HappyMonk Internship Test

Kunal Vaidya, kunavaidya@gmail.com

1. Problem

Selection of the best performing AF for classification task is essentially a naive (or brute-force) procedure wherein, a popularly used AF is picked and used in the network for approximating the optimal function. If this function fails, the process is repeated with a different AF, till the network learns to approximate the ideal function. So the task is to automatically learn optimal activation function using backpropagation. The activation function is of the form $g(x) = k_0 + k_1 x + k_2 x^2$ where k_0, k_1, k_2 are parameters which can be learned while training.

2. Implementation Details

For this demonstration a Feed Forward Neural Network is considered, the neural network consists of two hidden layers consisiting of n and p nodes respectively and a softmax layer in the output stage. For my implementation I have taken n = 2 and p = 2, although the number of layers can be changed by adding elements to the list **layer_dims**. The implementation of neural network is done completely in python using NumPy.

2.1 Dataset

The dataset used in the demonstration is **Banknote Authentication Dataset** which contains 1372 examples with a binary classification label for each example.

2.2 Initial Settings

All the parameters i.e Weight matrices, bias, k_0, k_1 and k_2 were initialized by sampling from a standard normal distribution. The learning rate was set to be 0.00001.

2.3 Algorithms

For updating parameters of the model I have used Mini Batch Gradient Descent i.e updating parameters after every minibatch is processed. Backpropagation and update equations are used as given in the test paper.

3. Parameter Updates

I have recorded the model parameters at different epochs to show tha the parameters are updating.

```
{'K': [0.9015907205927955, 0.5024943389018682, 0.9008559492644118],
 'W1': array([[ 0.01624345, -0.00611756, -0.00528172, -0.01072969],
       [ 0.00865408, -0.02301539,  0.01744812, -0.00761207]]),
 'W2': array([[ 0.01462108, -0.02060141],
       [-0.00322417, -0.00384054]]),
 'W3': array([[-0.00172428, -0.00877858],
       [ 0.00042214,  0.00582815]]),
 'b1': array([[ 0.3190391 ],
       [-0.24937038]]),
 'b2': array([[ 1.13376944],
       [-1.09989127]]),
 'b3': array([[-1.10061918],
       [ 1.14472371]])}
```

```
{'K': array([0.90184783, 0.50224011, 0.90084784]),
 'W1': array([[ 0.01620773, -0.00612439, -0.0052624 , -0.01072469],
       [ 0.00866166, -0.02297769,  0.01741097, -0.00762752]]),
 'W2': array([[ 0.00850346, -0.02502216],
       [-0.00123146, -0.00240054]]),
 'W3': array([[-0.00059944, -0.00815527],
       [ 0.00132459,  0.00632835]]),
 'b1': array([[ 0.31798158],
       [-0.2494087 ]]),
 'b2': array([[ 1.12816178],
       [-1.09806451]]),
 'b3': array([[-1.10016949],
       [ 1.14507902]])}
```


Figure 1: (a) Initial Parameters (b) Parameters after 5 epoch


```
{'K': array([0.90241784, 0.5020205 , 0.9008746 ]),
 'W1': array([[ 0.01617218, -0.00613113, -0.00524322, -0.01071971],
       [ 0.00866922, -0.02294006,  0.0173739 , -0.00764293]]),
 'W2': array([[ 0.00255463, -0.02932633],
       [ 0.00073763, -0.00097583]]),
 'W3': array([[ 0.00050789, -0.00753506],
       [ 0.00221311,  0.00682609]]),
 'b1': array([[ 0.31692689],
       [-0.24944682]]),
 'b2': array([[ 1.12270507],
       [-1.09625815]]),
 'b3': array([[-1.0997198 ],
       [ 1.14543433]])}
```

```
{'K': array([0.92707368, 0.50125321, 0.9031452 ]),
 'W1': array([[ 0.0157566 , -0.00620548, -0.00502187, -0.01066116],
       [ 0.00875837, -0.02249024,  0.0169306 , -0.00782594]]),
 'W2': array([[-0.05866135, -0.07404296],
       [ 0.02295472,  0.01525753]]),
 'W3': array([[ 0.01267739, -0.00027711],
       [ 0.01198461,  0.01265223]]),
 'b1': array([[ 0.30441142],
       [-0.24989294]]),
 'b2': array([[ 1.06667942],
       [-1.0759261 ]]),
 'b3': array([[-1.09434936],
       [ 1.1497239 ]])}
```


Figure 2: (a) Parameters after 10 epochs (b)Final parameters after 70 epochs

Here W_1, W_2, W_3 are weights of layer 1, 2, 3 respectively and array K are the parameters k_0, k_1, k_2. The initial value of K was [0.9015907205927955, 0.5024943389018682, 0.9008559492644118] and after 70 epochs K was [0.92707368, 0.50125321, 0.9031452].

4. Results and Plots

I used a model with n = 2 and p = 2 (number of neurons in hidden layers) for the implementation.

1) I achieved a accuracy of 77.65 % on training set and accuracy of 81.63 % on testing set after 70 epochs.
2) Loss on training set was 6.936341279342004 and on testing set it was 5.837728920179703.
3) F1-score on training set was 76.62 % and on testing set it was 81.08 %.

4.1 Loss function

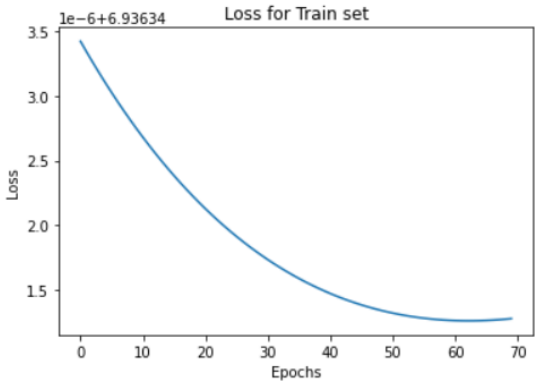

Figure 3: Loss function for train set

4.2 Train Accuracy vs Test Accuracy

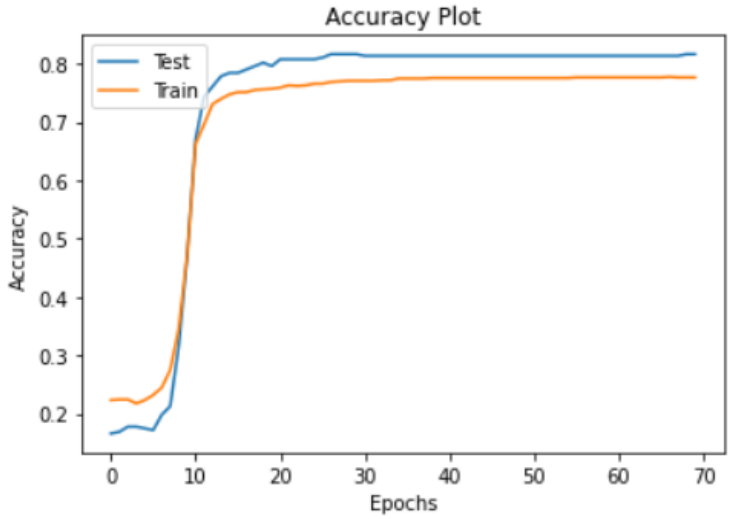

Figure 4: Accuracy Plot

4.3 Confusion Matrix

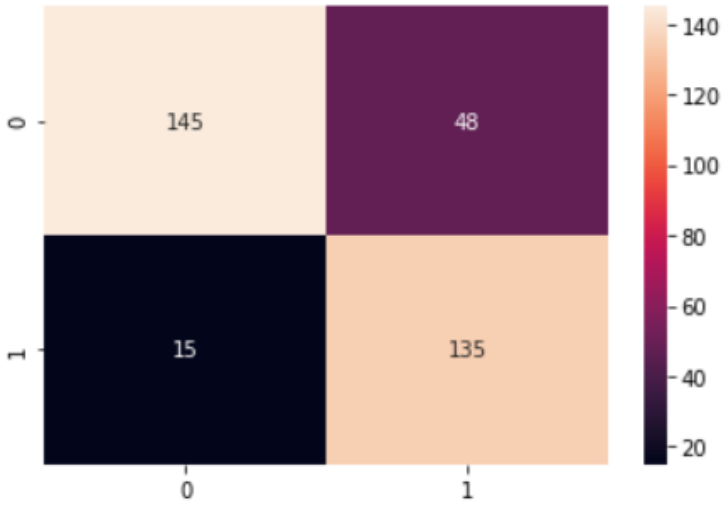

Figure 5: Confusion Matrix for Test set

5. Code

The Jupyter Notebook containing the implementation can be found at **https://github.com/KunalVaidya99/HappyMonk-Internship-Test.**