Deep Learning in Data Science: **Training a multi-linear classifier with a two layer network**

Alexander Bea - abea@kth.se

Assignment 2

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

[1] ## Installers

[2] ## Import libraries

[3] ## Functions: Decoding and displaying images

EXERCISE 1

Read in the data & initialize the parameters of the network

For this exercise, I will just use the data in the file data_batch_1 for training, the file data_batch_2 for validation and the file test_batch for testing. Also, transform it to have zero mean and normalize it.

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

[5] ## Functions: Initialize network parameters

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

[6] ## Functions: Evaluate Classifier, Cost and Accuracy

In the following the network parameters are intialized wherein M=50 (number of hidden layers), bias vectors are set to zero and entries in the weight matrices are random draws from a Gaussion ditribution with mean 0 and standard deviation of 1/sqrt(d) for layer 1 and 1/sqrt(m) for layer 2.

[7]
```
data, labels = trainOnSmallDataBatch() #load data
clf = NetClassifier(data, labels) #initialize network parameters
```

EXERCISE 2

Compute the gradients for the network parameters

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

[8] Functions: Compute & Compare Gradients Methods

The following generates the gradient comparing result that shows that the implemented analytical gradient method is close enough to be regarded as accurate.

[9] Compare Analytical with Numerical Gradient

```
Method Comparison: Analytical vs Numerical
+----------+------------+----------------------+
| Gradient |   Method   | Abs Diff Mean [e-06] |
+==========+============+======================+
| W1       | ANL vs NUM | 0.138                |
+----------+------------+----------------------+
| b1       | ANL vs NUM | 0.002                |
+----------+------------+----------------------+
| W2       | ANL vs NUM | 1.337                |
+----------+------------+----------------------+
| b2       | ANL vs NUM | 0.232                |
+----------+------------+----------------------+
```

EXERCISE 3

Train the network with cyclical learning rates

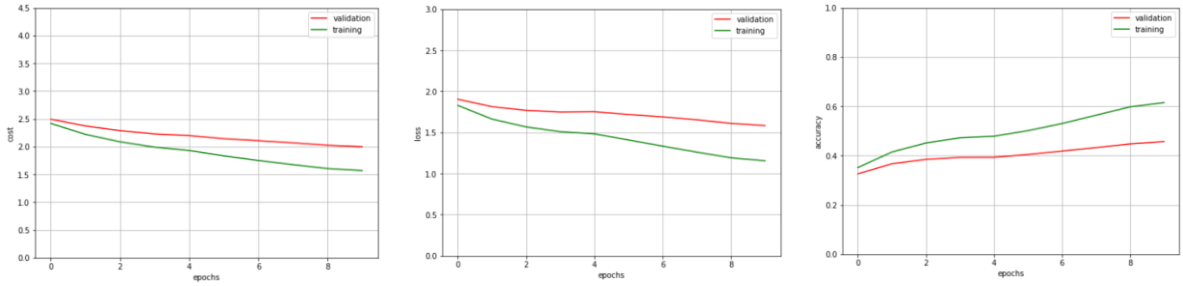
To avoid time-consuming searches for good values of eta, I have implemented mini-batch gradient descent training were the learning rate at each update step is pre-defined and eta_min is set to 1e-5, eta_max to 1e-1 and n_s=500 and batch size to 100. t should be run from 1 to 2*n_s and lambda=0.01.

The figure that the following training produces corresponds shows the cost, loss and accuracy for one cycle training. Accuracy achieved are displayed in the following table.

[10] Functions: Compute Mini Batch Gradient Descent

[11] Code: Trained network with One Cycle of Training Data

```
******************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.615          | 0.457        | 0.456         |
+----------------+--------------+---------------+
```

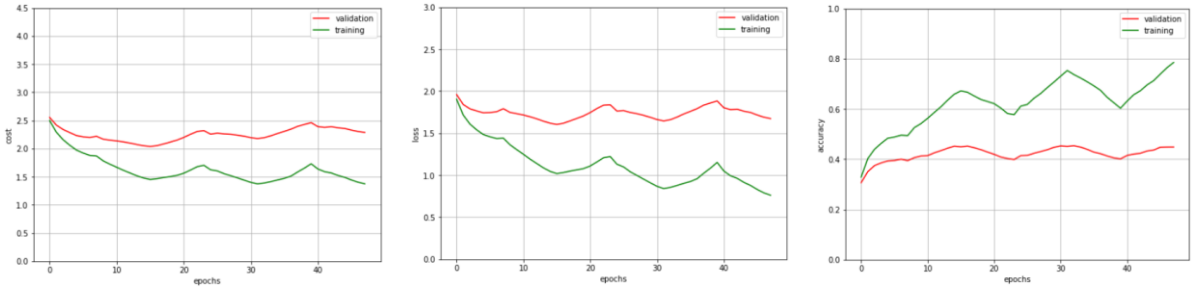


EXERCISE 4

Now, I will run with more cycles and larger n_s=800. In the following plots, it becomes clear that the loss and accuracy vary as the eta_t varies. Accuracy achieved are displayed in the following table.

[12] Code: Trained network with Three Cycles of Training Data

```
******************************************
+----------------+--------------+---------------+
| Train Accuracy | Val Accuracy | Test Accuracy |
+================+==============+===============+
| 0.785          | 0.449        | 0.445         |
+----------------+--------------+---------------+
```



Next, I need to optimize the value of regularization term lambda. This is first done by performing a coarse random search and then a fine random search to set lambda.

Coarse search

This search is performed by running a random search through training my network on a random initalization and measuring the performance (via accuracy on the validation set) multiple times as the hyper-parameter lambda randomly varies. I will also extend the data to include the other batches and with more training data I expect to need a lower lambda. Thereby, the randomly selected lambdas will be close to zero. Moreover, 5,000 images will be used as my validation set. Training is with n_s=2*floor(n/n_batch) i.e.* 2floor(45,000/100)=900. Also I set the learning rate to eta_min=1e-5, eta_max=1e-1.

[13] Functions: Load All Data

[14] Functions: Train the Classifier on All Data

In the following coarse search I have randomly selected five very low regularzation terms. In the below resulting table it becomes clear that the validation accuracy is highest where lambda is below 0.01, thereby the fine search will be between this interval.

[15] Code: Coarse Search

```
Lambda test done:0.001
Lambda test done:0.005
Lambda test done:0.01
Lambda test done:0.016
Lambda test done:0.02
********** COARSE SEARCH **********
n epochs=8, batch_s=100, n_s=900.0
************************************************************
```

Parameters	Train Accuracy	Val Accuracy	Test Accuracy
lambda=0.001	0.590	0.521	0.508
lambda=0.005	0.574	0.522	0.513
lambda=0.01	0.556	0.514	0.513
lambda=0.016	0.536	0.506	0.511
lambda=0.02	0.524	0.501	0.506

The best fine search that yielded the highest accuracy on the validation set was at lambda=0.006 as is displayed in the table below. For future work, I would explore running a much wider random search.

[16] Code: Fine Search

```
Lambda test done:0.0055
Lambda test done:0.006
Lambda test done:0.0065
Lambda test done:0.007
Lambda test done:0.0084
Lambda test done:0.0089
Lambda test done:0.0093
********** FINE SEARCH **********
n epochs=12, batch_s=100, n_s=900.0
************************************************************
```

Parameters	Train Accuracy	Val Accuracy	Test Accuracy
lambda=0.0055	0.588	0.531	0.519
lambda=0.006	0.585	0.537	0.523
lambda=0.0065	0.583	0.525	0.521
lambda=0.007	0.578	0.527	0.516
lambda=0.0084	0.571	0.524	0.521
lambda=0.0089	0.567	0.525	0.517
lambda=0.0093	0.570	0.526	0.519

The following is the best found lambda setting lambda=0.006, whereby all training data is used except for 1,000 example images in a validation set for 3 cycles. The hyper-parameters used are lambda=0.006, batch_s=100, n_s=980 [2*np.floor((50000-1000)/batch_s)] and n_epochs equivalent to 3 cycles. Final accuracies are reported in the table below.

[17] Code: Best Classifier

```
Lambda test done:0.006
******************** BEST CLASSIFIER ************************
n epochs=12, batch_s=100, n_s=980.0
************************************************************
+--------------+----------------+--------------+---------------+
|  Parameters  | Train Accuracy | Val Accuracy | Test Accuracy |
+==============+================+==============+===============+
| lambda=0.006 | 0.584          | 0.540        | 0.521         |
+--------------+----------------+--------------+---------------+
```