PROJECT REPORT

Hyperparameters

Hyperparameters are the variables which determines the network structure (e.g.: Number of Hidden Units) and the variables which determine how the network is trained (e.g.: Learning Rate).

Hyperparameters related to Training Algorithm

Batch Size

Batch size defines number of samples that are going to be propagated through the network.

For larger batch sizes, the convergence is slow. From the graph we can see that for smaller batch size the convergence is fast. Typically network trains faster with more number of mini-batches. That's because the weights are updated after propogation of each batch i.e., small batch size means more number of propogations and rapid parameter update. A too small a batch size can converge too soon so an intermediate batch size is generally preferred and by default keras batch size (if unspecified) is set to 32. A similar effect can be seen on validation data as well.

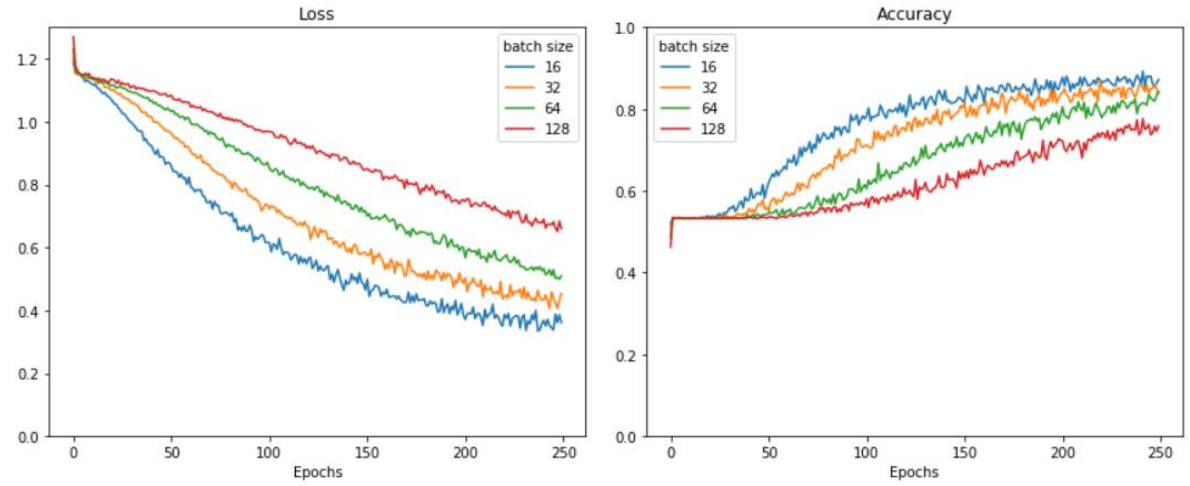

Fig 1: The Loss and Accuracy for different batch sizes on training data

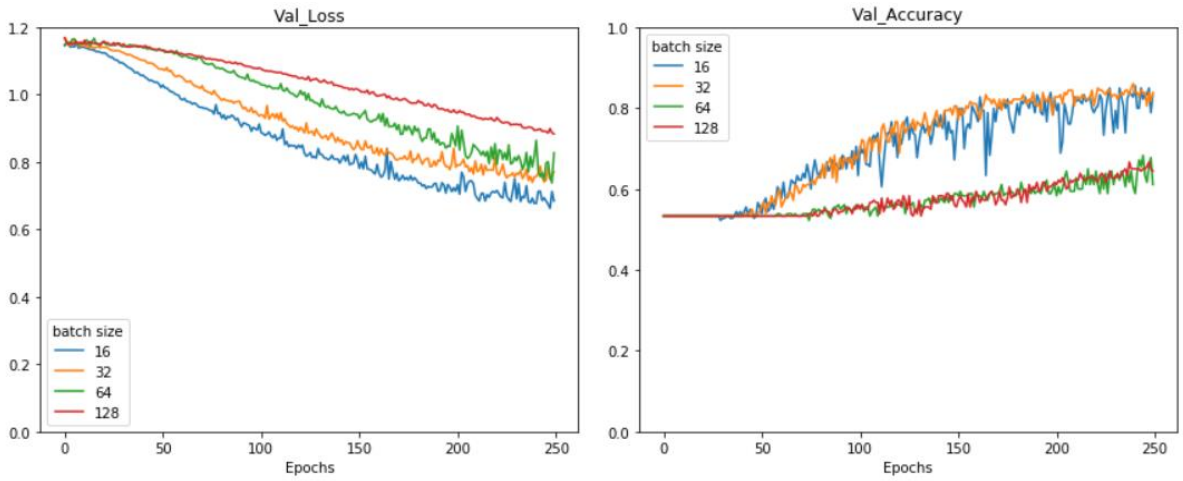

Fig 2: The Loss and Accuracy for different batch sizes on validation data

Learning Rate

The learning rate defines how quickly a network updates its parameters.
Low learning rate slows down the learning process but converges smoothly. Larger learning rate speeds up the learning but may not converge (i.e., we might jump beyond the minimum). We need to tune the learning rate in order for our algorithm to optimally converge to the minimum cost.

In order to show the effect of learning rate, I am going to fix the optimizer to SGD, epochs to 50 and batch size = 16.

The loss and Accuracy on training data for different learning rates can be seen in the table (Fig 4) below.

We can clearly see that with higher learning rate the accuracy starts to converge faster. This can also be seen in the graphs of Loss and Accuracy for different learning rates. As the epochs increase, the model with highest learning rate converges to an accuracy of 1 quickly. But, for learning rate = 0.8, the loss begins to diverge from minima rather than converge. The best learning rate is associated with good drop in loss and smooth convergence to minima.

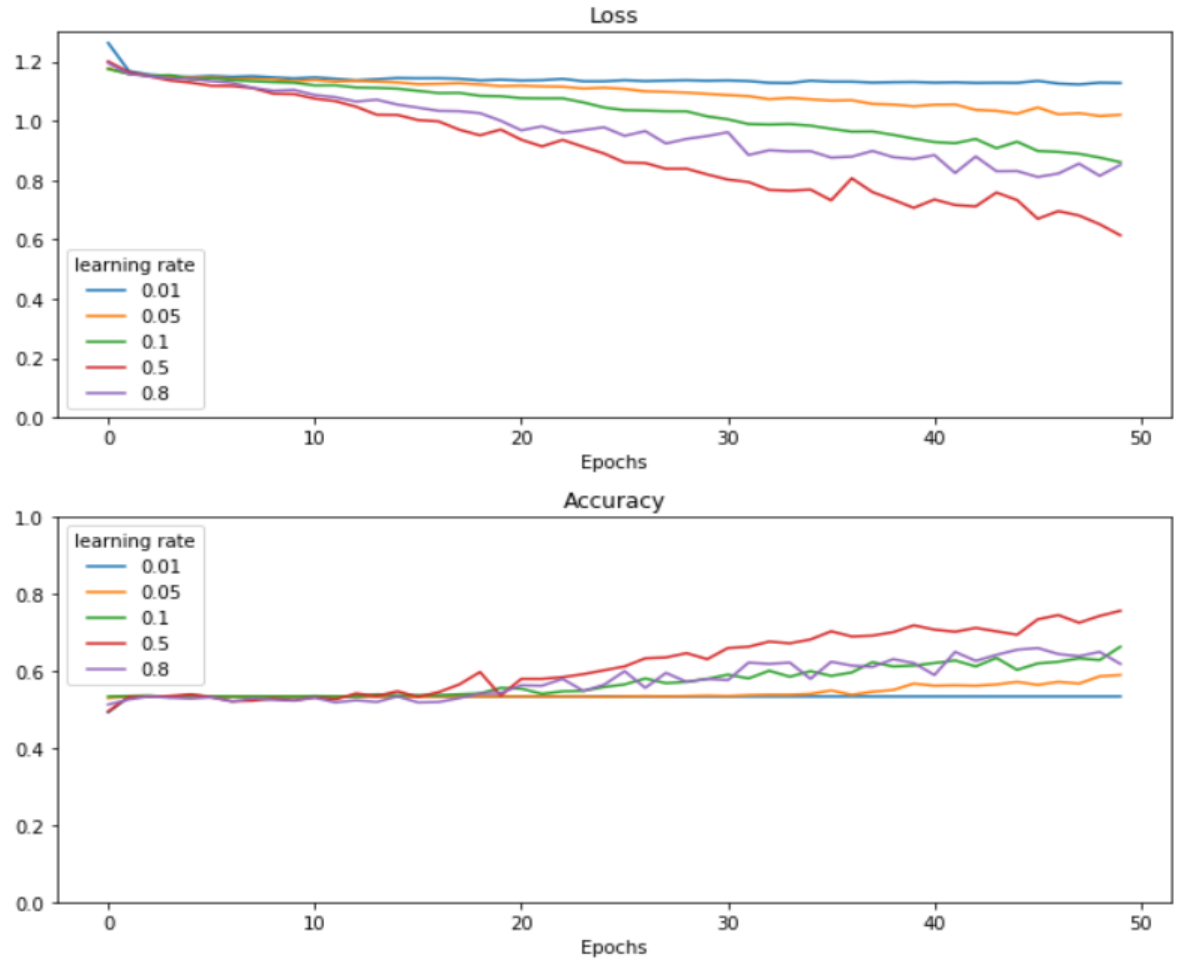

Fig 3: Loss and Accuracy for different learning rates on training data

learning rate	0.01		0.05		0.10		0.50		0.80	
metric	loss	acc	loss	acc	loss	acc	loss	acc	loss	acc
0	1.262989	0.492222	1.177285	0.528889	1.175534	0.533333	1.200888	0.494444	1.195273	0.512222
1	1.168341	0.532222	1.159092	0.533333	1.159231	0.534444	1.164785	0.532222	1.158453	0.525556
2	1.156222	0.533333	1.153539	0.533333	1.152664	0.535556	1.150415	0.532222	1.150819	0.533333
3	1.150988	0.533333	1.151297	0.534444	1.154600	0.531111	1.136463	0.534444	1.143156	0.530000
4	1.148719	0.533333	1.144800	0.533333	1.140335	0.536667	1.129373	0.538889	1.141746	0.527778
5	1.151784	0.533333	1.146861	0.533333	1.144688	0.533333	1.118816	0.531111	1.134261	0.531111
6	1.148911	0.533333	1.142236	0.533333	1.137723	0.533333	1.117936	0.521111	1.128449	0.520000
7	1.151127	0.533333	1.142756	0.533333	1.134256	0.533333	1.110858	0.522222	1.111839	0.527778
8	1.146976	0.533333	1.140943	0.533333	1.131122	0.533333	1.092168	0.527778	1.102017	0.524444
9	1.143588	0.533333	1.134175	0.533333	1.129254	0.533333	1.090712	0.523333	1.104570	0.522222
10	1.146940	0.533333	1.138357	0.533333	1.119994	0.534444	1.075429	0.531111	1.087497	0.530000
11	1.142482	0.533333	1.132902	0.533333	1.120484	0.533333	1.066991	0.525556	1.080219	0.517778
12	1.137159	0.533333	1.135379	0.533333	1.113016	0.533333	1.048115	0.541111	1.065543	0.523333
13	1.141112	0.533333	1.132895	0.533333	1.111703	0.538889	1.021153	0.534444	1.071749	0.518889
14	1.144919	0.533333	1.130107	0.533333	1.109052	0.536667	1.020452	0.546667	1.055433	0.533333
15	1.144061	0.533333	1.124222	0.532222	1.101530	0.536667	1.003129	0.532222	1.045067	0.517778
16	1.144334	0.533333	1.125370	0.533333	1.094314	0.536667	0.998564	0.544444	1.034210	0.518889
17	1.142577	0.533333	1.127417	0.533333	1.094829	0.538889	0.970522	0.564444	1.033014	0.528889
18	1.136870	0.533333	1.123178	0.533333	1.084885	0.541111	0.951663	0.596667	1.026377	0.541111
19	1.139409	0.533333	1.117799	0.533333	1.083649	0.555556	0.971190	0.535556	1.000907	0.542222
20	1.136591	0.533333	1.119336	0.533333	1.077053	0.553333	0.936724	0.578889	0.968550	0.562222
21	1.137667	0.533333	1.116640	0.533333	1.076306	0.540000	0.914233	0.578889	0.982158	0.561111
22	1.142165	0.533333	1.115603	0.533333	1.076379	0.546667	0.936228	0.583333	0.959656	0.578889
23	1.133923	0.533333	1.109445	0.533333	1.062767	0.547778	0.912909	0.591111	0.969364	0.547778
24	1.133905	0.533333	1.112109	0.532222	1.045546	0.557778	0.890358	0.601111	0.978719	0.563333
25	1.137596	0.533333	1.108152	0.533333	1.036671	0.564444	0.859926	0.611111	0.950003	0.598889
26	1.134606	0.533333	1.100025	0.534444	1.035147	0.580000	0.858281	0.632222	0.965934	0.555556

Fig 4: Table representing Loss and Accuracy for different learning rates on training data

Number of epochs

Number of epochs is the number of times the whole training data is shown to the network while training. If the number of epochs is too high, then the model overfits the training data and where as if epochs are less the model will underfit. If there is an overfit of the training data then the validation accuracy will be less. So number of epochs must be carefully chosen to get good accuracy on training as well as validation data.

Hyperparameters related to Network structure

Number of Hidden Layers and units

Hidden layers are the layers between input layer and output layer.
"Very simple. Just keep adding layers until the test error does not improve anymore."
Many hidden units within a layer with regularization techniques can increase accuracy.
Smaller number of units may cause underfitting.

Dropout

Dropout is regularization technique to avoid overfitting (increase the validation accuracy) thus increasing the generalizing power.

From the plots we can conclude that with no dropout, the accuracy increases very quickly because there is no drop of neurons and training data is used as is. As dropout increases and more neurons are dropped randomly, the training set accuracy is less (the increase is also less) and the cross-entropy loss is high. For higher dropout the convergence is slow because it takes more time to learn. The Validation Loss and Validation Accuracy can be explained similarly. So, a dropout in between 0.2 to 0.5 should be a good choice as it provides good balance in convergence and accuracy.

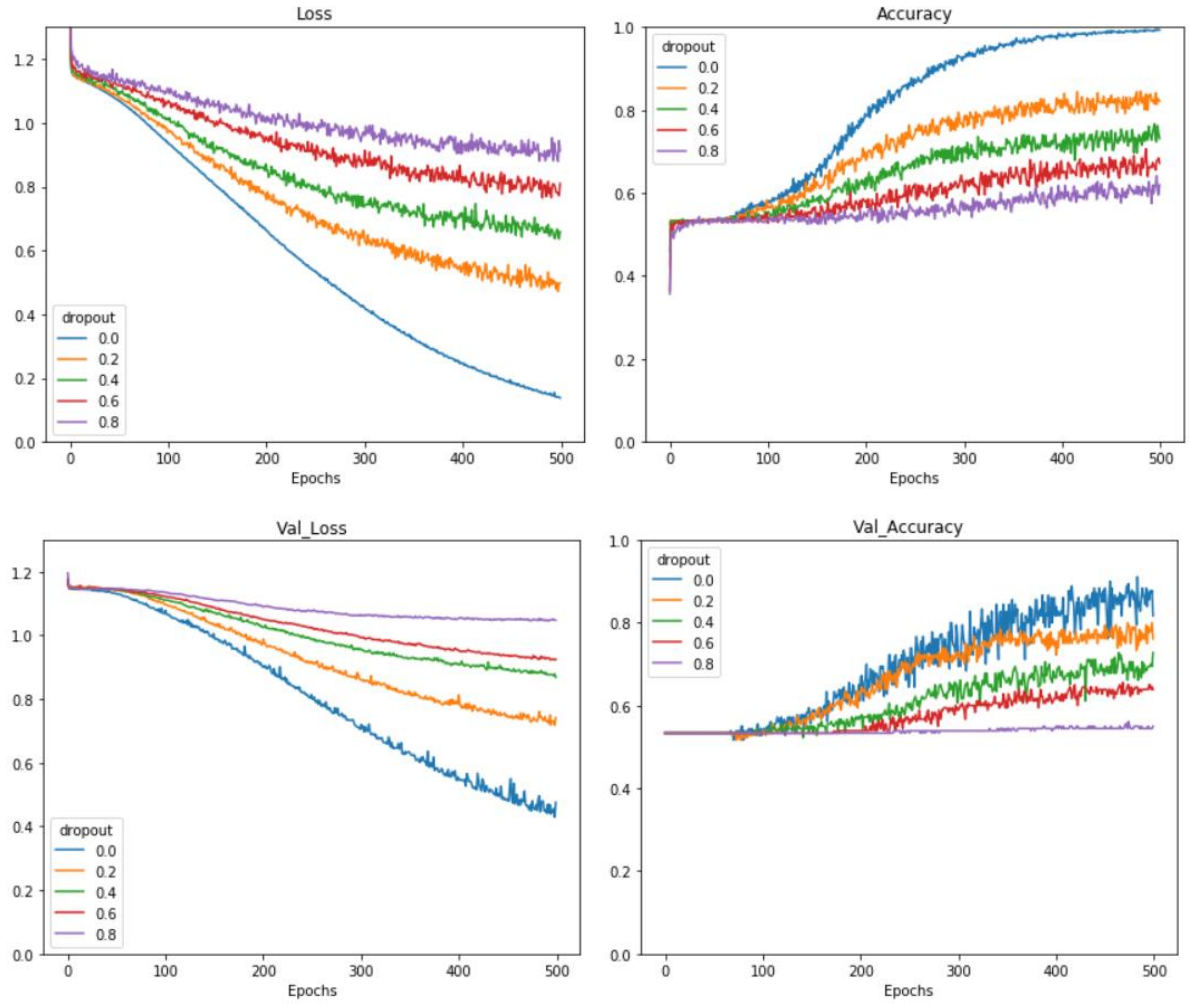

Fig 5: Variations is Loss and Accuracy for different dropout values on Training and Validation Data

Activation Function

They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

Activation functions are used to introduce nonlinearity to models, which allows deep learning models to learn nonlinear prediction boundaries.

Effect of different activation functions in the output layer:

Below are the graphs showing the effect of different activation functions in the hidden layer. The Accuracy is very poor at around 53% for both training data and Validation data for all the activation functions except "relu". As we can see that the Loss and Val_Loss is not improving with epochs a change in epochs won't improve the accuracy/loss. So, only 'relu' activation function is giving us an accuracy of around 91% for training data and 75% for validation data. Due to this reason, 'relu' is selected as the activation function for the hidden layer.

Other hyperparameters: Dropout = 0.2; output layer activation = softmax; optimizer = rmsprop, batch_size = 32, validation_split = 0.2 (nan is NONE or no activation function)

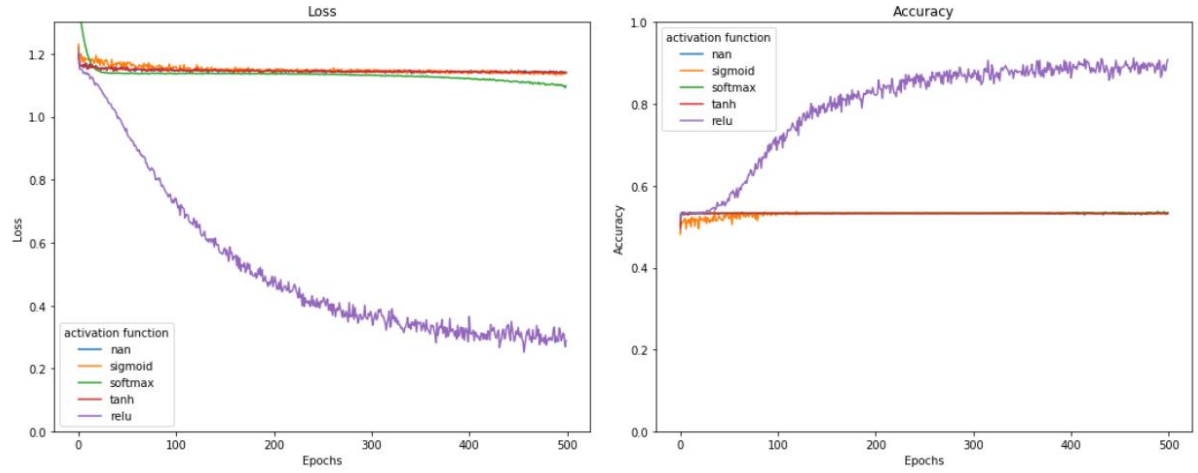

Fig 6: Loss and Accuracy for different activation functions in the hidden layer (Training Data)

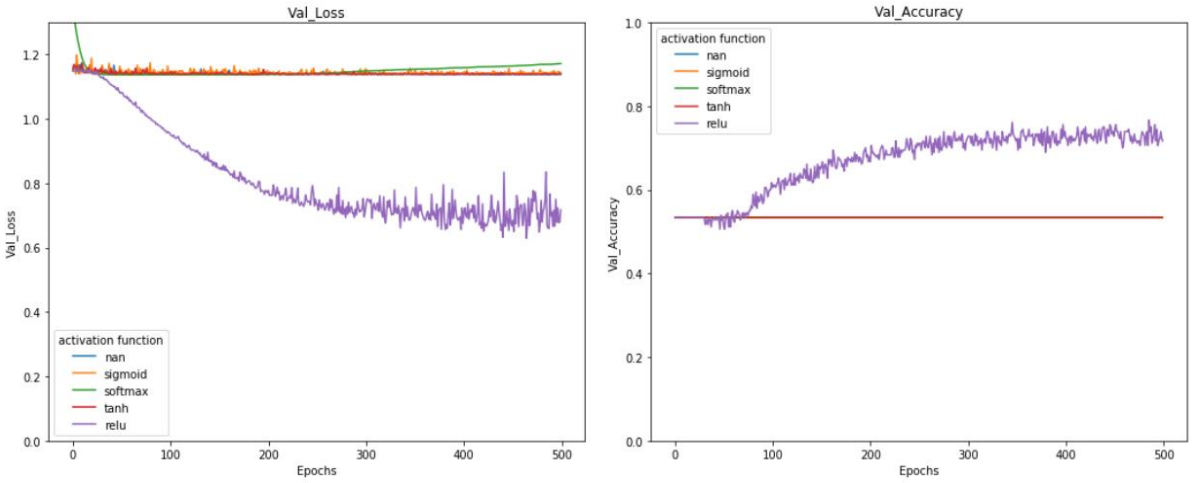

Fig 7: Loss and Accuracy for different activation functions in the hidden layer (Validation Data)

Effect of different activation functions in the output layer:

As we can see from the graphs below, Losses are converging only for Sigmoid and Softmax. Whereas, the losses for other functions (tanh, relu) the loss is above 1.2. Similarly, the accuracy is better for Sigmoid and Softmax whereas the accuracy for tanh and relu are fluctuating a lot and is poor. When we compare Sigmoid and Softmax accuary is better for softmax for both training and validation data. In conclusion, for our model, Softmax is a better activation func tion for the output layer.

Other Hyperparameters: Dropout = 0.2; output layer activation = softmax; optimizer = rmsprop, batch_size = 32, validation_split = 0.2

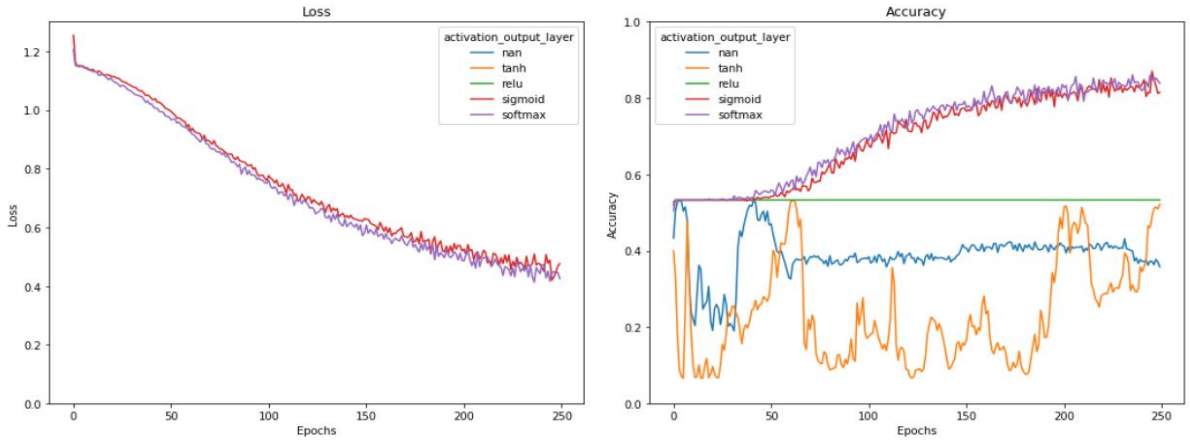

Fig 8: Loss and Accuracy for different activation functions in the output layer (Training Data)

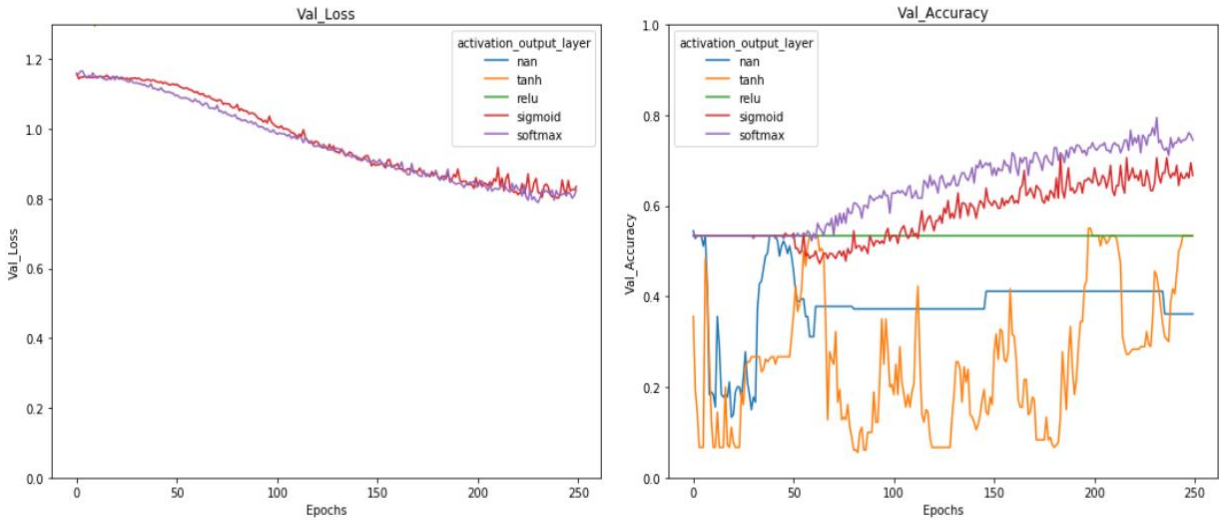

Fig 9: Loss and Accuracy for different activation functions in the output layer (Validation Data)

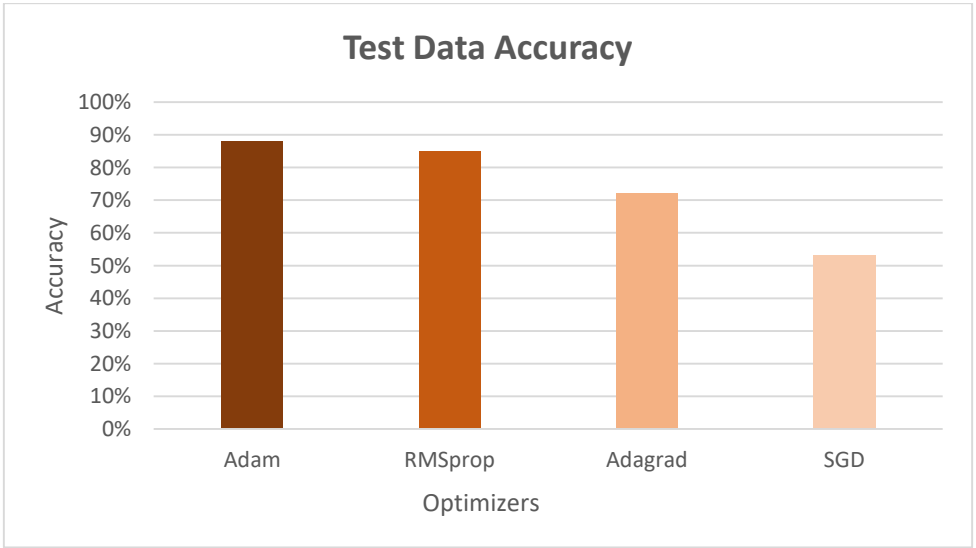

Fig 10: Accuracy for different Optimizers

Optimizer

Different optimizers provide different test data accuracy. As shown in the graph above (Fig 10), Adam optimizer has the highest accuracy.

Final Hyperparameters for our model

Learning rate = 0.001
Dropout = 0.2
Hidden layer activation function = relu
Output layer activation function = Softmax
Optimizer = Adam
Batch Size = 128

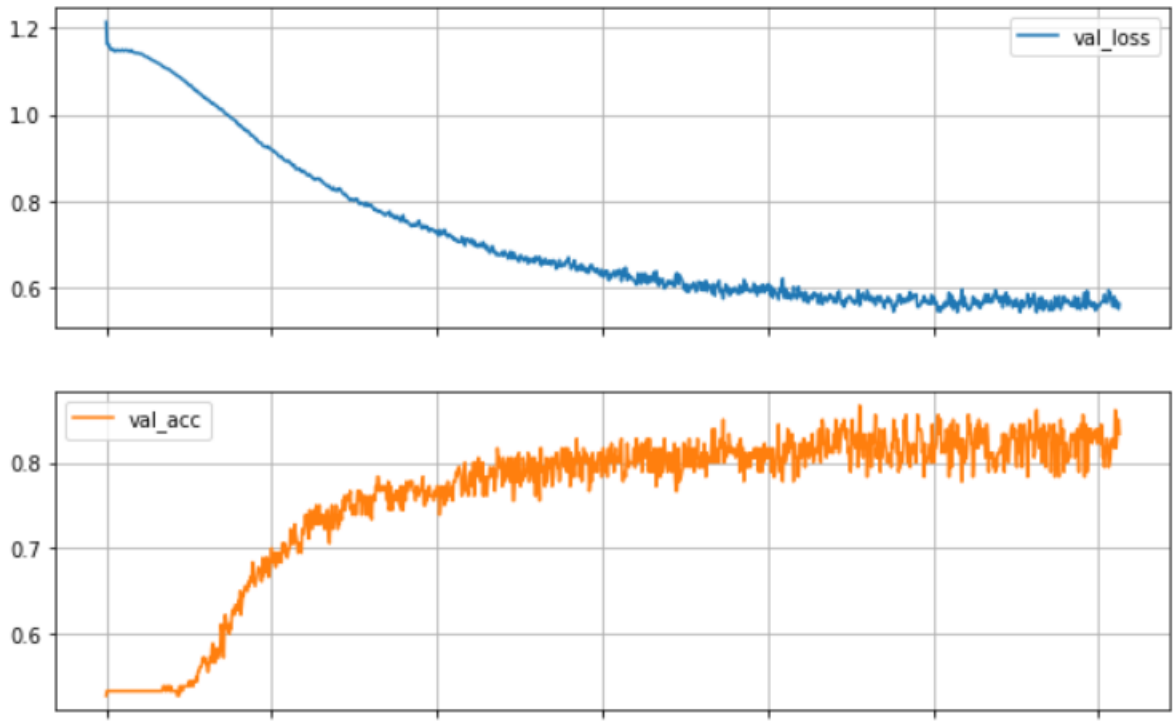

Fig 11: The graphs of loss and accuracy for Validation data

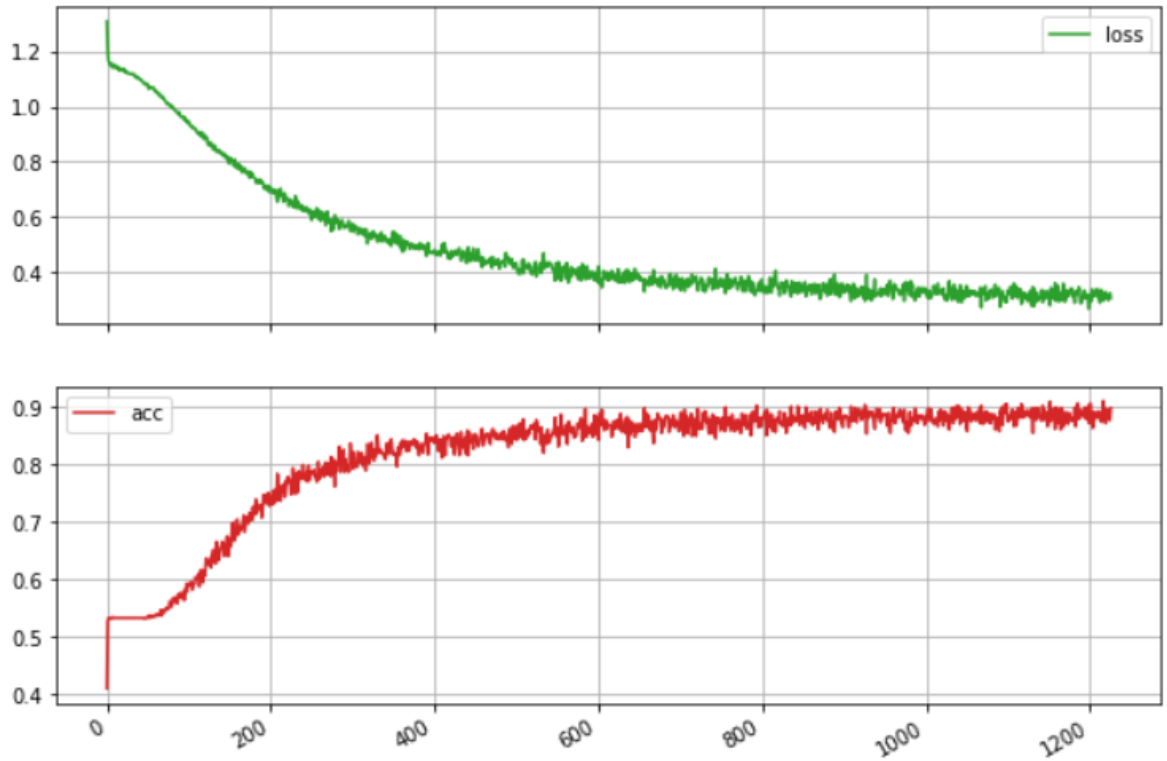

Fig 12: The graphs loss and accuracy for Training data

Callbacks

Early Stopping

The EarlyStopping callback monitors the validation loss and stops as soon as it is not improving. This way we can set epochs to a large number and let the callback decide when to stop.

TensorBoard

A callback to TensorBoard, which includes a suite of visualization tools. One such visualization is the Graph visualization. It helps to debug the code and also to visualize the model.