# Project Report
# Introduction To Machine Learning [CSE 574]
# Software 1.0 (FizzBuzz)

**Name:** Sagnik Ghosh **UB Person #:** 50289151

## Abstract

FizzBuzz is a programming task that is used to determine whether a number is multiple of three or five. The program takes a sequence of integer elements as input and goes by every single element. In the output, if an element is multiple of three, instead of number, it prints “Fizz”, if the element is multiple of five, it prints “Buzz”, if the element is divisible by both three and five, it prints “FizzBuzz” and if it is devisible neither by three nor by five, it prints “others”.

In this project, our objective is the same, but instead of normal programing technique, we use machine learning approach to solve this problem. In this approach we provide the machine a large set of sequential numbers, which are already labeled with appropriate label (i.e. Fizz, Buzz or FizzBuzz). From this piece of relevant data the machine learns to predict the labels correctly on previously unseen. More accurately the program predicts the labels, more efficient it is.

## What is Machine Learning?

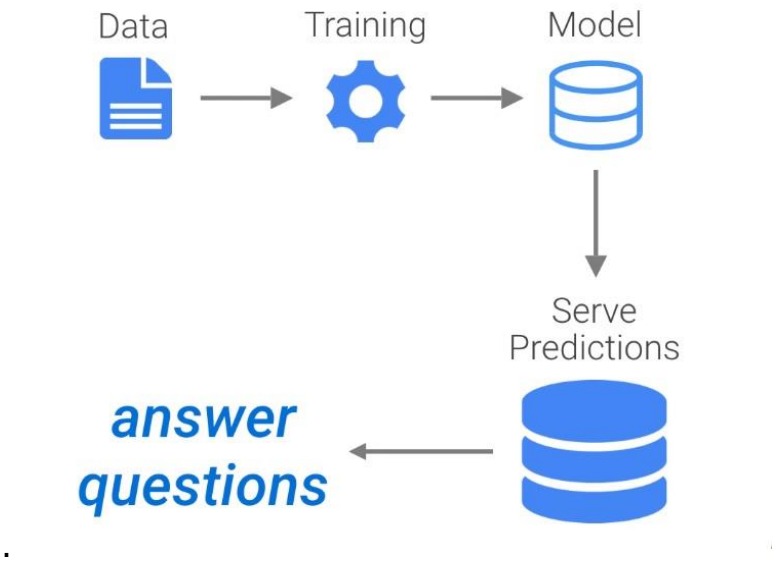
Machine learning is tools and technologies that can be utilized to answer questions to with our data, the data we generate in the form of pictures, music, documents, videos, spreadsheets etc.

Before machine learning, humans used to analyze data in adaptive systems to detect changes in the data patterns, however with the ever-increasing volume of data surpasses the ability of humans to make sense of data and manually write down the rules.

Thus, automated systems that can learn from the data and importantly the changes in data, to adapt in a shifting landscape, became more and more significant.

In short, the definition of the machine learning can be stated as: “using data to answer questions”.

We can split this definition into two parts, “using data” and “answering questions”. These two parts broadly describes the approach of machine learning. Using data, we can train our predictive model to answer questions or making predictions.




Training refers to using our data to inform the predictive model and fine-tuning it. This predictive model can then be used to serve a prediction on previously unseen data. As more data gathered, the model can learn more and can be improved in accuracy of prediction over time.

## What is neural network?

Artificial neural network is a computational model which is inspired from the way in which a human neuron process information. An artificial neural network consists of numerous neurons. Neurons are the basic computational unit in an artificial neural network. A neuron is also known as a node or unit. A node takes multiple input either from previous nodes or from an external source. Every input to a node has a weight assigned to it. The weight of an input is assigned based on its importance.

The node then performs a fixed operation ***f*** on the input values and the weights assigned to that input. This function is known as **activation function**. The activation function adds non-linearity to the output value,

activation function helps the neuron to learn non-linear representation. This is important for a neuron to learn non-linear representation because most real-world data are non-linear.

The following image is an example of a single neuron and the its activation function:

In this neuron, there are two inputs: X1 and X2 their corresponding weights are w1 and w2.

Now the activation function ***f*** is performed on the inputs and the weight values.

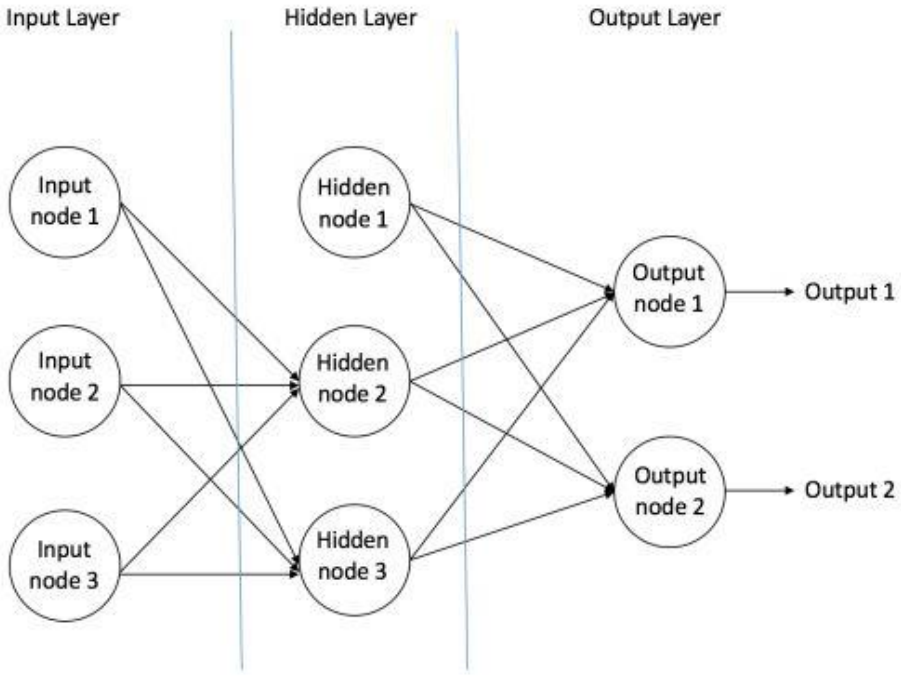
The value then passed on to the output of the neuron:

**Y = f (X1.w1 + X2.w2 + b)**

- **Feedforward Neural Network:**

Feedforward neural network is the one of the simplest forms of the neural networks. In a feedforward neural network, the neurons are arranged in multiple layers and their connections amongst the neurons of the adjacent layers and every connection have a weight associated with it.

The following image is an example of a feedforward neural network:



A feedforward neural network consists of three type of layers:

1. ***Input layer:*** The nodes of this layer takes input from the external source and pass it on to the nodes of the hidden layer. The nodes in the input layer do not perform any computation. The nodes of the input layer are known as “input nodes”.
2. ***Hidden layer:*** The nodes of the hidden layer do not have any connection with the outside world or any external source. They take input from the input layer and perform computation on the input data. Feedforward neural network may have on or multiple hidden layers. It might also have no hidden layers.
3. ***Output layer:*** The nodes of the output layer collects the data from the hidden layer and pass in on to the outside world. The nodes of the output layer are known as ‘output node”.

In a feedforward neural network, the data always flows in one direction, in a sequential order. From input to hidden layer and then to the output layers. There is no loop or cycle and data lever flows from a layer to any of the previous layer. Unlike feedforward neural network, in recurrent neural network, there are connection between nodes that transfer data backward.

There are two types of feedforward neural network:

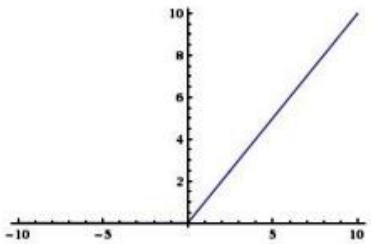
1. **Single layer perceptron:** This type of neural network contains no hidden layer.
2. **Multilayer perceptron:** This type of neural networks contains one more than one number of hidden layers.

## Model Parameters

**Model Definition**

```
In [181]: from keras.models import Sequential
          from keras.layers import Dense, Activation, Dropout
          from keras.callbacks import EarlyStopping, TensorBoard

          import numpy as np

          input_size = 10
          drop_out = 0.2
          first_dense_layer_nodes  = 256
          second_dense_layer_nodes  = 4

          def get_model():

              model = Sequential()

              model.add(Dense(first_dense_layer_nodes, input_dim=input_size))
              model.add(Activation('relu'))

              model.add(Dropout(drop_out))

              model.add(Dense(second_dense_layer_nodes))
              model.add(Activation('softmax'))

              model.summary()

              model.compile(optimizer='adadelta',
                            loss='categorical_crossentropy',
                            metrics=['accuracy'])

              return model
```

- **Dense:** In a neural network, a dense layer is the layer in which every single nodes are connected to every single nodes of the previous node, hence making it densely connected.
- **Activation:** activation function of a node takes the inputs and the weight values and performs some fixed mathematical operations with them. Activation functions bring non-linearity into the output neuron.
- **Relu Activation:** ReLU is a type of activation function. ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero. f(x) = max(0, x)




ReLU

- **Sequential():** sequential model is a model in which all the nodes in a single layer can process information coming from the previous node only, it cannot process any information that comes after.
- **Dropout:** Dropout is a technique where neurons are ignored (“dropped-out”), randomly, during training. This means that those neurons will not pass any information forward to the activation layer and there weights also will not be updated in backward pass. When some neurons are randomly dropped out during training, other neurons will have to step in and handle the representation required to make predictions for the missing neurons. As an effect, the network becomes less sensitive to the specific weights of the neurons. This results in a network that is capable of better generalization and is less likely to overfit the training data.
- **Softmax Activation:** Softmax activation is basically the normalized exponential probability of class observations. Like the logistic function, softmax provides us with a categorical output. The output class is generally a probability distribution. In this project we have four classes, Fizz, Buzz, FizzBuzz and others. Here we have used softmax function as the Activation Function in the Output layer of the Multi-layer perceptron to ensure that the outputs are probabilities and they add up to 1.
- **Optimizer:** Optimizer is a keras object, that minimizes the loss function of the program. Gradient descent is a type of optimizer. The followings are the examples of Keras optimizer:
    - keras.optimizers.Adadelta
    - keras.optimizers.Adam
    - keras.optimizers.Adamax
    - keras.optimizers.Nadam
    - keras.optimizers.RMSprop

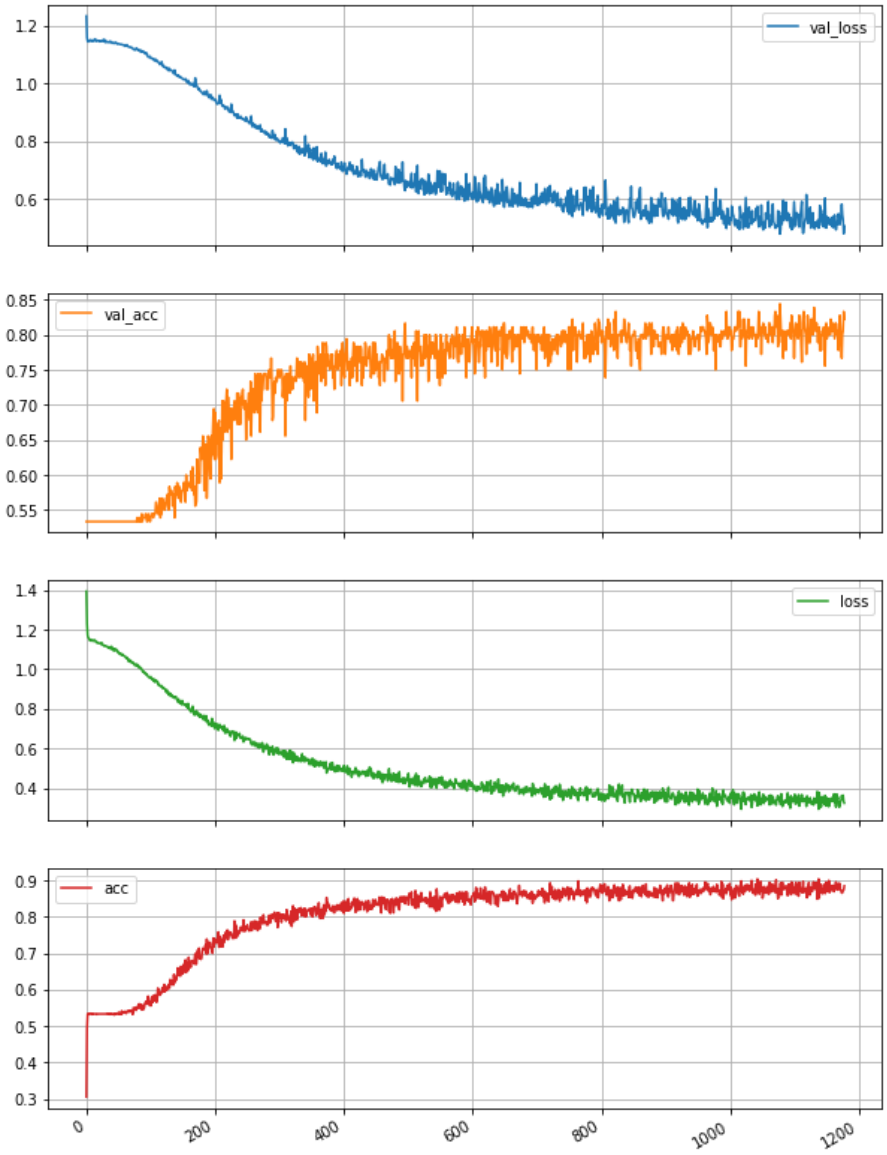
## Run Model

```
In [184]: validation_data_split = 0.2
          num_epochs = 10000
          model_batch_size = 128
          tb_batch_size = 32
          early_patience = 100

          tensorboard_cb   = TensorBoard(log_dir='logs', batch_size= tb_batch_size, write_graph= True)
          earlystopping_cb = EarlyStopping(monitor='val_loss', verbose=1, patience=early_patience, mode='min')

          # Read Dataset
          dataset = pd.read_csv('training.csv')

          # Process Dataset
          processedData, processedLabel = processData(dataset)
          history = model.fit(processedData
                              , processedLabel
                              , validation_split=validation_data_split
                              , epochs=num_epochs
                              , batch_size=model_batch_size
                              , callbacks = [tensorboard_cb,earlystopping_cb]
                             )
```

- **validation_data_split:** It defines what portion of the training data must be used for training purpose and what portion has to be used for validation purpose. If the value of validation_data_split is 0.2, 80% of the training data will be used for training and 20% will be used for validation.
- **num_epochs:** It defines the number of iterations over the entire dataset.
- **early_patience:** It defines after how many epochs it should stop processing, if there is no improvement in the learning rate.

## Steps followed

- Case 1: Default graph (without any changes) : Testing Accuracy: 87.0

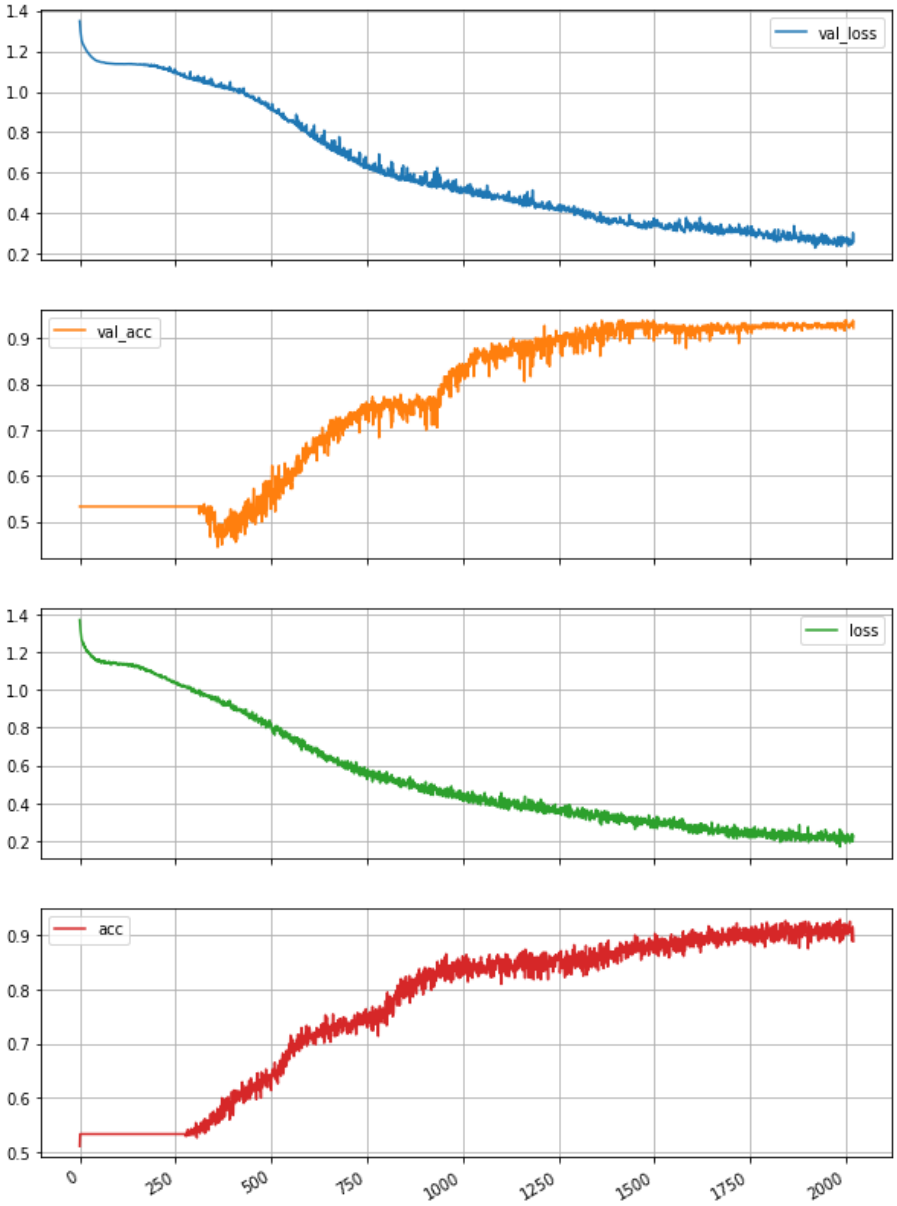


- Case 2: One dense layer added: Testing Accuracy: 91.0

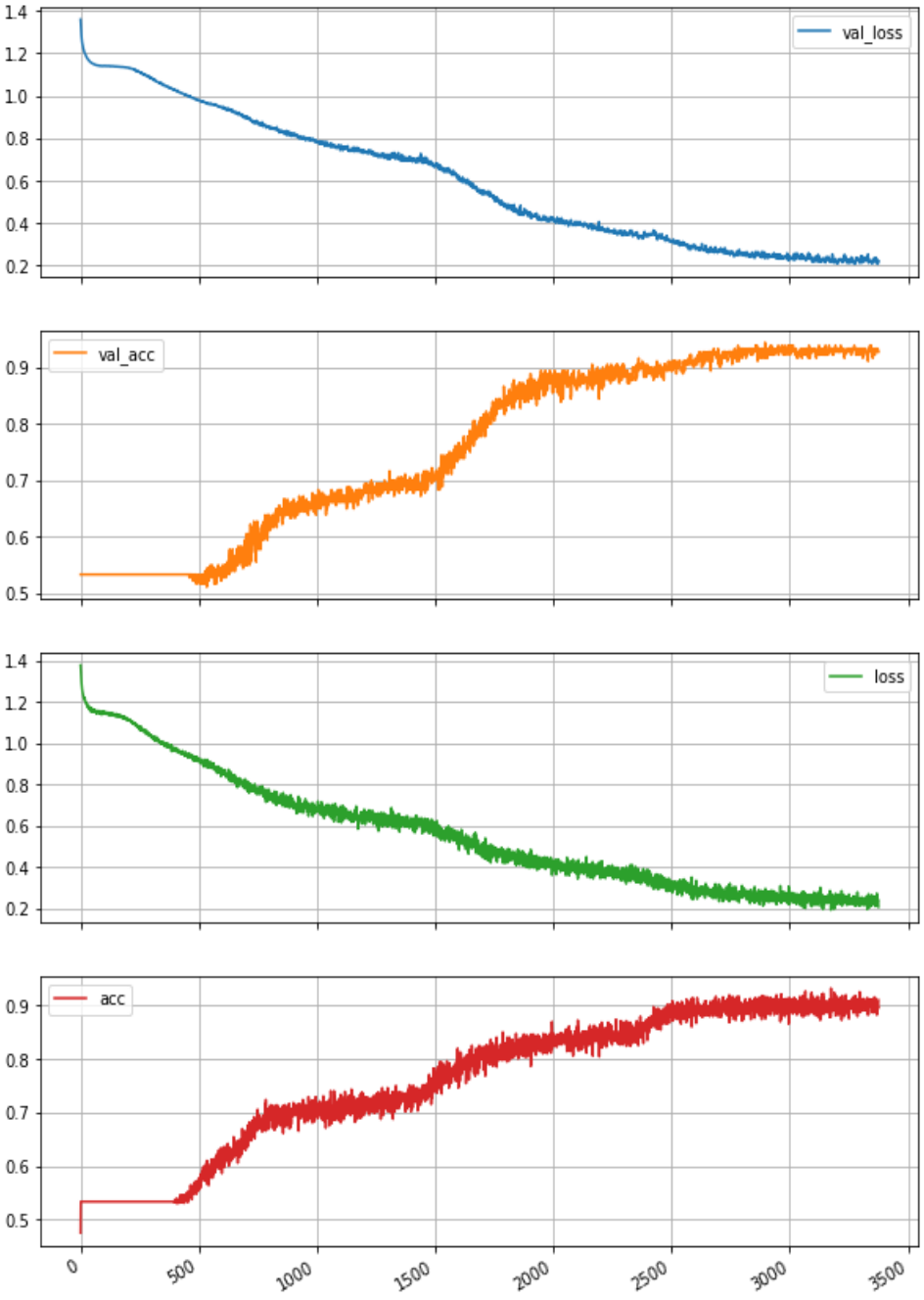
  first_dense_layer_nodes  = 512

  second_dense_layer_nodes  = 128

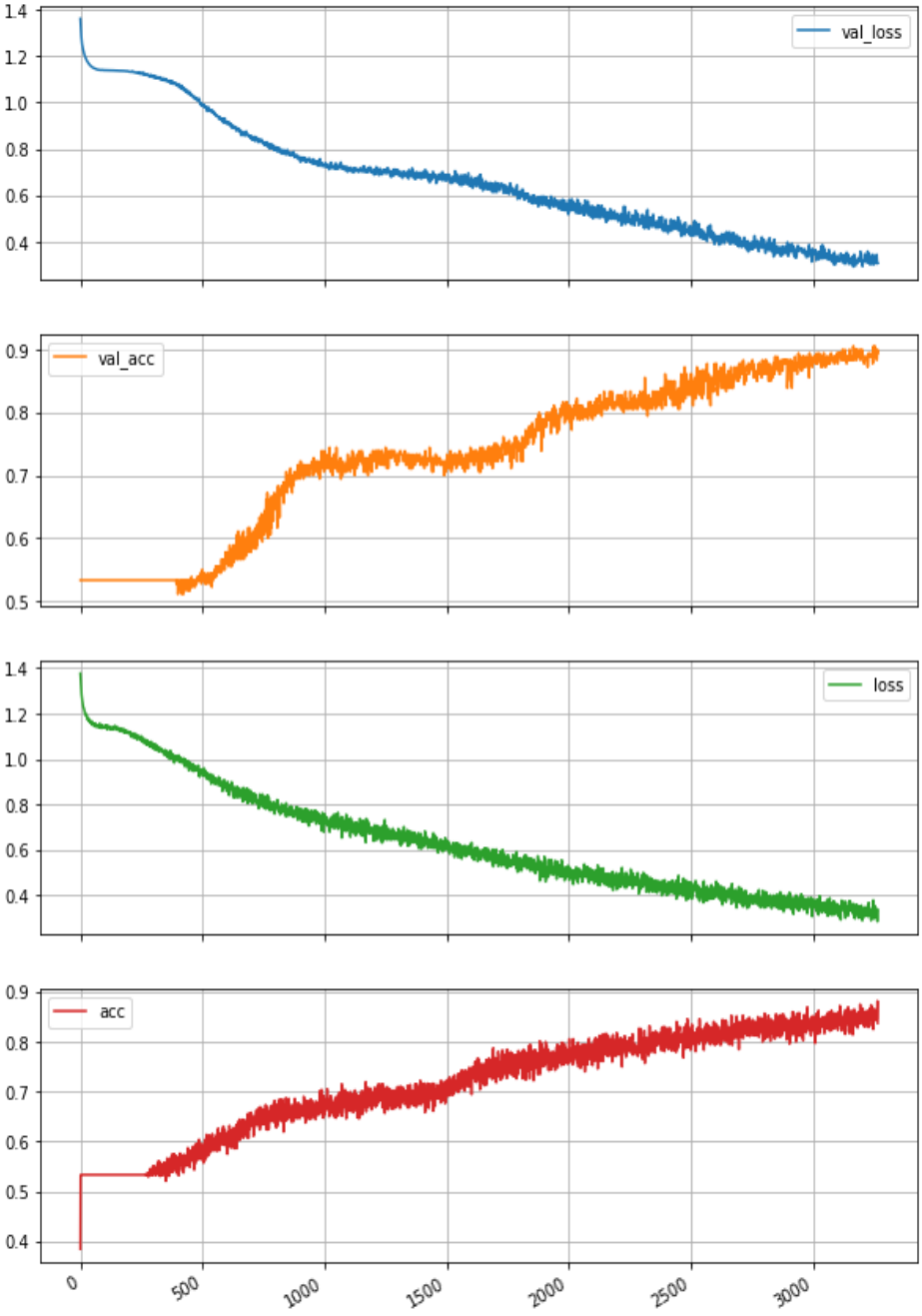
  third_dense_layer_nodes  = 4



- Case 3: Changed optimizer to 'adamax’: Testing Accuracy: 95.0



- Case 4: drop_out increased to 0.3: Testing Accuracy: 91.0



## References Used


- https://keras.io/
- https://www.coursera.org/learn/machine-learning/
- https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
- https://towardsdatascience.com/what-is-machine-learning-8c6871016736


## Conclusion

The case 3 scenario gave the maximum result (best accuracy) in this project. Therefore, in conclusion of this report it can be stated that the network setting in the Case 3 scenario is the most optimized setting for this project.