DD2424 Deep Learning in Data Science - Assignment 2

Marcus Hägglund - mahaggl@kth.se

Introduction

The goal of this assignment is to train and evaluate the performance of a *two layer neural network* in order to classify images from the CIFAR-10 dataset.

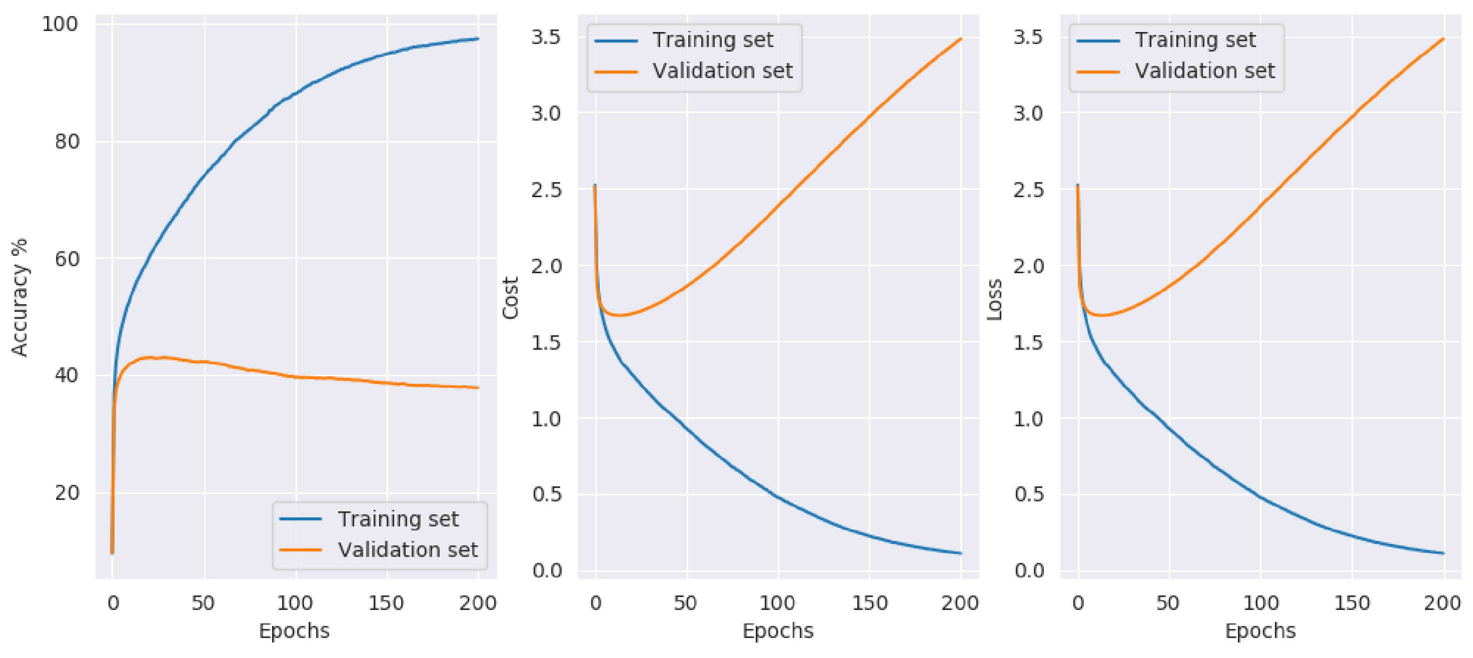
Computing the gradient

Before continuing we first have to verify that the gradients computed are sufficiently accurate. A comparsion is therefore made between the analytically computed gradients and the corresponding gradients computed numerically, for each layer in the network. Because it is computationally expensive to compute the cost for all entries in the weight matrices using the numerical methods we'll reduce the number of images and their dimensionality when computing the gradients for this comparison. The dimensionality of the images are brought down from 3072 to 100. We're also only using 20 samples and setting the tolerance to 1e-5.

The relative error between the analytical gradient and the gradients computed with the *Finite* and *Central*-difference methods are shown in the table below,

Layer #	Relative error Analytical vs Finite Difference	Relative error Analytical vs Centered Difference
1	1.03671e-06	4.3755e-10
2	2.28566e-06	2.35573e-10

A quick sanity check is also done by training the model for 200 epochs with no regularization and a learning rate set to 0.01. From this we expect the model to become very overfitted to the training data and therefore have a very low loss for the training set. The results are shown in the figure below

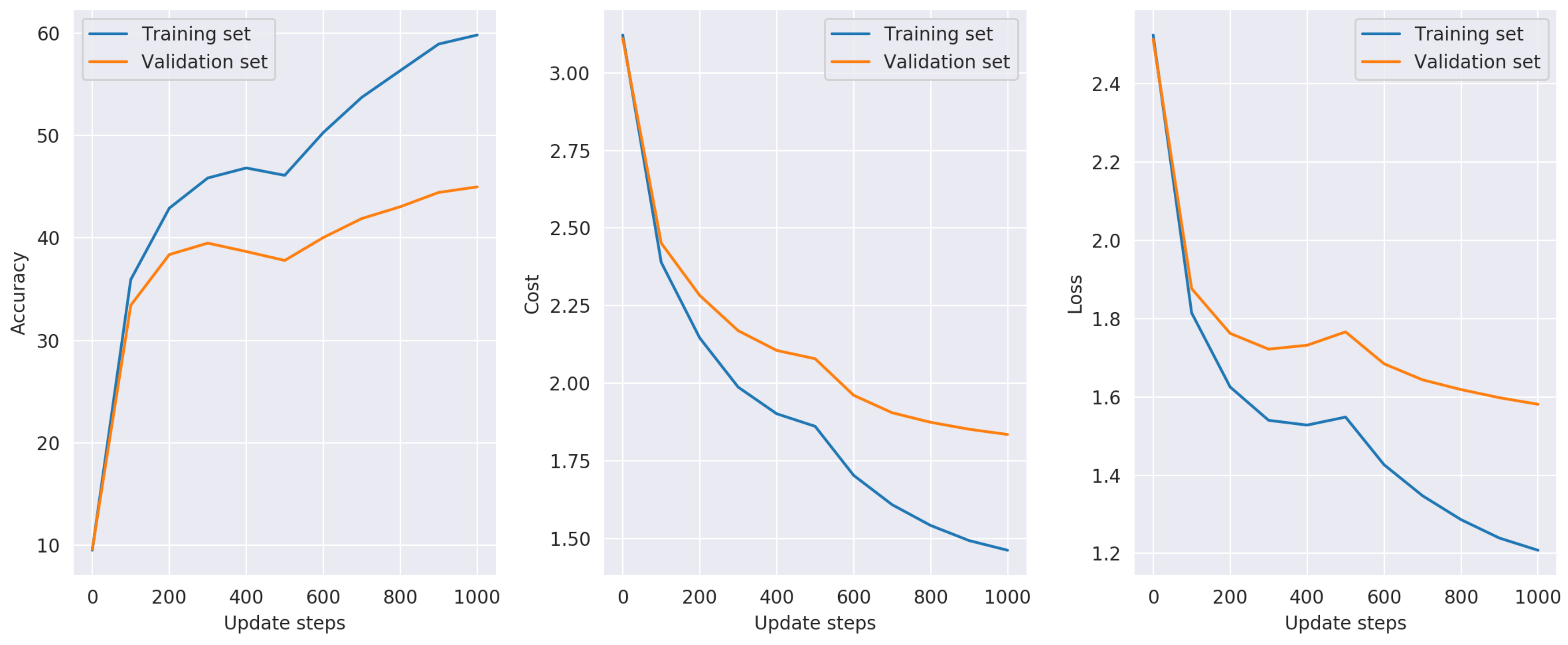


We're now quite confident in that the analytical gradient is sufficiently accurate since we've verified that the differences between the analytical and numerical gradients are small and that we're able to achieve a very overfitted model with a low loss on the training data. As such we'll proceed.

Train the network on a batch of data

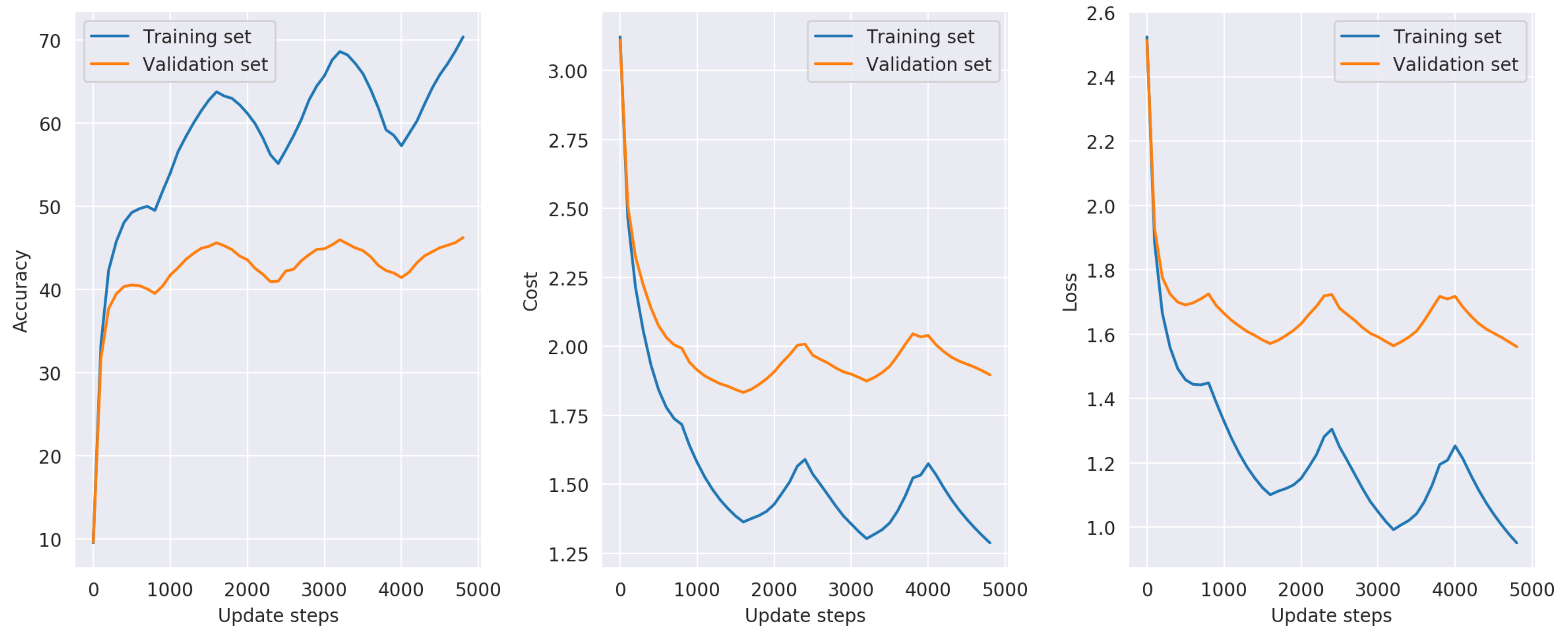
The network was then trained for one cycle with the given parameter values.

```
Model parameters:
   layer size:  50
   lambda:      0.01
   cycles:      1
   n_s:         500
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1

Training data:
   accuracy (untrained):       9.50%
   accuracy (trained):         59.81%
   cost (final):               1.46
Validation data:
   accuracy (untrained):       9.59%
   accuracy (trained):         44.98%
   cost (final):               1.84
Test data:
   accuracy (untrained):       9.82%
   accuracy (trained):         45.61%
   cost (final):               1.81
```



Now we'll train for 3 cycles.

```
Model parameters:
   layer size:  50
   lambda:      0.01
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1

Training data:
   accuracy (untrained):        9.50%
   accuracy (trained):          70.36%
   cost (final):                1.29
Validation data:
   accuracy (untrained):        9.59%
   accuracy (trained):          46.23%
   cost (final):                1.90
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          46.70%
   cost (final):                1.87
```



The cyclical pattern of the learning rate is clearly visible in both the accuracy and loss/cost plots. It seems to work well because by varying the magnitude of the learning rate, we're periodically taking larger "steps" when updating the weights and biases, which may allow us to move away from a local minimum of the cost function. As such, the neural network doesn't run the same risk of getting stuck at a local minima during the gradient descent and should therefore be able to achieve better performance in most cases.

Coarse search for *good* regularization parameter values

We'll now perform a *coarse* search for a good value of the regularization parameter. Samples for regularization parameters are generated with a uniform distribution over set range of possible values. The regularization parameters are generated with the following code

```
lambdas = lmin + (lmax - lmin)*np.random.uniform(size=num_parameters)
lambdas = np.power(10, lambdas)
```

Where the bounds were set as `lmin=-5` and `lmax=-1` and the number of regularization parameters to test were set to `num_parameters=8`.

For each regularization parameter, a network was initialised and trained for two cycles. The performance of each network was then recorded to produce the table below.

lambda	cycles	n_s	n_batches	eta_min	eta_max	accuracy (train)	accuracy (val)	accuracy (test)
0.000211	**2**	**980**	**100**	**1e-05**	**0.1**	**58.53%**	**50.00%**	**51.67%**
0.012474	2	980	100	1e-05	0.1	52.87%	50.20%	51.05%
3.7e-05	2	980	100	1e-05	0.1	58.72%	48.50%	51.20%
0.006893	2	980	100	1e-05	0.1	54.83%	50.60%	51.40%
0.025535	2	980	100	1e-05	0.1	49.16%	48.70%	48.40%
0.089497	2	980	100	1e-05	0.1	39.88%	40.90%	39.87%
7.9e-05	**2**	**980**	**100**	**1e-05**	**0.1**	**58.51%**	**50.30%**	**51.70%**
0.002405	**2**	**980**	**100**	**1e-05**	**0.1**	**57.33%**	**50.70%**	**51.94%**

The three best performing networks are shown in bold.

From this we can then identify the value of lambda that gave the best accuracy on the validation data. Then, we choose a smaller range and perform a finer search to see if we can do better. A decent smaller range seems to be a value between 0.005 and 0.05.

Fine search for *good* values for the regularization parameter

The same prodedure is done as for the coarse search but this time using a smaller range, decided by the results from the coarse search. The bounds were set to `lower_limit=-2` and `upper_limit=-3` and the number of parameters to test was 10. This produced the table below.

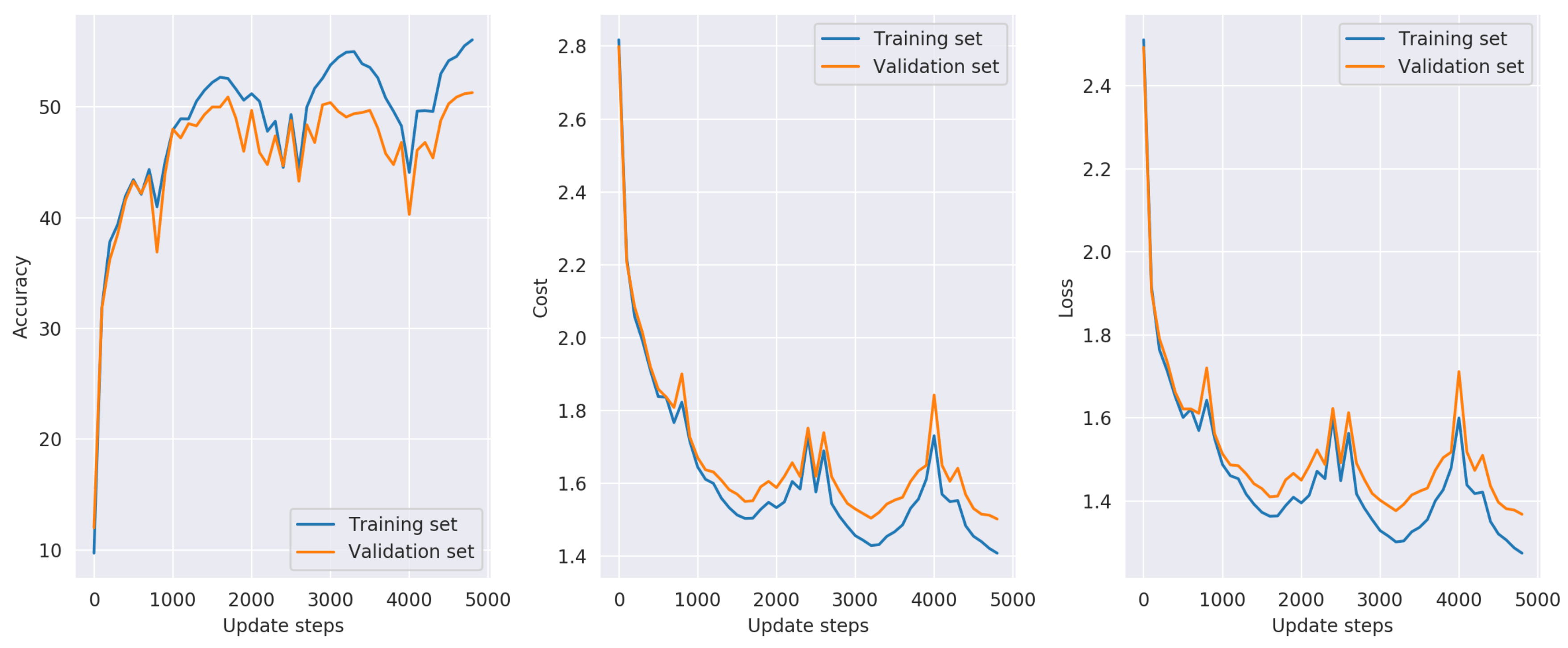
lambda	cycles	n_s	n_batches	eta_min	eta_max	accuracy (train)	accuracy (val)	accuracy (test)
0.002144	**2**	**980**	**100**	**1e-05**	**0.1**	**57.69%**	**50.40%**	**52.02%**
0.005943	2	980	100	1e-05	0.1	55.53%	50.50%	51.30%
0.001383	2	980	100	1e-05	0.1	58.29%	50.80%	51.60%
0.005124	**2**	**980**	**100**	**1e-05**	**0.1**	**56.00%**	**51.60%**	**52.02%**
0.007109	2	980	100	1e-05	0.1	54.99%	51.00%	51.87%
0.009726	2	980	100	1e-05	0.1	53.81%	51.40%	50.94%
0.001674	2	980	100	1e-05	0.1	57.65%	48.80%	51.71%
0.003938	**2**	**980**	**100**	**1e-05**	**0.1**	**56.60%**	**50.70%**	**51.98%**

Once again, the three best performing models are shown in bold.

Train the network on all training data

Now that we've found a good value for the regularization parameter we'll train the network on all the training data (batches 1-5) except for 1000 examples which will be reserved as a validation set. The training is then done for 3 cycles using the best regularization paramater value (0.005124) found during the parameter search.

```
Model parameters:
   layer size:  50
   lambda:      0.005124
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1

Training data:
   accuracy (untrained):        9.71%
   accuracy (trained):          56.07%
   cost (final):                1.41
Validation data:
   accuracy (untrained):        12.00%
   accuracy (trained):          51.30%
   cost (final):                1.50
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          51.40%
   cost (final):                1.52
```


Accuracy
Training set
Validation set
Update steps
Cost
Loss