Deep Learning Lab – Assingment 1

Guilherme Alessandri Alvim Miotto – 4653503

October 30, 2018

This short report describes the procedures followed during my work on the first assignment of the Deep Learning Lab course. In this assignment, we had to complete a code stub of a multilayer perceptron and check its performance on the MNIST dataset. Apart from the basic requirements of the assignment, I also implemented regularization by dropout and optimization by Adam. The following sections show the results obtained.

1 Gradient check

After filling in the code for the forward and backward passes, I checked the calculated gradients of an arbitrary network using the provided auxiliary function. The results are shown on Table 1. All deviations are below the threshold (10^{-5}). With gradients in place, I moved on to implement the optimization algorithms.

Gradient wrt	Value
Layer 1 weights	$1.83 \times 10{-}7$
Layer 1 bias	$7.70 \times 10{-}8$
Layer 2 weights	$1.49 \times 10{-}7$
Layer 2 bias	$2.17 \times 10{-}8$
Layer 3 weights	$7.95 \times 10{-}8$
Layer 3 bias	$1.59 \times 10{-}8$

Table 1: Results obtained by the gradient check

2 Optimization test

Before optimizing any hyper-parameter, I tested the training code using the provided 3-layer network and optimization setup. The solution converged and the training and validation errors were 0.3% and 2.5% respectively; a satisfactory result.

3 My network

After manually trying a couple of different network configurations, I decided to stick with a 4-layer network regularized by dropout and optimized via Adam. Network hyper-parameters are shown on Tables 2 and 3.

Hyperparameter	Value
Number of layers	4
Regularization method	Dropout
Optimization method	Adam
Number of epochs	30
Learning rate	$2.0 \times 10{-}4$

Table 2: Network general hyper-parameters

Layer	Number of Units	Dropout	Activation Function
1	600	30%	ReLU
2	500	20%	ReLU
3	400	0%	ReLU
4	10	0%	None

Table 3: Layer specific hyper-parameters

4 Results

As expected, before any training, the network has an accuracy of roughly 10%, which is the average performance of a series of random guesses. As training goes on, the loss and errors go down, as shown on Figure 1. At the end of the training, the error on the validation dataset was 1.76%.

The final result was evaluated on the test dataset. To that end, the network was re-trained on the "expanded" training dataset, i.e. training plus validation. The error rate was 1.90%. All statistical results are summarized on Table 4.

Finally, Figure 2 shows a set of misclassified digits randomly sampled from the test dataset. It is possible to notice that some of the misclassifications happen with really badly written digits. To label some of them may be a difficult task even for humans. This shows that, although the algorithm has definitely not achieved human performance, a great part of the problems it faces are those that would also affect a human.

Metric	Value
Training loss	0.0035
Training error	0.11%
Validation error	1.76%
Training time	13 min
Test error	**1.90**%

Table 4: Result statistics

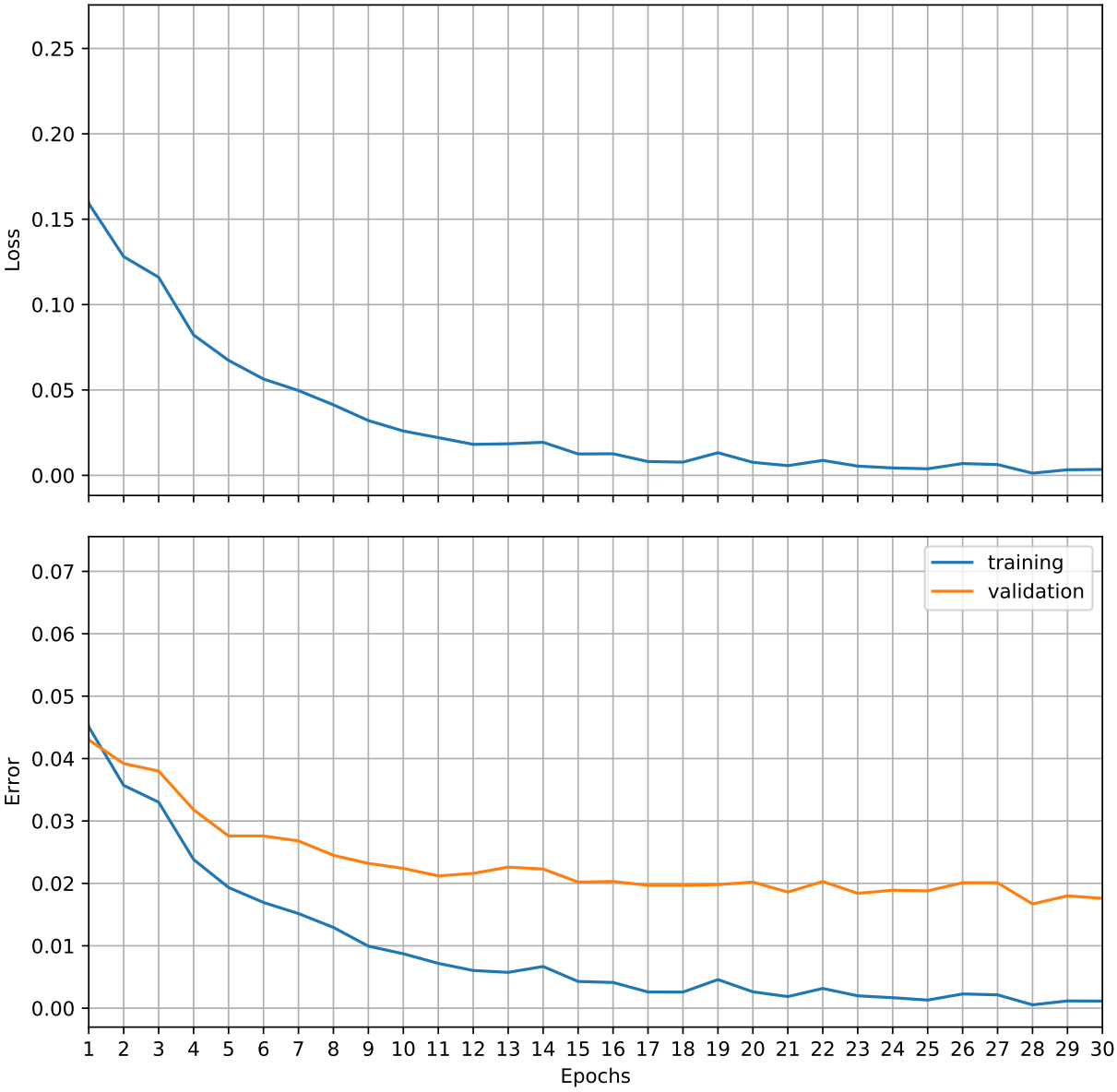

Figure 1: Loss, training error and validation error evolution along the epochs.

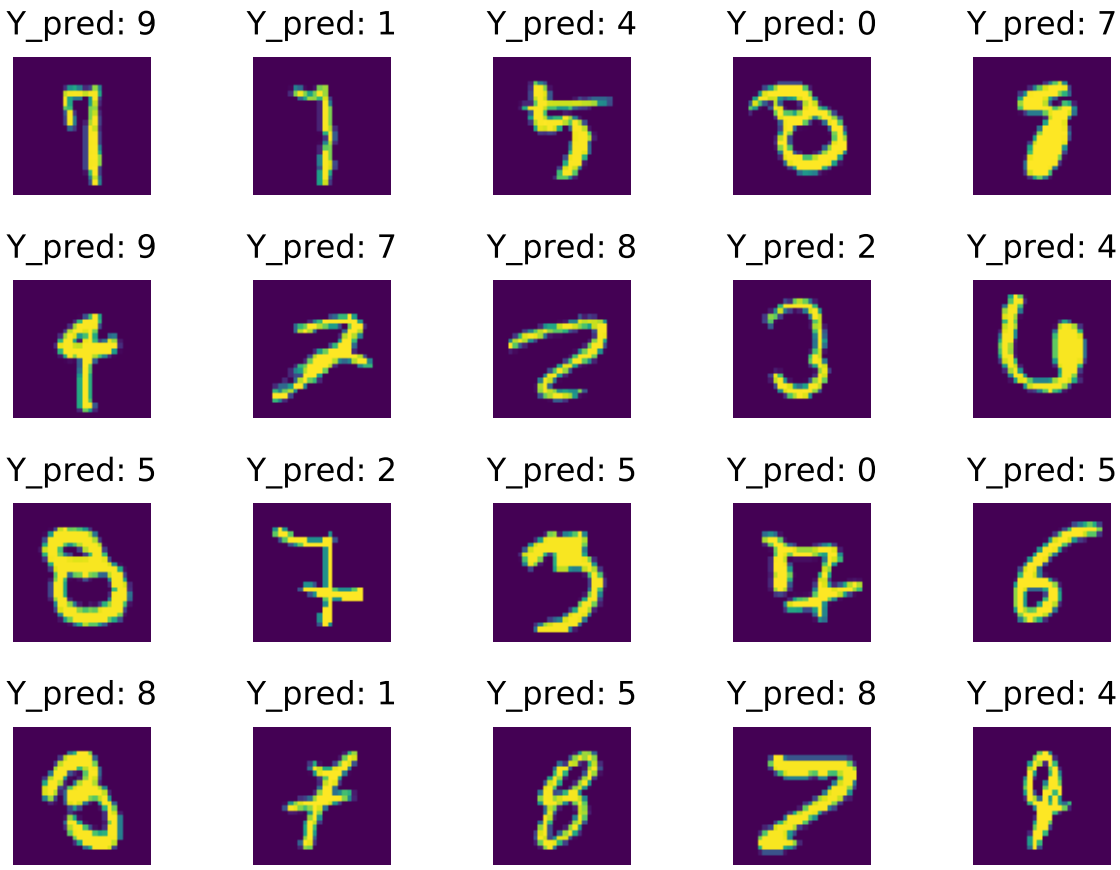

Figure 2: Missclassified digits randomly sampled from the test dataset.