ЛАБОРАТОРНА РОБОТА № 6

Тема: «ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ»

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

Хід роботи

Посилання на програмнй код на Github: https://github.com/NightSDay/AI-2023/tree/main/AI-subject/L-6

Завдання 1. Ознайомлення з Рекурентними нейронними мережами.

Лістинг класу RNN у файлі rnn.py:

```
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
```

					ДУ «Житомирська політехніка».23.121.16.000–Лр6			
Змн.	Арк.	№ докум.	Підпис	Дата				
Розроб.		Радченко Д.В.			Звіт з лабораторної роботи	Літ.	Арк.	Аркушів
Перевір.		Голенко М.Ю.					1	12
Керівник						ФІКТ Гр. ІПЗ-20-3		
Н. контр.								
Зав. каф.								

```
    # 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
```

Лістинг програмного коду у файлі LR_6_task_1.py:

```
import random
import numpy as np

from rnn import RNN
from data import train_data
```

```
# Створення словника
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)

# Призначення індексів 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'])
# print(idx_to_word[0])


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

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


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 звороднє 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:
            # Створення 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)
```

Лістинг програмного коду у файлі main.py:

```
from tasks.LR_6_task_1 import processData
from data import train_data, test_data


def train():
    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))


if __name__ == '__main__':
    train()
```

Результат виконання програми:

```
E:\AI-subject\L-6>python main.py
18 unique words found
--- Epoch 100
E:\AI-subject\L-6\main.py:11: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and wi
ll error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated Nu
mPy 1.25.)
  print('Train:\tLoss %.3f | Accuracy: %.3f' % (train_loss, train_acc))
Train:  Loss 0.688 | Accuracy: 0.552
E:\AI-subject\L-6\main.py:14: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and wi
ll error in future. Ensure you extract a single element from your array before performing this operation. (Deprecated Nu
mPy 1.25.)
  print('Test:\tLoss %.3f | Accuracy: %.3f' % (test_loss, test_acc))
Test:   Loss 0.696 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.670 | Accuracy: 0.655
Test:   Loss 0.718 | Accuracy: 0.500
--- Epoch 300
Train:  Loss 0.606 | Accuracy: 0.638
Test:   Loss 0.694 | Accuracy: 0.500
--- Epoch 400
Train:  Loss 0.590 | Accuracy: 0.638
Test:   Loss 0.663 | Accuracy: 0.650
--- Epoch 500
Train:  Loss 0.496 | Accuracy: 0.724
Test:   Loss 0.696 | Accuracy: 0.650
--- Epoch 600
Train:  Loss 0.448 | Accuracy: 0.776
Test:   Loss 0.594 | Accuracy: 0.550
--- Epoch 700
Train:  Loss 0.520 | Accuracy: 0.690
Test:   Loss 0.617 | Accuracy: 0.700
--- Epoch 800
Train:  Loss 0.014 | Accuracy: 1.000
Test:   Loss 0.064 | Accuracy: 0.950
--- Epoch 900
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.073 | Accuracy: 0.950
--- Epoch 1000
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.015 | Accuracy: 1.000

E:\AI-subject\L-6>
```

Рис. 1. Результат виконання програми

Висновки до завдання

У результаті проведення 1000 епох навчання спостерігалося велике поліпшення ефективності моделі. Починаючи з низької точності та великого втрат в перших епохах, модель прогресувала до високої точності під кінець навчання. На останніх етапах точність досягла 100% на якості тренувального, так і тестового наборів даних. Це свідчить про те, що модель ефективно навчилася аналізувати дані.

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

Лістинг програми:

```python
import numpy as np
import neurolab as nl
import matplotlib.pyplot 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()])

# Ініціалізація 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)

# Запуск мережі
output = net.sim(input)

# Побудова графіків
pl.subplot(211)
pl.plot(error)
pl.xlabel('Number of epochs')
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.tight_layout(w_pad=1.5)
pl.show()
```

Результат виконання програми:

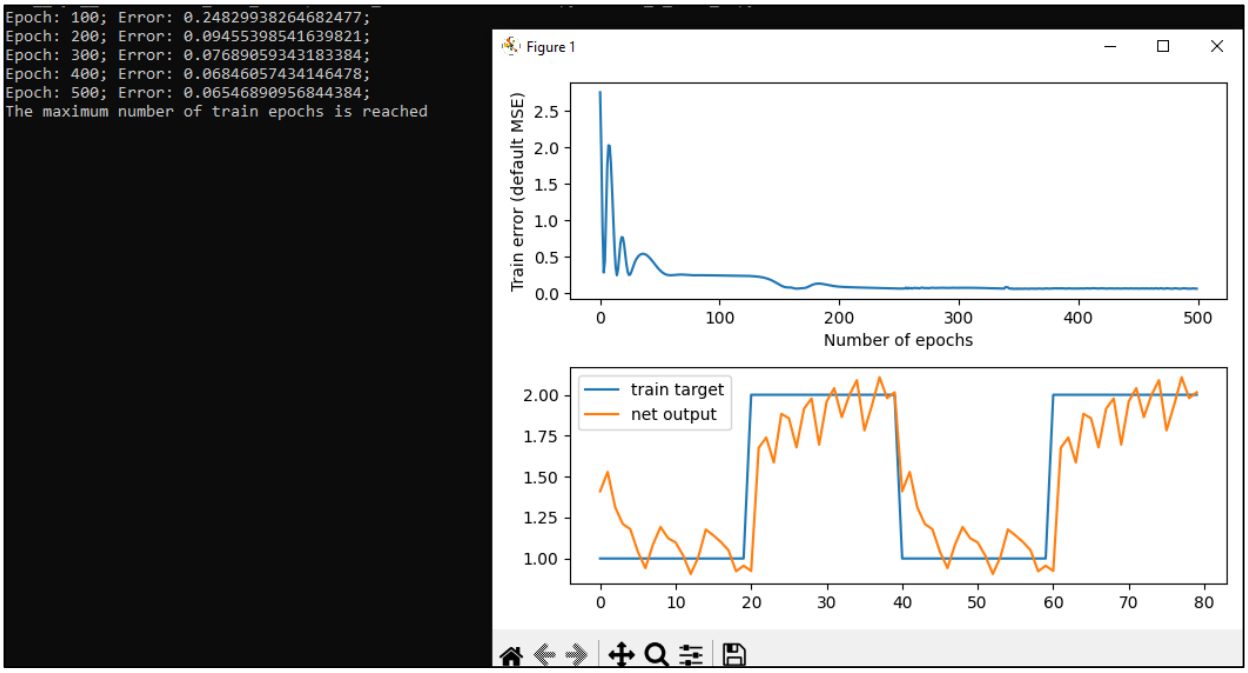

Рис. 2. Результат виконання програми

Висновки до завдання

За допомогою бібілотеки neurolab було створено нейронну мережу високої точності. Нейронна мережа навчилася апроксимувати дані і досягла задовільного рівня помилки тренування.

Завдання 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. 3. Результат виконання програми

Завдання 4. **Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop).**

Лістинг програмного коду для створення та навчання нейронної мережі Хопфілда:

```
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

# Створення та тренування мережі
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())
```

Результат створення та навчання нейронної мережі Хопфілда:

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

Рис. 4.1. Результат створення та навчання нейронної мережі Хопфілда

Протестуємо навчену нейронну мережу Хопфілда. Вважатимемо, що при відображені літери N були помилки

Лістинг програмного коду для тестування:

```
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))
```

Результат виконання програми:

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

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

Рис. 4.2. Результат виконання програми

Лістинг програми для тестування літер E, R, O:

```
# Тестування нейронної мережі
def test_defaced_letter(letter, target, net):
    test = np.asfarray(letter)
    test[test == 0] = -1
    output = net.sim([test])
    result = (output[0] == target).all()
    steps = len(net.layers[0].outs)
    return result, steps

target_letters = [target[0], target[1], target[2], target[3]]

print("\nTest on defaced N:")
result, steps = test_defaced_letter([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], target_letters[0], net)
print(result, 'Sim. steps', steps)

print("\nTest on defaced E:")
result, steps = test_defaced_letter([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], target_letters[1], net)
print(result, 'Sim. steps', steps)

print("\nTest of defaced R:")
result, steps = test_defaced_letter([1, 1, 0, 0, 0,
                                     1, 0, 0, 0, 1,
                                     1, 1, 1, 1, 0,
                                     0, 1, 0, 1, 0,
                                     1, 0, 0, 0, 1], target_letters[2], net)
print(result, 'Sim. steps', steps)

print("\nTest of defaced O:")
result, steps = test_defaced_letter([0, 1, 1, 1, 0,
                                     1, 0, 0, 0, 1,
                                     0, 0, 1, 0, 1,
                                     1, 0, 0, 0, 1,
                                     0, 1, 0, 1, 0], target letters[3], net)
print(result, 'Sim. steps', steps)
```

Результат виконання програми:

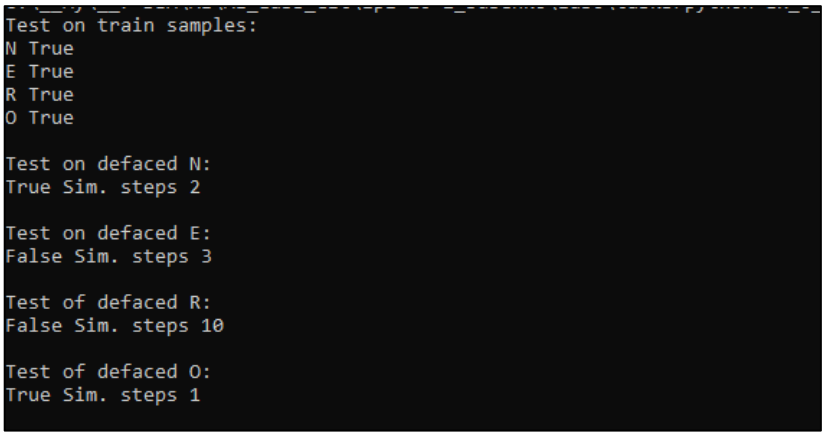

Рис. 4.3. Результат виконання програми

Висновки до завдання

Отже, отримані результати тестування мережі Хопфілда свідчать про її успішність у відновленні тренувальних шаблонів, таких як літери N, E, R та O. Однак, при деформації зразків мережа може виявляти обмежену ефективність, особливо в розпізнаванні деформованих літер E та R.

Завдання 5. Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних.

По аналогії з попереднім завданням візьміть перші букви ваших прізвища, шм'я та по-батькові (кирилицею). Закодуйте їх матрицею пікселів та кодом одиниць і нулів. Навчіть мережу розпізнавати ваші букви. Протестуйте мережу на можливість розпізнавання кожної букви шляхом внесення помилок в тест.

Лістинг програми:

```
      import numpy as np
import neurolab as nl

# Оновлені тренувальні та тестові дані для літер Б, О, В
target = np.array([[1, 1, 1, 1, 1,
                    1, 0, 0, 0, 0,
                    1, 1, 1, 1, 1,  # Б
                    1, 0, 0, 0, 1,
                    1, 1, 1, 1, 1],

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

```
                      1, 0, 1, 0, 0,
                      1, 0, 0, 1, 1,
                      0, 1, 1, 1, 0],  # O

                     [1, 1, 1, 1, 0,
                      1, 0, 0, 0, 1,
                      1, 1, 1, 1, 0,  # B
                      1, 0, 0, 0, 1,
                      1, 1, 1, 1, 0]])

target_defaced = np.array([[1, 1, 1, 1, 1,
                            0, 0, 0, 0, 0,
                            1, 1, 1, 1, 1,  # Б
                            1, 0, 0, 1, 1,
                            1, 1, 1, 1, 1],

                           [0, 1, 0, 1, 0,
                            1, 0, 0, 0, 1,
                            1, 1, 1, 0, 0,
                            1, 0, 0, 1, 1,
                            0, 1, 0, 1, 0],  # O

                           [1, 1, 1, 1, 0,
                            1, 0, 0, 0, 1,
                            1, 1, 1, 1, 0,  # B
                            1, 0, 0, 0, 1,
                            1, 0, 1, 1, 0]])

chars = ['Б', 'O', 'B']
target[target == 0] = -1

# Створення та тренування мережі
net = nl.net.newhop(target.reshape(-1, 25))  # Плоске представлення для мережі
Хопфілда
output = net.sim(target.reshape(-1, 25))
print("Test on train samples:")
for i in range(len(target)):
    print(chars[i], (output[i] == target[i]).all())


# Тестування нейронної мережі для всіх трьох літер
def test_defaced_letter(letter, target, net):
    test = np.asfarray(letter).flatten()
    test[test == 0] = -1
    output = net.sim([test])
    result = (output[0] == target.flatten()).all()
    steps = len(net.layers[0].outs)
    return result, steps


target_letters = [target[0], target[1], target[2]]

# Тестування для літер Б, O, B
for i in range(len(target_letters)):
    print(f"\nTest on defaced {chars[i]}:")
    result, steps = test_defaced_letter(target_letters[i], target_letters[i], net)
    print(result, 'Sim. steps', steps)
```

Результат виконання програми:

Рис. 5. Результат виконання програм

Висновки до завдання

Отже, можна дійти висновку, що як і в попередньому завданні при зміні 1-2 пікселів нейронна мережа Хопфілда правильно ідентифікує літеру.

Висновок: у ході виконання лабораторної роботи я, використовуючи спеціалізовані бібліотеки та мову програмування Python навчився та дослідив деякі типи нейронних мереж.