Result Report: Coding#1 Perceptron

To generate weights that fit the requirements I used random floats between -1 and 1. My activation function is a step function with θ = 0.

My code delivers in the example console output the following weights after 12 attempts:

Bias:	-0.6453137998252141
W_1:	0.4472889751090914
W_2:	0.26627829418455784

I calculated the weights multiple times, and one thing all possible results have in common is that the bias is always negative and W_1 and W_2 are always positive. This was also true for broader symmetric intervals. So, to get out of the interval quick a good start would be to generate the bias from a negative interval and the weights from a positive interval.

Source Code

```
from itertools import product
import random

def binary_combinations(n):
    '''returns list of all binary combinations necessary for the AND-Gate'''
    return list(product([0, 1], repeat=n))

def initilize_weight(dimension):
    '''Returns the random weights. Last weight in the list is the bias.'''
    return [random.uniform(-1, 1) for _ in range(dimension+1)]


def check_and_gate_result(gate_result):
    '''Returns the amount of correct and-gate classifications for a given gate
result'''
    correct_results = 0
    for i in range(len(gate_result)):
        if i == len(gate_result) - 1 and gate_result[i] == 1:
            correct_results += 1
        elif i != len(gate_result) - 1 and gate_result[i] == 0:
            correct_results += 1
    return correct_results


def and_gate(input, weights):
    # weights has the size of len(input) + 1 and the last entry is the bias
    sum = weights[len(input)]
    for i in range(len(input)):
        sum += input[i]*weights[i]
    # using the step function as the activation function
    if sum > 0:
        return 1
    return 0

# Main Loop from the excercise
def main(n):
    inputs = binary_combinations(n)
    print(inputs)
    result = []
    count = 0
    while check_and_gate_result(result) != len(inputs):
        result = []
        weights = initilize_weight(n)
        for i in range(len(inputs)):
            result.append(and_gate(inputs[i], weights))
        count += 1
        print("Try: " + str(count) + "\nCorrect classifications: " +
              str(check_and_gate_result(result)))
    print(weights)


main(2)
```

Console output:

```
[(0, 0), (0, 1), (1, 0), (1, 1)]

Try: 1

Correct classifications: 2

Try: 2

Correct classifications: 3

Try: 3

Correct classifications: 2

Try: 4

Correct classifications: 3

Try: 5

Correct classifications: 1

Try: 6

Correct classifications: 1

Try: 7

Correct classifications: 3

Try: 8

Correct classifications: 2

Try: 9

Correct classifications: 3

Try: 10

Correct classifications: 3

Try: 11

Correct classifications: 3

Try: 12

Correct classifications: 4

[0.4472889751090914, 0.26627829418455784, -0.6453137998252141]
```