Assignment #1

AAT4013 Deep Learning and Computer Vision

Assigned: Tuesday, 3/14, 2023

Due: Tuesday, 3/21, 2023 (by midnight)

- Colab code 2 sets를 사이버캠퍼스에 제출
  - assignment1-1.ipynb
  - assignment1-2.ipynb

각 code는 colab에서 수행되어 결과를 보여야 합니다.

Problem [1] Python의 list와 NumPy array를 사용한 연산의 속도

[1] Python에서 for-loop 을 사용한 list의 operation은 그 연산시간이 매우 길어질 수 있습니다.

그러나 NumPy에서는 array의 연산을 효율적으로 관리하여 그 시간을 단축시킬 수 있습니다. Problem [1]에서는 Python의 list와 NumPy array를 사용한 연산의 속도 차이를 보려고 합니다.

Python list와 NumPy array를 각각 2개씩 생성하고 다음 operations의 연산시간을 측정하는 code를 작성하시오. [20 points]
그 길이가 SIZE인 list/array는 아래와 같이 점진적으로 증가하도록 생성하시오.

[0, 1, 2, 3, …, SIZE-1]

(a) 각 element의 덧셈 (element-wise addition)

즉 더해진 list/array [0+0, 1+1, 2+2, 3+3, … , (SIZE-1) + (SIZE-1)]를 계산하는 시간을 측정하시오.

(b) 2개의 list/array의 dot product (내적)와 2개의 array의 dot product (내적)

즉 두 list/array의 내적 0*0 + 1*1 + 2*2 + 3*3 + … + (SIZE-1)*(SIZE-1) 을 계산하는 시간을 측정하시오.

우선 list/array를 생성하고 덧셈과 내적을 계산하고 계산시간을 출력하는 함수 `assignment1-1_function(SIZE)`를 작성하시오.

Problem [1] Python의 list와 NumPy array를 사용한 연산의 속도

assignment1-1_function 을 사용하여 **SIZE**가 **100, 10,000, 1,000,000** 인 **3가지 경우**에 대해 각 각 아래와 같이 출력하시오.

```
------------------------------------------------------------
SIZE = 1000000
------------------------------------------------------------
List Addition values at 0, SIZE-1 = 0 , 1999998
List addition time [ms] = --------
------------------------------------------------------------
List dot product = 333332833333500000
List dot product time [ms] = --------
------------------------------------------------------------
NumPy Addition values at 0, SIZE-1 = 0 , 1999998
NumPy addition time [ms] = --------
------------------------------------------------------------
NumPy dot product = 333332833333500000
NumPy dot product time [ms] = --------
------------------------------------------------------------
```

- 여기는 **list/array SIZE**가 **1000000** 인 경우
- **HALF, FULL size index**도 출력
- **Index**가 **0, SIZE-1**에서 **list elementwise addition** 값을 출력.
- 그리고 **list addition** 시간을 출력 **[ms]**
- **List**의 **Dot product** (내적) 값을 출력
- 그리고 내적 시간을 출력 **[ms]**
- **Index**가 **0, SIZE-1**에서 **NumPy elementwise addition** 값을 출력.
- 그리고 **NumPy addition** 시간을 출력 **[ms]**
- **NumPy**의 **Dot product** (내적) 값을 출력
- 그리고 내적 시간을 출력 **[ms]**

- 위 결과 예시에서 ------ 에 출력 수치가 보여야합니다.

- SIZE가 100, 10,000, 1,000,000 인 3가지 경우에 대해 연산시간을 비교하면 그 차이를 알 수 있습니다.

Problem [1] Python의 list와 NumPy array를 사용한 연산의 속도 (계속)

[1] 계속

List/Array의 생성과 연산시간의 측정하는 function **assignment1-1_function(SIZE)**은 아래의 code와 같이 시작하시오.

```
import numpy as np
import time

SIZE = 1000000
HALF = int(SIZE/2)
FULL = int(SIZE-1)

# . . .
L1 = range(SIZE)              # Python list
L2 = range(SIZE)              # Python list
A1 = np.arange(SIZE)          # NumPy array
A2 = np.arange(SIZE)          # NumPy array

start = time.time()           # read time here
# computation code between time.time()s
end = time.time()             # read time here

# printing code
# computation time in [ms]: (end-start)*1000

# . . .
```

- 이와 같이 computation 전후로 time을 읽고 그 차이를 출력하면 실행시간이 됩니다.

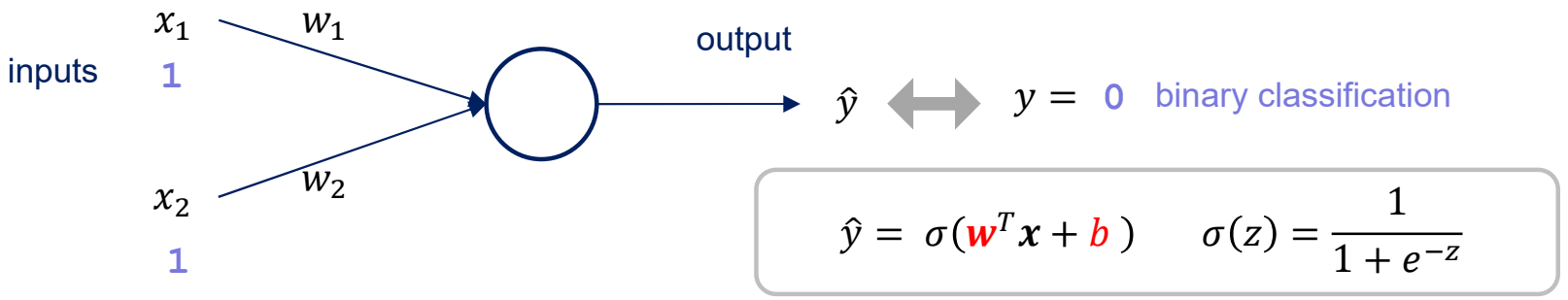
Problem [2] Single neuron model의 logistic regression code

[2] 교재에서는 2-inputs, 3-node hidden layer가 있는 2-3-1 neural net을 예제로 들고, 이의 feedforward/backpropagation code는 course github chapter 1의 **`Back_propagation.ipynb`** 입니다.

https://github.com/PacktPublishing/Modern-Computer-Vision-with-PyTorch

이를 우리가 분석했던 2-1 single-neuron net 에 적용되도록 수정하시오. [30 points]

이 2-1 net은 2-3-1보다 간단하므로 약간 간략화해야 합니다. 최종 **`weights`**와 이를 이용한 결과 $\hat{y}$ 을 보이시오.



- 수정하고 실행하여 결과가 나와있는 colab code를 제출하시오.
- 2-1 net과 2-3-1 net의 loss와 accuracy를 비교하고 그 차이의 원인을 생각해보세요. (제출 필요 없음)

```python
# 2-1 single-neuron logistic regression code
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
x = np.array([[1,1]])
y = np.array([[0]])

from copy import deepcopy
import numpy as np

def feed_forward(inputs, outputs, weights):
    # . . . 작성 . . .
    return mean_squared_error

def update_weights(inputs, outputs, weights, lr):
    # . . . 작성 . . .
    return updated_weights, original_loss

W = [      # W 초기값 설정
    np.array([-0.0053, 0.3793], dtype=np.float32).T,
    np.array([-0.0140], dtype=np.float32),
]

# 100번 update_weights 수행하고 loss plot을 보이는 code 작성 . . .

W    # 최종 Weight print

# 최종 추정치 output을 print하는 code 작성 . . .
```