ЛАБОРАТОРНА РОБОТА № 6

***Тема: «***ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ*»*

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

**Хід роботи**

Посилання на програмнй код на Github: <https://github.com/dengaevsky/Labs_AI/tree/main/lab6>

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

Лістинг класу RNN у файлі rnn.py:

**import** numpy **as** np  
**from** numpy.random **import** randn  
  
  
**class** RNN:  
 # A many-to-one Vanilla Recurrent Neural Network.  
  
 **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  
  
 # Biases  
 self.bh = np.zeros((hidden\_size, 1))  
 self.by = np.zeros((output\_size, 1))  
  
 **def** forward(self, inputs):  
 *'''  
 Perform a forward pass of the RNN using the given inputs.  
 Returns the final output and hidden state.  
 - inputs is an array of one hot vectors with shape (input\_size, 1).  
 '''* h = np.zeros((self.Whh.shape[0], 1))  
  
 self.last\_inputs = inputs  
 self.last\_hs = { 0: h }  
  
 # Perform each step of the 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):  
 *'''  
 Perform a backward pass of the RNN.  
 - d\_y (dL/dy) has shape (output\_size, 1).  
 - learn\_rate is a float.  
 '''* n = len(self.last\_inputs)  
  
 # Calculate dL/dWhy and dL/dby.  
 d\_Why = d\_y @ self.last\_hs[n].T  
 d\_by = d\_y  
  
 # Initialize dL/dWhh, dL/dWxh, and dL/dbh to zero.  
 d\_Whh = np.zeros(self.Whh.shape)  
 d\_Wxh = np.zeros(self.Wxh.shape)  
 d\_bh = np.zeros(self.bh.shape)  
  
 # Calculate dL/dh for the last h.  
 # dL/dh = dL/dy \* dy/dh  
 d\_h = self.Why.T @ d\_y  
  
 # Backpropagate through time.  
 **for** t **in** reversed(range(n)):  
 # An intermediate value: 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  
  
 # Next dL/dh = dL/dh \* (1 - h^2) \* Whh  
 d\_h = self.Whh @ temp  
  
 # Clip to prevent exploding gradients.  
 **for** d **in** [d\_Wxh, d\_Whh, d\_Why, d\_bh, d\_by]:  
 np.clip(d, -1, 1, out=d)  
  
 # Update weights and biases using gradient descent.  
 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

Top of Form

Лістинг програмного коду у файлі LR\_6\_task\_1.py:

**import** random  
**import** numpy **as** np  
  
**from** rnn **import** RNN  
**from** data **import** train\_data  
  
# Створення словника  
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)  
  
# Призначення індексів кожному слову  
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):  
 *'''  
 Повертає масив унітарних векторів  
 які представляють слова у введеному рядку тексту  
 - текст є рядком string  
 - Унітарний вектор має форму (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):  
 # Застосування функції Softmax для вхідного масиву  
 **return** np.exp(xs) / sum(np.exp(xs))  
  
# Ініціалізація нашої рекурентної нейронної мережі RNN  
rnn = RNN(vocab\_size, 2)  
  
  
**def** processData(data, backprop=**True**):  
 *'''  
 Повернення втрат RNN і точності для даних  
 - дані подані як словник, що відображує текст як True або False.  
 - backprop визначає, чи потрібно використовувати звороднє розподілення  
 '''* items = list(data.items())  
 random.shuffle(items)  
  
 loss = 0  
 num\_correct = 0  
  
 **for** x, y **in** items:  
 inputs = createInputs(x)  
 target = int(y)  
  
 # Пряме розподілення  
 out, \_ = rnn.forward(inputs)  
 probs = softmax(out)  
  
 # Обчислення втрат / точності  
 loss -= np.log(probs[target])  
 num\_correct += int(np.argmax(probs) == target)  
  
 **if** backprop:  
 # Створення dL/dy  
 d\_L\_d\_y = probs  
 d\_L\_d\_y[target] -= 1  
  
 # Зворотне розподілення  
 rnn.backprop(d\_L\_d\_y)  
  
 **return** loss / len(data), num\_correct / len(data)

Лістинг програмного коду у файлі main.py:

**from** tasks.LR\_6\_task\_1 **import** processData  
**from** data **import** train\_data, test\_data  
  
  
**def** train():  
 **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))  
  
  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 train()

Результат виконання програми:

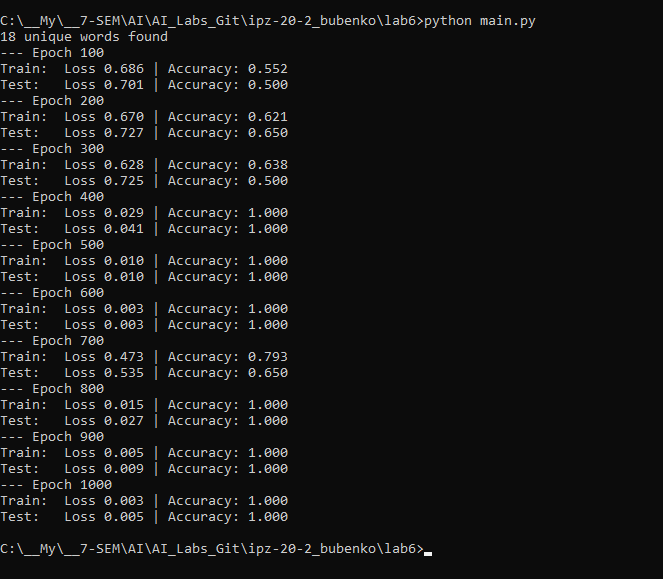


Рис. 1. Результат виконання програми

***Висновки до завдання***

У результаті проведення 1000 епох навчання спостерігалося велике поліпшення ефективності моделі. Починаючи з низької точності та великого втрат в перших епохах, модель прогресувала до високої точності під кінець навчання. На останніх етапах точність досягла 100% на якості тренувального, так і тестового наборів даних. Це свідчить про те, що модель ефективно навчилася аналізувати дані.

***Завдання 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:

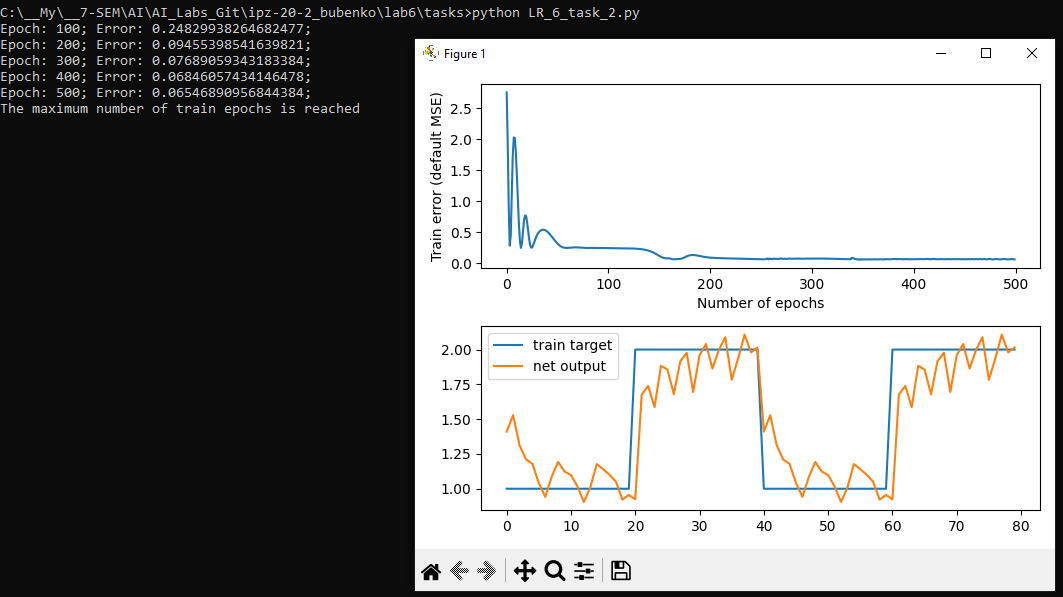


Рис. 2. Результат виконання програми

***Висновки до завдання***

За допомогою бібілотеки neurolab було створено нейронну мережу високої точності. Нейронна мережа навчилася апроксимувати дані і досягла задовільного рівня помилки тренування.

***Завдання 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 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 програми:

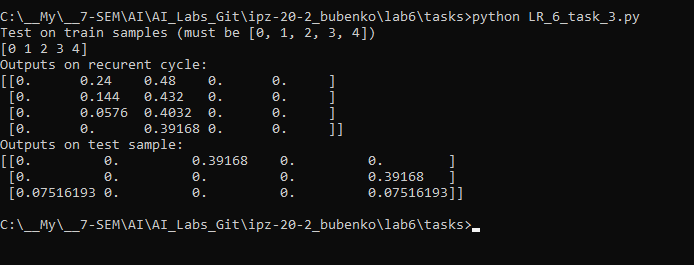


Рис. 3. Результат виконання програми

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

Лістинг програмного коду для створення та навчання нейронної мережі Хопфілда:

**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  
  
# Створення та тренування мережі  
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 створення та навчання нейронної мережі Хопфілда:

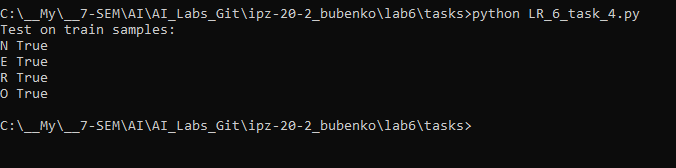


Рис. 4.1. Результат створення та навчання нейронної мережі Хопфілда

Протестуємо навчену нейронну мережу Хопфілда. Вважатимемо, що при відображені літери N були помилки

Лістинг програмного коду для тестування:

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 програми:

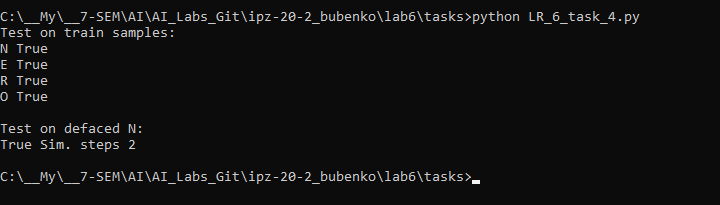


Рис. 4.2. Результат виконання програми

Лістинг програми для тестування літер E, R, O:

# Тестування нейронної мережі  
**def** test\_defaced\_letter(letter, target, net):  
 test = np.asfarray(letter)  
 test[test == 0] = -1  
 output = net.sim([test])  
 result = (output[0] == target).all()  
 steps = len(net.layers[0].outs)  
 **return** result, steps  
  
target\_letters = [target[0], target[1], target[2], target[3]]  
  
print("\nTest on defaced N:")  
result, steps = test\_defaced\_letter([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], target\_letters[0], net)  
print(result, 'Sim. steps', steps)  
  
print("\nTest on defaced E:")  
result, steps = test\_defaced\_letter([0, 0, 0, 0, 0,  
 0, 1, 1, 1, 1,  
 0, 1, 1, 1, 1,  
 0, 1, 1, 1, 1,  
 0, 0, 0, 0, 0], target\_letters[1], net)  
print(result, 'Sim. steps', steps)  
  
print("\nTest of defaced R:")  
result, steps = test\_defaced\_letter([1, 1, 0, 0, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 0,  
 0, 1, 0, 1, 0,  
 1, 0, 0, 0, 1], target\_letters[2], net)  
print(result, 'Sim. steps', steps)  
  
print("\nTest of defaced O:")  
result, steps = test\_defaced\_letter([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], target\_letters[3], net)  
print(result, 'Sim. steps', steps)

Результат виконання програми:

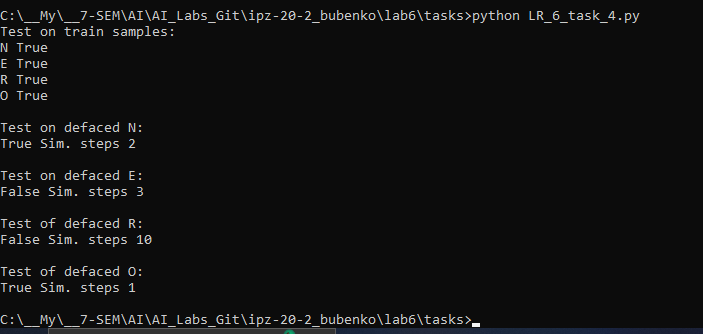


Рис. 4.3. Результат виконання програми

***Висновки до завдання***

Отже, отримані результати тестування мережі Хопфілда свідчать про її успішність у відновленні тренувальних шаблонів, таких як літери N, E, R та O. Однак, при деформації зразків мережа може виявляти обмежену ефективність, особливо в розпізнаванні деформованих літер E та R.

***Завдання 5.*** Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних.

По аналогії з попереднім завданням візьміть перші букви ваших прізвища, шм’я 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.

Лістинг програми:

**import** numpy **as** np  
**import** neurolab **as** nl  
  
# Оновлені тренувальні та тестові дані для літер Б, О, В  
target = np.array([[1, 1, 1, 1, 1,  
 1, 0, 0, 0, 0,  
 1, 1, 1, 1, 1, # Б  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 1],  
  
 [0, 1, 0, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 0, 1, 0, 0,  
 1, 0, 0, 1, 1,  
 0, 1, 1, 1, 0], # О  
  
 [1, 1, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 0, # В  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 0]])  
  
target\_defaced = np.array([[1, 1, 1, 1, 1,  
 0, 0, 0, 0, 0,  
 1, 1, 1, 1, 1, # Б  
 1, 0, 0, 1, 1,  
 1, 1, 1, 1, 1],  
  
 [0, 1, 0, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 0, 0,  
 1, 0, 0, 1, 1,  
 0, 1, 0, 1, 0], # О  
  
 [1, 1, 1, 1, 0,  
 1, 0, 0, 0, 1,  
 1, 1, 1, 1, 0, # В  
 1, 0, 0, 0, 1,  
 1, 0, 1, 1, 0]])  
  
chars = ['Б', 'О', 'В']  
target[target == 0] = -1  
  
# Створення та тренування мережі  
net = nl.net.newhop(target.reshape(-1, 25)) # Плоске представлення для мережі Хопфілда  
output = net.sim(target.reshape(-1, 25))  
print("Test on train samples:")  
**for** i **in** range(len(target)):  
 print(chars[i], (output[i] == target[i]).all())  
  
  
# Тестування нейронної мережі для всіх трьох літер  
**def** test\_defaced\_letter(letter, target, net):  
 test = np.asfarray(letter).flatten()  
 test[test == 0] = -1  
 output = net.sim([test])  
 result = (output[0] == target.flatten()).all()  
 steps = len(net.layers[0].outs)  
 **return** result, steps  
  
  
target\_letters = [target[0], target[1], target[2]]  
  
# Тестування для літер Б, О, В  
**for** i **in** range(len(target\_letters)):  
 print(f"\nTest on defaced {chars[i]}:")  
 result, steps = test\_defaced\_letter(target\_letters[i], target\_letters[i], net)  
 print(result, 'Sim. steps', steps)

Результат виконання програми:

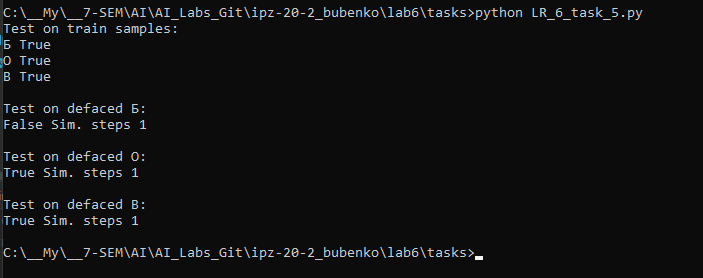


Рис. 5. Результат виконання програм

***Висновки до завдання***

Отже, можна дійти висновку, що як і в попередньому завданні при зміні 1-2 пікселів нейронна мережа Хопфілда правильно ідентифікує літеру.

***Висновок:*** у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python навчився та дослідив деякі типи нейронних мереж.