Project 1.1: Software 1.0 Versus Software 2.0

Kishan Dhamotharan/Person No 50287619

September 18, 2018

Abstract

This report is a summary of my knowledge on how machine learning works and how different hyper parameters impact the performance of the learning program.

1 Introduction

This is a very basic observation I could infer by plotting loss and accuracy (testing/training) at each epoch and is displayed below. This is a very interesting series of graphs as one can see how the machine from a state of having 100% loss, learns gradually over time. Model tries to reduce the loss at each epoch, by calculating the gradient descent at each epoch and back propagates.

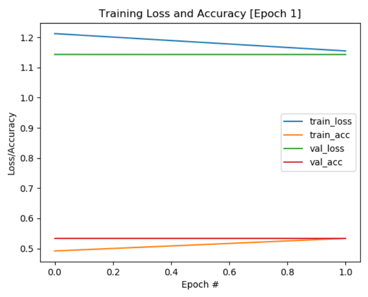

Figure 1: Epoch 1

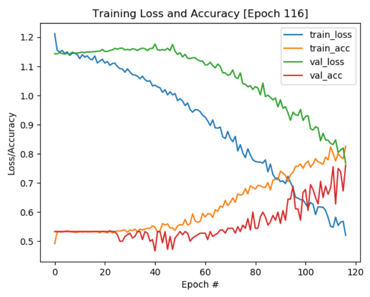

Figure 2: Epoch 116

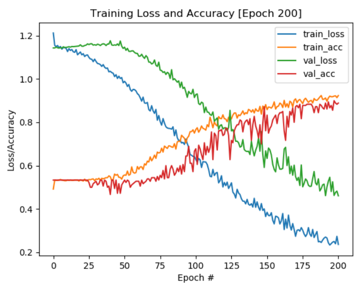

Figure 3: Epoch 200

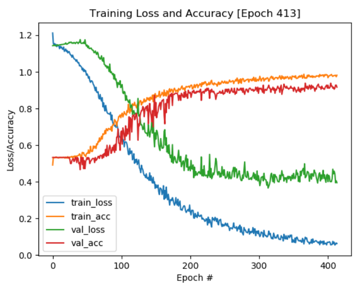

Figure 4: Epoch 413

2 Experiments and Interpretations

Following are the observation and inferences based on tweaking various hyper-parameters and model

2.1 Drop out

Drop out is used to randomly ignore few neurons during the training period, this makes the other neurons to step up and make up for the missing neurons and predict. Hence, the neural network as a whole won't be very sensitive to specific weights of neurons. Hence, achieving higher generalization and avoid over fitting. When we observe the below graph, we can see when the drop out is too little it has an accuracy of 98% and around 0.2 to 0.4 it has a stable plot and after which it strictly decreases. Things we infer from this are, the machine has over learnt when the dropout was 0.1 and when the dropout is high(0.6-0.9) it under learns.

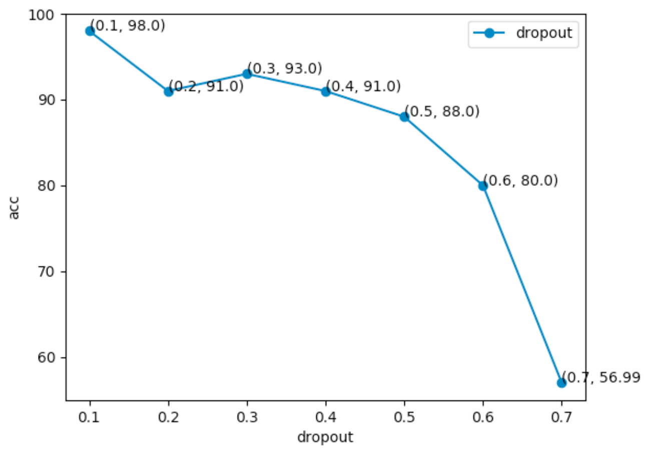

Figure 5: Raw (unprocessed) data. Replace this figure with the one you've made, that shows the resistivity.

2.2 Model batch size

Batch size is the number of sample that we pass to the model at one time. Batch size depends on the computation power of the CPU. Batch has effect on time efficiency of training. Choosing a very small batch size, the gradients found are very rough approx. of the actual ones, hence this will take a long time for the system to converge. Looking at the above graph we can not deduce any useful inference as there is only a mild fluctuation in the accuracy with change in the batch size changes, but the computation times were different. Smaller batches took more time as the to complete one epoch it needed more batch, hence making each epoch longer.

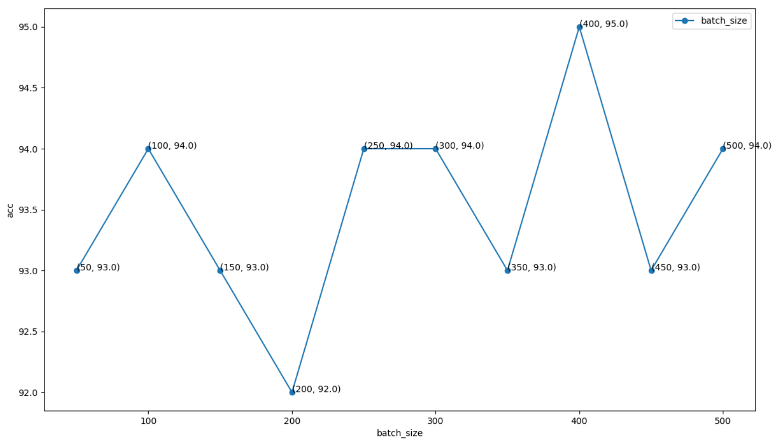

Figure 6: To observe the change in validation accuracy of the model batch size increases.

2.3 Num epochs and Early Stopping

Here we can observe that after a certain epochs there is not much change in the accuracy, this is because of the use of early-stop callback. Early stopping is a way to stop training when your loss starts to increase (decrease in validation). Early stopping also ensures that model does not overfit and also reduces the computation , by effectively choosing to stop before the actual no. of epochs (If the loss is stable for a while or loss increases).

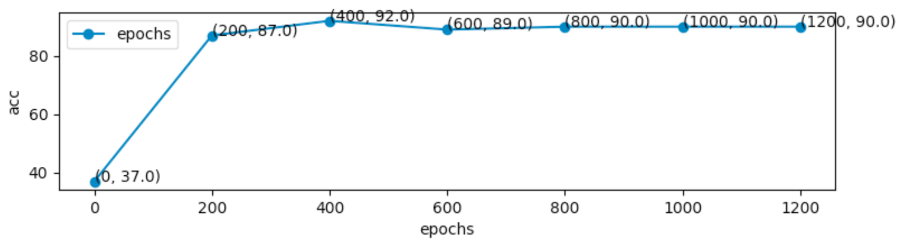

Figure 7: To observe the change in validation accuracy of the model when epochs increases

2.4 Learning rate

The below two graphs are plotted with two different learning rate, here we can observe a very interesting pattern. Observe the validation loss graph of the model with higher learning rate, we can see that the graph is zigzag with a lot of noise, reason being larger gradient descent at each epoch, which was because of the large learning rate. And if the learning rate is too high it might fail to converge at all. Too small learning rates will increase the computation time(will need more epoch as well).

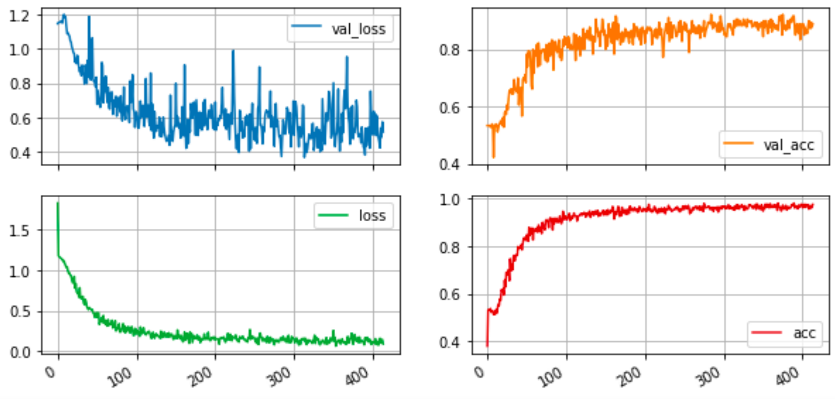

Figure 8: Learning rate: 0.01

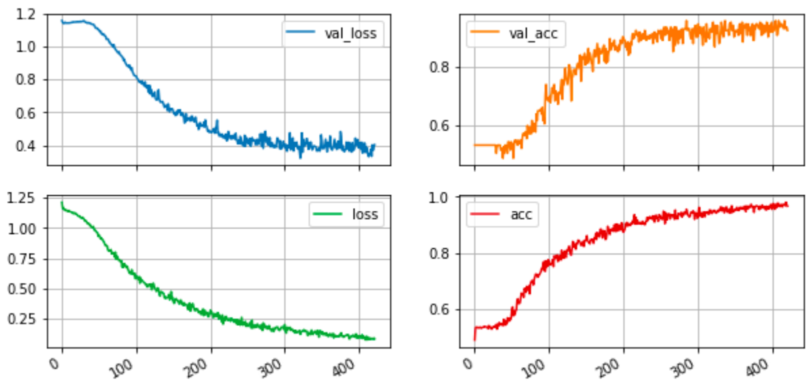

Figure 9: Learning rate: 0.001

2.5 Number of dense layers

We can observe in the below graph that adding certain number layer improves the performance. Adding more layers will help us to extract more features. But we can do that up to a certain extent. After that, instead of extracting features, we tend to over fit the data. Over fitting can lead to errors in some or the other form like false positives. For example, We start training our model for detecting gardens, but few images had pictures of gate in them. Later if the model gets a gate , it will be classified as garden. Adding more and more layers might improve the accuracy in the training and cross validation data. But this might have a very bad impact when a unseen data comes up.

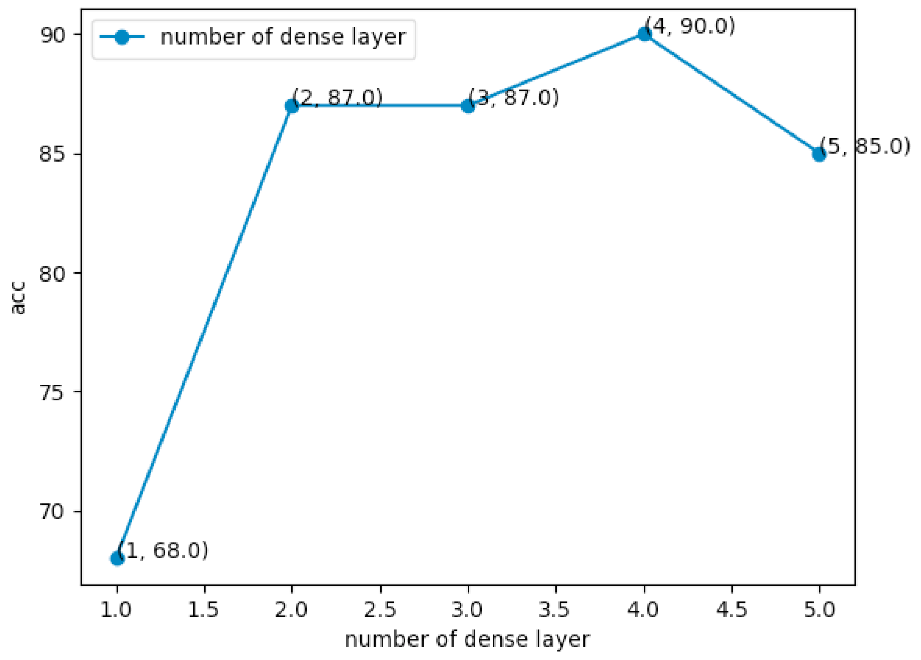

Figure 10: Impact of accuracy on increasing the number of dense layer

2.6 Different Activation functions with Different Optimizer

Why not Sigmoid? As FizzBuzz is a multi class classification problem we will needing probability of each class, but sigmoid just tells 0 or 1. Does not fit our model. Other Learnings - Sigmoid brings in the nonlinearity and results in 0 to 1 gives the probability of the output. But the drawback is that while calculating back-propagation, when the derivative is taken only a very small fraction of the error is sent back to the previous layer. Therefore taking lots of epochs to get better results.

Why not Tanh? Has the range between -1 to 1. When experimented with tanh, it does work well for our model. Even tanh is a good option to use.

Why Relu? Relu's gradient function returns true 0 for n¡0 and 1 for n¿1. This ensures that gradient is never lost. Relu too has disadvantage , the relu's range is from 0 to infinite it updates all the negative values to be 0, leaky Relu sloves this problem.

The below graph illustrates the performance of different activation functions with different optimizers. It can be observed that relu with rmsprop, nadam and tanh with nadam perform well in our case. Sigmoid and linear perform the least.

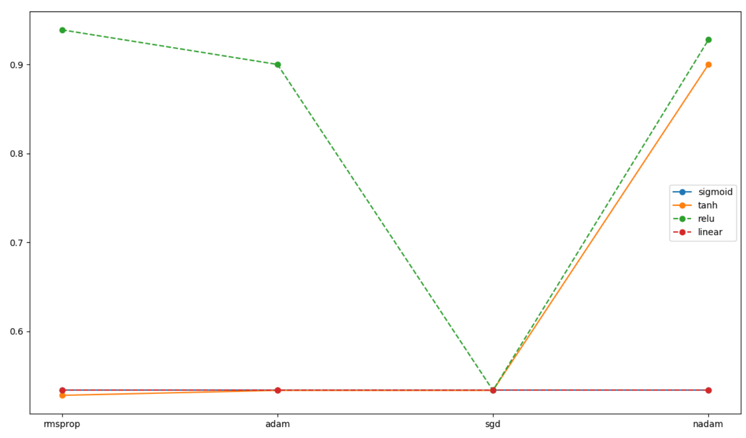

Figure 11: Accuracy with different combination of optimizers and activation functions

3 Results

3.1 Optimal hyper parameters

I could achieve 97% accuracy with the below values of hyper parameter. Failure/Accuracy: Buzz - 2 Failure
Fizz - 1 Failure
FizzBuzz - 0 Failure

Hyper-Parameter	Optimal Value
Drop out	0.31
Optimizer	Adam
Model batch size	128
No of Epochs	700
First Dense layer nodes	256
Second Dense layer nodes	256
Loss Function	categorical cross entropy
Learning Rate	0.001
Activation function	Relu, Softmax