|  |
| --- |
| **Project 1.1 Implementation** |

**Amit Menon**

Department of Computer Science

State University of New York at Buffalo, NY

*amitanil@buffalo.edu*

**Abstract**

The aim is to compare the two approaches (Software 1.1 and Software 2.0) to solve the FizzBuzz problem and analyze the performance of the Software 2.0 solution in effect to changes in certain parameters

**1 Modification of Hyperparameters**

**1.1 Modification of Epoch and Batch Sizes**

Epoch is one complete cycle of the whole training data. Beginning with a lower value of 4000 we find that accuracy is not able to peak in 4000 iterations. That implies that the model was not able to train itself efficiently in said cycles. This is called underfitting of the model as shown in the below graph.

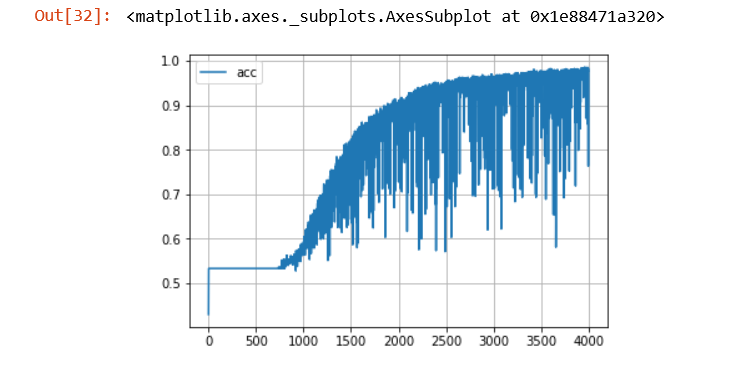


Figure .1 a)

As seen in the graph, the accuracy falters and rises quickly, and doesn’t max out completely reaffirming the observation, that the model isn’t well trained. Thus on increasing the epoch to 8000 we get Fig. 1.1. b). On this the graph the graph peaks out at 5000 and then radically thins down. This implies that at this point the model is trained to the extent that it won’t work on any data apart from training data. This is overfitting. Also its clear that that the optimum value lies in the region of 5000 iterations so taking epoch as 5500 we obtain Fig 1.1. c) and take epoch = 5500 as fixed value for modification of other parameters.

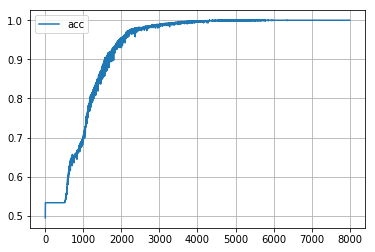


Figure 1.1 b)

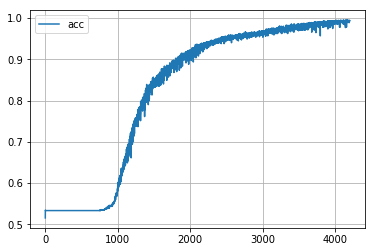


Figure 1.1 c)

Table :Accuracy results for variations on epoch

|  |  |
| --- | --- |
| **EPOCH** | **ACCURACY** |
| 5500 | 95 |
| 5500 | 95 |
| 5500 | 97 |
| 8000 | 94 |
| 8000 | 99 |
| 8000 | 96 |
| 4000 | 91 |
| 4000 | 98 |

Batches are the subset of data for which one sets of calculations are made for ease of calculation. Tweaking this value first to 90 gave Fig 1.1.d). The thickness of this plot implies that accuracy peaked and dropped quite a few times which implies that it was not optimally trained. So we increase the batch size to 150 to obtain Figure 1.1.e). This again reflects a thick plotting of the accuracy graph and doesn’t peak out, which implies that the batch size is greater than optimal

So we reduce the size again to 90 which gives result as Fig 1.1.f). Since this plots the thinnest curve, and converges correctly in the 5000 region itself, batch size of 100 is chosen as the optimum value

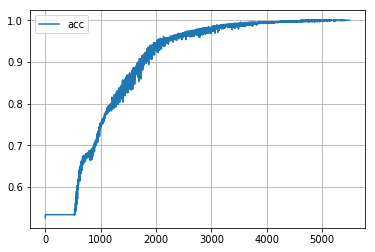


Figure 1.1 d) Batch size 90

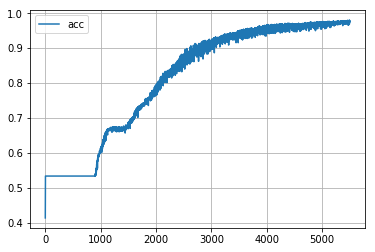


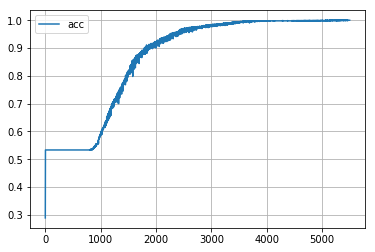
Figure 1.1 e) Batch size 150 

Figure 1.1 f) Batch size 100

Table : Comparison of Batch and Accuracy

|  |  |  |
| --- | --- | --- |
| **BATCH** | **ACCURACY** | **EPOCH** |
| 100 | 96 | 5500 |
| 150 | 80 | 5500 |
| 200 | 91 | 5500 |
| 90 | 92 | 5500 |

**1.2 Modification of values of Learning rate.**

We are using a stochastic gradient descent to correct our model and in that regards alpha is a factor that determines how quickly or slowly will our gradient descent converge to the minimum.

On initially tweaking the learning rate to 0.01, it is found that gradient descent took few number of iterations to even began converging to the local minimum as shown in the graph of accuracy vs iterations below:

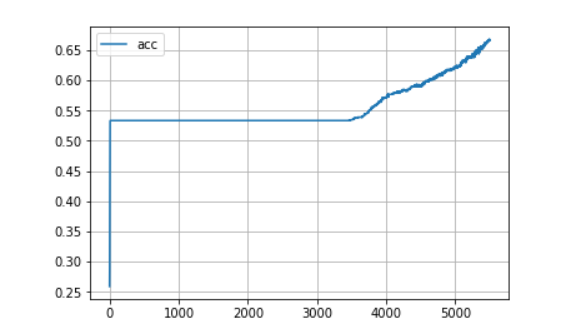


Figure 1.. a) Alpha 0.01

This is an indication that the learning rate is too slow. If the learning rate is too small, gradient descent will not converge in given number of iterations.

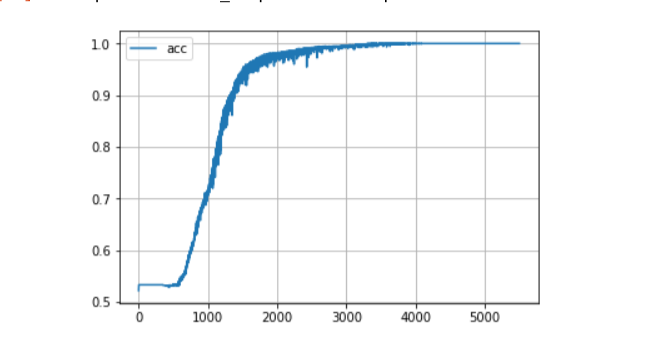


Figure 1.2. b) Alpha 0.08

On gradually increasing learning rate, gradient descent began to converge in less number of iterations until it takes extremely big jumps to converge and may skip the local minima altogether as shown in Fig 1.2 b). As we can see that in the plot the climb is way too steep which means gradient descent is taking the steps towards minima quickly in the right directions (less number of iterations).

**1.3 Number of Neurons nodes in Hidden Layer**

Number of nodes in the hidden layer is a very impactful factor in improving the model. As each node of a layer is an input to node in the next layer. As seen below in Fig 1.3.a) an increase in the number of nodes in the hidden layer makes the model reach greater accuracy in around 2000 iterations itself and without excessively thickening in between iterations. (Jumps are gradual in gradient descent).

The reason for this performance is because increase in number if nodes greatly increases the number of times cost computation is made at each layer and also allows gradients to back propagate in the network to influence values of the node in previous layer towards the value of corresponding node in output layer.

|  |  |  |  |
| --- | --- | --- | --- |
| **No of nodes** | **ACCURACY** | **EPOCH** | **BATCH** |
| 128 | 91 | 5500 | 100 |
| 128 | 92 | 5500 | 100 |
| 256 | 100 | 5500 | 100 |
| 256 | 99.1 | 5500 | 100 |
| 256 | 97.3 | 5500 | 100 |
| 256 | 97 | 5500 | 100 |

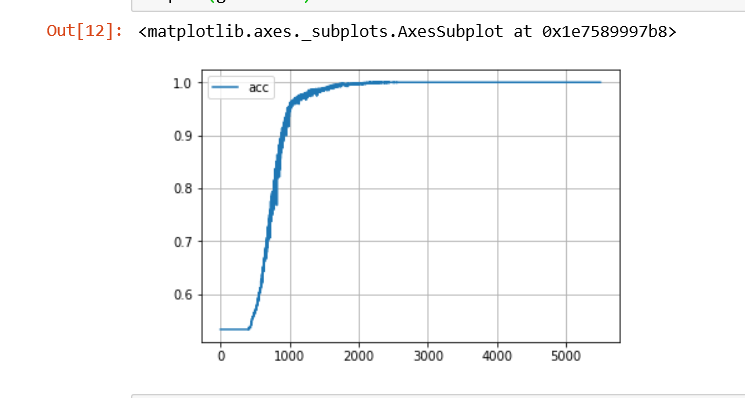


Figure 1.. a)

**3 Modification of Activation Functions**

We now analyze the performance impact when we change the activation function for the given hidden layer.

**3.1 Leaky\_relu**

The problem in using ReLu is that sometimes it can cause a node to die out (Dying relu problem). This happens when a large gradient back propagating can cause a node to be updated in such a way that it won’t fire at input point for any layer again. To solve this Leaky ReLu is used. Employing leaky relu as an activation function greatly improved performance of the model and the fluctuation in accuracy for different runs as seen below.

Table :Effect on Accuracy using Leaky Relu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No of nodes (Leaky Relu)** | **ACCURACY** | **EPOCH** | **BATCH** | **ALPHA** |
| 256 | 98 | 5500 | 100 | 0.2 |
| 256 | 98 | 5000 | 100 | 0.2 |
| 256 | 99.1 | 5500 | 100 | 0.2 |
| 256 | 93 | 5500 | 100 | 0.2 |
| 256 | 98 | 5000 | 100 | 0.5 |
| 256 | 100 | 5000 | 100 | 0.5 |

It is worth noting that tweaking the parameter “alpha” for leaky relu, further improves the accuracy though it is not completely reliant. Also increase in this alpha greatly increases computation time as not all of the nodes are fired. However if the value of alpha is increased even further then it results in over fitting as shown in figure 2 c) for which alpha = .75

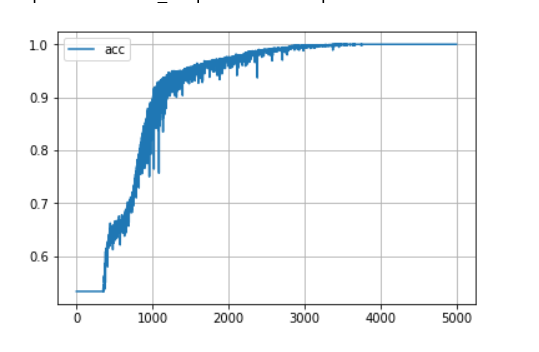
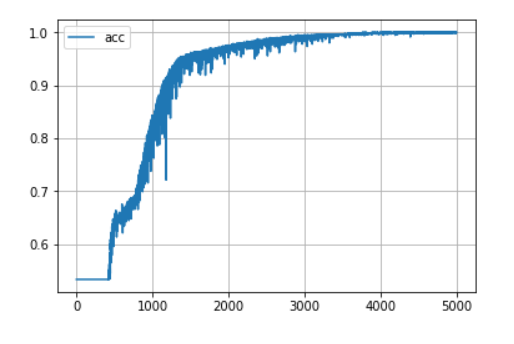


Figure 2 a) Leaky Relu

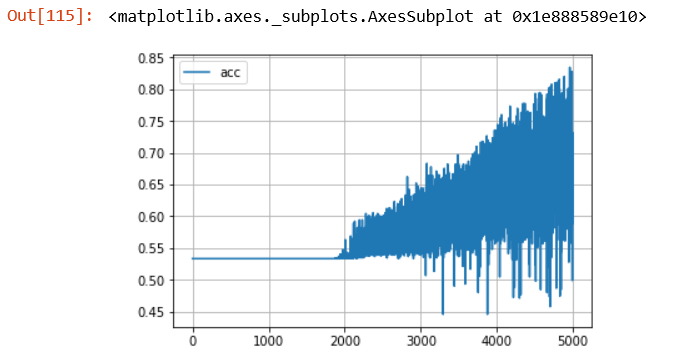


Figure 2 b) Leaky Relu overfitting

**4 Adding a dropout**

To prevent overfitting, we use the concept of dropout, which is essentially dropping outputs from nodes from hidden layer on the basis of certain probability which here is 0.99. Notice that with dropouts even as accuracy peaks, it doesn’t thin out completely which means that use of dropouts is allowing the model to generalize

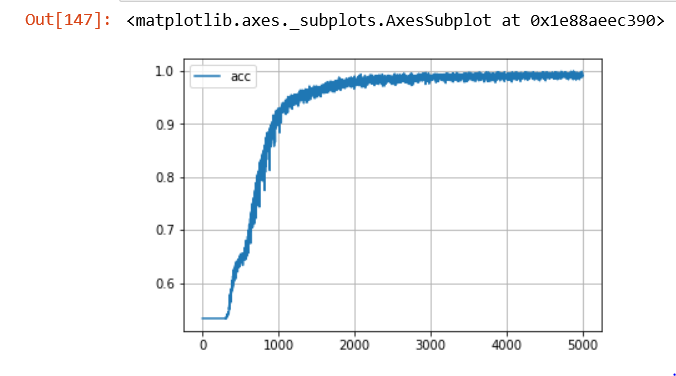


Figure Accuracy with dropout