

## ЛАБОРАТОРНА РОБОТА № 6

### ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

<https://github.com/MaximVengel/AI>

**Завдання 2.1.** Ознайомлення з рекурентними нейронними мережами.

LR\_6\_task\_1.py

```
import random
from numpy.random import randn
import numpy as np

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
```

					ДУ «Житомирська політехніка».23.121.06.000 – Лр6			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Венгель М.І.			Звіт з лабораторної роботи		Літ.	Арк.
Перевір.		Голенко М.Ю.						1
Керівник								12
Н. контр.							ФІКТ Гр. ІПЗ-20-2	
Зав. каф.								

```

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):
    '''
    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))

```

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				
Змн.	Арк.	№ докум.	Підпис	Дата		3

```

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)):

```

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

```

# 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

```

```

LR_6_task_1 x
C:\Users\38098\Desktop\lab06ai\venv\Scripts\p
18 unique words found
--- Epoch 100
Train: Loss 0.688 | Accuracy: 0.552
Test: Loss 0.699 | Accuracy: 0.500
--- Epoch 200
Train: Loss 0.668 | Accuracy: 0.638
Test: Loss 0.723 | Accuracy: 0.650
--- Epoch 300
Train: Loss 0.530 | Accuracy: 0.690
Test: Loss 0.611 | Accuracy: 0.650
--- Epoch 400
Train: Loss 0.390 | Accuracy: 0.845
Test: Loss 0.554 | Accuracy: 0.650
--- Epoch 500
Train: Loss 0.182 | Accuracy: 0.931
Test: Loss 0.410 | Accuracy: 0.800
--- Epoch 600
Train: Loss 0.688 | Accuracy: 0.552
Test: Loss 0.611 | Accuracy: 0.600
--- Epoch 700
Train: Loss 0.406 | Accuracy: 0.828
Test: Loss 0.214 | Accuracy: 0.900
--- Epoch 800
Train: Loss 0.144 | Accuracy: 0.966
Test: Loss 0.084 | Accuracy: 1.000

```

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

```

--- Epoch 900
Train: Loss 0.003 | Accuracy: 1.000
Test: Loss 0.004 | Accuracy: 1.000
--- Epoch 1000
Train: Loss 0.002 | Accuracy: 1.000
Test: Loss 0.002 | Accuracy: 1.000

Process finished with exit code 0

```

Рис. 1. LR\_6\_task\_1.py

```

Run: main x
C:\Users\38098\Desktop\lab06ai\venv\Scr
18 unique words found
--- Epoch 100
Train: Loss 0.689 | Accuracy: 0.552
Test: Loss 0.697 | Accuracy: 0.500
--- Epoch 200
Train: Loss 0.665 | Accuracy: 0.638
Test: Loss 0.731 | Accuracy: 0.650
--- Epoch 300
Train: Loss 0.702 | Accuracy: 0.621
Test: Loss 0.707 | Accuracy: 0.500
--- Epoch 400
Train: Loss 0.596 | Accuracy: 0.690
Test: Loss 0.668 | Accuracy: 0.700
--- Epoch 500
Train: Loss 0.008 | Accuracy: 1.000
Test: Loss 0.008 | 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 0.001 | Accuracy: 1.000
--- Epoch 1000
Train: Loss 0.001 | Accuracy: 1.000
Test: Loss 0.001 | Accuracy: 1.000

Process finished with exit code 0

```

Рис. 2. main.py

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				6
Змн.	Арк.	№ докум.	Підпис	Дата		

**Висновок до завдання:** на рисунках 1, 2 бачимо повідомлення “18 unique words found” це означає, що змінна **vocab** тепер буде мати перелік всіх слів, які вживаються щонайменше в одному навчальному тексті, далі відбувається тренування мережі, також бачимо виведення кожної соті епохи для відслідковування прогресу.

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

LR\_6\_task\_2.py

```
import neurolab as nl
import numpy as np

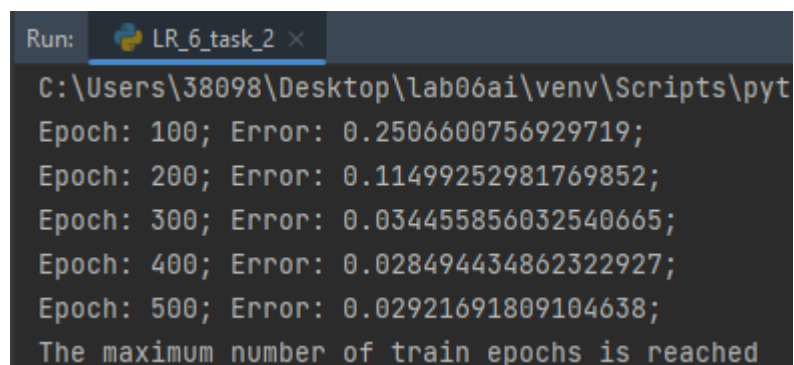
i1 = np.sin(np.arange(0, 20))
i2 = np.sin(np.arange(0, 20)) * 2
t1 = np.ones([1, 20])
t2 = np.ones([1, 20]) * 2
input = np.array([i1, i2, i1, i2]).reshape(20 * 4, 1)
target = np.array([t1, t2, t1, t2]).reshape(20 * 4, 1)
net = nl.net.newelm([-2, 2], [10, 1], [nl.trans.TanSig(), nl.trans.PureLin()])
net.layers[0].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
net.layers[1].initf = nl.init.InitRand([-0.1, 0.1], 'wb')
net.init()

error = net.train(input, target, epochs=500, show=100, goal=0.01)

output = net.sim(input)

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()
```



```
Run: LR_6_task_2 x
C:\Users\38098\Desktop\lab06ai\venv\Scripts\pyt
Epoch: 100; Error: 0.2506600756929719;
Epoch: 200; Error: 0.11499252981769852;
Epoch: 300; Error: 0.034455856032540665;
Epoch: 400; Error: 0.028494434862322927;
Epoch: 500; Error: 0.02921691809104638;
The maximum number of train epochs is reached
```

Рис. 3. Консоль.

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

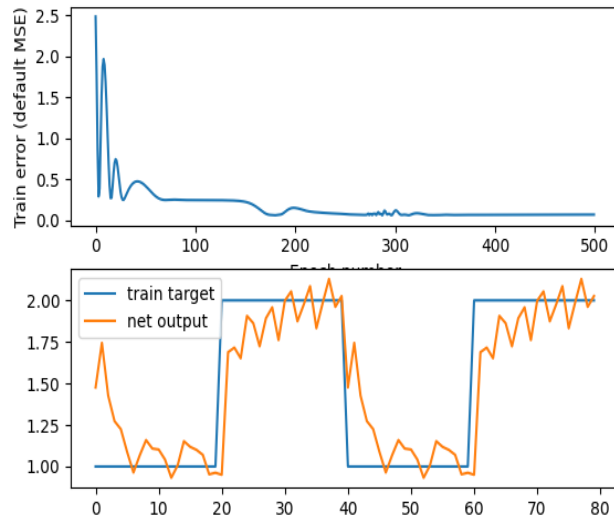


Рис. 4. Pylab

**Висновок до завдання:** використовуючи бібліотеки `neurolab` та `numpy` створив мережу з двома прошарками, створив модель сигналу для навчання та виконав тренування мережі.

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

LR\_6\_task\_3.py

```
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)
```



```

Run: LR_6_task_3 x
C:\Users\38098\Desktop\lab06ai\venv\Scripts\python.exe C:\Users\3
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. 5. Завдання 2.3

### Завдання 2.4. Дослідження рекурентної нейронної мережі Хопфілда Hopfield

Recurrent network (newhop)

LR\_6\_task\_4.py

```

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.asarray(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())

```

```

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))

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))

```

```

Run: LR_6_task_4 x
C:\Users\38098\Desktop\lab06ai\venv\S
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
True Sim. steps 2

Test on defaced E:
False Sim. steps 3

Process finished with exit code 0

```

Рис. 5. Завдання 2.4

**Висновок до завдання:** використав бібліотеки `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.

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

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

## LR\_6\_task\_5.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import numpy as np
import neurolab as nl

target = [[1, 1, 1, 1, 0,
            1, 0, 0, 0, 1,
            1, 1, 1, 1, 1,
            1, 0, 0, 0, 1,
            1, 1, 1, 1, 0],
          [1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1],
          [1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1],
          [1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1,
            1, 1, 0, 1, 1,
            1, 1, 1, 1, 1],
          ]
chars = ['Є', 'B', 'B']
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())

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

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

Результат виконання:

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				11
Змн.	Арк.	№ докум.	Підпис	Дата		

```

Run: LR_6_task_5 ×
C:\Users\38098\Desktop\lab06ai\venv\Scripts\p
Test on train samples:
E False
B True
B True

Test on defaced E:
False Sim. steps 1

Test on defaced B:
False Sim. steps 1

Process finished with exit code 0

```

Рис. 6. Завдання 2.5

**Висновок до завдання:** використав бібліотеки 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.

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

		Венгель М.І.			ДУ «Житомирська політехніка».23.121.06.000 – Лр6	Арк.
		Голенко М.Ю.				12
Змн.	Арк.	№ докум.	Підпис	Дата		