Deep Learning from Scratch

조용군

iceman4u@naver.com

매개변수 갱신

매개변수 갱신

✓ 신경망 학습

- 손실 함수의 값을 가능한 한 낮추는 매개변수 찾기
- 매개변수의 최적값을 찾는 최적화(optimization) 문제
- 매우 어려운 작업
  - 매개변수 공간이 넓고 복잡하므로
- 단서
  - 기울기(미분, Gradient)
- SGD (Stochastic Gradient Descent)
  - 기울기를 단서로 기울어진 방향으로 매개변수 값을 갱신하는 일을 여러 번 반복하여 점점 최적값에 다가가는 방법

매개변수 갱신

✓ 모험가 이야기

"색다른 모험가가 있습니다. 광활한 메마른 산맥을 여행하면서 날마다 깊은 골짜기를 찾아 발걸음을 옮깁니다. 그는 전설에 나오는 세상에서 가장 깊은 골짜기, '깊은 곳'을 찾아가려 합니다. 그것이 그의 여행 목적이죠. 게다가 그는 엄격한 '제약' 2개로 자신을 옭아맸습니다. 하나는 지도를 보지 않을 것, 또 하나는 눈가리개를 쓰는 것입니다. 지도도 없고 보이지도 않으니 가장 깊은 골짜기가 광대한 땅 어디에 있는지 알 도리가 없죠. 그런 혹독한 조건에서 이 모험가는 어떻게 '깊은 곳'을 찾을 수 있을까요? 어떻게 걸음을 옮겨야 효율적으로 '깊은 곳'을 찾을 수 있을까요?"

매개변수 갱신

✓ 모험가 이야기

"색다른 모험가가 있습니다. 광활한 메마른 산맥을 여행하면서 날마다 깊은 골짜기를 찾아 발걸음을 옮깁니다. 그는 전설에 나오는 세상에서 가장 깊은 골짜기, '깊은 곳'을 찾아가려 합니다. 그것이 그의 여행 목적이죠. 게다가 그는 엄격한 '제약' 2개로 자신을 옭아맸습니다. *하나는 지도를 보지 않을 것, 또 하나는 눈가리개를 쓰는 것입니다.* 지도도 없고 보이지도 않으니 가장 깊은 골짜기가 광대한 땅 어디에 있는지 알 도리가 없죠. 그런 혹독한 조건에서 이 모험가는 어떻게 '깊은 곳'을 찾을 수 있을까요? 어떻게 걸음을 옮겨야 효율적으로 '깊은 곳'을 찾을 수 있을까요?"

모험가(SGD)에게 중요한 단서

*"땅의 **기울기**"*

매개변수 갱신

✓ 확률적 경사 하강법(SGD)

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

```
Class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr

    def update(self, params, grads):
        for key in params.keys():
            params[key] -= self.lr * grads[key]
```

매개변수 갱신

✓ 확률적 경사 하강법(SGD)

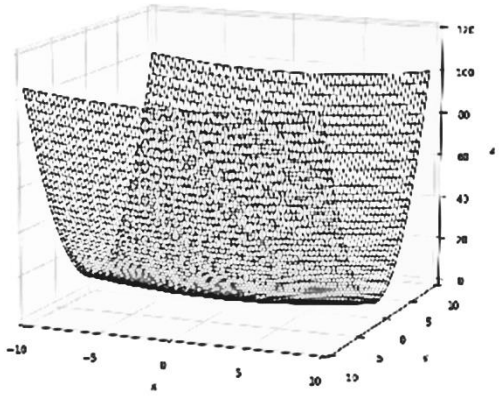
- 매개변수 갱신용 의사 코드

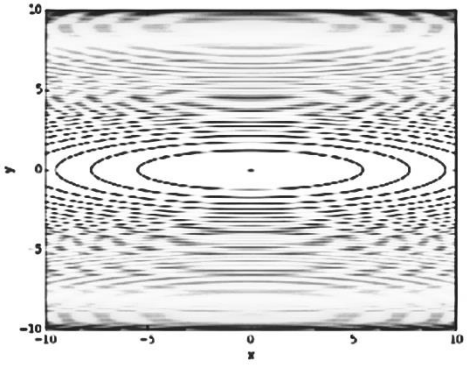
```
network = TwoLayerNet(...)
optimizer = SGD()
#optimizer = Momentum()
#optimizer = AdaGrad()
#optimizer = Adam()

for i in range(epoch):
    ...
    x_batch, t_batch = get_mini_batch(...) # 미니 배치
    grads = network.gradient(x_batch, t_batch)
    params = network.params
    optimizer.update(params, grads)
    ...
```

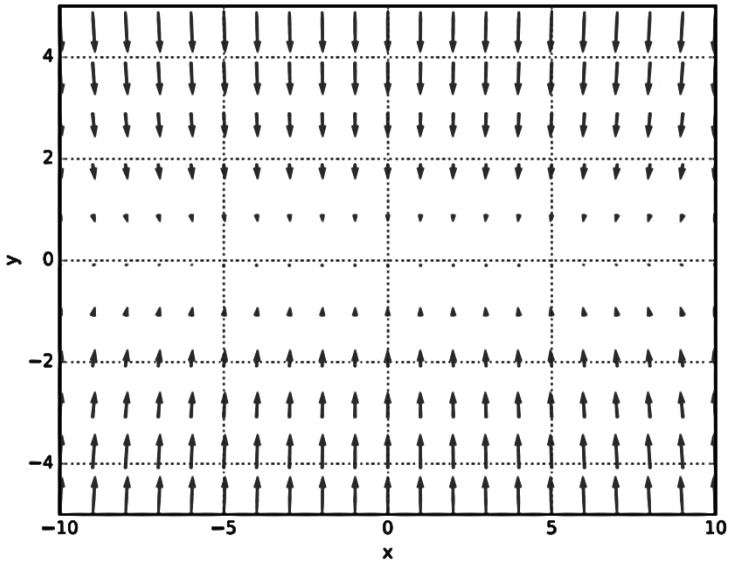
매개변수 갱신

- ✓ **SGD의 단점**
  - 문제에 따라 비효율
  - (손실) 함수가 비등방성(anisotropy)인 경우
    - 축에 따른 기울기가 다른 경우

$$f(x,y) = \frac{1}{20}x^2 + y^2$$



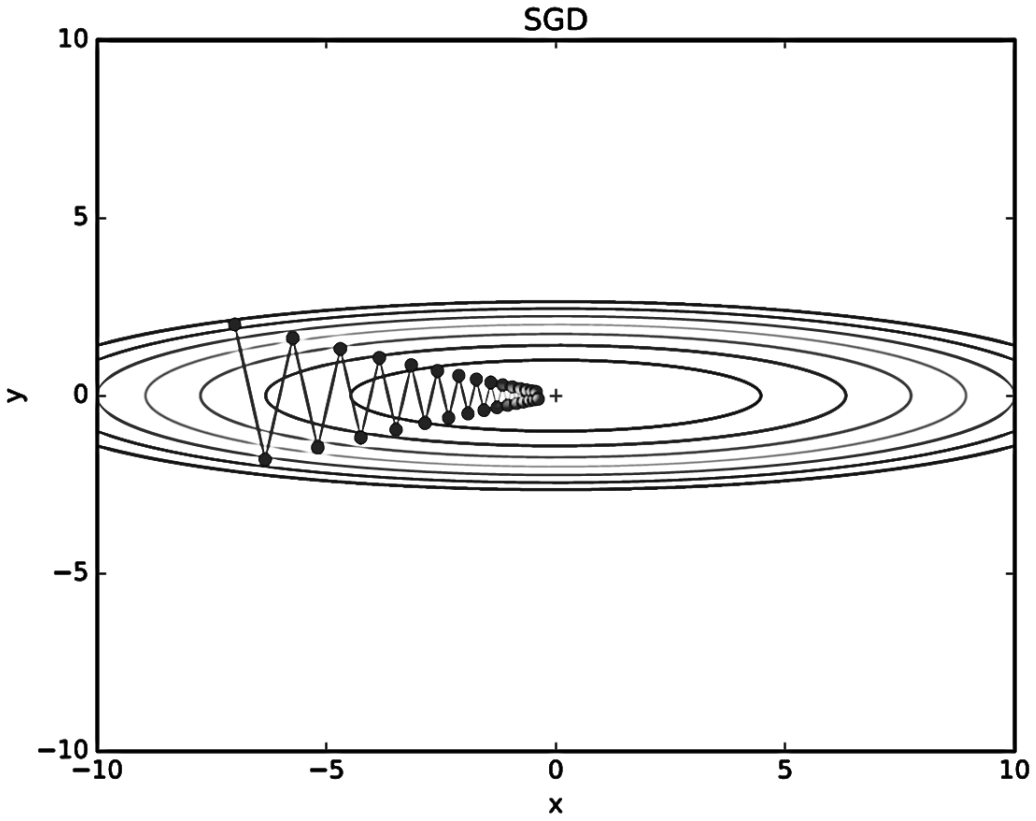




매개변수 갱신

✓ SGD의 단점

- SGD에 의한 최적화 갱신 경로



매개변수 갱신

✓ **모멘텀(Momentum)**

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$ v: 속도

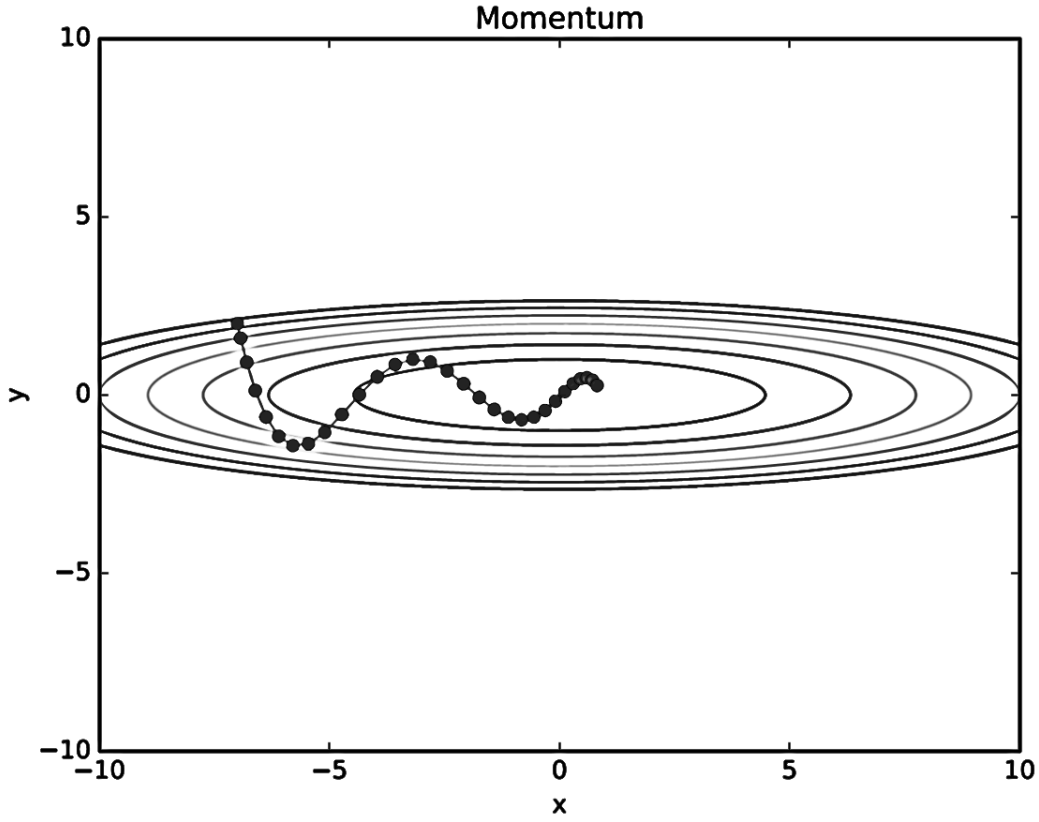
- 기울기가 0이 되어도 αv 만큼의 속도로 가중치 매개변수 공간을 이동

- common/optimizer.py

매개변수 갱신

✓ 모멘텀(Momentum)

- 모멘텀에 의한 최적화 갱신 경로



매개변수 갱신

✓ AdaGrad

- 학습률 감소 (learning rate decay) 기법
  - 학습을 진행하면서 학습률을 점차 줄여가는 방법

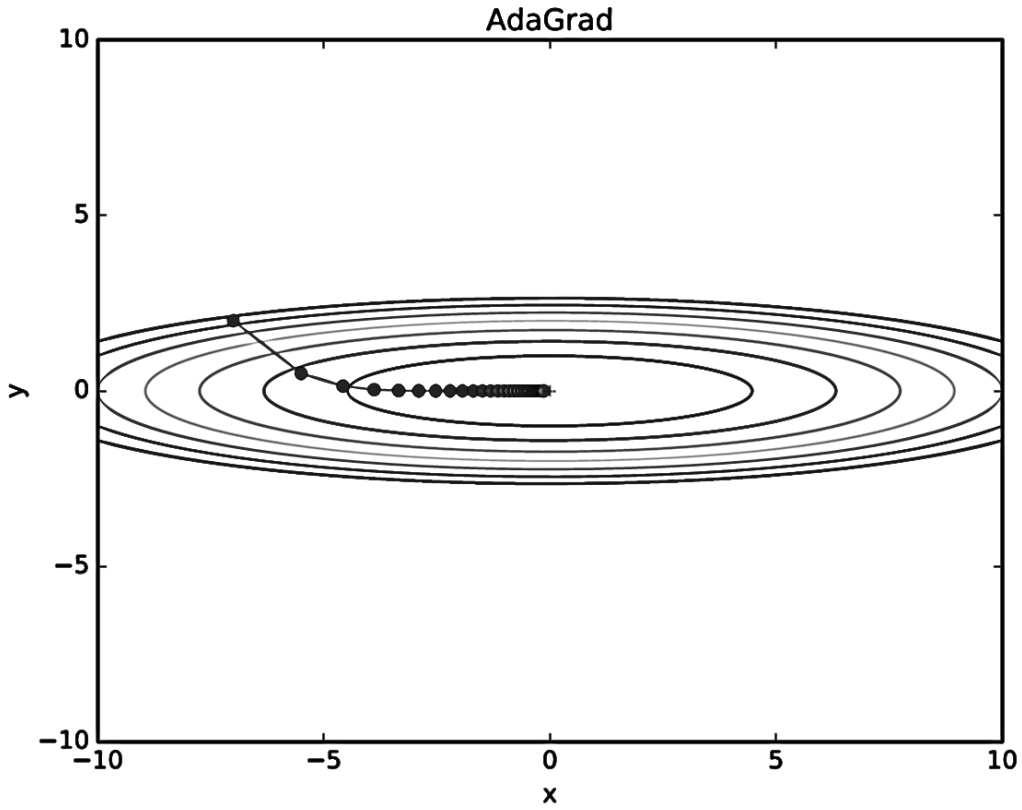
- 개별 매개변수에 적응적으로 학습률을 조정하면서 학습을 진행하는 알고리즘
  - 개별 매개변수에 적합한 '맞춤형' 학습률을 제공

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$

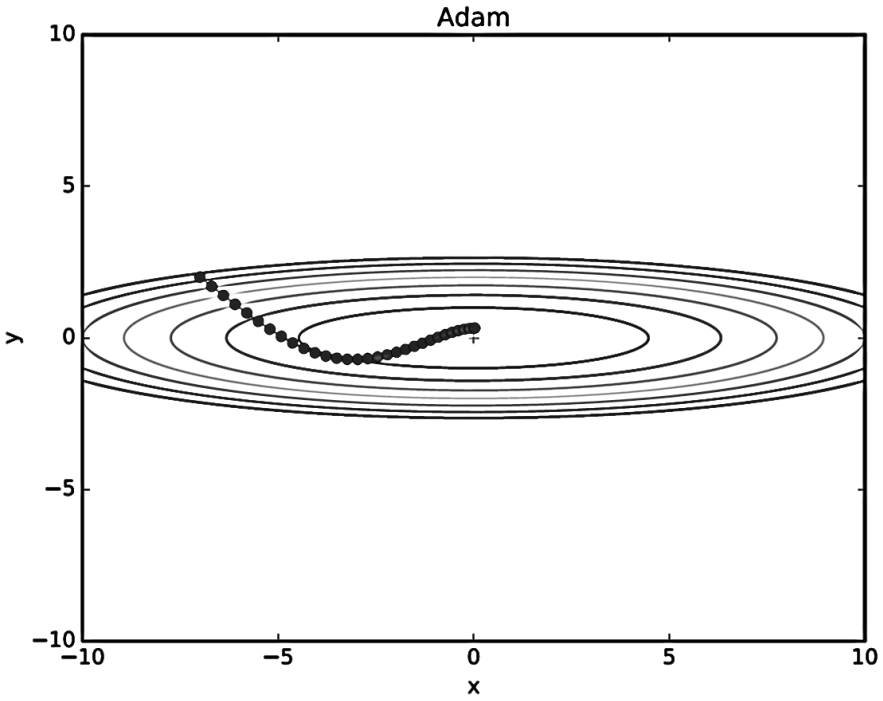
매개변수 갱신

- ✓ **AdaGrad**
  - AdaGrad에 의한 최적화 갱신 경로



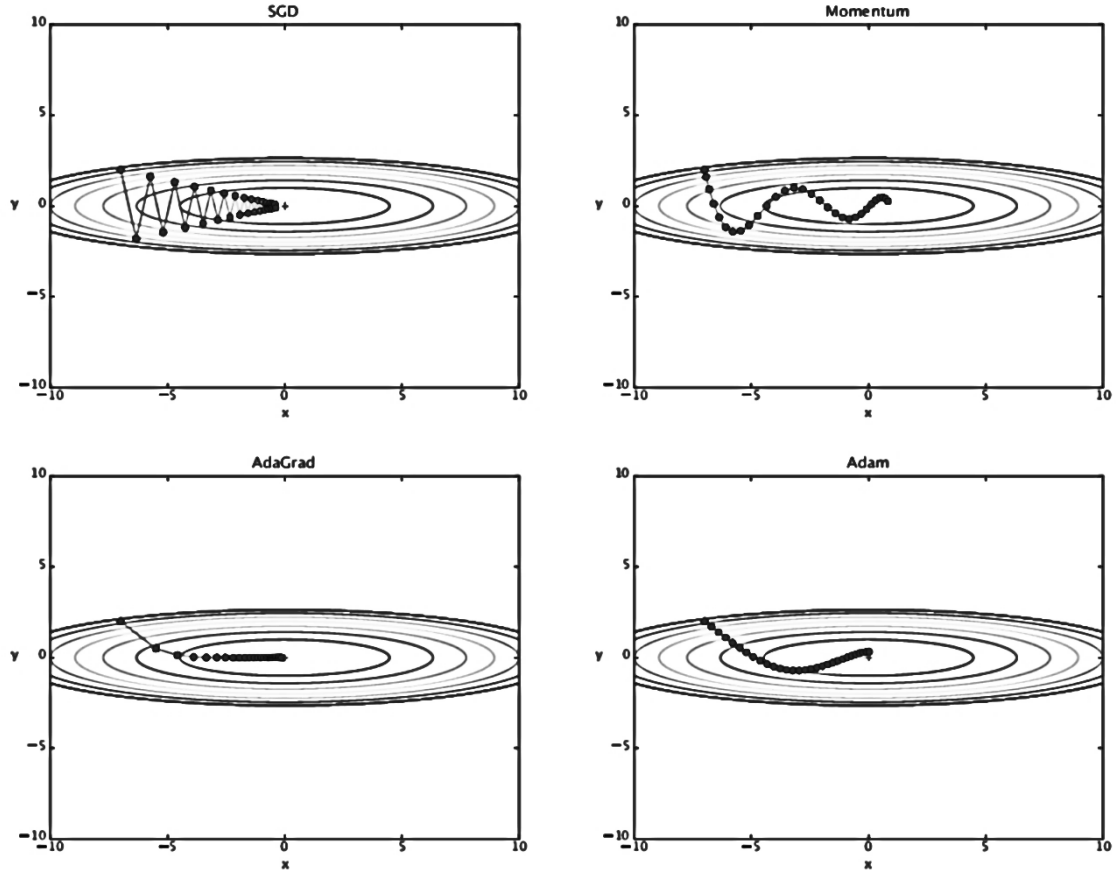
매개변수 갱신

- ✓ **Adam**
  - 모멘텀과 AdaGrad의 장점을 융합
  - 2015년에 제안된 새로운 방법
  - Adam에 의한 최적화 갱신 경로



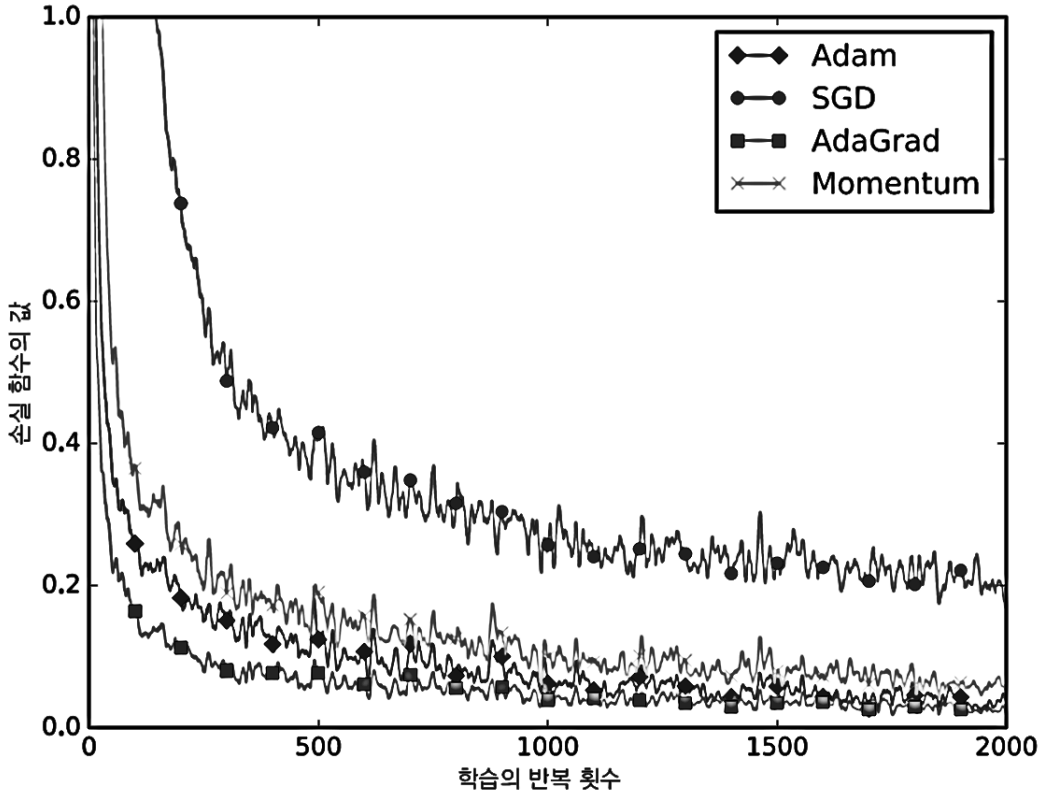
매개변수 갱신

✓ 어떤 갱신 방법을 이용할 것인가?



매개변수 갱신

- ✓ **MNIST 데이터셋으로 본 갱신 방법 비교**
  - ch06/optimizer_compare_mnist

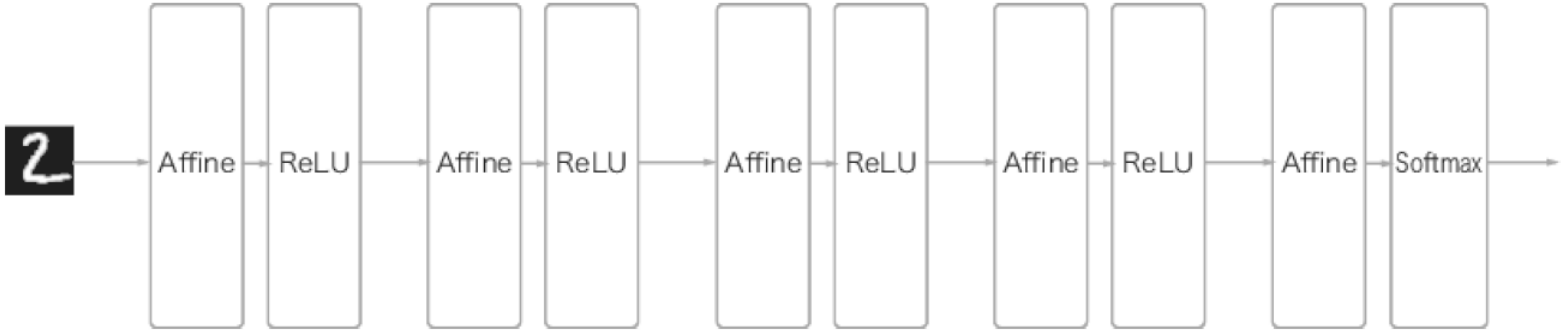


합성곱 신경망 CNN

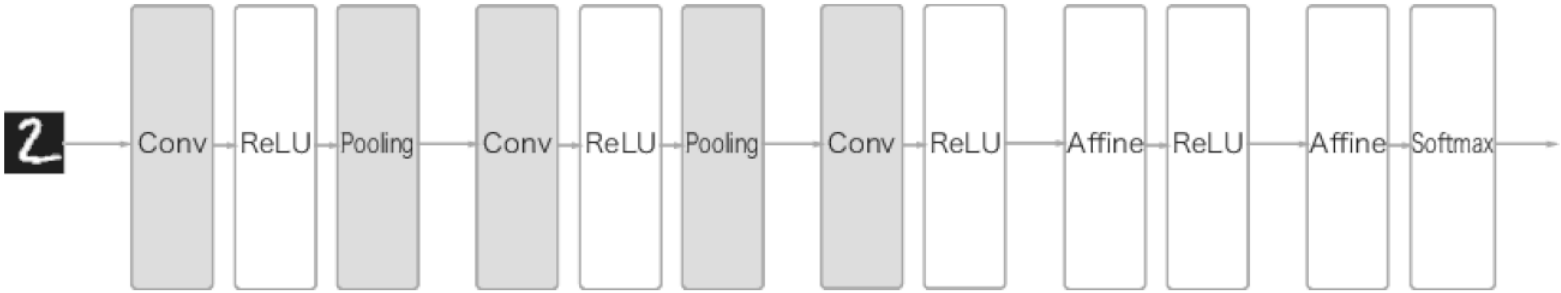
Affine 계층 vs. CNN

✓ **Affine 계층**

- 계층(레이어)간 노드들이 완전 연결되어 있는 신경망 계층



✓ **CNN**



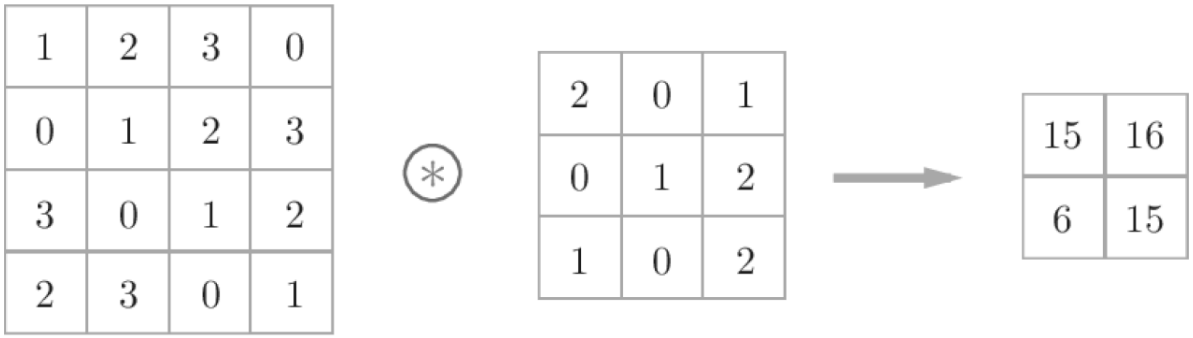
✓ 완전연결 계층의 문제점

- 데이터의 형상이 무시된다

- 입력 데이터가 이미지라면
  - 세로
  - 가로
  - 채널(색상)
  - 공간적 정보: 완전연결 계층에서는 공간적 정보를 무시하고 모든 입력 데이터를 동등한 뉴런으로 취급

✓ 합성곱 연산(컨벌루션 연산, convolution)

- 이미지 처리에서의 필터 연산과 같은 작업

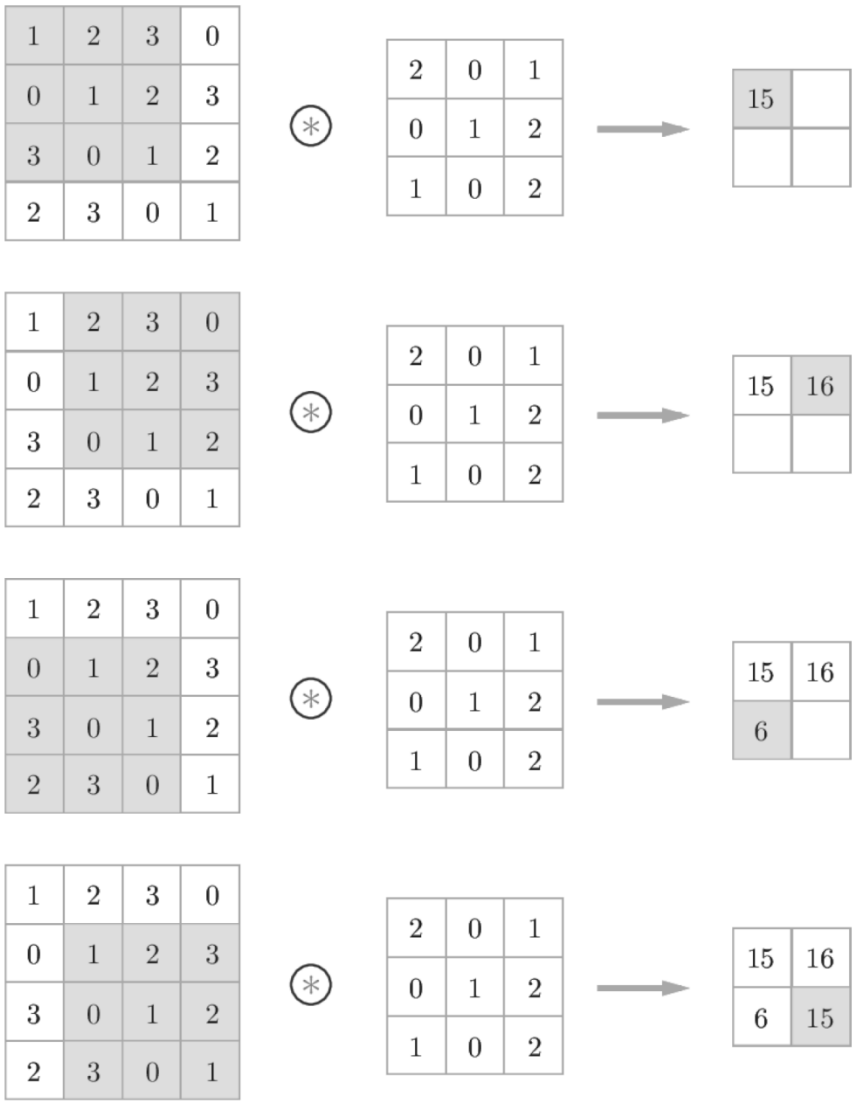


입력 데이터 필터

- 필터의 윈도우(window)를 일정 간격으로 이동해가며 입력 데이터에 적용
- 단일 곱셈-누산(fused muliply-add, FMA)

- 특징 맵(feature map)
  - CNN에서의 입출력 데이터

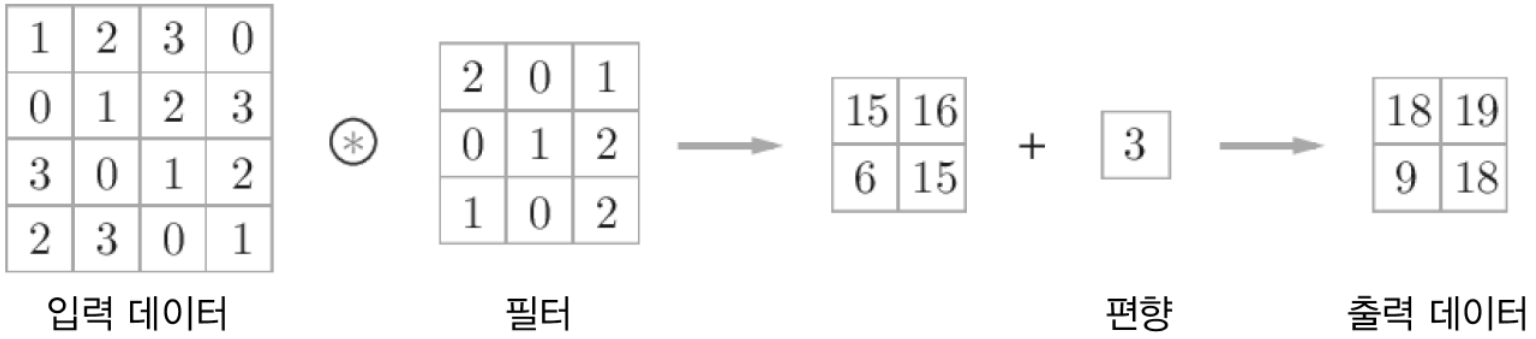
✓ **합성곱 연산의 계산 순서**



✓ 합성곱 연산

- http://cs231n.github.io/convolutional-networks/

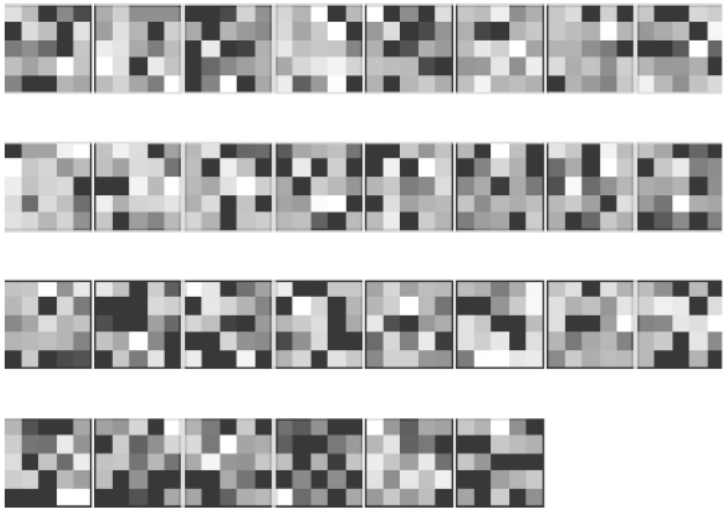
✓ 바이어스 고려



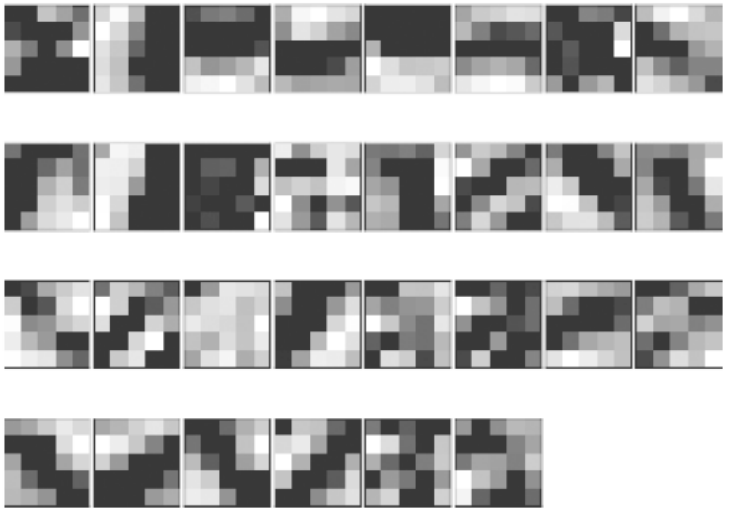
✓ 학습

- 필터의 매개변수가 다른 신경망의 가중치에 해당

학습 전



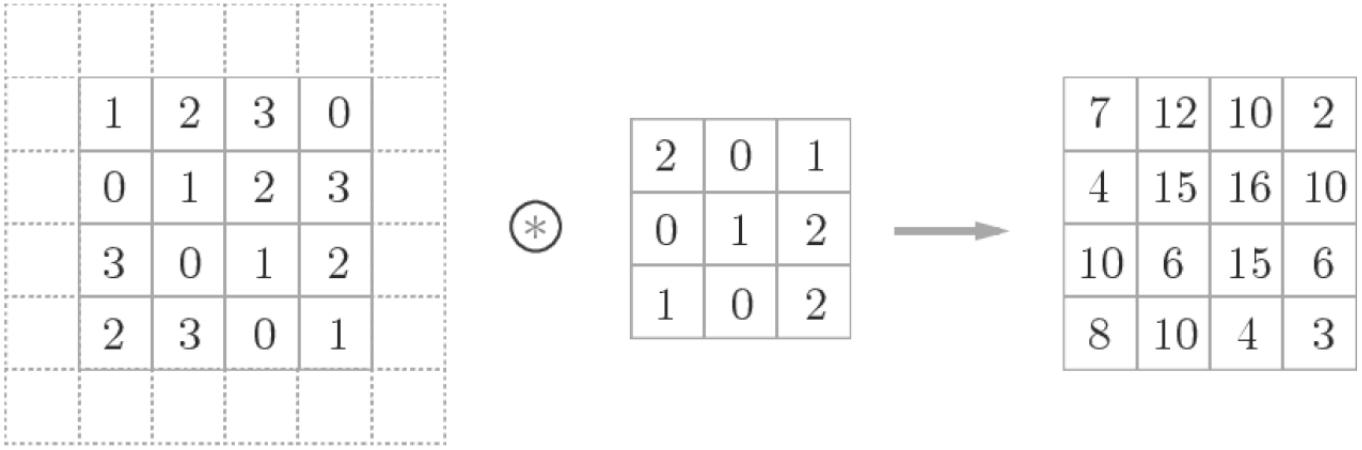
학습 후



✓ 패딩(padding)

- 컨벌루션(합성곱 연산)을 수행하기 전에 입력 데이터 주변을 특정 값으로 채우는 작업

- 폭 1짜리 패딩



✓ 패딩 사용 목적

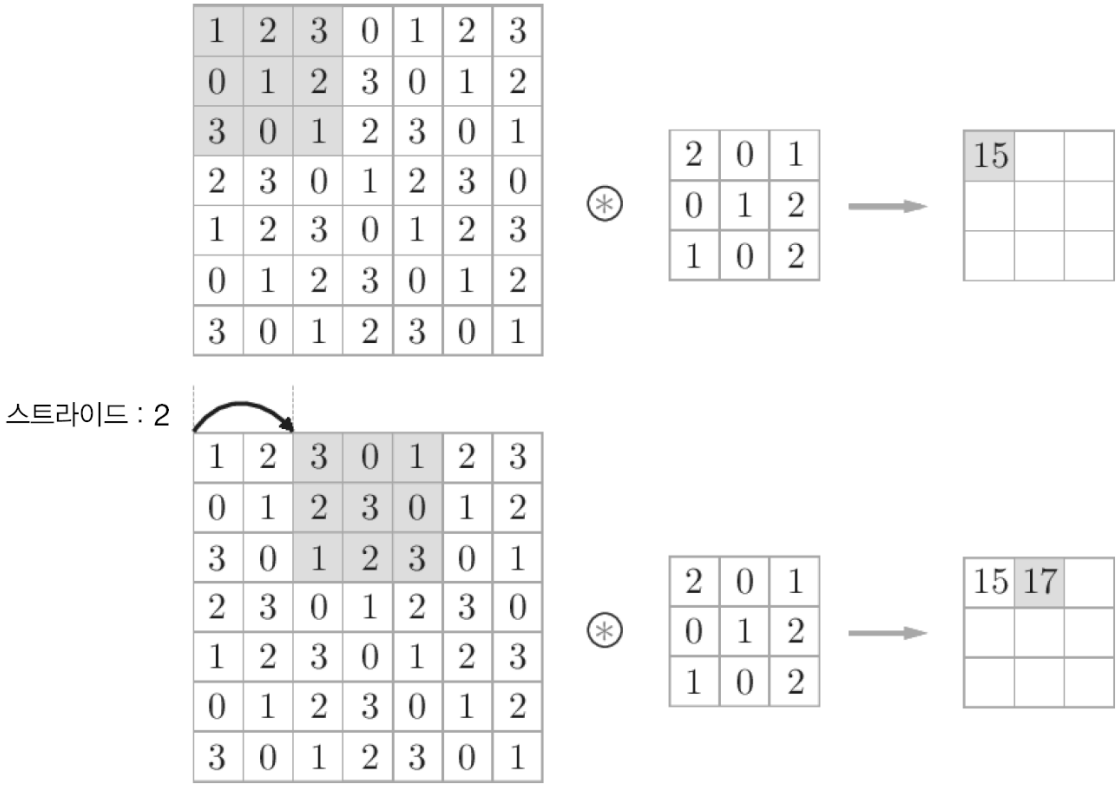
- 출력 크기 조정

- (4, 4) 입력 데이터에 (3, 3) 필터 적용
  - (2, 2) 출력
  - 컨벌루션을 여러 번 적용 → 출력 크기가 1 → 컨벌루션 적용 불가

- (4, 4) 입력 데이터에 패딩 폭 1 → (6, 6) 입력 데이터
  - (3, 3) 필터 적용 → (4, 4) 출력 데이터

✓ 스트라이드(stride)

- 필터 적용하는 위치 간격
- 스트라이드가 2인 경우



✓ 스트라이드, 패딩과 출력의 관계

- 스트라이드 ↑, 출력 ↓
- 패딩 ↑, 출력 ↑

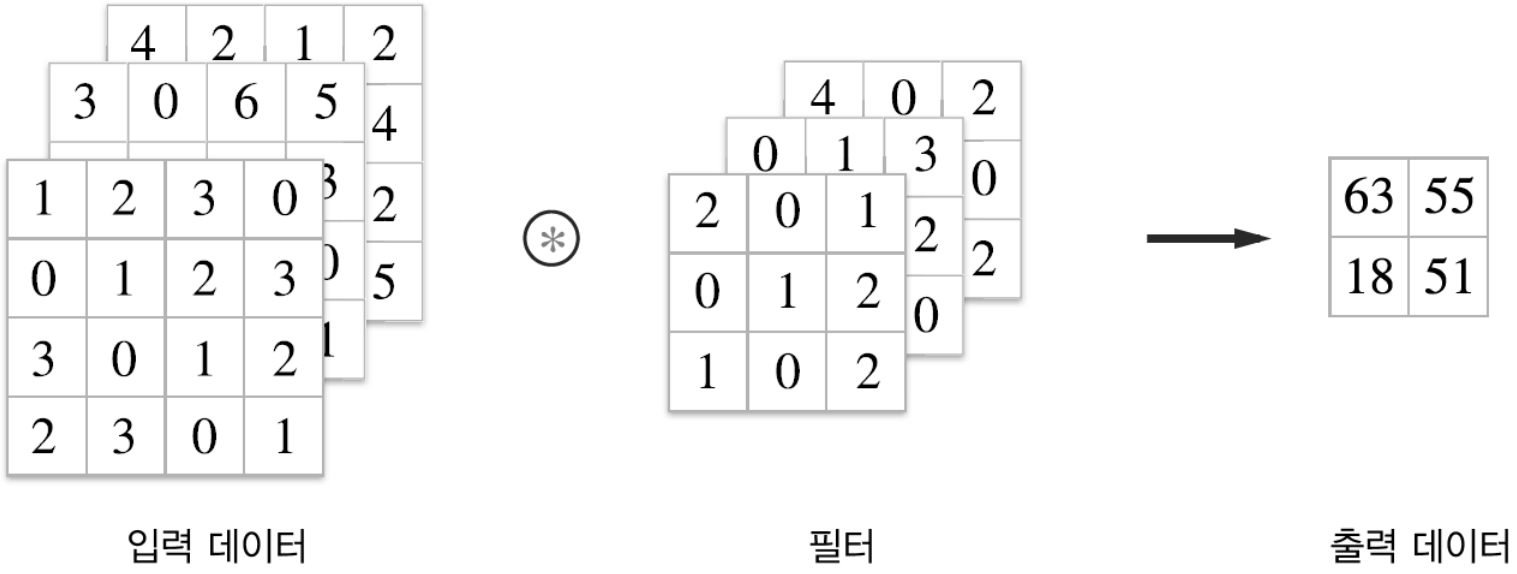
$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{\mathrm{W} + 2P - F\mathrm{W}}{S} + 1$$

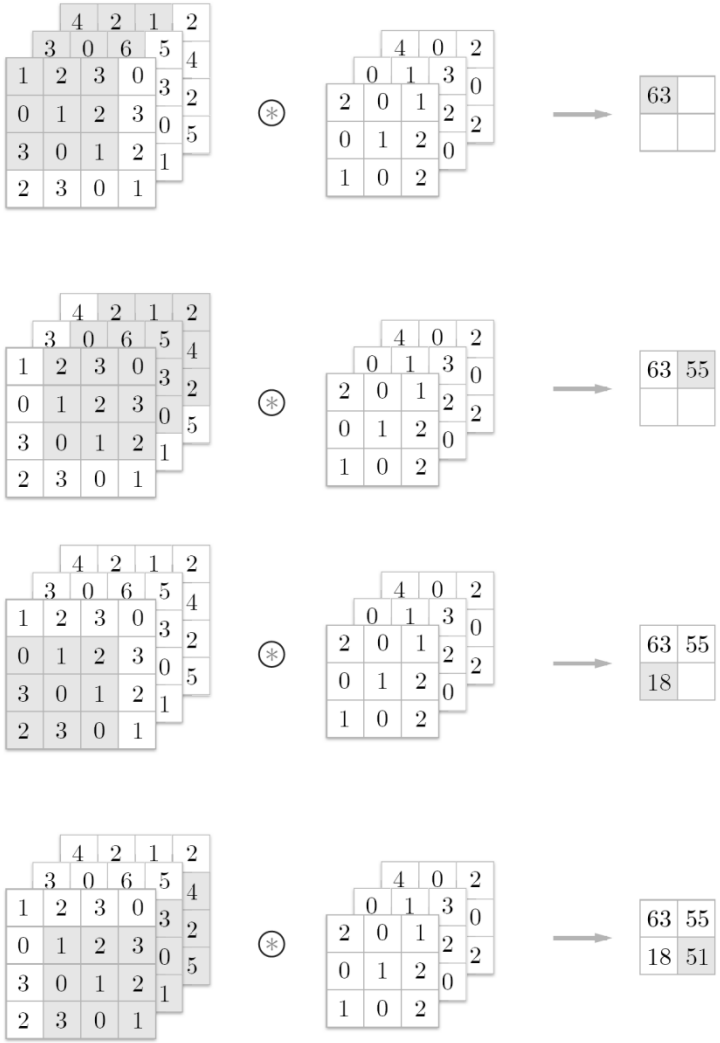
- 입력: (4, 4), 패딩:1, 스트라이드:1, 필터: (3, 3)
  - 출력은?
- 입력: (7, 7), 패딩:0, 스트라이드:2, 필터:(3, 3)
- 입력: (28, 31), 패딩:2, 스트라이드: 3, 필터: (5, 5)

✓ 3차원 데이터의 합성곱 연산

- 채널까지 고려한 이미지 데이터: 3차원 데이터
- 필터도 3차원: 입력 데이터의 채널 수와 필터의 채널 수가 일치해야 함
- 출력 데이터는 2차원

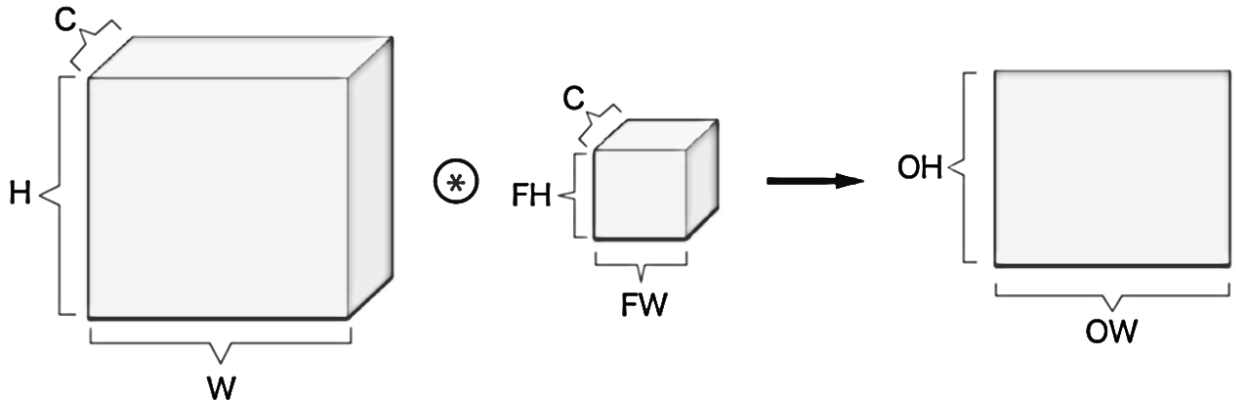


CNN


4 2 1 2
3 0 6 5
1 2 3 0
0 1 2 3
3 0 1 2
2 3 0 1
4 0 2
0 1 3
2 0 1
0 1 2
1 0 2
63
63 55
63 55
18
63 55
18 51

✓ 블록으로 생각하기

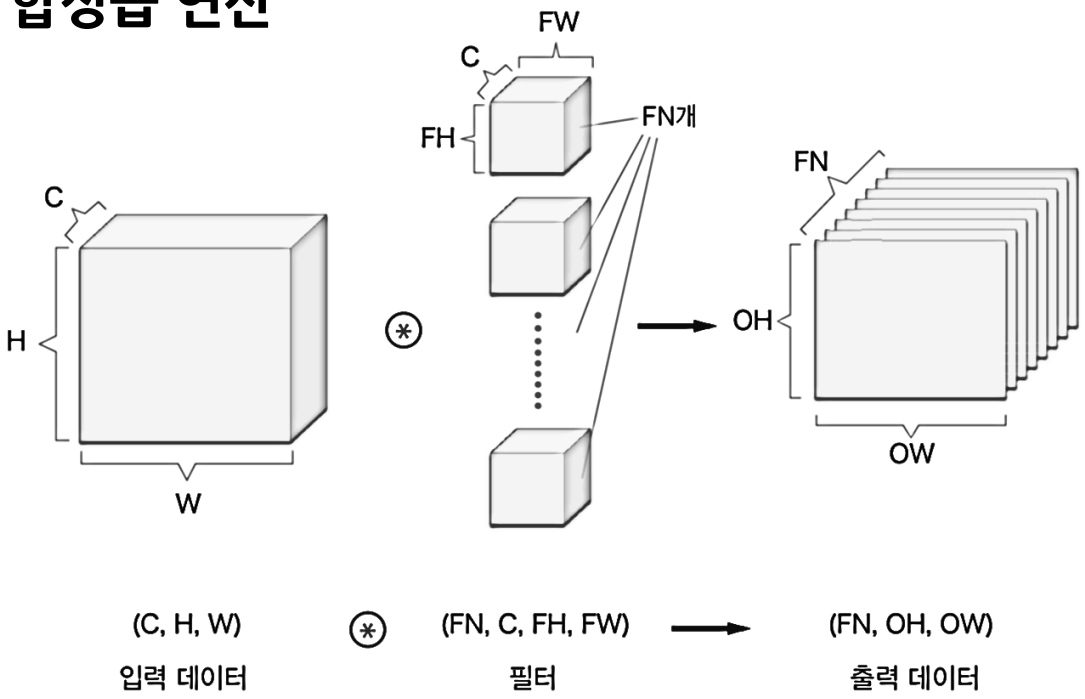
- 데이터와 필터를 직육면체 블록으로 간주
- (채널, 높이, 너비) 순서로 표기
  - (C, H, W)
  - (C, FH, FW)



(C, H, W) 입력 데이터 ⊛ (C, FH, FW) 필터 → (1, OH, OW) 출력 데이터

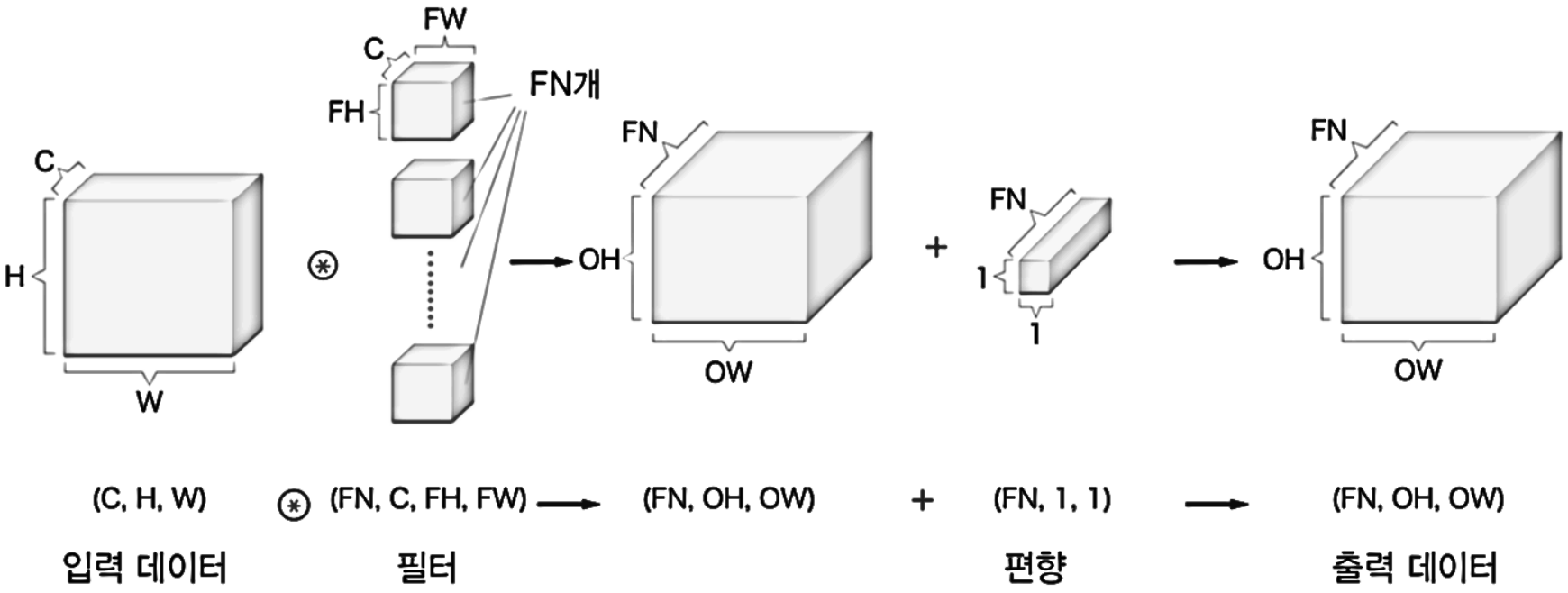
- ✓ **필터의 개수 ➔ 출력 특징 맵 개수**
  - 필터의 가중치 데이터: 4차원 데이터
    - (20, 3, 5, 5)

- ✓ **여러 필터를 사용한 합성곱 연산**



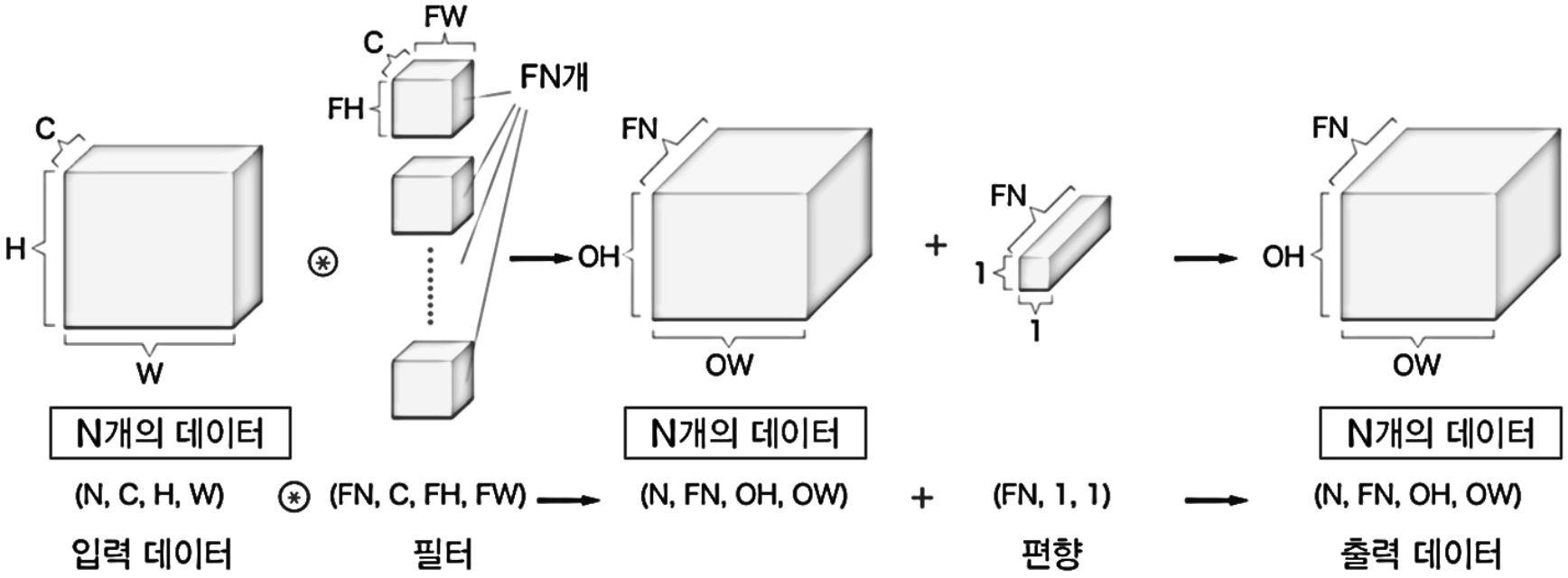
✓ 바이어스가 추가된 합성곱 연산의 처리 흐름

- 채널 당 바이어스 하나 할당



✓ 배치 처리

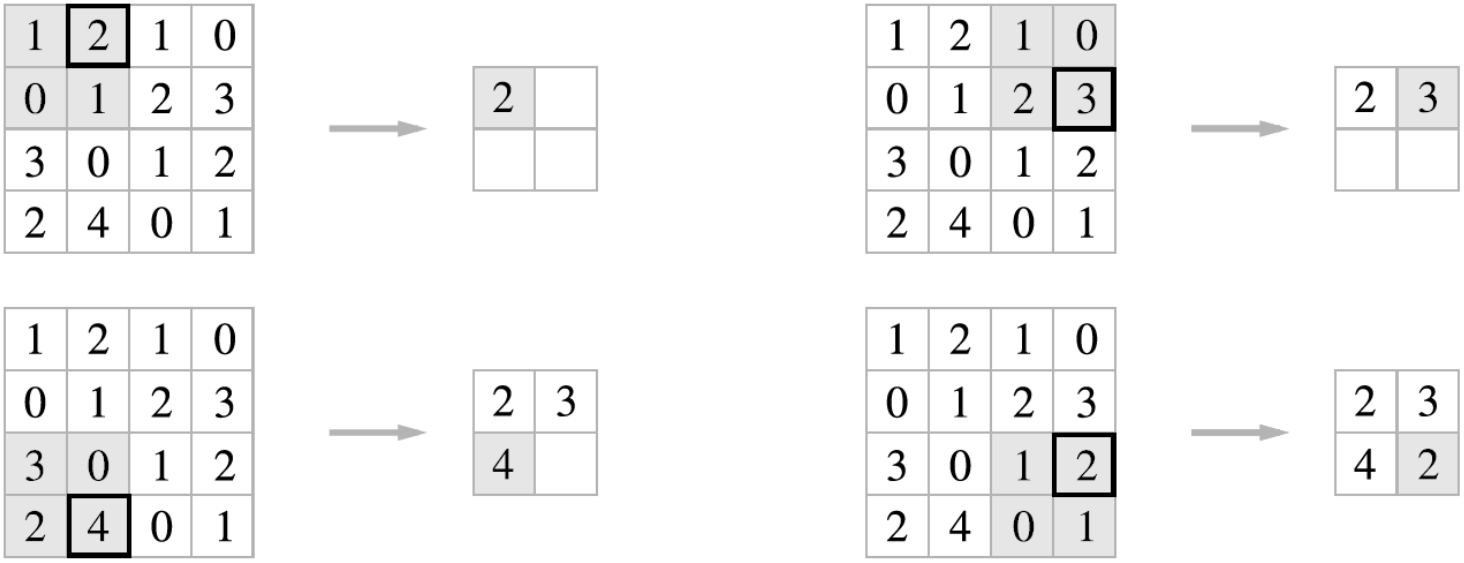
- 각 신경망 계층을 흐르는 데이터: 4차원
  - (데이터 수, 채널 수, 높이, 너비)
- N 데이터



✓ 풀링 계층(pooling layer)

- 풀링
  - 세로 및 가로 방향의 공간을 줄이는 연산
  - 최대 풀링, 평균 풀링

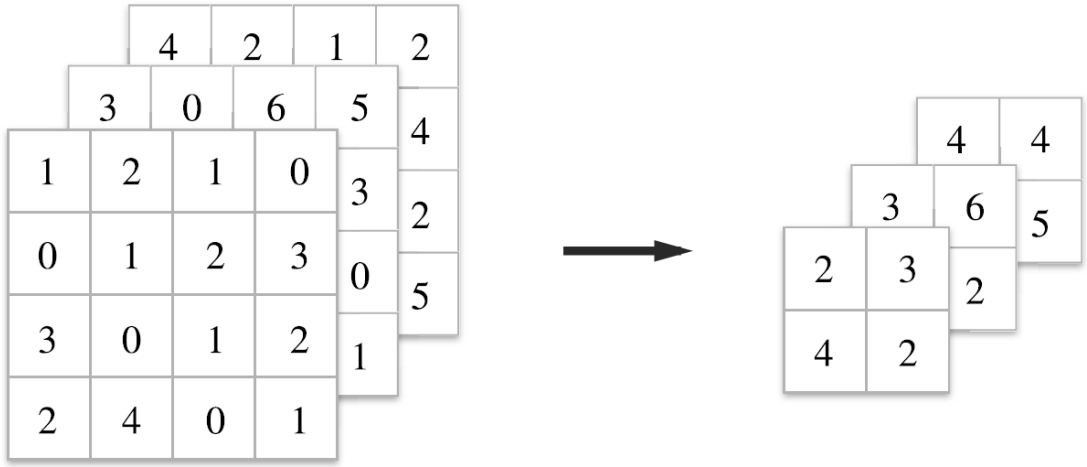
- 최대 풀링(max pooling)을 스트라이드 2로 처리하는 예



✓ 풀링 계층의 특징

- 학습해야 할 매개변수가 없다
  - 최댓값 혹은 평균을 취하는 명확한 처리하므로

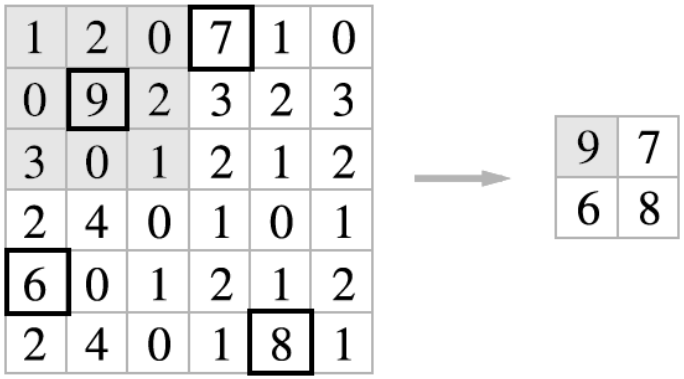
- 채널 수가 변하지 않는다
  - 채널마다 독립적으로 계산하므로

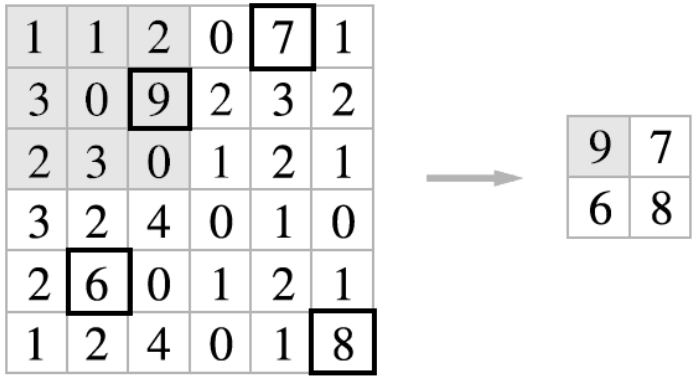


입력 데이터 출력 데이터

✓ 풀링 계층의 특징

- 입력의 변화에 강건하다(영향을 적게 받는다)
  - 입력 데이터가 조금 변해도 풀링의 결과는 잘 변하지 않는다



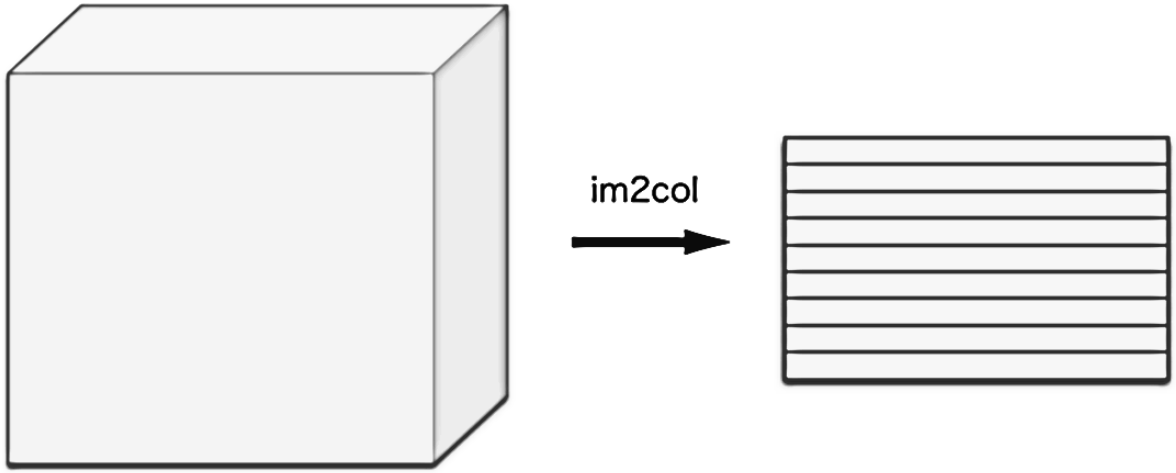


✓ 컨벌루션 계층 및 풀링 계층 구현하기

- 4차원 배열

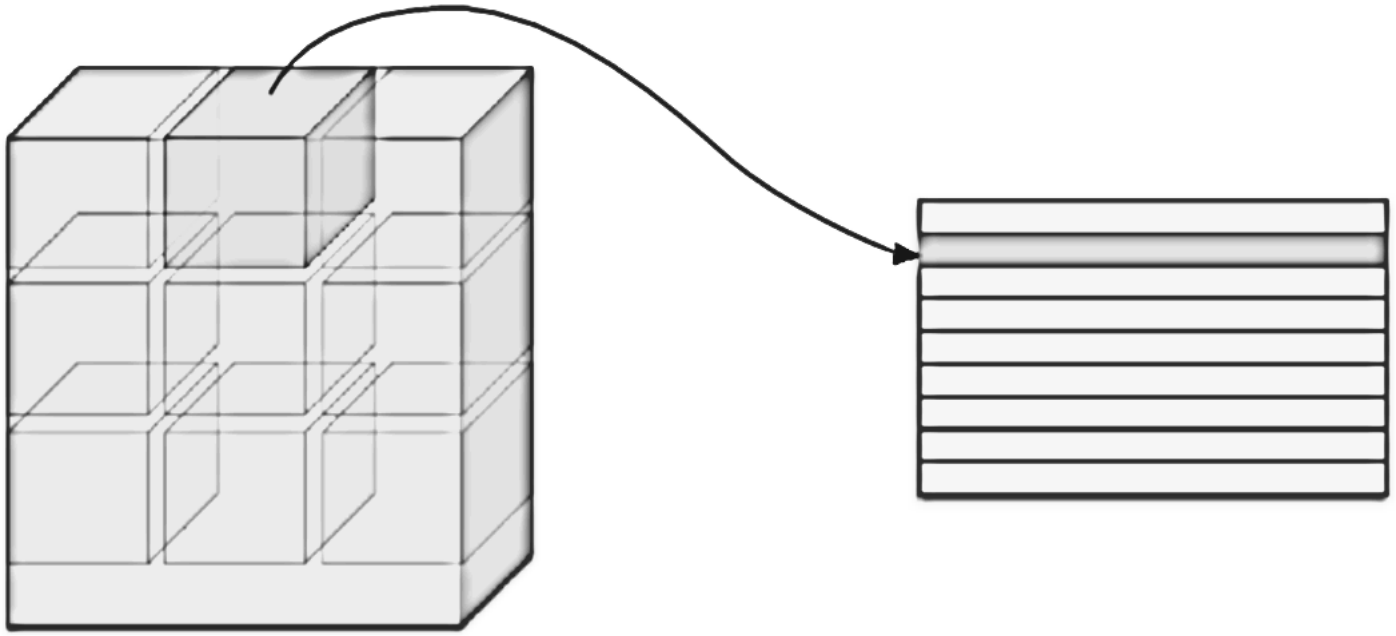
✓ Im2col 함수

- 4차원 배열의 입력 데이터를 필터링(가중치 계산)하기 좋게 전개하는 함수



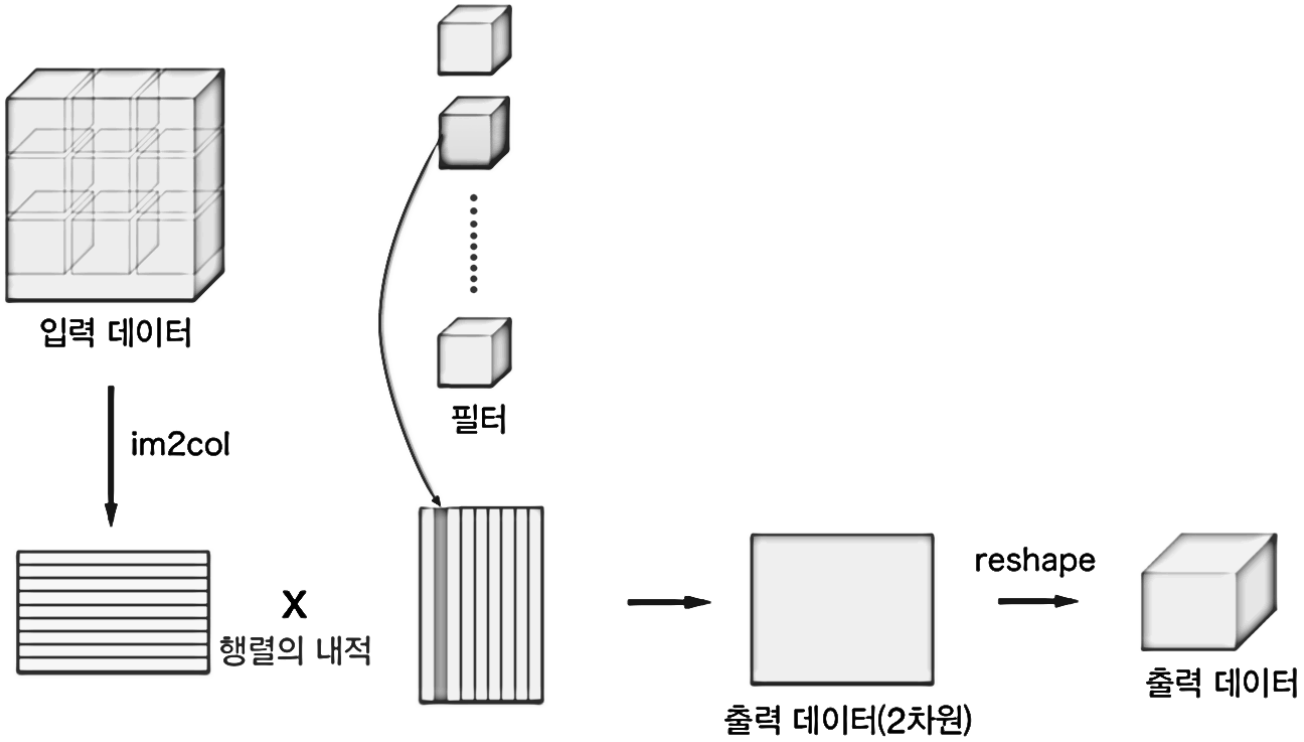
✓ Im2col 함수

- 입력 데이터에서 필터를 적용하는 영역(3차원 블록)을 한 줄로 전개



✓ 컨벌루션 연산의 필터 처리 상세 과정

- 필터를 세로로 1열로 전개
- Im2col이 전개한 데이터와 1열로 전개한 필터를 행렬 내적 연산
- 출력 데이터를 변형(reshape)



✓ 합성곱 계층 구현하기

- im2col 함수의 인터페이스
  - common/util.py

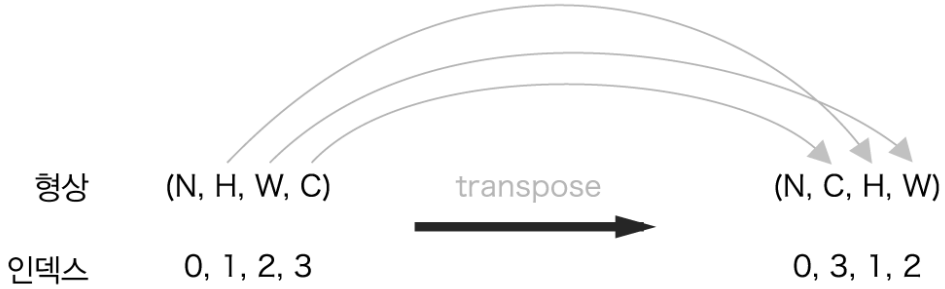
im2col(input_data, filter_h, filter_w, stride=1, pad=0)

*** input_data: (데이터 수, 채널 수, 높이, 너비)의 4차원 배열**

- 사용 예

✓ 합성곱 계층 구현하기

- common/layers.py
- Convolution 클래스
  - forward() 메소드
    - im2col 함수 이용하여 입력 데이터 전개
    - 필터 전개
    - 전개한 입력 데이터와 필터의 행렬 내적 연산
    - 출력 데이터의 형상을 재구성하고 축의 순서 변경

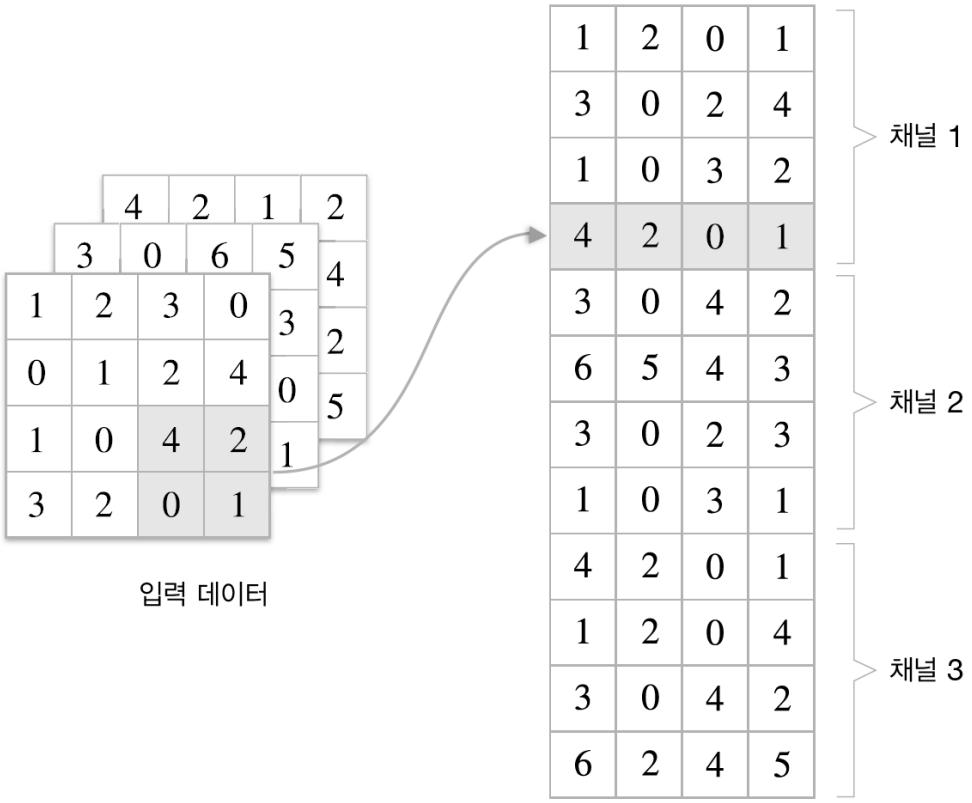


✓ 합성곱 계층 구현하기

- common/layers.py
- Convolution 클래스
  - backward() 메소드: 역전파
    - Affine 계층의 구현과 많이 유사
    - im2col 을 역으로 처리해야 함
    - col2im 함수: common/util.py

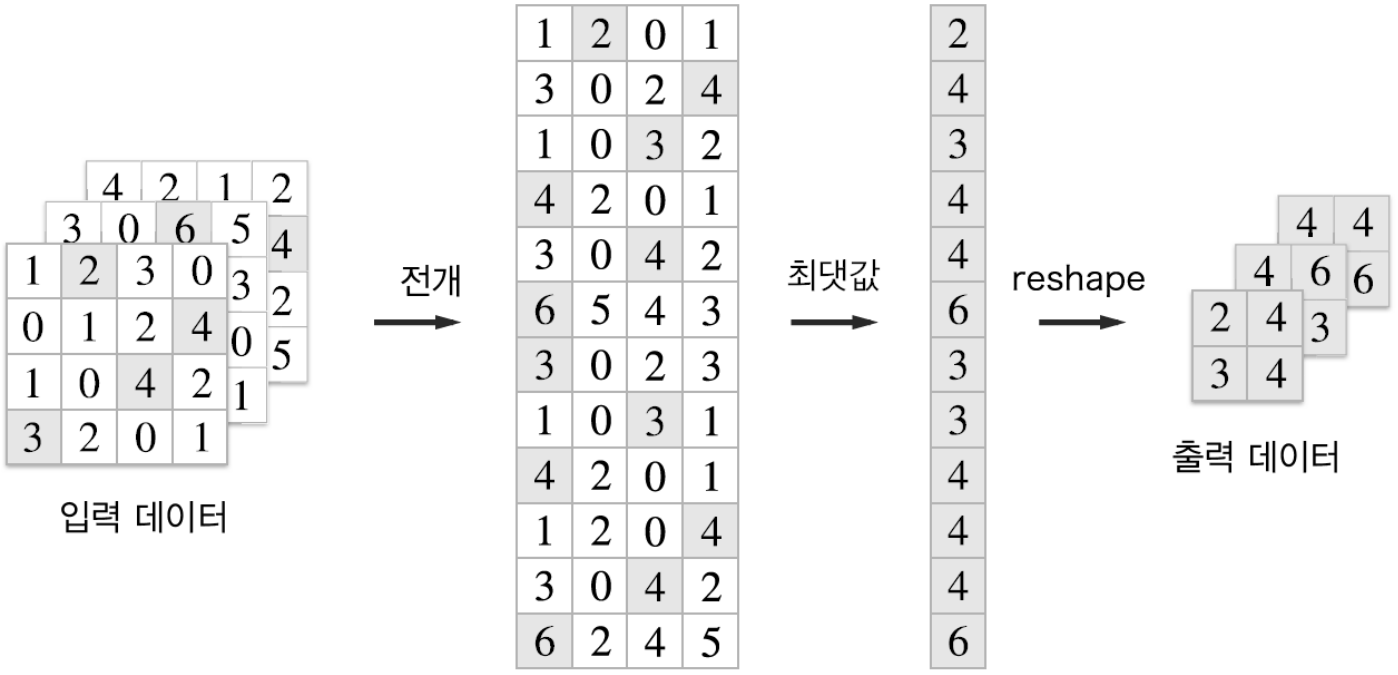
✓ 풀링 계층 구현하기

- im2col 함수를 사용해 입력 데이터 전개
- 채널이 독립인 점이 합성곱 계층과 다른 점



✓ 풀링 계층 구현하기

- 풀링 계층 구현의 흐름: 최대 풀링(가장 큰 원소는 회색으로 표시)
  - 입력 데이터 전개
  - 행별 최댓값
  - 적절한 모향으로 성형

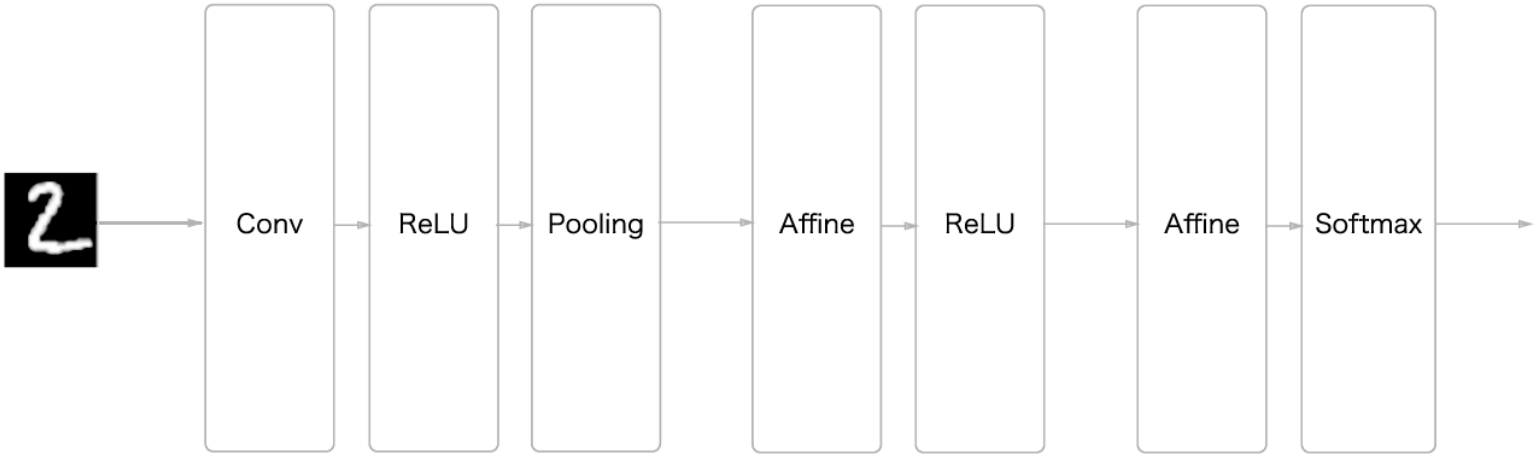


✓ 풀링 계층 구현하기

- common/layers.py
- Pooling 클래스
  - forward() 메소드
    - 입력 데이터 전개
    - 행별 최댓값
    - 적절한 모양으로 성형
  - backward() 메소드
    - ReLU 노드 구현 시 사용한 max의 역전파 참고

✓ CNN 구현하기

- 손글씨 숫자를 인식하는 CNN

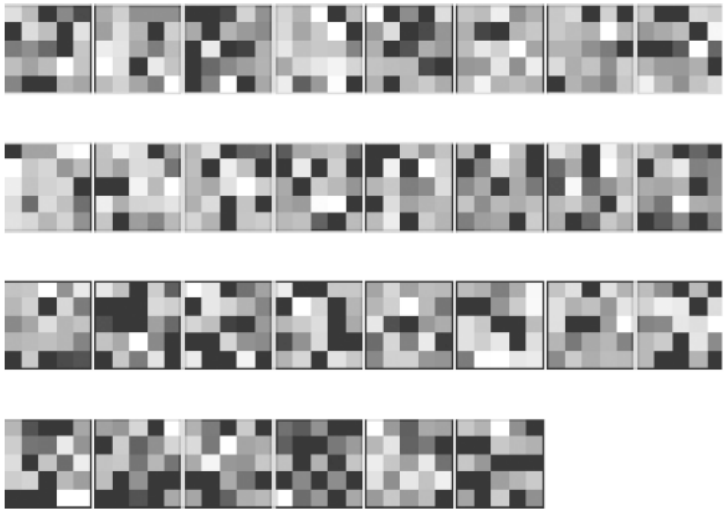


- ch07/simple_convnet.py

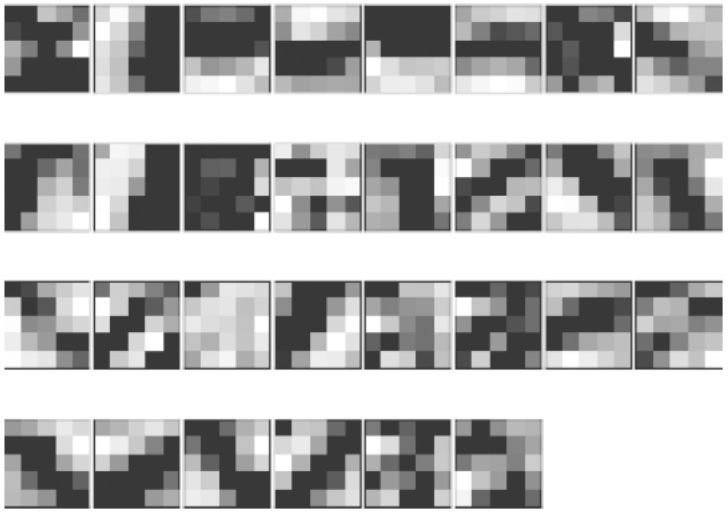
✓ CNN 시각화하기

- CNN의 필터 시각화
  - ch07/visualize_filter.py

학습 전

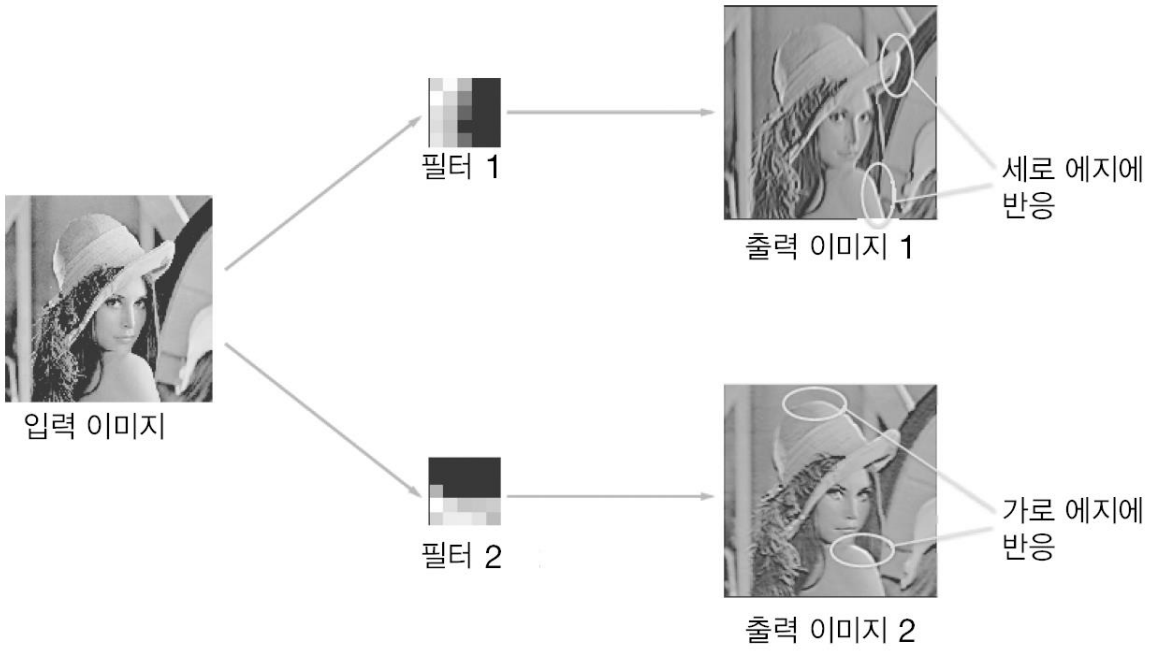


학습 후



✓ CNN 시각화하기

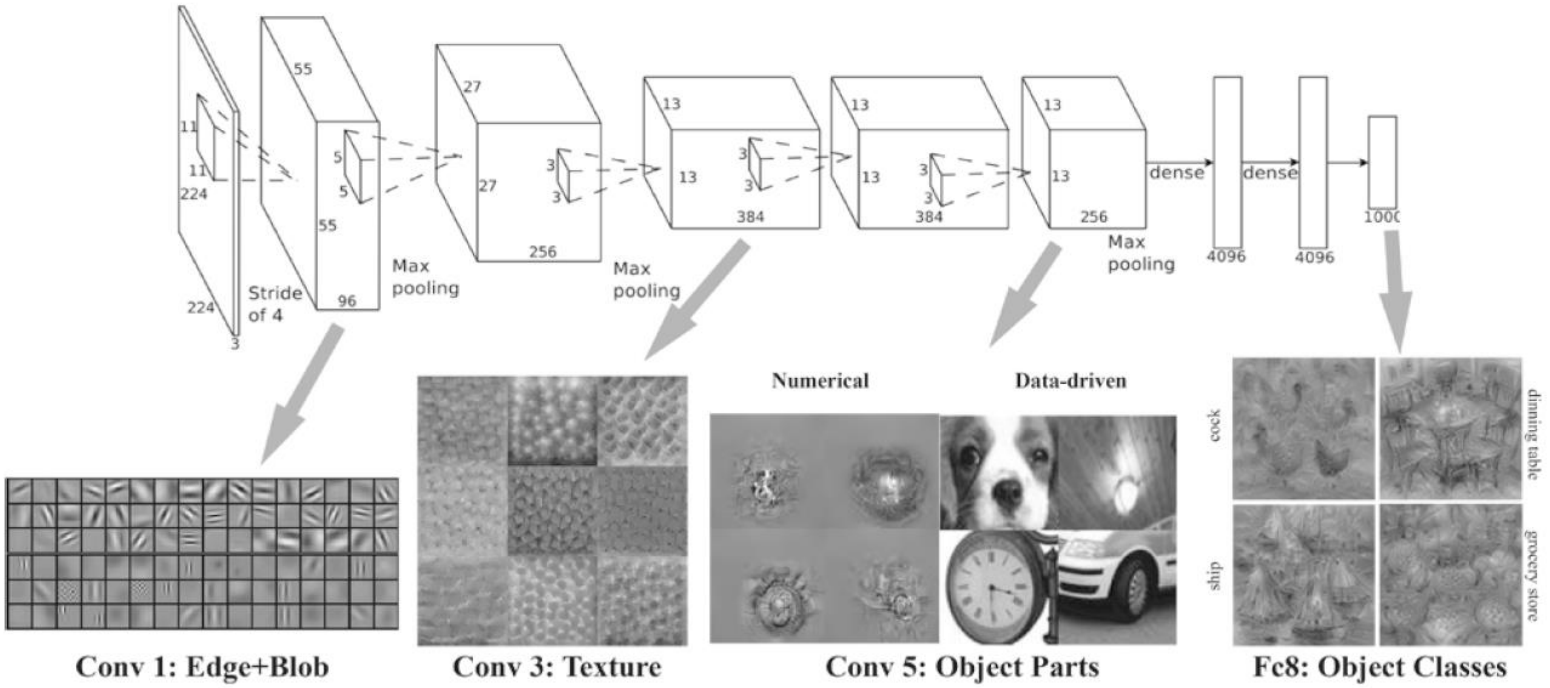
- 학습 후의 필터에 담긴 정보는 무엇일까?
  - 에지: 색상이 바뀐 경계선
  - 블롭: 국소적으로 덩어리진 영역



✓ CNN 시각화하기

- 층 깊이에 따른 추출 정보 변화
  - 층이 깊어지면서 더 복잡하고 추상화된 정보가 추출됨

- AlexNet이라는 8계층 CNN의 예



정리

- ✓ 매개변수 갱신 방법에는 확률적 경사 하강법(SGD) 외에도 모멘텀, AdaGrad, Adam 등이 있다.
- ✓ CNN은 지금까지의 완전연결 계층 네트워크에 합성곱 계층과 풀링 계층을 새로 추가한다.
- ✓ 합성곱 계층과 풀링 계층은 im2col (이미지를 행렬로 전개하는 함수)을 이용하면 간단하고 효율적으로 구현할 수 있다.
- ✓ CNN을 시각화해 보면 계층이 깊어질수록 고급 정보가 추출되는 모습을 확인할 수 있다.