ЛАБОРАТОРНА РОБОТА № 6

ОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи:

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

```
import random


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))

    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


from data import train_data, test_data

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)
word_to_idx = {w: i for i, w in enumerate(vocab)}
idx_to_word = {i: w for i, w in enumerate(vocab)}


def createInputs(text):
    '''
  Returns an array of one-hot vectors representing the words in the input text string.
  - text is a string
  - Each one-hot vector has shape (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):
    return np.exp(xs) / sum(np.exp(xs))


# Initialize our RNN!
rnn = RNN(vocab_size, 2)


def processData(data, backprop=True):
    items = list(data.items())
    random.shuffle(items)

    loss = 0
    num_correct = 0
    for x, y in items:
        inputs = createInputs(x)
        target = int(y)

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

        loss -= 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)


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))

import numpy as np
from numpy.random import randn


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))

    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
```

```
--- Epoch 100                          --- Epoch 600
Train:  Loss 0.688 | Accuracy: 0.552   Train:  Loss 0.366 | Accuracy: 0.776
Test:   Loss 0.698 | Accuracy: 0.500   Test:   Loss 0.500 | Accuracy: 0.800
--- Epoch 200                          --- Epoch 700
Train:  Loss 0.671 | Accuracy: 0.603   Train:  Loss 0.012 | Accuracy: 1.000
Test:   Loss 0.721 | Accuracy: 0.400   Test:   Loss 0.027 | Accuracy: 1.000
--- Epoch 300                          --- Epoch 800
Train:  Loss 0.587 | Accuracy: 0.638   Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.644 | Accuracy: 0.650   Test:   Loss 0.020 | Accuracy: 1.000
--- Epoch 400                          --- Epoch 900
Train:  Loss 0.412 | Accuracy: 0.793   Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.672 | Accuracy: 0.750   Test:   Loss 0.038 | Accuracy: 1.000
--- Epoch 500                          --- Epoch 1000
Train:  Loss 0.221 | Accuracy: 0.931   Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.408 | Accuracy: 0.850   Test:   Loss 0.054 | Accuracy: 0.950
```

Рис.1. Результат виконання LR_6_task_1.py

Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана

```
import neurolab as nl
import numpy as np

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)
import pylab as pl

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()
```

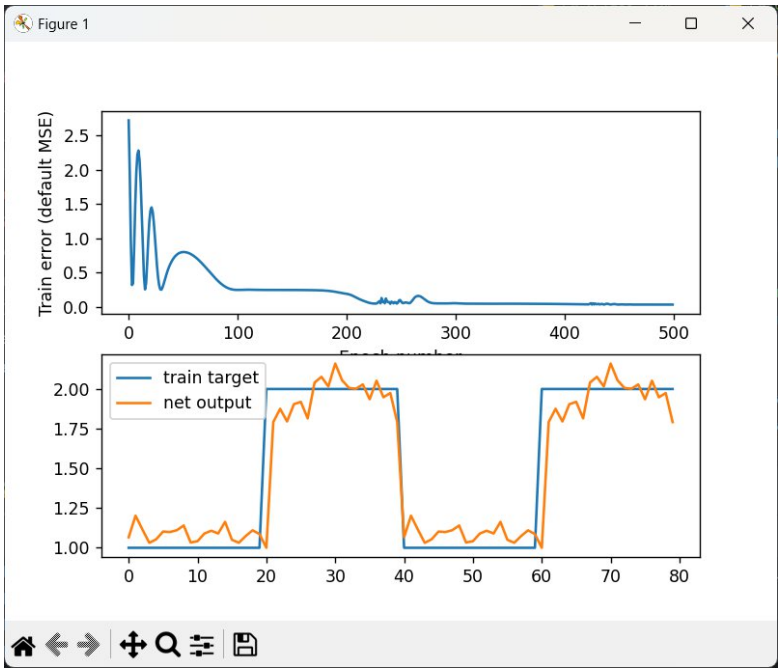

Рис.2. Результат виконання LR_6_task_2.py

```
Epoch: 100; Error: 0.2501490416191357;
Epoch: 200; Error: 0.1957298075484657;
Epoch: 300; Error: 0.056365530983169286;
Epoch: 400; Error: 0.044224181983255555;
Epoch: 500; Error: 0.03604369673326983;
The maximum number of train epochs is reached
```

Рис.3. Результат виконання LR_6_task_2.py в консолі

Завдання 2.3. Дослідження нейронної мережі Хемінга

```
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)
```

```
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.4. Результат виконання LR_6_task_3.py

Завдання 2.4. Дослідження рекурентної 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
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 M:")
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[1]).all(), 'Sim. steps', len(net.layers[0].outs))
```

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

Рис.5. Результат виконання LR_6_task_4.py

```
Test on defaced M:
False Sim. steps 2
```

Рис.6. Результат виконання LR_6_task_4.py

```
test = np.asfarray(
    [0, 0, 1, 0, 1,
     1, 1, 0, 0, 0,
     1, 1, 0, 1, 1,
     1, 0, 1, 1, 1,
     0, 1, 0, 1, 1],
)
```

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

Рис.7. Результат виконання LR_6_task_4.py

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

```
import numpy as np
import neurolab as nl

target = [[1, 0, 0, 0, 1,
           1, 0, 1, 1, 0,
           1, 1, 0, 0, 0,
           1, 0, 1, 1, 0,
           1, 0, 0, 0, 1],
          [1, 0, 0, 0, 1,
           1, 0, 1, 1, 0,
           1, 1, 0, 0, 0,
           1, 0, 1, 1, 0,
           1, 0, 0, 0, 1],
          [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]
          ]
chars = ['K', 'K', 'M']
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(target)):
    print(chars[i], (output[i] == target[i]).all())

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

```
Test on train samples:
K True
K True
M True

Test on defaced M:
False Sim. steps 1
```

Рис.8. Результат виконання LR_6_task_5.py

Репозиторій: https://github.com/Kochubei-Kostiantyn/AI_labs

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