ЛАБОРАТОРНА РОБОТА № 6

ДОСЛІДЖЕННЯ 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

Лістинг програми:

```
import numpy as np
import random

from rnn import RNN
from data import train_data, test_data

# 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)

# Призначити індекс кожному слову
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'])
print(idx_to_word[0])

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):
  # Застосування функції Softmax для вхідного масиву
  return np.exp(xs) / sum(np.exp(xs))
```

```
# Ініціалізація нашої рекурентної нейронної мережі RNN
rnn = RNN(vocab_size, 2)

inputs = createInputs('i am very good')
out, h = rnn.forward(inputs)
probs = softmax(out)
print(probs) # [[0.50000095], [0.49999905]]

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)

    # 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)

# 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))
```

Результат виконання завдання:

```
16
right
[[0.50000723]
 [0.49999277]]
--- Epoch 100
Train:  Loss 0.690 | Accuracy: 0.552
Test:   Loss 0.696 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.674 | Accuracy: 0.638
Test:   Loss 0.725 | Accuracy: 0.500
--- Epoch 300
Train:  Loss 0.640 | Accuracy: 0.672
Test:   Loss 0.730 | Accuracy: 0.500
--- Epoch 400
Train:  Loss 0.425 | Accuracy: 0.810
Test:   Loss 0.570 | Accuracy: 0.700
--- Epoch 500
Train:  Loss 0.271 | Accuracy: 0.931
Test:   Loss 0.674 | Accuracy: 0.550
--- Epoch 600
Train:  Loss 0.154 | Accuracy: 0.931
Test:   Loss 0.823 | Accuracy: 0.700
--- Epoch 700
Train:  Loss 0.012 | Accuracy: 1.000
Test:   Loss 0.293 | Accuracy: 0.850
--- Epoch 800
Train:  Loss 0.005 | Accuracy: 1.000
Test:   Loss 0.235 | Accuracy: 0.900
--- Epoch 900
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.233 | Accuracy: 0.950
--- Epoch 1000
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.233 | Accuracy: 0.950
```

Рис. 6.1 Результат виконання завдання

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

Лістинг програми:

```
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 = 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
plt.subplot(211)
plt.plot(error)
plt.xlabel('Epoch number')
plt.ylabel('Train error (default MSE)')

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

Результат виконання завдання:

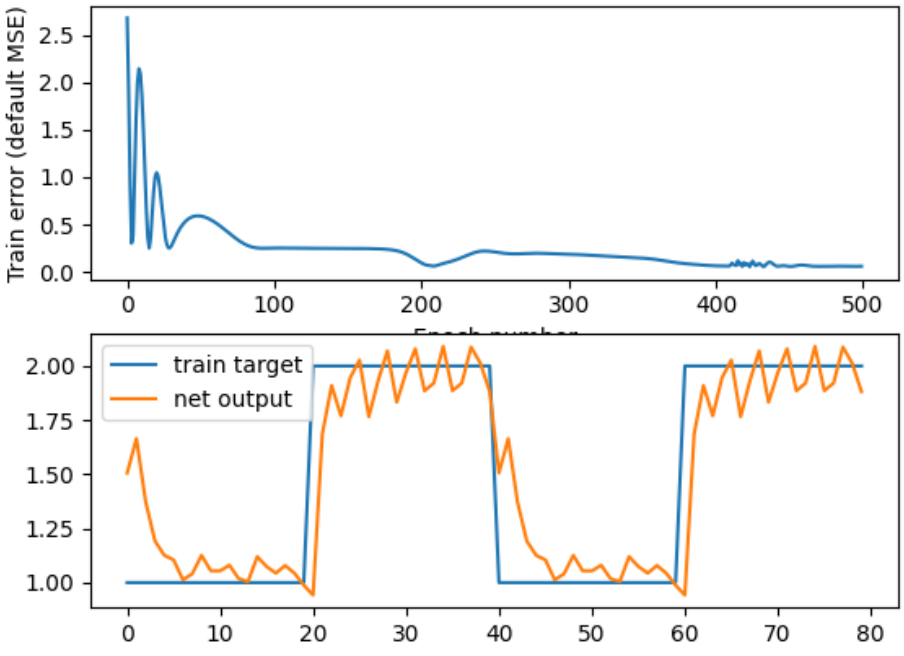

Рис. 6.2 Результат виконання завдання

```
Epoch: 100; Error: 0.25225388239729485;
Epoch: 200; Error: 0.10392224077372329;
Epoch: 300; Error: 0.18637543388811334;
Epoch: 400; Error: 0.0650986921196602;
Epoch: 500; Error: 0.060170289958326226;
The maximum number of train epochs is reached
```

Рис. 6.3 Результат виконання завдання

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 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)
```

Результат виконання завдання:

```
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. 6.4 Результат виконання завдання

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

Лістинг програми:

```
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

# 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, 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[0]).all(), 'Sim. steps', len(net.layers[0].outs))
```

Результат виконання завдання:

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

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

Рис. 6.5 Результат виконання завдання

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

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

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

Результат виконання завдання:

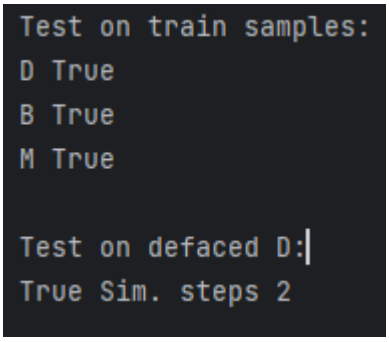

Рис. 6.6 Результат виконання завдання

Посилання на GitHub: https://github.com/BogdanStelmah/Basics-of-AI_labs

Висновок: На даній лабораторній роботі ми використовуючи спеціалізовані бібліотеки та мову програмування Python навчилися дослідити деякі типи нейронних мереж.