**CPEG 586 – DEEP LEARNING**

**HOMEWORK 3**

**Name:** Ivan Sangines Escrig

**ID#:** 968606

**Instructor:** Dr. Mahmood

**Date:** March 1, 2019

**TABLE OF CONTENT**

[**INTRODUCTION**](#_Toc446970371) 3

[**SCREENSHOTS:**](#_Toc446970373) 4

**[SOURCE CODE:](#_Toc446970374)**28

**[CONCLUSION:](#_Toc446970375)**43

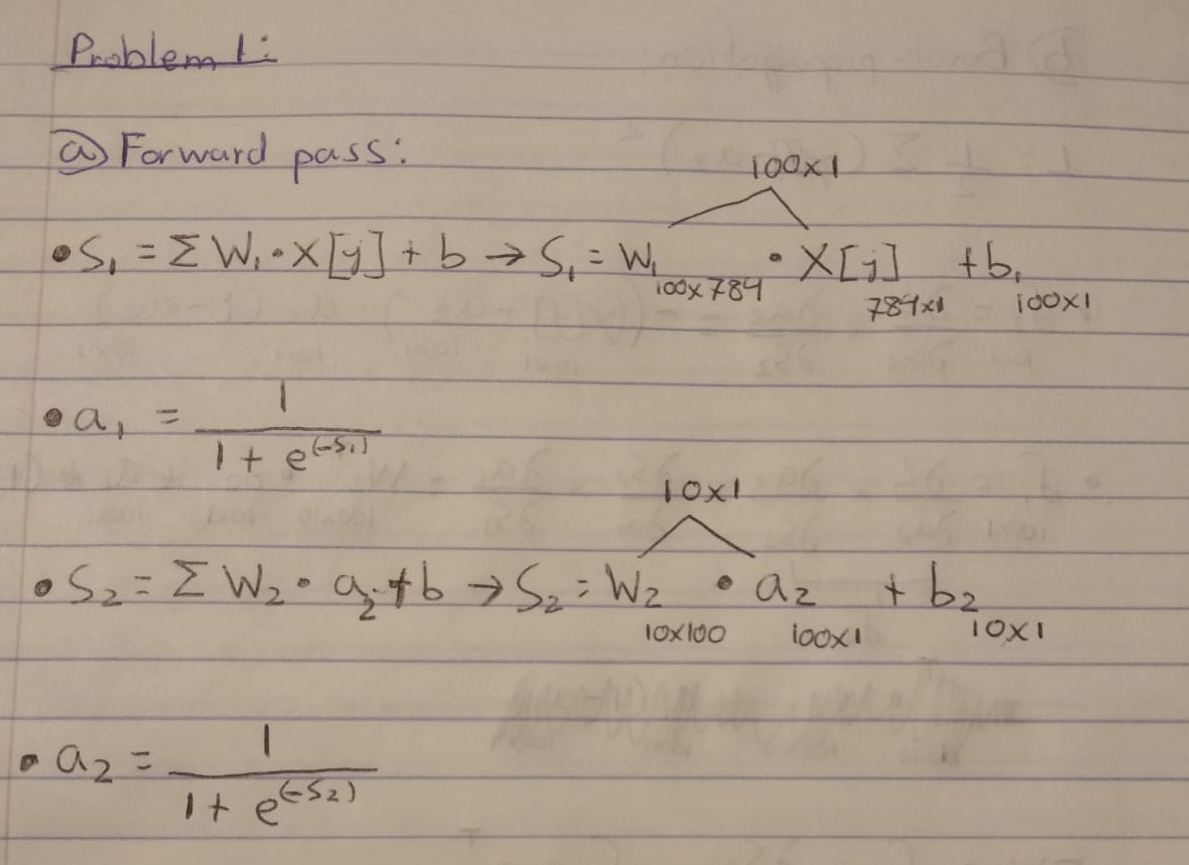
**INTRODUCTION:**

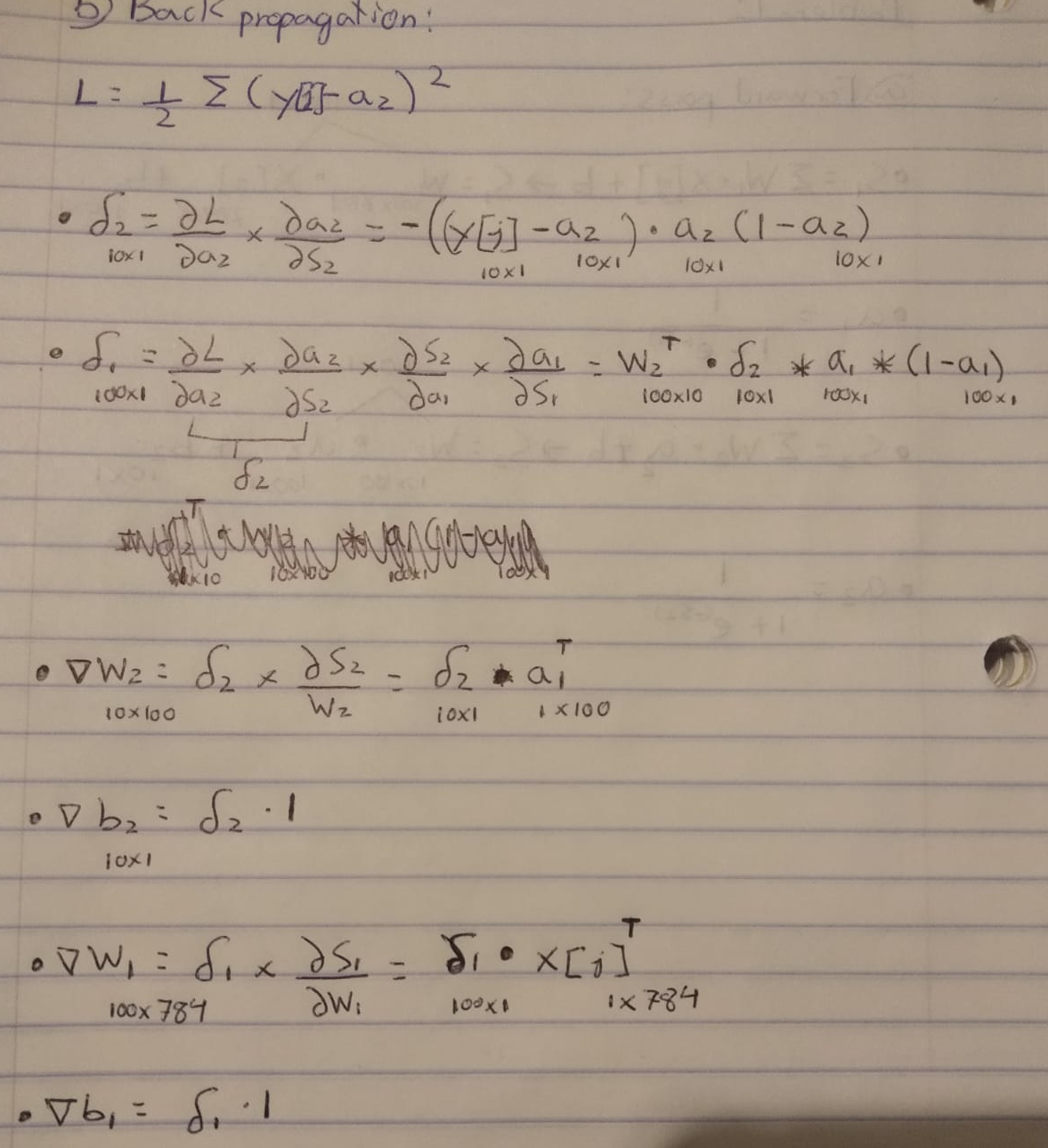
The purpose of this assignment is to understand better all the concepts regarding NN. This assignment will help me understand the relationship between all the components of a NN as well as the importance of setting up a good number of neurons in the hidden layer and number of epochs.

This assignment will also teach me the differences between different activation functions (sigmoid, tanh and RELU) and its effects with the loss function and accuracy.

**PROBLEM 1:**

This computations are done for a NN with 100 neurons in the hidden layer and 10 possible outputs.

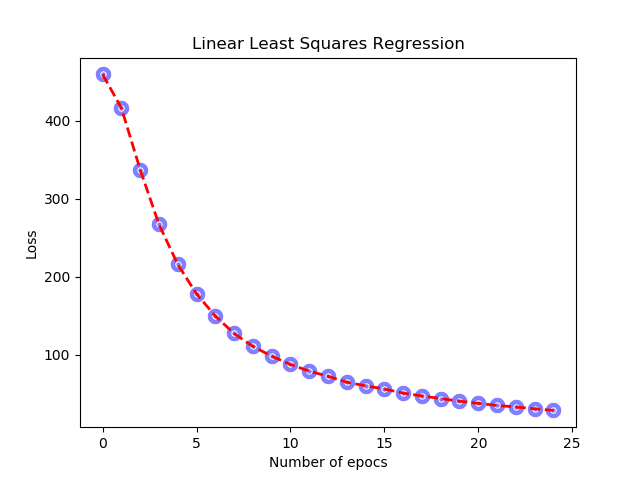




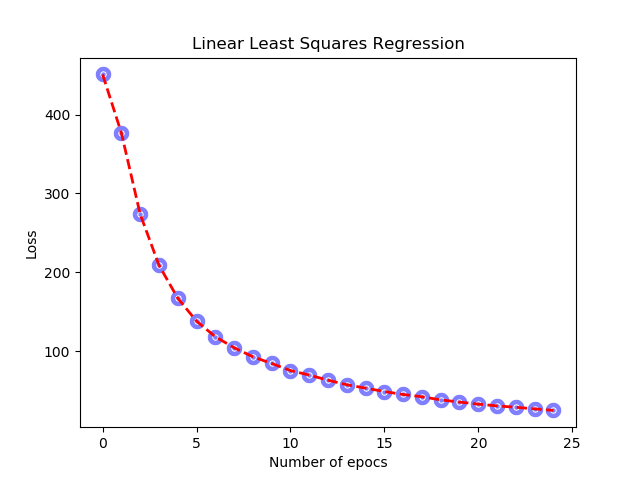
**PROBLEM 2:**

**GRADIENT DESCENT:**

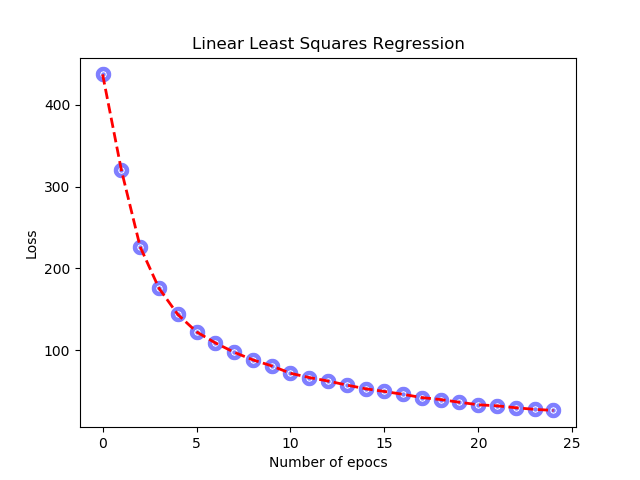
The following screenshots show the graph for the loss function for all the possible combinations of neurons in hidden layer and number of epochs from 25 to 150 in a SGD. As we can see on the following solutions, the best result is obtained on the 100x100 NN where I obtained an error below 3 and an accuracy of 0.9. We are able to reduce the error down to 2-3 however, the accuracy will always fluctuate between 0.88 and 0.90 which translates to 88-90%. This is because we still have things to learn, which will make our NN improve its performance and get even better accuracy.

****

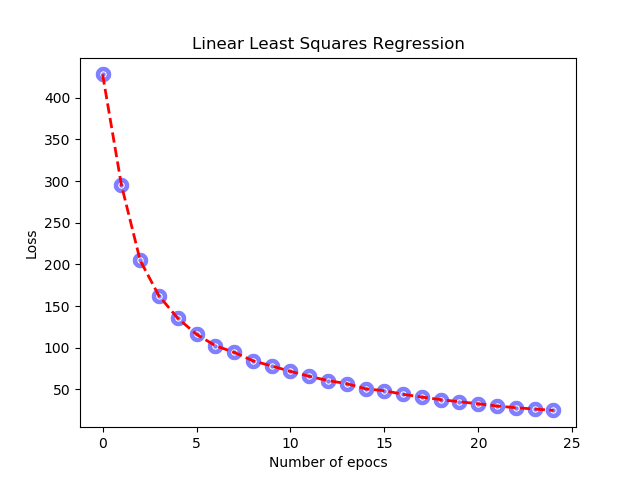
Accuracy for 25X25: **0.8931**



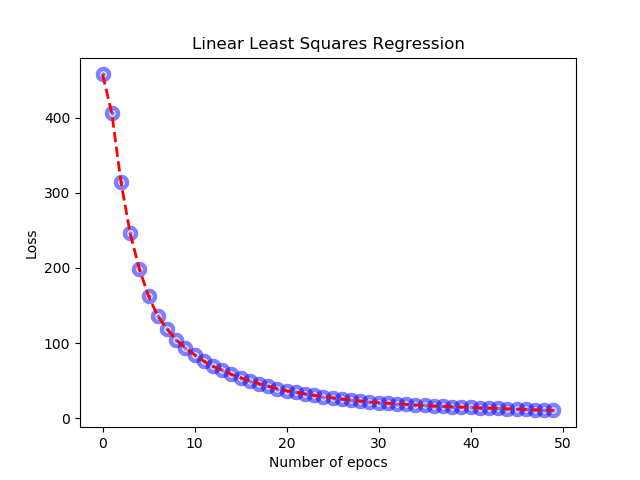
Accuracy for 50X25: **0.8925**



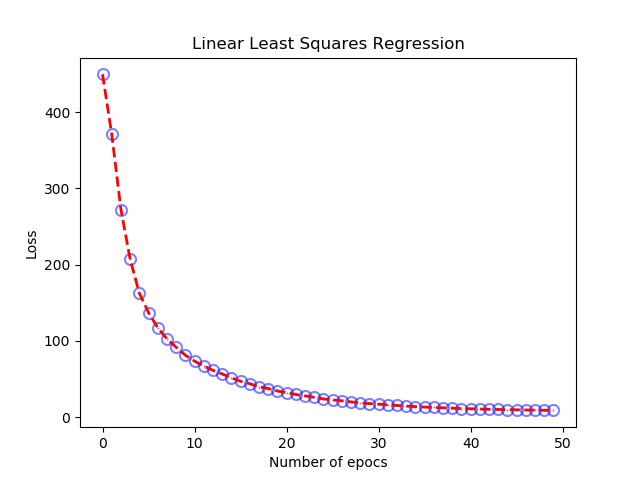
Accuracy for 100X25: **0.89**



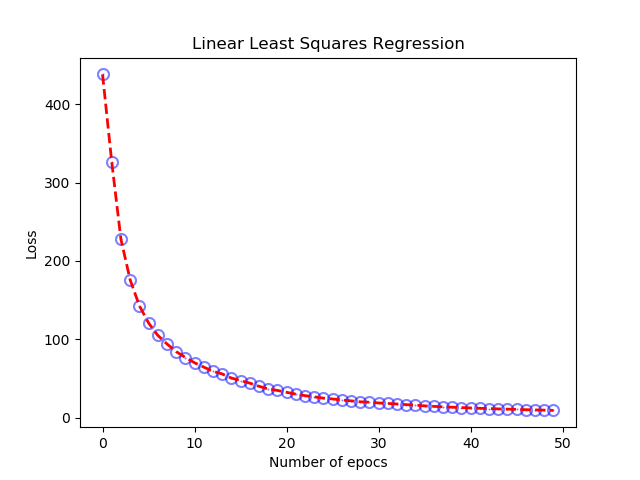
Accuracy for 150X25: **0.8927**



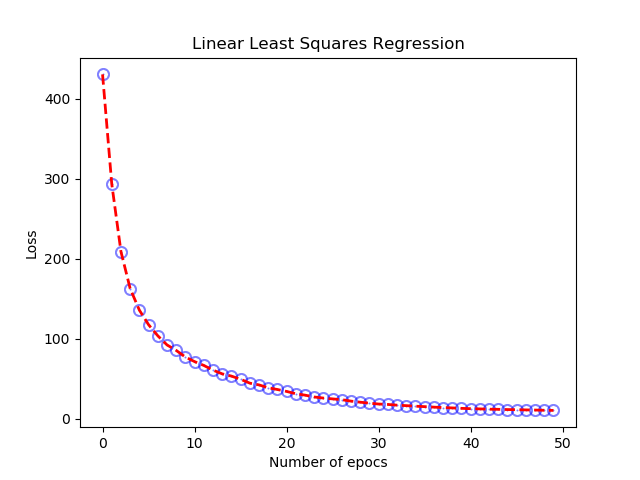
Accuracy for 25x50: **0.8897**



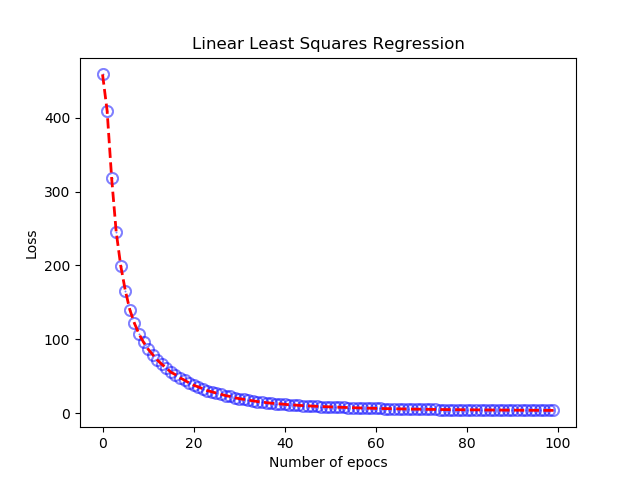
Accuracy for 50x50: **0.8946**



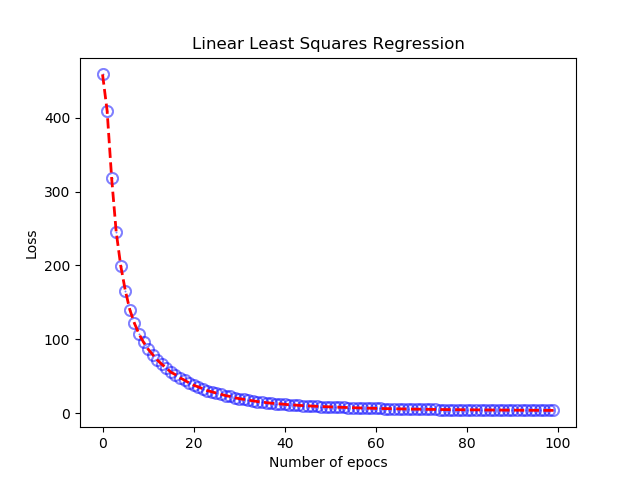
Accuracy for 100x50: **0.8957**



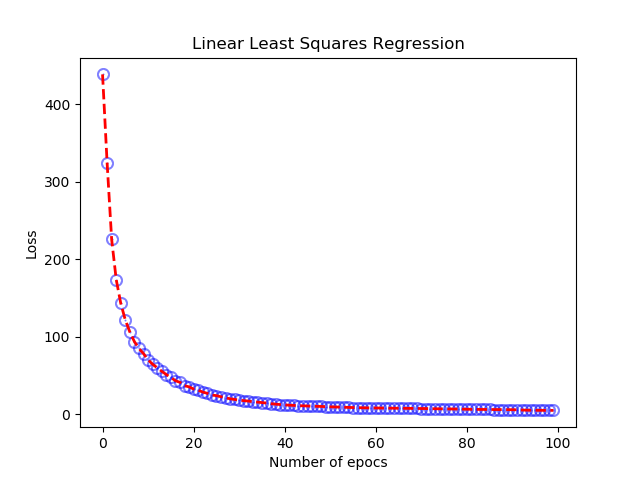
Accuracy for 150x50: **0.8935**



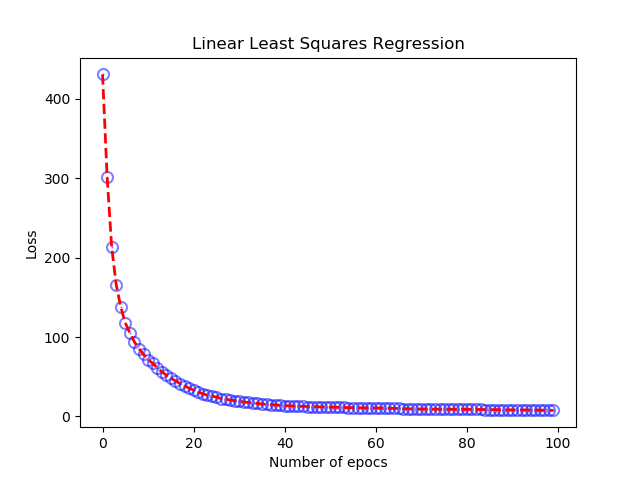
Accuracy for 25x100: **0.8897**



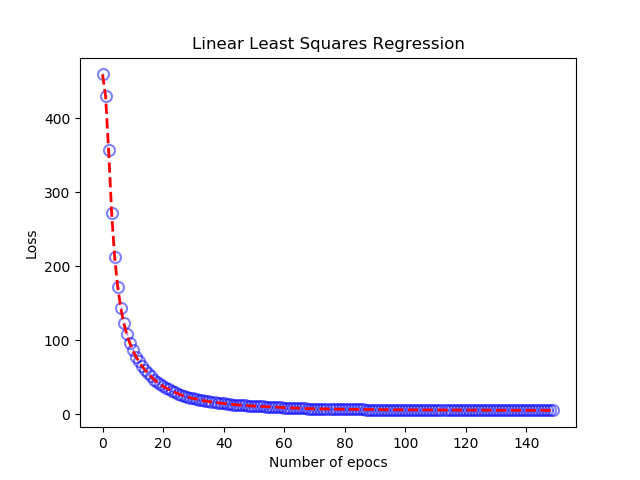
Accuracy for 50x100: **0.8958**



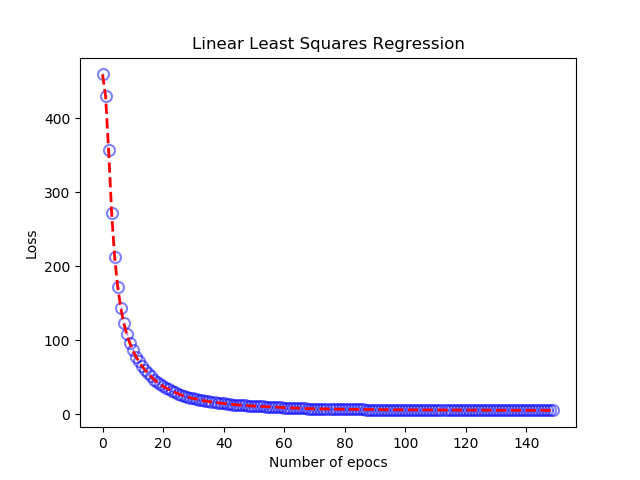
Accuracy for 100x100: **0.9 (best result)**



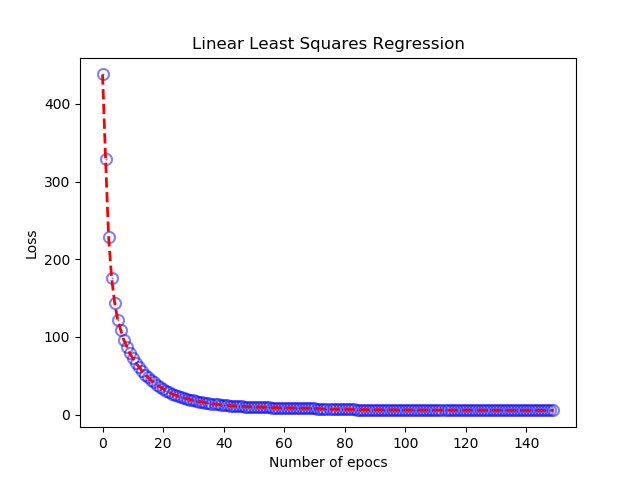
Accuracy for 150x100: **0.8944**



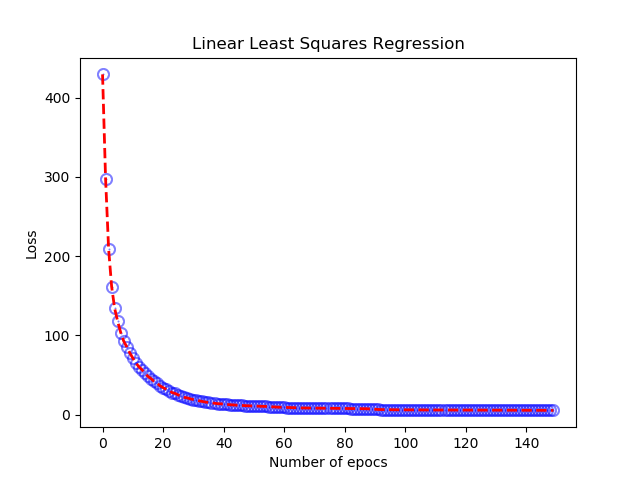
Accuracy for 25x150: **0.8894**



Accuracy for 50x150: **0.8968**

****

Accuracy for 100x150: **0.8951**

****

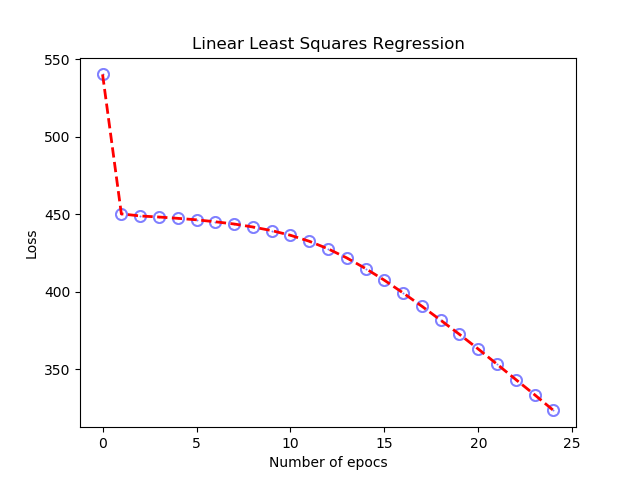
Accuracy for 150x150: **0.896**

**MINI BATCH:**

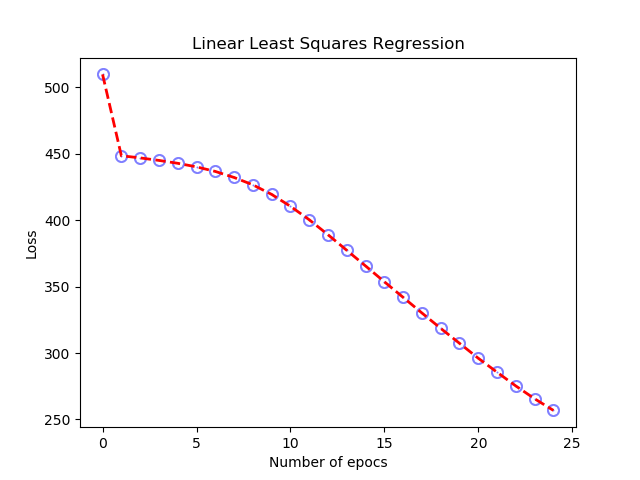
The following screenshots represent the loss function improvement for the different combinations of neurons in the hidden layer and the number of epochs (going from 25 to 150). As we can appreciate in the screenshots, the error is higher than in the SGD since the mini batch descends more gradually (we are taking the average gradient of 10 different training data). However, the accuracy at the end will be almost the same as the SGD fluctuating around 88%.

It is true that at the beginning, for a small number of neurons in the hidden layer as well as a small number of epochs, the error will be really high. However, this is happening because, as stated before, the mini batch is taking average of 10 training data in order to avoid future big jumps.

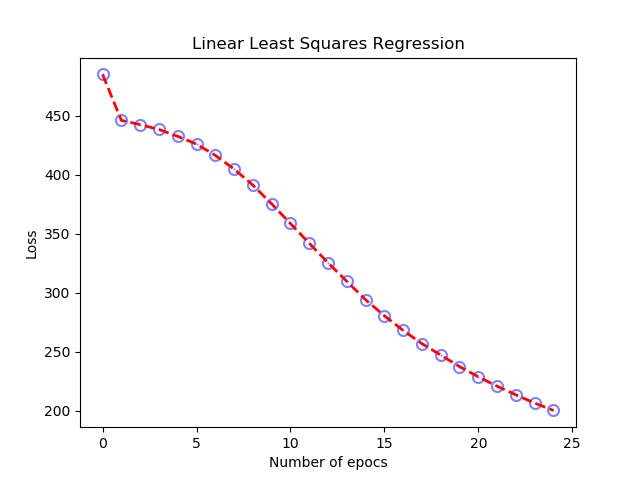
This will make the mini batch better technique than the SGD, it can take longer time to reduce the error to almost 0. However, with the mini batch we will have a more smooth function and better results with the right amount of neurons and epochs

****

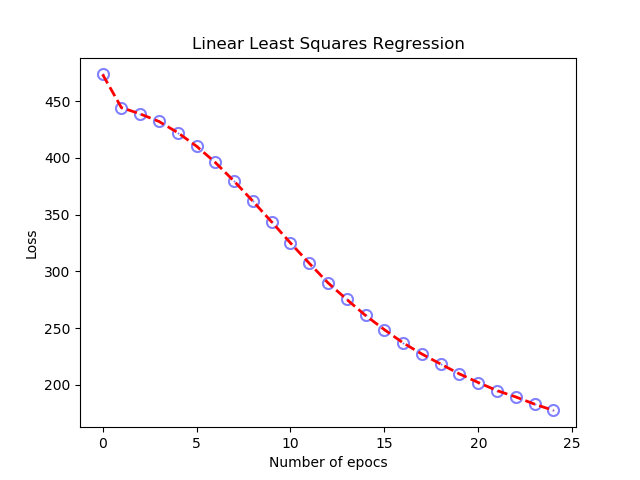
Accuracy for 25X25: **0.6233**



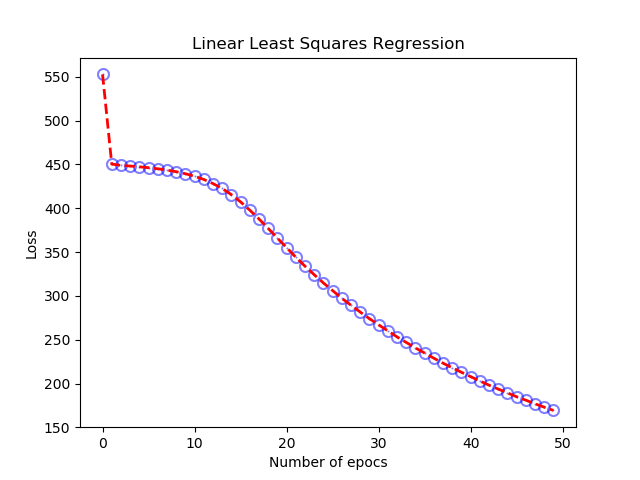
Accuracy for 50X25: **0.7756**



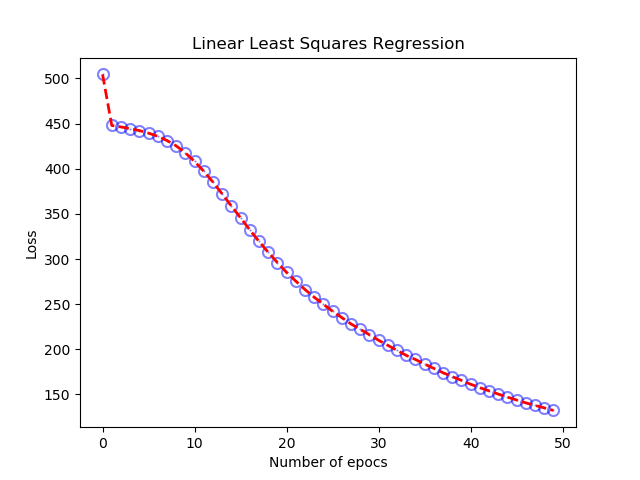
Accuracy for 100X25: **0.8334**



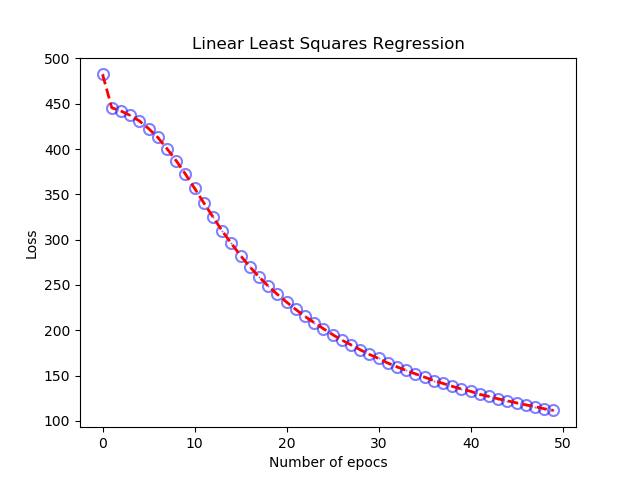
Accuracy for 150X25: **0.8447**



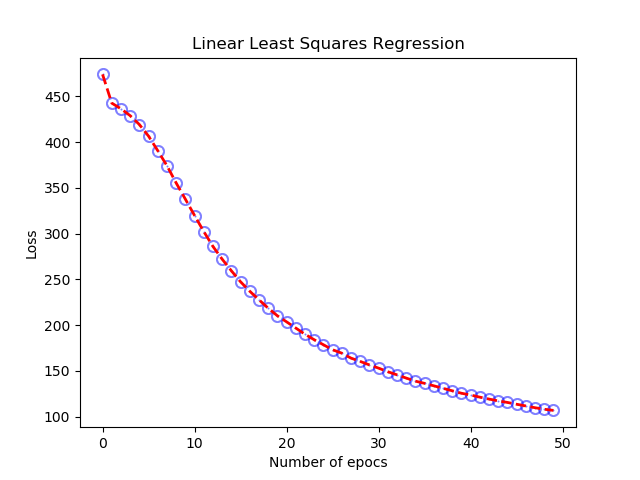
Accuracy for 25x50: **0.8421**



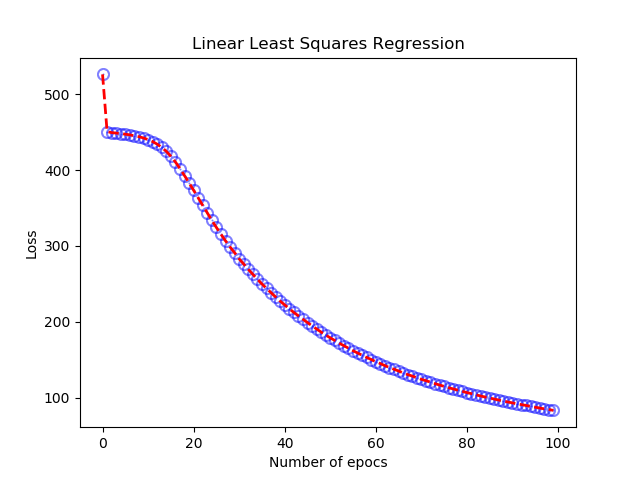
Accuracy for 50x50: **0.8738**



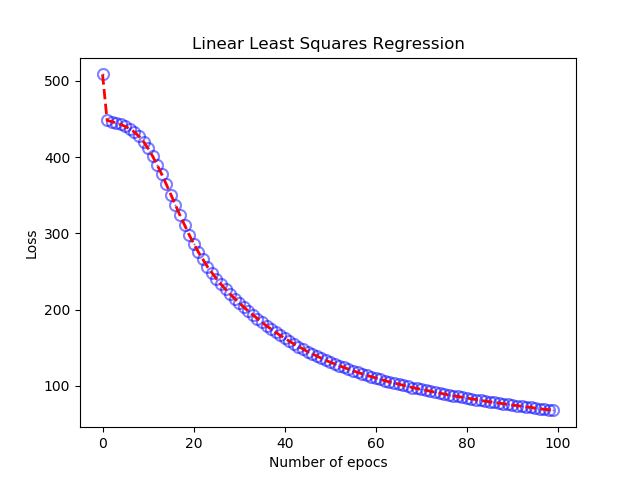
Accuracy for 100x50: **0.8803**



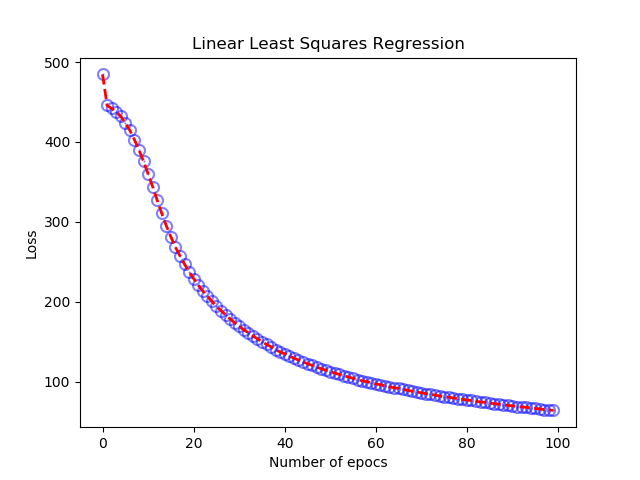
Accuracy for 150x50: **0.8796**



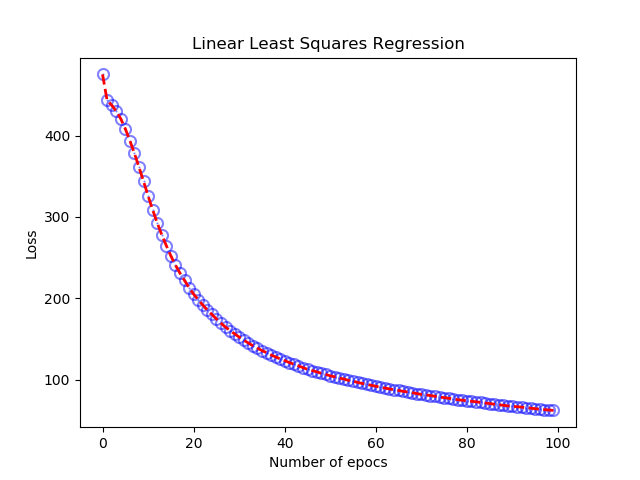
Accuracy for 25x100: **0.8829**



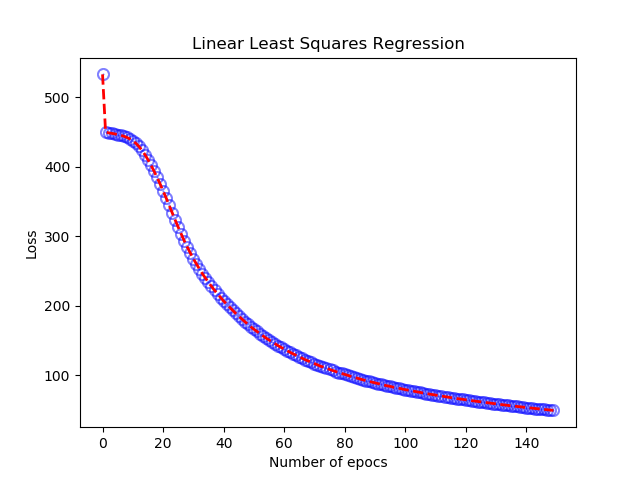
Accuracy for 50x100: **0.8915**



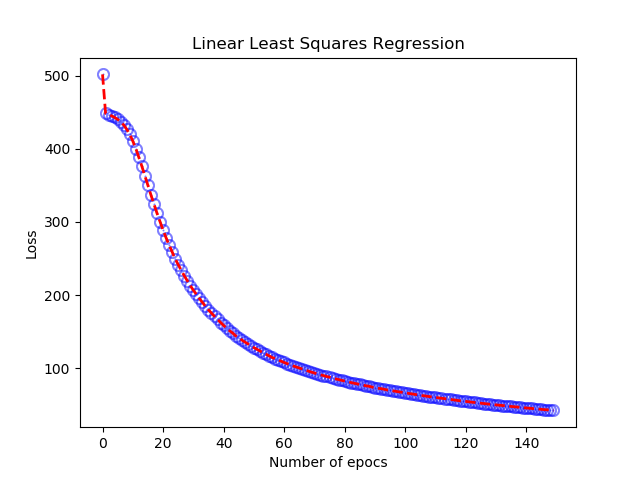
Accuracy for 100x100: **0.8892**



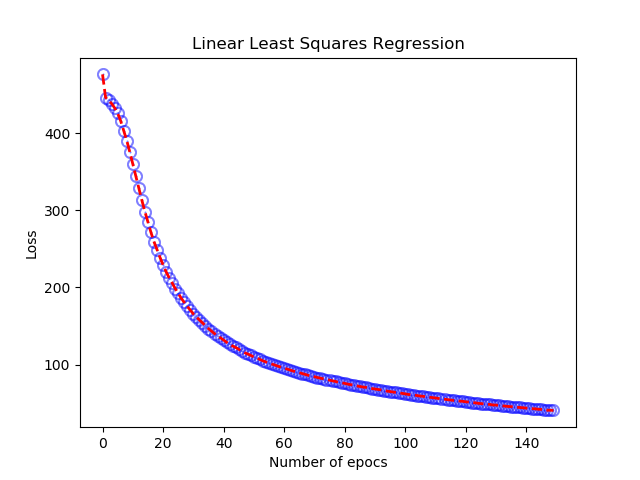
Accuracy for 150x100: **0.8881**



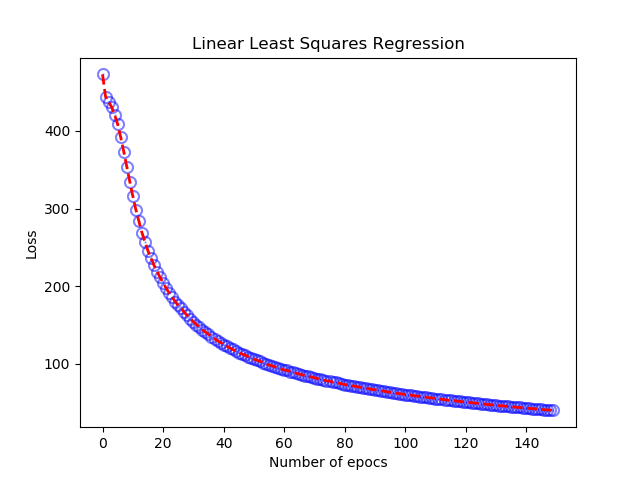
Accuracy for 25x150**: 0.893**



Accuracy for 50x150: **0.8914**

****

Accuracy for 100x150: **0.8903**

****

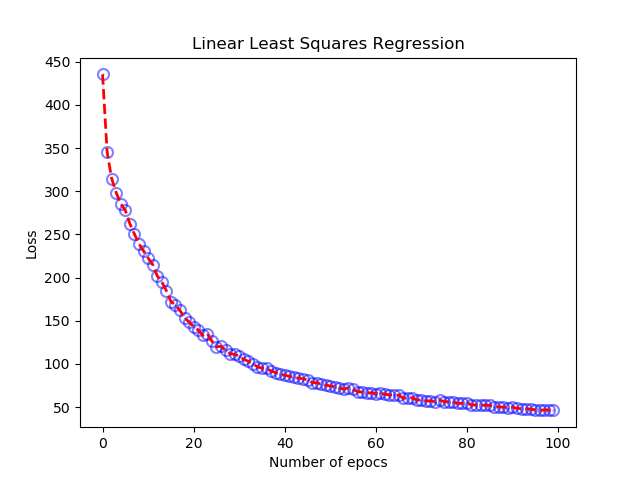
Accuracy for 150x150: **0.8912**

**TANH:**

In the following screenshot we can see the graph showing the error function after using the activation function Tanh. This example is done for a NN with 100 neurons in the hidden layer and 100 number of epochs. As we can see, the error is reduced by less than 10. Using the TanH as activation function, we can see and improvement of our results by applying Mini Batch instead of the SGD. For the Mini Batch I obtained an accuracy close to 90%, while using SGD gave me an accuracy of 83%.

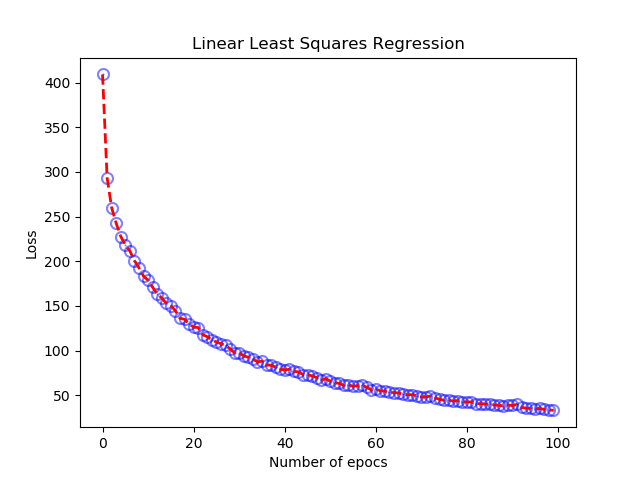
On the other hand, the only problem I had with this function as well as with the RELU was that they are functions that can produce values really far from each other. If the learning rate is not adjusted, there might be huge fluctuations and big jumps while training our NN.

**SGD:**

****

Accuracy: **0.8356**

**Mini Batch:**

****

Accuracy: **0.8983**

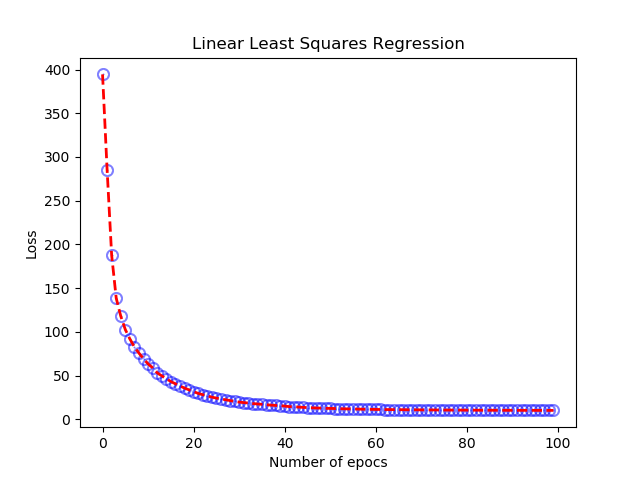
**RELU:**

In the following screenshot we can see the graph showing the results of the error function after using the activation function RELU. This example is done for a NN with 100 neurons in the hidden layer and 100 number of epochs.

As mentioned before, this function can get “crazy” and we need to adjust the learning rate in order to avoid huge alterations on our results. I adjusted the learning rate to 0.005 which means I am just modifying the weight by a 5% of the corresponding obtained gradient.

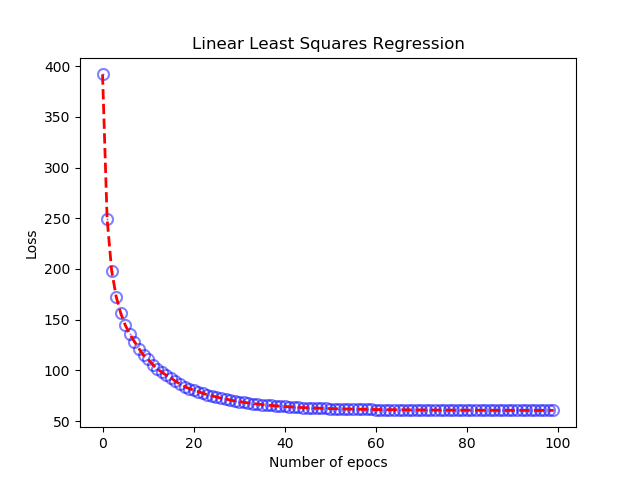
As we can observe, after adjusting the learning rate, we are also getting an error really close to 0 with an accuracy around 90%, for the SHD technique. However, for the Mini Batch, we got an error a little bit higher with a lower accuracy, this might be due to the averages done in the Mini Batch which can make the error reduce slower.

**SHD:**

****

Accuracy**: 0.8845**

**Mini Batch:**

****

Accuracy: **0.8073**

**SOURCE CODE:**

**Sigmoid SGD:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 150

numNeuronsLayer2 = 10

numEpochs = 150

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.1;

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = 1/(1+np.exp(-1\*s1))

s2 = np.dot(w2,a1)+b2

a2 = 1/(1+np.exp(-1\*s2))

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

#loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

delta2 = -np.multiply(np.multiply(trainY[i]-a2,a2),1-a2)

delta1 = np.multiply(np.multiply(np.dot(np.transpose(w2),delta2),a1),1-a1)

gradw2 = np.dot(delta2,np.transpose(a1))

gradb2 = delta2

gradw1 = np.dot(delta1, np.transpose(trainX[i]))

gradb1 = delta1

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_arr[n,0] = loss;

x\_arr[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**Sigmoid Mini Batch:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 25

numNeuronsLayer2 = 10

numEpochs = 50

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.1

gradb1 =0

gradb2=0

gradw1=0

gradw2 = 0

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = 1/(1+np.exp(-1\*s1))

s2 = np.dot(w2,a1)+b2

a2 = 1/(1+np.exp(-1\*s2))

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

#loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

delta2 = -np.multiply(np.multiply(trainY[i]-a2,a2),1-a2)

delta1 = np.multiply(np.multiply(np.dot(np.transpose(w2),delta2),a1),1-a1)

gradw2 += np.dot(delta2,np.transpose(a1))

gradb2 += delta2

gradw1 += np.dot(delta1, np.transpose(trainX[i]))

gradb1 += delta1

if(i%10 == 0):

# adjust the weights

w2 = w2 - learningRate \* (gradw2/10)

b2 = b2 - learningRate \* (gradb2/10)

w1 = w1 - learningRate \* (gradw1/10)

b1 = b1 - learningRate \* (gradb1/10)

gradb1=0

gradw1=0

gradb2=0

gradw2=0

print("epoch = " + str(n) + " loss = " + (str(loss)))

loss\_arr[n,0] = loss

x\_arr[n,0] = n

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = 1/(1+np.exp(-1\*s1)) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = 1/(1+np.exp(-1\*s2))

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**RELU SGD:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

#test = np.array([1,2,3,-2,-4,6])

#test[test<0] = 0

#test[test>0]=1

#print(test)

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 25

numNeuronsLayer2 = 10

numEpochs = 50

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.005;

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = np.maximum(0,s1)

s2 = np.dot(w2,a1)+b2

a2 = np.maximum(0,s2)

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

#loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

relu2\_dev = a2.copy()

relu2\_dev[relu2\_dev<0]=0

relu2\_dev[relu2\_dev>0]=1

delta2 = -np.multiply(trainY[i]-a2,relu2\_dev)

relu1\_dev = a1.copy()

relu1\_dev[relu1\_dev<0]=0

relu1\_dev[relu1\_dev>0]=1

delta1 = np.multiply(np.dot(np.transpose(w2),delta2),relu1\_dev)

gradw2 = np.dot(delta2,np.transpose(a1))

gradb2 = delta2

gradw1 = np.dot(delta1, np.transpose(trainX[i]))

gradb1 = delta1

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_arr[n,0] = loss;

x\_arr[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = np.maximum(0,s1) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.maximum(0,s2)

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**RELU Mini Batch:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

#test = np.array([1,2,3,-2,-4,6])

#test[test<0] = 0

#test[test>0]=1

#print(test)

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 25

numNeuronsLayer2 = 10

numEpochs = 50

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.005;

gradw2 =0

gradb2 =0

gradw1 =0

gradb1 =0

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = np.maximum(0,s1)

s2 = np.dot(w2,a1)+b2

a2 = np.maximum(0,s2)

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

#loss += (0.5 \* np.multiply((a2-trainY[i]),(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

relu2\_dev = a2.copy()

relu2\_dev[relu2\_dev<0]=0

relu2\_dev[relu2\_dev>0]=1

delta2 = -np.multiply(trainY[i]-a2,relu2\_dev)

relu1\_dev = a1.copy()

relu1\_dev[relu1\_dev<0]=0

relu1\_dev[relu1\_dev>0]=1

delta1 = np.multiply(np.dot(np.transpose(w2),delta2),relu1\_dev)

gradw2 += np.dot(delta2,np.transpose(a1))

gradb2 += delta2

gradw1 += np.dot(delta1, np.transpose(trainX[i]))

gradb1 += delta1

if (i%10==0):

# adjust the weights

w2 = w2 - learningRate \* gradw2/10

b2 = b2 - learningRate \* gradb2/10

w1 = w1 - learningRate \* gradw1/10

b1 = b1 - learningRate \* gradb1/10

gradw2 =0

gradb2 =0

gradw1 =0

gradb1 =0

loss\_arr[n,0] = loss;

x\_arr[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = np.maximum(0,s1) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.maximum(0,s2)

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**TanH SGD:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 100

numNeuronsLayer2 = 10

numEpochs = 100

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.05;

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = np.tanh(s1)

s2 = np.dot(w2,a1)+b2

a2 = np.tanh(s2)

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

delta2 = -np.multiply(trainY[i]-a2,1-np.multiply(a2,a2))

delta1 = np.multiply(np.dot(np.transpose(w2),delta2),1-np.multiply(a1,a1))

gradw2 = np.dot(delta2,np.transpose(a1))

gradb2 = delta2

gradw1 = np.dot(delta1, np.transpose(trainX[i]))

gradb1 = delta1

# adjust the weights

w2 = w2 - learningRate \* gradw2

b2 = b2 - learningRate \* gradb2

w1 = w1 - learningRate \* gradw1

b1 = b1 - learningRate \* gradb1

loss\_arr[n,0] = loss;

x\_arr[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = np.tanh(s1) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.tanh(s2)

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**TanH Mini Batch:**

import os

import sys

import cv2

import numpy as np

from sklearn.utils import shuffle

import matplotlib.pyplot as plt

train = np.empty((1000,28,28),dtype='float64')

trainY = np.zeros((1000,10,1))

test = np.empty((10000,28,28),dtype='float64')

testY = np.zeros((10000,10,1))

# Load in the images for training

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/'):

y = int(filename[0])

trainY[i,y] = 1.0

train[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Training1000/{0}'.format(filename),0)/255.0 # for color, use 1

i = i + 1

# Creating testing data

i = 0

for filename in os.listdir('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000'):

y = int(filename[0])

testY[i,y] = 1.0

test[i] = cv2.imread('C:/Users/ivans\_000/Desktop/MASTER/Spring2019/Deep\_Learning/Assignment2\_Sangines/Data/Test10000/{0}'.format(filename),0)/255.0

i = i + 1

trainX = train.reshape(train.shape[0],train.shape[1]\*train.shape[2],1)

testX = test.reshape(test.shape[0],test.shape[1]\*test.shape[2],1)

numNeuronsLayer1 = 100

numNeuronsLayer2 = 10

numEpochs = 100

loss\_arr = np.ndarray((numEpochs,1))

x\_arr = np.ndarray((numEpochs,1))

#-------Randomly initializing weights and bias for each layer of neurons values between -0.1 and 0.1---------

w1 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer1,784))

b1 = np.random.uniform(low=-1,high=1,size=(numNeuronsLayer1,1))

w2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,numNeuronsLayer1))

b2 = np.random.uniform(low=-0.1,high=0.1,size=(numNeuronsLayer2,1))

learningRate = 0.05;

gradw2 = 0

gradb2 = 0

gradw1 = 0

gradb1 = 0

#-------Training Neurons----------------

for n in range(0,numEpochs): #we will iterate 100 times through all images

loss = 0

trainX,trainY = shuffle(trainX, trainY) # shuffle data for stochastic behavior

for i in range(trainX.shape[0]): #each iteration is an image

# do forward pass

s1 = np.dot(w1,trainX[i])+b1

a1 = np.tanh(s1)

s2 = np.dot(w2,a1)+b2

a2 = np.tanh(s2)

# your equations for the forward pass

# do backprop and compute the gradients \* also works instead

# np.multiply

#y = list(trainY[i]).index(1)

loss += (0.5 \* ((a2-trainY[i])\*(a2-trainY[i]))).sum()

# your equations for computing the deltas and the gradients

delta2 = -np.multiply(trainY[i]-a2,1-np.multiply(a2,a2))

delta1 = np.multiply(np.dot(np.transpose(w2),delta2),1-np.multiply(a1,a1))

gradw2 += np.dot(delta2,np.transpose(a1))

gradb2 += delta2

gradw1 += np.dot(delta1, np.transpose(trainX[i]))

gradb1 += delta1

if (i%10==0):

# adjust the weights

w2 = w2 - learningRate \* gradw2/10

b2 = b2 - learningRate \* gradb2/10

w1 = w1 - learningRate \* gradw1/10

b1 = b1 - learningRate \* gradb1/10

gradw2 = 0

gradb2 = 0

gradw1 = 0

gradb1 = 0

loss\_arr[n,0] = loss;

x\_arr[n,0] = n;

print("epoch = " + str(n) + " loss = " + (str(loss)))

area = 2

colors = ['blue']

plt.scatter(x\_arr, loss\_arr, s=area, c=colors, alpha=0.5, linewidths=8) #drawing points using X,Y data arrays

plt.title('Linear Least Squares Regression')

plt.xlabel('Number of epocs')

plt.ylabel('Loss')

line, = plt.plot(x\_arr, loss\_arr, '--', linewidth=2) #line plot

line.set\_color('red')

plt.show()

print("done training , starting testing..")

#-----Testing Given Data----------

accuracyCount = 0

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

# do forward pass

s1 = np.dot(w1,testX[i]) + b1

a1 = np.tanh(s1) # np.exp operates on the array

s2 = np.dot(w2,a1) + b2

a2 = np.tanh(s2)

# determine index of maximum output value

a2index = a2.argmax(axis = 0)

if (testY[i,a2index] == 1):

accuracyCount = accuracyCount + 1

print("Accuracy count = " + str(accuracyCount/10000.0))

**CONCLUSION:**

This assignment really helped me to understand better all the concepts explained in class about NN. I can now understand better the relationship between the different components of a Neural Network (I understand when to use multiply or dot product depending on the dependency/relationship of the components, which was a very confusing topic during the lectures).

On the other hand, this assignment also helped me to get familiar with new activation functions such as RELU and Tanh. I got to understand their behavior and realize those are functions which can get really “crazy”, meaning the values for the new gradients can jump really far from each other. In order to obtain good results, I need to reduce the learning rate so I am just modifying the weights by a small percentage of the new obtained gradient.