In [ ]:

```python
from functions import *
import matplotlib.pyplot as plt
import numpy as np
import pickle
from tqdm import tqdm
```

In [ ]:

```python
# Functions
def LoadBatch(file):

    with open("data/"+file, 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')

    pixelDat = dict[b'data']
    labels = dict[b'labels']
    labelsOneHot = np.zeros((len(labels),10))

    for index in range(len(labels)): # Not efficient :)
        labelsOneHot[index][labels[index]] = 1

    return pixelDat, labelsOneHot, labels

def CalcS(X, W, b):
    return W @ X + b

def EvaluateClassifier(X, W1, W2, b1, b2): # Returns the final P values and the intermidiary activation values
    s1 = CalcS(X, W1, b1) # s1 is mxD, TODO: Check if we should diagnoalize b1
    H = np.maximum(0, s1) # H is mxD, , returns the element-wise max of s1 and 0
    s2 = CalcS(H, W2, b2) # s2 is mxC
    P = softmax(s2)
    return P, H

def ComputeCost(X, Y, W1, W2, b1, b2, lamb):
    J = 0
    P,_ = EvaluateClassifier(X, W1, W2, b1, b2)
    P_t = np.clip(P.T, 1e-15, None)
    for i in range(len(P_t)):
        J += -np.dot(Y[i], np.log(P_t[i], where=P_t[i] > 0))
    J /= len(X[0]) # Divide by dimensionality
    loss = J # For documentation
    J += lamb * (np.sum(np.power(W1,2)) + np.sum(np.power(W2,2))) # WTerm

    return J, P, loss

def ComputeAccuracy(X, y, W1, W2, b1, b2):
    nCorr = 0
    P,_ = EvaluateClassifier(X, W1, W2, b1, b2)
    for index in range(X.T.shape[0]):
        p = P.T[index]
        predClass = np.argmax(p)
        if predClass == y[index]:
            nCorr += 1

    acc = nCorr/X.T.shape[0]
    return acc

def ComputeGradients(X, Y, P, H, W1, W2, lamb, b_start=0, b_size=20): # TODO: Convert
    X_batch, Y_batch, P_batch, H_batch = X.T[b_start:b_start+b_size].T, Y[b_start:b_start+b_size].T, P.T[b_star
    G_vec = - (Y_batch-P_batch)
    # print("weights[-1]:", W2.shape)
    # print("data[-1]:", H_batch.shape)
    # print("weights[-2]:", W1.shape)
    # print("data[-2]:", X_batch.shape)
    # print(G_vec.shape)
    dJdW2 = (G_vec @ H_batch.T)/b_size
    dJdb2 = np.sum(G_vec, axis=1)[:, np.newaxis]/b_size # error in notes?, check [:, np.newaxis]

    G_vec_2 = W2.T @ G_vec
    H_batch[H_batch<0] = 0
    H_batch[H_batch>0] = 1

    G_vec_2 = np.multiply(G_vec_2, H_batch)

    dJdW1 = G_vec_2 @ X_batch.T / b_size # grad_W1 is MxD
    dJdb1 = np.sum(G_vec_2, axis=1)[:, np.newaxis]/b_size
    grad_W1 = dJdW1 + 2*lamb*W1
    grad_b1 = dJdb1
    grad_W2 = dJdW2 + 2*lamb*W2
    grad_b2 = dJdb2
    return grad_W1, grad_W2, grad_b1, grad_b2

def init_variables2(): # More training data
    X_train, Y_train, X_val, Y_val, X_test, Y_test = None, None, None, None, None, None
    y_train = None
    for file in ["data_batch_1", "data_batch_2", "data_batch_3", "data_batch_4", "data_batch_5", "test_batch"]:
        X, Y, y = LoadBatch(file)
        mean_X = np.mean(X, axis=0)
        std_X = np.std(X, axis=0)
        X = X - mean_X
        X = X / std_X
        X = X.T # Make x stored in columns
        if file in ["data_batch_1","data_batch_3", "data_batch_4", "data_batch_5"]:
            if X_train is None:
                X_train = X
                Y_train = Y
                y_train = y
            else:
                X_train = np.concatenate((X_train, X), axis=1)
                Y_train = np.concatenate((Y_train, Y), axis=0)
                y_train += y

        elif file == "data_batch_2":
            X_val,Y_val, y_val = X, Y, y
        else:
            X_test,Y_test, y_test = X, Y, y

    np.random.seed(111)
    eta_min = 0.00001
    eta_max = 0.1
    m = 50 # ?
    K = 10 # Number of labels
    d = len(X.T[0]) # dimensionality
    W1 = np.random.normal(0, 1/np.sqrt(d), (m, d))
    W2 = np.random.normal(0, 1/np.sqrt(m), (K, m))
    b1 = np.zeros((m,1)) # np.random.normal(0, 0.01, (m,1))
    b2 = np.zeros((K,1)) # np.random.normal(0, 0.01, (K,1))
    n_s=500

    return X_train, Y_train, y_train, X_val, Y_val, y_val, X_test, Y_test, y_test, W1, W2, b1, b2, eta_min, eta

def init_variables(): # Excercise 1
    X_train, Y_train, X_val, Y_val, X_test, Y_test = None, None, None, None, None, None
    for file in ["data_batch_1", "data_batch_2", "test_batch"]:
        X, Y, y = LoadBatch(file)
        mean_X = np.mean(X, axis=0)
        std_X = np.std(X, axis=0)
        X = X - mean_X
        X = X / std_X
        X = X.T # Make x stored in columns
        if file == "data_batch_1":
            X_train,Y_train, y_train = X, Y, y
        elif file == "data_batch_2":
            X_val,Y_val, y_val = X, Y, y
        else:
            X_test,Y_test, y_test = X, Y, y

    np.random.seed(111)
    eta_min = 0.00001
    eta_max = 0.1
    m = 50 # ?
    K = 10 # Number of labels
    d = len(X.T[0]) # dimensionality
    W1 = np.random.normal(0, 1/np.sqrt(d), (m, d))
    W2 = np.random.normal(0, 1/np.sqrt(m), (K, m))
    b1 = np.zeros((m,1)) # np.random.normal(0, 0.01, (m,1))
    b2 = np.zeros((K,1)) # np.random.normal(0, 0.01, (K,1))
    n_s=500

    return X_train, Y_train, y_train, X_val, Y_val, y_val, X_test, Y_test, y_test, W1, W2, b1, b2, eta_min, eta
```

In [ ]:

```python
X_train, Y_train, y_train, X_val, Y_val, y_val, X_test, Y_test, y_test, W1, W2, b1, b2, eta_min, eta_max,_ = ir
```

In [ ]:

```python
print(X_train.shape)
```

In [ ]:

```python
lamb = 0
S1 = CalcS(X_train, W1, b1)
H = np.maximum(0, S1)
S2 = CalcS(H, W2, b2)
P = softmax(S2)
J,_,_ = ComputeCost(X_train, Y_train, W1, W2, b1, b2, lamb)
acc = ComputeAccuracy(X_train, y_train, W1, W2, b1, b2)
grad_W1, grad_W2, grad_b1, grad_b2 = ComputeGradients(X_train, Y_train, P, H, W1, W2, lamb, b_start=0, b_size=1
```

TEST START

In [ ]:

```python
grad_weights,grad_bias = compute_grads_num_2(X_train.T[0:10].T, Y_train.T[0:10].T, [W1, W2], [b1, b2], lamb, h=
```

In [ ]:

```python
grad_W1, grad_W2, grad_b1, grad_b2 = ComputeGradients(X_train, Y_train, P, H, W1, W2, lamb, b_start=0, b_size=1
```

In [ ]:

```python
for i in range(len(grad_bias[0])):
    if grad_bias[0][i]-grad_b1[i] > 1e-7:
        print(grad_bias[0][i]-grad_b1[i])

for i in range(len(grad_weights[0])):
    for j in range(len(grad_weights[0][i])):
        if grad_weights[0][i][j]-(grad_W1[i][j]) > 1e-7:
            print(grad_weights[0][i][j]-(grad_W1[i][j]))
# diff = np.subtract(grad_b1, grad_bias[0])
# diff = np.absolute(diff)

# print(diff.max())
```

TEST END

In [ ]:

```python
lrhist = []
def getLr(t, eta_min, eta_max, n_s):
    if (int(t/n_s))%2 == 0:
        return eta_min + (eta_max-eta_min)*((t%n_s)/n_s)
    else:
        return eta_max - (eta_max-eta_min)*((t%n_s)/n_s)


def MiniBatchGD(X, Y, y, W1, W2, b1, b2, lamb, n_epochs, n_batch, eta_min, eta_max, X_val, Y_val, y_val, n_s):
    acc_hist,cost_hist, loss_hist, acc_hist_val, cost_hist_val, loss_hist_val,loss_hist_val = [] ,[], [], [], 
    # Train, initial val
    acc = ComputeAccuracy(X, y, W1, W2, b1, b2)
    cost, _, loss = ComputeCost(X, Y, W1, W2, b1, b2, lamb)
    acc_hist.append(acc), cost_hist.append(cost), loss_hist.append(loss)
    # Validation, initial val
    acc = ComputeAccuracy(X_val, y_val, W1, W2, b1, b2)
    cost, _, loss = ComputeCost(X_val, Y_val, W1, W2, b1, b2, lamb)
    acc_hist_val.append(acc), cost_hist_val.append(cost), loss_hist_val.append(loss)
    t = 0
    for epoch in tqdm(range(n_epochs)): # Main loop
        for batch in range(int(len(Y)/n_batch)):
            t+=1
            lr = getLr(t, eta_min, eta_max, n_s)
            lrhist.append(lr)
            P, H = EvaluateClassifier(X, W1, W2, b1, b2)
            grad_W1, grad_W2, grad_b1, grad_b2 = ComputeGradients(X, Y, P, H, W1, W2, lamb, b_start=batch*n_bat
            W1 = W1 - grad_W1*lr
            W2 = W2 - grad_W2*lr
            # grad_b = grad_b.reshape(b.shape)
            b1= b1 - grad_b1*lr
            b2 = b2 - grad_b2*lr
        # Train
        acc = ComputeAccuracy(X, y, W1, W2, b1, b2)
        cost, _, loss = ComputeCost(X, Y, W1, W2, b1, b2, lamb)
        acc_hist.append(acc), cost_hist.append(cost), loss_hist.append(loss)
        # Validation
        acc = ComputeAccuracy(X_val, y_val, W1, W2, b1, b2)
        cost, _, loss = ComputeCost(X_val, Y_val, W1, W2, b1, b2, lamb)
        acc_hist_val.append(acc), cost_hist_val.append(cost), loss_hist_val.append(loss)
        print("Epoch:", epoch, "Accuracy:", acc_hist[-1])

    return W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val

lamb = 0.01
n_epochs = 24 # 1.5
n_batch = 100
n_s = 2 * int(len(X_train[0]))
eta = 0.001

l_min = -4
l_max = -3
search = [i for i in range(10)]
W1hist, W2hist, b1hist, b2hist, cost_hist_list, acc_hist_list, loss_hist_list, cost_hist_list_val, acc_hist_lis
n = 0
for i in search:
    X_train, Y_train, y_train, X_val, Y_val, y_val, X_test, Y_test, y_test, W1, W2, b1, b2, eta_min, eta_max, r
    print(str(n*10)+"%")
    n+=1
    l = l_min + (l_max - l_min) * i / len(search)
    lamb = 10**l
    # 10**-3.8 bra
    W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val = MiniBatchGD(X=
    W1hist.append(W1), W2hist.append(W2), b1hist.append(b1), b2hist.append(b2), cost_hist_list.append(cost_hist
```

In [ ]:

```python
x = [i for i in range(len(lrhist))]
plt.clf()
plt.title("Learning rate")
plt.plot(x, lrhist, label = "Lr")
plt.legend()
plt.show()
```

In [ ]:

```python
# Plotting
x = [i for i in range(n_epochs+1)]

for i in range(len(W1hist)):
    W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val = W1hist[i], W2h
    l = l_min + (l_max - l_min) * i / len(search)
    lamb = 10**l
    print("\nLambda:", lamb, "\n---------------------------------------")
    plt.clf()
    plt.title("Cost graph")
    plt.plot(x, cost_hist, label = "Training")
    plt.plot(x, cost_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    plt.clf()
    plt.title("Loss graph")
    plt.plot(x, loss_hist, label = "Training")
    plt.plot(x, loss_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    plt.clf()
    plt.title("Accuracy graph")
    plt.plot(x, acc_hist, label = "Training")
    plt.plot(x, acc_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    print("Final test accuracy:", ComputeAccuracy(X_test, y_test, W1, W2, b1, b2))
```

In [ ]:

```python
lrhist = []
def getLr(t, eta_min, eta_max, n_s):
    if (int(t/n_s))%2 == 0:
        return eta_min + (eta_max-eta_min)*((t%n_s)/n_s)
    else:
        return eta_max - (eta_max-eta_min)*((t%n_s)/n_s)


def MiniBatchGD(X, Y, y, W1, W2, b1, b2, lamb, n_epochs, n_batch, eta_min, eta_max, X_val, Y_val, y_val, n_s):
    acc_hist,cost_hist, loss_hist, acc_hist_val, cost_hist_val, loss_hist_val,loss_hist_val = [] ,[], [], [], 
    # Train, initial val
    acc = ComputeAccuracy(X, y, W1, W2, b1, b2)
    cost, _, loss = ComputeCost(X, Y, W1, W2, b1, b2, lamb)
    acc_hist.append(acc), cost_hist.append(cost), loss_hist.append(loss)
    # Validation, initial val
    acc = ComputeAccuracy(X_val, y_val, W1, W2, b1, b2)
    cost, _, loss = ComputeCost(X_val, Y_val, W1, W2, b1, b2, lamb)
    acc_hist_val.append(acc), cost_hist_val.append(cost), loss_hist_val.append(loss)
    t = 0
    for epoch in tqdm(range(n_epochs)): # Main loop
        for batch in range(int(len(Y)/n_batch)):
            t+=1
            lr = getLr(t, eta_min, eta_max, n_s)
            lrhist.append(lr)
            P, H = EvaluateClassifier(X, W1, W2, b1, b2)
            grad_W1, grad_W2, grad_b1, grad_b2 = ComputeGradients(X, Y, P, H, W1, W2, lamb, b_start=batch*n_bat
            W1 = W1 - grad_W1*lr
            W2 = W2 - grad_W2*lr
            # grad_b = grad_b.reshape(b.shape)
            b1= b1 - grad_b1*lr
            b2 = b2 - grad_b2*lr
        # Train
        acc = ComputeAccuracy(X, y, W1, W2, b1, b2)
        cost, _, loss = ComputeCost(X, Y, W1, W2, b1, b2, lamb)
        acc_hist.append(acc), cost_hist.append(cost), loss_hist.append(loss)
        # Validation
        acc = ComputeAccuracy(X_val, y_val, W1, W2, b1, b2)
        cost, _, loss = ComputeCost(X_val, Y_val, W1, W2, b1, b2, lamb)
        acc_hist_val.append(acc), cost_hist_val.append(cost), loss_hist_val.append(loss)
        print("Epoch:", epoch, "Accuracy:", acc_hist[-1])

    return W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val

lamb = 0.01
n_epochs = 30 # 1.5
n_batch = 100
n_s = 800 #2 * int(len(X_train[0]))
eta = 0.001

W1hist, W2hist, b1hist, b2hist, cost_hist_list, acc_hist_list, loss_hist_list, cost_hist_list_val, acc_hist_lis
X_train, Y_train, y_train, X_val, Y_val, y_val, X_test, Y_test, y_test, W1, W2, b1, b2, eta_min, eta_max, n_s =
lamb = 0.01
W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val = MiniBatchGD(X=X_tr
W1hist.append(W1), W2hist.append(W2), b1hist.append(b1), b2hist.append(b2), cost_hist_list.append(cost_hist), a
```

In [ ]:

```python
# Plotting
x = [i for i in range(n_epochs+1)]

for i in range(len(W1hist)):
    W1, W2, b1, b2, cost_hist, acc_hist, loss_hist, cost_hist_val, acc_hist_val, loss_hist_val = W1hist[i], W2h
    plt.clf()
    plt.title("Cost graph")
    plt.plot(x, cost_hist, label = "Training")
    plt.plot(x, cost_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    plt.clf()
    plt.title("Loss graph")
    plt.plot(x, loss_hist, label = "Training")
    plt.plot(x, loss_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    plt.clf()
    plt.title("Accuracy graph")
    plt.plot(x, acc_hist, label = "Training")
    plt.plot(x, acc_hist_val, label = "Valuation")
    plt.legend()
    plt.show()
    print("Final test accuracy:", ComputeAccuracy(X_test, y_test, W1, W2, b1, b2))
```

In [ ]:

```python
# Plotting
x = [i for i in range(len(lrhist))]
plt.clf()
plt.title("Learning rate graph")
plt.plot(x, lrhist, label = "lr")
plt.legend()
plt.show()
```