Assignment 3 in DD2424
Deep Learning in Data Science

Marie-Ange Stefanos

may 2022

Question i)

State how you checked your analytic gradient computations and whether you think that your gradient computations are bug free for your k-layer network with batch normalization.

Once I have upgraded my code, I checked my the gradient computations by compared them numerically as previously, meaning using the relative error formula given in the Standford course mentioned in first assignment[1], because dividing by a low ϵ as asked in the instructions gives very high errors. That is why I have chosen to work with an expression of the error that does not depend on an external parameter. Moreover, this course indicates how low you should expect your error to be in practice: for 1e-7 and lower errors, you can be happy of your implementation.

Without Batch Norm

As asked in the instructions, I did numerical checks on networks with a small number of nodes in each layer (n = 3) and a much reduced dimensional input data (d = 10) to avoid numerical precision issues. I started with a 2-layer network, then a 3-layer network and then finally a 4-layer network. Obtained errors for each network can be seen on figure 1.

Settings values:

- seed value = 400
- dimension of the hidden layers : [50, 50]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$

```
LAYER 1:
W: 6.97e-08
b: 1.81e-08

LAYER 2:
W: 3.53e-08
b: 3.22e-10
```

(a) k = 2

```
LAYER 1:
W: 2.26e-07
b: 4.03e-08

LAYER 2:
W: 1.66e-07
b: 1.56e-09

LAYER 3:
W: 4.99e-08
b: 1.85e-10
```

(b) k = 3

```
LAYER 1:
W: 1.57e-07
b: 4.96e-09

LAYER 2:
W: 2.41e-06
b: 1.33e-08

LAYER 3:
W: 1.91e-05
b: 2.36e-09

LAYER 4:
W: 3.82e-07
b: 1.57e-10
```

(c) k = 4

Figure 1: Gradients comparisons using relative error for a k-layer network ($n \in \{2, 3, 4\}$).

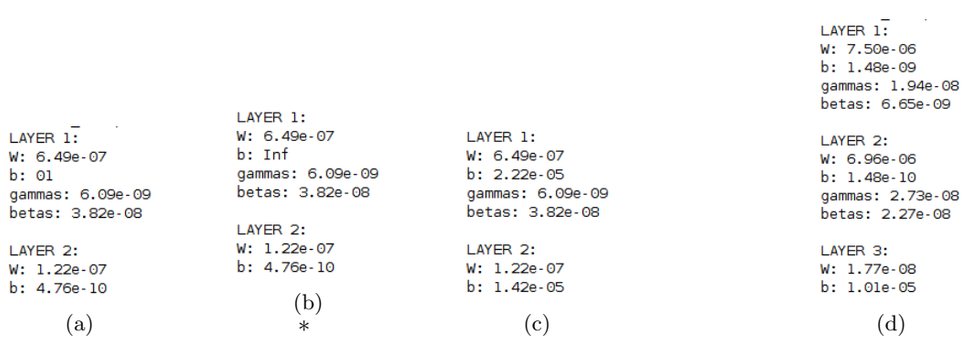
[1] Convolutional Neural Networks for Visual Recognition

As can be seen on figure 1, the errors obtained on W and b are small (1e-7 and below) meaning that I can consider that my gradient computations are correct. However, they are all below this recommended threshold except the error on W in the 4-layer network that are not, but still low (1e-6 or 1-5). This may be the discrepancy between the analytic and the numerical gradients that increases for the earlier layers as the gradient is back-propagated through the network that should be observed here (see fig. (1c)). This effect is not really obvious in my case but this may be due to the choice of this formula for the relative error, that gives great results.

With Batch Norm

When checking my gradients computations with the formula I used in all my labs until now (Stanford course relative error formula), I got a very small error on all gradients except b (W, gammas and beta), meaning below the threshold of 1e-7 and a high error on b gradients obtained at each layer except the last one. This result can be seen on fig. 2a.

As explained in the instructions, the network's bias parameters are superfluous when using batch normalization as I subtract away these biases when I normalize. These bias parameters are estimated as effectively zero vectors when I train. Therefore, the gradients of b has very small values, which explains why the computed error, whose the denominator is $\max(|f_a|, |f_n|)$, gets very high. In order to check that my gradients computations of b are correct, I changed this specific relative error formula. As expected, Wikipedia formula[2] gives an infinite error because of the null denominator (fig. 2b). That's why I used the formula given in the first lab's instructions in order to compute the error on the gradients of b. I got a better error of 1e-5, as can be seen fig. 2c.



(a) (b) * (c) (d)

Figure 2: Gradients comparisons using different relative error formula (Standford course for a), Wikipedia formula for b) and the one given in the lab 1 instructions for c) and d)) for a 2-layer network (a, b, c) and a 3-layer network (d).

I kept this formula to check whether my gradients computations are still correct with a 3-layer network and the errors remain below the threshold of 1e-7 (fig. 2d) so I can consider that my gradients computations are correct.

[2]Wikipedia - Approximation error page

Question ii)

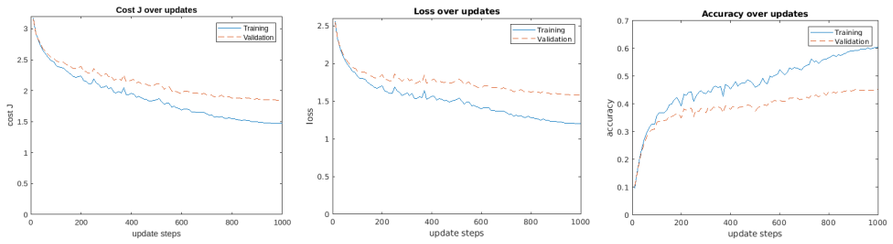
Include graphs of the evolution of the loss function when you train the 3-layer network with and without batch normalization with the given default parameter setting.

Without Batch Norm

2-layer network : Replicate the (default) results I achieved in Assignment 2 using a Xavier initialization

Settings values:

- seed value = 400
- dimension of the hidden layers : [50]
- number of cycle: nb_cycles = 1
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 500
- $\lambda = 0.01$
- update step: t $\in [\![1, 2n\_s]\!] = [\![1, 1000]\!]$



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

Figure 3: Replicate of assignment 2 results with the update code for k-layer networks

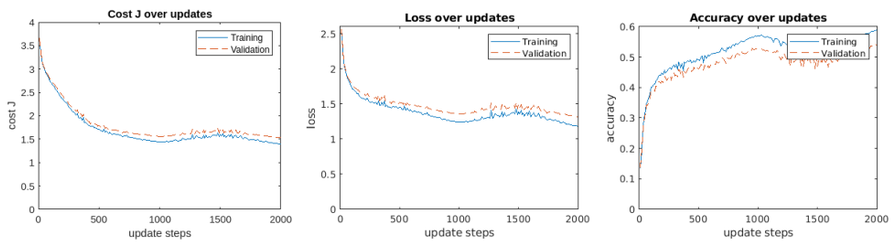
The curves are exactly the same as obtained in last assignment, as well as the accuracy of 45.7%, so I can consider that my implementation of the k-layer network is correct at least for k = 2.

3-layer network with He initialization

In order to increase the performances of network, I decided to implement a He initialization, which is more suitable when using ReLu as an activation function.

Settings values:

- seed value = 400
- dimension of the hidden layers : [50, 50]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

```
fprintf('Test set accuracy: %.1f%%\n', ComputeAccuracy(X_test, y_test, NetParams_star)*100)
```

```
Test set accuracy: 53.0%
```

(d) Test accuracy

Figure 4: Evolution of the loss, the accuracy over updates and performance of the 3-layer network with He initialization without batch normalization

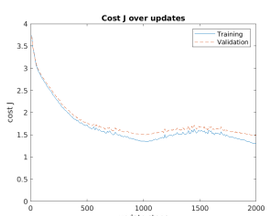
The performances of this 3-layer neural network are good (53% of accuracy on the test set) without any batch normalization. You can note that I obtained a slightly lower performance when using a Xavier initialization (52.5%) so I decided to always use He initialization in the following tasks of this assignments.

With Batch Norm

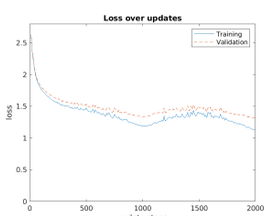
Next step is to compare the performances of the previous 3-layer network with the same settings but using batch normalization. Note that the γ_l and β_l vectors for $l \in [\![1, k-1]\!]$ have been initialized respectively with ones and zeros.

Settings values (exactly the same as in the previous 3-layer network):

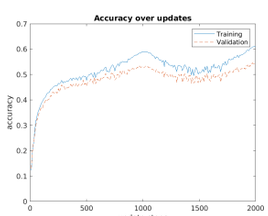
- seed value = 400
- dimension of the hidden layers : [50, 50]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$



(a) Cost through the updates



(b) Loss through the updates



(c) Accuracy through steps

```
fprintf('Test set accuracy: %.1f%%\n', ComputeAccuracy(X_test, y_test, NetParams_star, mu_av, v_av)*100)
```

```
Test set accuracy: 51.9%
```

(d) Test accuracy

Figure 5: Evolution of the cost, the loss, the accuracy over updates and performance of the 3-layer network with He initialization with batch normalization

As shows figure 5 (in particular 5a, 5b and 5c) the cost and the accuracy (train and validation) are very similar to the ones we obtained without batch normalization (respectively 4b, 4b, 4c). Besides, the gap between the blue curves (training curves) and the orange ones (validation curves) is the same as in figure 4. Furthermore, the accuracies obtained on the test set are very close to each other as well. It is even interesting to see that the accuracy on the test set obtained with batch normalization is lower than the one obtained without batch normalization (51.9% vs 53.0%). Thus, batch normalization is not necessary and even slows the running for not too deep neural network (with a small number of layers). Let us see what is happening with a deeper neural network.

Question iii)

Include graphs of the evolution of the loss function when you 9-layer network with and without batch normalization with the given default parameter setting.

This section is about understanding the importance of batch normalization for deep neural network (k-layer network with a high number of layers k) with the case of the following 9-layer network.

Without Batch Norm

This subsection focuses on the 9-layer network trained without batch normalization. The settings used for the training can be found below.

Settings values (same as before except the number of hidden layers and their dimension):

- seed value = 400
- dimension of the hidden layers : [50, 30, 20, 20, 10, 10, 10, 10]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$

(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

```
fprintf('Test set accuracy: %.1f%%\n', ComputeAccuracy(X_test, y_test, NetParams_star)*100)
```

```
Test set accuracy: 49.3%
```

(d) Test accuracy

Figure 6: Evolution of the loss, the accuracy over updates and performance of the 9-layer network with He initialization without batch normalization

The performance of this 9-layer neural network (49.3%, cf fig. 6d) is a little bit below the one obtained with a 3-layer network (51.9% with batch norm and 53% without). As explained in the instructions, it is not surprising since it becomes harder to train with variants of mini-batch gradient descent as a network becomes deeper. That is why using batch normalization can be a solution to overcome this difficulty, as can be seen in next subsection.

With Batch Norm

This subsection focuses on the 9-layer network trained with batch normalization. The settings used for the training can be found below.

Settings values (same as previous subsection):

- seed value = 400
- dimension of the hidden layers : [50, 30, 20, 20, 10, 10, 10, 10]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$

(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

```
fprintf('Test set accuracy: %.1f%%\n', ComputeAccuracy(X_test, y_test, NetParams_star, mu_av, v_av)*100)
```

```
Test set accuracy: 49.7%
```

(d) Test accuracy

Figure 7: Evolution of the loss, the accuracy over updates and performance of the 9-layer network with He initialization and with batch normalization

The different curves look a little bit better (lower cost and loss, higher accuracy) for both training (blue curves) and validation (oranges curves) sets, showing better performance than without batch normalization, which is confirmed by the accuracy obtained in the test set.

The performance of this 9-layer neural network (49.7%, cf fig. 7d) is indeed a little bit higher the one obtained with the same network without batch norm (49.3%). Hence, I obtained an accuracy on the test set closer to the one I had with the 3-layer network ($\approx$ 52-53%) showing that batch normalization increases the performance of deeper networks. However, I guess the performance obtained on the deep network without batch norm could have dropped a little bit more with a different initialization, which would have shown a more significant improvement of the deep network performance with batch norm.

Question iv)

State the range of the values you searched for lambda when you tried to optimize the performance of the 3-layer network trained with batch normalization, and the lambda settings for your best performing 3-layer network. Also state the test accuracy achieved by this network.

For the coarse search of lambda, I first performed a searched on the recommend interval of assignment 2 ($[\![l_{min}, l_{max}]\!] = [\![10^{-5}, 10^{-1}]\!]$) with a uniform grid with eight different values and for 2 cycles of learning rate evolution. The three best results I obtained can be summarized in table 1. The green row matches the values for which the accuracy is the greatest, the yellow ones are for the second and third greatest values of accuracy. The other settings have not changed from question iii), so you have the hyper-parameter settings for the 3 best performing networks I trained.

lambda	**accuracy (%)**
1e-5	51.84
1.429e-2	53.02
2.858e-2	50.92

Table 1: Final accuracy obtained on the validation set for the coarse search of lambdas

Once I got the hyper-parameter of the 3 best performing I trained, I can now perform a fine search around the lambda that matches the highest accuracy and using the second and third values as the limits: $[\![l_{min}, l_{max}]\!] = [\![10^{-5}, 2.858e-2]\!]$. The results are showed in table 2, where you can see each time the accuracy on the validation set (acc_valid and acc_test).

lambda	**acc_valid (%)**	**acc_test (%)**
1e-5	51.84	51.04
4.091e-3	53.34	52.98
8.172e-3	53.1	52.97
1.225e-2	53.04	51.86
1.633e-2	52.64	51.66
2.042e-2	52.18	51.01
2.450e-2	51.7	51.22
2.8581e-2	51.3	50.33

Table 2: Final accuracy obtained on the validation set for the coarse search of lambdas

As shows table 2, the accuracy on the validation set reached 53.34% for lambda = 4.091e-3 (green line). The next best two performances can be seen in yellow cells (53.1% and 53.04% accuracy on the validation set respectively for lambda = 8.172e-3 and 1.225e-2). However, if I needed to find an even better lambda in order to get even higher accuracy I would perform a fine search around the best lambda we got here (4.091e-3), meaning on $[\![l_{min}, l_{max}]\!] = [\![10^{-5}, 8.172e-3]\!]$ (that matches the orange cells on table 2). To conclude, thanks to this fine search, I this network achieved a test accuracy of almost 53% with the best lambda = 4.091e-3, as can be seen on table 2.

Question v)

Include the loss plots for the training with Batch Norm Vs no Batch Norm for the experiment related to Sensitivity to initialization and comment on your experimental findings.

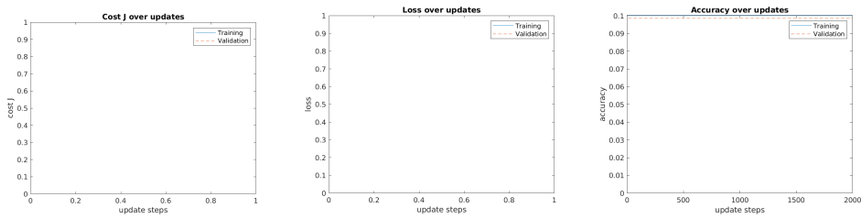
I first did some some tests using a lower stepsize n_s as suggested (2 * 45000 / n_batch = 900 instead of 2250 as follows) but the results were not significantly interesting since it worked only for the highest value of sigma = 0.1. That is why I changed it back to the usual value I used for rest of the assignment. Just in case the issue came from the initialization of the weights, I decided to change the seed value as well (from 400 to 300 for this question).

Settings values (default parameters for 3-layer network):

- seed value = 300
- sigmas = [1e-1, 1e-2, 1e-3]
- dimension of the hidden layers : [50, 50]
- number of cycle: nb_cycles = 2
- eta_min = 1e-5
- eta_max = 1e-1
- batch size: n_batch = 100
- stepsize: n_s = 5 * 45000 / n_batch = 2250
- $\lambda = 0.005$
- update step: t $\in [\![1, 4n\_s]\!] = [\![1, 9000]\!]$

I will focus on the figures I obtained with the highest value of sigma = 0.1 to show the importance of batch normalization but all the obtained figures can be found in appendix A. I will mostly work on the table of the accuracy obtained in each case 3.

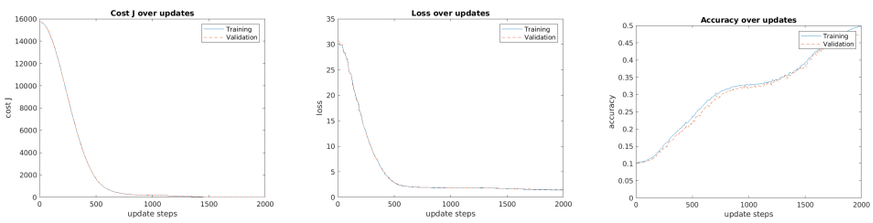
Without Batch Norm



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

Figure 8: Evolution of the loss, the accuracy over updates and performance of the 3-layer network with initialization with a constant sigma = 0.1 and no batch normalization.

With Batch Norm



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

Figure 9: Evolution of the loss, the accuracy over updates and performance of the 3-layer network with initialization with a constant sigma = 0.1 and batch normalization.

Overview on the obtained results and interpretation

sigma	0.1	0.01	0.001
with Batch Norm	47.3	38.8	10
without Batch Norm	10	10	10

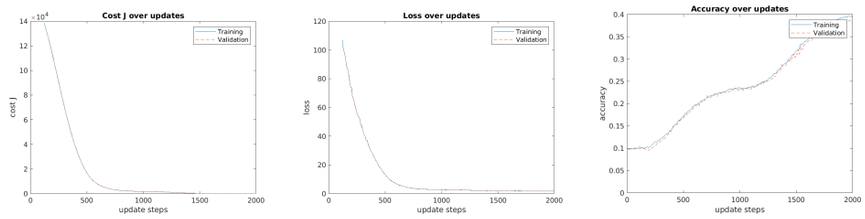
Table 3: Accuracy (a%) obtained on the test set for each value of sigma with and with batch normalization

Figure 8 shows that using a constant value of sigma cannot work even for high values of sigma (0.01 here). The loss function has not the expected aspect of a decreasing function and accuracy remains at 10%, which is equivalent to random attribution of the labels. Thus, the network cannot train with a constant value of sigma (results with a lower value of sigma = 0.01 can be seen in appendix A). However, figure 9 depicts a typical benefit from batch normalization since the exact same network works with the same constant sigma = 0.1 used for initialization. The cost and the loss function are decreasing function and the accuracy increase as expected. The accuracy obtained on the test set is about 47% (as shows table 3), which is little bit below previous results with He initialization but it is closer from it. Therefore, the network can learn properly even using a constant sigma for initiation as long as you use batch normalization and a high sigma (higher than 1e-2 in my case).

Table 3 summarizes the results I obtained when training the same network with different sigma values with and without batch normalization. Note that if it does not work without batch normalization, this process can help but gives better results (38.8% vs 47.3% of accuracy on the test set) for higher sigma values (respectively 0.01 vs 0.01). Moreover, it does not work on sigma value that are too small (0.001 here).

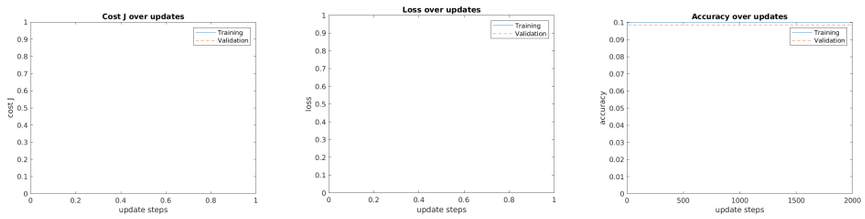
Appendix

A Using a constant sigma = 0.01 and 0.001 for initialization with stepsize n_s = 2250 and batch normalization



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

Figure 10: Evolution of the loss, the accuracy over updates and performance of the 3-layer network with initialization with a constant sigma = 0.01 and batch normalization.



(a) Cost through the updates (b) Loss through the updates (c) Accuracy through steps

Figure 11: Evolution of the loss, the accuracy over updates and performance of the 3-layer network with initialization with a constant sigma = 0.001 and batch normalization.