

딥러닝 예제 및 실습 I

수업 4

수업목표

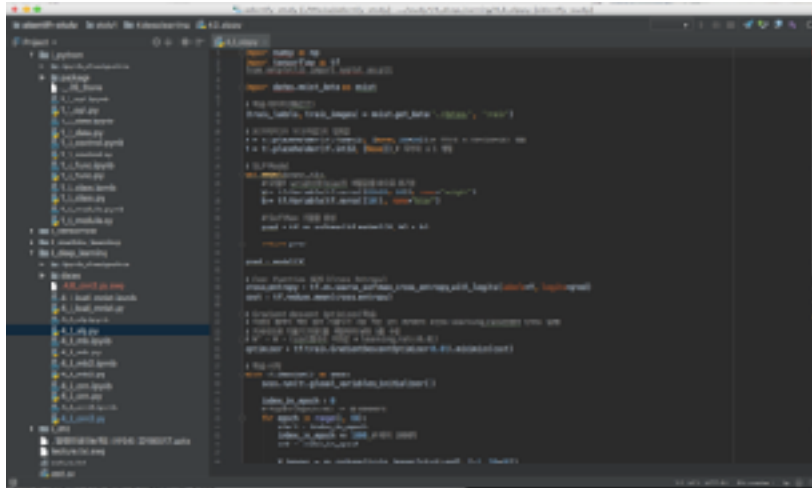
- 파이썬 언어에 대한 이해 & 실습
- 텐서플로우 실습
- 머신러닝 실습 : Linear & Logistic Regression
- 딥러닝 실습 : SLP, MLP, CNN

파이썬에 대한 이해 & 실습

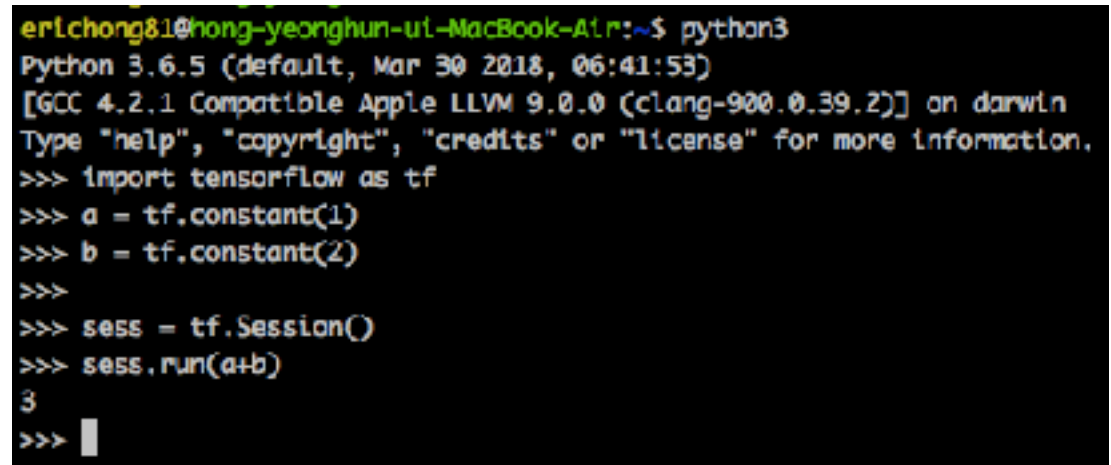
파이썬?

- 1991년, Guido van Rossum이 발표
- 고수준(High Level) 언어
- 다양한 사용자 층을 보유 (학생 ~ 전문가)
- 폭넓은 OS기기 지원과 풍부한 외부 라이브러리
- Python 버전 (Ver. 2 vs Ver. 3)
- Google
- .pyc (bytecode)

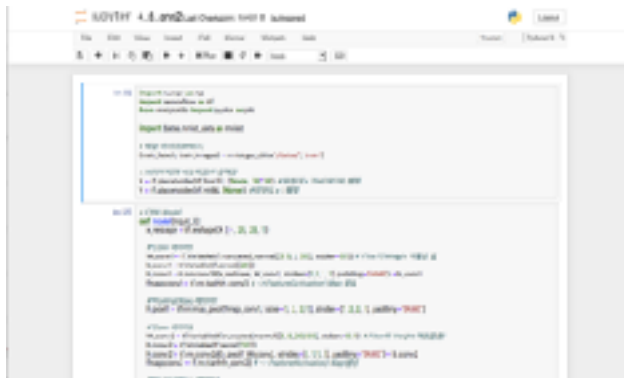
파이썬 - 실행



IDE (pycharm)



CLI - Interactive mode



Jupyter(설치형), Colaboratory(by Google)
Web-based editor



[파일 직접 실행]
ex) python3 파일명.py

<https://colab.research.google.com> (구글)
<https://jupyter.nims.re.kr> (수리과학연구소)

파이썬 - 예제 다운로드

- 다운로드

```
git clone https://github.com/francoisryu/aidentify-study.git
```

파이썬 - 설정

- Python3

- 패키지 관리자 설치

`sudo apt install python3-pip`

- PyCham

- 설정

기존 project 닫는다

라이브러리 경로 확인 (python cli에서 'os' import 후 'print(os.__file__)' 명령으로 확인)

File > Default Settings > Project Interpreter > 설정 > 경로 찾아 추가

(/usr/bin/python3.5인지 확인)

파이썬 라이브러리 관리

- native(pip3)
- anaconda
- virtualenv

- Jupyter 'jupyter notebook'

- 필수 라이브러리 설치

`sudo pip3 install jupyter`

- 커널 설정 & 추가

`python3 -m ipykernel install --user`

파이썬 기본문법 - 변수 & 자료형 & 연산

자료형	변수 = 형(자료) 데이터
문자	<code>text = "hello World"</code> <code>text = 'hello World'</code>
숫자	<code>number = 10</code> <code>octal = 0o10 (8진수)</code> <code>hex = 0xAB (16진수)</code>
실수	<code>float = 1.23</code> <code>float = -1.23</code>
참/거짓	<code>bool = True</code> <code>bool = False</code>

연산의 예)

[source 1-1]

```
text = "hello" + "World" => "helloworld"
text*2    => "helloworldhelloworld"
```

```
14 / 2    => 7 (나누기)
```

```
5 % 2     => 1 (나머지)
```

```
5 // 2    => 3 (소수점 버림)
```

```
1. + 2.2  => 3.2
```

```
2의 10승 => 2**10 => pow(2, 10)
```

```
2E5  => 2 * 지수 10의 5승 => 20000
```

```
1.23E-2 => 1.23 * 지수 10의 -2승의 곱 => 0.0123
```

파이썬은 ;와 같은 문자가 끝에 없습니다.

연산라이브러리 : *math, numpy*

상수는 대체적으로 대문자(TEXT), 변수는 소문자(text)를 사용합니다

파이썬 기본문법 - 데이터 자료형

자료형	변수 = 형(자료) 데이터	연산의 예) [source 1-2]	
튜플 (고정배열)	<pre>tuple = () tuple = (1,) tuple = (1,2,(3,))</pre>	<pre>tuple = (1, 2, '1', '2') tuple[0] => 1 tuple + (3, '3') => (1, 2, '1', '2', 3, '3') tuple[:2] => (1,2)</pre>	<pre>del(tuple[1]) append(tuple[1])</pre>
리스트 (유동배열)	<pre>list = [] list = [1,] list = [1,2,[3,]]</pre>	<pre>list = [1, 2, '1', '2'] list + [3, '3'] => [1, 2, '1', '2', 3, '3'] tuple[:2] => (1,2)</pre>	<pre>del(list[1]) append(list[1])</pre>
사전형	<pre>dic = {'idx1':val1,'idx2':val2}</pre>	<pre>dic = {'1':2, 1:'2'} dic['1'] => 2 dic = {'t': [1,2,3]}</pre>	

index는 0부터 시작

파이썬 기본문법 - 제어문

명칭	변수 = 형(자료) 데이터	제어문의 예) [source 1-3]
조건	if 조건문1: 조건문1이 참일때 수행 elif 조건문2: 조건문2가 참일때 수행 else 조건이 거짓일때 수행	<pre> condition = 1 if condition == 1: print("1") elif condition == 2: print("2") else: print("else") while condition <= 10: print(condition) condition = condition + 1 conditions = [1,2,3] for i in conditions: print(i) </pre>
while순환	while 조건문: 조건이 참인 동안 수행	
for순환	for 변수 in 데이터 자료형: 변수를 사용한 수행 문장	

*indent*로 구문의 모양을 강제
*if*와 비슷한 *switch*는 제공되지 않습니다.

파이썬 기본문법 - 함수, 클래스, 모듈

[source 1-4]

함수 사용 예)

```
def add(a, b):  
    return a+b  
  
print(add(1, 2))  
=> 3
```

[source 1-5]

클래스 사용 예)

```
class operation:  
    def __init__(self):  
        self.result = 0  
    def add(self, a, b):  
        self.result = a+b  
    def div(self, a, b):  
        self.result = a/b  
    def get(self):  
        return self.result  
  
op = operation()  
op.add(1, 2)  
print(op.get())  
=> 3
```

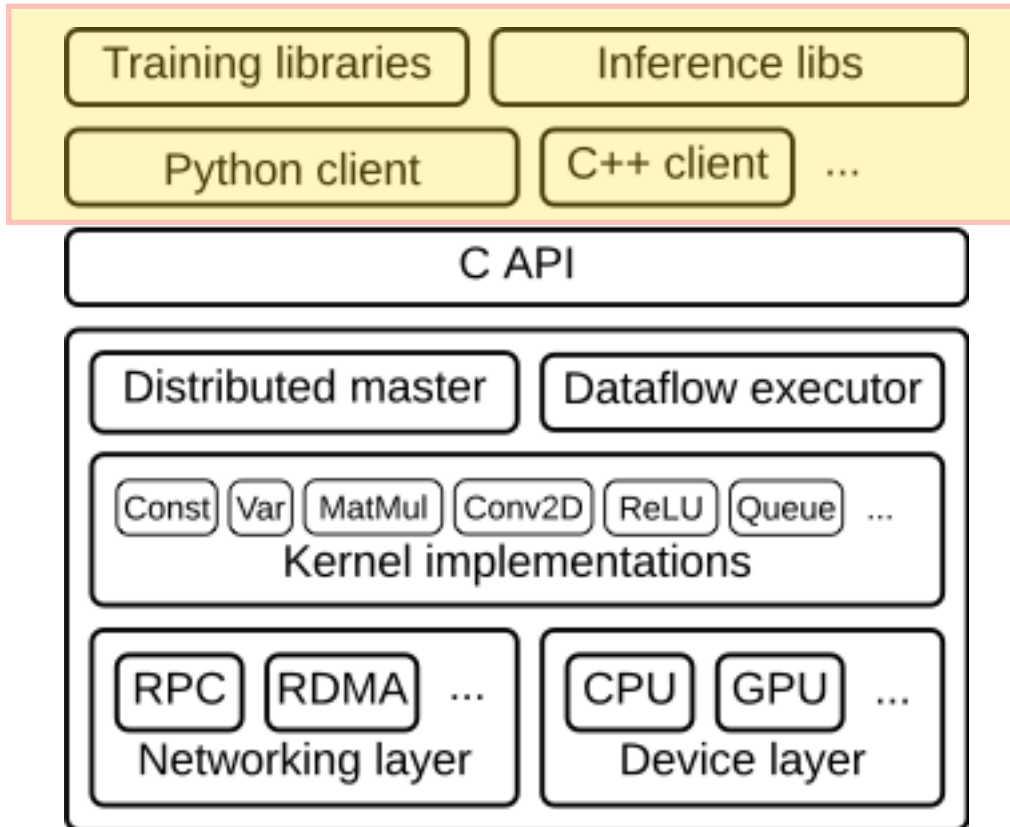
[source 1-6]

모듈 사용 예)

```
import package  
package.mod_func()  
예) import tensorflow  
     tensorflow.Session()  
  
from package import module  
module.func()  
  
import package as module  
module.func()  
예) import tensorflow as tf  
     tf.Session()  
  
from package import *  
func()    => __init__.py 에서 정의 필요
```

클래스에 접근제한자 *public*, *private*, *protected* 제공안함

파이썬 & GPU



CPU vs GPU



OpenCL(Computing Language) 호환

CUDA(Compute Unified Device Architecture)

GPGPU(General-purpose GPU)

Nvidia VGA Driver / CUDA Toolkit

파이썬 & GPU

nvidia-smi : **N**vidia **S**ystem **M**anagement **I**nterface

```
((tf14py27) ubuntu@tensorflow14-cuda8-cudnn6-pic:~/imagesearch$ nvidia-smi
Mon Jan  8 21:14:20 2018
```

NVIDIA-SMI 387.26		Driver Version: 387.26			
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util Compute M.
0	Tesla K80	Off	00000000:00:04.0	Off	0
N/A	40C	P0	76W / 149W	10950MiB / 11439MiB	1% Default

Processes:				GPU Memory Usage
GPU	PID	Type	Process name	
0	1733	G	/usr/lib/xorg/Xorg	14MiB
0	22256	C	python	10921MiB

```
((tf14py27) ubuntu@tensorflow14-cuda8-cudnn6-pic:~/imagesearch$
```

텐서플로우 실습

설치

현재 Tensorflow 버전은 1.8

지원 OS

- 맥OS 10.12.6 (시에라) 이후 버전
- 우분투 16.04 이후 버전
- 윈도우 7 이후 버전

지원 언어

- Python
- Java
- Go
- C

GPU 버전의 경우

- GPU Driver
- CUDA(Compute Unified Device Architecture) Toolkit 9.0
- CuDNN(CUDA Deep Neural Network Library) SDK v7

웹 버전 : <https://colab.research.google.com>

텐서플로우 - 설정

- 텐서플로우 & 관련 라이브러리 설치

```
sudo pip3 install tensorflow matplotlib numpy  
sudo apt install python3-tk
```

- 콘솔에서 import 되는지 확인

```
[코드 실행]  
import tensorflow  
tensorflow.Session()
```

=>

~/.bashrc 제일 하단에 아래 파란 부분 추가

경고 에러가 난다면, '**export TF_CPP_MIN_LOG_LEVEL=2**' 실행하고, ~/.bashrc에 추가해준다

Hello World

[source 2-1]

```
cd aidentify-study/study1/  
jupyter notebook
```

```
import tensorflow as tf
```

```
# 상수를 생성하여 기본 그래프에 추가 (상수 명령값을 출력하는 텐서를 하나 만든다)  
hello = tf.constant('Hello, World!')
```

```
# 세션 시작  
sess = tf.Session()
```

```
# 명령 실행  
print(sess.run(hello))
```

```
sess.close()
```

[source 2-2]

[개선]

```
with tf.Session() as sess:  
    ....print(sess.run(hello))  
  
    ....print(hello.eval())
```

계산 예제

[source 2-3]

```
import tensorflow as tf
```

```
# 상수를 생성하여 기본 그래프에 추가 (상수 명령값을 출력하는 텐서를 하나 만든다)
```

```
x = tf.constant(10)
```

```
y = tf.constant(5)
```

```
# add 노드, 파이썬의 기본 '+ 연산자'에 tf.add(x, y)를 재정의 한 것
```

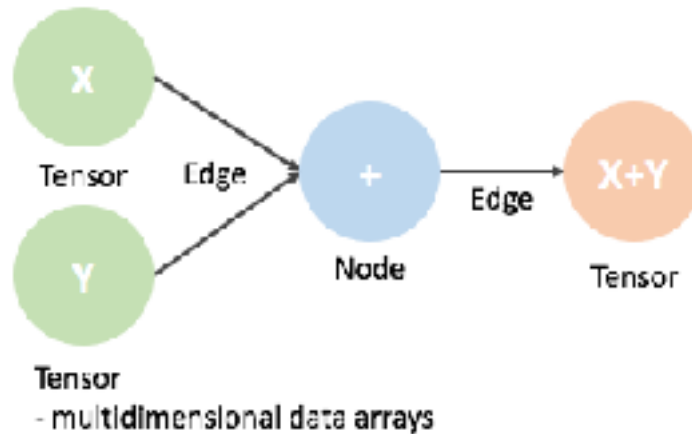
```
x_y = x + y
```

```
# 세션 시작
```

```
with tf.Session() as sess:
```

```
    # 그래프 계산 명령 실행
```

```
    print(x_y.eval())
```



계산 예제 (graph 추가)

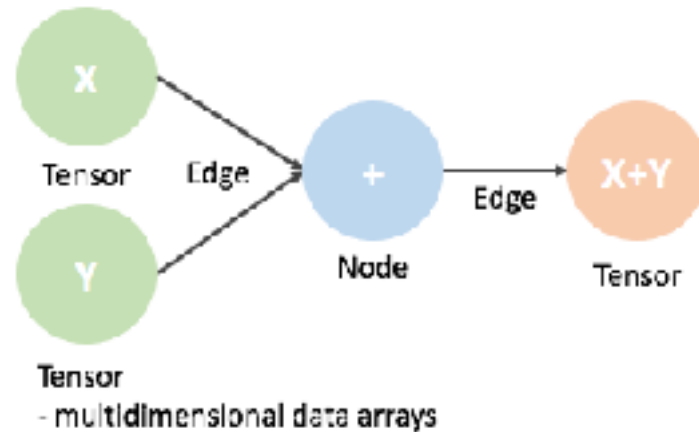
[source 2-4]

```
import tensorflow as tf

#그래프 생성
g = tf.Graph()
with g.as_default() as base_g:
    with base_g.name_scope("g1") as scope:
        # 상수를 생성하여 정의된 그래프에 추가 (상수 명령값을 출력하는 텐서를 하나 만든다)
        x = tf.constant(10, name="X")
        y = tf.constant(5, name="Y")
        x_y = tf.add(x, y, name="ADD")

# 세션 시작
with tf.Session(graph=g) as sess:
    # 그래프 계산 명령 실행
    print(sess.run(x_y))
```

그래프 : base_g



Graph와 세션

- 그래프를 정의하고, 세션으로 한번에 실행
- 그래프는 정의하지 않으면 기본으로 1개 주어짐
- 그래프 내에 텐서가 배치, 세션 실행 시, 연결된 노드가 모두 실행 됨
- 세션에 실행 환경에 대한 설정 가능 (원격 / 로컬)
- 'Session.graph'로 그래프 접근 가능

Data 처리

상수(Constant)

```
x = tf.constant(10, name="x")
```

변수(Variable) - 세션 이후, 초기화 필요함

```
w = tf.Variable(tf.random_normal([3, 2]), name="W")
```

```
w = tf.Variable(2, name="W")
```

```
b = tf.Variable(w.initialized_value() + 3, name="B")
```

```
with tf.Session() as sess:[source 2-5]
...     sess.run(w.initializer)
...     print(sess.run(w))
```

↓ 전체 초기화

```
init = tf.initialize_all_variables()
tf.Session() as sess:
...     sess.run(init)
...     print(sess.run(w)) [source 2-6]
```

치환형(Placeholder)

```
x = tf.placeholder(tf.int32, shape=(2, 2))
y = tf.matmul(x, x) # 행렬 곱 (x * x)
```

[source 2-7]

```
with tf.Session() as sess:
    #print(sess.run(y)) # placeholder 입력 값 할당 전에 출력하면 에러 발생

    matrix = [[1, 2],
               [1, 2]]
    print(sess.run(y, feed_dict={x: matrix})) # 성공적으로 출력
```

데이터 저장 / 로드

저장

[source 2-8]

```
import tensorflow as tf

x = tf.Variable(2)

init_op = tf.initialize_all_variables()

saver = tf.train.Saver()

sess = tf.Session()
sess.run(init_op)

save_path = saver.save(sess, "/tmp/model.ckpt")
```

로드

[source 2-9]

```
import tensorflow as tf

x = tf.Variable(0) # 값을 데이터 변수

saver = tf.train.Saver()

sess = tf.Session()

saver.restore(sess, "/tmp/model.ckpt") # 로드

print(sess.run(x)) # 로드된 값을 출력해본다
```

TensorBoard

텐서플로우에서 데이터의 흐름을 시각화 해주는 툴 (텐서플로우에서 기본으로 제공)
python3 2_10_tensorboard.py

실행) tensorboard --logdir=/tmp/tensorboard

<http://M1302:6006>

[source 2-10]

```
with tf.name_scope('op'):
    a = tf.constant(2, name="a")
    b = tf.constant(1, name="b")

with tf.name_scope('input'):
    x = tf.placeholder(tf.int32, name="X")
    y = tf.Variable(0, tf.int32, name="Y")

y = a * x + b

tf.summary.scalar('x', x)
tf.summary.scalar('y', y)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # TensorBoard 그래프용 로그 기록
    writer = tf.summary.FileWriter("/tmp/tensorboard", sess.graph)

    # summary 정보를 파일로 저장
    merged = tf.summary.merge_all()

    for i in range(10):
        summary, t = sess.run([merged, y], feed_dict={x: i})
        writer.add_summary(summary, i)

writer.close()
```



머신러닝 실습

- Linear Regression
- Logistic Regression

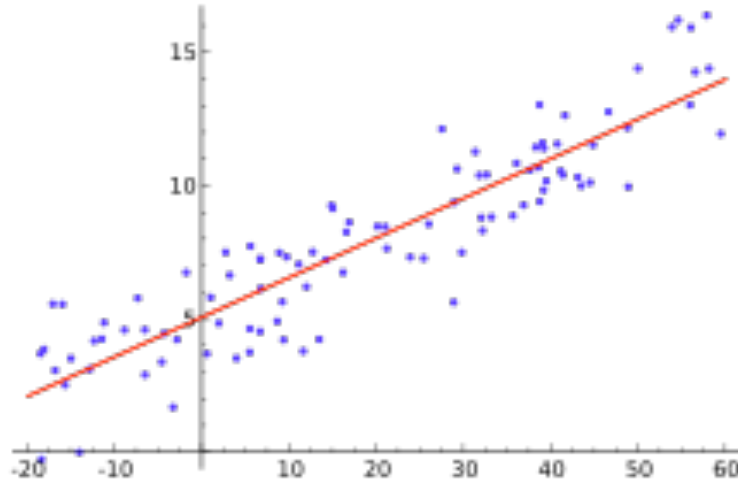
텐서플로우 툴 잡기

1. 데이터 준비(배열 값, 이미지..)
2. 입력 값 셋팅(데이터가 많은 경우 placeholders 사용)
3. 모델 셋업 (linear, logistics, multilayer...)
4. cost & 최적화 설계
5. 세션 시작(with 변수 초기화)
6. 학습 (배치와 루프 설정 -> optimizer실행)
7. 테스트(임의의 샘플 데이터로 cost를 구함)
8. 학습과 테스트 비교(필요하면 시각화 함)
9. 값 예측

해당 내용은 Single 그래프의 경우로, multi-graph의 경우에는 구조를 나누어서 세션에서 결과를 조합

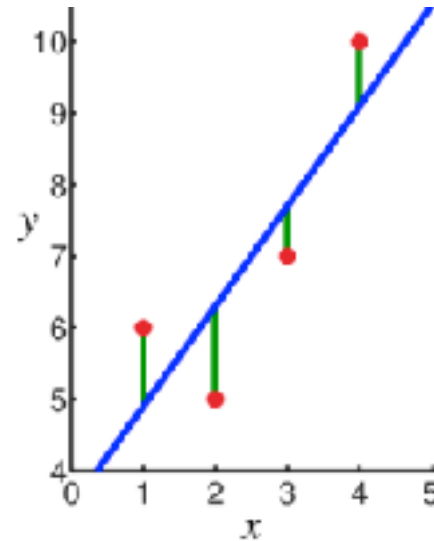
```
result = sess.run(sess.graph.get_tensor_by_name( scope명 ))
```

Linear Regression



$$Y = f(X) + \epsilon.$$

Y = Weight * X + Bias



최소제곱법 $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$

실측(데이터)과 예상(함수) 간의 거리를 제공하여 평균한다

[source 3-1] : linear Regression 기본

[source 3-2] : 파라미터 추가

[source 3-3] : 데이터 구조 변경(행렬) & 데이터 파일읽기

```
import tensorflow as tf
import matplotlib.pyplot as plt
```

[source 3-1]

1) 데이터 준비

```
train_X = [2.52, 4.22, 9.65, 4.22, 7.6, 4.2, 2.2, 3.5]
train_Y = [3.28, 7.22, 17.44, 8.22, 13.11, 9.2, 5.3, 6.3]
```

2) 입력 값 셋팅

```
X = tf.placeholder("float")
Y = tf.placeholder("float")
```

```
W = tf.Variable(0., name="weight")
b = tf.Variable(0., name="bias")
```

2) 모델 셋업(linear)

```
pred = W*X + b
```

Cost & 최적화 설계

```
cost = tf.reduce_mean(tf.square(pred-Y))
optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

Linear Regression

[source 3-1]

5) 세션 시작

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())
```

6) 학습 (epoch:3000)

```
for epoch in range(3000):  
    sess.run(optimizer, feed_dict={X: train_X, Y:train_Y})
```

cost계산

```
training_cost = sess.run(cost, feed_dict={X: train_X, Y:train_Y})  
training_W = sess.run(W)  
training_b = sess.run(b)  
print(epoch, training_cost, [training_W, training_b])
```

```
print("학습완료! (cost : " + str(training_cost) + ")")
```

Linear Regression

[source 3-1]

7) 테스트

```
test_X = [6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1]
test_Y = [1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03]
```

```
testing_cost = sess.run(cost, feed_dict={X: test_X, Y: test_Y})
print("테스트 완료! (cost : " + str(testing_cost) + ")")
```

8) 학습과 테스트 cost비교(절대값)

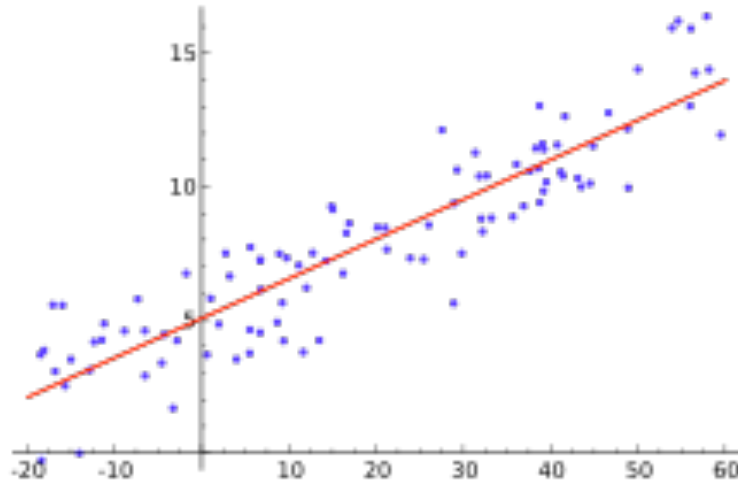
```
print("테스트와 학습의 cost차이 : ", abs(training_cost - testing_cost))
```

[화면에 찍어보는 기능 추가 가능]

9) 값 예측

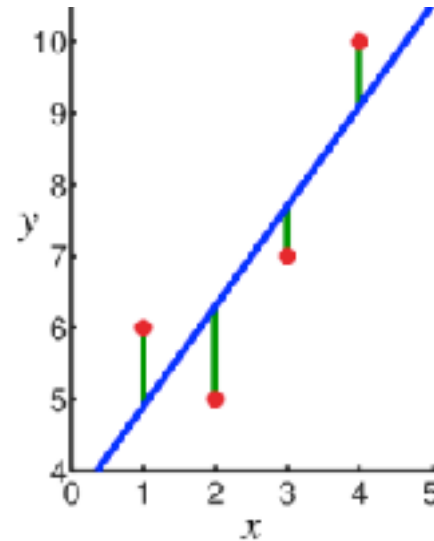
```
print("x가 3.3일때 : " + str(sess.run(pred, feed_dict={X: 3.3})))
```

Linear Regression - 다중 인자



$$Y = f(X) + \epsilon.$$

Y = Weight * X + Bias



최소제곱법 $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$

실측(데이터)과 예상(함수) 간의 거리를 제공하여 평균한다

[source 3-1] : linear Regression 기본

[source 3-2] : 파라미터 추가

[source 3-3] : 데이터 구조 변경(행렬) & 데이터 파일읽기

Linear Regression - 다중 인자

1) 데이터 준비

[source 3-2]

```
train_X = [3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779]  
train_X2 = [2.3, 3.4, 4.5, 5.71, 5.93, 3.168, 8.779]  
train_Y = [1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366]
```

2) 입력 값 셋팅

```
X = tf.placeholder("float")  
X2 = tf.placeholder("float")  
Y = tf.placeholder("float")
```

```
W = tf.Variable(0., name="weight")  
W2 = tf.Variable(0., name="weight")  
b = tf.Variable(0., name="bias")
```

2) 모델 셋업(linear)

```
pred = W*X + W2*X2 + b
```

Cost & 최적화 설계

```
cost = tf.reduce_mean(tf.square(pred-Y))  
optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

Linear Regression - 다중 인자

[source 3-2]

5) 세션 시작

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())
```

6) 학습 (epoch:3000)

```
for epoch in range(3000):  
    sess.run(optimizer, feed_dict={X: train_X, X2: train_X2, Y:train_Y})
```

cost계산

```
training_cost = sess.run(cost, feed_dict={X: train_X, X2: train_X2, Y:train_Y})
```

```
training_W = sess.run(W)
```

```
training_W2 = sess.run(W2)
```

```
training_b = sess.run(b)
```

```
print(epoch, training_cost, [training_W, training_W2, training_b])
```

```
print("학습완료! (cost : " + str(training_cost) + ")")
```


Linear Regression - 다중 인자

[source 3-2]

7) 테스트

```
test_X = [6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1]
test_X2 = [5.83, 3.668, 7.9, 6.91, 4.7, 7.7, 2.1, 1.1]
test_Y = [1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03]
```

```
testing_cost = sess.run(cost, feed_dict={X: test_X, X2: test_X2, Y: test_Y})
print("테스트 완료! (cost : " + str(testing_cost) + ")")
```

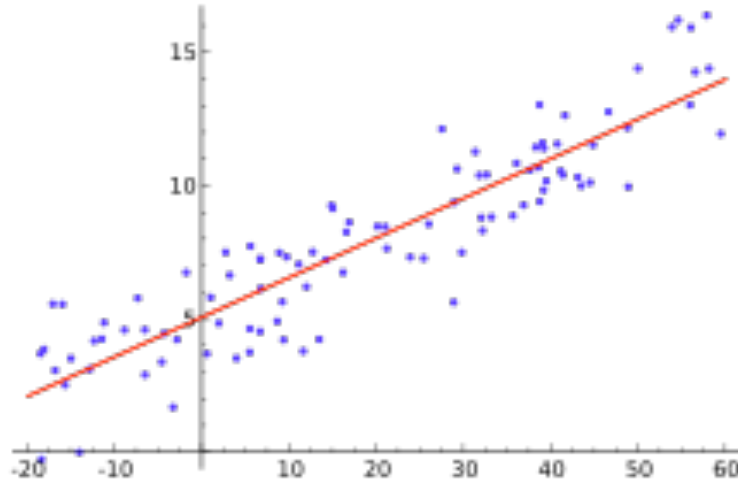
8) 학습과 테스트 cost비교(절대값)

```
print("테스트와 학습의 cost차이 : ", abs(training_cost - testing_cost))
# [화면에 찍어보는 기능 추가 가능]
```

9) 값 예측

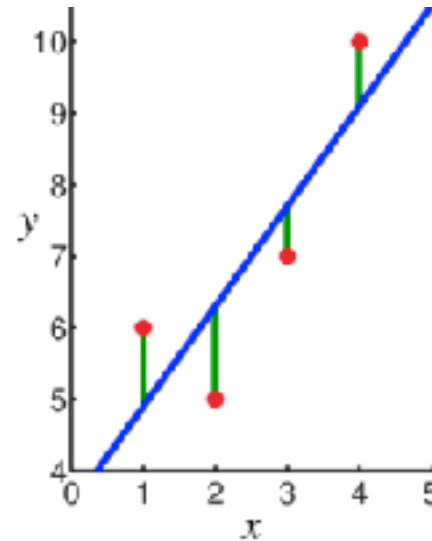
```
print("x가 3.3일때 : " + str(sess.run(pred, feed_dict={X: 3.3, X2:3.6})))
```

Linear Regression



$$Y = f(X) + \epsilon.$$

Y = Weight * X + Bias



최소제곱법 $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$

실측(데이터)과 예상(함수) 간의 거리를 제공하여 평균한다

[source 3-1] : linear Regression 기본

[source 3-2] : 파라미터 추가

[source 3-3] : 데이터 구조 변경(행렬) & 데이터 파일읽기

Linear Regression - matrix

[source 3-3]

1) 데이터 준비

```
# train_X = np.loadtxt('./datas/3_3_linear_matrix.csv', unpack=True, dtype='float')
```

```
train_X = [[3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779],  
           [2.3, 3.4, 4.5, 5.71, 5.93, 3.168, 8.779],  
           [1.,1.,1.,1.,1.,1.,1.]]
```

```
train_Y = [1.7, 2.76, 2.09, 3.19, 1.694, 1.573, 3.366]
```

2) 입력 값 셋팅

```
X = tf.placeholder("float")
```

```
Y = tf.placeholder("float")
```

```
W = tf.Variable(tf.zeros([1, 3], dtype=tf.float32), name="weight")
```

2) 모델 셋업(linear)

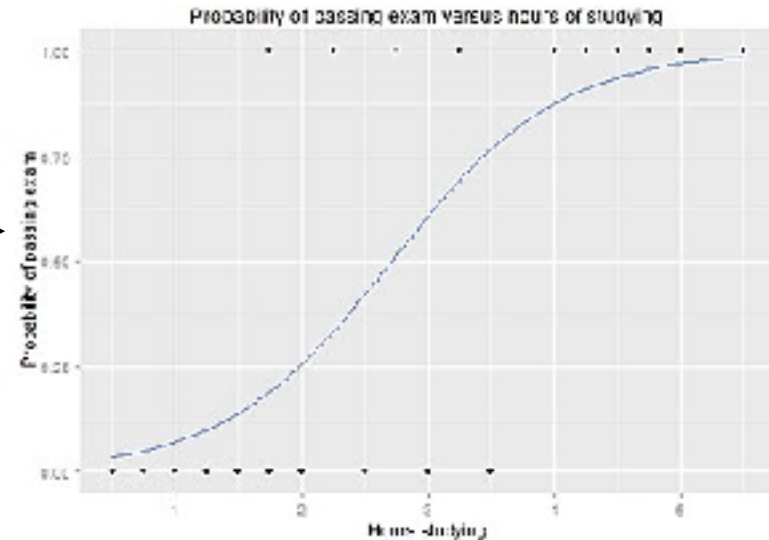
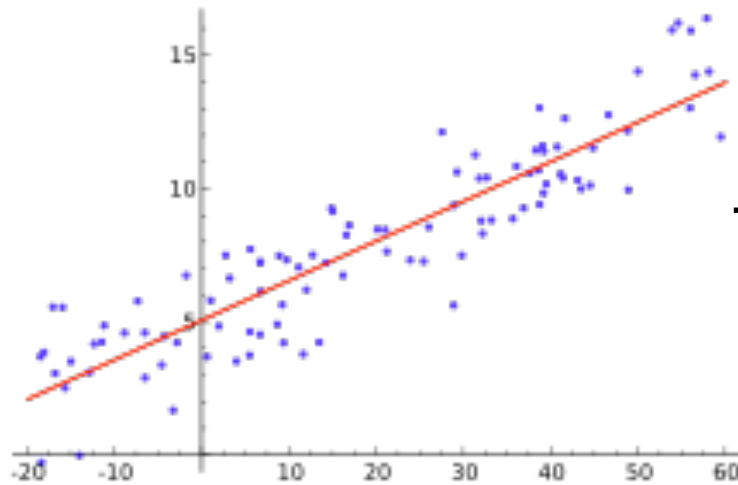
```
pred = tf.matmul(W, X)
```

Cost & 최적화 설계

```
cost = tf.reduce_mean(tf.square(pred-Y))
```

```
optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

Logistic Regression



sigmoid function (=logistic function)

$$g(x) = \frac{1}{1 + e^{-x}}$$

Cross Entropy

$$H(p, q) = - \sum_x p(x) \log q(x).$$



$$Cost(y, h) = \begin{cases} -\log(h), & \text{if } y = 1 \\ -\log(1 - h), & \text{if } y = 0 \end{cases}$$

$$Cost(y, h) = -y \log(h) - (1 - y) \log(1 - h)$$

[source 3-4] : 모델과 cost function 변경

Logistic Regression

1) 데이터 준비

[source 3-4]

```
train_X = [3.3, 4.4, 5.5, 6.71, 6.93, 4.168, 9.779]  
train_Y = [0, 0, 1, 1, 1, 0, 1]
```

2) 입력 값 셋팅

```
X = tf.placeholder("float")  
Y = tf.placeholder("float")
```

```
W = tf.Variable(0., name="weight")  
b = tf.Variable(0., name="bias")
```

2) 모델 셋업(linear)

```
#pred = 1. / (1. + tf.exp(-(W*X + b)))  
pred = tf.sigmoid(W*X + b)
```

Cost & 최적화 설계(Cross Entropy)

```
cost = tf.reduce_mean(tf.reduce_sum(-Y * tf.log(pred) - (1 - Y) * tf.log(1 - pred)))  
optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

테스트 부분도 동일하게 수정

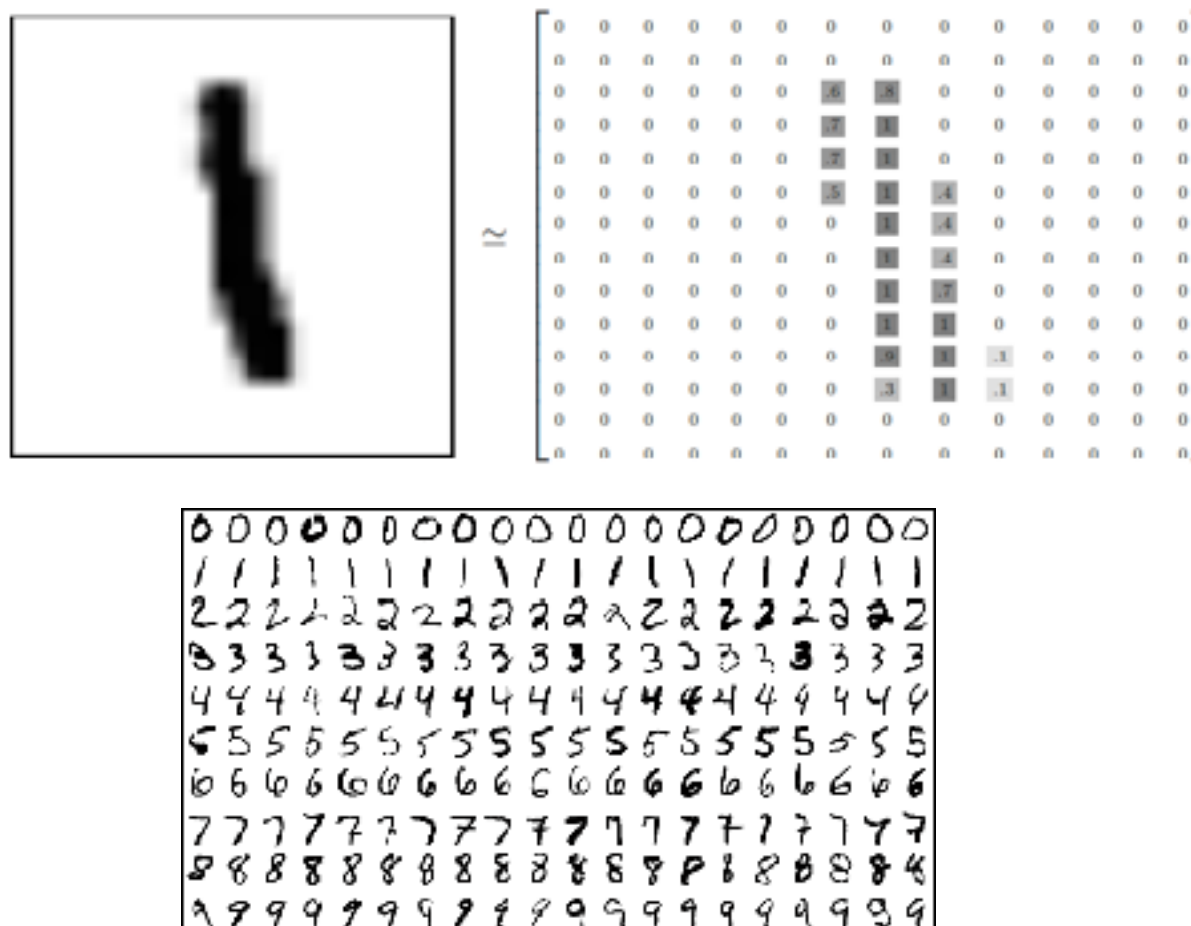
딥러닝 실습 (with MNIST)

- **SLP** (**S**ingle**L**ayer **P**erceptron)
- **MLP** (**M**ulti**L**ayer **P**erceptron)
- **CNN** (**C**onvolution **N**eural **N**etwork)

MNIST: 소개

필기체 숫자 인식에 주로 사용되는 데이터

<http://yann.lecun.com/exdb/mnist/>



MNIST: 데이터 로딩

학습 데이터(MNIST)

[source 4-1]

```
(train_labels, train_images) = mnist.get_data('./datas/', 'train')  
(test_labels, test_images) = mnist.get_data('./datas/', 'test')
```

데이터 사이즈 출력

```
print('total train data : ' + str(train_labels.size))  
print('total test data : ' + str(test_labels.size))
```

임의의 수 내용 출력

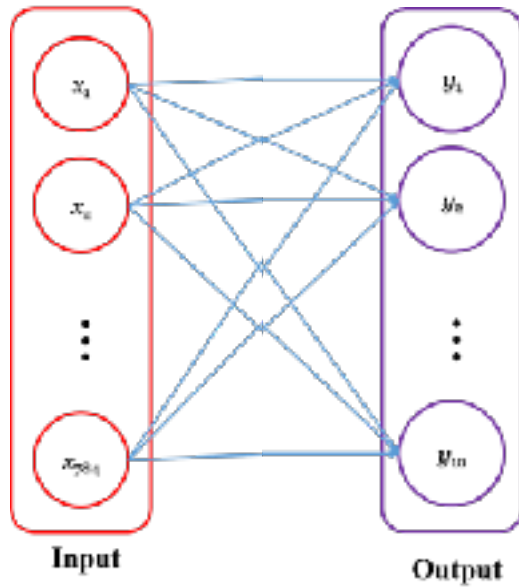
```
print(train_images[0])  
print(train_labels[0])
```

10개 수 출력(with matplotlib)

```
print('label: %s' % (train_labels[0:10]))
```

```
for i in range(10):  
    plt.subplot(1, 10, i+1)  
    plt.imshow(train_images[i], cmap='Greys_r')  
    plt.axis('off')  
plt.show()
```


SLP(SingleLayer Perceptron)



$28 \times 28 = 784$

10

입력 층을 기반으로 하는 'SingleLayer Perceptron'
activation으로는 Softmax & cross entropy를 사용한다

기존 linear regress 예제에서 숫자 입력 데이터를
MNIST 이미지 -> 숫자(10개) 데이터로 변환



SLP(SingleLayer Perceptron)

- 1) Logistics 소스 복사
- 2) 정의 부분 변경

[source 4-2]

학습 데이터(MNIST)

```
(train_labels, train_images) = mnist.get_data('./datas/', 'train')
```

X(이미지)와 Y(숫자값)의 입력값

```
X = tf.placeholder(tf.float32, [None, 28*28]) # 무한대 x 784(28*28) 행렬
```

```
Y = tf.placeholder(tf.int32, [None]) # 무한대 x 1 행렬
```

SLP Model

```
pred = model(X)
```

Cost Function 설계 (Cross Entropy)

```
cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=Y, logits=pred)
```

```
cost = tf.reduce_mean(cross_entropy)
```

Gradient descent Optimizer(학습)

```
optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

SLP(SingleLayer Perceptron)

[source 4-2]

3) 모델 함수 추가

```
# SLP Model
```

```
def model(input_X):
```

```
    # 모델의 weight와 bias의 배열값을 0으로 초기화
```

```
    W = tf.Variable(tf.zeros([28*28, 10]), name="weight")
```

```
    b = tf.Variable(tf.zeros([10]), name="bias")
```

```
    # SoftMax 모델을 생성
```

```
    pred = tf.nn.softmax(tf.matmul(X, W) + b)
```

```
    return pred
```

```
# SLP Model
```

```
pred = model(X)
```

SLP(SingleLayer Perceptron)

4) 학습 부분 변경

[source 4-2]

5) 실행 & 오류 디버깅

```
with tf.Session() as sess:
```

```
    sess.run(tf.global_variables_initializer())
```

```
    index_in_epoch = 0
```

```
    # 학습횟수(epoch:60) -> 총 60000개
```

```
    for epoch in range(1, 60):
```

```
        start = index_in_epoch
```

```
        index_in_epoch += 1000 # 배치 1000개
```

```
        end = index_in_epoch
```

```
        X_images = np.reshape(train_images[start:end], [-1, 28*28])
```

```
        Y_labels = train_labels[start:end]
```

```
        sess.run(optimizer, feed_dict={X: X_images , Y: Y_labels})
```

```
    # 로그
```

```
        training_cost = sess.run(cost, feed_dict={X: X_images , Y: Y_labels})
```

```
        print(epoch, training_cost)
```

```
    print("학습완료! (cost : " + str(training_cost) + ")")
```

SLP(SingleLayer Perceptron)

6) 검증 부분 변경

[source 4-2]

```
# 테스트 데이터로 테스트(총 10000개)
(test_labels, test_images) = mnist.get_data('./datas/', 'test')

test_X = np.reshape(test_images, [-1, 28*28])
test_Y = test_labels

testing_cost = sess.run(cost, feed_dict={X: test_X, Y: test_Y})
print("테스트 완료! (cost : " + str(testing_cost) + ")")

# 학습과 테스트 cost비교(절대값)
print("테스트와 학습의 cost차이 : ", abs(training_cost - testing_cost))
print("학습완료! (cost : " + str(training_cost) + ")")
```

SLP(SingleLayer Perceptron)

7) 값 예측 변경

[source 4-2]

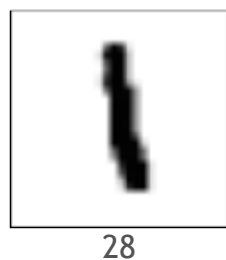
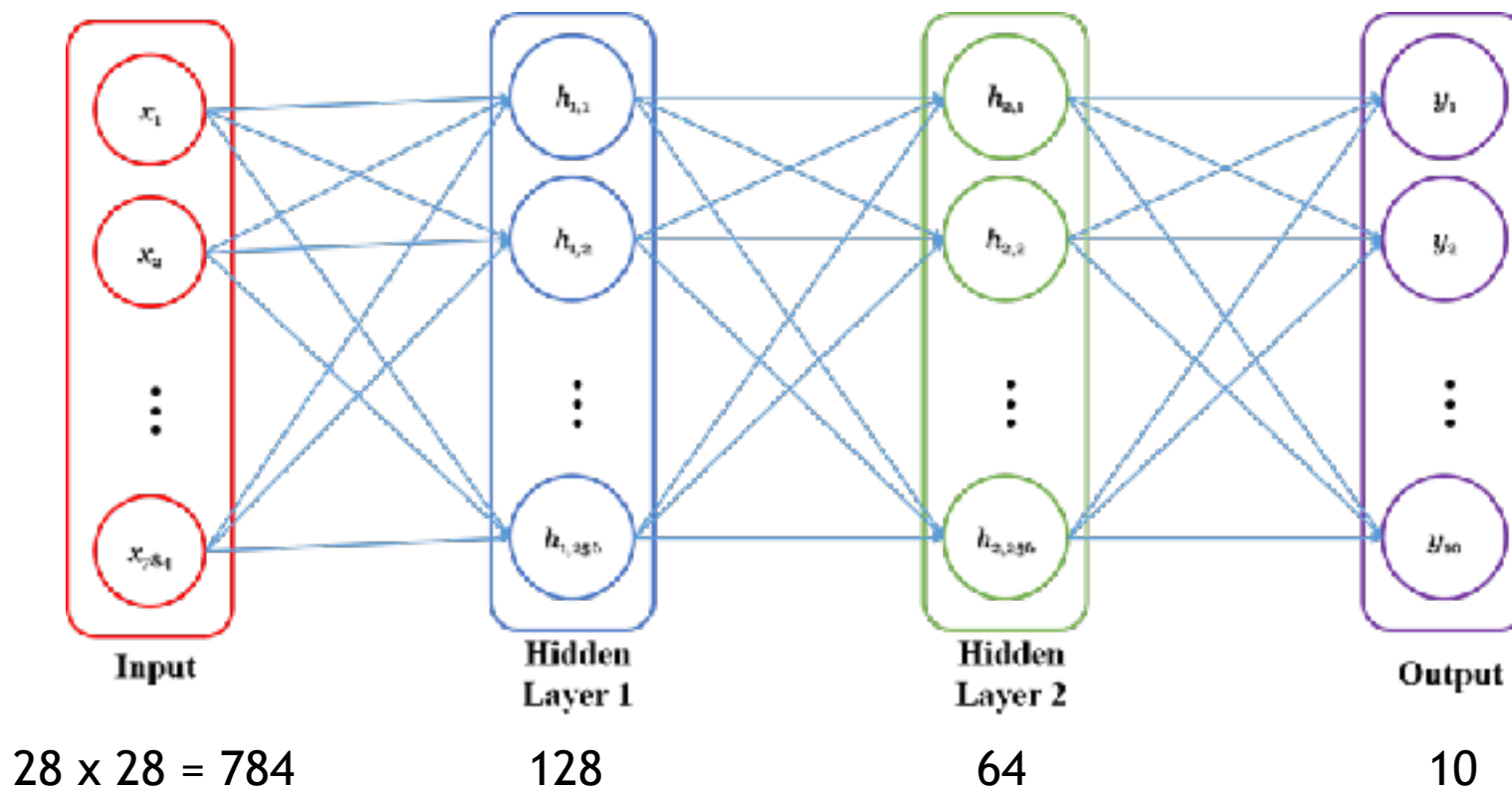
```
# 값 예측 (10개)
for i in range(20):
    x_test = np.reshape(train_images[i], [-1, 28*28])
    arr_data = sess.run(pred, feed_dict={X: x_test})

    pred_val = tf.argmax(arr_data, 1)
    real_val = train_labels[i]

    print("예측값: " + str(pred_val.eval()) + " / 실제값" + str(real_val) + " => " \
          + str(tf.equal(pred_val, real_val).eval()))
```

train cost 결과:
SLP(softmax)
=> cost : 1.688616

MLP(MultiLayer Perceptron)



28



숫자 1

MLP(MultiLayer Perceptron)

1) 모델 함수 변경

[source 4-3]

2) activation 함수 변경(sigmoid -> tanh)

MLP Model

```
def model(input_X):
```

```
    # 히든 레이어 1
```

```
    W1 = tf.Variable(tf.truncated_normal([28*28, 128], stddev=0.1))
```

```
    b1 = tf.Variable(tf.zeros([128]))
```

```
    h1 = tf.nn.sigmoid(tf.matmul(input_X, W1) + b1)
```

```
    # 출력(fully connected) 레이어 (10개 출력)
```

```
    class_num = 10
```

```
    W_fc = tf.Variable(tf.truncated_normal([128, class_num], stddev=0.1))
```

```
    b_fc = tf.Variable(tf.zeros([class_num]))
```

```
    pred = tf.matmul(h1, W_fc) + b_fc
```

```
    return pred
```

```
pred = model(X)
```

train cost 결과:

MLP(sigmoid) Layer 1개

=> cost : 2.017988

MLP(tanh) Layer 1개

=> cost : 1.9295076

MLP(MultiLayer Perceptron)

[source 4-3]

3) 모델에 '히든 레이어 2' 추가

```
# MLP Model
```

```
def model(input_X):
```

```
    # 히든 레이어 1
```

```
    [...]
```

```
    # 히든 레이어2
```

```
    W2 = tf.Variable(tf.truncated_normal([128, 64], stddev=0.1))
```

```
    b2 = tf.Variable(tf.zeros([64]))
```

```
    h2 = tf.nn.tanh(tf.matmul(h1, W2) + b2)
```

```
    # 출력(fully connected) 레이어 (10개 출력)
```

```
    class_num = 10
```

```
    W_fc = tf.Variable(tf.truncated_normal([64, class_num], stddev=0.1))
```

```
    b_fc = tf.Variable(tf.zeros([class_num]))
```

```
    pred = tf.matmul(h2, W_fc) + b_fc
```

```
    return pred
```

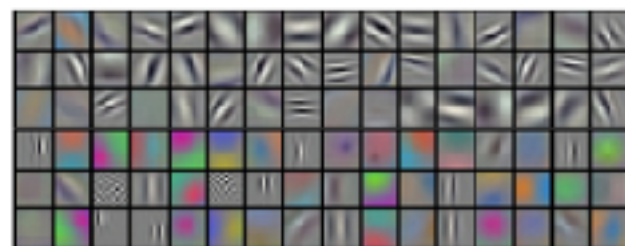
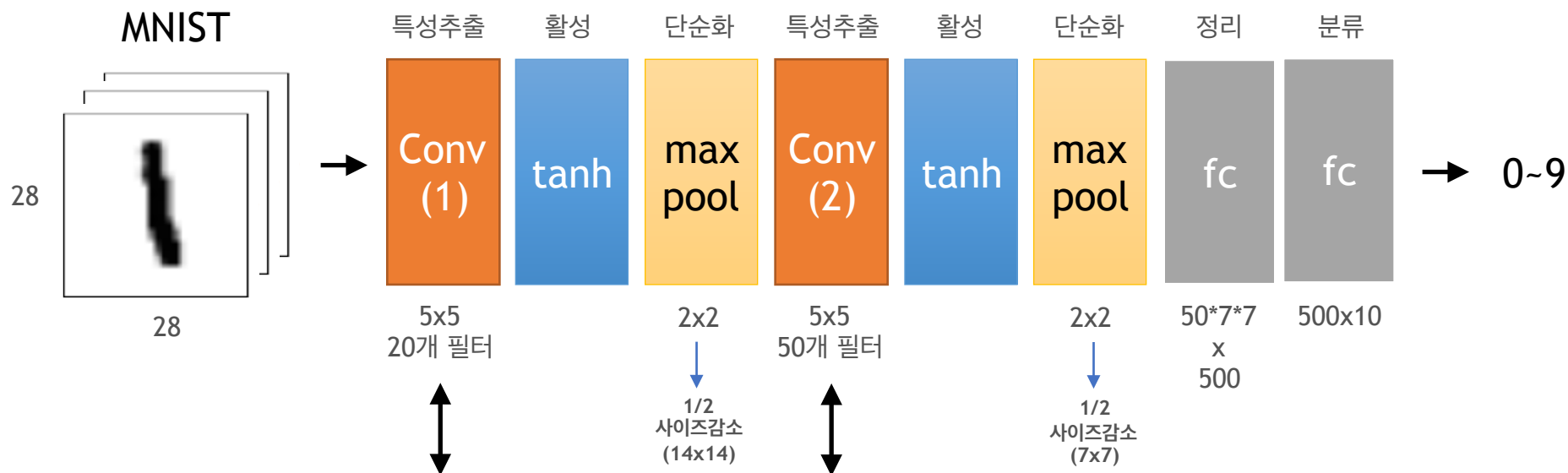
```
pred = model(X)
```

train cost 결과:

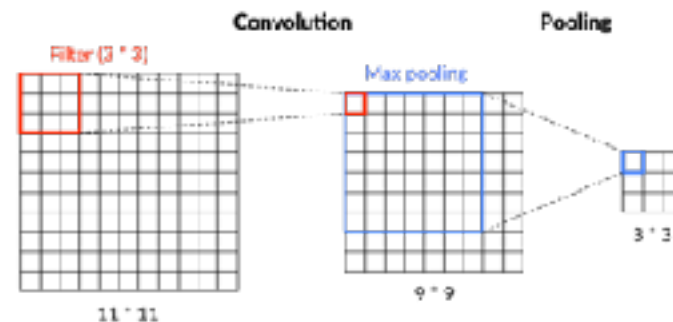
MLP(tanh) Layer 2개

=> cost : 1.7680303

CNN (Convolution Neural Network)

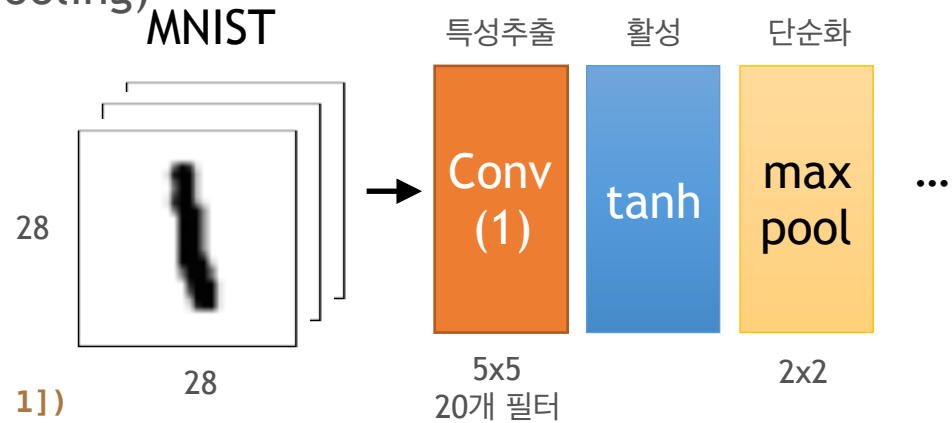


패턴을 활용, 이미지와 convolution(합성곱)을 하여 특성을 추출한다



CNN (Convolution Neural Network)

1) 모델 함수 변경 (Conv, Activate, Pooling)



```
def model(input_X):  
    x_reshape = tf.reshape(X, [-1, 28, 28, 1])  
  
    # Conv 레이어1  
    W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 20], stddev=0.1))  
    b_conv1 = tf.Variable(tf.zeros([20]))  
    h_conv1 = tf.nn.conv2d(x_reshape, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_conv1  
    fmap_conv1 = tf.nn.tanh(h_conv1) # -> Feature(Activation) Map 생성  
  
    # Pooling(Max) 레이어1  
    h_pool1 = tf.nn.max_pool(fmap_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')  
    .  
    .
```

CNN (Convolution Neural Network)

1) 모델 함수 변경 (Convolution Layer 1 설정)

```
def model(input_X):
```

```
    ...
```

```
    # fully-connected 레이어 1
```

```
    W_fc1 = tf.Variable(tf.truncated_normal([20 * 14 * 14, 500], stddev=0.1))
```

```
    b_fc1 = tf.Variable(tf.zeros([500]))
```

```
    h_pool1_flat = tf.reshape(h_pool1, [-1, 20 * 14 * 14])
```

```
    h_fc1 = tf.nn.tanh(tf.matmul(h_pool1_flat, W_fc1) + b_fc1)
```

```
    # 출력(fully connected) 레이어 2 (10개 출력)
```

```
    class_num = 10
```

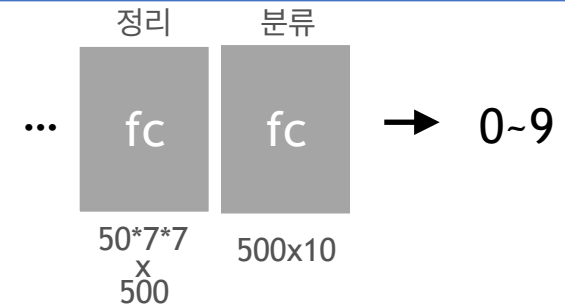
```
    W_fc2 = tf.Variable(tf.truncated_normal([500, class_num], stddev=0.1))
```

```
    b_fc2 = tf.Variable(tf.zeros([class_num]))
```

```
    pred = tf.matmul(h_fc1, W_fc2) + b_fc2
```

```
    return pred
```

```
pred = model(X)
```



train cost 결과:
CNN(tanh) Layer 1개
=> cost : 0.92411

CNN (Convolution Neural Network)

2) Convolution Layer 2 추가



...
Conv 레이어2

```
W_conv2 = tf.Variable(tf.truncated_normal([5, 5, 20, 50], stddev=0.1))
```

```
b_conv2 = tf.Variable(tf.zeros([50]))
```

```
h_conv2 = tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_conv2
```

```
fmap_conv2 = tf.nn.tanh(h_conv2) # -> Feature(Activation) Map 생성
```

Pooling(Max) 레이어2

```
h_pool2 = tf.nn.max_pool(fmap_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

fully-connected 레이어 1

```
W_fc1 = tf.Variable(tf.truncated_normal([50 * 7 * 7, 500], stddev=0.1))
```

```
b_fc1 = tf.Variable(tf.zeros([500]))
```

```
h_pool2_flat = tf.reshape(h_pool2, [-1, 50 * 7 * 7])
```

```
h_fc1 = tf.nn.tanh(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

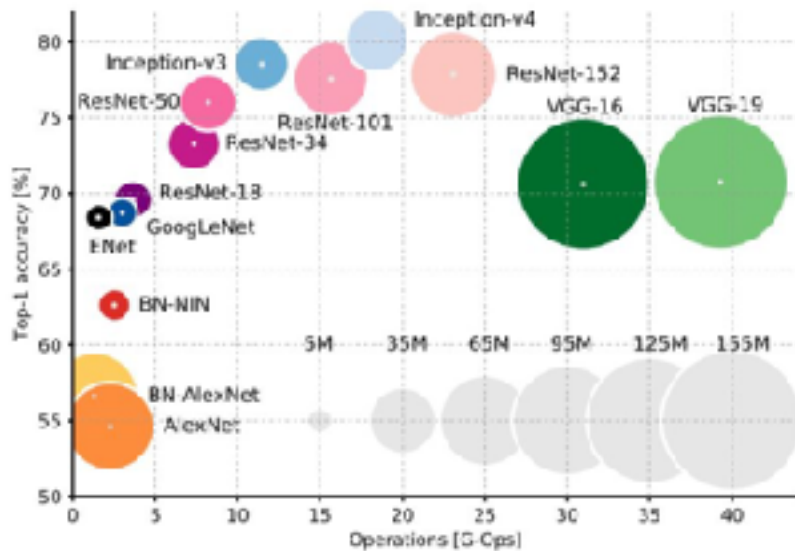
...

train cost 결과:

CNN(tanh) Layer 1개

=> cost : 0.5423556

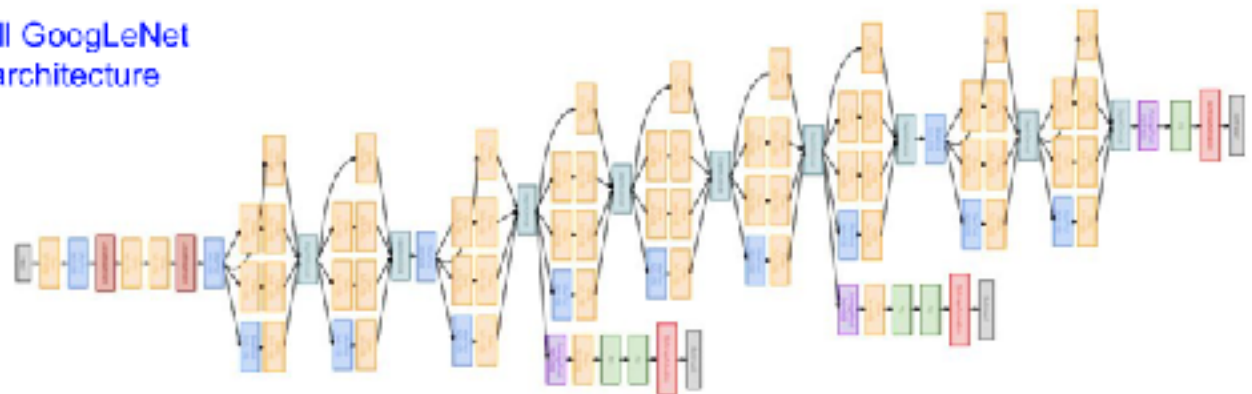
CNN (Convolution Neural Network)



CNN 알고리즘 종류

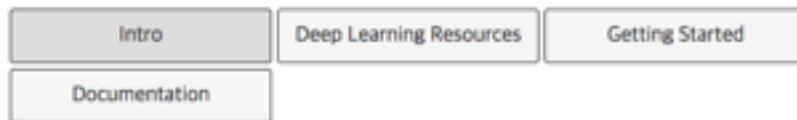
-> 레이어 구성에 따라 성능의 차이가 많이 남

Full GoogLeNet architecture



CNN (Convolution Neural Network)

<https://cs.stanford.edu/people/karpathy/convnetjs/>



ConvNetJS is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser. Open a tab and you're training. No software requirements, no compilers, no installations, no GPUs, no sweat.

Browser Demos

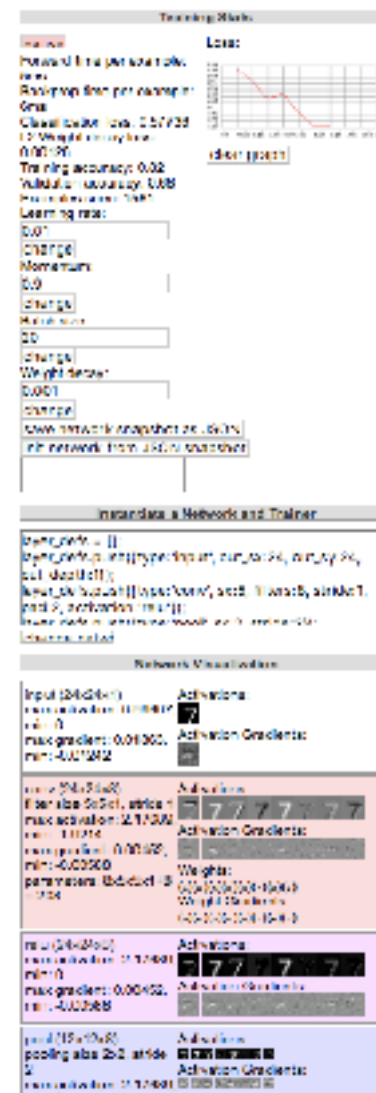
[Classify MNIST digits with a Convolutional Neural Network](#)



[Classify CIFAR-10 with Convolutional Neural Network](#)



[Interactively classify toy 2-D data with a Neural Network](#)





딥러닝 예제 및 실습 II

수업 6

수업목표

- 강화학습(RL)의 예로 OpenAI 플랫폼에 대한 소개
- Keras 소개
- 텐서플로우.js 소개 및 데모 실행
- 기타 (기타 데모, 파이썬 라이브러리, 유용한 사이트 소개)

OpenAI - Gym

- Open AI - 비영리 인공지능 연구회사(2015.12)
- 공식사이트 : <http://gym.openai.com>
- 텐서플로우와 Theano와 연동가능
- 강화학습(**R**einforcement **L**earning - RL)을 위한 라이브러리
- 설치 : `sudo pip3 install gym`

전체 환경을 리스트로 볼 수 있습니다

```
from gym import envs  
print(envs.registry.all())
```



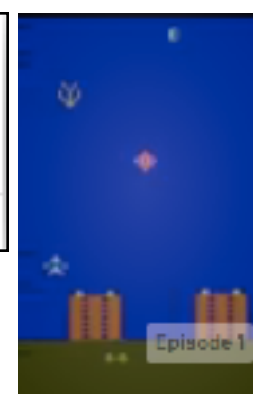
Box2D



Robotics



Classic Control



아타리게임(2600개)



MoJoCo

OpenAI - Gym

강화학습의 필요성

- 강화학습은 결정과 관련된 일반적이고 모든 문제를 다룰 수 있습니다.
- 강화학습 알고리즘은 세상의 많은 어려운 환경에서 좋은 결과를 이루기 위해 시작 되었습니다.

강화학습 연구의 한계

- 지속적으로 나은 벤치마크가 필요
- 실사용된 표준화된 환경의 부족



OpenAI Gym으로 해결

OpenAI - FrozenLake



agent-environment loop

Environment Name :
FrozenLake-v3 (얼음호수 건너기)

문제 해결 조건

S : Start
F : Frozen
H : Hole
G : Gole

```
import gym
```

```
env = gym.make('FrozenLake-v3')
```

=> S에서 H에 안빠지고 G까지 가는것(지도는 매번 변경)

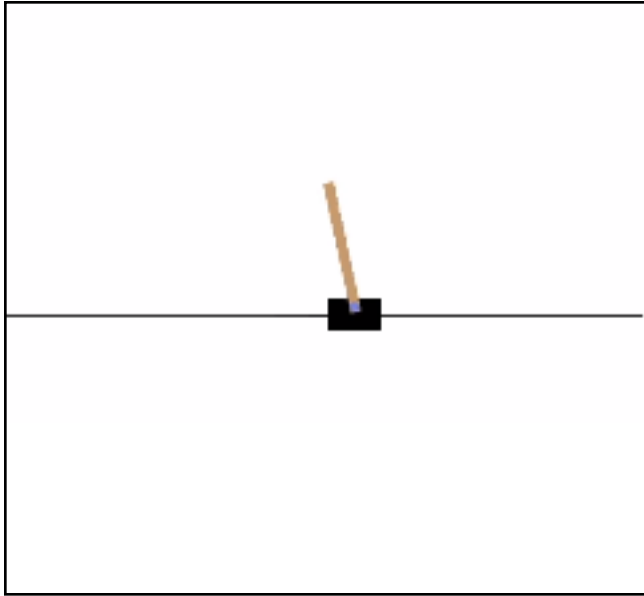
'sudo pip3 install readchar' 설치

[source 5-1] : 사람이 직접 조정

[source 5-2] : Q-Learning - random noise

[source 5-3] : Q-Learning - eGreedy

OpenAI - Gym



Environment Name : CartPole-v0 (막대세우기)

문제 해결 조건 (OpenAI 기준)

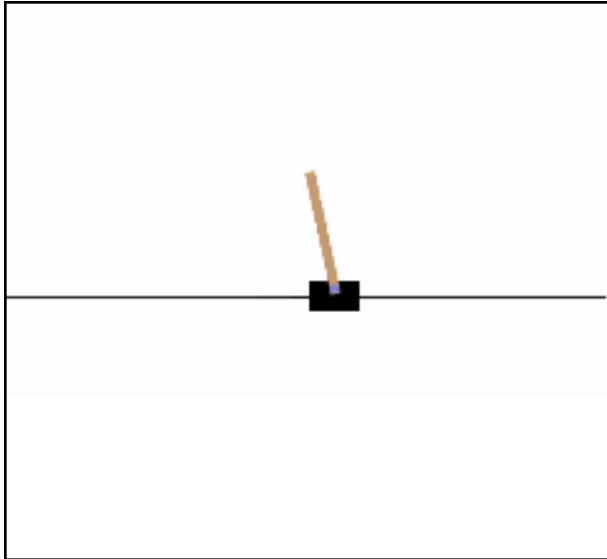
100 에피소드 연속으로 195 이상의 보상을 획득

- 에피소드 : 게임이 한번 시작해서 끝날 때까지 1에피소드
- Timestep : 게임을 얼마나 진행할지 정함
- step : 행동의 횟수
- observation : 관찰에 대한 수치
- reward : 보상
- Environment : 환경 (action, observation)
- Agent : Action을 하는 대상

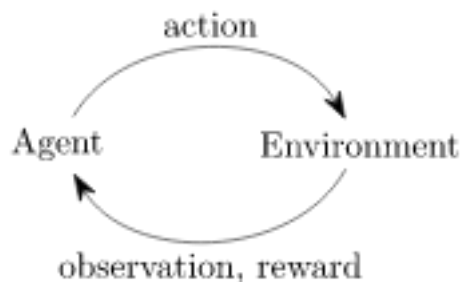


agent-environment loop

OpenAI - Gym



`env.reset()` 순환시작, `done`이 1이면 종료



agent-environment loop

```
import gym [source 5-4]

env = gym.make('CartPole-v0')

for i_episode in range(20): # 에피소드 20번

    observation = env.reset() # 리셋으로 초기 관찰값을 정함

    for t in range(100): # 100번 시도

        env.render() # 환경을 화면으로 출력

        print(observation) # 행동 전 환경에서 얻은 관찰값 출력

        action = env.action_space.sample()

        observation, reward, done, info = env.step(action)

        # 행동 취득 이후에 얻은 값 (관찰값, 보상, 완료여부, 디버깅 정보)

        if done:

            print("Episode finished after {} timesteps".format(t+1))

            break
```

OpenAI - Gym

환경(env) - agent가 활동하는 공간

- env.action_space : 행동하는 공간(0:왼쪽, 1:오른쪽, 두가지)
Discrete(2) -> (랜덤으로 0과 1을 출력)
- env.observation_space : 관찰하는 공간

행동(env.step)의 리턴 값

- opservation(object) : 환경정보를 포함 4차원 벡터
[카트위치, 카트속도, 막대기 각도, 막대기 끝 속도]
- reward(float) : 이전 행동에 대한 보상의 양
넘어지지 않으면 매 시간 1의 값을 갖는다
- done(boolean) : 완료신호
막대가 쓰러지거나 카트가 화면에서 벗어나면 종료
- info(dict) : 디버깅을 위한 정보

```
import gym
from gym import spaces

env = gym.make('CartPole-v0')

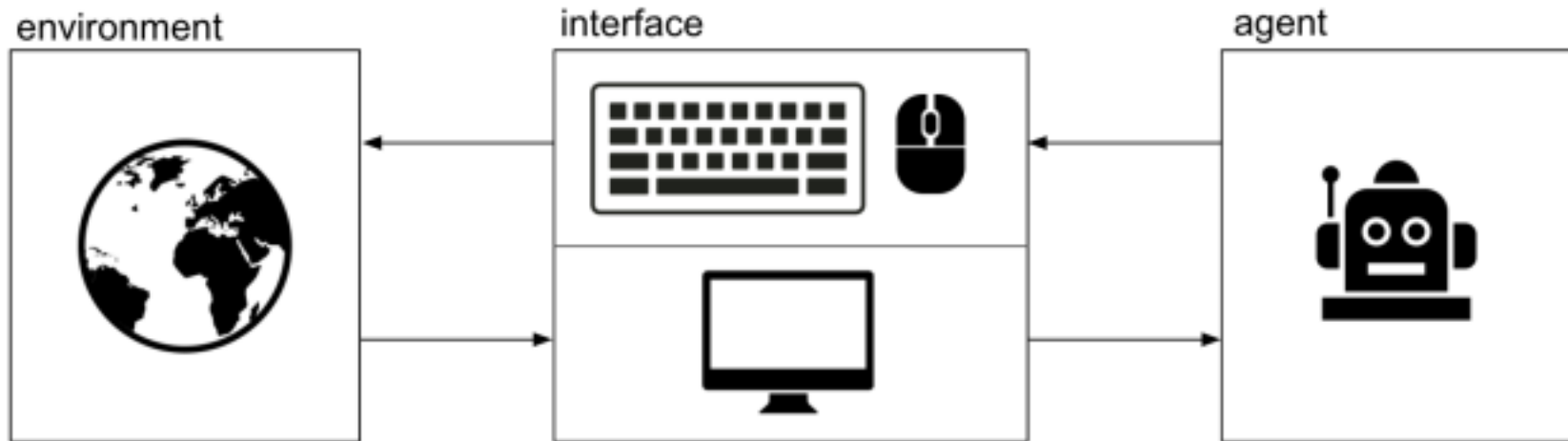
space = env.action_space
print(space.sample()) # 0,1만 출력

print(env.observation_space.high)
print(env.observation_space.low)
```

Q Learning을 이용한 학습

<https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947> => [source 5-5]

OpenAI - Universe



VNC를 통해 원격의 게임의 화면을 실시간으로 캡처하고 Gym라이브러리로 전달하고 이를 통해 키보드와 마우스를 AI 라이브러리에서 학습하여 제어
게임의 내부를 손대지 않고 단순히 실시간 화면을 전송 받는 것만으로 강화학습(RL)을 함



Tensorflow contrib 패키지

- Tensorflow의 High Level API 패키지 모음
- https://www.tensorflow.org/api_docs/python/tf/contrib
- community 라이브러리의 성격 -> 일부 official 패키지화
- Keras도 그 중 하나
- 독립적으로 설치 가능(백엔드 tensorflow, Theano, CNTK지원)
sudo pip3 install keras

Keras - 선형회귀 예제

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation
```

[source 5-6]

초기화

```
model = Sequential()
model.add(Dense(1, input_shape=(1,)))
```

훈련을 위해 모델 준비:

optimiser(stochastic gradient descent), loss(mean squared error)와 를 설정

```
model.compile(optimizer='sgd', loss='mse')
```

훈련데이터 준비 ($Y = 2X - 1$)

```
xs =[1, 2, 3, 4, 5, 6, 7, 8]
ys =[1, 3, 5, 7, 9, 11, 13, 15]
```

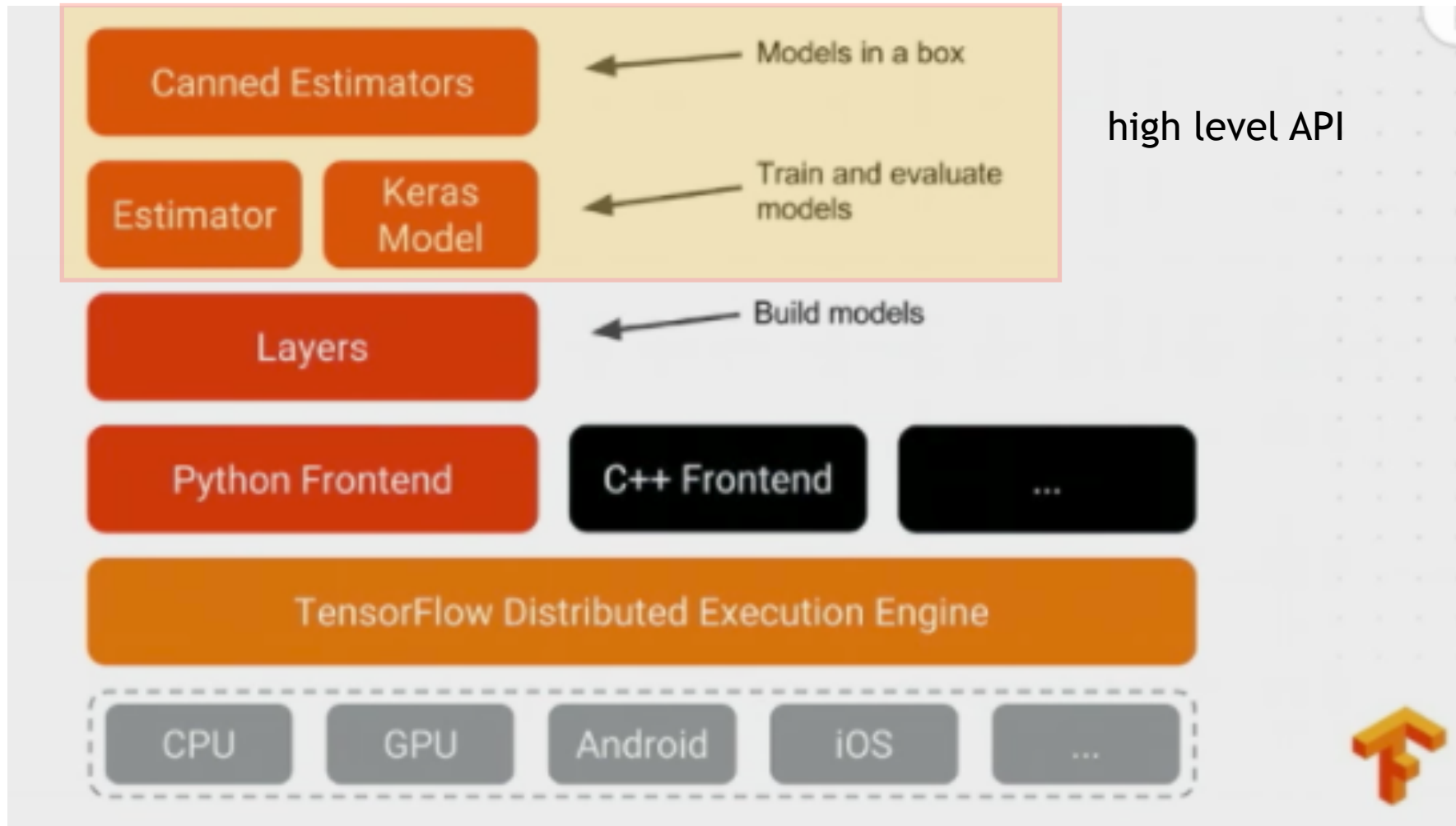
데이터로 훈련

```
model.fit(xs, ys, epochs=100)
```

새로운 입력값으로 테스트

```
model.predict([2])
```

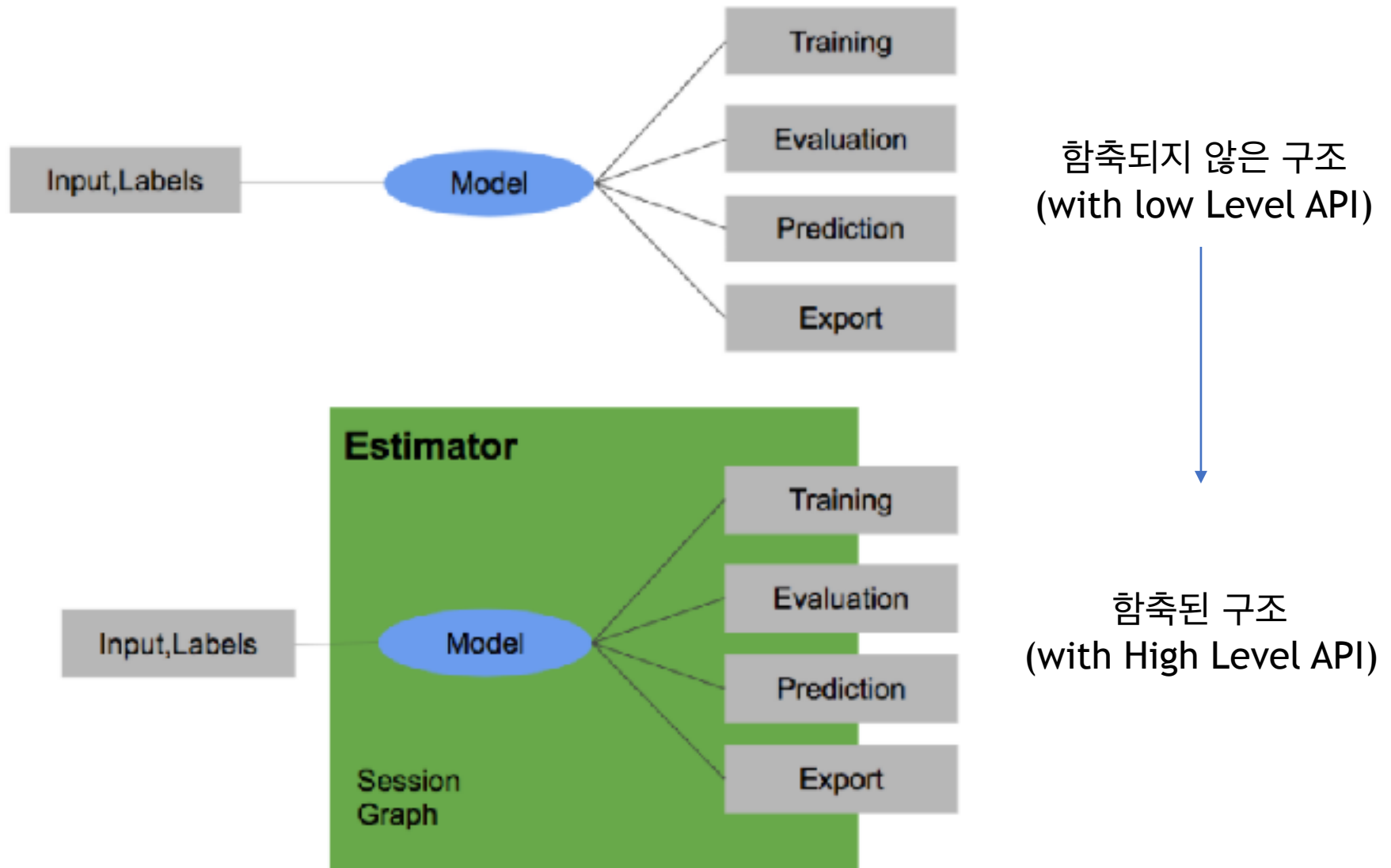
Tensorflow estimator 패키지



Tensorflow estimator 패키지



Tensorflow estimator 패키지



Tensorflow estimator 패키지

```
x_data = [2, 4, 6, 8, 10, 20, 1, 2, 3, 4, 5]
y_data = [4, 8, 12, 16, 20, 40, 2, 4, 5, 8, 10 ]

column_x = tf.feature_column.numeric_column("x", dtype=tf.float32)
columns = [column_x]

estimator = tf.contrib.learn.LinearRegressor(feature_columns=columns, optimizer="Adam")

input_fn_train = tf.estimator.inputs.numpy_input_fn( # 학습
    x = {"x":np.array(x_data[:6], dtype=np.float32)}, y = np.array(y_data[:6], dtype=np.float32),
    num_epochs=10, batch_size=5, shuffle=True
)
estimator.fit(input_fn = input_fn_train, steps=5000)

input_fn_eval = tf.estimator.inputs.numpy_input_fn( # 검증
    x = {"x":np.array(x_data[7:], dtype=np.float32)}, y = np.array(y_data[7:], dtype=np.float32),
    num_epochs=10, batch_size=5, shuffle=True
)
estimator.evaluate(input_fn = input_fn_eval, steps=10)

input_fn_predict = tf.estimator.inputs.numpy_input_fn( # 예측
    x = {"x":np.array([15,20,25,30], dtype=np.float32)},
    num_epochs=1, shuffle=True
)
result = list(estimator.predict(input_fn = input_fn_predict))

print(result)
```

[source 5-7]

tensorflow.js

- 공식사이트 : <https://js.tensorflow.org>
- 예제 : <https://github.com/tensorflow/tfjs-examples>
- 현재 0.11.6 버전
- 브라우저와 Node.JS 지원
- 웹(클라이언트), 서버 사이드에서 실행가능
- 웹) 브라우저에 연결된 GPU자원 사용
서버) GPU/CPU선택하여 Node 패키지 설치(npm)

Tensorflow.js - 선형회귀 예제

[source 5-8]

```
<html>
  <head>
    <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.11.6"> </script>
    <script>
      const model = tf.sequential();
      model.add(tf.layers.dense({units: 1, inputShape: [1]}));

      model.compile({loss: 'meanSquaredError', optimizer: 'sgd'});

      // 모델 ( $Y = 2X - 1$ )
      const xs = tf.tensor2d([1, 2, 3, 4, 5, 6, 7, 8], [8, 1]);
      const ys = tf.tensor2d([1, 3, 5, 7, 9, 11, 13, 15], [8, 1]);

      model.fit(xs, ys, {epochs: 1000}).then(() => {
        model.predict(tf.tensor2d([2], [1, 1])).print();
      });
    </script>
  </head>
  <body></body>
</html>
```

Tensorflow.js - Demo

<https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html>

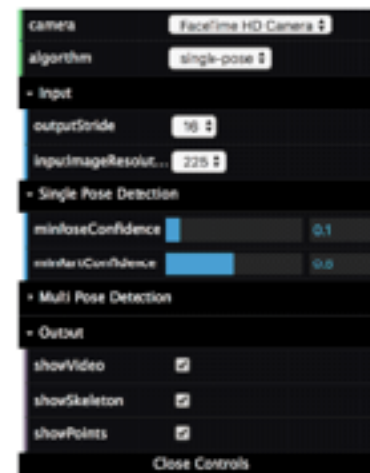
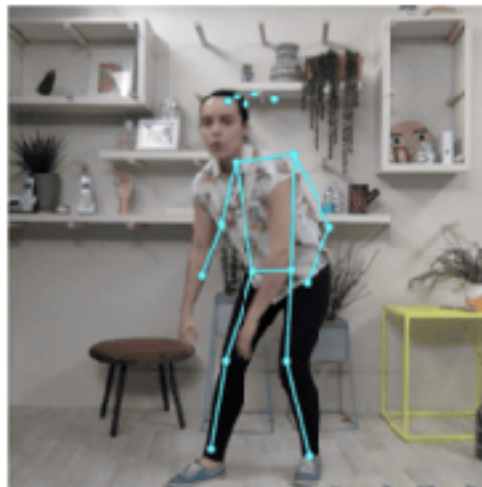
각자 핸드폰으로 테스트 가능

POSENET

Real time [Human Pose Estimation](#) in the browser.

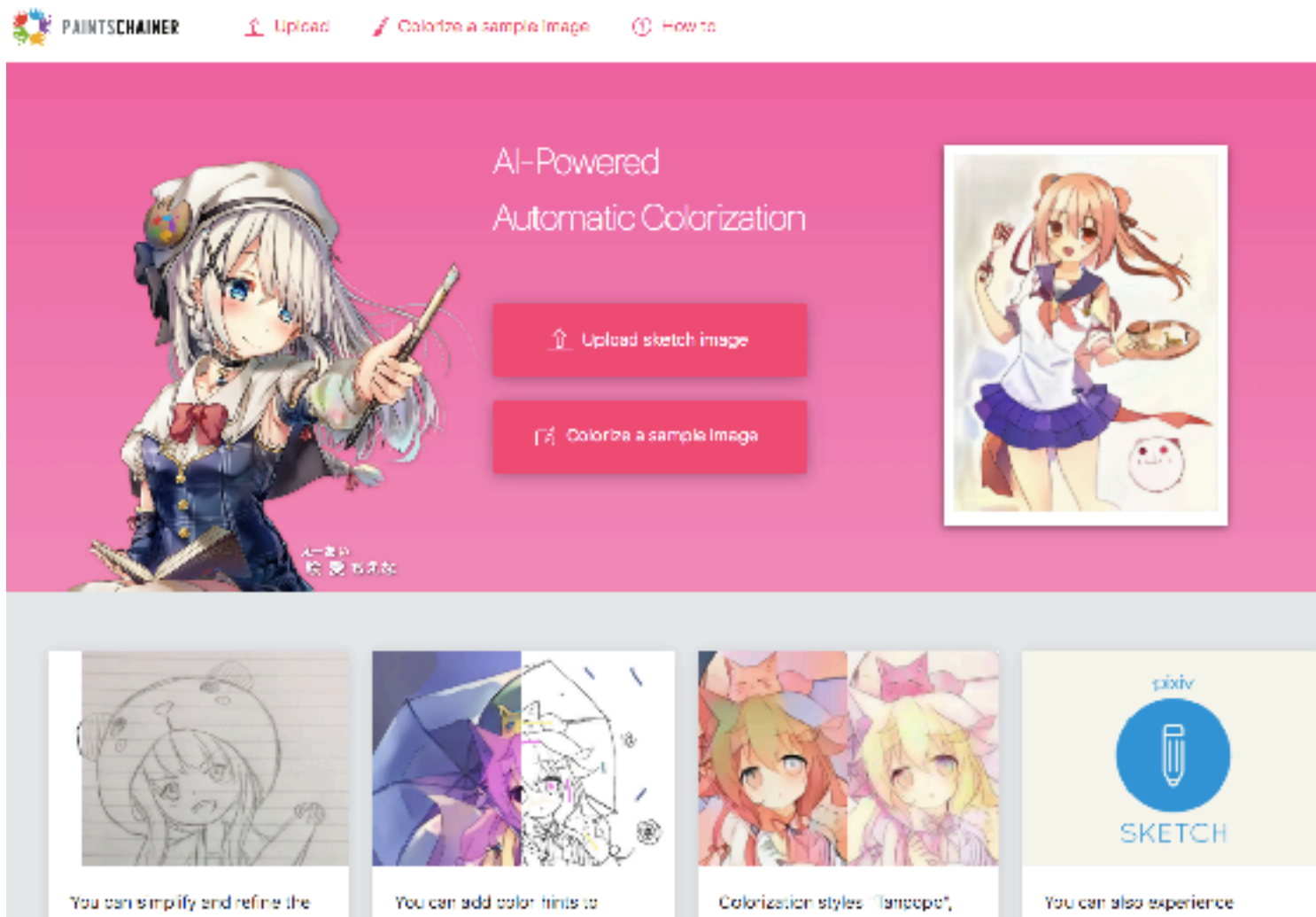
GO TO DEMO!

CODE



PaintChainer - Demo

https://paintschainer.preferred.tech/index_en.html



파이썬 외부 패키지 - list

패키지 명	분야	설명
numpy	수학, 데이터분석	수학 관련 라이브러리로 파이썬의 내부 라이브러리인 math보다 많은 함수를 제공합니다.
scipy	과학	통계, 푸리에 변환, 신호처리 등의 공학식을 함수로 제공합니다.
tensorflow, keras	인공지능	Google의 머신러닝 프레임워크로 numpy와 scipy를 기반으로 만들어졌습니다.
scikit-learn	머신러닝	딥러닝과 강화학습을 다루지 않으나 분류/인식 분야에 많이 사용됩니다.
pandas	데이터 분석	DB와 같이 데이터를 불러와 수학적으로 결과를 처리하는 라이브러리
matplotlib	그래프	결과를 화면에 다양하게 그래프로 그려주는 라이브러리
pillow	이미지	이미지 처리 (PIL가 유명하지만), 최근에 많이 발전되어 사용자가 늘어나는 추세
nltk	텍스트	자연어 처리, 각종 텍스트에서 특징적인 것을 뽑아오거나 비슷한 의미의 단어를 추출한다.
librosa pyAudioAnalysis	음성	음성처리에 사용하는 라이브러리

파이썬 외부 패키지 - list

패키지 명	분야	설명
Seaborn	그래프	matplotlib를 베이스로 확장한 그래프 라이브러리
scrapy	유틸	검색 봇처럼 인터넷의 모든 주소를 크롤링합니다.
pattern	텍스트	scrapy 와 nltk가 결합한 기능, 몇줄로 sns의 데이터를 긁어와서 문장 분석이 가능함
pyevolve	알고리즘	유전자 알고리즘을 라이브러리와 함
dask	데이터 처리	numpy와 pandas와 scikit-learn의 호환되는 자료형을 제공하며, 컨버팅된 데이터를 병렬로 처리할 수 있다.
HPAT	데이터 처리	인텔에서 제공하는 라이브러리로, 고성능 병렬컴퓨팅 프로젝트인 Open MPI의 기능을 이용 병렬처리 코드로 컴파일
Pyro	로보틱스	아이보, 룸바를 지원하는 파이썬 로보틱스 라이브러리
robotframework	스스	http://robotframework.org 로봇 테스트 프레임워크
gensim	텍스트	크크

유용한 사이트

<https://playground.tensorflow.org/> - 딥러닝의 각종 파라미터를 수정해서 결과를 직접 볼수 있는 곳

<https://cs.stanford.edu/people/karpathy/convnetjs> - 딥러닝 웹상에서 돌릴수 있는 예제

<http://datasciencefree.com> - 각종 데이터 분석 라이브러리 요약 정보와 데이터 제공

<https://www.kaggle.com/datasets> - 큰 데이터를 제공하고, 현상금을 주고 문제를 주어 풀도록 함

<https://machinelearning.apple.com/> - 애플의 머신러닝 관련 학술 저널

<http://eehoeskrap.tistory.com/m/175?category=618280> - Vision 분야의 Datasets 모음

