**ЛАБОРАТОРНА РОБОТА № 6**

**ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ**

**Мета роботи:**

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

**Хід роботи:**

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

Лістинг коду:

from data import train\_data, test\_data

import numpy as np

import random

from rnn import RNN

# 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

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

def processData(data, backprop=True):

    '''

    Повернення втрат RNN і точності для даних

    - дані подані як словник, що відображує текст як True або False.

    - backprop визначає, чи потрібно використовувати звороднє розподілення

    '''

    items = list(data.items())

    random.shuffle(items)

    loss = 0

    num\_correct = 0

    for x, y in items:

        inputs = createInputs(x)

        target = int(y)

        # Пряме розподілення

        out, \_ = rnn.forward(inputs)

        probs = softmax(out)

        # Обчислення втрат / точності

        loss -= np.log(probs[target])

        num\_correct += int(np.argmax(probs) == target)

        if backprop:

            # Создание dL/dy

            d\_L\_d\_y = probs

            d\_L\_d\_y[target] -= 1

            # Зворотне розподілення

            rnn.backprop(d\_L\_d\_y)

    return loss / len(data), num\_correct / len(data)

def softmax(xs):

    # Застосування функції Softmax для вхідного масиву

    return np.exp(xs) / sum(np.exp(xs))

# 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)

# 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]]

# Цикл тренування

for epoch in range(1000):

    train\_loss, train\_acc = processData(train\_data)

    if epoch % 100 == 99:

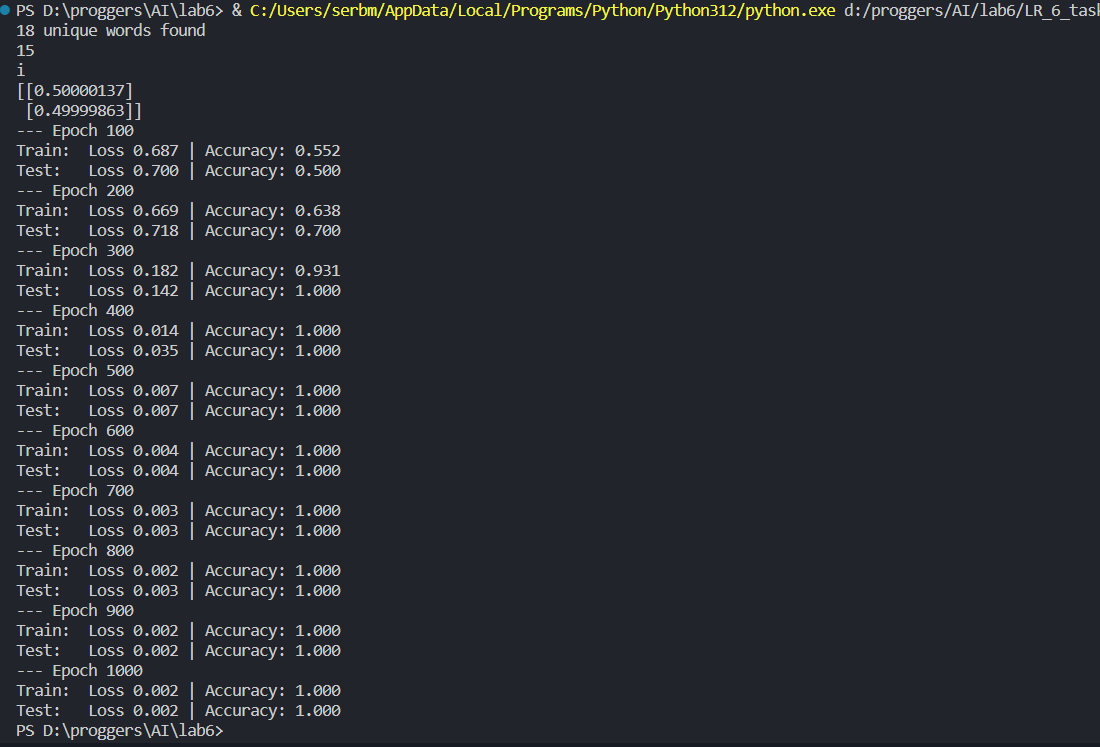
        print('--- Epoch %d' % (epoch + 1))

        print('Train:\tLoss %.3f | Accuracy: %.3f' % (train\_loss[0], train\_acc))

        test\_loss, test\_acc = processData(test\_data, backprop=False)

        print('Test:\tLoss %.3f | Accuracy: %.3f' % (test\_loss[0], test\_acc))

Результат виконання:



Висновок: З результату виведення можна зробити висновок, що мережа добре навчається та досягає 100% точності (Accuracy) вже на 400 епосі навчання.

**Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана**

**(Elman Recurrent network (newelm))**

Лістинг коду:

import neurolab as nl

import numpy as np

import pylab as pl

# Створення моделей 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

pl.subplot(211)

pl.plot(error)

pl.xlabel('Epoch number')

pl.ylabel('Train error (default MSE)')

pl.subplot(212)

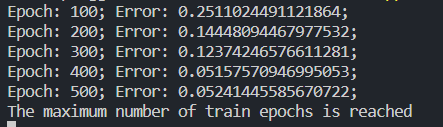
pl.plot(target.reshape(80))

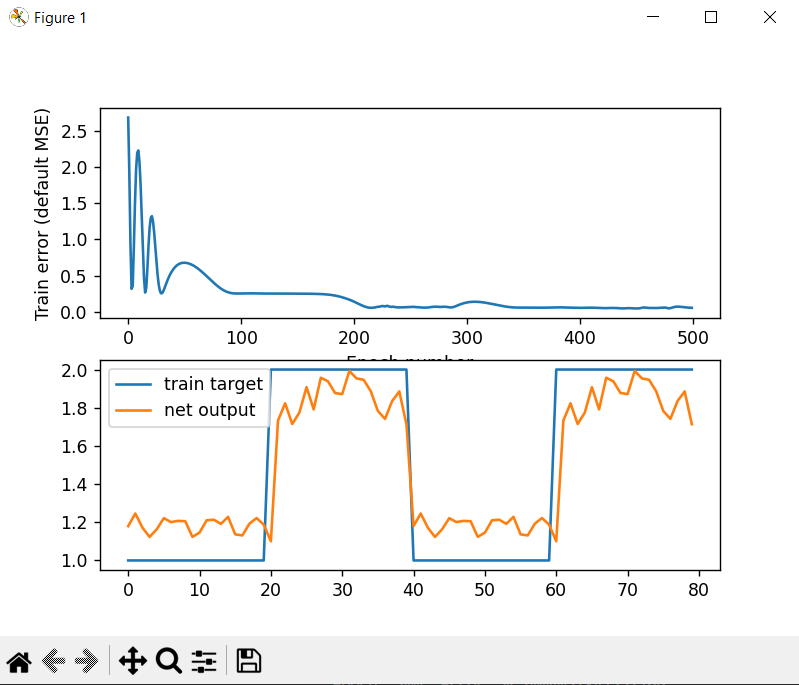
pl.plot(output.reshape(80))

pl.legend(['train target', 'net output'])

pl.show()

Результат виконання:





Висновок: З графіку помилок можна зробити висновок, що мережа зменшує кількість помилок протягом навчання. Нажаль, не було досягнуто цільового значення 0.01 помилок, та програма закінчила навчання після 500 епох.

З графіку апроксимації сигналу можна сказати, що мережа відтворює цільові патерни, хоч і не дуже точно.

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

Лістинг коду:

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:")

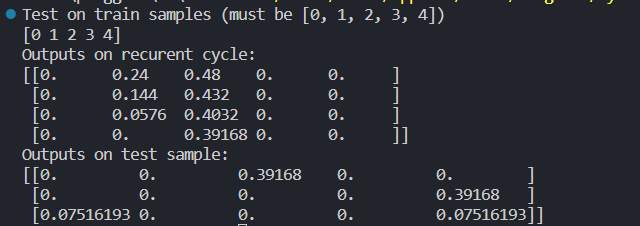
print(np.array(net.layers[1].outs))

output = net.sim(input)

print("Outputs on test sample:")

print(output)

Результат виконання:



**Завдання 2.4. Дослідження рекурентної нейронної мережі Хопфілда**

**Hopfield Recurrent network (newhop)**

Лістинг коду:

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))

print("\nTest on defaced O:")

test\_o = np.asfarray([1, 1, 1, 1, 1,

                      1, 0, 0, 0, 0,

                      1, 0, 0, 0, 0,

                      1, 0, 0, 0, 0,

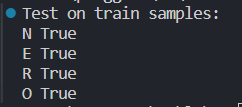
                      1, 0, 0, 0, 0])

test\_o[test\_o == 0] = -1

out\_o = net.sim([test\_o])

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

Результат виконання:







Висновок: Хопфілдова мережа, яку навчили на конкретних зразках, проявляє високу ефективність у відновленні літер, навіть у випадку невеликих змін чи пошкоджень.

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

Лістинг коду:

import numpy as np

import neurolab as nl

# Щ М Ю

target = [[1, 1, 1, 1, 0,

           1, 1, 1, 1, 0,

           1, 1, 1, 1, 0,

           1, 1, 1, 1, 0,

           1, 1, 1, 1, 1],

          [1, 1, 0, 1, 1,

           1, 0, 1, 0, 1,

           1, 0, 0, 0, 1,

           1, 0, 0, 0, 1,

           1, 0, 0, 0, 1],

          [1, 0, 0, 1, 0,

           1, 0, 1, 0, 1,

           1, 1, 1, 0, 1,

           1, 0, 1, 0, 1,

           1, 0, 0, 1, 0]]

chars = ['Щ', 'М', 'Ю']

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 'Ю':")

test\_you = np.asfarray([1, 0, 1, 1, 1,

                        1, 0, 1, 0, 1,

                        1, 0, 1, 0, 1,

                        1, 0, 1, 0, 1,

                        1, 0, 1, 1, 1])

test\_you[test\_you == 0] = -1

out\_you = net.sim([test\_you])

print((out\_you[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))

print("\nTest on defaced 'М':")

test\_m = np.asfarray([1, 1, 1, 1, 1,

                      1, 0, 1, 0, 1,

                      1, 0, 0, 0, 1,

                      1, 0, 0, 0, 1,

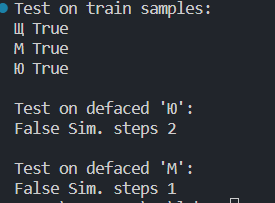
                      1, 0, 0, 0, 1])

test\_m[test\_m == 0] = -1

out\_m = net.sim([test\_m])

print((out\_m[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))

Результат виконання:



***Github:*** [***https://github.com/mtvi/ipz202\_Shcherback\_lab6***](https://github.com/mtvi/ipz202_Shcherback_lab6)

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