Project 1

Intro to Machine Learning

Illa Nuka Saranya
50248926

17-Sept-18
Saranya@buffalo.edu

PROBLEM DEFINITION

In this task, an integer is given as an input and corresponding string of *fizz, buzz, fizzbuzz* or *other* is given as output. If the given integer is divisible by both 3 and 5, it is printed as *Fizzbuzz* or if it is divisible only by 3 but not 5 it is printed as *fizz* or if it is divisible only 5 but not 3, it is printed as *buzz* or if it not divisible by either of them, it is printed as *other*.

The programs from both the softwares have to be tested on how well they perform in converting integers from 1 to 100 to the FizzBuzz labels.

THE LEARNING SYSTEM

- Here we use Keras which is a high-level neural networks API because of its user-friendliness, modularity, easy extensibility and work with python.
- The given problem is considered as categorical classification problem because we focus on the assignment of categorical target labels to the samples.
- We first have to build a model in neural network to perform the task of testing in software2.0. As the core abstraction of a model is a layer and there is a stack of layers that has input and output, we use **sequential** model to organize them.
- Convert the input to vector of activations by encoding them to binary representation. This is given to the first dense layer of the neural network (hidden) in the form of a csv file to activate the nodes of it.
- Do one-hot encoding on label data to convert it into binary class matrices of 4 categories.
- Create a simple model for which different layers are added. We first add a first **dense** layer called the hidden layer with certain number of nodes and also declare the shape of the given input vector.
- **Number of nodes** in a layer is arbitrarily chosen. The accuracy of the model increases with the increase in the number of nodes in the layer and tends to decrease after some time due to overfitting and high variance.
- A 'layer' does not have an activation. Each individual neuron has an activation. The **state** of a neuron is it's bias + all incoming connections (weight * activation from source neuron). The **activation** is the state of a neuron passed through an activation function.
- We then do the **activation** to provide linearity to the model and choose it in a way that it is helpful in backpropagation to adjust the weights.
- We do **regularization** by adding a **dropout** layer to randomly drop the neurons so as to reduce interdependent learning amongst them. The fraction of neurons that are to be dropped is given as a parameter of Dropout.
- Here we use the loss function **Softmax** as the activation function on the output layer which returns the probabilities of each class where target class has the highest probability.
- After defining the model, we **compile** the model using tensorflow in the backend because of its efficient numerical libraries.
    - Compiling the model: Use **categorical_crossentropy** as loss function to evaluate a set of weights as we have a *Multi-class classification problem.*

    - **Optimizer** is used to search through different weights for the network and optimize the loss function.
    - **Metrics** are collected and reported at the end of every epoch.
- After compiling, we have to **fit** the model to execute this on training data. We set the following parameters along with the processed data,
    - **Epoch**- Number of passes over the entire dataset.
    - **Batch Size** - Number of instances that are evaluated before a weight update in the network is performed.
    - **Callbacks**
        - *Tensorboard* - Visualization tool provided by tensorflow where callback writes a log for visualize dynamic graphs of training and test metrics, as well as activation histograms.
        - *Earlystopping* - Stop training when a monitored quantity with respect to a mode has stopped improving working with a patience of certain number of epochs, for which there is no improvement of the system.
    - **Validation-split**: The fraction of the training data held back on which we evaluate the loss and any model metrics at the end of each epoch without training it.
- Use Matplotlib, plotting library for the Python programming language, to take the model and produce histograms of the training data over different parameters.

REPORTS:

For different values of hyper-parameters, the following graphs are produced with metrics being plotted against the number of epochs.Optimization of training model depends on following factors. Number of hidden layers,Number of nodes in a layer,Dropout,Number of epochs,Validation_split,Activation function,Optimizer,loss function.

While optimizing the model, the following conditions are to be considered.

- Overfitting is a condition where the model memorizes the training dataset and the finds the best fit on it rather on the generalized data.
- Underfitting refers to a model that can neither model the training data nor generalize to new data.
- Too much of any hyper-parameter value leads to overfitting as opposed to the condition of under-fitting that happens due to less training on data with comparatively less values of hyper-parameters.

Activation functions:

- **Sigmoid** is a non linear activation function that is capable of giving non binary activations with a smooth gradient unlike step function.But there is vanishing gradient problem that can be seen in the graphs produced below.

f(x)=1/(1+e^-x)

- **Tanh** is non linear and is capable of giving non binary activations. It also has vanishing gradient problem. But tanh is better than sigmoid because of its higher derivative values and hence the strong gradient values.

tanh(x)=2/(1+e^(-2x)) -1

- **Relu** is Rectified Linear Unit - non-linear nature.
    - Not bound as it ranges from 0 to inf. Compared to tanh / sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
    - Non-saturation of its gradient, which greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid / tanh functions is the biggest advantage of relu. Using relu can kill the some weights for input less than 0 .

f(x)=max(0,x)

- **Leaky Relu:** Variant of relu that doesn't kill the weights for the negative values of x.Keeps the neurons alive with small derivate values on x.

f(x)= ax, x<0= x, x>=0

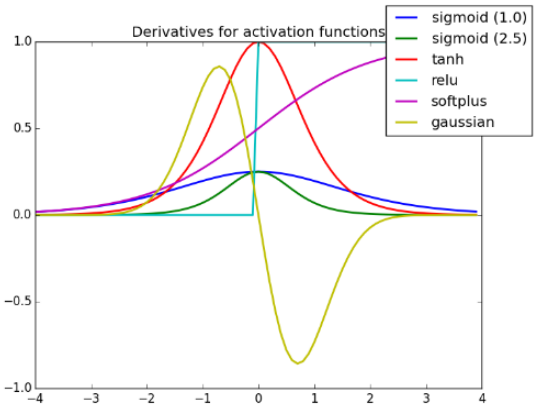


Fig: Derivative of Activation Functions

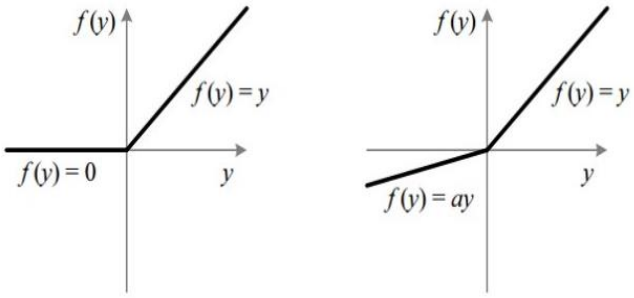


Fig : ReLU v/s Leaky ReLU

Vanishing gradient problem

- The vanishing gradient problem arises due to the nature of the back-propagation optimization which occurs in neural network.
- The weight and bias values in the various layers within a neural network are updated with each optimization iteration by stepping in the direction of the *gradient* of the weight/bias values with respect to the loss function.

Bias

- A bias unit is an "extra" neuron added to each pre-output layer that stores the value of 1. Bias units aren't connected to any previous layer and in this sense don't represent a true "activity".
- A bias value allows you to shift the activation function to the left or right**,** which may be critical for successful learning.

Optimizers

RMSPro - Good choice for recurrent neural networks.

Adagrad - Optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

Adadelta More robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients. This way, Adadelta continues learning even when many updates have been done.

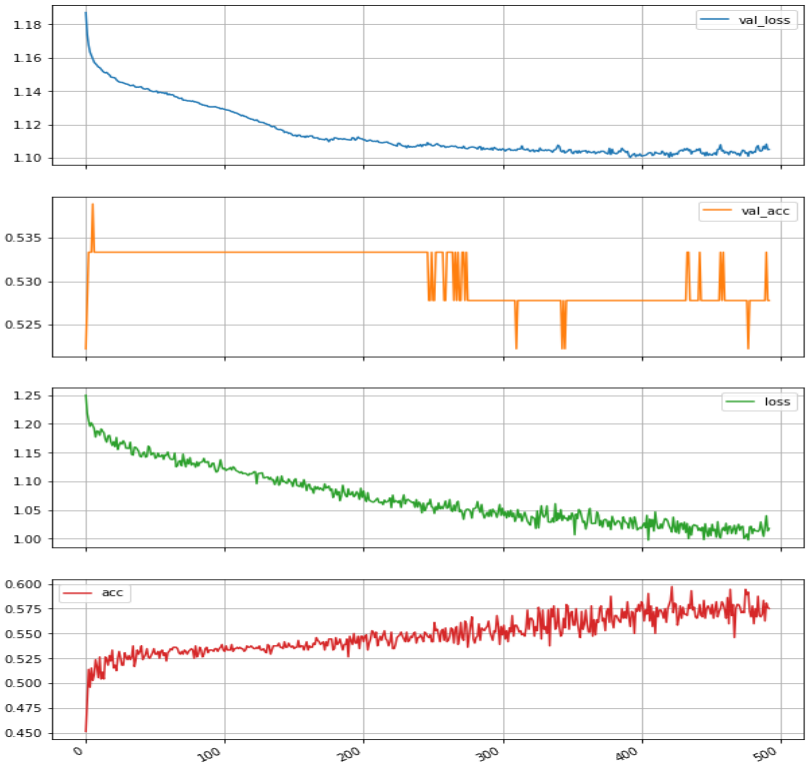
Sgd- Stochastic gradient descent, also known as incremental gradient descent, is an iterative method for optimizing a differentiable objective function, a stochastic approximation of gradient descent optimization.

Test 1

Errors: 14 ; Correct :86
Testing Accuracy: 86.0

- Validation split = 0.2
- epochs = 1000
- model batch size = 128
- tensor board batch size = 32
- patience = 100
- dropout = 0.2
- first dense layer nodes = 256
- No of hidden layers=1

Test 2: Decreasing the number of nodes



- Validation data split = 0.2
- Num epochs = 1000
- Model batch size = 128
- Tb batch size = 32
- Early patience = 100
- dropout = 0.2
- first dense layer nodes = 32
- No of hidden layers=1

Errors: 42 ; Correct :58

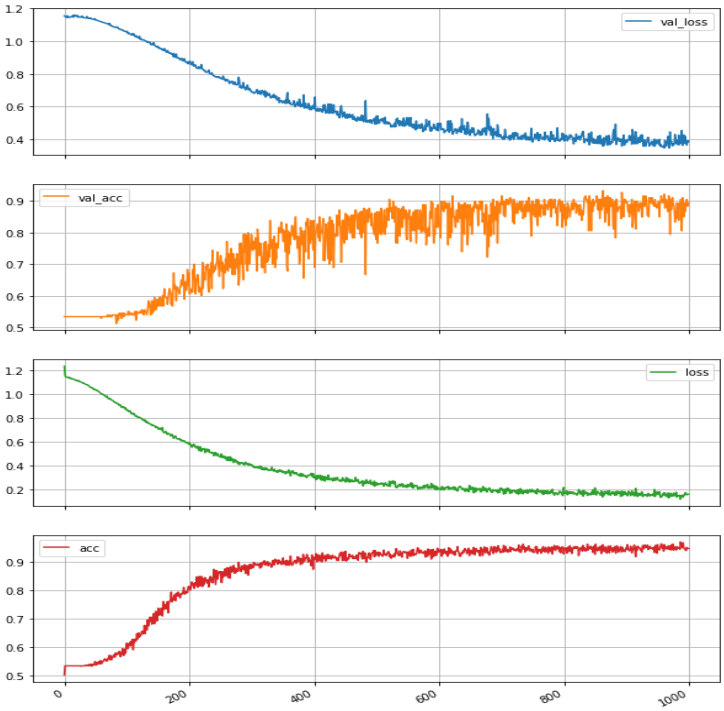
Testing Accuracy: 57.99999999999

Test 3:

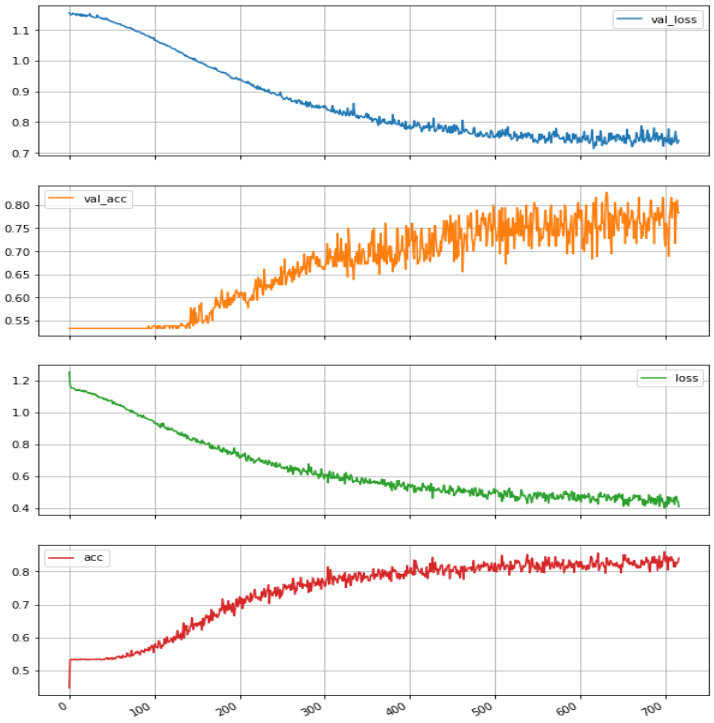
Errors: 9 ; Correct :91
Testing Accuracy: 91.0

- validation_data_split = 0.2
- num_epochs = 1000
- model_batch_size = 128
- tb_batch_size = 32
- early_patience = 100
- drop_out = 0.2
- first_dense_layer_nodes = 512
- activation function=relu
- No of hidden layers=1

As the number of nodes increases, the value of the accuracy increases till some time where it achieves maximum value and then decrease s afterwards.



Test 4



Increasing the dropout from 0.2 to 0.5 keeping the other values same.

Errors: 21; Correct :79

Testing Accuracy: 79.0

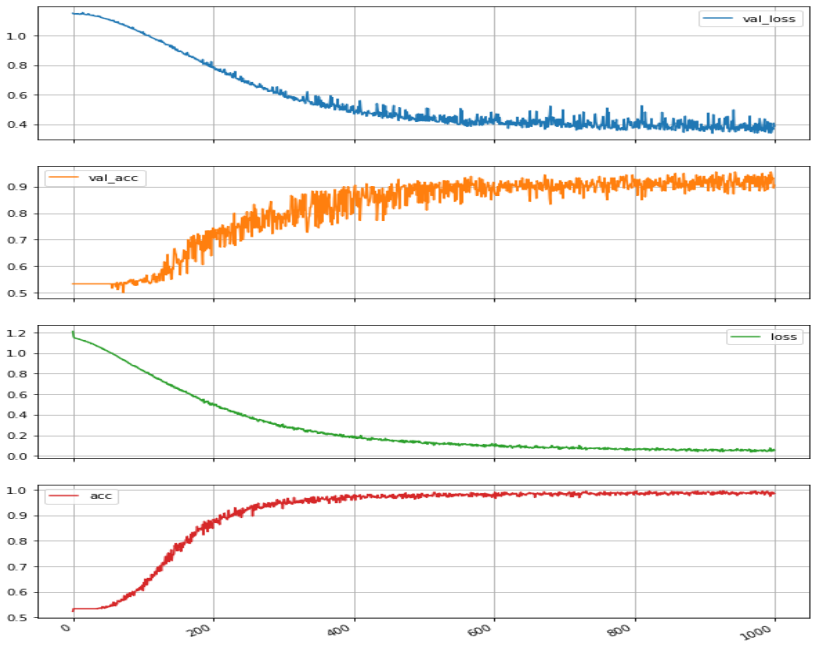
- No of hidden layers=1
- validation_data_split = 0.2
- num_epochs = 1000
- model_batch_size = 128
- tb_batch_size = 32
- early_patience = 100
- drop_out = 0.5
- first_dense_layer_nodes = 512
- activation function=relu

Test 5: Decreasing the dropout from 0.5 to 0.1 keeping the other values unchanged

Errors: 3 ; Correct :97

Testing Accuracy: 97.0

- validation_data_split = 0.2
- num_epochs = 1000
- model_batch_size = 128
- tb_batch_size = 32
- early_patience = 100
- drop_out = 0.1
- first_dense_layer_nodes = 512
- activation function =relu

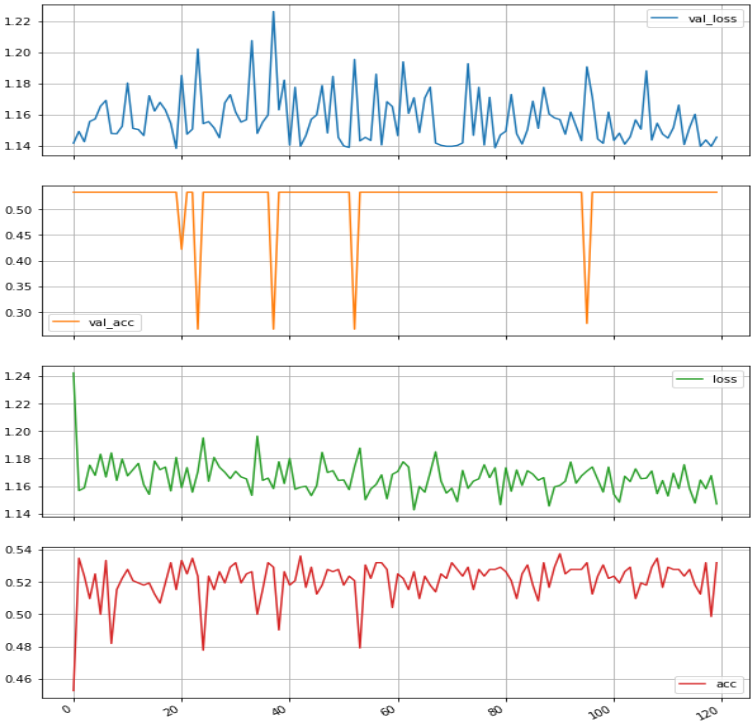


Dropout values range from 0-1.

Small dropout values lead to over-fitting

Large dropout values lead to under-fitting

Test 6: Changing the activation function to sigmoid from relu.



Errors: 47 Correct :53

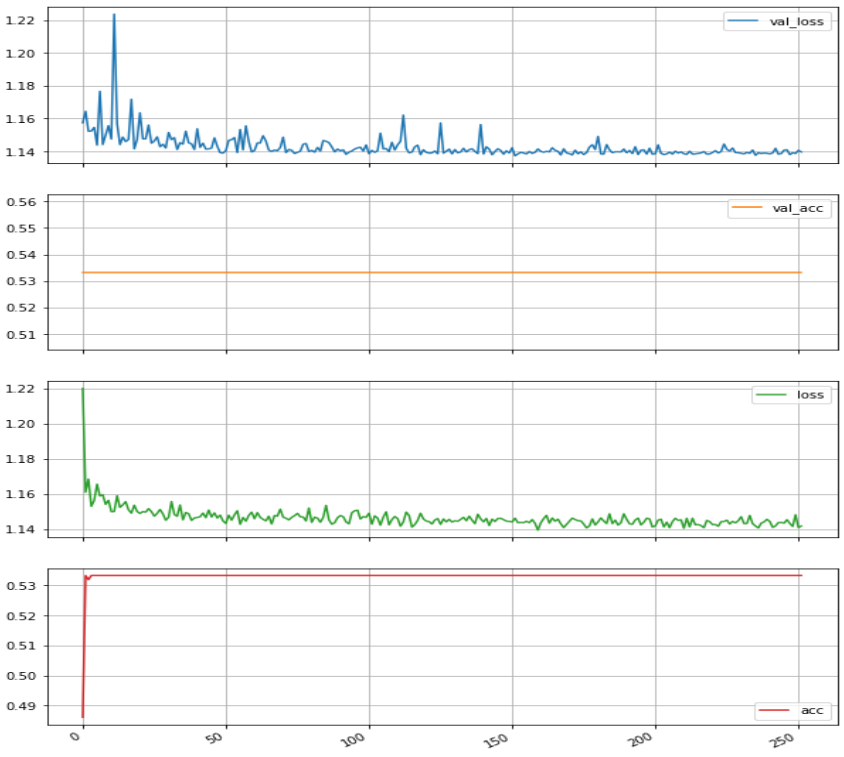
Testing Accuracy: 53.0

- validation_data_split = 0.2
- num_epochs = 1000
- model_batch_size = 128
- tb_batch_size = 32
- early_patience = 100
- drop_out = 0.1
- first_dense_layer_nodes = 512
- activation function =sigmoid

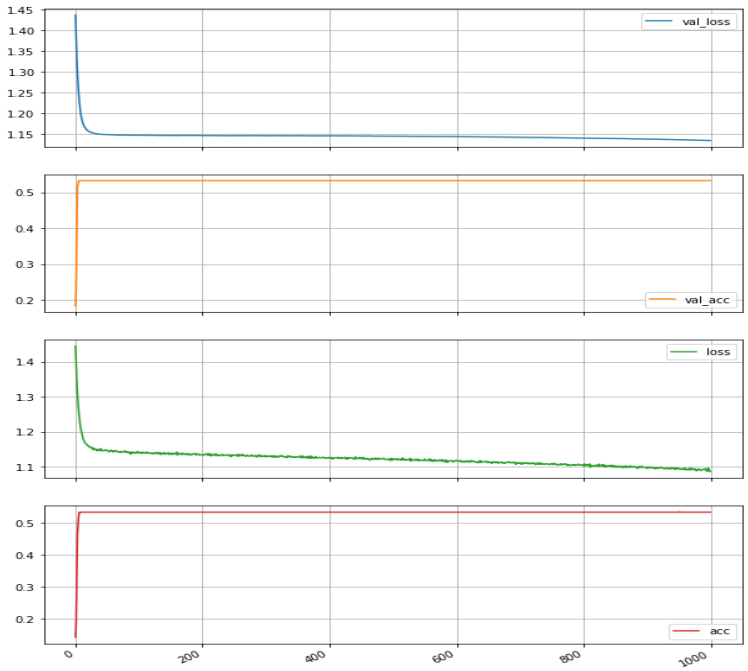
Test 7: Changing the activation function to tanh keepin all other best accuracy parameters constant.

Errors: 47; Correct :53
Testing Accuracy: 53.0

- validation data split = 0.2
- num epochs = 1000
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes = 512
- activation function =tanh



Test 8 : Changing the optimizer to 'SGD' and keeping the other values of the hyper-parameters unchaged where the best accuracy has happened till now.



Errors: 47 ; Correct :53

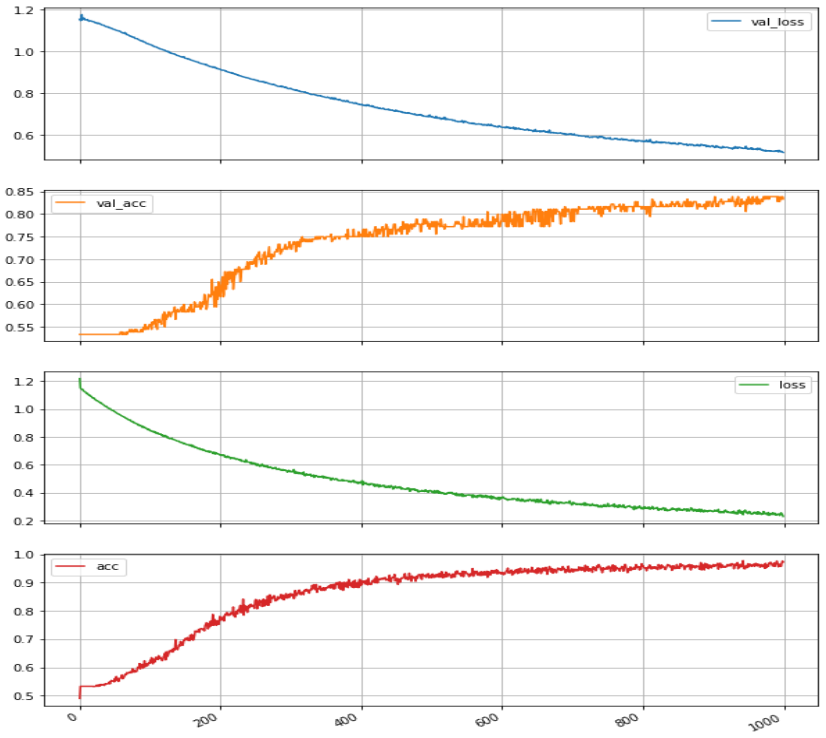
Testing Accuracy: 53.0

- validation data split = 0.2
- num epochs = 1000
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes = 512
- activation function =relu
- optimizer= sgd

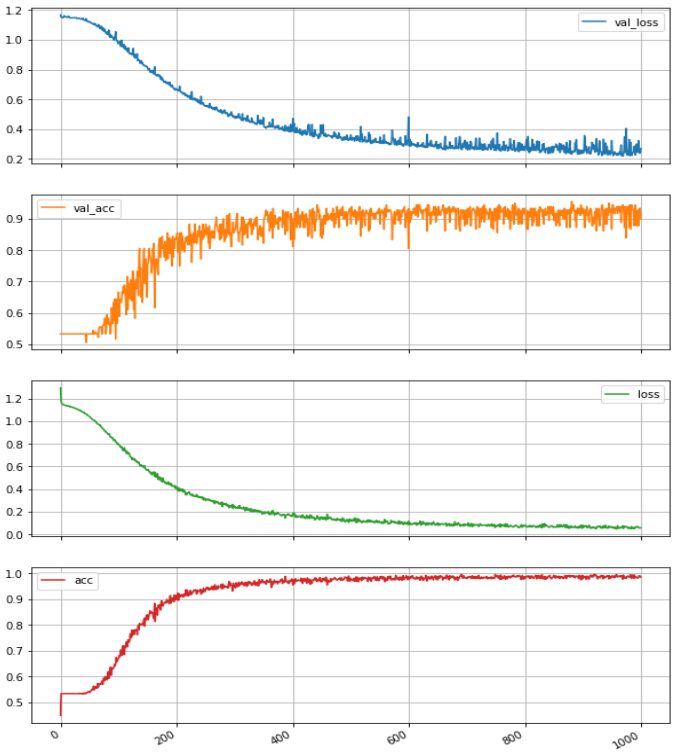
Test 9: Changing the optimizer to adagrad

Errors: 5 Correct :95
Testing Accuracy: 95.0

- validation data split = 0.2
- num epochs = 1000
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes = 512
- activation function =relu
- optimizer= adagrad



Test 10: Changing the optimizer to adadelta.



Errors: 1; Correct :99
Testing Accuracy: 99.0

- validation data split = 0.2
- num epochs = 1000
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes = 512
- activation function =relu
- optimizer= adadelta

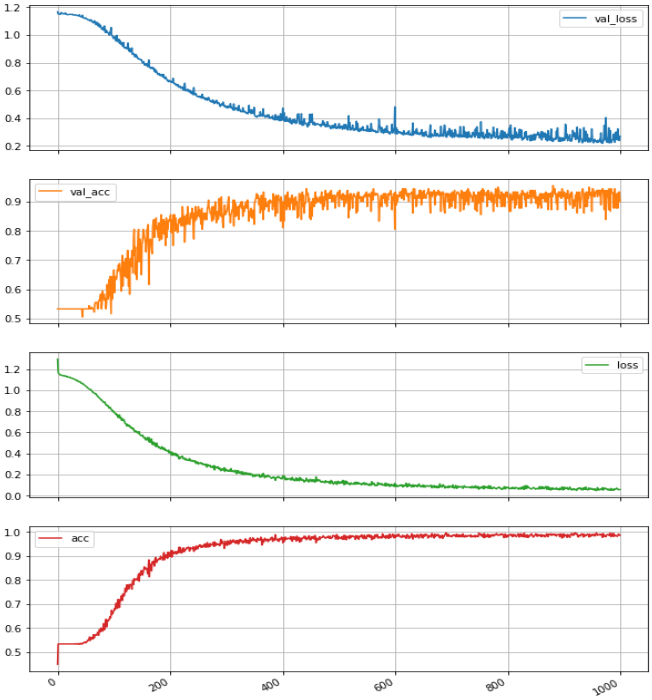
Test 11: Decreasing the number of epochs keeping the values unchanged

Errors: 47 Correct :53

Testing Accuracy: 53.0

- validation data split = 0.2
- num epochs = 10
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes=512
- activation function =relu
- optimizer= adadelta

Accuracy increases with the increase in the numb er of epochs on the dataset and then decreases a fter certain point of time due to overfitting.

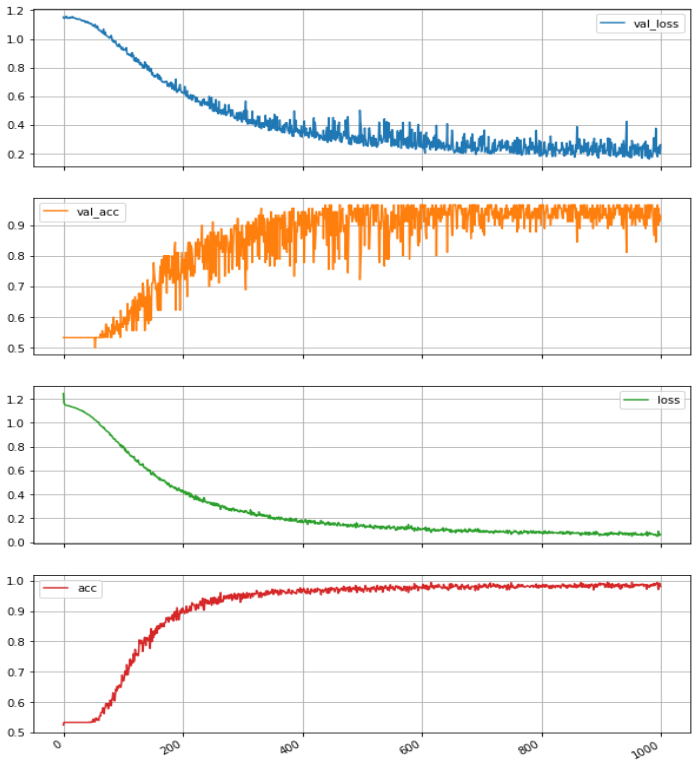


Of all the different combinations of hyper-parameters used, the following values of parameters gave the best accuracy.

Errors: 0; Correct :100

Testing Accuracy: 100.0

- validation data split = 0.1
- num of epochs = 1000
- model batch size = 128
- tb batch size = 32
- early patience = 100
- dropout = 0.1
- first dense layer nodes=512
- activation function =relu
- optimizer= adadelta



Conclusion:

Software 1.0 is logic based approach which gives 100 % accuracy in finding the fizzbuzz representation of any given input at any point of time whereas software 2.0 is machine learning approach where accuracy changes with various parameters. It is not possible to produce results/predictions by always using logic based approach especially when the data is unstructured. Though software 2.0 doesn't given better accuracy when compared to 1.0, it is very much useful in automating the predictions by training data on the model even when dealing with unstructured data.

References:

1. Keras Documentation,Sept 5,2018, Retrieved from https://keras.io/
2. Master machine learning by using it on *real-life applications*, even if you're starting from scratch, Sept 5,2018, Retrieved from https://machinelearningmastery.com/