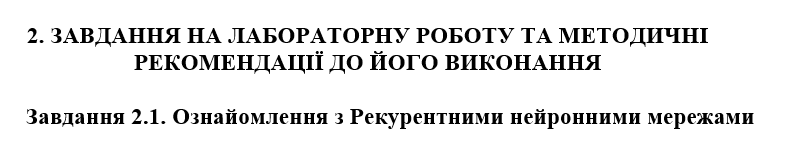
**ЛАБОРАТОРНА РОБОТА № 6**

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

**Посилання на GitHub: https://github.com/enot1k666/labsOAI.git**

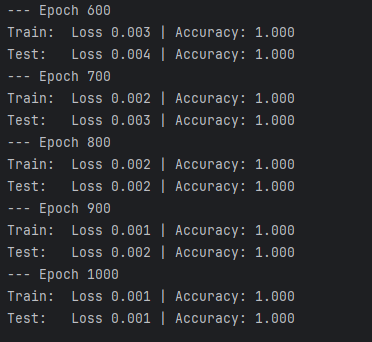
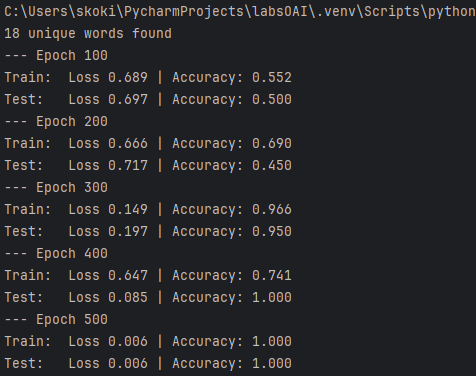


Лістинг програми:

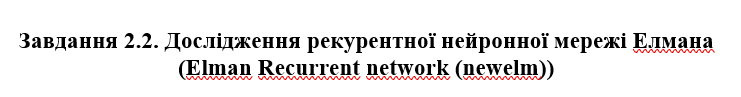
import random  
import numpy as np  
from numpy.random import randn  
  
  
class RNN:  
 def \_\_init\_\_(self, input\_size, output\_size, hidden\_size=64):  
 self.Whh = randn(hidden\_size, hidden\_size) / 1000  
 self.Wxh = randn(hidden\_size, input\_size) / 1000  
 self.Why = randn(output\_size, hidden\_size) / 1000  
  
 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))  
  
 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)  
  
 # Calculate dL/dWhy and dL/dby.  
 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  
  
 # Backpropagate through time.  
 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  
  
  
from data import train\_data, test\_data  
  
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)  
  
word\_to\_idx = {w: i for i, w in enumerate(vocab)}  
idx\_to\_word = {i: w for i, w in enumerate(vocab)}  
  
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 softmax(xs):  
 return np.exp(xs) / sum(np.exp(xs))  
  
  
rnn = RNN(vocab\_size, 2)  
  
  
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 = 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)  
  
  
# 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.item(), train\_acc))  
  
 test\_loss, test\_acc = processData(test\_data, backprop=False)  
 print('Test:\tLoss %.3f | Accuracy: %.3f' % (test\_loss.item(), test\_acc))

Результат виконання:



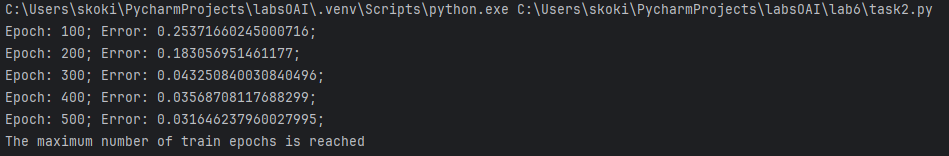
**Висновок**: Було створено просту рекурентну нейронну мережу. Під час тренування модель зменшує втрату та збільшує точність на навчальних даних з плином часу.

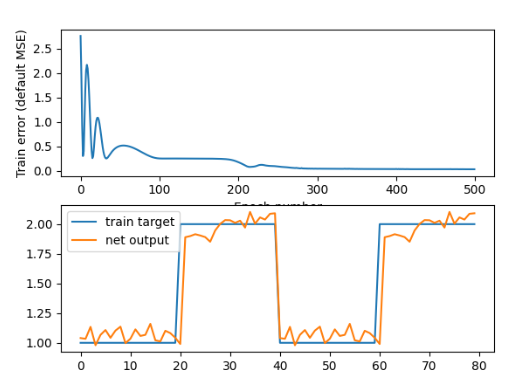


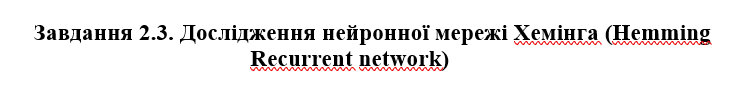
**Лістинг програми:**

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()

**Результат виконання:**



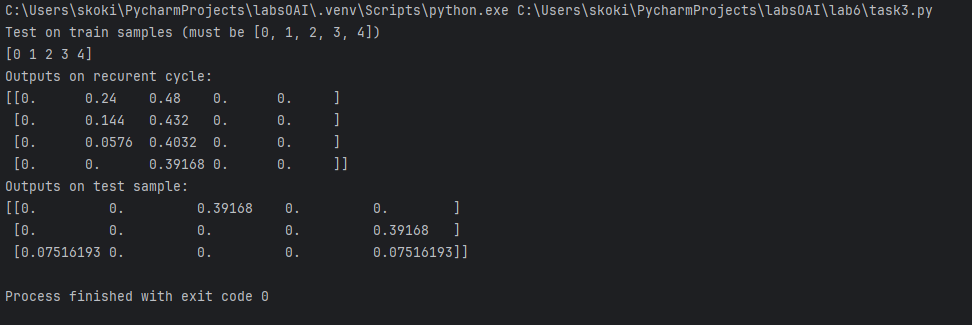


****

**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)

**Результат виконання:**

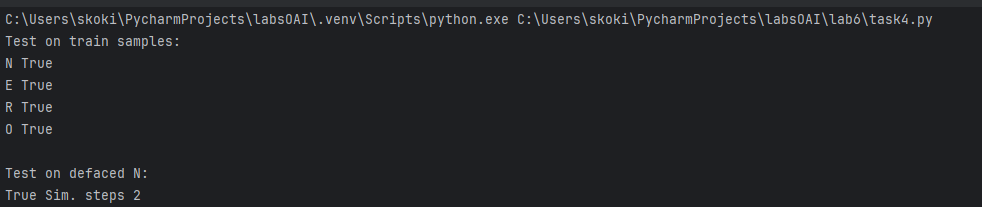


****

**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())

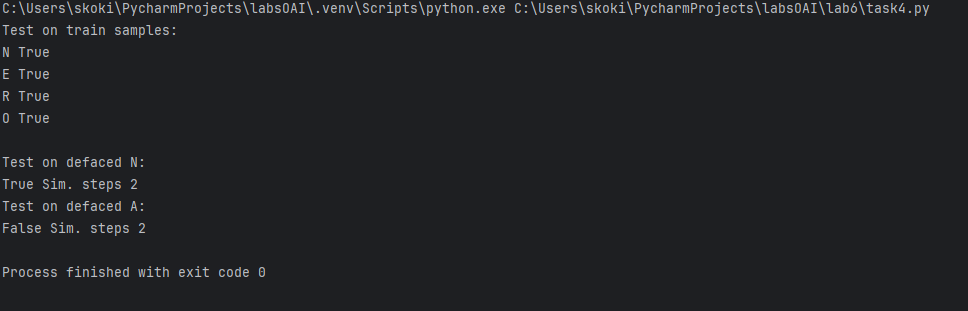
**Результат виконання:**



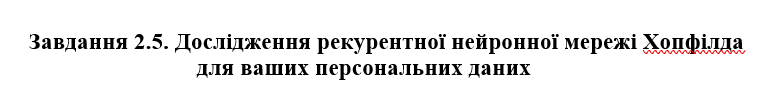
**Тест:**

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

**Результат виконання:**



**Висновок:** У коді використовується бібліотека neurolab для створення та тренування простої рекурентної нейронної мережі.Метою є розпізнавання букв (N, E, R, O) на основі заданих шаблонів. Код успішно тренує мережу на навчальних даних та протестовує її на них. Всі чотири букви (N, E, R, O) розпізнаються правильно.



**Лістинг програми:**

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

**Результат виконання:**

