CNN(Convolutional Neural Network)

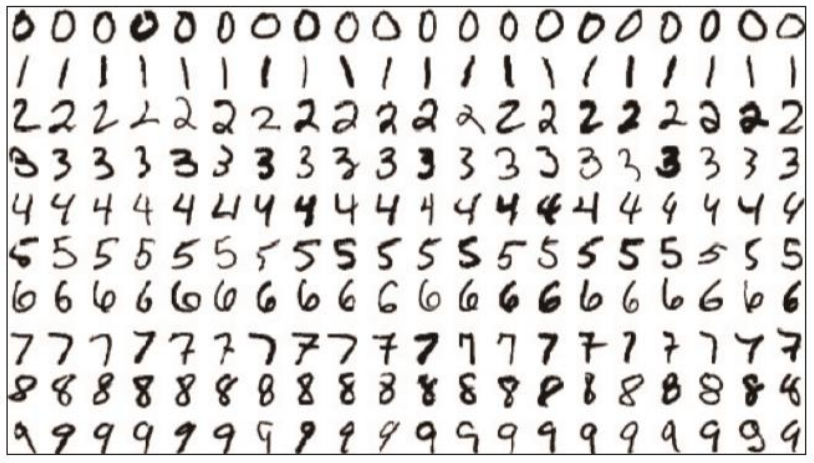
데이터 전처리

- 일반적인 사람에게 이 사진의 숫자를 읽어보라 하면 대부분 '504192'라고 읽을 것
- 그런데 컴퓨터에게 이 글씨를 읽게 하고 이 글씨가 어떤 의미인지를 알게 하는 과정은 쉽지 않음
- 숫자 5는 어떤 특징을 가졌고, 숫자 9는 6과 어떻게 다른지를 기계가 스스로 파악하여 정확하게 읽고 판단하게 만드는 것은 머신러닝의 오랜 진입 과제였음

CNN(Convolutional Neural Network)

데이터 전처리

- MNIST 데이터셋은 미국 국립표준기술원(NIST)이 고등학생과 인구조사 국 직원 등이 쓴 손글씨를 이용해 만든 데이터로 구성되어 있음
- 70,000개의 글자 이미지에 각각 0부터 9까지 이름표를 붙인 데이터셋
- 자신의 알고리즘과 다른 알고리즘의 성과를 비교해 보고자 한 번씩 도 전해 보는 가장 유명한 데이터 중 하나



CNN(Convolutional Neural Network)

데이터 전처리

- MNIST 데이터는 케라스를 이용해 간단히 불러올 수 있음

- mnist.load_ data() 함수로 사용할 데이터를 불러옴

```
from keras.datasets import mnist
```

CNN(Convolutional Neural Network)

데이터 전처리

- 이때 불러온 이미지 데이터를 X로, 이 이미지에 0~9까지 붙인 이름표를 Y_class로 구분하여 명명하겠음
- 또한, 70,000개 중 학습에 사용될 부분은 train으로, 테스트에 사용될 부분은 test라는 이름으로 불러옴
  - 학습에 사용될 부분: X_train, Y_class_train
  - 테스트에 사용될 부분: X_test, Y_class_test

```
(X_train, Y_class_train), (X_test, Y_class_test) = mnist.load_data()
```

CNN(Convolutional Neural Network)

데이터 전처리

- 케라스의 MNIST 데이터는 총 70,000개의 이미지 중 60,000개를 학습용으로, 10,000개를 테스트용으로 미리 구분해 놓고 있음

```
print("학습셋 이미지 수 : %d 개" % (X_train.shape[0]))
print("테스트셋 이미지 수 : %d 개" % (X_test.shape[0]))
```

```
학습셋 이미지 수: 60000 개
테스트셋 이미지 수: 10000 개
```

CNN(Convolutional Neural Network)

데이터 전처리

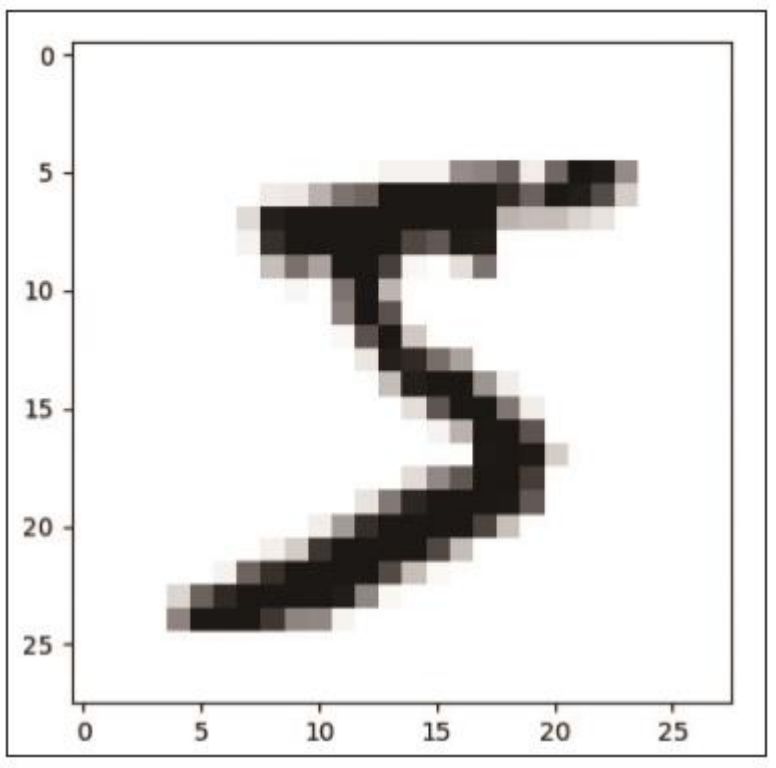
- 불러온 이미지 중 한 개만 다시 불러와 보자
- 이를 위해 먼저 matplotli 라이브러리를 불러옴
- imshow() 함수를 이용해 이미지를 출력할 수 있음
- 모든 이미지가 X_train에 저장되어 있으므로 X_train[0]을 통해 첫 번째 이미지를, cmap = 'Greys' 옵션을 지정해 흑백으로 출력되게 함

```
import matplotlib.pyplot as plt
plt.imshow(X_train[0], cmap='Greys')
plt.show()
```

CNN(Convolutional Neural Network)

데이터 전처리

- MNIST 손글씨 데이터의 첫 번째 이미지



CNN(Convolutional Neural Network)

데이터 전처리

- 이 이미지를 컴퓨터는 어떻게 인식할까?
- 이 이미지는 가로 28 × 세로 28 = 총 784개의 픽셀로 이루어져 있음
- 각 픽셀은 밝기 정도에 따라 0부터 255까지 의 등급을 매김
- 흰색 배경이 0이라면 글씨가 들어간 곳은 1~255까지 숫자 중 하나로 채

  워져 긴 행렬로 이루어진 하나의 집합으로 변환됨

```
for x in X_train[0]:
    for i in x:
        sys.stdout.write('%d₩t' % i)
    sys.stdout.write('₩n')
```

CNN(Convolutional Neural Network)

데이터 전처리

- 각 좌표를 숫자로 표현해보기

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	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	46	130	183	253	253	207	2	0	0	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	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

CNN(Convolutional Neural Network)

데이터 전처리

- 이렇게 이미지는 다시 숫자의 집합으로 바뀌어 학습셋으로 사용됨
- 우리가 앞서 배운 여러 예제와 마찬가지로 속성을 담은 데이터를 딥러닝에 집어 넣고 클래스를 예측하는 문제로 전환시키는 것
- 28 × 28 = 784개의 속성을 이용해 0~9까지 10개 클래스 중 하나를 맞히는 문제가 됨

CNN(Convolutional Neural Network)

데이터 전처리

- 주어진 가로 28, 세로 28의 2차원 배열을 784개의 1차원 배열로 바꿔 주어야 함
- 이를 위해 reshape() 함수를 사용
- reshape(총 샘플 수, 1차원 속성의 수) 형식으로 지정
- 총 샘플 수는 앞서 사용한 X_train.shape[0] 을 이용하고, 1차원 속성의 수는 이미 살펴본 대로 784개

```
X_train = X_train.reshape(X_train.shape[0], 784)
```

CNN(Convolutional Neural Network)

데이터 전처리

- 케라스는 데이터를 0에서 1 사이의 값으로 변환한 다음 구동할 때 최적의 성능을 보임
- 따라서 현재 0~255 사이의 값으로 이루어진 값을 0~1 사이의 값으로 바꿔야 함
- 바꾸는 방법은 각 값을 255로 나누는 것
- 이렇게 데이터의 폭이 클 때 적절한 값으로 분산의 정도를 바꾸는 과정을 데이터 정규화(normalization)라고 함

CNN(Convolutional Neural Network)

데이터 전처리

- 현재 주어진 데이터의 값은 0부터 255까지의 정수로, 정규화를 위해 255로 나누어 주려면 먼저 이 값을 실수형으로 바꿔야 함
- 따라서 다음과 같이 astype() 함수를 이용해 실수형으로 바꾼 뒤 255로 나눔

```
X_train = X_train.astype('float64')
X_train = X_train / 255
```

- X_test에도 마찬가지로 이 작업을 적용

```
X_test = X_test.reshape(X_test.shape[0], 784).astype('float64') / 255
```

CNN(Convolutional Neural Network)

데이터 전처리

- 이제 숫자 이미지에 매겨진 이름을 확인
- 우리는 앞서 불러온 숫자 이미지가 5라는 것을 눈으로 보아 짐작할 수 있음
- 실제로 이 숫자의 레이블이 어떤지를 불러오고자 Y_class_train[0]을 다음과 같이 출력

```
print("class : %d " % (Y_class_train[0]))
```

- 그러면 이 숫자의 레이블 값인 5가 출력되는 것을 볼 수 있음

```
class : 5
```

CNN(Convolutional Neural Network)

데이터 전처리

- 딥러닝의 분류 문제를 해결하려면 원-핫 인코딩 방식을 적용해야 함
- 즉, 0~9까지의 정수형 값을 갖는 현재 형태에서 0 또는 1로만 이루어진 벡터로 값을 수정해야 함
- 예를 들어 class가 '3'이라면, [3]을 [0,0,1,0,0,0,0,0,0,0]로 바꿔 주어야 하는 것

CNN(Convolutional Neural Network)

데이터 전처리

- 열어본 이미지의 class는 [5]였음
- 이를 [0,0,0,0,1,0,0,0,0,0] 로 바꿔야 함
- 이를 가능하게 해 주는 함수가 바로 np_utils.to_categorical() 함수
- to_categorical(클래스, 클래스의 개수)의 형식으로 지정

```
Y_train = np_utils.to_categorical(Y_class_train,10)
Y_test = np_utils.to_categorical(Y_class_test,10)
```

CNN(Convolutional Neural Network)

데이터 전처리

- 이제 변환된 값을 출력해 보자

```
print(Y_train[0])
```

- 아래와 같이 원-핫 인코딩이 적용된 것을 확인할 수 있음

```
[ 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.]
```

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

- 불러온 데이터를 실행할 차례
- 총 60,000개의 학습셋과 10,000개의 테스트셋을 불러와 속성 값을 지닌 X, 클래스 값을 지닌 Y로 구분하는 작업을 다시 한번

```
from keras.datasets import mnist

(X_train, Y_train), (X_test, Y_test) = mnist.load_data()

X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255

Y_train = np_utils.to_categorical(Y_train, 10)
Y_test = np_utils.to_categorical(Y_test, 10)
```

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

- 딥러닝을 실행하고자 프레임을 설정
- 총 784개의 속성이 있고 10개의 클래스가 있음
- 따라서 다음과 같이 딥러닝 프레임을 만들 수 있음

```
model = Sequential()
model.add(Dense(512, input_dim=784, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

- 입력 값(input_dim)이 784개, 은닉층이 512개 그리고 출력이 10개인 모델
- 활성화 함수로 은닉층에서는 relu를, 출력층에서는 softmax를 사용했음
- 딥러닝 실행 환경을 위해 오차 함수로 categorical_crossentropy, 최적화 함수로 adam을 사용

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

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

- 모델의 실행에 앞서 모델의 성과를 저장하고 모델의 최적화 단계에서 학습을 자동 중단하게끔 설정

- 10회 이상 모델의 성과 향상이 없으면 자동으로 학습을 중단

```
import os
from keras.callbacks import ModelCheckpoint,EarlyStopping

MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)
```

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

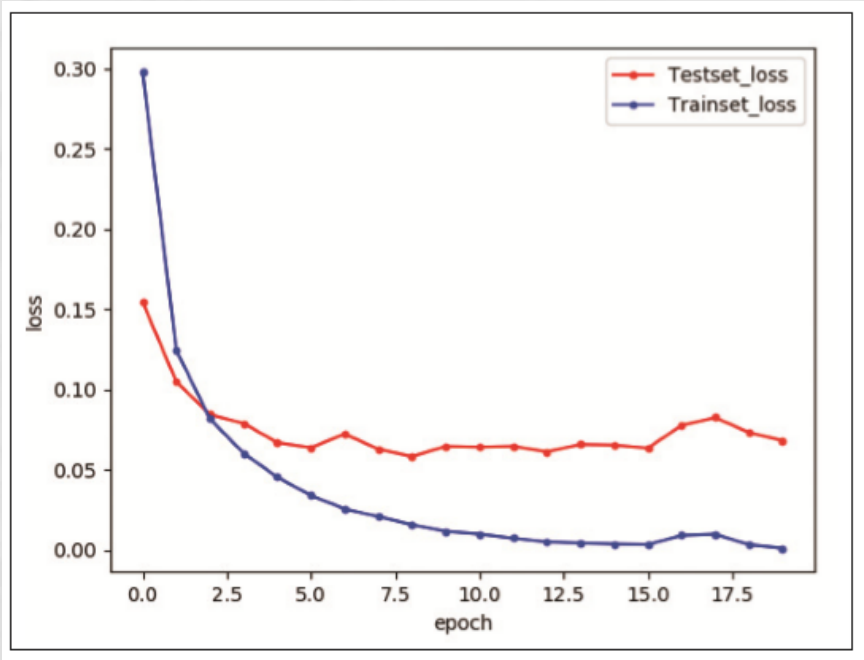
- 실행 결과를 그래프로 표현

```
import matplotlib.pyplot as plt
y_vloss = history.history['val_loss']

# 학습셋의 오차
y_loss = history.history['loss']

# 그래프로 표현
x_len = numpy.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

CNN(Convolutional Neural Network)

딥러닝 기본 프레임 만들기

- 20번째 실행에서 멈춘 것을 확인할 수 있음
- 베스트 모델은 10번째 에포크일 때이며, 이 모델의 테스트셋에 대한 정확도는 98.21%



CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

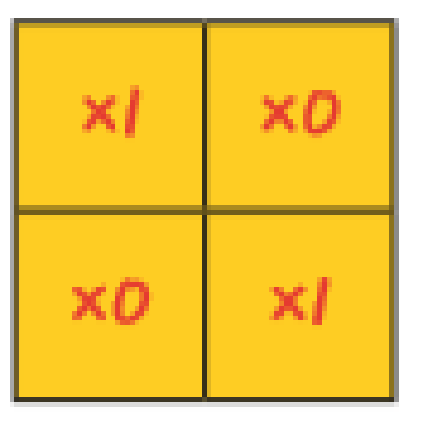
- 컨볼루션 신경망은 입력된 이미지에서 다시 한번 특징을 추출하기 위해 마스크 (필터, 윈도 또는 커널이라고도 함)를 도입하는 기법
- 예를 들어, 입력된 이미지가 다음과 같은 값을 가지고 있다고 함

1	0	1	0
0	1	1	0
0	0	1	1
0	0	1	0

CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

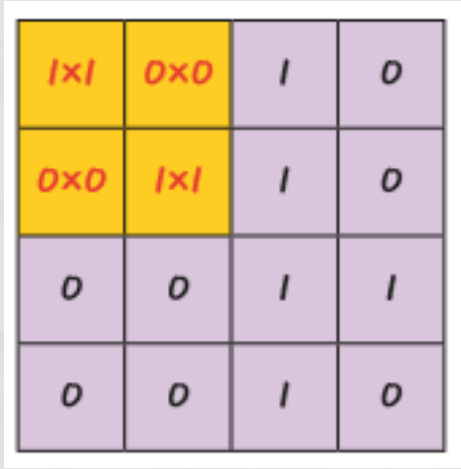
- 여기에 2×2 마스크를 준비함
- 각 칸에는 가중치가 들어있음
- 샘플 가중 치를 다음과 같이 ×1, ×0라고 함



CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

- 마스크를 맨 왼쪽 위칸에 적용시켜 봄



- 적용된 부분은 원래 있던 값에 가중치의 값을 곱해 줌

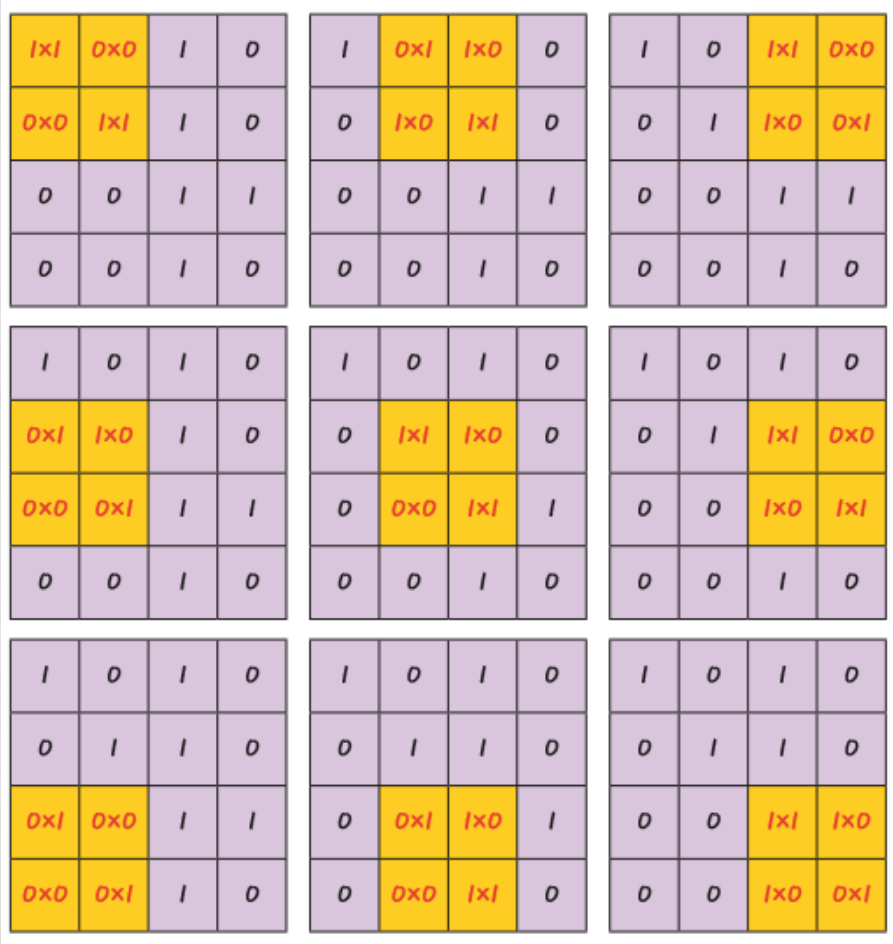
- 그 결과를 합하면 새로 추출된 값은 2가 됨

$$(1\times1)+(0\times0)+(0\times0)+(1\times1)=2$$

CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

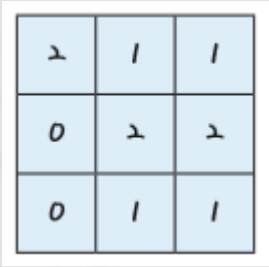
- 이 마스크를 한 칸씩 옮겨 모두 적용



CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

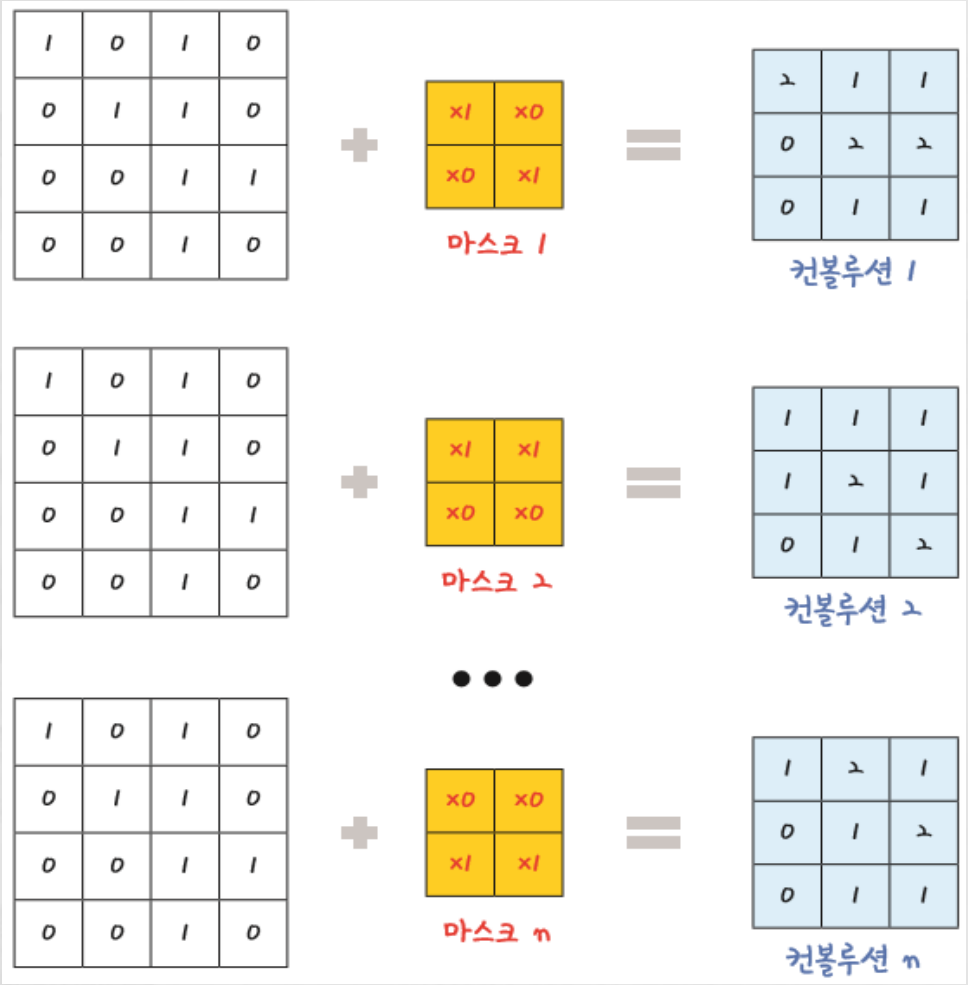
- 그 결과를 정리하면 다음과 같음



- 이렇게 해서 새롭게 만들어진 층을 컨볼루션(합성곱)이라고 부름
- 컨볼루션을 만들면 입력 데이터로부터 더욱 정교한 특징을 추출할 수 있음
- 이러한 마스크를 여러 개 만들 경우 여러 개의 컨볼루션이 만들어짐

CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)



CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

- 케라스에서 컨볼루션 층을 추가하는 함수는 Conv2D()
- 컨볼루션 층을 적용하여 MNIST 손글씨 인식률을 높임

```
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=(28, 28, 1), activation='relu'))
```

- 첫 번째 인자: 마스크를 몇 개 적용할지 정한다. 여기서는 32개의 마스크를 적용함
- kernel_size: 마스크(커널)의 크기를 정한다. kernel_size=(행, 열) 형식으로 정하며, 여기서는 3×3 크기의 마스크를 사용하게끔 정한다
- input_shape: Dense 층과 마찬가지로 맨 처음 층에는 입력되는 값을 알려 주어야 함 input_shape=(행, 열, 색상 또는 흑백) 형식으로 정한다. 만약 입력 이미지가 색상이면 3, 흑백이면 1을 지정.
- activation: 활성화 함수를 정의

CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

- 컨볼루션 층을 하나 더 추가

- 마스크 64개를 적용한 새로운 컨볼루션 층을 추가할 수 있음

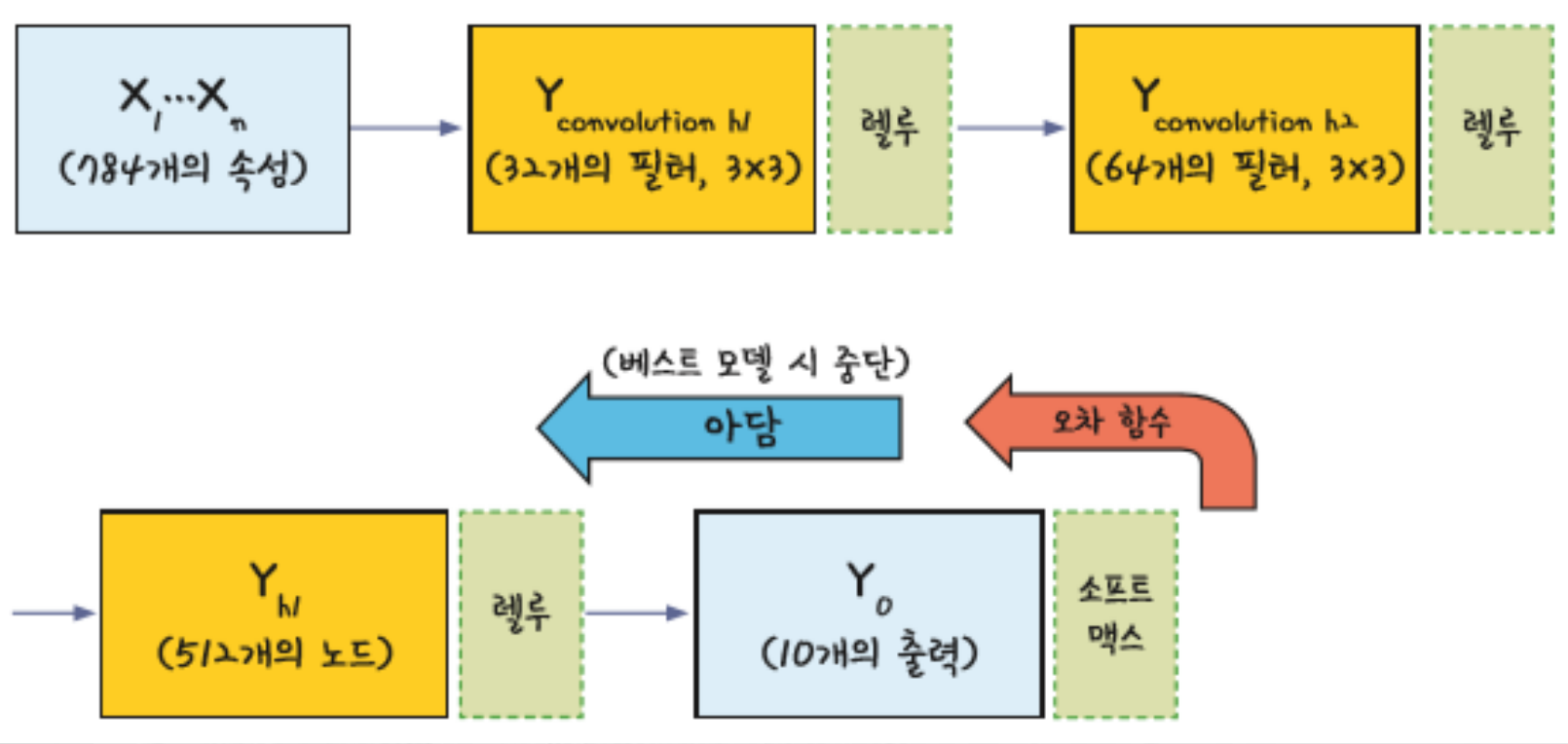
```
model.add(Conv2D(64, (3, 3), activation='relu'))
```

CNN(Convolutional Neural Network)

컨볼루션 신경망(CNN)

- 컨볼루션 층을 추가한 도식화

- 컨볼루션 층의 적용



CNN(Convolutional Neural Network)

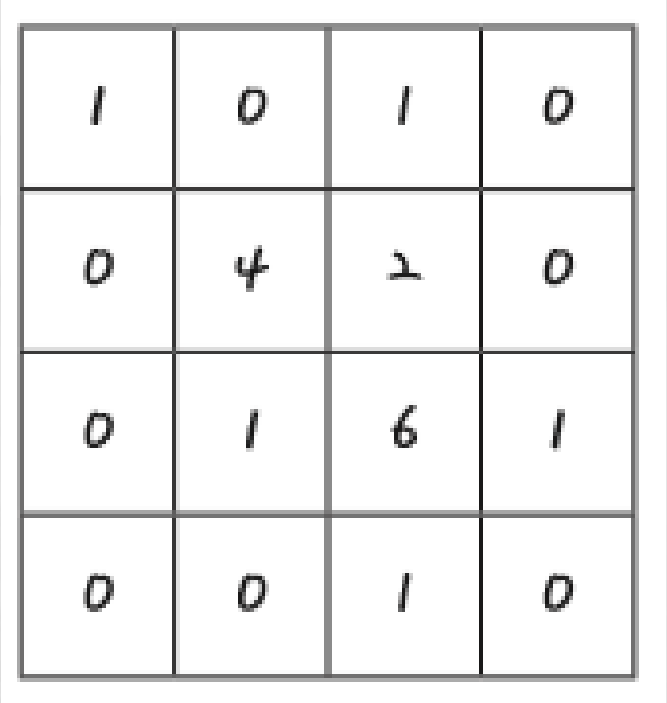
맥스 풀링

- 맥스 풀링 층을 추가
- 앞서 컨볼루션 층을 통해 이미지 특징을 도출하였음
- 하지만 그 결과가 여전히 크고 복잡하면 이를 다시 한번 축소해야 함
- 이 과정을 풀링(pooling) 또는 서브 샘플링(sub sampling)이라고 함

CNN(Convolutional Neural Network)

맥스 풀링

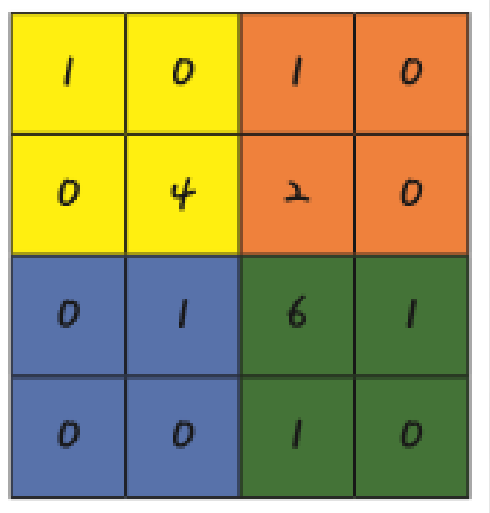
- 풀링 기법 중 가장 많이 사용되는 방법이 맥스 풀링(max pooling)
- 맥스 풀링은 정해진 구역 안에서 가장 큰 값만 다음 층으로 넘기고 나머지는 버림



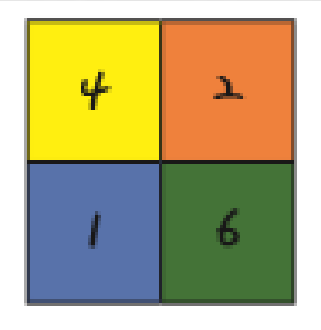
CNN(Convolutional Neural Network)

맥스 풀링

- 맥스 풀링을 적용하면 다음과 같이 구역을 나눔



- 그리고 각 구역에서 가장 큰 값을 추출



CNN(Convolutional Neural Network)

맥스 풀링

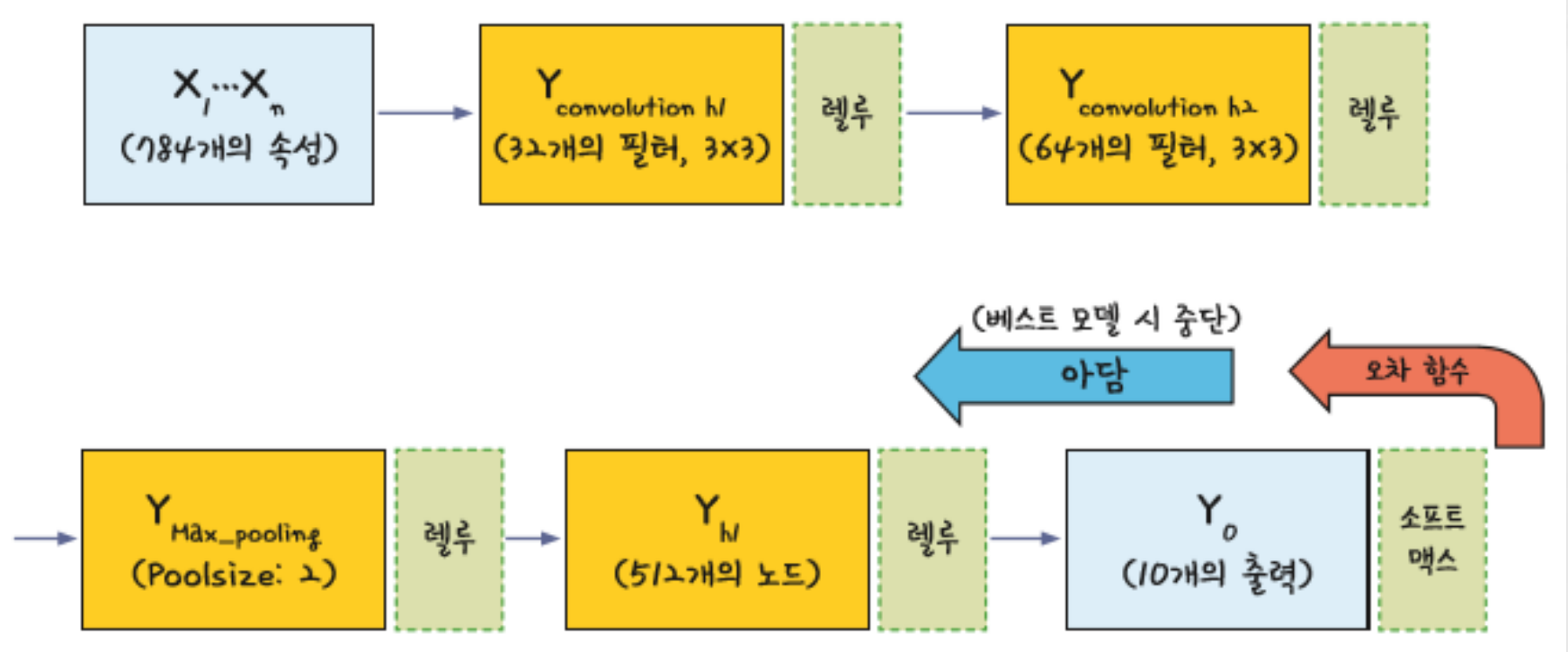
- 이 과정을 거쳐 불필요한 정보를 간추림
- 맥스 풀링은 MaxPooling2D() 함수를 사용해서 다음과 같이 적용할 수 있음

```
model.add(MaxPooling2D(pool_size=2))
```

CNN(Convolutional Neural Network)

맥스 풀링

- pool_size는 풀링 창의 크기를 정하는 것으로, 2로 정하면 전체 크기가 절반으로 줄어듦



CNN(Convolutional Neural Network)

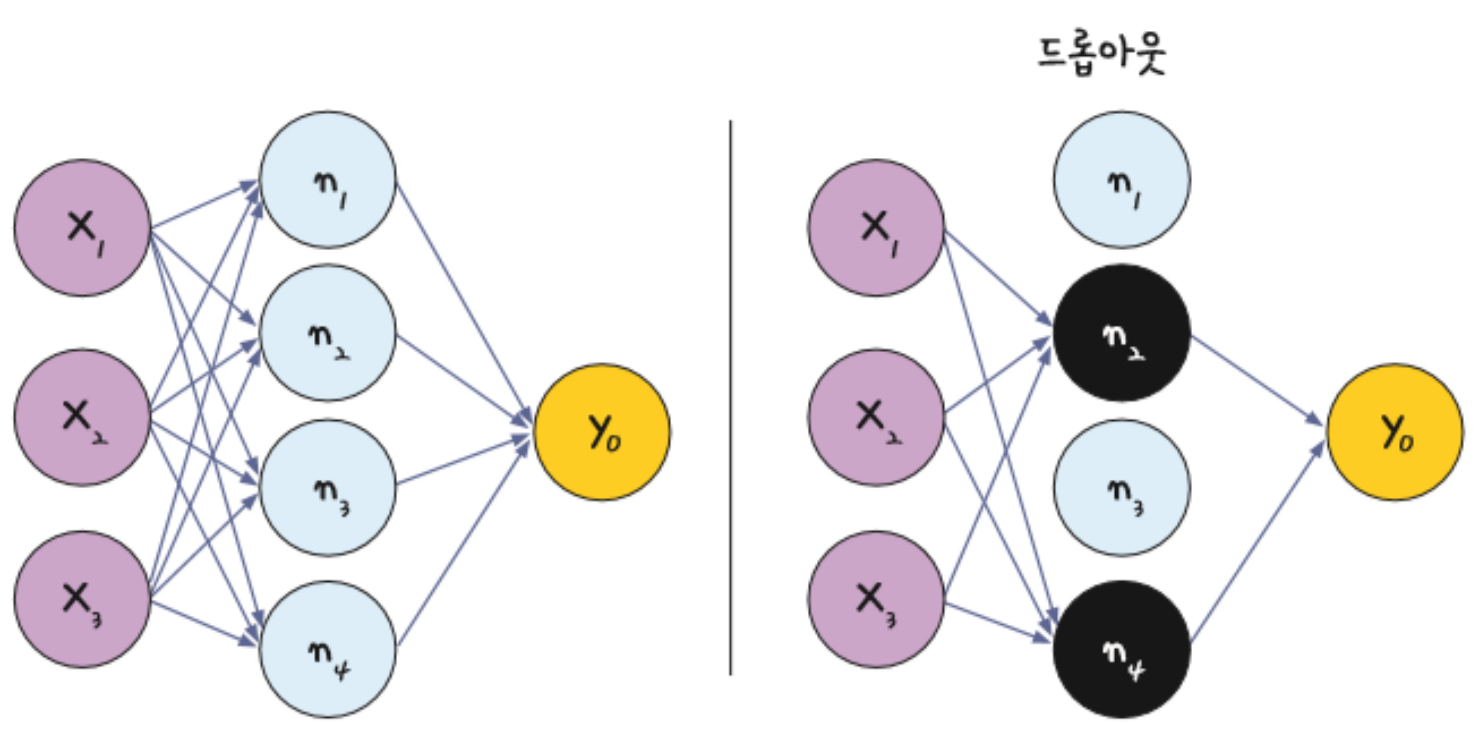
맥스 풀링

- 드롭아웃(drop out) , 플래튼(flatten)
- 노드가 많아지거나 층이 많아진다고 해서 학습이 무조건 좋아지는 것이 아니고 과적합이 발생할 수 있음
- 과접합을 피하는 간단하지만 효과가 큰 기법이 바로 드롭아웃(drop out) 기법
- 드롭아웃은 은닉층에 배치된 노드 중 일부를 임의로 꺼주는 것

CNN(Convolutional Neural Network)

맥스 풀링

- 드롭아웃의 개요
- 검은색으로 표시된 노드는 계산하지 않음



CNN(Convolutional Neural Network)

맥스 풀링

- 이렇게 랜덤하게 노드를 끔으로써 학습 데이터에 지나치게 치우쳐서 학습되는 과적합을 방지할 수 있음
- 케라스는 이를 손쉽게 적용하도록 도와줌
- 예를 들어, 25%의 노드를 끄려면 다음과 같이 코드를 작성

```
model.add(Dropout(0.25))
```

CNN(Convolutional Neural Network)

맥스 풀링

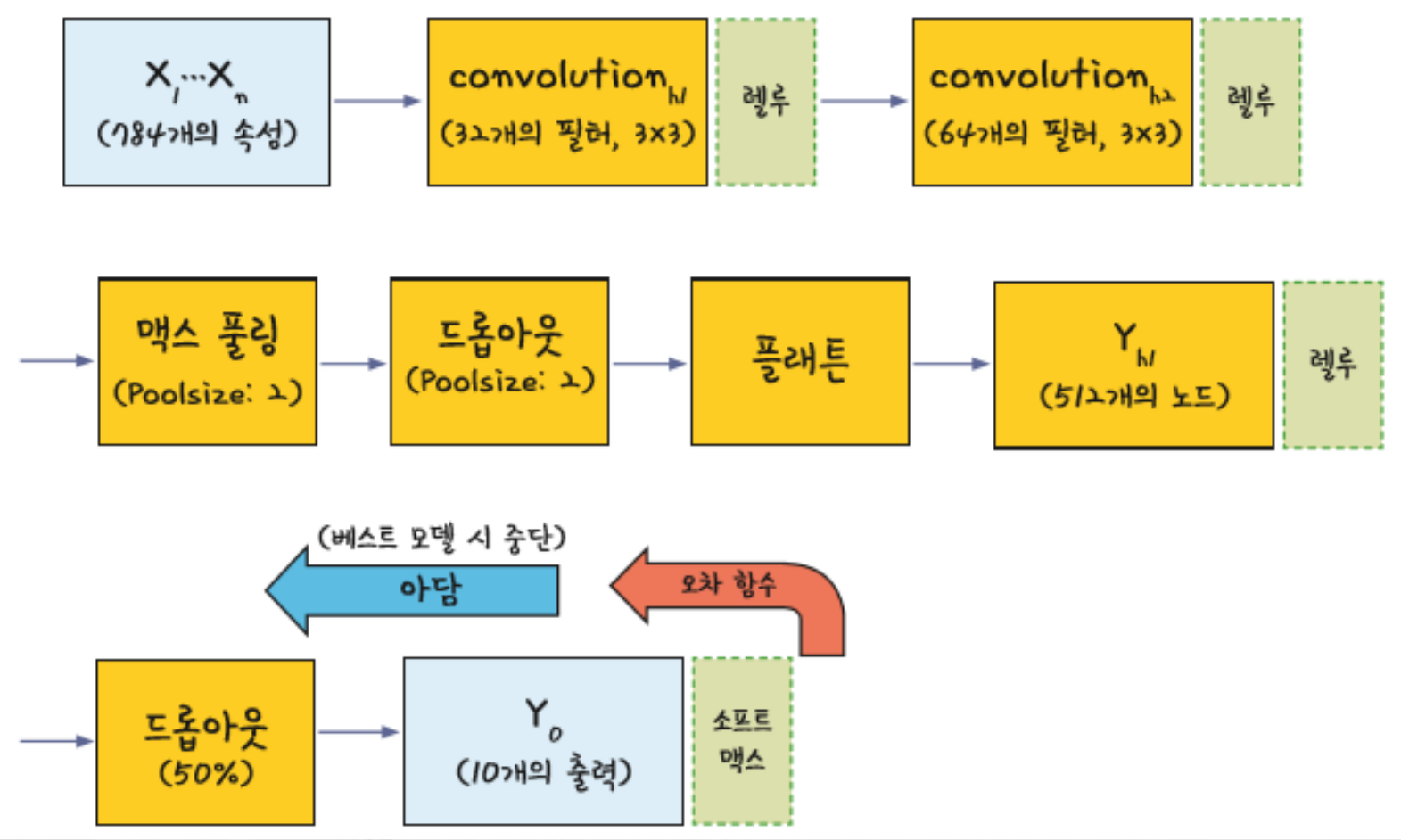
- Dense() 함수를 이용해 만들었던 기본 층에 연결
- 이때 주의할 점은 컨볼루션 층이나 맥스 풀링은 주어진 이미지를 2차원 배열인 채로 다룬다는 점
- 이를 1차원으로 바꿔주는 함수가 플래튼 Flatten() 함수

```
model.add(Flatten())
```

CNN(Convolutional Neural Network)

맥스 풀링

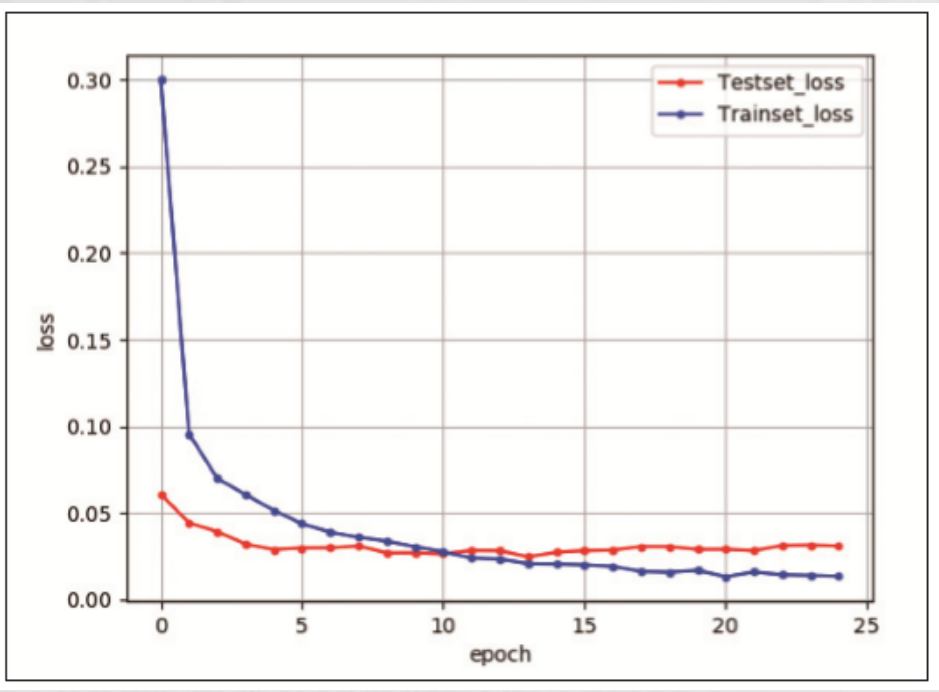
- 드롭아웃과 플래튼 추가



CNN(Convolutional Neural Network)

컨볼루션 신경망 실행하기

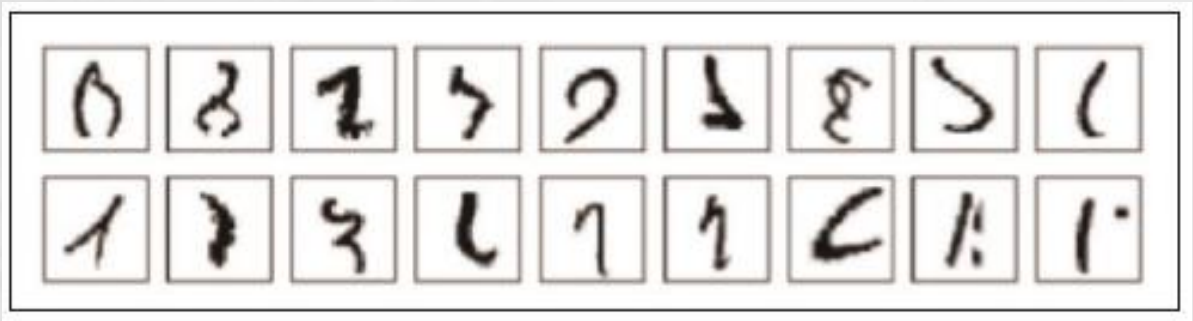
- 12번째 에포크에서 베스트 모델을 만들었고 23번째 에포크에서 학습이 자동 중단됨
- 테스트 정확도가 99.28%로 향상됨



CNN(Convolutional Neural Network)

컨볼루션 신경망 실행하기

- 0.9928, 즉 99.28%의 정확도는 10,000개의 테스트 이미지 중 9,928개를 맞추었다는 뜻
- 정확도가 98.21%였으므로 이보다 107개의 정답을 더 맞힌 것
- 100% 다 맞히지 못한 이유는 데이터 안에 다음과 같이 확인할 수 없는 글씨가 들어있었기 때문



RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

- 인공지능이 문장을 듣고 이해한다는 것은 많은 문장을 이미 학습해 놓았다는 것
- 그런데 문장을 학습하는 것은 우리가 지금까지 공부한 내용과는 성질이 조금 다름
  - 문장은 여러 개의 단어로 이루어져 있는데, 그 의미를 전달하려면 각 단어가 정해진 순서대로 입력되어야 하기 때문
- 즉, 여러 데이터가 순서와 관계없이 입력되던 것과는 다르게, 이번에는 과거에 입력된 데이터와 나중에 입력된 데이터 사이의 관계를 고려해야 하는 문제가 생기는 것

RNN(Recurrent Neural Network)

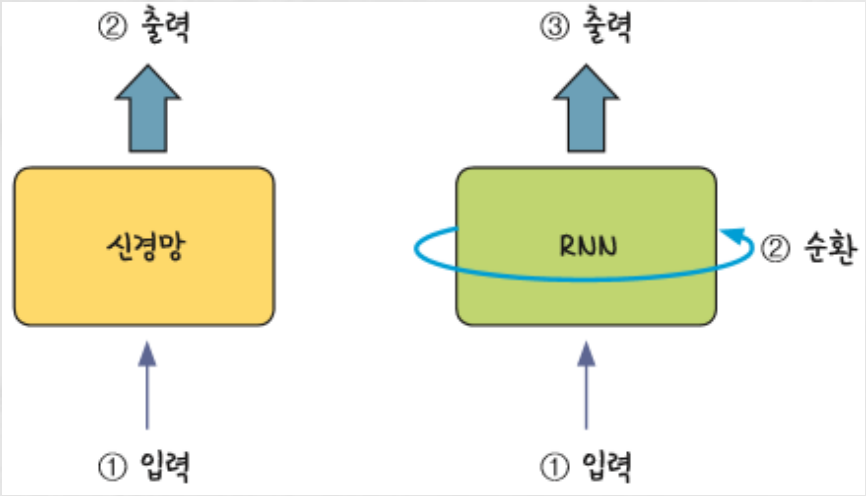
시퀀스 배열로 다루는 순환 신경망

- 이를 해결하기 위해 순환 신경망(Recurrent Neural Network, RNN) 방법이 고안됨
- 순환 신경망은 여러 개의 데이터가 순서대로 입력되었을 때 앞서 입력받은 데이터를 잠시 기억해 놓는 방법
- 그리고 기억된 데이터가 얼마나 중요한지를 판단하여 별도의 가중치를 줘서 다음 데이터로 넘어감
- 모든 입력 값에 이 작업을 순서대로 실행하므로 다음 층으로 넘어가기 전에 같은 층을 맴도는 것처럼 보임
  - 이렇게 같은 층 안에서 맴도는 성질 때문에 순환 신경망이라고 부름

RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

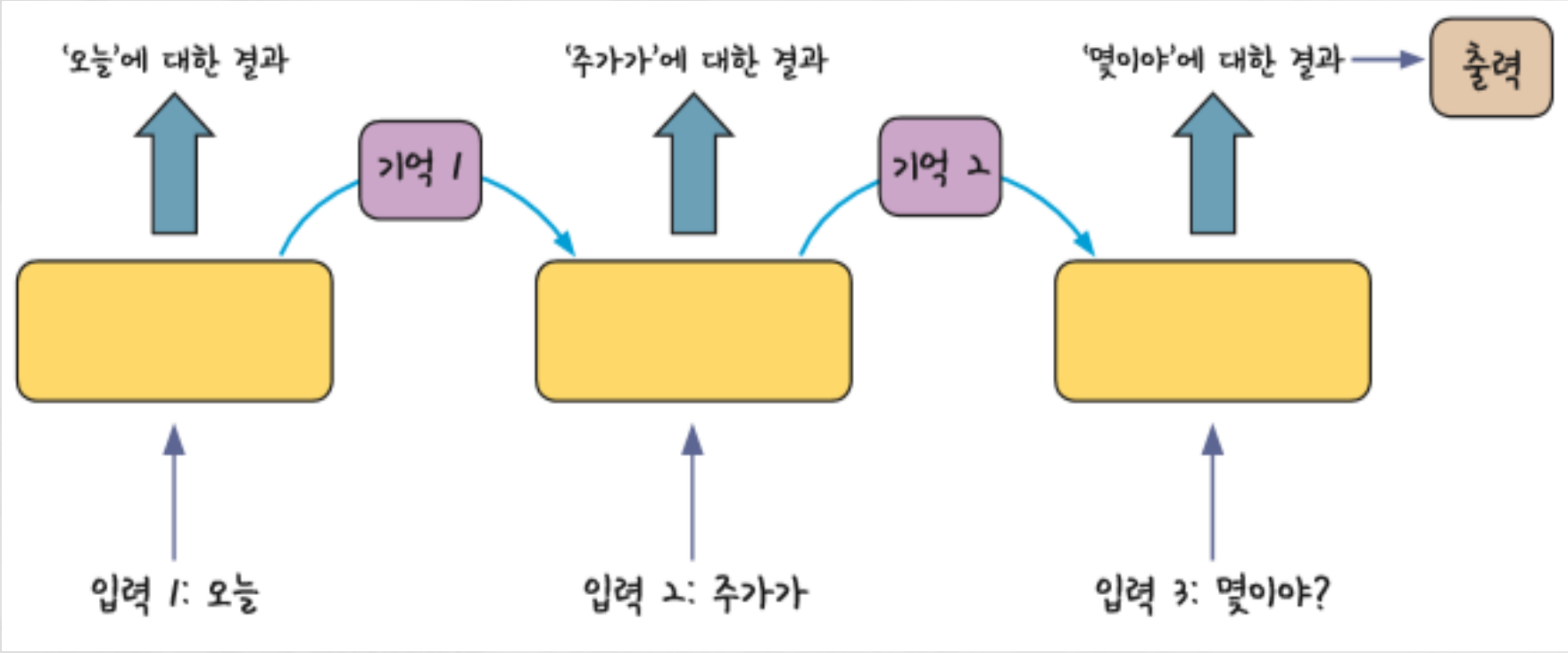
- 예를 들어 인공지능 비서에게 "오늘 주가가 몇이야?"라고 묻는다고 가정하자
- 그러면 오른쪽 그림의 2번에 해당하는 순환 부분에서 단어를 하나 처리할 때마다 단어마다 기억하여 다음 입력 값의 출력을 결정



RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

- 순환이 되는 가운데 앞서 나온 입력에 대한 결과가 뒤에 나오는 입력 값에 영향을 주는 것을 알 수 있음

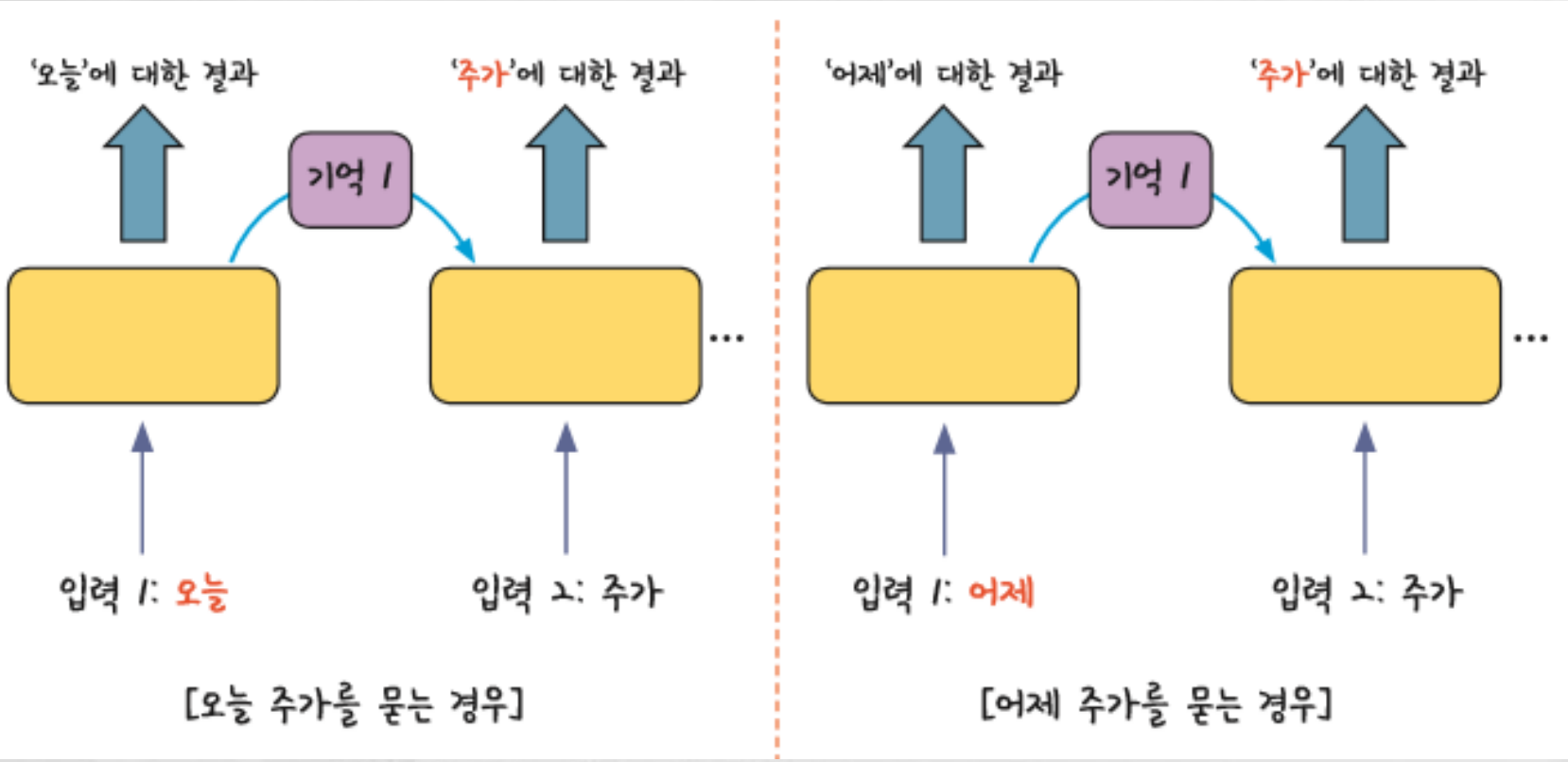


RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

- 2의 값은 양쪽 모두 '주가'이지만, 왼쪽의 주가는

  오늘을 기준으로, 오른쪽은 어제를 기준으로 계산되어야 함



RNN(Recurrent Neural Network)

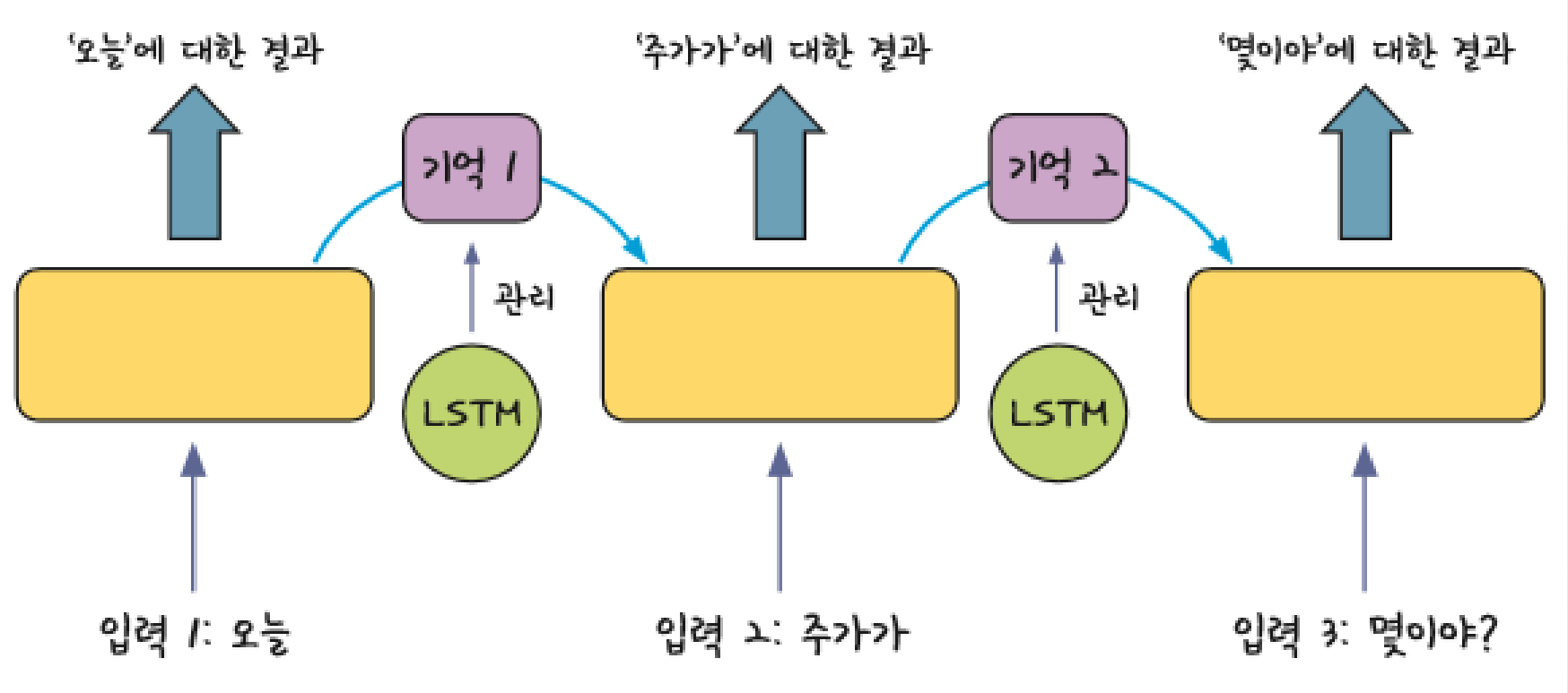
시퀀스 배열로 다루는 순환 신경망

- RNN이 처음 개발된 이후, RNN의 결과를 더욱 개선하기 위한 노력이 계속 되어옴
- 이 중에서 LSTM(Long Short Term Memory) 방법을 함께 사용하는 기법이 현재 가장 널리 사용되고 있음
- LSTM은 한 층 안에서 반복을 많이 해야 하는 RNN의 특성상 일반 신경망보다 기울기 소실 문제가 더 많이 발생하고 이를 해결하기 어렵다는 단점을 보완한 방법
- 즉, 반복되기 직전에 다음 층으로 기억된 값을 넘길지 안 넘길지를 관리하는 단계를 하나 더 추가하는 것

RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

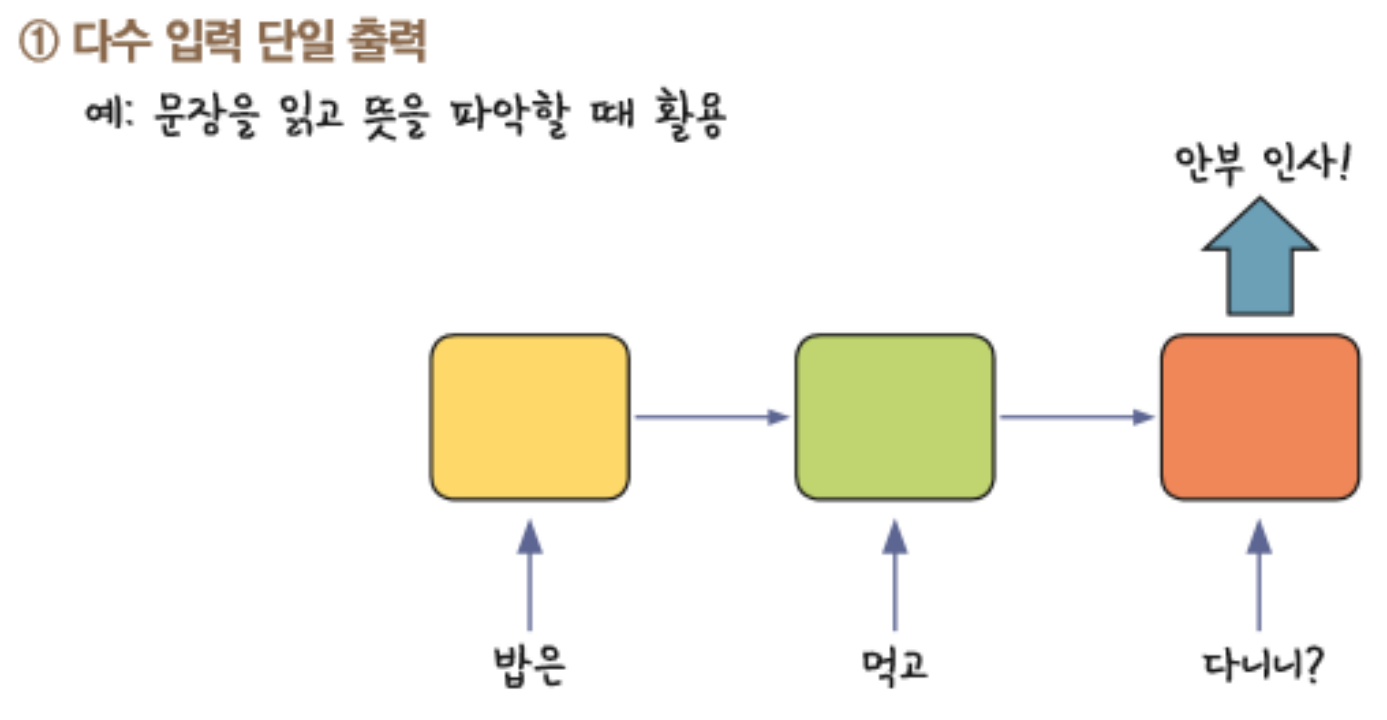
- LSTM은 기억값의 가중치를 관리하는 장치



RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

- RNN 방식의 장점은 입력 값과 출력 값을 어떻게 설정하느냐에 따라 여러 가지 상황에서 이를 적용할 수 있다는 것

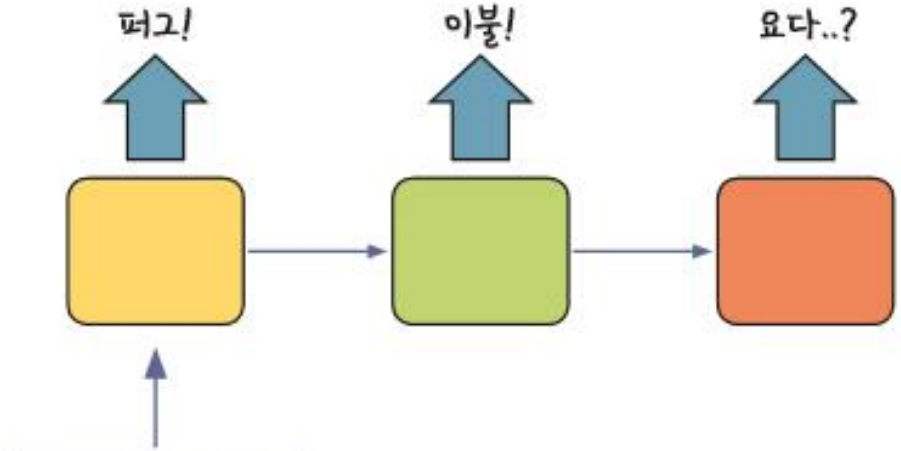


RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

② 단일 입력 다수 출력

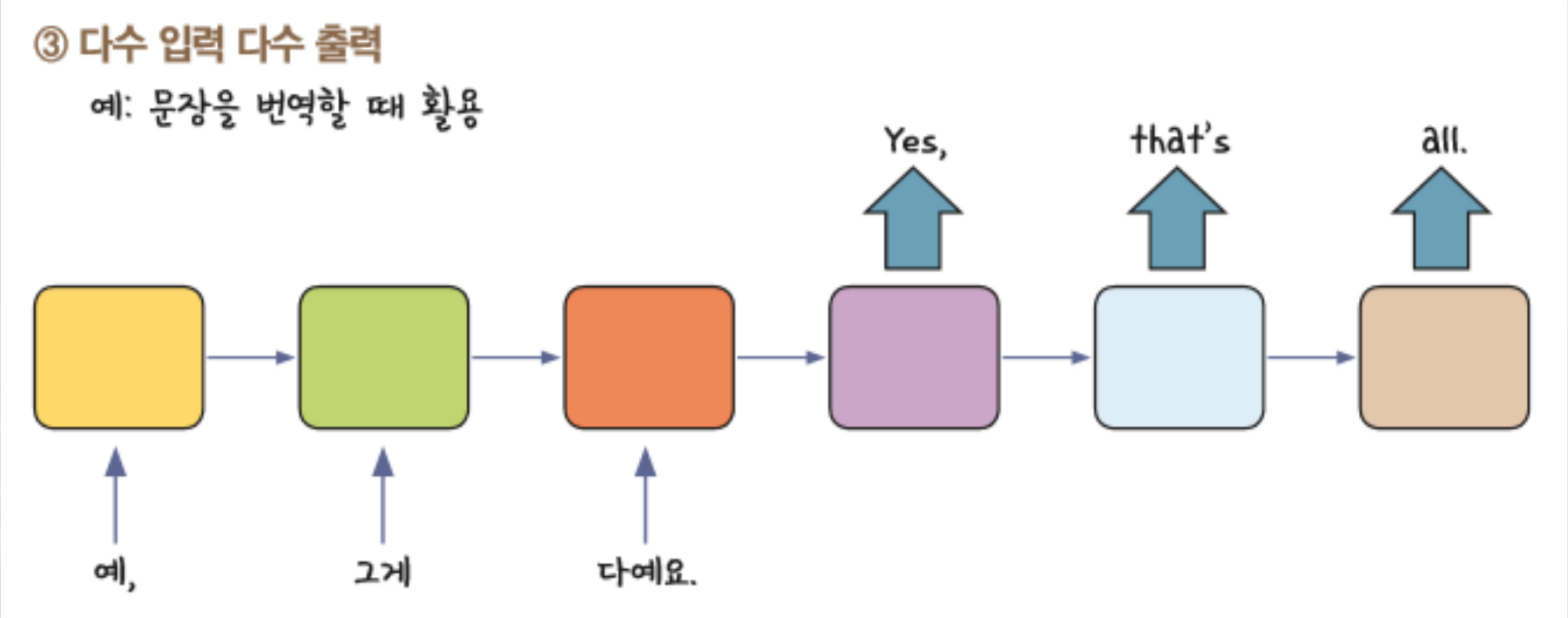
예: 사진의 캡션을 만들 때 활용



RNN(Recurrent Neural Network)

시퀀스 배열로 다루는 순환 신경망

- RNN 방식의 다양한 활용



RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 입력된 문장의 의미를 파악하는 것은 곧 모든 단어를 종합하여 하나의 카테고리로 분류하는 작업이라고 할 수 있음
- 예를 들어 "안녕. 오늘 날씨가 참 좋네"라는 말은 '인사' 카테고리에 분류해야 함
- 그리고 다음과 같이 조금 더 길고 전문적인 말도 정확하게 분류해야 함

중부 지방은 대체로 맑겠으나, 남부 지방은 구름이 많겠습니다.	→	날씨
올 초부터 유동성의 힘으로 주가가 일정하게 상승했습니다.	→	주식
이번 선거에서는 누가 이길 것 같아?	→	정치
퍼셉트론의 한계를 극복한 신경망이 다시 뜨고 있대.	→	딥러닝

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 이번에 실습할 내용은 이처럼 긴 텍스트를 읽고 이 데이터가 어떤 의미를 지니는지 카테고리로 분류하는 연습
- 실습을 위해 로이터 뉴스 데이터를 사용
- 로이터 뉴스 데이터는, 총 11,258개의 뉴스 기사가 46개의 카테고리로 나누어진 대용량 텍스트 데이터
- 데이터는 케라스를 통해 다음과 같이 불러올 수 있음

```
# 로이터 뉴스 데이터셋 불러오기
from keras.datasets import reuters
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 불러온 데이터를 학습셋과 테스트셋으로 나누는 방법은 다음과 같음

```
# 불러온 데이터를 학습셋과 테스트셋으로 나누기
(X_train, Y_train), (X_test, Y_test) = reuters.load_data(num_words=1000, test_split=0.2)
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 불러온 데이터를 학습셋과 테스트셋으로 나누는 방법은 다음과 같음

  reuter.lead_data() 함수를 이용해 기사를 불러왔음

- test_split 인자를 통해 20%를 테스트셋으로 사용

- 여기서 num_words라는 인자는 무엇을 의미하는지 알아보고자 먼저 불러온 데이터에 대해 몇 가지를 출력

```
# 데이터 확인하기
category = numpy.max(Y_train) + 1
print(category, '카테고리')
print(len(X_train), '학습용 뉴스 기사')
print(len(X_test), '테스트용 뉴스 기사')
print(X_train[0])
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- np.max() 함수로 y_train의 종류를 구하니 46개의 카테고리로 구분되어 있음을 알 수 있음(0부터 세기 때문에 1을 더해서 출력)
- 이 중 8,982개는 학습용으로, 2,246개는 테스트용으로 준비되어 있음

```
46 카테고리
8982 학습용 뉴스기사
2246 테스트용 뉴스기사
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207,.....]
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 그런데 print(len(X_train)으로 기사를 출력해 보니 단어가 나오는 게 아니라, [1, 2, 2, 8, 43…] 같은 숫자가 나옴
- 딥러닝은 단어를 그대로 사용하지 않고 숫자로 변환한 다음 학습할 수 있음
- 여기서는 데이터 안에서 해당 단어가 몇 번이나 나타나는지 세어 빈도에 따라 번호를 붙임
- 예를 들어, 3이라고 하면 세 번째로 빈도가 높은 단어라는 뜻
- 이러한 작업을 위해서 tokenizer() 같은 함수를 사용하는데, 케라스는 이 작업을 이미 마친 데이터를 불러올 수 있음

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 기사 안의 단어 중에는 거의 사용되지 않는 것들도 있음
- 모든 단어를 다 사용하는 것은 비효율적이므로 빈도가 높은 단어만 불러와 사용
- 이때 사용하는 인자가 바로 테스트셋과 학습셋으로 나눌 때 함께 적용했던 num_word=1000의 의미
- 빈도가 1~1000에 해당하는 단어만 선택해서 불러오는 것

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 또 하나 주의해야 할 점은 각 기사의 단어 수가 제각각 다르므로 단어의 숫자를 맞춰야 한다는 것
- 이때는 데이터 전처리 함수 sequence를 다음과 같이 이용할 수 있음
- 여기서 maxlen=100은 단어 수를 100개로 맞추라는 뜻
- 입력된 기사의 단어 수가 100보다 크면 100개째 단어만 선택하고 나머지는 버림
- 100에서 모자랄 때는 모자라는 부분을 모두 0으로 채움

```
from keras.preprocessing import sequence

# 데이터 전처리
x_train = sequence.pad_sequences(X_train, maxlen=100)
x_test = sequence.pad_sequences(X_test, maxlen=100)
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 이제 y 데이터에 원-핫 인코딩 처리를 하여 데이터 전처리 과정을 마침

```
# 데이터 전처리
y_train = np_utils.to_categorical(Y_train)
y_test = np_utils.to_categorical(Y_test)
```

- 데이터 전처리 과정이 끝났으므로 딥러닝의 구조를 만들 차례

```
# 모델의 설정
model = Sequential()
model.add(Embedding(1000, 100))
model.add(LSTM(100, activation='tanh'))
model.add(Dense(46, activation='softmax'))
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- Embedding 층과 LSTM 층이 새로 추가된 것이 보임
- Embedding 층은 데이터 전처리 과정을 통해 입력된 값을 받아 다음 층이 알아들을 수 있는 형태로 변환하는 역할을 함
- Embedding('불러온 단어의 총 개수', '기사당 단어 수') 형식으로 사용하며, 모델 설정 부분의 맨 처음에 있어야 함
- LSTM은 앞서 설명했듯이 RNN에서 기억 값에 대한 가중치를 제어함
- LSTM(기사당 단어 수, 기타 옵션)의 형태로 적용됨
- LSTM의 활성화 함수로는 Tanh를 사용

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

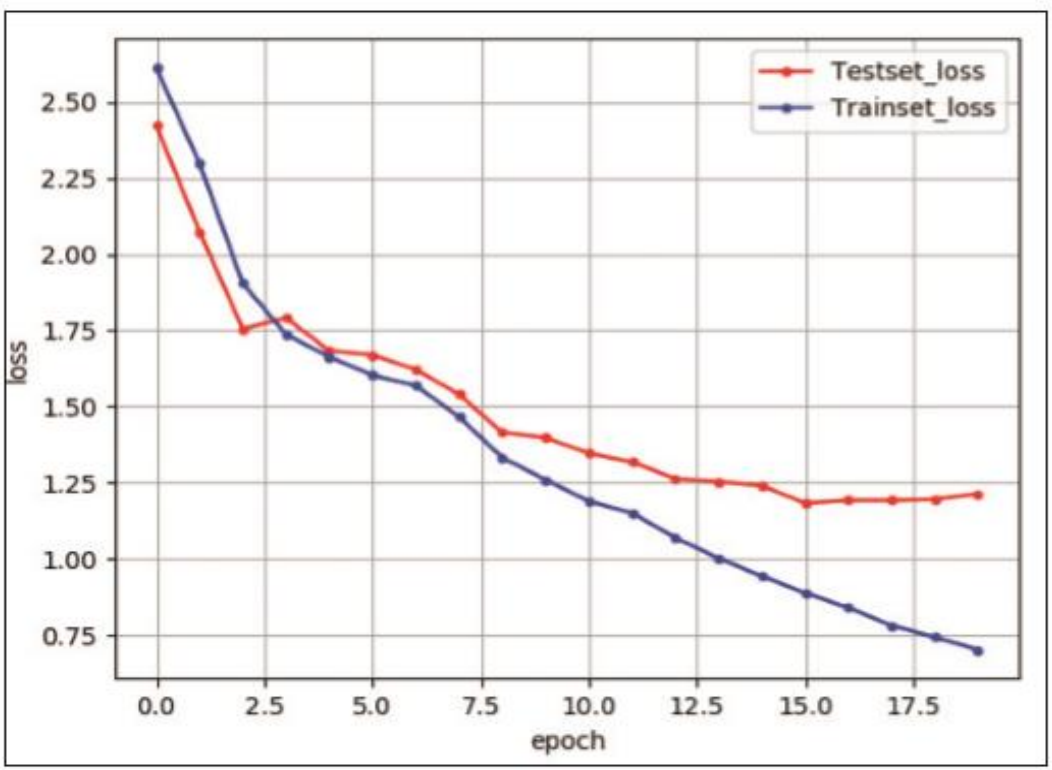
- 컴파일, 실행 및 정확도를 측정

```
# 모델의 컴파일
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 모델의 실행
history = model.fit(x_train, y_train, batch_size=100, epochs=20, validation_data=(x_test,
y_test))

# 테스트 정확도 출력
print("₩n Test Accuracy: %.4f" % (model.evaluate(X_test, Y_test) [1]))
```

RNN(Recurrent Neural Network)

LSTM을 이용한 로이터 뉴스 카페고리 분류하기

- 테스트셋에 대한 정확도가 0.7128을 보이고 있음
- 테스트 오차가 상승하기 전까지의 학습이 과적합 직전의 최적 학습 시간임



RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 이번에 사용할 인터넷 영화 데이터베이스(Internet Movie Database, IMDB)는 영화와 관련된 정보와 출연진 정보, 개봉 정보, 영화 후기, 평점에 이르기까지 매우 폭넓은 데이터가 저장된 자료
- 영화에 관해 남긴 2만 5000여 개의 영화 리뷰가 담겨 있으며, 해당 영화를 긍정적으로 평가했는지 혹은 부정적으로 평가했는지도 담겨 있음
- 로이터 뉴스 데이터와 마찬가지로 각 단어에 대한 전처리를 마친 상태
- 데이터셋에서 나타나는 빈도에 따라 번호가 정해지므로 빈도가 높은 데이터를 불러와 학습시킬 수 있음

RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 데이터 전처리 과정은 로이터 뉴스 데이터와 거의 같음

- 다만 클래스가 긍정 또는 부정 두 가지뿐이라 원-핫 인코딩 과정이 없음

```
# 학습셋과 테스트셋 지정하기
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=5000)

# 데이터 전처리
x_train = sequence.pad_sequences(X_train, maxlen=100)
x_test = sequence.pad_sequences(X_test, maxlen=100)
```

RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 모델을 다음과 같이 설정
- 마지막에 model.summary() 함수를 넣으면 현재 설정된 모델의 구조를 한 눈에 볼 수 있음

```
# 모델의 설정
model = Sequential()
model.add(Embedding(5000, 100))
model.add(Dropout(0.5))
model.add(Conv1D(64, 5, padding='valid', activation='relu', strides=1))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(55))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.summary()
```

RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 실행 결과는 다음과 같음

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         500000
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 100)         0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, None, 64)          32064
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, None, 64)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 55)                26400
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 56
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 558,520
Trainable params: 558,520
Non-trainable params: 0
_________________________________________________________________
```

RNN(Recurrent Neural Network)

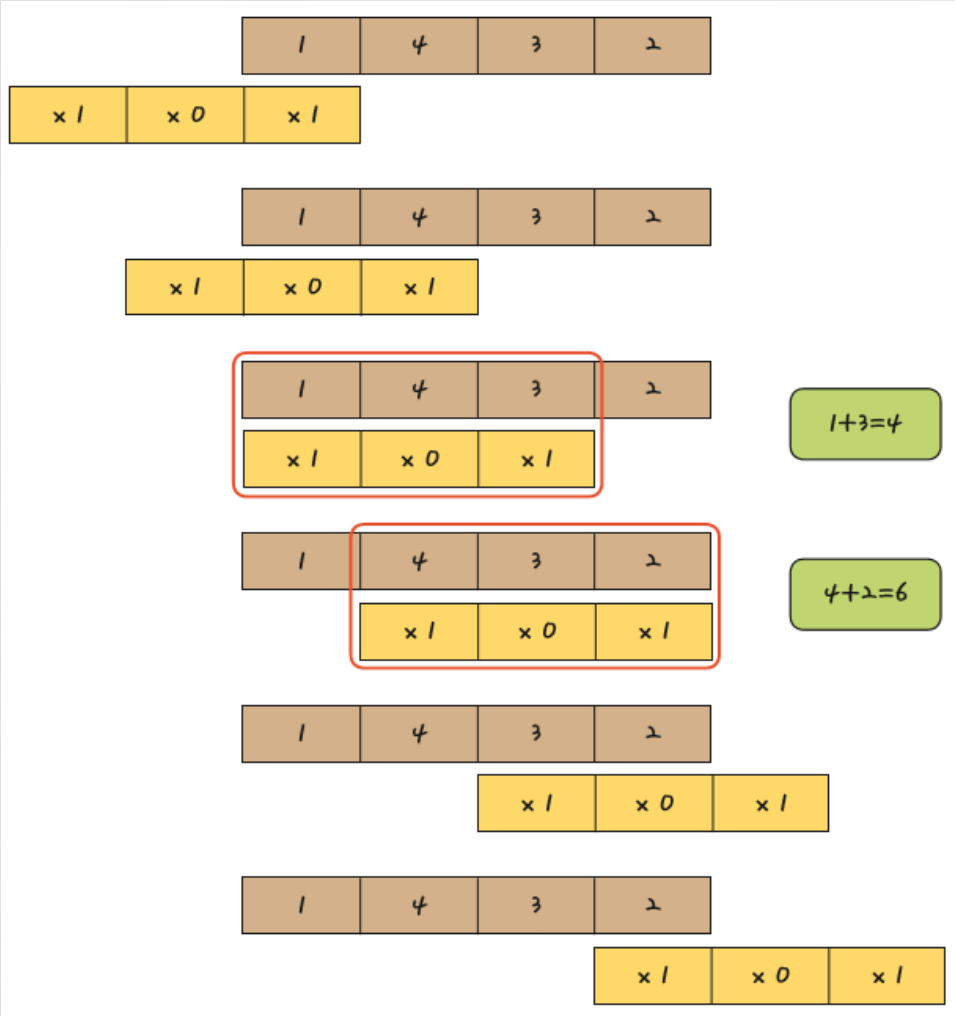
LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 앞서 Conv2D와 MaxPooling2는 앞서 “MINIST 손글씨 인식”에서 다루었음
- 하지만 2차원 배열을 가진 이미지와는 다르게 지금 다루고 있는 데이터는 배열 형태로 이루어진 1차원이라는 차이가 있음
- 역서 사용된 Conv1D는 Conv2D의 개념을 1차원으로 옮긴 것
- 컨볼루션 층이 1차원이고 이동하는 배열도 1차원

RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

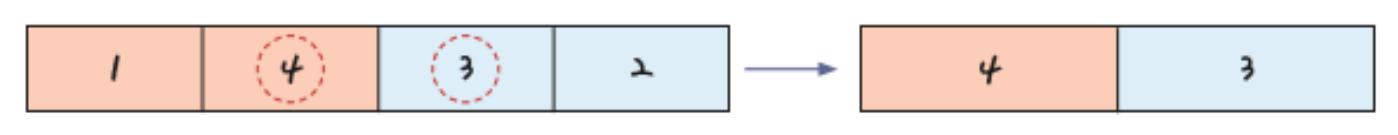
- Conv1D



RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

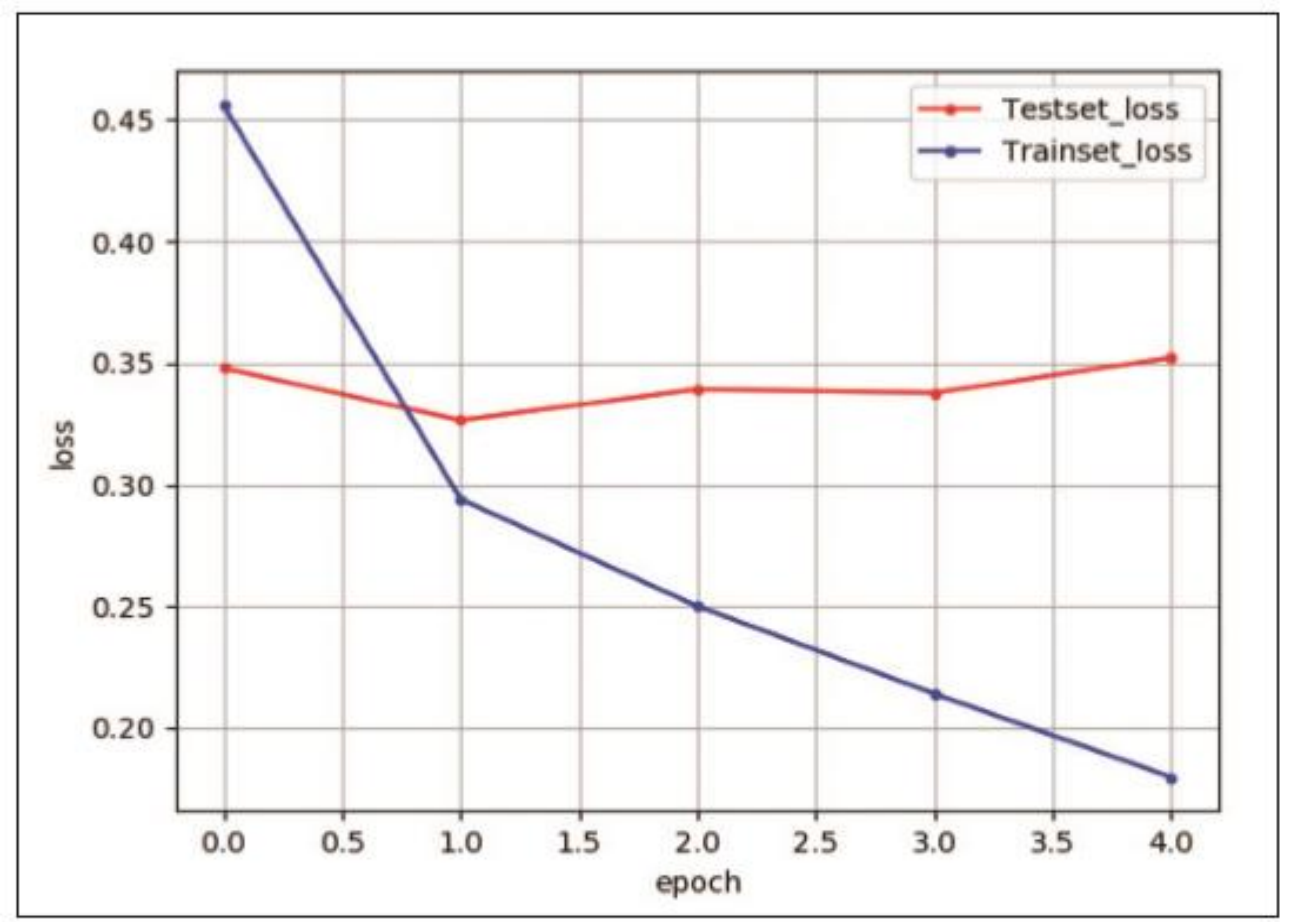
- MaxPooling1D 역시 마찬가지
- 2차원 배열이 1차원으로 바뀌어 정해진 구역 안에서 가장 큰 값을 다음 층으로 넘기고 나머지는 버림



RNN(Recurrent Neural Network)

LSTM과 CNN의 조합을 이용한 영화 리뷰 분류하기

- 그래프 출력 결과



Any Questions?