DD2424 Deep Learning in Data Science - Assignment 2

Marcus Hägglund - mahaggl@kth.se

Optimize the performance of the network

Now we make some changes to see if we can increase the performance of the network. There are many possible options to consider but I will mainly focus on

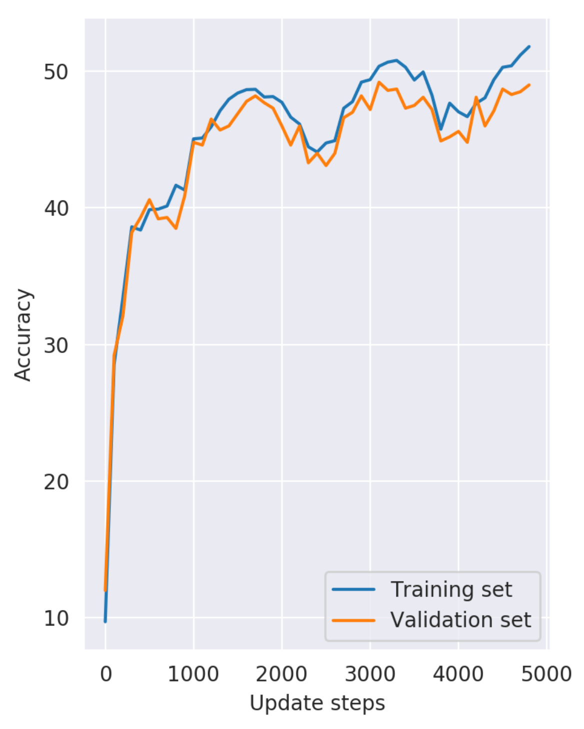
- Implement dropout
- Investigate if more hidden nodes improves the accuracy on the test data
- Add noise to the training samples

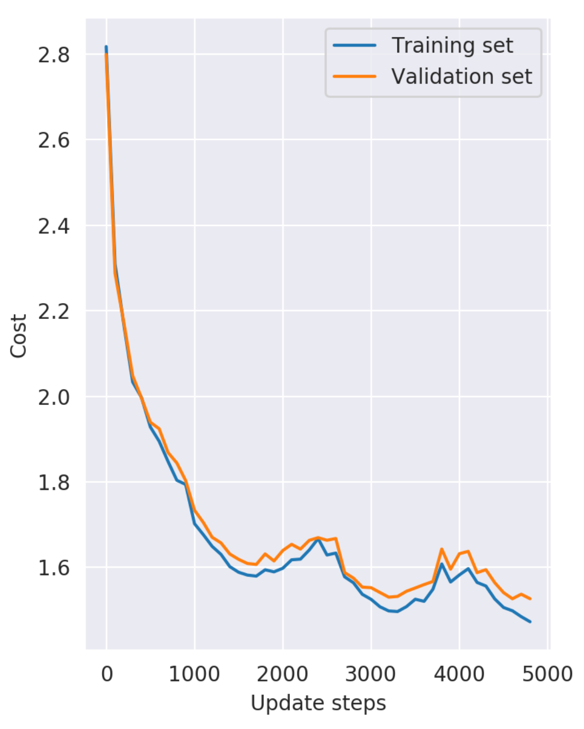
Dropout

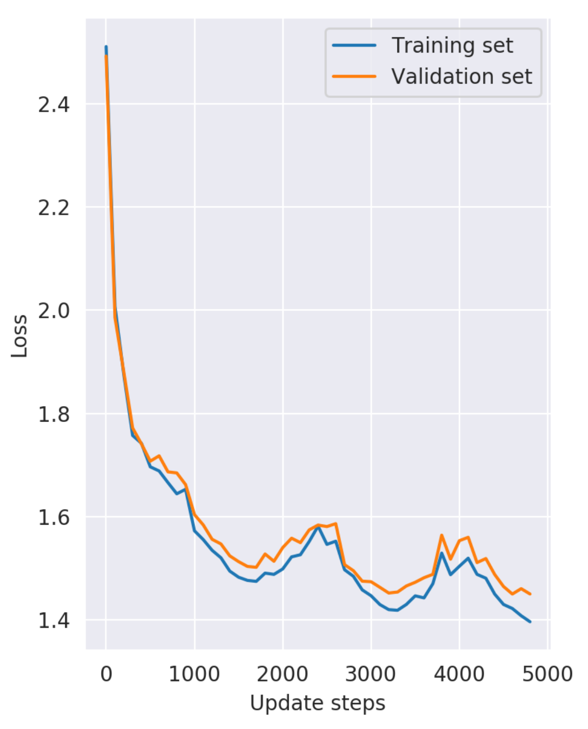
During training we'll kill the activations of neurons with a probability p for each hidden layer. By "killing" a neuron we'll set its output to zero, effectively killing the signal from that neuron, preventing it from propagating further in the network. This is a strategy used for regularization of neural networks.

Running dropout using `p=0.5` on a neural network with 50 nodes in the hidden layer we obtain the following results.

```
Model parameters:
   layer size:  50
   lambda:      0.005124
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1
   dropout:     0.5

Training data:
   accuracy (untrained):       9.71%
   accuracy (trained):         51.81%
   cost (final):               1.47
Validation data:
   accuracy (untrained):       12.00%
   accuracy (trained):         49.00%
   cost (final):               1.53
Test data:
   accuracy (untrained):       9.82%
   accuracy (trained):         49.89%
   cost (final):               1.52
```





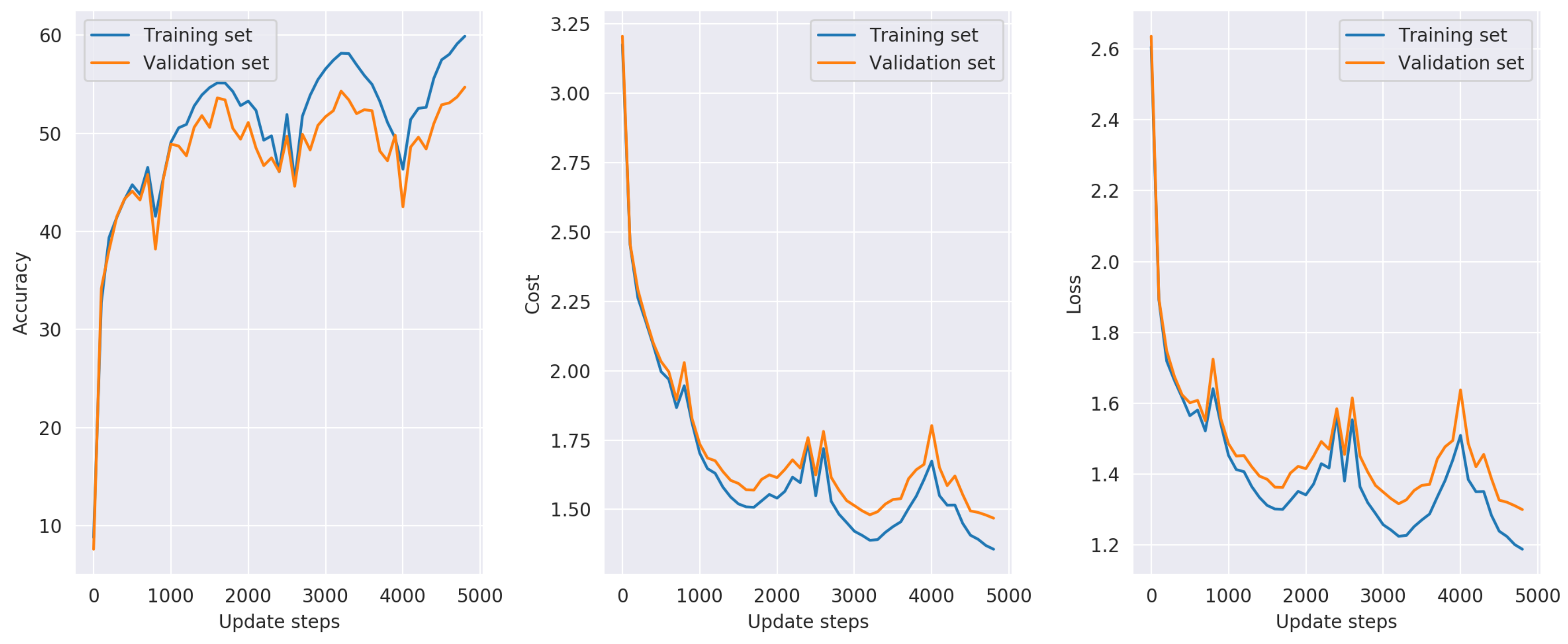


There's no real improvement on the prediction accuracy for the test and validation sets, but it had a significant impact when it comes to preventing the network from being overtrained. The accuracy on the training set is now very close to that of the validation and test sets!

Adding more nodes to the hidden layer

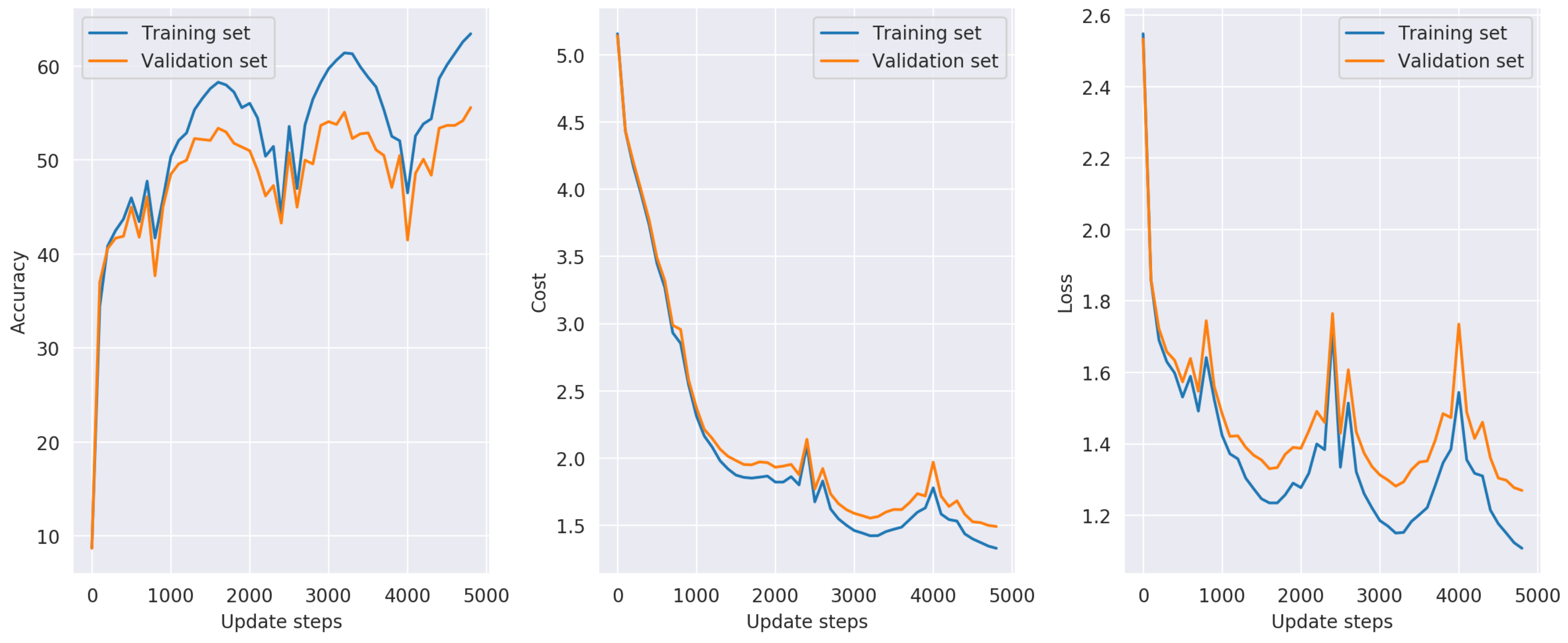
We'll first increase the number of nodes from 50 to 100 in the hidden layer.

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

Training data:
   accuracy (untrained):       8.80%
   accuracy (trained):         59.89%
   cost (final):               1.36
Validation data:
   accuracy (untrained):       7.60%
   accuracy (trained):         54.70%
   cost (final):               1.47
Test data:
   accuracy (untrained):       9.20%
   accuracy (trained):         53.30%
   cost (final):               1.50
```



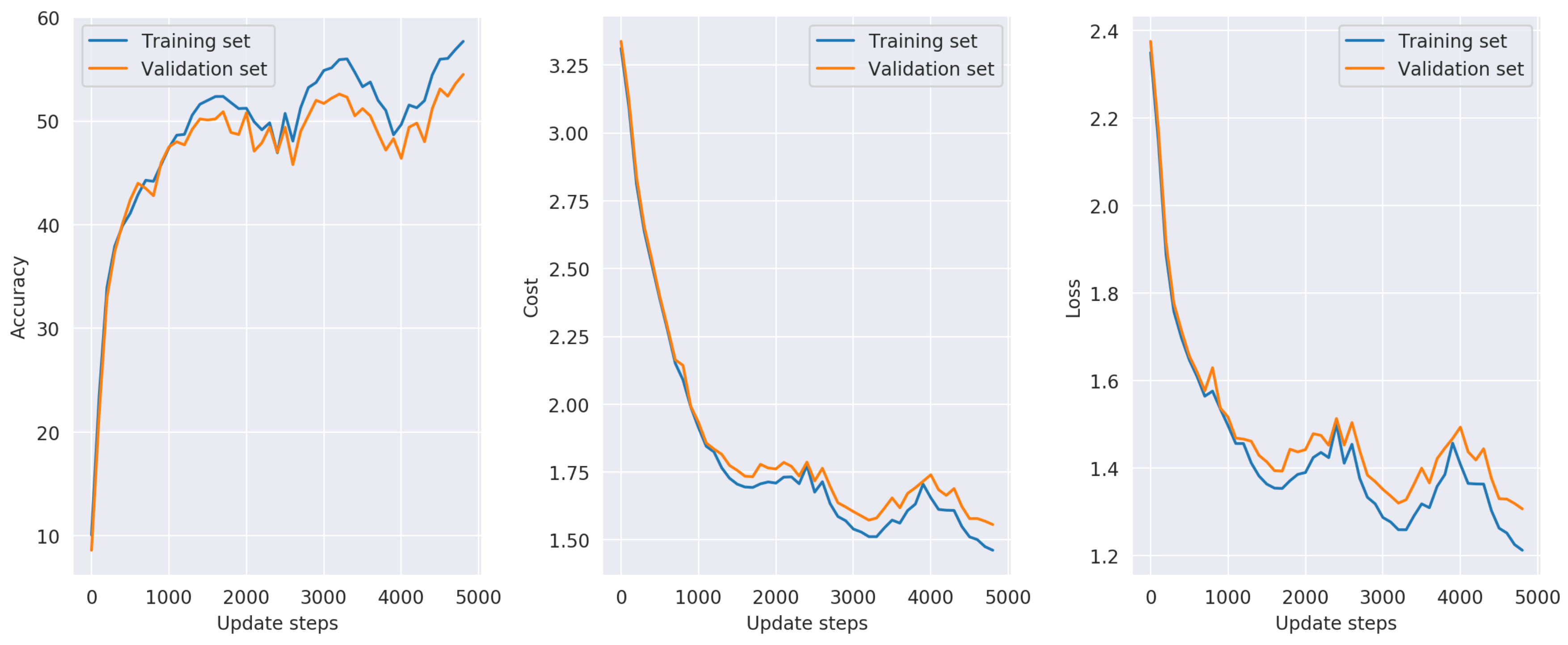
Now we increase the number of nodes even further, from 100 to 500 nodes in the hidden layer.

```
Model parameters:
   layer size:  500
   lambda:      0.005124
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1

Training data:
   accuracy (untrained):       8.74%
   accuracy (trained):         63.45%
   cost (final):               1.33
Validation data:
   accuracy (untrained):       8.70%
   accuracy (trained):         55.60%
   cost (final):               1.49
Test data:
   accuracy (untrained):       8.98%
   accuracy (trained):         54.97%
   cost (final):               1.51
```



Increasing the number of nodes seems to have helped improve the accuracy on the test data quite a bit, although we get a slightly more overfitted network. Since I've also implemented support for more than two layers I'll include a run with three hidden layers.

```
Model parameters:
   layer size:  (100, 50, 25)
   lambda:      0.005124
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.1

Training data:
   accuracy (untrained):        10.07%
   accuracy (trained):          57.69%
   cost (final):                1.46
Validation data:
   accuracy (untrained):        8.60%
   accuracy (trained):          54.50%
   cost (final):                1.56
Test data:
   accuracy (untrained):        10.17%
   accuracy (trained):          53.11%
   cost (final):                1.58
```



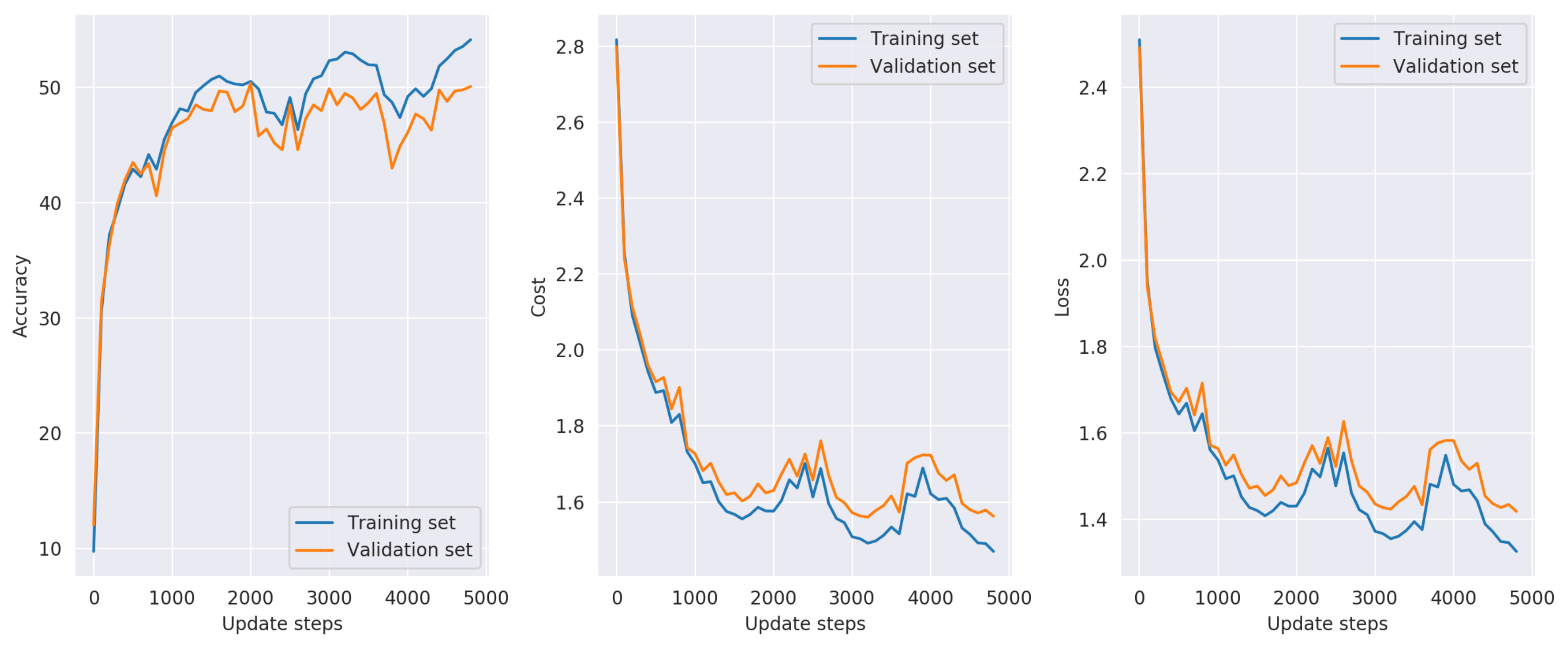
More layers also seem to improve the accuracy.

Add noise to training data

By adding noise to the data will make it more difficult for the network to make a precise fit to the training data and will therefore reduce the risk of overfitting the model.

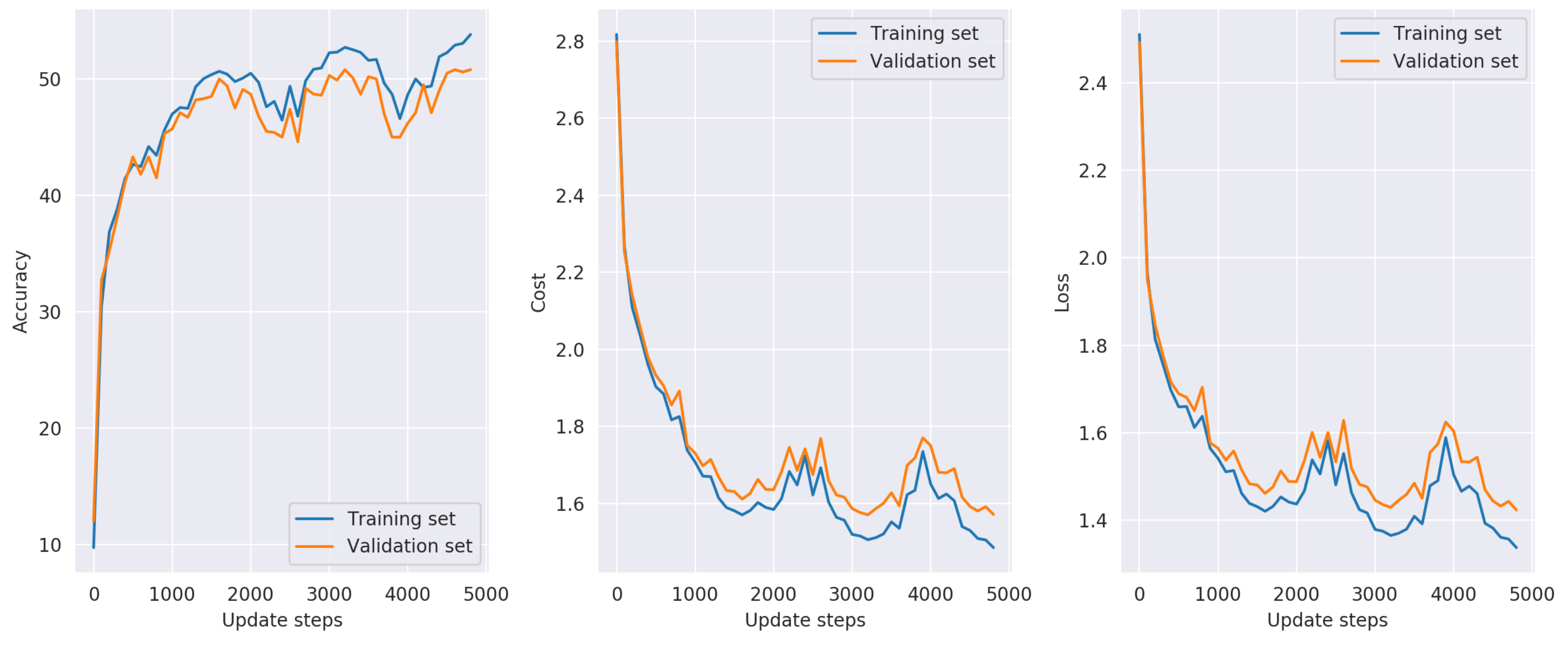
Add gaussian noise with mean 0 and standard deviation 0.01.

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

Training data:
   accuracy (untrained):        9.71%
   accuracy (trained):          54.16%
   cost (final):                1.47
Validation data:
   accuracy (untrained):        12.00%
   accuracy (trained):          50.10%
   cost (final):                1.56
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          50.52%
   cost (final):                1.59
```



Add salt&pepper noise

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

Training data:
   accuracy (untrained):        9.71%
   accuracy (trained):          53.83%
   cost (final):                1.49
Validation data:
   accuracy (untrained):        12.00%
   accuracy (trained):          50.80%
   cost (final):                1.57
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          50.48%
   cost (final):                1.60
```

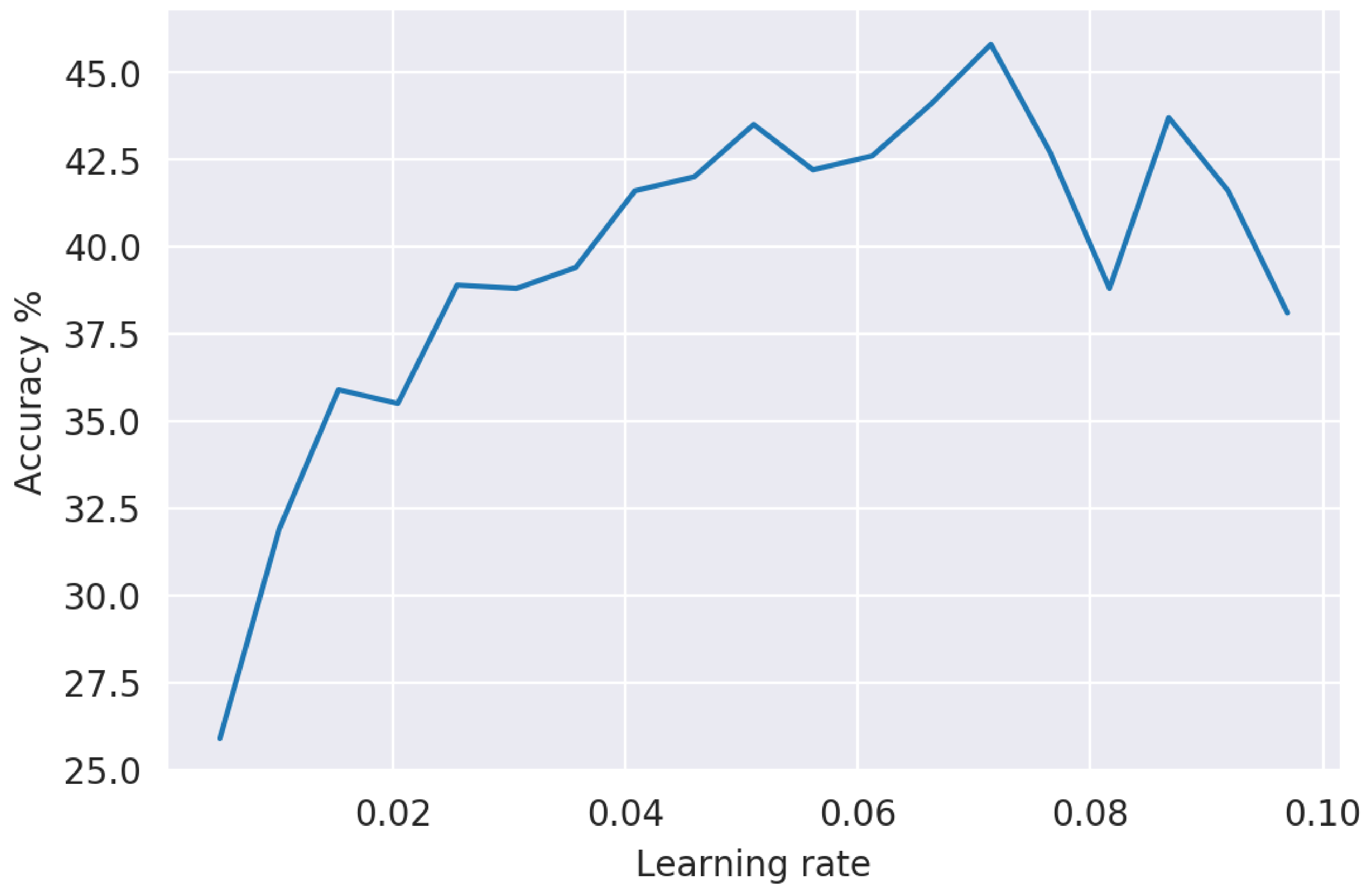


Adding noise helps with regularization of the model, slightly worse prediction accuracy.

Find *good* values for `eta_min` and `eta_max`

In order to find some decent values for our maximum and minimum learnin rate we'll perform a "Learning rate range test". The idea is to let the model run for half a cycle, allowing the learning rate to linearly increase from the minimum to the maximum value. By recording the accuracies achieved for different learning rates and then plotting these against each other. We then look at where the accuracy first starts to increase and where it starts to level out, becomes jagged or even declines. These two points are typically considered to be good upper and lower bounds for the learning rate.

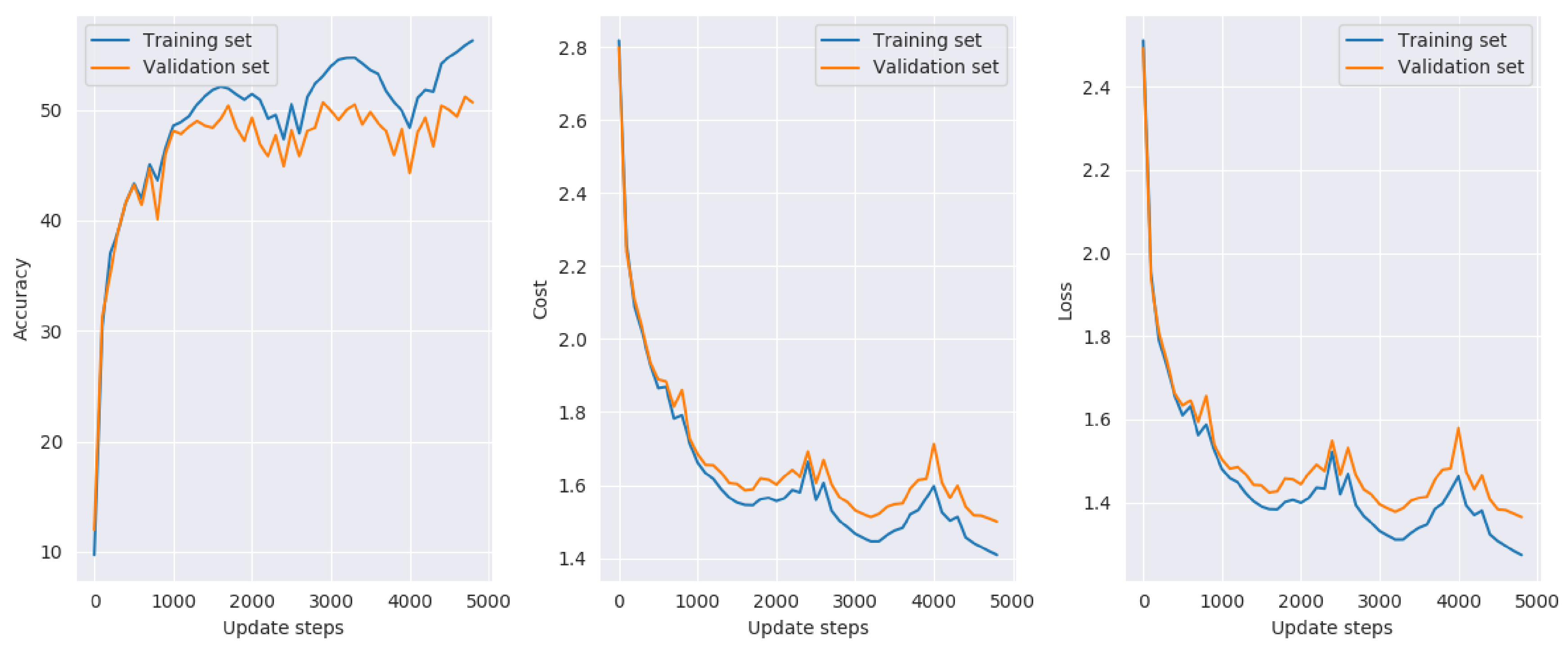
For our particular case we obtain



Juding by this plot it seems to be reasonable to set `eta_min = 1e-5` and `eta_max = 0.07`

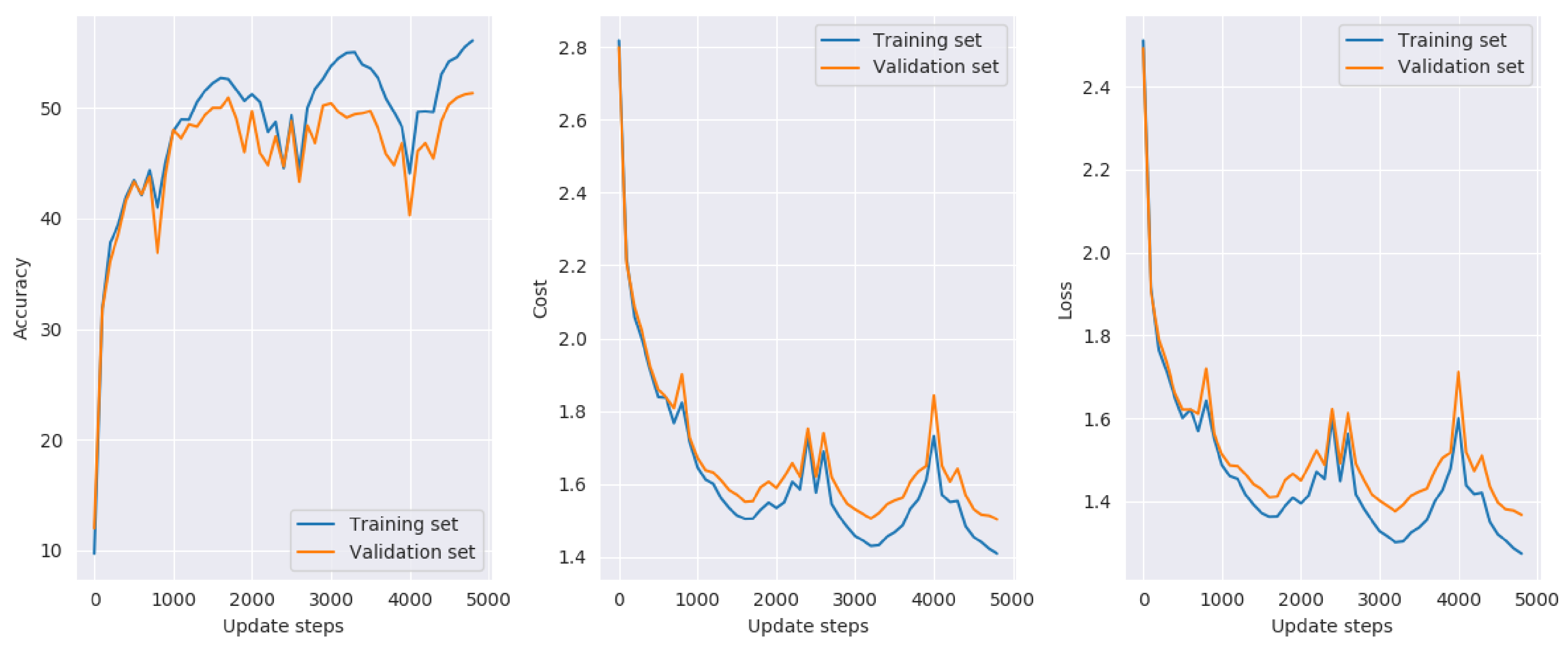
We'll finally train a model that uses the best values found for the regularization parameter and the learning rate boundary.

```
Model parameters:
   layer size:  50
   lambda:      0.005124
   cycles:      3
   n_s:         800
   n_batches:   100
   eta_min:     1e-05
   eta_max:     0.07

Training data:
   accuracy (untrained):        9.71%
   accuracy (trained):          56.27%
   cost (final):                1.41
Validation data:
   accuracy (untrained):        12.00%
   accuracy (trained):          50.70%
   cost (final):                1.50
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          51.90%
   cost (final):                1.52
```



And for reference, this is the network with the default boundaries for the learning rate.

```
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
```



Resulting in a very slight performance increase.