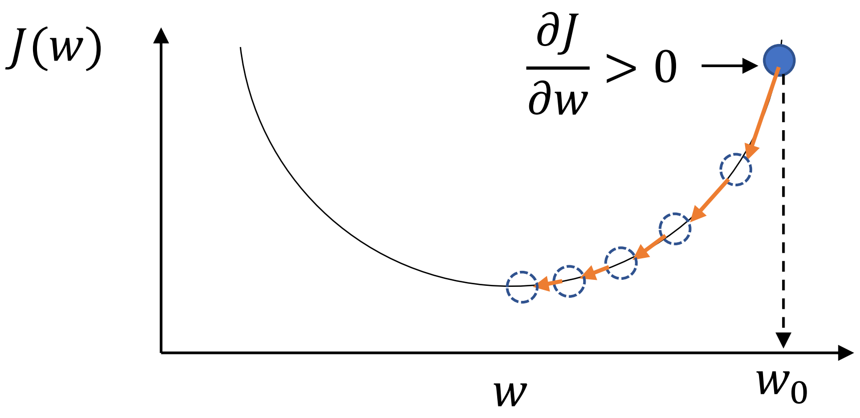
Basics of neural networks

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



I. calculate the slope $\frac{\partial J}{\partial w}$ corresponds to an initial w

II. Adjust w according to the local slope

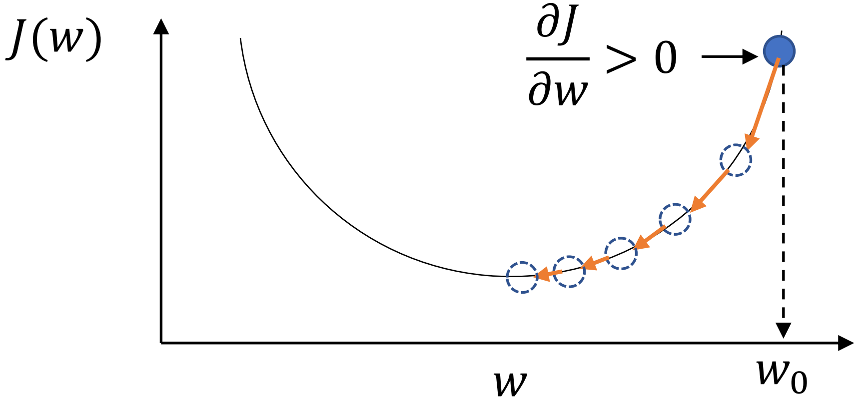
$w_{new} = w_{old} - \alpha \frac{\partial J}{\partial w}$, $\alpha > 0$ is the learning rate

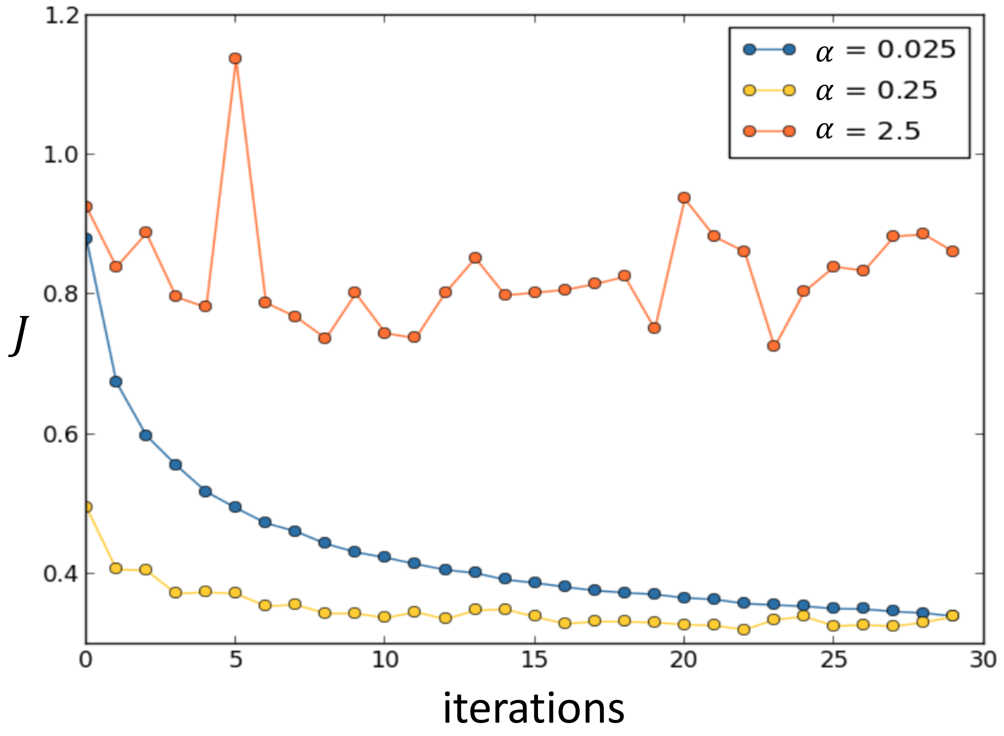
III. Iterate until $\frac{\partial J}{\partial w} = 0$

https://developers.google.com/machine-learning/crash-course/fitter/graph

How to find w and b? Gradient descent

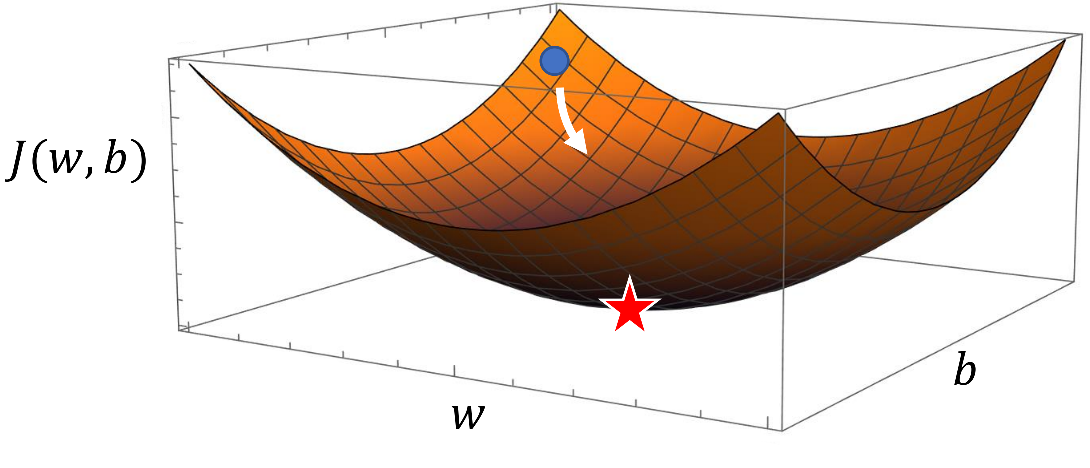
- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$





How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

- $-\nabla J$ gives the direction of the steepest decrease of J

$$-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

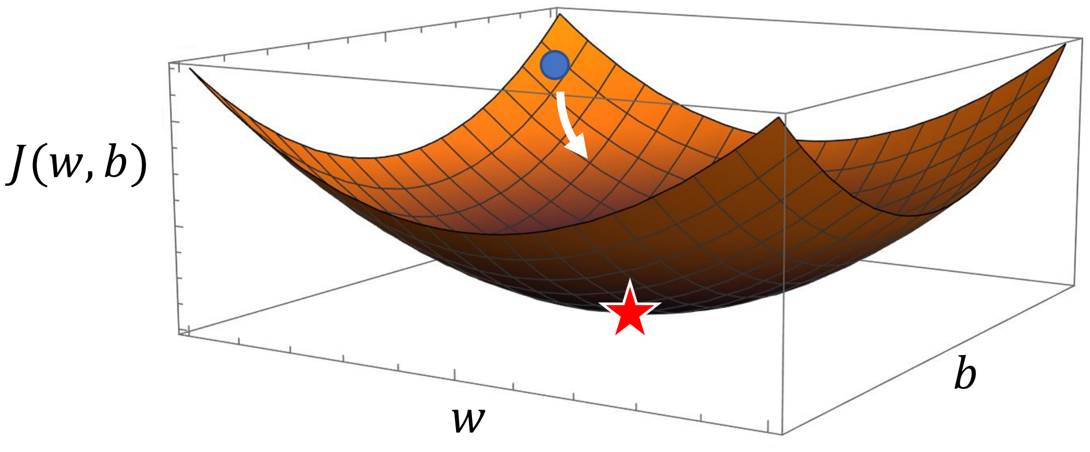
e.g. $-\nabla J(w_0, b_0) = -(10{,}1)$

Changing w reduces J 10 times faster than changing b

- $-\nabla J$ tells you which weights and biases reduce cost function J the fastest!

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

- $-\nabla J$ gives the direction of the steepest decrease of J

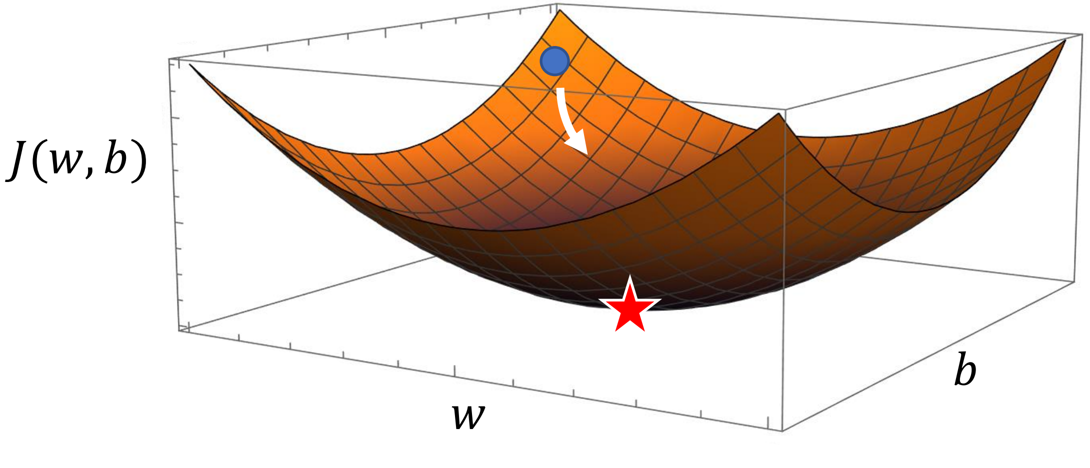
$$-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

- $(w_{new}, b_{new}) = (w_{old}, b_{old}) - \alpha\, \nabla J(w, b)$, α is the learning rate
- Iterate until $\nabla J = 0$

https://www.deeplearning.ai/ai-notes/optimization/

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2$



Gradient on a surface

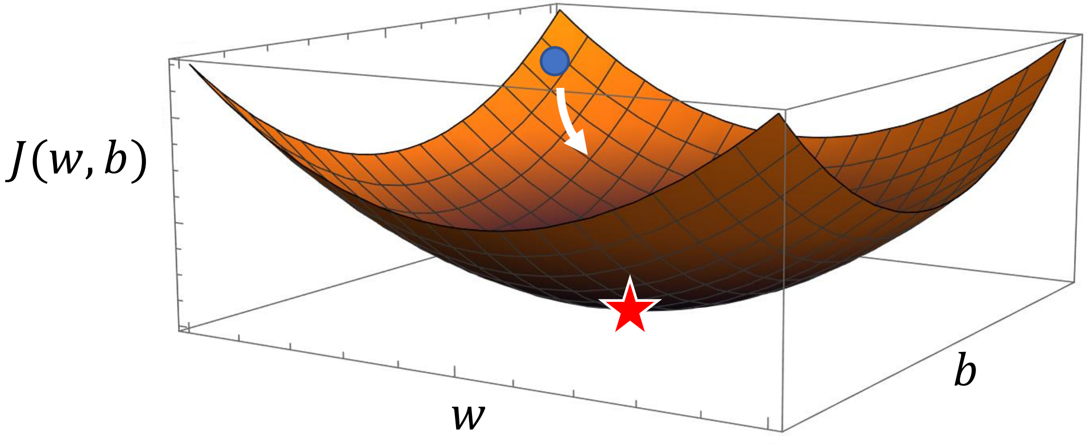
- $-\nabla J$ gives the direction of the steepest decrease of J

$$-\nabla J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$$

But how are $\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}$ calculated?

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$



$$L \equiv (y(x) - y_d)^2, \quad J = \frac{1}{m}\sum_{i=1}^{m} L$$

Back propagation (chain rule)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial (wx+b)}\frac{\partial (wx+b)}{\partial w} = 2(y - y_d)\sigma' x$$

1) For one example:

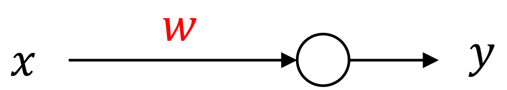
$$\frac{\partial L}{\partial w}(\mathrm{w}, \mathrm{b}, \mathrm{x}_\mathrm{d}^i, \mathrm{y}_\mathrm{d}^i) = 2\left(\sigma\left(w\mathrm{x}_\mathrm{d}^i + b\right) - \mathrm{y}_\mathrm{d}^i\right)\sigma' \, \mathrm{x}_\mathrm{d}^i$$

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} 2\left(\sigma\left(w\mathrm{x}_\mathrm{d}^i + b\right) - \mathrm{y}_\mathrm{d}^i\right)\sigma'\mathrm{x}_\mathrm{d}^i$$

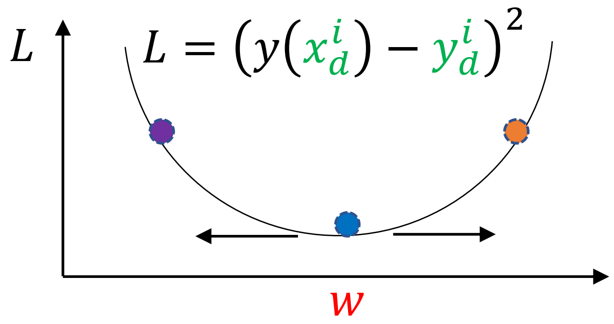
https://www.deeplearning.ai/ai-notes/optimization/

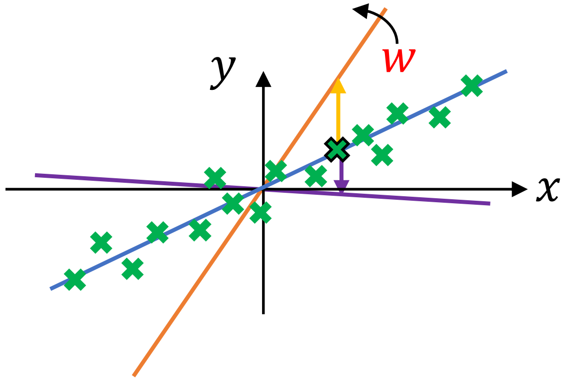
Example:



- Linear Regression Model $y = wx$
- Cost function $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L\left(y^i, y_d^i\right)$

1) For one example:



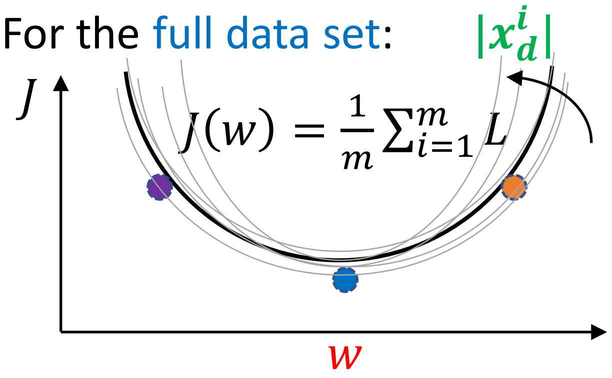


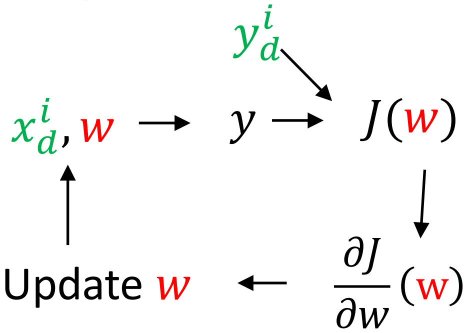
$$L \equiv (y(x) - y_d)^2, \; J = \frac{1}{m}\sum_{i=1}^{m} L \;\rightarrow\; \frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w}$$

Back propagation (chain rule)

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial (wx)}\frac{\partial (wx)}{\partial w} = 2(y - y_d)x$$

2) For the full data set:





1) For one example:

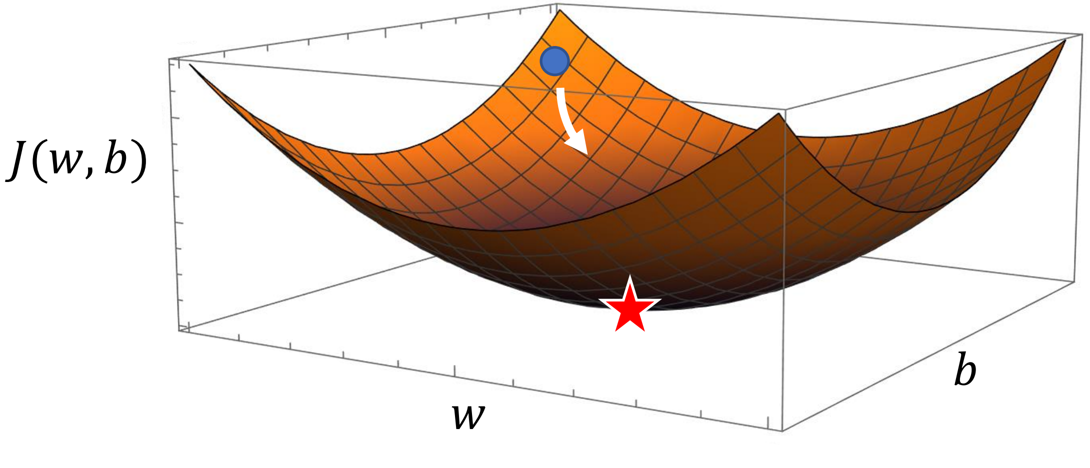
$$\frac{\partial L}{\partial w}\left(\mathrm{w}, \mathrm{b}, \mathrm{x}_d^i, \mathrm{y}_d^i\right) = 2\left(y\left(x_d^i\right) - \mathrm{y}_d^i\right)\mathrm{x}_d^i$$

2) For the full data set:

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w} = \frac{1}{m}\sum_{i=1}^{m} 2\left(y\left(x_d^i\right) - \mathrm{y}_d^i\right)\mathrm{x}_d^i$$

How to find w and b? Gradient descent

- E.g., Model $y = \sigma(wx + b)$
- Cost function $J(w, b) = \frac{1}{m}\sum_{i=1}^{m}\left(y\left(x_d^i\right) - y_d^i\right)^2 = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$



- $-\boldsymbol{\nabla} J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ **for a given data set at a given $\boldsymbol{w}, \boldsymbol{b}$ is known analytically**
- $(w_{new}, b_{new}) = (w_{old}, b_{old}) - \alpha\, \nabla J(w, b)$,
  α is the learning rate
- Iterate until $\nabla J = 0$

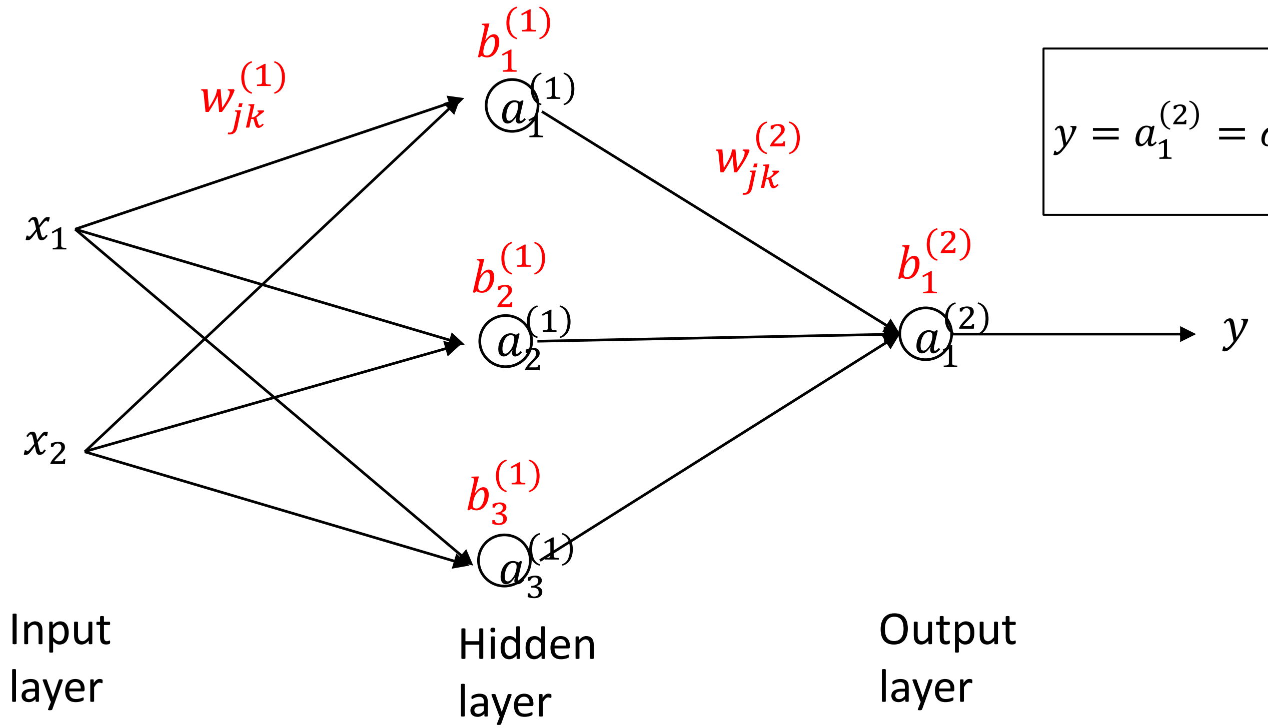
So far we have discussed…

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J
- Calculate $-\boldsymbol{\nabla} J(w, b) = -\left(\frac{\partial J}{\partial w}, \frac{\partial J}{\partial b}\right)$ for a simple model $y = \sigma(wx + b)$
  - One example
  - A full dataset

We know how to find w and b for a simple model $y = \sigma(wx + b)$.

What about finding w and b in a neural network?

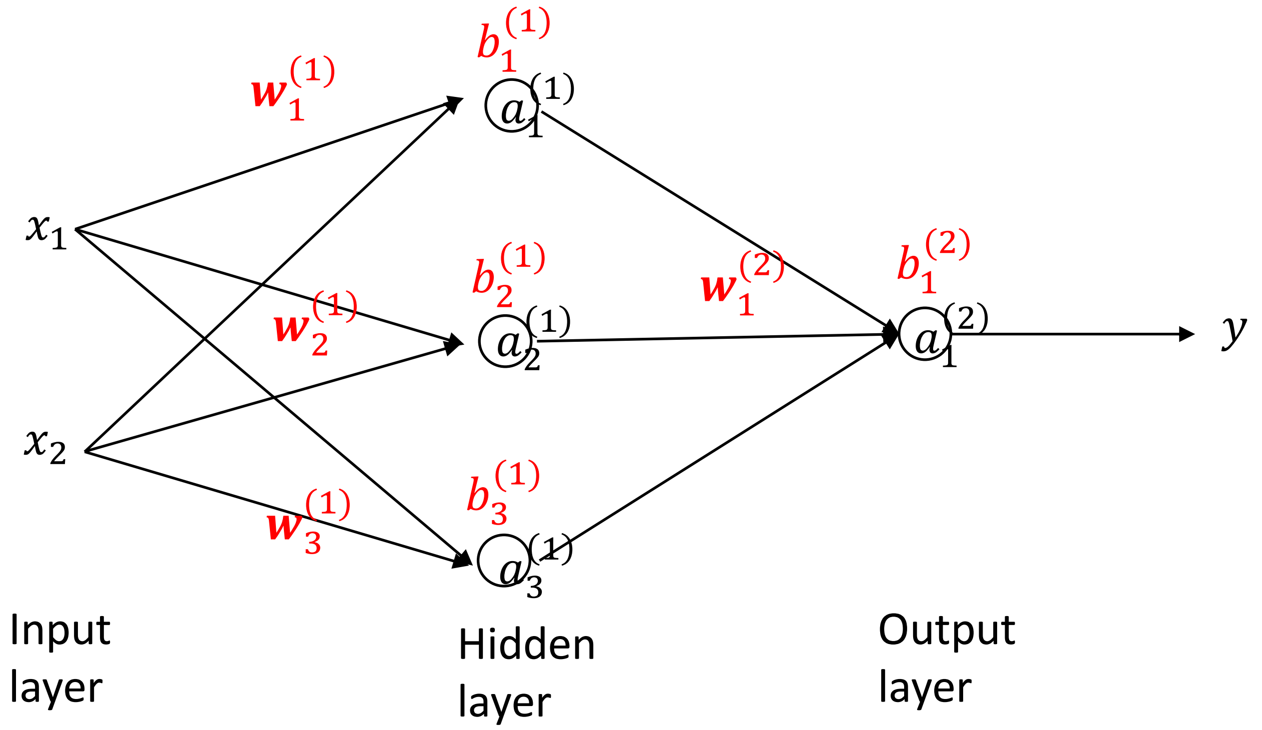
Writing a neural network in terms of vectors



$$\boxed{y = a_1^{(2)} = \sigma\left(\sum_{k=1}^{3} w_{1k}^{(2)} \sigma\left(\sum_{l=1}^{2} w_{kl}^{(1)} x_l + b_k^{(1)}\right) + b_1^{(2)}\right)}$$

# of units	j=1,2	j=1,2,3	j=1
Forward propagation	$a_j^{(0)} = x_j$	$z_j^{(1)} = \sum_{k=1}^{2} w_{jk}^{(1)} a_k^{(0)} + b_j^{(1)}$ $a_j^{(1)} = \sigma\left(z_j^{(1)}\right)$	$z_j^{(2)} = \sum_{k=1}^{3} w_{jk}^{(2)} a_k^{(1)} + b_j^{(2)}$ $a_j^{(2)} = \sigma\left(z_j^{(2)}\right) = z_j^{(2)}$

Writing a neural network in terms of vectors

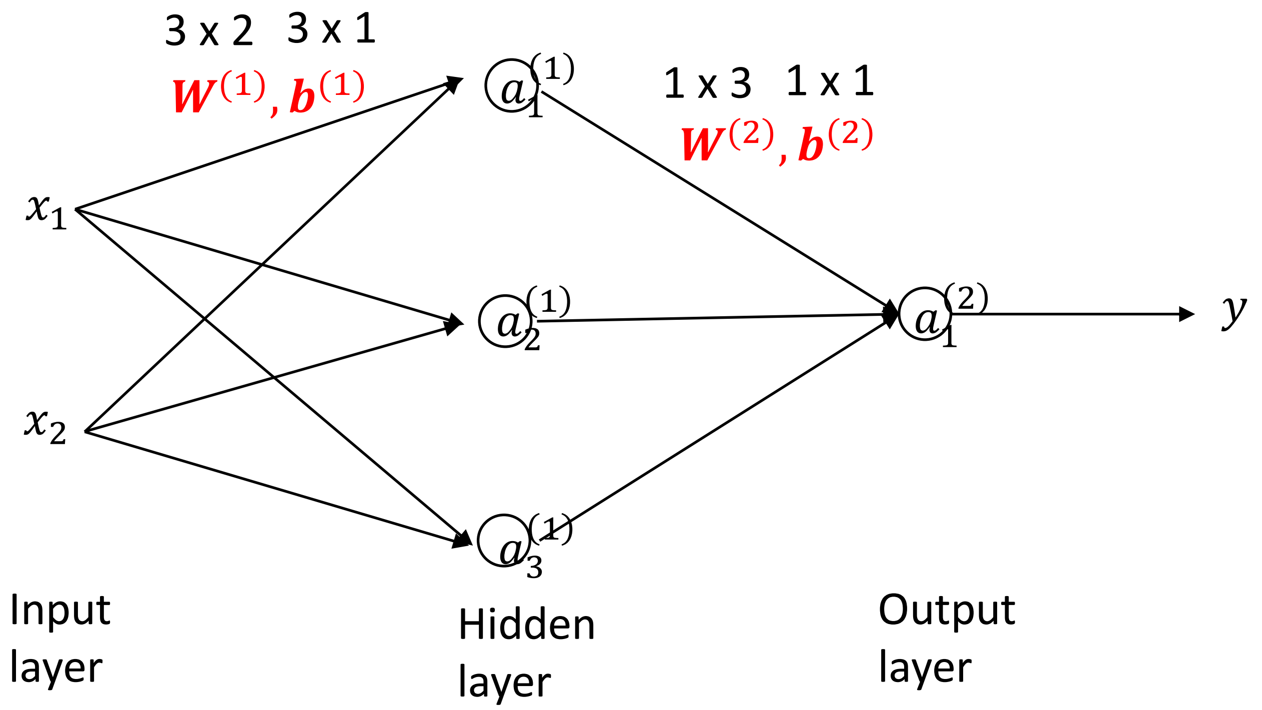


$$\boldsymbol{a}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{w}_j^{(1)} = \begin{bmatrix} w_{j1}^{(1)} \\ w_{j2}^{(1)} \end{bmatrix}$$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix}, \boldsymbol{w}_j^{(2)} = \begin{bmatrix} w_{j1}^{(2)} \\ w_{j2}^{(2)} \\ w_{j3}^{(2)} \end{bmatrix}$$

# of units	*j=1,2*	*j=1,2,3*	*j=1*
Forward propagation	$\boldsymbol{a}^{(0)} = \boldsymbol{x}$	$z_j^{(1)} = \mathbf{w}_j^{(1)T} \boldsymbol{a}^{(0)} + b_j^{(1)}$ $a_j^{(1)} = \sigma(z_j^{(1)})$	$z_j^{(2)} = \mathbf{w}_j^{(2)T} \boldsymbol{a}^{(1)} + b_j^{(2)}$ $a_j^{(2)} = \sigma(z_j^{(2)})$

Writing a neural network in terms of vectors



$$\boldsymbol{a}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix}$$

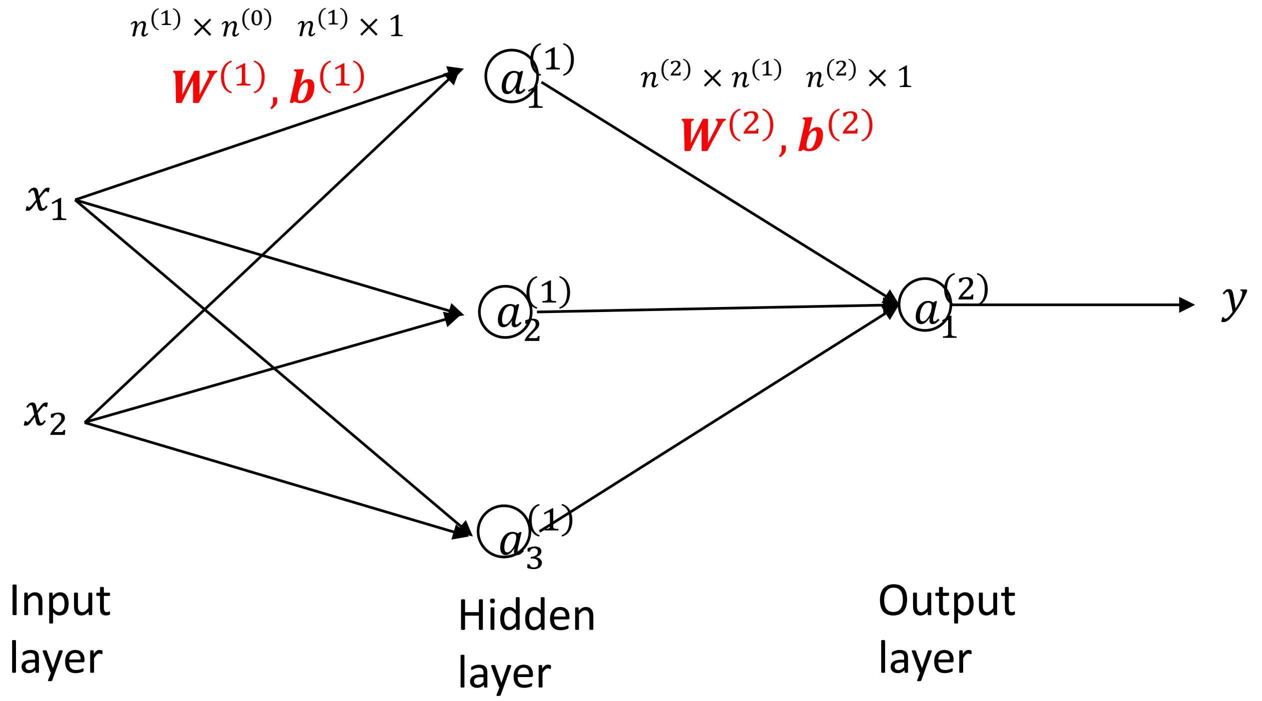
$$\boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}, \boldsymbol{z}^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{bmatrix}$$

# of units	*j=1,2*	*j=1,2,3*	*j=1*
Forward propagation	$\boldsymbol{a}^{(0)} = \boldsymbol{x}$	$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}$ $\boldsymbol{a}^{(1)} = \sigma(\boldsymbol{z}^{(1)})$	$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}$ $\boldsymbol{a}^{(2)} = \sigma(\boldsymbol{z}^{(2)})$

Writing a neural network in terms of vectors



$$\boldsymbol{a}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix}$$

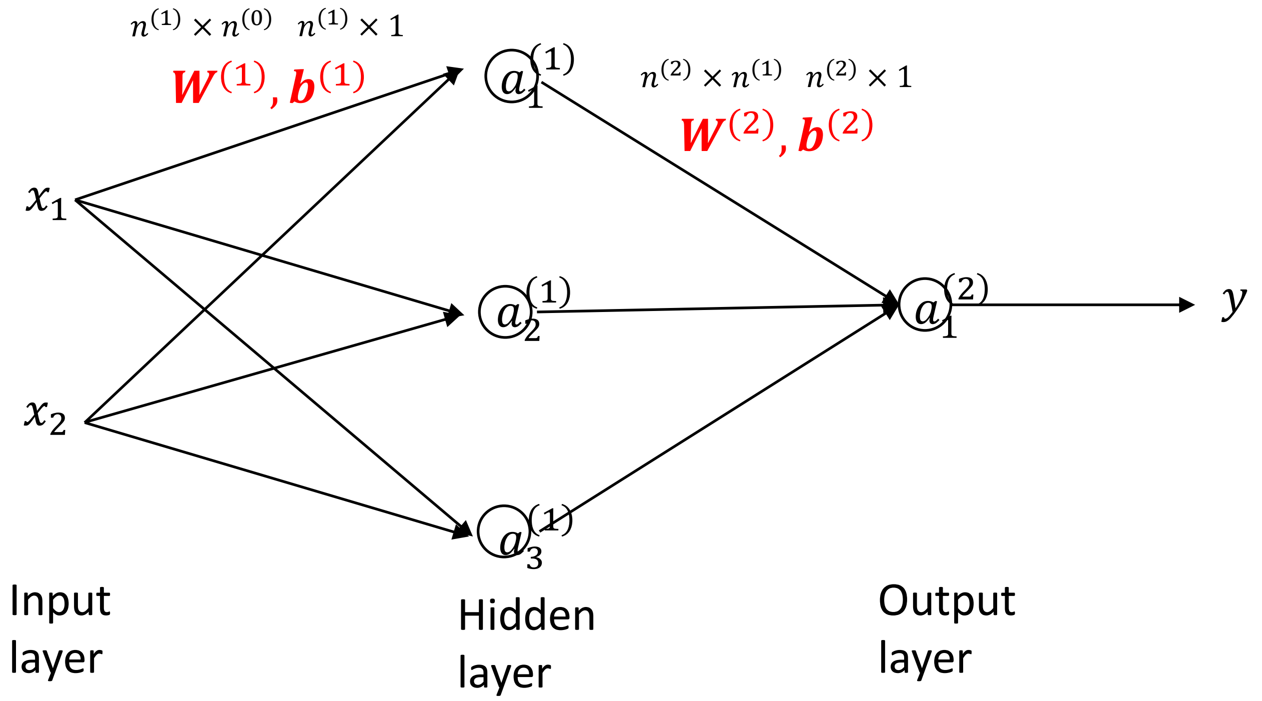
$$\boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}, \boldsymbol{z}^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{bmatrix}$$

# of units	$n^{(0)} = 2$	$n^{(1)} = 3$	$n^{(2)} = 1$
Forward propagation	$\boldsymbol{a}^{(0)} = \boldsymbol{x}$	$\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}$ $\boldsymbol{a}^{(1)} = \sigma(\boldsymbol{z}^{(1)})$	$\boldsymbol{z}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{a}^{(1)} + \boldsymbol{b}^{(2)}$ $\boldsymbol{a}^{(2)} = \sigma(\boldsymbol{z}^{(2)})$

Writing a neural network in terms of vectors



$$\boldsymbol{a}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \boldsymbol{W}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix}$$

$$\boldsymbol{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}, \boldsymbol{z}^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix},$$

$$\boldsymbol{W}^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \end{bmatrix}$$

In short, $y = a_1^{(2)} = \sigma(\sum_{k=1}^{3} w_{1k}^{(2)} \sigma\left(\sum_{l=1}^{2} w_{kl}^{(1)} x_l + b_k^{(1)}\right) + b_1^{(2)})$

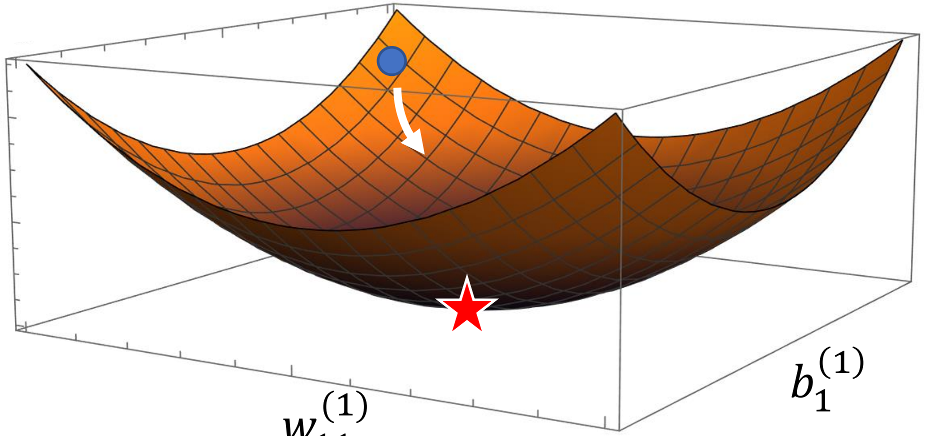
$\longrightarrow$ $y = \boldsymbol{a}^{(2)} = \sigma(\boldsymbol{W}^{(2)} \sigma(\boldsymbol{W}^{(1)} \boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)})$

We know how to find w and b for a simple model $y = \sigma(wx + b)$.

What about finding w and b in a neural network?
$y = a^{(2)} = \sigma(\boldsymbol{W}^{(2)}\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)})$

How to find w and b in a neural network?

- E. g., Model $y = a^{(2)} = \sigma(\boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) + \boldsymbol{b}^{(2)})$
- Cost function $J(\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(1)}, \boldsymbol{b}^{(2)}) = \frac{1}{m}\sum_{i=1}^{m} L(y^i, y_d^i)$



- For the full data set (i=1…m), compute

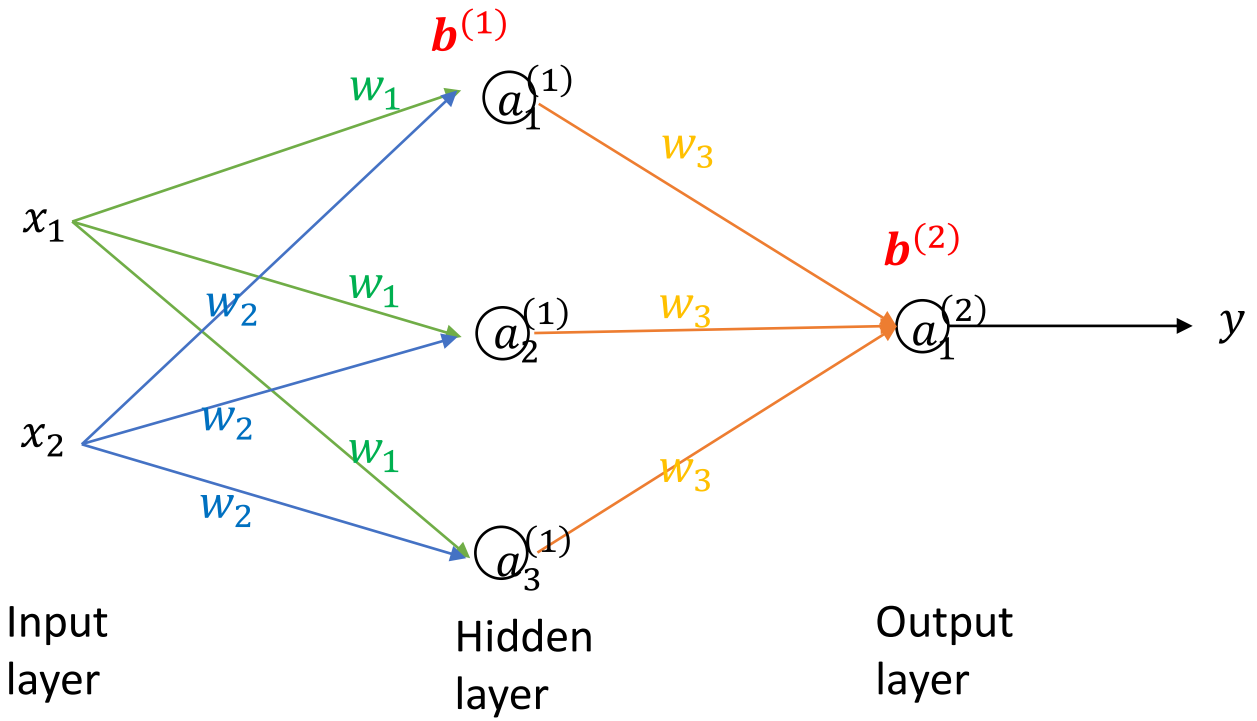
$$-\boldsymbol{\nabla} J = -\left(\frac{\partial J}{\partial w_{jk}^{(l)}}, \dots, \frac{\partial J}{\partial b_j^{(l)}}\right)$$

- ${w_{jk}}_{new}^{(l)} = {w_{jk}}_{old}^{(l)} - \alpha \frac{\partial J}{\partial w_{jk}^{(l)}}$, ${b_j}_{new}^{(l)} = {b_j}_{old}^{(l)} - \alpha \frac{\partial J}{\partial b_j^{(l)}}$

  α is the learning rate

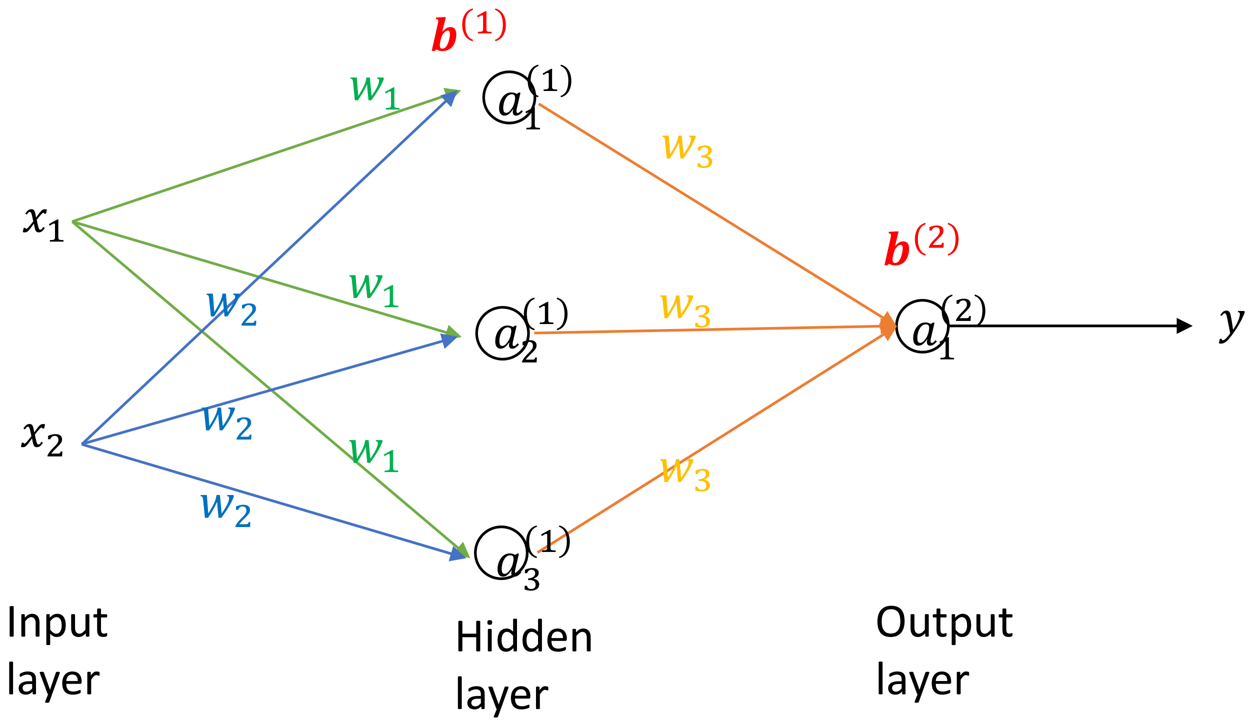
- Iterate until $\boldsymbol{\nabla} J = 0$

NN weight Initialization



Q: What would happen if all initial weights are chosen to be the same (e.g., all zeros)?

NN weight Initialization



More specifically, if weights are symmetric:

$$\boldsymbol{W}^{(1)} = \begin{bmatrix} w_1 & w_2 \\ w_1 & w_2 \\ w_1 & w_2 \end{bmatrix}, \boldsymbol{W}^{(2)} = [w_3 \quad w_3 \quad w_3]$$

(e.g. all zero weights, all constant weights)

Then $a_1^{(1)} = a_2^{(1)} = a_3^{(1)}$

and

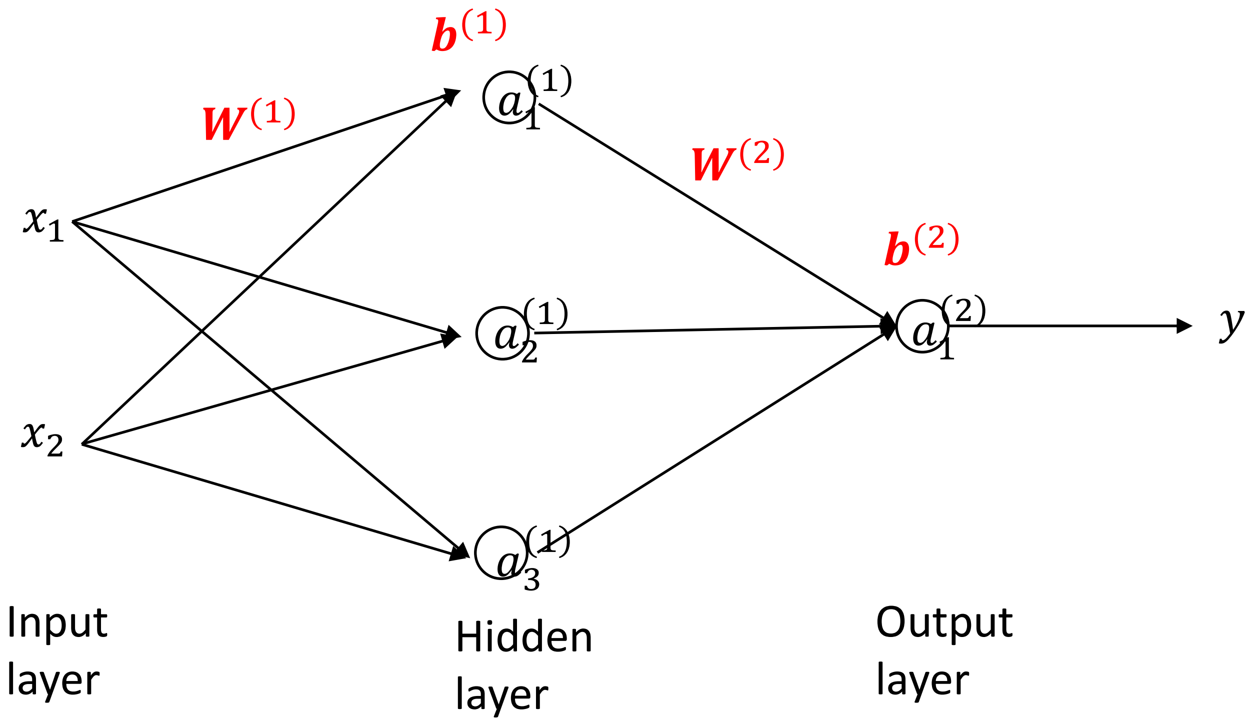
$$\begin{bmatrix} w_{1new} & w_{2new} \\ w_{1new} & w_{2new} \\ w_{1new} & w_{2new} \end{bmatrix} = \begin{bmatrix} w_1 & w_2 \\ w_1 & w_2 \\ w_1 & w_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial J}{\partial w_1} & \frac{\partial J}{\partial w_2} \\ \frac{\partial J}{\partial w_1} & \frac{\partial J}{\partial w_2} \\ \frac{\partial J}{\partial w_1} & \frac{\partial J}{\partial w_2} \end{bmatrix}$$

The NN never learns to have non-symmetric weights -> $a_1^{(1)} = a_2^{(1)} = a_3^{(1)}$ during training

If the NN weights are **symmetric** → Gradient descent will update these hidden units in the same way → All hidden units in the same layer will be identical throughout training iterations → Equivalent to NN with just 1 hidden unit.

https://www.deeplearning.ai/ai-notes/initialization/

NN weight Initialization



To make the different hidden units approximate different functions

→ Initialized weights $w_{jk}^{(l)}$ to random values

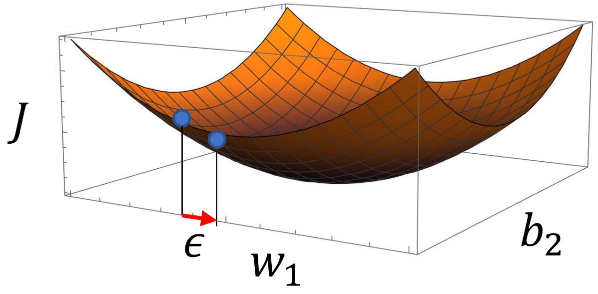
What about biases?

→ Biases can just be zeros

$$\boldsymbol{b}^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{b}^{(2)} = [0]$$

If the NN weights are **symmetric** → Gradient descent will update these hidden units in the same way → All hidden units in the same layer will be identical throughout training iterations → Equivalent to NN with just 1 hidden unit.

How are $\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{w_1}}, \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{b_1}}$ … calculated?



i. Slow: Calculate J at two adjacent locations, evaluate their differences

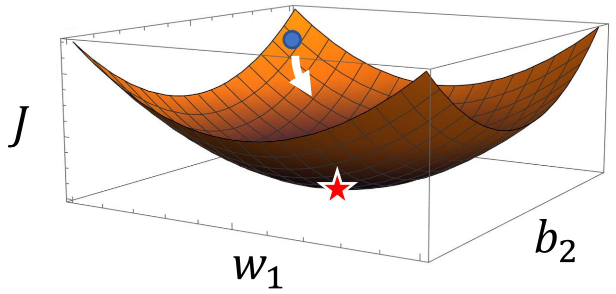
$\frac{\partial J}{\partial w_j} = \frac{J(w_0+\epsilon \boldsymbol{e_j})-J(w_0)}{\epsilon}$ For 1M weights, have to do 1M forward passes

ii. Fast: Back propagation

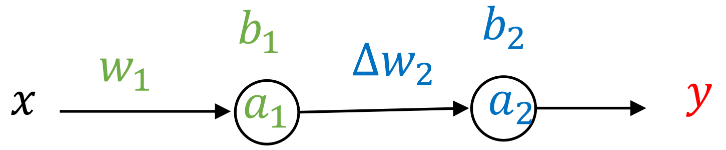
Compute derivatives w.r.t. all weights analytically with chain rules

Forward & Back Propagation: 1. single neuron

Exercise:
Model: $y = \sigma(w_2\sigma(w_1 x + b_1) + b_2)$



∇J vary with
w_1, w_2, b_1, b_2



How important is w_2 for changing the cost function J?

Forward propagation ⟶

$z_1 = w_1 x + b_1$
$a_1 = \sigma(z_1)$
→
$z_2 = w_2 a_1 + b_2$
$a_2 = \sigma(z_2)$
⇄
$y = a_2$
⇄
Cost function:
$J = (y - y_d)^2$

$\frac{\partial a_2}{\partial w_2}$ $\frac{\partial J}{\partial a_2}$

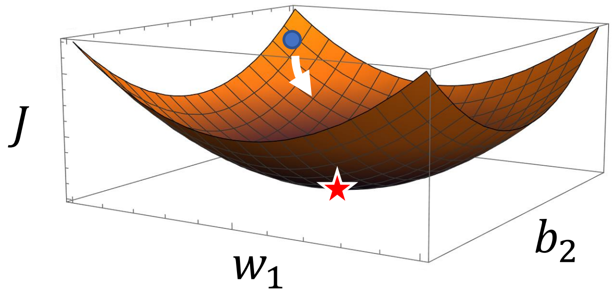
⟵ *Back propagation*

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$= 2(y - y_d)\sigma'(z_2)a_1$$

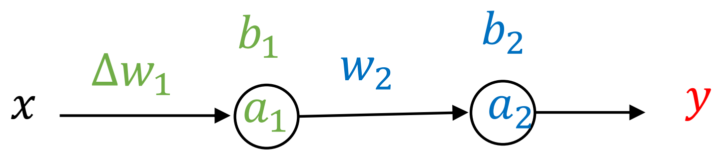
Forward & Back Propagation: 1. single neuron

Exercise:

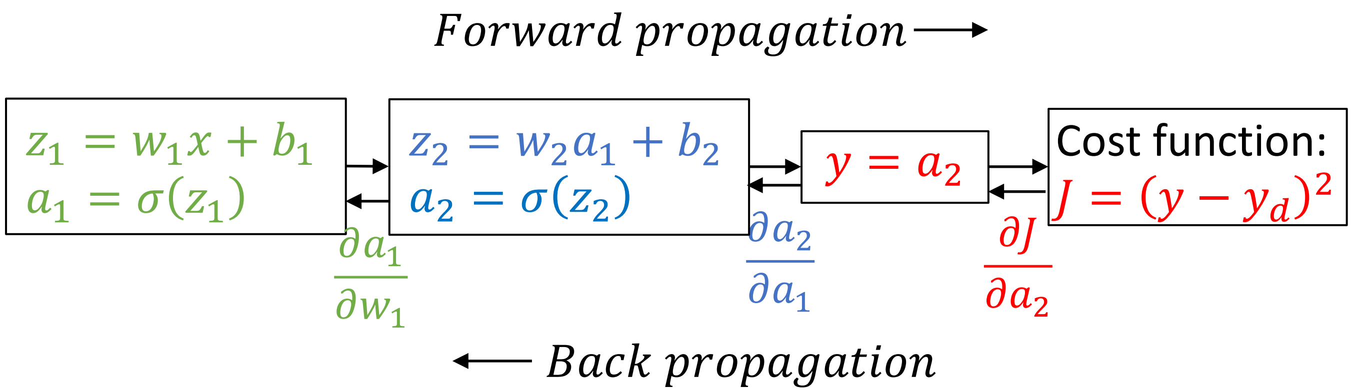
Model: $y = \sigma(w_2\sigma(w_1 x + b_1) + b_2)$



∇J vary with w_1, w_2, b_1, b_2

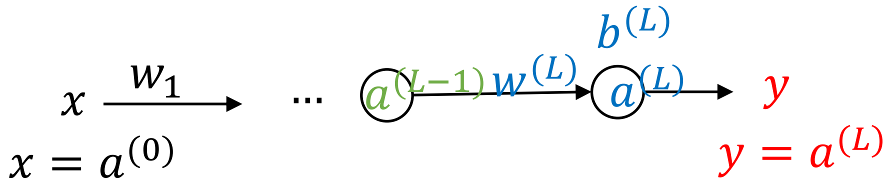


How important is w_1 for changing the cost function J?



$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial a_2}\frac{\partial a_2}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$= 2(y - y_d)\sigma'(z_2)w_2\sigma'(z_1)x$$

Forward & Back Propagation: 1. single neuron



Forward propagation

Input: $a^{(0)} = x$

...

$z^{(L-1)} = w^{(L-1)} a^{(L-2)} + b^{(L-1)}$

$a^{(L-1)} = \sigma(z^{(L-1)})$

$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$

Output: $y = a^{(L)}$

Cost function $J = (y - y_d)^2$

Back propagation (use chain rules)

$$\frac{\partial J}{\partial w^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial w^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})})\, a^{(L-1)}$$

$$\frac{\partial J}{\partial b^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial b^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})})$$

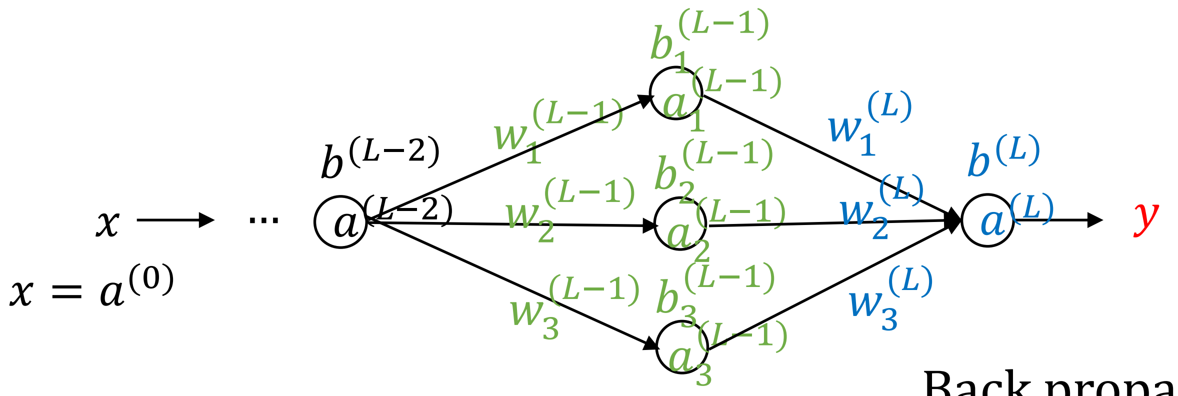
$$\frac{\partial J}{\partial w^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial w^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})}) w^{(L)} \sigma'(z^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial b^{(L-1)}}$$

$$= 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})}) w^{(L)} \sigma'(z^{(L-1)})$$

Forward & Back Propagation: 2. multiple hidden units



Forward propagation

Input: $a^{(0)} = x$

…

$z_k^{(L-1)} = w_k^{(L-1)} a^{(L-2)} + b_k^{(L-1)}$

$a_k^{(L-1)} = \sigma(z_k^{(L-1)})$

$z^{(L)} = \sum_{k=1}^{3} w_k^{(L)} a_k^{(L-1)} + b^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$

Output: $y = a^{(L)}$

Cost function $J = (y - y_d)^2$

Back propagation (use chain rules)

$$\frac{\partial J}{\partial w_k^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial w_k^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})})\, a_k^{(L-1)}$$

$$\frac{\partial J}{\partial b^{(L)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial b^{(L)}} = 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})})$$

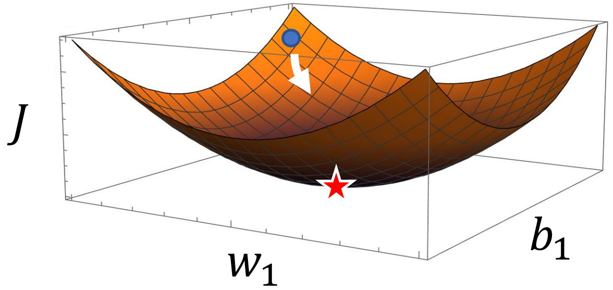
$$\frac{\partial J}{\partial w_k^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}}$$
$$= 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})}) w_k^{(L)} \sigma'(z_k^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b_k^{(L-1)}} = \frac{\partial J}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}}$$
$$= 2(a^{(L)} - y_d)\, \sigma'(\mathrm{z}^{(\mathrm{L})}) w_k^{(L)} \sigma'(z_k^{(L-1)})$$

Forward & Back Propagation: 2. multiple hidden units

Exercise:

Model: $y = \sigma\left(\sum_{k=1}^{3} w_k^{(2)}\, \sigma\left(w_k^{(1)} x + b_k^{(1)}\right) + b^{(2)}\right)$



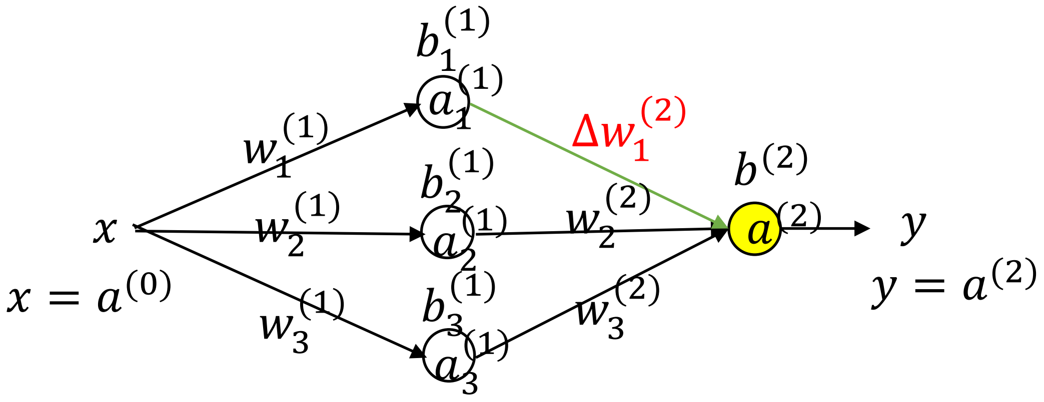
∇J vary with $w_k^{(1)}, w_k^{(2)}, b_k^{(1)}, \mathrm{b}^{(2)}$

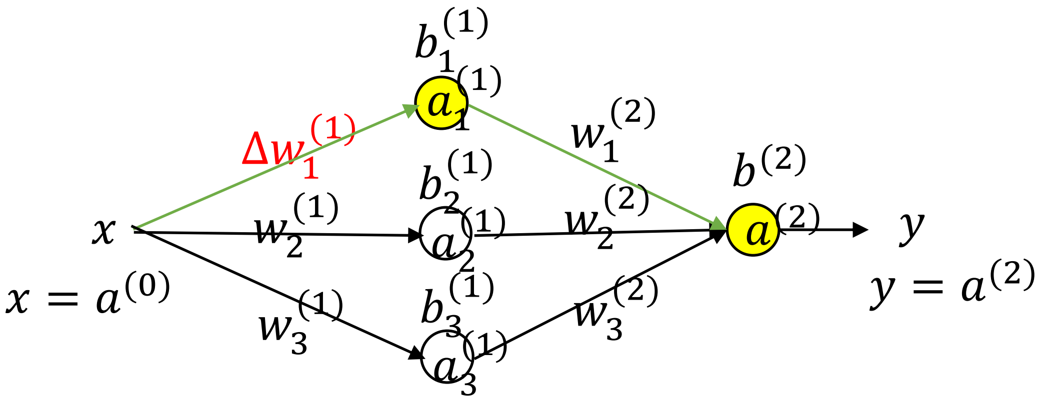
$$\frac{\partial J}{\partial w_1^{(2)}} = 2\left(a^{(2)} - y_d\right)\sigma'\left(z^{(2)}\right)a_1^{(1)}$$

$$= \mathrm{fn}\left(\mathrm{x}, \mathrm{y_d}, w_k^{(1)}, w_k^{(2)}, b_k^{(1)}, \mathrm{b}^{(2)}\right)$$

$$\frac{\partial J}{\partial w_1^{(1)}} = 2\left(a^{(2)} - y_d\right)\sigma'\left(z^{(2)}\right)w_1^{(2)}\sigma'\left(z_1^{(1)}\right)x$$

$$= \mathrm{fn}\left(\mathrm{x}, \mathrm{y_d}, w_k^{(1)}, w_k^{(2)}, b_k^{(1)}, \mathrm{b}^{(2)}\right)$$

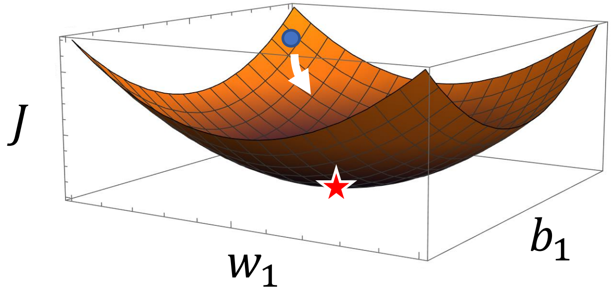




Forward & Back Propagation: 3. multiple outputs

Exercise:

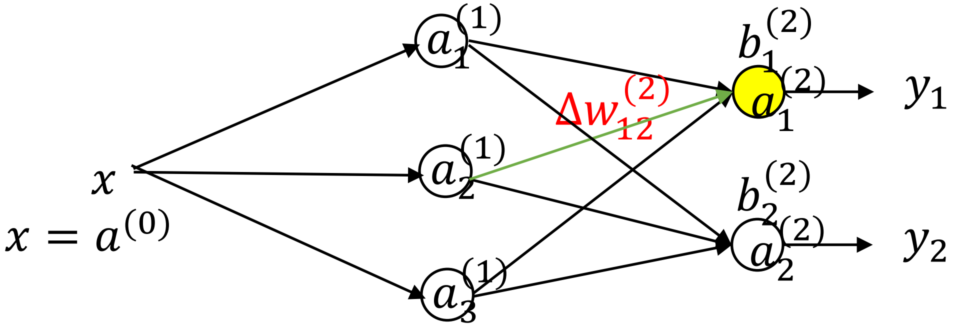
Model: $y_j = \sigma\left(\sum_{k=1}^{3} w_{jk}^{(2)}\, \sigma\left(w_k^{(1)} x + b_k^{(1)}\right) + b_j^{(2)}\right)$

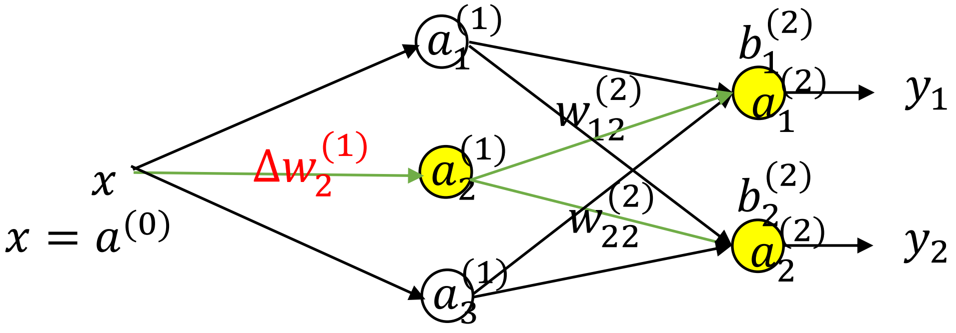


∇J vary with $w_k^{(1)}, w_{jk}^{(2)}, b_k^{(1)}, b_j^{(2)}$

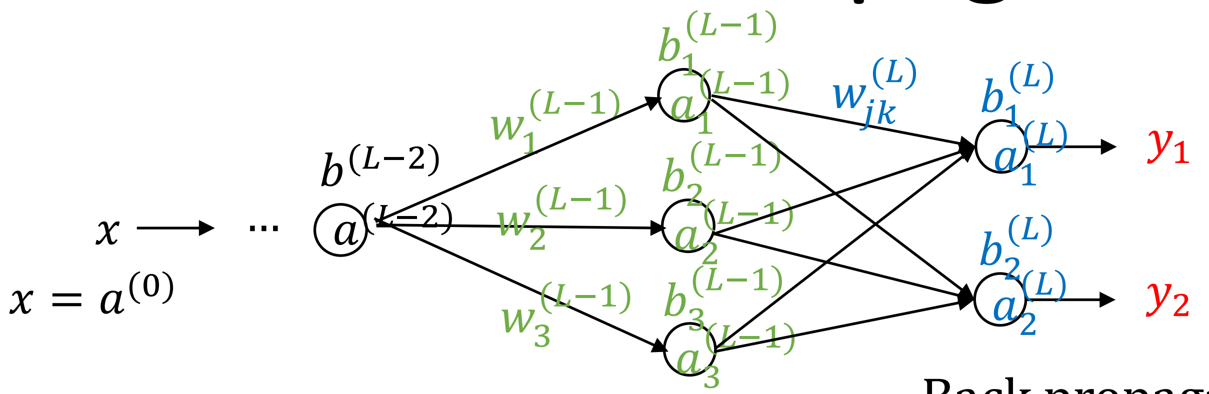
$$\frac{\partial J}{\partial w_{12}^{(2)}} = 2\left(a_1^{(2)} - y_{d,1}\right)\sigma'\left(z_1^{(2)}\right) a_2^{(1)}$$
$$= \mathrm{fn}(\mathrm{x}, \mathrm{y}_{\mathrm{d},1}, w_k^{(1)}, w_{1k}^{(2)}, b_k^{(1)}, \mathrm{b}_1^{(2)})$$

$$\frac{\partial J}{\partial w_2^{(1)}} = \sum_{j=1}^{2} 2(a_j^{(2)} - y_{d,j})\, \sigma'(z_j^{(2)})\mathrm{w}_{j2}^{(2)}\, \sigma'\left(z_2^{(1)}\right) x$$
$$= \mathrm{fn}(\mathrm{x}, \mathrm{y}_{\mathrm{d,j}}, w_k^{(1)}, w_{jk}^{(2)}, b_k^{(1)}, \mathrm{b}_j^{(2)})$$





Forward & Back Propagation: 3. multiple outputs



Back propagation (use chain rules)

$$\frac{\partial J}{\partial w_{jk}^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial w_{jk}^{(L)}} = 2(a_j^{(L)} - y_{d,j})\, \sigma'(\mathrm{z}_\mathrm{j}^{(\mathrm{L})}) a_k^{(L-1)}$$

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial b_j^{(L)}} = 2(a_j^{(L)} - y_{d,j})\, \sigma'(\mathrm{z}_j^{(\mathrm{L})})$$

$$\frac{\partial J}{\partial w_k^{(L-1)}} = \sum_{j=1}^{2} \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial w_k^{(L-1)}}$$
$$= \sum_{j=1}^{2} 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(\mathrm{L})}) \mathrm{w}_{jk}^{(\mathrm{L})}\ \sigma'(z_k^{(L-1)}) a^{(L-2)}$$

$$\frac{\partial J}{\partial b_k^{(L-1)}} = \sum_{j=1}^{2} \frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_k^{(L-1)}}{\partial b_k^{(L-1)}}$$
$$= \sum_{j=1}^{2} 2(a_j^{(L)} - y_{d,j})\, \sigma'(z_j^{(\mathrm{L})}) \mathrm{w}_{jk}^{(\mathrm{L})}\ \sigma'(z_k^{(L-1)})$$

Forward propagation

Input: $a^{(0)} = x$

…

$z_k^{(L-1)} = w_k^{(L-1)} a^{(L-2)} + b_k^{(L-1)}$

$a_k^{(L-1)} = \sigma(z_k^{(L-1)})$

$z_j^{(L)} = \sum_{k=1}^{3} w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)}$

$a_j^{(L)} = \sigma(z_j^{(L)})$

Output: $\mathrm{y_j} = a_j^{(L)}$

Cost function $J = \sum_{j=1}^{2} (y_j - y_{d,j})^2$

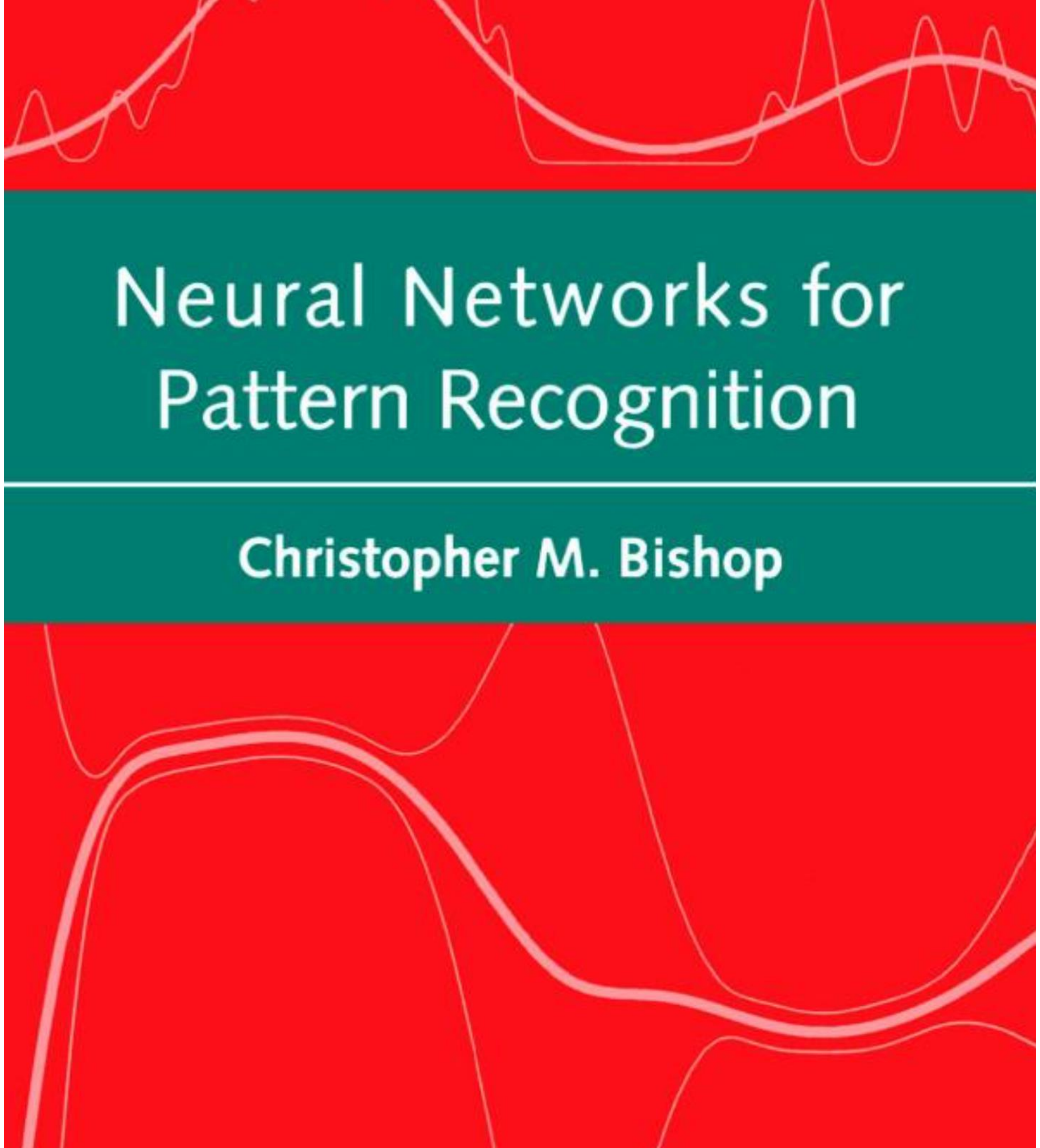
When training a neural network

Repeat these steps:

1. Forward propagate an input
2. Compute the cost function
3. Compute the gradients of the cost with respect to parameters using backpropagation
4. Update each parameter using the gradients, according to the optimization algorithm

Summary

- Universal function approximator
- Gradient descent: a method to find weights and biases that minimize J
- Calculate $\boldsymbol{\nabla} J$
  - One example
  - A full dataset
- Weight initialization
- Forward and backpropagation (a clever way to get gradients of J)



Chapter 4
The Multi-layer Perceptron