Deep Learning in Data Science: **Training a multi-linear classifier with k-layer networks**

Alexander Bea - abea@kth.se

Option 1, Assignment 3

In this assignment I had to train and test k-layer networks with multiple outputs to classify images from the CIFAR-10 dataset.

[1] Installers

[2] Import libraries

[3] Functions: Decoding and displaying images

EXERCISE 1

Upgrade Assignment 2 code to train & test k-layer networks

- Initialize and store the parameters of my network
- Apply the network input vectors and keep a record of the intermediary scores when I apply the network (forward pass)
- Compute the gradient of the cost function for a mini-batch relative to the parameters of the network

[4] Functions: Load data_batches (1, 2 and test) for training, validation and test

The following code is necessary to create my network layers.

[5] Functions: Create layers

[6] Functions: Initialize network (Change initiazliation herein for testing in the last assignment v)

The following function (Evalute Classifier, Cost and Accuracy) is from my previous assignment, however, they have been altered in small extent to consider new features of this assignment

[7] Functions: Evaluate Classifier, Cost and Accuracy

In the following the network parameters are intialized starting with a 2-layer network. Careful initialization is applied using He initialization.

```
[8]  data, labels = trainOnSmallDataBatch()
     layers = CreateLayers(
             shapes=[(50, 3072), (10, 50)],
             activations=["relu", "softmax"])
     clf = NetClassifier(data, labels, layers)
```

Compute the gradients for the network parameters

Next, the functions to compute the gradients (copied and altered from my previous assignment) of my k-layer network w.r.t. its weight and bias parameters.

[9] Functions: Compute Gradients

[10] Functions: Compare Gradients

i) HOW I CHECKED MY ANALYTICAL GRADIENTS

The following generates the gradient comparing result that shows that the implemented analytical gradient method is close enough to be regarded as accurate. The gradients are compared to a numerical method (centered difference method). A four layer neural network without batch normalization is applied on a reduced dataset of 5 images with 30 dimensions. What is noteworthy is that the discrepency between the analytical and numerical gradients increase for the earlier layers as the gradient is back-propogated through the network. Checks are done with no regularization i.e. lambda = 0

[11] Compare Analytical with Numerical Gradient

```
Method Comparison: Max Err between Analytical vs Numerical
+-----------+------------+---------------------+
| Gradient  |   Method   | Rel Diff Max [e-06] |
+===========+============+=====================+
| layer 1 W | ANL vs NUM | 3.260               |
+-----------+------------+---------------------+
| layer 1 b | ANL vs NUM | 0.981               |
+-----------+------------+---------------------+
| layer 2 W | ANL vs NUM | 9.083               |
+-----------+------------+---------------------+
| layer 2 b | ANL vs NUM | 0.934               |
+-----------+------------+---------------------+
| layer 3 W | ANL vs NUM | 2.664               |
+-----------+------------+---------------------+
| layer 3 b | ANL vs NUM | 0.023               |
+-----------+------------+---------------------+
| layer 4 W | ANL vs NUM | 95107.302           |
+-----------+------------+---------------------+
| layer 4 b | ANL vs NUM | 0.074               |
+-----------+------------+---------------------+
```

EXERCISE 2

Training multi-layer networks

Train the network using mini-batch gradient descent and cyclical learning rates and without batch normalization. Firstly, adapt the code from my previous assignment accordingly:

[12] Functions: Compute Mini Batch Gradient Descent

[13] Functions: Load All Data

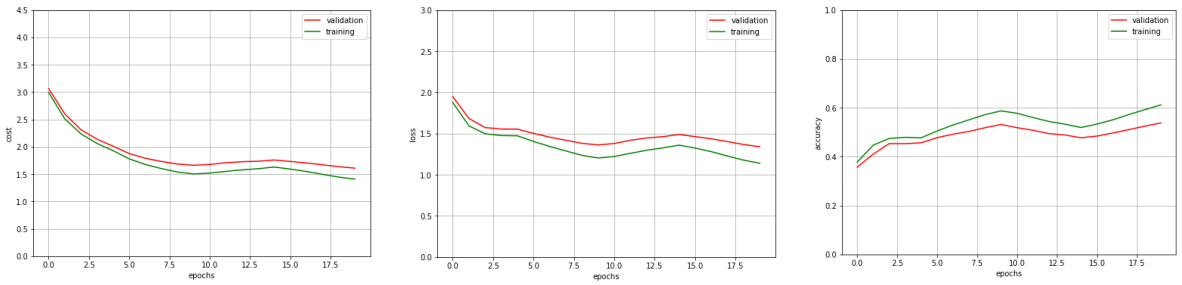
Now, the training function is defined where the hyper-parameters setting is n batch=100, eta min = 1e-5, eta max = 1e-1, lambda=.005, two cycles of training and n s = 5 * 45,000 / n batch (2250)

[14] Code: Training

Now run on a 2-layer networ with cyclical learning rate. Here it is clear that the results from assignemnt 2 are replicated. Test accuracy is displayed below:

```
[15] two_layers = CreateLayers(shapes=[(50, 3072), (10, 50)],activations=["relu", "softmax"])
     Training(two_layers, BN=False, trainingTimes=1)
```

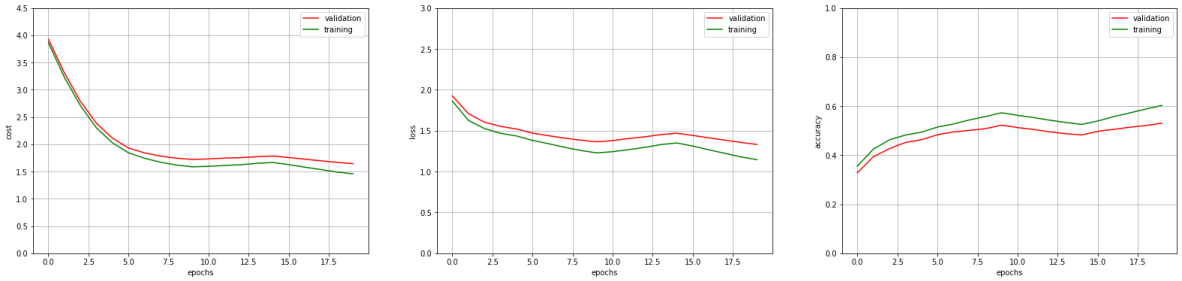
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.612          | 0.538        | 0.523         |
+----------------+--------------+---------------+
```



ii) GRAPHS OF EVOLUTION OF LOSS WITHOUT BATCH NORMALIZATION FOR 3-LAYER NETWORK

Next, a 3-layer network is produced with the same hyper-paramaters and no batch normalization.

```
[16] three_layers = CreateLayers(shapes=[(50, 3072), (50, 50), (10, 50)],activations=["relu", "relu", "softmax"])
     Training(three_layers, BN=False, trainingTimes=1)
```

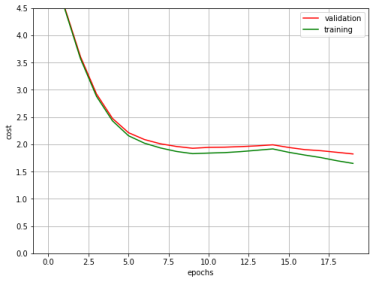


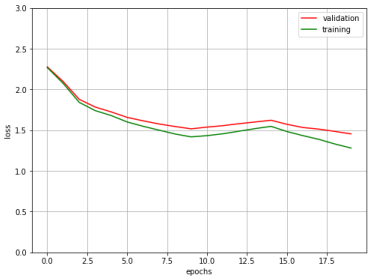
iii) GRAPHS OF EVOLUTION OF LOSS WITHOUT BATCH NORMALIZATION FOR 9-LAYER NETWORK

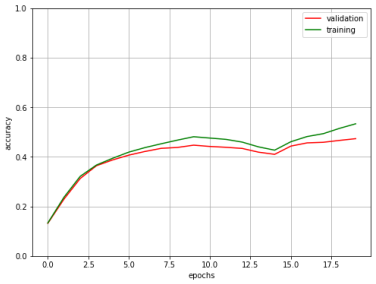
Next, a 9-layer network is produced with the same hyper-paramaters and no batch normalization.

```
[17] nine_layers = CreateLayers(shapes=[(50, 3072), (30, 50), (20, 30), (20, 20), (10, 20),(10, 10), (10, 10),
                                        (10, 10), (10, 10)],
            activations=["relu", "relu", "relu", "relu", "relu", "relu","relu", "relu", "softmax"])
     Training(nine_layers, BN=False, trainingTimes=1) # 9-layer model w\o batch norm
```

```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.533          | 0.473        | 0.466         |
+----------------+--------------+---------------+
```







EXERCISE 3

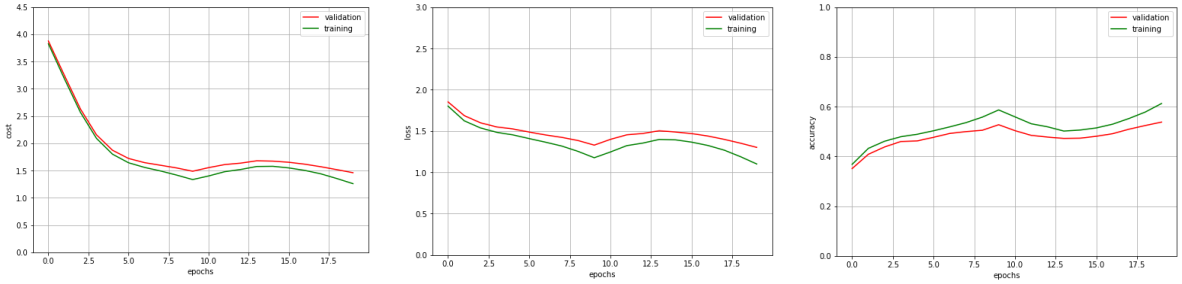
Implement batch normalization

ii) GRAPHS OF EVOLUTION OF LOSS WITH BATCH NORMALIZATION FOR 3-LAYER NETWORK

In the following batch normalization is applied, first to three layer network. If compated to the results above, there is not much difference.

```
[18] Training(three_layers, BN=True, trainingTimes=1)
```

```
**************************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.613          | 0.538        | 0.532         |
+----------------+--------------+---------------+
```

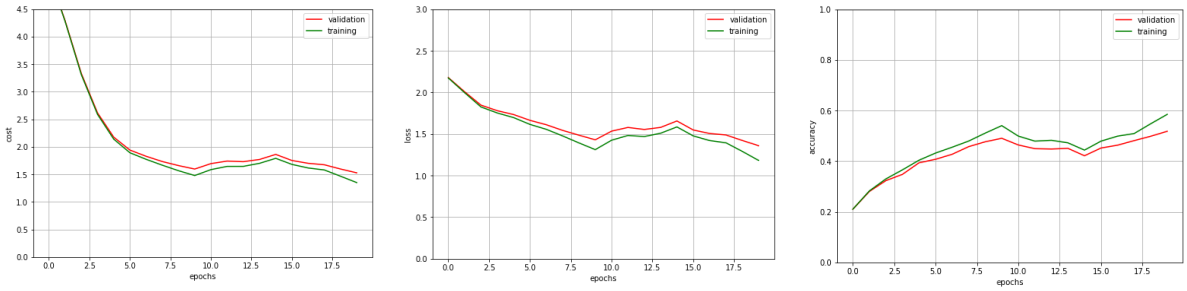


iii) GRAPHS OF EVOLUTION OF LOSS WITH BATCH NORMALIZATION FOR 9-LAYER NETWORK

However, as can be observed in the next results, whereby batch normalization is applied to the 9 layer network, it has a signifant impact. It is clear that for deeper neural network, batch normalization is very important.

```
[19] Training(nine_layers, BN=True, trainingTimes=1)
```

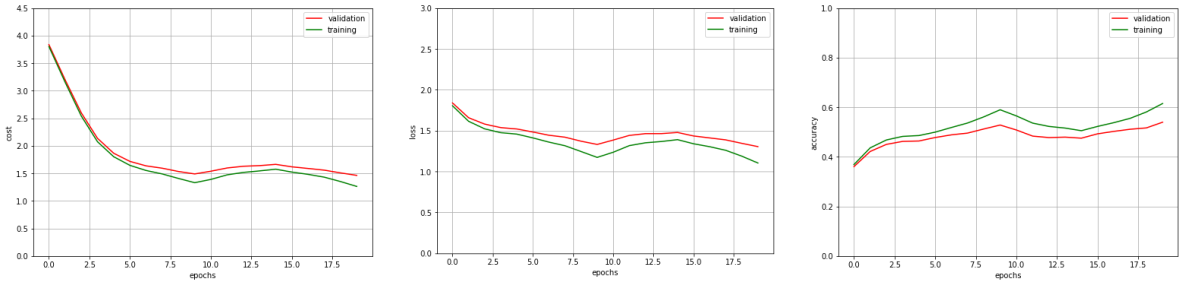
```
**************************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.585          | 0.518        | 0.510         |
+----------------+--------------+---------------+
```



Moving on, we want to replicate the same test accuracy in a 3 layer network as in the assignment specification of approx. 53.5% after training with the following hyper parameters He initialization and hyper-parameter settings of eta min = 1e-5, eta max = 1e-1, lambda=.005, two cycles of training and n s = 5 * 45,000 / n batch. After training the model 6 times, approx. 54% is achieved.

```
[20] Training(three_layers, BN=True, trainingTimes=6)
```

```
**************************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.617          | 0.539        | 0.527         |
+----------------+--------------+---------------+
```



iv) RANGE OF VALUES IN SEARCH FOR LAMBDA FOR 3-LAYER NETWORK WITH BATCH NORMALIZATION

Thereafter, the coarse search for lambda is performed.

[25] Functions: Search for Lambda

[26] Code: Coarse Search

```
Lambda test done:0.01
Lambda test done:0.02
Lambda test done:0.03
Lambda test done:0.04
Lambda test done:0.05
Lambda test done:0.06
********** COARSE SEARCH **************************
n epochs=20
************************************************************
+-------------+----------------+--------------+---------------+
| Parameters  | Train Accuracy | Val Accuracy | Test Accuracy |
+=============+================+==============+===============+
| lambda=0.01 | 0.596          | 0.538        | 0.525         |
+-------------+----------------+--------------+---------------+
| lambda=0.02 | 0.574          | 0.531        | 0.522         |
+-------------+----------------+--------------+---------------+
| lambda=0.03 | 0.559          | 0.526        | 0.516         |
+-------------+----------------+--------------+---------------+
| lambda=0.04 | 0.546          | 0.514        | 0.512         |
+-------------+----------------+--------------+---------------+
| lambda=0.05 | 0.540          | 0.511        | 0.509         |
+-------------+----------------+--------------+---------------+
| lambda=0.06 | 0.534          | 0.508        | 0.505         |
+-------------+----------------+--------------+---------------+
```

In the above coarse search I randomly selected five very low regularzation terms. In the result table it becomes clear that the validation accuracy is highest where lambda is below 0.02, thereby the fine search will be between 0.005 and 0.02

Continuing, the fine search for lambda is performed.

[28] Code: Fine Search

```
Lambda test done:0.0051
Lambda test done:0.0063
Lambda test done:0.0069
Lambda test done:0.0094
Lambda test done:0.0134
Lambda test done:0.0154
********************* FINE SEARCH *************************
n epochs=20
************************************************************
```

Parameters	Train Accuracy	Val Accuracy	Test Accuracy
lambda=0.0051	0.620	0.534	0.528
lambda=0.0063	0.609	0.535	0.528
lambda=0.0069	0.609	0.538	0.532
lambda=0.0094	0.602	0.537	0.532
lambda=0.0134	0.588	0.543	0.527
lambda=0.0154	0.583	0.532	0.529

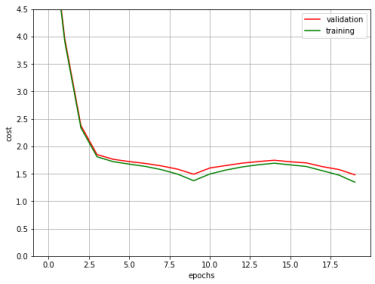
The best fine search that yielded the highest accuracy on the validation set was at lambda=0.0134 as is displayed in the table below. For future work, I would explore running a much wider random search.

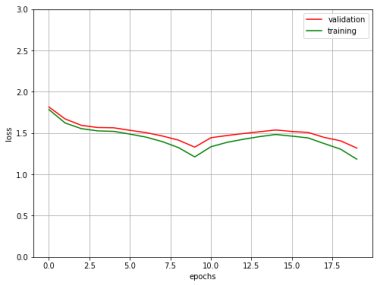
Finally, the optimal lambda is implemented

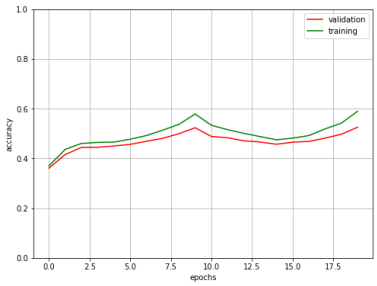
```
[32] Training(three_layers, BN=True, trainingTimes=6, lamda=0.0134)
```

```
**********************************************
```

Train Accuracy	Val Accuracy	Test Accuracy
0.587	0.532	0.527







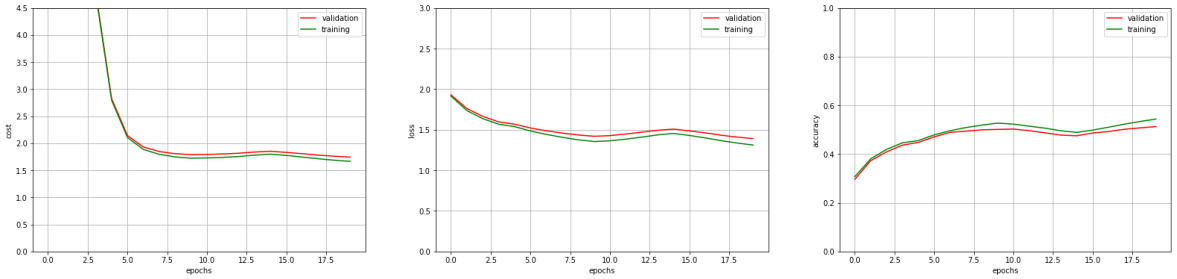
v) SENSITIVITY TO INITIALIZATION

Next we need to implement a sensitivity analysis to the initialization where I experiment on training with batch normalization and without it. Instead of He initialization I will apply sig=1e-1, 1e-3 and 1e-4. In order to test this

sig=1e-1 initialization without BN

```
[25] Training(three_layers, BN=False, trainingTimes=6, lamda=0.0134) # sig=1e-1
```

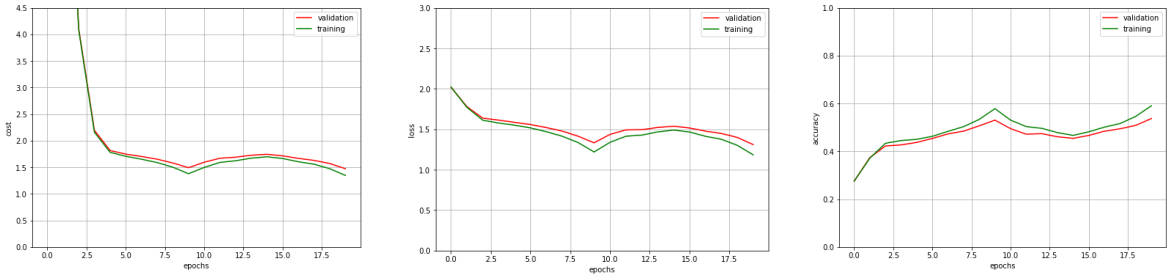
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.547          | 0.517        | 0.515         |
+----------------+--------------+---------------+
```



sig=1e-1 initialization with BN

```
[26] Training(three_layers, BN=True, trainingTimes=6, lamda=0.0134) # sig=1e-1
```

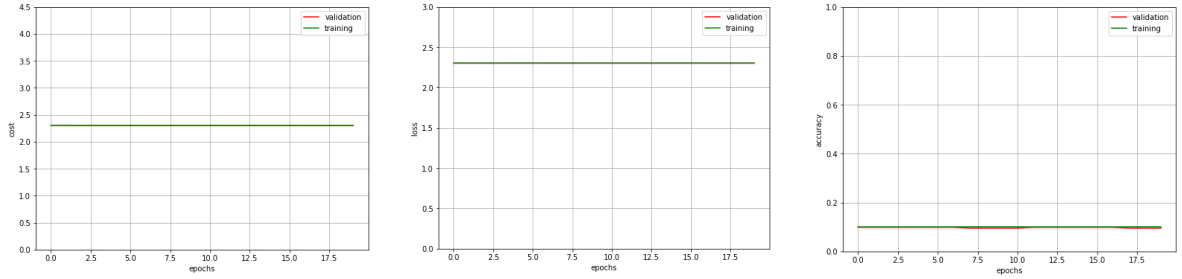
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.589          | 0.539        | 0.528         |
+----------------+--------------+---------------+
```



sig=1e-3 initialization without BN

```
[25] Training(three_layers, BN=False, trainingTimes=6, lamda=0.0134) # sig=1e-3
```

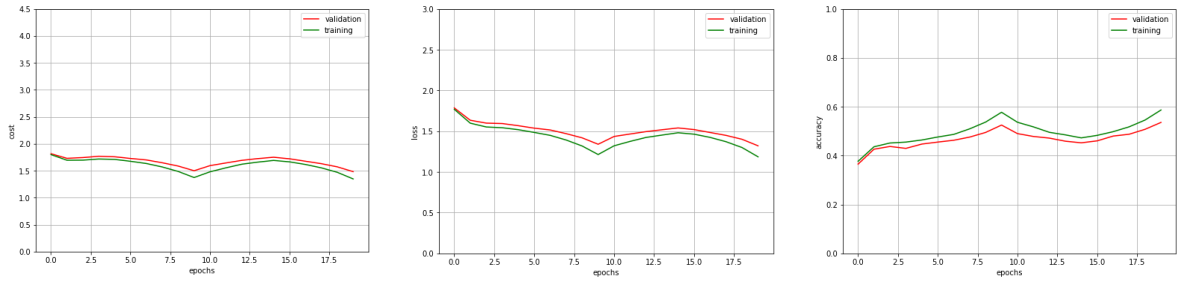
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.101          | 0.095        | 0.100         |
+----------------+--------------+---------------+
```



sig=1e-3 initialization with BN

```
[26] Training(three_layers, BN=True, trainingTimes=6, lamda=0.0134) # sig=1e-4
```

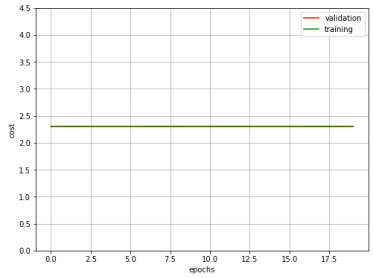
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.585          | 0.534        | 0.528         |
+----------------+--------------+---------------+
```

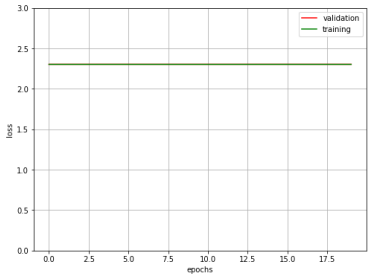


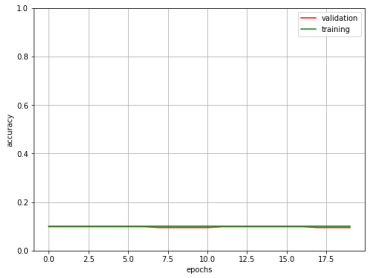
sig=1e-4 initialization without BN

```
[25] Training(three_layers, BN=False, trainingTimes=6, lamda=0.0134) # sig=1e-4
```

```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.101          | 0.095        | 0.100         |
+----------------+--------------+---------------+
```



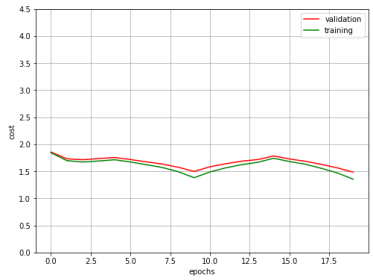


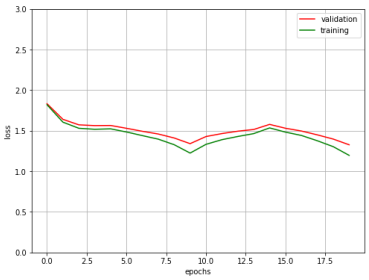


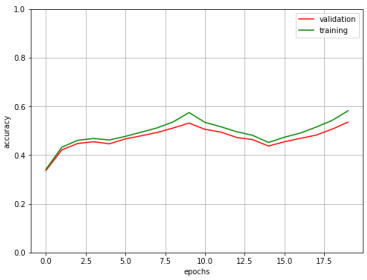
sig=1e-4 initialization with BN

```
[26] Training(three_layers, BN=True, trainingTimes=6, lamda=0.0134) # sig=1e-4
```

```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.586          | 0.534        | 0.527         |
+----------------+--------------+---------------+
```



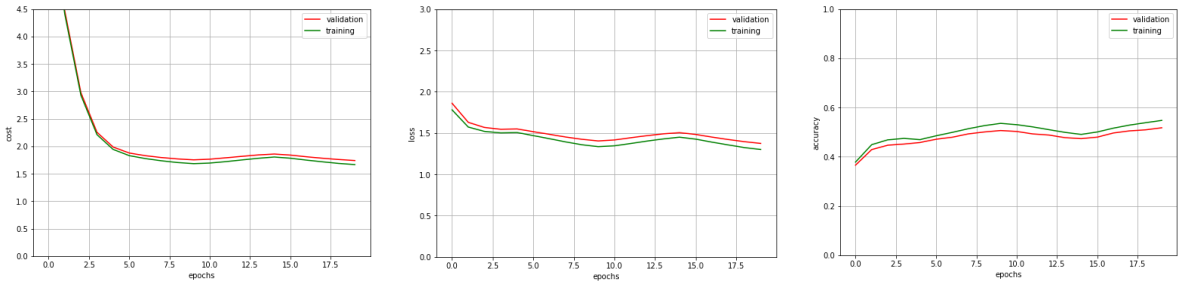




He initialization without BN

```
[33] Training(three_layers, BN=False, trainingTimes=6, lamda=0.0134) # He
```

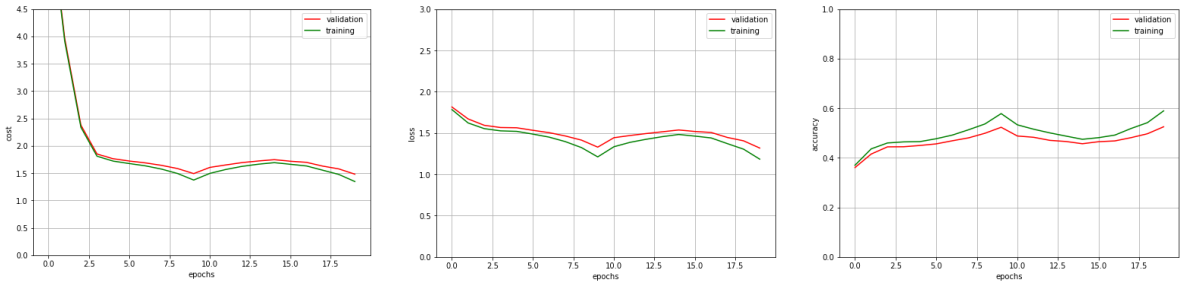
```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.550          | 0.515        | 0.517         |
+----------------+--------------+---------------+
```



He initialization with BN

```
[34] Training(three_layers, BN=True, trainingTimes=6, lamda=0.0134) # He
```

```
**********************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.587          | 0.532        | 0.527         |
+----------------+--------------+---------------+
```



As can be observed in the loss plots above, the networks are less sensitive to weight initialization when I train with batch normalization. It be concluded that batch normalization helps significantly the network to learn even if we start of with a bad initialization.