Jesse Li
SID: 23822462
CS189
HW6 Writeup

1) $$z_k = s(W_k \cdot h) \qquad s'(x) = s(x)(1 - s(x))$$
$$h_l = \tanh(V_l \cdot x) \qquad \tanh'(x) = 1 - \tanh^2 x$$

W_k is a row of W, h_l is the output of hidden unit l, V_l is a row of V.

$$\frac{dJ}{dW_k} = \left(\frac{dJ}{dz_k}\right)\left(\frac{dz_k}{dW_k}\right)$$
$$\frac{dJ}{dV_l} = \left(\frac{dJ}{dh_l}\right)\left(\frac{dh_l}{dV_l}\right) = \left(\sum_{k=1}^{10}\left(\frac{dJ}{dz_k}\right)\left(\frac{dz_k}{dh_l}\right)\right)\left(\frac{dh_l}{dV_l}\right)$$

Mean squared error: $$J = \frac{1}{2}\sum_{k=1}^{n_{out}}(y_k - z_k(x))^2$$

$$\frac{dJ}{dz_k} = z_k - y_k \qquad \frac{dz_k}{dW_k} = s'(W_k \cdot h)h = s(W_k \cdot h)(1 - s(W_k \cdot h))h$$
$$\frac{dJ}{dW_k} = (z_k - y_k)s(W_k \cdot h)(1 - s(W_k \cdot h))h \quad \rightarrow \quad \boldsymbol{\frac{dJ}{dW_{ij}} = (z_i - y_i)s(W_i \cdot h)(1 - s(W_i \cdot h))h_j}$$

$$\frac{dz_k}{dh_l} = s'(W_k \cdot h)W_{kl} = s(W_k \cdot h)(1 - s(W_k \cdot h))W_{kl} \qquad \frac{dh_l}{dV_l} = \left(1 - \tanh^2(V_l \cdot x)\right)x$$
$$\frac{dJ}{dV_l} = \left(\sum_{k=1}^{10}(z_k - y_k)\left(s(W_k \cdot h)(1 - s(W_k \cdot h))W_{kl}\right)\right)\left(1 - \tanh^2(V_l \cdot x)\right)x$$
$$\boldsymbol{\frac{dJ}{dV_{ij}} = \left(\sum_{k=1}^{10}(z_k - y_k)\left(s(W_k \cdot h)(1 - s(W_k \cdot h))W_{ki}\right)\right)\left(1 - \tanh^2(V_i \cdot x)\right)}x_j$$

Cross-entropy error: $$J = -\sum_{k=1}^{n_{out}}[y_k \ln z_k(x) + (1 - y_k)\ln(1 - z_k(x))]$$

$$\frac{dJ}{dz_k} = -\frac{y_k}{z_k} + \frac{1 - y_k}{1 - z_k} = \frac{-y_k + z_k}{z_k(1 - z_k)} \qquad \frac{dz_k}{dW_k} = s(W_k \cdot h)(1 - s(W_k \cdot h))h$$
$$\boldsymbol{\frac{dJ}{dW_{ij}} = \frac{-y_i + z_i}{z_i(1 - z_i)}s(W_i \cdot h)(1 - s(W_i \cdot h))h_j}$$

$$\frac{dz_k}{dh_l} = s(W_k \cdot h)(1 - s(W_k \cdot h))W_{kl} \qquad \frac{dh_l}{dV_l} = \left(1 - \tanh^2(V_l \cdot x)\right)x$$
$$\boldsymbol{\frac{dJ}{dV_{ij}} = \left(\sum_{k=1}^{10}\frac{-y_k + z_k}{z_k(1 - z_k)}\left(s(W_k \cdot h)(1 - s(W_k \cdot h))W_{ki}\right)\right)\left(1 - \tanh^2(V_i \cdot x)\right)x_j}$$

$W_{ij} = W_{ij} - \lambda(dJ/dW_{ij})$
$V_{ij} = V_{ij} - \lambda(dJ/dV_{ij})$

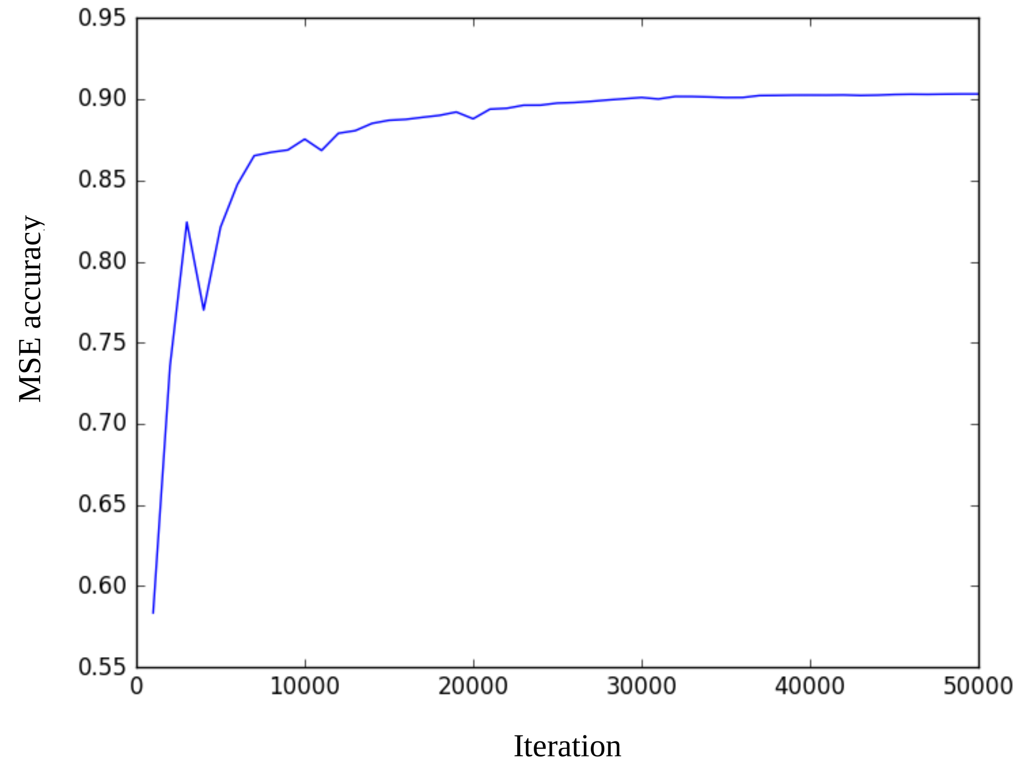
2) Learning rate for the mean squared error was initialized at 0.1 and the learning rate was decayed by multiplying by 0.9 each epoch. The learning rate for the cross entropy error was initialized at 0.01 and decayed by a factor of 0.7 each epoch. Training for both errors was stopped once the validation error stopped improving. The weights were initialized from a Gaussian distribution with mean 0 and standard deviation 0.01.

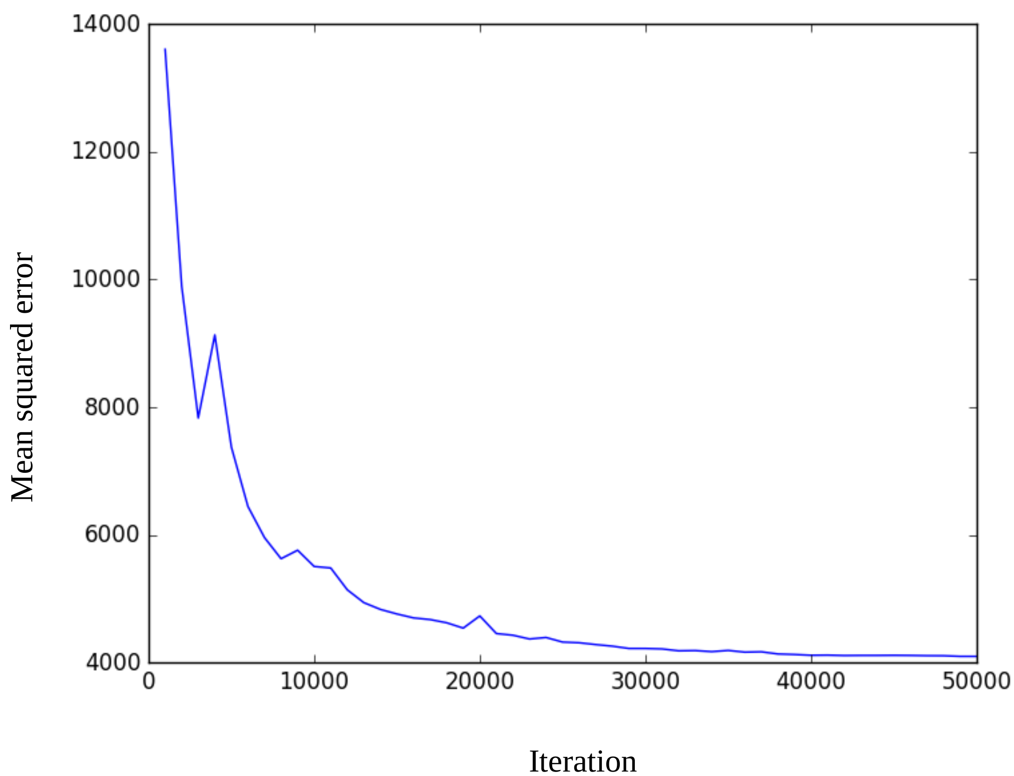
Best mean-squared error training accuracy = 0.97974
Best mean-squared error validation accuracy = 0.98

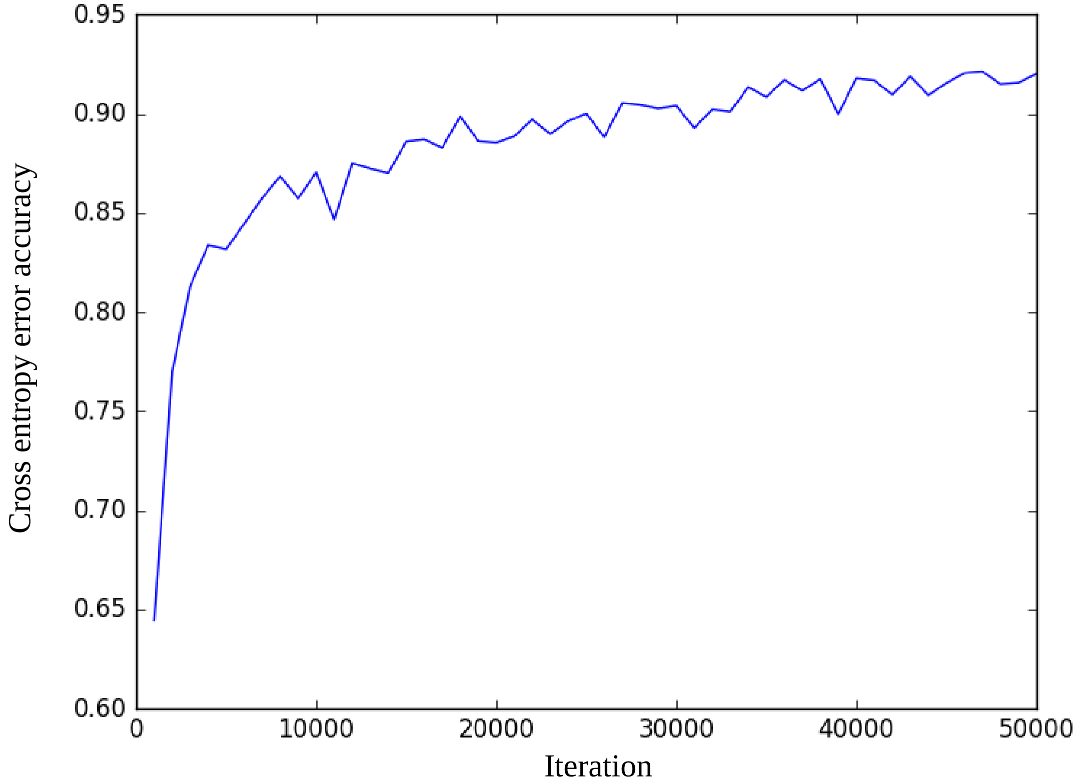
Best cross-entropy error training accuracy = 0.9067
Best cross-entropy error validation accuracy = 0.93894

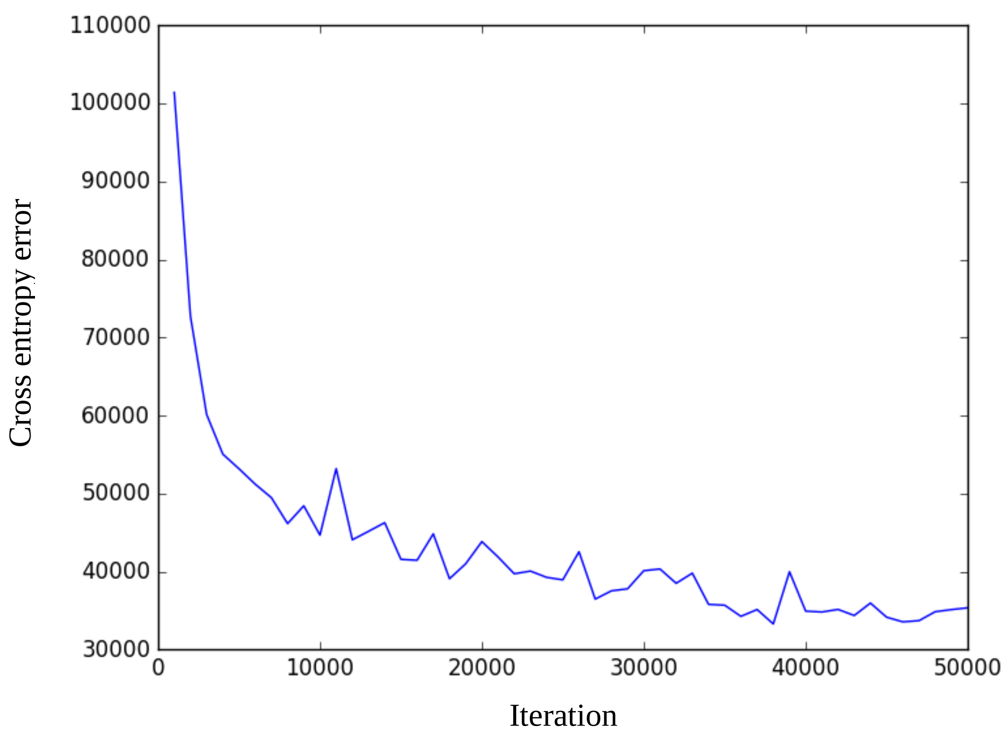
Best Kaggle score = 0.96620

For the MSE, took 1 epoch (3 minutes) to reach about 90% accuracy on training set. Learning rate was manually adjusted after 95% accuracy to reach higher accuracy rates. Cross entropy error took 1 epoch (3 minutes) to reach 92% training accuracy.


MSE accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
10000
20000
30000
40000
50000
Iteration


Mean squared error
14000
12000
10000
8000
6000
4000
0
10000
20000
30000
40000
50000
Iteration


0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Cross entropy error accuracy
0
10000
20000
30000
40000
50000
Iteration


110000
100000
90000
80000
70000
60000
50000
40000
30000
Cross entropy error
0
10000
20000
30000
40000
50000
Iteration

Using the cross-entropy error seems to give faster convergence, but also seems more sensitive to an improperly tuned learning rate. The decreased stability and increased learning rate can be seen in the above plots. I was able to get better overall performance from the mean squared error with a longer training time and smaller relative amount of learning rate tuning.

```
import numpy as np
import scipy.io
import random
import math
import matplotlib.pyplot as plt
import pickle
import pdb
import sklearn.preprocessing


def benchmark(pred_labels, true_labels):
    errors = pred_labels != true_labels
    err_rate = sum(errors) / float(len(true_labels))
    indices = errors.nonzero()
    return err_rate, indices

def montage_images(images):
    num_images=min(1000,np.size(images,2))
    numrows=math.floor(math.sqrt(num_images))
    numcols=math.ceil(num_images/numrows)
    img=np.zeros((numrows*28,numcols*28));
    for k in range(num_images):
        r = k % numrows
        c = k // numrows
        img[r*28:(r+1)*28,c*28:(c+1)*28]=images[:,:,k];
    return img

class neural_net(object):
    def __init__(self, inputLayerSize = 784, hiddenLayerSize = 200, outputLayerSize =
10, lamb = 1e-2, decay = 1, load = False):
        self.inputLayerSize = inputLayerSize
        self.outputLayerSize = outputLayerSize
        self.hiddenLayerSize = hiddenLayerSize
        # self.error_limit = error_limit
        self.lamb = lamb
```

```
        self.decay = decay
        self.load = load

    def squaredErrorTrain(self,images,vec_labels):
        if self.load:
            V,W = pickle.load(open( "best.p", "rb" ))
        else:
            V = np.random.normal(0,0.01,(self.hiddenLayerSize, self.inputLayerSize + 1))
            W = np.random.normal(0,0.01,(self.outputLayerSize, self.hiddenLayerSize + 1))
        iteration = 0
        samples = images.shape[0]
        indices = np.arange(samples)
        while True: #error > self.error_limit:
            if iteration % samples == 0:
                np.random.shuffle(indices)
            sample = indices[iteration % samples]
            x = np.append(images[sample], 1) # (785,)
            XV = np.append(np.dot(x, V.T), 1) # (201,)
            h = self.tanh(XV) # (201,)
            z = self.sigmoid(np.dot(h,W.T)) # (10,)
            delta = np.multiply((z - vec_labels[sample]),self.sigmoid_prime(np.dot(h,W.T)))
# (10,)
            dW = np.dot(delta.reshape((10,1)),h.reshape((1,201))) # (10,201)
            dV_first_half = np.multiply(np.dot(delta,W), self.tanh_prime(XV))[:-1]
            dV = np.dot(dV_first_half.reshape((200,1)),x.reshape(1,785))
            W = W - self.lamb*dW
            V = V - self.lamb*dV
            iteration += 1
            # if iteration % 1000 == 0:
            #     yield V,W
            if iteration % 1000 == 0:
                self.lamb = self.lamb*self.decay
                # Check gradients
                # img = images[sample].reshape((1,784))
                # eta = 1e-6
                # check_dW =
(self.squaredError(self.predict((V,W+eta),img),vec_labels[sample])
                #            - self.squaredError(self.predict((V,W-
eta),img),vec_labels[sample]))/(2*eta)
                # print('dW')
                # print(np.sum(dW))
                # print(check_dW)
                # check_dV =
(self.squaredError(self.predict((V+eta,W),img),vec_labels[sample])
                #            - self.squaredError(self.predict((V-
eta,W),img),vec_labels[sample]))/(2*eta)
```

```
            # print('dV')
            # print(np.sum(dV))
            # print(check_dV)
            # pdb.set_trace()

            pickle.dump((V,W), open("save.p", "wb"))
            yield V,W
    return V,W

def crossEntropyTrain(self,images,vec_labels):
    if self.load:
        V,W = pickle.load(open( "best.p", "rb" ))
    else:
        V = np.random.normal(0,0.01,(self.hiddenLayerSize, self.inputLayerSize + 1))
        W = np.random.normal(0,0.01,(self.outputLayerSize, self.hiddenLayerSize + 1))
    iteration = 0
    samples = images.shape[0]
    indices = np.arange(samples)
    while True: #error > self.error_limit:
        if iteration % samples == 0:
            np.random.shuffle(indices)
        sample = indices[iteration % samples]
        x = np.append(images[sample], 1) # (785,)
        XV = np.append(np.dot(x, V.T), 1) # (201,)
        h = self.tanh(XV) # (201,)
        z = self.sigmoid(np.dot(h,W.T)) # (10,)

        if 1 in z or 0 in z:
            pdb.set_trace()

        dJdz = np.divide((z-vec_labels[sample]), np.multiply(z,(1-z)))
        delta = np.multiply(dJdz,self.sigmoid_prime(np.dot(h,W.T))) # (10,)
        dW = np.dot(delta.reshape((10,1)),h.reshape((1,201))) # (10,201)
        dV_first_half = np.multiply(np.dot(delta,W), self.tanh_prime(XV))[:-1]
        dV = np.dot(dV_first_half.reshape((200,1)),x.reshape(1,785))
        W = W - self.lamb*dW
        V = V - self.lamb*dV
        iteration += 1
        if iteration % 1000 == 0:
            yield V,W
        if iteration % 50000 == 0:
            self.lamb = self.lamb*self.decay
            # Check gradients
            # img = images[sample].reshape((1,784))
            # eta = 1e-6
```

```
            # check_dW =
(self.crossEntropyError(self.predict((V,W+eta),img),vec_labels[sample])
            #           - self.crossEntropyError(self.predict((V,W-
eta),img),vec_labels[sample]))/(2*eta)
            # if check_dW == 0:
            #     print(self.predict((V,W+eta),img))
            # print('dW')
            # print(np.sum(dW))
            # print(check_dW)
            # check_dV =
(self.crossEntropyError(self.predict((V+eta,W),img),vec_labels[sample])
            #           - self.crossEntropyError(self.predict((V-
eta,W),img),vec_labels[sample]))/(2*eta)
            # print('dV')
            # print(np.sum(dV))
            # print(check_dV)
            # pdb.set_trace()
            pickle.dump((V,W), open("save.p", "wb"))
            yield V,W
        return V,W


    def tanh(self, z):
        return np.tanh(z)

    def tanh_prime(self,z):
        return 1 - np.tanh(z)**2

    def predict(self,weights, images):
        images = np.hstack((images, np.ones((images.shape[0],1))))
        h = self.tanh(np.dot(images,weights[0].T))
        h = np.hstack((h, np.ones((h.shape[0],1))))
        z = self.sigmoid(np.dot(h,weights[1].T))
        return z
        # compute labels of all images using the weights
        # return labels

    def sigmoid(self,z):
        # def sig(x):
        #     if x >= 0:
        #         z = np.exp(-x)
        #         return 1 / (1 + z)
        #     else:
        #         z = np.exp(x)
        #         return z / (1 + z)
        # f = np.vectorize(sig,otypes=[np.float64])
```

```
        return np.clip(1/(1+np.exp(-z)), 0.01, 0.99)
        # return 1/(1+np.exp(-z))

    def sigmoid_prime(self,z):
        return np.multiply(self.sigmoid(z), 1 - self.sigmoid(z))

    def squaredError(self,predictions, vec_labels):
        return np.sum((predictions-vec_labels)**2)/2

    def crossEntropyError(self,predictions, vec_labels):
        # a = np.log(predictions)
        # b = np.log(1 - predictions)
        error = -np.sum(np.multiply(vec_labels,np.log(predictions)) +
                np.multiply((1 - vec_labels), np.log(1 - predictions)))
        if np.isnan(error):
            print('Taking log of zero...')
        return error

train = scipy.io.loadmat("dataset/train.mat")
train_images= train["train_images"]
train_labels= train["train_labels"]
train_images = np.float64(train_images.reshape(-1, train_images.shape[-1]))
# train_images = sklearn.preprocessing.normalize(train_images,axis=0)
train_images = train_images/255
train_data = np.hstack((train_images.T, train_labels))
np.random.shuffle(train_data)
train_images = train_data[:50000,:-1]
train_labels = train_data[:50000,-1]
valid_images = train_data[50000:,:-1]
valid_labels = train_data[50000:,-1]



def vectorize_labels(labels):
    new_label = np.zeros((labels.shape[0],10))
    for i in range(new_label.shape[0]):
        new_label[i][labels[i]] = 1
    return new_label

vec_labels = vectorize_labels(train_labels)

# Mean squared Error
#--------------------------------------------------------------------------------
# net = neural_net(lamb = 1e-1, decay = 0.9, load = True)
# gen = net.squaredErrorTrain(train_images,vec_labels)
# error = 1
```

```
# while error > 0.02:
#     V,W = gen.__next__()
#     pred = net.predict((V,W),valid_images)
#     error = benchmark(np.argmax(pred, axis=1),valid_labels)[0]
#     print('valid error')
#     print(error)
#     train_pred = net.predict((V,W),train_images)
#     train_error = benchmark(np.argmax(train_pred, axis=1),train_labels)[0]
#     print('train error')
#     print(train_error)
#--------------------------------------------------------------------------------


# Cross Entropy Error
#--------------------------------------------------------------------------------
# net = neural_net(lamb = 1e-2,decay = 0.7,load= True)
# gen = net.crossEntropyTrain(train_images,vec_labels)
# error = 1
# while error > 0.02:
#     V,W = gen.__next__()
#     pred = net.predict((V,W),valid_images)
#     error = benchmark(np.argmax(pred, axis=1),valid_labels)[0]
#     print('valid acc')
#     print(1-error)
#     train_pred = net.predict((V,W),train_images)
#     train_error = benchmark(np.argmax(train_pred, axis=1),train_labels)[0]
#     print('train acc')
#     print(1-train_error)


# Show images
# img = montage_images(train_images.T.reshape(28,28,50000))
# plt.imshow(img)
# plt.show()
#--------------------------------------------------------------------------------


# Test data
#--------------------------------------------------------------------------------
# V,W = pickle.load(open( "best.p", "rb" ))

# test = scipy.io.loadmat("dataset/test.mat")
# test_images = test["test_images"]
# test_images = np.float64(test_images.reshape(test_images.shape[0], -1))
# test_images = test_images/255
# # test_images = sklearn.preprocessing.normalize(test_images,axis=1)
```

```
# net = neural_net()
# pred = np.argmax(net.predict((V,W),test_images), axis=1)

# numbers = np.arange(len(pred)) + 1
# test_predict = np.vstack((numbers,pred))
# np.savetxt("digits.csv", test_predict.transpose(), delimiter=",",fmt = '%u')

# Show images
# img = montage_images(test_images.T.reshape(28,28,10000))
# plt.imshow(img)
# plt.show()
#------------------------------------------------------------------------------




# Plots
#------------------------------------------------------------------------------
# net = neural_net(lamb = 1e-1, decay = 0.9)
# gen = net.squaredErrorTrain(train_images,vec_labels)
# errors = []
# accuracies = []
# while len(errors) < 50:
#     V,W = gen.__next__()
#     pred = net.predict((V,W),train_images)
#     accuracies.append(1 - benchmark(np.argmax(pred, axis=1),train_labels)[0])
#     errors.append(net.squaredError(pred,vec_labels))

# iterations = list(range(1000,51000,1000))
# plt.figure()
# plt.plot(iterations,errors)
# plt.figure()
# plt.plot(iterations,accuracies)
# plt.show()

net = neural_net(lamb = 1e-2, decay = 0.7)
gen = net.crossEntropyTrain(train_images,vec_labels)
errors = []
accuracies = []
while len(errors) < 50:
    V,W = gen.__next__()
    pred = net.predict((V,W),train_images)
    accuracies.append(1 - benchmark(np.argmax(pred, axis=1),train_labels)[0])
    errors.append(net.crossEntropyError(pred,vec_labels))
    print(len(errors))
```

```
iterations = list(range(1000,51000,1000))
plt.figure()
plt.plot(iterations,errors)
plt.figure()
plt.plot(iterations,accuracies)
plt.show()
#-----------------------------------------------------------------------------------
```