Lab - Basic Linear Algebra with Pythonic Code

Introduction

PDF 파일 다운로드

이번 Lab은 이번 주차에 간단한 선형대수의 수식들을 python code로 작성하는 것이 목적입니다. 2016년 3월 12일 인간계 바둑왕 이세돌을 완벽하게 이긴 알파고를 만든 기반엔 강화학습이라는 알고리즘이 존재하고 이 알고리즘을 이해하기 위한 Notation으로 기초적인 수학을 알고 있어야 합니다. 지금 당장 알파고처럼 어려운 프로그램을 만들자는 얘기는 아닙니다. 알파고와 같은 복잡한 수학 공식을 프로그래밍 언어로 만들기 전에 수업 시간에 배운 간단한 선형대수의 표현들을 코드로 변환하는 법을 먼저 배울 것 입니다.

숙제 파일(lab_bla.zip) 다운로드

먼저 해야 할 일은 숙제 파일을 다운로드 받는 것 입니다. Chrome 또는 익스플로러와 같은 웹 브라우저 주소창에 아래 주소를 입력합니다.

https://github.com/TeamLab/introduction_to_python_TEAMLAB_MOOC/blob/master/lab_assignment/lab_bla/lab_bla.zip

다운로드를 위해 `View Raw` 또는 `Download` 버튼을 클릭합니다. 또는 아래 다운로드 링크를 클릭하면 자동으로 다운로드가 됩니다. Lab morsecode - 다운로드
다운로드 된 `lab_bla.zip` 파일을 작업 폴더로 이동한 후 압축해제 후 작업하시길 바랍니다.
압축해제 하면 폴더가 `linux_mac` 과 `windows` 로 나눠져 있습니다. 자신의 OS에 맞는 폴더로 이동해서 코드를 수정해 주시기 바랍니다.

basic_linear_algebra.py 코드 구조

본 Lab은 vector와 matrix의 기초적인 연산을 수행하는 12개의 함수를 작성합니다. 각각 함수의 기능과 역할은 아래와 같다. 비교적 문제가 평이하니 웃는 얼굴로 도전해보기 바랍니다. 단 저희가 `one line code available` 이라고 작성한 코드의 경우 `list comprehenstion` 으로 표현이 가능하니 `for loop` 이 아닌 pythonic code로 도전해보시기 바랍니다.

함수명	함수내용

vector_size_check	vector 간 덧셈 또는 뺄셈 연산을 할 때, 연산이 가능한 사이즈인지를 확인하여 가능 여부를 True 또는 False로 반환함
vector_addition	vector간 덧셈을 실행하여 결과를 반환함, 단 입력되는 vector의 갯수와 크기는 일정하지 않음
vector_subtraction	vector간 뺄셈을 실행하여 결과를 반환함, 단 입력되는 vector의 갯수와 크기는 일정하지 않음
scalar_vector_product	하나의 scalar 값을 vector에 곱함, 단 입력되는 vector의 크기는 일정하지 않음
matrix_size_check	matrix 간 덧셈 또는 뺄셈 연산을 할 때, 연산이 가능한 사이즈인지를 확인하여 가능 여부를 True 또는 False로 반환함
is_matrix_equal	비교가 되는 n개의 matrix가 서로 동치인지 확인하여 True 또는 False를 반환함
matrix_addition	matrix간 덧셈을 실행하여 결과를 반환함, 단 입력되는 matrix의 갯수와 크기는 일정하지 않음
matrix_subtraction	matrix간 뺄셈을 실행하여 결과를 반환함, 단 입력되는 matrix의 갯수와 크기는 일정하지 않음
matrix_transpose	matrix의 역행렬을 구하여 결과를 반환함, 단 입력되는 matrix의 크기는 일정하지 않음
scalar_matrix_product	하나의 scalar 값을 matrix에 곱함, 단 입력되는 matrix의 크기는 일정하지 않음
is_product_availability_matrix	두 개의 matrix가 입력 되었을 경우, 두 matrix의 곱셈 연산의 가능 여부를 True 또는 False로 반환함
matrix_product	곱셈 연산이 가능한 두 개의 matrix의 곱셈을 실행하여 반환함

Problem #1 - vector_size_check (one line code available)

```
def vector_size_check(*vector_variables):
    return None
```

```
# 실행결과
print(vector_size_check([1,2,3], [2,3,4], [5,6,7])) # Expected value: True
print(vector_size_check([1, 3], [2,4], [6,7])) # Expected value: True
```

```
print(vector_size_check([1, 3, 4], [4], [6,7])) # Expected value: False
```

Problem #2 - vector_addition (one line code available)

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a+x \\ b+y \\ c+z \end{bmatrix}$$

```
def vector_addition(*vector_variables):
    return None
```

```
# 실행결과
print(vector_addition([1, 3], [2, 4], [6, 7])) # Expected value: [9, 14]
print(vector_addition([1, 5], [10, 4], [4, 7])) # Expected value: [15, 16]
print(vector_addition([1, 3, 4], [4], [6,7])) # Expected value: ArithmeticError
```

Problem #3 - vector_subtraction (one line code available)

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} - \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a-x \\ b-y \\ c-z \end{bmatrix}$$

```
def vector_subtraction(*vector_variables):
    if vector_size_check(*vector_variables) == False:
        raise ArithmeticError
    return None
```

```
# 실행결과
print(vector_subtraction([1, 3], [2, 4])) # Expected value: [-1, -1]
print(vector_subtraction([1, 5], [10, 4], [4, 7])) # Expected value: [-13, -6]
```

Problem #4 - scalar_vector_product (one line code available)

$$\alpha \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \alpha \times x \\ \alpha \times y \\ \alpha \times z \end{bmatrix}$$

```
def scalar_vector_product(alpha, vector_variable):
    return None
```

```
# 실행결과
print (scalar_vector_product(5,[1,2,3])) # Expected value: [5, 10, 15]
print (scalar_vector_product(3,[2,2])) # Expected value: [6, 6]
print (scalar_vector_product(4,[1])) # Expected value: [4]
```

Problem #5 - matrix_size_check (one line code available)

```
def matrix_size_check(*matrix_variables):
    return None
```

```
# 실행결과
matrix_x = [[2, 2], [2, 2], [2, 2]]
matrix_y = [[2, 5], [2, 1]]
matrix_z = [[2, 4], [5, 3]]
matrix_w = [[2, 5], [1, 1], [2, 2]]

print (matrix_size_check(matrix_x, matrix_y, matrix_z)) # Expected value: False
print (matrix_size_check(matrix_y, matrix_z)) # Expected value: True
print (matrix_size_check(matrix_x, matrix_w)) # Expected value: True
```

Problem #6 - is_matrix_equal (one line code available)

if $x = a, y = b, z = c, w = d$ then

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
def is_matrix_equal(*matrix_variables):
    return None
```

```
# 실행결과
matrix_x = [[2, 2], [2, 2]]
matrix_y = [[2, 5], [2, 1]]

print (is_matrix_equal(matrix_x, matrix_y, matrix_y, matrix_y)) # Expected value: F
print (is_matrix_equal(matrix_x, matrix_x)) # Expected value: True
```

Problem #7 - matrix_addition (one line code available)

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} + \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} x+a & y+b \\ z+c & w+d \end{bmatrix}$$

```
def matrix_addition(*matrix_variables):
    if matrix_size_check(*matrix_variables) == False:
        raise ArithmeticError
    return None
```

```
# 실행결과
matrix_x = [[2, 2], [2, 2]]
matrix_y = [[2, 5], [2, 1]]
matrix_z = [[2, 4], [5, 3]]

print (matrix_addition(matrix_x, matrix_y)) # Expected value: [[4, 7], [4, 3]]
print (matrix_addition(matrix_x, matrix_y, matrix_z)) # Expected value: [[6, 11], [
```

Problem #8 - matrix_subtraction (one line code available)

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} - \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} x-a & y-b \\ z-c & w-d \end{bmatrix}$$

```
def matrix_subtraction(*matrix_variables):
    if matrix_size_check(*matrix_variables) == False:
        raise ArithmeticError
    return None
```

```
# 실행결과
matrix_x = [[2, 2], [2, 2]]
matrix_y = [[2, 5], [2, 1]]
matrix_z = [[2, 4], [5, 3]]

print (matrix_subtraction(matrix_x, matrix_y)) # Expected value: [[0, -3], [0, 1]]
print (matrix_subtraction(matrix_x, matrix_y, matrix_z)) # Expected value: [[-2, -7
```

Problem #9 - matrix_transpose (one line code available)

Let

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}$$, Then $$A^T = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix}$$

```
def matrix_transpose(matrix_variable):
    return None
```

```
# 실행결과
matrix_w = [[2, 5], [1, 1], [2, 2]]
matrix_transpose(matrix_w)
```

Problem #10 - scalar_matrix_product (one line code available)

$$\alpha \times \begin{bmatrix} a & c & d \\ e & f & g \end{bmatrix} = \begin{bmatrix} \alpha \times a & \alpha \times c & \alpha \times d \\ \alpha \times e & \alpha \times f & \alpha \times g \end{bmatrix}$$

```
def scalar_matrix_product(alpha, matrix_variable):
    return None
```

```
# 실행결과
matrix_x = [[2, 2], [2, 2], [2, 2]]
matrix_y = [[2, 5], [2, 1]]
matrix_z = [[2, 4], [5, 3]]
```

```
matrix_w = [[2, 5], [1, 1], [2, 2]]

print(scalar_matrix_product(3, matrix_x)) #Expected value: [[6, 6], [6, 6], [6, 6]]
print(scalar_matrix_product(2, matrix_y)) #Expected value: [[4, 10], [4, 2]]
print(scalar_matrix_product(4, matrix_z)) #Expected value: [[8, 16], [20, 12]]
print(scalar_matrix_product(3, matrix_w)) #Expected value: [[6, 15], [3, 3], [6, 6]
```

Problem #11 - is_product_availability_matrix (one line code available)

The matrix product of A and B (written AB) is defined if and only if
Number of columns in A = Number of rows in B

```
def is_product_availability_matrix(matrix_a, matrix_b):
    return None
```

```
# 실행결과
matrix_x= [[2, 5], [1, 1]]
matrix_y = [[1, 1, 2], [2, 1, 1]]
matrix_z = [[2, 4], [5, 3], [1, 3]]

print(is_product_availability_matrix(matrix_y, matrix_z)) # Expected value: True
print(is_product_availability_matrix(matrix_z, matrix_x)) # Expected value: True
print(is_product_availability_matrix(matrix_z, matrix_w)) # Expected value: False /
print(is_product_availability_matrix(matrix_x, matrix_x)) # Expected value: True
```

Problem #12 - matrix_product (one line code available)

```
def matrix_product(matrix_a, matrix_b):
    if is_product_availability_matrix(matrix_a, matrix_b) == False:
        raise ArithmeticError
    return None
```

```
# 실행결과
matrix_x= [[2, 5], [1, 1]]
matrix_y = [[1, 1, 2], [2, 1, 1]]
matrix_z = [[2, 4], [5, 3], [1, 3]]
```

```
print(matrix_product(matrix_y, matrix_z)) # Expected value: [[9, 13], [10, 14]]
print(matrix_product(matrix_z, matrix_x)) # Expected value: [[8, 14], [13, 28], [5,
print(matrix_product(matrix_x, matrix_x)) # Expected value: [[9, 15], [3, 6]]
print(matrix_product(matrix_z, matrix_w)) # Expected value: False
```

숙제 template 파일 제출하기 (윈도우의 경우)

1. `windows` + `r` 를 누르고 cmd 입력 후 확인을 클릭합니다.
2. 작업을 수행한 폴더로 이동 합니다.
3. 밑에 명령어를 cmd창에 입력합니다.

```
install.bat
submit.bat [YOUR_HASH_KEY]
```

숙제 template 파일 제출하기 (Mac or Linux)

1. 터미널을 구동합니다.
2. 작업을 수행한 디렉토리로로 이동 합니다.
3. 밑에 bash창을 입력합니다.

```
bash install.sh
bash submit.sh [YOUR_HASH_KEY]
```

> backend.ai 서비스의 업데이트에 의해 실행전 반드시 `bash install.sh` 또는 `install.bat` 수행을 바랍니다.

Next Work

고생하셨습니다. 처음 해보는 거라 너무 생소하게 느꼈을 가능성이 클거라고 생각합니다. "너무 어렵다"라는 말도 많이 했을 거 같습니다. 이제 파이썬은 왠만큼 하실 수 있게 되셨습니다. 축하합니다.

> **Human knowledge belongs to the world** - from movie 'Password' -