Neural Networks

1 Introduction

The objective of this practical is to get a grasp of Neural Networks (NN) with Python. The reader should refers to the lecture slides of the course in "Artificial Intelligence".

Given a neural network characterized by 2 inputs, 2 outputs and a hidden layer, you will proceed as follows:

1. With an initial guess of the weights, implement a python function for doing a forward pass of the network (Sect. 2).
2. In the same function, implement the backpropagation of the network, and test everything with a dataset of two labeled samples. (Sect. 2).
3. With the same dataset, run the network (forward and back) to train it over *many* iterations (Sect. 3.1).
4. Finally, train and then test the network with the bigger dataset provided in with the provided datasets `NNTraining.data` and `NNTest.data` (Sect. 3.2).

For this practical, you will need some python packages. Add the following lines to your python script to ensure that all packages are imported correctly.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2 Programming the network

The goal of this section is to design the network shown in Fig. 1.

You dispose of two labeled samples. Each sample contains input $\mathbf{x}$ with corresponding target output $\mathbf{t}$:

$$\mathbf{x}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \mathbf{t}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} -1 \\ 3 \end{bmatrix} \quad \mathbf{t}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{1}$$

Proceed as follows.

1. Design a function for the forward pass, which takes as arguments vectors $\mathbf{x}$, $\mathbf{t}$ and $\mathbf{w}$ and returns the loss L (squared error between $\mathbf{y}$ and $\mathbf{t}$). Initialize the network with the following weights:

$$\mathbf{w} = \begin{bmatrix} 2 & -3 & -3 & 4 & 1 & -1 & 0.25 & 2 \end{bmatrix}^\top . \tag{2}$$

   You should obtain: $L = 0.245$ and $L = 0.330$, respectively. For this exercise, it is wise to define a function returning the sigmoid output $\sigma(z)$ (see lecture slides).

2. In the same function, implement the backpropagation of the network. Along with the loss L, the function should now return its gradient $\nabla_{\mathbf{w}} L$. You should obtain:

$$\begin{aligned} \nabla_{\mathbf{w}} L &= \begin{bmatrix} -0.12 & 0.07 & -0.06 & 0.03 & -0.10 & 0.13 & -0.02 & 0.02 \end{bmatrix}^\top \\ \nabla_{\mathbf{w}} L &= \begin{bmatrix} \sim 0 & \sim 0 & \sim 0 & \sim 0 & \sim 0 & \sim 0 & 0.28 & -0.025 \end{bmatrix}^\top . \end{aligned} \tag{3}$$

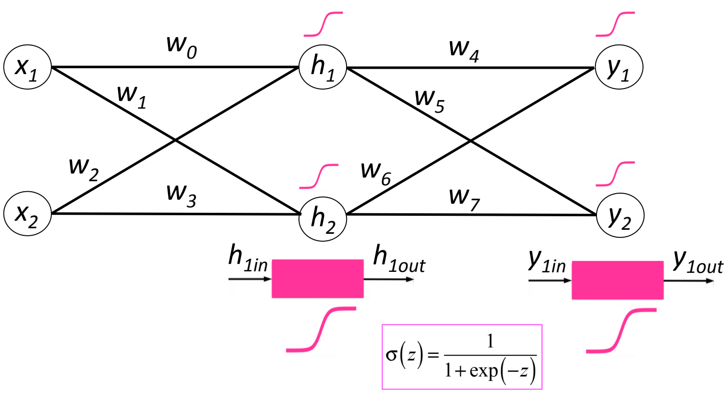


Figure 1: Neural network studied in this practical.

3 Training and testing the network

3.1 Training and testing on three samples

With the same dataset, run the network (forward and back) to train it until the loss function has diminished *enough*. Since the dataset consists in only two samples, we use a single batch coinciding with the whole dataset.

You will need to manually tune two hyperparameters:

1. the gradient descent step size ρ,

2. the number of times (iterations) N you compute the mean of the gradient over the batch[1].

After each iteration, store the loss L so you can plot its trend over the number of iterations. Tune ρ and N until you find a satisfactory trend (as in Fig. 2).

Test with a new labeled sample:

$$\mathbf{x}_3 = \begin{bmatrix} 1 \\ 4 \end{bmatrix} \qquad \mathbf{t}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{4}$$

and comment.

[1]Here, we use this criterion instead of the tolerance on weight variation seen in class.

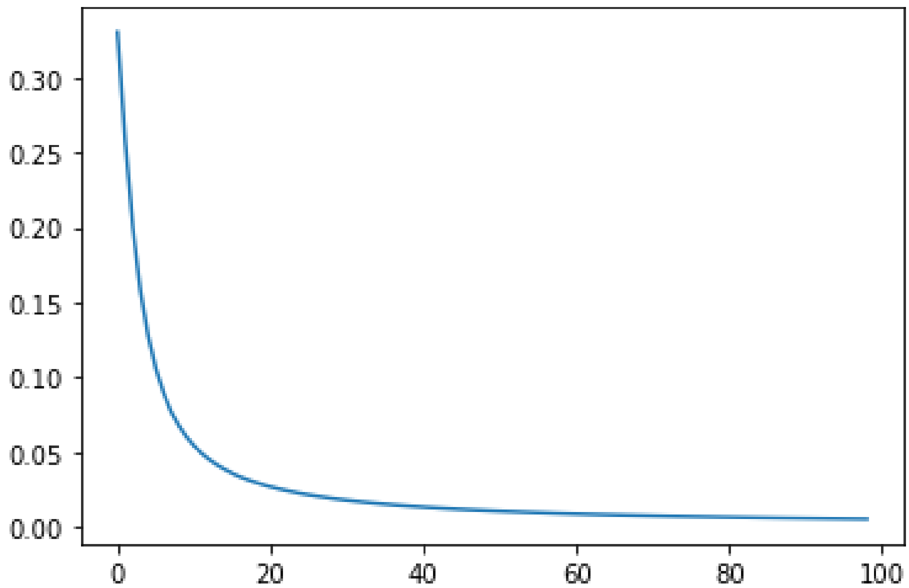


Figure 2: Evolution of the loss when training with two samples.

3.2 Training and testing on a large dataset

Now, train the network on the provided dataset `NNTraining.data`, starting from the same weight as before (equation (2)). To load the dataset you can use the following snippet:

```
data = pd.read_csv('NNTraining.data')
x1 = data.iloc[:,0]
x2 = data.iloc[:,1]
t1 = data.iloc[:,2]
t2 = data.iloc[:,3]
```

Since this dataset is much larger (200 samples) it can be split in equally sized batches for more efficient learning. After all samples in a batch have been passed forward, compute the average gradient over all samples, and update **w**. After each epoch (i.e., each time all batches have been passed forward), store loss L so you can plot its trend at the end of training.

This time you will need to manually tune three hyperparameters:

1. the gradient descent step size ρ,
2. the batch size m,
3. the number of epochs N.

Start with a single batch of size $m = 200$, then increase the number of batches and check how the loss function evolves. After a bit of hyperparameter tuning, I obtained the curve in Fig. 3. Compare the final loss you obtain here with the one obtained in Sec. 3.1.

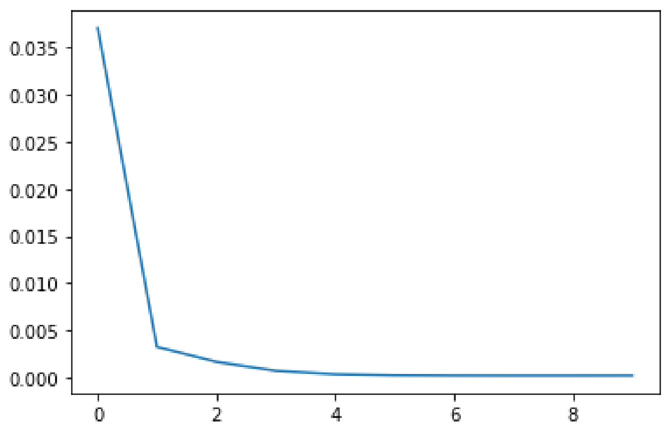
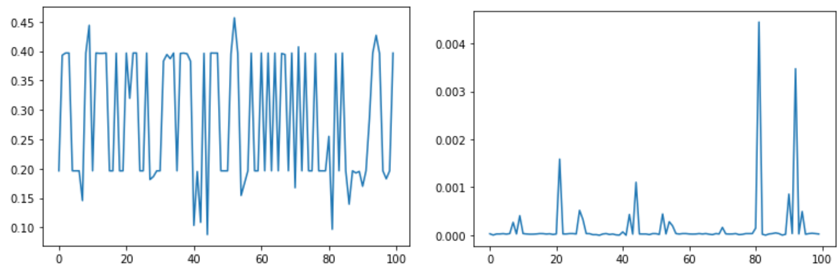


Figure 3: Evolution of the loss when training with `NNTraining.data`.

How does the trained network generalize to new data? To assess this, test the network (only forward) on the test dataset `NNTest.data` and plot the 100 losses. You can plot them before and after training to see the difference, as I did in Fig. 4.



(a) Before training (b) After training

Figure 4: Loss obtained when passing each of the 100 samples of `NNTest.data` through the network.