

2019.10.17

DEEP LEARNING

4기 김현우

CONTENTS

1. 딥러닝의 활용 분야
2. 빠르게 보는 딥러닝의 역사
3. 딥러닝의 핵심 구조
4. 딥러닝의 원리와 실전을 동시에 배워봅시다
(with tensorflow 실전 코드)

Chapter 1

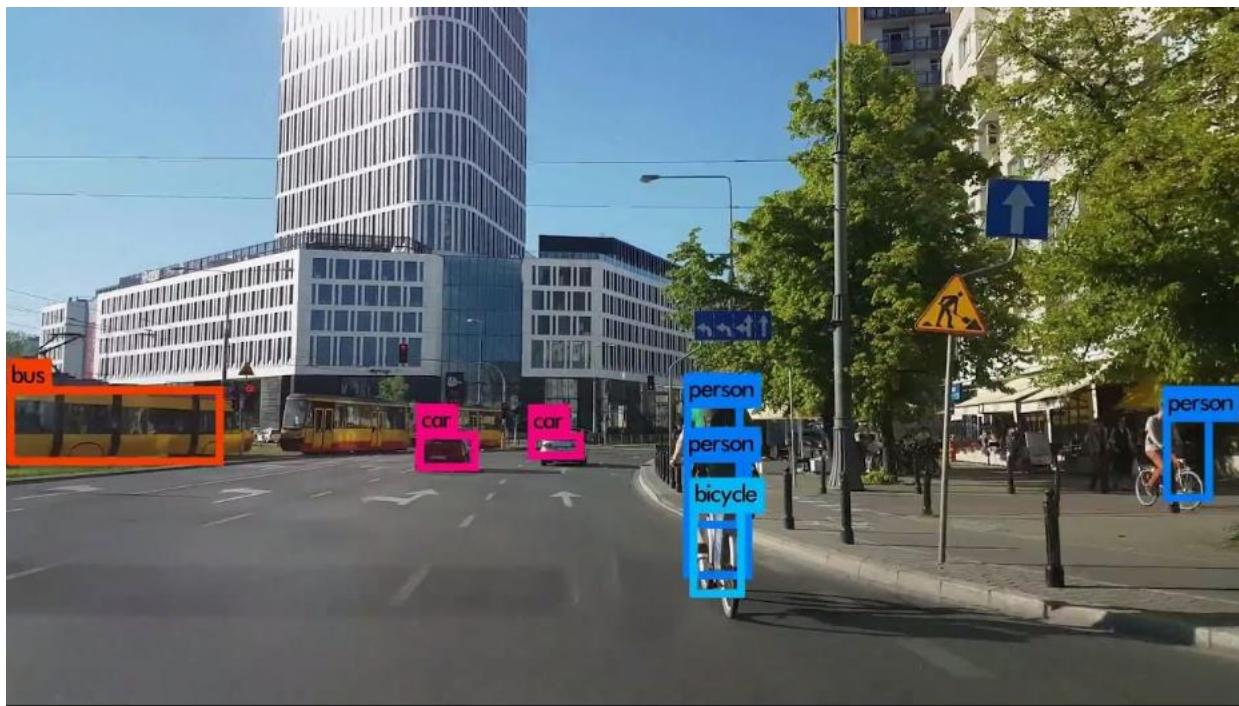
딥러닝이 활용되는 분야

Chapter 1 | NN 활용 예시

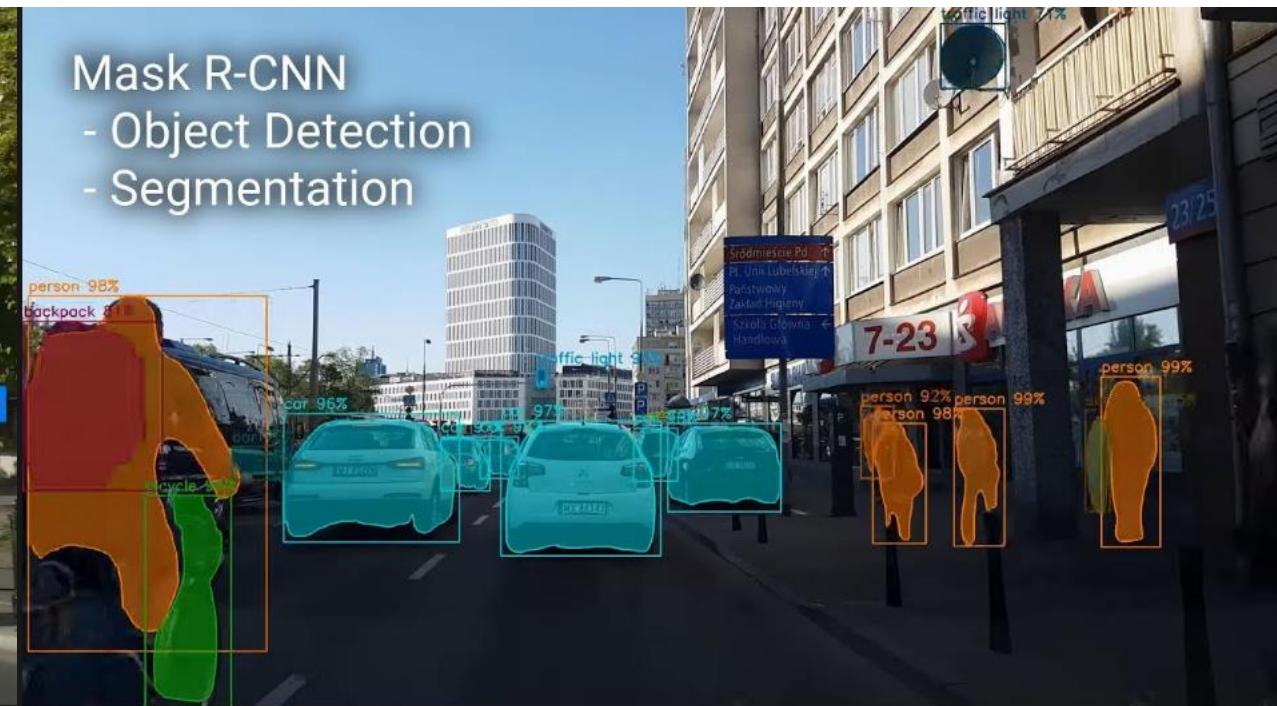
<https://www.youtube.com/watch?v=MPU2HistivI&t=14s>



Chapter 1 | NN 활용 예시



Mask R-CNN
- Object Detection
- Segmentation



Chapter 1 | NN 활용 예시



Chapter 1 | NN 활용 예시

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.



Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.



Mnih and Hinton, 2010

Chapter 1 | NN 활용 예시

No errors



A white teddy bear sitting in the grass



A man riding a wave on top of a surfboard

Minor errors



A man in a baseball uniform throwing a ball



A cat sitting on a suitcase on the floor

Somewhat related



A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
<https://pixabay.com/en/luggage-antique-cat-1643010/>
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>
<https://pixabay.com/en/handstand-lake-meditation-496008/>
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)

Chapter 1 | NN 활용 예시



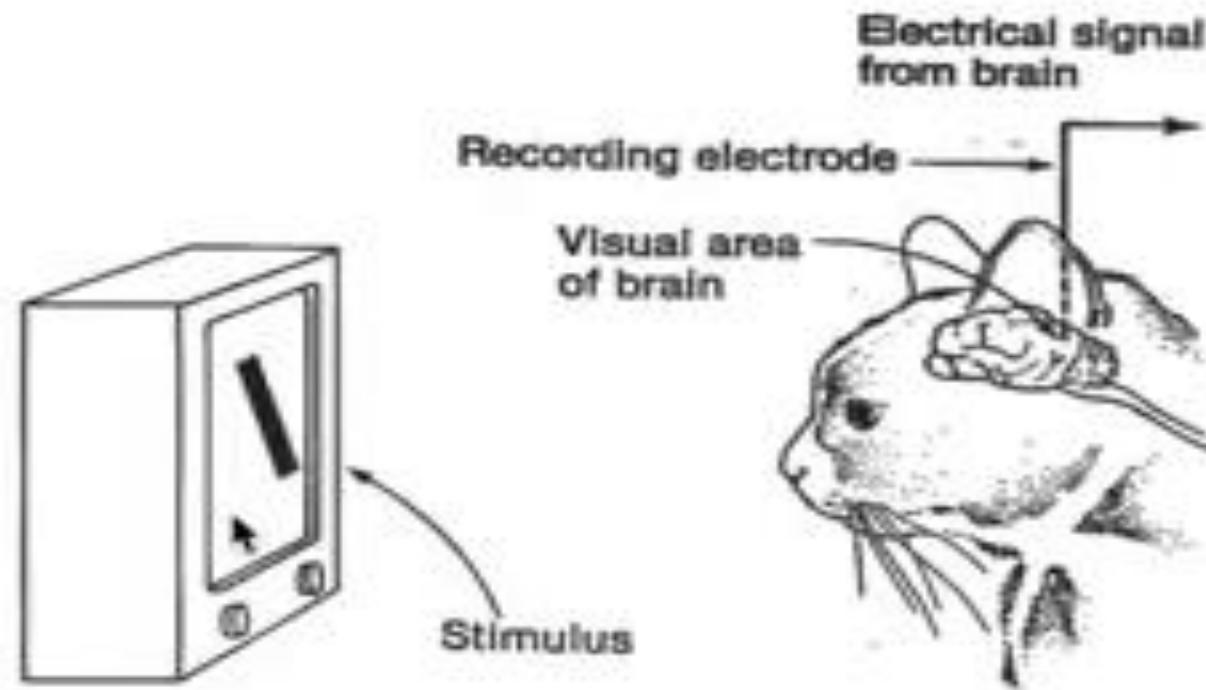
한국어 감지	영어
파파고는 딥러닝을 사용하여 만들어졌습니다.	Papago was made using deep learning.
X	파파고우 와즈 메이드 유징 딥 러닝.
23 / 5000	23 / 5000

■ 딥러닝의 장점이 무엇이길래 여러 곳에서 활용될까?

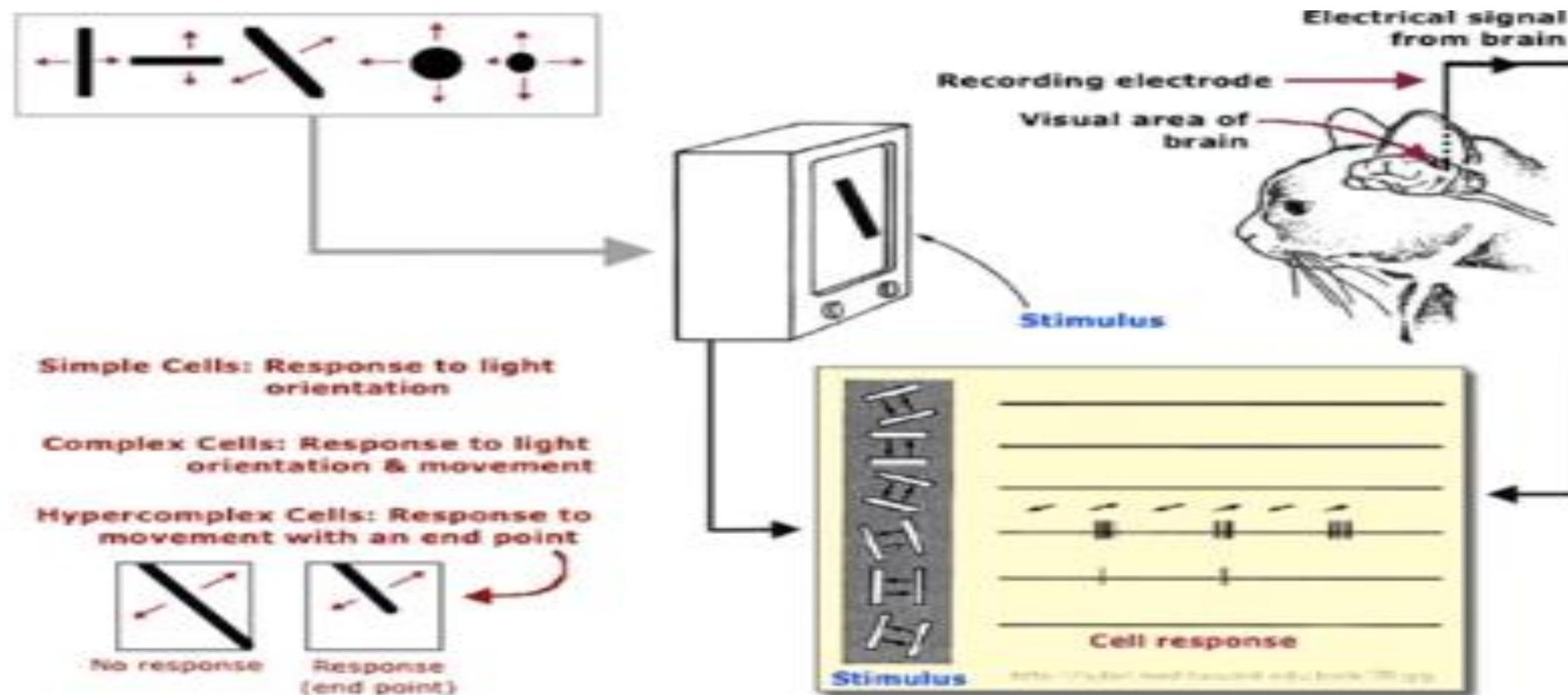
Chapter 2

간략히 배워보는 딥러닝의 역사

Chapter 2 | 딥러닝의 역사



Chapter 2 | 딥러닝의 역사



Chapter 2|딥러닝의 역사(자세히 궁금하신 분들을 위해)

초기 NN 학자들은 활성함수로 sigmoid 함수를 사용하였는데 sigmoid 함수는 입력 데이터의 값으로 0을 중심으로 일정 범위를 넘어가는 음수와 양수를 주었을 때 출력 데이터를 0과 1에 매우 근접한 값을 가지게 하는 함수이다. 이때 출력 데이터 값이 0이라는 것은 관련 정보를 후속 노드들에게 전달하지 않겠다는 것으로도 해석할 수 있다. 그런데 이러한 NN은 선언과 연언의 논리 상태는 잘 판단할 수 있었지만 베타적 논리합을 판단할 수 없는 문제에 봉착하게 되었고 그에 따라 NN은 오랜 시간 침체기에 빠졌다.

이러한 문제를 해결한 방법은 민스키가 주장한 MLP 방법을 이용한 NN이었다. 그는 기존 모델이 하나의 층(layer)만을 사용하는 것이 문제였다고 지적하고 여러 층을 사용하는 방법을 통해 이를 해결할 수 있음을 수학적으로 증명하였다. 하지만 민스키는 동시에 여러 층을 사용하는 방법은 각 가중치를 학습시킬 방법이 존재하지 않는다고 주장하여 NN은 다시 한번 그 존립에 위기를 맞았다.

하지만 힌턴이 1986년에 기존의 방법인 순전파와 달리 뒤에서부터 가중치를 학습시켜 나가는 방법인 역전파를 도입하면 이를 해결할 수 있다고 주장하였다. 하지만 이러한 역전파 방법 역시 NN을 구성하는 층이 많아질수록 앞의 층에 위치한 노드들의 가중치가 거의 변하지 않는 딜레마에 봉착하게 되었고 이는 20년 간 해결되지 않았다. 이러한 침체기에서 힌턴은 CIFAR 단체의 도움을 얻어 연구를 계속 진행하였고 각 노드들의 가중치의 초기값을 적절히 배정해주는 방법으로 기존의 NN이 가진 문제들을 해결하였다. 이후로 많은 사람들이 NN에 다시 관심을 갖게 되었고 힌턴은 이를 기념하여 새로운 NN에 딥러닝이라는 명칭을 부여하였다. 이를 기준으로 딥러닝의 성능은 점진적으로 향상되었는데 2012년 힌턴 교수의 제자 알렉스는 0이하의 입력 데이터 값은 후속 노드로 전달하지 않는 RELU 활성함수를 사용하여 비약적인 성능 향상을 보였다. 그는 기존의 역전파 방법에서 발생하는 문제는 가중치를 조절하는 것만으로 해결되지 않고 활성함수를 조절하는 방법이 추가적으로 필요하다고 보았고 이러한 접근법은 효과적이었다. 이후로 딥러닝은 매년 새로운 기록을 경신하며 이미지 분류 등 특정 분야에서 인간의 성능을 뛰어넘는 양상을 보이며 발전을 거듭하고 있다.

Chapter 2 | 딥러닝의 역사(요약)

1950년대 처음 고안 → 침체기

1960년대 민스키 교수의 다층(multi-layer) 방법 → 침체기

1980년대 힌턴 교수의 역전파 방법 → 침체기

2000년대 힌턴 포기하지 않고 계속 매달려 가중치 문제 해결 → 딥러닝이란 명칭 처음 등장.

2010년대 알렉스 넷을 시작으로 각종 통계기술과 컴퓨팅 능력의 향상으로 딥러닝 부흥기

Chapter 3

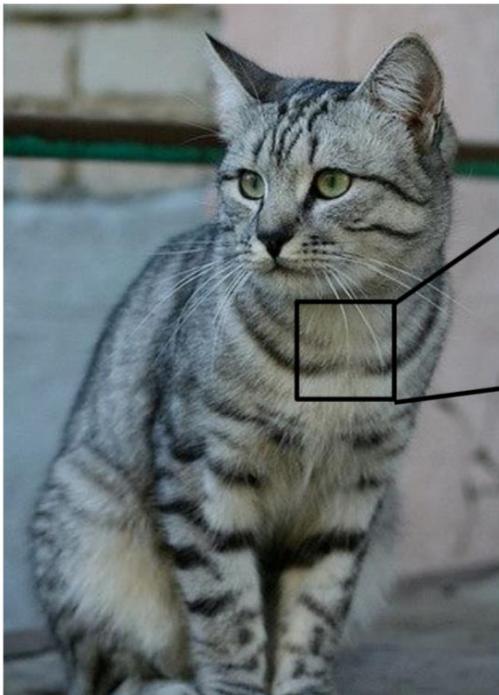
딥러닝의 핵심 구조



■ 컴퓨터는 이 고양이를 어떻게 인식할까?



Chapter 3 | 컴퓨터가 보는 것



This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

```
[[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
 [ 91 98 102 106 104 79 98 103 99 105 123 136 110 105 94 85]
 [ 76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
 [ 99 81 81 93 120 131 127 100 95 98 102 99 96 93 101 94]
 [106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
 [114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
 [133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
 [128 137 144 140 109 95 86 70 62 65 63 63 60 73 86 101]
 [125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
 [127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
 [115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
 [ 89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
 [ 63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
 [ 62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]
 [ 63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
 [ 87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
 [118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]
 [164 146 112 80 82 128 124 104 76 48 45 66 88 101 102 109]
 [157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
 [130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
 [128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]
 [123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
 [122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
 [122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]
```

What the computer sees

An image is just a big grid of numbers between [0, 255]:

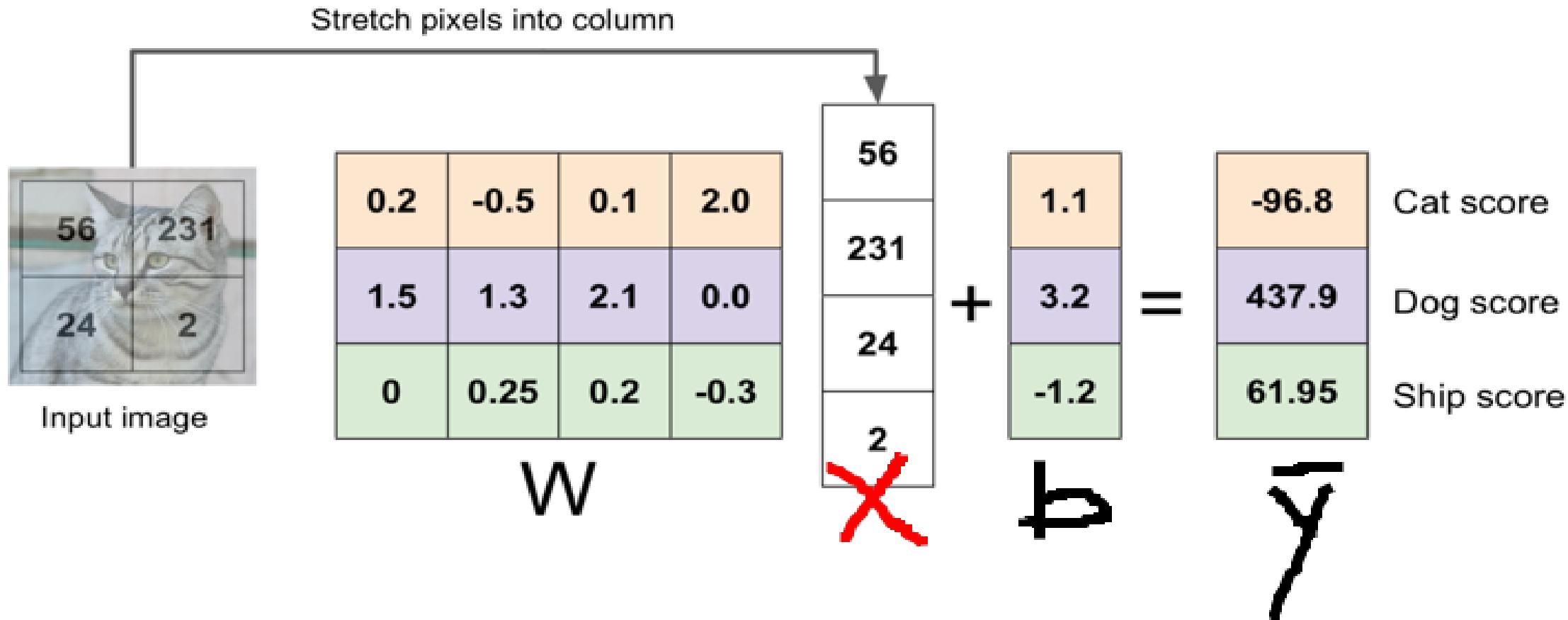
e.g. 800 x 600 x 3
(3 channels RGB)

■ 딥러닝의 핵심!!

■ 가설함수와 Weight 조절

■ 1. 가설함수를 세웁니다 ➔ $H(x) = Wx + b$

Chapter 3 | 픽셀은 4, 카테고리는 3으로 단순화해보자

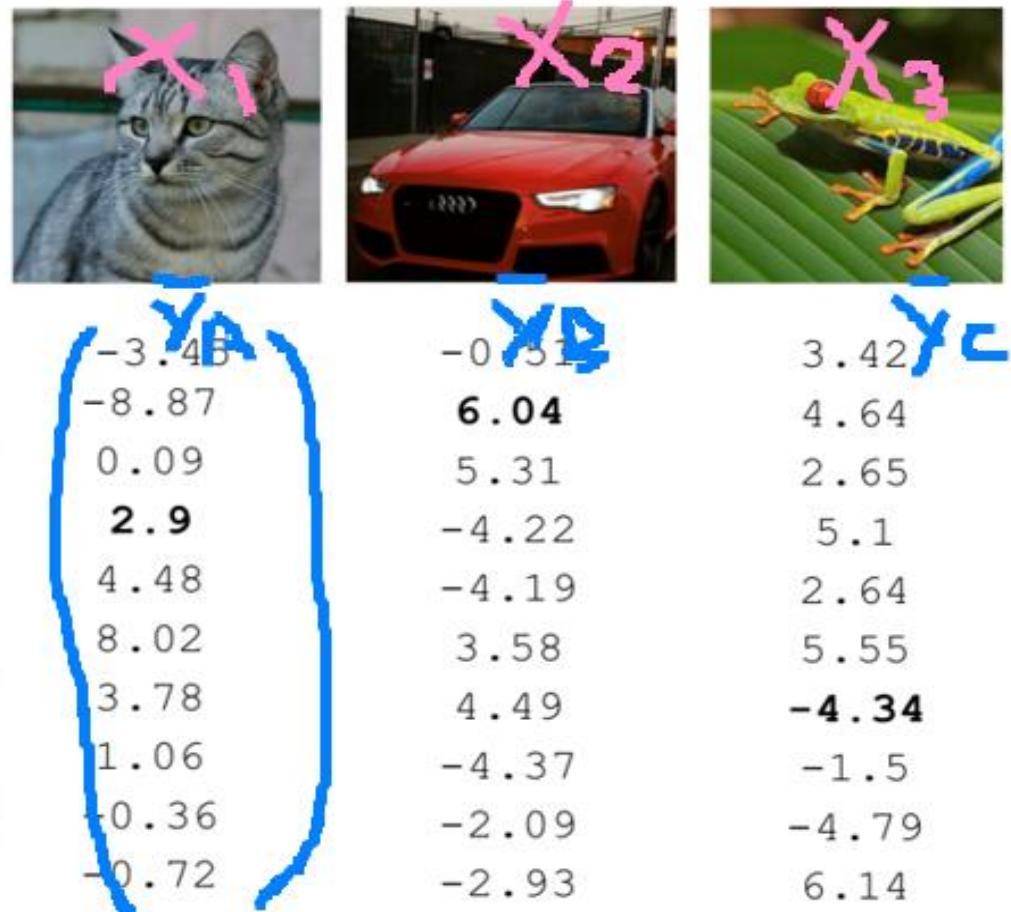


■ Input(x)으로 고양이 사진이 들어왔을 때 여러 종류(라벨) 중 고양이라는 output($H(x)$) 이 가장 크게 나오는 게 하는 것을 목표로 합니다

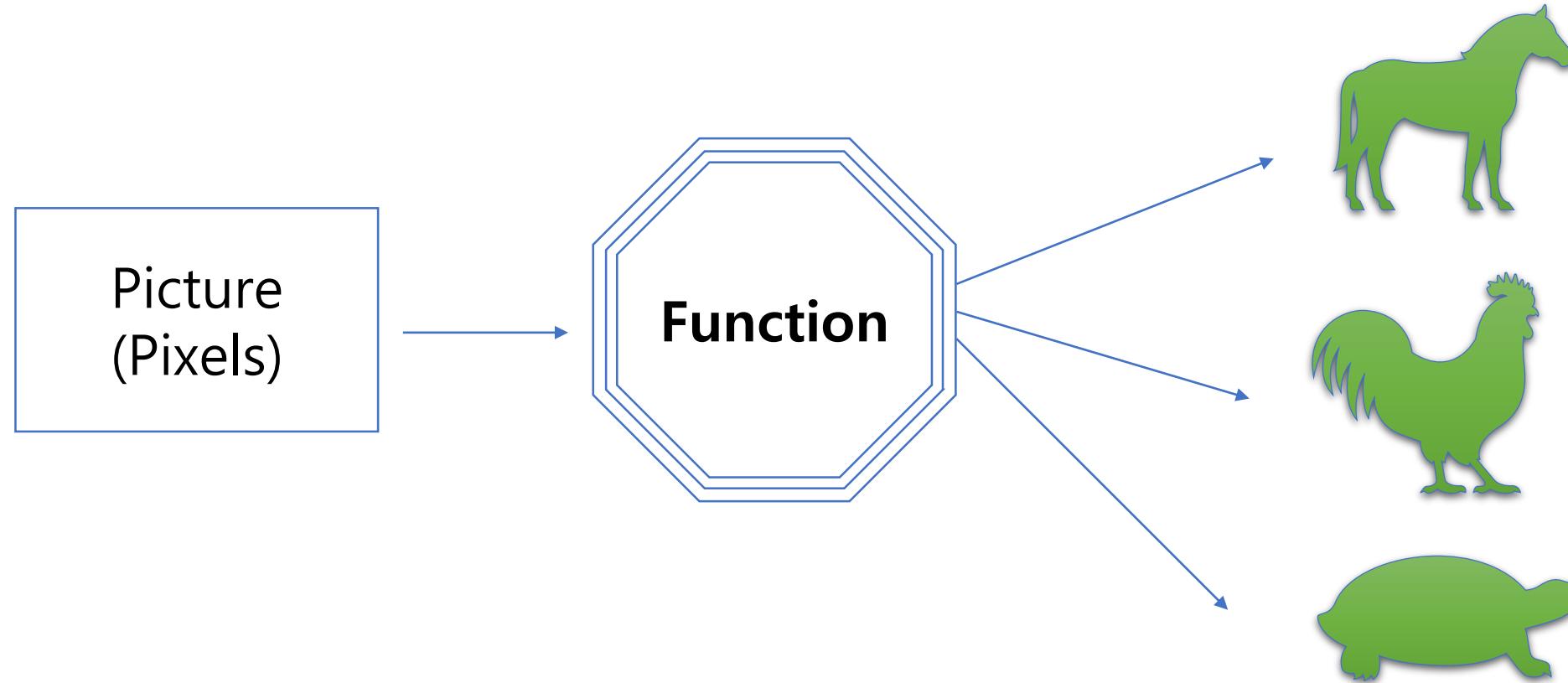


airplane	-3 . 45
automobile	-8 . 87
bird	0 . 09
cat	2 . 9
deer	4 . 48
dog	8 . 02
frog	3 . 78
horse	1 . 06
ship	-0 . 36
truck	-0 . 72

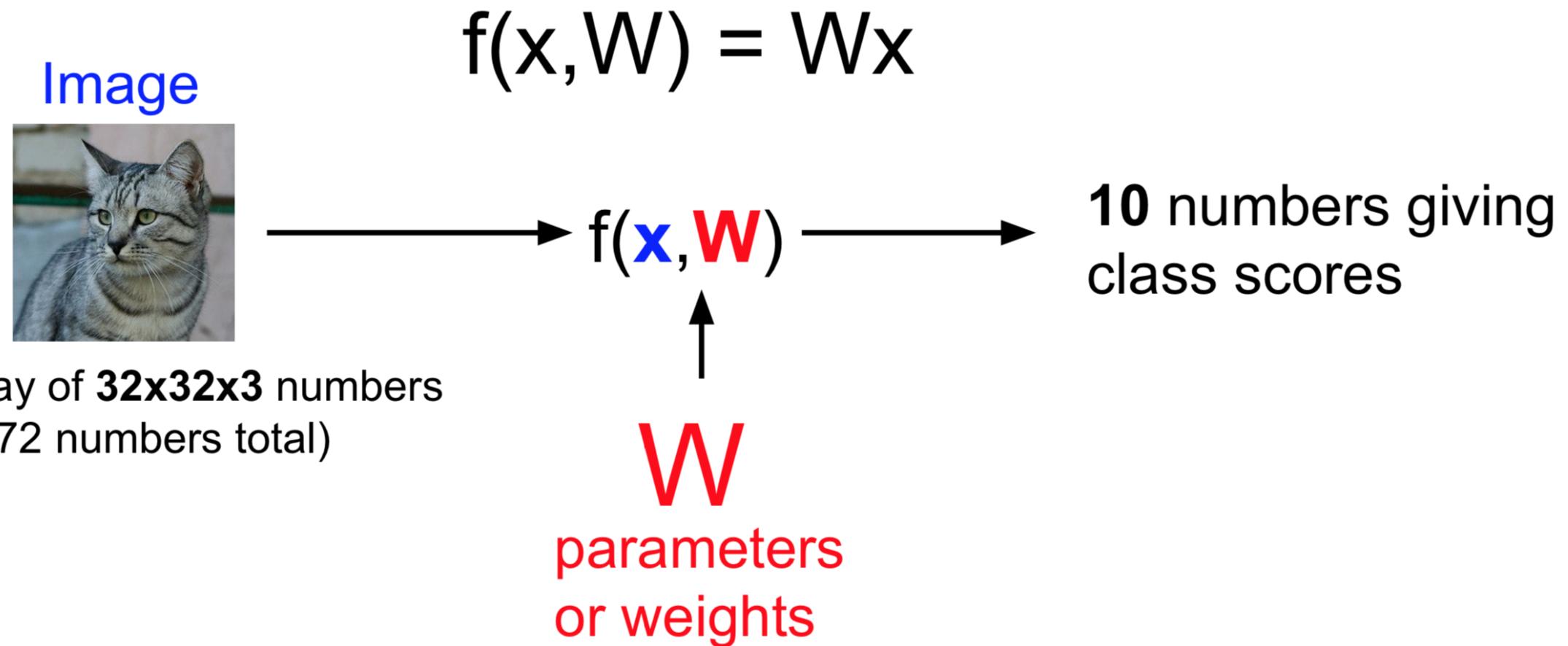
Chapter 3|딥러닝의 구조



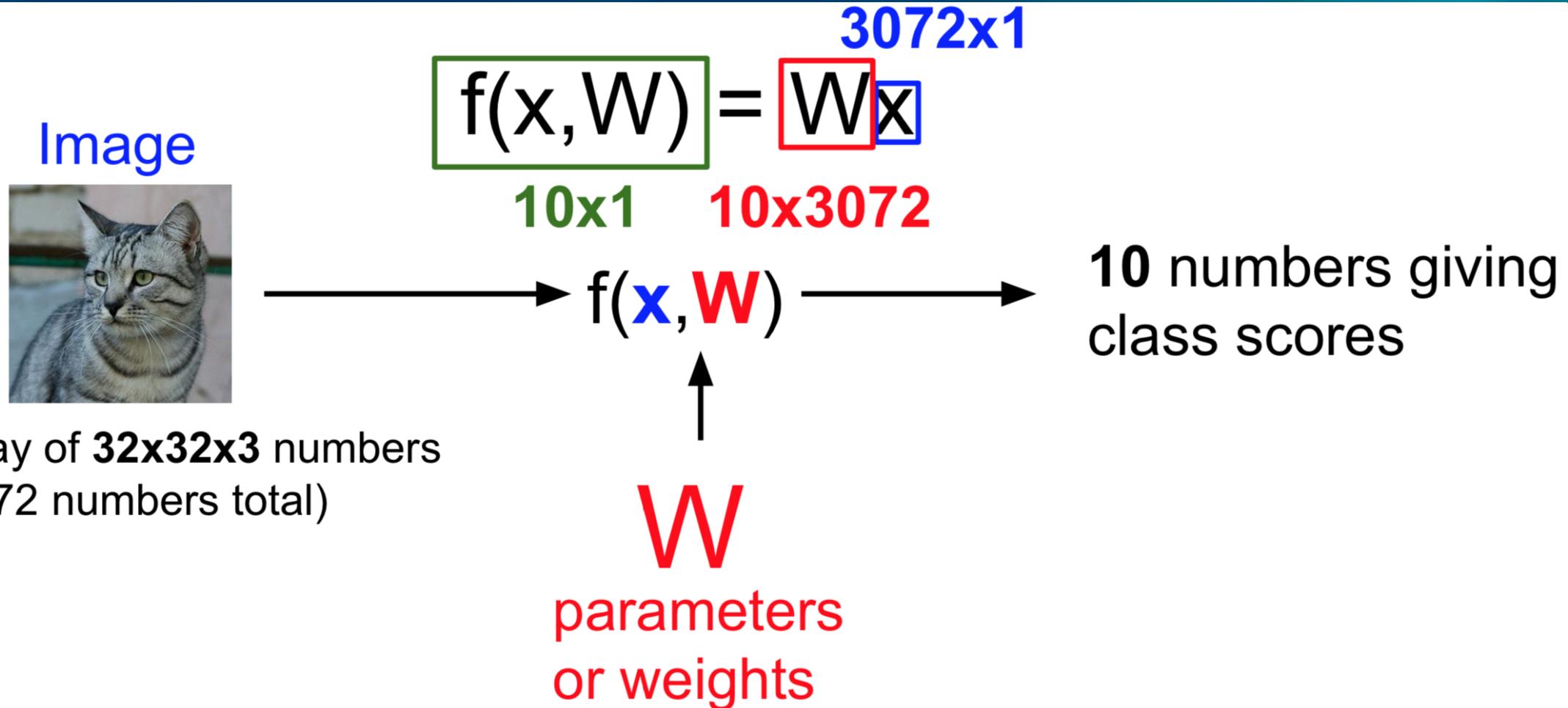
Chapter 3 | 딥러닝의 구조



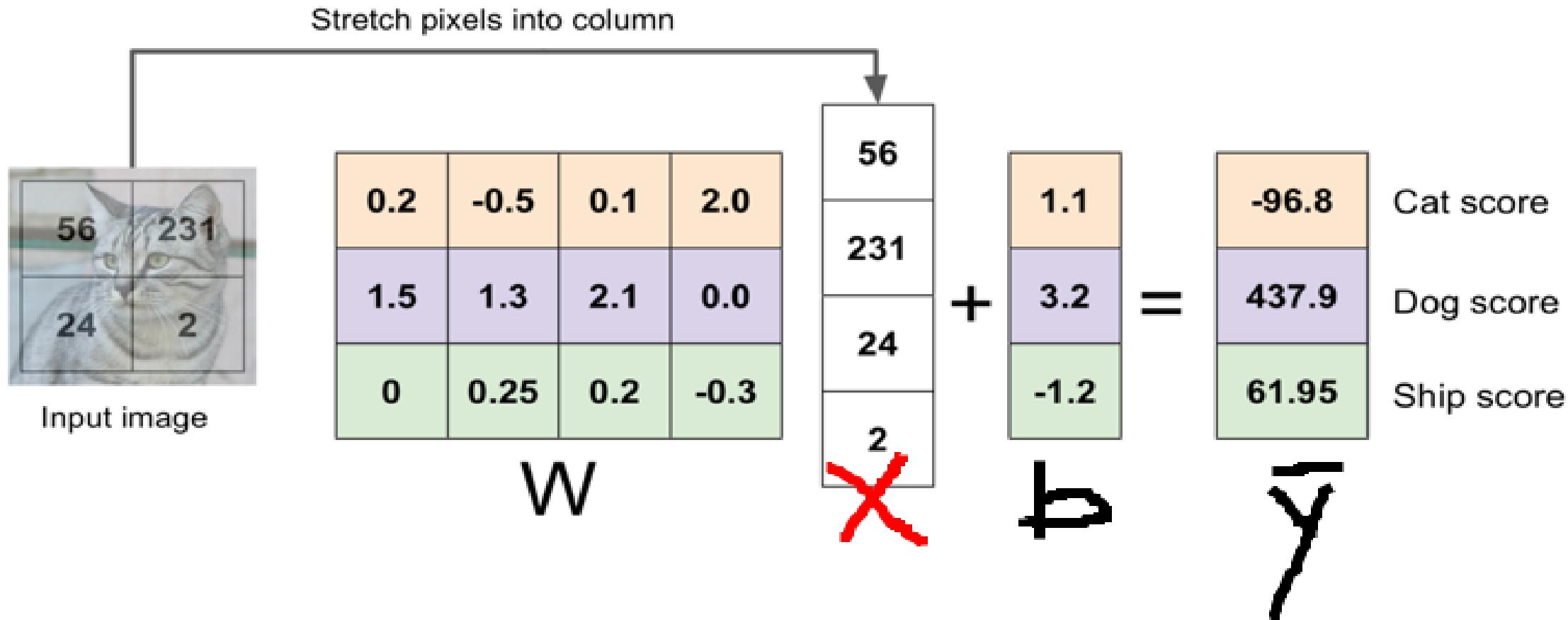
Chapter 3 | 더 큰 사이즈 input과 더 많은 클래스 분류



Chapter 3 | 더 큰 사이즈 input과 더 많은 클래스 분류



Chapter 3 | 픽셀은 4, 카테고리는 3으로 단순화해보자



■ 2. 원하는 결과를 예측하도록 Weight를 조절한다.

■ 딥러닝의 핵심=Weights에 얻은 지식을 저장
딥러닝 자체는 굉장히 컴퓨팅 파워를 요하는 기술이지만, Weights만 가져오면 되니
스마트폰, 웹브라우저 등에서도 사용 가능하게 됨

Chapter 4

딥러닝 실전과 원리를 동시에 배워보자

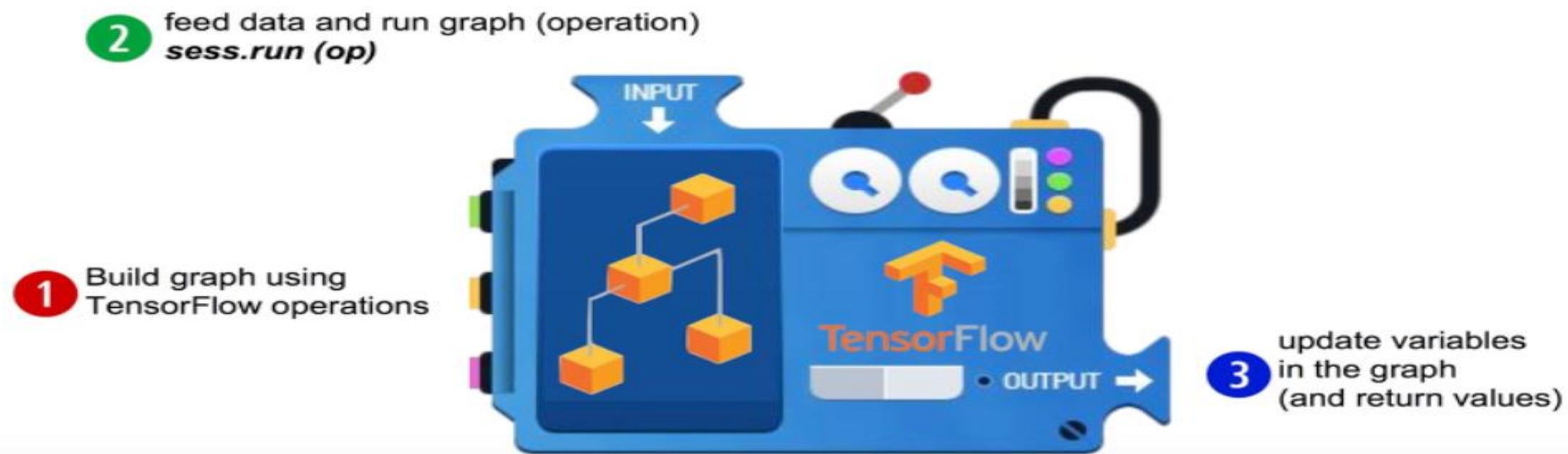
■ 왜 동시에 배웁니까?

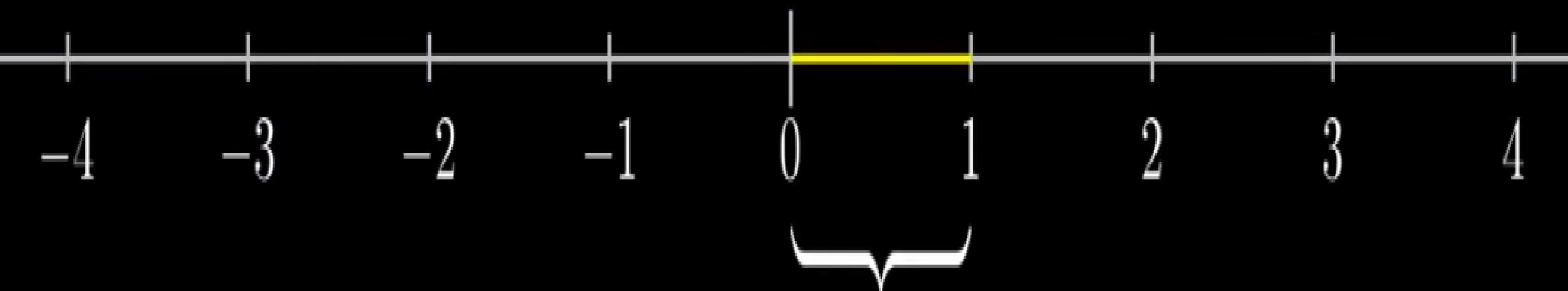
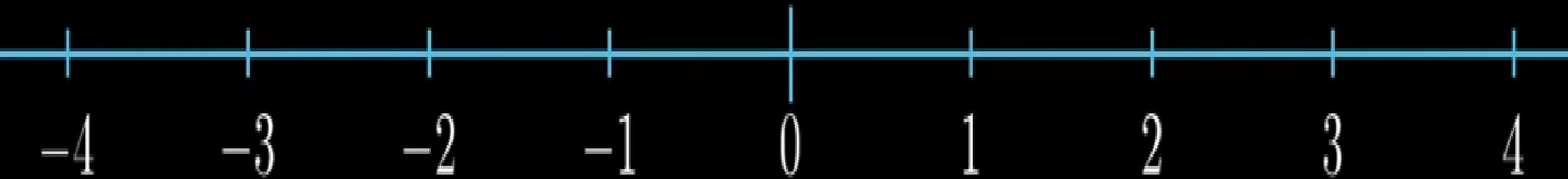
■ 딥러닝의 원리와 실제 활용법은 긴밀히 연결되어 있기 때문!

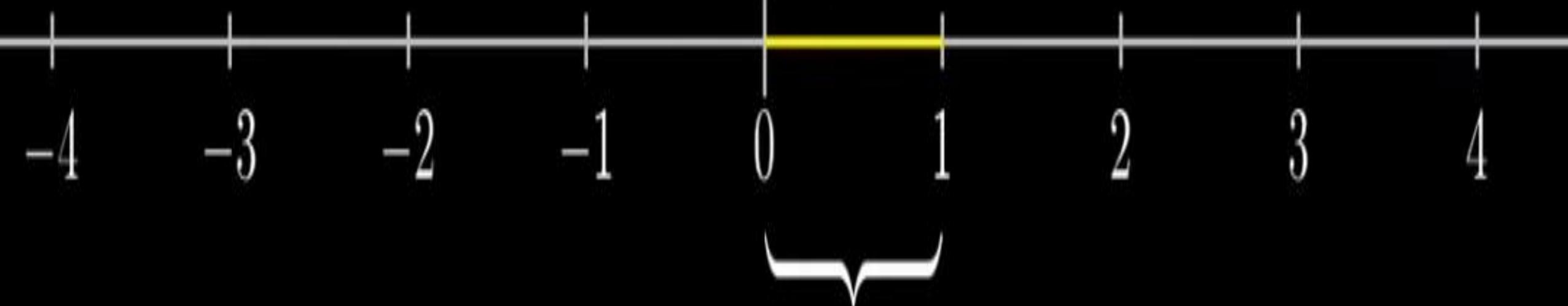
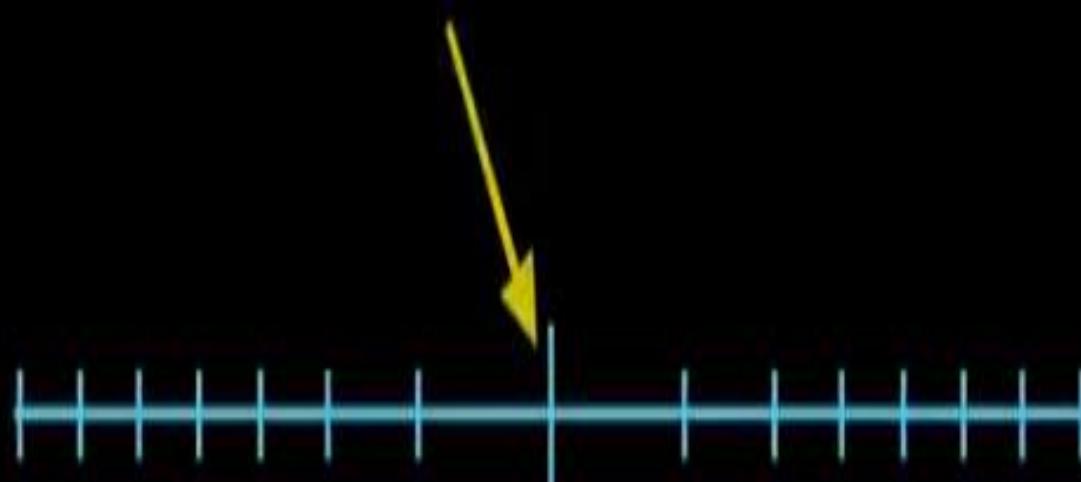
Chapter 4 | 텐서플로우 메카니즘

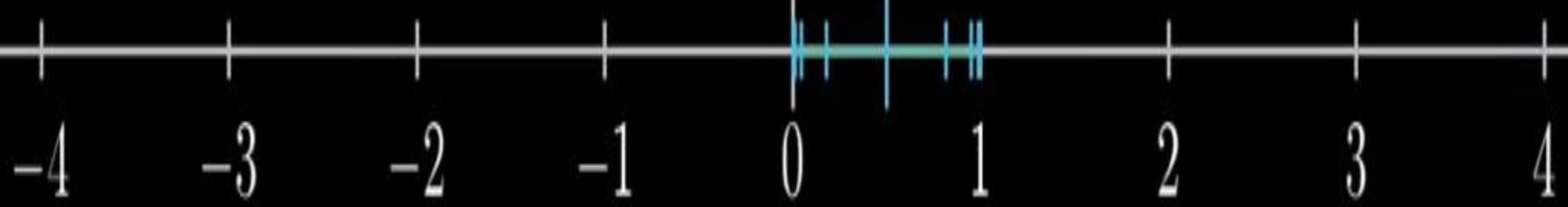
다운로드가 안되시면 pip install --upgrade tensorflow==1.12.0

TensorFlow Mechanics

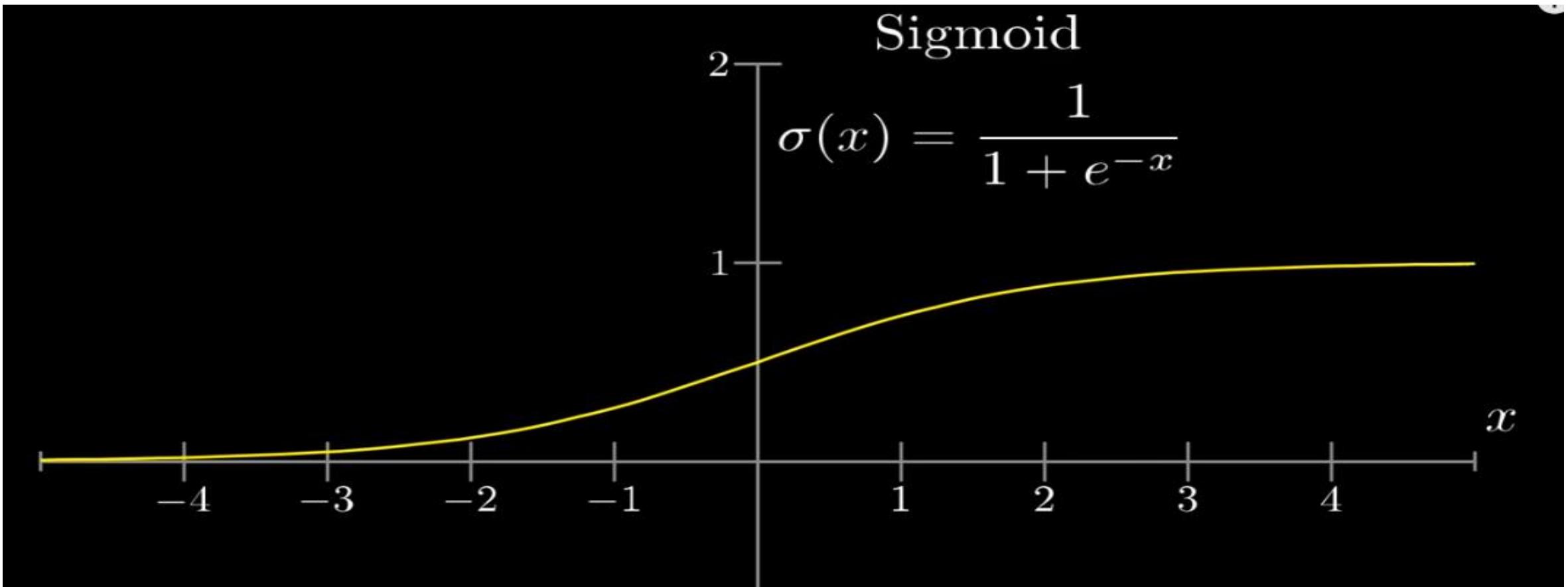








Chapter 4 | Sigmoid

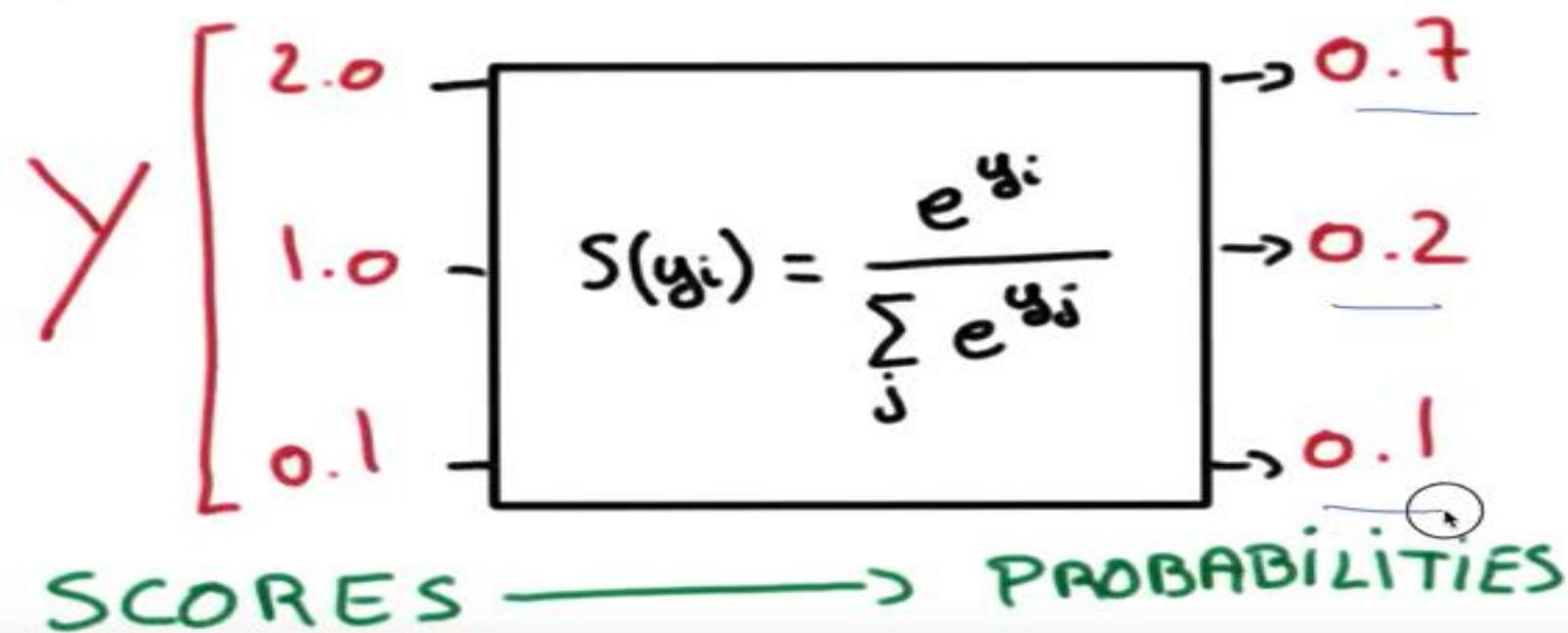




$$H(x) = Wx + b \quad \Rightarrow \quad H(X) = \frac{1}{1 + e^{-W^T X}}$$

Chapter 4 | Softmax

SOFTMAX

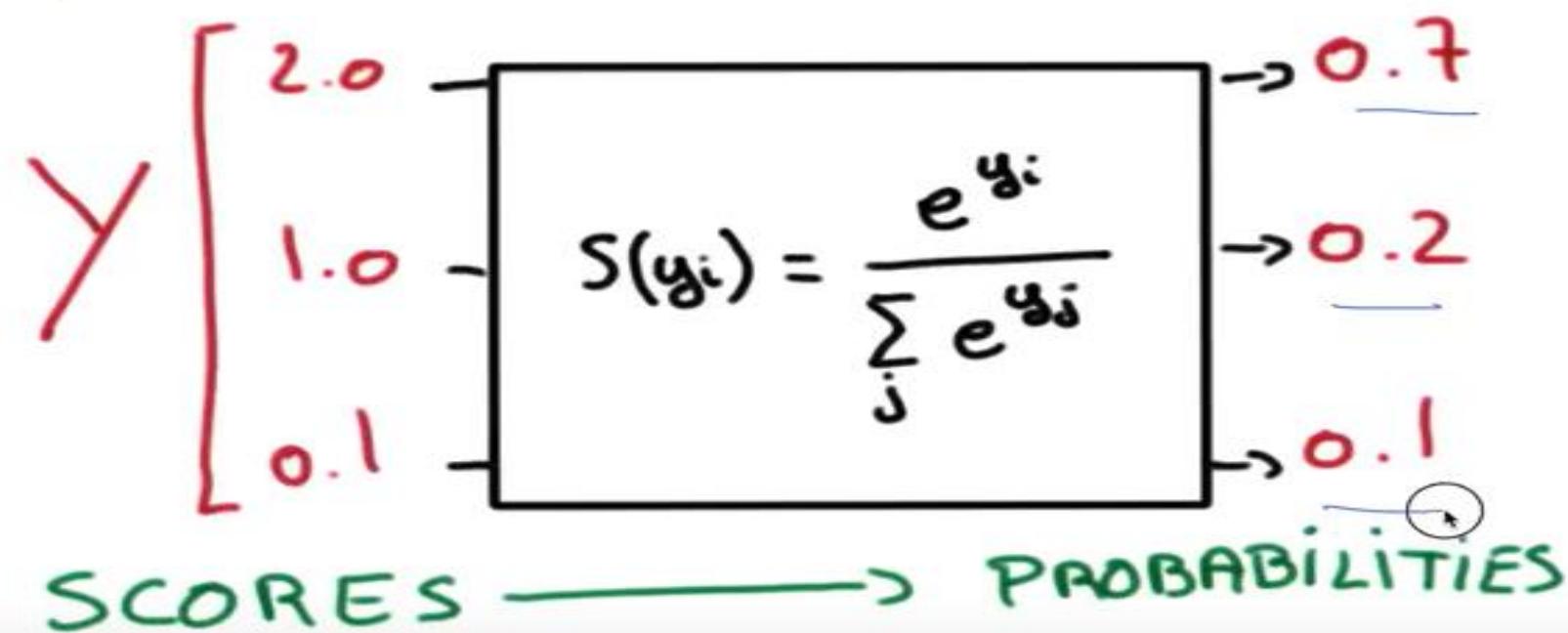


■ 왜 softmax를 사용할까?

■ 각각의 예측값을 0과 1사이로 만들어주고
모든 예측값의 합을 1로 만들어주기 때문.
즉 점수(score)의 확률(probabilities)화..!

Chapter 4 | Softmax

SOFTMAX



Chapter 4 | Softmax와 sigmoid

- Sigmoid는 주로 hidden 노드에 부착되고
- Softmax는 각 값을 확률처럼 만들어주기 때문에 classification을 하는 마지막 output 노드에 사용(다른 뉴런 출력값과 상대적 비교를 통해 출력값이 결정)

$$y = \frac{\exp(t)}{1 + \exp(t)}$$

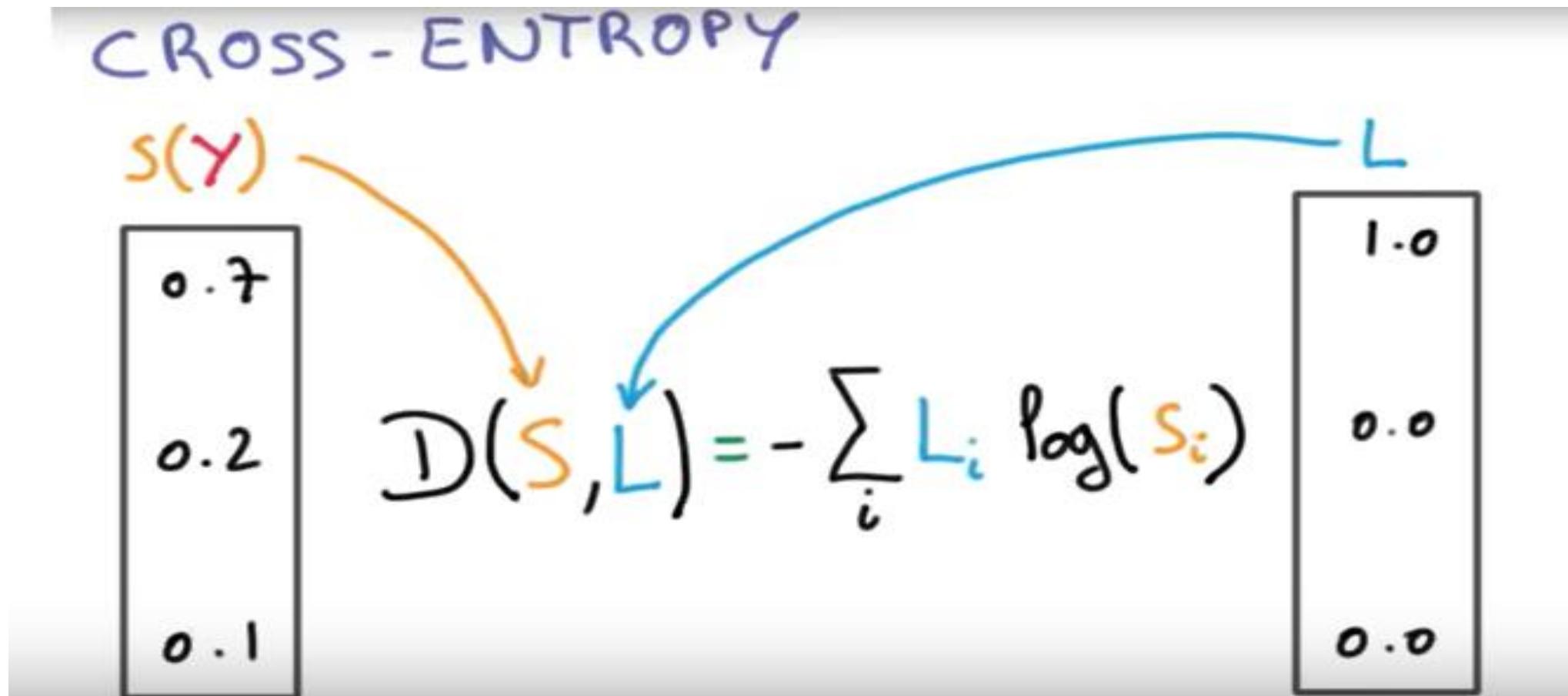
$$\frac{\exp(t_i)}{1 + \sum_{j=1}^{K-1} \exp(t_j)}$$

- 수학적으로 softmax 함수는 sigmoid 함수의 일반형
- (다른 클래스가 sigmoid는 2개인데 softmax는 k개 만큼)

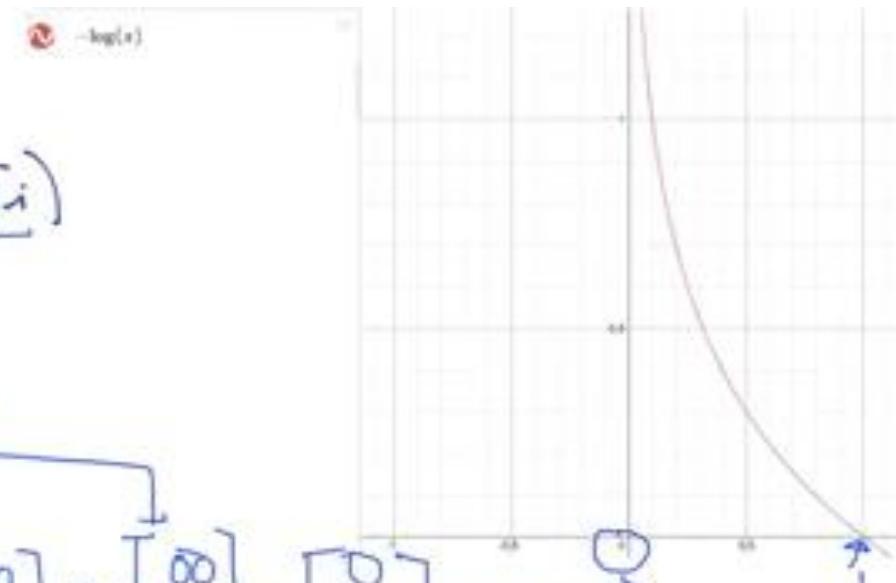
Chapter 4 | 중간 정리

- X라는 input을 받아 결과를 내는 가설함수 $H(x) + b$ 를 만듦
- 이 가설함수를 목적에 따라 sigmoid나 softmax를 사용하여 조금 변형.

Chapter 4 | 코스트 함수(Cross-entropy function)



Chapter 4 | 코스트 함수(Cross-entropy function)



$$-\sum_i L_i \log(s_i)$$

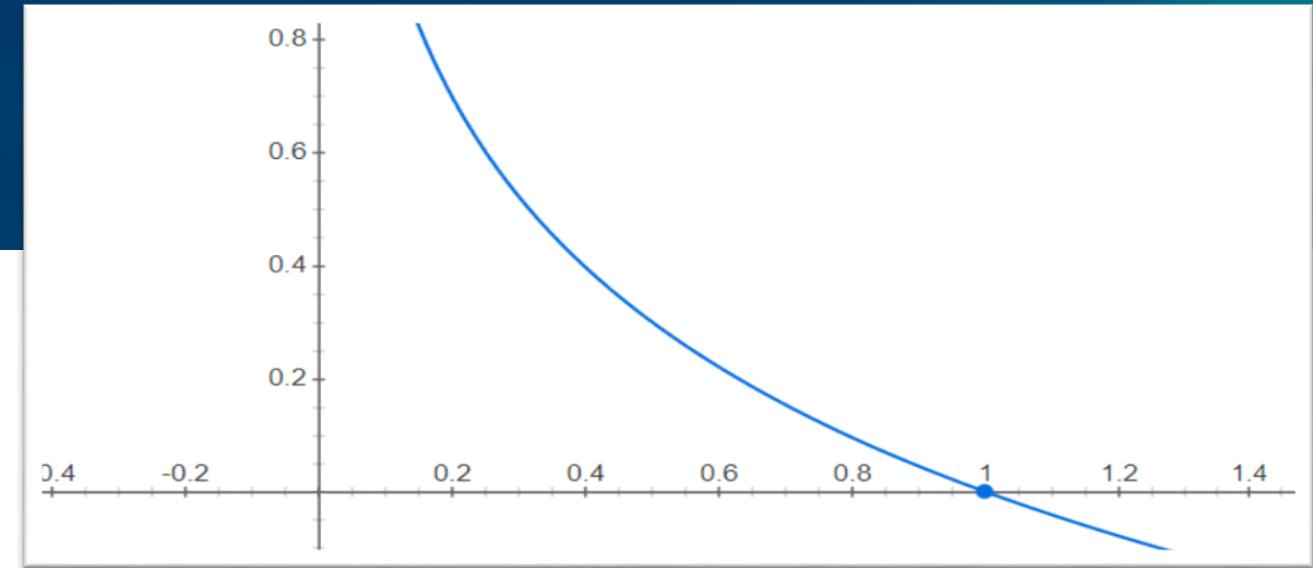
$$-\sum_i L_i \log(\bar{y}_i) = \sum_i L_i * -\log(\bar{y}_i)$$

$$\underline{Y} = \underline{L} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \underline{s}$$

$$\bar{Y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \text{ B } (0k) \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow 0$$

$$\bar{Y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A(X), \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ \infty \end{bmatrix} = \begin{bmatrix} 0 \\ \infty \end{bmatrix} \Rightarrow \infty$$

실제값 y_i \rightarrow 예측값 $H(x_i)$

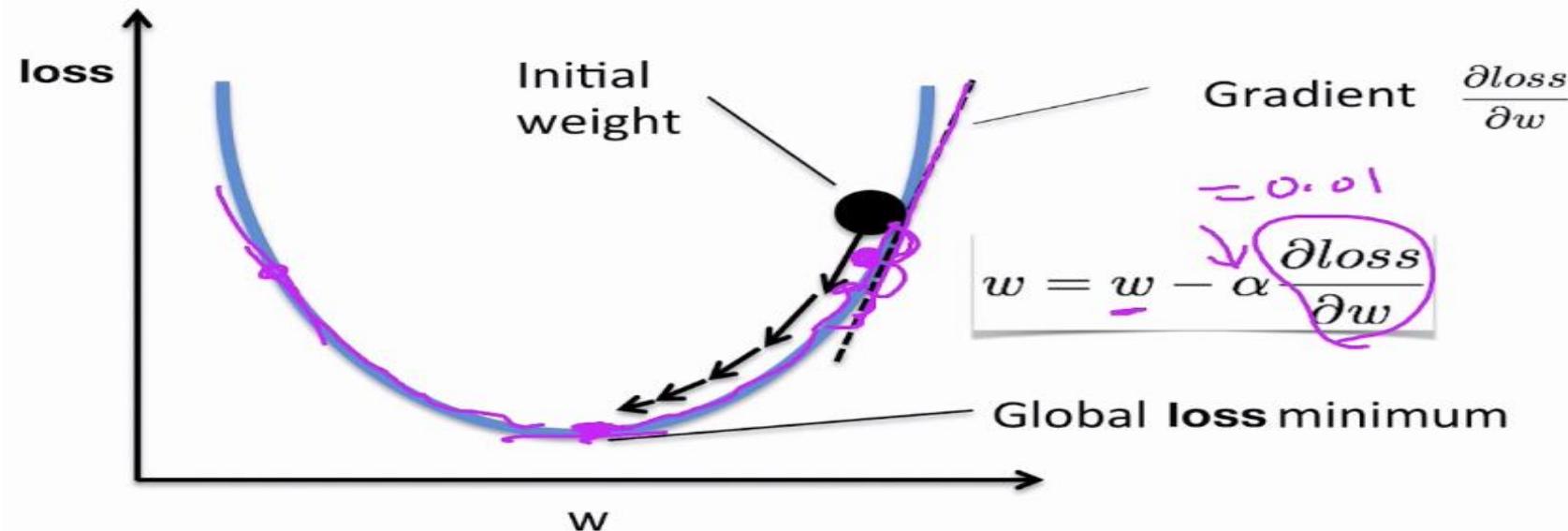


- 실제 값($y=L_i$)가 1일때 예측한 값($H(x)$)가 0이라면 $-\log(0)$ 으로 코스트가 매우 커진다.
반대로 실제 값(y)가 1일때 예측한 값($H(x)$)가 1이라면 $-\log(1)$ 으로 코스트가 0이 된다.

■ 즉 결국 크로스 엔트로피 함수는 실제와 예측이 일치할 때
코스트를 0으로 만들고 실제와 예측이 다르다면 코스트를 매우
크게 하는 것.

Chapter 4 | 기본적인 경사하강법

Gradient descent algorithm



$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

Chapter 4 | 중간 정리

- X라는 input을 받아 결과를 내는 가설함수 $H(x) + b$ 를 만듦
- 이 가설함수를 목적에 따라 sigmoid나 softmax를 사용하여 조금 변형.
- 가설함수가 정답을 맞추면 코스트가 낮게 틀리면 코스트가 높아지게끔 코스트 함수를 만듦
- 경사하강법을 이용하여 코스트가 낮아지게끔 w 를 조절

Chapter 4 | 텐서플로우 메카니즘

index

```
0      [[ 3 10  1]
1      [ 4  5  6]
2      [ 0  8  7]]
```

tf.argmax(a,0)

각 열에서 가장 큰 숫자의
인덱스 → [1 0 2]

0 1 2

```
[ [ 3 10  1]
[ 4  5  ]
[ 0  8  7] ]
```

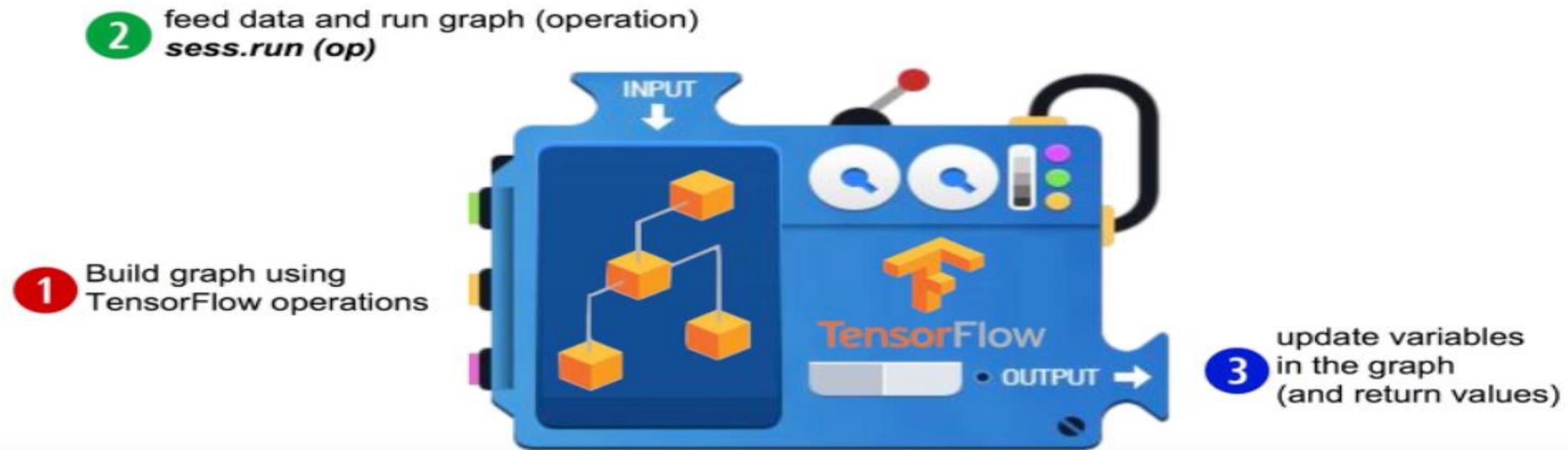
tf.argmax(a,1)

각 행에서 가장 큰
숫자의 인덱스 → [1 2 1]

■ 이때 우리가 찾는 것은 예측 값들 중 가장 큰 값으로 softmax를 거쳐 [0.1 0.1 0.6 0.1 0.1] 이런 식으로 되어있기 때문에 argmax 1을 사용해서 각 행에서 최대값을 뽑아낸다

Chapter 4 | 텐서플로우 메카니즘

TensorFlow Mechanics



epoch는 전체를 반복하여 돌리는 횟수를 말하고
Batch는 변수에 한번에 들어가야할 자료의 개수들.

■ Cost 변수를 다시 올라가서 보면
`tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,labels=y)`
으로 logits에 x가 label에 y가 들어있다.

Chapter 4

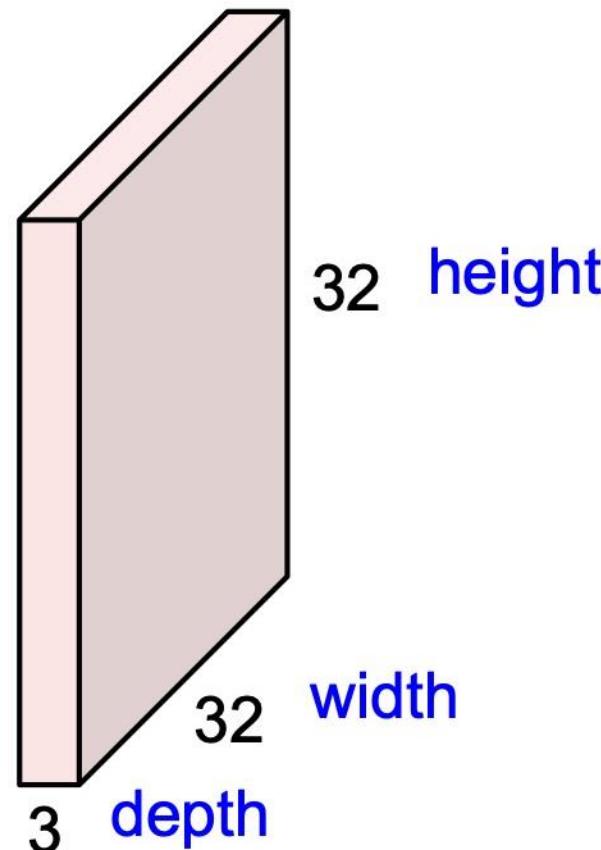
CNN 파트

Chapter 4 | CNN 동작 과정

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

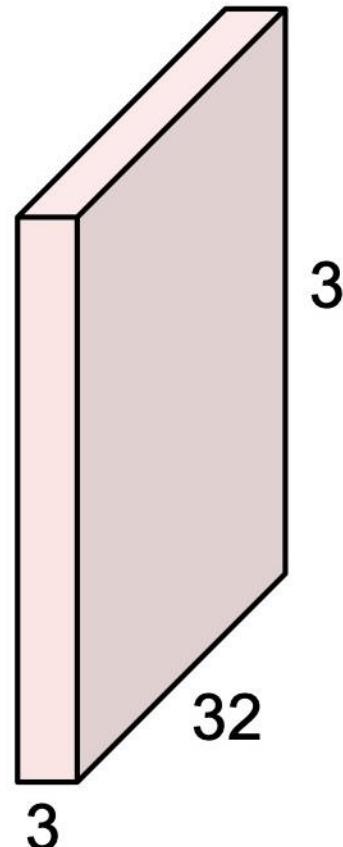
Chapter 4 | Convolutional Layer

32x32x3 image -> preserve spatial structure

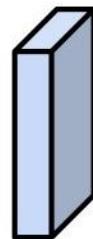


Chapter 4 | Convolutional Layer

32x32x3 image

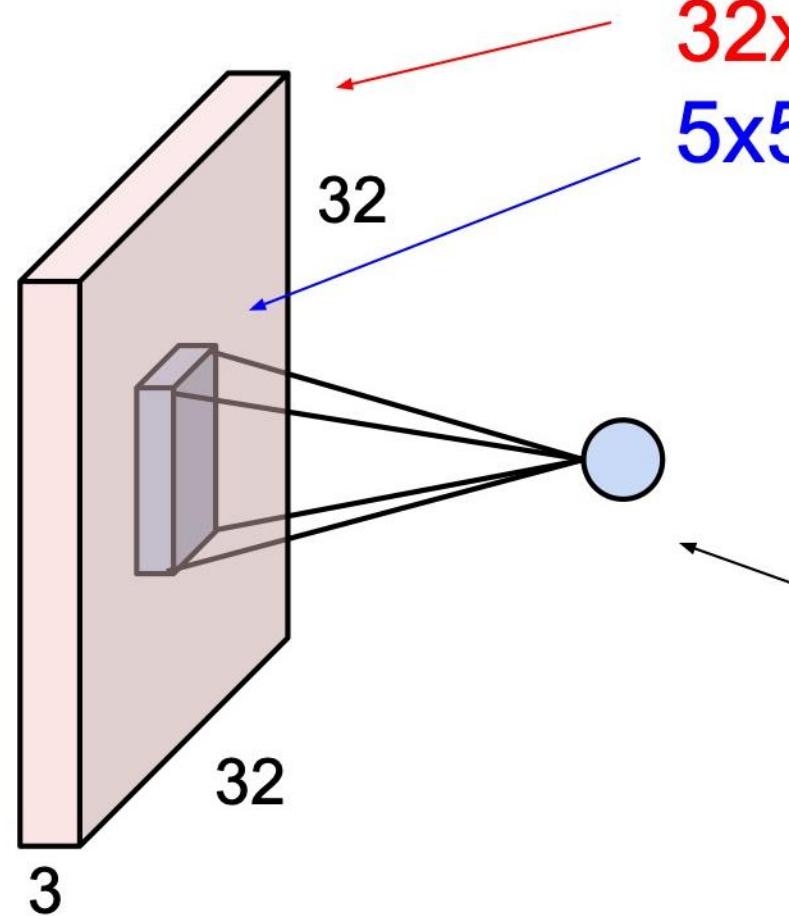


5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Chapter 4 | Convolutional Layer



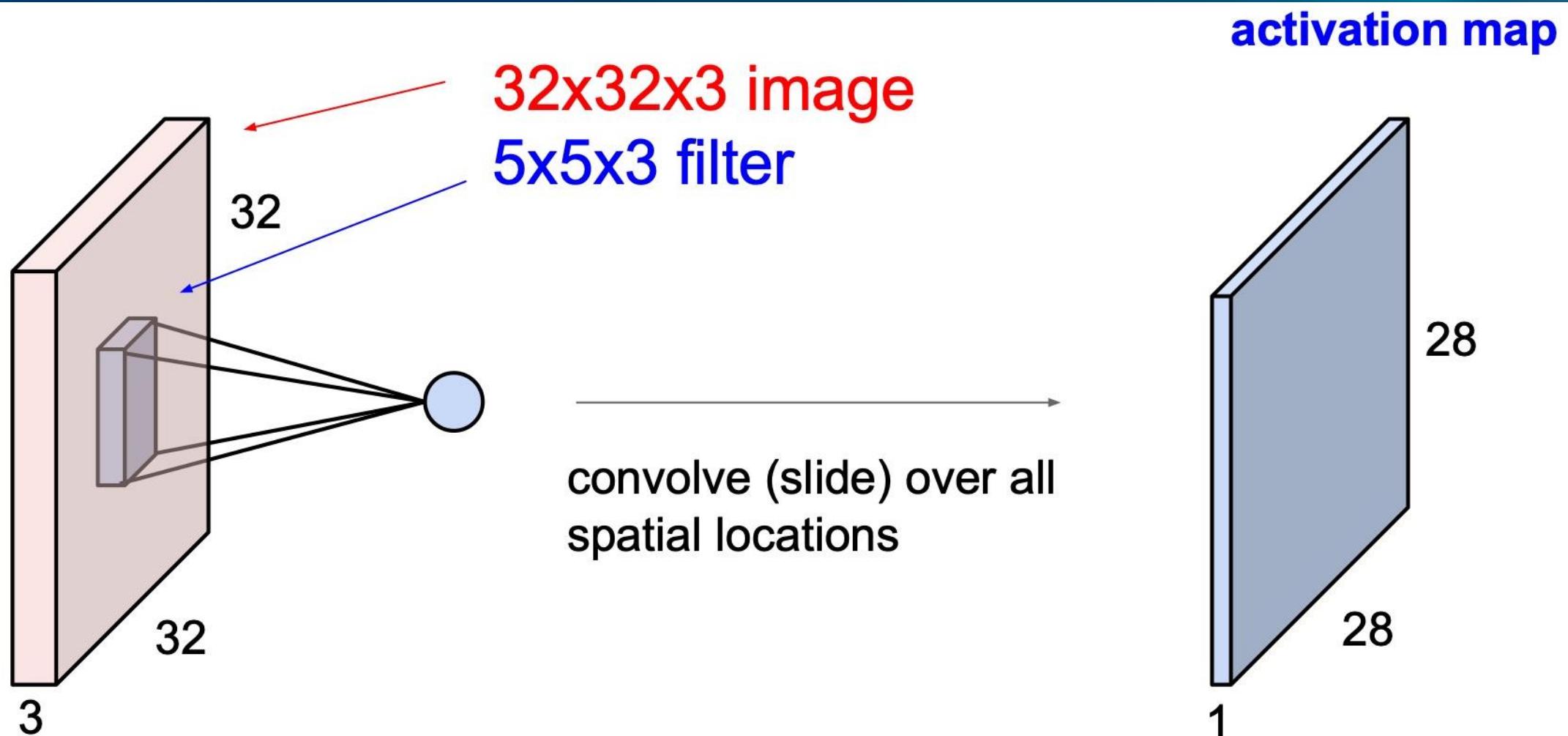
32x32x3 image
5x5x3 filter w

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix}$$

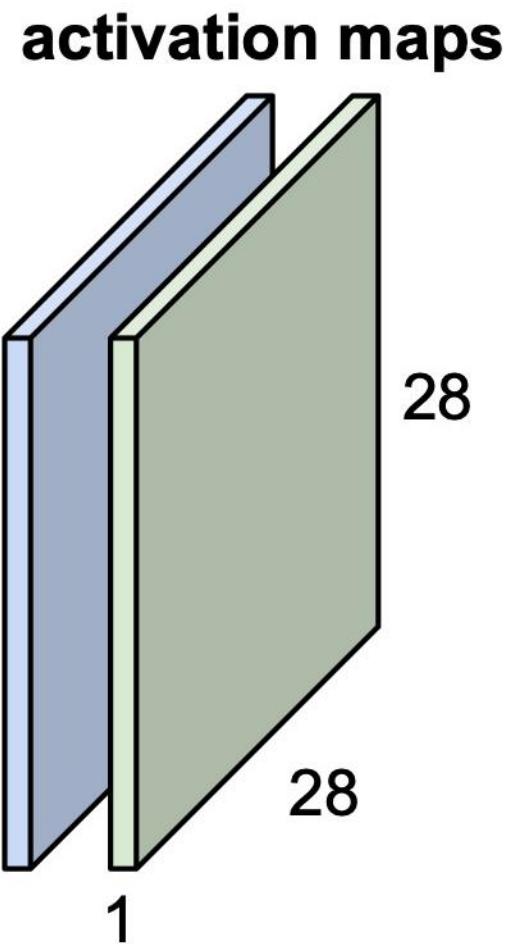
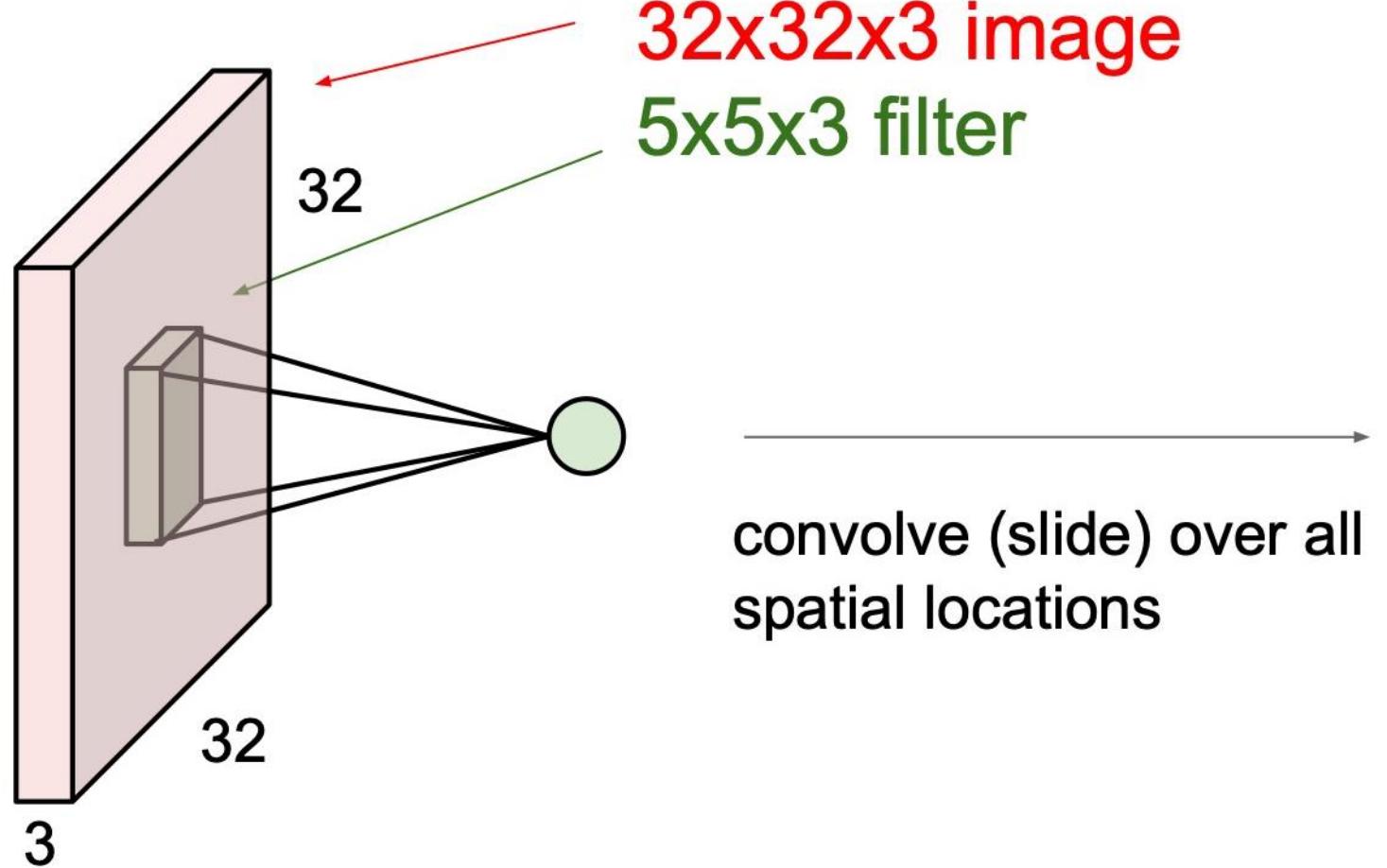
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. $5 \times 5 \times 3 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

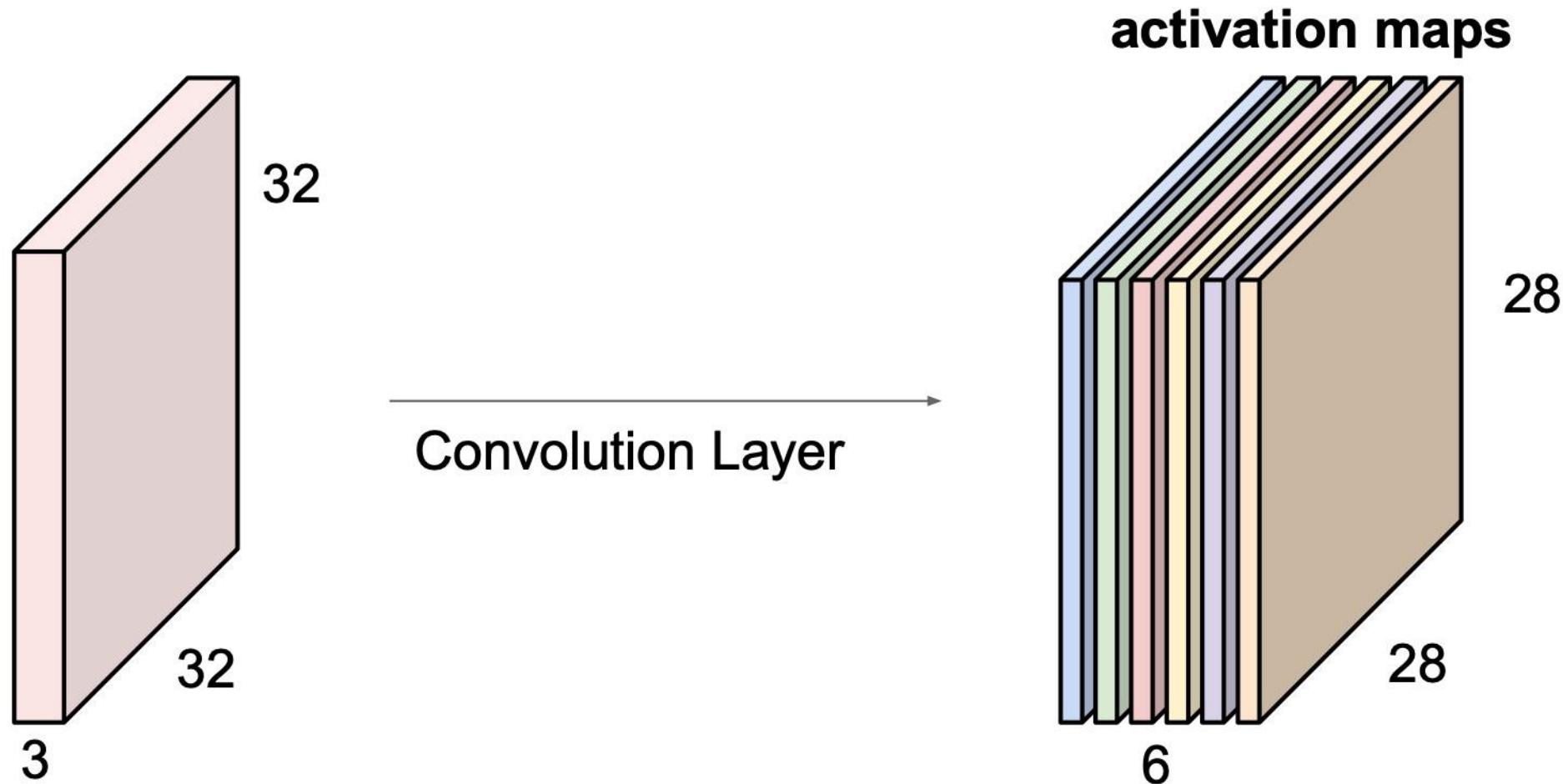
Chapter 4 | Activation Map



Chapter 4 | Activation Map



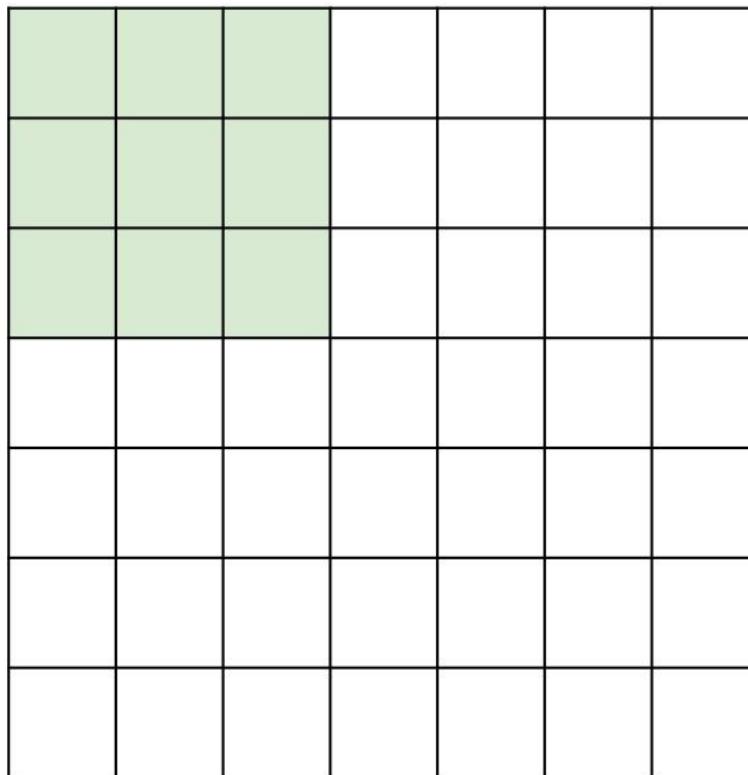
Chapter 4 | Activation Map



Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

7



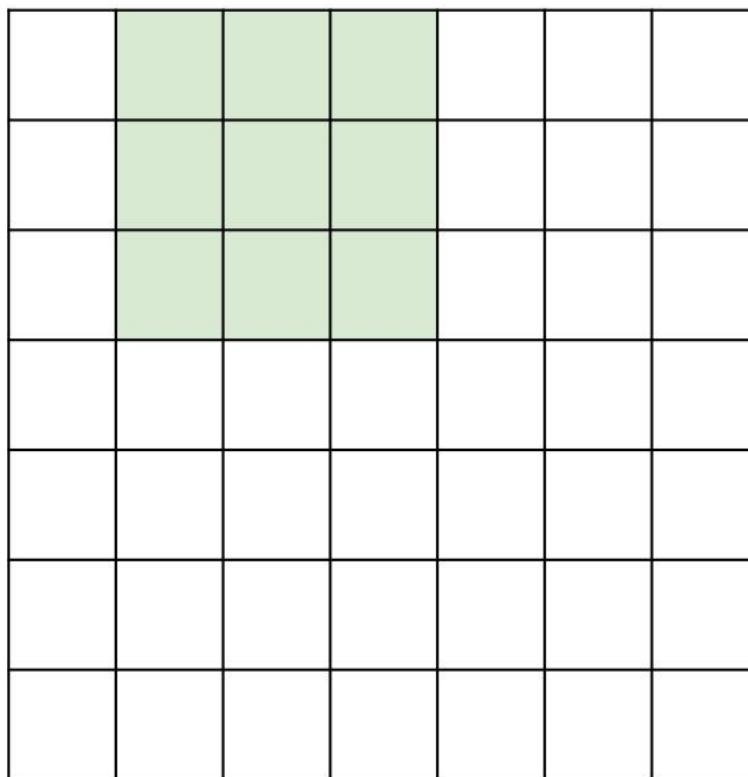
7x7 input (spatially)
assume 3x3 filter

7

Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

7



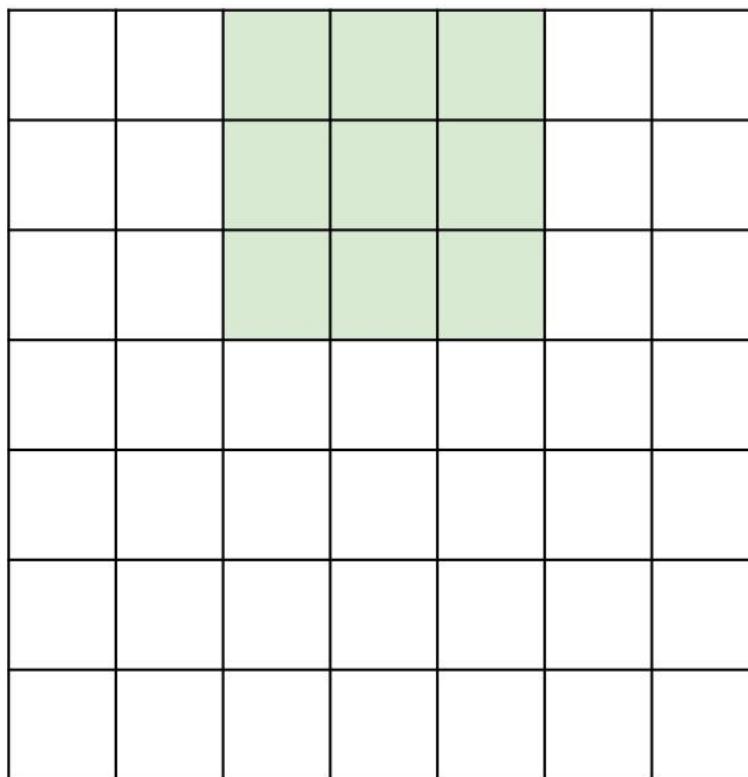
7x7 input (spatially)
assume 3x3 filter

7

Chapter 4| Activation Map in Toy Example

A closer look at spatial dimensions:

7



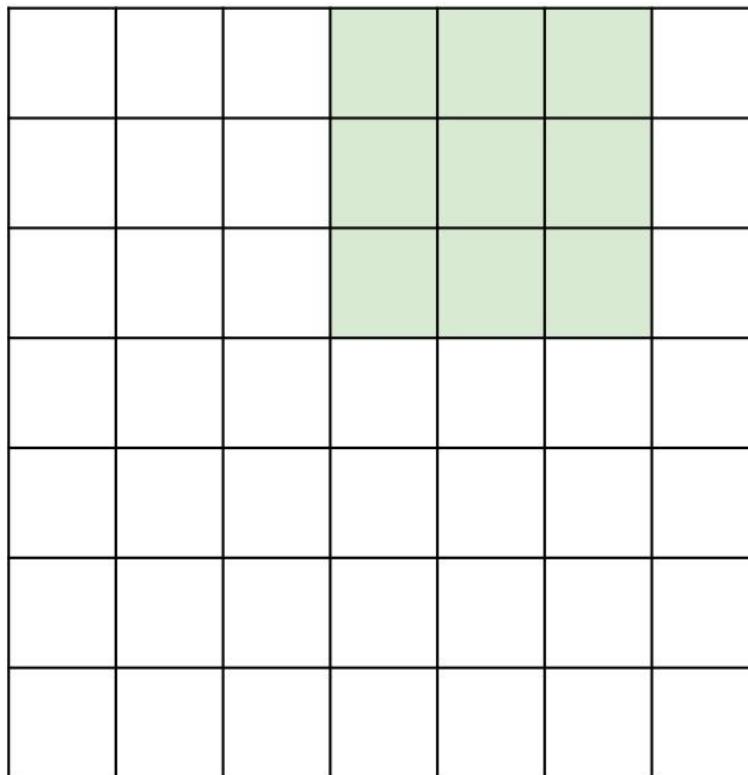
7x7 input (spatially)
assume 3x3 filter

7

Chapter 4| Activation Map in Toy Example

A closer look at spatial dimensions:

7



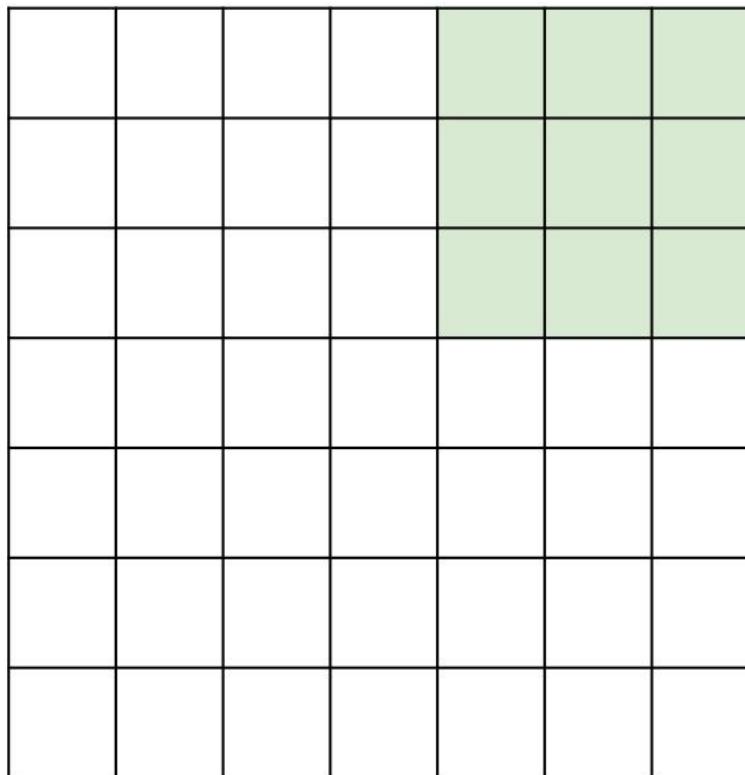
7x7 input (spatially)
assume 3x3 filter

7

Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

7

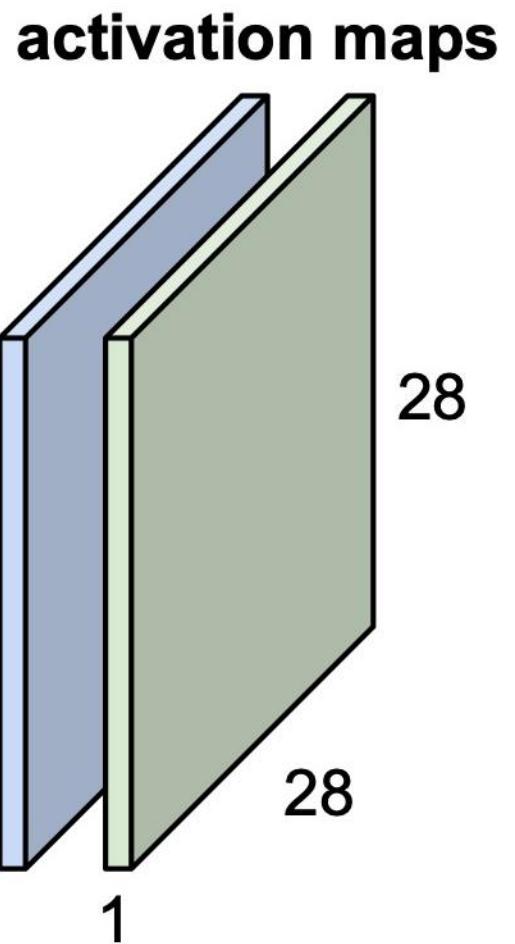
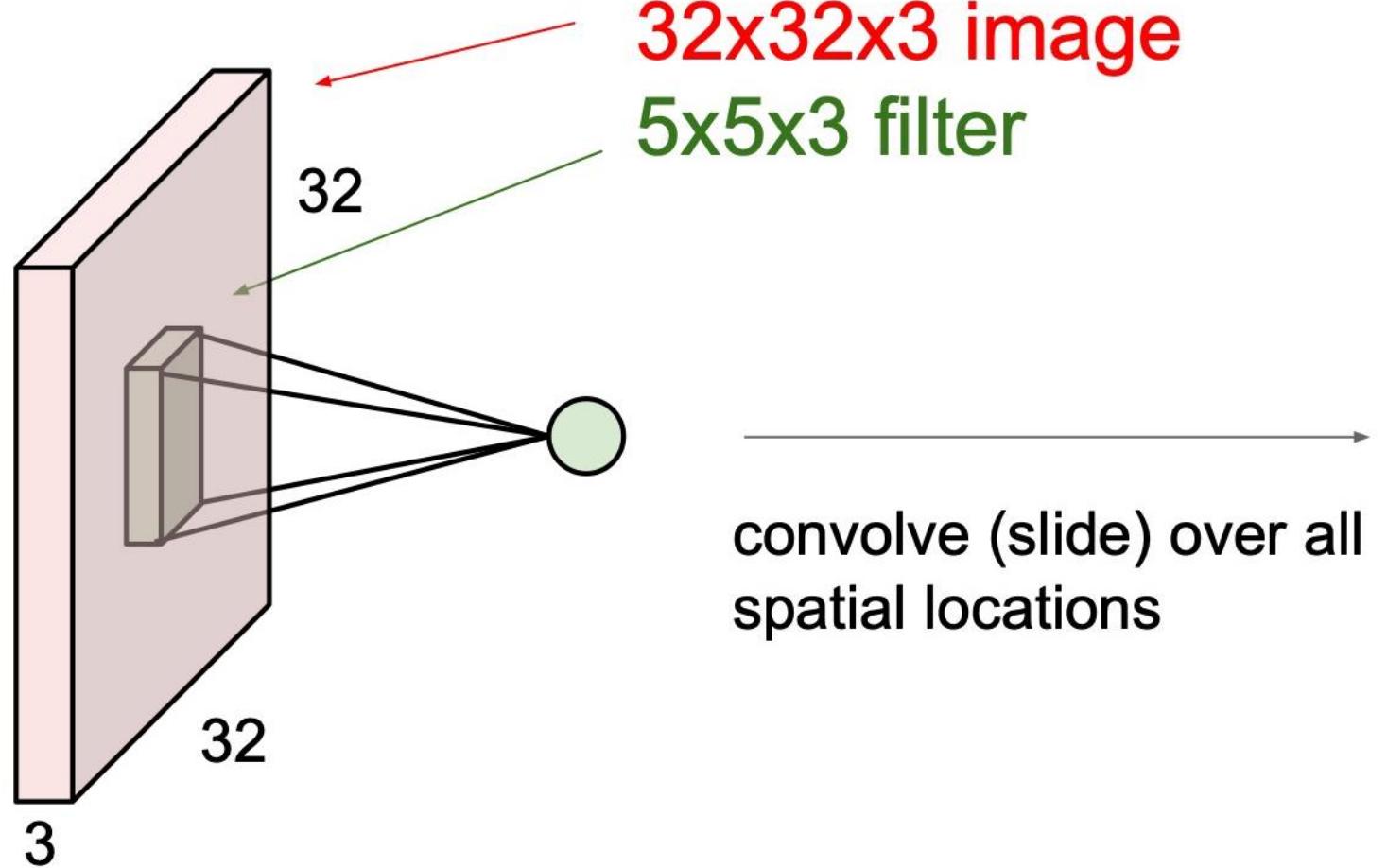


7x7 input (spatially)
assume 3x3 filter

7

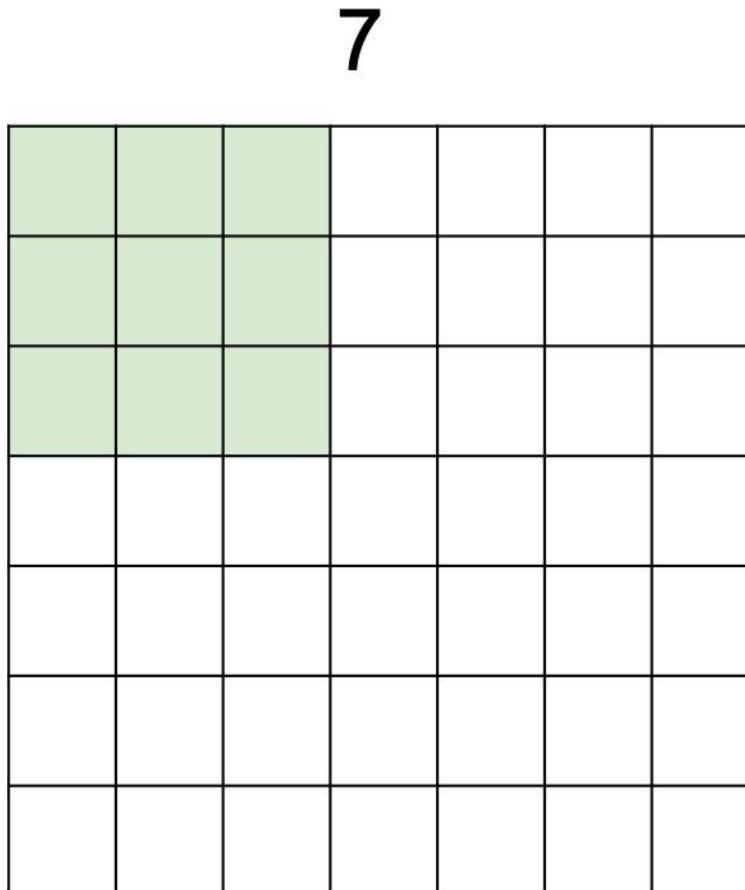
=> 5x5 output

Chapter 4 | Activation Map



Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

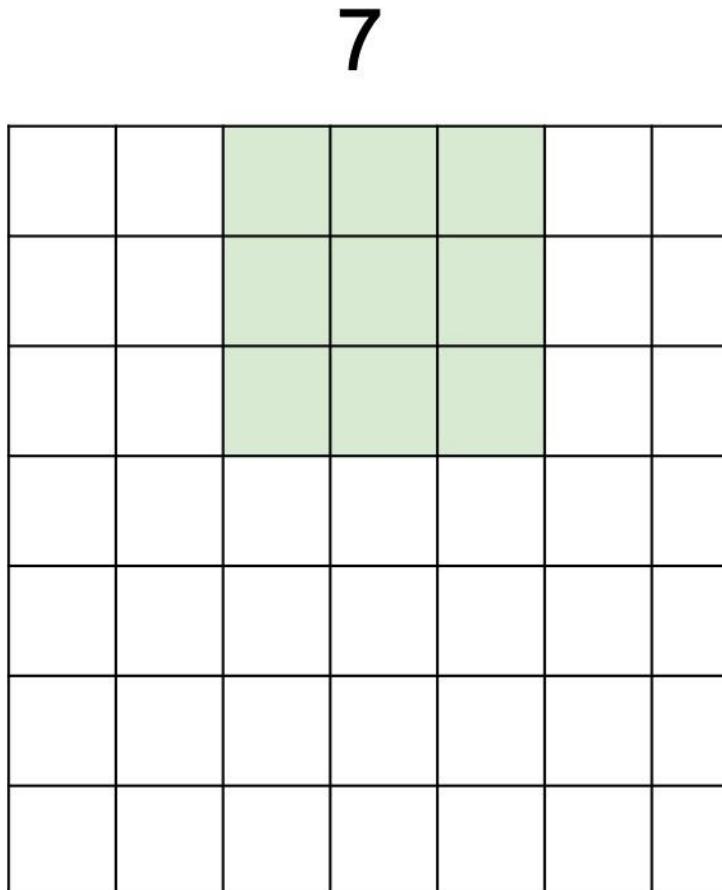


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

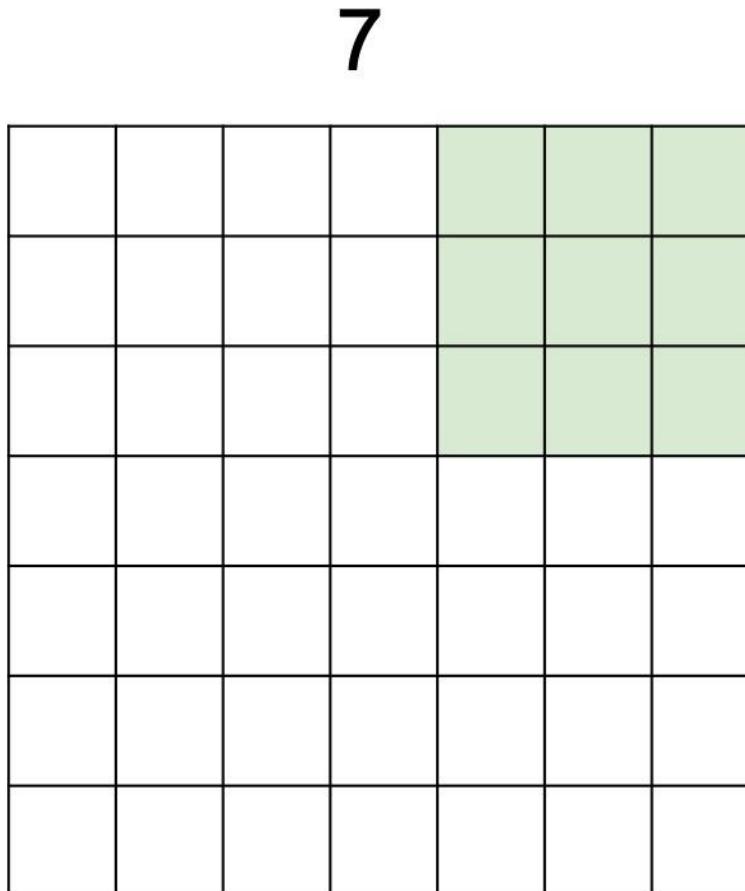


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Chapter 4 | Activation Map in Toy Example

A closer look at spatial dimensions:

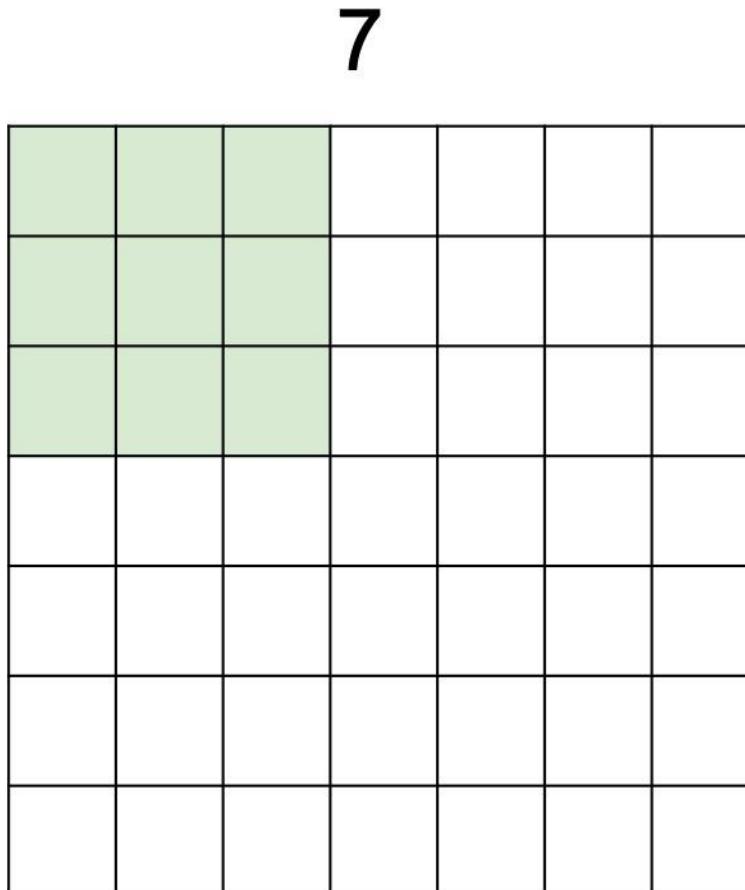


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Chapter 4 | Activation Map in Toy Example

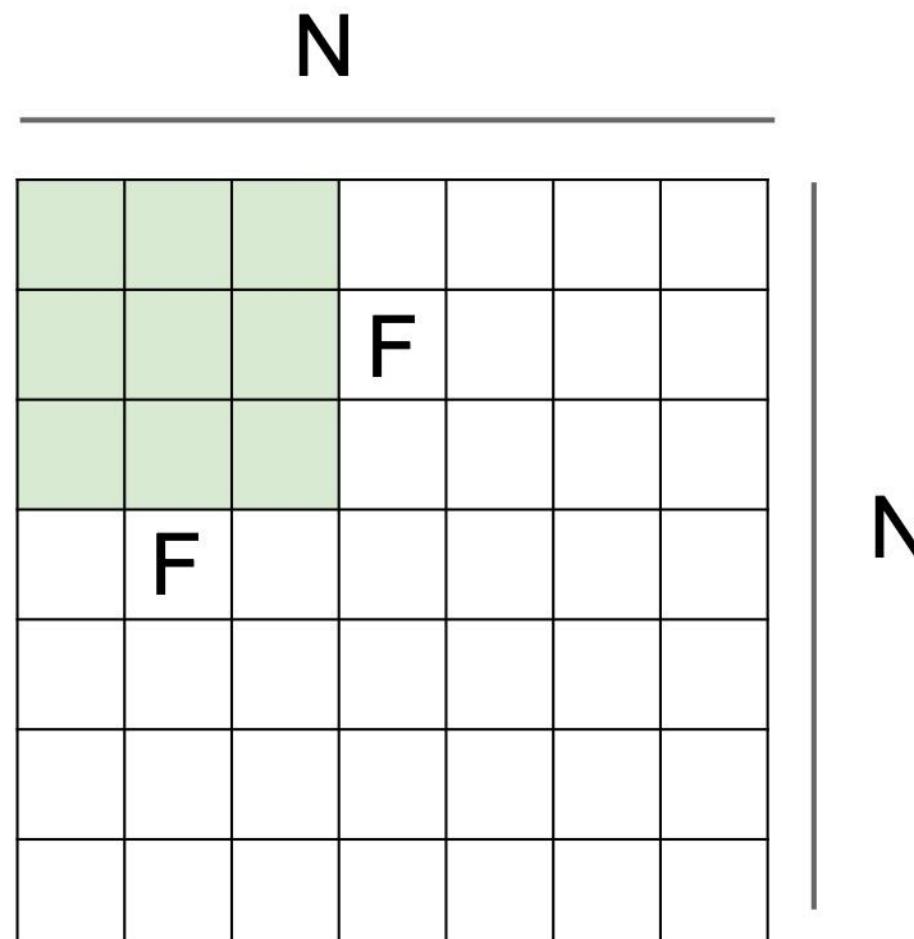
A closer look at spatial dimensions:



7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

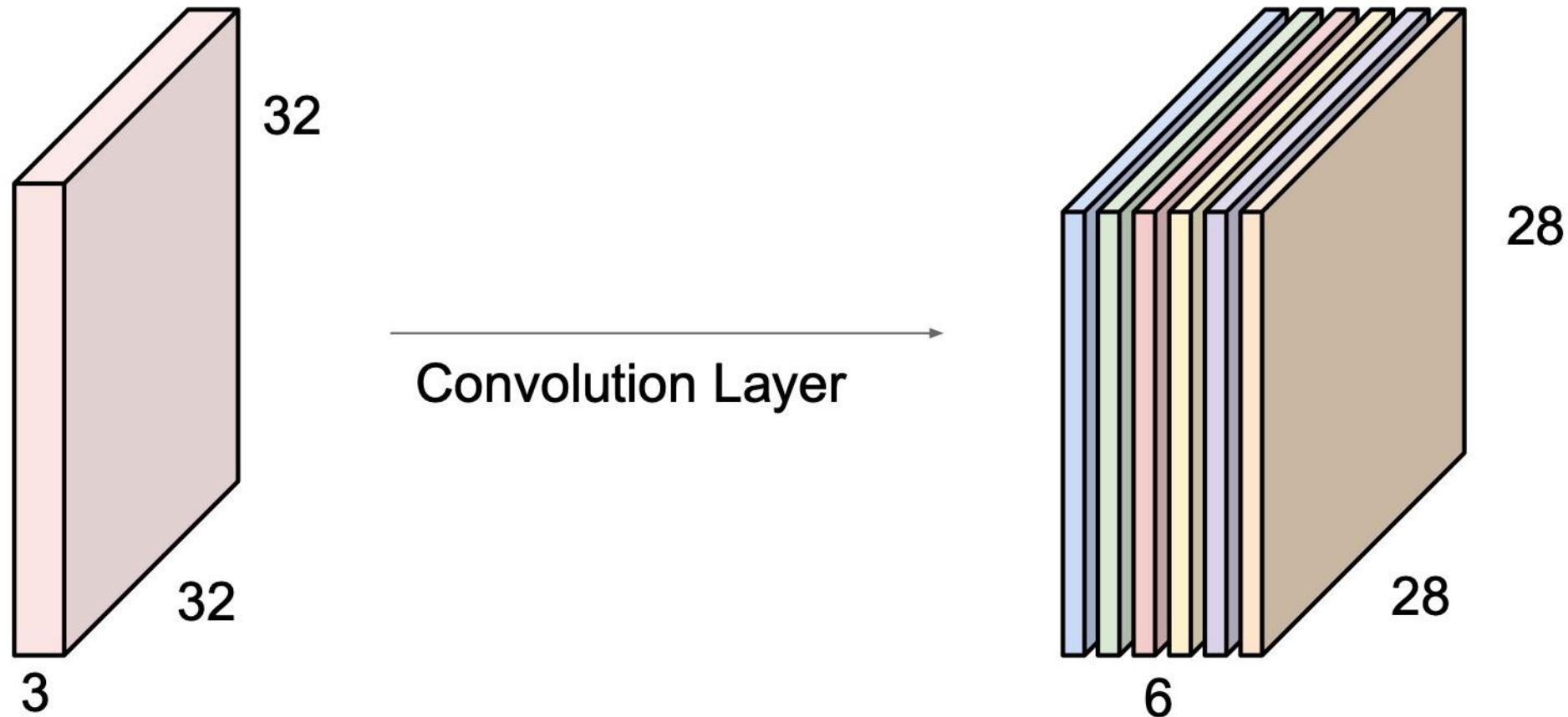
Chapter 4 | Activation Map in Toy Example



Output size:
 $(N - F) / \text{stride} + 1$

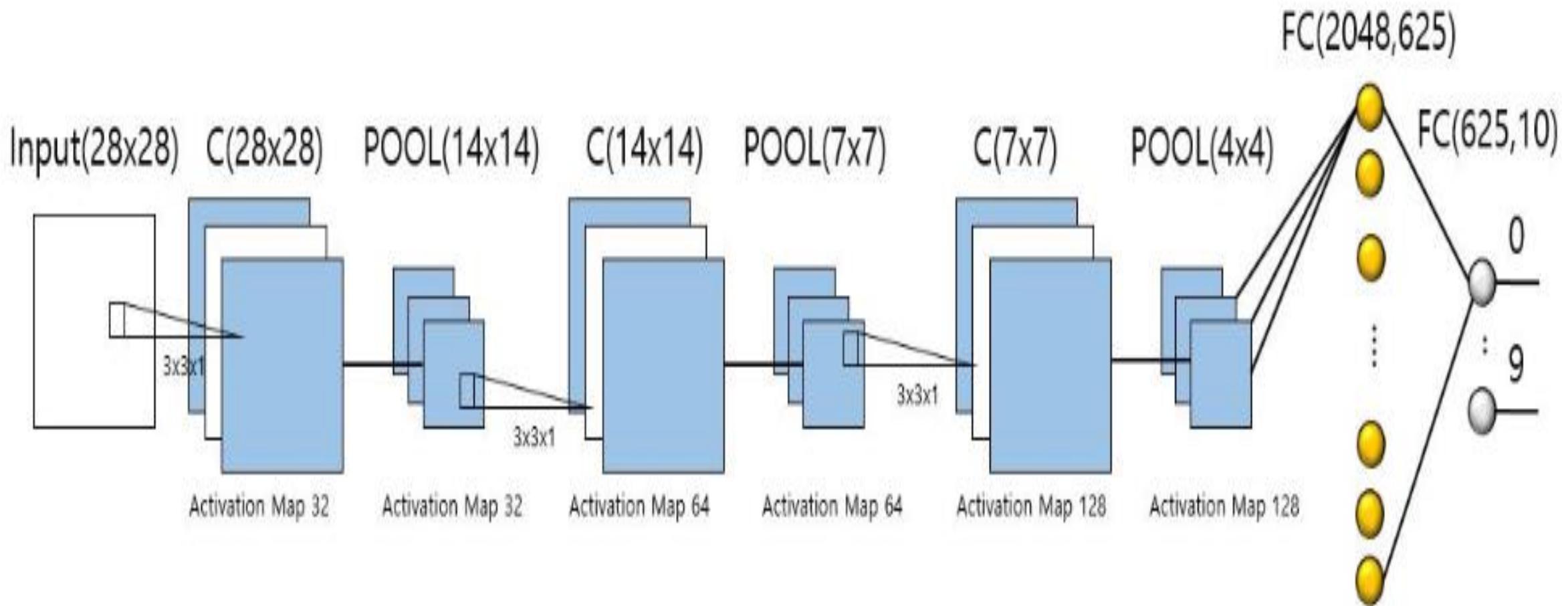
e.g. $N = 7$, $F = 3$:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33 : \backslash$

Chapter 4 | Activation Map



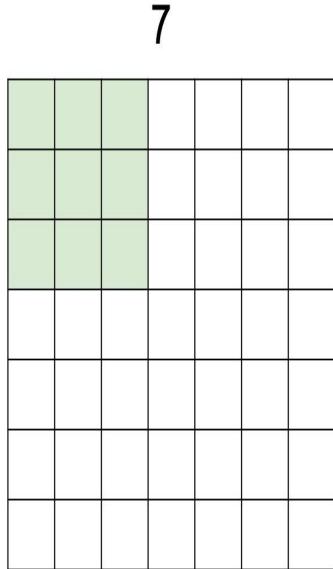
Chapter 4 | 중간 정리

- X라는 input을 사진 모양을 유지한 채로 받는다
- 작은 필터를 stride 단위만큼 움직이며 값을 뽑아내어 activation map을 하나 만든다.
- 또 같은 크기의 필터(다른 초기값)들 똑같은 stride 단위만큼 움직이며 값을 뽑아내어 activation map들을 만든다.

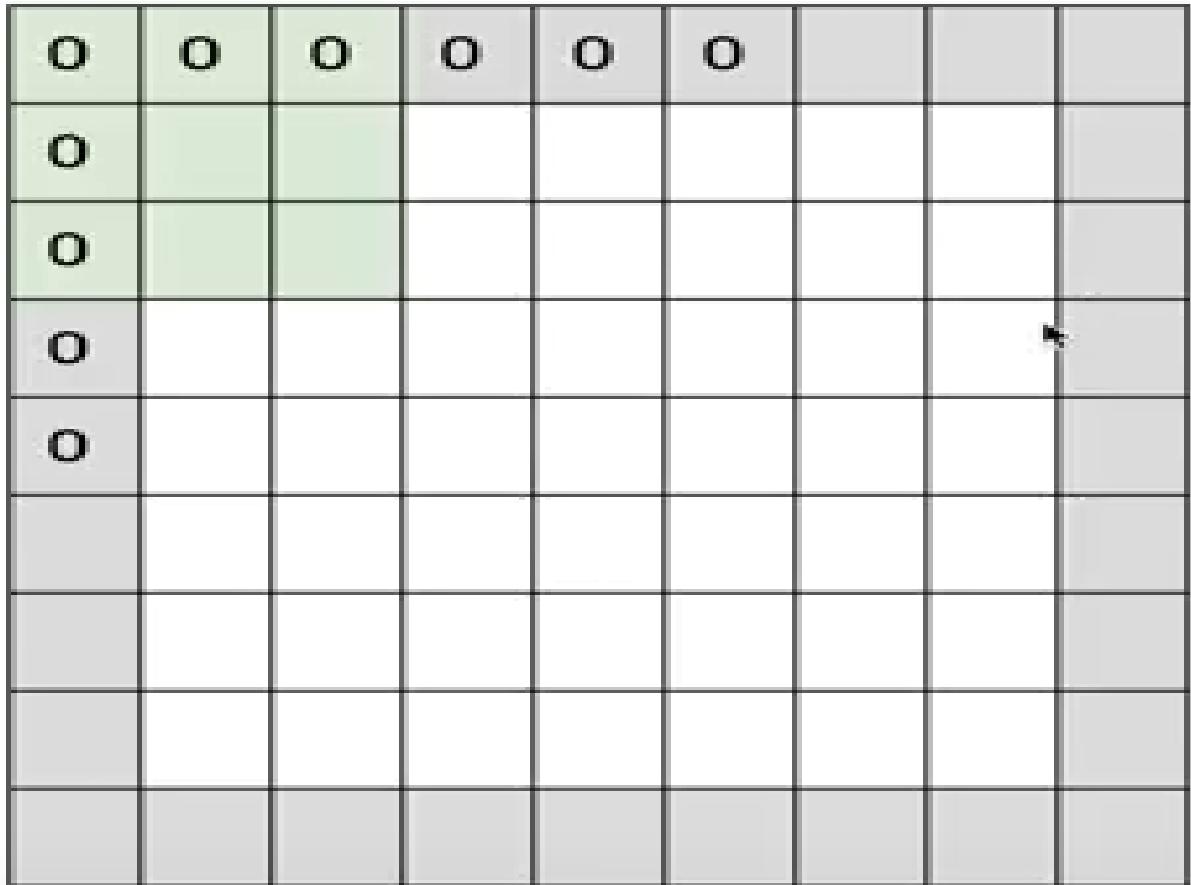


Chapter 4 | Padding

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

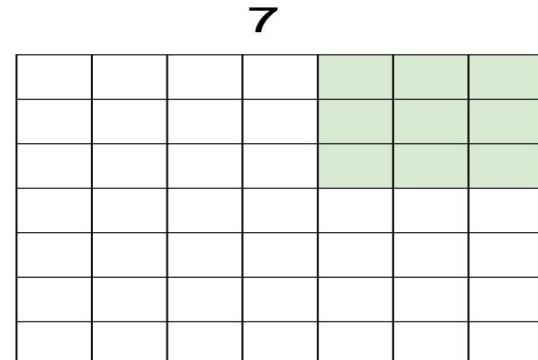


■ Padding 왜 할까?

- 1. 컴퓨터에게 모서리 부분이라는 것을 알려준다
- 2. Output 크기가 줄어드는 것을 막아준다.

■ 28*28 input을 3*3 filter로 처리하면 26*26가 나와야 하나
padding='SAME' 의 코드를 사용하여 28*28이 나오도록 함

A closer look at spatial dimensions:

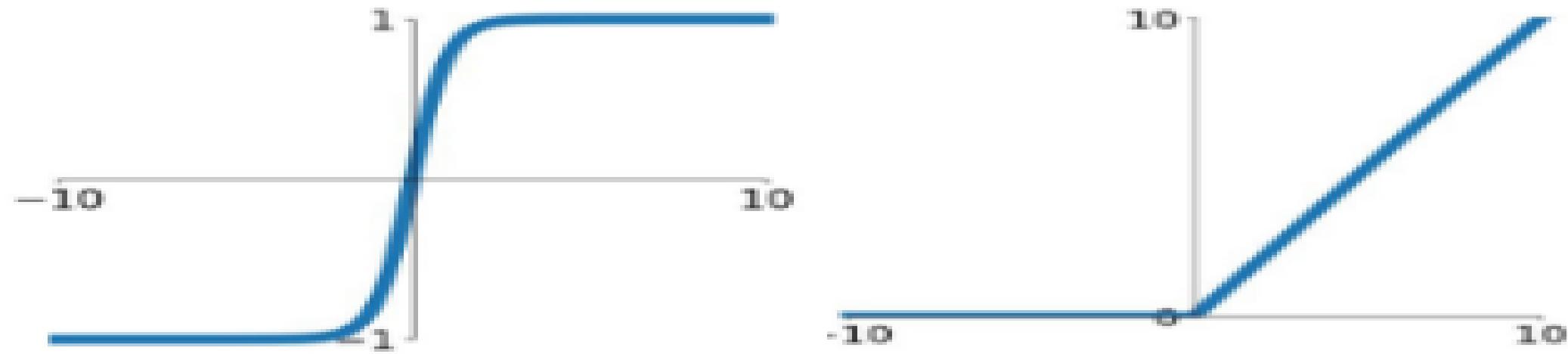


7

7x7 input (spatially)
assume 3x3 filter

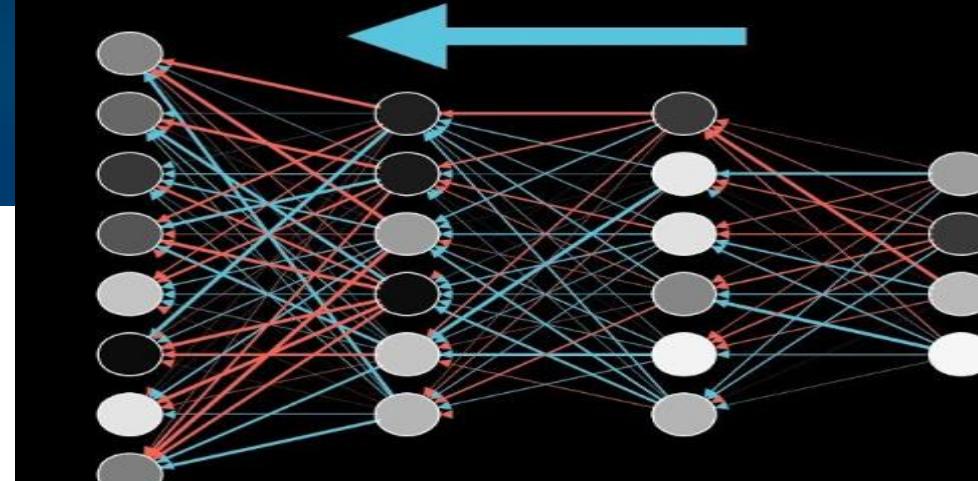
=> **5x5 output**

Chapter 4 | RELU(Rectified linear unit)함수



■ 왜 sigmoid 말고 RELU를 쓸까?

Backpropagation

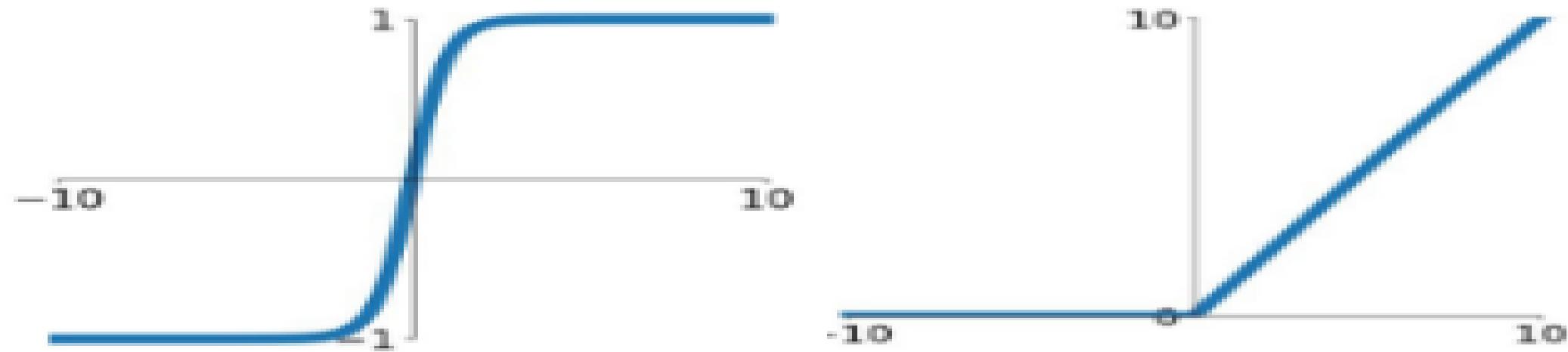


■ CNN은 층이 여러 개로 깊다.

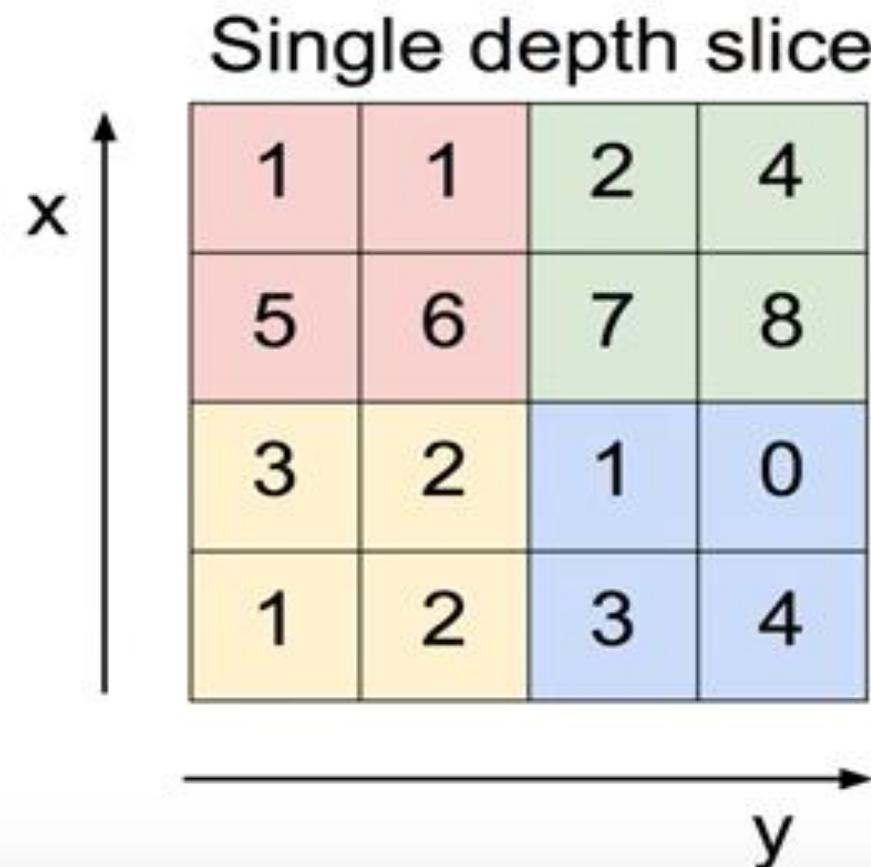
그와 동시에 뒤에서부터 거꾸로 가중치를 조절하는 역전파 방법을 쓰기 때문에 앞 노드들의 영향력이 매우 줄어든다.
이를 조금이라도 방지하고자 큰 입력 값이 들어오면 큰 값을 뒤로 보내게 하자는 취지.

→ 실증적으로 Alexnet이 성능 대박 향상

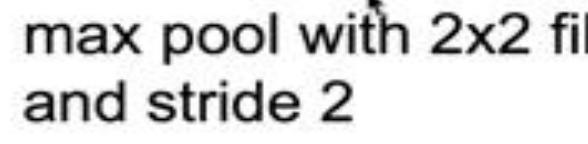
Chapter 4 | RELU(Rectified linear unit)함수



Chapter 4 | Max pooling

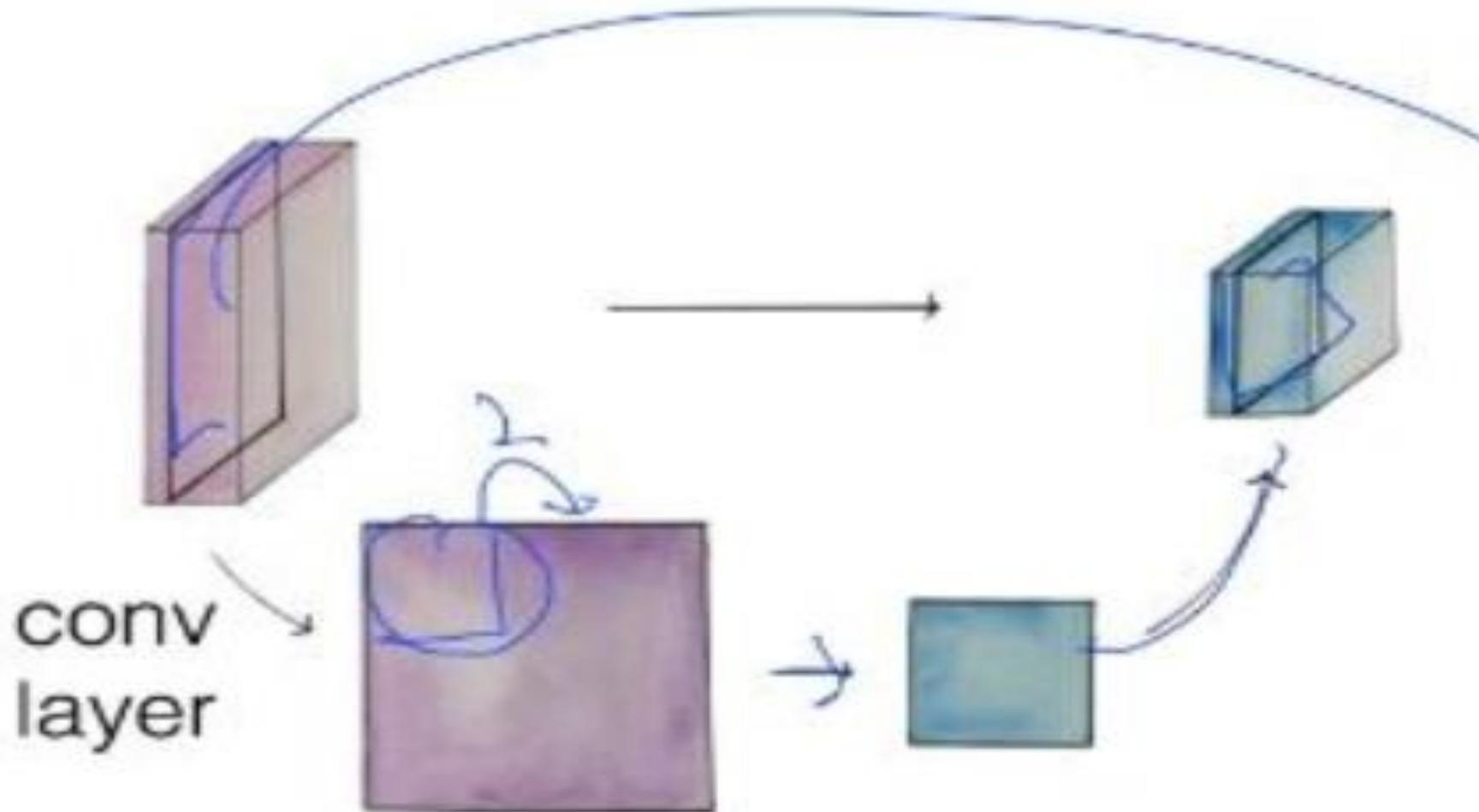


max pool with 2x2 filters
and stride 2



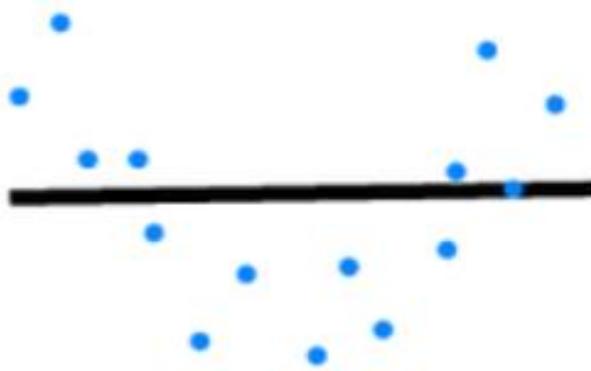
6	8
3	4

Chapter 4 | Max pooling

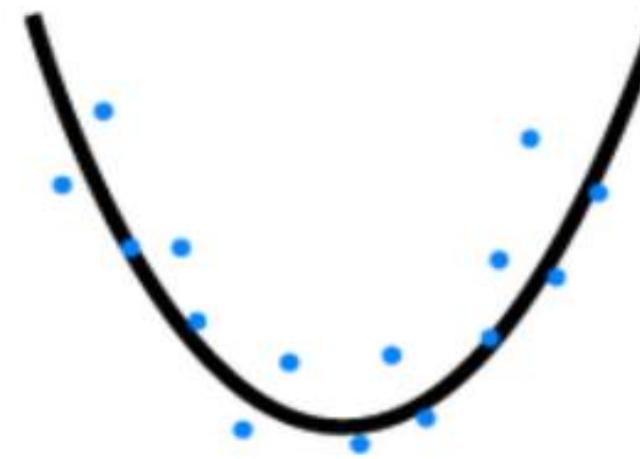


■ Max pooling 왜 할까?

■ 실증적으로 가장 CNN에 효과적이라는 것이 검증되었기 때문
→ 학자들은 이미지의 가장 주목할 특징을 잘 짊어 내기 때문에
라 '예상'



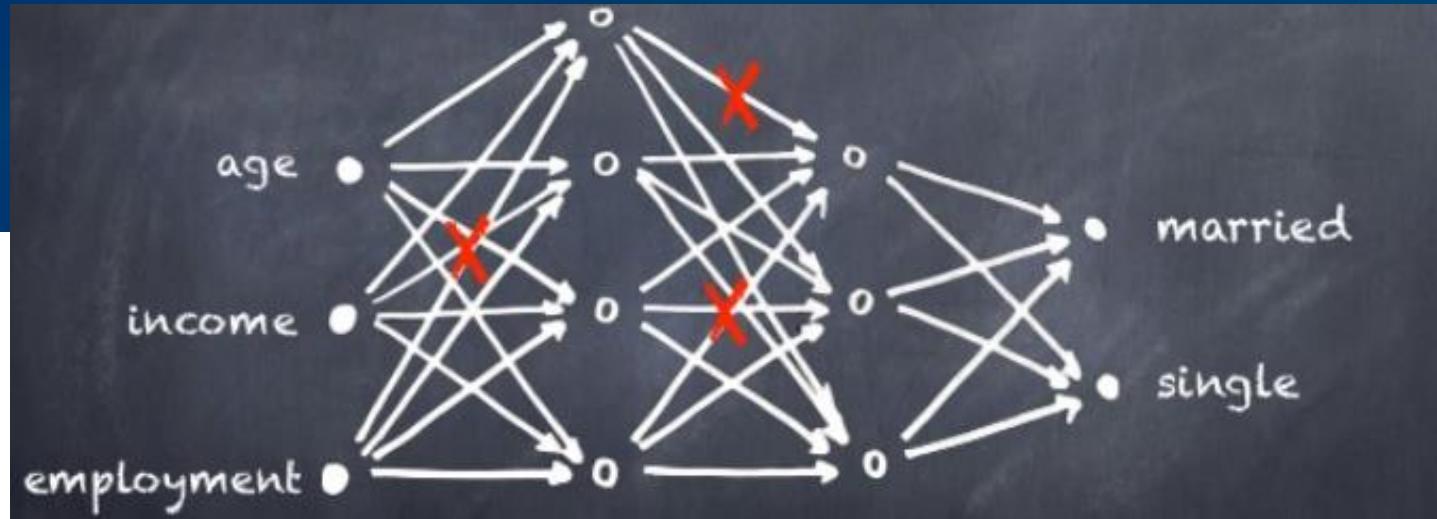
Underfitting



Desired



Overfitting

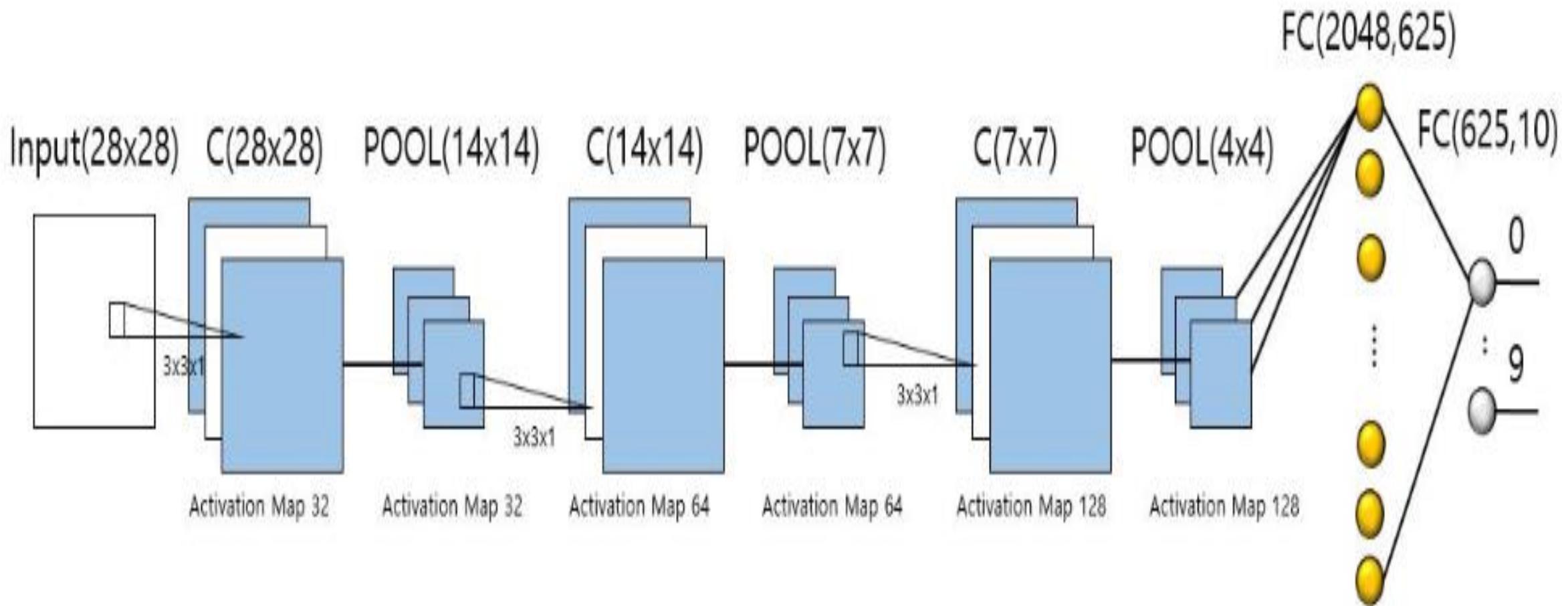


■ Keep_prob(드랍아웃율)로 오버피팅 방지하기 위해 사용.
노드들을 각 층에서 일정확률로 활용하지 않는 것을 말한다.

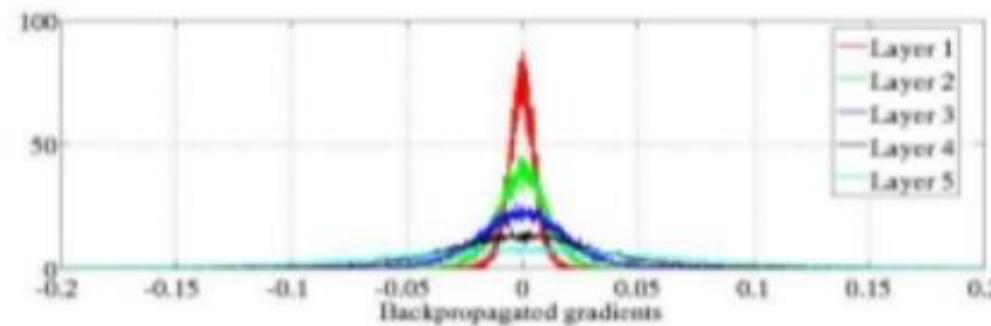
단, 최종 테스트시에는 모든 노드들 동원하여 예측해야하기 때문에 1로 해야한다.

Chapter 4 | 중간 정리

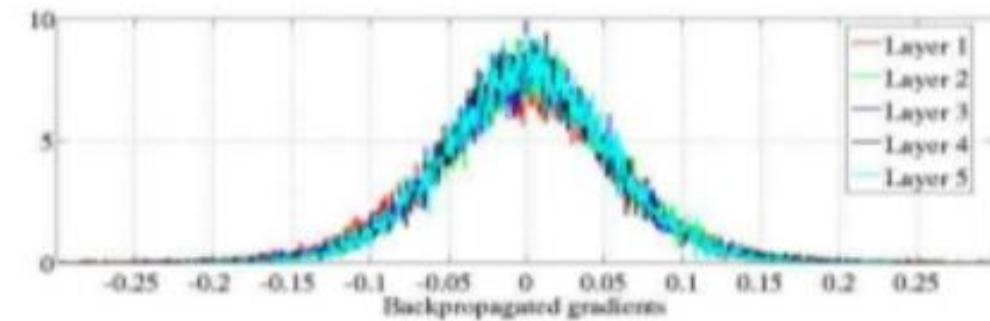
- X라는 input을 사진 모양을 유지한 채로 받는다
- 작은 필터를 stride 단위만큼 움직이며 값을 뽑아내어 activation map을 하나 만든다.
- 또 같은 크기의 필터(다른 초기값)들 똑같은 stride 단위만큼 움직이며 값을 뽑아내어 activation map들을 만든다.
- Padding,relu,max_pooling을 각각의 목적을 위해 사용한다.
- 이 과정을 3번 반복하여 3개의 층을 만든다.



Smart initialization



standard



"Xavier"

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

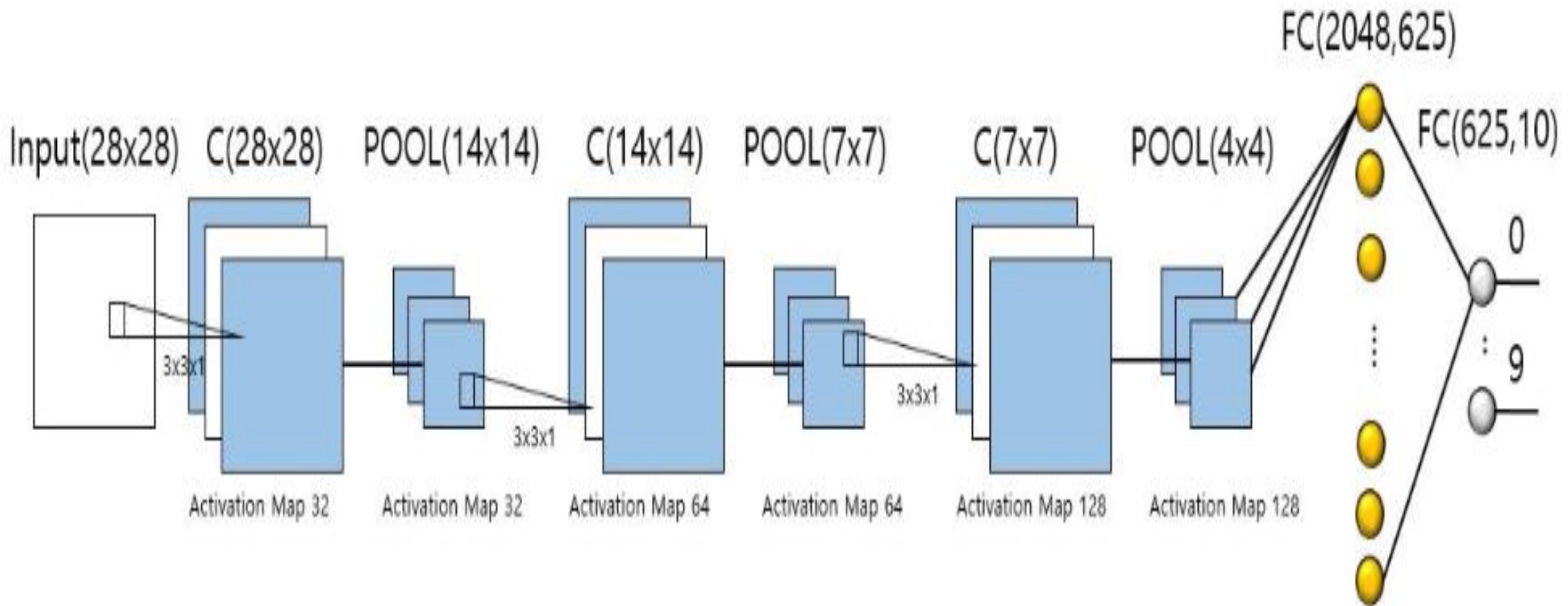
■ Xavier Initialize 왜 할까?

■ 통계 전문가의 대답:Xavier initializer는 입력값과 출력값 사이의 난수를 선택해서 입력값의 제곱근으로 나눈다. →???

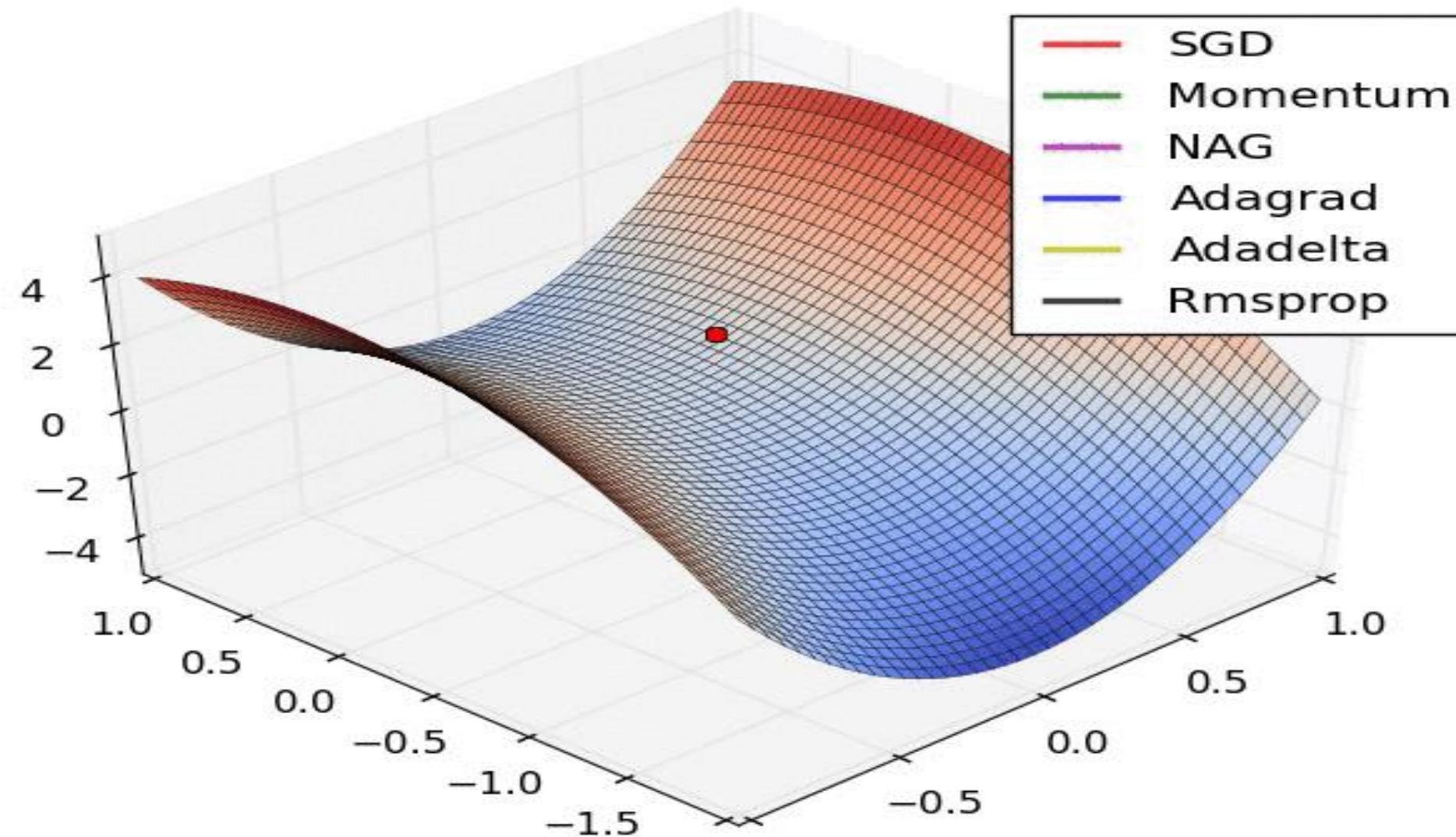
■ 쉬운 말로 한 대답: 기존의 정규 방법은 처음 값을 대충대충 던져주고 컴퓨터 한테 다 맡기는 유형이었다면, initializer를 쓰면 조금이라도 초기 방향성을 제시해주고 맡기는 것.

Chapter 4 | 중간 정리

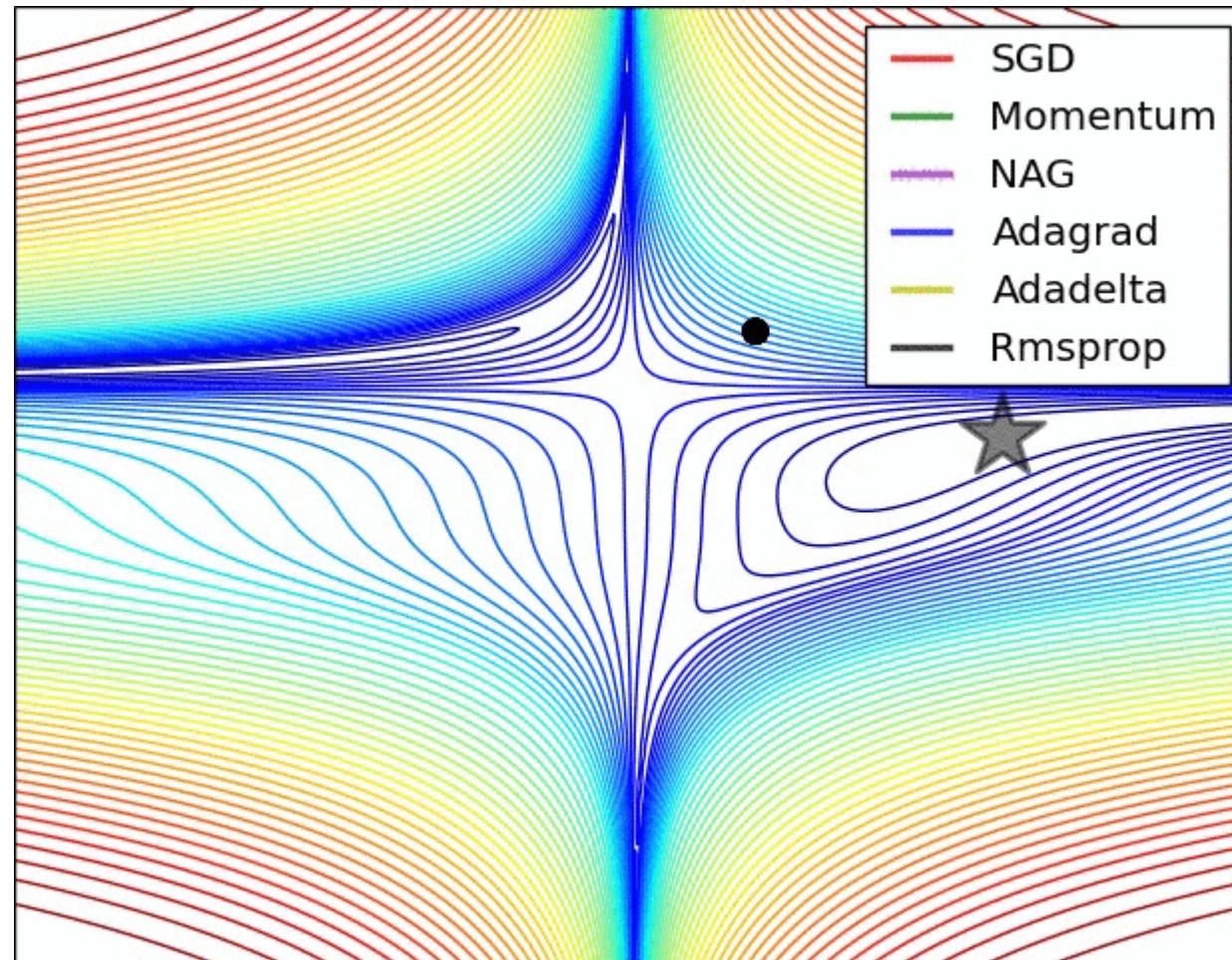
- X라는 input을 사진 모양을 유지한 채로 받는다
- 작은 필터를 stride 단위만큼 움직이며 값을 뽑아내어 activation map을 하나 만든다.
- 똑같은 크기의 필터(다른 초기값)들 똑같은 stride 단위만큼 움직이며 값을 뽑아내어 activation map들을 만든다.
- Padding,relu,max_pooling을 각각의 목적을 위해 사용한다.
- 이 과정을 3번 반복하여 3개의 층을 만든다.
- 사진 모양을 한 줄로 쪽펴서 우리가 처음 배웠던 기본 NN을 사용한다.



Chapter 4 | Adam optimizer



Chapter 4 | Adam optimizer



Chapter 4 | 결과

```
Learning started. It takes sometime.  
Epoch: 0001 cost = 0.371361985  
Epoch: 0002 cost = 0.100482825  
Epoch: 0003 cost = 0.074158054  
Epoch: 0004 cost = 0.062305827  
Epoch: 0005 cost = 0.054155033  
Epoch: 0006 cost = 0.047632870  
Epoch: 0007 cost = 0.042911727  
Epoch: 0008 cost = 0.041173837  
Epoch: 0009 cost = 0.037396228  
Epoch: 0010 cost = 0.035396742  
Epoch: 0011 cost = 0.032740152  
Epoch: 0012 cost = 0.030919549  
Epoch: 0013 cost = 0.029136615  
Epoch: 0014 cost = 0.027180246  
Epoch: 0015 cost = 0.026203343  
Learning Finished!  
Accuracy: 0.9937  
Label: [5]  
Prediction: [5]
```

■ 호기심과 정리

부록: 추천할 만한 강의나 컨텐츠

- Youtube:

- 모두를 위한 딥러닝
- 3BLUE1BROWN
- Andrew ng의 강의

기타:

Ratsgo's blog <https://ratsgo.github.io/>

■ 긴 시간 끝까지 들어주셔서 정말 감사합니다...!



E.O.D