深度學習實作與應用
Deep learning and its applications

2. Basic Neural Networks (3/3)

IM5062, Spring 2024

黃意婷

Outline

- Recap Loss functions
- Gradient descent
- Backward Propagation

Cross-entropy loss

- Binary cross-entropy loss

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -(1-y_i)\log\left[1-\text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right] - y_i \log\left[\text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right]$$

- Multiclass cross-entropy loss

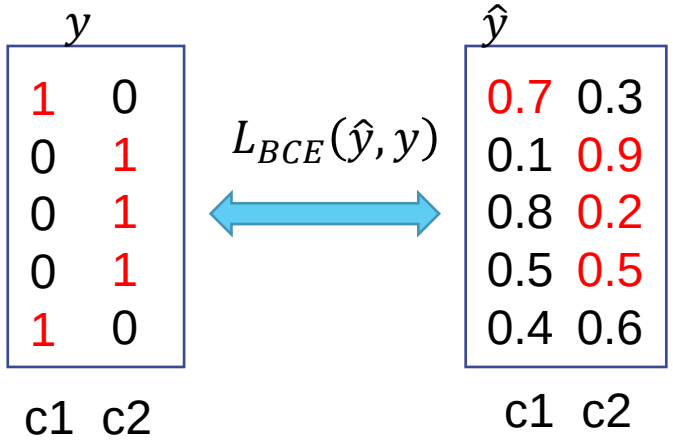
$$\begin{aligned} L[\boldsymbol{\phi}] &= -\sum_{i=1}^{I} \log\left[\text{softmax}_{y_i}\left[\mathbf{f}\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right] \\ &= -\sum_{i=1}^{I} \text{f}_{y_i}\left[\mathbf{x}_i, \boldsymbol{\phi}\right] - \log\left[\sum_{k=1}^{K} \exp\left[\ \text{f}_k\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right] \end{aligned}$$

Binary cross-entropy loss with one observation

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} -(1-y_i)\log\left[1-\text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right] - y_i \log\left[\text{sig}[\text{f}[\mathbf{x}_i|\boldsymbol{\phi}]]\right]$$

- Only with one observation:

$$L_{BCE}(\hat{y}, y) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$



Let’s see if this works for our sentiment example

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

Let’s first suppose the true label of this is y = 1 (positive)

It's hokey. There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

$x_1 = 3$: count(positive lexicon ∈ doc)

$x_2 = 2$: count(negative lexicon ∈ doc)

$$x_3 = 1: \begin{cases} 1 & \text{if } ''\text{no}'' \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$$

$x_4 = 3$: count(1st and 2nd pronouns ∈ doc)

$$x_5 = 0: \begin{cases} 1 & \text{if } ''!'' \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$$

$x_6 = \ln(66) = 4.19$: log(word count of doc)

Let's see if this works for our sentiment example

$$L_{BCE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

- True value is y=1. How well is our model doing?

$$\begin{aligned} p(+|x) &= P(Y=1|x) = \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\ &= \sigma(.833) \\ &= 0.70 \end{aligned}$$

- Pretty well!! What's the loss?

$$\begin{aligned} L_{CE}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))] \\ &= -[\log \sigma(w \cdot x + b)] \\ &= -\log(0.70) \\ &= 0.36 \end{aligned}$$

Let's see if this works for our sentiment example

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

- Suppose true value instead was y=0

$$p(-|x) = P(Y=0|x) = 1 - \sigma(w \cdot x + b)$$
$$= 0.30$$

- What's the loss?

$$L_{BCE}(\hat{y}, y) = -[\mathrm{y} \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

$$= -[\log \sigma(1 - w \cdot x + b)]$$

$$= -\log(0.30)$$

$$= 1.2$$

Let's see if this works for our sentiment example

- The loss when model was right (if true y = 1)

$$
\begin{aligned}
L_{BCE}(\hat{y}, y) &= -[\mathrm{y}\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
&= -[\log\sigma(w\cdot x+b)] \\
&= -\log(0.70) \\
&= 0.36
\end{aligned}
$$

- Is lower than the loss when model was wrong (if true y=0):

$$
\begin{aligned}
L_{BCE}(\hat{y}, y) &= -[\mathrm{y}\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\
&= -[\log\sigma(1-w\cdot x+b)] \\
&= -\log(0.30) \\
&= 1.2
\end{aligned}
$$

- See enough, loss was bigger when model was wrong!

Multiclass cross-entropy loss

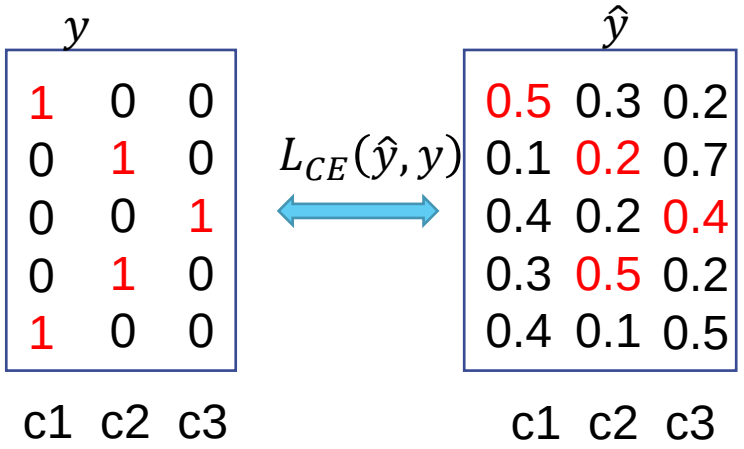
$$L[\boldsymbol{\phi}] = -\sum_{i=1}^{I} \log\left[\text{softmax}_{y_i}\left[\mathbf{f}\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right]$$

$$= -\sum_{i=1}^{I} \mathrm{f}_{y_i}\left[\mathbf{x}_i, \boldsymbol{\phi}\right] - \log\left[\sum_{k=1}^{K} \exp\left[\ \mathrm{f}_k\left[\mathbf{x}_i, \boldsymbol{\phi}\right]\right]\right]$$

- Only with one observation:

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log \widehat{y_k} = -\sum_{k=1}^{K} y_k \log \hat{p}(y = k|x)$$

Example



- $y = [\text{cat, dog,egg}] = [1, 0, 0], \hat{y} = [0.5, 0.3, 0.2]$
- Loss for class *i* $= -y_i \log \hat{y}_i$

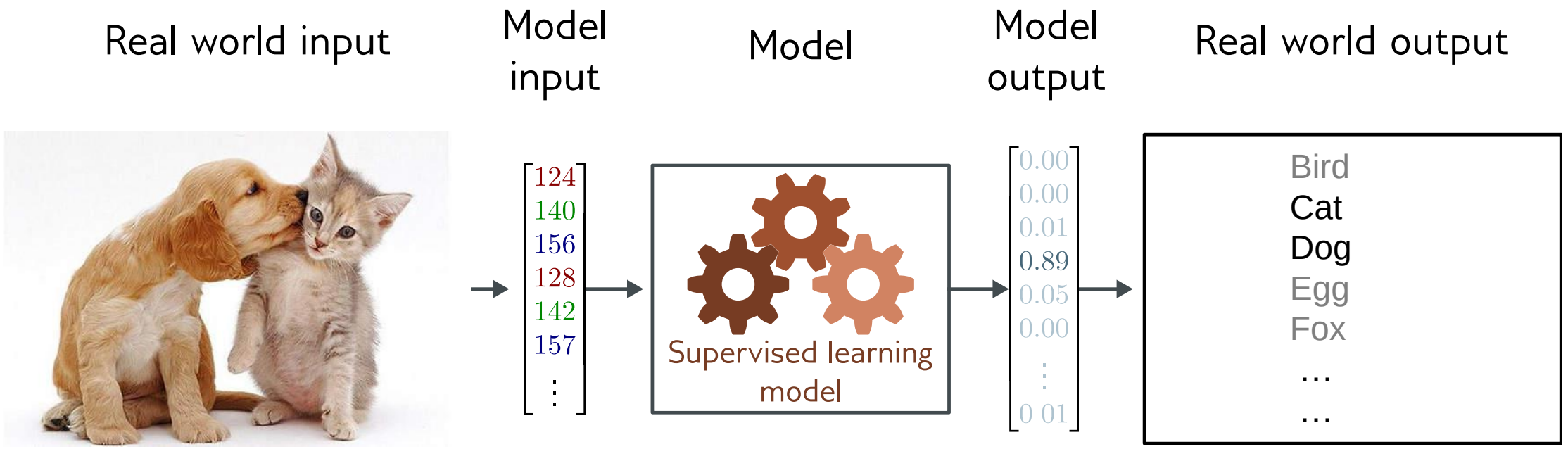
Probability of class i in target

Probability of class i in prediction

- Loss for class *cat* $= -y_i \log \hat{y}_i = -1log0.5 = 0.693$
- Loss for class *dog* $= -y_i \log \hat{y}_i = -0log0.3 = 0$
- Loss for class *egg* $= -y_i \log \hat{y}_i = -0log0.2 = 0$
- Cross entropy = loss for cat + loss for dog + loss for egg

  = 0.693 + 0 + 0

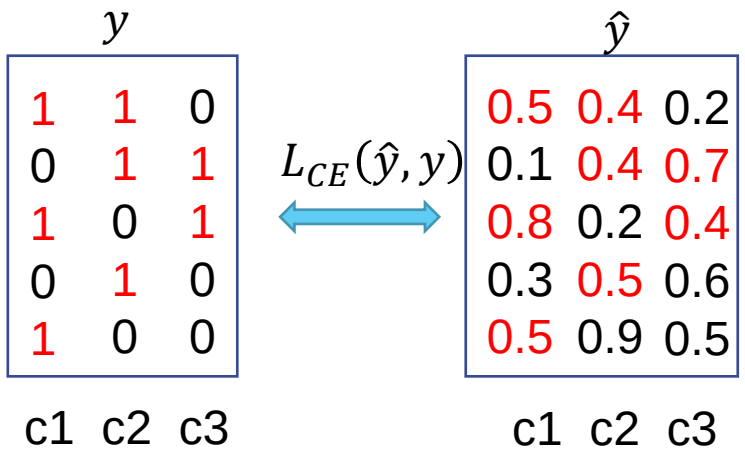
Multi-label multi-class classification



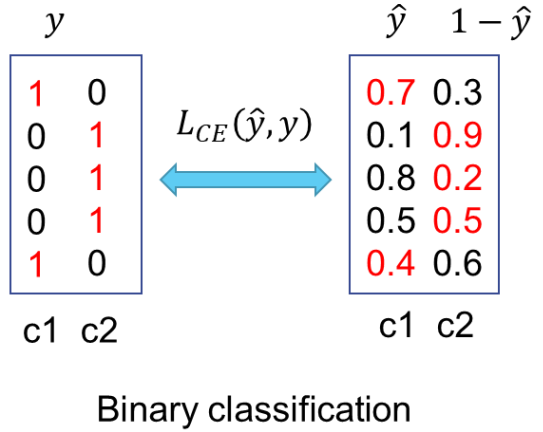
- Output layer: sigmoid()
- Output encoding: 1-hot encoding

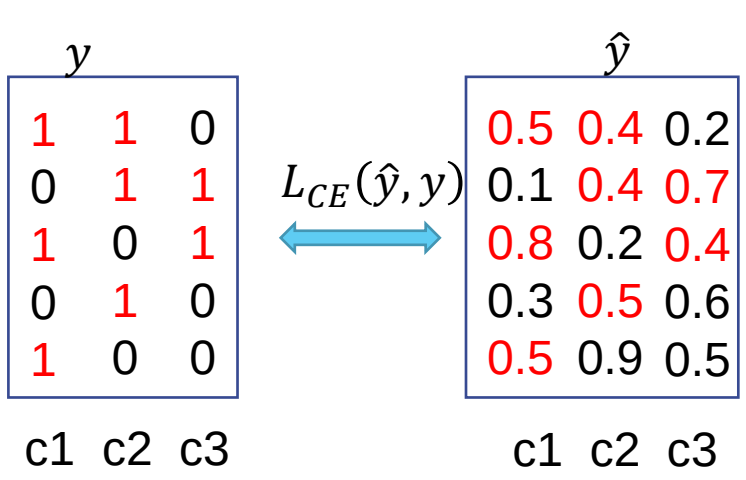
Question?

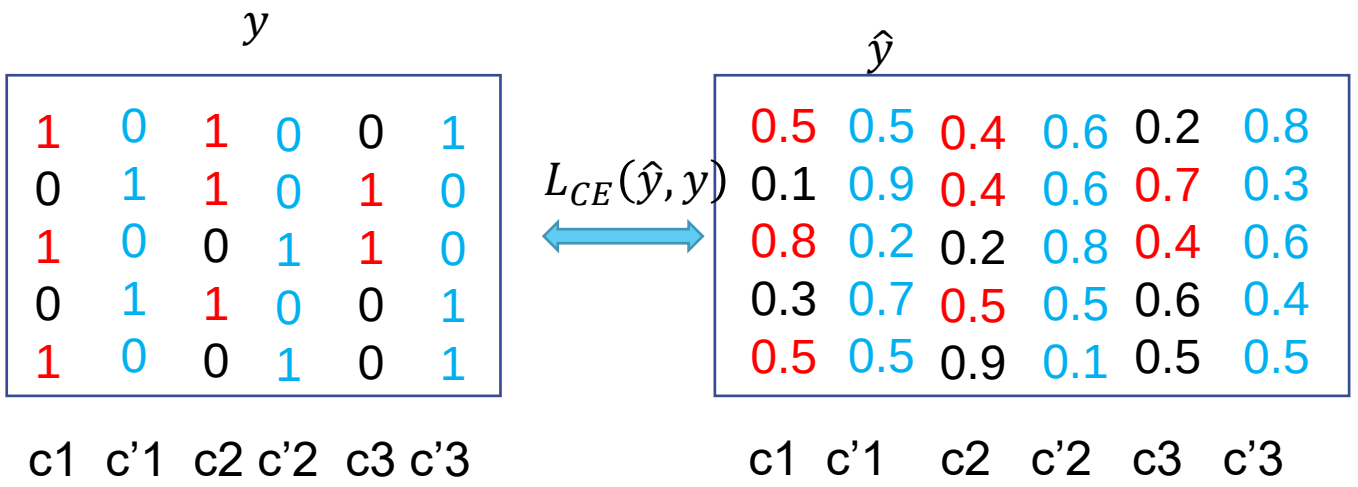
- 如果是multi-label multi class的問題，是不是也是用cross-entropy？



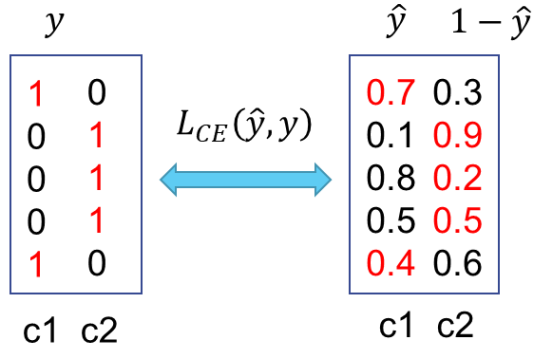
Question?


y
$\hat{y}$ $1-\hat{y}$
1 0
0 1
0 1
0 1
1 0
$L_{CE}(\hat{y}, y)$
0.7 0.3
0.1 0.9
0.8 0.2
0.5 0.5
0.4 0.6
c1 c2
c1 c2
Binary classification

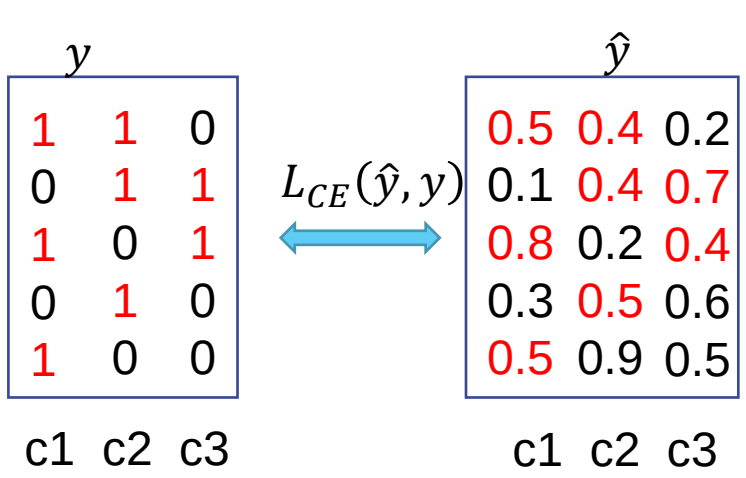

y
$\hat{y}$
1 1 0
0 1 1
1 0 1
0 1 0
1 0 0
$L_{CE}(\hat{y}, y)$
0.5 0.4 0.2
0.1 0.4 0.7
0.8 0.2 0.4
0.3 0.5 0.6
0.5 0.9 0.5
c1 c2 c3
c1 c2 c3

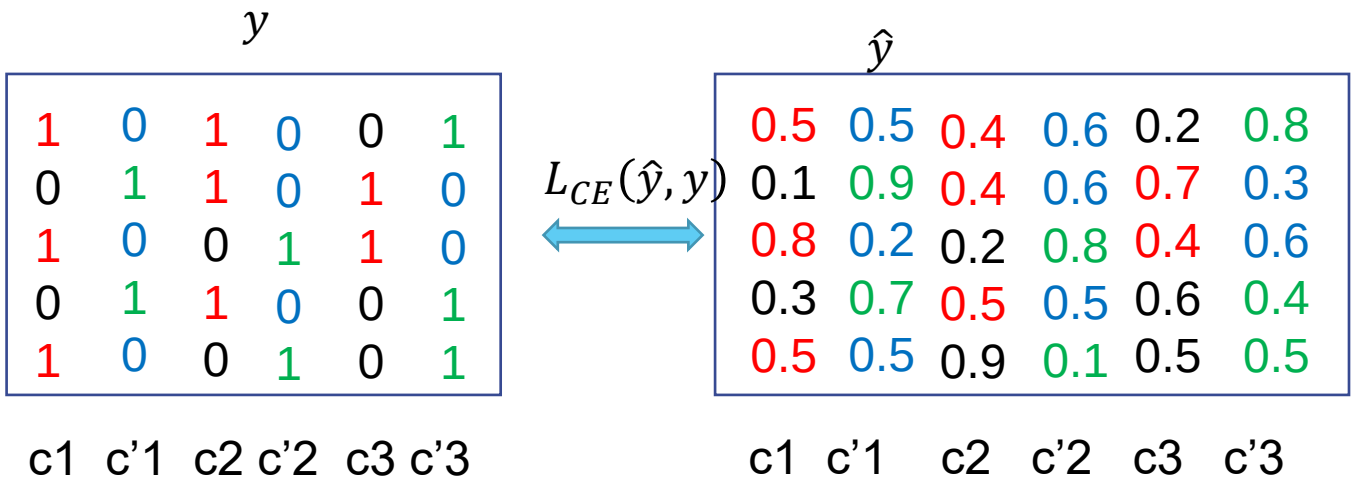

y
$\hat{y}$
1 0 1 0 0 1
0 1 1 0 1 0
1 0 0 1 1 0
0 1 1 0 0 1
1 0 0 1 0 1
$L_{CE}(\hat{y}, y)$
0.5 0.5 0.4 0.6 0.2 0.8
0.1 0.9 0.4 0.6 0.7 0.3
0.8 0.2 0.2 0.8 0.4 0.6
0.3 0.7 0.5 0.5 0.6 0.4
0.5 0.5 0.9 0.1 0.5 0.5
c1 c'1 c2 c'2 c3 c'3
c1 c'1 c2 c'2 c3 c'3

Question?



Binary classification





Example

y

c1	c'1	c2	c'2	c3	c'3
1	0	1	0	0	1
0	1	1	0	1	0
1	0	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1

$L_{CE}(\hat{y}, y)$

$\hat{y}$

c1	c'1	c2	c'2	c3	c'3
0.5	0.5	0.4	0.6	0.2	0.8
0.1	0.9	0.4	0.6	0.7	0.3
0.8	0.2	0.2	0.8	0.4	0.6
0.3	0.7	0.5	0.5	0.6	0.4
0.5	0.5	0.9	0.1	0.5	0.5

- $y = [\text{cat}, \text{dog}, \text{egg}] = [1, 1, 0], \hat{y} = [0.5, 0.4, 0.2]$
- Loss for class $i = -y_i \log \hat{y}_i$

Probability of class i in target

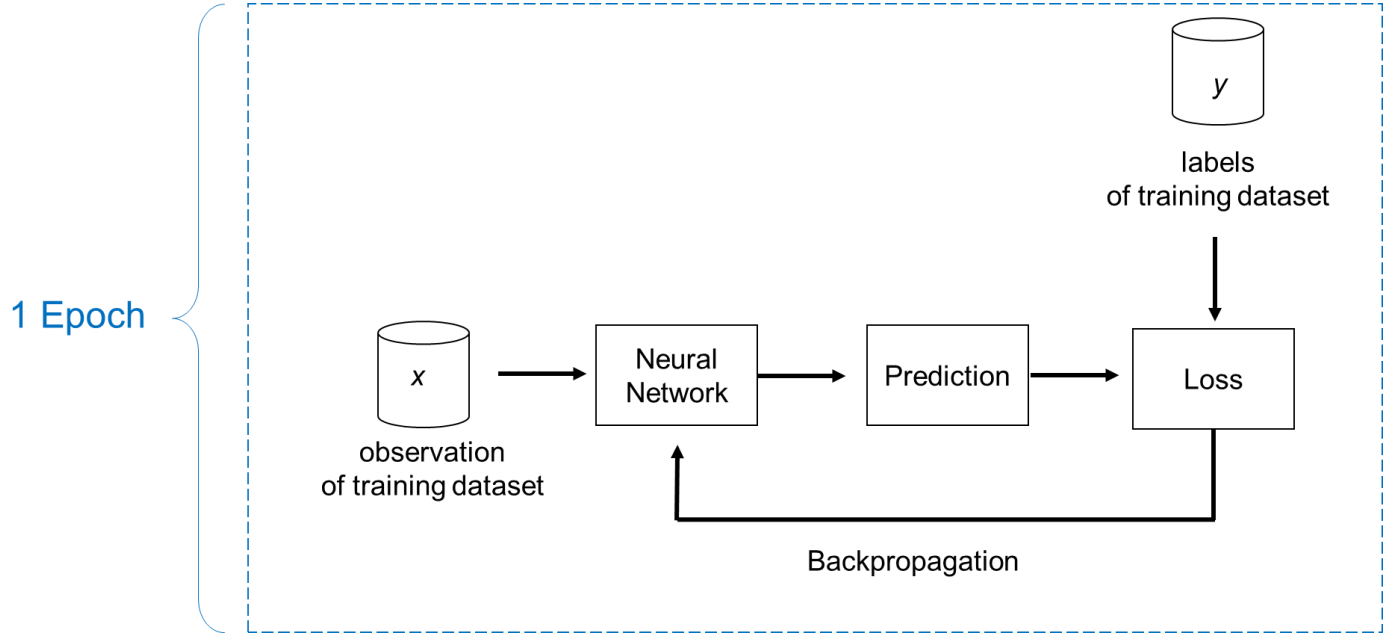
Probability of class i in prediction

- Loss for class *cat* $= -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] = -[1log0.5 + 0log0.5]$
- Loss for class *dog* $= -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] = -[1log0.4 + 0log0.6]$
- Loss for class *egg* $= -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] = -[0log0.2 + 1log0.8]$
- Cross entropy = loss for cat + loss for dog + loss for egg

  = (-log0.5) + (-log0.4) + (-log0.8)

Outline

- Recap Loss functions
- Gradient descent
- Backward Propagation



Our goal: minimize the loss

Let's make explicit that the loss function is parameterized by weights $\theta = (w, b)$

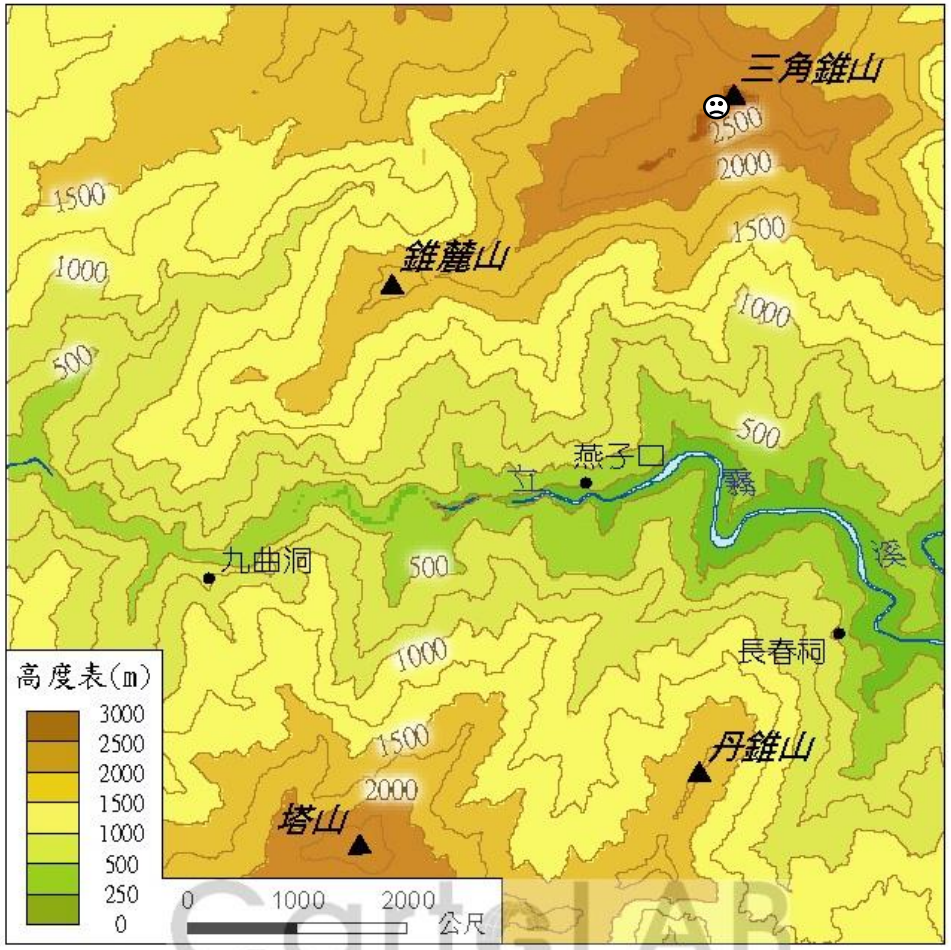
- And we will represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious

We want the weights that minimize the loss, **averaged** over all examples:

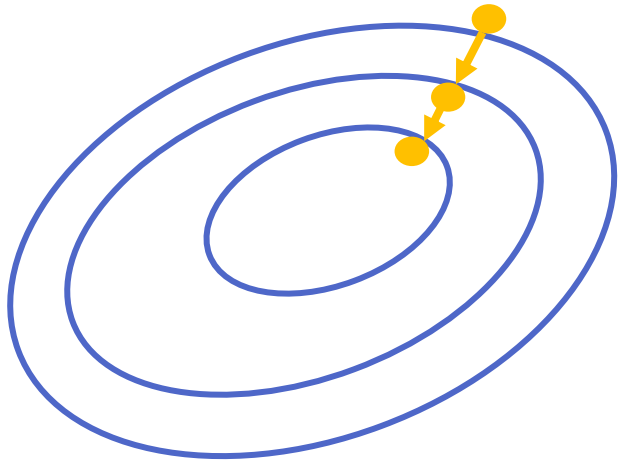
$$\hat{\theta} = argmin_{\theta} \frac{1}{m} L_{CE}(f(x^i; \theta), y^i)$$

Intuition of gradient descent

How do I get the bottom of this river canyon (燕子口)?

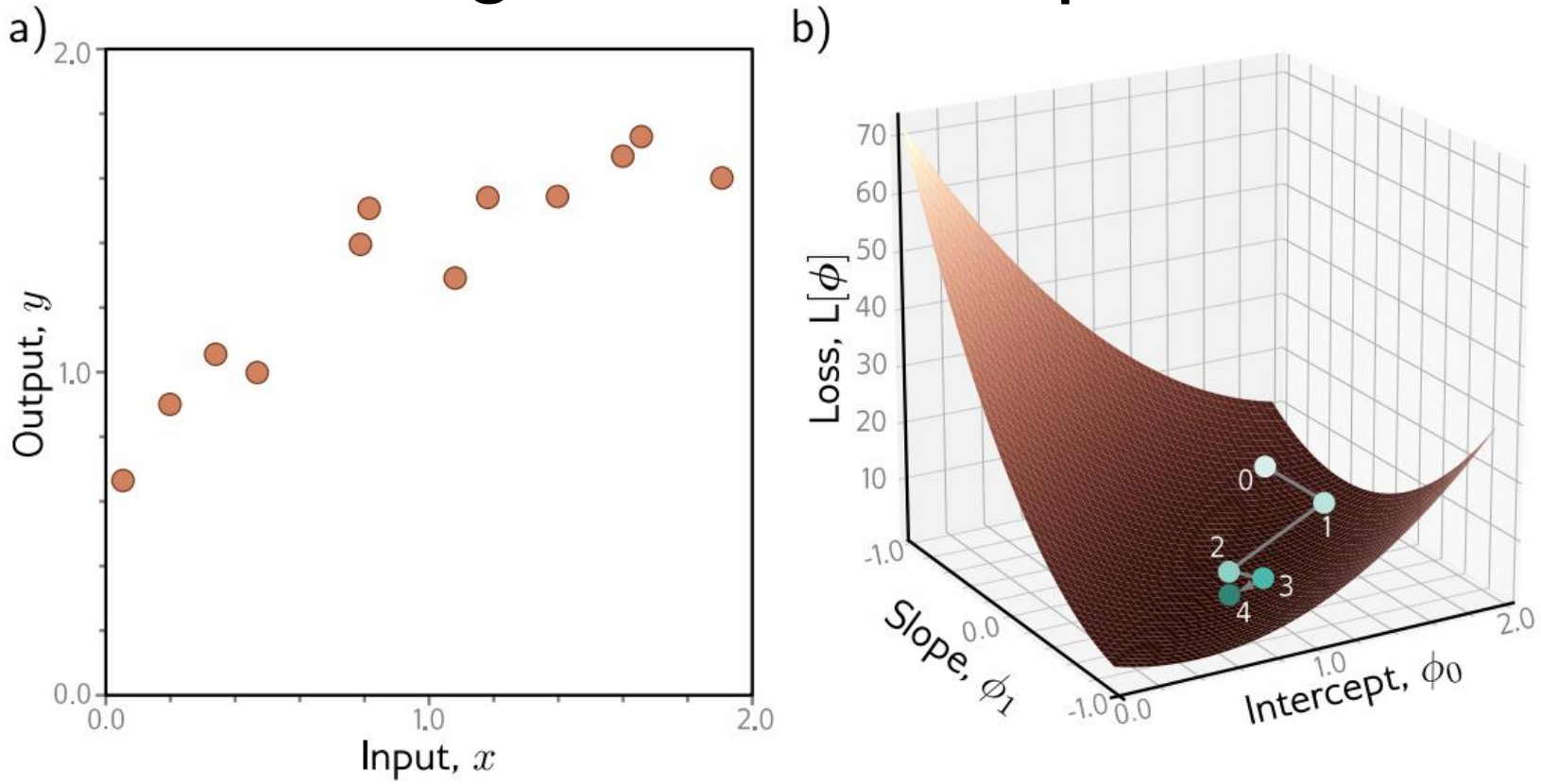


Look around me 360°
Find the direction of steepest slope down
Go that way



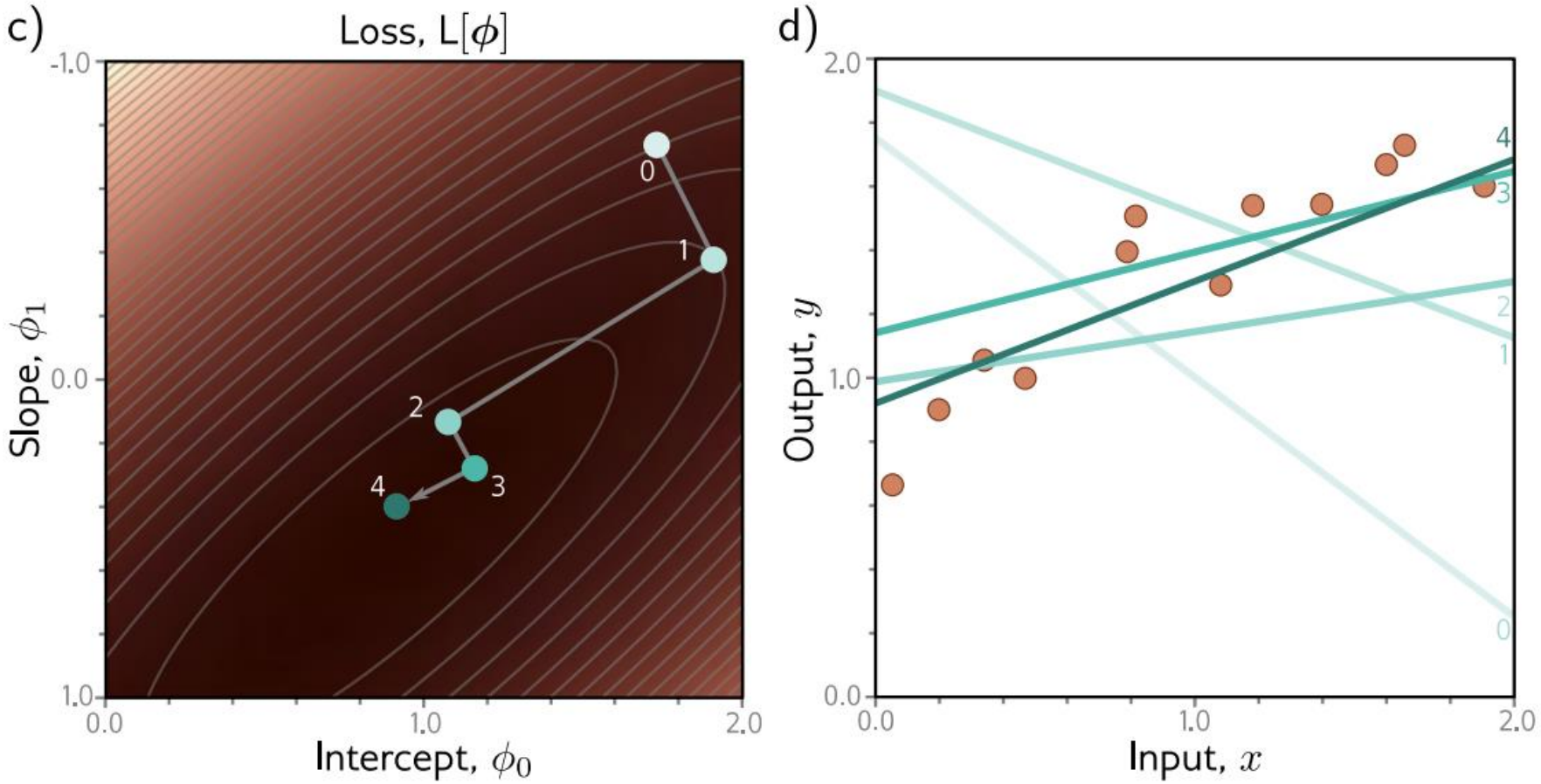
Source: 臺大地理系

Recall the linear regression example



a) Training set of *I* = 12 input/output pairs *{xi, yi}*.
b) Loss function showing iterations of gradient descent. We start at point 0 and move in the steepest downhill direction until we can improve no further to arrive at point 1. We then repeat this procedure. We measure the gradient at point 1 and move downhill to point 2 and so on.

Recall the linear regression example



The model with the parameters at point 0 (lightest line) describes the data very badly, but each successive iteration improves the fit. The model with the parameters at point 4 (darkest line) is already a reasonable description of the training data.

Gradient descent algorithm

Step 1. Compute the derivatives of the loss with respect to the parameters:

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial L}{\partial \phi_0} \\ \frac{\partial L}{\partial \phi_1} \\ \vdots \\ \frac{\partial L}{\partial \phi_N} \end{bmatrix}.$$

Step 2. Update the parameters according to the rule:

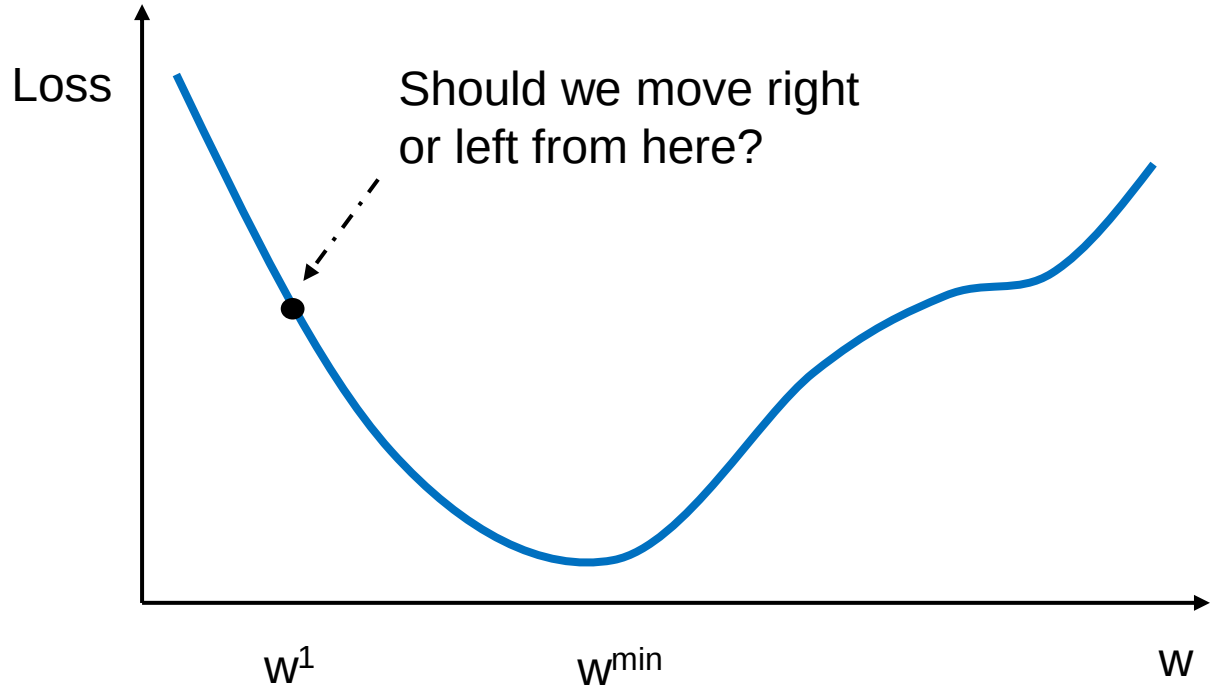
$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}},$$

where the positive scalar α determines the magnitude of the change.

Let's first visualize for a single scalar w

Q: Given current *w*, should we make it bigger or smaller?

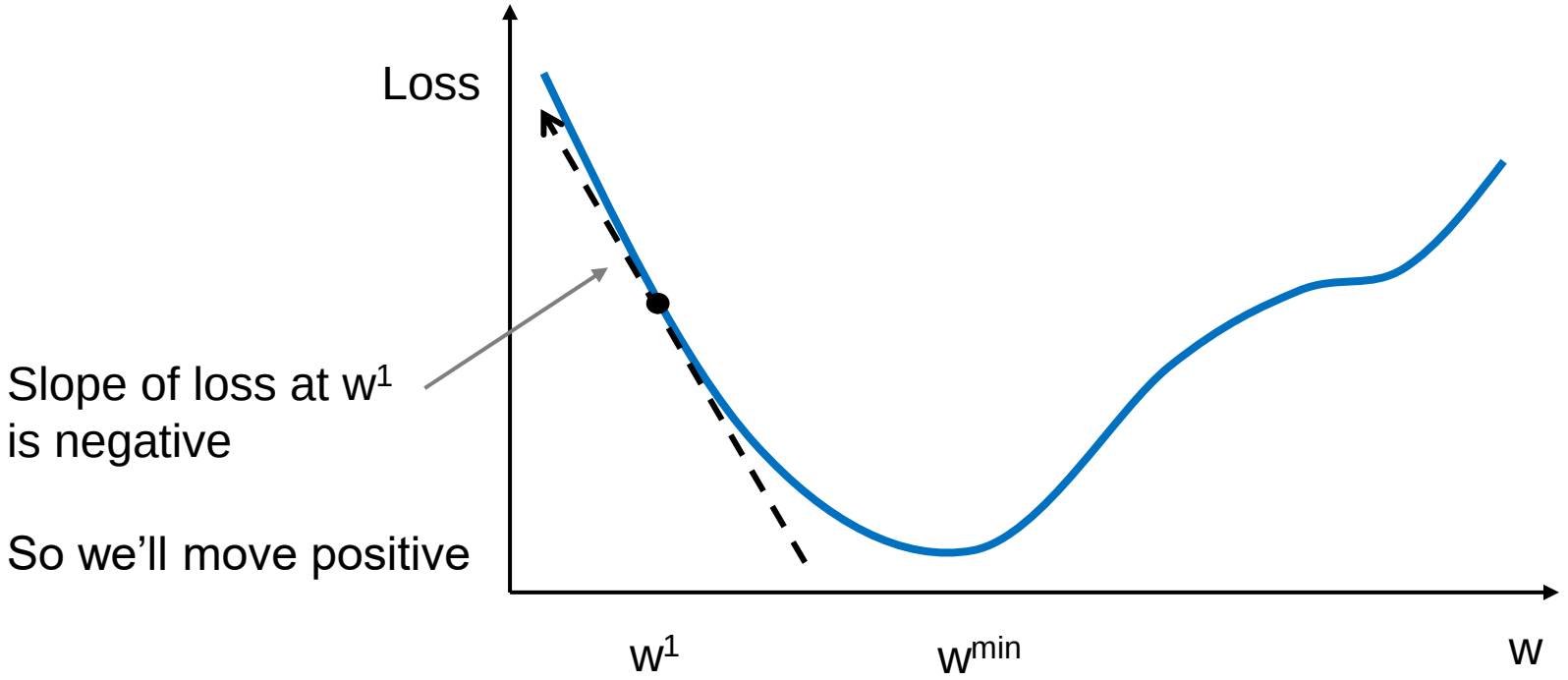
A: Move *w* in the reverse direction from the slope of the function.



Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

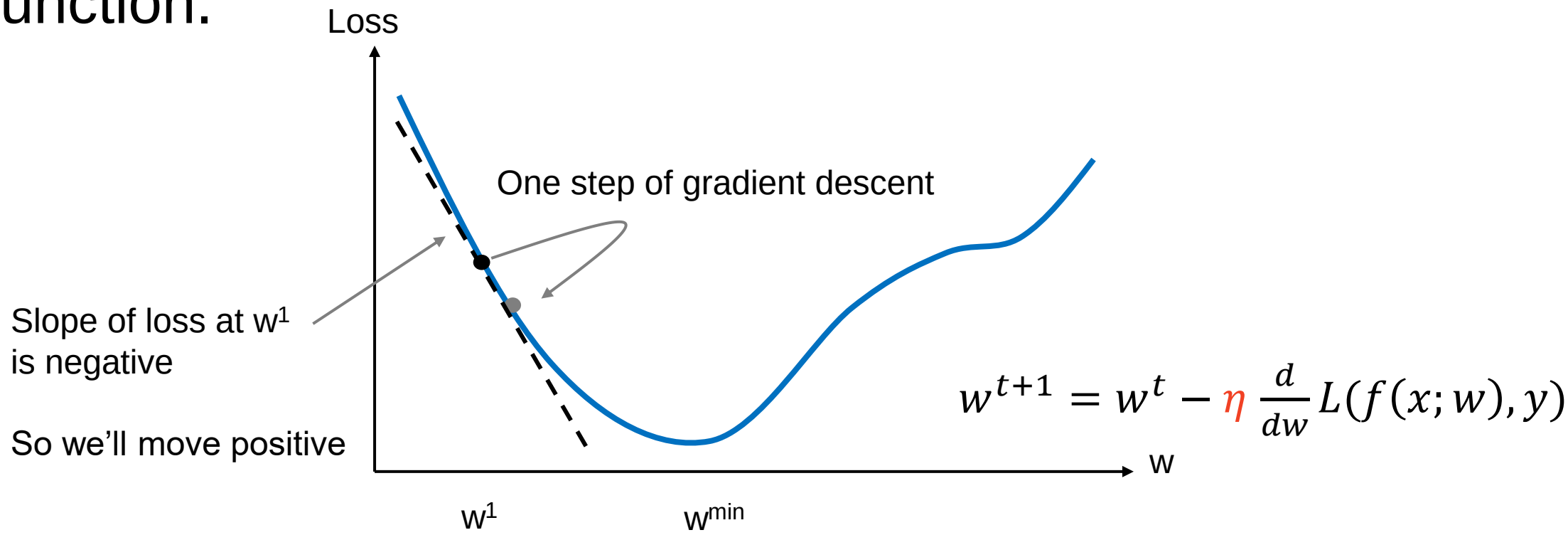
A: Move *w* in the reverse direction from the slope of the function.



Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function.



$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

Gradients

- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

$$\frac{d}{dw} L(f(x; w), y)$$

- **Gradient Descent**: Find the gradient of the loss function at the current point and move in the **opposite** direction.

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

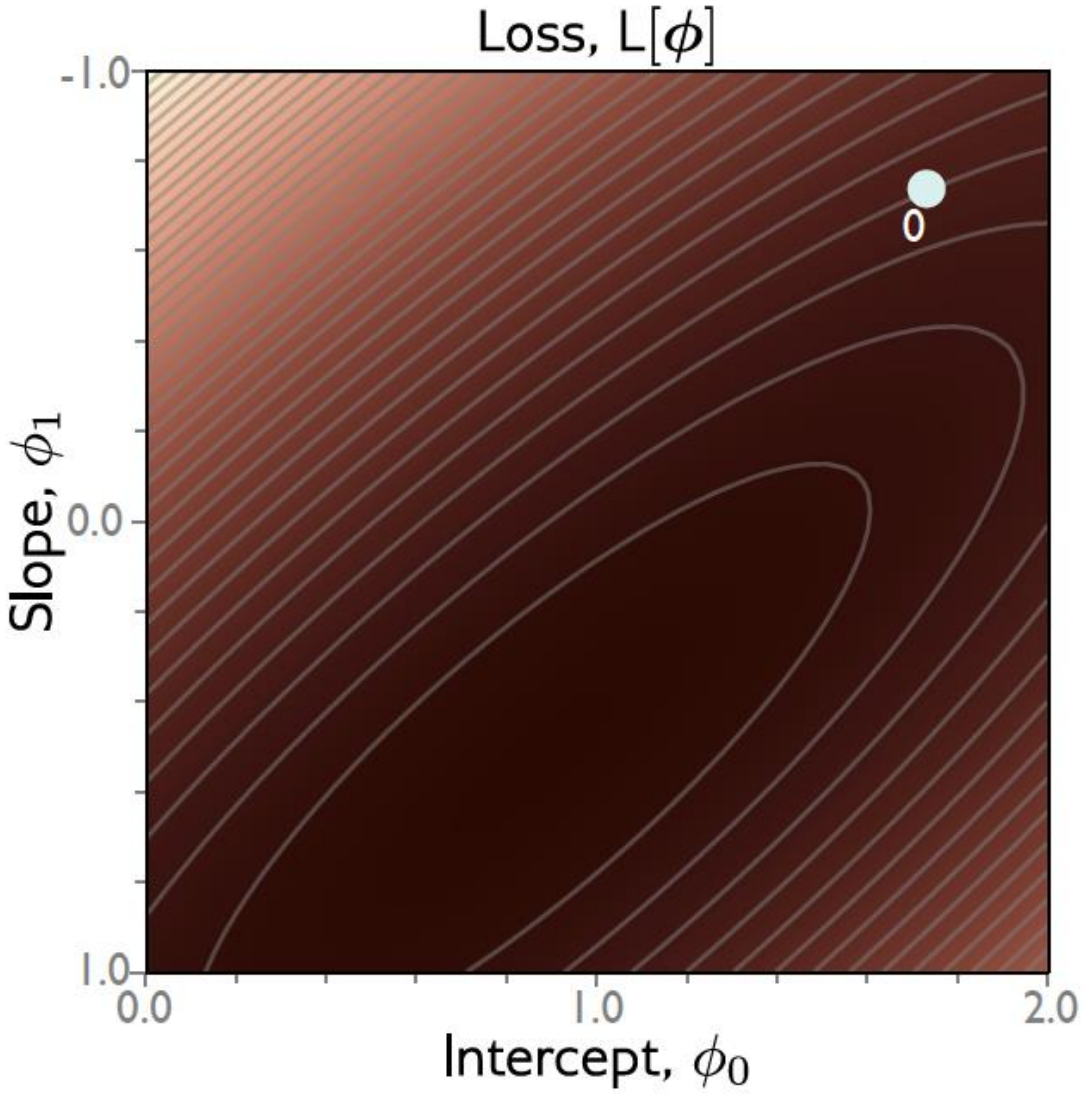
Gradient descent

- Choose a starting point
- Repeat to update the weight t = 1, 2, 3 …

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

  - The value of the gradient (slope in our example) $\frac{d}{dw} L(f(x; w), y)$ weighted by a learning rate η
    - Gradient: a direction that increases the value
    - Learning rate: a hyper-parameter specifies the step length
    - Higher learning rate means move w faster

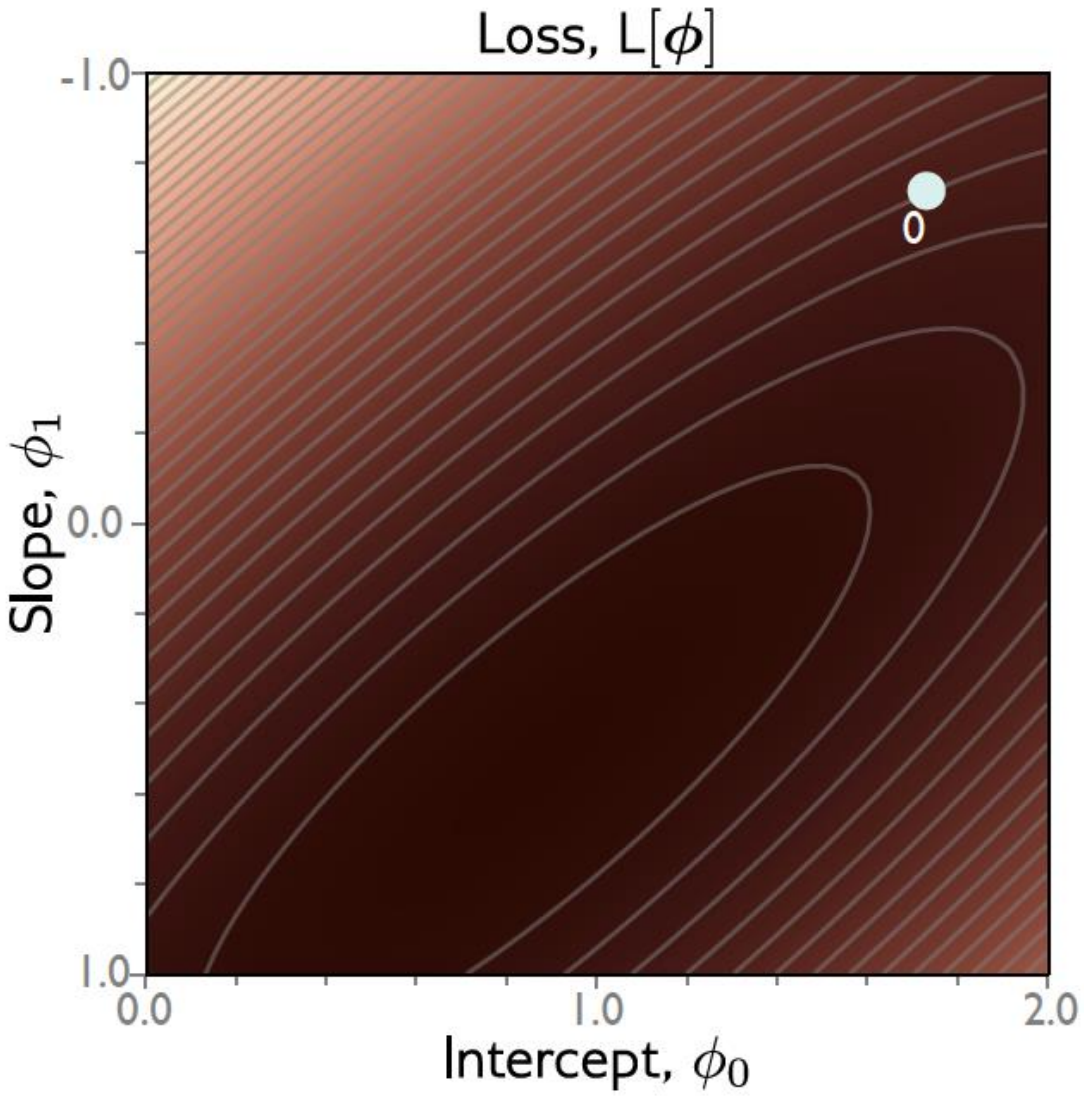
Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

Gradient descent

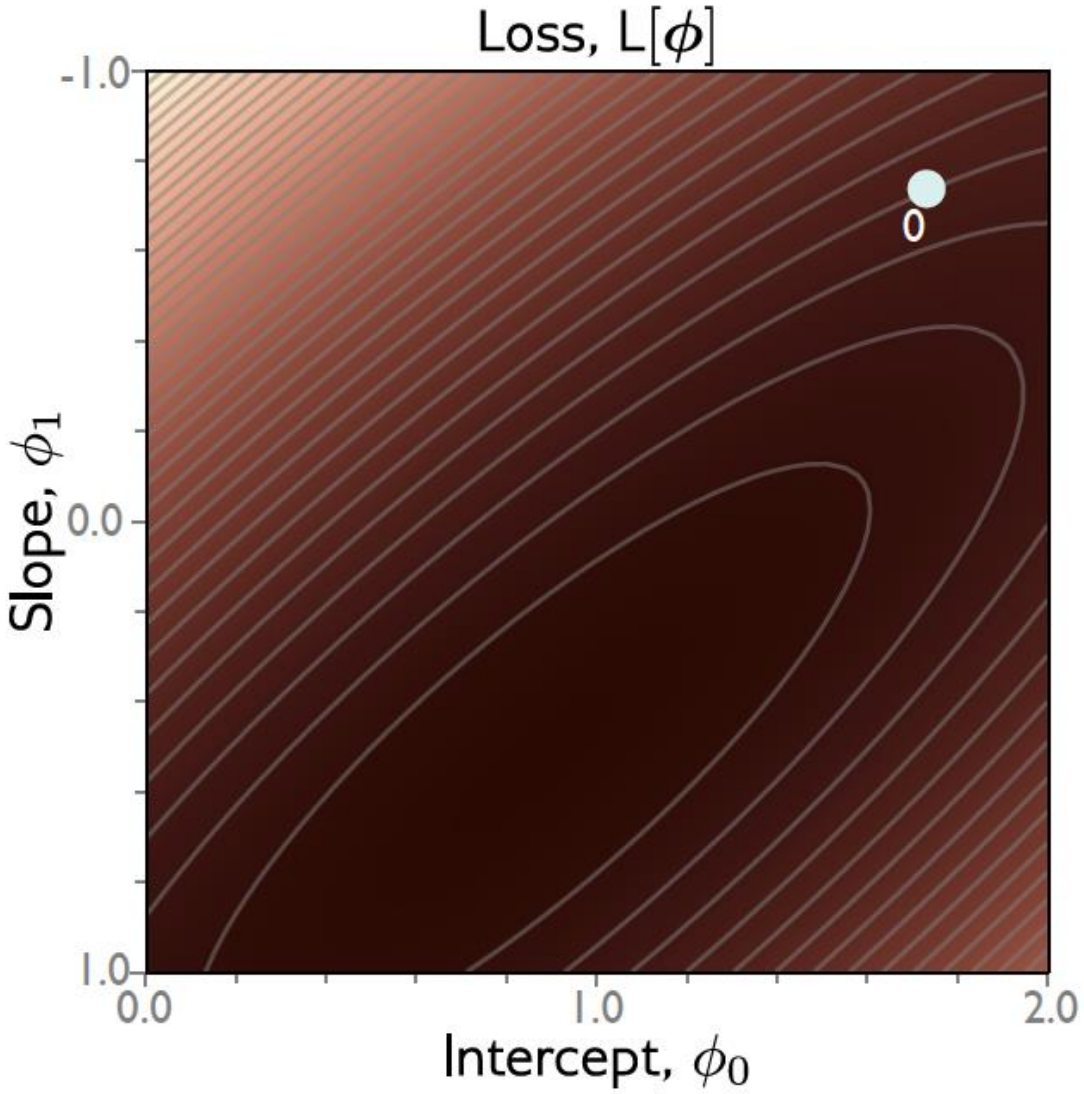


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

Gradient descent



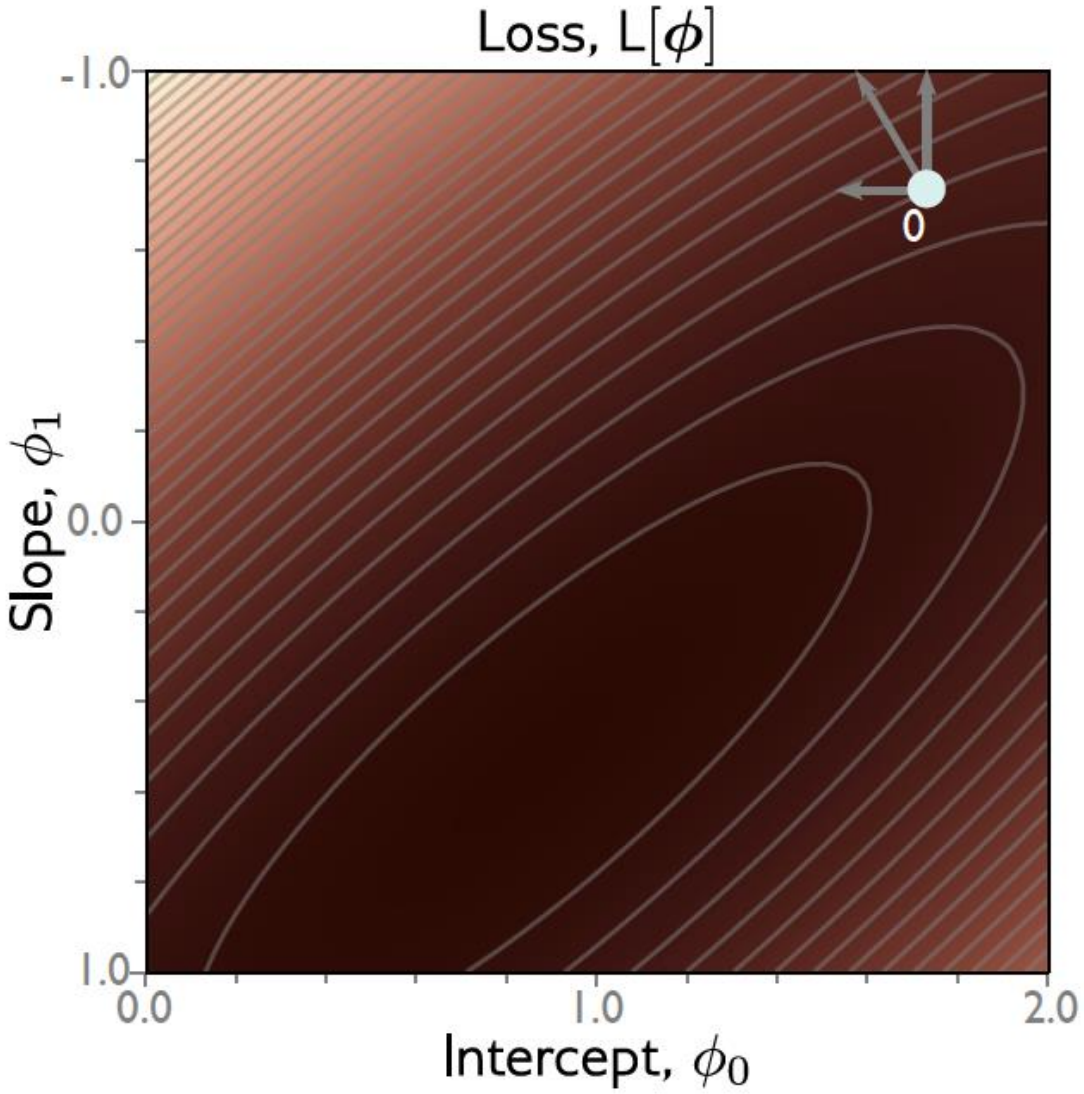
Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$L[\boldsymbol{\phi}] = \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} (\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2$$
$$= \sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2$$

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Gradient descent

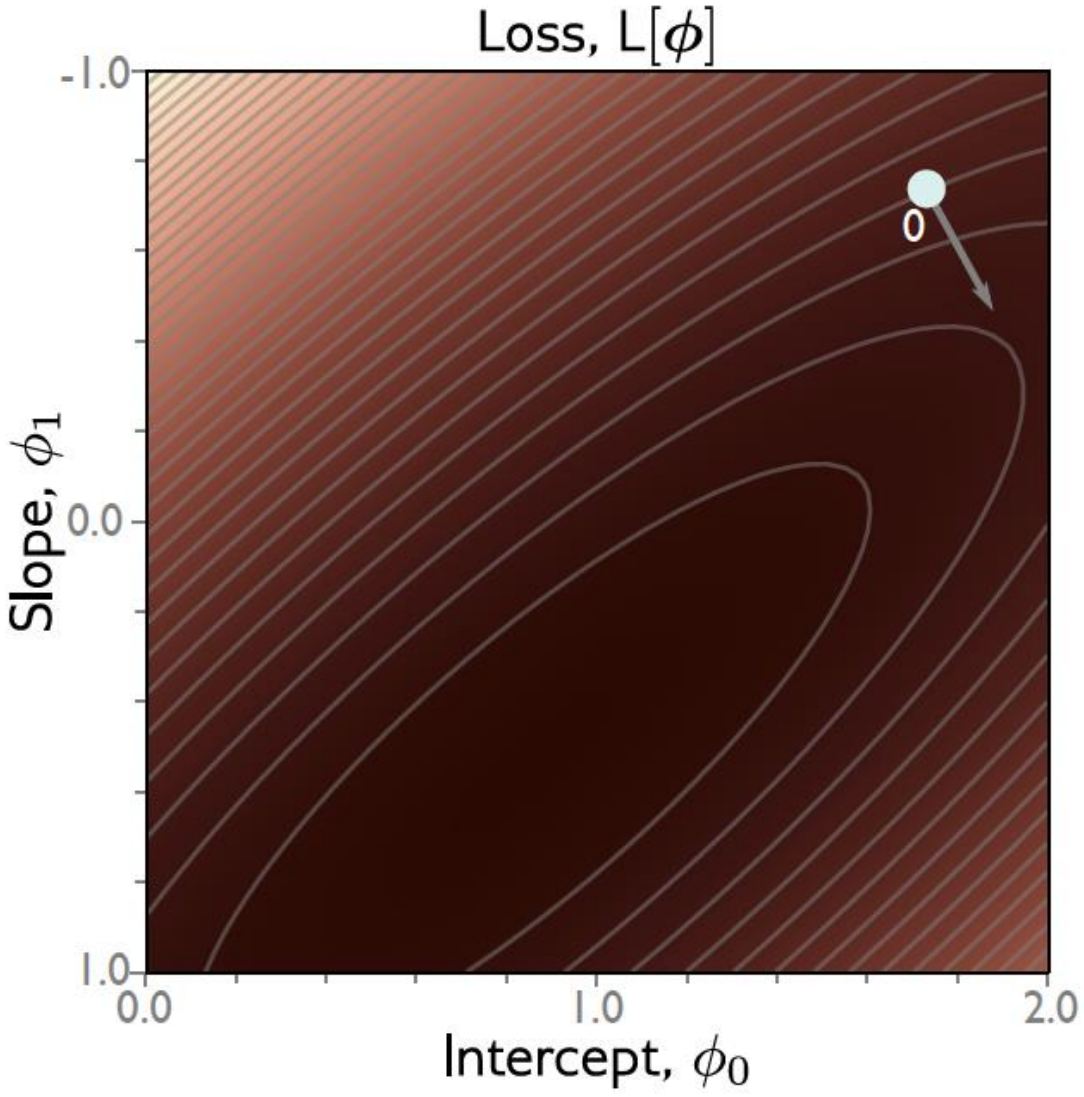


Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

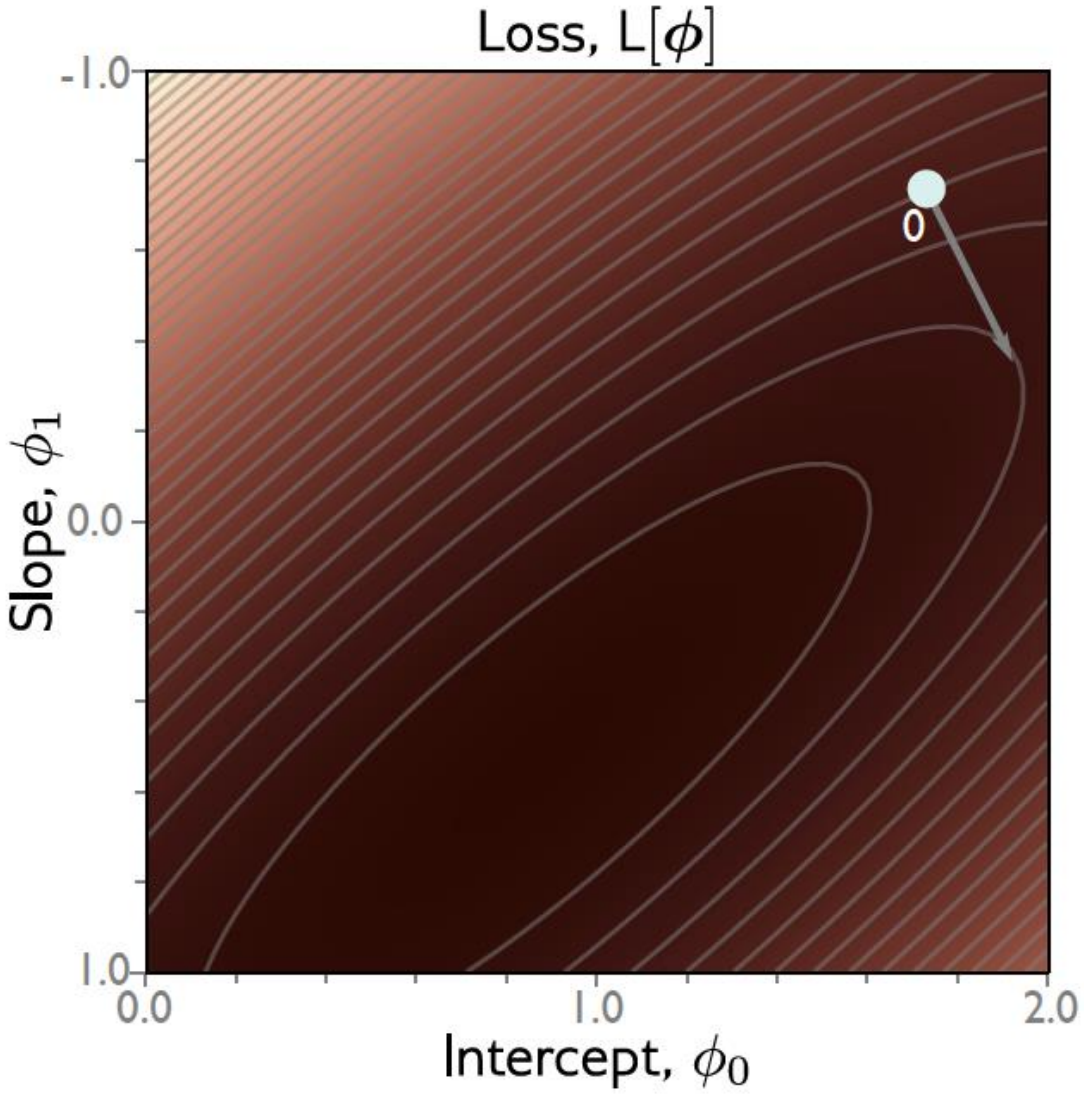
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size or learning rate if fixed

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

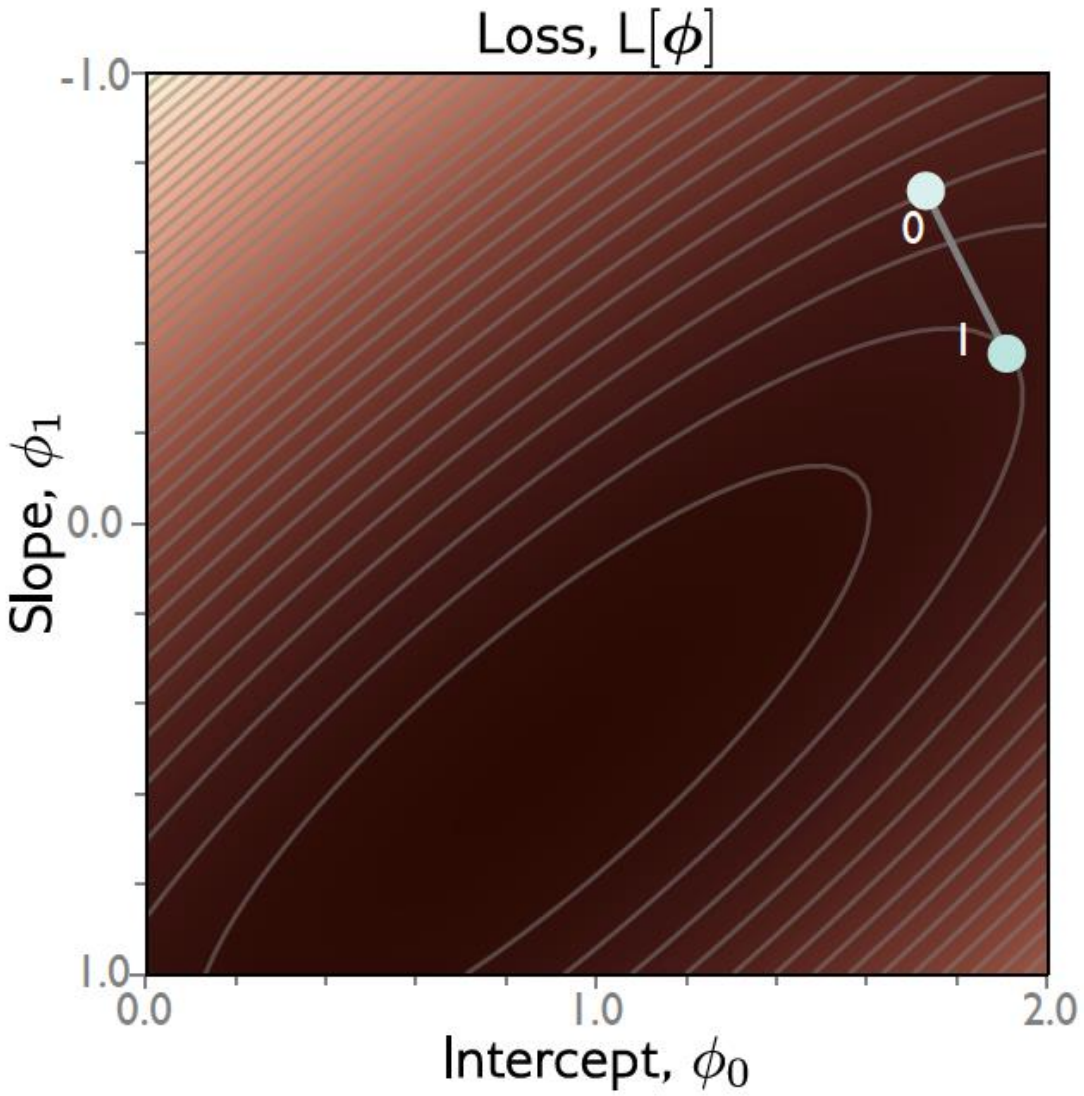
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

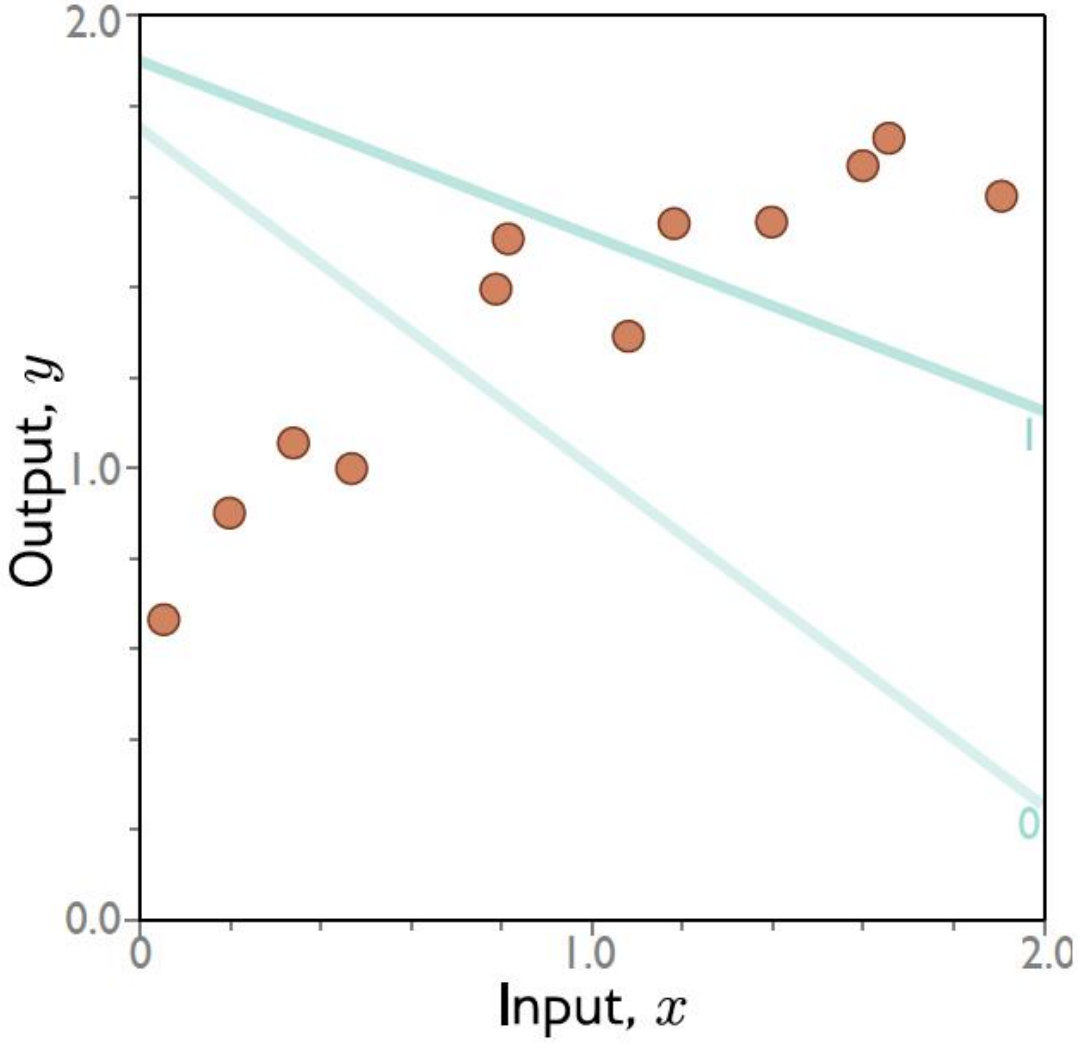
Step 2: Update parameters according to rule

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

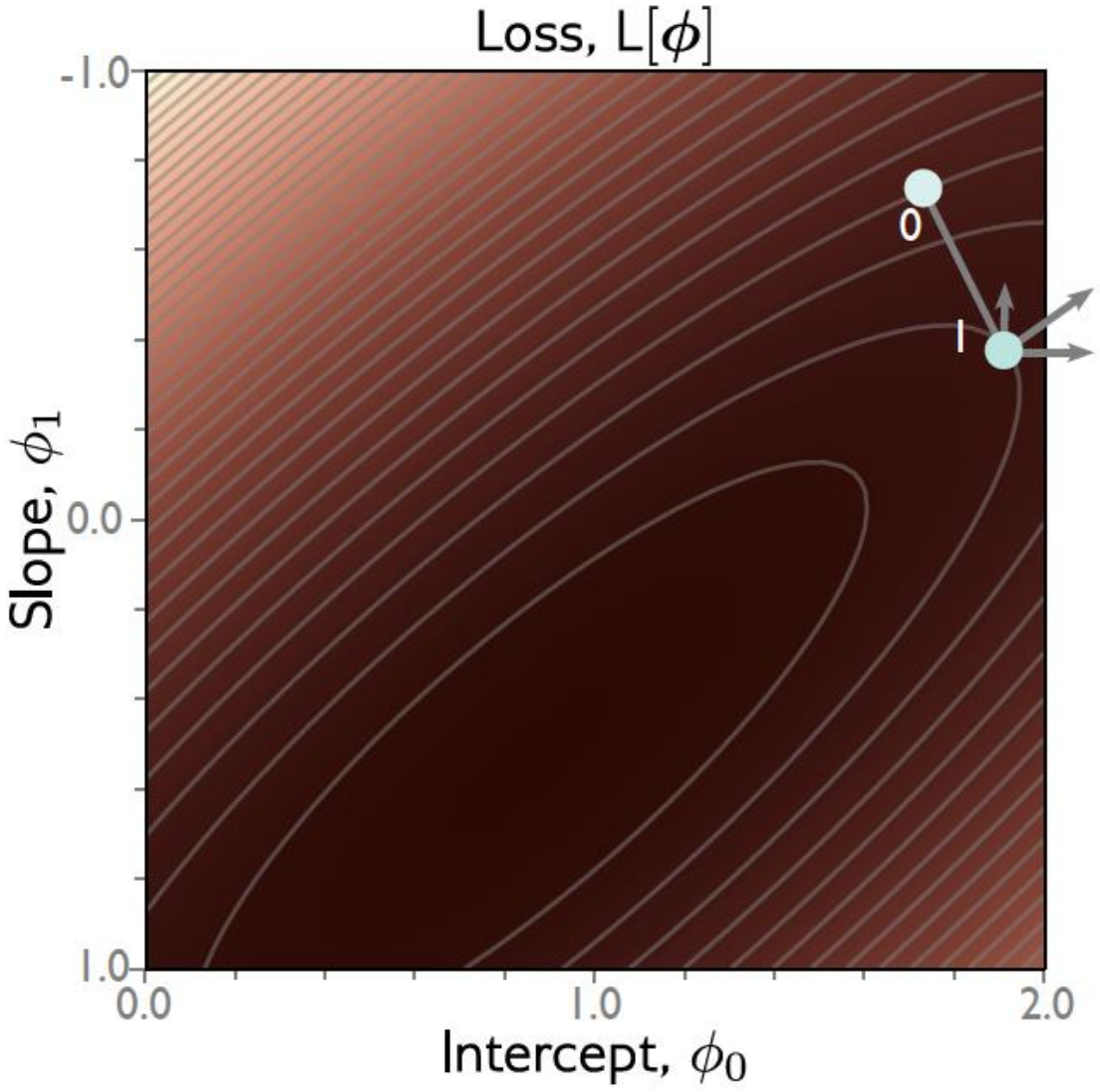
α = step size

Gradient descent





Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

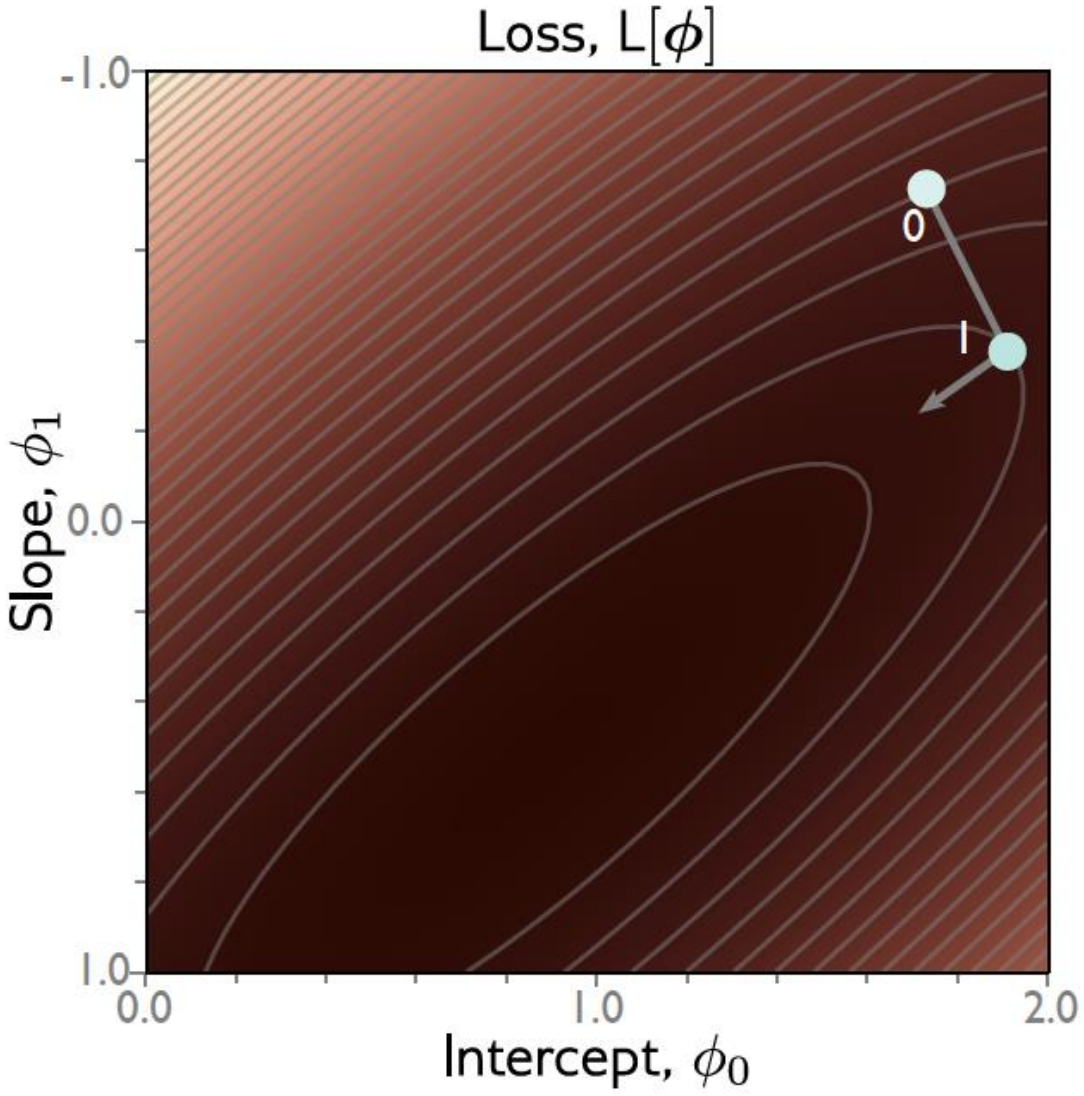
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

Step 2: Update parameters according to rule

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

α = step size

Gradient descent



Step 1: Compute derivatives (slopes of function) with Respect to the parameters

$$\frac{\partial L}{\partial \boldsymbol{\phi}} = \frac{\partial}{\partial \boldsymbol{\phi}} \sum_{i=1}^{I} \ell_i = \sum_{i=1}^{I} \frac{\partial \ell_i}{\partial \boldsymbol{\phi}}$$

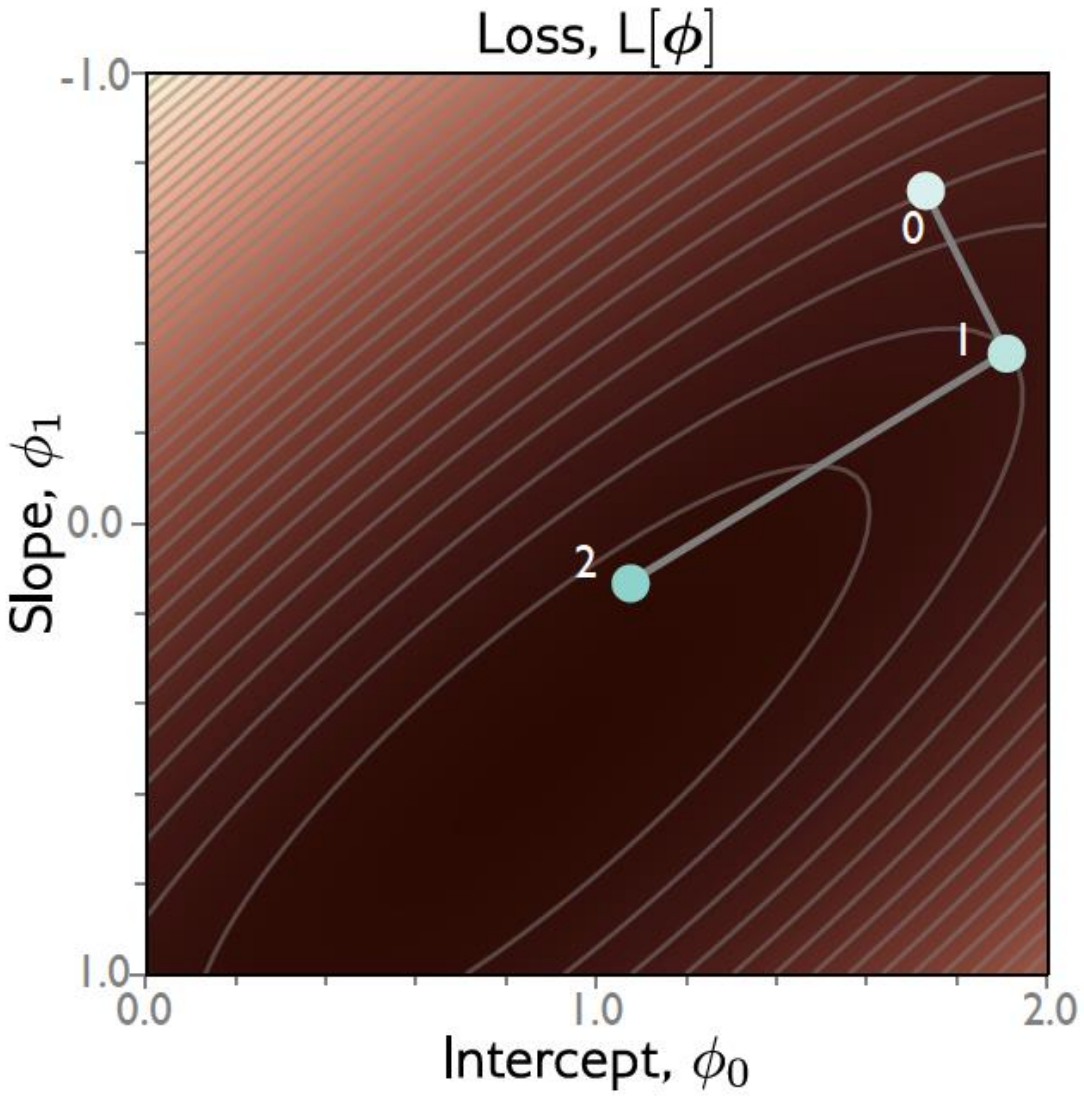
$$\frac{\partial \ell_i}{\partial \boldsymbol{\phi}} = \begin{bmatrix} \frac{\partial \ell_i}{\partial \phi_0} \\ \frac{\partial \ell_i}{\partial \phi_1} \end{bmatrix} = \begin{bmatrix} 2(\phi_0 + \phi_1 x_i - y_i) \\ 2x_i(\phi_0 + \phi_1 x_i - y_i) \end{bmatrix}$$

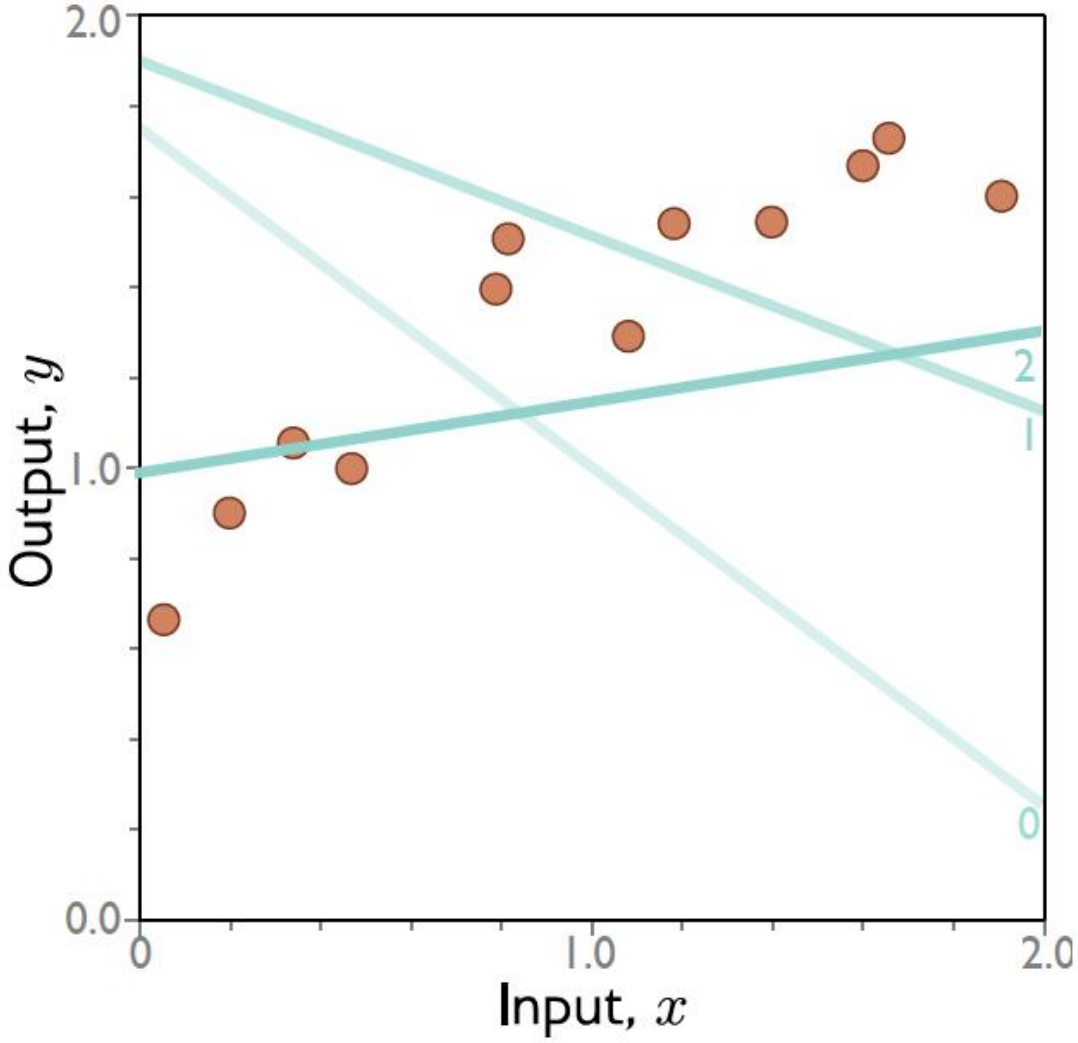
Step 2: Update parameters according to rule

$$\boldsymbol{\phi} \longleftarrow \boldsymbol{\phi} - \alpha \frac{\partial L}{\partial \boldsymbol{\phi}}$$

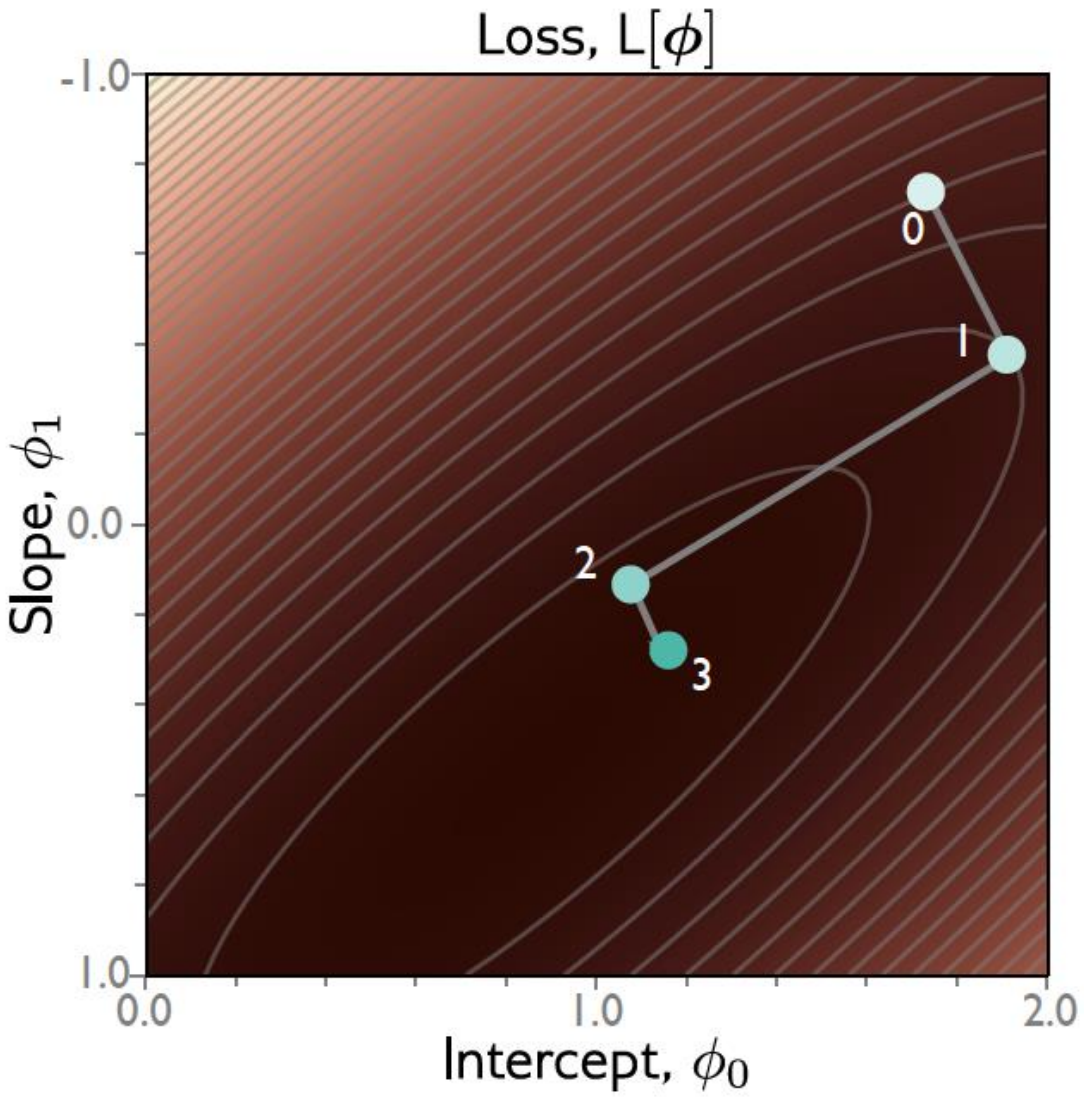
α = step size

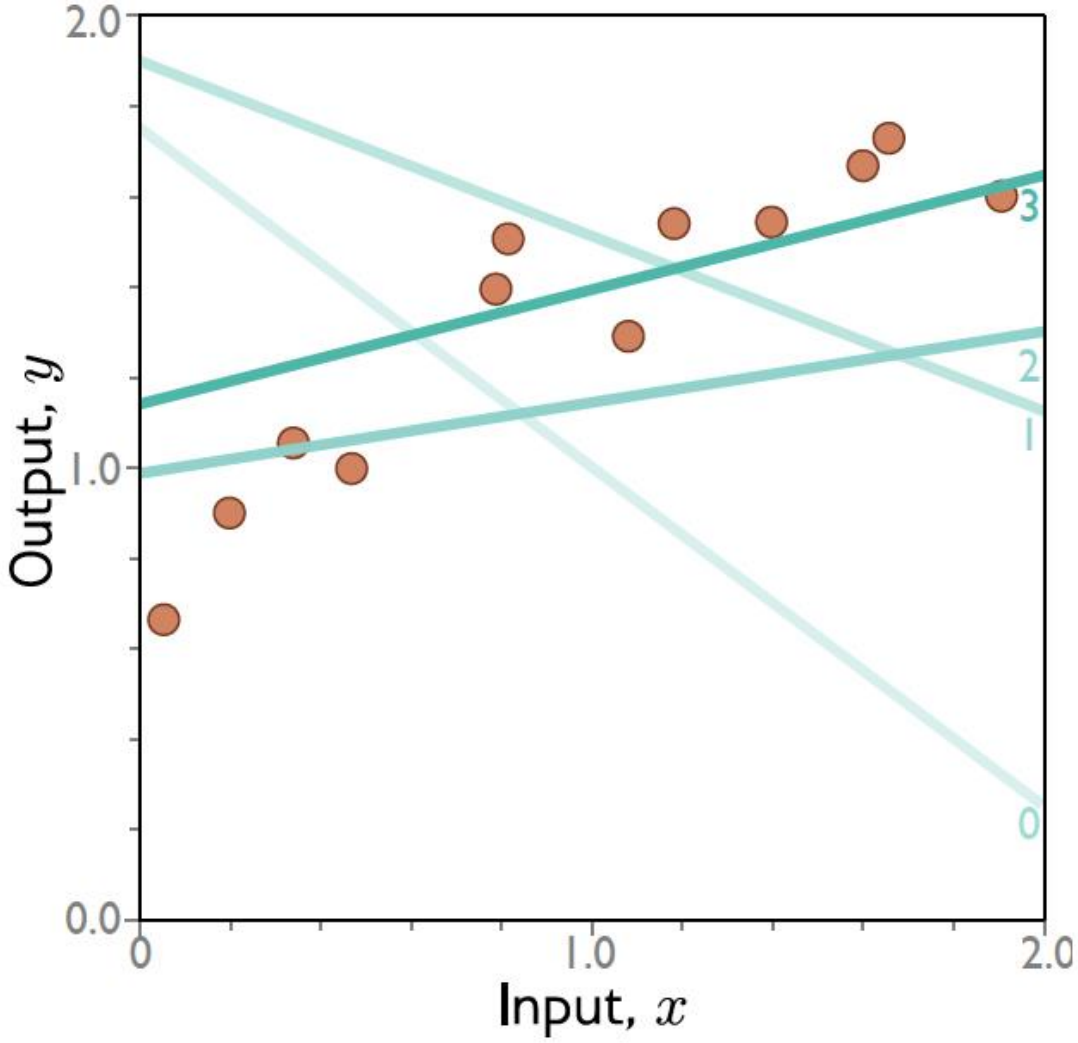
Gradient descent



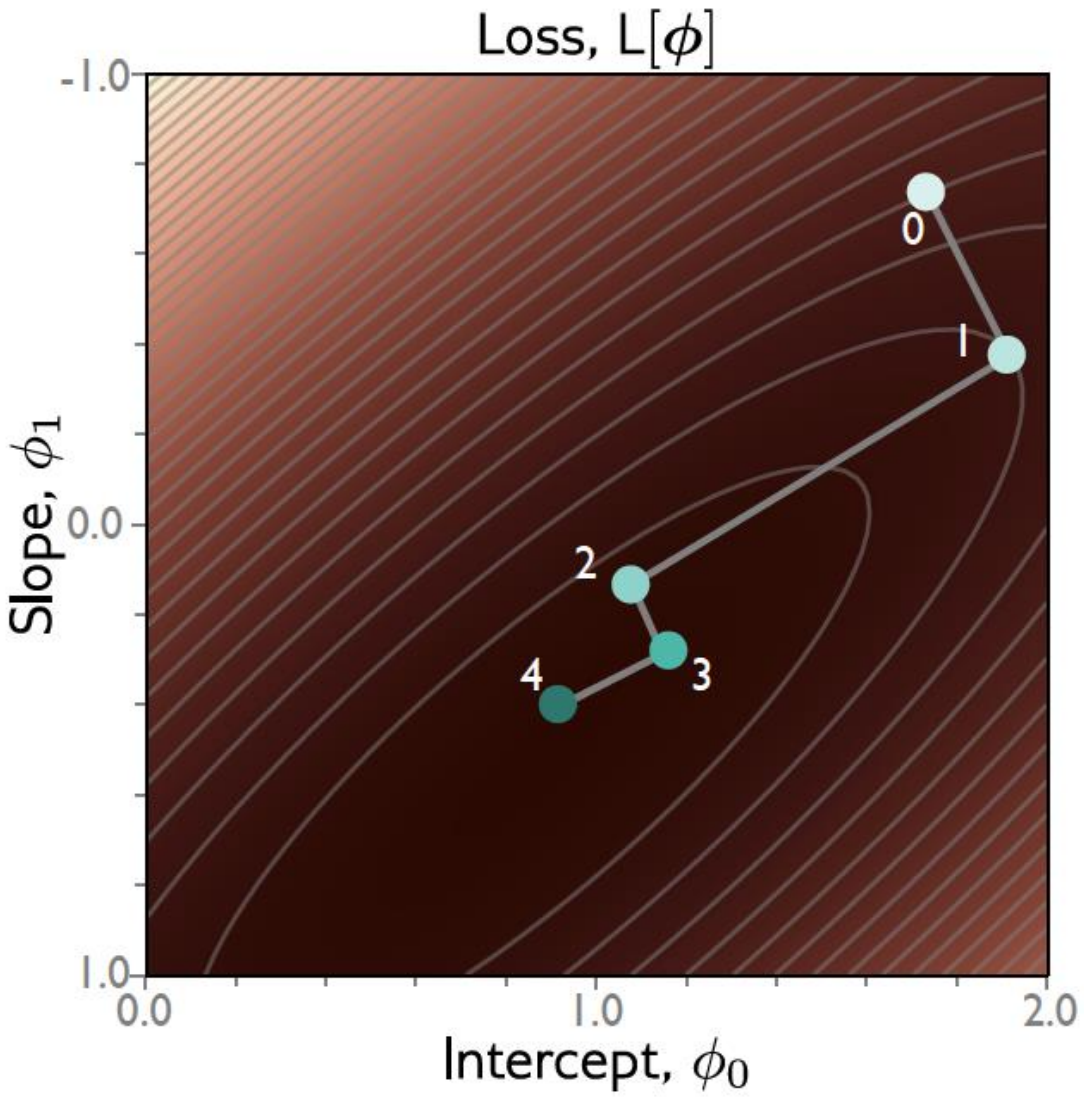


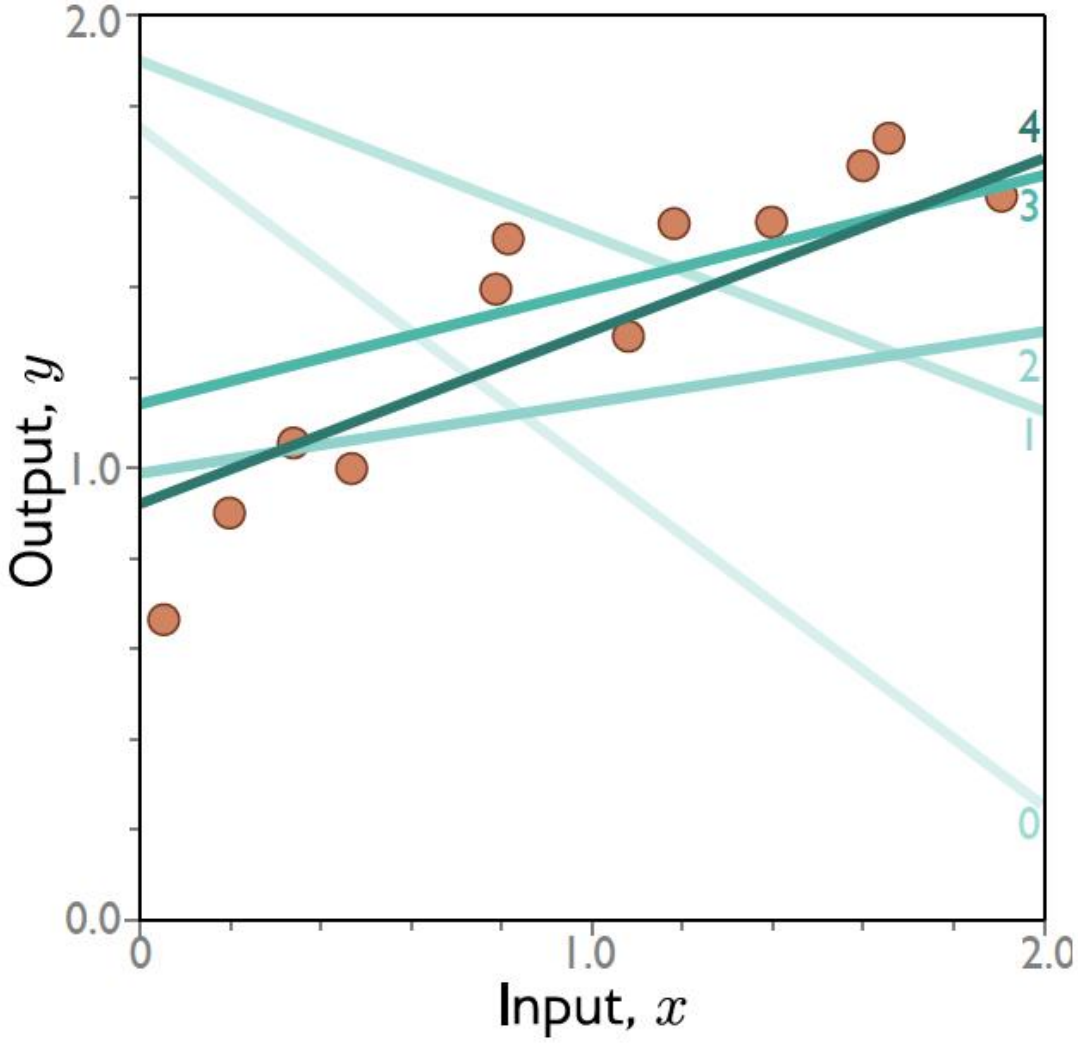
Gradient descent





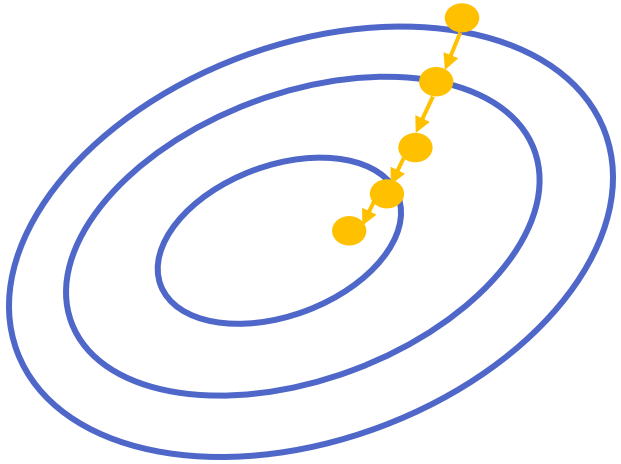
Gradient descent



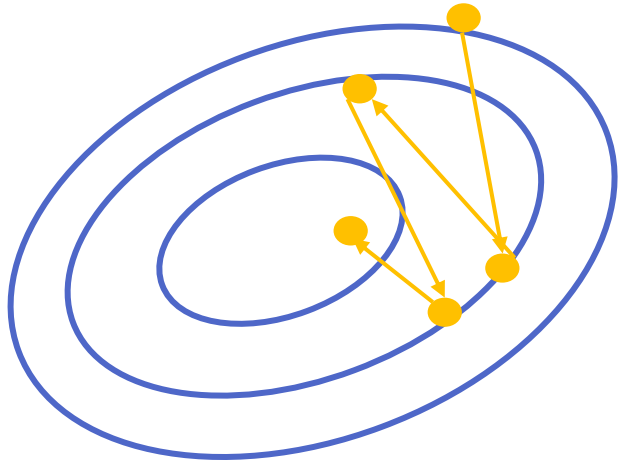


Choose a learning rate

- The learning rate η is a hyperparameter
  - too small: the learner will take too long
  - too big: the learner will take big steps and overshoot



Not too small



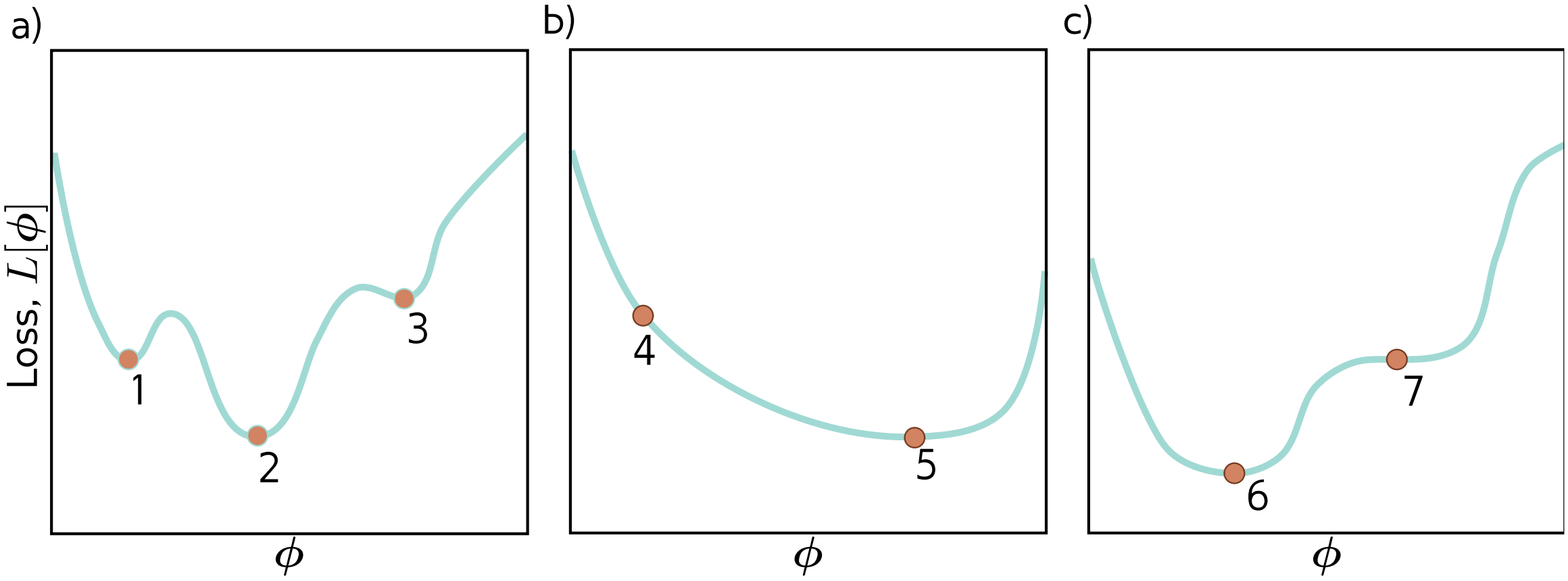
Not too big

Our goal: minimize the loss

For linear and logistic regression, loss function is convex

- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
  - Loss for neural networks is non-convex

Convex problems



Non convex

Convex

Non-Convex

Now let’s consider N dimensions

- We want to know where in the N-dimensional space (of the N parameters that make up θ) we should move.

- The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the *N* dimensions.

Real gradients

- Real gradients are much longer; lots and lots of weights
- For each dimension w_i , the gradient component *i* tells us the slope with respect to that variable.
  - "How much would a small change in w_i influence the total loss function L?"
  - We express the slope as a partial derivate ∂ of the loss ∂w_i
- The gradient is then defined as a vector of these partials.

The gradient

- We'll represent $\hat{y}$ as $f(x;\theta)$ to make the dependence on θ more obvious:

$$\nabla_\theta L(f(x;\theta),y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

- The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla\ L(f(x;\theta),y)$$

What are these partial derivatives for logistic regression?

- The loss function

$$L_{BCE}(\hat{y}, y) = -[\mathrm{y} \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

The elegant derivative of this function

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

Deriving the gradient of the binary cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = \frac{\partial}{\partial w_j} - [\mathrm{y} \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

$$= -[\frac{\partial}{\partial w_j} \mathrm{y} \log \sigma(w \cdot x + b) + \frac{\partial}{\partial w_j} (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

Deriving the gradient of the cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = \frac{\partial}{\partial w_j} - [\mathrm{y} \log \sigma(w \cdot x + b) + (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

$$= -[\frac{\partial}{\partial w_j} \mathrm{y} \log \sigma(w \cdot x + b) + \frac{\partial}{\partial w_j} (1 - y) \log(1 - \sigma(w \cdot x + b))]$$

Using the chain rule $f(x) = u\big(v(x)\big)$, $\frac{df}{dx} = \frac{du}{dv}\frac{dv}{dx}$ and the derivative of log $\frac{d}{dx}\ln(x) = \frac{1}{x}$

$$\frac{\partial L_{\mathrm{BCE}}(\hat{y}, y)}{\partial w_j} = -\frac{y}{\sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} \sigma(w \cdot x + b) - \frac{1 - y}{1 - \sigma(w \cdot x + b)} \frac{\partial}{\partial w_j} 1 - \sigma(w \cdot x + b)$$

Rearranging terms:

$$\frac{\partial L_{\mathrm{BCE}}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y}{\sigma(w \cdot x + b)} - \frac{1 - y}{1 - \sigma(w \cdot x + b)}\right] \frac{\partial}{\partial w_j} \sigma(w \cdot x + b)$$

Deriving the gradient of the cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y}{\sigma(w \cdot x + b)} - \frac{1 - y}{1 - \sigma(w \cdot x + b)}\right] \frac{\partial}{\partial w_j} \sigma(w \cdot x + b)$$

Now plugging in the derivative of sigmoid $\frac{d\sigma(z)}{dz} = \sigma(z)(1\text{-}\sigma(z))$, and chain rule again $\frac{df}{dx} = \frac{du}{dv}\frac{dv}{dx}$

$$\frac{\partial L_{CE}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y - \sigma(w \cdot x + b)}{\sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)]}\right] \sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)] \frac{\partial (w \cdot x + b)}{\partial w_j}$$

Deriving the gradient of the cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y}{\sigma(w \cdot x + b)} - \frac{1 - y}{1 - \sigma(w \cdot x + b)}\right] \frac{\partial}{\partial w_j} \sigma(w \cdot x + b)$$

Now plugging in the derivative of sigmoid $\frac{d\sigma(z)}{dz} = \sigma(z)(1\text{-}\sigma(z))$, and chain rule again $\frac{df}{dx} = \frac{du}{dv}\frac{dv}{dx}$

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y - \sigma(w \cdot x + b)}{\sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)]}\right] \sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)] \frac{\partial (w \cdot x + b)}{\partial w_j}$$

$$\frac{\partial L_{BCE}(\hat{y}, y)}{\partial w_j} = -\left[\frac{y - \sigma(w \cdot x + b)}{\sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)]}\right] \sigma(w \cdot x + b)[1 - \sigma(w \cdot x + b)] x_j$$

Deriving the gradient of the cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y}{\sigma(w\cdot x+b)} - \frac{1-y}{1-\sigma(w\cdot x+b)}\right]\frac{\partial}{\partial w_j}\sigma(w\cdot x+b)$$

Now plugging in the derivative of sigmoid $\frac{d\sigma(z)}{dz}=\sigma(z)(1-\sigma(z))$, and chain rule again $\frac{df}{dx}=\frac{du}{dv}\frac{dv}{dx}$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]\frac{\partial(w\cdot x+b)}{\partial w_j}$$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]x_j$$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -[y-\sigma(w\cdot x+b)]x_j$$

Deriving the gradient of the cross-entropy loss function for logistic regression

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y}{\sigma(w\cdot x+b)} - \frac{1-y}{1-\sigma(w\cdot x+b)}\right]\frac{\partial}{\partial w_j}\sigma(w\cdot x+b)$$

Now plugging in the derivative of sigmoid $\frac{d\sigma(z)}{dz}=\sigma(z)(1-\sigma(z))$, and chain rule again $\frac{df}{dx}=\frac{du}{dv}\frac{dv}{dx}$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]\frac{\partial(w\cdot x+b)}{\partial w_j}$$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -\left[\frac{y-\sigma(w\cdot x+b)}{\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]}\right]\sigma(w\cdot x+b)[1-\sigma(w\cdot x+b)]x_j$$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = -[y-\sigma(w\cdot x+b)]x_j$$

$$\frac{\partial L_{BCE}(\hat{y},y)}{\partial w_j} = [\sigma(w\cdot x+b)-y]x_j$$

Working through an example

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x; w), y)$$

- One step of gradient descent
- A mini-sentiment example, where the true y=1 (positive)
- Two features:
  - x_1 = 3 (count of positive lexicon words)
  - x_2 = 2 (count of negative lexicon words)
- Assume 3 parameters (2 weights and 1 bias) in θ^0 are zero:

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$

Example of gradient descent

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$
$$x_1 = 3; x_2 = 2$$

Update step for update θ is :

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta), y)$$

where $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_\theta L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$
$$x_1 = 3; x_2 = 2$$

Update step for update θ is :

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta), y)$$

where $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_\theta L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

Example of gradient descent

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$
$$x_1 = 3; x_2 = 2$$

Update step for update θ is :

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta), y)$$

where $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_\theta L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix}$$

0.5

Example of gradient descent

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$
$$x_1 = 3; x_2 = 2$$

Update step for update θ is :

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x;\theta), y)$$

where $\frac{\partial L_{CE}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

Gradient vector has 3 dimensions:

$$\nabla_\theta L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$w_1 = w_2 = b = 0$$
$$\eta = 0.1$$
$$x_1 = 3; x_2 = 2$$

$$\nabla_\theta L(f(x;\theta), y) = \begin{bmatrix} \frac{\partial L_{CE}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{CE}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient vector, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.15 \\ 0.1 \\ 0.05 \end{bmatrix}$$

GD, SGD, min-batch GD

- In **Gradient Descent**, as we have seen earlier as well, we take the entire dataset > calculate the loss function > update parameter.
- In the case of **Stochastic Gradient Descent**, we update the parameters after every single observation and we know that every time the weights are updated it is known as an iteration.
- In the case of **Mini-batch Gradient Descent**, we take a subset of data and update the parameters based on every subset.

Gradient descent summary

- Choose a starting point
- Repeat to update the weight t = 1, 2, 3 …

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(\boldsymbol{x}; w), \boldsymbol{y})$$

  - The value of the gradient (slope in our all examples) $\frac{d}{dw} L(f(\boldsymbol{x}; w), \boldsymbol{y})$ weighted by a learning rate η
    - Gradient: a direction that increases the value
    - Learning rate: a hyper-parameter specifies the step length
      - Higher learning rate means move *w* faster

Function **Stochastic Gradient Descent***(L(), f(), x, y)* returns θ

where: L is the loss function

f is a function parameterized by θ

x is the set of training input $x_1, x_2, \ldots, x_m$

y is the set of training outputs (labels) $y_1, y_2, \ldots, y_m$

$\theta \leftarrow 0$ # We will discuss the weight initialization later

repeat til done

For each training tuple (x_i, y_i) (in **random order**)

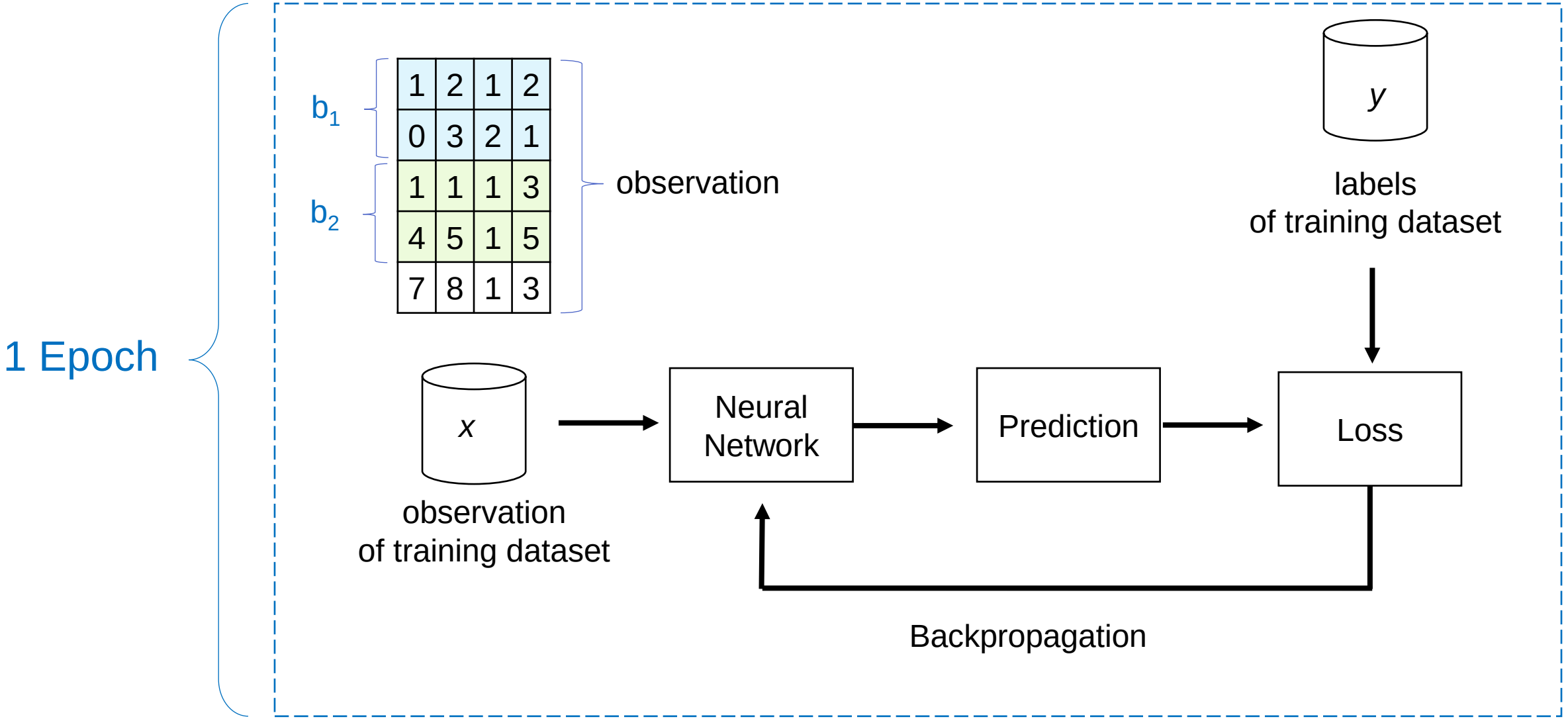
1. Compute $\hat{y}_i = f(x_i; \theta)$ # What is our estimated output ?

   Compute the loss $L(\hat{y}_i, y_i)$ # How far off is $\hat{y}_i$ from the true output y_i

2. $g \leftarrow \nabla_\theta L(f(x_i; \theta), y_i)$ # How should we move θ to maximize loss?
3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Training cycle



Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- That can result in choppy movements
- More common to compute gradient over mini-batches of training instances.
- Mini-batch training: *b* examples

Mini-batch Stochastic Gradient Descent (SGD)

- Randomly sample batch *b* example $x_1, x_2, \ldots, x_b$ to approximate the loss

$$\frac{1}{b}\sum_{i \in b} L(f(x;\theta)_i, y_i)$$

- *b* is the batch size, another important hyper-parameters.

Mini-batch SGD

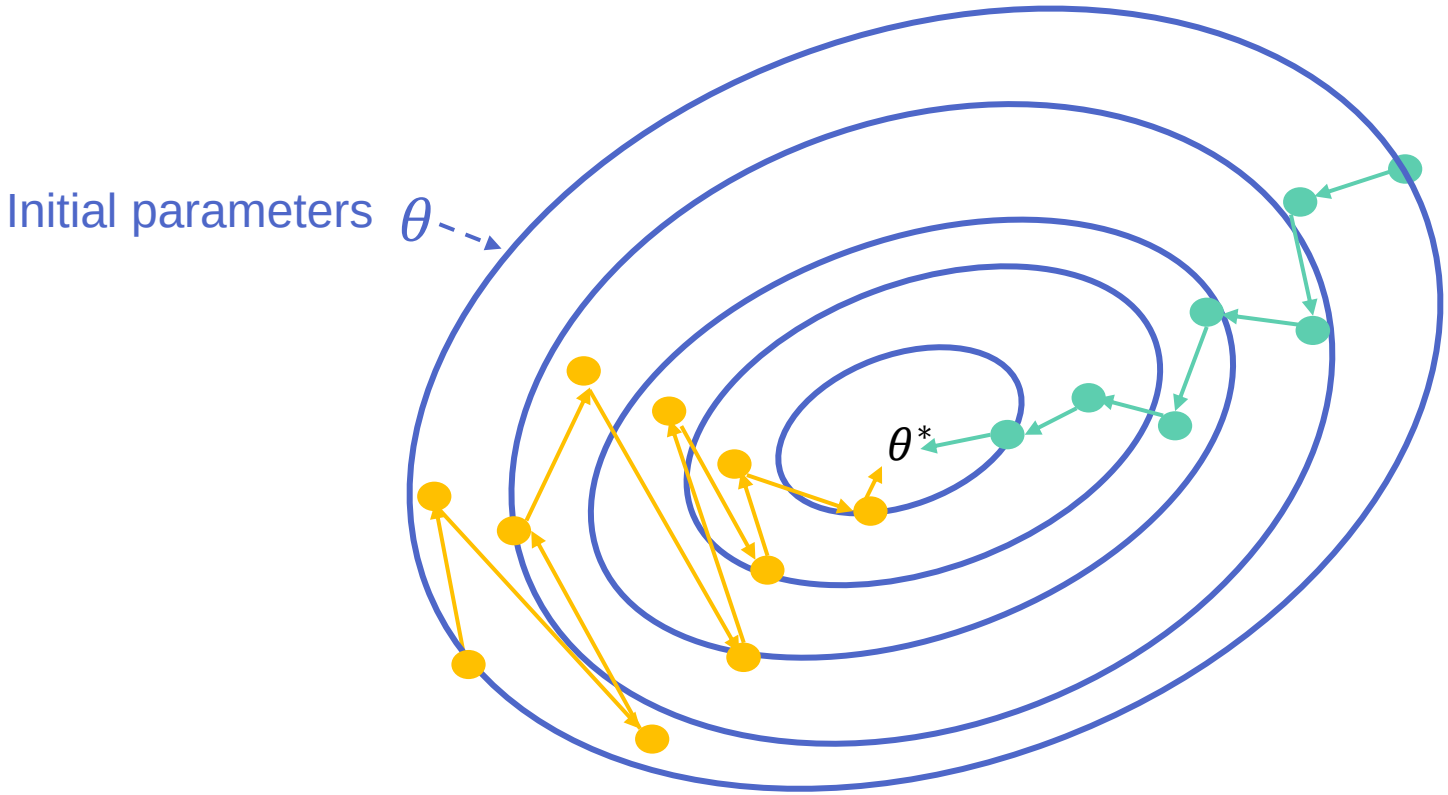
While stopping criterion not met do

Step 1. Sample a mini-batch of *m* examples from the training set $\{x^1, x^2, \ldots, x^m\}$ with corresponding targets $\{y^1, y^2 \ldots, y^m\}$

Step 2. Compute gradient estimate: $g_t = \nabla \left(\frac{1}{m} \sum_{i \in b} L(f(x;\theta)_i, y_i) \right)$

Step 3. Apply update: $\theta_{t+1} = \theta_t - \eta g_t$

SGD vs. mini-batch SGD



Choose a batch size

Not too small

Workload is too small (b=2, 4, 8, …etc), hard to fully utilize computation resources.

Not too big

Memory issue

Epochs, Batch, Iteration

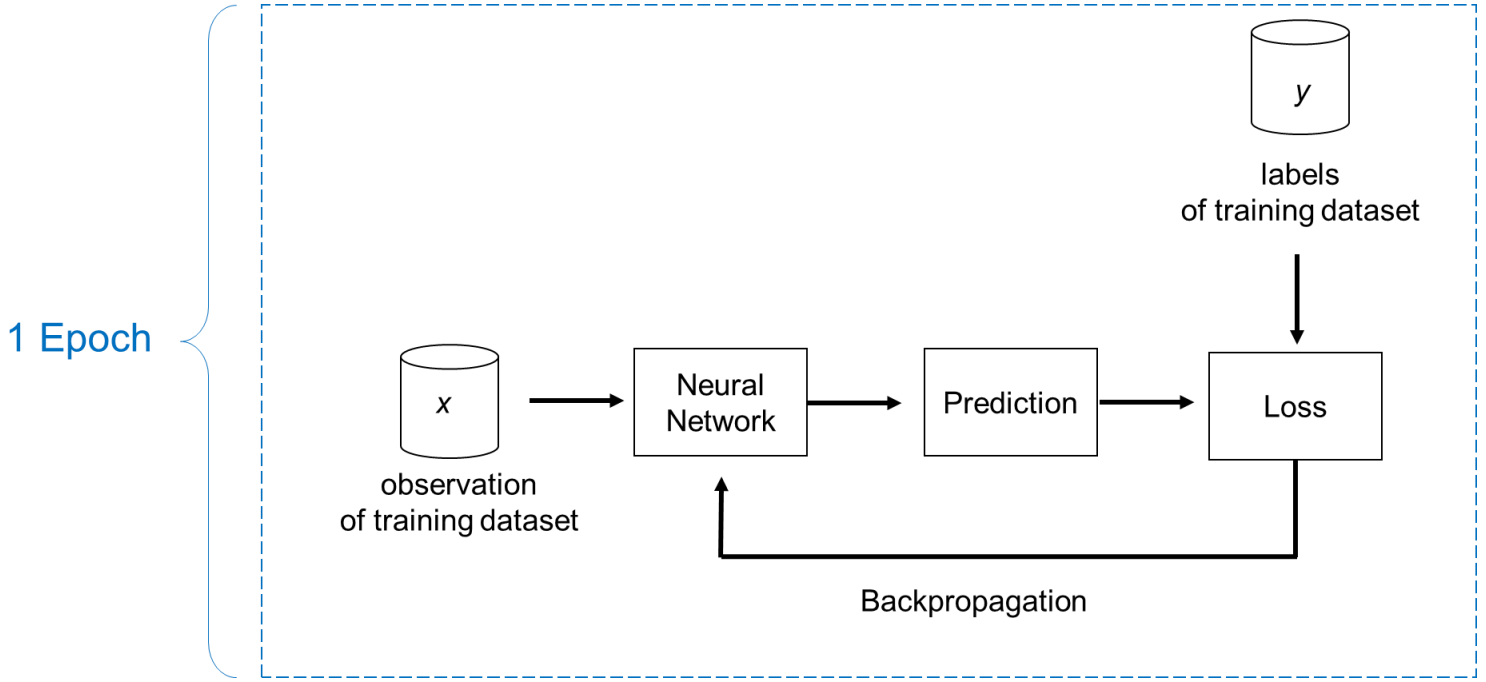
- Epoch:
  - One epoch is finished when the network has seen the whole dataset
- Batch size:
  - Number of training examples in one forward/backward pass.
- Number of iterations:
  - Number of passes, each pass using [batch size] number of examples.
  - If you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

Shuffling data up at the beginning of an epoch

- Normally, we would **shuffle up** all of the examples in the training set and then **portion** them off into batches.
  - Then, parameters are updated based on **the gradient of the loss function with respect to each batch**, within one epoch.
- This gives us a more complete sampling of batch gradients and improves the stochastic estimation of the optimal gradient, when examples are put into batches with different examples than they were matched with in the previous epoch.

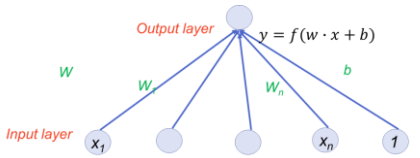
Outline

- Recap loss functions
- Gradient descent
- Backward Propagation

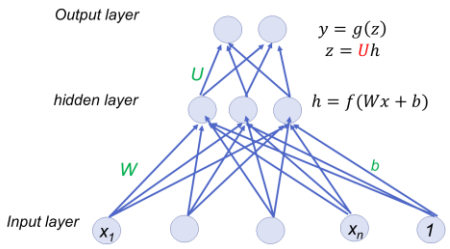


How can I find that gradient for every weight in the network?

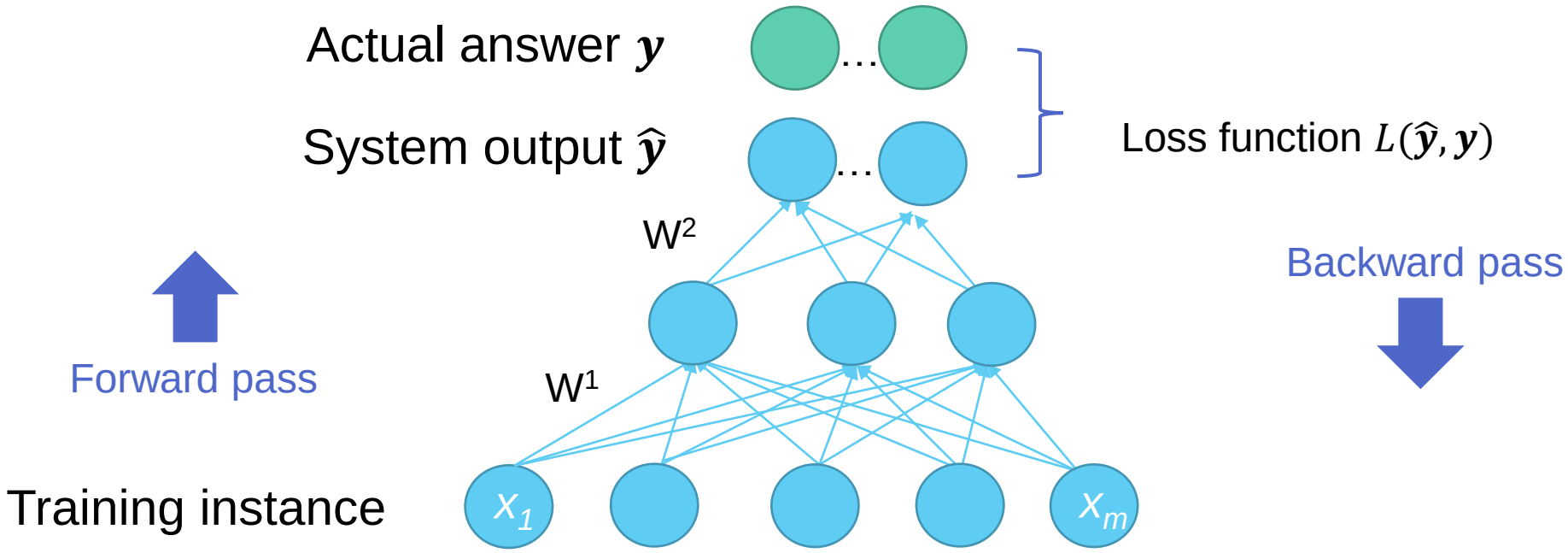
- These derivatives on the prior slide only give the updates for one weight layer: the last one!



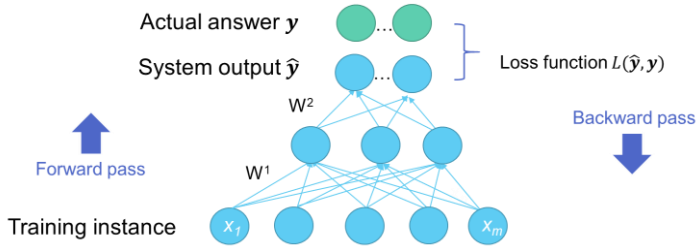
- What about deeper networks?
  - Lots of layers, different activation functions?
- Solution:
  - Even more use of the chain rule!!
  - Computation graphs and backward differentiation!



Intuition: training a 2-layer network



Intuition: Training a 2-layer network



For every training tuple (x, y)

- Run *forward* computation to find out estimate $\hat{y}$
- Run *backward* computation to update weights:
  - For every output node
    - Compute loss *L* between true y and the estimated $\hat{y}$
    - For every weight w^2 from hidden layer to the output layer
      - Update the weight
  - For every hidden node
    - Assess how much blame it deserves for the current answer
    - For every weight w^1 from input layer to the hidden layer
      - Update the weight

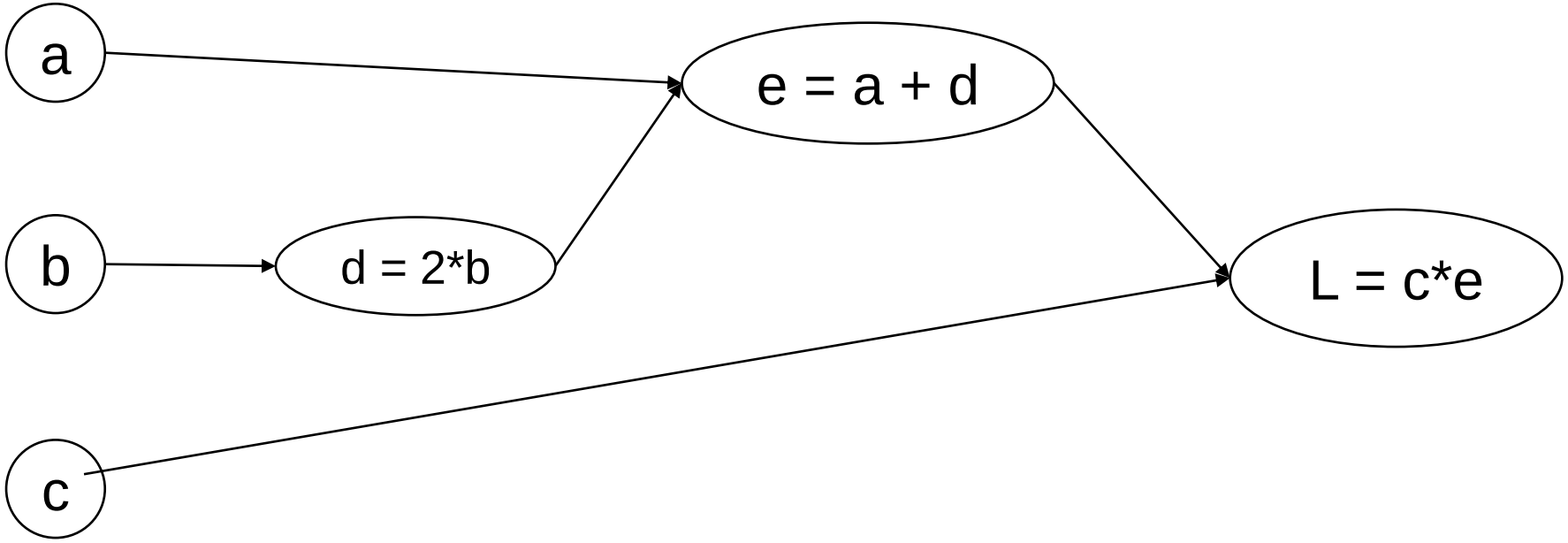
Why Computation Graphs

- For training, we need the derivative of the loss with respect to each weight in every layer of the network.
  - But the loss is computed only at the very end of the network!
- Solution: error backpropagation
  - Which relies on computation graphs.
- Computation Graphs
  - A computation graph represents the process of computing a mathematical expression.
    - Node: variable (scalar, vector …)
    - Edge: operation (simple function)

Example: $L(a, b, c) = c \times (a+2b)$

Computations:

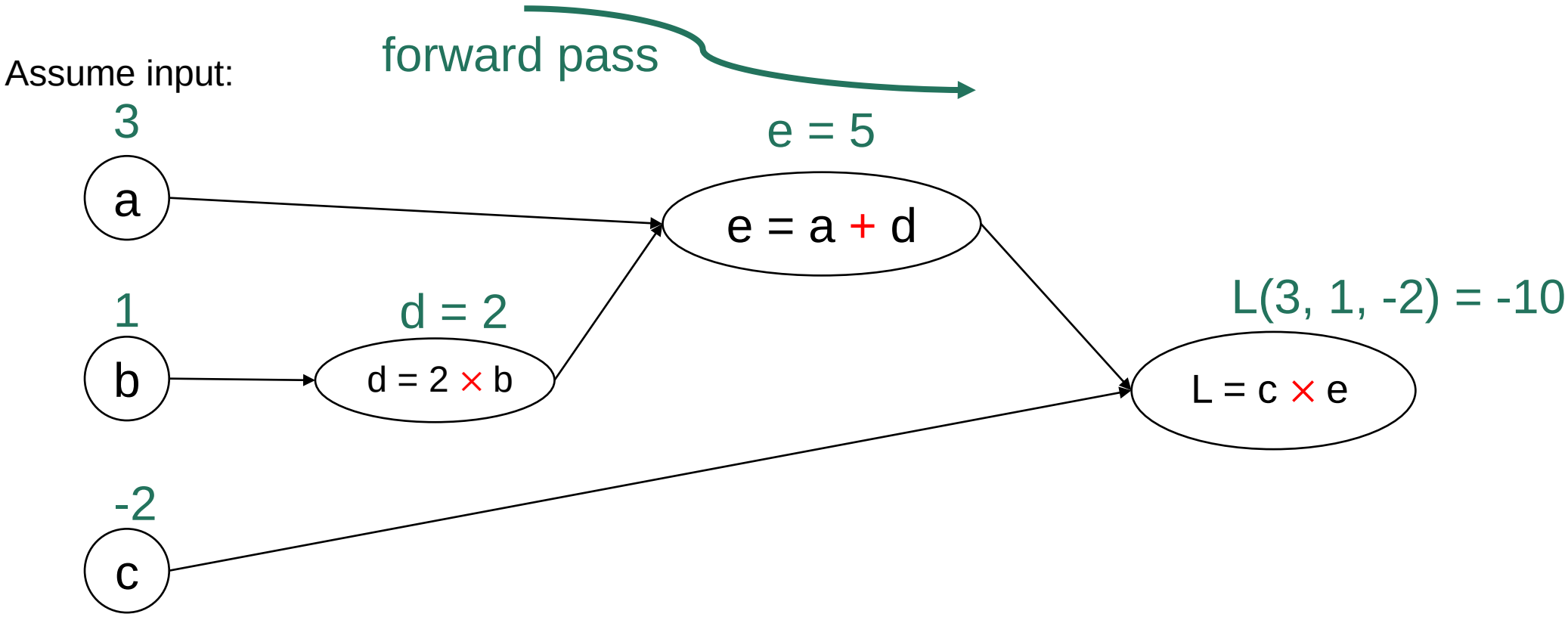
$$\begin{cases} d = 2 \times b \\ e = a + d \\ L = c \times e \end{cases}$$



Example: *L(a, b, c) = c(a+2b)*

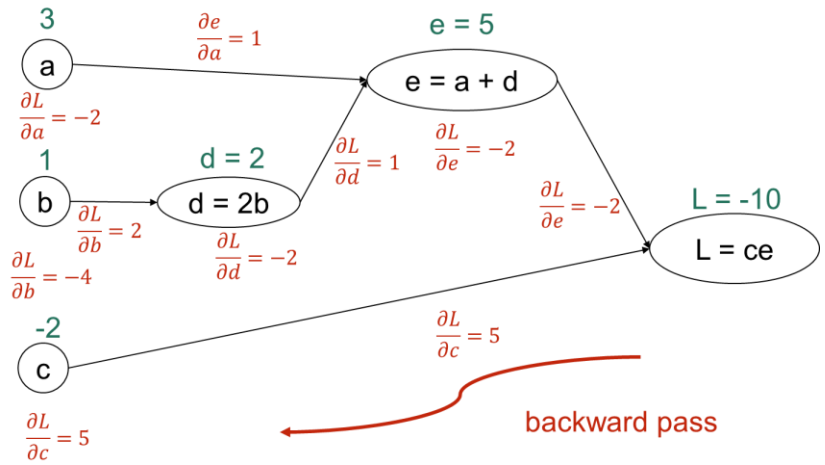
Computations:

$$\begin{cases} d = 2 \times b \\ e = a + d \\ L = c \times e \end{cases}$$



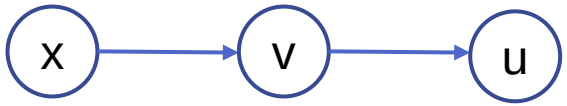
Backwards differentiation in computation graphs

- The importance of the computation graph comes from the backward pass
- This is used to compute the derivatives that we'll need for the weight update.

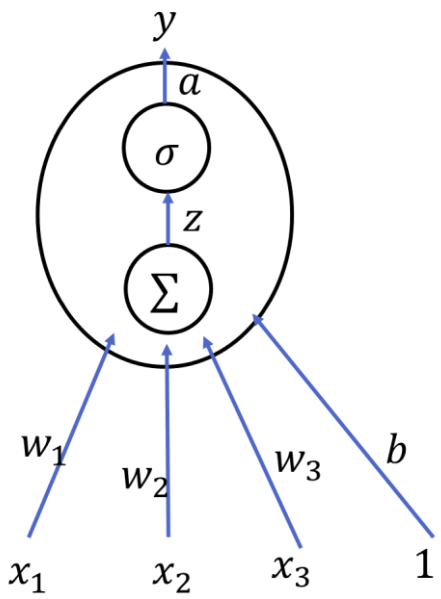


Where did that derivative come from?

Using the chain rule! $f(x) = u\big(v(x)\big)$ $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$



Intuition



Derivative of the weighted sum

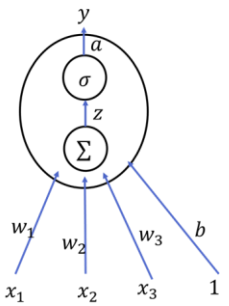
Derivative of the Activation

Derivative of the Loss

$$\frac{\partial \mathrm{L}}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}$$

Example: *L(a, b, c) = c(a+2b)*

$$d = 2 \times b$$
$$e = a + d$$
$$L = c \times e$$



Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial \mathrm{L}}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}$$

- We want: $\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}, \text{and } \frac{\partial L}{\partial c}$

- The derivative $\frac{\partial L}{\partial a}$, tells us how much a small change in a affects L.

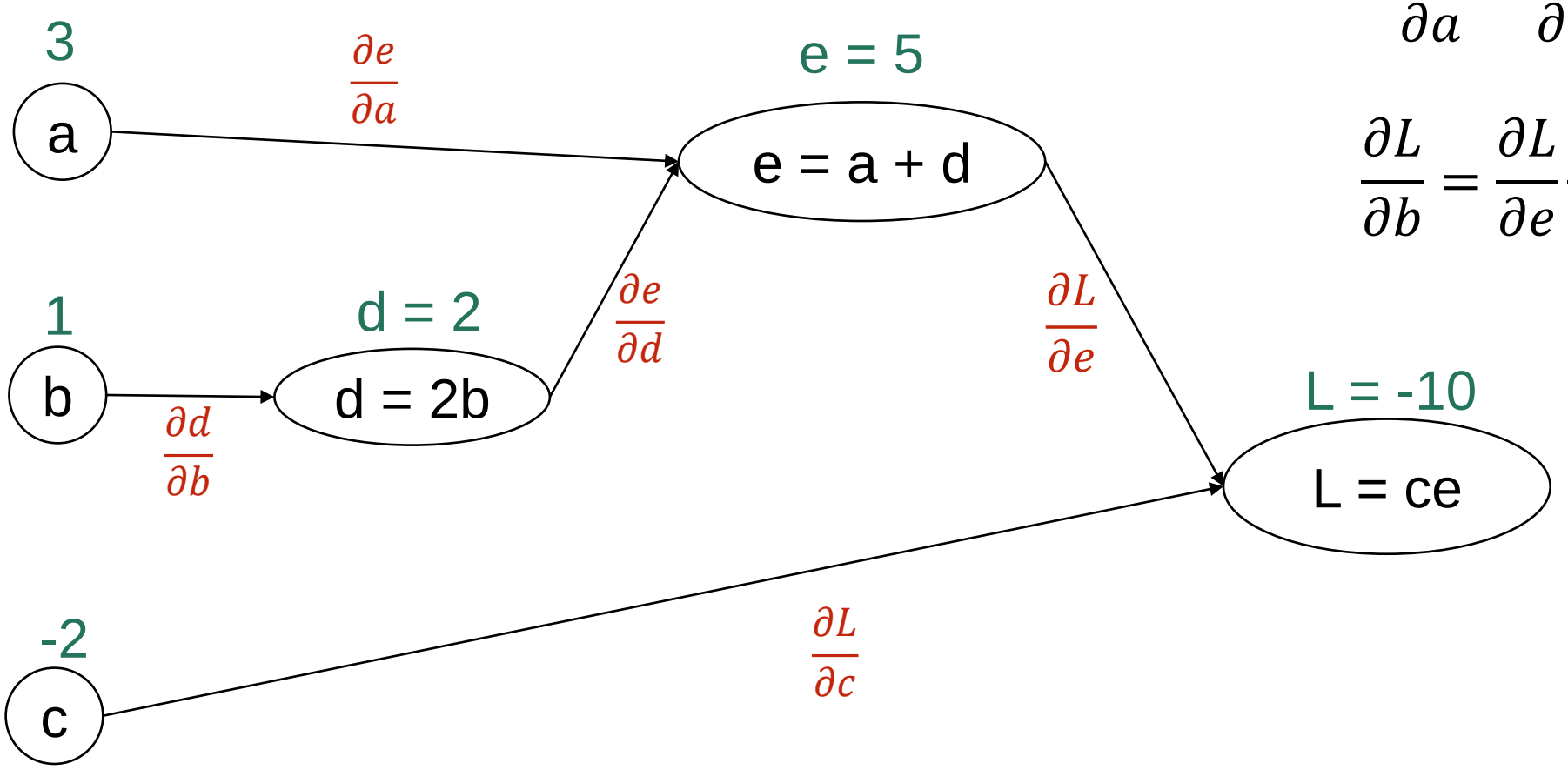
Example

$$d = 2 \times b$$
$$e = a + d$$
$$L = c \times e$$

$$\frac{\partial L}{\partial c} = e$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$



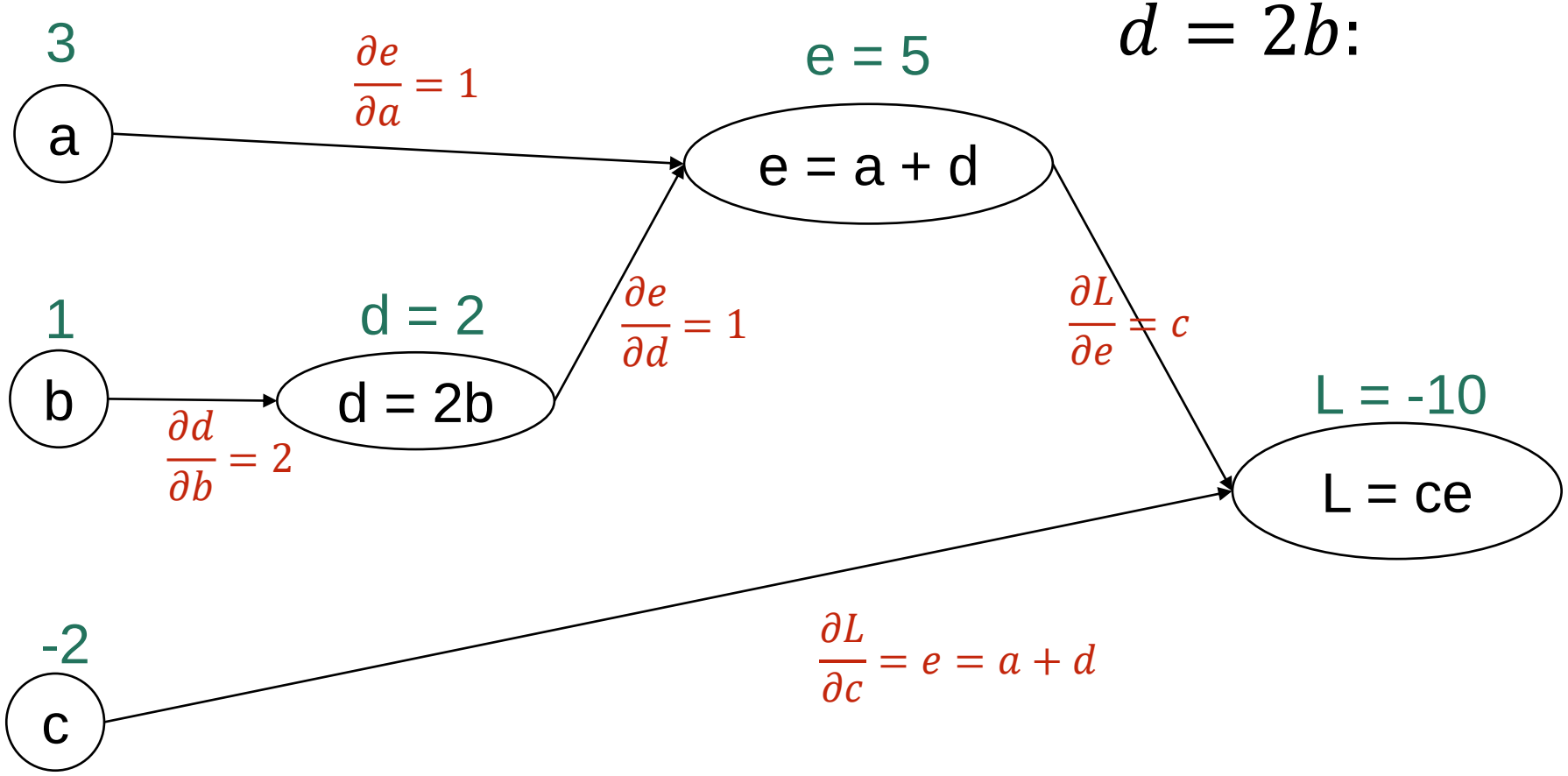
Example

$$d = 2 \times b$$
$$e = a + d$$
$$L = c \times e$$

$L = ce$: $\frac{\partial L}{\partial e} = c$, $\frac{\partial L}{\partial c} = e$

$e = a + d$: $\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$

$d = 2b$: $\frac{\partial d}{\partial b} = 2$



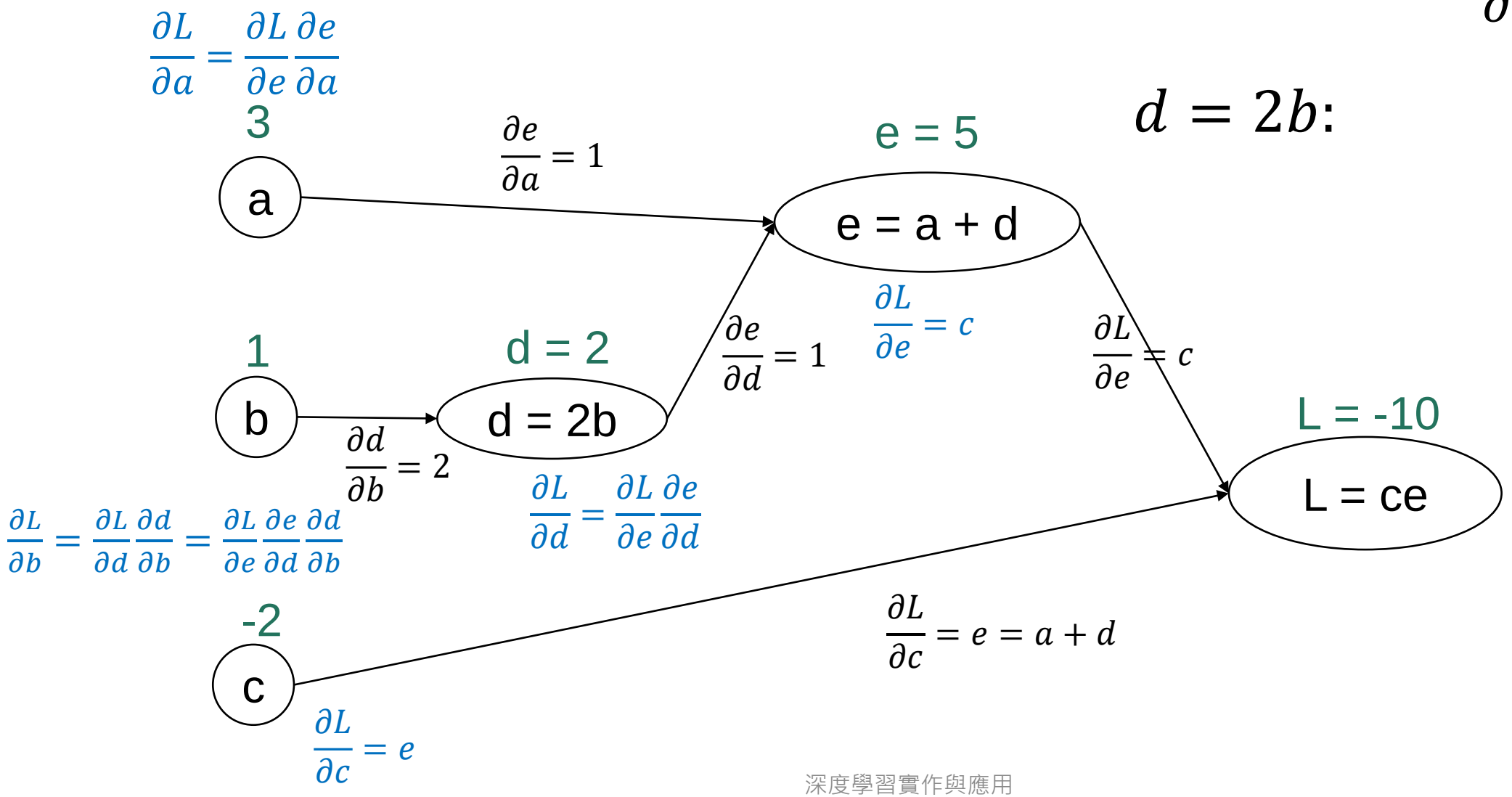
Example

$$d = 2 \times b$$
$$e = a + d$$
$$L = c \times e$$

$L = ce$: $\frac{\partial L}{\partial e} = c$, $\frac{\partial L}{\partial c} = e$

$e = a + d$: $\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$

$d = 2b$: $\frac{\partial d}{\partial b} = 2$



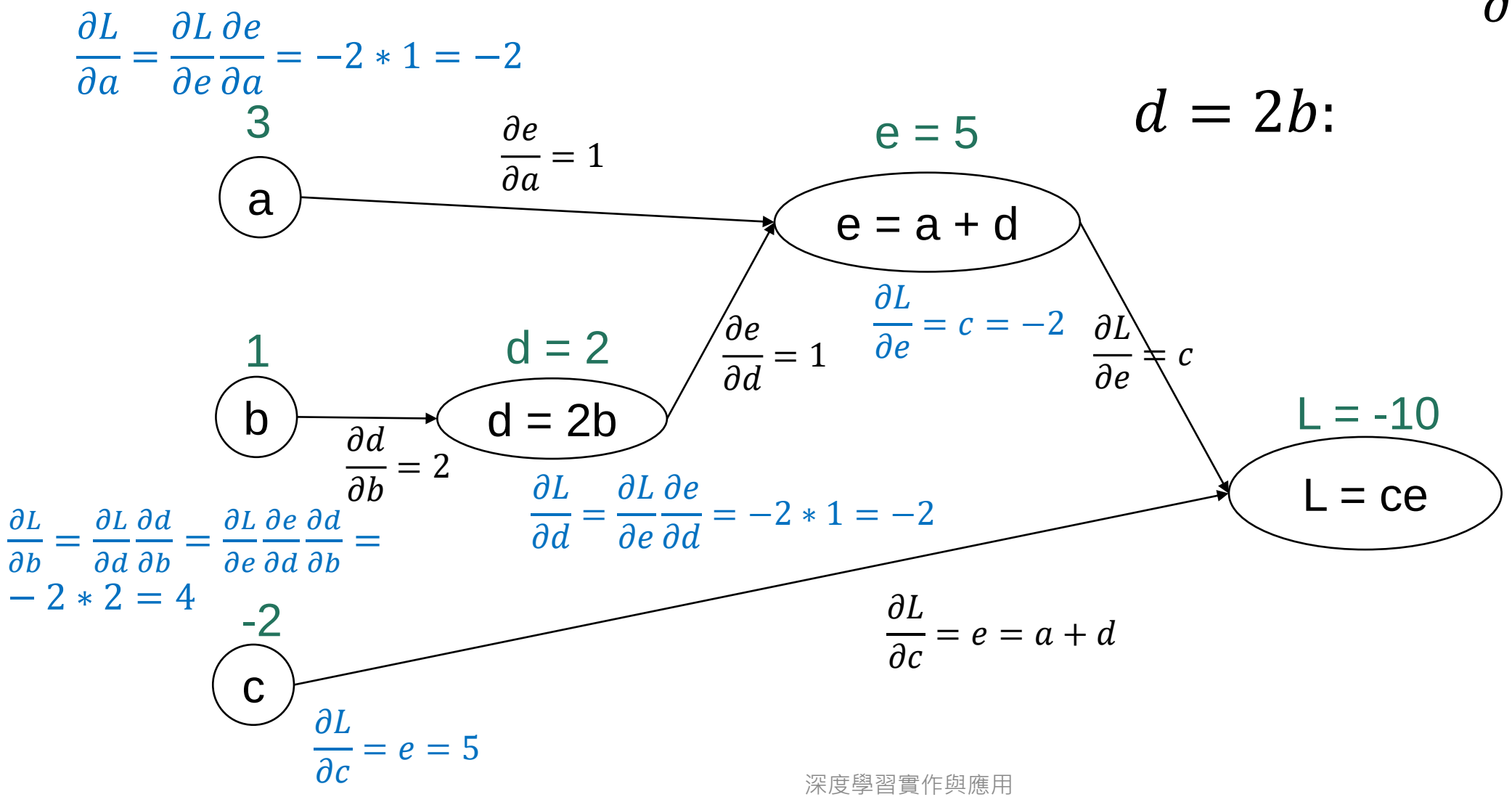
Example

$$d = 2 \times b$$
$$e = a + d$$
$$L = c \times e$$

$L = ce$: $\frac{\partial L}{\partial e} = c$, $\frac{\partial L}{\partial c} = e$

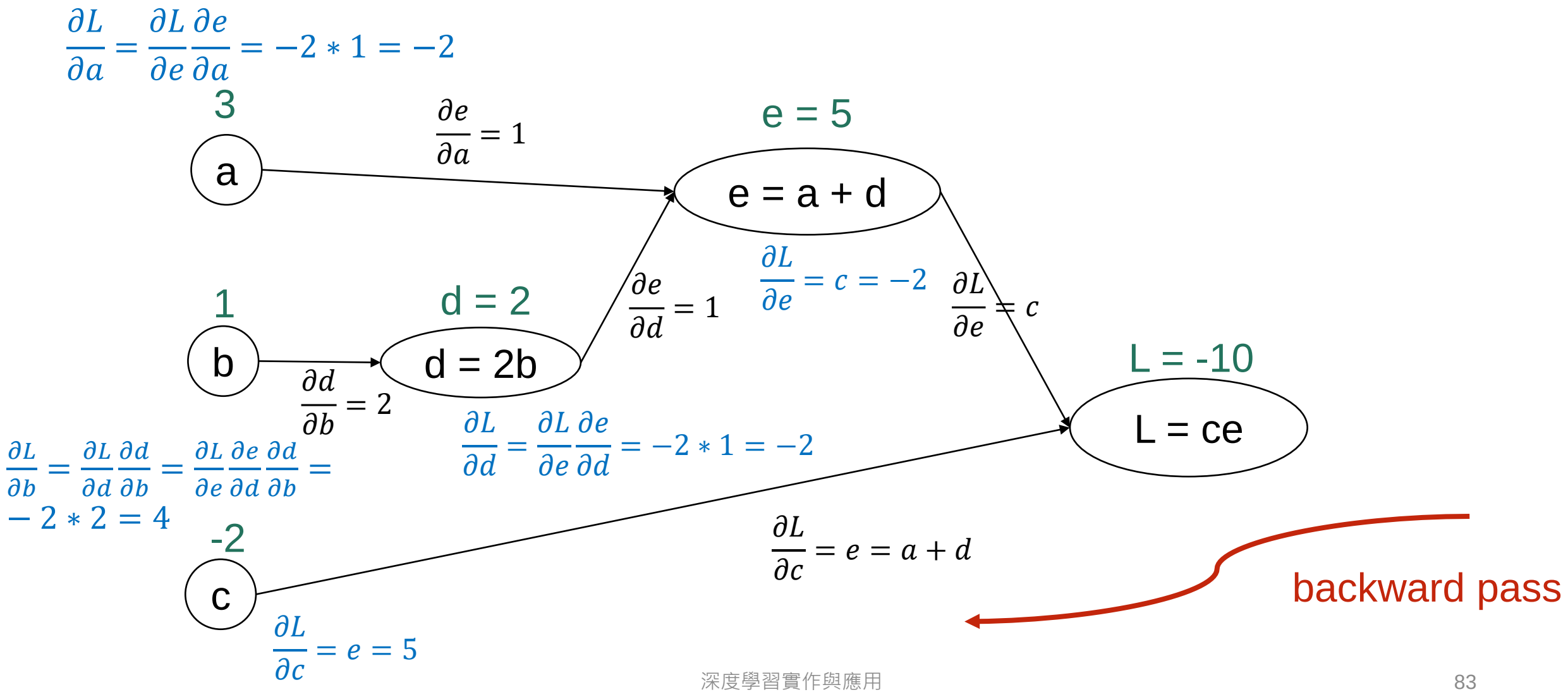
$e = a + d$: $\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$

$d = 2b$: $\frac{\partial d}{\partial b} = 2$

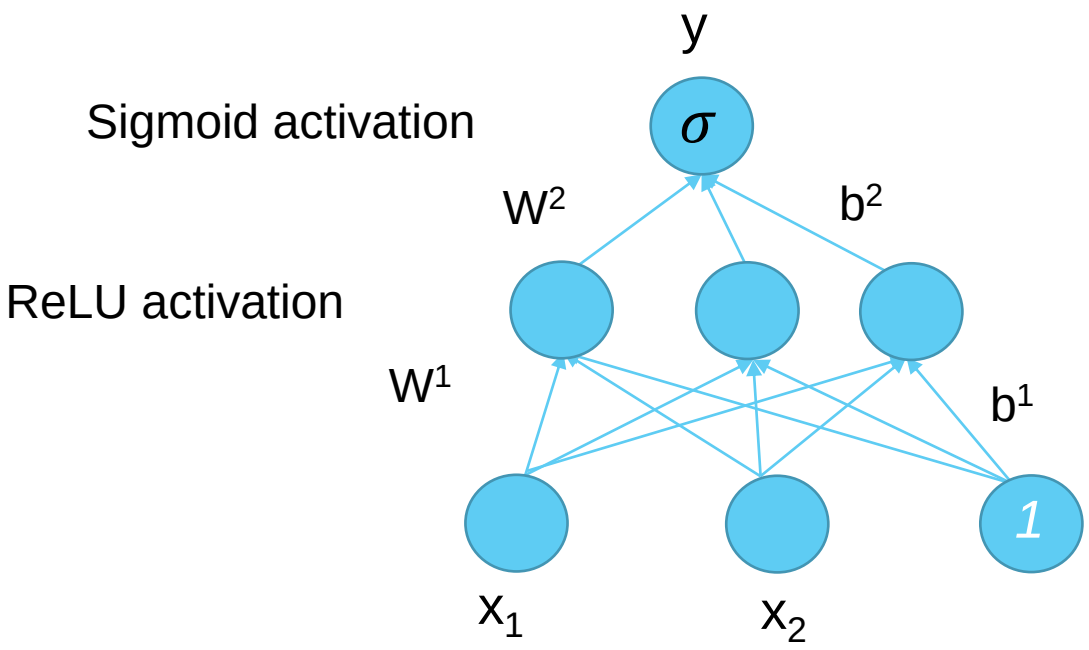


Example

The derivative $\frac{\partial L}{\partial c}$, tells us how much a small change in c affects L.



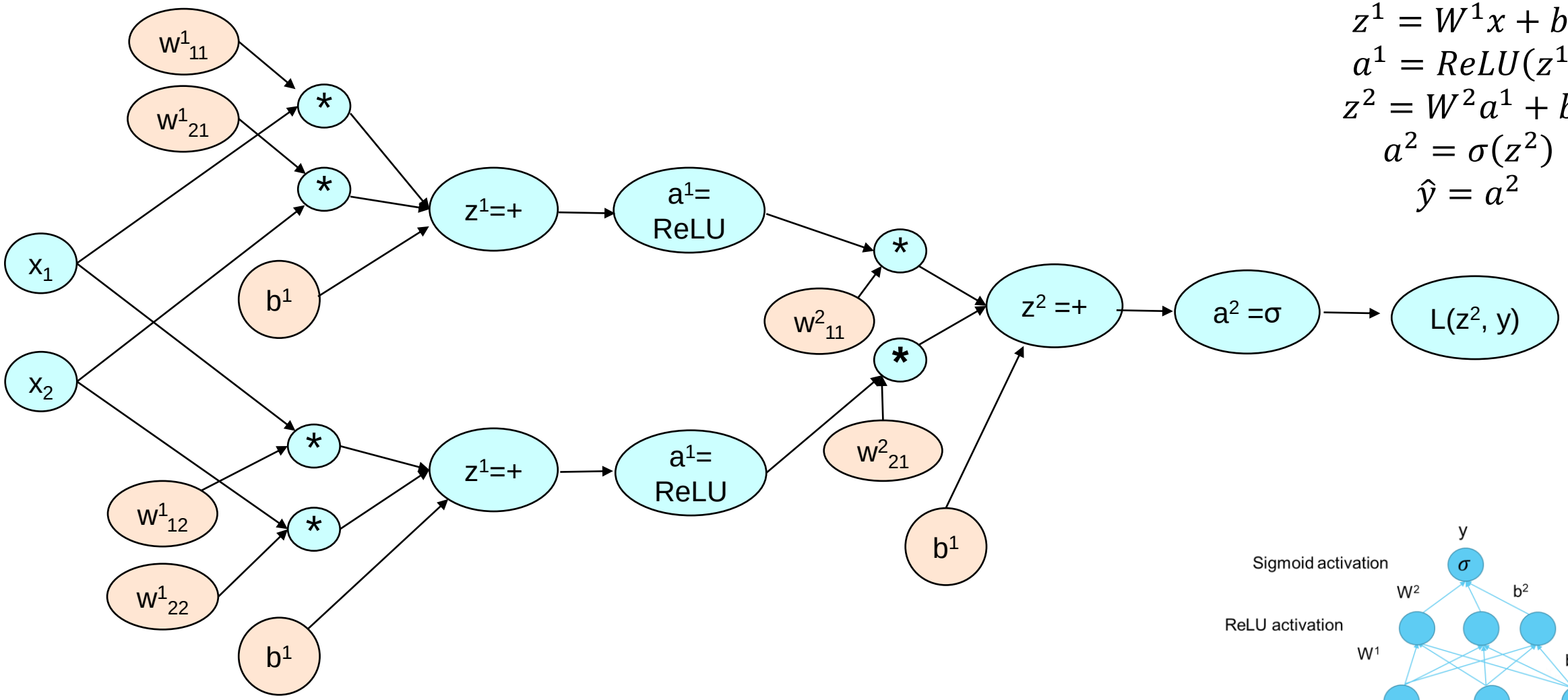
Backward differentiation on a two layer network



$$z^1 = W^1 x + b^1$$
$$a^1 = ReLU(z^1)$$
$$z^2 = W^2 a^1 + b^2$$
$$a^2 = \sigma(z^2)$$
$$\hat{y} = a^2$$

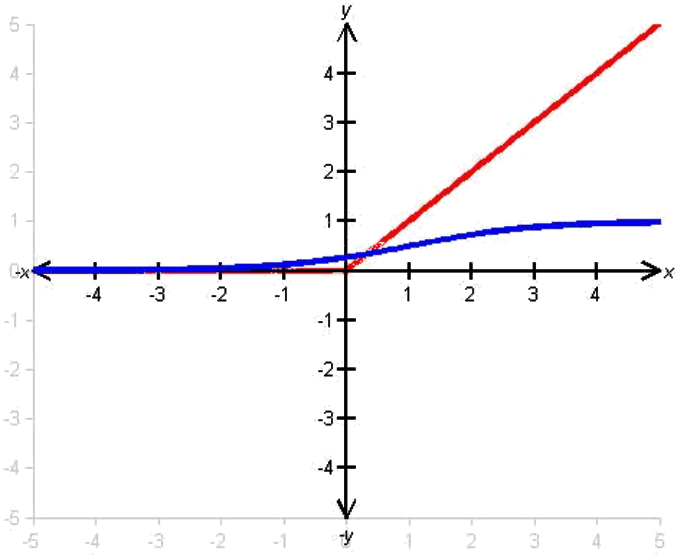
Backward differentiation on a 2-layer network

$$z^1 = W^1 x + b^1$$
$$a^1 = ReLU(z^1)$$
$$z^2 = W^2 a^1 + b^2$$
$$a^2 = \sigma(z^2)$$
$$\hat{y} = a^2$$



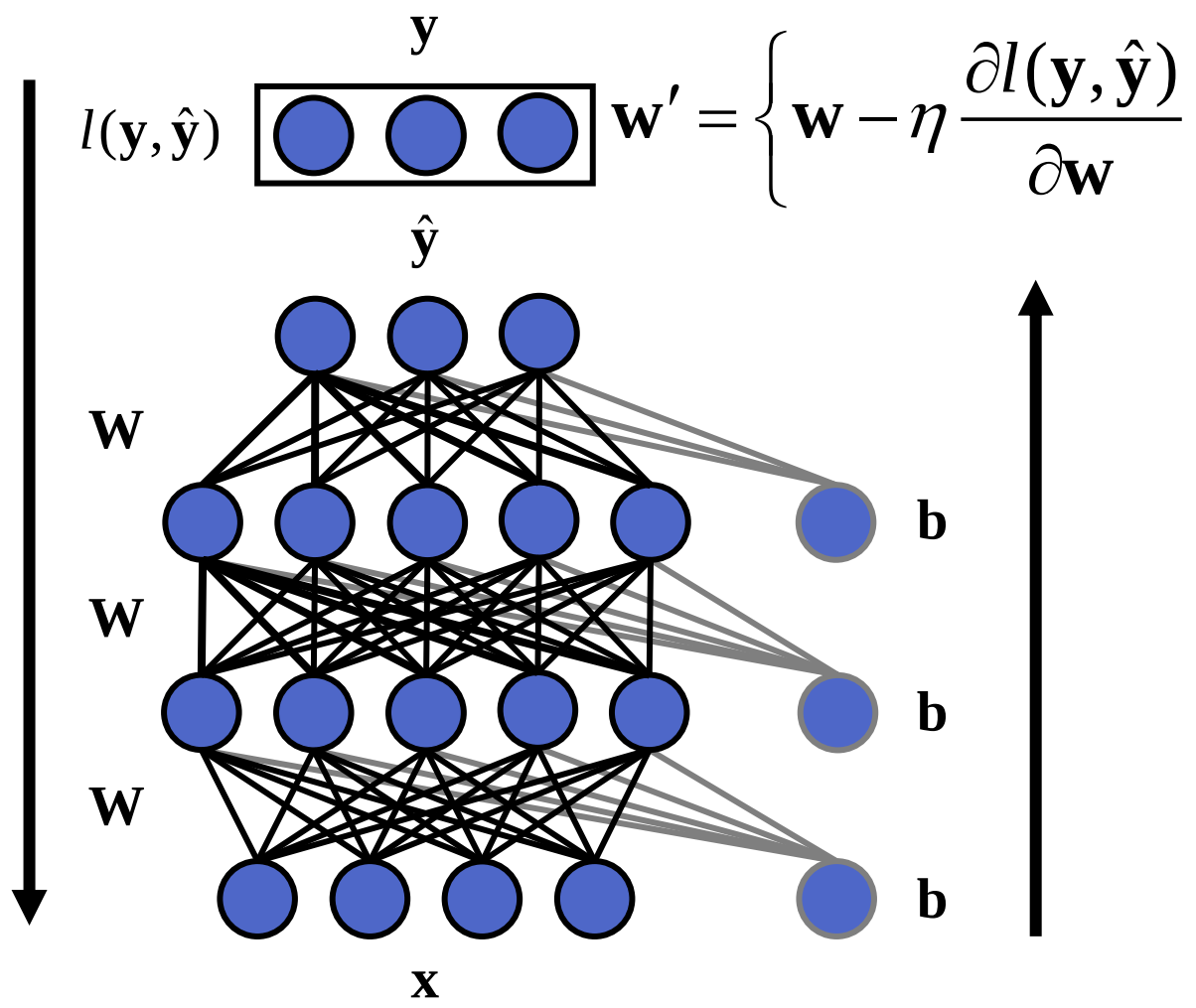
$ReLU$

- ReLU is an important part of the success story of modern neural networks.
  - The derivative of the output with respect to the input is always one for inputs greater than zero.
  - This contributes to the stability and efficiency of training and contrasts with the derivatives of sigmoid activation functions, which saturate (become close to zero) for large positive and large negative inputs.



Backpropagation

1. Design Network
2. Initialize Parameters
3. Feedforward
4. Feedback
5. Adjust Weights
6. Check Convergence



Backpropagation

$$\mathbf{w}' = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\mathbf{w} \cdots (\mathbf{w}\mathbf{x} + \mathbf{b}) \cdots + \mathbf{b}) + \mathbf{b}) + \mathbf{b})}{\partial \mathbf{w}}$$

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times (\cdots)$$

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{b}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{b}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{b})}{\partial \mathbf{b}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times 1$$

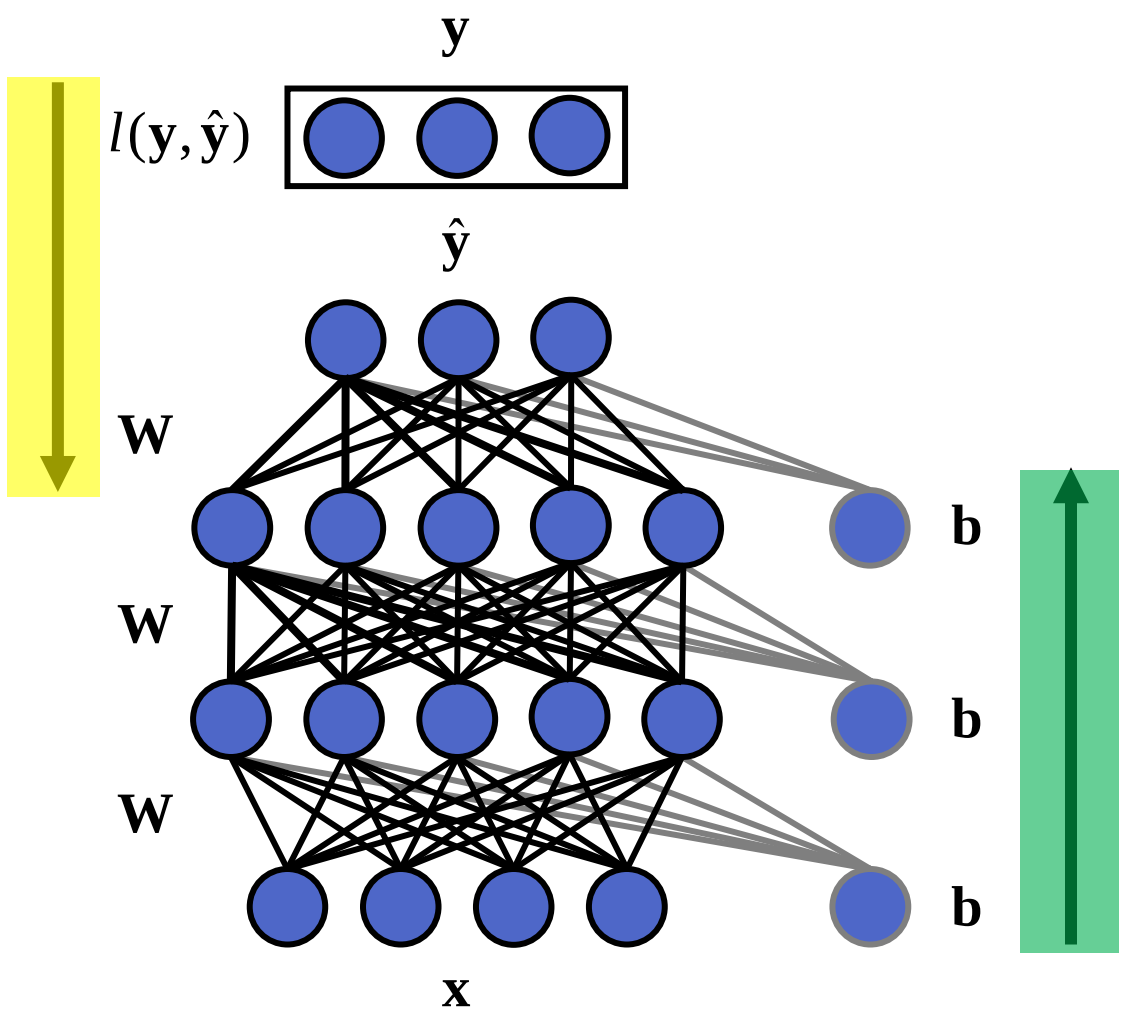
Backpropagation

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\cdots)+\mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots)+\mathbf{b})' \times \frac{\partial(\mathbf{w}(\cdots)+\mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots)+\mathbf{b})' \times \frac{\partial(\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots)+\mathbf{b})' \times (\cdots)$$



Backpropagation

$$\mathbf{w}' = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\mathbf{w} \cdots (\mathbf{w}\mathbf{x} + \mathbf{b}) \cdots + \mathbf{b}) + \mathbf{b}) + \mathbf{b})}{\partial \mathbf{w}}$$

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))' \times \frac{\partial (\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))' \times (\cdots)$$

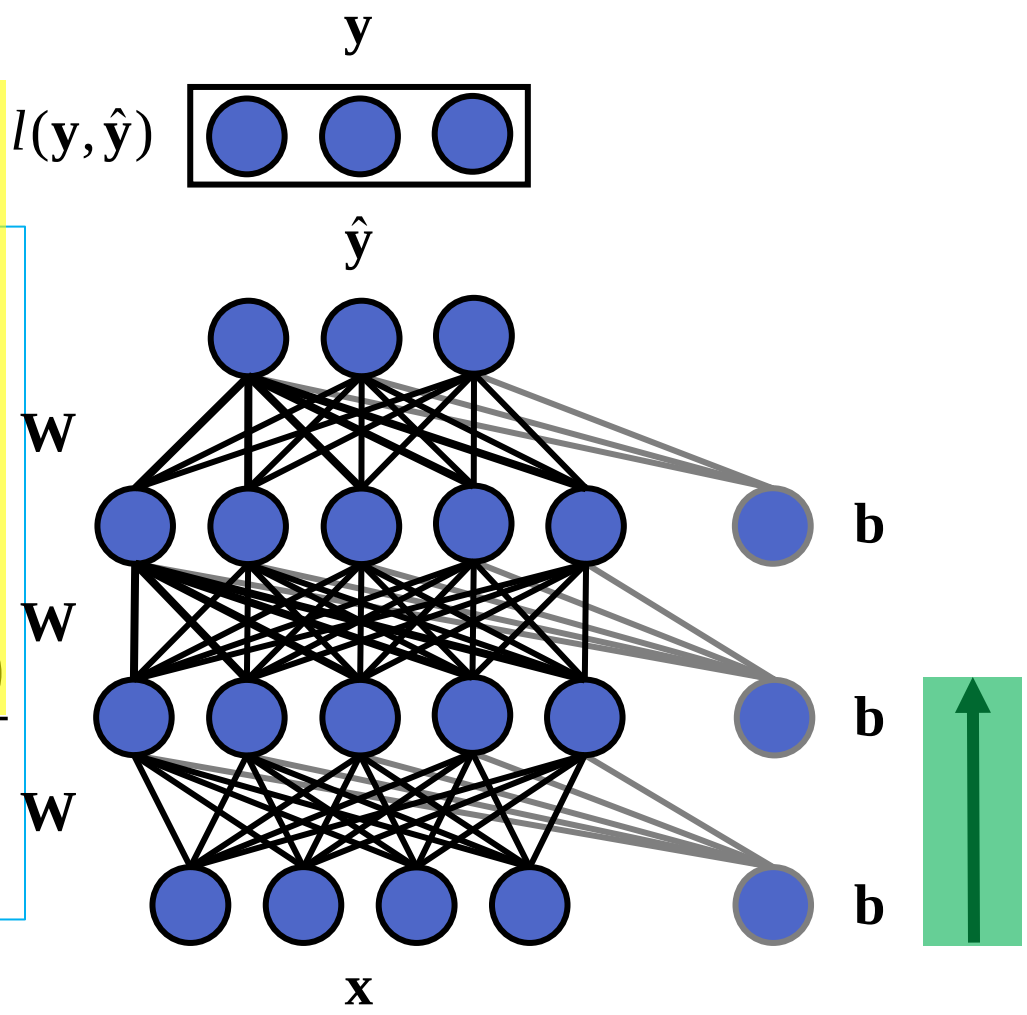
Backpropagation

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b})+\mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b})+\mathbf{b})' \times \frac{\partial(\mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b}))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b})+\mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b}))' \times \frac{\partial(\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b})+\mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots)+\mathbf{b}))' \times (\cdots)$$



Backpropagation

$$\mathbf{w}' = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} = \mathbf{w} - \eta \frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\mathbf{w} \cdots (\mathbf{w}\mathbf{x} + \mathbf{b}) \cdots + \mathbf{b}) + \mathbf{b}) + \mathbf{b})}{\partial \mathbf{w}}$$

$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\cdots) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\cdots) + \mathbf{b})' \times (\cdots)$$

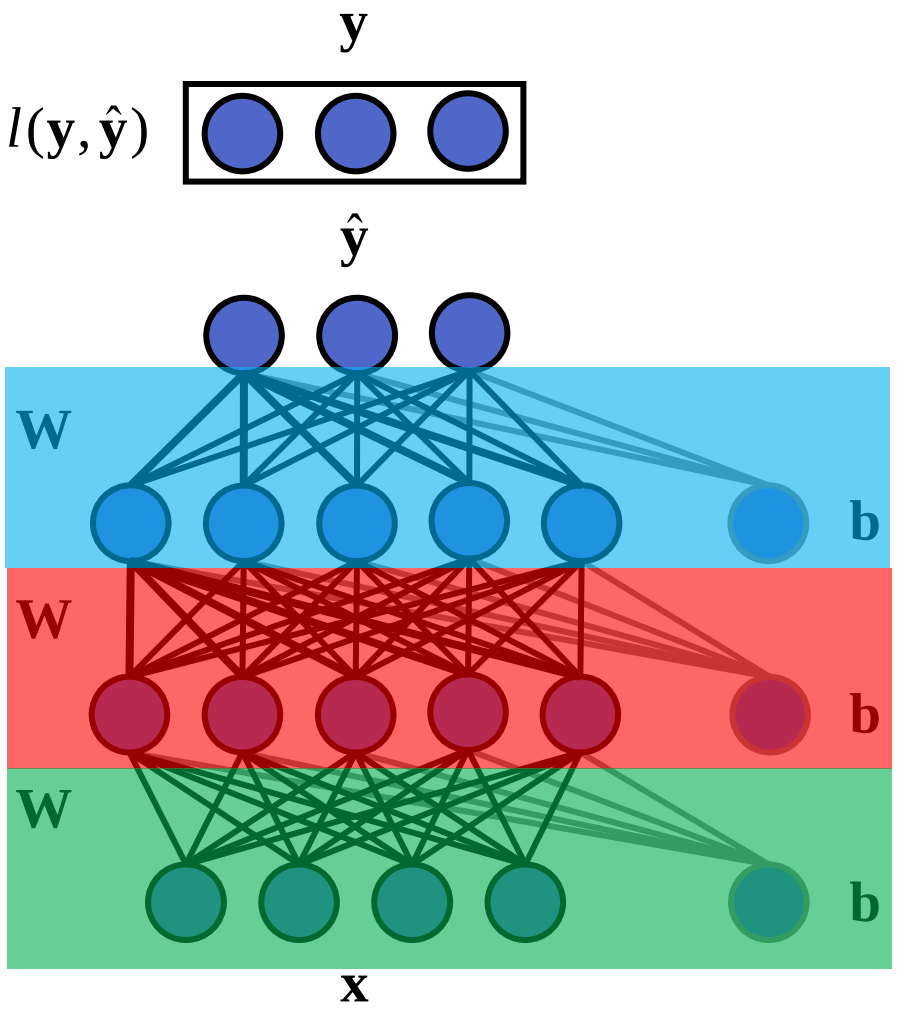
$$\frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times \frac{\partial (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))' \times \frac{\partial (\mathbf{w}(\cdots))}{\partial \mathbf{w}}$$

$$= l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}) + \mathbf{b})' \times (\mathbf{w}(\mathbf{w}(\cdots) + \mathbf{b}))' \times (\cdots)$$

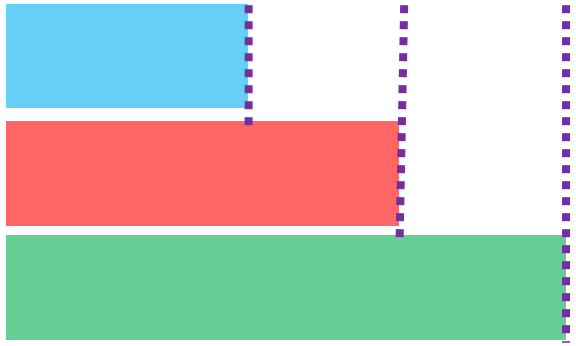
Backpropagation



$$\frac{\partial l(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{w}}$$

$$= \frac{\partial l(\mathbf{y}, \mathbf{w}(\mathbf{w}(\mathbf{w}\cdots(\mathbf{w}\mathbf{x}+\mathbf{b})\cdots+\mathbf{b})+\mathbf{b})+\mathbf{b})}{\partial \mathbf{w}}$$

$$= \frac{\partial l(\mathbf{y},\hat{\mathbf{y}})}{\partial \mathbf{w}} \times \frac{\partial \mathbf{w}}{\partial \mathbf{w}} \times \frac{\partial \mathbf{w}}{\partial \mathbf{w}} \times \frac{\partial \mathbf{w}}{\partial \cdots} \times \cdots \times \frac{\partial \cdots}{\partial \mathbf{w}}$$



Assignment 1

- To be announced on this Friday.
- Will be explained next week.

- Practice Pytorch now!