Neural Networks and Computational Intelligence

Assignment III: Learning by gradient descent

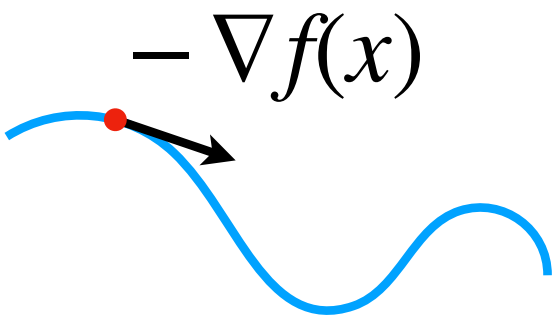
Group 12

Armand Colin S4994582

Andrea Gasparini S4993411

A.Y. 2021-2022

GitHub repository:
https://github.com/armand-colin/neural-networks-g12/tree/main/assignment3



Contents

1 Introduction

Stochastic gradient descent, or SGD, is an algorithm applied in machine learning. The goal of this algorithm is to minimise a cost function by iterative application of a gradient descent. When the cost function is a sum of costs of multiple examples, SGD iteratively applies the gradient descent of the example's cost on the accessible parameters of the function, to minimize the sum.

The goal of this assignment is to apply SGD on a shallow neural network, composed of several hidden layers, in order to fit a model to a data set, and eventually evaluate its accuracy.

2 Method

2.1 The regression problem

Suppose to have a data set $\{X^\mu, Y^\mu\}_{\mu=1}^P$, where we have P sets of N-dimensional feature vectors $X^\mu \in \mathbb{R}^N$ and real labels $Y^\mu \in \mathbb{R}$.

A regression problem will be to train a model so that its prediction $\sigma(X^\mu)$ (or output) of an example μ approaches Y^μ, and that for all μ.

2.2 The shallow neural network

In this assignment, we consider a simple shallow feed-forward neural network, with which output is defined as:

$$\sigma(X) = \sum_{k=1}^{K} v_k \, tanh(w_k \cdot X)$$

With K the number of hidden units, v_k the weight of each unit k in the output, and w_k the weight vector of a unit k.

We aim to minimize the cost function E:

$$E = \frac{1}{P} \sum_{\mu=1}^{P} e^\mu$$

Where e^μ is defined as the quadratic deviation, i.e.:

$$e^\mu = \frac{1}{2}(\sigma(X^\mu) - Y^\mu)^2$$

2.3 Gradient descent

Given a general shallow feed-forward neuron network of the form

$$\sigma(X) = h(\sum_{\mu=1}^{P} v_k \, g(w_k \cdot X))$$

We can write the gradient descent of the quadratic deviation e^μ of an example μ according to the weight vector w_k or according the unit weight v_k, respectively $\nabla_{w_k} e^\mu$ and $\nabla_{v_k} e^\mu$:

$\nabla_{w_k} e^\mu = \delta^\mu \, v_k \, g'(w_k \cdot X^\mu) X^\mu$
$\nabla_{v_k} e^\mu = \delta^\mu \, g(w_k \cdot X^\mu)$

With
$\delta^\mu = (\sigma(X^\mu) - Y^\mu) \, h'(\sum_{k=1}^{K} v_k \, g(w_k \cdot X^\mu))$

In our case, the function h is the identity function, so its derivative is worth 1. Moreover, we can compute the derivative of g:

$g(x) = tanh(x)$
$g'(x) = 1 - tanh^2(x)$

We can so simplify the gradient to our example:

$$\boxed{\begin{array}{l}\delta^\mu = (\sigma(X^\mu) - Y^\mu) \\ \nabla_{w_k} e^\mu = \delta^\mu \, v_k \, (1 - tanh^2(w_k \cdot X^\mu)) X^\mu \\ \nabla_{v_k} e^\mu = \delta^\mu \, tanh(w_k \cdot X^\mu)\end{array}}$$

2.4 Stochastic gradient descent

Now that we have our gradient terms, we can apply the SGD. This iterative algorithm is defined as follows:

- We randomly take an example μ from the data set of size P with a uniform probability $\frac{1}{P}$.
- We compute $\nabla_{w_k} e^\mu$ for every unit k
- We update the weight vector of the unit k according to the gradient descent of e^μ:
  $w_k = w_k - \eta \nabla_{w_k} e^\mu$
  With a given learning rate η. This parameter will by default be considered constant $\eta = 0.05$.
- We repeat the above steps for every example in the data set.

Going through all the examples μ is called an epoch. Usually, we will repeat the algorithm for several epochs, namely t_{max} epochs.

Note:
We could additionally update the units' weights with the units' weight vector, according to
$v_k = v_k - \eta \nabla_{v_k} e^\mu$

2.5 Validation

Given a data set, we would like to compare the fit of the model of training data to test data, that is not being used in the training. In this way, we can estimate the generalization of our model.

To do so, we take Q examples from the data set $\mathbb{D}$, that we call the test set, and P examples that we call the training set. Obviously, the initial data set should have sufficient examples, i.e. more that $P + Q$ examples. We then have $\mathbb{D}_{train}$ and $\mathbb{D}_{test}$, according to:

$\mathbb{D}_{train} = \{X^\mu, Y^\mu\}_{\mu=Q}^{P}$
$\mathbb{D}_{test} = \{X^\mu, Y^\mu\}_{\mu=1}^{Q}$

During the training, we can then evaluate the training and testing error:

$$E_{train} = \frac{1}{P - Q} \sum_{\mu=Q}^{P} e^\mu$$

$$E_{test} = \frac{1}{Q} \sum_{\mu=1}^{Q} e^\mu$$

Additionally, we randomize the indexes μ to train different models with different testing and training data sets to avoid some bias that can be induced by the initial data set.

3 Experimental setup

3.1 Training and validation

We've been given a data set containing 5000 examples of features of dimension 50. We are going to train our neural network with a given P and Q, and observe the corresponding E_{train} and E_{test}

for those trainings. Additionally, we are going to display the final weight vectors of the hidden units to see if there is any interesting behaviour to observe.

3.2 Effect of P

Further, we are going to take a look at the effect of P in the training and testing error. We are going to run several networks with values of P from 20 to 2000, and mean the results of the testing and training errors to have a better approximate of the corresponding performances. The number of networks used to mean the result will be denoted as n_{train}.

3.3 Influence of the learning rate η

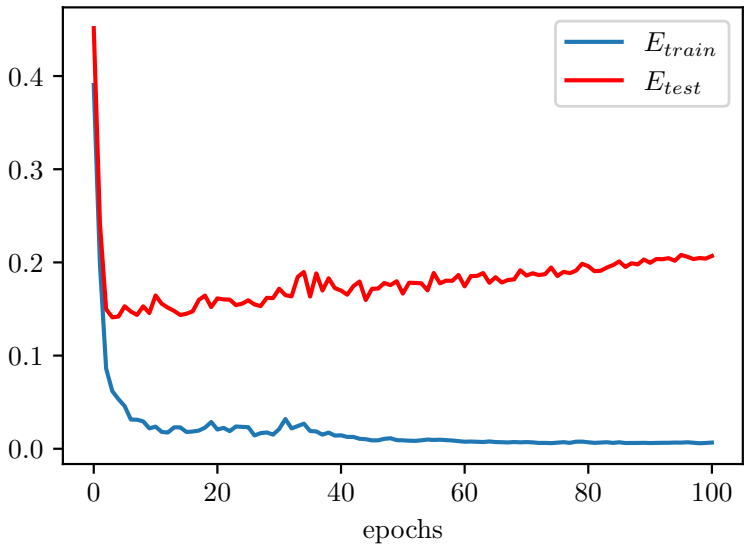
We are also going to observe the influence of the learning rate η on the network's performance, repeating the training process for several values of $\eta > 0$, from small ones such as 0.001 to bigger ones like 0.2, and then comparing the results in terms of E_{train} and E_{test}. Furthermore, we are going to consider a time dependent learning rate as discussed in class and see how this approach can yield better results in practice. In order to realize it, we defined the following decrease schedule based on the time step t:

$$\eta(t) = \frac{a}{b+t}$$

where $a, b > 0$ are constant parameters.
We are going to train with different values of b and a constant value of $a = 1.0$, aiming to observe how the behaviour change and to find the best performing one.

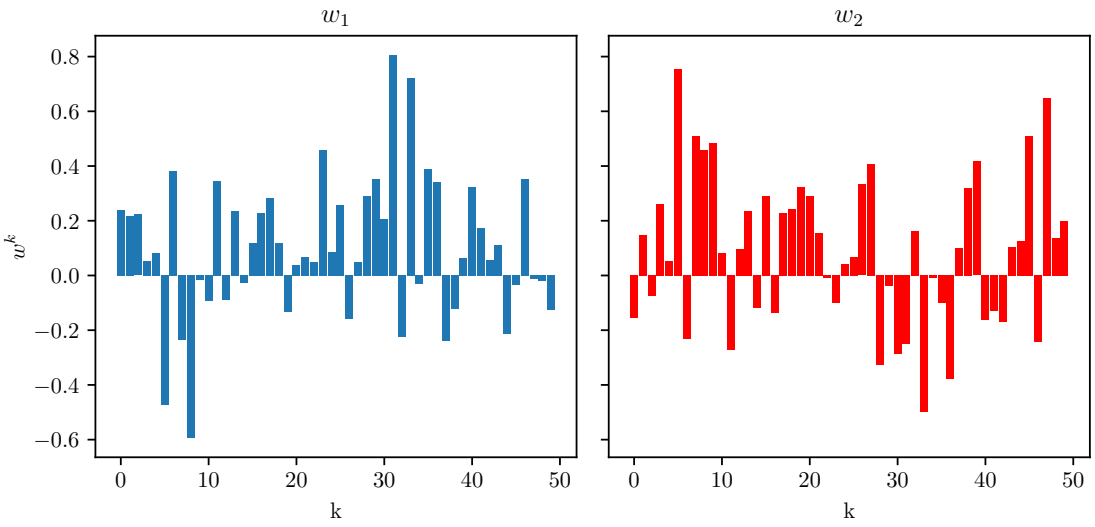
4 Results

4.1 Training and validation



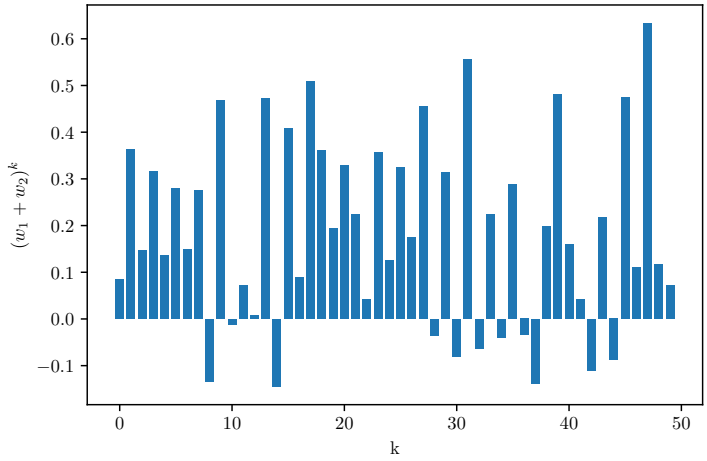
Parameters					
N	t_{max}	K	P	Q	η
50	100	2	100	100	0.05

Figure 1: Training and testing errors for simple training



Parameters					
N	t_{max}	K	P	Q	η
50	100	2	100	100	0.05

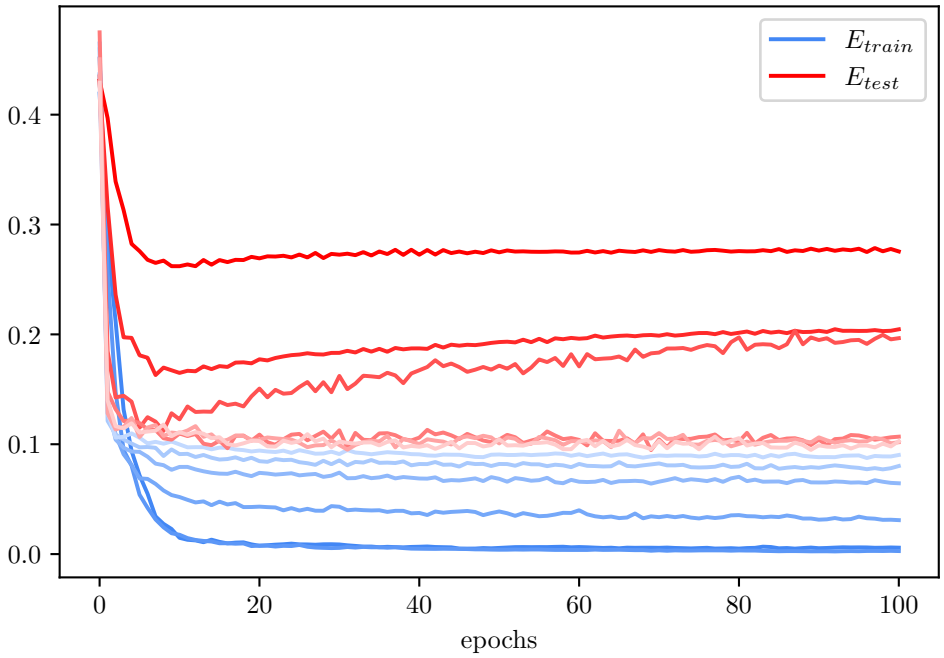
Figure 2: Weight vectors w_1 and w_2 at the end of the training



Parameters					
N	t_{max}	K	P	Q	η
50	100	2	100	100	0.05

Figure 3: Sum of weight vectors w_1 and w_2 at the end of the training

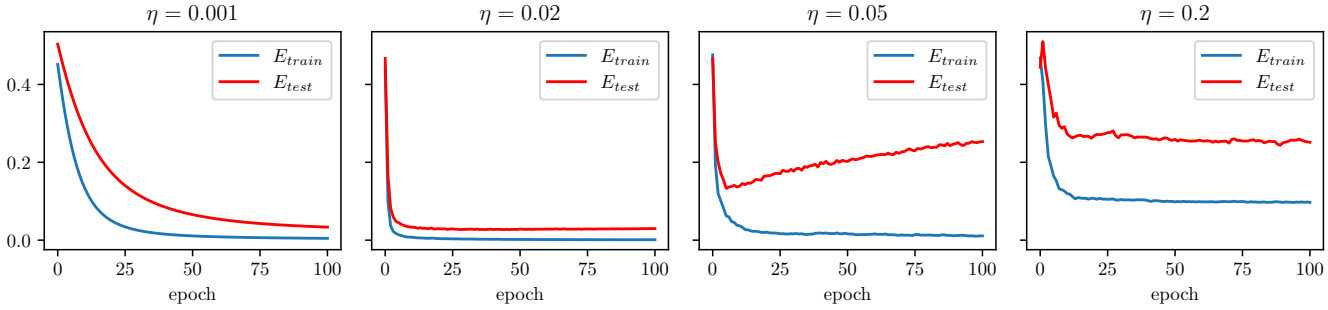
4.2 Effect of P



Parameters					
N	t_{max}	K	Q	η	n_{train}
50	100	2	100	0.05	10
Range of P					
20, 50, 200, 500, 1000, 2000					

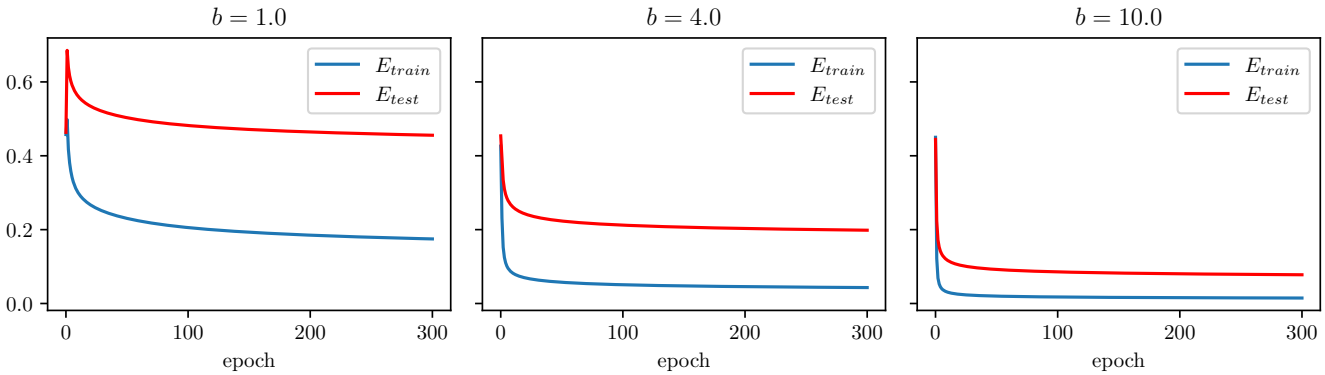
Figure 4: Mean training and testing errors for different values of P.
Lines are thick with $P = 20$, and gets lighter as $P \to 2000$

4.3 Influence of the learning rate η



Parameters					
N	t_{max}	K	P	Q	n_{train}
50	100	2	100	100	10

Figure 5: Training of models with constant learning rates



Parameters						
N	t_{max}	K	P	Q	n_{train}	a
50	300	2	100	100	20	1

Figure 6: Training of models with time dependent learning rates

5 Discussion

5.1 Training and validation

As we can see in Figure 1, our model succeeds to minimize E_{train}. At the beginning, E_{test} is also getting minimized. However, at around $t = 10$, we can see that E_{test} doesn't decrease anymore, but on the contrary grows. This is a sign of over-fitting, and could be predictable, considering the size of the training set.

With only $P = 100$, we can assume that our training sample is not that representative of the whole data set. This way, there is a point where the model is going to specialize on those examples, rather than on the underlying rule embedded in the data set.

In addition to that, we plotted the two final weights on Figure 2. On this figure, we can see that, in general, the two weight vectors do not have any similarity. When looking at Figure 3, we can observe that sometimes w_1 and w_2 balance each other, and sometimes add up to enforce a particular feature. This behaviour can be intended, as the final output is an average of the layers' output. Some features will sometimes force the weights in the right direction if the effect is clear, and sometimes the random nature of the initial weights will make them try to balance each other.

5.2 Effect of P

As clearly shown in Figure 4 the behaviour of E_{train} and E_{test} is highly affected by the dimension P of the training data set. In fact, we observed that the larger the training data set is, the more E_{test} goes down and similar to E_{train}, with the latter which goes up at the same time. For the higher values of P the two cost functions tend to almost match.

Another interesting finding visible in Figure 4 is that after a certain threshold of P (in our case $P \geq 500$) we have E_{test} which does not decrease anymore for greater sizes of the training data set. At the same time E_{train} continues to increase. This behaviour led us to think that even though we are increasing the size of training set, we are only adding samples that are not representative of the ones in the test set and therefore do not contribute to the performance. Adding more training examples would just be equivalent to train on the whole data set, and thus matching the test and training errors.

This experiment leads us to confirm the assumption previously made in subsection 5.1 for which a low value of P is very likely not representative of the whole data set and therefore training on a greater fraction yields to a better generalization.

5.3 Influence of the learning rate η

As we can see in Figure 5, for small values of the learning rate, we have smooth curves that slowly tend to a low value; with a small increase of η the curves stay smooth but reaching a "stable" point quicker. In both cases E_{train} and E_{test} are close between them, maintaining a similar shape.

For larger values of η we can notice a worsening behaviour in the curves, with E_{test} that stops decreasing earlier, getting further from E_{train}, and starting to have a *zigzagging* behaviour and eventually to get up again. In all the cases we can observe that E_{train} gets worse along with the learning rate η, even though in a slower way than E_{test}.

In Figure 6 we show the behaviour of the time dependent η and the effect of the parameter b on it. One interesting finding to show, that confirms both our expectations and the theory, is that the time dependency leads to a smooth behaviour without *zigzagging* or going up. This applies after the first time steps in which the value of η is very large and therefore yield an high pitch before smoothly going lower. However, thanks to the tuning we performed, we discovered that we could also avoid the high pitch using higher values of the parameter b.

Even though we obtained good results with $b = 10$, we did not manage to get a better minimization of the cost function than, for instance, with constant value of $\eta = 0.02$. Therefore, generally speaking, we can say that this schedule and these constant parameter values, work great to quickly obtain good results without the need of using further validation techniques (e.g. early stopping), but at the cost of a tuning phase of constant learning rates, we could obtain better results.

6 Conclusion

In this assignment we managed to implement the Stochastic Gradient Descent algorithm. This algorithm was used in a multiple shallow neural networks, to learn from examples, and study their performance.

We have observed the training and validation phases of a network and the effect of the training examples on the final performance. For good generalization, having a large amount of data will lead to learn a rule closer to the global one, rather than the one related to the used subset which could be biased. Even though the training accuracy will not be as good, the final accuracy will be closer to what we could expect with novel data.

Furthermore, we have observed the influence of the learning rate η on the network's performance. We tried out several constant values, compared the behaviour of the trained models and observed great optimization leads. We also implemented a time dependent learning rate and compared its behaviour with the constant values ones. Eventually, the time dependent learning rate can easily lead to good results, but at the same time a constant value could be a better option.

Note on individual workload Since we worked on a shared repository on GitHub and the majority of the time using a VoIP software, our individual workload is approximately equal.