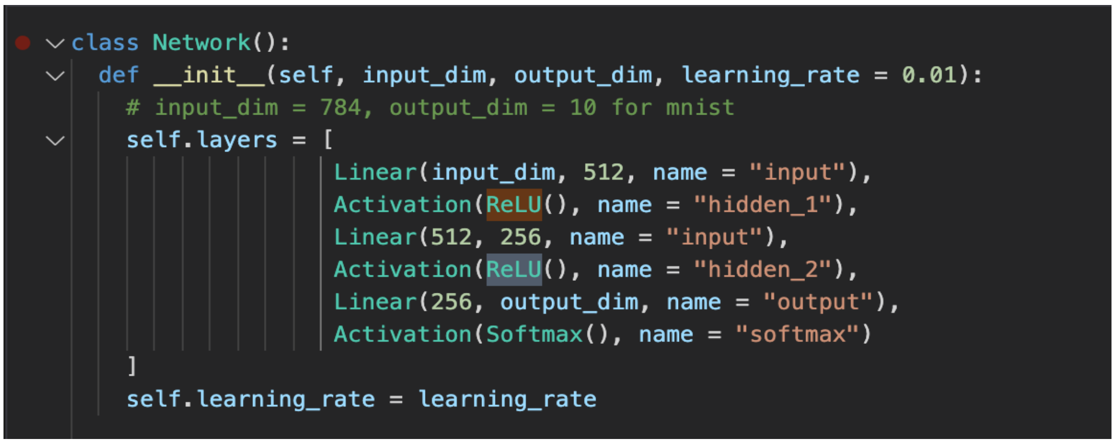
HW1_report

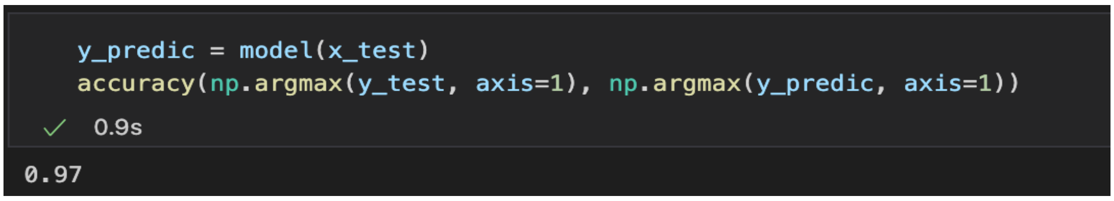
1.

I design a neural network which has one input layer, two hidden layer and one output layer.



```
class Network():
    def __init__(self, input_dim, output_dim, learning_rate = 0.01):
        # input_dim = 784, output_dim = 10 for mnist
        self.layers = [
                        Linear(input_dim, 512, name = "input"),
                        Activation(ReLU(), name = "hidden_1"),
                        Linear(512, 256, name = "input"),
                        Activation(ReLU(), name = "hidden_2"),
                        Linear(256, output_dim, name = "output"),
                        Activation(Softmax(), name = "softmax")
        ]
        self.learning_rate = learning_rate
```

The testing accuracy is about 97%

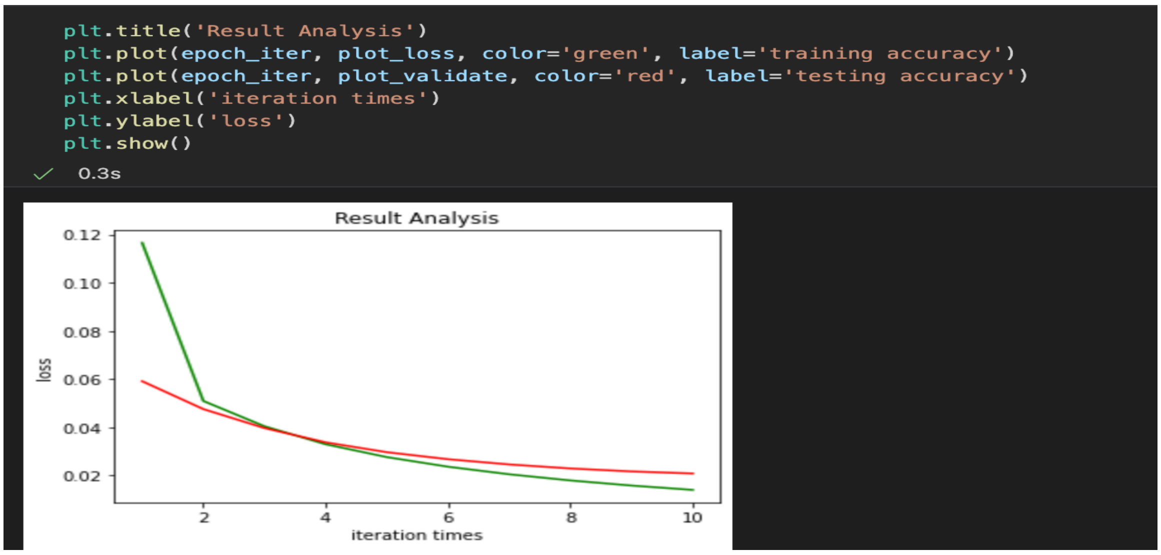


```
y_predic = model(x_test)
accuracy(np.argmax(y_test, axis=1), np.argmax(y_predic, axis=1))
✓ 0.9s
0.97
```

2.

Feedforward to get the input of next layer, using chain rule we could backward the output to the layer and tried to minimize the loss so that we could get the desirable weight and bias between layers.

3.



```
plt.title('Result Analysis')
plt.plot(epoch_iter, plot_loss, color='green', label='training accuracy')
plt.plot(epoch_iter, plot_validate, color='red', label='testing accuracy')
plt.xlabel('iteration times')
plt.ylabel('loss')
plt.show()
✓ 0.3s
```

4.

It is not always true. It really depends on the quality of data, and it may cause overfitting problem. Accuracy will increase with more hidden layers, but performance will decrease.

At this MNIST neural network, I first design merely one layer and it performance is not bad, but I redesign a two hidden layer neural network. This time I got much performance, you can tell from the loss graph. With more neuron, the converge speed is faster than before.

5.

To prevent out model from overfitting.