Deep Learning, Assignment 1

Carlo Saccardi, saccardi@kth.se

April 2022

INTRODUCTION

In this assignment, a neural network with one input layer was implemented with as many nodes as image pixels, one hidden layer of 50 hidden nodes the major part of time (with ReLu as activation function), and one output layer with as many nodes as possible image labels (10). The cross entropy loss function and an L2 regularization term on the weight matrices were applied. Finally, the predicted class corresponded to the label with the highest predicted probability.

Gradients comparison

To compare the gradients obtained by the network (analytical method) with the gradients computed with the finite difference method (numerical method), I computed the relative error, and checked that these values were small. I did this in different case scenarios: Firstly, I compared the gradients of the first 30 dimensions of the first image of data-set 'batch1'. Secondly, I compared the gradients of all dimensions of the first 10 images of data-set 'batch1'. Finally, I compared the gradients of all dimensions of the entire data-set 'batch1'

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

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

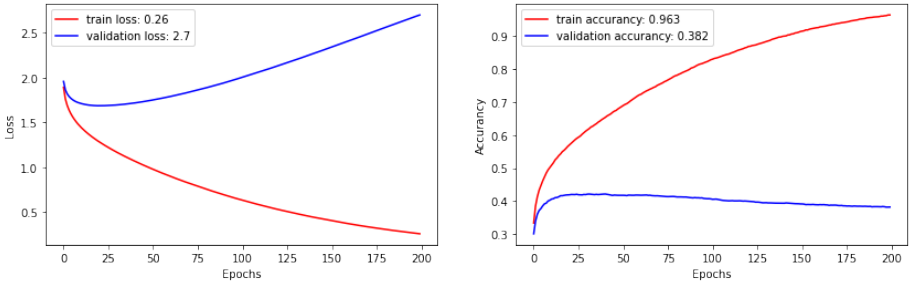
Comparison between gradients in last scenario

```
comparing gW1 and g_W1: the relative error is equal to 1.1893964378689202e-06
comparing gb1 and g_b1: the relative error is equal to 1.0871827791390477e-06
comparing gW2 and g_W2: the relative error is equal to 1.7249730486430448e-07
comparing gb2 and g_b2: the relative error is equal to 5.175830698916054e-06
```

Moreover, as a sanity check, the network was trained on a small amount of the training data (100 examples) with regularization turned off. The hope was to being able to overfit to the training data and get a very low loss on the

training data after training for a sufficient number of epochs ($\tilde{2}00$) and with a reasonable η. Being able to achieve this indicated that the computed gradient and mini-batch gradient descent algorithm were okay.

Sanity check

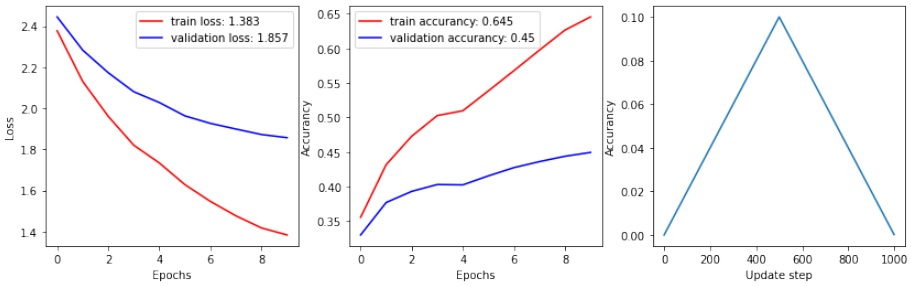


Cyclic learning

There is not really one optimal learning rate when training a neural network with vanilla (fixed η) mini-batch gradient descent. A too small learning rate will take too long to train and too large a learning rate may result in training diverging. For this assignment, the recent idea of exploiting cyclical learning rates was explored, as this approach eliminates much of the trial-and-error associated with finding a good learning rate. The main idea of cyclical learning rates is that during training the learning rate is periodically changed in a systematic fashion from a small value to a large one and then from this large value back to the small value. And this process is then repeated again and again until training is stopped. In this assignment triangular CLR were used and the training was run for a fixed number of complete cycles.

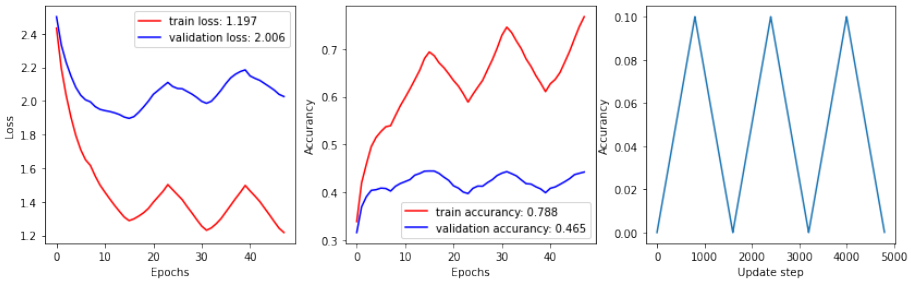
In figure 2 we can find the learning curves for the following parametrization case: batch size 100, 1 cycle, $\eta min = 1e-5$, $\eta max = 1e-1$, step size $= 500$, $\lambda = 0.01$. Note that it is equivalent to 10 epochs (1000 update steps), which is equal to 1 cycle x 2 x step size 500 / (10000 images / batch size 1)

Figure 2



In figure 3 we can find the learning curves for the following parametrization case: batch size = 100, 3 cycle, $\eta min = 1e-5$, $\eta max 1e-1$, step size = 800, $\lambda = 0.01$. Note that it is equivalent to 48 epochs, which is equal to 3 cycle $\cdot$ 2 $\cdot$ step size 800 / (10000 images / batch size 100). In the learning curves, we can clearly see how the loss and accuracy vary as ηt varies at each update step, improving when its values are closer to ηmin (or ending cycles), and worsening when its values are closer ηmax (or in the middle of cycles).

Figure 3



Grid search to find the best lambda

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, $\eta min = 1e-5$, $\eta max 1e-1$, step size = 2 times number of

images divided by batch size = 2·45000/100 = 900. The best lambda found was $\lambda = 0.007257$

Table 1

Out[17]:

	lambda	valid_accurancy
9	0.071557	0.4004
8	0.036906	0.4452
2	0.000563	0.4470
1	0.000495	0.4508
0	0.000342	0.4650
3	0.001512	0.4672
4	0.001568	0.4706
5	0.002577	0.4878
6	0.003833	0.4954
7	0.007257	0.5000

Then, a fine search over a defined range of values for lambda was performed. As for the range of *lambda* values, a list of 6 values between the range [0.006, 0.009] with equal width 0.0005 was used. 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 $\lambda = 0.0065$

Table 2

Out[25]:

	lambda	valid_accurancy
5	0.0085	0.5090
2	0.0070	0.5104
4	0.0080	0.5116
3	0.0075	0.5130
0	0.0060	0.5150
1	0.0065	0.5162

Train on the entire dataset with the best set of parameters

To conclude, for the best lambda setting found, the network was trained on all the training data (all the batch data), except for 1000 examples in a validation set, for $\sin 3$ cycles. The test accuracy obtained was equal to 0.5063.

Figure 4