Technical Report

Title – Implementing ANN on Iris Dataset and understanding changing of the weights and the bias at every epoch

Author – Anik Chatterjee

Introduction – Here I developed a model on Iris dataset that will predict the class of the Iris depending on the parameters ('SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm') and analyzed some parameters

Summary - Developed a model designed in the question with 2 dense layer with 2 normal activation function I put 'Relu' on them and the last classifier layer with the 'softmax' activation function and analyzed how the weights and bias are changing at every epoch with that analyzed the behavior of train VS test (Loss & Accuracy)

Details of the experiment –

First download the Iris data then

1. **categorized** the data and split into Data (X - first 4 columns) and the target (y - last column)
2. Split this 2 into 4 parts **X_train, X_test, Y_train, Y_test** (test value is getting 15% of total value)
3. The target(y) part is the categorical value and we know for make the model to understand those value we have do encoding used np_**utils.to_categorical** to make that encoding in Y_train, Y_test
4. Now data is ready for the model (Used **tensorflow 2** here)
5. 5.1 model = Sequential()

   model.add(Dense(16,input_dim = 4,activation="relu"))

   z1 = a0 _ w1 + b1
   a1 = g(z1)

   Used a Dense layer with 16 units input dimension is 4 and the activation function is relu

   model.add(Dense(8,activation="relu"))

   z2 = a1 _ w2 + b2
   a2 = g(z2)

   Used a Dense layer with 8 units and the activation function is relu again

   model.add(Dense(n_classes,activation="softmax"))

   z3 = a2 _ w3 + b3
   a3 = Softmax(z3)

   Used a Dense layer with n_classes units here it's 3 and the activation function is Softmax

   Total parameters - **243**

   5.2 Compile the model on the '**Categorical Cross-entropy'** used the **adam** optimizer with the default learning rate **(0.001)**

   Fit the model with a validation split of **30%** of train data with a **batch_size** of **16** trained on **100** epoch took near about 5-10 sec

Results and discussions – Checked the **f1** – score and **recall** as

precision recall f1-score support

0	0.78	1.00	0.88	7
1	0.85	0.79	0.81	14
2	0.00	0.00	0.00	2
accuracy			0.78	23
macro avg	0.54	0.60	0.56	23
weighted avg	0.75	0.78	0.76	23

and the **confusion metrics** –

[[7 0 0]

[2 11 1]

[0 2 0]]

And the **weights** on every epoch

0B -> L1N0: 0.0

0B -> L1N1: -0.058848243206739426

0B -> L1N2: -0.07102012634277344

0B -> L1N9: -0.057848602533340454

0B -> L1N10: 0.0

…….

0B -> L1N11: 0.0

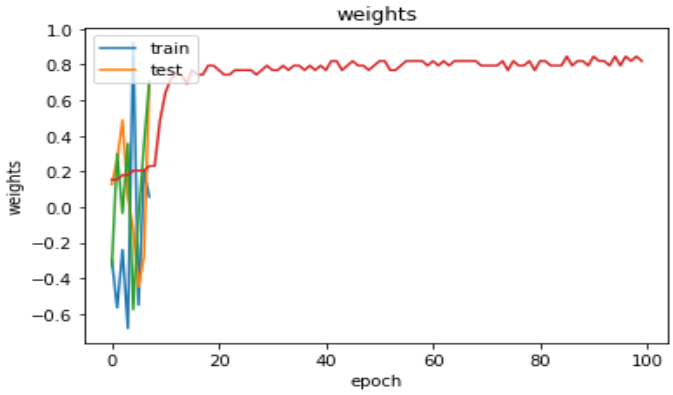
L0N0 -> L1N1 = -0.09267701208591461

……..

L2N7 -> L3N1 = 0.6934090852737427

L2N7 -> L3N2 = 0.7062824368476868

Here the L is the layer number and N is the node number

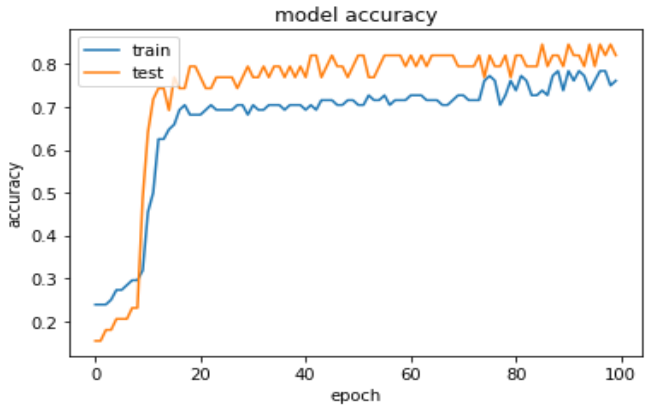


Accuracy checking –

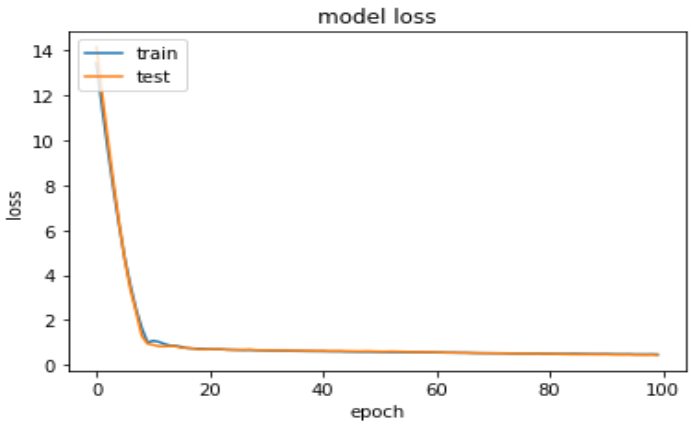
Test Accuracy: 78.261%

Train Accuracy: 75.591%

Train vs. Test accuracy plot



Training vs Test loss plot



Loss Function Plot with confusion metrics rates