Parameters:

Initial learning rate: 0.005

Decay by 0.5 for every 4 epoch

Initialize W and V as Gaussian distribution with 0 mean and 0.05 standard deviation.

Stopping criterion:

When dW/W and dV/V are both smaller than 1e-5 or 1e-6. In this case, that is like the models are not really learning.

Tricks:

Data Augmentation:

The images are rotated by a random degree (0-10) and are used as new training data.

Ensemble:

Train three models independently, the final the prediction result is the ensemble of the three models.

Accuracies:

SDG only:

Train Accuracy 0.99836

Validation Accuracy 0.9702

Kaggle Test Accuracy 0.957

Train time: roughly 30min

Mini-batch with data augmentation and ensemble:

Train Accuracy 0.9994

Validation Accuracy 0.9801

Kaggle Test Accuract 0.9770

Train Time: roughly 45min

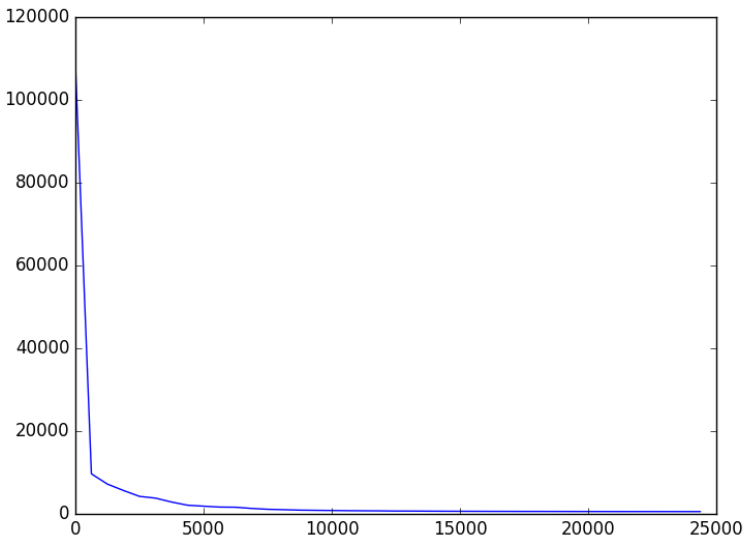
Final Kaggle Screen shot:

62	new	Xiaoyang Bai	0.97760	2	Thu, 03 Nov 2016 05:13:55
63	**↓33**	**HarryZeng**	**0.97740**	**5**	**Thu, 03 Nov 2016 13:35:18 (-3.1d)**
Your Best Entry ↑ Your submission scored **0.97700**, which is not an improvement of your best score. Keep trying!					
64	new	Onion Green	0.97720	4	Wed, 02 Nov 2016 00:03:34 (-0h)
65	↓34	MANUGOYAL	0.97700	1	Mon, 31 Oct 2016 05:44:57

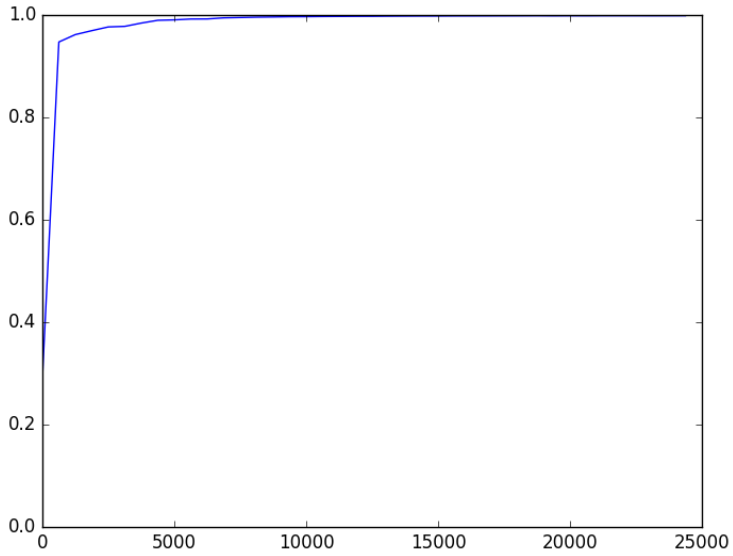
Plots:

All plots below are from mini-batch gradient descend.

Trainig Loss VS Iterations:



Training Accuracies VS Iterations:



7

b) Use notation in Lecture, we have

$\frac{\partial error}{\partial w_{ij}} = \frac{\partial error}{\partial s_j} \cdot \frac{\partial s_j}{\partial w_{ij}}$, where s_i is z_i in the problem.

From Lecture, we know $\frac{\partial s_i}{\partial w_{ij}} = x_{hid\,i}$

$$error = -\sum_{k=1}^{10} y_k \ln z_k(x)$$

$$= -\sum_k y_k \cdot \ln g_k(z)$$

$$\delta = \frac{\partial error}{\partial s_j} = \frac{\partial error}{\partial z_j}$$

$$= \frac{\partial -\sum_k y_k \cdot \ln g_k(z)}{\partial z_j}$$

$$= \frac{\partial -\sum_{k\neq j} y_k \cdot \ln g_k(z)}{\partial z_j} + \frac{\partial -y_j \cdot \ln g_j(z)}{\partial z_j}$$

$$= \sum_{k\neq j} y_k \cdot \frac{1}{g_k(z)} \cdot (-g_j(z) \cdot g_k(z)) - y_j \frac{1}{g_j(z)} \cdot g_j(z)(1 - g_j(z))$$

$$= \sum_{k\neq j} y_k g_j(z) + y_j g_j(z) - y_j$$

$$= \sum y_k g_j(z) - y_j = g_j(z) - y_j$$

$$\frac{\partial error}{\partial w_{ij}} = \left(\sum y_k g_j(w \cdot x_{hidden}) - y_j\right) \cdot x_{hidden,\,i}$$

$$\frac{\partial error}{\partial w} = \begin{bmatrix} g_1(w \cdot x_{hid}) - y_1 \\ \vdots \\ g_{10}(w \cdot x_{hid}) - y_{10} \end{bmatrix} \cdot \begin{bmatrix} x_{hid,1} \\ \vdots \\ x_{hid,200} \\ x_{hid\,201} \end{bmatrix}^T$$

$$= (G(w \cdot x_{hid}^T) - Y^T) \cdot x_{hid}$$

In [ ]:

```
from mnist import MNIST
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
import csv
import numpy as np
import scipy
import scipy.ndimage
import scipy.stats as stats

NUM_CLASSES = 10
SIGMA = 0.05
PIE = 3.1415926

# Under DEBUGGING mode, only run on 10% of train data.
# Just to make sure things are on the right track.
DEBUGGING = False
if (DEBUGGING):
    T_SIZE = 6000
    V_SIZE = 1000
else:
    T_SIZE = 60000
    V_SIZE = 10000
KAGGLE = False
EMSEMBLE = False
N_HID = 200

def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, _ = map(np.array, mndata.load_testing())
    return X_train, labels_train, X_test

def scale(X):
    X_normalized = np.zeros(X.shape)
    for i in range(0, X.shape[1]):
        mean = np.mean(X[:,i])
        sd = np.std(X[:,i])
        if (sd == 0):
            sd = 1e-8
        X_normalized[:,i] = (X[:,i] - mean) /  sd
    return X_normalized

def rotate(X_train, theta = 10):
    X_rotated = np.zeros(X_train.shape)
    for i in range(0, X_train.shape[0]):
        Xi = np.matrix(X_train[i]).reshape((28, 28))
        degree = np.random.rand() * theta
        sign = np.random.rand()
        if sign > 0.5:
            degree = -degree
        Xi = scipy.ndimage.interpolation.rotate(Xi,degree,reshape = Fals
e, mode = 'nearest')
        Xi = Xi.reshape((1, 784))
        X_rotated[i] = Xi
    return X_rotated

def one_hot(labels_vec):
```

```
    labels = np.zeros([labels_vec.size, NUM_CLASSES])
    labels[np.arange(labels_vec.size), labels_vec] = 1
    return labels

def preprocess(X_train, labels_train, X_test):


    X_train, X_test = scale(X_train), scale(X_test)

    # Shuffling
    # Reference:
    # http://stackoverflow.com/questions/4601373/better-way-to-shuffle-t
wo-numpy-arrays-in-unison
    rng_state = np.random.get_state()
    np.random.shuffle(X_train)
    np.random.set_state(rng_state)
    np.random.shuffle(labels_train)

    X_valid, labels_valid = X_train[0:V_SIZE], labels_train[0:V_SIZE]
    X_train, labels_train = X_train[V_SIZE:T_SIZE],
labels_train[V_SIZE:T_SIZE]

    # Data agmentation on training sets
    # X_train_rotated = rotate(X_train)
    # X_train = np.r_[X_train, X_train_rotated]
    # labels_train = np.r_[labels_train, labels_train]

    # Shuffle
    rng_state = np.random.get_state()
    np.random.shuffle(X_train)
    np.random.set_state(rng_state)
    np.random.shuffle(labels_train)

    # Add ones for bias
    X_train = np.c_[ X_train, np.ones(X_train.shape[0]) ]
    X_valid = np.c_[ X_valid, np.ones(X_valid.shape[0]) ]
    X_test = np.c_[ X_test, np.ones(X_test.shape[0]) ]

    return (X_train, labels_train), (X_valid, labels_valid), X_test

def predict(W, V, X):
    ''' From model and data points, output prediction vectors '''
    X_hid = V.dot(X.T)
    ones = np.ones(X.shape[0])
    ones = ones.reshape((1,X.shape[0]))
    X_hid = np.r_[ X_hid, ones ]
    X_hid = relu(X_hid)
    Z = W.dot(X_hid).T
    # Z = softmax(Z)
    results = Z.argmax(axis = 1)
    return Z, results

def emsemble_predict(Z1, Z2, Z3):
    r1 = Z1.argmax(axis = 1)
    r2 = Z2.argmax(axis = 1)
    r3 = Z3.argmax(axis = 1)
    r = np.c_[r1, r2, r3]
```

```
    r_emsemble = np.zeros(r1.shape)
    for i in range(r1.shape[0]):
        ri = r[i]
        ri = np.asarray(ri)[0]
        counts = np.bincount(ri)
        r_emsemble[i] = np.argmax(counts)
        if max(counts) == 1:
            r_emsemble[i] = ri[0]
    print(np.c_[r, r_emsemble])
    return r_emsemble

def progress(train_accuracy_arr, valid_accuracy_arr, loss_arr, i, num_it
er, W, V = []):
    print("{0:.2f}%".format(i / num_iter * 100))
    Z_train, pred_labels_train = predict(W, V, X_train)
    Z_valid, pred_labels_valid = predict(W, V, X_valid)
    train_accuracy = metrics.accuracy_score(labels_train, pred_labels_tr
ain)
    valid_accuracy = metrics.accuracy_score(labels_valid, pred_labels_va
lid)
    train_accuracy_arr.append(train_accuracy)
    valid_accuracy_arr.append(valid_accuracy)
    loss = cross_entro(Y_train, Z_train)
    loss_arr.append(loss)
    print("Train accuracy: {0}".format(train_accuracy))
    print("Validation accuracy: {0}".format(valid_accuracy))
    print("Training loss: {0:.2f}".format(loss))
    return train_accuracy_arr, valid_accuracy_arr, loss_arr

def softmax(x):
    e_x = np.exp(x - np.max(x, axis = 0))
    return e_x / e_x.sum(axis=0)

def low_b(x):
    if (x < 1e-8):
        return 1e-8
    else:
        return x
low_b = np.vectorize(low_b)

def cross_entro(Y, Z):
    Z_T = low_b(softmax(Z.T))
    return - np.sum(np.einsum('ij,ji->i', Y, np.log(Z_T)))

def relu(x):
    if x > 0:
        return x
    else:
        return 0
relu = np.vectorize(relu)

def relu_d(x):
    if x > 0:
        return 1
    else:
        return 0
relu_d = np.vectorize(relu_d)
```

```
def train_sgd(X_train, Y_train, epo=4, alpha=0.005, lam=0.5, reg_v =
0.0, reg_w = 0.0, num_iter=25000, batch_size = 50):
    ''' Build a model from X_train -> Y_train using stochastic gradient
 descent '''
    epo = epo / batch_size

    if (DEBUGGING):
        num_iter = num_iter // 100
    trn_ac = []
    trn_ls = []
    val_ac = []

    W = np.random.randn(NUM_CLASSES, N_HID + 1) * SIGMA
    V = np.random.randn(N_HID, X_train.shape[1]) * SIGMA

    for i in range(0, num_iter):
        sample_index = np.random.randint(X_train.shape[0],size=batch_siz
e)
        Xi = np.matrix(X_train[sample_index])
        Yi = np.matrix(Y_train[sample_index])

        X_hid_T = relu(V.dot(Xi.T))
        X_hid_T_bias = np.r_[ X_hid_T, np.ones((1,batch_size)) ]
        delta = softmax(W.dot(X_hid_T_bias)) - Yi.T
        dV = alpha * (np.multiply(relu_d(X_hid_T), W[:,0:N_HID].T.dot(de
lta)).dot(Xi) - V * reg_v / X_train.shape[0])
        dW = alpha * (delta.dot(X_hid_T_bias.T) - W * reg_w / N_HID)

        V = V - dV
        W = W - dW

        # Modify alpha
        if (i % (X_train.shape[0] * epo) == 0):
            alpha = alpha * lam

        # Print and record progress
        if (i % (num_iter // 40) == 0):
            trn_ac, val_ac, trn_ls = progress(trn_ac, val_ac, trn_ls, i,
 num_iter, W, V)

        # # Record the accuracy of the first 10% of iterations
        # if (i < num_iter // 10 and i % (num_iter // 1000) == 0):
        #     trn_ac, val_ac = progress(trn_ac, val_ac, i, num_iter, W,
 V)

    Plot
    iters = list(range(0, num_iter, num_iter // 40))
    plt.plot(iters, trn_ac)
    plt.axis([0, num_iter, 0, 1])
    plt.savefig('sdg_train_accuracies.png')
    plt.clf()
    plt.plot(iters, trn_ls)
    plt.savefig('sdg_train_losses.png')
    return W, V
```

```
if __name__ == "__main__":
    X_train, labels_train, X_test = load_dataset()
    (X_train, labels_train), (X_valid, labels_valid), X_test = preproces
s(X_train, labels_train, X_test)
    Y_train = one_hot(labels_train)

    if (EMSEMBLE):
        W1, V1 = train_sgd(X_train, Y_train)
        Z1 = predict(W1, V1, X_valid)[0]
        Z1_train = predict(W1, V1, X_train)[0]

        W2, V2 = train_sgd(X_train, Y_train)
        Z2 = predict(W2, V2, X_valid)[0]
        Z2_train = predict(W1, V1, X_train)[0]

        W3, V3 = train_sgd(X_train, Y_train)
        Z3 = predict(W3, V3, X_valid)[0]
        Z3_train = predict(W1, V1, X_train)[0]

        pred_labels_valid = emsemble_predict(Z1, Z2, Z3)
        pred_labels_train = emsemble_predict(Z1_train, Z2_train, Z3_trai
n)
    else:


        W, V = train_sgd(X_train, Y_train)
        pred_labels_train = predict(W, V, X_train)[1]
        pred_labels_valid = predict(W, V, X_valid)[1]

    print("Final results:")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_tra
in, pred_labels_train)))
    print("Validation accuracy: {0}".format(metrics.accuracy_score(label
s_valid, pred_labels_valid)))

    if (KAGGLE):
        if (EMSEMBLE):
            # W1, V1 = train_sgd(X_train, Y_train)
            Z1 = predict(W1, V1, X_test)[0]
            # W2, V2 = train_sgd(X_train, Y_train)
            Z2 = predict(W2, V2, X_test)[0]
            # W3, V3 = train_sgd(X_train, Y_train)
            Z3 = predict(W3, V3, X_test)[0]
            cat = emsemble_predict(Z1, Z2, Z3)
        else:
            # Categories
            cat = predict(W, V, X_test)[1]
        # Category column of output
        id = list(range(0,len(cat)))
        output = np.c_[np.matrix(id).T,cat]
        np.savetxt("foo.csv", output, delimiter=',', header="Id,Categor
y")
        print("updated foo")
```

index j denote output layer.
index i denote hidden layer
index l denote input layer

$$s_j = \sum_i w_{ij} \cdot x_i$$

$$s_i = \sum_l w_{li} \cdot x_l$$

$$\frac{\partial \text{error}}{\partial v_{li}} = \sum_j \frac{\partial \text{error}}{\partial s_j} \cdot \frac{\partial s_j}{\partial x_i} \cdot \frac{\partial x_i}{\partial s_i} \cdot x_l$$

$$= \sum_j \frac{\partial \text{error}}{\partial s_j} \cdot w_{ij} \cdot I_{si} \cdot x_l$$

I_{si} : indicator of $\sum_l v_{li} x_l > 0$ or not.

$$= \sum_j (g_j(z) - y_j) \cdot w_{ij} \cdot I_{si} \cdot x_l$$

$$\frac{\partial \text{error}}{\partial v} = I_{hid} \cdot W \cdot \delta \cdot X$$

I_{hid} is a diagonal matrix. Its diagonal element i denotes whether $X_{hid,i}$ is greater than 0.

$$\delta = G(W \cdot X_{hid}^T) - Y^T$$