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 програми:

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


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

# 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))
```

Лістинг програми (***RNN.py***):

```python
import numpy as np
from numpy.random import randn

class RNN:
    # Класична рекурентна нейронна мережа
    def __init__(self, input_size, output_size, hidden_size=64):
        # Вага
        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)
            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 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
```

Результат виконання програми:

```
18 unique words found
13
is
[[0.5000039]
 [0.4999961]]
--- Epoch 100
Train:  Loss 0.688 | Accuracy: 0.552
Test:   Loss 0.695 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.672 | Accuracy: 0.621
Test:   Loss 0.718 | Accuracy: 0.550
--- Epoch 300
Train:  Loss 0.095 | Accuracy: 1.000
Test:   Loss 0.074 | Accuracy: 1.000
--- Epoch 400
Train:  Loss 0.010 | Accuracy: 1.000
Test:   Loss 0.011 | Accuracy: 1.000
--- Epoch 500
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.007 | Accuracy: 1.000
```

Рис. 2.1.1 – Результат виконання завдання.

```
--- Epoch 600
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 700
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.003 | Accuracy: 1.000
--- Epoch 1000
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
```

Рис. 2.1.2 – Результат виконання завдання.

Висновок:

В результаті, можна побачити, що мережа навчається добре, досягаючи 100% точності на тестовому наборі даних після кількох сотень епох навчання.

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

Лістинг програми:

```
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 програми:

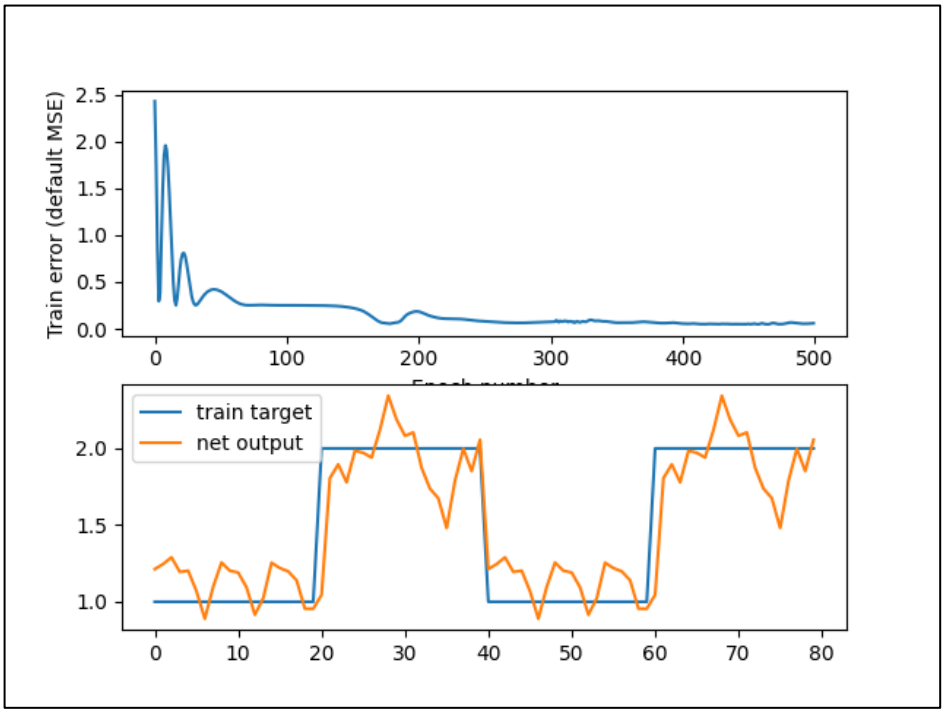

Рис. 2.2.1 – Результат виконання завдання (графік 1).

```
Epoch: 100; Error: 0.2505506790281932;
Epoch: 200; Error: 0.18463180949875296;
Epoch: 300; Error: 0.07386583857744349;
Epoch: 400; Error: 0.05605300703655687;
Epoch: 500; Error: 0.058860858407514614;
The maximum number of train epochs is reached
```

Рис. 2.2.2 – Результат виконання завдання.

Висновок:

Під час тренування виводиться значення помилки на кожні 100 епох. Помилка зменшується протягом епох, і тренування завершується, коли досягнуто цільового значення помилки (goal=0.01) або коли досягнуто максимальну кількість епох (epochs=500).

На підграфіках видно, що помилка тренування спадає з кожною епохою, що свідчить про успішність навчання. У другому підграфіку порівнюються цільові дані та виходи мережі, показуючи, що модель непогано відтворює цільові патерни.

У висновку можна сказати, що штучна нейронна мережа була успішною в навчанні на вхідних сигналах та відтворенні цільових вихідних сигналів, хоча для більш точної оцінки ефективності можна розглянути додаткові метрики та аналіз результатів.

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)
```

Результат виконання програми:

```
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. 2.3.1 – Результат виконання завдання.

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

Лістинг програми:

```
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 E:")
test_e = np.asfarray([1, 1, 1, 1, 1,
                      1, 0, 0, 0, 0,
                      1, 1, 0, 1, 1,
                      1, 0, 0, 0, 1,
                      1, 1, 1, 1, 1])

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

Результат виконання програми:

```
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
```

Рис. 2.4.1 – Результат виконання завдання.

Висновок:

Можна зробити висновок, що мережа Хопфілда, навчена на конкретних шаблонах, виявляється досить ефективною у відновленні літер з невеликими змінами або пошкодженнями.

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

# Г Д С
target = [[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],
          [0, 1, 1, 1, 0,
           0, 1, 0, 1, 0,
           0, 1, 0, 1, 0,
           1, 1, 1, 1, 1,
           1, 0, 0, 0, 1],
          [0, 1, 1, 1, 0,
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 1,
           0, 1, 1, 1, 0]]

chars = ['Г', 'Д', 'С']
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 'Г':")
test_g = np.asfarray([0, 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_g[test_g == 0] = -1
out_g = net.sim([test_g])
print((out_g[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))

########################################

print("\nTest on defaced 'Д':")
test = np.asfarray([0, 1, 1, 1, 0,
                    0, 1, 0, 1, 0,
                    0, 1, 1, 1, 0,
                    1, 1, 1, 1, 1,
                    1, 0, 1, 0, 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:

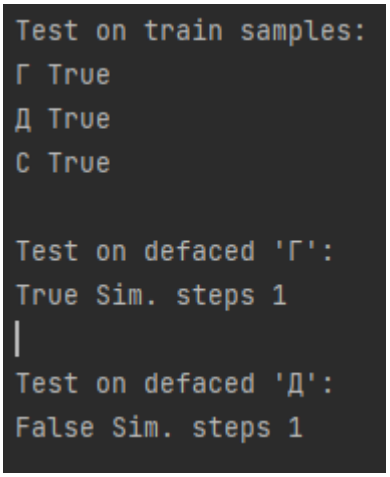

Рис. 2.5.1 – Результат виконання завдання.

Посилання на репозиторій: https://github.com/GrunytskyDmytro/Lab6_AI.git

Висновки по лабораторній роботі: в ході виконання лабораторної роботи використовуючи спеціалізовані бібліотеки та мову програмування Python навчився досліджувати деякі типи нейронних мереж.