ЛАБОРАТОРНА РОБОТА №6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Посилання на проект: https://github.com/ipz202-rev/AI-lab6

Хід роботи:

Завдання №6.1. Ознайомлення з Рекурентними нейронними мережами.

Лістинг файлу LR_6_task_1.py:

```
import numpy as np
import random
import warnings
from numpy.random import randn
from data import train_data, test_data

warnings.filterwarnings("ignore")


class RNN:
    # Класична рекурентна нейронна мережа

    def __init__(self, input_size, output_size, hidden_size=64):
        # Вес
        self.Whh = randn(hidden_size, hidden_size) / 1000
        self.Wxh = randn(hidden_size, input_size) / 1000
        self.Why = randn(output_size, hidden_size) / 1000

        # Зміщення
        self.bh = np.zeros((hidden_size, 1))
        self.by = np.zeros((output_size, 1))

    def forward(self, inputs):
        '''
        Виконання передачі нейронної мережі за допомогою вхідних даних
        Повернення результатів виведення та прихованого стану
        Вивід - це масив одного унітарного вектора з формою (input_size, 1)
        '''
        h = np.zeros((self.Whh.shape[0], 1))

        self.last_inputs = inputs
        self.last_hs = {0: h}

        # Виконання кожного кроку нейронної мережі RNN
        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
```

					ДУ«Житомирська політехніка».23.121.23.000 – Лр6		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Рябова Є.В.			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 12
Перевір.		Голенко М.Ю.					
Керівник						ФІКТ Гр. ІПЗ-20-2[2]	
Н. контр.							
Зав. каф.							

```
            self.last_hs[i + 1] = h

        # Підрахунок значення виводу
        y = self.Why @ h + self.by

        return y, h

    def backprop(self, d_y, learn_rate=2e-2):
        '''
        Виконання фази зворотного розповсюдження мережі RNN.
        - d_y (dL/dy) має форму (output_size, 1).
        - learn_rate є дійсним числом float.
        '''
        n = len(self.last_inputs)

        # ОБчислення dL/dWhy і dL/dby.
        d_Why = d_y @ self.last_hs[n].T
        d_by = d_y

        # Ініціалізація dL/dWhh, dL/dWxh, і dL/dbh до нуля.
        d_Whh = np.zeros(self.Whh.shape)
        d_Wxh = np.zeros(self.Wxh.shape)
        d_bh = np.zeros(self.bh.shape)

        # Обчислення dL/dh для останнього h.
        d_h = self.Why.T @ d_y

        # Зворотне розповсюдження по часу.
        for t in reversed(range(n)):
            # Среднее значение: 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

            # Далее dL/dh = dL/dh * (1 - h^2) * Whh
            d_h = self.Whh @ temp

        # Відсікаємо, щоб попередити розрив градієнтів.
        for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
            np.clip(d, -1, 1, out=d)

        # Обновляємо ваги і зміщення з використанням градієнтного спуску.
        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


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
    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))


# 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

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, train_acc))

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

```
D:\course-4\semester-1\ai\labó_project\venv\Scripts\python.exe
18 unique words found
1
am
[[0.49999506]
 [0.50000494]]
--- Epoch 100
Train:  Loss 0.689 | Accuracy: 0.552
Test:   Loss 0.697 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.667 | Accuracy: 0.638
Test:   Loss 0.712 | Accuracy: 0.700
--- Epoch 300
Train:  Loss 0.620 | Accuracy: 0.621
Test:   Loss 0.680 | Accuracy: 0.450
--- Epoch 400
Train:  Loss 0.293 | Accuracy: 0.879
Test:   Loss 0.150 | Accuracy: 1.000
--- Epoch 500
Train:  Loss 0.013 | Accuracy: 1.000
Test:   Loss 0.012 | Accuracy: 1.000
--- Epoch 600
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.003 | Accuracy: 1.000
--- Epoch 700
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.001 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.001 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.001 | Accuracy: 1.000
--- Epoch 1000
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.000 | Accuracy: 1.000

Process finished with exit code 0
```

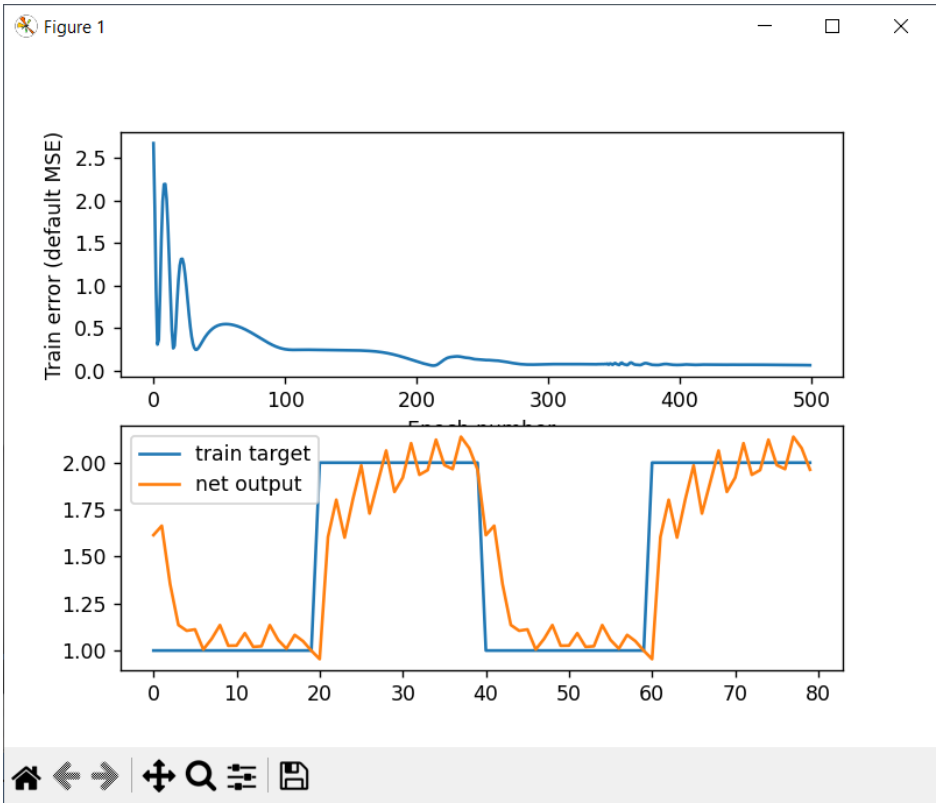
Рис.6.1.1. Результат виконання завдання.

У результаті виконання завдання, було створено просту рекурентну нейронну мережу для класифікації тексту з високою точністю на тестовому наборі. За 1000 епох навчання точність навчання і тестування досягає 100%, що може вказувати на можливе перенавчання.

Завдання №6.2. Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm)).

Лістинг файлу LR_6_task_2.py:

```
import neurolab as nl
import numpy as np
import matplotlib.pylab as pl

# Створення моелей 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 ваґів
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()
```




```
D:\course-4\semester-1\ai\lab6_project\venv\Scripts\python.exe
Epoch: 100; Error: 0.2582607994294791;
Epoch: 200; Error: 0.11573905135250948;
Epoch: 300; Error: 0.07886116782505774;
Epoch: 400; Error: 0.07065206041549395;
Epoch: 500; Error: 0.06745496381167852;
The maximum number of train epochs is reached

Process finished with exit code 0
```

Рис.6.2.1 – 6.2.2. Результат виконання завдання.

Рекурентна нейронна мережа Елмана показала зменшення помилки тренування з 0.258 до 0.067 протягом 500 епох, що свідчить про ефективне навчання.

Завдання №6.3. Дослідження нейронної мережі Хемінга (Hemming ecurrent network).

Лістинг файлу LR_6_task_3.py:

```
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)
```

```
D:\course-4\semester-1\ai\lab6_project\venv\Scripts\python.exe
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]]

Process finished with exit code 0
```

Рис.6.3.1. Результат виконання завдання.

Завдання №6.4. Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop).

Лістинг файлу LR_6_task_4.py:

```
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())
```

```
D:\course-4\semester-1\ai\lab6_project\venv\Scripts\python.exe
Test on train samples:
N True
E True
R True
O True

Process finished with exit code 0
```

Рис.6.4.1. Результат виконання завдання.

Протестуйте навчену нейронну мережу Хопфілда. Для цього будемо вважати, що при відображенні букви N були помилки (деякі білі пікселі стали чорними і навпаки).

Лістинг файлу LR_6_task_4.py:

```
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))
```

```
D:\course-4\semester-1\ai\lab6_project\venv\Scripts\python.exe
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
True Sim. steps 2

Process finished with exit code 0
```

Рис.6.4.2. Результат виконання завдання.

Лістинг файлу LR_6_task_4.py:

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

print("\nTest on defaced R:")
test = np.asfarray([1, 1, 1, 1, 0,
                    1, 0, 0, 0, 1,
                    1, 1, 0, 1, 0,
                    1, 0, 0, 1, 0,
                    1, 0, 0, 0, 1])
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))

print("\nTest on defaced O:")
test = np.asfarray([0, 1, 1, 1, 1,
                    1, 1, 0, 1, 1,
                    0, 0, 1, 0, 1,
                    0, 0, 0, 0, 1,
                    0, 1, 1, 1, 0])
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[3]).all(), 'Sim. steps', len(net.layers[0].outs))
```

```
D:\course-4\semester-1\ai\lab6_project\venv\Scripts\python.exe
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
True Sim. steps 2

Test on defaced E:
True Sim. steps 3

Test on defaced R:
True Sim. steps 1

Test on defaced O:
True Sim. steps 2

Process finished with exit code 0
```

Рис.6.4.3. Результат виконання завдання.

Рекурентна нейронна мережа Хопфілда успішно визнає чотири літери (N, E, R, O) за вихідними шаблонами, продемонструвавши повну збіжність. Навіть при поданні спотворених зображень літер, мережа вірно відновлює оригінальні шаблони з невеликою кількістю ітерацій.

Завдання №6.5. Побудова багатошарової нейронної мережі.

Лістинг файлу LR_6_task_5.py:

```
import numpy as np
import neurolab as nl

# P Є B
target = [[1, 1, 1, 0, 0,
           1, 0, 0, 1, 0,
           1, 1, 1, 0, 0,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 0],
          [0, 1, 1, 1, 1,
           1, 0, 0, 0, 0,
           1, 1, 1, 1, 1,
           1, 0, 0, 0, 0,
           0, 1, 1, 1, 1],
          [1, 1, 1, 1, 0,
           1, 0, 0, 0, 1,
           1, 1, 1, 1, 0,
           1, 0, 0, 0, 1,
           1, 1, 1, 1, 0]]

chars = ['P', 'Є', 'B']
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 P:")
test = np.asfarray([1, 1, 1, 1, 0,
                    1, 0, 0, 1, 0,
                    1, 1, 1, 0, 0,
                    1, 0, 0, 0, 0,
                    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))

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

print("\nTest on defaced B:")
test = np.asfarray([1, 1, 1, 1, 0,
                    1, 0, 0, 0, 1,
                    1, 0, 1, 1, 0,
                    1, 0, 0, 0, 1,
                    1, 1, 1, 1, 0])
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))
```

```
D:\course-4\semester-1\ai\labб_project\venv\Scripts\python.exe
Test on train samples:
P True
Є True
B True

Test on defaced P:
True Sim. steps 1

Test on defaced Є:
True Sim. steps 1

Test on defaced B:
True Sim. steps 1

Process finished with exit code 0
```

Рис.6.5.1. Результат виконання завдання.

Висновки: в ході виконання лабораторної роботи було досліджено деякі типи нейронних мереж, використовуючи спеціалізовані бібліотеки та мову програмування Python.