**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:** Дослідження рекурентної нейронної мережі Елмана.

Лістинг програми:

import numpy as np  
import random  
  
from rnn import RNN  
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))  
  
 test\_loss, test\_acc = processData(test\_data, backprop=False)  
 print('Test:\tLoss %.3f | Accuracy: %.3f' % (test\_loss, test\_acc))

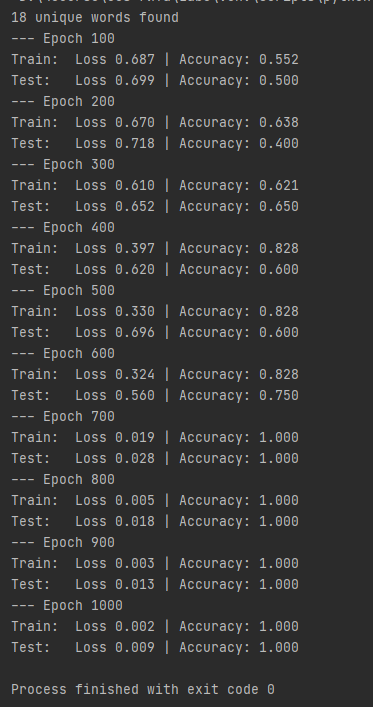


Рис.1 – Результутат виконання програми.

Висновки: По-перше, код є хорошою основою для навчання RNN для класифікації тексту. По-друге, продуктивність моделі залежатиме від багатьох факторів, включаючи кількість епох, архітектуру мережі та формат даних.

**Завдання №2:** Дослідження рекурентної нейронної мережі Елмана.

Лістинг програми:

Import neurolab as nl  
import numpy as np  
import matplotlib.pyplot as plt  
  
# 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  
  
plt.subplot(211)  
plt.plot(error)  
plt.xlabel('Epoch number')  
plt.ylabel('Train error (default MSE)')  
  
plt.subplot(212)  
plt.plot(target.reshape(80))  
plt.plot(output.reshape(80))  
plt.legend(['train target', 'net output'])  
plt.show()

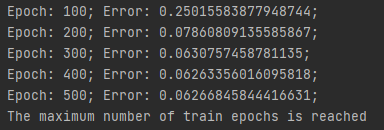


Рис.2 – Результутат виконання програми.

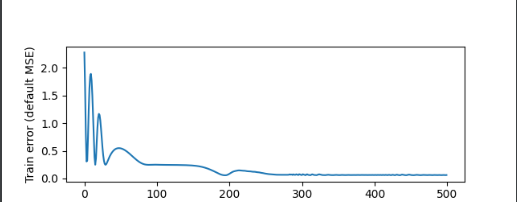


Рис.3 – Результутат виконання програми.

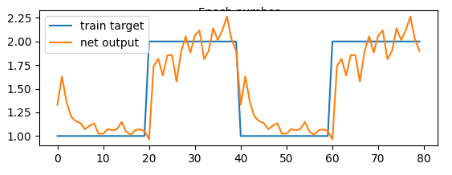


Рис.4 – Результутат виконання програми.

Висновки: код навчає нейронну мережу з двома шарами для класифікації двох сигналів. З графіка помилки видно, що мережа швидко навчається і досягає цільової помилки менше 0,01 за 500 епох. Загалом, код є ефективним способом навчання нейронної мережі для класифікації сигналів.

**Завдання №3:** Дослідження нейронної мережі Хемінга.

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)

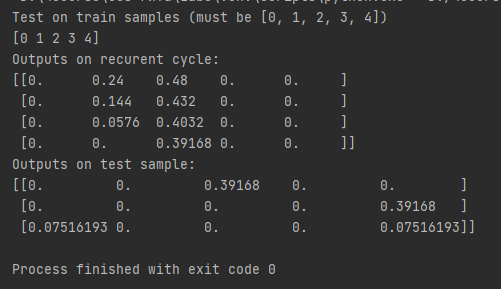


Рис.5 – Результутат виконання програми.

Висновки: з виведення коду видно, що мережа правильно відновлює всі шаблони навчання. Це означає, що мережа добре навчилася і може використовуватися для розпізнавання шаблонів.

**Завдання №4:** Дослідження рекурентної нейронної мережі Хопфілда.

Лістинг програми:

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

# 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))

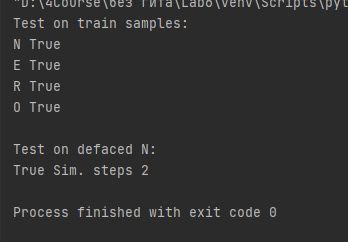


Рис.6 – Результутат виконання програми.

Висновки: код демонструє ефективне використання мереді Хопфілда для розпізнавання та відновлення бінарних шаблонів букв. Мережа показала хорошу здатність до навчання та стійкість до шуму.

**Завдання №5:** Дослідження рекурентної нейронної мережі Хопфілда.

Лістинг програми:

import numpy as np  
import neurolab as nl  
  
target = [[1, 0, 0, 1, 0,  
 1, 0, 1, 0, 0,  
 1, 1, 0, 0, 0,  
 1, 0, 1, 0, 0,  
 1, 0, 0, 1, 0],  
 [1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1],  
 [0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0]]  
  
chars = ['K', 'O', 'I']  
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 I:")  
test = np.asfarray([0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 0, 1, 0, 0, 0])  
test[test == 0] = -1  
out = net.sim([test])  
print((out[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))

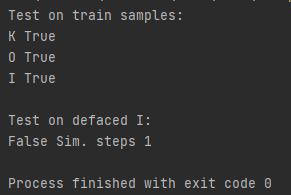


Рис.7 – Результутат виконання програми.

**Висновки:** в ході лабораторної роботи ми використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися дослідили мережі Хопфілда та їх функціонал.

Посилання на репозиторій: <https://github.com/RomanMatskevich/SHI.git>