April 6, 2018

Vibhavi Peiris

Student #: 1000597687

CSC412

Assignment 3

1. Problem 1: L2-Regularized Logistic Regression
   1. *Use code from A2 with 300 training points*

#A3 Q1A From my-A2 Q3C code

def one\_per\_class(images, labels):

out\_images = np.zeros((10,images.shape[1]))

out\_labels = np.zeros((10,10))

classes = np.where(labels == 1)[1] # get the class digit for each image by getting column idx of ones in labels

#get first image in training set with each class label

for i in range(0,10):

img\_num = np.where(classes == i)[0][0]

out\_images[i,:] = images[img\_num,:]

out\_labels[i,:] = labels[img\_num,:]

return out\_images,out\_labels

def cost\_function(w):

sum\_final = 0 #temporary create sum\_final var

dem = logsumexp(np.dot(np.transpose(w),grad\_images))

#mutliclass likelihood function is sum from 0 to k of label\*predictive\_log\_likelihood

for k in range(0,10):

log\_pc\_x = np.dot(np.transpose(w[:,k]),grad\_images) - dem

if k == 0:

sum\_final = np.dot(grad\_labels[k],log\_pc\_x)

else:

sum\_final = sum\_final + np.dot(grad\_labels[k],log\_pc\_x)

return sum\_final

def logistic\_gradient\_desc(iterations,lr):

#set globals so that cost function can access these values after usign autograd w.r.t. w

global current\_c

global grad\_images

global grad\_labels

w = np.zeros((784,10)) #create the weights

for i in range(0,iterations):

for img\_num in range(0,new\_images.shape[0]):

#get gradient of cost function/likelihood

grad\_images = new\_images[img\_num,:] #get current image

grad\_labels = new\_labels[img\_num,:] #get labels for current image

current\_c = img\_num #sinces we sampled 1 image for each class in order c = img\_num

cost\_grad = elementwise\_grad(cost\_function)

#update weights

w = w + lr\*cost\_grad(w)

print(i)

return w

#run A2 Q3C code

new\_images = train\_images

new\_labels = train\_labels

grad\_images=grad\_labels = new\_images #temporary just to create a gobal var for use with autograd

current\_c = 0 #temporary just to create a gobal var for use with autograd

weights = logistic\_gradient\_desc(1000, 0.01) #5000 iterations with a common learning rate of 0.01

save\_images(np.transpose(weights),'Q1a')

#A2 Q3d code

def avg\_pred\_log(w,images):

log\_pc\_x = 0

for i in range(0,images.shape[0]):

current\_log\_pc\_x = np.dot(np.transpose(w),images[i,:]) - logsumexp(np.dot(np.transpose(w),images[i,:]))

log\_pc\_x = log\_pc\_x + current\_log\_pc\_x

return np.sum(log\_pc\_x)/float(images.shape[0])

def predict\_regression(images, w):

predictions = np.zeros((images.shape[0],w.shape[1])) #N by 10

#find best class true class for each image

for i in range(0,images.shape[0]):

best\_class = np.argmax(np.dot(np.transpose(w),images[i,:])) #choose the class with highest

predictions[i,best\_class] = 1 #set index = digit to 1 as it is the best prediction for current image

return predictions

def Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,w):

#average for train

avg\_likelihood\_train = avg\_pred\_log(w,train\_images)

print("Avg train log likelihood ",avg\_likelihood\_train)

#average for test

avg\_likelihood\_test = avg\_pred\_log(w,test\_images)

print("Avg test log likelihood ",avg\_likelihood\_test)

#predictions for train

predict\_train = predict\_regression(train\_images, w)

total\_correct\_train = np.sum(np.nonzero(predict\_train)[1] == np.nonzero(train\_labels)[1]) #get total number of correct predictions

accuracy\_train = total\_correct\_train/float(train\_labels.shape[0]) #get accuracy

print('Train Accuracy ',accuracy\_train)

#predictions for test

predict\_test = predict\_regression(test\_images, w)

total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

accuracy\_test = total\_correct\_test/float(test\_labels.shape[0]) #get accuracy

print('Test Accuracy ',accuracy\_test)

#run A2 Q3D code

#Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,weights)



#A2 Q3d code

def avg\_pred\_log(w,images):

log\_pc\_x = 0

for i in range(0,images.shape[0]):

current\_log\_pc\_x = np.dot(np.transpose(w),images[i,:]) - logsumexp(np.dot(np.transpose(w),images[i,:]))

log\_pc\_x = log\_pc\_x + current\_log\_pc\_x

return np.sum(log\_pc\_x)/float(images.shape[0])

def predict\_regression(images, w):

predictions = np.zeros((images.shape[0],w.shape[1])) #N by 10

#find best class true class for each image

for i in range(0,images.shape[0]):

best\_class = np.argmax(np.dot(np.transpose(w),images[i,:])) #choose the class with highest

predictions[i,best\_class] = 1 #set index = digit to 1 as it is the best prediction for current image

return predictions

def Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,w):

#average for train

avg\_likelihood\_train = avg\_pred\_log(w,train\_images)

print("Avg train log likelihood ",avg\_likelihood\_train)

#average for test

avg\_likelihood\_test = avg\_pred\_log(w,test\_images)

print("Avg test log likelihood ",avg\_likelihood\_test)

#predictions for train

predict\_train = predict\_regression(train\_images, w)

total\_correct\_train = np.sum(np.nonzero(predict\_train)[1] == np.nonzero(train\_labels)[1]) #get total number of correct predictions

accuracy\_train = total\_correct\_train/float(train\_labels.shape[0]) #get accuracy

print('Train Accuracy ',accuracy\_train)

#predictions for test

predict\_test = predict\_regression(test\_images, w)

total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

accuracy\_test = total\_correct\_test/float(test\_labels.shape[0]) #get accuracy

print('Test Accuracy ',accuracy\_test)

#run A2 Q3D code

Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,weights) #REPORT FOR A3 Q1a

*Average train log likelihood -117.989022988*

*Average test log likelihood -97.3003729208*

*Train Accuracy 1.0*

*Test Accuracy 0.7727*

* 1. *MAP Estimator*







* 1. *Fit map*

#A3 Q1C

#logitsic regression with gradient descent using map

def grad\_desc(iterations,lr,sigma):

#set globals so that cost function can access these values after usign autograd w.r.t. w

global grad\_images

global grad\_labels

w = np.zeros((784,10)) #create the weights

for i in range(0,iterations):

for img\_num in range(0,new\_images.shape[0]):

#get gradient of cost function/likelihood

grad\_images = new\_images[img\_num,:] #get current image

grad\_labels = new\_labels[img\_num,:] #get labels for current image

cost\_grad = elementwise\_grad(cost\_function)

#update weights

w = w + lr\*cost\_grad(w)

#NEW ADDITION FOR A3

w = w - w/sigma\*\*2

print(i)

return w

#run cod for A3 q1c

print("Map logitsic regression")

##Testing for best sigma value was 36

#for i in range(1,10):

#sigma = i\*\*2 # from 5 to 100

#print(sigma)

#map\_weights = grad\_desc(100, 0.01,sigma) #5000 iterations with a common learning rate of 0.01

#save\_images(np.transpose(map\_weights),'Q1c')

#Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,map\_weights)

#sigma = 1/i\*\*2 #from 1 to 1/100

#print(sigma)

#map\_weights = grad\_desc(100, 0.01,sigma) #5000 iterations with a common learning rate of 0.01

#save\_images(np.transpose(map\_weights),'Q1c')

#Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,map\_weights)

sigma = 36

map\_weights = grad\_desc(1000, 0.01,sigma) #5000 iterations with a common learning rate of 0.01

save\_images(np.transpose(map\_weights),'Q1c')

Q3D\_report(train\_images,train\_labels,test\_images,test\_labels,map\_weights)

*Average train log likelihood -93.0039030291*

*Average test log likelihood -77.2325259507*

*Train Accuracy 1.0*

*Test Accuracy 0.7711*

*Best sigma value was 36. With 4,9,16,25 close behind.*



1. Problem 2: Bayesian Logistic Regression using Stochastic Variational Inference
   1. *Number of parameters?*

For w it is 784 \* 10 = 7840 parameters.

For ф its (mean + standard deviation) =7840+7840 = 14960 parameters.

* 1. *Code SVI.*

def elbo\_estimate(var\_params, logprob, num\_samples, rs):

"""Provides a stochastic estimate of the variational lower bound.

var\_params is (mean, log\_std) of a Gaussian."""

mean, log\_std = var\_params

samples = sample\_diag\_gaussian(mean,log\_std,num\_samples,rs)

log\_ps = logprob(samples)

log\_qs = diag\_gaussian\_log\_density(samples,mean,log\_std)

E\_q = np.sum(log\_ps-log\_qs)/num\_samples # E\_q(z|x)[log p(x,z) - log q(z|x)]

return E\_q

def logprob\_given\_data(params):

data\_logprob = logistic\_logprob(params,train\_images,train\_labels)

prior\_logprob = np.sum(np.sum(-np.log(np.sqrt(2\*np.pi\*prior\_std))-(params\*\*2)/(2\*prior\_std),axis=2),axis=1)

return data\_logprob + prior\_logprob

* 1. Compute accuracy for test

predict\_test = predict\_regression(test\_images, np.transpose(optimized\_params[0])) #A3 q1/A2 code

total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

accuracy\_test = total\_correct\_test/float(test\_labels.shape[0]) #get accuracy

print('\nTest Accuracy ',accuracy\_test)

##testing a bunch of std values, std = 1 is best with 77.68% and std =9 is second best 77.09%

#for i in range(1,10):

#prior\_std = i\*\*2 # from 5 to 100

#print(prior\_std)

#optimized\_params = adam(objective\_grad, init\_params, step\_size=0.05, num\_iters=100, callback=print\_perf)

#predict\_test = predict\_regression(test\_images, np.transpose(optimized\_params[0])) #A3 q1/A2 code

#total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

#accuracy\_test = total\_correct\_test/float(test\_labels.shape[0]) #get accuracy

#print('\nTest Accuracy ',accuracy\_test)

#prior\_std = 1/i\*\*2 #from 1 to 1/100

#print(prior\_std)

#optimized\_params = adam(objective\_grad, init\_params, step\_size=0.05, num\_iters=100, callback=print\_perf)

#predict\_test = predict\_regression(test\_images, np.transpose(optimized\_params[0])) #A3 q1/A2 code

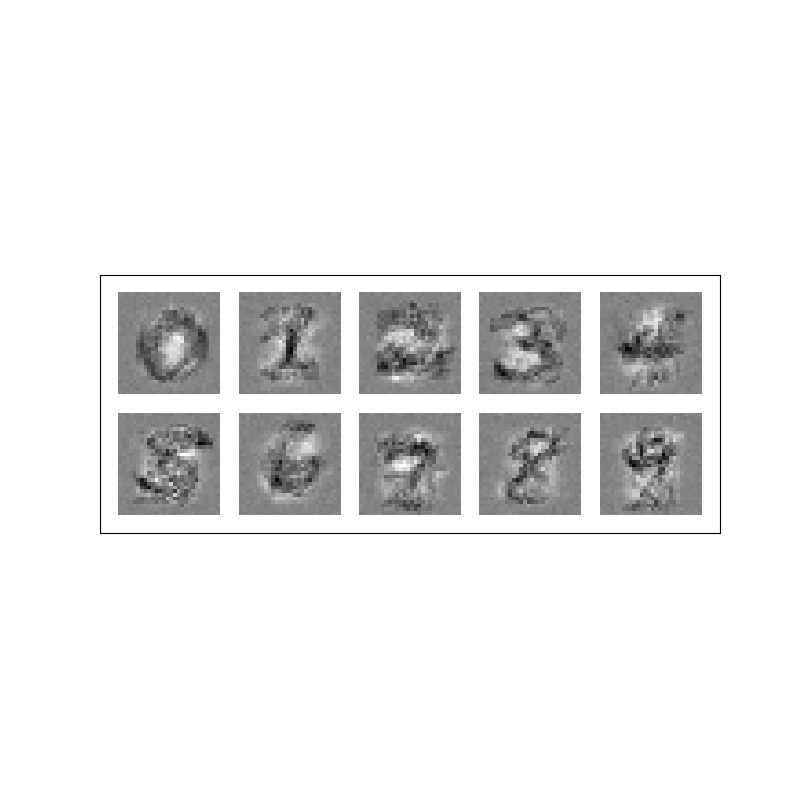
#total\_correct\_test = np.sum(np.nonzero(predict\_test)[1] == np.nonzero(test\_labels)[1]) #get total number of correct predictions

#accuracy\_test = total\_correct\_test/float(test\_labels.shape[0]) #get accuracy

#print('\nTest Accuracy ',accuracy\_test)

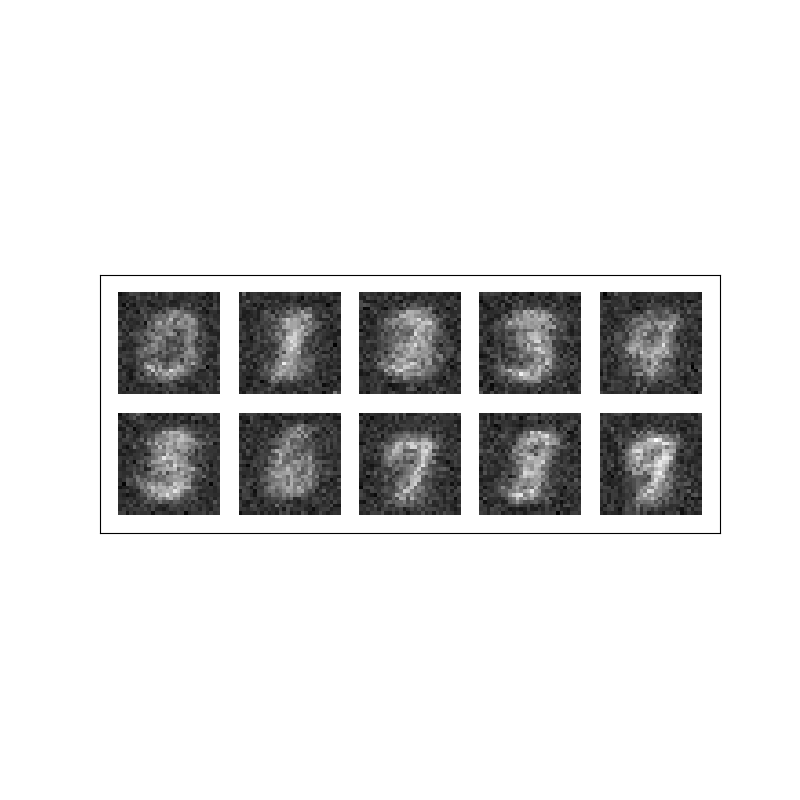
Standard deviation = 1 is the best with 77.68% and Standard deviation = 9 is second best 77.09%. These number are only slightly higher than MAP inferences accuracy.

* 1. *Plot 10 images*
     1. *Mean*



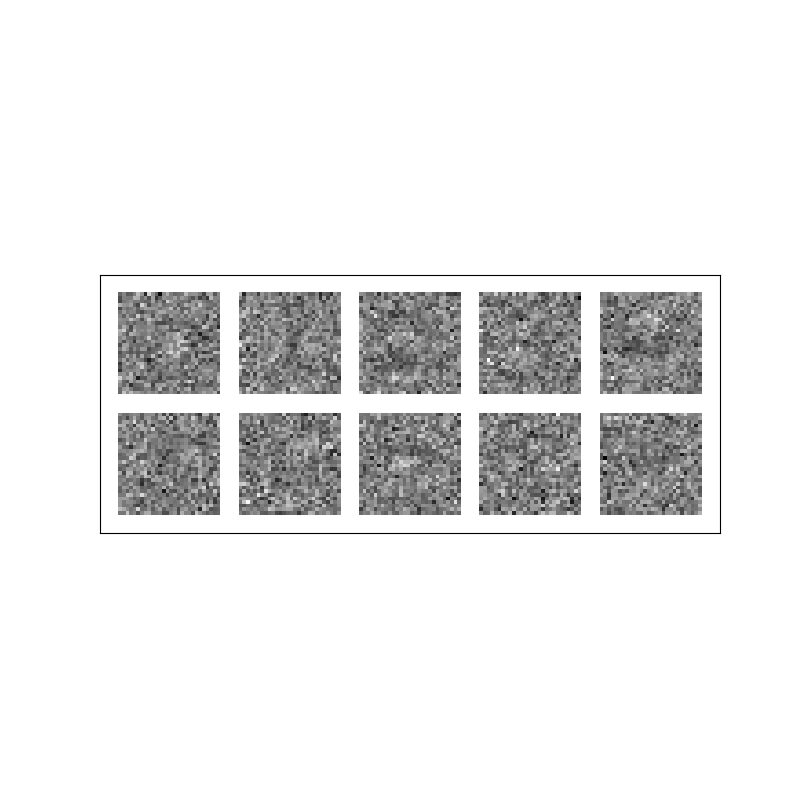
The mean is as I expected because the darkest regions would be the most common pixels of a digit (middle for 1).

* + 1. *Variation*



The variation would have the opposite effect of mean because the further away (up to a certain distance) from the digits mean locations the more variation on pixel values. I.E. for digit 1 the close pixels around the one would have high variation (due to all possible ways one is drawn)

* + 1. *Samples*



The sample is not what I expected, there is more noise. I expected it to be a combination of mean and variance.

* 1. *Single sample q*

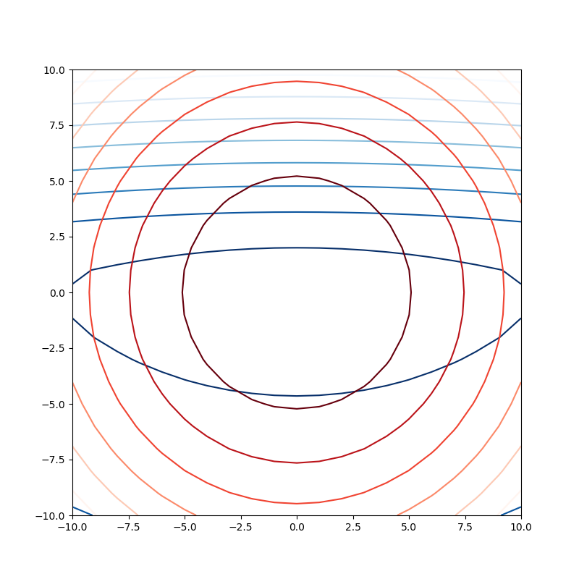
When xdєB that means that xd = 0. So wcdTxd will be 0. Which means it does not affect the optimal because p(t|w,x) is 0 (substitute wcdTxd=0 into the p(t|w,x equation).

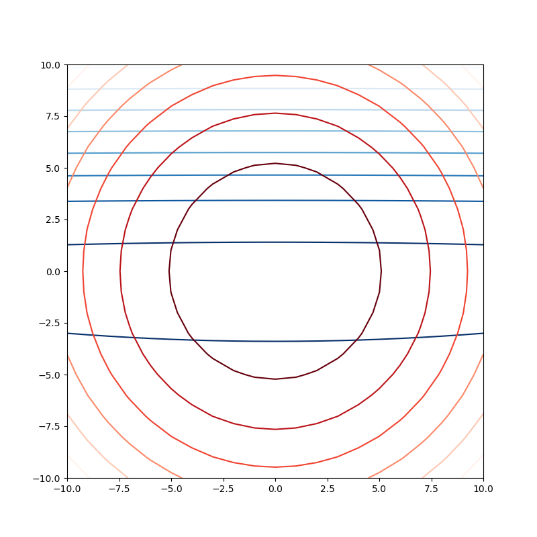
* 1. *Posterior questions*
     1. *Does training affect posterior and variational?*

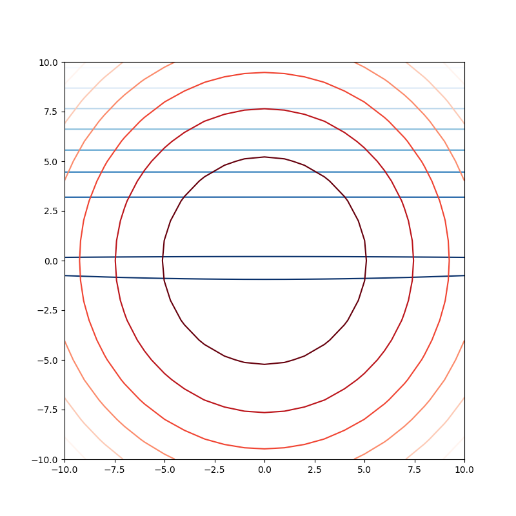
No, true posterior seems to stay the same through the training. Yes, Variational posterior changes.

* + 1. *How does standard deviation affect posterior?*

Increasing standard deviation makes the true posterior wider across the horizontal.

Standard deviation = 1

standard deviation = 10

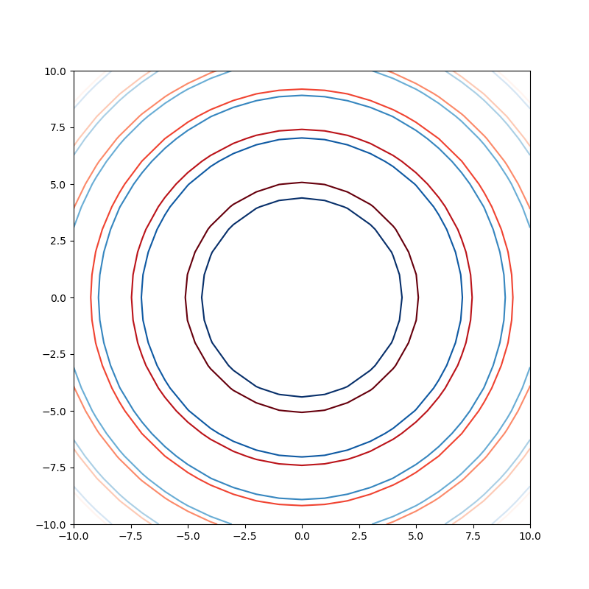
standard deviation = 100

* + 1. *Px2’s affect?*

It makes the true posterior oval shaped & different from variational posterior.

* + 1. *Changing px2’s affect?*

It makes true posterior more circular similar to variational.



* + 1. *Using improper flat model?*

No