Лабораторна робота №6

Тема: ДОСЛІДЖЕННЯ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

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

Хід роботи

GitHub репозиторій:
https://github.com/BashmanivskiyMaxim/Artificial_intelligence_labs

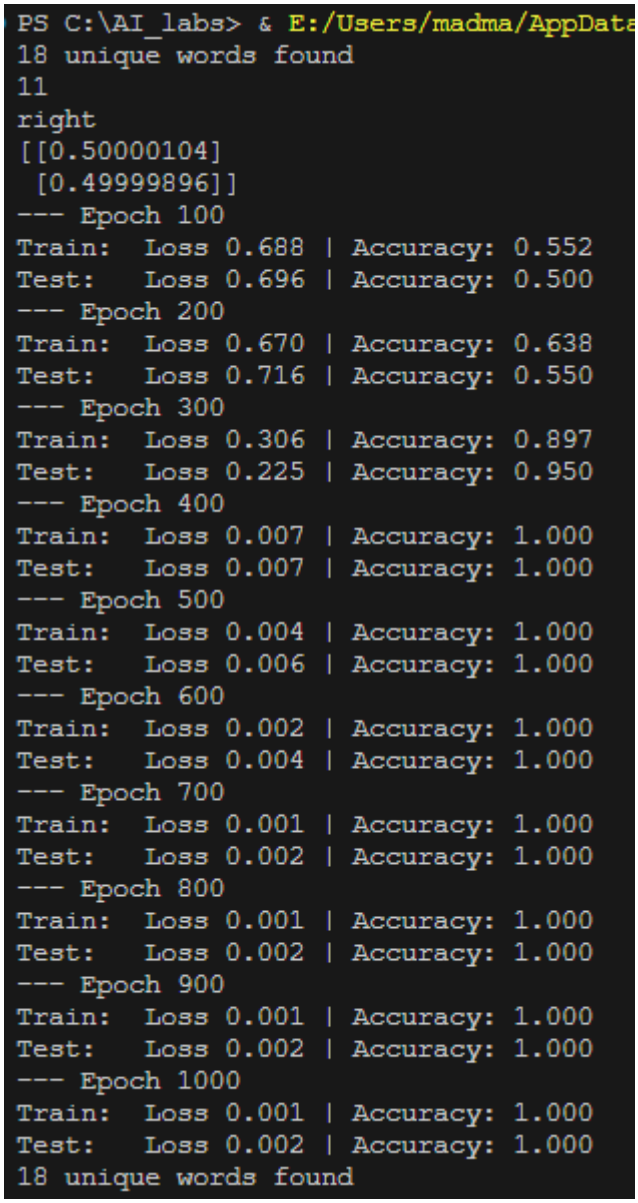
Завдання 2.1: Ознайомлення з Рекурентними нейронними мережами

Лістинг програми main.py:

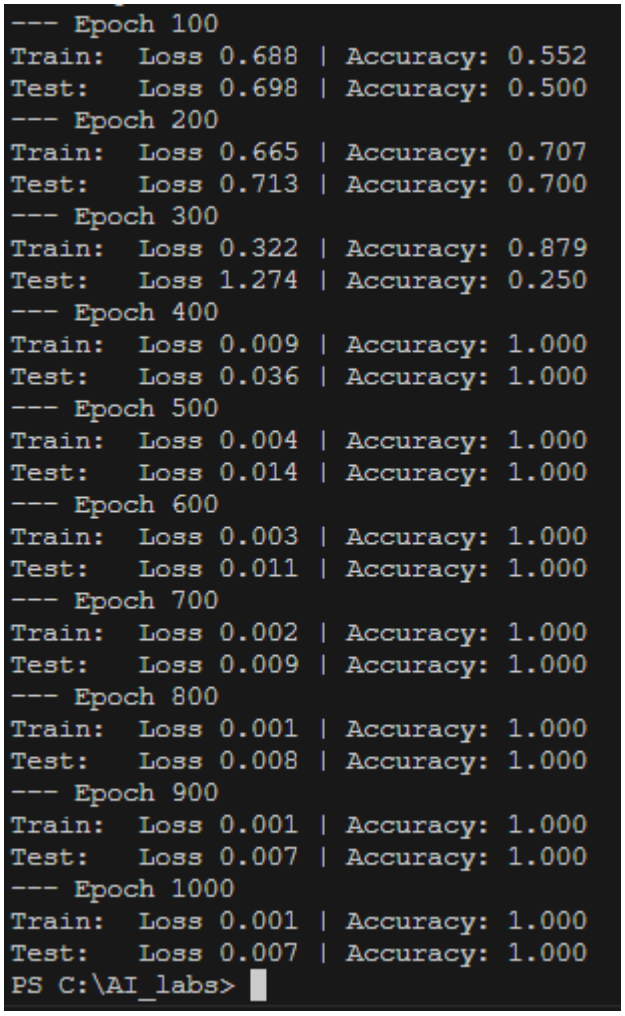
```
import numpy as np
import random

from LR_6_task_1 import RNN
from data import train_data, test_data

# Create the vocabulary.
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)

# Assign indices to each word.
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):
  '''
  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):
  # Applies the Softmax Function to the input array.
  return np.exp(xs) / sum(np.exp(xs))
```

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

def processData(data, backprop=True):
  '''
  Returns the RNN's loss and accuracy for the given data.
  - data is a dictionary mapping text to True or False.
  - backprop determines if the backward phase should be run.
  '''
  items = list(data.items())
  random.shuffle(items)

  loss = 0
  num_correct = 0

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

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

    # Calculate loss / accuracy
    loss -= np.log(probs[target])
    num_correct += int(np.argmax(probs) == target)

    if backprop:
      # Build dL/dy
      d_L_d_y = probs
      d_L_d_y[target] -= 1

      # Backward
      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, train_acc))

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

Результат виведення main.py:



```
PS C:\AI_labs> & E:/Users/madma/AppData
18 unique words found
11
right
[[0.50000104]
 [0.49999896]]
--- Epoch 100
Train:  Loss 0.688 | Accuracy: 0.552
Test:   Loss 0.696 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.670 | Accuracy: 0.638
Test:   Loss 0.716 | Accuracy: 0.550
--- Epoch 300
Train:  Loss 0.306 | Accuracy: 0.897
Test:   Loss 0.225 | Accuracy: 0.950
--- Epoch 400
Train:  Loss 0.007 | Accuracy: 1.000
Test:   Loss 0.007 | Accuracy: 1.000
--- Epoch 500
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.006 | Accuracy: 1.000
--- Epoch 600
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.004 | Accuracy: 1.000
--- Epoch 700
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.002 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.001 | 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
18 unique words found
```

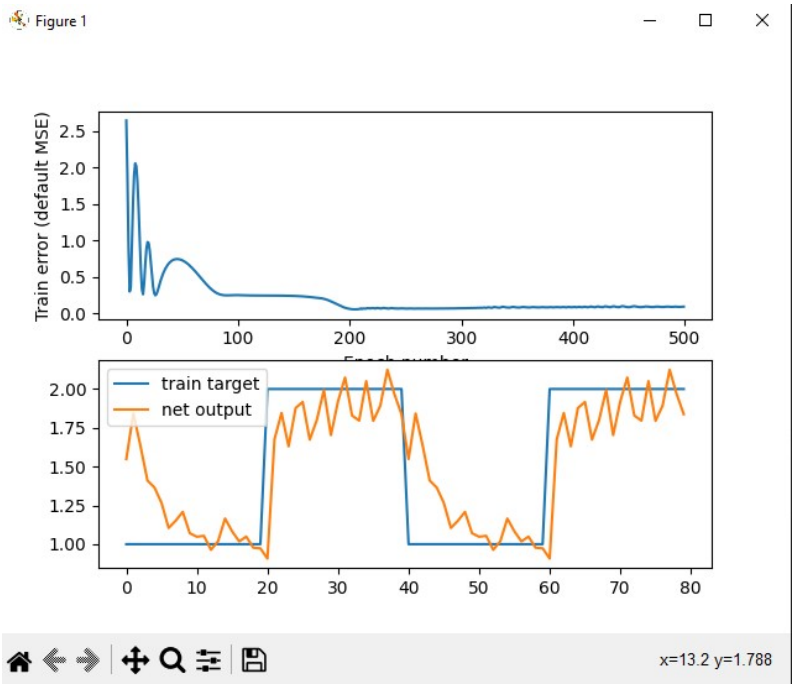


```
--- Epoch 100
Train:  Loss 0.688 | Accuracy: 0.552
Test:   Loss 0.698 | Accuracy: 0.500
--- Epoch 200
Train:  Loss 0.665 | Accuracy: 0.707
Test:   Loss 0.713 | Accuracy: 0.700
--- Epoch 300
Train:  Loss 0.322 | Accuracy: 0.879
Test:   Loss 1.274 | Accuracy: 0.250
--- Epoch 400
Train:  Loss 0.009 | Accuracy: 1.000
Test:   Loss 0.036 | Accuracy: 1.000
--- Epoch 500
Train:  Loss 0.004 | Accuracy: 1.000
Test:   Loss 0.014 | Accuracy: 1.000
--- Epoch 600
Train:  Loss 0.003 | Accuracy: 1.000
Test:   Loss 0.011 | Accuracy: 1.000
--- Epoch 700
Train:  Loss 0.002 | Accuracy: 1.000
Test:   Loss 0.009 | Accuracy: 1.000
--- Epoch 800
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.008 | Accuracy: 1.000
--- Epoch 900
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.007 | Accuracy: 1.000
--- Epoch 1000
Train:  Loss 0.001 | Accuracy: 1.000
Test:   Loss 0.007 | Accuracy: 1.000
PS C:\AI_labs>
```

Висновок: Рекурентна нейронна мережа (RNN) - це тип штучної нейронної мережі, яка призначена для роботи з послідовними даними, такими як текст, часові ряди або звуконаслідування. Основний принцип RNN полягає в тому, що вона має здатність зберігати попередні стани та використовувати їх для обробки поточного вхідного сигналу. RNN є потужним інструментом для роботи з послідовними даними, оскільки вони можуть зберігати попередні стани та враховувати їх при обробці нових вхідних даних. Це дозволяє їм моделювати залежності в часі. RNN складається з повторюючихся блоків, які мають два основних шари: вхідний шар і прихований шар. Попередній стан передається в мережу разом із вхідним сигналом на кожному кроці часу.

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

Графік помилки та апроксимації сигналу:



Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/
Epoch: 100; Error: 0.24853192463960783;
Epoch: 200; Error: 0.2361273281594297;
Epoch: 300; Error: 0.10277809227213353;
Epoch: 400; Error: 0.058687214855084636;
Epoch: 500; Error: 0.032874468907268126;
The maximum number of train epochs is reached
```

Висновок: мережа Елмана успішно навчена передбачати часові ряди на основі вхідних сигналів. Можна бачити, що помилка тренування зменшується з часом, і реальні та передбачені значення на другому графіку виглядають подібно, що свідчить про успішну роботу мережі.

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

Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/AI_labs/lab6/LR_6_task_3.py
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]]
PS C:\AI_labs>
```

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)
```

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

Інформація з вікна терміналу:

```
PS C:\AI_labs> & E:/Users/madma/AppDat
Test on train samples:
N True
E True
R True
O True
PS C:\AI_labs>
```

Інформація з вікна терміналу:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/Windo
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
True Sim. steps 2
PS C:\AI_labs>
```

Тестування з іншою буквою:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/
Test on train samples:
N True
E True
R True
O True

Test on defaced N:
True Sim. steps 2

Test on defaced E:
True Sim. steps 3
PS C:\AI_labs>
```

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())
# Тестування для N
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))
# Тестування для E
print("\nTest on defaced E:")
test_E = np.asfarray(
    [1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1])
test_E[test_E == 0] = -1
out_E = net.sim([test_E])
print((out_E[0] == target[1]).all(), "Sim. steps", len(net.layers[0].outs))
```

Висновок: Мережа успішно розпізнає тренувальні шаблони для літер "N," "E," "R" і "O." При тестуванні мережі зі зіпсованими версіями літер "N" і "E," вона все ще може правильно розпізнати ці літери навіть тоді, коли деякі пікселі змінено. Загалом, мережа Хопфілда виявляється досить ефективною у розпізнаванні шаблонів і може терпимо відноситися до незначних змін пікселів у вхідних шаблонах. Це демонструє одну з переваг мереж Хопфілда, а саме їхню здатність відтворювати шаблони, навіть коли їм надаються шумні або часткові вхідні дані.

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

Лістинг програми:

```
import numpy as np
import neurolab as nl

# Б М О
target = [
    [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],
    [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 = ["Б", "М", "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())
# Тестування для Б
print("\nTest on defaced Б:")
test = np.asfarray(
    [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0]
)
test[test == 0] = -1
out = net.sim([test])
print((out[0] == target[0]).all(), "Sim. steps", len(net.layers[0].outs))
# Тестування для М
print("\nTest on defaced M:")
test_E = np.asfarray(
    [1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
)
test_E[test_E == 0] = -1
out_E = net.sim([test_E])
print((out_E[0] == target[1]).all(), "Sim. steps", len(net.layers[0].outs))
# Тестування для О
print("\nTest on defaced O:")
test_E = np.asfarray(
    [0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0]
)
test_E[test_E == 0] = -1
out_E = net.sim([test_E])
print((out_E[0] == target[2]).all(), "Sim. steps", len(net.layers[0].outs))
```

Результат виконання програми:

```
PS C:\AI_labs> & E:/Users/madma/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/AI_labs/lab6/LR_6_task_5.py
Test on train samples:
Б True
M True
O True

Test on defaced Б:
False Sim. steps 3

Test on defaced M:
True Sim. steps 1

Test on defaced O:
True Sim. steps 1
PS C:\AI_labs>
```

При тестування Б зробив багато помилок щоб показати що мережа може помилятися.

Висновок: Код демонструє здатність мережі Хопфілда до розпізнавання букв "Б," "М," і "О," навіть у випадках, коли пікселі змінені або зіпсовані. Мережа виявляється досить стійкою до змін у вхідних даних і може успішно розпізнавати ці букви навіть після навчання.