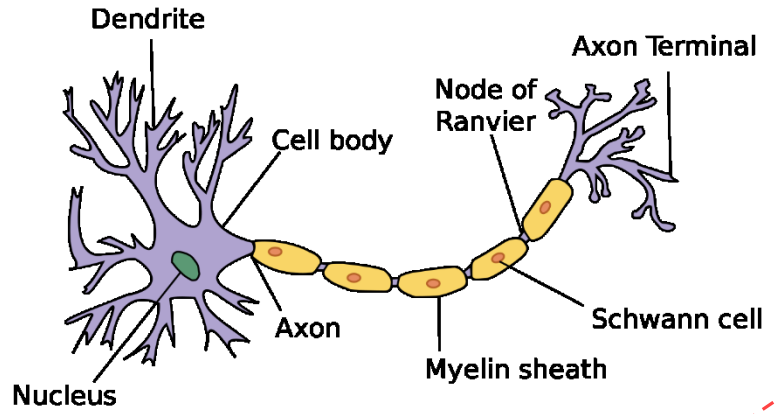
Review & Recap

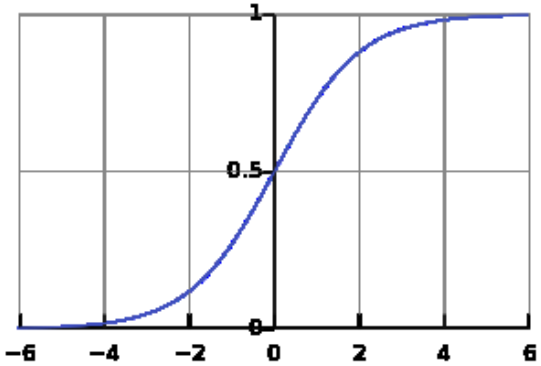
Fast Campus
Start Deep Learning with TensorFlow

The Goal of Deep Learning (Supervised Leraning)

- 내가 가진 data(이 세상에 있는 모든 data 중의 일부 sample)를 가지고 열심히 학습해서 내가 가지지 않은 data가 들어왔을 때 그 data에 대한 무언가를 잘 예측해 보자
- 무언가를 잘 예측?
  - Classification – ex) 개/고양이 분류
  - Regression – ex) 내일의 주가 예측
- 이 때, 어떤 예측에 필요한 특징(feature)를 사람이 직접 rule에 의해서 정해주지 말고 data를 많이 넣어서 data로부터 compute가 직접 feature를 찾아내도록 하자 – data driven approach!
  - 설명은 정확히 못하지만 사람은 이미 이렇게 하고 있는 것 같다
  - 찾아낸 feature는 사람이 해석 못해도 괜찮아 – black box
  - 그렇지만 사람이 이해할 수 있으면 더 좋겠지 – visualization, interpretation

Perceptron – Neuron을 모방해보자





sigmoid activation function

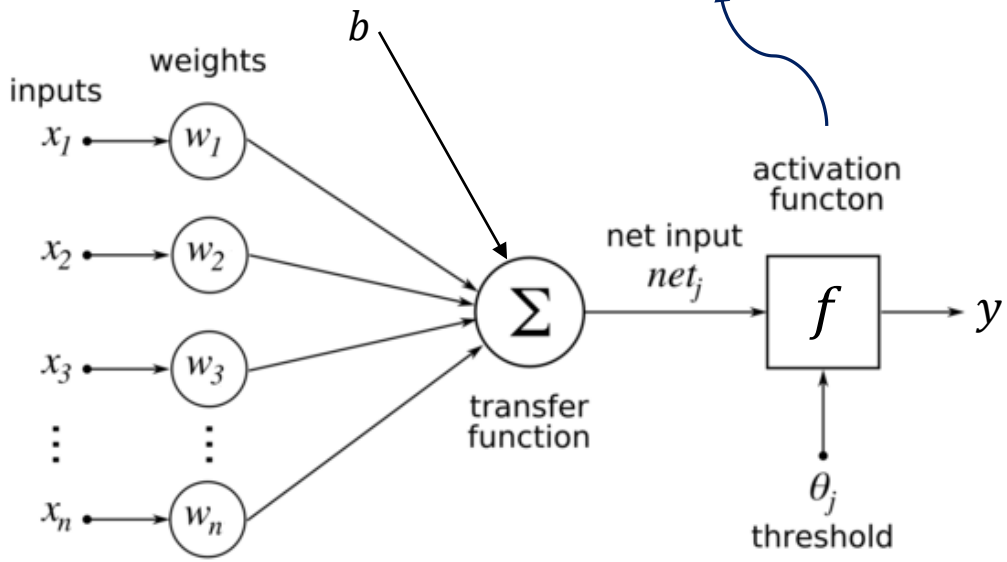
$$f(x) = \frac{1}{1 + e^{-x}}$$

$$y = f(\mathbf{wx} + \mathrm{b})$$

$$\boldsymbol{w} = [w_1\ w_2\ w_3\ \ldots\ w_n]$$

$$\boldsymbol{x} = [x_1\ x_2\ x_3\ \ldots\ x_n]^T$$

목표는 **w**를 찾는 것!

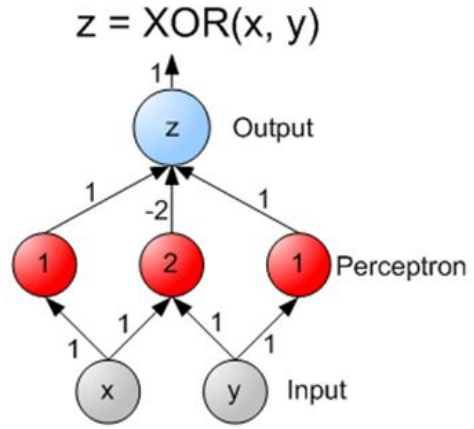


<Perceptron>

Perceptron으로 할 수 있는 것

- Perceptron 하나로는 linearly separable problem을 풀 수 있음
- 이론적으로 Perceptron을 잘 쌓으면(MLP – multi layer perceptron) 모든 함수를 근사할 수 있다
- 문제는 주어진 data만 가지고 우리의 목적을 잘 달성할 수 있는 우리가 모르는 어떤 함수를 어떻게 학습할 것인가이다
  - 어떻게 w를 찾을 것인가?

The *universal approximation theorem* for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds only for restricted classes of activation functions, e.g. for the sigmoidal functions. (Wikipedia.org)



Training

- Perceptron을 적당히 잘 쌓자 → network 구성
- Loss function을 잘 정하자
  - 우리가 원하는 목적이 달성되면 Loss가 0이 되고 원하는 목적에서 멀어지면 Loss가 커지도록
  - Classification – Cross-Entropy
  - Regression – Mean Squared Error
- Weight를 random한 값으로 초기화하자
- 가능하면, loss function을 weight로 미분해서 최솟값을 찾아보자 안되면 iterative 하게 loss가 줄어드는 방향으로 weight값을 조금씩 바꿔서 optimal solution을 찾아보자
  - Linear regression이 아닌 경우는 미분해서 한번에 최솟값이 되는 weight를 찾기 힘들기 때문에 iterative solution을 사용한다 → Gradient Descent!

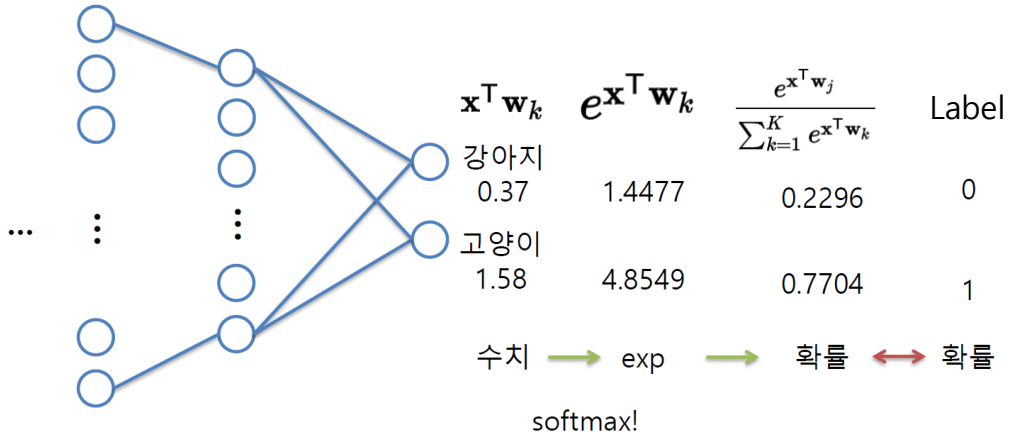
Loss Function – Regression

- 마지막 layer의 activation function
  - None! – network 마지막 layer의 output을 그대로 사용함

- Loss function – Mean Squared Error(MSE)
  - $L = \frac{1}{n}\sum(y - y^*)^2$, y는 $label$(정답), y^*는 $network\ output$ (예측값)

Loss Function – Classification

- 마지막 layer의 activation function
  - Softmax $P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$



- Loss Function – Cross Entropy
  - $L = \sum -y \log y^*$, y는 $label$(정답), y^*는 $network\ output$(예측값, softmax 결과)

Kullback–Leibler(KL) Divergence

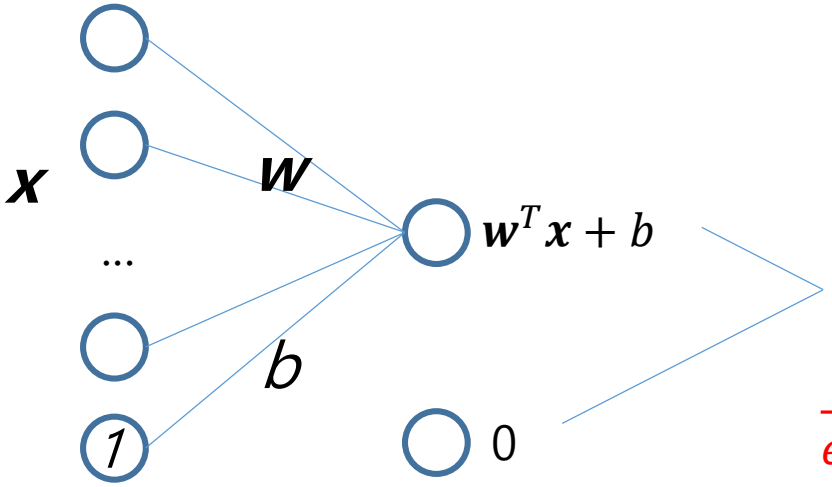
p의 entropy

p, q의 cross entropy

- 두 확률 분포간의 거리(?)를 나타냄
  - $D_{KL}(p \parallel q) = p \log \frac{p}{q} = p \log p + (-p \log q) = -H(p) + H(p, q) = H(p, q)$
  - p는 label, q는 network output 이라고 할 경우, network의 목표는 label이 나타내는 확률 분포 p와 최대한 가까운 q를 학습하고자 하는 것임
  - 일반적으로 label은 one-hot encoding 하기 때문에, 각 label의 값이 확률이라고 생각하면 p의 entropy, $H(p) = 0$
  - $D_{KL}(p \parallel q) > 0$

Binary Classification

- 마지막 layer의 activation function
  - Sigmoid – $H(x) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b) = \frac{1}{1+ e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}} = P(y|\boldsymbol{x})$

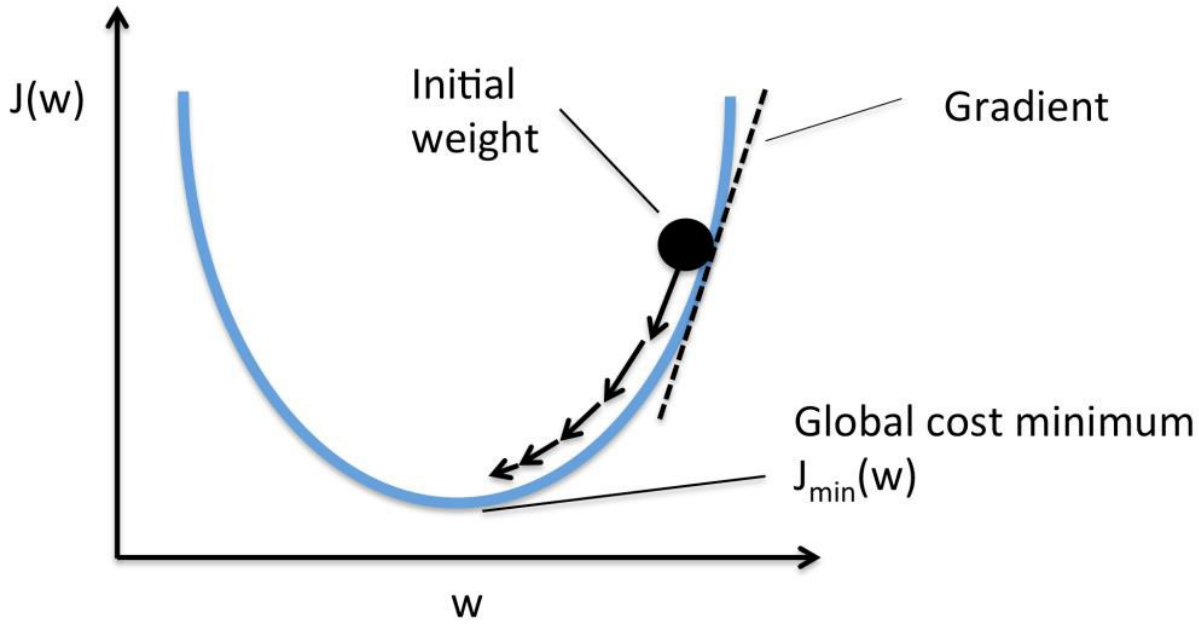


Softmax → sigmoid

$$\frac{e^{\boldsymbol{w}^T\boldsymbol{x}+b}}{e^{\boldsymbol{w}^T\boldsymbol{x}+b} + e^0} = \frac{1}{1 + e^{-(\boldsymbol{w}^T\boldsymbol{x}+b)}}$$

Gradient Descent

$$w_{new} = w - \eta \frac{\delta L}{\delta w}$$

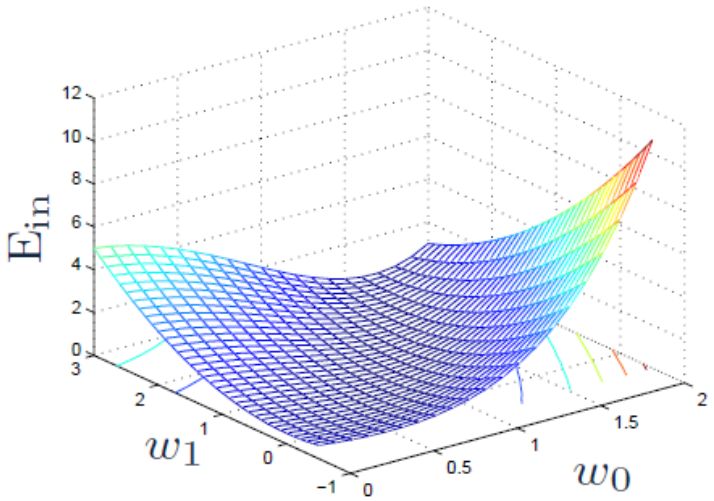


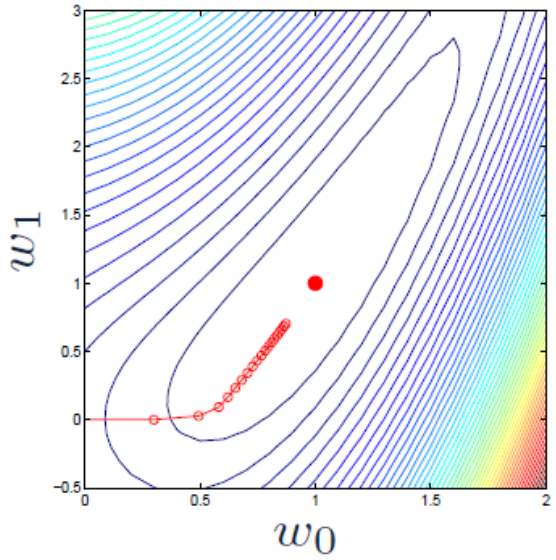
- 방향 : 그 지점에서의 – gradient
- 속력(보폭) : learning rate(η)

- 현재지점에서 gradient의 방향이 Loss를 가장 크게 증가시키는 방향이므로 그 반대 방향(-)으로 아주 조금씩(learning rate) 이동하자

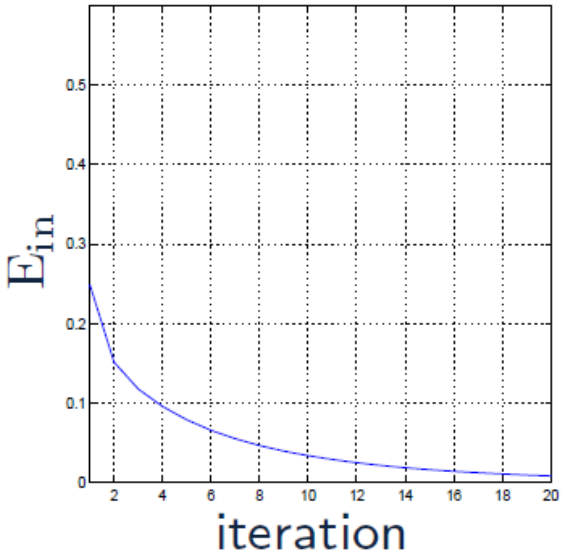
Gradient Decent Example

- **Global minimum 0 at (1,1)**

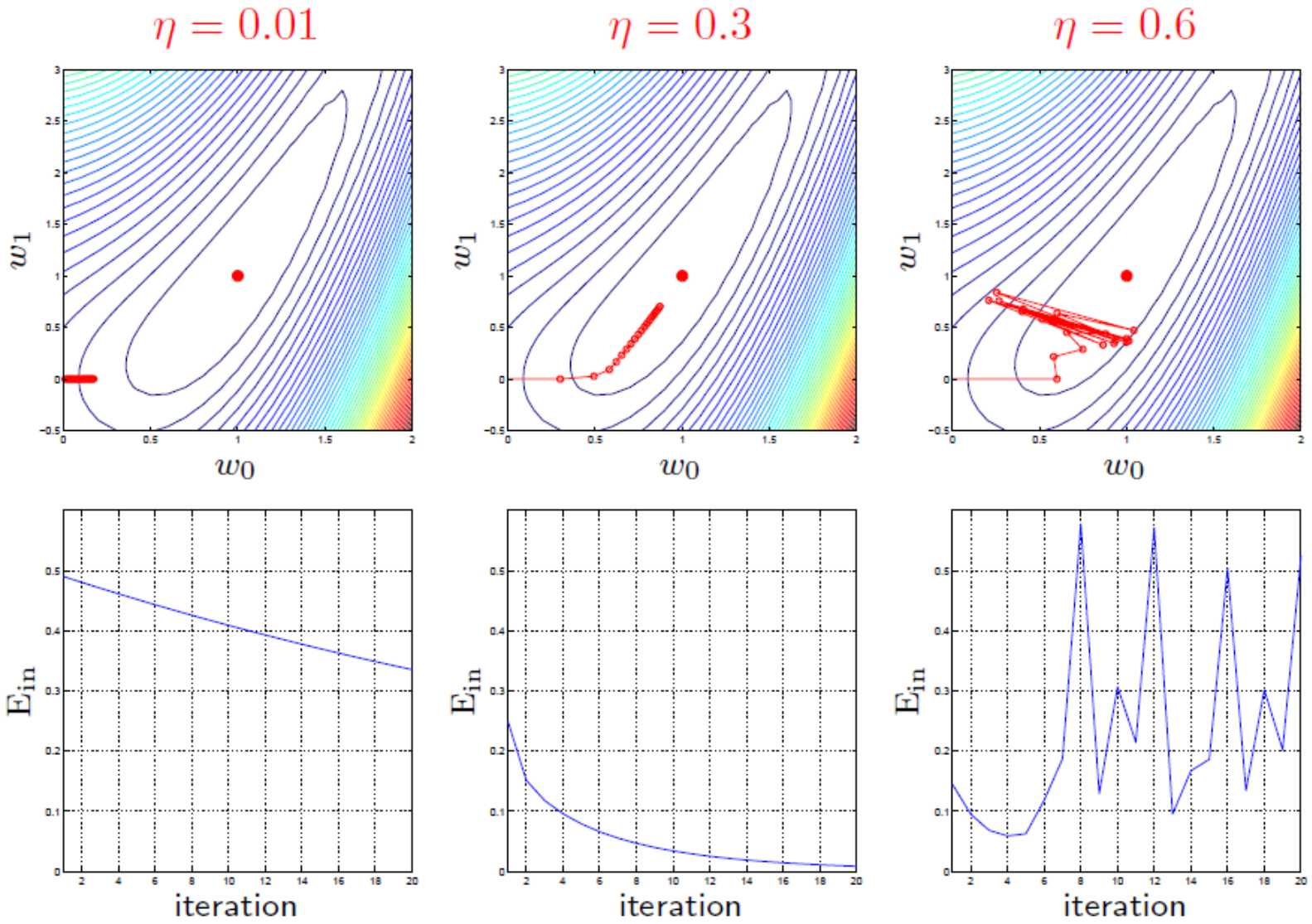




- **Start at (0,0)**
  - # iterations (steps) $= 20$
  - step size $\eta = 0.3$

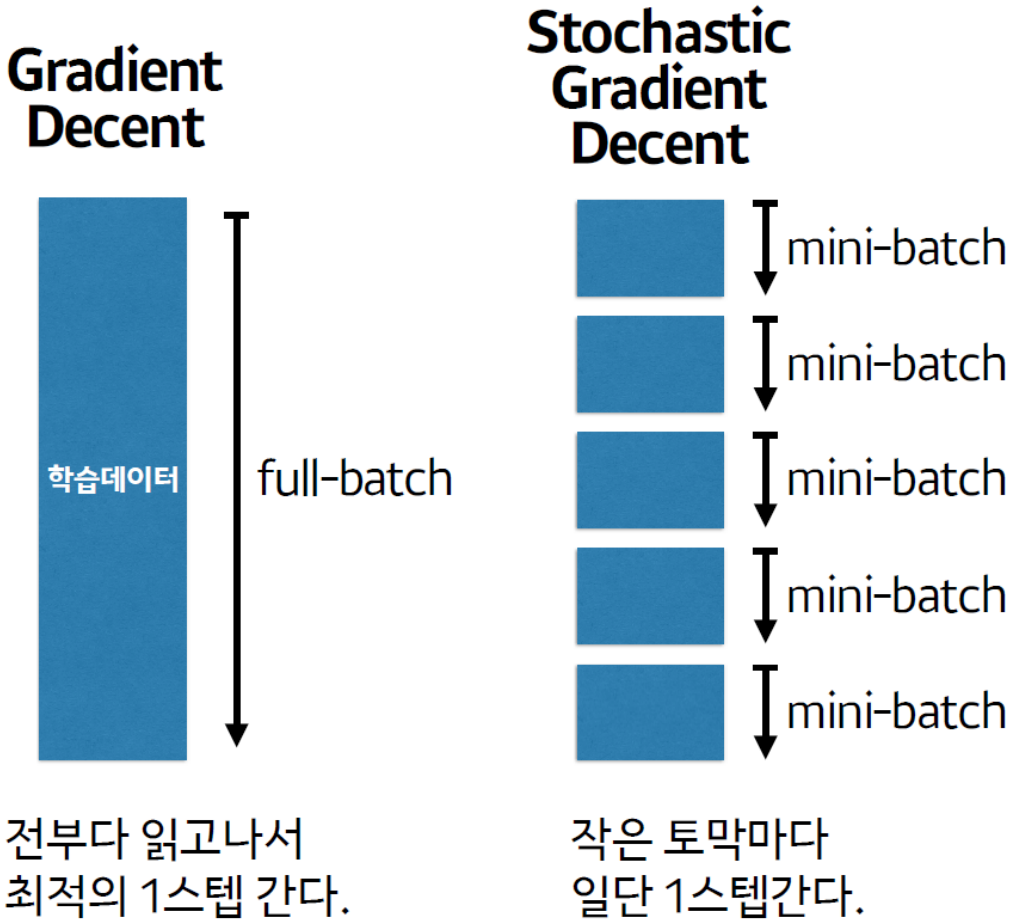


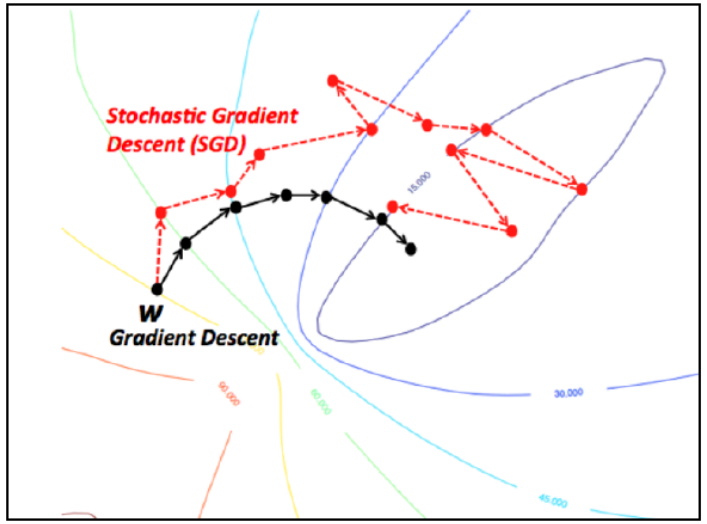
Learning Rate



Stochastic Gradient Descent, Mini-batch Training

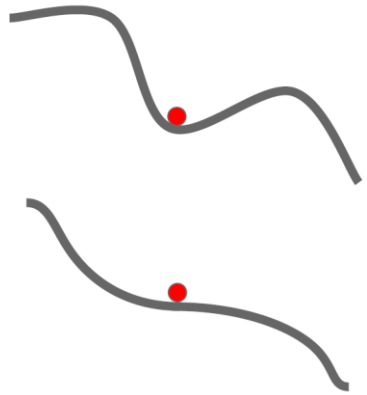
- Data가 너무 많아서 한번에 다 넣고 학습하면 시간도 오래걸리고, memory도 부족하게 됨



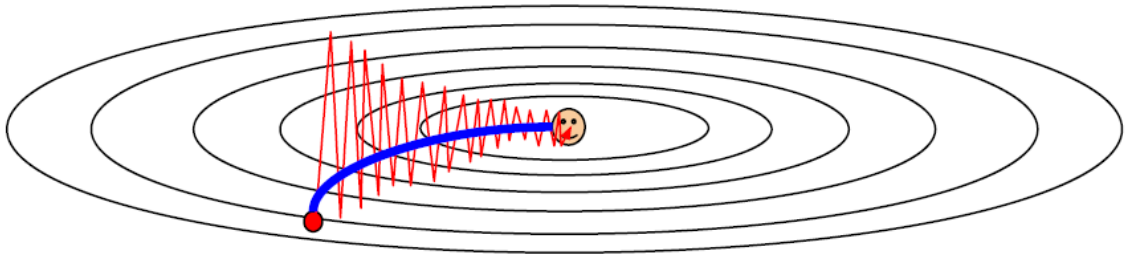


Problems of Gradient Descent

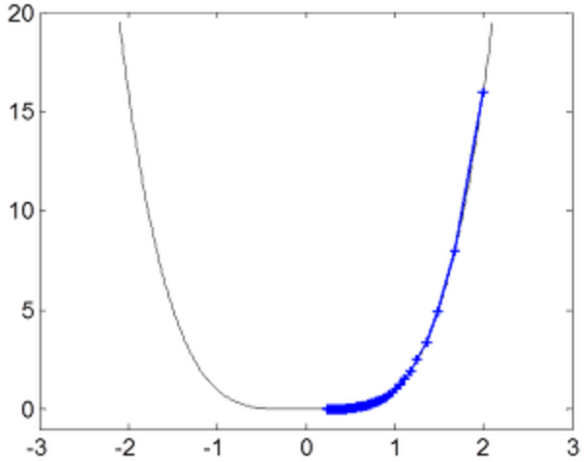
- (It can) stuck at local minima or saddle point(zero gradient)



- Poor Conditioning

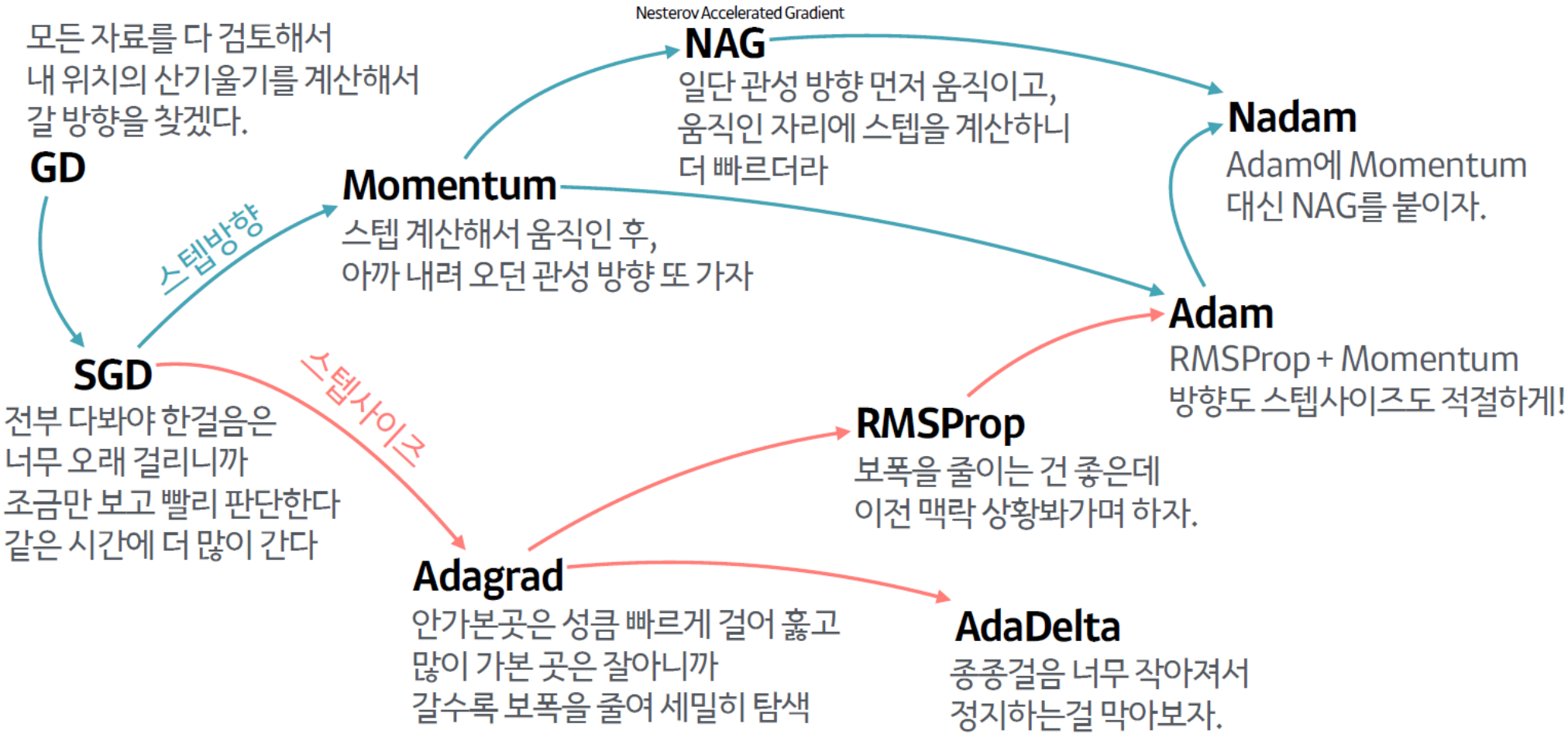


- Slow….
  - The closer to the optimal point, the smaller the gradient becomes.



Optimization Methods

- 잘 모르겠으면 Adam을 쓰자

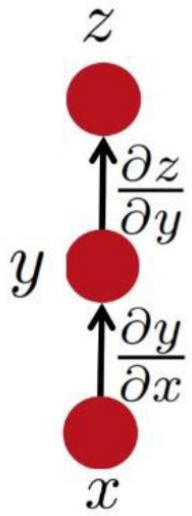


Slide Credit : 하용호@Kakao

그런데 미분은 어떻게??

- $w_{new} = w - \eta \frac{\delta L}{\delta w}$, 결국 $\frac{\delta L}{\delta w}$을 구하고 싶은데, w로 L을 어떻게 미분할까?
- Chain Rule을 이용!

Simple Chain Rule

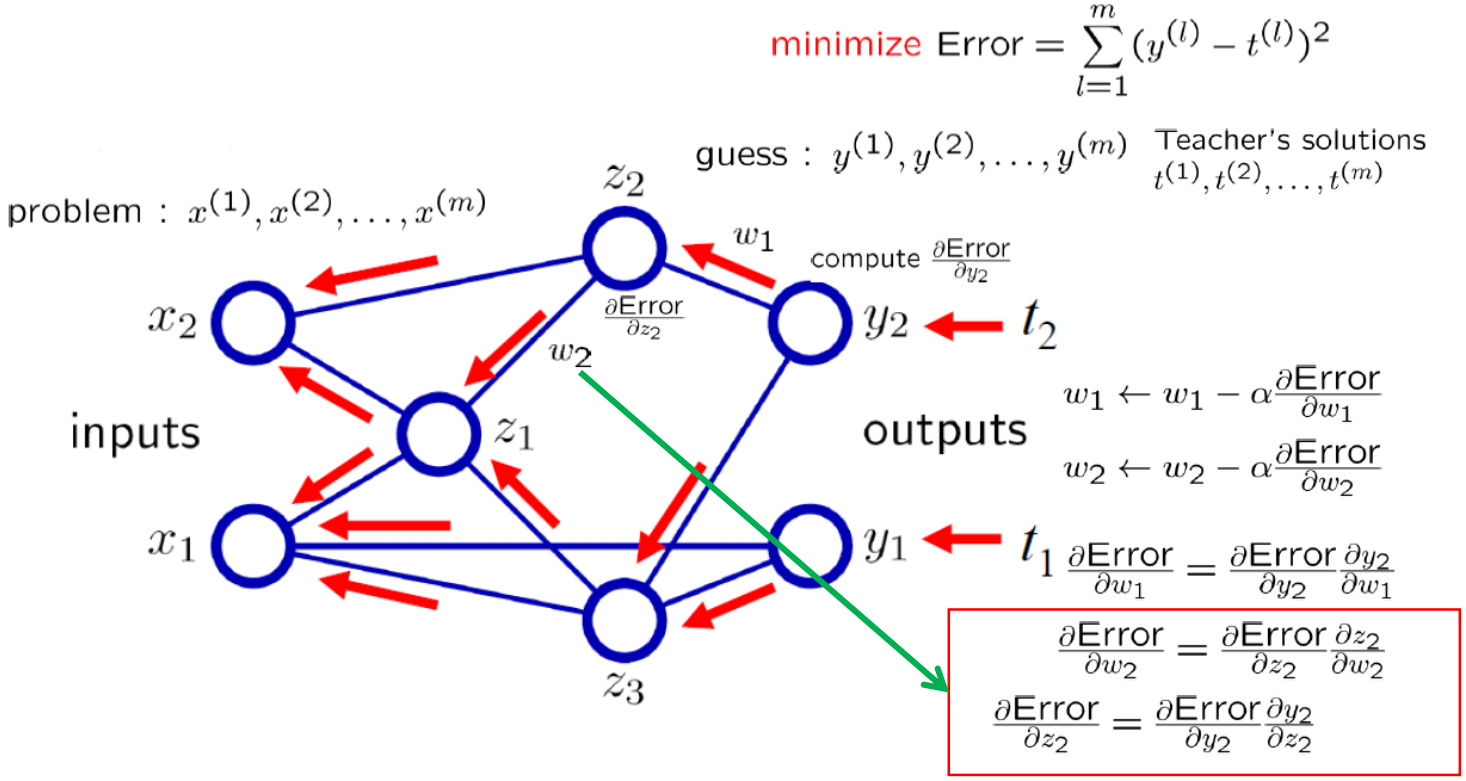


$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$

$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$

$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$



Back Propagation

Backpropagation: a simple example

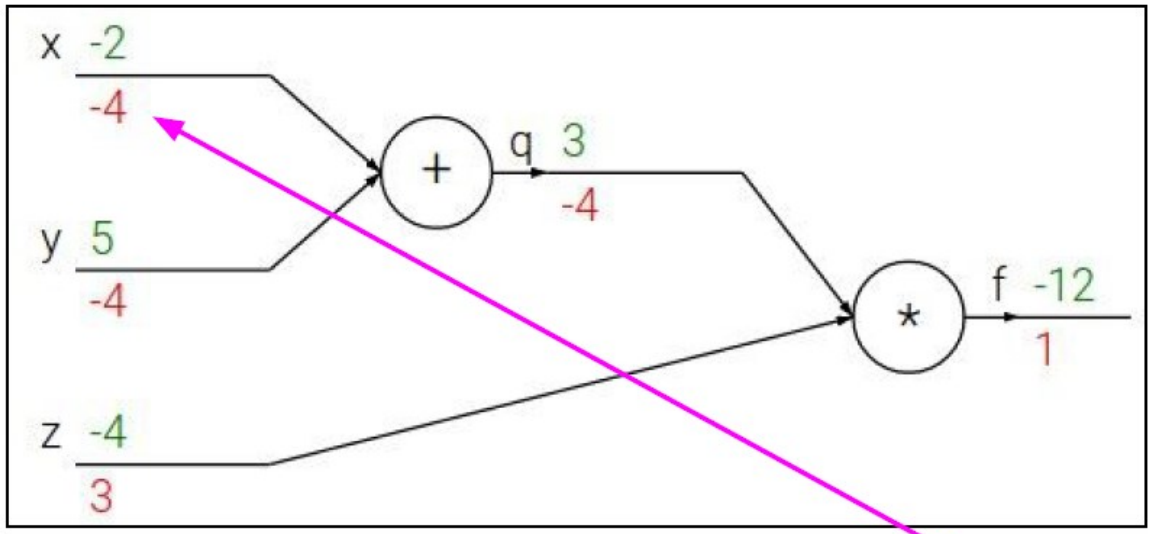
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

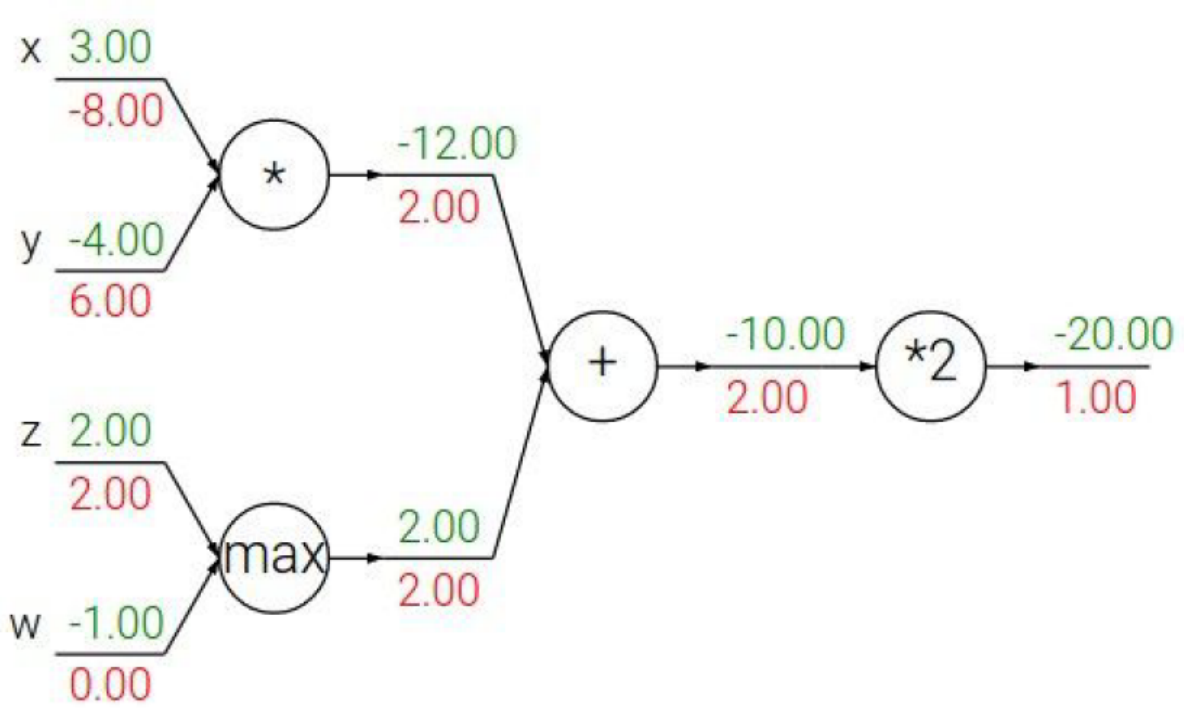
Back Propagation(Example)

Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

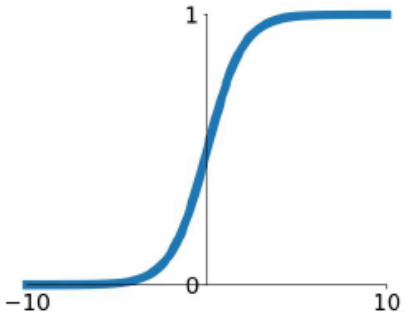
mul gate: gradient switcher



Activation Functions

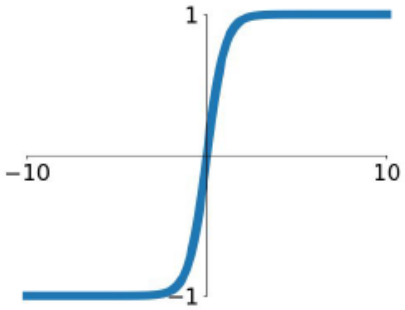
Sigmoid

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



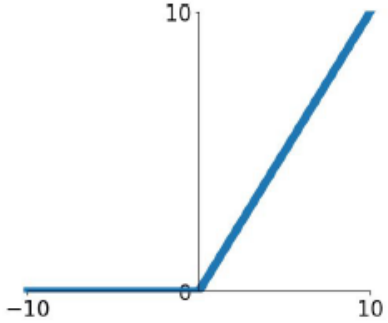
tanh

$$\tanh(x)$$



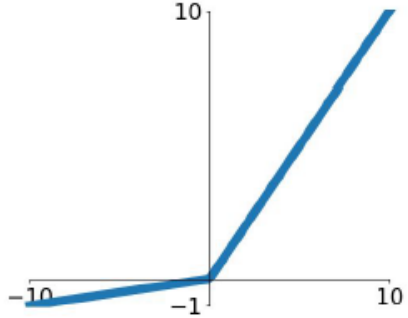
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

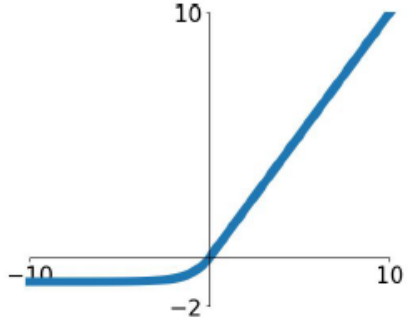


Maxout

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

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions – Rule of Thumb

- 1st layer 부터 n-1번째 layer까지
  - ReLU를 쓰자 (RNN계열은 sigmoid나 tanh를 씀)
  - 잘 안되면, leaky ReLU, ELU 등을 써보자
- 마지막(n번째) layer
  - Classification – sigmoid 혹은 softmax
  - Regression - 안씀

Weight Initialization

- Weight는 random value로 initialization한다
- 이 때 mean=0의 Gaussian distribution에서 random value를 뽑는데, variance를 얼마로 하느냐에 따라서
  - Variance를 크게 하면 weight에 큰 값들이 많아지고 이럴 경우 sigmoid나 tanh의 saturation 영역으로 들어가서 vanishing gradient 문제가 생길 수 있음
  - Variance를 작게 하면 weight 값들이 너무 작아져서 layer가 쌓이면 쌓일수록 output이 점점 0으로 수렴함 → 역시 gradient가 0에 가까워짐
  - Input의 수가 많으면 variance를 작게 하고, input의 수가 적으면 variance를 크게 하자, back-propagation 때에는 반대이므로 output의 수에 따라서도 variance를 조절하자 – Xavier init
- Batch normalization을 쓰면 이 문제를 해결할 수 있음

forward: $\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$

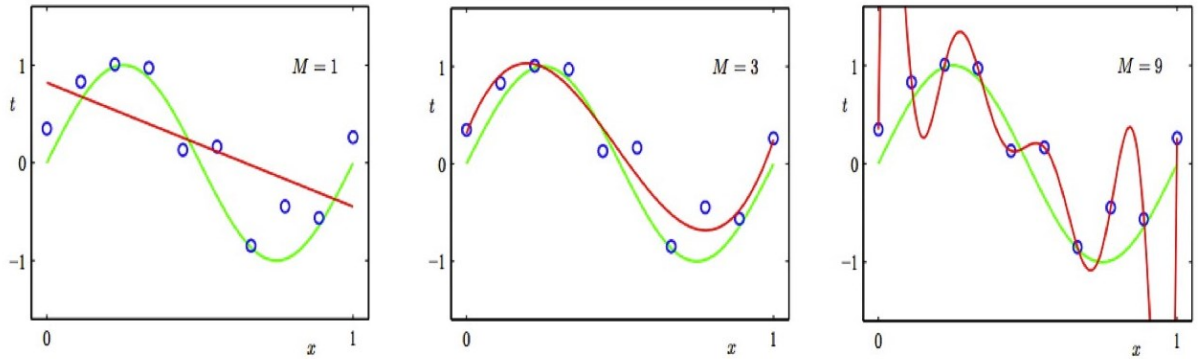
backward: $\mathrm{Var}(W_i) = \frac{1}{n_{\text{out}}}$

Xavier Initialization: $\mathrm{Var}(W_i) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$

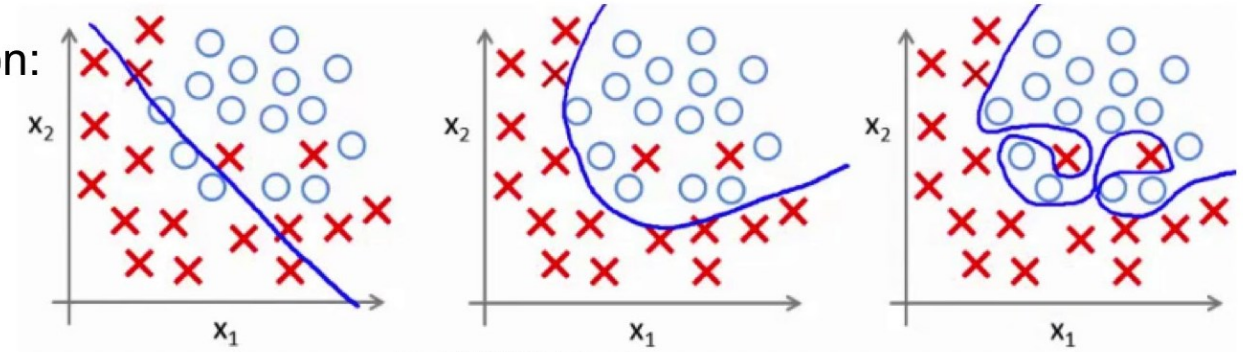
Fight Against Overfitting

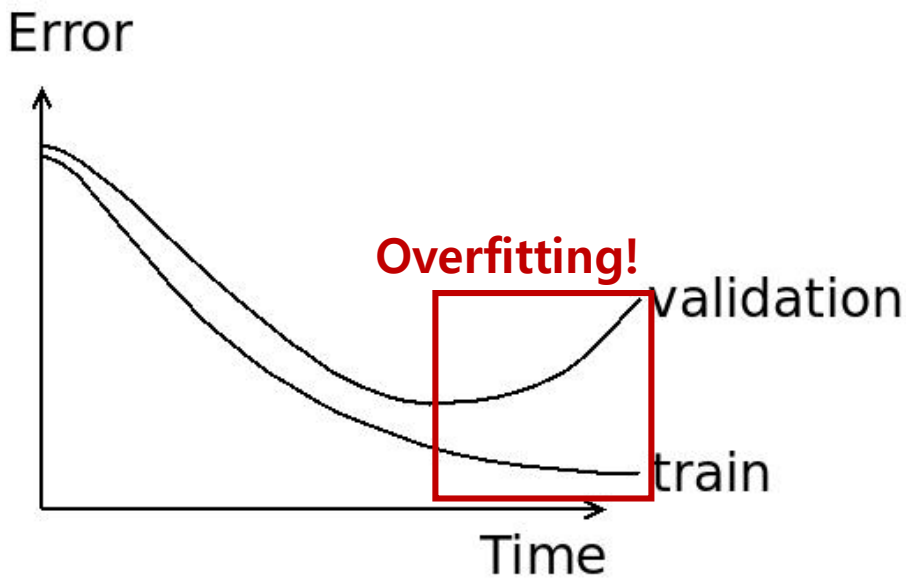
- Data가 많지 않아서 발생
- 학습한 data에만 최적화되어서, 학습하지 않은 data(test data)에 대한 추론 성능이 악화되는 현상

Regression:



Classification:





Weight Decay

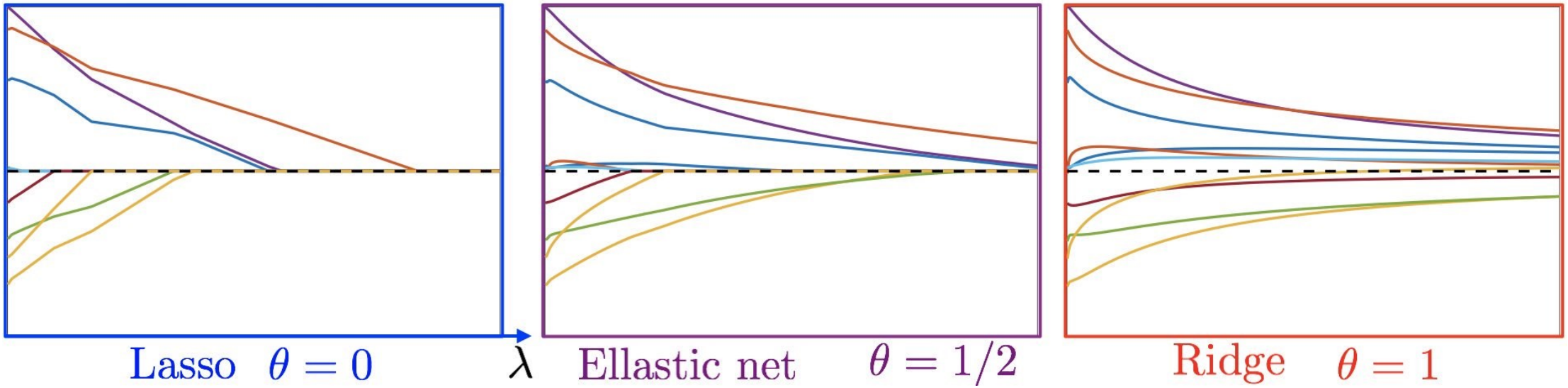
$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

In common use:

Ridge **L2 regularization** $R(W) = \sum_k \sum_l W_{k,l}^2$ (Weight decay)

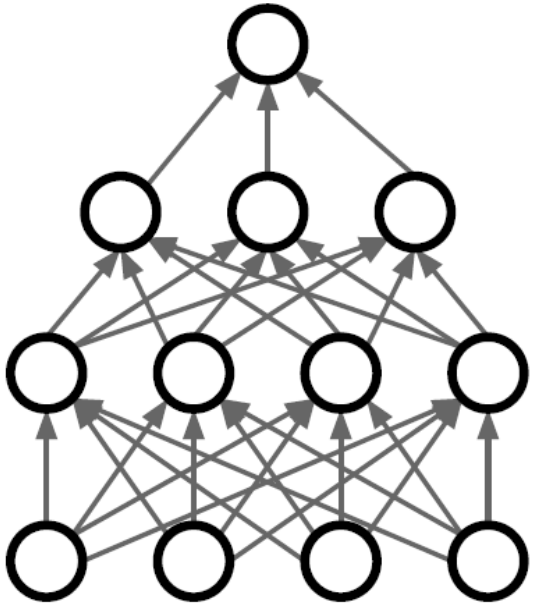
Lasso L1 regularization $R(W) = \sum_k \sum_l |W_{k,l}|$

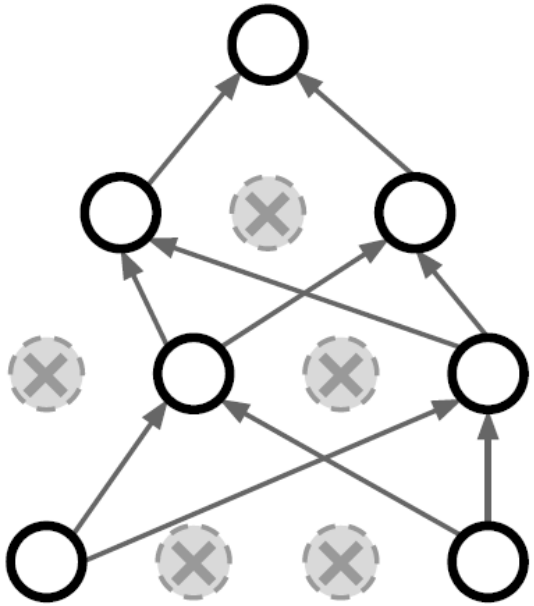
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$



Dropout

- In each forward pass, randomly set some neurons to zero
- Probability of dropping is a hyper parameter; 0.5 is common





Dropout is training a large **ensemble** of models (that share parameters).

Each binary mask is one model

An FC layer with 4096 units has $2^{4096} \approx 10^{1233}$ possible masks!
Only ~ 10^{82} atoms in the universe…

Slide Credit : Stanford CS231n

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

After normalization, allow the network to squash the range if it wants to

- Activation function 전에 주로 사용
  - Activation function의 active 영역으로 분포를 바꾸는 역할
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Note, the network can learn:

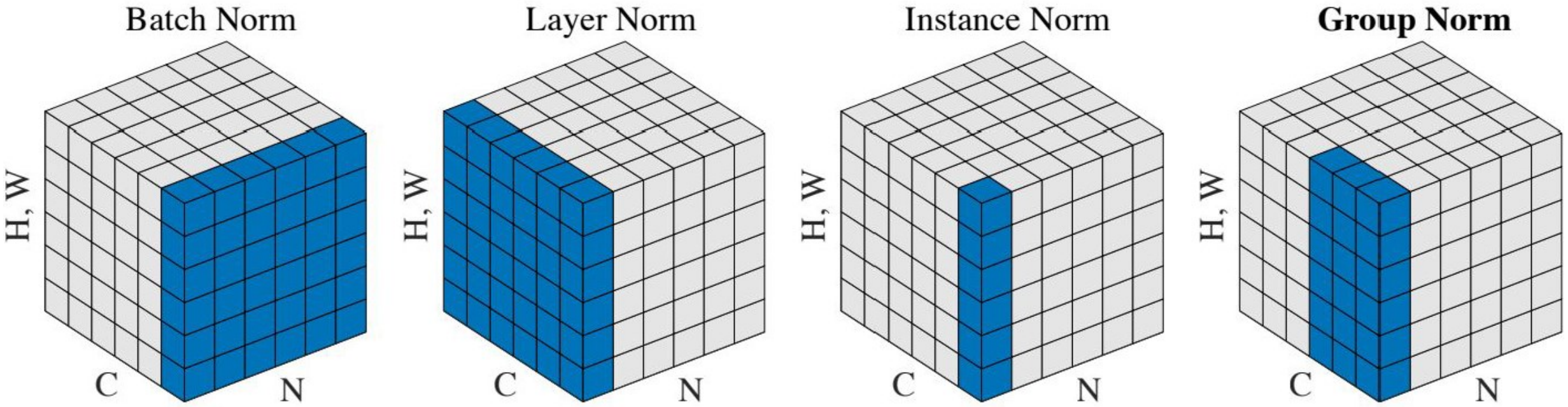
$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

Slide Credit : Stanford CS231n

Other Normalizations



Slide Credit : Stanford CS231n

Regularization: A Common Pattern

- Training : Add some kind of randomness

Output (label) Input (image)

$$y = f_W(x, z)$$

Random mask

- Testing : Average out randomness

$$y = f(x) = E_z\left[f(x, z)\right] = \int p(z) f(x, z) dz$$

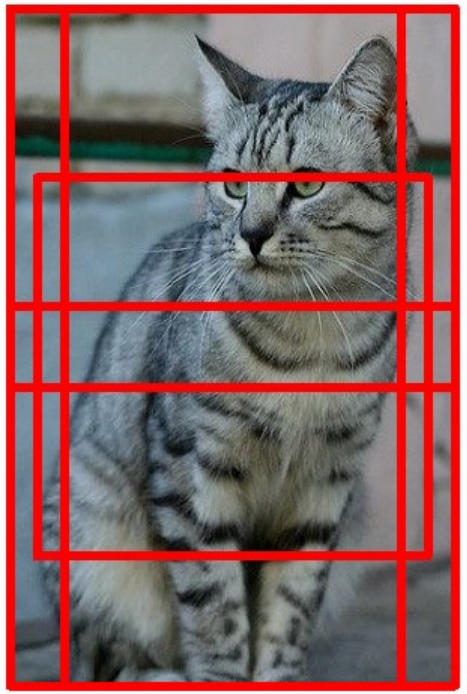
Slide Credit : Stanford CS231n

Data Augmentation - Example

- Training : sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224x224 patch



- Testing : average a fixed set of crops

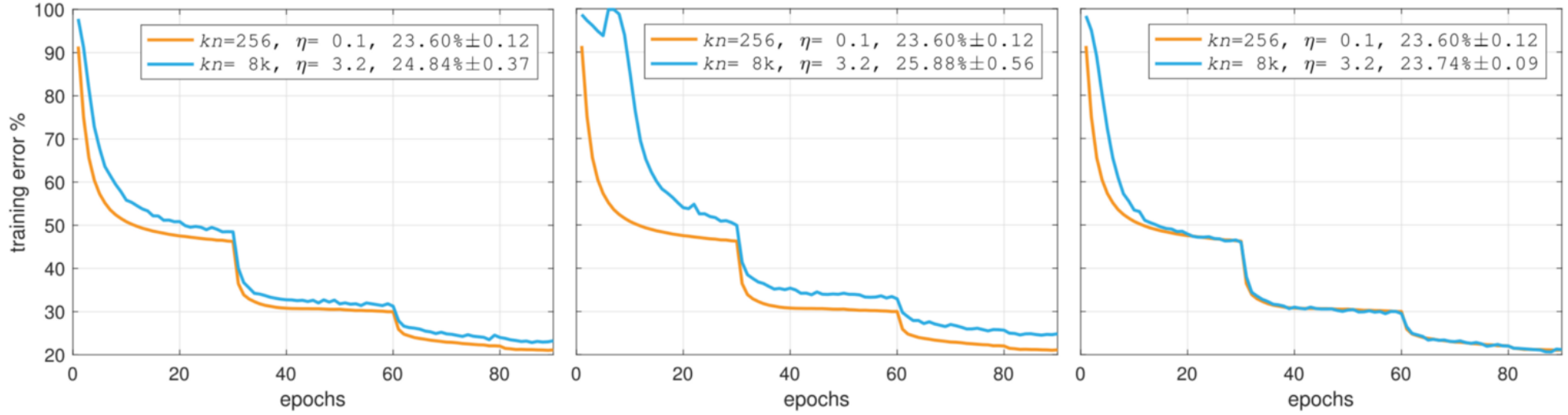
ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224x224 crop: 4 corners + center, + flips

Slide Credit : Stanford CS231n

Learning Rate Decay

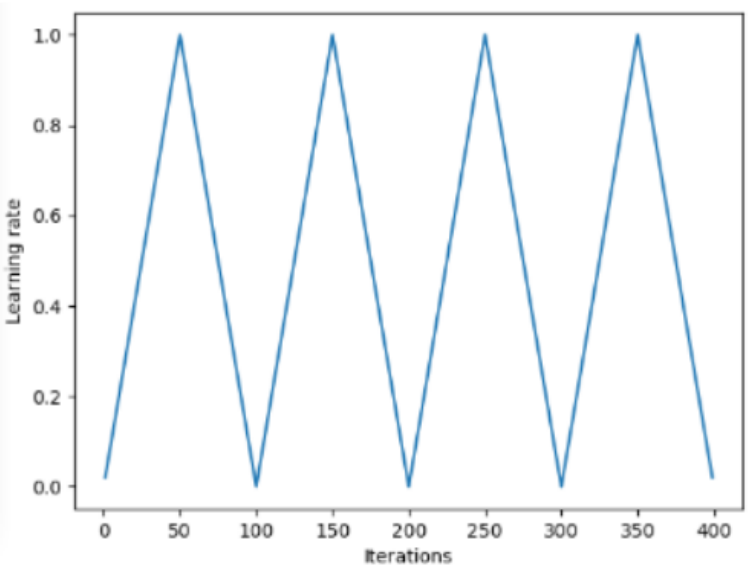
- Learning rate이 너무 크면 수렴을 못할 가능성이 있고, 너무 작으면 local minima 혹은 saddle point에서 못빠져나옴
- Learning Rate Decay
  - 처음에는 크게 움직이다가 일정 조건이 되면 learning rate을 낮춰서 점점 작게 움직이는 방법

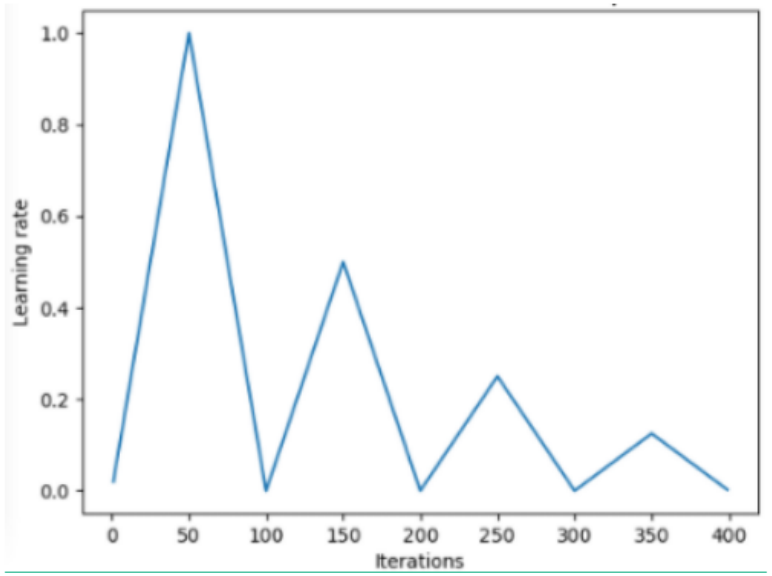


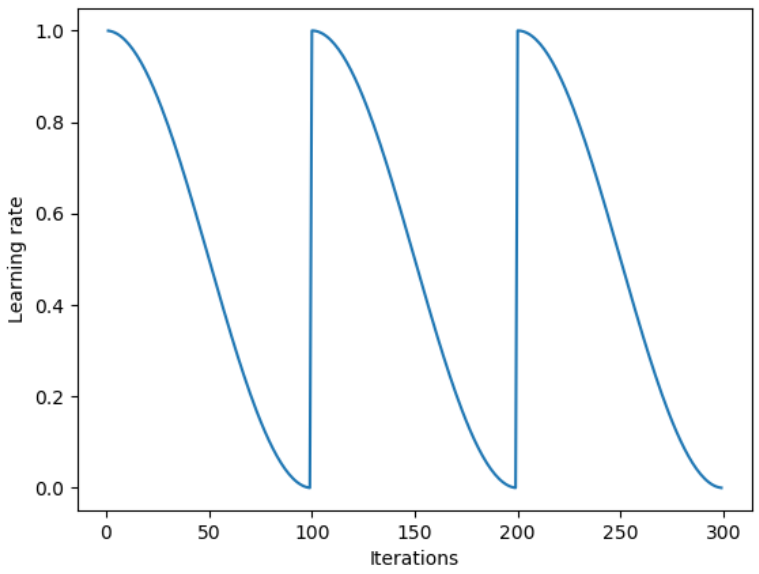
(a) no warmup (b) constant warmup (c) gradual warmup

Cyclic Learning Rate

- Learning rate decay로 saddle point를 빠져나갈 수 있을까?
- Saddle point에서는 learning rate을 키워서 빠져나가는 것이 효과적일 수 있음 → Learning rate을 주기적으로 변경해보자

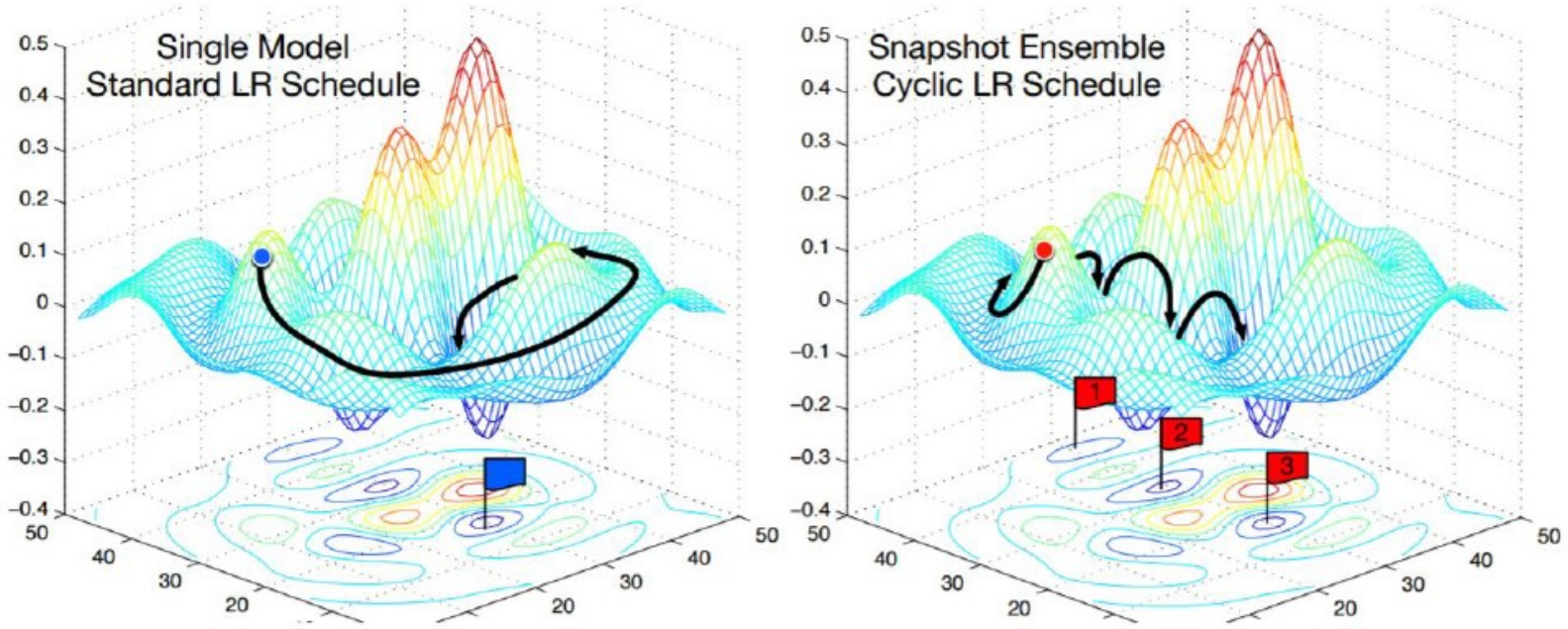


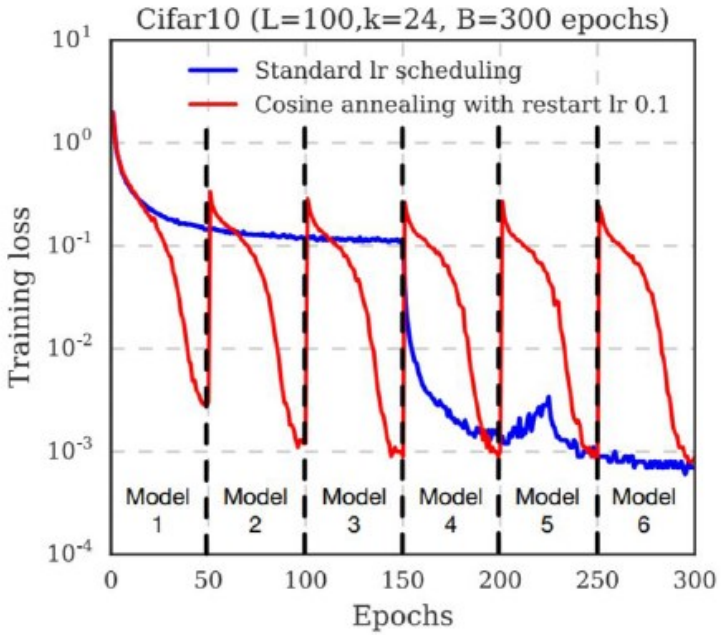




Model Ensembles: Tips and Tricks

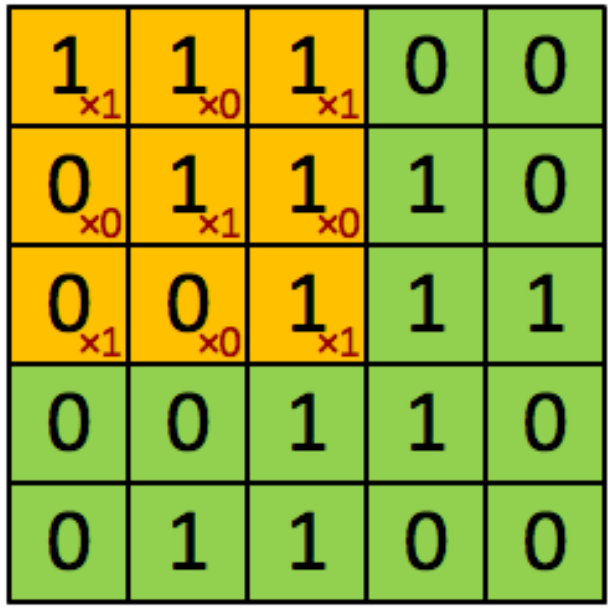
- Instead of training independent models, use multiple snapshots of a single model during training!

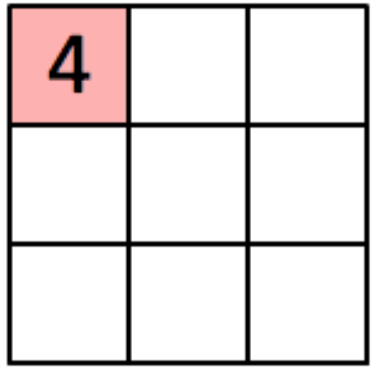




Slide Credit : Stanford CS231n

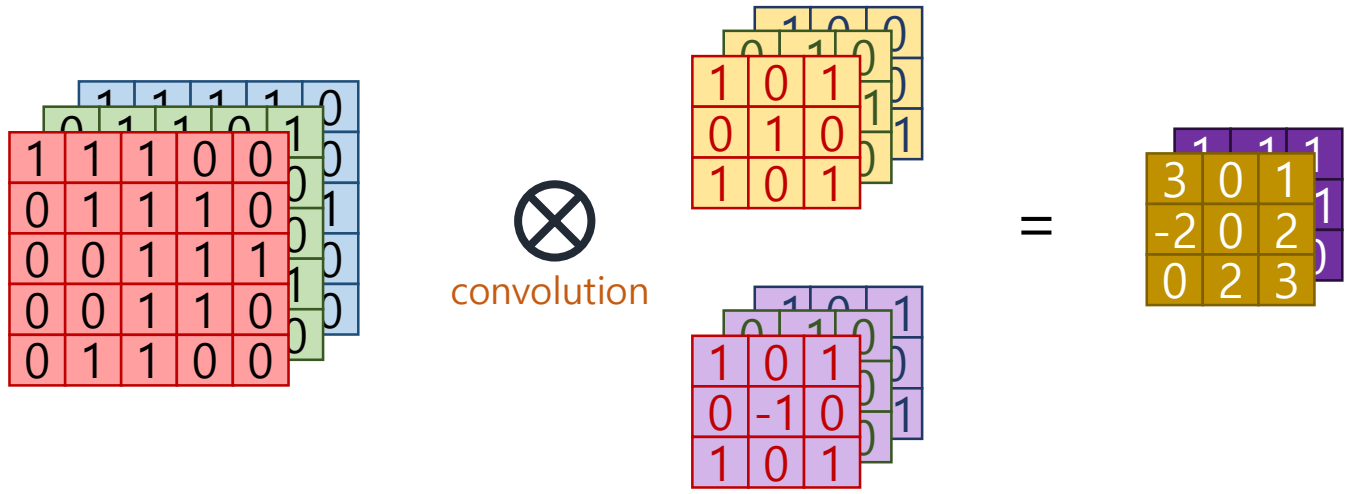
Convolution





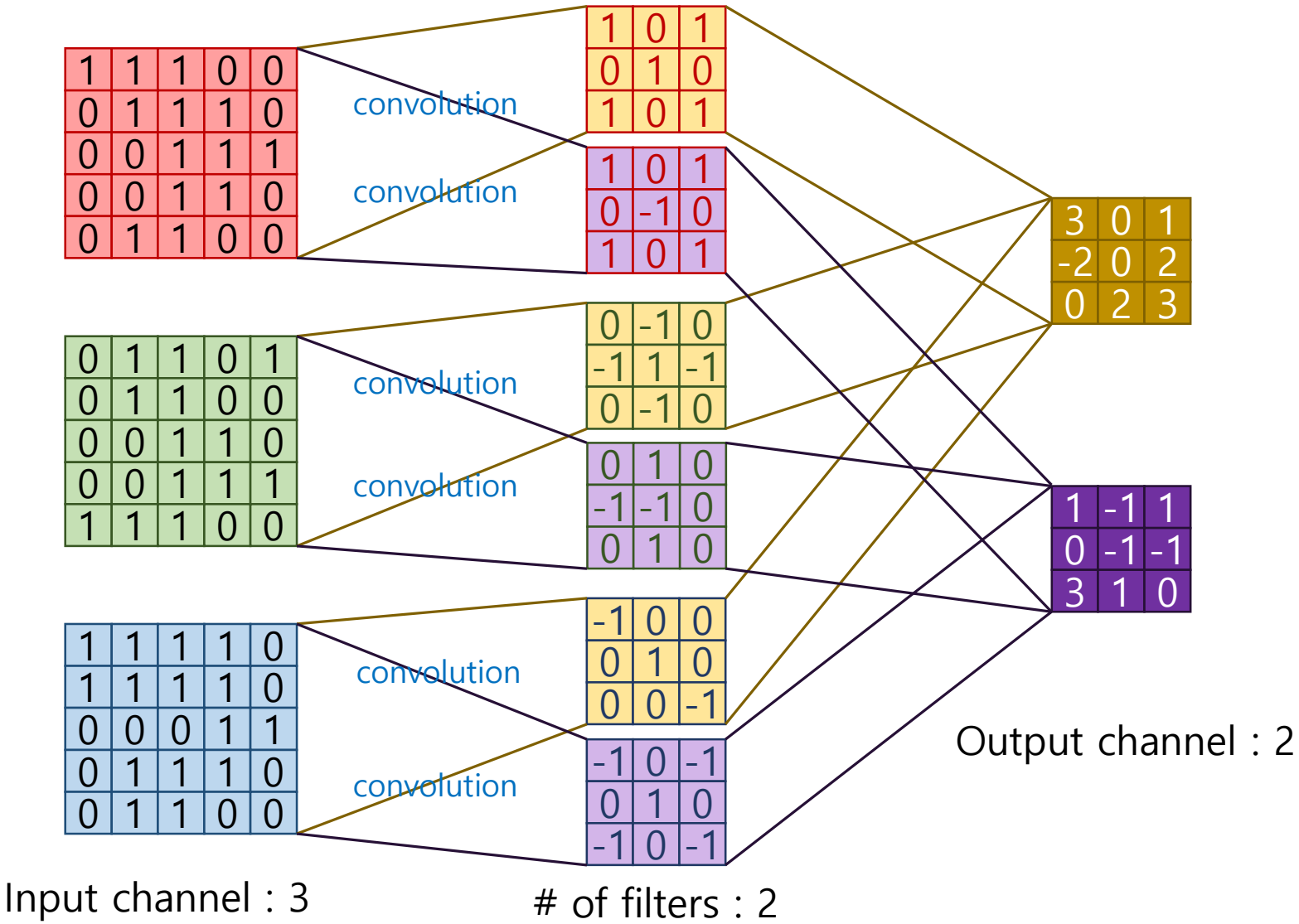
Credit : Leonardo's gitbook

Convolution
(Multi Channel, Many Filters)

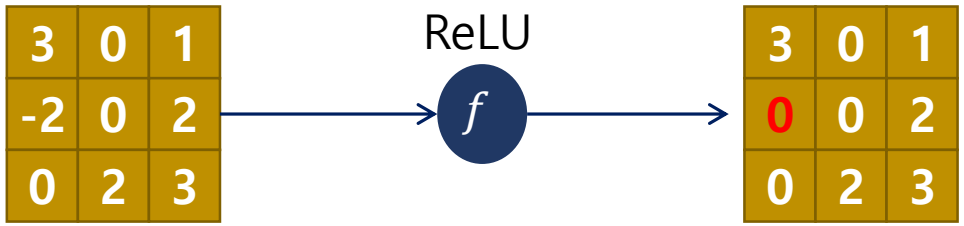


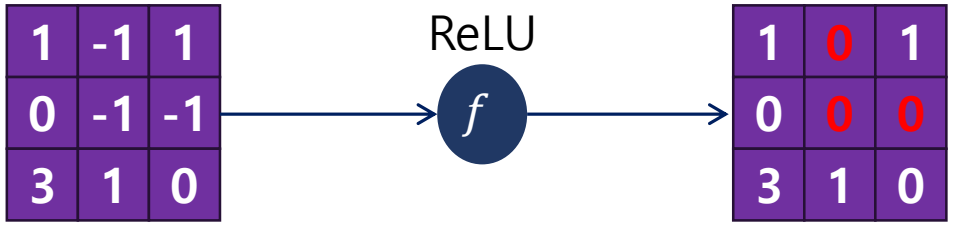
Input channel : 3 # of filters : 2 Output channel : 2

Convolution
(Multi Channel, Many Filters)



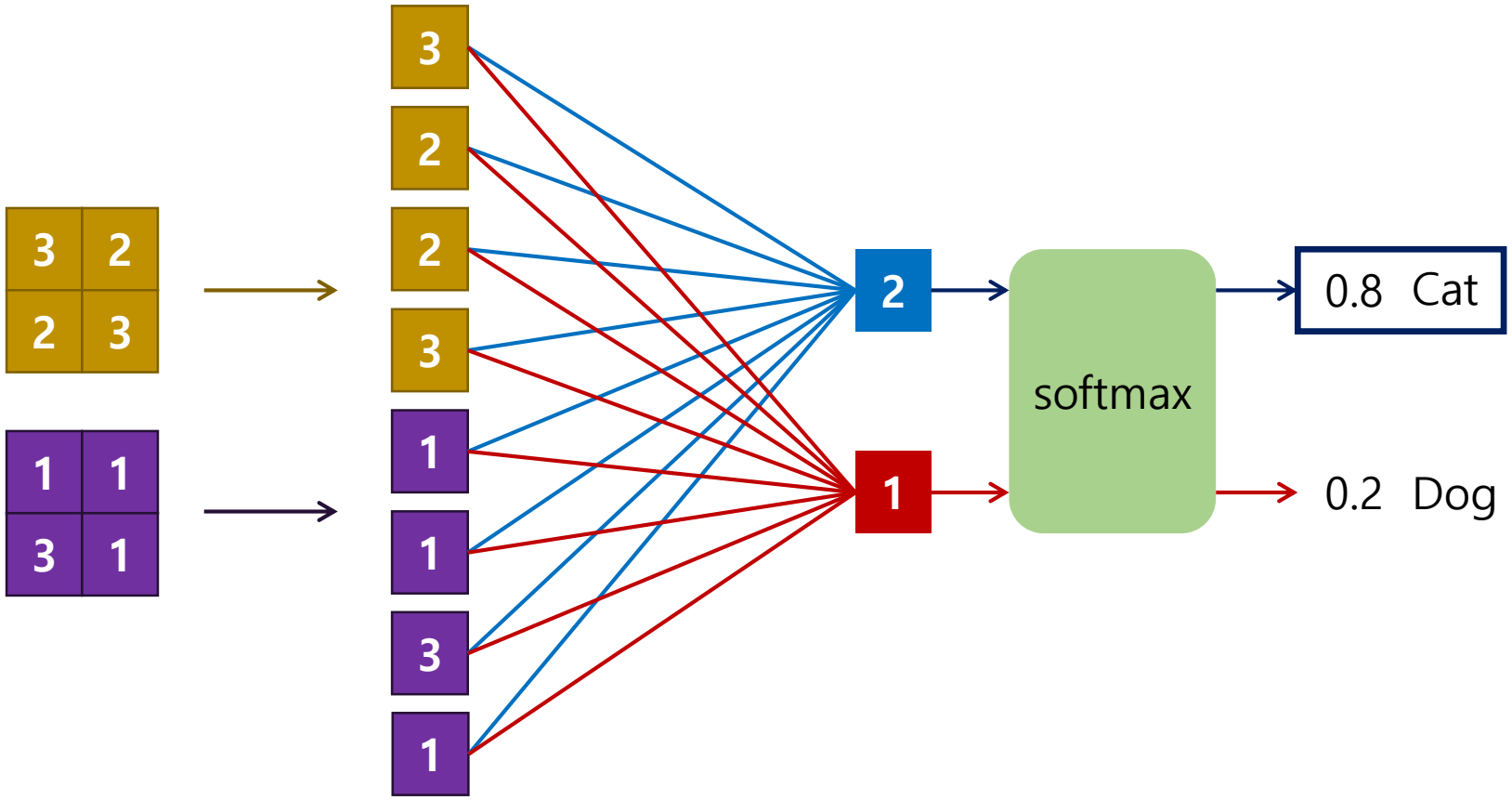
ReLU





2x2 Max Pooling with Stride=1

Fully-Connected Layer



CNN의 특징

- Convolution Layer – parameter(weight) sharing
- Good for local invariance – pooling
- 연산량은 Convolution layer가 대부분을 차지
- Parameter 수는 FC layer가 대부분을 차지

Model	Params (M)	Conv (%)	FC (%)	Ops (M)	Conv (%)	FC (%)
AlexNet	61	3.8	96.2	725	91.9	8.1
VGG-F	99	2.2	97.8	762	87.4	12.6
VGG-M	103	6.3	93.7	1678	94.3	5.7
VGG-S	103	6.3	93.7	2640	96.3	3.7
VGG-16	138	10.6	89.4	15484	99.2	0.8
VGG-19	144	13.9	86.1	19647	99.4	0.6
NIN	7.6	100	0	1168	100.0	0.0
GoogLeNet	6.9	85.1	14.9	1566	99.9	0.1

CNN 최근 trend

- Skip connection 을 많이 사용함
- 1x1 convolution으로 channel 수를 줄이는 bottle neck layer를 사용하여 deep network에서 channel이 커지는 것을 막음
- 연산량을 줄이기 위한 depthwise separable conv 등이 사용됨
- Global average pooling을 사용하여 fully connected layer를 1개 혹은 0개로 구성