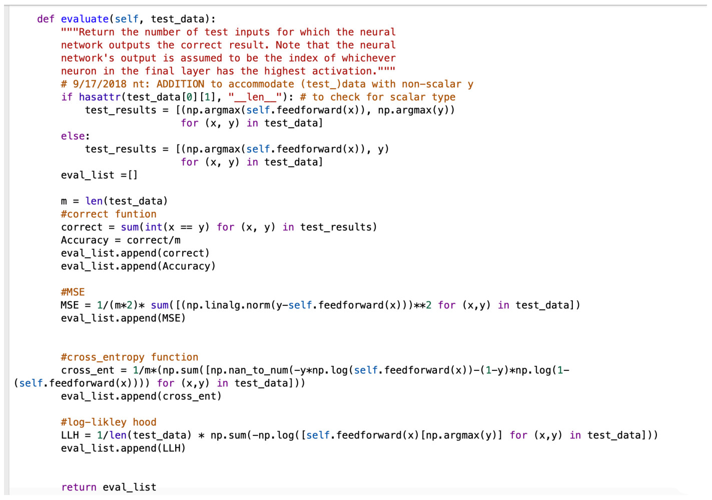
Kubra Iqbal
Homework 2

For Homework 2 – Started with checking if the results or the Iris fixed start experiments matched or not. The results matched with the given results provided in the homework files. The results turned out to be a different a few times when I was running the code but that was some notebook error – I refreshed the page a few times and it worked.

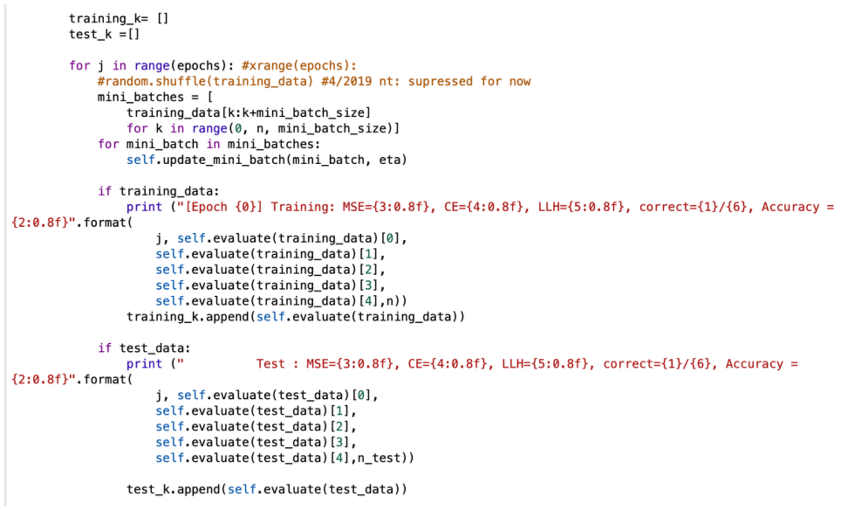
To begin with the homework, the first requirement was to edit the "evaluate ()" function. This is done so that addition to accuracy, it computes the following: MSE, Cross-entropy and log-likelihood. The function should also return those five results when the code is run.

The code snipped that was added to the file is attached below:

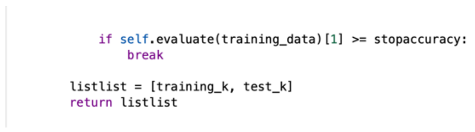


```
    def evaluate(self, test_data):
        """Return the number of test inputs for which the neural
        network outputs the correct result. Note that the neural
        network's output is assumed to be the index of whichever
        neuron in the final layer has the highest activation."""
        # 9/17/2018 nt: ADDITION to accommodate (test_)data with non-scalar y
        if hasattr(test_data[0][1], "__len__"): # to check for scalar type
            test_results = [(np.argmax(self.feedforward(x)), np.argmax(y))
                            for (x, y) in test_data]
        else:
            test_results = [(np.argmax(self.feedforward(x)), y)
                            for (x, y) in test_data]
        eval_list =[]

        m = len(test_data)
        #correct funtion
        correct = sum(int(x == y) for (x, y) in test_results)
        Accuracy = correct/m
        eval_list.append(correct)
        eval_list.append(Accuracy)

        #MSE
        MSE = 1/(m*2)* sum([(np.linalg.norm(y-self.feedforward(x)))**2 for (x,y) in test_data])
        eval_list.append(MSE)


        #cross_entropy function
        cross_ent = 1/m*(np.sum([np.nan_to_num(-y*np.log(self.feedforward(x))-(1-y)*np.log(1-
(self.feedforward(x)))) for (x,y) in test_data]))
        eval_list.append(cross_ent)

        #log-likley hood
        LLH = 1/len(test_data) * np.sum(-np.log([self.feedforward(x)[np.argmax(y)] for (x,y) in test_data]))
        eval_list.append(LLH)


        return eval_list
```

Moving forward, the SGD() function was edited.



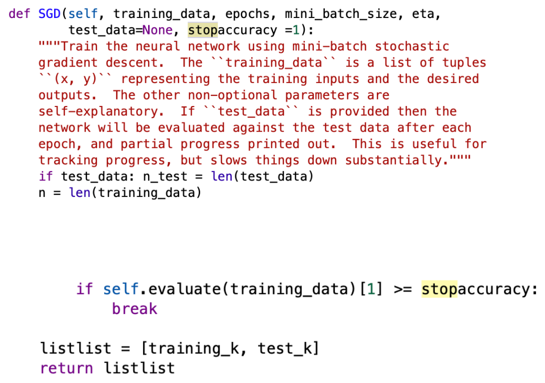
```
        training_k= []
        test_k =[]

        for j in range(epochs): #xrange(epochs):
            #random.shuffle(training_data) #4/2019 nt: supressed for now
            mini_batches = [
                training_data[k:k+mini_batch_size]
                for k in range(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)

            if training_data:
                print ("[Epoch {0}] Training: MSE={3:0.8f}, CE={4:0.8f}, LLH={5:0.8f}, correct={1}/{6}, Accuracy =
{2:0.8f}".format(
                    j, self.evaluate(training_data)[0],
                    self.evaluate(training_data)[1],
                    self.evaluate(training_data)[2],
                    self.evaluate(training_data)[3],
                    self.evaluate(training_data)[4],n))
                training_k.append(self.evaluate(training_data))

            if test_data:
                print ("         Test : MSE={3:0.8f}, CE={4:0.8f}, LLH={5:0.8f}, correct={1}/{6}, Accuracy =
{2:0.8f}".format(
                    j, self.evaluate(test_data)[0],
                    self.evaluate(test_data)[1],
                    self.evaluate(test_data)[2],
                    self.evaluate(test_data)[3],
                    self.evaluate(test_data)[4],n_test))

                test_k.append(self.evaluate(test_data))
```


```
            if self.evaluate(training_data)[1] >= stopaccuracy:
                break

        listlist = [training_k, test_k]
        return listlist
```

Edited the function **SGD()**; I added the list for training and testing data, training_k =[], test_k =[]
The self-evaluate function was called at the end of every epoch and print the results.
After performing this data, I collected the performance results that were returned by the evaluate function into training and testing data into individual results. Each list will be like a history since it is collected by the performance results for each data set.

Furthermore, added a function parameter 'stop accuracy' which was added with a default value of 1.0 that was required. This parameter is used to do the early stopping which would terminate the spoch loop prematurely.
Early stopping requires that you configure the network to be under constrained, meaning that it has more capacity than it was required for the problem. When training the network, a larger number of training epcohs is used than many normally required. To give the network plenty of opportunity to fit, then begin the overfitting the training dataset. There are three elements to using early stopping, monitoring model performance and triggering to stop training.

The screen shot below shows how I added it in the file.

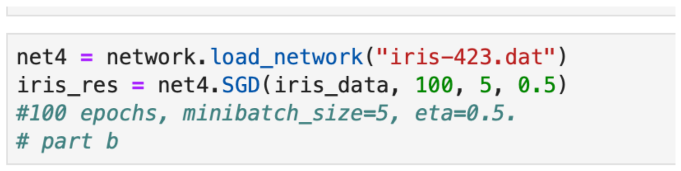

```
    def SGD(self, training_data, epochs, mini_batch_size, eta,
            test_data=None, stopaccuracy =1):
        """Train the neural network using mini-batch stochastic
        gradient descent.  The ``training_data`` is a list of tuples
        ``(x, y)`` representing the training inputs and the desired
        outputs.  The other non-optional parameters are
        self-explanatory.  If ``test_data`` is provided then the
        network will be evaluated against the test data after each
        epoch, and partial progress printed out.  This is useful for
        tracking progress, but slows things down substantially."""
        if test_data: n_test = len(test_data)
        n = len(training_data)



            if self.evaluate(training_data)[1] >= stopaccuracy:
                break

        listlist = [training_k, test_k]
        return listlist
```

Lastly, for this function backprop() – the local variable 'activations' is what was initially that was allocated with a structure which holds the activation value of all the layers in the network.
The code snippet below shows how I was able to edit the function and add and append.

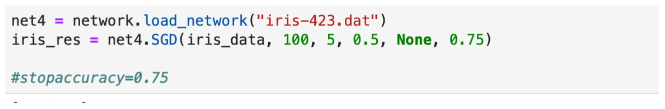
```
def backprop(self, x, y):
    """Return a tuple ``(nabla_b, nabla_w)`` representing the
    gradient for the cost function C_x.  ``nabla_b`` and
    ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
    to ``self.biases`` and ``self.weights``."""
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activations = [np.zeros([self.sizes[i],1]) for i in range(len(self.sizes))]

    zs = [] # list to store  all the z vectors
    activations [0] = x
    activation = x

    acc = iter(range(len(activations)-1)) #added this


     # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        for i in acc:
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = sigmoid(z)
            activations[i+1] = activation
```

For the second part of the assignment – on the application side. I started with adding a collection of code snippets to run the startup code and ensure that the output was matching the output that was provided in the HTML version of the code.

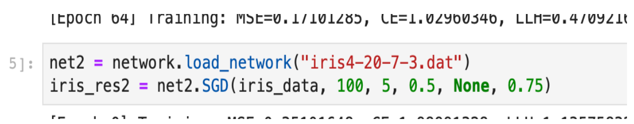
For Question 2/ parts "a" and "b" –



```
net4 = network.load_network("iris-423.dat")
iris_res = net4.SGD(iris_data, 100, 5, 0.5)
#100 epochs, minibatch_size=5, eta=0.5.
# part b
```



```
net4 = network.load_network("iris-423.dat")
iris_res = net4.SGD(iris_data, 100, 5, 0.5, None, 0.75)

#stopaccuracy=0.75
```

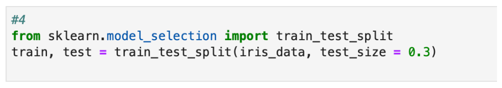


```
[Epoch 64] Training: MSE=0.17101285, CE=1.02960346, LLH=0.470921
```

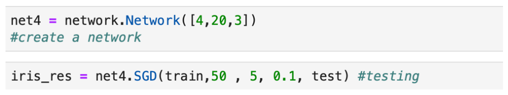
```
5]: net2 = network.load_network("iris4-20-7-3.dat")
    iris_res2 = net2.SGD(iris_data, 100, 5, 0.5, None, 0.75)
```

All the results matched the HTMLS that were provided when these code snippets attached above were added to the notebook.

For Question 4 – a,b and c
Split the dataset randomly into 70% training and 30% testing. The snippet below shows how I was able to do it. Moreover, created a new network which randomly initialized weights of the size provided.



```
#4
from sklearn.model_selection import train_test_split
train, test = train_test_split(iris_data, test_size = 0.3)
```



```
net4 = network.Network([4,20,3])
#create a network
```

```
iris_res = net4.SGD(train,50 , 5, 0.1, test) #testing
```

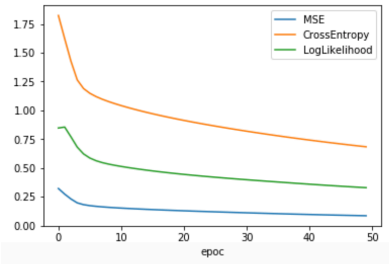
Furthermore, for question 4 d i and ii :
I used the results to plot two types of plots that are shown below. I ran the code approximately 8-10 times to come up with the results.

<u>The graph below compares the error curved of the three cost functions for the provided training set.</u> The MSE is furthest below, LogLikeihood is above that and lastly CrossEntropy is on the top of them both. They are not smoothly decreasing, it mostly a stagnant line, once they trop from a very high number.

In the start, they are pretty up high respectively and then have a big drop, and further become stagnant while decreasing slowly.

```
train_value = iris_res[0]
train_MSE = []
train_ce =[]
train_LLH = []
for i in train_value:
    train_MSE.append(i[2])
    train_ce.append(i[3])
    train_LLH.append(i[4])
```

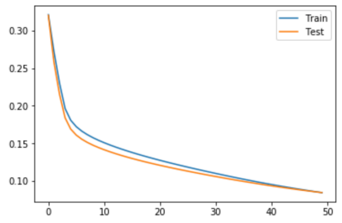
```
pltMSE_train = plt.plot(train_MSE)
pltce_train = plt.plot(train_ce)
pltLLH_train = plt.plot(train_LLH)
plt.legend(['MSE','CrossEntropy', 'LogLikelihood'], loc ='upper right')
plt.xlabel("epoc")
plt.show()


#4.d.1
```



<u>The graph below shows MSE cost function that shows error for training vs test.</u>

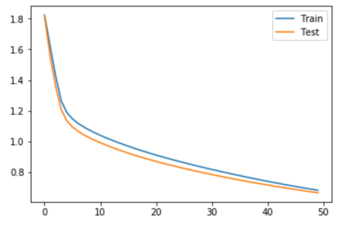
The graph below shows that they are overfitting – It shows that the function is too closely fit to a limited set of data points.

```
plotMSE_train = plt.plot(train_MSE)
plotMSE_test = plt.plot(testing_MSE)
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```



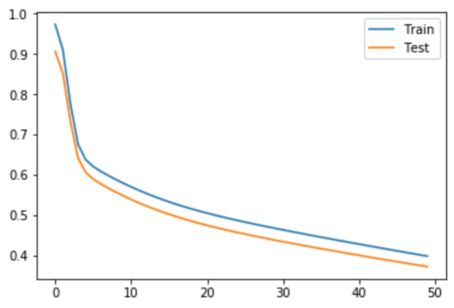
The graph below shows Cross entropy function – the training and testing lines are close to each other showing the same pattern.

```
pltce_train = plt.plot(train_ce)
plotce_test = plt.plot(testing_ce)
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```



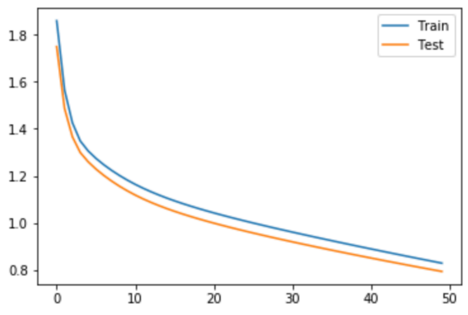
The graph below shows LogLikeliHood with Training and testing, the lines are very close to each other. Which means that the model is working perfectly. The lines are very close to each other and are almost following the same pattern. They are decreasing very steeply and then decreasing slowly. This also means that the model worked accurately.

```
pltLLH_train = plt.plot(train_LLH)
plotLLH_test = plt.plot(testing_LLH)
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```



The Graph below shows Cross Entropy with training and testing – the same pattern is being repeated. The training and testing lines are very close to each other, following the same pattern. Which means that the model worked.

```
pltce_train = plt.plot(train_ce)
plotce_test = plt.plot(testing_ce)
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```



Reflection:

This assignment did not come easy to me. There were a lot of parts for this assignment that I had to look up additional videos to understand and read different articles to understand the concepts. Maybe this is because I don't have a computer science background – but this assignment was very stressful.
I spent a lot of hours working on the code, I did learn a lot for sure, but I think it is a lot of work to do every week since I am taking two other classes and working.
If the assignments can be due after two weeks, I think that will be a huge help and will also help me learn better. For this assignment I spent a lot of hours trying to figure out most things, maybe the next ones will come a little easy to me, but it was a big tough challenge. I feel like the point shouldn't be that we have to finish all these homework's, but to learn as much possible and having to work on assignments which are due every weekend, I am struggling, learning less and not learning as much as I should.
I would rate this assignment to be the highest level on the difficulty – the last two assignments were not that hard for me because I do have a math background and I understood what was taught in the class as well regarding them. I am hoping the assignments coming forward will come easy to me.