Deep Learning, Assignment 3

Carlo Saccardi, saccardi@kth.se

May 2022

INTRODUCTION

In this assignment a k-layer neural network was Implemented. The network structure was as follow: one input layer (as many nodes as images pixels), k-1 hidden layer (with ReLu or rectified linear units as activation function), and one output layer (as many nodes as possible image labels and SoftMax as activation function). The loss function used to train the model was cross entropy (classification metric) with L2 regularization. Finally, the predicted class corresponded to the label with the highest predicted probability. For each layer, the weight matrices had the following shapes: as many rows as the output nodes, and as many columns as the input nodes. As for the bias, gammas and betas, the number of rows was equal to the correspondent weight matrix, but with only 1 column.
The dataset used in this assignment was, once again, the CIFAR 10 dataset. Specifically, 45.000 images were used for training, and the remaining 5.000 images for validation. The file 'test batch' was used for testing. The training, validation, and test set were all normalized with the training set's means and standard deviations.

1. State how you checked your analytic gradient computations and whether you think that your gradient computations are bug free.

To compare the gradients obtained by the network in the analytical method (without batch normalization) with the gradients computed with the numerical method (also without batch normalization), I computed the relative error, and checked that these values were small. I did this in three different case scenarios: Firstly, for 2-layer network with 50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. Secondly, for 3-layer network with 50/50 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization. Thirdly, for 4-layer network with 50/50/25 hidden nodes, I study the gradients of the 10 first dimensions of the first 5 images of the training set without regularization.
Once this was done, I studied the same scenarios (except for the 4-layer network) and compared the gradients obtained by the network in the analytical

method and the numerical method, this time with batch normalization.

$$RelativeError = \frac{|g_a - g_n|}{max(eps, |g_a| + |g_n|)}$$

Every time I evaluated this measure, I obtained small values in the order of $e-06, e-07$

2. 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.

In figures 1 and 2 we can find the learning curves for the default parametrization case with and without Batch Normalization respectively: network size 3-layer network with 50 nodes in each hidden state, batch size 100, 2 cycles, ηmin 1e-5, ηmax 1e-1, step size $5 * 45000/100 = 2250$, λ 0.005. Note that it is equivalent to 20 epochs, equal to 2 cycles $\cdot$ 2 $\cdot$ step size 2250 / (45000 images / batch size 100).

Figure 1, learning curves for the default parametrization without Batch Normalization and 3 layers.

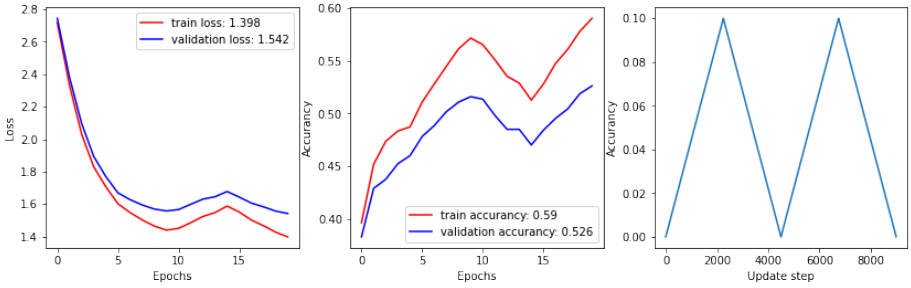
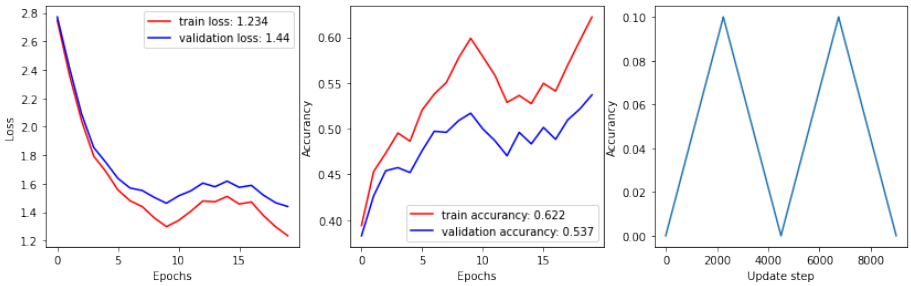


Figure 2, learning curves for the default parametrization with Batch Normalization and 3 layers.



As reported in the plots, the model reaches a validation accuracy of 0.526 when trained without batch normalization, and 0.537 in the second case..

2. 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.

In figures 3 and 4 we can find the learning curves for the default parametrization case with and without Batch Normalization respectively: network size 9-layer network with 50, 30, 20, 20, 10, 10, 10, 10 nodes in each hidden state, batch size 100, 2 cycles, ηmin 1e-5, ηmax 1e-1, step size $5 * 45000/100 = 2250$, λ 0.005. Note that it is equivalent to 20 epochs equal to 2 cycles $\cdot$ 2 $\cdot$ step size 2250 / (45000 images / batch size 100).

Figure 3, learning curves for the default parametrization without Batch Normalization and 9 layers.

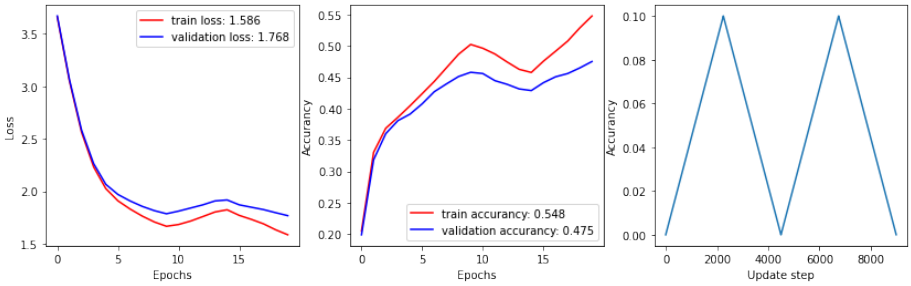
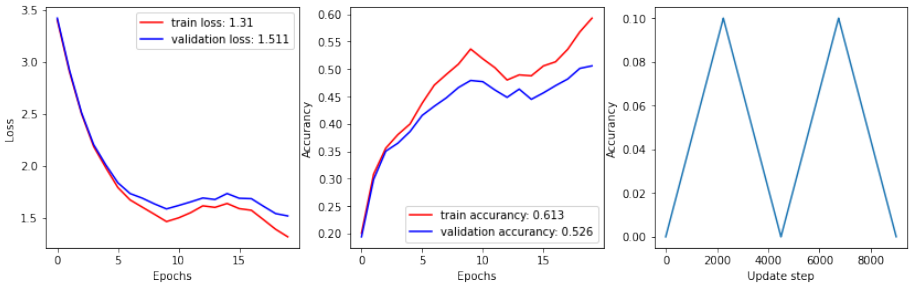


Figure 4, learning curves for the default parametrization with Batch Normalization and 9 layers.



The conclusion after these results is that, at least with the default parametrization, increasing the number of hidden layer to 9 does not achieve better results. Also, the Batch Normalization technique helps the deeper network to improve the results much more compared to the network with only 3 hidden layers.

4. 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.

As a start point, a coarse search over a very broad range of values for lambda was performed. Most of the training data available (45000 images) was used, and the rest for the validation (5000 randomly selected). About the range of *lambda* values, a list of 10 random values was sampled as follow:

$$\lambda_{random} = 10^x, \text{ where}$$
$$X \sim Uniform(-5, -1)$$

In Table 1 the different values of lambda and the correspondent validation accuracy scores for the following parametrization case are reported: batch size 100, 2 cycles, ηmin 1e-5, ηmax 1e-1, step size equal to 5 times number of images divided by batch size (in our case $5 \cdot 45000/100 = 2250$).

Table 1

Out[64]:

	lambda	valid_accurancy
9	0.071557	0.4972
8	0.036906	0.5062
1	0.000495	0.5216
0	0.000342	0.5260
2	0.000563	0.5270
4	0.001568	0.5274
5	0.002577	0.5332
3	0.001512	0.5362
6	0.003833	0.5362
7	0.007257	0.5364

Then, a more restricted search over a defined range of values for lambda was performed. The range of the fine search was based on the results obtained in the previous table, where the best performances were obtained with $\lambda = 0.007257$.. In Tabale 2 the different values of lambda and the correspondent validation accuracy scores for the same parametrization case as the previous search are reported. The best lambda found was $\lambda = 0.004$

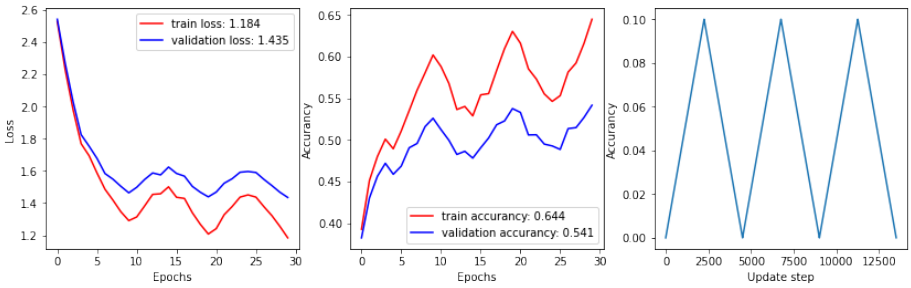
Table 2

Out[65]:

	lambda	valid_accurancy
0	0.0015	0.5336
2	0.0050	0.5336
4	0.0070	0.5336
5	0.0075	0.5354
3	0.0060	0.5392
1	0.0040	0.5414

Figure 5 reports the learning curve for the 3-layer network trained with the optimal lambda, for three cycles. the test accuracy achieved by this network was equal to: 0.53

Figure 5



5. 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.

Batch normalization benefits are that training becomes more stable, higher learning rates can be used and it acts as a form of regularization. Instead of using He initialization, this time the weights were initialized from a normal distribution with 0 mean and σ equal to the same value at each layer. For three

runs I set σ as 1e-1, 1e-3 and 1e-4 respectively and train the network with and without Batch Normalization. The following output reports the losses for each experiment. It turns put that Batch Normalization helps a lot to increase the performance of a network, with lower regularization terms.

Table 3

```
with eta = 1e-1 and with batch normalization, the loss is equal to:  1.8552084134923323
with eta = 1e-1 and without batch normalization, the loss is equal to:  1.8876362143131356
with eta = 1e-3 and with batch normalization, the loss is equal to:  1.5432658985326413
with eta = 1e-3 and without batch normalization, the loss is equal to:  2.206405923164953
with eta = 1e-4 and with batch normalization, the loss is equal to:  1.5469789176721254
with eta = 1e-4 and without batch normalization, the loss is equal to:  2.3028124107654597
```