Практична робота № 6

Варіант 13

Дослідження рекурентних нейронних мереж

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

Хід роботи:

Завдання 6.1: Ознайомлення з Рекурентними нейронними мережами. Прочитайте та виконайте дії згідно рекомендацій до виконання.

Лістинг файлу task-1.py

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

vocab = list(set([word for text in train_data.keys() for word in text.split()]))
vocab_size = len(vocab)

print(f"{vocab_size} unique words in the training data")

word_to_index = {word: i for i, word in enumerate(vocab)}
index_to_word = {i: word for i, word in enumerate(vocab)}
print(word_to_index)
print(index_to_word)


def create_inputs(text):
    inputs = []
    for w in text.split(' '):
        v = np.zeros((vocab_size, 1))
        v[word_to_index[w]] = 1
        inputs.append(v)

    return inputs


def softmax(xs):
    return np.exp(xs) / sum(np.exp(xs))


def process data(data, rnn, backprop=True):
    items = list(data.items())
    np.random.shuffle(items)
```

					ДУ «Житомирська політехніка».22.121.13.000 – Лр06		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Маковська О.Ю			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів 10
Перевір.		Пулеко І. В.					
Керівник						ФІКТ Гр. ІПЗ-19-1[2]	
Н. контр.							
Зав. каф.							

```
    loss = 0
    num_correct = 0

    for x, y in items:
        inputs = create_inputs(x)
        target = int(y)

        out, _ = rnn.forward(inputs)
        probs = softmax(out)

        loss -= float(np.log(probs[target]))
        num_correct += int(np.argmax(probs) == target)

        if backprop:
            d_L_d_y = probs
            d_L_d_y[target] -= 1

            rnn.backprop(d_L_d_y)

    return loss / len(data), num_correct / len(data)


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

        self.bh = np.zeros((hidden size, 1))
        self.by = np.zeros((output_size, 1))

        self.last_inputs = None
        self.last_hs = None

    def forward(self, inputs):
        h = np.zeros((self.Whh.shape[0], 1))
        self.last_inputs = inputs
        self.last_hs = {0: h}

        for i, x in enumerate(inputs):
            h = np.tanh(self.Wxh @ x + self.Whh @ h + self.bh)
            self.last_hs[i + 1] = h

        y = self.Why @ h + self.by
        return y, h

    def backprop(self, d_y, learn_rate=2e-2):
        n = len(self.last_inputs)

        d_Why = d_y @ self.last_hs[n].T
        d_by = d_y

        d_Whh = np.zeros(self.Whh.shape)
        d_Wxh = np.zeros(self.Wxh.shape)
        d_bh = np.zeros(self.bh.shape)

        d_h = self.Why.T @ d_y

        for t in reversed(range(n)):
            temp = ((1 - self.last_hs[t + 1] ** 2) * d_h)

            d_bh += temp
            d_Whh += temp @ self.last_hs[t].T
```

```
            d_Wxh += temp @ self.last_inputs[t].T

            d_h = self.Whh @ temp

        for d in [d_Wxh, d_Whh, d_Why, d_bh, d_by]:
            np.clip(d, -1, 1, out=d)

        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


if __name__ == "__main__":
    rnn = RNN(vocab_size, 2)

    for epoch in range(1000):
        train_loss, train_acc = process_data(train_data, rnn, backprop=True)

        if epoch % 100 == 99:
            print(f"Epoch {epoch + 1}")
            print(f"Train loss: {train_loss:0.3f}, Train acc: {train_acc:0.3f}")

            test_loss, test_acc = process_data(test_data, rnn, backprop=False)
            print(f"Test loss: {test_loss:.3f}, Test acc: {test_acc:.3f}\n")
```

```
C:\Python311\python.exe C:/Users/Ola/Documents/stu
18 unique words in the training data
{'at': 0, 'sad': 1, 'and': 2, 'now': 3, 'good': 4,
{0: 'at', 1: 'sad', 2: 'and', 3: 'now', 4: 'good',
Epoch 100
Train loss: 0.687, Train acc: 0.552
Test loss: 0.699, Test acc: 0.500

Epoch 200
Train loss: 0.670, Train acc: 0.672
Test loss: 0.714, Test acc: 0.650

Epoch 300
Train loss: 0.553, Train acc: 0.707
Test loss: 0.924, Test acc: 0.450

Epoch 400
Train loss: 0.405, Train acc: 0.828
Test loss: 0.708, Test acc: 0.550
```

Рис.6.1. task-1.py

Завдання 6.2: Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm)).

Рекурентні нейронні мережі Елмана часто застосовують для детектування амплітуди сигналу (апроксимації, представлення, наближення сигналів). Тобто нейронна мережа за попередніми відліками сигналу повинна передбачити наступні значення і якомога ближче до реальних.

Лістинг файлу task-2.py

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

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()

error = net.train(input, target, epochs=500, show=100, goal=0.01)
output = net.sim(input)

pl.subplot(211)
pl.plot(error)
pl.xlabel('Number of epochs')
pl.ylabel('Error (default SSE)')

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()
```

```
C:\Python311\python.exe C:/Users/Ola/Documents/stu
Epoch: 100; Error: 0.2482514423968246;
Epoch: 200; Error: 0.1097756592197506;
Epoch: 300; Error: 0.07901500373458648;
Epoch: 400; Error: 0.06732436848730902;
Epoch: 500; Error: 0.03906694842934907;
The maximum number of train epochs is reached
```

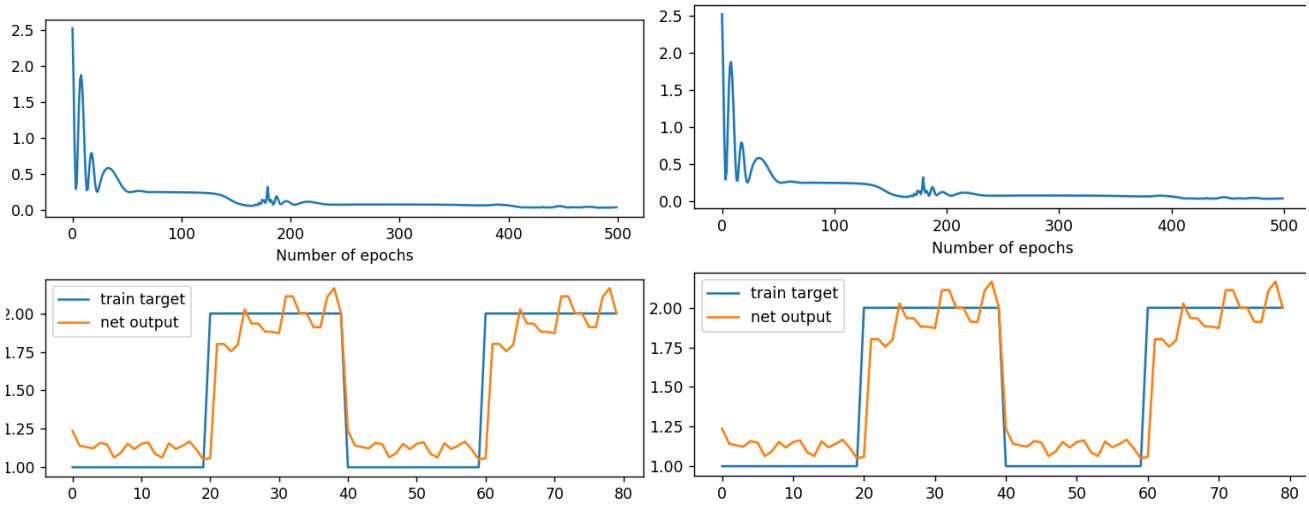

Рис.6.2. task-2.py

Завдання 6.3: Дослідження нейронної мережі Хемінга (Hemming Recurrent network).

Лістинг файлу 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]]

net = nl.net.newhem(target)

output = net.sim(target)
print("Test on train data (must be [0, 1, 2, 3, 4]):")
print(np.argmax(output, axis=0))

output = net.sim([input[0]])
print("Outputs on recurrent cycle:")
print(np.array(net.layers[1].outs))

output = net.sim(input)
print("Test on test sample:")
print(output)
```

```
C:\Python311\python.exe C:/Users/Ola/Documents/stud/4/AI_Py
Test on train data (must be [0, 1, 2, 3, 4]):
[0 1 2 3 4]
Outputs on recurrent 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.     ]]
Test on test sample:
[[0.         0.         0.39168    0.         0.        ]
 [0.         0.         0.         0.         0.39168   ]
 [0.07516193 0.         0.         0.         0.07516193]]
```

Рис.6.3. task-3.py

Завдання 6.4: Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop).

Необхідно навчити рекурентну нейронну мережу Хопфілда розпізнавати букви слова. Формат звернення до відповідної функції для neurolab подано у таблиці.

Лістинг файлу task-4.py

```
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(output)):
    print(chars[i], (output[i] == target[i]).all())

print("Test of 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
output = net.sim([test])
print((output[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))

print("Test of defaced E:")
test = np.asfarray([1, 1, 0, 1, 0,
                    1, 0, 0, 0, 0,
                    1, 1, 0, 0, 0,
                    1, 1, 0, 0, 0,
                    1, 1, 1, 1, 1])
test[test == 0] = -1
output = net.sim([test])
print((output[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))

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

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

```
C:\Python311\python.exe C:/Users/Ola/D
Test on train samples:
N True
E True
R True
O True
Test of defaced N:
True Sim. steps 2
Test of defaced E:
True Sim. steps 3
Test of defaced R:
True Sim. steps 1
Test of defaced O:
True Sim. steps 2
```

Рис.6.4. task-4.py

Завдання 6.5: Дослідження рекурентної нейронної мережі Хопфілда для ваших персональних даних

По аналогії з попереднім завданням візьміть перші букви Ваших Прізвища, Імя 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. Код тесту та результати тестування занесіть у звіт.

Вхідні дані: літери M, O, J.

Фрагмент лістингу файлу task-5.py

```
import numpy as np
import neurolab as nl

target = [[1, 0, 0, 0, 1,
           1, 1, 0, 1, 1,
           1, 1, 0, 1, 1,
           1, 0, 1, 0, 1,
           1, 0, 1, 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],
```

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

chars = ['M', 'O', 'J']
target = np.asfarray(target)
target[target == 0] = -1
```

```
C:\Python311\python.exe C:/Users/Ola/
Test on train samples:
M True
O True
J True
Test of defaced O:
False Sim. steps 1
Test of defaced M:
False Sim. steps 1
Test of defaced B:
False Sim. steps 1
```

Рис.6.5. task-5.py

https://github.com/avrorilka/AI_Python

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