Deep Learning in Data Science: Training a multi-linear classifier

Alexander Bea - abea@kth.se

Assignment 1, Exercise 1

In this assignment I had to train and test a one layer network with multiple outputs to classify images from the CIFAR-10 dataset. I trained the network using mini-batch gradient descent applied to a cost function that computes cross-entropy loss of the classifier applied to the labelled training data and an L2 regularization term on the weight matrix.

[1] Installers

[2] Import libraries

[3] Functions: Decoding and displaying images

EXERCISE 1. PART 1.

Read in and store the training, validation and test data

[4] Code: Load training-, validation- and test- data

EXERCISE 1. PART 2.

Transform training data to have zero mean

[5] Functions: Normalize data

[6] Code: Normalize data

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
X_train mean: 1.7869613196571282e-15
X_val mean: -0.0008092555147040919
X_test mean: 0.0019141973039233557
```

[6]

EXERCISE 1. PART 3.

Initialize parameters of the model W and b with entry to have Gaussian random values (incl. zero mean and standard deviation of .01)

```
[7] mean = 0.0
    s = 0.01
    d = X_train.shape[1]
    K = Y_train.shape[1]
    W = np.random.normal(mean, s, (K, d)) # Weight matrix; Normal (Gaussian) distribution
    b = np.random.normal(mean, s, (K, 1)) # Bias vector; Normal (Gaussian) distribution
```

EXERCISE 1. PART 4.

Function that evaluates the network function

[8] Functions: EvaluateClassifier and Softmax

```
[9] P = EvaluateClassifier(X_train[:100], W, b) #Check subset
    np.sum(P, axis = 0) # Check if the sums for each sample sum up to 1
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

EXERCISE 1. PART 5.

Function that computes the cost function

[10] Functions: Compute Cost and Cross Entropy Loss

```
[11] J = ComputeCost(X_train, Y_train, W, b, lamda = 0)
     print("Loss from Cost Function: " + str(J))
```

```
Loss from Cost Function: 2.32091356491174
```

EXERCISE 1. PART 6.

Function that computes the accuracy

[12] Functions: Compute Accuracy

```
[13] acc = ComputeAccuracy(X_train, y_train, W, b)
     print("Check accuracy: " + str(acc))
```

```
Check accuracy: 0.0898
```

EXERCISE 1. PART 7.

Function that evaluates, for a mini-batch, the gradients, of the cost function w.r.t. W and b

[14] ## Functions: Compute gradients and display differences between methods

Analytical (ANL) gradient computation is in the following result compared to the slow but accurate version based on the centered difference equation (CDM) and compared to the faster but less accurate finite difference method (FDM). The accuracy can be observed in the observed in the below tables which displays relative and absolute differences between the aformentioned methods. Note that absolute differences are less than 1e-6 and thereby considered to have produced the same result.

[15]
```
compareGradients(lamda=0.0, title="Without Regularization i.e. Lambda = 0.0")
```

Without Regularization i.e. Lambda = 0.0

Gradient	Method	Rel Diff Min [e+04]	Rel Diff Max [e+04]	Rel Diff Mean [e+04]	Abs Diff Max [e+06]	Abs Diff Mean [e+06]
W	ANL vs CDM	0.000	547.345	0.104	0.081	0.016
W	ANL vs FDM	0.000	1468.733	0.317	0.158	0.046
W	CDM vs FDM	0	2000	0.315	0.111	0.044
b	ANL vs CDM	0.000	0.016	0.004	0.039	0.015
b	ANL vs FDM	0.002	0.040	0.015	0.106	0.062
b	CDM vs FDM	0.002	0.034	0.011	0.089	0.051

[16]
```
compareGradients(lamda=1.0, title="With Regularization i.e. Lambda = 1.0")
```

With Regularization i.e. Lambda = 1.0

Gradient	Method	Rel Diff Min [e+04]	Rel Diff Max [e+04]	Rel Diff Mean [e+04]	Abs Diff Max [e+06]	Abs Diff Mean [e+06]
W	ANL vs CDM	0.000	85.277	0.051	0.125	0.027
W	ANL vs FDM	0.000	185.524	0.078	0.178	0.039
W	CDM vs FDM	0	204.082	0.047	0.089	0.021
b	ANL vs CDM	0.001	0.026	0.006	0.054	0.022
b	ANL vs FDM	0.001	0.037	0.009	0.080	0.031
b	CDM vs FDM	0	0.021	0.003	0.044	0.009

EXERCISE 1. PART 8.

Function that performs the mini-batch gradient descent algorithm to learn the network's parameters

As the below result shows, after the first epoch the cost score decreases and the accuracy increases for each epoch.

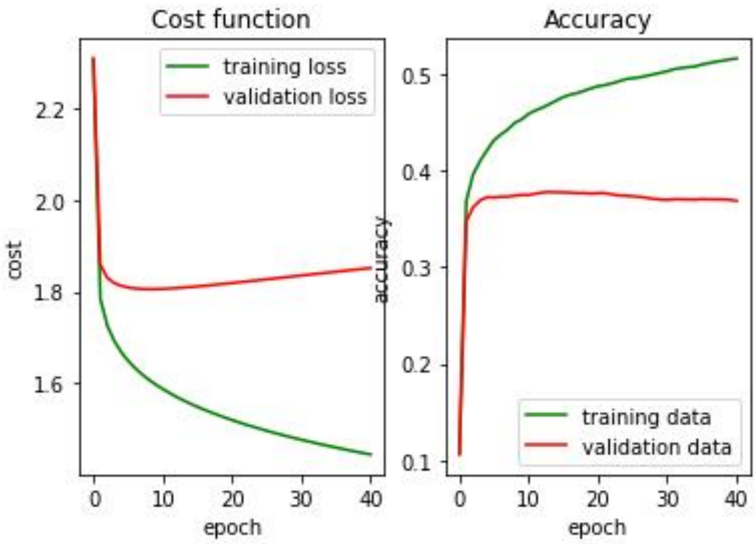
Learning rate: We can also tell from the same result, that when the learning rate (eta) is too large, the training of the model becomes unstable. This can be observed in the first figure where eta=0.1

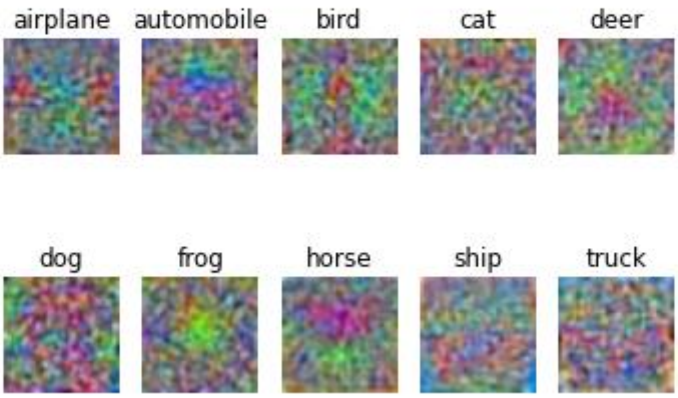
Regularization: The effect on accuracy when applying regularization is that it is narrower between the training data and validation data in difference to when not applying it. However, without regularization the accuracy is higher. Ideal is it not to have it too wide as this can be an indication of overfitting on the training data.

[17] Function: Mini-batch gradient descent

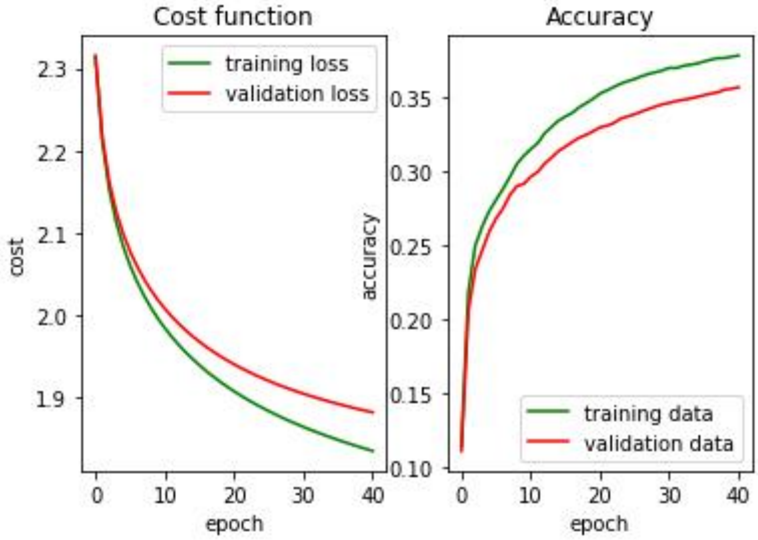
[18] Code: Run mini-batch gradient descent with difference parameters

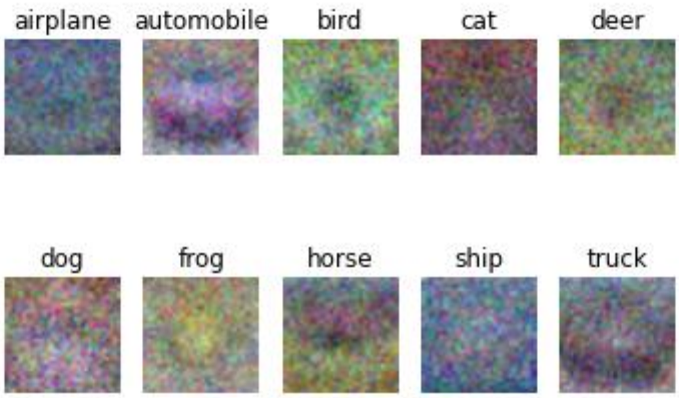
```
****************************************
lambda=0, n epochs=40, n batch=100, eta=0.1
****************************************
```



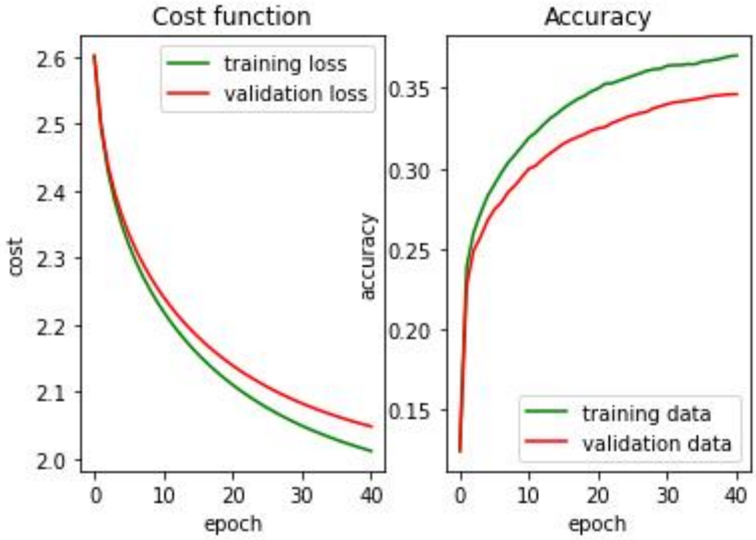


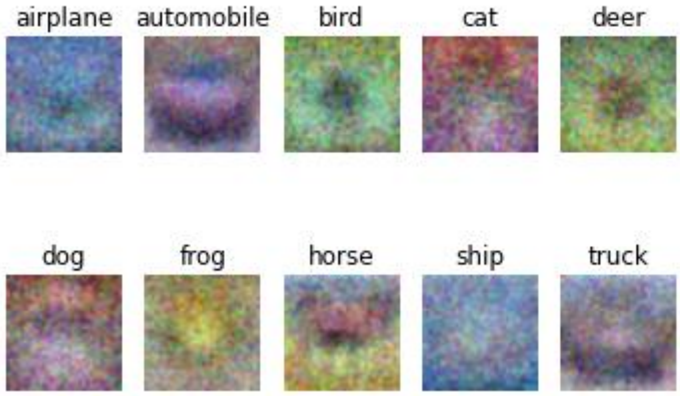
```
****************************************
lambda=0, n epochs=40, n batch=100, eta=0.001
****************************************
```



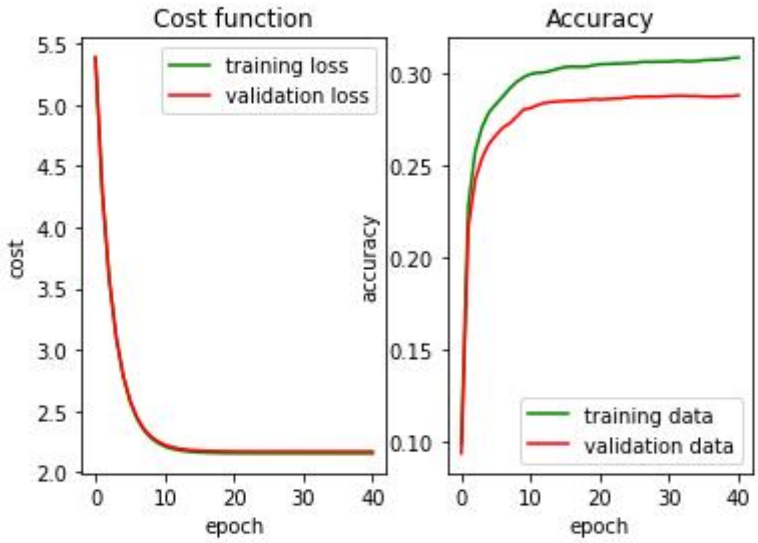


```
****************************************
lambda=0.1, n epochs=40, n batch=100, eta=0.001
****************************************
```





```
*****************************************
lambda=1, n epochs=40, n batch=100, eta=0.001
*****************************************
```



Parameters	Train Accuracy	Val Accuracy	Test Accuracy
lambda=0, n epochs=40, n batch=100, eta=0.1	0.516	0.368	0.372
lambda=0, n epochs=40, n batch=100, eta=0.001	0.378	0.356	0.362
lambda=0.1, n epochs=40, n batch=100, eta=0.001	0.370	0.346	0.355
lambda=1, n epochs=40, n batch=100, eta=0.001	0.309	0.288	0.307