

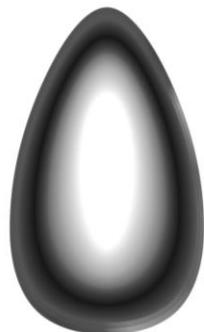


파이썬으로 익히는 딥러닝 기초
(Deep Learning with Python)

2017. 4.

강병호수석

Contents



- I. Environment Setup
- II. Perceptron
- III. Neural Network
- IV. Training
- V. Back Propagation
- VI. Training Techniques
- VII. Convolution Neural Network
- VIII. Deep Learning

작성자 소개



강병호 수석

medit74@sk.com

▪ 학력

- 1993년 숭실대학교 인공지능학과 입학
- 2000년 숭실대학교 컴퓨터학부 졸업 (학사)

▪ 근무 경력

- 2000년 SKC&C(현, SK주식회사) 입사
- 국내 프로젝트 수행
 - 2000년 ~ 2003년 SK Telecom 웹 프로젝트
 - 2010년 ~ 2014년 SK Telecom 웹/모바일 프로젝트
- 미국 프로젝트 수행
 - 2005년 ~ 2009년 Virgin Mobile USA 프로젝트
 - 2009년 ~ 2010년 미국 Mobile Banking 프로젝트
- 중국 프로젝트 수행
 - 2015년 ~ 2016년 SKC&C 중국법인 역량개발 총괄
- 베트남 프로젝트 수행
 - 2003년 ~ 2004년 베트남 S Telecom 영업시스템 운영

▪ 기타 경력

- 회사 사내 강사 (2012년~)
 - Spring Framework, Java Web Programming 등 다수
- 국가 자격증 문제 출제 위원 (Pilot, 2014년)
 - 한국소프트웨어기술협회(KOSTA) SW개발자 L4 자격

I. Environment

Setup



1. *Environment Overview*
2. **Python**
3. **Anaconda**
4. **Numpy**
5. **Matplotlib**
- b. **PyDev**
7. **Hands-on**

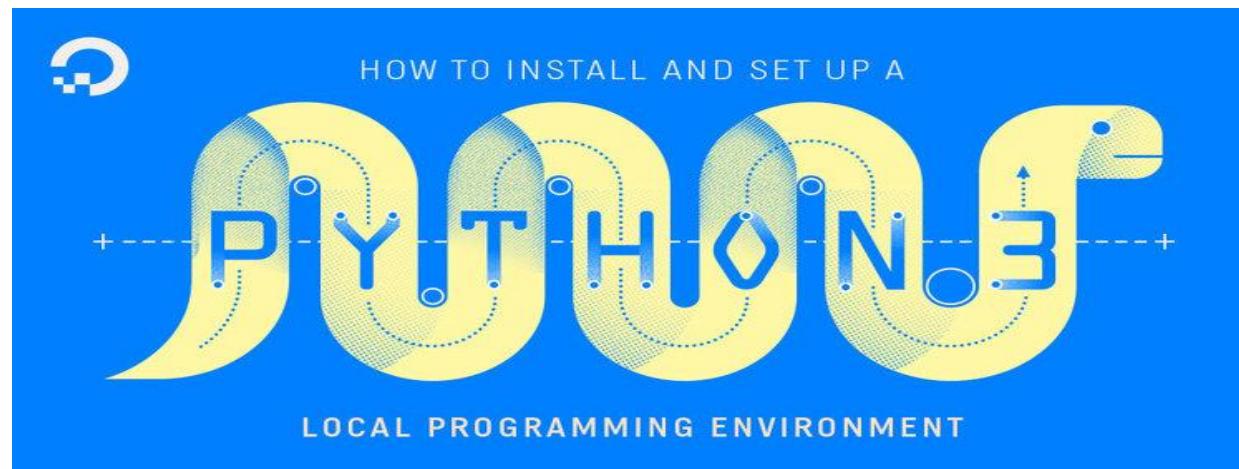
1. Environment Overview

I. Environment Setup

Python을 이용하여 딥러닝을 구현하기 위한 환경 설정.

- 이 과정의 학습을 위해 필요한 S/W 및 라이브러리

S/W / Lib	Description	Version	비고
Python	간단하고 배우기 쉬운 프로그래밍 언어 기계학습과 데이터 과학 분야에서 널리 사용	3.6.0	Anaconda에 포함
Anaconda	데이터 분석에 중점을 둔 배포판, 즉 사용자가 설치를 한 번에 수행 할 수 있도록 필요한 라이브러리 등을 하나로 정리해 놓은 것	4.3.1	
NumPy	수치 계산용 라이브러리	1.11.3	Anaconda에 포함
Matplotlib	그래프를 그려주는 라이브러리	2.0.0	Anaconda에 포함
PIL	Python Image Library	1.1.7	Anaconda에 포함
PyDev	Python개발용 eclipse plug-in	5.6.0	



Python은 간단하고 배우기 쉬운 오픈 소스 프로그래밍 언어

- **Python이란?**

- Python은 1990년 구도 반 로섬(Guido Van Rossum)이 개발한 Interpreter 언어이다. 즉 컴파일 과정이 없다.
- 사전적 의미는 고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀을 의미
- 교육 목적 뿐만 아니라 실무에서도 많이 활용.
- <http://www.python.org>



- **Python의 특징**

- 영어와 유사한 문법으로 프로그램을 작성하는 인간다운 언어
- 문법이 쉬워 빠르게 배울 수 있다.
- 코드는 읽기 쉽고 간결하면서도 성능도 뛰어나다.
- 개발 속도가 매우 빠르다.

"Life is too short, You need python."

- **Python 버전**

- 현재의 Python은 2.x와 3.x 두 가지 버전이 공존.
- 위 두 가지는 서로 호환되지 않음.

Anaconda는 데이터 분석에 중점을 둔 배포판

- **Anaconda란?**

- Python 기반의 Open Data Science Platform
- Python 및 Python Library 등을 하나로 정리해 둔 배포판.
- NumPy와 Matplotlib을 포함해 데이터 분석이 유용한 라이브러리가 포함.
- <http://www.continuum.io>

- **Anaconda Package 주요 List**

- Python 3.6의 경우 453개 패키지를 지원한다.

Name	Version	Summary	In Installer
djanggo	1.10.5	빠른 웹 개발을 위한 프레임워크	
flask	0.12	Micro Web Framework	Yes
jupyter	1.0.0	Python 개발 도구	Yes
matplotlib	2.0.0	그래프를 그려주는 라이브러리	Yes
numpy	1.11.3	수치 계산용 라이브러리	Yes
pandas	0.19.2	강력한 데이터 구조 분석 도구	Yes
python	3.6.0	Python 프로그래밍 언어	Yes
scipy	0.18.1	Python 과학 라이브러리	Yes



ANACONDA MAKES...



**DATA SCIENCE TEAMS
HAPPIER**

That means better and more results

딥러닝 구현 시 배열과 행렬 계산이 많은데 NumPy의 배열 클래스 `numpy.array`에는 이를 위한 method가 많이 있다.

- NumPy 라이브러리 import 하기

```
import numpy as np
```

- NumPy 배열 생성하기

- 1차원 배열 (벡터) 생성하기

```
x1 = np.array([1.0, 2.0, 3.0])
y1 = np.array([5.0, 10.0, 15.0])
```

- 2차원 배열 (행렬) 생성하기

```
x2 = np.array([[1.0, 2.0], [3.0, 4.0]])
y2 = np.array([[5.0, 10.0], [15.0, 20.0]])
```

- NumPy 배열 원소 접근하기

```
print(x1[0])
print(x2[1][0])
```

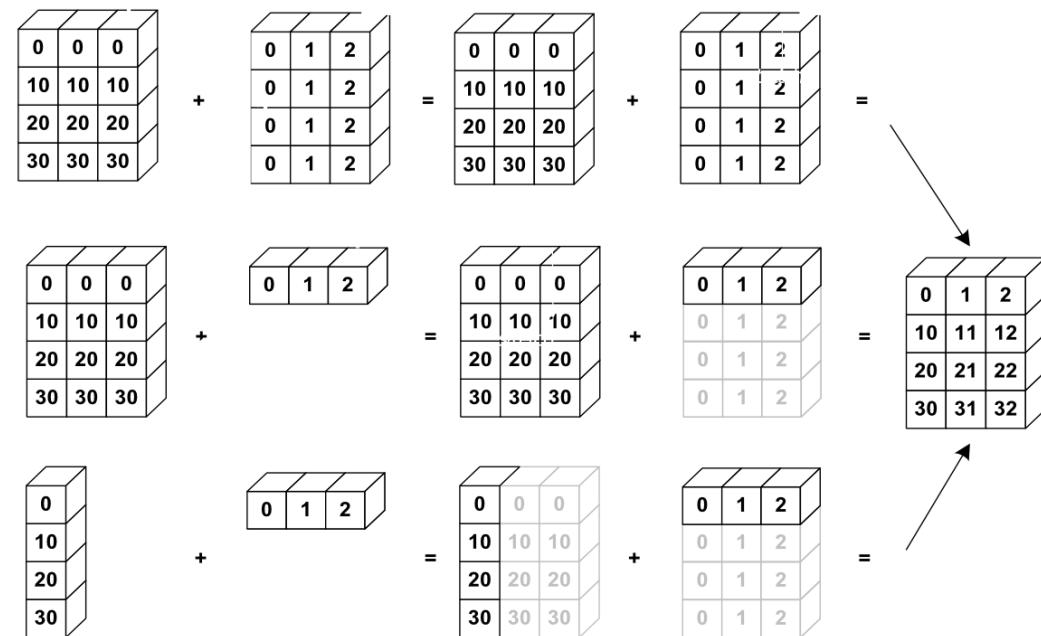
- NumPy 배열 연산하기

- NumPy 배열 산술 연산

```
print(x1 + y1)
print(x1 - y1)
print(x2 + y2)
print(x2 * y2)
```



- NumPy 배열 Broadcast 연산



딥러닝 구현 시 데이터 시각화를 위해 그래프를 그려주는 matplotlib를 사용한다.

- matplotlib 라이브러리 import

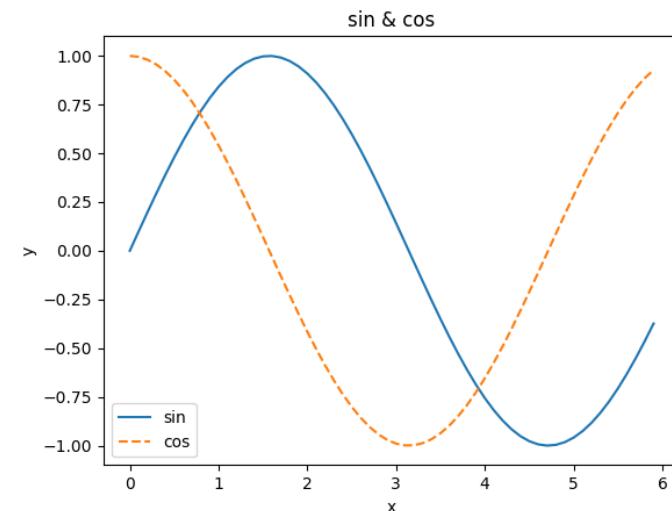
```
import matplotlib.pyplot as plt
#image 모듈에서 imread 함수만 import 함.
from matplotlib.image import imread
```

- 그래프 그리기

```
# 데이터준비
x = np.arange(0, 6, 0.1) #0에서 6까지 0.1 간격으로 배열생성
y1 = np.sin(x)
y2 = np.cos(x)

# 그래프 그리기
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle="--", label="cos")
plt.xlabel("x")
plt.ylabel("y")
plt.title("sin & cos")
plt.legend()
plt.show()
```

```
img = imread("../resources/Scream.jpg")
plt.imshow(img)
plt.show()
```



Eclipse에서 Python 개발 환경을 구축하기 위한 Plug-in

▪ PyDev란?

- Eclipse에서 실행되는 Python 개발환경
- Python, Jython, IronPython 개발을 위해서 사용
- <http://www.pydev.org/>

▪ PyDev 버전

- PyDev5.6.0은 Java 8과 Eclipse 4.6 (Neon)을 필요로 한다.

▪ Eclipse PyDev 설치

- Eclipse Marketplace에서 PyDev로 검색 후 설치
- Eclipse Preferences > PyDev > Interpreters > Python

Interpreters에서 Quick Auto-Config를 통한 Python 인터프리터 라이브러리 연결

▪ Eclipse 일반설정

- Eclipse Preferences > General > Workspace에서 Text file encoding을 UTF-8로 설정
- Eclipse Preferences > General > Appearance > Colors and Fonts에서 Text Font를 Consolas 9 사이즈로 설정



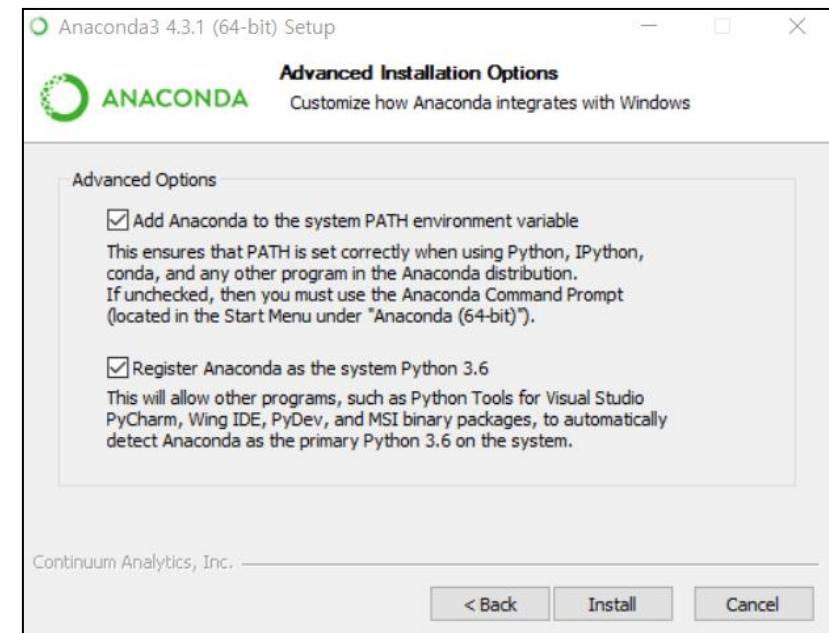
A screenshot of the Eclipse Marketplace interface. It shows a search bar with 'Pydev' typed in, and a result card for 'PyDev - Python IDE for Eclipse 5.6.0'. The card includes a small PyDev logo, a brief description, developer information ('by Brainwy Software, EPL'), and download statistics ('448 installs', '743K installs last month'). A 'More info' link and an 'Installed' button are also visible.

A screenshot of the Eclipse PyDev Preferences window under 'Python Interpreters'. It shows a table with one entry named 'python' located at 'C:\ProgramData\Anaconda3\py...'. To the right are buttons for 'New...', 'Quick Auto-Config', 'Advanced Auto-Config', 'Remove', 'Up', and 'Down'. Below the table are tabs for 'Libraries', 'Forced Builtins', 'Predefined', 'Environment', and 'String Substitution Variables'. Under 'System PYTHONPATH', there is a list of system library paths, each with options for 'New Folder', 'New Egg/Zip(s)', and 'Remove'.

#1 - Anaconda 설치하기

▪ Anaconda 설치

- <http://www.continuum.io> 에서 사용하고 있는 OS에 맞추어 Anaconda 다운로드 후 설치



▪ 설치 후 버전 확인

- 윈도우 명령 프롬프트 실행하여 확인

```
C:\>python --version  
Python 3.6.0 :: Anaconda 4.3.1 (64-bit)
```

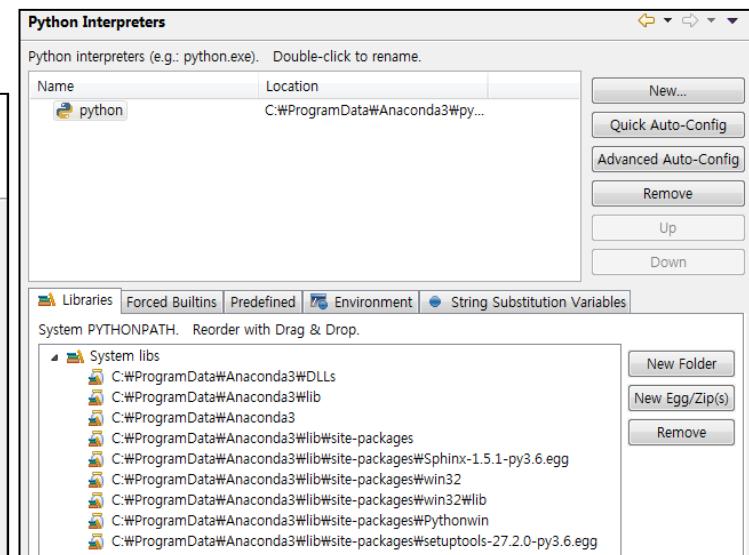
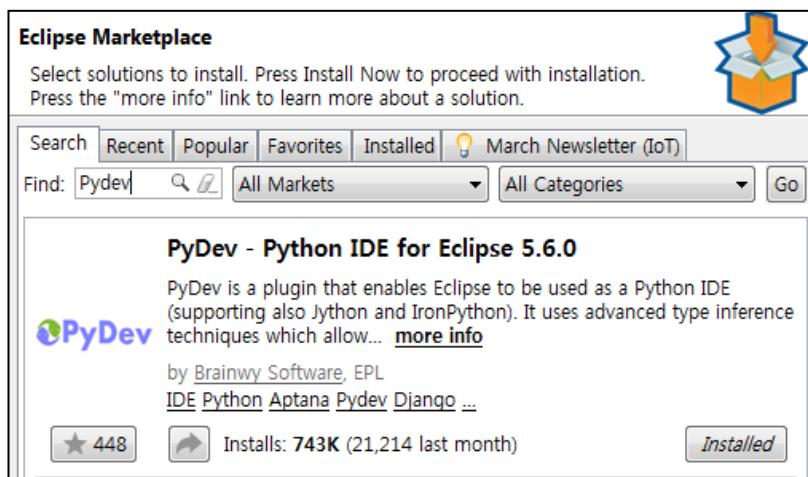
#2 - PyDev 설치하기

▪ 사전조건

- Java Platform (JDK) 8 이상 설치
- Eclipse Neon 이상 설치

▪ PyDev 설치

- Eclipse Marketplace에서 PyDev로 검색 후 설치
- Eclipse Preferences > PyDev > Interpreters > Python Interpreters에서 Quick Auto-Config를 통한 Python 인터프리터 라이브러리 연결



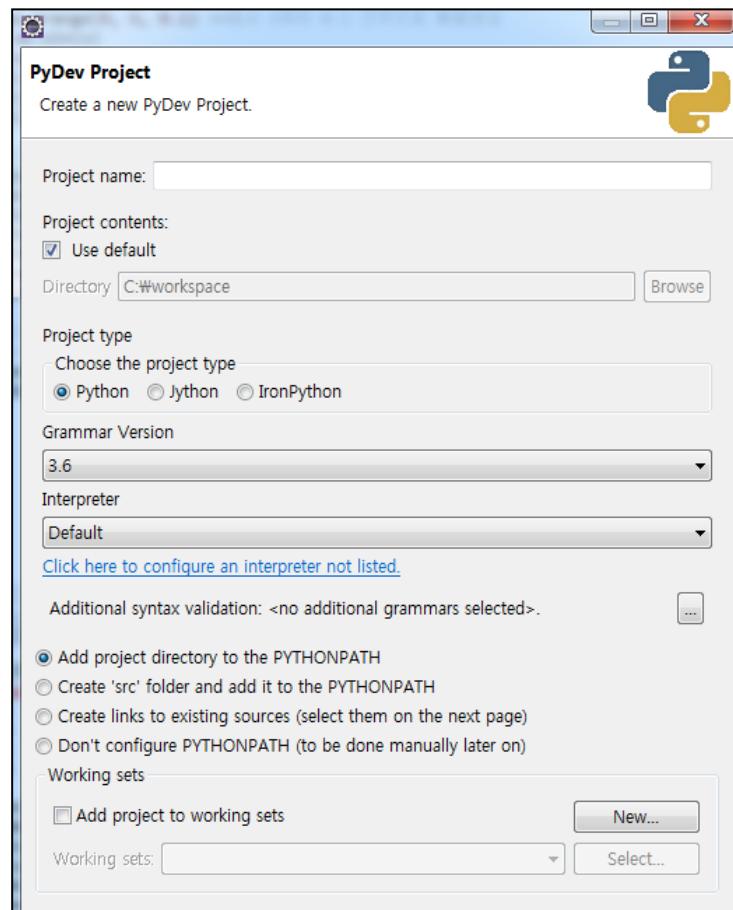
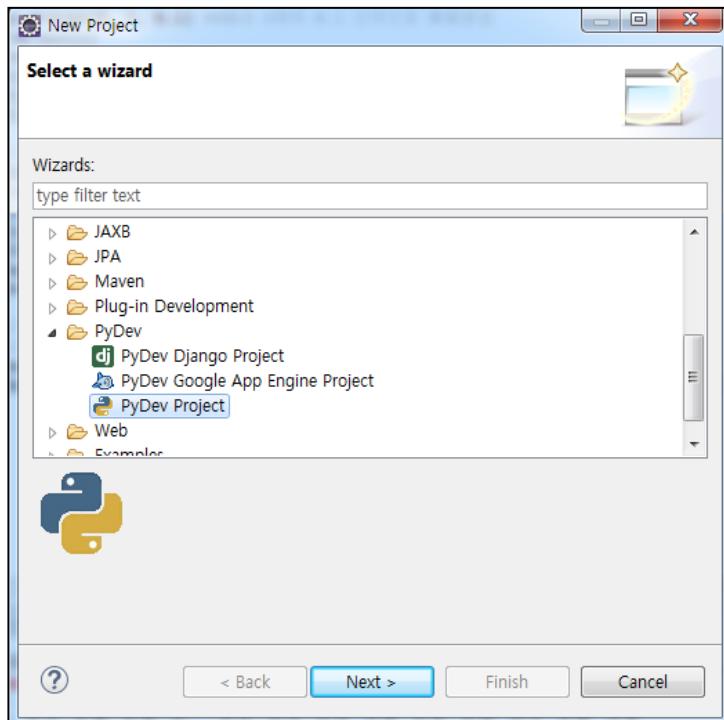
▪ Eclipse 일반설정

- Eclipse Preferences > General > Workspace에서 Text file encoding을 UTF-8로 설정
- Eclipse Preferences > General > Appearance > Colors and Fonts에서 Text Font를 Consolas 9 사이즈로 설정

#3 – Python Project 생성

▪ PyDev Project 생성

- Wizard를 통한 PyDev Project 생성
- Grammar Version을 3.6으로 선택



▪ PyDev Package 생성

- EnvironmentSetup 패키지 생성
- resources 폴더 생성 및 image file 복사 (여기선 Scream.jpg)

#4 - Python 이해하기 - 자료형

- mydatatype.py

```

# Number
print(123)
print(1.23)
print(1 + 2j) #복소수
print(0o34) #8진수
print(0xFF) #16진수

# String
print('Hello')
print("What's your name?")
print('"My name is Tiger" he said')
print("Life is too short.\nYou need python.")
print("I've been to %d countries." % 15)
print("%s is the best country in %d" % ("Swiss", 15))

# list type
list1 = ['my', 'python', 'list']
print(list1, type(list1))
list1.append('added')
print(list1)

# tuple type (the tuples cannot be changed unlike lists)
tuple1 = ('my', 'python', 'tuple')
print(tuple1, type(tuple1))
tuple2 = 'this', 'is', 'tuple', 'type', 'too'
print(tuple2, type(tuple2))

# dictionary type (Each key is separated from its value by a colon (:))
dict1 = {'company': 'SKC&C', 'location': 'Seongnam'}
print(dict1, type(dict1))
dict1['company'] = 'SK'
dict1['slogan'] = 'Creative ICT Factory'
print(dict1, type(dict1))

```

```

# set type
set1 = set([1,1,3,3,5,5])
print(set1, type(set1))
set2 = set("Hello")
print(set2, type(set2))

```

- 실행 결과 확인

```

123
1.23
(1+2j)
28
255
Hello
What's your name?
"My name is Tiger" he said
Life is too short.
You need python.
I've been to 15 countries.
Swiss is the best country in 15
['my', 'python', 'list'] <class 'list'>
['my', 'python', 'list', 'added'] <class 'list'>
('my', 'python', 'tuple') <class 'tuple'>
('this', 'is', 'tuple', 'type', 'too') <class 'tuple'>
{'company': 'SKC&C', 'location': 'Seongnam'} <class 'dict'>
{'company': 'SK', 'location': 'Seongnam', 'slogan': 'Creative ICT Factory'} <class 'dict'>
{1, 3, 5} <class 'set'>
{'l', 'H', 'o', 'e'} <class 'set'>

```

#5 - Python 이해하기 - 제어문

▪ myflowcontrol.py

```

import time

localtime = time.localtime(time.time())
localhour = localtime.tm_hour
print(localtime, type(localtime))
print(localtime.tm_hour, type(localtime.tm_hour))

#if statement
print("\n-- if statement --")
greeting = ""
if(localhour >= 6 and localhour < 11):
    greeting = "Good Morning"
elif(localhour >= 11 and localhour < 17):
    greeting = "Good Afternoon"
elif(localhour >= 17 and localhour <22):
    greeting = "Good Evening"
else:
    greeting = "Hello"
print(greeting, type(greeting))

#while statement
print("\n-- while statement --")
count = 0
while(count < 9):
    print ('The count is: ', count, type(count))
    count = count + 1

#for statement
print("\n-- for statement --")
fruits = ['banana', 'apple', 'mango']
for item in fruits:
    print (item, type(item))
for idx in range(len(fruits)):
    print (idx, "-", fruits[idx])

```

• 실행 결과 확인

```

time.struct_time(tm_year=2017, tm_mon=4, tm_mday=7,
tm_hour=10, tm_min=48, tm_sec=33, tm_wday=4, tm_yday=97,
tm_isdst=0) <class 'time.struct_time'>
10 <class 'int'>

-- if statement --
Good Morning <class 'str'>

-- while statement --
The count is: 0 <class 'int'>
The count is: 1 <class 'int'>
The count is: 2 <class 'int'>
The count is: 3 <class 'int'>
The count is: 4 <class 'int'>
The count is: 5 <class 'int'>
The count is: 6 <class 'int'>
The count is: 7 <class 'int'>
The count is: 8 <class 'int'>

-- for statement --
banana <class 'str'>
apple <class 'str'>
mango <class 'str'>
0 - banana
1 - apple
2 - mango

```

7. Hands-on

I. Environment Setup

#6 - Python 이해하기 - 함수

▪ myfunction.py

```
def sayHello(name):
    greeting = "Hello " + name + "."
    return greeting

def noReturn():
    print("This function doesn't have return.")

def keywordArgs(name, age):
    print("Hello! I'm", name + ",", age, "years old.")

def defaultArgs(name, age, nationality="korean"):
    print("Hello! I'm", name + ",", age, "years old. I'm a", nationality)

def variableLengthArgs(sentence, *args):
    for idx in range(len(args)):
        if (idx == 0):
            sentence = sentence + " " + args[idx]
        elif (idx <= len(args)-2):
            sentence = sentence + ", " + args[idx]
        else:
            sentence = sentence + " and " + args[idx] + "."
    print(sentence)

def returnTuples():
    cust1 = "Cass"
    cust2 = "Singha"
    return cust1, cust2

if __name__ == "__main__":
    print(sayHello('Singha'))
    noReturn()
    keywordArgs(age="44", name="Singha")
    defaultArgs("Cass", "23")
    variableLengthArgs("My hobbies are", "soccer", "horse riding", "travelling")
    variableLengthArgs("I've been to", "China", "Vietnam", "U.S.A", "India")
```

```
customers = returnTuples()
(cust1, cust2) = returnTuples()
print(customers, type(customers))
print(cust1, type(cust1))
```

• 실행 결과 확인

```
Hello Singha.
This function doesn't have return.
Hello! I'm Singha, 44 years old.
Hello! I'm Cass, 23 years old. I'm a
korean
My hobbies are soccer, horse riding
and travelling.
I've been to China, Vietnam, U.S.A and
India.
('Cass', 'Singha') <class 'tuple'>
Cass <class 'str'>
```

#7 - Python 이해하기 – 내장함수

- mybuiltin.py

```
print(abs(-1.2))
print(bool(1))
print(chr(97))
print(dict(one=1, two=2, three=3))
print(float("3.14"))
print(int('100'))
print(len('python'))
print(list('python'))
print(max([1,2,3]))
print(min([1,2,3]))
print(open("mydatatype.py", "rb").name)
print(pow(2,3))
print(list(range(0,100,10)))
print(round(1.2345, 2))
print(sorted([3,1,2]))
print(str(3.14))
print(sum([1,2,3]))
print(tuple('python'))
print(type("python"))

func = lambda a, b: 3*a+2*b
print(func(1,2))
```

- 실행 결과 확인

```
1.2
True
a
{'one': 1, 'two': 2, 'three': 3}
3.14
100
6
['p', 'y', 't', 'h', 'o', 'n']
3
1
mydatatype.py
8
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
1.23
[1, 2, 3]
3.14
6
('p', 'y', 't', 'h', 'o', 'n')
<class 'str'>
7
```

#8 - Python 이해하기 – 표준라이브러리

▪ mystandardlib.py

```

import sys
import os
import pickle
import time
import calendar
import random

# sys - Path 경로
print(sys.path)

# os- Directory
print(os.pardir)
print(os.getcwd())

```

```

# pickle - 객체의 형태를 유지하면서 파일에 저장하고 불러올수 있게 하는 객체
f = open("../resources/pickletest.txt", "wb")
data = {'company': 'SK', 'location': 'Seongnam', 'slogan': 'Creative ICT Factory'}
pickle.dump(data,f)
f.close
f = open("../resources/pickletest.txt", "rb")
data = pickle.load(f)
print(data)

# time - Date and time
print(time.time())
print(time.localtime(time.time()))
print(time.strftime('%Y.%m.%d %X', time.localtime(time.time())))

# calendar
localtime = time.localtime(time.time())
print(calendar.pmonth(localtime.tm_year, localtime.tm_mon))

# random
print(random.random())
print(random.randint(1,100))
data = [1,2,3,4,5]
random.shuffle(data)
print(data)

```

- 실행 결과 확인

```

['C:\\workspace\\MyPythonDeepLearning\\EnvironmentSetup', ... 결과 생략 ...]
..
C:\\workspace\\MyPythonDeepLearning\\EnvironmentSetup
{'company': 'SK', 'location': 'Seongnam', 'slogan': 'Creative ICT Factory'}
1491537539.9740002
time.struct_time(tm_year=2017, tm_mon=4, tm_mday=7, tm_hour=12, tm_min=58, tm_sec=59, ... 결과 생략 ...
2017.04.07 12:58:59
    April 2017
Mo Tu We Th Fr Sa Su
... 결과 생략 ...

```

#9 - Python 이해하기 – 파일처리

- myfile.py

```
...
파일열기
open(filename, mode)
r-read
w-write
a-append
b-binary
...
f = open("../resources/newfile.txt", "w")
for idx in range(1, 6):
    data = "%d번째 줄입니다.\n" % idx
    f.write(data)
f.close()

f = open("../resources/newfile.txt", "r")
data = f.read()
print(data)
f.close()

...
with문을 이용해서 파일열기
with block을 벗어나면 파일객체는 자동 Close
"""

with open("../resources/newfile.txt", "r") as f:
    f.read()
    print(data)
```

- 실행 결과 확인

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
```

```
1번째 줄입니다.
2번째 줄입니다.
3번째 줄입니다.
4번째 줄입니다.
5번째 줄입니다.
```

#10 - Python 이해하기 – import modules

- mymodule.py

```
'''  
import (패키지/이름).모듈이름  
from (패키지/이름).모듈이름 import 모듈함수  
'''  
  
import EnvironmentSetup.myfunction as mf  
from EnvironmentSetup.myfunction import noReturn  
  
print("-- mymodule.py --")  
print(mf.sayHello("Tiger"))  
noReturn()
```

- 실행 결과 확인

```
-- mymodule.py --  
Hello Tiger.  
This function doesn't have return.
```

#11 - Python 이해하기 – 클래스

- mymodule.py

```
class Calculator:  
  
    def __init__(self, first, second):  
        self.first = first  
        self.second = second  
  
    def plus(self):  
        return self.first + self.second  
  
    def minus(self):  
        return self.first - self.second  
  
    def multiply(self):  
        return self.first * self.second  
  
    def divide(self):  
        return self.first / self.second  
  
c = Calculator(3, 4)  
print(c.plus(), c.minus(), c.multiply(), c.divide())
```

- 실행 결과 확인

```
7 -1 12 0.75
```

#11 - NumPy 이해하기 – create array

▪ mynumpyarray.py

```
import numpy as np

...
ndarray creation
numpy.arange([start,]stop,[step,]dtype=None)
ndarray.ndim : number of dimensions
ndarray.shape : the size of the array in each dimension
ndarray.size : total number of elements
ndarray.dtype : the type of the elements
...

a = np.arange(15).reshape(3,5)
print(a, type(a))
print(a.ndim)
print(a.shape)
print(a.size)
print(a.dtype.name, a.dtype.itemsize)

a = np.arange(-5, 5, 0.5)
print(a)

...
ndarray creation
create an array from a regular python list
...
a1 = np.array([1.0, 2.0, 3.0])
a2 = np.array([[1.0, 2.0],[ 3.0, 4.0]])

print(a1, type(a1))
print(a2, type(a2))
print(a1.ndim, a1.shape, a1.size, a1.dtype)
print(a2.ndim, a2.shape, a2.size, a2.dtype)
```

• 실행 결과 확인

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]] <class 'numpy.ndarray'>
2
(3, 5)
15
int32 4
[-5. -4.5 -4. -3.5 -3. -2.5 -2. -1.5 -1. -0.5 0. 0.5 1. 1.5 2.
 2.5 3. 3.5 4. 4.5]
[ 1.  2.  3.] <class 'numpy.ndarray'>
[[ 1.  2.]
 [ 3.  4.]] <class 'numpy.ndarray'>
1 (3,) 3 float64
2 (2, 2) 4 float64
```

#12 - NumPy 이해하기 – operate array

▪ mynumpyoperation.py

```
import numpy as np

x1 = np.array([1.0, 2.0, 3.0])
y1 = np.array([5.0, 10.0, 15.0])
x2 = np.array([[1.0, 2.0], [3.0, 4.0]])
y2 = np.array([[5.0, 10.0], [15.0, 20.0]])
z1 = np.array([-1.0, -2.0])
z2 = np.array([[5.0], [10.0], [15.0]])

...
ndarray basic operation
...

print(x1 + y1)
print(x1 - y1)
print(x1 * y1)
print(x1 / y1)
print(x2 + y2)
print(x2 * y2)

...
ndarray broadcast
...

print(x2 + z1)
print(x2 * z1)
print(x1 + z2)
print(x1**2)
print(x1>=2)

...
shape manipulation
...

print(x2.flatten())
print(x2.reshape(2,2))
```

• 실행 결과 확인

```
[ 6. 12. 18.]
[ -4. -8. -12.]
[ 5. 20. 45.]
[ 0.2 0.2 0.2]
[[ 6. 12.]
 [ 18. 24.]]
[[ 5. 20.]
 [ 45. 80.]]
[[ 0. 0.]
 [ 2. 2.]]
[[-1. -4.]
 [-3. -8.]]
[[ 6. 7. 8.]
 [ 11. 12. 13.]
 [ 16. 17. 18.]]
[ 1. 4. 9.]
[False True True]
[ 1. 2. 3. 4.]
[[ 1. 2.]
 [ 3. 4.]]
```

#13 - NumPy 이해하기 – indexing, slicing and iterating

▪ mynumpyoperation.py

```
import numpy as np

a = np.arange(10)**2
print(a)
print(a[2])
print(a[3:5])
a[0:6:2] = -a[0:6:2] #from start to position 6 every element
print(a)
print(a[::-1]) #reversed a
for i in a:
    print(i+1)

a = np.arange(1,21,1).reshape(4,5)
print(a)
print(a[2,3])
print(a[0:4,1])
print(a[1,:])
print(a[-1,:])
```

• 실행 결과 확인

```
[ 0  1  4  9 16 25 36 49 64 81]
4
[ 9 16]
[ 0   1  -4   9 -16  25  36  49  64  81]
[ 81  64  49  36  25 -16   9  -4   1   0]
1
2
-3
10
-15
26
37
50
65
82
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
14
[ 2  7 12 17]
[ 6  7  8  9 10]
[16 17 18 19 20]
```

#14 - NumPy 이해하기 – universal functions

▪ mynumpyuniversal.py

```

import numpy as np

a = np.arange(0, 6, 1)
print(a)
print(np.sin(a))
print(np.cos(a))
print(np.tanh(a))
print(np.exp(a))

a = np.arange(0.01, 1, 0.05)
print(np.log(a))

a1 = np.array([2,3,4])
a2 = np.array([1,5,2])
print(np.maximum(a1,a2))
print(np.minimum(a1,a2))
print(np.sum(a1))

print(np.random.choice(100,5)) #0~100사이 중 5개 선택
print(np.random.rand(2,3)) # (2,3) Shape의 Random 값 리턴
print(np.random.rand(2,3)) # (2,3) Shape 표준정규분포로 초기화

...
numpy.argmax() : Returns the indices of the maximum values
...
a = np.array([6,2,3,1,4,5])
print(a)
print(np.argmax(a), a[np.argmax(a)])
a = np.array([[0.1,0.8,0.2],[0.3,0.2,0.5],[0.9,0.5,0.3]])
print(np.argmax(a, axis=1))

```

• 실행 결과 확인

```

[0 1 2 3 4 5]
[ 0.          0.84147098  0.90929743  0.14112001 -
 0.7568025   -0.95892427]
[ 1.          0.54030231  -0.41614684  -0.9899925  -
 0.65364362   0.28366219]
[ 0.          0.76159416  0.96402758  0.99505475
 0.9993293   0.9999092 ]
[ 1.          2.71828183    7.3890561   20.08553692
54.59815003
 148.4131591 ]
[-4.60517019 -2.81341072 -2.20727491 -1.83258146 -
1.56064775 -1.34707365
 -1.17118298 -1.02165125 -0.89159812 -0.77652879 -
0.67334455 -0.5798185
 -0.49429632 -0.41551544 -0.34249031 -0.27443685 -
0.21072103 -0.15082289
 -0.09431068 -0.04082199]
[2 5 4]
[1 3 2]
9
[69 37 83 51 61]
[[ 0.94980072  0.36666774  0.02717995]
 [ 0.05536549  0.6011632   0.78340661]]
[[ 0.62427247  0.09579107  0.16240299]
 [ 0.69994835  0.14973522  0.53502239]]
[6 2 3 1 4 5]
0 6
[1 2 0]

```

#15 - matplotlib 이해하기

▪ mymatplotlib.py

```

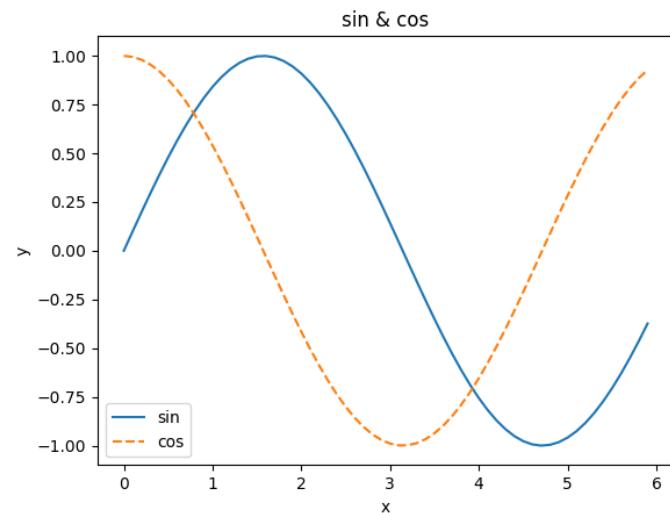
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as image

# 데이터준비
x = np.arange(0, 6, 0.1)
y1 = np.sin(x)
y2 = np.cos(x)
print(x)
print(y1)
print(y2)

# 그래프 그리기
plt.plot(x, y1, label="sin")
plt.plot(x, y2, linestyle="--", label="cos")
plt.xlabel("x")
plt.ylabel("y")
plt.title("sin & cos")
plt.legend()
plt.show()

# 이미지 표시하기
img = image.imread("../resources/Scream.jpg")
plt.imshow(img)
plt.show()

```



II. Perceptron



1. 최초의 인공신경망
2. 단층 퍼셉트론
3. 논리 회로
4. Bias
5. 단층 퍼셉트론의 한계
- b. 다층 퍼셉트론
7. Hands-on

1. 최초의 인공신경망

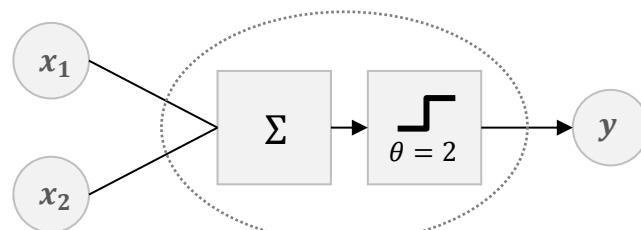
II. Perceptron

1943년 워렌 맥컬록(McCulloch)과 윌터 피츠(Pitts)는 생물학적 신경망 이론을 단순화 해서 최초로 인공신경망 이론을 제시.

▪ 인공신경망(ANN, Artificial Neural Network)

최초의 인공신경망은 생물학적 신경망을 단순화 해서 논리, 산술, 기호 연산을 구현할 수 있게 하였는데 이것을 TLU (Threshold Logic Unit, 한계 논리 단위)라고 정의했다.

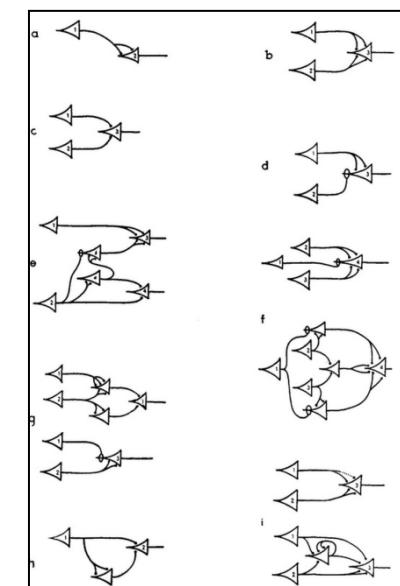
▪ 인공신경망 TLU를 AND 논리 연산에 적용한 예



입력 x1	입력 x2	출력 y
0	0	0
0	1	0
1	0	0
1	1	1

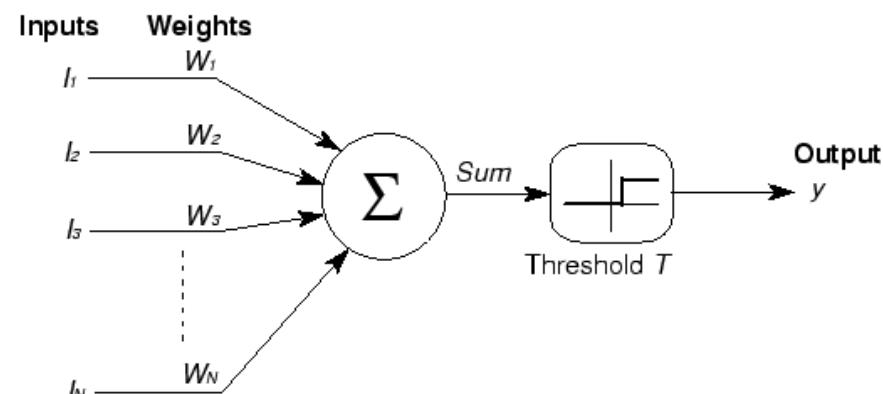
만약 $\theta=1$ 이면 OR 연산이 적용된 TLU가 된다.

McCulloch-Pitts 논문에 표현된 TLU



헵의 학습 (Hebb's Learning)

- 생물학적 신경망에서 뉴런 사이에서 신호전달 시 반복적 또는 지속적으로 신호가 자극됨. 이 결과로 뉴런 A에서 뉴런 B로 가는 경로인 시냅스 연결이 강화된다.
- Hebb은 이것이 신경 레벨의 일종의 학습/기억의 과정이라고 보고 인공신경망에 가중치 개념으로 도입했다.
- 이것이 헵의 규칙 또는 헵의 학습이라 한다.



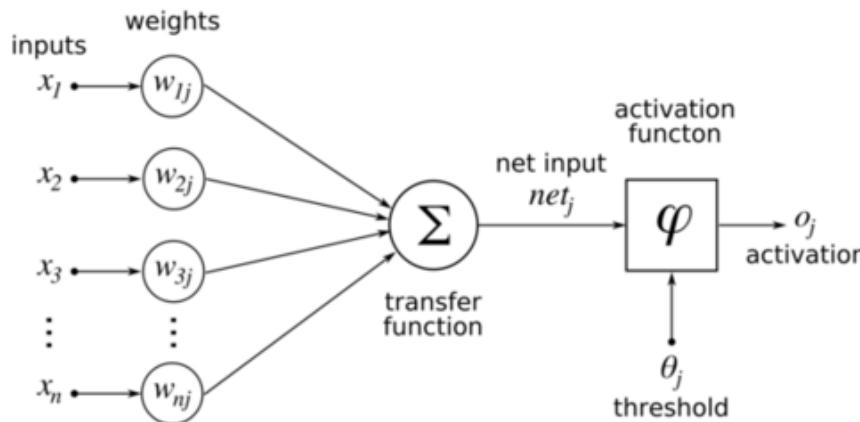
2. 단층 퍼셉트론 (SLP, Single-Layer Perceptron)

II. Perceptron

로센블래트는 사람의 시각 인지 과정에 영감을 얻어 퍼셉트론 이론으로 인공적인 시각 인지 과정을 구현하는 데 적용했다.

▪ 단층 퍼셉트론 (Single-Layer Perceptron)

McCulloch-Pitts TLU 이론 + Hebb의 가중치 이론으로 입력총과 출력총만으로 구성

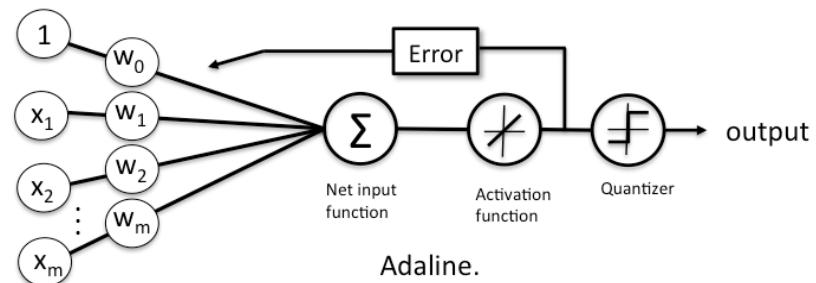


- 활성화 함수 (Activation Function)
 - ✓ 일정크기(Threshold θ)이상의 신호가 올 경우 값을 전달
 - ✓ 계단함수 대신 Sigmoid 함수나 tanh함수 또는 ReLU함수를 사용할 수도 있다.
- AND / OR Perception 예
 - ✓ AND : $y = 0.6x_1 + 0.6x_2$ 이고 $\theta = 1$
 - ✓ OR : $y = 1.2x_1 + 1.2x_2$ 이고 $\theta = 1$
- 적당한 가중치를 알아내는 학습이 필요함.



Widrow-Hoff Learning Rule

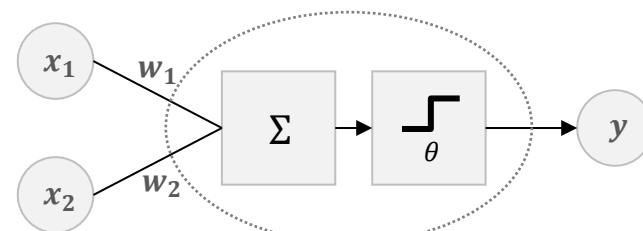
- 1960년 스탠포드대 Bernard Widrow교수는 Perceptron과 유사한 ADALINE (Adaptive Linear Neuron) 단층 신경망 모델과 위드로-호프 학습규칙(Widrow-Hoff Learning Rule)을 공개
- Widrow-Hoff Learning Rule (**Delta Rule**)
단층 신경망에서 적당한 가중치를 알아내기 위해 출력 층의 출력 값의 오차에 비례해 가중치를 조절하는 것.
- 델타 규칙은 신경망의 오차가 충분히 들어들 때 까지 학습 데이터를 반복해서 재 학습해야 한다.
- 델타 규칙은 후에 역전파의 기본이론이 된다.



Perceptron으로 AND, NAND, OR 논리 회로를 구조의 변경 없이 표현할 수 있다. 다른 것은 가중치와 임계 값 뿐이다.

- 입력이 2개인 Perceptron

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$



- 논리회로 진리표 (Truth-table)

입력		출력 (y)		
x_1	x_2	AND	NAND	OR
0	0	0	1	0
1	0	0	1	1
0	1	0	1	1
1	1	1	0	1

- 가중치와 임계 값을 조정하면 논리회로를 표현할 수 있음.

	w_1	w_2	θ
AND	0.5	0.5	0.7
NAND	-0.5	-0.5	-0.7
OR	0.5	0.5	0.2



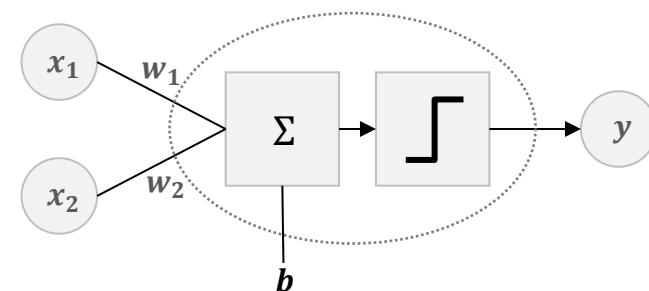
기계 학습

- 위 예제에서 가중치와 임계 값은 사람이 임의로 지정하였음.
- 기계학습은 이 값을 정하는 작업을 컴퓨터가 자동으로 하도록 하는 것.
- 즉, 신경망에서 학습이란 컴퓨터가 가중치를 정하는 것이며, 사람은 망의 구조(모델)을 고민하고 학습 데이터를 주는 일을 하는 것이다.

Bias (편향)을 도입하여 임계 값을 치환하면 같은 결과를 낼 수 있다. (θ 를 $-b$ 로 대체)

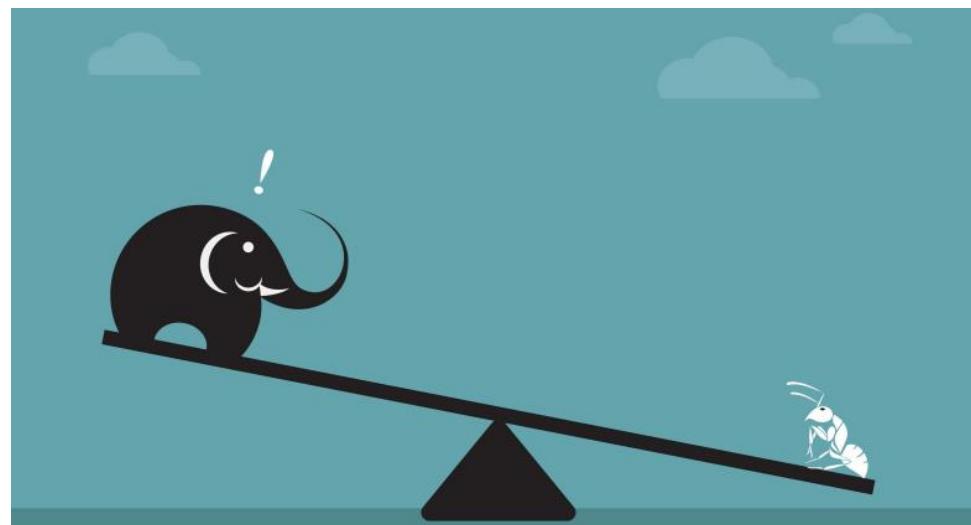
- 입력이 2개인 Perceptron (Bias의 도입)

$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$



- Bias (편향)

- 가중치 (w_1, w_2)는 각 입력 신호가 결과에 주는 영향력을 조절하는 매개변수이고, Bias는 뉴런(Node)이 얼마나 쉽게 활성화(여기에서는 결과를 1로 출력)하느냐를 조정하는 매개변수 임.
- Bias의 사전적 의미로는 '한쪽으로 치우쳐 균형을 깬다'라는 의미를 담고 있다.



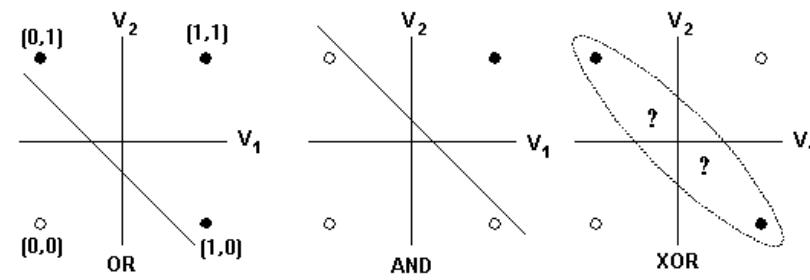
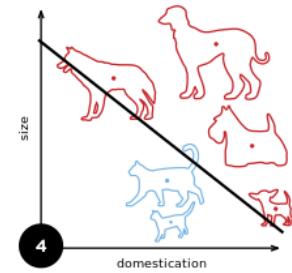
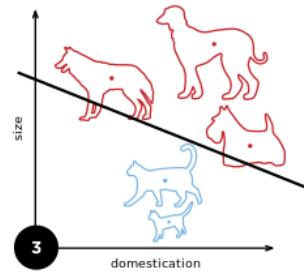
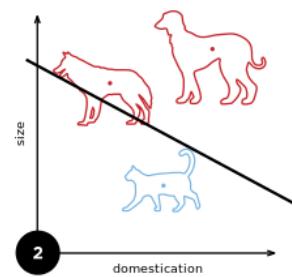
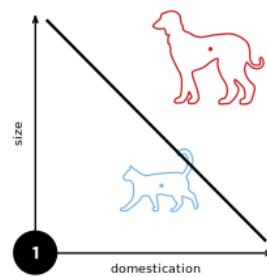
5. 단층 퍼셉트론의 한계

II. Perceptron

단층 퍼셉트론은 비선형 영역 문제를 분리할 수가 없다.

▪ Perceptron 이론의 한계 발견

- Perceptron은 학습이 진행될수록 선형 분리(linear boundary)를 업데이트하면서 학습
- 하지만, 간단한 XOR문제에는 적용할 수 없는 한계가 발견 → 인공지능의 1차 겨울을 초래



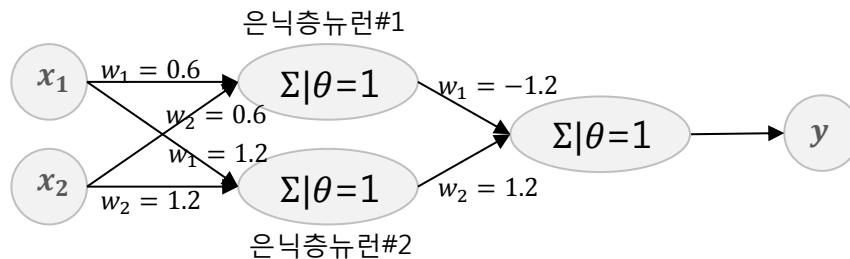
입력		출력 (y)
x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0

6. 다층 퍼셉트론 (MLP, Multi-Layer Perceptron)

II. Perceptron

XOR 문제는 단층 퍼셉트론에서는 해결할 수 없는데, 입력층과 출력층 사이에 은닉층 수를 하나만 더하면 쉽게 해결할 수 있다.

▪ 2층 Perceptron을 이용한 XOR 판별식

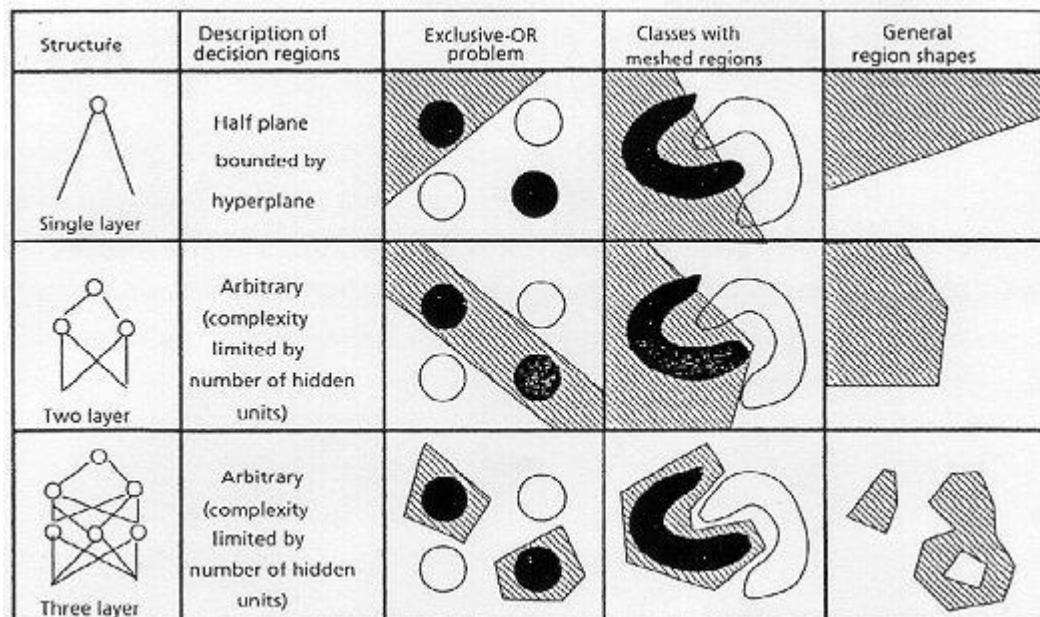


입력 x1	입력 x2	은닉층 뉴런#1	은닉층 뉴런#2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

[은닉층 뉴런#1은 AND연산, 은닉층 뉴런#2는 OR연산,
출력층은 XOR연산이 된다.]

▪ 은닉층 (Hidden Layer)

- Hidden Layer를 추가하면 선형분리 불가능 (linearly inseparable) 문제를 풀 수 있다.
 - ✓ 하지만, 다층 퍼셉트론 발견 당시에는 다층 퍼셉트론을 학습시킬 수학적 모델이 없었다.
 - ✓ 은닉층의 개수가 늘어나면 더욱 복잡한 문제를 풀 수 있으나 컴퓨터 계산량이 늘어난다.
- Hidden Layer의 역할
 - ✓ 앞 단계에서 받은 데이터(신호)를 필터링해서 좀 더 구체화 한 후 다음 단계 층으로 전달
 - ✓ 각 은닉층마다 몇 개의 뉴런(노드)이 최적인지는 문제에 따라 다르다.
- 신경망 층에 따라 Decision Boundary는 더 정확해 진다.



[2차원 평면에 분포된 데이터 세트를 은닉층 개수에 따라 분리 비교]

#1 – 단층 퍼셉트론

▪ myperceptron.py

- AND, NAND, OR Perceptron 구현하고, 특히 OR Perceptron 구현 시는 Bias를 활용해 본다.
- 구현된 함수는 가중치와 임계 값을 제외한 다른 코드는 똑같음을 확인한다.

```
import numpy as np

def andPerceptron(x1, x2):
    w1, w2, theta = 0.5, 0.5, 0.7
    netInput = x1*w1 + x2*w2
    if netInput <= theta:
        return 0
    elif netInput > theta:
        return 1

def nandPerceptron(x1, x2):
    w1, w2, theta = -0.5, -0.5, -0.7
    netInput = x1*w1 + x2*w2
    if netInput <= theta:
        return 0
    elif netInput > theta:
        return 1

def orPerceptron(x1, x2):
    w1, w2, bias = 0.5, 0.5, -0.2
    netInput = x1*w1 + x2*w2 + bias
    if netInput <= 0:
        return 0
    else:
        return 1
```

```
inputData = np.array([[0,0],[0,1],[1,0],[1,1]])
print("---And Perceptron---")
for xs1 in inputData:
    print(str(xs1) + " ==> " + str(andPerceptron(xs1[0], xs1[1])))
print("---Nand Perceptron---")
for xs2 in inputData:
    print(str(xs2) + " ==> " + str(nandPerceptron(xs2[0], xs2[1])))
print("---Or Perceptron---")
for xs3 in inputData:
    print(str(xs3) + " ==> " + str(orPerceptron(xs3[0], xs3[1])))
```

▪ 실행 결과 확인

```
--And Perceptron--
[0 0] ==> 0
[0 1] ==> 0
[1 0] ==> 0
[1 1] ==> 1
--Nand Perceptron--
[0 0] ==> 1
[0 1] ==> 1
[1 0] ==> 1
[1 1] ==> 0
--Or Perceptron--
[0 0] ==> 0
[0 1] ==> 1
[1 0] ==> 1
[1 1] ==> 1
```

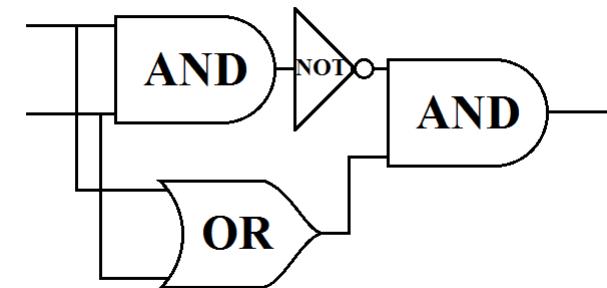
#2 – 다층 퍼셉트론

▪ myperceptron.py

- 단층 Perceptron을 쌓아서 다층 Perceptron을 생성해 본다.

```
... 코드 생략 ...
def xorPerceptron(x1, x2):
    s1 = nandPerceptron(x1, x2)
    s2 = orPerceptron(x1, x2)
    y = andPerceptron(s1, s2)
    return y

... 코드 생략 ...
print("---Xor Perceptron---")
for xs4 in inputData:
    print(str(xs4) + " ==> " + str(xorPerceptron(xs4[0], xs4[1])))
```



▪ 실행 결과 확인

```
---Xor Perceptron---
[0 0] ==> 0
[0 1] ==> 1
[1 0] ==> 1
[1 1] ==> 0
```

III. Neural Network

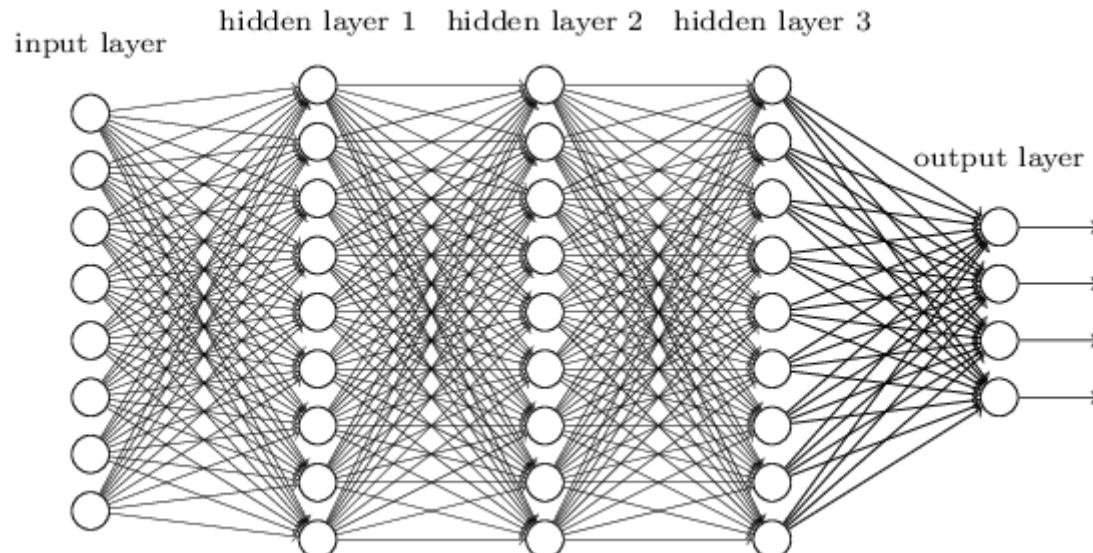


1. **Neural Network**
2. **Activation Function**
3. **Matrix Inner Product**
4. **Feed Forward**
5. **Output Layer**
- b. **MNIST Example**
7. **Hands-on**

신경망은 입력층, 은닉층, 출력층으로 구성되어 있다.

- **딥러닝이란?**

인간의 신경망(Neural Network) 이론을 이용한 인공 신경망 (ANN, Artificial Neural Network)의 일종으로, 계층 구조 (Layer Structure)로 구성하면서 입력층(Input Layer)과 출력층(Output Layer) 사이에 하나 이상의 은닉층(Hidden Layer)을 가지고 있는 심층 신경망(DNN, Deep Neural Network)이다.



Hidden Layer 수

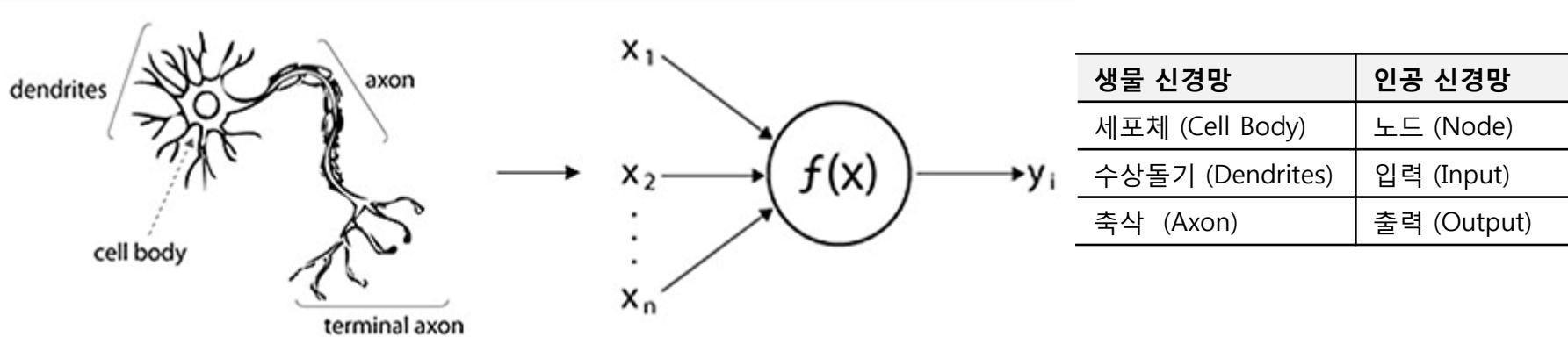
- Deep Learning의 깊이는 Hidden Layer 개수에 하나를 더하면 된다.
- 굳이 구별하면 2~3개 층의 신경망을 Shallow (Non-deep) Learning, 10개 이상이면 Very Deep Learning이라 한다.
- Deep Learning의 신경망 계층 수를 늘리면 연산에 필요한 복잡도가 제곱 크기로 늘어난다.

인공신경망(ANN)은 인간의 뇌 구조를 모방하여 모델링 한 수학적 모델이다.

- 신경세포 (Neuron, 뉴런)

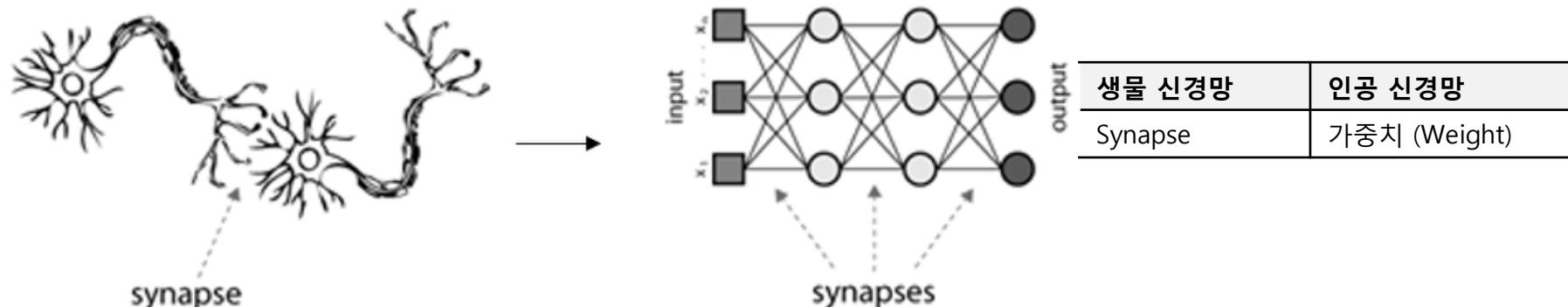
Neuron의 입력은 다수이고 출력은 하나이며, 여러 신경세포로부터 전달되어 온 신호들은 합산되어 출력된다.

합산된 값이 설정 값(역치, Threshold) 이상이면 출력 신호가 생기고 이하이면 출력 신호가 없다.



- 연결 : Synapse vs. weight

다수의 Neuron이 연결되어 의미 있는 작업을 하듯, 인공신경망의 경우도 노드들을 연결시켜 Layer를 만들고 연결 강도는 가중치로 처리된다.

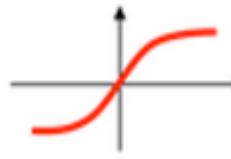
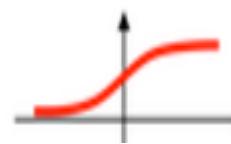
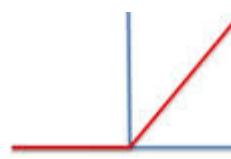


뇌 신경망에는 시냅스가 있는데 인공 신경망에서는 이런 방식을 모방한 활성화 함수를 이용한다.

- 활성화 함수 (Activation Function)

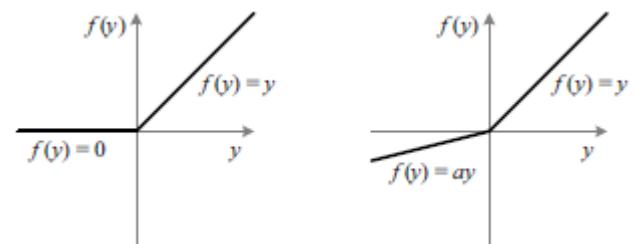
- Synapse는 전달된 전기 신호가 최소한의 자극 값을 초과하면 활성화되어 다음 뉴런으로 전기신호를 전달한다.
- 활성화 함수는 이것을 모방하여 값이 작을 때는 출력 값을 작은 값으로 막고, 일정한 값을 초과하면 출력 값이 급격히 커지는 함수를 이용한다.
- 신경망에서 Node는 전달받은 데이터를 가중치를 고려해 합산하고, 그 값을 활성화 함수를 적용해 다음 층에 전달한다.

- 은닉층에서 자주 이용되는 활성화 함수

tanh 함수 (Hyperbolic Tangent Function, 쌍곡 탄젠트 함수)	 $y = \tanh(x)$
Sigmoid 함수	 $y = \frac{1}{1 + e^{-x}}$
ReLU 함수 (Rectified Linear Unit Function)	 $y = 0 \text{ (if, } x < 0\text{)}y = x \text{ (if, } x \geq 0\text{)}$

- Parametric ReLU 함수

- ReLU함수는 0보다 작은 값에서 경사가 0이다.
- 그래서 이 부분에 작은 경사 값을 준 **Parametric ReLU**함수도 자주 이용된다.



- 예전에는 활성화 함수로 tanh함수나 Sigmoid함수를 자주 사용했으나 최근에는 ReLU 함수를 사용
→ ReLU 함수는 상대적으로 학습 속도가 빠르고 학습이 잘 됨.

2. Activation Function

III. Neural Network

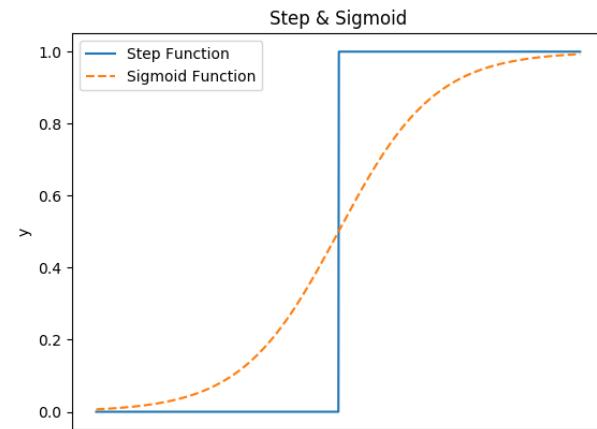
Python으로 활성화 함수를 구현해보자

▪ 계단 함수 (Step Function)

$$y = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

```
def stepFunction(x):
    y = [] # Empty List
    for value in x:
        if value > 0:
            y.append(1)
        else:
            y.append(0)

    return np.array(y)
```

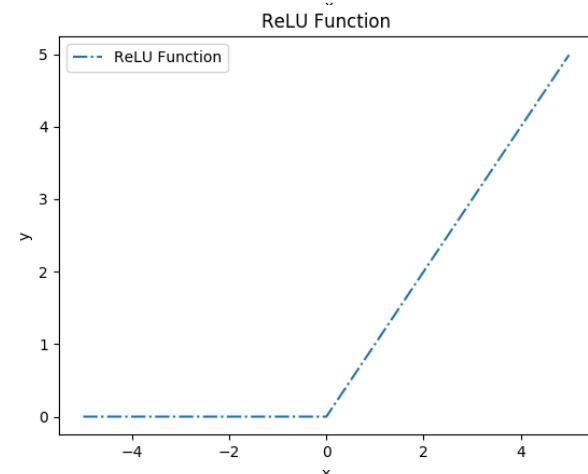


▪ 시그모이드 함수 (Sigmoid Function)

- 계산 결과는 항상 0과 1 사이에 있게 된다
- E는 자연상수로 2.7182...의 값을 갖는 실수

$$y = \frac{1}{1 + e^{-x}}$$

```
def sigmoidFunction(x):
    return 1/(1+np.exp(-x))
```



▪ ReLU 함수 (Rectified Linear Unit Function)

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

```
def reluFunction(x):
    return np.maximum(0, x)
```



비선형 함수

- 계단 함수나 시그모이드 함수처럼 값이 하나의 직선으로 나오지 않는 함수를 비선형함수라 함.
- 선형 함수의 예 : $f(x) = ax + b$
- 신경망에서 활성화 함수로 비선형 함수를 사용해야 함.
- 선형 함수를 사용하면 신경망을 깊게 하는 의미가 없어짐.

벡터와 행렬은 인경신경망을 이해하기 위한 기초 수학이다.

【 벡터 】

- 벡터는 방향과 크기의 의미를 모두 포함하는 수학으로 스칼라 값들을 순서대로 가진다.
- 선형대수학에서 n-차원 벡터
 - n개의 원소를 갖는 벡터로 열 벡터(Column Vector)로 표현
 - 열 벡터는 행 벡터(Row Vector)로 전치해서 표현하기도 함

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n]^T$$

【 행렬 】

- 행렬은 원소들이 놓여지는 행과 열을 가진다.
- 2×3 행렬의 예

$$\begin{bmatrix} 1 & 9 & -13 \\ 20 & 5 & -16 \end{bmatrix}$$

- 행 또는 열의 크기가 1이고, 다른 열 또는 행의 크기가 1이 아닐 때, 그 행렬은 벡터로 볼 수 있다.

【 행렬과 벡터의 연산 】

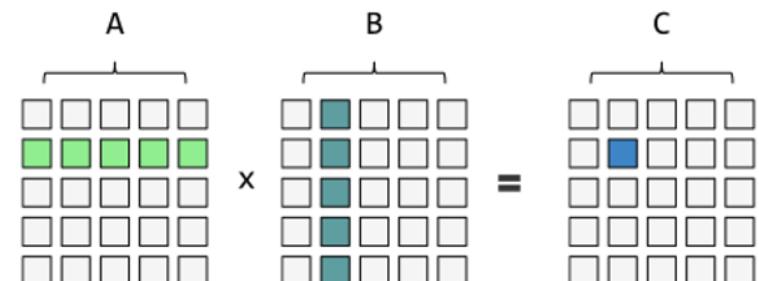
- 행렬과 벡터의 곱셈(내적)은 일정한 규칙에 따라 계산하며 행렬과 벡터의 크기에 따라 그 결과가 달라진다.

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = [1a + 2b]$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} a & c \\ b & d \end{bmatrix} = [1a + 2b \quad 1c + 2d]$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \cdot \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

$$= \begin{bmatrix} 1a + 2b + 3c & 1d + 2e + 3f \\ 4a + 5b + 6c & 4d + 5e + 6f \end{bmatrix}$$



$$C[i][j] = \text{sum}(A[i][k] * B[k][j]) \text{ for } k = 0 \dots n$$

Important

3. Matrix Inner Product

III. Neural Network

Python으로 행렬을 생성하고 내적을 계산해 보자

- numpy.ndarray를 이용한 다차원 배열 생성

```
npArr1 = np.array([[1,2],[3,4]]) # 2x2 행렬 (2차원배열)
npArr2 = np.array([[5,6],[7,8]]) # 2x2 행렬 (2차원배열)
npArr3 = np.array([[1,2,3],[4,5,6]]) # 2x3 행렬 (2차원배열)
npArr4 = np.array([[5,6],[7,8],[9,10]]) # 3x2 행렬 (2차원배열)
npArr5 = np.array([1,2]) # 2x1 벡터 (1차원 배열)
```

- 행렬의 내적

- NumPy의 dot()함수를 이용하여 내적을 쉽게 계산한다

```
npResult1 = np.dot(npArr1, npArr2) # matrix inner product (2x2 행렬)
npResult2 = np.dot(npArr3, npArr4) # matrix inner product (2x2 행렬)
npResult3 = np.dot(npArr4, npArr3) # matrix inner product (3x3 행렬)
npResult4 = np.dot(npArr4, npArr5) # matrix inner product (3x1 벡터)
```

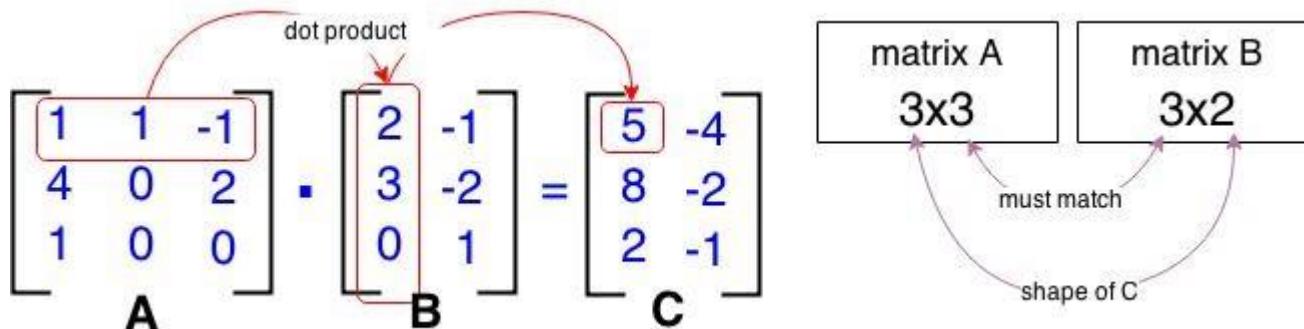
$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{pmatrix} = \begin{pmatrix} 46 & 52 \\ 109 & 124 \end{pmatrix}$$

$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 29 & 40 & 51 \\ 39 & 54 & 69 \\ 49 & 68 & 87 \end{pmatrix}$$

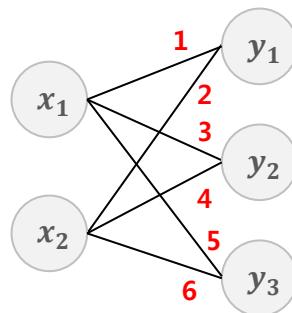
$$\begin{pmatrix} 5 & 6 \\ 7 & 8 \\ 9 & 10 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 17 \\ 23 \\ 29 \end{pmatrix}$$

- 행렬의 곱에서는 대응하는 차원의 원소 소의 일치 시켜야 한다.



NumPy 행렬을 이용해서 신경망의 계산을 수행한다.

- 신경망 계산을 행렬 내적 식으로 표현



$$(x_1 \quad x_2) \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = (y_1 \quad y_2 \quad y_3)$$

- 코드 구현 예

```
preNodes = np.array([1,2]) # 신경망의 이전계층의 노드 값
weight = np.array([[1,3,5],[2,4,6]]) # 가중치
netInput = np.dot(preNodes, weight) # 노드값과 가중치의 연산
print(netInput)
```

- 실행 결과 확인

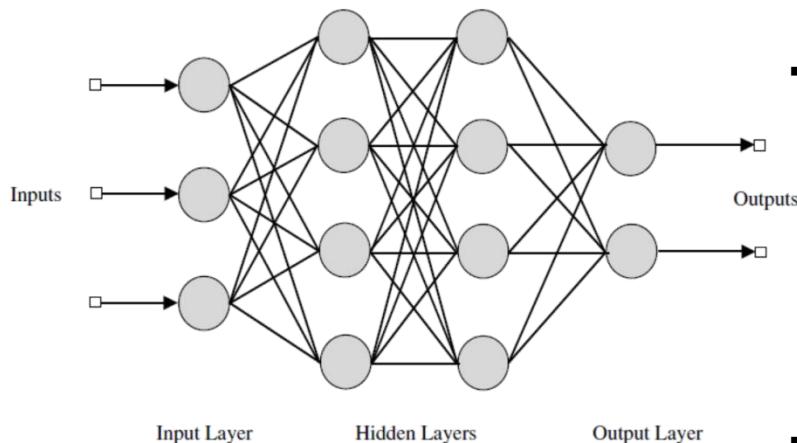
```
[ 5 11 17]
```



Excellent

신경망에서 정보의 흐름은 입력층 → 은닉층 → 출력층으로 진행되는데 이렇게 정보가 전방으로 전달되는 인공신경망을 피드포워드 신경망 (FNN, Feedforward Neural Network)라고 한다.

▪ Feedforward Neural Network



- Input Layer (입력층)
 - ✓ 입력 데이터를 받아들인다.
 - ✓ 입력층의 노드(뉴런) 수는 입력 데이터의 특성 개수와 일치 한다.
- Hidden Layer (은닉층)
 - ✓ 은닉층의 뉴런 수와 은닉층의 개수는 신경망 설계자의 직관과 경험에 의존
 - ✓ 은닉층의 뉴런 수가 너무 많으면 Overfitting이 발생, 너무 적으면 충분히 표현하지 못함.
 - ✓ 은닉층의 개수가 너무 많으면 비효율적 (예, 은닉층 수를 2배로 늘리면 컴퓨팅 시간은 400% 증가하지만 정확성은 10%로 증가함)
- Output Layer (출력층)
 - ✓ 해결하고자 하는 문제의 성격 (예, 필기체 숫자를 인식한다면 0에서 9까지 10개 노드로 설정)



학습이란?

- Feedforward 신경망에서 원하는 결과를 얻기 위해 뉴런 사이의 적당한 가중치를 알아내는 것



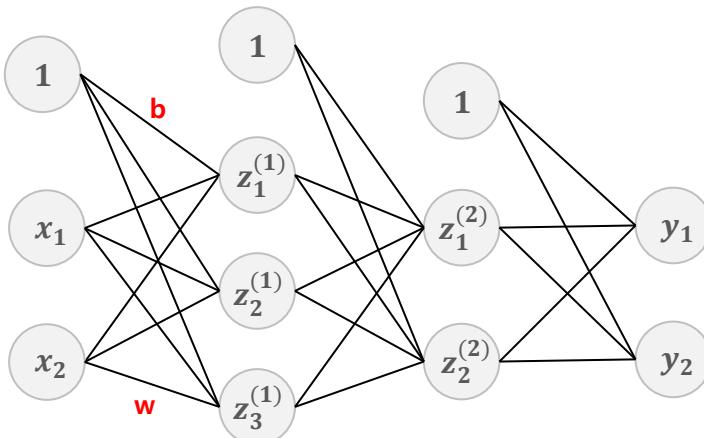
4. Feed Forward

III. Neural Network

Python으로 3층 신경망을 구현해 보자

- 3층 신경망의 구성 예

Layer	Node 수	Node Shape	Weight Shape	Bias shape	계산 식
입력층	2	2차원 Vector	2x3 Matrix	3차원 Vector	은닉층 (1 st) = 활성화함수((입력층x가중치1) + 편향1)
은닉층 (1 st)	3	3차원 Vector	3x2 Matrix	2차원 Vector	은닉층 (2 nd) = 활성화함수((은닉층 (1 st) x가중치2) + 편향2)
은닉층 (2 nd)	2	2차원 Vector	2x2 Matrix	2차원 Vector	출력층 = 활성화함수((은닉층 (2 nd) x가중치3) + 편향3)
출력층	2	2차원 Vector			



```
W1 = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]]) # 가중치1 : 2x3 shape
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])# 가중치2 : 3x2 shape
W3 = np.array([[0.1, 0.3], [0.2, 0.4]]) # 가중치3 : 2x2 shape
b1 = np.array([0.7, 0.8, 0.9]) # Bias1 : 3차원 Vector
b2 = np.array([0.7, 0.8]) # Bias2 : 2차원 Vector
b3 = np.array([0.7, 0.8]) # Bias3 : 2차원 Vector

ia = np.array([4.5, 6.2])
# 첫번째 은닉층의 계산
z1 = sigmoidFunction(np.dot(ia, W1) + b1)
print("1st 은닉층 값 : " + str(z1))
# 두번째 은닉층의 계산
z2 = sigmoidFunction(np.dot(z1, W2) + b2)
print("2nd 은닉층 값 : " + str(z2))
# 출력층의 계산
y = identityFunction(np.dot(z2, W3) + b3)
```

신경망은 분류(Classification)와 회귀(Regression)에 모두 이용할 수 있으며, 목적에 맞게 출력층을 설계 해야 한다.

- **출력층의 활성화 함수**

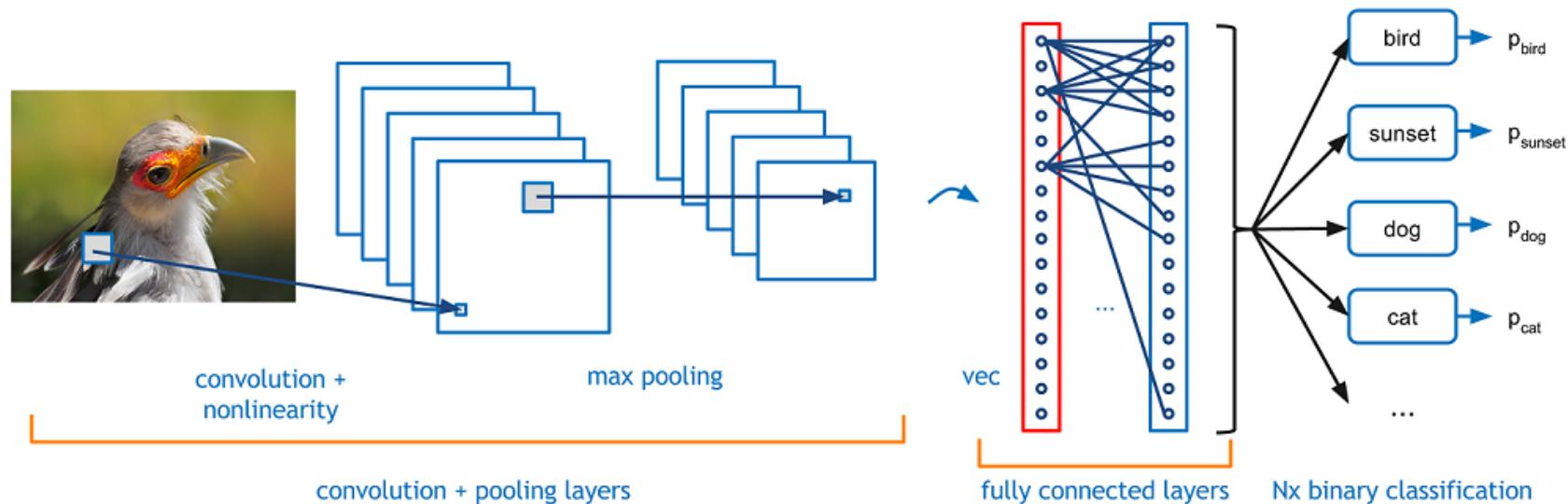
- 회귀 문제를 해결할 때 → 항등 함수 (identity function) 이용
- 분류 문제를 해결할 때 → 소프트맥스 함수 (softmax function) 이용

- **항등 함수 (identity function)**

- 입력을 그대로 출력, 즉 입력과 출력이 같다.

- **분류 문제에서 출력층의 Node 수 정하기**

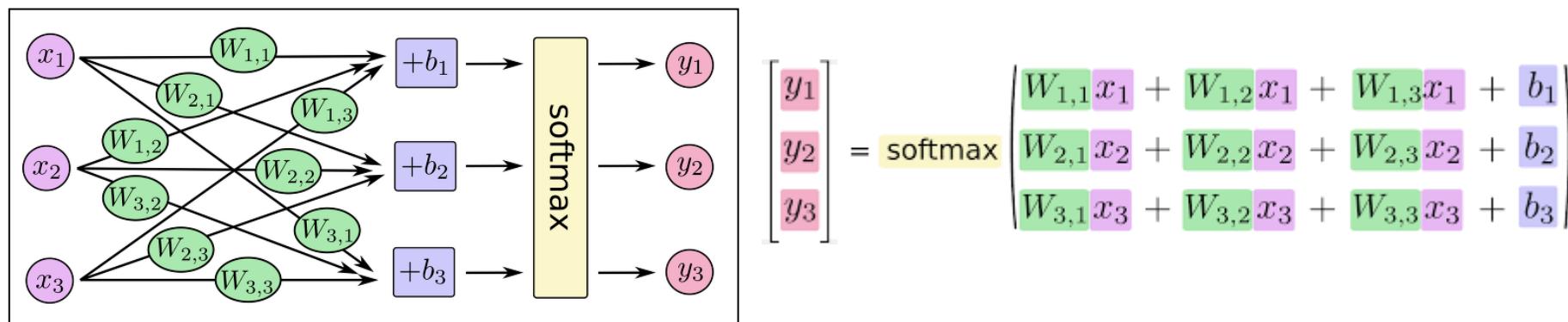
- 분류하고 싶은 클래스 수로 설정
- 예를 들어 숫자 0에서 9까지 하나로 분류하고 싶으면 출력층의 Node를 10개로 설정



분류 문제를 풀기 위해서 출력 층에서는 출력 값으로 확률 벡터를 얻는 softmax 함수를 자주 사용한다.

▪ Softmax 함수

- 출력 층에서의 Activation Function으로 자주 사용
- 인공신경망의 출력 값으로 확률 벡터를 얻고 싶을 때 사용 (출력 값의 해석을 용이하게 한다.)
- 분류 문제에서 학습 단계에 사용된 Softmax 함수는 추론 시에 생각하는 경우가 많다. (출력 값이 큰 Node는 변하지 않음으로...)



▪ Softmax 함수 원리

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \xrightarrow{\text{Softmax}} \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

- Softmax 함수는 각 output node의 출력을 0~1로 제한
- output node의 출력을 모두 합한 값은 항상 1
- Softmax function의 수식

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^J e^{y_j}}$$

e^{y_i} : 지수함수 (exponential function)

Python으로 Softmax함수를 구현해 보자

- Python으로 구현하기

```
def softmaxFunction(x):
    expX = np.exp(x)          # 지수 함수 (exponential function)
    sumExpX = np.sum(expX)    # 지수 함수 결과의 합
    return expX / sumExpX

a = np.array([2.3, -0.9, 3.6])
y4 = softmaxFunction(a)
print(y4, np.sum(y4))
```

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^J e^{y_j}}$$

- 지수 함수 계산 결과가 매우 크면 overflow가 발생할 수 있다.

```
def softmaxFunction(x):
    expX = np.exp(x)          # 지수 함수 (exponential function)
    sumExpX = np.sum(expX)    # 지수 함수 결과의 합
    return expX / sumExpX

a = np.array([900, 1000, 1000])
y4 = softmaxFunction(a)
print(y4, np.sum(y4))
```

- 실행 결과 확인

`Runtimewarning: overflow encountered in exp
[nan nan nan] nan`

- 이를 해결하기 위해 Softmax 함수 구현 시 입력 값 중 최대값을 찾아 빼주면 올바르게 계산할 수 있다.

```
def softmaxFunction(x):
    expX = np.exp(x - np.max(x))      # Overflow 대비를 위해 원소 중 최대값을 각 입력 신호에서 빼서 예방한다.
    sumExpX = np.sum(expX)            # 지수 함수 결과의 합
    return expX / sumExpX

a = np.array([900, 1000, 1000])
y4 = softmaxFunction(a)
print(y4, np.sum(y4))
```

6. MNIST Example

III. Neural Network

MNIST Data Set은 Handwritten Digit Image 집합이다.

- **MNIST**(Mixed National Institute of Standards and Technology) 란?

- 기계학습 분야에서 사용되는 손글씨 숫자 이미지 데이터 셋
- 실험용 데이터로 자주 사용된다.
- 0부터 9까지 숫자 이미지로 구성되어 있다
- 각 이미지가 실제 의미하는 숫자가 Label되어 있다.
- 훈련 이미지가 6만장, 시험 이미지가 1만장으로 준비되어 있다.
- 각 숫자는 가로, 세로 각각 28 픽셀 크기의 흑백 이미지이다
- https://en.wikipedia.org/wiki/MNIST_database



- **기계학습 알고리즘에 사용된 예**

Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6 ^[9]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52 ^[17]
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87 ^[18]
Non-Linear Classifier	40 PCA + quadratic classifier	None	None	3.3 ^[9]
Support vector machine	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56 ^[19]
Neural network	2-layer 784-800-10	None	None	1.6 ^[20]
Neural network	2-layer 784-800-10	elastic distortions	None	0.7 ^[20]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	elastic distortions	None	0.35 ^[21]
Convolutional neural network	6-layer 784-40-80-500-1000-2000-10	None	Expansion of the training data	0.31 ^[14]
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.27 ^[15]
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	elastic distortions	Width normalizations	0.23 ^[8]
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.21 ^[16]

6. MNIST Example

III. Neural Network

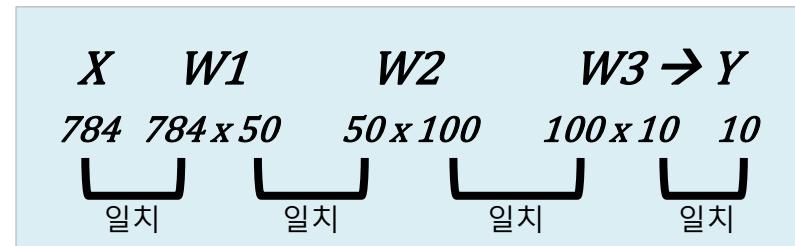
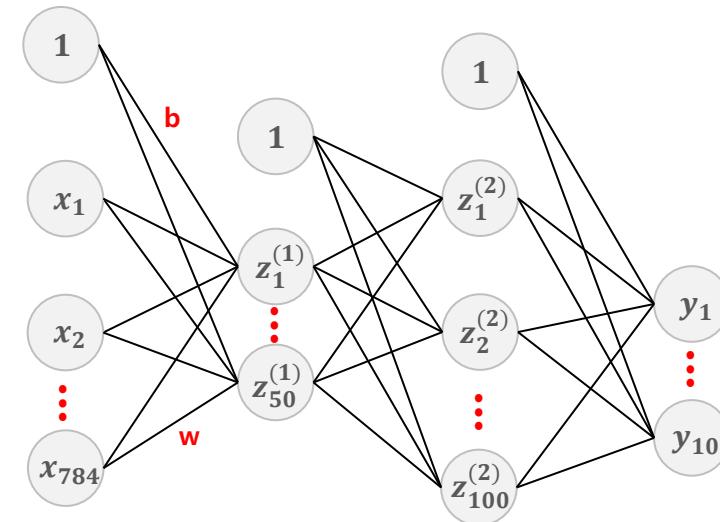
MNIST를 위한 신경망 설계 및 추론 과정

▪ MNIST를 위한 신경망 설계

- 입력층 : 784개 (이미지 크기가 28x28 픽셀임으로)
- 출력층 : 10개 (0에서 9로 분류함으로)
- 첫번째 은닉층 : 50개 (임의의 수)
- 두번째 은닉층 : 100개 (임의의 수)

▪ MNIST Example 준비

- 훈련데이터(6만장), 검증데이터(1만장) 다운로드
 - ✓ 데이터 정규화 : 0~255의 픽셀 값을 0.0~1.0으로 변환
 - ✓ 데이터 평탄화 : 28x28 배열을 784 배열로 평탄화
 - ✓ 레이블 one-hot encoding : 정답을 뜻하는 원소만 1이고 나머지는 0으로 표기법
예) [0,0,1,0,0,0,0,0,0] 는 2를 나타냄
- 가중치, 편향 값 설정



- 신경망 각 층의 배열 형상 추이
(Bias와 Activation Function은 생략)

6. MNIST Example

III. Neural Network

MINST를 위해 설계된 신경망을 구현하기

- Python으로 구현하기

```
import numpy as np
import pickle
from Common.mnist import load_mnist
from Common.functions import sigmoid, softmax

def getData():

    (trainImg, trainLbl), (testImg, testLbl) = \
        load_mnist(normalize=True, flatten=True, one_hot_label=False)

    print(trainImg.shape) # (60000, 784)
    print(trainLbl.shape) # (60000,)
    print(testImg.shape) # (10000, 784)
    print(testLbl.shape) # (10000,)

    return (trainImg, trainLbl), (testImg, testLbl)

def initNetwork():
    with open("../resources/sample_weight.pkl", "rb") as f:
        network = pickle.load(f)
    return network

def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)
    return y

    ...

Load dataset and initialize weight & bias
'''

(trainImg, trainLbl), (testImg, testLbl) = getData()
network = initNetwork();

'''

show first element
'''

firstImg = trainImg[0].reshape(28,28)
firstLbl = trainLbl[0]
print(firstLbl) # 5
plt.imshow(firstImg, cmap='gray', interpolation='nearest')
plt.show()

'''

predict
'''

accuracyCnt = 0
print(len(trainImg)) # 60000
for idx in range(len(trainImg)):
    y = predict(network, trainImg[idx])
    p = np.argmax(y)
    if p == trainLbl[idx]:
        accuracyCnt += 1

print(accuracyCnt)
print("Accuracy:" + str(accuracyCnt / len(trainImg)))
```

6. MNIST Example

III. Neural Network

MINST 배치 처리를 통한 빠른 연산 수행

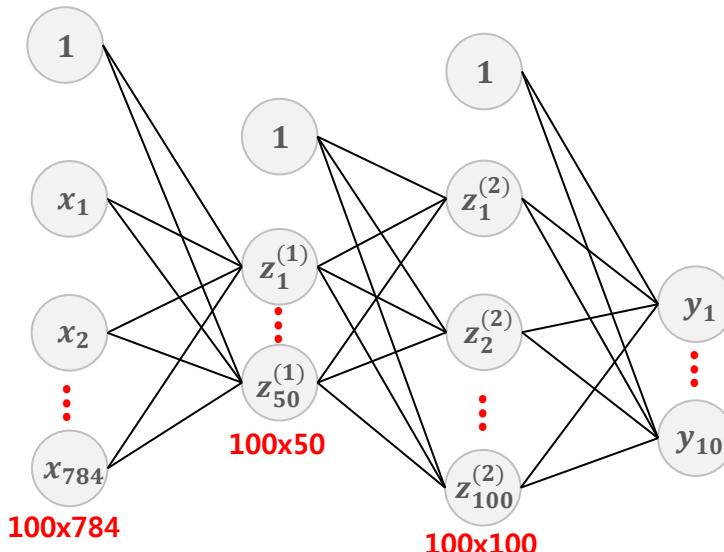
- 이미지 여러장을 한꺼번에 처리하기

- 100장을 한꺼번에 처리하는 경우의 형상

- ✓ 입력층은 100×784
- ✓ 첫번째 은닉층은 100×50
- ✓ 두번째 은닉층은 100×100
- ✓ 출력층은 100×10

- 배치 처리의 장점

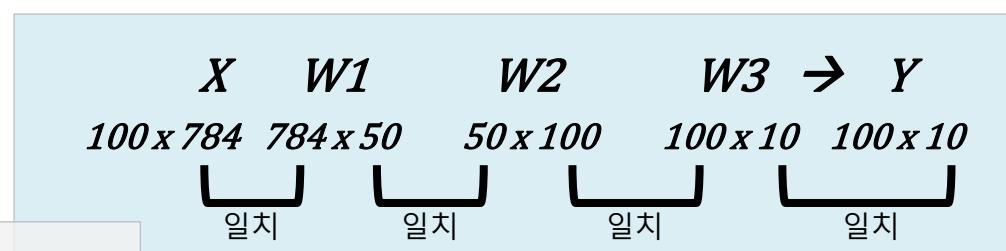
- 수치 계산 라이브러리 대부분이 큰 배열을 효율적으로 처리하는 데 최적화 되어 있음.
- 데이터 I/O 횟수를 줄여, CPU, GPU가 계산에 집중



- Python으로 구현하기

```
...
predict
...
batchSize = 100
accuracyCnt = 0
print(len(trainImg)) # 60000
for idx in range(0, len(trainImg), batchSize):
    y = predict(network, trainImg[idx:idx+batchSize])
    p = np.argmax(y, axis=1)
    accuracyCnt += np.sum(p == trainLbl[idx:idx+batchSize])

print(accuracyCnt)
print("Accuracy:" + str(accuracyCnt / len(trainImg)))
```



- 신경망 각 층의 배열 형상 추이
(Bias와 Activation Function은 생략)

#1 – Activation Function 구현하기

▪ myactivation.py

- 계단 함수, Sigmoid 함수, ReLU 함수를 구현해보고 시각화해 본다.

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

def stepFunction(x):
    y = [] # Empty List
    for value in x:
        if value > 0:
            y.append(1)
        else:
            y.append(0)

    return np.array(y)

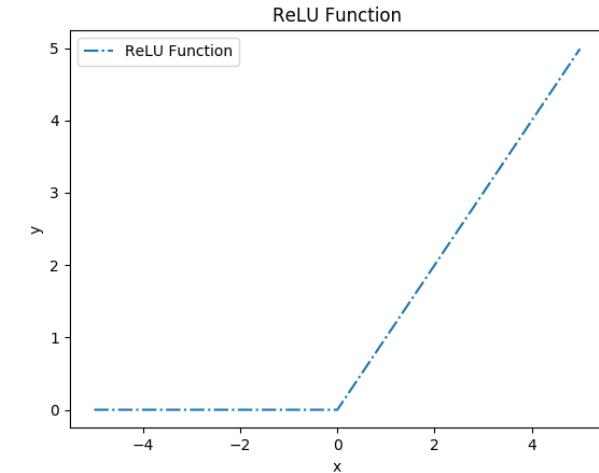
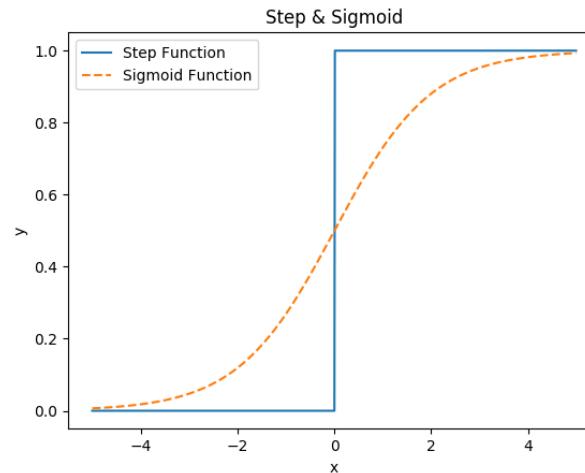
def sigmoidFunction(x):
    return 1/(1+np.exp(-x))

def reluFunction(x):
    return np.maximum(0, x)

x = np.arange(-5, 5, 0.01)
y1 = stepFunction(x)
y2 = sigmoidFunction(x)
y3 = reluFunction(x)
```

```
plt.plot(x, y1, label="Step Function" ,linestyle="-")
plt.plot(x, y2, label="Sigmoid Function",linestyle="--")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Step & Sigmoid")
plt.legend()
plt.show()

plt.plot(x, y3, label="ReLU Function" ,linestyle="-.")
plt.xlabel("x")
plt.ylabel("y")
plt.title("ReLU Function")
plt.legend()
plt.show()
```



#2 – Matrix Inner Product 계산해보기

▪ myinnerproduct.py

- numpy.ndarray 를 이용하여 배열을 생성하고 내적을 계산한다.

```
import numpy as np

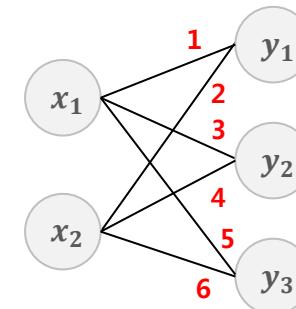
npArr1 = np.array([[1,2],[3,4]])          # 2x2 행렬 (2차원배열)
npArr2 = np.array([[5,6],[7,8]])          # 2x2 행렬 (2차원배열)
npArr3 = np.array([[1,2,3],[4,5,6]])      # 2x3 행렬 (2차원배열)
npArr4 = np.array([[5,6],[7,8],[9,10]])    # 3x2 행렬 (2차원배열)
npArr5 = np.array([1,2])                  # 2x1 벡터 (1차원배열)

npResult1 = np.dot(npArr1, npArr2) # matrix inner product (2x2 행렬)
npResult2 = np.dot(npArr3, npArr4) # matrix inner product (2x2 행렬)
npResult3 = np.dot(npArr4, npArr3) # matrix inner product (3x3 행렬)
npResult4 = np.dot(npArr4, npArr5) # matrix inner product (3x1 벡터)

print(npResult1)
print(npResult2)
print(npResult3)
print(npResult4)
```

- 신경망에 적용해 본다.

```
preNodes = np.array([1,2]) # 신경망의 이전계층의 노드 값
weight = np.array([[1,3,5],[2,4,6]]) # 가중치
netInput = np.dot(preNodes, weight) # 노드값과 가중치의 연산
print(netInput)
```



7. Hands-on

III. Neural Network

#3 – 3층 신경망 구현해보기

- myfeedforward.py

- 3층 신경망을 구성하고 신경망 계산 결과를 확인한다.

```
import numpy as np

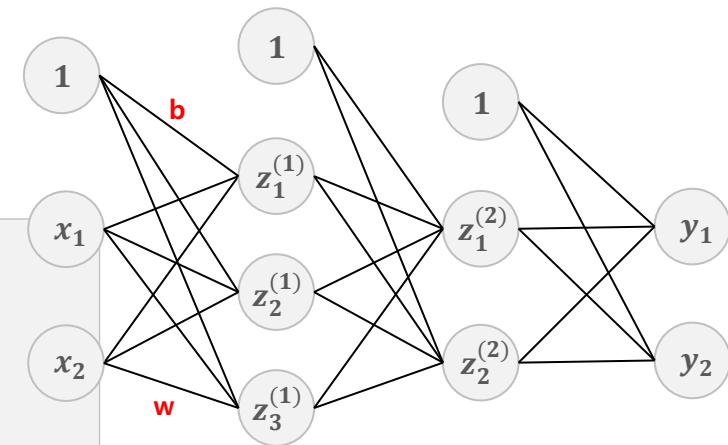
def initNeuralNetwork():
    neuralNetwork = {} # Python Dictionary 자료형
    neuralNetwork['W1'] = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
    neuralNetwork['W2'] = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
    neuralNetwork['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])
    neuralNetwork['b1'] = np.array([0.7, 0.8, 0.9])
    neuralNetwork['b2'] = np.array([0.7, 0.8])
    neuralNetwork['b3'] = np.array([0.7, 0.8])

    return neuralNetwork

def sigmoidFunction(x):
    return 1/(1+np.exp(-x))

def identityFunction(x):
    return x

def feedForward(neuralNetwork, ia):
    # 신경망 초기값을 가져온다.
    W1, W2, W3 = neuralNetwork['W1'], neuralNetwork['W2'], neuralNetwork['W3']
    b1, b2, b3 = neuralNetwork['b1'], neuralNetwork['b2'], neuralNetwork['b3']
    # 첫번째 은닉층의 계산
    z1 = sigmoidFunction(np.dot(ia, W1) + b1)
    print("1st 은닉층 값 : " + str(z1))
    # 두번째 은닉층의 계산
    z2 = sigmoidFunction(np.dot(z1, W2) + b2)
    print("2nd 은닉층 값 : " + str(z2))
    # 출력층의 계산
    y = identityFunction(np.dot(z2, W3) + b3)
    return y
```



```
neuralNetwork = initNeuralNetwork()
ia = np.array([4.5, 6.2])
print("입력층 값 : " + str(ia))
oa = feedForward(neuralNetwork, ia)
print("출력층 값 : " + str(oa))
```

- 실행 결과 확인

```
입력층 값 : [ 4.5 6.2]
1st 은닉층 값 : [ 0.91606157  0.99033948  0.9989626 ]
2nd 은닉층 값 : [ 0.78403932  0.90559276]
출력층 값 : [ 0.95952248  1.3974489 ]
```

#4 – Softmax 함수 구현보기

▪ myactivation.py

- Softmax 함수를 추가 구현하고 실행해 본다.

... 코드 생략 ...

```
'''  
Softmax 함수 구현  
입력파라미터 numpy.ndarray  
출력파라미터 numpy.ndarray  
'''  
  
def softmaxFunction(x):  
    expX = np.exp(x - np.max(x))      # Overflow 대비를 위해 원소 중 최대값을 각 입력 신호에서 빼서 예방한다.  
    sumExpX = np.sum(expX)            # 지수 함수 결과의 합  
    return expX / sumExpX  
  
... 코드 생략 ...  
  
a = np.array([2.3, -0.9, 3.6])  
y4 = softmaxFunction(a)  
print(y4, np.sum(y4))  
  
a = np.array([900, 1000, 1000])  
y4 = softmaxFunction(a)  
print(y4, np.sum(y4))
```

- 실행 결과 확인

```
[ 0.21231157  0.00865429  0.77903414] 1.0  
[ 1.86003799e-44   5.00000000e-01   5.00000000e-01] 1.0
```

#5 – MNIST 실습해보기

- **mymnist.py**

- 제공되는 파일을 이용하여 MNIST 를 위한 신경망을 구축해 본다.
- 제공되는 파일 : Common/mnist.py, Common/functions.py, resources/sample_weight.pkl
- 전체 소스코드 : MNIST Example – Python으로 구현하기 참조
- 실행 결과 확인

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
5
60000
56146
Accuracy:0.9357666666666666
```

➔ 올바르게 분류한 비율이 93.57%

#6 – MNIST 실습해보기 (배치처리)

▪ mymnist.py

- 100장의 이미지를 한꺼번에 처리하여 효율적으로 처리해 본다.

... 코드 생략 ...

```
...
predict
...
batchSize = 100
accuracyCnt = 0
print(len(trainImg)) # 60000
for idx in range(0, len(trainImg), batchSize):
    y = predict(network, trainImg[idx:idx+batchSize])
    p = np.argmax(y, axis=1)
    accuracyCnt += np.sum(p == trainLbl[idx:idx+batchSize])

print(accuracyCnt)
print("Accuracy:" + str(accuracyCnt / len(trainImg)))
```

- 실행 결과 확인

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
5
60000
56146
Accuracy:0.9357666666666666
```

➔ 계산 속도를 체감 비교해 본다.

IV. Training



1. Neural Network Training
 2. Cost Function
 3. Numerical Differentiation
 4. Gradient Descent Method
 5. MNIST Example
- b. Hands-on

1. Neural Network Training

IV. Training

학습이란 신경망에서 원하는 결과를 얻기 위해 뉴런 사이의 적당한 가중치를 알아내는 것. 즉 **가중치를 최적화 하는 것이다.**

- 신경망에서 지도학습

- 훈련 데이터 (Training Set) 준비 : 입력 데이터와 출력 데이터
- 신경망에 데이터 훈련 : 출력층 값은 확인
- 지도학습 데이터와 차이 (오차) 계산
- 오차가 최대한 작도록 가중치를 최적화

Supervised Learning

- 오차를 어떻게 측정하나요?

- Cost Function(비용함수)을 이용한다.

- 가중치를 어떻게 최적화 할까요?

- Gradient Method(경사하강법)을 이용한다.

- Gradient Method에서 경사는 어떻게 계산하나요?

- 수치 미분을 사용해 구할 수 있으나 계산 시간이 매우 오래 걸린다.

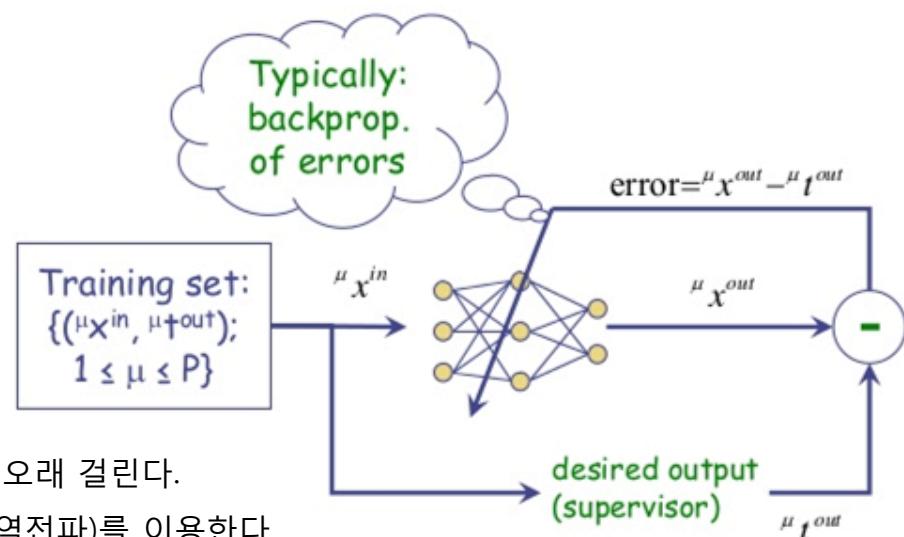
따라서 기울기를 효율적으로 계산하는 Back Propagation(역전파)를 이용한다.

- 심층 신경망(Deep Neural Network)에서도 학습이 잘 되게 하려면?

- Pre-Training(사전학습)을 이용한다.

- 심층 신경망(Deep Neural Network)에서 더 정밀하게 최적화하는 방법은 무엇인가요?

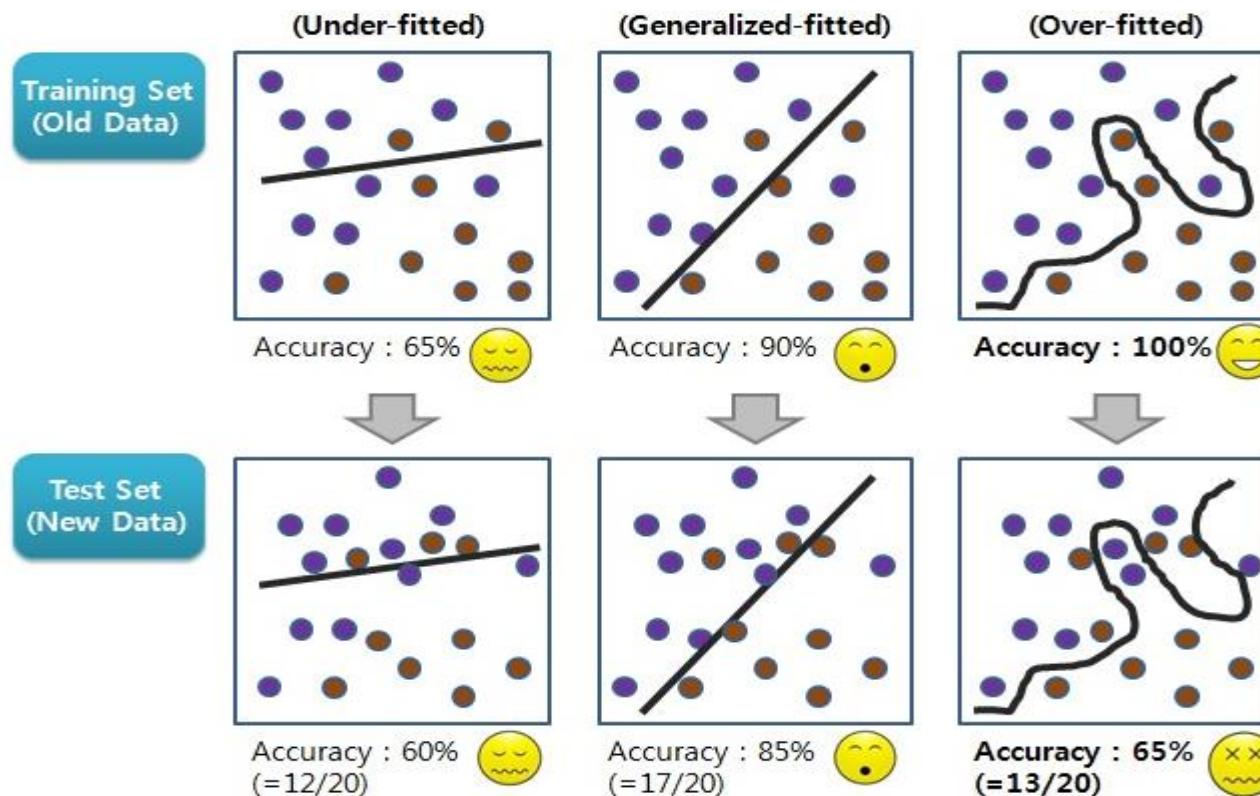
- Dropout 알고리즘을 이용한다.



1. Neural Network Training

IV. Training

학습 데이터에 너무 지나치게 맞추다 보면 일반화 성능이 떨어지는 모델을 얻게 되는 현상을 **과적합(Overfitting)**이라고 한다.



▪ Under Fitting

적정 수준의 학습을 하지 못하여 실제 성능이 떨어지는 경우

▪ Normal Fitting (Generalized Fitting)

적정 수준의 학습으로 실제 적정한 일반화 수준을 나타냄. 기계 학습이 지향하는 수준.

▪ Over Fitting

학습 데이터에 성능이 좋지만 실제 데이터에 관해 성능이 떨어짐. 특히 조심해야 함.

비용함수란 신경망 학습에서 학습 데이터에 대한 오차를 측정하는 척도이다.

- 평균제곱오차 (mean squared error, MSE)

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2$$

- y_k : 신경망의 출력 (신경망이 추정한 값)
- t_k : 정답 레이블
- k : 데이터의 차원 수 (MNIST에서는 0에서 9의 10개)

- 파이썬으로 구현 및 사용

```
import numpy as np

def meanSquaredError(y, t):
    return 0.5*np.sum((y-t)**2)

t = np.array([0, 0, 0, 0, 1, 0, 0, 0, 0]) # label = 5
y = np.array([0.1, 0.03, 0.05, 0.2, 0.9, 0.0, 0.1, 0.2, 0.12, 0.03]) # 5라고 추정
# 정답일 경우 MSE, CEE의 같은 적다.
print("-- 정답인 경우 --")
print("MSE :", meanSquaredError(y, t))

y = np.array([0.1, 0.03, 0.05, 0.2, 0.0, 0.1, 0.2, 0.12, 0.03, 0.9]) # 9라고 추정
# 오류일 경우 MSE, CEE의 같은 적다.
print("-- 오류인 경우 --")
print("MSE :", meanSquaredError(y, t))
```

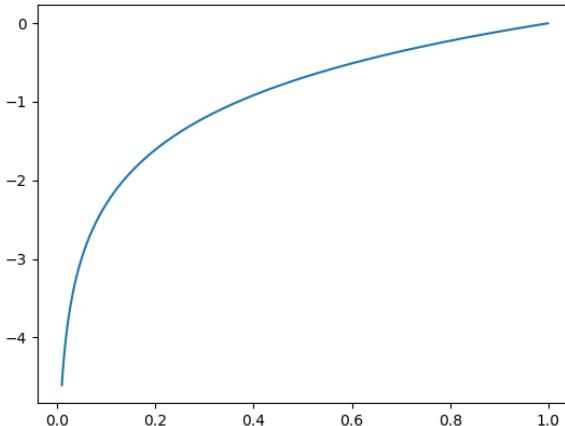
교차엔트로피오차 함수는 활성함수로 Sigmoid함수나 Softmax함수를 채택한 신경망과 함께 사용되는 경우가 많다.

- 교차 엔트로피 오차 (cross entropy error, CEE)

$$E = - \sum_k t_k \log y_k$$

- y_k : 신경망의 출력 (0에서 1사이어야 한다)
- t_k : 정답 레이블 (정답이 아닌 나머지는 t_k 가 0임으로 곱해도 0이 된다.)
- \log : 밑이 e인 자연로그 ($\log_e y_k$)

- 자연로그 e의 그래프 ($y = \log x$)



- x 가 1일 때 y 는 0이 된다.
- x 가 0에 가까워질수록 y 값은 점점 작아진다
- x 가 0이면 y 는 $-\infty$ 가 된다. (무한대)

- 파이썬으로 구현 및 사용

```
import numpy as np

def crossEntropyError(y, t):
    delta = 1e-7 #아주 작은 값 (y가 0인 경우 -inf 값을 예방)
    return -np.sum(t*np.log(y+delta))

t = np.array([0, 0, 0, 0, 1, 0, 0, 0, 0]) # label = 5
y = np.array([0.1, 0.03, 0.05, 0.2, 0.9, 0.0, 0.1, 0.2, 0.12, 0.03])
# 정답일 경우 MSE, CEE의 값은 적다.
print("-- 정답인 경우 --")
print("CEE :", crossEntropyError(y, t))

y = np.array([0.1, 0.03, 0.05, 0.2, 0.0, 0.1, 0.2, 0.12, 0.03, 0.9])
# 오류일 경우 MSE, CEE의 값은 적다.
print("-- 오류인 경우 --")
print("CEE :", crossEntropyError(y, t))
```

Mini-batch 학습이란 훈련 데이터로부터 일부만 골라서 학습을 수행하는 방식이다.

▪ Mini-batch 학습

- 가중치를 조절하기 위한 비용함수의 계산은 모든 훈련데이터를 1회 마치고 비용 함수의 합을 구해야 한다.
 - 하지만 데이터가 많은 경우 (MNIST는 6만개) 손실함수 이렇게 구하는 건 비 현실적이다.
 - 이를 개선하기 위해 신경망 학습에서 훈련 데이터의 일부만 골라 학습하는 방법을 **미니배치 학습**이라고 한다.
(즉, 표본을 뽑아 학습한다.)

▪ 파이썬으로 구현 (MNIST 예제)

```
...  
predict  
- MiniBatch Target Setting  
- Predict (추론 수행)  
- Cost Function (비용함수계산)  
...  
  
miniBatchSize = 10  
print(len(trainImg), "개 중에", miniBatchSize, "를 미니배치 학습.")  
miniBatchMask = np.random.choice(len(trainImg), miniBatchSize)  
trainImgBatch = trainImg[miniBatchMask]  
trainLblBatch = trainLbl[miniBatchMask]  
print("Random Choice 된 정답 레이블", trainLblBatch)  
  
y = predict(network, trainImgBatch)  
print("신경망 출력 결과 (10-배치크기, 10-분류크기) Shape\n", y)  
p = np.argmax(y, axis=1)  
print("최대값으로 추정한 결과", p)  
  
print(crossEntropyError(y, trainLblBatch))
```

- 실행 결과 확인

```
60000 개 중에 10 를 미니배치 학습.  
Random Choice 된 정답 레이블 [4 5 1 8 1 2 2 7 7 4]  
신경망 출력 결과 (10-배치크기, 10-분류크기) Shape  
[[ 1.41822547e-05    2.35181437e-06  
 1.51875982e-04    3.35034133e-06  
... 결과 생략 ...  
  
최대값으로 추정한 결과 [4 5 1 8 1 2 2 7 7 4]  
0.12456381321
```

- 비용함수인 crossEntropyError 함수의 구현도 배치 처리가 가능하게 변경되어야 함. (코드 생략)

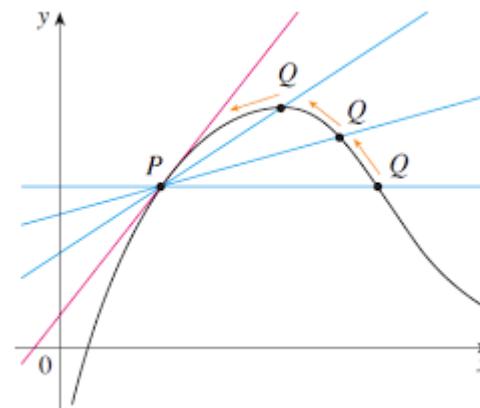
미분(Differentiation)은 특정 순간의 변화량이다.

▪ 미분 수식

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

- $\frac{df(x)}{dx}$: 미분, 즉 x 에 대한 $f(x)$ 의 변화량
 - h : 시간
 - $\lim_{h \rightarrow 0}$: 시간이 0에 가까워짐.

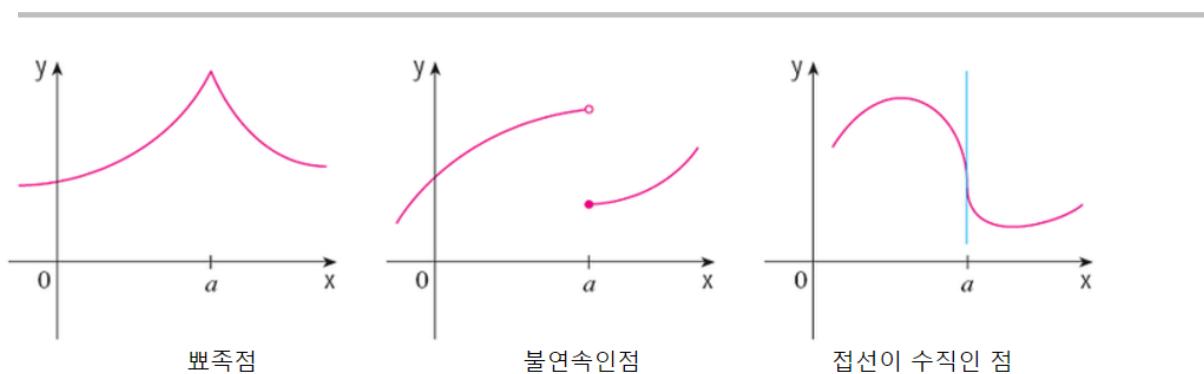
- 연속적이고 매끄러운 함수 (미분 가능한 함수)에서 특정 지점의 **접선의 기울기**는 그 점에서의 미분 값과 같다



- ✓ 점 Q가 점 P로 움직일 경우 직선 PQ의 기울기는 점 P에서의 접선의 기울기와 같다.

?

미분 가능하지 않는 경우



3. Numerical Differentiation

IV. Training

수치 미분(Numerical Differentiation)은 근사치를 이용해서 계산하는 방법이다.

▪ 수치 미분

- 시간을 나타내는 h 에 가급적 낮은 값을 넣어야 하는데 너무 작은 값 (예, 0.00....1) 을 넣으면 컴퓨터 계산에 문제가 발생한다. .
- 시간 h 는 무한의 0으로 줍히는 것은 불가능 하다.
- 따라서 근사치를 이용하여 계산한다. (h 는 0.0001이 가장 좋다)

▪ 파이썬으로 구현

```
def numericalDiff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)
```

▪ 수치 미분의 예

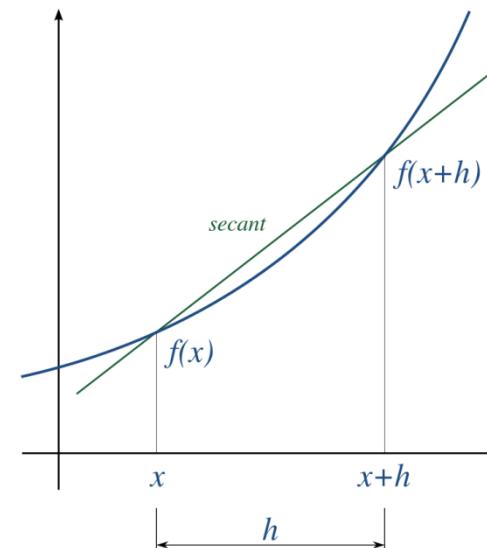
- $y = 0.01x^2 + 0.1x$ 의 수치 미분

```
def numericalDiff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)

def sampleFunc1(x):
    return 0.01*x**2 + 0.1*x

x = np.arange(0, 20, 0.1)
y = sampleFunc1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y)
plt.show()

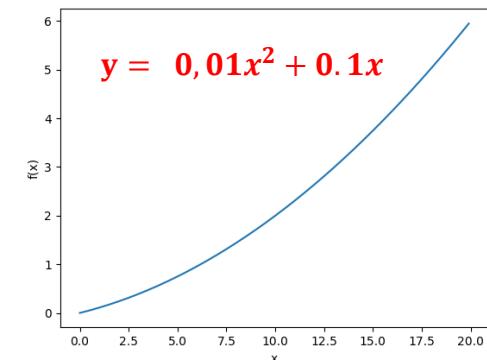
print(numericalDiff(sampleFunc1, 5))
print(numericalDiff(sampleFunc1, 10))
```



- 실행 결과 확인

```
0.1999999999990898
0.2999999999986347
```

- 수치 미분이 아닌 진정한 미분은 각각 0.2와 0.3이다.
- 수치 미분으로 계산하더라도 값의 차는 거의 없다.



변수가 2개 이상인 함수를 미분할 때 미분 대상 변수 외에 나머지 변수를 상수처럼 고정시켜 미분하는 것을 편미분이라 한다.

- 편 미분의 예

- $f(x_0, x_1) = x_0^2 + x_1^2$ 인 경우 $\frac{\partial f}{\partial x_0}$ 나 $\frac{\partial f}{\partial x_1}$ 처럼 쓴다.

- $x_0 = 3, x_1 = 4$ 일 때, 편미분 $\frac{\partial f}{\partial x_0}$ 을 구한다.

```
def numericalDiff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)

def sampleFunc2(x):
    return x**2 + 4**2

print(numericalDiff(sampleFunc2, 3.0))
```

- $x_0 = 3, x_1 = 4$ 일 때, 편미분 $\frac{\partial f}{\partial x_1}$ 을 구한다.

```
def numericalDiff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)

def sampleFunc3(x):
    return 3**2 + x**2

print(numericalDiff(sampleFunc3, 4.0))
```

모든 변수의 편미분을 벡터로 정리한 것을 **기울기(Gradient)**라고 한다.

▪ 기울기의 예

- $f(x_0, x_1) = x_0^2 + x_1^2$ 인 경우 기울기는 $(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1})$ 이다.

- $f(x_0, x_1) = x_0^2 + x_1^2$ 인 함수의 기울기 구하기 (변수 0,0 일 때 최소값 0을 가지는 함수)

```
def numeiralGradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) #x와 형상이 같은 배열을 생성
    for idx in range(x.size):
        tmpVal = x[idx]
        x[idx] = tmpVal + h
        print(idx, x)
        fxh1 = f(x)

        x[idx] = tmpVal - h
        print(idx, x)
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmpVal # 원래 값 복원

    return grad

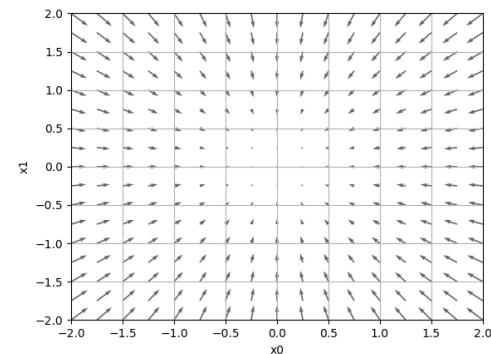
#  $f(x_0, x_1) = x_0^{**2} + x_1^{**2}$ 인 함수
def sampleFunc4(x):
    return x[0]**2 + x[1]**2

print(numeiralGradient(sampleFunc4, np.array([3.0, 4.0])))
print(numeiralGradient(sampleFunc4, np.array([0.0, 2.0])))
print(numeiralGradient(sampleFunc4, np.array([3.0, 0.0])))
```

• 실행 결과 확인

```
0 [ 3.0001 4.      ]
0 [ 2.9999 4.      ]
1 [ 3.       4.0001]
1 [ 3.       3.9999]
[ 6.     8.]
0 [ 1.0000000e-04
 2.0000000e+00]
0 [ -1.0000000e-04
 2.0000000e+00]
1 [ 0.       2.0001]
1 [ 0.       1.9999]
[ 0.     4.]
0 [ 3.0001 0.      ]
0 [ 2.9999 0.      ]
1 [ 3.0000000e+00
 1.0000000e-04]
1 [ 3.0000000e+00
 1.0000000e-04]
[ 6.     0.]
```

- 점 (3,4)에서의 기울기는 (6,8)
- 점 (0,2)에서의 기울기는 (0,4)
- 점 (3,0)에서의 기울기는 (6,0)



이 예제에서 최솟값은 x_0, x_1 가 0,0 일 때 함수 f 는 0 값을 가진다. 하지만, 신경망에서는 함수의 최소값(최소 오차)을 가지기 위해서 변수 x (가중치) 가 얼마 인지 알 수 없음으로 “**기울기의 반대 방향으로 갈수록 최솟값이 가까워짐**” 것을 이용하여 x 를 조정한다.

4. Gradient Descent Method

IV. Training

Gradient Descent Method의 최솟값을 계산하는 알고리즘을 이용해 신경망의 연결가중치를 최적화한다.

▪ Gradient Descent Method 란?

수치 계산에서 함수의 최솟값을 계산할 때 사용되는 가장 보편적인 알고리즘

a. 먼저 변수의 초기값을 선정

b. 변수 값에 해당하는 함수의 경사도 계산

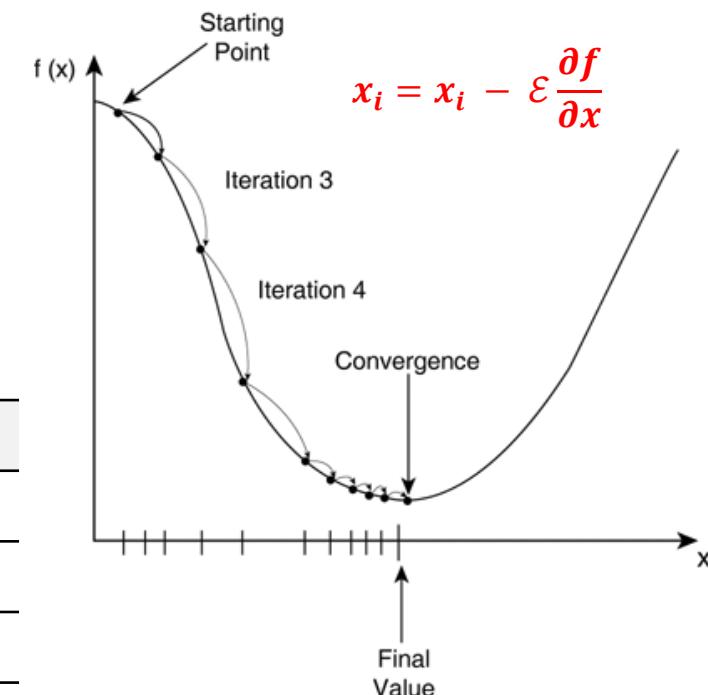
c. 변수를 경사 방향으로 움직여 다음 숫자값으로 설정

a~c를 반복하여 함수의 최솟값이 되는 변수값으로 근접해 나간다.

▪ Gradient Descent Method를 신경망에 적용

경사 하강법	신경망
변수 x	각 연결의 Weight
함수 f	비용함수(손실함수)
함수의 경사 $\frac{\partial f}{\partial x}$	가중치 매개변수에 대한 비용함수의 기울기
움직임의 정도 ϵ	학습률 (미리 적정한 값으로 정해야 함)

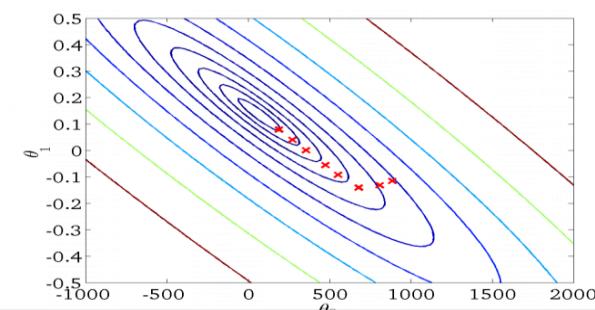
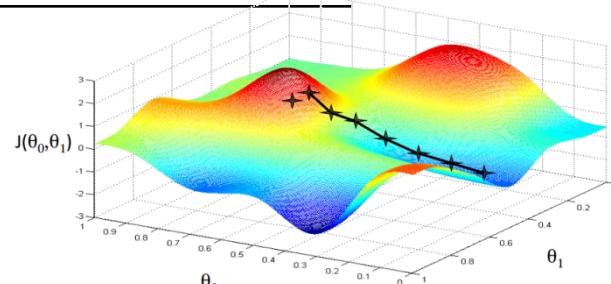
→ 비용함수를 계산하는 부분에서 훈련 데이터가 늘어날수록 시간이 많이 걸림.



2차원에서의 Gradient Descent Method
(Parameter가 하나인 경우)



Parameter가 두 개인 경우



4. Gradient Descent Method

IV. Training

$f(x_0, x_1) = x_0^2 + x_1^2$ 의 최솟값을 구하기 위해 Gradient(기울기)에 학습률을 이용하여 반복학습 해 본다. (x 값을 조정)

▪ 파이썬으로 구현

```
import numpy as np
import matplotlib.pyplot as plt
from Common.gradient_2d import numerical_gradient

'''
f = x[0]**2 + x[1]**2
initX = 초기값 (-3,4)
초기값에서 함수에 대한 기울기 (편미분에 의한 계산)를 구한 후
기울기 값과 학습률을 이용해서 x값을 이동
위를 stepNum(100번) 반복
'''

def gradientDescent(f, initX, learningRate, stepNum=100):
    x = initX
    xHistory = []

    for i in range(stepNum):
        xHistory.append(x.copy())
        grad = numerical_gradient(f,x)
        x -= learningRate*grad
        if(i == stepNum-1):
            print(x)

    return x, np.array(xHistory)

def sampleFunction(x):
    return x[0]**2 + x[1]**2
```

```
# 적정학습률
initX = np.array([-3.0,4.0])
learningRate = 0.1
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)

# 학습률이 큰 경우
initX = np.array([-3.0,4.0])
learningRate = 2
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)

# 학습률이 작은 경우
initX = np.array([-3.0,4.0])
learningRate = 0.001
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)
```

- 실행 결과 확인 (100회 반복 후 x 값의 조정된 모습)

```
[ -6.11110793e-10   8.14814391e-10]
[ -1.35262250e+12   2.26052540e+12]
[ -2.45570041   3.27426722]
```



Hyper parameter

- 학습에 의한 가중치 조정과는 다르게 학습률 같이 사람이 직접 설정하는 매개변수

4. Gradient Descent Method

IV. Training

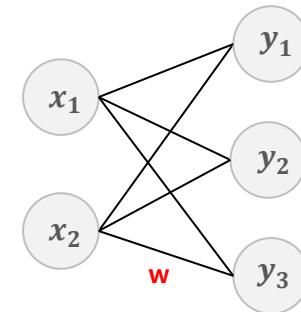
신경망에서의 기울기는 가중치에 대한 비용 함수(손실 함수)의 기울기이다.

- 예) 2x3 형상의 가중치 W , 손실 함수 L 인 신경망의 기울기는 아래와 같다.

- 가중치 W 의 형상과 기울기 $\frac{\partial L}{\partial W}$ 의 형상이 같다.

$$W = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \end{pmatrix}$$



- 입력층 2개, 출력층 3개, 은닉층이 없는 신경망에서 기울기를 구현해보자

```
import numpy as np
from Common.functions import softmax, cross_entropy_error
from Common.gradient import numerical_gradient

class SimpleNet:

    def __init__(self):
        self.W = np.random.randn(2,3) # 정규분포로 가중치 초기화

    def predict(self, x):
        z = np.dot(x, self.W)
        y = softmax(z)
        return y

    def loss(self, x, t):
        y = self.predict(x)
        loss = cross_entropy_error(y, t)
        return loss
```

```
simpleNet = SimpleNet()
x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])

def f(w): #w는 dummy
    return simpleNet.loss(x, t)

numerical_gradient(f,x)에서
f는 손실함수, x는 손실함수 f의 인수
즉, simpleNet.W[0,0], [0,1], [0,2], [1,0], [1,1], [1,2]
의 값으로 손실함수를 편미분한 결과 (Gradient)
...
gradient = numerical_gradient(f, simpleNet.W)
print("gradient:\n", gradient)
```

- 실행 결과 확인

```
gradient:
[[ 0.14511488  0.08426216 -0.22937704]
 [ 0.21767231  0.12639324 -0.34406556]]
```

“기울기의 반대 방향으로 갈수록 최솟값이 가까워짐”

→ 가중치 값을 기울기 반대로 조정하면 손실(오차)이 적어짐

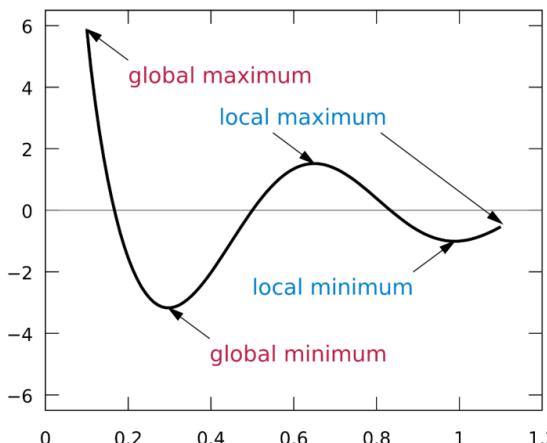
4. Gradient Descent Method

IV. Training

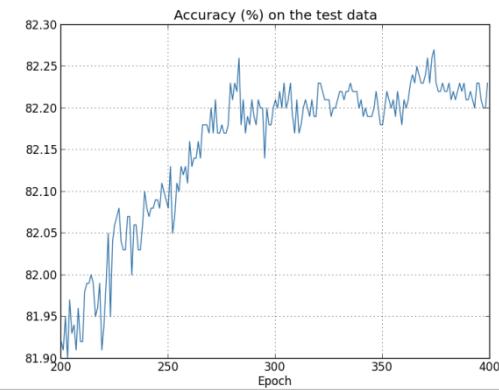
신경망의 지도학습에서 가중치를 갱신 값을 계산하는 방법(Gradient Descent Method)은 3가지가 있다.

▪ Gradient Descent Method의 종류

종류	계산방법
(Batch) Gradient Descent	<ul style="list-style-type: none">모든 데이터를 학습하고 오차를 이용하여 가중치 갱신 값을 계산한 다음 갱신단점 : 학습 데이터가 많으면 가중치 갱신 값 평균 계산 시 계산량 많음. SGD 대비 학습이 오래 걸림.단점 : 가중치 x의 초기 설정에 따라 Global minimum 값이 아닌 local minimum 값을 수렴할 수 있다.
Stochastic Gradient Descent (확률적 경사 하강법)	<ul style="list-style-type: none">하나의 학습 데이터마다 오차를 계산해서 신경망의 가중치를 바로 조절. 빠르다.단점 : 한 개의 학습 데이터마다 매번 가중치를 갱신함으로 신경망의 성능이 들쑥날쑥 변함.단점 : 최적의 학습률을 구하기 위해 일일이 튜닝하고 수렴조건을 조정해야 함
Mini-Batch Stochastic Gradient Descent	<ul style="list-style-type: none">훈련데이터에서 소량의 데이터를 적당히 선택해 학습 후 갱신 처리를 수행장점 : 위 두 가지 경사 하강법의 단점을 보완하고 장점을 취함.장점 : GPU 기반의 효율적인 병렬 컴퓨팅이 가능해 진다.



- 전체 학습 데이터를 한 번씩 모두 학습 시킨 횟수
- 학습 시킨 데이터도 재 학습 시키면 신경망이 개선



5. MNIST Example

IV. Training

MNIST에 사용될 2층 신경망을 클래스로 구현해 본다.

- 2층 신경망 클래스를 생성

```
import numpy as np
import Common.functions as cf
from Common.gradient import numerical_gradient

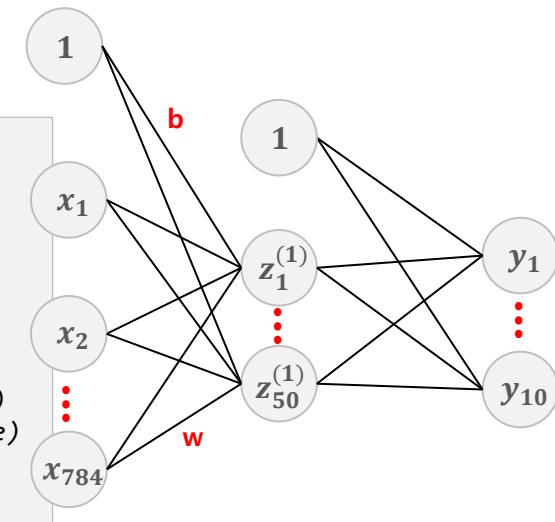
class MyTwoLayerNet:

    def __init__(self, inputSize, hiddenSize, outputSize, weightInitStd = 0.01):
        self.params = {}
        self.params['W1'] = weightInitStd * np.random.randn(inputSize, hiddenSize)
        self.params['W2'] = weightInitStd * np.random.randn(hiddenSize, outputSize)
        self.params['b1'] = np.zeros(hiddenSize)
        self.params['b2'] = np.zeros(outputSize)

    def predict(self, x):
        W1 = self.params['W1']
        W2 = self.params['W2']
        b1 = self.params['b1']
        b2 = self.params['b2']
        z = cf.sigmoid(np.dot(x, W1) + b1)
        y = cf.softmax(np.dot(z, W2) + b2)
        return y

    def loss(self, x, t):
        y = self.predict(x)
        return cf.cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
```



```
''' Gradient 구하기 (수치 미분을 이용) '''
def numericalGradient(self, x, t):
    lossW = lambda W : self.Loss(x, t)
    grads = {}
    grads['W1'] = numerical_gradient(lossW, self.params['W1'])
    grads['W2'] = numerical_gradient(lossW, self.params['W2'])
    grads['b1'] = numerical_gradient(lossW, self.params['b1'])
    grads['b2'] = numerical_gradient(lossW, self.params['b2'])
    return grads
```

5. MNIST Example

IV. Training

2층 신경망을 클래스를 이용해서 MNIST 판별을 위한 mini-batch 학습을 진행한다.

- Mini-batch 크기 : 100, 학습반복 : 1000회, 학습률 : 0.1로 학습을 진행해 본다.

```
import numpy as np
import matplotlib.pyplot as plt
from Common.mnist import load_mnist
from Training.mytwolayernet import MyTwoLayerNet

(trainImg, trainLbl), (testImg, testLbl) = load_mnist(one_hot_label=True)
network = MyTwoLayerNet(784, 50, 10)

# hyper parameters
itersNum = 1000 # 반복횟수
trainSize = trainImg.shape[0] # 60000
batchSize = 100 # mini-batch 크기
learningRate = 0.1 # 학습률

# 누적기록
trainLossList = []

print("-- Start Learning -- ")

for i in range(itersNum):
    # mini-batch 획득
    miniBatchMask = np.random.choice(trainSize, batchSize)
    trainImgBatch = trainImg[miniBatchMask]
    trainLblBatch = trainLbl[miniBatchMask]
    # Gradient 계산
    grad = network.numericalGradient(trainImgBatch, trainLblBatch)
    # 가중치, 편향 갱신
    for key in ('W1', 'W2', 'b1', 'b2'):
        network.params[key] -= learningRate*grad[key]
    # 비용함수(오차)의 변화 기록
    loss = network.loss(trainImgBatch, trainLblBatch)
    trainLossList.append(loss)
    print("iteration", i, ":", loss)

print("-- End Learning -- ")
```

#1 – Cost Function 이해하기

▪ mycostfunction.py

- Mean Squared Error와 Cross Entropy Error 비용함수를 구현해 본다.

```
import numpy as np

def meanSquaredError(y, t):
    return 0.5*np.sum((y-t)**2)

def crossEntropyError(y, t):
    delta = 1e-7 #아주 작은 값 (y가 0인 경우 -inf 값을 예방)
    return -np.sum(t*np.log(y+delta))

t = np.array([0, 0, 0, 0, 1, 0, 0, 0, 0]) # label = 5
y = np.array([0.1, 0.03, 0.05, 0.2, 0.9, 0.0, 0.1, 0.2, 0.12, 0.03]) # 5라고 추정
# 정답일 경우 MSE, CEE의 값은 적다.
print("-- 정답인 경우 --")
print("MSE :", meanSquaredError(y, t))
print("CEE :", crossEntropyError(y, t))

y = np.array([0.1, 0.03, 0.05, 0.2, 0.0, 0.1, 0.2, 0.12, 0.03, 0.9]) # 9라고 추정
# 오류일 경우 MSE, CEE의 값은 적다.
print("-- 오류인 경우 --")
print("MSE :", meanSquaredError(y, t))
print("CEE :", crossEntropyError(y, t))
```

- 실행 결과 확인

```
-- 정답인 경우 --
MSE : 0.06435
CEE : 0.105360404547
-- 오류인 경우 --
MSE : 0.96435
CEE : 16.118095651
```

#2 – Cost Function을 MNIST에 적용하기 (미니배치를 이용해 MNIST 신경망 출력결과를 확인하고 비용함수 결과를 계산)

▪ mymnist.py

- Cross Entropy Error 비용함수를 구현하여 MNIST에 적용해 본다.
- 실행 결과 확인

... 코드 생략 ...

```
def crossEntropyError(y, t):
    # 1차원 배열인 경우 2차원 배열로 변경
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    miniBatchSize = y.shape[0]
    ...

    y[np.arange(miniBatchSize), t] 은 정답 레이블에 해당되는 신경망의 출력을 추출
    예) 만약 배치사이즈가 10개라면 아래와 같이 10개의 신경망 추출에 대한 출력값을 추출
    y[0,2], y[1,7], y[2,0], y[3,9],...y[9,5]
    ...

    return -np.sum(t*np.log(y[np.arange(miniBatchSize), t])) / miniBatchSize
```

... 코드 생략 ...

```
miniBatchSize = 10
print(len(trainImg), "개 중에 ", miniBatchSize, "를 미니배치 학습한다.")
miniBatchMask = np.random.choice(len(trainImg), miniBatchSize)
trainImgBatch = trainImg[miniBatchMask]
trainLblBatch = trainLbl[miniBatchMask]
print("Random Choice 된 정답 레이블", trainLblBatch)

y = predict(network, trainImgBatch)
print("신경망 출력 결과 (10-배치크기, 10-분류크기) Shape\n", y)
p = np.argmax(y, axis=1)
print("최대값으로 추정한 결과", p)

print(crossEntropyError(y, trainLblBatch))
```

60000 개 중에 10 를 미니배치 학습.

Random Choice 된 정답 레이블 [4 5 1 8 1 2 2 7 7 4]
 신경망 출력 결과 (10-배치크기, 10-분류크기) Shape
 [[1.41822547e-05 2.35181437e-06
 1.51875982e-04 3.35034133e-06

... 결과 생략 ...

최대값으로 추정한 결과 [4 5 1 8 1 2 2 7 7 4]
 0.12456381321

#3 – 미분이해하기 (수치미분, 편미분)

▪ mydifferentiation.py

- 수치미분과 편미분을 구현해 본다.

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

def numericalDiff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)

def sampleFunc1(x):
    return 0.01*x**2 + 0.1*x

...
f(x) = x[0]**2 + x[1]**2
x[0] = 3, x[1] = 4 일 때 편미분 구하기 위한 함수
...

def sampleFunc2(x):
    return x**2 + 4**2
def sampleFunc3(x):
    return 3**2 + x**2

x = np.arange(0, 20, 0.1)
y = sampleFunc1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y)
plt.show()

print(numericalDiff(sampleFunc1, 5))
print(numericalDiff(sampleFunc1, 10))

print(numericalDiff(sampleFunc2, 3.0))
print(numericalDiff(sampleFunc3, 4.0))
```

- 실행 결과 확인

```
0.1999999999990898
0.2999999999986347
6.000000000000378
7.999999999999119
```

#4 – Gradient (기울기) 계산하기

▪ mydifferentiation.py

- $f(x_0, x_1) = x_0^2 + x_1^2$ 함수의 기울기를 계산을 구현하고 테스트한다.

... 코드 생략 ...

```
def numeiralGradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) #x와 형상이 같은 배열을 생성
    for idx in range(x.size):
        tmpVal = x[idx]
        x[idx] = tmpVal + h
        print(idx, x)
        fxh1 = f(x)

        x[idx] = tmpVal - h
        print(idx, x)
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmpVal # 원래 값 복원

    return grad

...
f(x0, x1) = x0**2 + x1**2 인 함수
...

def sampleFunc4(x):
    return x[0]**2 + x[1]**2

... 코드 생략 ...

print(numeiralGradient(sampleFunc4, np.array([3.0, 4.0])))
print(numeiralGradient(sampleFunc4, np.array([0.0, 2.0])))
print(numeiralGradient(sampleFunc4, np.array([3.0, 0.0])))
```

• 실행 결과 확인

```
0 [ 3.0001  4.      ]
0 [ 2.9999  4.      ]
1 [ 3.       4.0001]
1 [ 3.       3.9999]
[ 6.   8.]
0 [ 1.00000000e-04  2.00000000e+00]
0 [ -1.00000000e-04  2.00000000e+00]
1 [ 0.       2.0001]
1 [ 0.       1.9999]
[ 0.   4.]
0 [ 3.0001  0.      ]
0 [ 2.9999  0.      ]
1 [ 3.00000000e+00  1.00000000e-04]
1 [ 3.00000000e+00 -1.00000000e-04]
[ 6.   0.]
```

6. Hands-on

IV. Training

#5 – Gradient (기울기)와 학습률을 이용하여 반복 학습하면서 최솟값 찾아가기

- mylearningrate.py

- $f(x_0, x_1) = x_0^2 + x_1^2$ 함수의 기울기를 계산 후 기울기 값과 학습률을 이용해서 x 값을 조정한다.

```
import numpy as np
import matplotlib.pyplot as plt
from Common.gradient_2d import numerical_gradient

'''
f = x[0]**2 + x[1]**2
initX = 초기값 (-3, 4)
초기값에서 함수에 대한 기울기 (편미분에 의한 계산)를 구한 후
기울기 값과 학습률을 이용해서 x값을 이동
위를 stepNum(100번) 반복
'''

def gradientDescent(f, initX, learningRate, stepNum=100):
    x = initX
    xHistory = []

    for i in range(stepNum):
        xHistory.append(x.copy())
        grad = numerical_gradient(f,x)
        x -= learningRate*grad
        if(i == stepNum-1):
            print(x)

    return x, np.array(xHistory)

def sampleFunction(x):
    return x[0]**2 + x[1]**2
```

```
# 적정학습률
initX = np.array([-3.0,4.0])
learningRate = 0.1
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)

# 학습률이 큰 경우
initX = np.array([-3.0,4.0])
learningRate = 2
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)

# 학습률이 작은 경우
initX = np.array([-3.0,4.0])
learningRate = 0.001
x, xHistory = gradientDescent(sampleFunction, initX, learningRate)

# 그래프로 표현
plt.plot([-5,5],[0,0], '--b')
plt.plot([0,0],[-5,5], '--g')
plt.plot(xHistory[:,0],xHistory[:,1], 'o')
plt.xlim(-3.5, 3.5)
plt.ylim(-4.5, 4.5)
plt.xlabel("X0")
plt.ylabel("X1")
plt.show()
```

- 실행 결과 확인 (100회 반복 후 x 값의 조정된 모습)

```
[ -6.11110793e-10  8.14814391e-10]
[ -1.35262250e+12  2.26052540e+12]
[ -2.45570041  3.27426722]
```

#6 – 신경망에서 Gradient (기울기) 계산하기

▪ mysimplenet.py

- 입력층 2개, 출력층 3개, 은닉층이 없는 신경망에서 임의의 가중치로 오차와 기울기를 구현한다.

```
import numpy as np
from Common.functions import softmax, cross_entropy_error
from Common.gradient import numerical_gradient

class SimpleNet:

    def __init__(self):
        self.W = np.random.randn(2,3) # 정규분포로 가중치 초기화

    def predict(self, x):
        z = np.dot(x, self.W)
        y = softmax(z)
        return y

    def loss(self, x, t):
        y = self.predict(x)
        loss = cross_entropy_error(y, t)
        return loss

simpleNet = SimpleNet()
x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])
print("input values:", x)
print("initialized weight:\n", simpleNet.W)
print("labeled values:", t)
print("neuralnet output:", simpleNet.predict(x))
print("cost(loss) :", simpleNet.loss(x, t))
```

```
def f(w): #w는 dummy
    return simpleNet.loss(x, t)
...
numerical_gradient(f, x)에서
f는 손실함수, x는 손실함수 f의 인수
즉, simpleNet.W[0,0], [0,1], [0,2], [1,0], [1,1], [1,2]
의 값으로 손실함수를 편미분한 결과 (Gradient)
...
gradient = numerical_gradient(f, simpleNet.W)
print("gradient:\n", gradient)
```

- 실행 결과 확인

```
input values: [ 0.6  0.9]
initialized weight:
[[-0.06407468 -0.71326724 -1.33225821]
 [ 1.57714681 -0.91908203 -0.69465475]]
labeled values: [0 0 1]
neuralnet output: [ 0.88330389  0.06327864
 0.05341747]
cost(loss) : 2.92961749969
gradient:
[[ 0.52998234  0.03796718 -0.56794952]
 [ 0.79497351  0.05695077 -0.85192428]]
```

#7 – MNIST를 위한 2층 신경망 만들기

▪ mytwolayernet.py

```

import numpy as np
import Common.functions as cf
from Common.gradient import numerical_gradient

class MyTwoLayerNet:

    def __init__(self, inputSize, hiddenSize, outputSize, weightInitStd = 0.01):
        self.params = {}
        self.params['W1'] = weightInitStd * np.random.randn(inputSize, hiddenSize)
        self.params['W2'] = weightInitStd * np.random.randn(hiddenSize, outputSize)
        self.params['b1'] = np.zeros(hiddenSize)
        self.params['b2'] = np.zeros(outputSize)

    def predict(self, x):
        W1 = self.params['W1']
        W2 = self.params['W2']
        b1 = self.params['b1']
        b2 = self.params['b2']
        z = cf.sigmoid(np.dot(x, W1) + b1)
        y = cf.softmax(np.dot(z, W2) + b2)
        return y

    def loss(self, x, t):
        y = self.predict(x)
        return cf.cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

```

''' Gradient 구하기 (수치 미분을 이용) '''

```

def numericalGradient(self, x, t):
    lossW = lambda W : self.loss(x, t)
    grads = {}
    grads['W1'] = numerical_gradient(lossW, self.params['W1'])
    grads['W2'] = numerical_gradient(lossW, self.params['W2'])
    grads['b1'] = numerical_gradient(lossW, self.params['b1'])
    grads['b2'] = numerical_gradient(lossW, self.params['b2'])
    return grads

```

#8 – MNIST 신경망 학습하기

▪ mymnistlearning.py

```

import numpy as np
import matplotlib.pyplot as plt
from Common.mnist import load_mnist
from Training.mytwolayernet import MyTwoLayerNet

(trainImg, trainLbl), (testImg, testLbl) = load_mnist(one_hot_label=True)
network = MyTwoLayerNet(784, 50, 10)

# hyper parameters
itersNum = 1000 # 반복횟수
trainSize = trainImg.shape[0] # 60000
batchSize = 100 # mini-bach 크기
learningRate = 0.1 # 학습률

# 누적기록
trainLossList = []

print("-- Start Learning --")

for i in range(itersNum):
    # mini-batch 획득
    miniBatchMask = np.random.choice(trainSize, batchSize)
    trainImgBatch = trainImg[miniBatchMask]
    trainLblBatch = trainLbl[miniBatchMask]
    # Gradient 계산
    grad = network.numericalGradient(trainImgBatch, trainLblBatch)
    # 가중치, 편향 갱신
    for key in ('W1', 'W2', 'b1', 'b2'):
        network.params[key] -= learningRate*grad[key]
    # 비용함수(오차)의 변화 기록
    loss = network.loss(trainImgBatch, trainLblBatch)
    trainLossList.append(loss)
    print("iteration", i, ":", loss)

print("-- End Learning --")

```

```

# 그래프 그리기
x = np.arange(len(trainLossList))
plt.plot(x, trainLossList, label="Loss")
plt.xlabel("iteration")
plt.ylabel("Loss")
plt.show()

```

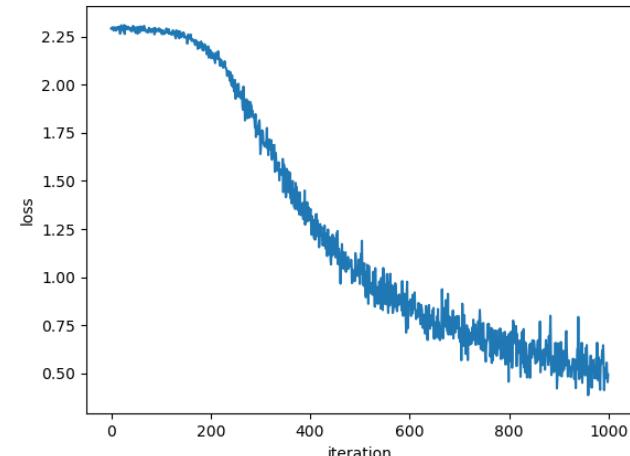
"신경망 학습을 위해 기울기를 수치미분을 통해서 구하면 계산 시간이 오래 걸린다."

• 실행 결과 확인

```

-- Start Learning --
iteration 0 : 2.29192863536
iteration 1 : 2.29489284268
iteration 2 : 2.28903658939
... 결과 생략 ...
iteration 997 : 0.489302945227
iteration 998 : 0.454912045635
iteration 999 : 0.494802915552
-- End Learning --

```



V. Back

Propagation

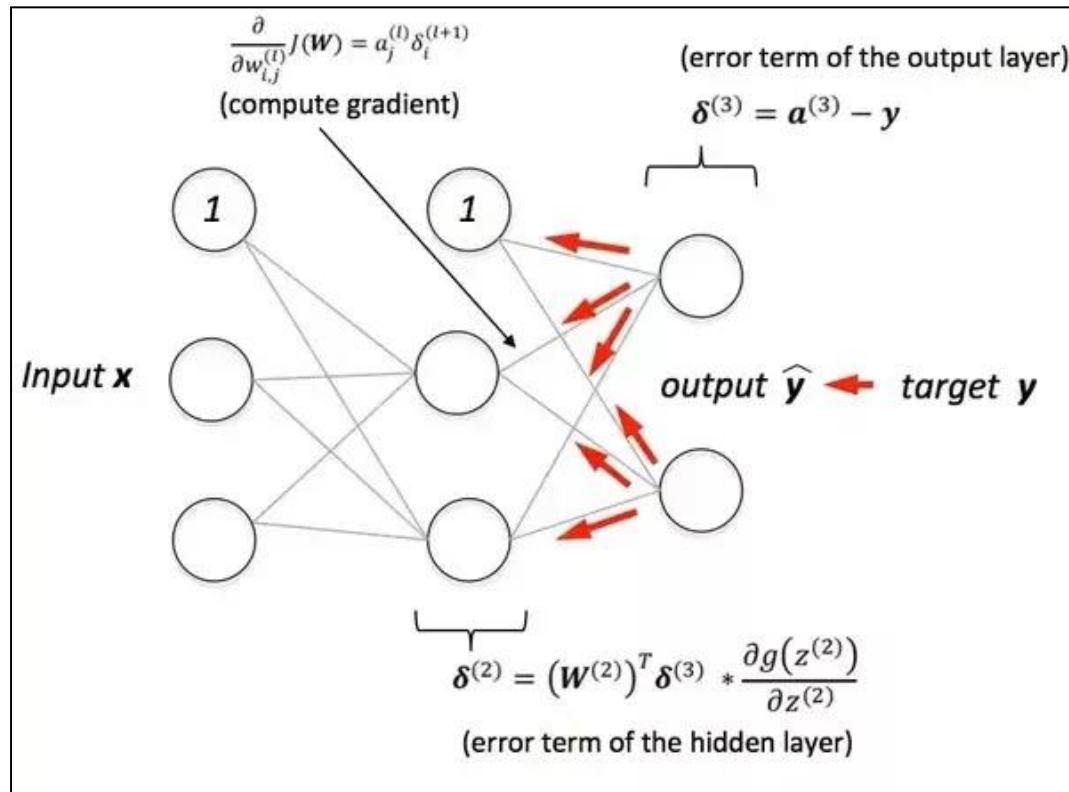


1. Back Propagation Algorithm
2. Computational Graph
3. 계층별 구현
4. MNIST Examples
5. 예전파의 문제
- b. Hands-on

예측된 결과와 실제값의 차이인 에러의 역전파를 통해 가중치를 구한다. (Backward propagation of errors)

- 역 전파 알고리즘 (Back propagation algorithm)

- Deep Learning의 대표적인 지도학습 알고리즘.
- 역 전파 : 역방향으로 오차를 전파
- Back propagation 알고리즘은 Gradient descent rule과 결합되어 인공신경망 학습 알고리즘으로 많이 이용된다.
- 학습된 출력 값과 실제 값과의 차이인 오차를 계산하여 Feedforward 반대인 역방향으로 전파(Propagation) 한다.
- 전파된 오차를 이용하여 가중치를 조정한다.



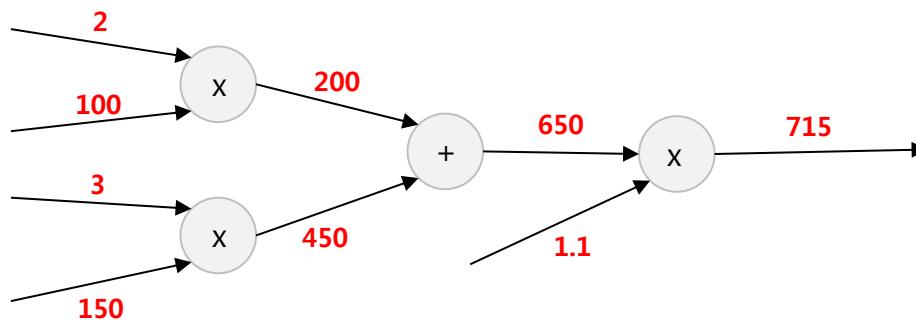
- 출력층의 오차 : 지도학습 데이터의 정답과 출력층의 출력값의 차이
- 은닉층의 오차 : 출력층의 오차가 가중치를 고려하여 역방향으로 전달
- 델타(δ)규칙 : 어떤 입력 노드가 출력 노드의 오차에 기여했다면, 두 노드의 연결 가중치는 오차에 비례해 조절한다.
- 델타(δ) : 델타는 활성함수의 도함수 (derivatives)를 이용해 계산됨.
즉, 델타를 이용해 가중치를 조절(compute gradient)하고 앞 은닉 노드들의 델타를 계산함.
- 비용함수(cost function) : 오차를 계산하는 함수로 Cross Entropy 함수가 많이 사용

→ 대부분의 딥러닝 오픈 프레임워크에는 역 전파 알고리즘이 구현되어 있음.

Computational Graph (계산 그래프)란 계산 과정을 그래프로 나타낸 것이다.

▪ Computational Graph

사과 2개, 귤 3개를 사는데 사과는 1개 100원, 귤은 150원이고 소비세는 10%일 때 지불 금액은 얼마인가?



- Computational Graph에서 계산을 왼쪽에서 오른쪽으로 진행하는 것을 **순전파 (forward propagation)**이라고 한다.
- 계산 그래프에서 종착점에서 출발점으로 계산하는 것을 **역전파(backward propagation)**이라고 한다.

▪ Computational Graph 의 장점

- 국소적 계산 : 전체에서 어떻게 진행되는 상관없이 자신과 관계된 정보만으로 계산
(예, 가운데 + Node의 경우 여러 식품을 구매하더라도 해당 노드는 더하는 계산만 하면 됨.)
- 중간 계산 결과를 모두 보관할 수 있다.
- 역전파를 통해 '미분'을 효율적으로 계산할 수 있다.
✓ 사과 가격, 귤 가격, 사과 개수, 귤 개수, 소비세에 대한 지불 금액의 미분을 계산할 수 있다.

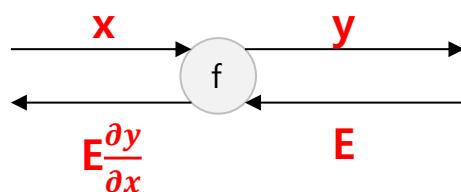
2. Computational Graph

V. Back Propagation

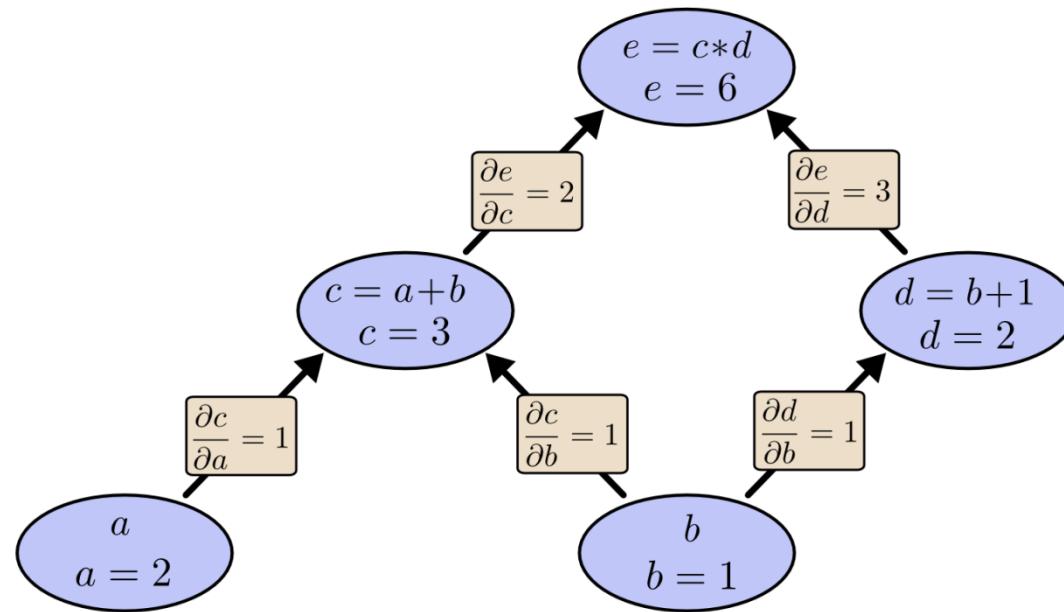
Computational Graph의 역전파 계산시는 미분을 계산하여 앞쪽 노드로 전파한다.

- Backward Propagation의 계산 절차

예) $y = f(x)$ 의 역전파



- 역전파의 계산 절차는 신호 E에 노드의 편미분 $\frac{\partial y}{\partial x}$ 을 곱한 후 다음 노드로 전달하는 것
- $y = x^2$ 인 경우, $\frac{\partial y}{\partial x} = 2x$ 가 된다.



2. Computational Graph

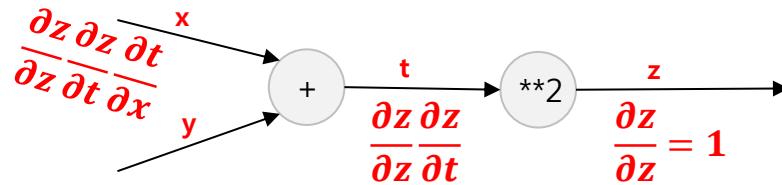
V. Back Propagation

Chain Rule(연쇄법칙)이란 합성 함수의 미분의 성질이며, 합성 함수의 미분은 합성함수를 구성하는 각 함수의 미분의 곱이다.

- 합성 함수의 예 ($z = (x + y)^2$)

순전파 계산

- $z = t^2$
- $t = x + y$



역전파 계산

- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial z} \frac{\partial z}{\partial t} \frac{\partial t}{\partial x}$
- $\frac{\partial z}{\partial t} = 2t$ 이고 $\frac{\partial z}{\partial z} = 1$ 임으로
$$\frac{\partial z}{\partial x} = 2t * 1 = 2(x + y)$$
 가 된다.

- $\frac{\partial z}{\partial x} \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = \frac{\partial z}{\partial x}$ 즉, x에 대한 z의 미분, 즉 역전파는 연쇄법칙의 원리와 같다.

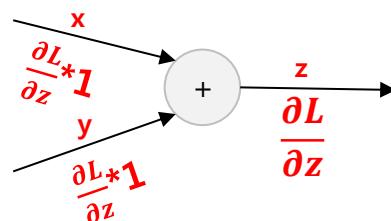
2. Computational Graph

V. Back Propagation

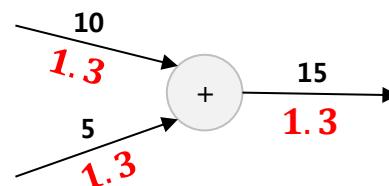
덧셈노드와 곱셈노드의 역전파를 예를 통해서 확인해 본다.

▪ 덧셈 노드 ($z = x + y$)

- $\frac{\partial z}{\partial x} = 1$ 이고 $\frac{\partial z}{\partial y} = 1$ 이다



$\frac{\partial L}{\partial z}$ 은 상류에서 전해진 미분 값

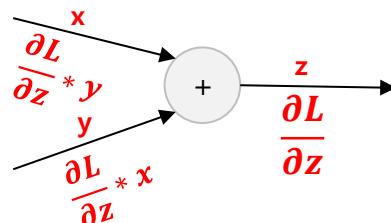


역전파 계산

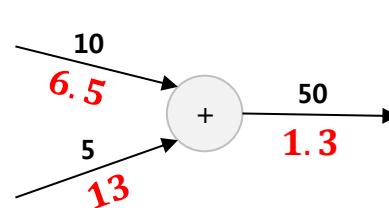
- 만약 상류에서 1.3이 흘러오면 다음 노드로 1.3을 전달한다.

▪ 곱셈 노드 ($z = xy$)

- $\frac{\partial z}{\partial x} = y$ 이고 $\frac{\partial z}{\partial y} = x$ 이다



$\frac{\partial L}{\partial z}$ 은 상류에서 전해진 미분 값



역전파 계산

- 만약 상류에서 1.3이 흘러오면 한 군데는 6.5 (1.3×5) 다른 하나는 13 (1.3×10)을 보낸다.
- 곱셈 역 전파 계산 시 순방향 입력 신호 값이 필요함.

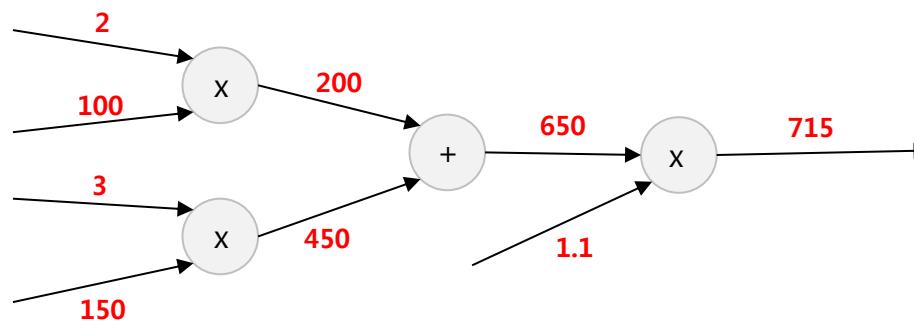
2. Computational Graph

V. Back Propagation

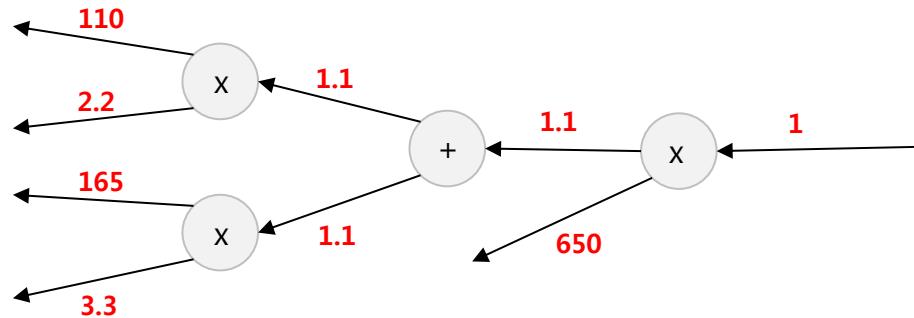
사과 쿨 쇼핑의 역전파를 계산해 보자

- 순전파

사과 2개, 쿨 3개를 사는데 사과는 1개 100원, 쿨은 150원이고 소비세는 10%일 때 지불 금액은 얼마인가?



- 역전파



2. Computational Graph

V. Back Propagation

사과 쿠 쇼핑의 계산 그래프와 순전파/역전파를 파이썬으로 구현해보자

▪ 덧셈 노드와 곱셈 노드의 구현

```
class AddLayer:  
  
    def forward(self, x, y):  
        return x+y  
  
    def backward(self, difOut):  
        dx = difOut  
        dy = difOut  
        return dx, dy  
  
class MulLayer:  
  
    # 순전파시 입력값 유지  
    def forward(self, x, y):  
        self.x = x  
        self.y = y  
        return x*y  
  
    def backward(self, difOut):  
        dx = difOut*self.y  
        dy = difOut*self.x  
        return dx, dy
```

▪ 순전파/역전파 활용

```
applePrice = 100  
orangePrice = 150  
appleCnt = 2  
orangeCnt = 3  
tax = 1.1  
  
calAppleLayer = Mullayer()  
calOrangeLayer = Mullayer()  
calSumLayer = Addlayer()  
calTaxLayer = Mullayer()  
  
# forward Propagation  
calApple = calAppleLayer.forward(applePrice, appleCnt)  
calOrange = calOrangeLayer.forward(orangePrice, orangeCnt)  
calSum = calSumLayer.forward(calApple, calOrange)  
calTax = calTaxLayer.forward(calSum, tax)  
print(calApple, calOrange, calSum, calTax)  
  
# backward Propagation  
difOut = 1  
difCalSum, difCalTax = calTaxLayer.backward(difOut)  
difCalApple, difCalOrange = calSumLayer.backward(difCalSum)  
difApplePrice, difAppleCnt = calAppleLayer.backward(difCalApple)  
difOrangePrice, difOrangeCnt = calOrangeLayer.backward(difCalOrange)  
print(difOut, difCalSum, difCalTax, difCalApple, difCalOrange)  
print(difApplePrice, difAppleCnt, difOrangePrice, difOrangeCnt)
```

• 실행 결과 확인

```
200 450 650 715.0000000000001  
1 1.1 650 1.1 1.1  
2.2 110.0000000000001 3.300000000000003 165.0
```

ReLU 계층을 Computational Graph를 이용해 구현해 보자. (backward 계산과정은 생략)

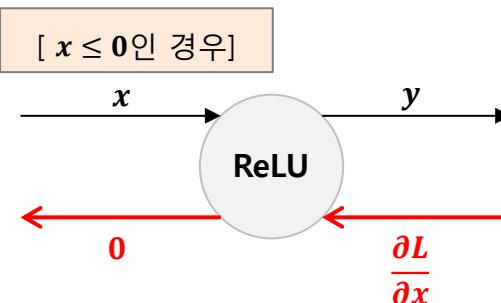
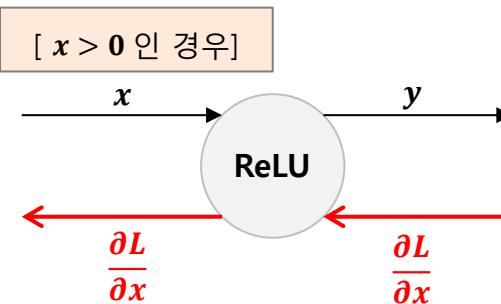
▪ ReLU (활성화 함수)

- forward propagation

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

- backward propagation

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$



▪ ReLU 구현 코드

```

class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
  
```

- Test

```

ReLU : input
[[ 1.1 -0.2]
 [-1.5  2.7]]
ReLU : forward
[[ 1.1  0. ]
 [ 0.   2.7]]
ReLU : dif. out
[[ 1.   1.]
 [ 1.   1.]]
ReLU : backward
[[ 1.   0.]
 [ 0.   1.]]
  
```

Sigmoid 계층을 Computational Graph를 이용해 구현해 보자. (backward 계산과정은 생략)

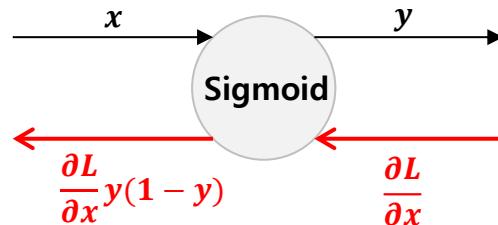
▪ ReLU (활성화 함수)

- forward propagation

$$y = \frac{1}{1 + e^{-x}}$$

- backward propagation

$$\frac{\partial y}{\partial x} = y(1 - y)$$



▪ Sigmoid 구현 코드

```

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
  
```

- Test

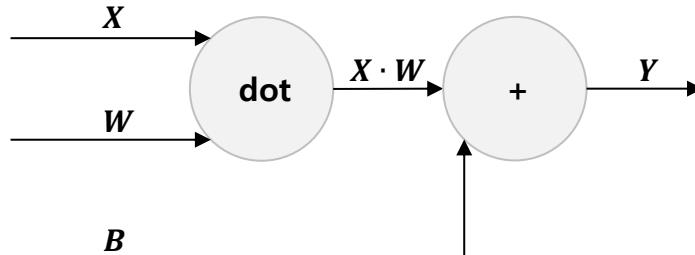
```

Sigmoid : input
[[ 1.1 -0.2]
 [-1.5  2.7]]
Sigmoid : forward
[[ 0.75026011  0.450166  ]
 [ 0.18242552  0.93702664]]
Sigmoid : dif. out
[[ 1.  1.]
 [ 1.  1.]]
Sigmoid : backward
[[ 0.18736988  0.24751657]
 [ 0.14914645  0.05900771]]
  
```

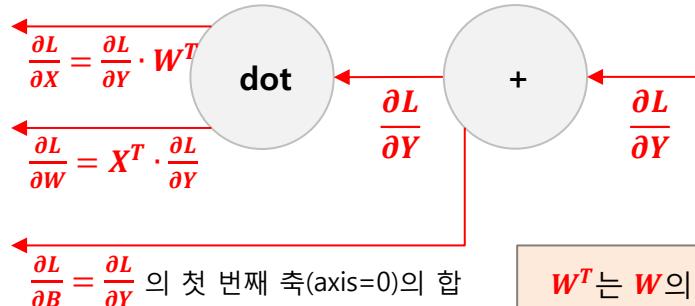
Affine 계층을 Computational Graph를 이용해 구현해 보자. (backward 계산과정은 생략)

- 신경망 순전파시 수행하는 행렬의 내적을 **Affine 변환**이라고 한다.

- forward propagation



- backward propagation



W^T 는 W 의 전치 행렬이다.

$$W = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \text{이면}$$

$$W^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \text{이다}$$

- Affine 구현 코드

```

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 대용
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

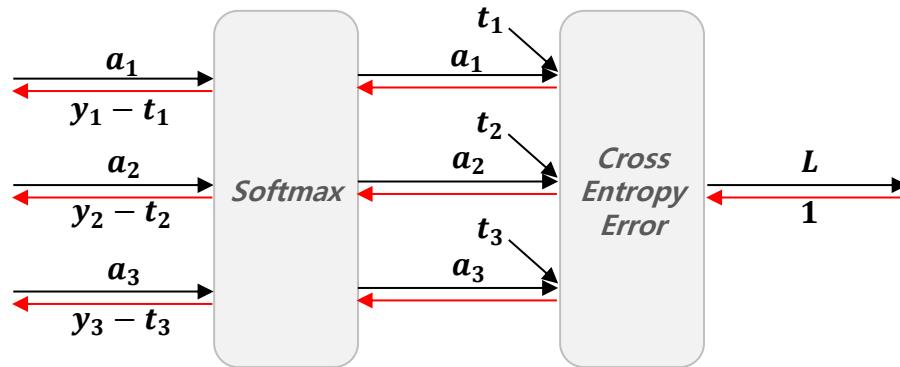
    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)
        return dx
  
```

Softmax-with-Loss 계층을 Computational Graph를 이용해 구현해 보자. (backward 계산과정은 생략)

▪ Softmax 함수와 Cross Entropy 함수를 같이 구현

- 간소화된 Softmax-with-Loss 의 계산 그래프



3개 분류로 가정

a_1, a_2, a_3 는 Softmax 계층의 입력

y_1, y_2, y_3 는 Softmax 계층의 출력 (정규화, 합계가 1)

t_1, t_2, t_3 는 정답 레이블

L 은 비용함수 값 (손실)

$y_1 - t_1, y_2 - t_2, y_3 - t_3$ 는 역전파 값 (즉, 오차가 역전파 된다.)

▪ Softmax-with-Loss 구현 코드

```

class SoftmaxWithLoss:
    def __init__(self):
        self.Loss = None # 손실함수
        self.y = None # softmax의 출력
        self.t = None # 정답 레이블(원-핫 인코딩 형태)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size:
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
  
```



출력층 고려사항

- Softmax 계층은 학습 시에는 활용하지만 추론 시에는 활용하지 않는다.
- 회귀 목적의 신경망에서는 항등함수의 손실함수로 평균제곱오차를 활용한다.

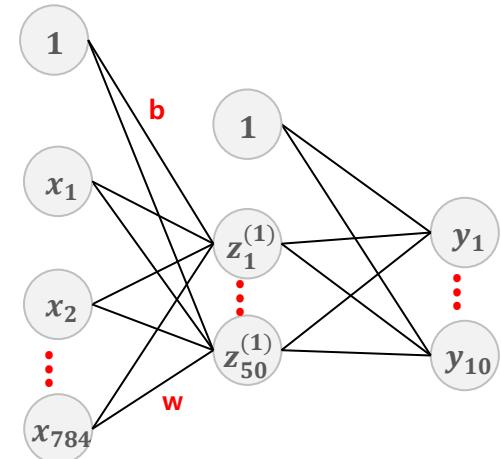
4. MNIST Example

V. Back Propagation

MNIST에 사용될 2층 신경망을 클래스를 변경해 본다.

- 2층 신경망 클래스를 생성

```
class MyTwoLayerNet:  
  
    def __init__(self, inputSize, hiddenSize, outputSize, weightInitStd = 0.01):  
        self.params = {}  
        self.params['W1'] = weightInitStd * np.random.randn(inputSize, hiddenSize)  
        self.params['W2'] = weightInitStd * np.random.randn(hiddenSize, outputSize)  
        self.params['b1'] = np.zeros(hiddenSize)  
        self.params['b2'] = np.zeros(outputSize)  
        # 계층생성  
        self.layers = OrderedDict()  
        self.layers['Affine1'] = cl.Affine(self.params['W1'], self.params['b1'])  
        self.layers['Relu1'] = cl.Relu()  
        self.layers['Affine2'] = cl.Affine(self.params['W2'], self.params['b2'])  
        self.lastLayer = cl.SoftmaxWithLoss()  
  
    def predict(self, x):  
        for layer in self.layers.values():  
            x = layer.forward(x)  
        return x  
  
    def loss(self, x, t):  
        y = self.predict(x)  
        return self.lastLayer.forward(y, t)  
  
    def accuracy(self, x, t):  
        y = self.predict(x)  
        y = np.argmax(y, axis=1)  
        t = np.argmax(t, axis=1)  
        accuracy = np.sum(y == t) / float(x.shape[0])  
        return accuracy
```



```
''' Gradient 구하기 (역전파를 이용) '''  
def gradient(self, x, t):  
    # 순전파  
    self.loss(x,t)  
    # 이후 역전파  
    dout = 1  
    dout = self.lastLayer.backward(dout)  
  
    layers = list(self.layers.values())  
    layers.reverse()  
    for layer in layers:  
        dout = layer.backward(dout)  
  
    grads = {}  
    grads['W1'] = self.layers['Affine1'].dw  
    grads['b1'] = self.layers['Affine1'].db  
    grads['W2'] = self.layers['Affine2'].dw  
    grads['b2'] = self.layers['Affine2'].db  
  
    return grads
```

2층 신경망을 클래스를 이용해서 MNIST 판별을 위한 mini-batch 학습을 진행한다.

- Mini-batch 크기 : 100, 학습반복 : 1000회, 학습률 : 0.1로 학습을 진행해 본다.

```
(trainImg, trainLbl), (testImg, testLbl) = load_mnist(one_hot_label=True)
network = MyTwoLayerNet(784, 50, 10)

# hyper parameters
itersNum = 1000 # 반복횟수
trainSize = trainImg.shape[0] # 60000
batchSize = 100 # mini-batch 크기
learningRate = 0.1 # 학습률

# 누적기록
trainLossList = []

print("-- Start Learning -- ")

for i in range(itersNum):
    # mini-batch 획득
    miniBatchMask = np.random.choice(trainSize, batchSize)
    trainImgBatch = trainImg[miniBatchMask]
    trainLblBatch = trainLbl[miniBatchMask]

    # Gradient 계산
    grad = network.gradient(trainImgBatch, trainLblBatch)

    # 가중치, 편향 갱신
    for key in ('W1', 'W2', 'b1', 'b2'):
        network.params[key] -= learningRate*grad[key]

    # 비용함수(오차)의 변화 기록
    loss = network.loss(trainImgBatch, trainLblBatch)
    trainLossList.append(loss)

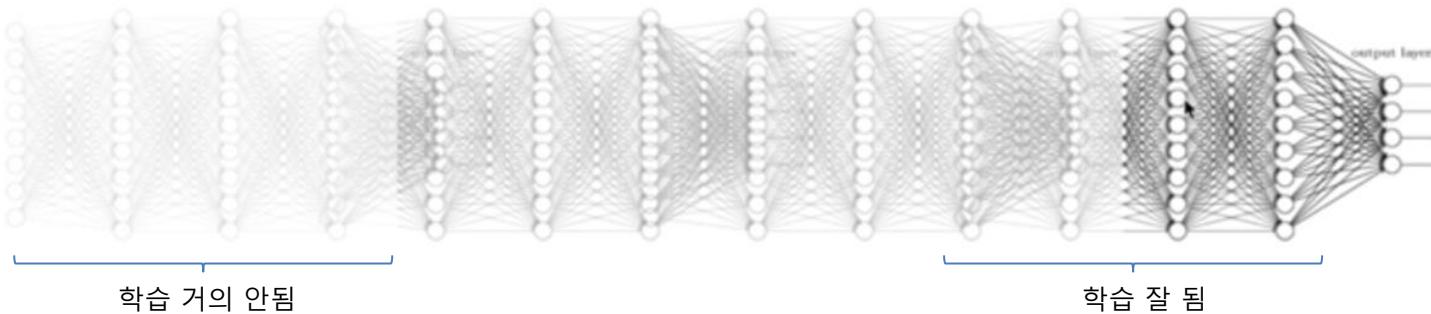
    print("iteration", i, ":", loss)

print("-- End Learning -- ")
```

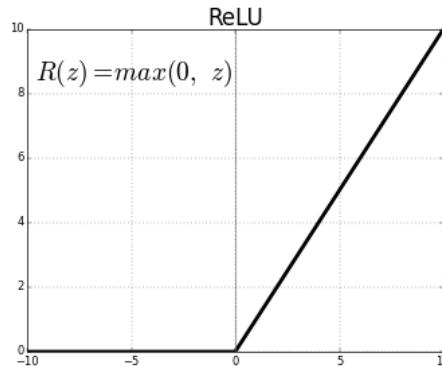
신경망의 계층을 깊게 만들면 역 전파 알고리즘으로 심층 신경망을 학습 시킬 때 어려움을 겪는다.

▪ Vanishing Gradient (기울기 소멸) 문제

- 역 전파 알고리즘으로 학습시킬 때 앞쪽의 은닉층으로 오차가 거의 전달 안됨 (즉, 학습이 잘 안됨)



- 해결 방안: 은닉층 활성화 함수로 ReLU(Rectified Linear Unit Function) 함수를 사용



- ✓ 간단한 구현 : 입력이 음수이면 0을 출력하고 양수이면 입력 값 그대로 출력
참고로 Sigmoid 함수는 입력 값이 아무리 커도 출력은 1을 넘지 못하였음.
- ✓ 학습 속도가 Sigmoid 보다 빠르고 Vanishing Gradient 문제가 적어 학습이 잘 됨.

▪ Over-fitting (과적합) 문제

- 신경망이 깊어지면 복잡한 모델이 되어 과적합에 취약
- Dropout / Batch Normalization 기법으로 해결

▪ 많은 계산량 문제

- 신경망이 깊어지면 가중치가 기하급수적으로 증가되고, 학습 데이터도 많이 필요하고, 계산량이 급증함.
- GPU의 활용과 배치 정규화(Batch Normalization, 2015년) 등의 알고리즘으로 개선

#1 – Computational Graph 이해하기

▪ mycomputational.py

```

class AddLayer:

    def forward(self, x, y):
        return x+y

    def backward(self, difOut):
        dx = difOut
        dy = difOut
        return dx, dy

class MulLayer:

    # 순전파시 입력값 유지
    def forward(self, x, y):
        self.x = x
        self.y = y
        return x*y

    def backward(self, difOut):
        dx = difOut*self.y
        dy = difOut*self.x
        return dx, dy

```

```

applePrice = 100
orangePrice = 150
appleCnt = 2
orangeCnt = 3
tax = 1.1

calAppleLayer = Mullayer()
calOrangeLayer = Mullayer()
calSumLayer = Addlayer()
calTaxLayer = Mullayer()

# forward Propagation
calApple = calAppleLayer.forward(applePrice, appleCnt)
calOrange = calOrangeLayer.forward(orangePrice, orangeCnt)
calSum = calSumLayer.forward(calApple, calOrange)
calTax = calTaxLayer.forward(calSum, tax)
print(calApple, calOrange, calSum, calTax)

# backward Propagation
difOut = 1
difCalSum, difCalTax = calTaxLayer.backward(difOut)
difCalApple, difCalOrange = calSumLayer.backward(difCalSum)
difApplePrice, difAppleCnt = calAppleLayer.backward(difCalApple)
difOrangePrice, difOrangeCnt = calOrangeLayer.backward(difCalOrange)
print(difOut, difCalSum, difCalTax, difCalApple, difCalOrange)
print(difApplePrice, difAppleCnt, difOrangePrice, difOrangeCnt)

```

• 실행 결과 확인

```

200 450 650 715.0000000000001
1 1.1 650 1.1 1.1
2.2 110.0000000000001 3.300000000000003 165.0

```

#2 – MNIST를 위한 2층 신경망 만들기 ▪ mytwolayernet.py

```

import numpy as np
import Common.layers as cl
from collections import OrderedDict

class MyTwoLayerNet:

    def __init__(self, inputSize, hiddenSize, outputSize, weightInitStd = 0.01):
        self.params = {}
        self.params['W1'] = weightInitStd * np.random.randn(inputSize, hiddenSize)
        self.params['W2'] = weightInitStd * np.random.randn(hiddenSize, outputSize)
        self.params['b1'] = np.zeros(hiddenSize)
        self.params['b2'] = np.zeros(outputSize)
        # 계층생성
        self.Layers = OrderedDict()
        self.Layers['Affine1'] = cl.Affine(self.params['W1'], self.params['b1'])
        self.Layers['Relu1'] = cl.Relu()
        self.Layers['Affine2'] = cl.Affine(self.params['W2'], self.params['b2'])
        self.lastLayer = cl.SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.Layers.values():
            x = layer.forward(x)
        return x

    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

```

''' Gradient 구하기 (역전파를 이용) '''

```

def gradient(self, x, t):
    # 순전파
    self.loss(x, t)
    # 이후 역전파
    dout = 1
    dout = self.lastLayer.backward(dout)

    layers = list(self.Layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    grads = {}
    grads['W1'] = self.layers['Affine1'].dw
    grads['b1'] = self.layers['Affine1'].db
    grads['W2'] = self.layers['Affine2'].dw
    grads['b2'] = self.layers['Affine2'].db

    return grads

```

#3 – MNIST 신경망 학습하기

▪ mymnistlearning.py

```

import numpy as np
import matplotlib.pyplot as plt
from Common.mnist import load_mnist
from Propagation.mytwolayernet import MyTwoLayerNet

(trainImg, trainLbl), (testImg, testLbl) = load_mnist(one_hot_label=True)
network = MyTwoLayerNet(784, 50, 10)

# hyper parameters
itersNum = 1000 # 반복횟수
trainSize = trainImg.shape[0] # 60000
batchSize = 100 # mini-bach 크기
learningRate = 0.1 # 학습률

# 누적기록
trainLossList = []

print("-- Start Learning --")

for i in range(itersNum):
    # mini-batch 획득
    miniBatchMask = np.random.choice(trainSize, batchSize)
    trainImgBatch = trainImg[miniBatchMask]
    trainLblBatch = trainLbl[miniBatchMask]
    # Gradient 계산
    grad = network.gradient(trainImgBatch, trainLblBatch)
    # 가중치, 편향 갱신
    for key in ('W1', 'W2', 'b1', 'b2'):
        network.params[key] -= learningRate*grad[key]
    # 비용함수(오차)의 변화 기록
    loss = network.loss(trainImgBatch, trainLblBatch)
    trainLossList.append(loss)
    print("iteration", i, ":", loss)

print("-- End Learning --")

```

```

# 그래프 그리기
x = np.arange(len(trainLossList))
plt.plot(x, trainLossList, label="Loss")
plt.xlabel("iteration")
plt.ylabel("Loss")
plt.show()

```

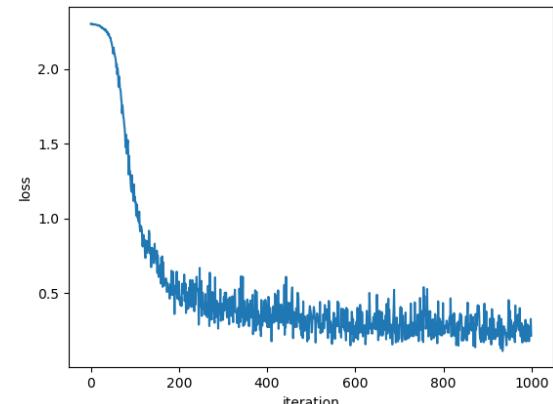
"신경망 학습을 위해 기울기를 수치미분을 통해서 구하면 계산 시간이 오래 걸린다."

• 실행 결과 확인

```

-- Start Learning --
iteration 0 : 2.29989748219
iteration 1 : 2.29828420777
iteration 2 : 2.30223055685
... 결과 생략 ...
iteration 997 : 0.220858715688
iteration 998 : 0.328162588765
iteration 999 : 0.216126172813
-- End Learning --

```



VI. Training

Techniques



1. 고급 경사하강법
2. 가중치 초기값 선정
3. 배치 정규화
4. 과적합 대응
5. Hands-on

신경망 학습의 목적은 매개변수(가중치, 편향)의 최적값을 찾는 것이다. 이를 **최적화(Optimization)**이라고 한다.

▪ 최적화 알고리즘

- 신경망 가중치, 편향의 최적화는 매우 어려운 작업. 수식으로 단번에 최솟값을 구할 수 있는 방법이 없다.
- 지금껏 최적화는 기울기를 이용해 기울어진 방향으로 매개변수 값을 갱신하는 일을 반복해서 최적의 값으로 다가감. (SGD, Stochastic Gradient Descent)
- SGD는 문제에 따라 비효율적일 때가 있는데 이를 보완하는 최적화 기법이 나오고 있다.

▪ 최적화 구현의 모듈화

- 딥러닝 구현 시 최적화를 담당하는 클래스를 분리해서 구현하면 기능을 모듈화 할 수 있다.
- 대부분의 딥러닝 프레임워크는 다양한 최적화 기법을 구현해 제공하며, 원하는 기법으로 쉽게 바꿀 수 있게 되어 있다.

[MNIST에서 Mini-batch SGD를 활용한 최적화]

```
# Gradient 계산
grad = network.numericalGradient(trainImgBatch, trainLblBatch)

# 가중치, 편향 갱신
for key in ('W1', 'W2', 'b1', 'b2'):
    network.params[key] -= learningRate*grad[key]
```

[Optimizer 구현]

```
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr
    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
class Momentum: ...
class Nesterov: ...
class AdaGrad:...
...
```

[Optimizer 호출]

```
optimizers[ "SGD" ] = SGD(Lr=0.95)
...
for i in range(max_iterations):
    ...
    grads = networks[key].gradient(x_batch, t_batch)
    optimizers[key].update(networks[key].params, grads)
```

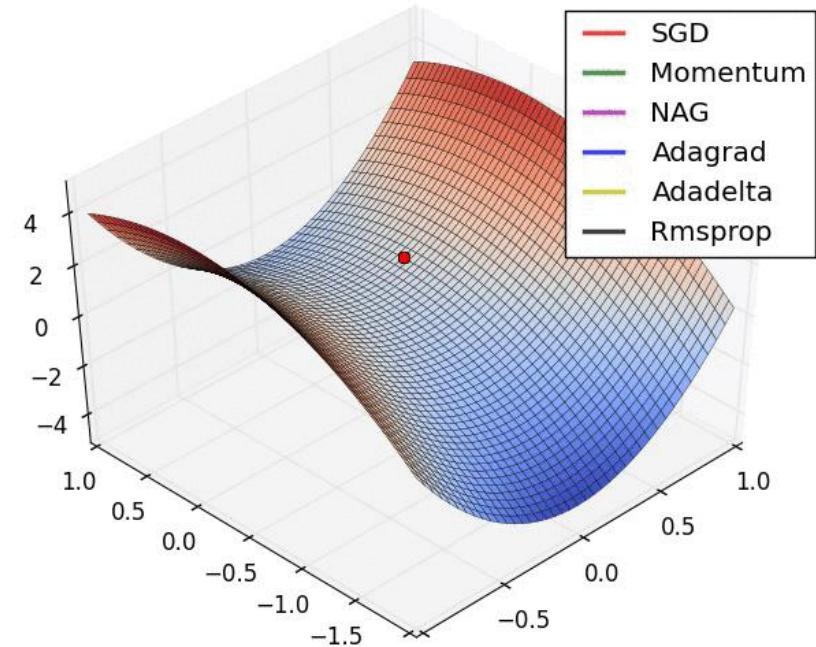
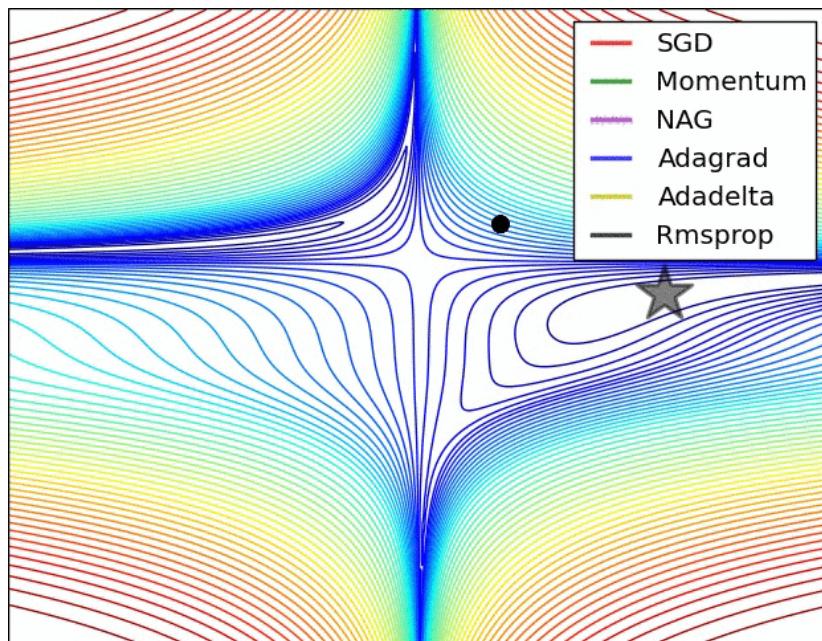
다양한 최적화 알고리즘을 비교해보자

▪ Gradient descent optimization algorithms

이름	특징
SGD (Stochastic Gradient Descent)	<ul style="list-style-type: none"> 기울어진 방향(기울기)으로 일정 거리(학습률)만 이동 단순하지만 문제에 따라 비효율적이다.
Momentum ('운동량')	<ul style="list-style-type: none"> 기울기 방향으로 물체가 가속되는 관성의 원리 적용. 속도 (Velocity)라는 변수가 추가 한 스텝전의 기울기를 일정한 비율로 반영하여 새로 계산된 기울기와 합해서 사용
Nesterov accelerated gradient (NAG)	<ul style="list-style-type: none"> Momentum에서 발전된 방법으로 momentum보다 더 공격적으로 속도(Velocity)를 활용 현재 가중치에서 기울기를 계산하기 전에, 먼저 학습했던 만큼 한 번 더 기본 상태로 계산
AdaGrad	<ul style="list-style-type: none"> 처음에는 학습률을 크게 하다가 조금씩 작게 학습하는 방법 전체 학습률을 조정하는 것이 아니라, 각각의 매개변수에 맞는 학습률을 조정. 과거의 기울기를 제곱해서 계속 더해감으로 학습이 진행될수록 갱신 강도가 약해짐
Adadelta	<ul style="list-style-type: none"> AdaGrad의 extension. 그러나 생각보다 잘 작동 안 함.
RMSprop	<ul style="list-style-type: none"> AdaGrad의 단점은 무한히 학습하다 보면 순간 갱신량이 0이 되어 학습이 안됨. 이를 개선하여 RMSprop은 먼 과거의 기울기는 잊고 새로운 기울기 정보를 크게 반영
Adam (Adaptive Moment Estimation)	<ul style="list-style-type: none"> Momentum과 AdaGrad의 이점을 조합. 학습률, 1차 momentum용 계수, 2차 momentum용 계수의 3개의 hyper paramter 설정

어떤 알고리즘을 선택해야 하는가?

▪ 최적화 기법 비교



- 모든 문제에 항상 뛰어난 기법은 없다. 즉, 문제 별로 최적화 기법을 선택 적용해야 한다.
- 학습률 등의 하이퍼 파라미터의 설정 등에 따라 최적화 알고리즘의 결과도 달라진다.
- 대체적으로 SGD, Momentum, Adagrad, Adam 등을 사용한다.
- Gradient descent optimization algorithms에 대해 자세히 알고 싶으면 아래를 참조한다.

<http://sebastianruder.com/optimizing-gradient-descent/>

신경망 학습에 가중치의 초기값 선정은 매우 중요하다.

- **Weight Decay 기법 (가중치 감소 기법)**

- 가중치를 작게 만들어 Overfitting을 억제하는 방법
- 가중치를 작게 만들려면 초기값을 최대한 작은 값에서 시작
- 초기값 선정 : 정규분포에서 Random하게 선택되는 값을 0.01배 한 작은 값 (표준편차가 0.01인 정규분포)

[MNIST 의 2층 신경망의 가중치 초기값 코드]

```
class MyTwoLayerNet:  
  
    def __init__(self, inputSize, hiddenSize, outputSize, weightInitStd = 0.01):  
        self.params = {}  
        self.params['W1'] = weightInitStd * np.random.randn(inputSize, hiddenSize)  
        self.params['W2'] = weightInitStd * np.random.randn(hiddenSize, outputSize)  
        self.params['b1'] = np.zeros(hiddenSize)  
        self.params['b2'] = np.zeros(outputSize)
```

- 가중치의 초기값을 0으로 만들면?
→ 학습이 되지 않는다. 순전파시 가중치가 모두 0이면 두 번째 층의 노드에 모두 같은 값이 전달 됨
- 표준편차가 1인 정규분포를 사용해서 초기값 선정 ($1 * \text{np.random.randn}(\text{inputSize}, \text{hiddenSize})$) 시?
→ 활성화함수로 Sigmoid 함수를 사용한다면 활성화 값이 0과 1로 치우쳐 분포됨.
→ 기울기 소실 (Gradient Vanishing)문제 발생 : 층이 깊을수록 학습이 안됨.
- 표준편차가 0.01인 정규분포를 사용해서 초기값 선정 ($1 * \text{np.random.randn}(\text{inputSize}, \text{hiddenSize})$) 시?
→ 활성화함수 값이 거의 같은 값을 출력함으로 층이 깊어지는 의미가 없음.

“각 층의 활성화 값은 적당히 고루 분포되어야 학습이 잘 이루어진다.”

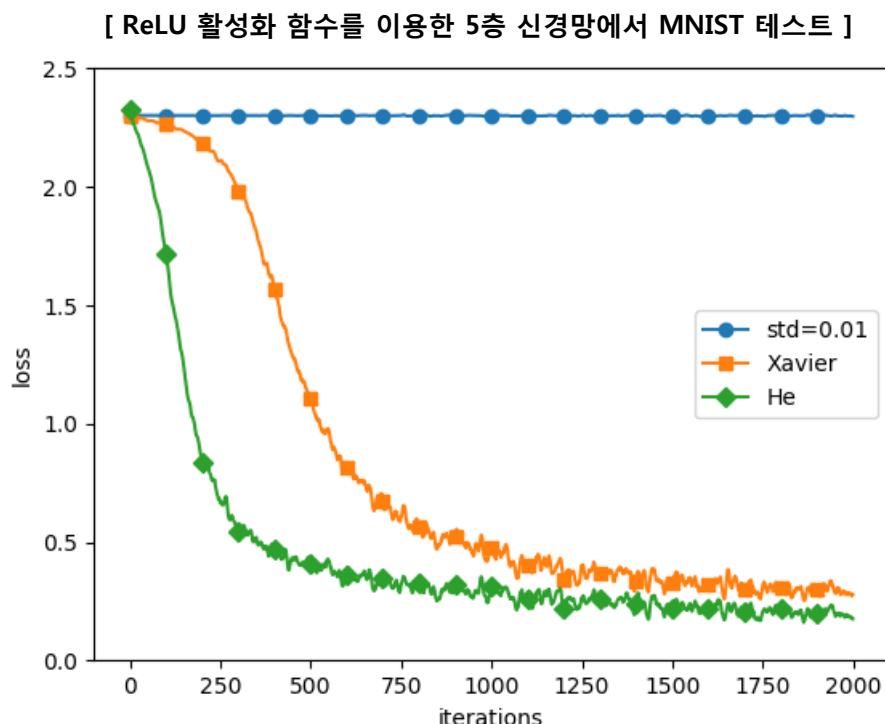
Sigmoid 함수나 tanh 함수를 활성화 함수로 이용할 때 Xavier 초기값을 하고 ReLU인 경우 He 초기값을 이용한다.

▪ Xavier (사비에르) 초기값

- Xavier Glorot와 Yoshua Bengio 논문에서 권장하는 가중치 초기값.
- 활성화 함수로 Sigmoid 함수나 tanh함수를 사용할 때 적용
- 일반적인 딥러닝 프레임워크가 표준적으로 이용
- 초기값의 표준편차가 $\sqrt{\frac{1}{n}}$ 인 정규분포로 초기화 (여기서 n은 앞 층 노드의 수)

▪ He 초기값

- Kaiming He 가 발견한 초기 값.
- 활성화 함수가 ReLU일 때 적용
- 초기값의 표준편차가 $\sqrt{\frac{2}{n}}$ 인 정규분포로 초기화
(여기서 n은 앞 층 노드의 수)

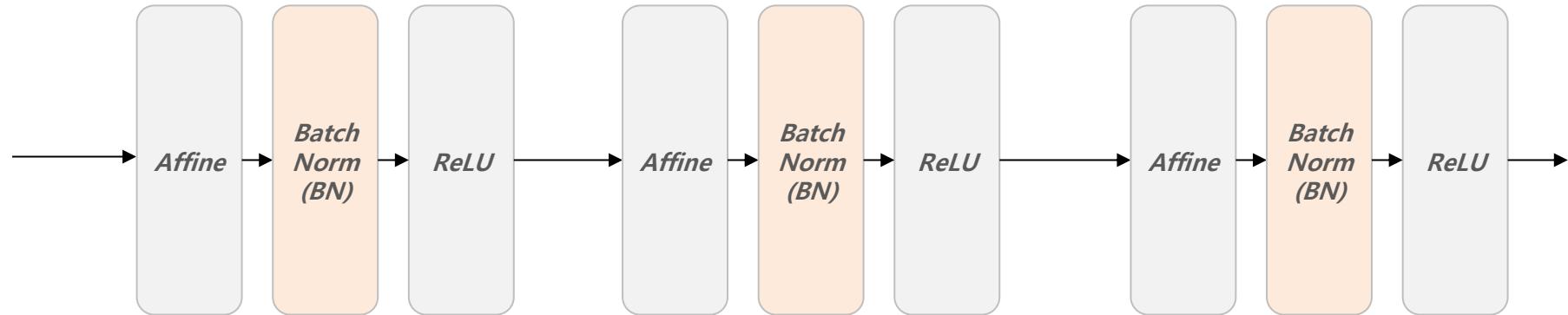


배치 정규화 (Batch Normalization)은 각 층에서의 활성화 값이 적당히 분포되도록 조정하는 것이다.

▪ Batch Normalization

Important

- 데이터 분포를 정규화하는 배치 정규화 (Batch Norm) 계층을 신경망에 삽입 (2015년도에 제안)



• Batch Normalization의 장점

- ✓ 학습 속도의 개선
- ✓ 가중치 초기값에 크게 의존하지 않음.
- ✓ Overfitting을 억제한다. (Dropout 등의 필요성 감소)

• Batch Normalization에 대해 좀 더 자세한 내용

- ✓ <https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/>

[Batch Normalization 수식]

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$$

"Batch Normalization의 경우 Deep Learning의 필수적인 기법이 되어가고 있다."

Over-fitting이란 신경망이 훈련 데이터에 지나치게 적응되어 실 데이터에는 성능이 안 나오는 것이다.

▪ 신경망에서의 Over-fitting

- 신경망에서의 Overfitting은 다음 경우에 자주 발생함
 - ✓ 매개변수가 많고 표현력이 높은 경우 (층이 많은 경우)
 - ✓ 훈련 데이터가 적은 경우
- 과적합 대응 방법 : Weight Decay, Dropout 등

▪ Weight Decay (가중치 감소)

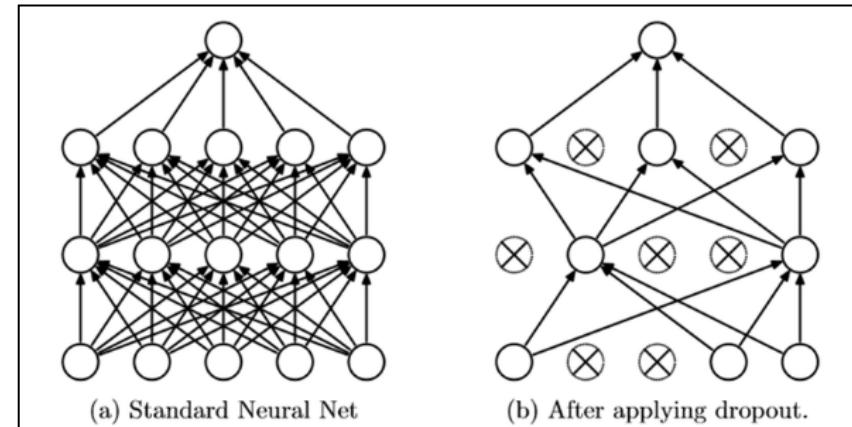
- 원래 over-fitting은 가중치 값이 커서 발생하는 경우가 많다.
- 가중치를 감소시키면 over-fitting이 억제되는 효과가 있다.
- 하지만 신경망 모델이 복잡해지면 이걸로 불충분하다.

▪ Dropout Algorithm

- 2012년 Hinton 교수가 제안한 것으로 과적합 문제를 개선.
- 신경망 학습 시 임의의 노드 몇 개를 생략해 학습하고, 다음 계산 시 또 다른 노드를 생략하고 학습
- 일반적으로 입력층에서 20%~25% 정도, 은닉층에서 50% 정도의 노드를 생략한다.
- 효과

"학습 시 학습 데이터에 의해 연결의 weight가 동조화 되는 현상 (**co-adaptation**)을 피한다."

* **Co-Adaptation** : 노이즈에서 발생한 잘못된 특성이 모든 노드에 전파되어 의미 있는 특성을 선별하는데 부정적인 영향을 미치는 것

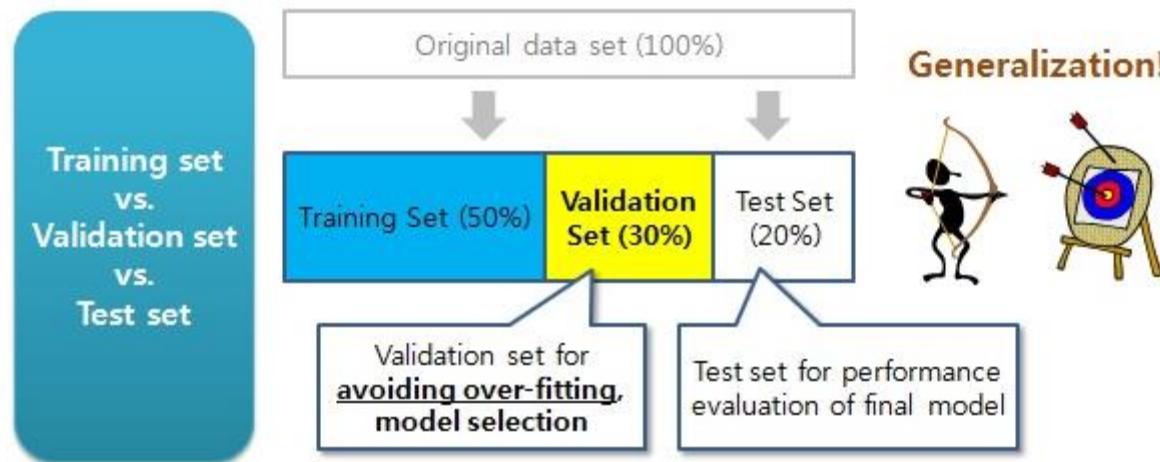
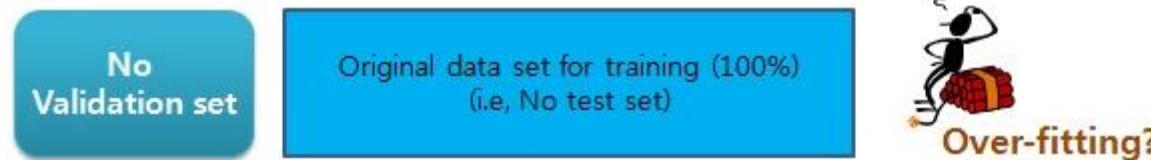


[Dropout Algorithm]

"Dropout의 경우 효과는 좋지만 학습 속도가 다소 느려진다."

"Batch Normalization의 경우 가중치 초기화 문제와 과적합 문제를 동시에 해결할 수 있다."

과적합을 피하는 다양한 방법 중 검증(Validation) 기법을 알아본다.



- 학습 데이터의 일부를 따로 떼어 놓아 학습이 아닌 검증용으로 사용하는 기법

보유하고 있는 데이터셋을 Training set (50%~60%), Validation set (20%~25%), Test set (20%~25%) 으로 구분

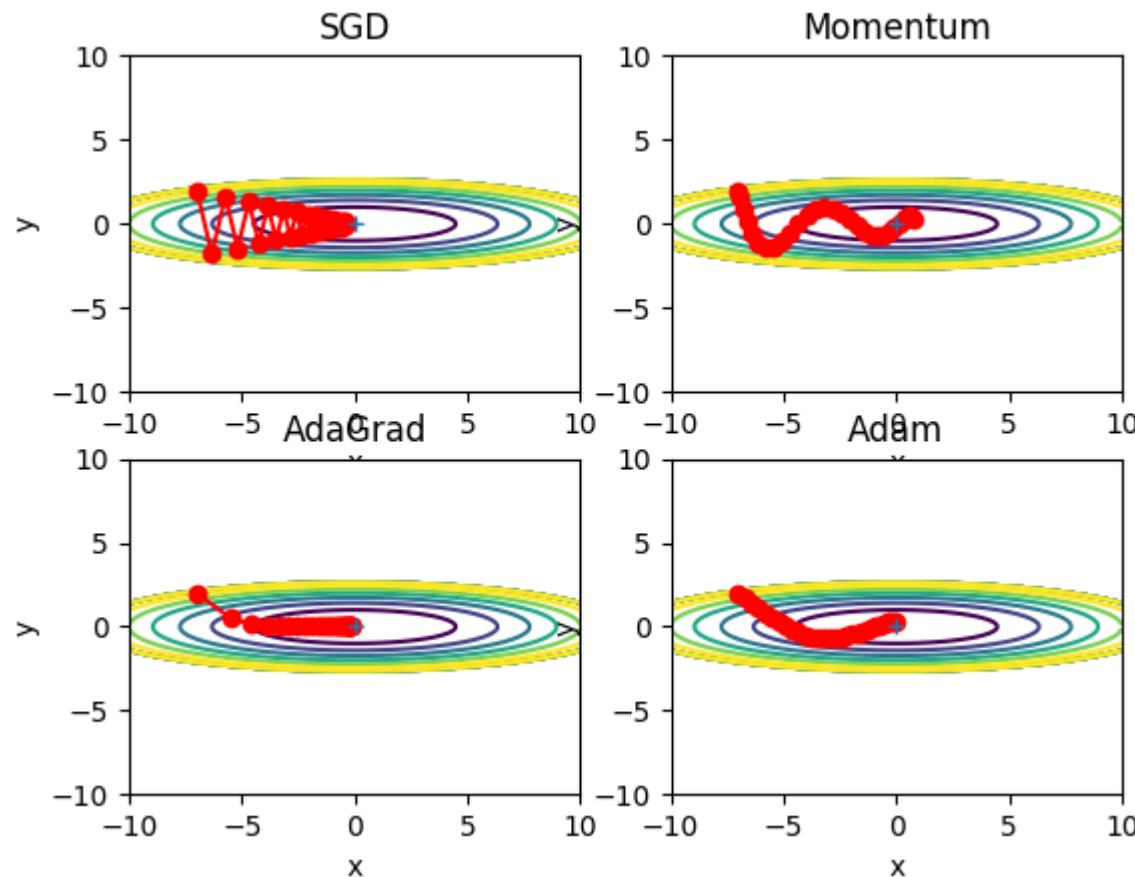
- (1) Training set 을 가지고 예측 혹은 분류 모델을 훈련시키고
- (2) Validation set을 가지고서 훈련 중인 모델이 혹시 과적합(over-fitting)인지 검증, 감시를 하면서 Hyper Parameter를 선정
- (3) Test set을 사용해서 최종 모델(final model)에 대해서 평가

- 교차 검증 (Cross Validation)

학습 데이터를 용도별로 나누는 것은 같은데, 처음에 나눠진 데이터를 그대로 사용하는 것이 아니라 중간중간에 서로 다시 나눠 줌

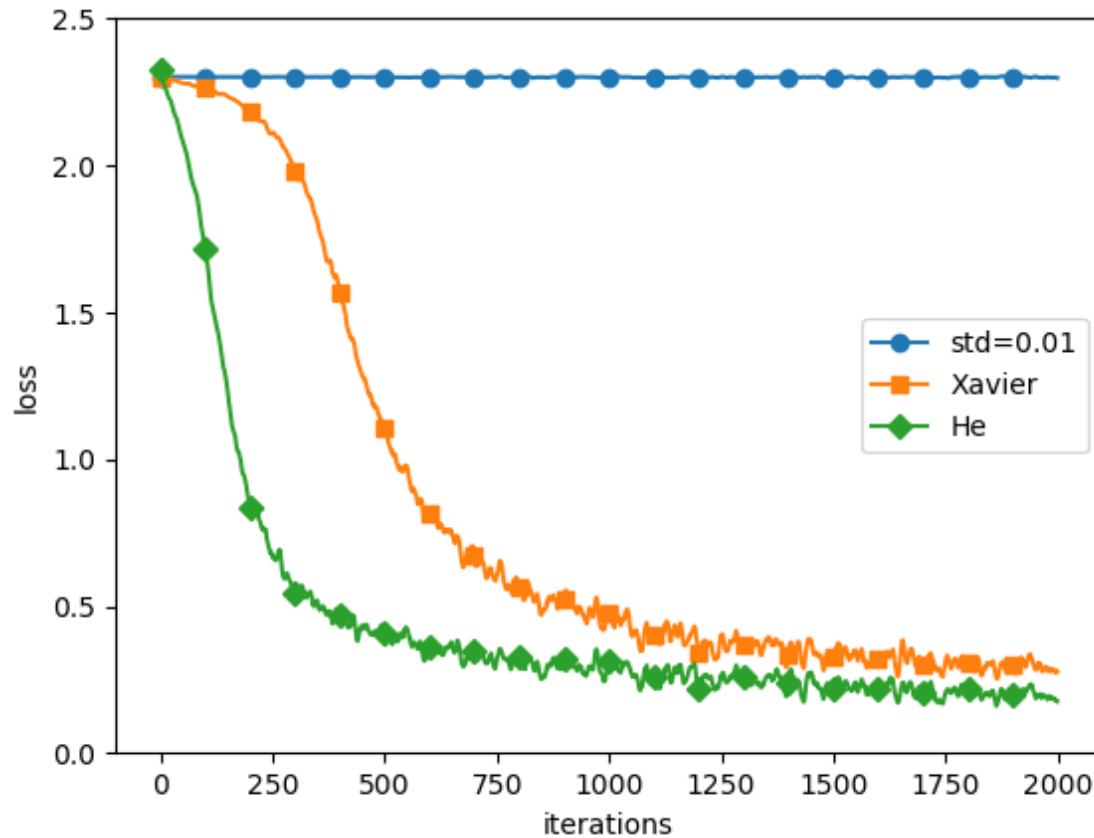
#1 – 경사하강법 비교하기

- myoptimizercomparenaive.py 소스와 실행결과 확인



#2 – 가중치 초기값 선정 방법 비교하기

- myweightinitcompare.py 소스와 실행결과 확인



VII. Convolution

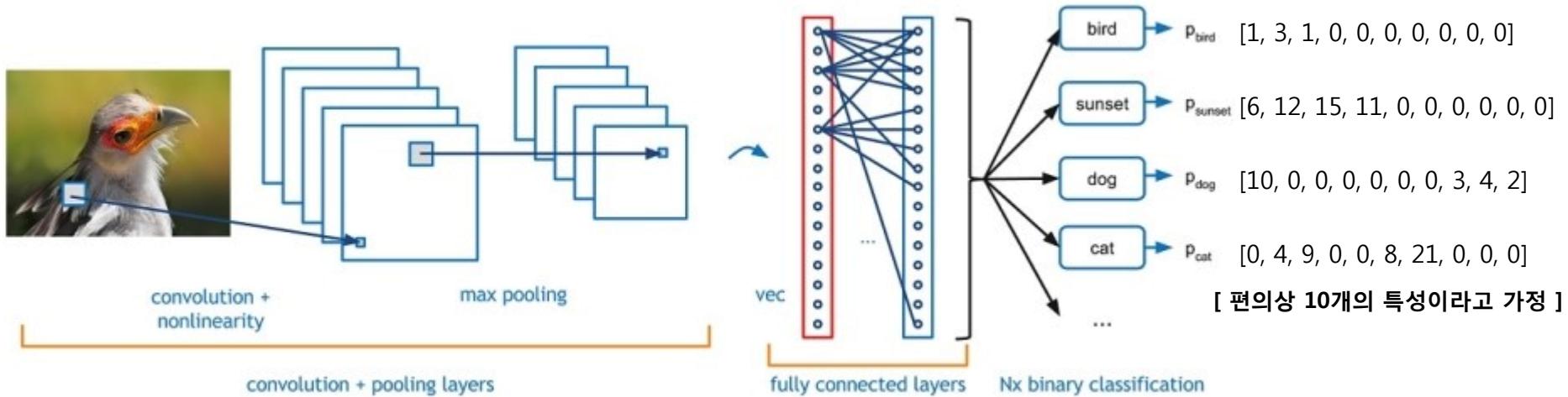
Neural Network



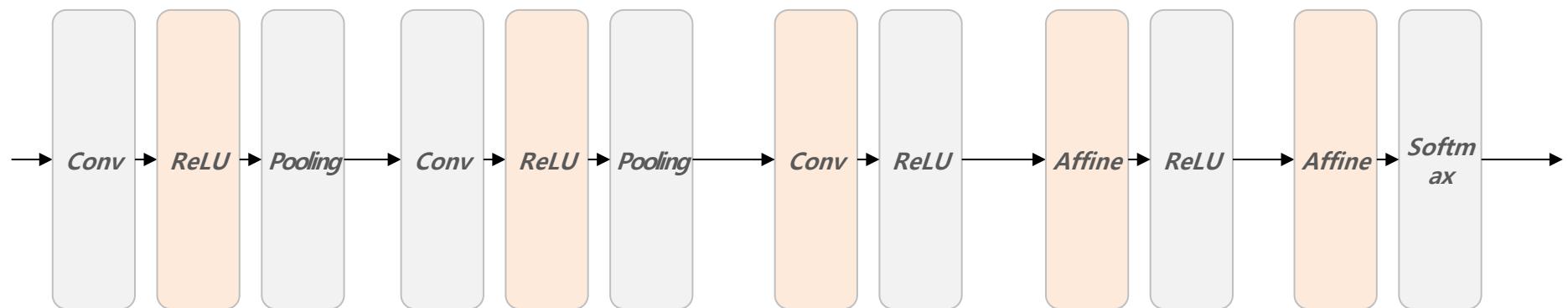
1. 전체 구조
2. Convolution Layer
3. Pooling Layer
4. CNN 구현
5. CNN 시각화
6. 대표적인 CNN

CNN은 Convolution Layer와 Pooling Layer를 포함하는 신경망으로 이미지 처리에 좋은 성능을 보인다.

- CNN은 Convolution과 Pooling을 반복하면서 영상을 줄이고 최종적으로 1차원 벡터 형태로 특성으로 압축하여 분류한다.



- CNN으로 이루어진 Network 예



2. Convolution Layer

VII. Convolution Neural Network

이미지는 통상 3차원 데이터 (세로, 가로, 색상(채널))인데 Convolution Layer에서는 형상을 유지하며 연산한다.

▪ Convolution 연산

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Image

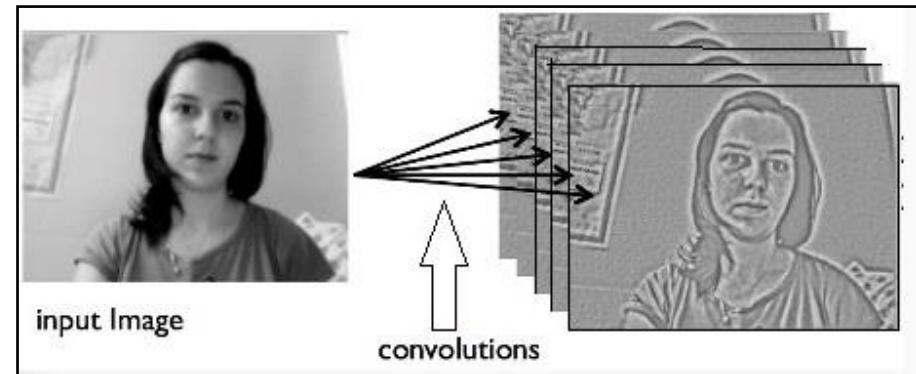
4		

Convolved Feature

1	1	1	0	0
0 x1	1 x0	1 x1	1	0
0 x0	0 x1	1 x0	1	1
0 x1	0 x0	1 x1	1	0
0	1	1	0	0

4	3	4
2		

- 기존 영상처리에 잘 사용하는 기법으로 영상필터로 사용된 기술
- 가중치를 갖는 마스크(Convolution Filter, 커널)를 이용해서 특징 추출 (Feature Map)
- 연산 시 입력과 필터에서 대응하는 원소끼리 곱한 후 그 총합을 구하는데 이를 fused multiply-add (FMA) 라고 한다.
- Convolution 의 결과 예 (Filter 수 만큼 Feature Map이 나온다.)



- CNN에서도 Bias는 존재하며, Bias는 1x1 Shape로 존재하여 필터 적용한 후에 데이터에 더해진다.
- 필터의 값은 학습에 의해 바뀐다.

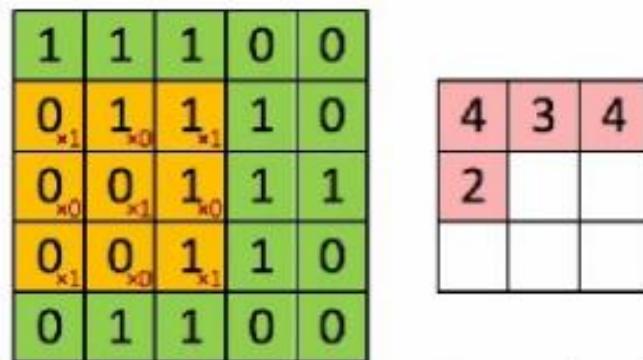
2. Convolution Layer

VII. Convolution Neural Network

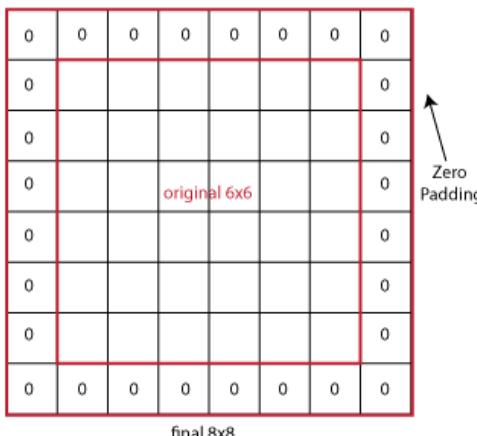
이미지는 통상 3차원 데이터 (세로, 가로, 색상(채널))인데 Convolution Layer에서는 형상을 유지하며 연산한다.

▪ Padding

- Convolution 연산 수행 전 입력 데이터 주변에 특정 값을 채우는 작업
- 출력 크기를 조정하려는 목적으로 사용
- 예) 아래 (5,5) 입력 데이터에 (3,3) 필터를 적용하면 이미지가 (3,3)으로 줄어든다. 즉 Convolution 연산을 반복하다 보면 크기가 작아진다.



- Padding 처리



- ✓ Padding 크기가 1이면 입력 데이터 사방 1 Pixel을 특정 값으로 채운다.
- ✓ Padding 크기는 정수로 설정하면 된다.

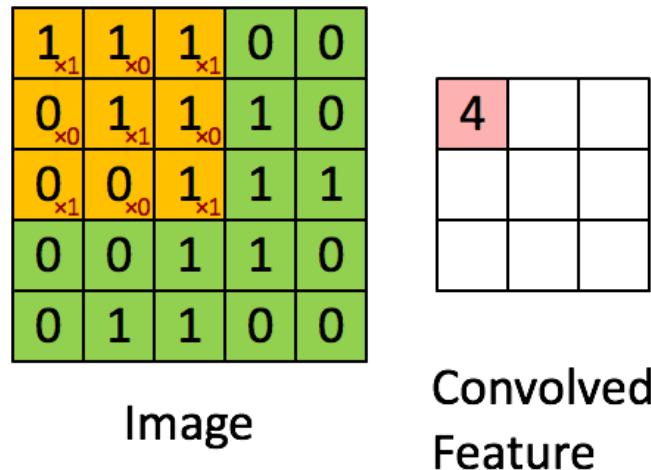
2. Convolution Layer

VII. Convolution Neural Network

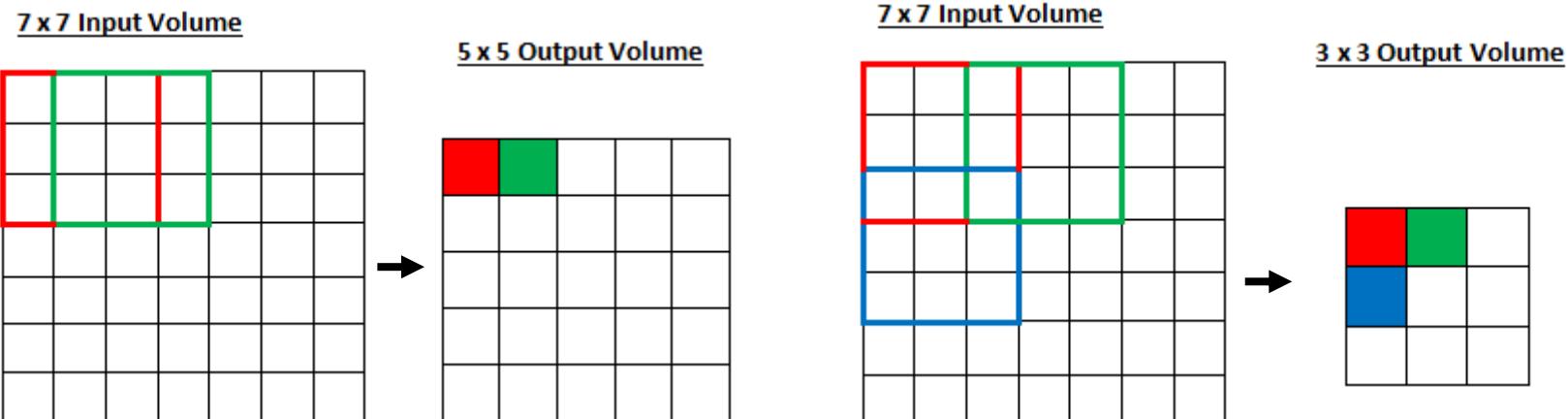
이미지는 통상 3차원 데이터 (세로, 가로, 색상(채널))인데 Convolution Layer에서는 형상을 유지하며 연산한다.

▪ Stride

- Filter를 적용하는 위치의 간격을 Stride라고 한다.
- 예) Stride가 1인 Convolution 연산



- Stride 크기를 키우면 출력 크기가 작아진다.



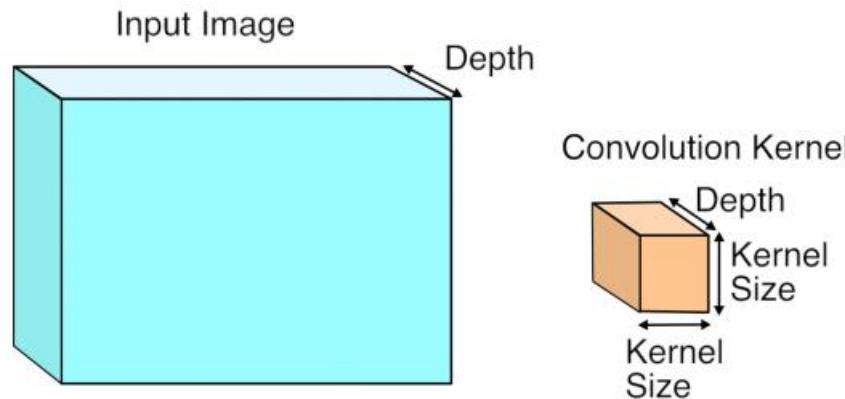
2. Convolution Layer

VII. Convolution Neural Network

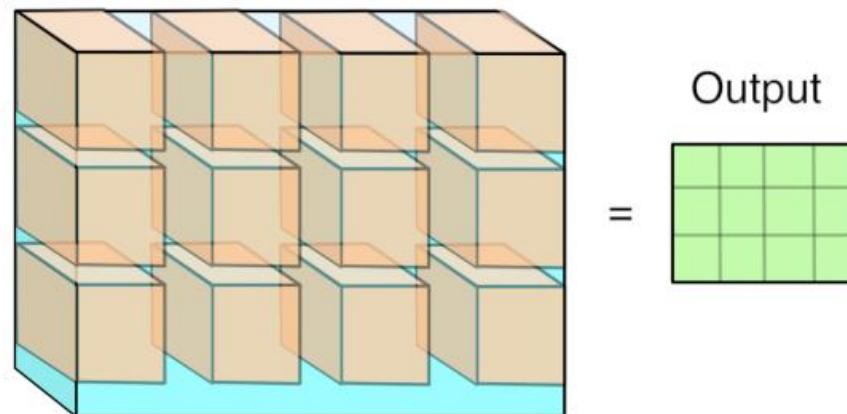
이미지는 통상 3차원 데이터 (세로, 가로, 색상(채널))인데 입력 데이터와 필터의 채널의 크기는 같아야 한다.

▪ 3차원 데이터의 Convolution 연산

- 3차원 Convolution 연산에서 입력 데이터와 필터의 채널 수가 같아야 한다



- 3차원 데이터의 Convolution 연산 결과는 2차원으로 나온다. (이 출력데이터를 한 장의 feature map이라고 한다.)

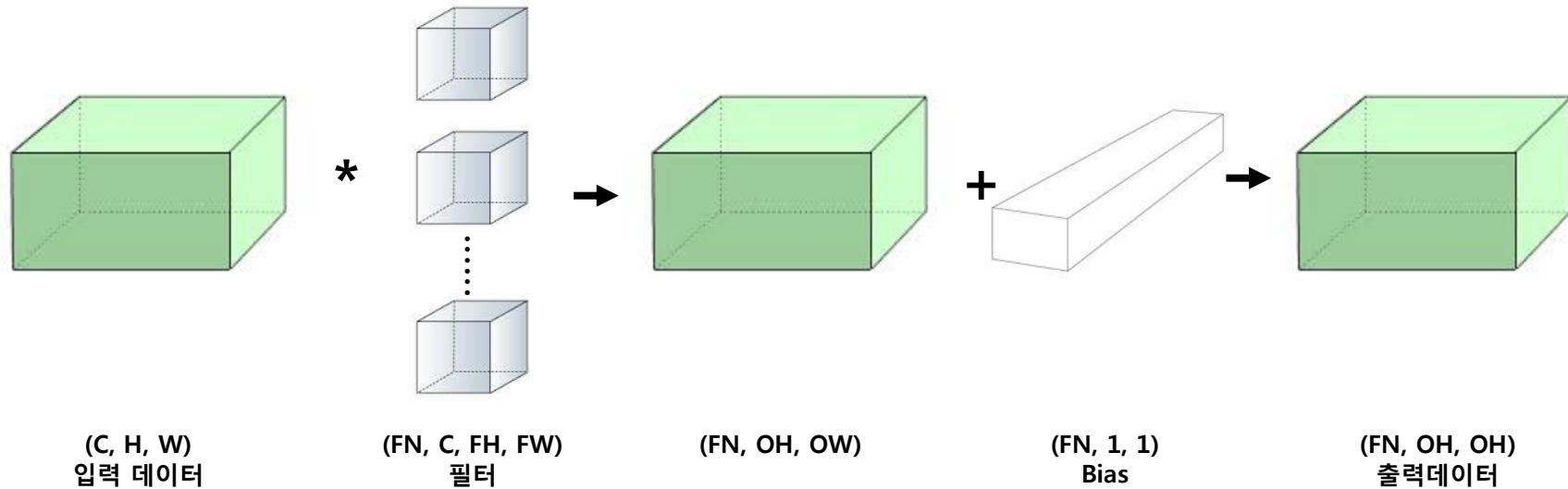


$$(C, H, W) * (C, FH, FW) \rightarrow (1, OH, OW)$$

- C : Channel
 - H : Height
 - W : Width
 - FH : Filter Height
 - FW : Filter Width
 - OH : Output Height
 - OW : Output Weight

Convolution 연산에 사용하는 Filter의 수를 늘려 다수의 Feature map을 생성하여 Block을 만든다.

- 다수의 Filter 적용 시 Convolution 연산



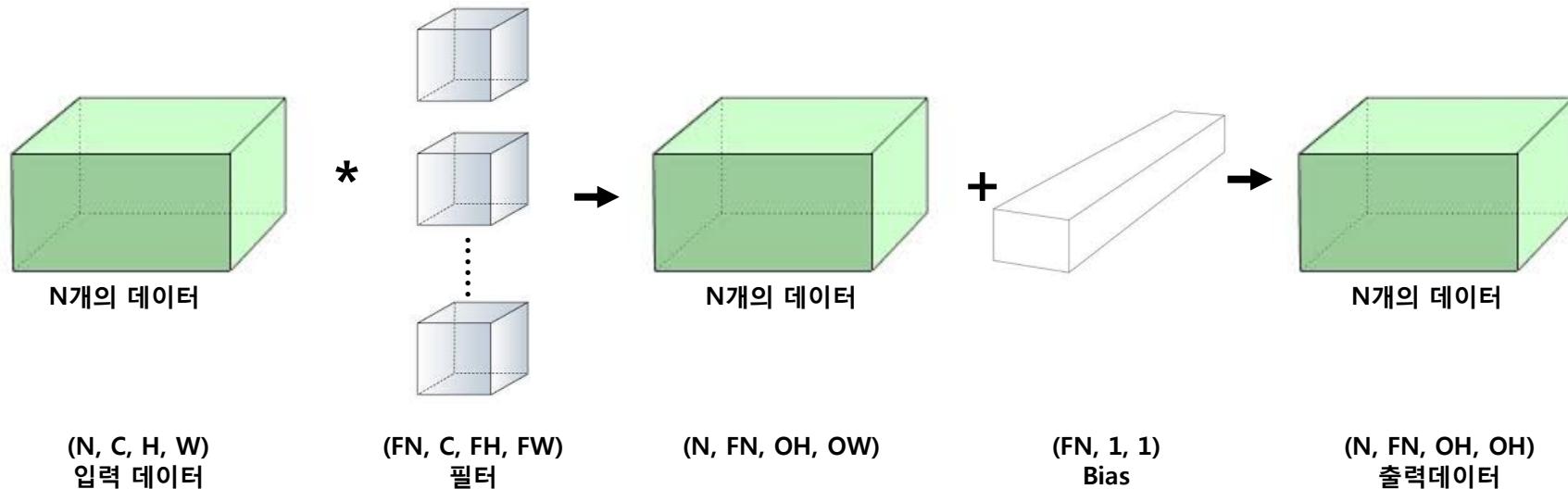
- 필터가 FN개이면 출력 맵도 FN개가 된다.
- 그 FN개의 맵을 모으면 형상이 (FN, OH, OW) 가 되며 이 블록을 다음 계층으로 넘긴다.
- Convolution 연산에서 Filter의 수를 고려해야 하며 Filter의 가중치는 4차원 데이터가 된다.
(출력 채널 수, 입력 채널 수, 높이, 너비)

2. Convolution Layer

VII. Convolution Neural Network

신경망에서 미니 배치로 학습해야 효율이 높다.

- 배치 처리를 추가한 Convolution 연산



- 배치 처리 시 데이터는 4차원 형상을 가진 채 각 계층을 타고 흐른다.
- 신경망에 4차원 데이터가 하나 흐를 때마다 데이터 N 개에 대한 Convolution 연산이 이루어 진다. (즉, N 개 배치 처리가 된다.)

3. Pooling Layer

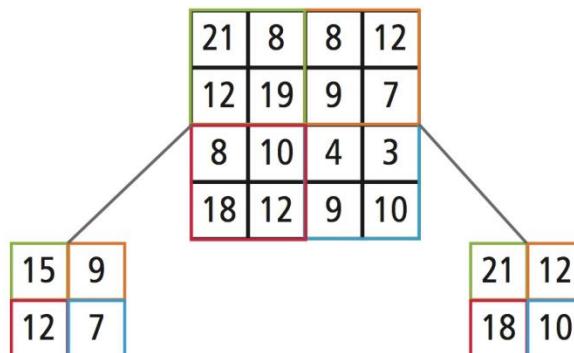
VII. Convolution Neural Network

CNN에서 Pooling은 통해 데이터의 크기를 줄이는 연산이다.

▪ Pooling(Subsampling)

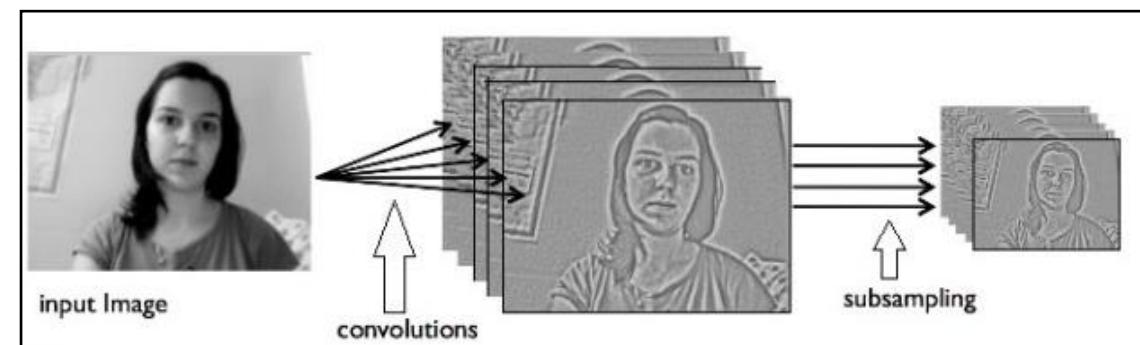
- Pooling 계층은 주위의 픽셀을 묶어서 하나의 대표 픽셀로 바꾼다. 즉 이미지의 차원을 축소한다.
- 이미지의 인식 대상이 한 쪽으로 치우치거나 돌아가 있어도 어느 정도 보상을 해 줌.
- 이미지 크기를 줄이고 과적합을 방지하는 데 도움이 됨
- 일반적으로 Pooling의 윈도우 크기와 Stride는 같은 값으로 설정한다.

Max Pooling (최대 풀링)	Average Pooling (평균 풀링)	Stochastic Pooling (확률적 풀링)
<ul style="list-style-type: none">• 컨볼루션 데이터에서 가장 큰 것을 대표 값으로 취함• 직관적 계산이 편리• 최대값이 노이즈이면 Overfitting 유발• 이미지 인식 분야에 주로 사용	<ul style="list-style-type: none">• 컨볼루션 데이터에서 모든 값의 평균을 취함• 직관적이고 간편• 낮은 활성화 값이 평균에 포함됨으로 특성 대비율을 떨어뜨림	<ul style="list-style-type: none">• 컨볼루션 데이터에서 임의의 확률로 한 개를 선정 취함• 실행 방법이 단순• 범위 내 동일한 값이 여러 개 있다면 확률적으로 선택 가능성이 높아 대표성을 띤다.



Average Pooling

Max Pooling



[Convolution후 Subsampling 처리]

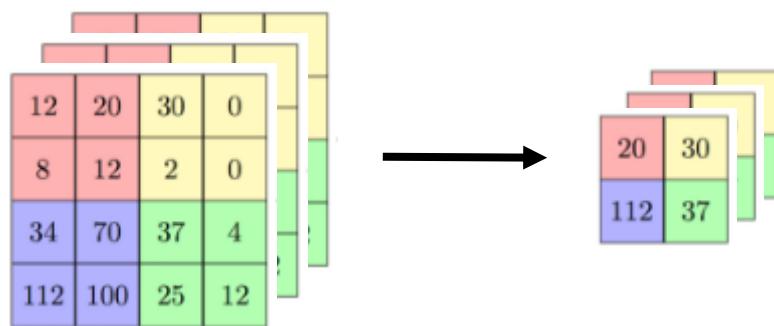
Pooling layer의 특징을 알아보자

- 학습해야 할 매개변수가 없다.

- Pooling은 대상 영역에서 Max 값 또는 평균 값을 취하는 명확한 처리임으로 학습이 필요 없다.

- 채널 수가 변하지 않는다.

- Pooling은 채널마다 독립적으로 계산함으로 입력 데이터 채널 수 그대로 출력 데이터로 보낸다.



- 입력의 변화에 영향을 적게 받는다. (강건하다)

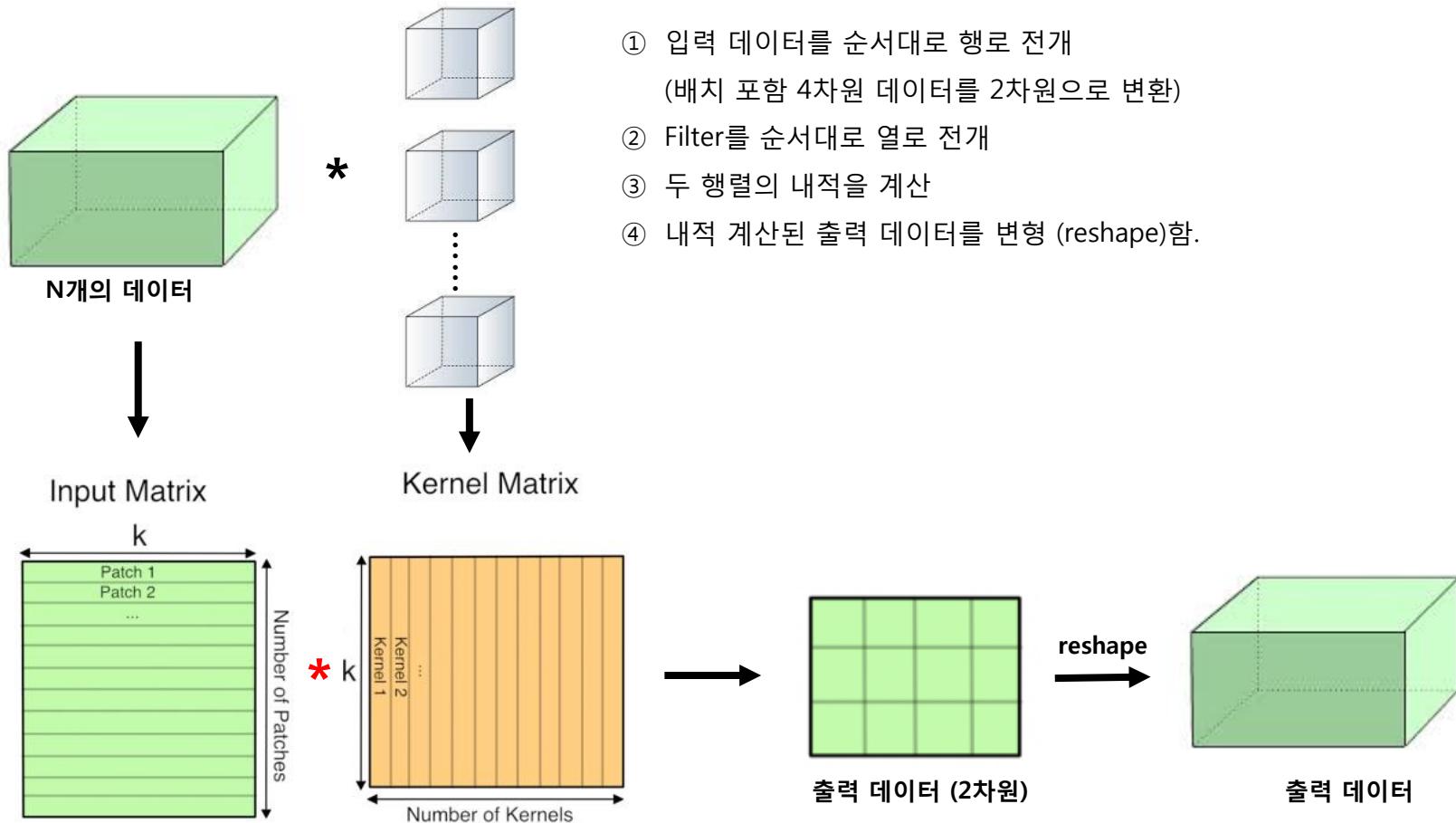
- 입력 데이터가 조금 변해도 Pooling의 결과는 잘 변하지 않는다.

Convolution Layer와 Pooling Layer는 im2col를 이용하면 효율적으로 구현할 수 있다.

▪ im2col

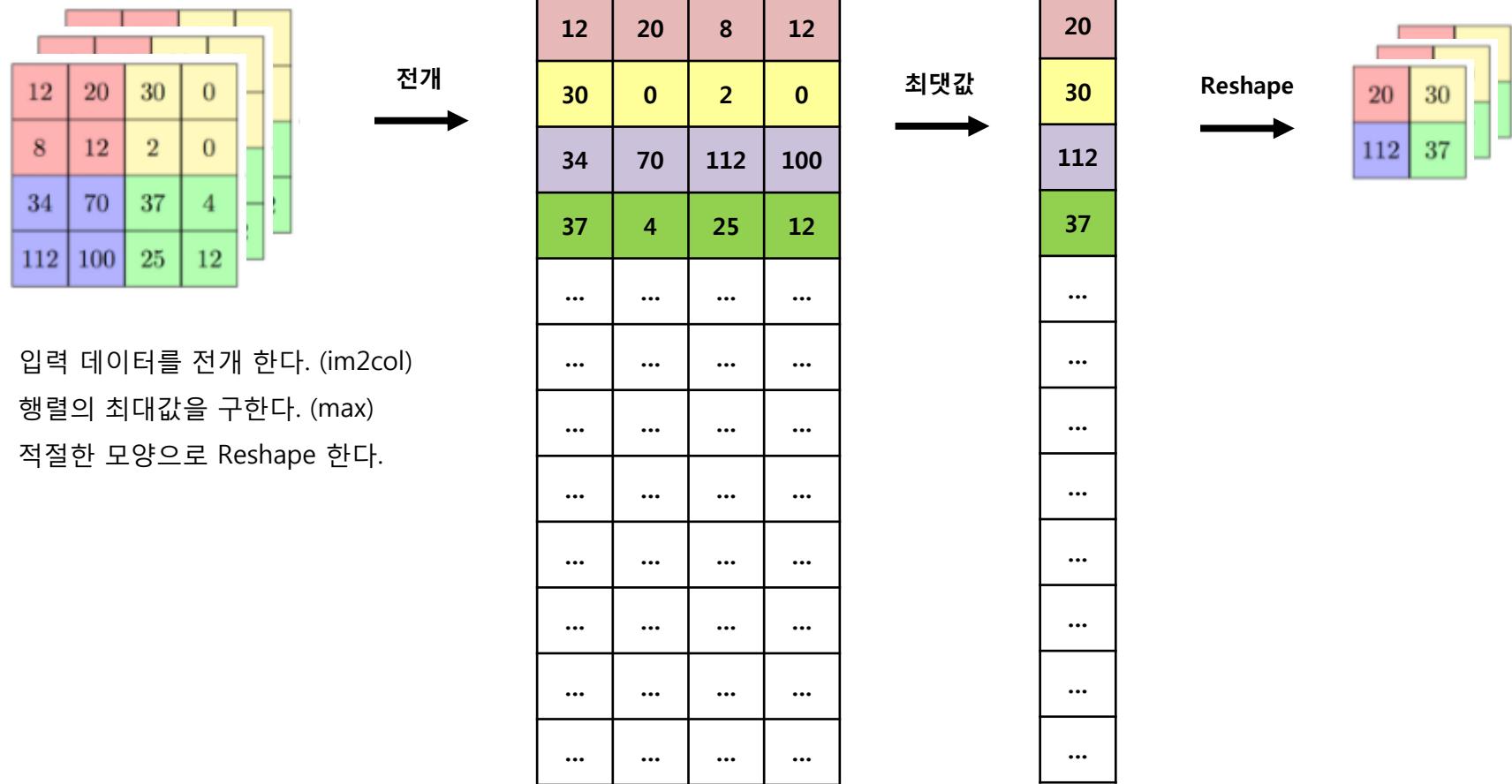
- Image to column, 즉 '이미지에서 행렬로'라는 뜻이다.
- Im2col은 입력 데이터를 filtering(가중치 계산)하기 좋게 전개하는 함수이다.

▪ Convolution Layer의 연산과정



Pooling Layer도 im2col을 이용하지만 채널마다 독립적으로 전개하는 것이 다르다.

- Pooling Layer의 연산과정

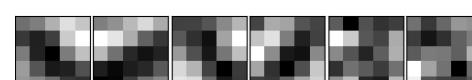
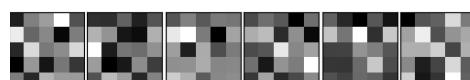
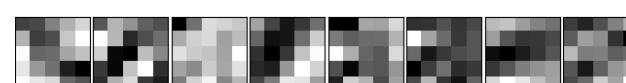
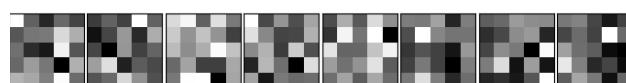
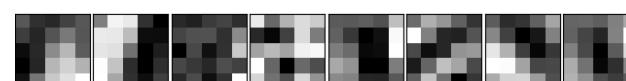
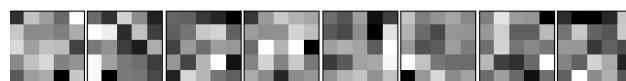
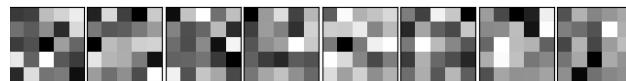


- ① 입력 데이터를 전개 한다. (im2col)
- ② 행렬의 최대값을 구한다. (max)
- ③ 적절한 모양으로 Reshape 한다.

Convolution Layer를 시각화하면 CNN의 동작 원리를 이해해 본다.

- MNIST 예제

- 첫 번째 Convolution Layer 에서는 학습 후 Edge 나 Blob(덩어리)의 저수준 정보가 추출된다.
- MNIST 에서 필터 30개, 채널1개, 5x5 크기인 필터 예 (형상 : 30, 1, 5, 5)
- 가중치의 값은 실수 이지만, 가장 작은 값은 0, 가장 큰 값은 255로 정규화하여 표시한다.
- 학습 전 필터는 Random 값으로 초기화 되지만, 학습 후 필터는 규칙적이 되어 있다.



[학습 전 필터의 가중치]

[학습 후 필터의 가중치]

첫 번째 Convolution Layer의 필터는 Edge와 Blob의 원시적인 정보를 추출한다.

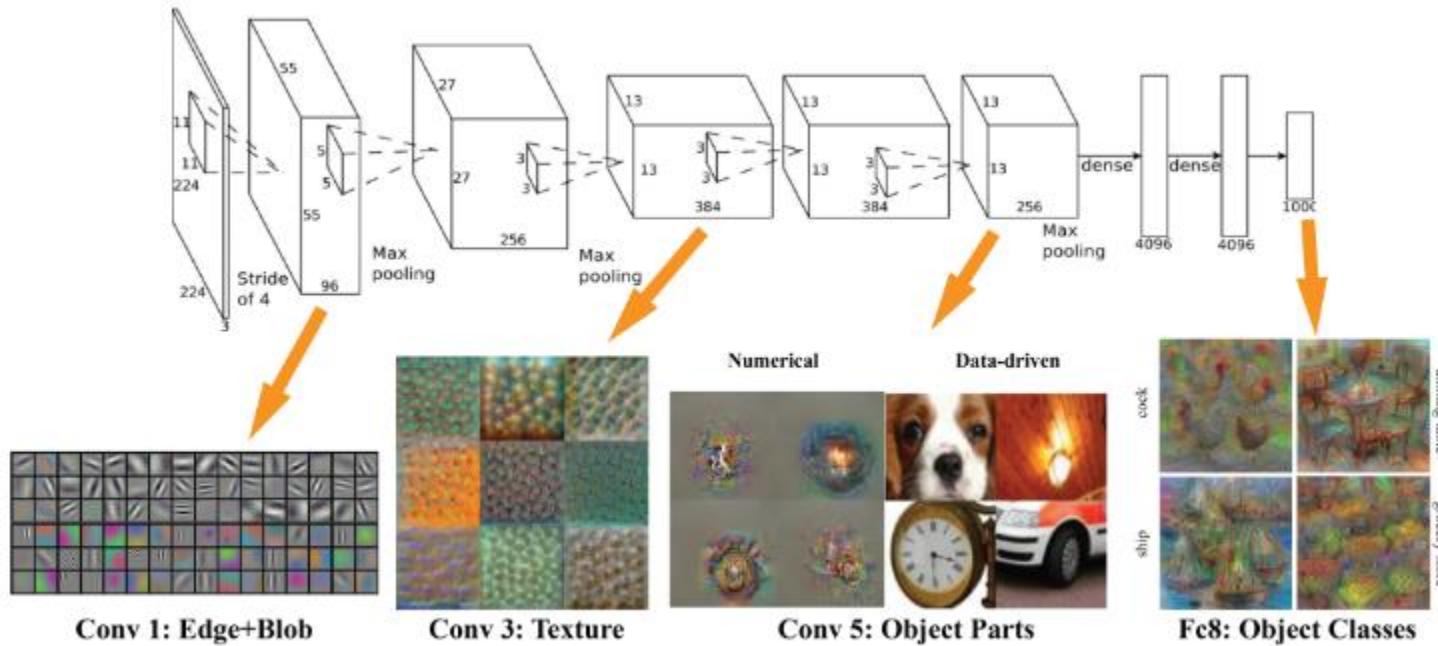
- 학습 후 첫 번째 Filter가 반응한 것.

- Edge (색상이 바뀐 경계선)에 반영
- Blob (국소적으로 둉어리진 영역)에 반영
- Edge나 Blob의 원시적인 정보가 뒷단 계층으로 전달 됨.



CNN에서는 층의 깊어질 수록 추출되는 정보가 추상화 된다.

- AlexNet (CNN)에서의 중간 계층에서 추출되는 정보 예



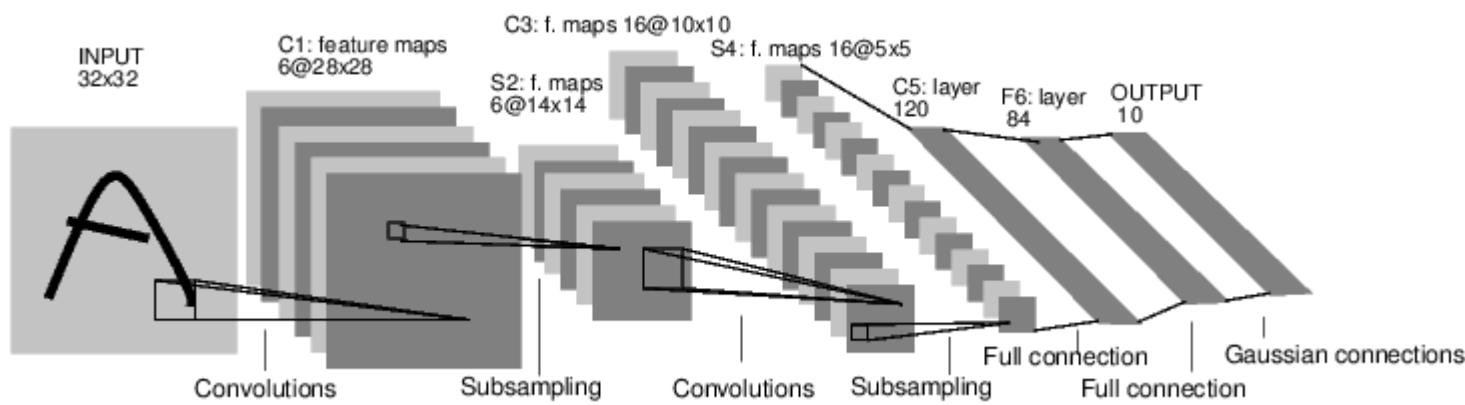
- 1번째 층에서는 Edge와 Blob, 3번째 층에서는 Texture, 5번째는 사물의 일부, 8번째는 Class 정보가 추출된다.
- 즉, 처음에는 단순한 것에 반응하지만 층이 깊어지면서 추상화된 정보가 추출된다.

- CNN을 적용한 MNIST 예제 처리과정 시각화 확인

- <https://transcranial.github.io/keras-js/#/mnist-cnn>

LeNet은 1998년 MNIST를 인식하기 위한 CNN으로 제안되었다.

- LeNet

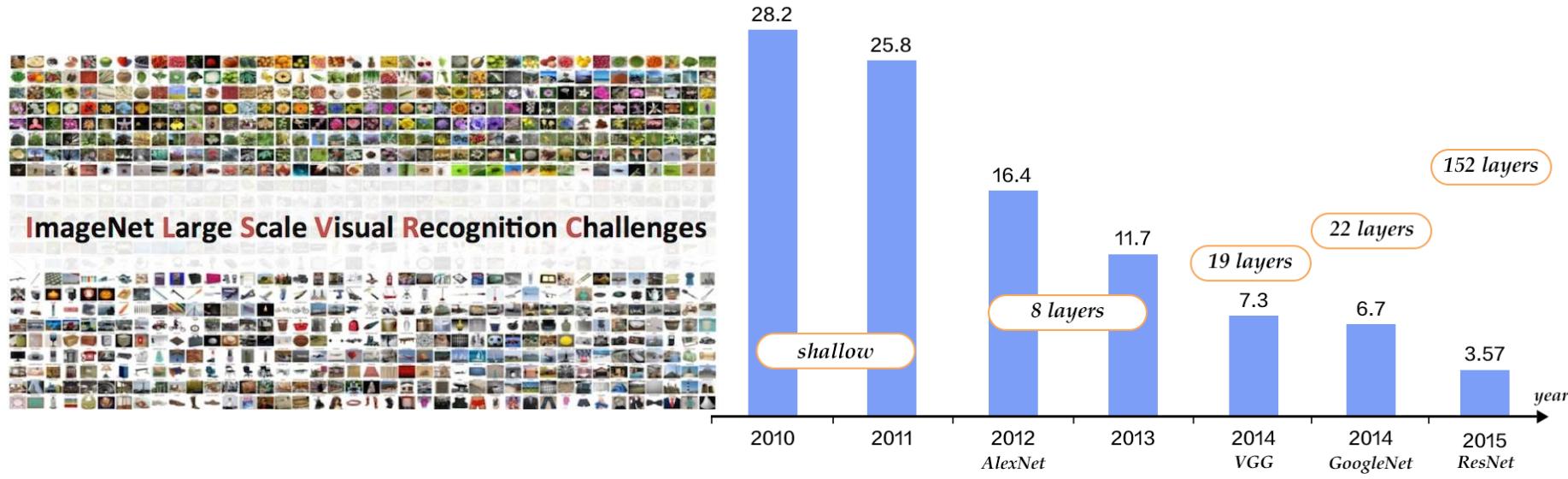


- 활성화 함수로 Sigmoid 함수를 사용. (현재 대부분의 CNN은 ReLU함수를 주로 사용)

이미지넷의 ILSVRC 대회를 통해 이미지 인식을 위한 Deep Learning이 큰 주목을 받음.

- ILSVRC (ImageNet Large Scale Visual Recognition Challenge)

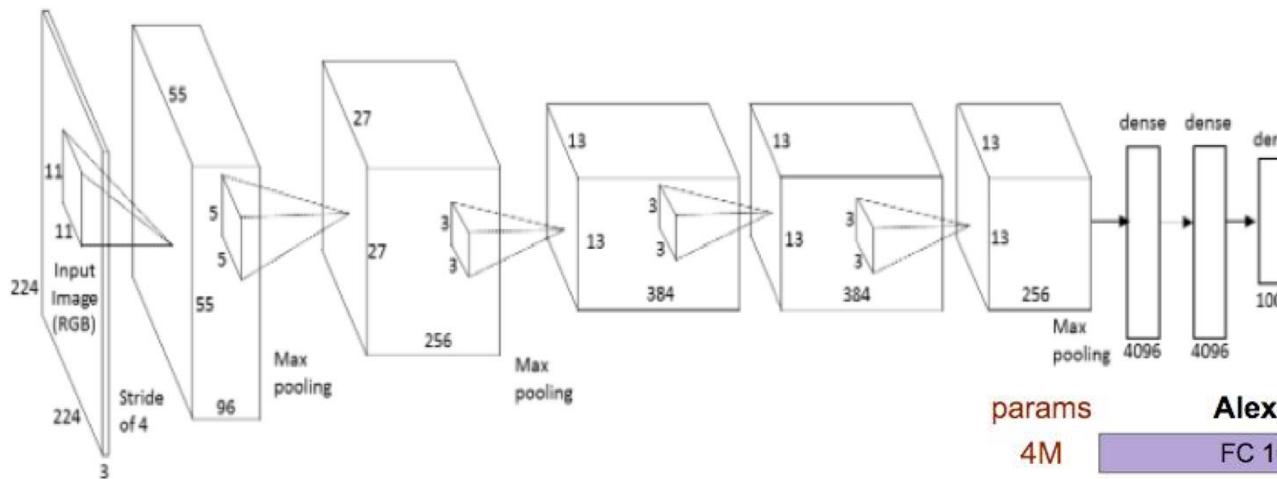
- ImageNet이 주관하는 이미지 인식 콘테스트 (매년)
 - ✓ 이미지넷이 보유한 레이블된 이미지를 학습데이터로 하여 1,000가지 종류로 분류하는 대회
- 2012년 CNN을 적용한 팀이 우승 후 2013년부터 딥러닝을 이용해 참가한 팀이 급증
- 기존의 이미지 인식 기법인 BoF(Bag of Features)와 비교해 정밀도가 압도적으로 높아짐



- 2015년에는 오류율이 3.57%까지 낮춰졌음. 이는 일반적인 사람의 인식 능력을 넘어선 결과임.

AlexNet은 2012년에 ILSVRC에서 우승한 CNN이다.

- AlexNet

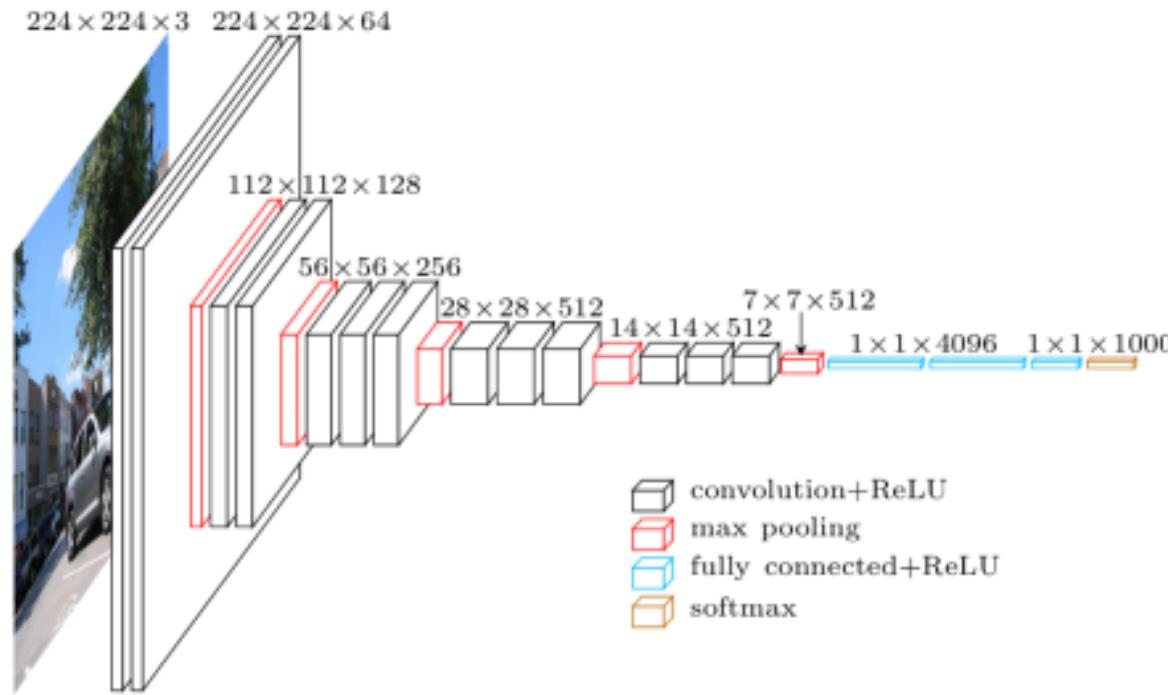


- 8층 CNN
- 활성화 함수로 ReLU함수를 사용
- Local Response Norm이라는 국소적 정규화 계층이 존재
- Dropout을 사용

params	AlexNet	FLOPs
4M	FC 1000	4M
16M	FC 4096 / ReLU	16M
37M	FC 4096 / ReLU	37M
442K	Max Pool 3x3s2	
1.3M	Conv 3x3s1, 256 / ReLU	74M
884K	Conv 3x3s1, 384 / ReLU	112M
307K	Conv 3x3s1, 384 / ReLU	149M
35K	Max Pool 3x3s2	
	Local Response Norm	
	Conv 5x5s1, 256 / ReLU	223M
	Max Pool 3x3s2	
	Local Response Norm	
	Conv 11x11s4, 96 / ReLU	105M

VGG는 2014년에 ILSVRC에서 2위한 CNN이다. (1위는 GoogleNet)

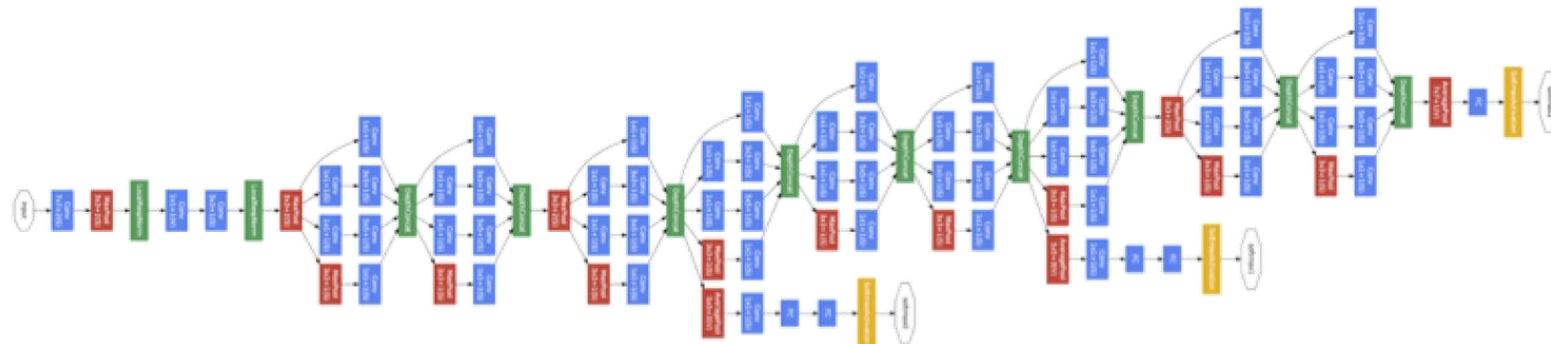
- VGG



- 3x3의 작은 필터를 사용한 Convolution Layer를 연속으로 설계
- Max Pooling을 통해 크기를 크기를 축소
- 마지막에 Fully Connected를 통해 결과 출력
- VGG는 층의 깊이에 따라 VGG16과 VGG19로 구분
- 구성이 간단하여 응용하기 편리함.

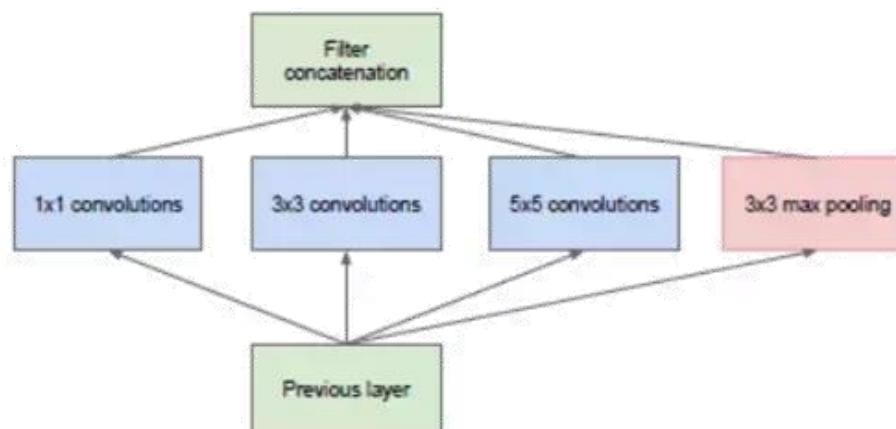
GoogleNet은 2014년에 ILSVRC에서 우승한 CNN이다.

- GoogleNet



- 복잡한 구성의 CNN
- Inception 구조라는 것을 통해 크기가 다른 필터를 여러 개 적용하여 그 결과를 결합함.

Convolution
Pooling
Softmax
Other



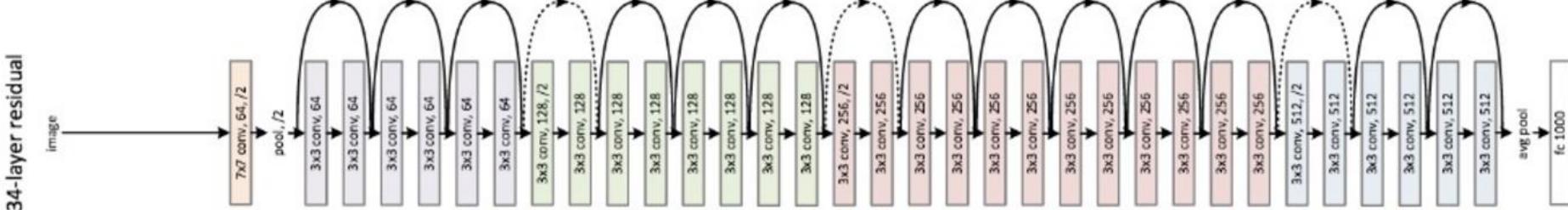
[Inception 구조]

6. 대표적인 CNN

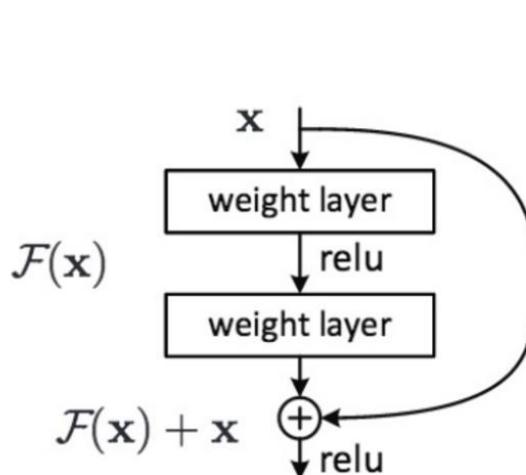
VII. Convolution Neural Network

ResNet은 2015년에 ILSVRC에서 우승한 CNN으로 Microsoft에서 개발했다.

▪ ResNet (Residual Network)



- 152층 CNN (아래 개와 고양이를 아주 잘 구별함.)
- 딥러닝 시 층이 많으면 학습이 잘 안되는 것을 해결하기 위해 Skip Connection을 도입
- Skip Connection 이란 입력 데이터를 Convolution Layer를 건너 뛰어 출력에 바로 더하는 구조
- ResNet은 Convolution Layer를 2개층마다 건너 뛰면서 층을 깊게 함.

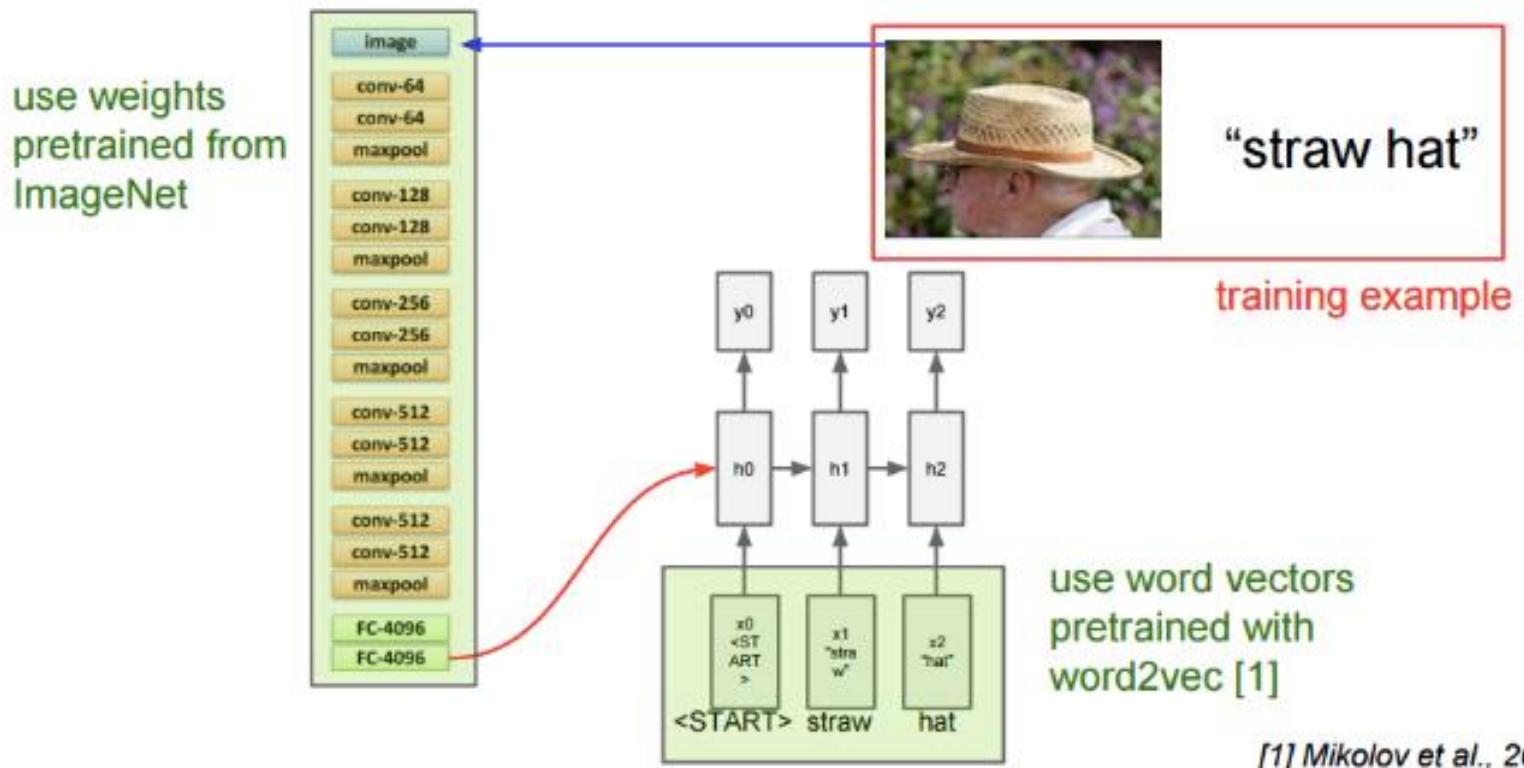


[Skip Connection 구조]



전이 학습 (Transfer Learning) 을 통해 이미 만들어진 CNN을 이용할 수 있다.

- 전이학습 (Transfer Learnin)



- 이미 학습된 가중치 혹은 그 일부를 다른 신경망에 복사한 다음, 그 상태로 재학습을 수행
- 예를 들어, VGG와 구성이 같은 신경망을 준비하고 미리 학습된 가중치를 초기값으로 설정 후, 새로운 데이터 셋으로 재 학습(Fine Tuning)을 실시
- 전이 학습은 보유한 데이터 셋이 적을 때 유용

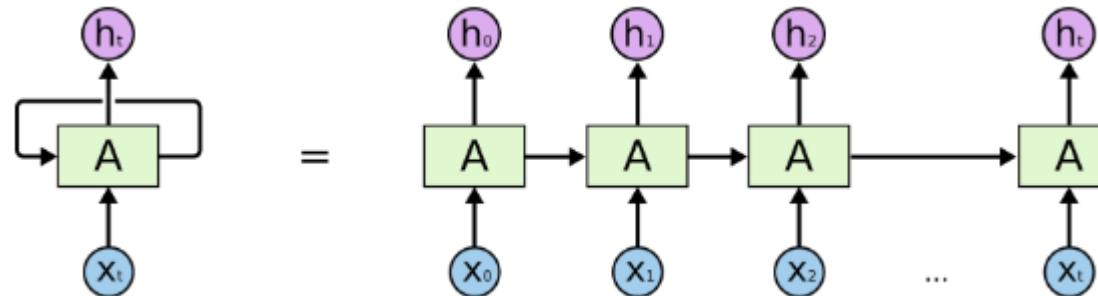
VIII. Deep Learning



1. 신경망의 다른 형태
2. 딥러닝의 다른 주제
3. 딥러닝 Framework
4. 딥러닝 역사
5. 딥러닝 성과
6. 기업별 활용 사례

RNN은 은닉층에서 재귀적 접속이 있는 것으로 입력 데이터가 가변적인 것에 많이 사용된다.

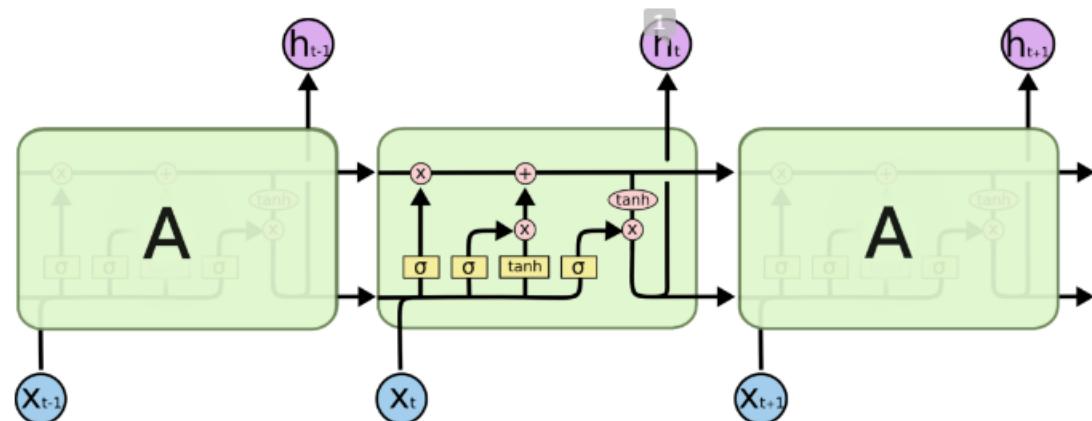
- **RNN (Recurrent Neural Network)**



- RNN은 좌측처럼 단순한 구조지만 학습 시에는 우측처럼 자기 순환하면서 학습한다. (신경망이 깊어진다.)
- RNN 학습은 BPTT (Back Propagation Through Time, 시간을 거슬러 올라가는 역전파) 학습 방법을 적용한다.
- 주로 Time-series data (시계열데이터, 시간의 흐름에 따라 변하는 데이터, 문장 및 음성 데이터)를 적용한다.

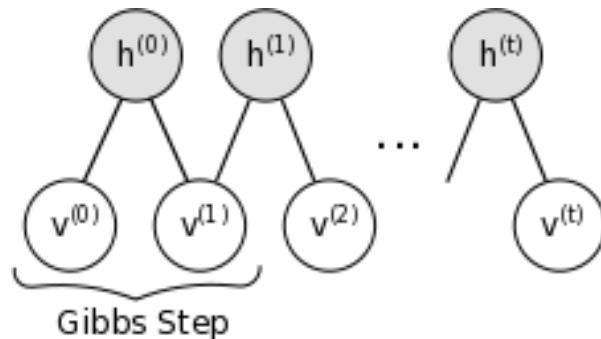
- **LSTM (Long Short Term Memory) 유닛**

- RNN은 인공신경망이 너무 깊어서 오랜 시간 전의 데이터를 잊어버리는 현상 (Vanishing Gradient Problem)이 존재
- Schmidhuber 교수의 Long Short Term Memory (LSTM, 장단기 기억) 유닛을 각 노드마다 배치하여 이 문제를 극복



DBN은 입력층과 출력층으로만 구성되어 있는 RBM (Restricted Boltzmann Machine) 이 쌓은 형태의 신경망이다.

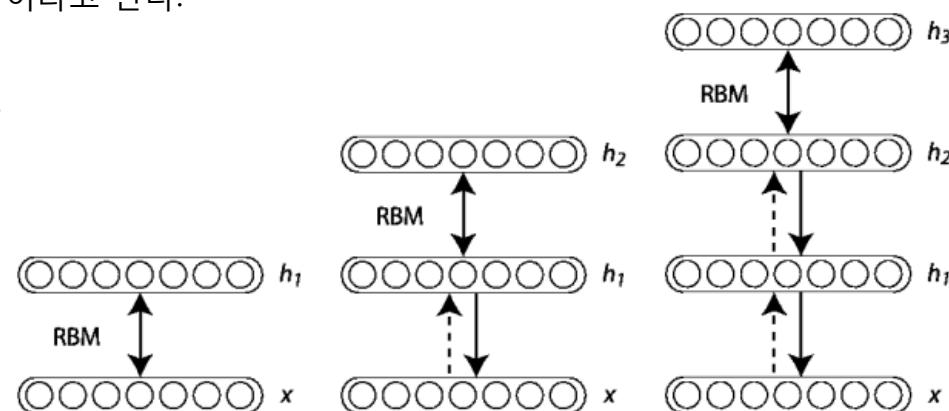
- RBM (Restricted Boltzmann Machine, 제한된 볼츠만 머신)



- RBM은 여러 가지 형태의 레이블된 데이터 또는 레이블되지 않은 데이터를 확률적 방법으로 판별하는 생성모델(Generative Model)이다.
- 생성모델(Generative Model)은 머신러닝에서 사용되는 확률적 **분류** 기법 중 하나이다.
- RBM은 이러한 생성모델을 기반으로 하며 입력층과 은닉층 2개 층으로 구성되어 있는 단층 신경망이다.

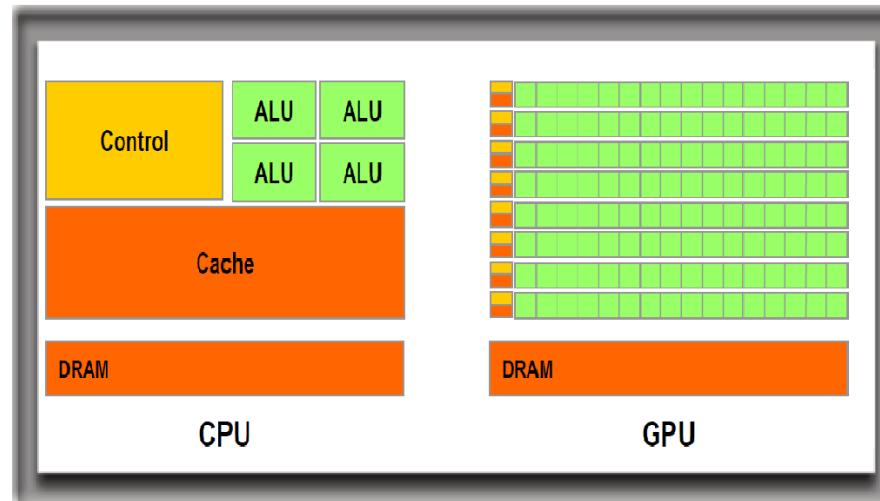
- DBN (Deep Belief Network, 심층 신뢰망)

- DBN은 RBM을 빌딩 블록과 같이 계속 층층이 쌓인 형태로 연결된 신경망이다.
- RBM이 여러 층으로 연결되어 있기 때문에 심층 신뢰망이라고 한다.
- DBN은 RBM을 기반으로 일종의 비지도 학습인 사전학습 (Pre-Training)을 통해 가중치를 어느 정도 보정해 놓고, 마지막 튜닝 과정을 통해 최종 가중치를 계산한다.
- 레이블된 데이터 세트가 충분하지 않은 경우 사용
- 1단계 RBM 학습이 종료되면 1단계 결과를 2단계 RBM의 입력 값으로 이용한다.



GPU를 이용한 병렬 계산은 고속 연산 처리가 가능해 딥러닝 활성화에 기여함.

▪ GPGPU (General-Purpose computing on Graphics Processing Units)



- GPU는 Graphic 처리에 특화되어 있는 H/W
 - ✓ 단순하면서도 높은 병렬 처리가 요구됨
 - ✓ CPU보다 월등히 빠른 연산 능력
- GPGPU
 - ✓ GPU를 그래픽처리 외에 범용적인 용도로 사용할 수 있도록 하자는 데서 출발
 - ✓ 인공신경망에 적용하여 계산량 문제를 크게 개선 시킴

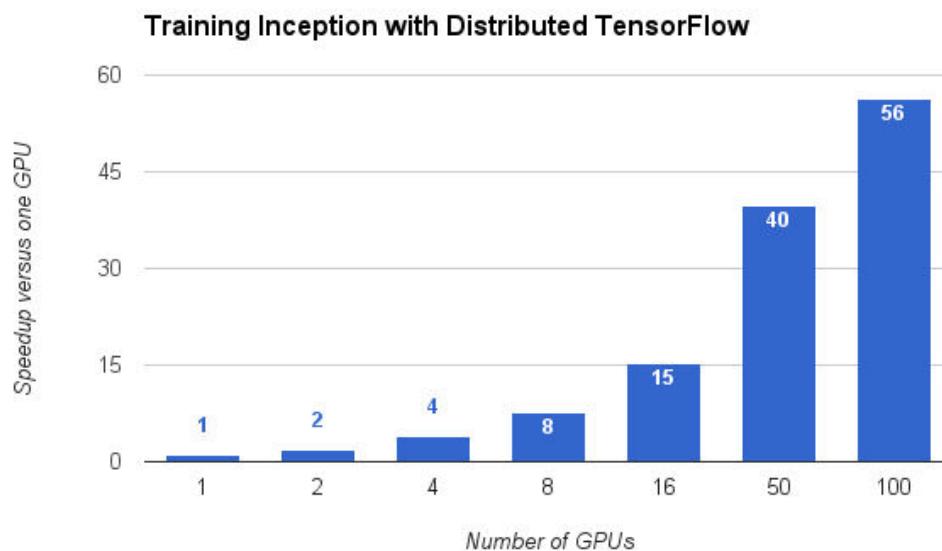
▪ CUDA (Compute Unified Device Architecture)



- 2006년 엔비디아가 발표
- GPU에서 수행하는 병렬 처리 알고리즘을 C 프로그래밍 언어를 비롯한 산업 표준 언어로 작성할 수 있게 함.
 - ✓ 이전에는 OpenGL, Direct3D 같은 것을 이용했어야 함.
 - ✓ 현재 Python, Perl, Java에서도 이용할 수 있도록 Wrapper도 제공
 - ✓ 그래픽 처리 구조를 몰라도 프로그래밍 가능함으로 GPGPU 활성화에 크게 기여
- cuDNN은 CUDA위에서 동작하는 library
- CUDA 개발정보 (CUDA Zone) : <https://developer.nvidia.com/cuda-zone>

딥러닝은 많은 시행착오를 통해 만들어져야 하는데 이를 위해 학습 시간을 줄이기 위한 **분산 학습**을 요구한다.

- **분산 학습의 효과**



- GPU 수가 늘어날 수록 학습 속도가 빨라짐 (100개 GPU 사용 시 56배 빨라짐)
→ 7일 학습을 3시간으로 줄일 수 있음.
- 분산 학습의 문제는 Framework에 위임해서 해결하는 편이 좋음.

- **분산 학습을 처리하기 좋은 Deep Learning Framework**

- Google Tensorflow
- Microsoft CNTK (Computational Network Toolkit)
- 거대한 Data Center의 저지연(low latency)과 고처리량(high throughput)이 필요함.

연산 정밀도를 줄여서 딥러닝 고속화.

- **부동 소수점 계산**

- Computer는 주로 64bit나 32bit 부동소수점을 이용해서 실수를 표현.
- Bit수가 늘수록 계산 오차는 줄어들지만, 그 만큼 계산에 드는 비용과 메모리 사용량이 늘어나고, 버스 대역폭에 부담을 준다.

- **Deep Learning은 높은 정밀도를 요구하지 않는다.**

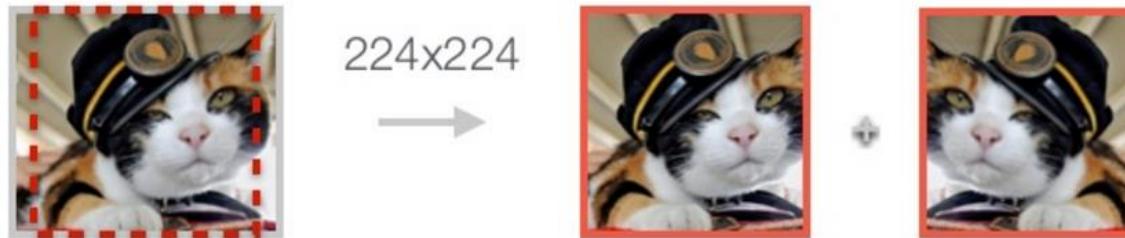
- 신경망 특성 상 입력 데이터에 노이즈가 조금 섞여 있어도 출력 결과가 달라지지 않는다.
- 즉, 가중치를 표현하는 데이터의 정밀도를 낮추어도 학습에 문제가 없다.
- 딥러닝은 16bit 반정밀도 (half-precision)만으로도 문제가 없음.
- nvidia GPU인 파스칼 아키텍처에서 16bit 부동소수점 연산이 지원됨.
- Numpy에는 16bit 부동소수점 type이 존재하지만 Storage만 16bit이고 연산은 16bit 연산이 아님.

딥러닝 모델의 정확도 향상을 위해 Dropout, Weight Decay 이외에도 **Data Augmentation**도 효과적인 방법이다.

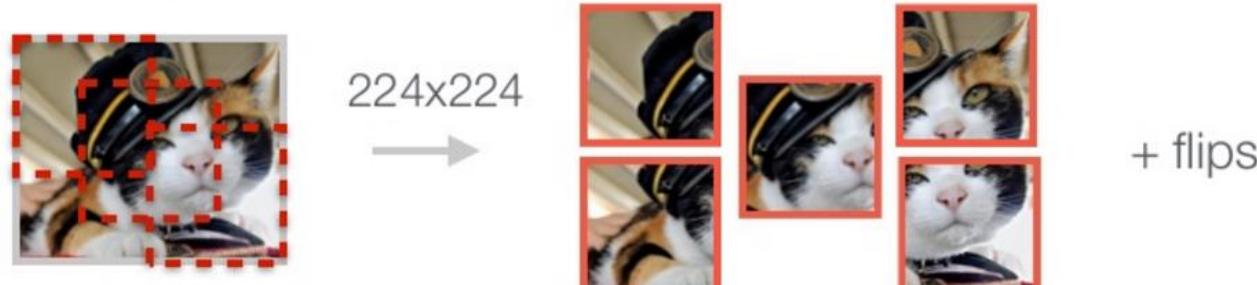
▪ Data 확장 (Data augmentation)

- 훈련 데이터가 몇 개 없을 때 훈련 데이터에 변화를 주어 데이터를 늘리는 기법
- 예를 들어, 입력 이미지를 좌우로 뒤집거나(flip), 회전하거나, 일부를 잘라내거나(crop) 세로로 이동하는 등의 변화를 주는 것.
- 밝기 조정, 확대 축소 등의 스케일 조정 등을 수행할 수도 있다.

Flip augmentation (= 2 images)



Crop+Flip augmentation (= 10 images)



딥러닝 오픈 프레임워크는 다양한 라이브러리와 사전학습 된 알고리즘을 포함하고 있다.

▪ 딥러닝 프레임워크

- 각 프레임워크마다 전문화된 딥러닝 분야와 차별화된 기능을 보유하고 있음.

프레임워크	사이트	개발
Theano	http://deeplearning.net/software/theano/	캐나다 몬트리올 대학 딥러닝 연구소 (MILA)
Caffe	http://caffe.berkeleyvision.org/	캘리포니아 버클리 대학 시각 및 학습 연구 센터 (BVLC)
Torch	http://torch.ch/	뉴욕대학교
TensorFlow	https://www.tensorflow.org/	구글 브레인 프로젝트 팀
DeepLearning4J	https://deepscale4j.org/	스카이마인드 (Startup)
MXNet	http://mxnet.io/	여러 대학과 회사의 협업 (아마존, 바이두, NVIDIA, MS등)



어떤 프레임워크를 선정해야 하는가?

- 컴퓨팅 환경을 고려 : 프로그래밍 언어, 운영체계, 하드웨어 플랫폼 (PC or Cloud), GPU 사용여부
- 사용 목적을 고려 : 간단한 개념증명인가? 수업이나 간단한 프로젝트인가? 실제 서비스 인가?
- 응용 분야를 고려 : 이미지 분석인가? 음성인식인가? 자연어 처리인가? 강화학습인가?



- **프레임워크 개요**

- Python 프로그래밍 언어를 위한 Deep Learning Framework
- 캐나다 몬트리올 대학 딥러닝 연구소 (MILA, Montreal Institute for Learning Algorithm)에서 개발
- 몬트리올 대학 요수아 벤지오 교수가 MILA의 책임자
- 오픈소스 라이선스 (BSD)로 제공

Theano logo, written in a blue serif font.

- **프레임워크 특징**

- Convolution Neural Network 와 Auto-Encoder (노이즈 제거 알고리즘), Deep Belief Network에 최적화된 프레임워크
- Theano Framework에 편리한 API를 보강한 상위 버전의 Framework가 많음.
(Blocks(블럭스), Keras(케라스), Lasange(라자냐), OpenDeep(오픈딥), PyLearn2(파이런2))
- 역전파 알고리즘 같은 인공신경망 구현에 편리
- Convolution 신경망 같은 분야에 유익한 학습자료(Tutorial) 제공

- **프레임워크 단점**

- 불충분한 에러 메시지
- 규모가 큰 모델은 시간이 많이 걸려 적합하지 않음.
- 사전학습(Pre-Training) 모델 지원 불충분
- 프레임워크 구조가 복잡

▪ 프레임워크 개요

- 캘리포니아 버클리 대학교 시작 및 학습 연구 센터 (BVLC, Berkeley Vison and Learning Center에서 개발)
- C/C++/Python을 지원
- 오픈소스 라이선스 (BSD)로 제공
- 산업 부분에서 일부 사용
 - ✓ Pinterest (소셜 미디어 서비스 업체), DeepDream (구글의 이미지 변환 프로젝트)의 기본 프레임워크, 야후에서는 Spark와 Caffe를 연동해서 사용
 - ✓ Caffe 이미지 분류 Demo 확인 : <http://demo.caffe.berkeleyvision.org/>



▪ 프레임워크 특징

- Computer Vision 분야 적용 시 적절 : Convolution 신경망을 포함한 이미지 분석 라이브러리를 제공.
- Caffe Model Zoo (많은 개발자들이 Caffe로 만든 모델을 Share 하는 곳) 제공
- Caffe 커뮤니티 활동이 왕성함.

▪ 프레임워크 단점

- 대학에서 만든 연구개발용 프레임워크로 빠르게 변하는 IT환경에 능동적인 대처가 어려움
- 최신 버전의 GPU 호환성 검증 및 최적화 지원이 늦음.
- 확장형 분산 컴퓨팅 API 지원 부족

▪ 프레임워크 개요

- 뉴욕대에서 개발한 과학계산용 프레임워크
 - : 행렬과 벡터 연산에 최적화된 라이브러리와 각종 수학 및 과학 계산에 필요한 함수 라이브러리를 편리하게 사용
- Lua (루아)라는 스크립트 언어 사용
- C/C++로 개발된 수학 라이브러리를 사용 (성능이 뛰어나다)
- 오픈소스 라이선스 (BSD)로 제공
- 산업계 사용 현황
 - ✓ 페이스북 인공지능 연구소 : 뉴욕대 출신 페이스북 앤 르쿤 교수 가 사용
 - ✓ 구글 딥마인드 : 강화학습에 필요한 다양한 라이브러리를 제공



▪ 프레임워크 특징

- Torch에 적용된 Tensor 라이브러리가 Python 계열의 NumPy와 비슷.
- Lua 스크립트 언어는 JavaScript와 비슷해서 배우기 쉽고 직관적이며 빠르고 쉽게 이해 가능
- 기본 엔진이 C/C++이라서 성능이 우수하다.

▪ 프레임워크 단점

- Lua 스크립트 사용자 그룹이 활성화되지 않아 생태계 조성에 한계가 있다.

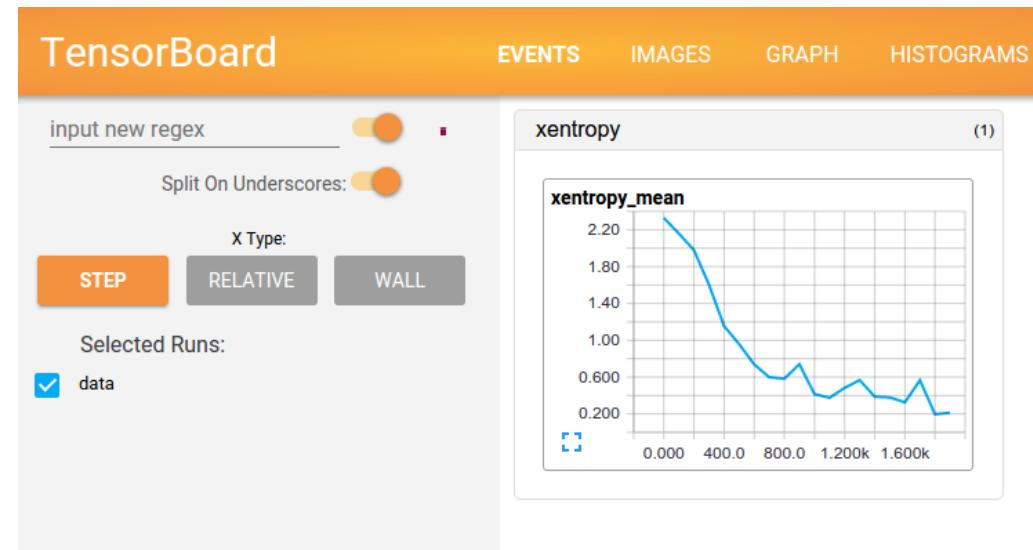
▪ 프레임워크 개요

- 구글이 2015년에 공개한 Deep Learning Framework
- Apache 오픈소스 라이선스 제공
- C/C++로 개발된 엔진 위에 Python을 지원하는 구조 (카페나 시아노와 유사)
- 딥러닝 신경망과 강화학습 알고리즘을 지원
- Java와 Scala 프로그래밍 언어는 지원하지 않는다.



▪ 프레임워크 특징

- 기본 엔진이 C/C++이라서 성능이 우수하다.
- GPU에 최적화되어 있다.
- 분산 병렬컴퓨팅 환경을 지원한다.
(단, 구글 Cloud 서비스인 구글 Cloud Machine Learning에서만 사용가능)
- TensorBoard라는 시각화 도구가 제공된다.



▪ 프레임워크 단점

- 사전학습 모델이 많지 않다.
- 다양한 플랫폼에서 최적화가 덜 되어 있다.

- **프레임워크 개요**

- SkyMind가 개발한 프레임워크
- 자바를 지원하는 프레임워크
- 연구 목적의 다른 프레임워크와는 달리 상용서비스를 위해 설계
- 아파치 2.0 라이선스로 공개
- Skymind는 유료로 전문적인 유지보수, 문제해결, 사용자 교육 지원
- IBM, Accenture, Chevron 등의 Reference 보유



- **프레임워크 특징**

- 과학계산 라이브러리로 DN4J(N-Dimensional Arrays for Java)라는 라이브러리를 직접 개발해서 사용
DN4J는 NumPy(파이썬 기반의 과학계산 라이브러리)보다 약 2배 정도 빠르다.
- 분산 컴퓨팅 환경은 Hadoop과 Spark와 통합해 사용할 수 있다.
- 심층신경망(DBN)과 RBM을 지원하고, Convolution 신경망 모델 지원
- 순환신경망과 장단기 기억법 알고리즘 제공
- <https://deeplearning4j.org/kr-index>

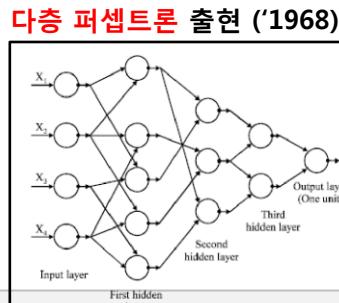
4. 딥러닝 역사

VIII. Deep Learning

딥러닝의 근간이 되는 시작된 인공신경망의 연구는 1950년대부터 시작되었다.



Perceptron(1958)



역전파 이론 ('1981, '1986)

- 폴 워보스가 74년에 적용, 81년에 발표
- 러멜하트, 제프리힌튼, 로날드윌리엄스가 역전파 학습모델 발표 ('1986)
→ 인공신경망 연구 활성화

0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9

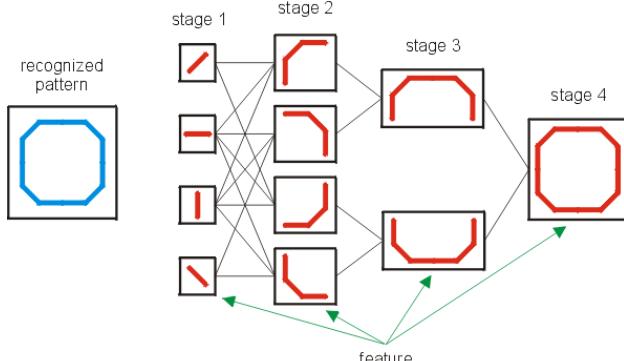
손글씨 문자 인식 연구 ('1998)

- CNN을 활용하여 높은 정밀도의 손글씨 문자 인식 (LeCun 팀)

딥러닝 용어 등장 ('2000)

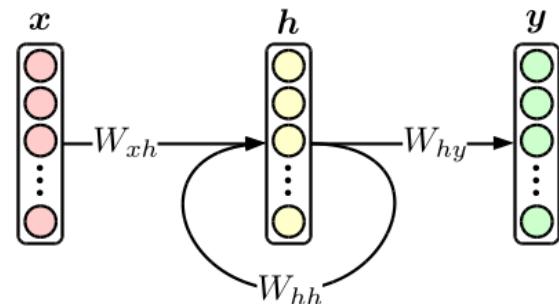
네오코그니트론(neocognitron) 모델 ('1979)

- 후쿠시마 쿠니히코가 발표 (일본)
- 최초로 신경생리학이 융합된 인공신경망
(뇌가 시작정보를 처리하는 구조를 기반으로 모델 작성)
- 현재 Convolution 신경망(CNN)과 매우 유사

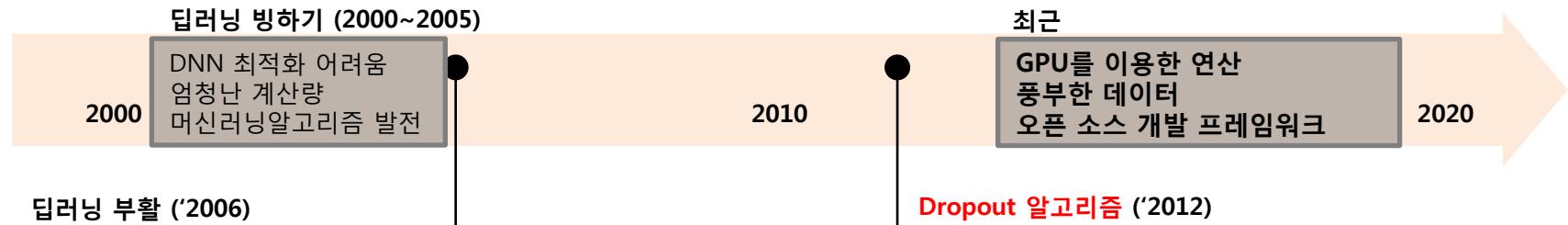


순환신경망 (RNN, Recurrent Neural Network, '1992)

- 장단기 기억법 (Long Short-Term Memory) 개발 ('1991)
: 출력층의 정보를 계속 메모리에 저장하여 역전파에 이용
- 순환신경망 개발 ('1992) : 장단기 기억법을 개선
: 후에 심층신경망 (DBN, Deep Belief Network)에 영향



2000년대 초 빙하기를 거치지만 2006년 부터 딥러닝 문제가 해결되면서 딥러닝이 부활되었다.



딥러닝 부활 ('2006)

- 심층신경망 (DBN)
 - ✓ 제프리 힌튼, 사이먼 오신데로, 이-화이테가 심층신경망 발표
 - ✓ RBM(Restricted Boltzmann Machine)이라는 **사전학습(Pre-Training)** 이용
- Auto-Encoder
 - ✓ 벤자오 팀이 발표, DNN을 최적화하는 **사전학습(Pre-Training)** 방법
 - ✓ 사전 학습을 통해 가중치 초기값을 정하면 DNN이라도 학습이 잘 됨

Dropout 알고리즘 ('2012)

- Sigmoid 형 함수 대신 ReLU 형 함수 사용
(*ReLU : Hinge(힌지)함수 또는 Ramp(램프)함수라고도 함)
- Dropout 알고리즘 : 학습 시 오버피팅(overfitting)을 줄임.
(*Overfitting : 노이즈까지 포함된 학습데이터를 과도하게 학습된 모델이 테스트나 검증 시 정확성을 떨어뜨리는 현상)

▪ Deep Learning은 왜 주목 받는가?

Difficulties		해결 방안
학습	DNN 학습이 잘 안 됨.	Unsupervised Pre-Training를 통한 해결
계산량	학습이 많은 계산이 필요함	H/W의 발전 및 GPU 활용
성능	다른 Machine Learning Algorithm의 높은 성능	Dropout 알고리즘 등으로 Machine Learning 대비 월등한 성능



Faster R-CNN을 이용한 사물 검출 (Object Detection)

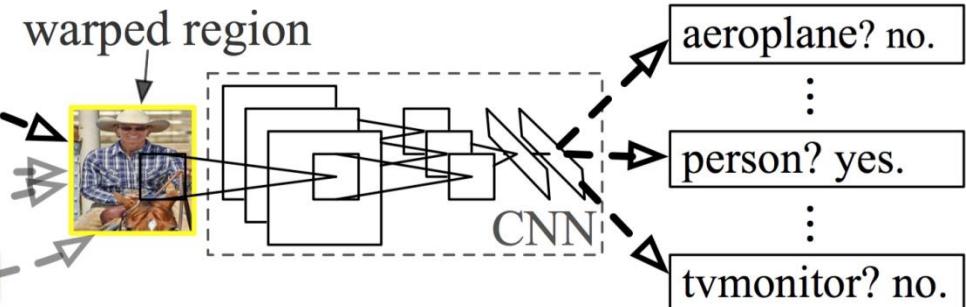
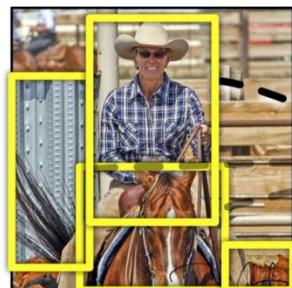
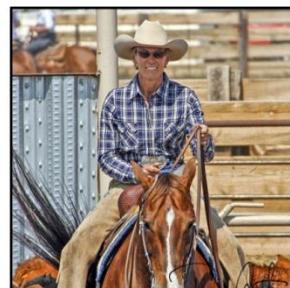
▪ 사물 검출

- 이미지 속에 담긴 사물의 위치와 종류를 알아내는 기술



▪ R-CNN (Regions with Convolution Neural Network)

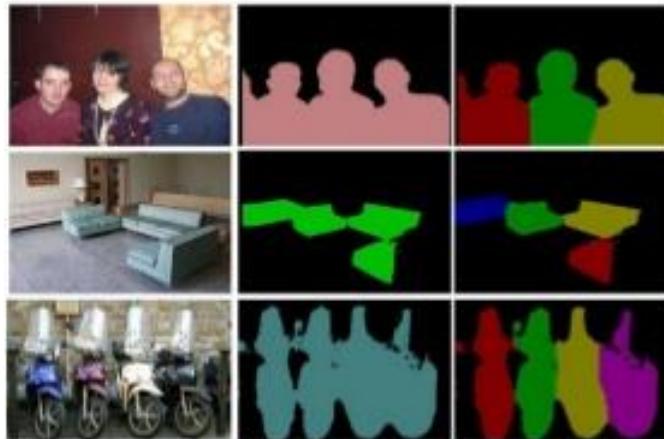
- 이미지 속에 담긴 사물의 위치와 종류를 알아내는 기술
- 입력 이미지에서 후보 영역을 추출(컴퓨터 비전 기술)한 후 CNN을 적용하여 종류를 구분함.
- Faster R-CNN은 후보 영역 추출도 CNN으로 처리한 기술로 더 빨리 처리한다.



Segmentation이란 이미지를 픽셀수준에서 분류하는 문제이다.

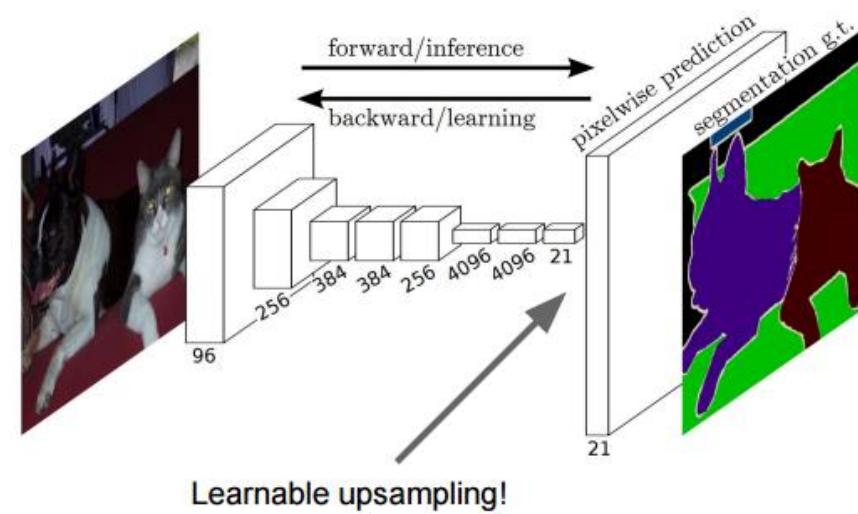
▪ Segmentation

- 이미지 Segmentation을 위해서는 원본이미지와 픽셀단위로 객체마다 채색된 지도 데이터를 사용해 학습한다.



▪ FCN (Fully Convolutional Network)

- Convolution Layer만으로 구성된 Network
- CNN에서는 마지막 출력은 1차원 벡터 였으나
FCN은 마지막에 공간의 크기를 확대 처리함.



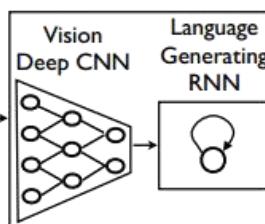
딥러닝을 이용한 이미지 주석 만들기

▪ A neural image caption generator

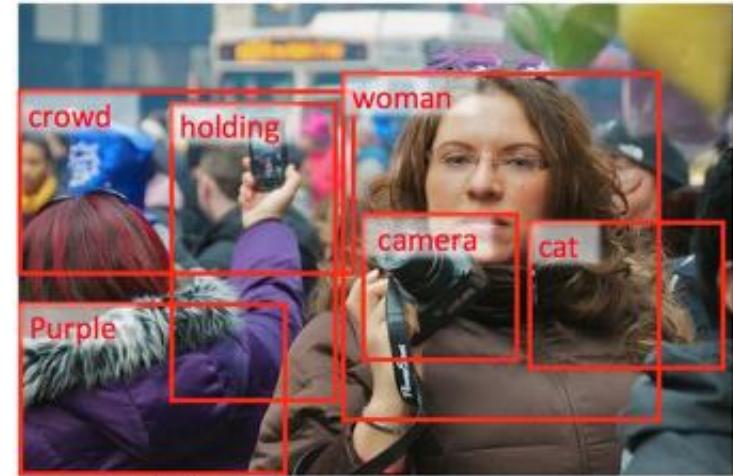
- Google의 Oriol Vinyals팀은 딥러닝을 이용해 이미지 주석을 만들어 내는 기술을 발표.

▪ NIC (Neural Image Caption) 모델

- 처리는 크게 두 단계로 나누어진다.
- 첫 단계 : 이미지를 처리해 추상화 하는 부분(CNN 활용)
- 두 번째 단계 : 이미지 정보를 이용해 문장을 작성하는 부분 (RNN 활용)



A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.



1. detect words woman, crowd, cat, camera, holding, purple
2. generate sentences A purple camera with a woman.
A woman holding a camera in a crowd.
...
A woman holding a cat.
3. re-rank sentences #1 A woman holding a camera in a crowd.

- 사진이나 자연어 같이 여러 종류의 정보를 조합하고 처리하는 것을 multimodal processing 이라고 한다.

딥러닝을 이용한 예술 - 그림 그리기

▪ A neural algorithm of artistic style (2015) - <https://arxiv.org/abs/1508.06576>

- 독일 튜빙겐 대학교 연구팀에서 딥러닝(CNN)을 이용해 유명 화가의 화풍을 학습한 후 이를 이용하여 그림을 그리다.
- 스타일 이미지의 화풍을 흡수하기 위한 '스타일 행렬' 이 있으며 스타일 행렬의 오차를 줄이도록 학습함.



원본 사진 (컨텐츠 이미지)



이중섭 화풍

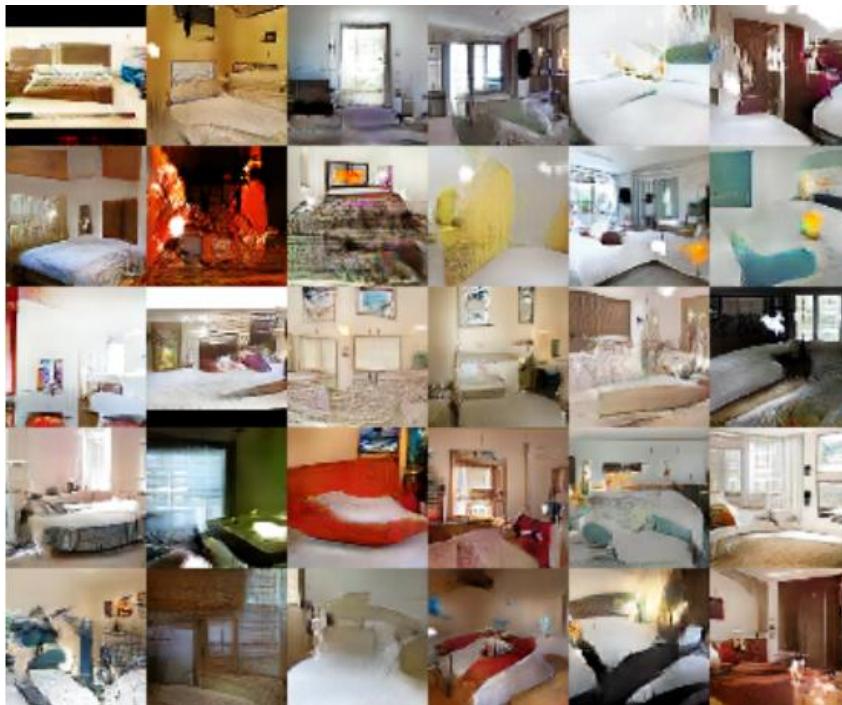
윌리엄 터너
'수송선의 난파'에드바르 몽크
'절규'

스타일 이미지

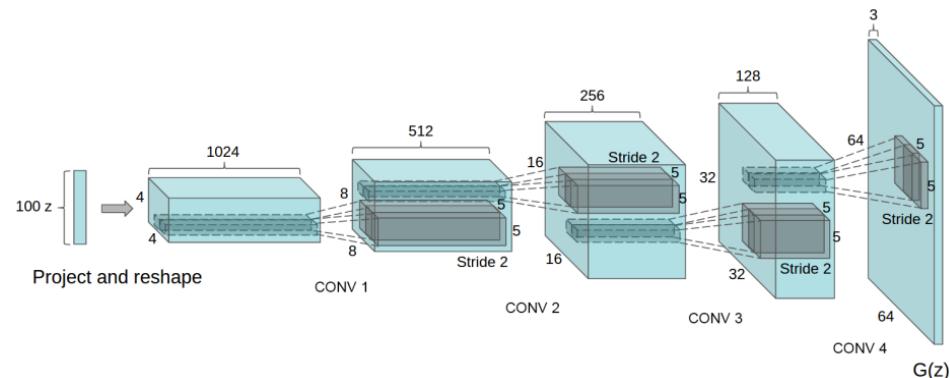
DCGAN 기법으로 이미지 생성하기

▪ DCGAN (Deep Convolutional Generative Adversarial Network)

- 대량의 이미지를 사용하여 학습 한 후, 학습 데이터에는 존재하지 않는 이미지를 생성
- DCGAN은 이미지를 생성하는 과정을 모델화
- DCGAN은 Generator(생성자)와 Discriminator(식별자) 2개의 신경망을 이용
- 생성자는 정교한 가짜 이미지를 생성 기술을 학습하고, 식별자는 생성자가 생성한 이미지가 진짜인지 가짜인지 판별하는 학습을 진행.



[DCGAN이 생성한 이미지]

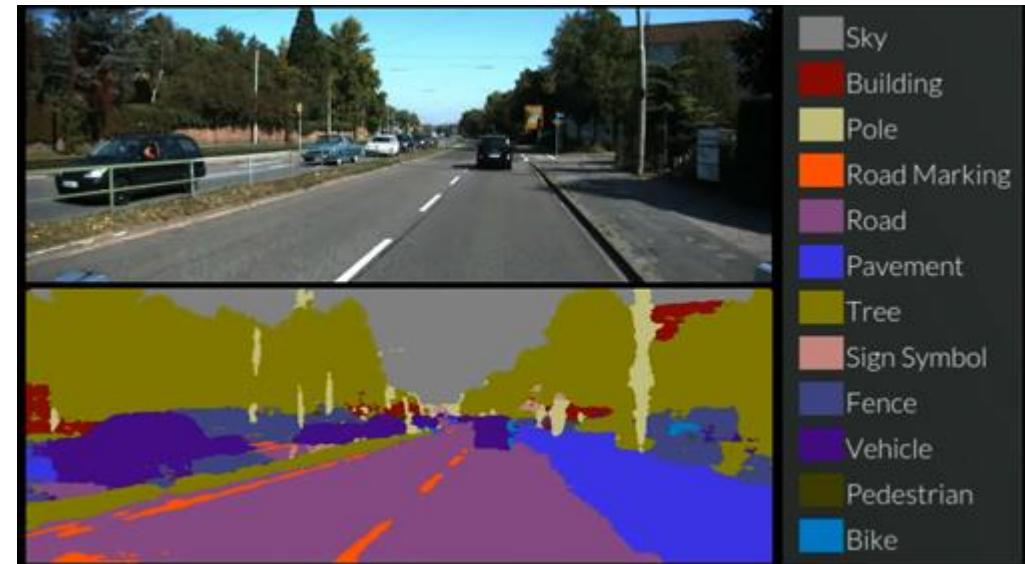


[DCGAN Generator 구조]

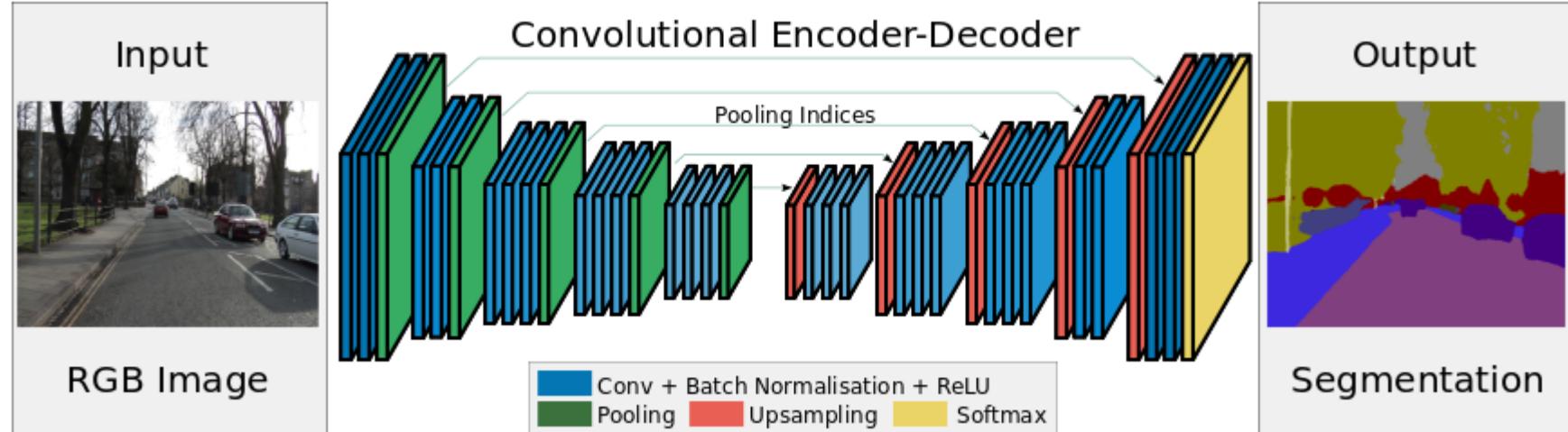
자율 주행을 위한 SegNet

▪ SegNet

- SegNet은 CNN 기반으로 주변 환경을 픽셀 수준으로 분할하여 인식함.



[SegNet Architecture]



딥러닝을 이용한 독순술

▪ Lip Reading Sentences in the Wild - <https://arxiv.org/pdf/1611.05358.pdf>

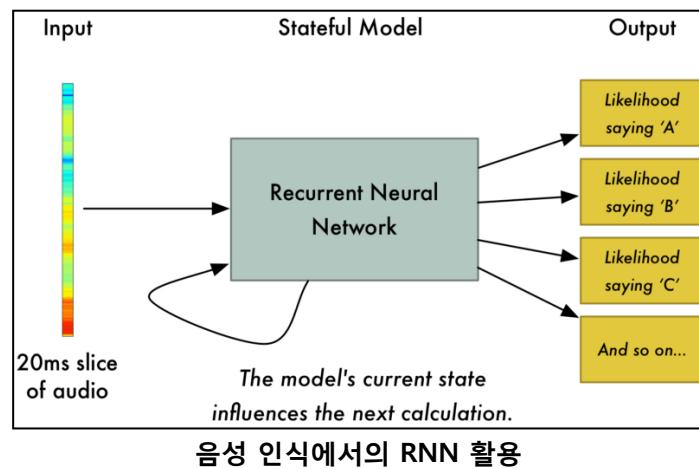
- 구글 DeepMind와 옥스포드대학 연구진이 딥러닝을 활용해 독순술 정확도를 대폭 끌어올린 시스템을 개발
- 훈련 데이터 : BBC 영상 5천 시간 분량으로 11만 8천여 개의 문장과 17,500여 개의 고유명사가 포함
- 테스트 결과 약 절반 수준을 인식 (참고, 독순술 전문가는 12.4% 인식)



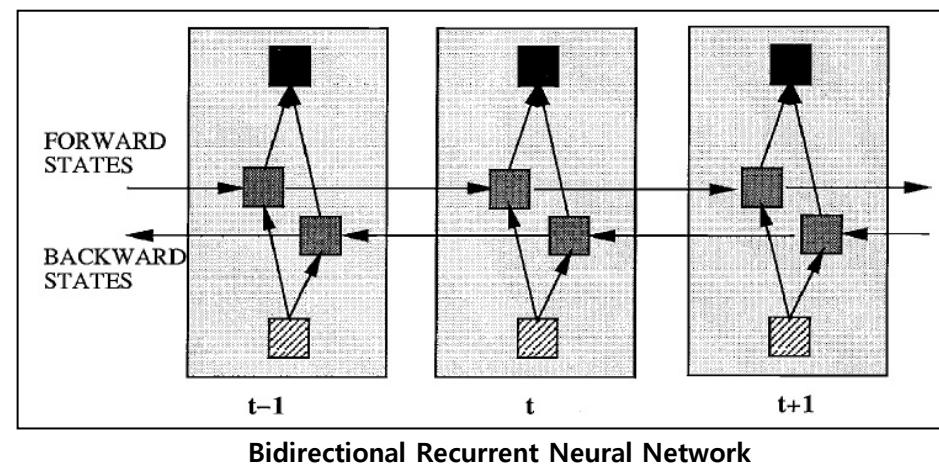
음성 인식 분야의 성과 (Speech to Text)

▪ 음성 인식을 학습

- 학습데이터로 대량의 음성과 음성이 나타내는 텍스트를 쌍을 이용해서 학습 모델을 작성
- Deep Learning 이전에는 Machine Learning 알고리즘인 Hidden Markov Model (은닉 마르코프 모델)을 주로 이용
- Deep Learning을 음성 인식에 이용한 후, Hidden Markov Model 보다 정확성이 대폭 개선됨.
- 음성 인식의 입력 데이터의 길이는 가변적임으로 순환 신경망 (RNN, Recurrent Neural Network)을 주로 이용



음성 인식에서의 RNN 활용



Bidirectional Recurrent Neural Network

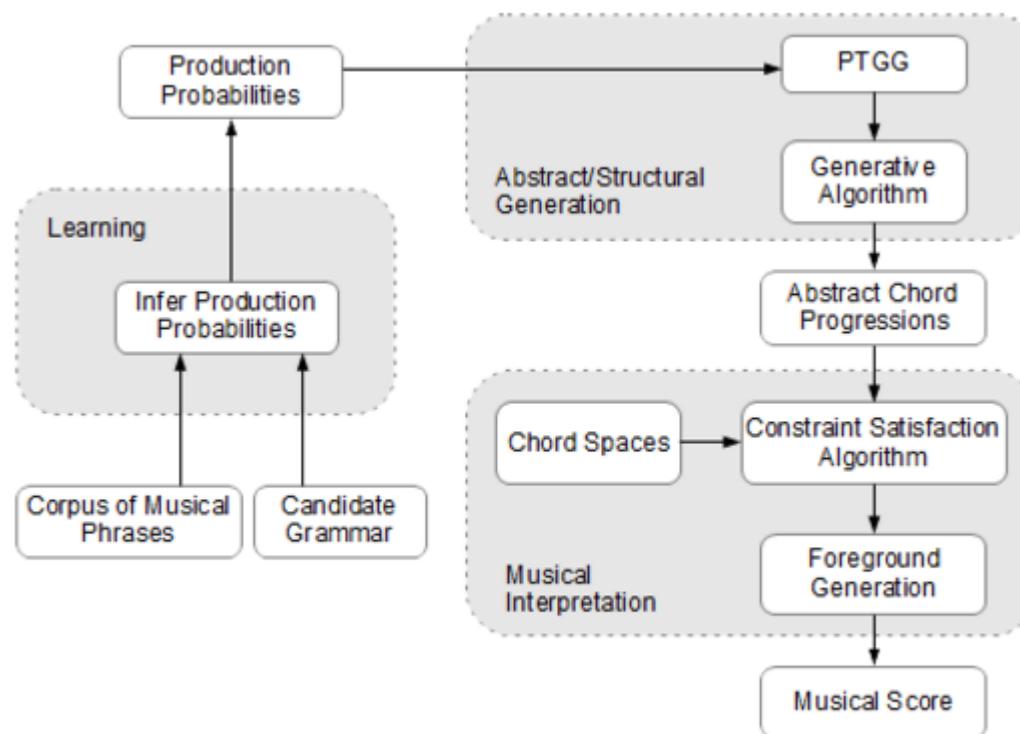
▪ Baidu Deep Speech

- Baidu의 Deep Speech는 은닉층으로 입력되는 값이 앞 층 뿐만 아니라 직후 층에도 입력되는 구조 즉, 양방향에서 입력된다는 의미로 Bidirectional Recurrent Neural Network (양방향 순환 신경망)를 사용했다.
- 사람의 사용하는 말의 문맥이 있어서 이를 활용한다는 의미임.

딥러닝을 이용한 예술 - 음악 작곡

▪ Kulitta - <http://donyaquick.com/kulitta/>

- 예일대 컴퓨터 사이언스 박사 Donya Quick이 개발한 인공지능 작곡
- 특정 장르의 음악에 대해 딥러닝을 통해 스스로 학습하고 비슷한 특성을 가진 창작곡을 내놓는 방식

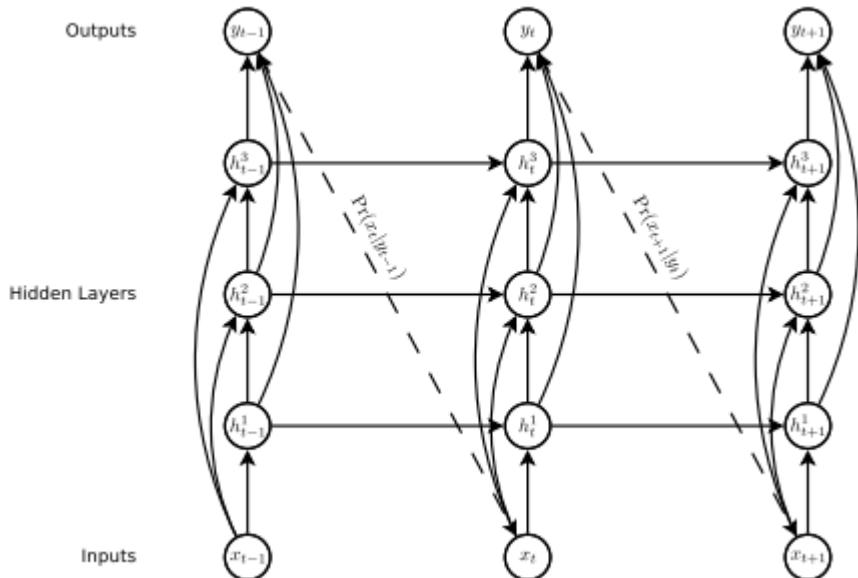


Kulitta's high-level architecture.

딥러닝을 이용한 필기

▪ Generating Sequences With Recurrent Neural Networks - <https://arxiv.org/pdf/1308.0850.pdf>

- 토론토 대학에서 LSTM RNN (Long Short-term Memory recurrent neural network)를 이용하여 손글씨를 만드는 기술 개발
- Demo 사이트 : <https://www.cs.toronto.edu/~graves/handwriting.html>



Deep recurrent neural network prediction architecture

Go digital, Be First !
 Go Digital, Be First !
 Go Digital, Be First !

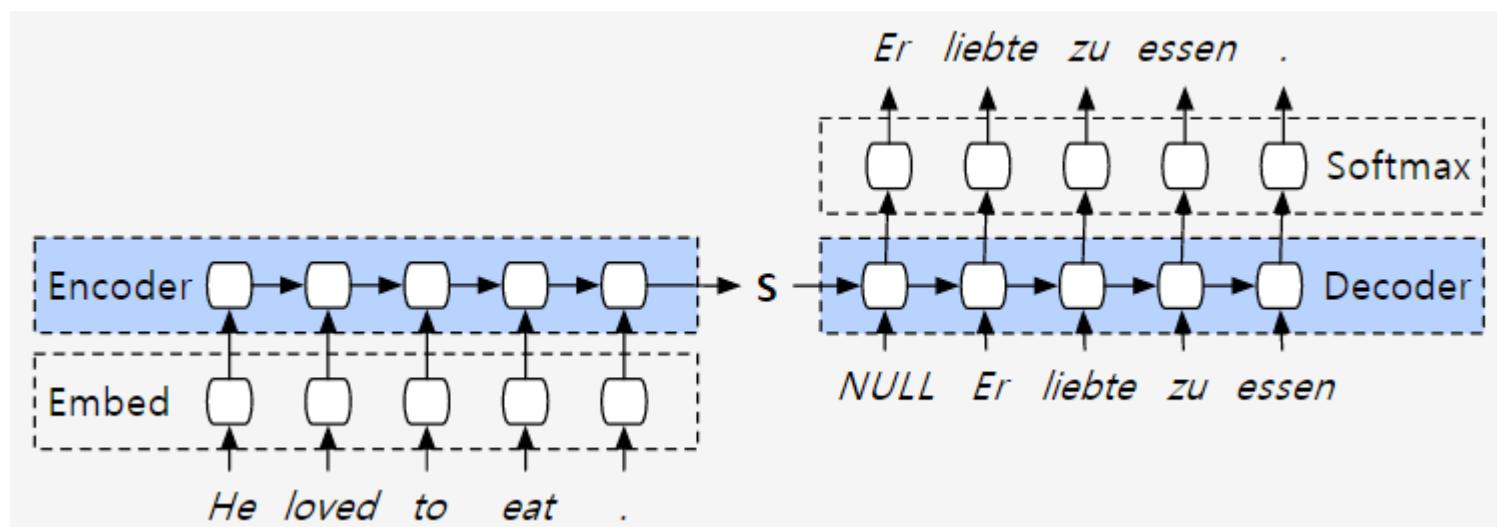
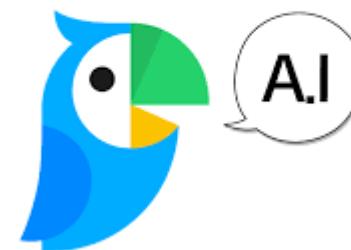
딥러닝을 이용한 번역이 등장하면서 기계 번역이 훨씬 자연스러워짐

▪ Neural Machine Translation (NMT)

- 기존의 통계 기반의 번역(SMT, Statistical Machine Learning)과는 달리 신경망 번역 문장 전체를 인식해서 번역한다.
- NMT는 인공신경망 구조만 잘 결정해 주면 학습이 많을수록 더 자연스러운 번역이 가능해진다.
- 인공신경망을 이용한 번역기

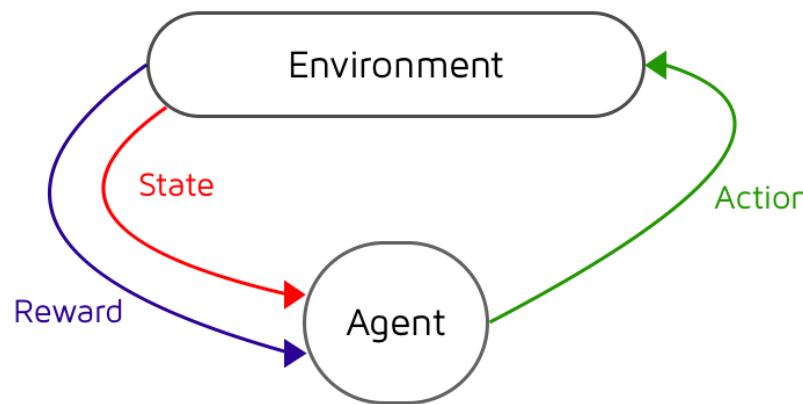


네이버 번역기 파파고,
인공지능으로 발전하다



딥러닝을 사용한 강화학습으로 Deep Q-Network(DQN)이 있다.

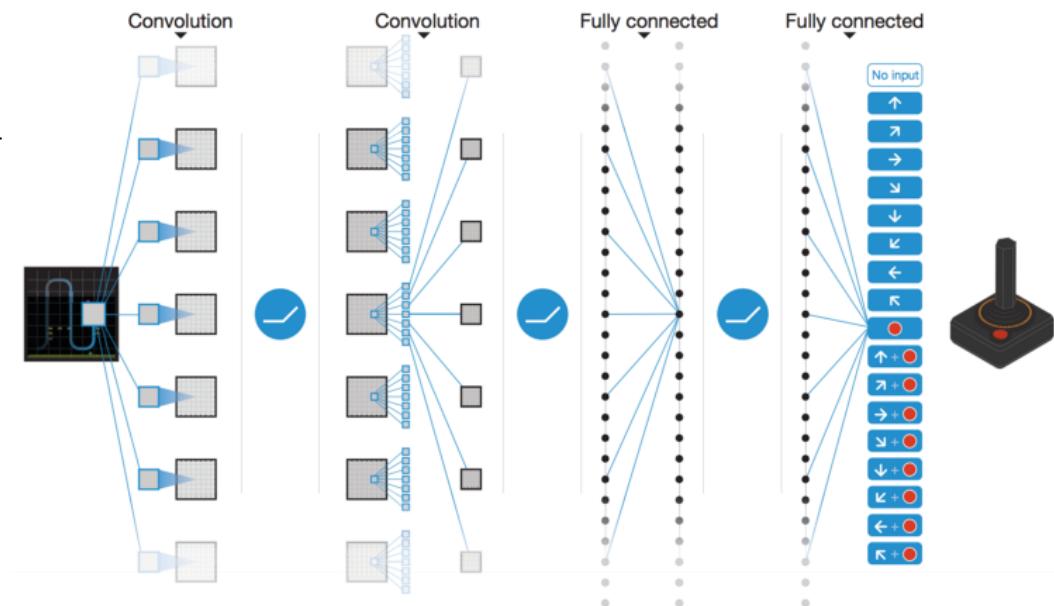
- 강화학습 (Reinforcement Learning)



- 시행착오 과정을 거쳐 학습하기 때문에 사람의 학습방식과 유사
- Agent는 환경으로부터 상태를 관측하고 이에 따른 적절한 행동을 하면 이 행동을 기준으로 환경으로부터 보상을 받는다.
- 관측 – 행동 – 보상의 상호작용을 반복하면서 환경으로부터 얻는 보상을 최대화하는 태스크를 수행하기 위한 일련의 과정.
- 관측 – 행동 – 보상의 과정을 경험(Experience)이라고도 한다.

- Deep Q-Network

- Q학습이라는 강화학습 알고리즘은 최적 행동 가치 함수로 최적 행동을 정함.
- 이 최적 행동 가치 함수를 딥러닝(CNN)으로 비슷하게 흉내 내어 사용



구글 외에도 Netflix에서도 Deep Learning에 엄청난 투자를 하고 있음.

【 Netflix 】

“추천엔진 성능 향상에 사활”



Leaderboard

Showing Test Score. [Click here to show quiz score](#)

Display top leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43

- Netflix 자체개발 영화 추천엔진 CineMatch 성능을 개선하고자 10억원 상금을 걸고 Open Innovation 대회 실시 (2006년 부터)
- 2009년 10.06% 성능 향상 기록으로 우승 팀 탄생
- 최근 딥러닝 기술을 적극 활용
- 현재 Netflix 소비 콘텐츠의 75%는 CineMatch가 추천한 것.

【 Google 】

“Deep Learning 기술 확보 위한 본격적 투자”



Demis Hassabis ('2014)



Geoffrey Hinton ('2013)

- 2011년 딥러닝 기술이 이미지 검색에 활용하는 PoC 진행 → 가능성 확인 후 본격적 이미지 검색 기술 개발 (Google Brain Project)
 - YouTube의 동영상 이미지 검색
- 2013년 Geoffrey Hinton 교수 영입
- 2014년 DeepMind 인수

구글을 따라잡기 위한 마이크로소프트와 바이두.

【 Microsoft】

"Deep Learning의 모범"

- 이미지 인식 데모 프로젝트 'Adam'



- ImageNet의 1,400만개 이미지를 22,000로 분류
- Google Brain 대비 30배 적은 컴퓨터로 50배 이상의 성능향상 두 배의 정확도로 성공
- Cortana : 음성인식 기반의 지능형 개인 도우미 앱 (11개 국가에서 사용 중)
- Skype Translator
- Email 클러터
(스팸차단, 이메일 자동분류)
- 주로 음성인식과 자연어 처리 분야에 집중
- IQ 테스트에 도전 (중국 Microsoft 딥러닝 연구소)



【 Baidu】

"Google에 대한 도전"

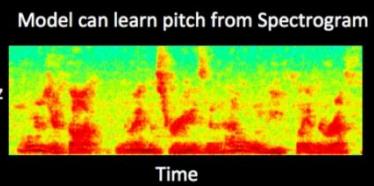
- 2014년 앤드류 응 교수 영입 (Google Brain 프로젝트 리딩)
- Deep Speech ('2014) 발표
 - 순환신경망(RNN) 활용 음성인식 프로그램
 - 소음이 심한 환경에서 소음 제거에 탁월
- Deep Speech 2 ('2015) 발표
 - 중국어 음성 인식 (동일한 음에 다른 의미)
- 앤드류 응 바이두 사임 (2017.3월)



Andrew Ng ('2014)

Mandarin is a *tonal* language

mā má mǎ mà ·ma
妈 麻 马 骂 吗

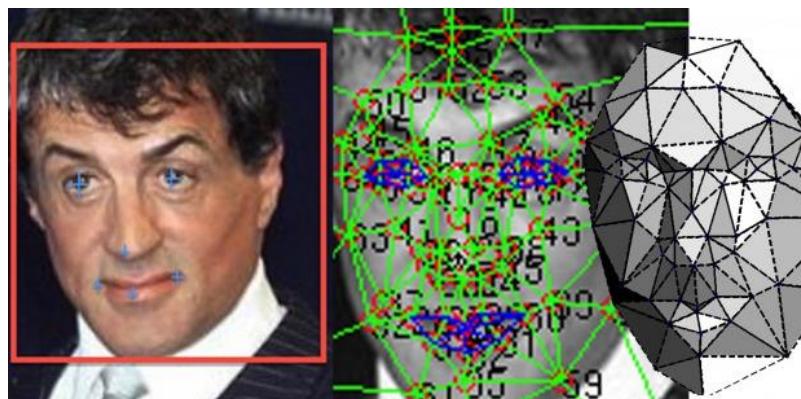


디지털 영토 확장을 위한 딥러닝의 활용 : 페이스북, IBM

【 Facebook 】

“사람을 더 많이 연결하기 위해 딥러닝 활용”

- 얀 르쿤 영입 ('2013)
세계 5대 딥러닝 전문가
- DeepFace 시스템 발표 ('2014)
 - 두 개 인물사진 비교 후 동일인 여부 판단 시스템
 - 정확도가 97.25% (참고로 사람은 97.53%)



- Facebook M ('2015)
 - 개인비서 (식당예약, 스케줄관리, 항공편 예약 등)

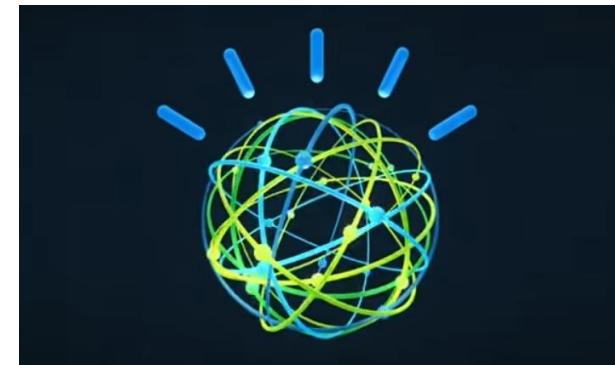
【 IBM 】

“Watson|| Deep Learning 적용”

- 몬트리올 대 교수와 벤지오 교수와 협업 (왓슨에 적용)
 - 세계 5대 딥러닝 전문가
- Jeopardy Show 우승 당시 ('2011) 인공지능 기술
 - 음성인식, 자연어처리
 - 확률기반의 질의 응답
- Watson 상품화
 - Watson에 딥러닝 기술 추가
 - <https://youtu.be/oTOAIIdqNi0>



yoshua bengio ('2014)



한발 늦은 애플과 여러 스타트업들

【 Apple 】

“인공지능에 소극적이었던 애플의 변화”

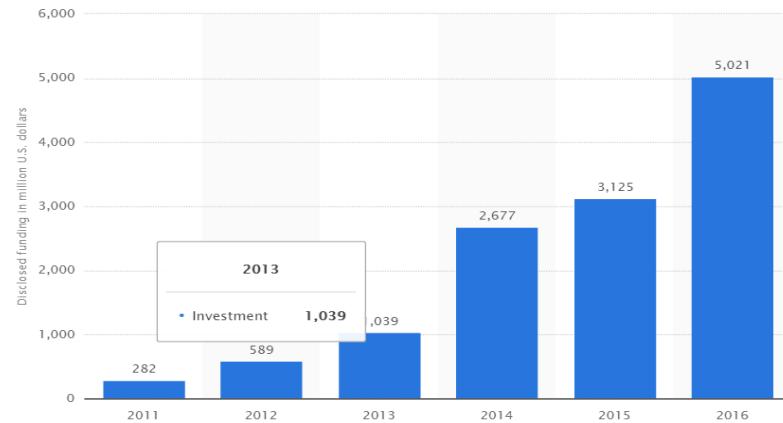
- 스타트업
시리 인수
(‘2010)



- 인수 당시 혁신적인 서비스라고 평가 받음.
- 최근 MS의 코타나나 구글 나우보다 떨어진다는 평가
- 2015년 이후 인공지능 분야 투자 확대
 - Perceptio 인수 : 스마트폰에 저장된 이미지 자동분류
 - VocalIQ 인수 : 자연어 처리 전문 스타트업
 - 인공지능 전문가 영입 지속

【 스타트업 】

“딥러닝 기반 AI Startup들의 확산”

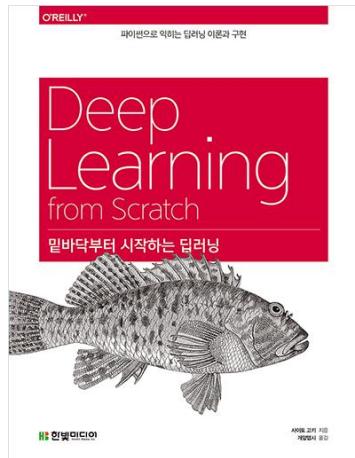


전 세계 인공지능 스타트업 투자규모 변화 (2011~2016)

■ 분야별 스타트업 현황

분야	설명	회사
알고리즘 또는 프레임워크	다양한 산업 및 사용처에서 공통적으로 필요한 알고리즘 및 개발 환경 기술 제공	Skymind, Nervana, MetaMind
컴퓨터 비전	컨볼루션 신경망을 이용한 컴퓨터 비전 정확도 증가	Madbtis, Lookflow
자연어 처리	순환신경망(RNN) 활용한 자연어처리 관련 스타트업	AlchemyAPI, Idibon
응용 서비스	헬스케어 같은 틈새시장 겨냥	Enlitic, StocksNeural

참고문헌





The End

감사합니다.

강병호수석 (SK주식회사 통신개발2팀)

medit74@sk.com