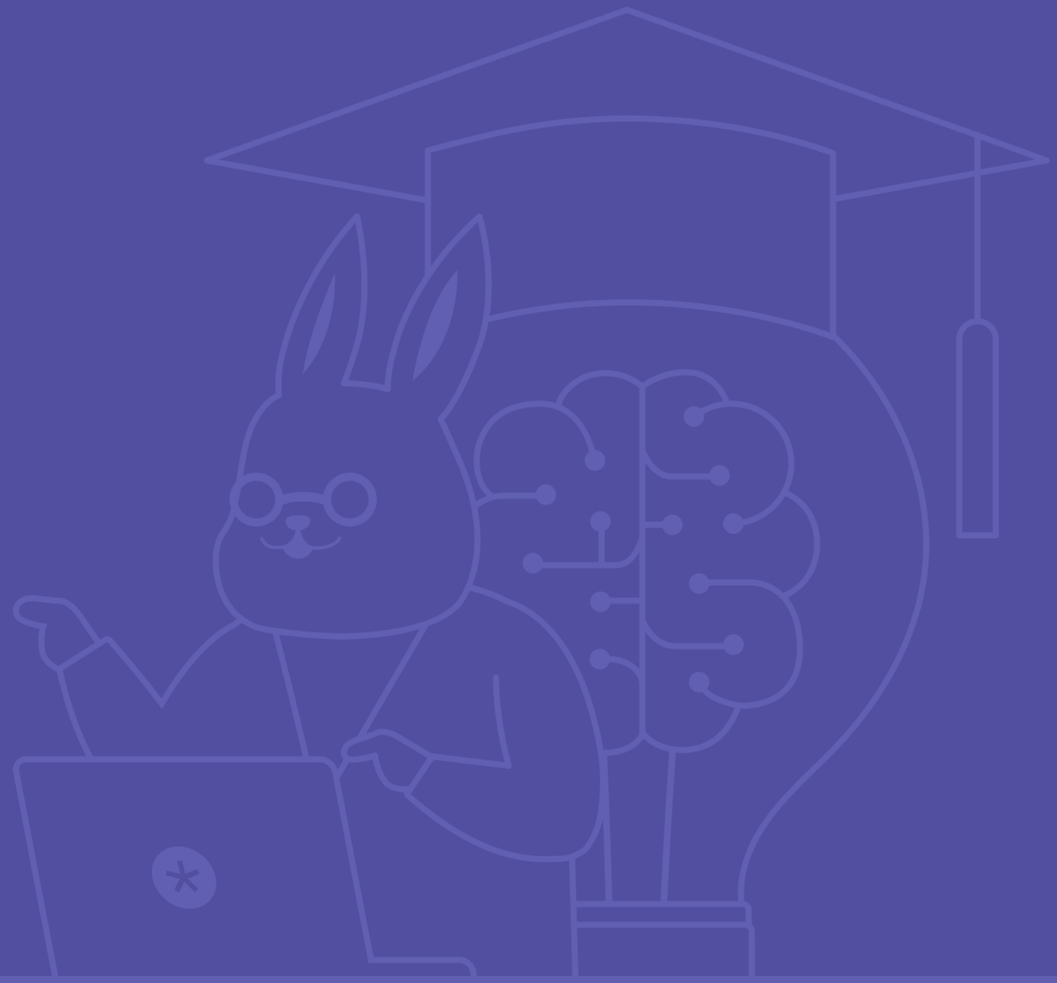
2020 AI College

11장 딥러닝의 개념 (2)

정민수 강사



/* elice */

Contents

Target

인공신경망에 대하여 이해한다.

인공신경망이 무엇인지 이해한다.

모델 학습에 대하여 이해한다.

모델을 학습에 필요한 손실함수, 학습 방법에 대하여 이해하고 Vanishing Gradient 현상에 대하여 이해한다.

과적합 방지 기술에 대하여 살펴본다.

과적합을 방지하기 위한 Regularization, Dropout 등의 기술에 대하여 살펴본다.

Data Augmentation에 대하여 살펴본다.

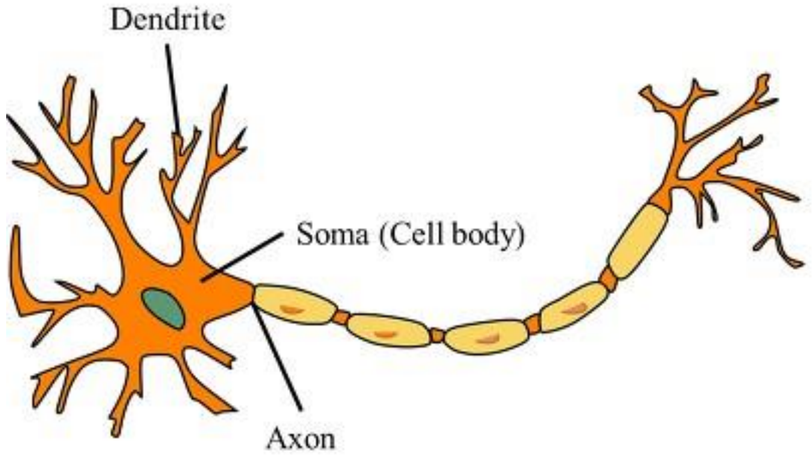
Data augmentation이 왜 필요하고 어떻게 하는 것인지에 대하여 살펴본다.

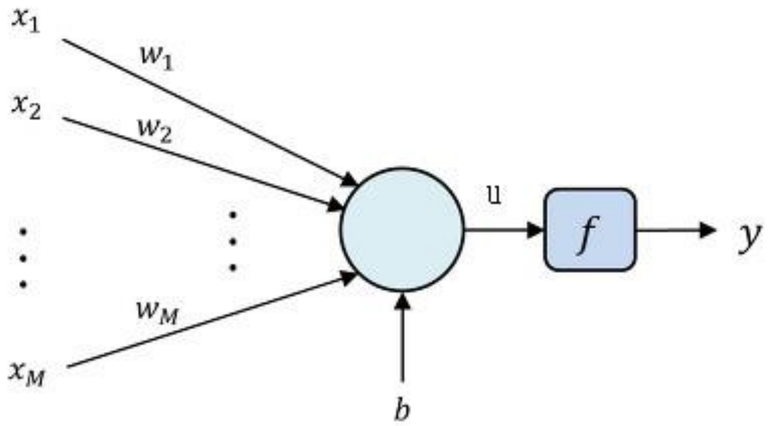
01

인공신경망

Artificial Neural Network (ANN)

- 뇌는 신경세포(neuron)와 신경세포를 연결하는 시냅스(synapse)를 통해서 신호를 주고 받음으로써 정보를 저장하고 학습
- 인공신경망(Artificial Neural Network)
  - 뇌의 학습방법을 수학적으로 모델링한 기계학습 알고리즘





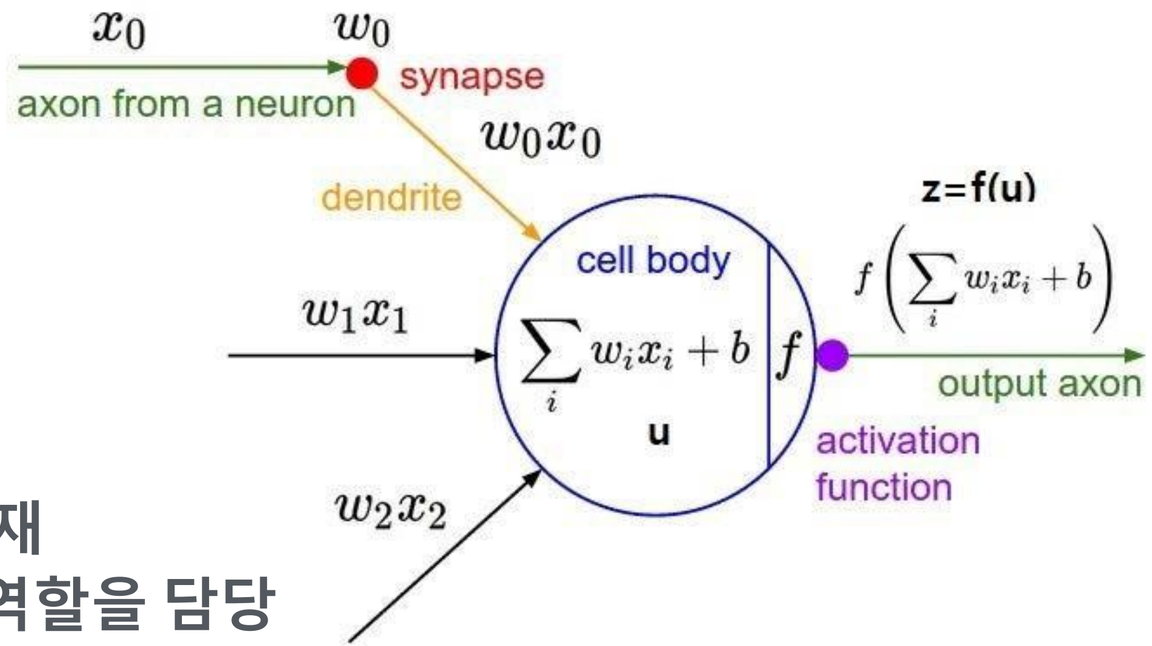
$$u = \sum_{i=1}^{M} w_i x_i + b$$

$$y = f(u)$$

/* elice */

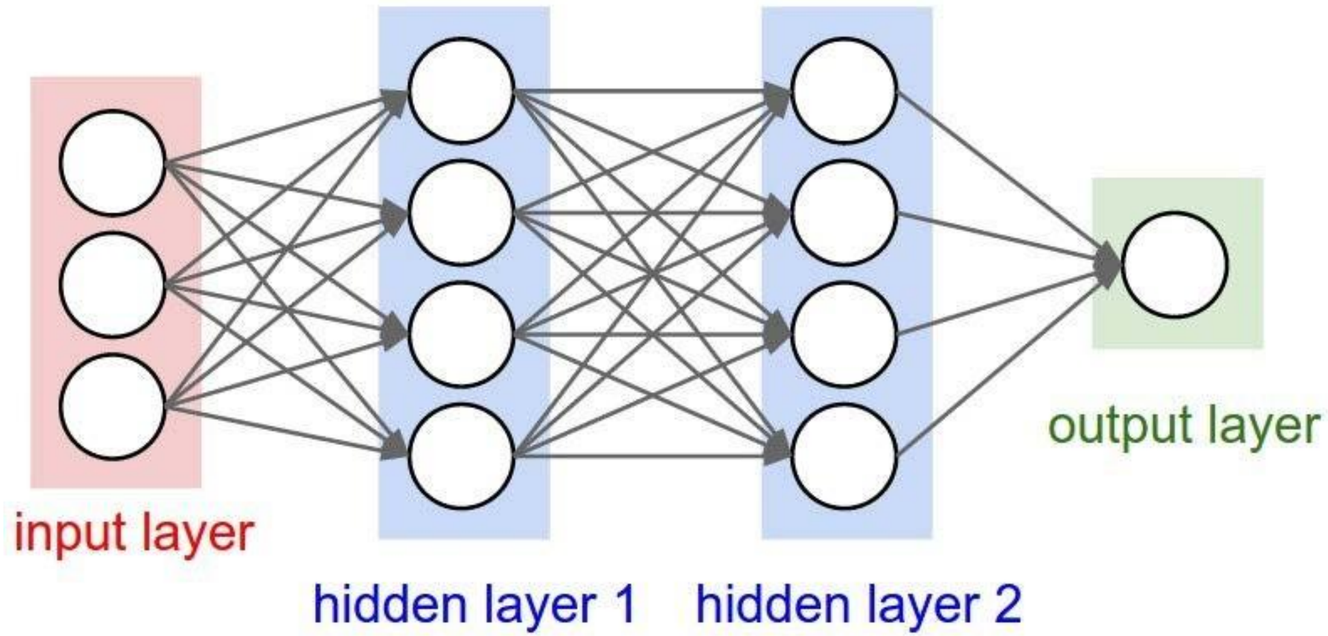
Artificial Neural Network (ANN)

- 기본용어
  - xi: 입력(Input)
  - wi: 가중치(Weight)
  - b: 편향(Bias)
  - f: 활성화(Activation) 함수
  - u: 선형결합(Net)
  - z: 출력(Output)
- 뉴런에는 선형 결합과 활성화 함수 기능이 존재
- 시냅스는 뉴런과 뉴런을 연결해주는 가중치 역할을 담당

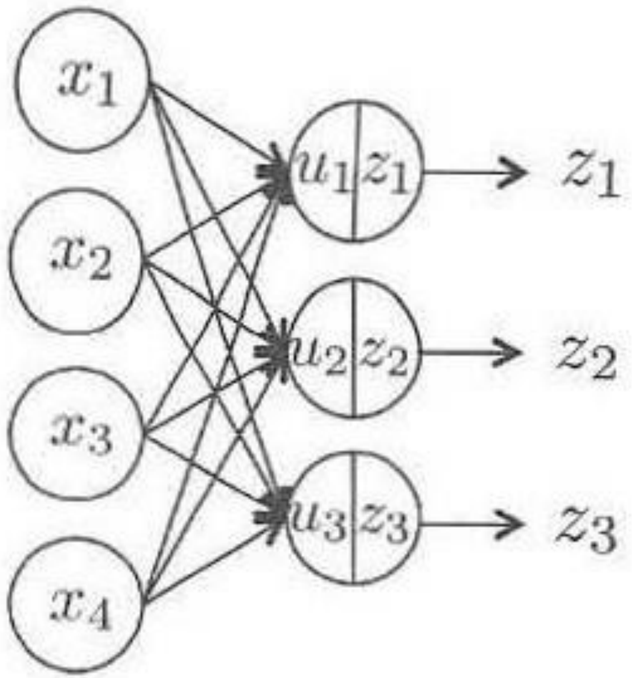


Artificial Neural Network (ANN)

- 인공신경망은 입력층, 히든층, 출력층으로 구성
- 각 뉴런의 출력은 직접 전달되는 정보에만 의존할 뿐, 다른 정보들과는 무관
- 이 때문에 병렬처리가 가능하므로 연산속도가 매우 빠름



예시 - 1개의 레이어로 이루어진 MLP



$$u_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + b_1$$

$$u_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{24}x_4 + b_2$$

$$u_3 = w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + w_{34}x_4 + b_3$$

예시 - 1개의 레이어로 이루어진 MLP

- 벡터 및 행렬로 표기

$$\mathbf{u} = \begin{bmatrix} u_1 \\ \vdots \\ u_J \end{bmatrix}, \ \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix}, \ \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_J \end{bmatrix}, \ \mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_J \end{bmatrix}$$

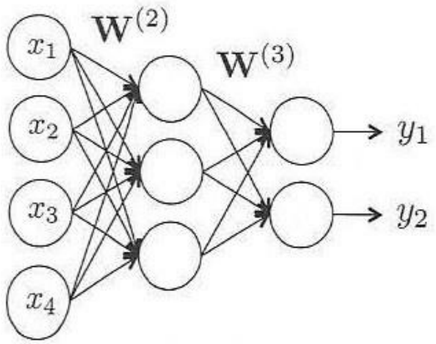
$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1I} \\ \vdots & \ddots & \vdots \\ w_{JI} & \cdots & w_{JI} \end{bmatrix}, \ \mathbf{f}(\mathbf{u}) = \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_J) \end{bmatrix}$$

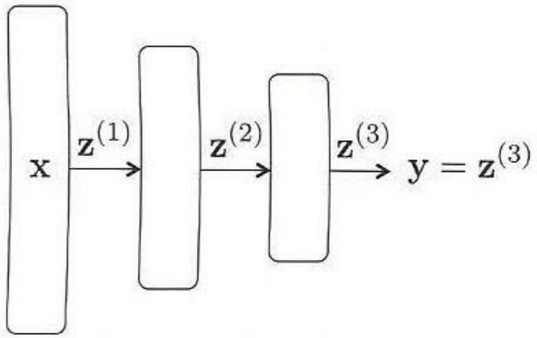
- 간단히 표현 가능

$$\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$\mathbf{z} = \mathbf{f}(\mathbf{u})$$

예시 - 2개의 레이어로 이루어진 MLP





$$\mathbf{u}^{(2)} = \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}$$

$$\mathbf{z}^{(2)} = \mathbf{f}(\mathbf{u}^{(2)})$$

$$\mathbf{u}^{(3)} = \mathbf{W}^{(3)}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

$$\mathbf{z}^{(3)} = \mathbf{f}(\mathbf{u}^{(3)})$$

- 임의의 개수의 레이어로 이루어진 신경망에서는 $\mathbf{z}^{(1)} = \mathbf{x}$라 두면

$$\mathbf{u}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{u}^{(l+1)})$$

손실함수(Loss function)

- 신경망에서 내놓는 결과값과 실제 결과값 사이의 차이를 정의하는 함수
- 신경망 학습의 목표
  - 손실함수를 최소화하는 것
  - 이를 위해 SGD 등의 학습 알고리즘 사용

손실함수(Loss function)

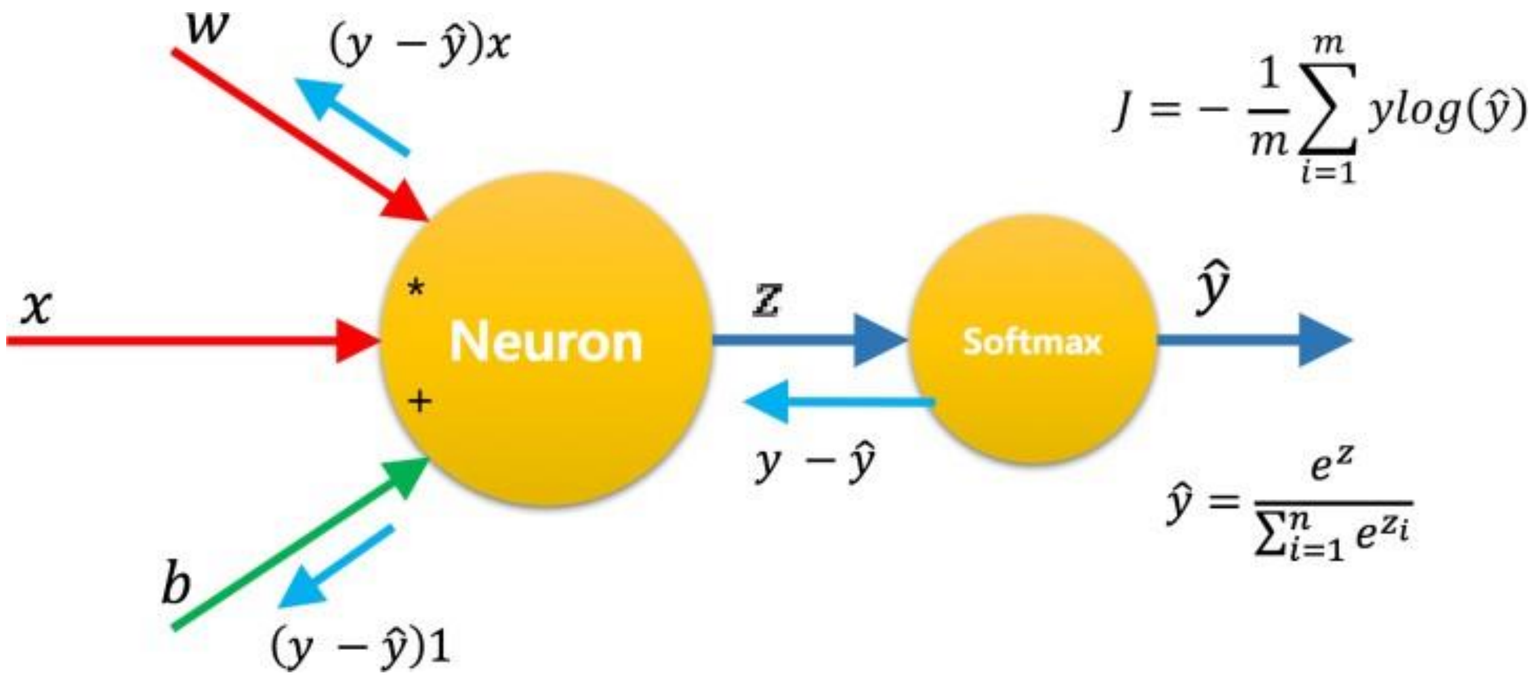
- 신경망의 최종출력을 $\mathbf{y} = \mathbf{z}^{(L)}$로 표기
- Train set을 $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{d}_1), (\mathbf{x}_2, \mathbf{d}_2), \cdots, (\mathbf{x}_N, \mathbf{d}_N)\}$라 하자

- 회귀(Regression)
  - 손실함수로 제곱오차(Mean-squared error)를 사용

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} E_n(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} ||\mathbf{d}_n - \mathbf{y}(\mathbf{x}_n; \mathbf{w})||^2$$

손실함수(Loss function)

- 분류(Classification)
  - 활성화 함수로 소프트맥스(Softmax) 함수 사용
  - 손실함수로 크로스 엔트로피(Cross entropy) 사용



02

모델 학습

딥러닝 모델 학습 과정

- 모델의 Loss function을 정의
- Loss function을 최소화하기 위해 최적화 알고리즘을 적용
- 최적화 알고리즘을 통해 Loss function을 최소화하는 parameter를 찾음

Gradient Descent

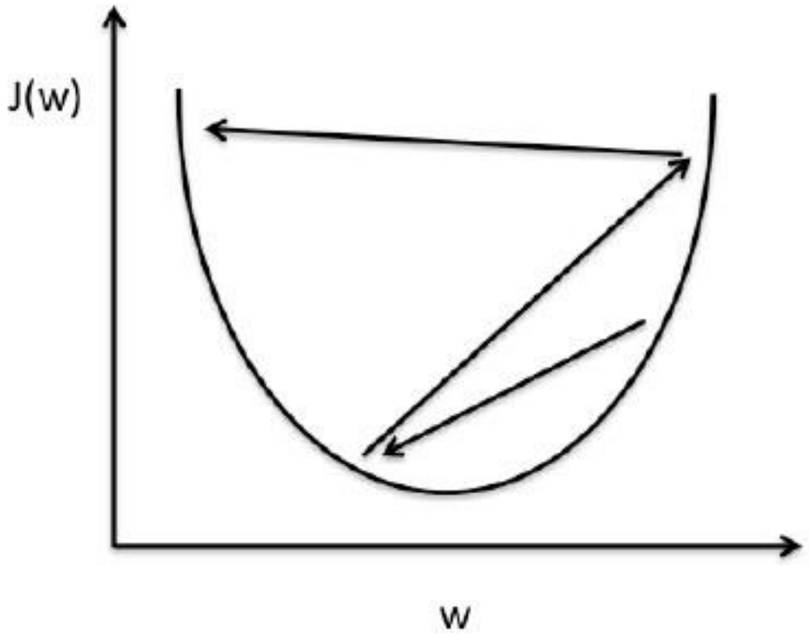
- 딥러닝 모델에서 기본적으로 사용되는 최적화 알고리즘
- 네트워크의 parameter들을 θ라고 했을 때, 손실함수 J(θ)의 값을 최소화하기 위해 기울기(gradient) ∇J(θ)를 이용하는 방법
- GD에서는 gradient의 반대 방향으로 일정 크기만큼 이동하는 것을 반복하여 손실함수의 값을 최소화하는 θ의 값을 찾음

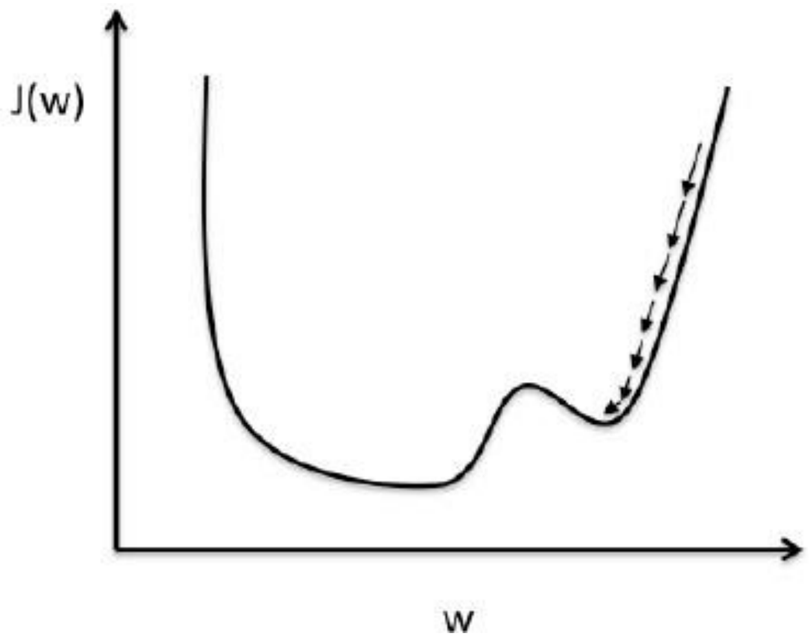
$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

- 3. 이 때 η는 미리 정해진 걸음의 크기(step size)로 학습률(learning rate)이라고 함
  - 보통 0.01~0.001 정도를 사용

Learning Rate

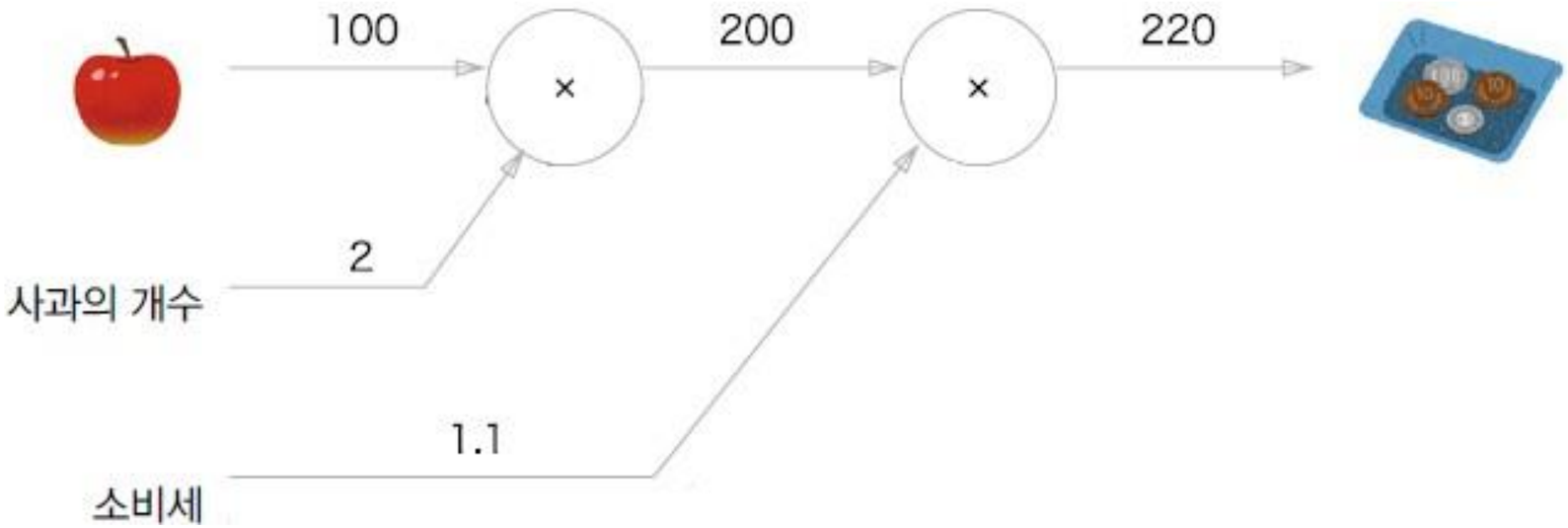
- GD에서 parameter를 최적화하기 위해 변경하는 정도
- 학습률이 너무 크면 듬성듬성하고 최소값(global minimum)을 지나쳐 갈 수 있음
- 학습률이 너무 작으면 학습을 촘촘히 해서 학습속도가 느려 지고 극소값(Local minimum)에 빠질 수 있음





계산 그래프

- 계산 그래프
  - 계산 과정을 그래프로 나타낸 것
  - 노드(node)와 엣지(edge)로 표현
  - 노드는 연산을, 엣지는 데이터가 흘러가는 방향을 나타냄

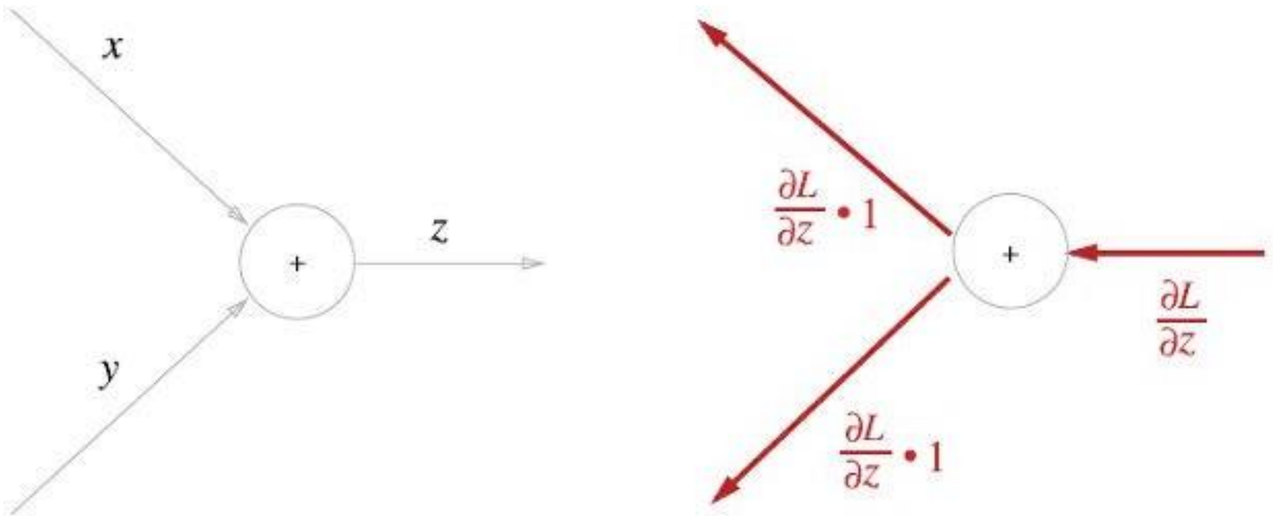


계산 그래프 예시 - 덧셈 노드

$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1$$

$$\frac{\partial z}{\partial y} = 1$$

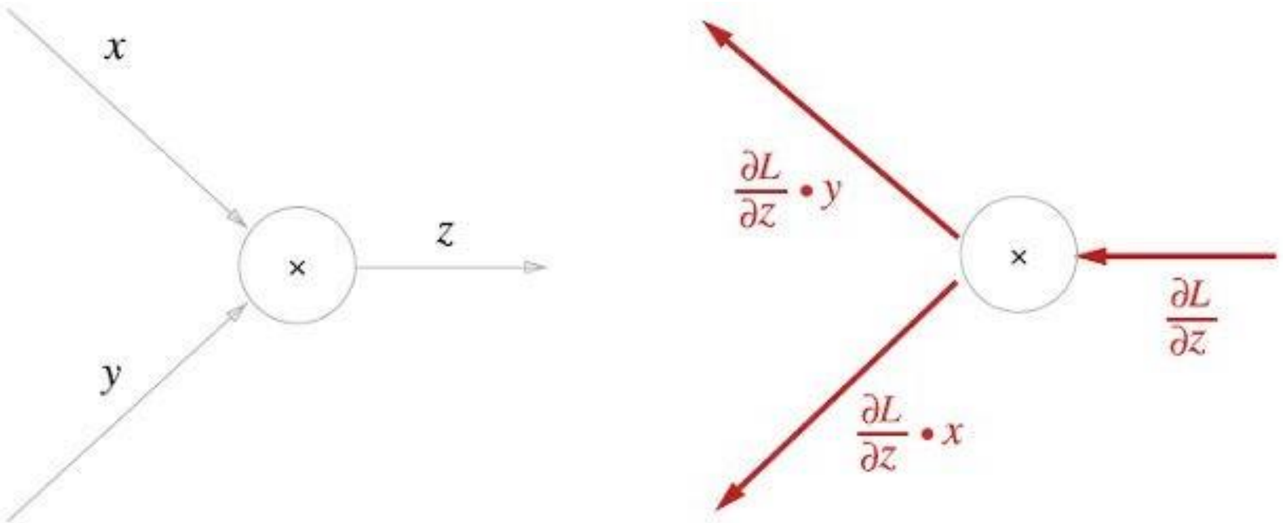


계산 그래프 예시 - 곱셈 노드

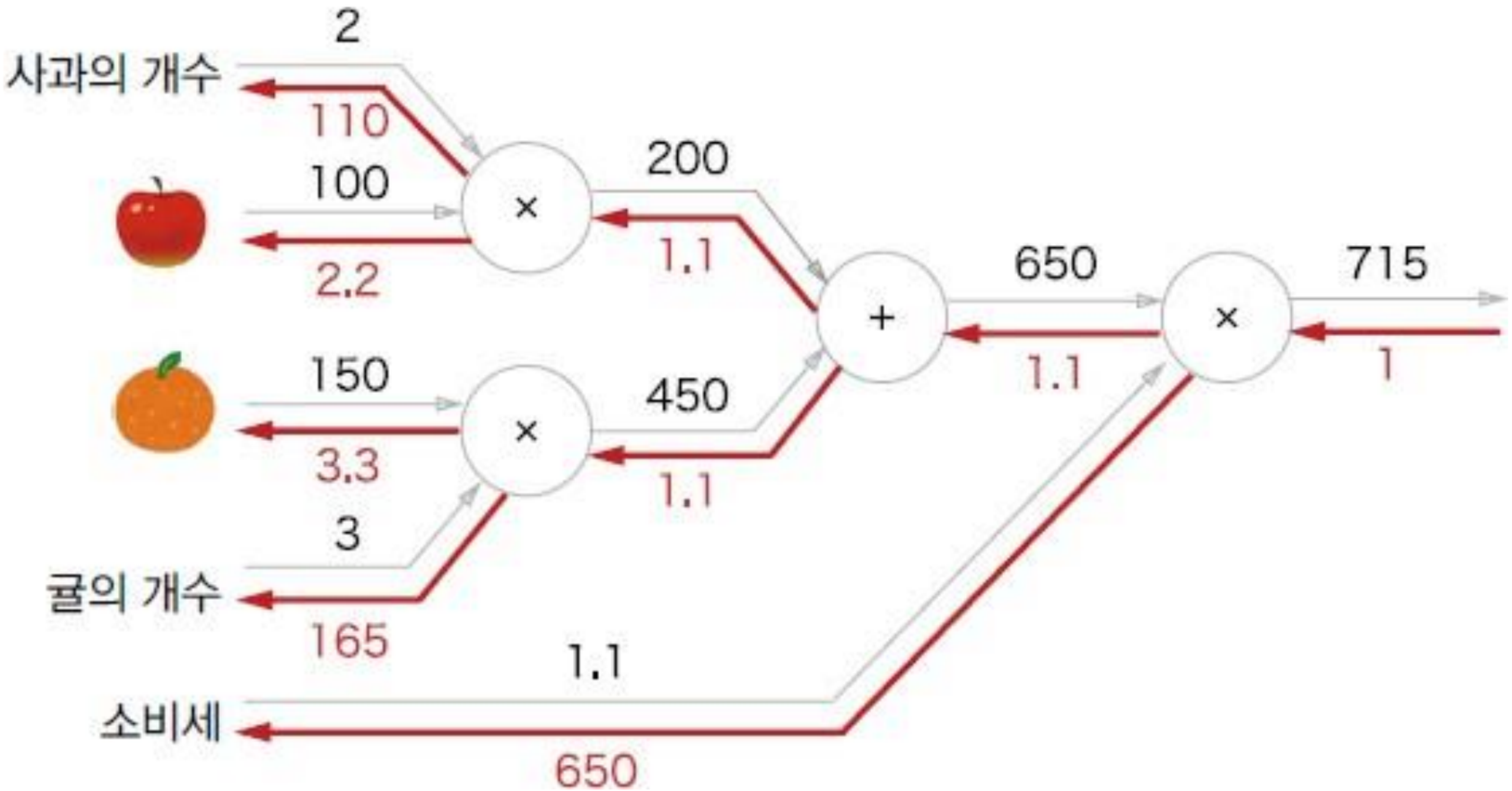
$$z = xy$$

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

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

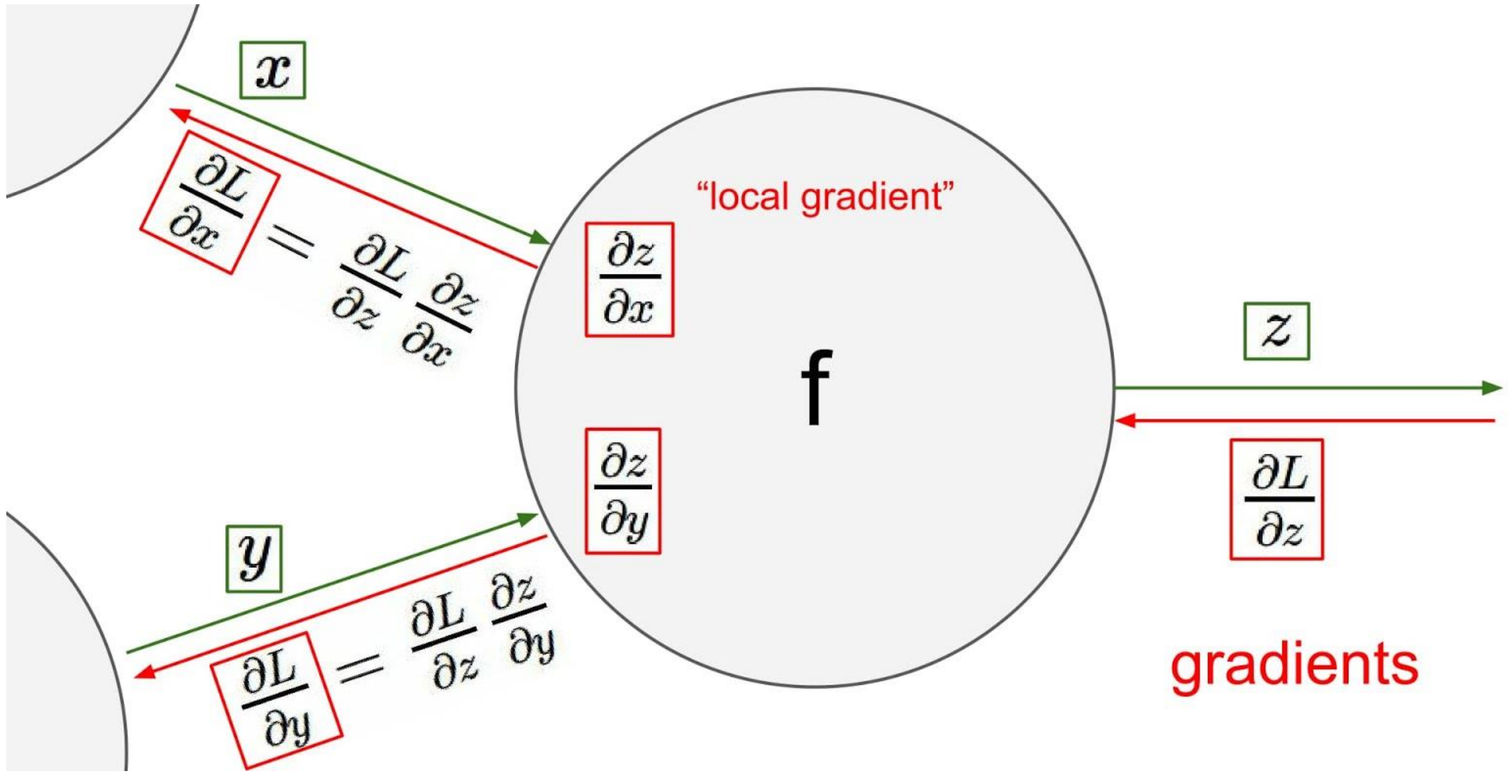


계산 그래프에서의 Back Propagation



Back Propagation

- 딥러닝 모델 학습 시 Chain Rule을 활용하여 효율적으로 각 parameter들의 gradient를 계산하는 방법



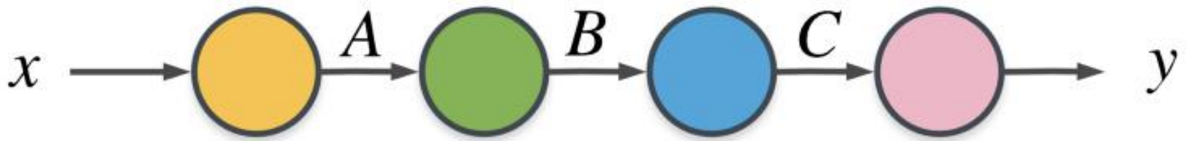
Chain Rule

If f and g are both differentiable and $F(x)$ is the composite function defined by $F(x) = f(g(x))$ then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x))\, g'(x)$$

Differentiate outer function

Differentiate inner function



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial C} \times \frac{\partial C}{\partial B} \times \frac{\partial B}{\partial A} \times \frac{\partial A}{\partial x}$$

Chain Rule 연습 문제

$$u = x^2 + 2y$$

$$x = r\sin t, \qquad y = \sin^2 t$$

일 때 $\frac{\partial u}{\partial r}, \frac{\partial u}{\partial t}$를 구하여라.

Chain Rule 연습 문제

$$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial r} = (2x)(\sin(t)) + (2)(0) = 2r\sin^2(t),$$

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial t} \\
&= (2x)(r\cos(t)) + (2)(2\sin(t)\cos(t)) \\
&= (2r\sin(t))(r\cos(t)) + 4\sin(t)\cos(t) \\
&= 2(r^2+2)\sin(t)\cos(t) \\
&= (r^2+2)\sin(2t).
\end{aligned}$$

Chain Rule 예시

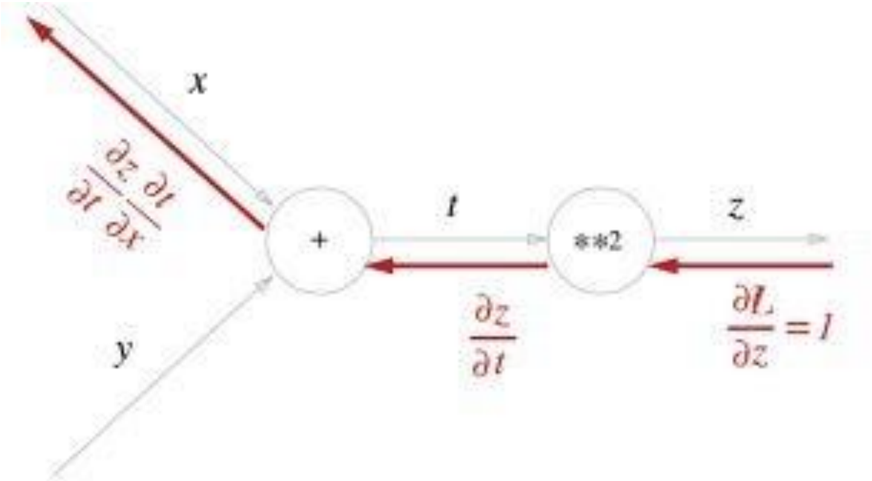
$$t = x + y$$

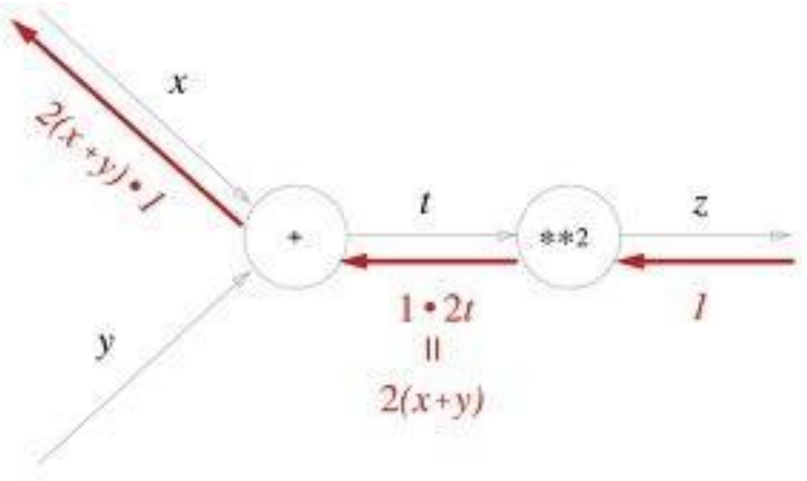
$$z = t^2$$

$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$

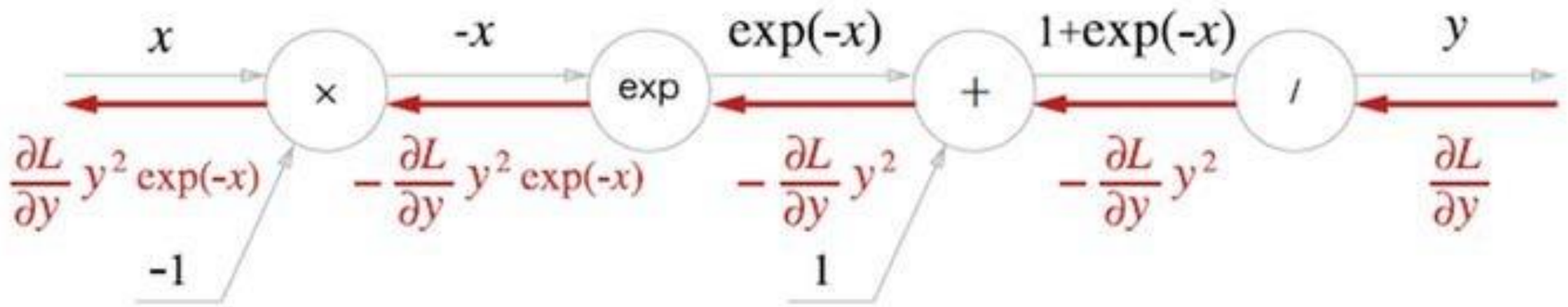
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x+y)$$

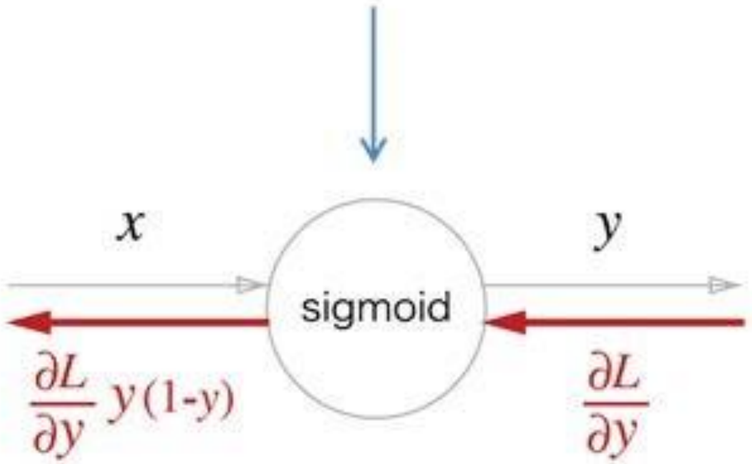




/* elice */

Chain Rule 예시 – sigmoid 함수



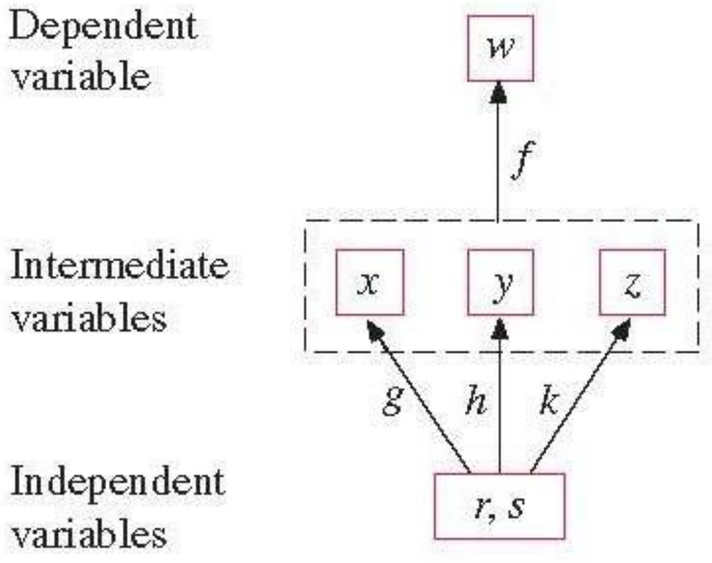


$$y = \frac{1}{1 + \exp(-x)}$$

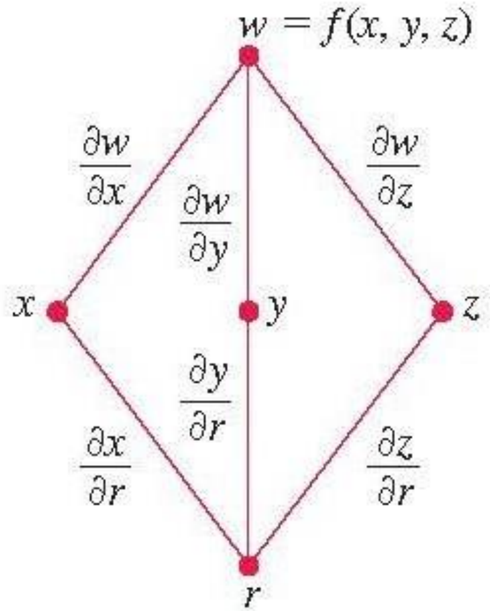
$$\begin{aligned}
\frac{\partial L}{\partial y} y^2 exp(-x) &= \frac{\partial L}{\partial y} \frac{1}{(1 + exp(-x))^2} \exp(-x) \\
&= \frac{\partial L}{\partial y} \frac{1}{1 + exp(-x)} \frac{exp(-x)}{1 + exp(-x)} \\
&= \frac{\partial L}{\partial y} y(1 - y)
\end{aligned}$$

/* elice */

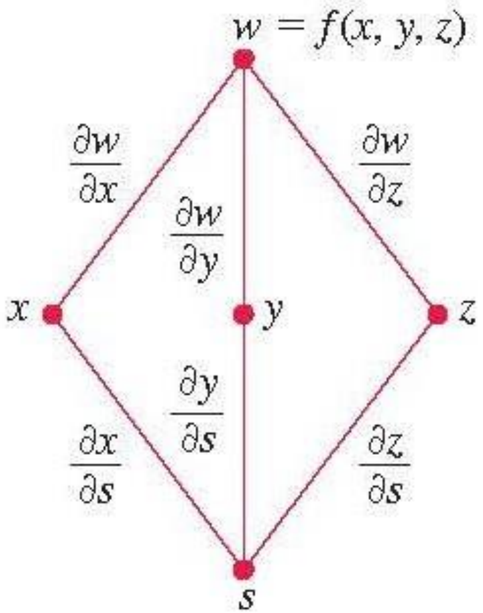
Chain Rule 예시 2



$$w = f(g(r, s), h(r, s), k(r, s))$$



$$\frac{\partial w}{\partial r} = \frac{\partial w}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial w}{\delta y}\frac{\partial y}{\partial r} + \frac{\partial w}{\partial z}\frac{\partial z}{\partial r}$$



$$\frac{\partial w}{\partial s} = \frac{\partial w}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial w}{\partial y}\frac{\partial y}{\partial s} + \frac{\partial w}{\partial z}\frac{\partial z}{\partial s}$$

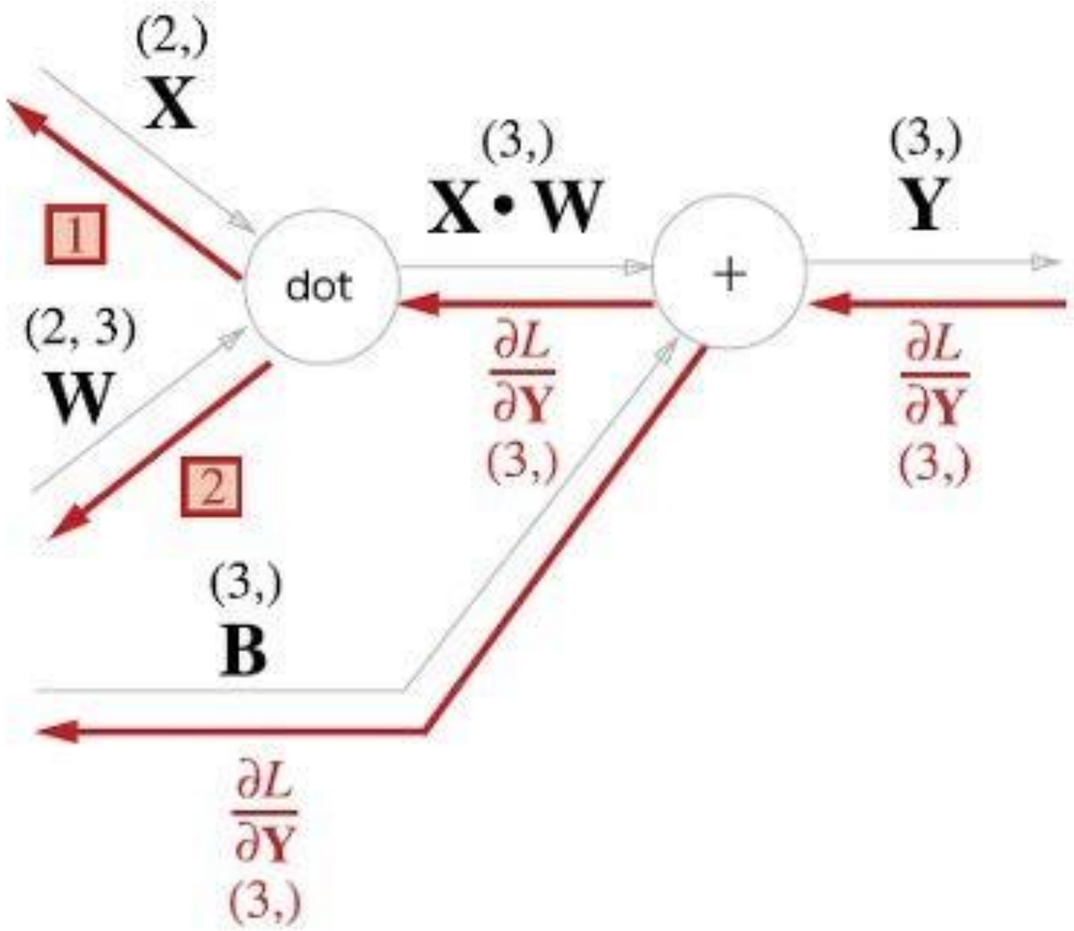
행렬 연산과 Back propagation

1 $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} W^T$

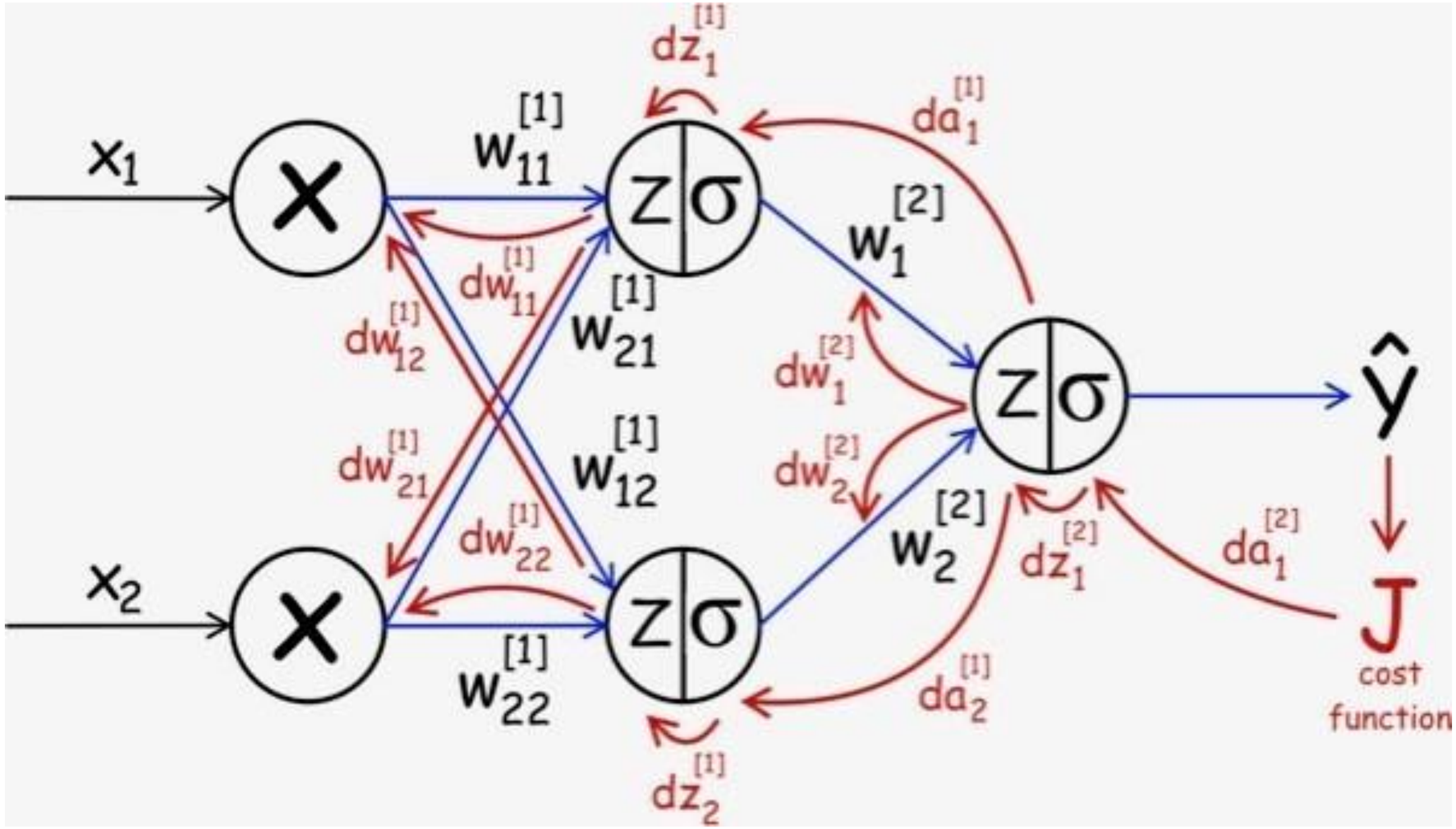
(2,) (3,) (3, 2)

2 $\frac{\partial L}{\partial W} = X^T \frac{\partial L}{\partial Y}$

(2, 3) (2, 1) (1, 3)



Back Propagation 구현 – 2 layer NN



/* elice */

Back Propagation 구현 – 2 layer NN

$$
\begin{aligned}
Z^{[1]} &= W^{[1]} X + b^{[1]} \\
A^{[1]} &= \sigma(Z^{[1]}) \\
Z^{[2]} &= W^{[2]} A^{[1]} + b^{[2]} \\
\hat{y} &= A^{[2]} = \sigma(Z^{[2]})
\end{aligned}
$$

$$
\begin{aligned}
W^{[1]} &=: W^{[1]} - \alpha \frac{dJ}{dW^{[1]}} \\
b^{[1]} &=: b^{[1]} - \alpha \frac{dJ}{db^{[1]}} \\
W^{[2]} &=: W^{[2]} - \alpha \frac{dJ}{dW^{[2]}} \\
b^{[2]} &=: b^{[2]} - \alpha \frac{dJ}{db^{[2]}}
\end{aligned}
$$

Back Propagation 구현 – 2 layer NN

$$J = -\frac{1}{n}\left(Y log\left(A^{[2]}\right) - (1-Y)\, log\left(1 - A^{[2]}\right)\right)$$

$$\frac{dJ}{dW^{[2]}} = \left[-\frac{Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}}\right]\left[A^{[2]}(1-A^{[2]})\right]\left[A^{[2]}\right]$$

$$\frac{dJ}{db^{[2]}} = \frac{dJ}{dA^{[2]}}\frac{dA^{[2]}}{dZ^{[2]}}\frac{dZ^{[2]}}{db^{[2]}}$$

$$= \left[A^{[2]} - Y\right]\left[1\right]$$

$$= \left[A^{[2]} - Y\right]$$

$$= dZ^{[2]}$$

Back Propagation 구현 – 2 layer NN

$$\begin{aligned}\frac{dJ}{dW^{[1]}} &= \frac{dJ}{dA^{[2]}}\frac{dA^{[2]}}{dZ^{[2]}}\frac{dZ^{[2]}}{dA^{[1]}}\frac{dA^{[1]}}{dZ^{[1]}}\frac{dZ^{[1]}}{dW^{[1]}} \\ &= \frac{dJ}{dZ^{[2]}}\frac{dZ^{[2]}}{dA^{[1]}}\frac{dA^{[1]}}{dZ^{[1]}}\frac{dZ^{[1]}}{dW^{[1]}} \\ &= \left[A^{[2]} - Y\right]\left[W^{[2]}\right]\left[g'\left(Z^{[1]}\right)\right]\left[A^{[0]}\right] \\ &= dZ^{[2]}W^{[2]}g'\left(Z^{[1]}\right)A^{[0]} \\ &= dZ^{[1]}A^{[0]}\end{aligned}$$

$$\begin{aligned}\frac{dJ}{db^{[1]}} &= \frac{dJ}{dA^{[2]}}\frac{dA^{[2]}}{dZ^{[2]}}\frac{dZ^{[2]}}{dA^{[1]}}\frac{dA^{[1]}}{dZ^{[1]}}\frac{dZ^{[1]}}{db^{[1]}} \\ &= \frac{dJ}{dZ^{[2]}}\frac{dZ^{[2]}}{dA^{[1]}}\frac{dA^{[1]}}{dZ^{[1]}}\frac{dZ^{[1]}}{db^{[1]}} \\ &= \left[A^{[2]} - Y\right]\left[W^{[2]}\right]\left[g'\left(Z^{[1]}\right)\right]\left[1\right] \\ &= dZ^{[2]}W^{[2]}g'\left(Z^{[1]}\right) \\ &= dZ^{[1]}\end{aligned}$$

- 다음 layer의 gradient 값들을 이용해 이전 layer의 gradient 값을 구할 수 있다. (병렬연산가능)

Back Propagation 구현 – L layer NN

- Initialize $W^{[1]}..W^{[L]}, b^{[1]}...b^{[L]}$
- Set $A^{[0]} = X$ (Input) , $L =$ Total Layers
- Loop epoch = 1 to max iteration
  - Forward Propagation
    - Loop $l = 1$ to $L - 1$
      - $Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$
      - $A^{[l]} = g\left(b^{[l]}\right)$
      - Save $A^{[l]}, W^{[l]}$ in memory for later use
    - $Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$
    - $A^{[L]} = \sigma\left(Z^{[L]}\right)$
  - Cost $J = -\frac{1}{n}\left(Y log\left(A^{[2]}\right) - (1-Y)\, log\left(1 - A^{[2]}\right)\right)$

Back Propagation 구현 – L layer NN

- Backward Propagation
  - $dA^{[L]} = -\frac{Y}{A^{[L]}} + \frac{1-Y}{1-A^{[L]}}$
  - $dZ^{[L]} = dA^{[L]}\sigma'\left(dA^{[L]}\right)$
  - $dW^{[L]} = dZ^{[L]}dA^{[L-1]}$
  - $db^{[L]} = dZ^{[L]}$
  - $dA^{[L-1]} = dZ^{[L]}W^{[L]}$
  - Loop $l = L-1$ to 1
    - $dZ^{[l]} = dA^{[l]}g'\left(dA^{[l]}\right)$
    - $dW^{[l]} = dZ^{[l]}dA^{[l-1]}$
    - $db^{[l]} = dZ^{[l]}$
    - $dA^{[l-1]} = dZ^{[l]}W^{[l]}$

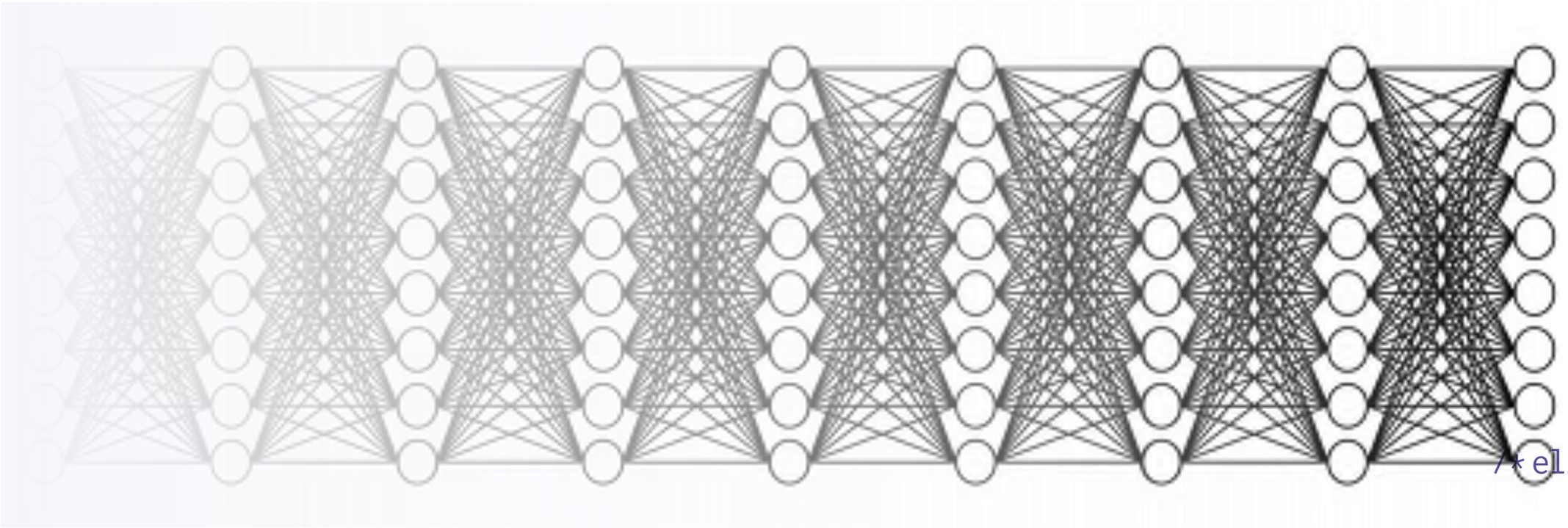
- Update W and b
  - Loop $l = 1$ to L
    - $W^{[l]} = W^{[l]} - \alpha.\, dW^{[l]}$
    - $b^{[l]} = b^{[l]} - \alpha.\, db^{[l]}$

딥러닝 모델 학습의 문제점

- Vanishing Gradient
  - Sigmoid -> ReLU로 변경하여 해결
- 잘못된 초기값 설정
  - Xavier Initialization, He Initialization으로 해결
- 과적합 문제
  - Regularization, Dropout으로 해결
- 불안정한 학습 과정
  - Batch Normalization으로 해결

Vanishing Gradient

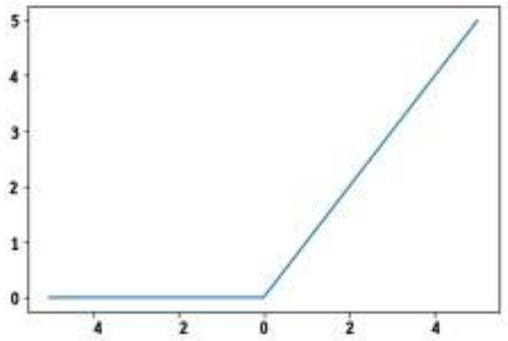
- 더 깊고 더 넓은 망을 학습시키는 과정에서 Output 값과 멀어질 수록 학습이 잘 안되는 현상



Vanishing Gradient

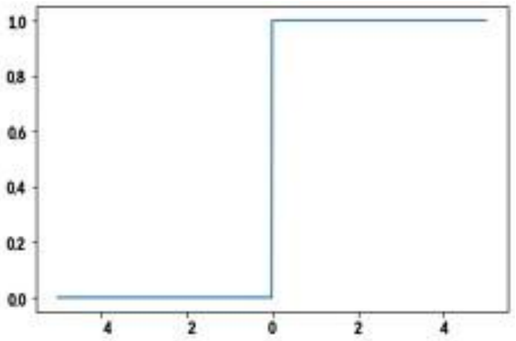
- 기존에 사용하던 sigmoid 함수 대신 ReLU 함수를 사용하여 해결
- 내부 hidden layer에는 ReLU를 적용하고 output layer에서만 Tanh를 적용

ReLU



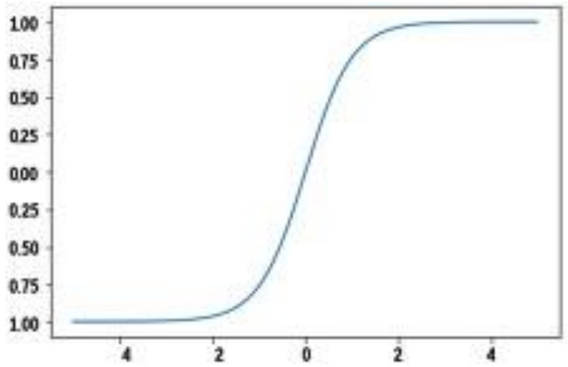
$$ReLU(x) = \max(0,\ x)$$

Rectified Linear Unit

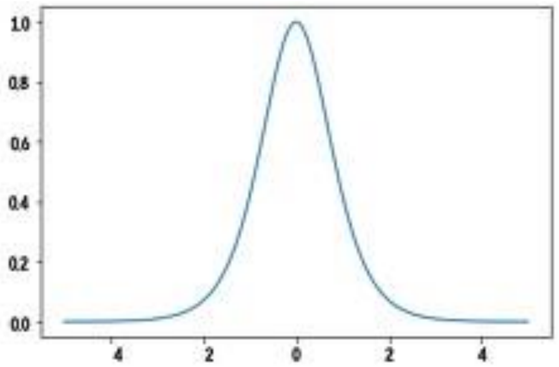


$$R'(y) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$

Tanh



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



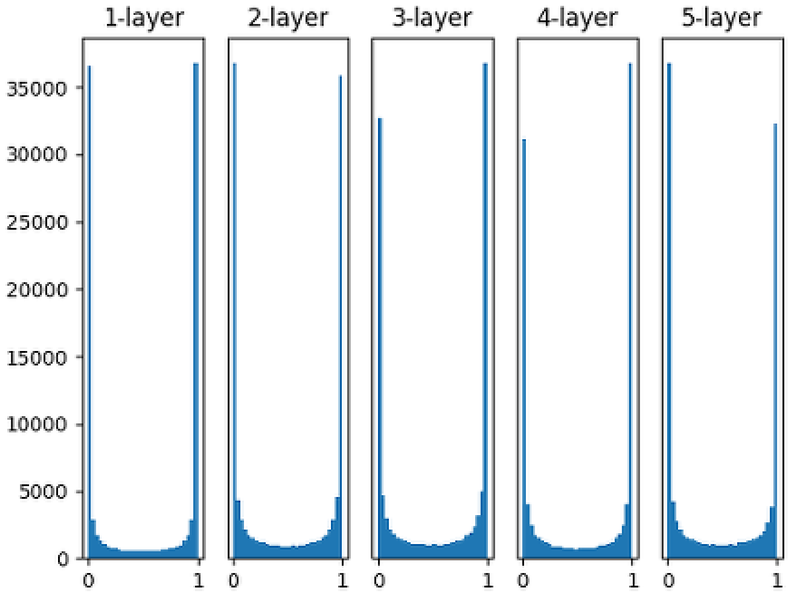
$$T'(x) = 1 - \tanh^2(x)$$

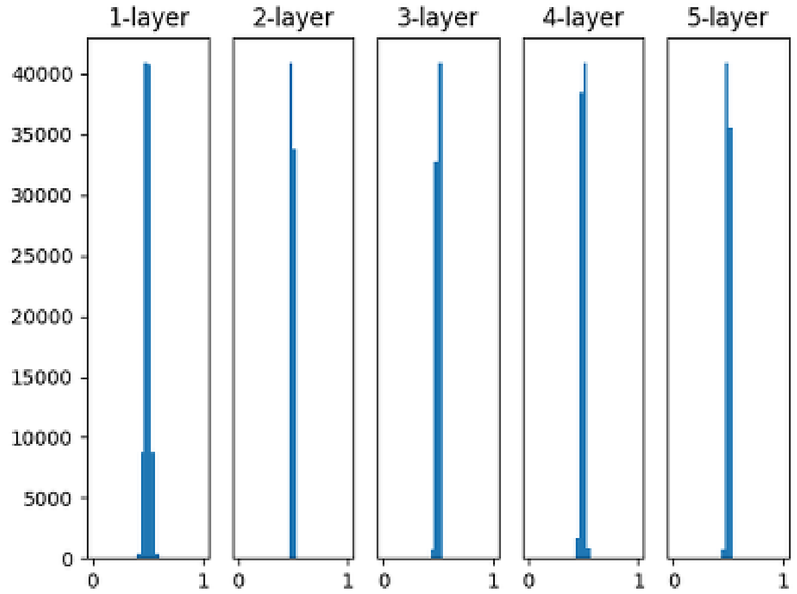
Parameter initialization

- 초기화의 중요성
  - t = wx + b에서 w가 100, b가 50이라면 x가 0.01이더라도 t는 51이 됨
    - 역전파시 sigmoid 함수(σ)를 통과시키면 σ'(51)가 반환
  - t가 5만 넘어도 σ'(t)는 0에 수렴
    - 기울기 소실(Vanishing gradient) 문제 발생
  - 입력층의 가중치 W를 모두 0으로 초기화한다면?
    - 순전파때 두번째 층의 뉴런에 모두 같은 값이 전달됨
    - 역전파때 두번째 층의 가중치가 모두 똑같이 갱신
    - 신경망의 표현력을 제한
  - Bias는 0으로 초기화 하는 것이 일반적으로 효율적

Naïve한 방법

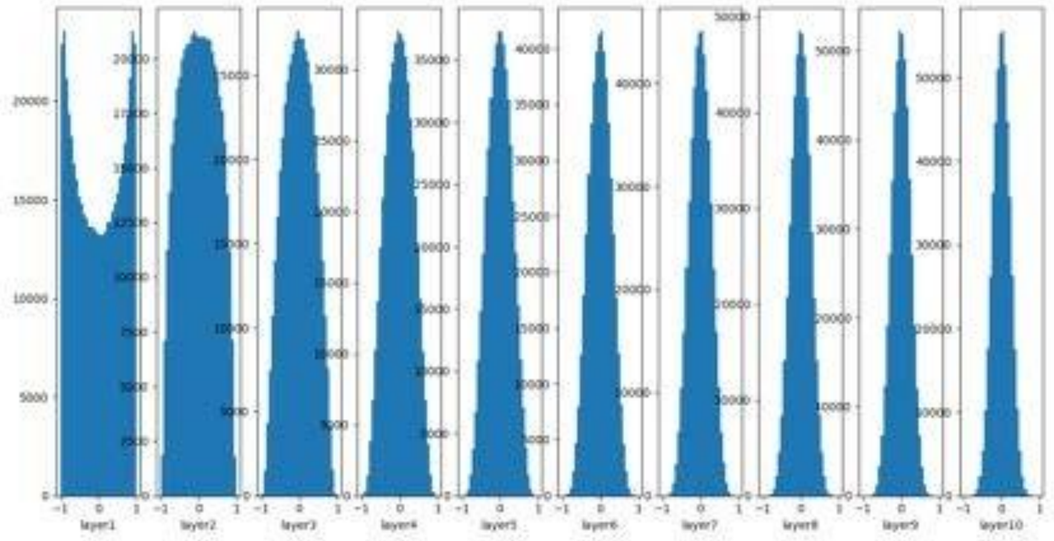
- 가중치를 표준 정규분포를 이용해 초기화
  - Sigmoid 함수의 출력값이 0 과 1에 치우치는 현상이 발생
    - 기울기 소실 문제 발생
- 표준편차를 0.01로 하는 정규분포로 초기화(Naïve)
  - 가중치가 0.5 중심으로 모여 있음 => 기울기 소실 문제 완화





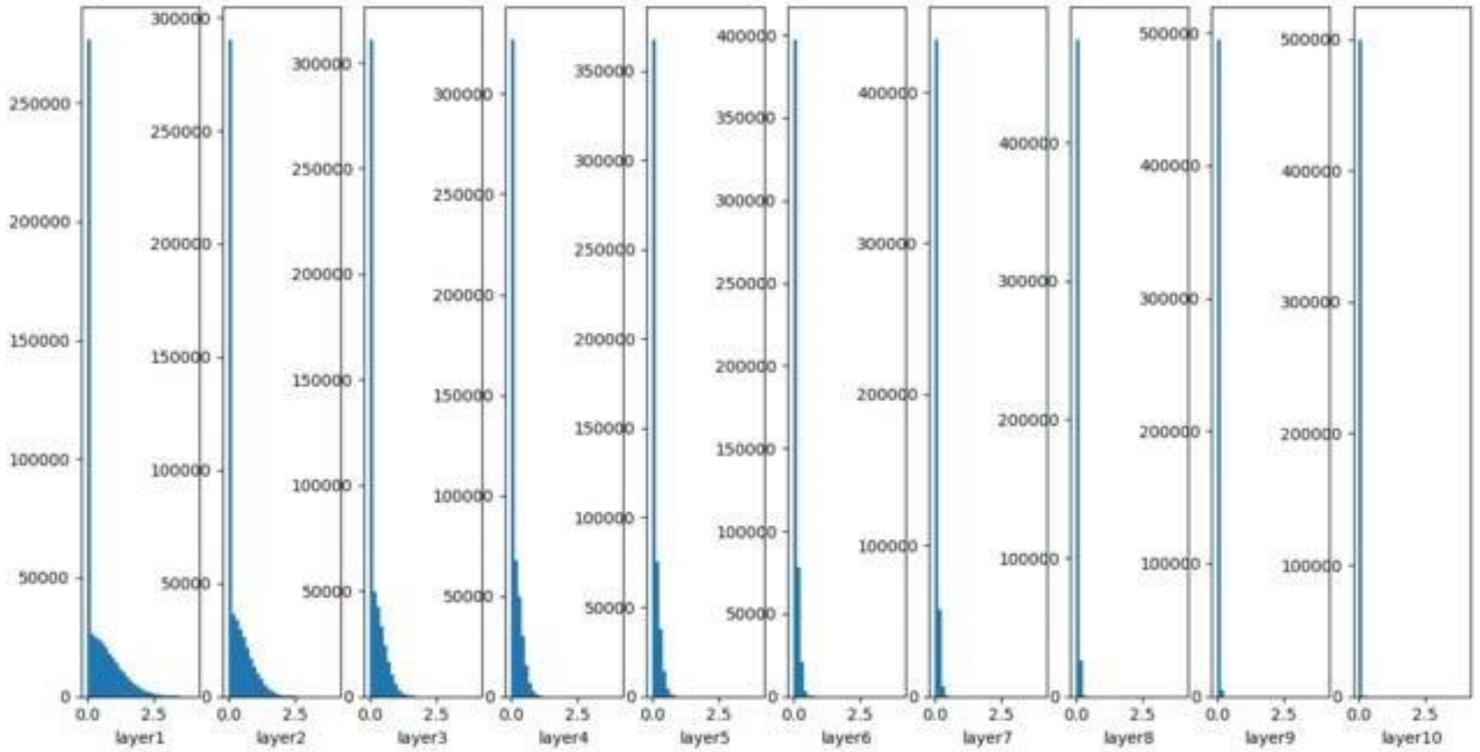
Xavier 초기화 방법

- 표준 정규 분포를 입력 개수의 제곱근으로 나누어 줌
  - `w = np.random.randn(n_input, n_output) / sqrt(n_input)`
- Sigmoid와 같은 S자 함수의 경우 출력값들이 정규 분포 형태를 가져야 안정적으로 학습 가능
- Sigmoid 함수와 Xavier 초기화 방법을 사용했을 경우의 그래프는 아래와 같음



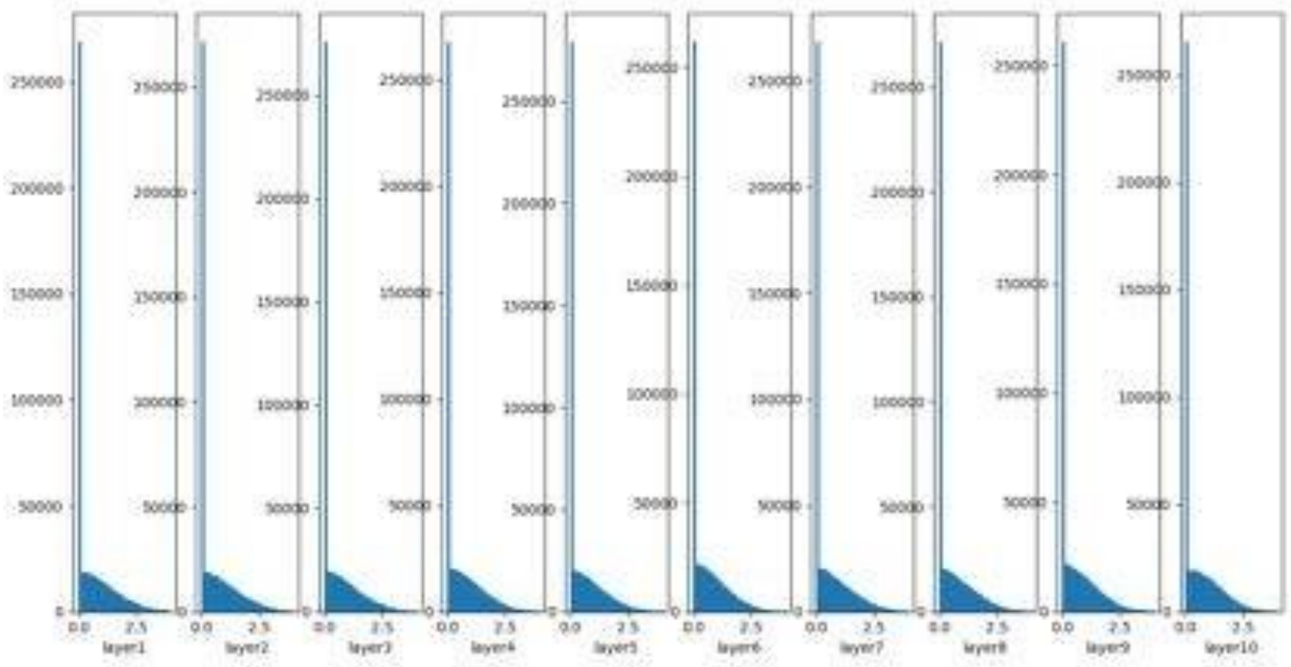
Xavier 초기화 방법

- ReLU 함수에는 Xavier 초기화가 부적합
- ReLU 함수와 Xavier 초기화 방법을 사용했을 경우의 그래프 레이어를 거쳐갈수록 값이 0에 수렴



He 초기화 방법

- 표준 정규 분포를 입력 개수 절반의 제곱근으로 나누어 줌
  - `w = np.random.randn(n_input, n_output) / sqrt(n_input / 2)`
- ReLU 함수와 He 초기화 방법을 사용했을 경우의 그래프는 아래와 같음
  - 10층 레이어에서도 평균과 표준편차가 0 으로 수렴하지 않음



Parameter Initialization

- Sigmoid, tanh의 경우 Xavier 초기화 방법이 효율적
- ReLU계의 활성화 함수 사용 시 Xavier 초기화보다는 He 초기화 방법이 효율적
- 최근의 대부분의 모델에서는 He초기화를 주로 선택

03

최적화 알고리즘

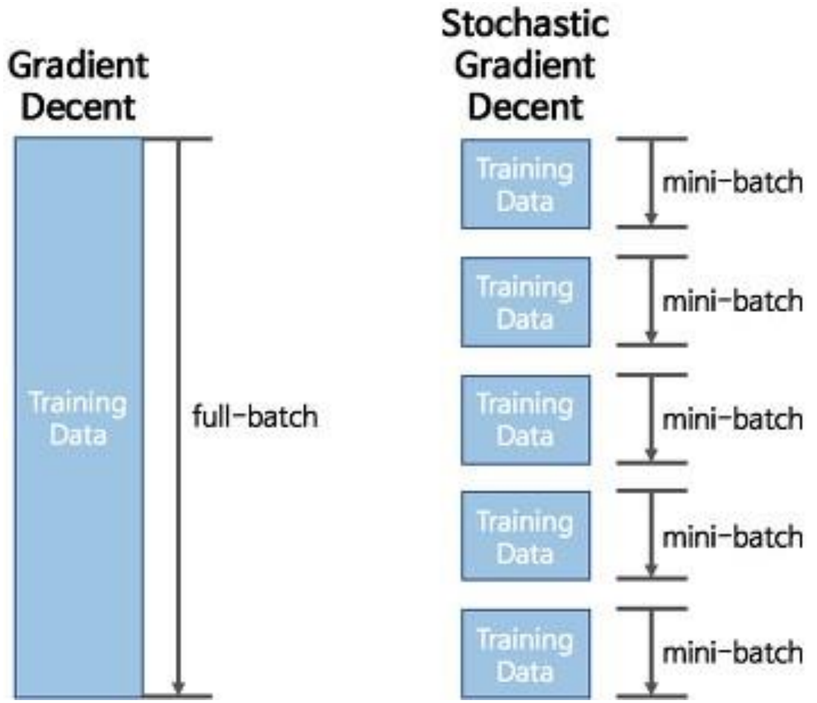
03 최적화 알고리즘

최적화 알고리즘의 종류

- GD(Gradient Descent)
- SGD(Stochastic Gradient Descent)
- Momentum
- AdaGrad
- RMSProp
- Adam

SGD(Stochastic Gradient Descent)

- 손실함수를 계산할 때 전체 training set을 사용하는 것을 Batch Gradient Descent라 함
- 계산량이 너무 많아지는 것을 방지하기 위해 보통 SGD를 사용
- 전체 데이터(batch) 대신 일부 조그마한 데이터의 모음인 미니배치(mini-batch)에 대해서만 손실함수를 계산



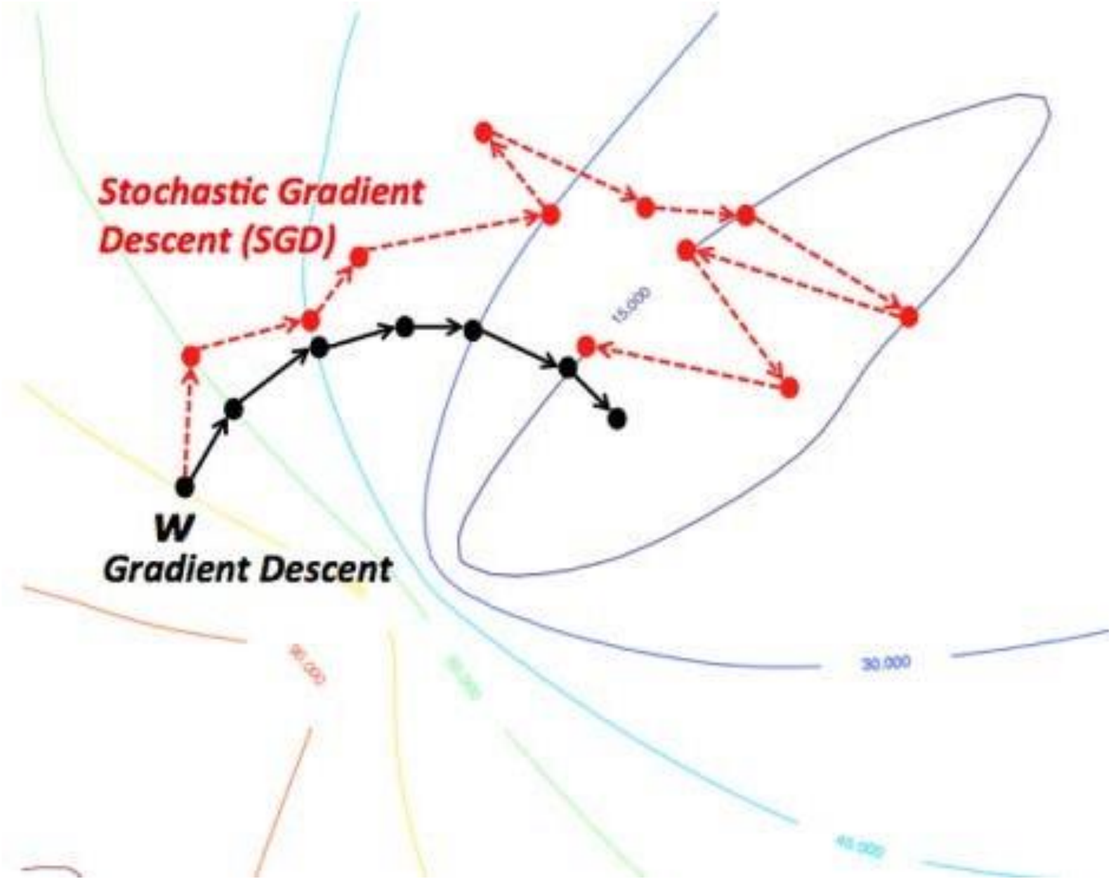
/* elice */

SGD(Stochastic Gradient Descent)

- 다소 부정확할 수 있지만, 훨씬 계산 속도가 빠르기 때문에 같은 시간에 더 많은 step을 갈 수 있음
- 여러 번 반복할 경우 보통 batch의 결과와 유사한 결과로 수렴

GD vs SGD

- GD
  - 모든 데이터를 계산
    - 1스텝 1시간 소요
    - 6 스텝 * 1시간 = 6시간
    - 확실한데 너무 느림
- SGD
  - 일부 데이터만 계산
    - 1스텝 5분 소요
    - 10 스텝 * 5분 => 50분
    - 조금 헤메지만 빠름

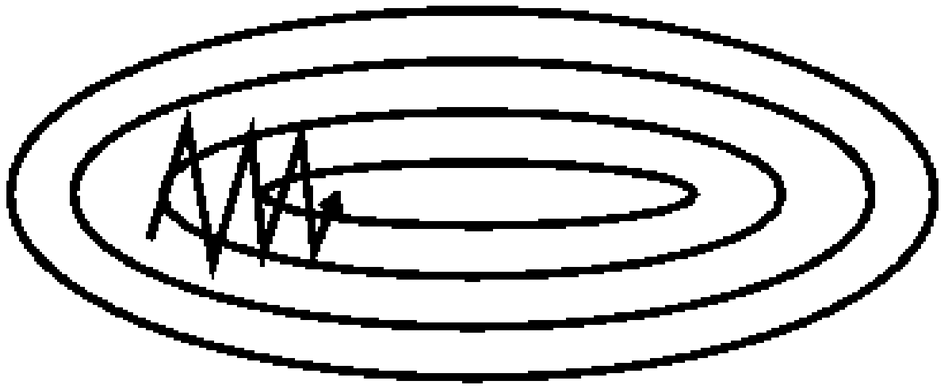


Momentum

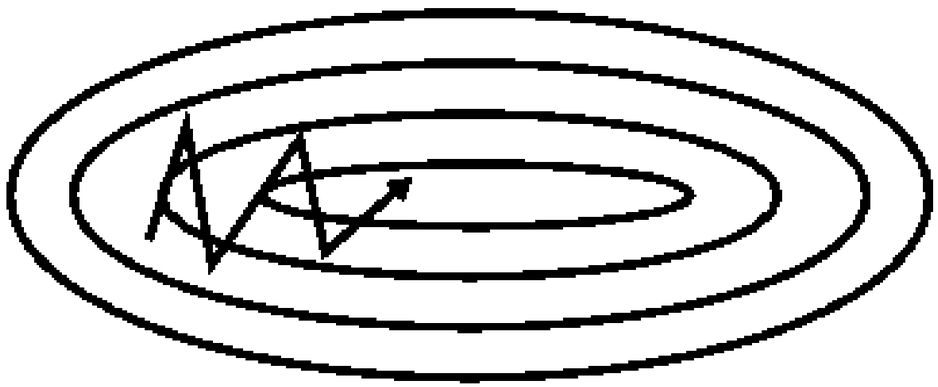
- 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식, 즉 일종의 관성을 주는 방식

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

$$\theta = \theta - v_t$$



without momentum



with momentum

AdaGrad(Adaptive Gradient)

- 기본적인 아이디어는 많이 변화하지 않은 변수들은 step size를 크게 하고, 많이 변화했던 변수들은 step size를 작게 하는 것
- 자주 등장하거나 변화를 많이 한 변수들은 optimum에 가까이 있을 확률이 높기 때문에 작은 크기로 이동하면서 세밀하게 조절
- 적게 변화한 변수들은 많이 이동해야 할 확률이 높기 때문에 먼저 빠르게 loss 값을 줄이는 방향으로 이동하려는 방식

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

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

- 학습을 계속 진행하면 step size가 너무 줄어든다는 문제점이 있음

RMSProp

- Adagrad의 단점을 해결하기 위해 합을 지수평균으로 대체
- G가 무한정 커지지는 않으면서 최근 변화량의 변수간 상대적인 크기 차이는 유지할 수 있음

$$G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G+\epsilon}} \cdot \nabla_\theta J(\theta_t)$$

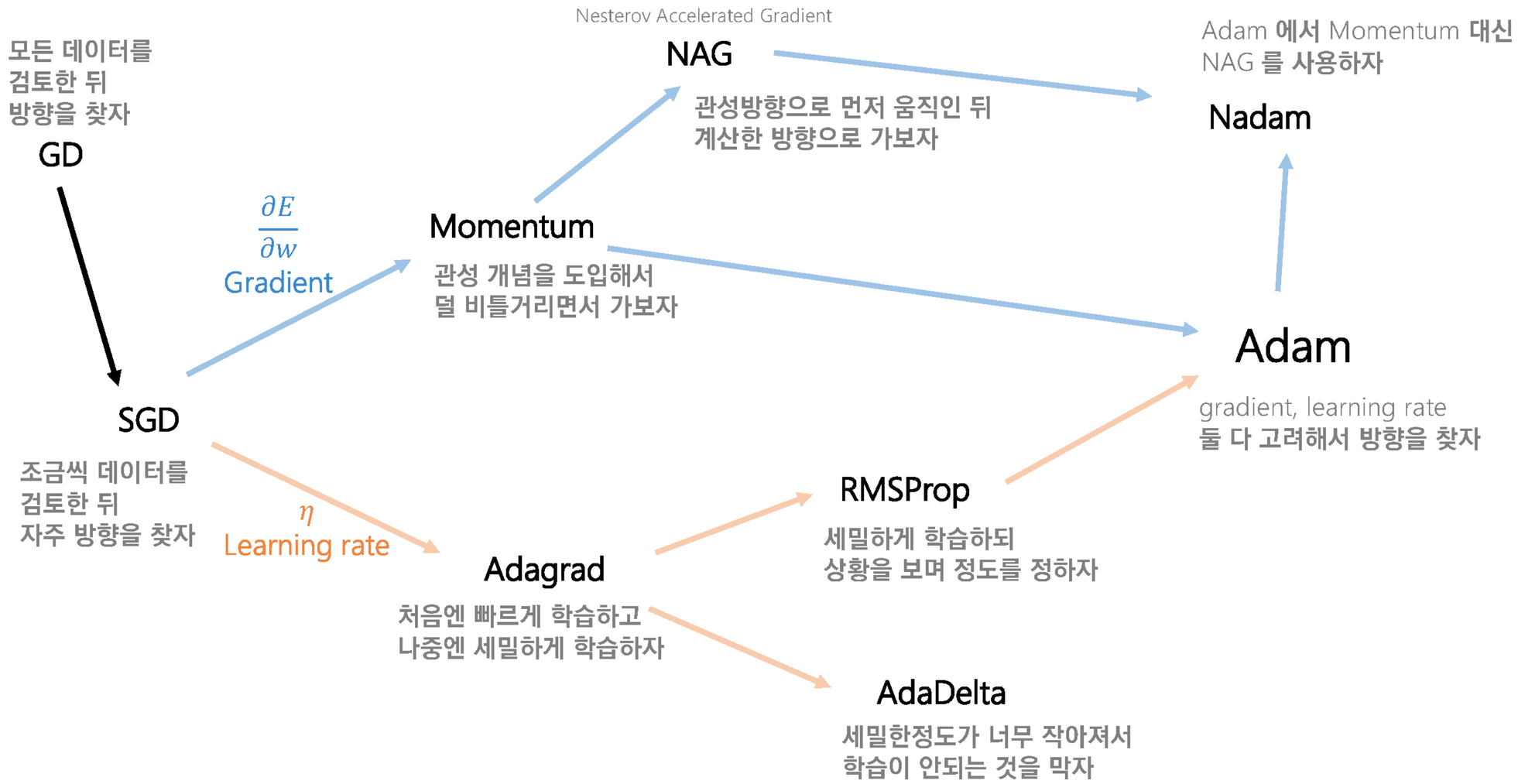
Adam(Adaptive Moment Estimation)

- Momentum과 RMSProp을 합친 알고리즘
- Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장
- RMSProp과 유사하게 기울기의 제곱값의 지수평균을 저장

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

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))$$

요약


모든 데이터를
검토한 뒤
방향을 찾자
GD
$\frac{\partial E}{\partial w}$
Gradient
SGD
조금씩 데이터를
검토한 뒤
자주 방향을 찾자
η
Learning rate
Momentum
관성 개념을 도입해서
덜 비틀거리면서 가보자
Nesterov Accelerated Gradient
NAG
관성방향으로 먼저 움직인 뒤
계산한 방향으로 가보자
Adagrad
처음엔 빠르게 학습하고
나중엔 세밀하게 학습하자
RMSProp
세밀하게 학습하되
상황을 보며 정도를 정하자
AdaDelta
세밀한정도가 너무 작아져서
학습이 안되는 것을 막자
Adam
gradient, learning rate
둘 다 고려해서 방향을 찾자
Nadam
Adam 에서 Momentum 대신
NAG 를 사용하자

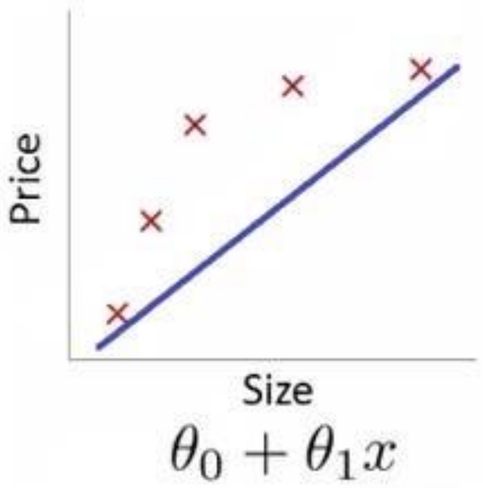
/* elice */

04

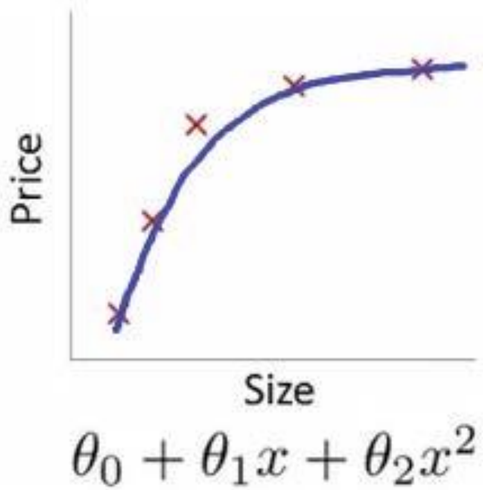
과적합 방지 기술

과적합(Overfitting)

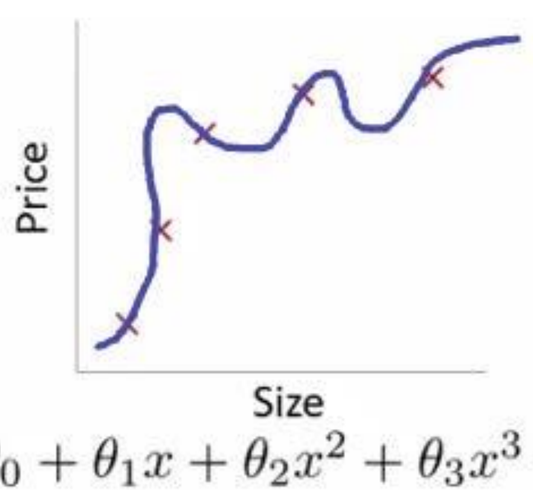
- Training 데이터의 많은 공통특성 이외에 지엽적인 특성까지 반영해 high variance로 훈련되어, Test 데이터에 대해서는 제대로 예측하지 못하는 현상
- 주로 파라미터가 많은 모델에 발생 (표현력이 높은 모델)



$\theta_0 + \theta_1 x$



$\theta_0 + \theta_1 x + \theta_2 x^2$

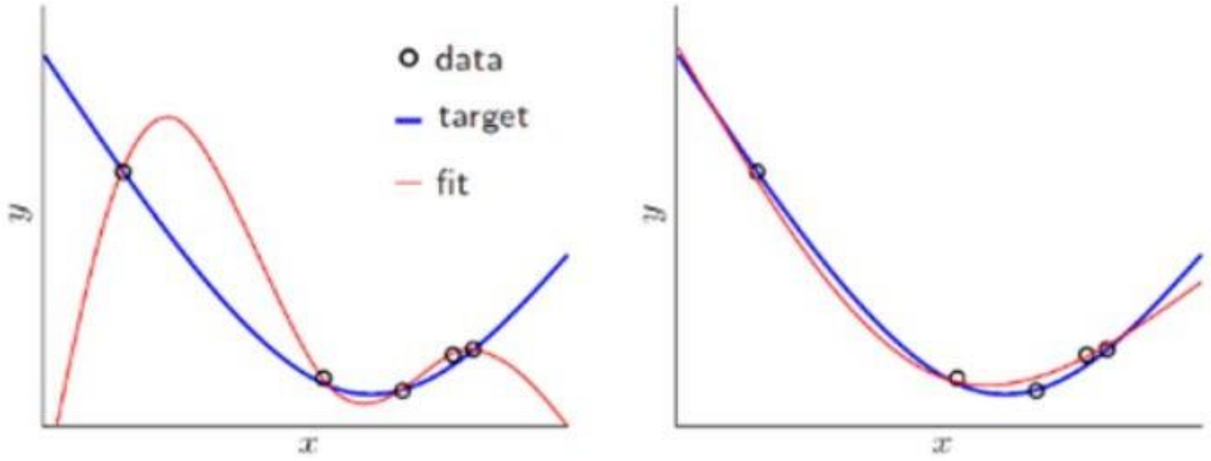


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

/* elice */

정규화(Regularization)

- 과적합을 억제하기 위해서 사용하는 기법
- 손실함수에 가중치의 크기를 포함
- 가중치가 작아지도록 학습한다는 것은 노이즈에 영향을 덜 받도록 하겠다는 것
  - Outlier의 영향을 적게 받음



(a) without regularization (b) with regularization

정규화(Regularization)

- L2 정규화

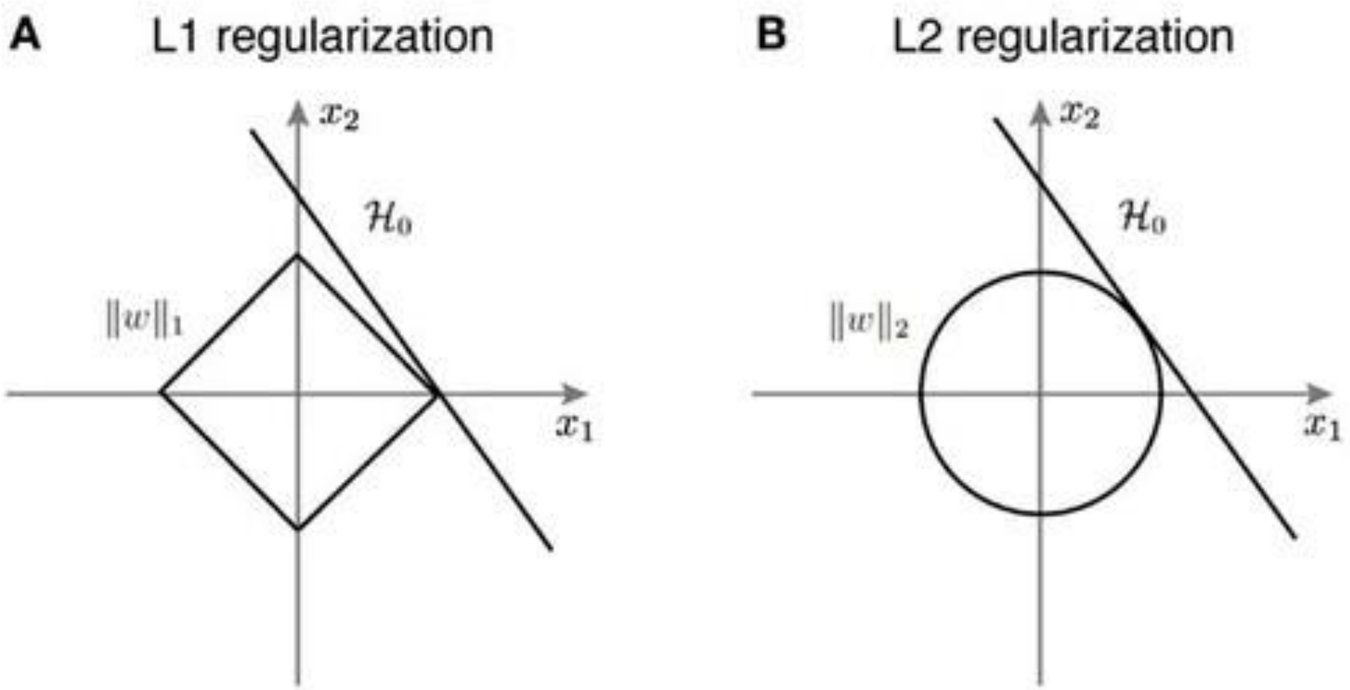
$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|^2\}$$

- L1 정규화
  - 작은 가중치들이 거의 0으로 수렴하여 몇개의 중요한 가중치들만 남는 경향이 있어서 Sparse model에 적합
  - 컨벡스(Convex) 최적화에 유용하게 쓰임

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y_i}) + \frac{\lambda}{2}|w|\}$$

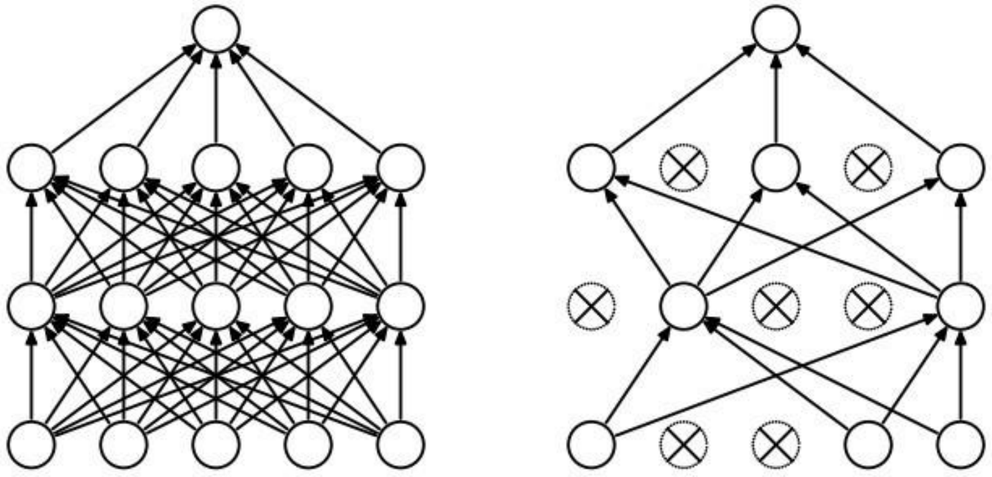
정규화(Regularization)

- 단, L1 Regularization 의 경우 아래 그림처럼 미분 불가능한 점이 있기 때문에 Gradient-base learning에는 주의가 필요



드랍 아웃(Dropout)

- 각 계층마다 일정 비율의 뉴런을 임의로 drop시켜 나머지 뉴런들만 학습하는 방법
- 드롭아웃을 적용하면 학습되는 노드와 가중치들이 매번 달라져 과적합을 효과적으로 예방
  - 망 내부의 앙상블 학습으로 볼 수 있음



(a) Standard Neural Net

(b) After applying dropout.

04 과적합 방지 기술

드랍 아웃(Dropout)

- 드롭아웃 비율은 은닉층 50%, 입력층 26% 정도가 일반적
- 다른 정규화 기법들과 상호 보완적으로 사용 가능
- 역전파는 ReLU처럼 동작
  - 순전파 때 신호를 통과시킨 뉴런은 역전파 때도 통과시키고, drop된 뉴런은 역전파 때도 신호를 차단
- Test 때는 모든 뉴런에 신호를 전달한다는 것에 주의

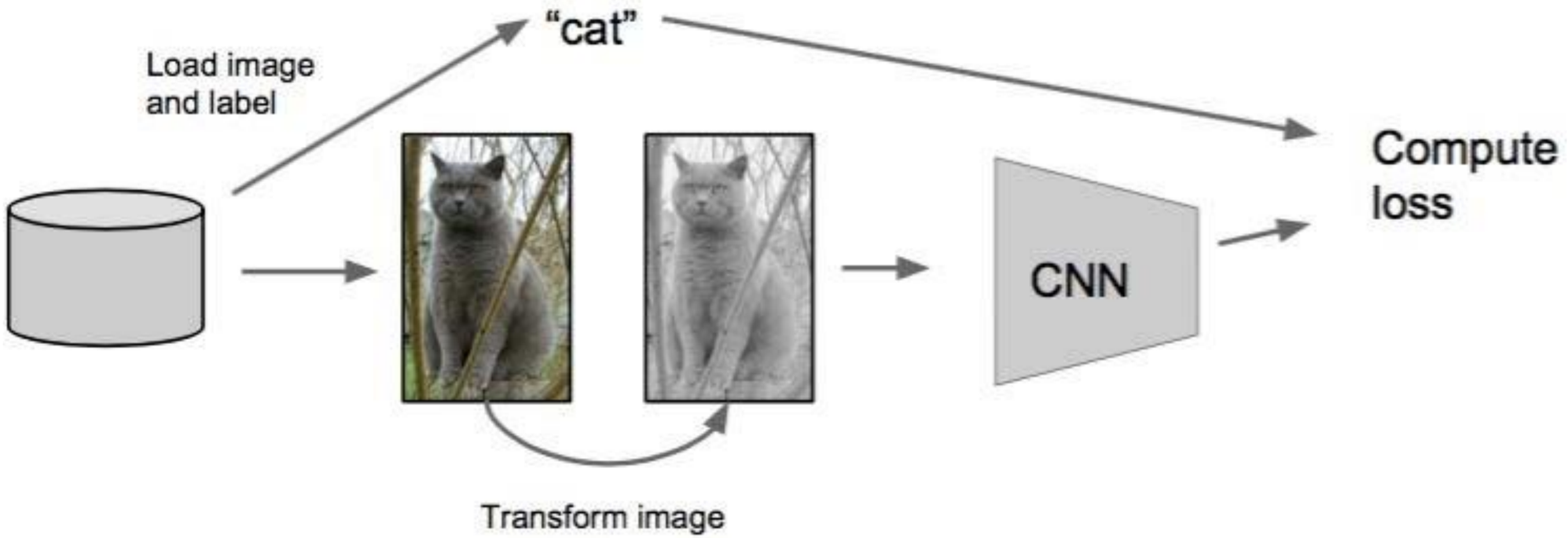
05

Data Augmentation

Data Augmentation

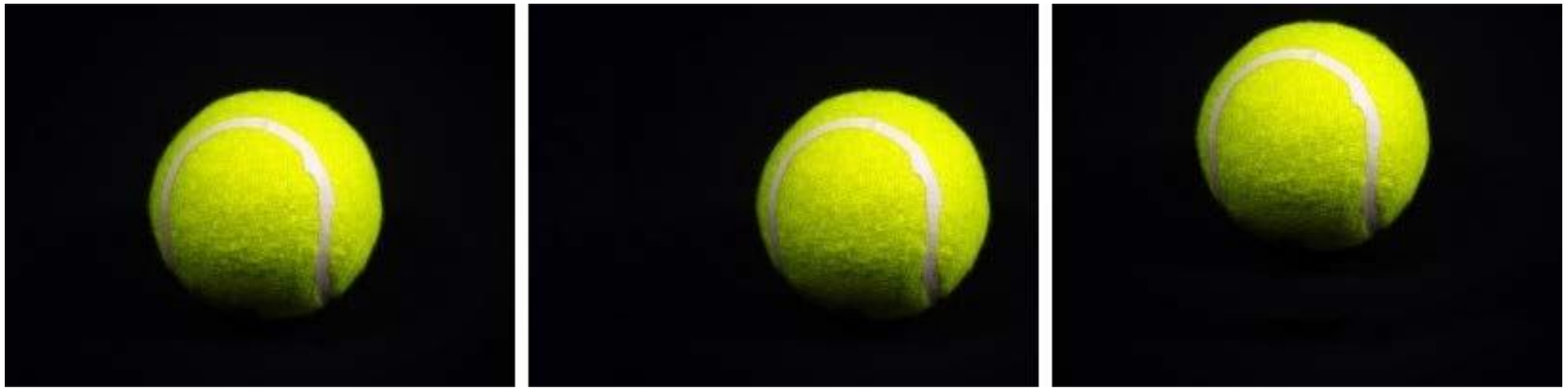
- 데이터를 늘려 네트워크(CNN 등)의 성능을 높이기 위해 사용하는 방법
  - 특히 데이터가 적을 때 사용하면 매우 효과적
- 이미지를 여러 방법을 통해 변형(transform) 한 뒤에 네트워크의 입력 이미지로 사용하는 방식
- 모델이 다양한 예외 혹은 변형된 케이스들에 대하여 학습하게 되어 robust해짐
  - 일종의 정규화(Regularization) 작업으로 과적합을 막는 효과도 있음

Data Augmentation



평행이동(Translation)

- 이미지에서 모든 픽셀을 오른쪽으로 1픽셀 이동
  - 사람의 눈에는 같은 이미지로 보임
- 컴퓨터는 이미지를 픽셀 벡터의 형태로 표현하고 인식
  - 원본 이미지와 다른 것으로 인식



좌우대칭(Horizontal Flip)

- 왼쪽만 바라보는 고양이 사진 70개를 넣어주면 오른쪽을 보는 고양이는 식별 불가능하게 됨
- 좌우대칭을 시켜주면 어느 쪽을 보더라도 맞출 수 있음

랜덤 크랍(Random Crop)

- 확률적으로 고양이를 꼬리를 보고 50%, 귀를 보고 30%로 판단한다고 할 때, 고양이가 상자 속에 들어가서 꼬리만 있는 사진을 사람은 꼬리만 보고도 고양이라고 판단할 수 있음
  - 가려짐(Occlusion)에 대응 가능
- Random crop을 한 이미지를 학습하면 각 부분만 보고도 고양이로 판단 가능
- 주의 사항
  - 오검출률(False positive rate)이 높아질 수 있으므로 상황에 따라 적절한 조절이 필요

밝기 조절(Brightness Change)

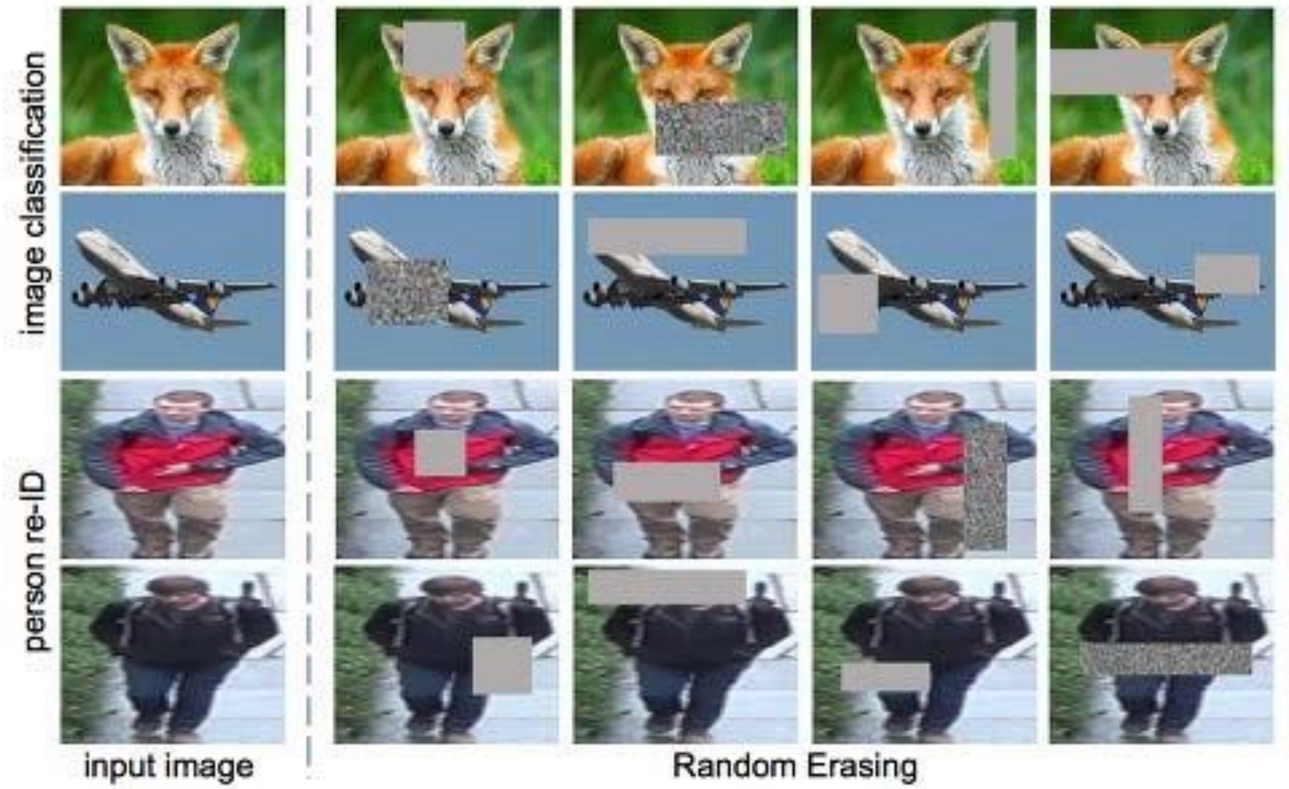
- 조명이나 빛의 반사 등에 의해 밝기가 변해도 모델이 인식 가능

크기변경(Rescale)

- 이미지의 크기가 바뀌어도 모델이 같은 이미지로 인식 가능
- 확대축소(Zoom)는 Crop후 Rescale하면 가능

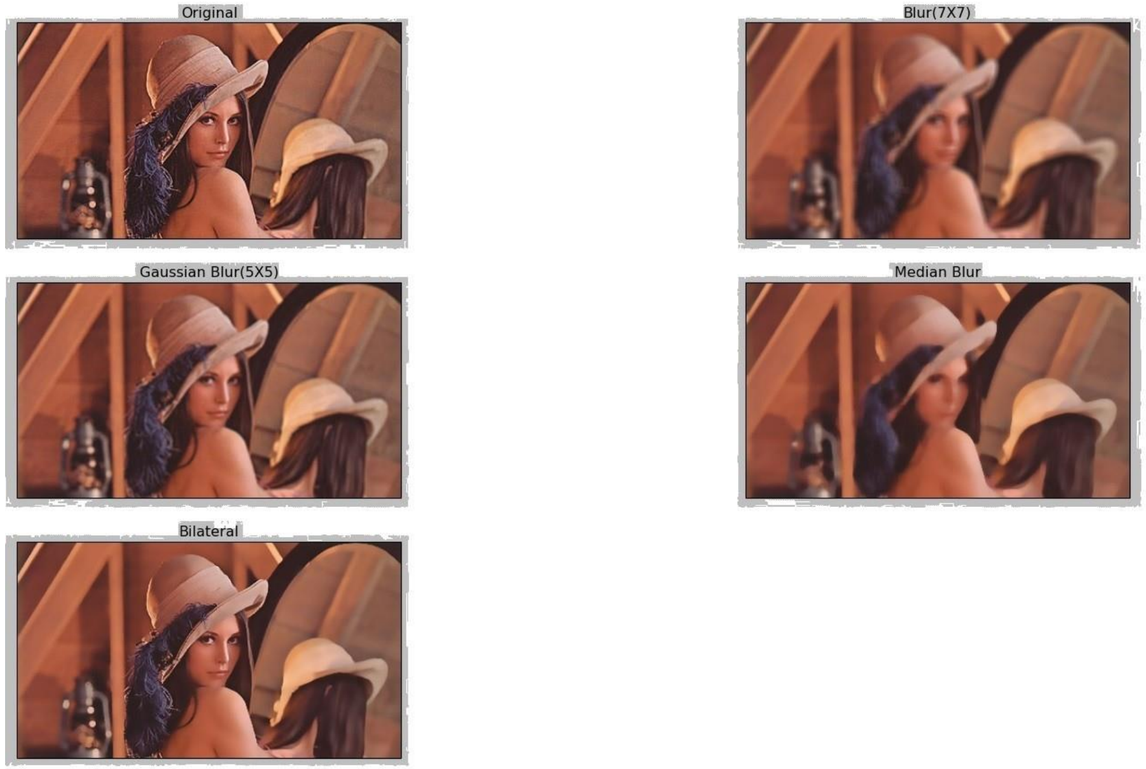
일부 지우기(Random Erasing)

- 가려짐(Occlusion)에 대응 가능



블러(Blurring)

- Gaussian blur, Bilateral blur, Median blur 등 사진을 흐리게 하는 많은 기법들도 모델의 학습에 도움이 됨

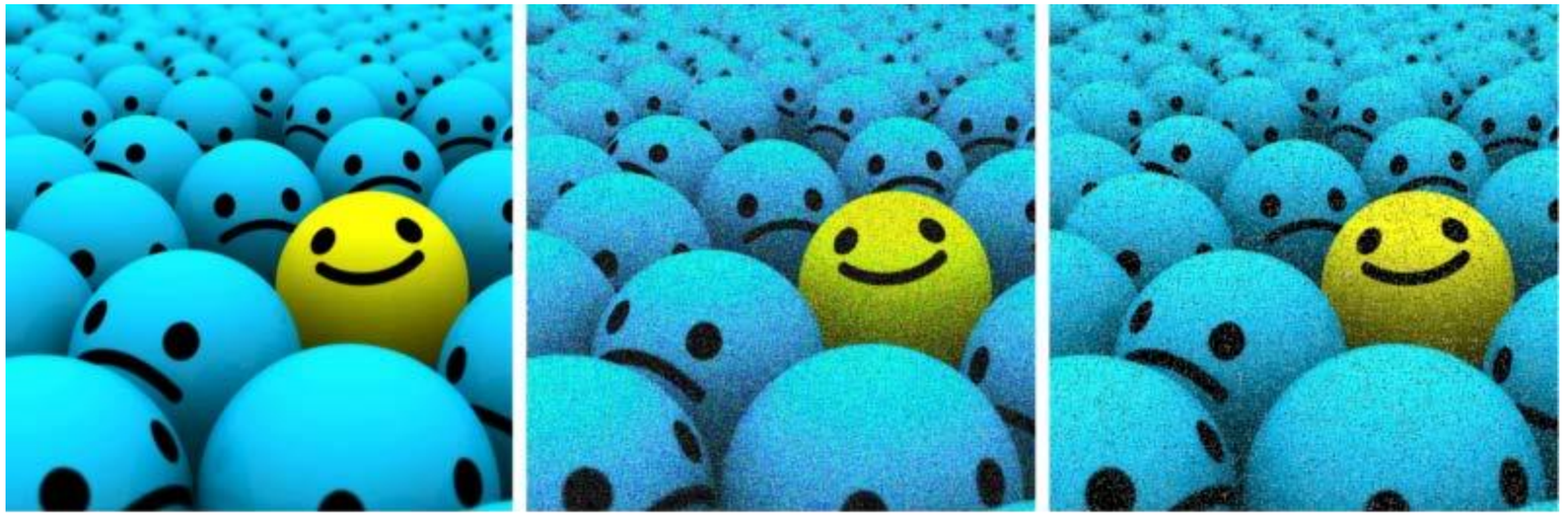


컬러 노이즈(Color Noise)

- Alex Krizhevsky가 AlexNet에서 Overfitting을 방지하기 위해 사용
- 각 RGB 칼럼의 principal vector, principal eigenvalue에 약간의 random 요소를 더해서 RGB를 미세하게 같은 방향, 크기로 변화시키는 방법 (PCA 사용)

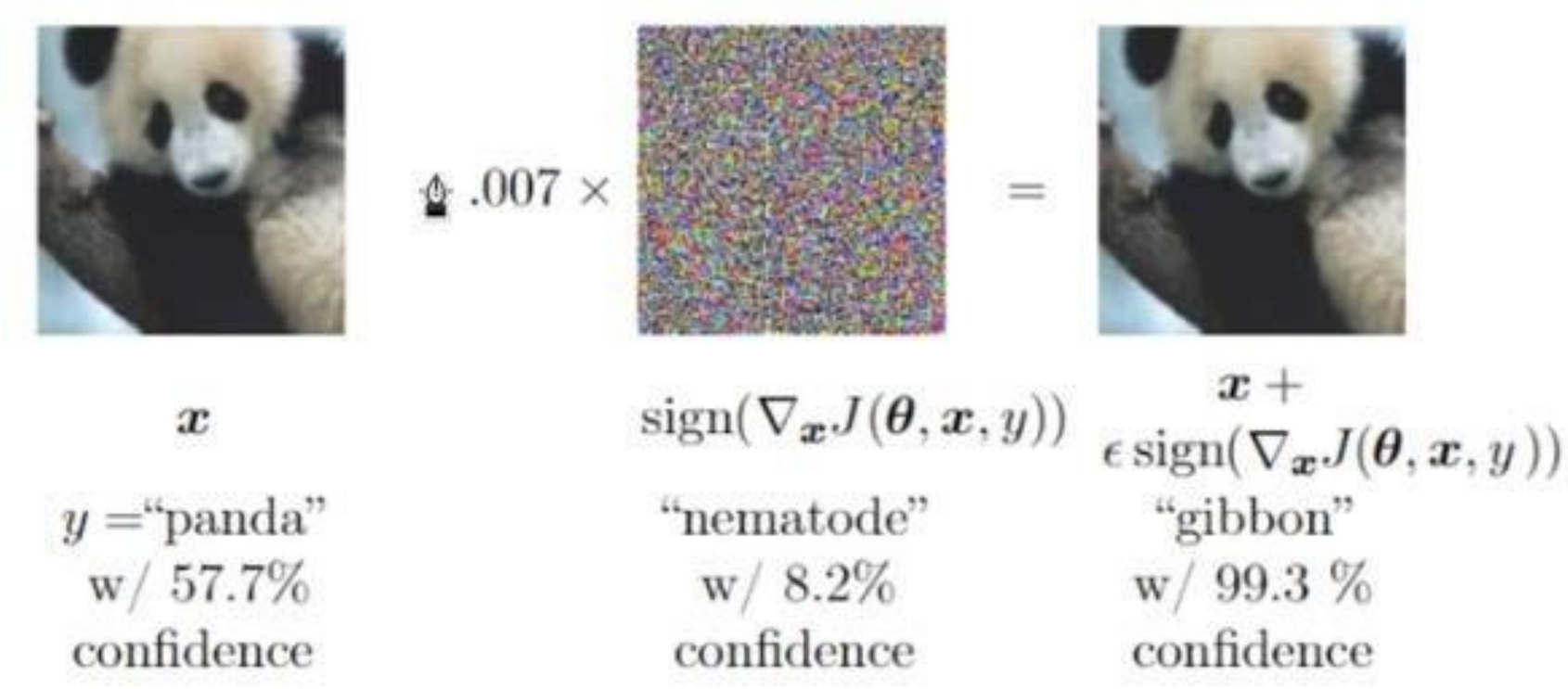
랜덤 노이즈(Random Noise)

- Gaussian noise 등의 노이즈를 입힐 수도 있음



랜덤 노이즈(Random Noise)

- Adversarial attack와 같은 현상도 있으므로 주의가 필요



Credit

/* elice */

코스 매니저

콘텐츠 제작자
정민수

강사
정민수

감수자

디자인

Contact

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io