**Практична робота № 6**

**Варіант 13**

**Дослідження рекурентних нейронних мереж**

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

**Хід роботи:**

**Завдання 6.1:** Ознайомлення з Рекурентними нейронними мережами. Прочитайте та виконайте дії згідно рекомендацій до виконання.

Лістинг файлу task-1.py

from data import train\_data, test\_data  
import numpy as np  
from numpy.random import randn  
  
vocab = list(set([word for text in train\_data.keys() for word in text.split()]))  
vocab\_size = len(vocab)  
  
print(f"{vocab\_size} unique words in the training data")  
  
word\_to\_index = {word: i for i, word in enumerate(vocab)}  
index\_to\_word = {i: word for i, word in enumerate(vocab)}  
print(word\_to\_index)  
print(index\_to\_word)  
  
  
def create\_inputs(text):  
 inputs = []  
 for w in text.split(' '):  
 v = np.zeros((vocab\_size, 1))  
 v[word\_to\_index[w]] = 1  
 inputs.append(v)  
  
 return inputs  
  
  
def softmax(xs):  
 return np.exp(xs) / sum(np.exp(xs))  
  
  
def process\_data(data, rnn, backprop=True):  
 items = list(data.items())  
 np.random.shuffle(items)

loss = 0  
 num\_correct = 0  
  
 for x, y in items:  
 inputs = create\_inputs(x)  
 target = int(y)  
  
 out, \_ = rnn.forward(inputs)  
 probs = softmax(out)  
  
 loss -= float(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)  
  
  
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))  
  
 self.last\_inputs = None  
 self.last\_hs = None  
  
 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  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 rnn = RNN(vocab\_size, 2)  
  
 for epoch in range(1000):  
 train\_loss, train\_acc = process\_data(train\_data, rnn, backprop=True)  
  
 if epoch % 100 == 99:  
 print(f"Epoch {epoch + 1}")  
 print(f"Train loss: {train\_loss:0.3f}, Train acc: {train\_acc:0.3f}")  
  
 test\_loss, test\_acc = process\_data(test\_data, rnn, backprop=False)  
 print(f"Test loss: {test\_loss:.3f}, Test acc: {test\_acc:.3f}\n")

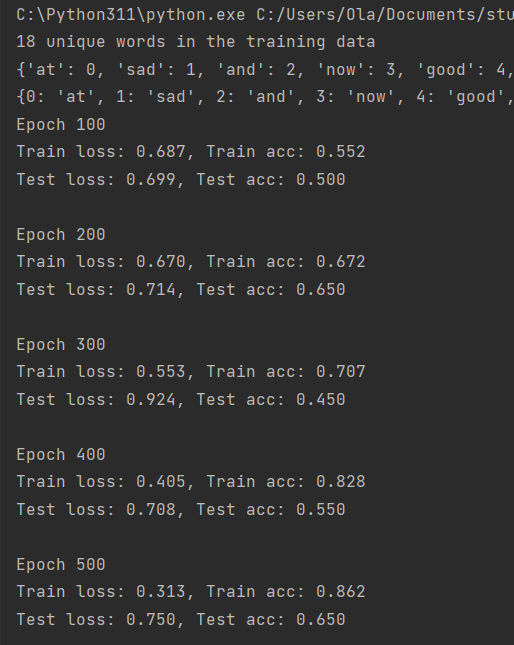


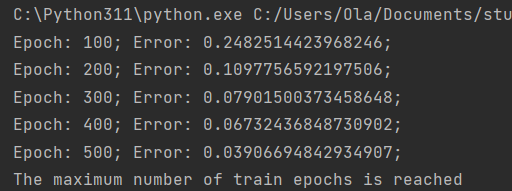
Рис.6.1. task-1.py

**Завдання 6.2:** Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm)).

Рекурентні нейронні мережі Елмана часто застосовують для детектування амплітуди сигналу (апроксимації, представлення, наближення сигналів). Тобто нейронна мережа за попередніми відліками сигналу повинна передбачити наступні значення і якомога ближче до реальних.

Лістинг файлу task-2.py

import numpy as np  
import neurolab as nl  
import pylab as pl  
  
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)  
  
pl.subplot(211)  
pl.plot(error)  
pl.xlabel('Number of epochs')  
pl.ylabel('Error (default SSE)')  
  
pl.subplot(212)  
pl.plot(target.reshape(80))  
pl.plot(output.reshape(80))  
pl.legend(['train target', 'net output'])  
pl.tight\_layout(w\_pad=1.5)  
pl.show()



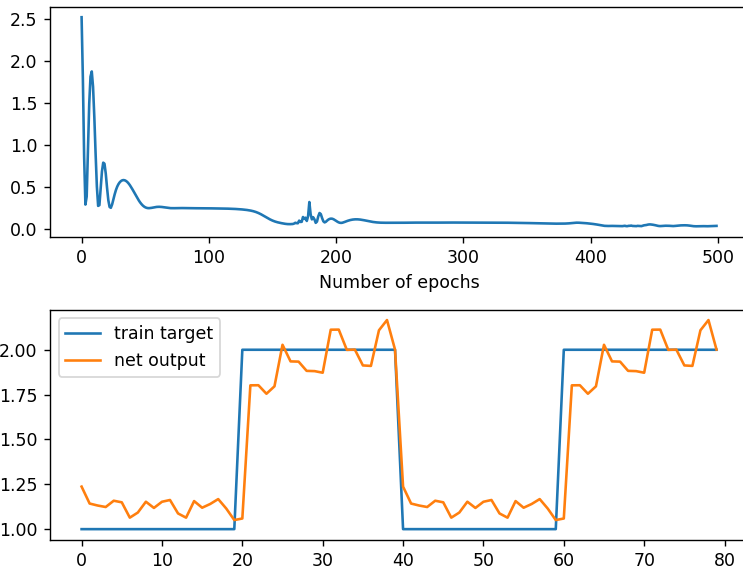
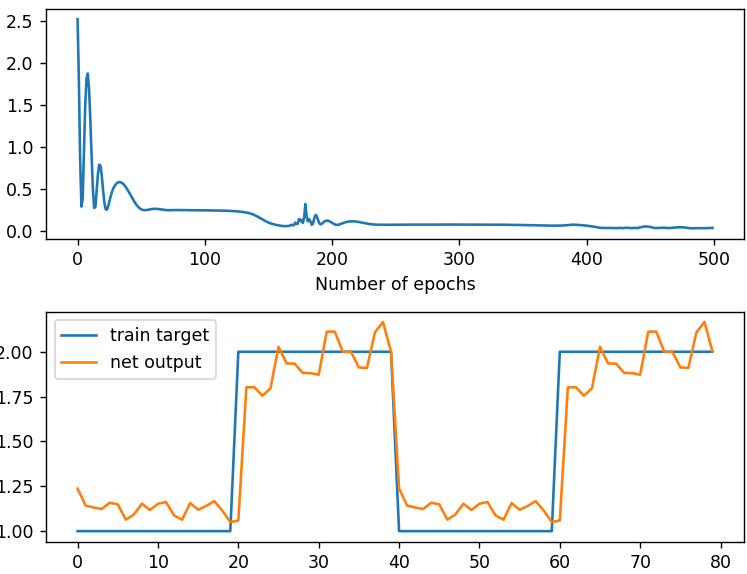


Рис.6.2. task-2.py

**Завдання 6.3:** Дослідження нейронної мережі Хемінга (Hemming Recurrent network).

Лістинг файлу 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]]  
  
net = nl.net.newhem(target)  
  
output = net.sim(target)  
print("Test on train data (must be [0, 1, 2, 3, 4]):")  
print(np.argmax(output, axis=0))  
  
output = net.sim([input[0]])  
print("Outputs on recurrent cycle:")  
print(np.array(net.layers[1].outs))  
  
output = net.sim(input)  
print("Test on test sample:")  
print(output)

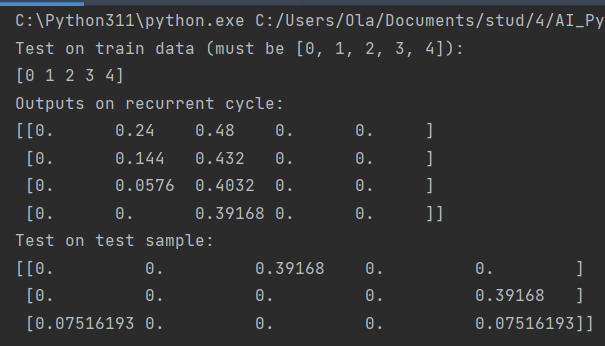


Рис.6.3. task-3.py

**Завдання 6.4:** Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop).

Необхідно навчити рекурентну нейронну мережу Хопфілда розпізнавати букви слова. Формат звернення до відповідної функції для neurolab подано у таблиці.

Лістинг файлу 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.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(output)):  
 print(chars[i], (output[i] == target[i]).all())  
  
print("Test of 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  
output = net.sim([test])  
print((output[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))  
  
print("Test of defaced E:")  
test = np.asfarray([1, 1, 0, 1, 0,  
 1, 0, 0, 0, 0,  
 1, 1, 0, 0, 0,  
 1, 1, 0, 0, 0,  
 1, 1, 1, 1, 1])  
test[test == 0] = -1  
output = net.sim([test])  
print((output[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))  
  
print("Test of defaced R:")  
test = np.asfarray([0, 1, 1, 0, 0,  
 1, 0, 0, 1, 0,  
 0, 1, 1, 0, 0,  
 1, 0, 0, 1, 0,  
 1, 0, 0, 0, 0])  
test[test == 0] = -1  
output = net.sim([test])  
print((output[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))  
  
print("Test of defaced O:")  
test = np.asfarray([0, 0, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 0, 1, 0, 0, 1,  
 0, 1, 0, 0, 1,  
 0, 1, 1, 1, 0])  
test[test == 0] = -1  
output = net.sim([test])  
print((output[0] == target[3]).all(), 'Sim. steps', len(net.layers[0].outs))

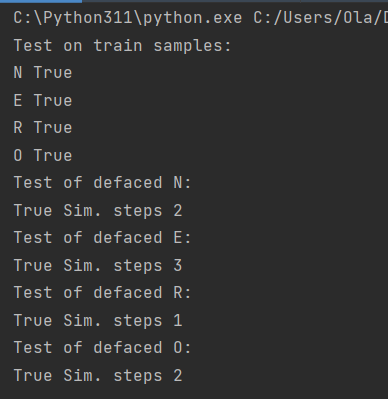


Рис.6.4. task-4.py

**Завдання 6.5:** Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних

По аналогії з попереднім завданням візьміть перші букви Ваших Прізвища, Імя та По батькові (кирилицею). Закодуйте їх матрицею пікселів та кодом одиниць і нулів. Навчіть мережу розпізнавати Ваші букви. Протестуйте мережу на можливість розпізнавання кожної букви шляхом внесення помилок в тест. Код тесту та результати тестування занесіть у звіт.

Вхідні дані: літери M, O, J.

Фрагмент лістингу файлу task-5.py

import numpy as np  
import neurolab as nl  
  
target = [[1, 0, 0, 0, 1,  
 1, 1, 0, 1, 1,  
 1, 1, 0, 1, 1,  
 1, 0, 1, 0, 1,  
 1, 0, 1, 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],  
  
 [1, 1, 1, 1, 1,  
 0, 0, 1, 0, 0,  
 0, 0, 1, 0, 0,  
 1, 0, 1, 0, 0,  
 1, 1, 1, 0, 0]]  
  
chars = ['M', 'O', 'J']  
target = np.asfarray(target)  
target[target == 0] = -1

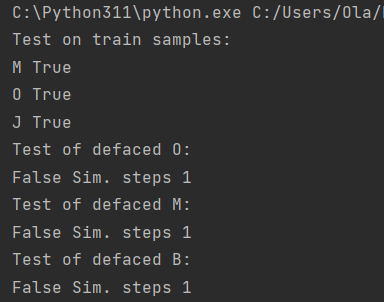


Рис.6.5. task-5.py

[**https://github.com/avrorilka/AI\_Python**](https://github.com/avrorilka/AI_Python)

**Висновки:** в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися досліджувати деякі типи нейронних мереж.