

Assignment 1: Neural Network

SCHOOL OF ADVANCED TECHNOLOGY

INT410: Advanced Pattern Recognition

Mengfan Li

ID number: 2032048

2020 Fall

Abstract

Assignment1 constructs a three-layer neural network with a hidden layer by using the principle of multi-layer perceptron (MLP) and the principle of back-propagation neural network (BP). Through the three-layer neural network model, the pattern classification of complex IRIS is carried out, and the accuracy is calculated.

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment.

Content

| | | |
|------------|---------------------------------------------------|----|
| 1 | Introduction..... | 1 |
| 2 | Materials..... | 1 |
| 3 | Methods, result and discussion | 1 |
| 3.1 | Objectives..... | 1 |
| 3.2 | Estimation of Classification Methods | 2 |
| 3.2.1..... | | 2 |
| 3.2.2..... | | 3 |
| 3.3 | MLP Algorithm..... | 4 |
| 3.3.1..... | | 4 |
| 3.3.2..... | | 5 |
| 3.3.3..... | | 6 |
| 3.3.4..... | | 7 |
| 3.3.5..... | | 8 |
| 3.3.6..... | | 8 |
| 3.3.7..... | | 8 |
| 3.3.8..... | | 10 |
| 4 | improvement | 10 |
| 4.1 | About the number of iteration time | 10 |
| 4.2 | About the number of hidden layer node number..... | 11 |
| 4.3 | About the seed effect the model accuracy | 13 |
| 5 | Conclusion | 14 |

1 Introduction

The main purpose of this study is that the use of three layer neural network, a reverse main technology, neural network for the classification of irises, each data, there are five characteristics, one of which is a specific pattern, we need through the first four characteristics, to predict the final design, and send the data set of pattern, comparing the characteristic value and the accuracy. Other things you'll need to know are activation function tan, gradient descent, Python programming, and reading and writing datasets.

2 Materials

Use PyCharm software and related libraries to write Python programs.

3 Methods, result and discussion

3.1 Objectives

Download the iris dataset. The dataset is a [150,5].

the code of download the data:

```
import pandas as pd
fresh = pd.read_csv('iris.data')
```

this is the result:

```
[8]: import pandas as pd
fresh = pd.read_csv('iris.data')
fresh
```

```
[8]:
```

| | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
|-----|-----|-----|-----|-----|----------------|
| 0 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 2 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 3 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 4 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| ... | ... | ... | ... | ... | ... |
| 144 | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| 145 | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| 146 | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| 147 | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| 148 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

149 rows × 5 columns

3.2 Estimation of Classification Methods

3.2.1

(5 marks) Read the Flare dataset into a list and shuffle it with the `random.shuffle` method. Hint: fix the random seed (e.g. `random.seed(17)`) before calling `random.shuffle`

Set the seed

```
# set a random seed, let the random stabilization
random.seed(17)
```

Set the data random:

Since I added a data matrix after the original data to represent the specific type, When Randomly

shuffling the data, I directly arranged the following data matrix randomly.

Code:

```
print(data)
# let the data random
random.shuffle(data)
print('New data')
print(data)
```

Result:

the no random dataset 1——5 data is

```
[[4.9, 3.0, 1.4, 0.2], [1, 0, 0]]
[[4.7, 3.2, 1.3, 0.2], [1, 0, 0]]
[[4.6, 3.1, 1.5, 0.2], [1, 0, 0]]
[[5.0, 3.6, 1.4, 0.2], [1, 0, 0]]
[[5.4, 3.9, 1.7, 0.4], [1, 0, 0]]
```

the have random dataset 1——5 data is:

```
[[0, 0, 4]], [[0.4, 2.7, 2.7, 4.0]], [[0, 0, 4]], [[0.2, 2.0, 2.4, 4.0]], [[0, 0, 4]]
#####
[[[6.2, 3.4, 5.4, 2.3], [0, 0, 1]], [[6.1, 2.9, 4.7, 1.4], [0, 1, 0]], [[6.5, 2.8, 4.6, 1.5], [0, 1, 0]], [[6.5, 3.0, 5.5, 1.8], [0, 0, 1]], [[5.0, 2.0, 3.5, 1.0], [0, 1, 0]], [[6.3, 2.7, 4.9, 1.8], [0, 0, 1]], [[4.4, 3.0, 1.3, 0.2],
```

So, have be random

3.2.2

(5 marks) Split the dataset into five parts to do cross-fold validation: Each of 5 subsets as used as test set and the remaining data was used for training. The 5 subsets were used for testing rotationally for evaluating the classification accuracy.

Since there are a total of 150 sets of data, the data is divided into five equal parts, each with 30 data. Then according to the strategy of 4 training and 1 test, there are 120 training sets and 30 test sets. Because the data has been randomized in the previous step, the data set is different for each time,

so the training set is different for each time. Cross-validation can be achieved, and the accuracy at the end is also an average accuracy.

Code:

```
#let the data random
random.shuffle(data)

# 0:99    ---> train
# 100:150 ---> test
training = data[0:119]
test = data[120:]
```

3.3 MLP Algorithm

3.3.1

(10 marks) The input feature vector is augmented with the 1 since

$$w^T + w_0 = [w^T \quad w_0] \begin{bmatrix} x \\ 1 \end{bmatrix}$$

Code:

```
# to add a offset node
self.input_data = input_data + 1

# to add a offset node
self.hidden_data = hidden_data + 1

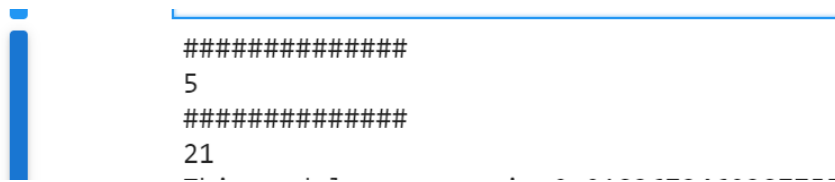
print('#####')
print(self.input_data)

print('#####')
print(self.hidden_data)
```

Result:

Since there are 4 nodes in the input layer and 20 nodes in the hidden layer, there are 5 in the

input layer and 21 in the hidden layer after adding an offset node



```
#####
5
#####
21
-----
```

3.3.2

(10 marks) The label \mathbf{l}_n of the n-th example is converted to a L dimensional vector \mathbf{t}_n follows (K is the number of the classes)

$$t_{nk} = \begin{cases} +1, & k = l \\ 0, & k \neq l \end{cases}$$

In this step, I will generate a diagonal matrix, then define [1 0 0] as the first, [0 1 0] as the second, and [0 0 1] as the third, and then expand it to the end of the corresponding data set to mark the corresponding category.

Code:

```
#add the feature to the data
for i in range(len(fresh_data_feature)):
    feature = []
    feature.append(list(fresh_data_feature[i]))
    if fresh_data[i][4] == 'Iris-setosa':
        feature.append([1, 0, 0])
    elif fresh_data[i][4] == 'Iris-versicolor':
        feature.append([0, 1, 0])
    else:
        feature.append([0, 0, 1])
    data.append(feature)
    print(feature)
```

Result:

Iris-setosa:

original data

```
2  4.9,3.0,1.4,0.2,Iris-setosa
```


process data:

```
[[4.9, 3.0, 1.4, 0.2], [1, 0, 0]]
```

Iris-versicolor

original data:

```
51 7.0,3.2,4.7,1.4,Iris-versicolor
```

process data:

```
[[7.0, 3.2, 4.7, 1.4], [0, 1, 0]]
```

Iris-virginica

original data

```
101 6.3,3.3,6.0,2.5,Iris-virginica
```

process data:

```
[[6.3, 3.3, 6.0, 2.5], [0, 0, 1]]
```

3.3.3

(10 marks) Initialize all weight w_{ij} of MLP network such as

$w_{ij} \in \left[-\sqrt{\frac{6}{D+1+L}}, \sqrt{\frac{6}{D+1+L}} \right]$ where D and L is the number of the input nodes and the output

nodes, respectively.

Code:

```
# set up a weight matrix
self.winput = makeMatrix(self.input_data, self.hidden_data)
self.woutput = makeMatrix(self.hidden_data, self.output_data)
# set the weight random ---> [(-(6/(i + 1 + j)) ** 0.5), (6/(i + 1 + j)) ** 0.5)]
for i in range(self.input_data):
    for j in range(self.hidden_data):
        self.winput[i][j] = rand(-(6/(i + 1 + j)) ** 0.5, (6/(i + 1 + j)) ** 0.5)
for j in range(self.hidden_data):
    for k in range(self.output_data):
        self.woutput[j][k] = rand(-(6/(i + 1 + j)) ** 0.5, (6/(i + 1 + j)) ** 0.5)
```

3.3.4

(20 marks) Choose randomly an input vector x to network and forward propagate through the network

$$a_j = \sum_{i=0}^d w_{ji}^{(1)} x_i$$
$$z_j = \tanh(a_j)$$
$$y_k = \sum_{j=0}^k w_{kj}^{(2)} z_j$$

The error rate is $\frac{1}{2} \sum_{l=1}^L (y_l - t_l)^2$ for the example x .

MLP code:

```
def data_updata(self, inputs):

    # activation the input layer
    for i in range(self.input_data - 1):
        self.ainput[i] = inputs[i]

    # activation the hidden layer
    for j in range(self.hidden_data):
        sum = 0.0
        for i in range(self.input_data):
            sum = sum + self.ainput[i] * self.winput[i][j]
        self.ahidden[j] = tanh(sum)

    # activation the output layer
    for k in range(self.output_data):
        sum = 0.0
        for j in range(self.hidden_data):
            sum = sum + self.ahidden[j] * self.woutput[j][k]
        self.aoutput[k] = tanh(sum)

    return self.aoutput[:]
```

Error rate code:

```
# calculation error
error += 0.5 * (targets[k] - self.aoutput[k]) ** 2
return error
```

3.3.5

(10 marks) Evaluate the δ_k for all output units

$$\delta_k = y_k - t_k$$

Code:

```
for k in range(self.output_data):
    error = targets[k] - self.aoutput[k]
    output_error_deltas[k] = dtanh(self.aoutput[k]) * error
```

3.3.6

(10 marks) Backpropagate the δ 's to obtain δ_j for each hidden unit in the network

$$\delta_j = \tanh(a_j)' \sum_k w_{kj} \delta_k = (1 - z_j^2) \sum_k w_{kj} \delta_k$$

Code:

```
for k in range(self.output_data):
    error = targets[k] - self.aoutput[k]
    output_error_deltas[k] = dtanh(self.aoutput[k]) * error
```

3.3.7

(10 marks) The derivative with respect to the first-layer and the second-layer weights are given

by

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad , \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Code:

```
# update the output layer weight
for j in range(self.hidden_data):
    for k in range(self.output_data):
        change = output_error_deltas[k] * self.ahidden[j]
        self.woutput[j][k] = self.woutput[j][k] + Learn_rate * change

# update the input layer weight
for i in range(self.input_data):
    for j in range(self.hidden_data):
        change = hidden_deltas[j] * self.ainput[i]
        self.winput[i][j] = self.winput[i][j] + Learn_rate * change
```

The framework of MLP algorithm is as follows, where $\eta = 0.001$ and $K = 20$ is the number of hidden nodes. Note that K , η , T are the hyperparameters of the network.

(a) The learning rate $\eta = 0.001$

Code:

```
# train module
def train(self, patterns, train_number, Learn_rate=0.001):
    for i in range(train_number):
        error = 0.0
        for p in patterns:
            inputs = p[0]
            targets = p[1]
            self.data_updata(inputs)
            error = error + self.BP(targets, Learn_rate)
```

(b) The hidden layer number $K = 20$

Code:

```
# input layer have 4 points
# hidden layer have 20 points
# output layer have 3 points
Model = T_L_Network(4, 20, 3)
```

(c) The T is the total iteration time. I set $T = 200$

Code:

```
Model.train(training, train_number=200)
```

3.3.8

(10 marks) In the test stage, the test example x is forwarded into the network to obtain the output $y_{L \times 1}$ and then assigned to the label with the maximum output value.

Code:

```
# test module
def test(self, patterns):
    count = 0
    for p in patterns:
        target = iris_lable[(p[1].index(1))]
        result = self.data_updata(p[0])
        index = result.index(max(result))
        count += (target == iris_lable[index])
    accuracy = float( count / len(patterns) )
    print('This model accuracy is:'+str(accuracy))
```

Result:

This model accuracy is:0.9310344827586207

4 improvement

4.1 About the number of iteration time

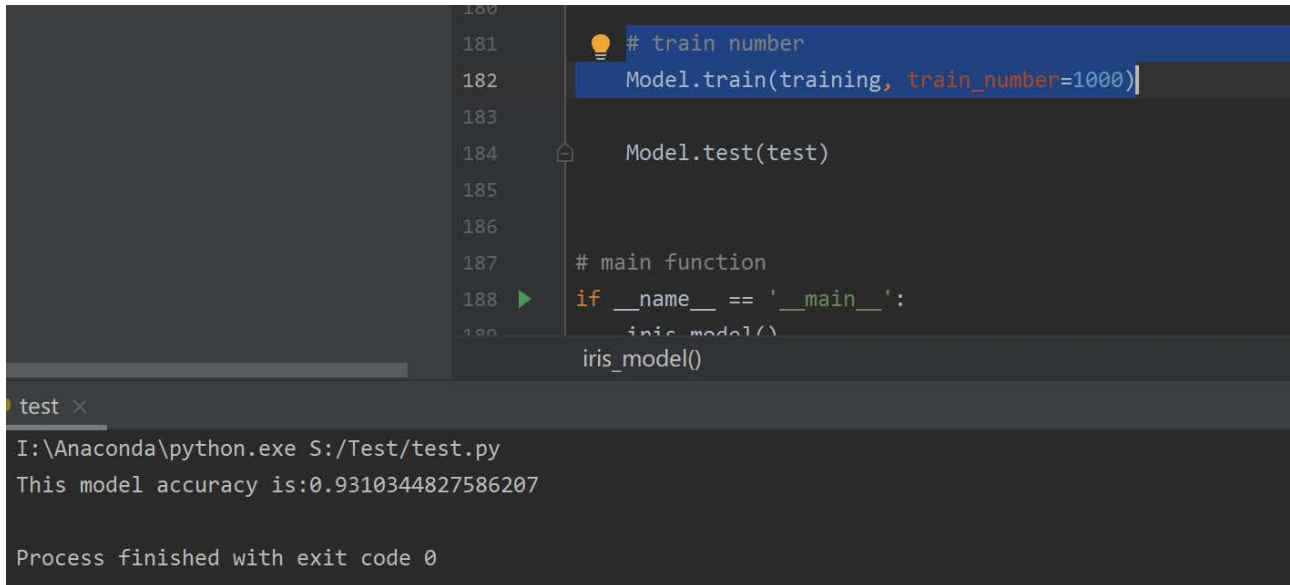
When the number of iteration time is 200, the model accuracy is 0.9310344, As shown in the 3.3.8 result.

When the number of iteration time is 1000, the model accuracy is still 0.9310344

Code:

```
# train number
Model.train(training, train_number=1000)
```

Result:



The screenshot shows a code editor with a dark theme. The code is as follows:

```
180
181 # train number
182 Model.train(training, train_number=1000)
183
184 Model.test(test)
185
186
187 # main function
188 if __name__ == '__main__':
189     iris_model()
190
```

Below the code editor is a terminal window titled 'test'. It shows the command prompt 'I:\Anaconda\python.exe S:/Test/test.py' and the output 'This model accuracy is:0.9310344827586207'. At the bottom, it says 'Process finished with exit code 0'.

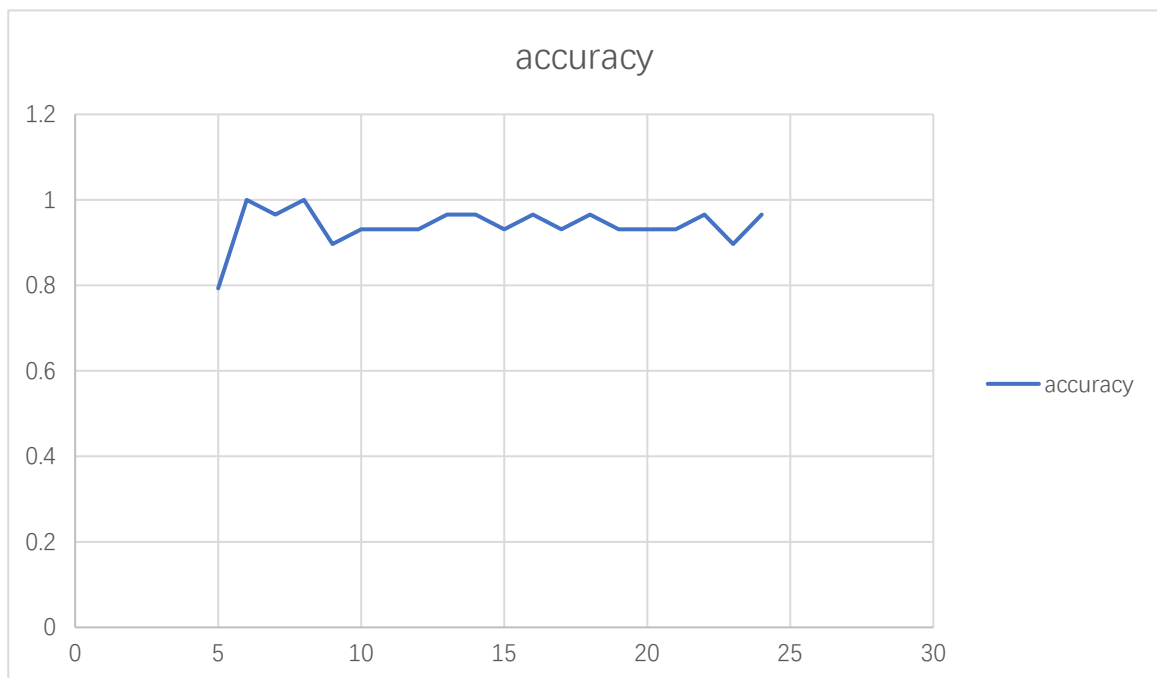
4.2 About the number of hidden layer node number

Set the model hidden layer node number define is 20, so the model accuracy is 0.9310344, As shown in the 3.3.8 result.

Change the node number from 5 to 24, the accuracy rate after statistics is shown in the table below:

| Hidden layer Node number | accuracy |
|-----------------------------|--------------------|
| 5 | 0.7931034482758621 |
| 6 | 1.0 |
| 7 | 0.9655172413793104 |
| 8 | 1.0 |
| 9 | 0.896551724137931 |
| 10 | 0.9310344827586207 |
| 11 | 0.9310344827586207 |
| 12 | 0.9310344827586207 |
| 13 | 0.9655172413793104 |
| 14 | 0.9655172413793104 |

| Hidden layer Node number | accuracy |
|-----------------------------|--------------------|
| 15 | 0.9310344827586207 |
| 16 | 0.9655172413793104 |
| 17 | 0.9310344827586207 |
| 18 | 0.9655172413793104 |
| 19 | 0.9310344827586207 |
| 20 | 0.9310344827586207 |
| 21 | 0.9310344827586207 |
| 22 | 0.9655172413793104 |
| 23 | 0.896551724137931 |
| 24 | 0.9655172413793104 |

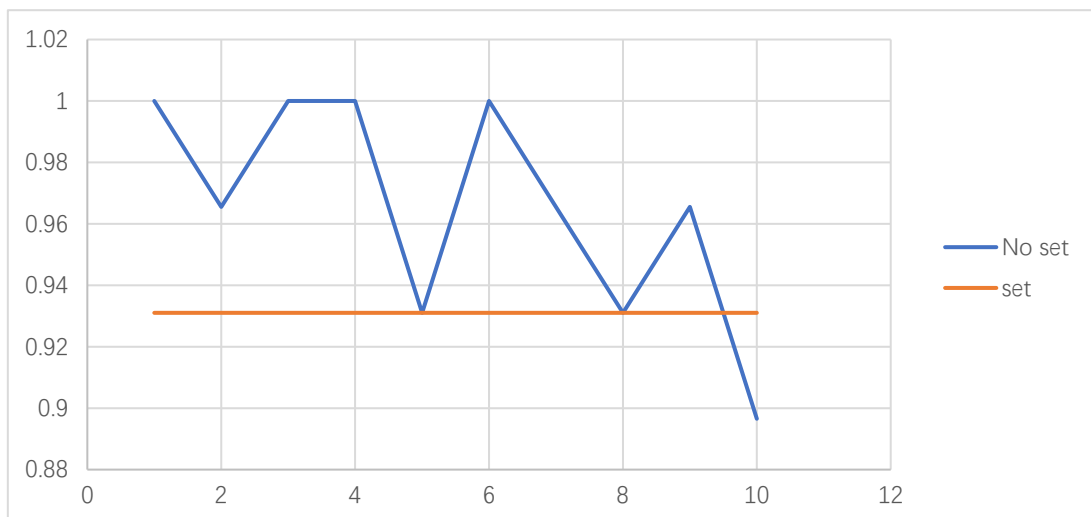


It can be seen that within the range of nodes in the experiment choosing 7、13、14、16、18、22、24 as the number of nodes in the hidden layer can obtain a better predictive value than 20 nodes

4.3 About the seed effect the model accuracy

At the same learning rate(LR = 0.001), hidden layer nodes(K = 20) and iteration times(T = 200), the experiment was repeated 10 times to check the effect of SEED on the results.

| The Nth experiment $N \in [1,10]$ | No seed Accuracy(X1) | Set seed Accuracy(X2) |
|--------------------------------------|-------------------------|--------------------------|
| 1 | 1.0 | 0.9310344827586207 |
| 2 | 0.9655172413793104 | 0.9310344827586207 |
| 3 | 1.0 | 0.9310344827586207 |
| 4 | 1.0 | 0.9310344827586207 |
| 5 | 0.9310344827586207 | 0.9310344827586207 |
| 6 | 1.0 | 0.9310344827586207 |
| 7 | 0.9655172413793104 | 0.9310344827586207 |
| 8 | 0.9310344827586207 | 0.9310344827586207 |
| 9 | 0.9655172413793104 | 0.9310344827586207 |
| 10 | 0.896551724137931 | 0.9310344827586207 |



So to take the variance of the results $D(X)$

$$D(X) = \sum_{i=1}^n (x_i - EX)^2 p_i$$

$$D(X1) = 0 \quad D(X2) = 0.03448$$

So, setting seed makes the result more accurate and stable.

5 Conclusion

This model can be well trained to form a three-layer neural network, which can be used to predict the results, and the prediction results are accurate.

After setting seed, the variance is 0 (4.3), and the variance without setting seed is only 0.03448 (4.3), which has high stability.