목차

5. 지도 학습(선형 모델)

5.1 1차원 입력 모델

5.2 1차원 입력 모델 해석해

5.3 2차원 입력 모델

5.4 선형 기저함수 모델

6. 지도 학습(분류 모델)

6.1 1차원 입력 모델

6.2 2차원 입력 모델

7. 딥런닝

7.1 딥런닝

7.2 오차 역전파법

7.3 keras를 이용한 실습

8. 딥런닝 실습(숫자 인식)

8.1 3층 구조

8.2 다층 구조

9. 쉴드 보드 실습

9.1 BH1750 조도 센서

9.2 CDS 조도 센서

9.3 remote control(2분류)

9.4 remote control(3분류)

1 개요

1.1 머신런닝이란

머신러닝은 data로부터 법칙성을 추출하는 통계적 방법.

법칙을 추출하여 향후 발생할 사건에 대한 예측, 분류하는 다양한 모델(알고리즘)을 만들어 냄.

응용예로는, 손글씨 문자 식별, 물체 식별, 음성인식, 주식예측, 질병 진단등....

딥런닝은 머신런닝의 일부로, 뇌 신경세포의 동작을 참고한 신경망 모델의 형태.

머신런닝 모델을 정리한 라이브러리(텐서플로, 케라스 등등)가 머신런닝을 쉽게 구현 가능.

이러한 라이브러리를 쉽게 사용하기 위해서는, 기본 원리와 이론을 알아야 응용이 가능.

이론을 바탕으로 직면한 문제에 대해 적절한 모델을 선택 적용할수 있음.

머신런닝에서 가장 중요한 일은, 목적에 맞는 모델을 선정하여 해결하는것임.

2.2 학습 방법

가장 간단한 모델에 대해, 수학적 해석방법을 이해하고, 차츰 확장 시켜 나가는것.

차원수가 늘어 나더라도, 기본을 확장하는것임으로 시간이 더 걸릴뿐.

수학적 해석이 이해되면, 프로그램으로 구현해 보기.

프로그램으로 구현해보면, 수식의 이해에 도움이 되고, 변화를 주어가며 추이 확인 가능.

수치와 함수를 그래프로 그려보기.

수학적 이해, 프로그램 이해가 그리 간단치 않고 범위가 넓어, 반드시 필요한 항목만 정리.

2.3 시작하기

본 sd card에는 인공지능을 구현하기 위한 Tool들이 탑재되어있음

주요 Tool

1. python3
2. jupyter-notebook and conda
3. 텐서플로 and 케라스

파워를 입력한후, terminal을 뛰워서,

./exeJupyter type하여 실행한후,

browser를 클릭하여, Jupyter Notebook icon을 click하면,

/home/pi directory가 나타나고,

/Ai_Book 디렉토리를 선택하면 본 교재에 대한 해설과 jupyter에 의해 실행될 내용이 포함됨.

외부 컴퓨터에서 라즈베리파이 **jupyter-notebook** 연결

어느정도 라즈베리파이와 접속을 많이 해본사람을 대상으로 하는 설명이지만,

라즈베리파이와 접속할려는 컴퓨터를 같은 공유기로 연결한후,

라즈베리파이 ip를 확인한후, 접속 컴퓨터의 browser에서

라즈베리파이 ip:8888을 입력하면, 라즈베리파이의 jupyter notebook에 접속이 된다.

그상태에서 위의 './exeJupyter'를 사용하여 jupyter notebook을 실행했을겨우 나타나는 화면에,

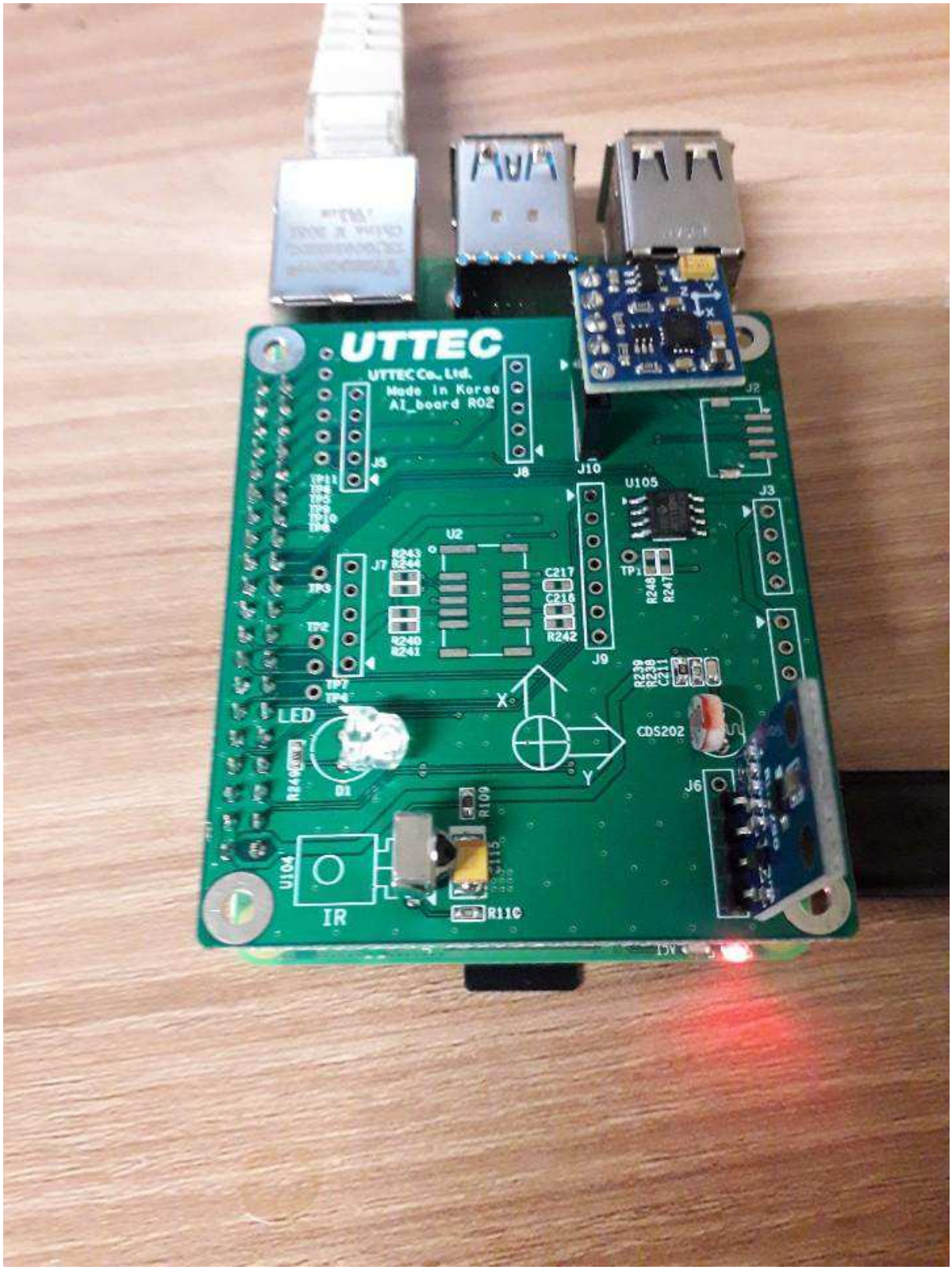
The Jupyter Notebook is running at: http://192.168.0.17:8888/?token=4aaf5f14fa5d7c93bb0c49c16ae3d0af5d106ea465a2e771

과 같은 내용일 보일텐데, token=의 이하부분을 copy하여 password부분에 첨가하여 실행하면 된다.

혹시 추가로 내용일 필요한 사람은 아래의 사이트를 참고하기 바란다.

https://blog.lukael.kr/rajeuberipaie-jupyter/

라즈베리파이에 장착된 **AI Shield**


UTTEC
UTTEC Co., Ltd.
Made in Korea
AI_board R02
J5
J8
J10
U105
J2
J3
U2
J7
C217
C216
R242
J9
TP1
R248
R247
R239
R238
C211
LED
X
Y
CDS202
J6
D1
R109
U104
IR
R110

In [ ]:

2 라즈베리파이

2.1 라즈베리파이 탄생:

영국의 라즈베리파이 재단이 기초 컴퓨터 과학 교육 증진및 취미활동을 위해 만든 ARM기반의 컴퓨터.

여러가지 모델이 있으나, 본 자료에서는 raspberry pi 4를 기준으로 진행한다.

특히 RAM의 크기에 따라 가격 차이가 많이 발생하지만, 가장 낮은 2G에서도 문제가 없음으로,

raspberry pi 4 Model이면 문제가 없을것으로 생각 한다.

이외의 모델에서는 고속의 계산이 필요한 부분에서 문제가 발생하거나, 속도가 매우 느릴수 있다.

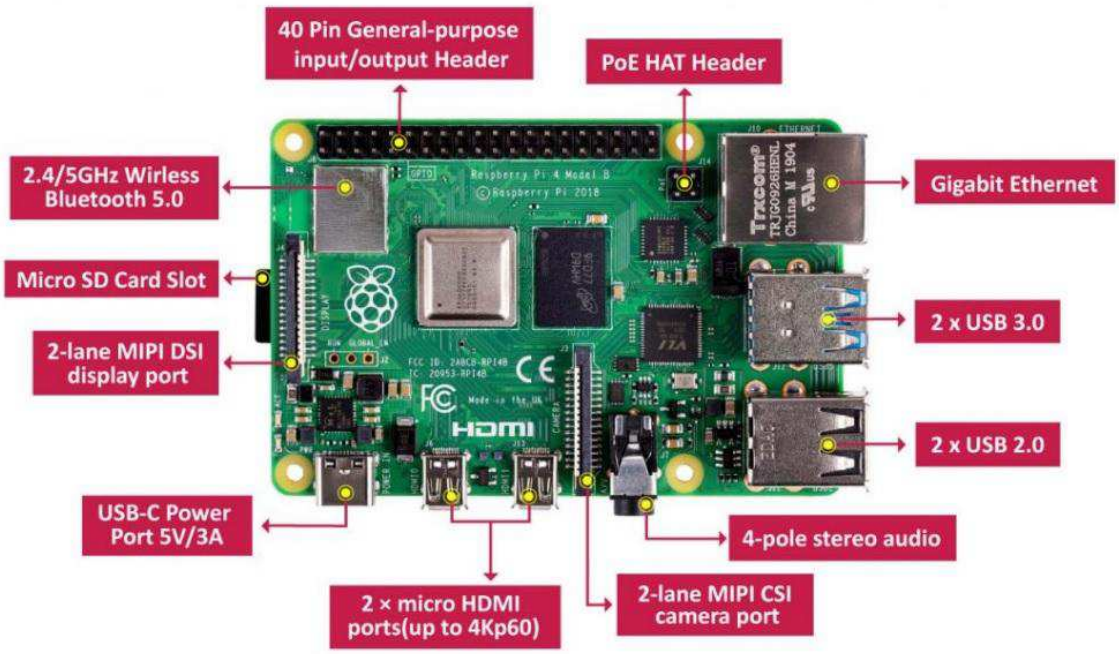
raspberry pi 3 Model이하는 7.2 오차 역전파법에서부터는 실행되지 않을경우가 있음

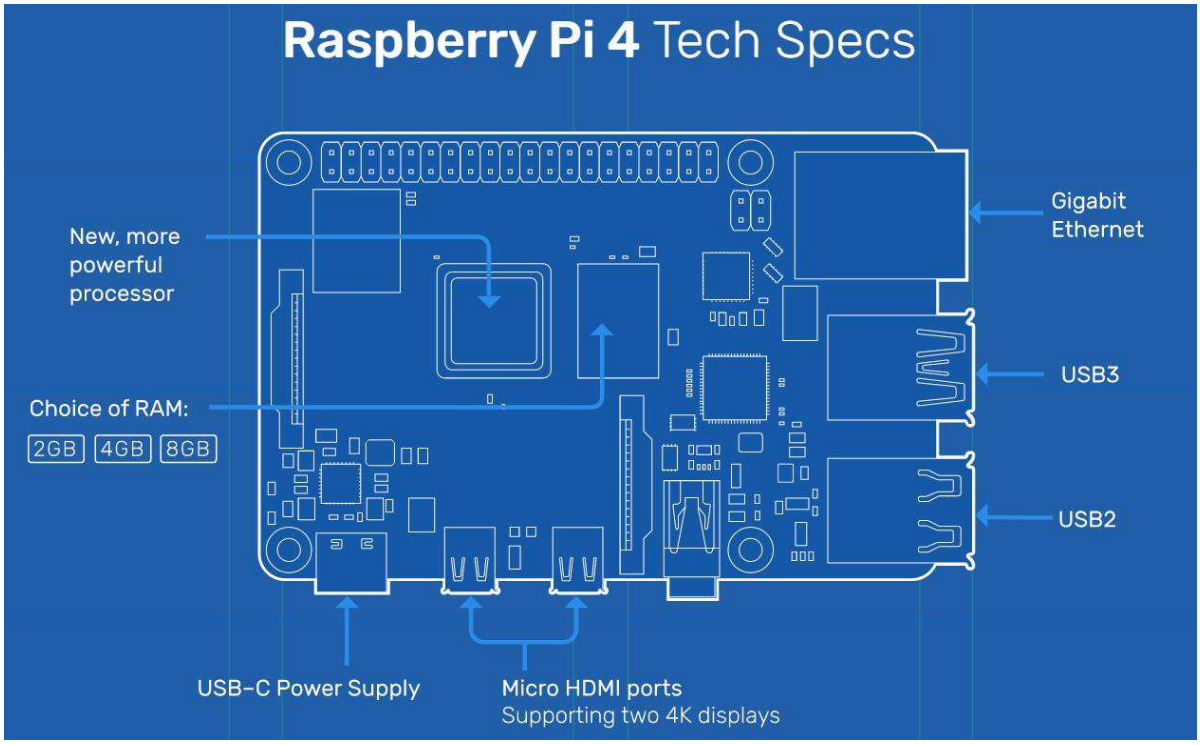
기본적인 외부 기기와의 연결을 위해 필요한 사항은, 아래의 사이트를 찾아가서 확인하기 바란다.

https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up

2.2 라즈베리파이 구조

raspberry pi4

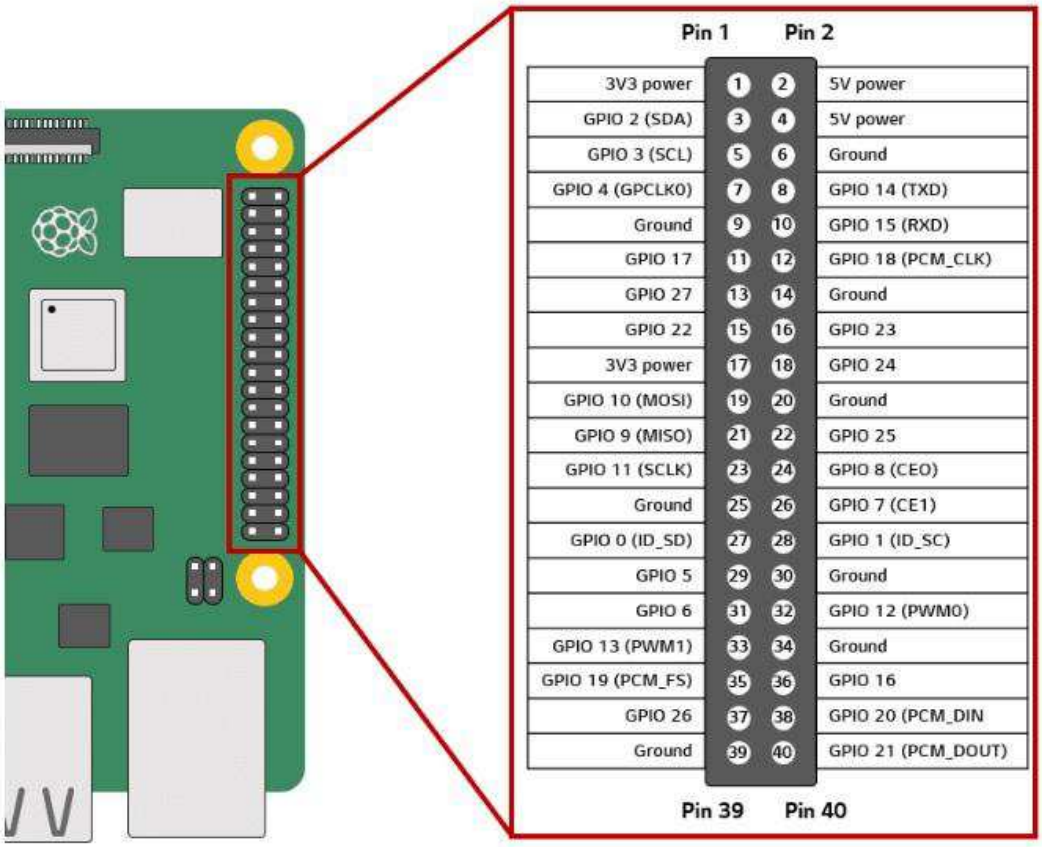




2.3 Raspberry Pi 4 Specifications

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- 2GB, 4GB or 8GB LPDDR4-3200 SDRAM (depending on model)
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
- Gigabit Ethernet
- USB 3.0 ports; 2 USB 2.0 ports.
- Raspberry Pi standard 40 pin GPIO header (fully backwards compatible with previous boards)
- 2 × micro-HDMI ports (up to 4kp60 supported)
- 2-lane MIPI DSI display port
- 2-lane MIPI CSI camera port
- 4-pole stereo audio and composite video port
- H.265 (4kp60 decode), H264 (1080p60 decode, 1080p30 encode)
- OpenGL ES 3.0 graphics
- Micro-SD card slot for loading operating system and data storage
- 5V DC via USB-C connector (minimum 3A*)
- 5V DC via GPIO header (minimum 3A*)
- Power over Ethernet (PoE) enabled (requires separate PoE HAT)
- Operating temperature: 0 – 50 degrees C ambient

2.4 raspberry pi pin map



Pin Map of Raspberry Pi 4

2.5 라즈베리파이 4 용도

일반 컴퓨터

- 초소형 범용 Linux PC

- software로 센서, 모터, 광원 등의 hardware를 직접 제어 가능

- 로봇, 드론, 지능형 센서에 활용 가능

- 일반 pc는 위의 pin map과 같은 GPIO가 없어, 직접 hardware를 제어할수 없음

- OS(operating system)는 Linux계열의 데비안을 탑재

2.6 교재 디렉토리 구조

/home/pi/Ai_Book 이하에 배치 되어있는데

/home/pi/Ai_Book/설명 --> 본 교재의 설명 파일

/home/pi/Ai_Book/실습 --> 본 교재의 실행 파일

/home/pi/Ai_Book/uttecAiExam --> shield board에 적용할수있는 module의 driver

AI에 필요 라이브러리와 운영에 필요한 중요 기능들은,

제공되는 **SD Card**에 **setup**되어있음

3 파이썬

3.1 파이썬과 AI

수치 해석및 기타 수학관련 program을 적용하는데 여러가지 tool이 있을수 있으나,

파이썬의 기본 문법과 여러가지 개발된 라이브러리가 AI에 적용하도록 용이하게 되어있어,

본 과정을 설명하는데 필요한 기본 파이썬 문법을 설명하고자 한다.

자세한 전문적인 내용은 python관련 서적을 참고하기 바람.

여기에서는 본 책자를 이해하는데 필요한 사항만 설명한다.

추가로 본격적인 파이썬에 대해 연구하고자 하는 사람은, 아래의 사이트를 방문해보기 바란다.

https://docs.python.org/ko/3/tutorial/index.ht

아래의 예제들은 기본에 해당됨으로, 초심자들은 직접 입력하여, error도 내보고, 수정도 하면서,

익숙해 지는데 노력해야 한다.

눈으로 보기만 해서는 자기것으로 만들수 없다.

어떤 언어든, 보고 듣고, 따라해보고, 말해보고 하는 과정에서 구사 능력이 달라지는것이다.

파이썬도 언어이기 때문에 똑같은 경우에 속한다.

jupyter notebook을 이용하여 실행하면 쉽게 적용할수 있을것이다.

In [ ]:

3.2 중요 함수 예제

사칙연산(+, -, *, /)

In [1]:
```
print('first:', 1 + 2)

print('second: ', (1 + 2 + 3 - 4 + 2 * 3)/5)
```

```
first: 3
second:  1.6
```

제곱(**)

In [2]:
```
print('제곱: ', 2**3)
```

```
제곱:  8
```

변수(variable)와 자료형

변수명에는 알파벳, 숫자, 밑줄을 사용할수 있다.

대문자 소문자는 구별된다.

변수명 첫자는 숫자를 사용할수 없다.

파이썬 변수형은 int(정수:integer), float(실수: real number), str(문자열: string), bool(참, 거짓),

list(배열), tuple(수정안되는 배열), ndarray(행열)등 기타 변수형이 있으나, 본 책자에서 사용되는

기본 자료형만 설명 하기로 한다.

In [3]:
```
# 자료형을 확인하자 할때는 type 함수를 이용한다.
# ex)
x_int = 100
print('정수형: ', type(x_int))

x_float = 100.1
print('실수형: ', type(x_float))

x_str = 'leaning'
print('문자열: ', type(x_str))
```

```
정수형:  <class 'int'>
실수형:  <class 'float'>
문자열:  <class 'str'>
```

print

test하는데, 결과를 확인하기위해 반드시 필요한 함수가 print로, 자주 사용하는 기능 몇가지를 설명한다.

In [4]:
```
#문자열과 함께 수치를 표시하고 싶은 경우
x = 1/3
print(x)
```

```
print('test 결과1: ', x)
print('test 결과2: '+str(x))
print('test 결과3: {0}이 출력 되었읍니다.'.format(x)) #format을 사용하면 자유로운 위치

y = 4
print('test 결과{0}: {1}이 출력'.format(y, x))
y += 1
print('test 결과{0}: {1:.2f}'.format(y, x))
```

```
0.3333333333333333
test 결과1:  0.3333333333333333
test 결과2: 0.3333333333333333
test 결과3: 0.33333333333333330이 출력 되었읍니다.
test 결과4: 0.33333333333333330이 출력
test 결과5: 0.33
```

list(배열)

In [5]:
```
x_list = [1, 1, 2, 4, 6]

x_element0 = x_list[0] # list명[요소 번호], 요소 번호는 0번부터 시작됨.

print('x_list: {0}, x_element0:{1}'.format(type(x_list), type(x_element0)))
# x_list는 list, x_element0는 int
```

```
x_list: <class 'list'>, x_element0:<class 'int'>
```

n차원 배열

In [6]:
```
x2_list =[[1,2,3], [4,5,6]]
print('x2_list: ', x2_list)
print('x2_list[1][2]: ', x2_list[1][2]) #모든 list는 행, 열 관계없이 0부터 시작. 1행, 2
print('len(x2_list): ', len(x2_list))
```

```
x2_list:  [[1, 2, 3], [4, 5, 6]]
x2_list[1][2]:  6
len(x2_list):  2
```

range

In [7]:
```
x_range = range(5, 10)
print('x_range: ', x_range)

list_x_range = list(x_range)
print('list_x_range: ', list_x_range)
```

```
x_range:  range(5, 10)
list_x_range:  [5, 6, 7, 8, 9]
```

basic statement: if, for, enumerate

In [8]:
```
x_if = 13
if x_if > 10:
    print('x_if is larger than 10')
else:
    print('x_if is smaller than 11')
```

```
x_if is larger than 10
```

비교 연산자

a == b : a와 b가 같다.

a > b : a가 b보다 크다.

a >= b : a가 b이상 이다.

a < b : a가 b보다 작다.

a <= b : a가 b이하 이다.

a != b : a와 b는 다르다.

In [9]:
```
x_for = list(range(5))
print('now start for statement: ')

for i in x_for:
    print(i, end = ',') #print에서 new line을 적용하지 않고 한줄에 나타내고자 할때 사

print('₩nend of for test.')
```

```
now start for statement:
0,1,2,3,4,
end of for test.
```

In [10]:
```
e_num = [2, 4, 6, 8, 10]

for i, n in enumerate(e_num):
    e_num[i] = n * 2 #i는 각 element 순번을, n에는 각 element의 값을 대입하게 된다.

print('final e_num: ', e_num)
```

```
final e_num:  [4, 8, 12, 16, 20]
```

vector: 배열(list)로는 vector관련 연산을 구현할수 없다.

따라서 vector에 관련된 연산을 위해서, 파이썬의 수학 연산 라이브러리인 numpy를 간단히 살펴본다.

간단한 예로 vector 덧셈을 생각해 보자

In [11]:
```
x1_list = [1, 2]
x2_list = [3, 4] #인 경우

#x1_list과 x2_list가 vector로 연산 된다면, [4, 6]의 결과가 나와야 한다.
x12_list = x1_list + x2_list
print('x12_list: ', x12_list)
```

```
x12_list:  [1, 2, 3, 4]
```

그러나 결과는 **[4, 6]**이 아닌, **[1, 2, 3, 4]**로 배열을 합친 결과가 나왔다.

numpy 라이브러리를 이용하여 실행해 보자

In [12]:
```
import numpy as np
x1_vector = np.array([1, 2])
x2_vector = np.array([3, 4])
print('x1_vector: {0}, x2_vector: {1}'.format(x1_vector, x2_vector))
```

```
x1_vector: [1 2], x2_vector: [3 4]
```

In [13]:
```
### list와의 차이는 element사이에 ','가 없다는것이다.
x12_vector = x1_vector + x2_vector
print('x12_vector: ', x12_vector)
```

```
x12_vector:  [4 6]
```

원하는 결과가 출력되었다. list와 np.array는 잘 구분하여 사용하도록 해야한다.

In [14]:
```
print(np.arange(10)) #자주 사용되는, 연속된 정수 벡터 생성

print(np.arange(5, 10))
```

```
[0 1 2 3 4 5 6 7 8 9]
[5 6 7 8 9]
```

In [15]:
```
x23 = np.array([np.arange(1, 4), np.arange(4, 7)]) #2행 3열 행열
print('x23: ')
print(x23)
print('행열: ', x23.shape)
```

```
x23:
[[1 2 3]
 [4 5 6]]
행열:  (2, 3)
```

In [18]:
```
print(np.zeros(10)) #10개의 '0' element를 갖는 1행 행열 생성
print(np.ones(10)) #10개의 '1' element를 갖는 1행 행열 생성
```

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

In [19]:
```
a = np.random.rand(5)
print(a)

b = np.random.rand(2, 3)
print(b)

xRand = np.random.rand(2, 3) #6개의 0.0 ~ 1.0사이의 임의의 실수 행열을 만드는 방법
print('xRand: ', xRand)
```

```
[0.80906796 0.8086976  0.61972119 0.93062568 0.11191235]
[[0.43368203 0.38059674 0.92639269]
 [0.05318904 0.94007202 0.29019642]]
xRand:  [[0.43623642 0.45638802 0.5817264 ]
 [0.47800764 0.66825224 0.18479564]]
```

vector 연산:(+, -, *, /)

In [20]:
```
import numpy as np
```

```
x_ar = np.array([np.arange(1, 6), np.arange(10, 15)])
print(x_ar)

y_ar = np.array([np.arange(5, 10), np.arange(1, 6)])
print(y_ar)
```

```
[[ 1  2  3  4  5]
 [10 11 12 13 14]]
[[5 6 7 8 9]
 [1 2 3 4 5]]
```

In [21]:
```
print('x_ar + y_ar: ', x_ar + y_ar)
```

```
x_ar + y_ar:  [[ 6  8 10 12 14]
 [11 13 15 17 19]]
```

vector연산과 배열(list: 우리가 일상 접하는 배열 연산)연산과는 차이가 있다.

vector(행열) 곱:

In [22]:
```
v_vector = np.array([[1, 2, 3], [4, 5, 6]])
w_vector = np.array([[1, 1], [2, 2], [3, 3]])

print('v_vector x w_vector: ', v_vector.dot(w_vector))
```

```
v_vector x w_vector:  [[14 14]
 [32 32]]
```

vetor곱은 순서와 행열의 조합이 규칙에 맞게 정리 되어야한다.

표현으로는 $v_vector \cdot w_vector$로 순위와 관계가 없을것 같지만.

In [23]:
```
print('w_vector x x_vector: ', w_vector.dot(v_vector))
```

```
w_vector x x_vector:  [[ 5  7  9]
 [10 14 18]
 [15 21 27]]
```

결과는 전혀다른 형태를 나타내게 된다. 조심해야 한다.

좀더 자세한 vector계산은 분야에 대해서 찾아보기 바란다.

또한 배열의 결과는 정방향(2x2, 3x3....)의 형태가 되도록 해야한다.

위의 예 에서는, 첫번째는 (2,3)x(3,2) = (2,2)의 배열 형태가 되고, 두번째는 (3,2)x(2,3)=(3,3)의 배열 결과가 된다.

따라서 vector의 계산은 사전에 입력되는 data구조를 잘 살펴야 한다.

Slicing: 배열 또는 vector, 행열의 일부분 또는 전부를 쉽게 표현하는 방법(python)

이 부분도 본 책자에서, 응용부분에 자주 사용됨으로, 기능은 정확히 이해할 필요가 있다.

In [24]:
```
#ex) for slicing
x_slice = np.arange(10)
```

```
print(x_slice)
print(x_slice[:5]) #처음부터 5이하까질
print(x_slice[5:]) #5부터 마지막까지
print(x_slice[2:6]) #2부터 6이하(5)까지
print(x_slice[0:8:3]) #0부터 7까지 3배수
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4]
[5 6 7 8 9]
[2 3 4 5]
[0 3 6]
```

함수(function): library를 사용하지 않고, 자신의 편리에 의해 만들어 사용.

함수부분은 중요하지만, 자체적으로 공부하여 정리 하여야 한다. 자신이 만들어야 하기 때문에....

인수와 변수는, 정해진 법칙이 있으나, 통상의 프로그램 언어와 차이가 없음

이부분도 아래의 사이트 참고

https://docs.python.org/ko/3/tutorial/index.html

파일 저장(file save):

파이썬 자체적으로 data file저장하기위한 함수가 많이 있음

jupyter에 의해, 산출된 data를 저장하고 읽기 위해서, 이부분은 기억해서 사용하는것이 좋겠다.

jupyter를 사용하지 않을 경우에는, 자체적으로 python에서 지원하는 파일 관련 함수를 사용하는것을 권장

본 책자는, AI의 기초를 배우는 과정에서 필요한 것으로, 향후 더 많은 응용을 위해서는,

일반화되는 파일저장 방법이나 실행방법을 연습하여, 자기가 자유롭게 활용할수 있기를 바란다.

아래 부분은, vector에 의해 생성된 data나, 입력할 vector data를 기준으로 하고 있어서,

numpy에서 제공하는 파일 저장과 읽기만을 설명한다.

In [25]:
```
d_save = np.random.randn(5)
print('d_save: ', d_save)
np.save('d_saveFile.npy', d_save)

d_read = [] # empty list
d_read = np.load('d_saveFile.npy')
print('readData: ', d_read)
```

```
d_save:  [-0.57603154 -2.79764259 -0.34028777 -0.80708613 -0.84906014]
readData:  [-0.57603154 -2.79764259 -0.34028777 -0.80708613 -0.84906014]
```

여러가지 변수를 함께 저장하고, return할수있는 함수: numpy.savez

In [26]:
```
d1 = np.array([1,2,3,4])
d1Name = 'testFile'
dataLength = len(d1)
```

```
print('array: {0}, name: {1}, length: {2}'.format(d1, d1Name, dataLength))
np.savez('testFile.npz', d1 = d1, d1Name = d1Name, dataLength = dataLength)


readFile = np.load('testFile.npz')
print('read array: {0}, name: {1}, length: {2}'.format(readFile['d1'], readFile['d1Nar
```

```
array: [1 2 3 4], name: testFile, length: 4
read array: [1 2 3 4], name: testFile, length: 4
```

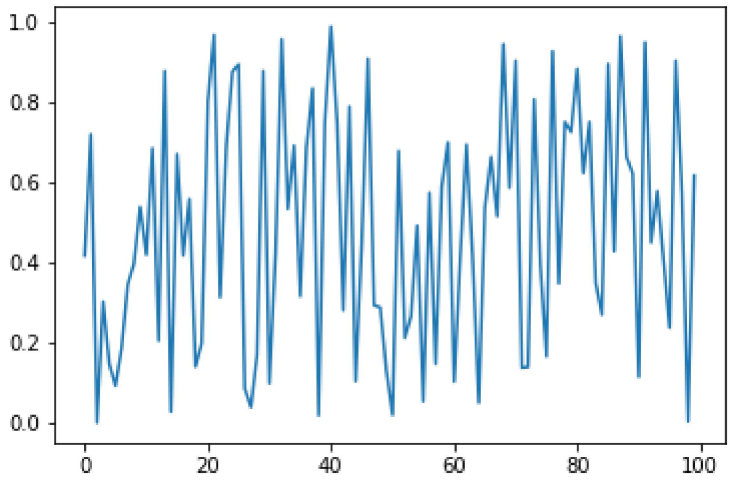
3.3 graph

data를 시각화하고, 경향을 파악하는데는 그래프를 그려보는것이 필요.

가장 간단한, 임의의 data를 display하는것과, 함수에 대한 형태를 display하는것으로 설명을 한다.

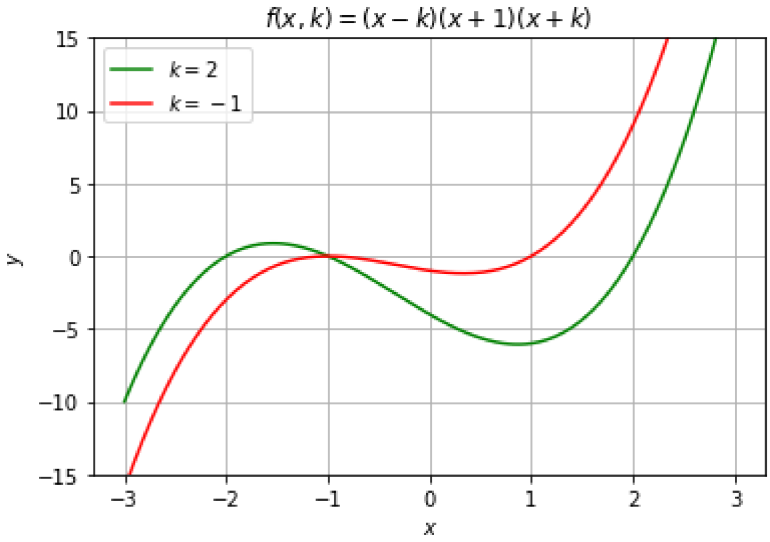
In [1]:
```
# 임의의 data에 대한 display

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(1)
x = np.arange(100) # 0~100 까지
y = np.random.rand(100) # 100개의 0 ~ 1.0사이의 random 실수

plt.plot(x, y) # x, y의 data 등록
plt.show
```

Out[1]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



In [2]:
```
#임의의 함수에 대한 display: f(x) = (x-1)(x+1)(x+3)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def f(x, k):
    return (x-k)*(x+1)*(x+k)

x = np.linspace(-3, 3, 200) # -3 ~ 3사이를 200단계로 나누어

plt.plot(x, f(x, 2), color = 'g', label='$k=2$')
plt.plot(x, f(x, -1), color = 'red', label='$k=-1$')
#color: r=빨강, b=파랑, g=녹색, c=시안, m=마젠타, y=노랑, k=검정, w=white
plt.legend(loc = 'upper left')
plt.ylim(-15, 15)
plt.title('$f(x, k) = (x-k)(x+1)(x+k)$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.show
```

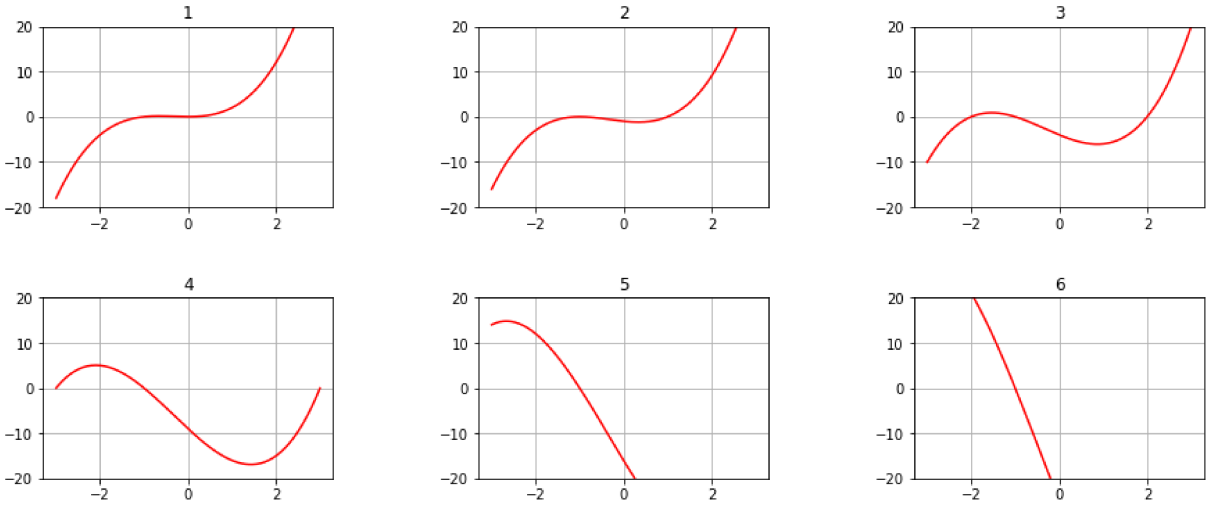
Out[2]: <function matplotlib.pyplot.show(*args, **kw)>



In [3]:

```
# 여러개의 그래프 그리기
plt.figure(figsize=(15, 6)) #각 그래프 size
plt.subplots_adjust(wspace=0.5, hspace=0.5) #각 그래프간 간격 설정
for i in range(6):
    plt.subplot(2, 3, i + 1) #각 그래프의 위치
    plt.title(i + 1) #각 그래프의 제목
    plt.plot(x, f(x, i), 'r') #각 그래프의 data setting
    plt.ylim(-20, 20) #y축 한계
    plt.grid(True)
plt.show # 그리기
```

Out[3]: <function matplotlib.pyplot.show(*args, **kw)>

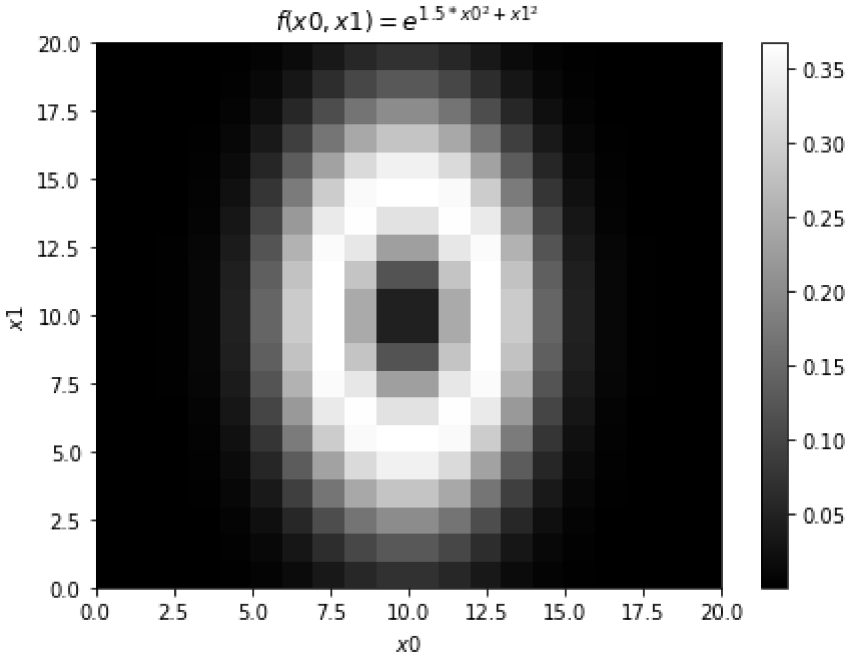


In [4]:

```
# 3차원 그리기
import numpy as np
import matplotlib.pyplot as plt

def f1(x0, x1):
    r = 1.5*x0**2 + x1**2
    return r*np.exp(-r)

xn = 20
x0 = np.linspace(-3, 3, xn) #x0범위: -3 ~ 3을 20개로 나눔
x1 = np.linspace(-2, 2, xn) #x1범위: -2 ~ 2를 20개로 나눔
y = np.zeros((len(x0), len(x1)))# y 배열 '0'으로 초기화
```

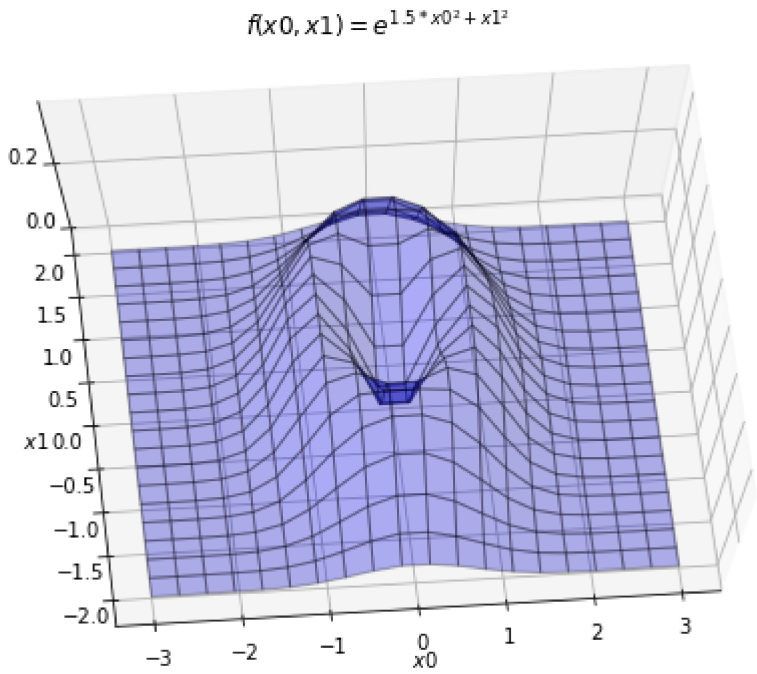
```
for i0 in range(xn): #y배열에 함수 결과값 설정
    for i1 in range(xn):
        y[i1, i0] = f1(x0[i0], x1[i1])

plt.figure(figsize = (7, 5))
plt.gray()
plt.pcolor(y)
plt.colorbar()

plt.title('$f(x0, x1) = e^{1.5*x0^2+x1^2}$')
plt.xlabel('$x0$')
plt.ylabel('$x1$')

plt.show
```

Out[4]: <function matplotlib.pyplot.show(*args, **kw)>



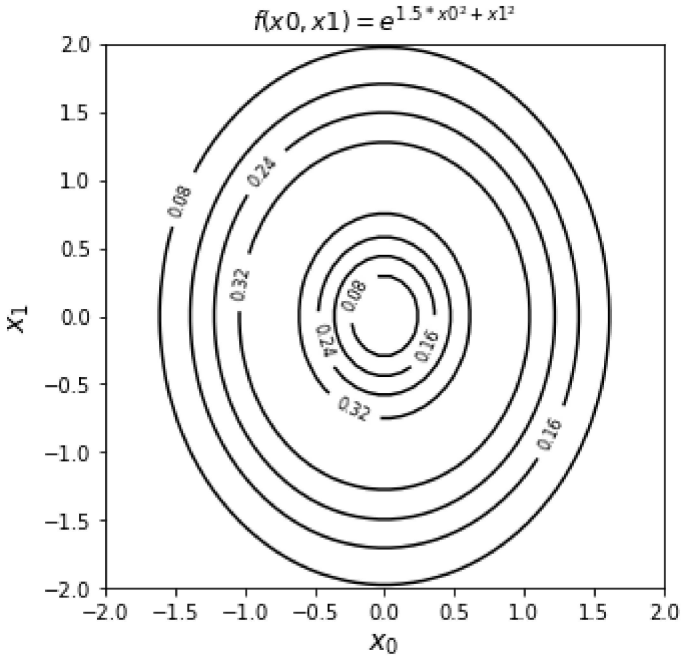
In [5]:

```
# 3차원 그리기

from mpl_toolkits.mplot3d import Axes3D

xx0, xx1 = np.meshgrid(x0, x1)

plt.figure(figsize = (10, 7))
ax = plt.subplot(1, 1, 1, projection = '3d')
ax.plot_surface(xx0, xx1, y, rstride = 1, cstride = 1, alpha = 0.3,
                color = 'blue', edgecolor = 'black')
ax.set_zticks((0, 0.2))
ax.view_init(75, -95)

plt.title('$f(x0, x1) = e^{1.5*x0^2+x1^2}$')
plt.xlabel('$x0$')
plt.ylabel('$x1$')

plt.show()
```



In [6]:

```
#등고선:contour 그리기
xn = 50
x0 = np.linspace(-2, 2, xn)
x1 = np.linspace(-2, 2, xn)

y = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        y[i1, i0] = f1(x0[i0], x1[i1])

xx0, xx1 = np.meshgrid(x0, x1)

plt.figure(1, figsize = (5, 5))

cont = plt.contour(xx0, xx1, y, 5, colors = 'black')
cont.clabel(fmt = '%3.2f', fontsize = 8)

plt.title('$f(x0, x1) = e^{1.5*x0^2+x1^2}$')
plt.xlabel('$x_0$', fontsize = 14)
plt.ylabel('$x_1$', fontsize = 14)
plt.show()
```


$f(x0, x1) = e^{1.5 * x0^2 + x1^2}$
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x_1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_0
0.08
0.24
0.32
0.08
0.24
0.16
0.32
0.16

4 수학

4.1 Vector

가로 vector: $\mathbf{d} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

세로 vector : $\mathbf{d^T} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

전치행렬(transposed matrix): 행과 열을 교환한 상태

AI부분에서 자주 나타나는 행렬 계산 방법임으로 형태와 의미를 명확히 알아두어야 한다.

수학에서 vector를 program으로 계산을 진행하기 위해 필요한것은,

파이썬에서 라이브러리를 사용하면 되는데,

numpy라는 라이브러리를 불러서 사용하면 된다.

In [1]:
```
import numpy as np
a = np.array([1,2])
print('가로 vector: a -->', a)
```

```
가로 vector: a --> [1 2]
```

세로 vector는 어떻게 나타낼까. 간단한 형태로

In [2]:
```
c = np.array([[1,2], [3,4]])
print('c의 형태: ')
print(c)
```

```
c의 형태:
[[1 2]
 [3 4]]
```

이 형태를 기본으로해서, 세로 형태를 나타낼려면

In [3]:
```
d = np.array([[1], [2]])
print(d)
```

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

이 형태가 2x1의 2차원 세로 벡터를 나타낸다.

세로 벡터는 대괄호가 한번 더 있기는 하지만, 의미상으로 동일하다.

그러면 d를 전치하면, a와 같이 되는지 확인해 보자.

In [4]:
```
print(d.T)
```

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

In [5]:

```
print(c.T)
```

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

Vector +(덧셈), -(뺄셈)

$$\mathbf{a} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 2+1 \\ 1+3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\mathbf{a} - \mathbf{b} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 2-1 \\ 1-3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

이것을 파이썬으로 구현해 보자.

In [6]:
```
a = np.array([2,1])
b = np.array([1,3])
print('덧셈: ', a+b)
print('뺄셈: ', a-b)
```

```
덧셈:  [3 4]
뺄셈:  [ 1 -2]
```

항상, 수식을 파이썬으로 검증해보는 습관을 갖도록 하자.

자꾸 연습하면, 추후 수식을 바로바로 파이썬으로 구현할수 있고,

파이썬 프로그램을 수식화 하기에도 수월해 진다.

벡터에 스칼라값을 곱하거나, 나누기, 더하기, 빼기등은 보통의 4칙 연산과 동일하게 취급하면 된다.

$$ex) 2\mathbf{a} = 2x \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2x2 \\ 1x2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

In [7]:
```
print('결과:', 2*a)
```

```
결과: [4 2]
```

vector의 내적(inner product)

$$\mathbf{b} \cdot \mathbf{c} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 2 \end{bmatrix} = 1\ x\ 4 + 3\ x\ 2 = 10$$

vector의 크기

2차원의 경우

$$|a| = \left\| \begin{matrix} a_0 \\ a_1 \end{matrix} \right\| = \sqrt{a_0^2 + a_1^2}$$

3차원의 경우

$$|a| = \left\| \begin{matrix} a_0 \\ a_1 \\ a_2 \end{matrix} \right\| = \sqrt{a_0^2 + a_1^2 + a_2^2}$$

일반화된 식으로는

$$|a| = \left\| \begin{matrix} a_0 \\ \vdots \\ a_{D-1} \end{matrix} \right\| = \sqrt{a_0^2 + \cdots + a_{D-1}^2}$$

예제의 파이썬 계산으로는;

In [8]:
```
a = np.array([1,3])
print(np.linalg.norm(a)) #이부분이 각 vector를 제곱하고, 더하여, 제곱평균을 내는 함수
```

3.1622776601683795

자주 사용되는 기호

합(sum)기호: $\sum$

$$\sum_{n=a}^{b} f(n) = f(a) + f(a+1) + \cdots + f(b)$$

vector(행열)에서 합의 기호 표기

(자주 사용됨으로 의미를 정확히 이해할 필요가 있음)

$$\mathbf{W} \cdot \mathbf{X} = W_0 X_0 + W_1 X_1 + \cdots + W_{D-1} X_{D-1} = \sum_{i=0}^{D-1} W_i X_i$$

$$\mathbf{w} = \left| \begin{matrix} w_0 \\ w_1 \\ \vdots \\ w_{D-1} \end{matrix} \right| \quad \mathbf{x} = \left| \begin{matrix} x_0 \\ x_1 \\ \vdots \\ x_{D-1} \end{matrix} \right|$$

ex) 합을 vector의 내적으로 구하는 예제

(파이썬의 for문과, numpy의 dot함수의 실행 속도 비교)

먼저 for문에 의해 수행되는 시간을 측정해 보자.(1 ~ 10000까지 더하는 기능)

In [9]:
```
import time
testNum = 1000000
sum = 0
start = time.time()
for i in range(1,testNum+1):
    sum += i
print('for result: ', sum)
forDelay = time.time() - start
print('for에 의한 계산시간: ', forDelay)
```

```
for result:  500000500000
for에 의한 계산시간:  0.38047099113464355
```

In [10]:
```
import numpy as np
a = np.ones(testNum)
b = np.arange(1, testNum+1)
start = time.time()
print('result:', a.dot(b))
dotDelay = time.time() - start
print('dot에 의한 계산시간: ', dotDelay)

print('for - dot: ', forDelay - dotDelay)
```

```
result: 500000500000.0
dot에 의한 계산시간:  0.05107617378234863
for - dot:  0.3293948173522949
```

위의 결과를 보면 testNum숫자가 크면 클수록

for에 의해 걸리는 시간보다 numpy의 dot함수에 의한것이 훨씬 빠르다는것을 알수있다.

곱(multiplication)의 기호: $\prod$

$$\prod_{n=a}^{b} f(n) = f(a)xf(a+1)x\cdots xf(b)$$

$$ex) \prod_{n=1}^{5} = 1x2x3x4x5$$

$\sum$ 과 $\prod$ 가중요한이유

빛은 여러 색이 모이면 하얀색쪽으로 이동하고, 색감은 여러 색을 섞으면 검은색쪽으로 변하고,

비빔밥과 같이, 여러가지 재료가 섞이면 맛있는 쪽으로 ?,

수학의 여러가지 함수들은 단순한 함수의 합으로 이루어지는 경우가 많다.

따라서 무엇인가에 수렴해야하는 경우에는 반드시 위의 기호를 만나거나 이해해야할때가 많으니,

반드시 기억해 놓기 바란다.

4.2 미분: $\frac{df(x)}{dx}$

머신 런닝은, 여러 조건에 의해 주어진 함수에 대해, 최소나 최대인 지점을 찾는 과정이 중요하다.

즉, 최소점이나 최대점 지역에서는 기울기가 0에 근사되는 성질이 있는데, 이 지점을 찾는 방법으로,

함수의 기울기를 구하는 방법이 미분이다.

오차 함수를 만들고, 그 오차 함수가 최소가되는 지점을 찾는것이다.

오차 함수에 대해, 미분을 적용하고, 미분에 의한 함수값이 0에 근사되는 지점을 찾고,

그 지점의 오차가 최소 오차점이라고 가정하여 최종결론에 준하는지 평가해 가는것이다.

추상적으로 생각할때, 오답이 최소가 되면, 정답에 가까워 진다고 생각하는것이다.

미분에 대한 확실한 이해 없이는 AI를 이해하는것은 불가능하다.

ex) $f(x) = x^2$ 함수를 x에 대해 미분을 한다고 하자

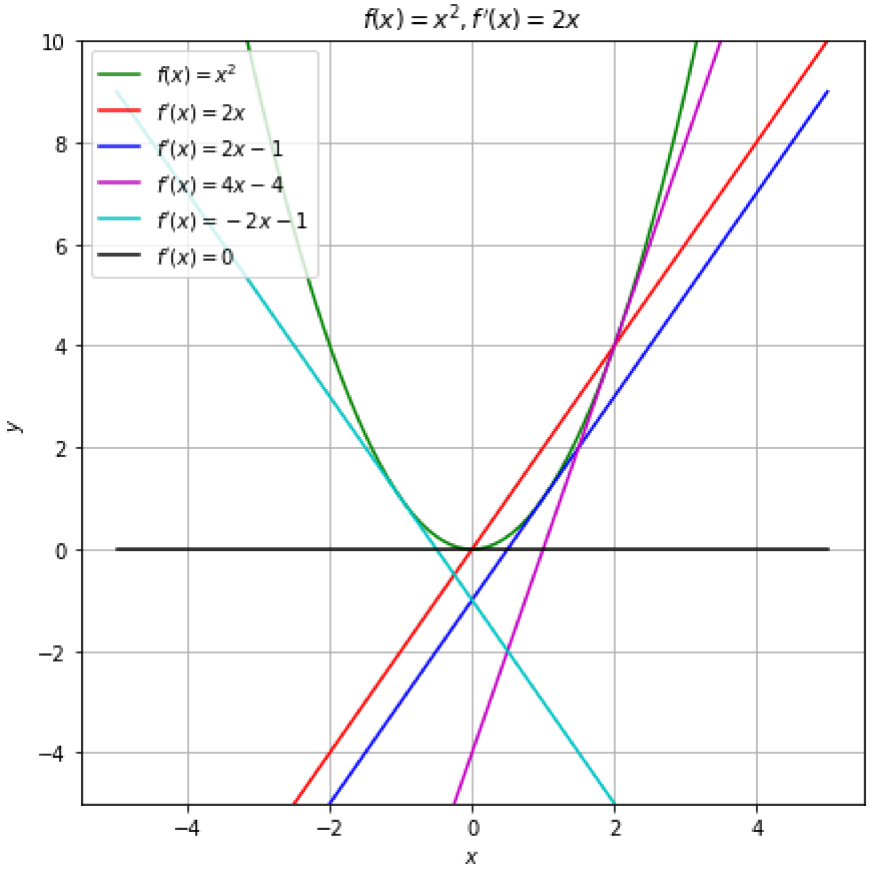
이것을 기호로 나타내면, $\frac{df(w)}{dw}$, $\frac{d}{dw}f(w)$, $f'(w)$로 표현한다.

In [1]:

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

def f(x):
    return x**2

def df(x):
    return 2*x

def df_2x(x):
    return 2*x - 1

def df_m2x(x):
    return -2*x - 1

def df_4x(x):
    return 4*x -4

def df_0x(x):
    return 0.0*x

x = np.linspace(-5, 5, 200) # -3 ~ 3사이를 200단계로 나누어

plt.figure(figsize = (7, 7))

plt.plot(x, f(x), color = 'g', label="$f(x) = x^2$")
plt.plot(x, df(x), color = 'r', label="$f'(x) = 2x$")
plt.plot(x, df_2x(x), color = 'b', label="$f'(x) = 2x -1$")
plt.plot(x, df_4x(x), color = 'm', label="$f'(x) = 4x - 4$")
plt.plot(x, df_m2x(x), color = 'c', label="$f'(x) = -2x - 1$")
plt.plot(x, df_0x(x), color = 'k', label="$f'(x) = 0$")
```

```
#color: r=빨강, b=파랑, g=녹색, c=시안, m=마젠타, y=노랑, k=검정, w=white

plt.legend(loc = 'upper left')
plt.ylim(-5, 10)
plt.title("$f(x) = x^2, f'(x) = 2x $")
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.show
```

Out[1]: <function matplotlib.pyplot.show(*args, **kw)>



x = 2 접점에서의 접점 함수, 기울기는 4, f(x) = 4x - 4

x = 1 접점에서의 접점 함수, 기울기는 2, f(x) = 2x - 1

x = -1 접점에서의 접점 함수, 기울기는 -2, f(x) = -2x - 1

x = 0 접점에서의 접점 함수, 기울기 0, f(x) = 0 --> f(x) = x^2 함수의 최저점 point

n차식의 미분 공식: $\frac{d}{dw}w^n = nw^{n-1}$

목적 변수가 포함되지 않는 함수

$f(x) = a^3 + xb^2 + 2$에 대한 w의 미분

$\frac{d}{dx}f(w) = \frac{d}{dw}(a^3 + xb^2 + 2) = 0$

목적 변수, w가 포함되지 않은 함수는 상수로 취급된다.

중첩된 함수의 미분

$$f(w) = g(w)^2$$

$$g(w) = aw + b$$

$$f(w) = (aw + b)^2 = a^2w^2 + 2abw + b^2$$

(n차식 미분 공식에 의해서**)**

$$\frac{d}{dw} f(w) = 2a^2w + 2ab$$

목적변수가 중첩된 함수의 미분: 연쇄 법칙

$$\frac{df}{dw} = \frac{df}{dg} \cdot \frac{dg}{dw}$$

해석: df/dg부분은 'f를 g로 미분한다', $f(w) = (aw + b)^2 = g(w)^2$를 연쇄 법칙을 이용하여 풀어보자

(n차 미분 공식에 따라**)**

$$\frac{df}{dg} = \frac{d}{dg} g^2 = 2g$$

$$\frac{dg}{dw} = \frac{d}{dw}(aw + b) = a$$

결과:

$$\frac{df}{dw} = \frac{df}{dg} \cdot \frac{dg}{dw} = 2ga = 2(aw + b)a = 2a^2w + 2ab$$

편미분**(partial differentiation):**

다변수 함수의 특정 변수를 제외한 나머지 변수를 상수로 가정하여

해당 변수로 미분하는것

기본형태:

$$f(w_0, w_1) = w_0^2 + 2w_0w_1 + 3$$

이 함수에서 목적 변수 w_0, w_1에 대해,

w_0에 대해 미분을 할때, w_1부분을 상수로 간주

w_1에 대해 미분을 할때, w_0부분을 상수로 간주하여 미분을 수행 한다.

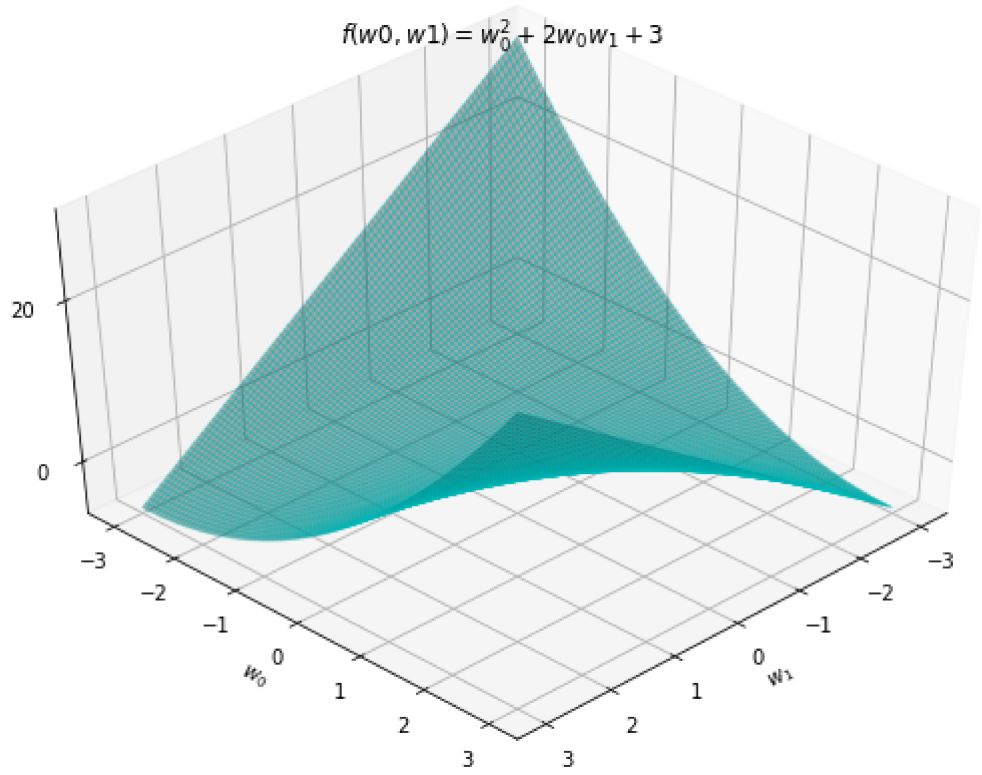
$$\frac{\partial f(w_0, w_1)}{\partial w_0}, \ \frac{\partial}{\partial w_0} f(w_0, w_1), \ f'_{w_0}$$

예) $f(w_0, w_1) = w_0^2 + 2w_0w_1 + 3$ 에 대한 w_0, w_1의 편미분을 수행한다.

$\frac{\partial f}{\partial w0} = 2w_0 + 2w_1, \ \frac{\partial f}{\partial w_1} = 2w_0 \implies n$차미분공시에따라.

In [2]:

```
# 3차원 그리기
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from mpl_toolkits.mplot3d import Axes3D

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

wn = 100
w0 = np.linspace(-3, 3, wn) #w0범위: -3 ~ 3을 100개로 나눔
w1 = np.linspace(-3, 3, wn) #w1범위: -3 ~ 3를 100개로 나눔

y = np.zeros((len(w0), len(w1)))# y 배열 '0'으로 초기화

for i0 in range(wn): #y배열에 함수 결과값 설정
    for i1 in range(wn):
        y[i0, i1] = f(w0[i0], w1[i1])

ww0, ww1 = np.meshgrid(w0, w1)

plt.figure(figsize = (10, 7))
ax = plt.subplot(1, 1, 1, projection = '3d')
ax.plot_surface(ww0, ww1, y, rstride = 1, cstride = 1, alpha = 0.3,
                color = 'k', edgecolor = 'c')
ax.set_zticks((0, 20))
# ax.view_init(75, -95)
ax.view_init(45, 45)

plt.title('$f(w0, w1) = w_0^2+2w_0w_1+3$')
plt.xlabel('$w_1$')
plt.ylabel('$w_0$')

plt.show()
```



위 그림에서, w1=-1의 면을 자르는것은
$f(w0, w1) = w_0^2 + 2w_0w_1 + 3$에서 w_1에 -1을 적용하여

$f(w0, -1) = w_0^2 - 2w_0 + 3$에 대한 미분을구하게되면

$\frac{\partial f}{\partial w_0} |_{w1=-1} = 2w_0 + 2w_1 |_{w1=-1} = 2w_0 - 2$

위 그림에서, w1=1의 면을 자르는것은

$\frac{\partial f}{\partial w_0} |_{w1=1} = 2w_0 + 2w_1 |_{w1=1} = 2w_0 + 2$

$\frac{\partial f}{\partial w_1}$에 대해서 살펴 보면

$\frac{\partial f}{\partial w_1} |_{w0=1} = 2w_0 |_{w0=1} = 2$임으로 상수가 나오는것은,

$w_0 = 1$에서, 기울기가 2인 경사면을 나타남을 보여준다.

$\frac{\partial f}{\partial w_1} |_{w0=-2} = 2w_0 |_{w0=-2} = -4$임으로 상수가 나오는것은,

$w_0 = -2$에서, 기울기가 -4인 경사면을 나타남을 보여준다.

이상을 살펴보변, w_0와 w_1에 대한 편미분은 각각 w_0방향의 기울기, w_1방향의 기울기를 나타낸다.

즉, 편미분에 의해 임의의(w_0, w_1)에서 두 방향의 기울기를 계산할수 있다.

이 두개의 기울기로 벡터형태의 모습을 볼수 있다.

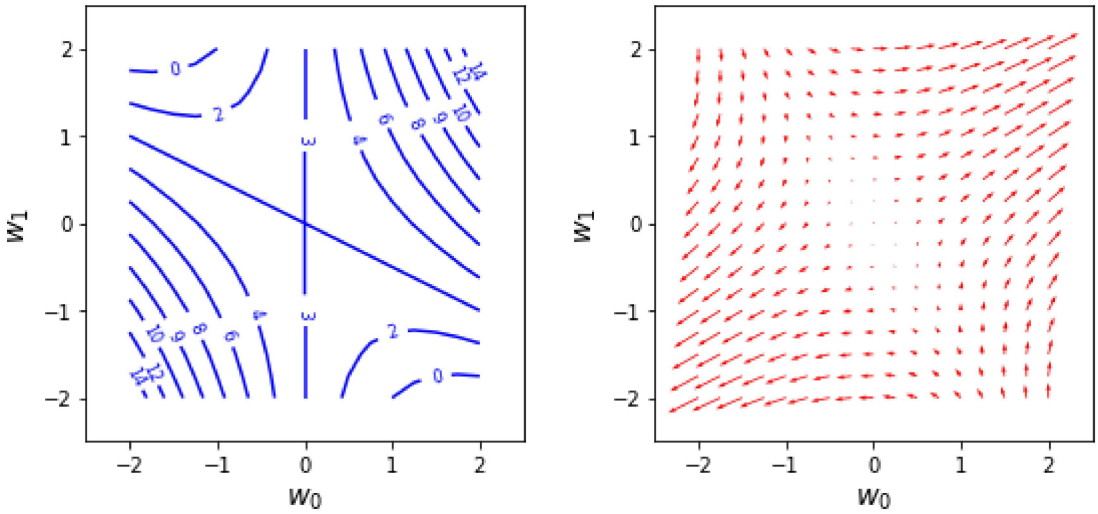
이는 f의 w에 대한 경사(기울기 벡터)로 하고, 경사는 방향과 크기로 나타낼수 있다.

수식으로 표현하면

$$\nabla_w f = \begin{bmatrix} \frac{\partial f}{\partial w_0} \\ \frac{\partial f}{\partial w_1} \end{bmatrix}$$

In [3]:

```
import numpy as np
import matplotlib.pyplot as plt

def f(w0, w1): # f의 정의
    return w0**2 + 2 * w0 * w1 + 3

def df_dw0(w0, w1): # f의 w0에 관한 편미분
    return 2 * w0 + 2 * w1 #w0와 w1모두에 의해 기울기가 영향을 받음

def df_dw1(w0, w1): # f의 w1에 관한 편미분
    return 2 * w0 + 0 * w1 #w0에 의해서만 기울기가 정해짐


w_range = 2
dw = 0.25

w0 = np.arange(-w_range, w_range + dw, dw) #-2 ~ 2까지 0.25간격으로
w1 = np.arange(-w_range, w_range + dw, dw) #-2 ~ 2까지 0.25간격으로
wn = w0.shape[0] # 17 간격
# print('wn: ', wn)
ww0, ww1 = np.meshgrid(w0, w1) # (D)

ff = np.zeros((len(w0), len(w1))) #def f(w0, w1): 함수 결과값 저장소 초기화

dff_dw0 = np.zeros((len(w0), len(w1))) #def df_dw0(w0, w1): 함수 결과값 저장소 초기화
dff_dw1 = np.zeros((len(w0), len(w1))) #def df_dw1(w0, w1): 함수 결과값 저장소 초기화

for i0 in range(wn): #각 함수값 계산및 저장
    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_dw1(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='b') # f의 등고선 표시
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, color='r') # f의 경사 벡터 표시
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()
```



그래프에 대한사항은, google에서 찾아보면 쉽게 찾아볼수 있다.

다변수의 중첩 함수 미분

$$f(g_0(w_0, w_1), g_1(w_0, w_1))$$

이것은 향후 여러 층으로 된 신경망의 학습 규칙을 도출할때 사용 됩니다.

좀더 심층적인 신경망을 적용할때 사용됨으로, 간단히 이해할 필요가 있다.

본 자료는 기초 부분을 다루고 있기때문에, 그다지 자주 나오지는 않지만 이해하고는 있어야 한다.

앞의 내용의 연장에서 설명됨으로, 그다지 어려운 전개는 아니다.

다변수 중첩 편미분의 연쇄 법칙

$$\frac{\partial}{\partial w_0} f(g_0(w_0, w_1), g_1(w_0, w_1)) = \frac{\partial f}{\partial g_0} \cdot \frac{\partial g_0}{\partial w_0} + \frac{\partial f}{\partial g_1} \cdot \frac{\partial g_1}{\partial w_0}$$

이런 형태를 일반화 공식으로 하면

$$\frac{\partial}{\partial w_0} f(g_0(w_0, w_1), g_1(w_0, w_1), \cdots, g_M(w_0, w_1)) = \sum_{m=0}^{M} \frac{\partial f}{\partial g_m} \cdot \frac{\partial g_m}{\partial w_0}$$

예제를 들어 간단히 결과를 내보자.

$f = (g_0 + 2g_1 - 1)^2,\ g_0 = w_0 + 2w_1 + 1,\ g_1 = 2w_0 + 3w_1 - 1$인 경우

$$\frac{\partial f}{\partial g_0} = 2(g_0 + 2g_1 - 1)$$

$$\frac{\partial f}{\partial g_1} = 2(g_0 + 2g_1 - 1) \cdot 2$$

$$\frac{\partial g_0}{\partial w_0} = 1$$

$\frac{\partial g_1}{\partial w_0} = 2$ 임으로, 이들을 대입하여 정리하면,

$\frac{\partial f}{\partial w0} = 2(g_0 + 2g_1 - 1) \cdot 1 + 2(g_0 + 2g_1 - 1) \cdot 2 \cdot 2 = 10g_0 + 20g_1 - 10$로 나타난다.

합과 미분의 교환

$$\frac{\partial}{\partial w} \sum_{n=1}^{3} nw^2 = \sum_{n=1}^{3} \frac{\partial}{\partial w} nw^2$$

앞 항은 합쳐진 함수들에 대해 미분을 수행하는것이고,

뒤 항은 각 함수들을 미분하여 미분된 항들의 합을 수행하는것인데,

이는 동일하다는 의미입니다.

이를 일반화하면

$$\frac{\partial}{\partial w} \sum_{n} f_n(w) = \sum_{n} \frac{\partial}{\partial w} f_n(w)$$

즉, 미분과 합의 기호는 순서를 바꿀수 있다.

앞의 항보다는 뒤 항이 자주 사용되는데,

이유는 미분을 먼저 계산하면 없어지는 항이 늘어나서 계산이 간편해 지기 때문이다.

In [ ]:

4.3 지수 함수와 로그함수

왜 지수 함수와 로그 함수를 사용해야 하는가?

자연 현상을 나타내는데, 지수함수와 로그함수형태가 적절.

미분을 수행하는데, 간단한 형태로 정리됨.

지수

지수의 정의:

a > 0, n을 양의 정수라면,

$a^0 = 1$

$a^{-n} = \frac{1}{a^n}$

$a^{\frac{1}{n}} = \sqrt[n]{a}$

지수의 공식:

a > 0, b > 0, m, n을 실수라고 하면

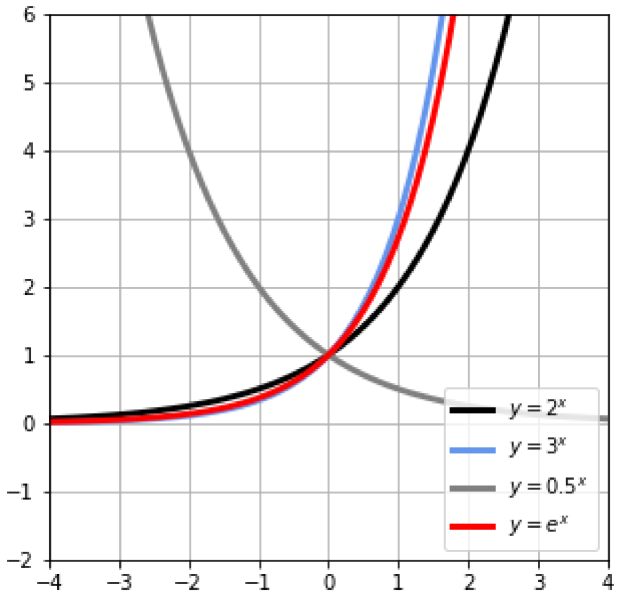
$a^n a^m = a^{n+m}$

$\frac{a^n}{a^m} = a^{n-m}$

$(a^n)^m = a^{nm}$

$(ab)^n = a^n b^n$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

x = np.linspace(-4, 4, 100)

y = 2**x #a = 2
y2 = 3**x #a = 3
y3 = 0.5**x
y4 = np.exp(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.plot(x, y4, 'red', linewidth=3, label='$y=e^x$')
plt.ylim(-2, 6)
plt.xlim(-4, 4)
plt.grid(True)
plt.legend(loc='lower right')
plt.show()
```



로그

$y = log_a x$

로그의 정의:

a를 1이 아닌 양의 실수라고 할때

$log_a a = 1$

$log_a 1 = 0$

로그의 공식:

a, b를 1이 아닌 양의 실수라면

$log_a xy = log_a x + log_a y$

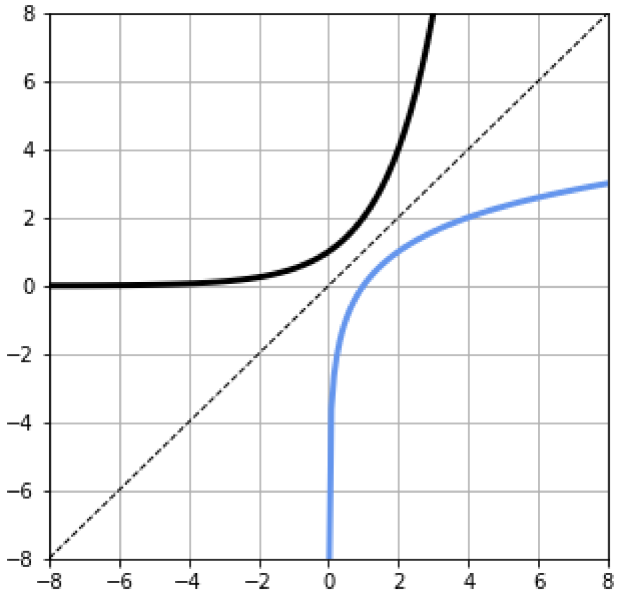
$log_a \frac{x}{y} = log_a x - log_a y$

$log_a x^y = y log_a x$

$log_a x = \frac{log_b x}{log_b a}$

In [2]:

```
x = np.linspace(-8, 8, 100)
y = 2**x


x2 = np.linspace (0.001, 8, 100) # np.log(0)는 에러가 되므로 0은 포함하지 않음
y2 = np.log (x2) / np.log (2) # 밑을 2로 한 log를 공식(7)로 계산

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()
```



로그는 곱셈의 항을 덧셈의 항으로 변환한다.

ex)

$$log\prod_{n=1}^{N} f(n) = \sum_{n=1}^{n} logf(n)$$

AI에서 주로 구하는 지점은 최소점을 찾는것이 주를 이루는데,

함수 f(x)가 주어지고 이 함수의 logf(x)의 최소점은 f(x)의 최소점이 된다.

역시 $e^{f(x)}$의 최소점도 f(x)의 최소점이 된다.

log함수의 장점은 곱셈을 덧셈으로 변환할때 사용되고,

e^x의 함수는 미분을 단순화 시킬때 사용된다.

특히 $f(x) = e^x$의 미분 $f'(x) = e^x$라는 성질을 이용하면, 풀이가 단순화 된다.

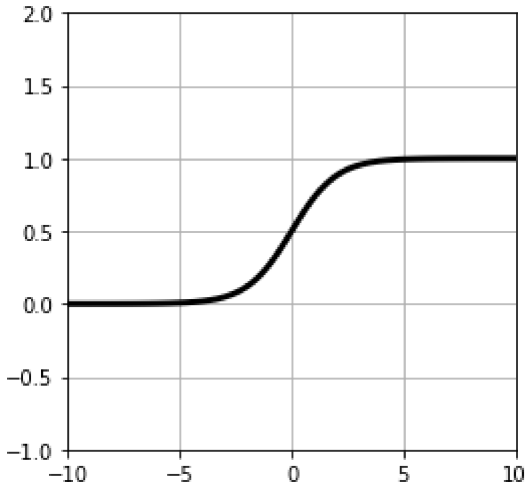
또한 log함수의 미분은

$$y'(x) = (logx)' = \frac{1}{x}$$

시그모이드 함수: $y = \frac{1}{1+e^{-x}}$

In [3]:

```
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()
```



이 함수의 특징은 x의 값이 $-\infty \cdots \infty$로 변할때, y의 값은 0~1사이로 변한다는것.

나중에 배울 분류문제나 신경망에서 뉴런의 특성에 중요한 함수로 나타난다.(중요)

이 함수의 미분형은:

$y' = y(1-y)$로 나타난다.(중요)

소프트맥스 함수: $y_i = \frac{e^{x_i}}{\sum_{j=0}^{K-1} e^{x_j}}$

이를 미분하면:

$\frac{\partial y_j}{\partial x_i} = y_j(\mathbf{I}_{ij} - y_i)$, 이 형태는 시그모이드 함수의 미분,

$y' = y(1-y)$와 유사한 형태를 구성하고 있다.

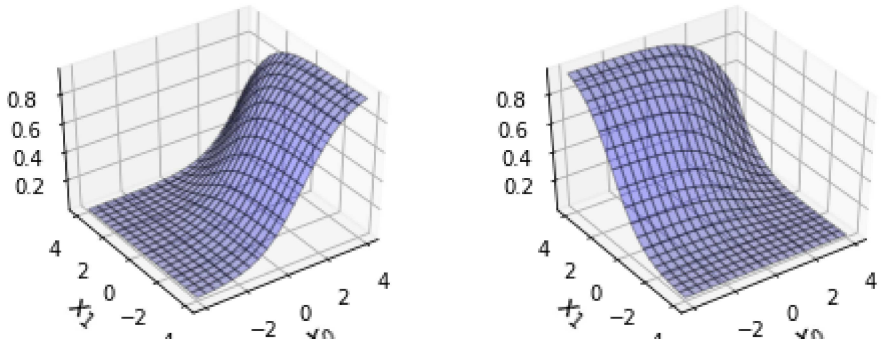
즉 시그모이드 함수를 다변수로 확장한것이 소프트맥스 함수라고 할수있다.

In [4]:
```
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)) # (A) 소수점 2 자리로 최대한 가깝게 표시
print(np.sum (y)) # (B) 합계를 표시
```

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

In [5]:
```
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()
```



소프트맥스 함수는 여러개의 변수(K)를 사용하는 분류 AI에 많이 사용된다.

이것은 나중에 적용할때 다시 설명한다.

가우스 함수: $y = ae^{-\frac{(x-\mu)^2}{\sigma^2}}$ 또는, $y = a\,exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$

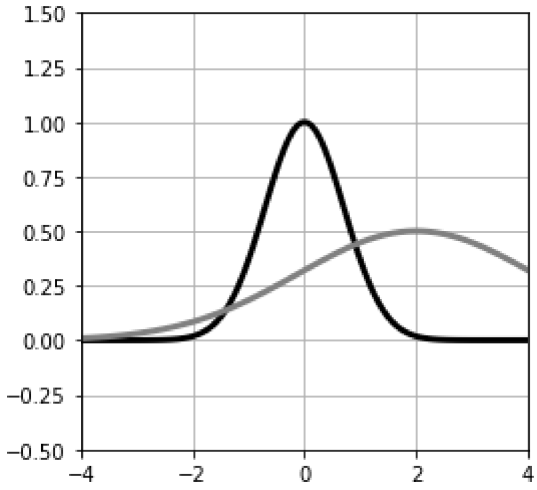
μ: 중심(평균)

σ: 확산(표준편차)

a: 높이

In [6]:

```
# 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()
```


1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−4
−2
0
2
4

4.4 행열

행열을 사용하면 많은 연립 방정식을 하나의 식으로 나타낼 수 있어 편리하다.

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix}$$

$a_{i,j}$ i행, j열의 요소

행열 덧셈 뺄셈

예제)

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1+7 & 2+8 & 3+9 \\ 4+10 & 5+11 & 6+12 \end{bmatrix} = \begin{bmatrix} 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

파이썬을 이용하여 계산하면

In [2]:
```
import numpy as np
A = np.array([[1,2,3],[4,5,6]])
B = np.array([[7,8,9],[10,11,12]])
A_plus_B = A + B
A_minus_B = A - B
print('vector plus: ', A_plus_B)
print('vector minus: ', A_minus_B)
```

```
vector plus:  [[ 8 10 12]
 [14 16 18]]
vector minus:  [[-6 -6 -6]
 [-6 -6 -6]]
```

행열 곱

예제)

$$\mathbf{A} = [1\ 2\ 3],\ \mathbf{B} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

$$\mathbf{A}B = [1\ 2\ 3] \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1X4 + 2X5 + 3X6 = 32$$

In [3]:
```
import numpy as np
A = np.array([1,2,3])
B = np.array([4,5,6])

A_mul_B= A.dot(B)
print('A_mul_B: ', A_mul_B)

B_mul_A = B.dot(A)
print('B_mul_A: ', B_mul_A)
```

```
A_mul_B:  32
```

```
B_mul_A:  32
```

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ -1 & -2 & -3 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 4 & -4 \\ 5 & -5 \\ 6 & -6 \end{bmatrix}$$

In [4]:
```
import numpy as np
A = np.array([[1,2,3],[-1,-2,-3]])
B = np.array([[4,-4],[5,-5],[6,-6]])

A_mul_B = A.dot(B)
print('A_mul_B: ', A_mul_B)

B_mul_A = B.dot(A)
print('B_mul_A: ', B_mul_A)
```

```
A_mul_B:  [[ 32 -32]
 [-32  32]]
B_mul_A:  [[ 8 16 24]
 [10 20 30]
 [12 24 36]]
```

위의 결과를 보면, 행열의 곱은, 교환 법칙이 성립되지 않는다.

일반 수식 $A \cdot B$와 다르다는 것이다. 반드시 기억해야한다.

단위 행열: $\mathbf{I}$

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In [5]:
```
# python numpy에서 identity
print(np.identity(3))
```

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

$$\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$$

In [6]:
```
#python으로 확인해 보자
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(A)

a = A.dot(np.identity(3))
print(a)
```

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

역행열: $\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$

역행열은 교환법칙이 성립된다.(중요)

전치(Transpose): $\mathbf{A}^T$

ex)

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix},\ \mathbf{A}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$(AB)^T = B^T A^T$

$(ABC)^T = C^T B^T A^T$

행열과 사상

행열은 벡터를 다른 벡터로 변화하는 규칙으로 볼수있다.

벡터라는 숫자의 조합을 좌표로 생각한다면, 행열은 좌표간 이동시키기 위한 규칙으로 볼수있다.

4.5 핵심공식

미분 연쇄 법칙

$f(w) = g(w)^2, g(w) = aw + b$

$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w} = 2(aw + b)a = 2a^2w + 2ab, \frac{\partial f}{\partial g} = 2g, \frac{\partial g}{\partial w} = a$

지수및 Log 미분

$y(x) = a^x, y'(x) = a^x log a, y = e^x, y'(x) = a^x$

$y(x) = log x, y(x)' = \frac{1}{x}$

시그모이드(Sigmoid) 함수

$\frac{1}{1 - e^{-x}}, y'(x) = y(1 - y)$

소프트맥스(Softmax) 함수

$u = e^{x0} + e^{x1} + e^{x2}$

$y_0 = \frac{e^{x0}}{u},\ y_1 = \frac{e^{x1}}{u},\ y_2 = \frac{e^{x2}}{u}$

$\frac{\partial y_j}{\partial x_i} = y_j(I_{ij} - y_i)$

제곱 오차 함수

$J = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)^2$

직선 Model

$y(x) = w_0x + w_1$

매개변수 구하기(경사하강법 또는 학습법칙)

$w(t+1) = w(t) - \alpha\frac{\partial J}{\partial w}$, 또는 $w(t+1) = w(t) - \alpha\frac{\partial E}{\partial w}$

$\frac{\partial J}{\partial w_0} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_n + w_1 - t_n)x_n = \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n$

$\frac{\partial J}{\partial w_1} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_n + w_1 - t_n) = \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)$

매개변수 해석해

$\frac{\partial J}{\partial w_0} = 0, \frac{\partial J}{\partial w_1} = 0, \frac{\partial J}{\partial w_2} = 0$

로지스틱 회귀 Model

$y = \sigma(w_0 x + w_1) = \frac{1}{1 + e^{-(w_0 x + w_1)}}$

$a = w_0 x + w_1$이면, $y = \sigma(a) = \frac{1}{1 + e^{-a}}$

선형 로지스틱 평균 교차 엔트로피 오차 함수

P(T|X) = $\prod_{n=0}^{N-1} P(t_n|x_n) = \prod_{n=0}^{N-1} y_n^{t_n}(1 - y_n)^{1-t_n}$인 경우

$E(w) = -\frac{1}{N} log P(T|X) = -\frac{1}{N} \sum_{n=0}^{N-1} \{t_n log y_n + (1 - t_n) log(1 - y_n)\}$

$E(w) = -\frac{1}{N} \sum_{n=0}^{N-1} (t_n log y_n + (1 - t_n) log(1 - y_n))$

로지스틱 회귀의 평균교차 엔트로피 오차함수 편미분

$\frac{\partial E}{\partial w_0} = \frac{1}{N} \sum_{n=0}^{N-1} (y_n - t_n) x_n, \frac{\partial E}{\partial w_1} = \frac{1}{N} \sum_{n=0}^{N-1} (y_n - t_n)$

평균교차 엔트로피 오차함수 편미분

$\frac{\partial E}{\partial w_{ki}} = \frac{1}{N} \sum_{n=0}^{N-1} (y_{nk} - t_{nk}) x_i$

신경망(소프트맥스)평균 교차 엔트로피 오차 함수

P(T|X) = $\prod_{n=0}^{N-1} P(t_n|x_n) = \prod_{n=0}^{N-1} y_{n0}^{t_{n0}} y_{n1}^{t_{n1}} y_{n2}^{t_{n2}} = \prod_{n=0}^{N-1} \prod_{k=0}^{K-1} y_{nk}^{t_{nk}}$인 경우

$E(w) = -\frac{1}{N} log P(T|X) = -\frac{1}{N} \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} t_{nk} log y_{nk}$

In [ ]:

5. 지도 학습(선형 모델)

5.1 1차원 입력 선형 모델

지도 학습(Supervised Learning):

훈련 data(Training Data)로부터 하나의 함수를 유추해내기 위한

머신러닝(Machine Learning)의 한 방법이다.

Vetor Type

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \\ \vdots \\ x_{N-1} \end{bmatrix}, \; t = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_n \\ \vdots \\ t_{N-1} \end{bmatrix}$$

x_n, t_n에서, x_n은 입력 변수, t_n은 목표 변수, n은 data의 순번을 의미한다.

여기에서는 교육 목적상, simulation data를 만들어서, 선형 입력 모델을 검토해보기로 한다.

아래의 예는, 5세부터 30세(x_n 변수)사이에, 키(t_n)의 data를 취하여, 해당 1차원(선형) 모델을 살펴 보기로 한다.

나이와 키의 관련 함수를 가상으로 만들어, simulation하기 위한 data를 만들어 봅시다.

가상 함수 f(x) = k_init - k_rate·$e^{-rate \cdot x_n}$ + 4*random(x_n)

k_init: 기본 키

k_rate: 나이와 관련된 키의 변화량

rate: 나이를 적용할 율

4*random(x_n): 동일한 나이에서 키의 변화

In [1]:

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


# 데이터 생성 ------------------------------
np.random.seed(seed=1) # 난수를 고정
X_min = 4 # X(나이)의 하한(표시 용)
X_max = 30 # XX(나이)의 상한(표시 용)
X_n = 16 # 시료수
X = 5 + 25 * np.random.rand(X_n)

# k_init = 170
# k_rate = 108
# rate = 0.2

k_init = 170
k_rate = 108
```

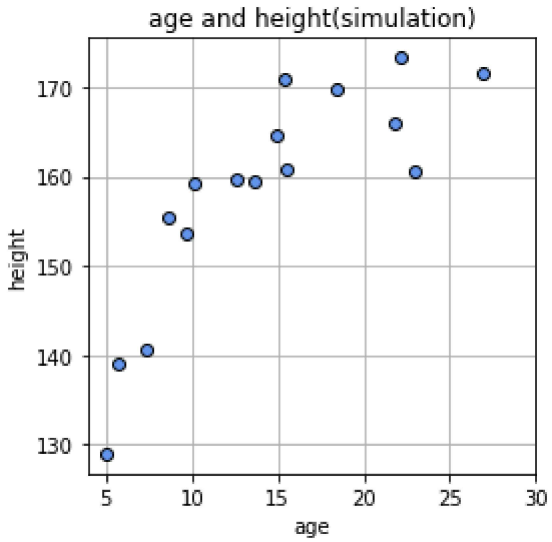
```
rate = 0.2
# T = Prm_c[0] - Prm_c[1] * np.exp(-Prm_c[2] * X) W
# + 4 * np.random.randn(X_n) # (A)
T = k_init - k_rate * np.exp(-rate * X) + 4 * np.random.randn(X_n)

print(np.round(X,2))
print(np.round(T,2))
np.savez('ch5_data.npz', X=X, X_min=X_min, X_max=X_max, X_n=X_n, T=T) # (B)
```

```
[15.43 23.01  5.   12.56  8.67  7.31  9.66 13.64 14.92 18.47 15.48 22.13
 10.11 26.95  5.68 21.76]
[170.91 160.68 129.   159.7  155.46 140.56 153.65 159.43 164.7  169.65
 160.71 173.29 159.31 171.52 138.96 165.87]
```

이것을 그래프로 그려보면,

In [2]:

```
plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')
plt.xlim(X_min, X_max)
plt.title('age and height(simulation)')
# plt.xlabel('나이') #error
plt.xlabel('age')
plt.ylabel('height')
plt.grid(True)
plt.show()
```



이 그래프만 보아서는 최선의 접근선을 찾아내기가 어렵다.

여기에 학습을 적용하여 최선의 접근선을 찾아 내는것이 본 과정의 목표이다.

이러한 현상과 data는 우리 생활 주변에 자주 나타나게된다.(응용편 조도측정부분에서 실습을 하게된다.)

우선 직선이라는 가정에서 접근해 보자

직선의 Model은

$$y(x) = w_0 x + w_1$$

w_0: 기울기, w_1: y 절편

입력 x에 y(x)를 출력하는 함수

여기에서, 본 그래프에 가장 근접하는 w_0와w_1을 찾는것이 목표이다.

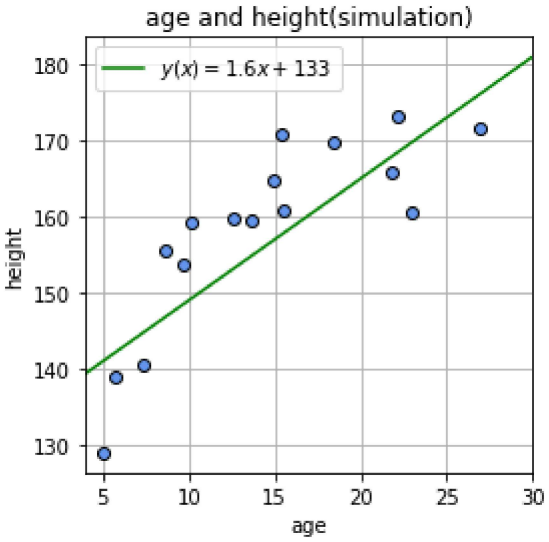
가장 쉬운 방법은 가장 근사한 직선을 그어보고 정하는것이다.

예) y(x) = 1.6x + 133

이것을 그려 보자.

In [3]:

```
def f(x):
    return 1.6*x + 133

x = np.linspace(4, 30, 16)

plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')

plt.plot(x, f(x), color = 'g', label='$y(x) = 1.6x + 133$')
plt.legend(loc = 'upper left')

plt.xlim(X_min, X_max)
plt.title('age and height(simulation)')
# plt.xlabel('나이') #error
plt.xlabel('age')
plt.ylabel('height')
plt.grid(True)
plt.show()
```



이렇게 해서는, 확신이 없다.

따라서 직선을 변화 시키면서, 각 점들과의 오차를 계산하여 최선의 선을 찾는것이 필요하다.

감각적으로 model(직선 model을 할것인가, 곡선 model을 할것인가)을 선정하되,

선정되면 어떻게 검증할것인가가 중요하다.

직선 model을 선정하되, 어떤 직선이 최선인가, 검증을 어떻게 할것인가가 문제이다.

여기에서는 제곱 오차 함수를 선정하여 검증하는 과정을 설명한다.

제곱 오차의 함수는:

$$\mathbf{J} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)^2$$

이 함수를 설명한다면,

$y_n = y(x_n) = w_0 x_n + w_1$으로써, w_0 , w_1을 변수로 변화 시켜가면서,

각각에 따라 출력되는 값을 의미한다.

표식은 간단하지만, 직관적으로 엄청난 계산이 포함된다고 생각할수 있다.

계산은 컴퓨터가 자동으로 해주기 때문에,

수식을 정확히 컴퓨터가 받아들일수있도록만 전개한다면 걱정할 필요가 없다.

위의 결과 J는 평균 제곱오차(mean square error: mse, MSE)로써,

위의 예는 각 점과 직선과의 차이 제곱의 평균이다.

w_0와w_1이 결정되면 J를 계산할수 있는데, 어떤 w_0와w_1의 쌍의 직선은,

데이터에서 크게 벗어나 J가 엄청 커진다.

위의 그래프를 보면 어떤 w_0와w_1를 선정하더라도 J가 0이 될수는 없다.

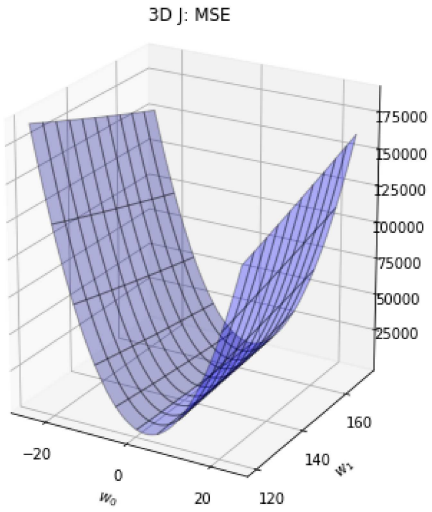
단지 최소값의 w_0와w_1를 찾아내는것이다.

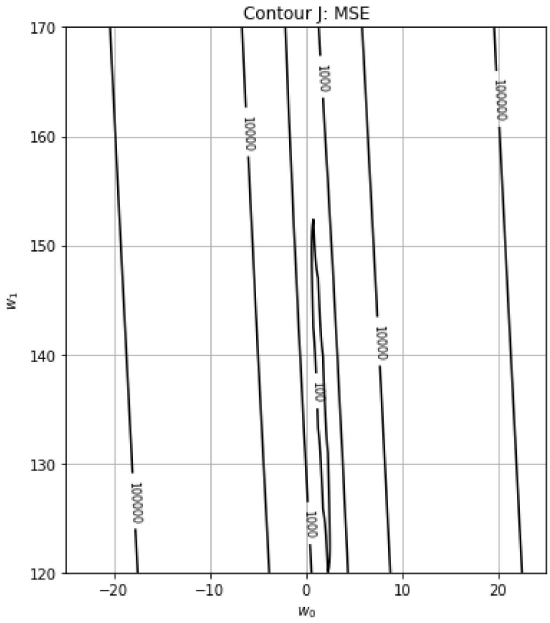
우선 w_0, w_1와 J의 관계를 그래프를 그려서, 전체 모양을 보자.

In [4]:

```
from mpl_toolkits.mplot3d import Axes3D
# 평균 오차 함수 ----------------------------
def mse_line(x, t, w): #J를 계산하는 함수.
    y = w[0] * x + w[1] # y(x) = w0*x + w1
    mse = np.mean((y - t)**2) # J = 1/N(y-t)^2
    return mse #return J

# 계산 ----------------------------------
wn = 100 # 등고선 표시 해상도
w0_range = [-25, 25] # w0의 범위
w1_range = [120, 170] # w1의 범위

w0 = np.linspace(w0_range[0], w0_range[1], wn) # w0의 간격과 data point, 100개 point
w1 = np.linspace(w1_range[0], w1_range[1], wn) # w1의 간격과 data point, 100개 point
ww0, ww1 = np.meshgrid(w0, w1)

# print(ww0)
J = np.zeros((len(w0), len(w0))) #J 행열의 초기화
for i0 in range(wn):
    for i1 in range(wn):
        J[i1, i0] = mse_line(X, T, (w0[i0], w1[i1])) # w0,w1의 각 Point에 대한 J값 저

# 표시 ----------------------------------
plt.figure(figsize=(15, 7))
# plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)

ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(ww0, ww1, J, rstride=10, cstride=10, alpha=0.3, #각 w0, w1에서의 J표시
```

```
                color='blue', edgecolor='black')
ax.set_xticks([-20, 0, 20])
ax.set_yticks([120, 140, 160])
ax.view_init(20, -60)

plt.title('3D J: MSE')
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')
# plt.zlabel('$J$')

plt.subplot(1, 2, 2)
cont = plt.contour(ww0, ww1, J, 30, colors='black',
                   levels=[100, 1000, 10000, 100000])
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
plt.title('Contour J: MSE')
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')

plt.show()
```





위 그래프를 보면, w_0의 변화에 따라 J의 값이 크게 변하는것을 알수있다.

옆의 등고선을 보면, $w_0 : 1$부근, $w_1 : 135$부근에서 최소값을 갖는것 같다.

이제는 최소값을 같는 값을 찾아야 하는데, 미분을 이용해서 그 값을 찾는 과정을 살펴보자.

우선 임의의 w_0, w_1에서의 J값의 기울기를 확인하고,

J가 감소하는 방향으로 w_0, w_1을 조금씩 진행한다.

이것을 반복하여 J가 가장 작아지는 w_0, w_1에 도달할 것이다.

어떤 임의의 w_0, w_1에서, 관찰해보면 골짜기 위쪽 방향은 J를 w_0, w_1로 편미분한 벡터

$\left[\frac{\partial J}{\partial w_0} \ \frac{\partial J}{\partial w_1} \right]^T$로 표시한다.

이것이 J의 w_0, w_1에 대한 기울기이고, $\nabla_{\mathbf{w}} J$로 나타낸다.

J를 최소화하려면 J의 기울기 반대 방향($-\nabla_{\mathbf{w}} J$)으로 진행하면 된다.

$\mathbf{w}$의 갱신 방법(학습 법칙)을 행열로 표기하면,

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \nabla_{\mathbf{w}} J|_{w(t)}$$

현재의 w값을 대입한, J의 w에 대한 편미분, $\nabla_{\mathbf{w}} J|_{w(t)}$은 w(t)지점에서의 J기울기를 나타낸다.

또한 α는 학습율이라고 하고, 여기서는 w의 갱신폭을 나타낸다.

$$\mathbf{w}_0(t+1) = \mathbf{w}_0(t) - \alpha \nabla_{\mathbf{w}_0} J|_{w0(t),w1(t)}$$

$$\mathbf{w}_1(t+1) = \mathbf{w}_1(t) - \alpha \nabla_{\mathbf{w}_1} J|_{w0(t),w1(t)}$$

$\mathbf{J} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(w_0 x_n + w_1 - t_n)^2$로 부터, 각 항별 편미분을 정리하면,

$$\frac{\partial J}{\partial w_0} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0 x_n + w_1 - t_n)x_n = \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n$$

$$\frac{\partial J}{\partial w_1} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0 x_n + w_1 - t_n) = \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)$$

정리하면,

$$\mathbf{w}_0(t+1) = \mathbf{w}_0(t) - \alpha\frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n$$

$$\mathbf{w}_1(t+1) = \mathbf{w}_1(t) - \alpha\frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)$$

이렇게 정리되었으니, 이 수식을 컴퓨터가 계산할수있도록 하면, 컴퓨터가 결론을 낼것이다.

프로그램을 전개하다보면, 수식의 의미를 더욱더 정확히 파악할수 있다.

물론 파이썬 프로그램에 익숙해있을때 이야기 이지만.

여하튼 위의 식을 프로그램으로 전개하면

In [5]:
```
def dmse_line(x, t, w): #각 w0, w1와 x배열, t배열을 받아
    y = w[0] * x + w[1] #x배열에 의한 y배열을 계산하고
    d_w0 = 2 * np.mean((y - t) * x) #w0에 의한 편미분
    d_w1 = 2 * np.mean(y - t) #w1에 의한 편미분
    return d_w0, d_w1
```

임의의 점(w0,w1)에서의 기울기를 확인하자.

In [6]:
```
d_w = dmse_line(X, T, [10, 165])
print(np.round(d_w, 1))

org_d_w = dmse_line(X, T, [1.6, 133])
print(np.round(org_d_w, 1))
```

```
[5046.3  301.8]
[-61.7  -4.5]
```

위의 결과는 (w0=10, w1=165)일때 w0의 기울기는=5046.3, w1의 기울기=301.8이라는 것이다.

이와 같이하여, 각 요소에 의한 편미분항을 계산하였음으로, 이것을 기반으로 최저점을 찾는 프로그램을 완성한다.

먼저 최저점을 찾는 함수만 정리해 본다.

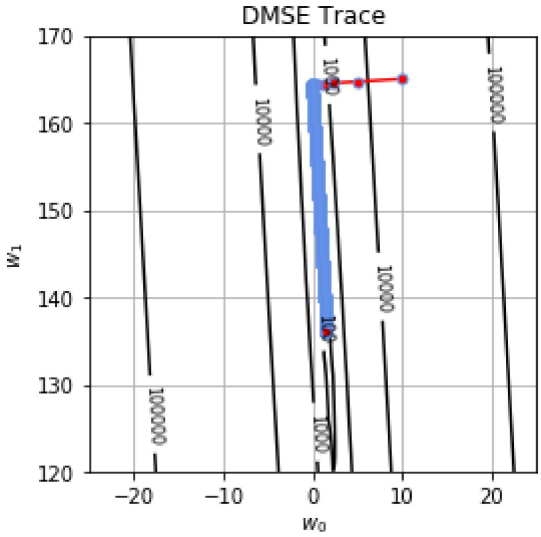
In [7]:
```
#최저점찾는 함수
def fit_line_num(x, t):
    w_init = [10.0, 165.0] # 초기 매개 변수, 어디에서부터 추적을 시작할것인가를 결정,
    #(w0,w1)의 그래프상으로 대략의 최저점을 확인후, 근처의 점부터 시작하는것이 좋다.

    alpha = 0.001 # 학습률
    i_max = 100000 # 반복의 최대 수, 무한 반복을  방지하기 위해
    eps = 0.1 # 반복을 종료 기울기의 절대 값의 한계, 0.0으로 하면 좋겠지만,
    #무한 반복을 막기위해 조금 큰수부터 줄여가는것이 좋다.

    w_i = np.zeros([i_max, 2]) #(w0,w1)배열 초기화
    w_i[0, :] = w_init #시작점 입력

    print('시작 Point: ', w_i[0])

    for i in range(1, i_max): #1~100000 반복 수행, 목표치 eps=0.1이하가 달성되면 반복
        dmse = dmse_line(x, t, w_i[i - 1]) #dmse: w0, w1의 error의 편미분 제곱 평균값
        w_i[i, 0] = w_i[i - 1, 0] - alpha * dmse[0] # w0(t) = w0(t-1) - a*J의 w0편미분
        w_i[i, 1] = w_i[i - 1, 1] - alpha * dmse[1] # w1(t) = w1(t-1) - a*J의 w1편미분
        if max(np.absolute(dmse)) < eps: # 종료판정, np.absolute는 절대치
            break

    w0 = w_i[i, 0] #종결된 w0의 값
    w1 = w_i[i, 1] #종결된 w1의 값
    w_i = w_i[:i, :] #종결되기 전까지 진행된 w0,w1의 행열값, 진행절차를 확인하기 위해
    return w0, w1, dmse, w_i
```

이상의 결과를 그래프로 확인해 본다.

In [8]:
```
plt.figure(figsize=(4, 4)) # MSE의 등고선 표시
xn = 100 # 등고선 해상도
w0_range = [-25, 25]
w1_range = [120, 170]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=(100, 1000, 10000, 100000))
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
# 구배법 호출
W0, W1, dMSE, W_history = fit_line_num(X, T)
# 결과보기
print('반복 횟수 {0}'.format(W_history.shape[0]))
print('완료 시점 W=[{0:.6f}, {1:.6f}]'.format(W0, W1))
print('dMSE=[{0:.6f}, {1:.6f}]'.format(dMSE[0], dMSE[1]))
print('MSE={0:.6f}'.format(mse_line(X, T, [W0, W1])))
plt.plot(W_history[:, 0], W_history[:, 1], '.-',
         color='red', markersize=10, markeredgecolor='cornflowerblue')

plt.title('DMSE Trace')
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')
```

```
plt.show()
```

```
시작 Point:  [ 10. 165.]
반복 횟수 13820
완료 시점 W=[1.539947, 136.176160]
dMSE=[-0.005794, 0.099991]
MSE=49.027452
```



위의 결과가 도출 되었다.

시작 Point: w0= 10, w1= 165에서부터 error가 감소하는 방향으로 진행, w0를 0의 방향으로 진행

반복 횟수: 13820회

완료 시점 w0=1.539947, w1=136.176160이다.

각각의 dMSE: w0_dMSE= -0.005794, w1_dMSE= 0.099991

MSE(전체 error값의 합): 49.027452

w0, w1의 수정 진행 방향은, 초기값에서 w0값의 변화가 큼으로, w0를 변화 시켜(10에서 0방향으로),

w0에 의한 최저점으로 진행한후, w1값을 변화시켜(165->136방향으로)가면서,

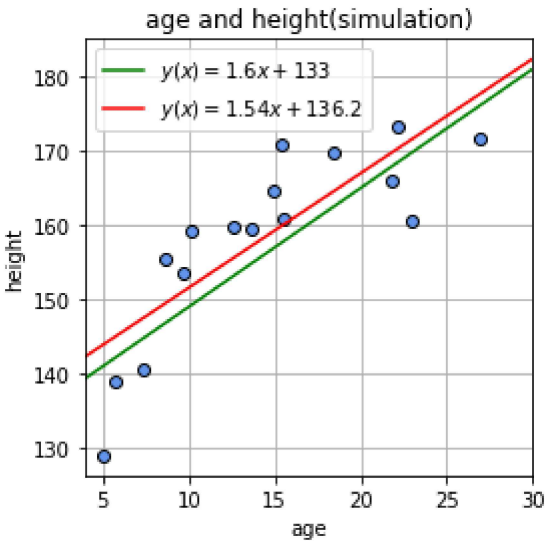
그 지점에서 다시 w0를 (0에서 1.5방향으로) 조금씩 이동하여간다.

이 지점을 적용하여 그래프를 다시 그려봅시다.

In [9]:

```
def f(x):
    return 1.6*x + 133

def f_new(x):
    return 1.539947*x + 136.17616

x = np.linspace(4, 30, 16)

plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')

plt.plot(x, f(x), color = 'g', label='$y(x) = 1.6x + 133$')
```

```
plt.plot(x, f_new(x), color = 'r', label='$y(x) = 1.54x + 136.2$')

plt.legend(loc = 'upper left')

plt.xlim(X_min, X_max)
plt.title('age and height(simulation)')
# plt.xlabel('나이') #error
plt.xlabel('age')
plt.ylabel('height')
plt.grid(True)
plt.show()
```



이와같은 과정으로 결과를 알아내는 방법을, 경사 하강법이라 한다. 컴퓨터의 도움에 의해 근사값을 구하는 방법이 수치 계산법이다.

또한 이러한 해를 수치해라고 한다.

이러한 직선 model의 경우는 방정식을 풀어가는 과정을 통해서 정확한 해를 구할수 있다. 이러한 해를 해석해라고 한다.

해석해를 사용하면 반복계산이 아니고,1회 계산으로 최적의 w를 구할수 있다.

그럼 해석해를 진행해 봅시다. 이를 통해서 문제의 본질을 이해할수있고 또한 해석해의 한계를 알아낼수도 있다.

해석해는 위와같은 1차 함수의 경우는 적용할수 있으나, 일반 선형의 경우는 해석해로 해결되지 않은것을 알수있다.

5.2 해석해

목표는 J가 극소화된는 지점 $\mathbf{w}$를 찾기이다.

그 지점의 기울기는 0이 됨으로, 기울기가 0이되는 지점의$\mathbf{w}$, $\frac{\partial J}{\partial w0} = 0$과 $\frac{\partial J}{\partial w_1} = 0$을 충족하는 w_0과w_1을 찾으면 됨.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#load data
outfile = np.load('ch5_data.npz')
X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']
```

In [2]:
```
def mse_line(x, t, w):
    y = w[0] * x + w[1]
    mse = np.mean((y - t)**2)
    return mse


def show_line(w):
    xb = np.linspace(X_min, X_max, 100)
    y = w[0] * xb + w[1]
    plt.plot(xb, y, color=(.5, .5, .5), linewidth=4)
```

$$\frac{\partial J}{\partial w_0} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_n + w_1 - t_n)x_n = 0$$

$$\frac{\partial J}{\partial w_1} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_n + w_1 - t_n) = 0$$

$$\frac{1}{N}\sum_{n=0}^{N-1} w_0x_n^2 = w_0\frac{1}{N}\sum_{n=0}^{N-1} x_n^2 = w_0 < x^2 >, < x^2 >= \frac{1}{N}\sum_{n=0}^{N-1} x_n^2$$

산술평균 기호: $\frac{1}{N}\sum_{n=0}^{N-1} x_n =< x >$

$$\frac{1}{N}\sum_{n=0}^{N-1}(w_0x_n + w_1 - t_n)x_n = \frac{1}{N}\sum_{n=0}^{N-1} w_0x_n^2 + \frac{1}{N}\sum_{n=0}^{N-1} w_1x_n - \frac{1}{N}\sum_{n=0}^{N-1} t_nx_n = 0$$

$$\frac{1}{N}\sum_{n=0}^{N-1} w_1x_n = w_1\frac{1}{N}\sum_{n=0}^{N-1} x_n = w_1 < x >, < x >= \frac{1}{N}\sum_{n=0}^{N-1} x_n$$

$$\frac{1}{N}\sum_{n=0}^{N-1} t_nx_n = \frac{1}{N}\sum_{n=0}^{N-1} t_nx_n =< tx >, < tx >= \frac{1}{N}\sum_{n=0}^{N-1} t_nx_n$$

이를 정리하면,

$$w_0 < x^2 > +w_1 < x > - < tx >= 0$$

같은 방법으로

$$\frac{\partial J}{\partial w_1} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0 x_n + w_1 - t_n) = w_0 < x > + w_1 - < t >= 0$$

$w_1 =< t > - w_0 < x >$임으로,

$w_0 < x^2 > + w_1 < x > - < tx >= 0$에 대입하여 정리하면,

$$w_0 = \frac{< tx > - < t >< x >}{< x^2 > - < x >^2}$$

$$w_1 =< t > - \frac{< tx > - < t >< x >}{< x^2 > - < x >^2}$$

이것이 $f(w)$의 해석해이다.

직선 Model인것이라는 가정에서는, 수치해보다는 해석해가 간단할수 있어,

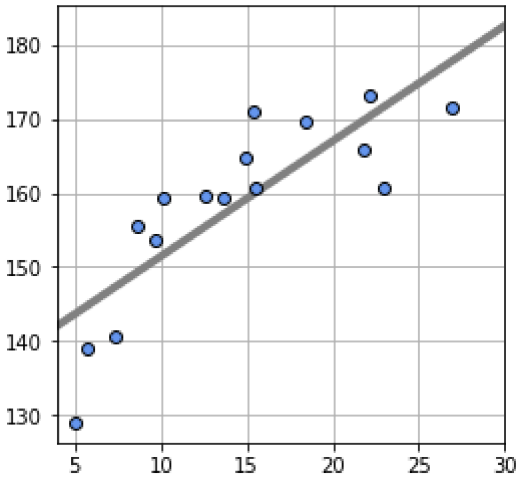
일부러 경사 하강법을 사용할 필요가 없으나,

해석해가 구해지지 않는 Model에서는 경사 하강법으로 해를 구해야 한다.

In [3]:

```
# 해석해 ------------------------------------
def fit_line(x, t):
    mx = np.mean(x)
    mt = np.mean(t)
    mtx = np.mean(t * x)
    mxx = np.mean(x * x)
    w0 = (mtx - mt * mx) / (mxx - mx**2)
    w1 = mt - w0 * mx
    return np.array([w0, w1])

# 메인 ---------------------------------
W = fit_line(X, T)
print("w0={0:.3f}, w1={1:.3f}".format(W[0], W[1]))
mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.figure(figsize=(4, 4))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=1.558, w1=135.872
SD=7.001 cm
```


180
170
160
150
140
130
5
10
15
20
25
30

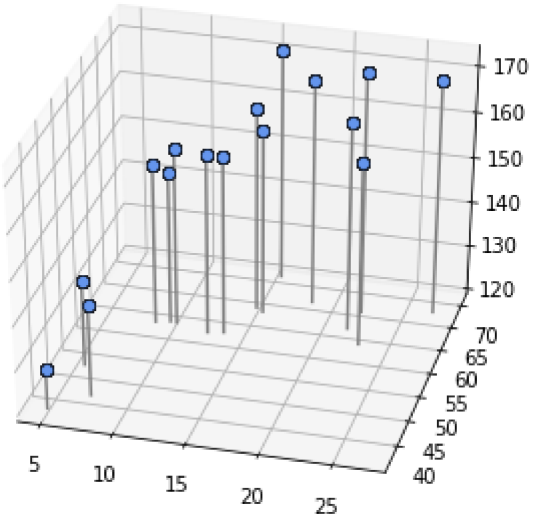
5.3 2차원 입력 model

$$\mathbf{X} = (x_0, x_1) \ => \ x_0 : age, x_1 : weight$$

$$weight = 23x\frac{height^2}{100} + noise$$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D #for raspberry pi 4

#X0: 나이, X1: 키

outfile = np.load('ch5_data.npz')
X0 = outfile['X']
X0_min = outfile['X_min']
X0_max = outfile['X_max']
X0_n = outfile['X_n']
T = outfile['T']

np.random.seed(seed=1) # 난수를 고정
X1 = 23 * (T / 100)**2 + 2 * np.random.randn(X0_n)
X1_min = 40
X1_max = 75
```

In [2]:
```
# 2차원 데이터의 표시 ------------------------
def show_data2(ax, x0, x1, t):
    for i in range(len(x0)):
        ax.plot([x0[i], x0[i]], [x1[i], x1[i]],
                [120, t[i]], color='gray')
        ax.plot(x0, x1, t, 'o',
                color='cornflowerblue', markeredgecolor='black',
                markersize=6, markeredgewidth=0.5)
        ax.view_init(elev=35, azim=-75)


# 메인 ----------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1,1,1,projection='3d')
show_data2(ax, X0, X1, T)
plt.show()
```



$y(x) = w_0x_0 + w_1x_1 + w_2$에서,

w_0, w_1, w_2에 다양한 값을 넣어 여러 위치와 기울기를 갖는 면을 나타낼수있음

2차원 입력의 해석해(data에 최적의 $\mathbf{w} = [w_0, w_1, w_2]$를 구하는것)

1차원에서와 마찬가지로 평균 제곱오차($\mathbf{J}$)를 나타내면

$$\mathbf{J} = \frac{1}{N}\sum_{n=0}^{N-1}(y(x_n) - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(w_0x_{n,m0} + w_1x_{n,1} + w_2 - t_n)^2$$

$\mathbf{w}$를 움직이면 면이 여러방향을 향하게 되고, 그에 따라 $\mathbf{J}$가 변하는데, $\mathbf{J}$가 가장 작아지는 $\mathbf{w} = [w_0, w_1, w_2]$를 구하는것

$\mathbf{J}$가 최소화하는 최적의 $\mathbf{w}$는 기울기가 0(편미분이 0), $\frac{\partial J}{\partial w_0} = 0, \frac{\partial J}{\partial w_1} = 0, \frac{\partial J}{\partial w_2} = 0$

w_0에대한편미분 : $\frac{\partial J}{\partial w_0} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_{n,0} + w_1x_{n,1} + w_2 - t_n)x_{n,0}$

$= 2\{w_0 < x_0^2 > + w_1 < x_0x_1 > + w_2 < x_0 > - < tx_0 >\} = 0$

w_1에대한편미분 : $\frac{\partial J}{\partial w_1} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_{n,0} + w_1x_{n,1} + w_2 - t_n)x_{n,1}$

$= 2\{w_0 < x_0x_1 > + w_1 < x_1^2 > + w_2 < x_1 > - < tx_1 >\} = 0$

w_2에대한편미분 : $\frac{\partial J}{\partial w_2} = \frac{2}{N}\sum_{n=0}^{N-1}(w_0x_{n,0} + w_1x_{n,1} + w_2 - t_n)$

$= 2\{w_0 < x_0 > + w_1 < x_1 > + w_2 - < t >\} = 0$

이 3개의 연립방정식을 정리하면,

$$w_0 = \frac{cov(t, x_1)cov(x_0, x_1) - var(x_1)cov(t, x_0)}{cov(x_0, x_1)^2 - var(x_0)var(x_1)}$$

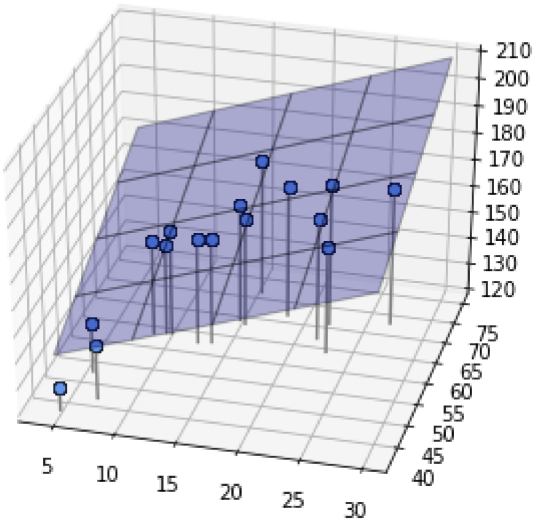
$$w_1 = \frac{cov(t, x_0)cov(x_0, x_1) - var(x_0)cov(t, x_1)}{cov(x_0, x_1)^2 - var(x_0)var(x_1)}$$

$$w_2 = -w_0 < x_0 > -w_1 < x_1 > + < t >$$

In [3]:
```
#면의 표시 ---------------------------------
```

```
def show_plane(ax, w):
    px0 = np.linspace(X0_min, X0_max, 5)
    px1 = np.linspace(X1_min, X1_max, 5)
    px0, px1 = np.meshgrid(px0, px1)
    y = w[0]*px0 + w[1] * px1 + w[2]
    ax.plot_surface(px0, px1, y, rstride=1, cstride=1, alpha=0.3,
                    color='blue', edgecolor='black')

#면의 MSE ----------------------------------
def mse_plane(x0, x1, t, w):
    y = w[0] * x0 + w[1] * x1 + w[2] # (A)
    mse = np.mean((y - t)**2)
    return mse

# 메인 ----------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1, 1, 1, projection='3d')
W = [1.5, 1, 90]
show_plane(ax, W)
show_data2(ax, X0, X1, T)
mse = mse_plane(X0, X1, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

```
SD=12.876 cm
```

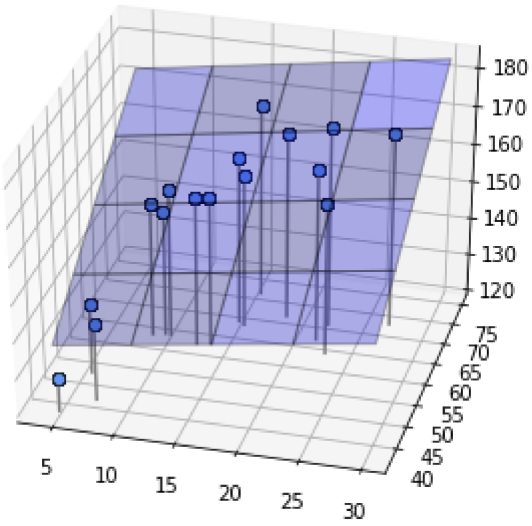


In [4]:

```
# 해석해 ----------------------------------
def fit_plane(x0, x1, t):
    c_tx0 = np.mean(t * x0) - np.mean(t) * np.mean(x0)
    c_tx1 = np.mean(t * x1) - np.mean(t) * np.mean(x1)
    c_x0x1 = np.mean(x0 * x1) - np.mean(x0) * np.mean(x1)
    v_x0 = np.var(x0)
    v_x1 = np.var(x1)
    w0 = (c_tx1 * c_x0x1 - v_x1 * c_tx0) / (c_x0x1**2 - v_x0 * v_x1)
    w1 = (c_tx0 * c_x0x1 - v_x0 * c_tx1) / (c_x0x1**2 - v_x0 * v_x1)
    w2 = -w0 * np.mean(x0) - w1 * np.mean(x1) + np.mean(t)
    return np.array([w0, w1, w2])


# 메인 ----------------------------------
plt.figure(figsize=(6, 5))
ax = plt.subplot(1, 1, 1, projection='3d')
W = fit_plane(X0, X1, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0], W[1], W[2]))
show_plane(ax, W)
show_data2(ax, X0, X1, T)
```

```
mse = mse_plane(X0, X1, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.show()
```

```
w0=0.5, w1=1.1, w2=89.0
SD=2.546 cm
```



결과를 보면 SD=2.546cm로, 1차원 입력일때 SD=7.001cm보다 오차가 많이 줄었음.

다차원 선형 회귀 모델

$$y(x) = w_0x_0 + w_1x_1 + \cdots + w_{D-1} + w_D$$

w_D는 절편이고 x가 곱해지지 않음. 만약 이 항이 없다면 어떤 $\mathbf{w}$에서도 $\mathbf{x} = [0, 0, \cdots, 0]$에서는 y가 0이됨

이 절편이 없으면, 그래프가 상하로 평행이동이 안됨.

행렬 표기법으로 정리하면

$$y(x) = w_0x_0 + w_1x_1 + \cdots + w_{D-1} = [w_0 \cdots w_{D1}] \begin{bmatrix} x_0 \\ \vdots \\ x_{D-1} \end{bmatrix} = \mathbf{w^T x}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ \vdots \\ w_{D-1} \end{bmatrix}$$

$$\mathbf{w^T} = [w_0 \cdots w_{D-1}]$$

$$J(w) = \frac{1}{N}\sum_{n=0}^{N-1}(y(x_n) - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(\mathbf{w^T x_n} - t_n)^2$$

$$\frac{\partial J}{\partial w_i} = \frac{1}{N}\sum_{n=0}^{N-1}\frac{\partial}{\partial w_i}(\mathbf{w^T x_n} - t_n)^2 = \frac{2}{N}\sum_{n=0}^{N-1}(\mathbf{w^T x_n} - t_n)x_{n,i}$$

J를 최소로 만드는 $\mathbf{w}$는 모든 w_i방향에 대한 기울기가 0인, $\frac{2}{N}\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)x_{n,i}=0$,

즉 $\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)x_{n,i}=0$

모든 w_i에 대해 전개해보면

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)x_{n,0}=0$,

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)x_{n,1}=0$,

$\vdots$

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)x_{n,D-1}=0$,

이것을 벡터형으로 전개하면,

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)[x_{n,0},x_{n,1},\cdots,x_{n,D-1}]=[0,0,\cdots,0]$

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n}-t_n)\mathbf{x_n^T}=[0,0,\cdots,0]$

$\sum_{n=0}^{N-1}(\mathbf{w^T x_n x_n^T}-t_n\mathbf{x_n^T})=[0,0,\cdots,0]$

$\mathbf{w^T}\sum_{n=0}^{N-1}\mathbf{x_n x_n^T}-\sum_{n=0}^{N-1}t_n\mathbf{x_n^T}=[0,0,\cdots,0]$

$\mathbf{w^T X^T X}-\mathbf{t^T X}=[0,0,\cdots,0]$, 여기에서 $\mathbf{X}$를 표시하면

$\mathbf{X}=\begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,D-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{0,D-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N-1,0} & x_{N-1,1} & \cdots & a_{N-1,D-1} \end{pmatrix}$, $\sum_{n=0}^{N-1}\mathbf{x_n x_n^T}=\mathbf{X^T X}$, $\sum_{n=0}^{N-1}t_n\mathbf{x_n^T}=\mathbf{t^T X}$

$\mathbf{w^T X^T X}-\mathbf{t^T X}=[0,0,\cdots,0]$에서부터, $\mathbf{w}=$의 형태로 전개를 나타내기위해

양변을 전치하면, $(\mathbf{w^T X^T X}-\mathbf{t^T X})^\mathbf{T}=[0,0,\cdots,0]^T$, 이에 다음의 법칙을 적용,
$(\mathbf{A}+\mathbf{B})^\mathbf{T}=\mathbf{A^T}+\mathbf{B^T}$

$(\mathbf{w^T X^T X})^\mathbf{T}-(\mathbf{t^T X})^\mathbf{T}=[0,0,\cdots,0]^T$, 여기에 $(\mathbf{A^T})^\mathbf{T}=\mathbf{A}$라는 관계식과
$(\mathbf{AB})^\mathbf{T}=\mathbf{B^T A^T}$를 적용하면

$(\mathbf{X^T X})^\mathbf{T}(\mathbf{w^T})^\mathbf{T}-\mathbf{X^T t}=[0,0,\cdots,0]^T$,

$(\mathbf{X^T X})\mathbf{w}-\mathbf{X^T t}=[0,0,\cdots,0]^T$,

$(\mathbf{X}^{\mathbf{T}}\mathbf{X})\mathbf{w} = \mathbf{X}^{\mathbf{T}}\mathbf{t}$,

$\mathbf{w} = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{t}$, 이것이 D차원 선형 회귀 모델의 해가 됨.

$(\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}$를 무어-펜로즈 의사역행렬이라고 함.

5.4 선형기저함수

지금까지는 직선 모델이라는 가정하에서 키를 예측했는데,

데이터형태를 달리보면 굽은 곡선에 데이터가 있는것으로 볼수도 있다.

곡선 모델이라고 할때는 선형 기저함수 모델을 적용하는것이 유리하다.

선형 기저함수는, 우선 기저 함수를 정하고, 그 기저 함수들의 조합으로 선형을 완성하는것으로 한다.

우선 기저 함수로 자주 사용하는 가우스 함수를 적용해보자

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


# 데이터 로드 ----------------------------
outfile = np.load('ch5_data.npz')
X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']
```

$$\phi_j(x) = exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

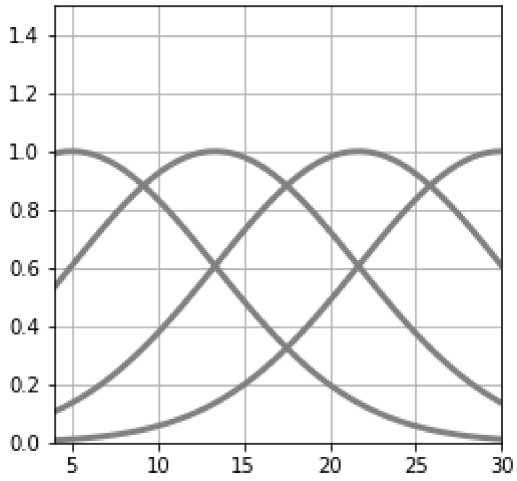
기저함수의 중심위치: μ_j

함수의 확장 정도: s

이 가우스 함수를 program하면

In [2]:
```
def gauss(x, mu, s):
    return np.exp(-(x - mu)**2 / (2 * s**2))
```

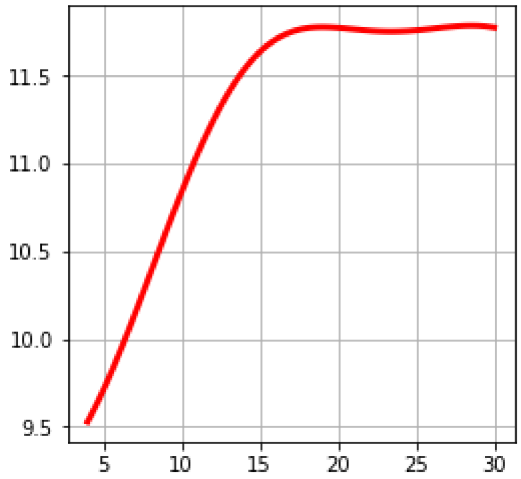
In [3]:
```
M = 4
plt.figure(figsize=(4, 4))
mu = np.linspace(5, 30, M)
s = mu[1] - mu[0] # (A)
xb = np.linspace(X_min, X_max, 100)
for j in range(M):
    y = gauss(xb, mu[j], s)
    plt.plot(xb, y, color='gray', linewidth=3)
plt.grid(True)
plt.xlim(X_min, X_max)
plt.ylim(0, 1.5)

plt.show()
```



$y(x) = w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) + w_4$에서,
$w_0 = -2, w_1 = 3, w_2 = -1, w_3 = 2, w_4 = 10$을 적용하여 그래프를 그려보면,

In [4]:

```
plt.figure(figsize=(4, 4))

yy = -2*gauss(xb, mu[0], s) + 3*gauss(xb, mu[1], s) - gauss(xb, mu[2], s) + 2*gauss(x
plt.plot(xb, yy, color='red', linewidth=3)
plt.grid(True)

plt.show()
```



여기에서 w_n을 가중치 매개변수라 하고, 이같은 계산을 가중치를 붙여 더한다고 한다.

특이한점은 마지막 $w_M = 10$이 곡선 상하의 평행 이동을 조절하는 중요한 역할을 한다.

이항은 추후 자주 나오게 되니, 잘 기억해놓기 바란다.

이항은 $\phi_4(x) = 1$이라고 하여, 전체 식을 일반화 하면

$$y(x, w) = \sum_{j=0}^{M} w_j\phi_j(x) = \mathbf{w}^T\phi(x)$$

이 결과에 평균 제곱 오차 J는 $J(w) = \frac{1}{N}\sum_{n=0}^{N-1}(\mathbf{w}^T\phi(x_n) - t_n)^2$

이것은 1차 직선 모델의 평균 제곱 오차 $J(w) = \frac{1}{N}\sum_{n=0}^{N-1}(\mathbf{w}^T\mathbf{x}_n - t_n)^2$과 유사한 형태를 가지고

있다.

선형기저함수(data와 추세를 맞추어 적용하는 model)

키는 나이가 들면서 점차 커지지만 일정한 값으로 수렴하게 되는 특성을 적용하여,

$y(x) = w_0 - w_1 e^{(-w_2 x)}$

이함수의 특성은 x가 증가 하면 w_0값으로 수렴함, w_1은 시작점을 결정하고, w_2는 기울기를 결정함.

이제 data에 맞는 w_0, w_1, w_2를 구하는데, 형태가 지수함수임으로 해석해를 구하기가 쉽지 않음,

따라서 수치해를 사용하여 해를 구함.

파이썬의 scipy.optimize에 포함된 minimize함수를 사용하여 최적 매개변수를 구하는 방법으로 진행.

이함수는 최소값을 구하는 함수와 매개 변수의 초기 값만 주면, 함수의 미분을 주지않아도 매개변수의 극소값을 알아냄.

이 함수의 model을 model_U로 하면,

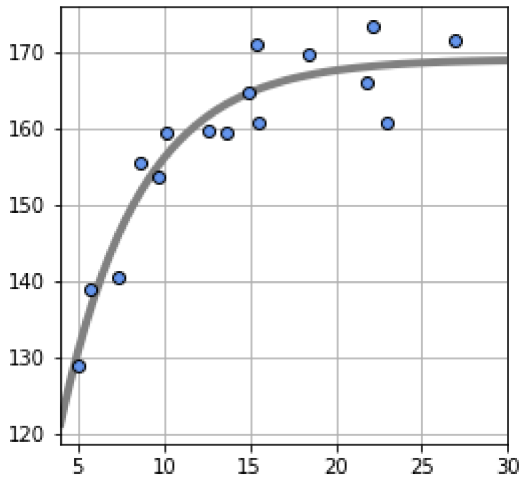
In [5]:
```
# 모델 U ------------------------------------
def model_U(x, w):
    y = w[0] - w[1] * np.exp(-w[2] * x)
    return y

# 모델 A 표시 -------------------------------
def show_model_U(w):
    xb = np.linspace(X_min, X_max, 100)
    y = model_U(xb, w)
    plt.plot(xb, y, c=[.5, .5, .5], lw=4)


# 모델 A의 MSE ------------------------------
def mse_model_U(w, x, t):
    y = model_U(x, w)
    mse = np.mean((y - t)**2)
    return mse
```

In [6]:
```
from scipy.optimize import minimize

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

outfile = np.load('ch5_data.npz')
X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']

# 모델 A의 매개 변수 최적화 -----------------
def fit_model_U(w_init, x, t):
    res1 = minimize(mse_model_U, w_init, args=(x, t), method="powell")
    return res1.x
```

In [7]:

```
plt.figure(figsize=(4, 4))
W_init=[100, 0, 0]
W = fit_model_U(W_init, X, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0], W[1], W[2]))
show_model_U(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue',markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_model_U(W, X, T)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

```
w0=169.0, w1=113.7, w2=0.2
SD=3.86 cm
```



결과는 SD=3.86cm로 직선 model SD=7.001cm보다 훨씬 작게 나타남.

model의 선택

AI를 함에있어서, 어떤 model이 더좋은지 어떻게 판단해야 할것인가.

In [8]:

```
# 교차 검증 model_A --------------------------
def kfold_model_U(x, t, k):
    n = len(x)
    mse_train = np.zeros(k)
    mse_test = np.zeros(k)
    for i in range(0, k):
        x_train = x[np.fmod(range(n), k) != i]
        t_train = t[np.fmod(range(n), k) != i]
        x_test = x[np.fmod(range(n), k) == i]
        t_test = t[np.fmod(range(n), k) == i]
        wm = fit_model_U(np.array([169, 113, 0.2]), x_train, t_train)
        mse_train[i] = mse_model_U(wm, x_train, t_train)
        mse_test[i] = mse_model_U(wm, x_test, t_test)
    return mse_train, mse_test


# 메인 ----------------------------------
K = 16
Cv_A_train, Cv_A_test = kfold_model_U(X, T, K)
mean_A_test = np.sqrt(np.mean(Cv_A_test))
# print("Gauss(M=3) SD={0:.2f} cm".format(mean_Gauss_test[1]))
print("Model A SD={0:.2f} cm".format(mean_A_test))
```

```
# SD = np.append(mean_Gauss_test[0:5], mean_A_test)
# M = range(6)
# label = ["M=2", "M=3", "M=4", "M=5", "M=6", "Model A"]
# plt.figure(figsize=(5, 3))
# plt.bar(M, SD, tick_label=label, align="center",
# facecolor="cornflowerblue")
# plt.show()
```

Model A SD=4.72 cm

6 Class(분류)

6.1 1차원 입력 2Class

회귀문제와 분류(class)문제

회귀문제는 목표 data가 연속된 수치, 분류문제에서 목표 data는 class.

분류 문제는 순서와 관계가 없고, 확률이 중요한 수학적 요소임.(예측의 불확실성이 존재함)

1차원 입력변수 x_n로 나타내고, 목표 변수 t_n으로 나타냄.

$$\mathbf{X} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}, \quad T = \begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{N-1} \end{bmatrix}$$

간단한 예제로, 무게에 의해 남,녀를 구분하는 문제를 생각해보자.

임시로 가공의 data를 만들어, 분석하는 방법을 진행.

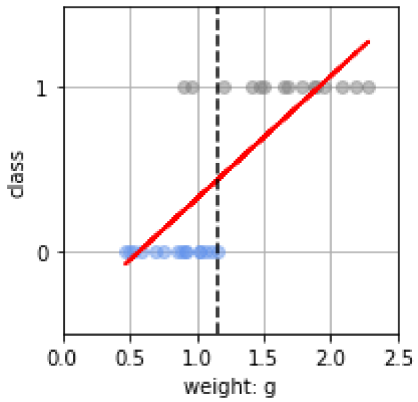
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
# 데이터 생성 ------------------------------
np.random.seed(seed=0) # 난수를 고정
X_min = 0
X_max = 2.5
X_n = 30
X_col = ['cornflowerblue', 'gray']
X = np.zeros(X_n) # 입력 데이터
T = np.zeros(X_n, dtype=np.uint8) # 목표 데이터
Dist_s = [0.4, 0.8] # 분포의 시작 지점
Dist_w = [0.8, 1.6] # 분포의 폭
Pi = 0.5 # 클래스 0의 비율

for n in range(X_n): #30개의 Data생성
    wk = np.random.rand()
    T[n] = 0 * (wk < Pi) + 1 * (wk >= Pi) # wk가 0.5보다 크면 1, 작으면 0,
    X[n] = np.random.rand() * Dist_w[T[n]] + Dist_s[T[n]] # 임의의 질량 생성(여, 남 ㄷ

print('X=' + str(np.round(X, 2)))
print('T=' + str(T))
```

```
X=[1.94 1.67 0.92 1.11 1.41 1.65 2.28 0.47 1.07 2.19 2.08 1.02 0.91 1.16
 1.46 1.02 0.85 0.89 1.79 1.89 0.75 0.9  1.87 0.5  0.69 1.5  0.96 0.53
 1.21 0.6 ]
T=[1 1 0 0 1 1 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0]
```

이 DATA를 표시해보면

In [12]:
```
def show_data1(x, t):
    K = np.max(t) + 1
    for k in range(K): # (A)
        plt.plot(x[t == k], t[t == k], X_col[k], alpha=0.5,
                 linestyle='none', marker='o') # (B)
        plt.plot(x, 0.75*x-0.43, color='red')
        plt.grid(True)
        plt.ylim(-.5, 1.5)
```

```
        plt.xlim(X_min, X_max)
        plt.yticks([0, 1])

#그림 그리기
fig = plt.figure(figsize=(3, 3))
show_data1(X, T)
plt.plot([1.15, 1.15], [-.5, 1.5], color='k', linestyle='--')
plt.xlabel('weight: g')
plt.ylabel('class')
plt.show()
```



이와 같은 분포의 data가 존재할때, 암컷과 수컷을 분리하는 경계선을 결정하는것이 목표인데,

이것을 특히 결정 경계라 하고, 이 결정 경계를 어떻게 결정하는것이 최선일까.

선형 회귀모델을 사용하는 방법

확률로 나타내는 분류

선형 model로 분류문제를 그대로 적용하는것은 오차를 해소하기위한 영향을 많이받아, 적당치 않음.

위 data의 그림에서 x<0.8g이면 암컷, x>1.2g이면 숫컷, 0.8g<x<1.2g일때는 암컷과 숫컷이 공존.

100%로 예측 불가능

0.8g<x<1.2g일때, 숫컷일 확률 1/3, 암컷일 확률 2/3

숫컷의 확률을 x의 범위에 따라 구분해 보면,

x<0.8g일때: 0

x>1.2g일때: 1

0.8g<x<1.2g일때: 1/3

이와같이 조건에 의한 확률을 표시하는 기호는: P(t=1|x)--> x의 변화에 따른 1일 확률을 의미

확률에 의한 방법으로 결정경계를 어떻게 나타내야 할까.

- 최대 가능도법

0.8<x<1.2범위에서, t의 결과값을 보면, 3회 t=0(암컷), 1회는 t=1(숫컷)

이 범위에서 P(t=1|x) = w 라면,

w는 0~1사이에 있고 T=0,0,0,1이라는 data를 생성했다고 가정

이를 통해 가장 타당한 w를 찾아내는것이 해법

쉽게 생각하면 총 4회에서 t=1이 1회 밖에 없기때문에 w=1/4(즉 0.25)이라고 생각.

T=0,0,0,1이 생성될 확률(즉 순서와 관계없이 0이 3회, 1이 1회 생성될 확률)--> 가능도

- 확률: 주어진 확률분포에서 해당 관측값이 나올 확률.
- 가능도: 주어진 관측값에서 이것이 해당 확률분포에서 나왔을 확률

확률 w=0.1일 경우, w=P(t=1|x)=0.1임으로 t=1일 확률 0.1, t=0일 확률 1-0.1 = 0.9에서의 가능도 =0.9 X 0.9 X 0.9 X 0.1 = 0.0729

확률 w=0.2일 경우 즉 2=P(t=1|x)=0.2임으로 t=1일 확률은 0.2, t=0일 확률 1-0.2=0.8에서 가능도 =0.8 X 0.8 X 0.8 X 0.2 = 0.1024

로 나타나는데,

정리해보면 w=0.1일때 가능도 0.0729, w=0.2일때 가능도 0.1024로 가능도가 높은 w=0.2쪽이 확률적으로 높다.

즉 가장 높은 가능도를 찾는것이 최대 가능도법.

P(t=1|x)=w(t=1이 될 확률), t=0이 될 확률은 (1-w)

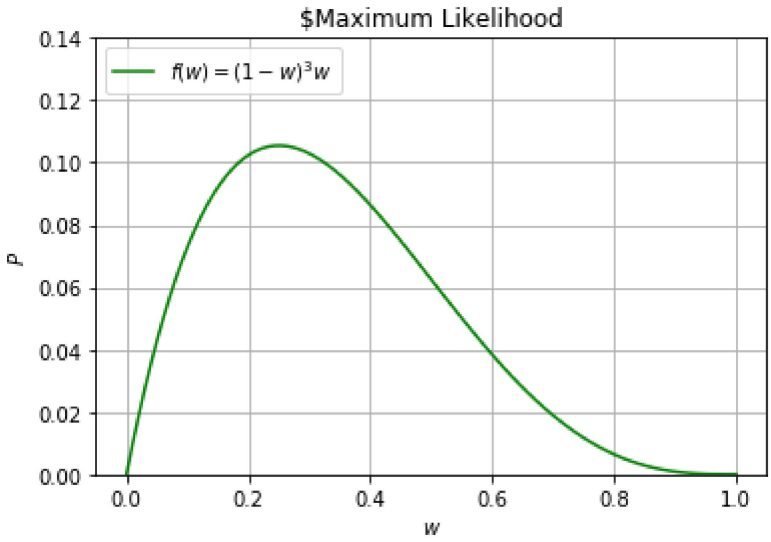
P(T=0,0,0,1|x)=$(1-w)^3w$

이것을 그래프로 그려보면이것을 그래프로 그려보면

In [3]:

```
#임의의 함수에 대한 display: f(x) = (x-1)(x+1)(x+3)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# def f(x, k):
#     return (x-k)*(x+1)*(x+k)

# x = np.linspace(-3, 3, 200) # -3 ~ 3사이를 200단계로 나누어

# plt.plot(x, f(x, 2), color = 'g', label='$k=2$')
# plt.plot(x, f(x, -1), color = 'red', label='$k=-1$')
w = np.linspace(0.0, 1.0, 100)
def f(w):
    return (1-w)**3*w

#color: r=빨강, b=파랑, g=녹색, c=시안, m=마젠타, y=노랑, k=검정, w=white
plt.plot(w, f(w), color = 'g', label='$f(w) = (1-w)^3 w$')
plt.legend(loc = 'upper left')
plt.ylim(-0.00, 0.14)
# plt.title('$f(x, k) = (x-k)(x+1)(x+k)$')
plt.title('$Maximum Likelihood')
plt.xlabel('$w$')
plt.ylabel('$P$')
plt.grid(True)
plt.show
```

Out[3]: <function matplotlib.pyplot.show(*args, **kw)>



최대가 되는 값을 해석적으로 찾아보기 위해 $(1-w)^3w$와 같이 연속된 곱셈을 다루는것은 매우 힘듦으로,

양변을 log를 취하여 덧셈형태로 바꾸어

logP = $log((1-w)^3w)=3log(1-w)+logw 로 나타낸다.

log는 단조 증가(즉 log안의 값이 증가하면 log결과 값이 증가함)함수임으로 P를 최대로 만드는 w는 logP를 최대로 만듬.

이를 log가능도라 하고, 최대화하는 매개 변수를 찾는것.

$$\frac{\partial}{\partial w} logP = \frac{\partial}{\partial w}[3log(1-w) + logw] = 0$$

$$3\frac{-1}{1-w} + \frac{1}{w} = \frac{-3w+1-w}{(1-w)w} = 0$$

$$-3w+1-w = 4w+1 = 0, \;\; w = \frac{1}{4}$$

이는 0.8<x<1.2의 범위헤서 확률은 일정하다는 가정으로부터 전개된것임.

실제 자연현상에서는 확률이 일정하게 유지하는 구간은 존재하지않는경우가 많음.

- 로지스틱 회귀 모델

로지스틱 회귀 모델은 $y = w_0x + w_1$를 시그모이드 함수$\sigma(x) = \frac{1}{1+e^{-x}}$에 적용하는것.

$$y = \sigma(w_0x + w_1) = \frac{1}{1+e^{-(w_0x+w_1)}}$$

이함수는 x가 $+\infty$로 변하면 1에 가까운값이 되고, $-\infty$로 변하면 0에 가까운 값으로 변환되어,

직선의 함수가 0~1의 범위내에 들어가서, 확률과 관련지어 생각할수있음.

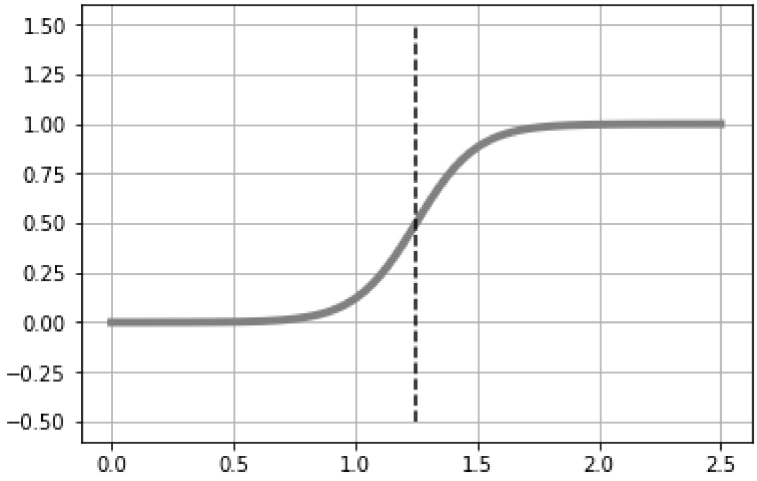
In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
def logistic(x, w):
    y=1/(1+np.exp(-(w[0]*x+w[1])))
    return y
```

In [5]:

```
def show_logistic(w):
    xb=np.linspace(X_min, X_max, 100)
    y=logistic(xb, w)
    plt.plot(xb, y, color='gray', linewidth=4)
    i=np.min(np.where(y>0.5))
    B=(xb[i-1]+xb[i])/2
    plt.plot([B, B], [-.5, 1.5], color='k', linestyle='--')
    plt.grid(True)
    return B

W=[8, -10]
show_logistic(W)
```

Out[5]: 1.25



- 교차 엔트로피 오차

로지스틱 회귀 모델을 통해 x가 t=1이 될 확률, y=$\sigma(w_0x+w_1)=P(t=1|x)$

매개변수 w_0, w_1이 곤충 데이터에 맞도록 최대가능도법을 적용함.

곤충의 데이터가 만들어질수 있는 가장 확률적으로 높은 매개 변수를 구하기 위함.

어떤 데이터에도 적용할수있는 가능도를 구한다.

곤충의 데이터가 이 모델에서 생성된 확률, 가능도를 구하자.

데이터가 하나뿐이라고 가정하고, 무게 x에 대해 t=1(숫컷)이라면, t=1이 모델에서 생성될 확률은 y값 자체이고, t=0이면 확률은 1-y

이를 수학적 형태로 나타내면 p(t|x) = $y^t(1-y)^{1-t}$

이를 실제상태에 적용해보면, t=1일때 $P(t=1|x)=y^1(1-y)^{1-1}$=y, t=0일때 $P(t=0|x)=y^0(1-y)^{1-0}$=1-y

데이터가 N개라면 X=$x_0,\cdots,x_{N-1}$에 대한 클래스 T=$t_0,\cdots,t_{N-1}$의 생성확률은 어떻게 될까?

하나하나의 데이터 생성확률은 모든 데이터에 대한 확률을 곱하면 됨으로

P(T|X) = $\prod_{n=0}^{N-1} P(t_n|x_n) = \prod_{n=0}^{N-1} y_n^{t_n}(1-y_n)^{1-t_n}$

이것에 LOG를 취하여 LOG가능도를 얻는데, 매개변수 w_0, w_1의 변화에 따른 log가능도가 최대가 되도록 구하면 됨.

logP(T|X)=$\sum_{n=0}^{N-1}(t_n log y_n + (1-t_n) log(1-y_n))$

선형회귀 모델에서 평균제곱오차가 최소가 되도록 매개변수(w_0, w_1)를 구했음으로, 동일한 방식으로 전개하기위해,

확률면에서는 logP(T|X)에 -1을 곱하는것이, 교차 엔트로피 오차(cross entropy error function)라고 함.

이것이 최소가 되는 매개 변수를 구하면 됨.

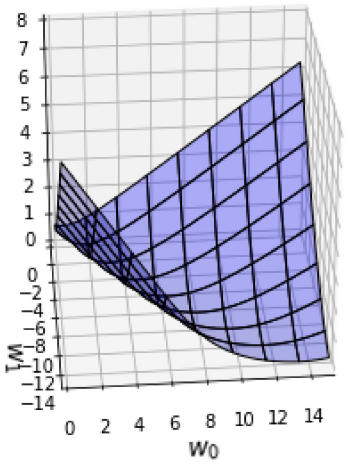
이를 N으로 나눈것을 평균 교차 엔트로피 오차: E(w)= -1$\frac{1}{N} log P(T|N)$= -1 $\frac{1}{N}\sum_{n=0}^{N-1}(t_n log y_n + (1-t_n) log(1-y_n))$

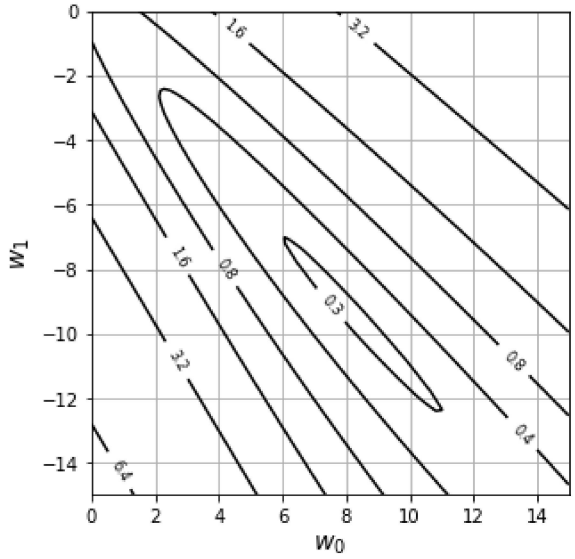
In [6]:
```
def cee_logistic(w, x, t):
    y=logistic(x, w)
    cee=0
    for n in range(len(y)):
        cee=cee-(t[n]*np.log(y[n])+(1-t[n])*np.log(1-y[n]))
    cee = cee/X_n
    return cee

W=[1, 1]
cee_logistic(W, X, T)
```

Out[6]: 1.0288191541851066

In [7]:
```
from mpl_toolkits.mplot3d import Axes3D


# 계산 ----------------------------------
xn = 80 # 등고선 표시 해상도
w_range = np.array([[0, 15], [-15, 0]]) #w0: 0~15, w1: -15~0
x0 = np.linspace(w_range[0, 0], w_range[0, 1], xn) #w0: 0, 0.18987342, ..... 15
x1 = np.linspace(w_range[1, 0], w_range[1, 1], xn) #w1: -15, -14.81012658, ..... 0
xx0, xx1 = np.meshgrid(x0, x1)
C = np.zeros((len(x1), len(x0))) #[[0,,,0], [0,,,,0]] 80x80
w = np.zeros(2) #w=[0, 0]
for i0 in range(xn):
    for i1 in range(xn):
        w[0] = x0[i0]
        w[1] = x1[i1]
        C[i1, i0] = cee_logistic(w, X, T)


# 표시 ----------------------------------
plt.figure(figsize=(12, 5))
#plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)
```

```
ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx0, xx1, C, color='blue', edgecolor='black',
                rstride=10, cstride=10, alpha=0.3)
ax.set_xlabel('$w_0$', fontsize=14)
ax.set_ylabel('$w_1$', fontsize=14)
ax.set_xlim(0, 15)
ax.set_ylim(-15, 0)
ax.set_zlim(0, 8)
ax.view_init(30, -95)


plt.subplot(1, 2, 2)
cont = plt.contour(xx0, xx1, C, 20, colors='black',
                   levels=[0.26, 0.4, 0.8, 1.6, 3.2, 6.4])
cont.clabel(fmt='%1.1f', fontsize=8)
plt.xlabel('$w_0$', fontsize=14)
plt.ylabel('$w_1$', fontsize=14)
plt.grid(True)
plt.show()
```





이 교차 엔트로피 오차를 최소화하는 매개변수해를 구하는데, 분석해는 구할수 없는데,

y_n이 비선형 시그모이드 함수를 포함하고 있어 수치해로 구해야 함.

경사하강법을 적용하는것은 위의 E(w)를 w_0, w_1에 대해 편미분을 수행해야 하는데,

각 Data n에 대해 $E_n(w) = -t_n log y_n - (1-t_n) log(1-y_n)$으로 표현하면,

미분과 합은 교환할 수있어

$$\frac{\partial}{\partial w_0} E(w) = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial}{\partial w_0} E_n(w)$$

먼저 $\frac{\partial E_n(w)}{\partial w_0}$을 구하고, 다음 단계를 진행한다.

$E_n(w)$는$E_n(y_n(a_n(w)))$로 중첩된 함수로 해석됨으로, w_0로 편미분하기 위해 연쇄법칙적용하면

$$\frac{\partial E_n}{\partial w_0} = \frac{\partial E_n}{\partial y_n} \frac{\partial y_n}{\partial a_n} \frac{\partial a_n}{\partial w_0}$$

$$\frac{\partial E_n}{\partial y_n} = \frac{\partial}{\partial y_n}(-t_n log y_n - (1-t_n) log(1-y_N)) = -t_n \frac{\partial}{\partial y_n} log y_n - (1-t_n)\frac{\partial}{\partial y_n} log(1-y_n) = + \frac{1-t_n}{1-y_n}$$

$$\frac{\partial y_n}{\partial a_n} = \frac{\partial}{\partial a_n}\sigma(a_n) = \sigma(a_n)(1-\sigma(a_n)) = y_n(1-y_n)$$

$$\frac{\partial a_n}{\partial w_0} = \frac{\partial}{\partial w_0}(w_0 x_n + w_1) = x_n$$

$$\frac{\partial E_n}{\partial w_0} = (-\frac{t_n}{y_n} + \frac{1-t_n}{1-y_n}) y_n(1-y_n)x_n = (-t_n(1-y_n) + (1-t_n)y_n)x_n = (y_n - t_n)x_n$$

결과적으로 $\frac{\partial E}{\partial w_0} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n$

w1에 대해서는, $\frac{\partial E}{\partial w_1} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)$

이와같은 결과를 함수로 구성하면 하기의 함수 프로그램으로 나타남.

In [8]:
```
def dcee_logistic(w, x, t):
    y = logistic(x, w)
    dcee = np.zeros(2)
    for n in range(len(y)):
        dcee[0] = dcee[0] + (y[n] - t[n]) * x[n]
        dcee[1] = dcee[1] + (y[n] - t[n])
    dcee = dcee / X_n
    return dcee


# --- test
W=[1, 1]
dcee_logistic(W, X, T)
```

Out[8]:
```
array([0.30857905, 0.39485474])
```

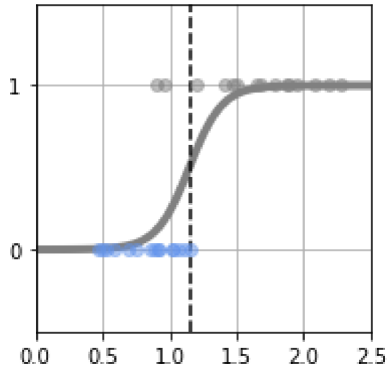
이것에 경사 하강법을 적용하여 최적의 매개변수(w_0, w_1)을 구하는 단계로 적용하는 함수는,

scipy.optimize라이브러리의 minimize()함수로 최저값을 알아냄.

In [9]:
```
from scipy.optimize import minimize


# 매개 변수 검색
def fit_logistic(w_init, x, t):
    res1 = minimize(cee_logistic, w_init, args=(x, t),
                    jac=dcee_logistic, method="CG") # (A)
    return res1.x


# 메인 ----------------------------------
plt.figure(1, figsize=(3, 3))
W_init=[1,-1]
W = fit_logistic(W_init, X, T)
print("w0 = {0:.2f}, w1 = {1:.2f}".format(W[0], W[1]))

B=show_logistic(W)
show_data1(X, T)
```

```
plt.ylim(-.5, 1.5)
plt.xlim(X_min, X_max)
cee = cee_logistic(W, X, T)
print("CEE = {0:.2f}".format(cee))
print("Boundary = {0:.2f} g".format(B))
plt.show()
```

```
w0 = 8.18, w1 = -9.38
CEE = 0.25
Boundary = 1.15 g
```



minimize()함수를 이용하여 훨씬 간단하고, 빠른 결과를 찾아낼수 있음.

결과로는 결정 경계는 1.15g이 되어, 직선 모델 제곱오차 최소화에서 구한 경계결정 1.24g보다 약간 왼쪽에 있음.

6.2 2차원 입력 data에 대한 class분류

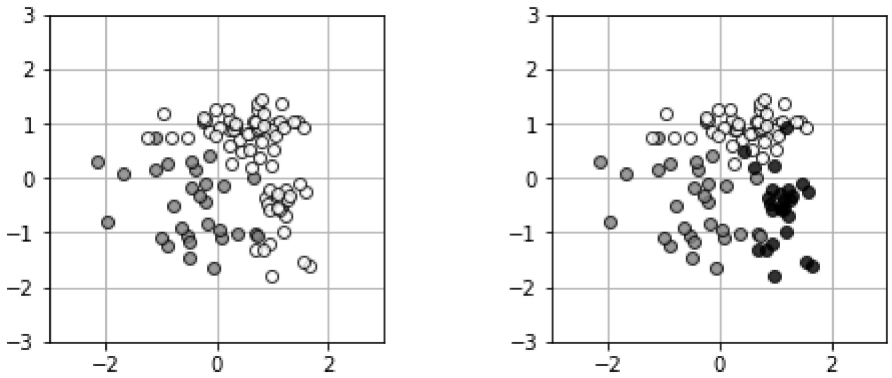
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


# 데이터 생성 ----------------------------------
np.random.seed(seed=1)  # 난수를 고정
N = 100 # 데이터의 수
K = 3 # 분포 수
T3 = np.zeros((N, 3), dtype=np.uint8)# 8bit data, 3차원
T2 = np.zeros((N, 2), dtype=np.uint8)# 8bit data, 2차원
X = np.zeros((N, 2)) #2차원 input
X_range0 = [-3, 3] # X0 범위 표시 용
X_range1 = [-3, 3] # X1의 범위 표시 용
Mu = np.array([[-.5, -.5], [.5, 1.0], [1, -.5]]) # 분포의 중심
Sig = np.array([[.7, .7], [.8, .3], [.3, .8]]) # 분포의 분산
Pi = np.array([0.4, 0.8, 1]) # (A) 각 분포에 대한 비율 0.4 0.8 1

for n in range(N):#input, output data생성
    wk = np.random.rand()#임의의 변수생서

    for k in range(K): # (B)
        if wk < Pi[k]: # Pi[k]가 임의로 생성된 wk보다 크면
            T3[n, k] = 1 #해당 output은 1
            break #아니면 나머지 T3[n, k] = 0으로 설정하고 빠져나감.


    for k in range(2): #아래의 공식에 따라, input data setting
        X[n, k] = (np.random.randn() * Sig[T3[n, :] == 1, k]
                   + Mu[T3[n, :] == 1, k])
T2[:, 0] = T3[:, 0] #2차원의 0차의 output은 3차원의 0차로 복사
T2[:, 1] = T3[:, 1] | T3[:, 2] #2차원의 1차의 output은 3차원의 1,2차의 OR 로직으로 설

print('------ X 5번째까지 -----')
print(X[:5, :])
print('------ T2 5번째까지 -----')
print(T2[:5, :])
print('------ T3 5번째까지 -----')
print(T3[:5, :])
```

```
------ X 5번째까지 -----
[[-0.14173827  0.86533666]
 [-0.86972023 -1.25107804]
 [-2.15442802  0.29474174]
 [ 0.75523128  0.92518889]
 [-1.10193462  0.74082534]]
------ T2 5번째까지 -----
[[0 1]
 [1 0]
 [1 0]
 [0 1]
 [1 0]]
------ T3 5번째까지 -----
[[0 1 0]
 [1 0 0]
 [1 0 0]
 [0 1 0]
 [1 0 0]]
```

이 입력과 출력을 그래프로 그려보면

In [2]:

```
def show_data2(x, t):
    wk, K = t.shape
    c = [[.5, .5, .5], [1, 1, 1], [0, 0, 0]] #Color 배열 setting
    for k in range(K):
        plt.plot(x[t[:, k] == 1, 0], x[t[:, k] == 1, 1], #k=0, k=1, k=2
                 linestyle='none', markeredgecolor='black',
                 marker='o', color=c[k], alpha=0.8)
        plt.grid(True)


# 메인 ----------------------------
plt.figure(figsize=(7.5, 3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1, 2, 1)
show_data2(X, T2)
plt.xlim(X_range0)
plt.ylim(X_range1)


plt.subplot(1, 2, 2)
show_data2(X, T3)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.show()
```



로지스틱 회귀 모델로 2차원 입력을 검토해 보면,

y=σ(a)

$a = w_0 x_0 + w_1 x_1 + w_2$

매개변수가 하나 증가 하였음

P(t=0|x), y는 클래스가 0인 확률, y=σ(a)=$\frac{1}{(1+e^{-a})}$

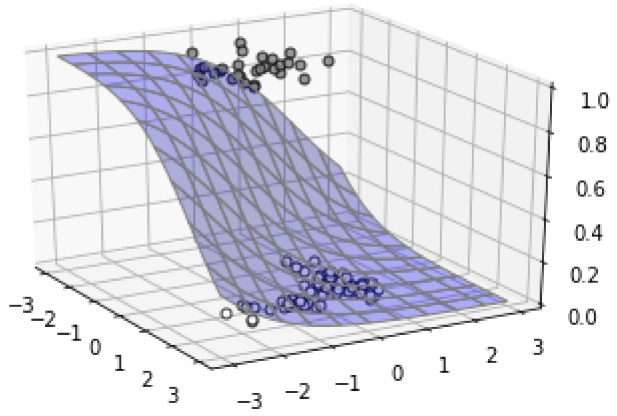
In [3]:

```
def logistic2(x0, x1, w):
    y = 1 / (1 + np.exp(-(w[0] * x0 + w[1] * x1 + w[2])))
    return y
```

In [4]:

```
from mpl_toolkits.mplot3d import axes3d


def show3d_logistic2(ax, w):
    xn = 50
    x0 = np.linspace(X_range0[0], X_range0[1], xn) #-3~3을 50개 간격으로
    x1 = np.linspace(X_range1[0], X_range1[1], xn) #-3~3을 50개 간격으로
    xx0, xx1 = np.meshgrid(x0, x1)
```

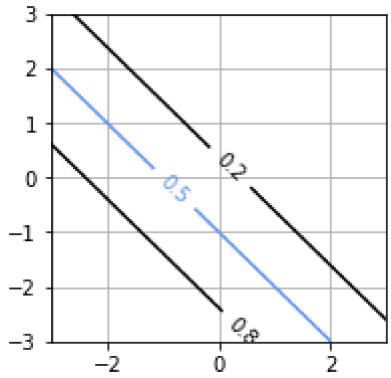
```
    y = logistic2(xx0, xx1, w) #확률 계산
    ax.plot_surface(xx0, xx1, y, color='blue', edgecolor='gray', #확률을 3D로 표시
                    rstride=5, cstride=5, alpha=0.3)


def show_data2_3d(ax, x, t):
    c = [[.5, .5, .5], [1, 1, 1]]
    for i in range(2):
        ax.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1], 1 - i, #2 output에 대해, 각 c
                marker='o', color=c[i], markeredgecolor='black',
                linestyle='none', markersize=5, alpha=0.8)
    Ax.view_init(elev=25, azim=-30)


# test ---
Ax = plt.subplot(1, 1, 1, projection='3d')
W=[-1, -1, -1] #매개 변수가 [-1,-1,-1]인 경우에
show3d_logistic2(Ax, W) #확률을 3d로 그림.
show_data2_3d(Ax,X,T2) #입, 출력 DATA를 그림 #입, 출력 DATA를 그림
```



2D로 등고선을 표시하여보면,

In [5]:

```
def show_contour_logistic2(w):
    xn = 50 # 파라미터의 분할 수
    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)
    y = logistic2(xx0, xx1, w)
    cont = plt.contour(xx0, xx1, y, levels=(0.2, 0.5, 0.8),
                       colors=['k', 'cornflowerblue', 'k'])
    cont.clabel(fmt='%1.1f', fontsize=10)
    plt.grid(True)


# test ---
plt.figure(figsize=(3,3))
W=[-1, -1, -1]
show_contour_logistic2(W)
```



이 모델의 교차 엔트로피 오차: E(w)

$$E(w) = -\frac{1}{N} log P(T|X) = -\frac{1}{N}\sum_{n=0}^{N-1}\{t_n log y_n + (1-t_n) log(1-y_n)\}$$

위의 식을 함수로 나타내면

In [6]:
```
def cee_logistic2(w, x, t):
    X_n = x.shape[0]
    y = logistic2(x[:, 0], x[:, 1], w)
    cee = 0
    for n in range(len(y)):
        cee = cee - (t[n, 0] * np.log(y[n]) +
                     (1 - t[n, 0]) * np.log(1 - y[n]))
    cee = cee / X_n
    return cee
```

E(w)에 대해, 매개변수에 따른 편미분을 구하면,

$$\frac{\partial E}{\partial w_0} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_0$$

$$\frac{\partial E}{\partial w_1} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_1$$

$$\frac{\partial E}{\partial w_2} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)$$

이 편미분에 대해 함수를 구하면

In [7]:
```
def dcee_logistic2(w, x, t):
    X_n=x.shape[0]
    y = logistic2(x[:, 0], x[:, 1], w)
    dcee = np.zeros(3)
    for n in range(len(y)):
        dcee[0] = dcee[0] + (y[n] - t[n, 0]) * x[n, 0]
        dcee[1] = dcee[1] + (y[n] - t[n, 0]) * x[n, 1]
        dcee[2] = dcee[2] + (y[n] - t[n, 0])
    dcee = dcee / X_n
    return dcee


# test ---
W=[-1, -1, -1]
dcee_logistic2(W, X, T2)
```

```
array([ 0.10272008,  0.04450983, -0.06307245])
```

Out[7]:

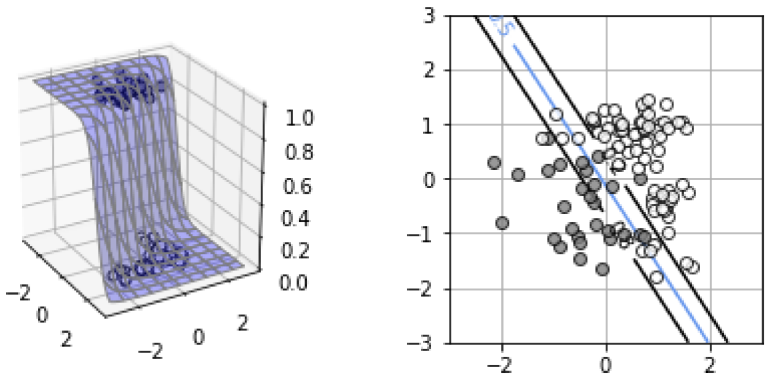
이값은, W=[-1, -1, -1]에서의 편미분 결과값

결국은 이 평균 교차 엔트로피 오차가 최소가 되는 매개변수를 구하는것인데,

In [8]:

```
from scipy.optimize import minimize


# 로지스틱 회귀 모델의 매개 변수 검색 -
def fit_logistic2(w_init, x, t):
    res = minimize(cee_logistic2, w_init, args=(x, t),
                   jac=dcee_logistic2, method="CG")
    return res.x


# 메인 ----------------------------------
plt.figure(1, figsize=(7, 3))
plt.subplots_adjust(wspace=0.5)


Ax = plt.subplot(1, 2, 1, projection='3d')
W_init = [-1, 0, 0]
W = fit_logistic2(W_init, X, T2)
print("w0 = {0:.2f}, w1 = {1:.2f}, w2 = {2:.2f}".format(W[0], W[1], W[2]))
show3d_logistic2(Ax, W) #data에 따른 확률 graph


show_data2_3d(Ax, X, T2) #data dot graph
cee = cee_logistic2(W, X, T2)
print("CEE = {0:.2f}".format(cee))


Ax = plt.subplot(1, 2, 2)
show_data2(X, T2) #Contour dot 그래프
show_contour_logistic2(W) #Contour 그래프
plt.show()
```

```
w0 = -3.70, w1 = -2.54, w2 = -0.28
CEE = 0.22
```



minimize()함수에 미분 함수 전달하고, 켤레기울기법으로 매개변수를 구함.

6.3 2차원 입력 3차원 분류(Class)

$$y_0 = \frac{e^{a_0}}{\sum_{k=0}^{K-1} e^{(a_k)}} = P(t = 0 | x)$$

$$y_1 = \frac{e^{a_1}}{\sum_{k=0}^{K-1} e^{(a_k)}} = P(t=1|x)$$

$$y_2 = \frac{e^{a_2}}{\sum_{k=0}^{K-1} e^{(a_k)}} = P(t=2|x)$$

$a_k = w_{k0}x_0 + w_{k1}x_1 + w_{k2} (k = 0, 1, 2)$

w_{ki}는 입력 x_i에서 분류(class) k의 입력 총합을 조절하는 매개 변수.

$$u = \sum_{k=0}^{K-1} e^{a_k}$$

$y_k = \frac{e^{a_k}}{u}$ (k=0,1,2)

입력 $x = [x_0, x_1, x_2]$인데, x_2는 항상 1.

출력 $y = [y_0, y_1, y_2]$로 $y_0 + y_1 + y_2 = 1$이 된다

모델의 매개 변수 w_{ki}는

$$W = \begin{bmatrix} W_{00} & W_{10} & W_{20} \\ W_{01} & W_{11} & W_{21} \\ W_{02} & W_{12} & W_{22} \end{bmatrix}$$

이 모델의 출력 y_0, y_1, y_2는 각 입력 x가 속할 확률

P(T=[1,0,0]|x)(class 0), P(T=[0,1,0]|x)(class 1), P(T=[0,0,1]|x)(class 2)를 나타내도록 학습

3 class용 로지스틱 회귀모델을 함수로 나타내면,

test에서는 3개의 $X[:3, 0]$ 과 $X[:3, 1]$쌍과 임시로 정한 W(매개변수의 조합)으로 실행.

In [9]:
```
def logistic3(x0, x1, w):
    K = 3
    w = w.reshape((3, 3))
    n = len(x1)
    y = np.zeros((n, K))
    for k in range(K):
        y[:, k] = np.exp(w[k, 0] * x0 + w[k, 1] * x1 + w[k, 2])
    wk = np.sum(y, axis=1)
    wk = y.T / wk
    y = wk.T
    return y


# test ---
print('------ test 결과 -----')
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9])
y = logistic3(X[:3, 0], X[:3, 1], W)
print(np.round(y, 3))
```

```
------ test 결과 -----
[[0.    0.006 0.994]
 [0.965 0.033 0.001]
 [0.925 0.07  0.005]]
```

첫 line은 x_0에 대한 class 0, 1, 2일 확률

두번째 line은 x_1에 대한 class 0, 1, 2일 확률

세번째 line은 x_2에 대한 class 0, 1, 2일 확률

교차 엔트로로피 오차

$P(T=[1,0,0]|X)=y_0$

$P(T=[0,1,0]|X)=y_1$

$P(T=[0,0,1]|X)=y_2$를 일반화하여, 한 입력 x에 대한 확률식을 나타내면

$P(T|X)=y_0^{t_0}y_1^{t_1}y_2^{t_2}$

이 식에대해, $P(T=[0,1,0]|X)=y_0^0y_1^1y_2^0=y_1$

모든 N개의 data가 생성될 확률은 모든 data에 곱하면 되는데,

$$P(T|X)=\prod_{n=0}^{N-1}P(t_n|x_n)=\prod_{n=0}^{N-1}y_{n0}^{t_{n0}}y_{n1}^{t_{n1}}y_{n2}^{t_{n2}}=\prod_{n=0}^{N-1}\prod_{k=0}^{K-1}y_{nk}^{t_{nk}}$$

이에대한 평균 교차 엔트로피 오차: E(W)

$$E(W)=-\frac{1}{N}logP(T|X)=-\frac{1}{N}\sum_{n=0}^{N-1}P(t_n|x_n)=-\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1}t_{nk}logy_{nk}$$

이를 함수로 나타내면

In [10]:
```
def cee_logistic3(w, x, t):
    X_n = x.shape[0]
    y = logistic3(x[:, 0], x[:, 1], w)
    cee = 0
    N, K = y.shape
    for n in range(N):
        for k in range(K):
            cee = cee - (t[n, k] * np.log(y[n, k]))
    cee = cee / X_n
    return cee


# test ----
print('------- test ------')
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9]) #test용 매개변수
cee_logistic3(W, X, T3)
```

```
------- test ------
```

Out[10]: 3.9824582404787288

경사 하강법에 위한 E(W)를 최소화 하는 W를 구하기 위해 E(W)를 각 w_{ki}에 대한 편미분을 표현하면

$$\frac{\partial E}{\partial w_{ki}}=\frac{1}{N}\sum_{n=0}^{N-1}(y_{nk}-t_{nk})x_i$$

In [11]:
```
test_dcee=0.0

def dcee_logistic3(w, x, t):
    X_n = x.shape[0]
    y = logistic3(x[:, 0], x[:, 1], w)
    dcee = np.zeros((3, 3)) # (클래스의 수 K) x (x의 차원 D+1)
    N, K = y.shape
    for n in range(N):
        for k in range(K):
            dcee[k, :] = dcee[k, :] - (t[n, k] - y[n, k])* np.r_[x[n, :], 1]
    dcee = dcee / X_n
    test_dcee = dcee
    return dcee.reshape(-1)


# test ----
print('------test:각 w에 대한 편미분 결과 --------')
print('N:{0}, K:{1}'.format(N,K))
W = np.array([1, 2, 3, 4 ,5, 6, 7, 8, 9])
result=dcee_logistic3(W, X, T3)
print(test_dcee)
print(result)
```

```
------test:각 w에 대한 편미분 결과 --------
N:100, K:3
0.0
[ 0.03778433  0.03708109 -0.1841851  -0.21235188 -0.44408101 -0.38340835
  0.17456754  0.40699992  0.56759346]
```

minimize()를 이용하여, 최소의 매개변수 검색수행함수를 만들면,

In [12]:
```
def fit_logistic3(w_init, x, t):
    res = minimize(cee_logistic3, w_init, args=(x, t),
                   jac=dcee_logistic3, method="CG")
    return res.x
```

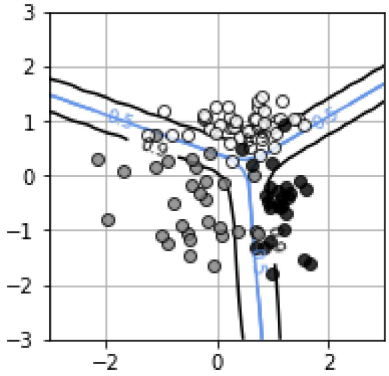
contour 결과를 표시하는 함수

In [13]:
```
def show_contour_logistic3(w):
    xn = 30 # 파라미터의 분할 수
    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)
    y = np.zeros((xn, xn, 3))
    for i in range(xn):
        wk = logistic3(xx0[:, i], xx1[:, i], w)
        for j in range(3):
            y[:, i, j] = wk[:, j]
    for j in range(3):
        cont = plt.contour(xx0, xx1, y[:, :, j],
                           levels=(0.5, 0.9),
                           colors=['cornflowerblue', 'k'])
        cont.clabel(fmt='%1.1f', fontsize=9)
    plt.grid(True)
```

In [14]:
```
W_init = np.zeros((3, 3))
W = fit_logistic3(W_init, X, T3)
print(np.round(W.reshape((3, 3)),2))
cee = cee_logistic3(W, X, T3)
print("CEE = {0:.2f}".format(cee))
```

```
plt.figure(figsize=(3, 3))
show_data2(X, T3)
show_contour_logistic3(W)
plt.show()
```

```
[[-3.2  -2.69  2.25]
 [-0.49  4.8  -0.69]
 [ 3.68 -2.11 -1.56]]
CEE = 0.23
```



이러한 다중 클래스 로지스틱 회귀 모델은 클래스간 경계선이 직선 조합으로 구성됨.

이모델은 모호성을 조건부 확률로 근사하는것이 장점임.

7 뉴런

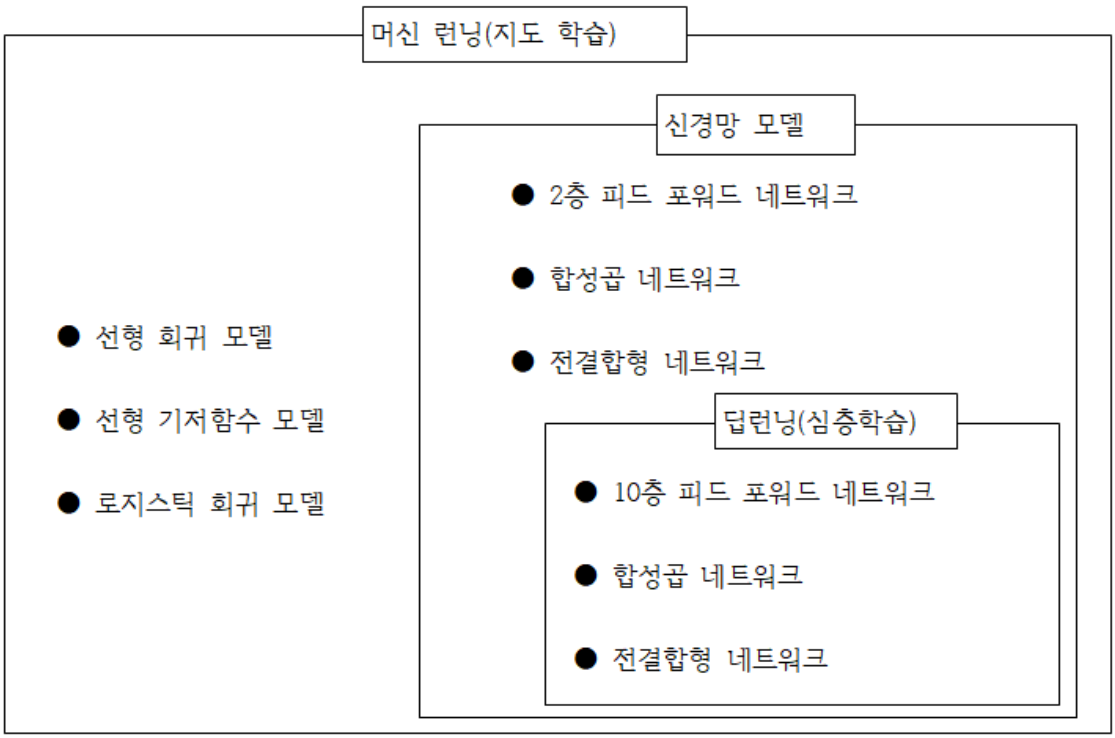
7.1 딥런닝

딥런닝은 머신런닝의 한가지 방법이고,

신경망 모델이라고 불리기도 하는데, 뇌의 신경 네트워크 모델을 묘사한 알고리즘.

층을 많이 이용한다고해서 딥 런닝이라고 함.

딥 런닝은 음성 인식, 이미지 인식등에 많이 활용되고 있음.



뉴런 모델(신경 세포 모델)

축삭: 케이블(신경 세포간 전기 펄스 연결선)

시냅스: 케이블과 신경세포 연결 단자

신경세포는 다른 세포로부터 전기적 펄스를 받으면, 세포의 전기적 레벨이 오르락 내리락함.

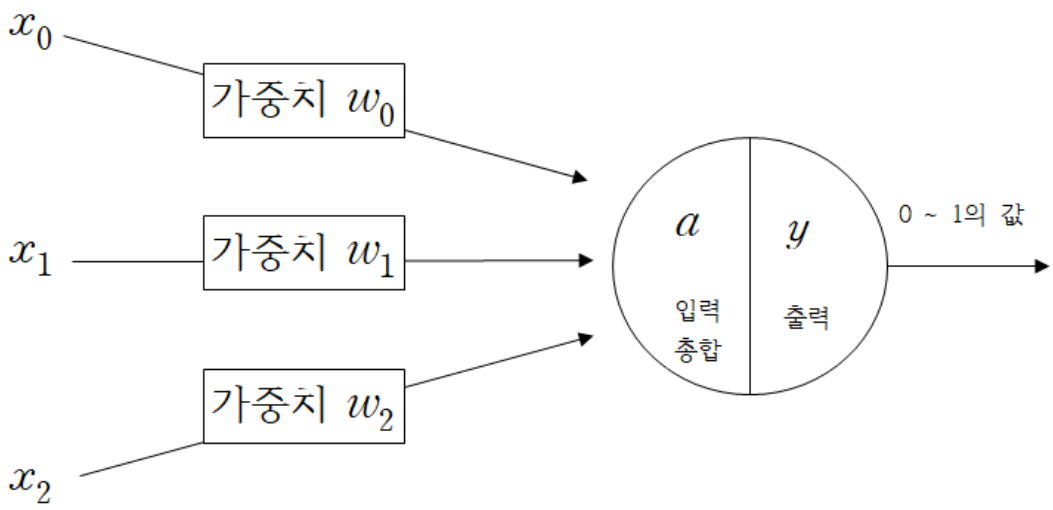
시냅스에는 몇가지 종류가 있는데, 그종류에 따라 전기적 레벨을 올리는 방향, 내리는 방향인지 정해짐.

올리거나 내리는 양은 시냅스의 상태에따라 달라짐.

세포막 전기적 레벨이 한계를 초과하면 전기적 펄스를 발신하고,

그 펄스는 축삭을 타고 다른 신경세포에 전달.

이런 신경세포의 움직임을 단순화한 수학적 모델이 뉴런 모델



$$a = w_0x_0 + w_1x_1 + w_2x_2$$

$$y = \sigma(a) = 1/(1+e^{(-a)})$$

뉴런 모델의 수학적 표현

input: x = $x_0, x_1, x_2(1)$, 여기서 특히 X_2는 1의 상수가 할당됨.

$$y = \sigma(a) = \frac{1}{1+e^{-a}}$$

$a = w_0x_0 + w_1x_1 + w_2x_2$

input은 양수와 음수값을 갖을수있는 실수.

여기서 전달 강도는 w_0, w_1로 x_2는 dummy input:1임,

결국 입력 총합, a=$w_0x_0 + w_1x_1 + w_2$가 a(세포막 시냅스)

$$a = \sum_{i=0}^{2} w_ix_i$$

$y = \frac{1}{1+e^{-a}}$, y: 0~1사이의 연속된 값.

a가 커질수록 발화 확률이 100%에 가깝고,

a가 음의 값으로 커질수록(적어 질수록) 발화 확률이 0%에 가까워짐. 즉 발화하지 않음.

이는 로지스틱 회귀모델,

x_0, x_1의 입력 변수 총합은 평면의 형태로 나타날수있고,

출력은 이 평면을 시그모이드 함수에 적용시켜, 범위 0~1사이로 나타나게 함.

입력 총합: a가 0인것은, 출력 y=$\frac{1}{1+1}$=0.5 이다

입력 차수에 2대신, 일반화를 위해, D를 적용한 입력 총합 a=$\sum_{i=0}^{D} w_i x_i, x_D = 1$

N개의 data set(x_n, t_n)에 대한 뉴런 모델의 학습 방법을 진행해보자

목표로하는 목적함수

$$E(w) = -\frac{1}{N}\sum_{n=0}^{N-1}\{t_n ln y_n + (1-t_n) ln(1-y_n)\}$$

이 오차 함수의 매개변수 기울기:

$$\frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_{ni}$$

매개변수 학습법칙은 그 기울기를 사용하면, 다음식으로 표현됨.

$$w_i(t+1) = w_i(t) - \alpha\frac{\partial E}{\partial w_i}$$

뉴런 모델은 입력 공간을 선으로 나눈다는 기능인데, 이를 많이 조합하면 강력한 힘이 발휘됨.

이런 뉴런의 집합체 모델을 신경망 모델이라함.

신호가 되돌아가는경로 없는, 한 방향으로만 흐르는 '피드 포워드 신경망'

2층 피드 포워드 신경망

2층 피드 포워드는, 2단의 뉴런층이 있다고 가정하는것.

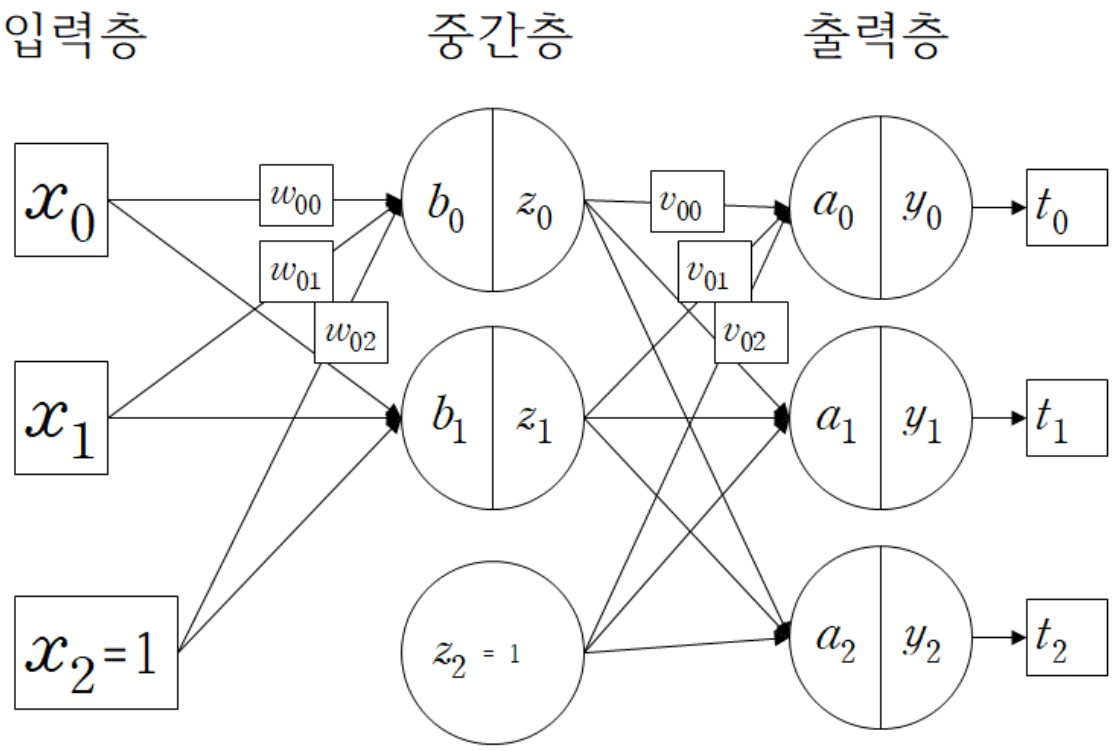
2차원의 수치를 3개의 카테고리로 분류

각각의 출력 뉴런의 출력값이 각각의 카테고리에 속하는 확률을 나타내도록 학습.

이 네트워크를 수식으로 전개 하면,

1층의 입력 총합 b:

$b_j = \sum_{i=0}^{2} w_{ji} x_i$로 나타내고



1층 뉴런의 출력 z:

$z_j = h(b_j)$는 $z_j = \sigma(b_j)$(시그모이드함수)로 할수있으나, 일반화함.

여기서 h( )나 $\sigma()$는 입력 총합에서 출력을 결정하는 함수로 활성화 함수라고 함.

2층의 입력 총합 a:

$a_k = \sum_{j=0}^{2} v_{kj} z_j$로 나타내고

2층 뉴런의 출력 y:

$y_k = \frac{e^{(a_k)}}{\sum_{i=0}^{2} e^{(a_i)}} = \frac{e^{(a_k)}}{u}, \; u = \sum_{i=0}^{2} e^{a_i}$(소프트맥스함수)

소프트맥스 함수를 사용했음으로 최종 출력 y:

$y = y_0 + y_1 + y_2 = 1$로 되어 확률의 의미를 갖음

신경망(소프트맥스)평균 교차 엔트로피 오차 함수

P(T|X) = $\prod_{n=0}^{N-1} P(t_n|x_n) = \prod_{n=0}^{N-1} y_{n0}^{t_{n0}} y_{n1}^{t_{n1}} y_{n2}^{t_{n2}} = \prod_{n=0}^{N-1} \prod_{k=0}^{K-1} y_{nk}^{t_{nk}}$인 경우

$$E(w) = -\frac{1}{N} log P(T|X) = -\frac{1}{N} \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} t_{nk} log y_{nk}$$

정리

중간층의 입력 총합: $b_j = \sum_{i=0}^{D} w_{ji} x_i$

중간층의 출력: $z_j = h(b_j)$

출력층의 입력 총합: $a_k = \sum_{i=0}^{M} v_{kj} z_j$

출력층의 출력: $y_k = \frac{e^{(a_k)}}{\sum_{i=0}^{K-1} e^{(a_i)}} = \frac{e^{(a_k)}}{u}$

x_D, z_M은 항상 1의 dummy값, 즉 더미 입력과 더미 뉴런.

구현

data 생성 생성

In [1]:
```
import numpy as np
# 데이터 생성 ------------------------------
np.random.seed(seed=1) # 난수를 고정
N = 200 # 데이터의 수
K = 3 # 분포의 수
T = np.zeros((N, 3), dtype=np.uint8)
X = np.zeros((N, 2))
X_range0 = [-3, 3] # X0의 범위, 표시용
X_range1 = [-3, 3] # X1의 범위, 표시용
Mu = np.array([[-.5, -.5], [.5, 1.0], [1, -.5]]) # 분포의 중심
Sig = np.array([[.7, .7], [.8, .3], [.3, .8]]) # 분포의 분산
Pi = np.array([0.4, 0.8, 1]) # 각 분포에 대한 비율
for n in range(N):
    wk = np.random.rand()
    for k in range(K):
        if wk < Pi[k]:
            T[n, k] = 1
            break
    for k in range(2):
        X[n, k] = np.random.randn() * Sig[T[n, :] == 1, k] + W
        Mu[T[n, :] == 1, k]
```

data를 훈련 data와 test data로 분할

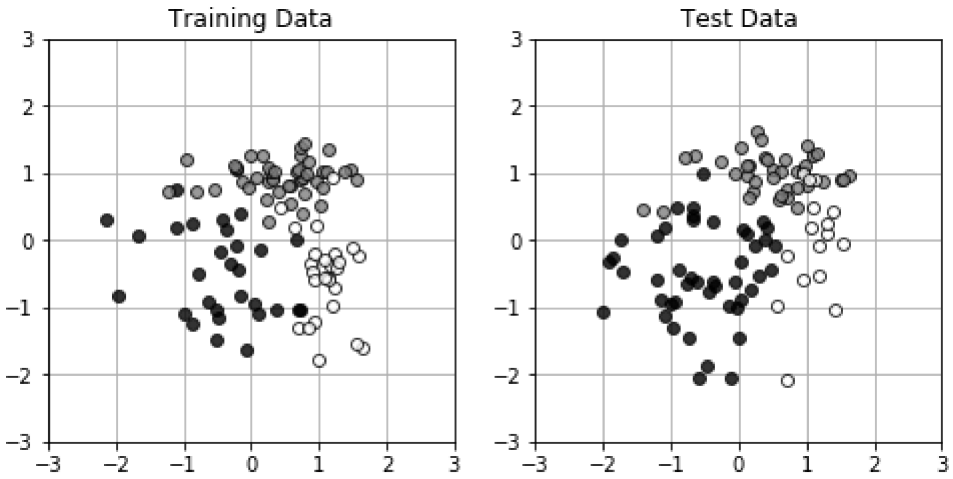
In [2]:
```
TestRatio = 0.5
X_n_training = int(N * TestRatio)
X_train = X[:X_n_training, :]
X_test = X[X_n_training:, :]
T_train = T[:X_n_training, :]
T_test = T[X_n_training:, :]


# -------- 데이터를 'class_data.npz'에 저장
```

```
np.savez('class_data.npz', X_train=X_train, T_train=T_train,
         X_test=X_test, T_test=T_test,
         X_range0=X_range0, X_range1=X_range1)
```

data를 그려 보면

In [3]:

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


# 데이터를 그리기 ----------------------------
def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                 linestyle='none',
                 marker='o', markeredgecolor='black',
                 color=c[i], alpha=0.8)
    plt.grid(True)


# 메인 ---------------------------------
plt.figure(1, figsize=(8, 3.7))
plt.subplot(1, 2, 1)
Show_data(X_train, T_train)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Training Data')

plt.subplot(1, 2, 2)
Show_data(X_test, T_test)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Test Data')
plt.show()
```



2층의 피드 포워드 신경망 함수: Feed Forward Neuron Network(FFNN)

입력 x의 data수: N x D(N개의 data set, D차원)

출력 y의 data수: N x K(N개의 data set, K 분류)

중간층의 매개변수(가중치) W: M x (D+1)의 행렬

출력층의 매개변수(가중치) V: K x (M+1)의 행렬

W와 V의 정보는 wv로 표시하는 벡터:

$$W = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix}$$

$$V = \begin{bmatrix} 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}$$

이에 대한 array는 wv= np.array([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14])

이는 매개변수를 한곳에 모아두어, 최적화 프로그램을 만들기 쉽다.

FNN의 프로그램 코드는

In [4]:
```
def Sigmoid(x):
    y = 1 / (1 + np.exp(-x))
    return y


# 네트워크 ------------------------
def FNN(wv, M, K, x):
    N, D = x.shape # 입력 차원
    w = wv[:M * (D + 1)] # 중간층 뉴런의 가중치
    w = w.reshape(M, (D + 1))
    v = wv[M * (D + 1):] # 출력층 뉴런의 가중치
    v = v.reshape((K, M + 1))
    b = np.zeros((N, M + 1)) # 중간층 뉴런의 입력 총합
    z = np.zeros((N, M + 1)) # 중간층 뉴런의 출력
    a = np.zeros((N, K)) # 출력층 뉴런의 입력 총합
    y = np.zeros((N, K)) # 출력층 뉴런의 출력
    for n in range(N):
        # 중간층의 계산
        for m in range(M):
            b[n, m] = np.dot(w[m, :], np.r_[x[n, :], 1]) # (A)
            z[n, m] = Sigmoid(b[n, m])
        # 출력층의 계산
        z[n, M] = 1 # 더미 뉴런
        wkz = 0
        for k in range(K):
            a[n, k] = np.dot(v[k, :], z[n, :])
            wkz = wkz + np.exp(a[n, k])
        for k in range(K):
            y[n, k] = np.exp(a[n, k]) / wkz
    return y, a, z, b


# test ---
WV = np.ones(15)
M = 2
K = 3
FNN(WV, M, K, X_train[:2, :])
```

Out[4]:
```
(array([[0.33333333, 0.33333333, 0.33333333],
        [0.33333333, 0.33333333, 0.33333333]]),
 array([[2.6971835 , 2.6971835 , 2.6971835 ],
        [1.49172649, 1.49172649, 1.49172649]]),
 array([[0.84859175, 0.84859175, 1.        ],
        [0.24586324, 0.24586324, 1.        ]]),
 array([[ 1.72359839,  1.72359839,  0.        ],
        [-1.12079826, -1.12079826,  0.        ]]))
```

수치 미분법

분류 문제에 대해, 호차함수는, 평균 교차 엔트로피 오차:E(w,v)

$$E(w,v) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} t_{nk} log(y_{nk})$$

이 평균 교차 엔트로피 오차를 CE_FNN함수로 구현로 구현

In [5]:
```
def CE_FNN(wv, M, K, x, t):
    N, D = x.shape
    y, a, z, b = FNN(wv, M, K, x)
    ce = -np.dot(np.log(y.reshape(-1)), t.reshape(-1)) / N
    return ce


# test ---
WV = np.ones(15)
M = 2
K = 3
CE_FNN(WV, M, K, X_train[:2, :], T_train[:2, :])
```

Out[5]: 1.0986122886681098

위의 함수를 보면 매개변수 w와 v를 일체화 시켜, wv를 사용함.

추후에도 이와 같이 적용함으로 잘 이해해 두어야 함.

또한 CE_FNN함수에는 입력 data x, 목표 data t, 네트워크 크기 M과 K를 입력.

FNN이 x에 대한 y를 출력하고, y와 t를 비교하여 크로스 엔트로피가 계산됨.

경사 하강법을 적용하기 위해서 오차 함수를 매개변수로 편미분이 필요,

이를 수치적 미분을 이용하여 값을 구할수 있음

$$\frac{\partial E}{\partial W}|_w \cong \frac{E(w+\epsilon) - E(w-\epsilon)}{2\epsilon}$$

여러개의 매개변수 w_0, w_1, w_2인 경우에는

$$\frac{\partial E}{\partial w_0}|_{w_0,w_1,w_2} \cong \frac{E(w_0+\epsilon, w_1, w_2) - E(w_0-\epsilon, w_1, w_2)}{2\epsilon}$$

w_1, w_2에 대해서도 마찬가지로 해당 매개변수외의 매개 변수를 고정하여 편미분 수행.

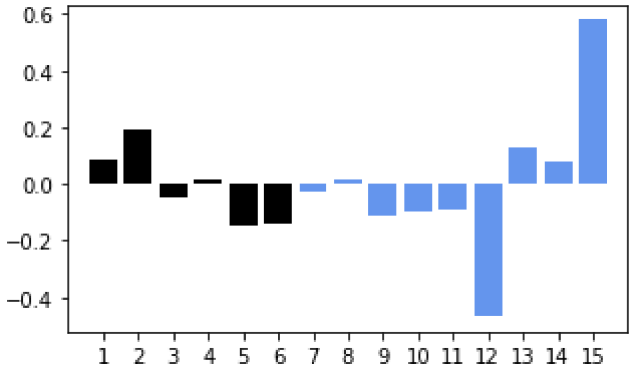
ϵ을 충분히 작게하여, 근사치에 가깝게 구할수 있으나, 많은 계산량과 시간이 소요됨

이러한 CE_FNN의 수치 미분을 출력하는 함수 dCE_FNN_num을 만들면.

In [6]:
```
def dCE_FNN_num(wv, M, K, x, t):
    epsilon = 0.001
    dwv = np.zeros_like(wv)
    for iwv in range(len(wv)):
        wv_modified = wv.copy()
        wv_modified[iwv] = wv[iwv] - epsilon
        mse1 = CE_FNN(wv_modified, M, K, x, t)
        wv_modified[iwv] = wv[iwv] + epsilon
```

```
        mse2 = CE_FNN(wv_modified, M, K, x, t)
        dwv[iwv] = (mse2 - mse1) / (2 * epsilon)
    return dwv


#--dVW의 표시 ------------------
def Show_WV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M * 3 + 1), wv[:M * 3], align="center", color='black')
    plt.bar(range(M * 3 + 1, N + 1), wv[M * 3:],
            align="center", color='cornflowerblue')
    plt.xticks(range(1, N + 1))
    plt.xlim(0, N + 1)


#-test----
M = 2
K = 3
nWV = M * 3 + K * (M + 1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)
dWV = dCE_FNN_num(WV, M, K, X_train[:2, :], T_train[:2, :])
print(dWV)
plt.figure(1, figsize=(5, 3))
Show_WV(dWV, M)
plt.show()
```

```
[ 0.0884813   0.19157999 -0.05139799  0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115648 -0.10104422 -0.09427964 -0.46855603
  0.13096434  0.08076649  0.57971252]
```



이 함수로 분류문제를 경사 하강법으로 푸는 함수: Fit_FNN_num이라 하고,

In [8]:

```
import time


# 수치 미분을 사용한 구배법 -------
def Fit_FNN_num(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha):
    wvt = wv_init
    err_train = np.zeros(n)
    err_test = np.zeros(n)
    wv_hist = np.zeros((n, len(wv_init)))
    epsilon = 0.001
    for i in range(n): # (A)
        wvt = wvt - alpha * dCE_FNN_num(wvt, M, K, x_train, t_train)
        err_train[i] = CE_FNN(wvt, M, K, x_train, t_train)
        err_test[i] = CE_FNN(wvt, M, K, x_test, t_test)
        wv_hist[i, :] = wvt
    return wvt, wv_hist, err_train, err_test


# 메인 --------------------------
```

```
startTime = time.time()
M = 2
K = 3
np.random.seed(1)
WV_init = np.random.normal(0, 0.01, M * 3 + K * (M + 1))
N_step = 1000 # (B) 학습 단계
alpha = 0.5
WV, WV_hist, Err_train, Err_test = Fit_FNN_num(
    WV_init, M, K, X_train, T_train, X_test, T_test, N_step, alpha)
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

```
Calculation time:654.243 sec
```

In [10]:

```
print('WV_init:{0}'.format(WV_init))
```

```
WV_init:[ 0.01624345 -0.00611756 -0.00528172 -0.01072969  0.00865408 -0.02301539
  0.01744812 -0.00761207  0.00319039 -0.0024937   0.01462108 -0.02060141
 -0.00322417 -0.00384054  0.01133769]
```

이 함수를 적용하기위해, 가중치(w,v)의 초기값으로 wv_init, 훈련 data, test data를 넣어

학습 단계별로 test data의 오차도 확인하여, 오버 피팅이 일어나지 않도록 확인.

test data의 정보는 매개변수의 학습에는 사용하지 않음.

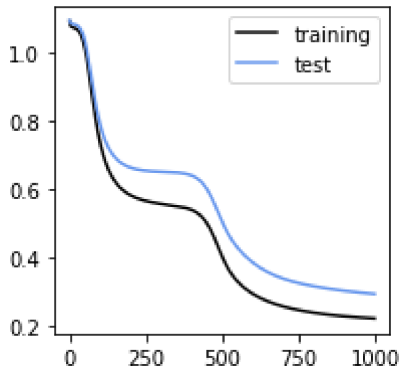
Fint_FNN_num(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha)에서,

n은 학습단계, alpha는 학습 상수, 출력은 최적화된 매개변수인 wvt

학습 단계에 의한 오차 함수의 변화

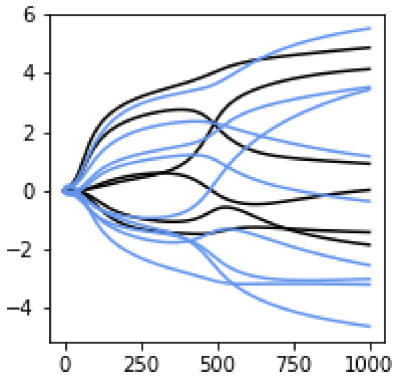
In [11]:

```
plt.figure(1, figsize=(3, 3))
plt.plot(Err_train, 'black', label='training')
plt.plot(Err_test, 'cornflowerblue', label ='test')
plt.legend()
plt.show()
```



In [12]:

```
plt.figure(1, figsize=(3, 3))
plt.plot(WV_hist[:, :M * 3], 'black')
plt.plot(WV_hist[:, M * 3:], 'cornflowerblue')
plt.show()
```



결과는 단계가 증가할수록 어떠한 값에 수렴해 감.

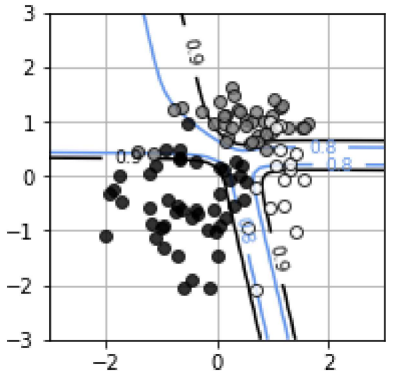
비선형성이 강한 신경망의 경우, 학습이 수렴해도 멈추지않고 더 노력하면,

학습이 빠르게 진행되는 단계가 있다.

이 data공간을 클래스0,1,2로 판정하는 영역의 경계를 표시하면계를 표시하면

In [13]:

```
def show_FNN(wv, M, K):
    xn = 60 # 등고선 표시 해상도
    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, 1), np.reshape(xx1, xn * xn, 1)]
    y, a, z, b = FNN(wv, M, K, x)
    plt.figure(1, figsize=(4, 4))
    for ic in range(K):
        f = y[:, ic]
        f = f.reshape(xn, xn)
        f = f.T
        cont = plt.contour(xx0, xx1, f, levels=[0.8, 0.9],
                           colors=['cornflowerblue', 'black'])
        cont.clabel(fmt='%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)


# 경계선 표시 -------------------------
plt.figure(1, figsize=(3, 3))
Show_data(X_test, T_test)
show_FNN(WV, M, K)
plt.show()
```



7.2 오차 역전파법

raspberry pi 4에서 실행 시켜야함

피드 포워드 신경망에 학습시키는 방법으로는 오차 역전파법(Backpropagation)이 있음.

네트워크의 출력에서 발생하는 오차의 정보를 사용,

출력층의 가중치 v_{kj}에서 중간층에 가중치 w_{ji}로

입력 방향의 반대로 가중치를 갱신해 나감.

사실 오차 역전파법은 경사 하강법, 즉 경사 하강법을 피드 포워드 네트워크에 적용하면,

오차 역전파법이 자연스럽게 도출됨.

$$E(w,v) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} t_{nk} ln(y_{nk})$$

우선 하나의 data n에만 해당하는 상호 엔트로피 E_n을 표현하면,

$$E_n(w,v) = -\sum_{k=0}^{K-1} t_k ln(y_k)$$

결국

$$E(w,v) = \frac{1}{N}\sum_{n=0}^{N-1} E_n(w,v)$$

위의 의미를 살펴보면, 평균 상호 엔트로피 오차는 data각각의 상호 엔트로피오차의 평균으로 해석됨.

합과 미분은 교환될수 있기 때문에, 각 data n에 대한 $\partial E_n/\partial w_{ji}$를 구하여 평균을 하면

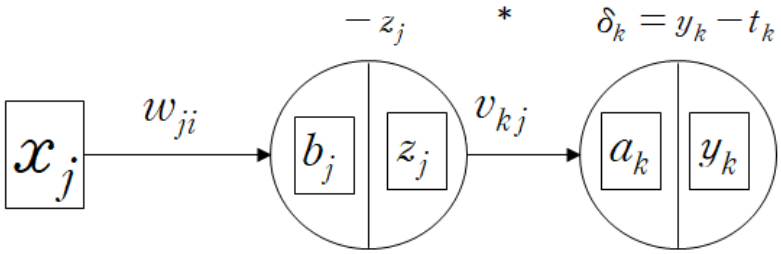
$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\frac{1}{N}\sum_{n=0}^{N-1} E_n = \frac{1}{N}\sum_{n=0}^{N-1}\frac{\partial E_n}{\partial w_{ji}}$$

본 예에서는 w,v임으로 E_n을 v_{kj}로 편미분한 식을 구하고, 다음으로 E_n을 w_{kj}로 편미분순서로 진행

$\partial E_n/\partial v_{kj}$구하기

$v_{kj}(t+1) = v_{kj}(t) - \alpha\delta_k z_j$의 의미

v_{kj}의 변화 =



$$\delta_j = h'(b_j) * \sum_{k=0}^{K-1} v_{kj}\delta_k$$

편미분의 연쇄율을 적용하여,

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial v_{kj}}$$

$E_n(w,v) = -\sum_{k=0}^{K-1} t_k ln(y_k)$에서 부터

$$\frac{\partial E}{\partial a_0} = \frac{\partial}{\partial a_0}(-t_0 lny_0 - t_1 lny_1 - t_2 lny_2)$$

t_k는 지도 신호이기 때문에 입력 총합 a_0로 변화하는것은 아니지만, 네트워크 출력인 y_k는 물론 입력 총합 a_0와 관계가 있음

t_k는 상수, y_k는a_0의 함수로 확장하면

$$\frac{\partial E}{\partial a_0} = -t_0\frac{1}{y_0}\frac{\partial y_0}{\partial a_0} - t_1\frac{1}{y_1}\frac{\partial y_1}{\partial a_0} - t_2\frac{1}{y_2}\frac{\partial y_2}{\partial a_0}$$

y가 a의 소프트맥스 함수임으로, $\frac{\partial y_0}{\partial a_0} = y_0(1-y_0)$

$$\frac{\partial y_1}{\partial a_0} = -y_0 y_1,\ \frac{\partial y_2}{\partial a_0} = -y_0 y_2$$

이들을 정리하면,

$$\frac{\partial E}{\partial a_0} = \frac{\partial}{\partial a_0}(-t_0 lny_0 - t_1 lny_1 - t_2 lny_2)$$

$= -t_0(1-y_0) + t_1 y_0 + t_2 y_0 = (t_0+t_1+t_2)y_0 - t_0 = y_0 - t_0$, 여기에서 $t_0+t_1+t_2 = 1$적용

y_0는첫뉴런출력으로, t_0는$test\ data$ 출력임으로$y_0 - t_0$는오차를나타내고있음, 마찬가지로 k=1,2의 경우

$\frac{\partial E}{\partial a_1} = y_1 - t_1, \frac{\partial E}{\partial a_2} = y_2 - t_2$

일반화 하면, $\frac{\partial E}{\partial a_k} = y_k - t_k = \delta_k$

$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial v_{kj}}$에서

$\frac{\partial a_k}{\partial v_{kj}}$를 구하는 과정

k=0의경우, $a_0 = v_{00}z_0 + v_{01}z_1 + v_{02}z_2$임으로

$\frac{\partial a_0}{\partial v_{00}} = z_0, \frac{\partial a_0}{\partial v_{01}} = z_1, \frac{\partial a_0}{\partial v_{02}} = z_2$

일반화 하면, $\frac{\partial a_0}{\partial v_{0j}} = z_j$

k=1, k=2의 경우도 같은 결과가 나타남으로, $\frac{\partial a_k}{\partial v_{kj}} = z_j$

결국 $\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial v_{kj}} = (y_k - t_k)z_j = \delta_k z_j$

v_{kj}의갱신규칙에의해서, $v_{kj}(t+1) = v_{kj}(t) - \alpha\frac{\partial E}{\partial v_{kj}} = v_{kj}(t) - \alpha\delta_k z_j$

경사 하강법을 적용하여, 도출된 결과인데 이결과의 의미를 살펴보면,

변경의 크기($v_{kj}(t+1) - v_{kj}(t) = -\alpha\delta_k z_j$)는 입력 크기($z_j$)와 앞에서 생기는 오차($\delta_k = (y_k - t_k)$)의 곱으로 결정됨.

오차(δ)는 실수이고, z_j는$\sigma(b_j)$임으로 항상 0~1사이 양수값

y_k는현재출력값이고, t_k는훈련시 나타난 목표$data$인데, 이값이 같으면 $\delta_k = (y_k - t_k) = 0$ 이되어 변화가 없고,

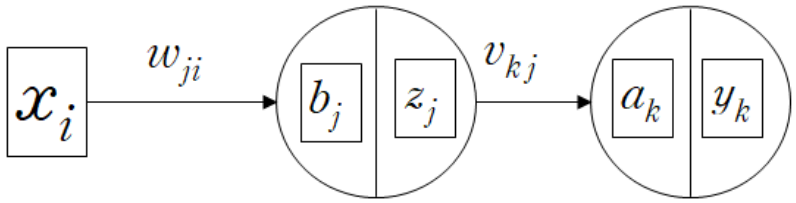
t_k가0인데, y_k가0보다크면, δ_k는양수가되어, $-\alpha\delta_k z_j$는음수가되어, v_{kj}는감소하는방향으로,

t_k가0인데, y_k가0보다작으면, δ_k는음수가되어, $-\alpha\delta_k z_j$는양수가되어, v_{kj}는증가

$\partial E_n / \partial w_{ji}$구하기

$w_{ji}(t+1) = w_{ji}(t) - \alpha\delta_j x_i$의 의미

w_{ji}의 변화 = $-x_i$ * δ_j



$$\delta_j = h'(b_j) * \sum_{k=0}^{K-1} v_{kj}\delta_k$$

이제 입력층에서 중간층 가중치 매개변수 w_{ji}의 학습 법칙을 살펴보자.

이것도 E를 w_{ji}로 편미분을 수행할 뿐임. 편미분의 연쇄법칙에 따라,

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial b_j}\frac{\partial b_j}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial b_j} = \delta_j$$

$$\frac{\partial b_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\sum_{i=0}^{D} w_{ji}x_i = x_i$$

$$w_{ji}(t+1) = w_{ji}(t) - \alpha\frac{\partial E}{\partial w_{ji}} = w_{ji}(t) - \alpha\delta_j x_i$$

이것은 v_{kj}의 갱신 규칙의 모습과 동일한 모양. 즉 w_{ji}도 결합전에 발생한 오차와 결합본래의 입력에 비례하는형태로 변경

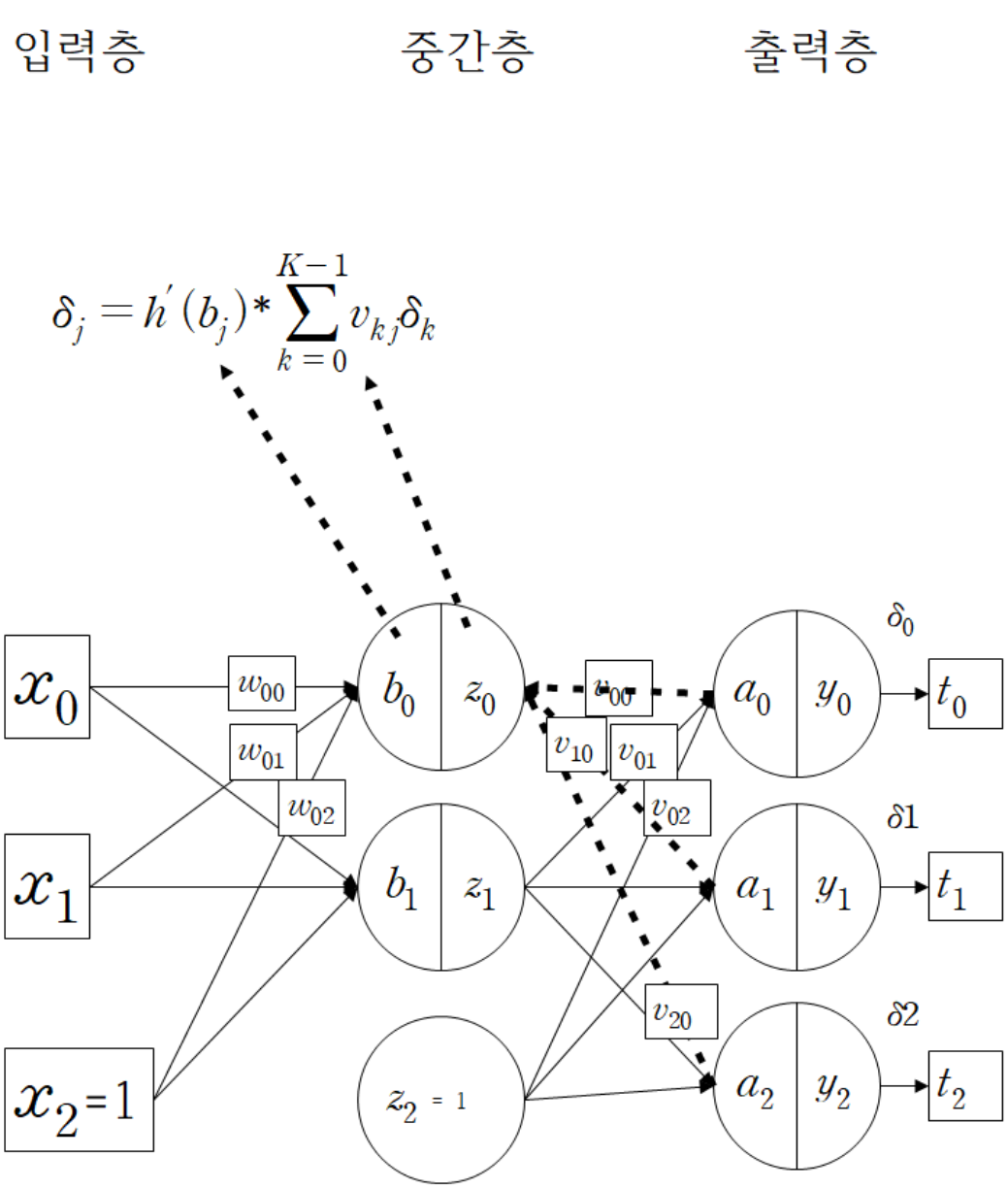
$$\delta_j = \frac{\partial E}{\partial b_j} = \left\{\sum_{k=0}^{K-1}\frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial z_j}\right\}\frac{\partial z_j}{\partial b_j}$$

$$\frac{\partial E}{\partial a_k} = \delta_k$$

$$\frac{\partial a_k}{\partial z_j} = \frac{\partial}{\partial z_j}\sum_{j=0}^{M} v_{kj}z_j = v_{kj}$$

$\frac{\partial z_j}{\partial b_j} = \frac{\partial}{\partial b_j}h(b_j) = h'(b_j)$, 중간층의 활성화 함수(시그모이드함수)

결국, $\delta_j = h'(b_j)\sum_{k=0}^{K-1} v_{kj}\delta_k$



이러한 결과로 인해 피드 포워드 신경망의 경사 하강법이 오차 역전파법이 됨.

이 일련의 절차는, $E(w,v) = \frac{1}{N}\sum_{n=0}^{N-1} E_n(w,v)$에서 data하나에 대한 갱신.

data가 N개 있음으로, 이러한 절차를 N번 처리해 학습의 1단계.

네트워크 매개변수(v_{kj}, w_{ji}) 갱신 방법

● 네트워크에 x를 입력하고 출력 y를 구하고, b,z,a도 저장해 놓는다.

$$b_j = \sum_{i=0}^{D} w_{ji} x_i$$

$$z_j = h(b_j)$$

$$a_k = \sum_{j=0}^{M} v_{kj} zj$$

$$y_k = e^{a_k} / (\sum_{l=0}^{K-1} e(a_l))$$

● y를 목표 data t와 비교해 출력 오차 δ_k를 개산

$$\delta_k = y_k - t_k$$

● 출력오차 δ_k를 사용하여 중간층 오차 δ_j를 계산

$$\delta_j = h'(b_j) \sum_{k=0}^{K-1} v_{kj} \delta_k$$

● 결합 본래의 신호 강도와 결합차의 오차 정보를 사용하여 가중치 매개 변수를 갱신

$$v_{kj}(t+1) = v_{kj}(t) - \alpha \delta_k z_j / N$$

$$w_{ji}(t+1) = w_{ji}(t) - \alpha \delta_j x_i / N$$

이 절차를 함수로 나타내면(오차 역전파법 함수)

In [1]:

```
def Sigmoid(x):
    y = 1 / (1 + np.exp(-x))
    return y


# 네트워크 ----------------------
def FNN(wv, M, K, x):
    N, D = x.shape # 입력 차원
    w = wv[:M * (D + 1)] # 중간층 뉴런의 가중치
    w = w.reshape(M, (D + 1))
    v = wv[M * (D + 1):] # 출력층 뉴런의 가중치
    v = v.reshape((K, M + 1))
    b = np.zeros((N, M + 1)) # 중간층 뉴런의 입력 총합
    z = np.zeros((N, M + 1)) # 중간층 뉴런의 출력
    a = np.zeros((N, K)) # 출력층 뉴런의 입력 총합
    y = np.zeros((N, K)) # 출력층 뉴런의 출력
    for n in range(N):
        # 중간층의 계산
        for m in range(M):
            b[n, m] = np.dot(w[m, :], np.r_[x[n, :], 1]) # (A)
            z[n, m] = Sigmoid(b[n, m])
        # 출력층의 계산
        z[n, M] = 1 # 더미 뉴런
```

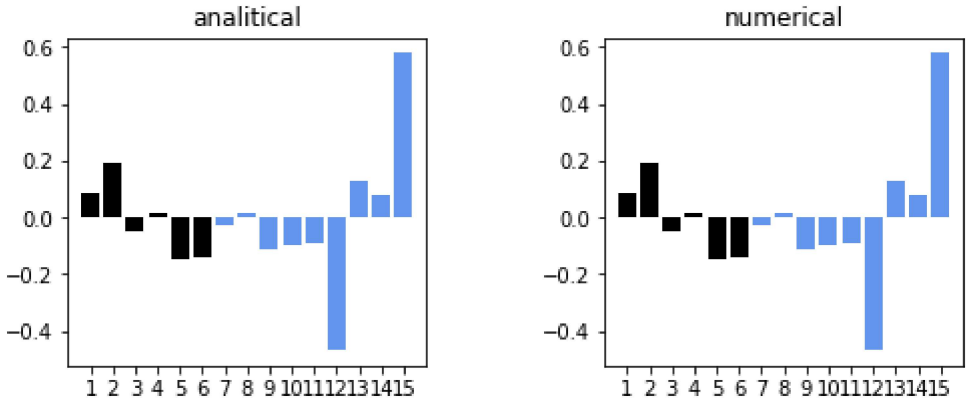
```
        wkz = 0
        for k in range(K):
            a[n, k] = np.dot(v[k, :], z[n, :])
            wkz = wkz + np.exp(a[n, k])
        for k in range(K):
            y[n, k] = np.exp(a[n, k]) / wkz
    return y, a, z, b

def CE_FNN(wv, M, K, x, t):
    N, D = x.shape
    y, a, z, b = FNN(wv, M, K, x)
    ce = -np.dot(np.log(y.reshape(-1)), t.reshape(-1)) / N
    return ce

def dCE_FNN_num(wv, M, K, x, t):
    epsilon = 0.001
    dwv = np.zeros_like(wv)
    for iwv in range(len(wv)):
        wv_modified = wv.copy()
        wv_modified[iwv] = wv[iwv] - epsilon
        mse1 = CE_FNN(wv_modified, M, K, x, t)
        wv_modified[iwv] = wv[iwv] + epsilon
        mse2 = CE_FNN(wv_modified, M, K, x, t)
        dwv[iwv] = (mse2 - mse1) / (2 * epsilon)
    return dwv
```

In [2]:

```
import numpy as np
import time
import matplotlib.pyplot as plt
%matplotlib inline

outfile = np.load('class_data.npz')
X_train = outfile['X_train']
T_train = outfile['T_train']
X_test = outfile['X_test']
T_test = outfile['T_test']
X_range0 = outfile['X_range0']
X_range1 = outfile['X_range1']

def dCE_FNN(wv, M, K, x, t):
    N, D = x.shape
#     print('D:', D)
#     print('N:', N)
    # wv을 w와 v로 되돌림
    w = wv[:M * (D + 1)]
    w = w.reshape(M, (D + 1))
    v = wv[M * (D + 1):]
    v = v.reshape((K, M + 1))
    # ① x를 입력하여 y를 얻음
    y, a, z, b = FNN(wv, M, K, x)
    # 출력 변수의 준비
    dwv = np.zeros_like(wv)
    dw = np.zeros((M, D + 1))
    dv = np.zeros((K, M + 1))
    delta1 = np.zeros(M) # 1층 오차
    delta2 = np.zeros(K) # 2층 오차(k = 0 부분은 사용하지 않음)
    for n in range(N): # (A)
        # ② 출력층의 오차를 구하기
        for k in range(K):
            delta2[k] = (y[n, k] - t[n, k])
        # ③ 중간층의 오차를 구하기
        for j in range(M):
            delta1[j] = z[n, j] * (1 - z[n, j]) * np.dot(v[:, j], delta2)
        # ④ v의 기울기 dv를 구하기
```

```
        for k in range(K):
            dv[k, :] = dv[k, :] + delta2[k] * z[n, :] / N
        # ④ w의 기울기 dw를 구하기
        for j in range(M):
            dw[j, :] = dw[j, :] + delta1[j] * np.r_[x[n, :], 1] / N
    # dw와 dv를 합체시킨 dwv로 만들기
    dwv = np.c_[dw.reshape((1, M * (D + 1))), W
                dv.reshape((1, K * (M + 1)))]
    dwv = dwv.reshape(-1)
    return dwv


#------Show VW
def Show_dWV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M * 3 + 1), wv[:M * 3],
            align="center", color='black')
    plt.bar(range(M * 3 + 1, N + 1), wv[M * 3:],
            align="center", color='cornflowerblue')
    plt.xticks(range(1, N + 1))
    plt.xlim(0, N + 1)


#-- 동작 확인
M = 2
K = 3
N = 2
nWV = M * 3 + K * (M + 1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)
print(WV)

dWV_ana = dCE_FNN(WV, M, K, X_train[:N, :], T_train[:N, :])
print("analytical dWV")
print(dWV_ana)


dWV_num = dCE_FNN_num(WV, M, K, X_train[:N, :], T_train[:N, :])
print("numerical dWV")
print(dWV_num)


plt.figure(1, figsize=(8, 3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1, 2, 1)
Show_dWV(dWV_ana, M)
plt.title('analitical')
plt.subplot(1, 2, 2)
Show_dWV(dWV_num, M)
plt.title('numerical')
plt.show()
```

```
[ 1.62434536 -0.61175641 -0.52817175 -1.07296862  0.86540763 -2.3015387
  1.74481176 -0.7612069   0.3190391  -0.24937038  1.46210794 -2.06014071
 -0.3224172  -0.38405435  1.13376944]
analytical dWV
[ 0.08848131  0.19158    -0.051398    0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115649 -0.10104422 -0.09427964 -0.46855604
  0.13096434  0.08076649  0.57971253]
numerical dWV
[ 0.0884813   0.19157999 -0.05139799  0.01281536 -0.14468029 -0.14242768
 -0.02992012  0.01351315 -0.11115648 -0.10104422 -0.09427964 -0.46855603
  0.13096434  0.08076649  0.57971252]
```



In [3]:

```
import time


# 해석적 미분을 사용한 구배법 -------
def Fit_FNN(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha):
    wv = wv_init.copy()
    err_train = np.zeros(n)
    err_test = np.zeros(n)
    wv_hist = np.zeros((n, len(wv_init)))
    epsilon = 0.001
    for i in range(n):
        wv = wv - alpha * dCE_FNN(wv, M, K, x_train, t_train) # (A)
        err_train[i] = CE_FNN(wv, M, K, x_train, t_train)
        err_test[i] = CE_FNN(wv, M, K, x_test, t_test)
        wv_hist[i, :] = wv
    return wv, wv_hist, err_train, err_test


# 메인 ----------------------------
startTime = time.time()
M = 2
K = 3
np.random.seed(1)
WV_init = np.random.normal(0, 0.01, M * 3 + K * (M + 1))
N_step = 1000
alpha = 1
WV, WV_hist, Err_train, Err_test = Fit_FNN(
    WV_init, M, K, X_train, T_train, X_test, T_test, N_step, alpha)
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

```
Calculation time:94.852 sec
```

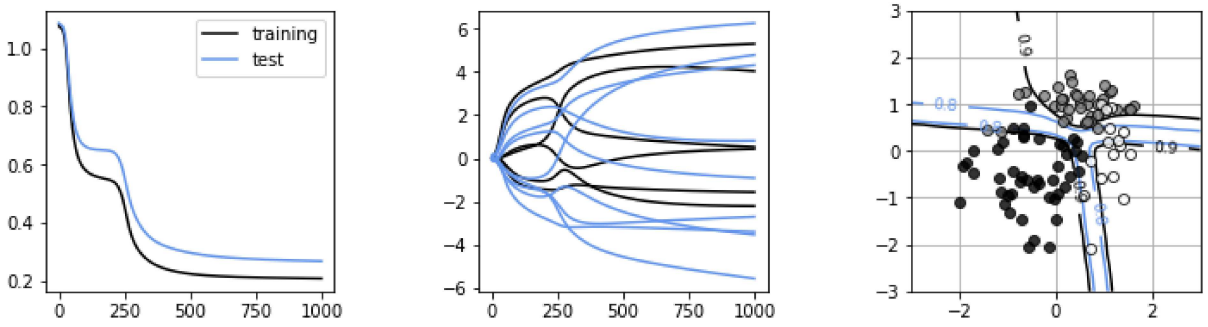
In [4]:

```
def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                 linestyle='none',
                 marker='o', markeredgecolor='black',
                 color=c[i], alpha=0.8)
    plt.grid(True)

def show_FNN(wv, M, K):
    xn = 60 # 등고선 표시 해상도
    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, 1), np.reshape(xx1, xn * xn, 1)]
```

```
    y, a, z, b = FNN(wv, M, K, x)
    plt.figure(1, figsize=(4, 4))
    for ic in range(K):
        f = y[:, ic]
        f = f.reshape(xn, xn)
        f = f.T
        cont = plt.contour(xx0, xx1, f, levels=[0.8, 0.9],
                           colors=['cornflowerblue', 'black'])
        cont.clabel(fmt='%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)
```

In [5]:

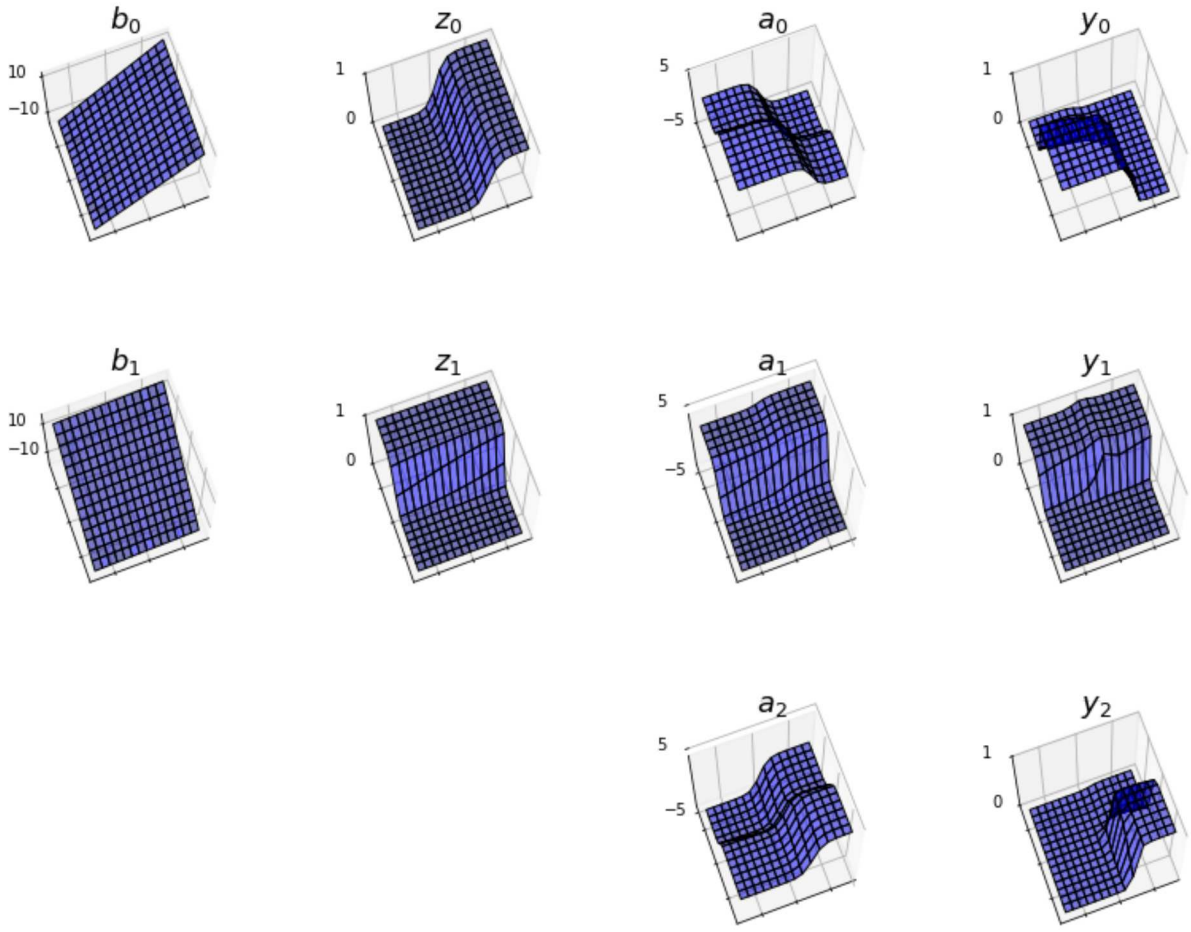
```
plt.figure(1, figsize=(12, 3))
plt.subplots_adjust(wspace=0.5)
# 학습 오차의 표시 ---------------------------
plt.subplot(1, 3, 1)
plt.plot(Err_train, 'black', label='training')
plt.plot(Err_test, 'cornflowerblue', label='test')
plt.legend()
# 가중치의 시간 변화 표시 ---------------------------
plt.subplot(1, 3, 2)
plt.plot(WV_hist[:, :M * 3], 'black')
plt.plot(WV_hist[:, M * 3:], 'cornflowerblue')
# 경계선 표시 --------------------------
plt.subplot(1, 3, 3)
Show_data(X_test, T_test)
M = 2
K = 3
show_FNN(WV, M, K)
plt.show()
```



In [6]:

```
# 리스트 7-1-(14)
from mpl_toolkits.mplot3d import Axes3D


def show_activation3d(ax, v, v_ticks, title_str):
    f = v.copy()
    f = f.reshape(xn, xn)
    f = f.T
    ax.plot_surface(xx0, xx1, f, color='blue', edgecolor='black',
                    rstride=1, cstride=1, alpha=0.5)
    ax.view_init(70, -110)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_zticks(v_ticks)
    ax.set_title(title_str, fontsize=18)


M = 2
K = 3
```

```
xn = 15 # 등고선 표시 해상도
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, 1), np.reshape(xx1, xn * xn, 1)]
y, a, z, b = FNN(WV, M, K, x)


fig = plt.figure(1, figsize=(12, 9))
plt.subplots_adjust(left=0.075, bottom=0.05, right=0.95,
                    top=0.95, wspace=0.4, hspace=0.4)

for m in range(M):
    ax = fig.add_subplot(3, 4, 1 + m * 4, projection='3d')
    show_activation3d(ax, b[:, m], [-10, 10], '$b_{0:d}$'.format(m))
    ax = fig.add_subplot(3, 4, 2 + m * 4, projection='3d')
    show_activation3d(ax, z[:, m], [0, 1], '$z_{0:d}$'.format(m))


for k in range(K):
    ax = fig.add_subplot(3, 4, 3 + k * 4, projection='3d')
    show_activation3d(ax, a[:, k], [-5, 5], '$a_{0:d}$'.format(k))
    ax = fig.add_subplot(3, 4, 4 + k * 4, projection='3d')
    show_activation3d(ax, y[:, k], [0, 1], '$y_{0:d}$'.format(k))


plt.show()
```



7.3 keras 실습: raspberry pi 4에서 실행

이미 실행한 과정을 keras library와 tensorflow를 적용하여 실행

In [1]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import time
np.random.seed(1) # 난수 초기화
import keras.optimizers #
from keras.models import Sequential #
from keras.layers.core import Dense, Activation #

#class_data.npz로 부터 data load
outfile = np.load('class_data.npz')
X_train = outfile['X_train']
T_train = outfile['T_train']
X_test = outfile['X_test']
T_test = outfile['T_test']
X_range0 = outfile['X_range0']
X_range1 = outfile['X_range1']
```

In [3]:
```
def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                 linestyle='none', marker='o',
                 markeredgecolor='black',
                 color=c[i], alpha=0.8)
    plt.grid(True)
```

Dense 초기화 함수의 구조

```
tf.keras.layers.Dense(
units, #input dimention
  activation=None, #적용할 activation함수명
    use_bias=True, #bias 적용 유무, default 적용
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

In [4]:
```
np.random.seed(1)


# Sequential 객체 model을 생성
model = Sequential()
#model에 층을 추가하여 네트워크의 구조를 형성해감,
#중간층으로, Dense(전결합형 층)
```

```
model.add(Dense(2, input_dim=2, activation='sigmoid', #2 뉴런 --> input_dim=2, activa
                kernel_initializer='uniform')) # 가중치 매개변수의초기값은 균일 난수로
# model.add(Dense(3,activation='softmax',
model.add(Dense(3, input_dim=3, activation='softmax',
                kernel_initializer='uniform'))
# 학습 방법을 설정, lr(학습 속도)
sgd = keras.optimizers.SGD(lr=2, momentum=0.0,
                           decay=0.0, nesterov=False)
# 학습, 목적함수를 교차 엔트로피 오차로 설정, 학습평가는 accuracy로 계산
model.compile(optimizer=sgd, loss='categorical_crossentropy',
#               metrics=['acc'])
              metrics=['accuracy'])


startTime = time.time()
# 학습 시작
#batch_size: 1단계분의 기울기 계산에 사용되는 학습 data 수,
#epochs: 전체 data를 학습에 상용한 횟수
history = model.fit(X_train, T_train, epochs=1000, batch_size=100,
#                    verbose=1, validation_data=(X_test, T_test)) # (E)
                    verbose=0, validation_data=(X_test, T_test)) # (E)


# 모델 평가 결과
score = model.evaluate(X_test, T_test, verbose=0) # (F)
print('cross entropy {0:3.2f}, accuracy {1:3.2f}'₩
      .format(score[0], score[1]))
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

```
cross entropy 0.26, accuracy 0.90
Calculation time:117.046 sec
```

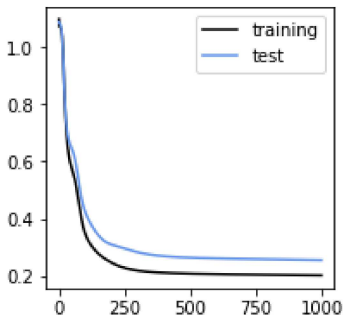
keras home page https://keras.io

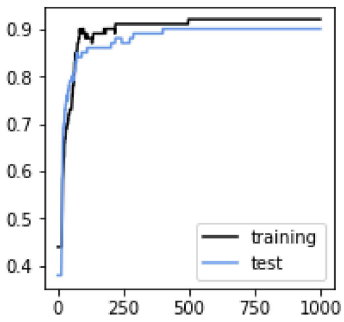
결과를 그려보면

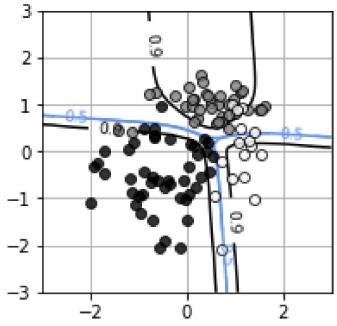
In [5]:

```
plt.figure(1, figsize = (12, 3))
plt.subplots_adjust(wspace=0.5)


# 학습 곡선 표시 -------------------------
plt.subplot(1, 3, 1)
plt.plot(history.history['loss'], 'black', label='training') # Training data의 학습곡
plt.plot(history.history['val_loss'], 'cornflowerblue', label='test') # Test data의 ㅎ
plt.legend()


# 정확도 표시 -------------------------
plt.subplot(1, 3, 2)
# plt.plot(history.history['acc'], 'black', label='training') # (C)
# plt.plot(history.history['val_acc'], 'cornflowerblue', label='test') # (D)
plt.plot(history.history['accuracy'], 'black', label='training') # Trasinig data의 정
plt.plot(history.history['val_accuracy'], 'cornflowerblue', label='test') # Test data
plt.legend()


# 경계선 표시 -------------------------
plt.subplot(1, 3, 3)
Show_data(X_test, T_test)
xn = 60 # 등고선 표시 해상도
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, 1), np.reshape(xx1, xn * xn, 1)]
y = model.predict(x) # (E)
K = 3
for ic in range(K):
    f = y[:, ic]
    f = f.reshape(xn, xn)
    f = f.T
    cont = plt.contour(xx0, xx1, f, levels=[0.5, 0.9], colors=[
        'cornflowerblue', 'black'])
    cont.clabel(fmt='%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)
plt.show()
```







In [ ]:

8 필기체 숫자 구분

8.1 3층 구조

입력 Data형태: 28x28(Gray scale image:256단계), 출력 Data형태: 0~9 숫자

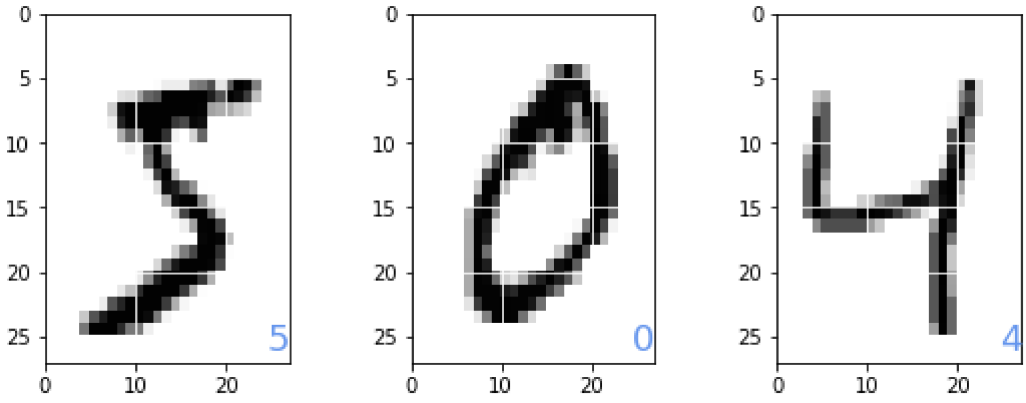
Train Data: 60,000, Test Data: 10,000

i번째 x_train data는 x_train[i,:,:], y_train data는 y_train[i]

In [1]:
```
#3개의 data를 나타내면,

from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnis
t.npz
11493376/11490434 [==============================] - 1s 0us/step
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(1, figsize=(12, 3.2))
plt.subplots_adjust(wspace=0.5)
plt.gray()
for id in range(3):
    plt.subplot(1, 4, id + 1)
    img = x_train[id, :, :]
    plt.pcolor(255 - img)
    plt.text(24.5, 26, "%d" % y_train[id],
             color='cornflowerblue', fontsize=18)
    plt.xlim(0, 27)
    plt.ylim(27, 0)
    plt.grid('off', color='white')
#     plt.grid('on', color='white')
plt.show()
```



2층 피드 포워드 네트워크 모델 적용층 피드 포워드 네트워크 모델 적용

In [3]:
```
from keras.utils import np_utils


x_train = x_train.reshape(60000, 784) # (A)
```

```
x_train = x_train.astype('float32') # (B)
x_train = x_train / 255 # (C)
num_classes = 10
# print(y_train[:2])

y_train = np_utils.to_categorical(y_train, num_classes) # (D)
# print(y_train[:2])

x_test = x_test.reshape(10000, 784)
x_test = x_test.astype('float32')
x_test = x_test / 255
y_test = np_utils.to_categorical(y_test, num_classes)
```

입력은 784차원, 출력층은 10개 분류, 10개의 뉴런 출력값이 확률을 나타내도록하기위해 활성화 함수는 소프트맥스 사용.

중간층은 16개로 하고, 활성화 함수는 시그모이드 함수를 사용.

이것을 정리하여 keras를 사용하는것으로 program하면

In [4]:
```
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam


model = Sequential() #
model.add(Dense(16, input_dim=784, activation='sigmoid')) # 입력은 784차원의 data, 16
model.add(Dense(10, activation='softmax')) # 확률을 나타내도록 softmax적용
model.compile(loss='categorical_crossentropy',
optimizer=Adam(), metrics=['accuracy']) # Adam(): 경사하강법의 일종
```

In [5]:
```
import time


startTime = time.time()
# history = model.fit(x_train, y_train, epochs=10, batch_size=1000,
history = model.fit(x_train, y_train, epochs=10, batch_size=5,
                    verbose=1, validation_data=(x_test, y_test)) # (A)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

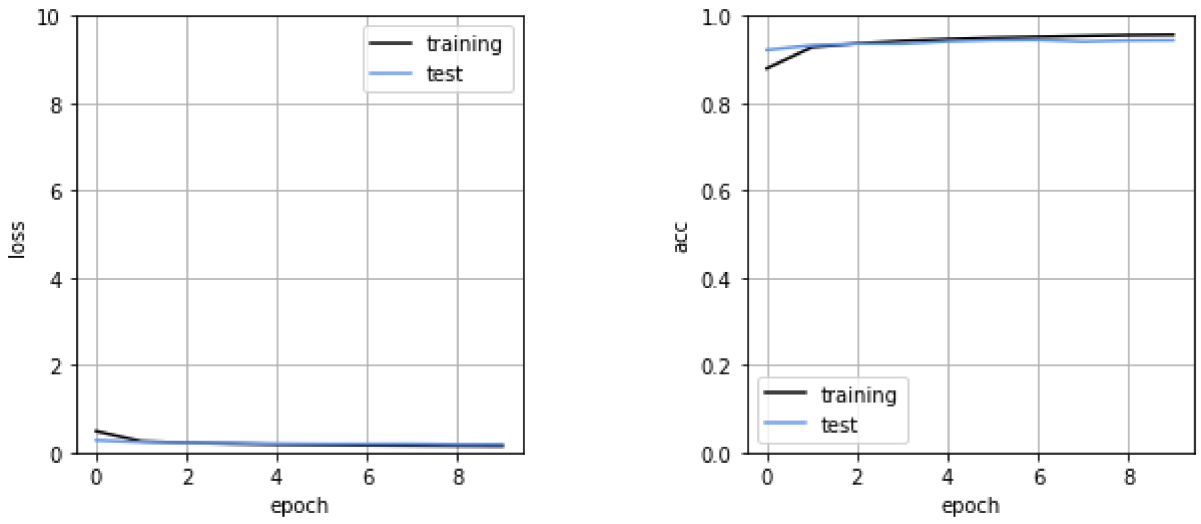
```
Epoch 1/10
12000/12000 [==============================] - 112s 9ms/step - loss: 0.4841 - accurac
y: 0.8794 - val_loss: 0.2754 - val_accuracy: 0.9219
Epoch 2/10
12000/12000 [==============================] - 116s 10ms/step - loss: 0.2512 - accurac
y: 0.9282 - val_loss: 0.2327 - val_accuracy: 0.9324
Epoch 3/10
12000/12000 [==============================] - 111s 9ms/step - loss: 0.2180 - accurac
y: 0.9365 - val_loss: 0.2189 - val_accuracy: 0.9360
Epoch 4/10
12000/12000 [==============================] - 112s 9ms/step - loss: 0.1975 - accurac
y: 0.9427 - val_loss: 0.2193 - val_accuracy: 0.9358
Epoch 5/10
12000/12000 [==============================] - 126s 11ms/step - loss: 0.1829 - accurac
y: 0.9462 - val_loss: 0.2002 - val_accuracy: 0.9413
Epoch 6/10
12000/12000 [==============================] - 126s 10ms/step - loss: 0.1711 - accurac
y: 0.9496 - val_loss: 0.1922 - val_accuracy: 0.9444
```

```
Epoch 7/10
12000/12000 [==============================] - 126s 11ms/step - loss: 0.1623 - accurac
y: 0.9516 - val_loss: 0.1927 - val_accuracy: 0.9449
Epoch 8/10
12000/12000 [==============================] - 128s 11ms/step - loss: 0.1558 - accurac
y: 0.9538 - val_loss: 0.1946 - val_accuracy: 0.9411
Epoch 9/10
12000/12000 [==============================] - 130s 11ms/step - loss: 0.1490 - accurac
y: 0.9558 - val_loss: 0.1836 - val_accuracy: 0.9436
Epoch 10/10
12000/12000 [==============================] - 127s 11ms/step - loss: 0.1443 - accurac
y: 0.9566 - val_loss: 0.1851 - val_accuracy: 0.9442
Test loss: 0.18505536019802094
Test accuracy: 0.9441999793052673
Computation time:1217.723 sec
```

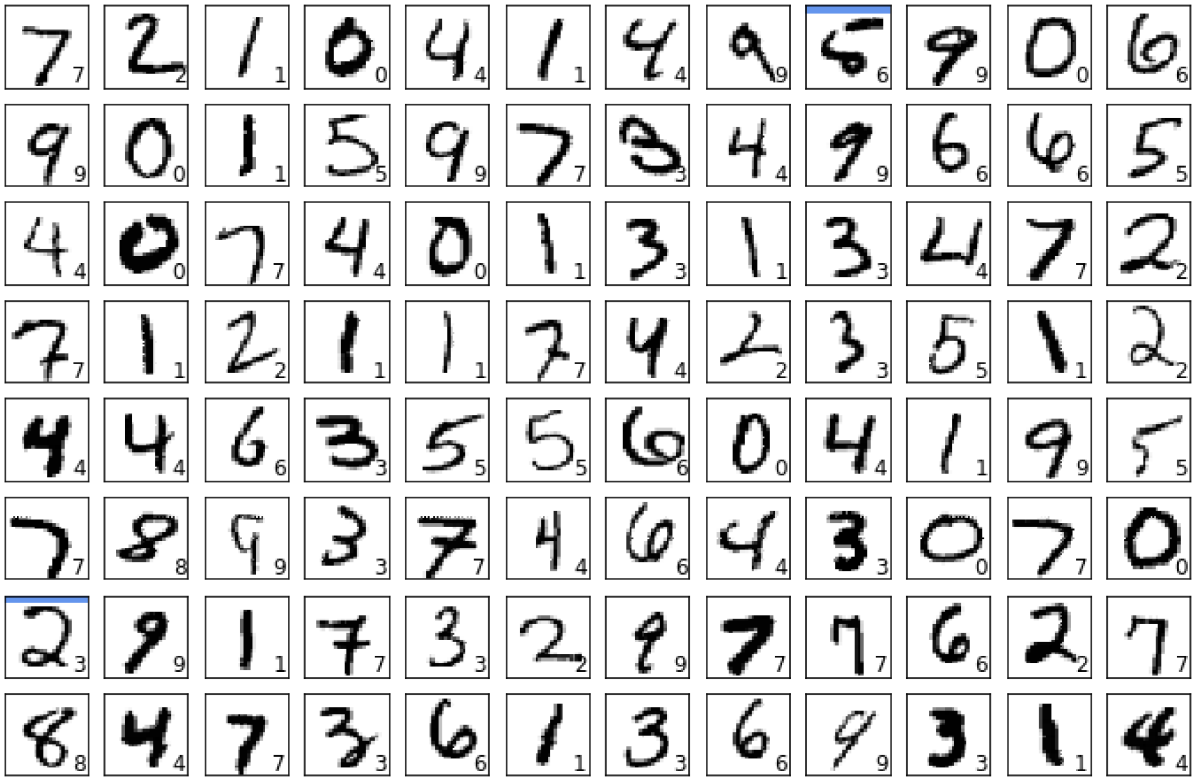
결과를 나타내보면

In [6]:

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


plt.figure(1, figsize=(10, 4))
plt.subplots_adjust(wspace=0.5)


plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='training', color='black')
plt.plot(history.history['val_loss'], label='test',
         color='cornflowerblue')
plt.ylim(0, 10)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')


plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='training', color='black')
plt.plot(history.history['val_accuracy'],label='test', color='cornflowerblue')
plt.ylim(0, 1)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('acc')
plt.show()
```



In [7]:
```
def show_prediction():
    n_show = 96
    y = model.predict(x_test) # Model에 test data를 입력하여, 결과를 확인, 즉 실사용상
    plt.figure(2, figsize=(12, 8))
    plt.gray()
    for i in range(n_show):
        plt.subplot(8, 12, i + 1)
        x = x_test[i, :]
        x = x.reshape(28, 28)
        plt.pcolor(1 - x)
        wk = y[i, :]
        prediction = np.argmax(wk)
        plt.text(22, 25.5, "%d" % prediction, fontsize=12)
        if prediction != np.argmax(y_test[i, :]): #예측값과 test결과값을 비교하여 일치
            plt.plot([0, 27], [1, 1], color='cornflowerblue', linewidth=5)
        plt.xlim(0, 27)
        plt.ylim(27, 0)
        plt.xticks([], "")
        plt.yticks([], "")
#-- 메인
show_prediction()
plt.show()
```



ReLU활성화 함수 적용(시그모이드 함수 대치)

In [8]:
```
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam


model = Sequential()
model.add(Dense(16, input_dim=784, activation='relu')) # (A)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(), metrics=['accuracy'])
```

```
startTime = time.time()
# history = model.fit(x_train, y_train, batch_size=1000, epochs=10,
history = model.fit(x_train, y_train, batch_size=10, epochs=10,
                    verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

```
Epoch 1/10
6000/6000 [==============================] - 57s 9ms/step - loss: 0.3591 - accuracy:
0.8982 - val_loss: 0.2392 - val_accuracy: 0.9310
Epoch 2/10
6000/6000 [==============================] - 63s 10ms/step - loss: 0.2224 - accuracy:
0.9355 - val_loss: 0.1995 - val_accuracy: 0.9439
Epoch 3/10
6000/6000 [==============================] - 66s 11ms/step - loss: 0.1926 - accuracy:
0.9436 - val_loss: 0.1805 - val_accuracy: 0.9465
Epoch 4/10
6000/6000 [==============================] - 67s 11ms/step - loss: 0.1743 - accuracy:
0.9495 - val_loss: 0.1719 - val_accuracy: 0.9517
Epoch 5/10
6000/6000 [==============================] - 67s 11ms/step - loss: 0.1627 - accuracy:
0.9516 - val_loss: 0.1657 - val_accuracy: 0.9520
Epoch 6/10
6000/6000 [==============================] - 67s 11ms/step - loss: 0.1536 - accuracy:
0.9552 - val_loss: 0.1652 - val_accuracy: 0.9520
Epoch 7/10
6000/6000 [==============================] - 72s 12ms/step - loss: 0.1466 - accuracy:
0.9577 - val_loss: 0.1597 - val_accuracy: 0.9539
Epoch 8/10
6000/6000 [==============================] - 72s 12ms/step - loss: 0.1402 - accuracy:
0.9591 - val_loss: 0.1608 - val_accuracy: 0.9527
Epoch 9/10
6000/6000 [==============================] - 74s 12ms/step - loss: 0.1356 - accuracy:
0.9598 - val_loss: 0.1552 - val_accuracy: 0.9526
Epoch 10/10
6000/6000 [==============================] - 71s 12ms/step - loss: 0.1325 - accuracy:
0.9606 - val_loss: 0.1656 - val_accuracy: 0.9506
Test loss: 0.1655847728252411
Test accuracy: 0.9506000280380249
Computation time:677.318 sec
```

In [9]:

```
show_prediction()
plt.show()
```

이 네트워크의 매개변수를 확인하는 방법

중간층 가중치 매개변수는 model.layers[0].get_weights()[0],

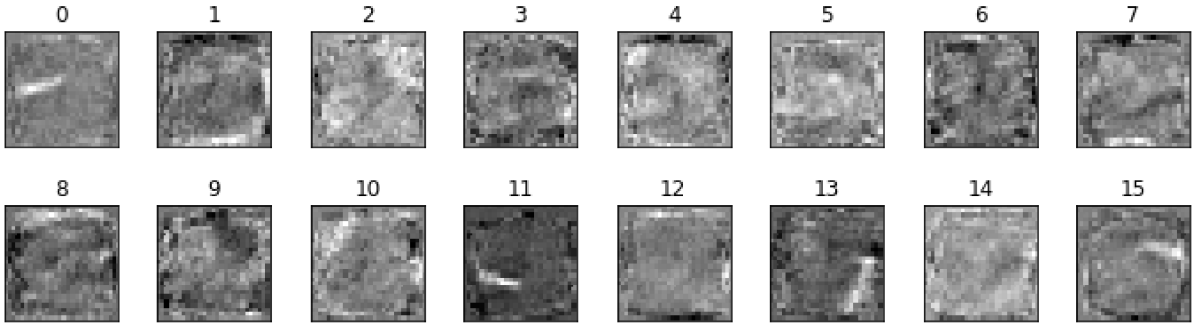
바이어스 매개변수는 model.layers[0].get_weights()[1]

출력층의 매개 변수는 model.layers[1].get_weights()[0],

바이어스 매개변수는 model.layers[1].get_weights()[1]

In [10]:

```
# 1층째의 weight 시각화
w = model.layers[0].get_weights()[0]
plt.figure(1, figsize=(12, 3))
plt.gray()
plt.subplots_adjust(wspace=0.35, hspace=0.5)
for i in range(16):
    plt.subplot(2, 8, i + 1)
    w1 = w[:, i]
    w1 = w1.reshape(28, 28)
    plt.pcolor(-w1)
    plt.xlim(0, 27)
    plt.ylim(27, 0)
    plt.xticks([], "")
    plt.yticks([], "")
    plt.title("%d" % i)
plt.show()
```



In [11]:

```
# 1층째의 무게 시각화
w = model.layers[1].get_weights()[0]
plt.figure(1, figsize=(12, 3))
plt.gray()
plt.subplots_adjust(wspace=0.35, hspace=0.5)
for i in range(10):
    print(w[:, i])
#     plt.subplot(2, 5, i + 1)
#     w1 = w[:, i]
#     w1 = w1.reshape(28, 28)
#     plt.pcolor(-w1)
#     plt.xlim(0, 27)
#     plt.ylim(27, 0)
#     plt.xticks([], "")
#     plt.yticks([], "")
#     plt.title("%d" % i)
# plt.show()
```

```
[-0.8845378   0.3052693   0.28402483 -0.5254166  -0.28698778  0.884413
 -0.32805055  0.44494528  0.14611728  0.05177419  0.14048529 -1.9591275
  0.34330606 -0.2628306  -0.00921323 -0.18877642]
[ 0.39641288  0.20687851  0.5098262  -0.04395254  0.37157482 -0.7900942
 -0.29004768 -0.9632464  -0.4490133   0.9435305   0.07029517  0.76622087
 -0.5309123   0.00386114 -0.9031469   0.40087238]
[ 0.7005871  -0.07761239  0.33150464 -0.30685696  0.12725021  0.6328398
  0.4339458   0.34246463  0.46050215  0.3317317  -0.31536618 -0.9338988
 -0.68681395 -1.3483859   0.283352    0.1578017 ]
[ 0.35847482 -0.04778852 -0.16425598  0.314271   -0.46819365 -0.01984981
  0.31093588  0.32269204  0.13851182  0.08618013 -0.8592853   0.6529109
  0.25248572 -0.05851312 -0.5362225   0.16287687]
[-0.72821903 -0.44267902  0.00403422  0.85664135 -0.43338674 -1.8937342
 -0.36128703  0.33708188  0.48856333 -0.76662076  0.26657233  0.23469983
 -0.39959285 -0.5817194   0.43932804 -1.8890287 ]
[-0.12591326 -0.01087733 -1.4364787  -0.09664261  0.29760894  0.11215827
  0.07310262  0.28769344  0.16895936 -0.72133094  0.38328066  0.5436826
  0.34011874  0.6515873  -0.48738173  0.8752138 ]
[-0.7401528   0.7337451  -0.3556884   0.10992327  0.46388438  0.17198296
 -0.21953167  0.44189677 -0.381472   -0.49462038  0.25924534 -1.9588534
 -0.4766854  -0.22780265  0.13052787  0.15221733]
[ 0.77894074 -0.3255422   0.62597966 -0.24696063 -0.2909258   0.2187577
 -0.6230745  -0.5111853   0.6069453  -0.3563605  -0.83841425 -1.0299776
  0.41601482  0.24182577  0.40750495 -0.09875393]
[-0.3691551  -0.15679944 -0.63126034  0.38319162  0.51606536  0.00661234
  0.32987487 -0.67407143 -0.18919295  0.08134646  0.10702017 -1.0699886
  0.27733415 -0.13941947 -0.1664611  -0.49819097]
[-1.7578363  -1.4213212   0.27145997  0.40717447 -0.6338168  -0.4248517
  0.01865304  0.1665182  -0.37508816 -0.25111258  0.07424408  0.51582855
  0.26835522 -0.15341431  0.53901404 -0.09641204]
<Figure size 864x216 with 0 Axes>
```

8.2 다층 구조

공간 필터

합성곱 신경망: 필터를 사용한 신경망(Convolution Neural Network: CNN)

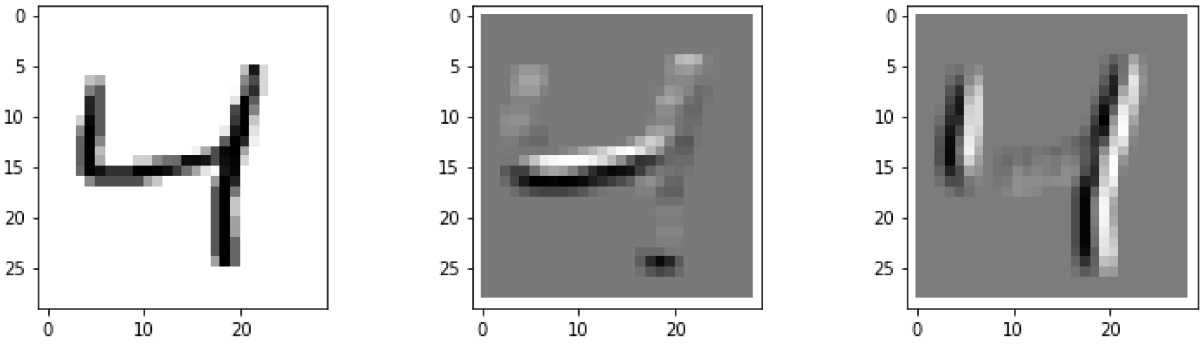
In [1]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

In [2]:
```
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
num_classes = 10
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

In [3]:
```
import matplotlib.pyplot as plt
%matplotlib inline


id_img = 2
myfil1 = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [-2, -2, -2]], dtype=float) # (A)
myfil2 = np.array([[-2, 1, 1],
                   [-2, 1, 1],
                   [-2, 1, 1]], dtype=float) # (B)


x_img = x_train[id_img, :, :, 0]
img_h = 28
img_w = 28
x_img = x_img.reshape(img_h, img_w)
out_img1 = np.zeros_like(x_img)
out_img2 = np.zeros_like(x_img)

# 필터 처리
for ih in range(img_h - 3):
    for iw in range(img_w - 3):
        img_part = x_img[ih:ih + 3, iw:iw + 3]
        out_img1[ih + 1, iw + 1] = ₩
            np.dot(img_part.reshape(-1), myfil1.reshape(-1))
        out_img2[ih + 1, iw + 1] = ₩
            np.dot(img_part.reshape(-1), myfil2.reshape(-1))


# - 표시
plt.figure(1, figsize=(12, 3.2))
plt.subplots_adjust(wspace=0.5)
plt.gray()
```

```
plt.subplot(1, 3, 1)
plt.pcolor(1 - x_img)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.subplot(1, 3, 2)
plt.pcolor(-out_img1)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.subplot(1, 3, 3)
plt.pcolor(-out_img2)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.show()
```



In [4]:

```
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time


model = Sequential()
model.add(Conv2D(8, (3, 3), padding='same',
                 input_shape=(28, 28, 1), activation='relu')) # (A)
model.add(Flatten()) # (B)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
startTime = time.time()
history = model.fit(x_train, y_train, batch_size=1000, epochs=20,
                    verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

```
Epoch 1/20
60/60 [==============================] - 28s 472ms/step - loss: 0.7732 - accuracy: 0.8
062 - val_loss: 0.3397 - val_accuracy: 0.9034
Epoch 2/20
60/60 [==============================] - 32s 531ms/step - loss: 0.3122 - accuracy: 0.9
105 - val_loss: 0.2692 - val_accuracy: 0.9213
Epoch 3/20
60/60 [==============================] - 29s 479ms/step - loss: 0.2587 - accuracy: 0.9
260 - val_loss: 0.2318 - val_accuracy: 0.9335
Epoch 4/20
60/60 [==============================] - 30s 499ms/step - loss: 0.2208 - accuracy: 0.9
373 - val_loss: 0.1995 - val_accuracy: 0.9436
Epoch 5/20
```

```
60/60 [==============================] - 29s 491ms/step - loss: 0.1910 - accuracy: 0.9
460 - val_loss: 0.1807 - val_accuracy: 0.9472
Epoch 6/20
60/60 [==============================] - 30s 494ms/step - loss: 0.1697 - accuracy: 0.9
521 - val_loss: 0.1596 - val_accuracy: 0.9574
Epoch 7/20
60/60 [==============================] - 31s 509ms/step - loss: 0.1516 - accuracy: 0.9
571 - val_loss: 0.1454 - val_accuracy: 0.9606
Epoch 8/20
60/60 [==============================] - 30s 492ms/step - loss: 0.1373 - accuracy: 0.9
616 - val_loss: 0.1372 - val_accuracy: 0.9638
Epoch 9/20
60/60 [==============================] - 29s 479ms/step - loss: 0.1255 - accuracy: 0.9
652 - val_loss: 0.1268 - val_accuracy: 0.9644
Epoch 10/20
60/60 [==============================] - 30s 500ms/step - loss: 0.1161 - accuracy: 0.9
683 - val_loss: 0.1216 - val_accuracy: 0.9664
Epoch 11/20
60/60 [==============================] - 30s 501ms/step - loss: 0.1076 - accuracy: 0.9
704 - val_loss: 0.1157 - val_accuracy: 0.9675
Epoch 12/20
60/60 [==============================] - 30s 493ms/step - loss: 0.1016 - accuracy: 0.9
717 - val_loss: 0.1092 - val_accuracy: 0.9689
Epoch 13/20
60/60 [==============================] - 29s 486ms/step - loss: 0.0957 - accuracy: 0.9
739 - val_loss: 0.1043 - val_accuracy: 0.9706
Epoch 14/20
60/60 [==============================] - 30s 503ms/step - loss: 0.0903 - accuracy: 0.9
752 - val_loss: 0.1011 - val_accuracy: 0.9710
Epoch 15/20
60/60 [==============================] - 30s 507ms/step - loss: 0.0853 - accuracy: 0.9
762 - val_loss: 0.0968 - val_accuracy: 0.9720
Epoch 16/20
60/60 [==============================] - 30s 495ms/step - loss: 0.0810 - accuracy: 0.9
773 - val_loss: 0.0970 - val_accuracy: 0.9725
Epoch 17/20
60/60 [==============================] - 30s 498ms/step - loss: 0.0771 - accuracy: 0.9
783 - val_loss: 0.0918 - val_accuracy: 0.9727
Epoch 18/20
60/60 [==============================] - 31s 514ms/step - loss: 0.0728 - accuracy: 0.9
799 - val_loss: 0.0889 - val_accuracy: 0.9744
Epoch 19/20
60/60 [==============================] - 31s 510ms/step - loss: 0.0692 - accuracy: 0.9
809 - val_loss: 0.0848 - val_accuracy: 0.9748
Epoch 20/20
60/60 [==============================] - 30s 505ms/step - loss: 0.0660 - accuracy: 0.9
812 - val_loss: 0.0820 - val_accuracy: 0.9752
Test loss: 0.08196050673723221
Test accuracy: 0.9751999974250793
Computation time:615.109 sec
```

In [5]:

```
def show_prediction():
    n_show = 96
    y = model.predict(x_test) # (A)
    plt.figure(2, figsize=(12, 8))
    plt.gray()
    for i in range(n_show):
        plt.subplot(8, 12, i + 1)
        x = x_test[i, :]
        x = x.reshape(28, 28)
        plt.pcolor(1 - x)
        wk = y[i, :]
        prediction = np.argmax(wk)
        plt.text(22, 25.5, "%d" % prediction, fontsize=12)
        if prediction != np.argmax(y_test[i, :]):
            plt.plot([0, 27], [1, 1], color='cornflowerblue', linewidth=5)
        plt.xlim(0, 27)
        plt.ylim(27, 0)
```

```
        plt.xticks([], "")
        plt.yticks([], "")
```

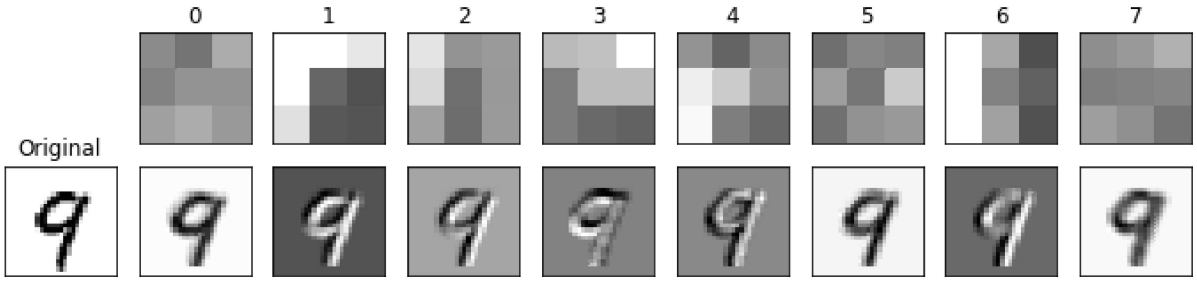
In [6]:

```
show_prediction()
plt.show()
```

In [7]:

```
plt.figure(1, figsize=(12, 2.5))
plt.gray()
plt.subplots_adjust(wspace=0.2, hspace=0.2)
plt.subplot(2, 9, 10)
id_img = 12
x_img = x_test[id_img, :, :, 0]
img_h = 28
img_w = 28
x_img = x_img.reshape(img_h, img_w)
plt.pcolor(-x_img)
plt.xlim(0, img_h)
plt.ylim(img_w, 0)
plt.xticks([], "")
plt.yticks([], "")
plt.title("Original")


w = model.layers[0].get_weights()[0] # (A)
max_w = np.max(w)
min_w = np.min(w)
for i in range(8):
    plt.subplot(2, 9, i + 2)
    w1 = w[:, :, 0, i]
    w1 = w1.reshape(3, 3)
    plt.pcolor(-w1, vmin=min_w, vmax=max_w)
    plt.xlim(0, 3)
    plt.ylim(3, 0)
    plt.xticks([], "")
    plt.yticks([], "")
    plt.title("%d" % i)
```

```
    plt.subplot(2, 9, i + 11)
    out_img = np.zeros_like(x_img)
    # 필터 처리
    for ih in range(img_h - 3):
        for iw in range(img_w - 3):
            img_part = x_img[ih:ih + 3, iw:iw + 3]
            out_img[ih + 1, iw + 1] = W
            np.dot(img_part.reshape(-1), w1.reshape(-1))
    plt.pcolor(-out_img)
    plt.xlim(0, img_w)
    plt.ylim(img_h, 0)
    plt.xticks([], "")
    plt.yticks([], "")
plt.show()
```



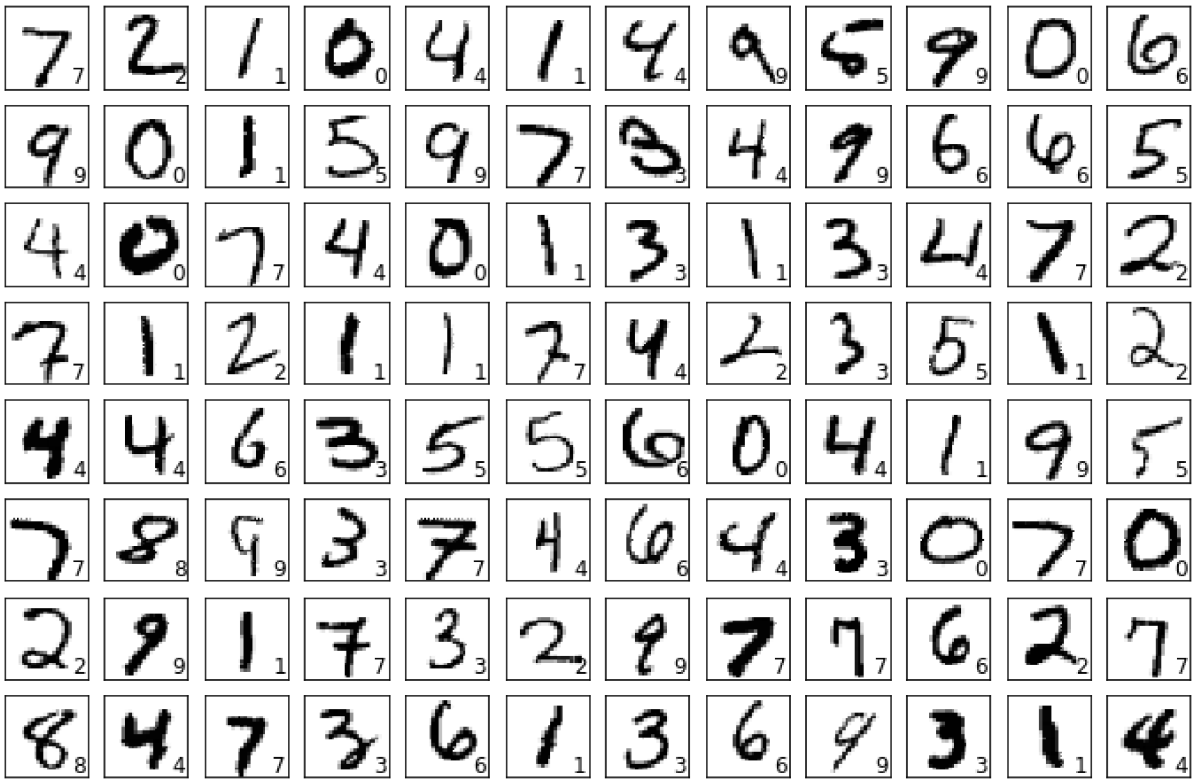
In [8]:

```
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import Adam
import time


model = Sequential()
model.add(Conv2D(16, (3, 3),
                 input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) # (A)
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) # (B)
model.add(Dropout(0.25)) # (C)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25)) # (D)
model.add(Dense(num_classes, activation='softmax'))


model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])


startTime = time.time()


history = model.fit(x_train, y_train, batch_size=1000, epochs=20,
                    verbose=1, validation_data=(x_test, y_test))


score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

```
Epoch 1/20
60/60 [==============================] - 243s 4s/step - loss: 0.6822 - accuracy: 0.795
6 - val_loss: 0.1315 - val_accuracy: 0.9606
Epoch 2/20
60/60 [==============================] - 252s 4s/step - loss: 0.1388 - accuracy: 0.957
6 - val_loss: 0.0641 - val_accuracy: 0.9792
Epoch 3/20
60/60 [==============================] - 260s 4s/step - loss: 0.0892 - accuracy: 0.972
4 - val_loss: 0.0458 - val_accuracy: 0.9861
Epoch 4/20
60/60 [==============================] - 261s 4s/step - loss: 0.0716 - accuracy: 0.978
2 - val_loss: 0.0386 - val_accuracy: 0.9878
Epoch 5/20
60/60 [==============================] - 261s 4s/step - loss: 0.0589 - accuracy: 0.981
4 - val_loss: 0.0356 - val_accuracy: 0.9882
Epoch 6/20
60/60 [==============================] - 264s 4s/step - loss: 0.0509 - accuracy: 0.984
6 - val_loss: 0.0288 - val_accuracy: 0.9905
Epoch 7/20
60/60 [==============================] - 264s 4s/step - loss: 0.0465 - accuracy: 0.985
2 - val_loss: 0.0272 - val_accuracy: 0.9914
Epoch 8/20
60/60 [==============================] - 264s 4s/step - loss: 0.0396 - accuracy: 0.987
8 - val_loss: 0.0261 - val_accuracy: 0.9906
Epoch 9/20
60/60 [==============================] - 265s 4s/step - loss: 0.0377 - accuracy: 0.988
0 - val_loss: 0.0235 - val_accuracy: 0.9922
Epoch 10/20
60/60 [==============================] - 290s 5s/step - loss: 0.0334 - accuracy: 0.989
5 - val_loss: 0.0236 - val_accuracy: 0.9930
Epoch 11/20
60/60 [==============================] - 266s 4s/step - loss: 0.0294 - accuracy: 0.990
7 - val_loss: 0.0217 - val_accuracy: 0.9931
Epoch 12/20
60/60 [==============================] - 266s 4s/step - loss: 0.0285 - accuracy: 0.991
0 - val_loss: 0.0222 - val_accuracy: 0.9936
Epoch 13/20
60/60 [==============================] - 286s 5s/step - loss: 0.0259 - accuracy: 0.991
4 - val_loss: 0.0226 - val_accuracy: 0.9932
Epoch 14/20
60/60 [==============================] - 380s 6s/step - loss: 0.0247 - accuracy: 0.992
1 - val_loss: 0.0221 - val_accuracy: 0.9935
Epoch 15/20
60/60 [==============================] - 267s 4s/step - loss: 0.0231 - accuracy: 0.992
5 - val_loss: 0.0227 - val_accuracy: 0.9925
Epoch 16/20
60/60 [==============================] - 267s 4s/step - loss: 0.0208 - accuracy: 0.993
2 - val_loss: 0.0203 - val_accuracy: 0.9943
Epoch 17/20
60/60 [==============================] - 330s 6s/step - loss: 0.0198 - accuracy: 0.993
5 - val_loss: 0.0206 - val_accuracy: 0.9934
Epoch 18/20
60/60 [==============================] - 269s 4s/step - loss: 0.0196 - accuracy: 0.993
4 - val_loss: 0.0244 - val_accuracy: 0.9918
Epoch 19/20
60/60 [==============================] - 304s 5s/step - loss: 0.0185 - accuracy: 0.994
0 - val_loss: 0.0224 - val_accuracy: 0.9927
Epoch 20/20
60/60 [==============================] - 382s 6s/step - loss: 0.0157 - accuracy: 0.994
6 - val_loss: 0.0203 - val_accuracy: 0.9938
Test loss: 0.02030540257692337
Test accuracy: 0.9937999844551086
Computation time:5760.290 sec
```

In [9]:

```
show_prediction()
plt.show()
```



9. 쉴드 보드 실습

9.1 BH1750 조도 센서

지도 학습:회귀 실습(led light--> photo sensor lux확인)

이번절에서는 pwm제어되는 led 불빛으로, photo sensor값을 읽어 들여, 그에 따른 지도 학습을 구현.

led를 라즈베리파이 port(port 12)에 직접연결하여, pwm신호를 보내어 밝기를 조정.

일정 거리에 있는 photo sensor에서 밝기를 측정.

photo sensor: BH1750FVI

결과값은, BH1750FVI의 특성이 좋아(밝기에 1차원 비례형태로 측정값이 나타남) 오차가 작음.

다음 실험(조도 센서를 CDS를 적용)에서는 CDS특성이 반비례관계이고, 특성에 오차가 큼.

PWM신호는 랜덤하게 생성시키고, 50회의 밝기 조정에 따른, 조도를 측정하고,

그에 따른 model의 회기 특성을 확인 한다.

In [1]:

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

import smbus
import time
import RPi.GPIO as GPIO


# GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT) #LED 밝기 조정을 위한 PORT 설정

# BH1750 MODE 설정
DEVICE = 0X23
POWER_DOWN = 0X00
POWER_ON = 0X01
RESET = 0X07

CONTINUOUS_LOW_RES_MODE = 0X13
CONTINUOUS_HIGH_RES_MODE_1 = 0X10
CONTINUOUS_HIGH_RES_MODE_2 = 0X11
ONE_TIME_HIGH_RES_MODE_1 = 0X20
ONE_TIME_HIGH_RES_MODE_2 = 0X21
ONE_TIME_LOW_RES_MODE = 0X23

bus = smbus.SMBus(1)
light = GPIO.PWM(12, 100)  # channel=12 frequency=1KHz

def convertToNumber(data):
    result=int((data[1]+(256*data[0]))/1.2)
    return (result)

#photo 출력 읽기
def readLight(addr=DEVICE):
    data = bus.read_i2c_block_data(addr, CONTINUOUS_HIGH_RES_MODE_1)
```

```
    return convertToNumber(data)

X_min = 0
X_max = 0
T_min = 0
T_max = 0
X_n = 50 #측정 횟수
np.random.seed(seed=1) # 난수를 고정

T = np.zeros(X_n)
X = np.random.rand(X_n)*100
X = np.round(X,0)

def main():
    light.start(0)
    count = 0
    finish = False
    light.ChangeDutyCycle(100)
    time.sleep(1)
#     50회 반복
    while not finish:
        light.ChangeDutyCycle(100 - X[count])
        time.sleep(0.5)
        T[count] = readLight()
        if (count%10) == 0:
            print("count: {0}, Light Level: {1}".format(count, T[count]))
        count += 1
        if count >= X_n:
            finish = True
            X_min = int(X.min())
            X_max = int(X.max())
            T_min = int(T.min())
            T_max = int(T.max())
            print('photo max:{0}, min:{1}'.format(T.max(), T.min()))
            print('led: max:{0}, min:{1}'.format(X_max, X_min))
            print('End of measure')
            np.savez('photo.npz', X=X, X_min=X_min, X_max=X_max, X_n=X_n, T=T, T_min=
    light.stop()
    GPIO.cleanup()

if __name__ == "__main__":
    main()
```

```
count: 0, Light Level: 107.0
count: 10, Light Level: 107.0
count: 20, Light Level: 138.0
count: 30, Light Level: 82.0
count: 40, Light Level: 154.0
photo max:154.0, min:74.0
led: max:99, min:0
End of measure
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
outfile = np.load('photo.npz')

X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']
T_min = outfile['T_min']
T_max = outfile['T_max']
```

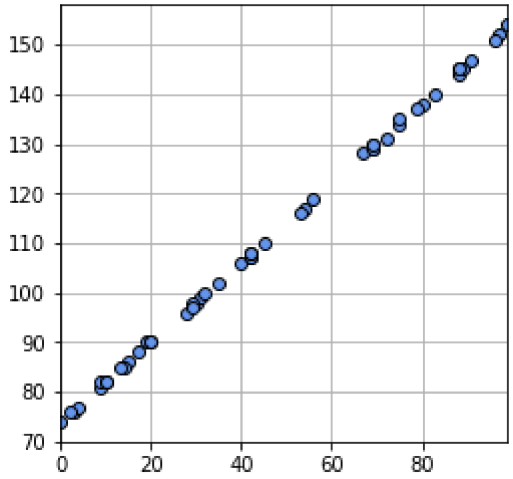
```
# 리스트 5-1-(2)
print(X)
# 리스트 5-1-(4)
print(np.round(T, 2))
```

```
[42. 72.  0. 30. 15.  9. 19. 35. 40. 54. 42. 69. 20. 88.  3. 67. 42. 56.
 14. 20. 80. 97. 31. 69. 88. 89.  9.  4. 17. 88. 10. 42. 96. 53. 69. 32.
 69. 83.  2. 75. 99. 75. 28. 79. 10. 45. 91. 29. 29. 13.]
[107. 131.  74.  98.  86.  81.  90. 102. 106. 117. 107. 129.  90. 144.
  76. 128. 108. 119.  85.  90. 138. 152.  99. 130. 145. 145.  82.  77.
  88. 145.  82. 108. 151. 116. 129. 100. 130. 140.  76. 134. 154. 135.
  96. 137.  82. 110. 147.  98.  97.  85.]
```

In [3]:

```
# 리스트 5-1-(5)
# 데이터 그래프 ----------------------------
plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')
plt.xlim(X_min, X_max)
print("min: {0}, max:{1}".format(X_min, X_max))
# plt.xlim(0, 99)
plt.grid(True)
plt.show()
```

```
min: 0, max:99
```

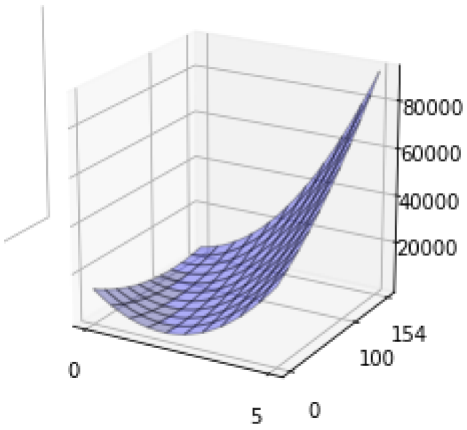


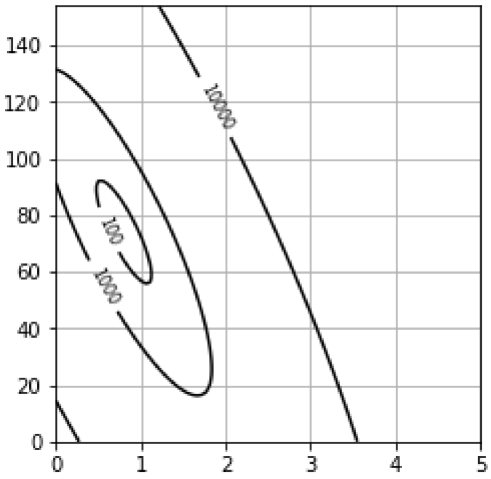
$$J=\frac{1}{N}\sum_{n=0}^{N-1}(y_n-t_n)^2$$

In [4]:

```
# 리스트 5-1-(6)
from mpl_toolkits.mplot3d import Axes3D
# 평균 오차 함수 ----------------------------
def mse_line(x, t, w):
    y = w[0] * x + w[1]
    mse = np.mean((y - t)**2)
    return mse

# 계산 ----------------------------------
xn = 100 # 등고선 표시 해상도
w0_range = [0, 5]
w1_range = [0, T_max]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
```

```
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))

# 표시 ------------------------------------
plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)

ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx0, xx1, J, rstride=10, cstride=10, alpha=0.3,
                color='blue', edgecolor='black')
ax.set_xticks([-5, 0, 5])
# ax.set_yticks([0, (T_min+T_max)/2, T_max])
ax.set_yticks([0, 100, T_max])
ax.view_init(20, -60)

plt.subplot(1, 2, 2)
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=[100, 1000, 10000, 100000])
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
plt.show()
```





$$y_n = y(x_n) = w_0 x_n + w_1$$

$$w_0(t+1) - w_0(t) = -\alpha \frac{2}{N} \sum_{n=0}^{N-1} (y_n - t_n) x_n$$

$$w_1(t+1) - w_1(t) = -\alpha \frac{2}{N} \sum_{n=0}^{N-1} (y_n - t_n)$$

In [5]:

```
# 리스트 5-1-(7)
# 평균 제곱 오차의 기울기 ------------------------
def dmse_line(x, t, w):
    y = w[0] * x + w[1]
    d_w0 = 2 * np.mean((y - t) * x)
    d_w1 = 2 * np.mean(y - t)
    return d_w0, d_w1
```

In [6]:

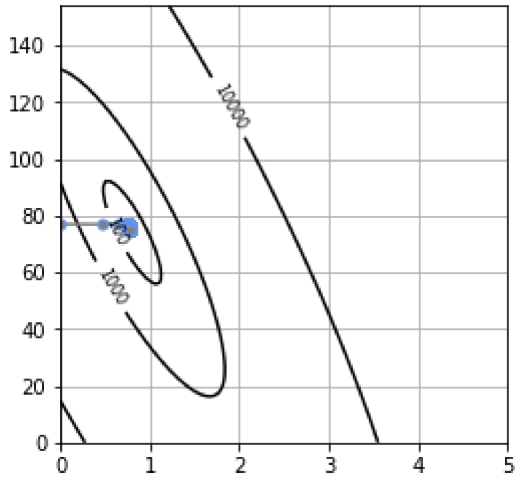
```
# 리스트 5-1-(8)
d_w = dmse_line(X, T, [1, 100])
print(np.round(d_w, 1))
```

```
[3666.    70.5]
```

In [7]:

```
# 리스트 5-1-(9)
# 구배법 ----------------------------------
def fit_line_num(x, t):
    w_init = [X_min, T_max/2] # 초기 매개 변수
    alpha = 0.0001 # 학습률
    i_max = 100000 # 반복의 최대 수
    eps = 0.5 # 반복을 종료 기울기의 절대 값의 한계
    w_i = np.zeros([i_max, 2])
    w_i[0, :] = w_init
    for i in range(1, i_max):
        dmse = dmse_line(x, t, w_i[i - 1])
        w_i[i, 0] = w_i[i - 1, 0] - alpha * dmse[0]
        w_i[i, 1] = w_i[i - 1, 1] - alpha * dmse[1]
        if max(np.absolute(dmse)) < eps: # 종료판정, np.absolute는 절대치
            break
    w0 = w_i[i, 0]
    w1 = w_i[i, 1]
    w_i = w_i[:i, :]
    return w0, w1, dmse, w_i


# 메인 ------------------------------------
plt.figure(figsize=(4, 4)) # MSE의 등고선 표시
xn = 100 # 등고선 해상도
w0_range = [0, 5]
w1_range = [0, T_max]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))

cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=(100, 1000, 10000, 100000))
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
# 구배법 호출
W0, W1, dMSE, W_history = fit_line_num(X, T)

# 결과보기
print('반복 횟수 {0}'.format(W_history.shape[0]))
print('W=[{0:.6f}, {1:.6f}]'.format(W0, W1))
print('dMSE=[{0:.6f}, {1:.6f}]'.format(dMSE[0], dMSE[1]))
print('MSE={0:.6f}'.format(mse_line(X, T, [W0, W1])))
plt.plot(W_history[:, 0], W_history[:, 1], '.-',
         color='gray', markersize=10, markeredgecolor='cornflowerblue')
plt.show()
```

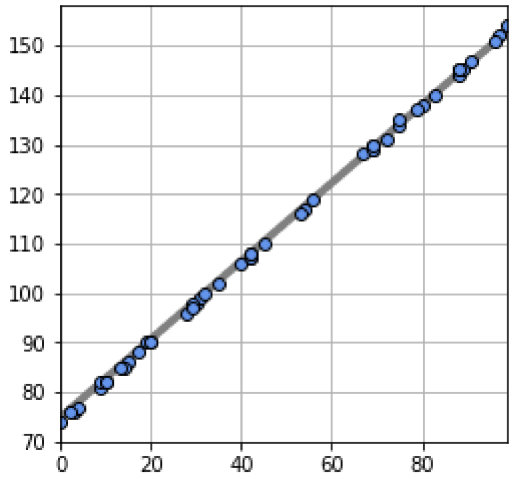
```
반복 횟수 21128
W=[0.788680, 74.891279]
dMSE=[-0.007498, 0.499976]
MSE=0.384079
```



In [8]:

```
# 리스트 5-1-(10)
# 선 표시 ---------------------------------
def show_line(w):
    xb = np.linspace(X_min, X_max, 100)
    y = w[0] * xb + w[1]
    plt.plot(xb, y, color=(.5, .5, .5), linewidth=4)


# 메인 -------------------------------------
plt.figure(figsize=(4, 4))
W=np.array([W0, W1])
mse = mse_line(X, T, W)
print("w0={0:.3f}, w1={1:.3f}".format(W0, W1))
# mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=0.789, w1=74.891
SD=0.620 cm
```



경사 하강법은 반복 계산에 의해 근사값을 구하는 수치 계산법.

이러한 풀이를 수치해

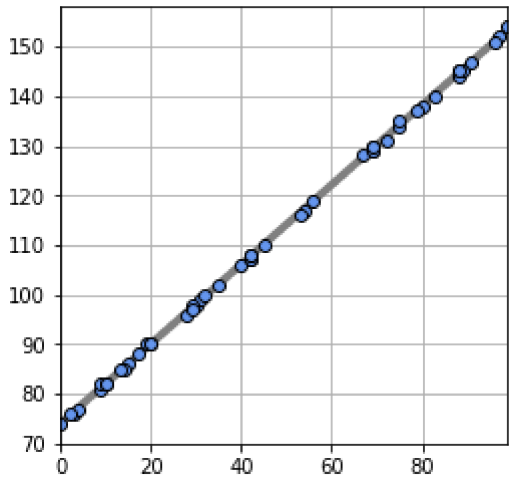
직선 모델의 경우는 근사적인 해석이 아니라 방정식을 해결하여 정확한 해를 구할수 있읍니다.

이러한 풀이를 해석해라 함.

In [9]:

```
# 리스트 5-1-(11)
# 해석해 ------------------------------------
def fit_line(x, t):
    mx = np.mean(x)
    mt = np.mean(t)
    mtx = np.mean(t * x)
    mxx = np.mean(x * x)
    w0 = (mtx - mt * mx) / (mxx - mx**2)
    w1 = mt - w0 * mx
    return np.array([w0, w1])

# 메인 ------------------------------------
W = fit_line(X, T)
print("w0={0:.3f}, w1={1:.3f}".format(W[0], W[1]))
mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.figure(figsize=(4, 4))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=0.801, w1=74.054
SD=0.418 cm
```



In [ ]:

9.2 CDS 조도 센서

지도 학습:회귀 실습(led light--> photo sensor lux확인)

photo sensor: cds + MCP3002(SPI ADC Driver)

주변 조도에따라 영향을 많이 받고, 그에 따라 설정값을 변경해야하는 경우가 많이 발생한다.

그에따라 관련값을 변경해가면서 정상적인 결과가 나올수있도록 조정하기 바랍니다.

참고로 이하의 TEST는 2021.05.22. 18:00에 진행한 결과이다.

In [1]:

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

import time
import spidev
import RPi.GPIO as GPIO

# GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT)

spi_ch = 0
# Enable SPI
spi = spidev.SpiDev(0, spi_ch)
spi.max_speed_hz = 1200000

def read_adc(adc_ch, vref = 3.3):

    # Make sure ADC channel is 0 or 1
    if adc_ch != 0:
        adc_ch = 1

    # Construct SPI message
    #  First bit (Start): Logic high (1)
    #  Second bit (SGL/DIFF): 1 to select single mode
    #  Third bit (ODD/SIGN): Select channel (0 or 1)
    #  Fourth bit (MSFB): 0 for LSB first
    #  Next 12 bits: 0 (don't care)
    msg = 0b11
    msg = ((msg << 1) + adc_ch) << 5
    msg = [msg, 0b00000000]
    reply = spi.xfer2(msg)

    # Construct single integer out of the reply (2 bytes)
    adc = 0
    for n in reply:
        adc = (adc << 8) + n
    # Last bit (0) is not part of ADC value, shift to remove it
    adc = adc >> 1
    # Calculate voltage form ADC value
    voltage = (vref * adc) / 1024
#     voltage = (vref * adc)/20
    return voltage

light = GPIO.PWM(12, 1000)  # channel=12 frequency=50Hz

X_min = 0
X_max = 0
```

```
T_min = 0
T_max = 0
X_n = 50
np.random.seed(seed=1) # 난수를 고정

T = np.zeros(X_n)
X = np.random.rand(X_n)*100
X = np.round(X,0)

# Report the channel 0 and channel 1 voltages to the terminal
try:
    light.start(0)
    count = 0
    finish = False
    light.ChangeDutyCycle(100)
    time.sleep(1)

    while not finish:
#         light.ChangeDutyCycle(100)
        light.ChangeDutyCycle(100 - X[count])
        time.sleep(0.5)
        T[count] = read_adc(0)
#         T[count]  = read_adc(1)
#         time.sleep(1.0)
        if (count%10) == 0:
            print("count: {0}, Light Level: {1: .1f}, pwm: {2: .1f}".format(count, T[c
        count += 1

        if count >= X_n:
            finish = True
            X_min = int(X.min())
            X_max = int(X.max())
            T_min = int(T.min())
            T_max = int(T.max())
            print('photo max:{0: .1f}, min:{1: .1f}'.format(T.max(), T.min()))
            print('led: max:{0: .1f}, min:{1: .1f}'.format(X_max, X_min))
            print('End of measure')
            np.savez('photo1.npz', X=X, X_min=X_min, X_max=X_max, X_n=X_n, T=T, T_min

finally:
    GPIO.cleanup()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: RuntimeWarning: This
channel is already in use, continuing anyway.  Use GPIO.setwarnings(False) to disable
warnings.
  # This is added back by InteractiveShellApp.init_path()
count: 0, Light Level:  2.7, pwm:  42.0
count: 10, Light Level:  2.7, pwm:  42.0
count: 20, Light Level:  2.3, pwm:  80.0
count: 30, Light Level:  3.2, pwm:  10.0
count: 40, Light Level:  2.2, pwm:  99.0
photo max: 3.3, min: 2.2
led: max: 99.0, min: 0.0
End of measure
```

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
outfile = np.load('photo1.npz')

X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']
```

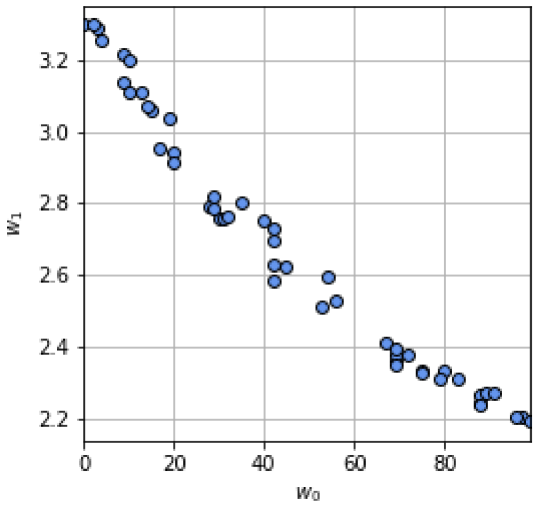
```
T_min = outfile['T_min']
T_max = outfile['T_max']

# 리스트 5-1-(2)
print(X)
# 리스트 5-1-(4)
print(np.round(T, 2))
```

```
[42. 72.  0. 30. 15.  9. 19. 35. 40. 54. 42. 69. 20. 88.  3. 67. 42. 56.
 14. 20. 80. 97. 31. 69. 88. 89.  9.  4. 17. 88. 10. 42. 96. 53. 69. 32.
 69. 83.  2. 75. 99. 75. 28. 79. 10. 45. 91. 29. 29. 13.]
[2.69 2.38 3.3  2.76 3.06 3.21 3.04 2.8  2.75 2.59 2.73 2.36 2.94 2.24
 3.29 2.41 2.63 2.53 3.07 2.91 2.33 2.2  2.76 2.38 2.27 2.27 3.14 3.25
 2.95 2.24 3.2  2.58 2.2  2.51 2.35 2.76 2.39 2.31 3.3  2.33 2.19 2.33
 2.79 2.31 3.11 2.62 2.27 2.82 2.78 3.11]
```

In [3]:

```
# 리스트 5-1-(5)
# 데이터 그래프 ----------------------------
plt.figure(figsize=(4, 4))
plt.plot(X, T, marker='o', linestyle='None',
         markeredgecolor='black', color='cornflowerblue')
plt.xlim(X_min, X_max)
print("min: {0}, max:{1}".format(X_min, X_max))
# plt.xlim(0, 99)
plt.grid(True)
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')

plt.show()
```

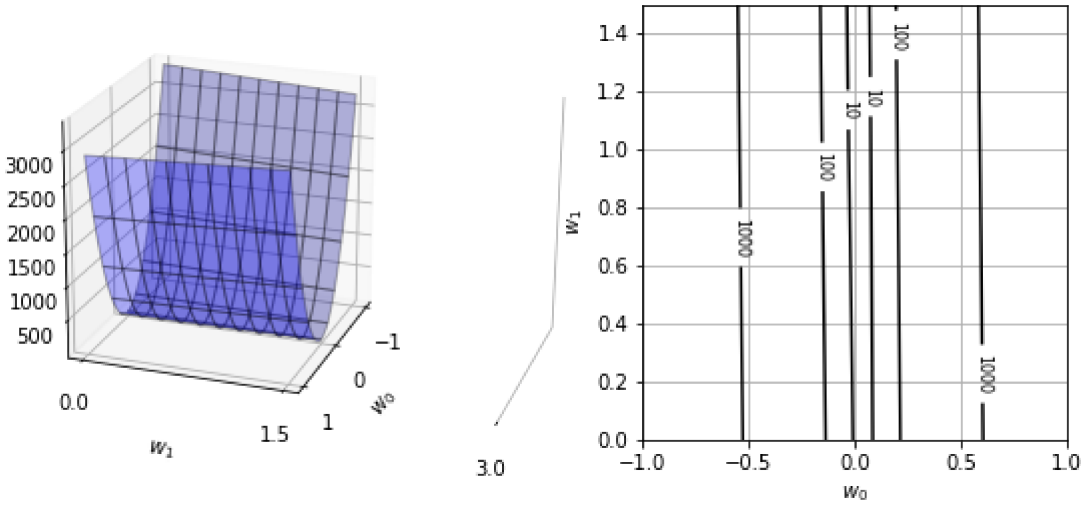
```
min: 0, max:99
```



In [11]:

```
# 리스트 5-1-(6)
from mpl_toolkits.mplot3d import Axes3D
# 평균 오차 함수 ----------------------------
def mse_line(x, t, w):
    y = w[0] * x + w[1]
    mse = np.mean((y - t)**2)
    return mse

# 계산 ----------------------------------
xn = 100 # 등고선 표시 해상도
w0_range = [-1, 1]
w1_range = [0, T_max/2]

x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
```

```
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))

for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))

# 표시 ------------------------------------
plt.figure(figsize=(9.5, 4))
plt.subplots_adjust(wspace=0.5)

ax = plt.subplot(1, 2, 1, projection='3d')
ax.plot_surface(xx0, xx1, J, rstride=10, cstride=10, alpha=0.3,
                color='blue', edgecolor='black')
ax.set_xticks([1, 0, -1])
ax.set_yticks([0, T_max/2, T_max])
ax.view_init(20, 20)
plt.xlabel('$w_0$')
plt.ylabel('$w_1$')

plt.subplot(1, 2, 2)
cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=[10, 100, 1000, 10000])
cont.clabel(fmt='%1.0f', fontsize=8)

plt.xlabel('$w_0$')
plt.ylabel('$w_1$')

plt.grid(True)
plt.show()
```



In [12]:

```
# 리스트 5-1-(7)
# 평균 제곱 오차의 기울기 -----------------------
def dmse_line(x, t, w):
    y = w[0] * x + w[1]
    d_w0 = 2 * np.mean((y - t) * x)
    d_w1 = 2 * np.mean(y - t)
    return d_w0, d_w1
```

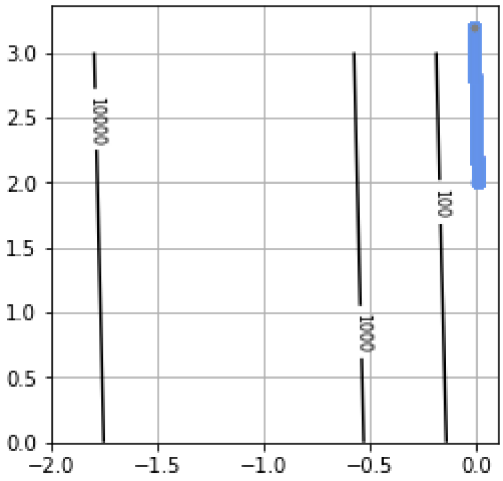
In [13]:

```
# 리스트 5-1-(8)
d_w = dmse_line(X, T, [1, 100])
print(np.round(d_w, 1))
```

```
[15359.1   288.2]
```

In [18]:
```
# 리스트 5-1-(9)
# 구배법 ----------------------------------
def fit_line_num(x, t):
    w_init = [X_min, T_min] # 초기 매개 변수
    alpha = 0.0001 # 학습률
    i_max = 100000 # 반복의 최대 수
    eps = 0.0005 # 반복을 종료 기울기의 절대 값의 한계
    w_i = np.zeros([i_max, 2])
    w_i[0, :] = w_init
    for i in range(1, i_max):
        dmse = dmse_line(x, t, w_i[i - 1])
        w_i[i, 0] = w_i[i - 1, 0] - alpha * dmse[0]
        w_i[i, 1] = w_i[i - 1, 1] - alpha * dmse[1]
        if max(np.absolute(dmse)) < eps: # 종료판정, np.absolute는 절대치
            break
    w0 = w_i[i, 0]
    w1 = w_i[i, 1]
    w_i = w_i[:i, :]
    return w0, w1, dmse, w_i


# 메인 ------------------------------------
plt.figure(figsize=(4, 4)) # MSE의 등고선 표시
xn = 100 # 등고선 해상도
w0_range = [-2, 0]
w1_range = [0, T_max]
x0 = np.linspace(w0_range[0], w0_range[1], xn)
x1 = np.linspace(w1_range[0], w1_range[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
J = np.zeros((len(x0), len(x1)))
for i0 in range(xn):
    for i1 in range(xn):
        J[i1, i0] = mse_line(X, T, (x0[i0], x1[i1]))

cont = plt.contour(xx0, xx1, J, 30, colors='black',
                   levels=(100, 1000, 10000, 100000))
cont.clabel(fmt='%1.0f', fontsize=8)
plt.grid(True)
# 구배법 호출
W0, W1, dMSE, W_history = fit_line_num(X, T)

# 결과보기
print('반복 횟수 {0}'.format(W_history.shape[0]))
print('W=[{0:.6f}, {1:.6f}]'.format(W0, W1))
print('dMSE=[{0:.6f}, {1:.6f}]'.format(dMSE[0], dMSE[1]))
print('MSE={0:.6f}'.format(mse_line(X, T, [W0, W1])))
plt.plot(W_history[:, 0], W_history[:, 1], '.-',
         color='gray', markersize=10, markeredgecolor='cornflowerblue')
plt.show()
```

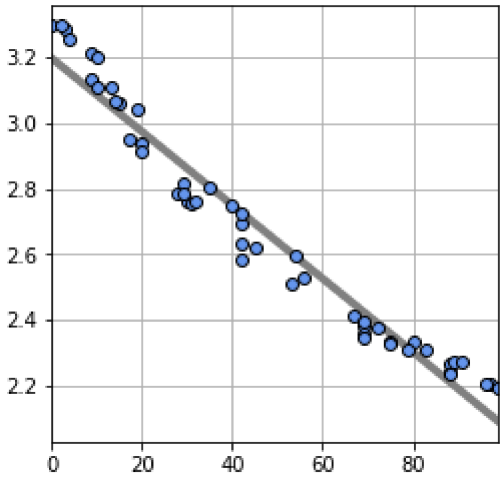
```
반복 횟수 99999
W=[-0.011205, 3.197568]
dMSE=[0.000027, -0.001824]
MSE=0.005014
```



In [19]:

```
# 리스트 5-1-(10)
# 선 표시 ---------------------------------
def show_line(w):
    xb = np.linspace(X_min, X_max, 100)
    y = w[0] * xb + w[1]
    plt.plot(xb, y, color=(.5, .5, .5), linewidth=4)


# 메인 -------------------------------------
plt.figure(figsize=(4, 4))
W=np.array([W0, W1])
mse = mse_line(X, T, W)
print("w0={0:.3f}, w1={1:.3f}".format(W0, W1))
# mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=-0.011, w1=3.198
SD=0.071 cm
```



경사 하강법은 반복 계산에 의해 근사값을 구하는 수치 계산법.

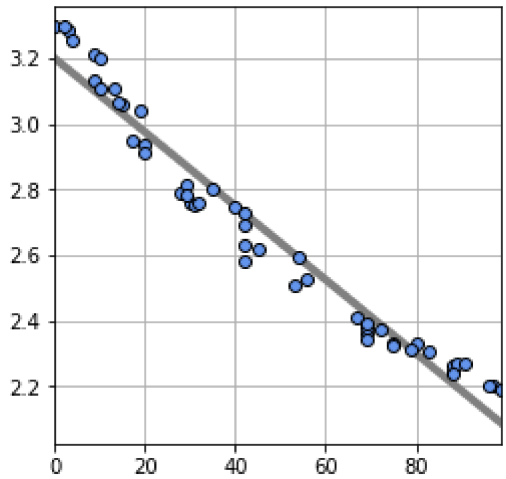
이러한 풀이를 수치해

직선 모델의 경우는 근사적인 해석이 아니라 방정식을 해결하여 정확한 해를 구할수 있읍니다.

이러한 풀이를 해석해라 함.

In [20]:
```
# 리스트 5-1-(11)
# 해석해 ------------------------------------
def fit_line(x, t):
    mx = np.mean(x)
    mt = np.mean(t)
    mtx = np.mean(t * x)
    mxx = np.mean(x * x)
    w0 = (mtx - mt * mx) / (mxx - mx**2)
    w1 = mt - w0 * mx
    return np.array([w0, w1])

# 메인 ------------------------------------
W = fit_line(X, T)
print("w0={0:.3f}, w1={1:.3f}".format(W[0], W[1]))
mse = mse_line(X, T, W)
print("SD={0:.3f} cm".format(np.sqrt(mse)))
plt.figure(figsize=(4, 4))
show_line(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```

```
w0=-0.011, w1=3.201
SD=0.071 cm
```



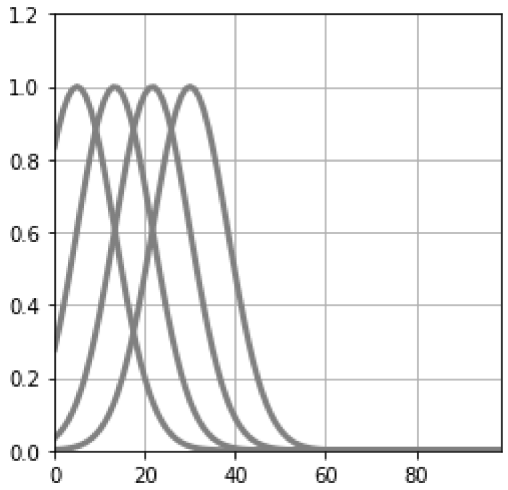
선형 기저함수 모델

가우스 함수

In [21]:
```
# --- 리스트 5-2-(1)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


# 데이터 로드 ---------------------------
outfile = np.load('photo1.npz')
X = outfile['X']
X_min = outfile['X_min']
X_max = outfile['X_max']
X_n = outfile['X_n']
T = outfile['T']
```

In [22]:
```
# --- 리스트 5-2-(2)
# 가우스 함수 --------------------------------
def gauss(x, mu, s):
    return np.exp(-(x - mu)**2 / (2 * s**2))
```

In [23]:
```
# 리스트 5-2-(3)
# 메인 ----------------------------------
M = 4
plt.figure(figsize=(4, 4))
mu = np.linspace(5, 30, M)
s = mu[1] - mu[0] # (A)
xb = np.linspace(X_min, X_max, 100)
for j in range(M):
    y = gauss(xb, mu[j], s)
    plt.plot(xb, y, color='gray', linewidth=3)
plt.grid(True)
plt.xlim(X_min, X_max)
plt.ylim(0, 1.2)
plt.show()
```



In [24]:
```
# 리스트 5-2-(4)
# 선형 기저 함수 모델 ----------------
def gauss_func(w, x):
    m = len(w) - 1
    mu = np.linspace(5, 30, m)
    s = mu[1] - mu[0]
    y = np.zeros_like(x) # x와 같은 크기로 요소가 0의 행렬 y를 작성
    for j in range(m):
        y = y + w[j] * gauss(x, mu[j], s)
    y = y + w[m]
    return y
```

In [25]:
```
# 리스트 5-2-(5)
# 선형 기저 함수 모델 MSE ----------------
def mse_gauss_func(x, t, w):
    y = gauss_func(w, x)
    mse = np.mean((y - t)**2)
    return mse
```

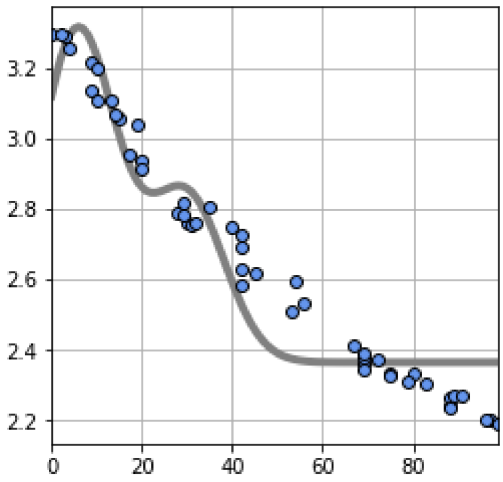
In [26]:
```
# 리스트 5-2-(6)
# 선형 기저 함수 모델 정확한 솔루션 ----------------
def fit_gauss_func(x, t, m):
    mu = np.linspace(5, 30, m)
```

```
    s = mu[1] - mu[0]
    n = x.shape[0]
    psi = np.ones((n, m+1))
    for j in range(m):
        psi[:, j] = gauss(x, mu[j], s)
    psi_T = np.transpose(psi)


    b = np.linalg.inv(psi_T.dot(psi))
    c = b.dot(psi_T)
    w = c.dot(t)
    return w
```

In [27]:

```
# 리스트 5-2-(7)
# 가우스 기저 함수 표시 ----------------------
def show_gauss_func(w):
    xb = np.linspace(X_min, X_max, 100)
    y = gauss_func(w, xb)
    plt.plot(xb, y, c=[.5, .5, .5], lw=4)


# 메인 ----------------------------------
plt.figure(figsize=(4, 4))
M = 4
W = fit_gauss_func(X, T, M)
show_gauss_func(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_gauss_func(X, T, W)
print('W='+ str(np.round(W,1)))
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

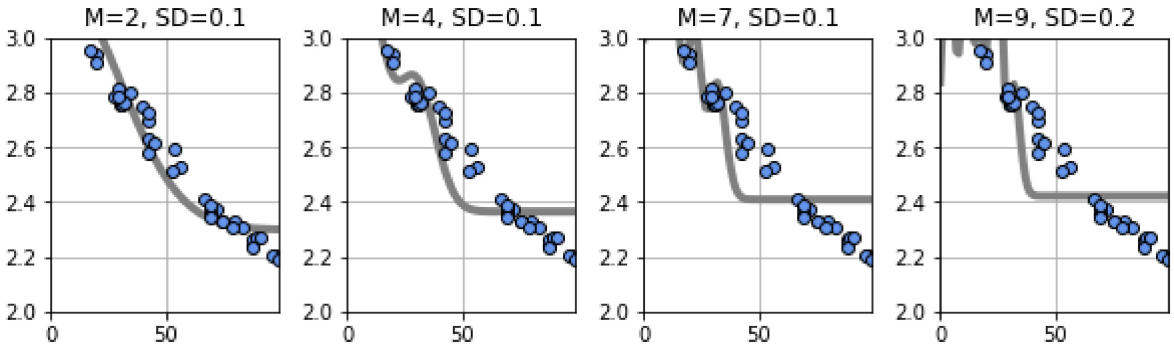
```
W=[0.9 0.1 0.  0.5 2.4]
SD=0.10 cm
```



In [28]:

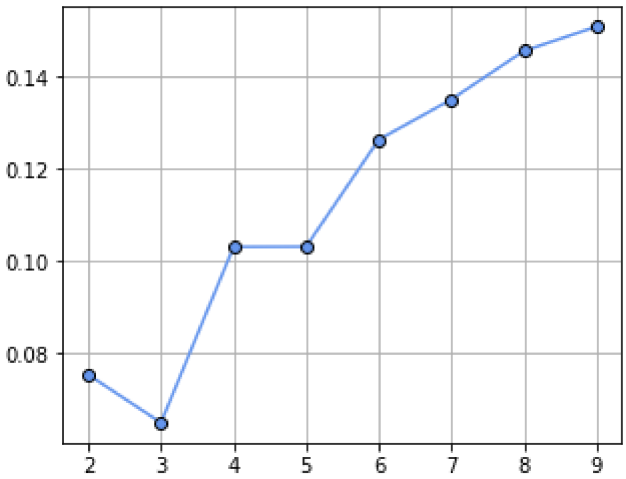
```
# 리스트 5-2-(8)
plt.figure(figsize=(10, 2.5))
plt.subplots_adjust(wspace=0.3)
M = [2, 4, 7, 9]
for i in range(len(M)):
    plt.subplot(1, len(M), i + 1)
    W = fit_gauss_func(X, T, M[i])
    show_gauss_func(W)
    plt.plot(X, T, marker='o', linestyle='None',
```

```
                color='cornflowerblue', markeredgecolor='black')
    plt.xlim(X_min, X_max)
    plt.grid(True)
    plt.ylim(T_min, T_max)
    mse = mse_gauss_func(X, T, W)


    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```



In [29]:

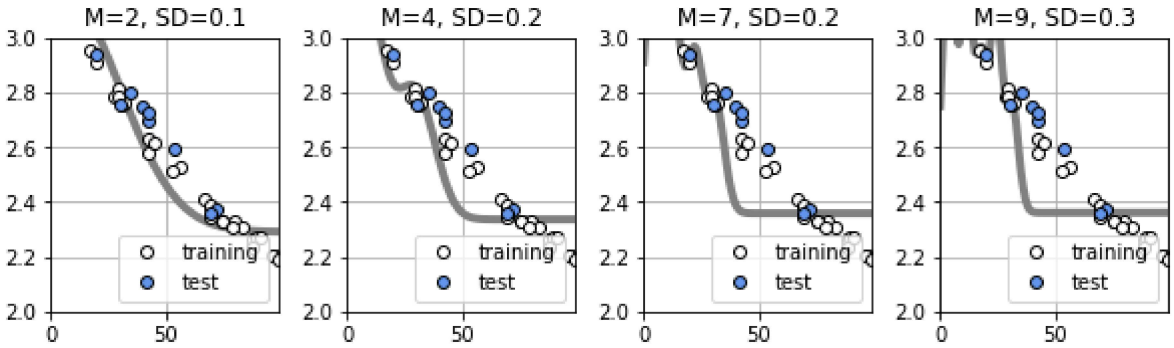
```
# 리스트 5-2-(9)
plt.figure(figsize=(5, 4))
M = range(2, 10)
mse2 = np.zeros(len(M))
for i in range(len(M)):
    W = fit_gauss_func(X, T, M[i])
    mse2[i] = np.sqrt(mse_gauss_func(X, T, W))
plt.plot(M, mse2, marker='o',
         color='cornflowerblue', markeredgecolor='black')
plt.grid(True)
plt.show()
```



In [30]:

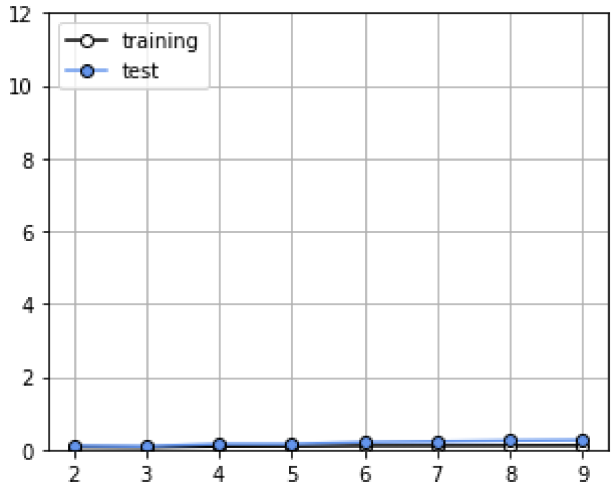
```
# 리스트 5-2-(10)
# 훈련 데이터와 테스트 데이터 ------------------
X_test = X[:int(X_n / 4 + 1)]
T_test = T[:int(X_n / 4 + 1)]
X_train = X[int(X_n / 4 + 1):]
T_train = T[int(X_n / 4 + 1):]
# 메인 ----------------------------------
plt.figure(figsize=(10, 2.5))


plt.subplots_adjust(wspace=0.3)
M = [2, 4, 7, 9]
```

```
for i in range(len(M)):
    plt.subplot(1, len(M), i + 1)
    W = fit_gauss_func(X_train, T_train, M[i])
    show_gauss_func(W)
    plt.plot(X_train, T_train, marker='o',
             linestyle='None', color='white',
             markeredgecolor='black', label='training')
    plt.plot(X_test, T_test, marker='o', linestyle='None',
             color='cornflowerblue',
             markeredgecolor='black', label='test')
    plt.legend(loc='lower right', fontsize=10, numpoints=1)
    plt.xlim(X_min, X_max)
    plt.ylim(T_min, T_max)
    plt.grid(True)
    mse = mse_gauss_func(X_test, T_test, W)
    plt.title("M={0:d}, SD={1:.1f}".format(M[i], np.sqrt(mse)))
plt.show()
```



In [31]:

```
# 리스트 5-2-(11)
plt.figure(figsize=(5, 4))
M = range(2, 10)
mse_train = np.zeros(len(M))
mse_test = np.zeros(len(M))
for i in range(len(M)):
    W = fit_gauss_func(X_train, T_train, M[i])
    mse_train[i] = np.sqrt(mse_gauss_func(X_train, T_train, W))
    mse_test[i] = np.sqrt(mse_gauss_func(X_test, T_test, W))
plt.plot(M, mse_train, marker='o', linestyle='-',
         markerfacecolor='white', markeredgecolor='black',
         color='black', label='training')
plt.plot(M, mse_test, marker='o', linestyle='-',
         color='cornflowerblue', markeredgecolor='black',
         label='test')
plt.legend(loc='upper left', fontsize=10)
plt.ylim(0, 12)
plt.grid(True)
plt.show()
```



In [32]:
```
# 리스트 5-2-(12)
# K 분할 교차 검증 ----------------------------
def kfold_gauss_func(x, t, m, k):
    n = x.shape[0]
    mse_train = np.zeros(k)
    mse_test = np.zeros(k)
    for i in range(0, k):
        x_train = x[np.fmod(range(n), k) != i] # (A)
        t_train = t[np.fmod(range(n), k) != i] # (A)
        x_test = x[np.fmod(range(n), k) == i] # (A)
        t_test = t[np.fmod(range(n), k) == i] # (A)
        wm = fit_gauss_func(x_train, t_train, m)
        mse_train[i] = mse_gauss_func(x_train, t_train, wm)
        mse_test[i] = mse_gauss_func(x_test, t_test, wm)
    return mse_train, mse_test
```

In [33]:
```
# 리스트 5-2-(13)
np.fmod(range(10),5)
```

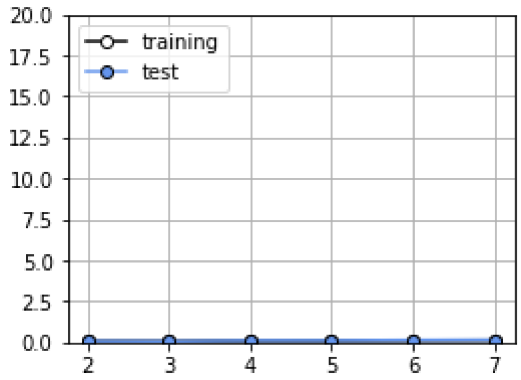
Out[33]:
```
array([0, 1, 2, 3, 4, 0, 1, 2, 3, 4], dtype=int32)
```

In [34]:
```
# 리스트 5-2-(14)
M = 4
K = 4
kfold_gauss_func(X, T, M, K)
```

Out[34]:
```
(array([0.00978268, 0.00906765, 0.00923018, 0.01296746]),
 array([0.01404229, 0.01519937, 0.02644996, 0.00440227]))
```

In [35]:
```
# 리스트 5-2-(15)
M = range(2, 8)
K = 16
Cv_Gauss_train = np.zeros((K, len(M)))
Cv_Gauss_test = np.zeros((K, len(M)))
for i in range(0, len(M)):
    Cv_Gauss_train[:, i], Cv_Gauss_test[:, i] =W
                    kfold_gauss_func(X, T, M[i], K)
mean_Gauss_train = np.sqrt(np.mean(Cv_Gauss_train, axis=0))
mean_Gauss_test = np.sqrt(np.mean(Cv_Gauss_test, axis=0))


plt.figure(figsize=(4, 3))
plt.plot(M, mean_Gauss_train, marker='o', linestyle='-',
         color='k', markerfacecolor='w', label='training')
```

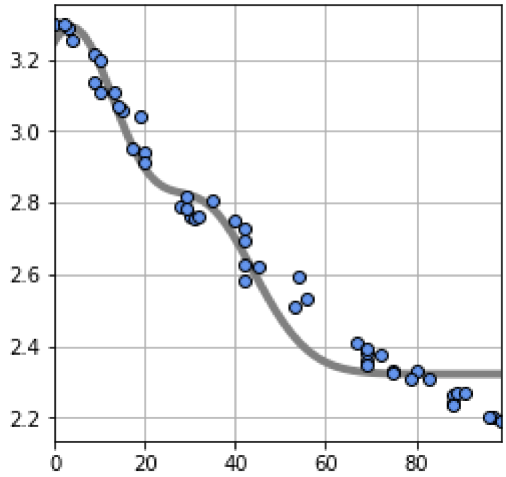
```
plt.plot(M, mean_Gauss_test, marker='o', linestyle='-',
         color='cornflowerblue', markeredgecolor='black', label='test')
plt.legend(loc='upper left', fontsize=10)
plt.ylim(0, 20)
plt.grid(True)
plt.show()
```



In [36]:

```
# 리스트 5-2-(16)
M = 3
plt.figure(figsize=(4, 4))
W = fit_gauss_func(X, T, M)
show_gauss_func(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue', markeredgecolor='black')
plt.xlim([X_min, X_max])
plt.grid(True)
mse = mse_gauss_func(X, T, W)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

SD=0.07 cm



모델A

In [37]:

```
# 리스트 5-2-(17)
# 모델 A ---------------------------------
def model_A(x, w):
    y = w[0] - w[1] * np.exp(-w[2] * x)
    return y


# 모델 A 표시 -----------------------------
def show_model_A(w):
    xb = np.linspace(X_min, X_max, 100)
```

```
    y = model_A(xb, w)
    plt.plot(xb, y, c=[.5, .5, .5], lw=4)


# 모델 A의 MSE ----------------------------
def mse_model_A(w, x, t):
    y = model_A(x, w)
    mse = np.mean((y - t)**2)
    return mse
```

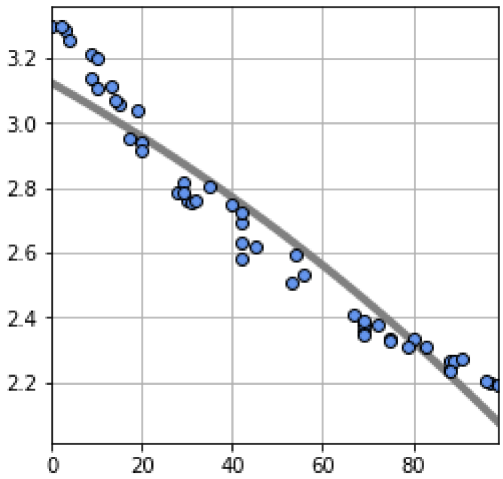
In [38]:

```
# 리스트 5-2-(18)
from scipy.optimize import minimize


# 모델 A의 매개 변수 최적화 -----------------
def fit_model_A(w_init, x, t):
    res1 = minimize(mse_model_A, w_init, args=(x, t), method="powell")
    return res1.x
```

In [39]:

```
# 리스트 5-2-(19)
# 메인 ------------------------------------
plt.figure(figsize=(4, 4))
W_init=[100, 0, 0]
W = fit_model_A(W_init, X, T)
print("w0={0:.1f}, w1={1:.1f}, w2={2:.1f}".format(W[0], W[1], W[2]))
show_model_A(W)
plt.plot(X, T, marker='o', linestyle='None',
         color='cornflowerblue',markeredgecolor='black')
plt.xlim(X_min, X_max)
plt.grid(True)
mse = mse_model_A(W, X, T)
print("SD={0:.2f} cm".format(np.sqrt(mse)))
plt.show()
```

```
w0=4.5, w1=1.3, w2=-0.0
SD=0.09 cm
```

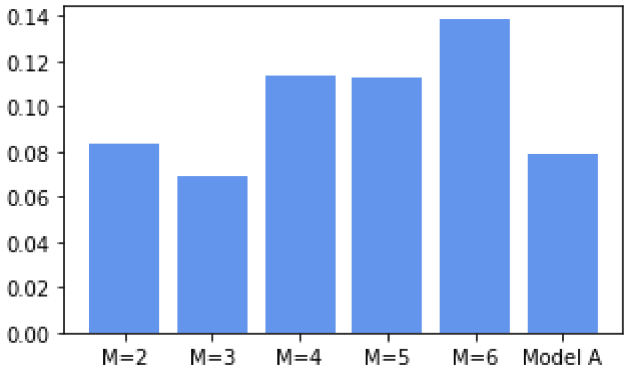


In [40]:

```
# 리스트 5-2-(20)
# 교차 검증 model_A -------------------------
def kfold_model_A(x, t, k):
    n = len(x)
    mse_train = np.zeros(k)
    mse_test = np.zeros(k)
    for i in range(0, k):
        x_train = x[np.fmod(range(n), k) != i]
        t_train = t[np.fmod(range(n), k) != i]
```

```
        x_test = x[np.fmod(range(n), k) == i]
        t_test = t[np.fmod(range(n), k) == i]
        wm = fit_model_A(np.array([169, 113, 0.2]), x_train, t_train)
        mse_train[i] = mse_model_A(wm, x_train, t_train)
        mse_test[i] = mse_model_A(wm, x_test, t_test)
    return mse_train, mse_test


# 메인 ----------------------------------
K = 16
Cv_A_train, Cv_A_test = kfold_model_A(X, T, K)
mean_A_test = np.sqrt(np.mean(Cv_A_test))
print("Gauss(M=3) SD={0:.2f} cm".format(mean_Gauss_test[1]))
print("Model A SD={0:.2f} cm".format(mean_A_test))
SD = np.append(mean_Gauss_test[0:5], mean_A_test)
M = range(6)
label = ["M=2", "M=3", "M=4", "M=5", "M=6", "Model A"]
plt.figure(figsize=(5, 3))
plt.bar(M, SD, tick_label=label, align="center",
facecolor="cornflowerblue")
plt.show()
```

```
Gauss(M=3) SD=0.07 cm
Model A SD=0.08 cm
```



In [ ]: