DD2424 Deep Learning in Data Science - Assignment 3

Marcus Hägglund - mahaggl@kth.se

Introduction

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

Computing the gradient

Once again we have to verify that the gradients computed are sufficiently accurate for each layer. 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 10. We're also only using 20 samples and setting the tolerance to 1e-5.

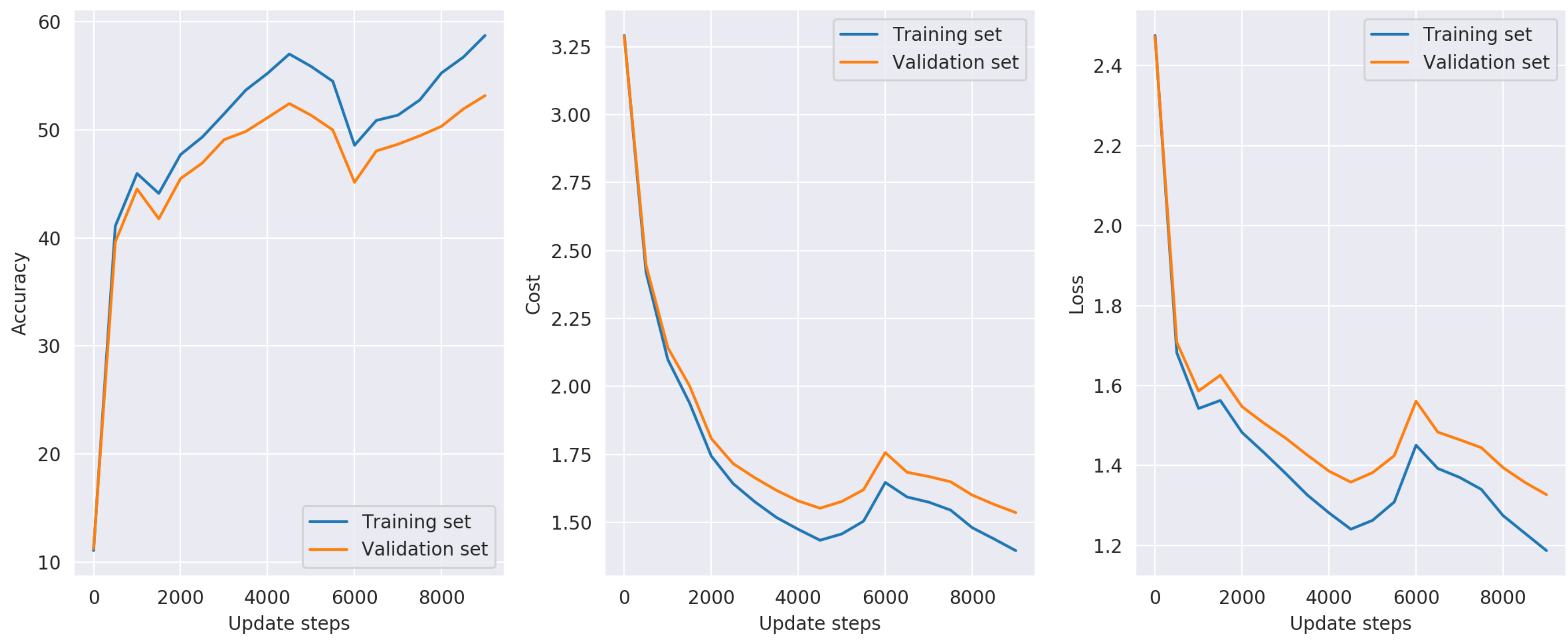
Layer #	Number of nodes	Relative error Weights
1	50	7.89252e-10
2	50	9.23274e-10
3	50	1.09783e-09
4	10	5.70973e-10

We'll also verify that the gradients are correct after implementing the batch normalization.

Train the network

The network is now trained on on all the training data batches (1-5) except for 5000 samples which will be reserved as a validation set. The training is then done for 2 cycles using `n_s = 5 * 45000 / n_batch` and He initialization of the weights.

```
Model parameters:
   hidden layers:       (50, 50)
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he

Training data:
   accuracy (untrained):        11.05%
   accuracy (trained):          58.72%
   cost (final):                1.40
Validation data:
   accuracy (untrained):        11.24%
   accuracy (trained):          53.16%
   cost (final):                1.54
Test data:
   accuracy (untrained):        10.84%
   accuracy (trained):          53.36%
   cost (final):                1.54
```

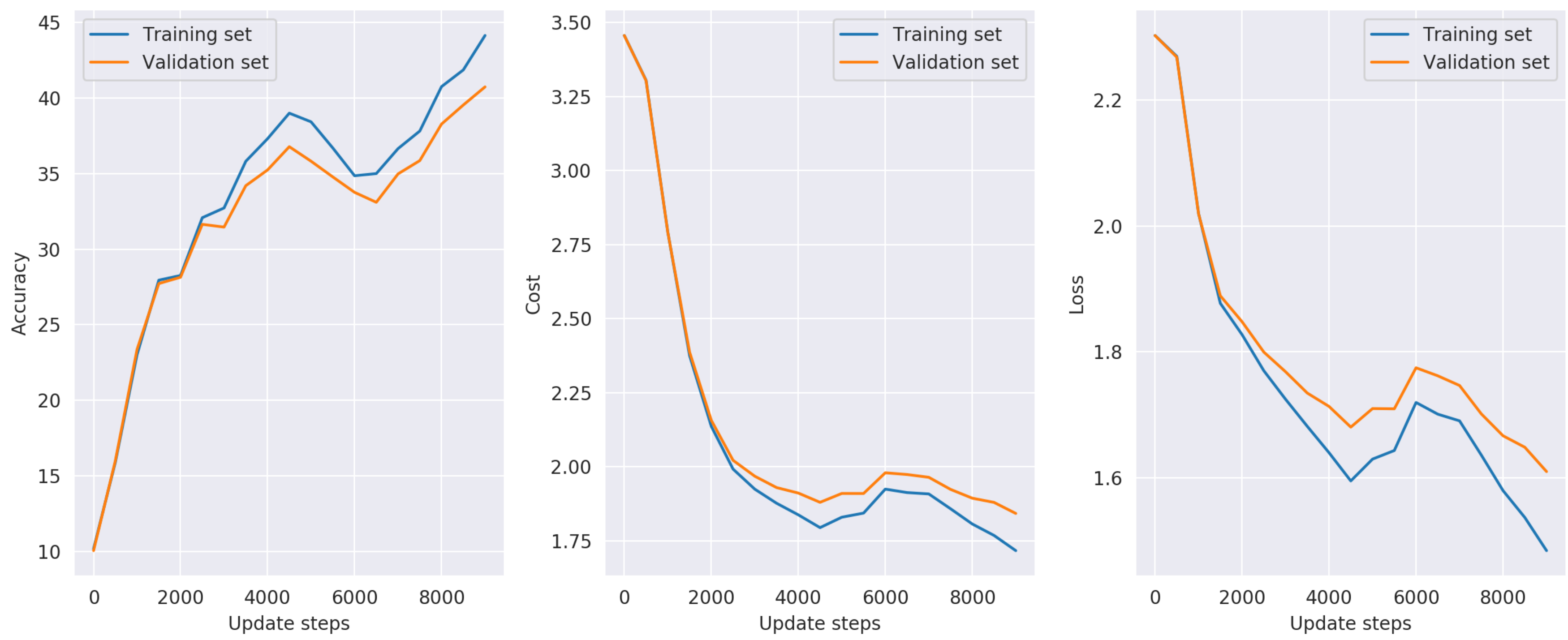


Now, consider instead a 9-layer network with the following number of hidden nodes in each layer `[50, 30, 20, 20, 10, 10, 10, 10]` and see what this does to the networks performance. We'll first verify that the gradients are still accurate for a deeper network.

Layer #	Number of nodes	Relative error Weights
1	50	6.8968e-09
2	30	5.3197e-09
3	20	3.79473e-09
4	20	0.00166769
5	10	2.39833e-09
6	10	1.52859e-09
7	10	2.19497e-09
8	10	2.75328e-09
9	10	3.7868e-09

We're still good and so the network can now be trained with some confidence in the results.

```
Model parameters:
   hidden layers:       (50, 30, 20, 20, 10, 10, 10, 10)
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he

Training data:
   accuracy (untrained):        10.16%
   accuracy (trained):          44.14%
   cost (final):                1.72
Validation data:
   accuracy (untrained):        10.04%
   accuracy (trained):          40.74%
   cost (final):                1.84
Test data:
   accuracy (untrained):        10.59%
   accuracy (trained):          40.37%
   cost (final):                1.85
```



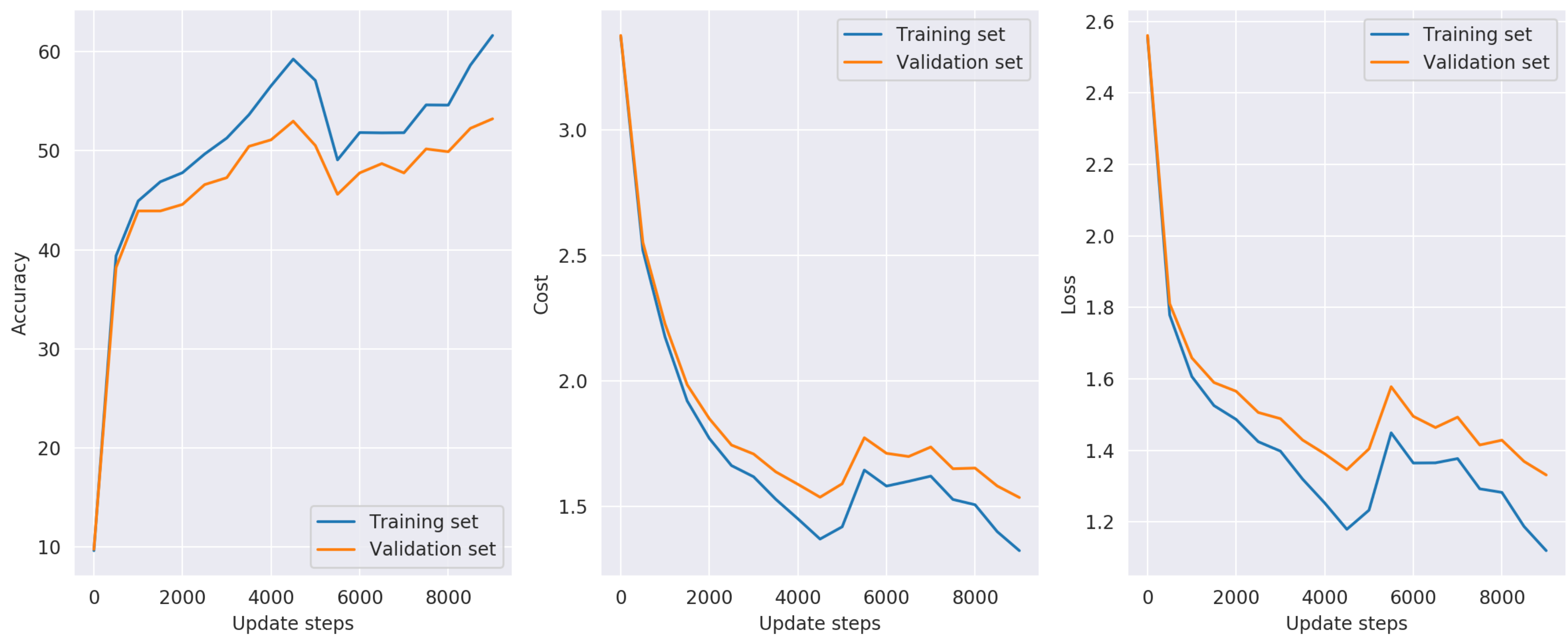
Implement BatchNormalization

After implementing the batch normalization we'll first check the gradients once again to see if they are still accurate.

Layer #	Number of nodes	Relative error Weights	Relative error Gamma
1	50	4.58123e-10	3.28953e-10
2	50	3.38265e-10	3.26867e-10
3	50	3.8832e-10	3.42983e-10
4	10	1.58677e-10	N/A (output layer)

Since they seem to be fine we'll carry on and train the network using batch normalization and shuffling of the training data. All other parameters are kept the same as in the case without batch normalization.

```
Model parameters:
   hidden layers:       (50, 50)
   BN:                  True
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he
   shuffle:             True

Training data:
   accuracy (untrained):        9.64%
   accuracy (trained):          61.63%
   cost (final):                1.32
Validation data:
   accuracy (untrained):        9.78%
   accuracy (trained):          53.22%
   cost (final):                1.53
Test data:
   accuracy (untrained):        9.22%
   accuracy (trained):          53.37%
   cost (final):                1.53
```

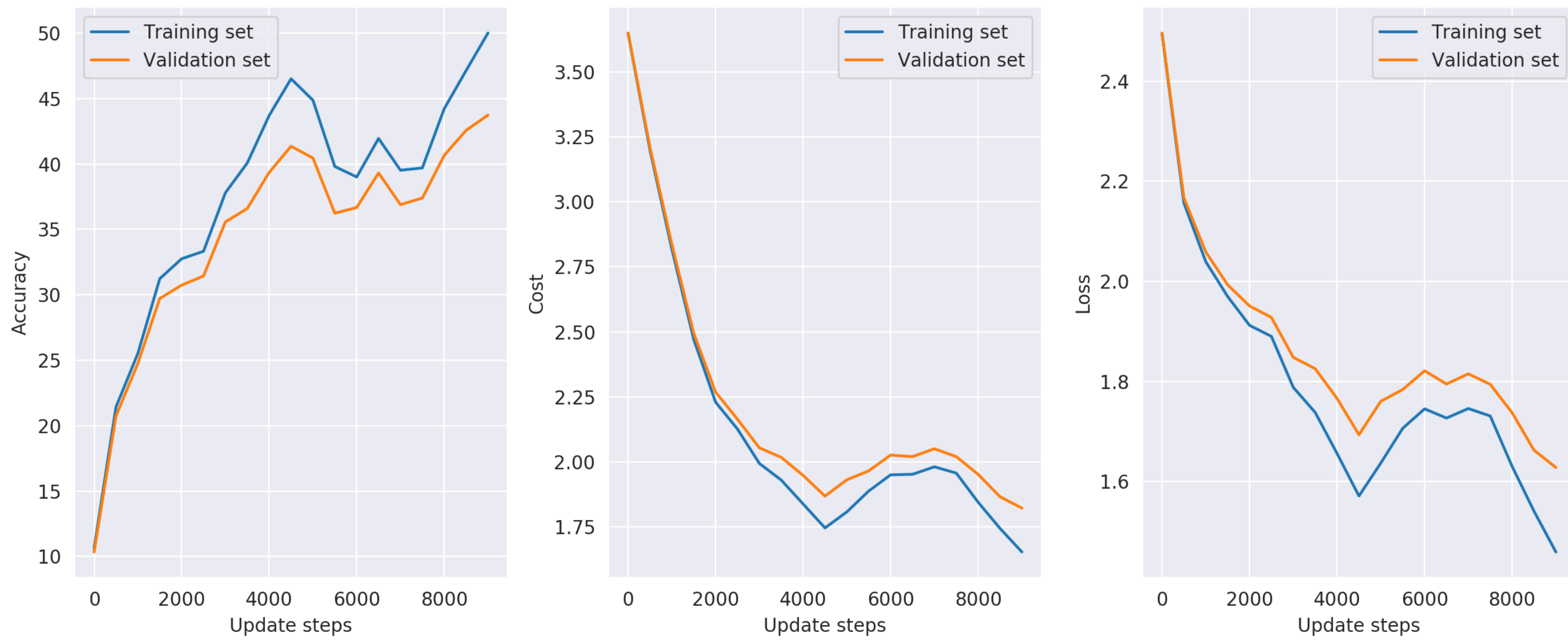


We'll now check the impact of batch normalization on a deeper network, namely the 9-layer one which we ran earlier. First a quick verification that the gradients are still accurate for the deeper network structure,

Layer #	Number of nodes	Relative error Weights	Relative error Gamma
1	50	1.03419e-09	2.02826e-10
2	30	2.57194e-10	2.13918e-10
3	20	2.10523e-10	2.60176e-10
4	20	2.31004e-10	2.40764e-10
5	10	2.09771e-10	2.02915e-10
6	10	2.72691e-10	1.06608e-10
7	10	1.743e-10	1.45683e-10
8	10	2.74625e-10	9.2376e-11
9	10	1.94551e-10	N/A (output layer)

Once again, the gradients computations look good and so we may proceed with training the network.

```
Model parameters:
   hidden layers:       (50, 30, 20, 20, 10, 10, 10, 10)
   BN:                  True
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he
   shuffle:             True

Training data:
   accuracy (untrained):        10.63%
   accuracy (trained):          50.02%
   cost (final):                1.65
Validation data:
   accuracy (untrained):        10.34%
   accuracy (trained):          43.74%
   cost (final):                1.82
Test data:
   accuracy (untrained):        10.53%
   accuracy (trained):          43.76%
   cost (final):                1.84
```



There is clearly a significant performance boost when using batch normalization on networks with a deeper architecture. When training the network using BN, the accuracy of the 9-layer network increased by 3.39 percentage points. The 3-layer network on the other hand had approximately the same accuracy on the test data in both cases.

Parameter search

We'll now perform a *coarse-to-fine* search for a good value of the regularization parameter. The search is done in the same way that it was done in the previous assignment. Each network is trained for 1 cycle with batch normalization for different values of the regularization parameter. All other parameters are set to the default values. An initial coarse search is first done in order to narrow down the region where a suitable value of the regularization parameter may lie. The coarse search is therefore performed over the range `1e-1 to 1e-5`.

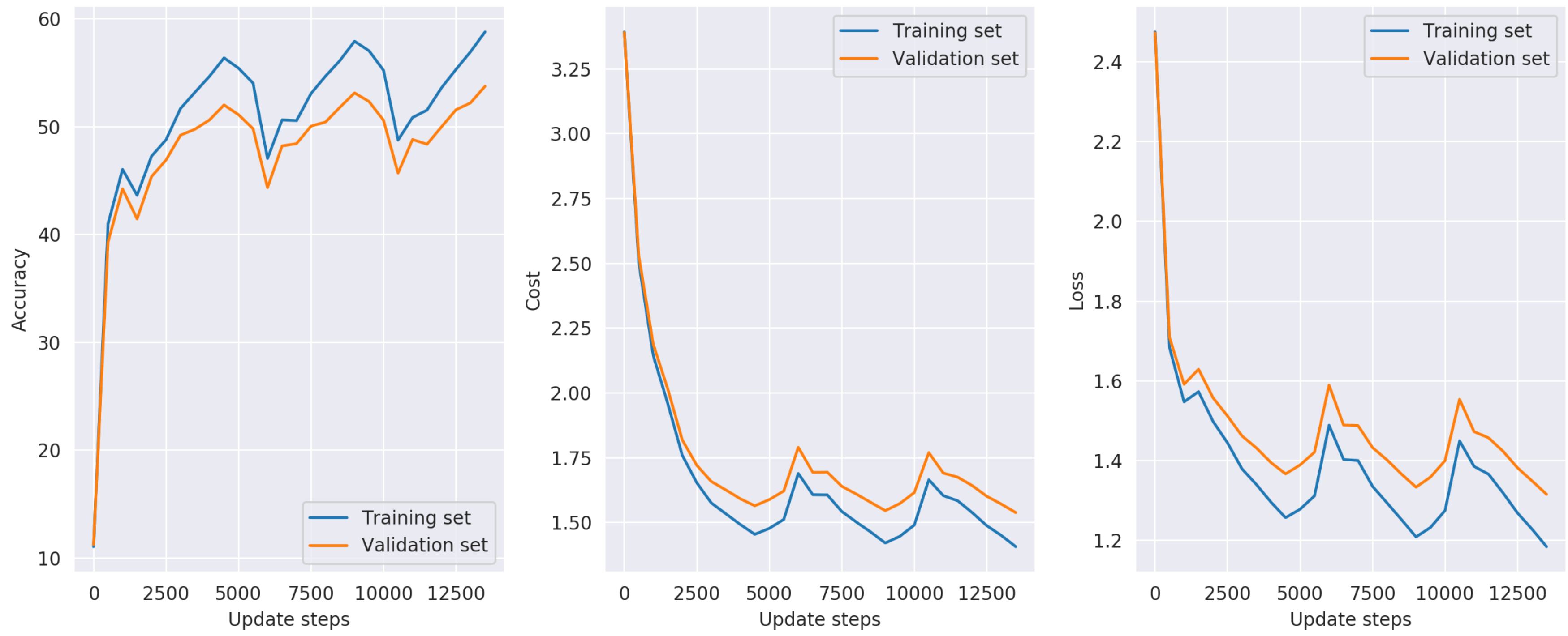
hidden layers	BN	lambda	accuracy (train)	accuracy (val)	accuracy (test)
(50, 50)	True	0.1	51.71%	49.50%	49.65%
(50, 50)	**True**	**0.01**	**58.00%**	**52.44%**	**52.05%**
(50, 50)	True	0.001	58.22%	51.16%	50.87%
(50, 50)	True	0.0001	57.81%	50.96%	50.39%
(50, 50)	True	0.00001	57.61%	50.20%	49.55%

The coarse search indicates that a good value is in the range `1e-2` to `1e-3` and so we'll perform a finer search in this region.

hidden layers	BN	lambda	accuracy (train)	accuracy (val)	accuracy (test)
(50, 50)	True	0.01	58.00%	52.44%	52.05%
(50, 50)	**True**	**0.005623**	**58.73%**	**51.54%**	**52.34%**
(50, 50)	True	0.003162	58.90%	52.50%	51.24%
(50, 50)	True	0.001778	58.86%	51.48%	50.82%
(50, 50)	True	0.001	58.22%	51.16%	50.87%

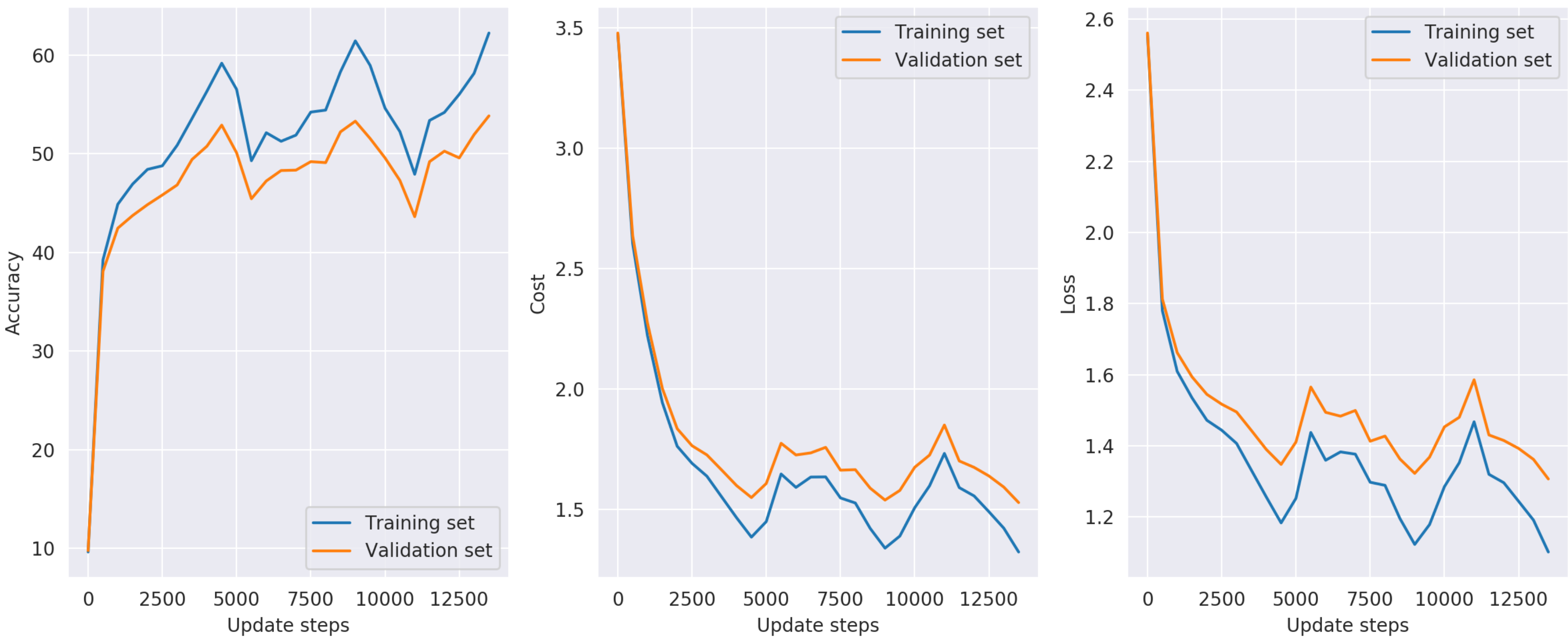
The best value of the regularization parameter found is `0.005623`. A network is then trained for 3 cycles using the value found which gave the following result,

```
Model parameters:
   hidden layers:        (50, 50)
   BN:                   False
   lambda:               0.005623
   cycles:               3
   n_s:                  2250
   n_batches:            100
   eta_min:              1e-05
   eta_max:              0.1
   initialization:       he

Training data:
   accuracy (untrained):        11.05%
   accuracy (trained):          58.80%
   cost (final):                1.41
Validation data:
   accuracy (untrained):        11.24%
   accuracy (trained):          53.76%
   cost (final):                1.54
Test data:
   accuracy (untrained):        10.84%
   accuracy (trained):          53.58%
   cost (final):                1.55
```



Another network is then trained using batch normalization and shuffling of the training data,

```
Model parameters:
   hidden layers:        (50, 50)
   BN:                   True
   lambda:               0.005623
   cycles:               3
   n_s:                  2250
   n_batches:            100
   eta_min:              1e-05
   eta_max:              0.1
   initialization:       he
   shuffle:              True

Training data:
   accuracy (untrained):        9.64%
   accuracy (trained):          62.24%
   cost (final):                1.32
Validation data:
   accuracy (untrained):        9.78%
   accuracy (trained):          53.84%
   cost (final):                1.53
Test data:
   accuracy (untrained):        9.22%
   accuracy (trained):          52.87%
   cost (final):                1.55
```



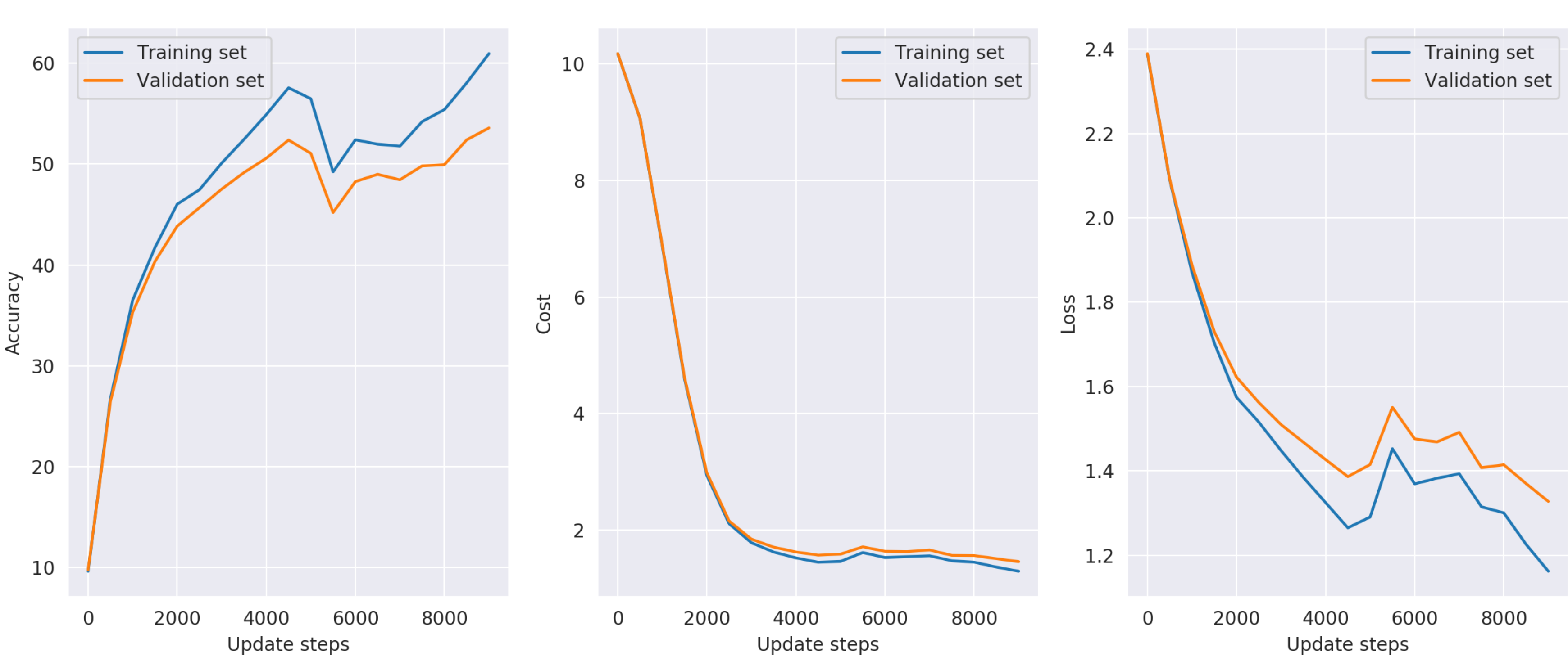
Sensitivity to initialization

We'll now investigate the networks sensitivity to the weight initialization with and without using batch normalization. To do this we'll instead initiate the weights of each layer using a normal distribution with mean zero and standard deviation sigma where we'll try a couple of values on sigma, specifically `sigma=1e-1` , `sigma=1e-3` and `sigma=1e-4` .

Starting out with `sigma=1e-1` for the networks with and without BN.

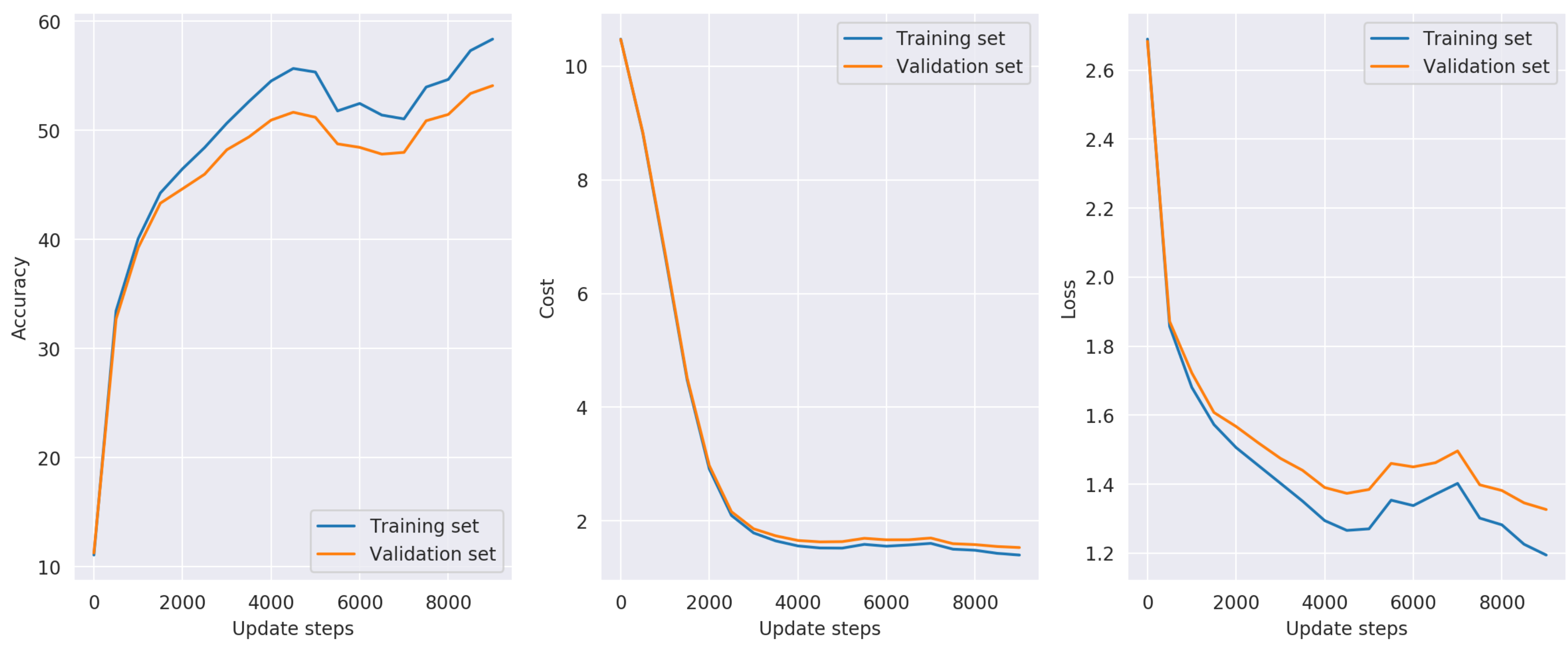
With BN

```
Model parameters:
   hidden layers:          (50, 50)
   BN:                     True
   lambda:                 0.005
   cycles:                 2
   n_s:                    2250
   n_batches:              100
   eta_min:                1e-05
   eta_max:                0.1
   initialization:         0.1
   shuffle:                True

Training data:
   accuracy (untrained):          9.64%
   accuracy (trained):            60.95%
   cost (final):                  1.30
Validation data:
   accuracy (untrained):          9.78%
   accuracy (trained):            53.58%
   cost (final):                  1.46
Test data:
   accuracy (untrained):          9.22%
   accuracy (trained):            52.03%
   cost (final):                  1.48
```



Without BN

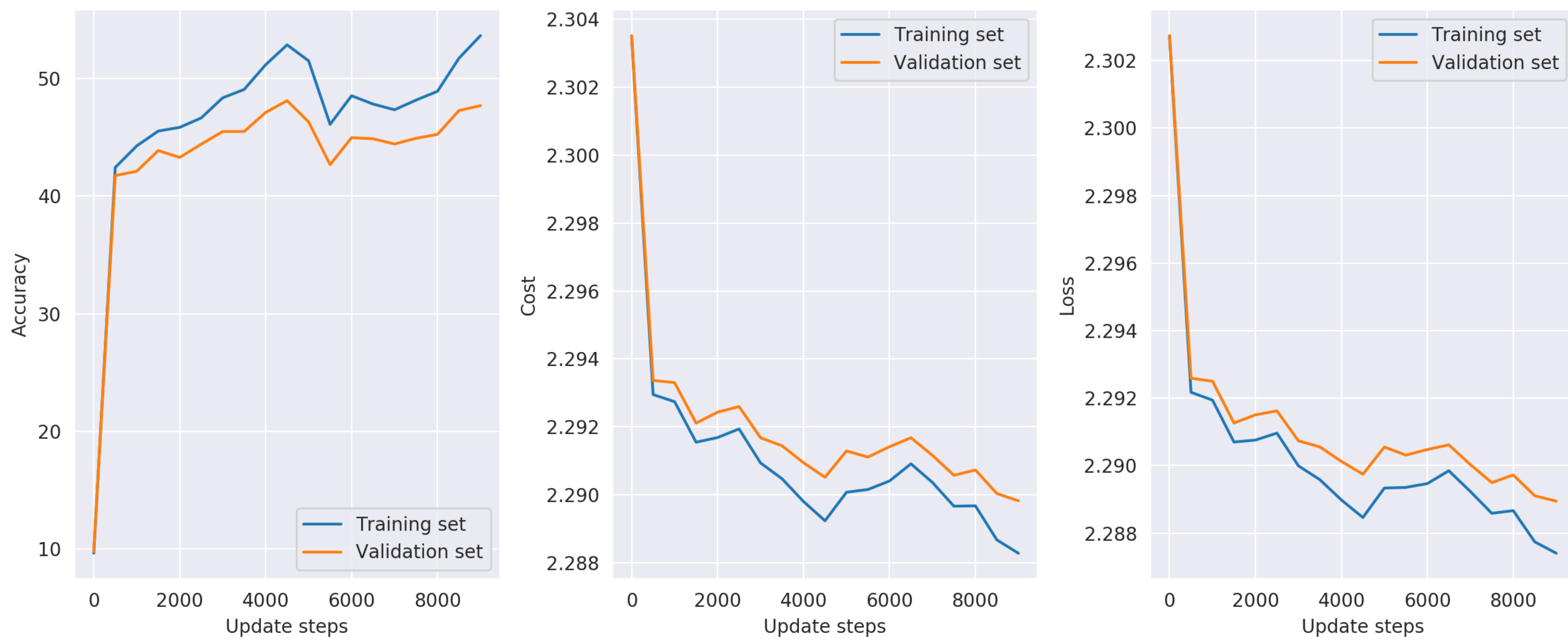
```
Model parameters:
   hidden layers:       (50, 50)
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      0.1
   shuffle:             True

Training data:
   accuracy (untrained):        11.05%
   accuracy (trained):          58.37%
   cost (final):                1.40
Validation data:
   accuracy (untrained):        11.24%
   accuracy (trained):          54.10%
   cost (final):                1.53
Test data:
   accuracy (untrained):        10.84%
   accuracy (trained):          53.03%
   cost (final):                1.54
```



Continuing with `sigma=1e-3`

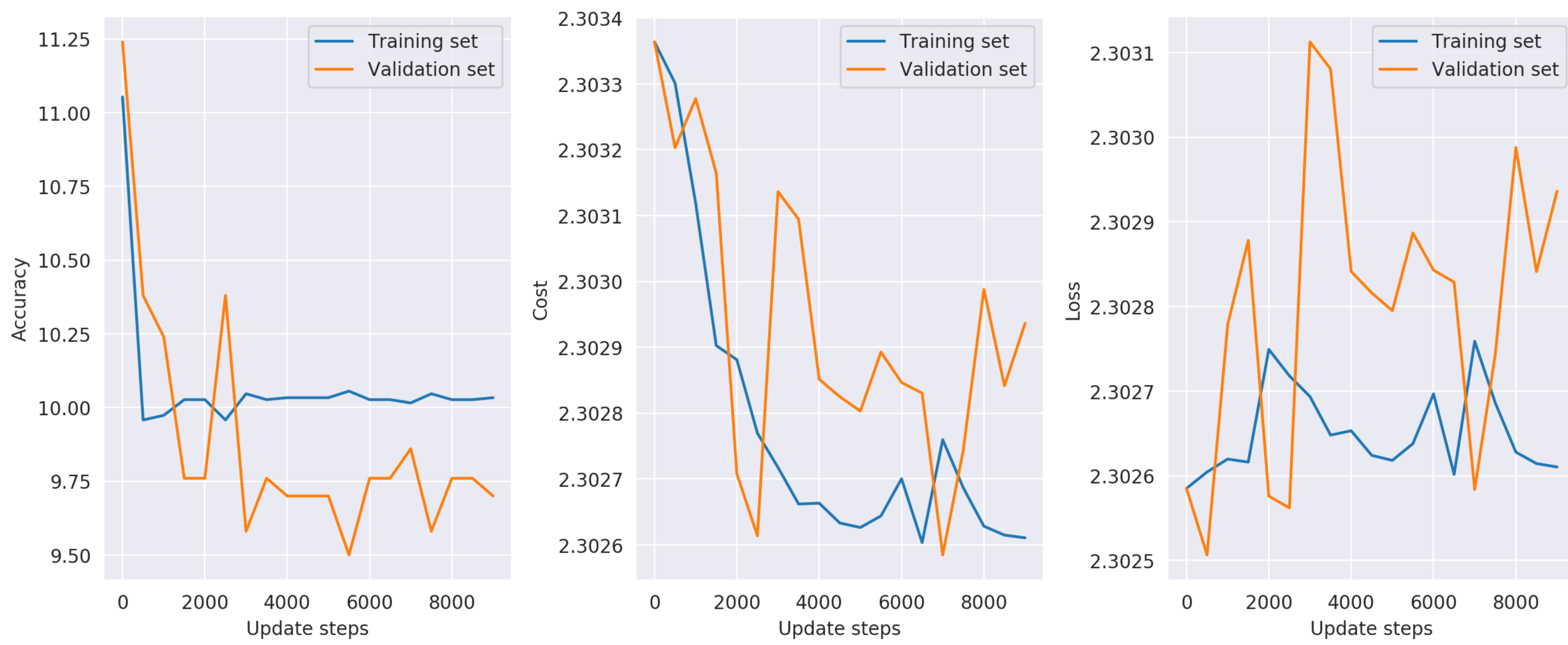
With BN

```
Model parameters:
   hidden layers:       (50, 50)
   BN:                  True
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      0.001
   shuffle:             True

Training data:
   accuracy (untrained):        9.64%
   accuracy (trained):          53.63%
   cost (final):                2.29
Validation data:
   accuracy (untrained):        9.78%
   accuracy (trained):          47.68%
   cost (final):                2.29
Test data:
   accuracy (untrained):        9.22%
   accuracy (trained):          48.05%
   cost (final):                2.29
```



Without BN

```
Model parameters:
   hidden layers:       (50, 50)
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      0.001
   shuffle:             True

Training data:
   accuracy (untrained):        11.05%
   accuracy (trained):          10.03%
   cost (final):                2.30
Validation data:
   accuracy (untrained):        11.24%
   accuracy (trained):          9.70%
   cost (final):                2.30
Test data:
   accuracy (untrained):        10.84%
   accuracy (trained):          10.00%
   cost (final):                2.30
```



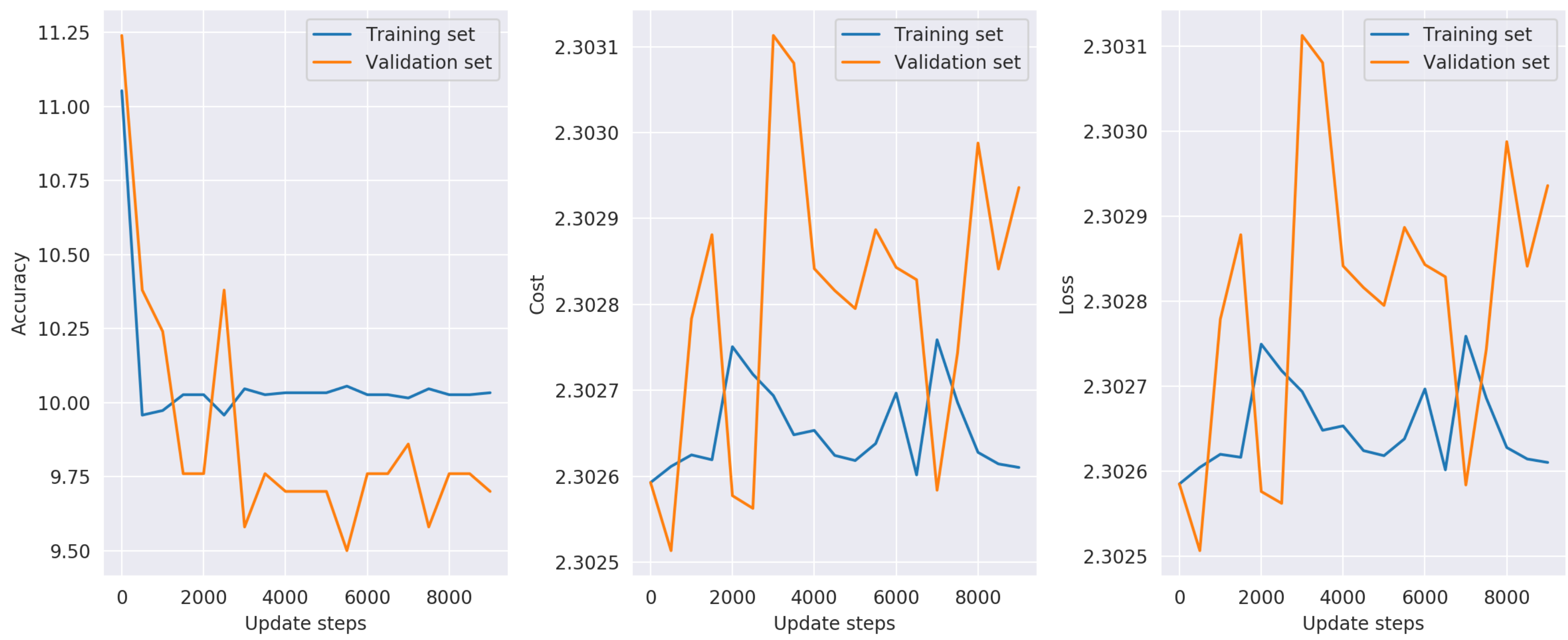
And finally setting `sigma=1e-4`

With BN

```
Model parameters:
   hidden layers:        (50, 50)
   BN:                   True
   lambda:               0.005
   cycles:               2
   n_s:                  2250
   n_batches:            100
   eta_min:              1e-05
   eta_max:              0.1
   initialization:       0.0001
   shuffle:              True

Training data:
   accuracy (untrained):         9.64%
   accuracy (trained):           53.54%
   cost (final):                 2.30
Validation data:
   accuracy (untrained):         9.78%
   accuracy (trained):           48.26%
   cost (final):                 2.30
Test data:
   accuracy (untrained):         9.22%
   accuracy (trained):           47.97%
   cost (final):                 2.30
```

Without BN

```
Model parameters:
   hidden layers:        (50, 50)
   BN:                   False
   lambda:               0.005
   cycles:               2
   n_s:                  2250
   n_batches:            100
   eta_min:              1e-05
   eta_max:              0.1
   initialization:       0.0001
   shuffle:              True

Training data:
   accuracy (untrained):         11.05%
   accuracy (trained):           10.03%
   cost (final):                 2.30
Validation data:
   accuracy (untrained):         11.24%
   accuracy (trained):           9.70%
   cost (final):                 2.30
Test data:
   accuracy (untrained):         10.84%
   accuracy (trained):           10.00%
   cost (final):                 2.30
```



The results are summarized in the table below,

hidden layers	BN	initialization	accuracy (train)	accuracy (val)	accuracy (test)
(50, 50)	True	0.1	57.70%	52.32%	51.99%
(50, 50)	False	0.1	53.19%	50.72%	50.64%
(50, 50)	True	0.001	52.16%	48.00%	48.13%
(50, 50)	**False**	0.001	10.06%	9.50%	**10.00%**
(50, 50)	True	0.0001	52.00%	48.60%	47.76%
(50, 50)	**False**	0.0001	10.06%	9.50%	**10.00%**

Quite similar results can be obtained when training networks with different initializations using batch normalization. A network trained without batch normalization appears to suffer from stability issues when the initial weights are too close to zero.