A method using MLP to recognize and classify handwritten characters in Python

Tianyu Zhou,34196994, Student, Department of Aerospace and Mechanical Engineering, University of Florida

Yijun Guo, 02427984, Student, Department of Electrical and Computer Engineering, University of Florida

Rui Xiao, 98519256, Student, Department of Aerospace and Mechanical Engineering, University of Florida

***Abstract*— This report shows a method using MLP (Multi-layer Perceptrons) and other machine learning algorithm to recognize and classify hand-written letters. The result using different method with different parameters are shown and compared in this report. The data are provided by the instructor, includes handwritten single letters and handwritten characters from pieces of mail. With proper parameters set for the MLP model, an accuracy over 90% for test data have been obtained.**

***Index Terms*— Python, pattern recognition, MLP, Machine learning, PCA, activation function.**

I. Introduction

THIS project is about recognition of characters written by hand. To be specific, to recognize "a","b", "c", "d" , "h", "i", "j" and "k", and labels their image accordingly from 1 to 8. All other letters or images without these eight characters should be labeled as "-1" which stands for unknown. The algorithm we can use are PG (Probabilistic Generative), KNN (K-Nearest Neighbors), K-mean (K-Means clustering) and MLP (Multi-layer Perceptron). Since MLP have several parameters that can be adjusted to obtain the highest accuracy, also people have used MLP to recognize characters [1] [2], we choose MLP as the method of this project.

After choosing MLP as the algorithm, some parameters need to be determined: initial data size (which determines the number of perceptron in input layer), number of hidden layers, number of perceptron in each hidden layers, number of perceptron in output layer, value of learning rate and activation function we use. Those parameters will be determined below. Also some data preprocessing methods such as PCA or data whitening can be used.

II. Implementation

We use python to realize our algorithm, however, before writing the code we need to choose the value of the parameters listed in Introduction (some parameters can be determined only after some experiments, we will set an initial value for them and modify it according to the result of the experiments):

A. *Initial data size*

The data provided contains images of handwritten characters from "a" to "d" and "h" to "k" with labels from 1 to 8 corresponding to the character. However, the image in numpy form is not in the same size, which makes it impossible to compute the results using the whole unprocessed dataset. As a result, some preprocessing of the image size need to be done.

Taking a look at the data set, we found that the maximum length of image is 60 pixels and the maximum width is also 60 pixels. So we reshape all images into 60X60 using "transform.reshape" function in skimage package.

B. *Number of hidden layers*

The number of hidden layers will affect the running time and the result of the model. According to Universal Approximation Theorem, "a single hidden layer is sufficient for a multilayer perceptron to compute a uniform ϵϵ approximation to a given training set - provided you have the *right* number of neurons and the *right* activation function"[3]. But a single hidden layer neural network cannot guarantee best performance. Considering more hidden layers will lead to higher possibility of over-fitting. We choose a neural network with 2 hidden layers.

C. *Number of perceptron in each hidden layers*

The number of perceptron in the first hidden layer must be less than N-1 where N stands for the number training samples. Otherwise the result will be unrelated with the input data. Generally speaking, the more number of perceptron in each hidden layers, the more likely we get high accuracy with the training data. However, we have to take overtraining into consideration. There is no way to get the best number without doing some experiment, so we tried different numbers of perceptron in each hidden layers and compared the accuracy using cross-validation. In the end, we pick the number of perceptron in each hidden layers to be 512 and 128 for the two hidden layers.

D. Number of perceptron in output layer

The number of perceptron in output layer determines how to calculate the result of the neural network. Just like the number of perceptron in hidden layers, the size of output layer needs some experiment too. After a few trails we pick 10 perceptron in the output layer.

E. Activation function

The activation function for hidden layer and output layer can be different. We pick Relu as the activation function for hidden layers because Relu can avoid Vanishing Gradient Problem (while other common function such as softmax cannot). Since we have 9 different labels in our classify process, we need to choose a function that can provide a wide range of outputs. In the end we pick Logistics to be the output layer's activation function.

F. Data preprocessing

If the data is too large to compute or has redundant dimensions, PCA is the best way to reduce the size of data without losing those important aspects of data. Also data whitening which contains the process of PCA can be an option. However, those preprocessing method are not necessary.

As a result, the experiment will contain:

1. Reshape the data and make it applicable for the computation. Preprocess the data and split for cross-validation.
2. Build the neural network model.
3. Using the model and training data to train the model and keep optimizing the model until the training reaches the iteration times.
4. Test the model with the test data and compute the accuracy.

III. Experiments

The goal of the experiment is to find a MLP model that can get the best classification result of 9 different groups of characters. The data given are images of 8 characters in npy form and their labels. The model should output labels with 8 letters and the other contribute to unknown.

A. Data modification

The data provided are in different sizes and only have 8 groups. Firstly we reshape all those data into 60*60 pixels since it is the largest size among all images given. **Then we generate some 60*60 pixels matrixes of 0 and 1, add them into the data and label them as -1 (unknown)**. Those 0 and 1s are randomly generated for that we do not know what those unknown image should be. This method may be not good enough, because there will be half 0 and half one in 60*60 matrix. However, the true condition of the letter will not look like that. Take some images of other letters should be a good way, but it will cost a lot of time to build a large data library. For this project, we don't need to build a very complex system which will cost more time for testing. So we choose the easy way to build the unknown library. Then the data and corresponding labels then are split into train and test group for cross-validation.

B. Build a model

To find the best model we need to adjust the parameters in the model to get the best performance. But first we need a model that can run. We use "torch" package in Python to build the MLP model.

We use "optim.SGD" to pick SGD (stochastic gradient descent) as the optimizer. This optimizer will use mini-batch to perform back-propagation.

We use "nn.CrossEntropyLoss" to set CrossEntropy as loss function, then we can optimize the model by calling the attribute function of loss function and optimizer in Python.

C. Adjust parameters

Since there are lot parameters that we can adjust, we will change a parameter with all other parameters fixed. After we get a parameter value that performs well, we fix this parameter to that value and change another parameter. **Then we will determine all those parameters one at a time.**

1. *Activation function*:

Activation Function (hidden layer + output layer)	Train accuracy	Test accuracy
Relu + Logistics	0.99933	0.90765
Relu + Relu	0.99821	0.90357
Logistics + Logistics	0.99509	0.89588
Relu + Softmax	0.92795	0.87188

Table 1 comparison between different activation functions

From Table 1 we can see Relu for hidden layer and Logistics for output layer will give us the best result with all other parameters fixed. **So we choose Relu + Logistics as our activation functions of the model.**

2. *Value of learning rate:*

The learning rate will affect the time cost for training and also the accuracy of the model.

Learning rate	Train accuracy	Test accuracy
0.1	0.99866	0.90448
0.01	0.90162	0.86057
0.4	0.99174	0.89316

Table 2 comparison between different Learning rate values

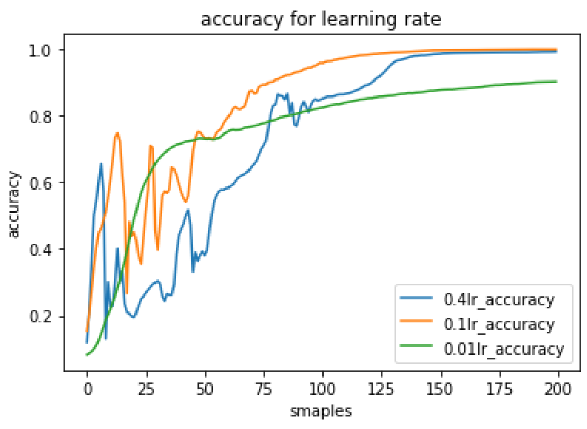


Figure 1 Comparison between different Learning rate values

We can see from Table 2 and Figure 1 if the learning rate is too large (E.g. 0.4) the accuracy will bounce a lot and the test accuracy will be low. If the learning rate is small (E.g. 0.01) the model will stop at the local optima so the accuracy will be terrible. To avoid those two situations, **we pick a learning rate equals to 0.1 which is proper**.

3. *Number of perceptron in output layer*

We have 9 different groups need classification, so an algorithm should be designed to set correct labels for input data. **We train our model using an algorithm that will find the maximum value among all output neurons and set the number of that neuron as the label for the input [4]**. For example, if the max value appears in No.2 neuron in the output layer, the output label will be set to 2. The unknown label -1 will be set as 0 in this model and all "0"s in the predicted labels will be relabeled as -1 in the end.

For this labeling algorithm, we need at least 9 output neurons. The neurons after the first 9th will not affect the result since the train process will make the maximum lies in those useful neurons to minimize the loss function. As a result, we set the number of perceptron in output layer to be 10 so we have one spare for future usage.

4. *Number of hidden layers and number of perceptron in each hidden layer*

Number of hidden layers	Train accuracy	Test accuracy
1	0.99888	0.90991
2	0.99821	0.90765
3	0.99888	0.90402

Table 3 Comparison between different Numbers of hidden layers

The number of hidden layers does not really matter according to the result shown in Table 2. That probably because the input layer has 3600 perceptron and there are hundreds of neurons in the hidden layer, as a result, the model is complex enough so it will not be affected by the number of hidden layers.

However, the more hidden layers we have the more time the calculation will cost and the more likely we are going to over-train our model. 1 hidden layer may not be enough to solve a different task. **So we choose to have 2 hidden layers in our model**

Number of perceptron in each hidden layer	Train accuracy	Test accuracy
1024,128	0.99933	0.90946
512,128	0.99888	0.91036
128,32	0.99843	0.90448

Table 4 Comparison between different numbers of perceptron in each hidden layer

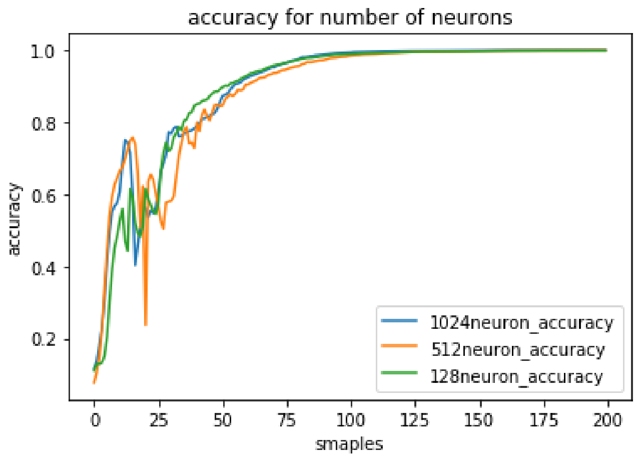


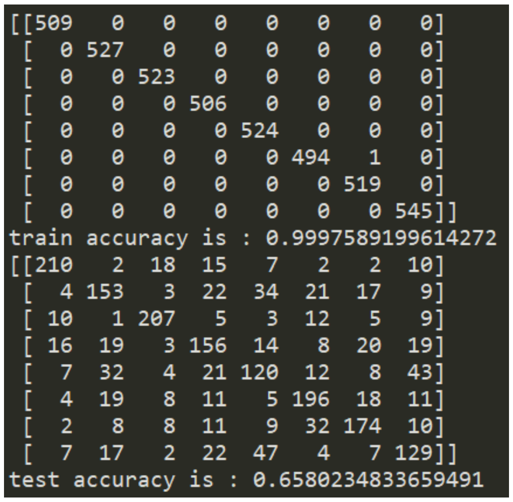
Figure 2 Comparison between numbers of perceptron in each hidden layer

From Table 4 and Figure 2 that both train accuracy and test accuracy of different numbers of neurons in hidden layer does not change much. The reason maybe the same as described above, that is, because the model is complex enough and will not be influenced by the number of neurons.

In Table 4 we can see that 512 neurons in the first hidden layer and 128 neurons in the second hidden layer will get slightly higher test accuracy. **So we choose a model with 512 neurons in the first hidden layer and 128 neurons in the second hidden layer.**

5. *PCA and data whitening*

PCA and data whitening are useful data preprocessing methods. When running those experiments above, our running time is not very long and the test accuracy is relatively high. So PCA and data whitening is not necessary for this project. We used data whitening hoping we can get an even better result in a shorter time. The results are shown below:



```
[[509   0   0   0   0   0   0   0]
 [  0 527   0   0   0   0   0   0]
 [  0   0 523   0   0   0   0   0]
 [  0   0   0 506   0   0   0   0]
 [  0   0   0   0 524   0   0   0]
 [  0   0   0   0   0 494   1   0]
 [  0   0   0   0   0   0 519   0]
 [  0   0   0   0   0   0   0 545]]
train accuracy is : 0.9997589199614272
[[210   2  18  15   7   2   2  10]
 [  4 153   3  22  34  21  17   9]
 [ 10   1 207   5   3  12   5   9]
 [ 16  19   3 156  14   8  20  19]
 [  7  32   4  21 120  12   8  43]
 [  4  19   8  11   5 196  18  11]
 [  2   8   8  11   9  32 174  10]
 [  7  17   2  22  47   4   7 129]]
test accuracy is : 0.6580234833659491
```

Figure 3 confusion matrix and accuracy for training (upper) and testing (lower) using PCA

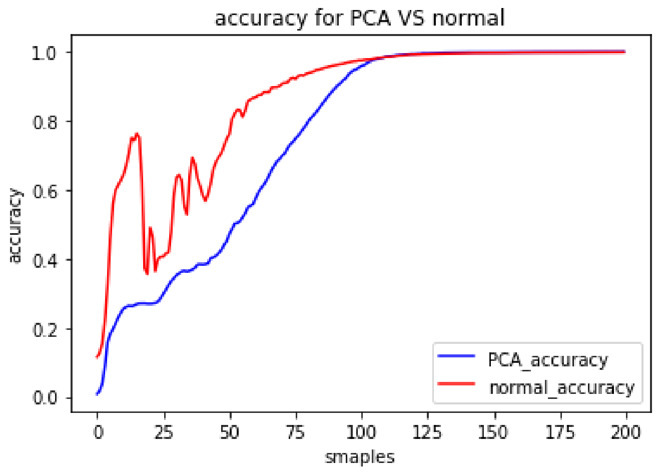


Figure 4 Train accuracy using PCA VS not using PCA (normal)

We keep 40*40 pixels as the principal components. Figure 3&4 shows that PCA works well with the training data but performs badly with testing data.

The reason may be that during PCA we reshape the data into a vector and calculate the mean and variance of all data. Then we use mean and variance to get some new data. However, the origin data are 0 and 1 while the new data will not be 0 and 1. There will be a big difference between old and new data in mathematical logic. And this process may result in a loss of the original geometry information. So, although the train data performs will, the accuracy of the test data will be low. If we can find a way to use PCA to deal with the bool type data or something else without losing any original information, it may leads to a better result.

IV. Conclusions

After several experiments, our model result in a MLP model with 2 hidden layers (512 neurons in the first and 128 in the second); activation function for hidden layers is Relu and activation function for output layer is Logistics; optimizer is SGD and loss function is CrossEntropy function; learning rate is set to be 0.1. Labeling algorithm is described in the section "*Number of perceptron in output layer*"

However, since those parameters are picked one at a time. The combination of those parameters may not be the global optimization of the model.

V. References

[1] Amit Choudhary, Savita Ahlawat, Rahul Rishi, Vijaypal Singh Dhaka, "Performance analysis of feed forward MLP with various activation functions for handwritten numerals recognition", Computer and Automation Engineering (ICCAE) 2010 The 2nd International Conference on, vol. 5, pp. 852-856, 2010.

[2] D.S. Yeung, "A neural network recognition system for handwritten Chinese character using structure approach", *Proceeding of the World Congress on Computational Intelligence*, vol. 7, pp. 4353-4358, June 1994.

[3] Alina Zare, EEL 5840 Lecture note #27

[4] https://www.jianshu.com/p/65aed5b33cf2