Faculty of Engineering

Ain Shams University

2021-2022

Neural Networks and Their Applications

CSE 616

PG2015

Assignment #: 1

Title: ***Wine Quality Deep Neural Networks Results and Discussion***

Made by:

2100966 حمدي أسامه محمد السيد فرج

Mechatronics PG Student

`

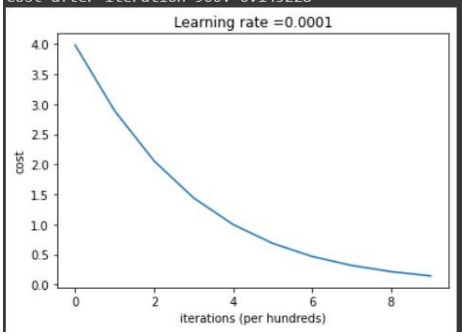
A neural network with one hidden layer of 30 nodes is designed.

Forward propagation, cost computation, backward propagation, and parameter updating functions are all hardcoded and used in a model function used to identify number of iterations and learning rate.

The first design had all activation functions of the sigmoid type and a cross entropy cost function. The output layer was a single node sigmoid function, and it was not a very good choice for this multi class classifier as it has an output range of 0-1 only.

```
#training the model and getting parameters to be used in testing

#getting the real wine quality from training data

ground_truth = np.reshape(np.append(training_data[:,11],[1]),[800,1])
parameters=L_layer_model(final_training_set.T, ground_truth.T, [11,30,1], learning_rate = 0.0001, num_iterations = 1000, print_cost=True)
```

```
Cost after iteration 0: 3.980618
Cost after iteration 100: 2.887310
Cost after iteration 200: 2.054196
Cost after iteration 300: 1.439626
Cost after iteration 400: 0.997545
Cost after iteration 500: 0.685420
Cost after iteration 600: 0.467998
Cost after iteration 700: 0.318004
Cost after iteration 800: 0.215252
Cost after iteration 900: 0.145228
```




```
#testing the model using the testing_data set and the trained parameters

prediction,cache = L_model_forward(testing_data[:,0:11].T,parameters)

RMSE=np.sqrt(np.square(np.subtract(testing_data[:,11],prediction.T)).mean() )

prediction

RMSE
```

```
4.760951621159981
```

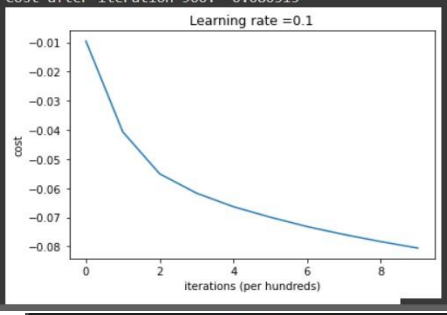
This bad performance, even when the cost is decreasing as iterations pass, was shown evidently in the RMSE when it is calculated. A value of 4.7 error means that wine quality can vary by a large value in the prediction classifier.

`

To solve this problem, we make the output layer size equal to the possible wine qualities available, which is 11 nodes. Each node in the output layer is a sigmoid function; note that for optimal results, SoftMax activation function should be used as it is based upon giving probability for every class in a multi class classifier. The class with the highest probability is the output prediction of the forward pass of the neural network. We used the sigmoid activation function for the output layer because its formulation is relatively simple, unlike the tedious task of solving for the SoftMax function derivative. We also need to modify the cross-entropy function to become a categorial cross entropy that allows for more than 2 classification categories. Note that we also modified the activation function of the hidden layer to be ReLU as it is known that ReLU performance is superior to sigmoid because of the saturation of gradient that occurs in the sigmoid. By making these modifications, the following results are achieved.

```
#getting the real wine quality from training data

ground_truth = np.reshape(np.append(training_data[:,11],[1]),[800,1])

ground_truth_hot = np.reshape(np.append(truth_training_classification,[1,1,1,1,1,1,1,1,1,1,1]),[800,11])

parameters=L_layer_model(final_training_set.T, ground_truth_hot.T, ground_truth, [11,30,11], learning_rate = 0.1, num_iterations = 1000, print_
```

```
Cost after iteration 0: -0.009530
Cost after iteration 100: -0.040667
Cost after iteration 200: -0.055050
Cost after iteration 300: -0.061684
Cost after iteration 400: -0.066326
Cost after iteration 500: -0.069929
Cost after iteration 600: -0.073153
Cost after iteration 700: -0.075866
Cost after iteration 800: -0.078305
Cost after iteration 900: -0.080515
```



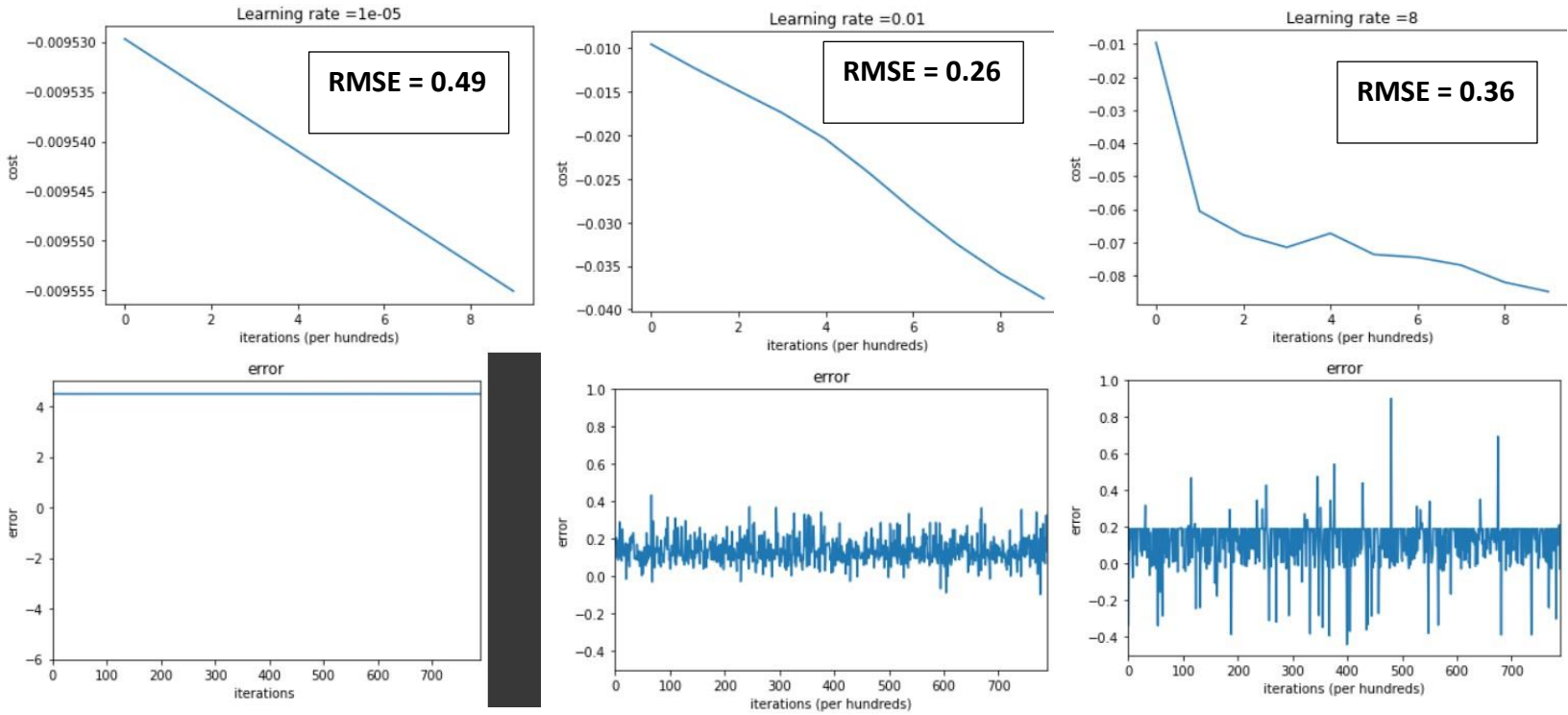

```
#testing the model using the testing_data set and the trained parameters

prediction,cache = L_model_forward(testing_data[:,0:11].T,parameters)

#making a matrix full of zeros
truth_testing_classification = np.zeros([len(testing_data),11])

#placing 1 in the index of the right classification
for testing_example in range(0,len(testing_data)):
  truth_testing_classification[testing_example][int(testing_data[:,11:12][testing_example][0])]=1

RMSE=np.sqrt(np.square(np.subtract(truth_testing_classification,prediction.T)).mean() )

print(prediction.T[1])
print(truth_testing_classification[1])
RMSE
```

```
[0.01597656 0.01501821 0.01682685 0.01659245 0.00506496 0.37146778
 0.20716352 0.18398676 0.14464597 0.01678149 0.01706825]
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
0.2985674455886031
```

`

As seen in the new results, the RMSE is much better than before with a value of 0.29, and an example of the prediction output is printed and compared to the ground truth of the testing data. A wine sample of quality 6 is successfully predicted as the sixth number in the prediction output is the highest. Other values are near zero, which is a good indication. Note that using SoftMax would lead to better results, but this would require using deep network frameworks, which is banned in this assignment.

Studying the effect of changing the learning rate, we find that large learning rates lead to overshooting in the neural network as seen below in the error graph. Higher values than a learning rate of 8 lead to NAN values of cost, which indicates divergence in the neural network, and it will not converge with such a high overshoot. Also, as seen below, lowering the learning rate reduces the overshoot and thus has better performance as the RMSE also is reduced. However, having a very small learning rate makes the learning process very slow (non-existent in the case of 0.00001), and the error does not change in the 1000 iterations of this training.



In the end, we find that having a fast learning rate at the first of the training is good as the neural network reaches the minimum point faster, but a low learning rate is good at the end of the training to fine tune the results. Also, the negative values of the cost function are somewhat surprising as I thought that cost functions are never negative, but as long as the cost is decreasing (higher magnitudes of negative numbers), I believe it is fine. Finally, using SoftMax and better prepared DL frameworks would lead to better results as more features are available for us to use there. The codes used here were referenced from Deep Learning Specialization tutorials, and some modifications were added to them to suit my usage and available data.