W3WI DS304.1 Applied Machine Learning Fundamentals

Exercise Sheet # 5 - Neural Networks / Deep Learning

Question 1 EX 2021 (Perceptron)

Under what circumstances does the *Perceptron* learning algorithm converge?

Question 2 EX 2020 (Number of network parameters)

You want to train a neural network on the *MNIST* dataset to recognize hand-written digits. The images of 10 possible digits (the classes) have a resolution of 28×28 pixels.

The MLP (*multi-layer perceptron*) used for the task has two hidden layers with 64 and 32 units, respectively. Each layer has a constant bias input and the classes are one-hot encoded.

How many adjustable network parameters does the model have?

Question 3 EX 2020 (Neural networks for regression)

Your colleague suggests to use neural networks to solve a regression task. Which activation function would you have to use in the output layer of your network to achieve the desired result?

Question 4 EX 2021 (Activation functions)

Which statements regarding the activation functions of neural networks are correct?

- ☐ Activation functions should be non-linear.
- ☐ The *softmax* activation function is usually used in the output layer of a neural network.
- ☐ One problem of the *ReLU* function is the vanishing gradient.
- ☐ The *ReLU* activation is computed according to $\min(0, x)$.

Question 5 EX 2021, modified (Network architectures)

What kind of neural network is usually applied to classify (a) images, and (b) sequences? Do some research and explain these types of networks.

Question 6 2023, modified (Network training with gradient descent)

Your task is to train the neural network depicted in figure 1 using the *stochastic gradient descent* algorithm. The current training example is given by the vector $\boldsymbol{x} := \begin{pmatrix} 0 & 1 \end{pmatrix}^\intercal$. The label is one-hot encoded and given by $\boldsymbol{y} := \begin{pmatrix} 1 & 0 \end{pmatrix}^\intercal$.

Network architecture: The network consists of one hidden layer with *ReLU* activation as well as an output layer with *sigmoid* activation. For simplicity you decide to use the *least squares error* as the loss function (as shown in the lecture).

Compute the weight updates for all network parameters when applying the learning rate $\alpha := 0.5$.

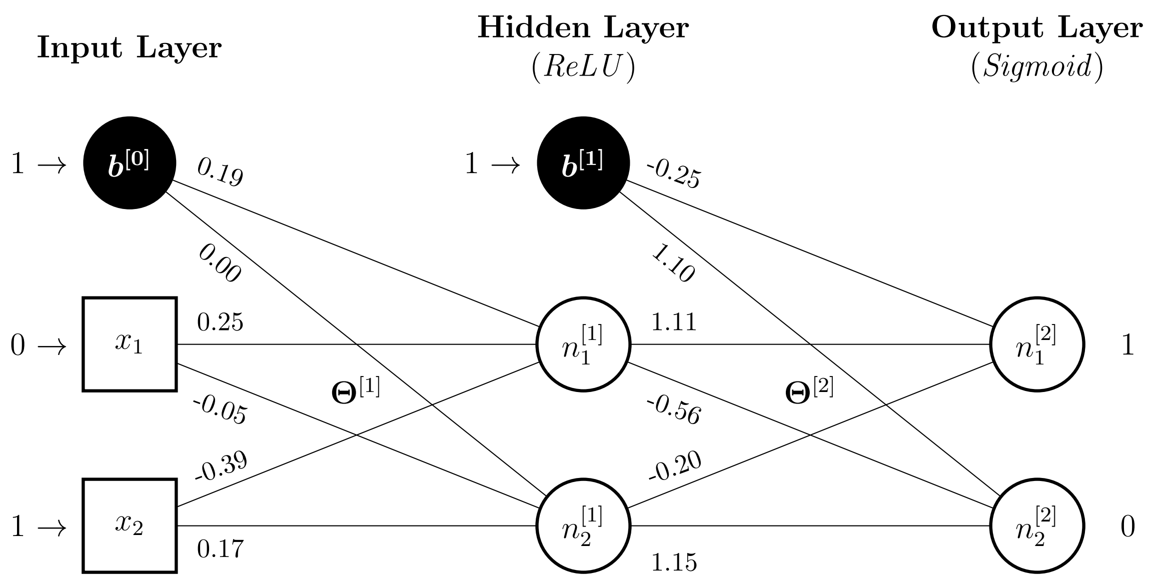

Figure 1: Visualization of the neural network used for the task above.

Question 7 (Implement a neural network using PyTorch)

Generate a training dataset using the Python snippet given below. This snippet generates a **spiral dataset** consisting of $n = 2$ classes. This dataset is highly non-linear. The dataset is depicted in figure 2 below.

Please work through the following tasks:

1. Start by installing the `PyTorch` library. For this, download the installer officially provided on the `PyTorch` website: `https://pytorch.org/get-started/locally/`.

2. Implement and train a neural network on the training dataset mentioned above. You are free to use any network architecture you think is useful. Plot the decision boundary generated by your model!

3. Play around with the hyper-parameters of your network (# hidden layers, # neurons, loss function, etc.) and see how this influences the decision boundary.

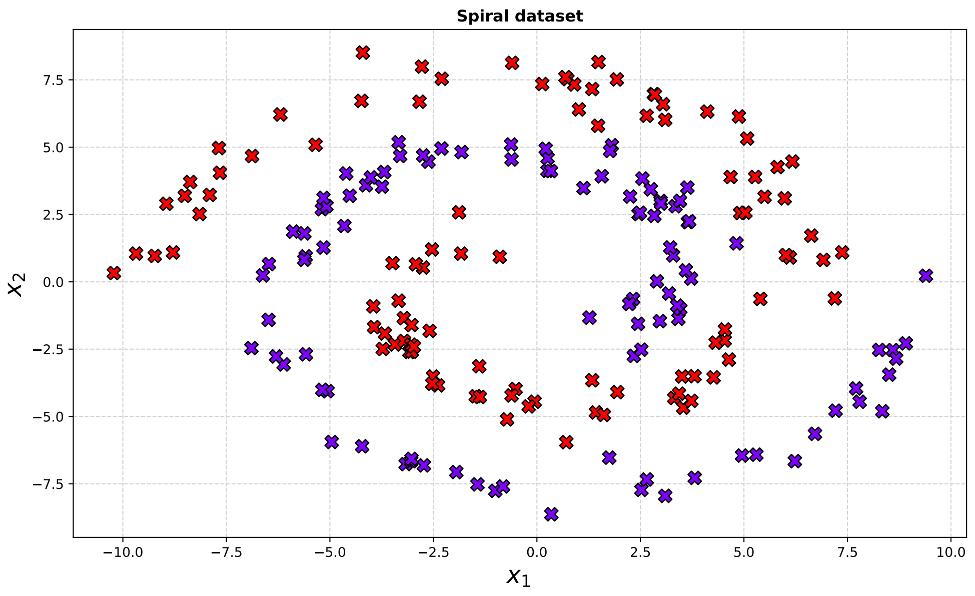

Figure 2: Plot of the spiral dataset consisting of two classes.

```python
# IMPORTS
# ----------------------------------------------------------------
import numpy as np


# DATA CREATION
# ----------------------------------------------------------------
def make_spiral(n_samples=100):
    """
    Generates a spiral data set.

    :param n_samples:   number of data points to be generated
    :return:            X, y (data set)
    """
    # class 0
    t = 0.75 * np.pi * (1 + 3 * np.random.rand(1, n_samples))

    x1 = t * np.cos(t)
    x2 = t * np.sin(t)
    y = np.zeros_like(t)

    # class 1
    t = 0.75 * np.pi * (1 + 3 * np.random.rand(1, n_samples))

    x1 = np.hstack([-x1, t * np.cos(t)])
    x2 = np.hstack([-x2, t * np.sin(t)])
    y = np.hstack([y, np.ones_like(t)])
    # concatenate feature vectors
    X = np.concatenate((x1, x2))
    # add some noise
    X += 0.50 * np.random.randn(2, 2 * n_samples)

    return X.T, y[0]


# generate the dataset
X, y = make_spiral(100)
```