Jesse Li

SID: 23822462

CS189

HW6 Writeup



Wk is a row of W, *hl* is the output of hidden unit *l*, *Vl* is a row of V.



Mean squared error:



Cross-entropy error:

Wij = Wij – λ(dJ/dWij)

Vij = Vij – λ(dJ/dVij)

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

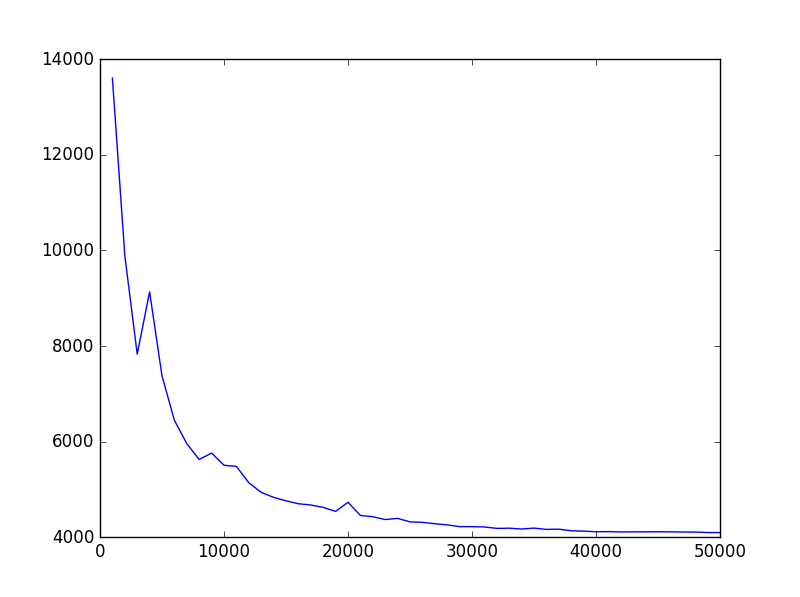
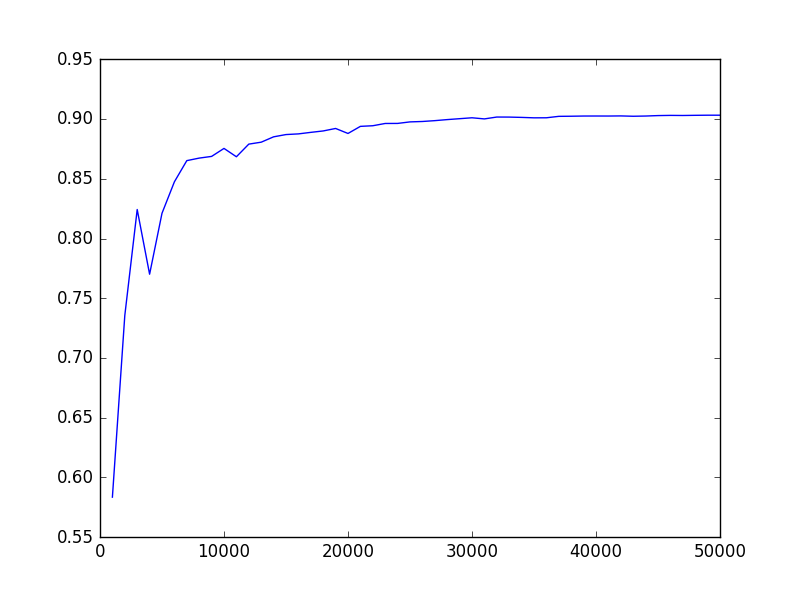
Best mean-squared error validation accuracy = 0.98

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.

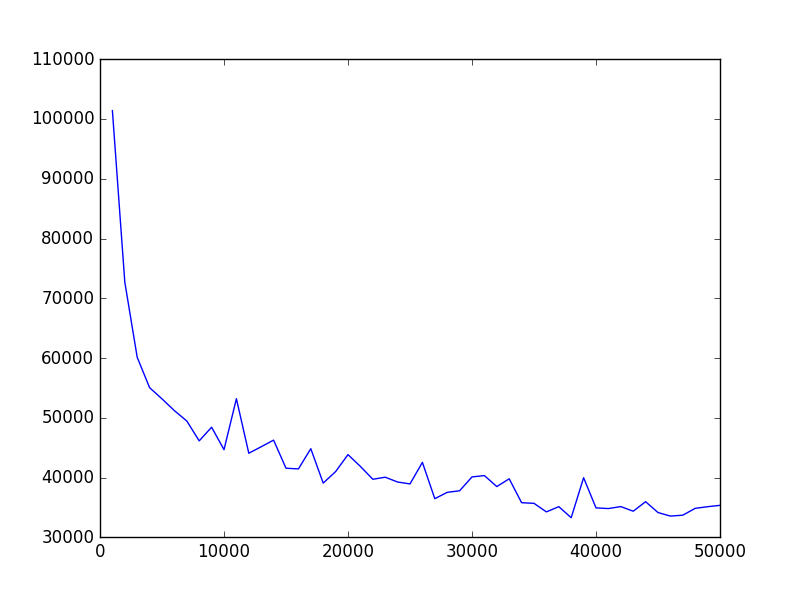
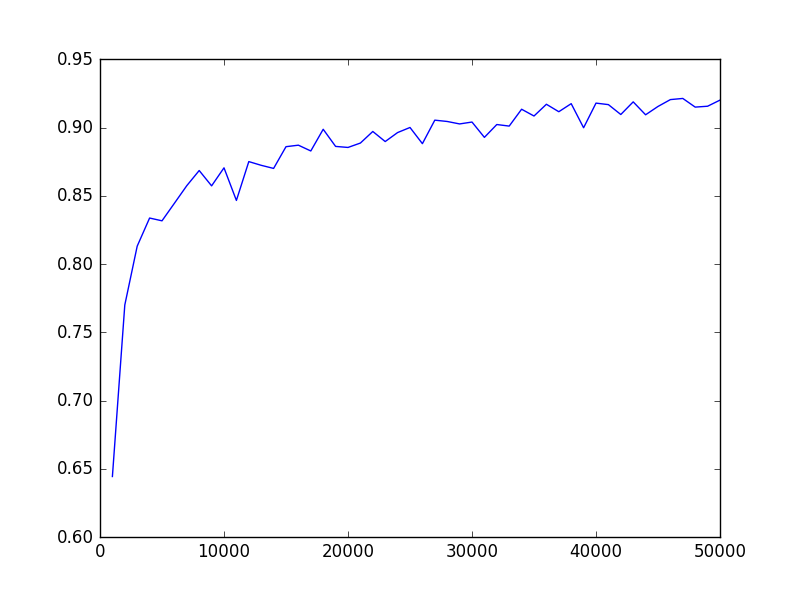


Mean squared error

MSE accuracy

Iteration

Iteration



Cross entropy error

Cross entropy error accuracy

Iteration

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()

#----------------------------------------------------------------------------------