CSE 574: Introduction to Machine Learning

Project 1.1: Software 1.0 Versus Software 2.0

Anunay Rao
anunayra@buffalo.edu

1 Introduction

The project is to compare the two problem solving approaches to software development: the logic based approach (Software 1.0) and the machine learning approach (Software 2.0). The task of FizzBuzz, which takes an integer input and if the integers is divisible by 3, 5, both 3 and 5, and none then it outputs Fizz, Buzz, FizzBuzz, and other respectively, is taken for the same. Software 1.0 is designed with simple if-else logic in python and will give perfect output. On the other hand, Software 2.0 is designed with machine learning approach. In order to design Software 2.0 we have to design a machine learning model which takes into considerations various hyper-parameters which affects the performance of our program. First, we have to design the model, train the model and then test it on the test data. Keras is used as the machine learning framework for designing the model which uses Tensorflow at the backend. For the purpose of training we will take into account the integers ranging from 101 to 1000 (900 samples). The test data will be the set of integers ranging from 1 to 100 (100 samples). The machine learning model will be composed of one input layer, any number of hidden layer and one output layer. The input layer of the model will have 10 nodes taking 1 or 0 as input. As we will require 10 bits to represent numbers up to 1000 (2^{10} = 1024) in binary form. The model consists of one hidden layer which can contain any number of nodes (256 here). The output layer contains 4 nodes each corresponding to different categories (Fizz, Buzz, FizzBuzz, Other). The activation function used in the first layer is ReLu (Rectified linear unit) and at the output layer is softmax which outputs the values from 0 to 1 which can then be considered as probabilities for each category for the given input. The input integers belong to the category which has highest probability.

2 Basic Design of the Model

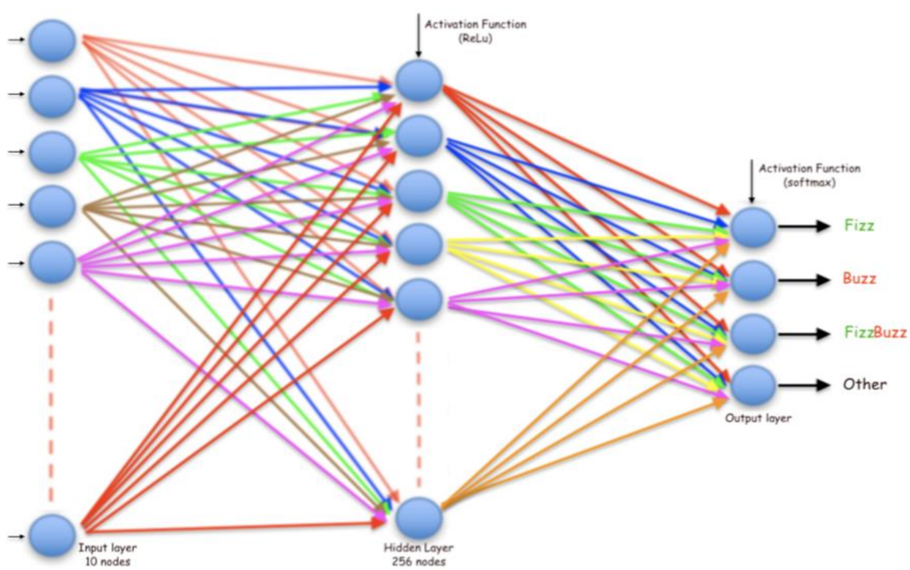

Figure 1: FizzBuzz Neural Network model

3 Effect of various Hyper-parameters

This section presents the experiments conducted by varying the values of Hyper-parameters and how they affect the performance of the program.

3.1 Methodology

Since the model will produce different results for the same network configuration due to randomness in initialization of weights, randomness in training process, random shuffling of data to make batches and randomness due to other factors the methodology adopted to compare the performance of the program is to execute the program for five consecutive times and take the mean of the result keeping the network configuration same in each run.

3.2 Default Model Configuration

Number of Input = 10

Number of Output = 4

Number of nodes in First layer= 256

Dropout = 0.2

Activation function at First layer: ReLU

Optimizer at First layer: RMSprop

Loss function: Categorical Crossentropy

Validation Data Split = 0.2

Number of Epoch = 10000

Model Batch Size = 128

Early Stopping = enabled

3.3 Effect of Validation data split

It represents the percentage of data to be used in order to perform the unbiased evaluation of the model during training. For example, if validation data set = 0.2 then if we have 900 samples then 720 will be used for training and 180 will be used for evaluating the model.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Run1	85.0	84.0	79.0	82.0	69.0	54.0	50.0	54.0	51.0
Run2	92.0	80.0	83.0	79.0	81.0	52.0	53.0	51.0	50.0
Run3	80.0	84.0	83.0	82.0	77.0	54.0	52.0	53.0	51.0
Run4	88.0	82.0	82.0	69.0	76.0	53.0	49.0	53.0	52.0
Run5	87.0	83.0	85.0	81.0	83.0	54.0	52.0	53.0	47.0
Mean	86.4	82.6	82.4	78.6	77.2	53.4	51.2	52.8	50.2

Table 1: Accuracy values in different runs with different Validation data split

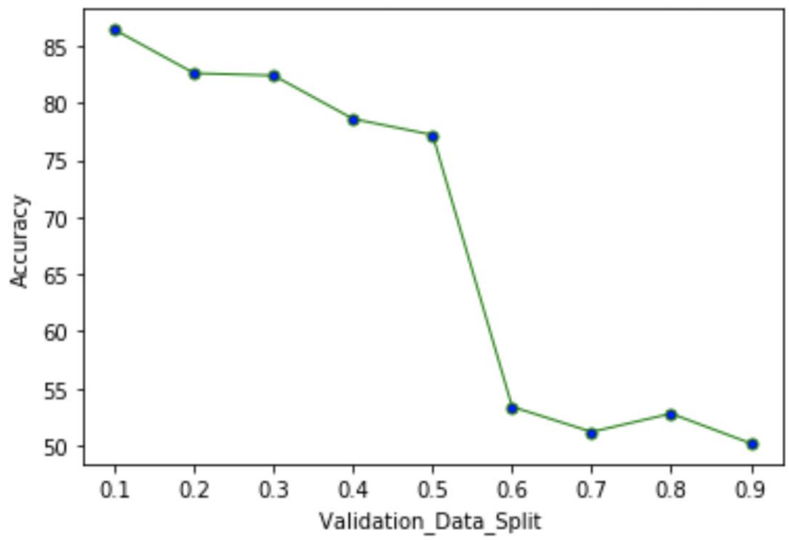

Figure 2: Variation of Accuracy with Validation Data Split

Observation

As the value of validation data split increases the accuracy calculated on the test data decreases because with increasing validation data split the model was trained for less number of samples.

3.4 Effect of Number of Epoch

The epoch consists of forward pass and the backward pass over the entire dataset. So ONE epoch is defined as the single forward pass over the entire data set followed by the corresponding backward pass in which the weights of the nodes are manipulated to get better accuracy.

	200	400	600	800	1000	1200	1400
Run1	59.0	73.0	78.0	92.0	90.0	85.0	77.0
Run2	59.0	72.0	88.0	91.0	77.0	82.0	81.0
Run3	63.0	67.0	84.0	84.0	81.0	78.0	86.0
Run4	65.0	74.0	84.0	84.0	89.0	89.0	84.0
Run5	58.0	84.0	80.0	84.0	78.0	83.0	80.0
Mean	60.8	74.0	82.8	87.0	83.0	83.4	81.6

Table 2:Accuracy values in different runs with different no. of Epoch

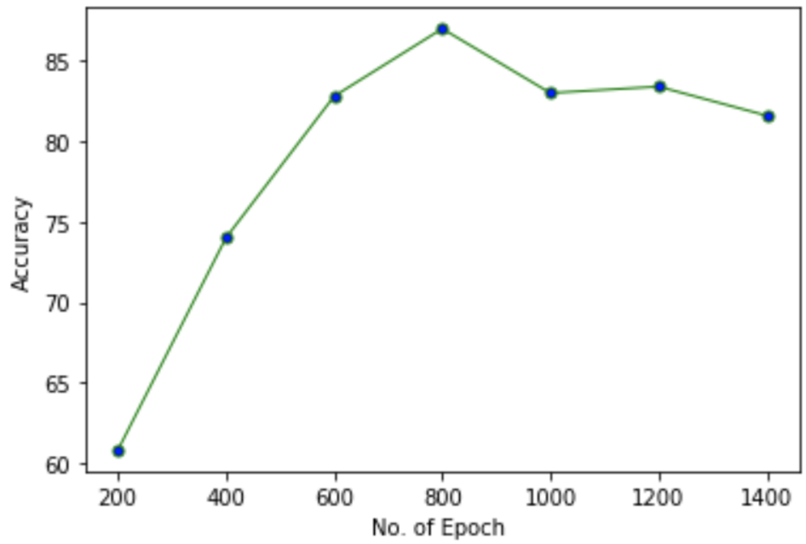

Figure 3: Variation of Accuracy with No. of Epoch

Observation

As the value of epoch increases the accuracy on test data increases and then becomes steady as the early stopping comes into play. Early stopping stops the training if there is no increases in the accuracy or equivalently when the loss cannot be minimized further. Since the entire dataset cannot be passed to the model for training the role of batch size comes into play

3.4 Effect of Batch Size

3.4.1 Accuracy

Since the entire dataset cannot be passed to the model for training the role of batch size comes into play. The batch size defines the number samples to be passed to the model for training and after each batch the weights are updated. Each epoch consists of multiple batches.

	60	90	120	180	360	720
Run1	82.0	82.0	83.0	77.0	85.0	84.0
Run2	82.0	86.0	88.0	81.0	79.0	83.0
Run3	85.0	77.0	86.0	88.0	79.0	77.0
Run4	90.0	86.0	85.0	89.0	88.0	72.0
Run5	84.0	83.0	83.0	91.0	83.0	87.0
Mean	84.6	82.8	85.0	85.2	82.8	80.6

Table 3:Accuracy values in different runs with different Batch size

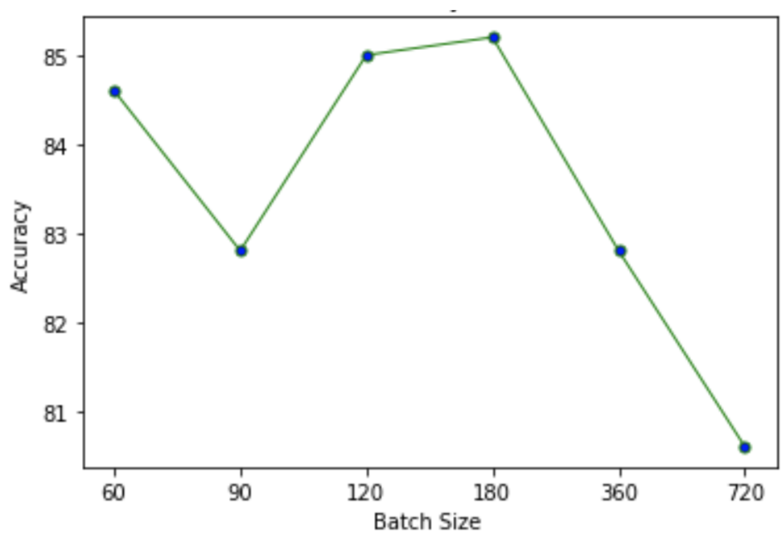

Figure 4: Variation of Accuracy with Batch Size

Observation

As the batch size increases accuracy on test data increases after a dip up to certain level but then starts decreasing. Accuracy is found highest between the batch size 120 and 180. Accuracy varied from 80 to 85%.

3.4.2 Early Stopping

It is used to stop the training of the model if the accuracy stops increasing thereby preventing overfitting.

Batch Size	Early Stopping
60	885
90	950
120	1233
180	1503
360	1635
720	2401

Table 5:Batch Size vs Early Stopping

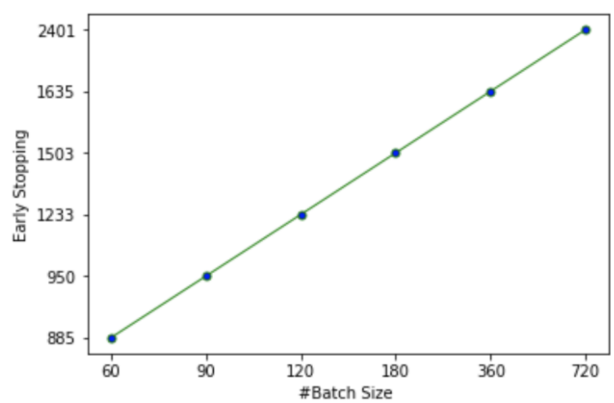

Figure 4: Variation of Early Stopping with Batch Size

Observation

As the batch size increases early stopping also increases as it takes more epoch to adjust the weights to achieve certain level of accuracy on test data. Early stopping helps us to prevent overfitting of the model.

Note: The Accuracy in all these cases are comparable.

3.5 Effect of Dropout

Dropout defines the percentage of nodes or neurons to be dropped or ignored from the layer while training in order to avoid overfitting. Dropout is a regularization technique where neurons are randomly selected to be ignored. The ignored neurons have no contribution in forward pass and their weights are also not updated in the backward pass.

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Run1	90.0	80.0	81.0	87.0	74.0	70.0	62.0	55.0	53.0
Run2	95.0	89.0	83.0	76.0	78.0	76.0	69.0	54.0	52.0
Run3	82.0	85.0	84.0	69.0	70.0	72.0	66.0	54.0	53.0
Run4	91.0	76.0	80.0	86.0	75.0	75.0	66.0	55.0	52.0
Run5	87.0	81.0	71.0	76.0	71.0	64.0	64.0	65.0	53.0
Mean	89.0	82.2	79.8	78.8	73.6	71.4	65.4	56.6	52.6

Table 5: Accuracy values in different runs with different Dropout

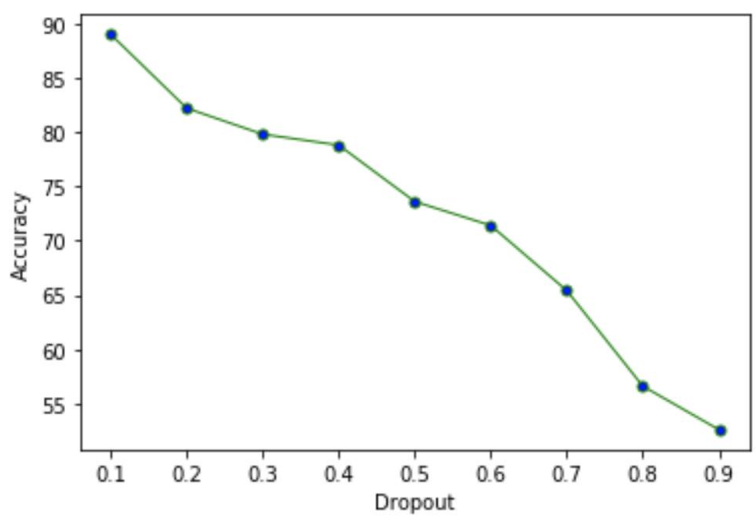

Figure 6: Variation of Accuracy with Dropout

Observation

As the value of dropout increases the accuracy on test data decreases because as the more number of neurons are dropped resulting in less model whose neurons are not properly trained.

3.6 Effect of Number of Nodes in First Hidden Layer

Hidden layer can consist of any number of layers with each containing any number of neurons. Here the hidden layer has one layer with 256 neurons or nodes.

	64	128	256	512	1024	2048
Run1	71.0	77.0	74.0	92.0	95.0	98.0
Run2	64.0	76.0	76.0	88.0	100.0	95.0
Run3	65.0	80.0	86.0	92.0	94.0	96.0
Run4	69.0	81.0	89.0	93.0	96.0	98.0
Run5	64.0	80.0	80.0	92.0	95.0	98.0
Mean	66.6	78.8	81.0	91.4	96.0	97.0

Table 6: Accuracy values in different runs with different no of nodes

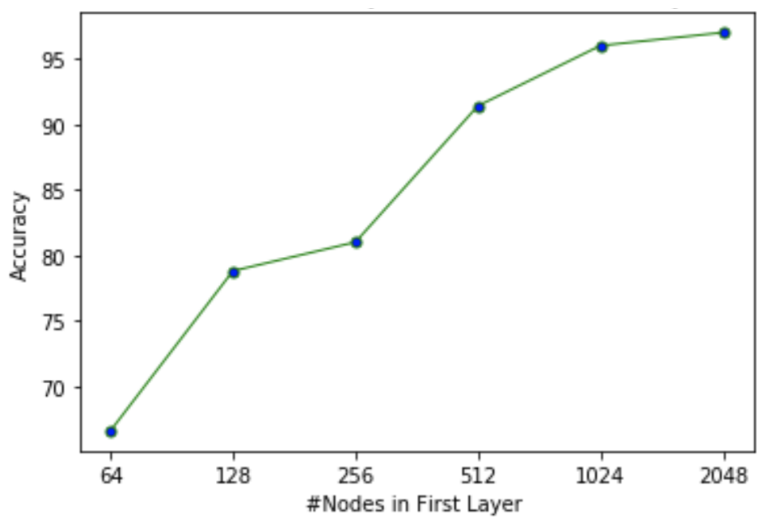

Figure 7: Variation of Accuracy with No. of nodes in First layer

Observation

As the number of nodes in the first layer increases the accuracy on test data increases as the with more number of neuron comes more computing power but then training time increases.

3.7 Effect of Adding an Extra layer as Hidden layer

As stated above, the model can have multiple hidden layers therefore lets checkout the effect of adding an extra layer with different number of neurons. The activation function in this layer is also ReLU.

	32	64	128	256	512
Run1	88.0	93.0	91.0	90.0	90.0
Run2	93.0	91.0	87.0	95.0	96.0
Run3	94.0	94.0	93.0	88.0	88.0
Run4	94.0	91.0	92.0	94.0	89.0
Run5	92.0	93.0	92.0	90.0	90.0
Mean	92.2	92.4	91.0	91.4	90.6

Table 7: Accuracy values in different runs with different no of nodes in Extra layer

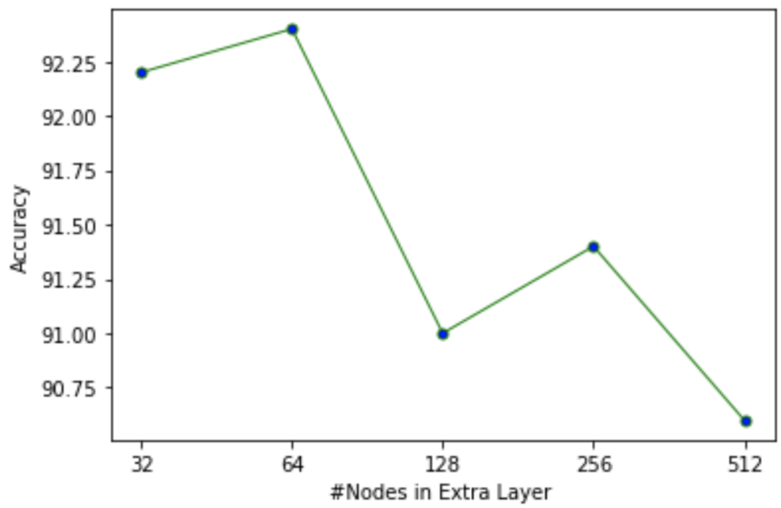

Figure 8: Variation of Accuracy with no. of nodes in Extra layer

Observation

As the number of nodes in the extra layer increases the accuracy on data set varies from 90% to 92%. But we can see the overall increase in the performance but again it comes with the increase in training time.

3.8 Effect of different Optimizers on Training Accuracy, Training loss, Validation Accuracy and Validation loss.

Here we will see the effect of five different optimizers namely, RMSprop, adam, Adagrad, SGD, and Adadelta. The number of Epoch are set to 1500.

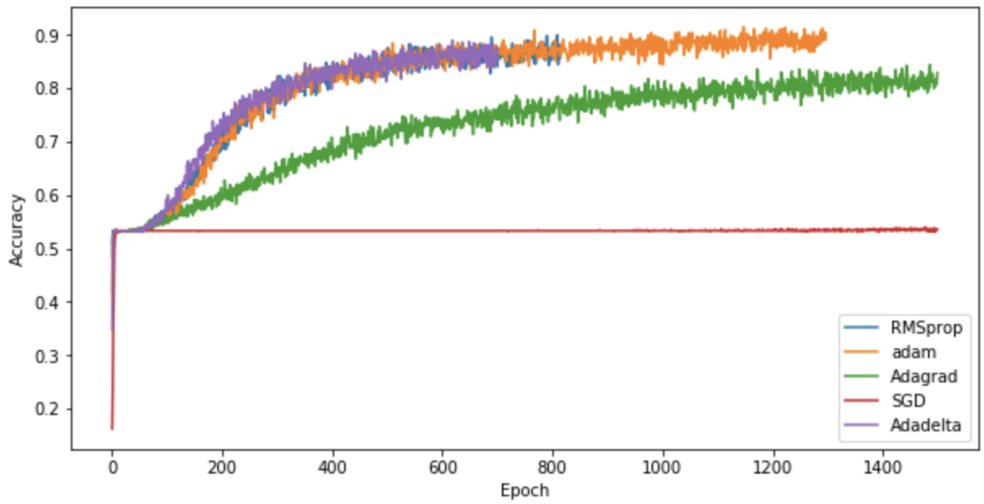

Figure 9: Training Accuracy

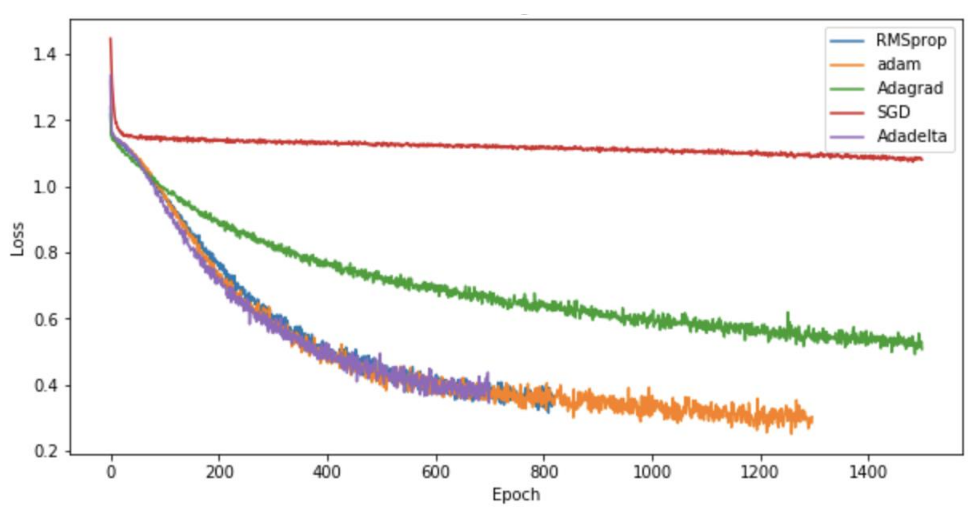

Figure 10: Training Loss

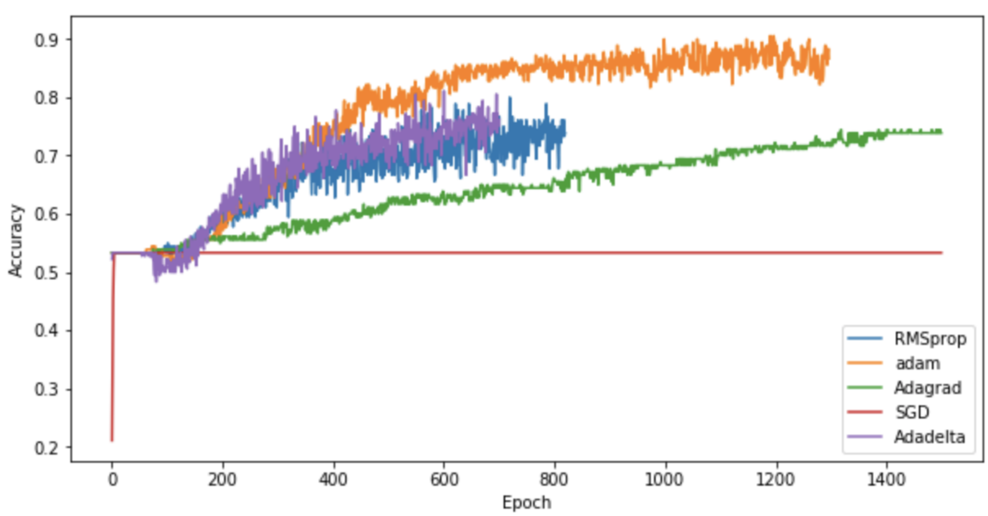

Figure 11: Validation Accuracy

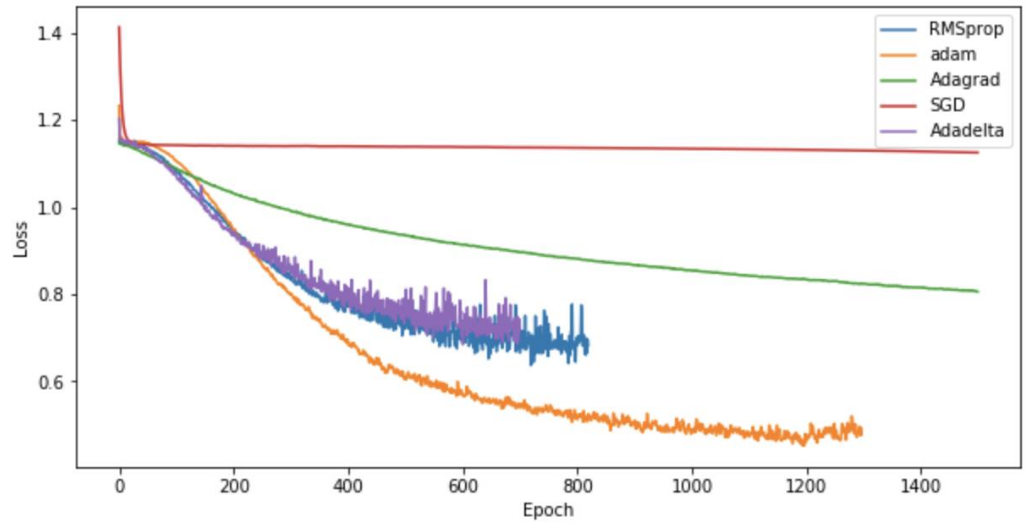

Figure 12: Validation Loss

Observation

As seen from the graph Adam performed best followed by RMSprop, Adadelta, Adagrad and SGD. Adam is known to derive the best properties of Adagrad and RMSprop and is easy to configure as well.

3.6 Effect of Different Activation Functions

The Activation function is used to convert a input of a node in artificial neural network to an output signal which is an input to the next layer. It adds non-linear properties to the network. If we do not use Activation function then the output signal will be a linear function as artificial neural network feeds the sum of product of input with corresponding weights to the Activation function without with it will be a linear polynomial function and has limited power. Let's see the effect of some activation functions namely ReLU, sigmoid, tanh, linear and LeakyReLU(alpha=0.1).

	ReLu	Sigmoid	tanh	Linear	LeakyRelu
Run1	81.0	53.0	53.0	53.0	93.0
Run2	80.0	53.0	53.0	53.0	83.0
Run3	84.0	53.0	53.0	53.0	91.0
Run4	87.0	53.0	53.0	53.0	89.0
Run5	81.0	53.0	53.0	53.0	89.0
Mean	82.6	53.0	53.0	53.0	89.0

Table 8: Accuracy values for different runs for different Activation functions

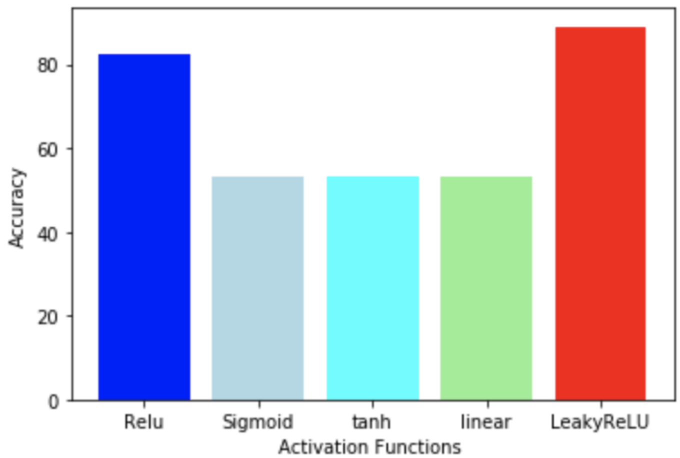

Figure 13: Performance of Activation Function

Observation

As seen from the above bar chart, LeakyReLU performed better than ReLU. Further sigmoid, tanh and linear performed similarly and gave the accuracy 53%. Sigmoid is mainly used for binary classification task.

4 Common Questions Answered!

4.1 What are Hyper-parameters?

Hyper-parameters are the configuration of the model we change to increase the accuracy of the model or to get more skillful predictions. Their values cannot be estimated from the data and has to be set before the learning process begins.

4.2 What is Learning Rate?

Learning rate defines how quickly the model updates its configuration. Low learning rate slows down the learning rate but converges smoothly whereas larger learning rate boosts up the learning process but may not converge.

4.3 What is Softmax?

Softmax is an activation function mainly used in multiple classification problems. The output from the softmax varies from 0 to 1 and the sum of all the output is equal to 1 thus the output from the softmax function can be thought of as a probability that the input will belong to each target class.

References

[1] Keras.io. (2018). Keras Documentation. [online] Available at: https://keras.io/.

[2] Towards Data Science. (2018). Machine Learning – Towards Data Science. [online] Available at: https://towardsdatascience.com/machine-learning/home [Accessed 10 Sep. 2018].

[3] Brownlee, J. (2018). Machine Learning Mastery. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/