**ЛАБОРАТОРНА РОБОТА №6**

**ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ**

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

**Хід роботи:**

**Завдання 1: Ознайомлення з Рекурентними нейронними мережами**

Лістинг програми (файл LR\_6\_task\_1.py):

import numpy as np  
import random  
  
from rnn 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:

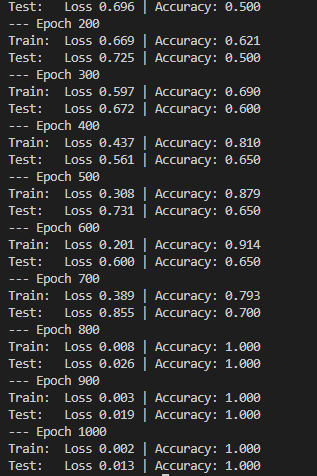


Рис. 1. Результат виконання

Зробіть висновок

Під час виконання завдання була створена рекурентна нейронна мережа, яка успішно навчилася.

**Завдання 2: Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))**

Лістинг програми:

import neurolab as nl  
import numpy as np  
import matplotlib.pyplot as plt  
  
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)  
  
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 для незбалансованих даних:

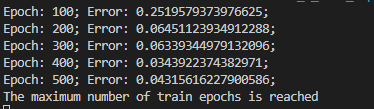
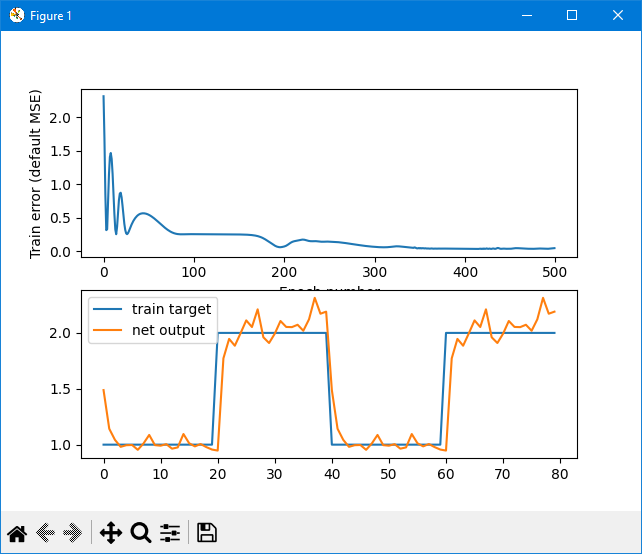


Рис. 2.1 – 2.2. Результат виконання

Зробіть висновок

Під час виконання завдання помилка навчання зменшилася до 0.0565 після 500 ітерацій, що свідчить про ефективність навчання. Досягнута максимальна кількість епох, і тепер мережу можна використовувати для передбачення або аналізу подібних даних.

**Завдання 3: Дослідження нейронної мережі Хемінга (Hemming Recurrent network)**

Проведіть дослідження нейронної мережі Хемінга (Hemming Recurrent network). Для цього виконайте наступні рекомендації.

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]]  
  
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 для метрики precision:

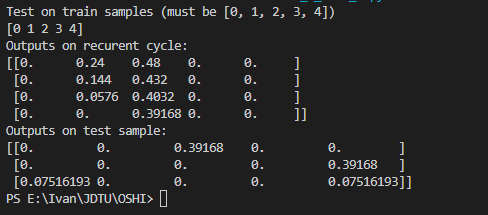


Рис. 3. Результат виконання

**Завдання 4: Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop)**

Необхідно навчити рекурентну нейронну мережу Хопфілда розпізнавати букви слова.

Лістинг програми:

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(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:

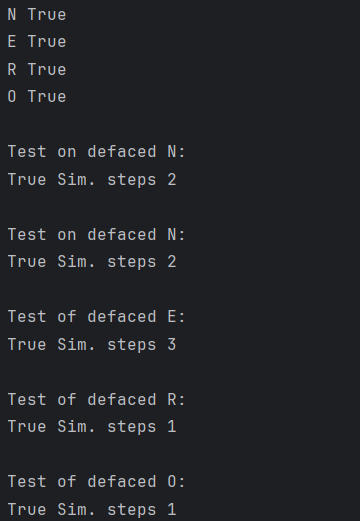


Рис. 4.1. Результат виконання, інформація з вікна терміналу

Протестуйте навчену нейронну мережу Хопфілда. Для цього будемо вважати, що при відображенні букви N були помилки (деякі білі піксели стали чорними і навпаки). Якщо навчання пройшло правильно то мережа при невеликій кількості помилок буде вгадувати букву правильно. Лістинг програми (доповнення):

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:

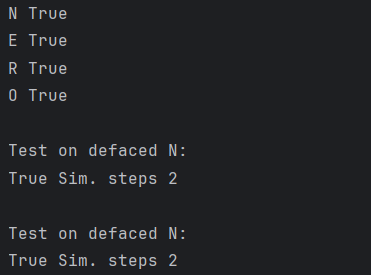


Рис. 4.2. Результат виконання, інформація з вікна терміналу

Спробуйте протестувати інші букви з помилками. Код тесту та результати з вікна терміналу занесіть у звіт.

Лістинг програми (доповнення):

print("\nTest 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("\nTest 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("\nTest 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:

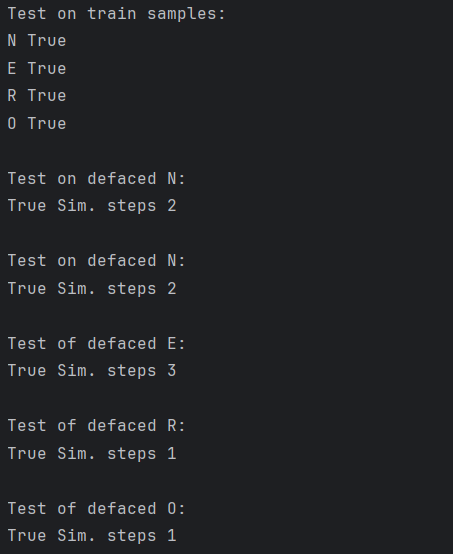


Рис. 4.3. Результат виконання

Зробіть висновок

В результаті виконання програми було успішно навчено рекурентну нейронну мережу Хопфілда розпізнавати букви "NERO" та їх модифіковані версії. Мережа ефективно розпізнала навчальні букви і впоралася зі спотвореними версіями, демонструючи їхню стійкість до змін.

**Завдання 5: Побудова багатошарової нейронної мережі**

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.

Лістинг програми:

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

Результат виконання:

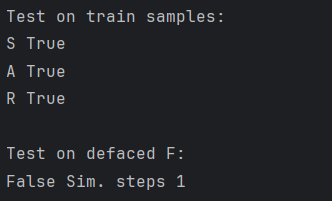


Рис. 5.1. Результат виконання

***Висновки***: Під час виконання лабораторної роботи було проведено дослідження різних типів нейронних мереж, використовуючи спеціалізовані бібліотеки та мову програмування Python.