The Research about the MNIST Database of Handwritten Digits by Applying Python

Dongxiao Ma, Jiajian Chang, Chen-Yin Yu

Machine Learning I DATS 6202

George Washington University

August 17, 2019

Contents

I.Introduction

Handwriting digits recognition is the ability of a computer system to recognize the handwritten inputs like digits, characters etc. from a wide variety of sources like emails, papers, images, letters etc. This has been a topic of research for decades. The most popular handwriting digits dataset is the MNIST database

The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Hence, it was necessary to build a new database by mixing NIST's datasets. The new training set was completed with enough examples from SD-3, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with SD-3 examples starting at pattern # 35,000 to make a full set with 60,000 test patterns. Only a subset of 10,000 test images (5,000 from SD-1 and 5,000 from SD-3) is available on this site. The full 60,000 sample training set is available. The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and

5,000 patterns from SD-1. The 60,000 patterns training set contained examples from approximately 250 writers [1].

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 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 centered in a 28x28 image by computing the center of mass of the pixels, and translating **the image s**o as to position this point at the center of the 28x28 field. The labels are numbers from zero to nine.

TRAINING SET LABEL FILE (train-labels-idx1-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
.....			
xxxx	unsigned byte	??	label

The labels values are 0 to 9.

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
.....			
xxxx	unsigned byte	??	pixel

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black).

Figure 1. Dataset illustration

These files are not in any standard image format. All the integers in the files are stored in the Binary format (high endian). From figure1, it is concluded that we can read the training set label from the 8th offset because that the first four is magic number and the 4th to 7th is the number of labels. For the same reason, the training set image file should be read from the 16th offset. Figure 2 is the visualization of the training set

label file. It can be seen that the number from offset 8 is no bigger than nine, and the number from offset 16 is no bigger than 255 in the training set image file. The image is a 28x28 matrix in which the value of each element is among 0 to 255. Each element represents the color of one pixel and the number represents how black the pixel is. 0 means white and 255 means black [2].

```
1    Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2    00000000: 00 00 08 01 00 00 EA 60 05 00 04 01 09 02 01 03
3    00000010: 01 04 03 05 03 06 01 07 02 08 06 09 04 00 09 01
4    00000020: 01 02 04 03 02 07 03 08 06 09 00 05 06 00 07 06
5    00000030: 01 08 07 09 03 09 08 05 09 03 03 00 07 04 09 08
6    00000040: 00 09 04 01 04 04 06 00 04 05 06 01 00 00 01 07
7    00000050: 01 06 03 00 02 01 01 07 09 00 02 06 07 08 03 09
8    00000060: 00 04 06 07 04 06 08 00 07 08 03 01 05 07 01 07
9    00000070: 01 01 06 03 00 02 09 03 01 01 00 04 09 02 00 00
```

Figure 2. Visualization of the training set label file (hex format)

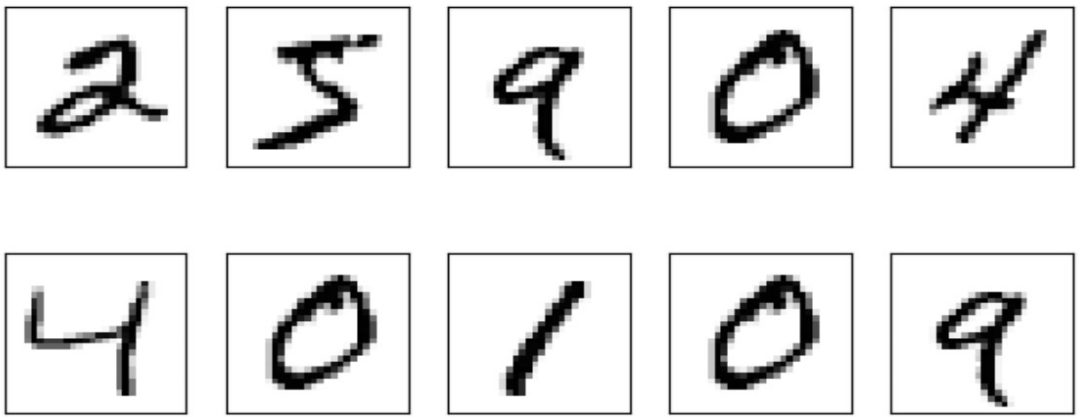


Figure 3. Images of the handwriting digits

The dataset brings us a classification problem which is supervised learning. All images should be classified to ten classes.

II. Neural Network Structure

There is a pre-process of the original data. The Neural Network requires each input data to be a vector. Hence, the original 28x28 image matrix should be flatten into a vector with the length of 784. Moreover, the label should be translated into the one hot encoding in which only one bit of the state vector is asserted for any given state [3]. All other state bits are zero. Thus, if there are n states then n state flops are required. State decode is simplified, since the state bits themselves can be used directly to indicate whether the machine is in a particular state. No additional logic is required. There are numerous advantages to using the one hot design methodology:

• One hot encoding enables the neural network model to calculate the distance between the target and prediction more accurately such as Euclidean distance, Cosine similarity

• Maps easily into register-rich FPGA architectures such as QuickLogic and Xilinx.

• One-hot state machines are typically faster. Speed is independent of the number of states, and instead depends only on the number of transitions into a particular state

• Don't have to worry about finding an "optimal" state encoding. This is particularly beneficial as the machine design is modified, for what is "optimal" for one design may no longer be best if you add a few states and change some others. One-hot is equally "optimal" for all machines.

• Modifications are straightforward. Adding and deleting states, or changing excitation equations, can be implemented easily without affecting the rest of the

machine.

• There is typically no area penalty over highlyencoded machines.

• Critical paths are easy to find using static timing analysis.

• Easy to debug.

Therefore, the label should be transformed from one integer into a vector with length of ten. For instance, the vector is [1, 0, 0, 0, 0, 0, 0, 0, 0, 0], which means that the label is 0.

It is essential to decide the layer of the neural network. To decide the layer, we start with a single layer neural network with ten neurons because the classes is ten. The transfer function is Softmax function, which make the output 10 probability [4].

$$a_i = f(n_i) = \exp(n_i) \div \sum_{j=1}^{S} \exp(n_j).$$

Figure 4. Equation of Softmax Function

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} a_1^m\left(\sum_{i=1}^{S^m} a_i^m - a_1^m\right) & -a_1^m a_2^m & \cdots & -a_1^m a_{S_m}^m \\ -a_2^m a_1^m & a_2^m\left(\sum_{i=1}^{S^m} a_i^m - a_2^m\right) & \cdots & -a_2^m a_{S_m}^m \\ \vdots & \vdots & & \vdots \\ -a_{S_m}^m a_1^m & -a_{S_m}^m a_2^m & \cdots & a_{S_m}^m\left(\sum_{i=1}^{S^m} a_i^m - a_{S_m}^m\right) \end{bmatrix}$$

Figure 5. Derivative of Softmax Function

Hence, we choose the cross entropy loss function to calculate the similarities between the prediction result and the target. KL-Divergence represent the similarities

between two distribution. If the entropy fix, the cross entropy will be change with the KL-Divergence. Hence, the cross entropy represents the KL-Divergence. The reason why we pick cross entropy as the loss function is that Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. In Figure 6, y represents the target and s represent the prediction

$$\nabla e_{(s)} = (-\frac{y_1}{s_1}, -\frac{y_2}{s_2}, \cdots, -\frac{y_k}{s_k})$$

Figure 6. Derivative of Cross Entropy

Here is the summary of the required information for the neural network:

Problem Type: classification

Classes: 10

Input format: 1X 784 vector

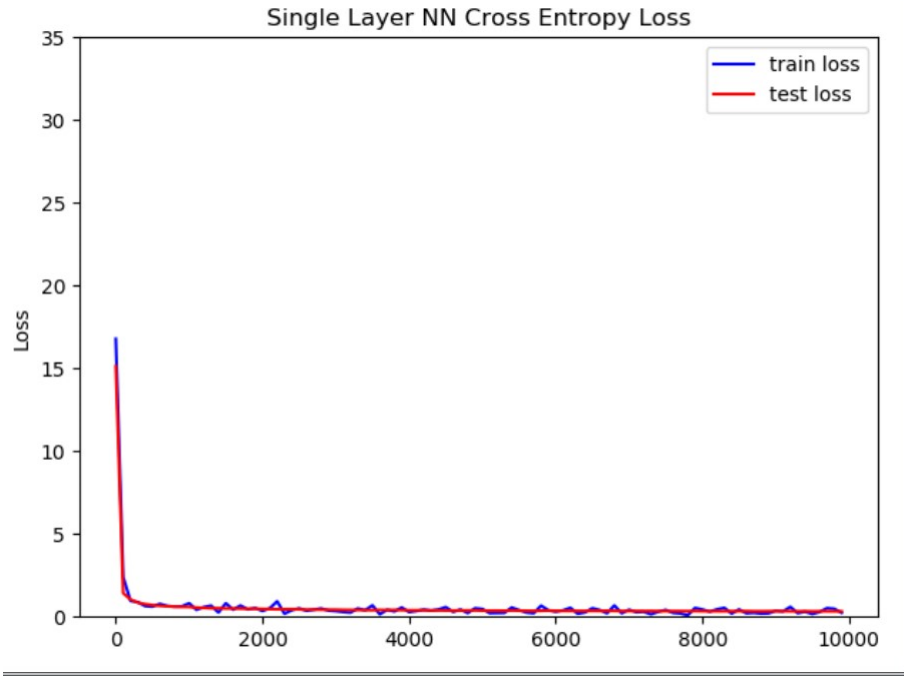
Batch Size: 100 input

Output format: 1X 10 vector(Outcome probability distribution)

Target format: 1X 10 vector(One hot encoding)

Cost function: Cross entropy

Then, we use the backpropagation and the gradient descent to train this neural network. The accuracy of this neural network is 91%. The loss function is plotted in Figure 7



From Figure7, It can be conclude that the numbers of layer and neuron is not enough because there is no overfitting in the diagram and the 91% accuracy is not enough. Hence, we decide to add more layer and neurons to recognize there images more accurately. We build 4 layers neural network. All weights are initialized among -0.2 to 0.2 and all bias are initialized with 0.1. The layer structures are:

1.First layer consists of 200 neurons. Hence, the weight is a 784 by 200 matrix, the bias is a 1 by 200 matrix. The transfer function is Relu function

2.Second layer consists of 100 neurons. Hence, the weight is a 200 by 100 matrix, the bias is a 1 by 100 matrix. The transfer function is Relu function

3.Third layer consists of 50 neurons. Hence, the weight is a 100 by 50 matrix, the bias is a 1 by 50 matrix. The transfer function is Relu function

4.The fourth layer consists of 50 neurons. Hence, the weight is a 50 by 10 matrix, the bias is a 1 by 10 matrix. The transfer function is Softmax function

From Figure 8, It can be seen that there are many 'sharp mountains' which means

that the gradient step is to large to find the minimum point. Hence, we should use the learning rate decay to make the curve smooth. The learning decay rate is 1/exp [5]. On the other hand, the cost function goes up during the last iterations. Therefore, we need to drop out some neurons to solve this problem [6]. Hence, we choose the drop out rate 0.2, and the decay function: 0.0001 + 0.003 * (1/e)^(step/2000)).

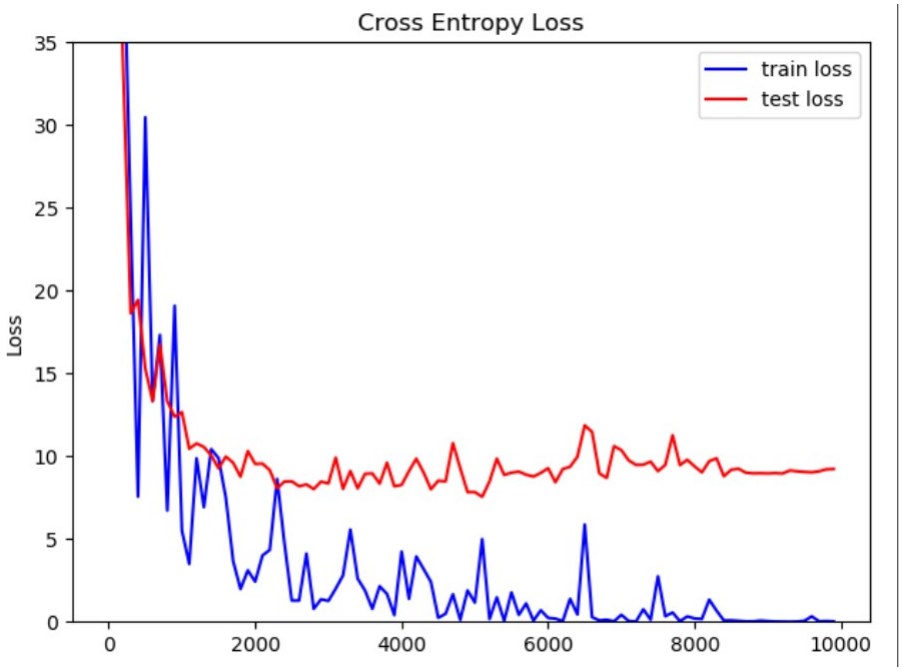


Figure 8. 4 layers NN Cross Entropy Loss

After dropping out and learning rate decay, the result is shown in Figure 9. There is no overfitting and the curve is smooth. However, the accuracy is still 98% which does not change. Although we construct five layers with more neurons, the accuracy still maintain in 98% level. The reason is that we flatten the image into a vector. We only consider the relationship in rows, not both in rows and columns. In the other word, the shape feature of the image is ignored. If we want to improve the accuracy, we should utilize the convolution neural network structure to train these data

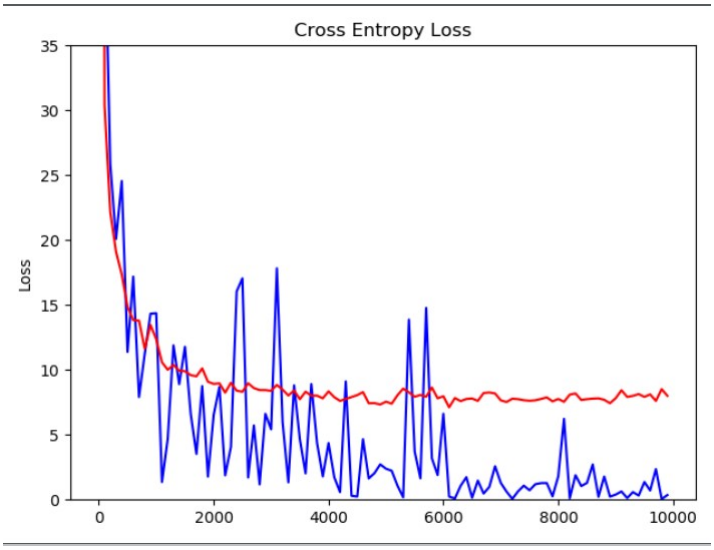


Figure 9. Modified 4 layers NN Cross Entropy Loss

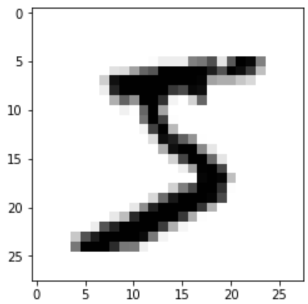
III. Process about how to build one Multi-Layer Perceptron

After figure out the character of the MNIST dataset, we need to prepare the test data. Thanks for the hardworking of Keras library, the data has been collected and is already well-formatted for processing by applying two-line python codes.[7]

```
5    mnist = keras.datasets.mnist
6    (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Figure 10: Import data

By using Jupiter notebook, the first number of the MNIST dataset can be plotted in picture and matrix way.




```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

Figure 11: First number 5 in two display formats

```
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
```

Figure 12: Reshape the 28*28 pixels pictures to the 1-D vectors which have 784 features

So, the last thing should be done before the experiment is to get the right labels of these input data. The labels for this dataset are numerical values from 0 to 9 — but it's important that our algorithm treats these as items in a set, rather than ordinal values. In this dataset, the value "0" isn't smaller than the value "9", they are just two different values from our set of possible classifications.

Thus, when we're making predictions about categorical data (as opposed to values

from a continuous range), the best practice is to use the 'one - hot' encoded vector. This means that we create a vector as long as the number of categories we have, and force the model to set exactly one of the positions in the vector to 1 and the rest to 0.[12]

```
5 training labels:  [5 0 4 1 9]
5 training lables as encoded vectors:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

Figure 13: Encoded version labels

As can be seen in the below figure 5, we use the cross entropy to calculate as the loss function. Before we talk about the cross entropy, we have to introduce some background knowledge like the Kullback-Leibler Divergence, which is used to measure the distance of two distributions.[13]

```
model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy', # change to sparse_categorical_crossentropy if choose the normalise way
              metrics = ['accuracy'])
```

Figure 14: Loss function: Cross Entropy

$$
\begin{aligned}
D_{KL}(p||q) &= \sum_{x \in X} p(x) log \frac{p(x)}{q(x)} \\
&= \sum_{x \in X} p(x) log\, p(x) - \sum_{x \in X} p(x) log\, q(x) \\
&= -H(p) - \sum_{x \in X} p(x) log\, q(x)
\end{aligned} \tag{1}
$$

Figure 14: Kullback-Leibler Divergence

In function (2), the H(p) means the entropy, we estimate there are two distributions, the cross entropy of them in the given sample space can be defined as [14]:

$$CE(p,q) = -\sum_{x \in X} p(x) log\, q(x) = H(p) + D_{KL}(p||q) \tag{2}$$

Figure 15: Cross Entropy function

Combined the (1) and (2), we can easily get:

$$CE(p,q) = -\sum_{x \in X} p(x) log\, q(x) \tag{3}$$

That's the definition of the cross entropy. Back to our MLP, the true label can be seen as p(x) and predicted label (output layer) is q(x), after the one-hot encoded version transfer, it becomes a vector with ten features like [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.], the sum of the probability is 1, [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.] shows it belongs to class 5, meanwhile, data go through the softmax classifier is also a probability distribution, this is the output layer, it may look like [0.05, 0.06, 0.06, 0.22, 0.51, 0.13, 0.01, 0.02, 0.03, 0.15], the sum of [0.05, 0.06, 0.06, 0.22, 0.51, 0.13, 0.01, 0.02, 0.03, 0.15] is 1 as well [12]. By putting these two one-dimension vectors into the function (3), we can get the value of the cross entropy, the smaller means the higher match accuracy between the prediction and the truth. Besides, knowing the loss will lead the neural network to change the weight in it.[9]

That's the part about the one-hot, softmax and the cross entropy. Next, I'd like to introduce the architecture of my Multi-Layer Perceptron. I build the 5-hidden-layer model by using the Keras API, each hidden layer has 10 neurons, training the whole network need 20s, but the accuracy is meet my demand, it only has 89% accuracy and the validated accuracy is 91% for the test dataset.[8][9][10]

```
Epoch 1/10
60000/60000 [==============================] - 2s 38us/sample - loss: 1.5255 - acc: 0.5069
Epoch 2/10
60000/60000 [==============================] - 2s 39us/sample - loss: 0.7484 - acc: 0.7777
Epoch 3/10
60000/60000 [==============================] - 2s 33us/sample - loss: 0.5142 - acc: 0.85000s - loss: 0.52
07 - acc: 0
Epoch 4/10
60000/60000 [==============================] - 2s 36us/sample - loss: 0.4309 - acc: 0.8749
Epoch 5/10
60000/60000 [==============================] - 2s 36us/sample - loss: 0.3943 - acc: 0.8849
Epoch 6/10
60000/60000 [==============================] - 3s 53us/sample - loss: 0.3632 - acc: 0.8954
Epoch 7/10
60000/60000 [==============================] - 2s 36us/sample - loss: 0.3436 - acc: 0.9013
Epoch 8/10
60000/60000 [==============================] - 2s 30us/sample - loss: 0.3295 - acc: 0.9054
Epoch 9/10
60000/60000 [==============================] - 2s 36us/sample - loss: 0.3140 - acc: 0.9095
Epoch 10/10
60000/60000 [==============================] - 2s 34us/sample - loss: 0.2998 - acc: 0.9135
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             multiple                  7850
_________________________________________________________________
dense_42 (Dense)             multiple                  110
_________________________________________________________________
dense_43 (Dense)             multiple                  110
_________________________________________________________________
dense_44 (Dense)             multiple                  110
_________________________________________________________________
dense_45 (Dense)             multiple                  110
_________________________________________________________________
dense_46 (Dense)             multiple                  110
=================================================================
Total params: 8,400
Trainable params: 8,400
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 1s 57us/sample - loss: 0.3139 - acc: 0.9121
0.31390852688252924 0.9121
```

Figure 15: Accuracy of the 5-hiddden-layer network

Then, I change the number of the neurons in 1024 to test the accuracy, without surprise, the accuracy increases significantly to 98.49% and the validated accuracy is 97% for the test dataset. [15]

```
Epoch 1/10
60000/60000 [==============================] - 25s 411us/sample - loss: 1.6180 - acc: 0.9066
Epoch 2/10
60000/60000 [==============================] - 25s 411us/sample - loss: 0.1242 - acc: 0.9636
Epoch 3/10
60000/60000 [==============================] - 25s 410us/sample - loss: 0.1003 - acc: 0.9706
Epoch 4/10
60000/60000 [==============================] - 25s 418us/sample - loss: 0.0841 - acc: 0.9762
Epoch 5/10
60000/60000 [==============================] - 25s 415us/sample - loss: 0.0745 - acc: 0.9780
Epoch 6/10
60000/60000 [==============================] - 25s 421us/sample - loss: 0.0689 - acc: 0.9807
Epoch 7/10
60000/60000 [==============================] - 26s 430us/sample - loss: 0.0607 - acc: 0.9833
Epoch 8/10
60000/60000 [==============================] - 26s 425us/sample - loss: 0.0614 - acc: 0.9830 - loss:
Epoch 9/10
60000/60000 [==============================] - 25s 422us/sample - loss: 0.0621 - acc: 0.9836
Epoch 10/10
60000/60000 [==============================] - 25s 411us/sample - loss: 0.0586 - acc: 0.9846
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             multiple                  803840
_________________________________________________________________
dense_36 (Dense)             multiple                  1049600
_________________________________________________________________
dense_37 (Dense)             multiple                  1049600
_________________________________________________________________
dense_38 (Dense)             multiple                  1049600
_________________________________________________________________
dense_39 (Dense)             multiple                  1049600
_________________________________________________________________
dense_40 (Dense)             multiple                  10250
=================================================================
Total params: 5,012,490
Trainable params: 5,012,490
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 2s 217us/sample - loss: 0.1194 - acc: 0.9724
0.11940110468470085 0.9724
```

Figure 16: Change the number of neuron to 1024 per layer

In addition, we can plot the accuracy and loss for using the train dataset only.[15]

```python
import matplotlib.pyplot as plt

history = model.fit(x_train, y_train, epochs=3,validation_split=0.1, batch_size=32, verbose=1)

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

Figure 17: Codes for plotting

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/3
54000/54000 [==============================] - 5s 89us/sample - loss: 0.2711 - acc: 0.9231 - val_loss: 0.
2397 - val_acc: 0.9310
Epoch 2/3
54000/54000 [==============================] - 5s 90us/sample - loss: 0.2619 - acc: 0.9260 - val_loss: 0.
2381 - val_acc: 0.9317
Epoch 3/3
54000/54000 [==============================] - 5s 98us/sample - loss: 0.2553 - acc: 0.9274 - val_loss: 0.
2254 - val_acc: 0.9363
```

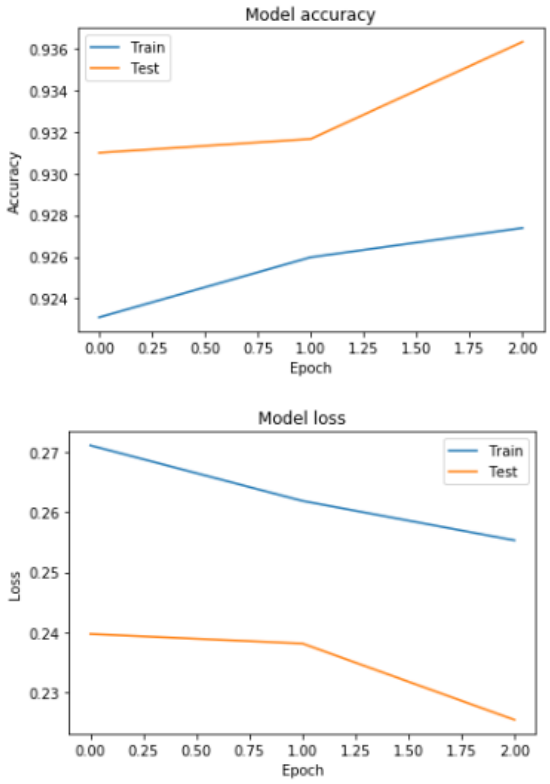
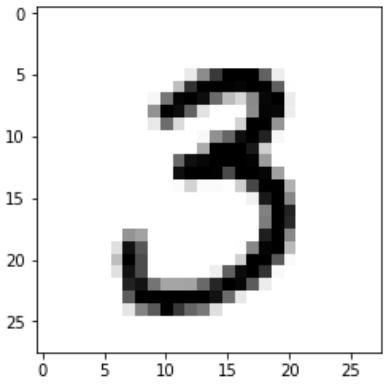


Figure 18: Model accuracy & Model loss

Finally, we can try our prediction model, as we can see, it predicts number 3 correctly.

```
In [54]: plt.imshow(x_test[90],plt.cm.binary)
         plt.show()
```




```
import numpy as np

print(np.argmax(predictions[90]))
```

```
3
```

Figure 19: Predication of the 91th 'image'

From what is above, I have introduced the steps about how to build my model and some concept about function I used in the model. I think my test model is quite good, when I saw its accuracy finally reach to 98%, however, there are some questions which still makes me confused, such like I add the dropout layer at first, but the results of this change did decrease the accuracy instead of improving it. In my opinion, it may be the overfitting problem of my model is not so obvious that the dropout layer just drops the correct output for some hidden layers. Another problem which confused me is whether I can improve the accuracy through increasing more neurons is that just because more neurons can deal with more complicated information. Admittedly, there are many problems I still not figure out, and I will change more characters to see the comparisons of the Multi-Layer Perceptron.

IV. Hand Written Digit Recognition Based on Support Vector Machine

i. Preliminary Work

Store and Visualize the Dataset

In order to read and visualize the Mnist data, I chose to read it as numpy and save it as csv format in the entire project.

The load-mnist function returns two arrays. The first one is a n x m-dimensional NumPy array(images), where n is the number of samples (number of rows) and m is the number of features (number of columns). The training data set contains 60,000

samples and the test data set contains 10,000 samples. Each picture in the MNIST data set consists of 28 x 28 pixels, each pixel being represented by a gray value. Here, we expand the 28 x 28 pixels into a one-dimensional row vector, which is the row in the image array (784 values per line, or each row represents a single image). The second array (tag) returned by the load-mnist function contains the corresponding target variable, which is the class label of the handwritten number (integer 0-9).[16]

Once the dataset is saved as a CSV file, we can reload them into the program using NumPy's 'genfromtxt' function.

Read the Dataset

Use the " model-selection -train-test-split" statement when randomly dividing the training set and the test set. It commonly used in cross-validation, the function is to randomly select train data and test data from the sample.[17]

Save the Print Statement in a Log file

Python's logging module provides a common logging system that can be used by third-party modules or applications. This module provides different log levels and can log in different ways, such as files, HTTP GET/POST, SMTP, Socket, etc., and even implement specific logging methods.[18]

Pickle

The pickle module is a module used to persist objects in Python. The so-called persistence of the object, that is, all the information such as the data type, storage structure, and storage content of the object is saved as a file for the next use.

For example, if you save an array as a file by pickle, then when you read the file by pickle next time, you still read an array instead of an array that looks like an array.

Save data to the data1.pkl file via the dump function in the pickle module.

The first parameter is the name of the object to save. The second parameter is the class file object file to which it is written. File must have a write() interface. file can be a file opened in 'w' mode. If protocol>=1, the file object needs to be opened in binary mode. The third parameter is the version of the protocol used for serialization, 0: ASCII protocol, the serialized object is represented by printable ASCII code; (1: old-fashioned binary protocol; new binary protocol introduced in 2:2.3 version, more than before More efficient; -1: Use the highest protocol supported by the current version. Protocols 0 and 1 are compatible with older versions of python. The default value of protocol is 0.) [19]

ii. Core algorithm

Basically, the idea of outer control loop is to input X origin and Y origin to gain the Phi origin and Theta origin instead of just input the value of Phi origin and Theta origin. As figure 14 shows, first we input the X origin with constant 1, and the value pass through the product module to convert it from ground coordinate to body coordinate, then we get output and we put this value into the sum module. After that phase, the X error shows and we put it into the PID controller, the output from the PID is the Phi origin. The process of how to achieve the Theta origin use the same method of Phi's

Linear Support Vector Machine

Svm is a two-class classifier. So that, it only answers questions that belong to a positive or negative class. The problems to be solved in reality are often multiple types of problems, such as text classification, such as digital recognition.

The learning of svm is actually to find the separation hyperplane, which is to solve the Lagrange factor in the above formula. The ai is the Lagrange factor and N is the number of samples.

$$w^* = \sum_{i=1}^{N} a_i^* y_i x_i$$

$$b^* = y_j - \sum_{i=1}^{N} a_i^* y_i (x_i \cdot x_j)$$

$$\sum_{i=1}^{N} a_i^* y_i (x_i \cdot x_j) + b^* = 0$$

$$f(x) = sign(\sum_{i=1}^{N} a_i^* y_i (x_i \cdot x_j) + b^*)$$

Figure 20. The Simple SVM model Function

The dual problem of introducing the slack variable is very different from the dual problem that is not introduced a slack variable. The range of the Lagrange factor ai is different. Ai $\geq$ 0 is not introduced into the slack variable, and 0 $\leq$ ai $\leq$ C is introduced. The slack variable is eliminated in the mathematical transformation. [20]

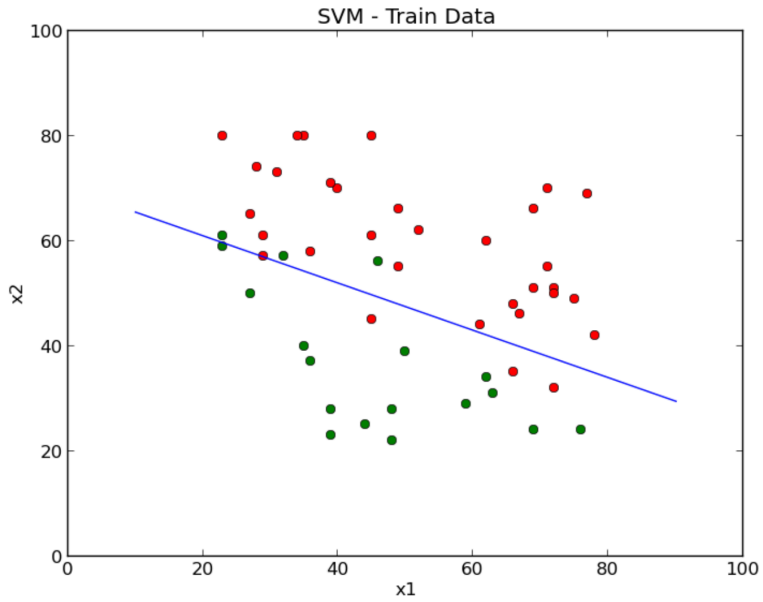


Figure 21. The two types of samples cannot be completely separated

$$\max_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$
$$\text{s.t.} \quad y_i(w\cdot x_i + b) \geq 1 - \xi_i, \qquad i = 1,2,\cdots,N$$
$$\xi_i \geq 0, \qquad i = 1,2,\cdots,N$$

$$\min_{a} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N} a_i$$
$$\text{s.t.} \quad \sum_{i=1}^{N} a_i y_i = 0$$
$$0 \leq a_i \leq C, \qquad i = 1,2,\cdots,N$$

$$b^* = y_j - \sum_{i=1}^{N} a_i^* y_i (x_i \cdot x_j)$$

Figure 22. Relaxation variable make the Function Change

Nonlinear SVM

Sometimes our data cannot be separated by linear functions, but nonlinearity can be divided. So after a certain mathematical mapping, the data becomes linearly divided. Let χ be the input space, specifying a mapping function φ(x). If for all x, z∈χ, the kernel function K(x, z) is satisfied. K(x,z)=φ(x)·φ(z). Where φ(x) · φ(z) is the inner

product. With a kernel function, you can talk about data xi mapping to higher dimensions, even infinite dimensions. [21]

$$\min_{a} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j y_i y_j (x_i \cdot x_j) - \sum_{i=1}^{N} a_i$$

$$\min_{a} \quad \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{N} a_i$$

$$b^* = y_j - \sum_{i=1}^{N} a_i^* y_i K(x_i \cdot x_j)$$

$$f(x) = sign(\sum_{i=1}^{N} a_i^* y_i K(x_i, x_j) + b^*)$$

Figure 23. Replace the minimized objective function in the dual problem with a kernel function

Parameters meaning

According to "**clf = svm.SVC**" the parameters are meaning: [22]

C: The penalty coefficient C, C of the objective function is larger, the higher the accuracy rate is tested for the training set, but the weaker the generalization ability is, the smaller the C value is, the less the penalty for misclassification is, the fault tolerance is allowed, and the generalization ability is stronger.

Shrinking: helps to calculate the effect

Verbose: show details... is to allow redundant output

Coef0: constant term of the kernel function, useful for poly and sigmoid kernel functions

Degree: the dimension of the polynomial poly kernel function, the default is 3, other kernel functions will ignore

Gamma: not clear, but the effect is obvious after tuning; kernel function parameters of rbf, poly and sigmoid

Tol: the error value of the stop training, the default is 1e-3

Result Analysis

Confusion Matrix

When it comes to classification problems, we often need to analyze the experimental results by visualizing the confusion matrix to get the reference ideas. The row is true value and the column is the predicted value.[23]

```
Confusion Matrix:
 [[546   0   4   2   0  13  15   0  11   3]
 [  0 644   2   2   1   5   3   2   8   2]
 [  7  13 508  11   3   4  21   2  37   1]
 [  6   3  18 502   1  26   9   7  35   2]
 [  3   2   2   1 526   5  12   2  22  26]
 [  5   1   3  22   4 426  21   0  32   6]
 [  1   0   2   0   2  12 577   0   5   0]
 [  1   1  15   7   5   3   0 560  10  26]
 [  3  13   7   7   4  36  15   1 480   9]
 [  2   4   2  13  27   9   1  21  19 500]]

Confusion Matrix for Test Data:
 [[ 921    0    4    2    0   11   28    3    9    2]
 [   0 1120    3    1    0    1    5    0    4    1]
 [   7   17  844   14    5   10   34   10   87    4]
 [  12    2   22  813    1   68   10   10   67    5]
 [   3    5    6    3  859    5   31    4   19   47]
 [   7    4    3   21    2  751   40    9   49    6]
 [   5    3    2    0    5   16  921    0    6    0]
 [   0    8   37    9    9    8    1  910    9   37]
 [   4   16    9   11    9   61   18    7  831    8]
 [   6    7    2   17   45   13    0   28   69  822]]
```

Figure 24. The confusion function of SVM

GGplot

Ggplot is a landscaping of matplot graphics. From the visual confusion function, we can visually see that 1 has the highest recognition accuracy rate and 5 has the lowest recognition accuracy rate.

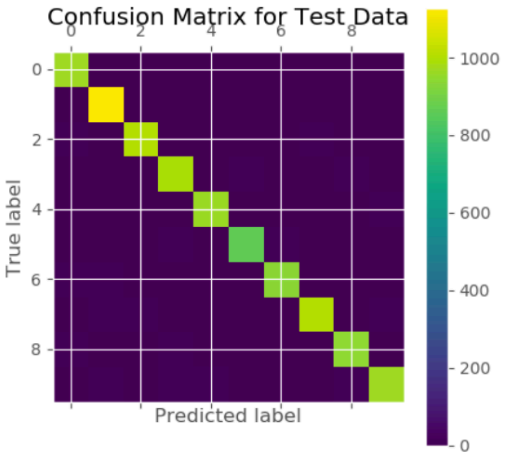


Figure 25. The SVM confusion function

SVC：

There is not much difference on 0,1,2,7

but the accuracy of other numbers has been greatly improved.

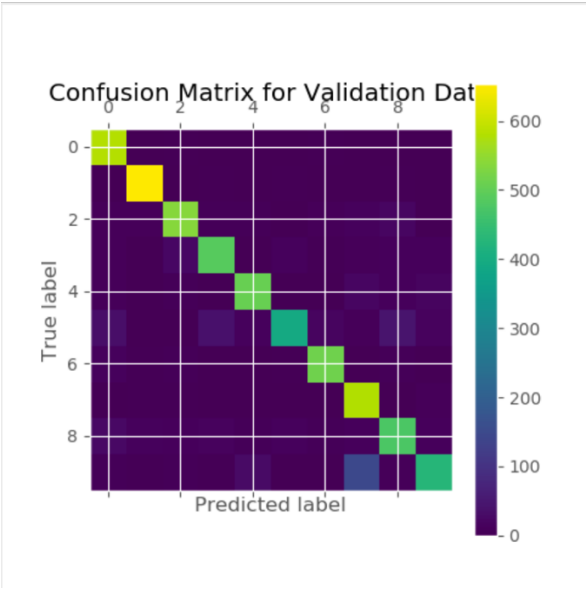


Figure 26. The Linear SVM confusion function

Linear SVC：

1 has the highest accuracy rate

5 has the lowest correct rate and is easily recognized as 0, 3, 8

9 is easily identified as 4 & 7

Conclusion:

From1. 9 Test of different parameters

	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9
C	0.5	0.1	0.1	0.01	0.01	0.01	0.01	0.8	0.01
gamma	0.05	auto	0.1	0.2	0.3	0.4	0.05	0.05	0.05
tol	0.05	0.01	0.01	0.05	0.05	0.05	0.05	0.1	0.1
Train Accuracy	0.981	0.981	0.9773	0.981	0.981	0.981	0.981	0.98116	0.98116
Test Accuracy	0.9785	0.9783	0.9771	0.9785	0.9785	0.9785	0.9785	0.9785	0.9785

Test8 and 9 are the best parameter sets. From Test4 to Test8, we can know that the size of gamma does not affect the correct rate of SVM when the data is stable.

We can conclude by comparing Test2 to Test8. gama=0.1 is an interference data. And by comparing Test7 and Test9, you can get the conclusion that increasing the value of tol can increase the correct rate slightly.

The Linear Support Vector Classification Accuracy is 0.869. This is as low as we expected.

Reference

[1] "THE MNIST DATABASE," *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 20-Aug-2019].
[2] R. Vaidya, D. Trivedi, S. Satra, and P. M. Pimpale, "Handwritten Character Recognition Using Deep-Learning," *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018.
[3] "The One Hot Technique in Finite-State Machine Design," *FSM-Based Digital Design Using Verilog HDL*, pp. 105–143, Jan. 2008.
[4]M. T. Hagan, H. B. Demuth, M. H. Beale, and Jesús Orlando De, *Neural network design*. S. l.: s. n., 2016.
[5] "Google Cloud Platform," *GitHub*. [Online]. Available: https://github.com/GoogleCloudPlatform. [Accessed: 20-Aug-2019].
[6] "Ghosh4AI - Overview," *GitHub*. [Online]. Available: https://github.com/Ghosh4AI. [Accessed: 20-Aug-2019]
[7] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 19-Aug-2019].
[8] "How to create a basic MLP classifier with the Keras Sequential API," *Machine Curve*, 27-Jul-2019.
[9]D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep Big Multilayer Perceptrons for Digit Recognition," in *Neural Networks: Tricks of the Trade*, vol. 7700, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 581–598.
[10] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.
[11]"DeepLearning tutorial（3）MLP 多层感知机原理简介+代码详解 - wepon 的专栏 - CSDN 博客." [Online]. Available: https://blog.csdn.net/u012162613/article/details/43221829. [Accessed: 19-Aug-2019].
[12] "数据处理——One-Hot Encoding - null 的专栏 - CSDN 博客." [Online]. Available: https://blog.csdn.net/google19890102/article/details/44039761. [Accessed: 19-Aug-2019].

[13] "categorical_crossentropy VS. sparse_categorical_crossentropy," *Jovian's Notes*, 17-Feb-2018. [Online]. Available: https://jovianlin.io/cat-crossentropy-vs-sparse-cat-crossentropy/. [Accessed: 19-Aug-2019].

[14] "损失函数 Losses - Keras 中文文档." [Online]. Available: https://keras.io/zh/losses/#sparse_categorical_crossentropy. [Accessed: 19-Aug-2019].

[15]"MNIST 机器学习入门 - TensorFlow 官方文档中文版 - 极客学院 Wiki." [Online]. Available: http://wiki.jikexueyuan.com/project/tensorflow-zh/tutorials/mnist_beginners.html. [Accessed: 19-Aug-2019].

[16] "详解 MNIST 数据集 - Liu-Cheng Xu - CSDN 博客." [Online]. Available: https://blog.csdn.net/simple_the_best/article/details/75267863. [Accessed: 19-Aug-2019].

[17] "sklearn.model_selection.train_test_split 划分训练集和测试集 - I am what i am - CSDN 博客." [Online]. Available: https://blog.csdn.net/liuxiao214/article/details/79019901. [Accessed: 19-Aug-2019].

[18] "log 输出日志 - 浩子的博客 - CSDN 博客." [Online]. Available: https://blog.csdn.net/u013851082/article/details/72782943. [Accessed: 19-Aug-2019].

[19] "SVM 支持向量机 - 刘洪江的流水帐." [Online]. Available: http://liuhongjiang.github.io/tech/blog/2012/12/26/svm/. [Accessed: 19-Aug-2019].

[20] "1.4. Support Vector Machines — scikit-learn 0.21.3 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/svm.html#svm-classification. [Accessed: 19-Aug-2019].

[21] "sklearn.svm.SVC — scikit-learn 0.21.3 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 19-Aug-2019].

[22] "Confusion matrix — scikit-learn 0.21.3 documentation." [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html. [Accessed: 19-Aug-2019].

[23] "样式美化(matplotlib.pyplot.style.use) - You_are_my_dream 的博客 - CSDN 博客." [Online]. Available: https://blog.csdn.net/You_are_my_dream/article/details/53464662. [Accessed: 19-Aug-2019]