ЛАБОРАТОРНА РОБОТА № 6

ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи:

Посилання на репозиторій: https://github.com/AlexanderSydorchuk/AI-Lab6

Завдання 2.1. Ознайомлення з Рекурентними нейронними мережами

Лістинг програми:

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

# створення словника
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")

# призначаємо індекс кожному слову
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 accuracy:
{train_acc:0.3f}")

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

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab6\venv\Scripts\py
18 unique words in the training data
{'bad': 0, 'happy': 1, 'am': 2, 'was': 3, 'i': 4, 'is':
{0: 'bad', 1: 'happy', 2: 'am', 3: 'was', 4: 'i', 5: 'i
Epoch 100
Train loss: 0.688, Train accuracy: 0.552
Test loss: 0.697, Test accuracy: 0.500

Epoch 200
Train loss: 0.662, Train accuracy: 0.638
Test loss: 0.725, Test accuracy: 0.450

Epoch 300
Train loss: 0.146, Train accuracy: 1.000
Test loss: 0.101, Test accuracy: 1.000

Epoch 400
Train loss: 0.013, Train accuracy: 1.000
Test loss: 0.061, Test accuracy: 0.950

Epoch 500
Train loss: 0.005, Train accuracy: 1.000
Test loss: 0.006, Test accuracy: 1.000

Epoch 600
Train loss: 0.003, Train accuracy: 1.000
Test loss: 0.003, Test accuracy: 1.000
```

Отримана найпростіша рекурентна нейронна мережа, яка здатна ефективно навчатися.

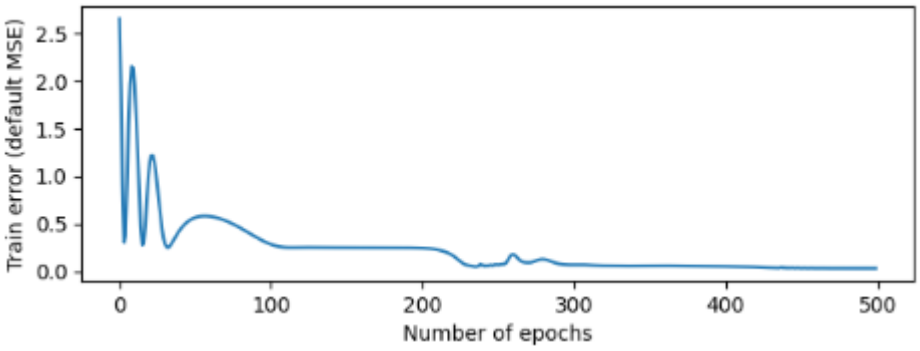
Завдання 2.2. Дослідження рекурентної нейронної мережі Елмана (Elman Recurrent network (newelm))

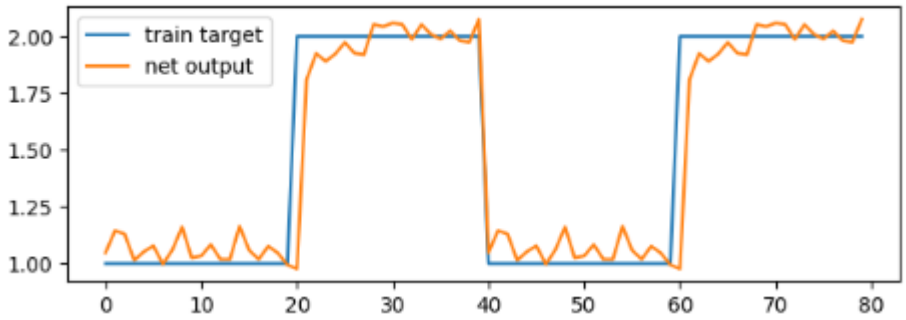
Лістинг програми:

```
import numpy as np
import neurolab as nl
import matplotlib.pyplot as pl

# Створення моделей 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()])

# Ініціалізація 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)
# Запуск мережі
output = net.sim(input)

# Побудова графіків
pl.subplot(211)
pl.plot(error)
pl.xlabel('Number of epochs')
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.tight_layout(w_pad=1.5)
pl.show()
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab6\venv\Scrip
Epoch: 100; Error: 0.28940056522772206;
Epoch: 200; Error: 0.24545265401358876;
Epoch: 300; Error: 0.06818902789008431;
Epoch: 400; Error: 0.05224284683990612;
Epoch: 500; Error: 0.031703053911605675;
The maximum number of train epochs is reached
```





Як ми бачимо, за відносно невеликої кількості коду завдяки підключеним готовим бібліотекам вдалося створити модель нейронної мережі, що виконує поставлене завдання з високою точністю.

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

Лістинг 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
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)
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\Lab6\venv\Scripts\python.exe
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]]

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

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, 1, 1, 1,
                    0, 0, 0, 0, 0,
                    1, 1, 0, 1, 1,
                    1, 0, 0, 0, 0,
                    1, 0, 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([1, 1, 0, 1, 0,
                    1, 0, 0, 0, 1,
                    1, 1, 1, 1, 0,
                    0, 0, 0, 1, 0,
                    1, 0, 0, 0, 1])
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, 1, 1, 1, 0,
                    1, 0, 0, 0, 1,
                    0, 0, 1, 0, 1,
                    1, 0, 0, 0, 1,
                    0, 1, 0, 1, 0])
test[test == 0] = -1
output = net.sim([test])
print((output[0] == target[3]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtze\AI\
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 1

Process finished with exit code 0
```

Подана нейромережа показала гарний результат при роботі з матрицями бінарних даних і може вгадувати літери навіть із помилково введеними елементами матриці.

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

# C O C
target = [[0, 1, 1, 1, 0,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 0,
           1, 0, 0, 0, 0,
           0, 1, 1, 1, 0],

          [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],

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

chars = ['C', 'O', 'C']
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(output)):
    print(chars[i], (output[i] == target[i]).all())

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

Результат виконання програми:

```
"D:\DFiles\LP\4 Karzen\1 Rihtz
Test on train samples:
C True
O True
C True
Test of defaced C:
True Sim. steps 1
Test of defaced O:
True Sim. steps 1
Test of defaced C:
True Sim. steps 1
```

В результаті вдалося навчити нейромережу розпізнавати власні ініціали з невеликою кількістю помилок.

Висновки: в ході лабораторної роботи, використовуючи спеціалізовані бібліотеки та мову програмування Python було опановано деякі типи нейронних мереж.