Міністерство освіти і науки України

Державний університет ,,Житомирська політехніка”

Кафедра: Фікт

Група: ІПЗ-20-1

# Лабораторна робота №6

«ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ»

Виконав: Денисюк Н. Г.

Прийняв: Голенко М. Ю.

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

**Виконання роботи:**

**Завдання №1:**

Код:

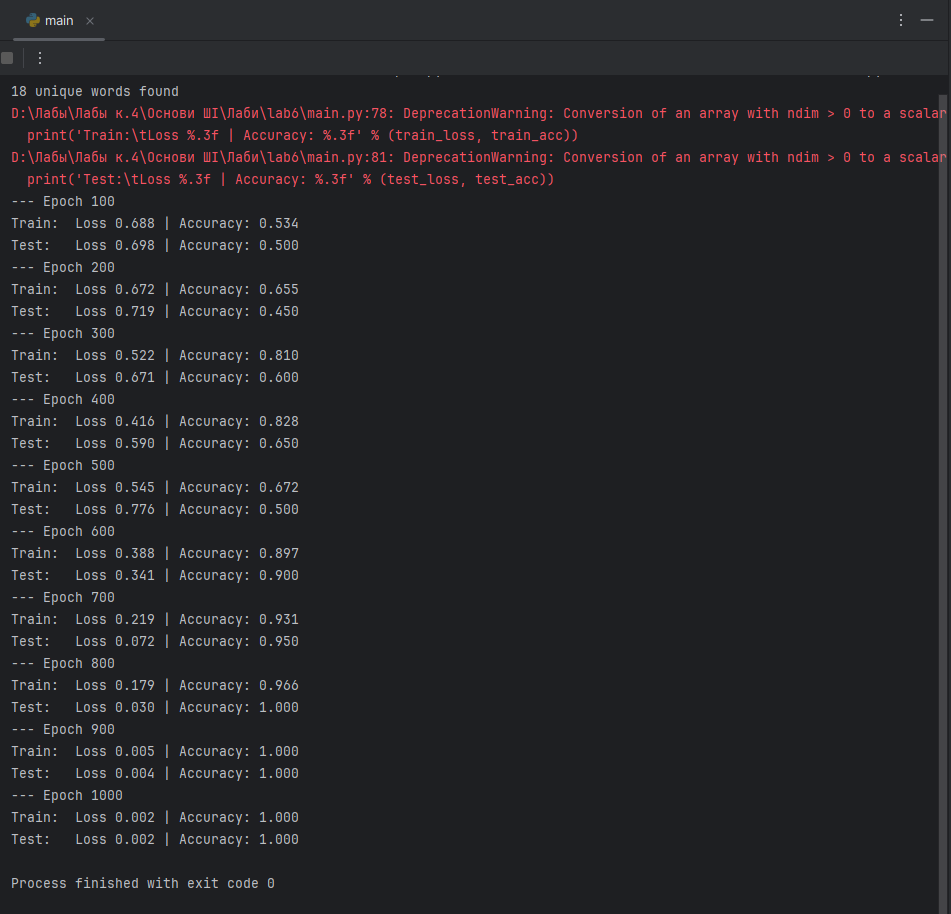
LR\_6\_task\_1.py:

import numpy as np  
from numpy.random import randn  
  
class RNN:  
 # Класична рекурентна нейронна мережа  
  
 def \_\_init\_\_(self, input\_size, output\_size, hidden\_size=64):  
 # Weights  
 self.Whh = randn(hidden\_size, hidden\_size) / 1000  
 self.Wxh = randn(hidden\_size, input\_size) / 1000  
 self.Why = 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):  
 *'''  
 Виконання передачі нейронної мережі за допомогою вхідних даних  
 Повернення результатів виведення та прихованого стану  
 Вивід – це масив одного унітарного вектора з формою (input\_size, 1)  
 '''* h = np.zeros((self.Whh.shape[0], 1))  
  
 self.last\_inputs = inputs  
 self.last\_hs = { 0: h }  
  
 # Виконання кожного кроку в нейронній мережі RNN  
 for i, x in enumerate(inputs):  
 h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)  
 self.last\_hs[i + 1] = h  
  
 # Compute the output  
 y = self.Why @ h + self.by  
  
 return y, h  
  
 def backprop(self, d\_y, learn\_rate=2e-2):  
 *'''  
 Виконання фази зворотного розповсюдження мережі RNN.  
 - d\_y (dL/dy) має форму (output\_size, 1).  
 - learn\_rate є дійсним числом float.  
 '''* n = len(self.last\_inputs)  
  
 # Підрахунок dL/dWhy та dL/dby.  
 d\_Why = d\_y @ self.last\_hs[n].T  
 d\_by = d\_y  
  
 # Ініціалізація dL/dWhh, dL/dWxh, і dL/dbh до нуля  
 d\_Whh = np.zeros(self.Whh.shape)  
 d\_Wxh = np.zeros(self.Wxh.shape)  
 d\_bh = np.zeros(self.bh.shape)  
  
 # Обчислення dL/dh для останнього h.  
 # dL/dh = dL/dy \* dy/dh  
 d\_h = self.Why.T @ d\_y  
  
 # Зворотне розповсюдження по часу.  
 for t in reversed(range(n)):  
 # Среднее значение: dL/dh \* (1 - h^2)  
 temp = ((1 - self.last\_hs[t + 1] \*\* 2) \* d\_h)  
  
 # dL/db = dL/dh \* (1 - h^2)  
 d\_bh += temp  
  
 # dL/dWhh = dL/dh \* (1 - h^2) \* h\_{t-1}  
 d\_Whh += temp @ self.last\_hs[t].T  
  
 # dL/dWxh = dL/dh \* (1 - h^2) \* x  
 d\_Wxh += temp @ self.last\_inputs[t].T  
  
 # Далі dL/dh = dL/dh \* (1 - h^2) \* Whh  
 d\_h = self.Whh @ temp  
  
 # Відсікаємо, щоб попередити розрив градієнтів.  
 for d in [d\_Wxh, d\_Whh, d\_Why, d\_bh, d\_by]:  
 np.clip(d, -1, 1, out=d)  
  
 # Обновляємо ваги і зміщення з використанням градієнтного спуску.  
 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

main.py:

import numpy as np  
import random  
  
from LR\_6\_task\_1 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))

**Результат виконання:**



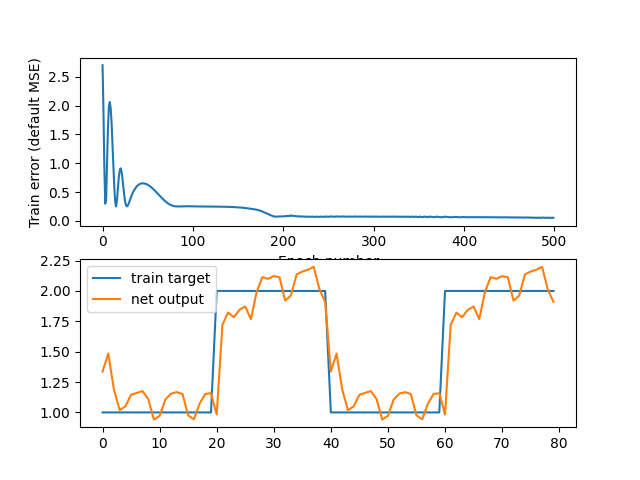
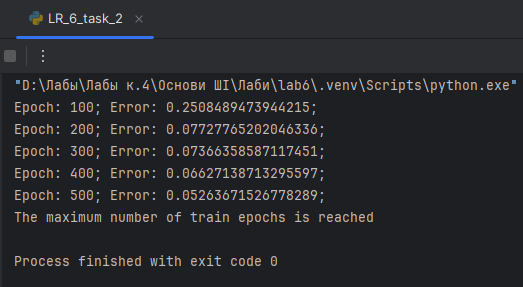
**Завдання №2:** Дослідження рекурентної нейронної мережі Елмана

(Elman Recurrent network (newelm))

Код:

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()

**Результат виконання:**

**** ****

Графік навчальної помилки показує швидке зниження помилки на початку навчання, з подальшим поступовим зменшенням до стабільного рівня.

Графік виходу мережі (net output) порівняно з цільовими значеннями (train target) показує, що мережа досить добре слідує за цільовими значеннями, хоча і є деякі відхилення.

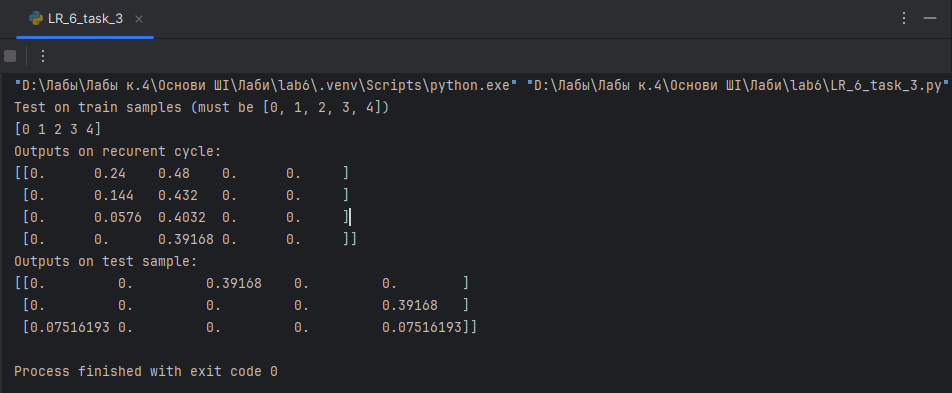
В цілому, нейронна мережа демонструє задовільний рівень навчання, хоча можливо подальше навчання або налаштування гіперпараметрів може покращити результати.

**Завдання №3:** Дослідження нейронної мережі Хемінга (Hemming Recurrent network)

Код:

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:**



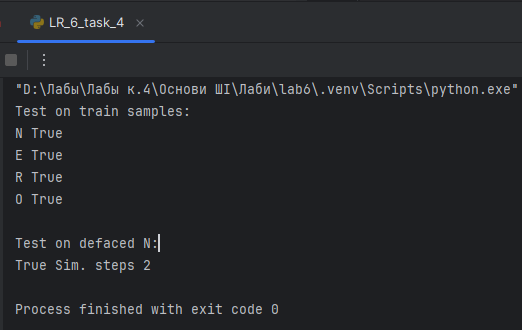
**Завдання №4:** Дослідження рекурентної нейронної мережі Хопфілда

Hopfield Recurrent network (newhop)

Код:

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))

**Результат виконання:**

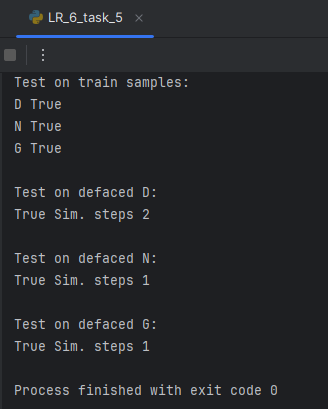


**Завдання №5:** Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних

Код:

import numpy as np  
import neurolab as nl  
  
# Визначення піксельних матриць для літер Д, Н, Г  
D = [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]  
  
N = [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]  
  
G = [0, 1, 1, 1, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 0, 1, 1, 1, 0]  
  
chars = ['D', 'N', 'G']  
target = [D, N, G]  
target = np.asfarray(target)  
target[target == 0] = -1  
  
# Створення та навчання мережі Хопфілда  
net = nl.net.newhop(target)  
  
# Тестування на тренувальних зразках  
output = net.sim(target)  
print("Test on train samples:")  
for i in range(len(target)):  
 print(chars[i], (output[i] == target[i]).all())  
  
# Внесення незначних помилок у тестові зразки та перевірка розпізнавання  
  
print("\nTest on defaced D:")  
test\_D = [1, 1, 1, 1, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 1, 1,  
 1, 1, 1, 1, 0] # Змінений останній піксель  
test\_D = np.asfarray(test\_D)  
test\_D[test\_D == 0] = -1  
out\_D = net.sim([test\_D])  
print((out\_D[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))  
  
print("\nTest on defaced N:")  
test\_N = [1, 0, 0, 0, 1,  
 1, 1, 0, 0, 1,  
 1, 0, 1, 0, 1,  
 1, 0, 0, 1, 0, # Змінений останній піксель  
 1, 0, 0, 0, 1]  
test\_N = np.asfarray(test\_N)  
test\_N[test\_N == 0] = -1  
out\_N = net.sim([test\_N])  
print((out\_N[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))  
  
print("\nTest on defaced G:")  
test\_G = [0, 1, 1, 1, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 1, 1, 0, # Змінений останній піксель  
 1, 0, 0, 0, 1,  
 0, 1, 1, 1, 0]  
test\_G = np.asfarray(test\_G)  
test\_G[test\_G == 0] = -1  
out\_G = net.sim([test\_G])  
print((out\_G[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))

**Результат виконання:**

****

**Висновок:** використовуючи спеціалізовані бібліотеки та мову програмування Python навчився дослідижувати деякі типи нейронних мереж.