DD2424 - Lab 3

Magnus Pierrau

June 2020

Introduction

In this lab we expand our previous code for neural networks in order to create a k-layer neural network with multiple outputs and train it using mini-batch gradient descent, a cross-entropy score function with an L_2 regularization term and a cyclical learning rate, to classify images from the CIFAR-10 dataset. Additionally we investigate how Batch Normalization impacts the performance and stability of the network. We also perform a parameter search for the optimal regularization parameter, λ.

Checking analytical gradient

In this lab we introduce two new parameters, γ and β, for scaling and shifting the score as we normalize each batch.

In order to make sure that our numerical gradients are correctly implemented we compare them to numerically approximated gradients, computed by the central difference formula using $h = 1e - 4$.

In order to avoid edge-cases and gradients differing because of random initializations, we let the network burn in for 10 steps before comparing the gradients.

In order to avoid issues with large dimensionality, and computational cost we consider a data dimensionality of $d = 20$ and $N = 100$ samples.

The gradients were checked for a 2-, 3- and 9-layer network with and without batch normalization. I will not account for all 36 parameters of the 9-layer network here, but refrain to presenting the relative errors in the first and last layers.

Exact relative errors are presented in Table 1.

The results in 1 indicate that the analytical gradients are correctly computed up to a relative error of magnitude 10^{-9}-10^{-8}, which is sufficient. We also find that the gradient is correctly computed at both the beginning and end when fitting the network to data (after 400 steps).

One issue is the b parameter, which is estimated to 0 as we introduce batch normalization. For this parameter we cannot divide by the sum of the numerical and analytical gradient, and instead divide by a small epsilon, which we choose to be $\varepsilon = 1e - 7$. Either way, printing the parameters shows that they are

	Relative error	
Par.	Init.	Trained
W_1	4.2e-09	1.9e-07
W_9	3.8e-09	1.4e-08
b_1	8.9e-10	3.6e-04
b_9	9e-07*	8.5e-09
γ_1	1.3e-08	1.7e-08
γ_9	2.8e-09	2.2e-09
β_1	2.2e-09	1.2e-08
β_9	9.0e-09	7.7e-09

Table 1: Relative errors between numerical (finite difference method) and analytical gradients. Tested on reduced training data ($N = 5$) with initial parameters (Init.) and after training for 2 cycles with $n_s = 100$, nBatch=5, and the rest parameters according to table 2 (Trained).

estimated to a magnitude 10^{-20}, and thus effectively zero, which is what we would expect to find.

Another eye catching figure is that of 3.6e-04 for β_1 after training. When inspecting the errors for each dimension we find that the relative error for the other 9 dimensions are of magnitude 10^{-10} - 10^{-9}, whilst one dimension has relative error of 4.9e-03. This might be explained by the change in parameter for this specific dimension causing the score to pass a kink, causing a larger change in gradient than for the analytical solution (as explained in the additional reading material from Stanford).

In large we conclude that our gradient computations are correct.

3-layer Network with/without Batch Normalization

In this section we use data batches 1-4 and 5000 additional samples from data batch 5 for training, for a total of 45000 samples. The final 5000 samples from data batch 5 are used for validation. We test on the separate test data batch. The same procedure is used for the 9-layer network.

Throughout the lab, unless specified, we will use the following parameters:

Additionally, the training batches are randomly shuffled after each epoch. We use a random seed in order to get the same initial parameters and batch shuffling between BN and non-BN runs for the same architectures. This should minimize any difference that could arise from varying initializations and shuffling orders.

We now train a network with two hidden layers, with structure as presented in 3.

Figures 1 and 2 show the evolution of the loss for the 3-layered network as

Parameter	Value
`n_cycles`	2
`n_batch`	100
`n_s`	5*N/Batch Size
`eta_max`	1e-1
`eta_min`	1e-5
`lambda`	5e-3
`alpha`	0.9
Initialization	Kaiming He

Table 2: Parameter settings used throughout the assignment, unless otherwise specified.

Index	Layer type	Dims (out , in) / [ActFunc]
1	FC	(50 , 3072)
2	Act	[ReLu]
3	FC	(50 , 50)
4	Act	[ReLu]
5	FC	(10 , 50)
6	SoftMax	-

Table 3: Network structure for 3-layer Network. FC = Fully Connected layer, Act = Activation Layer, ActFunc = Activate Function.

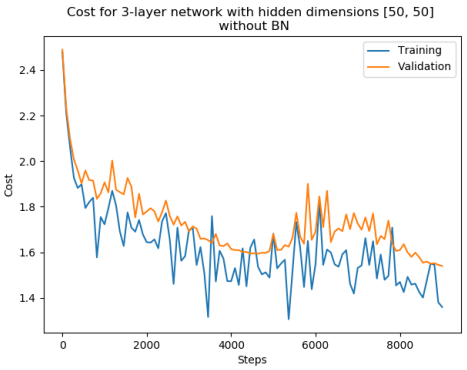

Figure 1: Cost for the 3-layer network before applying Batch Normalization (BN).

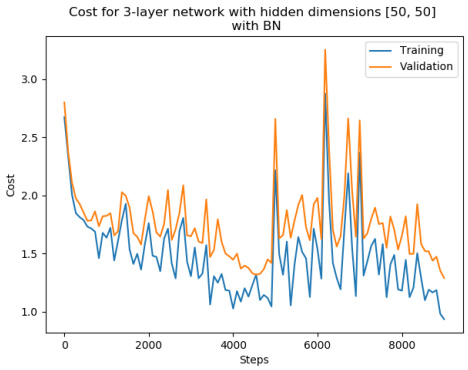

Figure 2: Cost for the 3-layer network after applying Batch Normalization (BN).

BN	Accuracy (Test)	Cost (Validation)
Yes	0.534	1.289
No	0.462	1.306

Table 4: Caption

Index	Layer type	Dims (out , in) / [ActFunc]
1	FC	(50 , 3072)
2	Act	[ReLu]
3	FC	(30 , 50)
4	Act	[ReLu]
5	FC	(20 , 30)
6	Act	[ReLu]
7	FC	(20 , 20)
8	Act	[ReLu]
9	FC	(10 , 20)
10	Act	[ReLu]
11	FC	(10 , 10)
12	Act	[ReLu]
13	FC	(10 , 10)
14	Act	[ReLu]
15	FC	(10 , 10)
16	Act	[ReLu]
17	FC	(10 , 10)
18	SoftMax	-

Table 5: Network structure for 9-layer Network. FC = Fully Connected layer, Act = Activation Layer, ActFunc = Activate Function.

we train it, without (1) and with (2) batch normalization. We note that the performance improves when using BN, with the network with BN achieving a test accuracy of 53.4%, while the one without BN achieves 46.2% on the same set.

9-layer Network with/without Batch Normalization

We now extend our network architecture to have nine hidden layers. This allows the network to learn more complicated patterns, but is more challenging to train. We train the network on the same data and with the same parameter settings as before, specified in table 2.

Figures 3 and 4 show the evolution of the cost over the two cycles for the two network variants. As for the previous task, when applying BN, the loss becomes more spiky, but yields a lower final cost, as seen in 6. In the same table we also

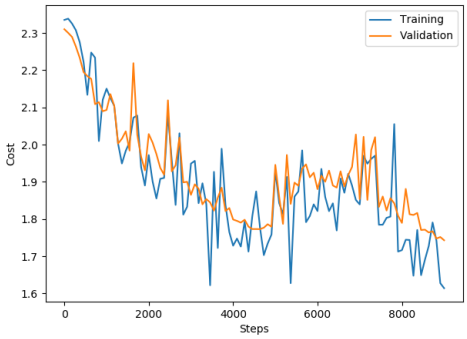

Figure 3: Cost for the 9-layer network before applying Batch Normalization (BN).

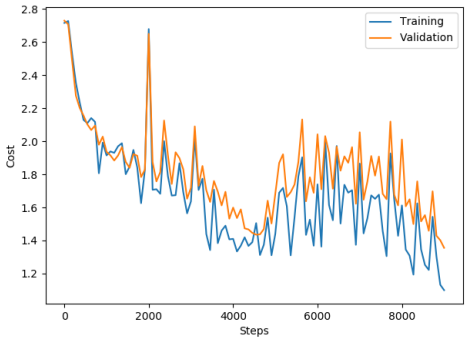

Figure 4: Cost for the 9-layer network after applying Batch Normalization (BN).

BN	Accuracy (Test)	Cost (Validation)
Yes	0.518	1.367
No	0.368	1.739

Table 6: Results for 9-layered network with and without Batch Normalization (BN).

find the accuracies on the test set for both runs, and find that the performance has greatly decreased for the network without BN, while it decreases by a couple of percent for the run with BN.

Lambda search

In order to find an optimal parameter value for the regularization term, λ, we perform a search. We initially perform a coarse search on [1e-04,1e-01], followed by a finer search over [1e-04,2e-02], and finally an even finer search over [4e-3,7e-3]. All searches are done on a uniform interval with 10 steps. The searches were done using standard parameter settings and `n_s = 2*N/nBatch = 9000`, and the same shuffle and initialization seeds for all runs. The initial coarse search was done using only 2 cycles, while the latter finer ones used 3 cycles. We record the result every 50 steps.

The results for all three runs are visualized together in figure 5, and in more detail for the two final runs in 6.

From these experiments we find that the optimal lambda is 6.33e-03, which gave a validation accuracy of 53.98%. When running the 3-layer network for 5 cycles using this lambda and `n_s = 5*N/nBatch = 22500` we achieve a test accuracy of 53.92%.

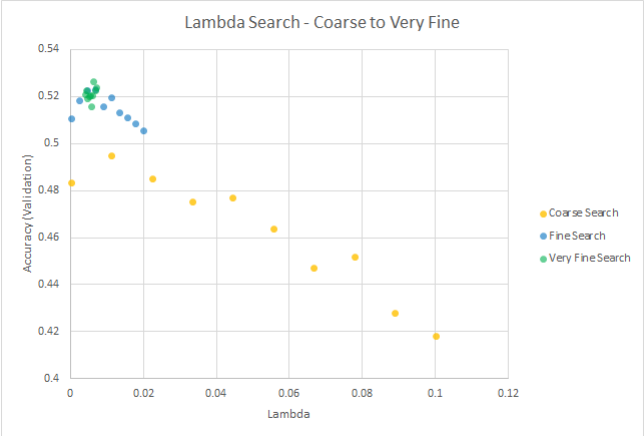

Figure 5: Diagram showing every searched lambda and the corresponding accuracy on the validation set.

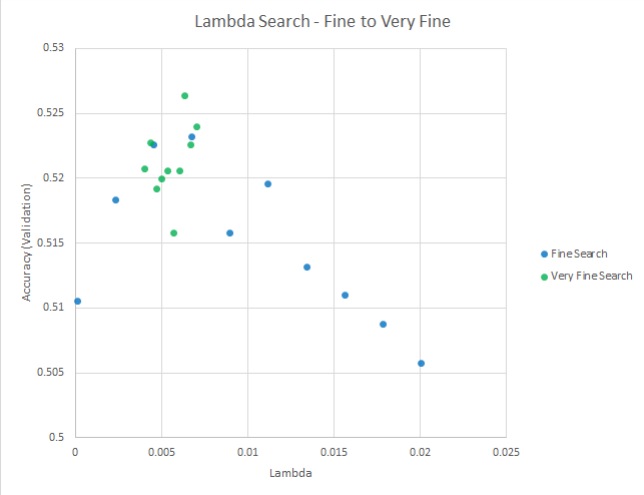

Figure 6: Diagram showing every searched lambda during fine and very fine search, and the corresponding accuracy on the validation set.

Sensitivity to initialization

For this part of the lab we initialize all weight parameters randomly from a normal distribution with expected value 0 and standard deviation σ. We vary the value of σ between [1e-1,1e-3,1e-5] and record the accuracy for the 3-layer network with and without Batch Normalization to see how the robustness of the network changes after applying this new technique.

The results are shown in figures 7 - 12.

We find that for small values of sigma (1e-5 and 1e-3), the cost fluctuates around 2.3 for the two cycles, and does not manage to learn any features, as seen in figures 7 and 9. The network using BN, on the other hand, immediately begins to learn, and the cost decreases within 100 steps (fig 8).

The graphs for 1e-5 and 1e-3 look almost identical for the unnormalized runs, but there are differences at the tenth decimal place. This is due to using the same seed, and the random generator initializing the weights identically, with two order of magnitudes difference (eg. -0.1 and -0.001).

For 1e-01 there is learning for both cases, as seen in figures 11 and 12, with the BN-run outperforms the non-BN one, as expected. The reason why the non-BN network learns when using sigma of size 1e-01, is that this is a sensible initiation, and gives quite similar values as He-initialization would do for a layer with dimensions [50,50,10].

For the normalized case, the graphs are overall similar, but not equal. However, the important take home message is that the BN allows the algorithm to learn, even with unstable initializations, making it more robust.

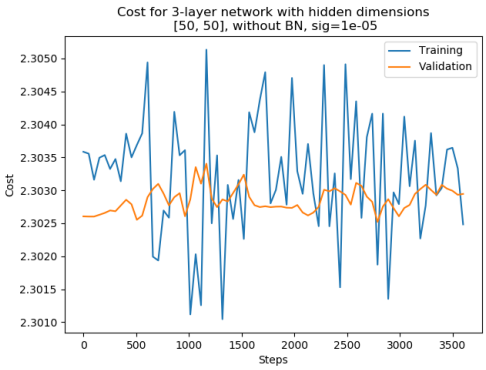

Figure 7: Cost for the 9-layer network before applying Batch Normalization (BN).

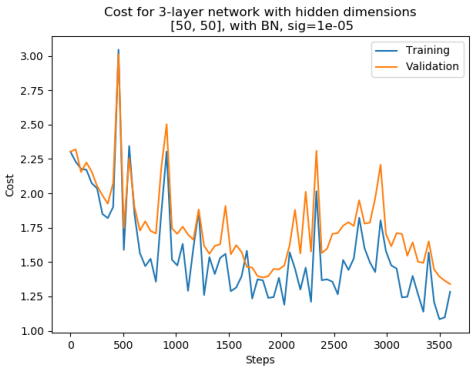

Figure 8: Cost for the 9-layer network after applying Batch Normalization (BN).

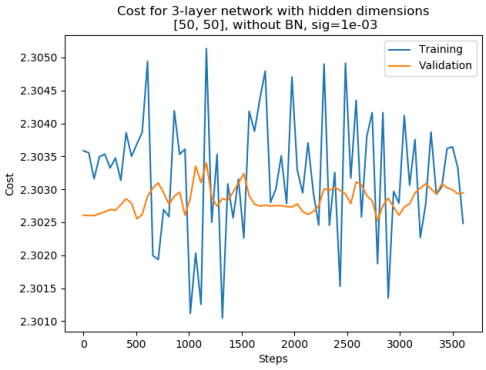

Figure 9: Cost for the 9-layer network before applying Batch Normalization (BN).

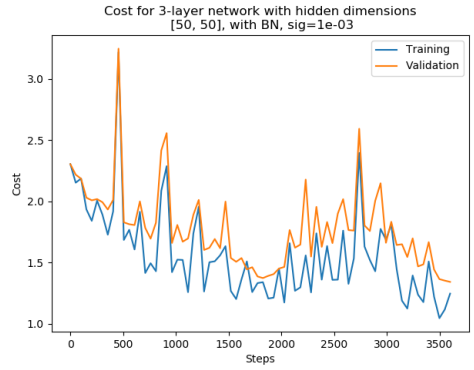

Figure 10: Cost for the 9-layer network after applying Batch Normalization (BN).

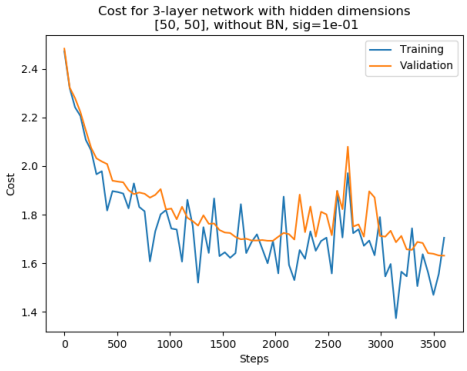

Figure 11: Cost for the 9-layer network before applying Batch Normalization (BN).

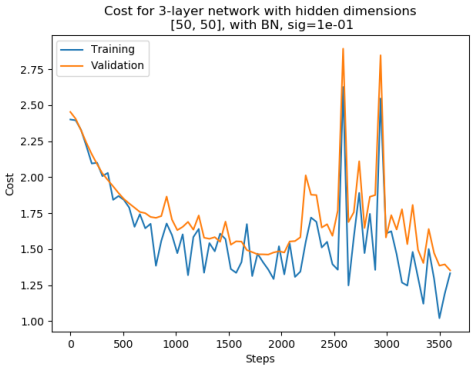

Figure 12: Cost for the 9-layer network after applying Batch Normalization (BN).