Artificial Intelligence Portfolio

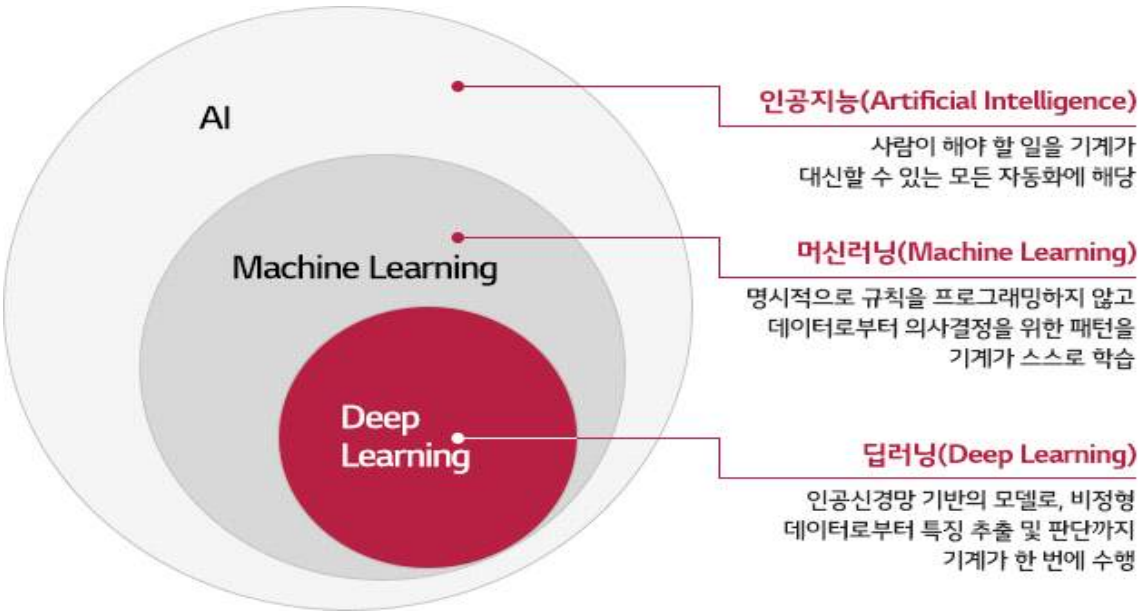
과목	인공지능응용프로그래밍
교수	강환수교수님
학과	컴퓨터정보공학과
학번	20191789
이름	윤한조

2020.11.20

Contents

1. 인공지능과 딥러닝 개요

1) 인공지능과 머신러닝과 딥러닝



인공지능 : 컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술

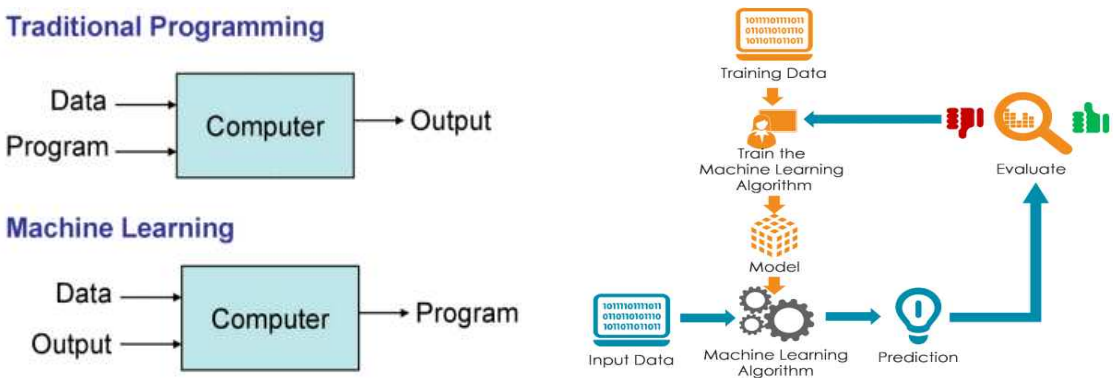
머신러닝 : 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야

딥러닝 : 다중 계층의 신경망 모델을 사용하는 머신러닝의 일종

: 특징과 데이터가 많을수록 딥러닝에 적합

2) 머신러닝

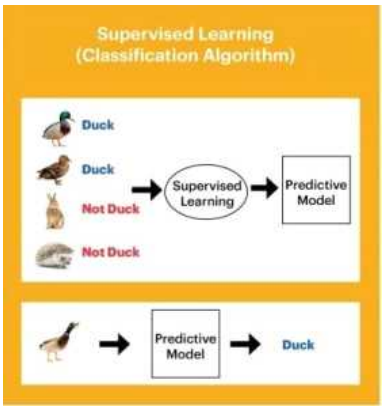
- 주어진 데이터를 기반으로 기계가 스스로 학습하여 성능을 향상 시키거나 최적의 해답을 찾기 위한 학습 지능 방법
- 스스로 데이터를 반복적으로 학습하여 기술을 터득하는 방식
- 명시적 프로그래밍을 하지 않아도 컴퓨터가 학습할 수 있도록 해주는 인공지능의 한 형태
- 많은 데이터 유입 → 컴퓨터 학습량↑ → 스마트↑ → 작업 수행 능력 & 정확도↑



3) 머신러닝의 분류

① 지도학습(supervised learning)

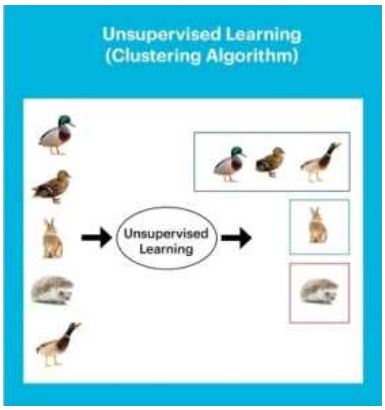
올바른 입력과 출력의 쌍으로 구성된 정답의 훈련 데이터로부터 입출력 간의 함수를 학습시키는 방법



k-최근접 이웃 (k-Nearest Neighbors)
선형 회귀 (Linear Regression)
로지스틱 회귀 (Logistic Regression)
서포트 벡터 머신 (Support Vector Machines (SVM))
결정 트리 (Decision Tree)와 랜덤 포레스트 (Random Forests)

② 자율학습(unsupervised learning)

정답이 없는 훈련 데이터(unlabeled data)를 사용하여 데이터 내에 숨어있는 어떤 관계를 찾아내는 방법



군집	k-평균 (k-Means)
	계층 군집 분석 (Hierarchical Cluster Analysis (HCA))
	기댓값 최대화 (Expectation Maximization)
시각화와 차원축소	주성분 분석 (Principal Component Analysis (PCA))
	커널 (kernel PCA)
	지역적 선형 임베딩 (Locally-Linear Embedding (LLE))
연관 규칙 학습	어프라이어리 (Apriori)
	이클렛 (Eclat)

③ 강화학습(reinforcement learning)

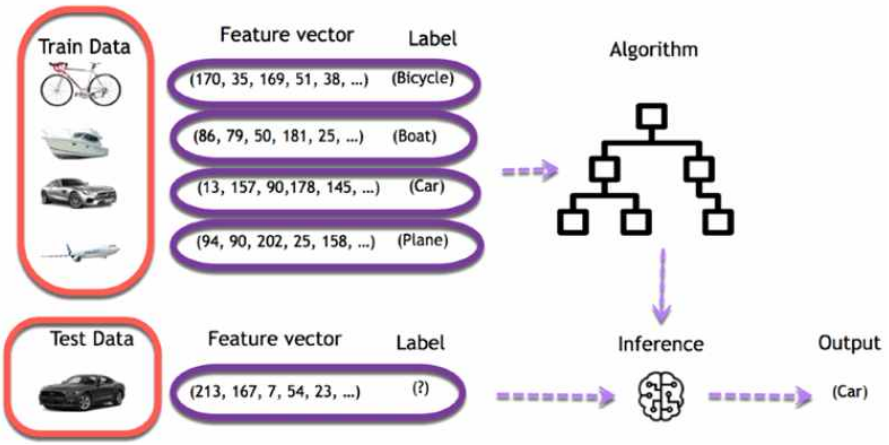
- 잘한 행동에 대해 보상을 주고 잘못한 행동에 대해 벌을 주는 경험을 통해 지식을 학습
- Agent가 어떤 행동을 해야 많은 보상을 받을 수 있는지 찾아내는 방법으로 학습 데이터 없이 스스로의 시행 착오만으로 학습을 진행

ex) 딥마닝의 알파고, 자동 게임 분야

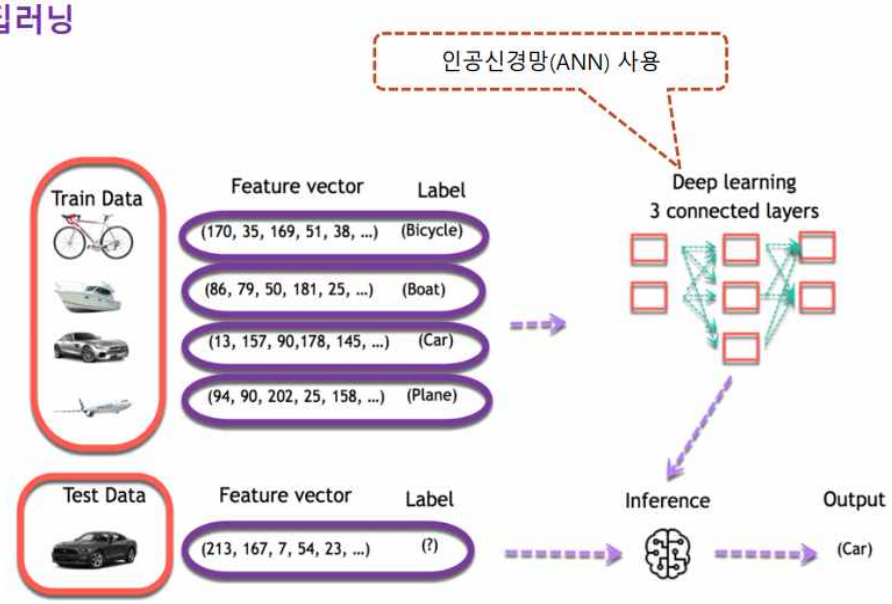
4) 머신러닝과 딥러닝 비교

	머신러닝	딥러닝
데이터 의존성	중소형 데이터 세트에서 탁월한 성능	큰 데이터 세트에서 뛰어난 성능
하드웨어 의존성	저가형 머신	GPU가 있는 강력한 기계가 필요 (DL은 상당한 양의 행렬 곱셈을 수행)
기능공학	데이터를 나타내는 기능을 이해	데이터를 나타내는 최고의 기능을 이해할 필요X
실행시간	몇 분에서 몇 시간	최대 몇 주 (신경망은 상당한 수의 가중치를 계산)

머신러닝

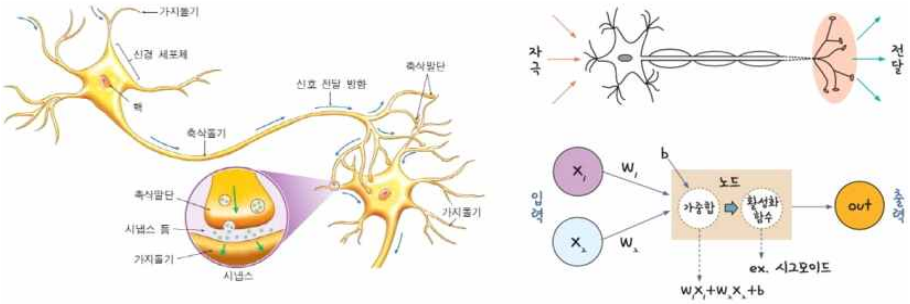


딥러닝



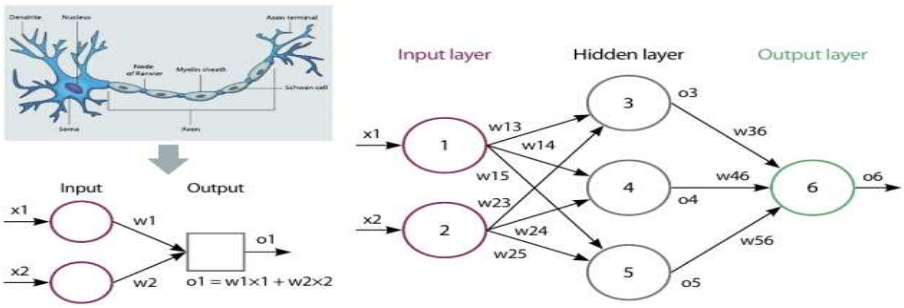
5) 인공신경망 (ANN : Artificial Neural Network)

- 신경세포 뉴런(Neuron)의 동작 원리에 기초한 기술로, 이를 모방하여 만든 가상의 신경
- 뇌와 유사한 방식으로 입력되는 정보를 학습하고 판별하는 신경 모델



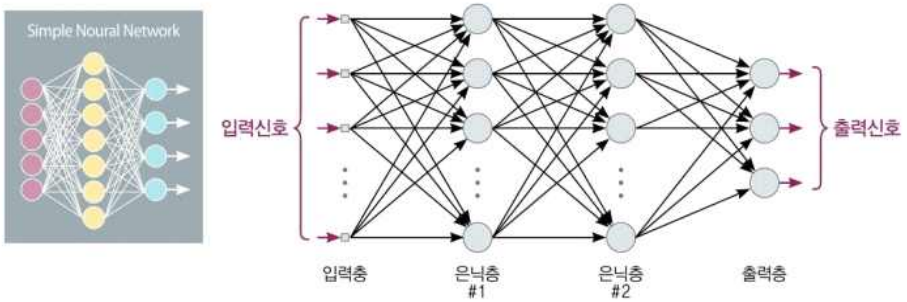
6) 인공신경망 구조와 MLP(Multi Layer Perceptron)

- 입력층(input layer)과 출력층(output layer)
  : 다수의 신호(input)를 입력 받아서 하나의 신호(output)를 출력
- 중간의 은닉층(hidden layer) : 여러 개의 층으로 연결하여 하나의 신경망을 구성



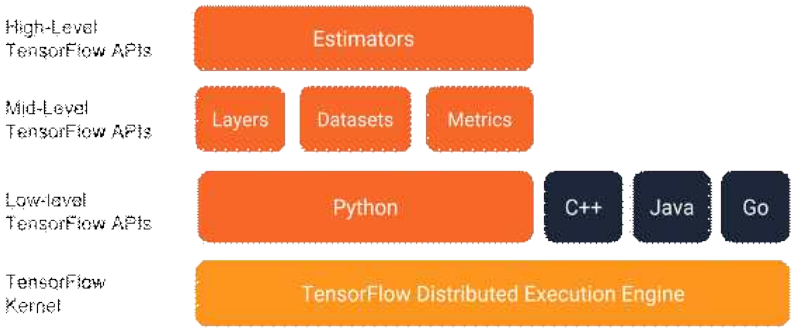
7) 심층신경망 (DNN : Deep Neural Network)

- 다중 계층인 심층신경망(deep neural network)을 사용
- 학습 성능을 높이는 고유 특징들만 스스로 추출하여 학습하는 알고리즘
- 입력 값에 대해 여러 단계의 심층신경망을 거쳐 자율적으로 사고 및 결론 도출



2. 텐서플로 이해

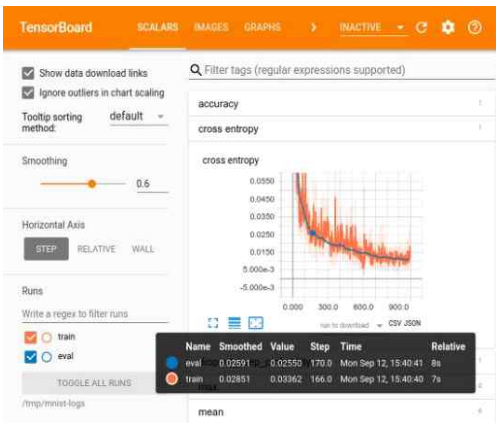
1) 텐서플로(TensorFlow) 개요



- 구글(Google)에서 만든 라이브러리로, 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리
- 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공
  (데스크톱, 모바일, 웹, 클라우드 개발용 API를 제공)
- 텐서플로 자체는 기본적으로 C++로 구현 (Python, Java, Go 등 다양한 언어를 지원)
- 파이썬을 최우선으로 지원 : 대부분의 편한 기능들이 파이썬 라이브러리로만 구현

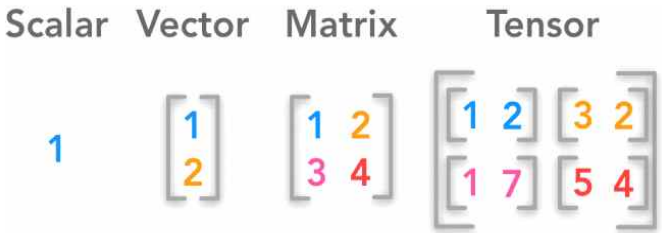
2) 텐서보드(TensorBoard)

브라우저에서 실행 가능한 시각화 도우미. 딥러닝 학습 과정을 추적하는데 유용하게 사용



3) 텐서 개요

텐서(Tensor) : 모든 데이터



딥러닝에서 데이터를 표현하는 방식	0-D 텐서	스칼라	10
	1-D 텐서	벡터	[10, 20, 30]
	2-D 텐서	행렬	[[1, 2, 3], [4, 5, 6]]
n차원 행렬(배열)	텐서는 행렬로 표현할 수 있는 n차원 형태의 배열을 높은 차원으로 확장 [[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]		

4) TensorFlow 계산 과정

- 모두 그래프(Graph)라고 부르는 객체 내에 저장되어 실행
- 그래프를 계산하려면 외부 컴퓨터에 이 그래프 정보를 전달하고 그 결과값을 받아야 함

Session

- 이 통신과정을 담당하는 것이 세션(Session)이라고 부르는 객체
- 생성, 사용, 종료 과정이 필요

① 세션 생성 : Session 객체 생성

② 세션 사용 : run 메서드에 그래프를 입력하면 출력값을 계산하여 반환

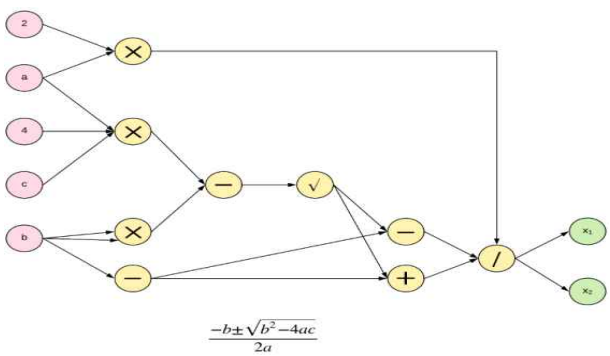
③ 세션 종료 : close 메서드

: with 문을 사용하면 명시적으로 호출 불필요

```
x = tf.constant(3)
y = x**2

sess = tf.Session()
print(sess.run(x))
print(sess.run(y))
sess.close()
```

5) 데이터 흐름 그래프(dataflow graph)

- 텐서 형태의 데이터들이 딥러닝 모델을 구성하는 연산들의 그래프를 따라 흐르면서 연산
- 딥러닝에서 데이터를 의미하는 Tensor와 DataFlow Graph를 따라 연산이 수행되는 형태(Flow)를 합쳐 TensorFlow란 이름이 나오게 됨



3. 텐서플로 기초 프로그래밍

[버전 사용]

버전 사용 방법	%tensorflow_version 1.x
	%tensorflow_version 2.x
Import 하기 전	위 매직 명령어 사용
	사용 중에 바꾸려면 '런타임 다시 시작' 후 바로 - %tensorflow_version 1.x - %tensorflow_version 2.x
2.2 사용 중에 1.x으로 변경	런타임 다시 시작(ctrl+M.) 후 바로 실행 - %tensorflow_version 1.x

[코랩에서 1.x 사용 지정]

- Hello World에서 시작
- Session : 그래프를 실행시키는 객체
  : 만들어진 그래프에 실제 값의 흐름을 수행해 결과가 나오도록 하는 객체

```
%tensorflow_version 1.x

import tensorflow as tf

hello = tf.constant('Hello World!')

sess = tf.Session()
print(sess.run(hello))
sess.close()

a = 10
b = 10
print(tf.add(a, b))

sess = tf.Session()
print(sess.run(tf.add(a, b)))
sess.close()
```

[텐서플로 2.0으로 실행]

- 이미 1.x을 사용 중이라면 런타임 다시 시작 후 그대로 import
- 값만 보려면 메소드 numpy() 사용

```
[1] import tensorflow as tf
    tf.__version__

    '2.3.0'

[2] a = tf.constant(5)
    b = tf.constant(3)
    print(a+b)

    tf.Tensor(8, shape=(), dtype=int32)

[3] c = tf.constant('Hello, world!')
    print(c)
    print(c.numpy())

    tf.Tensor(b'Hello, world!', shape=(), dtype=string)
    b'Hello, world!'
```

[Tensor Shapes, type]

```
t = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	....

```
[12] a = tf.constant([1, 2, 3])
     a.shape

     TensorShape([3])

[13] a = tf.constant([[1, 2, 3], [4, 5, 6]])
     a.shape

     TensorShape([2, 3])

[14] a = tf.constant([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
     a.shape

     TensorShape([2, 2, 3])

[20] a

     <tf.Tensor: shape=(2, 2, 3), dtype=int32, numpy=
     array([[[1, 2, 3],
             [4, 5, 6]],

            [[1, 2, 3],
             [4, 5, 6]]], dtype=int32)>
```

[조건 연산 tf.cond()]

- tf.cond(pred, true_fn=None, false_fn=None, name=None)
- pred를 검사해 참이면 true_fc 반환, 거짓이면 false_fc 반환

```
[6] x = tf.constant(1.)
    bool = tf.constant(True)
    res = tf.cond(bool, lambda: tf.add(x, 1.), lambda: tf.add(x, 10.))

    print(res)
    print(res.numpy())

    tf.Tensor(2.0, shape=(), dtype=float32)
    2.0


[7] x = tf.constant(2)
    y = tf.constant(5)
    def f1(): return tf.multiply(x, 17)
    def f2(): return tf.add(y, 23)
    r = tf.cond(tf.less(x, y), f1, f2)

    r.numpy()

    34
```

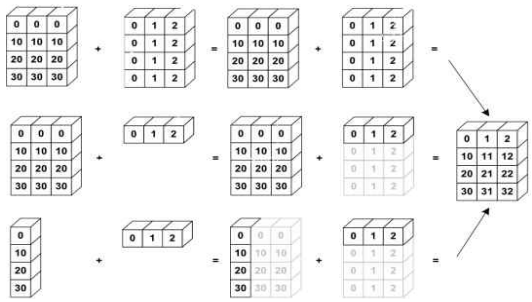
[1차원 배열 텐서]

```
[14] # 1차원 배열 텐서
     t = tf.constant([1, 2, 3])
     t

     <tf.Tensor: shape=(3,), dtype=int32, numpy=array([1, 2, 3], dtype=int32)>

[15] x = tf.constant([1, 2, 3])
     y = tf.constant([5, 6, 7])

     print((x+y).numpy())

     [ 6  8 10]

[16] a = tf.constant([5], dtype=tf.float32)
     b = tf.constant([10], dtype=tf.float32)
     c = tf.constant([2], dtype=tf.float32)
     print(a.numpy())

     d = a * b + c

     print(d)
     print(d.numpy())

     [5.]
     tf.Tensor([52.], shape=(1,), dtype=float32)
     [52.]
```

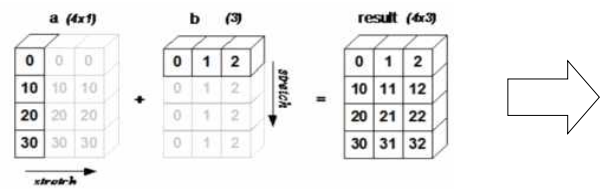
[배열 텐서 연산]

텐서의 브로드캐스팅

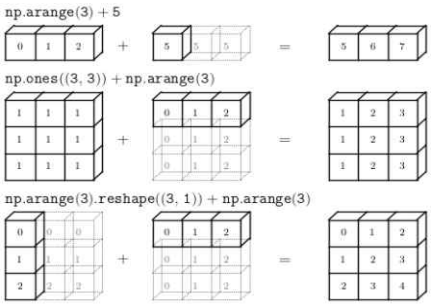
: Shape이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용하여 Shape을 맞춤



Numpy	mp.arange()




```
[17] x = tf.constant([[0], [10], [20], [30]])
     y = tf.constant([0, 1, 2])

     print((x+y).numpy())

     [[ 0  1  2]
      [10 11 12]
      [20 21 22]
      [30 31 32]]

[44] import numpy as np

     print(np.arange(3))
     print(np.ones((3, 3)))
     print()

     x = tf.constant((np.arange(3)))
     y = tf.constant([5], dtype=tf.int64)
     print(x)
     print(y)
     print(x+y)

     [0 1 2]
     [[1. 1. 1.]
      [1. 1. 1.]
      [1. 1. 1.]]

     tf.Tensor([0 1 2], shape=(3,), dtype=int64)
     tf.Tensor([5], shape=(1,), dtype=int64)
     tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```




```
[45] x = tf.constant((np.arange(3)))
     y = tf.constant([5], dtype=tf.int64)
     print((x+y).numpy())

     x = tf.constant((np.ones((3, 3))))
     y = tf.constant(np.arange(3), dtype=tf.double)
     print((x+y).numpy())

     x = tf.constant(np.arange(3).reshape(3, 1))
     y = tf.constant(np.arange(3))
     print((x+y).numpy())

     [5 6 7]
     [[1. 2. 3.]
      [1. 2. 3.]
      [1. 2. 3.]]
     [[0 1 2]
      [1 2 3]
      [2 3 4]]
```

[텐서플로 연산]

- tf.add()

```
[46] a = 2
     b = 3
     c = tf.add(a, b)
     print(c.numpy())

     5

[47] x = 2
     y = 3
     add_op = tf.add(x, y)
     mul_op = tf.multiply(x, y)
     pow_op = tf.pow(add_op, mul_op)

     print(pow_op.numpy())

     15625

[48] a = tf.constant(2.)
     b = tf.constant(3.)
     c = tf.constant(5.)

     # Some more operations.
     mean = tf.reduce_mean([a, b, c])
     sum = tf.reduce_sum([a, b, c])

     print("mean = ", mean.numpy())
     print("sum = ", sum.numpy())

     mean =  3.3333333
     sum =  10.0
```

[행렬 곱셈]

행렬 곱(내적)	Numpy	np.dot(a, b)
		a.dot(b)
	Tf	tf.matmul

$$\begin{matrix} A & & B & & A * B \end{matrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1\cdot6+2\cdot5+3\cdot4 & 1\cdot3+2\cdot2+3\cdot1 \\ 4\cdot6+5\cdot5+6\cdot4 & 4\cdot3+5\cdot2+6\cdot1 \end{pmatrix}$$

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

$$C_{ij} = \sum_k A_{ik} B_{kj} = A_{ik} B_{kj}$$

- tf.matmul() 2차원 행렬 곱셈

```
[7]  x = [[2.]]
     m = tf.matmul(x, x)
     print(m)
     print(m.numpy())

     tf.Tensor([[4.]], shape=(1, 1), dtype=float32)
     [[4.]]
```

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} X \begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} = \begin{bmatrix}4 & 4\\10 & 8\end{bmatrix}$$

$$\begin{bmatrix}2 & 0\\1 & 2\end{bmatrix} X \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} = \begin{bmatrix}2 & 4\\7 & 10\end{bmatrix}$$

```
[9]  # Matrix multiplications 1
     matrix1 = tf.constant([[1., 2.], [3., 4.]])
     matrix2 = tf.constant([[2., 0.], [1., 2.]])

     gop = tf.matmul(matrix1, matrix2)
     print(gop.numpy())

     [[ 4.  4.]
      [10.  8.]]
```

```
[10] # Matrix multiplications 2
     gop = tf.matmul(matrix2, matrix1)
     print(gop.numpy())

     [[ 2.  4.]
      [ 7. 10.]]
```

- 행렬의 같은 위치 원소와의 곱

```
[15] # 연산자 오버로딩 지원
     print(a)
     # 텐서로부터 numpy 값 얻기:
     print(a.numpy())
     print(b)
     print(b.numpy())
     print(a * b)

     tf.Tensor(
     [[1 2]
      [3 4]], shape=(2, 2), dtype=int32)
     [[1 2]
      [3 4]]
     tf.Tensor(
     [[2 3]
      [4 5]], shape=(2, 2), dtype=int32)
     [[2 3]
      [4 5]]
     tf.Tensor(
     [[ 2  6]
      [12 20]], shape=(2, 2), dtype=int32)
```

```
[14] # NumPy값 사용
     import numpy as np

     c = np.multiply(a, b)
     print(c)

     [[ 2  6]
      [12 20]]
```

[텐서플로 연산]

- tf.add()
- tf.multiply()
- tf.pow()
- tf.reduce_mean()
- tf.reduce_sum()

```
[46] a = 2
     b = 3
     c = tf.add(a, b)
     print(c.numpy())

     5
```

```
[47] x = 2
     y = 3
     add_op = tf.add(x, y)
     mul_op = tf.multiply(x, y)
     pow_op = tf.pow(add_op, mul_op)

     print(pow_op.numpy())

     15625
```

```
[48] a = tf.constant(2.)
     b = tf.constant(3.)
     c = tf.constant(5.)

     # Some more operations.
     mean = tf.reduce_mean([a, b, c])
     sum = tf.reduce_sum([a, b, c])

     print("mean = ", mean.numpy())
     print("sum = ", sum.numpy())

     mean =  3.3333333
     sum =  10.0
```

[tf.rank]

- 행렬의 차수를 반환

```
[28] my_image = tf.zeros([2, 5, 5, 3])
     my_image.shape

     TensorShape([2, 5, 5, 3])

[19] tf.rank(my_image)

     <tf.Tensor: shape=(), dtype=int32, numpy=4>
```

[shape와 reshape]

- shape

```
[27] rank_three_tensor = tf.ones([3, 4, 5])
     rank_three_tensor.shape

     TensorShape([3, 4, 5])

[29] rank_three_tensor.numpy()

     array([[[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]],

            [[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]],

            [[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]]], dtype=float32)
```

- reshape

```
# 기존 내용을 6x10 행렬로 형태 변경
matrix = tf.reshape(rank_three_tensor, [6, 10])
matrix

<tf.Tensor: shape=(6, 10), dtype=float32, numpy=
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]], dtype=float32)>
```

```
[24] # 기존 내용을 3x20 행렬로 형태 변경
     # -1은 차원 크기를 계산하여 자동으로 결정하라는 의미
     matrix8 = tf.reshape(matrix, [3, -1])
     matrix8

     <tf.Tensor: shape=(3, 20), dtype=float32, numpy=
     array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
             1., 1., 1., 1., 1.],
            [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
             1., 1., 1., 1., 1.],
            [1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
             1., 1., 1., 1., 1.]], dtype=float32)>
```

- reshape에서 -1 사용

```
[25] # 기존 내용을 4x3x5 텐서로 형태 변경
     matrixAlt = tf.reshape(matrix8, [4, 3, -1])
     matrixAlt

     <tf.Tensor: shape=(4, 3, 5), dtype=float32, numpy=
     array([[[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]],

            [[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]],

            [[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]],

            [[1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.],
             [1., 1., 1., 1., 1.]]], dtype=float32)
```

```
[26] # 형태가 변경된 텐서의 원소 개수는 원래 텐서의 원소 개수와 같습니다.
     # 그러므로 다음은 원소 개수를 유지하면서
     # 마지막 차원에 사용 가능한 수가 없기 때문에 에러를 발생합니다.
     yet_another = tf.reshape(matrixAlt, [13, 2, -1])  # 에러!

     ---------------------------------------------------------------------------
     InvalidArgumentError                      Traceback (most recent call last)
     <ipython-input-26-4531640a3825> in <module>()
          2 # 그러므로 다음은 원소 개수를 유지하면서
          3 # 마지막 차원에 사용 가능한 수가 없기 때문에 에러를 발생합니다.
     ----> 4 yet_another = tf.reshape(matrixAlt, [13, 2, -1])  # 에러!

     ------------------------------------------------------------ 4 frames ------------------------------------------------------------
     /usr/local/lib/python3.6/dist-packages/tensorflow/python/eager/execute.py in
     quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
          58     ctx.ensure_initialized()
          59     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     ---> 60                                         inputs, attrs, num_outputs)
          61   except core._NotOkStatusException as e:
          62     if name is not None:

     InvalidArgumentError: Input to reshape is a tensor with 60 values, but the requested
     shape requires a multiple of 26 [Op:Reshape]

     SEARCH STACK OVERFLOW
```

[자료형 변환]

- tf.cast

```
[48] # 정수형 텐서를 실수형으로 변환.
     float_tensor = tf.cast(tf.constant([1, 2, 3]), dtype=tf.float32)
     float_tensor

<tf.Tensor: shape=(3,), dtype=float32, numpy=array([1., 2., 3.], dtype=float32)>

[49] float_tensor.dtype

tf.float32
```

[변수 Variable]

- 텐서플로 그래프에서 tf.Variable의 값을 사용하려면 이를 단순히 tf.Tensor로 취급

assign(), assign_add()	값을 변수에 할당
read_value()	현재 변수값 읽기

```
[39] v = tf.Variable(0.0)
     v

<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=0.0>

[50] w = v + 10
     w

<tf.Tensor: shape=(), dtype=float32, numpy=17.0>

[44] w.numpy()

10.0

[46] v = tf.Variable(2.0)
     v.assign_add(5)
     v

<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=7.0>

[47] v.read_value()

<tf.Tensor: shape=(), dtype=float32, numpy=7.0>
```

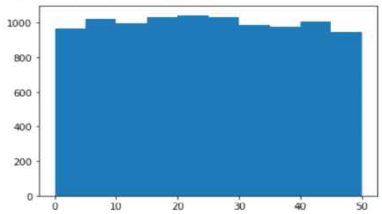
4. 텐서플로 난수 활용

1) 균등분포 난수

- tf.random.uniform([1], 0, 1) → ([배열], 시작, 끝)
- 배열이 증가할수록 사각형, bins → 막대의 개수

```
[6]  1 # 3.7 랜덤한 수 얻기 (균일 분포)
     2 rand = tf.random.uniform([1],0,1)
     3 print(rand)

     tf.Tensor([0.5543064], shape=(1,), dtype=float32)

[8]  1 rand = tf.random.uniform([5, 4],0,1)
     2 print(rand)

     tf.Tensor(
     [[0.43681145 0.84187937 0.9562702  0.7846168 ]
      [0.6079582  0.95665395 0.9038415  0.19482386]
      [0.51012075 0.8609252  0.9433547  0.9636986 ]
      [0.2134043  0.9559026  0.5170028  0.4017253 ]
      [0.0141474  0.15949261 0.23697984 0.7221806 ]], shape=(5, 4), dtype=float32)

[11] 1 rand = tf.random.uniform([1000],0,10)
     2 print(rand[:10])

     tf.Tensor(
     [5.1413307 1.548908  8.911686  9.880335  5.5388713 5.6710424 6.80269
      1.9444573 7.549943  6.573516 ], shape=(10,), dtype=float32)
```

```
[14] 1 import matplotlib.pyplot as plt
     2 rand = tf.random.uniform([10000],0,50)
     3 plt.hist(rand, bins=10)

     (array([ 965., 1020.,  994., 1032., 1043., 1030.,  987.,  976., 1008.,
              945.]),
     array([6.0796738e-04, 4.9998469e+00, 9.9990854e+00, 1.4998324e+01,
            1.9997562e+01, 2.4996801e+01, 2.9996040e+01, 3.4995281e+01,
            3.9994518e+01, 4.4993759e+01, 4.9992996e+01], dtype=float32),
     <a list of 10 Patch objects>)
```

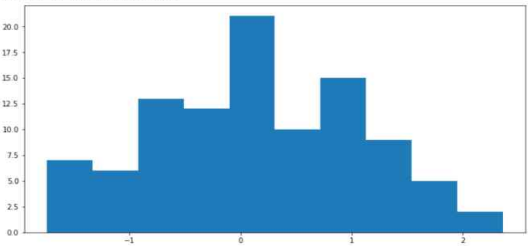


2) 정규분포 난수

- tf.random.normal([4],0,1) → ([크기], 평균, 표준편차)
- 표준편차가 증가할수록 종 모양

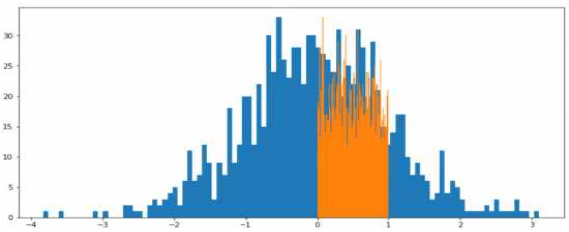
```
[53] 1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
     2 rand = tf.random.normal([4],0,1)
     3 print(rand)

     tf.Tensor([-0.5962639   0.47093895  1.9455601  -0.42773333], shape=(4,), dtype=float32)

[54] 1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
     2 rand = tf.random.normal([2, 4],0,2)
     3 print(rand)

     tf.Tensor(
     [[-2.145662    0.64699423  2.0760484  -1.4640687 ]
      [ 1.3588632  -0.9740333   1.4347676  -1.3747462 ]], shape=(2, 4), dtype=float32)
```

```
[52] 1 import matplotlib.pyplot as plt
     2 rand = tf.random.normal([100], 0, 1)
     3 plt.hist(rand, bins=10)

     (array([ 7.,  6., 13., 12., 21., 10., 15.,  9.,  5.,  2.]),
      array([-1.7424849 , -1.332153  , -0.921821  , -0.511489  , -0.10115702,
              0.30917495,  0.7195069 ,  1.129839  ,  1.5401709 ,  1.9505029 ,
              2.3608348 ], dtype=float32),
     <a list of 10 Patch objects>)
```



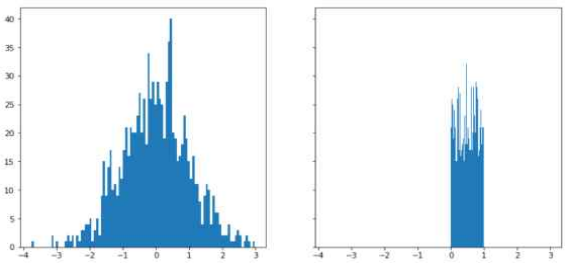
3) 균등분포와 정규분포의 비교

```
[57] 1 import matplotlib.pyplot as plt
     2 rand1 = tf.random.normal([1000],0, 1)
     3 rand2 = tf.random.uniform([2000], 0, 1)
     4 plt.hist(rand1, bins=100)
     5 plt.hist(rand2, bins=100)
```



4) 균등분포와 정규분포를 부분으로 그리기

```
import matplotlib.pyplot as plt
rand1 = tf.random.normal([1000],0, 1)
rand2 = tf.random.uniform([2000], 0, 1)

plt.rcParams["figure.figsize"] = (12,6)
fig, axes = plt.subplots(1, 2, sharex=True, sharey=True)
axes[0].hist(rand1, bins=100)
axes[1].hist(rand2, bins=100)
```



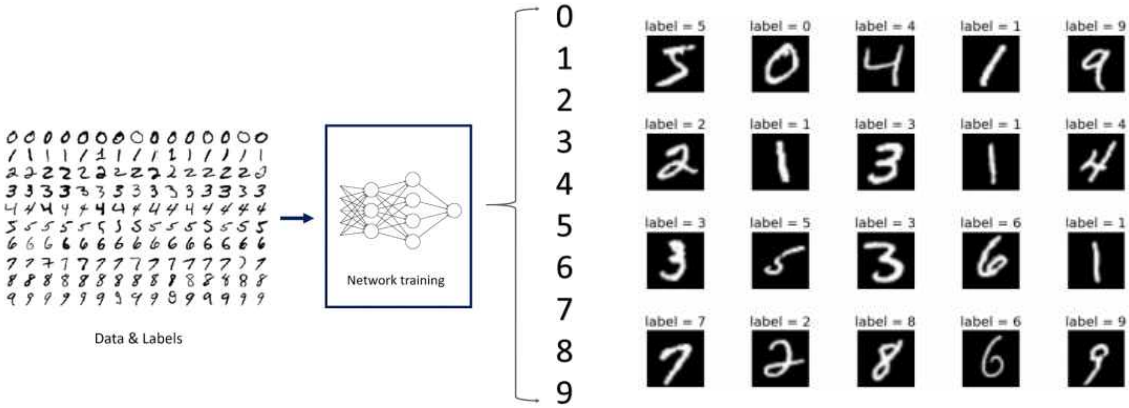
5) shuffle

- tf.random.shuffle(a)

```
[29] import numpy as np
     a = np.arange(10)
     print(a)
     tf.random.shuffle(a)

     [0 1 2 3 4 5 6 7 8 9]
     <tf.Tensor: shape=(10,), dtype=int64, numpy=array([7, 9, 1, 4, 3, 5, 8, 6, 2, 0])>

[26] import numpy as np
     a = np.arange(20).reshape(4, 5)
     a

     array([[ 0,  1,  2,  3,  4],
            [ 5,  6,  7,  8,  9],
            [10, 11, 12, 13, 14],
            [15, 16, 17, 18, 19]])

[27] tf.random.shuffle(a)

     <tf.Tensor: shape=(4, 5), dtype=int64, numpy=
     array([[ 0,  1,  2,  3,  4],
            [15, 16, 17, 18, 19],
            [10, 11, 12, 13, 14],
            [ 5,  6,  7,  8,  9]])>
```

5. MNIST 손글씨 프로그래밍

1) MNIST(Modified National Institute of Standards and Technology) 데이터셋



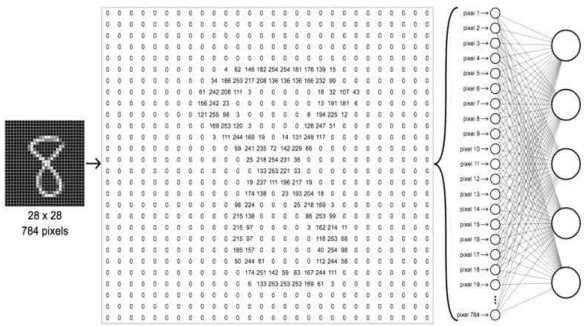
- 손으로 쓴 자릿수에 대한 데이터 집합
- 필기 숫자 이미지와 정답인 레이블의 쌍으로 구성
- 숫자의 범위는 0에서 9까지, 총 10개의 패턴을 의미
- 필기 숫자 이미지 : 크기가 28 x 28 픽셀인 회색조 이미지
- 레이블 : 필기 숫자 이미지가 나타내는 실제 숫자, 0에서 9

ex) tensorflow.keras.datasets.mnist

tensorflow.examples.tutorials.mnist

2) 손글씨 하나의 구조

- 784픽셀 (28 X 28)
- 내부 값은 0~255, 이 값을 0~1로 수정해서 사용



3) 딥러닝 과정

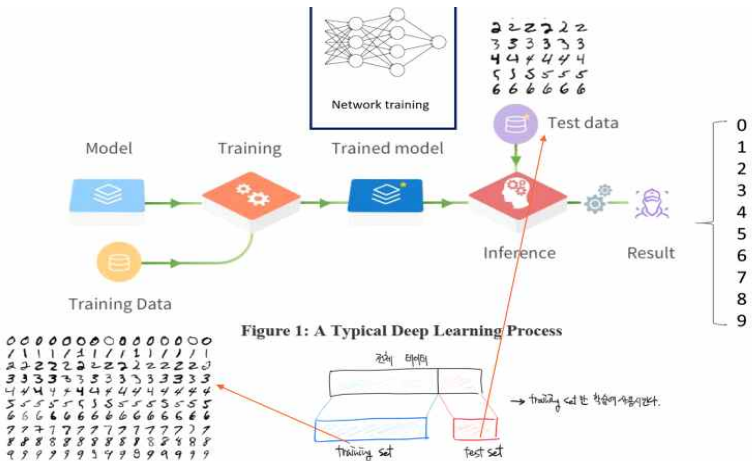


Figure 1: A Typical Deep Learning Process

4) 케라스 딥러닝 구현

① define : 딥러닝 모델 생성

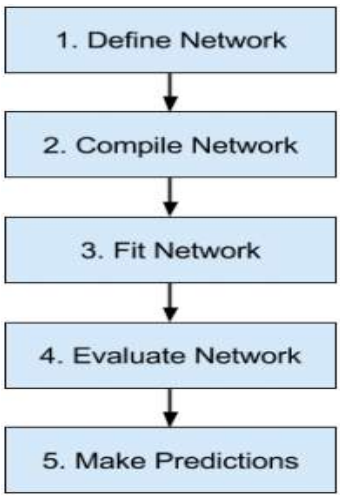
② compile : 주요 훈련방법을 설정

- 최적화 방법(optimizers)
- 손실 함수(losses)
- 훈련 모니터링 지표(metrics)

③ fit : 훈련

④ evaluate : 테스트 데이터를 평가

⑤ predict : 정답을 예측



5) MNIST 손글씨 데이터 로드 코드

- 훈련 데이터 손글씨와 정답 : x_train, y_train → 6만개
- 테스트 데이터 손글씨와 정답 : x_test, y_test → 1만개

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

6) MNIST 손글씨 데이터 구조 확인

```
[62]  x_train.shape
      (60000, 28, 28)

[68]  y_train.shape
      (60000,)

[70]  x_train[0]
      array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
              18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
               0,   0],
             [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
             253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
               0,   0],
             [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
             253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
               0,   0],
      ...

[71]  y_train[0]
      5
```

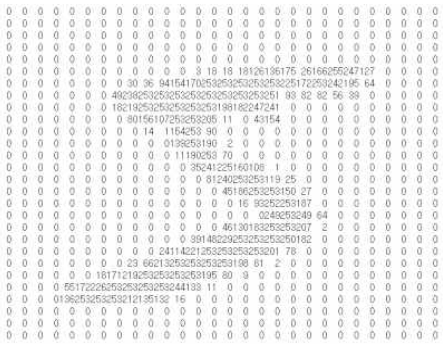
```
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# MNIST 형태를 알아 봅시다. 데이터 수, 행렬 형태 등
print(x_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)

# MNIST 훈련 데이터의 내부 첫 내용도 알아보자.
print(x_train[0])
print(y_train[0])

# MNIST 테스트 데이터의 내부 첫 내용도 알아보자.
print(x_test[0])
print(y_test[0])
```

이차원 28 x 28

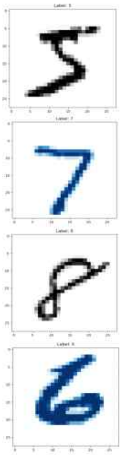
```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)

[[...]]
5

[[...]]
7
```

7) 행렬 내용 직접 출력

```
import sys

for x in x_train[0]:
    for i in x:
        sys.stdout.write('%3d' % i)
    sys.stdout.write('\n')
```



8) 첫 손글씨와 마지막 손글씨 그려 보기

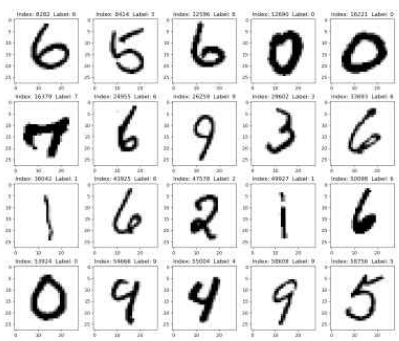


```
# MNIST 데이터(훈련, 테스트)의 내부 첫 내용을 그려보자.
import matplotlib.pyplot as plt

tmp = "Label: " + str(y_train[0])
plt.title(tmp)
plt.imshow(x_train[0], cmap="Greys")
plt.show()

tmp = "Label: " + str(y_test[0])
plt.title(tmp)
plt.imshow(x_test[0], cmap='Blues')
plt.show()

# MNIST 데이터(훈련, 테스트)의 내부 마지막 내용을
그려보자.
idx = len(x_train) - 1
tmp = "Label: " + str(y_train[idx])
plt.title(tmp)
plt.imshow(x_train[idx], cmap="Greys")
plt.show()

idx = len(x_test) - 1
tmp = "Label: " + str(y_test[idx])
plt.title(tmp)
plt.imshow(x_test[idx], cmap='Blues')
plt.show()
```

9) 훈련용 데이터 60000개 중에서 임의 손글씨 출력

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

from random import sample
nrows, ncols = 4, 5 #출력 가로 세로 수

# 출력할 첨자 선정
idx = sorted(sample(range(len(x_train)), nrows * ncols))
print(idx)
```

```
[577, 4733, 6096, 7075, 15445, 15592, 22448, 22721, 23361, … ]
```



```
#랜덤하게 20개의 훈련용 자료를 그려 보자.
from random import sample

nrows, ncols = 4, 5 #출력 가로 세로 수
# 출력할 첨자 선정
idx = sorted(sample(range(len(x_train)), nrows * ncols))
#print(idx)

count = 0
plt.figure(figsize=(12, 10))

for n in idx:
    count += 1
    plt.subplot(nrows, ncols, count)
    tmp = "Index: " + str(n) + "  Label: " +
str(y_train[n])
    plt.title(tmp)
    plt.imshow(x_train[n], cmap='Greys')

plt.tight_layout()
plt.show()
```

10) 딥러닝 과정

- 모델 구성(개발) : 블랙박스(black box)
- 모델 훈련(train)

  : 모델이 문제를 해결하도록 훈련, 어린아이가 부모에게 훈련받는 것에 비유

  : 학습 방법 및 모니터링 지표 설정

  ex) 경사하강법(내리막 경사 따라가기)

  손실함수(Loss Function)

  모니터링 지표 metrics

- 예측 : inference, prediction

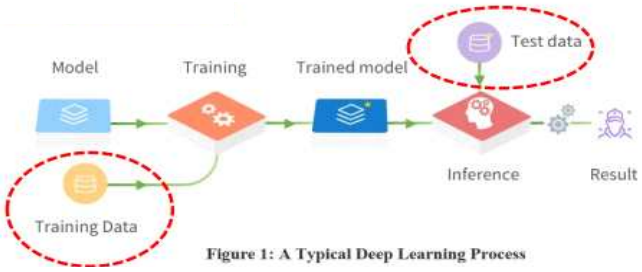


Figure 1: A Typical Deep Learning Process

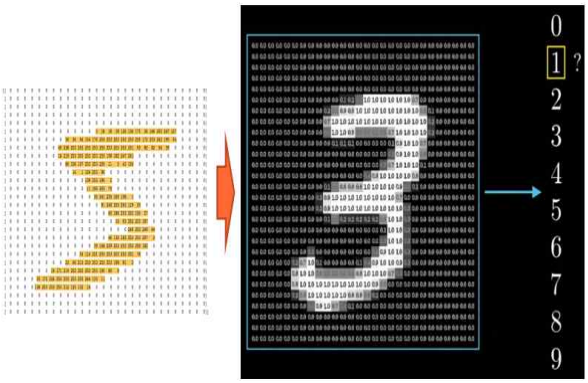
11) 딥러닝 구현 순서

0. 필요 모듈 임포트

1. 훈련과 정답 데이터 지정

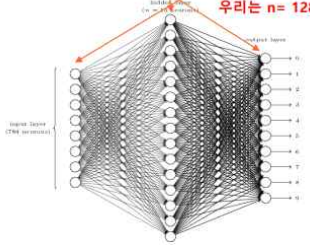
   - 데이터 전처리(옵션)

```
import tensoflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0
```



2. 모델 구성

```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



3. 학습에 필요한 최적화 방법과 손실함수 등 설정

옵티마이저 : 입력된 데이터와 손실함수를 기반으로 모델(w와 b)을 업데이트하는 메커니즘

손실함수 : 훈련 데이터에서 신경망의 성능을 측정하는 방법

: 모델이 옳은 방향으로 학습될 수 있도록 도와주는 기준값

훈련과 테스트 과정을 모니터링할 지표 : 여기에서는 정확도만 고려

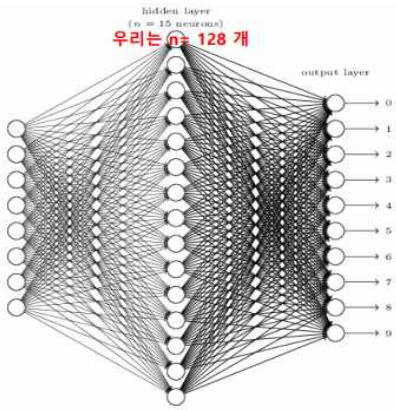
- 구성된 모델 요약(옵션) : compile 전에도 summary() 가능

```
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
              # metrics=['accuracy', 'mse'])

# 모델 요약 표시
model.summary()
```

- model.summary() : 각 층의 구조와 패라미터 수 표시

: 가중치(weights)와 편향(biases)

총 패라미터 수 : 모델이 구해야 할 수의 개수 (101,770)




```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

4. 생성된 모델로 훈련 데이터 학습

- model.fit() : 훈련 횟수 epochs에 지정

```
# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0248 - accuracy: 0.9913
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0234 - accuracy: 0.9920
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0236 - accuracy: 0.9918
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0232 - accuracy: 0.9920
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0229 - accuracy: 0.9922
<tensorflow.python.keras.callbacks.History at 0x7f4a78e6cd30>
```

5. 테스트 데이터로 성능 평가

- 테스트 데이터 또는 다른 데이터로 결과 예측(옵션)
- model.evaluate() : 손실 값과 예측 정확도 반환 (loss, accuracy)

```
# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0868 - accuracy: 0.9805
[0.08675415068864822, 0.9804999828338623]
```

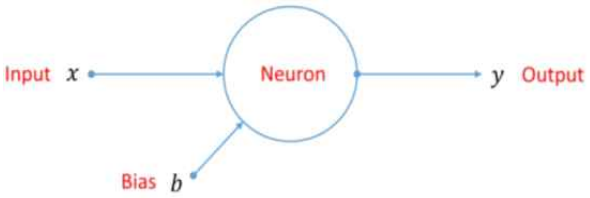
98%의 정확도로
손글씨를 맞춤

6. 인공신경망의 이해

1) 인공 신경세포(Artificial Neuron)

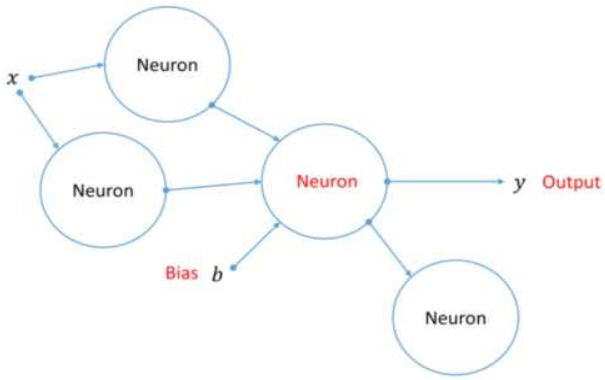
뉴런 - 입력

- 편향(bias) : 편향을 조정해 출력을 맞춤



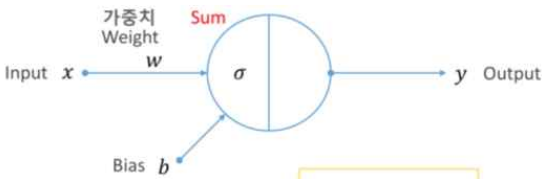
Input x	Output y
Size of house	Price
Time spent for studying	Score in exam

신경망(network) : 뉴런의 연결



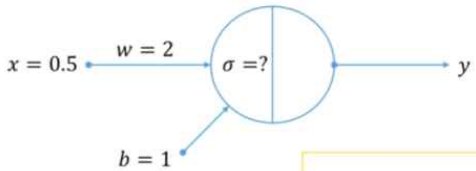
2) 뉴런 연산

[뉴런 식]



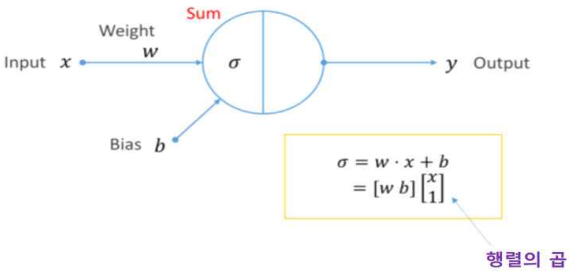
$$\sigma = w \cdot x + b$$

[가중치와 편향]



$$\begin{aligned}\sigma &= w \cdot x + b \\ &= 2 \cdot 0.5 + 1 \\ &= 2\end{aligned}$$

3) 행렬 곱 연산



4) 활성화 함수

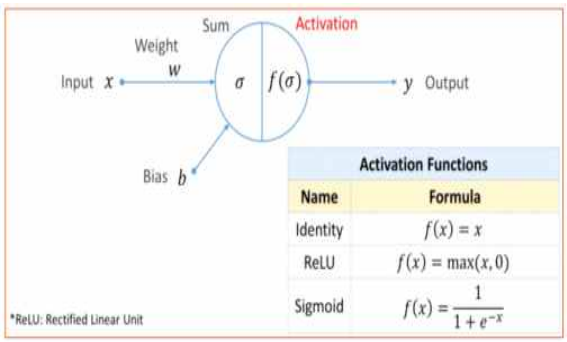
- ReLU (Rectified(정류된) Linear Unit(선형 함수, y=x를 의미))

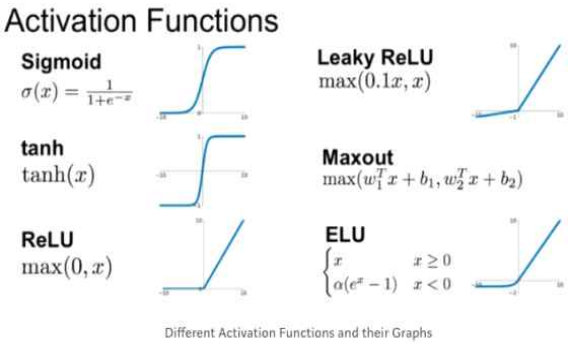
: 선형 함수를 정류하여 0 이하는 모두 0으로 한 함수 (양수만 사용)

: 2010년 이후 층이 깊어질수록(deep) 많이 활용

: 양수를 그대로 반환하므로 값의 왜곡이 적어지는 효과

: max(x, 0)



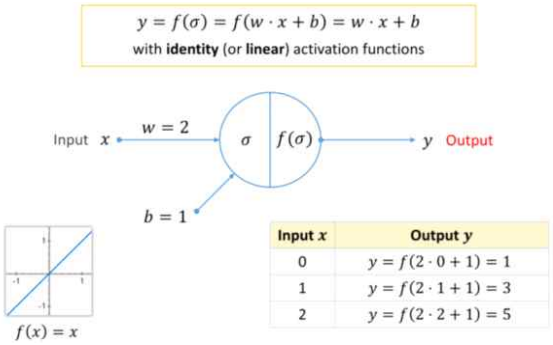


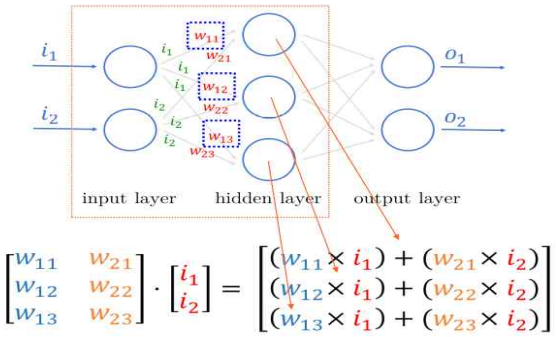
- Sigmoid

: s자 형태의 곡선이라는 의미, 예전에 많이 사용

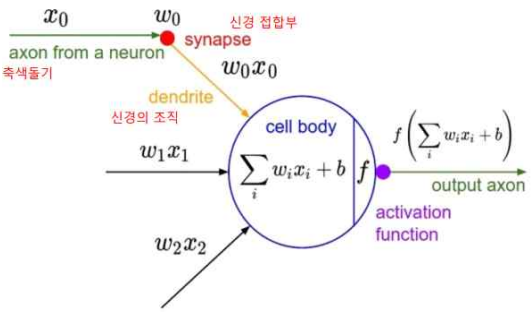
5) 입출력의 예

출력 함수로 동일(identity) 함수(또는 리니어 함수)를 적용



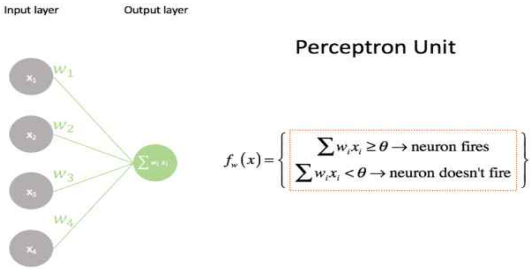


6) 일반화된 인공신경망



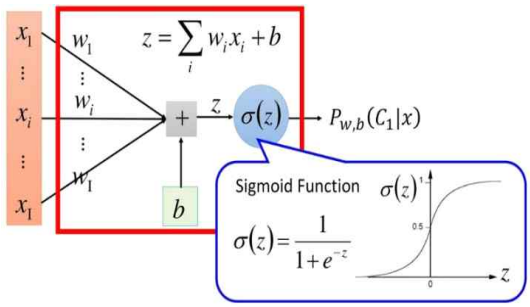
7) 활성화 함수와 편향

결과값이 임계값 역할 : 결과가 임계값 이상이면 활성화, 미만이면 비활성화



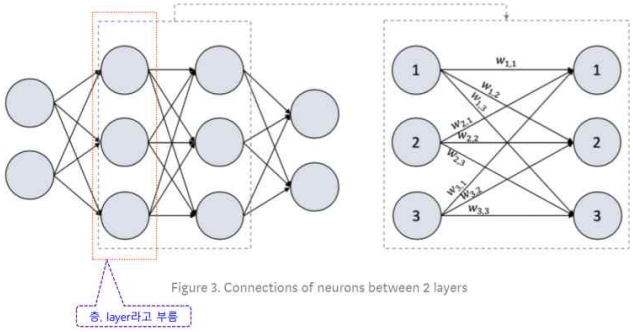
8) 인공신경망의 시그모이드 함수

활성화 함수의 예 : 시그모이드 함수 → 출력 값이 (0~1)



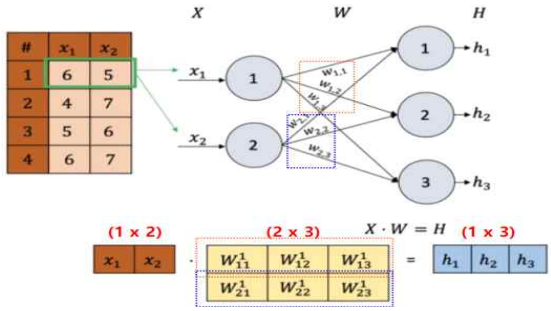
9) 가중치

3X3의 가중치 실수

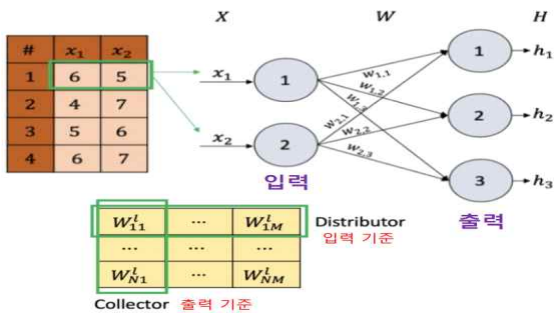


10) 인공신경망 행렬 연산

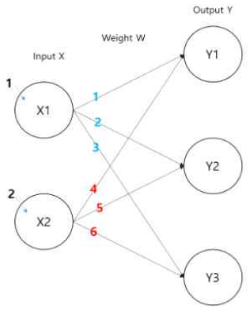
[신경망 행렬 계산]



[뉴런 계산]



[계산 사례]



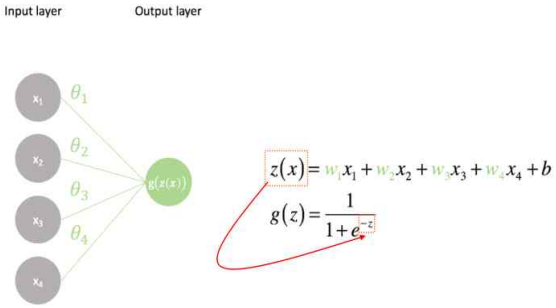
X * W = Y

1 x 2 * 2 x 3 = 1 x 3

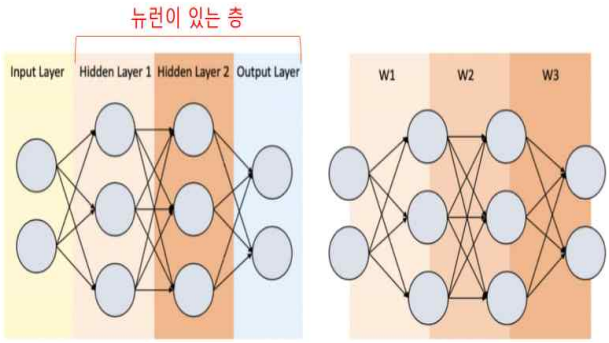
(1 2) $\begin{pmatrix}1&2&3\\4&5&6\end{pmatrix}$ (9 12 15)

11) 하나의 출력 뉴런 연산

활성화 함수로 시그모이드 함수 적용



12) 층과 가중치



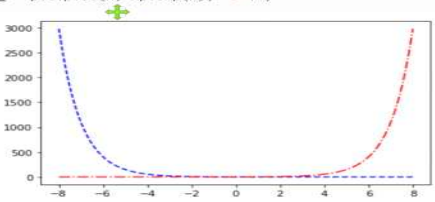
13) 활성화 함수 그리기

[자연수와 자연수의 지수 승]

- e : 자연수, 오일러 수 (2.71828)
- $y = e^{-x}$
- $y = e^{x}$

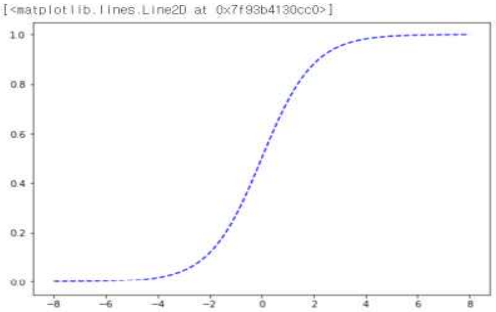
```
[76] import numpy as np
     np.e

     2.718281828459045
```

```
[77] import numpy as np
     import matplotlib.pylab as plt

     plt.figure(figsize=(6, 4))
     x = np.linspace(-8, 8, 100)
     plt.plot(x, np.exp(-x), 'b--')
     _ = plt.plot(x, np.exp(x), 'r-.')
```



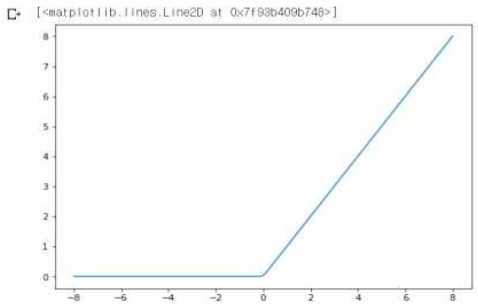
[시그모이드 함수]

```
[44] 1 import numpy as np
     2 import matplotlib.pylab as plt
     3
     4 def sigm_func(x): # sigmoid 함수
     5     return 1 / (1 + np.exp(-x))
     6
     7 # 시그모이드 함수 그리기
     8 plt.figure(figsize=(8, 6))
     9 x = np.linspace(-8, 8, 100)
    10 plt.plot(x, sigm_func(x), 'b--')

     [<matplotlib.lines.Line2D at 0x7f93b4130cc0>]
```



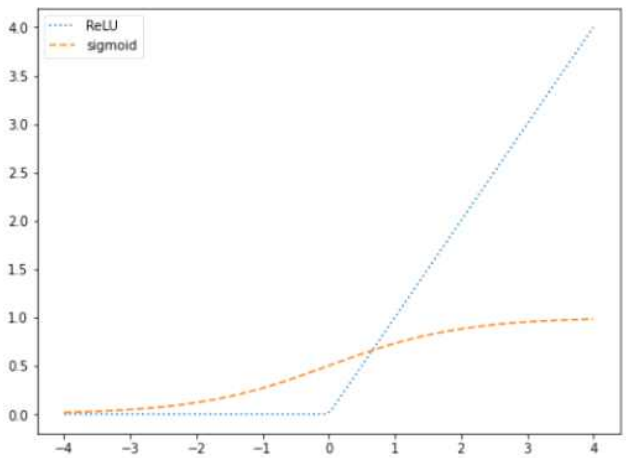
[ReLU 함수]

```
[45] 1 import numpy as np
     2 import matplotlib.pylab as plt
     3
     4 def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수
     5     return np.maximum(0, x)
     6     #return (x>0)*x # same
     7
     8 # ReLU 함수 그리기
     9 plt.figure(figsize=(8, 6))
    10 x = np.linspace(-8, 8, 100)
    11 plt.plot(x, relu_func(x))

     [<matplotlib.lines.Line2D at 0x7f93b409b748>]
```



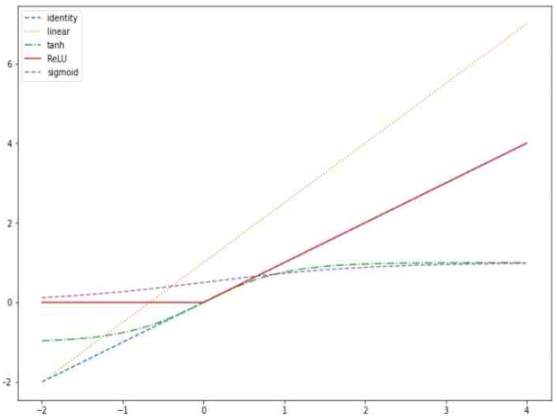
[시그모이드 ReLU 함께 그리기]

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

# ReLU(Rectified Linear Unit
# (정류된 선형 유닛) 함수
def relu_func(x):
    return np.maximum(0, x)
    #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수
    return 1 / (1 + np.exp(-x))

# 그래프 그리기
plt.figure(figsize=(8, 6))
x = np.linspace(-4, 4, 100)
y = np.linspace(-0.2, 2, 100)

plt.plot(x, relu_func(x), linestyle=':', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```



[다양한 활성화 함수와 결과]

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

def identity_func(x): # 항등함수
    return x

def linear_func(x): # 1차함수
    return 1.5 * x + 1 # a기울기(1.5), y절편b(1) 조정가능

def tanh_func(x): # TanH 함수
    return np.tanh(x)

def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수
    return np.maximum(0, x)
    #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수
    return 1 / (1 + np.exp(-x))

# 그래프 그리기
plt.figure(figsize=(12, 8))
x = np.linspace(-2, 4, 100)

plt.plot(x, identity_func(x), linestyle='--', label="identity")
plt.plot(x, linear_func(x), linestyle=':', label="linear")
plt.plot(x, tanh_func(x), linestyle='-.', label="tanh")
plt.plot(x, relu_func(x), linestyle='-', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```



14) 논리 게이트 AND OR XOR 신경망 구현

[AND 게이트 구현]

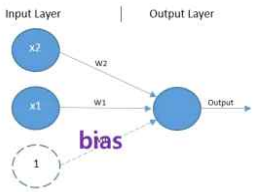
뉴런 구조 : 입력 2개, 편향, 출력 1

구할 값 : 가중치 2개와 편향 1개

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

```
# tf.keras 를 이용한 AND 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2,)),
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```



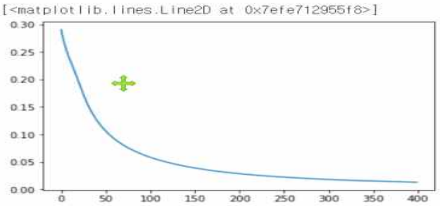

```
history = model.fit(x, y, epochs=400, batch_size=1)
```

```
4/4 [==============================] - 0s 1ms/step - loss: 0.0145
Epoch 372/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 373/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 374/400
```

[손실 값 그래프와 결과 예측]

```
[54] # 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시
     import matplotlib.pyplot as plt
     plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7efe712955f8>]
```



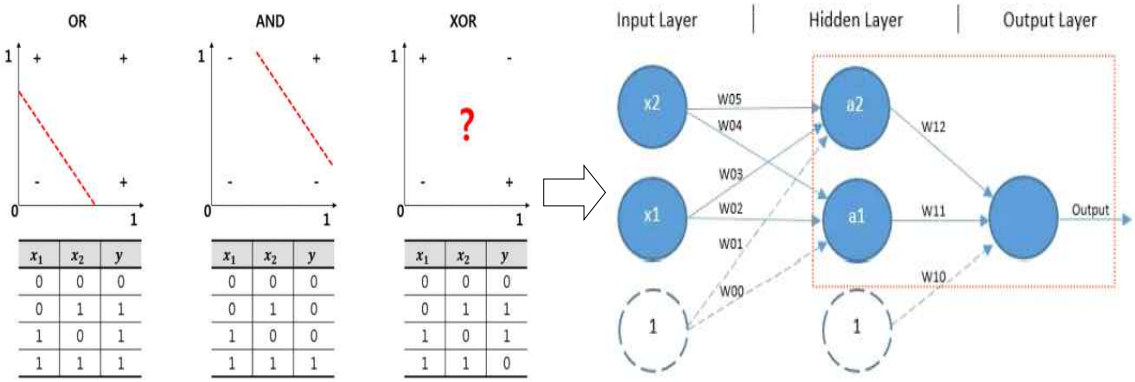

```
[59] model.predict(x)
```

```
array([[0.8535386 ],
       [0.12342612],
       [0.12364786],
       [0.00339741]], dtype=float32)
```

[가중치와 편향 값 알아보기]

```
[60] for weight in model.weights:
         print(weight)

<tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>
<tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>

[61] model.weights[0]

<tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>

[62] model.weights[1]

<tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>
```

[XOR 게이트]



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

[Sequential 모델]

Dense 층 : 가장 기본적인 층

- 인자 units, activation : 뉴런 수와 활성화 함수
- 인자 input_shape : 첫 번째 층에서만 정의, 입력의 차원을 명시

  ex) (2,) → 2개의 입력을 받는 1차원

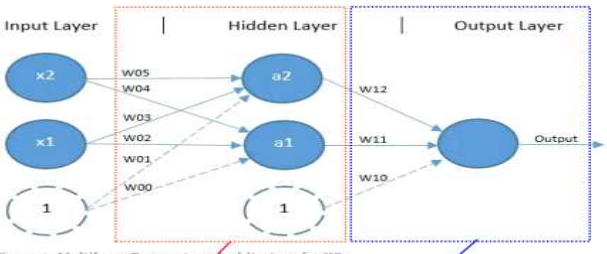


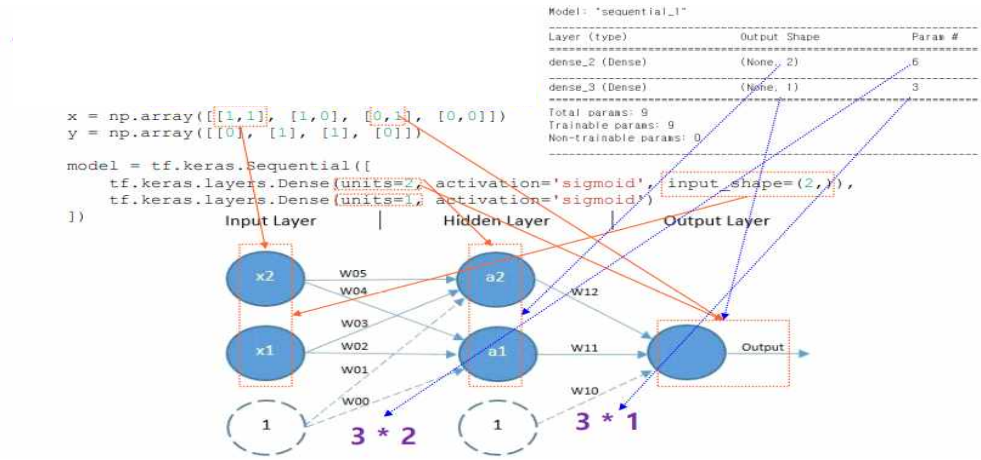
Figure 4: Multilayer Pereceptron Architecture for XOr

```
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

[Sequential 모델과 딥러닝 구조]

입력, 은닉, 출력 층

- 패러미터 수 : (입력측 뉴런 수 + 1) * (출력측 뉴런 수)



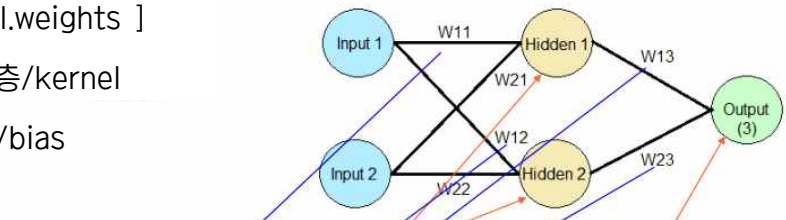
[XOR 게이트 구현 소스]

```
# 3.27 tf.keras 를 이용한 XOR 네트워크 계산
import tensorflow as tf
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

# 3.28 tf.keras 를 이용한 XOR 네트워크 학습
history = model.fit(x, y, epochs=2000, batch_size=1)

# 3.29 tf.keras 를 이용한 XOR 네트워크 평가
print(model.predict(x))

# 3.30 XOR 네트워크의 가중치와 편향 확인
for weight in model.weights:
    print(weight)
```

```
Epoch 1999/2000
4/4 [==============================] - 0s 2ms/step - loss: 0.0017
Epoch 2000/2000
4/4 [==============================] - 0s 1ms/step - loss: 0.0017
[[0.04060324]
 [0.9609237 ]
 [0.96031225]
 [0.04571233]]
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

[가중치와 model.weights]

- 가중치 결과 : 층/kernel
- 편향 결과 : 층/bias



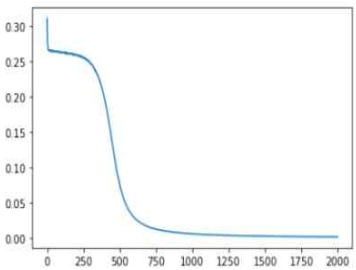

```
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

[XOR 모델의 학습 과정 시각화]

손실(loss) 또는 오류 값의 변화

- 가로는 에폭의 수 : 학습 횟수가 증가하면서 계속 손실은 작아짐

```
# 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
```



7. 회귀와 분류

1) 회귀(regresson)와 분류(classification)

	회귀 모델	분류 모델
예측값	연속적인 값	불연속적인 값
예시	캘리포니아의 주택 가격이 얼마인가요?	주어진 이메일 메시지가 스팸인가요, 스팸이 아닌가요?
	사용자가 이 광고를 클릭할 확률이 얼마인가요?	이 이미지가 강아지, 고양이 또는 햄스터의 이미지인가요?

[회귀의 어원]

회귀분석 (regression analysis)	관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법
	회귀분석은 시간에 따라 변화하는 데이터나 어떤 영향, 가설적 실험, 인과 관계의 모델링 등의 통계적 예측에 이용
회귀	영어 regress 리그레스[*]의 원래 의미 → 옛날 상태로 돌아가는 것을 의미

2) 선형 회귀와 로지스틱 회귀

[단순 선형 회귀분석(Simple Linear Regression Analysis)]

- 입력 : 특징이 하나 | 출력 : 하나의 값
- 키로 몸무게 추정

$$H(x) = Wx + b$$

[다중 선형 회귀분석(Multiple Linear Regression Analysis)]

- 입력 : 특징이 여러 개 | 출력 : 하나의 값
- 역세권, 아파트 평수, 주소로 아파트값을 추정

$$y = W_1x_1 + W_2x_2 + \ldots W_nx_n + b$$

[로지스틱 회귀(Logistic Regression)]

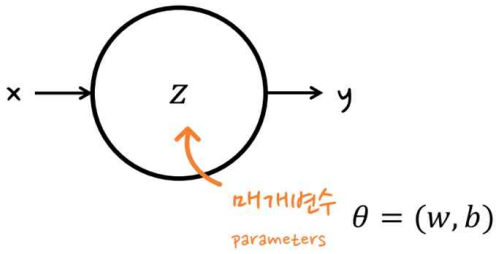
- 이진 분류(Binary Classification)
- 입력 : 하나 또는 여러 개 | 출력 : 0 아니면 1
- 타이타닉의 승객 정보로 죽음을 추정

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격

[인공지능이란? W와 b 구하기]

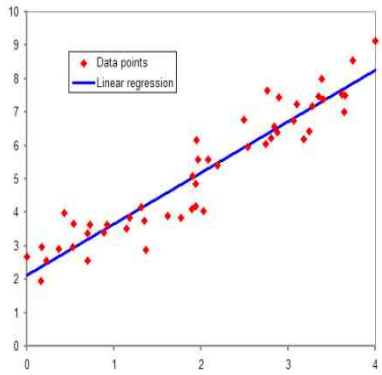
다음 식에서 가중치 W와 편향 b를 구하기 → W와 b를 매개변수 함

$$H(x) = Wx + b$$



[선형 회귀]

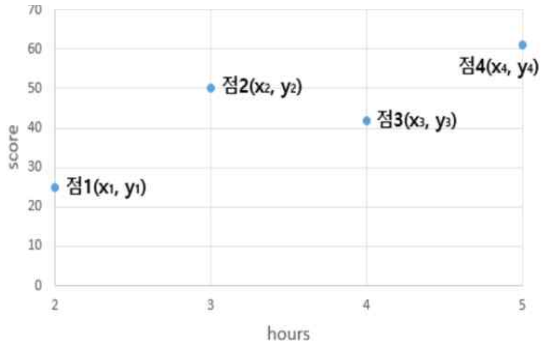
Linear regression
데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법
Y = aX + b
기울기 a와 절편인 b를 구하는 것
국어와 수학 성적 키와 몸무게 치킨과 맥주의 판매량 기저귀와 맥주의 판매량



- 딥러닝 분야에서 선형 회귀는 Y = wX + b → 가중치 w와 편향인 b를 구하는 것

[선형 회귀 문제 사례]

- 공부 시간이 x라면, 점수는 y
- 알려준 데이터로부터 x와 y의 관계를 유추
  : 학생이 6시간을 공부하였을 때의 성적
  : 그리고 7시간, 8시간을 공부하였을 때의 성적을 예측



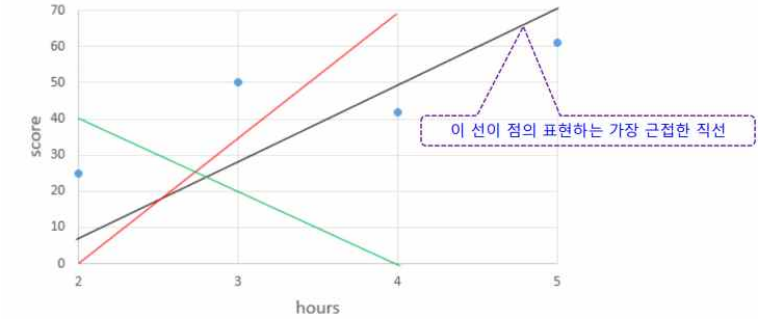
hours(x)	score(y)
2	25
3	50
4	42
5	61

[가설]

머신러닝 : y와 x간의 관계를 유추한 식을 가설(Hypothesis) (H(x)에서 H는 Hypothesis를 의미)

$$H(x) = Wx + b$$

W:기울기, 가중치
b: 절편, 편향



- 선형 회귀에서 해야할 일은 결국 적절한 W와 b를 찾아내는 일
- 딥러닝 알고리즘이 하는 것이 바로 적절한 W와 b를 찾아내는 일

[손실 함수(Loss function)]

- 목적 함수(Objective function), 비용 함수(Cost function)라고도 부름
- 머신러닝은 W와 b를 찾기 위해서 손실함수를 정의
- 실제 값과 가설로부터 얻은 예측 값의 오차를 계산하는 식
- 예측 값의 오차를 줄이는 일에 최적화된 식
- 손실함수 값을 최소화하는 최적의 W와 b를 찾아내려고 노력
- 평균 제곱 오차(Mean Squared Error, MSE) 등을 사용

$$\frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

실제 값 / 예측 값

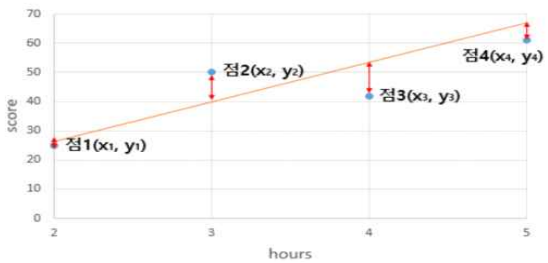
[손실함수 : MSE]

W와 b의 값을 찾아내기 위해 오차의 크기를 측정할 방법이 필요

→ W:13 b:1로 예측한다면 y=13x+1 직선이 예측한 함수로 예측 값을 추정

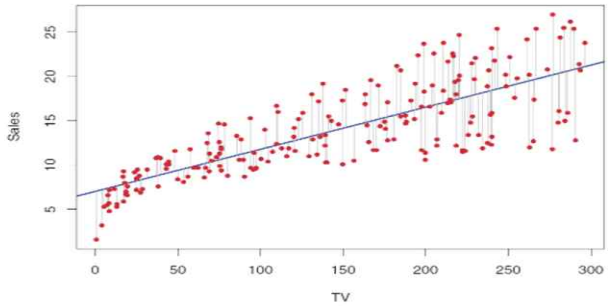
hours(x)	2	3	4	5
실제값	25	50	42	61
예측값	27	40	53	66
오차	-2	10	-7	-5

$$\frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$



[손실 함수 MSE 이해]

- 오차는 실제 데이터(빨간 점)와 예측 선(파란 선)의 차이의 제곱의 합



[손실 함수를 W와 b의 함수로]

$$cost(W,b) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

- 평균 제곱 오차를 W와 b에 의한 비용 함수(Cost function)로 재정의
- 모든 점들과의 오차↑ → 평균 제곱 오차↑, 오차↓ → 평균 제곱 오차↓

평균 제곱 오차 W,b → minimize cost(W,b)

: cost(W, b)를 최소가 되게 만드는 W와 b를 구하면 결과적으로 y와 x의 관계를 가장 잘 나타내는 직선을 그릴 수 있게 됨

3) 최적화 과정

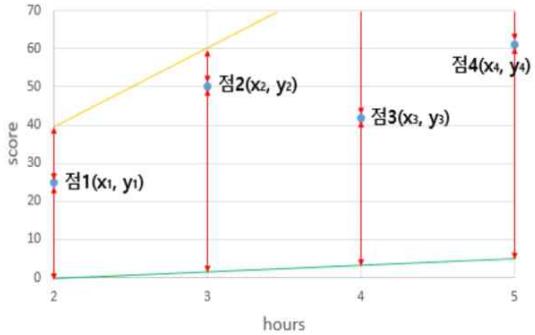
[옵티마이저(Optimizer) : 최적화 과정]

- 머신러닝에서 학습(training)

  : 최적화 알고리즘(Optimizer algorithms), 적절한 W와 b를 찾아내는 과정

- Gradient Descent(경사 하강법)

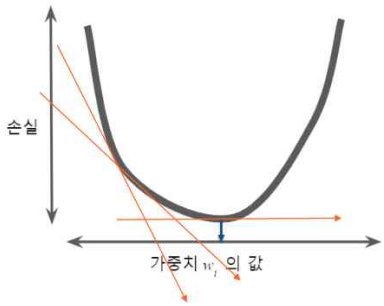
  : 비용 함수(Cost Function)의 값을 최소로 하는 W와 b를 찾는 방법



[손실과 가중치]

- 손실과 가중치 wi을 대응한 그림, 항상 볼록 함수 모양을 함
- 볼록 문제에는 기울기가 정확하게 0인 지점인 최소값이 하나만 존재

  → 이 최소값에서 손실 함수가 수렴, 결국 기울기를 구해야 함

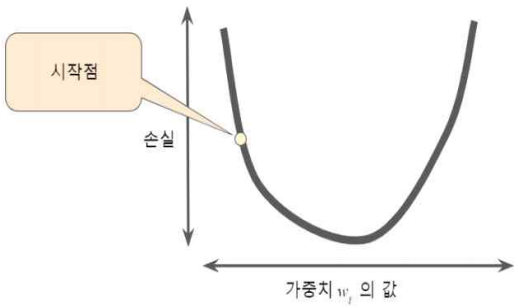


[경사하강법]

- 시작 값(시작점)을 선택

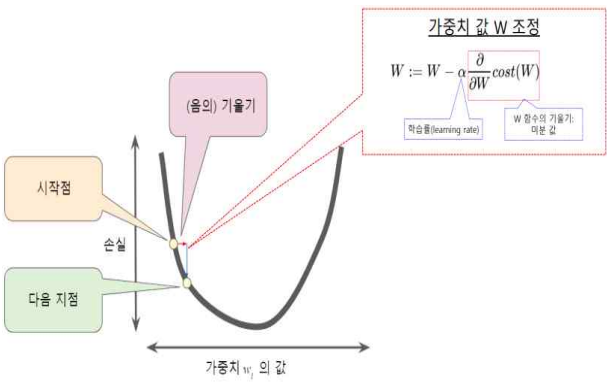
  (시작점은 별로 중요X → 많은 알고리즘에서는 0으로 설정하거나 임의의 값을 선택)

- 시작점에서 손실 곡선의 기울기를 계산 (단일 가중치에 대한 손실의 기울기=미분 값)



[가중치의 조정]

- 기울기가 0인 지점을 찾기 위해 기울기의 반대 방향으로 이동
- 현재의 기울기가 음수이면 다음 가중치 값은 현재의 값보다 크게 조정



[학습률]

다음 가중치 값 결정 방법

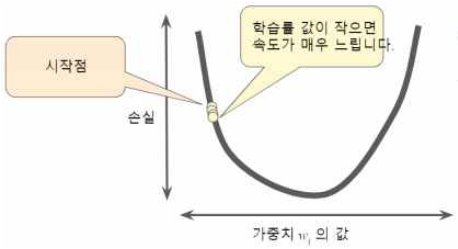
- 기울기에 학습률(또는 보폭이라 불리는 스칼라)를 곱하여 다음 지점을 결정

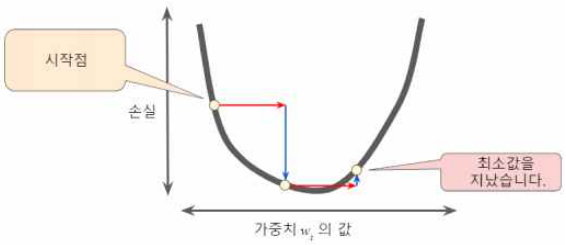
  ex) 기울기가 -2.5이고 학습률이 0.01이면 w = w − (-2.5 x 0.01) = w + 0.025

(경사하강법 알고리즘은 이전 지점으로부터 0.025 떨어진 지점을 다음 지점으로 결정)

학습률의 값

- 너무 작게 설정 → 학습 시간이 매우 오래 걸림
- 너무 크게 설정 → 다음 지점이 곡선의 최저점을 무질서하게 이탈할 우려가 있음

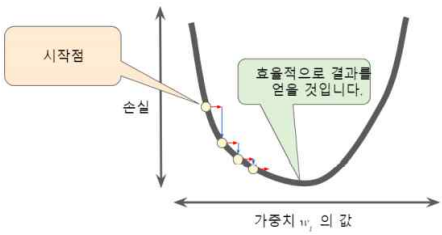




적절한 학습률 설정

- 손실함수의 기울기가 작다면 더 큰 학습률 시도 가능

  → 작은 기울기를 보완하고 더 큰 보폭을 만들어 냄



[초매개변수와 학습률]

초매개변수(hyperparameter)

: 딥러닝에서 우리가 설정하는 값

: 모델 학습을 연속적으로 실행하는 중에 개발자 본인에 의해 조작되는 '손잡이'

ex) 학습률은 초매개변수 중 하나, 매개변수와 대비되는 개념

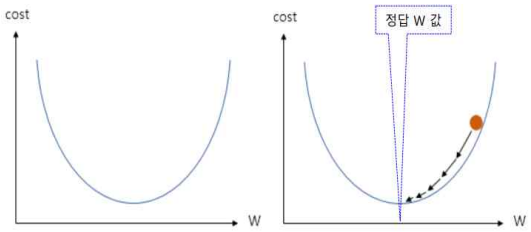
학습률 α

- W의 값을 변경할 때 얼마나 큰 폭으로 이동할지를 결정
- 학습률 α의 값을 무작정↑ → W의 값이 발산하는 상황
- 학습률 α가 지나치게↓ → 학습 속도 저하, 적당한 α의 값을 찾아내는 것도 중요
- 0.001에서 0.1 정도 사용

[cost가 가장 최소값을 가지게 하는 W를 찾는 일]

y = Wx라는 가설 H(x)

- 비용 함수의 값 cost(W) : 설명의 편의를 위해 편향 b 없이 단순히 가중치 W만을 사용



[비용 함수와 최적의 W 구하기]

비용함수(Cost function)

$$cost(W) = \frac{1}{n}\sum_{i}^{n}[y_i - H(x_i)]^2$$

Cost를 최소화하는 W를 구하기 위한 식

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

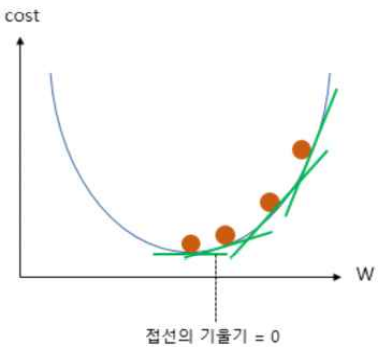
학습률(learning rate)

W 함수의 기울기: 미분 값

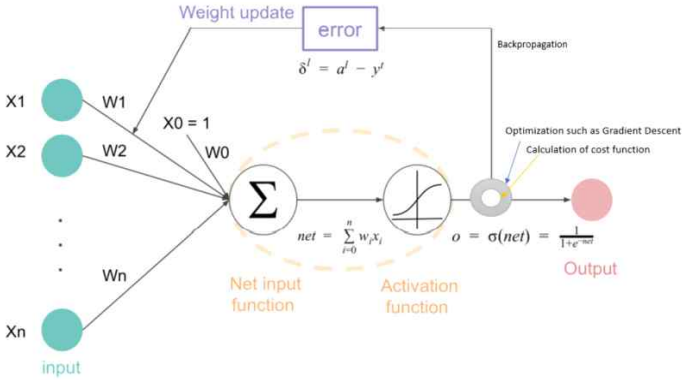
- 해당 식은 접선의 기울기가 0이 될 때까지 반복
- 현재 W에서의 접선의 기울기와 α와 곱한 값을 현재 W에서 빼서 새로운 W의 값으로 다음 손실을 계산
- 학습률(알파): 기울기가 최소인 다음 w로 가기 위한 비율

[경사 하강법 정리]

- 내리막 경사 따라가기
- 접선의 기울기
  : 맨 아래의 볼록한 부분에서는 결국 접선의 기울기가 0
- cost가 최소화가 되는 지점은 접선의 기울기가 0이 되는 지점
  : 또한 미분값이 0이 되는 지점
- 경사 하강법의 아이디어
  : 비용 함수(Cost function)를 미분하여 현재 W에서의 접선의 기울기를 구하고
  : 접선의 기울기가 낮은 방향으로 W의 값을 변경하고 다시 미분
  : 이 과정을 접선의 기울기가 0인 곳을 향해 W의 값을 변경하는 작업을 반복



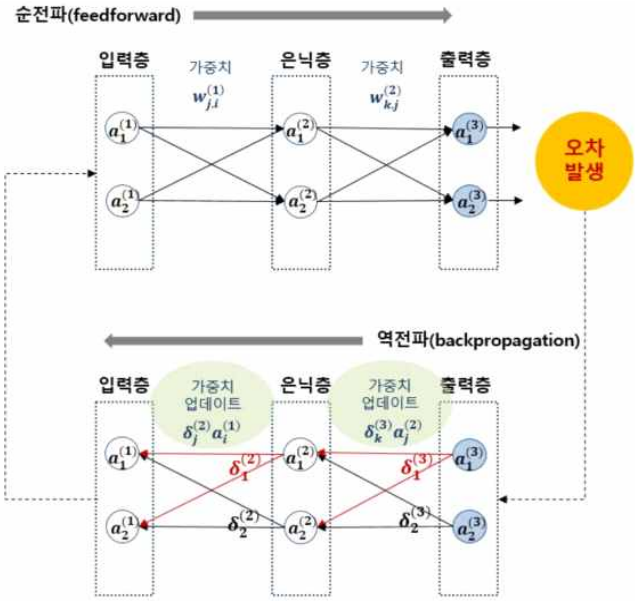
[손실함수를 최소로 하는 W와 b 구하는 과정]



[오차역전파]

순전파 : 입력층에서 출력층으로 계산해 최종 오차를 계산하는 방법

역전파 : 오차 결과 값을 통해서 다시 역으로 input 방향으로 오차가 적어지도록 다시 보내며 가중치를 다시 수정하는 방법



4) 선형 회귀 y = 2x 예측

[선형 회귀 문제]

y = 2x 에 해당하는 값을 예측

훈련(학습) 데이터	테스트 데이터	다음 x에 대해 예측되는 y를 출력
x_train = [1, 2, 3, 4] y_train = [2, 4, 6, 8]	x_test = [1.2, 2.3, 3.4, 4.5] y_test = [2.4, 4.6, 6.8, 9.0]	[3.5, 5, 5.5, 6]

[선형 회귀 케라스 구현(1)]

- 하나의 Dense 층 : 입력은 1차원, 출력도 1차원
- 활성화 함수 linear : 디폴트 값, 입력 뉴런과 가중치로 계산된 결과값이 그대로 출력

```
import tensorflow as tf

# ① 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]

# ② 모델 구성(생성)
model = tf.keras.models.Sequential([
    #                   출력, 입력=여러 개 원소의 일차원 배열, 그대로 출력
    tf.keras.layers.Dense(1, input_shape=(1, ), activation='linear')
    #Dense(1, input_dim=1)
])
```

[선형 회귀 케라스 구현(2)]

확률적 경사하강법(Stochastic Gradient Descent)

- optimizer='SGD‘
- 경사하강법의 계산량을 줄이기 위해 확률적 방법으로 경사하강법을 사용
- 전체를 계산하지 않고 확률적으로 일부 샘플로 계산

```
# ③ 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력 정보를 지정
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse',
              metrics=['mae', 'mse'])
```

mae

- 평균 절대 오차(MAE)
- 모든 예측과 정답과의 오차 합의 평균
- n = 오차의 개수, Σ = 합을 나타내는 기호

mse

- 오차 평균 제곱합(Mean Squared Error, MSE)
- 모든 예측과 정답과의 오차 제곱 합의 평균

[선형 회귀 모델 정보]

```
# 모델을 표시(시각화)
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

[선형 회귀 모델 학습(훈련)]

히스토리 객체

- 매 에포크 마다의 훈련 손실값 (loss)
- 매 에포크 마다의 훈련 정확도 (accuracy)
- 매 에포크 마다의 검증 손실값 (val_loss)
- 매 에포크 마다의 검증 정확도 (val_acc)

```
# ④ 생성된 모델로 훈련 데이터 학습
# 훈련과정 정보를 history 객체에 저장
history = model.fit(x_train, y_train, epochs=500)
```

```
Epoch 374/500
1/1 [==============================] - 0s 1ms/step - loss: 4.2576e-04 - mae: 0.0172 - mse: 4.2576e-04
Epoch 375/500
1/1 [==============================] - 0s 1ms/step - loss: 4.2321e-04 - mae: 0.0171 - mse: 4.2321e-04
Epoch 376/500
1/1 [==============================] - 0s 2ms/step - loss: 4.2068e-04 - mae: 0.0171 - mse: 4.2068e-04
Epoch 377/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1817e-04 - mae: 0.0170 - mse: 4.1817e-04
Epoch 378/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1566e-04 - mae: 0.0170 - mse: 4.1566e-04
Epoch 379/500
1/1 [==============================] - 0s 1ms/step - loss: 4.1318e-04 - mae: 0.0169 - mse: 4.1318e-04
```

[선형 회귀 모델 성능 평가 및 예측]

- 성능 평가

```
# ⑤ 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [2.4, 4.6, 6.8, 9.0]

print('손실', model.evaluate(x_test, y_test))
```

```
1/1 [==============================] - 0s 1ms/step - loss: 0.0012 - mae: 0.0313 - mse: 0.0012
손실: [0.0012317538494244218, 0.031307220458984375, 0.0012317538494244218]
```

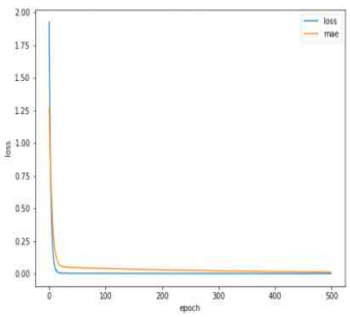
- 예측

```
# x = [3.5, 5, 5.5, 6]의 예측
print(model.predict([3.5, 5, 5.5, 6]))

pred = model.predict([3.5, 5, 5.5, 6])
# 예측 값만 1차원으로
print(pred.flatten())
print(pred.squeeze())
```

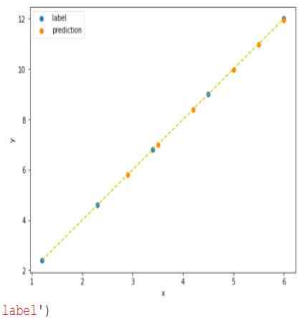
```
[[ 6.9934297]
 [ 9.975829 ]
 [10.969961 ]
 [11.964094 ]]
[ 6.9934297  9.975829  10.969961  11.964094 ]
[ 6.9934297  9.975829  10.969961  11.964094 ]
```

[손실과 mae 시각화]

```
import matplotlib.pylab as plt

# 그래프 그리기
fig = plt.figure(figsize=(8, 6))

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['mae'], label='mae')
#plt.plot(history.history['mse'], label='mse')

plt.legend(loc='best')
plt.xlabel('epoch')
plt.ylabel('loss')
```



[예측 값 시각화]




```
import matplotlib.pylab as plt

x_test = [1.2, 2.3, 3.4, 4.5, 6.0]
y_test = [2.4, 4.6, 6.8, 9.0, 12.0]

# 그래프 그리기
fig = plt.figure(figsize=(8, 6))

plt.scatter(x_test, y_test, label='label')
plt.plot(x_test, y_test, 'y--')

x = [2.9, 3.5, 4.2, 5, 5.5, 6]
pred = model.predict(x)
plt.scatter(x, pred.flatten(), label='prediction')

plt.legend(loc='best')
plt.xlabel('x')
plt.ylabel('y')
```

4) 선형 회귀 y = 2x + 1 예측

[케라스로 예측]

- 케라스와 numpy 사용
- 학습에 3개 데이터

  x = [0, 1, 2, 3, 4] → x[:3]

  y = [1, 3, 5, ?, ?] → y[:3]

- 예측, 뒤 2개 데이터 사용

  x = [0, 1, 2, 3, 4] → x[3:]

  y = [1, 3, 5, ?, ?] → y[3:]

```
import tensorflow as tf
import numpy as np

#훈련과 테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

#인공신경망 모델 사용
model = tf.keras.models.Sequential()

#은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

#모델의 파라미터를 지정하고 모델 구조를 생성
#최적화 알고리즘: 확률적 경사 하강법(SGD: Stochastic Gradient Descent)
#손실 함수(loss function): 평균제곱오차(MSE: Mean Square Error)
model.compile('SGD', 'mse')

#생성된 모델로 훈련 자료로 입력(x[:2])과 출력(y[:2])을 사용하여 학습
#키워드 매개변수 epoch(에폭): 훈련반복횟수
#키워드 매개변수 verbose: 학습진행사항 표시
model.fit(x[:3], y[:3], epochs=1000, verbose=0)

#테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])

#학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```

[케라스로 예측 순서]

① 케라스 패키지 임포트

- import tensorflow as tf
- import numpy as np

② 데이터 지정

- x = numpy.array([0, 1, 2, 3, 4])
- y = numpy.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

③ 인공신경망 모델 구성

- model = tf.keras.models.Sequential()
- model.add(tf.keras.layers.Dense(출력수, input_shape=(입력수,)))

④ 최적화 방법과 손실 함수 지정해 인공신경망 모델 생성

- model.compile(' SGD ' , ' mse ')

⑤ 생성된 모델로 훈련 데이터 학습

- model.fit(…)

⑥ 성능 평가

- model.evaluate(…)

⑦ 테스트 데이터로 결과 예측

- model.predict(…)