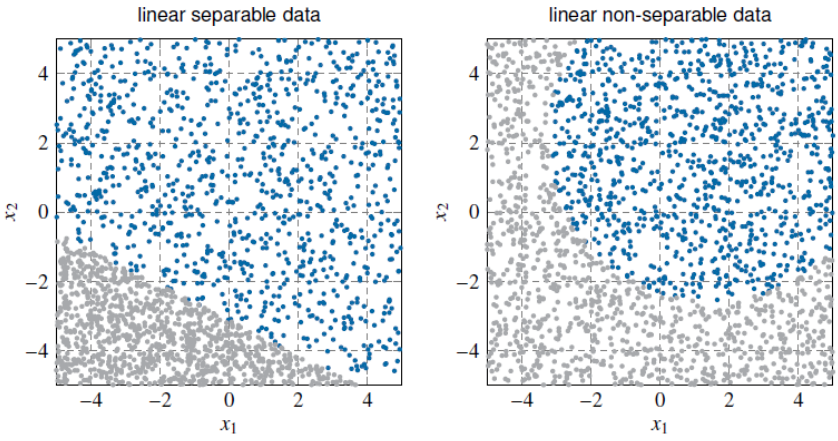
OST Ostschweizer Fachhochschule

Problem 1: Neural Network from Scratch

1. Description

In this problem, you will have to develop a simple neural network from scratch – so without the help of any high-level libraries. You are given two (simulated) training datasets and validation datasets with 2 predictors X_1 and X_2, and 2 possible classes (0 and 1) each. The two datasets are shown below.



Your tasks are:

- Develop a single neuron using only the Python NumPy library.
- Develop a neural network with one hidden layer using only the Python NumPy library.

The mathematical details for both methods are in the appendix of this document. For both tasks, there is already Python code available as a starting point. Your task is to implement the missing parts, which are marked with “TODO” comments in the code.

2. Formal requirements

You need to create a Python script, which implements a simple neuron and a neural network for the given dataset, which uses only the libraries NumPy and Matplotlib. All calculations that are required for the neural network must be implemented using NumPy.

Your submission on Moodle must contain:

- Python script “neuron.py” which contains your implementation of the single neuron.
- Python script “network.py” which contains your implementation of the neural network.

3. Submission & deadlines

Deadline for submission:	30 March 2025 until 23:59
Submission via:	Moodle – DL FS2025 – Submission of Problem 1
	See formal requirements on what must be submitted.

4. Evaluation

Your solution is marked with “pass” or “fail”. It is graded as “pass”, if

- Your solution was submitted on time.
- At least **2 out of 3 points** in the following evaluation is reached:

Criteria	0 Points	1 Point	Comment
1. Correct implementation of neuron	☐	☐	
2. Correct implementation of neural network	☐	☐	
3. Formal requirements are met (see 2.)	☐	☐	
Total points:	___ / 3 Points		
Pass / Fail:	☐ Pass ☐ Fail		

Deep Learning

Project 1: Neural Networks from Scratch

Mathematical Background

ICAI

FS 2025

1 A Single Neuron

We start with the simplest possible network that consists of a single neuron with two inputs (see Figure 1).

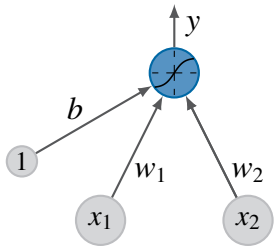


Figure 1: A single neuron with two inputs.

As activation function of this neuron the sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{1.1}$$

is used, where a is called the *pre-activation* and is given by

$$a(\boldsymbol{w}, b) = \boldsymbol{w}^T \boldsymbol{x} + b = w_1 x_1 + w_2 x_2 + b.$$

The predicted output of the neuron is calculated as

$$y(\boldsymbol{w}, b) = \sigma(a) = \sigma(x_1 w_1 + x_2 w_2 + b).$$

The mean square error (MSE) over a mini-batch of N samples is used as loss function

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - t_i)^2. \tag{1.2}$$

Here, y_i is the predicted output of the neuron for the input $\boldsymbol{x}_i$, and t_i is the corresponding (true) target value.

The training procedure as well as the evaluation of the network (neuron) have already been implemented in the file `train_neuron.py`. The training and the evaluation, however, are not working properly yet, since the functions of the module `neuron.py` are not implemented yet.

It is your task to implement all the functions in the module `neuron.py`. To test your functions, you can use the unit test in the module `neuron.py` (code inside the `if` statement). As soon as your functions have passed the unit test, you can train and test the network with the script `train_neuron.py`. Try to understand the code of this script. Also vary the parameters (such as `linearly_separable`, `batch_size`, `learning_rate`, `n_epochs`) and try to understand the different outcomes.

Hint:
Start by calculating the gradient $\nabla J(\boldsymbol{w}, b)$ by hand / on paper.

2 A Tiny Neural Network

With a single neuron, it is only possible to classify linearly separable datasets. However, the second dataset in is not linearly separable. Hence, a neural network with more capacity is needed. We will use a neural network as shown in Figure 2. Again, the sigmoid function (1.1) is used as the activation function and the mean squared error (MSE) (1.2) is used as the loss function.

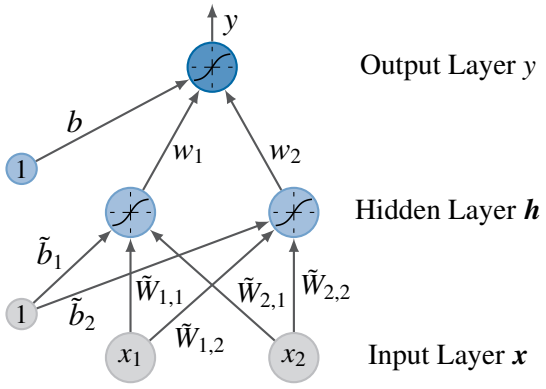


Figure 2: A neural network with two inputs and two hidden neurons.

The calculation of the output y is somewhat more complicated here:

$$y(\boldsymbol{w}, b, \tilde{\boldsymbol{W}}, \tilde{\boldsymbol{b}}) = \sigma\left(\boldsymbol{w}^T \boldsymbol{h} + b\right) = \sigma\left(\boldsymbol{w}^T \sigma(\tilde{\boldsymbol{W}}^T \boldsymbol{x} + \tilde{\boldsymbol{b}}) + b\right).$$

Again, the training procedure and the evaluation of the network have already been implemented in the file `train_network.py`. The training and the evaluation, however, are not working properly yet, since the functions of the module `network.py` are not implemented yet.

It is your task to implement all the functions in the module `network.py`. To test your functions, you can use the unit test in the module `network.py` (code inside the `if` statement). As soon as your functions have passed the unit test, you can train and test the network with the script `train_network.py`. Try to understand the code of this script. Also vary the parameters (such as `linearly_separable`, `batch_size`, `learning_rate`, `n_epochs`) and try to understand the different outcomes.