ЛАБОРАТОРНА РОБОТА № 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи

Завдання 1. Ознайомлення з Рекурентними нейронними мережами

Лістинг коду 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)

    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")
```

```
18 unique words in the training data
{'all': 0, 'now': 1, 'was': 2, 'am': 3, 'happy': 4, 'sad': 5, 'earlier': 6,
{0: 'all', 1: 'now', 2: 'was', 3: 'am', 4: 'happy', 5: 'sad', 6: 'earlier',
Epoch 100
Train loss: 0.688, Train acc: 0.552
Test loss: 0.696, Test acc: 0.500

Epoch 200
Train loss: 0.669, Train acc: 0.621
Test loss: 0.723, Test acc: 0.650

Epoch 300
Train loss: 0.562, Train acc: 0.672
Test loss: 0.750, Test acc: 0.500

Epoch 400
Train loss: 0.451, Train acc: 0.776
Test loss: 0.515, Test acc: 0.800

Epoch 500
Train loss: 0.210, Train acc: 0.897
Test loss: 0.537, Test acc: 0.750

Epoch 600
```

Рис.6.1 – Робота самостійно створеної рекурентної нейронної мережі

Створена нейронна мережа повністю відповідає вимогам поставленої задачі та ефективно навчається зі збільшенням якості та зменшенням втрат.

Завдання 2. Дослідження рекурентної нейронної мережі Елмана

Лістинг коду 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()
```

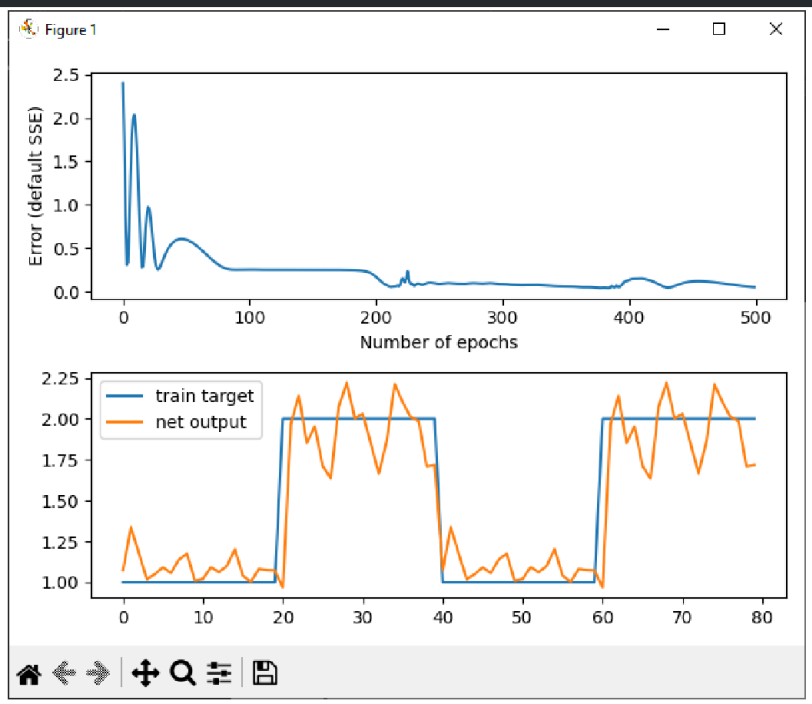

Рис.6.2 – Графіки помилок вихідних даних та співпадіння з тренувальними

```
Epoch: 100; Error: 0.2537632506254803;
Epoch: 200; Error: 0.18023055569162597;
Epoch: 300; Error: 0.08680546970711644;
Epoch: 400; Error: 0.12715194449936496;
Epoch: 500; Error: 0.05126693846960685;
The maximum number of train epochs is reached
```

Рис.6.3 – Звітність по навчанню

Використання бібліотеки neurolab дозволяє спростити та пришвидшити розробку нейронних мереж, якість та точність яких є високою.

Завдання 3. Дослідження нейронної мережі Хемінга

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)
```

```
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.6.4 – Результат роботи програми

Завдання 4. Дослідження рекурентної нейронної мережі Хопфілда

Лістинг коду 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))
```

```
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.6.5 – Навчання програми та перевірка на зіпсованих даних

Використання нейронної мережі Хопфілда є ефективним для розрізнення літер на основі навчальних наборів.

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

Вхідні дані: літери О, М, В.

Лістинг коду файлу Task_5.py:

```
import numpy as np
import neurolab as nl

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

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

chars = ['O', 'M', 'B']
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 O:")
test = np.asfarray([0, 1, 1, 1, 0,
                    0, 0, 0, 0, 1,
                    0, 0, 0, 0, 1,
                    1, 0, 0, 0, 1,
                    0, 1, 1, 1, 0])
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 M:")
test = np.asfarray([1, 0, 0, 0, 0,
                    1, 1, 0, 1, 1,
                    1, 0, 0, 0, 1,
                    1, 0, 0, 0, 1,
                    1, 0, 1, 0, 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 B:")
test = np.asfarray([0, 1, 1, 1, 1,
```

```
                    1, 0, 0, 0, 1,
                    1, 1, 1, 0, 0,
                    1, 0, 0, 0, 1,
                    0, 1, 1, 1, 1])
test[test == 0] = -1
output = net.sim([test])
print((output[0] == target[2]).all(), 'Sim. steps', len(net.layers[0].outs))
```

```
Test on train samples:
O True
M True
B True
Test of defaced O:
True Sim. steps 1
Test of defaced M:
True Sim. steps 1
Test of defaced B:
True Sim. steps 3
```

Рис.6.6 – Навчання та використання нейронної мережі для розпізнавання ініціалів імені (O, M, B)

Висновок: протягом виконання завдань лабораторної роботи було отримано навички дослідження певних типів нейронних мереж використовуючи спеціалізовані бібліотеки та мову програмування Python.

Для виконання завдань було розроблено та використано власні функції та клас рекурентної нейронної мережі, використано нейронні мережі з функціоналу бібліотеки neurolab.

Код завдань зберігається у репозиторії за посиланням: https://github.com/nikitoss888/AI_LR6.