DD2424 Deep Learning in Data Science - Assignment 3

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

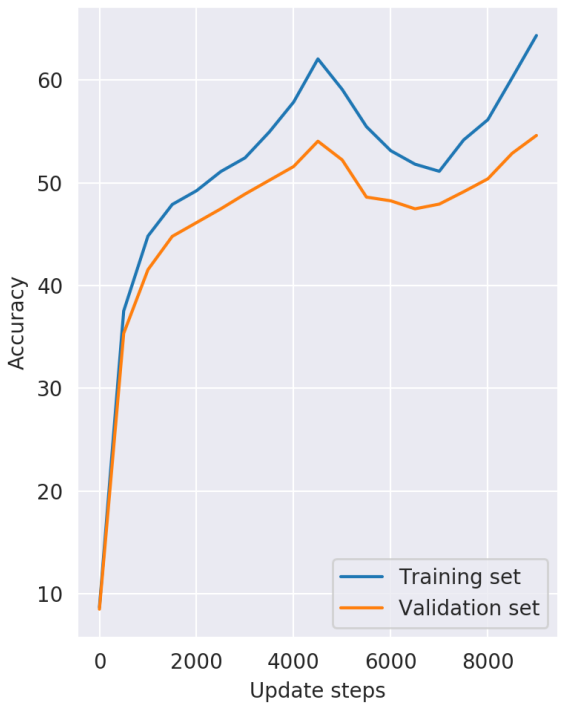
- Investigate if a deeper network architecture improves the accuracy on the test data
- Use dropout
- Add noise to the training samples

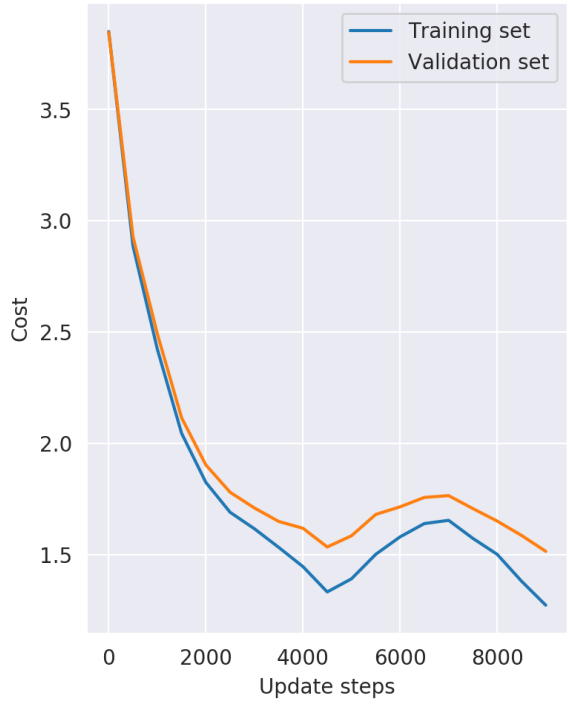
Investigate network architecture

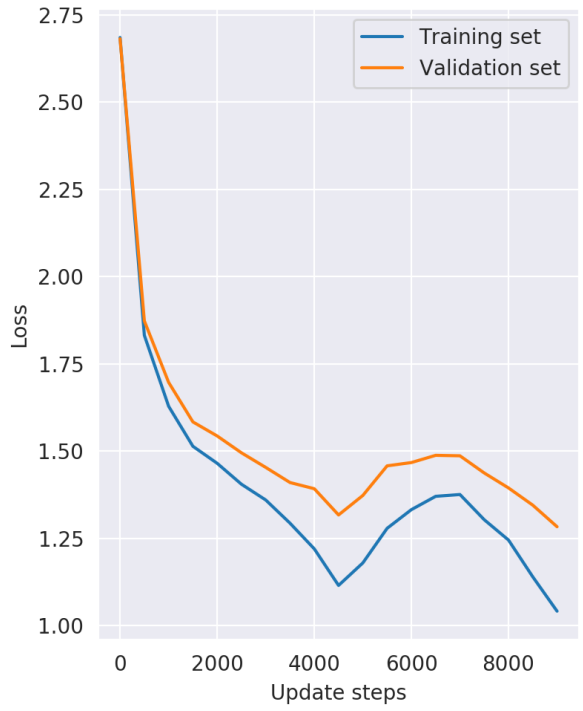
We'll use batch normalization and investigate the performance of networks with different depths.

First up is a network with 3 hidden layers

```
Model parameters:
   hidden layers:       (60, 50, 40)
   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):        8.68%
   accuracy (trained):          64.33%
   cost (final):                1.27
Validation data:
   accuracy (untrained):        8.50%
   accuracy (trained):          54.60%
   cost (final):                1.52
Test data:
   accuracy (untrained):        8.49%
   accuracy (trained):          53.72%
   cost (final):                1.54
```

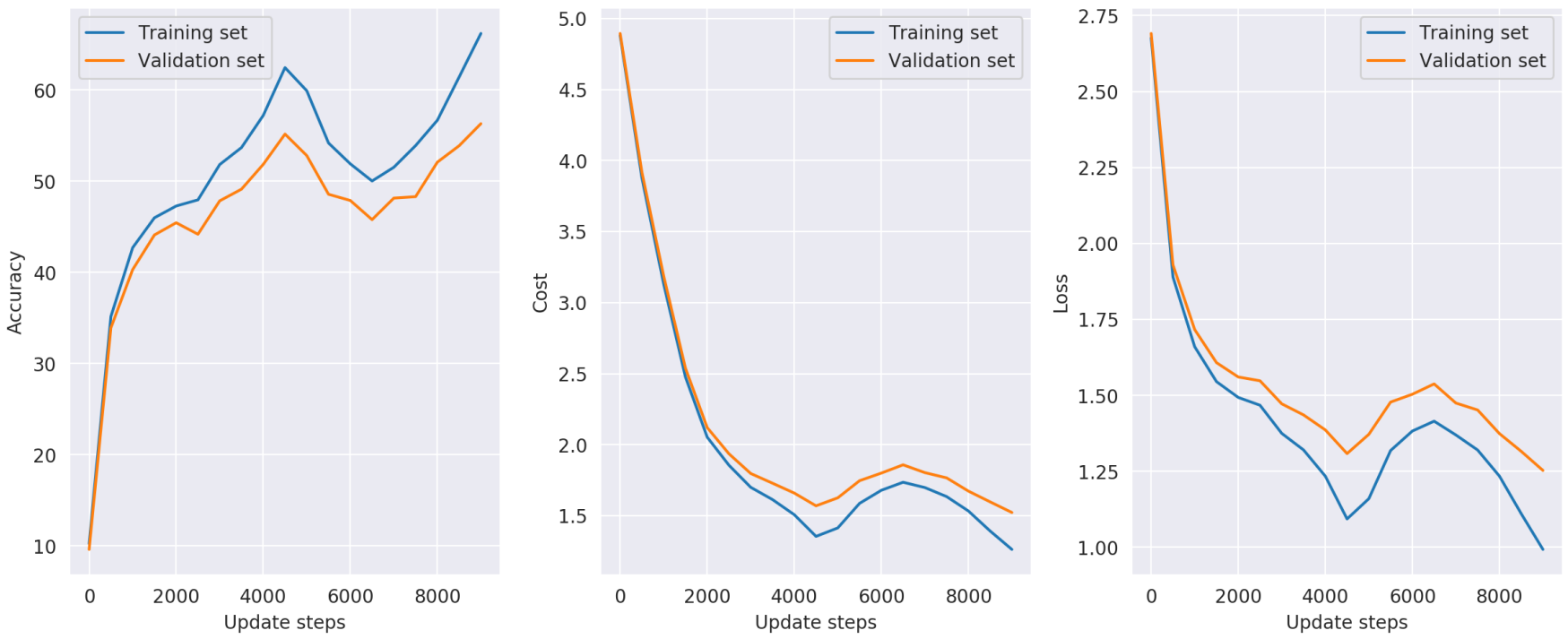






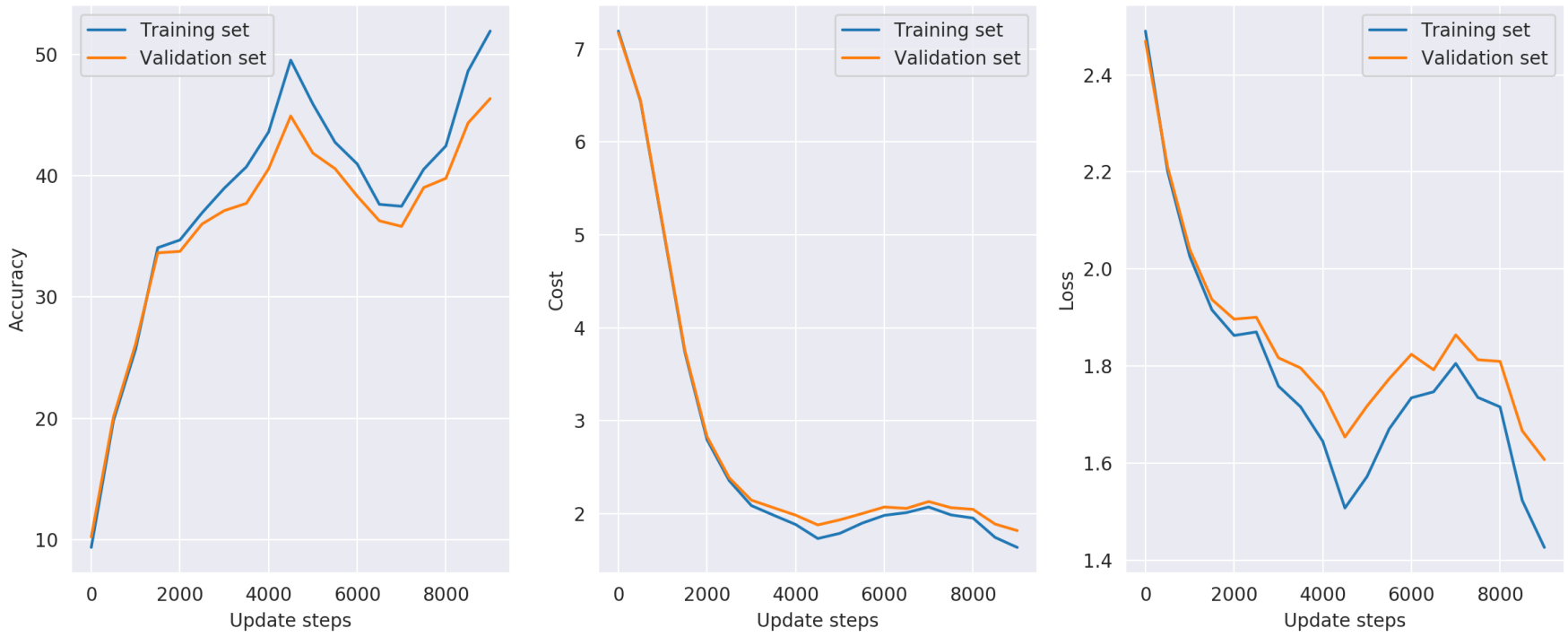
Then a network with 6 hidden layers,

```
Model parameters:
   hidden layers:       (80, 70, 60, 50, 40, 30)
   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):        10.26%
   accuracy (trained):          66.19%
   cost (final):                1.26
Validation data:
   accuracy (untrained):        9.62%
   accuracy (trained):          56.30%
   cost (final):                1.52
Test data:
   accuracy (untrained):        10.53%
   accuracy (trained):          54.83%
   cost (final):                1.57
```



And finally a network with 12 hidden layers

```
Model parameters:
   hidden layers:       (120, 110, 100, 90, 80, 70, 60, 50, 40, 30,
20, 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):        9.35%
   accuracy (trained):          51.92%
   cost (final):                1.64
Validation data:
   accuracy (untrained):        10.22%
   accuracy (trained):          46.36%
   cost (final):                1.82
Test data:
   accuracy (untrained):        10.26%
   accuracy (trained):          45.61%
   cost (final):                1.84
```



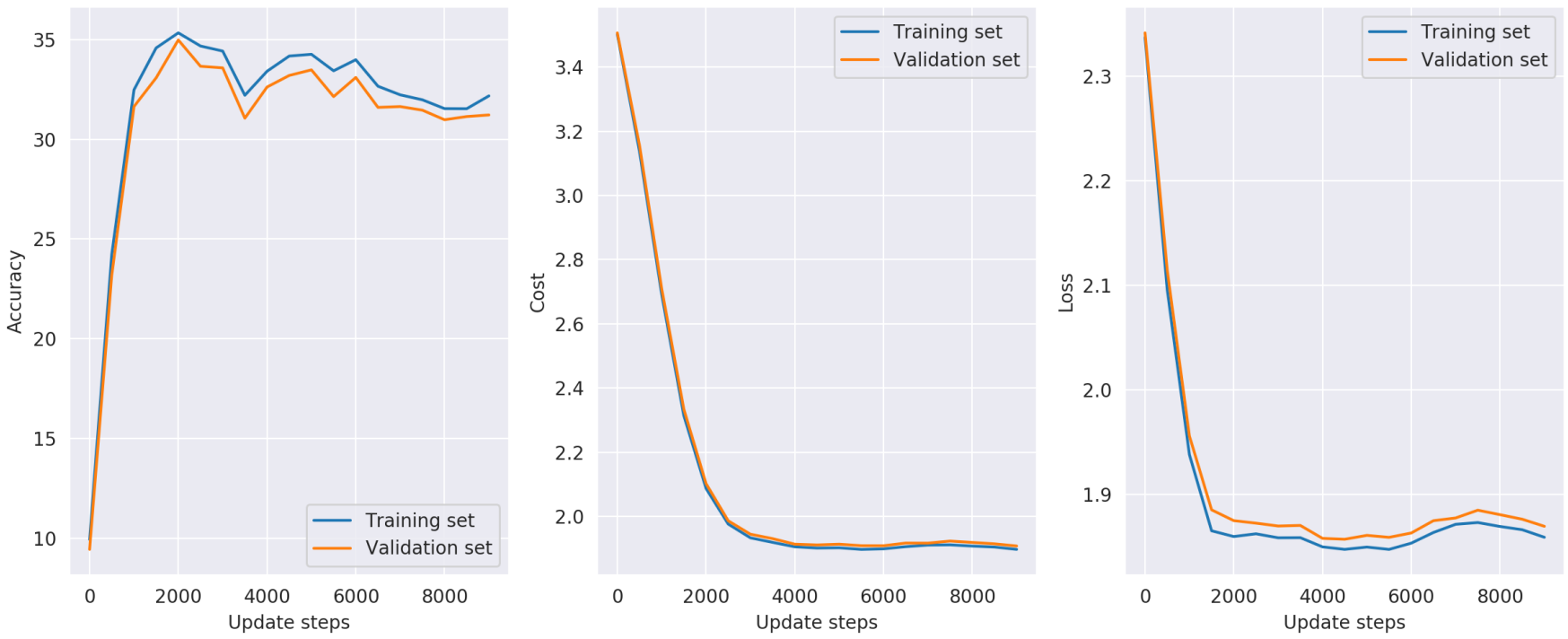
Increasing the number of hidden layers allows the network to learn more of the patterns in the training data, increasing its training accuracy. There seems to be a sweet spot where a further increase of the depth of the neural network does not translate into better performance. The increased complexity of the network will instead likely cause it to become overfitted and as such it'll perform worse on the unseen test data. In order to prevent the more complex network from becoming overfitted we may need to look into some different methods of increasing the amount of regularization. We'll therefore consider implementing dropout and adding random noise to the training data.

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 which prevents it from propagating further in the network. This is a common method used to increase the amount of regularization in the network.

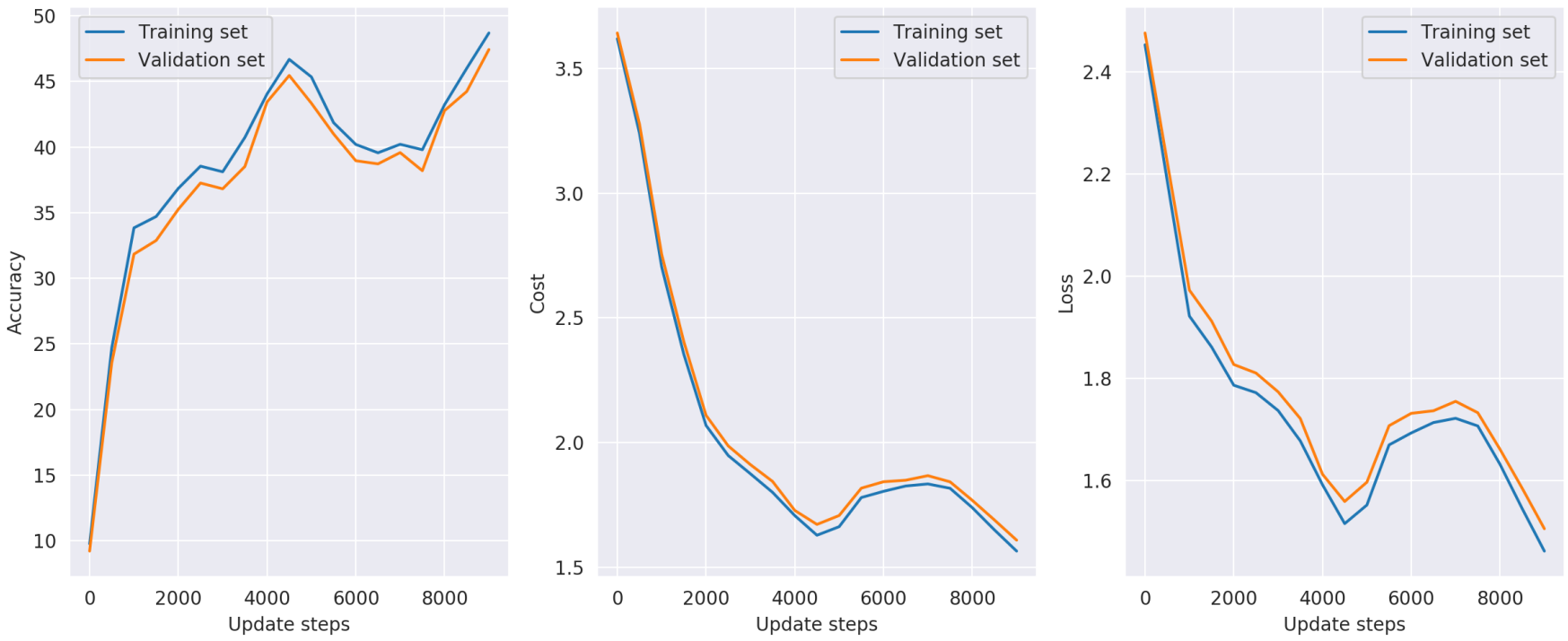
Running dropout using `p=0.5` on a neural network with a deeper architecture and no batch normalization.

```
Model parameters:
   hidden layers:       (60, 50, 40)
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he
   dropout:             0.5
   shuffle:             True

Training data:
   accuracy (untrained):        9.93%
   accuracy (trained):          32.18%
   cost (final):                1.90
Validation data:
   accuracy (untrained):        9.44%
   accuracy (trained):          31.22%
   cost (final):                1.91
Test data:
   accuracy (untrained):        9.67%
   accuracy (trained):          31.74%
   cost (final):                1.90
```



Training the same network as above but now with batch normalization,

```
Model parameters:
   hidden layers:       (60, 50, 40)
   BN:                  True
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he
   dropout:             0.5
   shuffle:             True

Training data:
   accuracy (untrained):        9.75%
   accuracy (trained):          48.70%
   cost (final):                1.57
Validation data:
   accuracy (untrained):        9.20%
   accuracy (trained):          47.44%
   cost (final):                1.61
Test data:
   accuracy (untrained):        9.32%
   accuracy (trained):          47.07%
   cost (final):                1.60
```



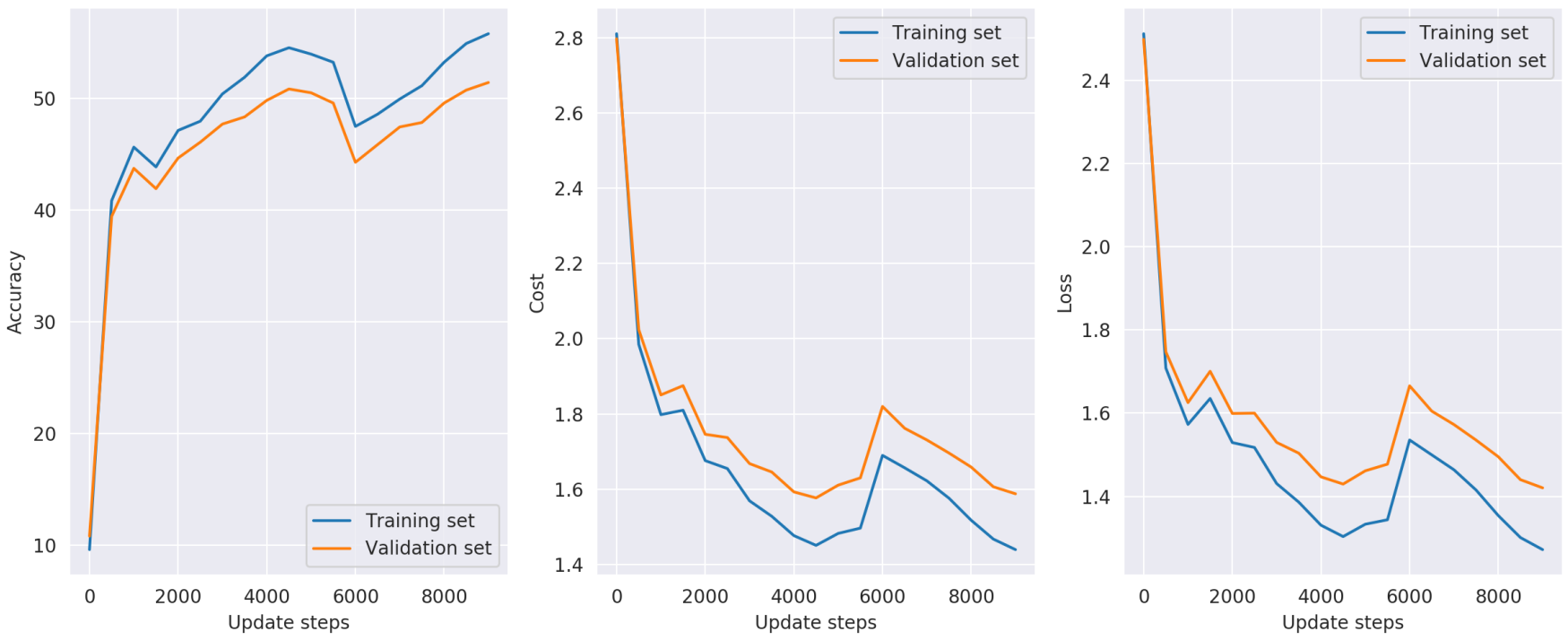
It seems that training the network with batch normalization eliminates the need for dropout since it provides similar regularization.

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.

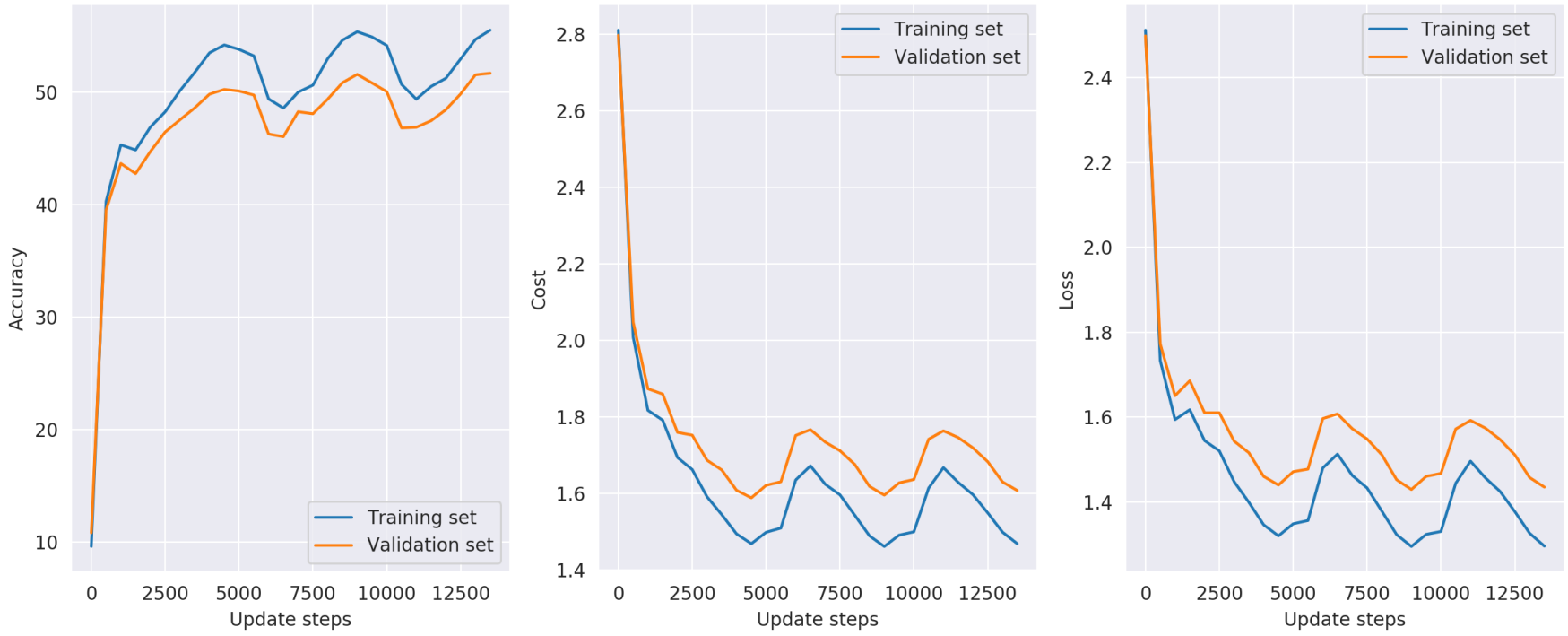
Add gaussian noise with mean 0 and standard deviation 0.01.

```
Model parameters:
   hidden layers:       50
   BN:                  False
   lambda:              0.005
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   noise:               gaussian

Training data:
   accuracy (untrained):        9.63%
   accuracy (trained):          55.81%
   cost (final):                1.44
Validation data:
   accuracy (untrained):        10.84%
   accuracy (trained):          51.44%
   cost (final):                1.59
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          50.86%
   cost (final):                1.60
```



Add salt&pepper noise

```
Model parameters:
   hidden layers:       50
   BN:                  False
   lambda:              0.005
   cycles:              3
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   noise:               s&p

Training data:
   accuracy (untrained):        9.63%
   accuracy (trained):          55.51%
   cost (final):                1.47
Validation data:
   accuracy (untrained):        10.84%
   accuracy (trained):          51.68%
   cost (final):                1.61
Test data:
   accuracy (untrained):        9.82%
   accuracy (trained):          50.67%
   cost (final):                1.63
```

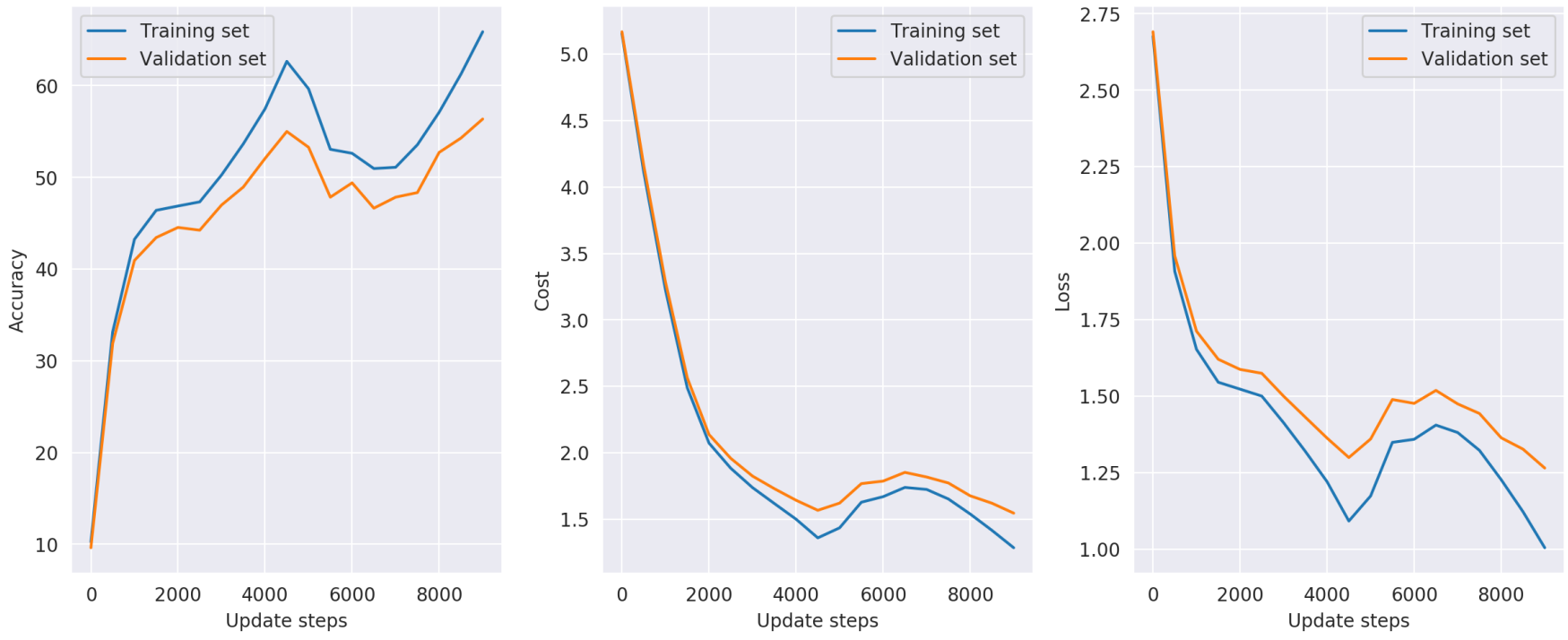


The added noise did indeed add some regularization to the model since there was a reduction in the prediction accuracy on the training data. This did however not give any improvements on the prediction accuracy on the test data, it even got slightly worse.

Training the final model

Finally by combining some of the improvements mentioned above and a bit of trial and error the best network found was the following,

```
Model parameters:
   hidden layers:       (80, 70, 60, 50, 40, 30)
   BN:                  True
   lambda:              0.005623
   cycles:              2
   n_s:                 2250
   n_batches:           100
   eta_min:             1e-05
   eta_max:             0.1
   initialization:      he
   shuffle:             True

Training data:
   accuracy (untrained):        10.26%
   accuracy (trained):          65.87%
   cost (final):                1.28
Validation data:
   accuracy (untrained):        9.62%
   accuracy (trained):          56.36%
   cost (final):                1.54
Test data:
   accuracy (untrained):        10.53%
   accuracy (trained):          55.13%
   cost (final):                1.57
```



Which has a 55.13% prediction accuracy on the test data.