DD2424 Assignment 3

This is the report for assignment 3 in course dd2424 Deep Learning in Data Science. Mostly about the multi-layer neural network with cifar-10. The code is written in Matlab.

Gradient Check

Without Batch Normalization

I successfully managed to compute the gradient analytically by computing the absolute difference between this result and the result from centered difference formula up to 5 layers. The difference is less than 1e-6.

The function for computing gradient analytically is called *Computegradients*, function for computing gradient numerically is called *ComputeGradsNumSlowOld* and the function I wrote to compare the gradient between analytical and numerical computation is called *CheckingGradient* in the script.

With Batch Normalization

I also successfully managed to compute the gradient of weight, bia and normalization parameters analytically by computing the absolute difference between this result and the result from centered difference formula up to 5 layers. The difference is less than 1e-6.

The function for computing gradient analytically is called *ComputegradientsBN*, function for computing gradient numerically is called *ComputeGradsNumSlow option1* and the function I wrote to compare the gradient between analytical and numerical computation is called *CheckingGradient BN* in the script.

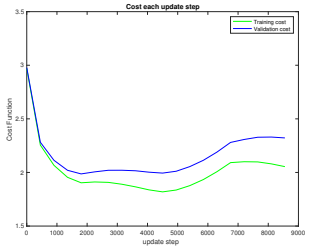
3-layer Network

In this section, I tried to run a 3 layer network to train the model with and without batch normalization. The hyper-Parameters are set to be default, which are eta_{min} = 1e-5, eta_{max} = 1e-1, λ=.005, two cycles of training and n_s = 5 * 45,000 / n_{batch} with the *epoch* is 20. The number of nodes are both 50. Here I used He Initialization.

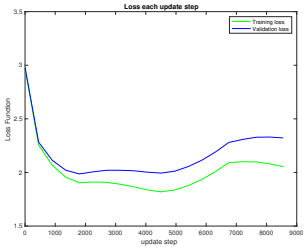
Learning outcomes:
Author(s): Chia-Hsuan Chou

Without Batch Normalization

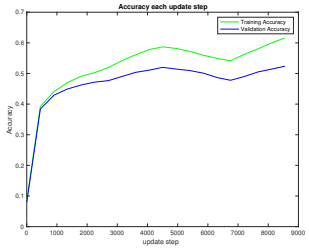
The accuracy in test data is around 53.47%. The following figures show the cost, loss, learning rate and accuracy after each epoch.



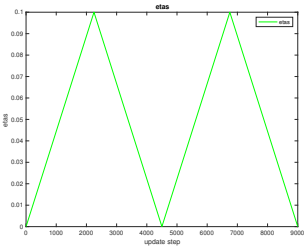
(a) Cost Function



(b) Loss Function



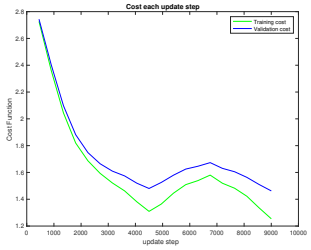
(a) Accuracy during Training



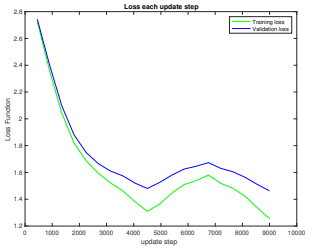
(b) Learning Rate

With Batch Normalization

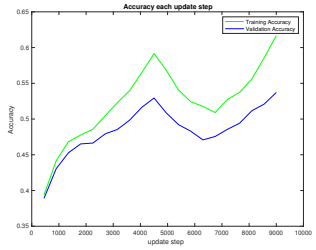
The accuracy in test data is around 52.9%. The following figures show the cost, loss, learning rate and accuracy after each epoch.



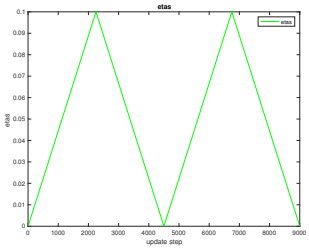
(a) Cost Function



(b) Loss Function



(a) Accuracy during Training



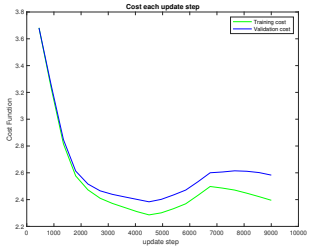
(b) Learning Rate

9-layer Network

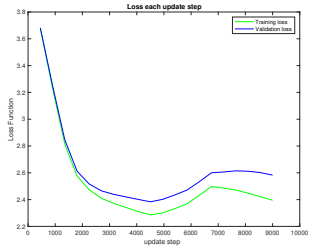
In this section, I tried to run a 9 layer network to train the model with batch normalization. The hyper-Parameters are set to be default, which are eta_{min} = 1e-5, eta_{max} = 1e-1, $lambda$=.005, two cycles of training and n_s = 5 * 45,000 / n_{batch} and $epoch$ is 20. Here I used He Initialization and the nodes in each layers are 50,30,20,20,10,10,10,10.

Without Batch Normalization

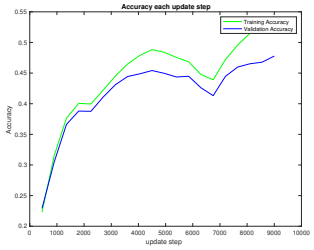
The accuracy in test data without batch normalization is around 47.71%. The following figures show the cost, loss, learning rate and accuracy after each epoch.



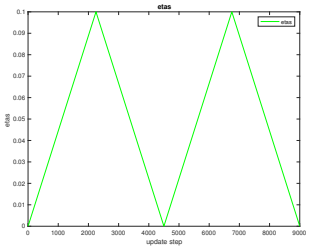
(a) Cost Function



(b) Loss Function



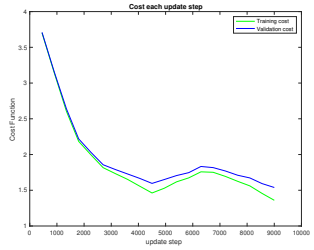
(a) Accuracy during Training



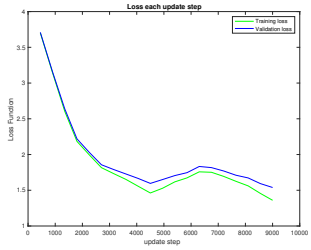
(b) Learning Rate

With Batch Normalization

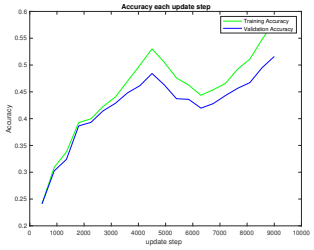
The accuracy in test data is around 51.42% with 2 cycle learning rate. The following figures show the cost, loss, learning rate and accuracy after each epoch.



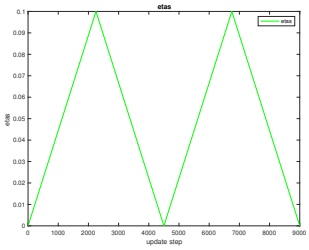
(a) Cost Function



(b) Loss Function



(a) Accuracy during Training



(b) Learning Rate

Coarse-to-Fine Search

In this section, I will find a proper regularization parameter λ in 3 layers network with batch normalization. After finding a good parameter, I will find a better

range of learning rate and then try to play with different number of cycles to get better results. The parameters are set as default but not using any method of initialization. The first table shows my search on λ with eta_{min} = 1e-5, eta_{max} = 1e-1 and 2 cycle of learning. The table is ordered by the accuracy.

Range	λ	acc
1e-3 to 1e-2	0.0051	53.09%
1e-5 to 1e-2	0.0077	52.98%
1e-3 to 1e-2	0.0030	52.26%
1e-5 to 1e-1	1.1450e-04	51.83%
1e-5 to 1e-3	3.3738e-04	51.64%
1e-5 to 1e-4	6.3570e-05	51.54%
1e-4 to 1e-3	1.0647e-04	51.30%

Table 1: coarse-to-fine search for λ

From the table 1, I found that when λ is in the range of 0.001 and 0.001, the accuracy on test data has better performance. Therefore, I am going to use the best λ I have found, that is, λ equals to 0.0051, for my next search on the *eta*. The second table shows the search of range of learning rate *eta* with 2 cycle learning rate:

eta_{min}	eta_{max}	acc
1e-5	1e-1	53.09%
1e-5	1e-2	51.83%
1e-5	1e-3	47.16%
1e-5	1e-4	32.36%
1e-4	1e-1	53.15%
1e-4	1e-2	51.24%
1e-4	1e-3	47.89%
1e-3	1e-1	52.44%
1e-3	1e-2	50.67%
1e-2	1e-1	52.10%

Table 2: coarse-to-fine search for eta_{min} and eta_{max}

From the second table, I found that with *eta* is between 1e-4 and 1e-1, the accuracy is really high in test data. Therefore, I am going to use this range of *eta* to experiment further, and get the three best performance for two difference initialization:

- 2 cycles, default setting with 3 layers. Numbers of nodes in each layers are 150 and 50. λ = 0.0051, *eta* range found above with accuracy equals to 55.28 %.(Using Xavier Initialization).

- 2 cycles, default setting with 3 layers. Numbers of nodes in each layers are 150 and 50. $\lambda = 0.0051$, *eta* range found above with accuracy equals to 55.13 %.(Using He Initialization).
- 4 cycles, default setting with 3 layers. Numbers of nodes in each layers are 100 and 50. $\lambda = 0.0051$, *eta* range found above with accuracy equals to 54.53 %.(Using He Initialization).
- 2 cycles, default setting with 3 layers. Numbers of nodes in each layers are 100 and 50. $\lambda = 0.0051$, *eta* range found above with accuracy equals to 54.51 %.(Using Xavier Initialization).
- 4 cycles, default setting with 3 layers. Numbers of nodes in each layers are 100 and 50. $\lambda = 0.0051$,*eta* range found above with accuracy equals to 54.5 %.(Using Xavier Initialization).
- 2 cycles, default setting with 3 layers. Numbers of nodes in each layers are 100 and 50. $\lambda = 0.0051$, *eta* range found above with accuracy equals to 53.72 %.(Using He Initialization).

Since I didn't reach 53.5% as described in the instruction every time with number of nodes 50 for both, I increased the number of node in first layer and made sure the model didn't overfit, the accuracy was higher and even reached 55% in test data.

Sensitivity Initialization

To compare the performance with different initialization and the difference between model with and without batch normalization, I will train the model with He Initialization, Xavier Initialization and no initialization (with variance eqauls to 0.01) and compare the difference among them.

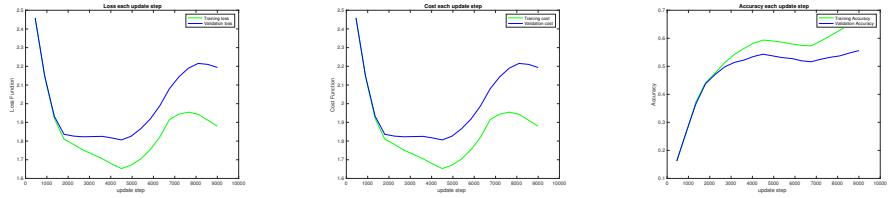
I will train models with 3-layers network with number of nodes are 100 and 50, respectively. The λ is set as 0.0051, eta_{min} is 1e-4, eta_{max} is 1e-1,n_{batch} is 100, *epoch* is 20 and n_s is 5 x $45000/n_{batch}$, which is 2 cycles learning.

Without Batch Normalization

Table 3 shows the comparison between different initialization:

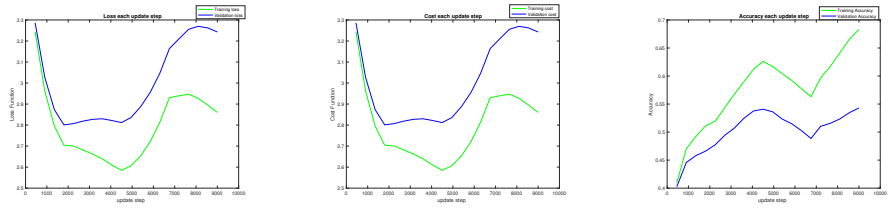
Initialization	Train Acc	Test acc
var = 0.01	66.67%	53.12%
He Init.	68.32%	53.48%
Xavier Init.	68.44%	53.92%

Table 3: The comparison of results between different initialization without Batch Normalization



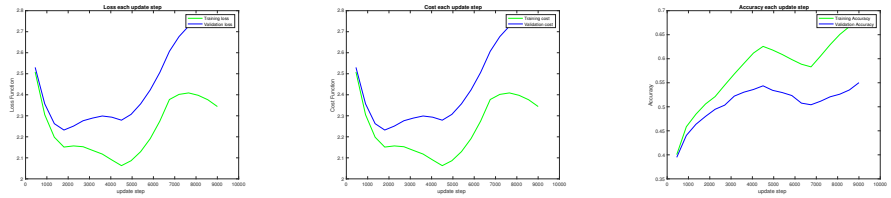
(a) Loss (b) Cost (c) Accuracy

Figure 9: Initialization with var = 0.01 in case without Batch Normalization



(a) Loss (b) Cost (c) Accuracy

Figure 10: He Initialization in case without Batch Normalization



(a) Loss (b) Cost (c) Accuracy

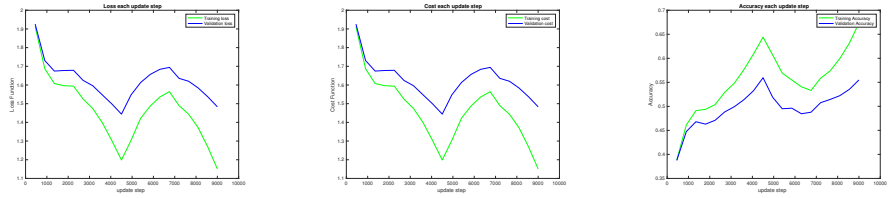
Figure 11: Xavier Initialization in case without Batch Normalization

With Batch Normalization

Table 4 shows the comparison between different initialization:

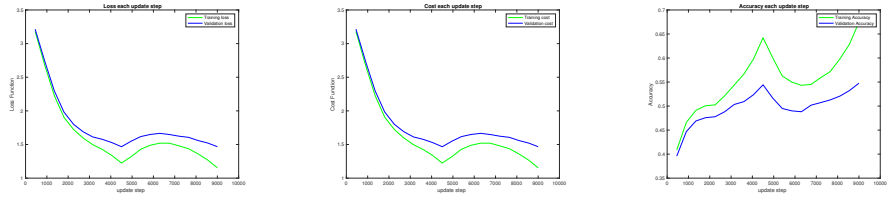
Initialization	Train Acc	Test acc
var = 0.01	66.36%	54.22%
He Init.	67.46%	54.01%
Xavier Init.	67.38%	54.63%

Table 4: The comparison of results between different initialization with Batch Normalization



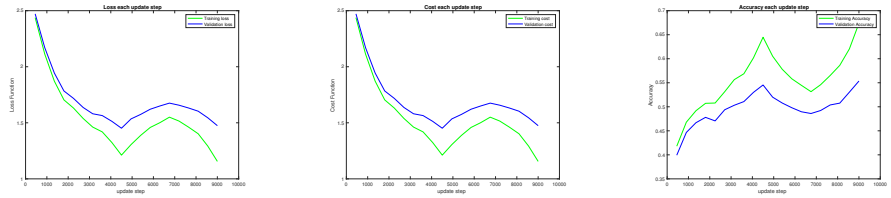
(a) Loss (b) Cost (c) Accuracy

Figure 12: Initialization with var = 0.01 in case without Batch Normalization



(a) Loss (b) Cost (c) Accuracy

Figure 13: He Initialization in case with Batch Normalization



(a) Loss (b) Cost (c) Accuracy

Figure 14: Xavier Initialization in case with Batch Normalization

From the comparison above, Seeing the loss figures with and without batch normalization. I found that with batch normalization, the model is more stable. The loss figures with the model without batch normalization is much more oscillating with the cycle learning rate, unlike the model with batch normalization,

the loss is more stable.

Also, The batch normalization acts as a form of bit regularization, we can see this from the accuracy plots. The difference of accuracy between training data and validation data become very large after 7500 update steps in the models without batch normalization, even larger than the loss before training started.

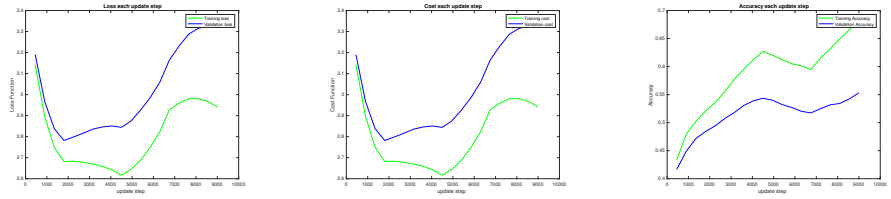
On the other hand, Batch Normalization gives little bit better results, as showed in table 3 and table 4. Also, the models with batch normalization are less sensitive to the method of initialization, the training accuracy and test accuracy showed in table 4 has less difference among different initialization compared to the accuracy in table 3.

To proof that with batch normalization, higher learning rate can be used, I did some experiments and compared the results between models with batch normalization and models without batch normalization under different learning rate. I fixed the hyper-parameters excepts different learning rate and used He Initialization . Table 5 shows the comparison :

eta	With BN	W/o BN
1e-5 to 1e-1	54.43%	53.01%
1e-4 to 1e-1	54.40%	53.49%
1e-3 to 1e-1	53.89%	54.05%
1e-2 to 1e-1	53.27%	53.61%

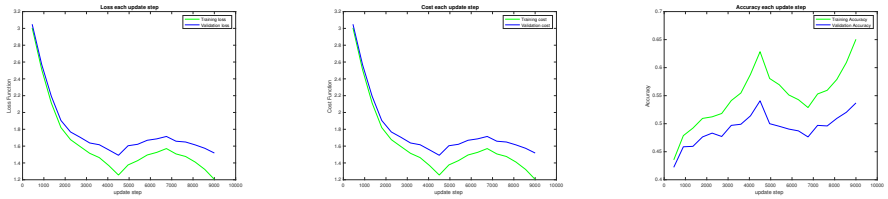
Table 5: comparison between models with batch normalization and models without batch normalization under different learning rate.

From table 5 we can hardly see the different between results with and without batch normalization because the accuracy on test data doesn't drop dramatically in models without batch normalization when the learning rate is high. Thus, we have to look into the cost during training time. The following figures show the cost, loss and accuracy during training time when eta_{min} is 0.01 and eta_{max} is 0.1 in different cases:



(a) Loss (b) Cost (c) Accuracy

Figure 15: High learning rate in case without Batch Normalization



(a) Loss (b) Cost (c) Accuracy

Figure 16: High learning rate in case with Batch Normalization

when looking through the cost and loss during training time, in the model without batch normalization, cost and loss become comparatively large in the end of the training time, even larger than before starting to train the model. On the other hand, although the model with batch normalization receive little lower accuracy on test data, the cost and loss is acceptable in the end of training time.