

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

Лабораторна робота № 9  
з дисципліни «Системи штучного інтелекту»  
на тему «Штучні нейронні мережі. Прогнозування часових рядів»  
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 прогнозувати часові ряди.

## 2. ХІД РОБОТИ

За завданням варіанту необхідно написати програму, яка моделюватиме штучну нейронну мережу, що прогнозує часові ряди за декількома попередніми значеннями. Для варіанта виданий певний часовий ряд (табл. 1).

Табл. 1: Часовий ряд для заданого варіанту

$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$	$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$
1,19	5,61	0,89	6,00	1,04	5,98	0,03	6,00	1,83	4,23	0,60	4,15	0,13	5,01	1,87

Щоб вирішити поставлену задачу, використаємо нейронну мережу, яка складатиметься з одного шару. Цей шар міститиме 1 вихідний нейрон з 3 входами (рис. 1).

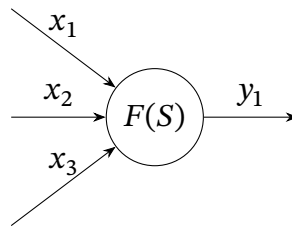


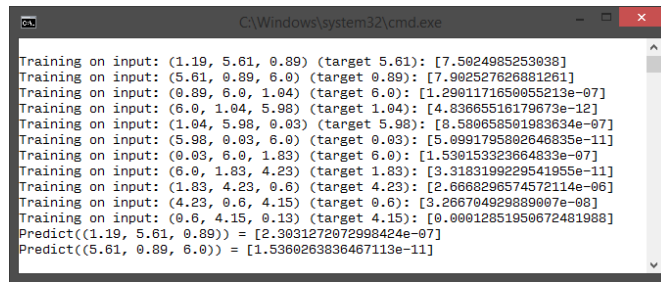
Рис. 1: Топологія використаної нейронної мережі

Розроблюємо програму для моделювання обраної нейронної мережі з можливістю зворотного поширення для моделювання заданого часового ряду. Вона складатиметься з двох модулів: моделі нейронної мережі (рис. А.1) та основного модуля (рис. А.2). Запускаємо розроблену програму і спостерігаємо за результатом (рис. 2).

Програма запускається, коректно працює та надає реалістичні результати для вхідних тренувальних і тестових наборів даних.

## 3. ВИСНОВОК

Виконуючи дану лабораторну роботу, ми отримали початкові навички по створенню штучних нейронних мереж, що здатні прогнозувати часові ряди.



```
C:\Windows\system32\cmd.exe
Training on input: (1.19, 5.61, 0.89) (target 5.61): [7.5024985253938]
Training on input: (5.61, 0.89, 6.0) (target 0.89): [7.902527626881261]
Training on input: (0.89, 6.0, 1.04) (target 6.0): [1.2901171659955213e-07]
Training on input: (6.0, 1.04, 5.96) (target 1.04): [4.89665516179673e-12]
Training on input: (1.04, 5.96, 0.03) (target 5.96): [9.580658501983634e-07]
Training on input: (5.96, 0.03, 6.0) (target 0.03): [5.0091795002646835e-11]
Training on input: (0.03, 6.0, 1.83) (target 6.0): [1.53015323864833e-07]
Training on input: (6.0, 1.83, 4.23) (target 1.83): [3.3183199229541955e-11]
Training on input: (1.83, 4.23, 0.6) (target 4.23): [2.8668296574572114e-06]
Training on input: (4.23, 0.6, 4.15) (target 0.6): [3.266704929889007e-08]
Training on input: (0.6, 4.15, 0.13) (target 4.15): [0.00012851950672481988]
Predict((1.19, 5.61, 0.89)) = [2.3031272072998424e-07]
Predict((5.61, 0.89, 6.0)) = [1.5360263836467113e-11]
```

Рис. 2: Результат роботи розробленої програми

## А. ЛІСТИНГ РОЗРОБЛЕНОЇ ПРОГРАМИ

---

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

---

```
1  import logging
2  logging.basicConfig(level=logging.WARNING)
3
4  import typing as typ
5  import math
6  import random
7
8
9  def logistic(s: float) -> float:
10     res = 1 / (1 + math.exp(-s)) * 10
11     return res
12
13
14  def logistic_prime(s: float) -> float:
15     numerator = math.exp(-s)
16     denominator = (1 + math.exp(-s))**2
17     res = numerator / denominator
18     return res
19
20
21  class Neuron(object):
22
23     @classmethod
24     def random(cls, inputs: int = 3):
25         weights = [
26             random.random()
27             for _ in range(inputs)
28         ]
29         return cls(weights=weights)
30
31     def __init__(
```

```

32         self,
33         weights: typ.List[float],
34         act_fn: typ.Callable[[float], float] = logistic,
35         act_fn_prime: typ.Callable[[float], float] = logistic_prime,
36         name: str = "",
37     ):
38         self.weights = weights
39         self.act_fn = act_fn
40         self.act_fn_prime = act_fn_prime
41         self.name = name
42
43     def __str__(self):
44         return self.name
45
46     @staticmethod
47     def _calc_weighted_sum(weights: typ.List[float], inputs):
48         logging.debug(
49             "Weights = {} \n"
50             "Inputs = {}"
51             .format(weights, inputs)
52         )
53         weighted_inputs = [w_n * x_n for w_n, x_n in zip(weights, inputs)]
54         logging.debug(
55             "Weighted = {}".format(weighted_inputs)
56         )
57         weighted_sum = sum(weighted_inputs)
58         logging.debug("weighted_sum = {}".format(weighted_sum))
59         return weighted_sum
60
61     def calc_output(self, inputs):
62         weighted_sum = self._calc_weighted_sum(self.weights, inputs)
63         output = self.act_fn(weighted_sum)
64         return output
65
66     def calc_error_pd(
67         self,
68         input_val: float,
69         output_val: float,
70         target: float,
71         weighted_sum: float,
72     ):
73         diff = output_val - target
74         derivative = self.act_fn_prime(weighted_sum)
75         error_pd = diff * derivative * input_val
76         return error_pd
77
78     def calc_error_pds(self, inputs, outputs, target):

```

```

79         weighted_sum = self._calc_weighted_sum(
80             weights=self.weights,
81             inputs=inputs
82         )
83
84         error_pds = []
85         for input_val, output_val in zip(inputs, outputs):
86             delta = self.calc_error_pd(
87                 input_val=input_val,
88                 output_val=output_val,
89                 target=target,
90                 weighted_sum=weighted_sum,
91             )
92             error_pds.append(delta)
93
94         logging.debug("Error_pds = {}".format(error_pds))
95         return error_pds
96
97     def update_weights(self, weight_delta):
98         for idx, w in enumerate(self.weights):
99             self.weights[idx] += weight_delta
100
101
102 class NeuralNetwork(object):
103
104     @classmethod
105     def random(cls, inputs=3):
106         neurons = [Neuron.random()]
107         obj = cls(neurons=neurons)
108         return obj
109
110     def __init__(
111         self,
112         neurons: typ.List[Neuron],
113         learning_speed: float = 0.35,
114     ):
115         self.neurons = neurons
116         self.learning_speed = learning_speed
117
118     def feed_forward(self, inputs):
119         outputs = []
120         for neuron in self.neurons:
121             out = neuron.calc_output(inputs)
122             outputs.append(out)
123
124         return outputs
125

```

```

126     def feed_backward(self, inputs, output, target):
127         for n in self.neurons:
128             error_pds = n.calc_error_pds(
129                 inputs=inputs,
130                 outputs=output,
131                 target=target
132             )
133             weight_deltas = [-self.learning_speed * e for e in error_pds]
134             weight_delta_avg = sum(weight_deltas) / len(weight_deltas)
135             n.update_weights(weight_delta_avg)
136
137     def train(self, training_input, target):
138         output = self.feed_forward(training_input)
139         logging.debug(
140             "Input: {} Target: {} Output: {}".format(
141                 training_input, target, output
142             )
143         )
144         self.feed_backward(training_input, output, target)
145         return output
146
147     def predict(self, inputs):
148         return self.feed_forward(inputs)

```

---

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

---

```

1  import itertools as it
2  import neuron as n
3
4
5  DATASET = [1.19, 5.61, 0.89, 6.00, 1.04, 5.98, 0.03, 6.00, 1.83, 4.23,
6             0.60, 4.15, 0.13, 5.01, 1.87]
7
8  def group_n_wise(iterable, n=2):
9      t = it.tee(iterable, n)
10
11     for idx, _ in enumerate(t, start=1):
12         for subset in t[idx:]:
13             next(subset, None)
14
15     return zip(*t)
16
17
18 def main():
19
20     nn = n.NeuralNetwork.random(inputs=3,)

```

```

21
22     # Split the dataset into train and test iterators
23     # These iterators are 3-wise subsets of the original dataset
24     train = group_n_wise(DATASET[:13], 3)
25     test = group_n_wise(DATASET[:13], 3)
26     # Split the target values too, since they come from the original
       ↪ dataset
27     train_target, test_target = DATASET[1:13], DATASET[13:]
28
29     for i, target in zip(train, train_target):
30         out = nn.train(i, target)
31         print(
32             "Training on input: {} (target {}): {}".format(i, target, out)
33         )
34
35
36     for i, target in zip(test, test_target):
37         prediction = nn.predict(i)
38         print(
39             "Predict({}) = {}".format(i, prediction)
40         )
41
42
43
44 if __name__ == "__main__":
45     main()

```

---