

Лабораторна робота 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

Мета: *використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідити деякі типи нейронних мереж.*

Хід роботи

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

```
import random

class RNN:
    # A many-to-one Vanilla Recurrent Neural Network.
    def __init__(self, input_size, output_size, hidden_size=64):
        # Weights
        self.Whh = randn(hidden_size, hidden_size) / 1000
        self.Wxh = randn(hidden_size, input_size) / 1000
        self.Why = randn(output_size, hidden_size) / 1000
        # Biases
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))

    def forward(self, inputs):
        """
        Perform a forward pass of the RNN using the given inputs.
        Returns the final output and hidden state.
        - inputs is an array of one hot vectors with shape (input_size, 1).
        """
        h = np.zeros((self.Whh.shape[0], 1))
        self.last_inputs = inputs
        self.last_hs = {0: h}
        # Perform each step of the RNN
        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
            self.last_hs[i + 1] = h
        # Compute the output
        y = self.Why @ h + self.by
        return y, h

    def backprop(self, d_y, learn_rate=2e-2):
        """
        Perform a backward pass of the RNN.
        - d_y (dL/dy) has shape (output_size, 1).
        - learn_rate is a float.
        """
        n = len(self.last_inputs)
        # Calculate dL/dWhy and dL/dby.
        d_Why = d_y @ self.last_hs[n].T
        d_by = d_y
        # Initialize dL/dWhh, dL/dWxh, and dL/dbh to zero.
        d_Whh = np.zeros(self.Whh.shape)
        d_Wxh = np.zeros(self.Wxh.shape)
        d_bh = np.zeros(self.bh.shape)
```

					ДУ «Житомирська політехніка».20.121.12.						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Анкудевич Д.Р.						Літ.	Арк.	Аркушів	
Перевір.		Філіпов В.О.								1	11
Керівник								ФІКТ Гр. ІПЗк-20-1			
Н. контр.											
Зав. каф.											

```

12 # Calculate dL/dh for the last h.
13 # dL/dh = dL/dy * dy/dh
14 d_h = self.Why.T @ d_y
15 # Backpropagate through time.
16 for t in reversed(range(n)):
17     # An intermediate value: dL/dh * (1 - h^2)
18     temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)
19     # dL/db = dL/dh * (1 - h^2)
20     d_bh += temp
21     # dL/dWhh = dL/dh * (1 - h^2) * h_{t-1}
22     d_Whh += temp @ self.last_hs[t].T
23     # dL/dWxh = dL/dh * (1 - h^2) * x
24     d_Wxh += temp @ self.last_inputs[t].T
25     # Next dL/dh = dL/dh * (1 - h^2) * Whh
26     d_h = self.Whh @ temp
27 # Clip to prevent exploding gradients.
28 for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
29     np.clip(d, -1, 1, out=d)
30 # Update weights and biases using gradient descent.
31 self.Whh -= learn_rate * d_Whh
32 self.Wxh -= learn_rate * d_Wxh
33 self.Why -= learn_rate * d_Why
34 self.bh -= learn_rate * d_bh
35 self.by -= learn_rate * d_by
36 from data import train_data, test_data
37 # Create the vocabulary.
38 vocab = list(set([w for text in train_data.keys() for w in text.split(' ')]))
39 vocab_size = len(vocab)
40 print('%d unique words found' % vocab_size)
41 # Assign indices to each word.
42 word_to_idx = {w: i for i, w in enumerate(vocab)}
43 idx_to_word = {i: w for i, w in enumerate(vocab)}
44 # print(word_to_idx['good'])
45 # print(idx_to_word[0])
46 def createInputs(text):
47     """

```

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		2

```

        # Update weights and biases using gradient descent.
        self.Whh -= learn_rate * d_Whh
        self.Wxh -= learn_rate * d_Wxh
        self.Why -= learn_rate * d_Why
        self.bh -= learn_rate * d_bh
        self.by -= learn_rate * d_by
    from data import train_data, test_data
    # Create the vocabulary.
    vocab = list(set([w for text in train_data.keys() for w in text.split(' ')]))
    vocab_size = len(vocab)
    print('%d unique words found' % vocab_size)
    # Assign indices to each word.
    word_to_idx = {_w: i for i, w in enumerate(vocab)}
    idx_to_word = {_i: w for i, w in enumerate(vocab)}
    # print(word_to_idx['good'])
    # print(idx_to_word[0])
    def createInputs(text):
        """
        Returns an array of one-hot vectors representing the words in the input text string.
        - text is a string
        - Each one-hot vector has shape (vocab_size, 1)
        """
        inputs = []
        for w in text.split(' '):
            v = np.zeros((vocab_size, 1))
            v[word_to_idx[w]] = 1
            inputs.append(v)
        return inputs
    def softmax(xs):
        # Applies the Softmax Function to the input array.

```

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		3

```

78     Returns an array of one-hot vectors representing the words in the input text string.
79     - text is a string
80     - Each one-hot vector has shape (vocab_size, 1)
81     '''
82     inputs = []
83     for w in text.split(' '):
84         v = np.zeros((vocab_size, 1))
85         v[word_to_idx[w]] = 1
86         inputs.append(v)
87     return inputs
88 def softmax(xs):
89     # Applies the Softmax Function to the input array.
90     return np.exp(xs) / sum(np.exp(xs))
91 # Initialize our RNN!
92 rnn = RNN(vocab_size, 2)
93 def processData(data, backprop=True):
94     '''
95     Returns the RNN's loss and accuracy for the given data.
96     - data is a dictionary mapping text to True or False.
97     - backprop determines if the backward phase should be run.
98     '''
99     items = list(data.items())
100    random.shuffle(items)
101    loss = 0
102    num_correct = 0
103    for x, y in items:
104        inputs = createInputs(x)
105        target = int(y)
106        # Forward
107        out, _ = rnn.forward(inputs)
108        probs = softmax(out)
109        # Calculate loss / accuracy
110        loss -= np.log(probs[target])
111        num_correct += int(np.argmax(probs) == target)

```

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

```

112     if backprop:
113         # Build dL/dy
114         d_L_d_y = probs
115         d_L_d_y[target] -= 1
116         # Backward
117         rnn.backprop(d_L_d_y)
118     return loss / len(data), num_correct / len(data)
119 # Training loop
120 for epoch in range(1000):
121     train_loss, train_acc = processData(train_data)
122     if epoch % 100 == 99:
123         print('--- Epoch %d' % (epoch + 1))
124         print('Train:\tLoss %.3f | Accuracy: %.3f' % (train_loss, train_acc))
125         test_loss, test_acc = processData(test_data, backprop=False)
126         print('Test:\tLoss %.3f | Accuracy: %.3f' % (test_loss, test_acc))
127 import numpy as np
128 from numpy.random import randn
129 class RNN:
130     # A many-to-one Vanilla Recurrent Neural Network.
131     def __init__(self, input_size, output_size, hidden_size=64):
132         # Weights
133         self.Whh = randn(hidden_size, hidden_size) / 1000
134         self.Wxh = randn(hidden_size, input_size) / 1000
135         self.Why = randn(output_size, hidden_size) / 1000
136         # Biases
137         self.bh = np.zeros((hidden_size, 1))
138         self.by = np.zeros((output_size, 1))
139     def forward(self, inputs):
140         """
141         Perform a forward pass of the RNN using the given inputs.
142         Returns the final output and hidden state.
143         - inputs is an array of one hot vectors with shape (input_size, 1).
144         """
145         h = np.zeros((self.Whh.shape[0], 1))
146         self.last_inputs = inputs
147         self.last_hs = {0: h}
148         # Perform each step of the RNN
149         for i, x in enumerate(inputs):
150             h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
151             self.last_hs[i + 1] = h

```

		Анкудевич Д.Р.			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філінов В.О.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

```

152     # Compute the output
153     y = self.Why @ h + self.by
154     return y, h
155     def backprop(self, d_y, learn_rate=2e-2):
156         """
157         Perform a backward pass of the RNN.
158         - d_y (dL/dy) has shape (output_size, 1).
159         - learn_rate is a float.
160         """
161         n = len(self.last_inputs)
162         # Calculate dL/dWhy and dL/dby.
163         d_Why = d_y @ self.last_hs[n].T
164         d_by = d_y
165         # Initialize dL/dWhh, dL/dWxh, and dL/dbh to zero.
166         d_Whh = np.zeros(self.Whh.shape)
167         d_Wxh = np.zeros(self.Wxh.shape)
168         d_bh = np.zeros(self.bh.shape)
169         # Calculate dL/dh for the last h.
170         # dL/dh = dL/dy * dy/dh
171         d_h = self.Why.T @ d_y
172         # Backpropagate through time.
173         for t in reversed(range(n)):
174             # An intermediate value: dL/dh * (1 - h^2)
175             temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)
176             # dL/db = dL/dh * (1 - h^2)
177             d_bh += temp
178             # dL/dWhh = dL/dh * (1 - h^2) * h_{t-1}
179             d_Whh += temp @ self.last_hs[t].T
180             # dL/dWxh = dL/dh * (1 - h^2) * x
181             d_Wxh += temp @ self.last_inputs[t].T
182             # Next dL/dh = dL/dh * (1 - h^2) * Whh
183             d_h = self.Whh @ temp
184         # Clip to prevent exploding gradients.
185         for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
186             np.clip(d, -1, 1, out=d)
187         # Update weights and biases using gradient descent.
188         self.Whh -= learn_rate * d_Whh
189         self.Wxh -= learn_rate * d_Wxh
190         self.Why -= learn_rate * d_Why
191         self.bh -= learn_rate * d_bh
192         self.Whh -= learn_rate * d_Whh
193         self.Wxh -= learn_rate * d_Wxh
194         self.Why -= learn_rate * d_Why
195         self.bh -= learn_rate * d_bh
196         self.by -= learn_rate * d_by

```

Рис 1. Лістинг коду файла LR_6_task_1.py

```
18 unique words found
--- Epoch 100
Train: Loss 0.689 | Accuracy: 0.552
Test: Loss 0.697 | Accuracy: 0.500
--- Epoch 200
Train: Loss 0.671 | Accuracy: 0.621
Test: Loss 0.721 | Accuracy: 0.500
--- Epoch 300
Train: Loss 0.566 | Accuracy: 0.655
Test: Loss 0.618 | Accuracy: 0.650
--- Epoch 400
Train: Loss 0.393 | Accuracy: 0.828
Test: Loss 0.716 | Accuracy: 0.550
--- Epoch 500
Train: Loss 0.300 | Accuracy: 0.914
Test: Loss 0.582 | Accuracy: 0.700
--- Epoch 600
Train: Loss 0.259 | Accuracy: 0.914
Test: Loss 0.372 | Accuracy: 0.800
--- Epoch 700
Train: Loss 0.089 | Accuracy: 0.966
Test: Loss 0.088 | Accuracy: 1.000
--- Epoch 800
Train: Loss 0.004 | Accuracy: 1.000
Test: Loss 0.009 | Accuracy: 1.000
--- Epoch 900
Train: Loss 0.002 | Accuracy: 1.000
Test: Loss 0.004 | Accuracy: 1.000
--- Epoch 1000
Train: Loss 0.001 | Accuracy: 1.000
Test: Loss 0.003 | Accuracy: 1.000

Process finished with exit code 0
```

```
18 unique words found
--- Epoch 100
Train: Loss 0.688 | Accuracy: 0.552
Test: Loss 0.696 | Accuracy: 0.500
--- Epoch 200
Train: Loss 0.665 | Accuracy: 0.569
Test: Loss 0.720 | Accuracy: 0.500
--- Epoch 300
Train: Loss 0.129 | Accuracy: 0.948
Test: Loss 0.239 | Accuracy: 0.950
--- Epoch 400
Train: Loss 0.012 | Accuracy: 1.000
Test: Loss 0.013 | Accuracy: 1.000
--- Epoch 500
Train: Loss 0.005 | Accuracy: 1.000
Test: Loss 0.006 | Accuracy: 1.000
--- Epoch 600
Train: Loss 0.003 | Accuracy: 1.000
Test: Loss 0.004 | Accuracy: 1.000
--- Epoch 700
Train: Loss 0.002 | Accuracy: 1.000
Test: Loss 0.003 | Accuracy: 1.000
--- Epoch 800
Train: Loss 0.002 | Accuracy: 1.000
Test: Loss 0.002 | Accuracy: 1.000
--- Epoch 900
Train: Loss 0.001 | Accuracy: 1.000
Test: Loss 0.002 | Accuracy: 1.000
--- Epoch 1000
Train: Loss 0.001 | Accuracy: 1.000
Test: Loss 0.001 | Accuracy: 1.000
```

Рис 2. Результат файлу файла LR_6_task_1.py

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

Ми спостерігаємо повідомлення на рисунку 1-2 “18 unique words found” це означає, що зміна vocab тепер буде мати перелік всіх слів, які вживаються щонайменше в одному навчальному тексті. Рекурентна нейронна мережа не розрізняє слів – лише числа. Тому у словнику 18 унікальних слів, кожне буде 18-мірним унітарним вектором. І далі відбувається тренування мережі. Виведення кожної соті епохи для відслідковування прогресу

Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))

```

1 import neurolab as nl
2 import numpy as np
3 i1 = np.sin(np.arange(0, 20))
4 i2 = np.sin(np.arange(0, 20)) * 2
5 t1 = np.ones([1, 20])
6 t2 = np.ones([1, 20]) * 2
7 input = np.array([i1, i2, i1, i2]).reshape(20 * 4, 1)
8 target = np.array([t1, t2, t1, t2]).reshape(20 * 4, 1)
9 net = nl.net.newelm([-2, 2], [10, 1], [nl.trans.TanSig(), nl.trans.PureLin()])
10 net.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
11 net.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
12 net.init()
13 # Тренування мережі
14 error = net.train(input, target, epochs=500, show=100, goal=0.01)
15 # Запустить мережу
16 output = net.sim(input)
17 # Побудова графіків
18 import pylab as pl
19 pl.subplot(211)
20 pl.plot(error)
21 pl.xlabel('Epoch number')
22 pl.ylabel('Train error (default MSE)')
23 pl.subplot(212)
24 pl.plot(target.reshape(80))
25 pl.plot(output.reshape(80))
26 pl.legend(['train target', 'net output'])
27 pl.show()

```

Рис 3. Лістинг коду файла LR_6_task_2.py

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				8
Змн.	Арк.	№ докум.	Підпис	Дата		


```
C:\Users\ankud\AppData\Local\Programs\Python\Python311\python.exe C:\Users\ankud\Desktop\myAi\lab6\LR_6_task_2.py
Epoch: 100; Error: 0.249948217177879;
Epoch: 200; Error: 0.06941158107799532;
Epoch: 300; Error: 0.0777007206414863;
Epoch: 400; Error: 0.07752372455762327;
Epoch: 500; Error: 0.052260905763095764;
The maximum number of train epochs is reached
```

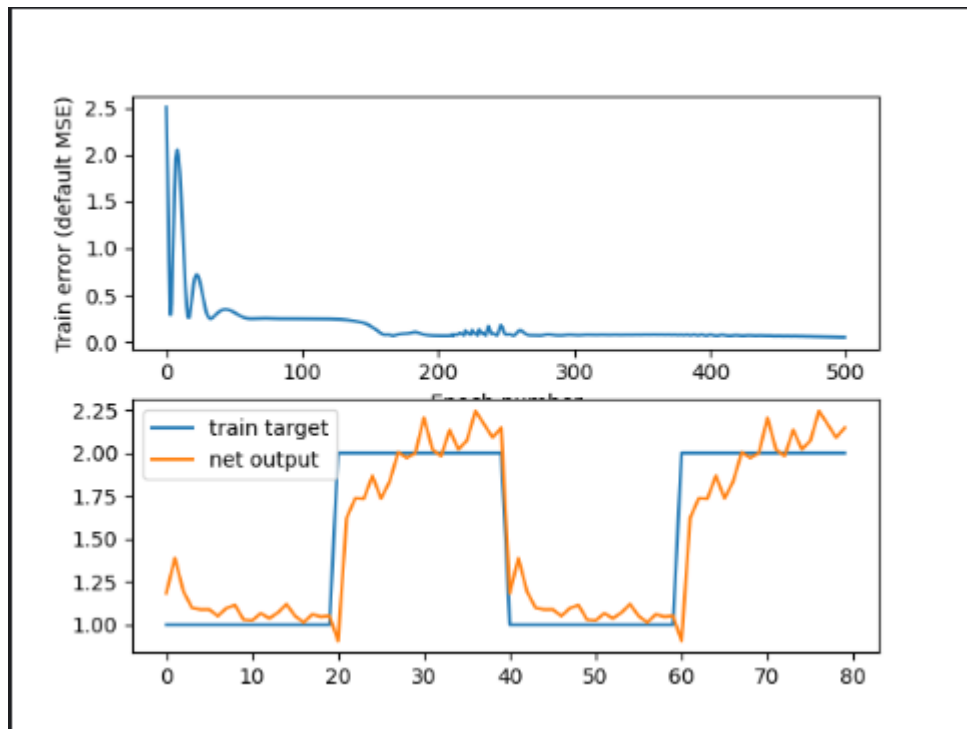


Рис 4. Результат файлу файла LR_6_task_2.py

Завдання 2.3. Дослідження нейронної мережі Хемінга (Hemming Recurrent network)

```
1 import numpy as np
2 import neurolab as nl
3 target = [[-1, 1, -1, -1, 1, -1, -1, 1, -1],
4           [1, 1, 1, 1, -1, 1, 1, -1, 1],
5           [1, -1, 1, 1, 1, 1, 1, -1, 1],
6           [1, 1, 1, 1, -1, -1, 1, -1, -1],
7           [-1, -1, -1, -1, 1, -1, -1, -1, -1]]
8 input = [[-1, -1, 1, 1, 1, 1, 1, -1, 1],
9          [-1, -1, 1, -1, 1, -1, -1, -1, -1],
10         [-1, -1, -1, -1, 1, -1, -1, 1, -1]]
11 # Створення та тренування нейромережі
12 net = nl.net.newhem(target)
13 output = net.sim(target)
14 print("Test on train samples (must be [0, 1, 2, 3, 4])")
15 print(np.argmax(output, axis=0))
16 output = net.sim([input[0]])
17 print("Outputs on recurrent cycle:")
18 print(np.array(net.layers[1].outs))
19 output = net.sim(input)
20 print("Outputs on test sample:")
21 print(output)
```

Рис 5. Лістинг коду файла LR_6_task_3.py

```
C:\Users\ankud\AppData\Local\Programs\Python\Python311\python.exe C:\Users\ankud\Desktop\myAi\lab6\LR_6_task_2.py
Epoch: 100; Error: 0.25018861995361846;
Epoch: 200; Error: 0.08281631954456395;
Epoch: 300; Error: 0.06651881082310343;
Epoch: 400; Error: 0.06128829239366555;
Epoch: 500; Error: 0.055342861089084636;
The maximum number of train epochs is reached
```

Рис 6. Результат файлу файла LR_6_task_3.py

		Анкудевич Д.Р.			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 2.4. Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop)

```

1 import numpy as np
2 import neurolab as nl
3
4 target = np.array([
5     [1, 0, 0, 0, 1,
6      1, 1, 0, 0, 1,
7      1, 0, 1, 0, 1,
8      1, 0, 0, 1, 1,
9      1, 0, 0, 0, 1],
10    [1, 1, 1, 1, 1,
11     1, 0, 0, 0, 0,
12     1, 1, 1, 1, 1,
13     1, 0, 0, 0, 0,
14     1, 1, 1, 1, 1],
15    [1, 1, 1, 1, 0,
16     1, 0, 0, 0, 1,
17     1, 1, 1, 1, 0,
18     1, 0, 0, 1, 0,
19     1, 0, 0, 0, 1],
20    [0, 1, 1, 1, 0,
21     1, 0, 0, 0, 1,
22     1, 0, 0, 0, 1,
23     1, 0, 0, 0, 1,
24     0, 1, 1, 1, 0]
25 ])
26 chars = ['N', 'E', 'R', 'O']
27
28 target[target == 0] = -1
29
30 net = nl.net.newhop(target)
31
32 output = net.sim(target)
33 print("Test on train samples:")

```

Рис 7. Лістинг коду файла LR_6_task_4.py

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

```

32 output = net.sim(target)
33 print("Test on train samples:")
34 for i in range(len(target)):
35     print(chars[i], (output[i] == target[i]).all())
36
37 print("\nTest on defaced M:")
38
39 test = np.array(
40     [0, 0, 0, 0, 0,
41      1, 1, 0, 0, 1,
42      1, 1, 1, 1, 1,
43      0, 1, 1, 1, 1,
44      1, 0, 0, 0, 1]
45 )
46 test[test == 0] = -1
47
48 out = net.sim([test])
49 print((out[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))
50

```

```

C:\Users\ankud\AppData\Local\Programs\Python\Python311\python.exe C:\Users\ankud\Desktop\myAi\lab6\LR_6_task_4.py
Test on train samples:
N True
E True
R True
O True

```

Рис 8. Результат файлу файла LR_6_task_4.py

Як бачимо, навчання пройшло правильно і мережа при невеликій кількості помилок вгадала букви правильно.

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Завдання 2.5. Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних

```

1 import numpy as np
2 import neurolab as nl
3
4 # Змінені дані
5 target = np.array([
6     [1, 0, 0, 0, 1,
7      1, 1, 0, 0, 1,
8      1, 0, 1, 0, 1,
9      1, 0, 0, 1, 1,
10     [1, 0, 0, 0, 1],
11     [1, 1, 1, 1, 1,
12     [1, 0, 0, 0, 0,
13     [1, 1, 1, 1, 1,
14     [1, 0, 0, 0, 0,
15     [1, 1, 1, 1, 1],
16     [1, 1, 1, 1, 0,
17     [1, 0, 0, 0, 1,
18     [1, 1, 1, 1, 0,
19     [1, 0, 0, 1, 0,
20     [1, 0, 0, 0, 1],
21     [0, 1, 1, 1, 0,
22     [1, 0, 0, 0, 1,
23     [1, 0, 0, 0, 1,
24     [1, 0, 0, 0, 1,

```

```

20     1, 0, 0, 0, 1],
21     [0, 1, 1, 1, 0,
22     1, 0, 0, 0, 1,
23     1, 0, 0, 0, 1,
24     1, 0, 0, 0, 1,
25     0, 1, 1, 1, 0]
26 ])
27 chars = ['A', 'N', 'K', 'O']
28
29 target[target == 0] = -1
30
31 net = nl.net.newhop(target)
32
33 output = net.sim(target)
34 print("Test on train samples:")
35 for i in range(len(target)):
36     print(chars[i], (output[i] == target[i]).all())
37
38 print("\nTest on defaced M:")
39

```

```

31 net = nl.net.newhop(target)
32
33 output = net.sim(target)
34 print("Test on train samples:")
35 for i in range(len(target)):
36     print(chars[i], (output[i] == target[i]).all())
37
38 print("\nTest on defaced M:")
39
40 # Змінені дані
41 test = np.array(
42     [0, 0, 0, 0, 0,
43     1, 1, 0, 0, 1,
44     1, 1, 1, 1, 1,
45     0, 1, 1, 1, 1,
46     1, 0, 0, 0, 1]
47 )
48 test[test == 0] = -1
49
50 out = net.sim([test])
51 print((out[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))

```

Рис 9. Лістинг коду файла LR_6_task_5.py

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

```

C:\Users\ankud\AppData\Local\Programs\Python\Python311\python.exe C:\Users\ankud\Desktop\myAi\lab6\LR_6_task_5.py
Test on train samples:
A True
N True
K True
O True

Test on defaced M:
False Sim. steps 3

Process finished with exit code 0

```

Рис 10. Результат файлу файла LR_6_task_5.py

Висново: під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python навчився досліджувати деякі типи нейронних мереж.

Посилання на гіт хаб : <https://github.com/AnkuNoName/ai>

		Анкудевич Д.Р			ДУ «Житомирська політехніка».20.121.12 – Лр6	Арк.
		Філіпов В.О.				15
Змн.	Арк.	№ докум.	Підпис	Дата		