PART 3

합성곱 신경망(CNN)

1장. 이론편

2장. 코딩편

딥러닝 & 강화학습 담당
이재화 강사

이 장에서 다룰 내용

1. 다층 퍼셉트론의 문제점 그리고 새로운 구조의 등장
2. '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?
3. 합성곱 연산, 패딩, 보폭, 채널
4. 효율적인 합성곱 연산 방법
5. 아담 알고리즘
6. 과소적합과 과대적합
7. 다양한 정규화 기법들

1.1 다층 퍼셉트론의 문제점 그리고 새로운 구조의 등장

CNN

Convolutional

Neural

Network

NEW Hidden Layer

합성곱 계층

풀링 계층

(param x)

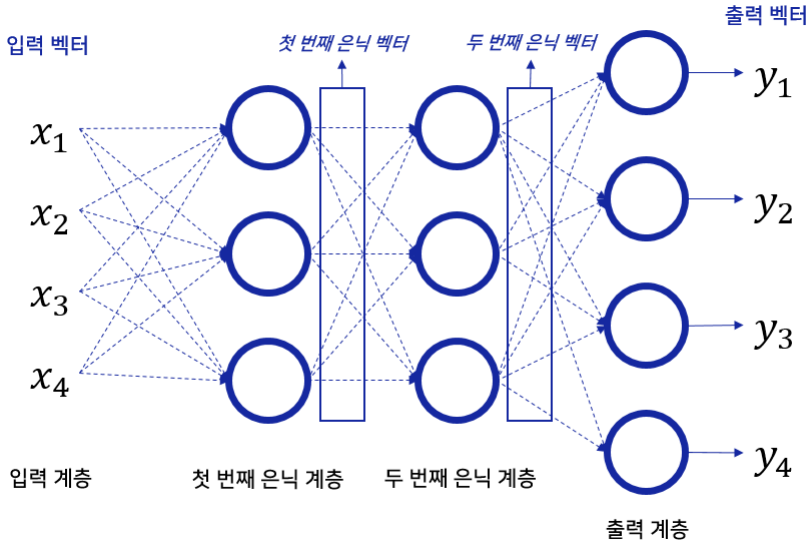
소수의 파라미터만을
집중 학습

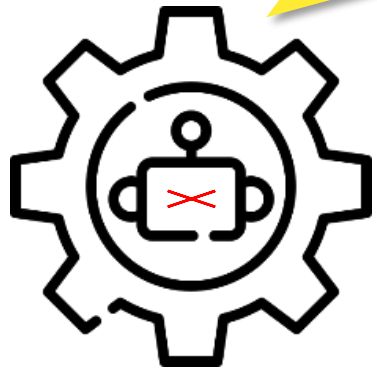
1.1 다층 퍼셉트론의 문제점 그리고 새로운 구조의 등장

- 만약 '이미지' 가 들어오게 된다면?

다층 퍼셉트론
(완전연결계층)

다층 퍼셉트론이 '마스크 이미지'를 골고루 학습하려면
온갖 위치에 등장하는 '마스크 이미지' 가 필요!





"학습도 느려지고 있어!"

"빅데이터가 필요해!"

에이림

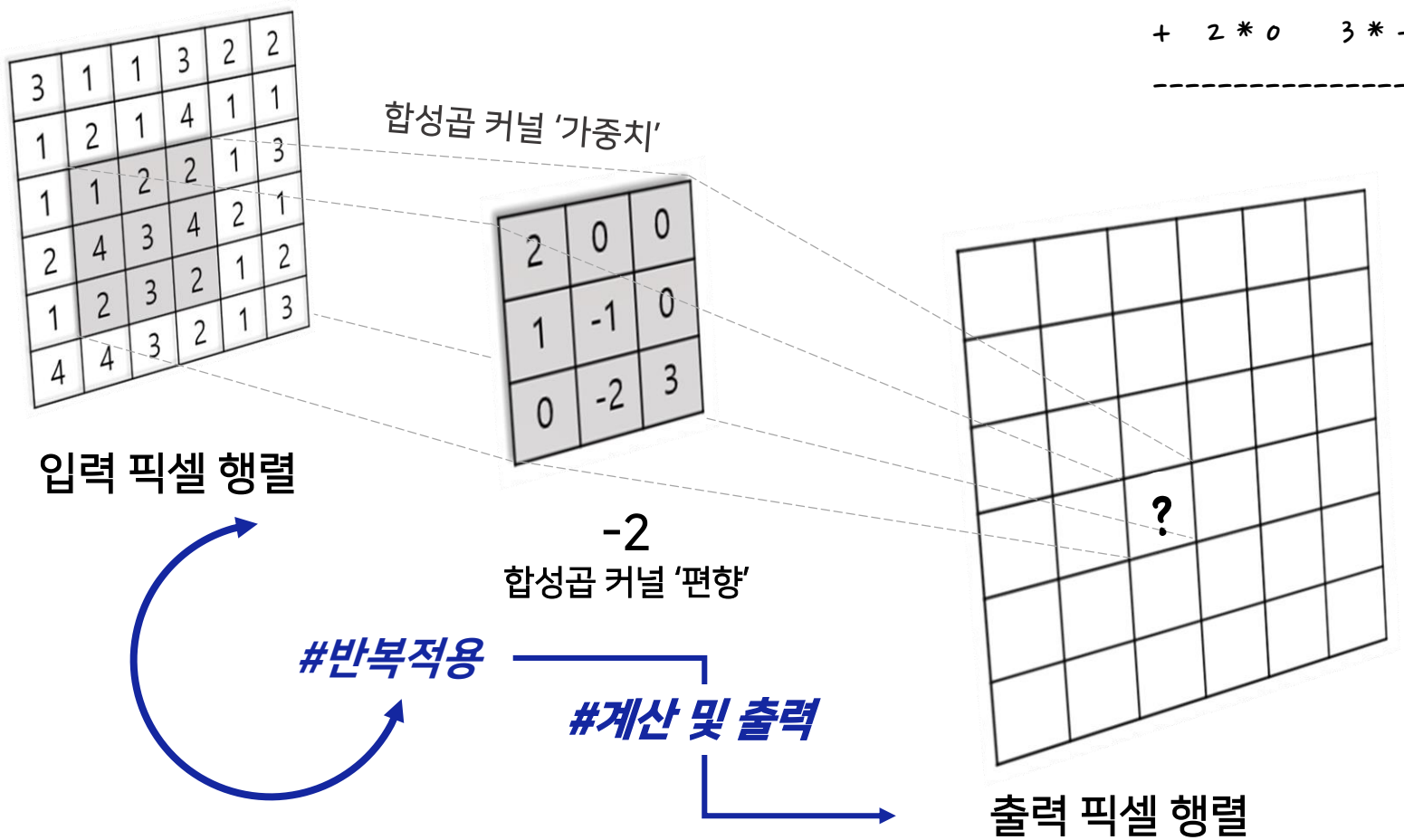
새로운 신경망 구조를 만들어 줄께!

- ✓ 엄청난 수의 픽셀
- ✓ 모든 위치의 '마스크' 에 대한 '이미지 파일' 이 필요

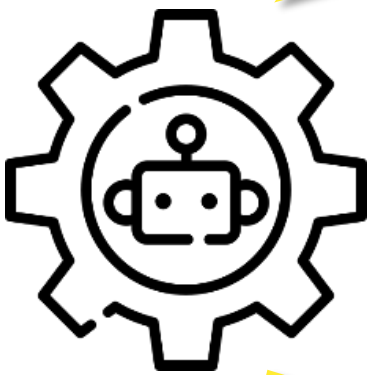
- ✓ 입력 벡터 크기 증가
- ✓ 가중치 파라미터의 크기 증가

- ✓ 합성곱 계층의 커널 활용
- ✓ 풀링 계층의 활용

1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?




```
 1*2   2*0   2*0
 4*1   3*-1  4*0
+ 2*0   3*-2  2*3
-------------------
   3  ('입력'과 '가중치'의 연산)
+ -2 ('편향')
------
   ?
```



✓ '중심 위치'는 '계산할 출력 픽셀의 위치' 이어야 합니다.

✓ '커널의 반복이용' 으로 커널에 대한 학습 횟수가 늘어나는 효과가 생겨서 학습이 빠르게 이뤄진다.

✓ 같은 커널이 모든 위치에 적용되기 때문에 '한 곳에서 포착된 패턴'이 다른 곳에 이용될 수 있다.

1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?

padding 방식의 합성곱 연산 ('편향' 생략)

0	0	0	0	0	0	0	0
0	3	1	1	2	2	2	0
0	1	2	1	4	1	1	0
0	1	1	2	2	1	3	0
0	2	4	3	4	2	1	0
0	1	2	3	2	1	2	0
0	4	4	3	2	1	3	0
0	0	0	0	0	0	0	0

입력 픽셀 행렬
(6 * 6 → 8 * 8)

2	0	0
1	-1	0
0	-2	3

합성곱 커널

```
0 * 2   0 * 0   0 * 0
0 * 1   3 * -1  1 * 0
+ 0 * 0   1 * -2  2 * 3
-------------------------
                1
```

1	1	10	-7	2	-2
0	9	5	-2	16	-2
7	3	9	0	8	-2
2	5	3	2	10	-1
3	4	7	6	16	-3
-4	2	5	7	5	3

출력 픽셀 행렬
(6 * 6)

- ✓ '0' 이라는 값으로 **이미지를 확장**해 둘러싸준다.
- ✓ **입력 픽셀 크기**와 **출력 픽셀 크기**의 값이 일정하다.
- ✓ 일반적으로 더 많이 사용되는 방식이다.

1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?

valid 방식의 합성곱 연산 ('편향' 생략)

3	1	1	2	2	2
1	2	1	4	1	1
1	1	2	2	1	3
2	4	3	4	2	1
1	2	3	2	1	2
4	4	3	2	1	3

입력 픽셀 행렬
(6 * 6)

2	0	0
1	-1	0
0	-2	3

합성곱 커널

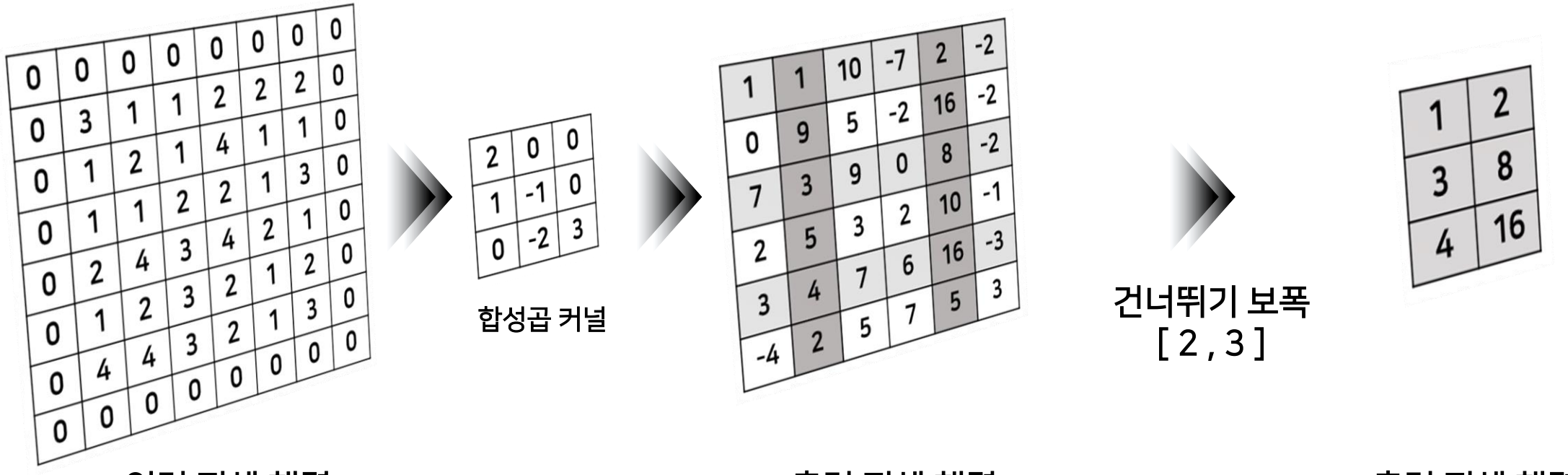
9	5	-2	16
3	9	0	8
5	3	2	10
4	7	6	16

출력 픽셀 행렬
(6 * 6 → 4 * 4)

- ✓ '커널'이 실제 입력 범위를 벗어나지 않을 때만 '출력 픽셀'을 생성
- ✓ 출력 이미지의 크기가 입력 이미지 보다 줄어들게 된다.

1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?

건너뛰기에 의한 합성곱 출력 생성 방식 ('편향' 생략)



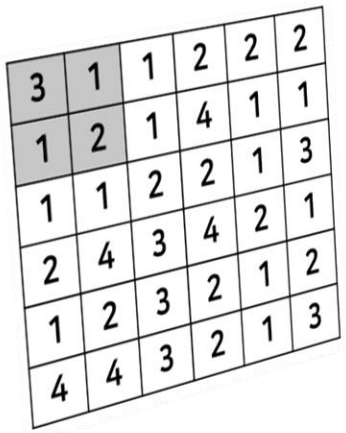
입력 픽셀 행렬
(6 * 6 → 8 * 8)

출력 픽셀 행렬
(6 * 6)

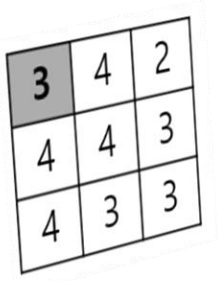
출력 픽셀 행렬
(3 * 2)

- ✓ 이미지 크기를 줄여주기에 **처리 부담을 감소**
- ✓ 너무 많은 **정보를 무시**해버림

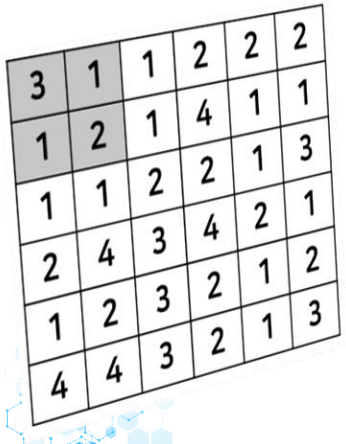
1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?



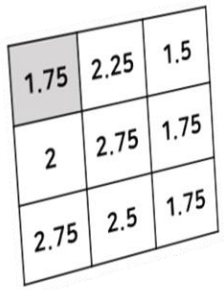
2 * 2
최대치 풀링

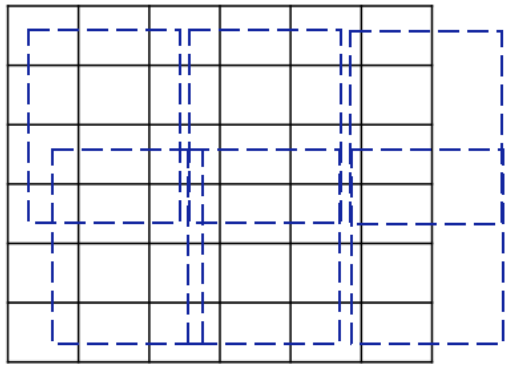


'최대치' 풀링 처리
'평균치' 풀링 처리



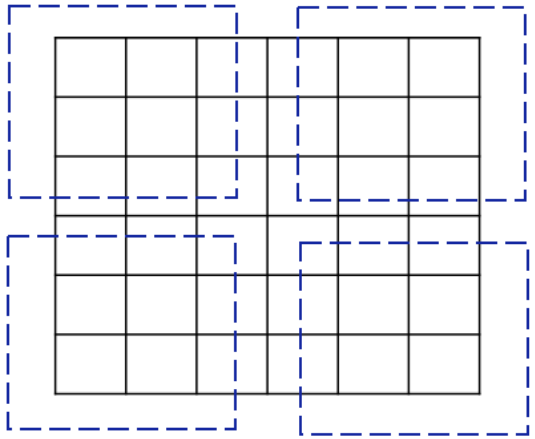
2 * 2
평균치 풀링





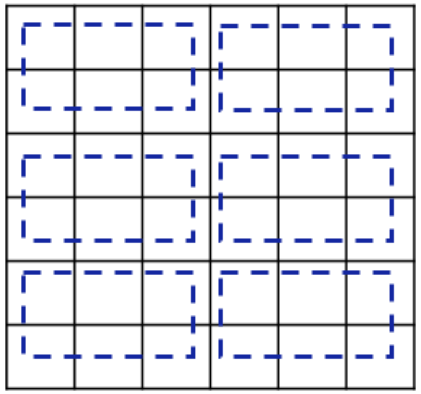
A image_size = 6 * 6
kernel_size = 4 * 3
stride = 2 * 2

 # '효율성'이 다소 떨어진다.



B image_size = 6 * 6
kernel_size = 4 * 4
stride = 4 * 4

 # 조심스러운 '처리'가 필요



C image_size = 6 * 6
kernel_size = 2 * 3
stride = 2 * 3

 # 이미지 크기가 보폭의 배수
 # 입력 픽셀의 자투리 없이 분할 가능

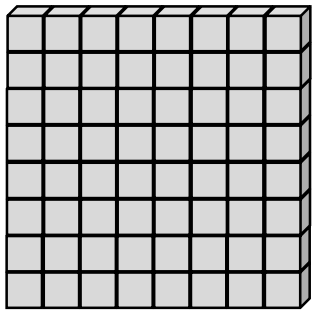
1.2 '합성곱 계층'과 '풀링 계층'이란 어떠한 것이죠?

풀링 연산

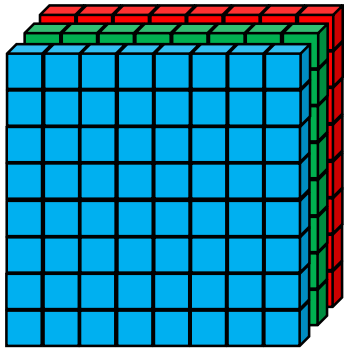
- ✓ **입력만으로 대표값**을 구하는 과정
  ('파라미터' 필요 없음)
- ✓ 합성곱 계층 사이에 배치되어 점점 작은 해상도의 특징맵에 **정보**가 **'집약'**되게 만드는데 이용
- ✓ 풀링 계층의 처리는 합성곱 계층이 **넓은 영역의 정보**를 **대표값으로 대체**하면서 처리

Part 3. 합성곱 모델

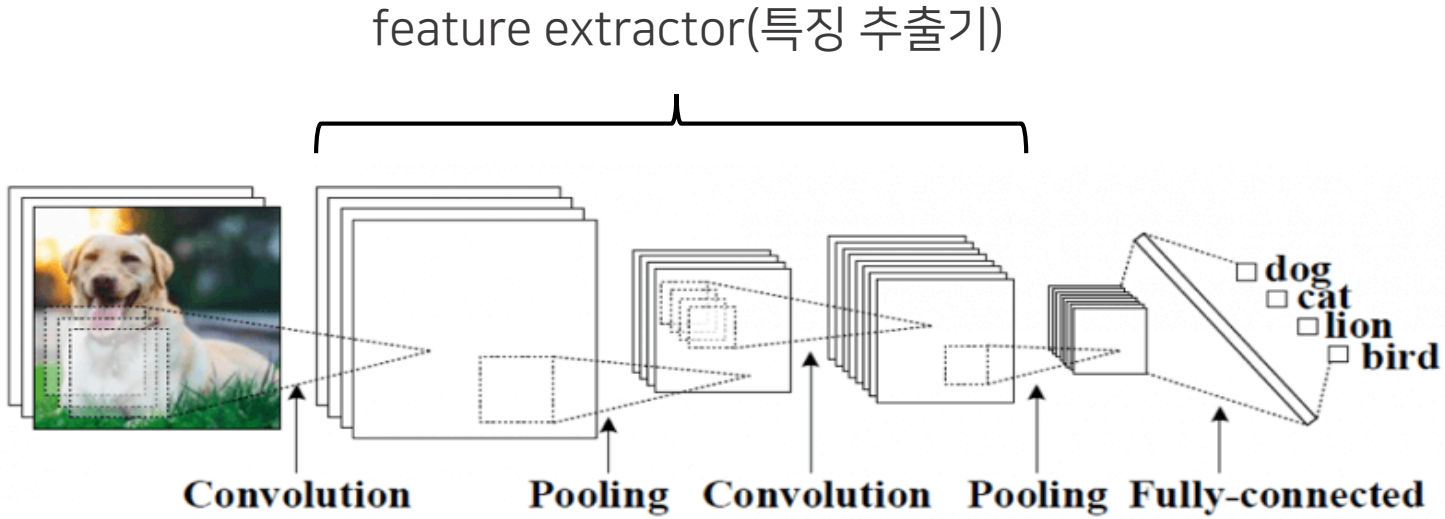
1.4 채널의 도입과 커널의 확장



A. 흑백이미지



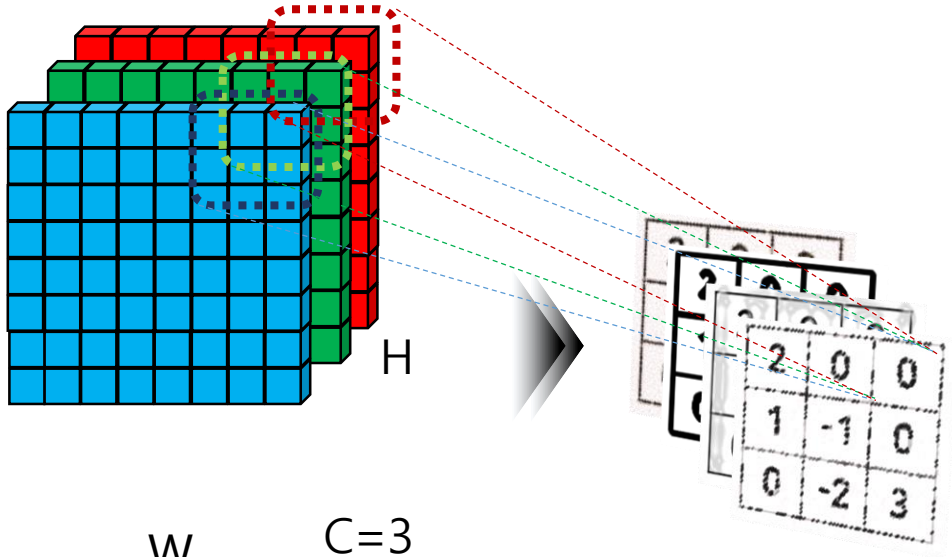
B. 컬러이미지



✓ 필터 혹은 커널 : 입력 영역의 **특정 패턴**이나 **특징**에 민감하게 반응하는 경향으로**학습되는 경향**

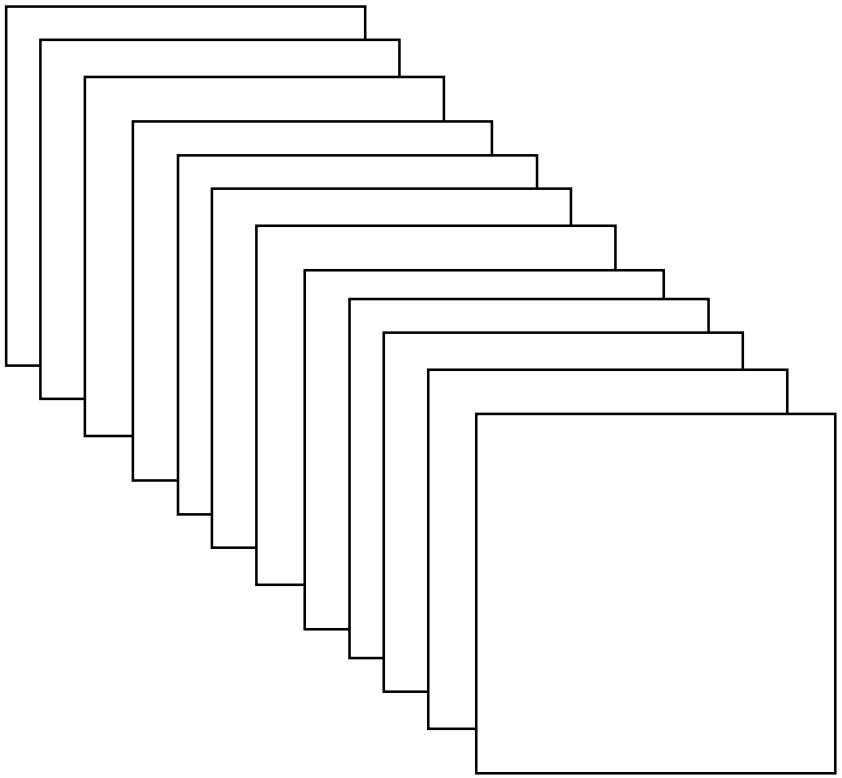
✓ 합성곱 계층의 **입력 채널 수**는 **'최초의 이미지'** 혹은 **'이전 계층'**이 생성한 **'출력 채널 수'**로 정해지지만, **출력 채널 수**는 신경망 설계자가 **자유롭게 정할 수 있음**.

1.4 채널의 도입과 커널의 확장



컬러이미지 [H,W,3]

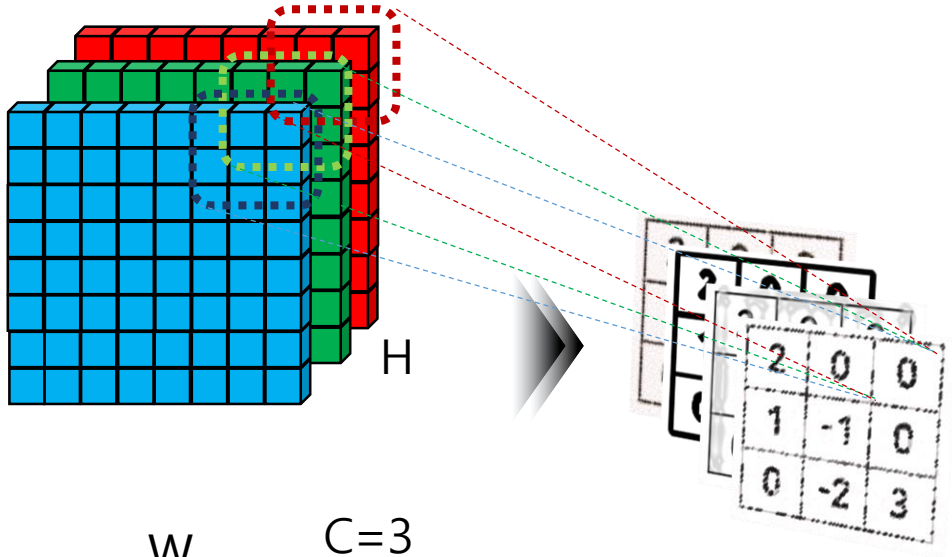
『feature map』



Q. '커널'은 어떤 모양이 되어야 할까요?

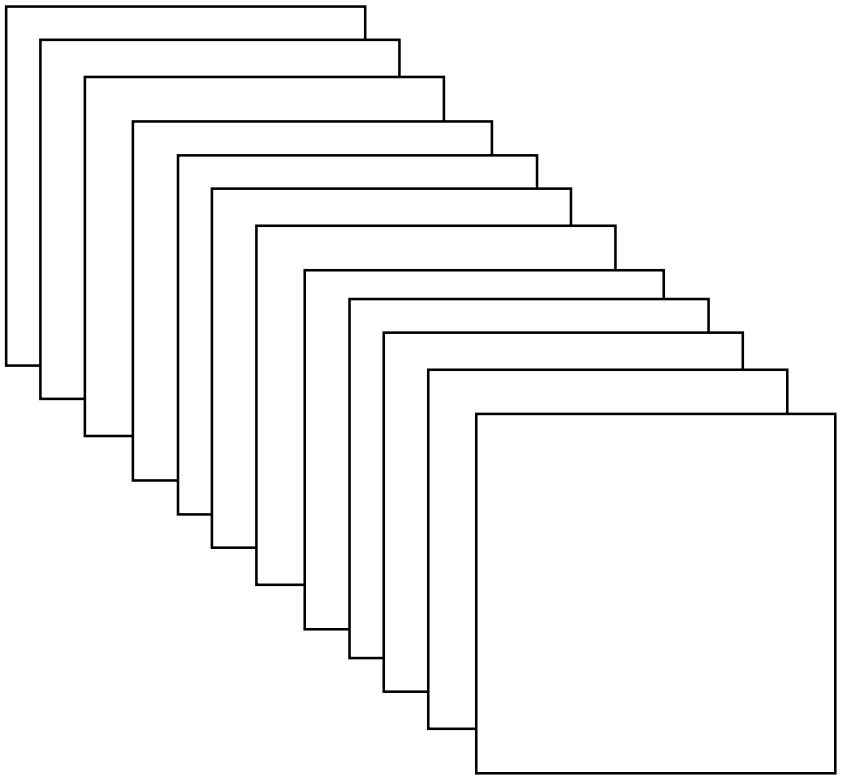
- ✓ 이미지의 작은 사각 영역에서 하나의 특징을 온전히 포착하려면 '커널'은 해당 영역의 **모든 채널에 대한 입력 픽셀값을 볼 수 있어야 한다.**
  즉, 하나의 특징을 포착하는 **'커널'**은 사각 영역 입력의 모든 채널을 처리하는 **3차원 가중치**를 가져야 합니다.
- ✓ **출력 채널 수 만큼의 특징맵**을 만들어 내기 위해서는 **커널이 출력 채널 수 만큼 있어야 합니다.**
  '커널'은 입력 영역의 크기에 입력 채널과 출력 채널이 모두 반영된 **4차원 구조**가 되어야 함.

1.4 채널의 도입과 커널의 확장



컬러이미지 [H,W,3]

『feature map』



Q. '합성곱 계층의 입출력 데이터' 는 어떤 모양이 되어야 할까요?

✓ 2차원 해상도(H,W)에 채널(C)이 추가, 미니배치 단위의 학습 때문에 4차원 구조.

즉, 합성곱 계층의 처리는 아래와 같다.

4차원 입력 - [미니배치 크기, 입력 이미지 행 수, 입력 이미지 열 수, 입력 채널 수]

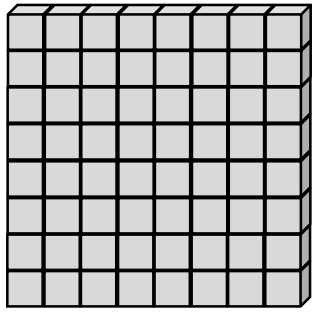
4차원 커널 - [커널 행 수, 커널 열 수, 입력 채널 수, 출력 채널 수]

4차원 출력 - [미니배치 크기, 출력 이미지 행 수, 출력 이미지 열 수, 출력 채널 수]

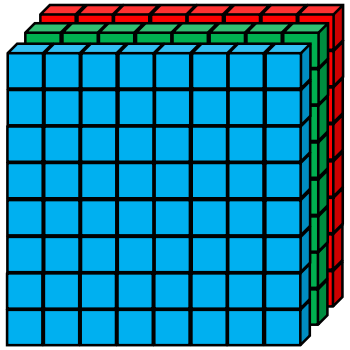
Part 3. 합성곱 모델

1.4 채널의 도입과 커널의 확장

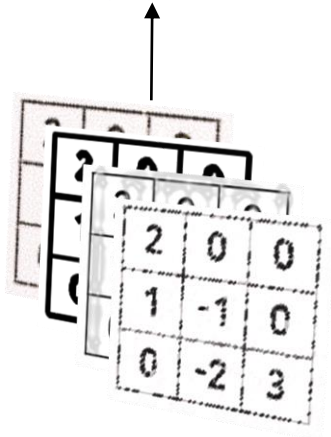
 # 다양한 특징과 패턴



A. 흑백이미지 [H,W,1]

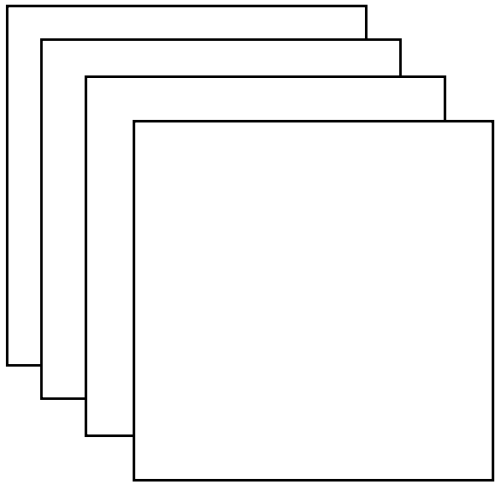


B. 컬러이미지 [H,W,3]

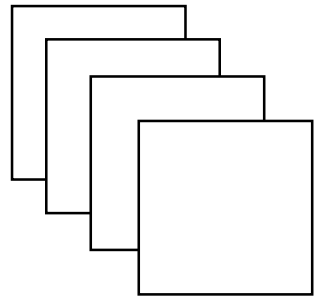


*입력 영역의 **특정 패턴**이나*
__특징__에 민감하게 반응하는 경향으로
학습되는 경향

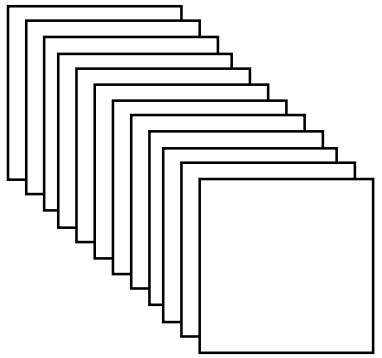
『feature map』



『feature map』



『feature map』



1.4 채널의 도입과 커널의 확장

#4차원 정보

4차원 입력 - [미니배치 크기, 입력 이미지 행 수, 입력 이미지 열 수, 입력 채널 수]
4차원 커널 - [커널 행 수, 커널 열 수, 입력 채널 수, 출력 채널 수]
4차원 출력 - [미니배치 크기, 출력 이미지 행 수, 출력 이미지 열 수, 출력 채널 수]

2차원 그림으로 표현?

수식으로 표현!

✓ 입력 형태 $[mb, xh, xw, xchn]$

✓ 커널 형태 $[kh, kw, xchn, ychn]$

✓ 출력 형태 $[mb, yh, yw, ychn]$

✓ $same\ padding$
✓ $stride = [1,1]$

✓ 입력과 출력의 크기 동일

✓ $yh = xh$
✓ $yw = xw$

 # 출력 형태

$$[mb, xh, xw, ychn]$$

1.4 채널의 도입과 커널의 확장

Q. '채널 개념'이 추가되었을때 '풀링 계층'은
어떻게 변해야 하는걸까요?

A. 기본적으로 풀링은 **이미지 해상도를 줄이기 위해 적용**

A. 해상도를 점점 추려가면서 즉 작게만들어 가면서
특징맵을 점점 작게 만드는 것 또한 매우 흔한 일.

A. 여러 채널의 특징맵들이 표현하는 서로 다른
도메인의 특징값들에 대해 **평균이나 최대치**를 구하는
일은 **직관적으로 그리 유용하지 않음**.
(※ 특별히 성과를 거둔 연구도 찾아보기 어렵)

➢ 풀링 층을 거쳐도 대개 **이미지 해상도가**
줄어들기만 할 뿐 채널 수는 **변하지 않고 그대로 유지**

1.5 합성곱과 풀링의 역전파 처리

✓ **합성곱 계층의 역전파 처리 방법**은 원리상 **완전연결 계층에서의 역전파 방식**과 비슷

순전파

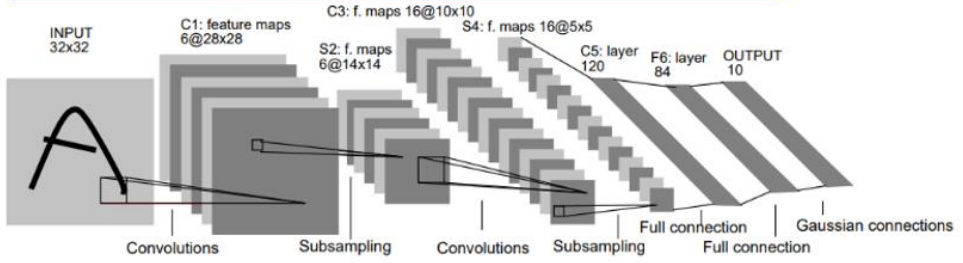


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

역전파

✓ 입력 픽셀과 커널 가중치를 짝지어 곱하기.

✓ 커널 가중치와 입력 픽셀은 서로가 서로의 계수로서 편미분값이 된다.

출력 성분의 손실 기울기를 곱셈.
입력 픽셀과 **커널 가중치**의 해당 출력 성분에 대한
'손실 기울기'를 구할 수 있음

1.5 합성곱과 풀링의 역전파 처리

✓ 4차원 입력과 4차원 커널로부터
4차원 출력을 얻어내는 **매우 복잡한 과정.**

✓ 입력 픽셀과 커널 가중치는 **여러 차례 반복적으로 이용**되면서
많은 출력 성분에 영향을 미친다.

✓ 이론상 입력 픽셀과 커널 가중치의 **손실 기울기**는
영향을 미친 모든 출력 성분으로 부터 부분적인
손실 기울기를 구해 합산해야 구할 수 있게 된다.

『완전연결계층』과 비슷!?!

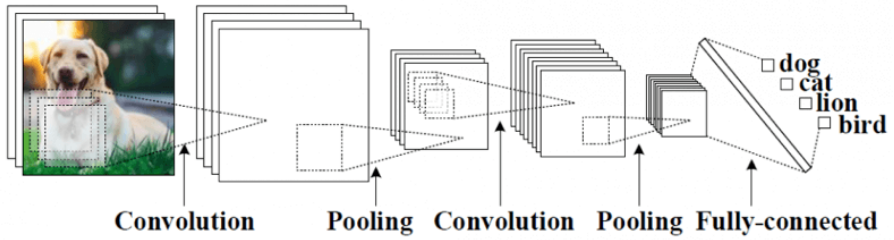
✓ 하나의 입력 성분은 **다수의 가중치에 곱**해지게 됨

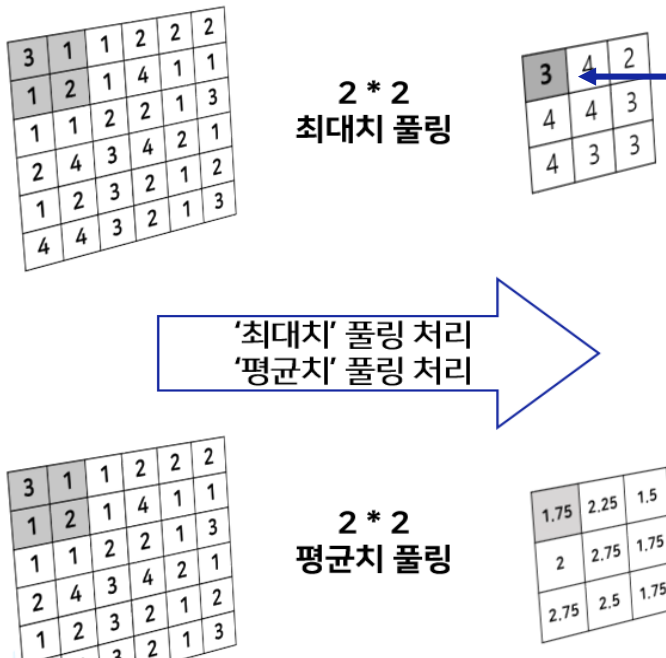
✓ 가중치 역시 다수의 입력성분에 곱해지게 되면서,
역전파에서는 영향을 미친 모든 출력 성분으로 부터
손실 기울기를 구해 합산하는 과정이 **행렬 연산에 가려져 실행.**

1.5 합성곱과 풀링의 역전파 처리

#풀링의 역전파?!

✓ 풀링 계층은 파라미터를 갖고있지 않다.





✓ '출력'에 반영되는것은 **커널 영역의 최댓값.**

따라서 역전파 처리 과정에서는 최대치로 선정된 위치의 입력만

출력의 손실 기울기를 전달하고 **나머지 입력에 대해서는 손실기울기를 0으로 지정**.

✓ 만약 같은 **최대값이 동시에 발생한 경우** 최댓값 중 하나를 임의로 정해

출력의 손실기울기를 그 원소에 몰아주는 방법이 가장 많이 쓰이고 있습니다.

✓ **평균 풀링**의 경우는 모든 입력 원소가 균등하게 출력에 반영되기 때문에,

출력의 손실기울기를 입력 원소들에게 **균등하게 1/N 해서 나눠주면** 쉽게 해결

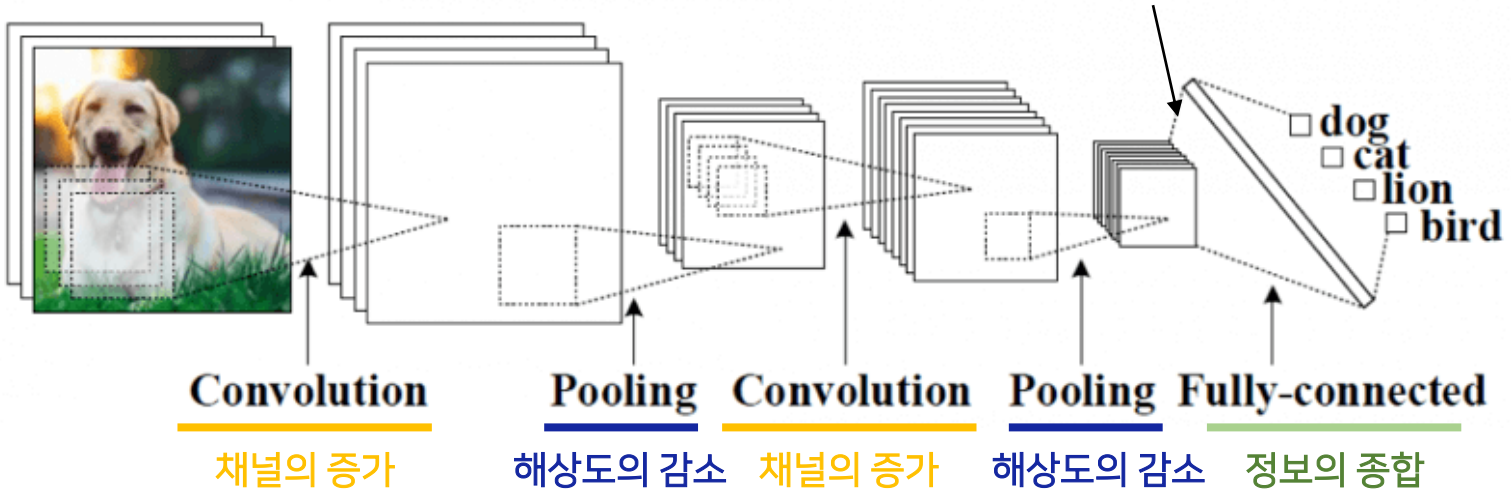
따라서, 역전파 과정에서도 출력의 손실기울기에 1/N 값을 곱해,

각 입력 성분의 손실기울기로 전달.

1.6 합성곱 신경망의 일반적인 구성

4차원 텐서 형태 → 2차원 행렬

(미니배치 데이터 축을 기준)

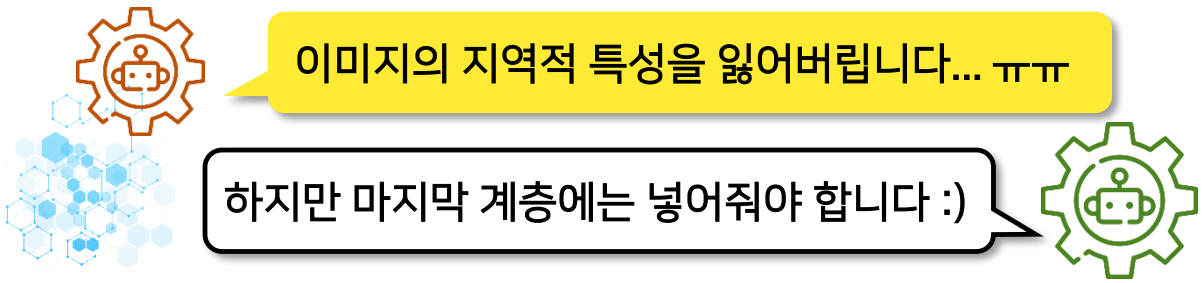


(※ 여러 채널에 흩어져 있는 정보를 종합)

✓ **합성곱 계층은** 이미지 해상도를 유지하면서도 채널 수를 늘리는 역할

✓ **풀링 계층**은 채널 수를 유지하면서 이미지 해상도를 줄이는 역할

Q. 만약 중간에 '**완전연결계층**'을 넣으면 어떻게 되나요?



1.7 아담 알고리즘

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba*
University of Toronto
jimmy@psi.utoronto.ca

ABSTRACT

We introduce *Adam*, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which *Adam* was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss *AdaMax*, a variant of *Adam* based on the infinity norm.

1 INTRODUCTION

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. Many problems in these fields can be cast as the optimization of some scalar parameterized objective function requiring maximization or minimization with respect to its parameters. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function. Often, objective functions are stochastic. For example, many objective functions are composed of a sum of subfunctions evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual subfunctions, i.e. stochastic gradient descent (SGD) or ascent. SGD proved itself as an efficient and effective optimization method that was central in many machine learning success stories, such as recent advances in deep learning (Deng et al., 2013; Krizhevsky et al., 2012; Hinton & Salakhutdinov, 2006; Hinton et al., 2012a; Graves et al., 2013). Objectives may also have other sources of noise than data subsampling, such as dropout (Hinton et al., 2012b) regularization. For all such noisy objectives, efficient stochastic optimization techniques are required. The focus of this paper is on the optimization of stochastic objectives with high-dimensional parameters spaces. In these cases, higher-order optimization methods are ill-suited, and discussion in this paper will be restricted to first-order methods.

We propose *Adam*, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients; the name *Adam* is derived from adaptive moment estimation. Our method is designed to combine the advantages of two recently popular methods: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman & Hinton, 2012), which works well in on-line and non-stationary settings; important connections to these and other stochastic optimization methods are clarified in section 5. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing.

*Equal contribution. Author ordering determined by coin flip over a Google Hangout.

#아담 알고리즘 한줄 요약:

*파라미터에 적용되는 실질적인 **'학습률'**을 **개별 파라미터 별로 동적으로 조절**해 경사하강법의 동작을 보완하고 **학습 품질을 높여주는 방법.***

1.7 아담 알고리즘

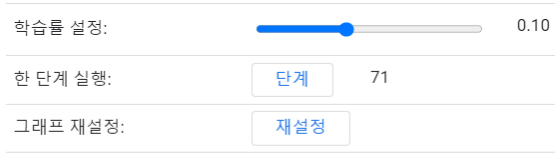
Adaptive moments

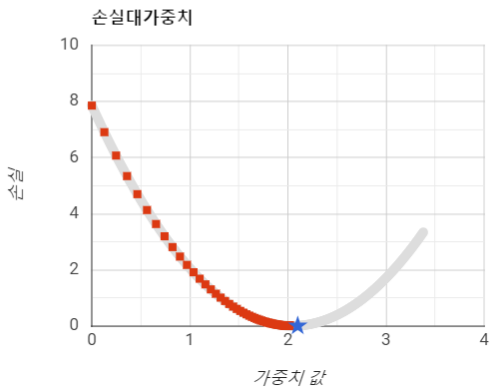
모멘텀 - 관성력
(Moments)

즉, 최근의 파라미터값의 변화 추세를 나타내는 정보.

아담 알고리즘은 이런 모멘텀 정보와 함께 2차 모멘텀 정보까지 활용.

#경사 하강법





역전파 처리 때 해당 파라미터의 손실 기울기와 학습률을 곱한 값을

각 파라미터에서 빼주는 간단한 방식으로 학습을 수행

1.7 아담 알고리즘

Adaptive moments

✓ 모멘텀 정보와 2차 모멘텀 정보가 **전체적인 신경망 차원에서 관리되는 단일한 값이 아니라는 것.**

✓ 이 값은 개별 파라미터 수준에서 따로 계산되고 관리,
즉, 아담 알고리즘을 이용할 경우 파라미터 하나마다 모멘텀 정보와 2차 모멘텀 정보가 따라붙게 되어
파라미터 관리에 필요한 메모리 소비량이 대략 3배에서 4배까지 증가.

✓ **미니 배치 데이터를 처리해 학습이 이뤄질 때마다 파라미터 값들은 물론 모멘텀 정보도 다시 계산**
즉, 계산 부담 증가.

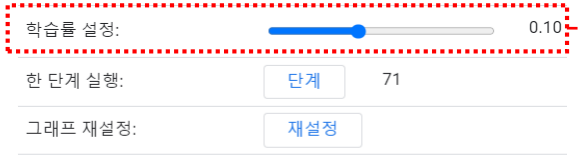
✓ 전체적으로 메모리 소비와 계산량이 증가하는 것은 피할 수 없다.

강력한 만큼 조건들도 만만치 않네!!

1.7 아담 알고리즘

Adaptive moments

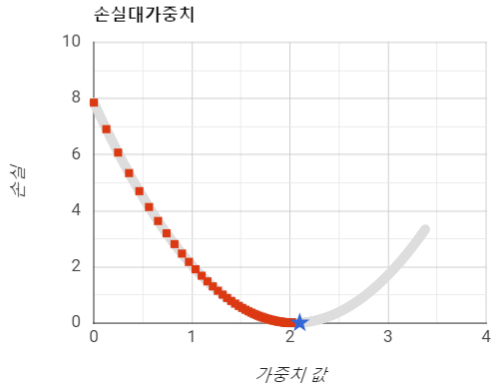
#경사 하강법



파라미터별로 실질적인 학습률을 보정해 적용, 품질 저하 방지

파라미터를 갱신하는 '마지막 순간'에 학습률이 이용되게 됩니다.

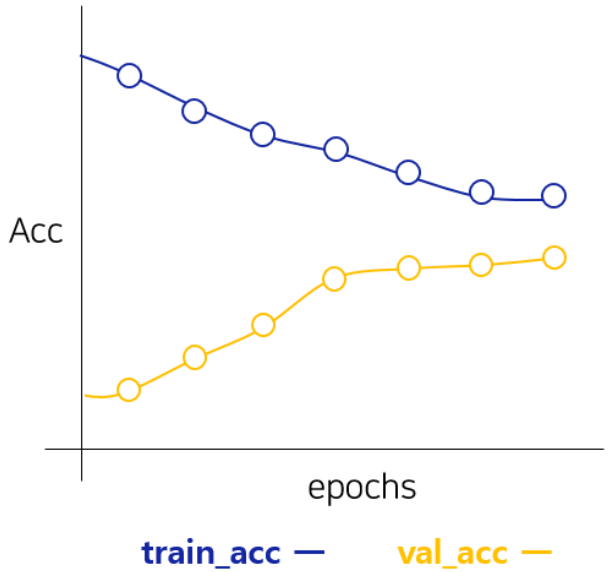
그래서 우리가 학습률을 찾는 노력이 완전히 사라진것은 아님.



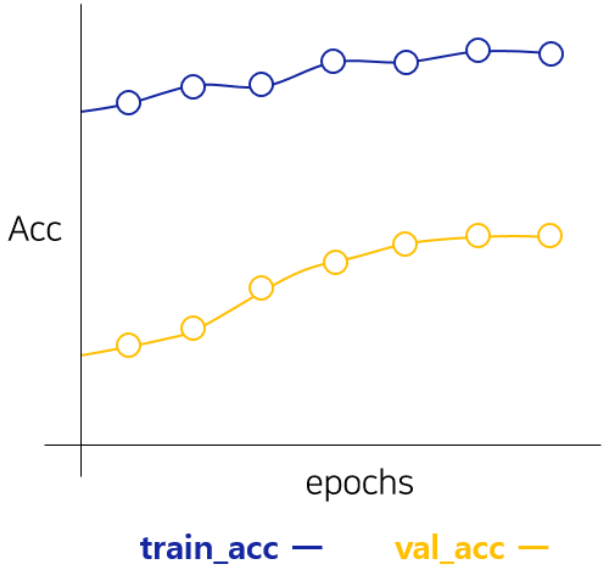
"초기 학습률 … 학습률 … 학습률…"

1.8 과소적합과 과대적합

#과소적합



#과대적합



문제점

- ✓ 문제의 난이도에 비해 **모델의 수용 용량이 부족할 때** 주로 발생.
- ✓ 세대가 지남에 따라 **학습 데이터의 정확도**가 개선되지 않고 **오히려 악화**되고 있다.
- ✓ 검증 데이터와의 **폭이 작다.**

- ✓ 전체적인 문제 특성을 파악하지 못한 채, 학습 데이터 자체의 **특성을 외움.**
- ✓ 학습 단계와 비교했을때 **정확도가 크게 떨어지는 모습을 보여줌.**
- ✓ 과적합은 문제의 난이도에 비해 **데이터가 부족할 때 주로 발생.**

해결방안

- ➢ 모델이 감당할 수 있는 **용량 자체를 키워주기.**
- ➢ **학습 횟수를 늘리기**
- ➢ **데이터 양 자체를 늘려** 품질을 높이는 등의 조치 필요.

- ✓ 가장 이상적인 방식은 **양질의 데이터를 충분히 더 확보**하는 것.
- ✓ **정규화 기법**의 활용

1.9 다양한 정규화 기법들

- 정규화 기법은 성능 향상을 위한 것
- 신경망 개발 과정에서는 평가 단계가 정규화 단계에 대한 모의 실험 역할을 수행

L2 | L1 | Noise injection | Drop out | Batch normalization

정규화란?

빠르게 달려야 하는 육상 선수를 훈련시킬 때 일부러 모래주머니를 차게 하는 것처럼

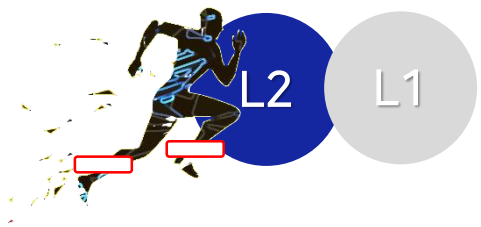
딥러닝 알고리즘에 **일부러 학습을 방해하는 제약**을 가해,

더 많은 실력을 쌓도록 유도하는 방법론

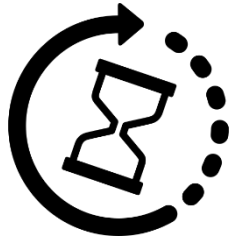
1.9 다양한 정규화 기법들

A $L' = L + \frac{1}{2}\lambda \sum_{i=1}^{n} w_i^2$

정규화항을 추가하여 복잡한 모델에서 손실함수가 최솟값인 경우에도 **과적합**을 막는다.



B $P_{L2} = \frac{1}{2}\lambda \sum_{i=1}^{n} w_i^2$

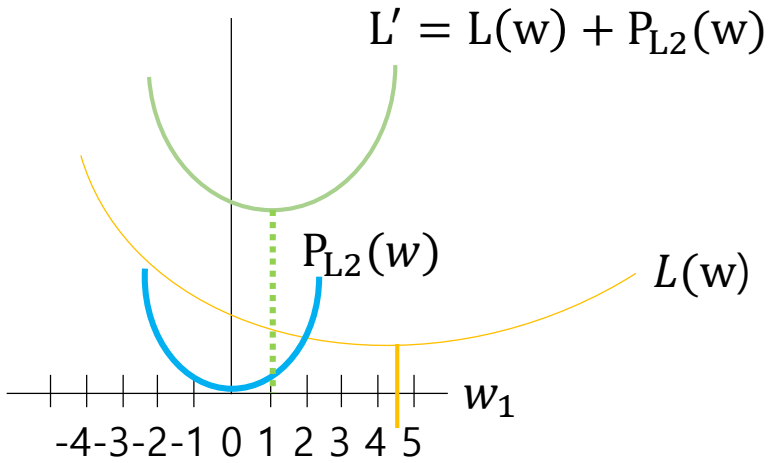


학습중...

가중치 파라미터 절댓값 중 일부 혹은 전부가 과도하게 커지는 방향으로 나아가는 경우 발생

- ✓ L2 손실, L1 손실은 **모두 절댓값이 큰 파라미터에 대해 불이익**을 할당.
- ✓ 값의 폭주를 막고 **작은 절댓값의 파라미터들로 문제를 풀도록 압박**
- ✓ 단, 이때 **편향**은 **일반적으로 포함하지 않는다.**
  편향을 적용하지 않았을 때, 더 L2 효과가 향상

※ λ(lambda) : 정칙화율, 손실율, ...
정규화 항의 영향을 정하는 양의 상수

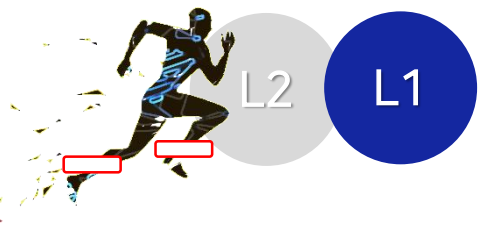


C *#임의의 손실함수 그래프*
최솟값은 4.5 부근

D *# 람다값 제외하고 정칙화항 그래프*
#$\frac{1}{2}w_1^2$ 의 모양인 0점을 지나는 단순 2차함수

E *#두 함수를 더한 그래프가 그려진 것을 확인할 수 있고,*
#일반 손실함수의 최소값보다 훨씬 작은 최소값을 구할 수가 있음.
#가중치가 너무 커지게 되는것을 방지해서 작은 값에 가까워 지게 함.

1.9 다양한 정규화 기법들



$$L' = L + \alpha \sum_{i=1}^{n} |w_i|$$

$$P_{L1} = \alpha \sum_{i=1}^{n} |w_i|$$

✓ L1은 일률적으로 정해진 값을 덜어내는 방법

파라미터가 작은값이라면은 0 혹은 0에 가까운 파라미터 값을 양산.

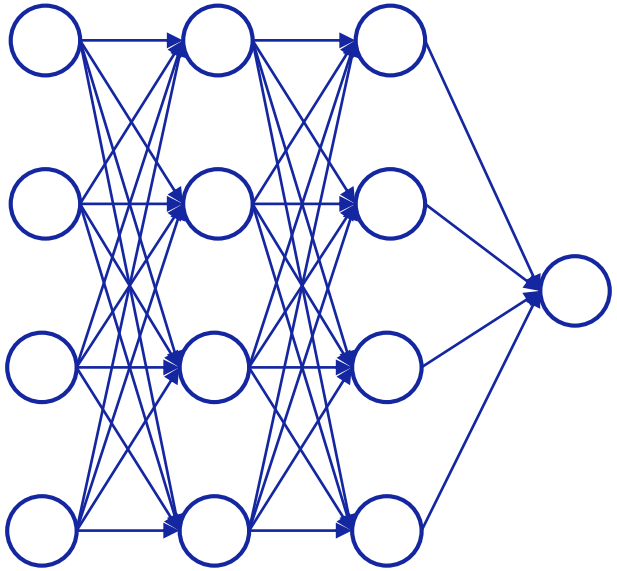
✓ L2와 마찬가지로 과적합을 억제하는 효과를 가지고 있기는 하지만,

그보다 가중치 분포에 미치는 영향이 더 크다.

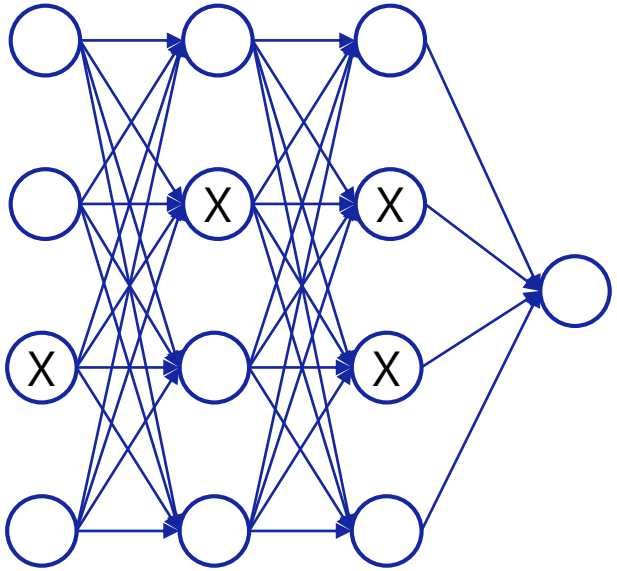
➢ 가중치들이 0 혹은 0에 가까운 원소를 많이 포함하는 일명 희소 텐서가 되는 것을 바란다면 L1손실을 이용.

1.9 다양한 정규화 기법들

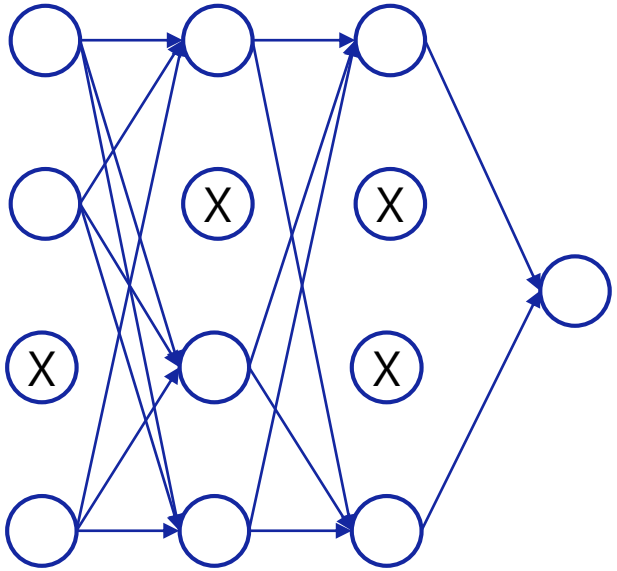
- ✓ 드롭아웃은 과적합 방지를 위해 도입되는 정규화 기법.
- ✓ 계산량을 줄이기 위한 기법은 아니다.
- ✓ 성능 향상을 위해 계산량을 늘려 투자하는 기법.



A. Dropout 적용 전

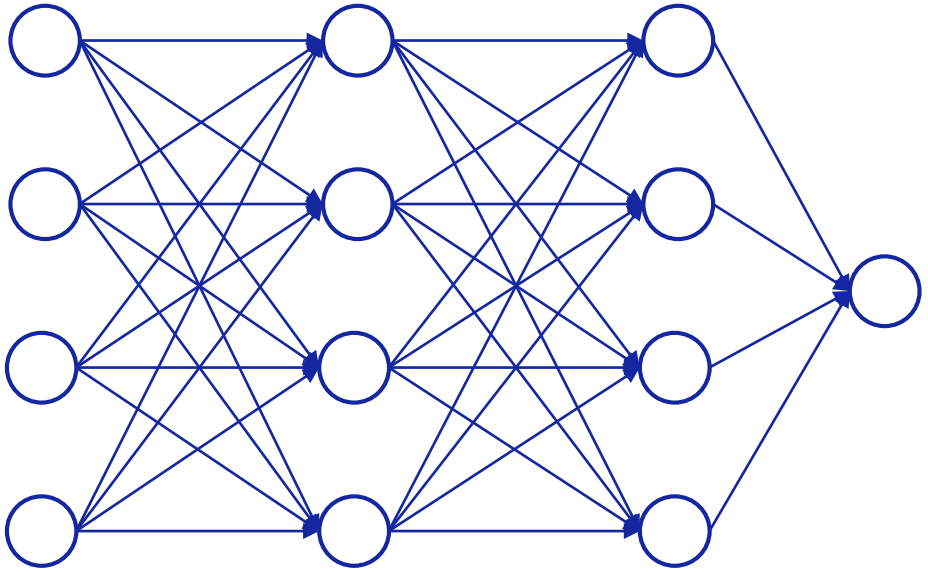


B. Dropout 처리
(난수함수 처리)



C. Dropout의 실질적 효과
(계산량은 오히려 늘어남)

1.9 다양한 정규화 기법들



↑
40%의 확률로
dropout

↑
dropout_계층 삽입
keep ratio = 60%

- 드롭아웃 기법은 **학습시점에서만 적용**되야 하고,
  학습 이후에는 **모든 퍼셉트론들이 다시 투입**되어 문제를 해결
- 같은 문제를 반복해서 풀더라도, **그 처리를 담당하는 퍼셉트론 그룹을 매번 다르게 배당**함으로써 문제 자체를 일정한 패턴형태로
  단순 암기해버리기 어렵게 만드는데서 얻을 수 있습니다.
- 드롭아웃의 무작위 처리는 **지속해서 학습과정에 교란**을 가져오는데
  이 덕분에 학습과정이 일종의 **매너리즘에 빠지지 않고 학습**
- 드롭아웃 기법은 실제로 처리에 반영되는 퍼셉트론 수를 줄여버림으로서
  학습을 어렵게 만드는 장애요인으로 작용하고, **학습속도를 떨어뜨리는 반작용 발생**
- 과적합 현상이 줄어들고, 학습과정에서 성능과 평가 단계에서 성능의 차이가 줄어들면서
  실전에서 우수한 품질을 기대할 수 있음.

1.9 다양한 정규화 기법들

- ✓ **미니배치 내의 데이터**들에 대해 벡터 성분별로 **정규화를 수행**하는 방식.
- ✓ 정규화는 **대상 값들에 동일한 선형 변환**을 가해줌으로써 **평균 0, 표준편차 1의 분포**로 만들어 주는 처리.
- ➢ 입력 성분 간의 **분포 차이로 인한 가중치 학습의 불균형을 방지**하기 위해 도입

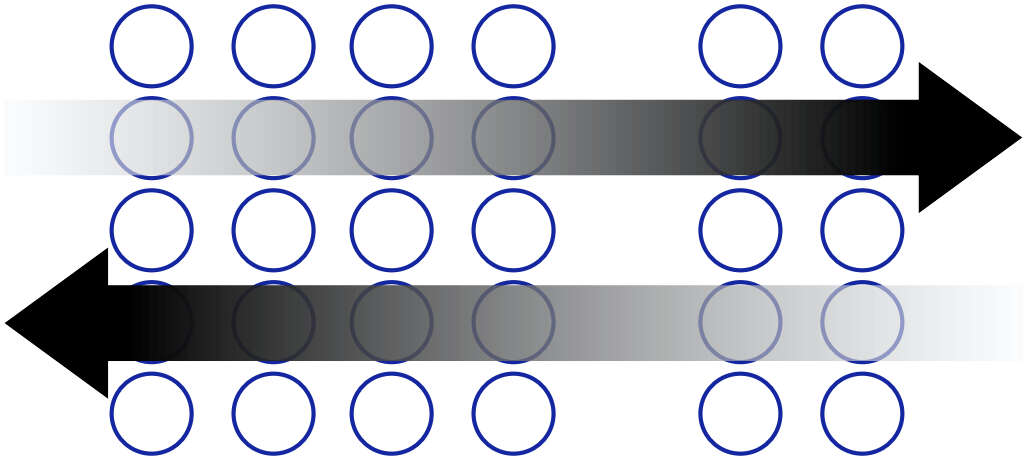
신경망의 성능을 올리기 위해서는 신경망의 구조를

특이하게! 난해하게 쌓아야 한다는데?!

드롭아웃과 배치 정규화가 등장했으니

이제 그럴필요 없어!

『깊은 신경망』



- ✓ 신경망을 깊게 쌓다보면은 **역전파 처리 중 층을 거듭할 수록** 파라미터 성분에 따라 **기울기가 급격히 소멸 혹은 폭주**하면서 **학습이 제대로 이뤄지지 못하는 경우 발생**
- ➢ 네트워크각 층의 입력을 구성하는 **성분별 분포가 심하게 달라서 발생**

1.9 다양한 정규화 기법들

이 작은 표본집단에 불과한 **미니 배치 데이터**에 정규화를 적용한다는 것이 정말 효과가 있다고?

Batch normalization

오히려 Dropout 보다 효과가 좋아!

- ✓ **미니배치 데이터**를 정규화 대상으로 삼는다는
- ✓ **전체 데이터셋 정규화**는 최초의 입력 데이터 즉, **처음에만 가능한 기법**.
  - ➢ 그렇기 때문에 **은닉계층에서 생산하는 은닉 벡터에 대해 적용 불가**
- ✓ 매번 달라지는 미니배치 평균과 분산의 **이용으로 일종의 잡음 주입**
  즉, **노이즈 효과 발생.**
  - ➢ 신경망은 **더 어렵게 학습**을 진행, 미니배치 구성에 에포크마다 무작위로 바뀐다면 **배치 정규화는 학습 과정을 끊임없이 교란.**
  (★ 드롭아웃과도 같은 효과)

작은 미니배치는 배치 정규화 효과가 극단적으로 나타날 수 있기 때문에, 배치 정규화를 이용할 때 너무 작은 미니배치보다는 **약간 크기를 조금 키운 미니배치에서 학습 결과가 더 좋게 나와.**

같은 수의 학습 데이터를 처리할 때 **미니배치 크기를 키워주게 되면**은 자연스럽게 미니배치 처리 횟수가 줄어들면서 **결과적으로 학습 속도를 높이는 부수적인 효과** 까지 얻는다고!