ЛАБОРАТОРНА РОБОТА № 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися дослідити деякі типи нейронних мереж.
GitHub репозиторій: **https://github.com/SMTH666/OAI-Lab**

2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

Лістинг програми:

```
import random
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)

        # Calculate dL/dWhy and dL/dby.
```

```
        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

        # Backpropagate through time.
        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):
    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))


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)


# Training loop
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.item(),
train_acc))

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

Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab06\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab06\LR_6_task_1.py
18 unique words found
--- Epoch 100
Train:  Loss 0.689 | Accuracy: 0.552
Test:   Loss 0.697 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.667 | Accuracy: 0.621
Test:   Loss 0.717 | Accuracy: 0.500
--- Epoch 300
Train:  Loss 0.569 | Accuracy: 0.655
Test:   Loss 0.673 | Accuracy: 0.550
--- Epoch 400
Train:  Loss 0.392 | Accuracy: 0.862
Test:   Loss 0.625 | Accuracy: 0.600
--- Epoch 500
Train:  Loss 0.319 | Accuracy: 0.879
Test:   Loss 0.796 | Accuracy: 0.600
--- Epoch 600
Train:  Loss 0.146 | Accuracy: 0.948
Test:   Loss 1.215 | Accuracy: 0.650
--- Epoch 700
Train:  Loss 0.035 | Accuracy: 1.000
Test:   Loss 0.574 | Accuracy: 0.900
--- Epoch 800
Train:  Loss 0.005 | Accuracy: 1.000
Test:   Loss 0.854 | Accuracy: 0.850
--- Epoch 900
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.961 | Accuracy: 0.850
--- Epoch 1000
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 1.017 | Accuracy: 0.850

Process finished with exit code 0
```

Висновок: Було створено просту рекурентну нейронну мережу. Під час тренування модель зменшує втрату та збільшує точність на навчальних даних з плином часу.

Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))

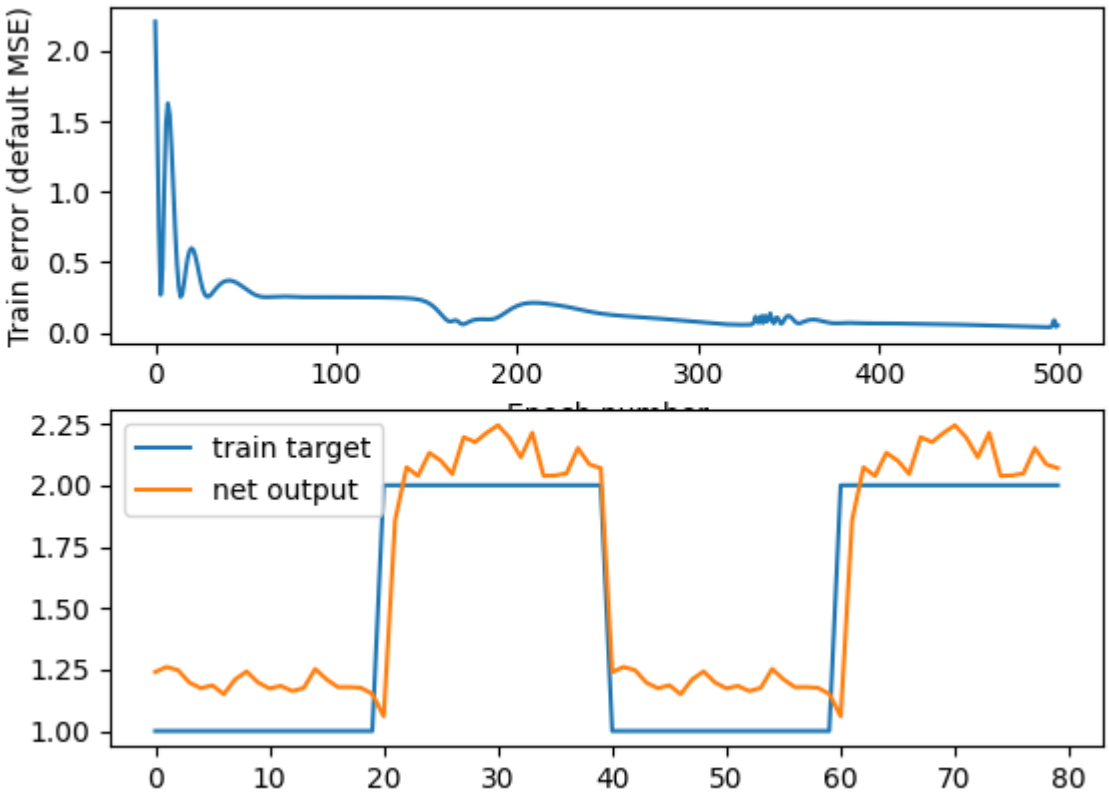
Лістинг програми:

```python
import neurolab as nl
import numpy as np
import matplotlib.pyplot as plt

# 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_data = np.array([i1, i2, i1, i2]).reshape(20 * 4, 1)
target_data = 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 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_data, target_data, epochs=500, show=100, goal=0.01)

# Запустіть мережу
output_data = net.sim(input_data)

# Побудова графіків
plt.subplot(211)
plt.plot(error)
plt.xlabel('Epoch number')
plt.ylabel('Train error (default MSE)')

plt.subplot(212)
plt.plot(target_data.reshape(80))
plt.plot(output_data.reshape(80))
plt.legend(['train target', 'net output'])
plt.show()
```

Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab06\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab06\LR_6_task_2.py
Epoch: 100; Error: 0.250730375888175;
Epoch: 200; Error: 0.18257766949374266;
Epoch: 300; Error: 0.07613983663012003;
Epoch: 400; Error: 0.06467893716737802;
Epoch: 500; Error: 0.05121285800899521;
The maximum number of train epochs is reached

Process finished with exit code 0
```



Завдання 2.3. Дослідження нейронної мережі Хемінга (Hemming Recurrent network)

Лістинг програми:

```
import numpy as np
import neurolab as nl

target = np.array([[-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_data = np.array([[-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)

# Тестування на тренувальних зразках
output_train = net.sim(target)
print("Test on train samples (must be [0, 1, 2, 3, 4]):")
print(np.argmax(output_train, axis=0))

# Тестування на першому тестовому зразку
output_recurrent = net.sim([input_data[0]])
print("Outputs on recurrent cycle:")
```

```
print(np.array(net.layers[1].outs))

# Тестування на всіх тестових зразках
output_test = net.sim(input_data)
print("Outputs on test samples:")
print(output_test)
```

Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab06\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab06\LR_6_task_3.py
Test on train samples (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.     ]]
Outputs on test samples:
[[0.         0.         0.39168    0.         0.        ]
 [0.         0.         0.         0.         0.39168   ]
 [0.07516193 0.         0.         0.         0.07516193]]

Process finished with exit code 0
```

Завдання 2.4. Дослідження рекурентної нейронної мережі Хопфілда Hopfield Recurrent network (newhop)

Лістинг 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())
```

Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab06\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab06\LR_6_task_4.py
Test on train samples:
N True
E True
R True
O True

Process finished with exit code 0
```

Висновок: У коді використовується бібліотека neurolab для створення та тренування простої рекурентної нейронної мережі.Метою є розпізнавання букв (N, E, R, O) на основі заданих шаблонів. Код успішно тренує мережу на навчальних даних та протестовує її на них. Всі чотири букви (N, E, R, O) розпізнаються правильно.

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

target = [[0, 1, 1, 1, 0,
           0, 1, 0, 1, 0,
           0, 1, 1, 1, 0,   # P
           0, 1, 0, 0, 0,
           0, 1, 0, 0, 0],
          [0, 0, 1, 0, 0,
           0, 1, 0, 1, 0,
           1, 1, 1, 1, 1,   # A
           1, 0, 0, 0, 1,
           1, 0, 0, 0, 1],
          [0, 0, 0, 0, 0,
           0, 1, 1, 1, 0,
           0, 0, 1, 0, 0,   # I
           0, 0, 1, 0, 0,
           0, 1, 1, 1, 0]
          ]
chars = ['Y', 'A', 'Y']
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 A:")
test = np.asfarray(
    [0, 0, 1, 0, 0,
     0, 1, 0, 1, 0,
     1, 1, 1, 1, 1,   # A
     1, 0, 0, 0, 1,
     1, 0, 0, 0, 1],
)
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[1]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Результат виконання:

```
C:\Users\toxa1\PycharmProjects\Lab06\venv\Scripts\python.exe C:\Users\toxa1\PycharmProjects\Lab06\LR_6_task_5.py
Test on train samples:
Y True
A True
Y True

Test on defaced A:
True Sim. steps 1

Process finished with exit code 0
```