20171664 소프트웨어학부 이범석

HW4 p91 ~ p165 In and Out Practice

실습 결과

1. numpy를 import했습니다.

In [1]:
```
#리스트 4-1-(1)
import numpy as np
```

2. np.array를 사용하여 벡터 a를 정의했습니다.

In [2]:
```
#리스트 4-1-(2)
a=np.array([2,1])
print(a)
```

```
[2 1]
```

3. type을 이용하여 a의 type이 numpy.ndarray형임을 알았습니다.

In [3]:
```
#리스트 4-1-(3)
type(a)
```

Out[3]:
```
numpy.ndarray
```

4. ndarray형 2차원 배열을 나타냈습니다.

In [4]:
```
#리스트 4-1-(4)
c=np.array([[1,2],[3,4]])
print(c)
```

```
[[1 2]
 [3 4]]
```

5. 세로 벡터를 나타냈습니다.

In [5]:
```
#리스트 4-1-(5)
d=np.array([[1],[2]])
print(d)
```

```
[[1]
 [2]]
```

6. 전치를 나타냈습니다.

In [6]:
```
#리스트 4-1-(6)
print(d.T)
```

```
[[1 2]]
```

7. ndarray 형 리스트 2개를 더했습니다.

   a와 b가 list 형식이 아닌 벡터로 다루어지고 있는 것을 알 수 있습니다.

In [7]:
```
#리스트 4-1-(7)
a = np.array([2,1])
b = np.array([1,3])
print(a+b)
```

```
[3 4]
```

8. 뺄셈도 해 보았습니다.

In [8]:
```
#리스트 4-1-(8)
a = np.array([2,1])
b = np.array([1,3])
print(a - b)
```

```
[ 1 -2]
```

9. 스칼라에 벡터를 곱했습니다.

   스칼라 값이 벡터의 요소 전체에 적용되었습니다.

In [9]:
```
#리스트 4-1-(9)
print(2 * a)
```

```
[4 2]
```

10. dot을 이용하여 내적을 계산했습니다.

In [10]:
```
#리스트 4-1-(10)
b = np.array([1,3])
c = np.array([4,2])
print(b.dot(c))
```

```
10
```

11. np.linalg.norm()을 이용하여 벡터의 크기를 구했습니다.

In [11]:
```
#리스트 4-1-(11)
a = np.array([1,3])
print(np.linalg.norm(a))
```

```
3.1622776601683795
```

12. for문을 사용하지 않고 합을 내적으로 계산했습니다.

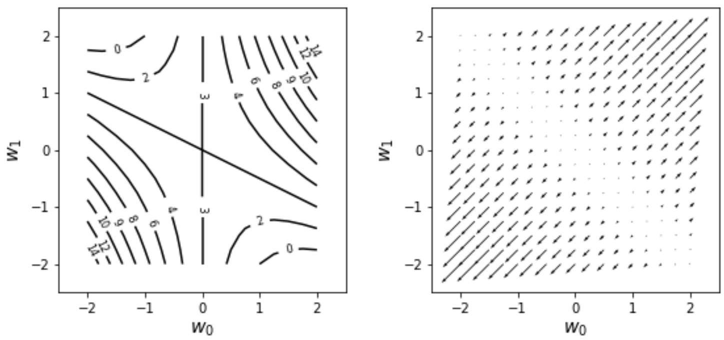
In [12]:
```
#리스트 4-2-(1)
import numpy as np
a = np.ones(1000)
b = np.arange(1,1001)
print(a.dot(b))
```

```
500500.0
```

13. 경사를 그림으로 나타냈습니다.

   오차 함수의 최소점을 구하기 위해 계산한 오차 함수의 경사입니다.

In [13]:
```
#리스트 4-2-(2)
import numpy as np
import matplotlib.pyplot as plt

def f(w0, w1):
    return w0**2 + 2*w0*w1 + 3
def df_dw0(w0, w1):
    return 2*w0 + 2*w1
def df_dw1(w0, w1):
    return 2*w0 + 0*w1

w_range = 2
dw = 0.25

w0 = np.arange(-w_range, w_range + dw, dw)
w1 = np.arange(-w_range, w_range + dw, dw)
wn = w0.shape[0]
ww0, ww1 = np.meshgrid(w0, w1)
ff = np.zeros((len(w0), len(w1)))
dff_dw0 = np.zeros((len(w0), len(w1)))
dff_dw1 = np.zeros((len(w0), len(w1)))
for i0 in range(wn):
    for i1 in range(wn):
        ff[i1, i0] = f(w0[i0], w1[i1])
        dff_dw0[i1, i0] = df_dw0(w0[i0], w1[i1])
        dff_dw1[i1, i0] = df_dw0(w0[i0], w1[i1])

plt.figure(figsize=(9,4))
plt.subplots_adjust(wspace=0.3)
plt.subplot(1, 2, 1)
cont = plt.contour(ww0, ww1, ff, 10, colors = 'k')
cont.clabel(fmt = '%2.0f', fontsize = 8)
plt.xticks(range(-w_range, w_range + 1, 1))
plt.yticks(range(-w_range, w_range + 1, 1))
plt.xlim(-w_range - 0.5, w_range + .5)
plt.ylim(-w_range - .5, w_range + .5)
plt.xlabel('$w_0$', fontsize=14)
plt.ylabel('$w_1$', fontsize=14)

plt.subplot(1,2,2)
plt.quiver(ww0,ww1,dff_dw0,dff_dw1)
plt.xlabel('$w_0$', fontsize=14)
plt.ylabel('$w_1$', fontsize=14)
plt.xticks(range(-w_range, w_range + 1, 1))
plt.yticks(range(-w_range, w_range + 1, 1))
plt.xlim(-w_range - 0.5, w_range + .5)
plt.ylim(-w_range - .5, w_range + .5)
plt.show()
```



14. np.array를 사용하여 행렬을 정의했습니다.

In [15]:
```
# 리스트 4-3-(2)
A = np.array([[1,2,3],[4,5,6]])
print(A)
```

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

15. b도 정의했습니다.

In [16]:
```
# 리스트 4-3-(3)
B = np.array([[7,8,9,],[10,11,12]])
print(B)
```

```
[[ 7  8  9]
 [10 11 12]]
```

16. a+b와 a-b를 계산했습니다.

In [17]:
```
# 리스트 4-3-(4)
print(A + B)
print(A - B)
```

```
[[ 8 10 12]
 [14 16 18]]
[[-6 -6 -6]
 [-6 -6 -6]]
```

17. 행렬에 스칼라 배를 곱했습니다.

In [18]:
```
# 리스트 4-3-(5)
A = np.array([[1,2,3],[4,5,6]])
print(2 * A)
```

```
[[ 2  4  6]
 [ 8 10 12]]
```

18. A와 B의 내적 계산을 했습니다.

In [19]:
```
# 리스트 4-3-(6)
A = np.array([1,2,3])
B = np.array([4,5,6])
print(A.dot(B))
```

```
32
```

19. *연산자를 사용하여 요소끼리 곱셈을 수행했습니다.

In [20]:
```
# 리스트 4-3-(7)
A = np.array([1,2,3])
B = np.array([4,5,6])
print(A * B)
```

```
[ 4 10 18]
```

20. 나눗셈도 수행했습니다.

In [21]:
```
# 리스트 4-3-(8)
A = np.array([1,2,3])
B = np.array([4,5,6])
print(A / B)
```

```
[0.25 0.4  0.5 ]
```

21. 행렬의 크기가 동일하지 않은 행렬 곱을 수행했습니다.

In [22]:
```
# 리스트 4-3-(9)
A = np.array([[1,2,3],[-1,-2,-3]])
B = np.array([[4,-4],[5,-5],[6,-6]])
print(A.dot(B))
```

```
[[ 32 -32]
 [-32  32]]
```

22. np.identity(n) 명령을 이용하여 n*n 단위 행렬을 생성했습니다.

In [23]:
```
# 리스트 4-3-(10)
print(np.identity(3))
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

23. 단위 행렬에 행렬을 곱했습니다.

    값은 변하지 않았습니다.

In [24]:
```
# 리스트 4-3-(11)
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
I = np.identity(3)
print(A.dot(I))
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

24. np.linalg.inv(A) 명령을 이용하여 A의 역행렬을 구했습니다.

In [25]:
```
# 리스트 4-3-(12)
A = np.array([[1,2],[3,4]])
invA = np.linalg.inv(A)
print(invA)
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

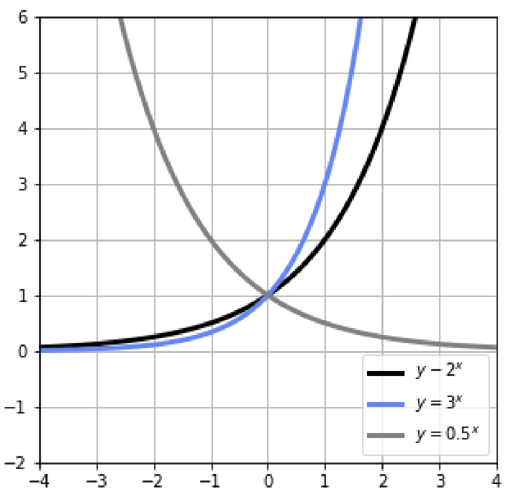
25. 행렬 전치를 수행했습ㄴ디ㅏ.

In [26]:
```
# 리스트 4-3-(13)
A = np.array([[1,2,3],[4,5,6]])
print(A)
print(A.T)
```

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

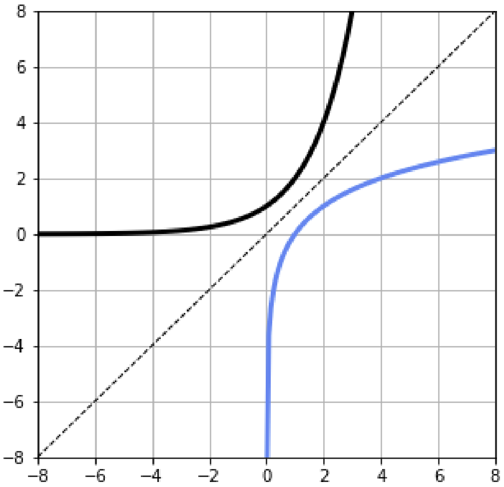
26. a를 밑으로 한 지수 함수를 그래프로 표현했습니다.

In [27]:
```
# 리스트 4-4-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
x = np.linspace(-4, 4, 100)
y = 2**x
y2 = 3**x
y3 = 0.5**x

plt.figure(figsize=(5, 5))
plt.plot(x, y, 'black', linewidth=3, label='$y-2^x$')
plt.plot(x, y2, 'cornflowerblue', linewidth=3, label='$y=3^x$')
plt.plot(x, y3, 'gray', linewidth=3, label='$y=0.5^x$')
plt.ylim(-2,6)
plt.xlim(-4,4)
plt.grid(True)
plt.legend(loc='lower right')
plt.show()
```



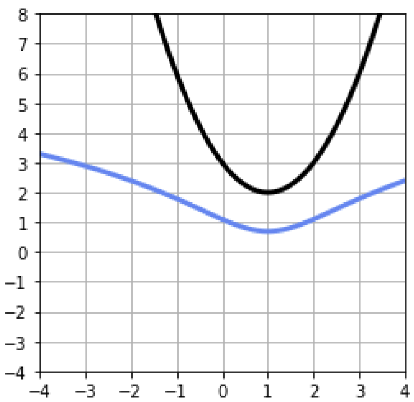
27. a를 밑으로 한 로그 함수를 표현했습니다.

In [28]:
```
# 리스트 4-4-(2)
x = np.linspace(-8, 8, 100)
y = 2**x

x2 = np.linspace(0.001, 8, 100)
y2 = np.log (x2)/np.log(2)

plt.figure(figsize=(5, 5))
plt.plot(x, y, 'black', linewidth=3,)
plt.plot(x2, y2, 'cornflowerblue', linewidth=3)
plt.plot(x, x, 'black', linestyle='--', linewidth=1)
plt.ylim(-8,8)
plt.xlim(-8,8)
plt.grid(True)
plt.show()
```



28. 함수에 로그를 취해도 최소값을 취하는 값은 변하지 않는다는 것을 그래프로 표현했습니다.

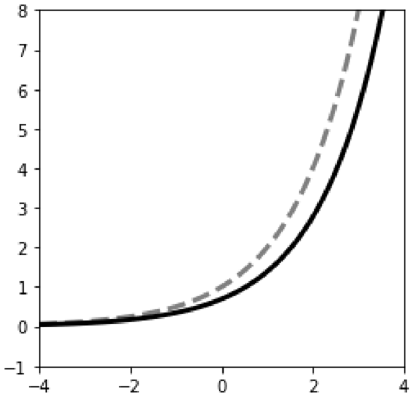
In [29]:
```
# 리스트 4-4-(3)
x = np.linspace(-4, 4, 100)
y = (x - 1)**2 + 2
logy = np.log(y)

plt.figure(figsize=(4, 4))
plt.plot(x, y, 'black', linewidth=3)
plt.plot(x, logy, 'cornflowerblue', linewidth=3)
plt.yticks(range(-4,9,1))
plt.xticks(range(-4,5,1))
plt.ylim(-4,8)
plt.xlim(-4,4)
plt.grid(True)
plt.show()
```



29. 지수 함수의 미분을 수행했습니다.

a = e 인 경우는 미분해도 변하지 않았습니다.

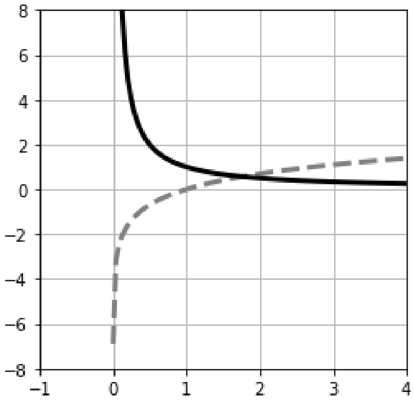
In [41]:
```python
# 리스트 4-4-(4)
x = np.linspace(-4, 4, 100)
a = 2 # 이 값을 여러 가지로 바꿔보자
y = a**x
dy = np.log(a) * y

plt.figure(figsize =(4,4))
plt.plot(x,y,'gray',linestyle='--',linewidth=3)
plt.plot(x,dy,color='black',linewidth=3)
plt.ylim(-1,8)
plt.xlim(-4,4)
plt.show()
```



30. 로그 함수의 미분을 수행했습니다.

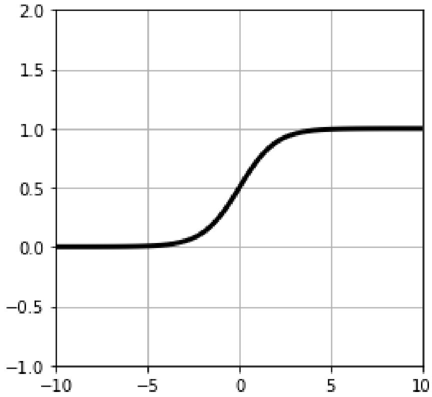
반비례 식이 되었습니다.

In [31]:
```python
# 리스트 4-4-(5)
x = np.linspace(0.001, 4, 100)
y = np.log(x)
dy = 1/x

plt.figure(figsize=(4,4))
plt.plot(x, y, 'gray', linestyle='--',linewidth=3)
plt.plot(x, dy, color='black',linewidth=3)
plt.ylim(-8,8)
plt.xlim(-1,4)
plt.grid(True)
plt.show()
```



31. 시그모이드 함수를 그래프로 나타내었습니다.

In [32]:
```
# 리스트 4-4(6)
x = np.linspace(-10, 10, 100)
y = 1/(1 + np.exp(-x))

plt.figure(figsize=(4,4))
plt.plot(x,y,'black',linewidth=3)

plt.ylim(-1,2)
plt.xlim(-10,10)
plt.grid(True)
plt.show()
```



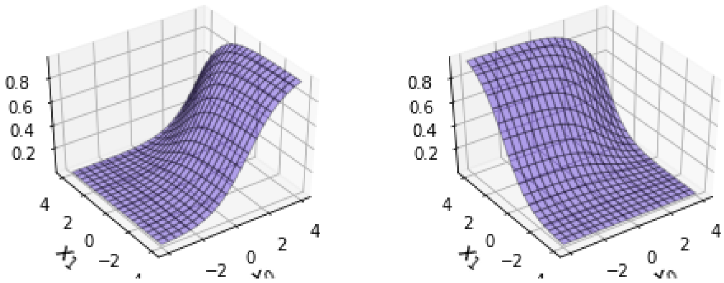
32. 소프트맥스 함수를 만들어 테스트했습니다.

In [33]:
```
# 리스트 4-4(7)
def softmax(x0,x1,x2):
    u = np.exp(x0)+np.exp(x1)+np.exp(x2)
    return np.exp(x0) / u, np.exp(x1) / u, np.exp(x2) / u

# test
y = softmax(2,1,-1)
print(np.round(y,2))
print(np.sum(y))
```

```
[0.71 0.26 0.04]
1.0
```

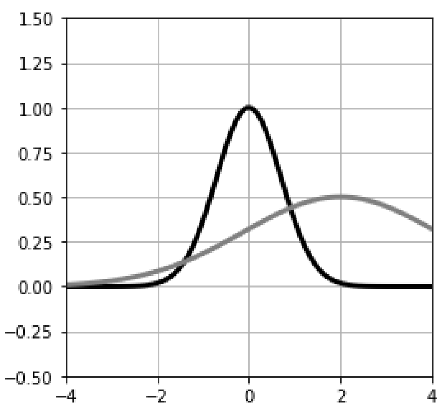
33. 소프트맥스 함수를 그림으로 출력했습니다.

In [34]:
```python
# 리스트 4-4-(8)

from mpl_toolkits.mplot3d import Axes3D

xn = 20
x0 = np.linspace(-4,4,xn)
x1 = np.linspace(-4,4,xn)

y = np.zeros((xn,xn,3))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0, :] = softmax(x0[i0], x1[i1], 1)

xx0, xx1 = np.meshgrid(x0,x1)
plt.figure(figsize=(8,3))
for i in range(2):
    ax = plt.subplot(1,2,i+1,projection='3d')
    ax.plot_surface(xx0, xx1, y[:,:,i],
                    rstride = 1, cstride = 1, alpha = 0.3,
                        color='blue',edgecolor='black')
    ax.set_xlabel('$x_0$',fontsize=14)
    ax.set_ylabel('$x_1$',fontsize=14)
    ax.view_init(40,-125)

plt.show()
```



34. 가우스 함수를 그래프로 나타냈습니다.

In [35]:
```python
# 리스트 4-4-(9)
def gauss(mu,sigma,a):
    return a* np.exp(-(x-mu)**2 / sigma**2)

x = np.linspace(-4,4,100)
plt.figure(figsize =(4,4))
plt.plot(x, gauss(0,1,1), 'black', linewidth=3)
plt.plot(x, gauss(2,3,0.5), 'gray', linewidth=3)
plt.ylim(-.5,1.5)
plt.xlim(-4,4)
plt.grid(True)
plt.show()
```



35. 가우스 함수를 정의했습니다.

In [36]:
```python
# 리스트 4-5-(1)

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
%matplotlib inline

#가우스함수
def gauss(x, mu,sigma):
    N, D = x.shape
    c1 = 1/(2 * np.pi)**(D/2)
    c2 = 1/(np.linalg.det(sigma)**(1/2))
    inv_sigma = np.linalg.inv(sigma)
    c3 = x - mu
    c4 = np.dot(c3, inv_sigma)
    c5 = np.zeros(N)
    for d in range(D):
        c5 = c5 + c4[:, d] * c3[:, d]
    p = c1 * c2 * np.exp(-c5/2)
    return p
```

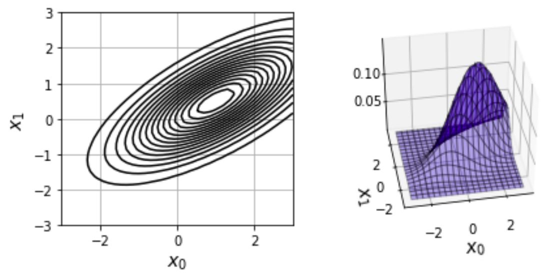
36. 가우스 함수에 수치를 대입하여 테스트했습니다.

In [37]:
```python
# 리스트 4-5-(2)
x = np.array([[1,2],[2,1],[3,4]])
mu = np.array([1,2])
sigma = np.array([[1,0],[0,1]])
print(gauss(x, mu, sigma))
```

```
[0.15915494 0.05854983 0.00291502]
```

37. 가우스 함수를 등고선과 3D로 표시했습니다.

In [38]:
```
# 리스트 4-5-(3)
X_range0 = [-3,3]
X_range1 = [-3,3]

#등고선표시-----------------
def show_contour_gauss(mu, sig):
    xn = 40
    x0 = np.linspace(X_range0[0], X_range0[1], xn)
    x1 = np.linspace(X_range1[0], X_range1[1], xn)
    xx0, xx1 = np.meshgrid(x0, x1)
    x = np.c_[np.reshape(xx0, xn * xn,order='F'), np.reshape(xx1, xn * xn, order='F')]
    f = gauss(x, mu, sig)
    f = f.reshape(xn, xn)
    f = f.T
    cont = plt.contour(xx0, xx1, f, 15, colors='k')
    plt.grid(True)
# 3D 표시-----------------
def show3d_gauss(ax,mu,sig):
    xn = 40
    x0= np.linspace(X_range0[0], X_range0[1], xn)
    x1= np.linspace(X_range1[0], X_range1[1], xn)
    xx0, xx1 = np.meshgrid(x0, x1)
    x = np.c_[np.reshape(xx0, xn * xn,order='F'), np.reshape(xx1, xn * xn, order='F')]
    f = gauss(x, mu, sig)
    f = f.reshape(xn, xn)
    f = f.T
    ax.plot_surface(xx0, xx1, f,
                    rstride=2, cstride=2, alpha=0.3,
                    color='blue', edgecolor='black')
#메인-----------------
mu = np.array([1,0.5])
sigma = np.array([[2,1],[1,1]])
Fig = plt.figure(1, figsize=(7,3))
Fig.add_subplot(1,2,1)
show_contour_gauss(mu,sigma)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.xlabel('$x_0$',fontsize = 14)
plt.ylabel('$x_1$',fontsize = 14)
Ax = Fig.add_subplot(1,2,2,projection='3d')
show3d_gauss(Ax,mu,sigma)
Ax.set_zticks([0.05, 0.10])
Ax.set_xlabel('$x_0$',fontsize=14)
Ax.set_ylabel('$x_1$',fontsize=14)
Ax.view_init(40,-100)
plt.show()
```



수행 소감

이해가 어려웠던 가우스 함수, 소프트맥스 함수 등등을 그래프나 등고선같은 그림으로, 직관적으로 확인할 수 있어서

조금이나마 이해하기 편했던 것 같습니다. 제 지식으로는 난이도가 있는 부분이여서 조금 더 노력하도록 하겠습니다.

긴 글 읽어 주셔서 감사합니다.

이범석

국민대학교 소프트웨어학부, 20171664

Mobile 010-6401-6042

qpwoeiru6486@gmail.com

ijkoo16@kookmin.ac.kr