 2017 Research on Handwritten Numbers Recognition Based on KNN Algorithm *Journal of Chengdu University (Natural Science Edition)* 4 382-384.
[7] Chuang Z, Ming W and Jun G 2003 Research on Similarity Recognition of Handwritten Numerals Based on SVM *CEA*. 11 36-38 + 47.
[8] Yan X, Haowei Y and Zhi L 2018 Research on Handwritten Numbers Recognition Based on Convolutional Neural Network and TensorFlow *Shanghai Electric Technology* 39 (01) 35-38 + 65