

## 2장 신경망의 수학적 구성 요소

아래 요소들을 직관적으로 살펴본다.

- 텐서(tensor)
- 텐서 연산
- 경사 하강법

## 2.1 신경망 소개

## 케라스로 MNIST 데이터셋 불러오기

- 손글씨 숫자 인식 용도 데이터셋: 70,000개의 샘플 포함
- 레이블(타겟): 0부터 9까지 10개의 범주(category, class)
- 훈련 세트
  - 모델 학습 용도
  - 샘플: 28x28 픽셀 크기의 이미지 60,000개
- 테스트 세트
  - 학습된 모델 성능 테스트 용도
  - 샘플: 28x28 픽셀 크기의 이미지 10,000개



그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)

## 신경망 구조 지정

아래 신경망의 구조는 다음과 같다.

- 층(layer): 2개의 Dense 층 사용. 입력 데이터로부터 표현(representation) 추출. 즉 입력 데이터를 새로운 표현으로 변환.
- 층 연결: Sequential 모델 활용. 완전 연결(fully connected). 조밀(densely)하게 연결되었다고 함.
- 첫째 층
  - 512개의 유닛(unit) 사용. 즉 512개의 특성값으로 이루어진 표현 추출.
  - 활성화 함수(activation function): 렐루(relu) 함수
- 둘째 층
  - 10개의 유닛 사용. 10개의 범주를 대상으로 해당 범부에 속할 확률 계산. 모든 확률의 합은 1.
  - 활성화 함수: 소프트맥스(softmax) 함수

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential([
    layers.Dense(512, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

## 신경망 컴파일

구조가 지정된 신경망을 훈련이 가능한 모델로 만드는 과정이며 아래 세 가지 사항이 지정되어야 한다.

- **옵티마이저(optimizer)**: 모델의 성능을 향상시키는 방향으로 가중치를 업데이트하는 알고리즘
- **손실 함수(loss function)**: 훈련 중 성능 측정 기준
- **모니터링 지표**: 훈련과 테스트 과정을 모니터링 할 때 사용되는 평가 지표(metric). 손실 함수값을 사용할 수도 있고 아닐 수도 있음. 아래 코드에서는 정확도(accuracy)만 사용.

```
model.compile(optimizer="rmsprop",  
              loss="sparse_categorical_crossentropy",  
              metrics=["accuracy"])
```

## 모델 훈련

컴파일된 객체 모델을 훈련한다.

- `fit()` 메서드 호출: 훈련 세트와 레이블을 인자로 사용
- `epochs`: 에포크(전체 훈련 세트 대상 반복 훈련 횟수)
- `batch_size`: 가중치 업데이트 한 번 실행할 때 사용되는 샘플 수

```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

## 모델 활용: 예측하기

훈련에 사용되지 않은 손글씨 숫자 이미지 10장에 대한 모델 예측값을 확인하기 위해 `predict()` 메서드를 이용한다.

```
test_digits = test_images[0:10]  
predictions = model.predict(test_digits)
```



## 성능 테스트

테스트 세트 전체에 대한 성능 평가는 `evaluate()` 메서드를 활용한다. 성능평가에 사용되는 지표는 앞서 모델을 컴파일할 때 지정한 정확도(accuracy)가 사용된다.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"test_acc: {test_acc}")
```

## 2.2 신경망에 사용되는 데이터 표현

- **텐서**: 기본적으로 숫자를 담은 컨테이너(container).
- 앞서 **텐서**(tensor)라고 불리는 넘파이 어레이(NumPy array) 이용
- **행렬**: 2 개의 차원으로 구성된 텐서로 표현
- **텐서의 차원**: 임의로 많을 수 있으며, 텐서에 사용되는 차원을 **축**(axis)이라 부름.
- **랭크**(rank): 차원의 수. 행렬은 rank가 2인 텐서.

## 주의사항

- tensorflow 라이브러리가 제공하는 **Tensor** 자료형: NumPy의 어레이 자료형과 매우 비슷
- 다만 **Tensor** 는 GPU를 활용한 연산을 지원하고 넘파이 어레이는 그렇지 않음.
- 하지만 두 자료형 사이의 형변환이 존재하여 keras 모델 훈련 등 필요할 때 적절한 형변환이 자동으로 이루어짐.

## 데이터 텐서 사례

앞으로 다룰 데이터는 크게 아래 네 종류의 텐서들이다.

- 벡터 데이터: (샘플 수, 특성 수) 모양의 2D 텐서
- 시계열 또는 시퀀스 데이터: (샘플 수, 타임 스텝 수, 특성 수) 모양의 3D 텐서
- 이미지 데이터: (샘플 수, 가로, 세로, 채널 수) 또는 (샘플 수, 채널 수, 가로, 세로) 모양의 4D 텐서
- 동영상 데이터: (샘플 수, 프레임 수, 가로, 세로, 채널 수) 모양의 5D 텐서

## 스칼라(0D 텐서)

- 스칼라(scalar): 숫자 하나로 이루어진 텐서
- 차원이 없다는 의미에서 0D 텐서 부름.
- 예제: 넘파이의 `float32`, `float64` 등

## 벡터 (1D 텐서)

- 각 샘플은 여러 개의 특성값으로 이루어진 벡터
- 데이터 배치는 따라서 2D 텐서, 즉 벡터들의 어레이로 표현됨.
- 0번 축은 샘플 축, 1번 축은 **특성 축**임.
- 예제 1
  - 사람의 나이, 우편 번호, 소득 세 개의 특성으로 구성된 이니구 통계 데이터
  - 10만 명의 통계 데이터를 포함한 데이터셋은 **(100000, 3)** 모양의 2D 텐서임.
- 예제 2
  - 2만개의 단어 각각이 하나의 문서에서 사용된 빈도수로 구성된 데이터는 길이(차원) 2만인 벡터로 표현 가능.
  - 500개의 문서를 대상으로 할 경우 **(500, 20000)** 모양의 2D 텐서로 표현됨.

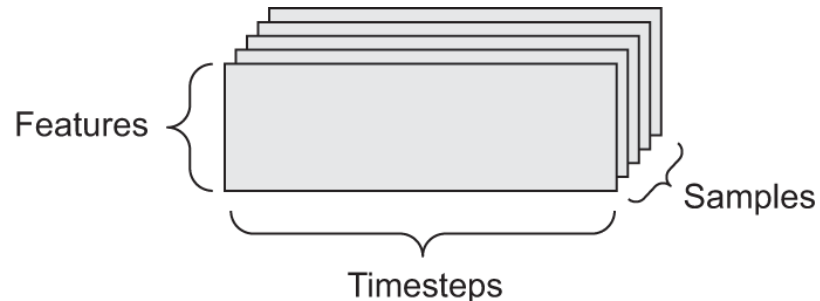
## 행렬(2D 텐서)

- 행렬: 동일한 크기의 벡터로 이루어진 어레이
- 행(row)과 열(column) 두 개의 축 사용
- 예제: 넘파이의 2차원 어레이



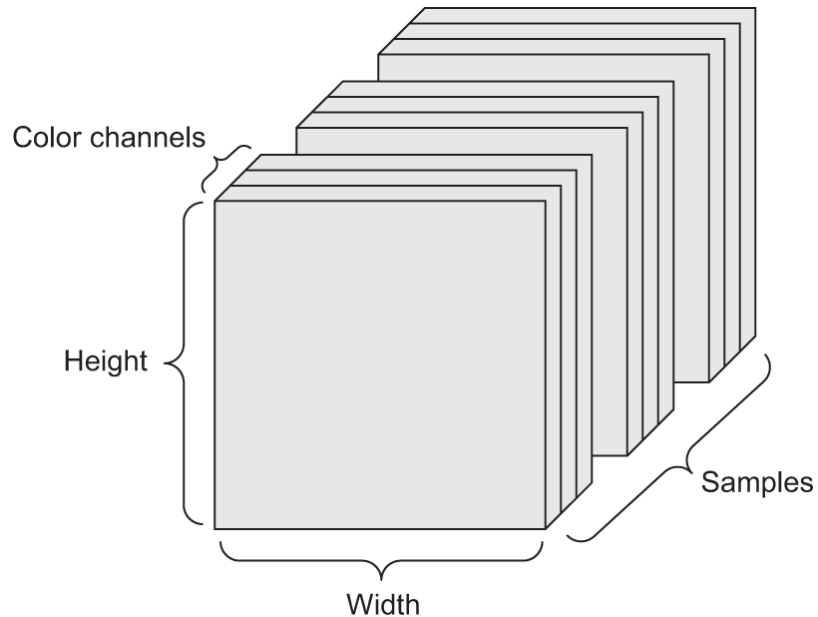
## 시계열 또는 시퀀스 데이터(3D 텐서)

- 증시 데이터, 트윗 데이터 등 시간의 흐름 또는 순서가 중요한 데이터를 다룰 때 사용
- 예제 1(시간의 흐름)
  - 매 분마다 측정된 (현재 증시 가격, 지난 1분 동안의 최고가, 지난 1분 봉안의 최저가) 데이터
  - 하루에 390번 250일 동안 측정해서 얻은 데이터셋은  $(250, 390, 3)$  모양의 3D 텐서.
- 예제 2(순서)
  - 하나의 트윗: 최대 280개의 문자 사용 가능. 문자는 총 128개 중에 하나.
  - 하나의 트윗을  $(280, 128)$  모양의 2D 텐서로 표현 가능. 문자 하나를 0과 1로 구성된 길이가 128인 벡터로 표현. 사용된 문자의 위치에서는 1, 나머지는 0.
  - 백만 개의 트윗은  $(1000000, 280, 128)$  모양의 3D 텐서.



## 이미지 데이터(4D 텐서)

- 이미지는 보통 3D 텐서로 표현: (가로, 세로, 컬러 채널 수)
  - 컬러 이미지의 컬러 채널 수: 3
  - 흑백 이미지의 컬러 채널 수: 1
- 256x256 크기의 흑백 이미지 128개로 구성된 배치: (128, 256, 256, 1) 모양의 4D 텐서
- 256x256 크기의 컬러 이미지 128개로 구성된 배치: (128, 256, 256, 3) 모양의 4D 텐서



## 동영상 데이터(5D 텐서)

- 동영상은 프레임(frame)으로 구성된 순차 데이터. 하나의 프레임은 컬러 이미지. 따랏 3D 텐서로 표현.
- 따라서 하나의 동영상은 (프레임 수, 가로, 세로, 컬러 채널 수) 모양의 4D 텐서로 표현 가능.
- 여러 개의 동영상으로 이루어진 데이터셋: (동영상 수, 프레임 수, 가로, 세로, 컬러 채널 수) 모양의 5D 텐서.
- 예제: 144x256 크기의 프레임으로 구성된 60초 길이의 동영상. 초당 4개의 프레임 구성.
  - 총 240 프레임.
  - 총 10개의 동영상으로 구성된 데이터셋: (10, 240, 144, 256, 3) 모양의 5D 텐서

## 텐서의 주요 속성

- 축 개수(랭크, `ndim`)
  - 텐서에 사용된 축(차원)의 개수
  - 음이 아닌 정수
- 모양(`shape`)
  - 각각의 축에 사용된 (벡터의) 차원 수
  - 정수로 이루어진 튜플
- 자료형(`dtype`)
  - 텐서에 사용된 항목들의 자료형
  - `float16`, `float32`, `float64`, `int8`, `string` 등이 가장 많이 사용됨.
  - 8, 16, 32, 64 등의 숫자는 해당 숫자를 다루는 데 필요한 메모리의 비트(bit) 크기를 가리킴. 즉, 텐서에 사용되는 항목들을 일괄된 크기의 메모리로 처리함.

## 2.3 텐서 연산

신경망 모델의 훈련은 기본적으로 텐서와 관련된 몇 가지 연산으로 이루어진다. 예를 들어 이전 신경망에 사용된 케라스 레이어를 살펴보자.

```
keras.layers.Dense(512, activation="relu")
```

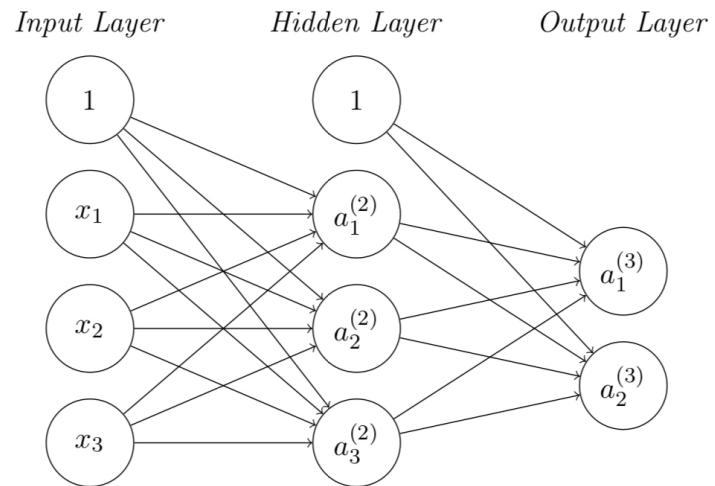
위 층이 하는 일은 데이터셋의 변환이며 실제로 이루어지는 연산은 다음과 같다.

```
output = relu(dot(input, W) + b)
```

- 점곱(`dot(input, W)`): 입력 텐서와 가중치 텐서의 곱
- 덧셈(`dot(input, W) + b`): 점곱의 결과 텐서와 벡터 `b`의 합
- `relu` 함수:  $\text{relu}(x) = \max(x, 0)$

## 항목별 연산

넘파이의 어레이 연산처럼 텐서의 연산도 항목별로 이루어진다. 예를 들어, `relu()` 함수는 각 항목에 적용된다.



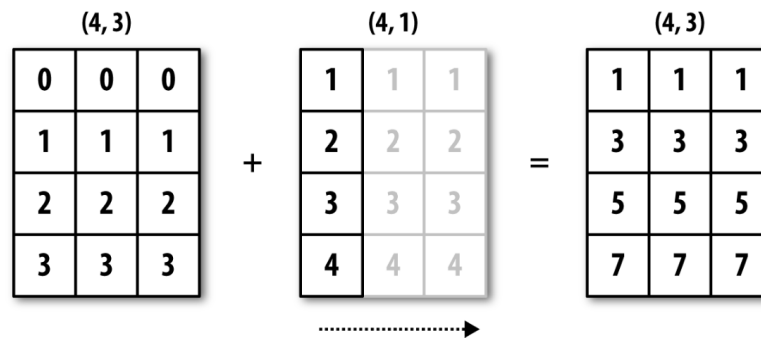
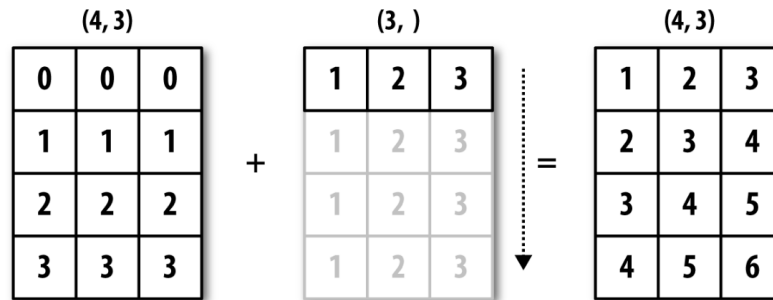
## 브로드캐스팅(Broadcasting)

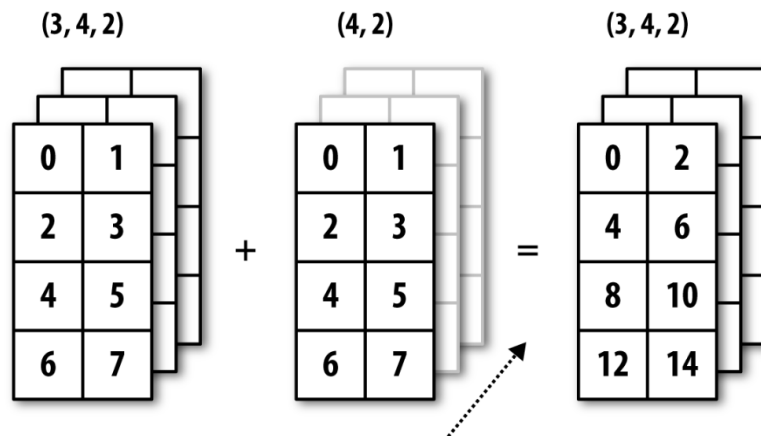
두 텐서의 모양이 아래와 같을 때 항목별 연산을 위한 브로드캐스팅이 가능하다.

$(a, b, \dots, n, n + 1, \dots, m)$  와  $(n, n + 1, \dots, m)$

- 둘째 텐서에 대해 브로드캐스팅 발생
- $a$  부터  $n-1$  까지의 축에 대해 축(axis)이 자동으로 추가됨.







텐서 연산의 기하학적 의미

이동: 벡터 합

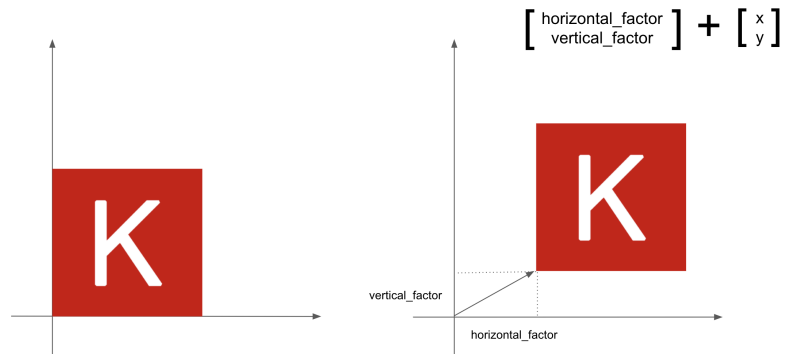


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)

회전: 점곱

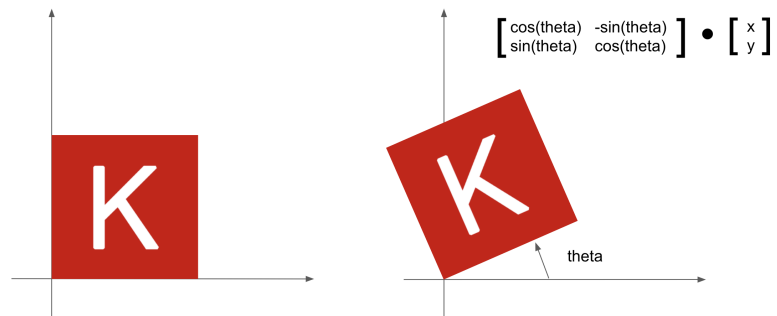


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)

스케일링: 점곱

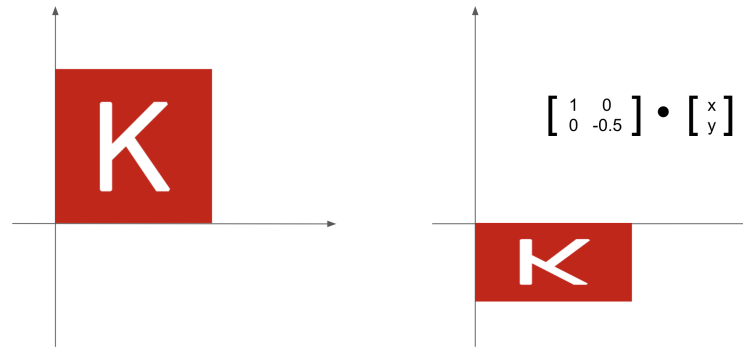


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)

## 아핀 변환

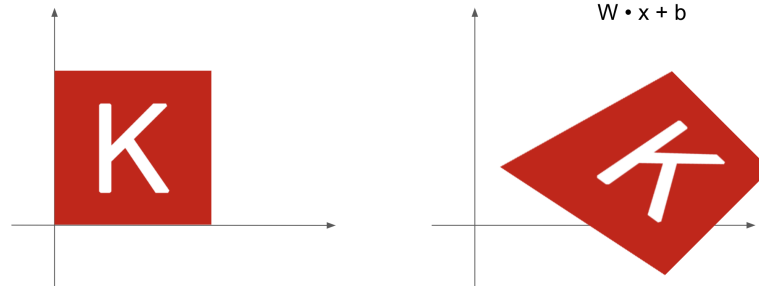


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)

## 아핀 변환과 relu 활성화 함수

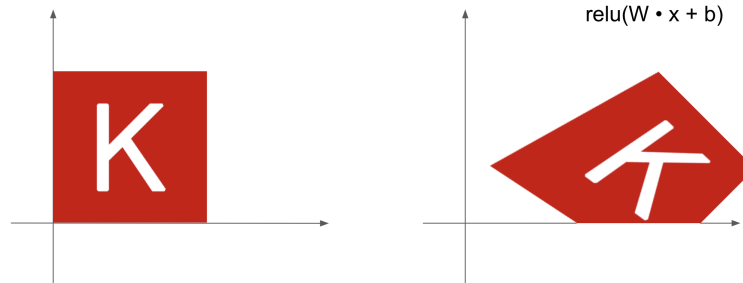


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)



## 딥러닝의 기하학적 의미

- 신경망은 기본적으로 앞서 언급된 텐서 연산의 조합에 불과함.
- 즉, 고차원 공간에서의 매우 복잡한 기하학적 변환을 단순한 텐서 연산의 조합으로 해결함.
- 예제: 3차원 매니폴드
  - 빨간 종이와 파란 종이 두 장을 겹쳐 뭉친 입력값
  - 연속된 종이 펼치기 과정을 통해 명료하게 구분할 수 있는 두 장의 종으로 펼치기
  - 펼치기 과정에 사용된 과정: 손가락으로 조금씩 펼치기
  - 딥러닝 모델의 많은 층(layer)에서 이루어지는 일과 크게 다르지 않음.

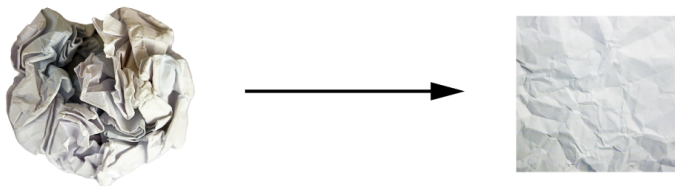


그림 출처: [Deep Learning with Python\(Manning MEAP\)](#)