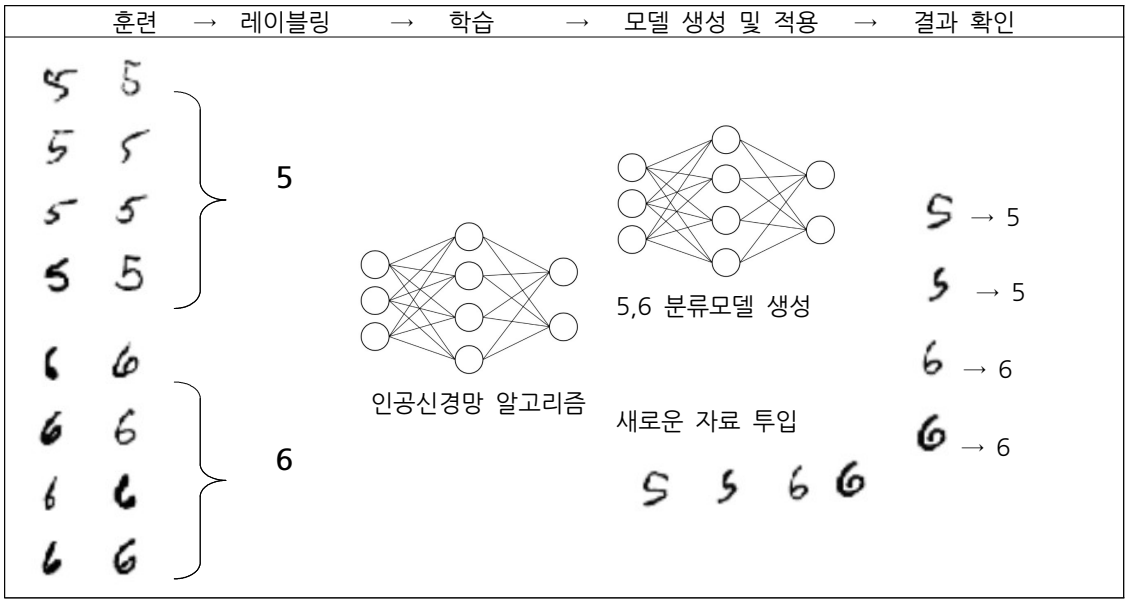
인공지능수학. **Ⅲ. 이미지 자료의 표현과 분류**	고등학교 학번, 이름:	

0. INTRO

사람은 사진 속의 개와 고양이를 쉽게 구별할 수 있다. 이는 많은 경험을 통해 학습한 결과이지만 그 구별 기준을 설명하기는 어렵다. 그러므로 인공지능이 사진 속의 개와 고양이를 구별하는 기준을 사람이 직접 정해주는 것은 매우 힘든 일이다.

딥러닝은 인공신경망을 여러 층으로 연결하여 구현한 것으로, 많은 자료들로부터 스스로 사물을 구별하는 기준을 학습하도록 설계된 인공지능이다. 개와 고양이를 구별하거나, 손글씨를 해석하는 문제, 지문이나 얼굴과 같은 생체 정보를 인식하는 문제는 인공지능을 통해 이미지를 분류하고 판별하는 대표적인 예이다.

다음은 딥러닝을 이용하여 많은 손글씨 자료로부터 5와 6의 특징을 학습하고, 새로운 자료를 인식하여 분류하는 과정을 나타낸 것이다.

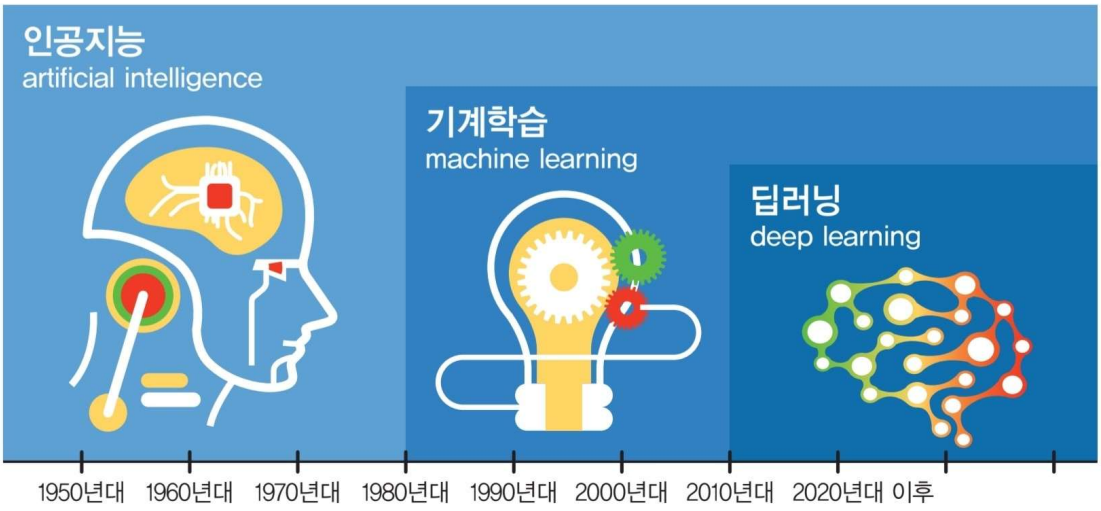


└ 딥러닝을 이용한 손 글씨 인식 과정

0.1 인공지능의 분류

인공지능이란, 말 그대로 인간의 지능을 컴퓨터로 구현해 본 것이라고 할 수 있다. 컴퓨터의 발달과 더불어 과연 인간의 지능을 컴퓨터가 대신 수행할 수 있을지에 대한 도전으로부터 등장한 말이다.

1950년대에 프랑크 로젠블라트의 퍼셉트론의 발표와 더불어 인공지능 시대가 시작되었다. 이를 바탕으로 1980~1990년대에 복잡한 인공신경망에 수학 개념의 본격적 도입으로 기계학습(Machine Learning, ML)이 발전하기 시작하였고 2010년대에 이르러 딥러닝(Deep Learning, DL)이 인공지능을 주도하였다.



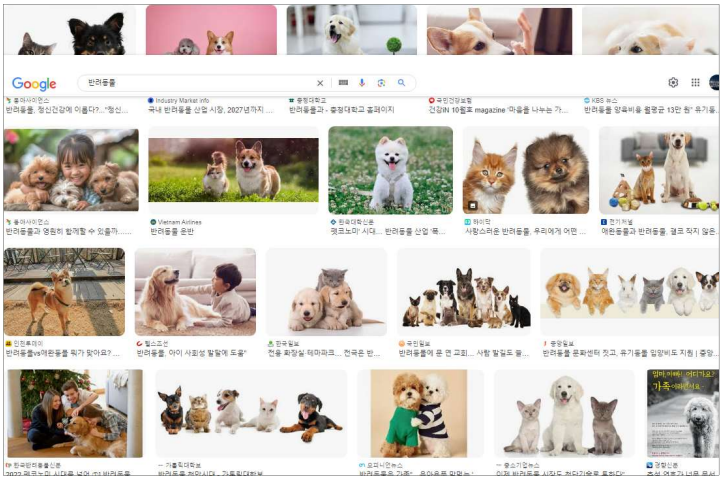
0.2 기계학습(Machine Learning)

기계학습은 경험을 통해 정보처리능력을 향상시키는 컴퓨터 알고리즘의 연구로 인공지능의 한 분야이다. 예를 들어, 기계학습을 통해서 수신한 이메일이 스팸 메일인지, 정상 메일인지를 구분할 수 있도록 훈련할 수 있다. 일반적으로 기계학습은 **머신러닝**이라고도 불리며, 다음과 같이 분류된다.

기계학습 (머신러닝) { 지도학습, 비지도학습, 강화학습

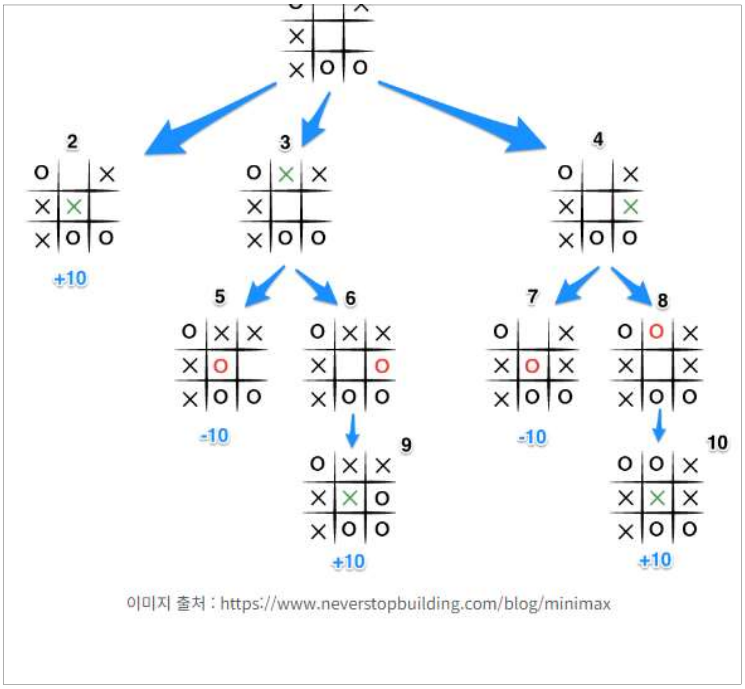
0.2.1 비지도학습

기계학습을 시키기 위해 문제와 정답을 함께 제공하여 학습시키는 지도학습과 달리, 비지도학습에서는 정답을 제공하지 않고 기계가 스스로 추론하게 한다. 대표적으로, 유사도를 이용하여 주어진 데이터를 비슷한 것끼리 분류하는 군집화가 비지도학습에 해당하겠다.



0.2.2 강화학습

강화학습은 현재 상태에서의 최적의 행위를 선택하고 그에 대한 보상이나 벌점을 부여하는 방식의 학습 방법이다. 대표적으로 알파고와 같은 사례가 있다.



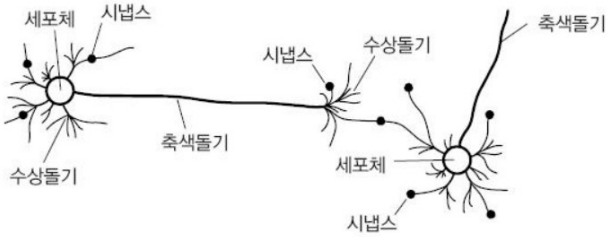
0.1 목차

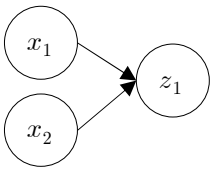
1. 인공신경망의 정보 전달

1.1 신경 세포의 정보 전달

인간의 뇌는 복잡하게 연결된 약 1000억 개의 신경 세포로 구성되어 있다. 신경 세포의 가지 돌기는 다른 신경 세포가 전달하는 자극을 받아들이는데, 전달되는 자극의 세기는 신경 세포 사이의 연결 강도에 따라 조절된다.

신경 세포체는 가지 돌기가 신경 세포로부터 전달받은 자극들을 통합하여 축삭돌기로 전달한다. 이때 축삭 돌기는 절달된 자극의 세기가 **임곗값**보다 큰 경우에만 활성화되어 말단에서 다른 신경 세포에 자극을 전달한다.





1.2 인공신경망의 정보 전달

인간의 신경 세포가 이와 같이 자극을 전달하는 체계를 수학적으로 모델링한 것이 인공신경망이다. 또한 인간의 신경망을 이루는 기본 단위인 신경 세포를 모방한 것이 인공신경망에서 정보를 전달하는 기본 단위인 **퍼셉트론**이다.

실제 뇌	인공신경망
신경세포	퍼셉트론

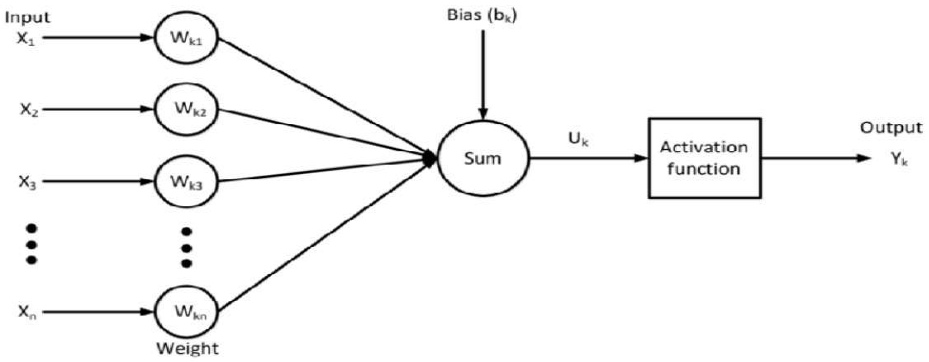
신경 세포와 마찬가지로 퍼셉트론 역시 여러 개의 정보를 동시에 입력받는다. 이 때, 출력에 영향을 주는 입력값의 중요도를 **가중치**를 이용하여 조절한다.

즉, 퍼셉트론으로 입력되는 n개의 값 $x_1, x_2, \cdots, x_n$에 가중치 $w_1, w_2, \cdots, w_n$을 각각 곱하여 더한 값

$$x = x_1w_1 + x_2w_2 + \cdots x_nw_n$$

이 임곗값 θ보다 작으면 퍼셉트론은 0을 출력, 임곗값 θ보다 크거나 같으면 퍼셉트론은 0이 아닌 상수 c를 출력한다.

이를 활성화함수 $\sigma(x) = \begin{cases} 0 & (x < \theta) \\ c & (x \geq \theta) \end{cases}$ 와 같이 나타낼 수 있다.



[문제1] 입력값 x_i와 가중치 w_i, 활성화함수 $\sigma(x)$가 아래와 같은 퍼셉트론의 출력값 c를 구하여라.

〈입력값 x_i〉

x_1	x_2
1	2

〈가중치 w_i〉

w_1	w_2
0.8	0.4

〈활성화함수〉

$$\sigma(x)=\begin{cases}0 & (x<1)\\ 2 & (x\geq 1)\end{cases}$$

1.3 배타적 논리합 연산(XOR)

인공신경망으로 모델링한 인공지능이 인간의 뇌처럼 단순 계산 외에 논리적 사고도 할 수 있는지 확인하기 위해 대표적인 논리 연산 중 하나인 배터직 논리합 연산(XOR)이 퍼셉트론으로 가능한지에 대하여 알아보자.

배타적 논리합이란 아래와 같이 두 입력값 중 반드시 하나만 1인 경우에만 출력값이 1이고 그렇지 않은 경우에는 출력값이 0인 연산을 의미한다.

x_1	x_2	$\sigma(x)$
0	0	0
0	1	1
1	0	1
1	1	0

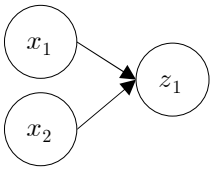
그러나 하나의 퍼셉트론은 배타적 논리합 연산을 수행할 수 없다. 아래 문제를 통해 왜 그런지 알아보자.

[문제2] 입력값이 x_1, x_2이고 가중치가 $w_1=a, w_2=b$, 활성화함수가 $\sigma(x)=\begin{cases}0 & (x<\theta)\\ 1 & (x\geq\theta)\end{cases}$인 퍼셉트론의 출력값 $\sigma(x)$이 위 배타적 논리합의 표와 같도록 하는 임곗값 θ이 존재하지 않음을 설명하시오.

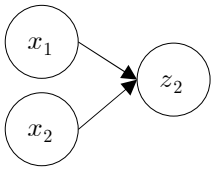
1.4 다층 퍼셉트론 (딥러닝으로!)

한 개의 퍼셉트론으로 구성된 인공신경망은 간단한 배타적 논리합 연산도 수행할 수 없다는 것을 보았다. 그러나 여러 개의 신경 세포가 복잡하게 연결된 인간의 신경망처럼 여러 개의 퍼셉트론을 연결한 인공신경망으로 이 문제를 해결할 수 있다.

이제 입력값이 x_1, x_2이고 출력값이 각각 z_1, z_2인 두 퍼셉트론을 생각해 보자.

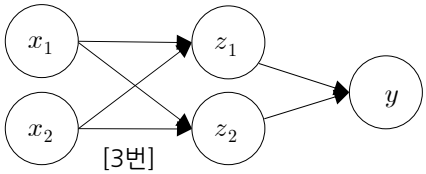


[1번]



[2번]

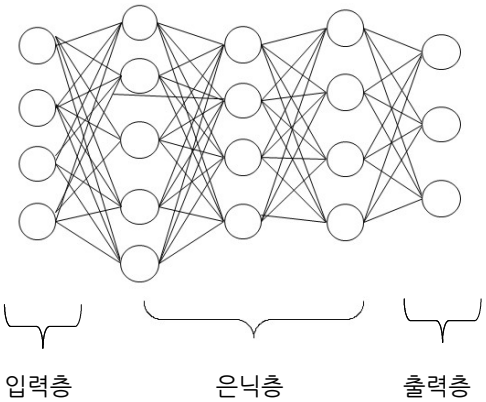
이제 이 두 퍼셉트론 [1번], [2번]을 연결하여 아래와 같은 새로운 인공신경망 [3번]을 만들 수 있다.



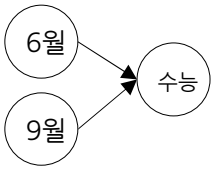
[3번]

이 때 입력값 x_1, x_2가 있는 층을 **입력층**, 출력값 y가 있는 층을 **출력층**, 입력층과 출력층을 제외한 내부의 층을 **은닉층**이라고 한다.

이와 같은 방법으로 입력층와 출력층 사이에 은닉층이 여러 개 있도록 설계할 수 있는데 이를 **다층 퍼셉트론** 또는 **심층 신경망**이라고 한다.



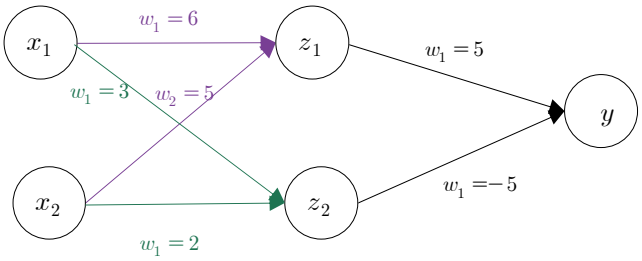
[문제3-1] 아래와 같이 6월, 9월 모의고사 성적으로부터 수능 점수를 예측하기 위한 퍼셉트론이 있다.



여기서 적절한 은닉층을 만들어 다층 퍼셉트론을 자유롭게 만들어 보아라.

[문제3-2] 입력값, 가중치, 활성화함수가 각각 아래와 같은 다층 퍼셉트론의 아래 연산표를 채워 아래 연산이 배타적 논리합 연산 결과임을 보여라.

퍼셉트론



활성화함수

$$\sigma(x) = \begin{cases} 0 & (x < 4) \\ 1 & (x \geq 4) \end{cases}$$

연산표

x_1	x_2	z_1	z_2	y
0	0			
0	1			
1	0			
1	1			

2. 이미지 분류

2.1 행렬을 이용한 이미지 표현

[문제4] 오른쪽과 같이 성분이 0 또는 1인 행렬에서 성분이 1인 칸만 연필로 색칠해 보자. 어떤 숫자와 닮았는가?

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

[그림1]

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

[그림2]

위 문제를 통하여 행렬을 이용해 간단한 이미지를 나타낼 수 있음을 알 수 있다. 이제, 아래 행렬이 나타내는 이미지 [그림3]은 [그림1], [그림2] 중 무엇과 닮았는지 알아보려고 한다.

$$\begin{pmatrix} 0 & \square & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

[그림3]

이 때, 유사한 정도에 따라 분류하는 경우에는 해밍 거리를, 가능성의 정도에 따라 분류하는 경우에는 퍼셉트론을 이용한다.

2.2 해밍거리를 이용한 이미지 분류

개념

두 이미지를 나타내는 행렬이 각각 $A=(a_{ij})_{m\times n}$, $B=(b_{ij})_{m\times n}$

이고, 모든 성분이 0 또는 1일 때, **두 이미지 A, B의 해밍거리** $H(A,B)$를 아래와 같이 정한다.

$$H(A,B) = (\text{두 행렬 } A, B\text{에서 값이 서로 다른 성분의 개수})$$

[문제5] 세 이미지 P, Q, R를 나타내는 행렬이 다음과 같을 때, $H(P,Q)$와 $H(Q,R)$를 구하시오.

$$P=\begin{pmatrix}0&1&0\\1&0&0\\0&1&0\end{pmatrix},\quad Q=\begin{pmatrix}0&0&1\\1&1&0\\1&0&1\end{pmatrix},\quad R=\begin{pmatrix}0&1&0\\0&1&1\\1&1&0\end{pmatrix}$$

두 행렬에서 값이 서로 다른 성분의 개수가 적을수록 두 행렬이 유사하므로, 두 **이미지의 해밍거리가 작을수록 두 이미지가 유사하다고 판단한다.**

[문제6] 2.1절의 [그림3]은 [그림1], [그림2] 중 무엇과 유사하다고 할 수 있는지, 즉, 인공지능은 [그림3]을 무슨 숫자로 인식할지 해밍 거리를 이용해 설명해 보아라.

2.3 퍼셉트론을 이용한 이미지 분류

먼저 행렬의 각 성분을 일렬로 나열하는 법(**평탄화**, flatting)을 배우자. 아래와 같이 각 행렬의 행을 연속적으로 이어붙이면 행렬의 모든 성분을 일렬로 나열할 수 있다.

$$A=\begin{pmatrix}a_1&a_2&\cdots&a_5\\a_6&a_7&\cdots&a_{10}\\\vdots&\vdots&\ddots&\vdots\\a_{31}&a_{32}&\cdots&a_{35}\end{pmatrix}\quad\Rightarrow\quad a_1\ a_2\ \cdots\ a_5\ a_6\ a_7\ \cdots\ a_{10}\ \cdots\ a_{34}\ a_{35}$$

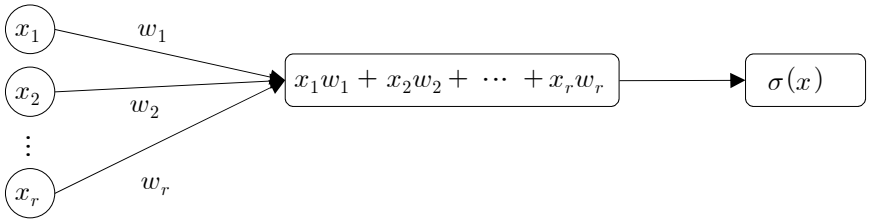
[문제7] 행렬 $A=\begin{pmatrix}1&3&5\\4&2&2\\5&6&1\end{pmatrix}$의 모든 성분을 일렬로 나열하시오.

퍼셉트론에서 입력값 x_i와 가중치 w_i가 아래와 같이 행렬로 제시된다고 하자. 그러면 방금 배운대로 행렬의 성분을 일렬로 나열하는 법을 이용하여 입력값과 가중치의 곱의 합 x를 아래와 같이 구할 수 있다.

개념

퍼셉트론에서 입력값 행렬 $X=(x_{ij})_{m\times n}$와 이에 대한 가중치 행렬 $W=(w_{ij})_{m\times n}$, 활성화 함수 $\sigma(x)$에 대해, 입력값, 가중치의 곱의 합 x를 아래와 같이 구할 수 있다.

$$x = x_1w_1 + x_2w_2 + \cdots + x_{mn}w_{mn}$$



이제, 입력되는 이미지가 [그림1]일 가능성을 계산하는 퍼셉트론을 P_1, [그림2]일 가능성을 계산하는 퍼셉트론을 P_2라 하자. 두 퍼셉트론 P_1, P_2에서 가중치 행렬 W_1, W_2을 아래와 같이 정하자.

$W_1 = 1/$([그림1] 행렬의 모든 성분의 합) $\times$ ([그림1]행렬)
$W_2 = 1/$([그림2] 행렬의 모든 성분의 합) $\times$ ([그림2]행렬)

그러면 가중치 행렬 W_1, W_2는 다음과 같다.

$$W_1 = \frac{1}{19}\times([\text{그림1}]\text{행렬}) = \begin{pmatrix} 0 & \frac{1}{19} & \frac{1}{19} & \frac{1}{19} & 0 \\ \frac{1}{19} & 0 & 0 & 0 & \frac{1}{19} \\ \frac{1}{19} & 0 & 0 & \frac{1}{19} & \frac{1}{19} \\ \frac{1}{19} & 0 & \frac{1}{19} & 0 & \frac{1}{19} \\ \frac{1}{19} & \frac{1}{19} & 0 & 0 & \frac{1}{19} \\ \frac{1}{19} & 0 & 0 & 0 & \frac{1}{19} \\ 0 & \frac{1}{19} & \frac{1}{19} & \frac{1}{19} & 0 \end{pmatrix}, \quad W_2 = \frac{1}{17}\times([\text{그림2}]\text{행렬}) = \begin{pmatrix} 0 & \frac{1}{17} & \frac{1}{17} & \frac{1}{17} & 0 \\ \frac{1}{17} & 0 & 0 & 0 & \frac{1}{17} \\ \frac{1}{17} & 0 & 0 & 0 & \frac{1}{17} \\ 0 & \frac{1}{17} & \frac{1}{17} & \frac{1}{17} & 0 \\ \frac{1}{17} & 0 & 0 & 0 & \frac{1}{17} \\ \frac{1}{17} & 0 & 0 & 0 & \frac{1}{17} \\ 0 & \frac{1}{17} & \frac{1}{17} & \frac{1}{17} & 0 \end{pmatrix}$$

이제 두 퍼셉트론 P_1, P_2를 이용하여 [그림3]이 [그림1], [그림2] 중 어느 것과 더 유사한지 알아보자.

[문제8] [그림3] 행렬을 입력값으로 하였을 때 P_1에서의 (입력값)×(가중치)의 합과 P_2에서의 (입력값)×(가중치)의 합을 각각 구하고 어느 것이 큰지 구하시오.

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

[그림3]

위 [문제]에서 P_1보다 P_2에서의 (입력값)×(가중치) 값이 높다는 결론을 얻었을 것이다. 이에 따라 인공지능은 [그림3]을 [그림1]보다 [그림2]와 더 유사하다고 판단하여 [그림3]을 숫자 8로 분류한다.

[문제9] (열린 문제) [그림3]을 숫자 0이 아니라 8로 분류하는 인공지능 결과에 대해 어떻게 생각하는지 자유롭게 이야기해 보자.

3. 이미지 분류 실습
(손글씨 데이터베이스 딥러닝을 통한, 이미지와 가장 유사한 숫자 인식)

3.1 mnist 데이터베이스 소개

mnist 데이터는 인공지능 연구가 LeCun 교수가 만든 데이터셋으로, 0에서 9까지의 10개 정수를 인간이 손으로 쓴 글씨 이미지 70,000개를 담고 있는 데이터베이스이다. 이 중 60,000개는 훈련용 데이터로, 10,000개는 테스트용 데이터로 이루어져 있으며 인공지능을 처음 학습할 때 대부분 mnist 데이터셋을 통해 실습할 만큼 딥러닝을 쉽게 이해할 수 있는 실습에 널리 쓰인다.

3.2 실습

3.2.1 패키지 임포트

아래와 같이 numpy 패키지를 임포트하자. 나머지 필요한 패키지는 코드 중간중간에 임포트할 것이다.

In
```
import numpy as np
```

3.2.2 데이터 다운로드 함수 정의 (평탄화, 정규화)

mnist 데이터베이스로부터 데이터를 다운로드받아 정규화, 평탄화하는 함수 load_mnist()를 정의하자.
평탄화의 수학적 과정을 관찰할 수 있도록 평탄화과정 출력 코드도 삽입하였다. 한편, 평탄화, 정규화 여부를 뜻하는 각 속성 flatten, normalize의 경우 normalize의 기본값은 False로 하였다. 조금 이따가, 아직 정규화를 거치지 않은 이미지 행렬을 한번 관찰해 볼 필요가 있어 정규화 옵션은 일단 비활성화해 두었다.
(코드 가져오기 : https://cha-record.studio/인공지능수학-3단원-실습코드/)

In

```
def load_mnist(flatten=True, normalize = False) :
    # mnist 모듈 임포트
    from tensorflow.keras.datasets import mnist
    # 내려받은 데이터를 훈련용, 테스트용으로 분리
    (x_train, t_train), (x_test, t_test) = mnist.load_data()

    # 정규화
    if normalize :
        # 각 값을 255로 나누어 0과 1사이 값으로 만듦.
        x_train, x_test = x_train / 255.0 , x_test / 255.0

        x_train = x_train.astype(np.float32)    # x_train을 부동소수점으로 표기
        x_test = x_test.astype(np.float32)  # x_test을 부동소수점으로 표기

    # 평탄화 과정
    if flatten :
        # 평탄화 이전 x_train, x_test의 형태 출력
        print('<평탄화 이전 데이터 개수와 형태>')
        print('평탄화 이전 x_train 형태::', '개수:', x_train.shape[0], '사이즈: ', x_train.shape[1:])
        print('평탄화 이전 x_test 형태::', '개수:', x_test.shape[0], '사이즈: ', x_test.shape[1:])

        # 평탄화(이미지개수는 그대로 두고 사이즈는 한 줄로 바꿈.)
        x_train = x_train.reshape(x_train.shape[0], -1)
        x_test = x_test.reshape(x_test.shape[0], -1)

        # 평탄화 이후 x_train, x_test의 형태 출력
        print('  ')
        print('<평탄화 이후 데이터 개수와 형태>')
        print('평탄화 이후 x_train 형태::', '개수:', x_train.shape[0], '사이즈: ', x_train.shape[1:])
        print('평탄화 이후 x_test 형태::', '개수:', x_test.shape[0], '사이즈: ', x_test.shape[1:])

    return (x_train, t_train), (x_test, t_test)

# load함수를 호출하고 변수에 할당
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
```

출력 결과는 아래와 같다. 실제로 처음에 28x28 사이즈의 2차원 행렬이었던 각각의 데이터가 평탄화 784x1의 일차원 배열로 바뀌었음을 알 수 있다.

Out

```
<평탄화 이전 데이터 개수와 형태>
평탄화 이전 x_train 형태:: 개수: 60000 사이즈:  (28, 28)
평탄화 이전 x_test 형태:: 개수: 10000 사이즈:  (28, 28)

<평탄화 이후 데이터 개수와 형태>
평탄화 이후 x_train 형태:: 개수: 60000 사이즈:  (784,)
평탄화 이후 x_test 형태:: 개수: 10000 사이즈:  (784,)
```

3.2.3 이미지 데이터 행렬 꺼내보기

인공지능 과정을 더 수행하기 전에 먼저 데이터를 직접 관찰해 보고 넘어가도록 하자. 6만개의 훈련용 데이터를 모두 볼 수는 없지만, 그 중 하나를 꺼내어 관찰할 수는 있다. 훈련용 데이터를 모아놓은 리스트 x_train의 첫 번째 성분을 관찰해 보자.

In

```
# 훈련용 데이터 셋의 첫번째 데이터를 이미지행렬로 나타내기

for i, x in enumerate(x_train[0]):
    print(f'{x:3}', end=' ')  # x를 세자리 정수로 표현하고 공백 삽입
    if (i + 1) % 28 == 0: # 28번째 숫자마다 줄바꿈
        print()
```

Out

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   3  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
  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

784개의 정수로 표현된 데이터를 28개씩 끊어서 배열해 본 결과 손글씨 5와 닮았음을 알 수 있다. 실제로 이 데이터는 숫자 5를 뜻하는 이미지로 입력되어 있다.

3.2.4 인공지능 모델 생성

이제 본격적으로 딥러닝 모델을 만들어 보자.

먼저 여기에 필요한 다양한 패키지를 아래와 같이 임포트하자.

In

```
## 패키지 임포트
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt  # 시각화 도구
```

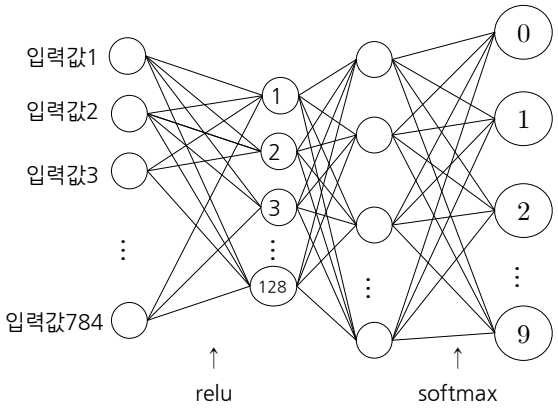
아까 만들어 둔 함수 load_mnist()를 이용해 데이터를 내려 받아, train_images, train_labels, test_images, test_lables와 같은 변수에 할당하자. 이 과정에서 평탄화, 정규화를 모두 거치도록 각 속성 flatten, normalize의 값을 True로 한다. 정규화, 평탄화를 거친 데이터를 관찰해 보고 싶다면 코드 하단의 주석을 해제하여 test_images[0]을 출력해 보아도 무방하다.

In

```
## load_mnist()함수로 데이터 내려받아 훈련용, 테스트용 변수에 할당(정규화, 평탄화 모두 시행)
(train_images, train_labels), (test_images, test_labels) = load_mnist(flatten=True, normalize=
True)

# print(test_images[0])
```

이제 다양한 활성화함수를 이용해 **다층 퍼셉트론**을 만들 것이다. 우리가 만들려 하는 다층 퍼셉트론은 아래와 같이 생겼다. 첫 번째 은닉층은 128개의 뉴런으로, 그리고 relu, softmax등의 활성화함수를 이용한다.



In

```
## 모델 생성 : 다층 퍼셉트론(MLP) 이용

model = models.Sequential([

    # 은닉층1 구성 (뉴런개수:128, 활성화함수:relu)
    layers.Dense(128, activation='relu'),
    # 은닉층2 구성 (몰라도 됨.)
    layers.Dropout(0.2),
    # 출력층 구성 (뉴런의 개수는 10개(숫자 0~9), 활성화함수 softmax)
    # (은닉층을 거쳐 계산된 (이미지와 정수 k간 유사 확률)을 출력층의 각 퍼셉트론 0~9로 출력)
    layers.Dense(10, activation='softmax')

])

# 모델 컴파일 : 손실함수, 옵티마이저, 평가지표 설정 (아직 몰라도 됨)

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

참고로 활성화함수 중 relu함수는 아래와 같은 함수이다. 음수와 같이 의미 없는 제외하고 유의미한 값을 해당 값 그대로 출력해주는 함수라 할 수 있겠다.

$$y = \begin{cases} 0 & (x < 0) \\ x & (x \geq 0) \end{cases}$$

3.2.5 모델 학습 및 평가

이제 모델에 훈련용 데이터를 학습시키고, 성능을 평가하자. 이 과정은 수학 외 과정이므로 다소 이해가 가지 않더라도 적절히 받아들여도 무방하다.

In

```
## 모델 훈련 : 훈련용 데이터를 사용하여 모델을 학습시키기

model.fit(train_images, train_labels, epochs=5)



## 모델 평가 : 테스트 데이터로 모델 성능 평가
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'₩nTest Accuracy : {test_acc * 100:.2f} %')
```

아래와 같이 97%에 임박하는 정확도를 얻었다.

Out

```
Epoch 1/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2962 -
accuracy: 0.9138
Epoch 2/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1448 -
accuracy: 0.9573
Epoch 3/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1058 -
accuracy: 0.9683
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0871 -
accuracy: 0.9733
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0752 -
accuracy: 0.9763
313/313 - 1s - loss: 0.0761 - accuracy: 0.9772 - 629ms/epoch - 2ms/step

Test Accuracy : 97.72 %
```

3.2.6 예측 및 시각화

이제 잘 훈련된 딥러닝 모델을 이용해 실제로 이미지를 판독하여 가장 유사한 숫자를 얻어내자.
우리가 만든 모델 model의 예측함수 predict에 테스트용 이미지 리스트 test_images를 전달값으로 입력하여 각 이미지를 판독하고 그 결과를 predicted_labels라는 리스트에 담으려 한다.
그리고 테스트 이미지 25개를 추출하여 이미지와 그 예측 결과를 함께 출력해 보자.

In

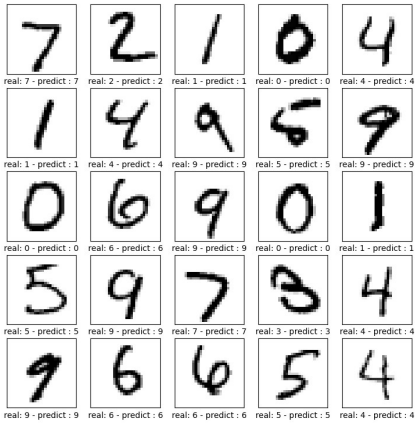
```
## 테스트 이미지에서 예측값 얻기
predictions = model.predict(test_images)
predicted_labels = [tf.argmax(p) for p in predictions]

## 예측 결과 시각화

plt.figure(figsize=(9,9))  # 새로운 그림 생성 및 사이즈를 9*9로 조정
for i in range(25) :    # test_images의 첫 25개 그림에 대해 시행
  plt.subplot(5, 5, i+1)
  plt.xticks([])  # 축 눈금 비활성화
  plt.yticks([])
  plt.grid(False)  # 그리드 비활성화
  plt.imshow(test_images[i].reshape(28,28), cmap = plt.cm.binary) # 이미지 리스트를 이진색상 그림으로 표시
  plt.xlabel(f'real: {test_labels[i]} - predict : {predicted_labels[i]}') # 실제값, 예측값 표시

plt.show()  # 화면에 나타내기
```

Out




```
313/313 [==============================] - 1s 4ms/step
```

첫 번째 이미지의 경우 실제 값은 7, 딥러닝 모델이 판독한 결과도 7이다. 실제로 거의 모든 사람들의 눈에도 이 숫자는 명백히 7이다. 그 외에도 25개의 데이터가 대체로 올바르게 판독되었음을 알 수 있다.