ITE4052 Computer Vision

Deep Learning Review

Dong-Jin Kim

Spring 2024

HANYANG UNIVERSITY

※The parts of slides are from CS231n

Image Classification

The **simplest** task in visual recognition!

(assume given set of discrete labels)
{dog, cat, truck, plane, ...} ← Image class

→ cat

Result of Image classification

Image Classification Challenges

Viewpoint variation

Scale variation

Deformation

Occlusion

Illumination conditions

Background clutter

Intra-class variation

Data-Driven Approach

→ All kinds of Image classification challenges을 해결할 수 있다.

1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

Example training set

Parametric Approach : **Linear Classifier**

$f(x, W) = Wx$ (+b)

3072x1

10x1

10x1

10x3072

10 numbers, indicating class scores

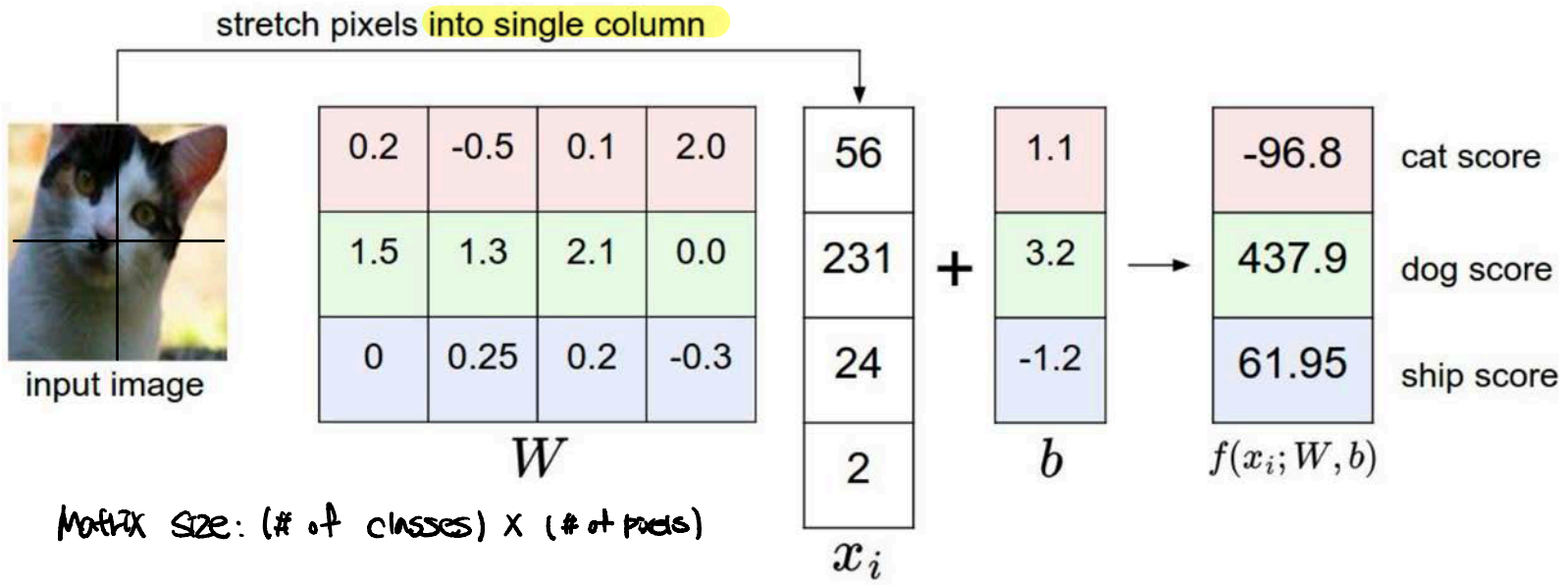
[32x32x3] = 3072
array of numbers 0...1

parameters, or "weights"

3072-dimension vector

Parametric Approach : **Linear Classifier**

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Interpreting a Linear Classifier: Visual

Linear Classifier Formulation

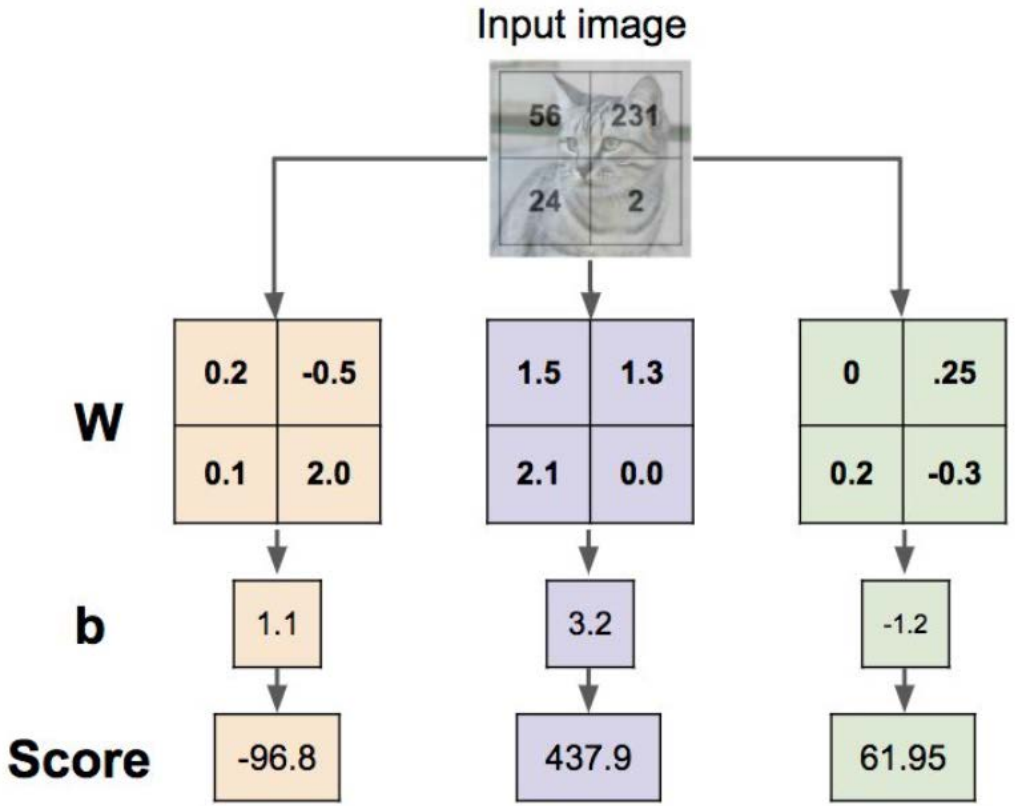
$$f(x_i, W, b) = Wx_i + b$$

$f_i = W_i x + b_i$

Similarity

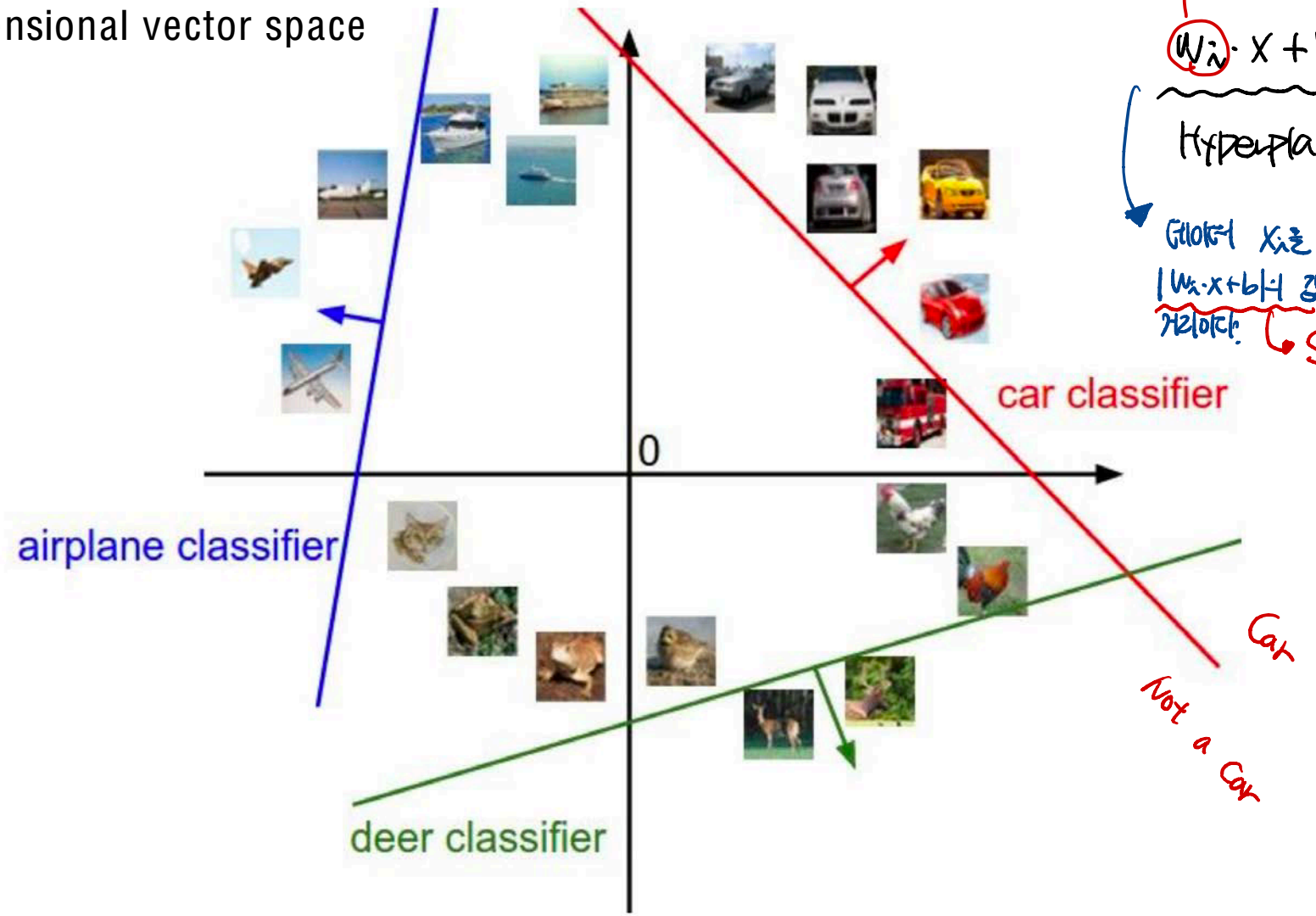
Example trained **weights** of a linear classifier trained on CIFAR-10:

Prototype



Interpreting a Linear Classifier: Geometric

In 32x32x3=3072 dimensional vector space



Class 하나 → $W \in R^{1\times 3072}$

$W_i \cdot x + b = 0$

Hyperplane

입력된 x_i를 통해서,
$W_i \cdot x + b$와 얼마나 떨어진 Hyperplane과의 거리인가? → Score

Car

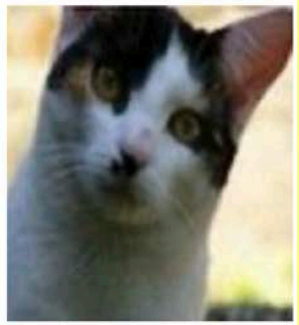
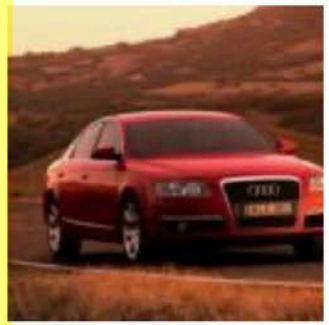
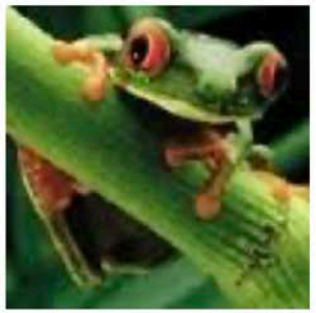
Not a Car

Training a Linear Classifier

Suppose: 3 training examples, 3 classes.
With some W, the scores are:

$$f(x, W) = Wx$$

3x30172

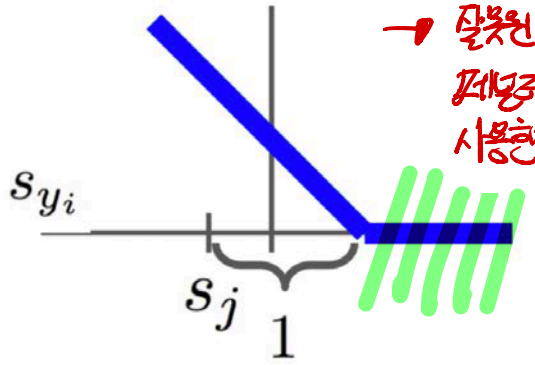




cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

1. **Multiclass SVM loss:**

"Hinge loss"

→ 잘못된 예측을 강조하는 패널티를 부과하기 위해서 사용한다.



2. Softmax

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

Loss

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Wrong Answer

ex) Car - cat : $\max(0, 5.1 - 3.2 + 1) = 2.9$

Softmax Classifier

= Multinomial Logistic Regression

Softmax = $\frac{e^{s_k}}{\sum_j e^{s_j}}$ ← exponential

$\sum_j e^{s_j}$ Normalization

→ Softmax를 이용하여 구함.

Scores as **probabilities** (>=0, sum to 1) of the classes.

$$P(Y = k \mid X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

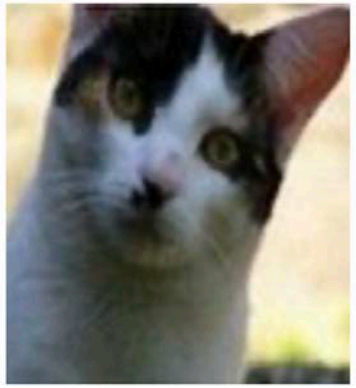
where $s = f(x_i; W)$

Y=k: 예측 / X=x_i: Data given / Softmax

정답이 1이기 때문에

Cat probability ↑

→ Cat probability ↓ and frog probability ↓



cat	**3.2**
car	5.1
frog	-1.7

Want to maximize the **log likelihood** of the correct class:

$$L_i = -\log P(Y = y_i \mid X = x_i)$$

→ 정답인 곳만 보겠다.

→ Minimize negative log likelihood

In summary :

$$L_i = -\log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

Softmax Classifier

= Multinomial Logistic Regression

Comparing with **ground truth** (one-hot vector)
:Kullback–Leibler (**KL**) **Divergence**

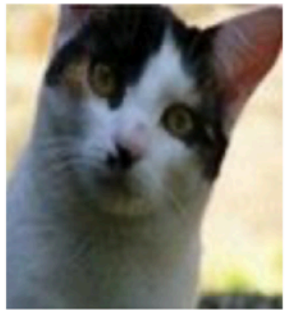
$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

one-hot vector

정답 class 에서만 P=[1,0,0]

$D_{KL}(P||Q)$

$= 1 \cdot \log(\frac{1}{0.13}) + 0 + 0$



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q

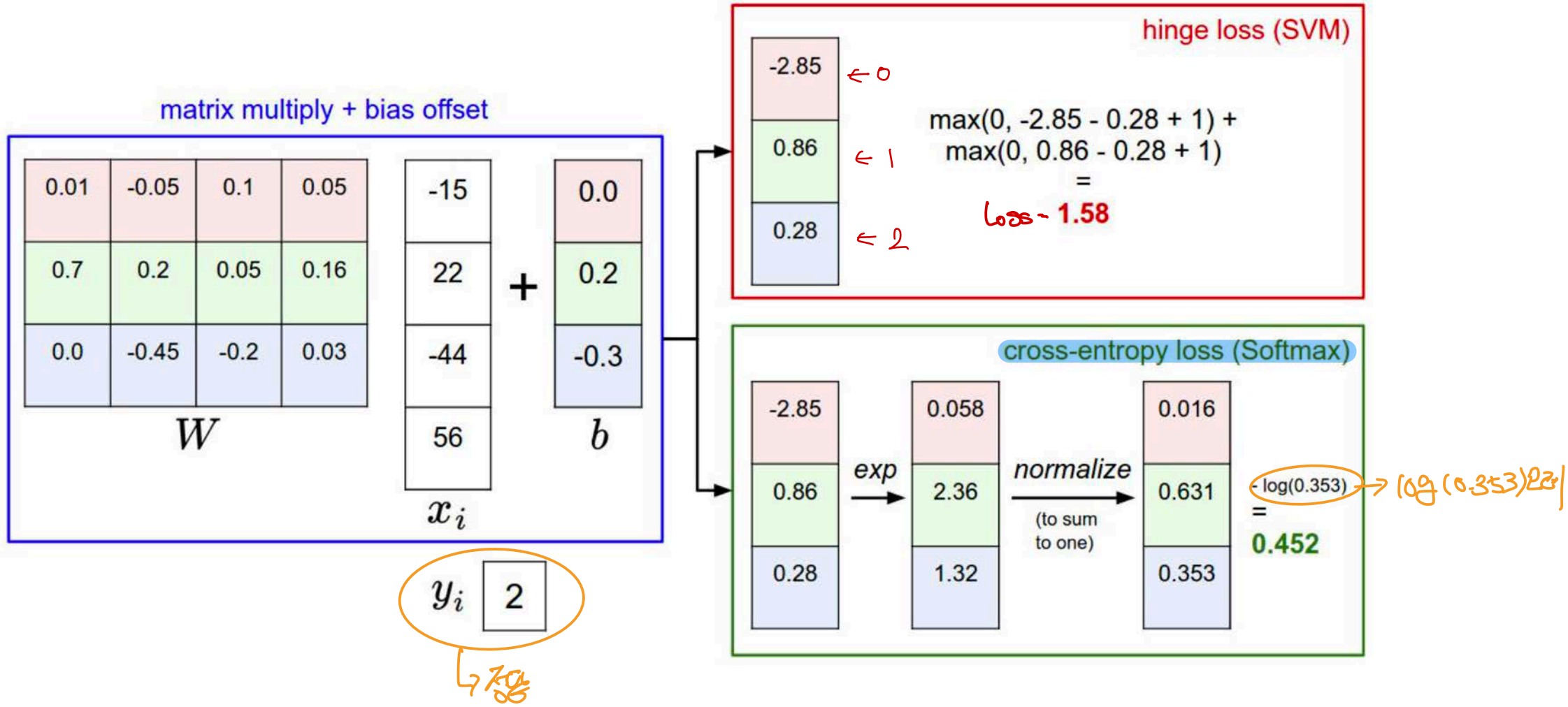
cat	**3.2**	exp	**24.5**	normalize	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

L_i = -log(0.13)
= **0.89**

SVM vs Softmax (Cross Entropy Loss)



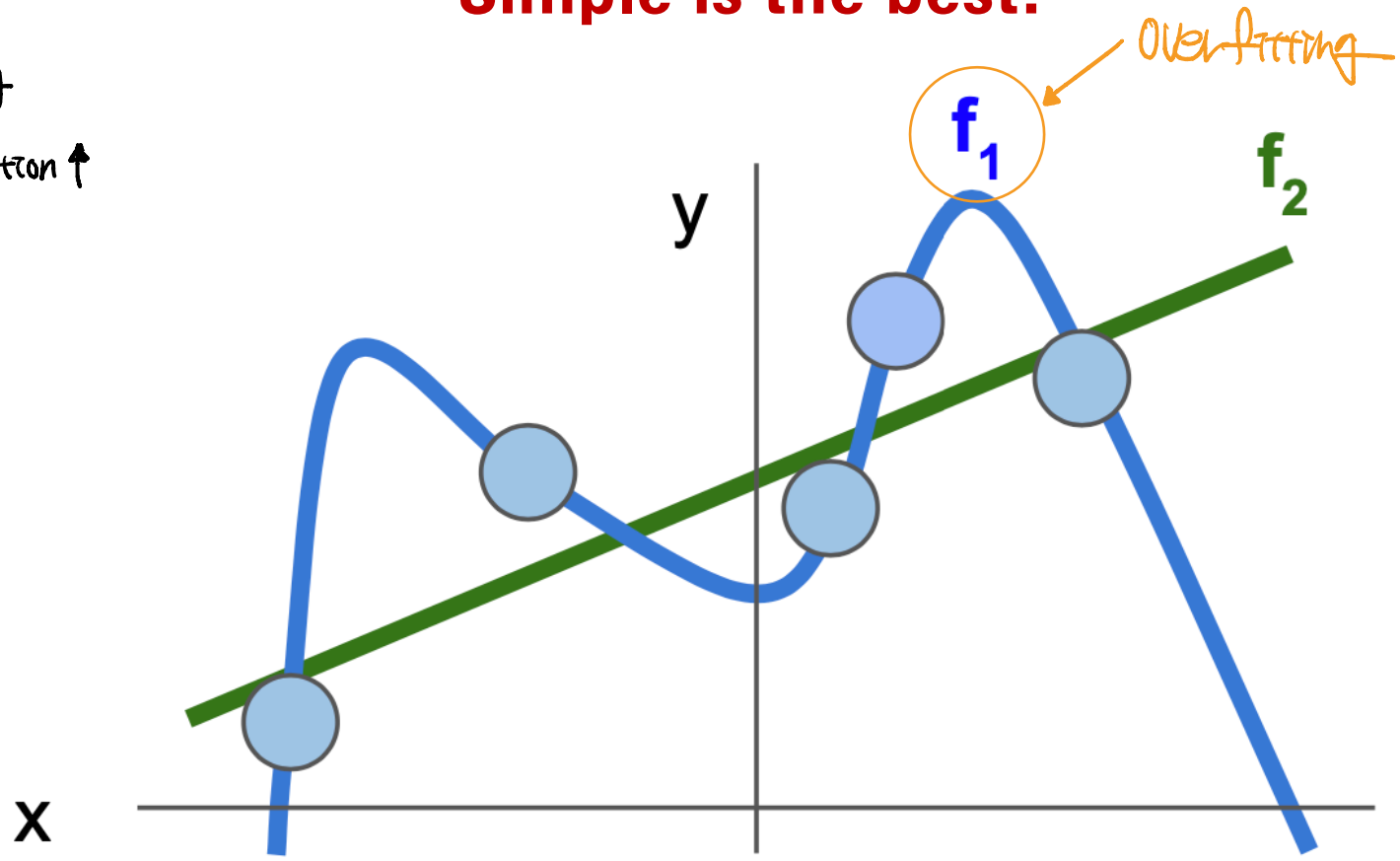
Regularization Intuition

: **Simpler** Models

If not overfitting

Loss↑ but regularization↑

"Simple is the best!"



Regularization avoids **overfitting** on the training data

Too much fitting on training data

Weight Regularization

: Regularization Strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i\left(f\left(x_i, W\right), y_i\right)}_{\text{Data loss}} \underbrace{- \lambda R(W)}_{\text{Regularization}}$$

Data loss :
Following data distribution.

Regularization :
Preventing **overfitting**.

In common use:

L2 regularization	$R(W) = \sum_k \sum_l W_{k,l}^2$
L1 regularization	$R(W) = \sum_k \sum_l \lvert W_{k,l} \rvert$
Elastic net (L1 + L2)	$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + \lvert W_{k,l} \rvert$

More complex :

Dropout

Batch normalization

Stochastic depth, etc

Optimization

Follow the Slope – Analytic Gradient

Analytic Gradient

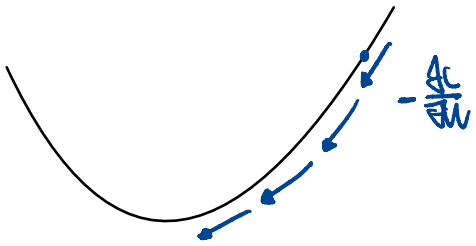
The loss is just a function of W :

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want : $\nabla_W L$



Use **calculus** to compute an analytic gradient

➔ **partial derivative**!

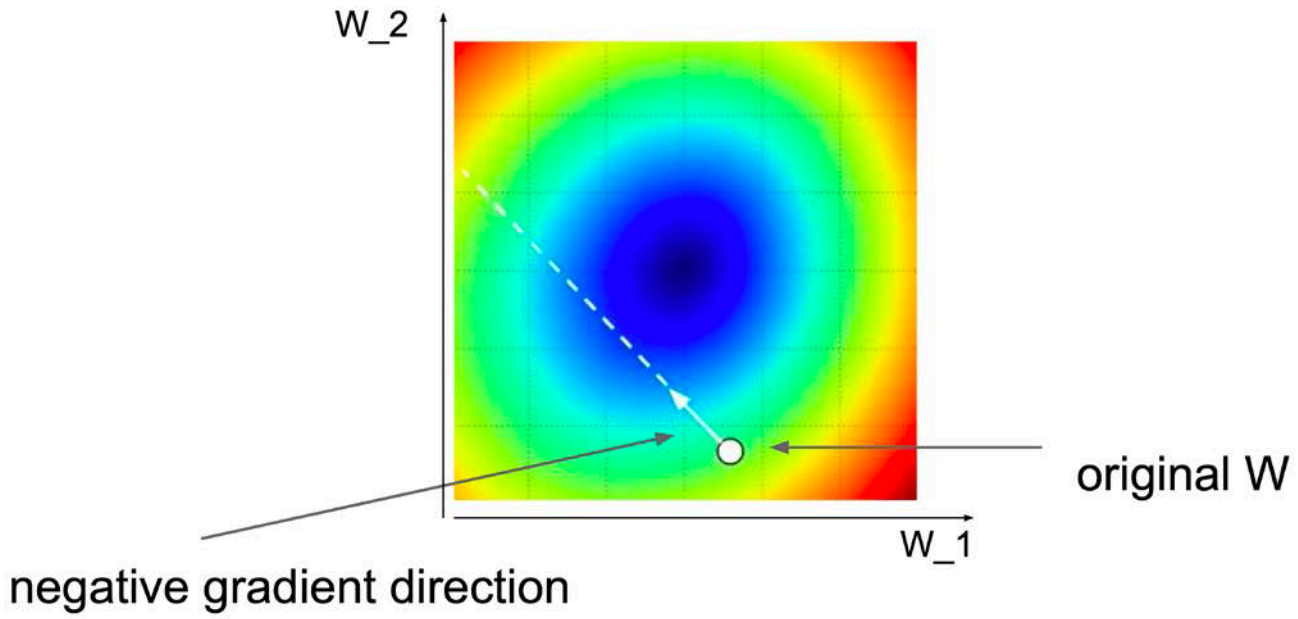
Gradient Descent

Full – batch

↳ $\frac{1}{N}\sum_{i=1}^{N}\frac{\partial L}{\partial W_i}$

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fn, data, weights)
    weights += - step_size * weights_grad # perform parameter
update
```



Mini-batch Gradient Descent

Randomly selected Batch Sample

$$\frac{1}{B}\sum_{i=1}^{B}\frac{\partial L}{\partial W_i}$$

Only use a **small portion** of the training set to compute the gradient

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256
examples
    weights_grad = evaluate_gradient(loss_fn, data_batch,
weights)
    weights += - step_size * weights_grad # perform parameter
update
```

Batch-size

$\frac{N}{B}$ 개의 Gradient descent 실행

→ 반복계산하면 Batch GD와 유사한 결과

→ 빠른 속도

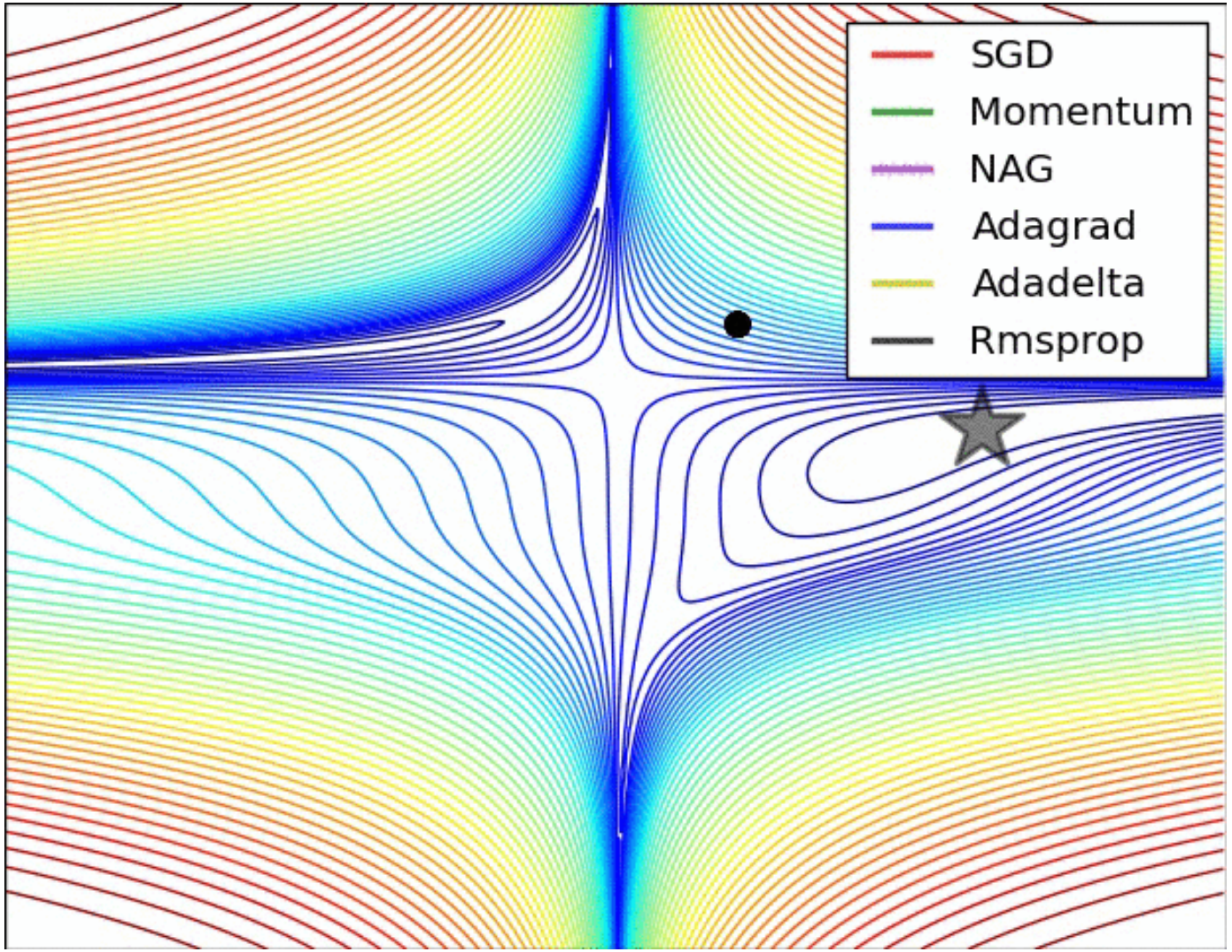
참고

Various Optimization Methods

- Momentum
- Adagrad
- RMSProp
- Adam
- ⋮

Common mini-batch sizes are 32/64/128/... examples

Gradient Descent



Stochastic Gradient Descent (SGD)

Mini-Batch Size 에따른 Sampling Error → 학습력

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

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i\left(x_i, y_i, W\right) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i\left(x_i, y_i, W\right) + \lambda \nabla_W R(W)$$

Full sum expensive when N is large!

→ Reason why use small number of batch

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights)grad # perform parameter update
```

Learning Rate Schedules

$W \leftarrow W - \alpha \frac{\partial L}{\partial W}$

Learning rate (step size)

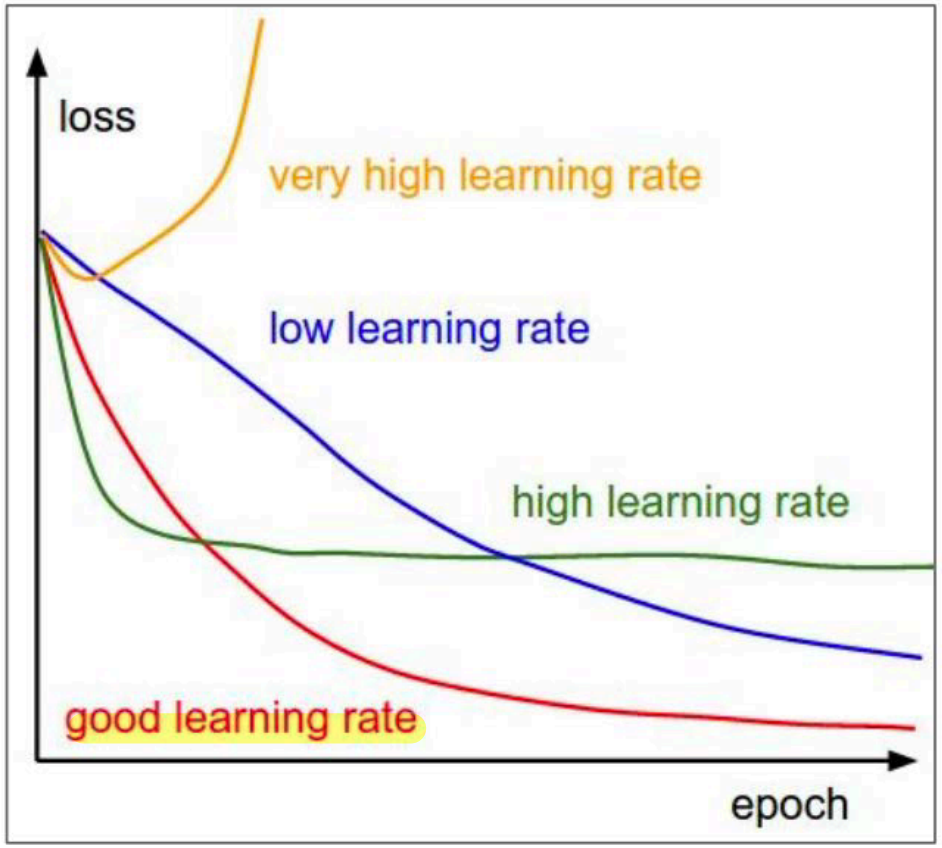
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Learning rate

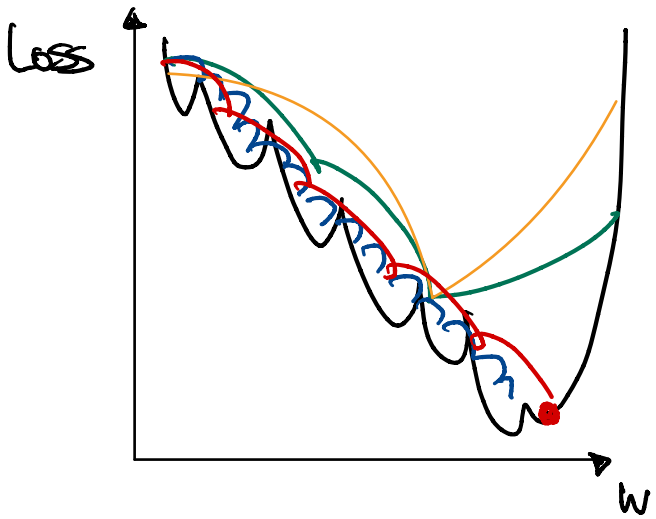
Learning Rate Schedules

Learning rate as a hyperparameter.

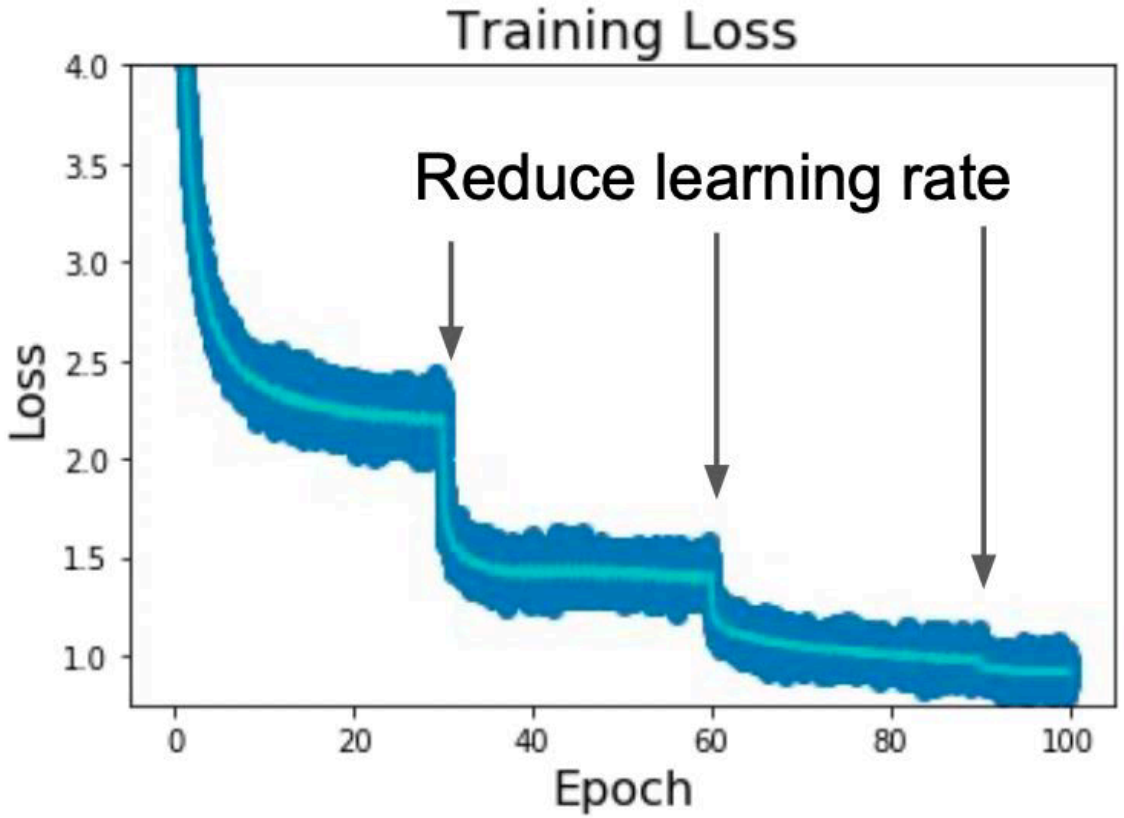


Q. Which one of these learning rates is best to use?

→ 큰 Learning rate에서 작은 Learning rate로 줄여가며 최적의 Learning rate를 찾는다.



Learning Rate Decay



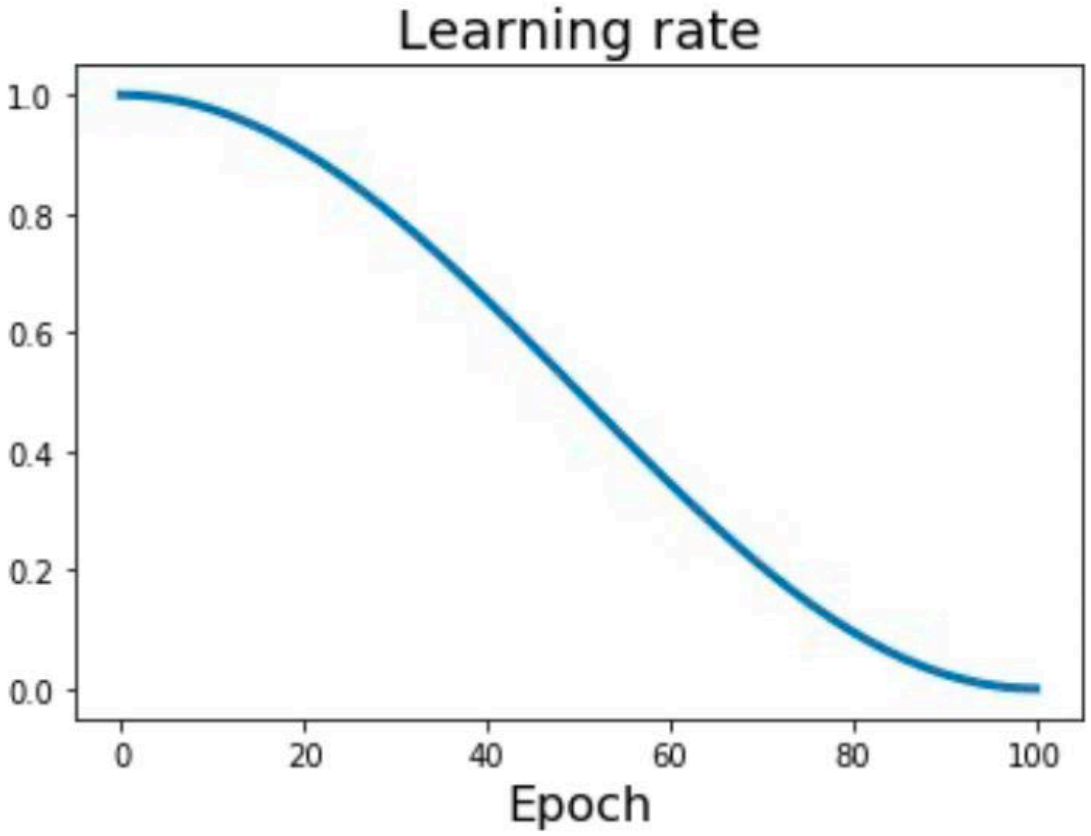
Step: Reduce LR at a few fixed points
(E.g. multiply LR by 0.1 after epochs 30, 60, 90)

Epoch 마다 LR을 줄임

학습초반: W가 멀다 → LR↑

학습후반: W가 가까 → LR↓

Learning Rate Decay



Step: Reduce LR at a few fixed points
(E.g. multiply LR by 0.1 after epochs 30, 60, 90)

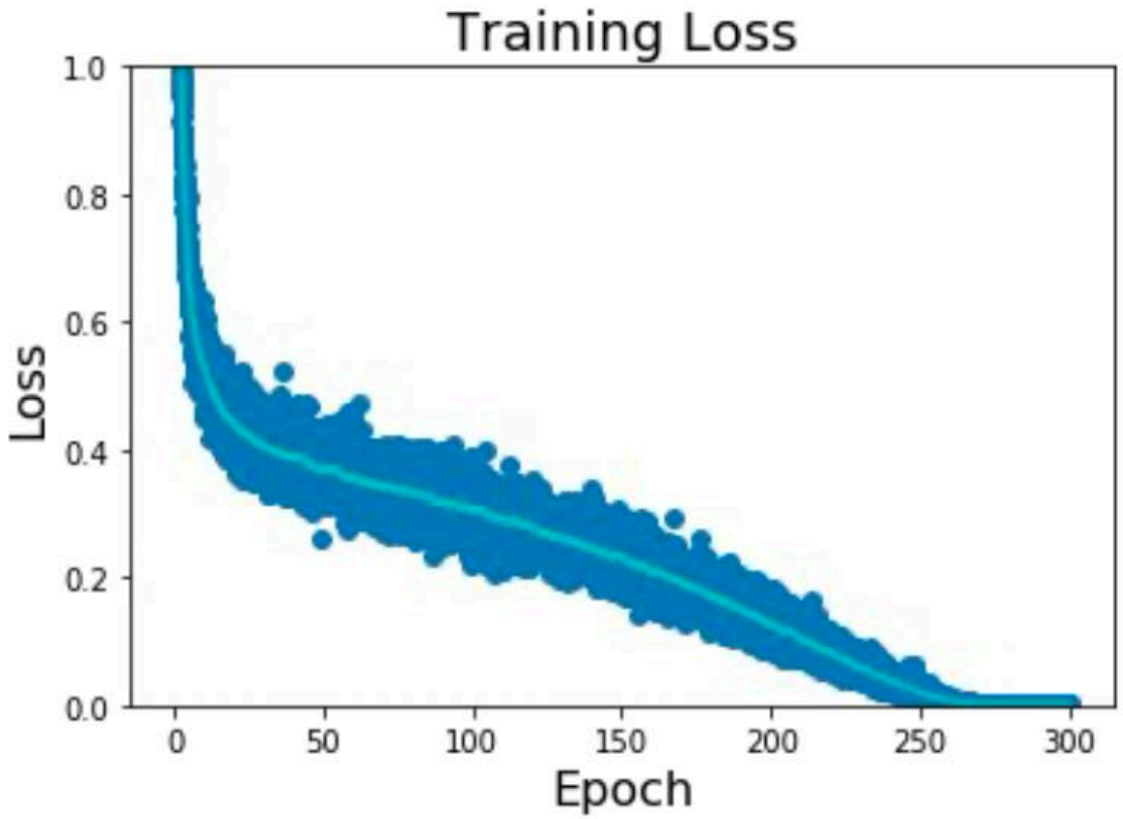
Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Learning Rate Decay



Step: Reduce LR at a few fixed points
(E.g. multiply LR by 0.1 after epochs 30, 60, 90)

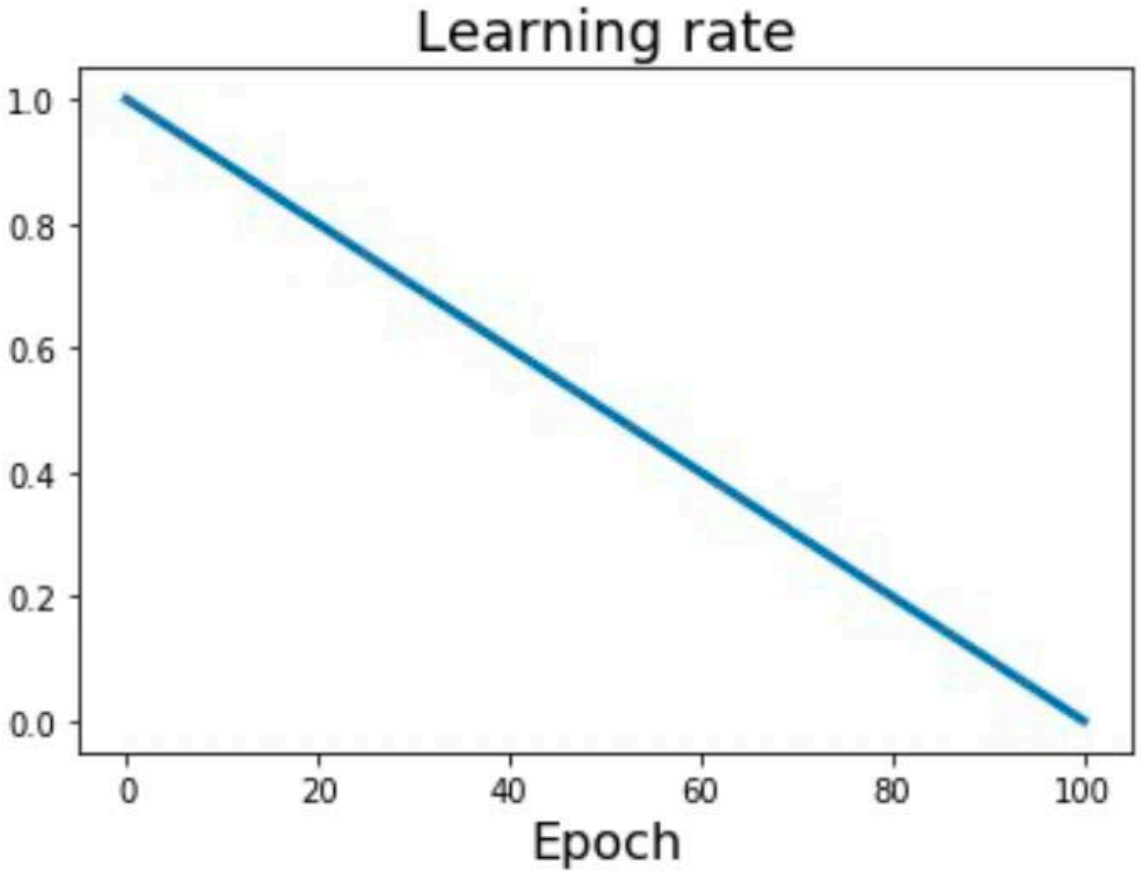
Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Learning Rate Decay



Step: Reduce LR at a few fixed points
(E.g. multiply LR by 0.1 after epochs 30, 60, 90)

Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$

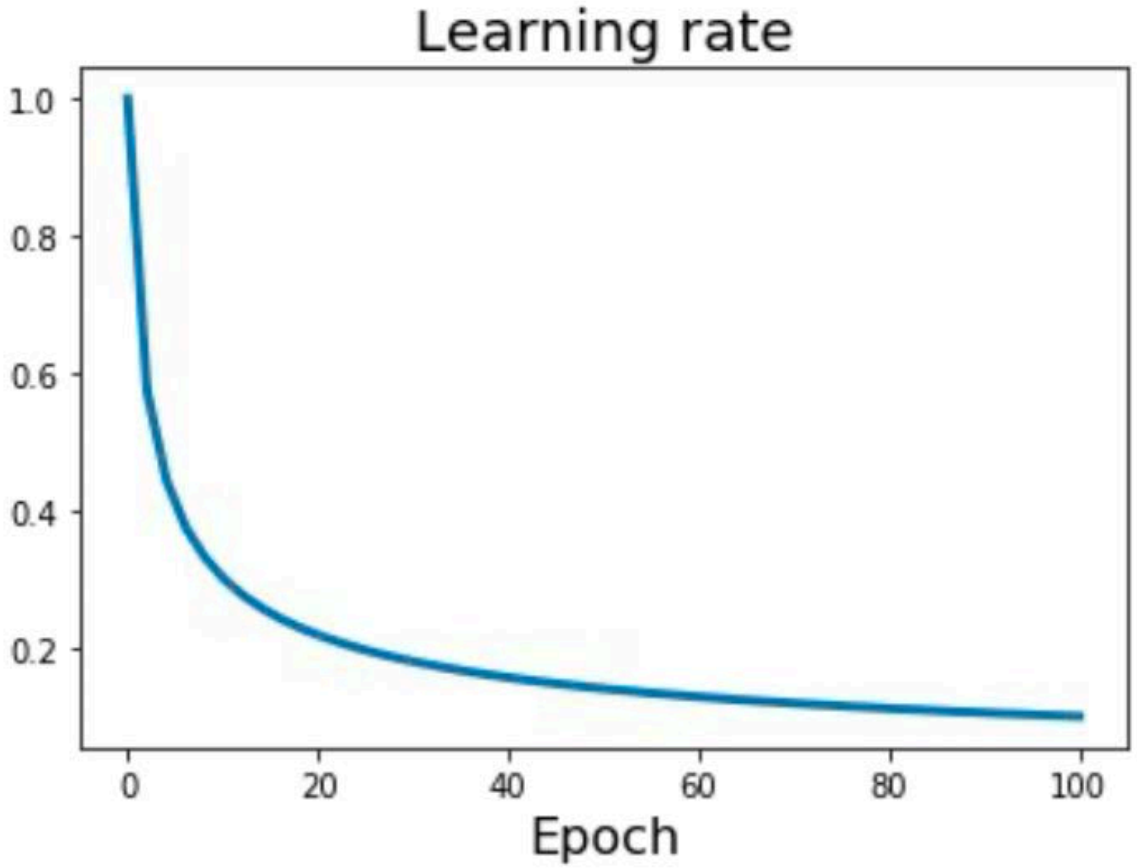
Linear: $\alpha_t = \alpha_0(1 - t/T)$

α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Learning Rate Decay



Epoch 마다 learning rate 조정

Step: Reduce LR at a few fixed points
(E.g. multiply LR by 0.1 after epochs 30, 60, 90)

Cosine: $\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(t\pi/T))$

Linear: $\alpha_t = \alpha_0(1 - t/T)$

Inverse sqrt : $\alpha_t = \alpha_0/\sqrt{t}$

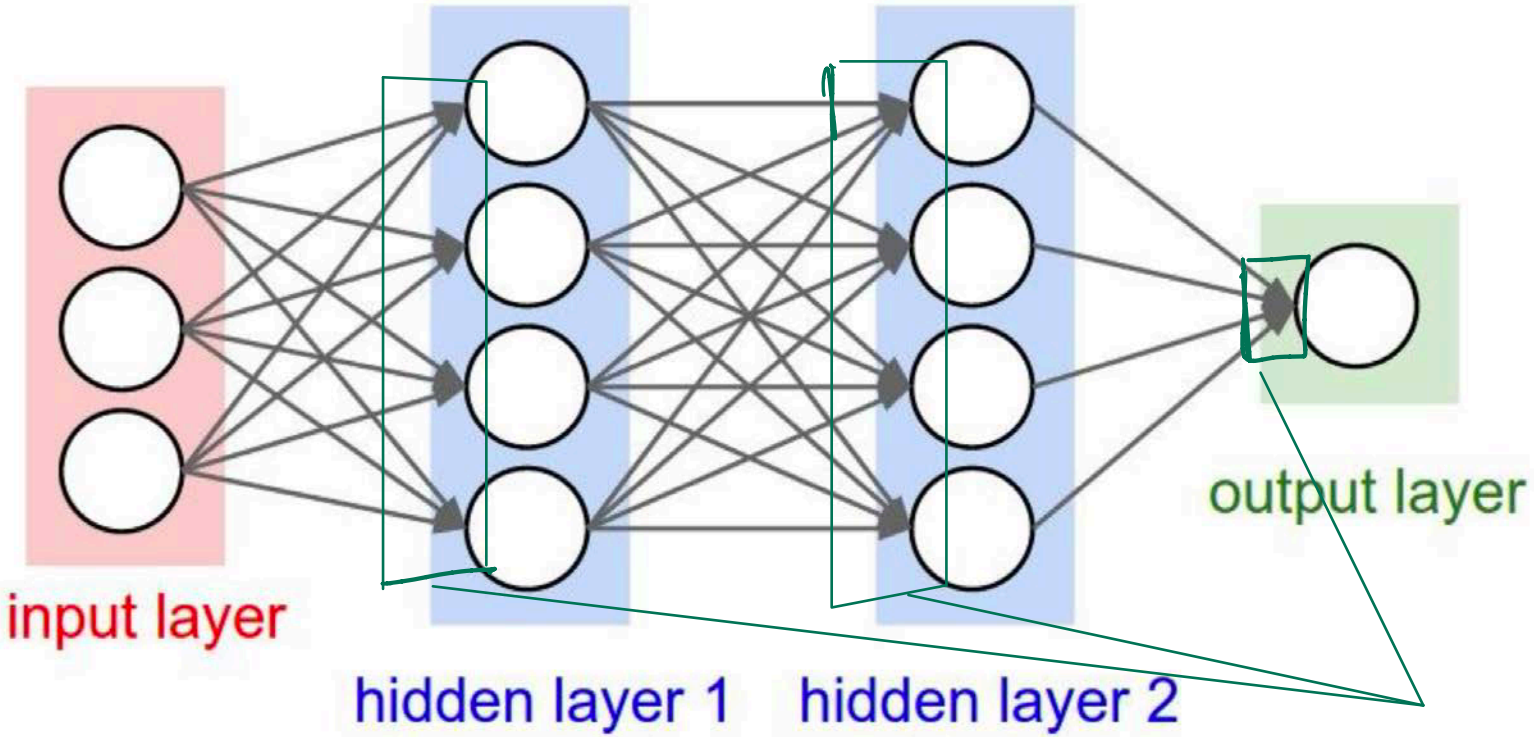
α_0 : Initial learning rate

α_t : Learning rate at epoch t

T : Total number of epochs

Neural Networks

Neural Networks: Architectures



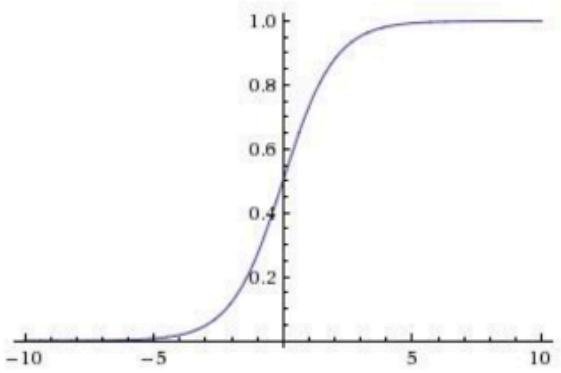
"3-layer Neural Net" or "2-hidden-layer Neural Net"

This Neural Networks is "Fully-connected" layers

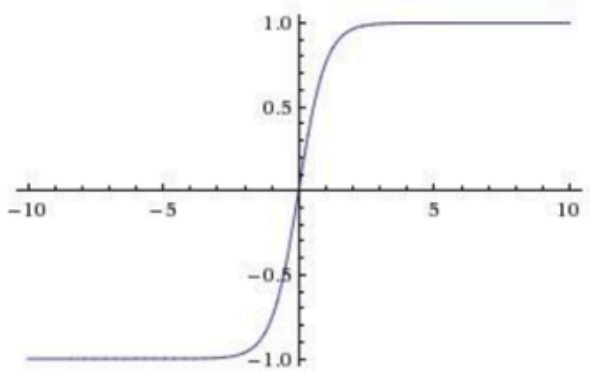
모든 Input Neuron과 Output Neuron이 연결

Activation Functions : Non-linear function

Sigmoid $\sigma(x) = 1/(1+e^{-x})$



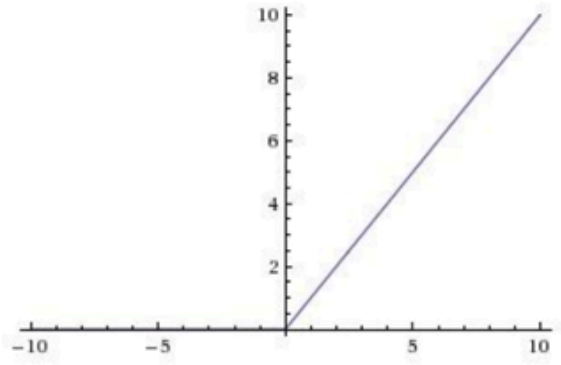
tanh $\tanh(x)$



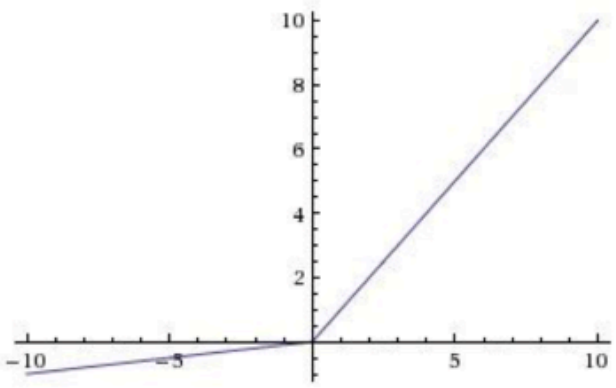
Maxout

$$\max\left(w_1^T x + b_1, w_2^T x + b_2\right)$$

ReLU $\max(0, x)$



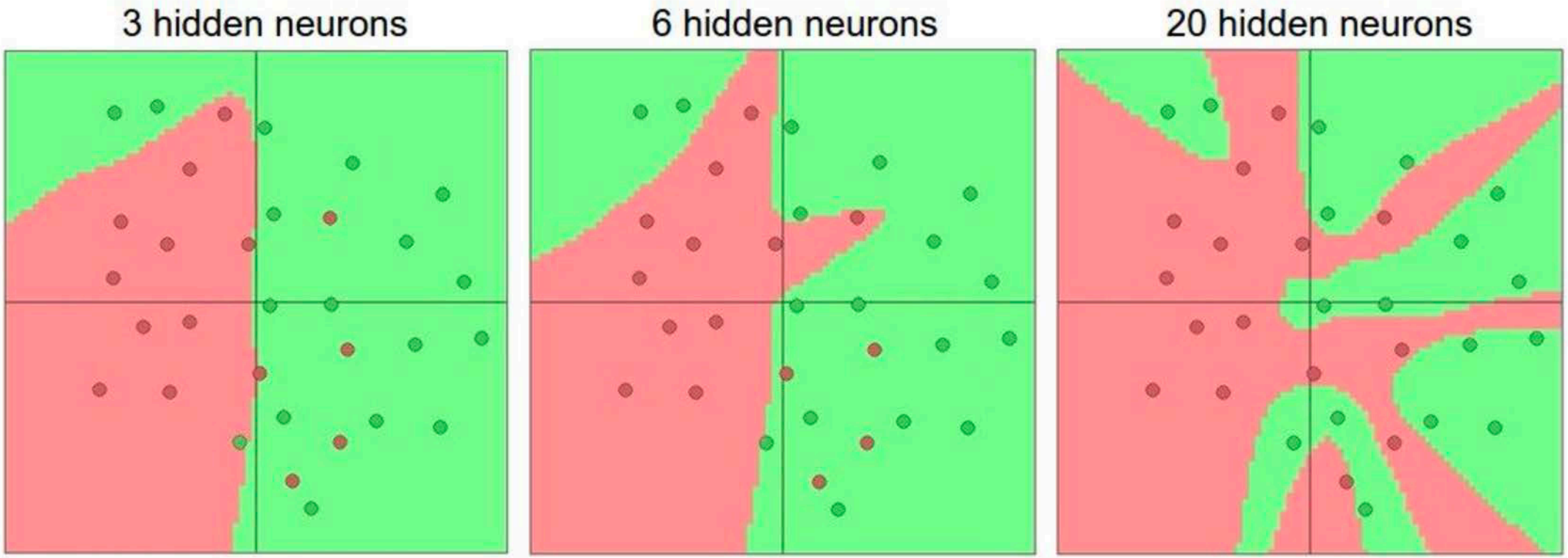
LeakyReLU $\max(0.1x, x)$



ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

Setting the Number of Layers and Sizes



Layer ↑ ⟶ 비선형성 증가, more complex decision boundary

⟶ Deep learning의 핵심 아이

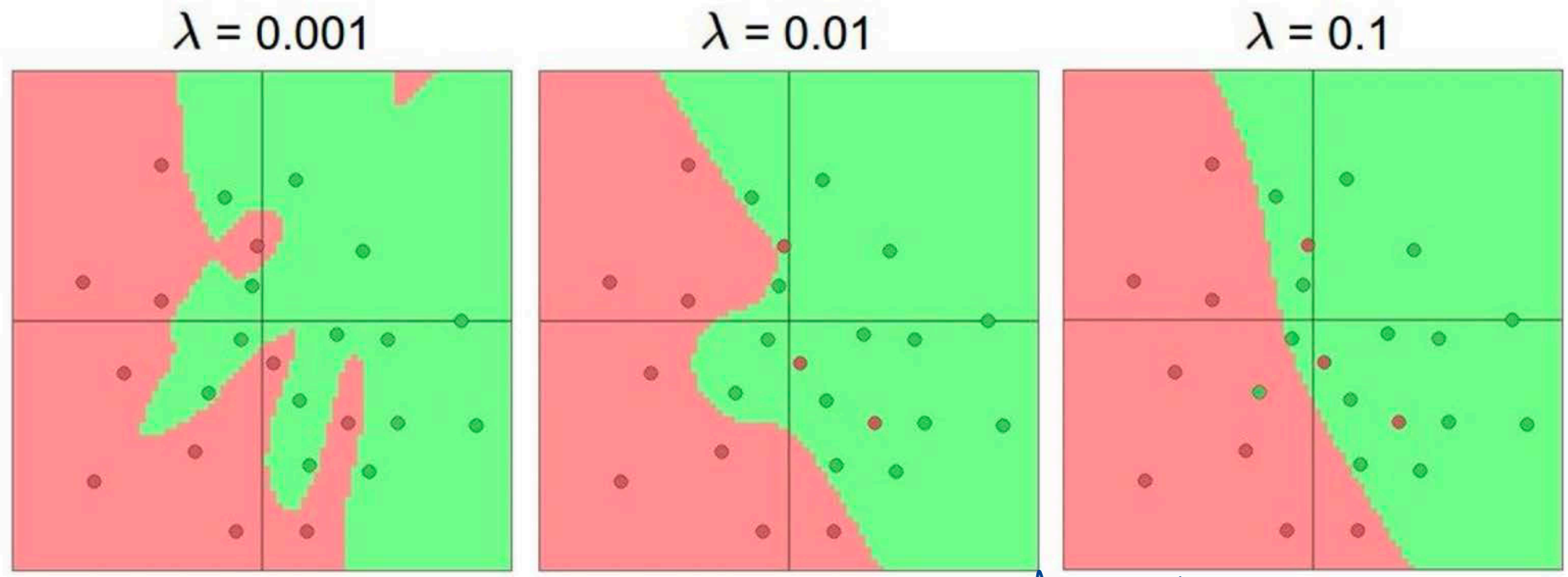
more neurons = more capacity of data

Setting the Number of Layers and Sizes

* More layers & neurons but, add some regularization

$\frac{1}{N}\sum L + \lambda R(W)$

Do not use size of neural network as a regularizer. Use **stronger regularization** instead :

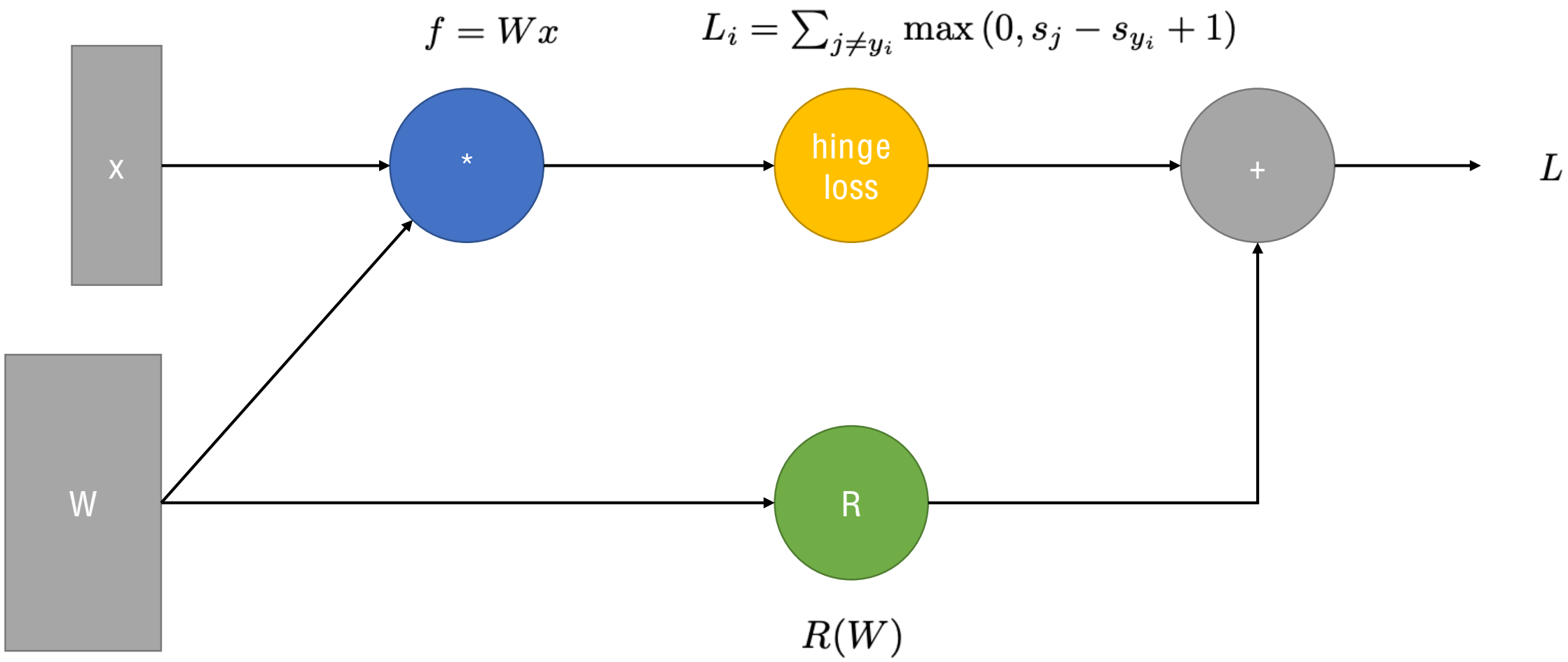


$\lambda \uparrow \rightarrow$ overfitting ↓

*Overfitting 된다면 model size 줄이지말고, Regularization을 강하게!

Backpropagation

Computational Graph



Backpropagation

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial q} \cdot \frac{\partial q}{\partial x}$$

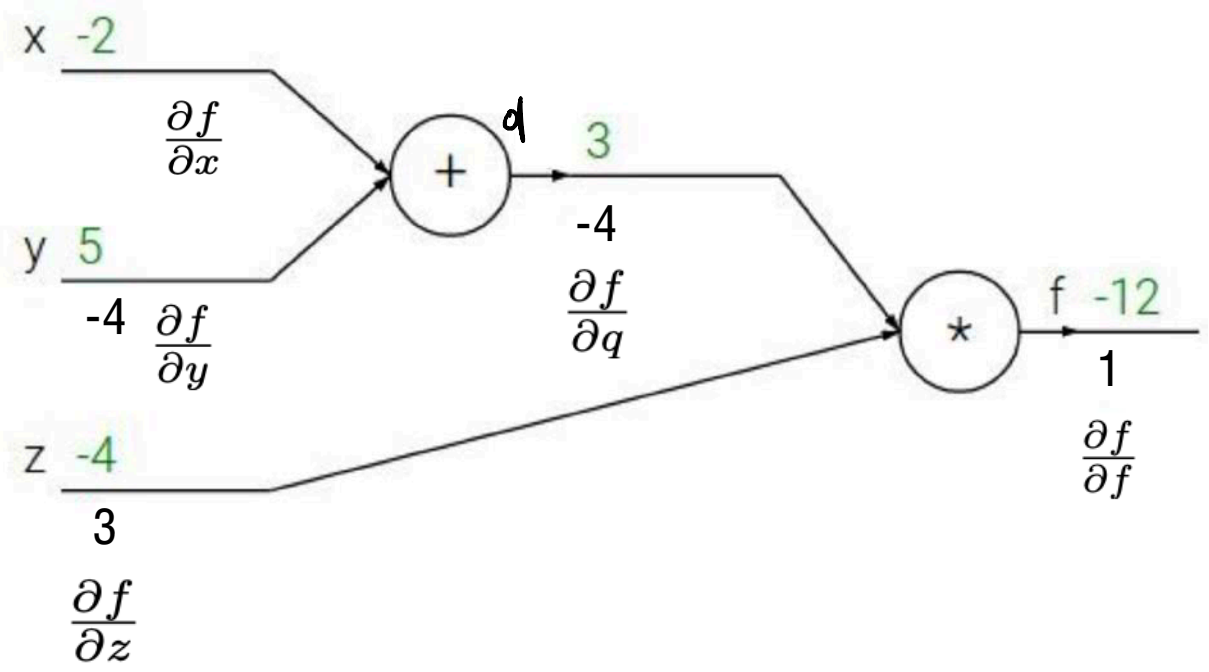
$f(x, y, z) = (x + y)z$

e.g. $x = -2, y = 5, z = -4$

$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

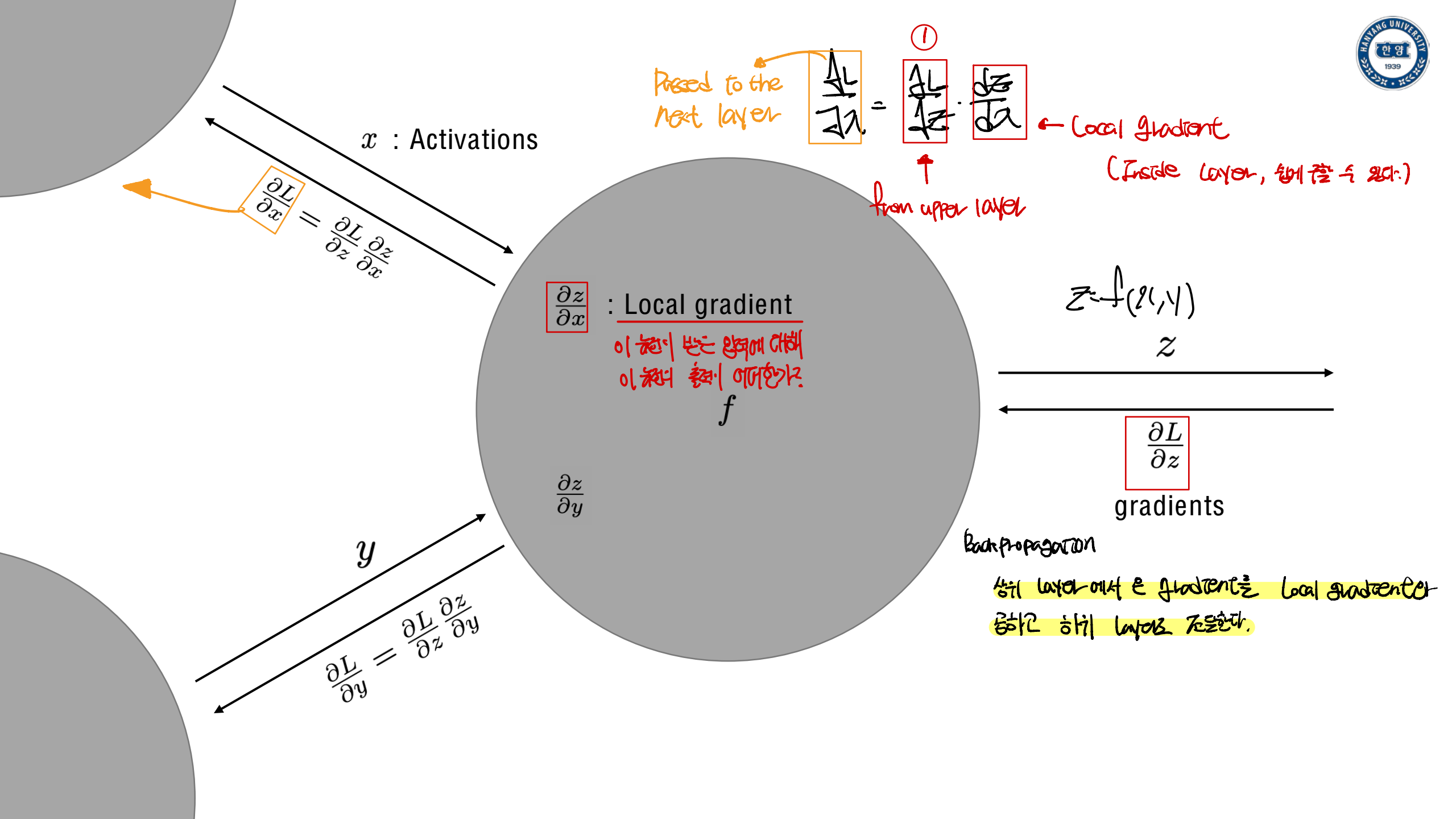
$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

Want : $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain Rule : $\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$

$-2 \cdot 1 = -2$



이 layer의 input에 대해 이 layer 출력이 얼마인가.

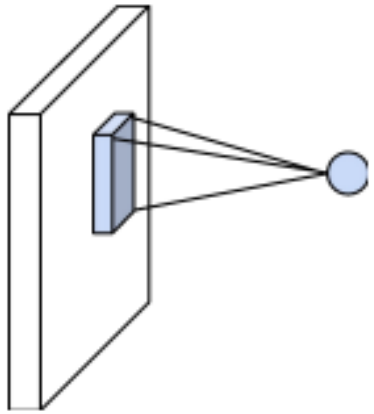
Backpropagation

상위 layer에서 온 gradient를 local gradient와 곱하고 하위 layer로 전달한다.

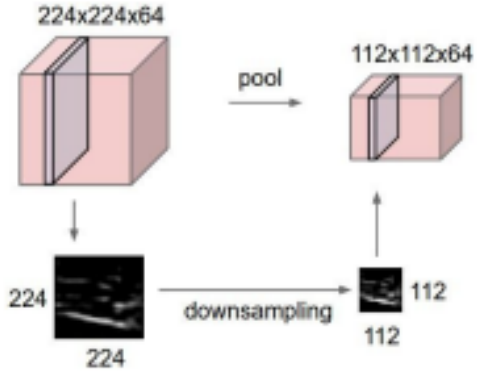
Convolutional Neural Networks

Components of **CNNs**

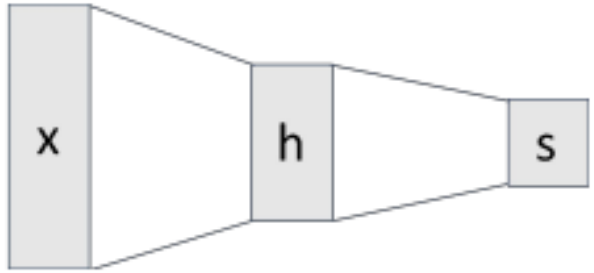
Convolution Layers



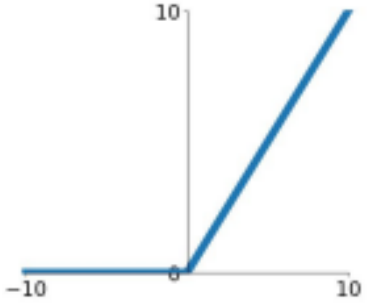
Pooling Layers



Fully-Connected Layers



Activation Function



Normalization

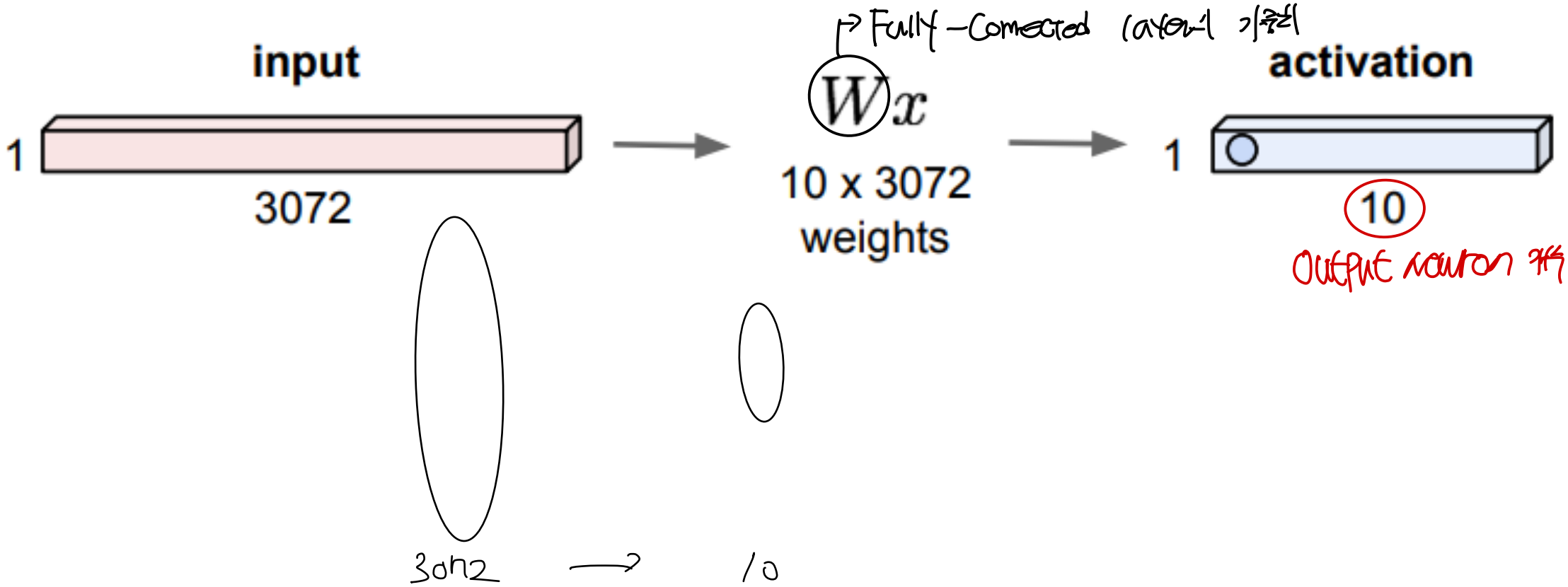
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Question: How should we put them together?

Recap: Fully Connected Layer

Fully Connected 1-dim vector를 Input으로 받아 10개의 출력 Neuron으로

32x32x3 image → stretch to 3072x1

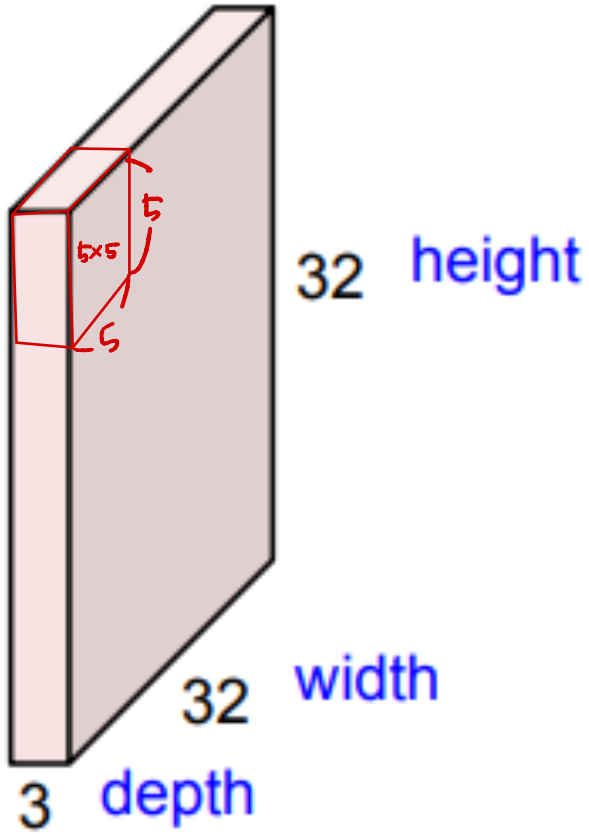


Convolution Layer

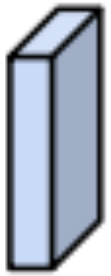
Filter always extend the full depth of the input volume

Filter의 Depth는 Image와 동일해야함

32x32x3 image



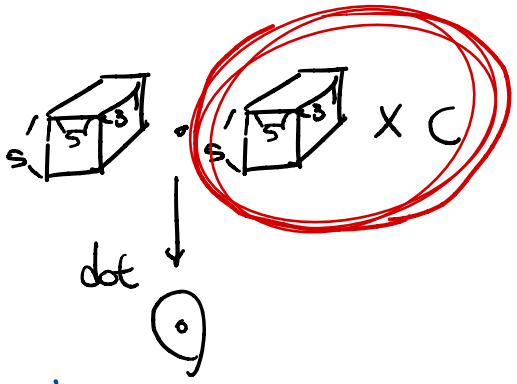
5x5x3 filter

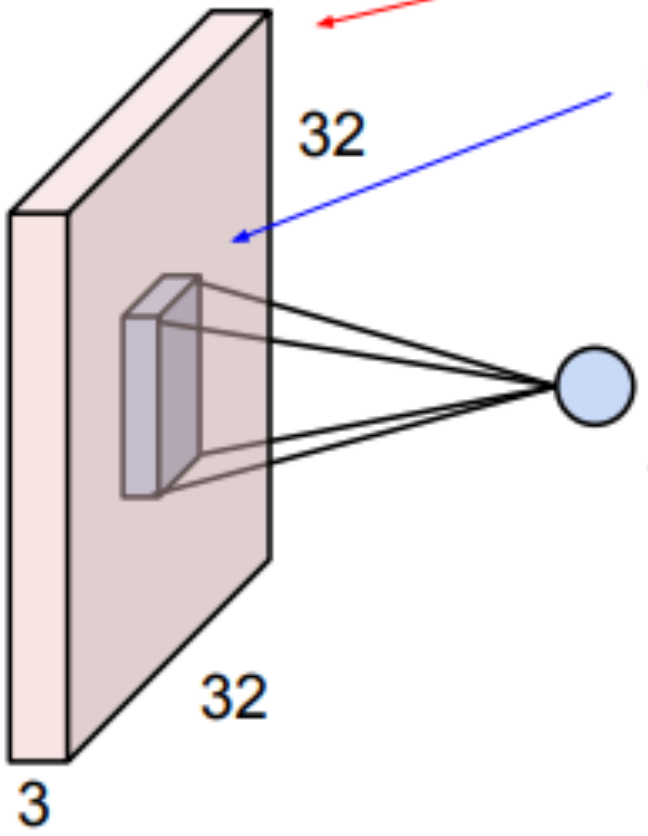


→ Scalar value

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

Convolution Layer





32x32x3 image

5x5x3 filter $w \in R^{75 \times 1}$

5x5x3 = 75

1 number:

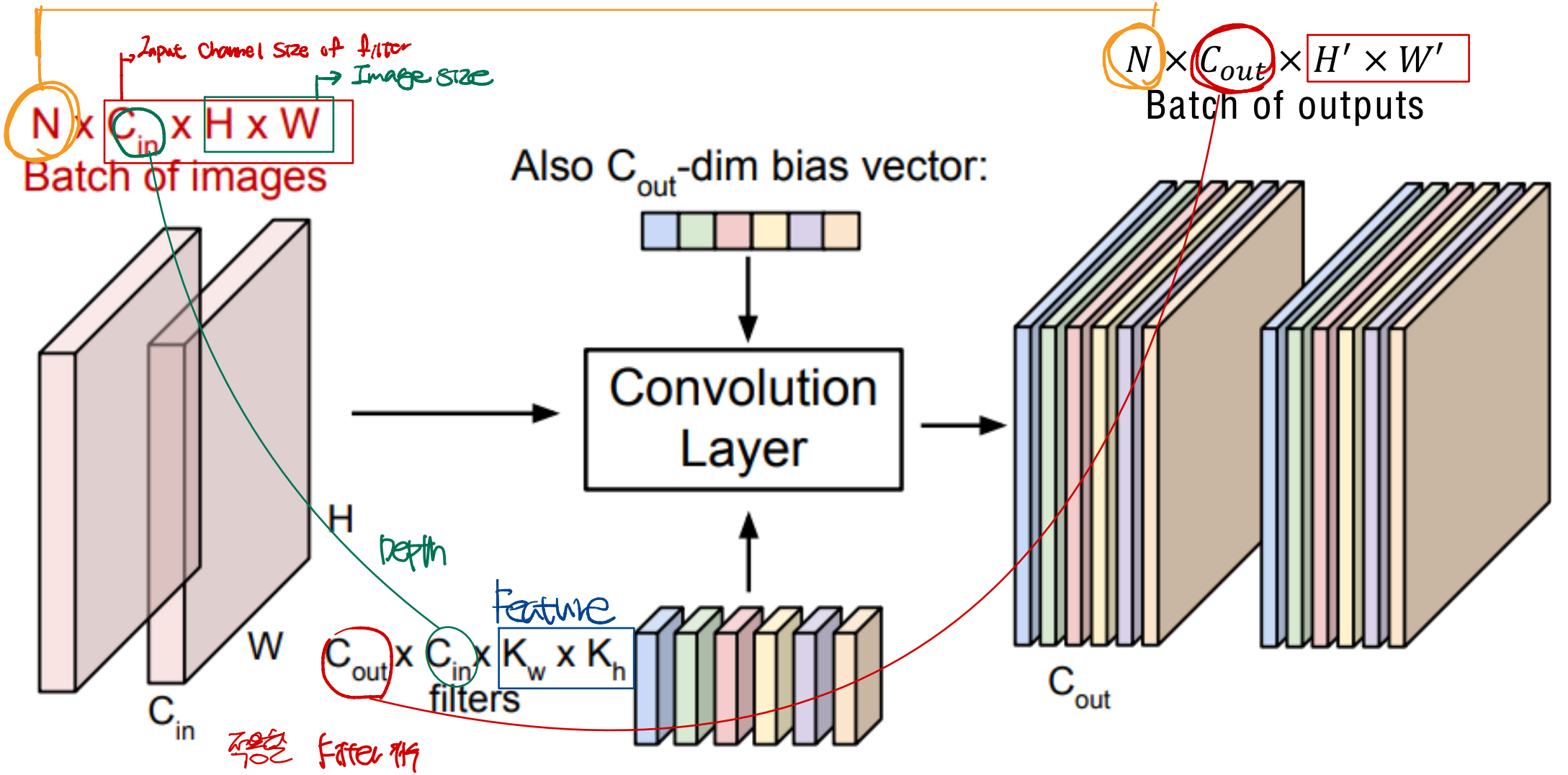
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

← 75-dimensional 내적 = 한 개의 숫자

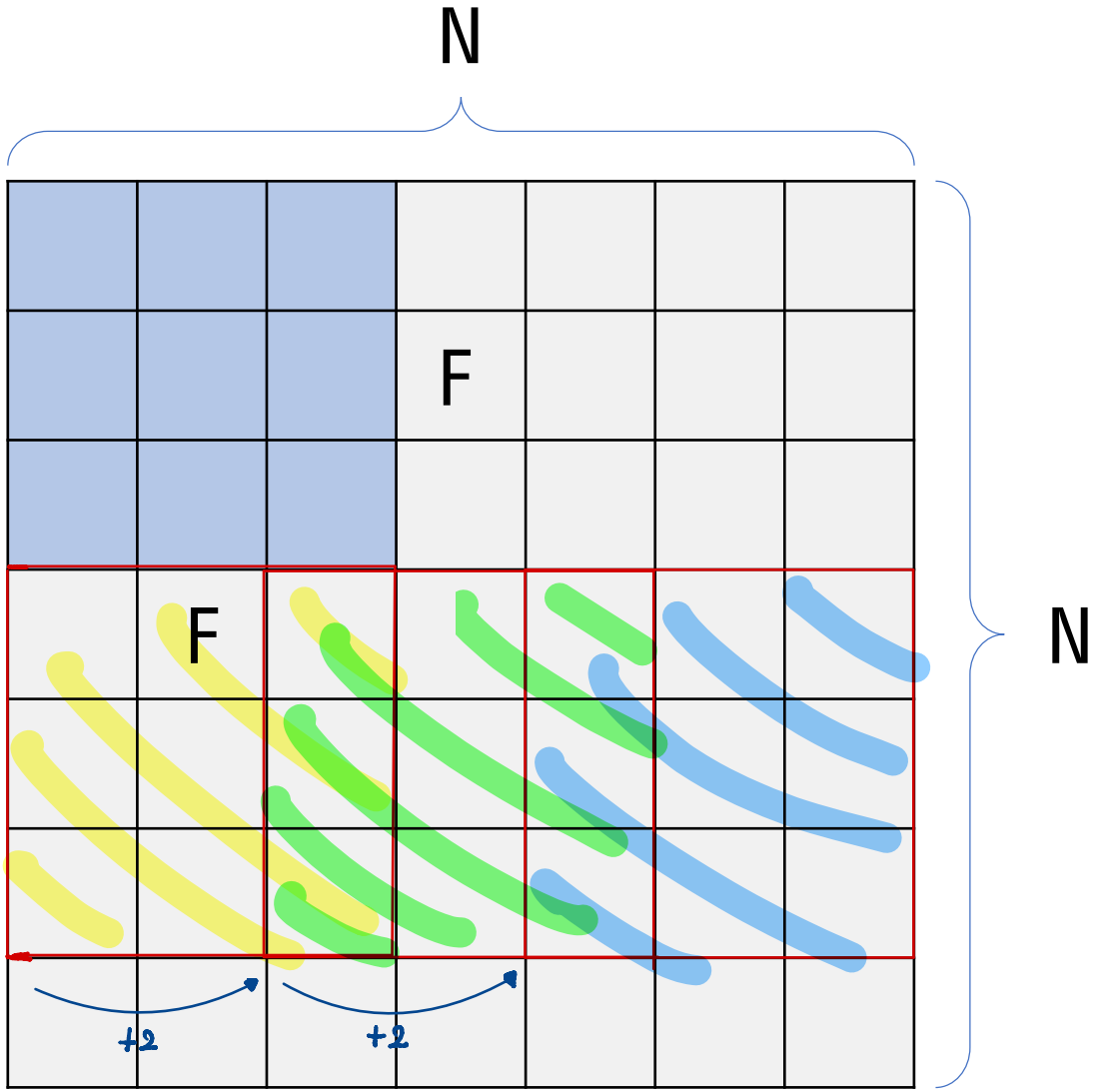
Convolution Layer

N개의 (H×W×Cin) 이미지는 Cout개의 (Kw, Kh, Cin) filter와 Convolution 하면 N × Cout × H′×W′



A closer look at spatial dimensions

HxWxD ← kernel ⇒ H'xW'
(x,y,D)

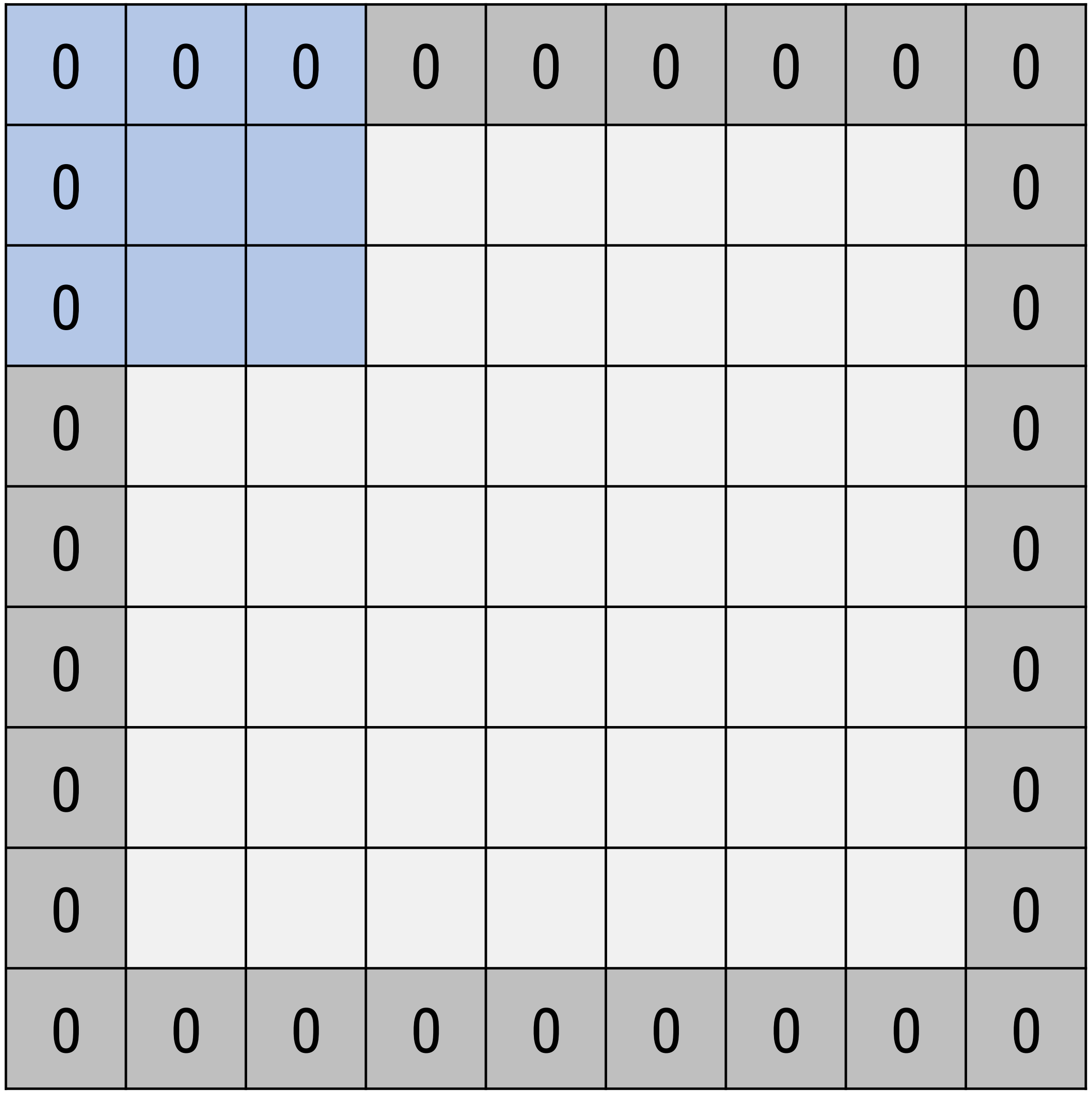


Output size
: (N - F) / stride + 1

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

정수로 Convolution 불가
해결 뒷 장 확인

In practice: Common to zero pad the border



Output size = $\frac{N+2P-F}{S}+1$

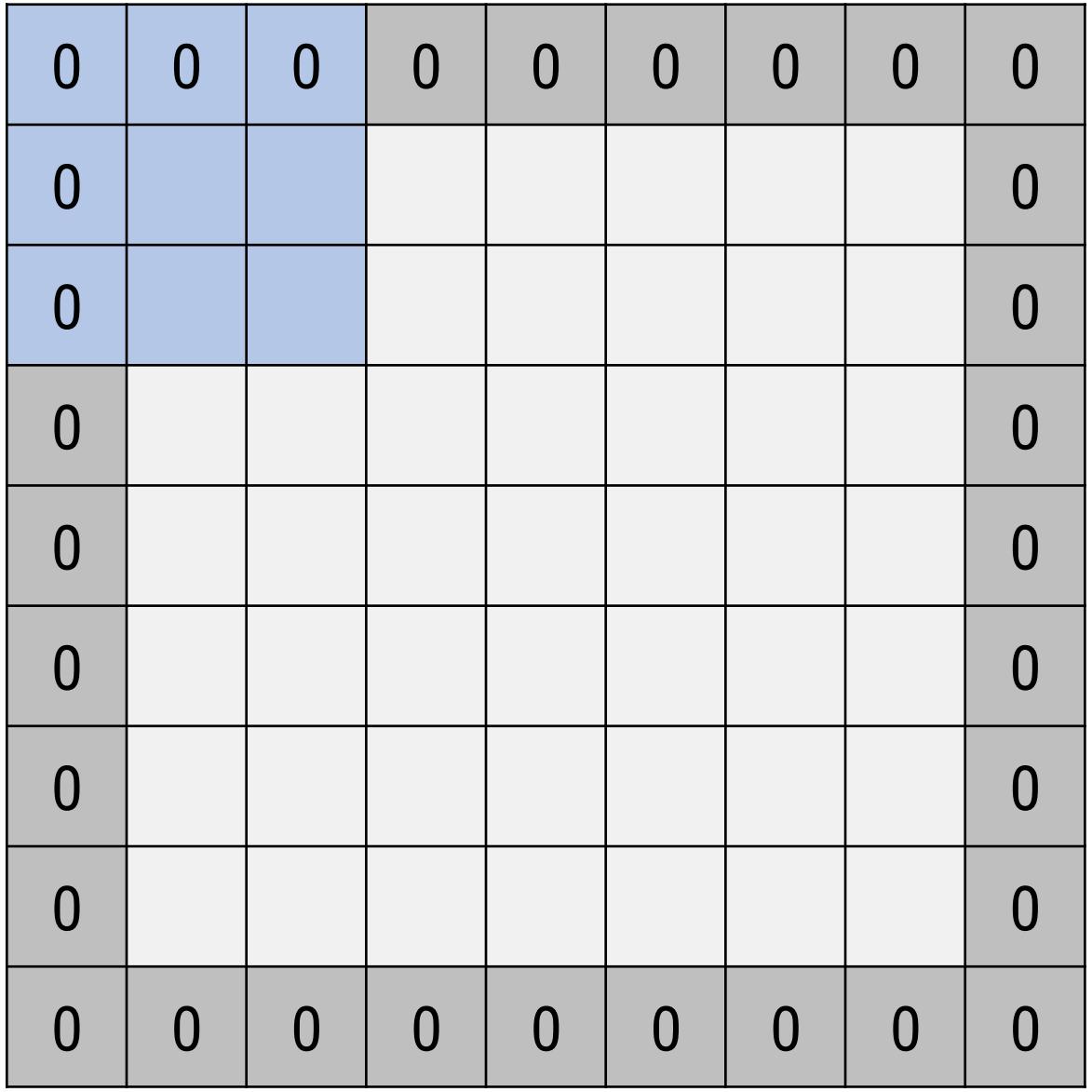
Output size $= \frac{N+2-F}{S}+1 = \frac{7+2-3}{1}+1 = 7$

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border

Q. what is the output?

(recall)
(N - F) / stride + 1

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border

Q. what is the output?
A. 7x7 output

Receptive Fields · output의 특정 Pixel을 결정하는 Input

For convolution with kernel size K, each element in the output depends on a K x K **receptive field** in the input

Convolution의 결과로 생기는 output은 Input의 어느 영역에 의존하는가?

Input Output

kernel size=K

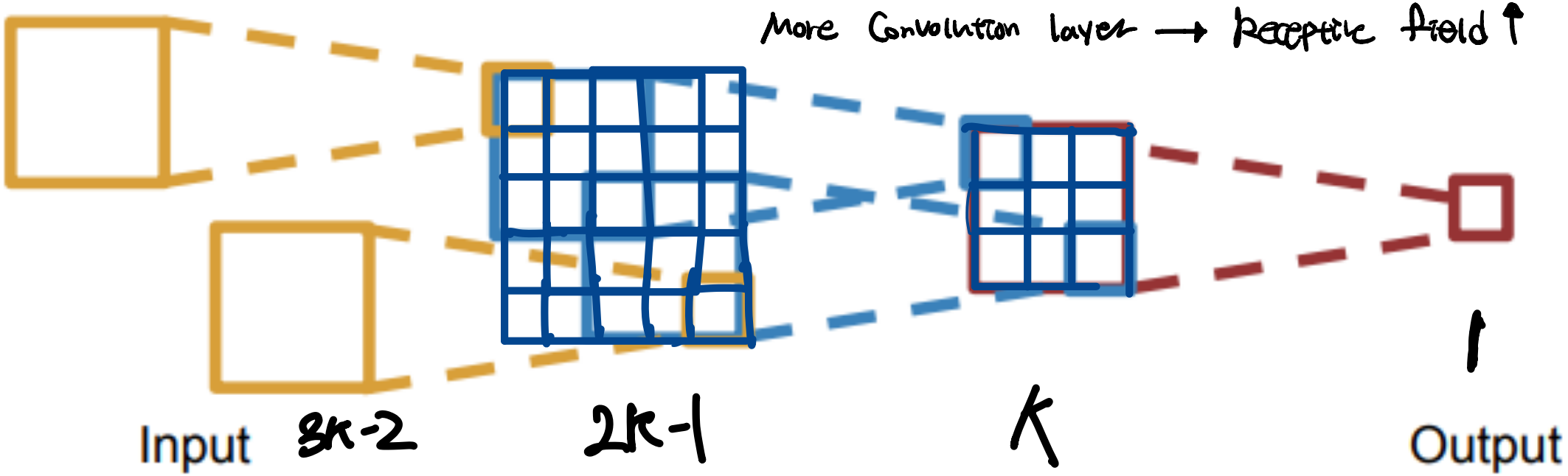
Receptive Fields

□ → (K×K) → K×K-1 →

Receptive field는 layer별로 다름.

Each successive convolution adds K – 1 to the receptive field size
With L layers the receptive field size is 1 + L * (K – 1)

More Convolution layer → Receptive field ↑

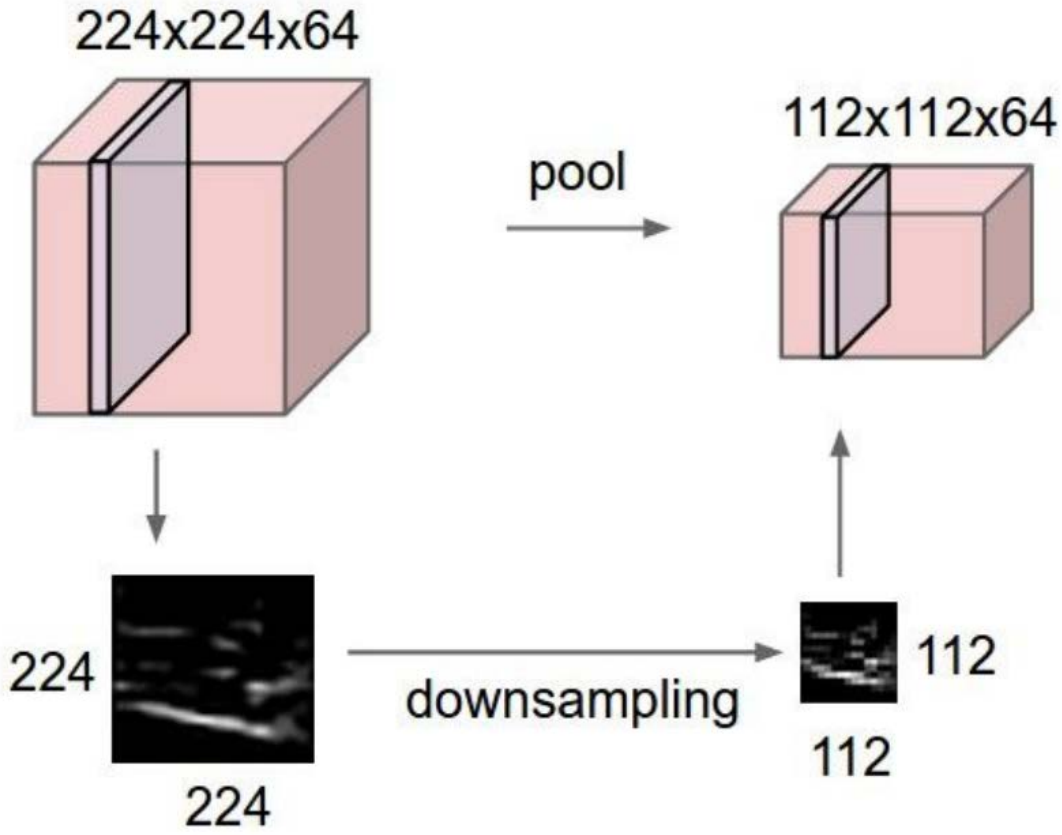


Be careful – "receptive field in the input" vs. "receptive field in the previous layer"

2냥 다른 언례!

Pooling Layer

Shrinks the representations **smaller.**

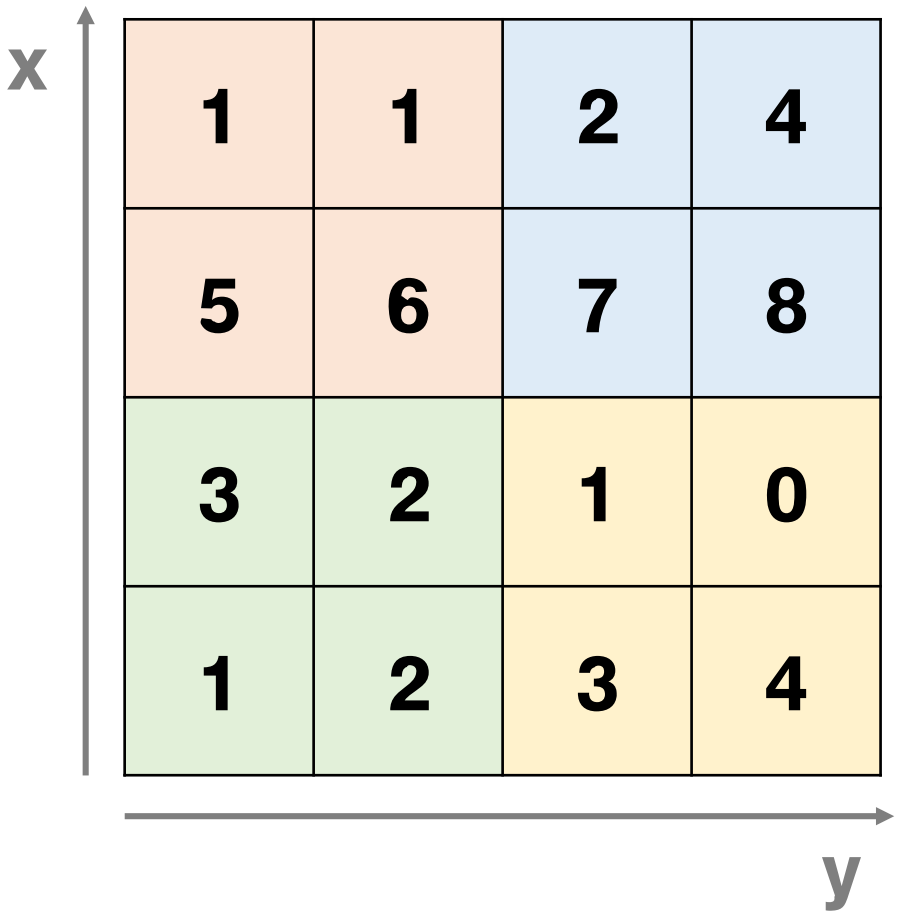


Max Pooling : Choose largest value in each filter

Activation ↑ => more informative

Max ← 특징이 가장 잘됨

Single depth slice



max pool with 2x2 filters and stride 2

6	8
3	4

$\frac{N-F}{S} + 1 = \frac{N}{2}$

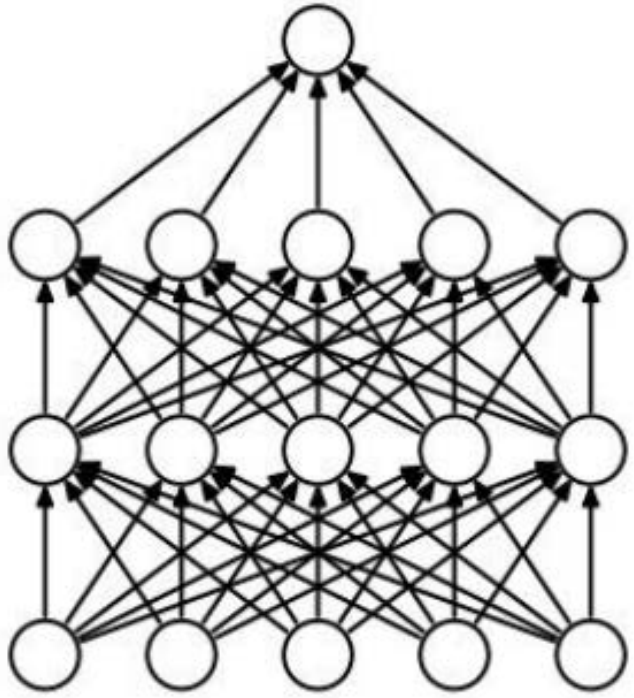
Stride = Filter size

→ 원래의 $\frac{1}{2}$배

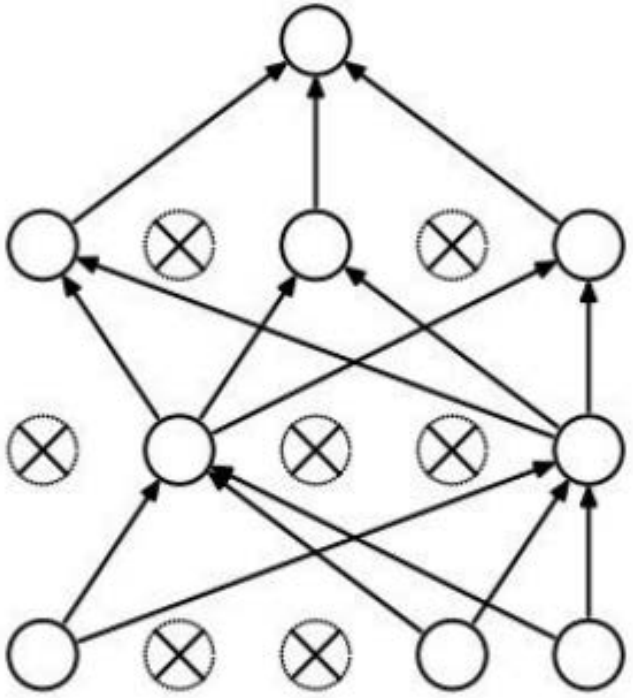
Regularization

Regularization: Dropout

- Randomly set some **neurons to zero** (commonly use 0.5 probability)
- Can be seen as an **ensembling** of sub-models.



(a) Standard Neural Net

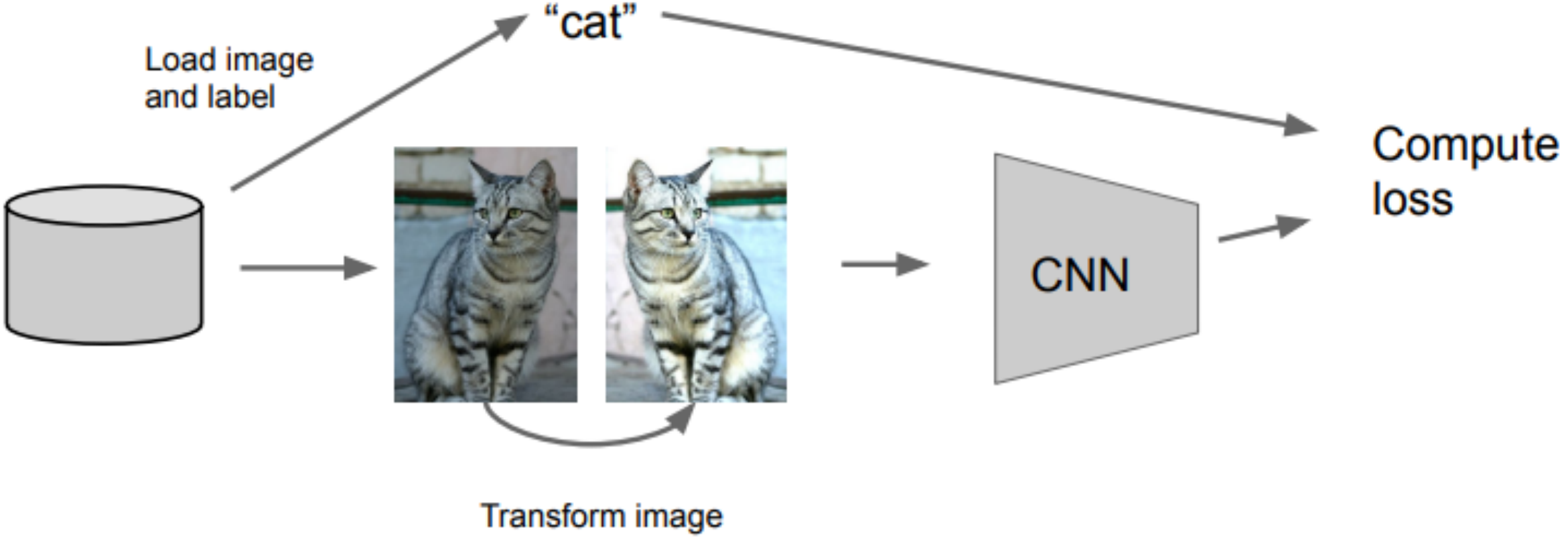


(b) After applying dropout.

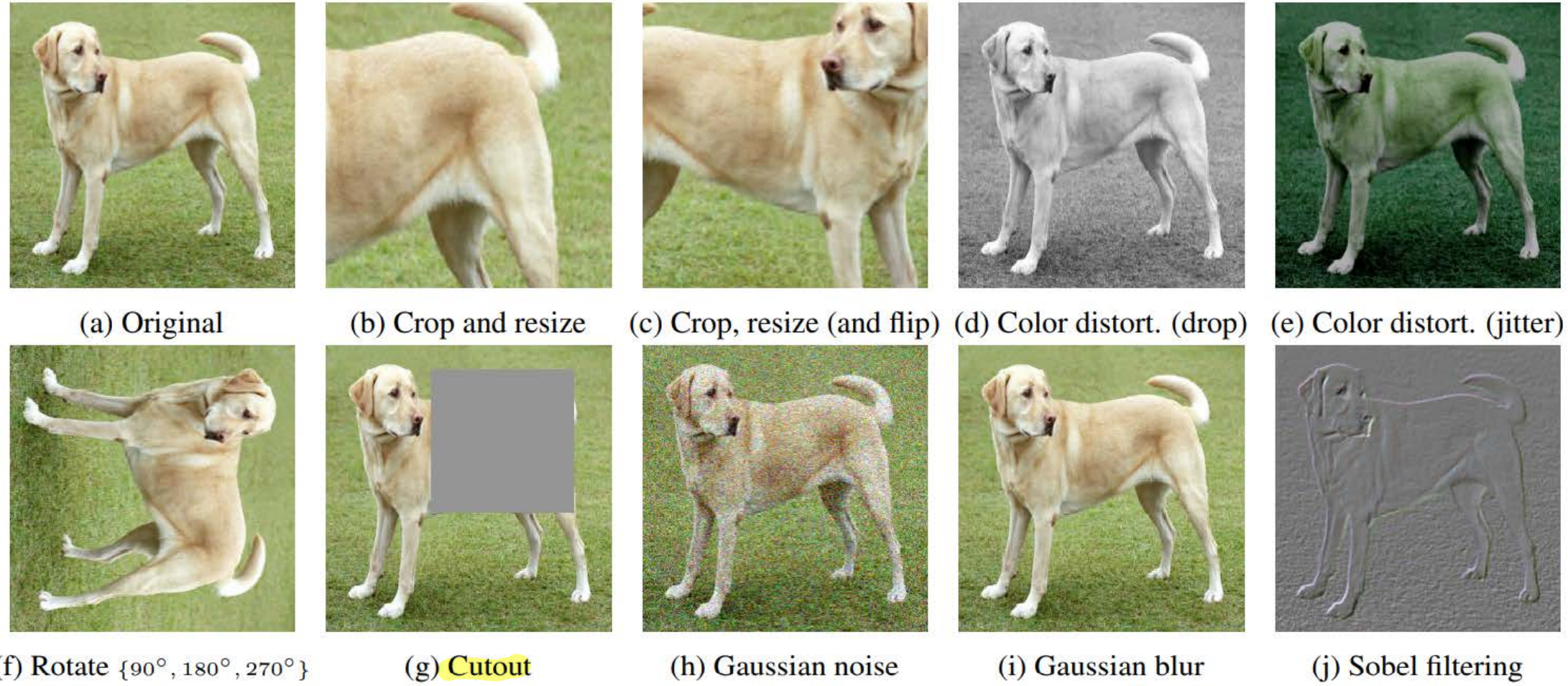
Regularization: Data Augmentation

Model 성능↑ : Training data를 늘려야 함 (Increasing Samples)

→ 해내의 사진의 일부 또는 변형을 여러장 사용하는 것이 중요한 것임.



Data Augmentation



[Chen et al., ICML 2020]

Data Augmentation

Mixup and **CutMix**

강아지 사진과 고양이 사진을 결합한다.

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4

[Yun et al., ICCV 2019]

ITE4052 Computer Vision

Camera & Image Formation

Dong-Jin Kim

Spring 2025