**Лабораторна робота №6**

**Тема:** ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

**Репозиторій:** https://github.com/danielwinter13/AI-Zyma-IPZ-20-1./tree/master/lr6

**Хід роботи:**

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

**Лістинг програми**

import random  
  
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  
  
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))  
  
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

Зображення, що містить текст

Автоматично згенерований опис

**Рис.1** Виконання файлу main.py

Зображення, що містить текст, надворі

Автоматично згенерований опис

**Рис.2** Виконання файлу task1.py

Ми спостерігаємо повідомлення на рисунку 1-2 “18 unique words found” це означає, що зміна vocab тепер буде мати перелік всіх слів, які вживаються щонайменше в одному навчальному тексті. Рекурентна нейронна мережа не розрізняє слів – лише числа. Тому у словнику 18 унікальних слів, кожне буде 18-мірним унітарним вектором. І далі відбувається тренування мережі. Виведення кожної сотої епохи для відслідковування прогресу

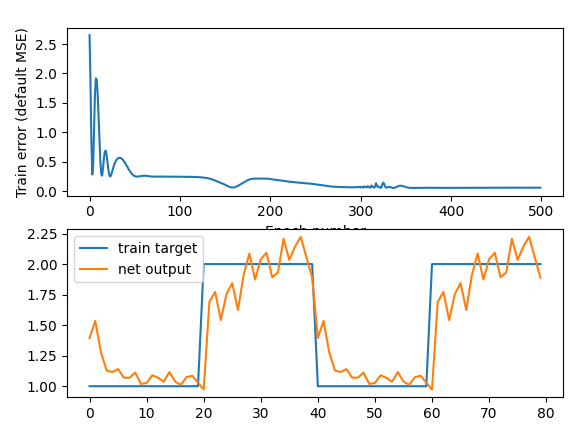
**Завдання 2.** Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))

import neurolab as nl  
import numpy as np  
  
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()  
# 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()

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 3** Виконання файлу task2.py



**Рис. 4** Виконання програми

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

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)

Зображення, що містить текст, електроніка, знімок екрана

Автоматично згенерований опис

**Рис. 5** Виконання програми

**Завдання 2.4.** Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop)

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 E:")  
test =np.asfarray(  
 [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],  
 )  
test[test==0] = -1  
out = net.sim([test])  
print ((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 6** Виконання програми

print("\nTest on defaced A:")  
test =np.asfarray(  
 [0, 0, 0, 0, 0,  
 1, 1, 1, 1, 1,  
 0, 1, 1, 1, 1,  
 0, 1, 1, 1, 1,  
 0, 0, 0, 0, 1],  
 )  
test[test==0] = -1  
out = net.sim([test])  
print ((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 7** Виконання програми

print("\nTest on defaced M:")  
test =np.asfarray(  
 [0, 0, 0, 0, 0,  
 1, 1, 0, 0, 1,  
 1, 1, 1, 1, 1,  
 0, 1, 1, 1, 1,  
 1, 0, 0, 0, 1],  
 )  
test[test==0] = -1  
out = net.sim([test])  
print ((out[0] == target[1]).all(), 'Sim. steps',len(net.layers[0].outs))

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 8** Виконання програми

Як бачимo, навчання пройшло правильно і мережа при невеликій кількості помилок вгадала букви правильно.

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

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

Зображення, що містить текст

Автоматично згенерований опис

**Рис. 9** Виконання програми

Зробив деякі заміни. Результат був True. Якщо навчання пройшло правильно то мережа при невеликій кількості помилок буде вгадувати букву правильно. Значить все вірно.

**Висновок:** під час виконання лабараторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python навчився досліджувати деякі типи нейронних мереж***.***