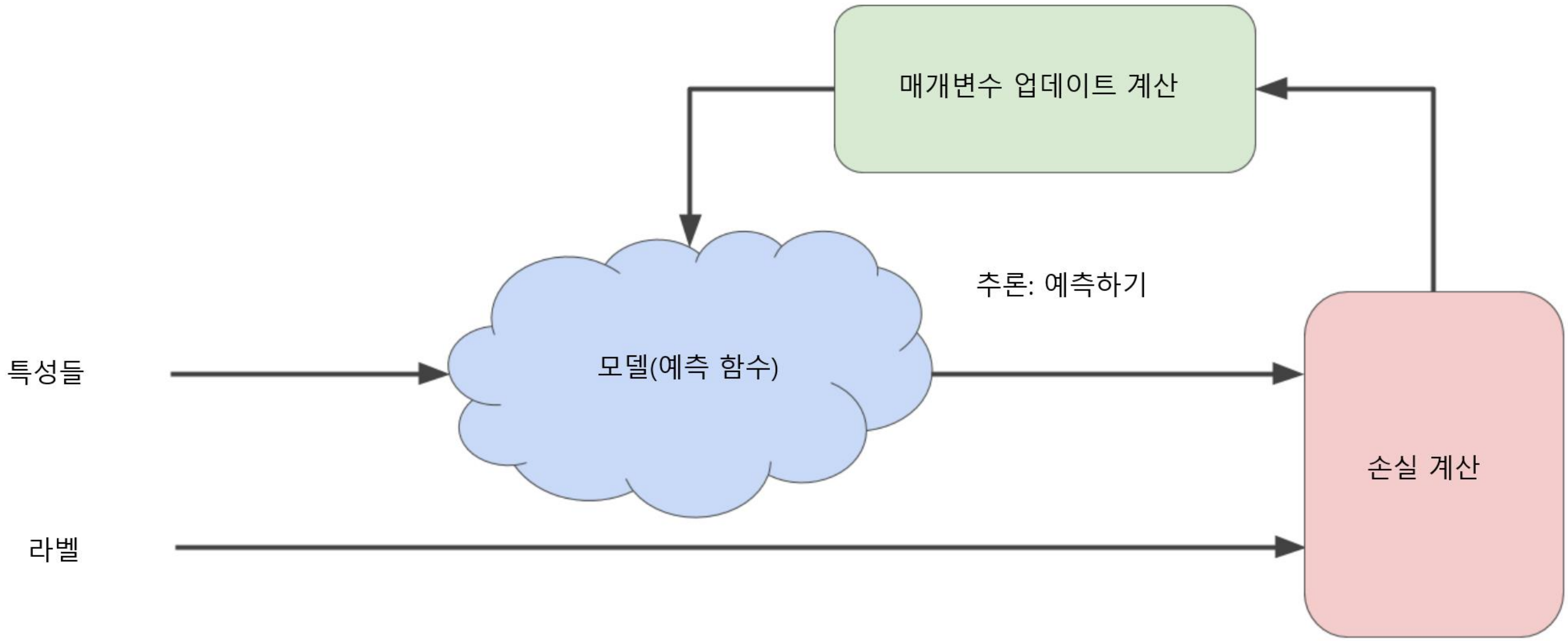
Reducing Loss

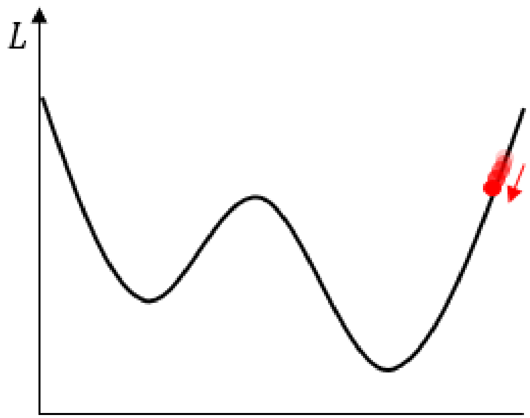
❑ 경사하강법 (Gradient Descent Algorithm)



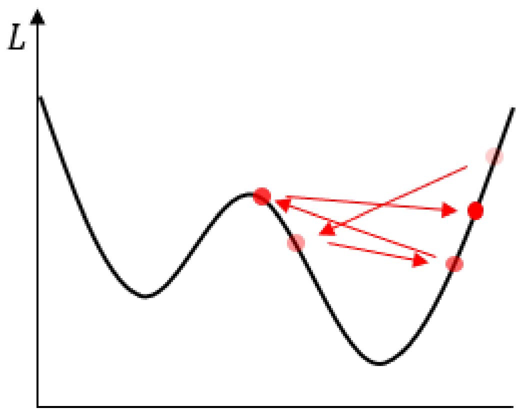
Reducing Loss

❑ 경사하강법 (Gradient Descent Algorithm)

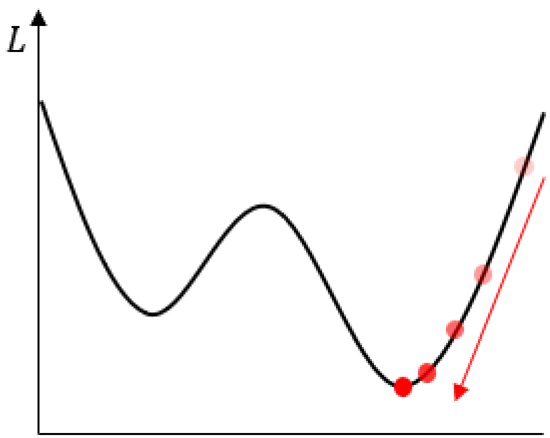
- 경사하강법 알고리즘은 기울기에 학습률 (Learning Rate) 또는 보폭이라 불리는 스칼라를 곱하여 다음 지점을 결정



학습률이 너무 작을 때



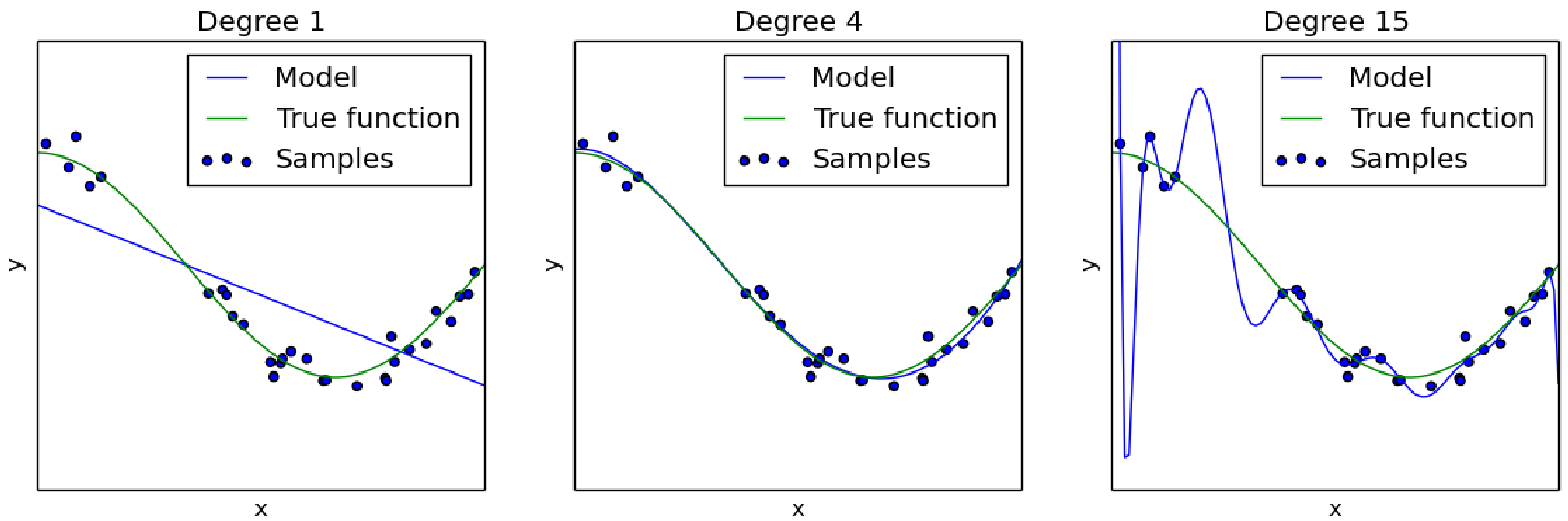
학습률이 너무 클 때



학습률이 너무 적당할 때

Challenge of Learning

❑ 적당히…

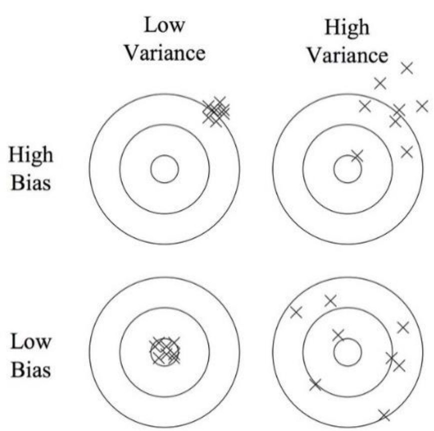


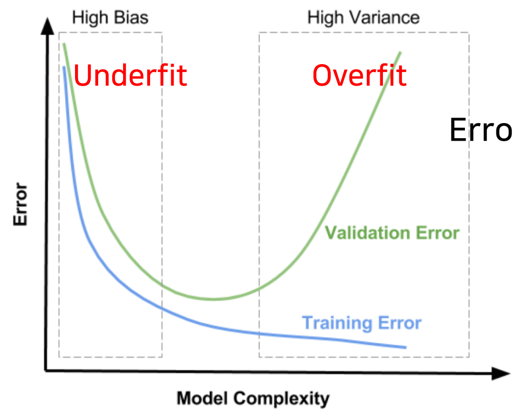
Under-fitted Model　　Balanced-Model　　Over-fitted Model

Challenge of Learning

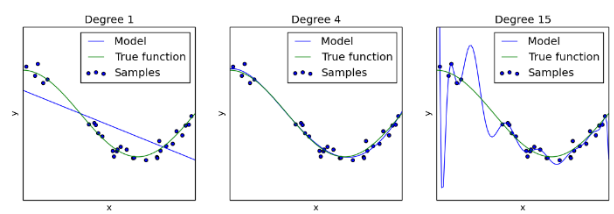
❑ 편향-분산 (Bias-Variance) Trade-off

- 편향 :학습 알고리즘에서 잘못된 가정을 했을 때 발생하는 오차
- 분산 : 트레이닝 셋에 내재된 작은 변동 때문에 발생하는 오차
- 모델을 선택할 때,
  - 트레이닝 데이터의 규칙을 정확하게 포착하는 것 뿐만이 아니라,
  - 보이지 않는 범위에 대해서 일반화(generalization)까지 하는 것이 이상적 (But, 불가능)



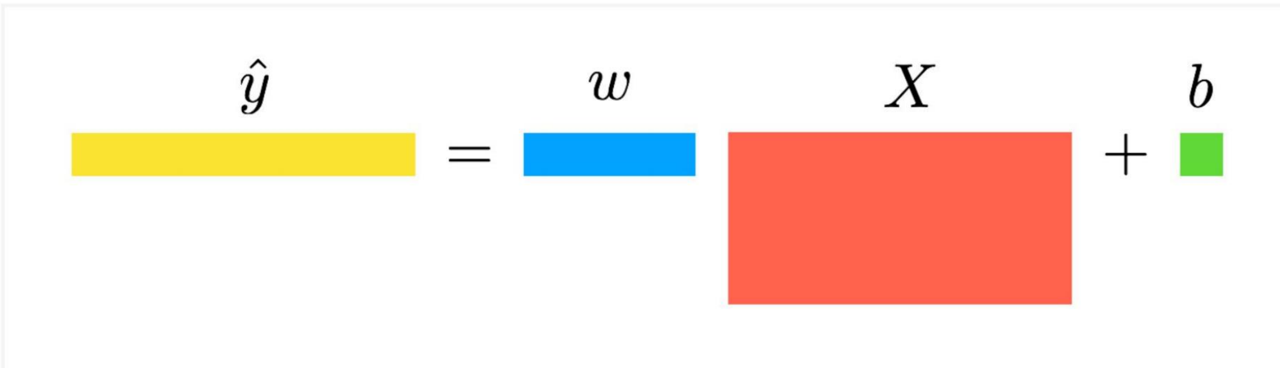


Error(X) = noise(X) + bias(X) + variance(X)



Challenge of Learning

❑ 미분 기반의 학습



$$\hat{y} = wX + b$$

$$\mathcal{L} = \frac{1}{m}||\hat{y} - y||^2$$

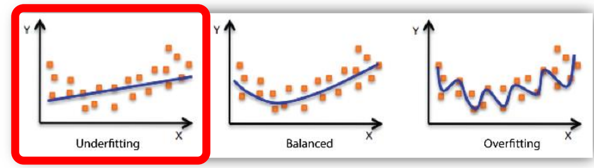
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{2}{m}(\hat{y} - y)X^\intercal$$

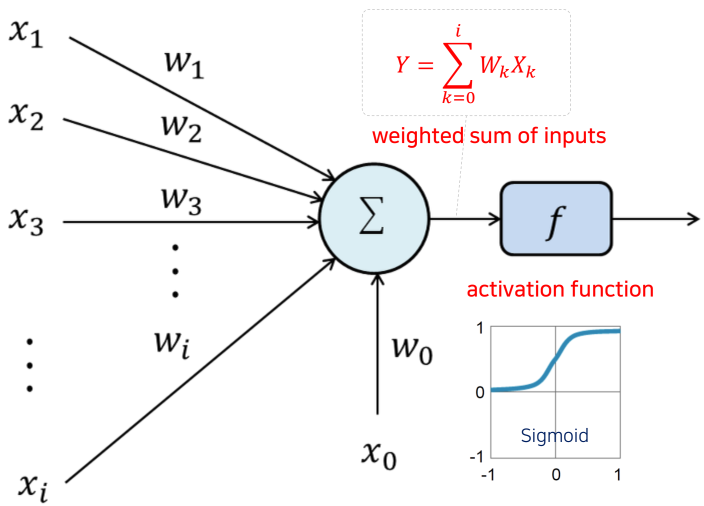
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{2}{m}(\hat{y} - y)\vec{1}$$

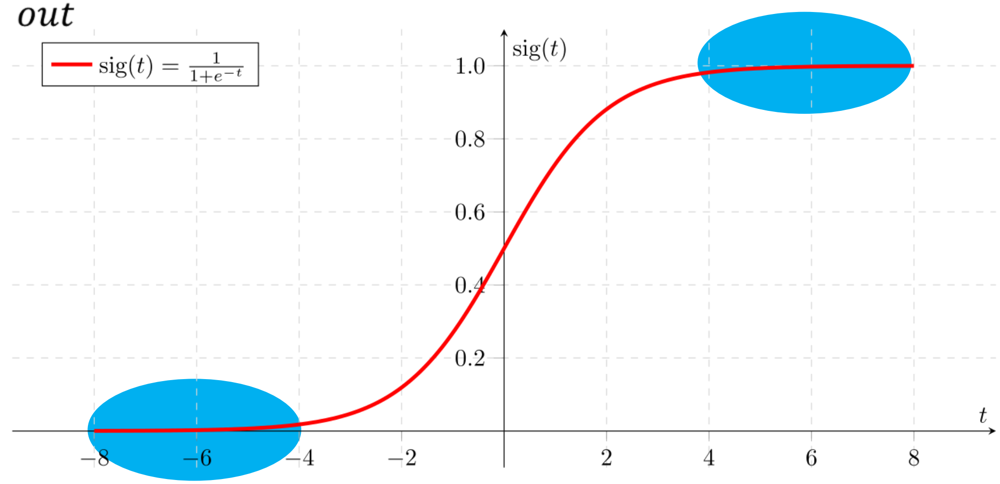
Challenge of Learning

❑ Underfitting

- 원인 : 기울기가 0인 지점 존재



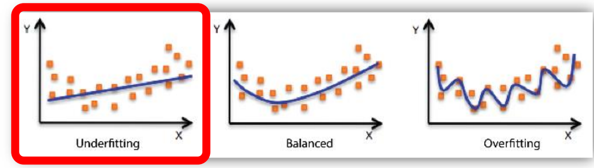


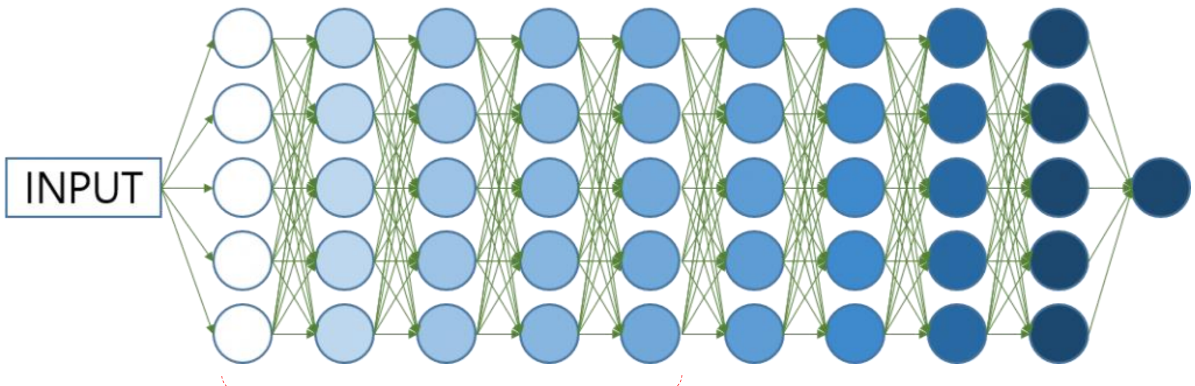


Challenge of Learning

❑ Underfitting

- 기울기 소실 (Vanishing Gradient)

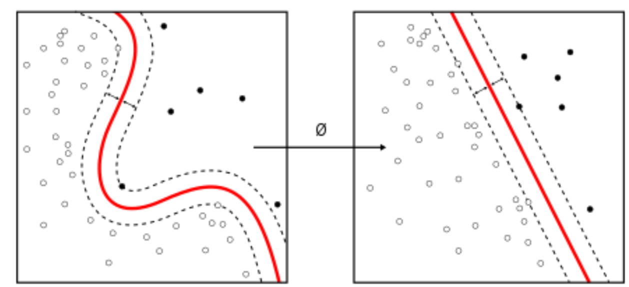




뒤로 갈수록 학습이 잘 안됨

Weight의 업데이트 = 에러 낮추는 방향 X 학습률 X ~~기울기~~

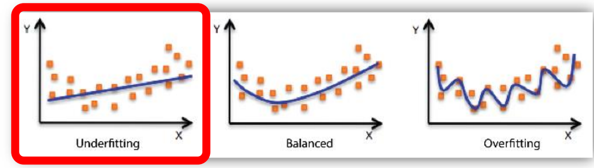
$$-\gamma \nabla F(a^n) \qquad - \qquad \gamma \qquad \nabla F(a^n)$$

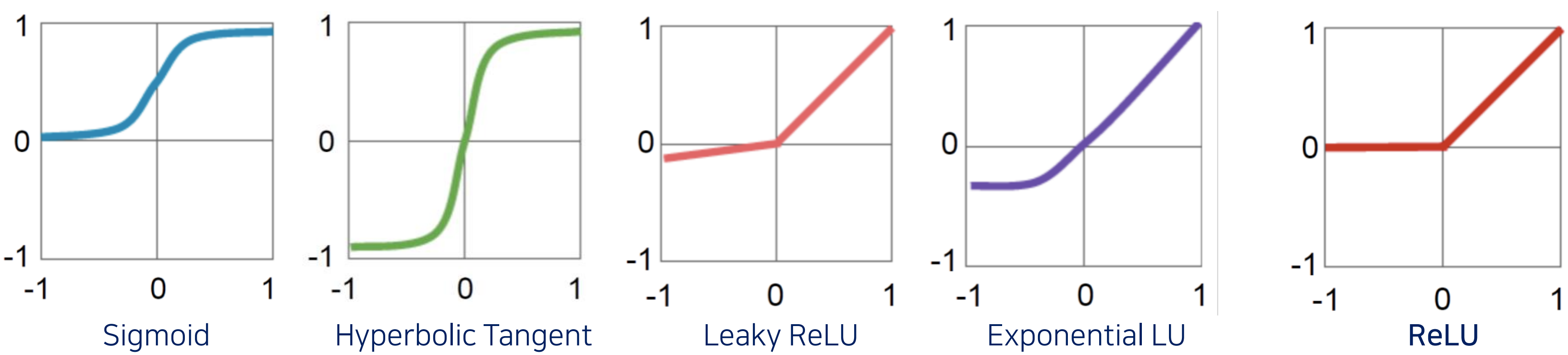


Challenge of Learning

❑ Underfitting

- 해결
  - 활성화 함수의 변경으로 해결
- 다양한 활성화 함수





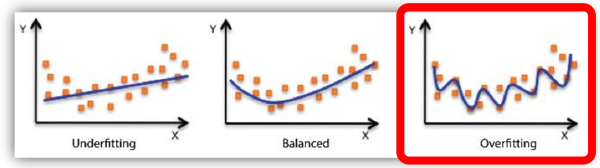
기울기 소실 존재

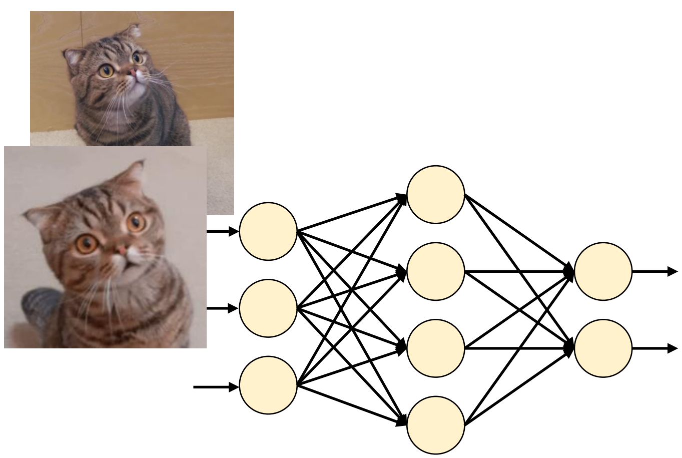
H/W로 만들기 용이

Challenge of Learning

❑ Overfitting

- 원인
  - 매개변수가 많고 표현력이 높은 모델
  - 훈련데이터가 적을 때





뚱뚱해서 고양이 아님

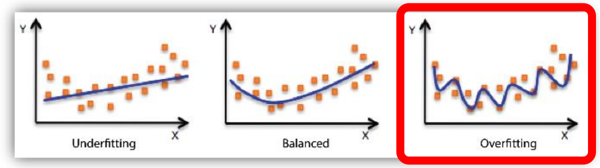
갈색이라 고양이 아님

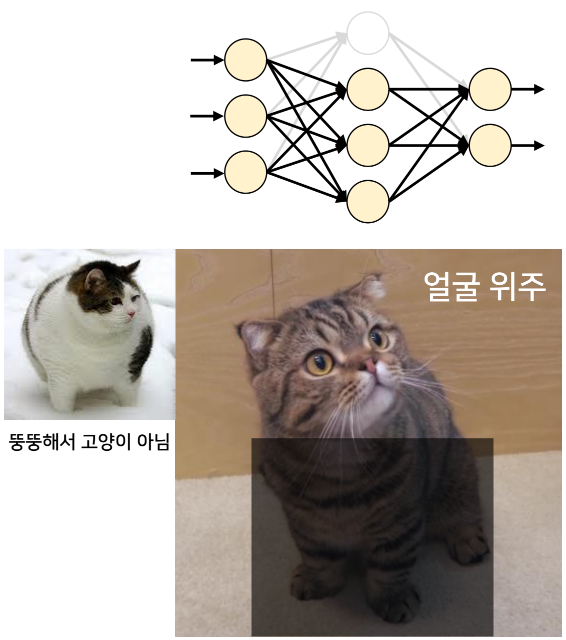
귀쳐져서 고양이 아님

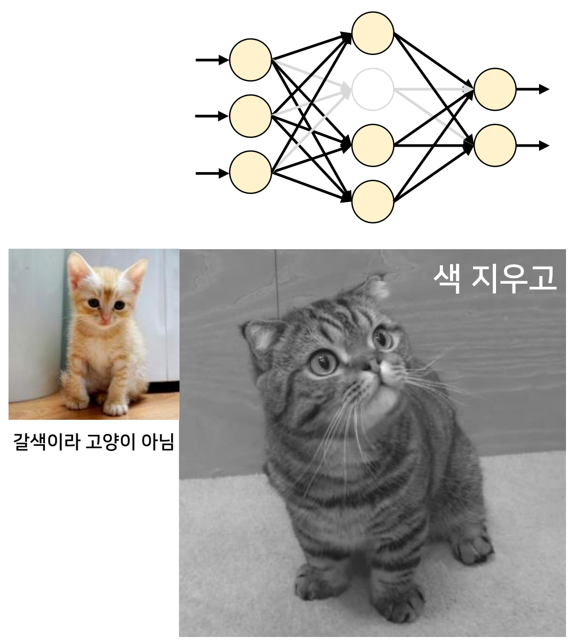
Challenge of Learning

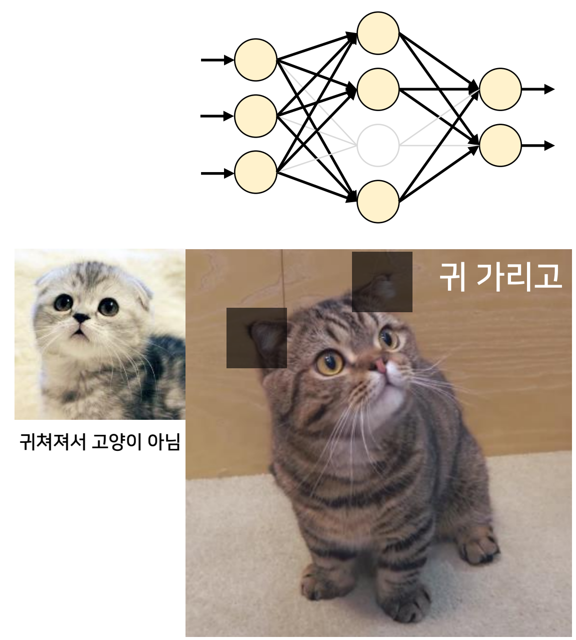
❑ Overfitting

- 해결
  - Dropout



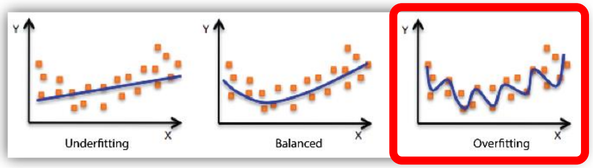




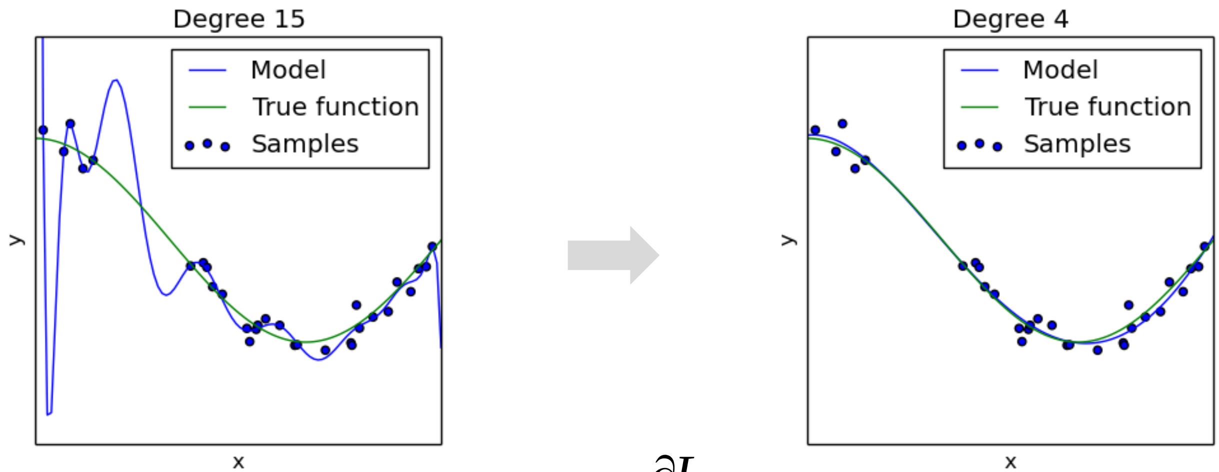


Challenge of Learning

❑ Overfitting



- 해결
  - Weight Decay (Regularization)



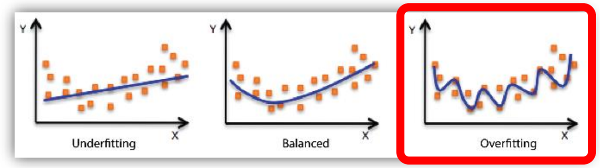
$$W \leftarrow W - \eta\left(\frac{\partial L}{\partial W} + \lambda W\right)$$

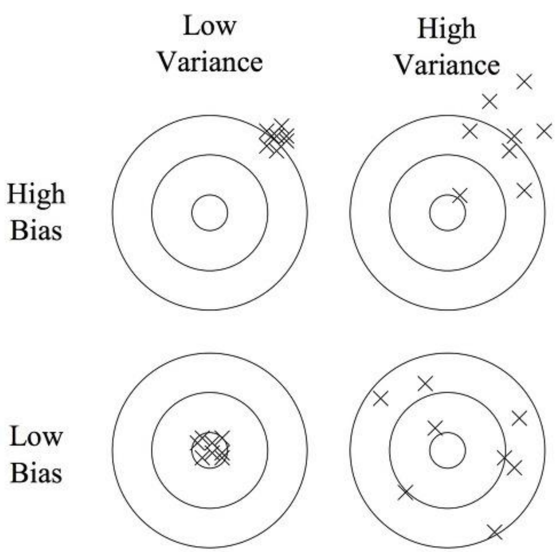
기울기의 급격한 변화를 방지

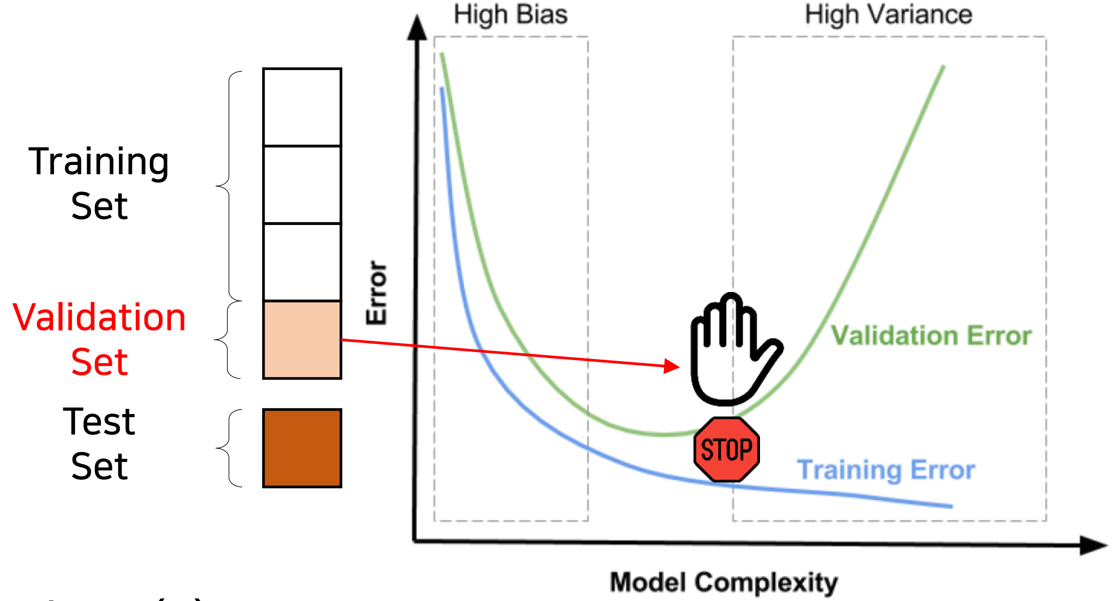
Challenge of Learning

❑ Overfitting

- 해결
  - Validation Set





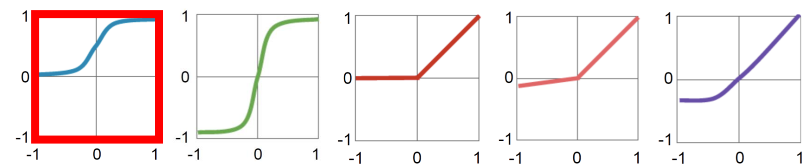


Error(X) = noise(X) + bias(X) + variance(X)

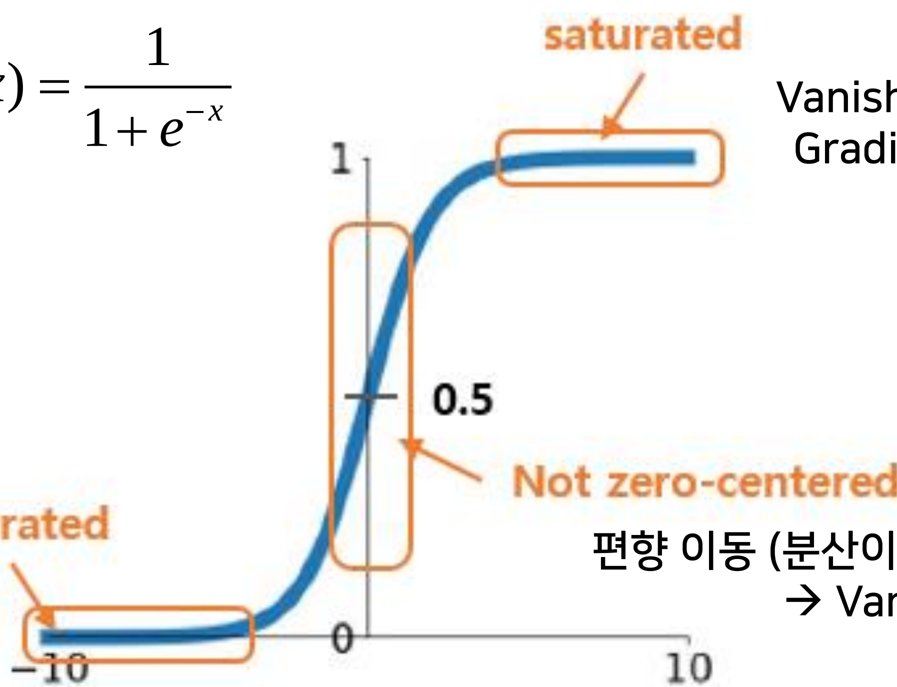
Challenge of Learning

❑ Activation Functions

- Sigmoid



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



Vanishing Gradient

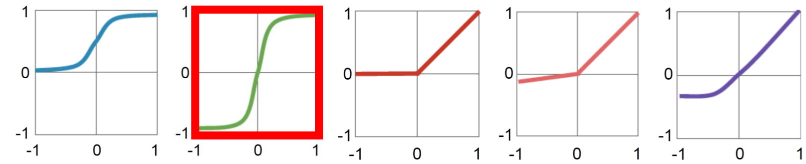
Vanishing Gradient

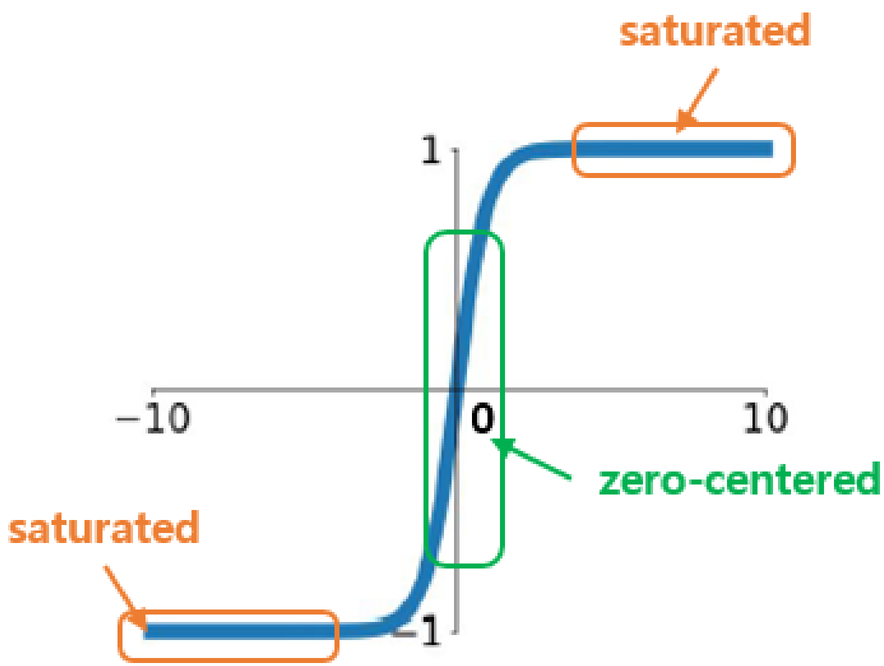
편향 이동 (분산이 계속 커져) → 출력이 0 or 1
→ Vanishing Gradient

Challenge of Learning

❑ Activation Functions

- Hyperbolic Tangent





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

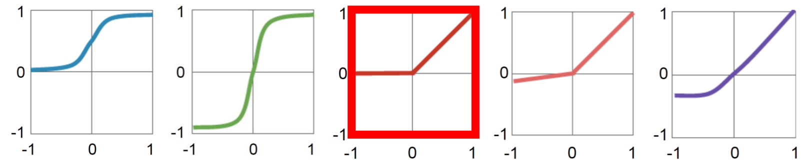
$$= \frac{2}{1+e^{-2x}} - 1$$

$$= 2\sigma(2x) - 1$$

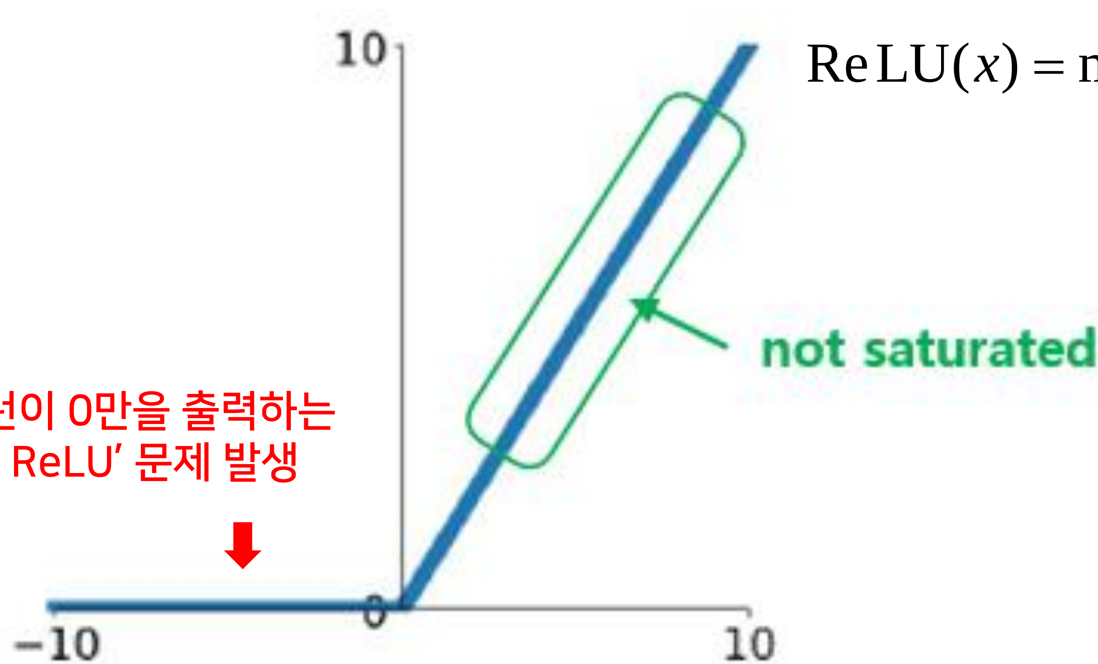
Challenge of Learning

❑ Activation Functions

- Rectified Linear Unit



$$\mathrm{ReLU}(x) = \max(0, x)$$

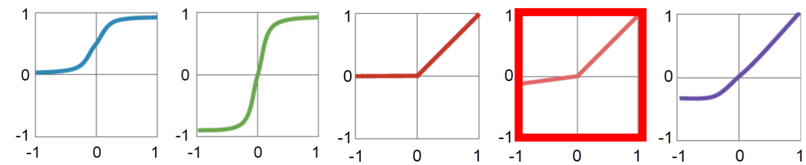


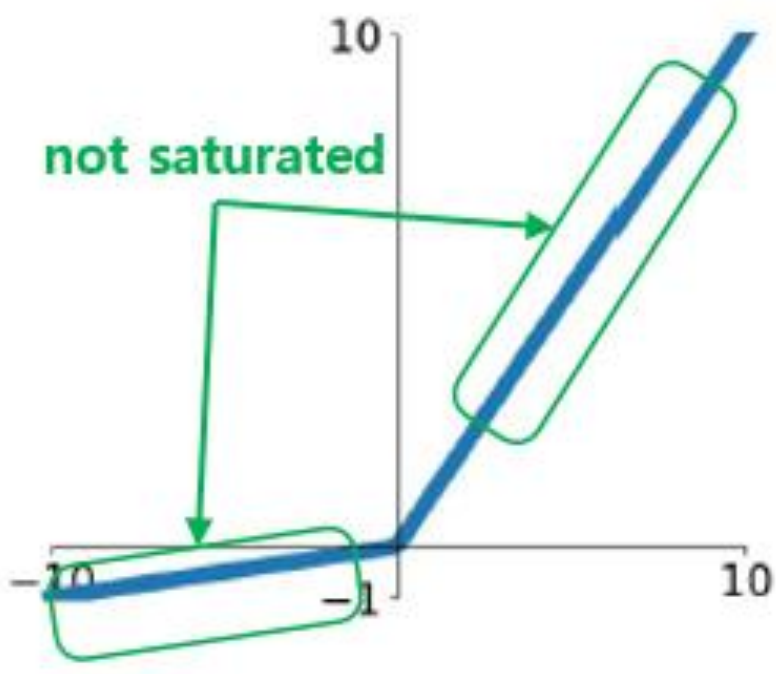
일부 뉴런이 0만을 출력하는
'Dead ReLU' 문제 발생

Challenge of Learning

❑ Activation Functions

- LeakyReLU & PReLU





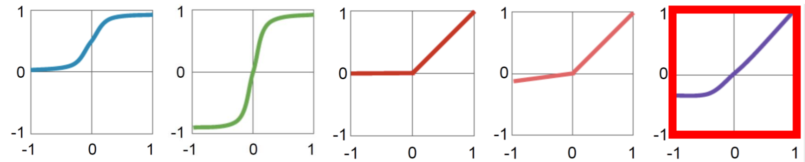
$$\text{Leaky ReLU}_{\alpha}(x) = \max(\alpha x, x)$$

PReLU(Parametric ReLU)는 Leaky ReLU와 식이 동일하지만, LeakyReLU에서 하이퍼파라미터인 α를 가중치 매개변수와 마찬가지로 α의 값도 학습되도록 역전파에 의해 α의 값이 변경되는 함수. PReLU는 대규모 이미지 데이터셋에서는 ReLU보다 성능이 좋았지만, 소규모 데이터셋에는 오버피팅 될 위험

Challenge of Learning

❑ Activation Functions

- Exponential Linear Unit

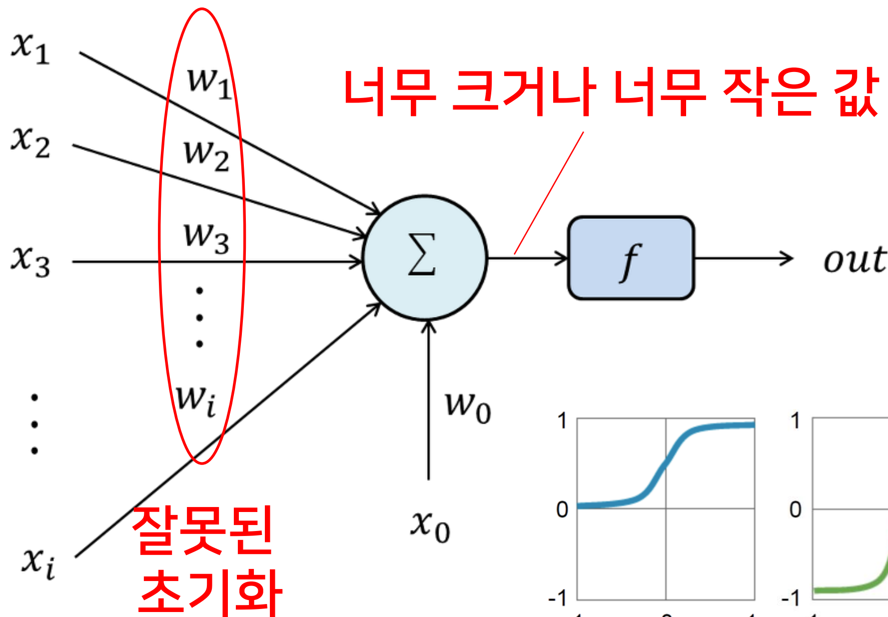


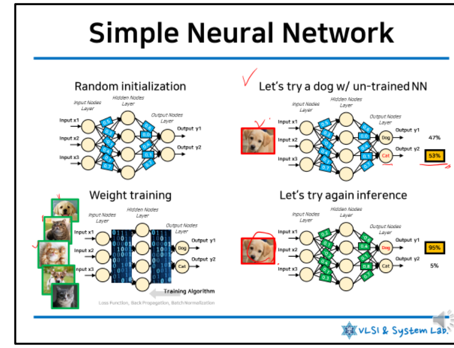
$$ELU_{\alpha}(x) = \begin{cases} \alpha(\exp(x) - 1) & if \;\; x \; < \; 0 \\ x & if \;\; x \; \geq \; 0 \end{cases}$$

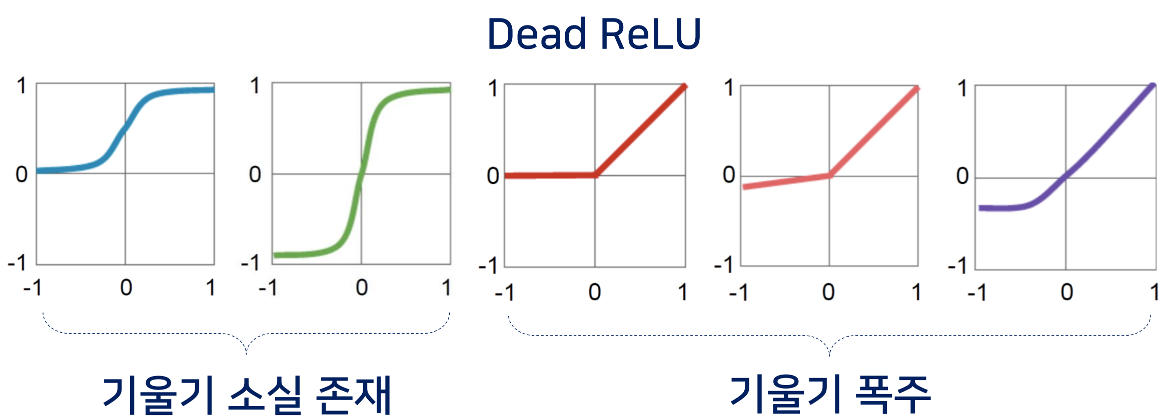
- x < 0 일 때 ELU 활성화 함수 출력의 평균이 0에 가까워지기 때문에 편향 이동이 감소하여 그래디언트 소실 문제를 줄여준다. 하이퍼파라미터인 α는 x가 음수일 때 ELU가 수렴할 값을 정의하며 보통 1로 설정
- x < 0 이어도 그래디언티가 0이 아니므로 죽은(dead) 뉴런을 만들지 않는다.
- α = 1일 때 ELU는 x = 0 에서 급격하게 변하지 않고 모든 구간에서 매끄럽게 변하기 때문에 경사하강법에서 수렴속도가 빠르다.

Challenge of Learning

❑ 가중치 초기화 (Weight Initialization)

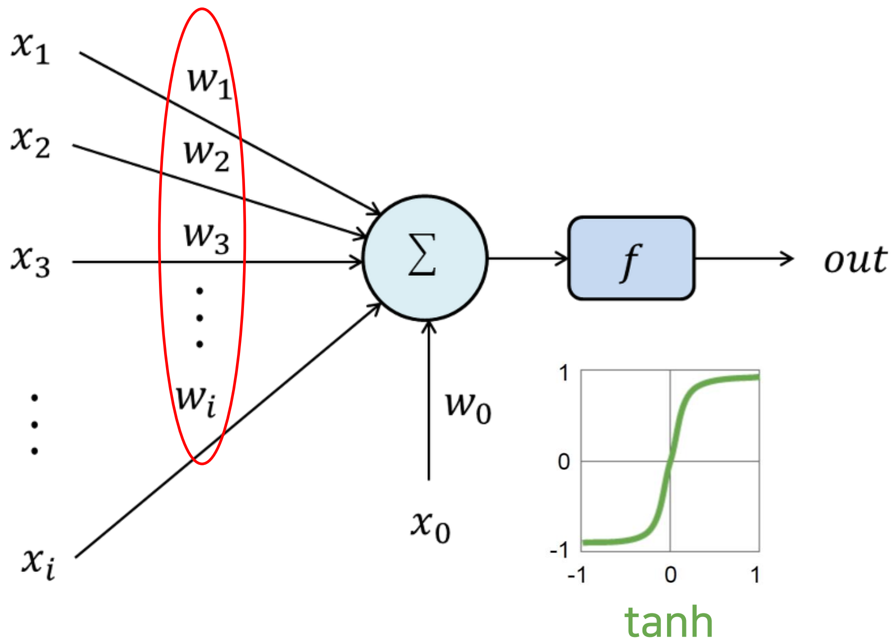






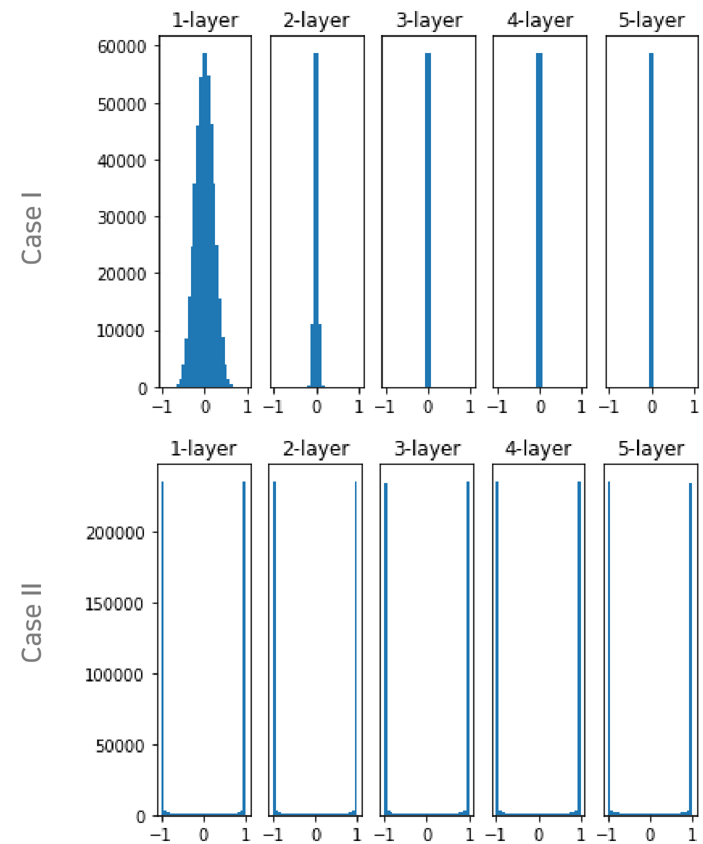
Challenge of Learning

❑ 가중치 초기화 (Weight Initialization)



Case I : (μ, σ) = (0, 0.01)

Case II : (μ, σ) = (0, 1)



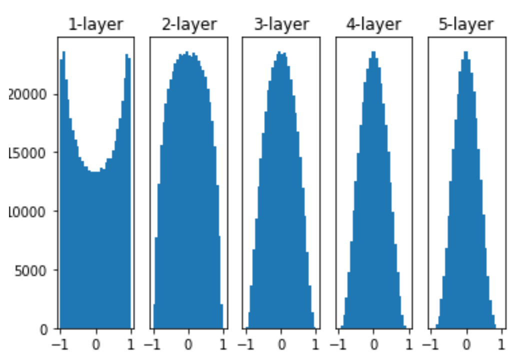
VLSI & System Lab.

Challenge of Learning

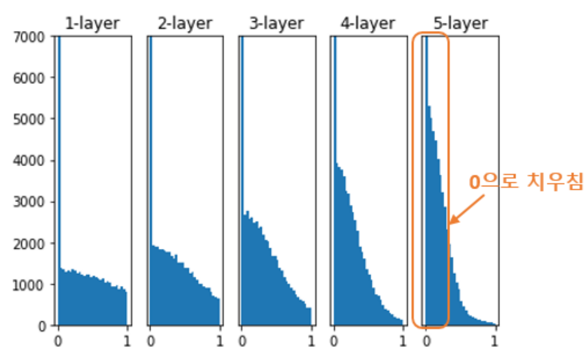
❑ 가중치 초기화 (Weight Initialization)

- Xavier 초기화 (tf.contrib.Xavier_initializer)
  - 신경망 깊이가 문제 (기울기 소실/폭주)를 일으킴 ? → 너무 크지도 너무 작지도 않게 레이어 수로 나누자!
- He 초기화 (tf.keras.initializers.he_xxx) : ReLU에서도 잘 되게 해보자

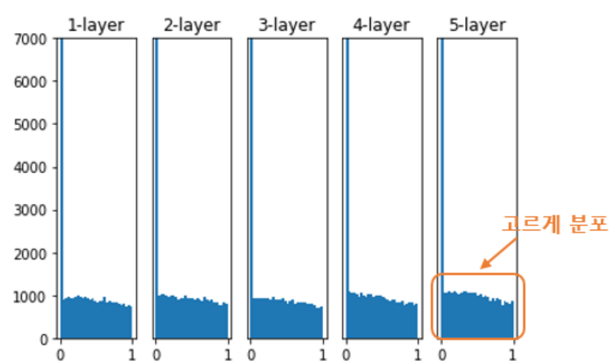
tanh + Xavier



ReLU + Xavier



ReLU + He

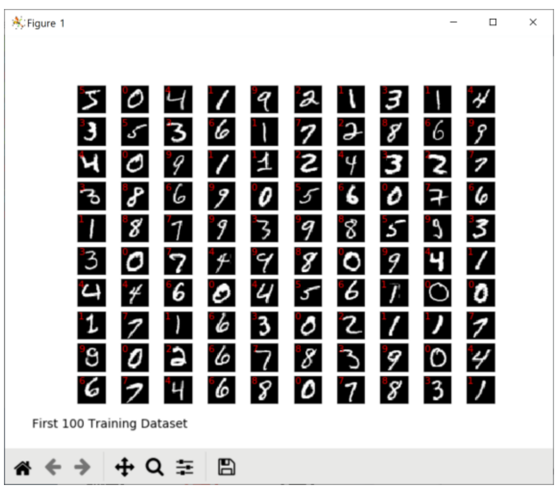


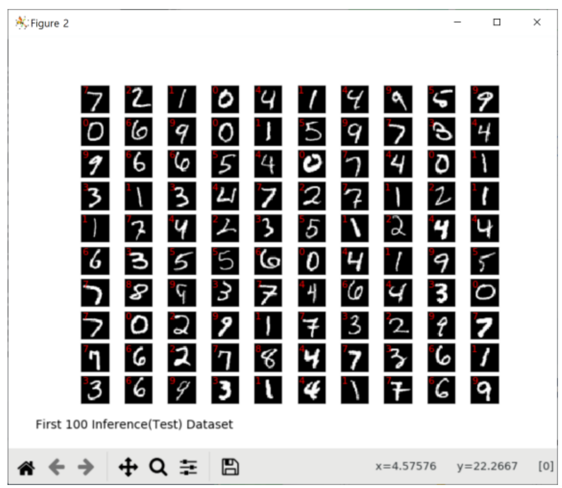
VLSI & System Lab.

Optimizer of Learning

❑ MNIST Dataset (손글씨-숫자 인식)

파일	목적
train-images-idx3-ubyte.gz	학습 셋 이미지 - 55000개의 트레이닝 이미지, 5000개의 검증 이미지
train-labels-idx1-ubyte.gz	이미지와 매칭되는 학습 셋 레이블
t10k-images-idx3-ubyte.gz	테스트 셋 이미지 - 10000개의 이미지
t10k-labels-idx1-ubyte.gz	이미지와 매칭되는 테스트 셋 레이블

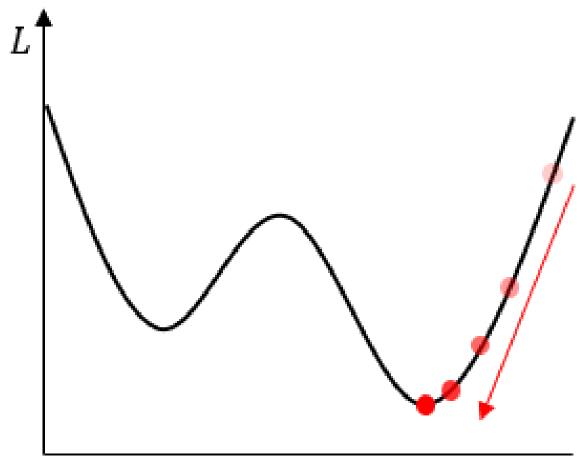




Optimizer of Learning

❑ MNIST Dataset (손글씨-숫자 인식)

파일	목적
train-images-idx3-ubyte.gz	학습 셋 이미지 - 55000개의 트레이닝 이미지, 5000개의 검증 이미지
train-labels-idx1-ubyte.gz	이미지와 매칭되는 학습 셋 레이블
t10k-images-idx3-ubyte.gz	테스트 셋 이미지 - 10000개의 이미지
t10k-labels-idx1-ubyte.gz	이미지와 매칭되는 테스트 셋 레이블

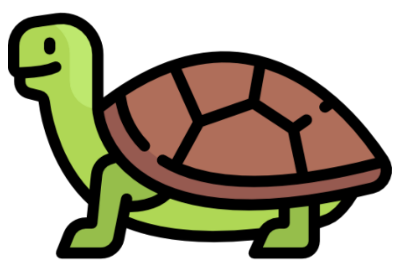


Normal Gradient Descent 기반 학습

55000개 데이터를 넣고 한 Step

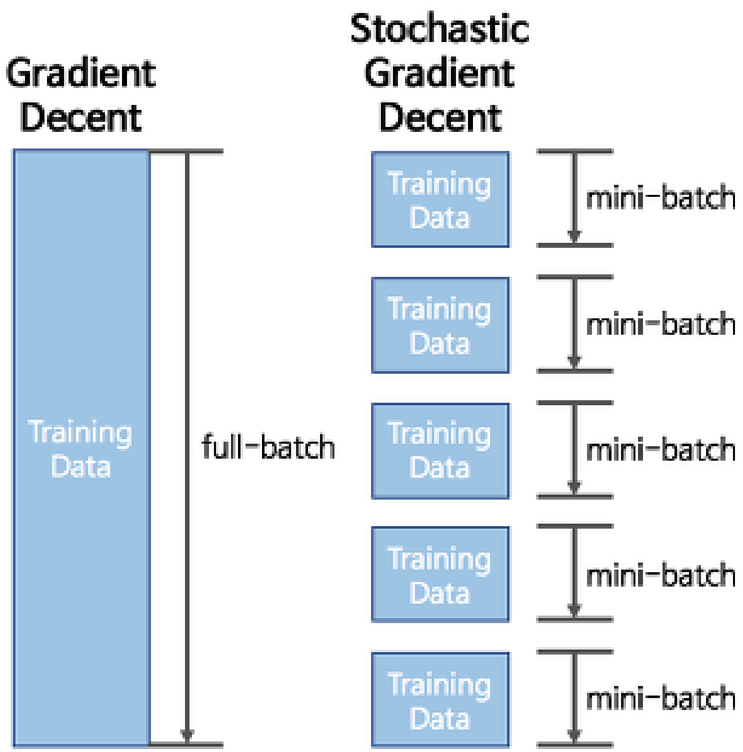
55000개 데이터를 넣고 한 Step

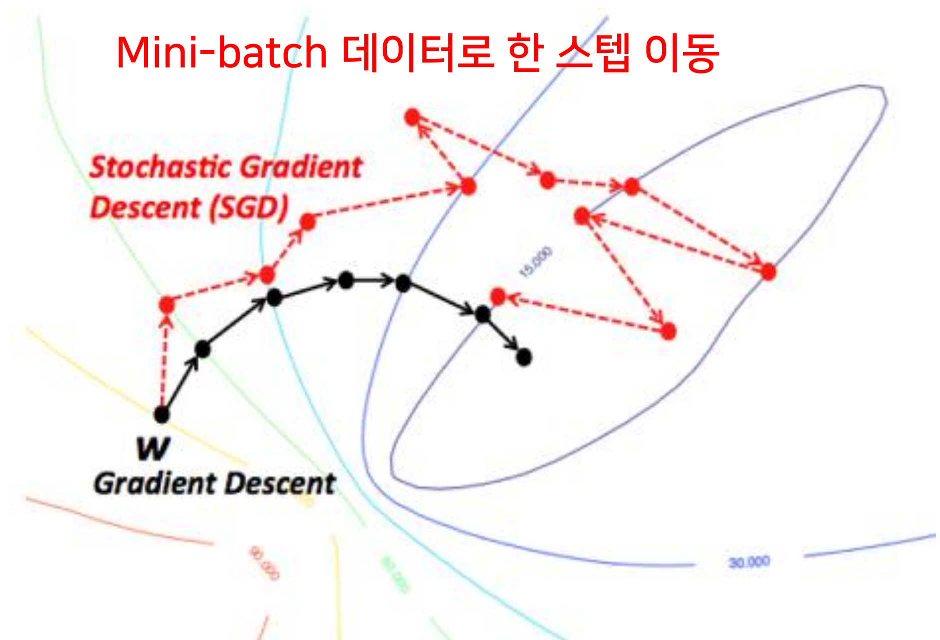
55000개 데이터를 넣고 한 Step



Optimizer of Learning

❑ Stochastic Gradient Descent (SGD)





Optimizer of Learning

❑ More than SGD (Stochastic Gradient Descent)

- SGD
  - 경사하강법은 무작정 기울어진 방향으로 이동하는 방식이기 때문에 탐색경로가 비효율적이어서 한참을 탐색
- 다른 Optimizer들
  - Momentum / Nesterov Accelerated Gradient (NAG)
  - Adaptive Gradient (Adagrad)
  - RMSProp
  - Adaptive Delta (AdaDelta)
  - Adaptive Moment Estimation (Adam)

Optimizer of Learning

❑ Learning Notation

- $\theta = \theta - \eta \nabla_{\theta} J(\theta)$

θ : Parameter η : Learning Rate $J(\theta)$: Loss Function

$\nabla_{\theta} J(\theta)$: Gradient of Loss

❑ Momentum

- Gradient Descent를 통해 이동하는 과정에 일종의 '관성'을 주는 것
- Time step t 에서의 이동벡터

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$

$$= \eta \nabla_{\theta} J(\theta)_t + \gamma \eta \nabla_{\theta} J(\theta)_{t-1} + \gamma^2 \eta \nabla_{\theta} J(\theta)_{t-2} + \cdots$$

얼마나 momentum을 줄 것인지
에 대한 momentum term으로,
보통 0.9 정도의 값을 사용

Optimizer of Learning

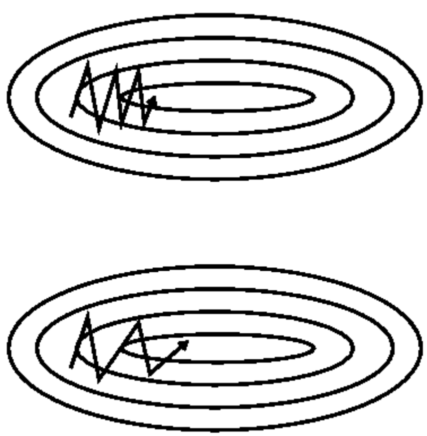
❑ Learning Notation

- $\theta = \theta - \eta \nabla_{\theta} J(\theta)$

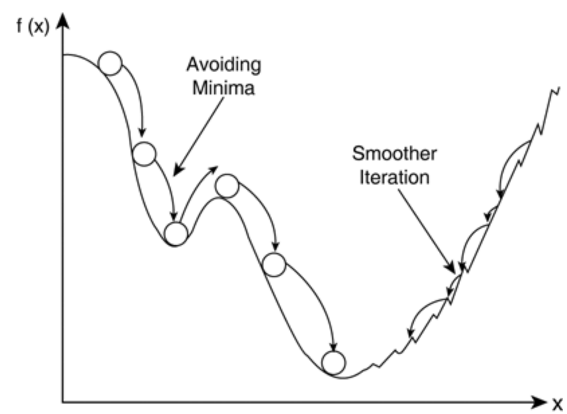
θ : Parameter η : Learning Rate $J(\theta)$: Loss Function

$\nabla_{\theta} J(\theta)$: Gradient of Loss

❑ Momentum



방향을 좀 더 잘 찾고

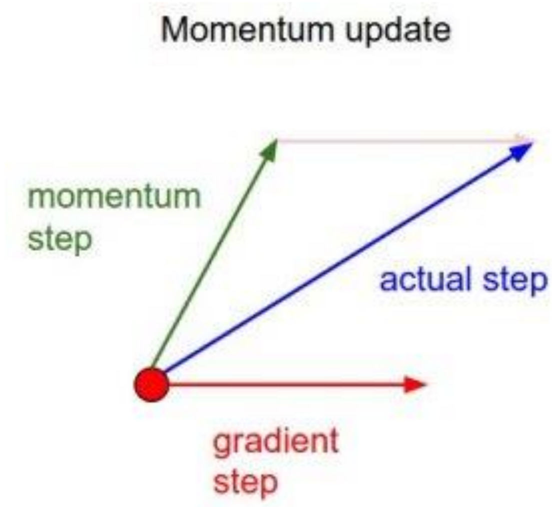


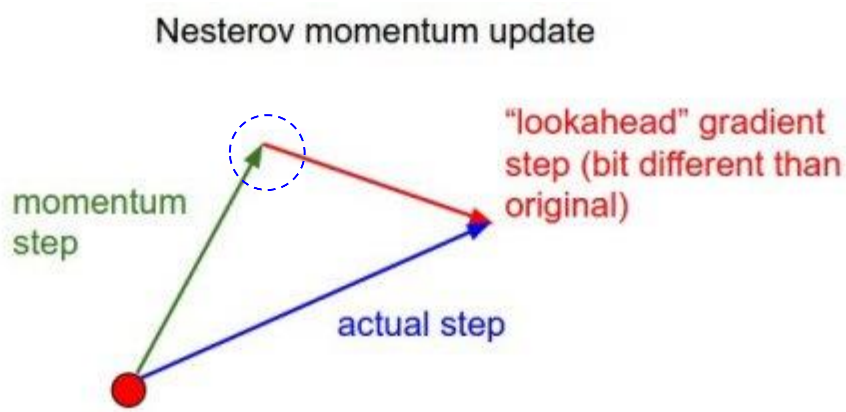
Local Minima에서 탈출 가능

But,
과거에 이동했던 양을 변수별로 저장해야하므로 변수에 대한 메모리가 기존의 두배

Optimizer of Learning

❑ Nesterov Accelerated Gradient (NAG)





$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

현재 위치에서의 기울기와 모멘텀 스텝을 독립적으로 계산하고 합침

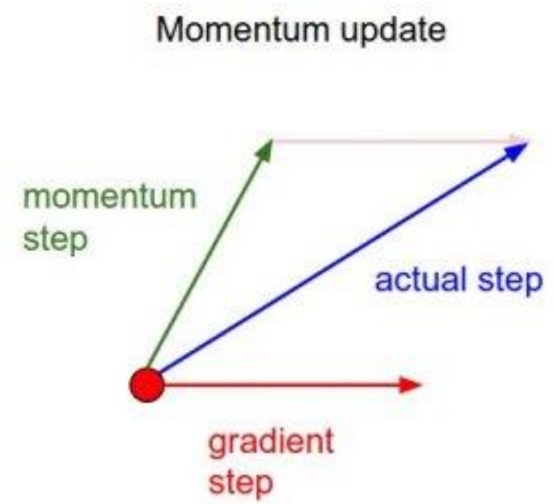
$$\theta = \theta - v_t$$

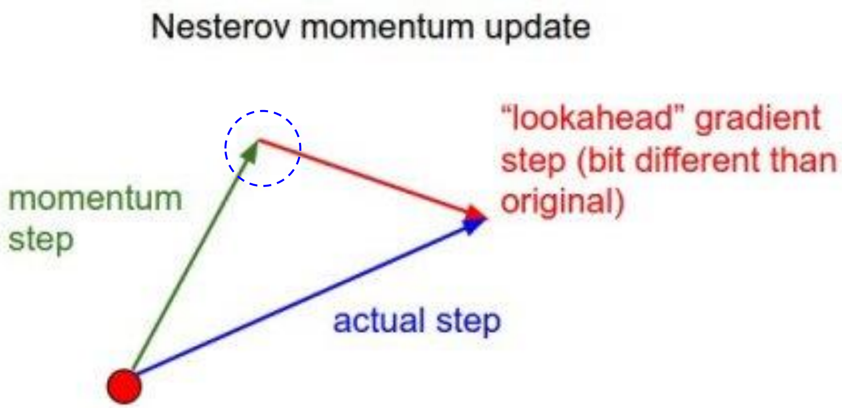
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

모멘텀 스텝을 먼저 이동했다고 생각한 후 그 자리에서의 기울기를 구해서 기울기 스텝을 이동

Optimizer of Learning

❑ Nesterov Accelerated Gradient (NAG)





멈춰야 할 시점에서도 관성에
의해 훨씬 멀리 갈수도 있음

일단 모멘텀으로 이동을 반정도 한 후
어떤 방식으로 이동해야 할 지를 결정

(모멘텀 방식의 빠른 이동 + 적절한 시점에서 제동)

Optimizer of Learning

❑ Adaptive Gradient (Adagrad)

- 자주 등장하거나 변화를 많이 한 변수들의 경우 최적점에 가까이 있을 확률이 높음

'지금까지 많이 변화했던 변수들은 step size를 작게 하고,
지금까지 많이 변화하지 않은 변수들은 step size를 크게 하자'

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

보통 adagrad에서 step size로는 0.01

ϵ: 0 나누기 방지 (10^{-4}~10^{-8})

$$G_t = G_{t-1} + [\nabla_\theta J(\theta_t)]^2$$

Neural Network의 parameter가 k개라고 할 때, G_t는 k 차원의 벡터

$*element - wise$

Optimizer of Learning

❑ Adaptive Gradient (Adagrad)

- 자주 등장하거나 변화를 많이 한 변수들의 경우 최적점에 가까이 있을 확률이 높음

‘지금까지 많이 변화했던 변수들은 step size를 작게 하고,
지금까지 많이 변화하지 않은 변수들은 step size를 크게 하자’

- 장점 : 학습 속도 담금질 (Step Size Decay) 필요 없어짐
- 단점 : 학습을 계속하면 Step Size가 너무 줄어 안 움직임
  - 개선한 알고리즘 : RMSProp & AdaDelta

Optimizer of Learning

❑ RMSProp (제프리 힌튼 교수가 제안)

Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$G_t = G_{t-1} + [\nabla_\theta J(\theta_t)]^2$$

$*element - wise$

RMSProp

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$G_t = \gamma G_{t-1} + (1-\gamma)[\nabla_\theta J(\theta_t)]^2$$

$*element - wise$

G_t가 무한정 커지지 않으면서,
최근 변화량의 상대적 크기 차이 유지

Optimizer of Learning

❑ Adaptive Moment Estimation (Adam)

RMSProp + Momentum

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$G_t = \gamma G_{t-1} + (1-\gamma)[\nabla_\theta J(\theta_t)]^2$$

$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

Adam

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

$$m_t = \beta_1 m_{t-1} - (1-\beta_1)\nabla_\theta J(\theta)$$

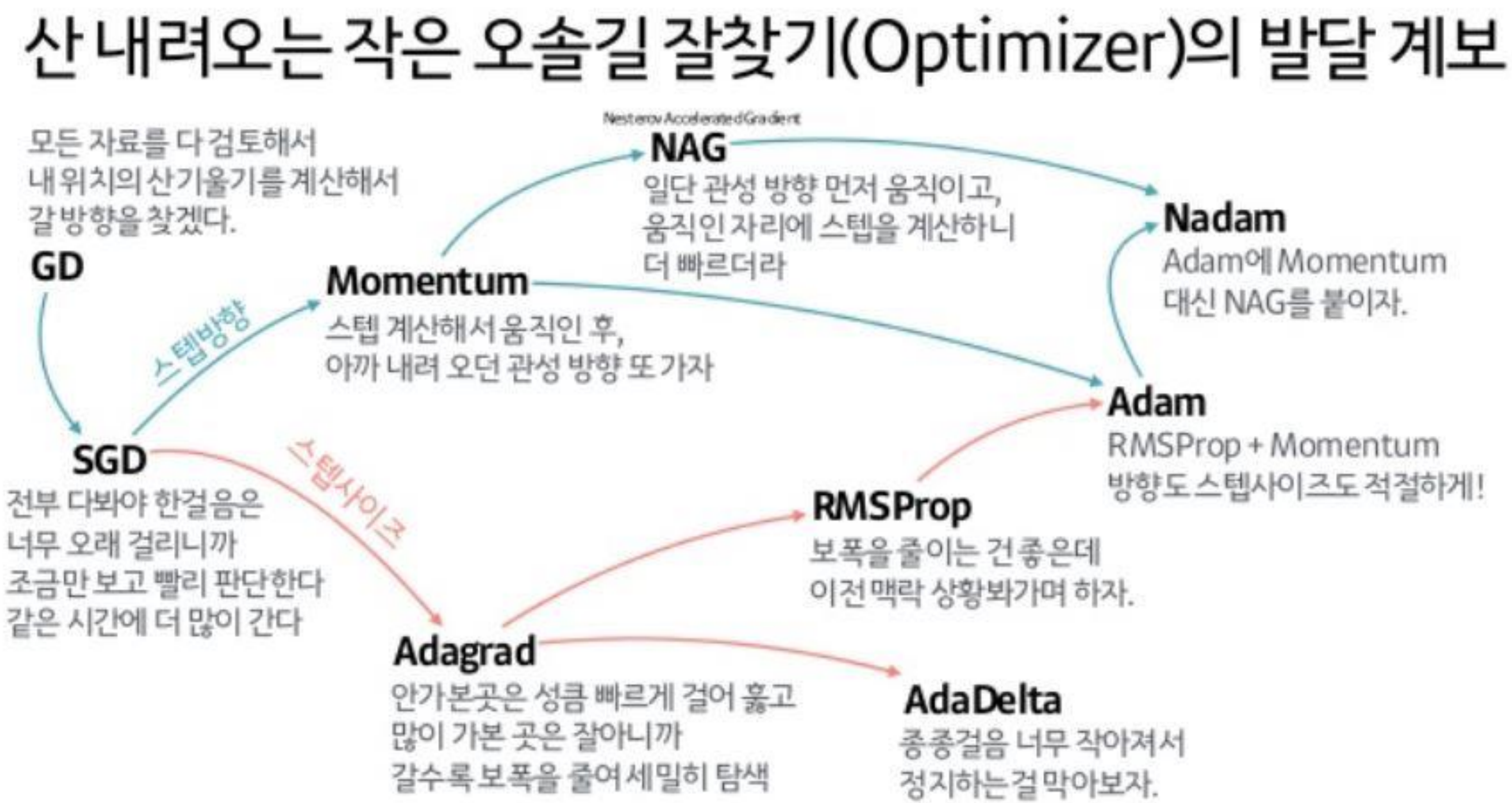
$$v_t = \beta_2 v_{t-1} - (1-\beta_2)[\nabla_\theta J(\theta)]^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

보통 β_1 로는 0.9, β_2로는 0.999, ϵ으로는 10^{-8} 정도의 값을 사용

Adam에서는 m과 v가 처음에 0으로 초기화되어 있기 때문에 학습의 초반부에서는 m_t,v_t가 0에 가깝게 bias 되어있을 것이라고 판단하여 이를 unbiased 하게 만들어주는 작업을 거침

Optimizer of Learning



Outline

- ❑ **인공지능 관련 동향**
- ❑ **딥러닝의 기초**
  - 용어 및 분류(Classification)/회기(Regression) 모델
  - 퍼셉트론 (Perceptron) 및 MNIST Example
- ❑ **학습 및 추론 알고리즘의 이해**
  - Backpropagation
  - Challenge of Learning
  - Optimizer of Learning
- ❑ **딥러닝 가속기**
  - CNN 및 인공 신경망 가속기 연구 동향