Deep Learning in Data Science

Assignment 1

Isak Gamnes Sneltvedt

April 4th, 2022

DD2424

Assignment 1

Gradient comparison

For the gradient comparison I checked the absolute differences between the numerical and analytical computed vectors. The function I wrote checks if the value is less than 1^{-7} which I chose after reading the Standford's course Convolutional Neural Networks for Visual Recognition (https://cs231n.github.io/neural-networks-3/gradcheck). There it says that if the absolute difference is less than 1^{-7}, you can be happy.

Loss and Cost function plots

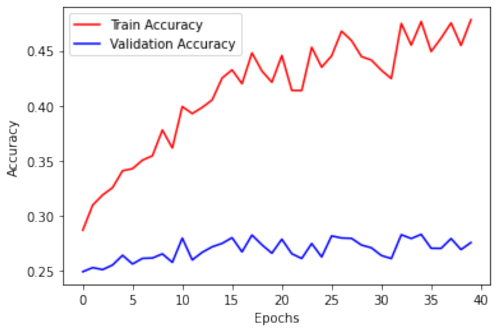

Figure 1: Showing accuracy with lambda = 0, eta = 0.1

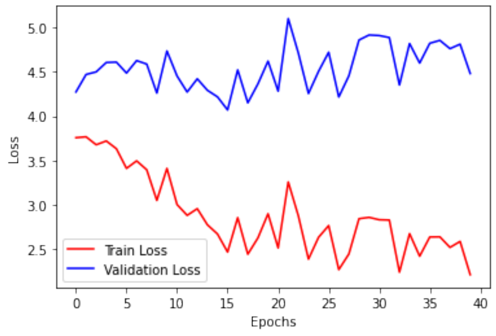

Figure 2: Showing loss with lambda = 0, eta = 0.1

In these plots we are able to see the difference between training with a high and low learning rate. When we have a eta = .1 we can see that the accuracy (figure 1) "jumps" back and forth. This is due to the high learning rate. The eta makes it so that the weight and bias are adjusted by quite a lot in each update. Compared to the plot with eta = 0.001, we can

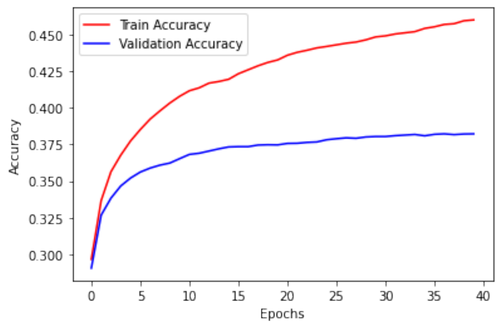

Figure 3: Showing accuracy with lambda = 0, eta = 0.001

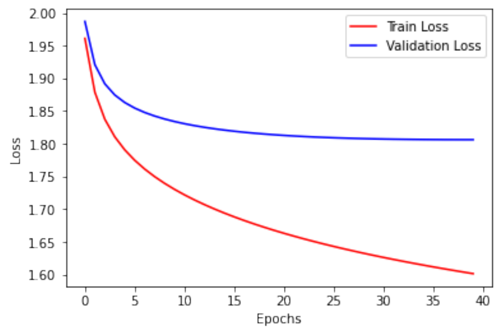

Figure 4: Showing loss with lambda = 0, eta = 0.001

see that the accuracy (figure 3) is way smoother. However, the performance in both cases is way better for training than for the validation. This is a classic example of overfitting. The overfitting can be avoided using regularization, which we do not do in this case due to the lambda value being set to 0.

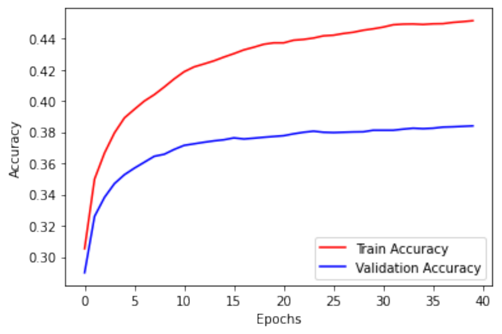

Figure 5: Showing accuracy with lambda = 0.1, eta = 0.001

This show how the performance has increased after adding the regularization. We can see that the loss is low for both the test and validation data. There is still a small gap between training and validation accuracy, but it is greatly reduced compared to the ones seen in

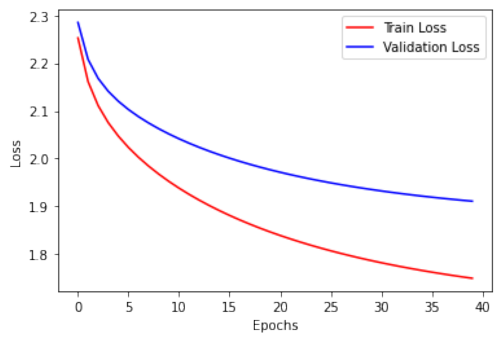

Figure 6: Showing loss with lambda = 0.1, eta = 0.001

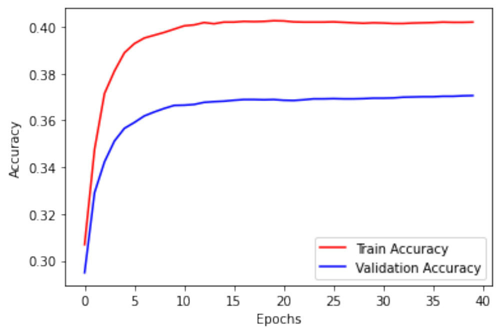

Figure 7: Showing accuracy with lambda = 1, eta = 0.001

figure 1 and figure 3

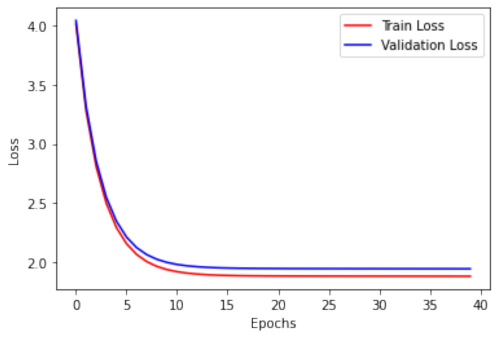

Figure 8: Showing loss with lambda = 1, eta = 0.001

Weight montages

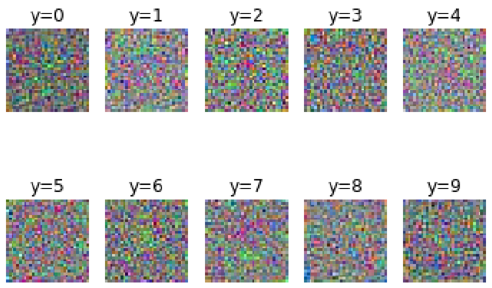

Figure 9: Showing weight with lambda = 0, eta = 0.1

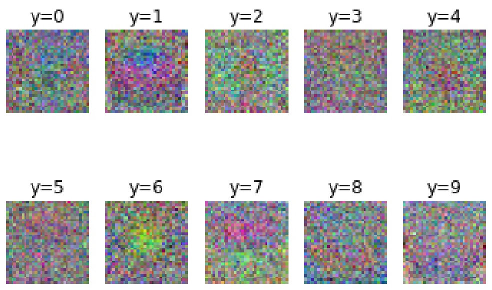

Figure 10: Showing weight with lambda = 0, eta = 0.001

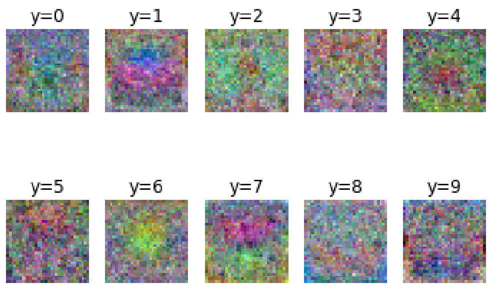

Figure 11: Showing weight with lambda = 0.1, eta = 0.001

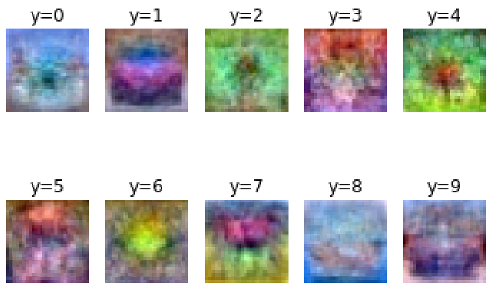

Figure 12: Showing weight with lambda = 1, eta = 0.001

Final test accuracy

After the testing I got these accuracy values:

Lambda	n_epochs	n_batch	Eta	Accuracy
0	40	100	0.1	27.09%
0	40	100	0.001	26.75%
0.1	40	100	0.001	31.23%
1	40	100	0.001	37.45%

Bonus points 1

A

All the data sets were added by reading from all the different files and concatenating the numpy arrays into a single one. The slipping was also done with 49000 samples in the training set and 1000 samples in the validation set.

```python
datasets = ['../Datasets/cifar-10-batches-py/data_batch_1',
                '../Datasets/cifar-10-batches-py/data_batch_2',
                '../Datasets/cifar-10-batches-py/data_batch_3',
                '../Datasets/cifar-10-batches-py/data_batch_4',
                '../Datasets/cifar-10-batches-py/data_batch_5'
                ]

train_x1, train_Y1, train_encoded_Y1 = load_batch(datasets[0])
    train_x2, train_Y2, train_encoded_Y2 = load_batch(datasets[1])
    train_x3, train_Y3, train_encoded_Y3 = load_batch(datasets[2])
    train_x4, train_Y4, train_encoded_Y4 = load_batch(datasets[3])
    train_x5, train_Y5, train_encoded_Y5 = load_batch(datasets[4])

    train_x = np.concatenate((train_x1, train_x2, train_x3, train_x4, train_x5[:,:9000]), axis=1)
    train_Y = np.concatenate((train_Y1, train_Y2, train_Y3, train_Y4, train_Y5[:9000]), axis=0)
    train_encoded_Y = np.concatenate((train_encoded_Y1, train_encoded_Y2, train_encoded_Y3, train_encoded_Y4, train_encoded_Y5[:,:9000]), axis=1)

    validation_x = train_x5[:,9000:]
    validation_Y = train_Y5[9000:]
    validation_encoded_Y = train_encoded_Y5[:,9000:]
```

B

The flipping was added by giving each batch a 50% chance of being flipped or not. The flip is not reverted after, as a new flip in a new epoch will only flip it back it the original image.

The code used to perform the flip (Line 4-6):

```python
1  for batch in range(0, total_length, n_batch):
2      max_batch_idx = batch+n_batch
3      X_batch = X[:,batch:max_batch_idx]
4      if allow_flipping:
5          if randint(0,1) > .5:
6              X_batch = np.flip(X_batch, axis=0)
7      grad_W, grad_b = compute_gradients( X_batch,
8                                          Y[:,batch:max_batch_idx],
9                                          W,
10                                         b,
11                                         lambda_val)
```

C

For the grid search, I went through multiple different values for the hyper parameters and compared the resulting accuracy. The hyper parameters I went through are:

Lambda	n_batch	Eta
0.1	10	0.1
1.0	100	0.001
2.0	200	-

The grid search was performed with flipping enabled, step decay and all 50.000 data samples. The results were quite similar in performance, with only a few percent separating them. The one that got the best result was with lambda = 1.0, eta = 0.1 and n_batch = 10.

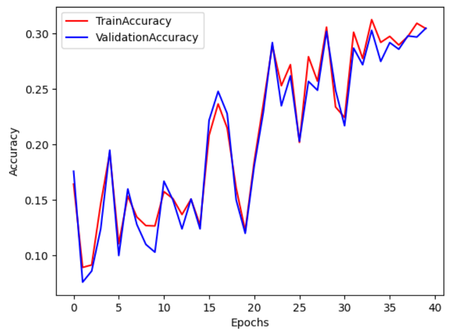

Figure 13: Showing best result from grid search with lambda=1.0, eta=0.1, $n_batch = 10$

As seen in figure 13, the results got worse after adding the flipping and the step decay. Therefore, I decided to train the model again using the hyper parameters found in the grid search and see the performance without these. The results from this test was that the flipping made the performance worse. These are the results from running without flipping, with step decay and the hyper parameters found in the grid search:

The accuracy increase looks a bit weird, so I would like to get a short feedback whether this is right or wrong.

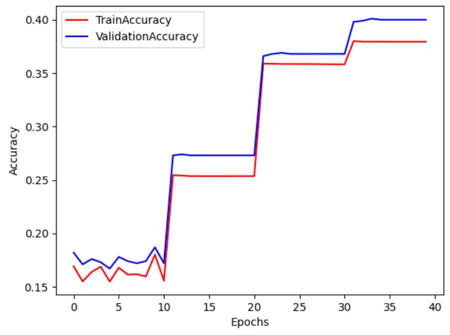

Figure 14: Showing results of hyper parameters without flipping

D

Weight decay was implemented by using the modulo operator.

The code used to perform weight decay:

```
if epoch%10 == 0 and epoch!=0:
    eta = .1*eta
    tqdm.write('Weight decay performed.. eta is now {}'.format(eta))
```

Terminal output:

```
Weight decay performed.. eta is now 0.0001
Weight decay performed.. eta is now 1e-05
Weight decay performed.. eta is now 1e-06
90%|--------------------------->      | 36/40 [01:54<00:12,  3.19s/it]
```