Research on MNIST handwritten digits dataset by applying MLP

In this part, we will create several kinds of simple neural network and compare their performance on the MNIST handwritten digits dataset. The MNIST dataset is a classic problem for getting started with neural networks and for the aim for all the networks we tested is the same: get the recognition accuracy, then take an input image (28x28 pixels) of a handwritten single digit (0–9) and classify the image as the appropriate digit.

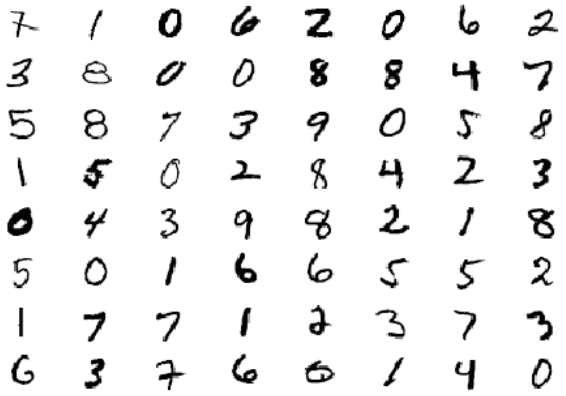


Figure 1: A random selection of MNIST digits.

After figure out what we are going to do, we need to prepare the test data. The data can be gathered from the website http://yann.lecun.com/exdb/mnist/, but it need some unzipping and extracting steps before we achieve the pure data. However, 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.[1]

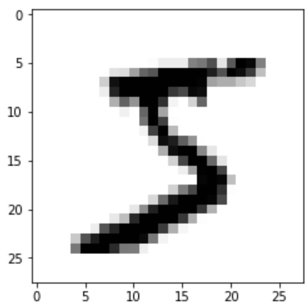
```
mnist = keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

Figure 2: import data

Once the dataset has been imported, we just need to format it appropriately for our

neural network. In this part of the article, the Multi-Layer Perceptrons are focused merely. They do no recognize such things as ‘2D’. It means the input data must be a vector. Instead of several 28 * 28 pixels image, we translate them that are all same length 784 (28 *28 = 784). [5]

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 3: 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 4: 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.[6]

```
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 4: 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.[7]

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

Figure 5: 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 6: 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 [8]:

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

Figure 7: 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 [6]. 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.[3]

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.[2][3][4]

```
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 6: 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. [9]

```
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 7: Change the number of neuron to 1024 per layer

In addition, we can plot the accuarcy and loss for using the train dataset only.[9]

```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 8: 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
```

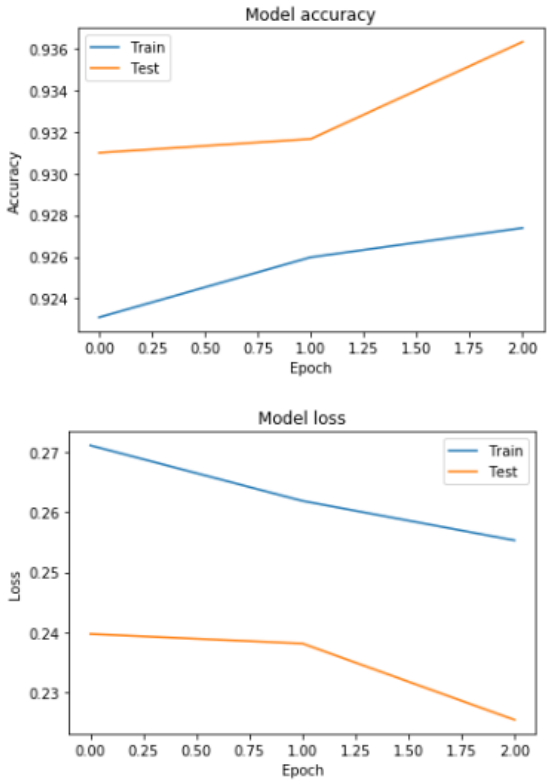
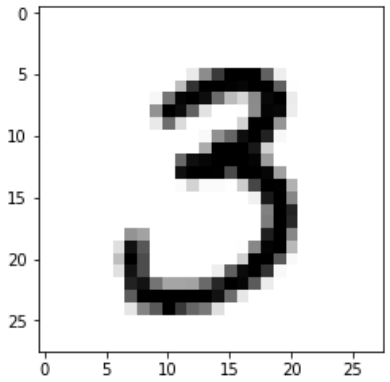


Figure 9: Model accuracy & Model loss

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 10: Predication of the 91th ‘image’

Discussion

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.

Reference:

[1] “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 19-Aug-2019].

[2] “How to create a basic MLP classifier with the Keras Sequential API,” *Machine Curve*, 27-Jul-2019.

[3]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.

[4]F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.

[5]“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].

[6] “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].

[7] “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].

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

[9]“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].