Deep Learning Lab

Excercise 01

Feed-Forward Neural Networks

José Arce y de la Borbolla

October 29, 2018

Abstract

This report shows the results of training a set of neural networks with different parameters with the purpose of classifying the handwritten numerical digits of the MNIST data set.

1 Results

The Jupyter notebook already contained a Neural Network configuration, with an architecture as the one shown in figure 1, which will be used as a benchmark for further tests. Tables 1 and 2 show the parameterization and the metrics of that particular network respectively, as well as introducing the notation to be used in this paper.

Parameter	**Description**	**Value**
L	Number of (hidden) layers	4
n_x	Number of inputs	$28 \times 28 = 784$
n_y	Number of outputs	10
n^l	Number of units at layer l	$\{100, 100, 10, 10\}$
$g^l(z^{l-1})$	Activation Function al layer l	$\{\text{relu}(z), \text{relu}(z), z, \text{softmax}(z)\}$
A	Optimization Algorithm	sgd
η	Learning Rate	0.1
b	Batch Size	64
E	Number of Epochs	20
σ^0	Std Dev for Weight Initialization	0.1

Table 1: Benchmark Parameters.

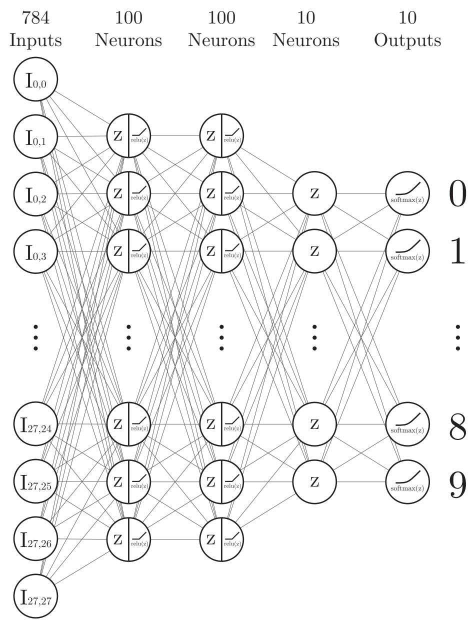


Figure 1: Benchmark Neural Network

Using the benchmark neural network configuration from the Jupyter Notebook, we get the following learning curve and metrics.

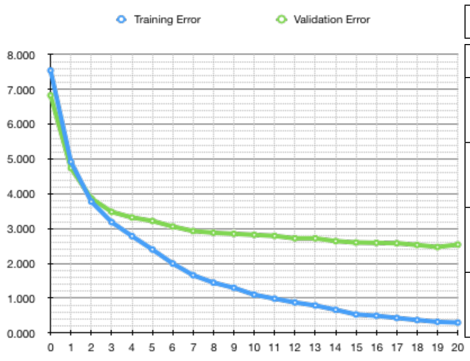


Figure 2: Learning Curve

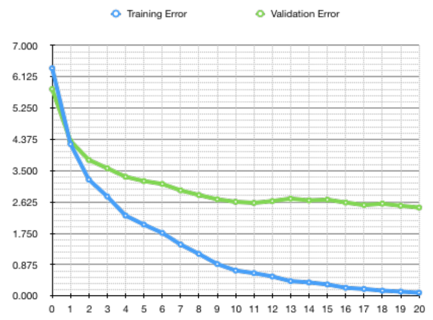
Metric	Description	Units	Value
t_T	Training Time	s	55.3670
$J^0(\hat{y}, y)$	Validation Loss at Epoch 0		23.2588
$J^E(\hat{y}, y)$	Validation Loss at Last Epoch		23.1343
ε^0	Validation Error at Epoch 0	%	6.83
ε^E	Validation Error at Last Epoch	%	2.54

Table 2: Benchmark Metrics.

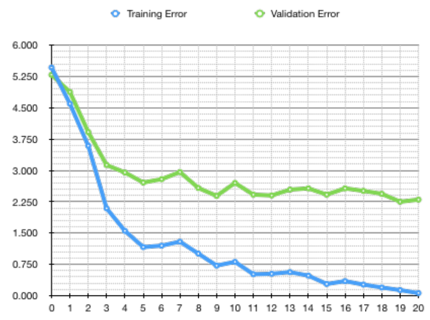
After running the training for the benchmark network, a series of tests were conducted by slightly altering the network parameters. For that purpose, the function `train()` was programmed, which takes a list of parameters in the form of dictionaries, creates a network for each entry in the list after said parameters, runs the neural network object's own train function, and finally logs the parameters in the `networks.txt`, and the outcome of each training epoch in `training.txt`. Both files are appended after each call to the `train()` function to store all of the data from the experimentation, and are related by a Network Identifier, which is assigned incrementally after each network configuration. The `networks.txt`, contains a comment field to more easily identify the changes made to the network, whereas the `training.txt` is a csv ideal to be imported in a spreadsheet program. The benchmark configuration corresponds to network 2 in the log files.

1.1 Changing the Learning Rate

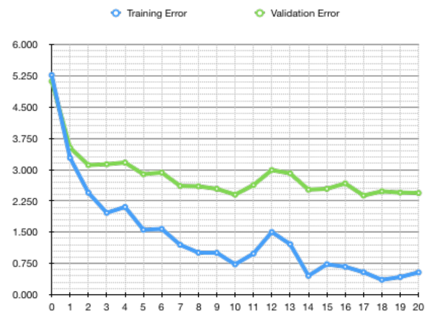
From the original settings, the learning rate (η) was the only parameter changed to the values shown in the captions of the following figures. The test cases correspond to the networks in the text files with the id's shown in the captions. The training time is also included in the captions for reference.



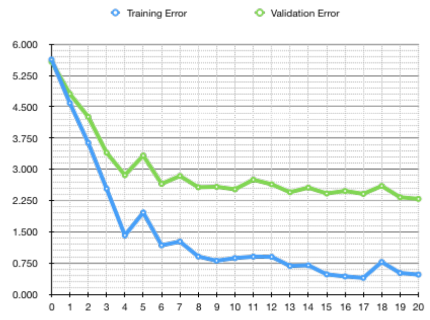
(a) $id = 5$, $\eta = 0.15$,
$t_T = 55.53s$, $\varepsilon^E = 2.46\%$



(b) $id = 6$, $\eta = 0.30$
$t_T = 55.00s$, $\varepsilon^E = 2.3\%$



(c) $id = 13$, $\eta = 0.45$
$t_T = 73.66s$, $\varepsilon^E = 2.29\%$



(d) $id = 14$, $\eta = 0.60$
$t_T = 69.53s$, $\varepsilon^E = 2.29\%$

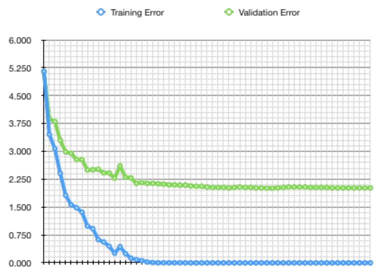
Figure 3: Changes to Learning Rate

As it can be appreciated in the figures, increasing the learning rate can make the slope of the learning curve more steep, meaning that during the first epochs, the network will learn faster, thus reducing the loss and error functions more rapidly. At the same time, greater values of η can lead to overshooting the local minimum during optimization, therefore an oscillation is perceptible in the plots. Because using $\eta = 0.30$ yielded a smaller error, while at the same time not adding a lot instability in the form of ripples to the learning curve nor taking extra time, it was selected for the following changes in parameterization.

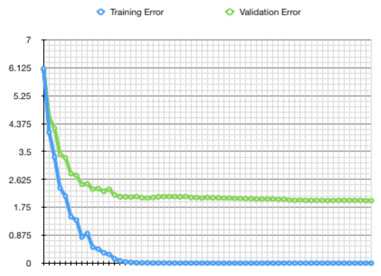
1.2 Adding Learning Rate Decay

After experimenting with several configuration changes, like modifying the batch size, increasing the maximum number of epochs, adding more hidden layers and with different activation functions, and still not being able to drop the error percentage below 2% in the validation data set, a Learning Rate decay was implemented in the code, such as after each iteration of training with the whole train dataset, the learning rate was scaled down by some fixed value.

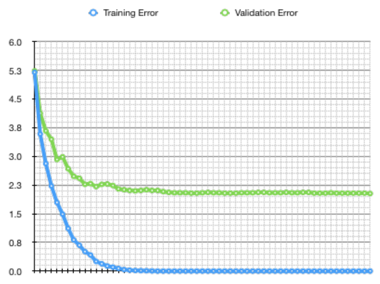
Using the configuration from the previous attempts but with an amount of epochs $E = 60$, the results of using a decay rate were as follows:



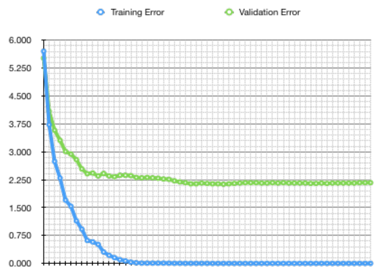
(a) $\eta^0 = 0.30$, $\eta^{e+1} = 0.99 \cdot \eta^e$
$t_T = 159.53s$, $\varepsilon^E = 2.02\%$



(b) $\eta^0 = 0.30$, $\eta^{e+1} = 0.98 \cdot \eta^e$
$t_T = 157.12s$, $\varepsilon^E = 1.96\%$



(c) $\eta^0 = 0.30$, $\eta^{e+1} = 0.97 \cdot \eta^e$
$t_T = 158.60s$, $\varepsilon^E = 2.03\%$



(d) $\eta^0 = 0.30$, $\eta^{e+1} = 0.96 \cdot \eta^e$
$t_T = 159.36s$, $\varepsilon^E = 2.17\%$

Figure 4: Changes to Learning Rate

It can be seen that the network consistently approaches the minimum in a much softer way than before, in that the overshoots caused by choosing a large value for η in the first epoch diminish after every pass, and that it reaches an even lower error rate than in the previous attempts. Larger values of decay lead to longer times for the convergence towards the minimum of the loss function and are therefore not displayed.

1.3 Discussion

After performing tests on various network configurations, the best error rate obtained from the MLP was 1.96%, using the following configuration with a learning rate decay of 0.98% every epoch. By the last epoch, the learning rate had dropped to about 30% of the original value, that is: $\eta^{60} = 0.0892$. Changing other hyper-parameters affected the processing time for the training, or required more epochs to even get to a low enough error on the training set. Other more advanced techniques like weight decay or dropout, as well as different optimization strategies like momentum gradient descent or RMSprop could be implemented to improve the classification further.