INTRODUCTION TO MACHINE LEARNING
FALL 2018
PROJECT 1.1

Logic:

We start by writing a simple function catering all the conditions that are required to map input number to the values in the set {"fizz"," buzz", "fizz buzz", "other"}

Training and Testing datasets:

Both the training and the testing datasets are made with two attributes namely "input" and "label" and the data is preprocessed and converted into the 10bit binary code which can also be said as the normalization of the data which helps the model learn the data fast and converge better at the end.

Model Creation:

The model that was created in this implementation is of a sequential composition which is basically a stack of layers. In this model, there is one input layer with 10 nodes, two hidden layers with 256 and 300 nodes in each and finally there is an output layer with 4 nodes. All the nodes in one layer are connected to all nodes in the next layer. Every layer has an activation function that induces non-linearity. The activated function used for the first hidden layer and the second hidden layer are 'relu' and 'sigmoid' respectively. The 'relu' activation function basically removes the negative part of the inputs. The 'sigmoid' activation function maps the input between 0 to 1 so that the neural network so that the values do not increase exponentially. There is a dropout layer between the two hidden layers to dropout to avoid the overfitting where the model memorizes the training set a lot and then imposes higher degree polynomials to fit the data resulting in the convolutions and the higher training accuracy and comparatively less testing accuracy. Dropout forces the model to not memorize how much a node contributes preventing the overfitting. The output layer has 4 nodes corresponding to the 4 values of the label that was mentioned above. The activation function corresponding to the output layer is 'softmax' which basically returns the probabilities of the output when there are more than one possible outputs from which we decide the result by choosing the output value with the highest probability. The optimizer used is the 'Adam' and the loss function used is the 'categorical_crossentropy'. Here, the accuracy measure is evaluated during the training and testing.

Hyper Parameters:

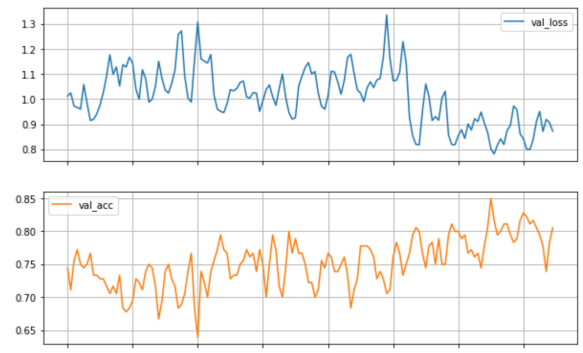
Hyper parameters are the parameters which are set on beforehand that help the model in learning the data. Some of the hyper parameters are the learning rate, optimizer, batch size etc.

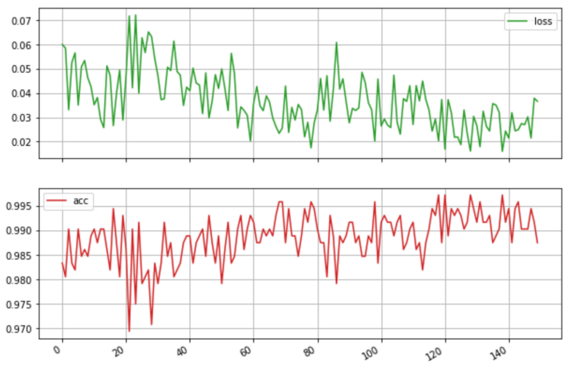
Learning rate:

It is the parameter that defines how fast the model optimizes itself and minimizes the loss function. With the less learning rate, the model takes more time to descend down the curve but it is a good idea that we do not lose any value. In contrast if the learning rate is high there is a high chance that the model loses any value and the coverage is not good as compared to the less learning rate.

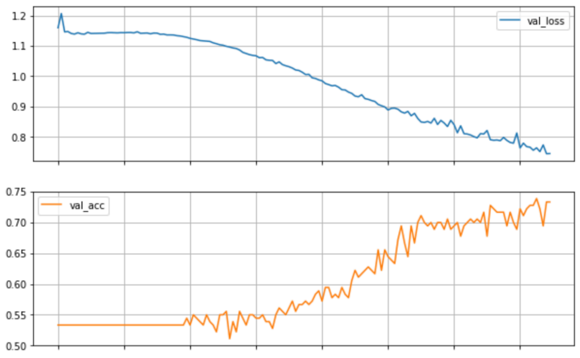
In this implementation, I have used the Adam optimizer and changed the learning rates by keeping the epoch and batch size values constant at 1000 and 128 respectively.

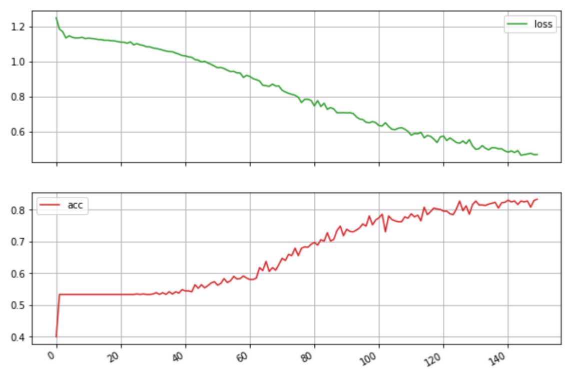
For learning rate 0.1:
(epoch on x-axis)



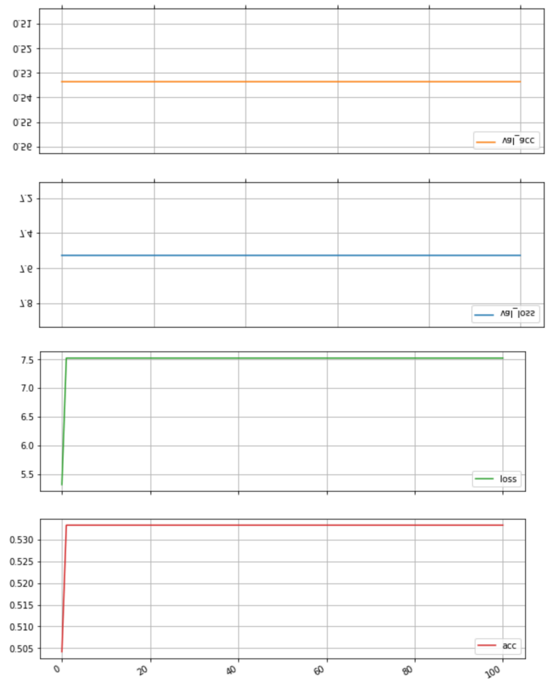


For learning rate 0.01:
(epoch on x-axis)





For learning rate 0.9:
(epoch on x-axis)

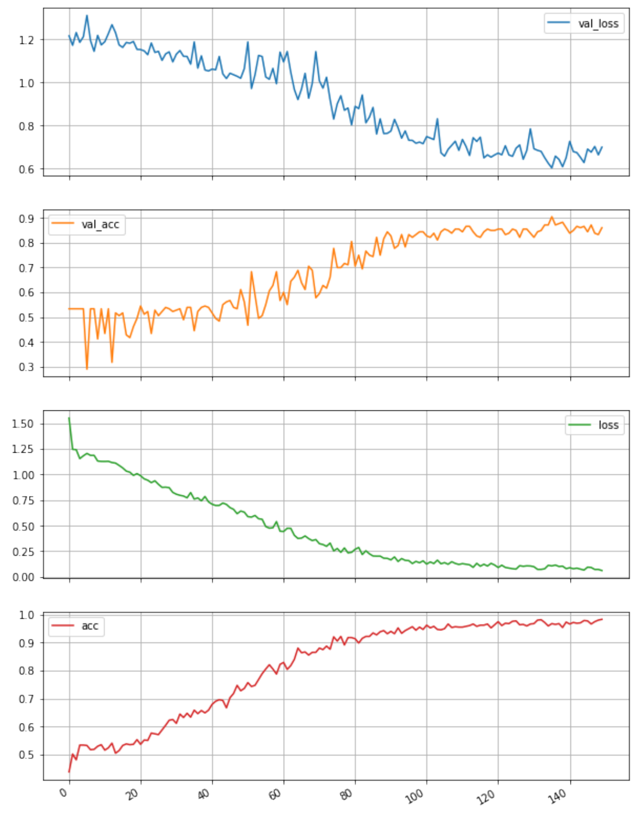


As observed from the graphs when we increase the value of learning rate, the accuracy decreases stating that the coverage by the model with the higher learning rate is minimal.

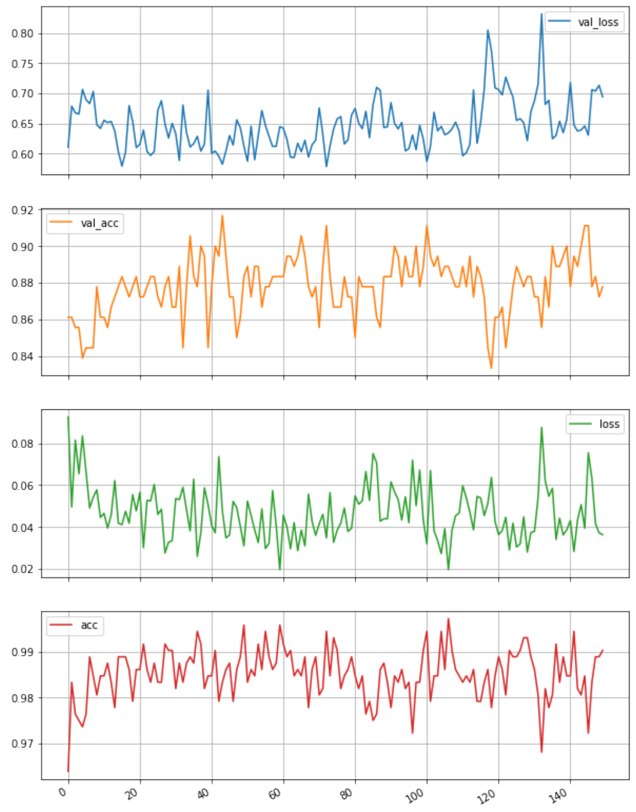
Batch Size:

Batch size the number of the samples that are being sent to all the layers. With the smaller batch size, the model tends to train faster when compared to the case when the batch size is larger. Now keeping the other parameters constant, let us change the batch size and check the variations in the loss and accuracy. Here, the learning rate is kept constant at 0.01 and the epoch at 150

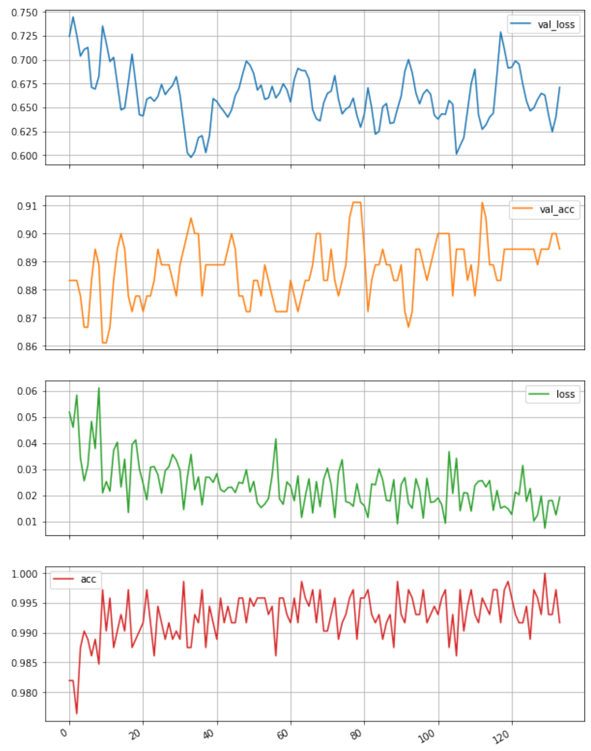
For batch size as 50:



For batch size 100:



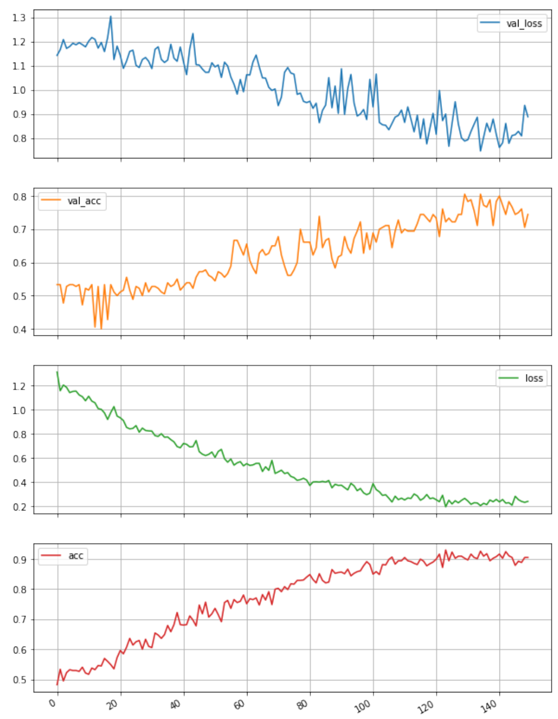
For batch size 150:



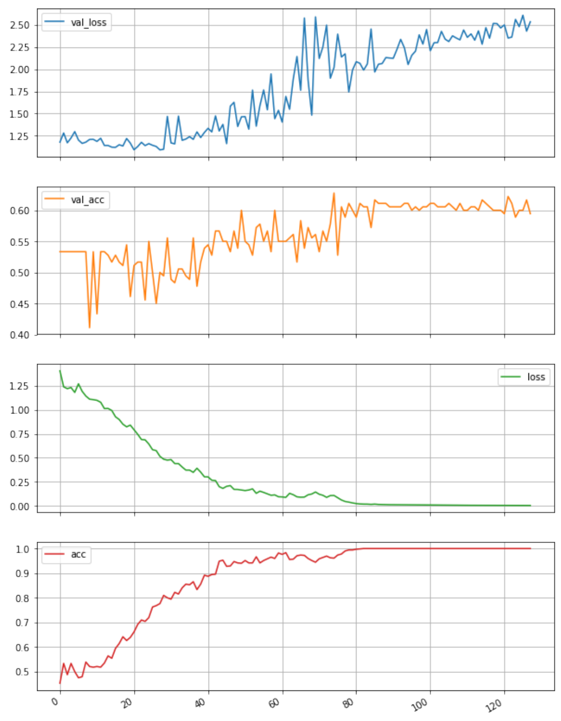
Dropout:

We use dropout to avoid the overfitting where the model memorizes the training set a lot and then imposes higher degree polynomials to fit the data resulting in the convolutions and the higher training accuracy and comparatively less testing accuracy. Dropout forces the model to not memorize how much a node contributes preventing the overfitting. The dropout function makes sure that the model doesn't attain the higher accuracy all of a sudden by nullifying some of the neuron randomly. It is seen from the observations that the model with higher dropout rate take more time for the convergence.

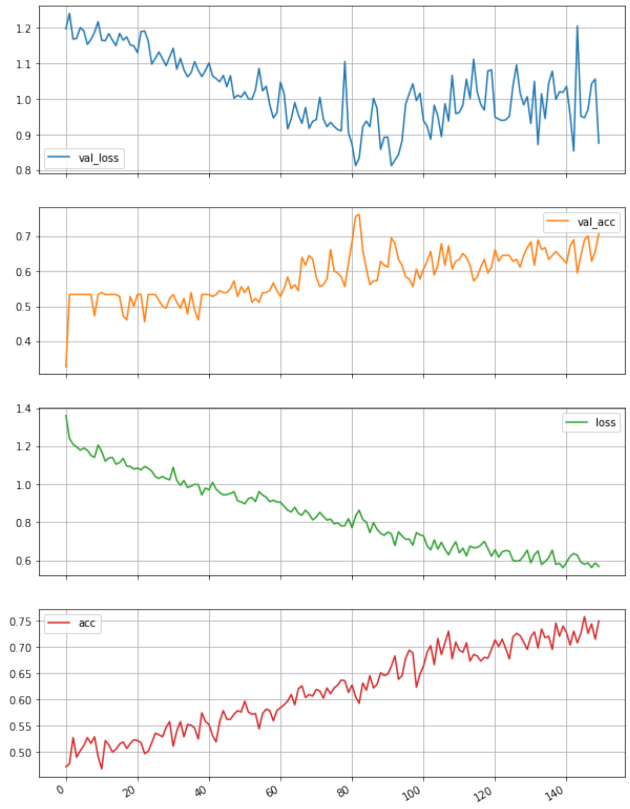
For dropout rate 0:



For dropout rate 0.3:



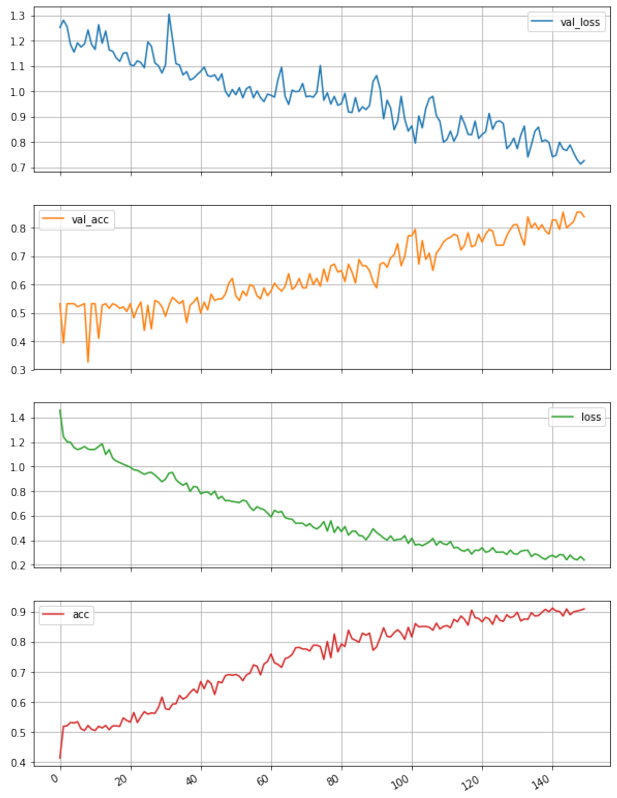
For dropout rate 0.6:



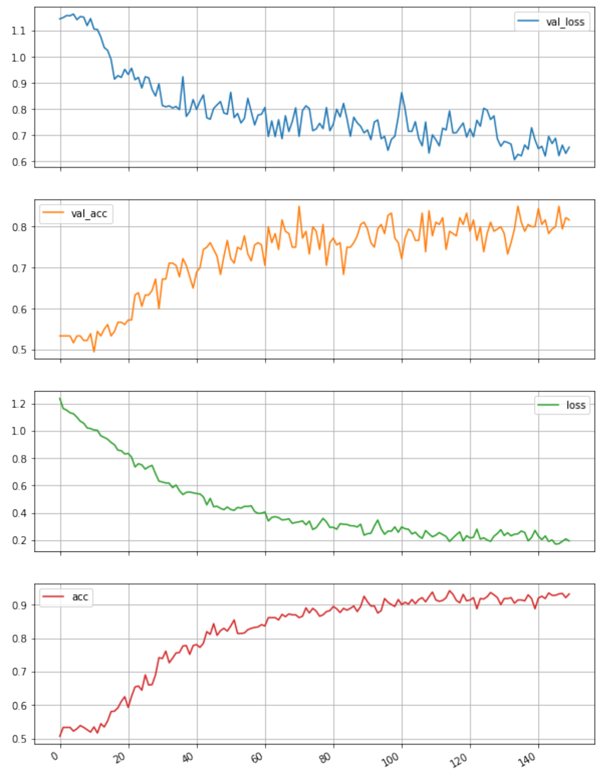
Activation Function:

After the layers are defined and connected, the activation functions define how the neurons behave and introduce the non- linearity to the neural network. Now let us observe the accuracy changes by changing the activation function of one of the layer in our model. Let us change only the activation function of the second hidden layer by keeping the other parameters constant.
The epoch value is 150, batch size is 50, dropout value=0.3, optimizer is adam with learning rate 0.01. With the activation function as the sigmoid the accuracy was around 94% and the least accuracy was obtained with the tanh. The highest accuracy was obtained with the relu activation function at about 96%. This is the reason why relu was chosen for as the activation function.

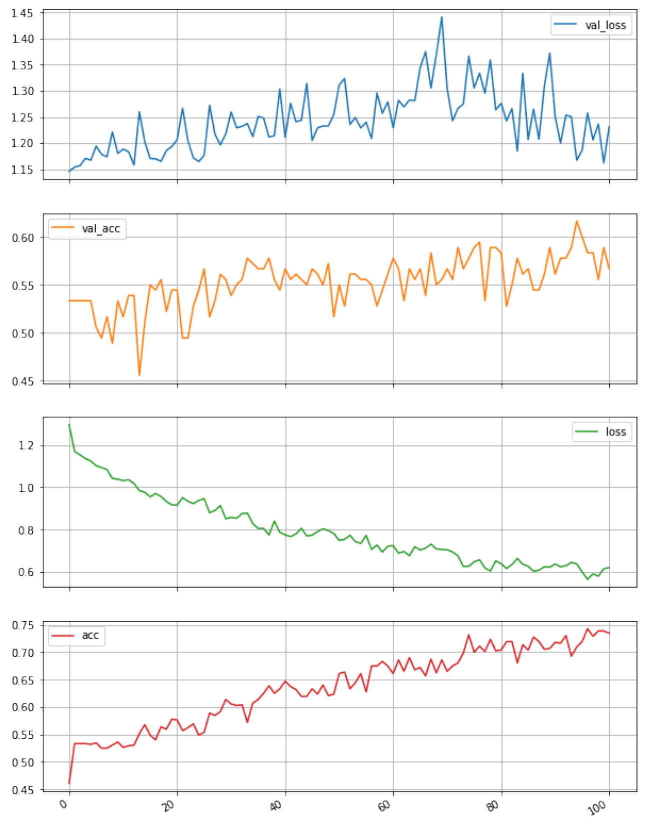
With activation function as sigmoid



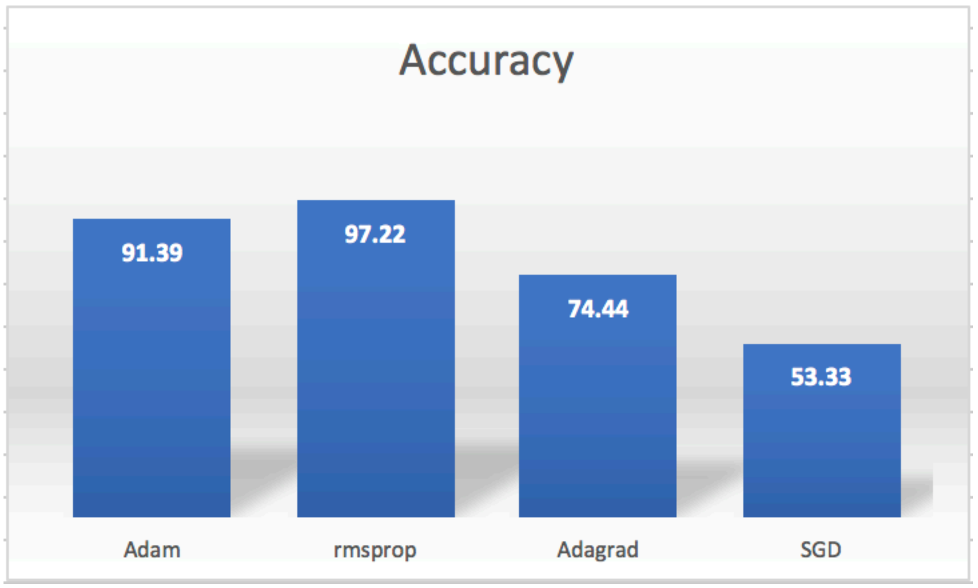
With activation function as relu



With activation function as tanh



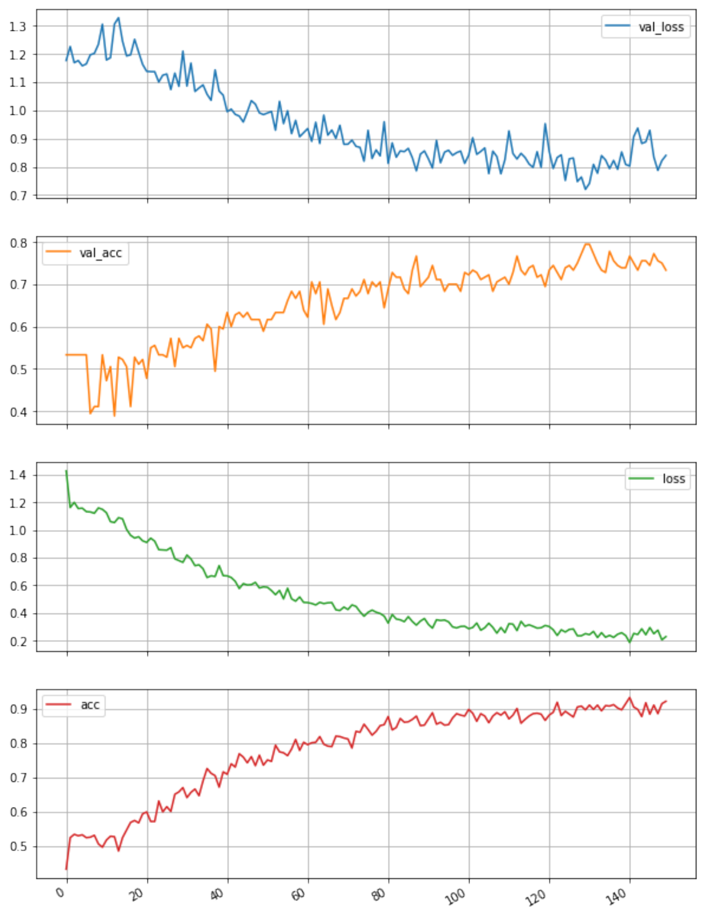
Optimizer:
Let us check whether the change in optimizer has an effect in the accuracy of the model. Keeping the other parameters like batch size, epoch, activation functions, dropout rate, learning rate and changing the optimizer, the accuracies were seen as shown below.



Parameters in my implementation:

input_size = 10
drop_out = 0.2
first_hidden_layer_nodes = 256
second_hidden_layer_nodes = 300
output_layer_nodes = 4
block_size=50
num_epochs=150
optimizer = adam with lr = 0.01
activation functions:
for first hidden layer -> relu
for second hidden layer -> sigmoid
for output layer -> softmax

Graphs for testing data and the validation data:



Conclusion:
It is seen that with the implementation as stated above, the validation data accuracy of around 92% has been achieved.