Project 1.1 : Software 1.0 Versus Software 2.0 Disha Mehra
Instructor: Prof. Sargur N. Srihari Person No. :50288911
TA: Mihir Chauhan, Tiehang Duan, Alina Vereshchaka and others

Objective

The project is to compare two problem solving approaches to software development: the logic-based approach (Software 1.0) and the machine learning approach (Software 2.0).

Task

We have taken the task of FizzBuzz and implemented it with both approaches (Software 1.0 and Software 2.0). In this task an integer divisible by 3 is printed as Fizz, an integer divisible by 5 is printed as Buzz and an integer divisible by both 3 and 5 is printed as FizzBuzz. If an integer is not divisible by 3 or 5 or 15, it simply prints the input as output (for this last case, the input number can be classied as Other in Software 2.0, it should then be handled using Software 1.0, which prints the input as output).

Software 1.0 And Software 2.0

Software 1.0 consists of explicit instructions to the computer written by a programmer where each line of code identifies a specific point in program space with desirable behavior whereas Software 2.0 approach is to specify some goal on the way the program behaves, a rough skeleton of code is written and the subset of the program space is provided to search , its finds a program that works best as in shows the desired behavior, with the use of computational resources provided.

Software 2.0 is better than Software 1.0 in numerous ways. One of them being that the modules can mold into an optimal that is we can easily backtrack to the whole if two Software 2.0 modules that were originally trained separately interact. Software 2.0 uses constant memory as there is no dynamically allocated memory and one doesn't have to worry about the memory leaks. These are few benefits of Software 2.0 as compared to Software 1.0.

Neural Networks: Software 2.0 Approach

Simplest definition of an artificial neural network:

"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

-Dr. Robert Hecht-Nielsen(inventor of Neurocomputer)

Artificial neural networks are typically organized in layers. These layers are made up of a number of interconnected nodes which contains an activation function. A pattern is fed to the network via the input layer, which communicates to one or more hidden

layers where the actual processing is done via a system of weighted connections. The hidden layer then further links to an output layer where the answer is output.

For a neural network to learn, there is a feedback process called back-propagation. This involves comparing the actual output with the expected output that the network was meant to produce and using the difference between them to modify the weights of the connections between the layers in the network, working from the output units through the hidden units to the input units-going backward.

In time, back-propagation causes the network to learn, reducing the difference between actual and expected output to the point where the two are exactly similar, so the network figures out the things exactly as it should.

FizzBuzz using Software 2.0

FizzBuzz program is implemented in Software 2.0 by using Python, Tensorflow and Keras. Software 1.0 program is written in Python itself and has a perfect accuracy. Software 2.0 has used Tensorflow which is an open source machine learning library and Keras which is high-level neural networks API, capable of running on top of TensorFlow.

Type of Model:
We have used Sequential model(linear stack of layers) to create the network. We add dense layers that is a layer where each unit is connected to each unit in the next layer.

Layers:
We have added 10 nodes in the input layer as our data(training and testing) can be easily accommodated in that and 4 nodes in the output layer as we have only four types of output(Fizz, Buzz, FizzBuzz). We add hidden layer nodes in between the input and the output layers. And each input node is connected to each node in the hidden layer with some weights and so are the nodes are connected from hidden layer to the output layer.

Activation Function:
After adding the different layers we add activation function whose role in a neural network is to produce a non-linear decision boundary(whether to fire this output/signal or not) via non-linear combinations of the weighted inputs. We are using ReLU as an activation function between input layer and hidden layer and Softmax between hidden layer and output layer.

ReLU is used as an activation function because it has no gradient vanishing problem and it does not activate all the neurons at the same time. It means that if you look at the ReLU function if the input is negative it will convert it to zero and the neuron does not get activated. This means that at a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
The softmax function is often used in the final layer of a neural networks as it normalizes the input array in scale of [0, 1]. Also, sum of the softmax outputs is always equal to 1. So, neural networks model classifies the instance as a class that have an index of the maximum output.

Dropout:
When we create the model we also specify the dropout which is a regularization technique, that aims to reduce the complexity of the model with the goal to prevent over fitting. Using dropout", you randomly deactivate certain units (nodes) in a layer. Dropout is only applied during the training and not at the validating time.

Configuring Model:
Before training a model, we need to configure the learning process, which is done via the compile method. It receives three arguments:

- Optimizer: Optimizer helps in changing the weight(gradient decent). It helps in achieving the reduction of loss function by changing the weights.
- Loss function: This is the objective that the model will try to minimize. We have used categorical cross entropy as loss function here. Categorical cross entropy is used for the multi-class classification problems where each example belongs to a single class and as our problem is multi-class problem hence we are using categorical cross entropy.
- Metrics: For any classification problem we require the accuracy.

Training Model:
Once we have configured the model , we go towards the training of the model where there are different hyper parameters which are responsible to make our model work efficiently.

- Validation data split: In validation data split the training data set is divided into two parts: i)Training ii)Validation. Once the model is trained on the training data set, it then tests itself on the validation data set.
- Number of epochs: Maximum number of epochs that the model will run for.
- Model batch size:Epoch contains multiple batches to fit in the training data. And model batch size refers to the maximum number of training that a batch con contain.
- Early patience: This indicates the maximum number of epochs after which if the loss remains constant or instead the loss starts increasing, training of the model stops.
- Callbacks: A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during its training.

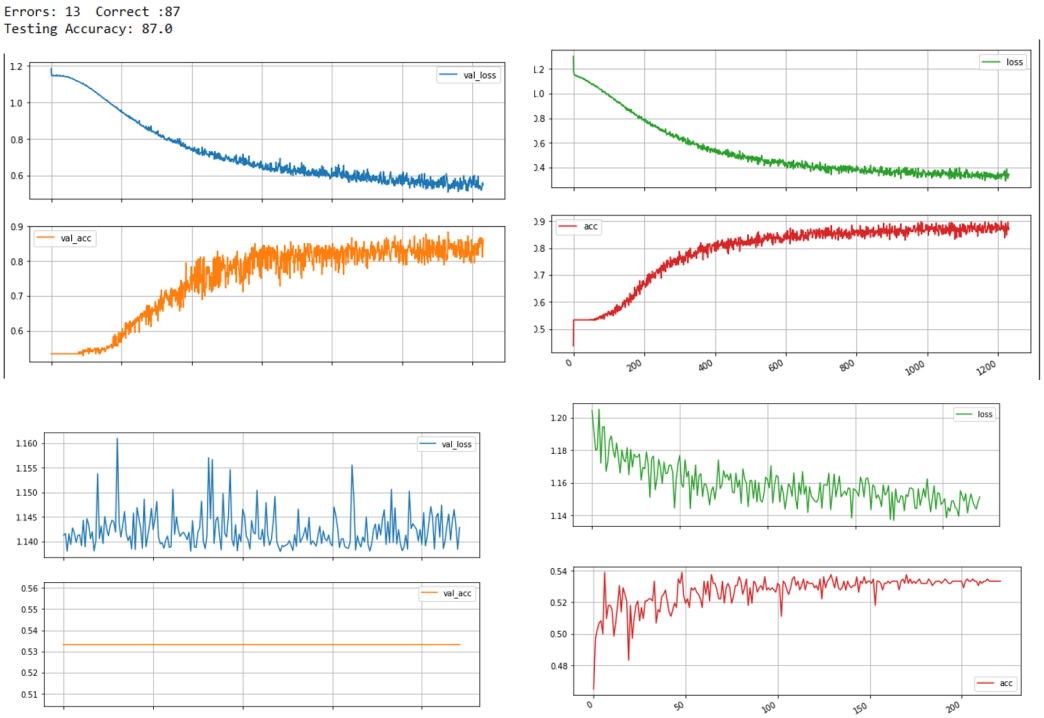
Testing Model:
Once the data is trained we test our model with the testing data set. Each and every number in the testing data set is converted into 10 bit binary equivalent. Once converted it passes through our model where every bit is multiplied with some weights along with the addition of bias and activation function at the node of hidden layer. After the bits reach the hidden layer they are again multiplied with some weights and activation function Softmax acts on the output and normalizes the input array in scale

of [0,1]. We get binary class matrix as our output which is then converted to the desired labels of Fizz, Buzz, FizzBuzz or other. At the end we calculate the accuracy of our model by calculating how many number were termed with incorrect labels and how many were termed as correct.

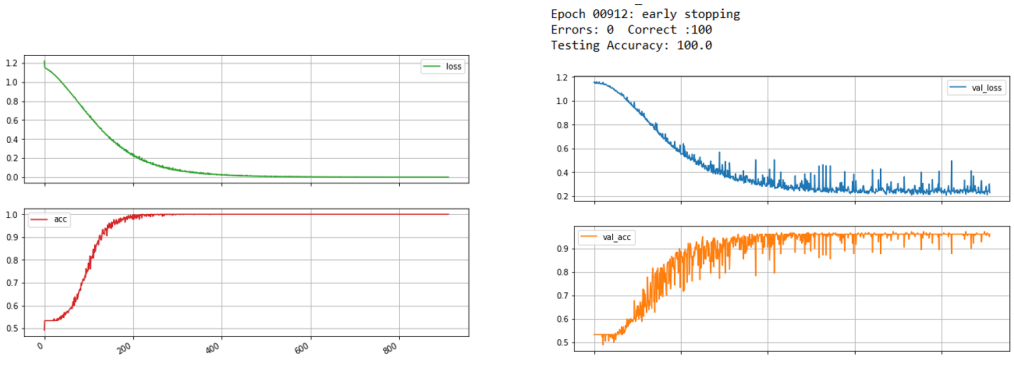
Analysis & Testing

I have changed multiple hyper parameters for the proper analysis of this algorithm.

- Activation function: When the problem is run with ReLU activation function its accuracy is far better than when the activation function is changed to sigmoid.(Refer below)



- Dropout and Hidden layer nodes: When the problem is run with the change in drop out from 0.2 to 0.0001 and nodes in the hidden layer to 1000. This will lead to a very high accuracy but on careful observation it was the because of the overfitting which could be identified by comparing the val loss and loss whose difference was highly comparable. (Refer below)

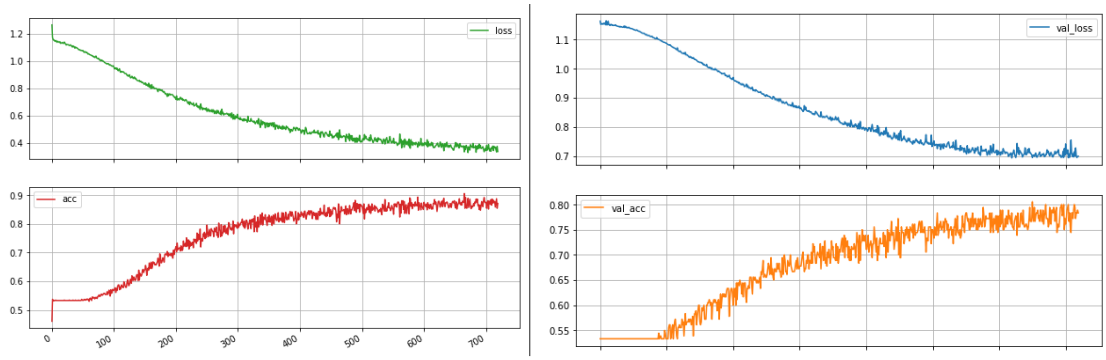


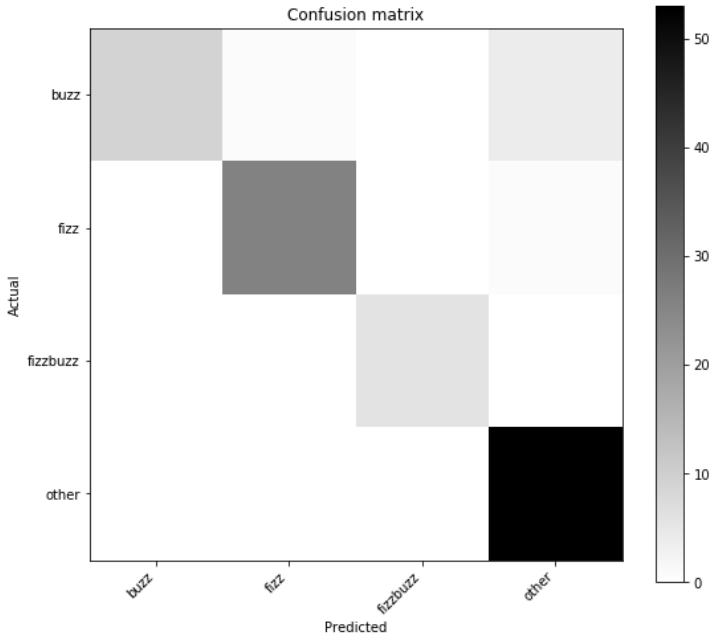
- Hidden layer nodes: When the hidden layer nodes change from 100 to 1000 the accuracy increases from 75% to 90% as it the model is properly trained.

Final Evaluation

While testing various hyper parameters I came across a particular case where the accuracy of the model was 94% without the over fitting of the model(Refer below).I ran the same program a couple of times to check whether the accuracy remains the same or is it variable and it was always above 90%. Variable accuracy is due to a number of factors important one being how the model is trained and what weights are given to the bits at the time of computation inside the neural network.

- Decreasing the drop out from 0.2 to 0.1 to let the model train better.
- Increasing the hidden layer nodes to 310.





References

[1] https://medium.com/@karpathy/software-2-0-a64152b37c35

[2] http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html

[3] https://www.explainthatstuff.com/introduction-to-neural-networks.html

[4] https://datascience.stackexchange.com/

[5] https://www.quora.com/

[6] https://www.tensorflow.org/

[7] https://keras.io/