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

```python
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)
```

Рис. 1 Код програми

```
      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)

    # Calculate dL/dh for the last h.
    # dL/dh = dL/dy * dy/dh
    d_h = self.Why.T @ d_y
```

Рис. 2 Код програми

```
    # Backpropagate through time.
    for t in reversed(range(n)):
      # An intermediate value: dL/dh * (1 - h^2)
      temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)

      # dL/db = dL/dh * (1 - h^2)
      d_bh += temp

      # dL/dWhh = dL/dh * (1 - h^2) * h_{t-1}
      d_Whh += temp @ self.last_hs[t].T

      # dL/dWxh = dL/dh * (1 - h^2) * x
      d_Wxh += temp @ self.last_inputs[t].T

      # Next dL/dh = dL/dh * (1 - h^2) * Whh
      d_h = self.Whh @ temp

    # Clip to prevent exploding gradients.
    for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
      np.clip(d, -1, 1, out=d)

    # 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
```

Рис. 3 Код програми

```python
# 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.
  return np.exp(xs) / sum(np.exp(xs))

# Initialize our RNN!
```

Рис. 4 Код програми

```python
rnn = RNN(vocab_size, 2)

def processData(data, backprop=True):
  '''
  Returns the RNN's loss and accuracy for the given data.
  - data is a dictionary mapping text to True or False.
  - backprop determines if the backward phase should be run.
  '''
  items = list(data.items())
  random.shuffle(items)

  loss = 0
  num_correct = 0

  for x, y in items:
    inputs = createInputs(x)
    target = int(y)

    # Forward
    out, _ = rnn.forward(inputs)
    probs = softmax(out)

    # Calculate loss / accuracy
    loss -= np.log(probs[target])
    num_correct += int(np.argmax(probs) == target)

    if backprop:
      # Build dL/dy
      d_L_d_y = probs
      d_L_d_y[target] -= 1
```

Рис. 5 Код програми

```
      # Backward
      rnn.backprop(d_L_d_y)

  return loss / len(data), num_correct / len(data)

# Training loop
for epoch in range(1000):
  train_loss, train_acc = processData(train_data)

  if epoch % 100 == 99:
    print('--- Epoch %d' % (epoch + 1))
    print('Train:\tLoss %.3f | Accuracy: %.3f' % (train_loss, train_acc))

    test_loss, test_acc = processData(test_data, backprop=False)
    print('Test:\tLoss %.3f | Accuracy: %.3f' % (test_loss, test_acc))

import numpy as np
from numpy.random import randn

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
```

Рис. 6 Код програми

```
    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.
    '''
```

Рис. 7 Код програми

```
    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)

    # Calculate dL/dh for the last h.
    # dL/dh = dL/dy * dy/dh
    d_h = self.Why.T @ d_y

    # Backpropagate through time.
    for t in reversed(range(n)):
      # An intermediate value: dL/dh * (1 - h^2)
      temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)

      # dL/db = dL/dh * (1 - h^2)
      d_bh += temp

      # dL/dWhh = dL/dh * (1 - h^2) * h_{t-1}
      d_Whh += temp @ self.last_hs[t].T

      # dL/dWxh = dL/dh * (1 - h^2) * x
      d_Wxh += temp @ self.last_inputs[t].T

      # Next dL/dh = dL/dh * (1 - h^2) * Whh
```

Рис. 8 Код програми

```
    d_h = self.Whh @ temp

  # Clip to prevent exploding gradients.
  for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
    np.clip(d, -1, 1, out=d)

  # 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
```

Рис. 9 Код програми

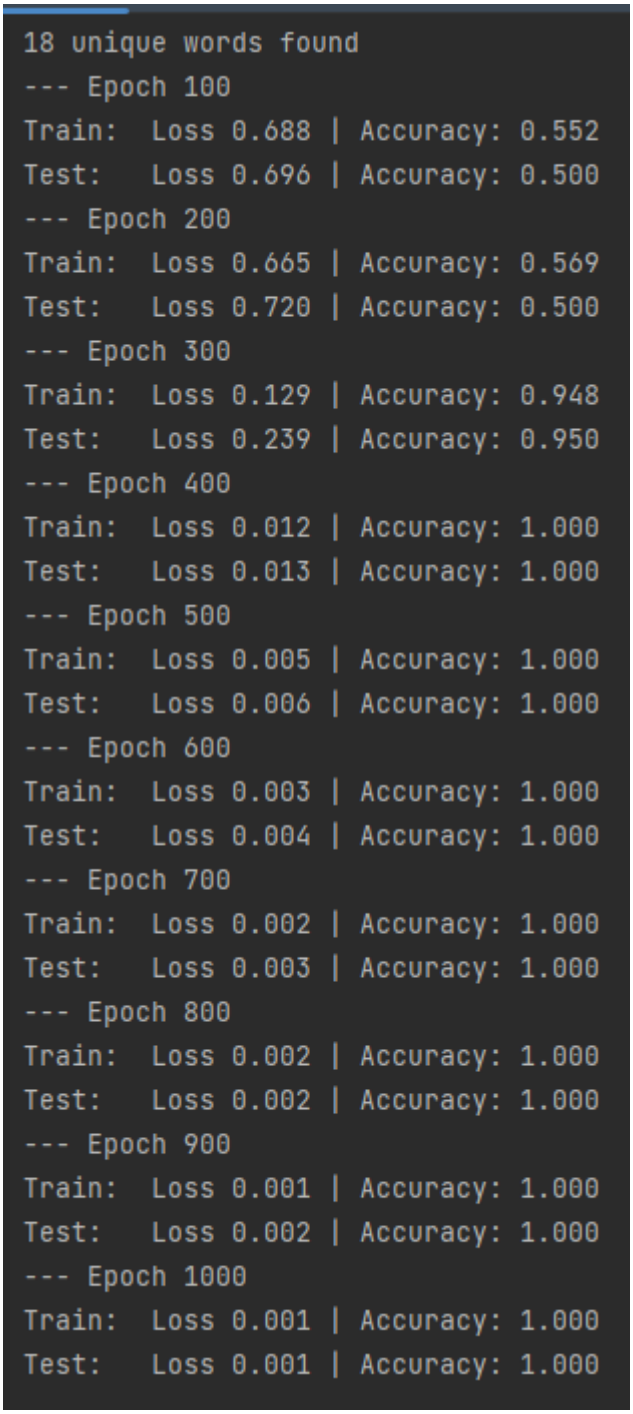

Рис. 10 Виконання файлу main.py

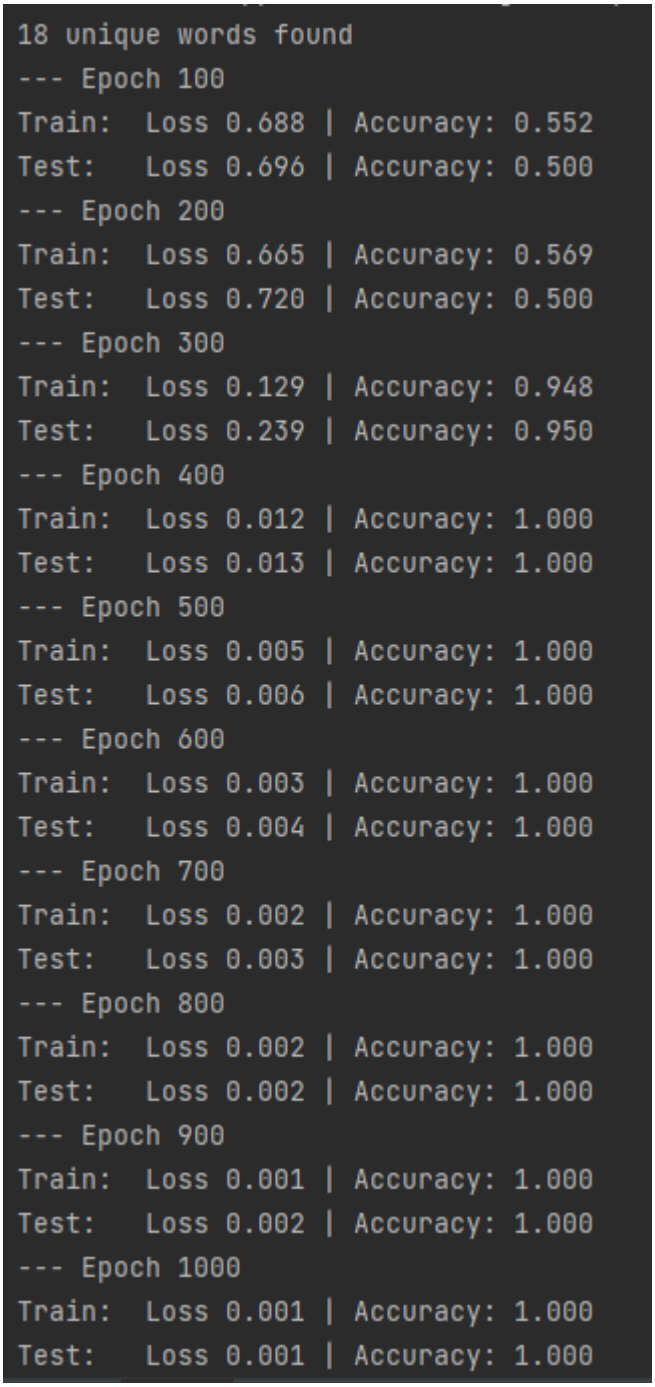

Рис. 11 Виконання файлу LR_6_task_1.py

Висновок: На рисунку 10 -11 ми бачимо виведенне повідомлення “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 сотої епохи для відслідковування прогресу.

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

```
import neurolab as nl
import numpy as np

i1 = np.sin(np.arange(0, 20))
i2 = np.sin(np.arange(0, 20)) * 2
t1 = np.ones([1, 20])
t2 = np.ones([1, 20]) * 2
input = np.array([i1, i2, i1, i2]).reshape(20 * 4, 1)
target = np.array([t1, t2, t1, t2]).reshape(20 * 4, 1)
net = nl.net.newelm([[-2, 2]], [10, 1], [nl.trans.TanSig(), nl.trans.PureLin()])
net.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
net.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
net.init()
# Тренування мережі
error = net.train(input, target, epochs=500, show=100, goal=0.01)
# Запустіть мережу
output = net.sim(input)
# Побудова графіків
import pylab as pl
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('Train error (default MSE)')

pl.subplot(212)
pl.plot(target.reshape(80))
pl.plot(output.reshape(80))
pl.legend(['train target', 'net output'])
pl.show()
```

Рис. 12 Код програми

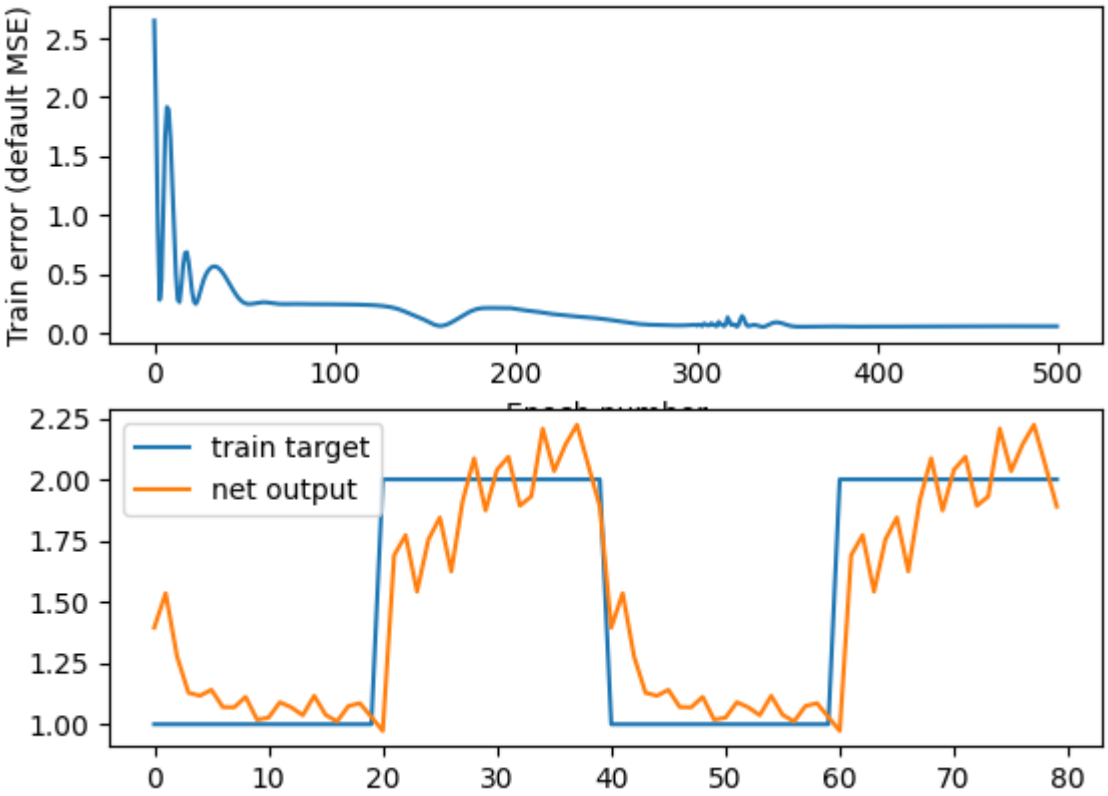

Рис. 13 Виконання програми

```
Epoch: 100; Error: 0.24787839011370708;
Epoch: 200; Error: 0.21098065965151394;
Epoch: 300; Error: 0.07392123798190661;
Epoch: 400; Error: 0.05908399231418123;
Epoch: 500; Error: 0.06151130691352045;
The maximum number of train epochs is reached
```

Рис. 14 Виконання програми

Висновок: Під час виконання 2 завдання імпортував neurolab та numpy, створив модель сигналу для навчання мережі, створив мережу з двома прошарками, натренерував мережу та запустив. Результат можна побачити на рис. 13-14

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

```
import numpy as np
import neurolab as nl

target = [[-1, 1, -1, -1, 1, -1, -1, 1, -1],
          [1, 1, 1, 1, -1, 1, 1, -1, 1],
          [1, -1, 1, 1, 1, 1, 1, -1, 1],
          [1, 1, 1, 1, -1, -1, 1, -1, -1],
          [-1, -1, -1, -1, 1, -1, -1, -1, -1]]

input = [[-1, -1, 1, 1, 1, 1, 1, -1, 1],
         [-1, -1, 1, -1, 1, -1, -1, -1, -1],
         [-1, -1, -1, -1, 1, -1, -1, 1, -1]]
# Створення та тренування нейромережі
net = nl.net.newhem(target)

output = net.sim(target)
print("Test on train samples (must be [0, 1, 2, 3, 4])")
print(np.argmax(output, axis=0))

output = net.sim([input[0]])
print("Outputs on recurent cycle:")
print(np.array(net.layers[1].outs))

output = net.sim(input)
print("Outputs on test sample:")
print(output)
```

Рис. 15 Код програми

```
Test on train samples (must be [0, 1, 2, 3, 4])
[0 1 2 3 4]
Outputs on recurent cycle:
[[0.      0.24    0.48    0.      0.     ]
 [0.      0.144   0.432   0.      0.     ]
 [0.      0.0576  0.4032  0.      0.     ]
 [0.      0.      0.39168 0.      0.     ]]
Outputs on test sample:
[[0.         0.         0.39168    0.         0.        ]
 [0.         0.         0.         0.         0.39168   ]
 [0.07516193 0.         0.         0.         0.07516193]]

Process finished with exit code 0
```

Рис. 16 Виконання програми

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

```
import numpy as np
import neurolab as nl

target = [[1,0,0,0,1,
           1,1,0,0,1,
           1,0,1,0,1,
           1,0,0,1,1,
           1,0,0,0,1],
          [1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1,
           1,0,0,0,0,
           1,1,1,1,1],
          [1,1,1,1,0,
           1,0,0,0,1,
           1,1,1,1,0,
           1,0,0,1,0,
           1,0,0,0,1],
          [0,1,1,1,0,
           1,0,0,0,1,
           1,0,0,0,1,
           1,0,0,0,1,
           0,1,1,1,0]]
chars = ['N', 'E', 'R', 'O']
target = np.asfarray(target)
target[target == 0] = -1
net = nl.net.newhop(target)
output = net.sim(target)
print("Test on train samples:")
for i in range(len(target)):
    print(chars[i], (output[i] == target[i]).all())
```

Рис. 17 Код програми

```
Test on train samples:
N True
E True
R True
O True

Process finished with exit code 0
```

Рис. 18 Виконання програми

```
print("\nTest on defaced N:")
test =np.asfarray([0,0,0,0,0,
                   1,1,0,0,1,
                   1,1,0,0,1,
                   1,0,1,1,1,
                   0,0,0,1,1])
test[test==0] = -1
out = net.sim([test])
print((out[0] == target[0]).all(), 'Sim. steps',len(net.layers[0].outs))
```

Рис. 19 Код програми

```
Test on defaced N:
True Sim. steps 2
```

Рис. 20 Виконання програми

```
print("\nTest on defaced E:")
test =np.asfarray(
                  [0, 0, 0, 0, 0,
                   0, 1, 1, 1, 1,
                   0, 1, 1, 1, 1,
                   0, 1, 1, 1, 1,
                   0, 0, 0, 0, 0],
                  )
test[test==0] = -1
out = net.sim([test])
print((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))
```

Рис. 21 Код програми

```
Test on defaced E:
False Sim. steps 3
```

Рис. 22 Виконання програми

Висновок: Під час виконання 4 завдання імпортував neurolab та numpy, заніс вхідні дані у вигляді складного списку та привів до форми, що сприймається функцією з бібліотеки, Створив та навчив нейронну мережу Хопфілда. Протестував навчену нейронну мережу Хопфілда. Для цього замінив деякі білі пікселі стали чорними і навпаки. Результат був True(рис. 20). Якщо навчання пройшло правильно то мережа при невеликій кількості помилок буде вгадувати букву правильно. Значить все вірно. Потім вирішив протестувати наступну букву повністю змінив білі та чорні пікселі. Результат був успішний(рис. 22).

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

```
import numpy as np
import neurolab as nl

target = [[1, 1, 1, 1, 1,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1],
          [1, 1, 1, 1, 1,
           1, 0, 0, 0, 1,
           1, 1, 1, 1, 1,
           1, 0, 0, 0, 1,
           1, 1, 1, 1, 1],
          [1, 1, 1, 1, 1,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 0,
           1, 1, 1, 1, 1]
          ]
chars = ['П', 'В', 'С']
target = np.asfarray(target)
target[target == 0] = -1
net = nl.net.newhop(target)
output = net.sim(target)
print("Test on train samples:")
for i in range(len(target)):
    print(chars[i], (output[i] == target[i]).all())
```

Рис. 23 Код програми

```
Test on train samples:
П True
В True
С True
```

Рис. 24 Виконання програми

```
print("\nTest on defaced П:")
test =np.asfarray([1, 1, 1, 1, 1,
                   1, 0, 0, 0, 1,
                   1, 0, 0, 0, 1,
                   1, 0, 1, 0, 1,
                   1, 0, 0, 0, 0])
test[test==0] = -1
out = net.sim([test])
print((out[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Рис. 25 Код програми

```
Test on defaced П:
True Sim. steps 1
```

Рис. 26 Виконання програми

Висновок: Під час виконання 5 завдання імпортував neurolab та numpy, заніс вхідні дані у вигляді складного списку та привів до форми, що сприймається функцією з бібліотеки, створив та навчив нейронну мережу Хопфілда. Протестував навчену нейронну мережу Хопфілда. Для цього замінив деякі білі піксели стали чорними і навпаки. Результат був True(рис. 26). Якщо навчання пройшло правильно то мережа при невеликій кількості помилок буде вгадувати букву правильно. Значить все вірно.

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

Репозиторій: