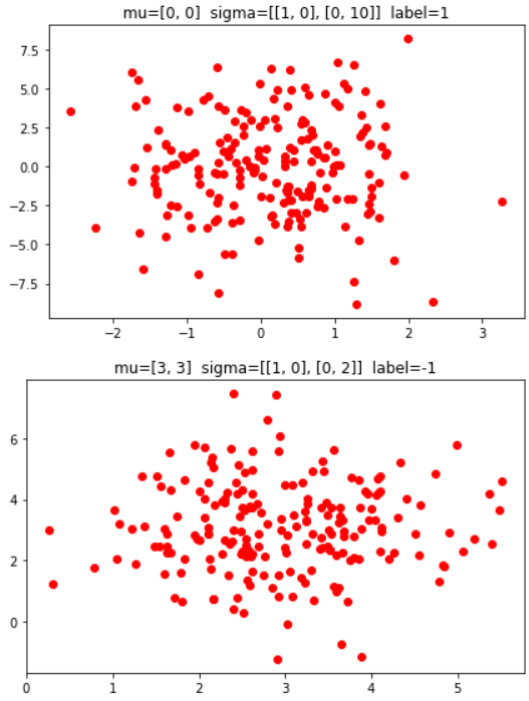
MLP Report

11911839 聂雨荷

1. Part I

1.1 Task 1

This is two Gaussian distributions and sample 200 points from each



1.2 Task2

I have implemented the perceptron.py.

1.3 Task3

Given the random seed = 42, the classification accuracy on the test set is 95%

[18]:
```
def to_input_and_sample(data_set):
    inputs = list()
    labels = list()
    for data in data_set:
        inputs.append([data[0], data[1]])
        labels.append(data[2])
    return inputs, labels
```

[19]:
```
train_inputs, train_labels = to_input_and_sample(train_set)
test_inputs, test_labels = to_input_and_sample(test_set)
```

[20]:
```
p = pe.Perceptron(2)
p.train(train_inputs, train_labels)
```

[21]:
```
def get_accuracy(p, test_inputs, test_labels):
    cnt = 0
    for i in range(len(test_inputs)):
        predict_label = p.forward(test_inputs[i])
        if predict_label == test_labels[i]:
            cnt += 1
    print('The accuracy of the test set is: {}%'.format((cnt/len(test_inputs) * 100)))
    return cnt/len(test_inputs)
```

[22]:
```
acc = get_accuracy(p, test_inputs, test_labels)
```

```
The accuracy of the test set is: 95.0%
```

1.4 Task4

If the means of the two Gaussians are too close, the accuracy become worse.

```
mu1 = [0, 0], mu2 = [10, 10]
The accuracy of the test set is: 100.0%
mu1 = [1, 1], mu2 = [9, 9]
The accuracy of the test set is: 100.0%
mu1 = [2, 2], mu2 = [8, 8]
The accuracy of the test set is: 100.0%
mu1 = [3, 3], mu2 = [7, 7]
The accuracy of the test set is: 100.0%
mu1 = [4, 4], mu2 = [6, 6]
The accuracy of the test set is: 85.0%
mu1 = [5, 5], mu2 = [5, 5]
The accuracy of the test set is: 57.49999999999999%
mu1 = [6, 6], mu2 = [4, 4]
The accuracy of the test set is: 85.0%
mu1 = [7, 7], mu2 = [3, 3]
The accuracy of the test set is: 98.75%
mu1 = [8, 8], mu2 = [2, 2]
The accuracy of the test set is: 100.0%
mu1 = [9, 9], mu2 = [1, 1]
The accuracy of the test set is: 100.0%
```

If the variances of the two Gaussians are too high, the accuracy become better

```
sigma1 = [[0, 0], [0, 0]], sigma2 = [[0, 0], [0, 0]]
The accuracy of the test set is: 100.0%
sigma1 = [[1, 0], [0, 1]], sigma2 = [[1, 0], [0, 1]]
The accuracy of the test set is: 91.25%
sigma1 = [[2, 0], [0, 2]], sigma2 = [[2, 0], [0, 2]]
The accuracy of the test set is: 86.25%
sigma1 = [[3, 0], [0, 3]], sigma2 = [[3, 0], [0, 3]]
The accuracy of the test set is: 83.75%
sigma1 = [[4, 0], [0, 4]], sigma2 = [[4, 0], [0, 4]]
The accuracy of the test set is: 76.25%
sigma1 = [[5, 0], [0, 5]], sigma2 = [[5, 0], [0, 5]]
The accuracy of the test set is: 72.5%
sigma1 = [[6, 0], [0, 6]], sigma2 = [[6, 0], [0, 6]]
The accuracy of the test set is: 63.74999999999999%
sigma1 = [[7, 0], [0, 7]], sigma2 = [[7, 0], [0, 7]]
The accuracy of the test set is: 67.5%
sigma1 = [[8, 0], [0, 8]], sigma2 = [[8, 0], [0, 8]]
The accuracy of the test set is: 72.5%
sigma1 = [[9, 0], [0, 9]], sigma2 = [[9, 0], [0, 9]]
The accuracy of the test set is: 56.25%
```

2. Part II

2.1 Task1

I have implemented the mlp_numpy.py and the modules.py.

2.2 Task2

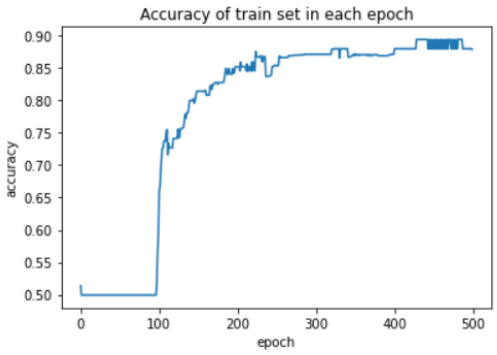
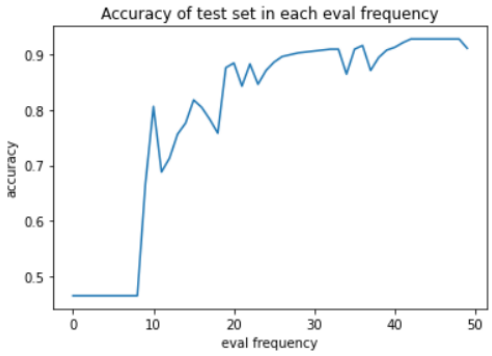
I have implemented the train_mlp_numpy.py.

2.3 Task3

The default parameters are listed below:

```
# Default constants
DNN_HIDDEN_UNITS_DEFAULT = '20,12,6,5'  # 隐藏层
LEARNING_RATE_DEFAULT = 1e-2  # 学习率
MAX_EPOCHS_DEFAULT = 500  # epoch
EVAL_FREQ_DEFAULT = 10  # 每多少个epoch进行一次输出
BATCH = 10  # 梯度下降批处理
```

The accuracy of training data and test data are shown as the figures below:

training data	test data
	

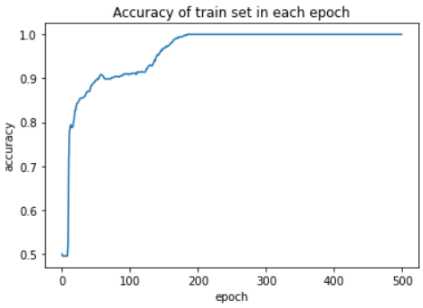
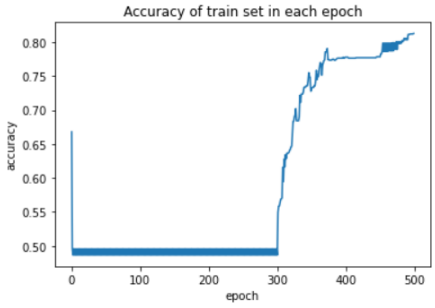
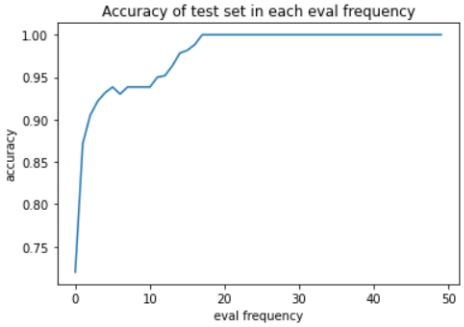
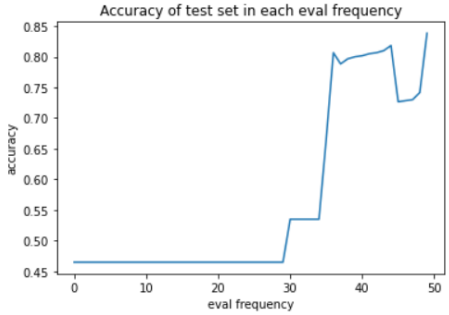
3. Part III

3.1 Task1

I have modified the train method in train_mlp_numpy.py.

I used the mini-batch method to compute gradient descent. You can specified certain batch number and get the result. (The execute method is shown in the last section of the report)

3.2 Task2

	stochastic gradient descent	batch gradient descent(batch = 30)
train data		
test data		

4. How to execute file

4.1 Jupyter notebook

Import files

In order to successfully execute all the file, you need to import several packages

```
import numpy as np
import random
from scipy import stats
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from pandas.core.frame import DataFrame
from sklearn import datasets
```

And two self-defined files.

```
import perceptron as pe
import train_mlp_numpy
```

Execute Code

Please follow the code written in the Assignment1.ipynb to see the execution result.

MLP problem

For Task2-3, if you use jupyter notebook, after importing previous package. You can use the following code to define your own MLP network and see the execution result.

An example is shown below:

```
train_mlp_numpy.main(n_hidden='20,12,6,5', lr=1e-2, epoch=500, eval_freq=10,
batch=30)
```

- `n_hidden` : the architecture of hidden layers
- `lr` : learning rate
- `epoch` : training epoch
- `eval_freq` : eval frequency in calculate the accuracy in the test data
- `batch` : batch number used in mini-batch gradient descent

If you don't specified certain variable, it will directly use the default parameters.

4.2 Command Line

MLP problem

You can use the following command. Make sure you are in the same directory hierarchy with train_mlp_numpy.py.

```
python train_mlp_numpy.py --dnn_hidden_units 20,12,6,5 --learning_rate 0.01 --
max_steps 500 --eval_freq 10 -- batch 30
```