실습 Zero
:기계학습 구현에 자주 쓰는 기초 문법들

SW융합학부 양희경

실습 내용

실습 내용	인공지능기초	심층학습
0. AWS 접속 및 Jupyter notebook 사용법	●	●
1. 파이썬 기초(Optional)		
2. 벡터, 행렬 연산, 그래프 그리기	●	
3. 파이토치 기초		●
4. MNIST 데이터로드		

실습 내용

1. **파이썬 기초(Optional)**
2. 벡터, 행렬 연산, 그래프 그리기

{**인공지능기초, 심층학습**}
{기계학습, 자연어처리}
실습에 필수

3. 파이토치 기초
4. MNIST 데이터로드

{심층학습}, **{자연어처리}**
실습에 필수

- 라이브러리: Numpy, Pyplot, PyTorch

1. 파이썬 기초 (Optional)

1. 파이썬 기초(Optional)

프린트

```
print ("Hello, world")

# integer
x = 3
print ("정수: %01d, %02d, %03d, %04d, %05d"
      % (x,x,x,x,x))

# float
x = 256.123
print ("실수: %.0f, %.1f, %.2f"
      % (x,x,x))

# string
x = "Hello, world"
print ("문자열: [%s]" % (x))
```

```
Hello, world
정수: 3, 03, 003, 0003, 00003
실수: 256, 256.1, 256.12
문자열: [Hello, world]
```

1. 파이썬 기초 (Optional)

반복문, 조건문

```
contents = ["Regression", "Classification", "SVM", "Clustering", "Demension reduction",
            "NN", "CNN", "AE", "GAN", "RNN"]
for con in contents:
    if con in ["Regression", "Classification", "SVM", "Clustering", "Demension reduction"]:
        print ("%s 은(는) 기계학습 내용입니다." %con )
    elif con in ["CNN"]:
        print ("%s 은(는) convolutional neural network 입니다." %con)
    else:
        print ("%s 은(는) 심층학습 내용입니다.")
```

```
Regression 은(는) 기계학습 내용입니다.
Classification 은(는) 기계학습 내용입니다.
SVM 은(는) 기계학습 내용입니다.
Clustering 은(는) 기계학습 내용입니다.
Demension reduction 은(는) 기계학습 내용입니다.
%s 은(는) 심층학습 내용입니다.
CNN 은(는) convolutional neural network 입니다.
%s 은(는) 심층학습 내용입니다.
%s 은(는) 심층학습 내용입니다.
%s 은(는) 심층학습 내용입니다.
```

1. 파이썬 기초 (Optional)

반복문과 인덱스

```
for (i,con) in enumerate(contents):
    print ("[%d/%d]: %s" %(i, len(contents), con))
```

```
[0/10]: Regression
[1/10]: Classification
[2/10]: SVM
[3/10]: Clustering
[4/10]: Demension reduction
[5/10]: NN
[6/10]: CNN
[7/10]: AE
[8/10]: GAN
[9/10]: RNN
```

1. 파이썬 기초 (Optional)

함수

```
def sum(a,b):
    return a+b

x = 10.0
y = 20.0
print ("%.1f + %.1f = %.1f" %(x, y, sum(x,y)))
```

```
10.0 + 20.0 = 30.0
```

1. 파이썬 기초 (Optional)

리스트

```
a = []
b = [1,2,3]
c = ["Hello", ",", "world"]
d = [1,2,3,"x","y","z"]
x = []
print (x)

x.append('a')
print (x)

x.append(123)
print (x)

x.append(["a", "b"])
print x
```

```
[]
['a']
['a', 123]
['a', 123, ['a', 'b']]
```

1. 파이썬 기초 (Optional)

딕셔너리(dictionary)

```
dic = dict()
dic["name"] = "Heekyung"
dic["town"] = "Goyang city"
dic["job"] = "Assistant professor"
print dic
```

```
{'town': 'Goyang city', 'job': 'Assistant professor', 'name': 'Heekyung'}
```

1. 파이썬 기초 (Optional)

클래스

```
class Student:
    # 생성자
    def __init__(self, name):
        self.name = name
    # 메써드
    def study(self, hard=False):
        if hard:
            print "%s 학생은 열심히 공부합니다." %self.name
        else:
            print "%s 학생은 공부합니다." %self.name

s = Student('Heekyung')
s.study()
s.study(hard=True)
```

```
Heekyung 학생은 공부합니다.
Heekyung 학생은 열심히 공부합니다.
```

실습 내용

1. 파이썬 기초(Optional)

2. 벡터, 행렬 연산, 그래프 그리기

3. 파이토치 기초

4. MNIST 데이터로드

- 라이브러리: Numpy, Pyplot, PyTorch

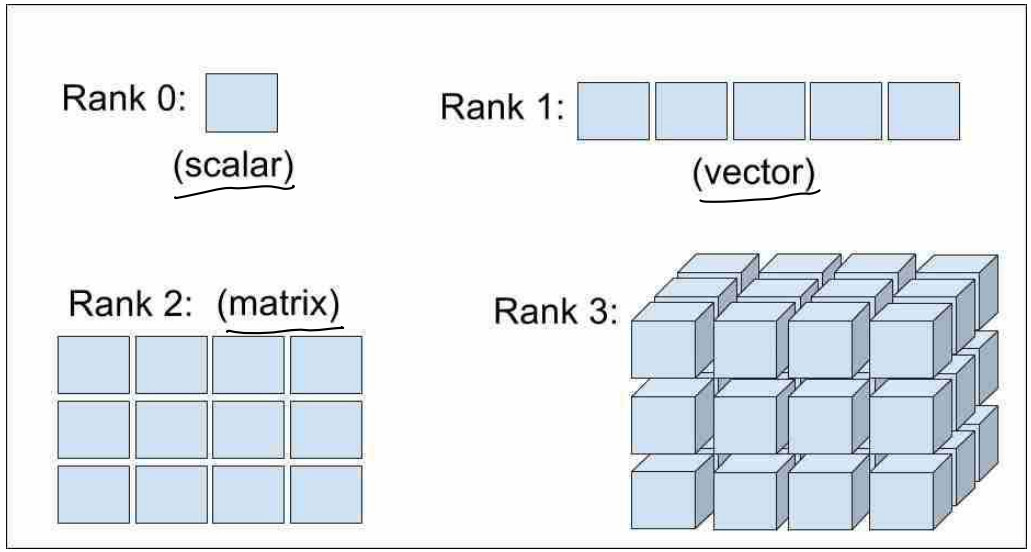
2. 벡터, 행렬 연산, 그래프 그리기

- 텐서(Tensor)?
  - 다차원 배열을 일반화한 것
  - Scalar, vector, matrix 등이 포함됨

데이터
feature

- 랭크(Rank)?
  - 텐서의 차원
  - Rank 0 tensor: Scalar
  - Rank 1 tensor: Vector
  - Rank 2 tensor: Matrix
  - Rank n tensor

shape: 텐서 각 축의 크기



© mc.ai

2. 벡터, 행렬 연산, 그래프 그리기

라이브러리(패키지) 로드

```
import numpy as np
```

2. 벡터, 행렬 연산, 그래프 그리기

프린트

```
def print_val(x):
    print "Type:", type(x)
    print "Shape:", x.shape
    print "값:₩n", x
    print " "
```

2. 벡터, 행렬 연산, 그래프 그리기

rank 1 np array

```
x = np.array([1, 2, 3])
print_val(x)

x[0] = 5
print_val(x)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3,)
값:
[1 2 3]

Type: <type 'numpy.ndarray'>
Shape: (3,)
값:
[5 2 3]
```

2. 벡터, 행렬 연산, 그래프 그리기

rank 2 np array

```
y = np.array([[1,2,3], [4,5,6]])
print_val(y)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 3)
값:
[[1 2 3]
 [4 5 6]]
```

rank 2 zeros

```
a = np.zeros((2,2))
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0. 0.]
 [0. 0.]]
```

rank 2 ones

```
a = np.ones((3,2))
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3, 2)
값:
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

rank 2 단위 행렬(identity matrix)

```
a = np.eye(3,3)
print_val(a)
```

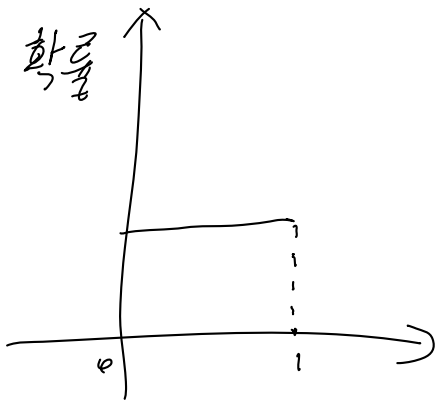
```
Type: <type 'numpy.ndarray'>
Shape: (3, 3)
값:
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

랜덤 행렬(uniform: 0~1 사이 모든 값들이 나올 확률이 같음)

```
a = np.random.random((4,4))
print_val(a)
```

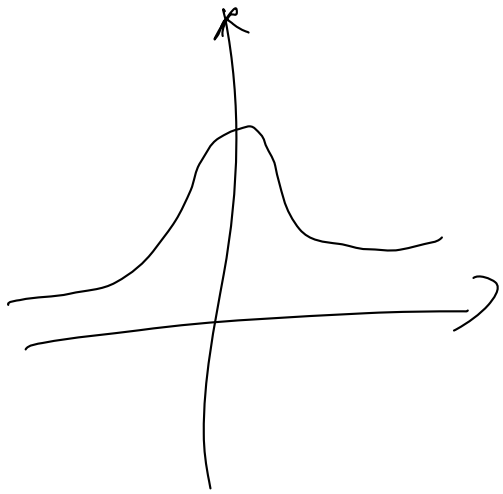
```
Type: <type 'numpy.ndarray'>
Shape: (4, 4)
값:
[[0.51188499 0.24694867 0.88542043 0.3671004 ]
 [0.56884716 0.92298505 0.17967754 0.4287874 ]
 [0.01890182 0.61939264 0.95876775 0.96522488]
 [0.88609591 0.89879732 0.39578545 0.05220255]]
```



랜덤 행렬(Gaussian: 0을 평균으로 하는 가우시안 분포를 따르는 랜덤값)

```
a = np.random.randn(4,4)
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4, 4)
값:
[[-1.09098861  1.94289236  1.54194447  0.95220594]
 [ 1.71442595 -0.96202191 -0.37320804 -1.32446336]
 [-1.97234896  0.0756659  -1.05772135  0.34975056]
 [ 0.70735129  0.64681658 -0.09711383  0.34148813]]
```



2. 벡터, 행렬 연산, 그래프 그리기

np array indexing

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print_val(a)
```

```
Type: <type 'numpy.ndarray'>
Shape: (3, 4)
값:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
b = a[:2, 1:3]  # 행 0~1, 열 1~2
print_val(b)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[2 3]
 [6 7]]
```

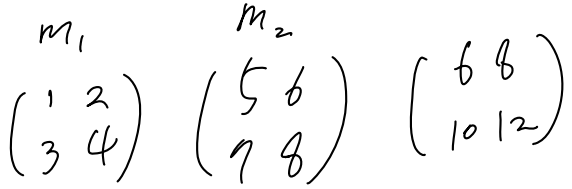
행렬의 n번째 행 얻기

```
row1 = a[1, :]  # 1번째 행
print_val(row1)
```

```
Type: <type 'numpy.ndarray'>
Shape: (4,)
값:
[5 6 7 8]
```

2. 벡터, 행렬 연산, 그래프 그리기

행렬의 원소별 연산 - 더하기




```
m1 = np.array([[1,2], [3,4]], dtype=np.float64)
m2 = np.array([[5,6], [7,8]], dtype=np.float64)
```

```
# elementwise sum
print_val(m1 + m2)
print_val(np.add(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 6.  8.]
 [10. 12.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 6.  8.]
 [10. 12.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

빼기

```
# elementwise difference
print_val(m1 - m2)
print_val(np.subtract(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[-4. -4.]
 [-4. -4.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[-4. -4.]
 [-4. -4.]]
```

곱하기

```
# elementwise product
print_val(m1 * m2)
print_val(np.multiply(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 5. 12.]
 [21. 32.]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[ 5. 12.]
 [21. 32.]]
```

2. 벡터, 행렬 연산, 그래프 그리기

나누기

```
# elementwise division
print_val(m1 / m2)
print_val(np.divide(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0.2        0.33333333]
 [0.42857143 0.5       ]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

루트

```
# elementwise square root
print_val(np.sqrt(m1))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

2. 벡터, 행렬 연산, 그래프 그리기

행렬 연산

```
m1 = np.array([[1,2], [3,4]])  # (2,2)
m2 = np.array([[5,6], [7,8]])  # (2,2)
v1 = np.array([9,10])  # (2,1)   # [[9,10]]  (1,2)
v2 = np.array([11,12])  # (2,1)

print_val(m1)
print_val(m2)
print_val(v1)
print_val(v2)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 2]
 [3 4]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[5 6]
 [7 8]]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[ 9 10]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[11 12]
```

2. 벡터, 행렬 연산, 그래프 그리기

벡터-벡터 연산 - 내적연산

dot product (내적)

(9, 10) · (11, 12) = 9×11 + 10×12

```
print_val(v1.dot(v2))
print_val(np.dot(v1, v2))
```

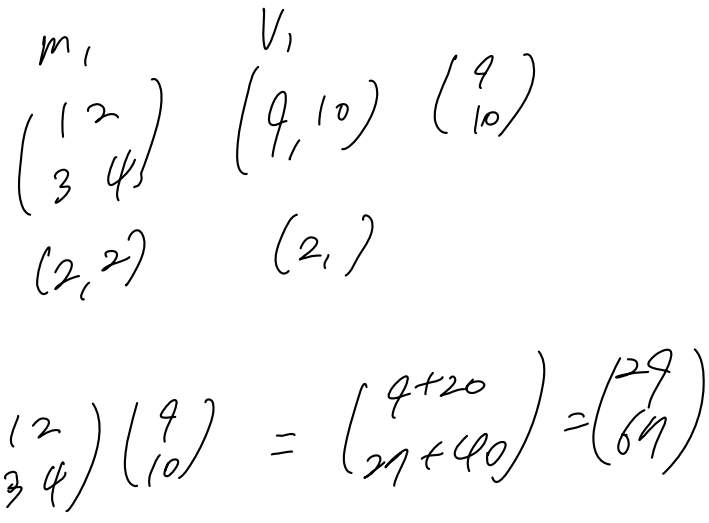
```
Type: <type 'numpy.int64'>
Shape: ()
값:
219

Type: <type 'numpy.int64'>
Shape: ()
값:
219
```

2. 벡터, 행렬 연산, 그래프 그리기

벡터-행렬 연산 - 내적연산

```
print_val(m1.dot(v1))  # (2,2) x (2,1) -> (2,1)
print_val(np.dot(m1, v1))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[29 67]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[29 67]
```



2. 벡터, 행렬 연산, 그래프 그리기

행렬-행렬 연산 - 내적연산

```
print_val(m1.dot(m2))
print_val(np.dot(m1, m2))
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[19 22]
 [43 50]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[19 22]
 [43 50]]
```

m_1 m_2

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

2x2 2x2

2. 벡터, 행렬 연산, 그래프 그리기

transpose matrix

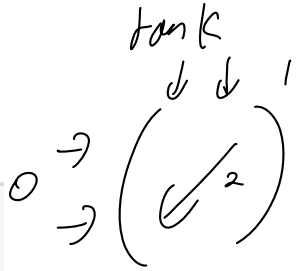
전치 행렬 (transpose)

```
print_val(m1)
print_val(m1.T)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 2]
 [3 4]]

Type: <type 'numpy.ndarray'>
Shape: (2, 2)
값:
[[1 3]
 [2 4]]
```

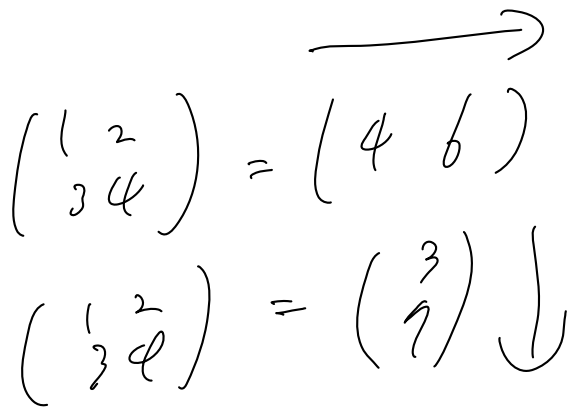
2. 벡터, 행렬 연산, 그래프 그리기

합




```
print_val(np.sum(m1))  # 행렬의 모든 원소의 합
print_val(np.sum(m1, axis=0))  # shape[0] (행) 을 압축시키자. (2,2) -> (2,)
print_val(np.sum(m1, axis=1))  # shape[1] (열) 을 압축시키자. (2,2) -> (2,)
```

```
Type: <type 'numpy.int64'>
Shape: ()
값:
10

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[4 6]

Type: <type 'numpy.ndarray'>
Shape: (2,)
값:
[3 7]
```



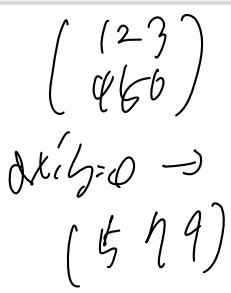
2. 벡터, 행렬 연산, 그래프 그리기

```
m1 = np.array([[1,2,3], [4,5,6]])
print m1
print m1.shape  # (2,3)
```

```
[[1 2 3]
 [4 5 6]]
```

```
print np.sum(m1)
print np.sum(m1, axis=0)  # shape[0] (행) 을 압축시키자. (2,3) ->
print np.sum(m1, axis=1)  # shape[1] (열) 을 압축시키자. (2,3) ->
```

21



2. 벡터, 행렬 연산, 그래프 그리기

zeros-like

```
m1 = np.array([[1,2,3],
               [4,5,6],
               [7,8,9],
               [10,11,12]])
m2 = np.zeros_like(m1)  # m1과 같은 형태의 0으로 이루어진 np array
print_val(m1)
print_val(m2)
```

→ 같은 shape

```
Type: <type 'numpy.ndarray'>
Shape: (4, 3)
값:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

Type: <type 'numpy.ndarray'>
Shape: (4, 3)
값:
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

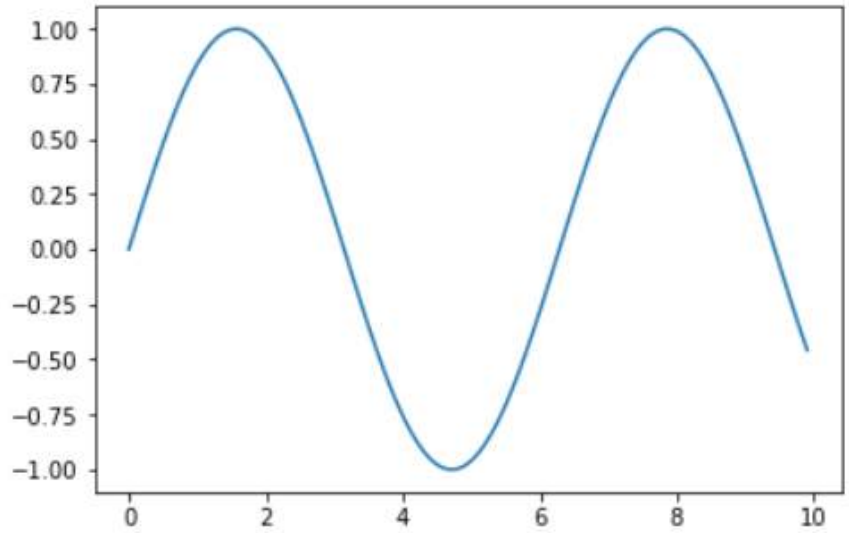
2. 벡터, 행렬 연산, 그래프 그리기

Matplot library

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# sin 커브
x = np.arange(0, 10, 0.1)  # 0~10 까지 0.1 간격의 숫자 배열
y = np.sin(x)

plt.plot(x, y)
```

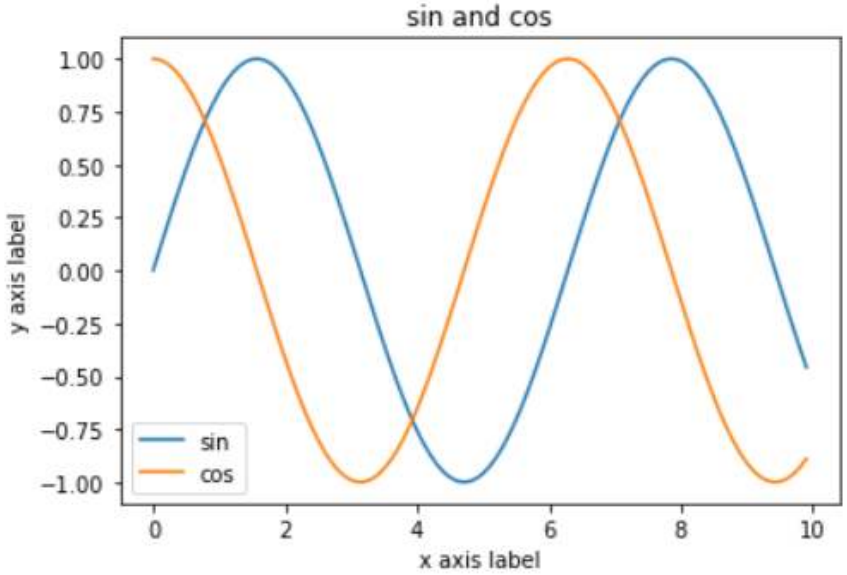
```
[<matplotlib.lines.Line2D at 0x7f202b9d7810>]
```



2. 벡터, 행렬 연산, 그래프 그리기

한 번에 두 개 그래프 그리기

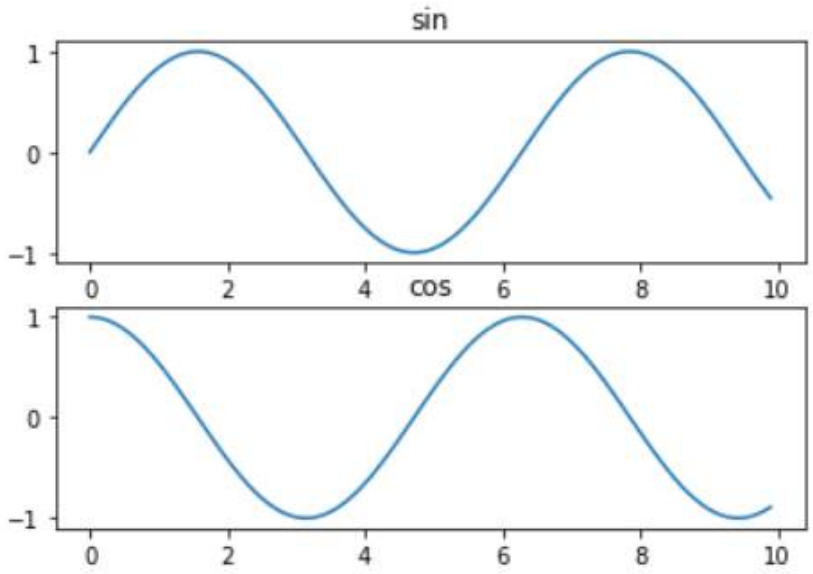
```
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('sin and cos')
plt.legend(['sin', 'cos'])

plt.show()
```



2. 벡터, 행렬 연산, 그래프 그리기

Subplot

```
plt.subplot(2, 1, 1)  # (2,1) 형태 플랏의 첫 번째 자리에 그리겠다
plt.plot(x, y_sin)
plt.title('sin')

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('cos')

plt.show()
```



실습 내용

1. 파이썬 기초(Optional)
2. 벡터, 행렬 연산, 그래프 그리기

{기계학습, 심층학습}
{인공지능기초, 자연어처리}
실습에 필요

3. 파이토치 기초
4. MNIST 데이터로드

{심층학습}, {자연어처리}
실습에 필요

- 라이브러리: Numpy, Pyplot, PyTorch

3. 파이토치 기초

3. 파이토치 기초

```
import torch
```

```
x = torch.rand(2,3,2)  # uniform: 0~1 사이 모든 값들이 나올 확률이 같음
print_val(x)

x = torch.randn(2,3)  # Gaussian: 0을 평균으로 하는 가우시안 분포를 따르는 랜덤값
print_val(x)
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 3, 2])
값:
tensor([[[0.2631, 0.7858],
         [0.2050, 0.8946],
         [0.9223, 0.6038]],

        [[0.9886, 0.4532],
         [0.9419, 0.4712],
         [0.3295, 0.3157]]])

Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 3])
값:
tensor([[ 0.6841,  0.3819, -0.8850],
        [ 0.4542,  0.3165,  0.3829]])
```

3. 파이토치 기초

```
x = torch.randperm(4)  # 0 ~ n-1 까지 값을 원소로 갖는 배열 -> 랜덤 배열
print_val(x)
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([4])
값:
tensor([2, 1, 0, 3])
```

```
# [start, end) 구간을 step 단위로 나눠서 배열로 생성
x = torch.arange(0, 3, step=0.5)
print_val(x)
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([6])
값:
tensor([0.0000, 0.5000, 1.0000, 1.5000, 2.0000, 2.5000])
```

3. 파이토치 기초

```
print_val( torch.ones(3,4) )
print_val( torch.zeros(3,4) )
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([3, 4])
값:
tensor([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]])

Type: <class 'torch.Tensor'>
Shape: torch.Size([3, 4])
값:
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```

```
print_val( torch.Tensor(2,4) )
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 4])
값:
tensor([[1.2994e+14, 4.5740e-41, 1.2994e+14, 4.5740e-41],
        [0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00]])
```

3. 파이토치 기초

```
print_val( torch.Tensor([2,3]) )

print_val( torch.Tensor([[2,3], [3,4]]) )
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([2])
값:
tensor([2., 3.])

Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 2])
값:
tensor([[2., 3.],
        [3., 4.]])
```

```
# 텐서 타입 변환
x = torch.FloatTensor(2,3)
print_val(x)

x = x.type_as(torch.IntTensor())
print_val(x)
```

```
Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 3])
값:
tensor([[6.7297e-37, 2.1990e-40, 2.3511e-38],
        [3.8688e+25, 9.2754e-39, 4.9720e-40]])

Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 3])
값:
tensor([[          0,           0,           0],
        [-2147483648,           0,           0]], dtype=torch.int32)
```

3. 파이토치 기초

```
# 텐서 크기 반환
torch.FloatTensor(3,4,5).size()
```

```
torch.Size([3, 4, 5])
```

```
# numpy array -> torch tensor
import numpy as np
np_x = np.ndarray((2,3), dtype=int)
print_val(np_x)

x = torch.from_numpy(np_x)
print_val(x)
```

```
Type: <type 'numpy.ndarray'>
Shape: (2, 3)
값:
[[140193482005464 140193482005464 140193431408288]
 [        9392928 140192827454528               65]]

Type: <class 'torch.Tensor'>
Shape: torch.Size([2, 3])
값:
tensor([[140193482005464, 140193482005464, 140193431408288],
        [        9392928, 140192827454528,              65]])
```

실습 내용

1. 파이썬 기초(Optional)

2. 벡터, 행렬 연산, 그래프 그리기

3. 파이토치 기초

4. MNIST 데이터로드

- 라이브러리: Numpy, Pyplot, PyTorch

4. MNIST 데이터로드

- Modified National Institute of Standards and Technology database
- 손으로 쓴 숫자 데이터 셋
- 0~9 카테고리 있음
- 학습 데이터 60,000, 테스트 데이터 10,000
- [Yann LeCun의 웹사이트]

4. MNIST 데이터로드

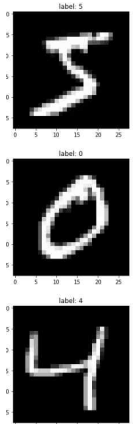
```
import torchvision.datasets as dset
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torch.autograd import Variable
import matplotlib.pyplot as plt
%matplotlib inline
```

4. MNIST 데이터로드

0) MNIST 는 어떻게 생겼나?

```
# 1. MNIST 로드
mnist_train=dset.MNIST("", train=True,transform=transforms.ToTensor(),   #train 용으로 쓰겠다.
                       target_transform=None, download=True)

# 2. 그려보기
# MNIST data 하나 형태 출력
image, label = mnist_train.__getitem__(0)
print image.size(), label

image, label = mnist_train[1]
print image.size(), label

print mnist_train.__len__()
print "mnist_train 길이:", len(mnist_train)

# 그리기
print mnist_train[0][1]  # label
print mnist_train[0][0].size()  # image

for i in range(3):
    img = mnist_train[i][0].numpy()  # image 타입을 numpy 로 변환 (1,28,28)
    #print "label:", mnist_train[i][1]
    plt.title('label: %d' %mnist_train[i][1] )
    plt.imshow(img[0], cmap='gray')
    plt.show()
```

```
torch.Size([1, 28, 28]) 5
torch.Size([1, 28, 28]) 0
60000
mnist_train 길이: 60000
5
torch.Size([1, 28, 28])
```



4. MNIST 데이터로드

1) MNIST train, test dataset 가져오기

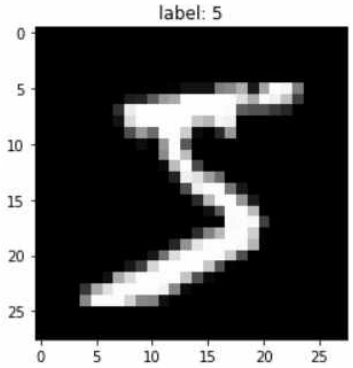
```
# "": 현재 폴더에 MNIST 있음
mnist_train=dset.MNIST("", train=True,transform=transforms.ToTensor(),   #train 용으로 쓰겠다.
                       target_transform=None, download=True)
mnist_test=dset.MNIST("", train=False,transform=transforms.ToTensor(),   #test 용으로 쓰겠다.
                       target_transform=None, download=True)
```

4. MNIST 데이터로드

2) 대략적인 데이터 형태

```
print "mnist_train 길이:", len(mnist_train)
print "mnist_test 길이:", len(mnist_test)

# 데이터 하나 형태
image, label = mnist_train.__getitem__(0)  # 0번째 데이터
print "image data 형태:", image.size()
print "label: ", label

# 그리기
img = image.numpy()  # image 타입을 numpy 로 변환 (1,28,28)
plt.title("label: %d" %label )
plt.imshow(img[0], cmap='gray')
plt.show()
```

```
mnist_train 길이: 60000
mnist_test 길이: 10000
image data 형태: torch.Size([1, 28, 28])
label:  5
```



4. MNIST 데이터로드

3) 데이터 로드함수

학습시킬 때 batch_size 단위로 끊어서 로드하기 위함

```
batch_size = 16
```

```
train_loader = torch.utils.data.DataLoader(list(mnist_train)[:batch_size*100], batch_size=batch_size,
                                           # mnist_train 를 트레인 시키자.
                                           shuffle=True, num_workers=2,
                                           drop_last=True)  # batch_size 만큼 나눌 때 나머지는 버려라
test_loader = torch.utils.data.DataLoader((mnist_test), batch_size=batch_size,
                                           shuffle=False, num_workers=2,
                                           drop_last=True)
```

4. MNIST 데이터로드

4) 데이터 로드함수 이해하기

```
n = 3  # 샘플로 그려볼 데이터 개수
for i, [imgs, labels] in enumerate(test_loader):  # batch_size 만큼
    if i>5:
        break

    print "[%d]" %i
    print "한 번에 로드되는 데이터 크기:", len(imgs)

    # GPU 에 로드되기 위함. 만약 CPU로 설정되어 있다면 자동으로(?) CPU로 로드됨
    x = Variable(imgs)#.cuda()     # (batch_size, 1, 28, 28)
    x = x.reshape((x.shape[0], x.shape[2], x.shape[3]))  # (batch_size, 1, 28, 28) -> (batch_size, 28, 28)
    y_ = Variable(labels)#.cuda()  # (batch_size)

    print x.shape
    print y_.shape

    # 그리기
    for j in range(n):
        img = imgs[j].numpy()  # image 타입을 numpy 로 변환 (1,28,28)
        img = img.reshape((img.shape[1], img.shape[2]))  # (1,28,28) -> (28,28)
        #print img.shape

        plt.subplot(1, n, j+1)  # (1,3) 형태 플랏의 j 번째 자리에 그리겠다
        plt.imshow(img, cmap='gray')
        plt.title("label: %d" %labels[j] )
    plt.show()
```

```
[0]
한 번에 로드되는 데이터 크기: 16
torch.Size([16, 28, 28])
torch.Size([16])
```