Machine Learning *COMP 5/6630*

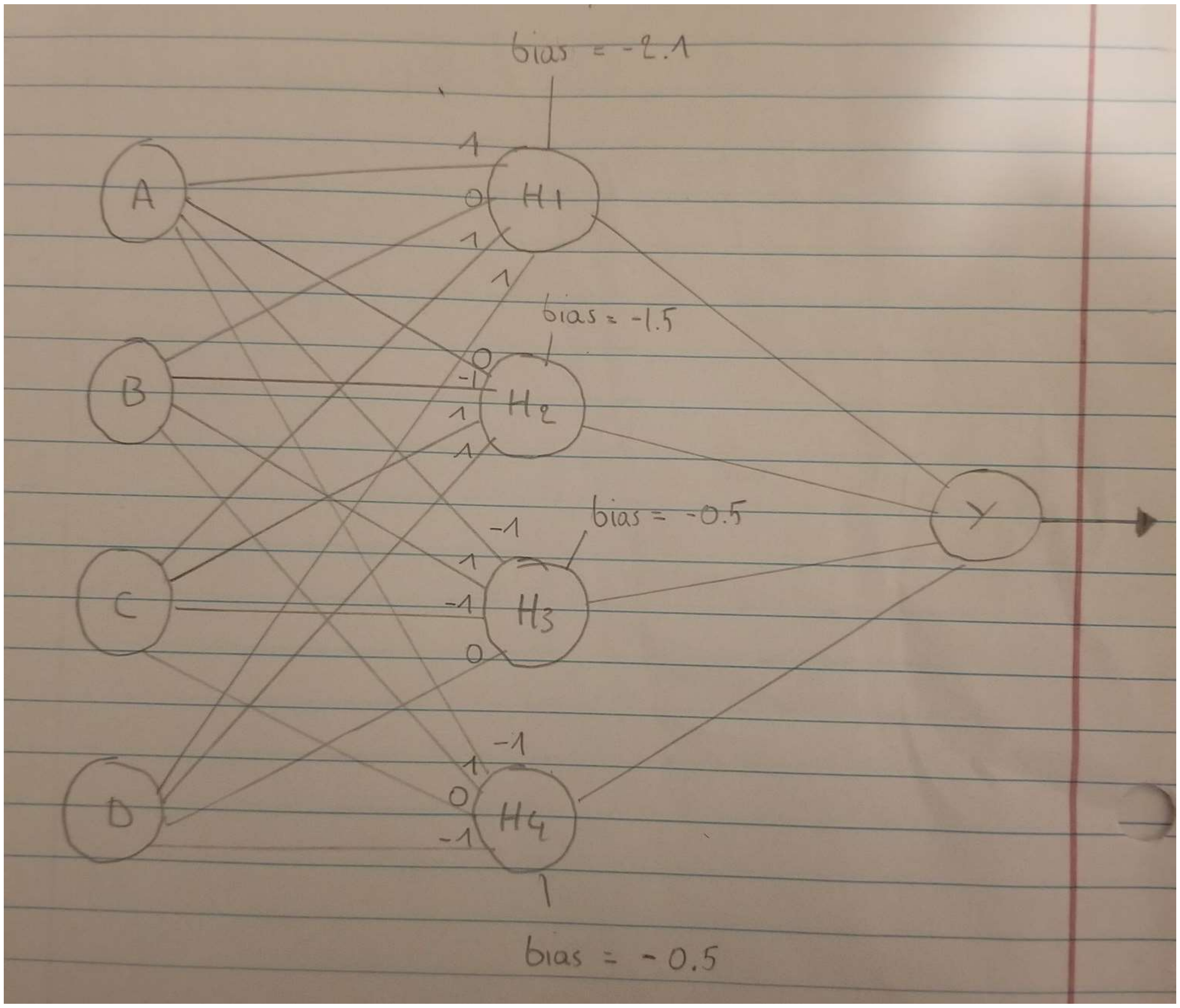
# HW06: Neural networks

Remember that only PDF submissions are accepted.

1. Create a neural network with only one hidden layer (of any number of units) that implements $(A \vee \neg B) \oplus (\neg C \wedge \neg D)$. Draw your network, and show all weights of each unit. For simplicity use the step function as the link function, i.e., $f(x) = 1[x \geq 0]$. Assume that you have access to a bias feature at each layer.

   An XOR operation can be written in terms of an AND and OR operation. This will result into the rewriting of the formula above, in the following way: (A ∧ C ∧ D) ∨ (¬B ∧ C ∧ D) ∨ (¬A ∧ B ∧ ¬C) ∨ (¬A ∧ B ∧ ¬D)

   Now we can represent this by using a neural network with one hidden layer that contains four nodes.

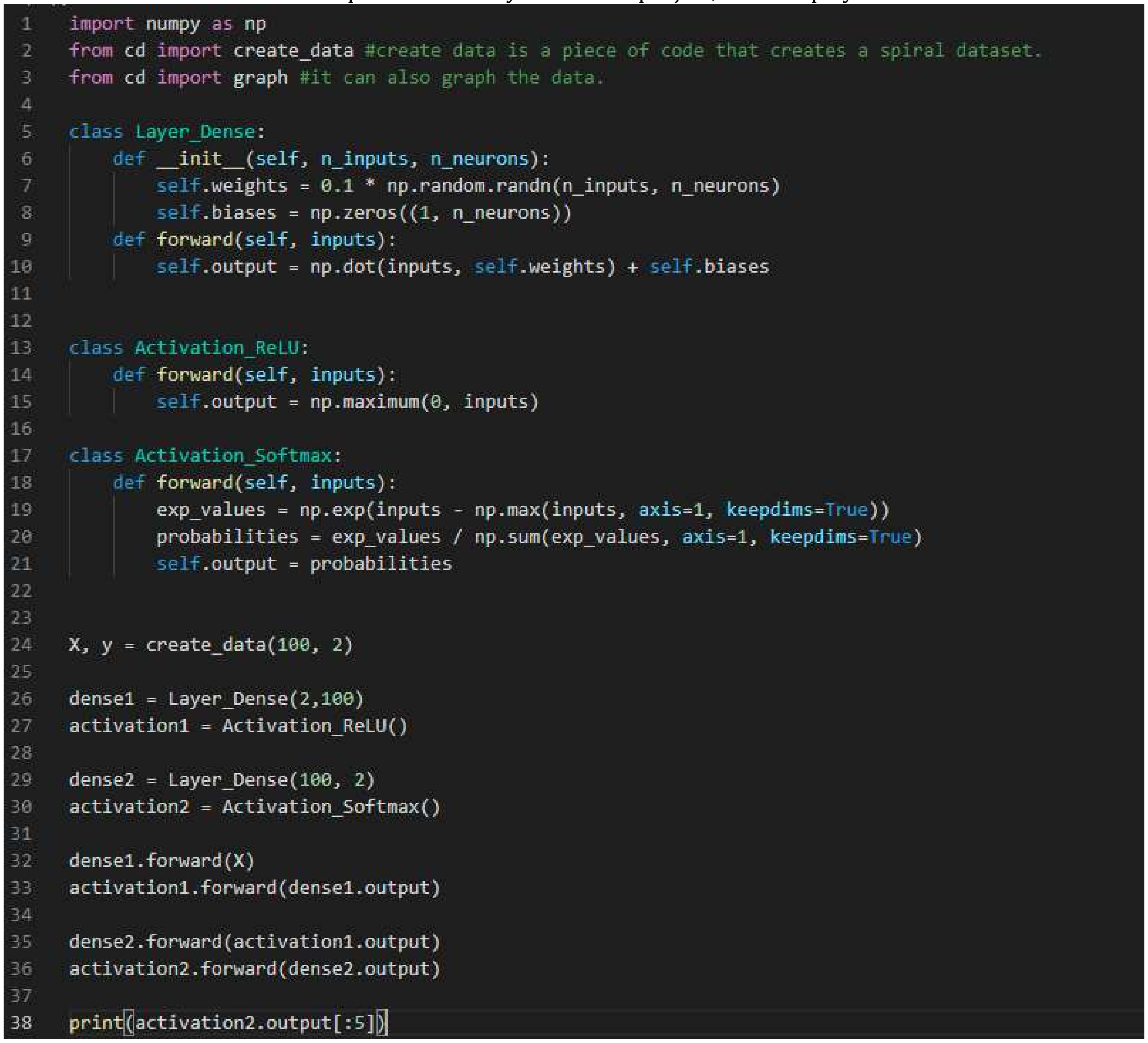


2. Suppose you initialized all the weights to zero in a two layer neural network and ran back-propagation. Explain why this will lead to a network where all the hidden nodes are identical, i.e., produce identical outputs. How can you avoid this problem?

   If all weights are initialized to zero, then all the gradient descent updates will be the same and therefor lead to the identical outputs. One way to avoid this issue is by setting all the weights to a random value.

3. (Programming) Now implement a two-layer neural network for binary classification with a hidden layer of H=100 units and K=2 output units, one for each class. Assume that each hidden unit is connected to all the inputs and a bias feature. You can use the ReLU and Sigmoid functions as the link/activation function.

   Just hand-in your code, and you don't have to play with data for now. (But you are encouraged to play with some toy data you find on your own to validate if your implementation is correct or not). You can reuse the code to this question in your mini-project, and play with data then.



```python
import numpy as np
from cd import create_data #create data is a piece of code that creates a spiral dataset.
from cd import graph #it can also graph the data.

class Layer_Dense:
    def __init__(self, n_inputs, n_neurons):
        self.weights = 0.1 * np.random.randn(n_inputs, n_neurons)
        self.biases = np.zeros((1, n_neurons))
    def forward(self, inputs):
        self.output = np.dot(inputs, self.weights) + self.biases


class Activation_ReLU:
    def forward(self, inputs):
        self.output = np.maximum(0, inputs)

class Activation_Softmax:
    def forward(self, inputs):
        exp_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))
        probabilities = exp_values / np.sum(exp_values, axis=1, keepdims=True)
        self.output = probabilities


X, y = create_data(100, 2)

dense1 = Layer_Dense(2,100)
activation1 = Activation_ReLU()

dense2 = Layer_Dense(100, 2)
activation2 = Activation_Softmax()

dense1.forward(X)
activation1.forward(dense1.output)

dense2.forward(activation1.output)
activation2.forward(dense2.output)

print(activation2.output[:5])
```