ЗВІТ

З лабораторної роботи №6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

КН-20-1 навчальної групи

Кірія Даніли Олеговича варіант №6

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

Завдання 1.

```
"D:\LABS\PRESENT\AI (Python)\Artificia
18 unique words found
10
is
```

```
--- Epoch 100
Train:  Loss 0.689 | Accuracy: 0.552
Test:   Loss 0.699 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.665 | Accuracy: 0.638
Test:   Loss 0.733 | Accuracy: 0.650
--- Epoch 300
Train:  Loss 0.592 | Accuracy: 0.621
Test:   Loss 0.678 | Accuracy: 0.550
--- Epoch 400
Train:  Loss 0.440 | Accuracy: 0.793
Test:   Loss 0.509 | Accuracy: 0.750
--- Epoch 500
Train:  Loss 0.238 | Accuracy: 0.897
Test:   Loss 0.483 | Accuracy: 0.700
```

```
--- Epoch 600
Train:  Loss 0.022 | Accuracy: 1.000
Test:   Loss 0.072 | Accuracy: 0.950
--- Epoch 700
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.005 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.003 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 1000
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
```

Код програми:

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

warnings.filterwarnings("ignore")


def createInputs(text):
```

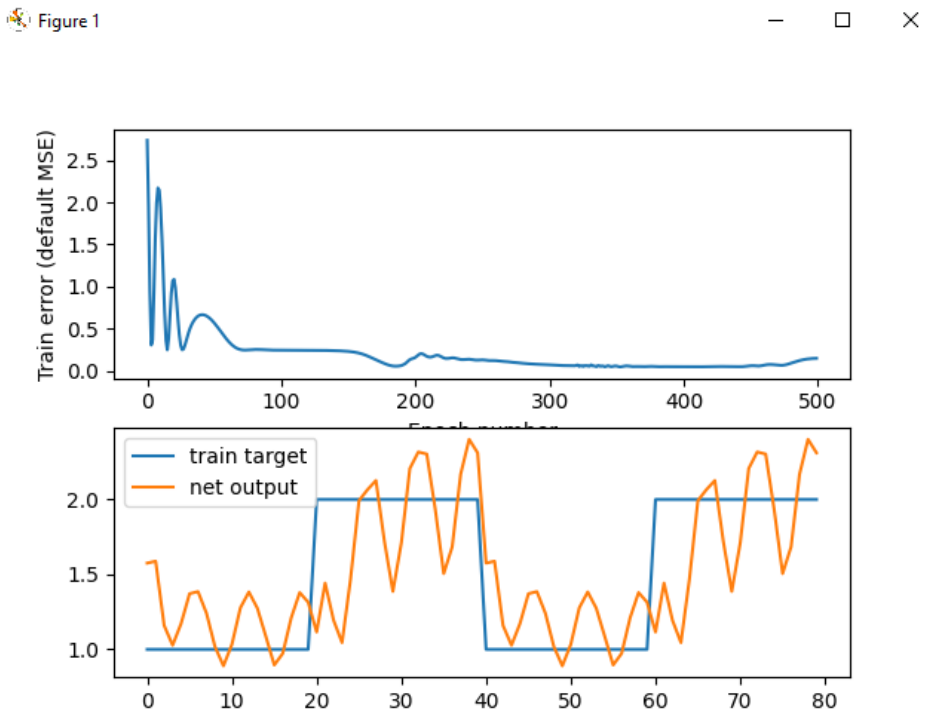
```
    """
    Повертає масив унітарних векторів
    які представляють слова у введеному рядку тексту
    - текст є рядком string
    - Унітарний вектор має форму (vocab_size, 1)
    """

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


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)D

        # 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)


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

```
        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 фази прямого поширення нейронної мережі з
        використанням введених даних.
        Повернення підсумкової видачі та прихованого стану.
        - Вхідні дані у масиві однозначного 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


# 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

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)

# 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, train_acc))

        test_loss, test_acc = processData(test_data, backprop=False)
        print('Test:\tLoss %.3f | Accuracy: %.3f' % (test_loss, test_acc))
```

Завдання 2:

```
"D:\LABS\PRESENT\AI (Python)\Artificial-intell
Epoch: 100; Error: 0.25143268832481314;
Epoch: 200; Error: 0.06550809804624877;
Epoch: 300; Error: 0.0446863158902996;
Epoch: 400; Error: 0.0352140014508809;
Epoch: 500; Error: 0.029275690392878674;
The maximum number of train epochs is reached
```




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

warnings.filterwarnings("ignore")

# Створення моелей 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()
```

Завдання 3:

```
"D:\LABS\PRESENT\AI (Python)\Artificial-intelligence\LAB4\
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]]
```

Код 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)
```

Завдання 4:

```
"D:\LABS\PRESENT\AI (Python)\Artificial-intelligence
Test on train samples:
N True
E True
R True
O True

Test on defaced O:
True Sim. steps 1
```

```
"D:\LABS\PRESENT\AI (Python)\
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
False Sim. steps 3
```

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

Завдання 5:

```
"D:\LABS\PRESENT\AI (Python)\Artific
Test on train samples:
К True
Д True
О True

Test on defaced K:
True Sim. steps 1
```

Код програми:

```
import numpy as np
import neurolab as nl

# К Д О
target = [[1, 0, 0, 0, 1,
           1, 0, 0, 1, 0,
           1, 1, 1, 0, 0,
           1, 0, 0, 1, 0,
           1, 0, 0, 0, 1],
          [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, 1,
           1, 0, 0, 0, 1,
           0, 1, 1, 1, 0]]

chars = ['К', 'Д', 'О']
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 K:")
test = np.asfarray([1, 0, 0, 0, 1,
                    1, 0, 0, 1, 0,
                    1, 0, 1, 0, 0,
                    1, 0, 0, 1, 0,
                    1, 0, 0, 0, 1])
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[0]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Висновки: під час виконання лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python, було отримано практичні навички з дослідження деяких типів нейронних мереж.

GitHub: https://github.com/invicibleee/Artificial-intelligence.git