ЛАБОРАТОРНА РОБОТА 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

Лістинг програми:

```
import random
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 = np.random.randn(hidden_size, hidden_size) / 1000
        self.Wxh = np.random.randn(hidden_size, input_size) / 1000
        self.Why = np.random.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):

        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):

        n = len(self.last_inputs)
```

					ДУ «Житомирська політехніка».22.121.14.000 – Лр6		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Сірач А.С.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів ZZ
Перевір.		Філіпов В.О.					
Керівник						ФІКТ Гр. ІПЗ-19-3[2]	
Н. контр.							
Зав. каф.							

```
        # 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)}


def createInputs(text):
    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):
    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):

        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):

        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
```

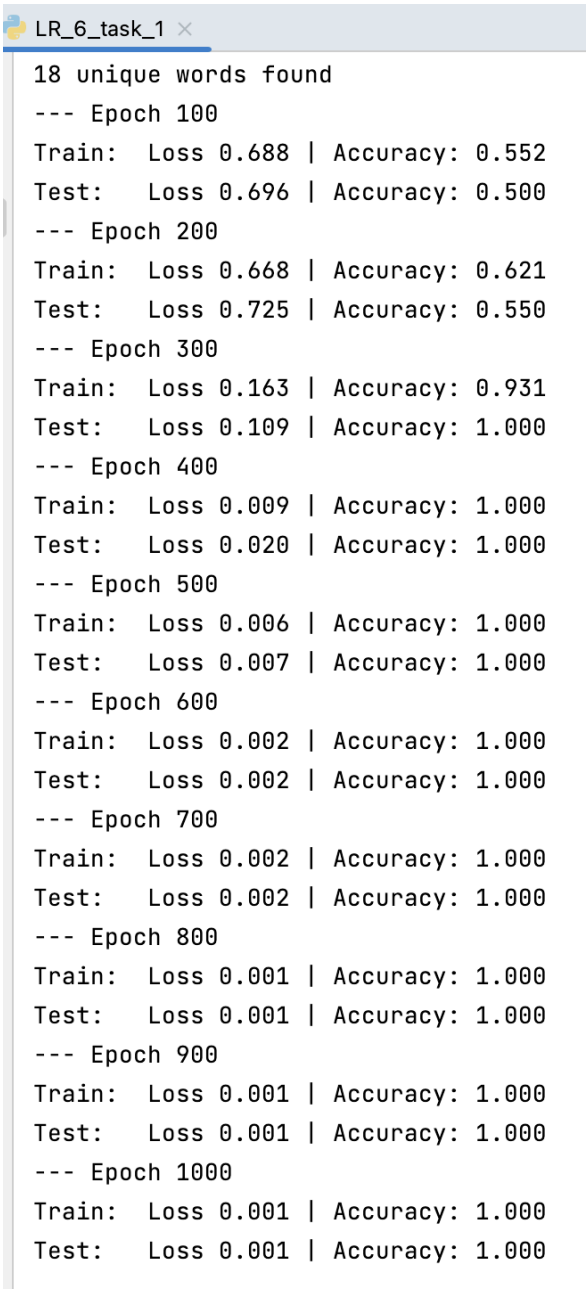

Рис. 6.1 Виконання файлу LR_6_task_1.py

Висновок: На рисунку 6.1 зверху можна побачити виведенне повідомлення “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.

Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))

Лістинг програми:

```
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()
```

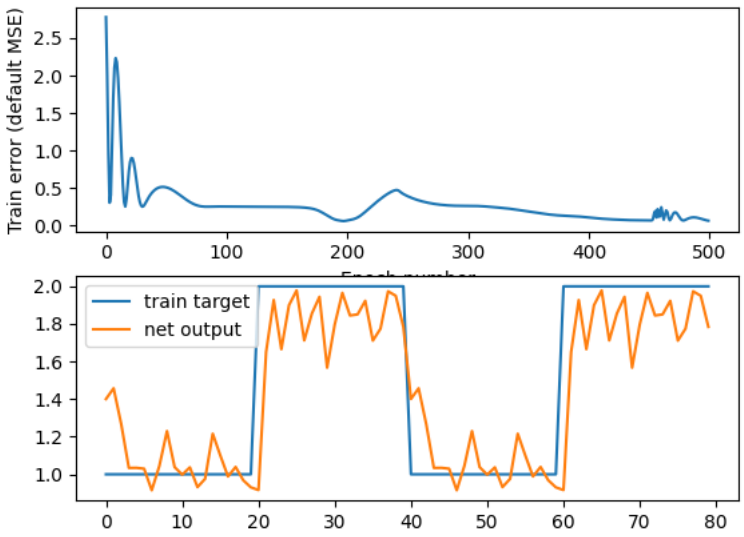

Рис. 6. 2 Виконання програми

```
Epoch: 100; Error: 0.25183930006424604;
Epoch: 200; Error: 0.06368298617167414;
Epoch: 300; Error: 0.26187672375580323;
Epoch: 400; Error: 0.10895572106802684;
Epoch: 500; Error: 0.06428923243796886;
The maximum number of train epochs is reached
```

Рис. 6. 3 Виконання програми

Висновок: Під час виконання 2 завдання було імпортовано бібліотеки neurolab та numpy, створено модель сигналу для навчання мережі та мережу з двома прошарками. В результаті виконання програмного коду було отримано результати, які можна побачити на рис. 2-3

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 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)
```

```
Test on train samples (must be [0, 1, 2, 3, 4])
[0 1 2 3 4]
Outputs on recurent cycle:
[[0.      0.24    0.48    0.      0.     ]
 [0.      0.144   0.432   0.      0.     ]
 [0.      0.0576  0.4032  0.      0.     ]
 [0.      0.      0.39168 0.      0.     ]]
Outputs on test sample:
[[0.         0.         0.39168    0.         0.        ]
 [0.         0.         0.         0.         0.39168   ]
 [0.07516193 0.         0.         0.         0.07516193]]
```

Рис. 6. 4 Виконання програми

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

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 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))
```

```
Test on train samples:
N True
E True
R True
O True

Test on defaced E:
False Sim. steps 3
```

Рис. 6. 5 Виконання програми

Висновок: Під час виконання 4 завдання було імпортовано бібліотеки neurolab та numpy, було внесено вхідні дані у вигляді складного списку та подання їх в такій формі, яка сприймається функцією з бібліотеки. Було створенно та навчено нейронну мережу Хопфілда розпізнавати літери. В результаті викоання коду було отримано результат True, що означає позитивний результат навчання мережі. Якщо навчання пройшло правильно то мережа при невеликій кількості помилок буде вгадувати букву правильно.

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

target =  [[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],

            [0, 0, 1, 0, 0,
            0, 1, 0, 1, 0,
            1, 0, 0, 0, 1,
```

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

```
Test on train samples:
C True
A True
C True

Test on defaced C:
True Sim. steps 1
```

Рис. 6. 6 Виконання програми

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

Посилання на GitHub: https://github.com/annasirach/AI_IPZ193_Sirach