Project 1.1 FizzBuzz Implementation Report

Redwan Ibne Seraj Khan(UBITname: rikhan)
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY-14228
rikhan@buffalo.edu

Abstract

FizzBuzz is a problem where we have to find out the corresponding numbers for which we have to print out Fizz, Buzz, FizzBuzz and Other. If a number is divisible by 3, 5, 3 and 5 and none of the two, we have to print out Fizz, Buzz, FizzBuzz and Other respectively. The following report talks about the implementation of FizzBuzz using a general python approach and a machine learning approach.

1 Logic Based Solution [Software 1.0]

Software 1.0 is a logic based solution of the FizzBuzz problem. In this solution we code explicitly about what we want the program to do. As a result, after running the software it was seen to have an accuracy of 100%. The software was tested against a testing set data which had values from 1 to 100 each labelled correctly. After running the testing dataset with the output of software 1.0, it produced the following results.

```
----------Accuracy of Software 1.0-----------
Errors: 0  Correct :100
Testing Accuracy: 100.0
```

2 Machine Learning Based Solution [Software 1.0]

In this software, a machine learning model was build and dataset were divided into training and testing sets. Afterwards the machine learning model was trained with the training set and later on tested with the testing set. The full dataset was split in 1:9 ratio for making the testing and the training sets. Throughout the training phase, different models were build using different hyper parameters and as a result different results were obtained. A detailed explanation of the results that were obtained after tinkering with the different hyper parameters will be shown below.

2.1 Effect of different dropout rates and its effect on cross entropy loss

The model was tested with different dropout rates having 2 dense layer and using the activation function relu after the first layer and activation softmax after the last layer. The results that were obtained at different dropout rates while training the data and the results obtained finally are shown below. The results have been copy pasted into this report directly after running the code in jupyter notebook.

Dropout rate = 0.1

```
Epoch 924/10000
720/720 [==============================] - 0s 22us/step - loss:
0.1574 - acc: 0.9514 - val_loss: 0.5648 - val_acc: 0.8444
Epoch 00924: early stopping


Errors: 13 Correct :87
Testing Accuracy: 87.0
```

If dropout is applied just before softmax activation,

```
Errors: 6 Correct :94
Testing Accuracy: 94.0
```

Dropout rate = 0.2

```
Epoch 1271/10000
720/720 [==============================] - 0s 22us/step - loss:
0.3213 - acc: 0.8861 - val_loss: 0.5624 - val_acc: 0.8722

Errors: 17 Correct :83
Testing Accuracy: 83.0
```

Dropout rate = 0.3

```
Epoch 850/10000
720/720 [==============================] - 0s 21us/step - loss:
0.4801 - acc: 0.7986 - val_loss: 0.7725 - val_acc: 0.7833
Epoch 00850: early stopping

Errors: 27 Correct :73
Testing Accuracy: 73.0
```

Dropout rate = 0.4

```
Epoch 704/10000
720/720 [==============================] - 0s 22us/step - loss:
0.6060 - acc: 0.7708 - val_loss: 0.8701 - val_acc: 0.6611
Epoch 00704: early stopping

Errors: 20 Correct :80
Testing Accuracy: 80.0
```

Dropout rate = 0.5

```
Epoch 1214/10000
720/720 [==============================] - 0s 43us/step - loss:
0.6400 - acc: 0.7347 - val_loss: 0.8179 - val_acc: 0.7000
Epoch 01214: early stopping

Errors: 20 Correct :80
Testing Accuracy: 80.0
```

Dropout rate = 0.6

```
Epoch 887/10000
720/720 [==============================] - 0s 22us/step - loss:
0.7173 - acc: 0.7236 - val_loss: 0.7998 - val_acc: 0.6556
Epoch 00887: early stopping
Errors: 34  Correct :66
Testing Accuracy: 66.0
```

Dropout rate = 0.7

```
Epoch 715/10000
720/720 [==============================] - 0s 87us/step - loss:
0.8161 - acc: 0.6722 - val_loss: 0.9846 - val_acc: 0.5556
Epoch 00715: early stopping

Errors: 35  Correct :65
Testing Accuracy: 65.0
```

Dropout rate = 0.8

```
Epoch 495/10000
720/720 [==============================] - 0s 43us/step - loss:
0.9078 - acc: 0.6125 - val_loss: 1.0695 - val_acc: 0.5500
Epoch 00495: early stopping

Errors: 41  Correct :59
Testing Accuracy: 59.0
```

If dropout is applied just before softmax performance deteriorates.

```
Errors: 45  Correct :55
Testing Accuracy: 55.0
```

This is because right before the last layer, network has no ability to "correct" errors induced by dropout before the classification happens.

Dropout rate = 0.9

```
Epoch 515/10000
720/720 [==============================] - 0s 43us/step - loss:
1.0394 - acc: 0.5472 - val_loss: 1.1283 - val_acc: 0.5333
Epoch 00515: early stopping

Errors: 47  Correct :53
Testing Accuracy: 53.0
```

Dropout rate = 1

```
Epoch 1157/10000
720/720 [==============================] - 0s 43us/step - loss:
0.0031 - acc: 1.0000 - val_loss: 0.4343 - val_acc: 0.8889
Epoch 01157: early stopping

Errors: 7  Correct :93
Testing Accuracy: 93.0
```

If dropout layer is just before the softmax layer,

```
Errors: 8  Correct :92
Testing Accuracy: 92.0
```

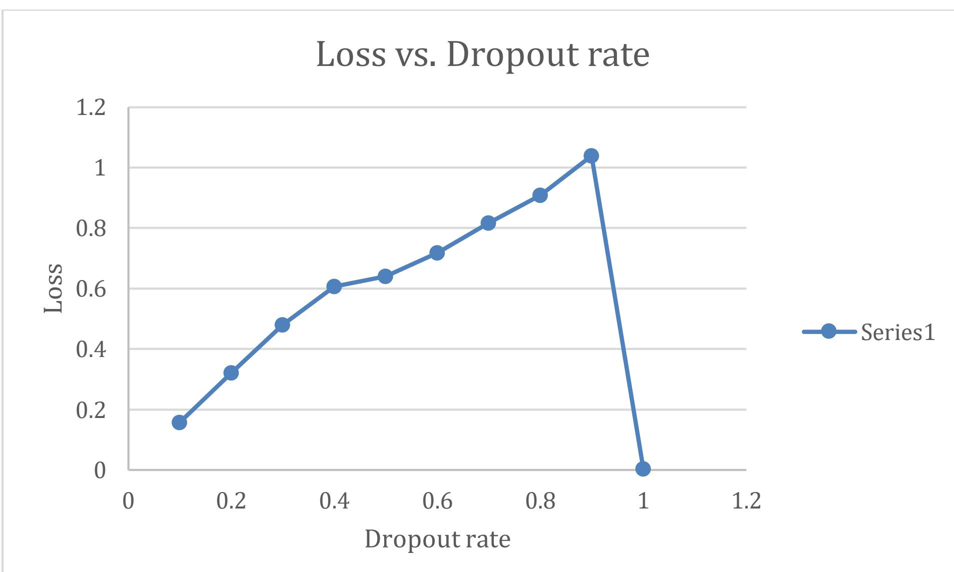
No dropout layer

```
Epoch 911/10000
720/720 [==============================] - 0s 21us/step - loss:
0.0146 - acc: 1.0000 - val_loss: 0.5003 - val_acc: 0.8611
Epoch 00911: early stopping

Errors: 7  Correct :93
Testing Accuracy: 93.0
```

Our network is relatively small compared to the dataset. As a result, regularization to a great extent with higher dropout rates was unnecessary. Hence it hurt the performance when we were increasing the dropout rates. As can be seen, we have a better accuracy when we did not use any dropout layer. Our training time was also limited and training always stopped early. As we were not training until convergence, we should not have used a large dropout rate.

The results obtained can be plotted into a graph



2.2 Effect of different number of layers in the network

The model was tested with a dropout rate of 0.2 having different number of dense layers and using the activation function relu after the first layer and activation softmax after the last layer. The results that were obtained finally at different number of dense layers have been shown below. The results have been copy pasted into this report directly after running the code in jupyter notebook.

2 dense layers. Performance is seen to be 87%.

```
Errors: 13 Correct :87
Testing Accuracy: 87.0
```

3 dense layers. Performance improved.

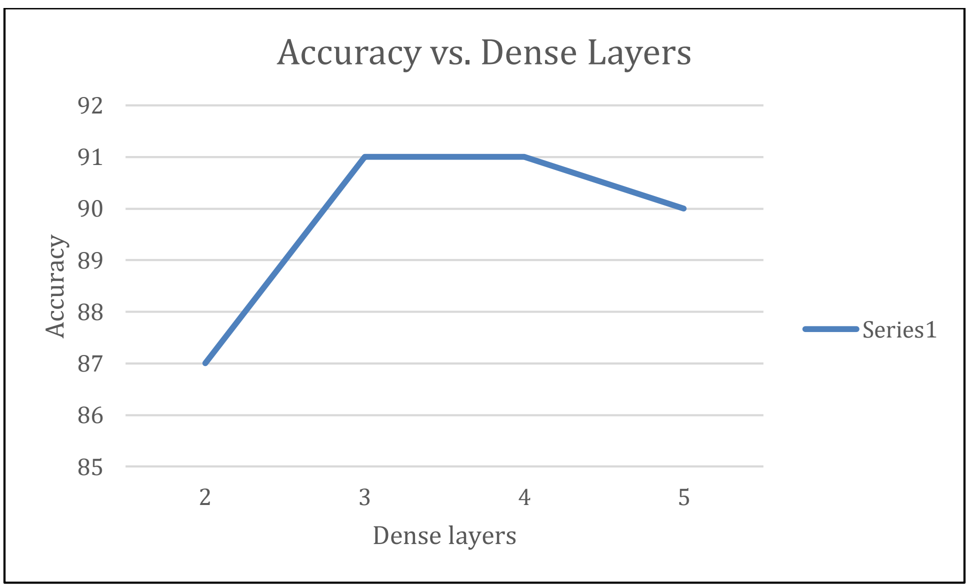
```
Errors: 9 Correct :91
Testing Accuracy: 91.0
```

4 dense layers. Performance same.

```
Errors: 9 Correct :91
Testing Accuracy: 91.0
```

5 dense layers. Performance almost the same.

```
Errors: 10 Correct :90
Testing Accuracy: 90.0
```



The performance improved after adding an extra third layer. This shows that adding an extra layer creates space for more computations and thus giving a higher accuracy. But after a particular number of layers the performance remains the same rather than improving. That showed that if we just keep on increasing layers, that would not help. We also need to think about adding activation functions and adding dropout layers with different dropout rates. There are also several optimizations that could be tried out to improve the performance of our model.

2.3 Effect of different number of nodes in each layer

After testing the model with different layers, the model was tested with different nodes in each layer to witness the effect of number of nodes that might affect the performance of a model. At first the number of nodes in the first dense layer was taken to be 500 instead of the initial 256 nodes. The results obtained have been pasted directly from the output of running the code in jupyter notebook.

First_dense_layer_nodes = 500 and droprate=0.2

```
Epoch 840/10000
720/720 [==============================] - 0s 21us/step - loss:
0.0011 - acc: 1.0000 - val_loss: 0.2143 - val_acc: 0.9222
Epoch 00840: early stopping

Errors: 4  Correct :96
Testing Accuracy: 96.0
```

First_dense_layer_nodes = 1000 and droprate=0.2

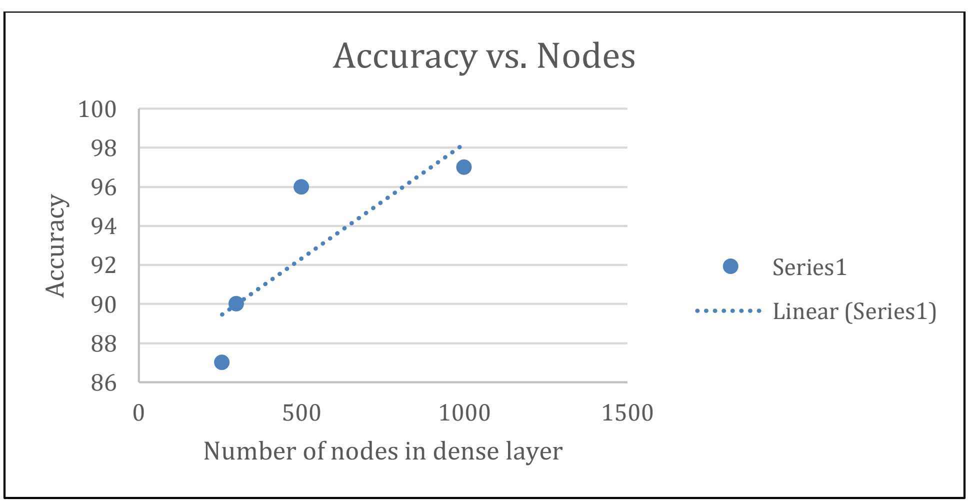
```
Epoch 534/10000
720/720 [==============================] - 0s 69us/step - loss:
0.0017 - acc: 1.0000 - val_loss: 0.2725 - val_acc: 0.9000
Epoch 00534: early stopping

Errors: 3  Correct :97
Testing Accuracy: 97.0
```

First_dense_layer_nodes = 1000 and droprate=0.1

```
Epoch 452/10000
720/720 [==============================] - 0s 20us/step - loss:
6.9301e-04 - acc: 1.0000 - val_loss: 0.0835 - val_acc: 0.9722
Epoch 00452: early stopping

Errors: 0  Correct :100
Testing Accuracy: 100.0
```



It was seen that increasing the number of nodes in the first dense layer improved the performance. Tweaking the dropout rate to be 0.1 further improved the performance to 100%.

2.4 Effect of different optimizations methods

After testing the model with different nodes, layer and drop rates the model was tested with different optimizations. To test different optimizations two dense layers were used. The first dense layer had 1000 nodes and the second dense layer had 10 nodes. Activation 'relu' was applied after the first dense layer node and activation softmax was applied after the third dense layer node. A dropout rate of 0.1 was taken.

The results that were obtained with the different optimizations when the model was designed in the way above is shown below. The results have been directly pasted after running the model in jupyter notebook.

Optimization Adagrad

```
Errors: 1  Correct :99
Testing Accuracy: 99.0
```

Optimization Adadelta

```
Errors: 0  Correct :100
Testing Accuracy: 100.0
```

Optimization Rmsprop

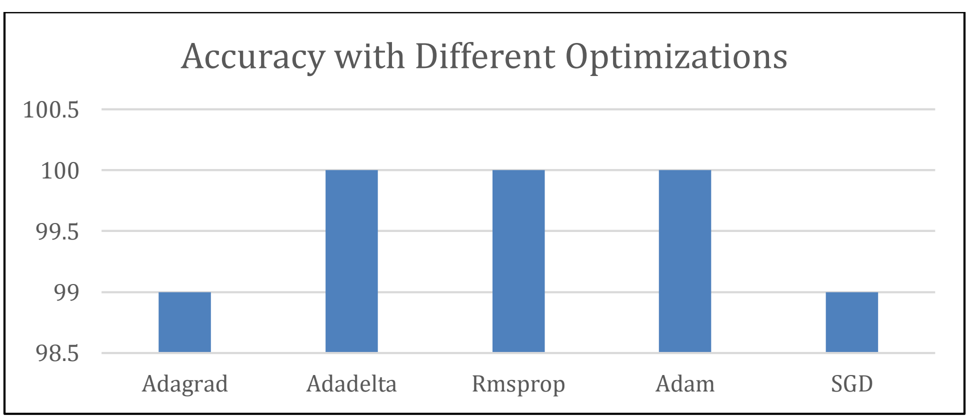
```
Errors: 0  Correct :100
Testing Accuracy: 100.0
```

Optimization Adam

```
Errors: 0  Correct :100
Testing Accuracy: 100.0
```

Optimization SGD

```
Errors: 1  Correct :99
Testing Accuracy: 99.0
```



It was seen that out of all the optimizations, Adadelta, Rmsprop and Adam performed the best. However, it was found that if the dense layers, number of nodes and drop rates were fixed in an efficient way, almost all of the optimizations gave relatively the same accuracy.

2.5 Effect of Different Activation Functions

The same model that was shown in the effect of different optimizations, was now tested with different activation functions. The results obtained have been directly pasted after running the model in jupyter notebook.

Relu after the first dense layer and softmax after the third dense layer

```
Epoch 4423/10000
720/720 [==============================] - 0s 79us/step - loss: 
0.0306 - acc: 0.9986 - val_loss: 0.1400 - val_acc: 0.9500
Epoch 04423: early stopping

Errors: 0  Correct :100
Testing Accuracy: 100.0
```

Tanh after the first dense layer and softmax after the third dense layer

```
Epoch 523/10000
720/720 [==============================] - 0s 35us/step - loss: 
1.1425 - acc: 0.5333 - val_loss: 1.1376 - val_acc: 0.5333
Epoch 00523: early stopping

Errors: 47  Correct :53
Testing Accuracy: 53.0
```

Sigmoid after the first dense layer and softmax after the third dense layer

```
Epoch 506/10000
720/720 [==============================] - 0s 24us/step - loss: 
1.1530 - acc: 0.5319 - val_loss: 1.1397 - val_acc: 0.5333
Epoch 00506: early stopping

Errors: 47  Correct :53
Testing Accuracy: 53.0
```

It was seen that Relu and tanh performed really well with our current dataset. Using sigmoid produced bad results.

3 Network Setting for the Best Performance [Software 2.0]

There are some network settings from which the best performance was obtained. The main important step was to figure out how many dense layers to use, what should be the number of nodes in each layer and the drop rate. Careful selection of activation functions and optimizations also improved the performance to a great deal when ideal number of dense layers and number of nodes were not used. The different parameters of the model defined for those settings can be shown as below. All of the models along with the alternatives shown below produced an accuracy of 100%. The input size was taken to be 10 and dropout rate was 0.1 or 0.2. The first dense layer had 1000 nodes, second dense layer had 10 nodes and the third dense layer(output layer) had 4 nodes. The best model was taken to be sequential. After adding the first dense layer nodes, activation relu/rmsprop/adadelta/adam function was added. Then the second dense layer was added. Following that a dropout layer with a dropout rate of 0.1 or 0.2 was taken. After that another dense layer was added which is actually the output layer with 4 nodes. After the output layer softmax activation function was added.