

Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 8
з дисципліни «Системи штучного інтелекту»
на тему «Штучні нейронні мережі. Моделювання логічних функцій»
Варіант № 8

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірила:
Яковенко Л. В.

Київ 2020

1. МЕТА РОБОТИ

Отримати початкові навички по створенню штучних нейронних мереж, що здатні виконувати прості логічні функції.

2. ХІД РОБОТИ

За завданням варіанту необхідно написати програму, яка моделюватиме логічні функції І, АБО, НЕ та виключне АБО. Крім того, щоб отримати максимальний бал, необхідно також змоделювати логічну функцію, яка має задану таблицю істинності (табл. 1).

Табл. 1: Таблиці істинності цільової логічної функції

x_1	x_2	x_3	$Y(x_1, x_2, x_3)$
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	1

Щоб змоделювати звичайні логічні функції, досить використати параметри нейронів із завдання на лабораторну роботу. Однак для заданої функції необхідно визначити параметри нейронної мережі самостійно.

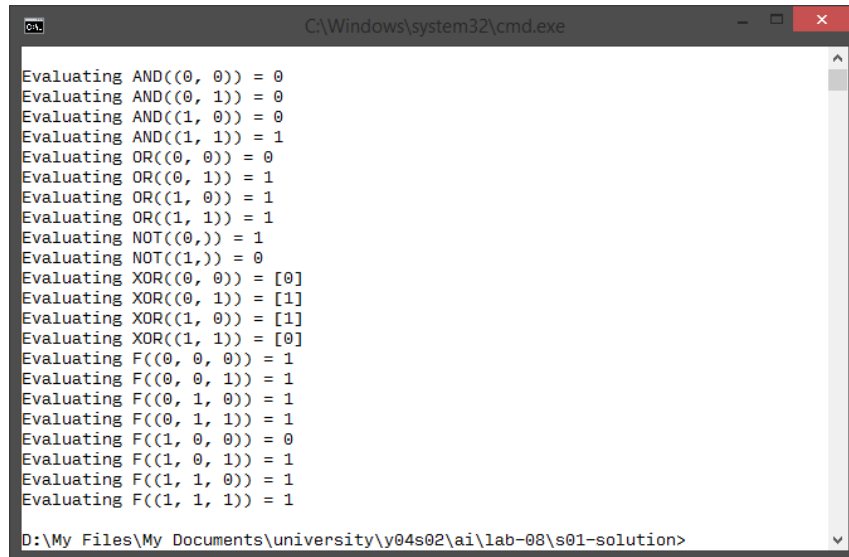
Так як функція задана неповною таблицею, її значення для інших наборів вхідних даних нас не цікавлять і можуть бути довільними. Тому спробуємо змоделювати її за допомогою одного нейрона з 3 входами. Щоб змоделювати логічну функцію, необхідно визначити ваги входів w_i та порогове значення T ; залишимо лінійну функцію активації: $F(S) = 0$, якщо $S < 0$, інакше $F(S) = 1$.

Щоб знайти ці значення, припустимо, що порогове значення $T = 0$. Тоді бажані значення нейрона із таблиці істинності можна представити у вигляді системи нерівностей:

$$\begin{aligned} \begin{cases} F(S_1) = 1 \\ F(S_2) = 1 \\ F(S_3) = 0 \\ F(S_4) = 1 \end{cases} &= \begin{cases} S_1 \geq T \\ S_2 \geq T \\ S_3 < T \\ S_4 \geq T \end{cases} = \begin{cases} w_1x_{12} + w_2x_{12} + w_3x_{13} \geq 0 \\ w_1x_{21} + w_2x_{22} + w_3x_{23} \geq 0 \\ w_1x_{31} + w_2x_{32} + w_3x_{33} < 0 \\ w_1x_{41} + w_2x_{42} + w_3x_{43} \geq 0 \end{cases} = \begin{cases} 0w_1 + 0w_2 + 0w_3 \geq 0 \\ 0w_1 + 1w_2 + 0w_3 \geq 0 \\ 1w_1 + 0w_2 + 0w_3 < 0 \\ 1w_1 + 1w_2 + 1w_3 \geq 0 \end{cases} \\ &= \begin{cases} w_2 \geq 0 \\ 1w_1 < 0 \\ 1w_1 + 1w_2 + 1w_3 \geq 0 \end{cases} \end{aligned}$$

Розв'язавши систему нерівностей, отримаємо: $w_1 \geq 0$, $w_2 < 0$, $w_3 \geq w_2 - w_1$. Підставимо будь-які зручні значення, які задовольняють нерівності, і отримаємо шукані ваги $w_1 = 1$, $w_2 = -1$, $w_3 = 1$.

Визначивши необхідні значення, розробляємо програму, що моделюватиме штучну нейронну мережу. Вона складатиметься з власне моделі нейронної мережі (лістинг А.1) та основного модуля (лістинг А.2). Запускаємо розроблену програму і спостерігаємо результат (рис. 1).



```
C:\Windows\system32\cmd.exe

Evaluating AND((0, 0)) = 0
Evaluating AND((0, 1)) = 0
Evaluating AND((1, 0)) = 0
Evaluating AND((1, 1)) = 1
Evaluating OR((0, 0)) = 0
Evaluating OR((0, 1)) = 1
Evaluating OR((1, 0)) = 1
Evaluating OR((1, 1)) = 1
Evaluating NOT((0,)) = 1
Evaluating NOT((1,)) = 0
Evaluating XOR((0, 0)) = [0]
Evaluating XOR((0, 1)) = [1]
Evaluating XOR((1, 0)) = [1]
Evaluating XOR((1, 1)) = [0]
Evaluating F((0, 0, 0)) = 1
Evaluating F((0, 0, 1)) = 1
Evaluating F((0, 1, 0)) = 1
Evaluating F((0, 1, 1)) = 1
Evaluating F((1, 0, 0)) = 0
Evaluating F((1, 0, 1)) = 1
Evaluating F((1, 1, 0)) = 1
Evaluating F((1, 1, 1)) = 1

D:\My Files\My Documents\university\y04s02\ai\lab-08\s01-solution>
```

Рис. 1: Результат роботи розробленої програми

Як видно, розроблена програма коректно працює і обчислює правильні результати як для логічних функцій, так і для довільної функції, заданої таблицею істинності.

3. ВИСНОВОК

Виконуючи дану лабораторну роботу, ми отримали початкові навички по створенню штучних нейронних мереж, що здатні виконувати прості логічні функції.

А. ЛІСТИНГ РОЗРОБЛЕНОЇ ПРОГРАМИ

Лістинг А.1: Файл `neuron.py`: модель нейронної мережі

```
1 import typing as typ
2 import logging
3
4 logger = logging.basicConfig(level=logging.WARNING)
```

```

5
6
7 def act_func_linear(S: int, threshold: float):
8     if S < threshold:
9         return 0
10    else:
11        return 1
12
13
14 def act_func_logistic(S: int, threshold: float):
15     if S < threshold:
16         return 0
17    else:
18        return 1
19
20
21 class MPNeuron(object):
22
23     def __init__(
24         self,
25         weights: typ.List[int],
26         threshold: int,
27         act_func: typ.Callable[[int], bool] = act_func_linear,
28         name: str = "",
29     ):
30         self.weights = weights
31         self.act_func = act_func
32         self.threshold = threshold
33         self.name = name
34
35     def __str__(self):
36         return self.name
37
38     @staticmethod
39     def _weighted_sum(weights, inputs):
40         logging.debug(
41             "Weights = {}".format(weights)
42             "Inputs = {}".format(inputs)
43         )
44         weighted_inputs = [x_n * w_n for x_n, w_n in zip(inputs, weights)]
45         logging.debug(
46             "Weighted = {}".format(weighted_inputs)
47         )
48         S = sum(weighted_inputs)
49         logging.debug("S = {}".format(S))
50         return S
51

```

```

52
53     def eval(self, *inputs):
54         S = self._weighted_sum(self.weights, inputs)
55         res = self.act_func(S, self.threshold)
56         return res
57
58
59 class ForwardPropagationNeuralNet(object):
60
61     @classmethod
62     def from_list(
63         cls,
64         layer_list: typ.List[typ.List[typ.Tuple]],
65         name: str = ""
66     ):
67         """Constructs a neural network from its' neurons specifications.
68         """
69         layers = []
70         for layer in layer_list:
71             layer_neurons = []
72             for weights, threshold in layer:
73                 n = MPNeuron(weights=weights, threshold=threshold)
74                 layer_neurons.append(n)
75
76             layers.append(layer_neurons)
77
78         return cls(layers=layers, name=name)
79
80     def __str__(self):
81         return self.name
82
83     def __init__(self, layers: typ.List[typ.List[MPNeuron]], name: str =
84         ↵   ""):
85         self.layers = layers
86         self.name = name
87
88     def eval(self, *inputs):
89         """Evaluates the result of the neural network on given inputs."""
90         next_inputs = [*inputs]
91         for layer in self.layers:
92             next_inputs = [neuron.eval(*next_inputs) for neuron in layer]
93
94         return next_inputs

```

Лістинг А.2: Файл `main.py`: основний модуль

```
1  import itertools as it
2  import neuron as n
3
4  XOR_LAYER_LIST = [
5      [
6          ([1, -1], 0.5), # Neuron 1,
7          ([-1, 1], 0.5) # Neuron 2
8      ],
9      [
10         ([1, 1], 0.5)
11     ]
12 ]
13
14  FX_NEURON = {
15      "weights": [-1, 1, 1],
16      "threshold": 0,
17  }
18
19  def test_neuron(n, input_count: int = 2):
20      for inputs in it.product([0, 1], repeat=input_count):
21          res = n.eval(*inputs)
22          print(
23              "Evaluating {}({}) = {}".format(n, inputs, res)
24          )
25
26
27  def main(*args, **kwargs):
28      # Task 1: model an AND logic gate
29      and_func = n.MPNeuron(
30          weights=[1, 1],
31          threshold=1.5,
32          name="AND",
33      )
34      test_neuron(and_func)
35
36      # Task 2: model an OR logic gate
37      or_func = n.MPNeuron(
38          weights=[1, 1],
39          threshold=0.5,
40          name="OR",
41      )
42      test_neuron(or_func)
43
44
45      # Task 3: model a NOT logic gate
46      not_func = n.MPNeuron(
```

```

47         weights=[-1.5],
48         threshold=-1,
49         name="NOT",
50     )
51     test_neuron(not_func, input_count=1)
52
53     # Task 4: model a XOR logic gate
54     layer_list = [
55         [
56             ([1, -1], 0.5), # Neuron 1,
57             ([-1, 1], 0.5) # Neuron 2
58         ],
59         [
60             ([1, 1], 0.5)
61         ]
62     ]
63     xor_nn = n.ForwardPropagationNeuralNet.from_list(
64         layer_list=layer_list, name="XOR"
65     )
66     test_neuron(xor_nn)
67
68     # Task 5: Model F(x_1, x_2, x_3)
69     fx_neuron = n.MPNeuron(*FX_NEURON, name="F")
70     test_neuron(fx_neuron, input_count=3)
71
72     if __name__ == "__main__":
73         main()

```
