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)
      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
    # 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
# 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):
  '''
```

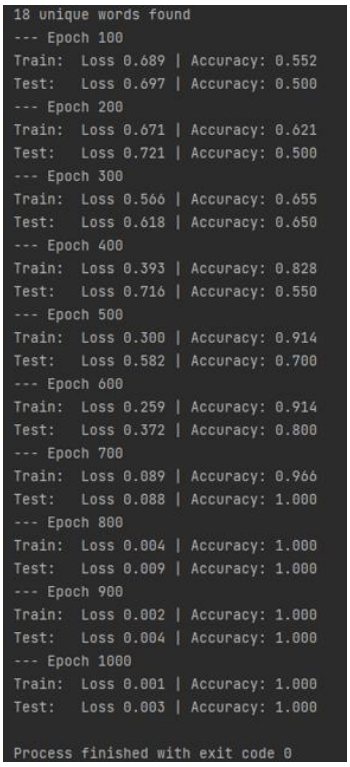
```
    # Update weights and biases using gradient descent.
    self.Whh -= learn_rate * d_Whh
    self.Wxh -= learn_rate * d_Wxh
    self.Why -= learn_rate * d_Why
    self.bh -= learn_rate * d_bh
    self.by -= learn_rate * d_by
from data import train_data, test_data
# Create the vocabulary.
vocab = 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.
```

```
  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!
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
      # 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
    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
```

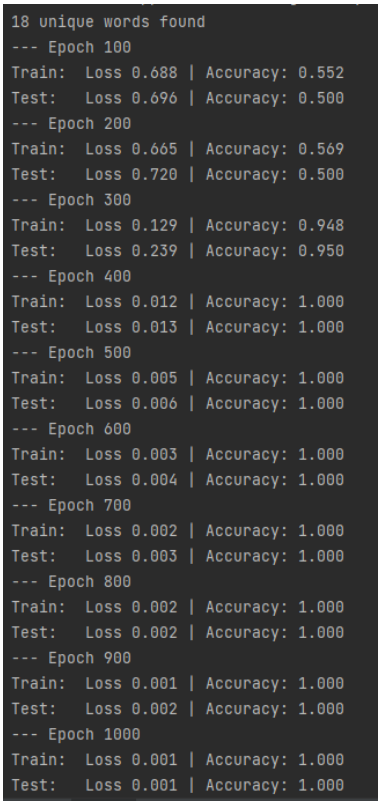
```
    # Compute the output
    y = self.Why @ h + self.by
    return y, h
  def backprop(self, d_y, learn_rate=2e-2):
    '''
    Perform a backward pass of the RNN.
    - d_y (dL/dy) has shape (output_size, 1).
    - learn_rate is a float.
    '''
    n = len(self.last_inputs)
    # Calculate dL/dWhy 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)
    # 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
      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
  # 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 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

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 сотої епохи для відслідковування прогресу

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

```python
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 3. Лістинг коду файла LR_6_task_2.py

```
C:\Users\ankud\AppData\Local\Programs\Python\Python311\python.exe C:\Users\ankud\Desktop\myAi\labó\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
```

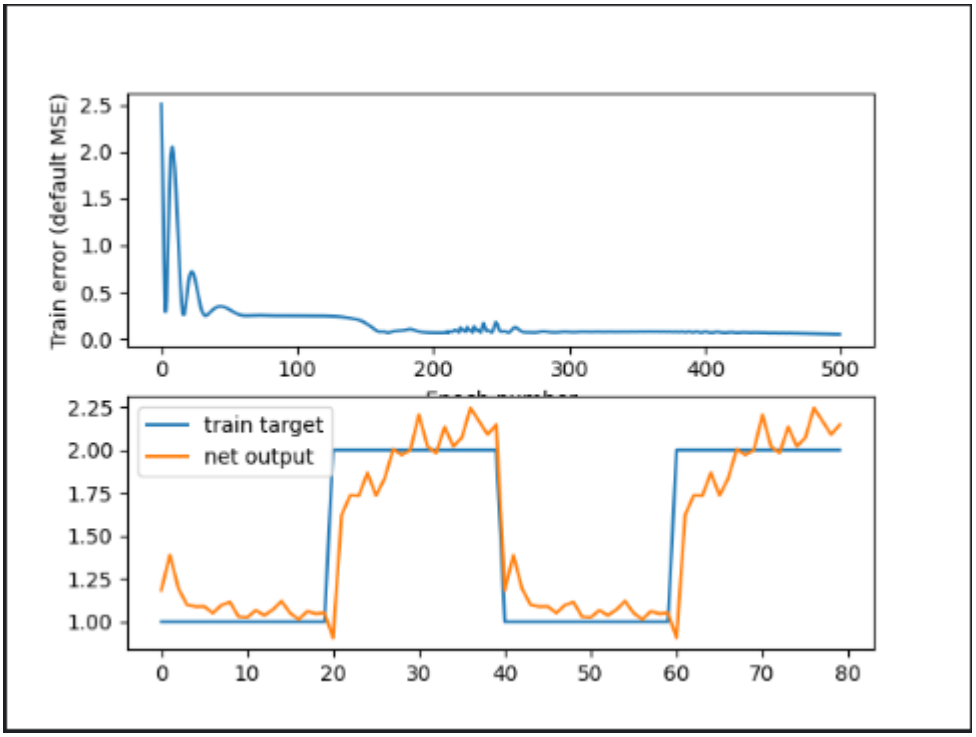


Рис 4. Результат файлу файла LR_6_task_2.py

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

```python
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 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

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

```python
import numpy as np
import neurolab as nl

target = np.array([
    [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[target == 0] = -1

net = nl.net.newhop(target)

output = net.sim(target)
print("Test on train samples:")
```

Рис 7. Лістинг коду файла LR_6_task_4.py

```python
output = net.sim(target)
print("Test on train samples:")
for i in range(len(target)):
    print(chars[i], (output[i] == target[i]).all())

print("\nTest on defaced M:")

test = np.array(
    [0, 0, 0, 0, 0,
     1, 1, 0, 0, 1,
     1, 1, 1, 1, 1,
     0, 1, 1, 1, 1,
     1, 0, 0, 0, 1]
)
test[test == 0] = -1

out = net.sim([test])
print((out[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))

```

```
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.

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

```python
import numpy as np
import neurolab as nl

# Змінені дані
target = np.array([
    [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,
     1, 0, 0, 0, 1,
     1, 0, 0, 0, 1,
     1, 0, 0, 0, 1,
     0, 1, 1, 1, 0]
])
chars = ['A', 'N', 'K', 'O']

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())

print("\nTest on defaced M:")
```

```
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())

print("\nTest on defaced M:")

# Змінені дані
test = np.array(
    [0, 0, 0, 0, 0,
     1, 1, 0, 0, 1,
     1, 1, 1, 1, 1,
     0, 1, 1, 1, 1,
     1, 0, 0, 0, 1]
)
test[test == 0] = -1

out = net.sim([test])
print((out[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Рис 9. Лістинг коду файла LR_6_task_5.py

```
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 лабараторної роботи**,** використовуючи спеціалізовані бібліотеки та мову програмування Python навчився досліджувати деякі типи нейронних мереж**.**

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