Deep Learning in Data Science

Rakin Ali 2023-04-02

Numerically vs Analytical Gradients:

Calculating Gradients can be a complicated process and painstakingly slow. In this lab, an analytical calculation of gradients was used to speed up the process of calculating gradients. The analytical gradient calculation was heavily inspired by lecture 3 slides 101 to 104. The gradient's numerical calculation was done using forward difference and was modified from the 'functions.py' python file given in the assignment description. Since the gradients with respect to weight and bias respectively are matrixes with sizes k x d and K x 1, comparing them was a bit difficult. Taking the norm of respective matrixes and then taking the difference was the most reasonable way to calculate the difference in precision. Below is a table where the norm of the difference with different Batch sizes. Bear in mind that the larger the batch size, the larger the difference should be. A difference of $|10^{-6}|$ and lower was deemed to be acceptable.

	Batchsize = 100	**Batchsize = 1000**
Difference in Weight	4.878786442536142e-07	5.59574201983569e-07
Difference in Bias	7.452809265845332e-07	2.4636875418254394e-06

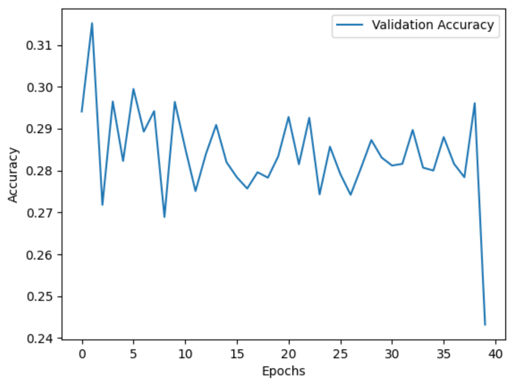
Table 1: Difference in weight and bias calculations by analytical and numerical gradient calculations

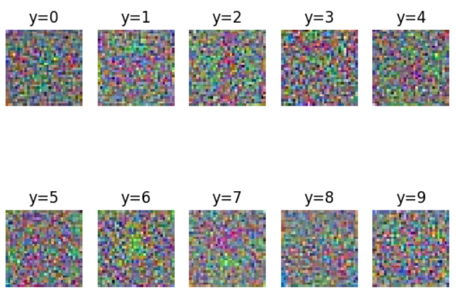
From the table above, we see that the difference between analytical or forward difference numerical calculations is very close, meaning the analytical gradient method can be used in this assignment. In hindsight, I should have tried out different batch sizes and different segments of the data as this was only on the data from 0-100 and 0-1000. Creating a shuffling method for the data and then the difference in weight and bias would have been better however by the time I realised this, I was writing my report with the deadline nearby.

Different Parameter settings:

A mini-batch gradient descent function was created in this assignment to train a model of how to differentiate between 10 different classes. Below are the results. Observe that when Lamda = 0 then the loss and cost graphs are exactly the same. When Lamda is not zero then observe how the values in the y-axis differentiate.

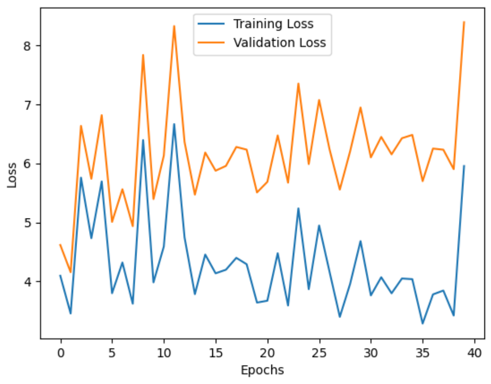
Lamda = 0, Epochs = 40, Batchsize = 100, learning rate = 0.1

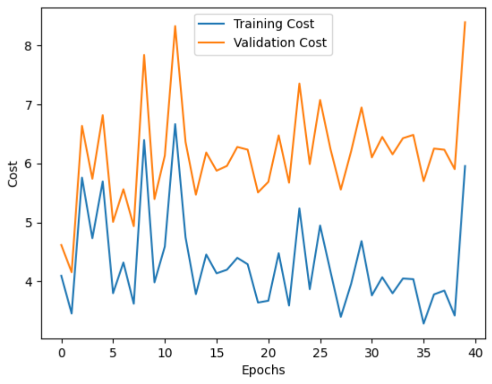




Validation accuracy **Weights visualisation**

Figur 1: Shows the validation accuracy and the visualisation of weights



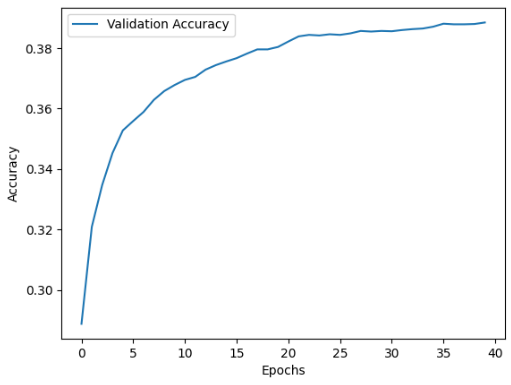


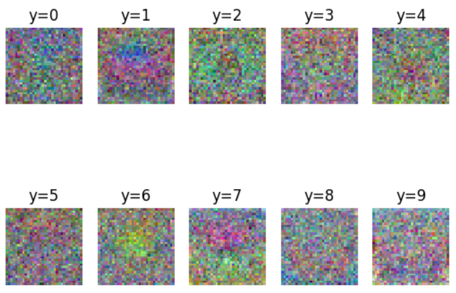
Training and validation loss Training and validation cost

Figur 2: Training and validation loss(left) and cost(right). Since Lamda= 0 they are the same

Comment: Cost and validation loss are the same since the lambda value is 0 meaning the “extra weights” are not punished, in other words, there is no regularisation being performed. On top of this, the learning rate (*eta in the assignment description)* has a large value which means the model takes drastic changes during training which causes large variations over the epochs. This could explain why the training loss and accuracy fluctuates and oscillates between improving and being worser.

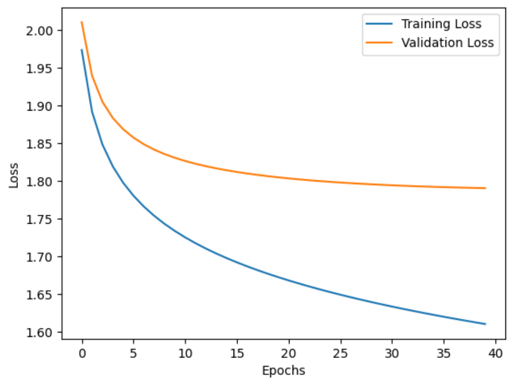
<u>Lamda = 0, Epochs = 40, Batchsize= 100, Learning rate = 0.001</u>

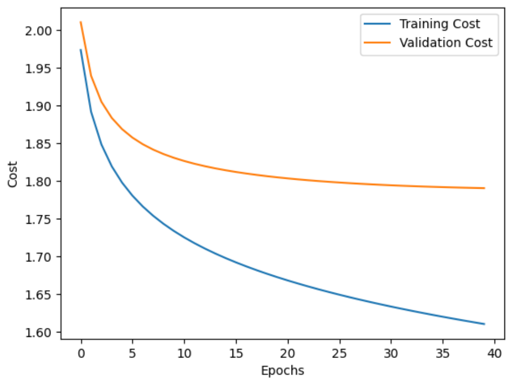




Validation accuracy **Weight visualisation**

Figure 3: Shows the validation accuracy and the visualisation of weights



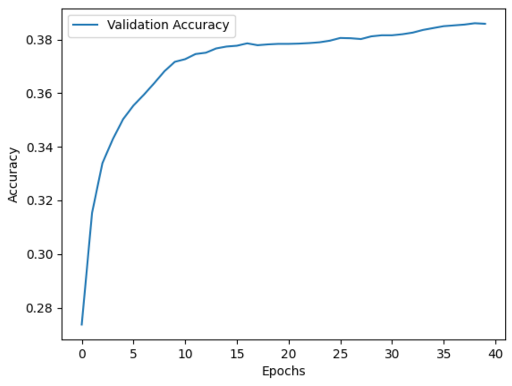


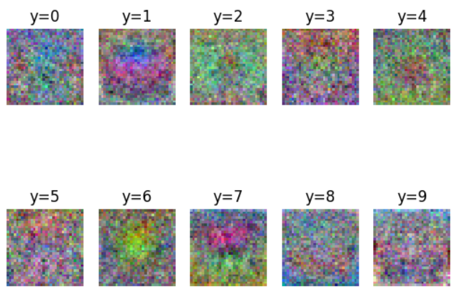
Training and Validation Loss **Training and Validation cost**

Figure 4: Training and validation loss and cost. Since Lamda= 0 they are the same

Comment: The graphs, accuracy and training & validation cost/loss seems more smooth and generally better. Significantly reducing the learning rate provided better results. Both validation loss and cost are the same again because no regularisation is performed.

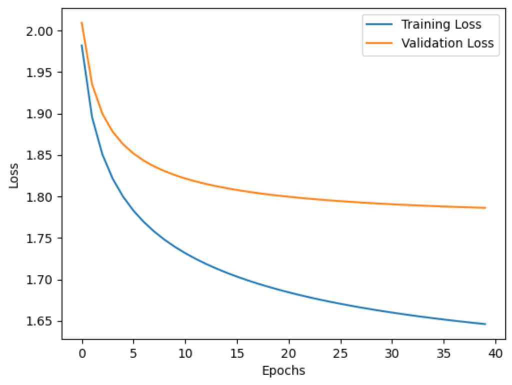
Lamda = 0.1, Epochs = 40, Batchsize = 100, Learning rate = 0.001

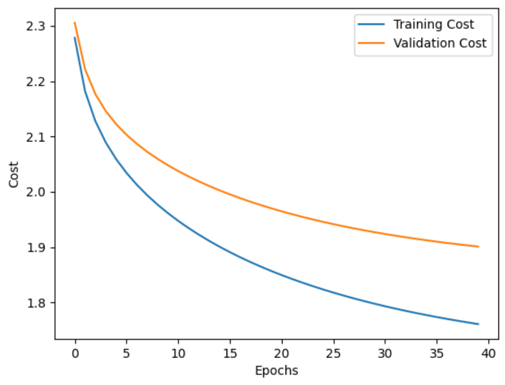




Validation accuracy **Weight visualisation**

Figure 5: Shows the validation accuracy and the visualisation of weights



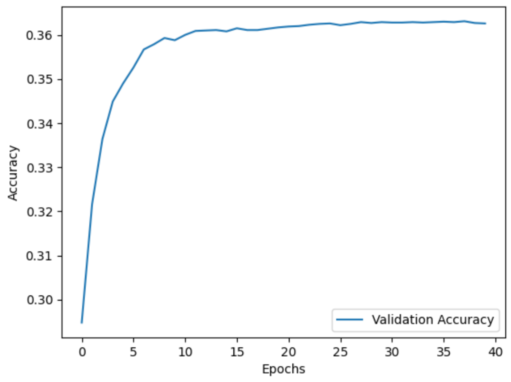


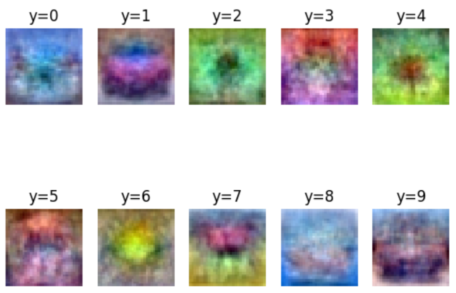
Training and Validation Loss **Training and Validation cost**

Figure 6: Training and validation loss and cost.

Comment: The visualisation of weights looks a lot more clear. The images are differentiatable, meaning some of the pictures are easy to differentiate from others. Since Lamda is a small number, there is a small difference between training/validation loss compared to training/validation cost. The cost, obviously, will be a bit higher as the adjustment of weights are punished (regularisation being performed) and so the optimisation process is trying to find the optimal weights that does not cost too much. Since lamda is small, the variance will be large but not gigantic as in the previous settings. Also because learning rate is small, the “changes” implemented will not be drastic either meaning the model adjusts better.

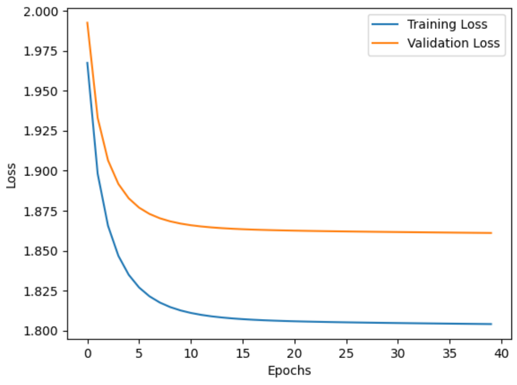
Lamda = 1, Epoch = 40, Batchsize = 100, Learning rate = 0.001

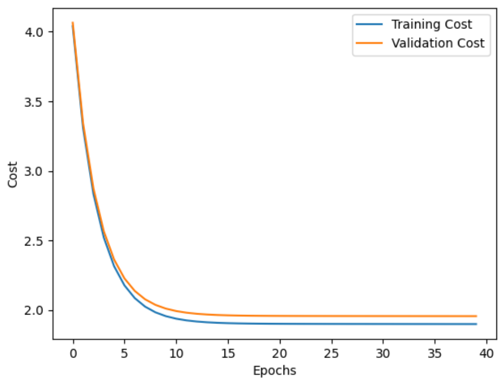




Validation accuracy **Weight visualisation**

Figure 7: Shows the validation accuracy and the visualisation of weights





Training and Validation Loss **Training and Validation cost**

Figure 8: Training and validation loss(left) and cost(right).

Comments: The images in the weights looks much more clear and differentiable. The training/validation costs stables out around 2 which is roughly at 10 epochs. The training and validation loss also stables at around 10 epochs. Unfortunately the accuracy is not the highest compared to settings 2 and 3. That being said, the difference in accuracy is roughly 2%.

Conclusion:

From the different parameter settings. A few things was observed and noticed.

A high learning rate means the model takes larger steps towards the solution however that comes with pros and cons. It can lead to faster convergence as the model takes larger step which can be good if the data is large. It also could potentially help the model to escape local minimas which a low learning rate could get stuck in. Hoever it could also lead to overshooting which is when the model 'overshoots' the optimal solution which can lead to unstable and unpredictable behaviour. This was seen in figure 1 when the learning rate was 0,1. This could explain why the model's accuracy oscillated. Learning rate is dependent on the specific problem and should be selected carefully through trial and errors. There are many reports and field of studies conducted.

Regularisation prevents a model from overfitting by trying to make the weights smaller. This could improve generalisation of the model on unseen data. Also it makes the weights more interpretable by reducing the impact of noise and irrelevant features which can help identify the most important features in the data. This was seen in figure 7 settings 4 where the lamda value was the highest; the images became more clear and more intretable however the accuracy was not the highest in that settings. I am assuming that a high lamda value can make optimisation problems more complex and computationally expensive as I had a lot of errors when coding the gradient functions where lamda had a value higher than 0.

A conclusion on whether a batchsize and epochs matter cannot really be drawn based on 4 different parameters settings where they were essentially the same. An assumption can however be made on the amount of epochs. In almost all parameter settings, at around 20 epochs, the model stabilised and the "learning" and accuracy improvement was not as steep as in the earlier epochs. I believe the armount of epochs needed to reach an optimal solution could be reliant on the learning rate and weights. If a model learns really slow then a large amount of epochs would be needed.