Deep Learning

Implementation of Linear regression in Keras

Keras and 뉴런, Dense로 구현하는 모델구조
Example of Linear Regression Model

Yoon Joong Kim
Department of Computer Engineering, Hanbat National University
yjkim@hanbat.ac.kr

내용

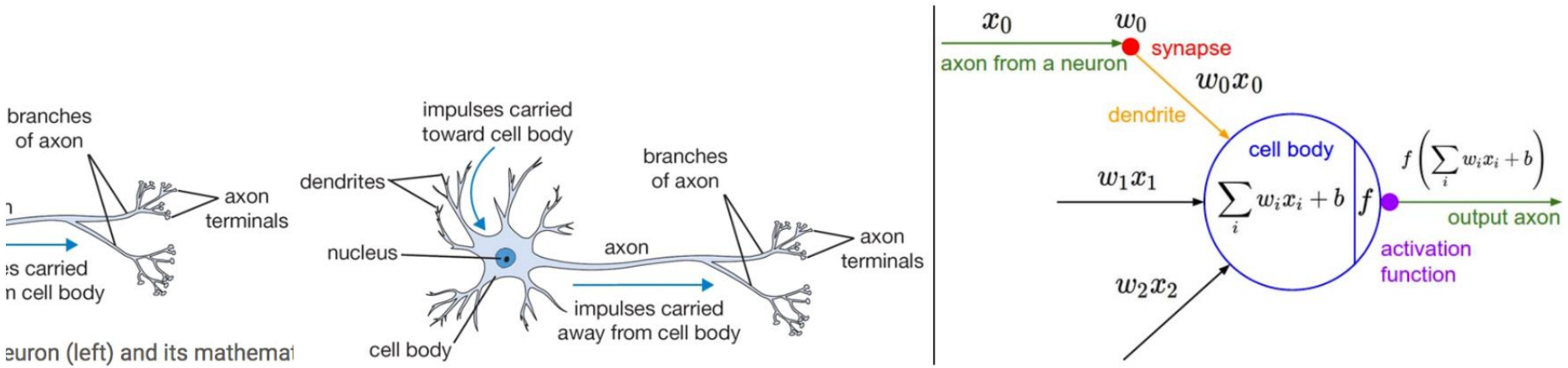
1. Keras의 개요
2. 뉴런의 개념
3. Dense layer
   - Dense
   - Dense로 표현되는 모델
     - 단순선형회귀모델, 다중선형회귀모델, 로지스틱회귀모델,softmaxclassification
4. Keras로 구현되는 모델의 개발절차
5. 연습문제
   - EX01 A simple linear regression example in Keras
     - modules
     - data 준비
     - 단순선형회귀모델개발
       - 모델 구조정의, 학습방법 설정, 학습
     - 모델 검증 및 예측결과 그래프로 출력(plotting)
   - Ex02. Exercise
   - EX03. Multivariable Linear Regression
   - EX04. Linear regression with data from file

1. Keras의 개요

- 케라스(Keras)는 파이썬으로 작성된 오픈 소스 신경망 라이브러리이다.
  - MXNet, Deeplearning, 텐서플로, Microsoft Cognitive Toolkit 또는 Theano 위에서 수행할 수 있다.
  - 딥 신경망과의 빠른 실험을 가능케 하도록 설계되었으며 최소한의 모듈 방식의 확장 가능성에 초점을 둔다.
  - ONEIROS(Open-ended Neuro-Electronic Intelligent Robot Operating System) 프로젝트의 연구의 일환으로 개발되었으며[3] 주 개발자이자 유지보수자는 구글의 엔지니어 Francois Chollet이다.
  - 2017년, 구글의 텐서플로 팀은 텐서플로의 코어 라이브러리에 케라스를 지원하기로 결정하였다. Chollet은 케라스가 단대단(end-to-end) 기계 학습 프레임워크가 아닌 인터페이스의 역할을 염두에 두었다.
  - 더 높은 수준의 더 직관적인 추상화 집합을 표현함으로써 백엔드 과학 컴퓨팅 라이브러리임에도 불구하고 신경망을 구성하기 쉽게 만들어준다.
  - 마이크로소프트 또한 CNTK 백엔드를 케라스에 추가하는 작업을 수행하고 있으며 기능은 현재 CNTK 2.0과 더불어 베타 릴리스 단계에 있다.

2. 뉴런의 개념

- 케라스의 핵심 데이터 구조는 모델이고, 이 모델을 구성하는 것이다. Dense 레이어는 뉴런을 구현하도록 설계되었다. 기본 개념, 역할 등에 대해서 살펴본다.
- 신경계를 모사한 뉴런
  - 인간의 뉴런과 이를 모델링한 퍼셉트론
    - 뉴런 axon(축삭돌기) dendrite(수상돌기)/ synapse – nucleus(핵) – axon(축삭돌기)
    - 퍼셉트론 x_0, x_1, x_2 x_0w_0, x_1w_1, x_2w_2 $y=f(\sum x_iw_i + b)$ y

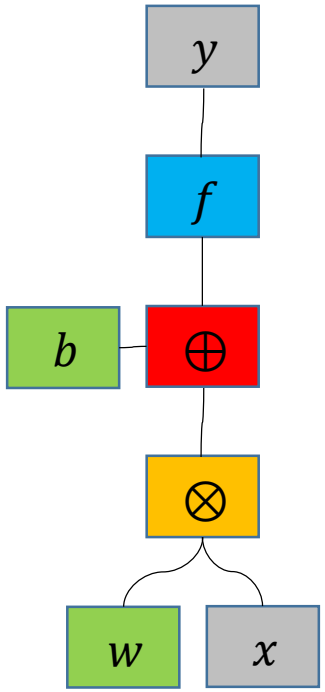


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

(출처: http://cs231n.github.io/neural-networks-1/)

2. 뉴런(퍼셉트론)의 개념

- 하나의 뉴런에서 나온 신호(x)가 다른 하나의 뉴런에 전달되어 출력(y)이 만들어지는 개념
- 한 개의 입력 받아 하나의 출력으로 전달하는 뉴런(퍼셉트론)의 모델



$y = f(wx + b)$

활성화함수(activation), linear, sigmod, relu

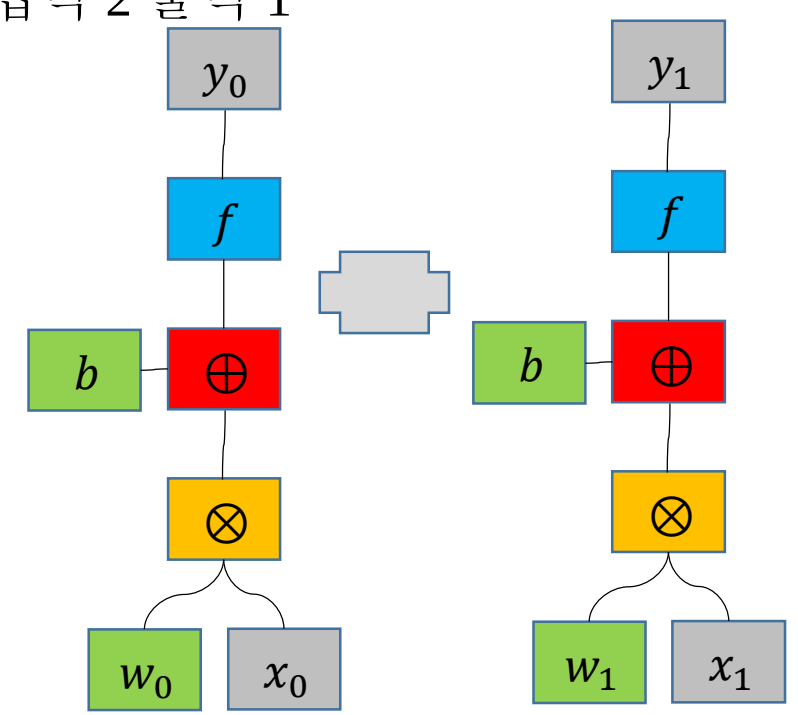
$wx + b$

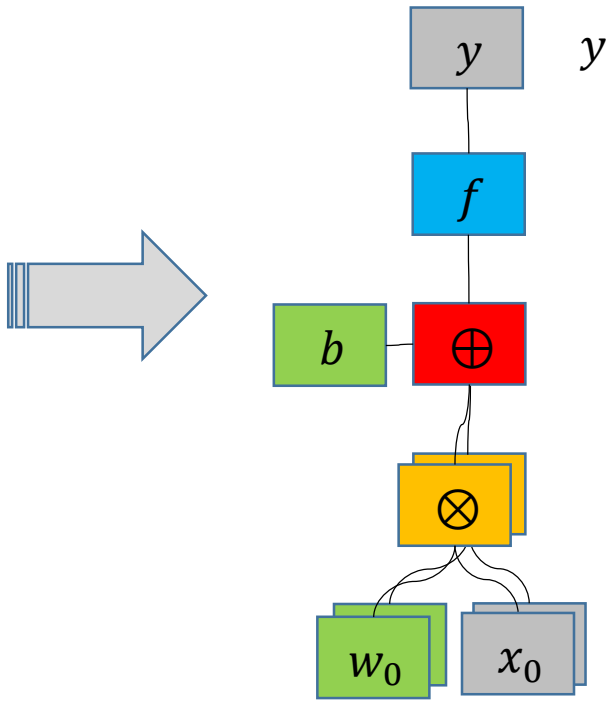
wx

Dense(units=1,input_dim=1)

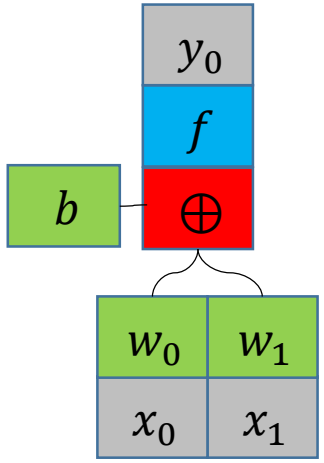
2. 뉴런의 개념

- 두개의 뉴런에서 나온 신호들(x_0 , x_1)가 하나의 뉴런에 전달되어 출력(y)이 만들어 지는 개념
- 입력 2 출력 1





$$y = f(w_0x_0 + w_1x_1 + b)$$



Dense(units=1,input_dim=1)

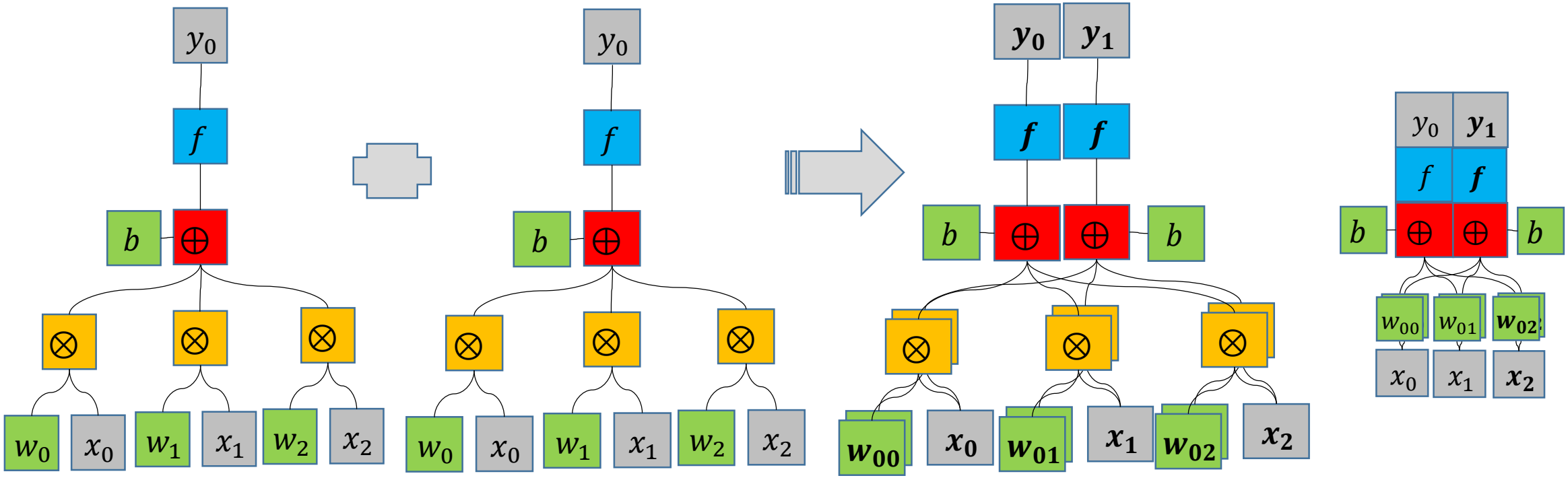
Dense(units=1,input_dim=1)

Dense(units=1,input_dim=2)

2. 뉴런의 개념

- 3개의 뉴런에서 나온 신호(x)가 두개 뉴런에 전달되어 출력(y)이 만들어지는 과정의 개념
- 입력 3 출력 2

$$y_0 = f(w_{00}x_0 + w_{01}x_1 + b_0)$$
$$y_1 = f(w_{10}x_0 + w_{11}x_1 + b_1)$$



Dense(units=1,input_dim=3)

Dense(units=1,input_dim=3)

Dense(units=2,input_dim=3)

3. Dense layer

- keras.layers.Dense(units,
  activation=None,
  use_bias=True,
  kernel_initializer='glorot_uniform',
  bias_initializer='zeros',
  kernel_regularizer=None,
  bias_regularizer=None,
  activity_regularizer=None,
  kernel_constraint=None,
  bias_constraint=None,
  **kwargs)
- Densely(fully) connected NN layer
- output = activation(dot(input, kernel) + bias) 연산의 구현
- activation은 activation 인자에 따른 요소별 활성화 함수이고(linear,sigmoid,softmax)
- kernel 은 layer를 구성하는 weights matrix이고,
- bias 는 layer에 의해 생성된 bias 벡터이다.(use_bias 가 True 일 경우만 적용됨).
- 입력의 차원이 2보다 크면 평탄화(flattened)된 후 연산 된다.

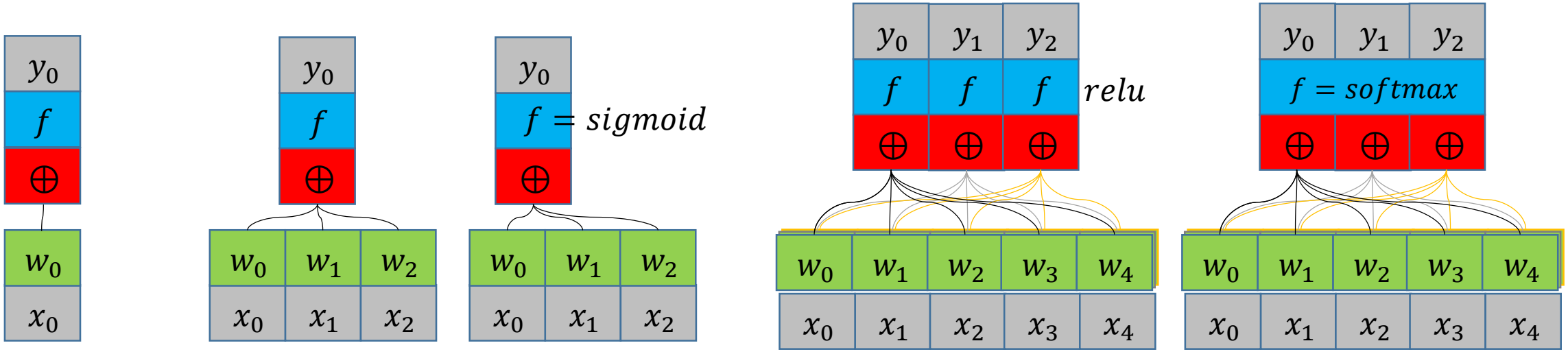
3. Dense layer

- 인수
  - units : 출력 뉴런의 수, 양의 정수, 출력 공간의 차원.
  - Activation : 사용할 활성화 기능. 아무것도 지정하지 않으면 활성화가 적용되지 않습니다 (예 : " 선형 " 활성화 : a(x) = x).
  - use_bias : 레이어가 바이어스 벡터를 사용하는지 여부를 나타내는 부울입니다.(True)
  - kernel_initializer: 커널 가중치 행렬의 이니셜 라이저.
    - 'glorot_uniform': Xavier uniform initializer,'random_uniform','uniform':균일분포, 'normal':가우시안분포
  - bias_initializer : 바이어스 벡터의 이니셜 라이저.
  - kernel_regularizer : 커널 가중치 행렬에 적용되는 정규화 기능.
  - bias_regularizer : 바이어스 벡터에 적용되는 정규화 기능.
  - activity_regularizer: 레이어의 출력에 적용되는 정규화 기능 ("활성화").
  - kernel_constraint : 커널 가중치 행렬에 적용되는 제약 함수.
  - bias_constraint : 바이어스 벡터에 적용된 구속 함수.
  - input_dim: integer,입력데이터의 차원

3. Dense layer

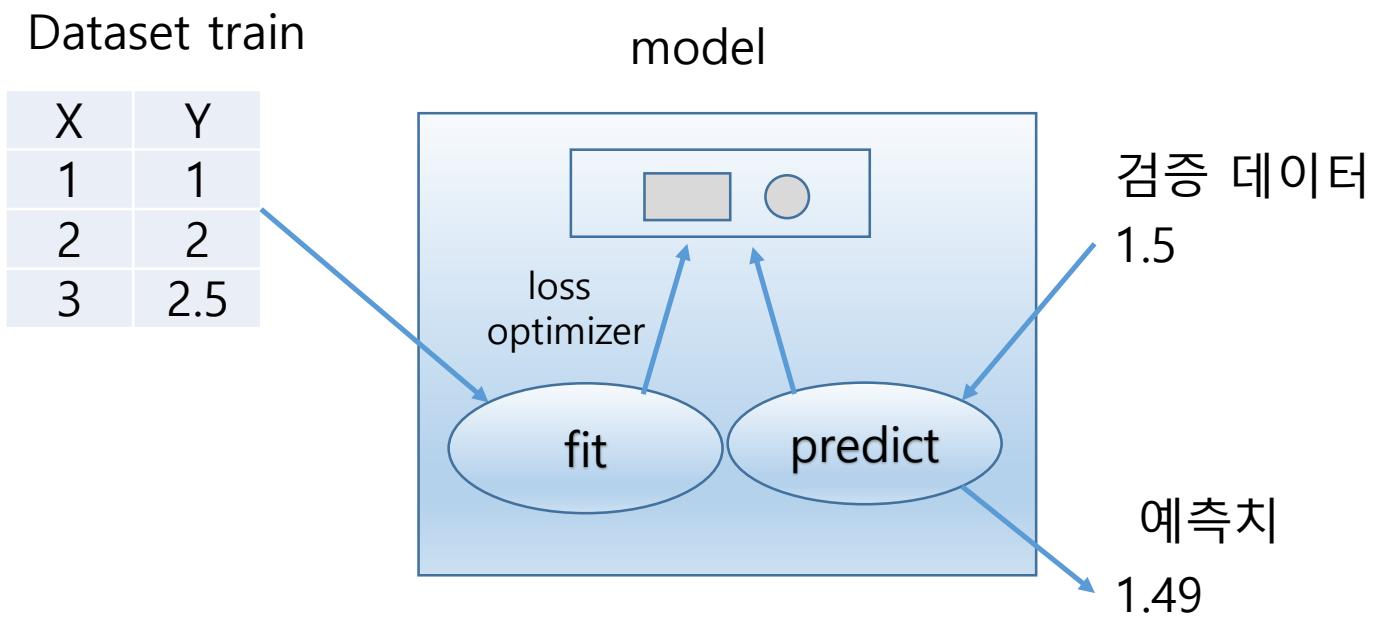
Dense로 표현되는 모델의 종류

- Dense(units=1,input_dim=1) #단순선형회귀모델 출력범위:[-∞ ~ + ∞]
- Dense(units=1,input_dim=3) #다중선형회귀모델 출력범위:[-∞ ~ + ∞]
- Dense(units=1,input_dim=3,activation='sigmoid')
  #다중로지스틱회귀모델(binary classification) 출력범위 [0.0~1.0]
- Dense(units=3,input_dim=5,activation='relu') #출력범위:[0.0 ~ + ∞]
- Dense(units=3,input_dim=5,activation='softmax') #출력이 확률로, 출력의 합=1.0



4. Keras로 구현되는 모델의 개발절차

- 개발 절차
- 데이터셋 생성
- 모델개발
  - model 정의
  - 학습방법 설정
    - loss 및 optimizer
  - 학습
    - fitting
- 모델검증(predict)
  - 학습된 모델의 성능 분석
  - model.predict([1])
  - model.predict([1.5])
    model.predict([1,2])
    model.predict(dataX)



Examples

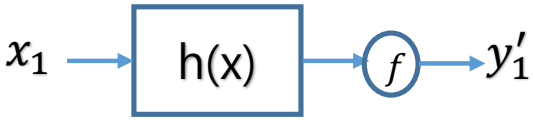
- EX01 A simple linear regression example in Keras
  - modules
  - data 준비
  - 단순선형회귀모델개발
    - 모델 구조정의, 학습방법 설정, 학습
  - 모델 검증 및 예측결과 그래프로 출력(plotting)
- Ex02. Exercise
- EX03. Multivariable Linear Regression
- EX04. Linear regression with data from file

Ex01. A simple linear regression example in Keras

- Example구현에 필요한 Library
  - from keras.models import Sequential
    - 모델정의 클래스
  - From keras.layers import Dense
    - Densely connected layer
  - Import keras.optimizers as optimizers
    - 최적화기,학습기
  - Import tensorflow as tf
    - TensorFlow module
  - Import matplotlib.pylab as plt
    - Graph plotting module

Ex01 . A simple linear regression in Keras model

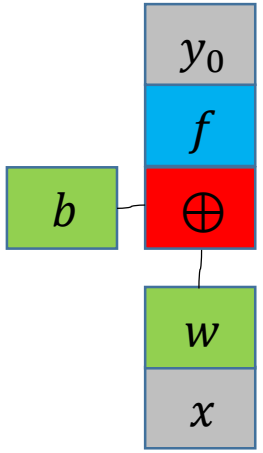
- Dataset 생성
  - 학습용 X and Y data
  - trainX = [1, 2, 3]
  - trainY = [1, 2, 2.5]
- 모델개발
  - 모델 구조정의



  - 단순선형회귀모델(simple linear regression model) – a perceptron
  - $h(x) = wx + b$
  - $y_1' = h(x)$

```
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import tensorflow as tf
import matplotlib.pylab as plt
```

모델구조



$y_0 = f(h(x)) = f(wx + b)$

$h(x) = wx + b$

Code in keras

Dense(units=1,input_dim=1,activation='linear')

Ex01. A simple linear regression in Keras mo

```
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import tensorflow as tf
import matplotlib.pylab as plt
```

- 모델 구조 설정
  - # simple linear regression model
  - Model=Sequential()
  - Model.add(
    Dense(units=1, #출력 노드 수 $y_0 = f(h(x)) = wx + b$
    input_dim=1, #입력차원 (None,1)(1,1)=>(None,1)
    activation= ' linear')#선형(기본)
)
  - Model.summary() #model의 구조 출력

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

- 모델 학습방법 설정(loss 및 optimizer)
  - model.compile(loss= 'mse' , # 손실함수설정: mean_squared_error '
    optimizer= 'adam' # 학습기 설정 : adam optimizer
)

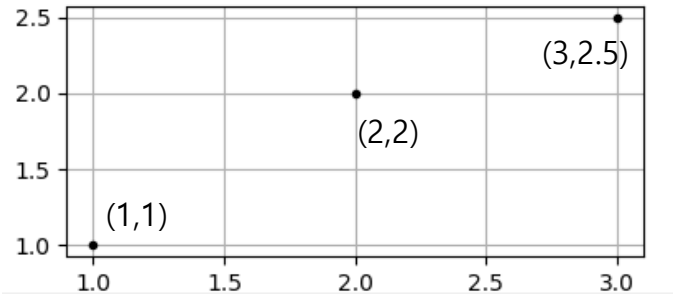
$$\text{trainX} = \{x_i\}$$
$$trainY = \{y_i\}$$
$$\hat{y}_i = model.predict(x_i)$$
$$loss = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$
$$adam.optimize(loss)$$

Ex01. A simple linear regression example in Keras

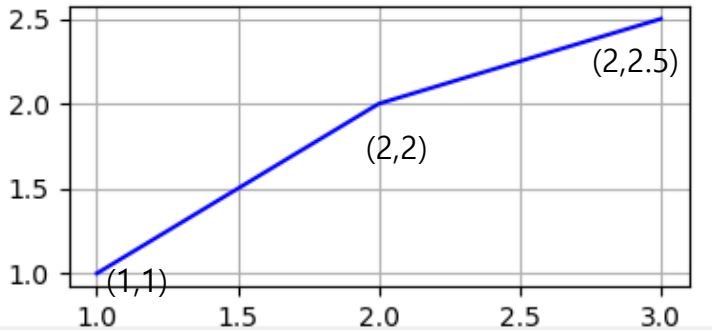
- 모델 학습
  - model.fit(trainX,trainY, #학습용 data
    epochs=200, #200회 반복학습
    verbose=1) #학습과정 정보 출력량 설정 0,1,2
- 모델 검증
  - 학습된 모델의 성능 분석
  - model.predict([1.0]) # 입력의 차원 1이 맞아야 한다, 출력shape=(none,1)
    model.predict([1.0,2.0]) #[[1.1],[2.1]]
    model.predict(dataX)

Ex01. A simple linear regression example in Keras

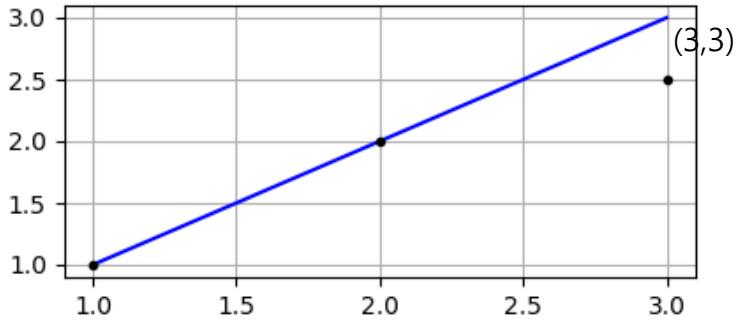
- 데이터를 그래프로 출력하기
  - plot data on screen
    - import matplotlib.pylab as plt
    - plt.plot(X,Y,color,…)
    - plt.plot([1,2,3],[1,2,2.5],’k.’)
      #점 출력(1,1),(2,2),(3,2.5)



    - plt.plot([1,2,3],[1,2,2.5],’b’)
      #선으로 출력

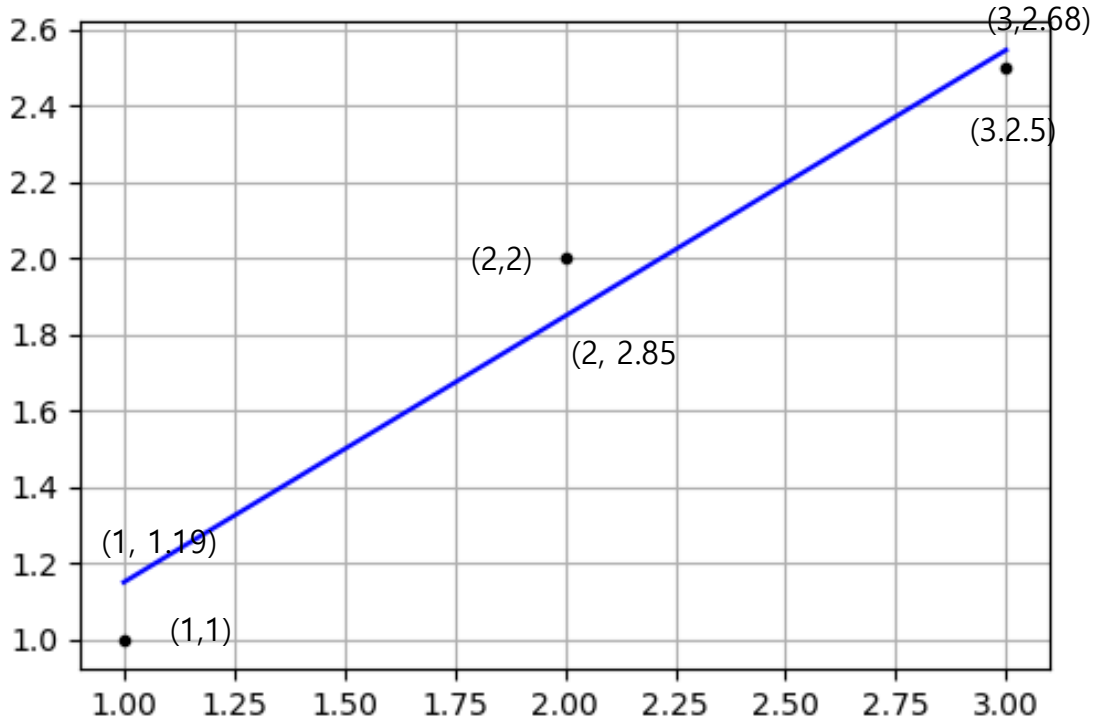


    - plt.plot([1,2,3],[1,2,3],’b’, [1,2,3],[1,2,2.5],’k.’)
      #중복출력



Ex01. A simple linear regression example in Keras

- Plot train data and predicted data
  - import matplotlib.pylab as plt
  - Train data
    - trainX = [1, 2, 3]
    - trainY = [1, 2, 2.5]
    - predY=model.predict(trainX)
      #predX=trainX
      #predY=[[1.1904] [1.8550] [2.6851]]



  - plt.plot(trainX, model.predict(trainX), 'b', trainX,trainY, 'k.')
  - plt.grid()
  - plt.show()

Ex01. A simple linear regression example in Keras

```
#데이터셋 생성
trainX = [1, 2, 3]
trainY = [1, 2, 2.5]

#model 정의
model=Sequential()
model.add(Dense(1, input_dim=1))
model.summary()
model.compile(loss= 'mse',optimizer= 'adam' )

#fitting(학습)
Model.fit(trainX,trainY,epochs=2000,verbose=1)

print(model.predict([1.5]))
Print(model.predict([1.0, 2.0, 3.0]))

#예측치와 학습용데이터 그래프출력
plt.plot(trainX, model.predict(trainX), 'b', trainX,trainY, 'k.')
plt.show()
```

```
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import tensorflow as tf
import matplotlib.pylab as plt
```

#단순선형회귀모델 $y_0 = f(h(x)) = wx + b$

#model weights 출력

#손실함수, 학습기

#학습데이터, 학습반복회수,학습정보출력량

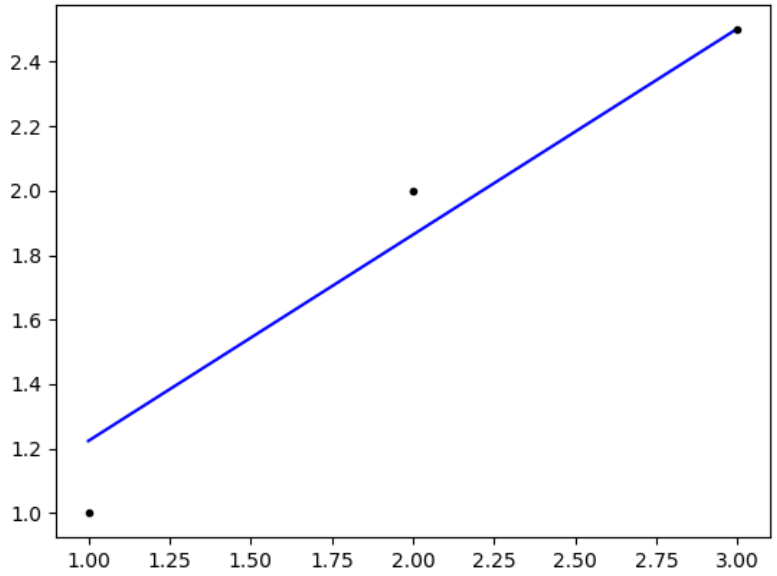
#검증 예측값 계산 [[1.543674]]

#검증 예측값 계산 [[1.2484958] [1.868777] [2.4890585]]

Ex01. A simple linear regression example in Keras

```
#데이터셋 생성
trainX = [1, 2, 3]
trainY = [1, 2, 2.5]

#model 정의
model=Sequential()
model.add(Dense(1, input_dim=1))                    #단순선
model.summary()                                     #model v
model.compile(loss= 'mse',optimizer= 'adam' )#손실함

#fitting(학습)
Model.fit(trainX,trainY,epochs=2000,verbose=1) #학습데이터, 학습반복회수,

print(model.predict([1.5]))          #검증 예측값 계산  [[1.543674]]
Print(model.predict([1.0, 2.0, 3.0]))#검증 예측값 계산  [[1.2484958]  [1.

#예측치와 학습용데이터 그래프출력
plt.plot(trainX, model.predict(trainX), 'b', trainX,trainY, 'k.')
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
Epoch 1/200
3/3 [==============================] - 3s 1s/step - loss: 0.9372 - acc: 0.0000e+00
Epoch 2/200
3/3 [==============================] - 0s 997us/step - loss: 0.7445 - acc: 0.3333
Epoch 3/200
3/3 [==============================] - 0s 997us/step - loss: 0.5923 - acc: 0.3333
Epoch 198/200
3/3 [==============================] - 0s 998us/step - loss: 0.0155 - acc: 0.6667
Epoch 199/200
3/3 [==============================] - 0s 997us/step - loss: 0.0155 - acc: 0.6667
Epoch 200/200
3/3 [==============================] - 0s 997us/step - loss: 0.0155 - acc: 0.6667
```



Ex02. Exercise

- 다음 학습시간의 데이터를 이용하여 시험점수를 예측하는 선형회귀모델
  - 데이터준비 : 학습 및 검증데이터
  - 모델구조 설정
    - 모델정의 : Linear regression model을 정의
  - 모델학습방법 설정
    - loss, optimizer
  - 모델 학습
  - 모델 검증
    - 검증데이터(validation data)의 예측 점수를 계산하시오.
    - Train data (X,Y)와 validation data(X,Y)를 graph로 그리고 비교 분석하시오.

Train data

X(hours)	Y(scores)
10	90
9	80
3	50
2	30

Validation data

X(hours)	$prediction\ \bar{Y}$
10	
9.5	
1.5	
2	

EX03. Multivariable Linear Regression

- (univariable) Linear Regression
- Data

x	y
1	1
2	2
3	2.5

- Model 정의
  - $H(\mathrm{x}|\mathrm{w}, b) = \mathrm{xw} + b$
- 학습
- 검증

```
(Univariable) Linear Regression
#dataset 생성
X=[1,                  X=np.array([[1],
   2,                              [2],
   3])                             [3]])
Y= [1,                 Y=np.array([[1],
    2,                             [2],
   2.5])                           [2.5]])

#model 정의
model=Sequential()
model.add(Dense(1,     input_dim=1)) #(None,1)(1,1)=(None,1)
model.summary()                      #model 구조(weights) 출력
Model.compile(loss='mse',            #loss, optimizer 설정
              optimizer= ' adam')

#model  학습
Model.fit(X,Y,epochs=2000,verbose=1)      #학습

#model  평가
p=model.predict(np.array([[90, 90, 90]]))   # 검증예측값계산  [[178.51509]]
print(p)
```

EX03. Multivariable Linear Regression

- Dataset 생성

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

- Model 정의
  - $H(X|W,b) = XW + b = [x_0\ x_1\ x_2]\begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} + \mathrm{b}$

- 학습
- 검증

```
X=np.array([[73,80,75],
            [93,88,93],
            [89,91,90],
            [96,98,100],
            [73,66,70]])
Y=np.array([[152],
            [185],
            [180],
            [196],
            [142]])

#model 정의
model=Sequential()
model.add(Dense(1,    input_dim=3)) #(None,3)(3,1)=(None,1)
model.summary()                     #model 구조(weights) 출력
Model.compile(loss='mse',           #loss, optimizer 설정
              optimizer= ' adam')

Model.fit(X,Y,epochs=2000,verbose=1)      #학습

p=model.predict(np.array([[90, 90, 90]]))   # 검증예측값계산  [[178.51509]]
print(p)

print(model.layers[0].get_weights())
#W:[[0.35855338],      [1.2938051 ],      [0.35843384]]
#b: [0.37252703]
```

EX03. Multivariable Linear Regression

```
#lec41KL multivariable Linear regression
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras.layers.core as core

X=np.array([[73,80,75],
    [93,88,93],
    [89,91,90],
    [96,98,100],
    [73,66,70]])
Y=np.array([[152],
    [185],
    [180],
    [196],
    [142]])
model=Sequential()                          #model 정의
model.add(Dense(1,activation='linear',      #(None,3)(3,1)=(None,1)
        input_dim=3))
model.summary()                             #model weights 출력
model.compile(loss='mse',                   #loss, optimizer 설정
  optimizer='adam')

model.fit(X,Y,epochs=2000,verbose=1)   #학습

p=model.predict(np.array([[90, 90, 90]]))#예측값계산  [[178.51509]]
print(p)
```

```
model=Sequential()                          #model 정의
model.add(Dense(1,activation='linear',      #(None,3)(3,1)=(None,1)
        input_dim=3))
model.summary()                             #model weights 출력
model.compile(loss='mse',                   #loss, optimizer 설정
  optimizer='adam')

model.fit(X,Y,epochs=2000,verbose=1)   #학습

p=model.predict(np.array([[90, 90, 90]]))#예측값계산  [[178.51509]]
print(p)

print(model.layers[0].get_weights())
#W:[[0.35855338],      [1.2938051 ],      [0.35843384]]
#b: [0.37252703]
```

EX03. Multivariable Linear Regression

```
#lec41KL multivariable Linear regression
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras.layers.core as core

X=np.array([[73,80,75],
   [93,88,93],
   [89,91,90],
   [96,98,100],
   [73,66,70]])
Y=np.array([[152],
   [185],
   [180],
   [196],
   [142]])
model=Sequential()                          #model 정의
model.add(Dense(1,activation='linear' ,  #(None,3)(3,1)=(None,1
        input_dim=3))
model.summary()                             #model weights 출력
model.compile(loss= 'mse' ,                 #loss, optimizer 설정
  optimizer='adam' )

model.fit(X,Y,epochs=2000,verbose=1)     #학습

p=model.predict(np.array([[90, 90, 90]]))#예측값계산  [[178.51
print(p)
```

```
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2000
5/5 [==============================] - 3s 642ms/step - loss: 43979.3984
Epoch 2/2000
5/5 [==============================] - 0s 998us/step - loss: 43898.9336
Epoch 3/2000
5/5 [==============================] - 0s 798us/step - loss: 43806.2070
```

```
Epoch 1997/2000
5/5 [==============================] - 0s 399us/step - loss: 1.5561
Epoch 1998/2000
5/5 [==============================] - 0s 399us/step - loss: 1.5561
Epoch 1999/2000
5/5 [==============================] - 0s 399us/step - loss: 1.5561
Epoch 2000/2000
5/5 [==============================] - 0s 399us/step - loss: 1.5561
[[181.94154]]
Press any key to continue . . .
```

Ex04. Linear regression with data from file

- 데이터셋 생성
  - 파일에서 읽기
  - numpy.loadtxt

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101

- Linear regression 모델정의
- 학습
- 검증

Ex04. Multivariable linear regression

- Load data from file with numpy.loadtxt

파일의 내용 'data-01-test-score.csv'

```
73,80,75,152
93,88,93,185
89,91,90,180
96,98,100,196
73,66,70,142
53,46,55,101
```

```
XY = np.loadtxt('data-01-test-score.csv', delimiter=',', dtype=np.float32)
X = XY[:, :-1] # 모든 행, 마지막을 제외한 모든 열
Y = XY[:, -1] # 모든 행, 마지막 열
```

```
numpy library 배열 다루기
- Indexing, Slicing, Iterating

a = np.array(
      [[1, 2, 3, 4],
       [5, 6, 7, 8],
       [9, 10, 11, 12]])

print(a[:, 1])   # [2 6 10]

print(a[-1])    # [9 10 11 12]

print(a[-1, :]) # [9 10 11 12]

print(a[0:2, :])
#[[1 2 3 4]
# [5 6 7 8]]
```

```
print(XY.shape,XY)
XY:
(6, 4)
[[ 73.  80.  75. 152.]
 [ 93.  88.  93. 185.]
 [ 89.  91.  90. 180.]
 [ 96.  98. 100. 196.]
 [ 73.  66.  70. 142.]
 [ 53.  46.  55. 101.]]
```

```
print(X.shape,X)
X:
(6, 3)
[[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]]
```

```
print(Y.shape,Y)
Y:
(6,)
[[152.]
[185.]
[180.]
[196.]
[142.]
[101.]]
or
(6,) [[152.] [185.] [180.] [196.] [142.] [101.]]
```

Ex04. Multivariable linear regression

```
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import tensorflow as tf
import matplotlib.pylab as plt
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

#학습용 데이터 읽기
XY=np.loadtxt('data/data-01-test-score.txt',dtype=float,delimiter=',')
X=XY[:,:-1]
Y=XY[:,-1]
#print(XY.shape,XY); print(X.shape,X); print(Y.shape,Y)

#model 정의
model  =Sequential()
model.add(Dense(1, input_dim=3))       #(None,3)(3,1)=(None,1)
model.summary()                        #model 구조(weights) 출력
model.compile(loss='mse',optimizer='adam')    #loss, optimizer 설정

Model.fit(X,Y,epochs=2000,verbose=1) #학습

P=model.predict(np.array([[90, 90, 90]]))#예측값 계산  [[178.51509]]
Print(p)
```

Ex04. Multivariable linear regression

```
from keras.models import Sequential
from keras.layers import Dense
import keras.optimizers as optimizers
import tensorflow as tf
import matplotlib.pylab as plt
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

#학습용 데이터 읽기
XY=np.loadtxt('data/data-01-test-score.txt',dtype=float,delimiter=',')
X=XY[:,:-1]
Y=XY[:,-1]
#print(XY.shape,XY); print(X.shape,X); print(Y.shape,Y)

#model 정의
model=Sequential()
model.add(Dense(1, input_dim=3))        #(None,3)(3,1)=(None,1)
model.summary()                         #model 구조(weights) 출력
model.compile(loss='mse',optimizer='adam')    #loss, optimizer 설정

Model.fit(X,Y,epochs=2000,verbose=1) #학습

P=model.predict(np.array([[90, 90, 90]]))#예측값 계산  [[178.51509]]
Print(p)
```

```
(6, 4) [[ 73.  80.  75. 152.]
 [ 93.  88.  93. 185.]
 [ 89.  91.  90. 180.]
 [ 96.  98. 100. 196.]
 [ 73.  66.  70. 142.]
 [ 53.  46.  55. 101.]]
(6, 3) [[ 73.  80.  75.]
 [ 93.  88.  93.]
 [ 89.  91.  90.]
 [ 96.  98. 100.]
 [ 73.  66.  70.]
 [ 53.  46.  55.]]
(6,) [152. 185. 180. 196. 142. 101.]
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 4
=================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
_________________________________________________________________
Epoch 1/2000
6/6 [==============================] - 3s 575ms/step - loss: 46707.8438
Epoch 2/2000
6/6 [==============================] - 0s 832us/step - loss: 46629.4570
Epoch 3/2000
6/6 [==============================] - 0s 831us/step - loss: 46539.1133
```

```
Epoch 1997/2000
6/6 [==============================] - 0s 499us/step - loss: 6.9706
Epoch 1998/2000
6/6 [==============================] - 0s 499us/step - loss: 6.9370
Epoch 1999/2000
6/6 [==============================] - 0s 332us/step - loss: 6.9038
Epoch 2000/2000
6/6 [==============================] - 0s 332us/step - loss: 6.8707
[[178.45174]]
Press any key to continue . . .
```

Examples