**ЛАБОРАТОРНА РОБОТА № 6**

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

***Мета:*** використовуючи спеціалізовані бібліотеки та мову

програмування Python навчитися дослідити деякі типи нейронних мереж.

**Хід роботи:**

**Посилання на репозиторій: https://github.com/AlexanderSydorchuk/AI-Lab6**

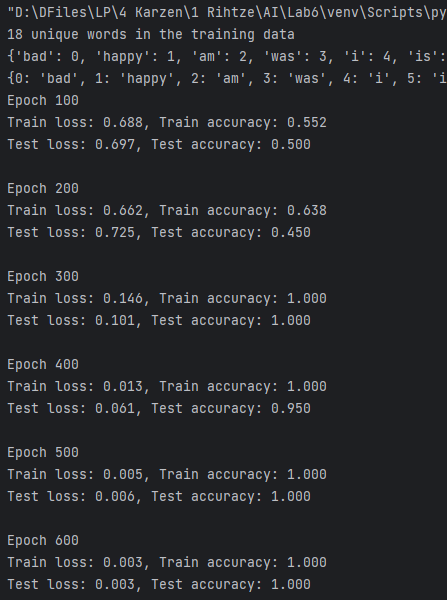
**Завдання 2.1.** Ознайомлення з Рекурентними нейронними мережами

Лістинг програми:

from data import train\_data, test\_data  
import numpy as np  
from numpy.random import randn  
  
# створення словника  
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")  
  
# призначаємо індекс кожному слову  
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 accuracy: {train\_acc:0.3f}")  
  
 test\_loss, test\_acc = process\_data(test\_data, rnn, backprop=False)  
 print(f"Test loss: {test\_loss:.3f}, Test accuracy: {test\_acc:.3f}\n")

Результат виконання програми:



Отримана найпростіша рекурентна нейронна мережа, яка здатна ефективно навчатися.

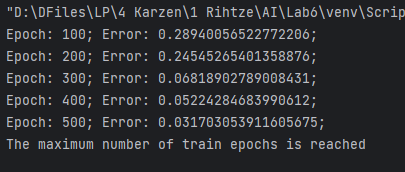
**Завдання 2.2.** Дослідження рекурентної нейронної мережі Елмана

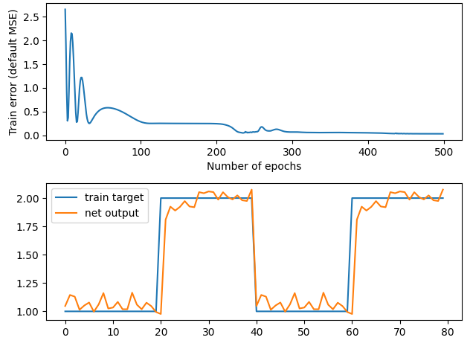
(Elman Recurrent network (newelm))

Лістинг програми:

import numpy as np  
import neurolab as nl  
import matplotlib.pyplot 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()])  
  
# Ініціалізація 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)  
# Запуск мережі  
output = net.sim(input)  
  
# Побудова графіків  
pl.subplot(211)  
pl.plot(error)  
pl.xlabel('Number of epochs')  
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.tight\_layout(w\_pad=1.5)  
pl.show()

Результат виконання програми:





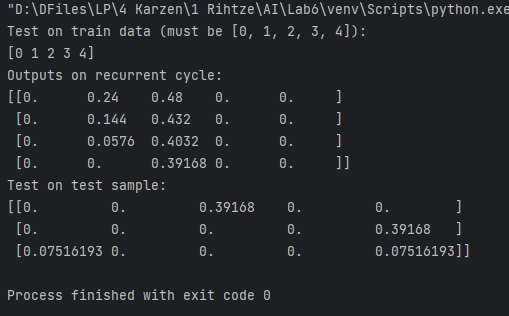
Як ми бачимо, за відносно невеликої кількості коду завдяки підключеним готовим бібліотекам вдалося створити модель нейронної мережі, що виконує поставлене завдання з високою точністю.

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

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  
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)

Результат виконання програми:

****

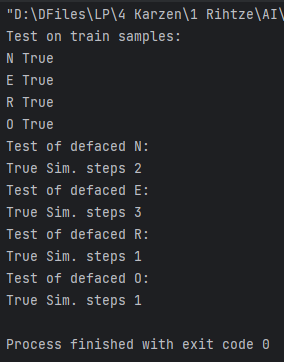
**Завдання 2.4.** Дослідження рекурентної нейронної мережі Хопфілда

Hopfield Recurrent network (newhop)

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  
  
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, 1, 1, 1,  
 0, 0, 0, 0, 0,  
 1, 1, 0, 1, 1,  
 1, 0, 0, 0, 0,  
 1, 0, 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([1, 1, 0, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 0,  
 0, 0, 0, 1, 0,  
 1, 0, 0, 0, 1])  
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, 1, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 0, 0, 1, 0, 1,  
 1, 0, 0, 0, 1,  
 0, 1, 0, 1, 0])  
test[test == 0] = -1  
output = net.sim([test])  
print((output[0] == target[3]).all(), 'Sim. steps', len(net.layers[0].outs))

Результат виконання програми:

****

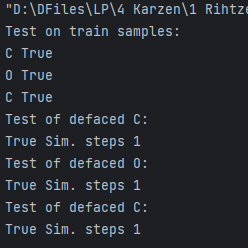
Подана нейромережа показала гарний результат при роботі з матрицями бінарних даних і може вгадувати літери навіть із помилково введеними елементами матриці.

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

Лістинг програми:

import numpy as np  
import neurolab as nl  
  
# С О С  
target = [[0, 1, 1, 1, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 0, 0, 0,  
 0, 1, 1, 1, 0],  
  
 [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],  
  
 [0, 1, 1, 1, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 0, 0, 0,  
 1, 0, 0, 0, 0,  
 0, 1, 1, 1, 0]]  
  
chars = ['C', 'O', 'C']  
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(output)):  
 print(chars[i], (output[i] == target[i]).all())  
  
print("Test of defaced С:")  
test = np.asfarray([0, 1, 1, 1, 0,  
 1, 1, 0, 0, 0,  
 0, 0, 1, 0, 0,  
 1, 0, 0, 0, 0,  
 0, 1, 0, 1, 0])  
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 О:")  
test = np.asfarray([0, 1, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 0, 0, 0, 1,  
 1, 0, 1, 1, 1,  
 0, 1, 0, 0, 0])  
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 С:")  
test = np.asfarray([0, 1, 1, 1, 0,  
 1, 1, 0, 0, 0,  
 0, 0, 1, 0, 0,  
 1, 0, 0, 0, 0,  
 0, 1, 0, 1, 0])  
test[test == 0] = -1  
output = net.sim([test])  
print((output[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))

Результат виконання програми:

****

В результаті вдалося навчити нейромережу розпізнавати власні ініціали з невеликою кількістю помилок.

***Висновки:***  в ході лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python було опановано деякі типи нейронних мереж.