Artificial Intelligence and Machine Learning

Exercises – Neural Networks / Deep Learning

Question 1 (Perceptron) ⊛

Under what circumstances does the Perceptron learning algorithm converge?

Question 2 (Number of network parameters) ⊛

You want to train a neural network on the *MNIST* dataset to recognize hand-written digits. The images show $K = 10$ possible digits (the classes) and have a resolution of 28×28 pixels. The MLP 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 (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? Which cost function would be best for training?

Question 4 (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 $\text{ReLU}(x) = \min\{0, x\}$.

Question 5 (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 (Network training with gradient descent)

The neural network depicted in figure 1 is given to you. It consists of one hidden layer with three hidden units and sigmoid activations, and an output layer with two output neurons and softmax activations.

The weights are initialized with

$$\boldsymbol{W}^{[1]} := \begin{pmatrix} 0.10 & 0.20 \\ 0.15 & -0.10 \\ -0.20 & 0.30 \end{pmatrix} \in \mathbb{R}^{3\times 2}, \qquad \boldsymbol{W}^{[2]} := \begin{pmatrix} 0.00 & -0.10 & 0.20 \\ -0.20 & 0.05 & 0.15 \end{pmatrix} \in \mathbb{R}^{2\times 3}.$$

The bias weights are

$$\boldsymbol{b}^{[1]} := \begin{pmatrix} 0.10 \\ -0.10 \\ 0.05 \end{pmatrix} \in \mathbb{R}^{3}, \qquad \boldsymbol{b}^{[2]} := \begin{pmatrix} 0.20 \\ 0.15 \end{pmatrix} \in \mathbb{R}^{2}.$$

Perform one iteration of stochastic gradient descent based on the training example $(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x} := (0, 1)^\top$ are the features, and $\boldsymbol{y} := (1, 0)^\top$ the one-hot encoded label (two classes). Use the learning rate $\alpha := 0.1$. Round all intermediate results to three decimal places. Steps to do in detail:

1. Compute the forward pass through the network.
2. Backward pass: Compute the error gradients for all layers.
3. Backward pass: Compute the weight gradients for all layers.
4. Update the weights using the gradient descent update formula.

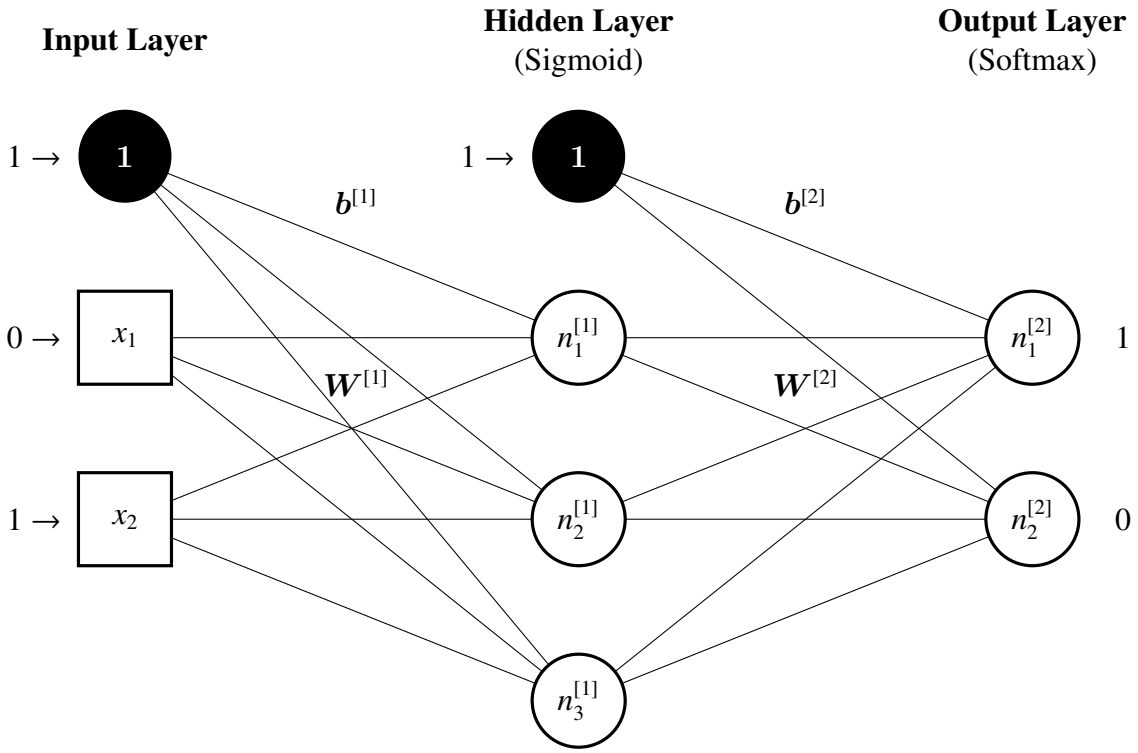

Figure 1: Visualization of the neural network used in the task.

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 $K = 2$ classes (purple and yellow crosses). The dataset is highly non-linear as can be seen in figure 2 below:

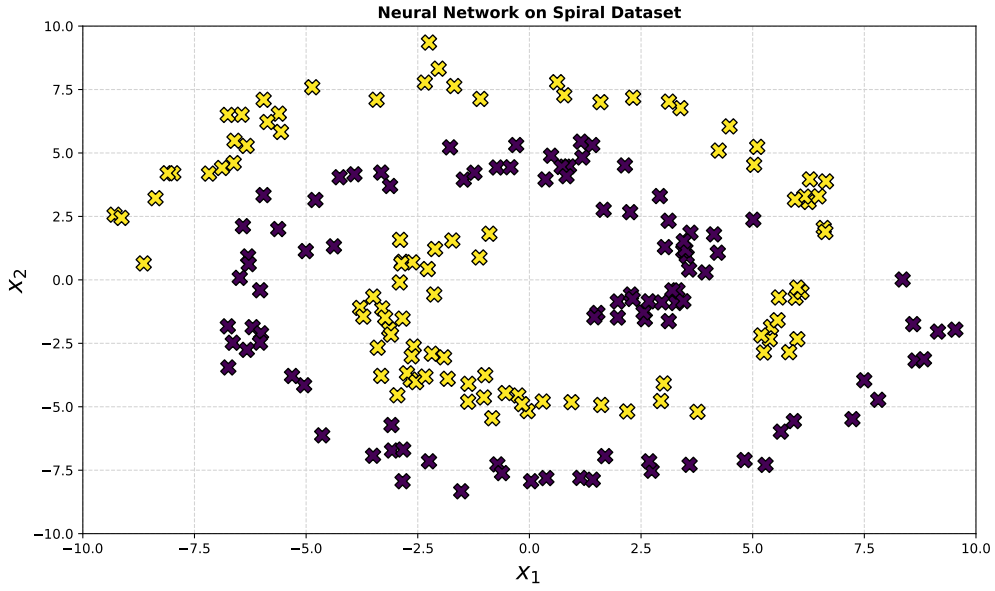

Figure 2: Plot of the spiral dataset consisting of two classes.

```
import numpy as np
np.random.seed(42)

def make_spiral(n_samples=100):
    # 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)])
```

```
23      # concatenate data points for both classes
        X = np.concatenate((x1, x2))
25      # add some noise
        X += 0.50 * np.random.randn(2, 2 * n_samples)
27
        return X.T, y[0]
29
    # generate the dataset
31  X, y = make_spiral(100)
```

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/ and follow the instructions.

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

3. Play around with the hyperparameters of your network (# hidden layers, # neurons, loss function, etc.) and see how this influences the decision boundary. Can you find a network architecture that classifies all training records correctly?

4. Implement an MLP from scratch (only using Numpy) and train it on the dataset generated above.