Deep Learning in Data Science, DD2424

Short report on lab assignment 1

Back-propagation

Anna Sánchez Espunyes

March 26, 2021

1 Main objectives and scope of the assignment

In this assignment you will train and test a one layer network with multi-ple outputs to classify images from the CIFAR-10 dataset. You will train the network 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 an L2 regularization term on the weight matrix.

2 Methods

In this assignment we worked using Python and some of its well-known libraries, including matplotlib, numpy. We have also used the PyCharm IDE and Github.

3 Results and discussion

3.1 Gradient calculation

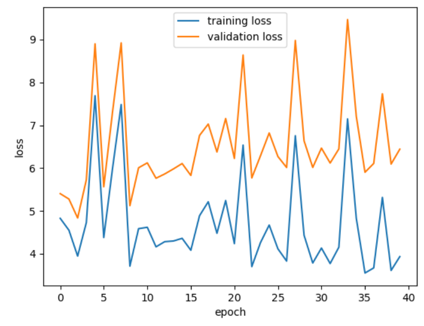
Gradient calculation is a key step when updating the weights. In the assignment the gradients are calculated analytically but an extra numerical function is provided to check if the calculations are precise. After checking whether the calculations are correct, it can be stated that the computations are accurate. I have compared the numerically and analytically computed gradient by examining their absolute differences. The absolute differences is in the order of 1e-7, which is a threshold small enough to consider the calculations valid.

3.2 Exercise 1: Training a multi-linear classifier

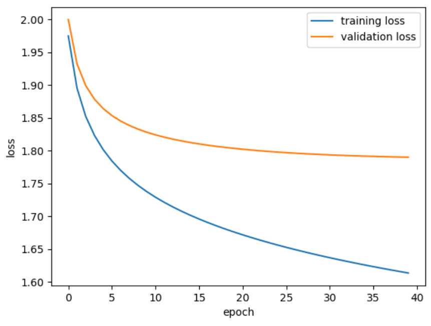
A multi-linear classifier has been trained for different parameter settings. For each setting, it can be seen the graphs of the cost function on the training data and the validation data after each epoch of the mini-batch gradient descent algorithm (Figure 1) and learnt weight matrix after the completion of training (Figure 2). It is worth noticing that in Figure 1 the loss axis is different depending on the settings.

The final test accuracy for each setting is the following:

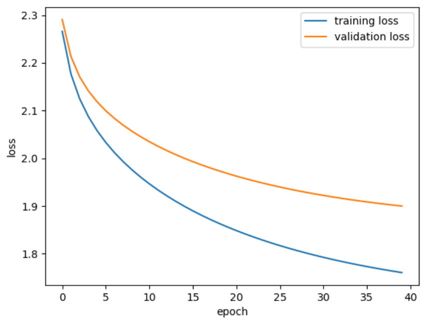
- lambda=0, n epochs=40, n batch=100, eta=.1. Test error: 29.71



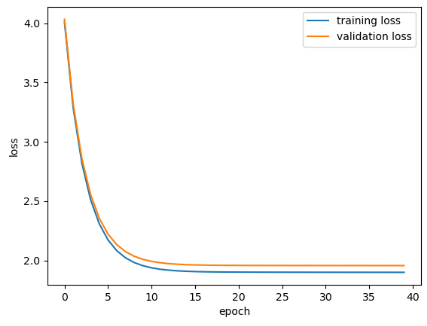
(a) (lambda=0, n epochs=40, n batch=100, eta=.1)



(b) (lambda=0, n epochs=40, n batch=100, eta=.001



(c) lambda=.1, n epochs=40, n batch=100, eta=.001



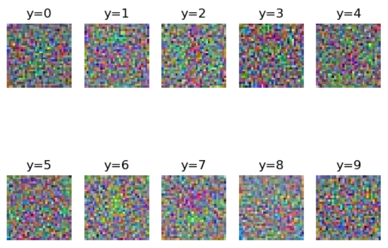
(d) lambda=1, n epochs=40, n batch=100, eta=.001

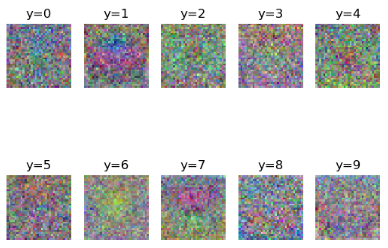
Figure 1: Loss for training and validation set in every epoch for different hyper-parameters

- lambda=0, n epochs=40, n batch=100, eta=.001. Test error: 38.88
- lambda=.1, n epochs=40, n batch=100, eta=.001. Test error: 39.02
- lambda=1, n epochs=40, n batch=100, eta=.001. Test error: 37.49

From these results, meaningful insights can be extracted. The first one is the importance of choosing an appropriate learning rate. The learning rate indicates the size of the step towards the minimum of the function. If the learning rate is too big, we can miss the minimum and never reach convergence. This can be studied comparing Figure 1a and Figure 1b. For the same parameter settings and only changing the learning rate, we get two different situations: in Figure 1a the algorithm does not converge and in Figure 1b we converge to good results. On the other hand, if the learning rate is too small we may not get to the minimum of the function.

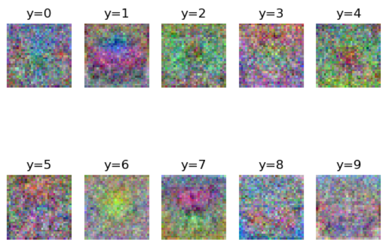
The second insight is the importance of regularization. Regularization prevents us from over-fitting (a situation where the model is too adapted to the training data and does not generalize well for unseen data). By adding a penalty term, the weights do not reach very high values. In Figure 1c and Figure **??** we can compare the effect of increasing the penalty term. In this case we get worse results, which means that the model may be in an under-fitting situation. It is important to use regularization to avoid over-fitting but not choose to high values of regularization to avoid under-fitting.

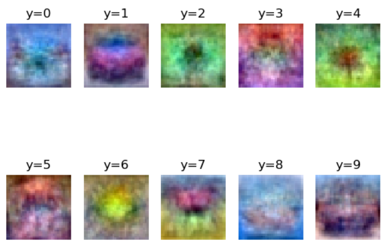




(a) (lambda=0, n epochs=40, n batch=100, eta=.1)

(b) (lambda=0, n epochs=40, n batch=100, eta=.001





(c) lambda=.1, n epochs=40, n batch=100, eta=.001

(d) lambda=1, n epochs=40, n batch=100, eta=.001

Figure 2: Weight matrix representation for different hyper-parameters