choice of test-train split ratio

```
def div_train_test(attribute, label, sample_num):

    num_test = sample_num * 0.4

    attribute_test = []
    label_test = []

    attribute_train = []
    label_train = []

    index_total = np.array(range(sample_num))
    np.random.shuffle(index_total)

    for i in range(sample_num):
        if (i < num_test):
            attribute_test.append(attribute[index_total[i]])
            label_test.append(label[index_total[i]])
            continue

        attribute_train.append(attribute[index_total[i]])
        label_train.append(label[index_total[i]])

    attribute_train = np.array(attribute_train)
    label_train = np.array(label_train)
    attribute_test = np.array(attribute_test)
    label_test = np.array(label_test)
    return attribute_train, label_train, attribute_test, label_test
```

Figure 1

Based on this function in Figure 1 to divide the original dataset into train dataset and validation dataset (6 : 4), to avoid the overfitting. And the generated train and validation dataset are randomly choose use the np.random.shuffle function at each epoch. And the reason I choose this ratio is that if there are less number of data in validation dataset, then if one of the data is classified wrongly, then it will have a large Impact on the validation performance. And I use the correct rate of the trained MLP performance on the validation dataset to do an early stopping, to avoid the overfitting, shown in Figure 2.

```
def train_classfier(attribute, label):

    sample_num = 0
    for i in label:
        sample_num += 1
    classfier = NN()

    correct_rate = 0
    for j in range(100):
        # get the train dataset and test dataset
        attribute_train, label_train, attribute_test, label_test = div_train_test(attribute, label, sample_num)
        #print("j: ",j)
        correct = 0
        if(correct_rate < 0.96):
            for i in range(len(attribute_train)):
                if ((i % 1000) == 0):
                    print(i)
                    #print("b1: ", classfier.W1)
                #print("input: ",attribute_train[i])
                classfier.train(attribute_train[i], label_train[i])
            for i in range(len(attribute_test)):
                #print(i)
                #print("attribute_test[i].shape", attribute_test[i].shape)
                predict = classfier.feedfoward(attribute_test[i])
                predict = np.rint(predict)

                if check_output_same(predict, label_test[i]):
                    correct += 1

            correct_rate = correct / len(label_test)
            print(correct_rate)
    return classfier
```

Figure 2

number of nodes and activation functions

Because I use the correct rate of the performance on the divided validation dataset to do an early stopping, and at each epoch I randomly choose a different validation dataset, as long as the number of hidden layer nodes can meet my stopping criterion, then the number will work. In this design, I choose 392 nodes in my hidden layer, which is half of the size of the input nodes.

And the activation function I choose is sigmoid function, and the error I use in the bp is the MSE, just like the example in the Set2 Slides in ECE657. And I do the online training in this question, and the sigmoid function works fine.

training and testing accuracies

```
from test_mlp import test_mlp, STUDENT_NAME, STUDENT_ID
from acc_calc import accuracy
from q4 import NN
import pandas as pd
if __name__ == "__main__":
        y_pred = test_mlp('./test_data.csv')

        test_labels = pd.read_csv('./test_label.csv').to_numpy()

        test_accuracy = accuracy(test_labels, y_pred) * 100
        print("test_accuarcy: ", test_accuracy)
```

```
test_accuarcy:  96.29943845190482
```

Figure 3

Based on the given test_mlp and acc_calc, I do a pre-test before submitting the assignment.
From above, we have known that I use the correct_rate to do an early stopping, and the correct_rate I choose is 0.96. As shown in Figure3, I import my saved trained NN object to do a test on the whole original dataset to see the performance, and the test accuracy works fine, which is around 96%.

Comments

I will submit my source code in the dropbox, you can see the comments on my chain rule, which is the main part of the BP algorithm, and how I do training and divide the dataset. And the saved model is trained based on the correct rate 0.96 stopping criterion. And if there are something wrong when run my model, can send me the email t5meng@uwaterloo.ca. if you run my q4.py source code, you need to change the local path to your path in the testing computer; however, I have saved the trained model in trained_model.pkl using pickle.