Training Neural Networks

주재걸 교수

KAIST 김재철AI대학원

DAVIAN

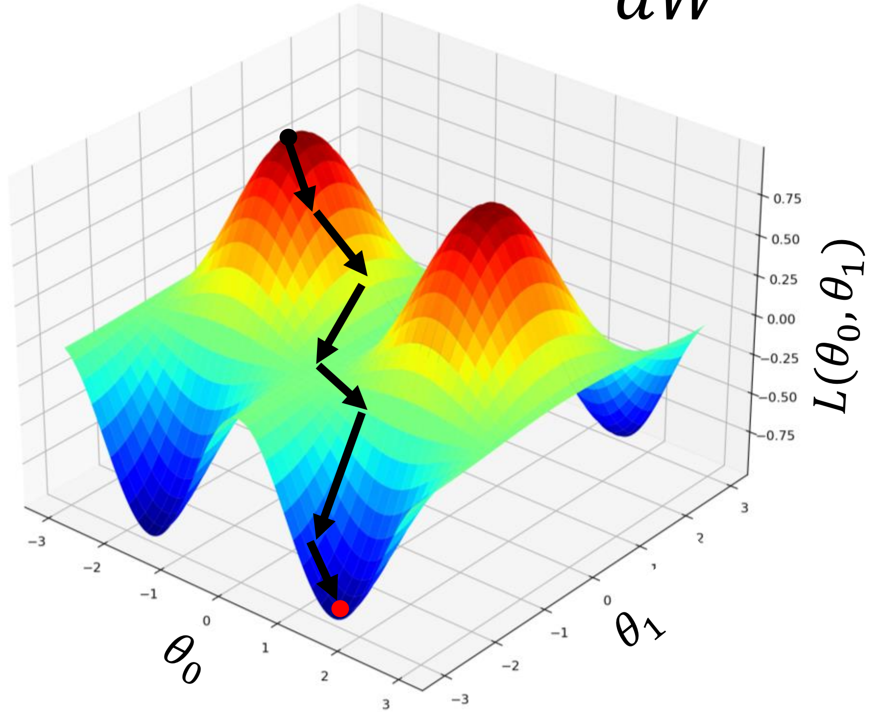
Data and Visual Analytics Lab

Training Neural Networks via Gradient Descent

loss function

Given the optimization problem, $\boxed{\min_{W} L(W)}$, where W is the neural network parameters, we optimize W using gradient descent approach:

$$W := W - \alpha \frac{dL(W)}{dW}$$

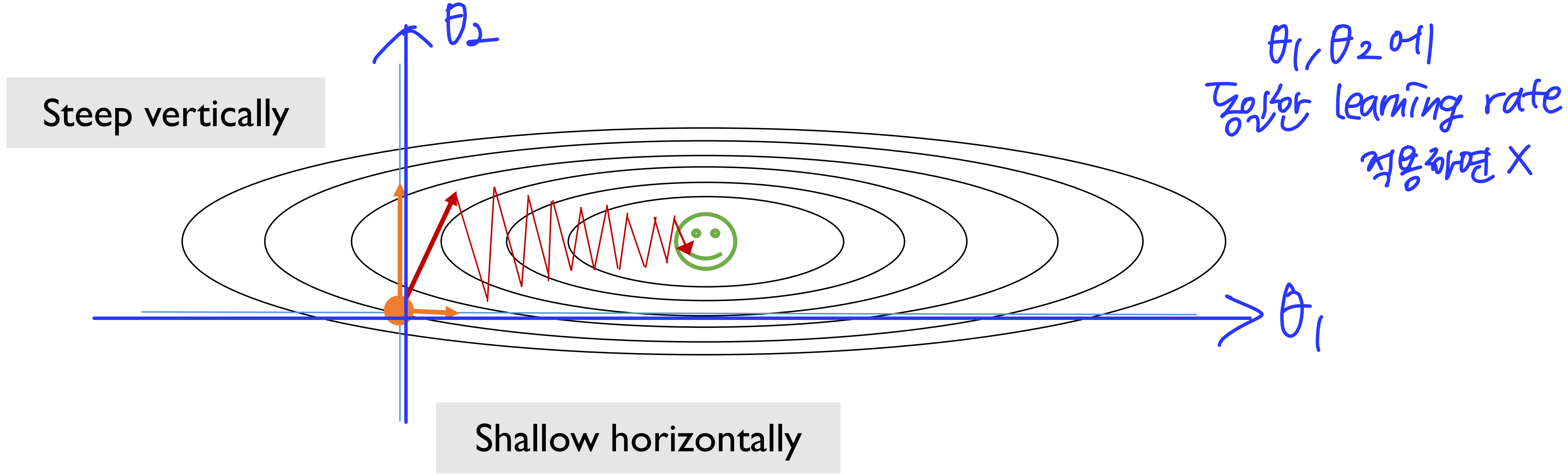


Poor Convergence Case of Naïve Gradient Descent

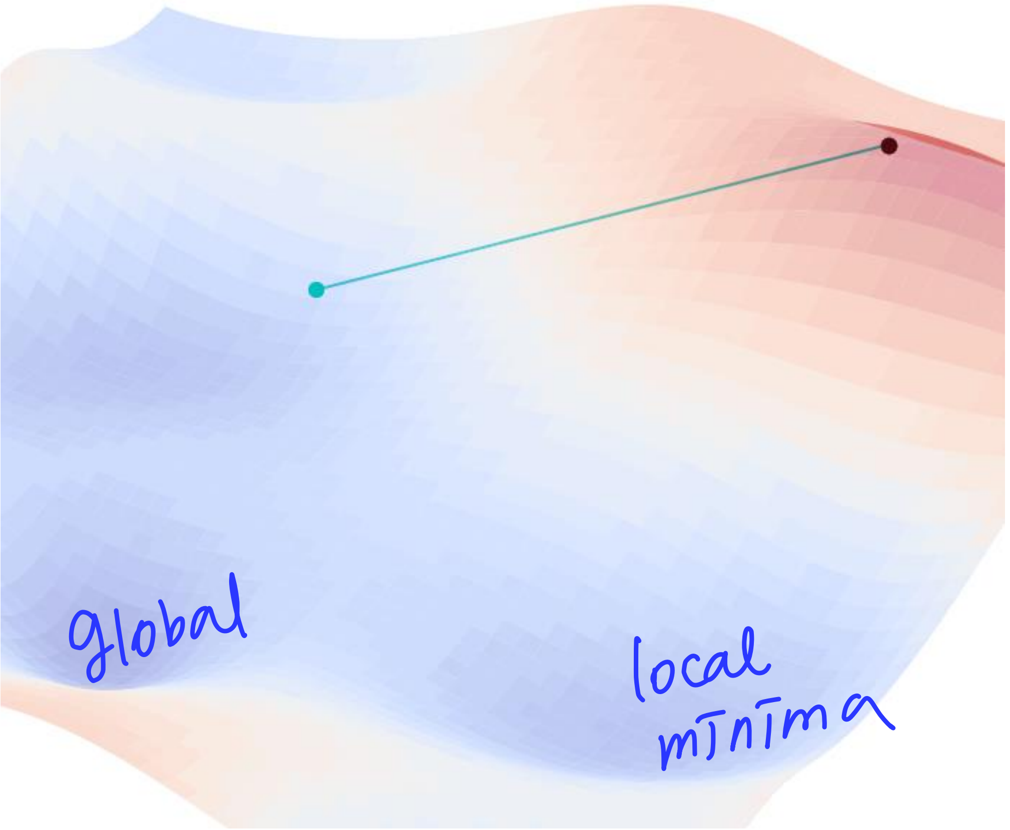
Suppose loss function is steep vertically but shallow horizontally:

Q: What is the trajectory along which we converge towards the minimum with SGD?

Very slow progress along flat direction, jitter along steep one



Various Gradient Descent Methods



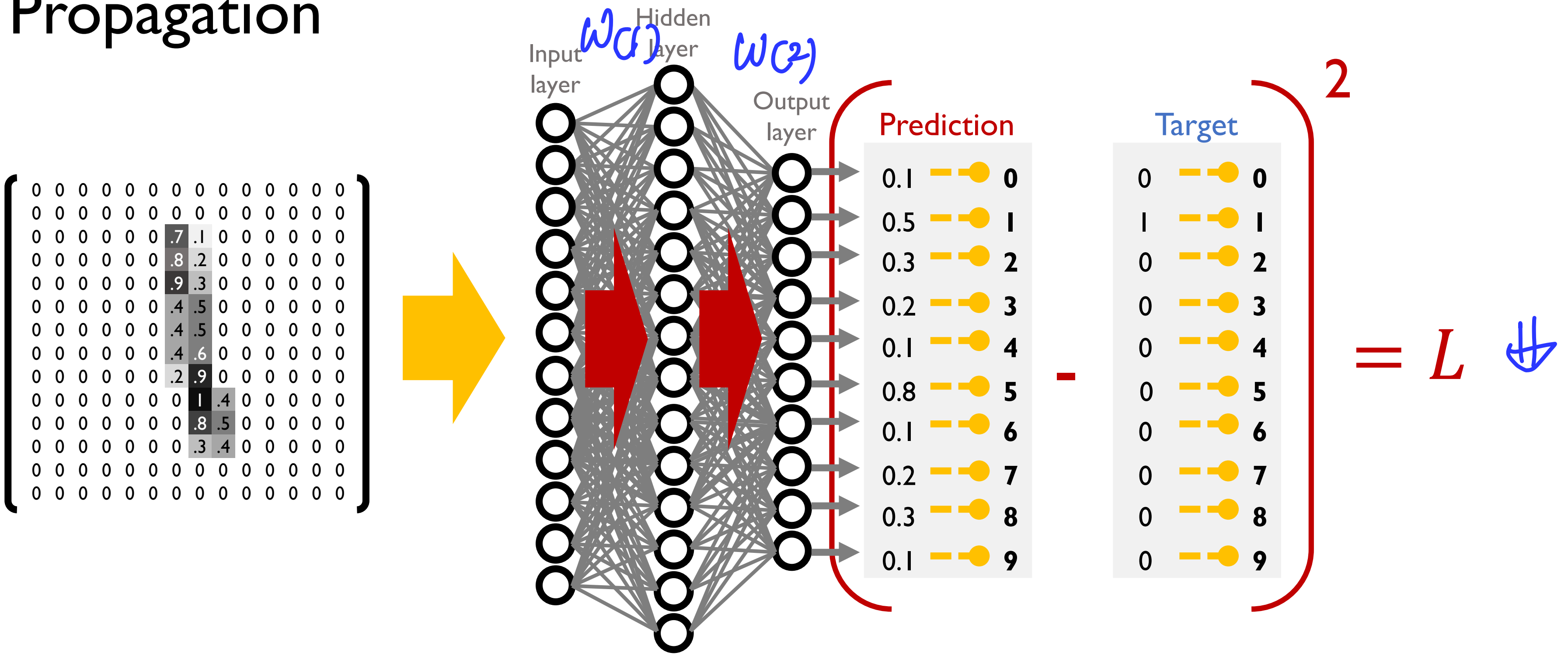
- GD
- Momentum
- Adagrad
- RMSProp
- Adam 갑!!
- Adadelta[1]
- Ftrl[2]

http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

[1] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. http://arxiv.org/abs/1212.5701
[2] H. Brendan McMahan et. al., (2013). Ad Click Prediction: a View from the Trenches, KDD

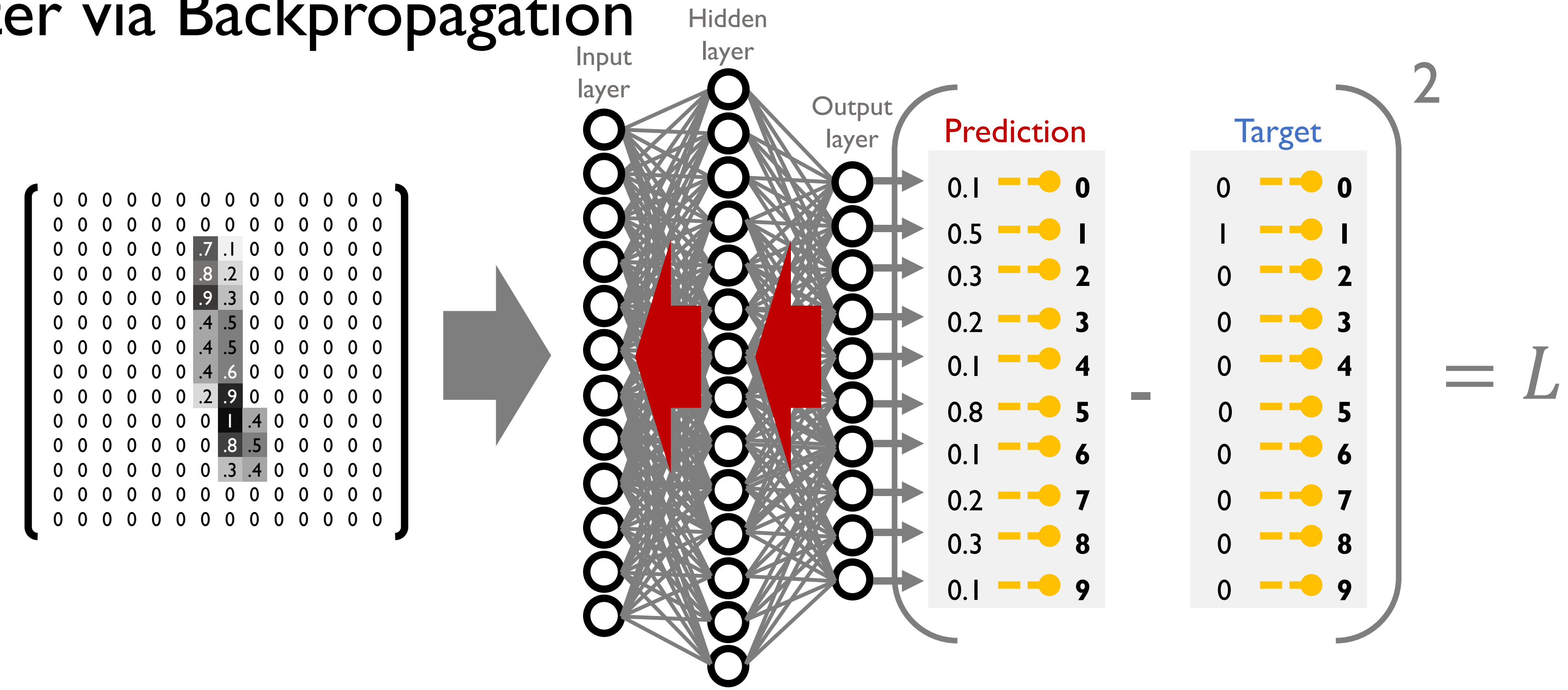
Backpropagation to Compute Gradient in Neural Networks

First, given an input data item, compute the loss function value via Forward Propagation



Backpropagation to Compute Gradient in Neural Networks

Afterwards, compute the gradient with respect to each neural network parameter via Backpropagation

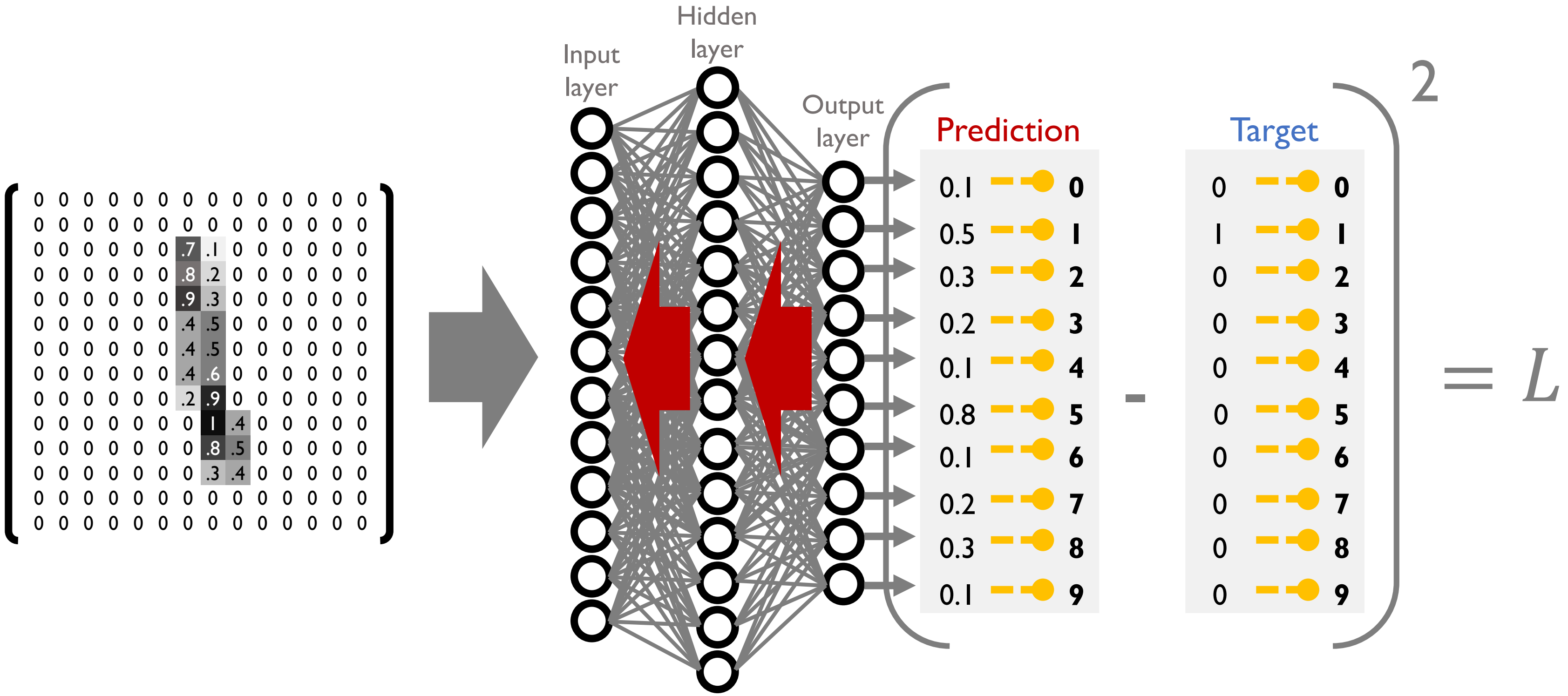


$$\frac{\partial L}{\partial W^{(1)}} \qquad \frac{\partial L}{\partial W^{(2)}}$$

back propagation
손실함수 편미분해서 update.

Backpropagation to Compute Gradient in Neural Networks

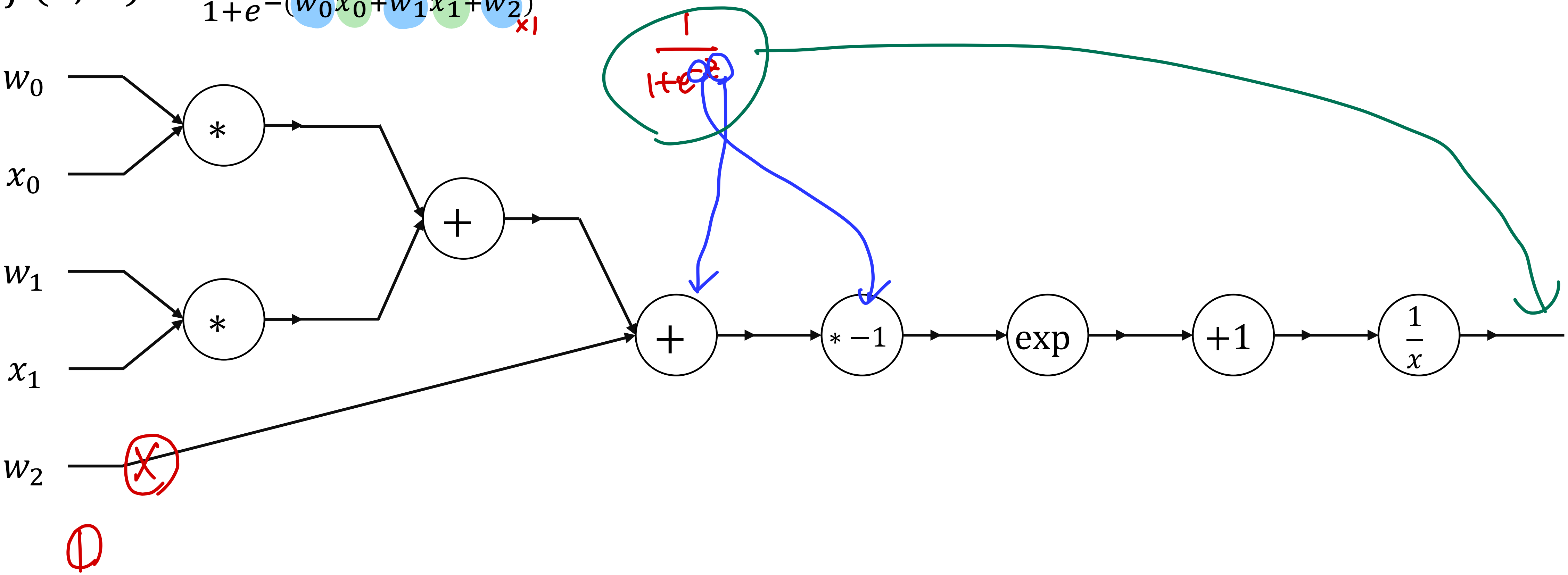
Finally, update the parameters using gradient descent algorithm



$$W^{(1)} := W^{(1)} - \alpha \frac{\partial L}{\partial W^{(1)}} \qquad W^{(2)} := W^{(2)} - \alpha \frac{\partial L}{\partial W^{(2)}}$$

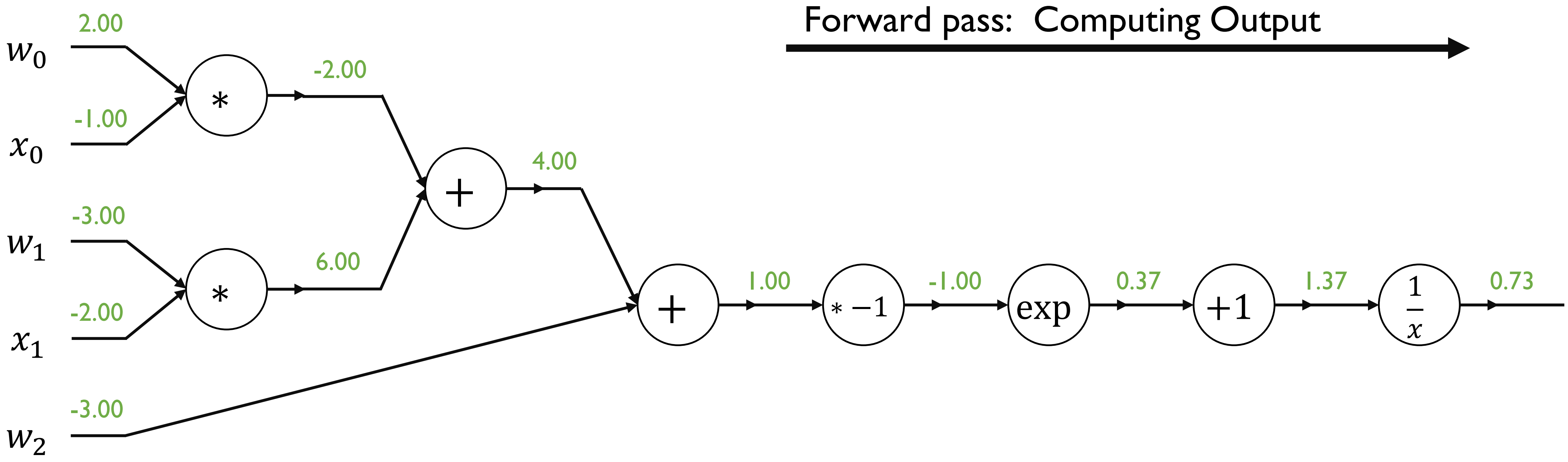
Computational Graph of Logistic Regression

$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Forward Propagation of Logistic Regression

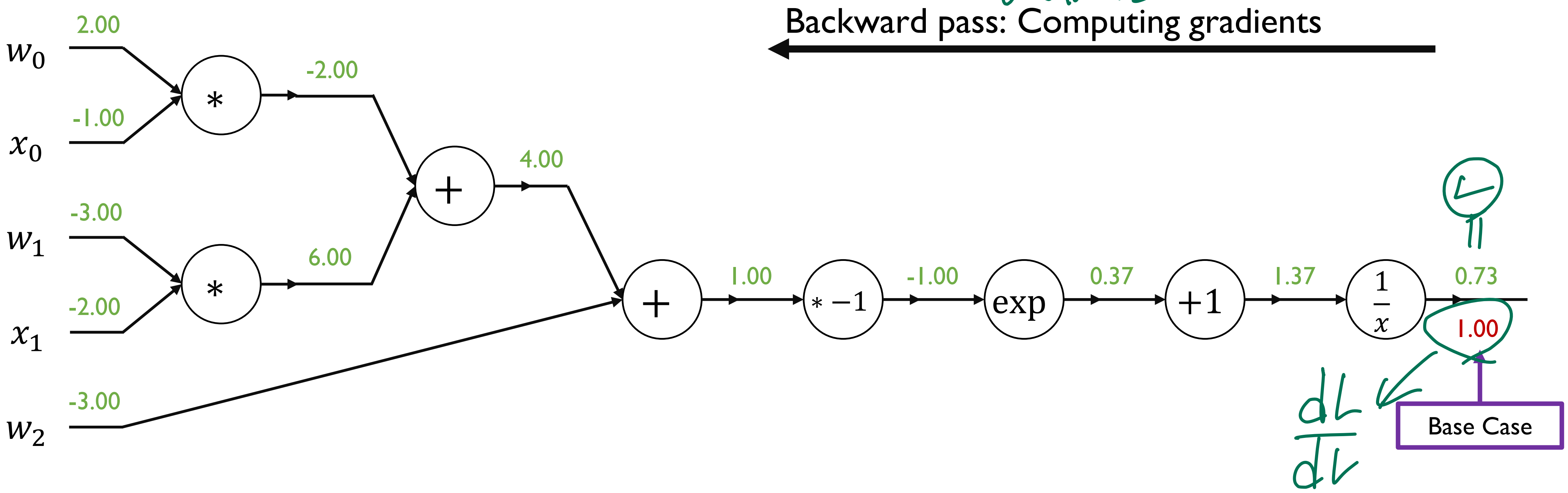
$$f(x,w) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



Backpropagation of Logistic Regression

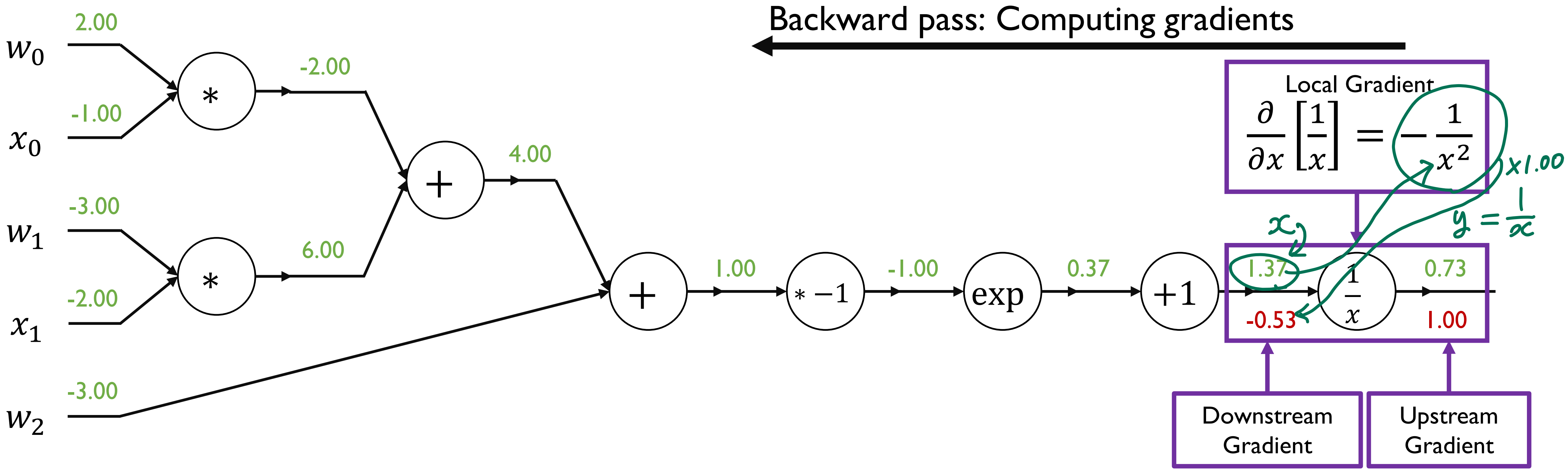
$$f(x, w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

편미분 값 구함.



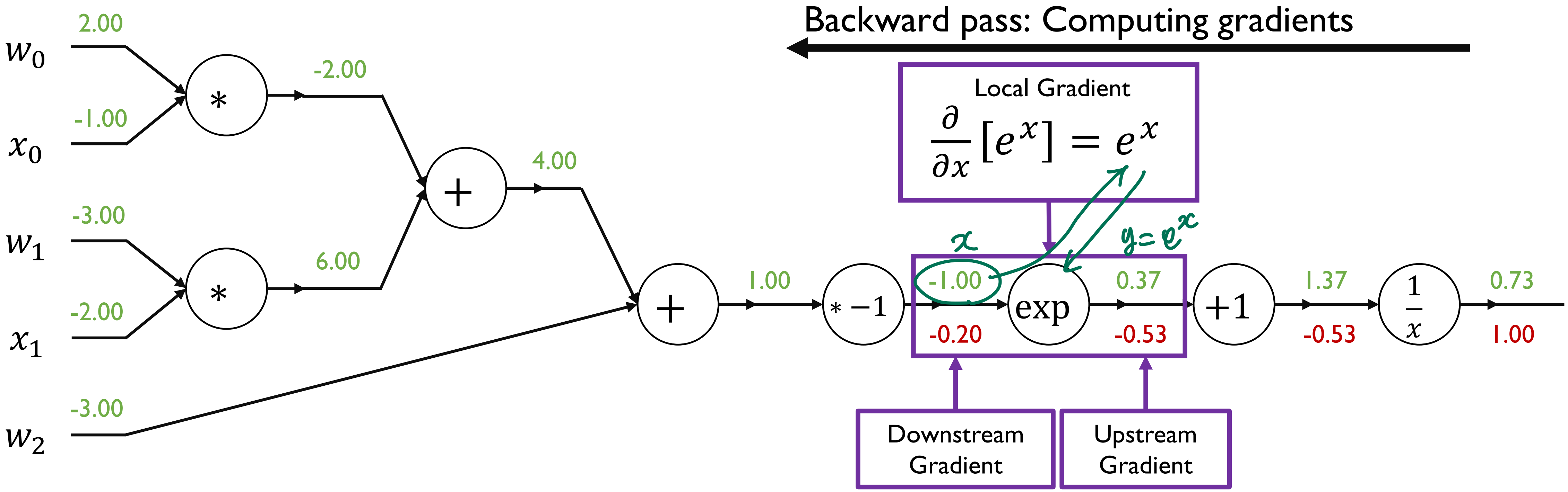
Backpropagation of Logistic Regression

$$f(x, w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



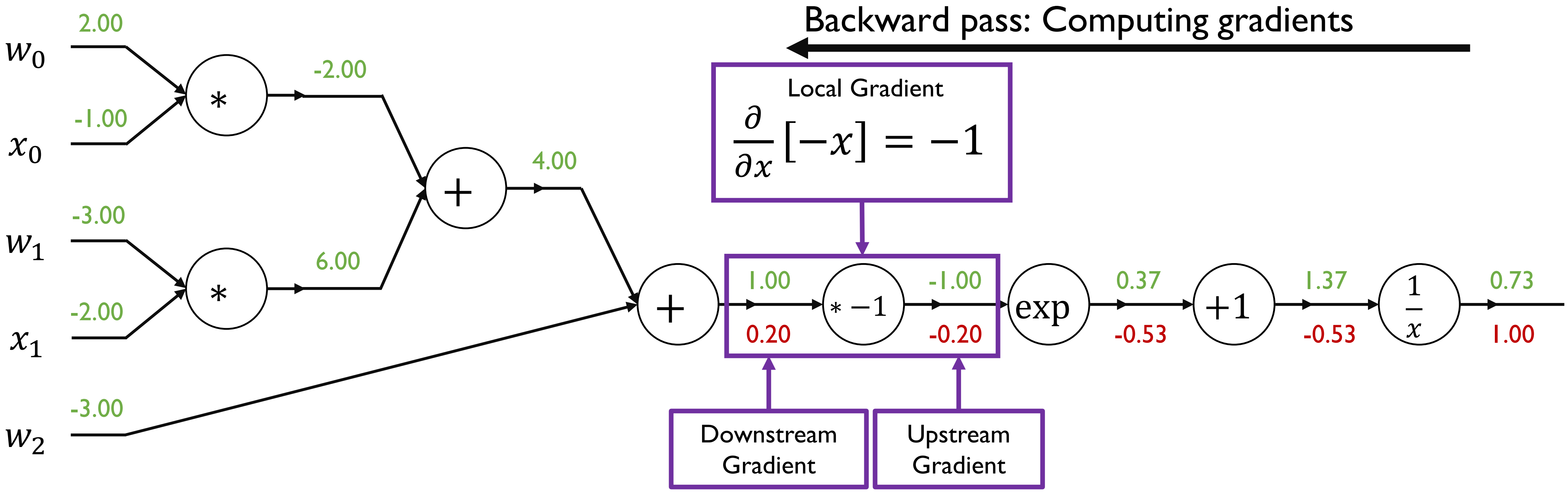
Backpropagation of Logistic Regression

$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



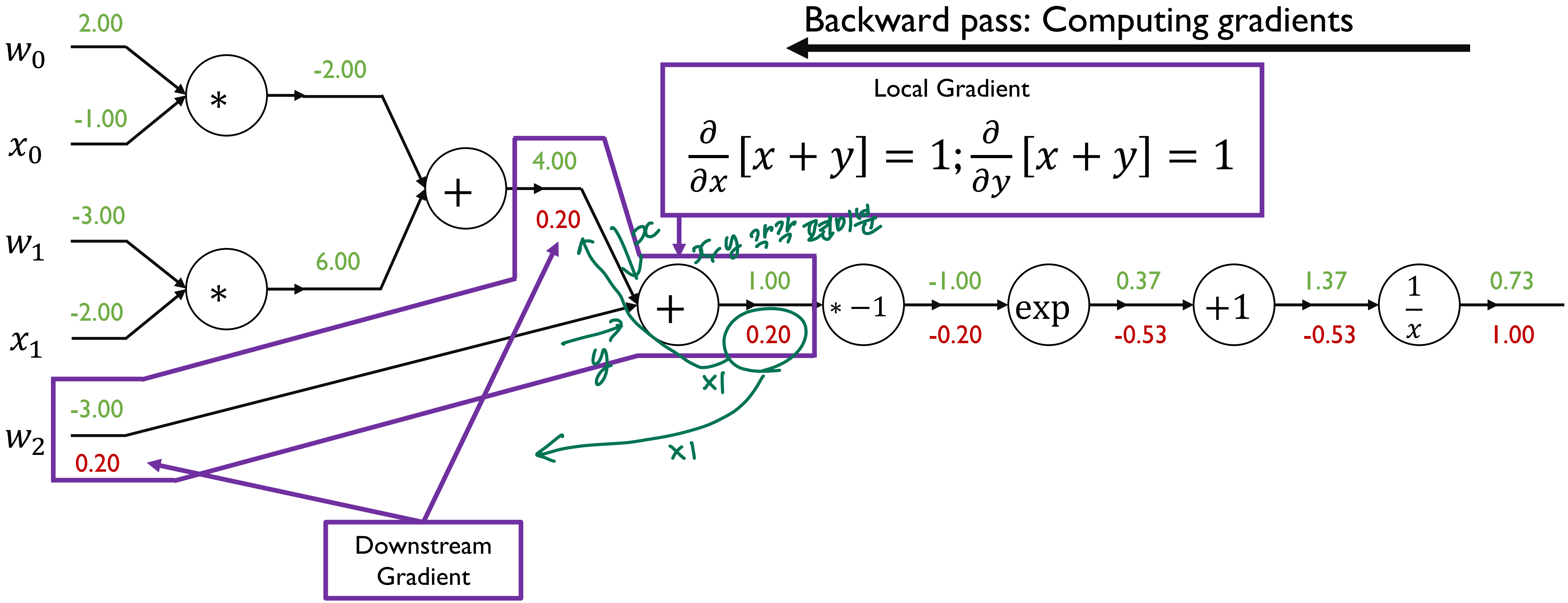
Backpropagation of Logistic Regression

$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



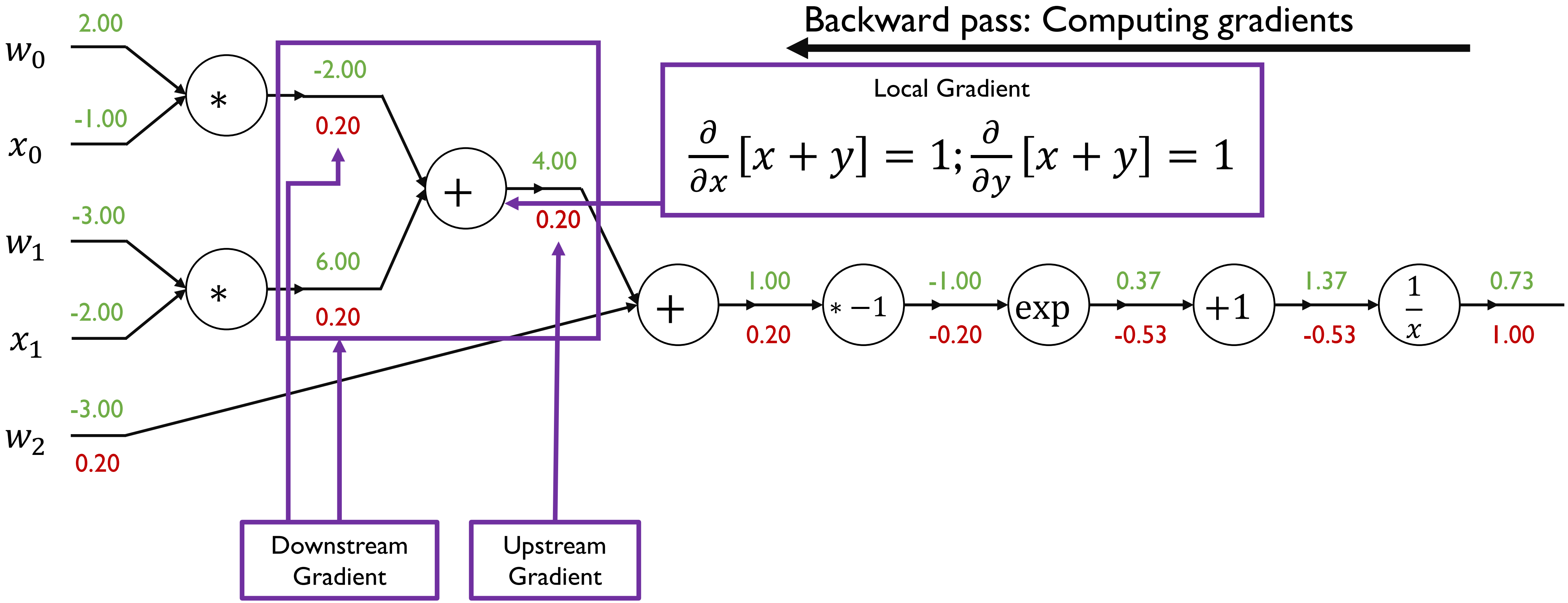
Backpropagation of Logistic Regression

$$f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



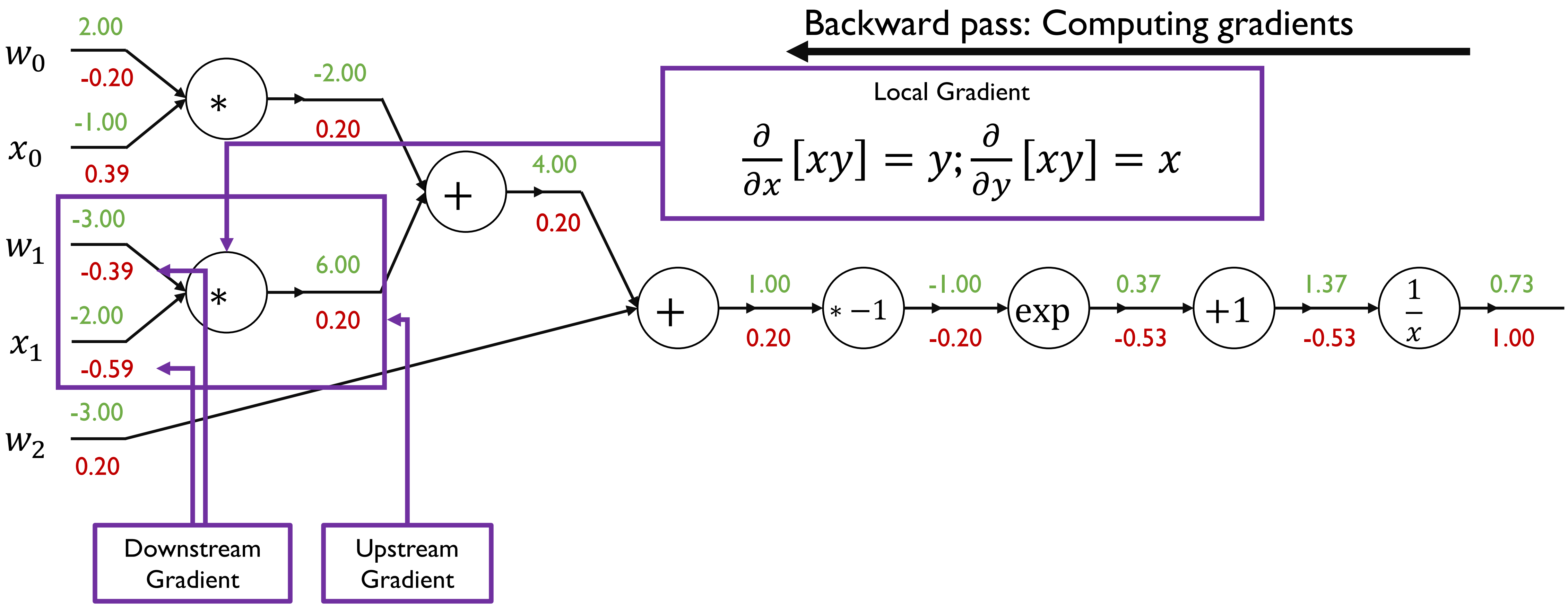
Backpropagation of Logistic Regression

다른 예시: $f(x,w) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



Backpropagation of Logistic Regression

다른 예시: $f(x,w) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

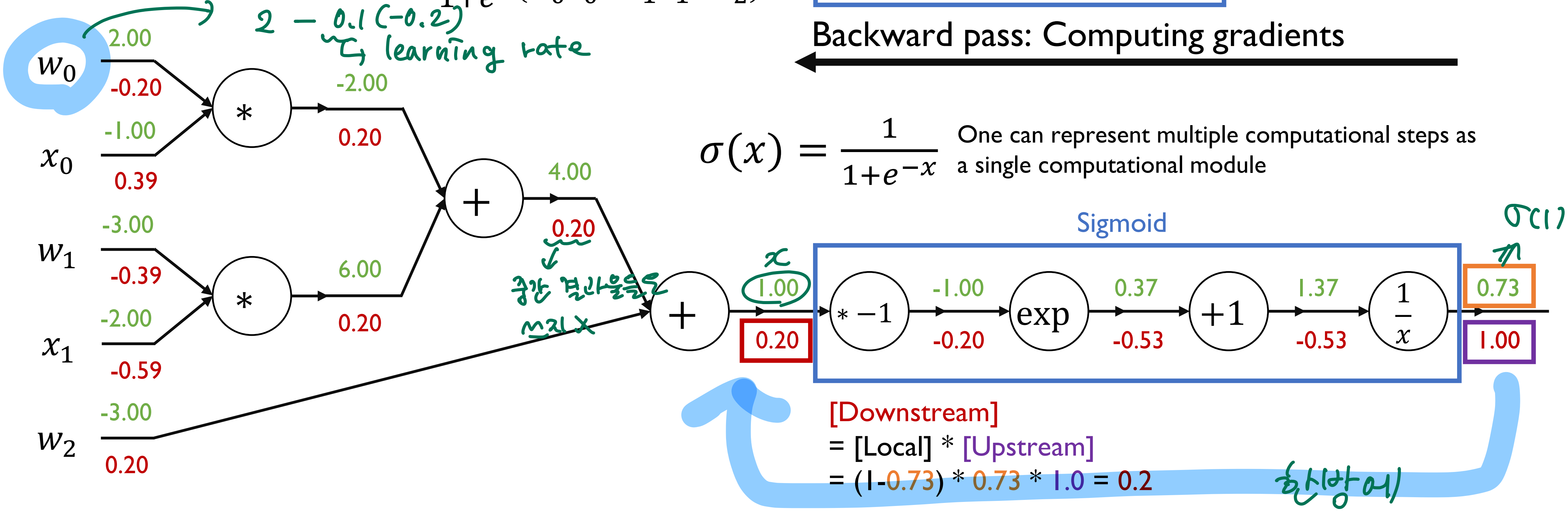


Backpropagation of Logistic Regression

다른 예시: $f(x,w) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} = \boxed{\sigma(w_0x_0 + w_1x_1 + w_2)}$

Backward pass: Computing gradients

$\sigma(x) = \dfrac{1}{1+e^{-x}}$ One can represent multiple computational steps as a single computational module

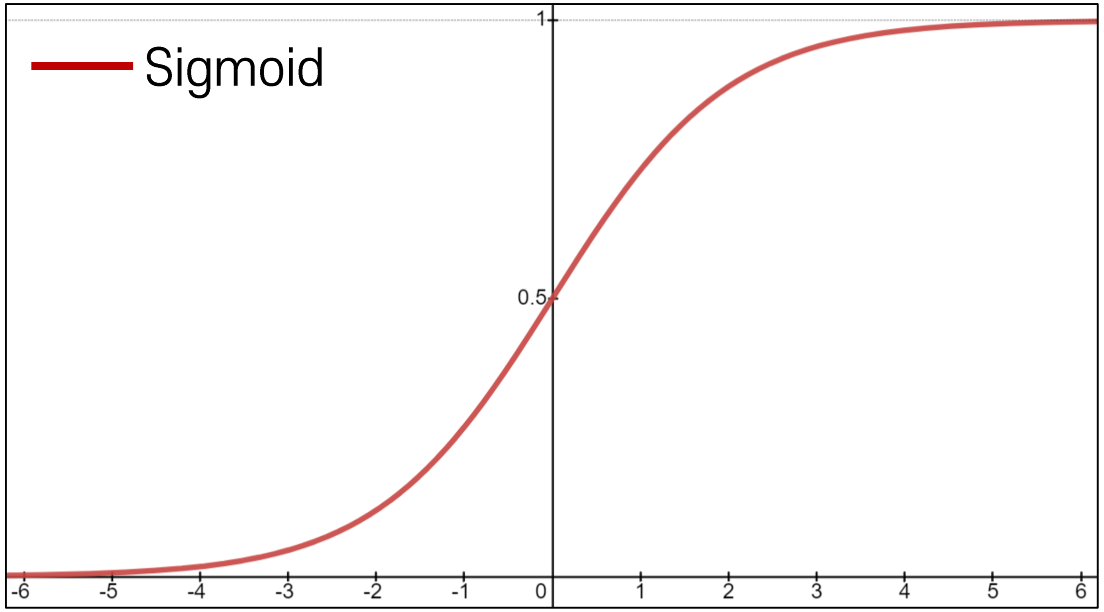


[Downstream]
= [Local] * [Upstream]
= (1-0.73) * 0.73 * 1.0 = 0.2

Sigmoid local gradient $\dfrac{\partial}{\partial x}[\sigma(x)] = \dfrac{e^{-x}}{(1+e^{-x})^2} = \left(\dfrac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\dfrac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$

↳ (1 − 0.73) 0.73

Sigmoid Activation



Sigmoid

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

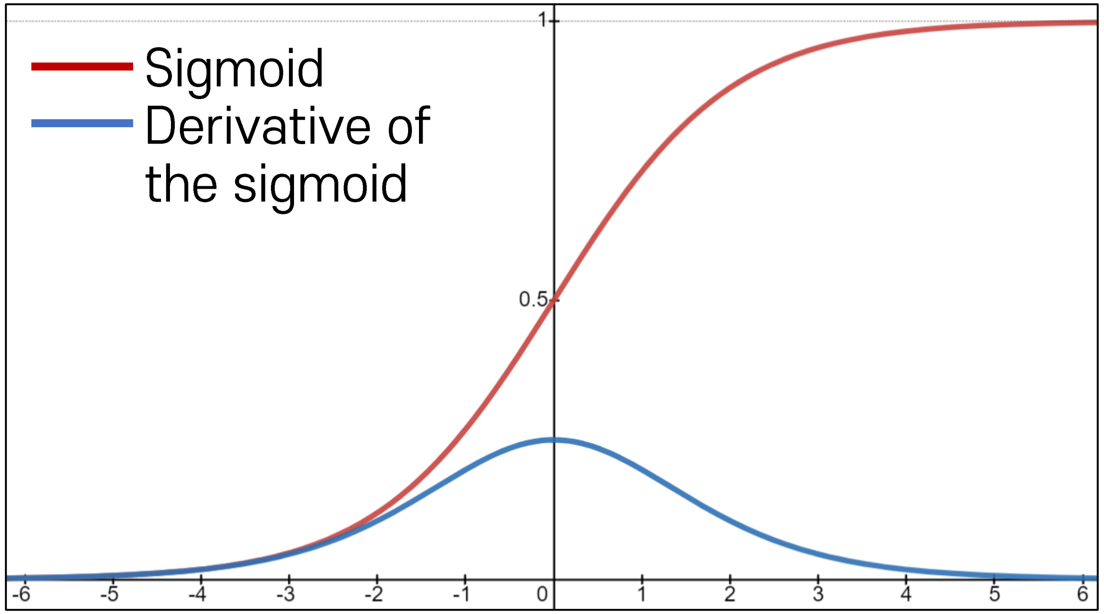
- Maps real numbers in $\boxed{(-\infty, \infty)}$ into a range of $\boxed{[0, 1]}$
- gives a probabilistic interpretation

확률값으로 해석ㅇ가

Historically, sigmoid activation function gives nice interpretation of saturating firing rate of a neuron

Problems of Sigmoid Activation

$0 < \overset{\boxed{0\sim1}}{\sigma(x)} \cdot (1-\sigma(x)) \le \frac{1}{4}$



Sigmoid

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

- Saturated neurons kills the gradients
- The gradient value $\sigma(x) \le \frac{1}{4}$, which decreases the gradient during backpropagation, i.e., causing a gradient vanishing problem

back propagation 계속 수행시
gradient 점차 작아짐.

점점
update 이뤄지지 x
⇓
$\boxed{\text{gradient vanishing}}$

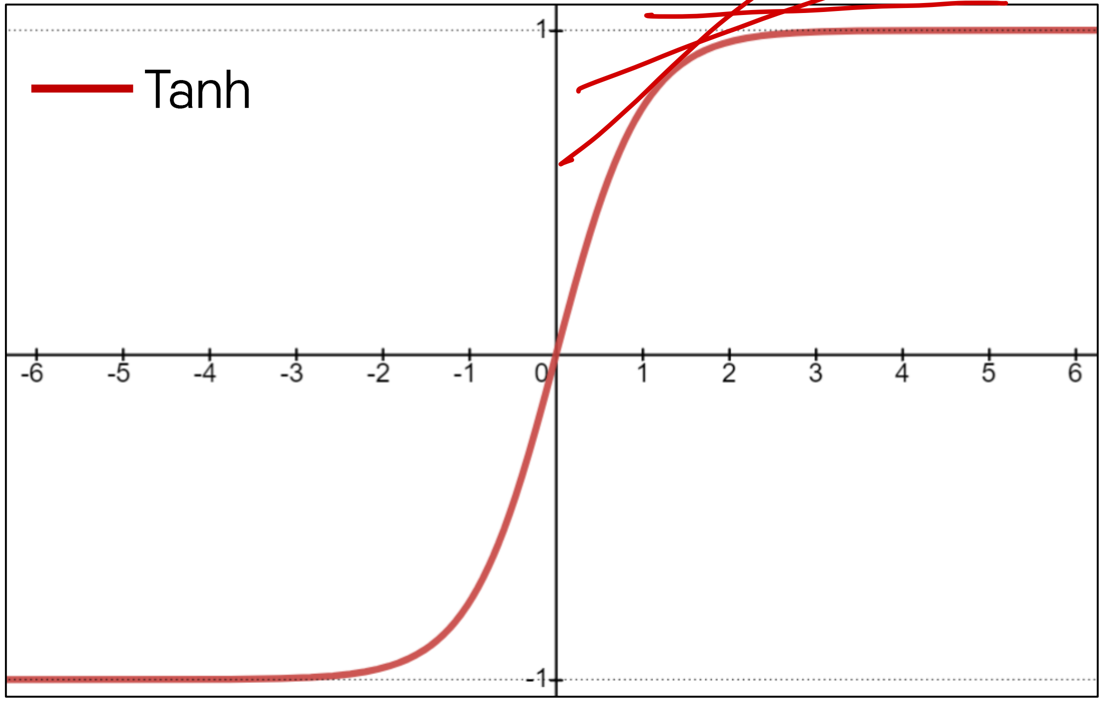
$x \rightarrow$ Sigmoid $\rightarrow \sigma(x)$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial x} \quad \longleftarrow \quad \frac{\partial L}{\partial \sigma}$$

Tanh Activation

$-\infty, \infty \Rightarrow -1, 1$

$=0$



Tanh

- $\tanh(x) = 2 \times \text{sigmoid}(x) - 1$
- Squashes numbers to range [-1, 1]

Strength

학습 더 빠른 효과있음.

- Zero-centered (average is 0)

Weakness

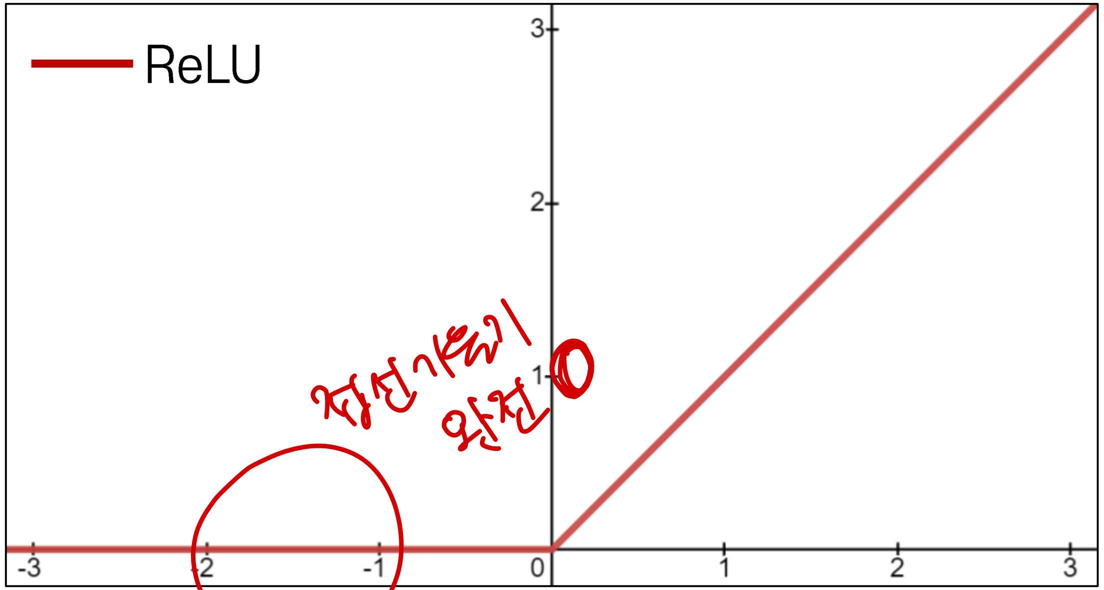
- Still kills gradients when saturated, i.e., still causing a gradient vanishing problem

$\boxed{0 \sim \frac{1}{2}}$ 임. → gradient vanishing 미세히 줄어.

ReLU Activation

layer 쌓면 good?

$-\infty, \infty \Rightarrow 0, \infty$



The slope of the function
$x \geq 0$: 1 (bypass)
$x < 0$: 0 (gating)

ReLU (Rectified Linear Unit)

- $f(x) = \max(0, x)$

↳ 연산 간단

Strength

- Does not saturate (in + region)
- Very computationally efficient
- Converge much faster than sigmoid/tanh

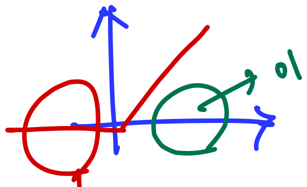
Weakness

- Not zero-centered output
- Gradient is completely zero for $x < 0$

Batch Normalization

Motivation of Batch Normalization

- Saturated gradients when random initialization is done
- The parameters are not updated → Hard to optimize (in red region)

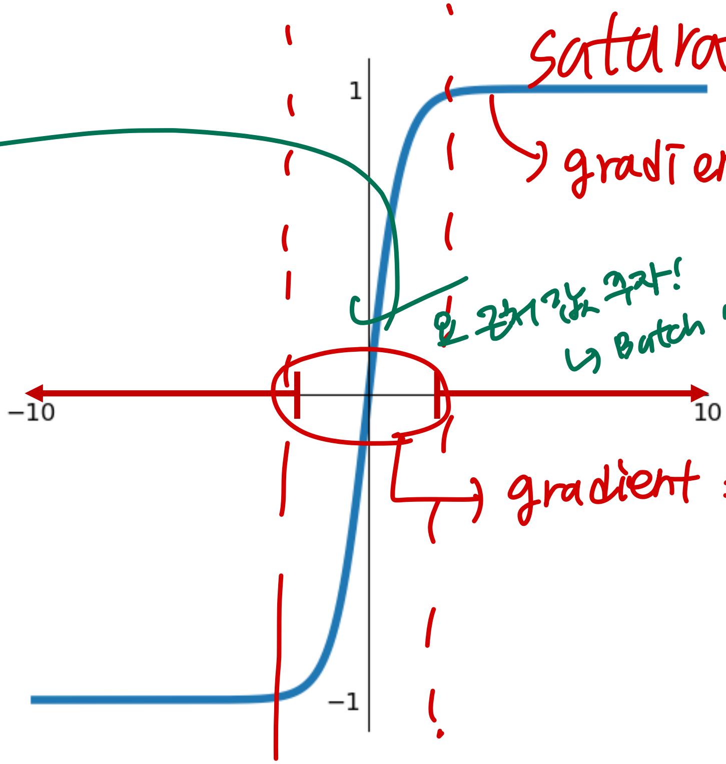


이 값들이 없어야 함.

다양한 data item 필요

거의 0 이면 학습 불가능.

Tanh activation



saturated 영역

gradient = 0

요 근처 값 주자!

↳ Batch normalization 아이디어!

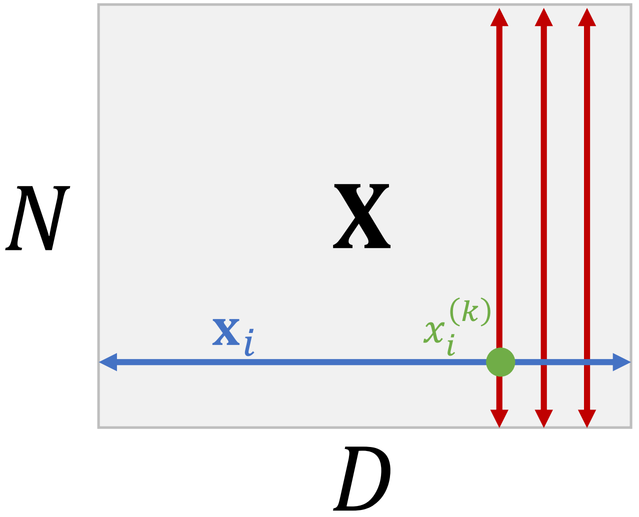
gradient 크게 줌

Definition of Batch Normalization

"You want unit Gaussian activations? just make them so."

- We consider a batch of activations at some layer to make each dimension unit Gaussian

1. Compute the empirical mean $\mathbb{E}[x^{(k)}]$ and variance $\mathrm{Var}[x^{(k)}]$ independently for each dimension k



2. Normalize

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

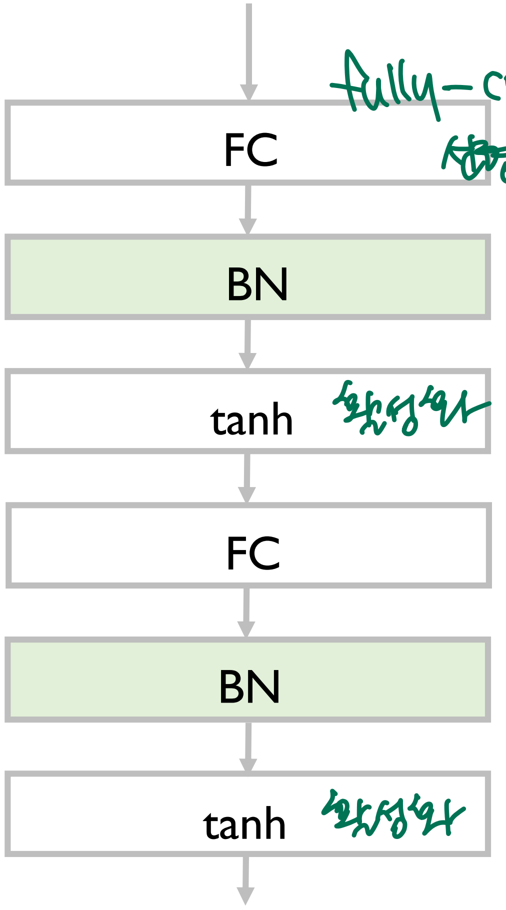
tanh

정규분포 따르게.

This is a vanilla differentiable function

Ioffe, Sergey, and Christian, Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.". In Proceedings of the 32nd International Conference on Machine Learning (pp. 448–456). PMLR, 2015.

Batch Normalization Process



fully-connected layer는 선형결합 수행한 이후기 BN 추가함.

Usually inserted
after fully-connected or convolutional layers,
and before nonlinearity.

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Problem:
Do we necessarily want a unit Gaussian input to a tanh layer?

무조건 정규분포 따르게 하는건 neural network가 잘 추출한 정보 무시하는 과정일 수 있음.

Ioffe, Sergey, and Christian, Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.". In Proceedings of the 32nd International Conference on Machine Learning (pp. 448–456). PMLR, 2015.

Batch Normalization

기울기 소실 문제 해결 할 수 있는 장치.

Normalize:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

ax+b

$$y_i^{(k)} = \gamma^{(k)} \hat{x}_i^{(k)} + \beta^{(k)}$$

원하는 정보로 복원

→ 평균 b, 분산 a^2 으로 바꿈.

원래 평균 분산으로 복원 가능.

Note, the network can learn:

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

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

to recover the identity mapping

← 평균 분산 스스로 결정하도록

Determined while training neural network

Ioffe, Sergey, and Christian, Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.". In Proceedings of the 32nd International Conference on Machine Learning (pp. 448–456). PMLR, 2015.

Batch Normalization

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

$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // mini-batch mean

$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$ // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ // normalize

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

1 0 초기값.

- Improves gradient flow through the network
- Reduces the strong dependence on initialization

Ioffe, Sergey, and Christian, Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.". In Proceedings of the 32nd International Conference on Machine Learning (pp. 448–456). PMLR, 2015.

THANK YOU