



인공지능응용프로그래밍

포트폴리오

이름:염경훈

학번:20161887

반:E반

머리말

2학기 또한 온라인 강의로 수업을 하게 되어 무척 아쉽다. 그래도 1학기보다는 강의의 질이 향상된 거 같고 또한 온라인 강의의 장점이라고도 할 수 있는 영상 다시 보기 등 여러 가지를 활용해 놓친 부분을 다시 공부할 수 있어서 좋은 거 같다. 이번에 배우게 된 인공지능 프로그래밍도 많이 어렵긴 하지만 재미도 있고 앞으로 많은 분야에서 인공지능이 활용될 것이 분명하기 때문에 더욱 열심히 공부를 하는 중이다. 아직 많이 어렵지만 포트폴리오를 만들며 놓쳤던 부분을 다시 한번 점검하고 직접 공부도 해보며 완성해 갈 때쯤엔 더욱 성장해 있었으면 좋겠다

1. 인공지능과 딥러닝 개요

가. AI 시작

1) 앨런 튜링

- 1950년, 논문 <Computing machinery and untelligence>을 발표
- 인공지능 실험

2) 인공지능의 처음 사용

- 1956년 다트머스대 학술대회: 세계 최초의 AI프로그램인 논리 연산기 발표

나. AI와 딥러닝 역사

1) 1940년대부터 시작한 분야

- 두 번의 흑한기를 지냄

2) 2010년 이후 여러 문제 해결

- 최고의 전성기를 누림

다. 딥러닝의 인기

1) 딥러닝의 문제가 해결되고 있는 과정

- 빅데이터, 계산 속도, 알고리즘

라. 인공지능과 머신러닝, 딥러닝

1) 인공지능(AI: Artificial Intelligence)

- 컴퓨터가 인간처럼 지적 능력을 갖게 하거나 행동하도록 하는 모든 기술
- 머신러닝

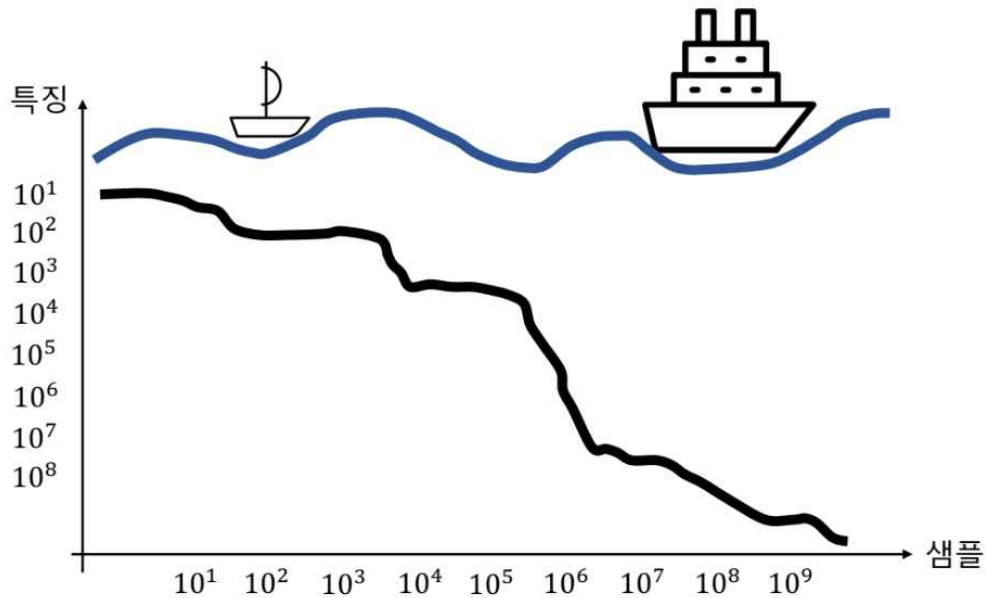
(1) 기계가 스스로 학습할 수 있도록 하는 인공지능의 한 연구 분야

(2) SVM: 수학적 방식의 학습 알고리즘

(3) 딥러닝: 다중 계층의 신경망 모델을 사용하는 머신러닝의 일종

마. 기계학습과 딥러닝

1) 특징과 데이터가 많을수록 딥러닝에 적합



바. 머신러닝

1) 기계학습이라고도 부르는 머신러닝

- 주어진 데이터를 기반으로 기계가 스스로 학습하여 성능을 향상 시키거나 최적의 해답을 찾기 위한 학습 방법
- 스스로 데이터를 반복적으로 학습하여 기술을 터득하는 방식

사. 머신러닝 분류 개요

1) 머신러닝은 지도학습과 자율학습, 그리고 강화학습으로 분류

- 지도학습
 - (1) 입력과 출력의 쌍으로 구성된 정답의 훈련 데이터로부터 입출력간의 함수를 학습시키는 방법
- 비지도학습
 - (2) 정답이 없는 훈련 데이터를 사용하여 데이터 내에 숨어 있는 관계를 찾아내는 방법
- 강화학습
 - (3) 잘한 행동에는 보상을 주고 잘못된 행동에 대해 벌을 주는 경험을 통해 지식을 학습하는 방법

아. 머신러닝과 딥러닝 비교

머신 러닝과 딥 러닝의 차이점

	기계 학습	딥 러닝
데이터의 존성	중소형 데이터 세트에서 탁월한 성능	큰 데이터 세트에서 뛰어난 성능
하드웨어 의존성	저가형 머신에서 작업하십시오.	GPU가있는 강력한 기계가 필요합니다. DL은 상당한 양의 행렬 곱셈을 수행합니다.
기능 공학	데이터를 나타내는 기능을 이해해야 함	데이터를 나타내는 최고의 기능을 이해할 필요가 없 습니다
실행 시간	몇 분에서 몇 시간	최대 몇 주. 신경망은 상당한 수의 가중치를 계산해 야합니다.

2. 인공신경망과 DNN

가. 인공신경망에서 시작된 딥러닝

1) 퍼셉트론

- 세계 최초의 인공 신경망을 제안: 프랭크 로젠블랫
- 신경망에서는 방대한 양의 데이터를 신경망으로 유입

2) 현재 발전해 여러 분야에서 활용

- 자율비행, 자율주행, 필기체 인식, 음성인식, 언어 번역

나. ANN 개요

1) 인공 신경망(ANN: Artificial Neural Network) 사용

- 인간의 뇌는 1000억개의 뉴런으로 구성
- 인간의 신경세포인 뉴런을 모방하여 만든 가상의 신경

다. 인공신경망 구조와 MLP

1) MLP(Multi Layer Perceptron)

- 입력층과 출력층: 다수의 신호를 받아서 하나의 신호를 출력
- 중간층의 은닉층: 여러개의 층으로 연결하여 하나의 신경망을 구성

라. DNN(deep neural network)

1) 심층신경망

- 다중 계층인 심층신경망을 사용

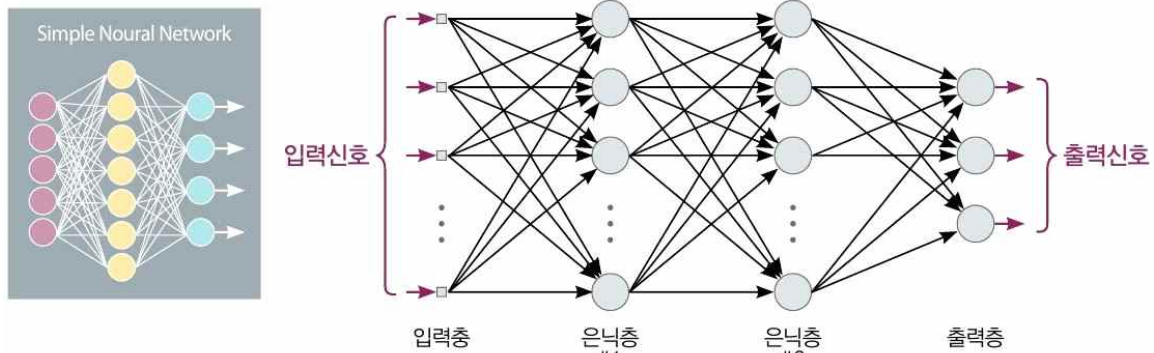


그림 10.28 ▶ 단일계층과 딥러닝의 다중계층 신경망

3. DNN 활용과 GPU

가. 딥러닝 활용

1) 이세돌을 이긴 알파고

2) 다양한 분야에서 활용

- 인간과 대화하는 지능형 에이전트와 실시간 채팅이 가능한 챗봇
- 얼굴을 비롯한 생체인식, 사물인식, 자동차 번호판 인식등
- x-ray사진 판독과 각종 진단등의 의료 분야
- 자율비행이나 자율주행 분야
- 주식이나 펀드, 환율, 일기예보 등의 예측 분야

나. 구글 딥마인드

1) 딥마인드

- 원래 데미스 하사비스가 창업한 영국의 벤처기업
- 2014년 구글이 인수

2) 2016년 알파고

- 구글의 딥마인드에서 개발한 인공지능 바둑 프로그램

다. 그래픽 처리 장치 GPU

- 1) 그래픽 연산 처리를 하는 전용 프로세서
- 2) GPU란 용어는 1999년 엔디비아에서 처음 사용

라. GPGPU

- 1) 일반 CPU 프로세서를 돕는 보조 프로세서로서의 GPU
- 2) 중앙 처리 장치가 맡았던 응용 프로그램들의 계산에 GPU를 사용하는 기술
- 3) 딥러닝의 심층신경망에서 빅데이터를 처리하기 위해 대량의 행렬과 벡터를 사용
- 4) 12개 GPU가 2000개의 CPU와 비슷한 계산 능력

마. CUDA

- 1) GPU 업체인 NVIDIA의 GPU를 사용하기 위한 라이브러리 소프트웨어

바. 구글의 TPU

- 1) 데이터 분석 및 딥러닝용 칩으로서 벡터, 행렬 연산의 병렬처리에 특과

4. 텐서플로 개요

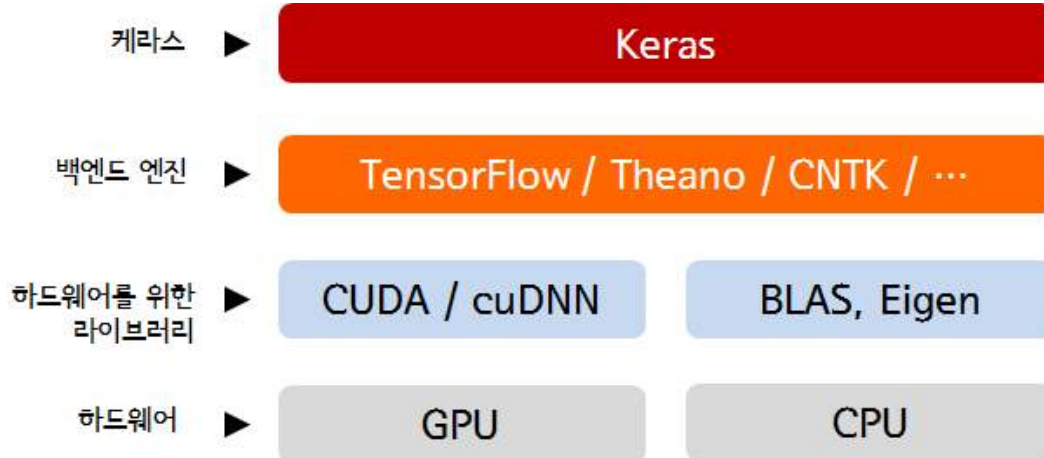
가. 딥러닝 라이브러리 개요

- 1) 딥러닝 구현을 위한 클래스 및 함수 제공
- 2) 다양한 라이브러리 활용
 - 텐서플로, 케라스, 파이토치
 - 가장 적합한 언어는 파이썬

나. 케라스의 개요

- 1) 원래는 독자적인 고수준 라이브러리

2) 현재는 Tensorflow의 고수준 API로도 사용



다. 텐서플로 개요

1) 구글에서 만든 라이브러리

- 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리
- Python, Java, Go 등 다양한 언어를 지원
- 텐서플로 자체는 기본적으로 C++로 구현

라. 텐서 개요

1) Tensor: 모든 데이터

- 딥러닝에서 데이터를 표현 하는 방식
 - (1) 0차원 텐서 : 스칼라
 - (2) 1차원 텐서 : 벡터
 - (3) 2차원 텐서 : 행렬 등
 - (4) n차원 행렬 : 텐서는 행렬로 표현할 수 있는 n차원 형태의 배열을 높은 차원으로 확장

마. TensorFlow 계산 과정

1) Session

- 세션 생성
- 세션 사용
- 세션 종료

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

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

5. 개발환경

가. 주 개발 환경

- 1) 구글의 Colab
- 2) 구글 계정 필요

나. 구글 코랩

- 1) 클라우드 기반의 무료 Jupyter 노트북 개발 환경
- 2) Google Drive + Jupyter Notebook
- 3) 장점
 - 구글 드라이브와 연계
 - 깃허브와 연계

6. 텐서플로 기초 프로그래밍

가. tensorflow 불러오기

- 1) 코랩에서 버전 사용방법


```
%tensorflow_version 1.x
```
- 2) 텐서 출력
 - 값만 보려면 메소드 `numpy()`

나. Tensor Shape, Type

Data type	Python type	Description
DT_FLOAT	<code>tf.float32</code>	32 bits floating point.
DT_DOUBLE	<code>tf.float64</code>	64 bits floating point.
DT_INT8	<code>tf.int8</code>	8 bits signed integer.
DT_INT16	<code>tf.int16</code>	16 bits signed integer.
DT_INT32	<code>tf.int32</code>	32 bits signed integer.
DT_INT64	<code>tf.int64</code>	64 bits signed integer.

다. Shape, Type 예제 코드

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

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

```
↳ TensorShape([3])
```

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

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

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

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

```
▶ 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)>
```

라. 조건 연산 `tf.cond()`

1) `tf.cond(pred, true_fn=None, false_fn=None, name=None)`

- `pred`를 검사해 참이면 `true_fn`반환 거짓이면 `false_fn`반환

마. `tf.cond()` 예제 코드

```
[ ] 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
```

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

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

tf.Tensor(11.0, shape=(), dtype=float32)
11.0
```

```
[ ] x = tf.constant(2.)
    bool = tf.constant(True)
    def f1(): return tf.multiply(x, 2)
    def f2(): return tf.add(x, 10)

    r = tf.cond(bool, f1, f2)
    print(r)
    print(r.numpy())

tf.Tensor(4.0, shape=(), dtype=float32)
4.0
```

바. 1차원 배열 텐서 예제코드

```
▶ t = tf.constant([1,2,3])
  t

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

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

```
print(x+y)
print([(x+y).numpy()])
```

```
↳ tf.Tensor([ 6  8 10], shape=(3,), dtype=int32)
   [ 5 12 21]
```

```
[ ] a = tf.constant([1,2])
    b = tf.constant([5,6])
```

```
a*b
```

```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([ 5, 12], dtype=int32)>
```

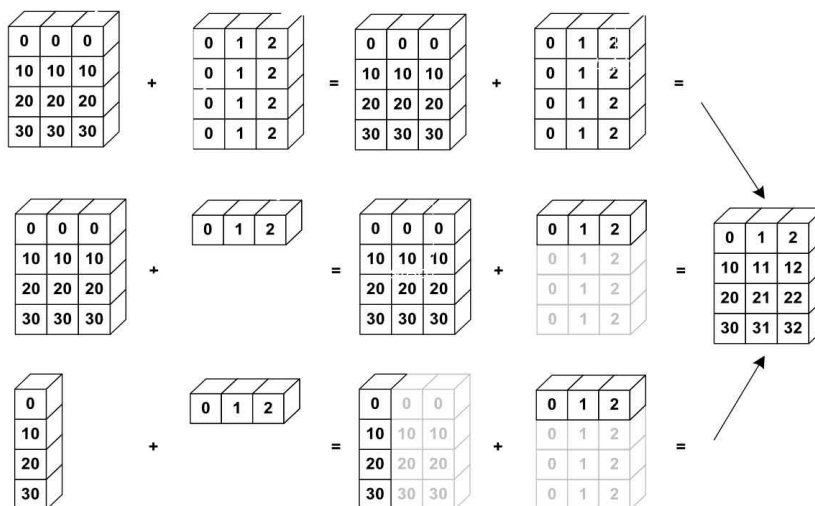
```
[ ] c = tf.constant([4,5])
    print(a*b+c)
```

```
tf.Tensor([ 9 17], shape=(2,), dtype=int32)
```

사. 배열 텐서 연산

1) 텐서의 브로드 캐스팅

- Shape이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용하여 Shape을 맞춤



아. 브로드캐스팅 코드1

```
[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)
```

자. 브로드캐스팅 코드2

```
[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]]
```

차. tf.add() 코드

```
[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

카. 행렬 곱셈

1) Numpy

- `np.dot(a,b)`
- `a.dot(b)`

A

B

A * B

2) Tf

- `tf.matmul`

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix}$$

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

타. `tf.matmul()`

```
[50] # 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())

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

```
↳ [[ 4.  4.]
    [10.  8.]
    [[ 2.  4.]
     [ 7. 10.]
```

파. 행렬의 같은 위치 원소와의 곱

▼ 행렬, 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)
```

하. 텐서플로 연산

- 1) `tf.multiply()`
- 2) `tf.pow()`
- 3) `tf.reduce_mean()`
- 4) `tf.reduce_sum()`

```
[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

5) `tf.rank`

```
[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>`

6) `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
matrixB = tf.reshape(matrix, [3, -1])
matrixB

```
<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)>
```

7) Reshape에서 -1 사용

[25] # 기존 내용을 4x3x5 텐서로 형태 변경
matrixAlt = tf.reshape(matrixB, [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.]],

        [[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.]],

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

8) 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
```

9) 변수 Variable

- `assign`, `assign_add`: 값을 변수에 할당
- `read_value`: 변수 값 읽기

7. 텐서플로의 난수 활용

가. 균등분포 난수

1) `tf.random.uniform([1], 0, 1)`

- (배열, 시작, 끝)

```
[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)
```

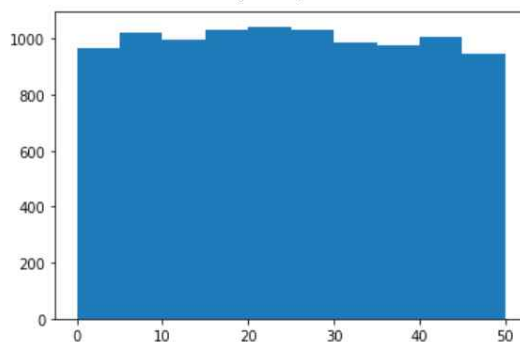
```
[11] 1 rand = tf.random.uniform([1000],0,10)
      2 print(rand[:10])
```

```
↳ tf.Tensor(
[5.1413307 1.548909 8.911686 9.880335 5.5388713 5.6710424 6.80269
 1.9444573 7.549943 6.573516 ], shape=(10,), dtype=float32)
```

2) 균등분포 1000개 그리기

```
[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>)
```



나. 정규분포 난수

1) tf.random.normal([4],0,1)

- (크기, 평균, 표준편차)

```
[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)
```

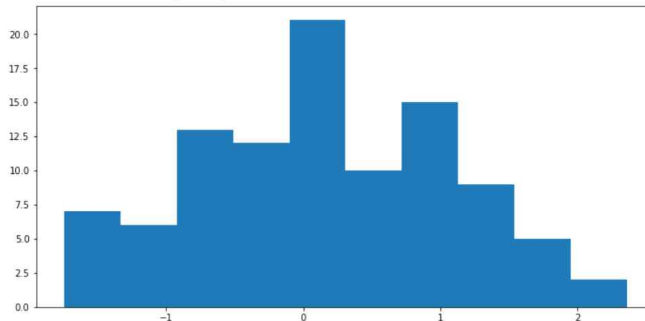
```
[54] 1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([2, 4],0,2)
      3 print(rand)
```

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

2) 정규분포 100개 그리기

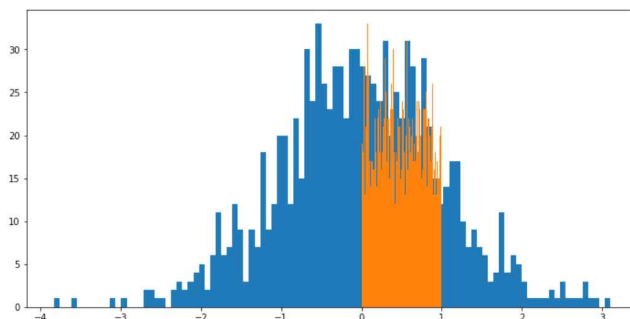
```
[52] 1 import matplotlib.pyplot as plt
      2 rand = tf.random.normal([100], 0, 1)
      3 plt.hist(rand, bins=10)

↳ (array([ 7.,  6., 13., 12., 21., 10., 15.,  9.,  5.,  2.]),
array([-1.7424849, -1.392153 , -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 정규분포 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)
```



다. shuffle

1) tf.random.shuffle(a)

```
[25] 1 import numpy as np
      2 a = np.arange(10)
      3 print(a)
      4 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] 1 import numpy as np
      2 a = np.arange(20).reshape(4, 5)
      3 a

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

```
[27] 1 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]])>
```

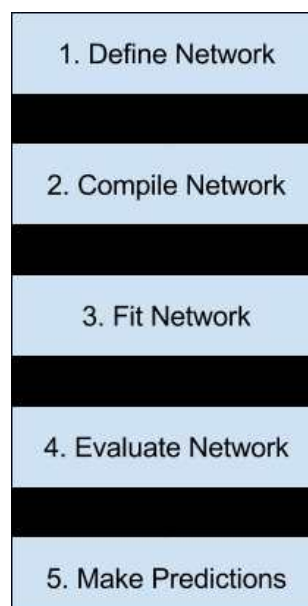
8. MNIST 이해를 위한 손글씨 보기

가. MNIST

- 1) 미국 국립 표준기술원
- 2) 손으로 쓴 자릿수에 대한 데이터 집합
- 3) 필기 숫자 이미지와 접합인 레이블의 쌍으로 구성
- 4) 필기 숫자 이미지

나. 케라스 딥러닝 구현

- 1) 딥러닝 모델 만들기
 - define
- 2) 주요 훈련 방법 설정
 - compile
- 3) 훈련시키기
 - fit
- 4) 테스트 데이터를 평가
 - evaluate
- 5) 정답예측
 - predict



다. MNIST 손글씨 데이터

로드 코드

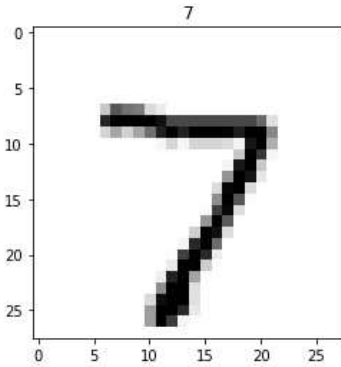
5) 테스트 데이터 첫 손글씨

```

n = 0
ttl = str(y_test[n])
plt.figure(figsize=(6,4))
plt.title(ttl)
plt.imshow(x_test[n], cmap='Greys')

```

<matplotlib.image.AxesImage at 0x7ffb27e51550>



6) 0~59999중의 임의 수 20개 선택

```

from random import sample
nrows, ncols = 4,5

idx = sorted(sample(range(len(x_train)), nrows * ncols))
print(idx)

```

[616, 14498, 15580, 23878, 26573, 29175, 29554, 32312, 33352, 35805, 40186, 40523, 42968, 49840, 50021, 51754, 55070, 58540, 58897,

7) 랜덤하게 20개

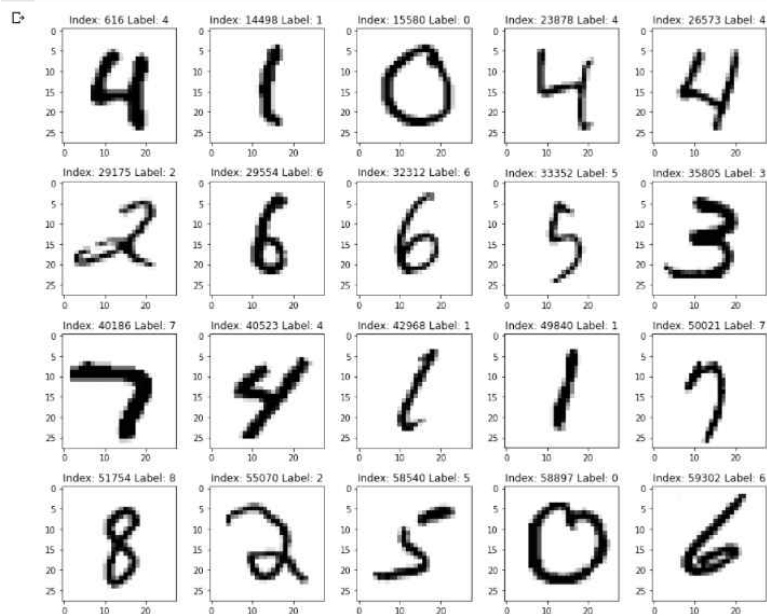
```

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()

```



9. MNIST데이터 딥러닝 모델 적용 예측

가. 딥러닝 과정

- 1) 0에서 9까지의 분류
 - 클래스 10개
- 2) 모델구성
 - 블랙 박스
- 3) 모델 훈련: train
 - 모델이 문제를 해결하도록 훈련
 - 학습 방법 및 모니터링 지표 설정
 - (1) 경사하강법
 - (2) 손실함수
 - (3) 모니터링 지표
- 4) 예측

나. 딥러닝 구현

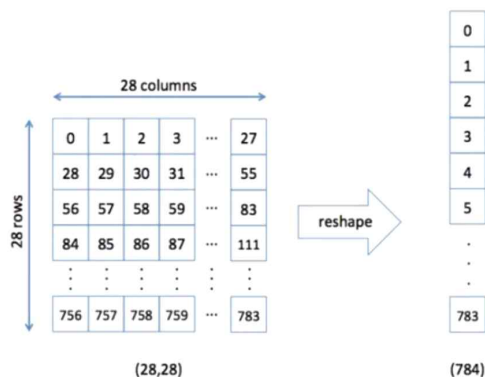
1) 데이터 전처리

```
[ ] x_train, x_test = x_train/ 255.0, x_test/255.0
```

2) 모델 구성

```
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')
])
```

- Flatten(input_shape=(28,28)),
 - (1) 모델에서 2차원 그림을 1차원으로 평탄화



- Dense: 완전연결층

3) 구성된 모델 요약 및 표시

```
[ ] model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

```
model.summary()
```

```
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) 모델 훈련

```
[ ] model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.2950 - accuracy: 0.9137
Epoch 2/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.1449 - accuracy: 0.9563
Epoch 3/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.1058 - accuracy: 0.9680
Epoch 4/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0880 - accuracy: 0.9731
Epoch 5/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.0757 - accuracy: 0.9759
<tensorflow.python.keras.callbacks.History at 0x7ffb1b8fdef0>
```

5) 모델 평가

```
[ ] model.evaluate(x_test, y_test)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.0715 - accuracy: 0.9786
[0.07146061956882477, 0.978600025177002]
```

10. MNIST 손글씨 예측과 오류 확인

가. 첫 번째 손글씨 예측 결과를 확인

1) model.predict(input)

```
print(x_test[:1].shape)
pred_result = model.predict(x_test[:1])
print(pred_result.shape)
print(pred_result)
print(pred_result[0])

(1, 28, 28)
(1, 10)
[[3.14520236e-07 1.25305314e-06 8.25220013e-07 5.56669023e-04
 1.15167244e-10 4.46474431e-08 2.59965638e-12 9.99435723e-01
 9.60898916e-08 5.08073435e-06]]
[[3.14520236e-07 1.25305314e-06 8.25220013e-07 5.56669023e-04
 1.15167244e-10 4.46474431e-08 2.59965638e-12 9.99435723e-01
 9.60898916e-08 5.08073435e-06]]
```


나. One Hot Encoding

- 1) 데이터가 취할 수 있는 모든 단일 범주에 대해 하나의 새 열을 생성

다. 배열에서 가장 큰 값의 첨자 구하기

1) np.argmax()

```
import numpy as np

print(np.argmax([5,4,10,1,2]))
print(np.argmax([3,1,4,9,6,7,2]))
print(np.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

2
3
[1 0 2]

2) tf.argmax()

```
print(tf.argmax([5,4,10,1,2]))
print(tf.argmax([3,1,4,9,6,7,2]))
print(tf.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

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

11. MNIST 손글씨 예측과 결과 확인

가. 활성화 함수 softmax()

나. 평탄화 메소드 flatten

다. 드롭아웃 개념

- 1) 층에서 결과 값을 일정 비율로 제거하는 방법
 - 오버피팅 문제를 해결하는 정규화 목적을 위해서 필요

라. 테스트 데이터 모두 예측해보기

▶ `from random import sample`

```
pred_result = model.predict(x_test)
print(pred_result.shape)
print(pred_result[0])
print(np.argmax(pred_result[0]))

pred_labels = np.argmax(pred_result, axis=1)
print(pred_labels)
print(y_test)
```

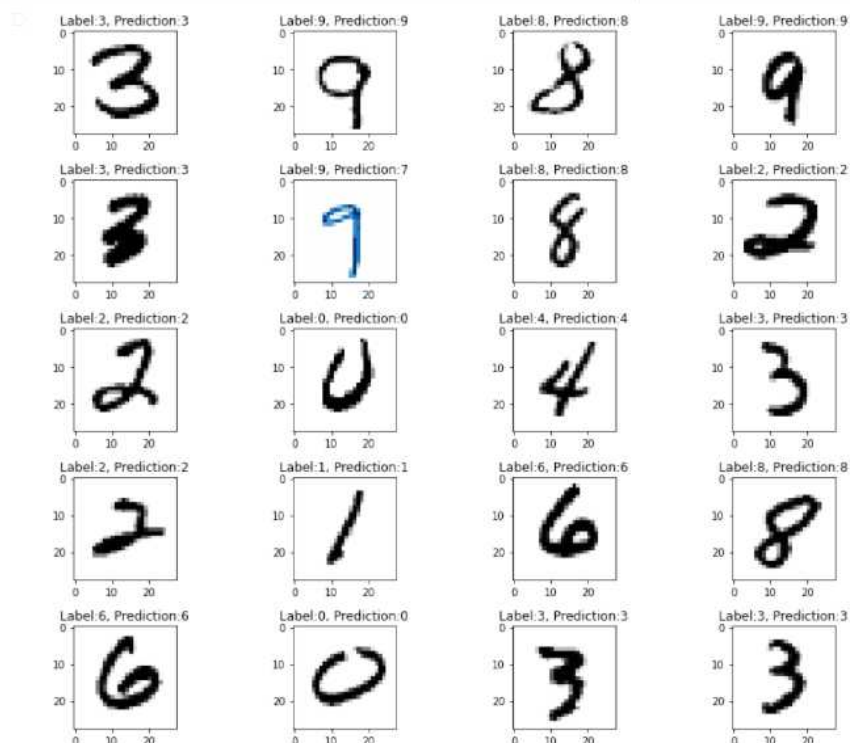
```
↳ (10000, 10)
[3.1451995e-07 1.2530519e-06 8.2522155e-07 5.5666850e-04 1.1516681e-10
 4.4647358e-08 2.5996516e-12 9.9943572e-01 9.6089899e-08 5.0807248e-06]
7
[7 2 1 ... 4 5 6]
[7 2 1 ... 4 5 6]
```

마. 임의의 20개 샘플 예측값과 정답 그리기

```
① n_rows, n_cols = 5, 4
samples = sorted(sample(range(len(x_test)), n_rows * n_cols))
```

```
[ ] count = 0
plt.figure(figsize=(12,10))
for n in samples:
    count += 1
    plt.subplot(n_rows, n_cols, count)
    cmap = 'Greys' if (pred_labels[n] == y_test[n]) else 'Blues'
    plt.imshow(x_test[n].reshape(28,28), cmap=cmap, interpolation='nearest')
    tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

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



바. 메소드 `flatten()` 미사용

```
[ ] x_train = x_train.reshape((60000, 28*28))
    x_test = x_test.reshape((10000, 28*28))

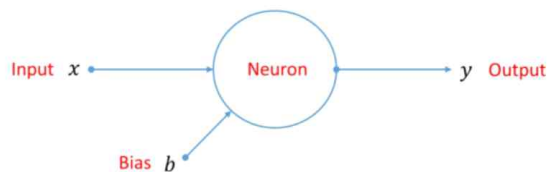
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape = (28*28,)),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

12. 인공 신경망 퍼셉트론의 이해

가. 인공 신경 세포

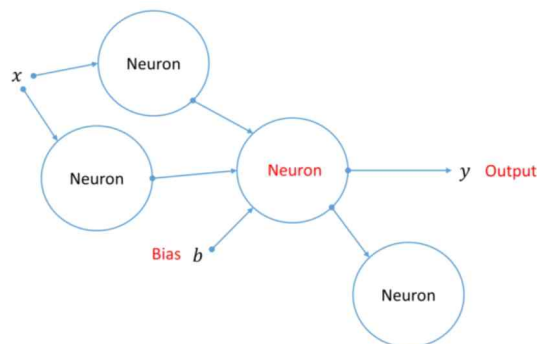
1) 뉴런

- 입력
- 편향



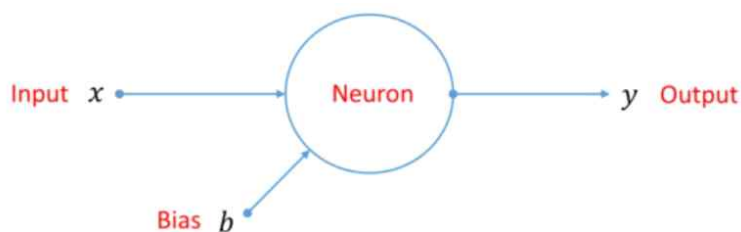
2) 신경망

- 뉴런의 연결



나. 입력과 출력

1) 편향을 조정해 출력을 맞춤

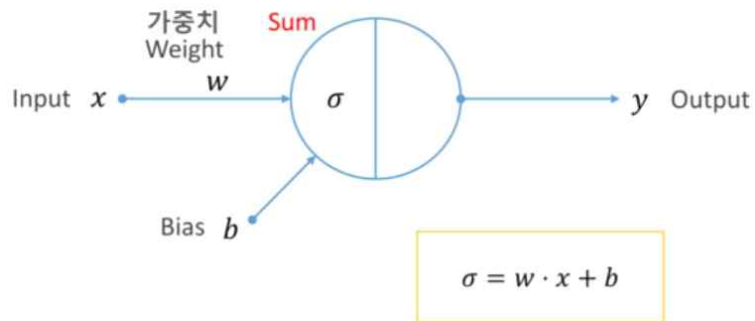


Input x	Output y
Size of house	Price
Time spent for studying	Score in exam

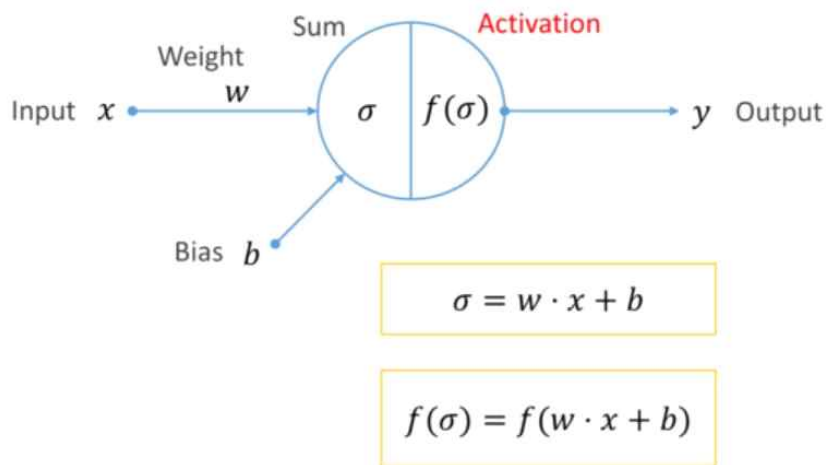
다. 뉴런 연산

1) 뉴런식

- $a = wx + b$



라. 활성화 함수

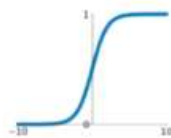


마. 다양한 활성화 함수 종류

Activation Functions

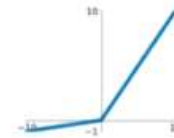
Sigmoid

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



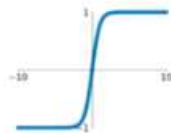
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ReLU

$$\max(0, x)$$



ELU

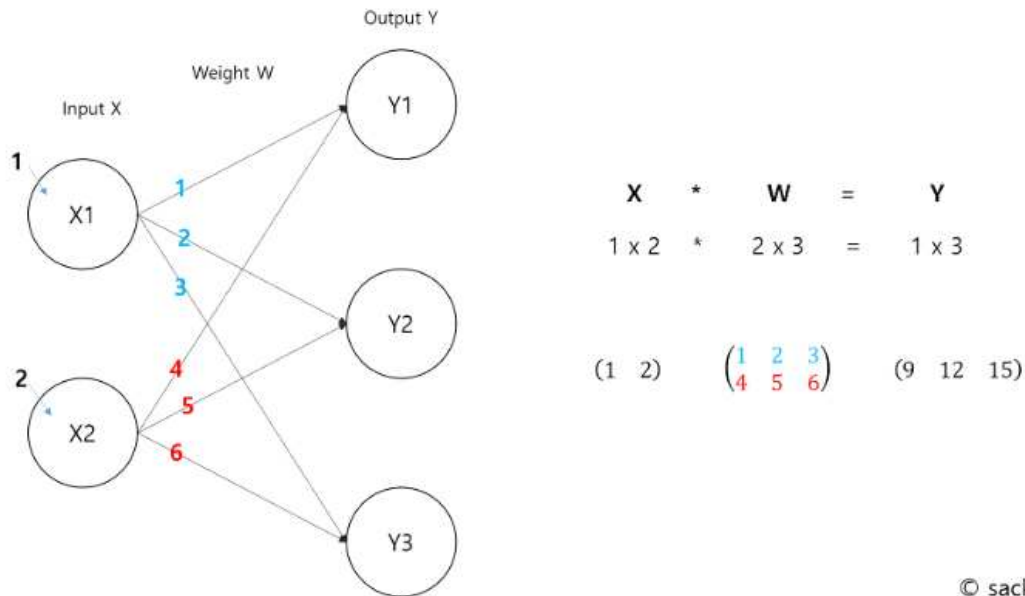
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Different Activation Functions and their Graphs

13. 인공 신경망 행렬 연산

가. 계산 사례



14. 활성화 함수 그리기

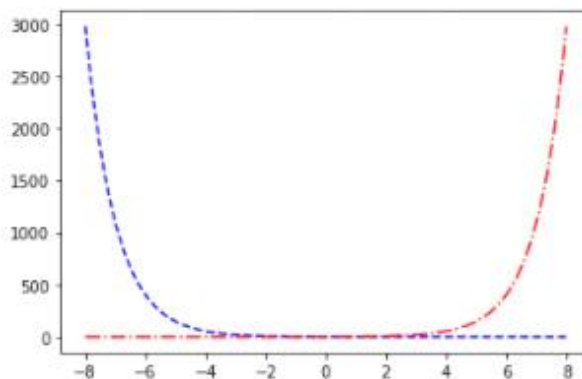
가. 자연수와 자연수의 지수 승

```
import numpy as np
np.e
```

2.718281828459045

```
[ ] import matplotlib.pyplot 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-.')
```

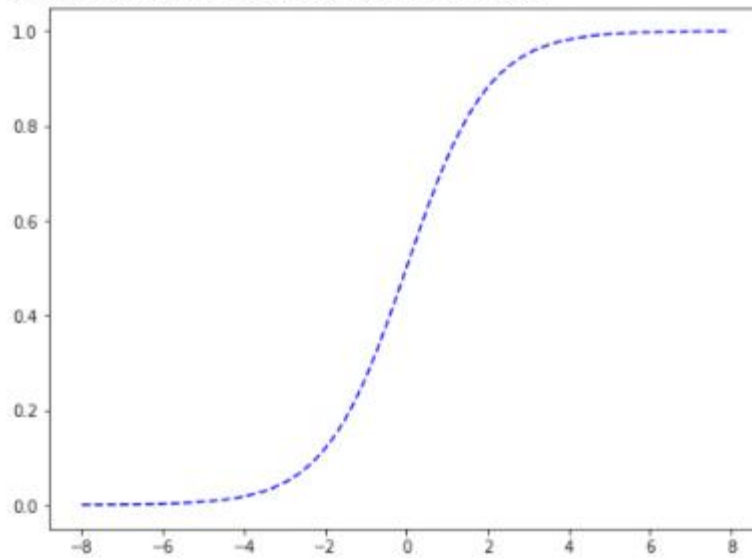


나. 시그모이드 함수

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

plt.figure(figsize=(8, 6))
x = np.linspace(-8, 8, 100)
plt.plot(x, sigm_func(x), 'b--')
```

[<matplotlib.lines.Line2D at 0x7f386835e6d8>]

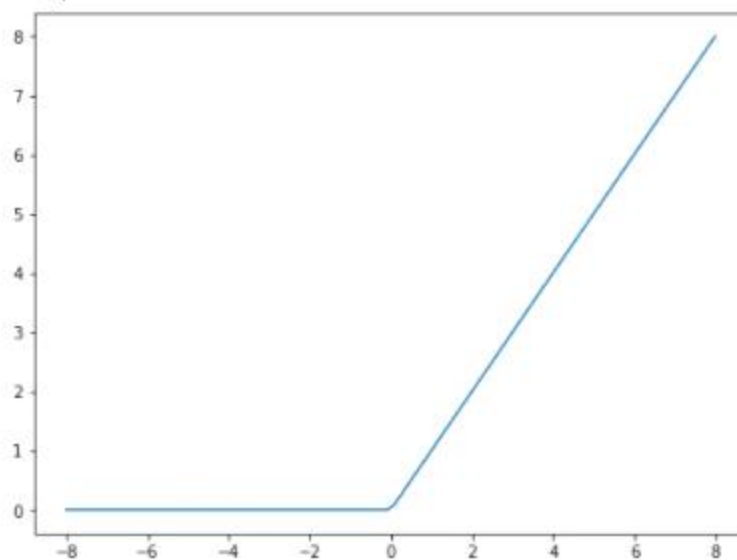


다. ReLU 함수

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

plt.figure(figsize=(8, 6))
x = np.linspace(-8, 8, 100)
plt.plot(x, relu_func(x))
```

[<matplotlib.lines.Line2D at 0x7f3867e49c50>]



라. 활성화 함수 결과

```
def identity_func(x):
    return x;

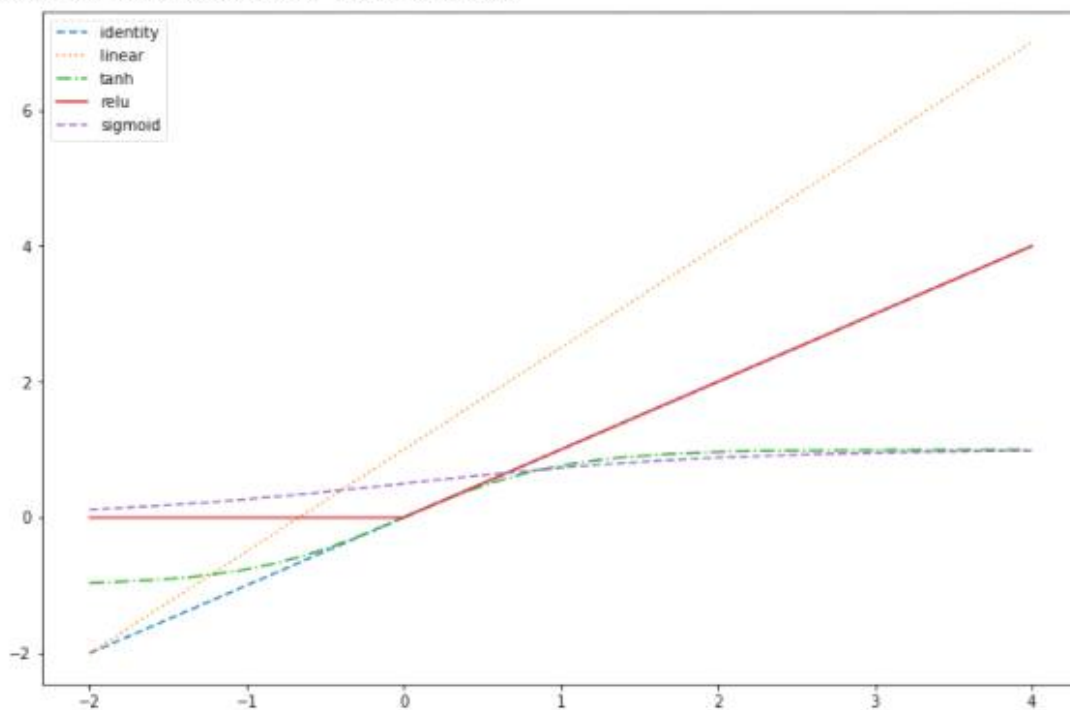
def linear_func(x):
    return 1.5 * x + 1

def tanh_func(x):
    return np.tanh(x)

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, sigmoid_func(x), linestyle='--', label="sigmoid")
plt.legend(loc="upper left")
```

<matplotlib.legend.Legend at 0x7f3867d57b70>



15. 인공 신경망 행렬 연산 코드

▼ 뉴런의 행렬 연산

```
[14] x = [[1, 2]]
      w = [[1, 2, 3], [4, 5, 6]]

      y = tf.matmul(x, w)
      y.numpy()
```

```
↳ array([[ 9, 12, 15]], dtype=int32)
```

가. 특징2, 샘플수 4개의 행렬 연산

```
x = [[6,5]]
w = [[1,2,3], [4,5,6]]

y = tf.matmul(x,w)
y.numpy()
```

```
↳ array([[26, 37, 48]], dtype=int32)
```

```
[ ] x = [[6,5], [4,7], [5,6], [6,7]]
      w = [[1,2,3], [4,5,6]]

      y = tf.matmul(x,w)
      y.numpy()
```

```
array([[26, 37, 48],
       [32, 43, 54],
       [29, 40, 51],
       [34, 47, 60]], dtype=int32)
```

나. 행렬의 순서를 바꾼 계산

```
▶ w = [[1,4], [2,5], [3,6]]
      x = [[6,4,5,6], [5,7,6,7]]

      y = tf.matmul(w,x)
      y.numpy()
```

```
↳ array([[26, 32, 29, 34],
       [37, 43, 40, 47],
       [48, 54, 51, 60]], dtype=int32)
```


16. 논리 게이트 AND XOR 신경망

가. AND게이트

```

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"

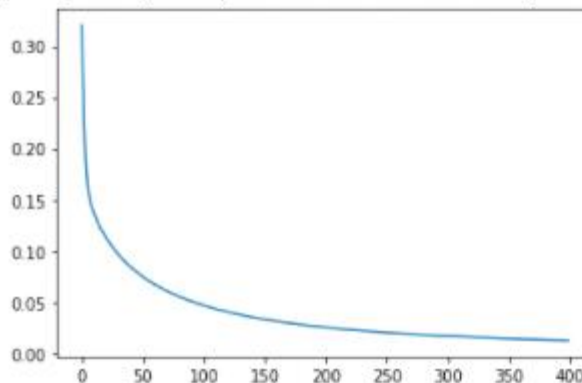
Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	3

Total params: 3
Trainable params: 3
Non-trainable params: 0

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

```
[5] import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

[<matplotlib.lines.Line2D at 0x7ff8257d2358>]



```

model.predict(x)

array([[0.85832906],
       [0.11893138],
       [0.11898062],
       [0.00299984]], dtype=float32)

```

나. XOR 게이트

```

▶ 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()

```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	6
dense_2 (Dense)	(None, 1)	3
Total params: 9		
Trainable params: 9		
Non-trainable params: 0		

```

▶ history = model.fit(x, y, epochs=2000, batch_size=1)

```

```

[ ] print(model.predict(x))

```

```

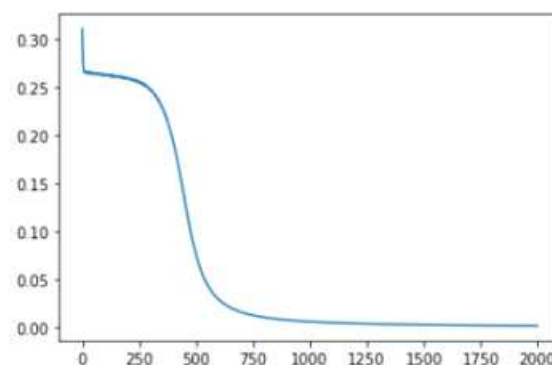
[[0.0265516 ]
 [0.9724283 ]
 [0.9724369 ]
 [0.02379802]]

```

```

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

```



17. 회귀와 분류

가. 회귀 모델

- 1) 연속적인 값을 예측

나. 분류 모델

- 1) 불연속적인 값을 예측

18. 선형 회귀(linear regression)

가. 단순 선형 회귀 분석

- 1) 입력: 특징이 하나
- 2) 출력: 하나의 값

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

나. 다중 선형 회귀 분석

- 1) 입력: 특징이 여러개
- 2) 출력: 하나의 값

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

다. 로지스틱 회귀

- 1) 이진 분류
- 2) 입력: 하나 또는 여러개
- 3) 출력: 0 아니면 1

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

라. 인공지능이란? w 와 b 구하기

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

- 1) 다음식에서 가중치 w 와 편향 b 를 구하기

마. 주요 용어

1) 가설

- 가중치와 편향
- 기울기와 절편

2) 손실 함수

- MSE(Mean Square Error 평균제곱오차)
- Categorical cossentropy
- Sparse Categorical crossentropy

3) 경사 하강법

- 내리막 경사 따라 가기

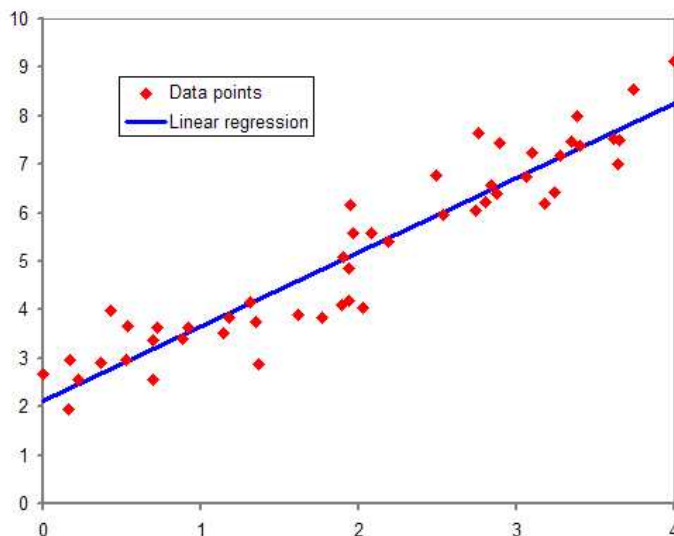
4) 학습률

- 대표적인 하이퍼패러미터

바. 선형 회귀

1) Linear regression

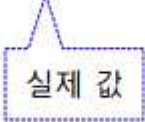
- 데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법
- $Y = wX + b$ 에서 가중치 w 와 편향 b 를 구하는 것



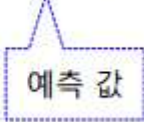
사. 손실 함수(Loss function)

- 1) 실제 값과 가설로부터 얻은 예측 값의 오차를 계산하는 식
- 2) 목적함수, 비용함수라고도 부름
- 3) 평균 제곱 오차(MSE)등을 사용

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



실제 값

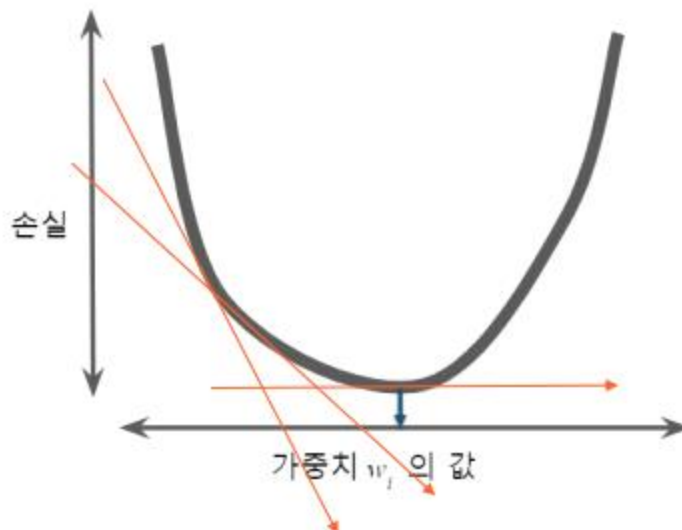


예측 값

19. 최적화 과정

가. 옵티마이저

- 1) 최적화 알고리즘
- 2) 적절한 w 와 b 를 찾아내는 방법(경사 하강법)



나. 학습률

- 1) 기울기에 학습률을 곱하여 다음 지점을 결정
- 2) 너무 적게 설정하면 학습 시간이 매우 오래걸림
- 3) 0.001에서 0.1정도 사용

다. 비용 함수와 최적의 w구하기

1) 비용 함수

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

2) Cost를 최소화하는 W를 구는 식

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

학습률(learning rate)

W 함수의 기울기: 미분 값

라. 오차역전파

1) 순전파

- 입력층에서 출력층으로 계산해 최종 오차를 계산하는 방법

2) 역전파

- 오차 결과 값을 통해서 다시 역으로 input방향으로 오차가 적어지도록 다시 보내며 가중치를 다시 수정하는 방법
- 엄청난 처리 속도의 증가

20. 선형회귀 $y=2x$ 예측

```

▶ import tensorflow as tf

x_train = [1,2,3,4]
y_train = [2,4,6,8]

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(1, input_shape=(1,), activation='linear')
])

```

```

[ ] model.compile(optimizer='SGD', loss='mse', metrics=['mae', 'mse'])

```

```

[ ] model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	2

Total params: 2

Trainable params: 2

Non-trainable params: 0

```

▶ history = model.fit(x_train, y_train, epochs=500)

```

```

[ ] 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: 0.0053 - mae: 0.0648 - mse: 0.0053
 손실 [0.005275942850857973, 0.06479358673095703, 0.005275942850857973]

```

[ ] print(model.predict([3.5, 5, 5.5, 6]))

```

```

pred = model.predict([3.5, 5, 5.5, 6])
print(pred.flatten())
print(pred.squeeze())

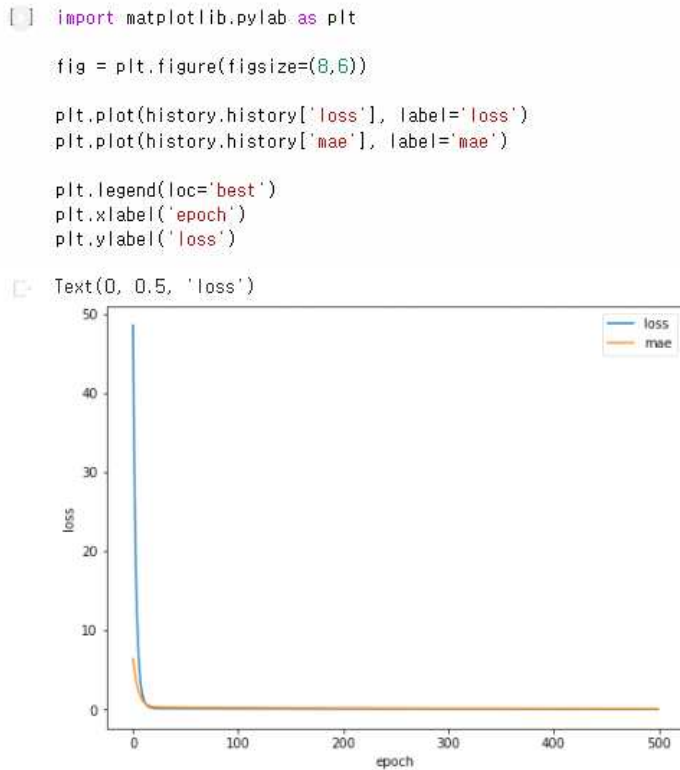
```

```

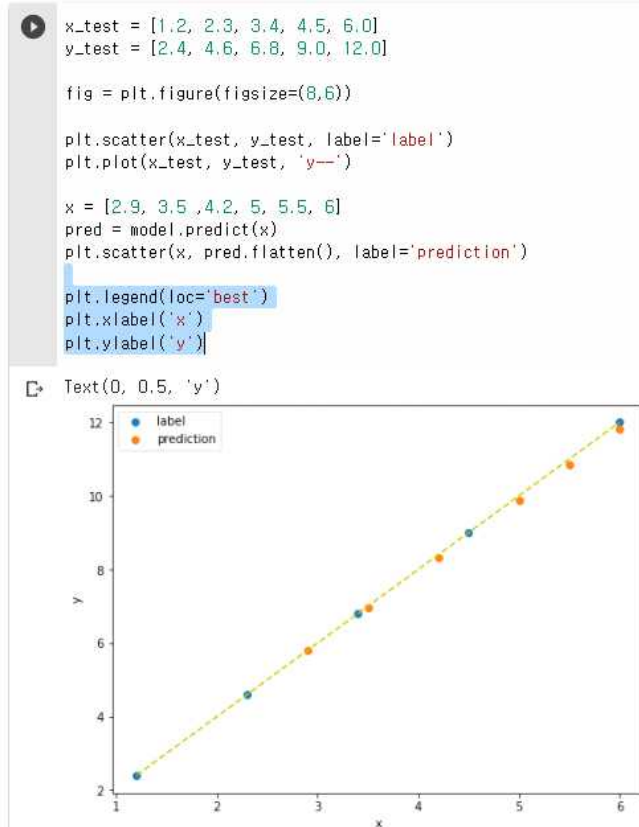
[[ 6.9670215]
 [ 9.878667 ]
 [10.8492155]
 [11.819763 ]]
[ 6.9670215  9.878667 10.8492155 11.819763 ]
[ 6.9670215  9.878667 10.8492155 11.819763 ]

```

가. 속실과 mae 시각화



나. 예측 값 시각화



21. 선형 회귀 $y=2x+1$ 예측

```

import numpy as np

x = np.array([0,1,2,3,4])
y = np.array([1,3,5,7,9])

model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

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

model.fit(x[:3], y[:3], epochs=1000, verbose=0)

print("Targets(정답):", y[3:])

print('Predictions(예측):', model.predict(x[:3].flatten()))

```

```

Targets(정답): [7 9]
Predictions(예측): [[1.0004959]
 [3.0001383]
 [4.9997807]]

```

22. 2018년 대한민국 인구증가율과 고령 인구비율

```

# 그림 4.2 출력 코드
import math

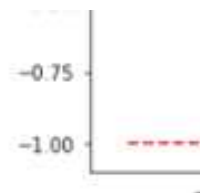
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

x = np.arange(-5, 5, 0.01)
sigmoid_x = [sigmoid(z) for z in x]
tanh_x = [math.tanh(z) for z in x]

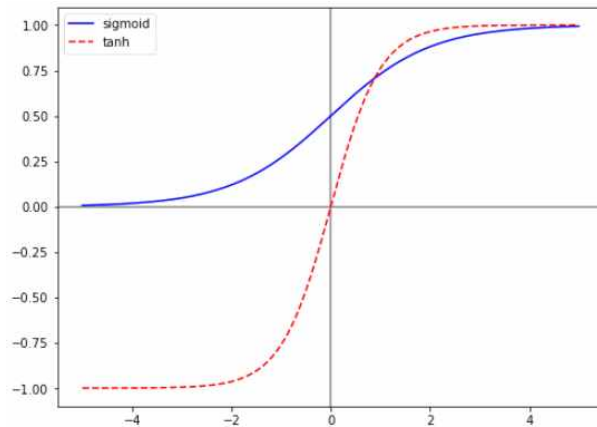
plt.figure(figsize=(8, 6))

plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.legend()
plt.show()

```



tanh



가. 딥러닝 모델

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=6, activation='tanh', input_shape=(1,)),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1), loss='mse')
model.summary()
```

나. 학습

```
[10] 1 # 4.8 딥러닝 네트워크의 학습
      2 model.fit(X, Y, epochs=10)

Epoch 1/10
1/1 [=====] - 0s 1ms/step - loss: 256.3840
Epoch 2/10
1/1 [=====] - 0s 824us/step - loss: 116.0593
```

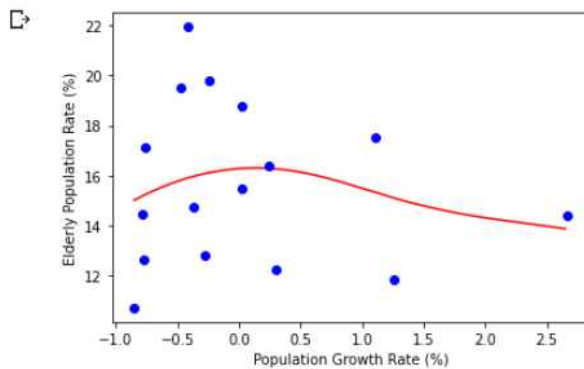
다. 예측

```
[11] 1 # 4.9 딥러닝 네트워크의 Y값 예측
      2 model.predict(X)
```

```
array([[16.27147 ],
       [15.190466],
       [15.10627 ],
       [16.290047],
       [15.327049],
       [16.291916],
       [16.117289],
       [15.818182],
       [15.214785],
       [15.965492],
       [15.012625],
       [15.909767],
       [16.086227],
       [16.287073],
       [15.238832],
       [13.875053]], dtype=float32)
```

라. 결과 시각화

```
[12] 1 # 4.10 딥러닝 네트워크의 회귀선 확인
      2 import matplotlib.pyplot as plt
      3
      4 line_x = np.arange(min(X), max(X), 0.01)
      5 line_y = model.predict(line_x)
      6
      7 plt.plot(line_x, line_y, 'r-')
      8 plt.plot(X, Y, 'bo')
      9
     10 plt.xlabel('Population Growth Rate (%)')
     11 plt.ylabel('Elderly Population Rate (%)')
     12 plt.show()
```

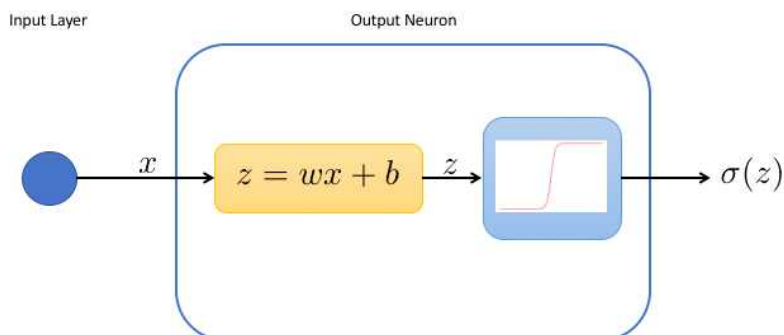


23. 이항분류 및 다항 분류

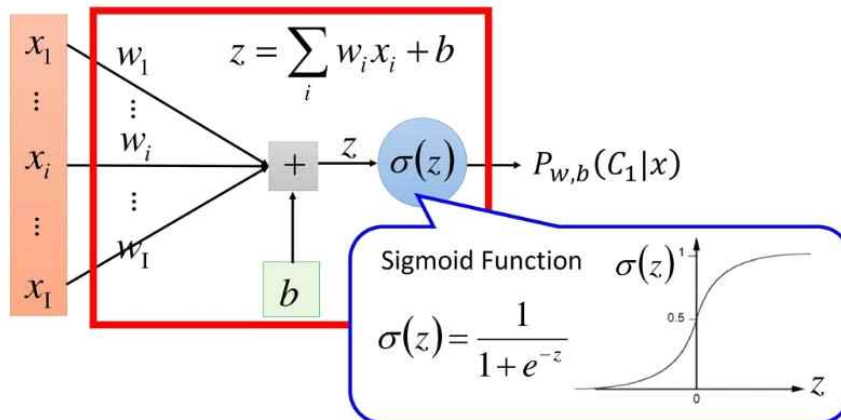
가. 이진 분류

- 1) 두가지로 분류하는 방법
- 2) 4개의 결과

나. 이진 분류 개념



다. 이진 분류 활성화 함수



라. 시그모이드 함수

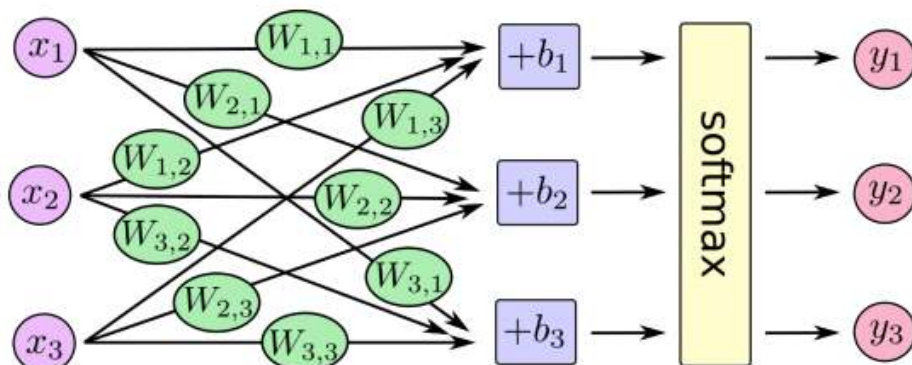
1) 이진분류 모델의 출력층에 주로 사용되는 활성화 함수

- 0과1사이의 값으로 출력
- 특정 임계값 이상이면 양성 이하면 음성

마. 다중 분류

1) 소프트맥스 함수

- 분류의 마지막 활성화 함수로 사용
- 모든 y_i 의 합은 1



- 뉴런의 결과를 e의 지수승으로 하여 모든 합으로 나눈 결과

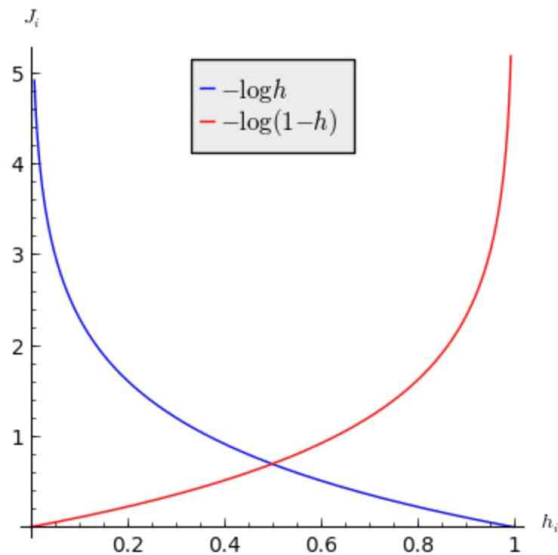
바. 대표적 다중분류

1) MNIST 손글씨

사. 분류에서의 손실 함수

1) 크로스 엔트로피

- $y=1$ 일 때: 파란색
- $y=0$ 일 때: 빨간색



24. 이항분류:레드와인과 화이트와인구분

가. 와인 데이터 셋

```
import pandas as pd
red = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-red.csv', sep=';')
white = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-
databases/wine-quality/winequality-white.csv', sep=';')
print(red.head())
print(white.head())
```

나. 와인 데이터셋 합치기

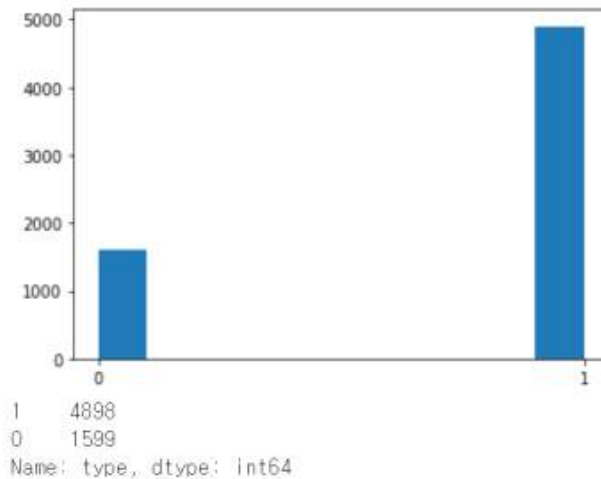
```
red['type'] = 0
white['type'] = 1
print(red.head(2))
print(white.head(2))

wine = pd.concat([red, white])
print(wine.describe())
```

다. 레드와인 화이트와인 수

```
import matplotlib.pyplot as plt
plt.hist(wine['type'])
plt.xticks([0, 1])
plt.show()

print(wine['type'].value_counts())
```



25. 정규화

가. 정규화 이후

1) 최소 0, 최대 1

```
wine_norm = (wine - wine.min()) / (wine.max() - wine.min())
print(wine_norm.head())
print(wine_norm.describe())
```

나. 학습 데이터와 테스트 데이터 분리

- 1) 특징에서 마지막 값을 정답으로
- 2) 정답을 원 핫 인코딩으로

26. 과제

- 가. <https://github.com/ykh9759/2020-2-AI/tree/master/code>

27. 시험기간 연습 코딩

- 가. <https://github.com/ykh9759/2020-2-AI/blob/master/MyTest/MyAlTest.ipynb>

꼬리말

어느새 벌써 포트폴리오가 완성 됐다. 정말 며칠 동안 만들면서 정말 많은 도움이 된거 같다. 특히 어려웠던 부분을 다시 한번 되돌아 봐서 좋았고 시험 때 잘 몰랐던 부분을 다시 확인할 수 있었다. 앞으로도 남은 인공지능 팀 프로젝트도 무사히 마무리하고 올 한해를 잘 마무리 했으면 좋겠다. 이번 학기를 통해 처음 인공지능이라는 분야를 접했지만 정말 쉽지 않은 것 같다. 하지만 언젠가는 도움이 될거라 생각하고 열심히 해야겠다. 마지막 학기라 너무 아쉽지만 남은 한달 힘내보자!