Software 1.0 vs Software 2.0

Parth Shah
SEAS, University at Buffalo
Buffalo, NY, 41214
Person Number: 50291125
parthnay@buffalo.edu

Abstract

The aim of this project is to use the logic-based approach (Software 1.0) and the machine learning approach (Software 2.0) for the same problem so as to compare the methodologies of the two. For Software 2.0 i.e. – machine learning approach a simple multi-layer-perceptron with one hidden layer using the python framework keras with tensorflow. The comparison is done on the accuracy of the two methods.

1 Introduction

The given problem is FizzBuzz. In this integer divisible by 3 is printed as Fizz, and integer divisible by 5 is printed as Buzz. An integer divisible by both 3 and 5 is printed as FizzBuzz. If an integer doesn't fall in either of the three categories then it is just printed as it is in Software 1.0 and as other in Software 2.0. The test is to perform the function FizzBuzz on integers 1-100 and generate the output labels.

1.1 Software 1.0

This is the normal or logic based approach to FizzBuzz. Here the input is iterated over one by one and checked for divisibility by 15, 5 and 3 and if they are divisible then print FizzBuzz, Buzz and Fizz and if not divisible then print the input as is. These are written to an output file.

1.2 Software 2.0

This is the machine learning approach to the task. The tools available for this are Machine learning frameworks like tensorflow, PyTorch, Gluon, sklearn, etc. which are used for creating neural network models that can be trained to solve complex problems.[1] In the method taken we use tf.keras which is Tensorflow's implementation of Keras which provides high-level API to build machine learning models and to set up a simple multi-layer perceptron network with one hidden layer. Or a 2 level deep neural network.

We have our test data as 1-100, taking 101-1000 as training data for the model of which we will make partitions to then use them as cross-validation set and others for training. We are using a 2 dense layer neural network, with 256 hidden units and 4 output units both of which are run with different activation function which are provided by Keras along with other hyperparameters like optimizers, batch size, decay, etc.

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).[2] Hyperparameters are set before training(before optimizing the weights and bias).

2 Implementation

Software 1.0 is a straight forward implementation using a nested if-else loop and calculating the divisibility with 15, 5 and 3 using break statements in between.

Software 2.0 takes an input file (test set), a defined model, a test set and outputs an output file with predicted labels. For this part a sequential model neural network was chosen. 2 layers were defined with 256 and 4 hidden units each. Each layer was assigned an activation function and the whole model was defined over an optimizer. The set model was run with parameters such as 10000 epochs and a dropout rate of 0.1. An early stopping was set so as to avoid unnecessary epochs and accuracy was recorded. The graphs were plotted for validation_accuracy, validation_loss, test_accuracy, and test_loss.

3 Experimentation

The given model was defined by a set of hyperprameters such as learning rate, number of epochs, loss function, regularizer, optimizer, etc. For the purpose of this project, some hyperparameters were chosen to observe the change in accuracy of the model for different values of the hyperparameters.

The hyperparameters altered were:

1. Activation Function
2. Optimizer
3. Number of hidden neurons

3.1 Activation Function

The activation function of a node defines the output of that node given an input or set of inputs. The activation function is usually an abstraction representing the rate of action potential firing in the cell. In its simplest form, this function is binary—that is, either the neuron is firing or not. [3] Keras provides various activation functions viz. softmax, elu, relu, tanh, sigmoid, and linear.The various activation functions are defined as

Sigmoid -

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Relu

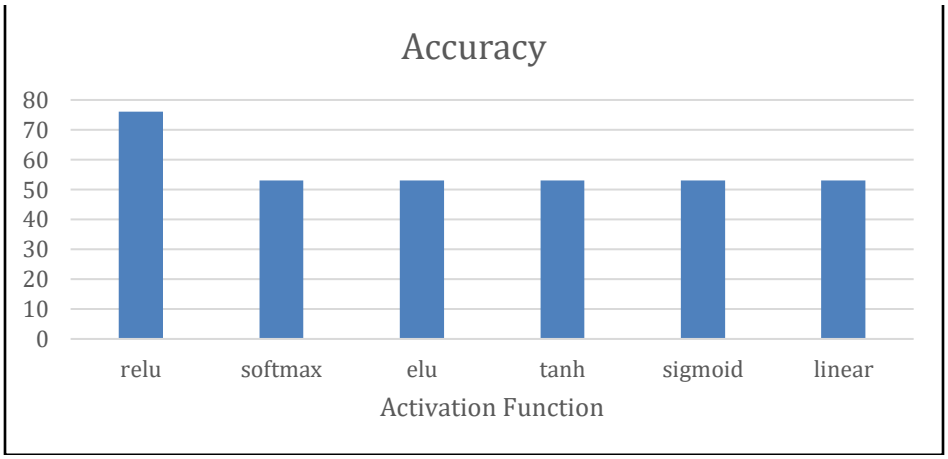
$$f(x) = max(x, 0)$$

Softmax

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

tanh

$$f(x) = \tanh(x)$$

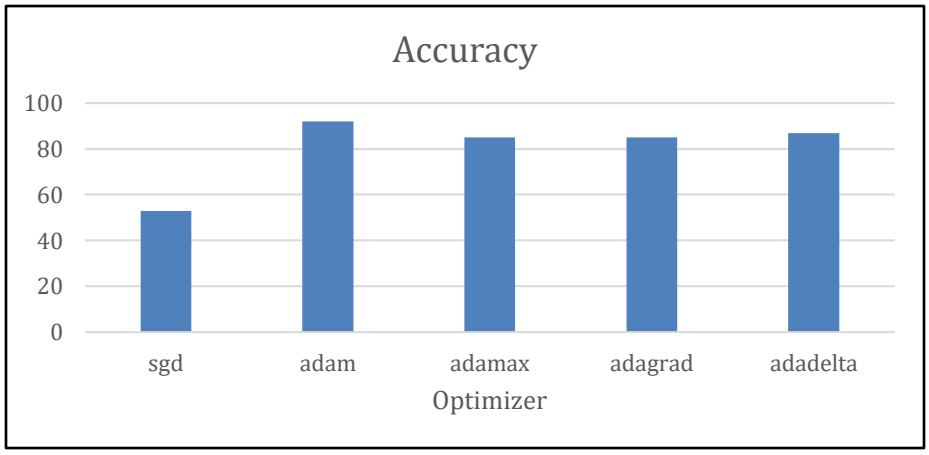


Accuracy vs Activation Functions

We used all the activation functions separately for the dense layer and observed the output accuracy and decide to go with 'relu' as the selected activation function.

3.2 Optimizer

Optimizer finds the weight values of neural network to minimize the objective function [4] Optimization is done to rectify losses by calculating a change in the hyperparameter that will lead to less loss and hence a better accuracy. Keras provides various Optimizers such as sgd, rmsprop, adam, adamax, and adamax. Optimizers can be adaptive or fixed formula based. From the observations on the accuracy using these optimizers.

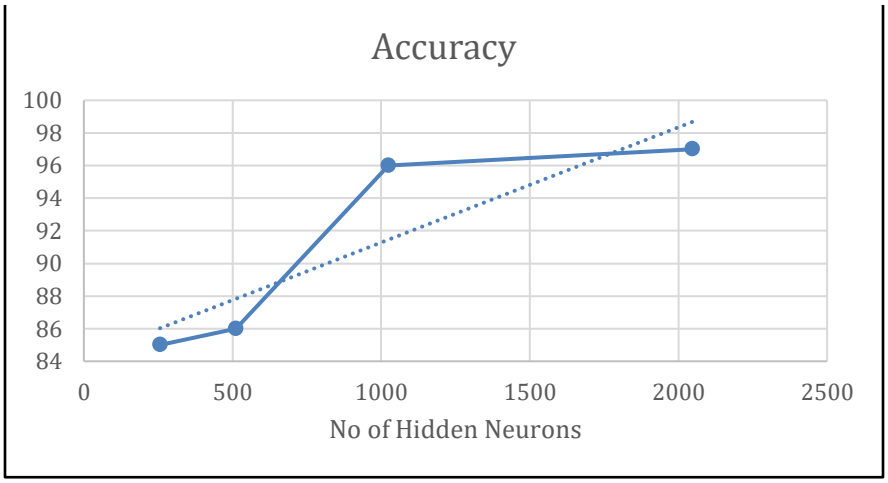


Accuracy vs Optimizers

Adam tends to perform the best among the others and hence we will proceed with adam as optimizer

3.3 Number of hidden layer neurons

In the neural network used for the problem, we defined 2 layers apart from the input layer. They are the hidden dense layer and the output layer. A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function.[5] Altering the number of hidden layer neurons returned the following results which can be graphed as follows -



Accuracy vs No of Neurons

As seen, an upward trend is noticed as the number of hidden layer neurons are increased. However increasing the number of hidden layer neurons increases the calculations and the processing time. Thus we will restrict our hidden neurons to 1024.

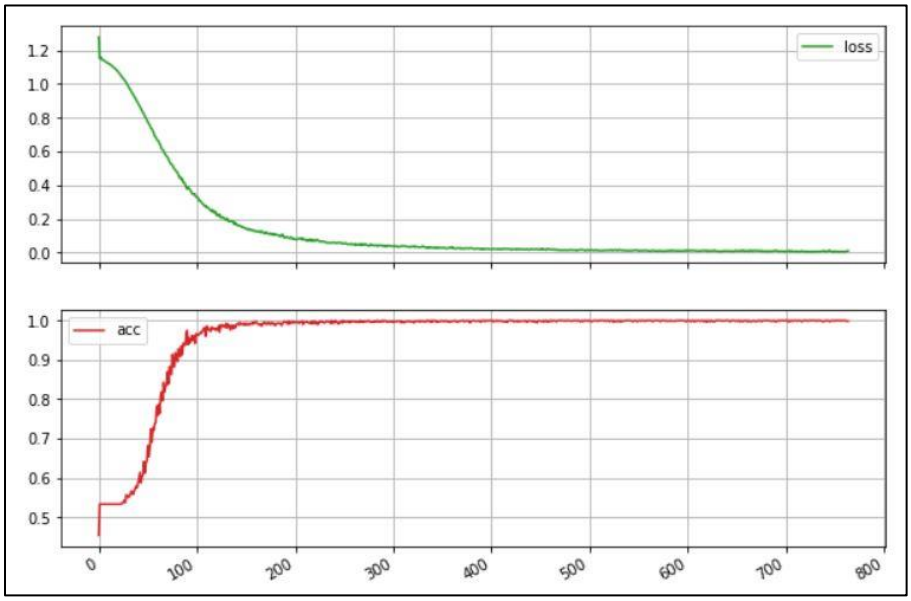
4 Conclusion

So, by altering the hyperparameters a model with accuracy of 97% is generated. This model has the following configuration.

Activation Function for layer 1	'relu'
Optimizer	'adam'
No of Hidden layer neurons	1024
Accuracy	97%

Table 1: Model Configuration

Its accuracy and loss graphs can be seen below.



Loss and Accuracy for final model

Acknowledgments

This report was prepared in accordance to the Project1.1.pdf provided and Keras Documentation[5] available on the website.

References

[1] https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch

[2] https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

[3] https://en.wikipedia.org/wiki/Activation_function

[4] http://courses.mai.liu.se/FU/MAI0083/Report_Mina_Nikanfs.pdf

[5] https://keras.io/