ЛАБОРАТОРНА РОБОТА № 6
ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи:

Завдання 1

Лістинг програми:

```
from data import train_data, test_data
# Створити словник
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) # знайдено 18 унікальних слів

# Призначити індекс кожному слову
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']) # 16 (це може змінитися)
print(idx_to_word[0]) # сумно (це може змінитися)
import numpy as np
def createInputs(text):
 '''
Повертає масив унітарних векторів
 які представляють слова у введеному рядку тексту
 - текст є рядком string
 - Унітарний вектор має форму (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
import numpy as np
from numpy.random import randn
```

					ДУ «Житомирська політехніка».23.122.4.000 – Лр6		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Дяченко В.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів
Перевір.		.					
Керівник						ФІКТ Гр. КН-20-1(1)	
Н. контр.							
Зав. каф.							

```
class RNN:
    def __init__(self, input_size, output_size, hidden_size=64):
        # Вес
        self.Whh = np.random.randn(hidden_size, hidden_size) / 1000
        self.Wxh = np.random.randn(hidden_size, input_size) / 1000
        self.Why = np.random.randn(output_size, hidden_size) / 1000

        # Зміщення
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))

    def forward(self, inputs):
        h = np.zeros((self.Whh.shape[0], 1))

        # Виконання кожного кроку в нейронній мережі RNN
        for x in inputs:
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)

        # Обчислення виходу
        y = self.Why @ h + self.by

        return y, h

def softmax(xs):
 # Застосування функції Softmax для вхідного масиву
 return np.exp(xs) / sum(np.exp(xs))
# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN(vocab_size, 2)
inputs = createInputs('i am very good')
out, h = rnn.forward(inputs)
probs = softmax(out)
print(probs) # [[0.50000095], [0.49999905]]
import numpy as np
import random

# Задамо параметри
vocab = list(set([w for text in train_data.keys() for w in text.split(' ')]))
vocab_size = len(vocab)

word_to_idx = {w: i for i, w in enumerate(vocab)}
idx_to_word = {i: w for i, w in enumerate(vocab)}

class RNN:
    def __init__(self, input_size, output_size, hidden_size=64):
        # Вес
        self.Whh = np.random.randn(hidden_size, hidden_size) / 1000
        self.Wxh = np.random.randn(hidden_size, input_size) / 1000
        self.Why = np.random.randn(output_size, hidden_size) / 1000
```

```
        # Зміщення
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))

    def softmax(self, xs):
        # Застосування функції Softmax для вхідного масиву
        return np.exp(xs) / np.sum(np.exp(xs))

    def forward(self, inputs):
        h = np.zeros((self.Whh.shape[0], 1))
        self.last_inputs = inputs
        self.last_hs = {0: h}

        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
            self.last_hs[i + 1] = h

        y = self.Why @ h + self.by
        return y, h

    def backprop(self, d_y, learn_rate=2e-2):
        n = len(self.last_inputs)
        d_Why = d_y @ self.last_hs[n].T
        d_by = d_y

        d_Whh = np.zeros(self.Whh.shape)
        d_Wxh = np.zeros(self.Wxh.shape)
        d_bh = np.zeros(self.bh.shape)

        d_h = self.Why.T @ d_y

        for t in reversed(range(n)):
            temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)
            d_bh += temp
            d_Whh += temp @ self.last_hs[t].T
            d_Wxh += temp @ self.last_inputs[t].T
            d_h = self.Whh @ temp

        for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
            np.clip(d, -1, 1, out=d)

        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


def createInputs(text):
```

```
    inputs = []
    for w in text.split(' '):
        v = np.zeros((vocab_size, 1))
        v[word_to_idx[w]] = 1
        inputs.append(v)
    return inputs

def processData(data, backprop=True):
    items = list(data.items())
    random.shuffle(items)
    loss = 0
    num_correct = 0

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

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

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

        if backprop:
            d_L_d_y = probs
            d_L_d_y[target] -= 1
            rnn.backprop(d_L_d_y)

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

# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN(vocab_size, 2)

# Цикл тренування
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))
```

```
18 unique words found
1
not
```

Рис.1. К-ть унікальних слів у нашому словнику, індекс обраного слова "good", а також слово, яке має індекс 0.

```python
def softmax(xs):
 # Застосування функції Softmax для вхідного масиву
 return np.exp(xs) / sum(np.exp(xs))
# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN(vocab_size, 2)
inputs = createInputs('i am very good')
out, h = rnn.forward(inputs)
probs = softmax(out)
print(probs) # [[0.50000095], [0.49999905]]
```

```
[[0.50000243]
 [0.49999757]]
```

Рис.2. Ймовірність кожного класу (good/bad).

```
--- Epoch 100
Train:  Loss 0.689 | Accuracy: 0.552
Test:   Loss 0.698 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.669 | Accuracy: 0.655
Test:   Loss 0.725 | Accuracy: 0.600
--- Epoch 300
Train:  Loss 0.575 | Accuracy: 0.638
Test:   Loss 0.652 | Accuracy: 0.700
--- Epoch 400
Train:  Loss 0.426 | Accuracy: 0.845
Test:   Loss 0.652 | Accuracy: 0.650
--- Epoch 500
Train:  Loss 0.308 | Accuracy: 0.879
Test:   Loss 0.687 | Accuracy: 0.700
--- Epoch 600
Train:  Loss 0.191 | Accuracy: 0.914
Test:   Loss 0.611 | Accuracy: 0.650
--- Epoch 700
Train:  Loss 0.460 | Accuracy: 0.759
Test:   Loss 0.696 | Accuracy: 0.600
--- Epoch 800
Train:  Loss 0.044 | Accuracy: 1.000
Test:   Loss 0.553 | Accuracy: 0.800
--- Epoch 900
Train:  Loss 0.009 | Accuracy: 1.000
Test:   Loss 0.354 | Accuracy: 0.850
--- Epoch 1000
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.256 | Accuracy: 0.950
```

```
18 unique words found
--- Epoch 100
Train:  Loss 0.688 | Accuracy: 0.552
Test:   Loss 0.697 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.668 | Accuracy: 0.672
Test:   Loss 0.716 | Accuracy: 0.550
--- Epoch 300
Train:  Loss 0.492 | Accuracy: 0.707
Test:   Loss 1.138 | Accuracy: 0.400
--- Epoch 400
Train:  Loss 0.435 | Accuracy: 0.776
Test:   Loss 0.591 | Accuracy: 0.650
--- Epoch 500
Train:  Loss 0.359 | Accuracy: 0.845
Test:   Loss 0.471 | Accuracy: 0.750
--- Epoch 600
Train:  Loss 0.109 | Accuracy: 0.966
Test:   Loss 0.501 | Accuracy: 0.850
--- Epoch 700
Train:  Loss 0.015 | Accuracy: 1.000
Test:   Loss 0.615 | Accuracy: 0.850
--- Epoch 800
Train:  Loss 0.006 | Accuracy: 1.000
Test:   Loss 0.505 | Accuracy: 0.950
--- Epoch 900
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.507 | Accuracy: 0.950
--- Epoch 1000
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.518 | Accuracy: 0.950
```

Рис.3. Результати роботи рекурентної нейронної мережі (1 варіант – без використання готового файлу, 2 варіант – main.py).

Висновки: виконавши дане завдання ми навчили рекурентну нейронну мережу (RNN) розрізняти текст за двома класами. Як можна побачити, результати

позитивні, з кожною епохою все більша точність (Accuracy: 0.950) та менші втрати. Якщо ми почнали з 50% правильної класифікації, в кінці отримали 100% на тренувальних та 95% на тестових.

Завдання 2

Лістинг програми:

```
import neurolab as nl
import numpy as np
# Створення моелей сигналу для навчання
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)
# Створення мережі з 2 прошарками
net = nl.net.newelm([[-2, 2]], [10, 1], [nl.trans.TanSig(),
nl.trans.PureLin()])
# Ініціалізуйте початкові функції вагів
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()
```

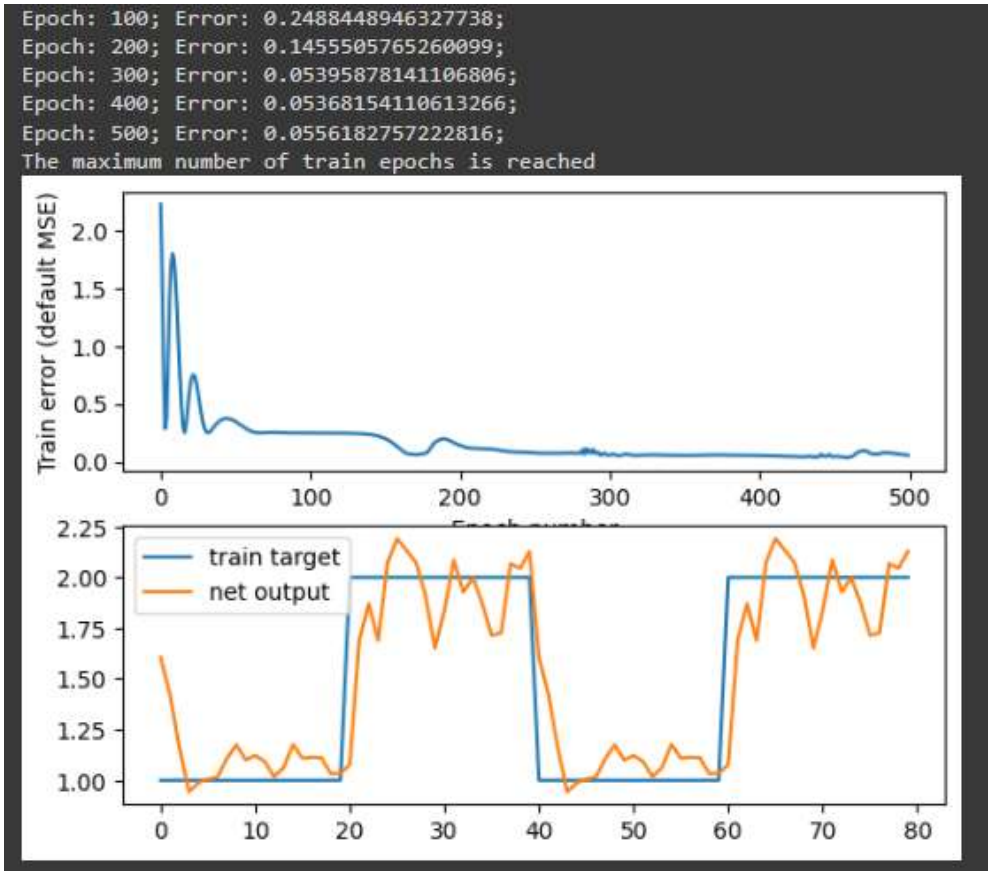


Рис.4. Графіки помилки та апроксимації сигналу, та інформація з вікна терміналу.

Висновки: в ході виконання даного завдання ми створили модель нейромережі з двома прошарами. Використовується тангенс-сигмоїдна функція активації для прихованого прошарку і лінійна функція активації для вихідного прошарку. Під час тренування мережі помилка падає до певного моменту (приблизно 300 епох), після чого зупиняється на величині близькій до **0.53**.

На другому графіку видно, що тренувальна цільова функція практично залишається незмінною, тоді як вихід моделі (net output) коливається і не зовсім точно відтворює цільовий сигнал. Помилка може бути пов'язана зі складністю моделі та параметрами тренування, такими як кількість нейронів в прихованому прошарку, швидкість навчання, кількість епох тренування і інші. Хоча результат і не відповідає очікуванням (помилка **0.01**), він все одно приблизно описує тренувальну ф-цію.

Завдання 3

Лістинг програми:

```
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)
```

```
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]]
```

Рис.5. Результат створення та тренування нейронної мережі Хемінга.

Завдання 4

Лістинг програми:

```
import numpy as np
import neurolab as nl
# N E R O
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
# Create and train network
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))
```

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

Рис.6. Результат створення та навчання нейронної мережі Хемінга.

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

Рис.7. Тестування навченої мережі.

Протестуємо інші букви з помилками.

Лістинг програми:

```
print("\nTest on defaced E:")
test =np.asfarray( [1,1,1,0,1,
 1,0,0,0,0,
 1,1,1,1,1,
 1,0,0,0,0,
 1,1,0,1,1])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))
print("\nTest on defaced R:")
test =np.asfarray([0,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,0,
 1,0,1,1,0,
 1,0,0,0,1])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[2]).all(), 'Sim. steps',len(net.layers[0].outs))
print("\nTest on defaced O:")
test =np.asfarray([0,1,1,1,0,
 1,0,0,1,1,
 1,0,0,1,1,
 1,0,0,1,1,
 0,1,1,1,0])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[3]).all(), 'Sim. steps',len(net.layers[0].outs))
```

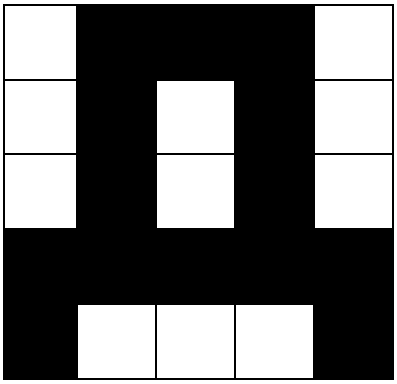
```
Test on defaced E:
True Sim. steps 2

Test on defaced R:
True Sim. steps 1

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

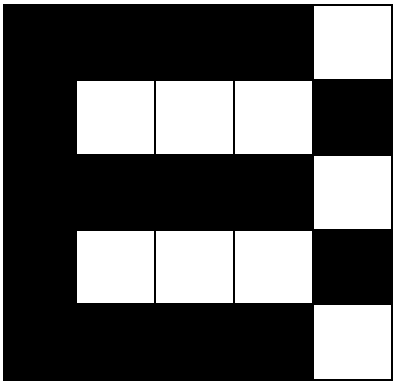
Рис.8. Результат тестування.

Висновки: В ході виконання даного завдання ми створили та навчили нейронну мережу Хопфілда. З результатів видно, що мережа успішно розпізнає задані літери, коли тести виконуються на навчальних даних. Специфікація

"кількість ітерацій (Sim. steps)" вказує на кількість кроків, які нейромережа виконує для тестування, щоб збігтися до правильного шаблону під час розпізнавання зразка з помилками. Кожна літера мала певну к-ть помилок (1-3), тому кількість кроків симуляції може бути різною.

Завдання 5





Лістинг програми:

```
import numpy as np
import neurolab as nl

target = [[0,1,1,1,0,
 0,1,0,1,0,
 0,1,0,1,0,
 1,1,1,1,0,
 1,0,0,0,1
 ],
 [1,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,0],
 [1,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,0]]
chars = ['Д', 'В', 'В']
target = np.asfarray(target)
target[target == 0] = -1
# Create and train network
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 Д:")
test =np.asfarray([1,1,1,1,0,
 0,1,0,1,0,
 0,1,0,1,0,
 1,1,1,1,0,
 1,1,0,0,1
])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[0]).all(), 'Sim. steps',len(net.layers[0].outs))
```

```
Test on train samples:
Д True
В True
В True
```

Рис.9. Результат створення та навчання нейронної мережі Хемінга.

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

Рис.10. Тестування навченої мережі. (2 помилки)

Лістинг програми:

```
print("\nTest on defaced В:")
test =np.asfarray([1,1,1,1,0,
 1,0,0,0,1,
 1,1,1,1,1,
 1,0,0,0,1,
 1,1,1,1,1])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))
print("\nTest on defaced В:")
test =np.asfarray([0,0,0,0,0,
0,0,0,0,1,
0,0,0,0,0,
0,0,0,0,0,
0,1,1,1,1])
test[test==0] = -1
out = net.sim([test])
print ((out[0] == target[2]).all(), 'Sim. steps',len(net.layers[0].outs))
```

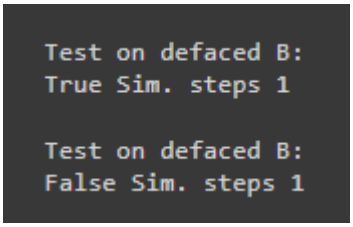


Рис.11. Результат тестування. (1 та 12 відповідно).

Висновки: Для всіх трьох букв результати показуються як "True", що означає - мережа правильно розпізнала букви після навчання. Також вимагається лише 1 крок симуляції, тому можна сказати, що мережа навчилася розпізнавати букви "Д", "В" з високою точністю. (В останньому прикладі спеціально зроблено багато помилок для наочності, вибірка невелика, тому може визначати букви лише при певній к-ть помилок).