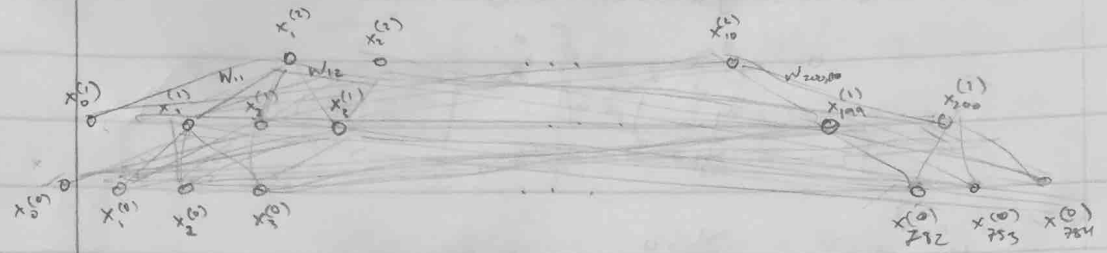
1. Derive SGD



① $g_j^{(2)}(s) = \dfrac{e^{s_j^{(2)}}}{\sum_{k=1}^{num} e^{s_k^{(2)}}} \rightarrow \dfrac{\partial g_j^{(2)}}{\partial s_i} = \begin{cases} g_j(s)(1-g_j(s)) & i=j \\ -g_i(s)\,g_j(s) & i \neq j \end{cases}$

② $g_j^{(1)}(s) = \max(0, s)$

③ $J = -\sum_{k=1}^{num} y_k \ln g_k^{(2)}(s)$ (cross-Entropy loss)

④ $W_{new} \leftarrow W_{old} - \eta \nabla_W J$

Compute $\nabla_W J = \begin{bmatrix} \partial J/\partial w_{11} \\ \vdots \\ \partial J/\partial w_{ij} \\ \vdots \end{bmatrix}$

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \underbrace{\frac{\partial J}{\partial s_j^{(l)}}}_{\delta_j^{(l)}} \underbrace{\frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}}_{x_i^{(l-1)}}$$

Base case $l = L = 2$

$$\delta_j^{(l)} = \frac{\partial J}{\partial s_j^{(l)}} = -\frac{\partial}{\partial s_j^{(l)}} \sum_{k=1}^{num} y_k \ln g_k^{(2)}(s)$$

$$= -\sum_{k=1}^{num} y_k \frac{\partial}{\partial s_j^{(l)}} \ln g_k^{(2)}(s)$$

$$= -\sum_{k=1}^{num} \frac{y_k}{g_k^{(2)}(s)} \frac{\partial g_k^{(2)}(s)}{\partial s_j}$$

$$= -\frac{y_j}{g_j^{(2)}(s)} g_j^{(2)}(s)(1 - g_j^{(2)}(s)) + \sum_{k \neq j}^{num} \frac{y_k}{g_k^{(2)}(s)} (-g_k(s)\, g_j(s))$$

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

$$= \sum_{k=1}^{num} y_k g_j^{(2)}(s) - y_j$$

$$= g_j^{(2)}(s) \sum_{k=1}^{num} y_k - y_j = g_j^{(2)}(s) - y_j$$

$$= \frac{e^{s_j}}{\sum e^{s_k}} - y_j$$

Induction case : $l < L \quad (l=1)$

$$\delta_j^{(l-1)} = \frac{\partial J}{\partial s_j^{(l-1)}} = \sum_i \frac{\partial \sigma}{\partial s_i^{(l)}} \frac{\partial s_i^{(l)}}{\partial x_j^{(l-1)}} \frac{\partial x_j^{(l-1)}}{\partial s_j^{(l-1)}}$$

$$= \sum_i \delta_i^{(l)} w_{ij}^{(l)} \cdot g_j^{(l-1)}(s)$$

$$= g_j'^{(l-1)}(s) \sum_i \delta_i^{(l)} w_{ij}^{(l)}$$

$$= \begin{cases} \sum \delta_i^{(2)} w_{ij}^{(2)} & \text{if } s_j^{(1)} > 0 \quad (x_j^{(1)} > 0) \\ 0 & \text{if } s_j^{(1)} \le 0 \quad (x_j^{(1)} = 0) \end{cases}$$

$$\frac{\partial J}{\partial w_{ij}} = \delta_j^{(2)} x_i^{(1)} \qquad \frac{\partial J}{\partial v_{ij}} = \delta_j^{(1)} x_i^{(0)}$$

In [2]:
```
from mnist import MNIST
import random
import sklearn.metrics as metrics
from sklearn import preprocessing
import matplotlib.pyplot as plt
import numpy as np
import scipy
import pdb
import warnings
import csv

%matplotlib inline

NUM_FEATURES = 784
NUM_HIDDEN = 200
NUM_CLASSES = 10
```

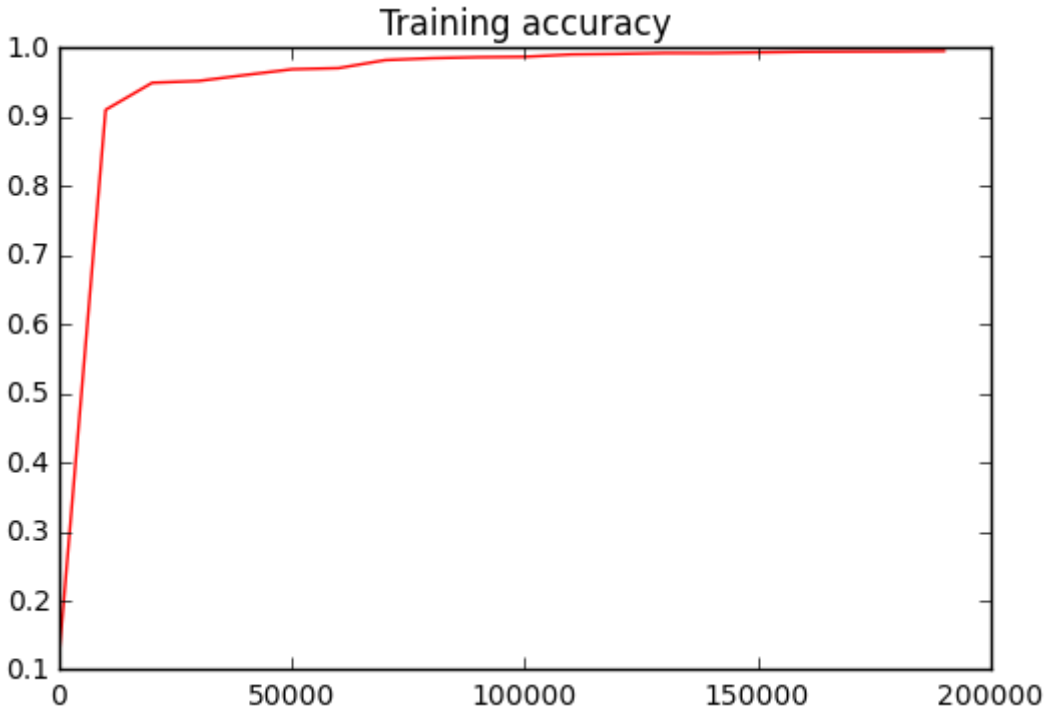
In [3]:

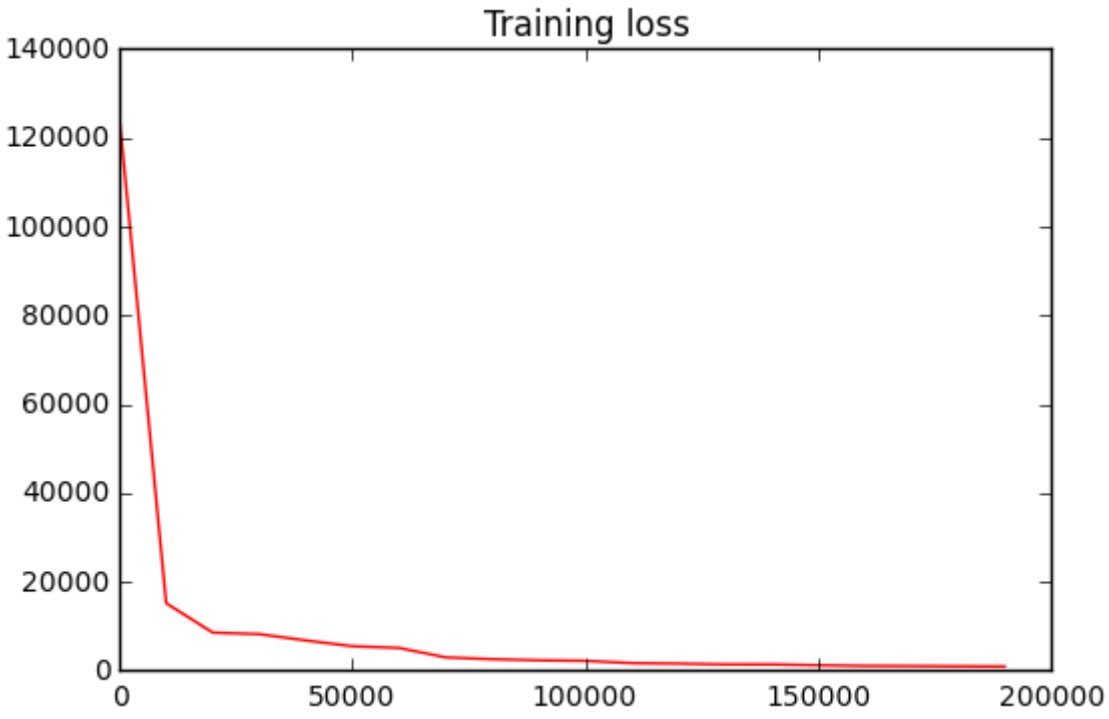
```
def load_dataset():
    mndata = MNIST('data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, _ = map(np.array, mndata.load_testing())
    return X_train, labels_train, X_test

def split_dataset(data, size=50000):
    return data[:size], data[size:]

def train_neural_network(X_train, y_train, alpha=0.05, decay=0.5, num_it
er=10000):
    epoch = X_train.shape[0]
    V = np.random.normal(0, 0.1, (NUM_FEATURES+1, NUM_HIDDEN))
    W = np.random.normal(0, 0.1, (NUM_HIDDEN+1, NUM_CLASSES))
    training_accuracies = []
    training_losses = []
    for i in range(num_iter):
        if i % 1000 == 0:
            print("iteration i: " + str(i))
        if i % epoch == 0:
            alpha *= decay
            print("alpha is: " + str(alpha))
        if i % 10000 == 0:
            pred_labels_train_iter = predict(V, W, X_train)
            labels_train = np.argmax(y_train, axis=1)
            # training accuracy
            train_accuracy = metrics.accuracy_score(labels_train, pred_l
abels_train_iter)
            print("Iteration " + str(i) + " Training accuracy: {0}".form
at(train_accuracy))
            training_accuracies.append(train_accuracy)
            # training loss
            pre_out = add_one(relu(X_train.dot(V))).dot(W)
            output = np.apply_along_axis(softmax, axis=1, arr=pre_out)
            train_loss = cross_entropy_loss(output, y_train)
            print("Iteration " + str(i) + " Training loss: {0}".format(t
rain_loss))
            training_losses.append(train_loss)

        sample_index = random.randint(0, X_train.shape[0]-1)

        x_0, y = X_train[sample_index], y_train[sample_index]
        x_1 = np.append(relu(x_0.dot(V)), np.ones(1))
        x_2 = softmax(x_1.dot(W))

        delta_2 = x_2 - y
        delta_1 = mask(W.dot(delta_2)*np.where(x_1>0.0, 1.0, 0.0), 200)

        gradient_V = np.outer(x_0, delta_1)
        gradient_W = np.outer(x_1, delta_2)
        V -= alpha * gradient_V
        W -= alpha * gradient_W
    return V, W, training_accuracies, training_losses

def predict(V, W, X):
    ''' From model and data points, output prediction vectors '''
    pre_out = add_one(relu(X.dot(V))).dot(W)
```

```
    output = np.apply_along_axis(softmax, axis=1, arr=pre_out)
    return np.argmax(output, axis=1)

def relu(s):
    '''Hidden layer activation function'''
    return np.maximum(0, s)

def softmax(s):
    '''Output layer activation function as multi-class case of sigmoi
d'''
    s -= np.max(s)
    return normalize(np.exp(s))

def cross_entropy_loss(output, y):
    loss = 0
    for i in range(y.shape[0]):
        loss += -y[i].dot(np.log(output[i]))
    return loss

def one_hot(labels_train):
    '''Convert categorical labels 0,1,2,....9 to standard basis vectors
 in R^{10} '''
    return np.eye(NUM_CLASSES)[labels_train]

def add_one(X):
    return np.c_[X, np.ones(X.shape[0])]

def mask(X, shape):
    mask = np.ones(201, dtype=bool)
    mask[200] = False
    return X[mask]

def standardize(X, data_mean):
    ''' Standardize columns to have mean 0 and unit variance
        Geometrically whitens everything but the important region'''
    return (X - data_mean)/255.0

def normalize(X):
    ''' Normalize so columns sum to one '''
    return X / np.sum(X)

def show_image(X, i):
    im = X[i].reshape(28, 28)*255 #Image.fromarray(X[i].reshape(28, 28)*
255)
    plt.gray()
    plt.imshow(im)

def training_loss(X_train, y_train, X_validate, y_validate, alpha=1e-4,
decay=0, num_iter=200000):
    training_accuracies = []
    validation_accuracies = []
    for num_iter in np.arange(0, num_iter, 20000):
        V, W = train_neural_network(X_train, y_train, alpha, decay, num_
iter)
        pred_y_train = predict(V, W, X_train)
        pred_y_validate = predict(V, W, X_validate)
        train_accuracy = metrics.accuracy_score(y_train, pred_y_train)
```

```
        validate_accuracy = metrics.accuracy_score(y_validate, pred_y_va
lidate)
        print("Train accuracy with alpha: " + str(alpha) + ", decay: " +
 str(decay) + ", and iterations: {0}".format(train_accuracy))
        print("Validation accuracy with alpha: " + str(alpha) + ", deca
y: " + str(decay) + ", and iterations: {0}".format(validate_accuracy))
        training_accuracies.append(train_accuracy)
        validation_accuracies.append(validate_accuracy)
    return training_accuracies, validate_accuracies

def plot_metric(metric, title):
    iterations = np.arange(0, 200000, 10000)
    plt.plot(iterations, metric, 'r-')
    plt.title(title)
    plt.show()
    # plt.savefig(title)
```

In [4]:
```
if __name__ == "__main__":
    X_train, labels_train, X_test = load_dataset()

    X_train, X_validate = split_dataset(X_train.astype(float))
    labels_train, labels_validate = split_dataset(labels_train)
    # preprocessing statistics only computed on training set
    data_mean = np.mean(X_train, axis=0)
    X_train, X_validate, X_test = standardize(X_train, data_mean), stand
ardize(X_validate, data_mean), standardize(X_test, data_mean)
    # add one
    X_train, X_validate, X_test = add_one(X_train), add_one(X_validate),
 add_one(X_test)

    y_train = one_hot(labels_train)
    y_validate = one_hot(labels_validate)


    V, W, accuracies, losses = train_neural_network(X_train, y_train, al
pha=0.05, decay=0.5, num_iter=200000)
    pred_labels_train = predict(V, W, X_train)
    pred_labels_validate = predict(V, W, X_validate)

    # Preload V, W
    #V = np.load("V.npy")
    #W = np.load("W.npy")
    pred_labels_test = predict(V, W, X_test)

    print("Neural Network")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_tra
in, pred_labels_train)))
    print("Validation accuracy: {0}".format(metrics.accuracy_score(label
s_validate, pred_labels_validate)))

    np.save("V", V)
    np.save("W", W)
    np.save("data_mean", data_mean)
    np.save("accuracies", accuracies)
    np.save("losses", losses)

    c = csv.writer(open("kaggle.csv", "wt"))
    c.writerow(['Id', 'Category'])
    for i in range(len(pred_labels_test)):
        c.writerow((i+1, int(pred_labels_test[i])))

    #accuracies = np.load("accuracies.npy")
    #losses = np.load("losses.npy")

    plot_metric(accuracies, 'Training accuracy')
    plot_metric(losses, 'Training loss')
```


Training accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
50000
100000
150000
200000


Training loss
140000
120000
100000
80000
60000
40000
20000
0
0
50000
100000
150000
200000

Parameters:

- learning rate (alpha) = 0.05
- decay rate (decay) = 0.5
- number of iterations (num_iter) = 200000
- weight initialization = gaussian(0, 0.1)

Running Time:

- 3 minutes

Kaggle Score: 0.97940

In [ ]: