아티스트를 위한 머신러닝 & 딥러닝

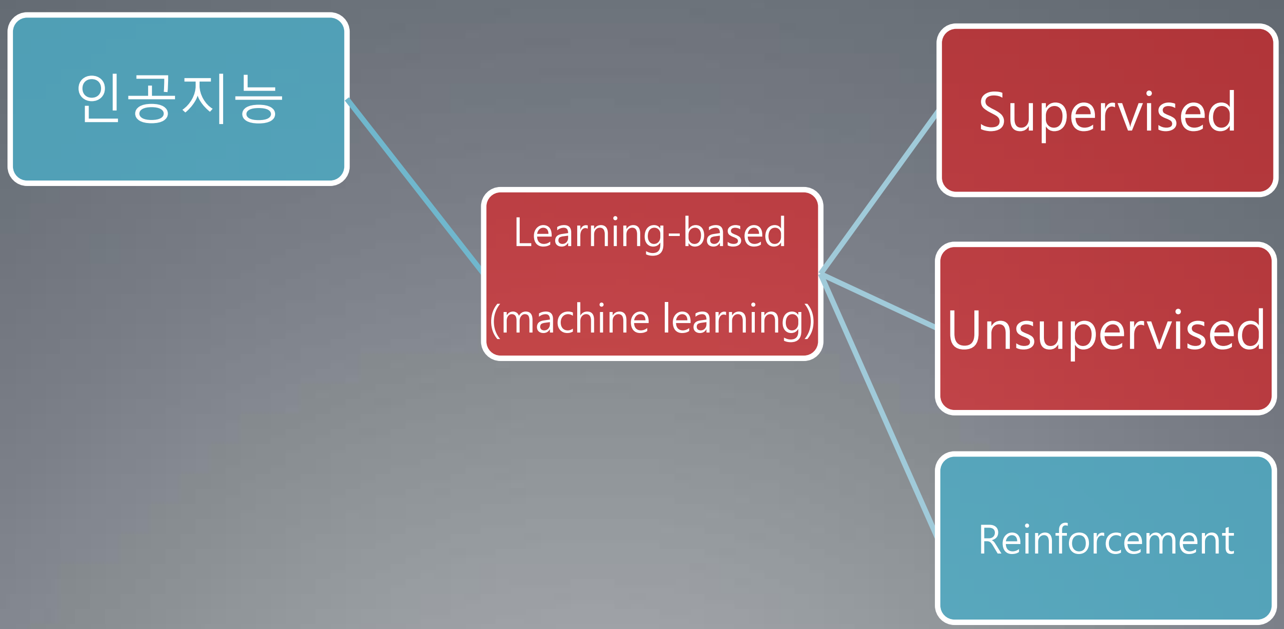
텐서플로를 활용한 딥러닝 #2

서울대학교 & V.DO / 김대식

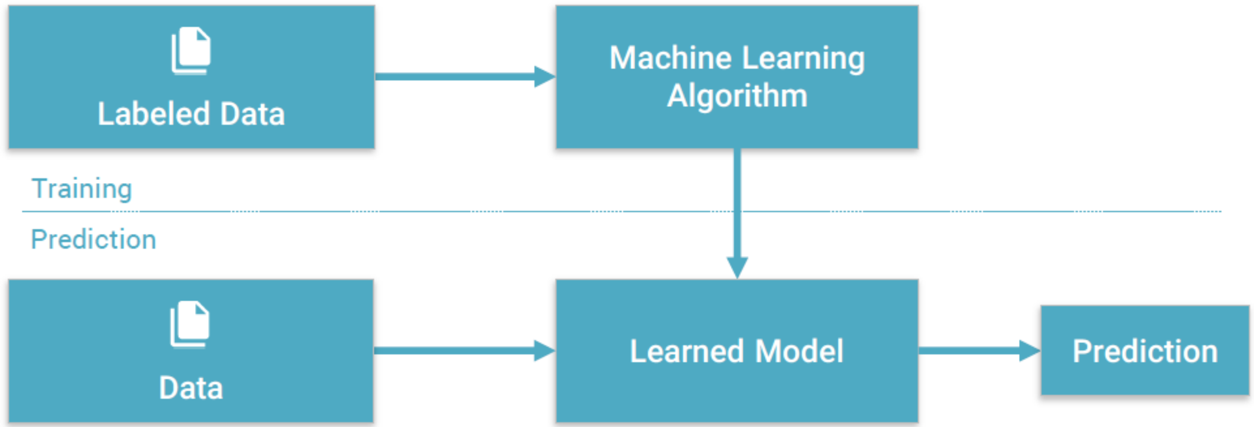
art center nabi v.do

Recap

우린 Learning 배웁니다



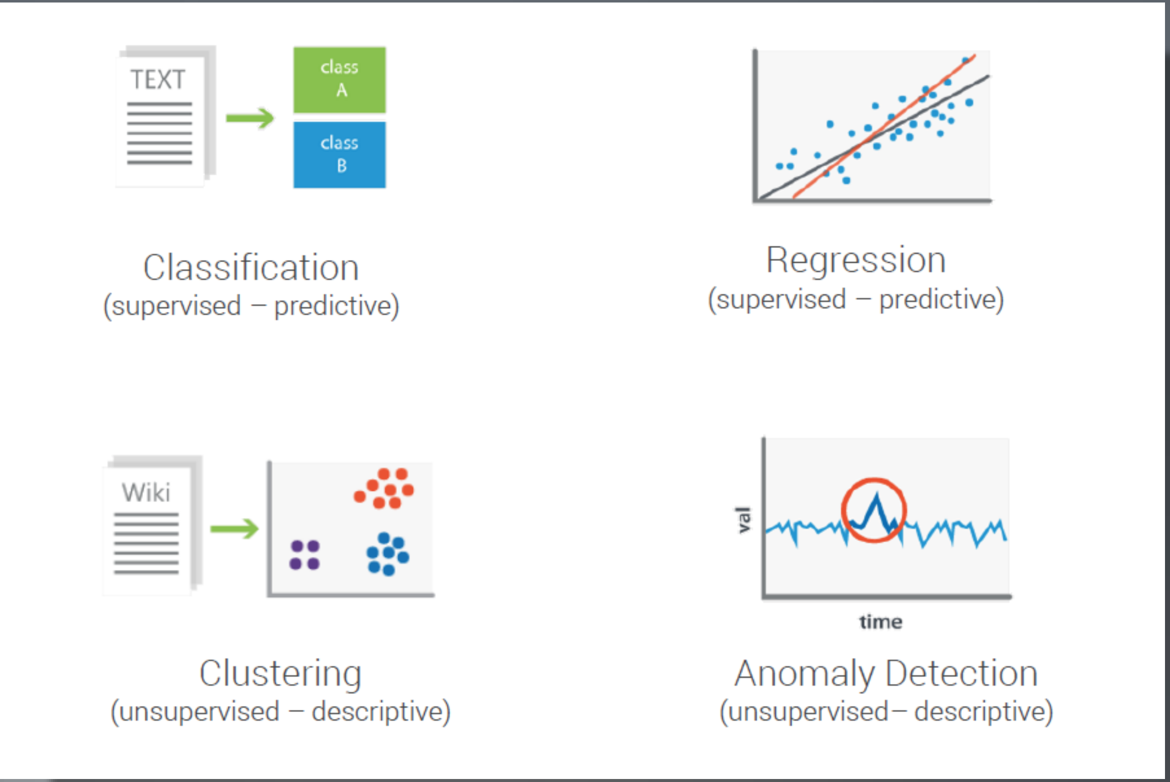
Machine Learning



Provides various techniques that can learn from and make predictions on data

art center nabi v.do

Types of Machine Learning



Linear Regression

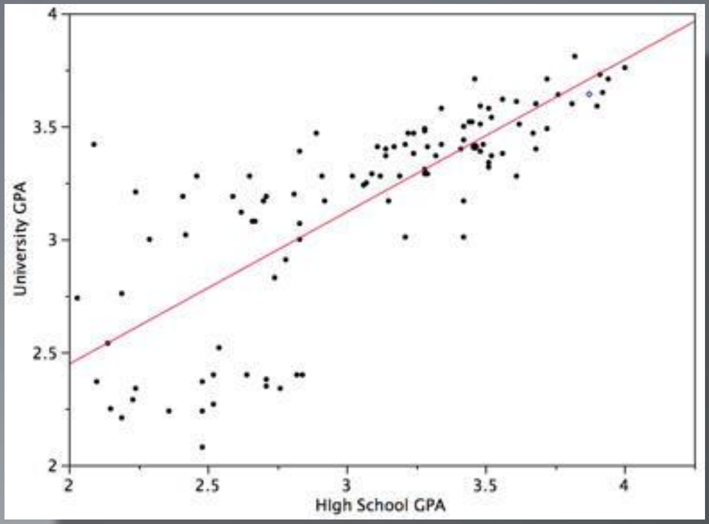
(선형 회귀)

선형 회귀 분석(Linear Regression Analysis)

통계학에 가까운 개념

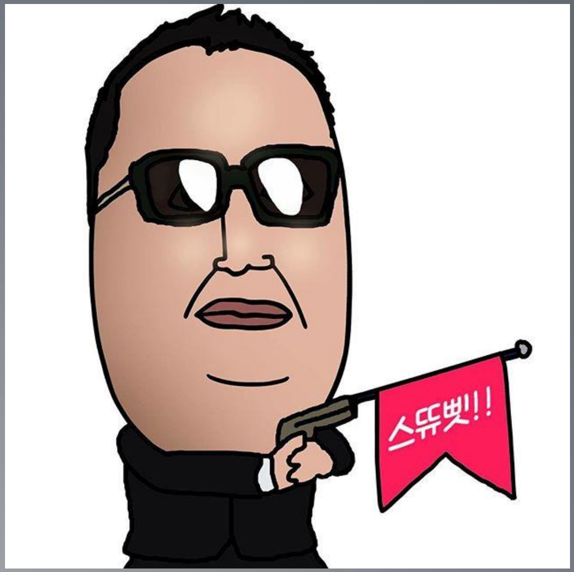
: X와 Y의 관계 분석, 관계에 대한 가설을 검정

→ 선형관계(Linear) 가정



왜 배우는가?

안타깝지만 연구해야할 세상일 중에 이런 관계는 별로 없다.



art center nabi v.do

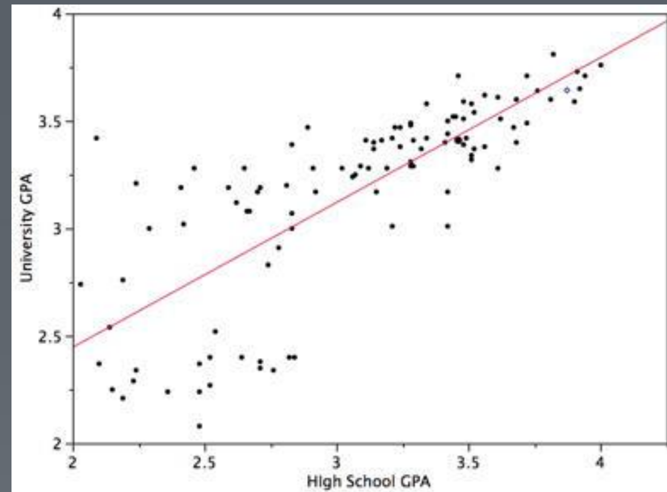
왜 배우는가?

하지만 이것도 못하면
더 복잡한 것을 못한다.

왜 배우는가?

선형 회귀 ≑ 신경망

$$y = b + wx = wx$$



Simple Linear Regression Model

Dependent Variable; Population Y intercept; Population Slope Coefficient; Independent Variable; Random Error term

$$Y_i = \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} + \underbrace{\varepsilon_i}_{\text{Random Error component}}$$

선형회귀 문제의 다양한 접근법

통계학 vs 기계학습

A Closed form solution:

We defin

Then we

:

normal equatio

Thus, the value

equation

the pseudo

Maximum likelihood and least squares

Given observed inputs $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, and targets $\mathbf{t} = [t_1, \dots, t_N]^T$, we obtain the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}).$$

Taking the logarithm, we get

N

where

is the

Gradient descent algorithm

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1) \\
&\}
\end{aligned}
$$

Correct: Simultaneous update

$$
\begin{aligned}
&\text{temp0} := \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
&\text{temp1} := \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
&\theta_0 := \text{temp0} \\
&\theta_1 := \text{temp1}
\end{aligned}
$$

Dependent Variable

Population Y intercept

Population Slope Coefficient

Independent Variable

Random Error term

$$Y_i = \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} + \underbrace{\varepsilon_i}_{\text{Random Error component}}$$

```
. regress cholesterol time_tv

      Source |       SS       df       MS              Number of obs =     100
-------------+------------------------------           F(  1,    98) =   17.47
       Model |  5.04902329     1  5.04902329           Prob > F      =  0.0001
    Residual |  28.3220135    98  .289000137           R-squared     =  0.1513
-------------+------------------------------           Adj R-squared =  0.1426
       Total |  33.3710367    99  .337081179           Root MSE      =  .53759

------------------------------------------------------------------------------
 cholesterol |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
     time_tv |   .0440691   .0105434     4.18   0.000     .0231461    .0649921
       _cons |  -2.134777   1.813099    -1.18   0.242    -5.732812    1.463259
------------------------------------------------------------------------------
```

비슷하지만 다르다

통계학 >
: 샘플을 통해 모수의 통계적 특성을 추출 및 분석, 가설 검정

기계학습 >
: 샘플의 특성(train set)을 학습하여 새로운 샘플(test set)의 값을 예측

기계학습

= 짬뽕 학문

= 통계학 + 대수학 + 최적화이론 + 프로그래밍학 + ...

- 통계적 접근도 충분히 가능

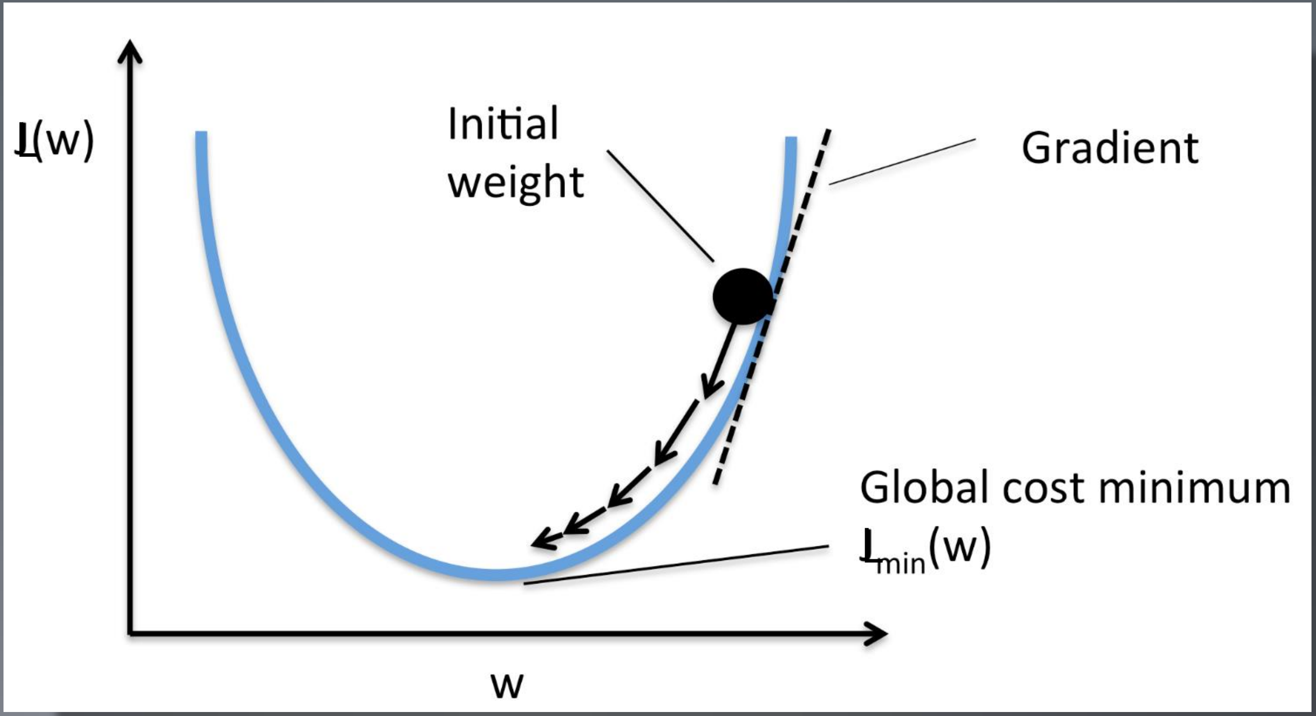
→하지만 많은 가정이 필요

- **Gradient decent**

→실용적이면서 딥러닝과의 연계

→ 수업은 이것으로!

Gradient decent (경사 하강법)



Gradient decent (경사 하강법)

$$f(x_i) = f_{W,b}(x_i) = b + \sum_{j=1}^{p} W_j x_{ij} \tag{1}$$

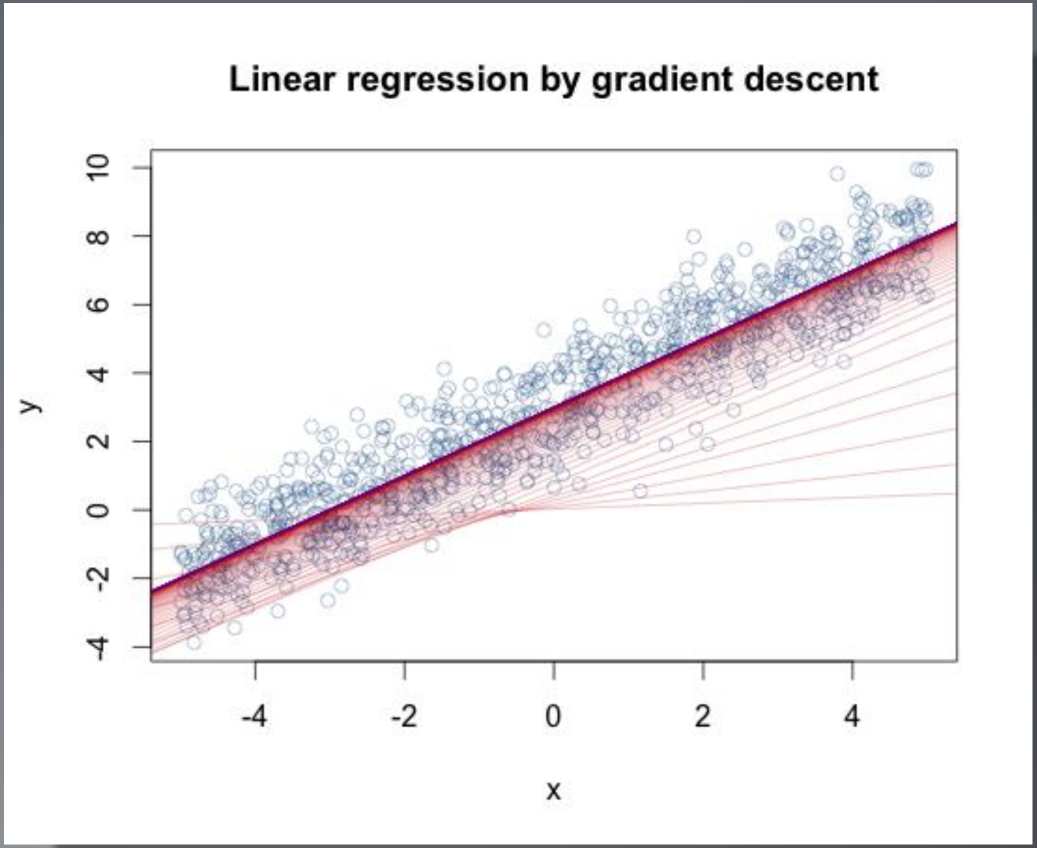
$$L(W, b) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i) - y_i)^2 \tag{2}$$

$$\frac{\partial L}{\partial W} = \frac{2}{n} \sum_{i=1}^{n} (f(x_i) - y_i) x_i \qquad \frac{\partial L}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} (f(x_i) - y_i) \tag{3}$$

$$\begin{aligned} W &\leftarrow W - \alpha \frac{\partial L}{\partial W} \\ b &\leftarrow b - \alpha \frac{\partial L}{\partial b} \end{aligned} \tag{4}$$

반복적인 최적화 기법 !

Gradient decent (경사 하강법)

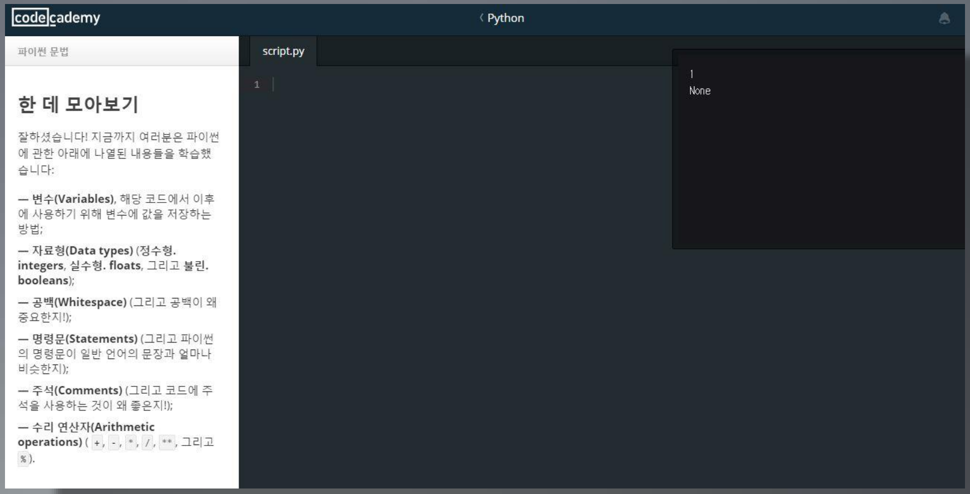


실습

https://github.com/chuckgu/nabi

파이썬 기초 학습

https://www.codecademy.com/ko/tracks/python-ko



art center nabi v.do

Tensorflow 기본

텐서플로를 한다는 것:

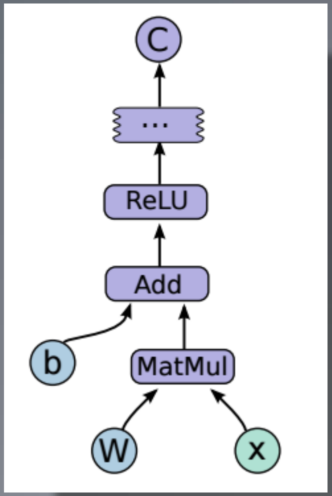
주어진 DATA 를
구현한 Algorithm 에
흘려 학습하는 과정

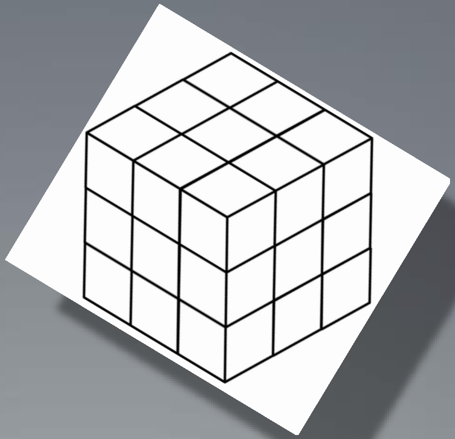
Tensorflow 기본

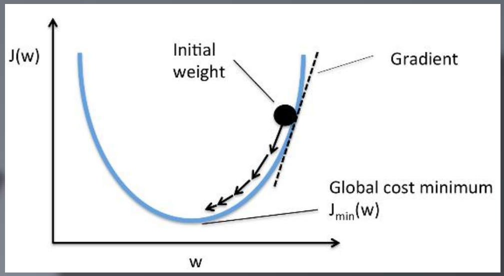
1.Graph를 그린다

2.데이터(Tensor)를 넣는다

3.Loss를 이용하여 학습







art center nabi v.do

Tensorflow 기본

1.Graph를 그린다

Subject, Object

Verb

Placeh

Varia

Const

eration

Core graph data structures

- `tf.Graph`
- `tf.Operation`
- `tf.Tensor`

art center nabi v.do

Tensorflow 기본

1.Graph를 그린다

Subject, Object

Placeholder

Variable

Constant

```
tf.placeholder(dtype, shape=None, name=None)
```

Data를 받기 위한 바구니

Dtype: data의 type
Shape: data의 모양
Name: placeholder의 이름

Tensorflow 기본

1.Graph를 그린다

Subject, Object

Placeholder

Variable

Constant

```
# Create a variable.
w = tf.Variable(<initial-value>, name=<optional-name>)
```

```
W = tf.Variable(tf.random_normal(shape=[784, 10], stddev=0.01), name="weights")

b = tf.Variable(tf.zeros([1, 10]), name="bias")
```

초기화 이후에 계속 변할 수 있는 값

Initial value: Tensor형태의 초기값
Name: variable의 이름

꼭 초기화 해야함!

```
init = tf.global_variables_initializer()

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

Tensorflow 기본

1.Graph를 그린다

Subject, Object

Placeholder

Variable

Constant

```
constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
    verify_shape=False
)
```

바뀌지 않는 값을 tf.Tensor로 저장

Value: tf.constant가 가지는 값
Dtype: data의 type
Shape: data의 모양
Name: placeholder의 이름

Tensorflow 기본

1.Graph를 그린다

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Tensorflow 기본

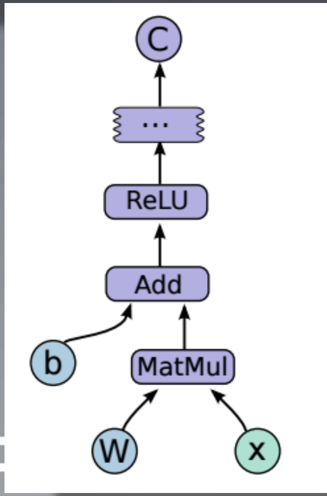
1.Graph를 그린다

- operation은 이름을 가지며 추상적인 연산을 의미함
- Graph-construction time에 operation이 생성(노드)
- 딥 러닝에 사용되는 대표적인 operation 및 class로는 constant, Variable, placeholder, Matrix operation, neural-net building block 등이 있음

Verb

Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural-net building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Operation

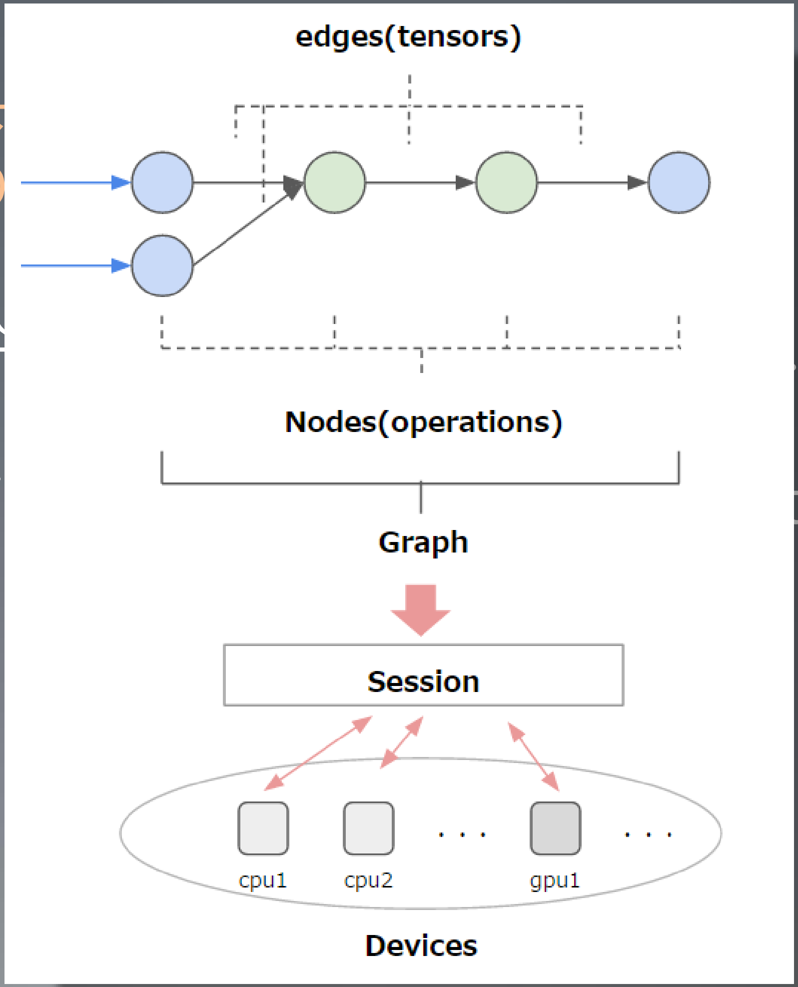


Tensorflow 기본

2.데이터(Tensor)를 넣는다

Session : 텐서
일어

- run 함수
- Feed_di tensor를 받는 인자



art center nabi v.do

Tensorflow 기본

2.데이터(Tensor)를 넣는다

꼭 placeholder에서 정해놓은 type, shape 지켜야함!

```
sess.run(optimizer, feed_dict={X: x, Y: y})
```

```
import tensorflow as tf

# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

# create a constant of type float 32-bit, shape is a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b # short for tf.add(a, b)

with tf.Session() as sess:
    # feed [1, 2, 3] to place holder a via the dict {a: [1, 2, 3]}
    # fetch value of c
    print(sess.run(c, {a: [1, 2, 3]}))
```

```
[ 6.  7.  8.]
```

Tensorflow 기본

3.Loss를 이용하여 학습

Loss란?

= 내가 잘못해서 생긴 정답과의 차이
= 내가 예측한 $\bar{y}$ 와 정답 y와의 차이

Loss함수의 결정

y의 형태에 따라
- MSE(mean square error)
- Binary cross-entropy
- Cross-entropy

Tensorflow 기본

3.Loss를 이용하여 학습

y의 형태에 따라

- MSE(mean square error)
  실제 숫자을 예측하는 경우

```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
```

- Binary cross-entropy
  정답이 0 또는 1 인 경우

```
tf.nn.sigmoid_cross_entropy_with_logits(logits, targets,
name=None)
```

- Cross-entropy
  정답이 유한개의 클래스인 경우

```
xent = tf.nn.softmax_cross_entropy_with_logits(logits, labels)
```

Tensorflow 기본

3.Loss를 이용하여 학습

Optimizers

The Optimizer base class provides methods to compute gradients for a loss and apply gradients to variables. A collection of subclasses implement classic optimization algorithms such as GradientDescent and Adagrad.

You never instantiate the Optimizer class itself, but instead instantiate one of the subclasses.

- `tf.train.Optimizer`
- `tf.train.GradientDescentOptimizer`
- `tf.train.AdadeltaOptimizer`
- `tf.train.AdagradOptimizer`
- `tf.train.AdagradDAOptimizer`
- `tf.train.MomentumOptimizer`
- `tf.train.AdamOptimizer`
- `tf.train.FtrlOptimizer`
- `tf.train.ProximalGradientDescentOptimizer`
- `tf.train.ProximalAdagradOptimizer`
- `tf.train.RMSPropOptimizer`

Tensorflow 기본

3.Loss를 이용하여 학습

```
# 텐서플로우에 기본적으로 포함되어 있는 함수를 이용해 경사 하강법 최적화를 수행합니다.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# 비용을 최소화 하는 것이 최종 목표
train_op = optimizer.minimize(cost)

# 세션을 생성하고 초기화합니다.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    # 최적화를 100번 수행합니다.
    for step in range(100):
        # sess.run 을 통해 train_op 와 cost 그래프를 계산합니다.
        # 이 때, 가설 수식에 넣어야 할 실제값을 feed_dict 을 통해 전달합니다.
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})
```

```
#  Note, minimize() knows to modify W and b because Variable objects are trainable=True by default
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(c), \
                "W=", sess.run(W), "b=", sess.run(b))
```

Tensorflow 기본

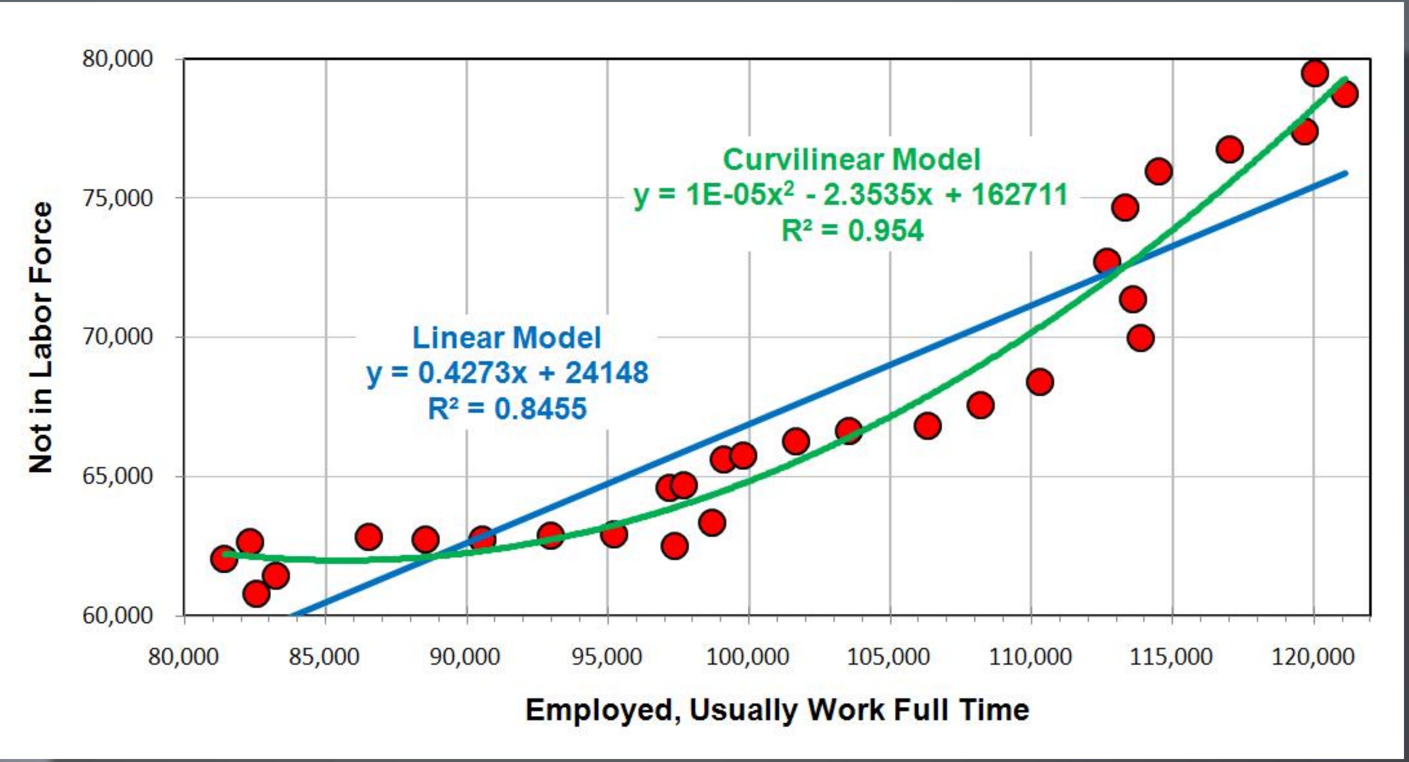
Linear Regression 실습

Tensorflow 기본

```
import tensorflow as tf
import numpy as np

x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.initialize_all_variables()

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

for step in xrange(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))
```

```
(0, array([-0.02093266], dtype=float32), array([ 0.47095078], dtype=float32))
(20, array([ 0.05205543], dtype=float32), array([ 0.32374111], dtype=float32))
(40, array([ 0.08607709], dtype=float32), array([ 0.30689433], dtype=float32))
(60, array([ 0.09595685], dtype=float32), array([ 0.3020021], dtype=float32))
(80, array([ 0.09882588], dtype=float32), array([ 0.3005814], dtype=float32))
(100, array([ 0.09965905], dtype=float32), array([ 0.30016884], dtype=float32))
(120, array([ 0.09990099], dtype=float32), array([ 0.30004904], dtype=float32))
(140, array([ 0.09997127], dtype=float32), array([ 0.30001423], dtype=float32))
(160, array([ 0.09999166], dtype=float32), array([ 0.30000415], dtype=float32))
(180, array([ 0.09999759], dtype=float32), array([ 0.3000012], dtype=float32))
(200, array([ 0.09999931], dtype=float32), array([ 0.30000037], dtype=float32))
```

Linear의 한계



art center nabi v.do

Non-linear의 위력
:세상은 non linear하다

art center nabi v.do

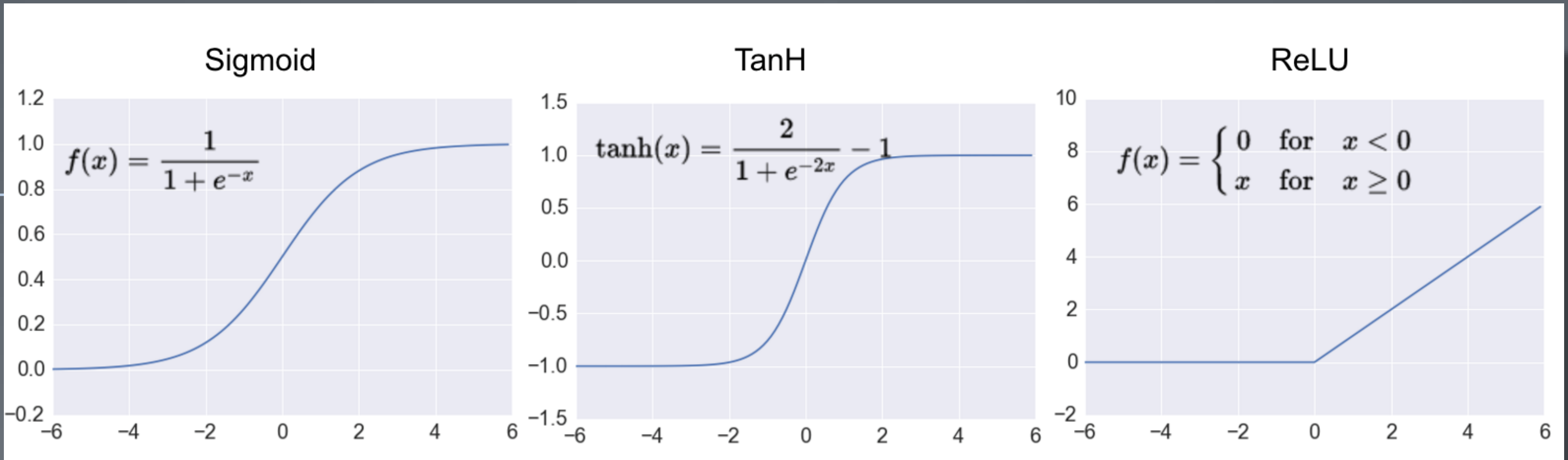
Non-linear를 풀기 위한
세상의 다양한 노력들

→ 그것을 가장 잘해내는 것이 현재 딥러닝뿐

Linear를 Non-linear로 바꾸려면?

Activation 함수 쓰자!

Activation 함수란?



새로운 함수들이 쏟아짐

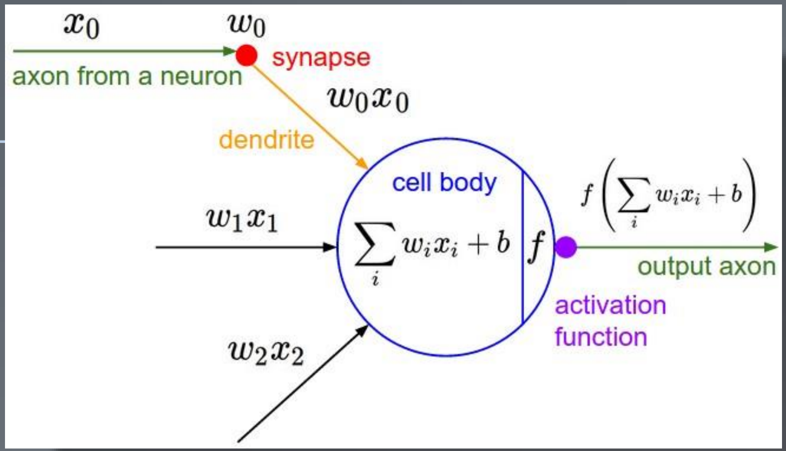
선형 회귀 ≒ 신경망

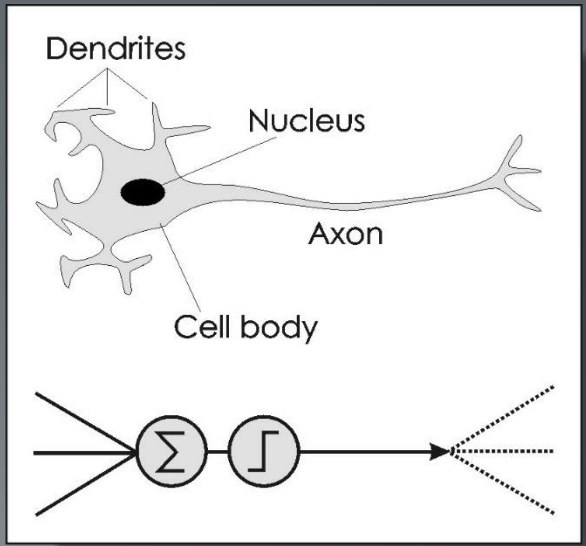
$$y = b + wx = wx$$

선형 회귀 + activation F
=신경망

$$y = \sigma(b + wx)$$

인공 뉴런 vs 생물학적 뉴런





뉴럴 네트워크은 세상에
모든 함수를 표현해 낸다

*Approximation by superpositions of a sigmoidal function, by George Cybenko (1989). The result was very much in the air at the time, and several groups proved closely related results. Cybenko's paper contains a useful discussion of much of that work. Another important early paper is Multilayer feedforward networks are universal approximators, by Kurt Hornik, Maxwell Stinchcombe, and Halbert White (1989). This paper uses the Stone-Weierstrass theorem to arrive at similar results.

http://neuralnetworksanddeeplearning.com/chap4.html

감사합니다