HW7_P1

November 24, 2018

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
```

In [2]:
```
dfTrainData = pd.read_csv('C:/Users/weich/Google Drive/Rice University/3rd Semester/ELE
dfTrainLabel = pd.read_csv('C:/Users/weich/Google Drive/Rice University/3rd Semester/EL
dfTestData = pd.read_csv('C:/Users/weich/Google Drive/Rice University/3rd Semester/ELEC
dfTestLabel = pd.read_csv('C:/Users/weich/Google Drive/Rice University/3rd Semester/ELE
```

In [3]:
```
def GetData():
    TrainData_temp = dfTrainData.values[:,1:]
    TrainLabel_temp = dfTrainLabel.values[:,1:]
    shuffle_list = list(range(60000))
    np.random.shuffle(shuffle_list)

    TrainData = (TrainData_temp[shuffle_list[:50000],:]/255).astype('float64')
    TrainLabel = TrainLabel_temp[shuffle_list[:50000],:]
    ValidData = (TrainData_temp[shuffle_list[50000:],:]/255).astype('float64')
    ValidLabel = TrainLabel_temp[shuffle_list[50000:],:]
    TestData = (dfTestData.values[:,1:]/255).astype('float64')
    TestLabel = dfTestLabel.values[:,1:]
    return TrainData, TrainLabel, ValidData, ValidLabel, TestData, TestLabel
```

In [4]:
```
# Problem1.1
class NeuralNetwork(object):

    def __init__(self, nn_input_dim, nn_hidden_dim , nn_output_dim):
        self.nn_input_dim = nn_input_dim
        self.nn_hidden_dim = nn_hidden_dim
        self.nn_output_dim = nn_output_dim

        self.W1 = np.random.randn(self.nn_input_dim, self.nn_hidden_dim) / np.sqrt(sel
        self.b1 = np.zeros((1, self.nn_hidden_dim))
        self.W2 = np.random.randn(self.nn_hidden_dim, self.nn_output_dim) / np.sqrt(se
        self.b2 = np.zeros((1, self.nn_output_dim))
```

```
def actFun(self, z):

    return 1.0 / (1+np.exp(-z))

def diff_actFun(self, z):

    fz = 1.0 / (1+np.exp(-z))
    return fz * (1-fz)

def feedforward(self, X):

    self.z1 = np.dot(X, self.W1) + self.b1
    self.a1 = self.actFun(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
    exp_scores = np.exp(self.z2)
    self.probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    return self.probs

def calculate_loss(self, X, y):

    num_examples = len(X)
    self.feedforward(X)

    y_onehot = OneHotEncoder(categories=[range(10)], sparse=False).fit_transform(y
    data_loss = - np.sum(np.log(self.probs) * y_onehot) / num_examples

    return data_loss

def predict(self, X):

    self.feedforward(X)
    return np.argmax(self.probs, axis=1)

def backprop(self, X, y):

    num_examples = len(X)

    # dL/da2
    dLda2 = - OneHotEncoder(categories=[range(10)], sparse=False).fit_transform(y.
    # da2/dz2
    da2dz2 = np.zeros(self.probs.shape + (self.probs.shape[-1],))
    for i in range(num_examples):
        da2dz2[i, :, :] = np.diag(self.probs[i, :]) - np.outer(self.probs[i, :], se
    # dz2/da1
    dz2da1 = self.W2[None, :, :]
    # da1/dz1
    da1_dz1 = np.zeros(self.z1.shape + (self.z1.shape[-1],))
```

```python
        for i in range(num_examples):
            da1_dz1[i, :, :] = np.diag(self.diff_actFun(self.z1[i, :]))
        # dL/dz2
        dLdz2 = np.einsum('ijk,ik->ij', da2dz2, dLda2)
        # dL/dz1
        dLdz1 = np.einsum('ijk,ik->ij', da1_dz1 @ dz2da1, dLdz2)

        # dW, db
        dW2 = dLdz2.T @ self.a1
        db2 = dLdz2.sum(0)[:, None]
        dW1 = dLdz1.T @ X
        db1 = dLdz1.sum(0)[:, None]

        return dW1.T, dW2.T, db1.T, db2.T

    def fit_model(self, X, y, epsilon, print_loss=True):

        # Forward propagation
        self.feedforward(X)

        # Backpropagation
        dW1, dW2, db1, db2 = self.backprop(X, y)

        # Gradient descent parameter update
        self.W1 += -epsilon * dW1
        self.b1 += -epsilon * db1
        self.W2 += -epsilon * dW2
        self.b2 += -epsilon * db2

def main():
    model = NeuralNetwork(nn_input_dim=784, nn_hidden_dim=300 , nn_output_dim=10)
    val_loss = np.zeros(30)
    train_loss = np.zeros(30)
    val_acc = np.zeros(30)
    train_acc = np.zeros(30)
    test_acc = np.zeros(30)
    for epoch in range(30):
        # shuffle
        [TrainData, TrainLabel, ValidData, ValidLabel, TestData, TestLabel] = GetData(

        batch_size = 1000

        # Training
        # Training loss
        loss = 0
        for i in range(50):

            X = TrainData[batch_size*i:batch_size*(i+1),:]
```

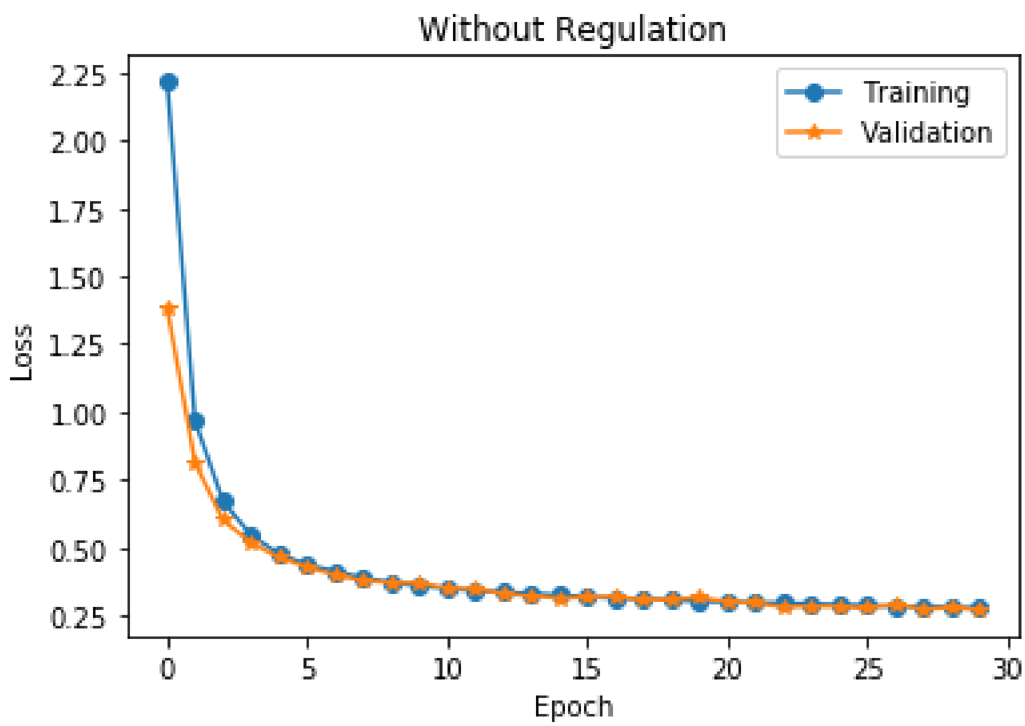
```
            y = TrainLabel[batch_size*i:batch_size*(i+1)]
            model.fit_model(X, y, (1/(i+1))**0.5)

            loss += model.calculate_loss(X, y)

        train_loss[epoch] = loss

        # Training accuracy
        prediction = model.predict(X)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == y[j]):
                count += 1
        train_acc[epoch] = count/len(prediction)
        print("Loss/Accuracy of Training at epoch %i: %f/%f" % (epoch+1, train_loss[epo

        # Validation
        # Validation loss
        loss = 0
        for i in range(10):

            X = ValidData[batch_size*i:batch_size*(i+1),:]
            y = ValidLabel[batch_size*i:batch_size*(i+1)]

            loss += model.calculate_loss(X, y)

        val_loss[epoch] = loss

        # Validation accuracy
        prediction = model.predict(ValidData)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == ValidLabel[j]):
                count += 1
        val_acc[epoch] = count/len(prediction)
        print("Loss/Accuracy of Validation at epoch %i: %f/%f" % (epoch+1, val_loss[epo

        # Testing accuracy
        prediction = model.predict(TestData)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == TestLabel[j]):
                count += 1
        test_acc[epoch] = count/len(prediction)
        print("Accuracy of Testing at epoch %i: %f" % (epoch+1, test_acc[epoch]))

    # plot
    plt.figure(1)
```

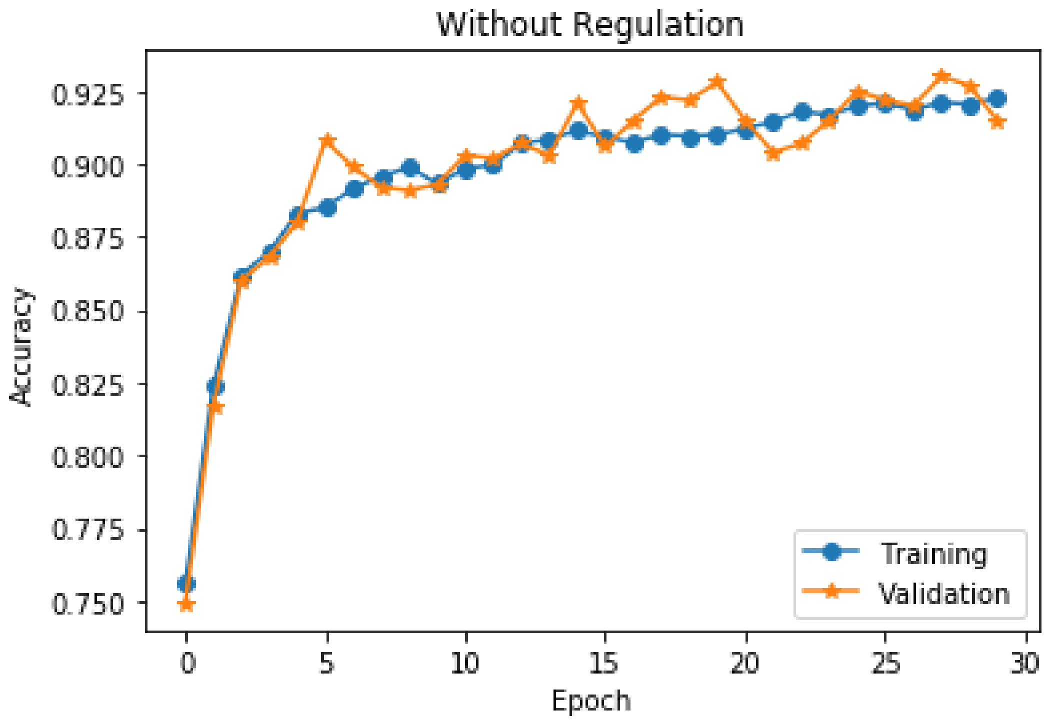
```
        plt.plot(range(30), np.array(train_loss)/50, '-o')
        plt.plot(range(30), np.array(val_loss)/10, '-*')
        plt.legend(('Training', 'Validation'))
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.title('Without Regulation')
        plt.show()

        plt.figure(2)
        plt.plot(range(30), np.array(val_acc), '-o')
        plt.plot(range(30), np.array(train_acc), '-*')
        plt.legend(('Training', 'Validation'))
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.title('Without Regulation')
        plt.show()

        plt.figure(3)
        plt.plot(range(30), np.array(test_acc), '-o')
        plt.legend(('Testing'))
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.title('Without Regulation')
        plt.show()

    if __name__ == "__main__":
        main()
```

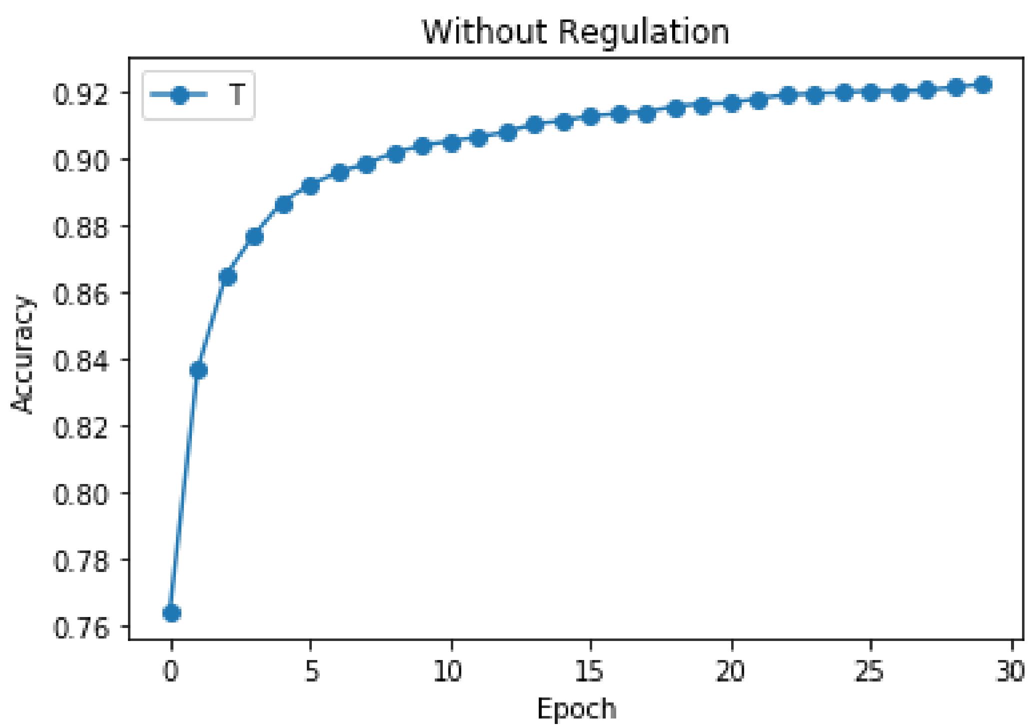
```
Loss/Accuracy of Training at epoch 1: 2.216976/0.749000
Loss/Accuracy of Validation at epoch 1: 1.384690/0.755800
Accuracy of Testing at epoch 1: 0.764100
Loss/Accuracy of Training at epoch 2: 0.965627/0.817000
Loss/Accuracy of Validation at epoch 2: 0.811742/0.823700
Accuracy of Testing at epoch 2: 0.837000
Loss/Accuracy of Training at epoch 3: 0.670556/0.860000
Loss/Accuracy of Validation at epoch 3: 0.600403/0.861900
Accuracy of Testing at epoch 3: 0.865100
Loss/Accuracy of Training at epoch 4: 0.545456/0.868000
Loss/Accuracy of Validation at epoch 4: 0.518921/0.869800
Accuracy of Testing at epoch 4: 0.877200
Loss/Accuracy of Training at epoch 5: 0.476991/0.880000
Loss/Accuracy of Validation at epoch 5: 0.463157/0.883000
Accuracy of Testing at epoch 5: 0.887000
Loss/Accuracy of Training at epoch 6: 0.436331/0.908000
Loss/Accuracy of Validation at epoch 6: 0.429574/0.885000
Accuracy of Testing at epoch 6: 0.892400
Loss/Accuracy of Training at epoch 7: 0.412683/0.899000
Loss/Accuracy of Validation at epoch 7: 0.398517/0.891500
```

```
Accuracy of Testing at epoch 7: 0.896000
Loss/Accuracy of Training at epoch 8: 0.388378/0.892000
Loss/Accuracy of Validation at epoch 8: 0.378767/0.895700
Accuracy of Testing at epoch 8: 0.898700
Loss/Accuracy of Training at epoch 9: 0.372182/0.891000
Loss/Accuracy of Validation at epoch 9: 0.370825/0.898900
Accuracy of Testing at epoch 9: 0.902000
Loss/Accuracy of Training at epoch 10: 0.357780/0.893000
Loss/Accuracy of Validation at epoch 10: 0.369921/0.893200
Accuracy of Testing at epoch 10: 0.904100
Loss/Accuracy of Training at epoch 11: 0.350020/0.903000
Loss/Accuracy of Validation at epoch 11: 0.349854/0.898400
Accuracy of Testing at epoch 11: 0.905500
Loss/Accuracy of Training at epoch 12: 0.340454/0.902000
Loss/Accuracy of Validation at epoch 12: 0.348849/0.899600
Accuracy of Testing at epoch 12: 0.906700
Loss/Accuracy of Training at epoch 13: 0.336195/0.907000
Loss/Accuracy of Validation at epoch 13: 0.330048/0.907000
Accuracy of Testing at epoch 13: 0.908200
Loss/Accuracy of Training at epoch 14: 0.330726/0.903000
Loss/Accuracy of Validation at epoch 14: 0.321457/0.908500
Accuracy of Testing at epoch 14: 0.910600
Loss/Accuracy of Training at epoch 15: 0.325289/0.921000
Loss/Accuracy of Validation at epoch 15: 0.314740/0.911500
Accuracy of Testing at epoch 15: 0.911600
Loss/Accuracy of Training at epoch 16: 0.319180/0.906000
Loss/Accuracy of Validation at epoch 16: 0.316628/0.909100
Accuracy of Testing at epoch 16: 0.913000
Loss/Accuracy of Training at epoch 17: 0.313077/0.915000
Loss/Accuracy of Validation at epoch 17: 0.320732/0.907400
Accuracy of Testing at epoch 17: 0.913800
Loss/Accuracy of Training at epoch 18: 0.311248/0.923000
Loss/Accuracy of Validation at epoch 18: 0.305755/0.910100
Accuracy of Testing at epoch 18: 0.914200
Loss/Accuracy of Training at epoch 19: 0.306206/0.922000
Loss/Accuracy of Validation at epoch 19: 0.309412/0.909600
Accuracy of Testing at epoch 19: 0.915900
Loss/Accuracy of Training at epoch 20: 0.301068/0.928000
Loss/Accuracy of Validation at epoch 20: 0.315631/0.910000
Accuracy of Testing at epoch 20: 0.916700
Loss/Accuracy of Training at epoch 21: 0.299803/0.915000
Loss/Accuracy of Validation at epoch 21: 0.300351/0.912200
Accuracy of Testing at epoch 21: 0.916900
Loss/Accuracy of Training at epoch 22: 0.297202/0.904000
Loss/Accuracy of Validation at epoch 22: 0.295286/0.914500
Accuracy of Testing at epoch 22: 0.918300
Loss/Accuracy of Training at epoch 23: 0.296648/0.907000
Loss/Accuracy of Validation at epoch 23: 0.281208/0.918100
```

```
Accuracy of Testing at epoch 23: 0.919400
Loss/Accuracy of Training at epoch 24: 0.292792/0.915000
Loss/Accuracy of Validation at epoch 24: 0.283910/0.916800
Accuracy of Testing at epoch 24: 0.919800
Loss/Accuracy of Training at epoch 25: 0.289855/0.925000
Loss/Accuracy of Validation at epoch 25: 0.283243/0.919900
Accuracy of Testing at epoch 25: 0.920100
Loss/Accuracy of Training at epoch 26: 0.288665/0.922000
Loss/Accuracy of Validation at epoch 26: 0.275614/0.920900
Accuracy of Testing at epoch 26: 0.920500
Loss/Accuracy of Training at epoch 27: 0.282487/0.920000
Loss/Accuracy of Validation at epoch 27: 0.291806/0.918600
Accuracy of Testing at epoch 27: 0.920400
Loss/Accuracy of Training at epoch 28: 0.283389/0.930000
Loss/Accuracy of Validation at epoch 28: 0.273273/0.921200
Accuracy of Testing at epoch 28: 0.920800
Loss/Accuracy of Training at epoch 29: 0.279649/0.927000
Loss/Accuracy of Validation at epoch 29: 0.279025/0.920500
Accuracy of Testing at epoch 29: 0.921600
Loss/Accuracy of Training at epoch 30: 0.278670/0.915000
Loss/Accuracy of Validation at epoch 30: 0.271470/0.922800
Accuracy of Testing at epoch 30: 0.922600
```




Without Regulation
Accuracy
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
5
10
15
20
25
30
Epoch
Training
Validation


Without Regulation
Accuracy
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0
5
10
15
20
25
30
Epoch
T

In [ ]:

In [ ]:

In [5]:
```
# Problem1.2
class NeuralNetwork(object):

    def __init__(self, nn_input_dim, nn_hidden_dim , nn_output_dim, reg_lambda=0.0001)
        self.nn_input_dim = nn_input_dim
        self.nn_hidden_dim = nn_hidden_dim
        self.nn_output_dim = nn_output_dim
        self.reg_lambda = reg_lambda

        self.W1 = np.random.randn(self.nn_input_dim, self.nn_hidden_dim) / np.sqrt(sel
        self.b1 = np.zeros((1, self.nn_hidden_dim))
        self.W2 = np.random.randn(self.nn_hidden_dim, self.nn_output_dim) / np.sqrt(se
        self.b2 = np.zeros((1, self.nn_output_dim))

    def actFun(self, z):

        return 1.0 / (1+np.exp(-z))

    def diff_actFun(self, z):

        fz = 1.0 / (1+np.exp(-z))
        return fz * (1-fz)

    def feedforward(self, X):

        self.z1 = np.dot(X, self.W1) + self.b1
        self.a1 = self.actFun(self.z1)
        self.z2 = np.dot(self.a1, self.W2) + self.b2
        exp_scores = np.exp(self.z2)
        self.probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

        return self.probs

    def calculate_loss(self, X, y):

        num_examples = len(X)
        self.feedforward(X)

        y_onehot = OneHotEncoder(categories=[range(10)], sparse=False).fit_transform(y
        data_loss = - np.sum(np.log(self.probs) * y_onehot) / num_examples

        # Add regulatization term to loss (optional)
        data_loss += self.reg_lambda * (np.sum(np.square(self.W1)) + np.sum(np.square(
```

```
        return data_loss

    def predict(self, X):

        self.feedforward(X)
        return np.argmax(self.probs, axis=1)

    def backprop(self, X, y):

        num_examples = len(X)

        # dL/da2
        dLda2 = - OneHotEncoder(categories=[range(10)], sparse=False).fit_transform(y.r
        # da2/dz2
        da2dz2 = np.zeros(self.probs.shape + (self.probs.shape[-1],))
        for i in range(num_examples):
            da2dz2[i, :, :] = np.diag(self.probs[i, :]) - np.outer(self.probs[i, :], se
        # dz2/da1
        dz2da1 = self.W2[None, :, :]
        # da1/dz1
        da1_dz1 = np.zeros(self.z1.shape + (self.z1.shape[-1],))
        for i in range(num_examples):
            da1_dz1[i, :, :] = np.diag(self.diff_actFun(self.z1[i, :]))
        # dL/dz2
        dLdz2 = np.einsum('ijk,ik->ij', da2dz2, dLda2)
        # dL/dz1
        dLdz1 = np.einsum('ijk,ik->ij', da1_dz1 @ dz2da1, dLdz2)

        # dW, db
        dW2 = dLdz2.T @ self.a1
        db2 = dLdz2.sum(0)[:, None]
        dW1 = dLdz1.T @ X
        db1 = dLdz1.sum(0)[:, None]

        return dW1.T, dW2.T, db1.T, db2.T

    def fit_model(self, X, y, epsilon, print_loss=True):

        # Forward propagation
        self.feedforward(X)

        # Backpropagation
        dW1, dW2, db1, db2 = self.backprop(X, y)

        # Add regularization terms (b1 and b2 don't have regularization terms)
        dW2 += self.reg_lambda * self.W2
        dW1 += self.reg_lambda * self.W1
```

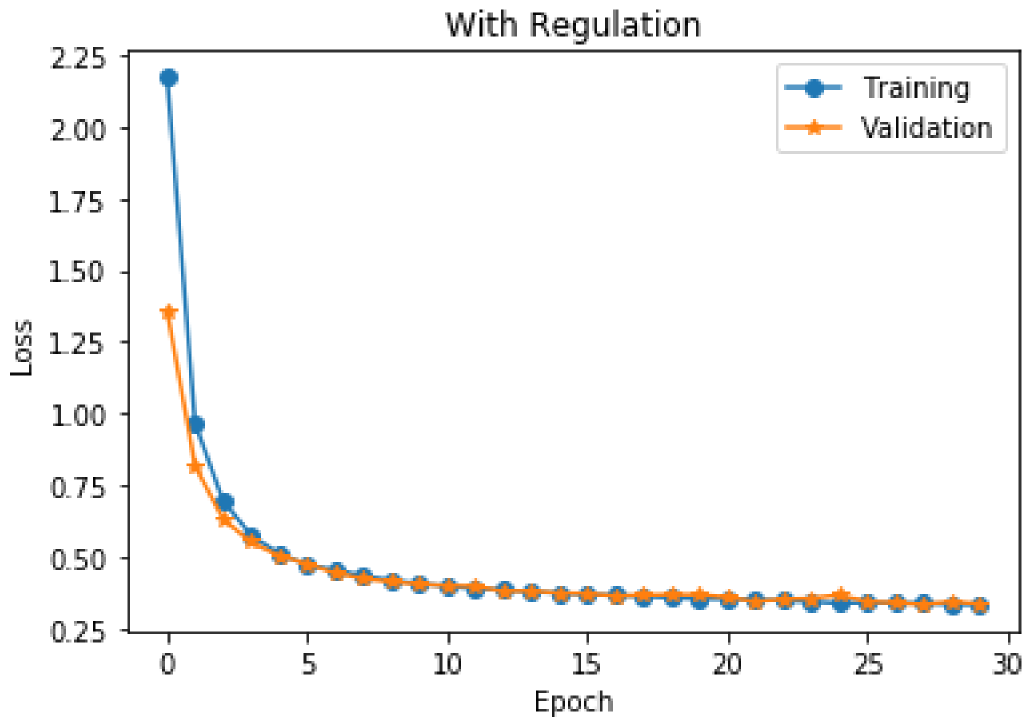
```
        # Gradient descent parameter update
        self.W1 += -epsilon * dW1
        self.b1 += -epsilon * db1
        self.W2 += -epsilon * dW2
        self.b2 += -epsilon * db2

def main():
    model = NeuralNetwork(nn_input_dim=784, nn_hidden_dim=300 , nn_output_dim=10)
    val_loss = np.zeros(30)
    train_loss = np.zeros(30)
    val_acc = np.zeros(30)
    train_acc = np.zeros(30)
    test_acc = np.zeros(30)
    for epoch in range(30):
        # shuffle
        [TrainData, TrainLabel, ValidData, ValidLabel, TestData, TestLabel] = GetData(

        batch_size = 1000

        # Training
        # Training loss
        loss = 0
        for i in range(50):

            X = TrainData[batch_size*i:batch_size*(i+1),:]
            y = TrainLabel[batch_size*i:batch_size*(i+1)]
            model.fit_model(X, y, (1/(i+1))**0.5)

            loss += model.calculate_loss(X, y)

        train_loss[epoch] = loss

        # Training accuracy
        prediction = model.predict(X)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == y[j]):
                count += 1
        train_acc[epoch] = count/len(prediction)
        print("Loss/Accuracy of Training at epoch %i: %f/%f" % (epoch+1, train_loss[epo

        # Validation
        # Validation loss
        loss = 0
        for i in range(10):

            X = ValidData[batch_size*i:batch_size*(i+1),:]
            y = ValidLabel[batch_size*i:batch_size*(i+1)]
```

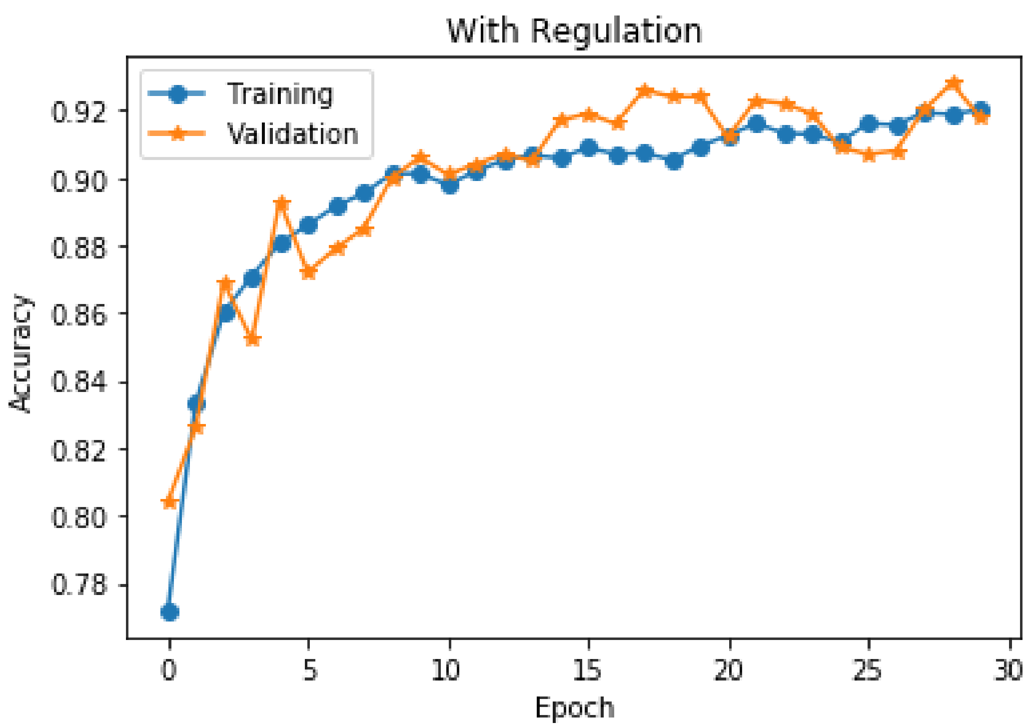
```
            loss += model.calculate_loss(X, y)

        val_loss[epoch] = loss

        # Validation accuracy
        prediction = model.predict(ValidData)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == ValidLabel[j]):
                count += 1
        val_acc[epoch] = count/len(prediction)
        print("Loss/Accuracy of Validation at epoch %i: %f/%f" % (epoch+1, val_loss[epo


        # Testing accuracy
        prediction = model.predict(TestData)
        count = 0
        for j in range(len(prediction)):
            if(prediction[j] == TestLabel[j]):
                count += 1
        test_acc[epoch] = count/len(prediction)
        print("Accuracy of Testing at epoch %i: %f" % (epoch+1, test_acc[epoch]))

    # plot
    plt.figure(1)
    plt.plot(range(30), np.array(train_loss)/50, '-o')
    plt.plot(range(30), np.array(val_loss)/10, '-*')
    plt.legend(('Training', 'Validation'))
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.title('With Regulation')
    plt.show()

    plt.figure(2)
    plt.plot(range(30), np.array(val_acc), '-o')
    plt.plot(range(30), np.array(train_acc), '-*')
    plt.legend(('Training', 'Validation'))
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.title('With Regulation')
    plt.show()

    plt.figure(3)
    plt.plot(range(30), np.array(test_acc), '-o')
    plt.legend(('Testing'))
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
```

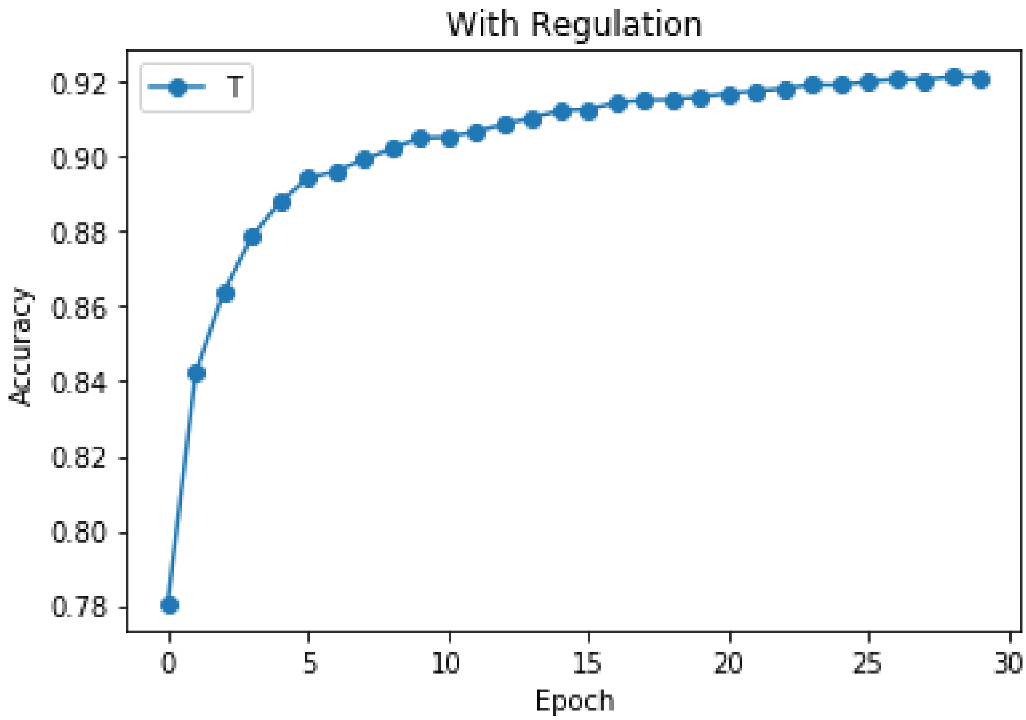
```python
        plt.title('With Regulation')
        plt.show()

if __name__ == "__main__":
    main()
```

```
Loss/Accuracy of Training at epoch 1: 2.173809/0.804000
Loss/Accuracy of Validation at epoch 1: 1.359370/0.771800
Accuracy of Testing at epoch 1: 0.780500
Loss/Accuracy of Training at epoch 2: 0.965206/0.826000
Loss/Accuracy of Validation at epoch 2: 0.816663/0.833200
Accuracy of Testing at epoch 2: 0.842100
Loss/Accuracy of Training at epoch 3: 0.695431/0.869000
Loss/Accuracy of Validation at epoch 3: 0.631913/0.860200
Accuracy of Testing at epoch 3: 0.863900
Loss/Accuracy of Training at epoch 4: 0.575395/0.852000
Loss/Accuracy of Validation at epoch 4: 0.550999/0.870900
Accuracy of Testing at epoch 4: 0.878600
Loss/Accuracy of Training at epoch 5: 0.511136/0.893000
Loss/Accuracy of Validation at epoch 5: 0.503792/0.880600
Accuracy of Testing at epoch 5: 0.887800
Loss/Accuracy of Training at epoch 6: 0.472606/0.872000
Loss/Accuracy of Validation at epoch 6: 0.475978/0.886000
Accuracy of Testing at epoch 6: 0.894200
Loss/Accuracy of Training at epoch 7: 0.454293/0.879000
Loss/Accuracy of Validation at epoch 7: 0.442968/0.891500
Accuracy of Testing at epoch 7: 0.896100
Loss/Accuracy of Training at epoch 8: 0.433896/0.885000
Loss/Accuracy of Validation at epoch 8: 0.424141/0.895400
Accuracy of Testing at epoch 8: 0.899200
Loss/Accuracy of Training at epoch 9: 0.418188/0.900000
Loss/Accuracy of Validation at epoch 9: 0.412189/0.901200
Accuracy of Testing at epoch 9: 0.902100
Loss/Accuracy of Training at epoch 10: 0.406751/0.906000
Loss/Accuracy of Validation at epoch 10: 0.404409/0.901200
Accuracy of Testing at epoch 10: 0.905000
Loss/Accuracy of Training at epoch 11: 0.398014/0.901000
Loss/Accuracy of Validation at epoch 11: 0.399145/0.897900
Accuracy of Testing at epoch 11: 0.905200
Loss/Accuracy of Training at epoch 12: 0.388333/0.904000
Loss/Accuracy of Validation at epoch 12: 0.397591/0.901900
Accuracy of Testing at epoch 12: 0.906700
Loss/Accuracy of Training at epoch 13: 0.384753/0.907000
Loss/Accuracy of Validation at epoch 13: 0.379402/0.905200
Accuracy of Testing at epoch 13: 0.908600
Loss/Accuracy of Training at epoch 14: 0.377936/0.905000
Loss/Accuracy of Validation at epoch 14: 0.380904/0.906700
Accuracy of Testing at epoch 14: 0.910400
```

```
Loss/Accuracy of Training at epoch 15: 0.373657/0.917000
Loss/Accuracy of Validation at epoch 15: 0.373086/0.905800
Accuracy of Testing at epoch 15: 0.912300
Loss/Accuracy of Training at epoch 16: 0.369118/0.919000
Loss/Accuracy of Validation at epoch 16: 0.371598/0.908900
Accuracy of Testing at epoch 16: 0.912600
Loss/Accuracy of Training at epoch 17: 0.365699/0.916000
Loss/Accuracy of Validation at epoch 17: 0.363583/0.907000
Accuracy of Testing at epoch 17: 0.914400
Loss/Accuracy of Training at epoch 18: 0.361221/0.926000
Loss/Accuracy of Validation at epoch 18: 0.365981/0.907400
Accuracy of Testing at epoch 18: 0.915100
Loss/Accuracy of Training at epoch 19: 0.357256/0.924000
Loss/Accuracy of Validation at epoch 19: 0.368264/0.905300
Accuracy of Testing at epoch 19: 0.915200
Loss/Accuracy of Training at epoch 20: 0.353097/0.924000
Loss/Accuracy of Validation at epoch 20: 0.367816/0.909400
Accuracy of Testing at epoch 20: 0.916000
Loss/Accuracy of Training at epoch 21: 0.352362/0.912000
Loss/Accuracy of Validation at epoch 21: 0.358335/0.912500
Accuracy of Testing at epoch 21: 0.916800
Loss/Accuracy of Training at epoch 22: 0.351242/0.923000
Loss/Accuracy of Validation at epoch 22: 0.344182/0.916100
Accuracy of Testing at epoch 22: 0.917500
Loss/Accuracy of Training at epoch 23: 0.346933/0.922000
Loss/Accuracy of Validation at epoch 23: 0.351526/0.913200
Accuracy of Testing at epoch 23: 0.918200
Loss/Accuracy of Training at epoch 24: 0.343034/0.919000
Loss/Accuracy of Validation at epoch 24: 0.355231/0.912900
Accuracy of Testing at epoch 24: 0.919200
Loss/Accuracy of Training at epoch 25: 0.338584/0.909000
Loss/Accuracy of Validation at epoch 25: 0.365970/0.910500
Accuracy of Testing at epoch 25: 0.919100
Loss/Accuracy of Training at epoch 26: 0.340424/0.907000
Loss/Accuracy of Validation at epoch 26: 0.344333/0.916100
Accuracy of Testing at epoch 26: 0.920000
Loss/Accuracy of Training at epoch 27: 0.338932/0.908000
Loss/Accuracy of Validation at epoch 27: 0.339354/0.915600
Accuracy of Testing at epoch 27: 0.920700
Loss/Accuracy of Training at epoch 28: 0.338231/0.920000
Loss/Accuracy of Validation at epoch 28: 0.333026/0.919300
Accuracy of Testing at epoch 28: 0.920500
Loss/Accuracy of Training at epoch 29: 0.333044/0.928000
Loss/Accuracy of Validation at epoch 29: 0.343918/0.918800
Accuracy of Testing at epoch 29: 0.921400
Loss/Accuracy of Training at epoch 30: 0.332884/0.918000
Loss/Accuracy of Validation at epoch 30: 0.335759/0.920100
Accuracy of Testing at epoch 30: 0.921200
```


With Regulation
Training
Validation
Loss
Epoch
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
5
10
15
20
25
30


With Regulation
Training
Validation
Accuracy
Epoch
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0
5
10
15
20
25
30


With Regulation
T
Accuracy
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0
5
10
15
20
25
30
Epoch

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```