ЛАБОРАТОРНА РОБОТА № 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи:

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

Хід роботи:

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

Лістинг коду:

```
from data import train_data, test_data
import numpy as np
import random
from rnn import RNN

# Створити словник
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)  # знайдено 18 унікальних слів

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 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 зворотнє розподілення
    '''
    items = list(data.items())
    random.shuffle(items)

    loss = 0
```

Зм.						
Розроб.	Щербак М.Ю.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 10
Перевір.	Голенко М.Ю.					
Керівник					ФІКТ Гр. ІПЗ-20-2[2]	
Н. контр.						
Зав. каф.						

```
    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:
            # Создание 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)


def softmax(xs):
    # Застосування функції Softmax для вхідного масиву
    return np.exp(xs) / sum(np.exp(xs))


# Призначити індекс 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'])  # 16 (це може змінитися)
print(idx_to_word[0])  # сумно (це може змінитися)

# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN(vocab_size, 2)

inputs = createInputs('i am very good')
out, h = rnn.forward(inputs)
probs = softmax(out)
print(probs)  # [[0.50000095], [0.49999905]]

# Цикл тренування
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[0], train_acc))

        test_loss, test_acc = processData(test_data, backprop=False)
        print('Test:\tLoss %.3f | Accuracy: %.3f' % (test_loss[0], test_acc))
```

Результат виконання:

```
PS D:\proggers\AI\lab6> & C:/Users/serbm/AppData/Local/Programs/Python/Python312/python.exe d:/proggers/AI/lab6/LR_6_tas
18 unique words found
15
i
[[0.50000137]
 [0.49999863]]
--- Epoch 100
Train:  Loss 0.687 | Accuracy: 0.552
Test:   Loss 0.700 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.669 | Accuracy: 0.638
Test:   Loss 0.718 | Accuracy: 0.700
--- Epoch 300
Train:  Loss 0.182 | Accuracy: 0.931
Test:   Loss 0.142 | Accuracy: 1.000
--- Epoch 400
Train:  Loss 0.014 | Accuracy: 1.000
Test:   Loss 0.035 | Accuracy: 1.000
--- Epoch 500
Train:  Loss 0.007 | Accuracy: 1.000
Test:   Loss 0.007 | Accuracy: 1.000
--- Epoch 600
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.004 | Accuracy: 1.000
--- Epoch 700
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.003 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.003 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 1000
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
PS D:\proggers\AI\lab6>
```

Висновок: З результату виведення можна зробити висновок, що мережа добре навчається та досягає 100% точності (Accuracy) вже на 400 епосі навчання.

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

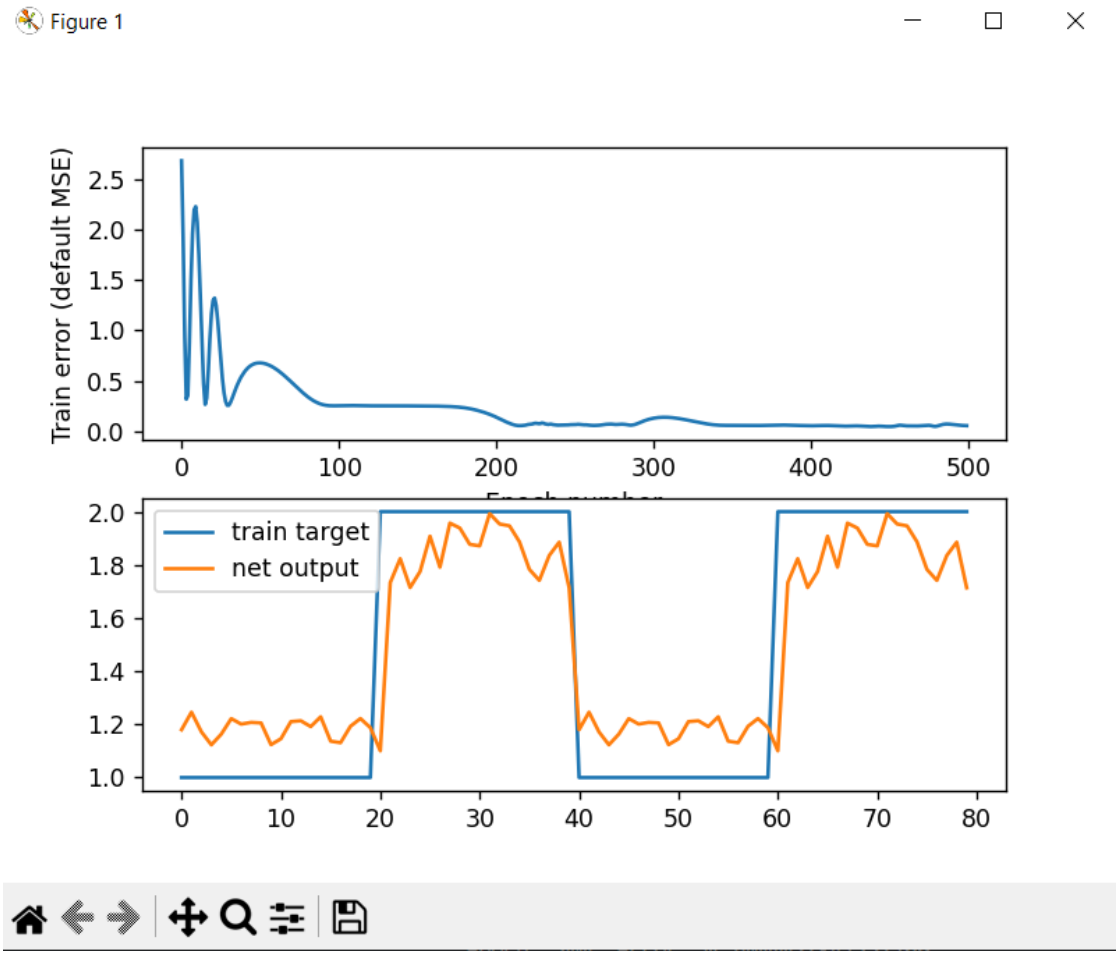
Лістинг коду:

```
import neurolab as nl
import numpy as np
import pylab 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()])

# Ініціалізуйте 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)
# Запустіть мережу
output = net.sim(input)

# Побудова графіків
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()
```

Результат виконання:

```
Epoch: 100; Error: 0.2511024491121864;
Epoch: 200; Error: 0.14448094467977532;
Epoch: 300; Error: 0.12374246576611281;
Epoch: 400; Error: 0.05157570946995053;
Epoch: 500; Error: 0.05241445585670722;
The maximum number of train epochs is reached
```



Висновок: З графіку помилок можна зробити висновок, що мережа зменшує кількість помилок протягом навчання. Нажаль, не було досягнуто цільового значення 0.01 помилок, та програма закінчила навчання після 500 епох.
З графіку апроксимації сигналу можна сказати, що мережа відтворює цільові патерни, хоч і не дуже точно.

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

Лістинг коду:

```
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:

```
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]]
```

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

Лістинг коду:

```
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

# Create and train network
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))

print("\nTest on defaced O:")
test_o = np.asfarray([1, 1, 1, 1, 1,
                      1, 0, 0, 0, 0,
                      1, 0, 0, 0, 0,
                      1, 0, 0, 0, 0,
                      1, 0, 0, 0, 0])

test_o[test_o == 0] = -1
out_o = net.sim([test_o])
print((out_o[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Результат виконання:

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

```
Test on defaced N:
True Sim. steps 2
```

```
Test on defaced O:
True Sim. steps 3
```

Висновок: Хопфілдова мережа, яку навчили на конкретних зразках, проявляє високу ефективність у відновленні літер, навіть у випадку невеликих змін чи пошкоджень.

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

Лістинг коду:

```python
import numpy as np
import neurolab as nl

# Щ М Ю
target = [[1, 1, 1, 1, 0,
           1, 1, 1, 1, 0,
           1, 1, 1, 1, 0,
           1, 1, 1, 1, 0,
           1, 1, 1, 1, 1],

          [1, 1, 0, 1, 1,
           1, 0, 1, 0, 1,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1],

          [1, 0, 0, 1, 0,
           1, 0, 1, 0, 1,
           1, 1, 1, 0, 1,
           1, 0, 1, 0, 1,
           1, 0, 0, 1, 0]]

chars = ['Щ', 'М', 'Ю']
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(target)):
    print(chars[i], (output[i] == target[i]).all())


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


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

Результат виконання:

```
Test on train samples:
Щ True
М True
Ю True

Test on defaced 'Ю':
False Sim. steps 2

Test on defaced 'М':
False Sim. steps 1
```

Github: *https://github.com/mtvi/ipz202_Shcherback_lab6*

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