Machine Learning CSE574

David Jegan Abishek Dominique
Person no 50290785
University at Buffalo
davidjeg@buffalo.edu

Abstract

The report gives a contrast difference between the traditional programming methodology and Machine learning model. I have leveraged the Fizz-buzz problem to point out the intricacies and advantages ML (Software 2.0) proposes over conventional programming model (Software 1.0).

1 Introduction

1.1 Fizzbuzz logic

We print specific outputs based on the divisibility of the number with 3 and 5. We have Fizz, buzz and Fizzbuzz for 3, 5, and 15 respectively.

1.2 Fizzbuzz in ML

Fizzbuzz is a classic 4 class classification problem.
We teach the machine to predict numbers based on the divisibility.
We use a standard ML framework named 'Tensorflow' to assist us.

1.3 Approach

There are four stages in this problem

1. Create training data
2. Setting up the model
3. Training and tweaking the model
4. Validating the model

2 Methodology

2.1 Creating Training data

For the algorithm to learn we provide a training dataset. This data is crucial for the prediction and must contain examples for all prediction classes (eg., Fizz, Buzz, Fizzbuzz etc).

We consider training set of values from 101 to 1000. We cannot pass these parameters as just sting/int. It should have a 'label' for the machine to understand the type of data. Also we cannot train the model with the test data.

2.2 Setting up the model

2.2.1 Neural Networks

A ML model contains a network of entities called Neurons, which are responsible for running a mathematical equation. There are layers of these neurons, and are connected to the next level via weights.

$$Neuron = \sum(Weight * Input) + Bias$$

2.2.2 Activation

The ML model consists of a network of neurons. Each neuron is linked with the next level via weights. The output for a neuron is called activation.

2.2.3 Activation functions

Inorder to check if the neurons are active, we find the output for each neuron and calculate a scalar value. Based on this output we decide an algorithm which is sensitive to input and has faster computation time. Example for Activation functions are Sigmoid, ReLu, Leaky Relu, Tanh etc.

2.2.4 Input/Output format

Rather than passing the inputs as string/int, we pass it in their binary formats, inorder to get a vector of activations. So we consider 10 as input size (Since $2^{10} = 1024$)
On the same lines, the output in this problem is a 4 class classification (Fizz, Buzz, Fizzbuzz or None). So we consider the output size as 4.

2.2.5 Layers of NN

There are three types of layers, ie., Input, Hidden and Output layers. The input and Output is a single layer whereas the hidden layer can be any number depending upon the problem to be solved (features to be extracted).
The input and output layer directly map to the inputs and outputs respectively, whereas the hidden layer

helps to translate/extract more information from the input layer and converts it into a context to be used by the output layer.

2.2.6 Normalization

Normalization makes sure the input does not exceed the natural range a function was defined for.

2.2.7 Weights

The weights connecting the neurons help to find the output. These values are updated via a method called Back-propagation. It can be initialized randomly or using a trained weight.

2.2.8 Learning Rate

The rate at which an model forgets its existing belief system is called learning rate. If the rate is high then the model adapts quickly and if its slow, it learns slowly.

2.2.9 Cost Function

This method helps to increase the accuracy of the model. The value must be low in order to mitigate errors. This is done by randomly finding the lowest minima for a function until we find a convergence point (Stochastic Gradient Descent). Example of cost function includes Quadratic, Cross-entropy, Exponentional cost, Hellinger distance etc.

2.3 Training and tweaking the model

2.3.1 Overfitting

When the model memorizes the training data, it stops being general and fails for test values. To avoid overfitting we must regularize the model.

2.3.2 Epoch and Blocksize

The number of iterations done on the training set is called epoch. Batchsize is the number of sample taken for processing by the model at a time.

2.3.3 Tweaking

We obtain the results of our training. We can increase the accuracy by changing the methods and variables via trial and error method.

2.3.4 Validation set

Instead of directly running the model on the test data, we run it over validation set and tweak attributes accordingly.

2.4 Validation

The model is run against the test set, and the results are calculated. The input is in binary bits of size 10, and the output is the Fizzbuzz method.

3 Implementation

The Software 1.0 returns an hundred percent accuracy because the algorithm works for all test cases and therefore does not predict the supposed output. Rather it follows an imperatively valid formula to identify the result. The output of this algorithm is used as a standard for SOftware 2.0 Whereas in Software 2.0, the algorithm learns based on the test cases provided, and makes a decision with the training. Thus the results can be tweaked if the hyperparameters of the function is changed.

1. Generating the training dataset

   We use software 1.0 to generate ground-truth values. A sample python script is written to loop from 101 to 1000. The ML model requires a label to recognize the result. So the 'Fizz/Buzz/Fizzbuzz/None' results are tabulated and are appended along with the number.

2. Creating the model

   We then move on to define the model with the input, output and hidden layers. The weights are defined and the learning rate is chosen by trial and error method The activation function is chosen and the hidden & output layer computation is defined. Also the loss functions and SGD functions are defined.

3. Training the model

   The Epoch and the Batch size are decided and the model is trained with a value randomly generated from the training set. Based on the Epoch value, the total training set is looped through multiple times to help the model generate patterns.The model is then tested with the test set which consists of values from 1 to 100.

4. Tweaking the model

   The accuracy of the model is then compared with the obtained result and the original label of the test set. Based on the output, hyperparameters are changed to obtain better results.

4 Findings

4.1 When different activation functions are used

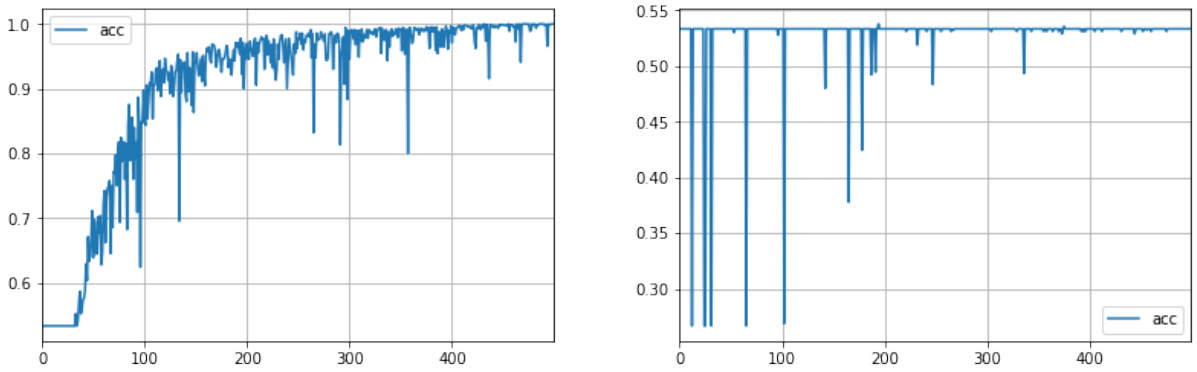

Figure 1: Predication accuracy for ReLU (left) and Sigmoid activation methods (right)

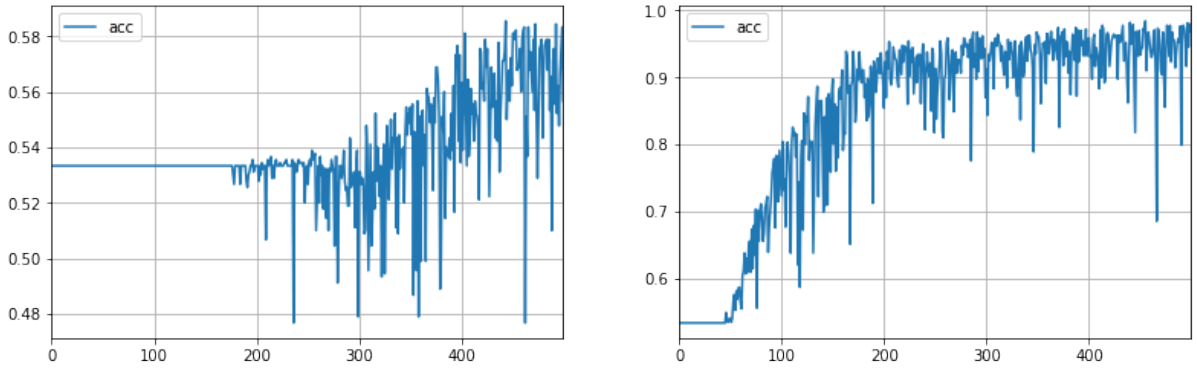

Figure 2: Predication accuracy for Tanh (left) and LeakyReLU activation methods (right)

For other constant parameters, the activation functions were changed to understand how the function impacts the accuracy of predication.

ReLU offers better accuracy than sigmoid/tanh, because of its sensitivity for large numbers and sparsity property. We also observe that ReLU becomes more reliable than leakyReLU as the number of Epochs increase.

4.2 When the Epoch increases

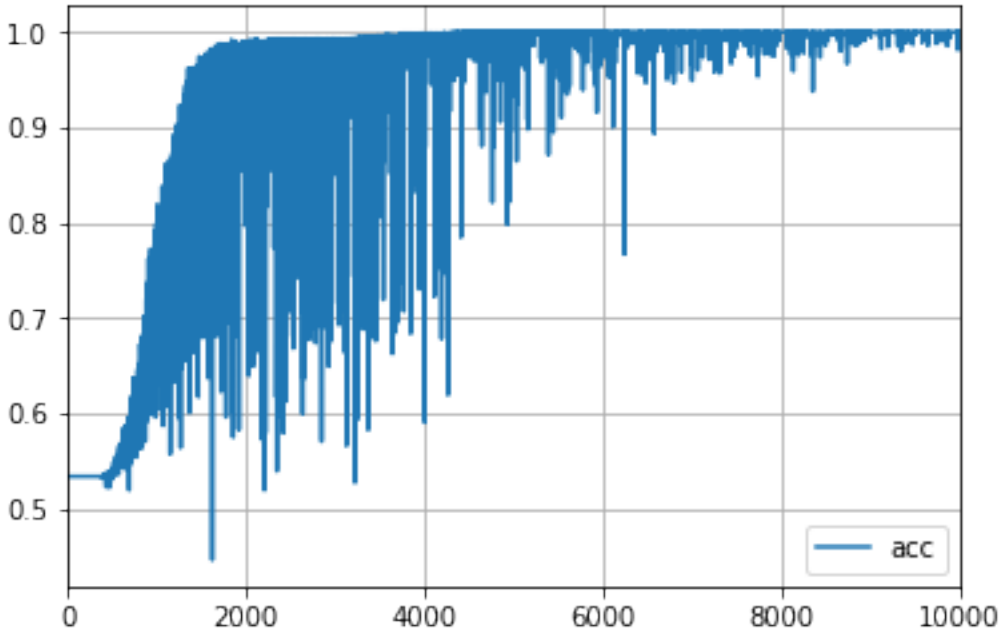

Figure 3: Predication accuracy corresponding to the number of Epochs

We find that the accuracy for epoch 1 starts with 53% . But as the number of Epoch increases, the accuracy increases, and at epoch around 3000, the accuracy becomes 100%. Thus the number of times the model goes through the dataset, it learns better.

4.3 When Batch size increases

We observe that smaller batch size leads to more computation memory and better accuracy. As the batch size increases the speed of the algorithm increases.

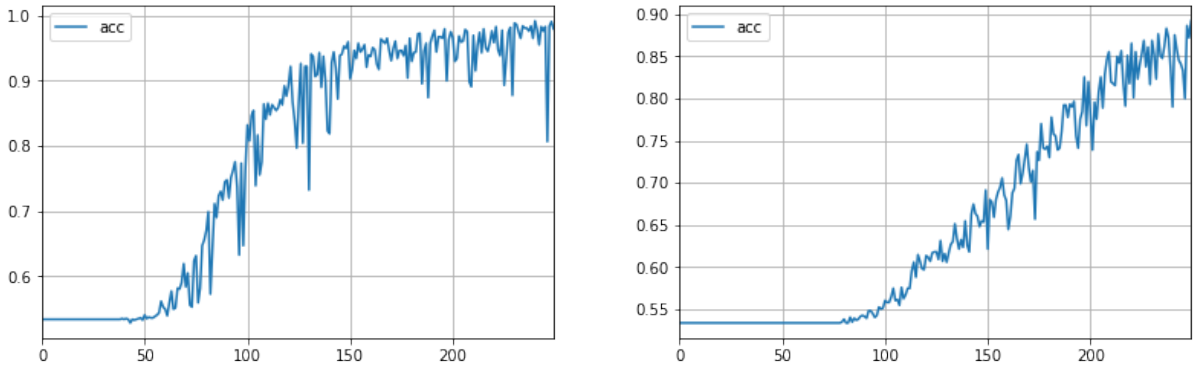

Figure 4: Predication accuracy for Batch size of 10 (left) and 20 (right)

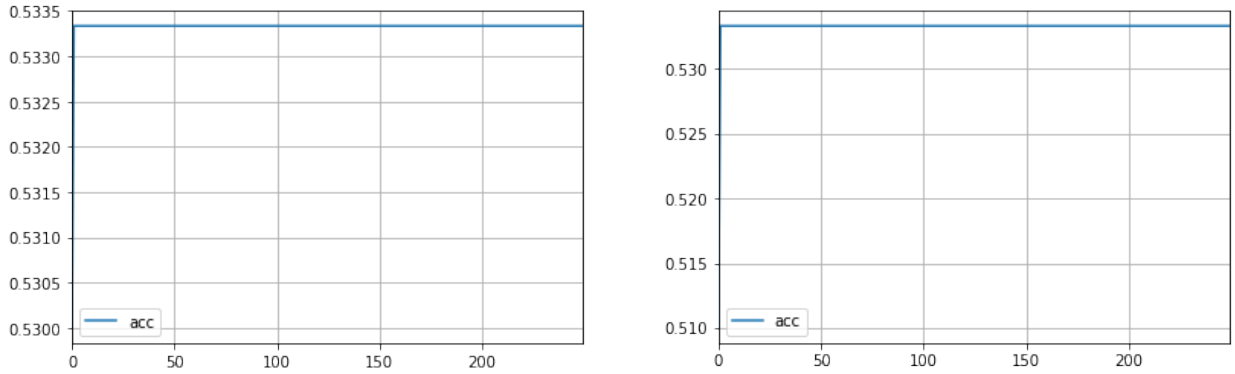

Figure 5: Predication accuracy for Batch size of 100 (left) and 250 (right)

We observe that Block size of 10/20, gives accuracy in the range of 90/80 percent respectively. But with larger block sizes of 100/250 we get lower accuracy. But choosing a very low block size might affect the computation speed, so a tradeoff between accuracy and speed must be chosen accordingly.

4.4 When learning rate changes

For very high learning rate (1.0) , the loss is high. Whereas, for lower learning rates, the loss decreases. But yet, for very low learning rate, the loss is very high, because the algorithm finds the best local gradient, which in turn is not the best solution. At an optimal learning rate, the accuracy is increased (0.1 to 0.5).

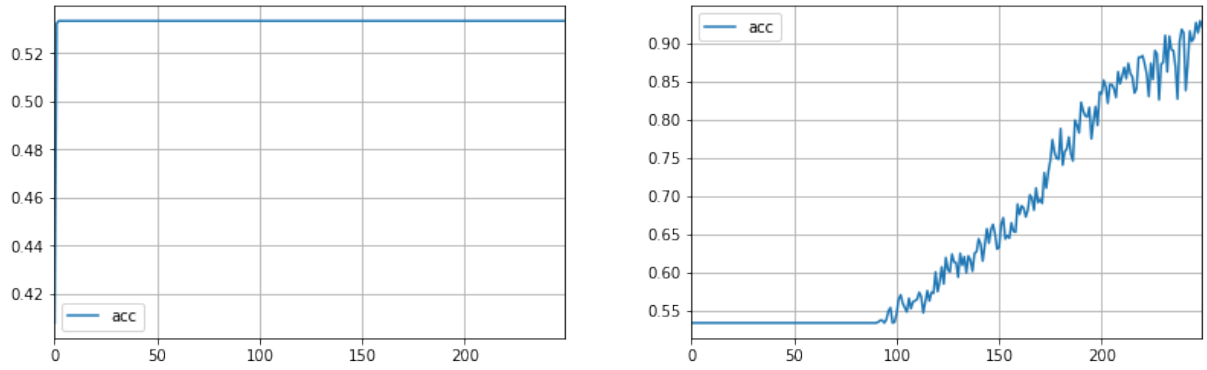

Figure 6: Predication accuracy for Learning rate of 0.01 (left) and 0.1 (right)

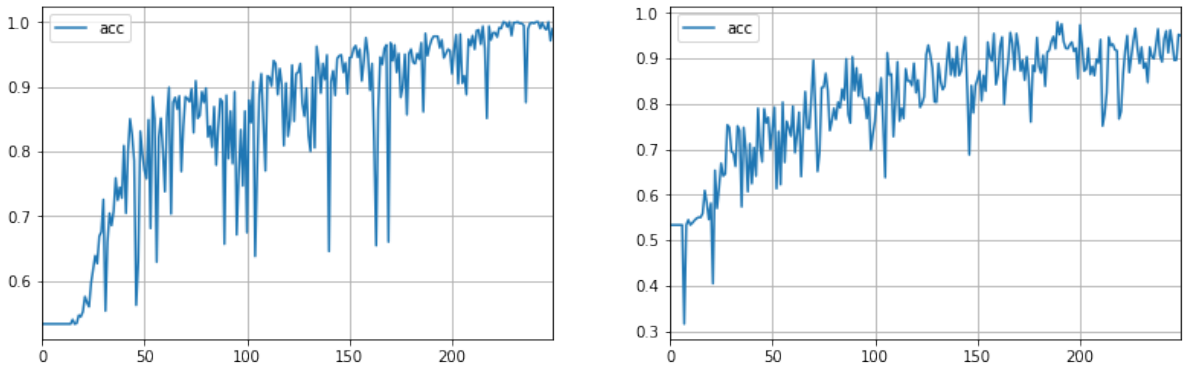

Figure 7: Predication accuracy for Learning rate of 0.5 (left) and 1.0 (right)

4.5 When hidden layers change

When the number of hidden layers increase their is no change, especially for this use-case. Also, increasing the layers, increases the chance of over-fitting as the effectiveness of back-propagation is reduced. We can observe that there is negligible improvement in accuracy.

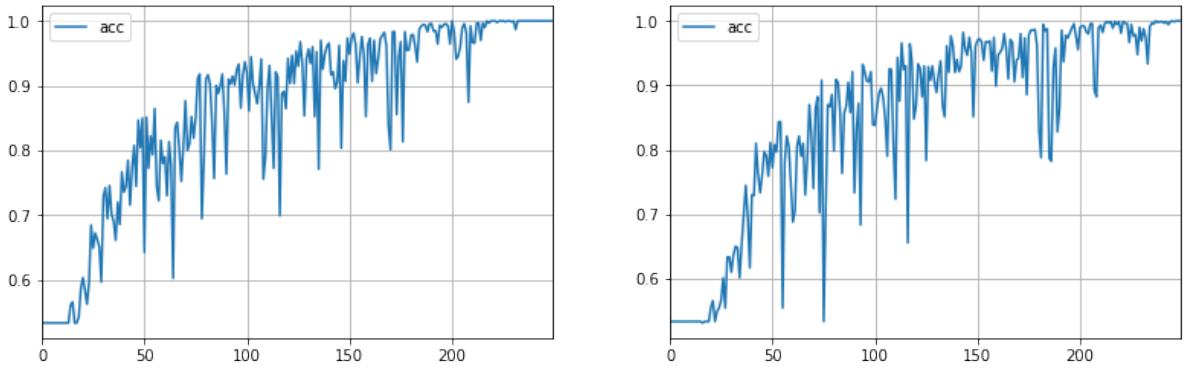

Figure 8: Predication accuracy for with 1 (left) and 2 (right) hidden layers

4.6 When number of nodes in the hidden layer changes

The number of nodes decide the level of features extracted from the image. With more number of nodes, the model can understand patterns better thus resulting in increased efficiency. For low nodes, the model does not predict (underfitting) and for high number of nodes the model begins to memorize the training sets and thus underperform (overfitting). This can be avoided by choosing the most optimal number of nodes (100 to 250).

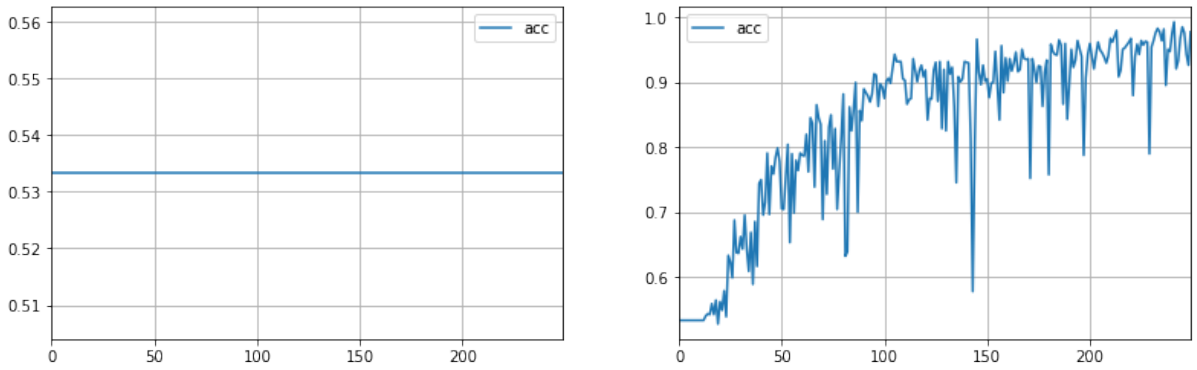

Figure 9: Predication accuracy for with 1 (left) and 100 (right) hidden layer nodes

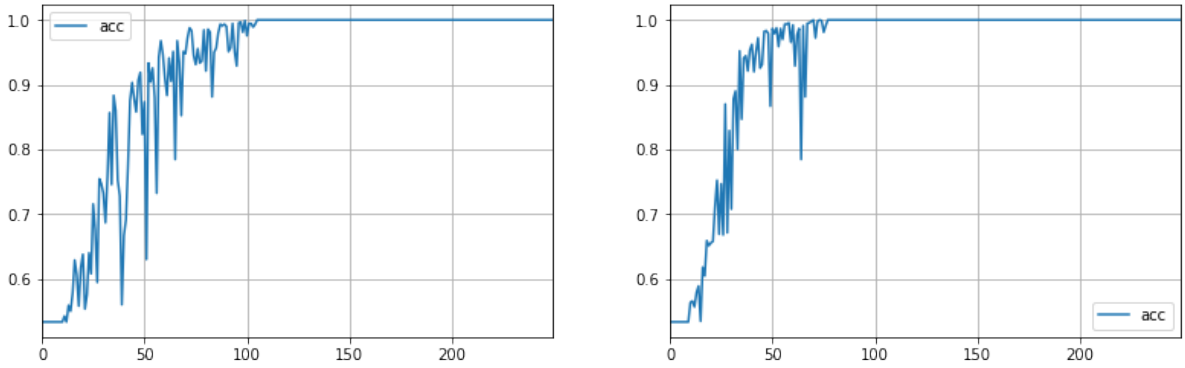

Figure 10: Predication accuracy for with 500 (left) and 1000 (right) hidden layer nodes

4.7 When input is increased

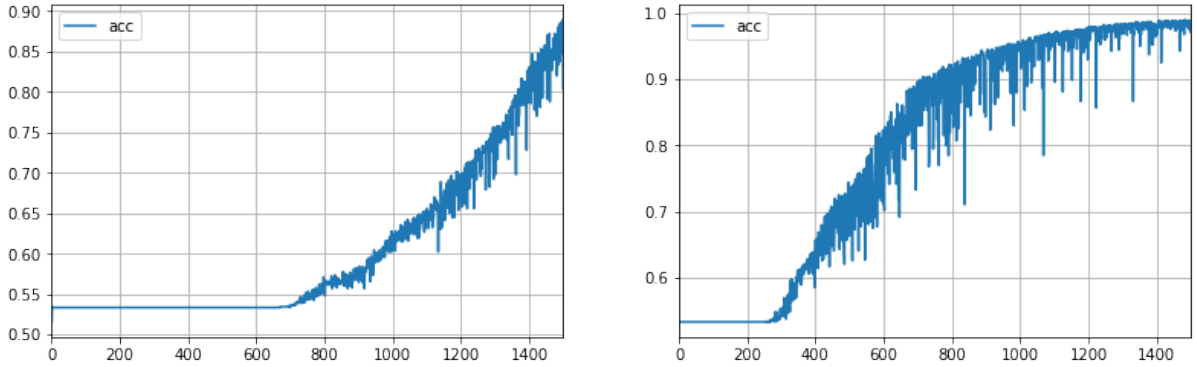

Figure 11: Predication accuracy for with input size 10 (left) and 12 (right)

When the input training dataset size is increased, we understand that the accuracy of the model increases. This is because the model has more attributes to form pattern. Here, for input size 10, the accuracy increases after 700 epochs, whereas for input size 12, the accuracy increases after 300 epochs.

4.8 Early stopping

Early stopping prevents overfitting. The model might perform really well in the train set, but its performace is bad in the test set, because of overfitting. Early stopping avoids overfitting.

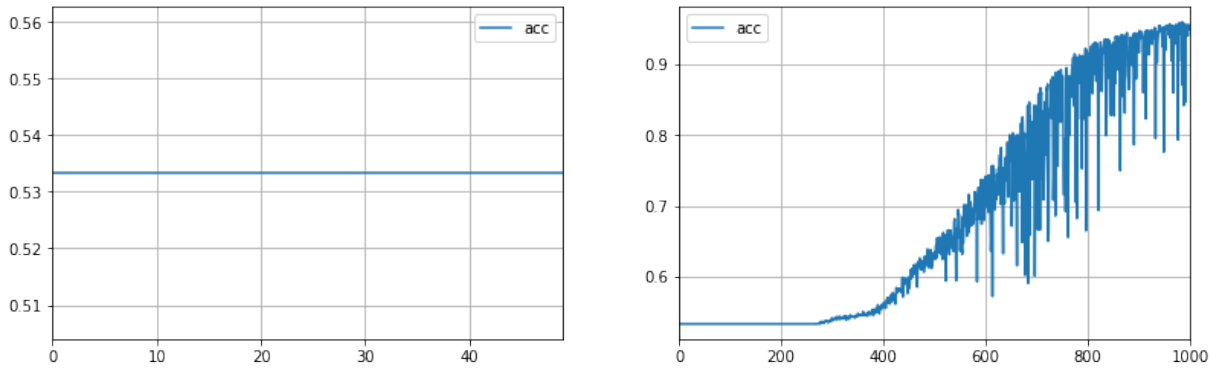

Figure 12: Predication accuracy for Early-stopping value is 50(left) and 1000 (right)

We find that very low early-stopping value, gives a model with low accuracy, because the model has not learnt enough. Very high early fitting rises over-fitting. So we must choose an optimal value for early-stopping.

4.9 Training in sequence

We also find that training the model with values in a random manner is more efficient compared to training in sequence. Though the initial accuracy is better for sequential model, as epoch increases, the accuracy for random model increases.

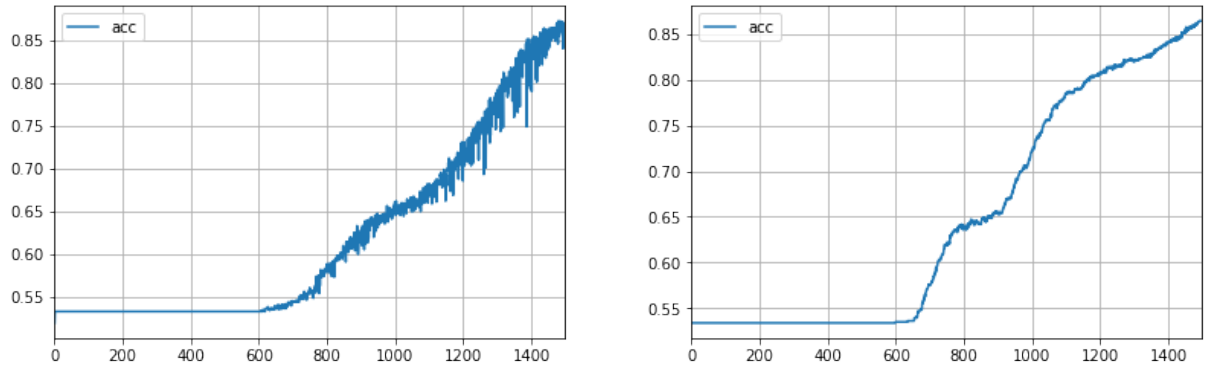

Figure 13: Predication accuracy for training in randomly(left) and sequentially (right)

4.10 Hard to predict

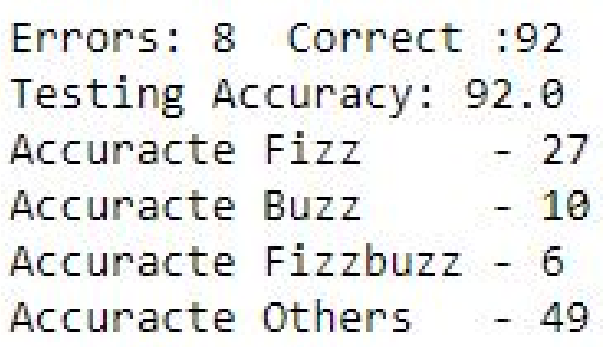

Figure 14: Predication accuracy results

Table 1: Accuracy for each Class

	Actual	Predicted
Fizz	27	27
Buzz	14	10
Fizzbuzz	6	6
Other	53	49

- We find that it is quite easy to predict the divisibility of 3, than 5. The reason lies in the fact that the numbers are in their binary format.

- But inorder to increase the accuracy, I tune the parameters further to achieve an accuracy of 100%

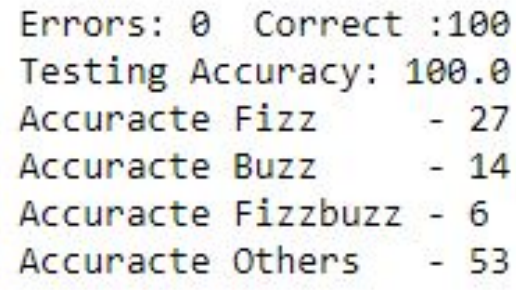

Figure 15: Predication accuracy results

Table 2: Accuracy for each Class

	Actual	Predicted
Fizz	27	27
Buzz	14	14
Fizzbuzz	6	6
Other	53	53

References

1. http://joelgrus.com/2016/05/23/fizz-buzz-in-tensorflow/
2. https://news.ycombinator.com/item?id=11753627
3. https://www.reddit.com/r/MachineLearning/comments/4kp658/tensorflow_fizzbuzz/
4. https://cedar.buffalo.edu/ srihari/CSE676/1.4.2%20Fizzbuzz.pdf
5. https://blog.algorithmia.com/solve-fizzbuzz-machine-learning-scikit-learn/
6. https://www.youtube.com/watch?v=BgBrYpihvLY
7. https://softeng.oicr.on.ca/minh_ha/2018/06/22/Tensorflow-fizz-buzz/
8. https://stats.stackexchange.com/questions/358311/is-it-possible-to-do-fizz-buzz-in-deep-learning
9. http://rickyhan.com/jekyll/update/2018/02/16/tensorflow-fizzbuzz-revisited.html