Report k-layer-network

1 Analytic gradient check

- Check analytic gradient computations: First, I implemented the whole code for k layers. To test the computation, I replicated the assignment 2, 2-layers with 50 nodes and I compared if the gradients were the same for the old and new computation:

  ```
  xs, p = evaluate_classifier(data, Ws, bs)
  grad_Ws, grad_bs = compute_gradient(onehot_labels, p, Ws, xs, lbd=0)
  grad_W1_old, grad_b1_old, grad_W2_old, grad_b2_old =
  compute_gradient_old(data, onehot_labels, p, Ws, xs, lbd=0)
  assert (grad_Ws[0]-grad_W1_old).all() == 0, "Wrong computation of gradient of W1"
  assert (grad_Ws[1]-grad_W2_old).all() == 0, "Wrong computation of gradient of W2"
  assert (grad_bs[0]-grad_b1_old).all() == 0, "Wrong computation of gradient of b1"
  assert (grad_bs[1]-grad_b2_old).all() == 0, "Wrong computation of gradient of b1"
  ```

  I got the 0 for the 4 values printed indicating that my computations are correct. Than I tested the whole code for the same parameters and got an accuracy of 45%.

- k-layer network with batch normalization is bug free: To make sure the gradient was correct, since we didn't get a compute gradient slow for batch normalization, I had to debug the whole code on jupyter notebook until I got a computation without error and an accuracy of 52% for 3 layers and [50,50] nodes.

Thus, I think my implementation is bug free.

2 Loss function for 3-layers network

- Without batch normalization, I get an accuracy of 53% for default parameter setting and hidden layers of 50 nodes each:

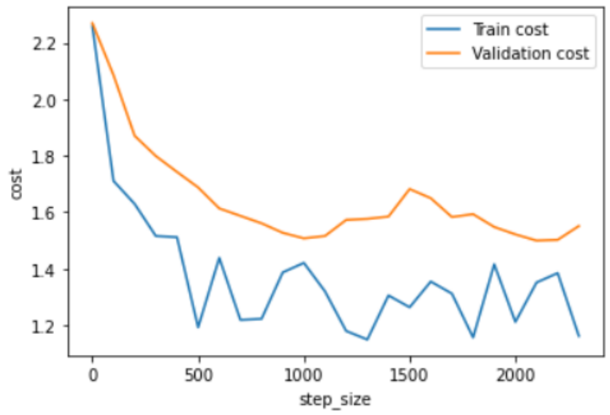
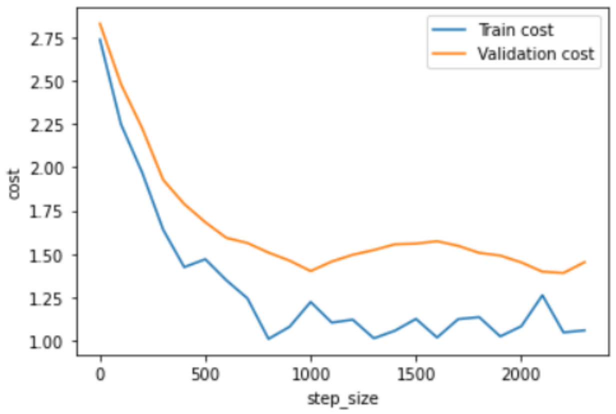


Figure 1: Evolution of the loss function when you train the 3-layer network

- With batch normalization: I get an accuracy of 54% for 3 layers with 50,50 nodes and eta min = 1e-4, eta max = 1e-1, lambda=.005, two cycles of training and n s = 5 * 45,000 / n batch.



```
0.543
```

Figure 2: Evolution of the loss function when you train the 3 layers network with BN

3 Loss function for 9-layers network

- Without batch normalization, I get an accuracy of 46% for default parameter setting and hidden layers of [50, 30, 20, 20, 10, 10, 10, 10] nodes:

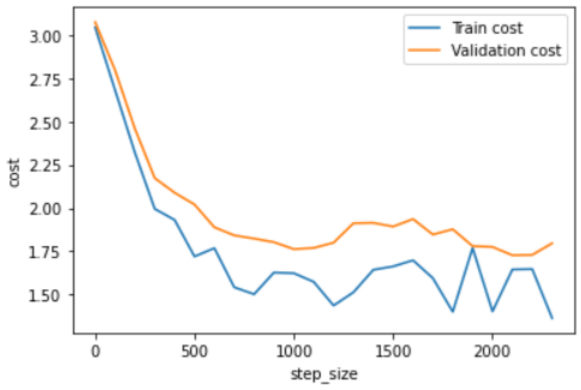
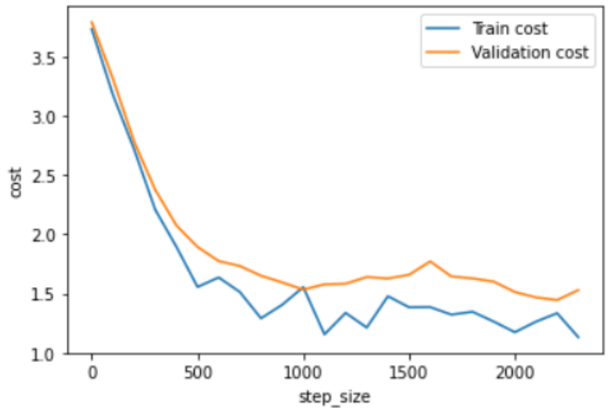


Figure 3: Evolution of the loss function when you train the 9-layer network

- With batch normalization, I get an accuracy of 50% for default parameter setting and with a batch normalization, I spent a whole week trying to figure out why the accuracy was this low from monitoring the gradient to printing every step in the network, and the TAs couldn't help me too to raise my accuracy:



```
0.5
```

Figure 4: Evolution of the loss function when you train the 9-layer network with batch normalization

4 Coarse search of lambda

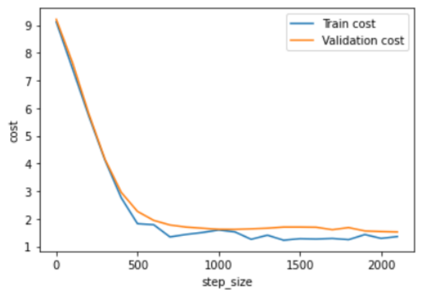
For the coarse search of lambda I did a linear search for $l_{min}, l_{max} = -5, -1$:

```
l = l_min + (l_max - l_min)*np.random.rand(1, 1)
lbd = (10 ** l) [0][0]
```

Than I choose values around this value according to a normal distribution with 0.01 around the lambda. The best lambda was 0.00507 with an accuracy of 55%.

5 Sensitivity to initialization

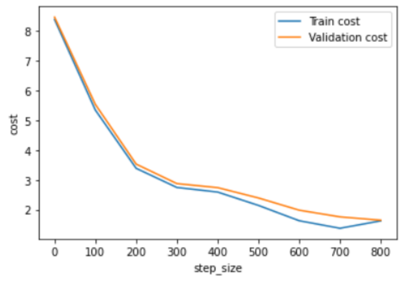
- Sigma = 1e-1:
  - Without batch normalization:



```
0.5372
```

Figure 5: Loss plot for 3-layer without batch normalization and sigma = 1e-1

  - With batch normalization:



```
0.518
```

Figure 6: Loss plot for 3-layer with batch normalization and sigma = 1e-1

- Sigma = 1e-3:
  - Without batch normalization:

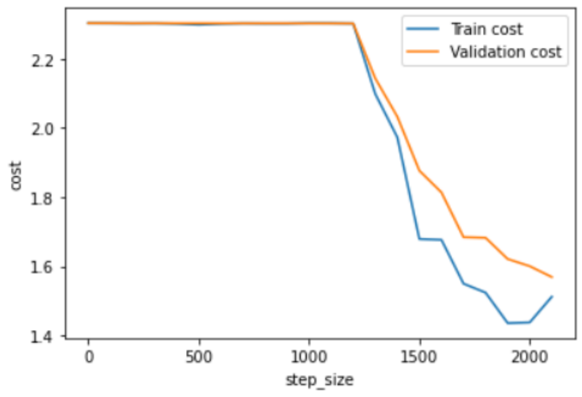
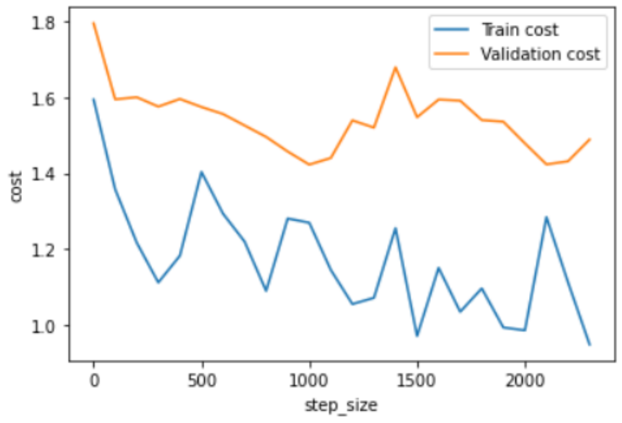


Figure 7: Loss plot for 3-layer without batch normalization and sigma = 1e-3

  - With batch normalization:



```
0.511
```

Figure 8: Loss plot for 3-layer with batch normalization and sigma = 1e-3

- Sigma = 1e-4:
  - Without batch normalization:

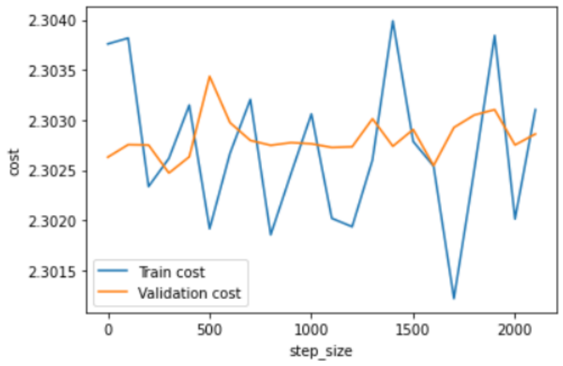
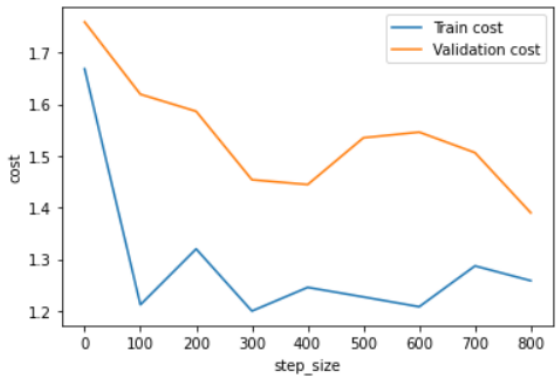


Figure 9: Loss plot for 3-layer without batch normalization and sigma = 1e-3

  - With batch normalization:



```
: 0.522
```

Figure 10: Loss plot for 3-layer with batch normalization and sigma = 1e-3