
CSE 574 : Introduction to Machine Learning (Fall 2018)

Project 1.1: Software 1.0 versus Software 2.0

Name: Tushar Vaidya

UBID: tvaidya

UB Person #: 50289665

Introduction

The report talks about the Machine Learning approach for performing the **FizzBuzz** task. The FizzBuzz task takes the integer number as a input and outputs the category to which the number belongs. If the number is divisible by 3 the category is Fizz, if it is divisible by 5 the category is Buzz, if it is divisible by both 3 and 5 the category is FizzBuzz and the remaining category is Other. The training data that we will be using consists of numbers from 101 to 1000 with labels to which category they belong whereas the testing data consists of numbers from 1 to 100. Our main focus is to improve the prediction of the category when testing data is presented to the model by tuning various hyper parameters that are present

Background

Before jumping into the implementation I think it is very important to understand the different terms used in the model :

a. Activation function : Activation function basically acts as our decider function. When we calculate the weighted sums of a neuron's input the range of this output is from –infinity to +infinity. Given such a huge range it becomes difficult to decide which neuron to activate and which one not to. Applying an Activation function helps us to decide on this. There are various activation functions available such as :

- **Sigmoid** function which cuts the ouput range to [0,1].
- **Tanh** function which cuts the output range to [-1,1].
- **Rectified Linear Unit** (**ReLU)** is an activation function (f(x)=max(0,x)) which maps all the negative values to zero and if it is positive it maps it to the value

itself. ReLU is a popular activation function since it overcomes the vanishing gradient problem seen in the sigmoid and tanh since they map the large input range to a very small one.

b. Optimizer functions **:** While training the model the weights are randomly initialized and the goal of training is to adjust these weights such that we get the best prediction. After one iteration is done we get an error in the prediction which needs to be back propagated to the model to adjust the weights and minimize the error. Optimization function helps to minimize the error by calculating a gradient(partial derivative of the loss function w.r.t the weights) to adjust the initial weights. Applying optimizer functions helps us reach to the global minima which is our target to get the best prediction. How accurately we train our model and how much time it requires to reach the local minima depends majorly on which optimizer function we use. There are various optimizer functions available such as RMSprop, Adam, Adagrad etc. Adams combines RMSprop with momentum which is the knowledge where we should be heading in order to reach the minima.
c. Dropout : Dropout is required to avoid the problem of overfitting of data which occurs when the model gets familiar with the training data and learns it too well because of which it does not predict as expected when a new data is presented to the model. Dropout simply drops few neurons (which depends on the dropout rate we give, which is a fraction from 0 to 1 exclusive) from the network temporarily.
d. Early Stopping : Early stopping is a function used when training our model. In simple words it monitors a value and stops training if the value stops improving. The values we provide to the function's arguments determines how it behaves. We have to pass the value that needs to be monitored during the training phase and the mode which tells whether the value being monitored should decrease or increase ideally.
e. Learning Rate : Learning rate is a hyper parameter which can be thought of as how long step should be while heading towards the minima. It is very important to tune learning rate to a perfect value. If it is set too big, then we might overshoot and miss the minima and if it is set too small, then it might take too long to reach the minima.

Implementation

We are using the **Backpropagation** algorithm to train our model. What this means is that initially we are selecting the weights randomly and after each iteration we adjust the weights so that we can get better predictions. The implementation has an input layer, a hidden layer and one output layer. Given the task where we need to predict the category to which the input integer belongs, we have used **categorical crossentropy** as our loss function. The optimizer function that we have used is Adam. **Figure 1 and 2** shows the

graphs for val_loss and val_accuracy for the implementation having **RMSprop** as the optimizer funcation and the implementation having **Adam** respectively. The early stopping in case of Adam is called after 695 epochs and the testing accuracy was 96.00 % whereas in case of RMSprop the early stopping is called after 940 epochs which clearly shows that RMSprop requires more epochs to train the same dataset (Refer 2b). Also the testing accuracy with RMSprop is 94.00%. The Early stopping function we have implemented monitors val_loss with mode as "min" and patience has value 100 epochs.

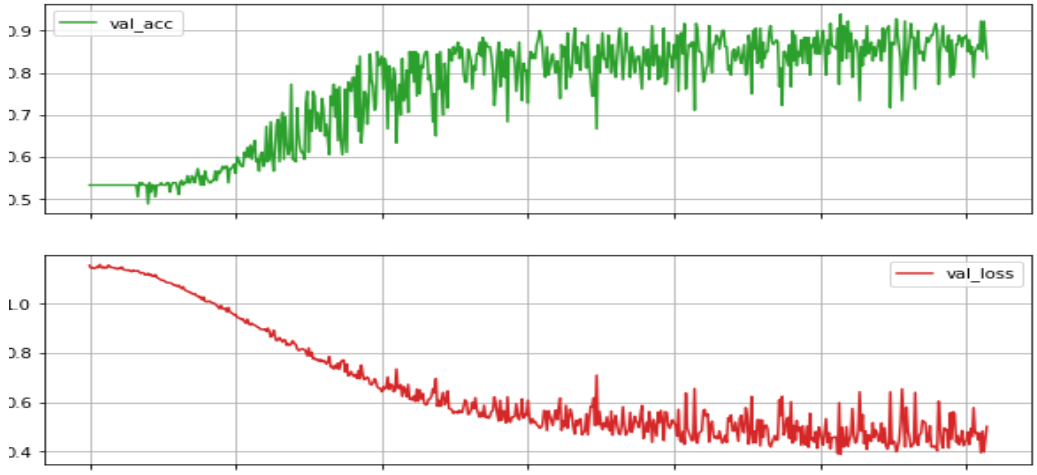

Figure 1

(x=Number of Epochs y=val_accuracy/val_loss)

RMSprop Optimizer Function

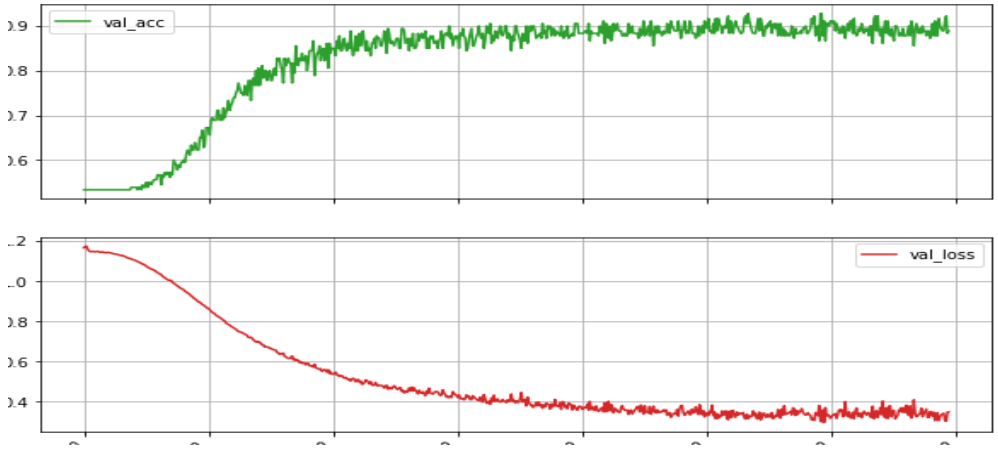

Figure 1

(x=Number of Epochs y=val_accuracy/val_loss)

Adam Optimizer Function

We have also used **ReLU** as an activation function for the hidden layer **softmax** for the output layer. We have also introduced Dropout in our implementation in order to avoid the over fitting of data. Dropout rate gives us the fraction of input units which are selected randomly and are dropped from the network temporarily. Thus we can clearly infer that changing the Dropout rate will change the loss and val_loss (loss seen on the validation data which is a part of training data itself) values. In order to understand the effect of dropout rate on val_loss find below the graph (Figure 3) which gives various val_loss values for different Dropout rates.

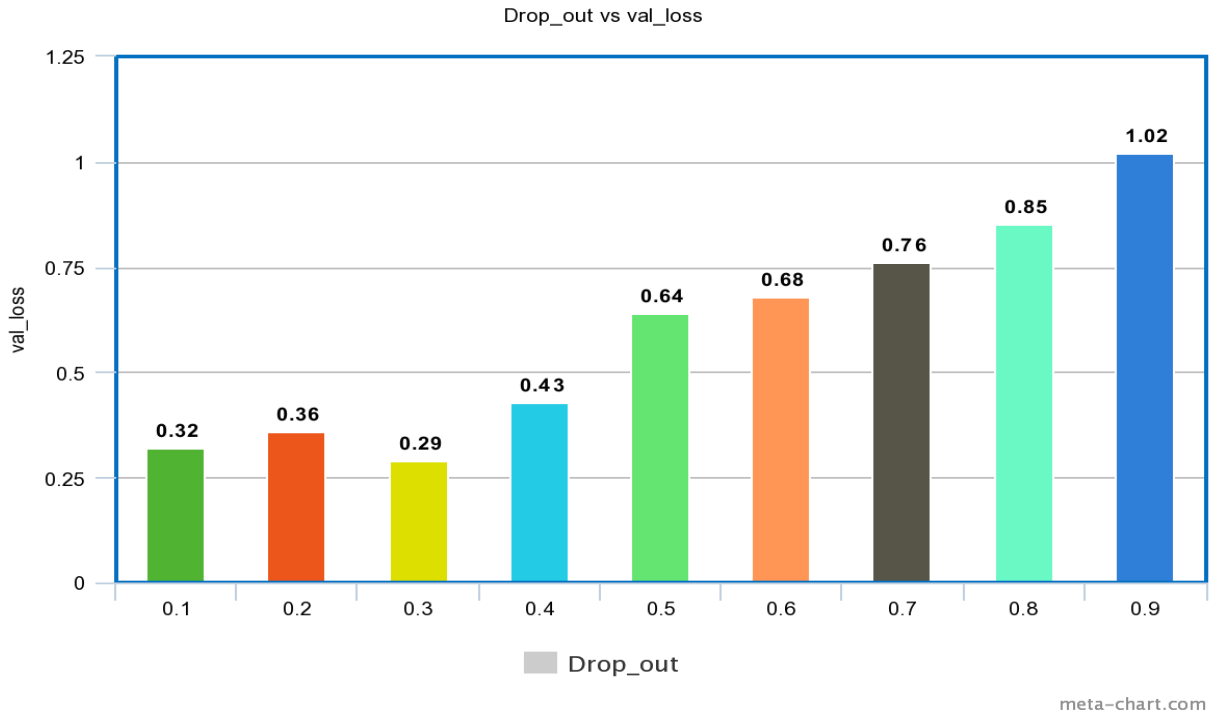

Figure 3

From the above graph we can infer that the val_loss is least when the Dropout rate is set to 0.3 which is the exact value we have used for the implementation. With Dropout rate and optimizer function set we need the learning rate value to be tuned correctly. The graph given below (Figure 4) shows the effect of various selected learning rate values on the prediction accuracy of the model when presented with the testing data. From the graph it can be inferred that we achieve the maximum prediction accuracy when the learning rate value is 0.01and degrades as we increase the learning rate further.

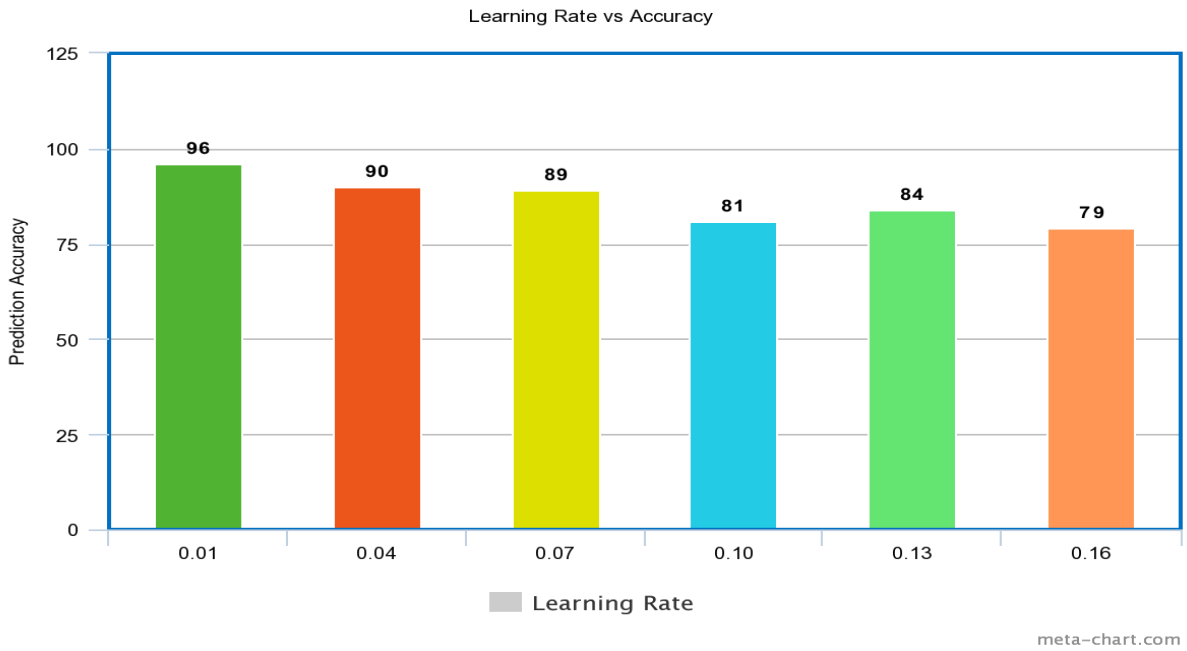

Figure 4

Conclusion

For the FizzBuzz task we implemented a model using Keras APIs. We implemented a model which comprised of Input, hidden and output layers. We used categorical crossentropy as our loss function, Adam as the optimizer function, ReLU and softmax functions as our activation function and EarlyStopping and dropout(0.3) to avoid overfitting of the data. We used all these techniques in order to achieve the best possible prediction as discussed in this report. The testing accuracy of the model for the FizzBuzz task is 96.00% which means out of 100 testing inputs only 4 of them were predicted incorrect. All the 4 were classified in the Other category instead of Buzz category.

References

- https://keras.io/
- https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
- http://ruder.io/optimizing-gradient-descent/index.html#rmsprop