ASSIGNMENT 2

Diego Argüello Ron

Deep Learning in Data Science (DD2424)

18 April 2018

1 Introduction

The objective of this assignment is to improve the results obtained in the previous one by adding a hidden layer to the network.

2 Checking Gradients

Like in the first assignment, making sure that the gradients are being computed correctly is of the utmost importance. Thus, the analitically calculated gradients were compared with the ones calculated numerically by obtaining the relative error between them. Being g_n the value of the numerically computed gradient and g_a the value of the analytically computed one, the relative error will be:

$$\frac{|g_a - g_n|}{max(esp, |g_a| + |g_n|)} \tag{1}$$

where esp is a very small positive number, taken as 0.
The relative error was calculated for mini-batches of size 1, 10, 50, and 100 with no regularization.
The next results were obtained:

Batch size	ϵ_{b1}	ϵ_{b2}	ϵ_{W1}	ϵ_{W2}
1	$6.1139 \cdot 10^{-8}$	$2.0411 \cdot 10^{-10}$	$1.3711 \cdot 10^{-7}$	$1.0261 \cdot 10^{-8}$
10	$4.38 \cdot 10^{-7}$	$2.616 \cdot 10^{-9}$	$8.1477 \cdot 10^{-7}$	$5.4621 \cdot 10^{-8}$
50	$1.2746 \cdot 10^{-6}$	$3.2844 \cdot 10^{-9}$	$2.3531 \cdot 10^{-6}$	$1.5545 \cdot 10^{-7}$
100	$3.1353 \cdot 10^{-6}$	$7.3062 \cdot 10^{-9}$	$5.155 \cdot 10^{-6}$	$3.1555 \cdot 10^{-7}$

As it can be seen in the table, the gradients are being calculated properly. It is worth noticing how the error increases with the cercany to the input layer.

On the other hand,another sanity check has been successfully carried out. In this case, if the network is able to overfit to the training data and get a very low loss on it has been checked. Thus, the network has been trained on 100 examples and 200 epochs, with a learning rate of 0.1. This can be seen implemented in the code: SanityCheck.m, being the results:

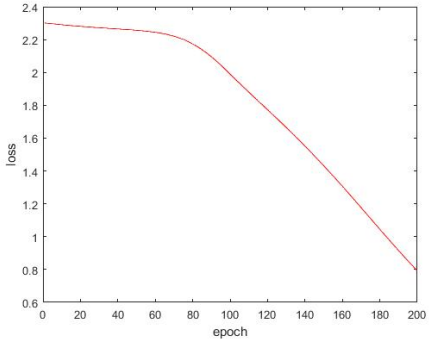

Figure 1: Loss function for training using 200 epochs and $\eta = 0.1$

3 Speeding Training adding Momentum

In order to decrease the training time, momentum terms will be added into the mini-batch update steps. This is done by changing the code corresponding to the mini-batch learning function (MiniBatchGD):

```
n_batch = GDparams.n_batch;
eta = GDparams.eta;
N = size(X, 2);  %number of images
Wv = {zeros(size(W{1})), zeros(size(W{2}))};
bv = {zeros(size(b{1})), zeros(size(b{2}))};

for j = 1 : N/n_batch

    j_start = (j - 1)*n_batch + 1;
    j_end = j*n_batch;
    inds = j_start : j_end;
    Xbatch = X(:, inds);
    Ybatch = Y(:, inds);
    H = firstLayer(Xbatch, W, b);
    P = EvaluateClassifier(H, W, b);
    [grad_W, grad_b] = ComputeGradients(Xbatch, Ybatch, P, H, W, b, lambda)
    ;
    Wv{1} = r*Wv{1} + eta*grad_W{1};
    Wv{2} = r*Wv{2} + eta*grad_W{2};
    bv{1} = r*bv{1} + eta*grad_b{1};
    bv{2} = r*bv{2} + eta*grad_b{2};
    W{1} = W{1} - Wv{1};
    W{2} = W{2} - Wv{2};
    b{1} = b{1} - bv{1};
    b{2} = b{2} - bv{2};
end

Wout = W;
bout = b;
```

Listing 1: Momentum

Above, it can be seen how the function *MiniBatchGDold.m* has been changed (renamed as *MiniBatchGD.m*) by adding the momentum terms. Thus,it is expected that the training times will be reduced.Nevertheless, another sanity check must be carried out in order to make sure that the network works properly. Thus, it will be seen that it is able to fit a small amount of training data at the same time that the parameter ρ is changed.

On the other hand, the learning rate will be decreased by a factor known as decay rate (0.95) after each epoch of training.

As it can be seen below the convergence is faster as the value of ρ is increased and at the end, for a value of $\rho = 0.99$ the convergence has been achieved for 5 epochs.

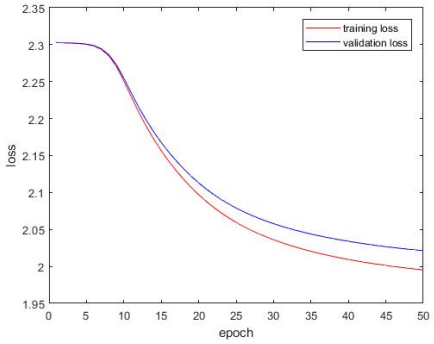

Figure 2: $\rho = 0$

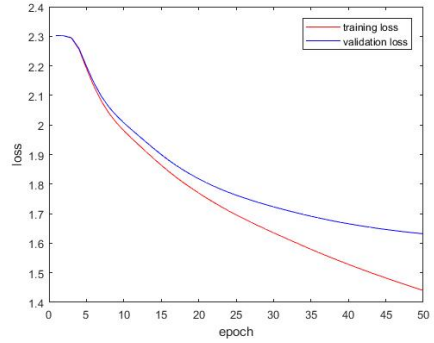

Figure 3: $\rho = 0.5$

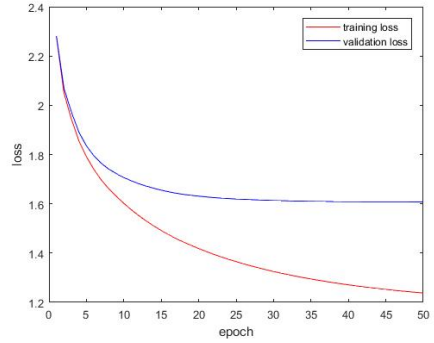

Figure 4: $\rho = 0.9$

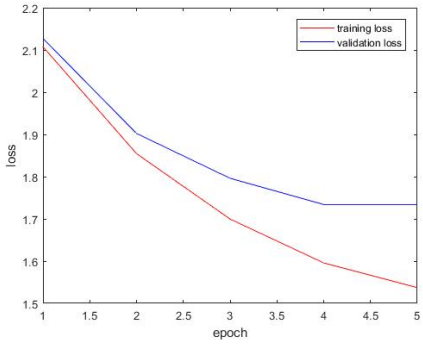

Figure 5: $\rho = 0.99$

4 Training the Network

The first step when training the network is finding an optimal set of parameters by performing a random searches. During all this process, $\rho = 0.9$.

Firstly, a rough search of the proper learning rate value was carried out by obtaining the loss function with respect to the value of η:

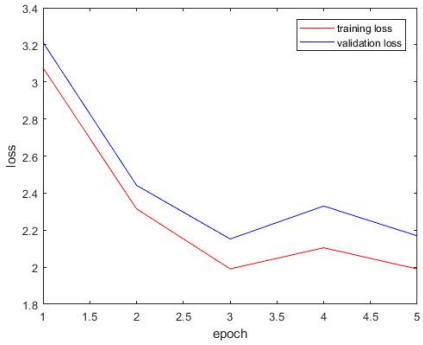

Figure 6: loss for $\eta = 0.3$

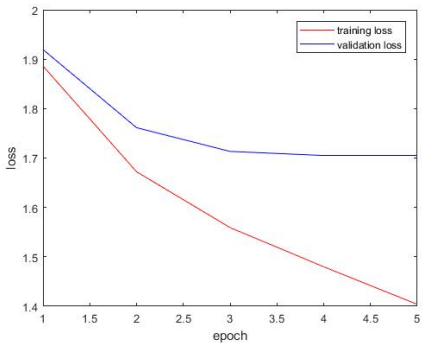

Figure 7: loss for $\eta = 0.05$

As expected, if the learning rates are small, the training loss will barely change, meanwhile for large learning rates the learning will be unstable. So, it can be concluded the value of the learning rate that must be tried is: $0.7 > \eta > 0.01$. It must be pointed out that 10/15 epochs have been used to carry out this test.

Thus, by setting bounds to the values of λ and η and selecting the best values of the accuracy for the different pairs, the boundaries are being reduced until getting an optimal pair of values.

Three intervals have been tested in oder to get the values of the hyper-parameters: $0.7 > \eta > 0.01$ and $0.1 > \lambda > 1 \cdot 10^{-6}$; $0.1 > \eta > 0.01$ and $1 \cdot 10^{-4} > \lambda > 1 \cdot 10^{-7}$; $0.06 > \eta > 0.02$ and $1 \cdot 10^{-5} > \lambda > 1 \cdot 10^{-6}$ .The pairs calculated were 75 for the first two intervals and 50 for the last one because of the high time it took to calculate the values.

The pairs were calculated as follows:

$$e = e_{min} + (e_{max} - e_{min}) * rand(1,1)\eta = 10^e \tag{2}$$

$$l = l_{min} + (l_{max} - l_{min}) * rand(1,1)\lambda = 10^l \tag{3}$$

Finally, the best accuracy obtained was 44.02 % for $\lambda = 1.8035 \cdot 10^{-6}$ and $\eta = 0.0265$. The network has been trained on all the training data, and tested 1000 examples as a validation set, for 30 epochs. Being the result:

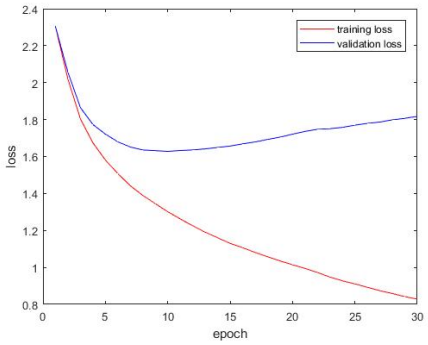

Figure 8: loss for chosen hyper-parameters

As it can be seen, the main problem of the network is that it is unable to generalize properly. It is worth noticing that a high variance is a consequence of the network having been trained too much (overfitting) or that it is too complex. Due to the fact that the accuracy is still low (both generalization and variance are poor) overfitting looks like the most probable answer to this problem. On the other hand, increasing the complexity of the network and calculating other hyper-parameters would be interesting in order to improve the generalization and variance.