

Update: Course Staff

Instructor: Tejas Gokhale
Assistant Professor, CSEE



Wednesday 2:30 – 3:30 PM

ITE 342-B

gokhale@umbc.edu

TA: Aydin Ayanzadeh
Ph.D. Student (Research: Deep Learning, LLMs)



Thursday 9 – 11 PM

ITE 334

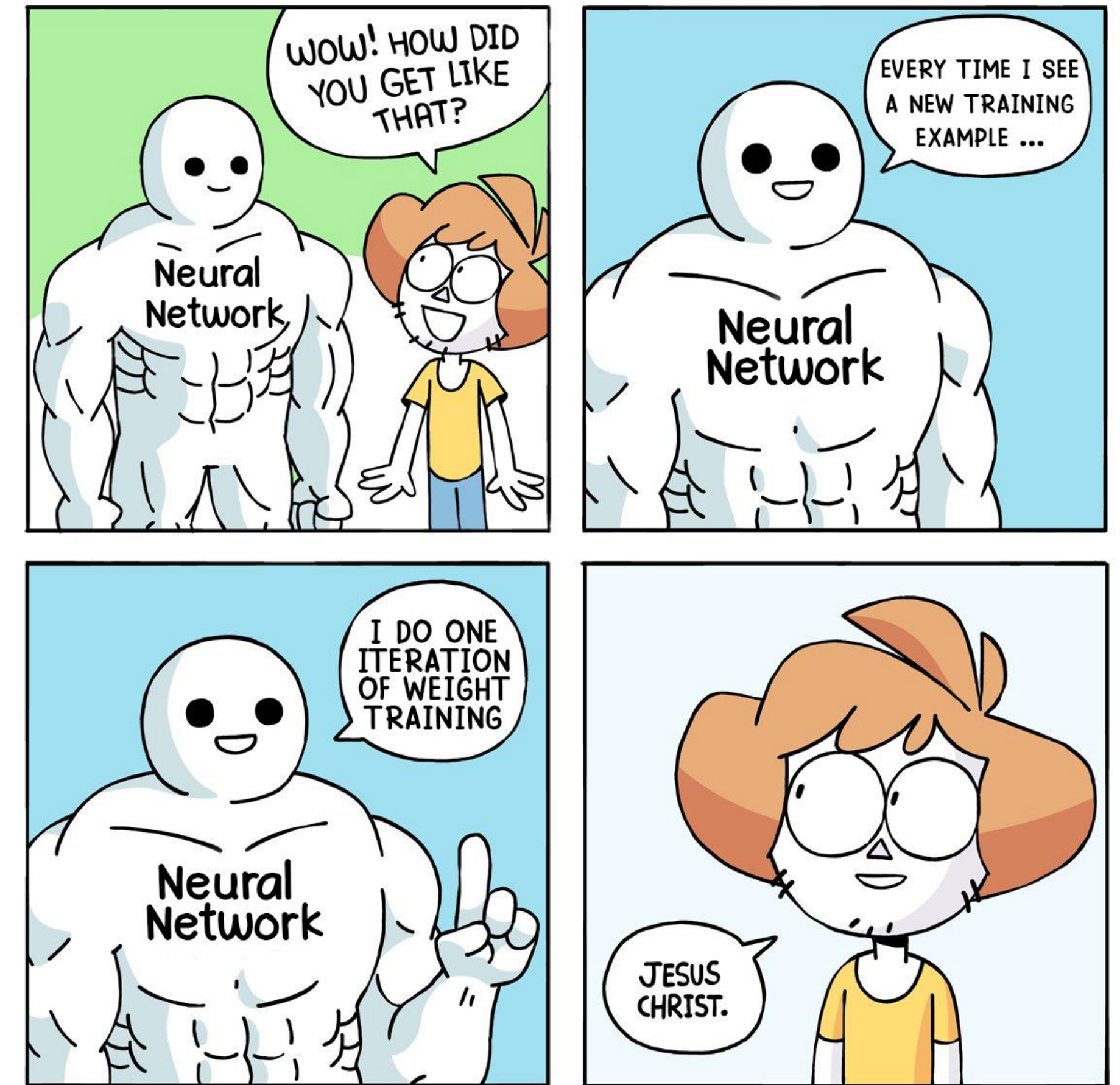
aydina1@umbc.edu

**Office
Hours**

CMSC 475/675 Neural Networks



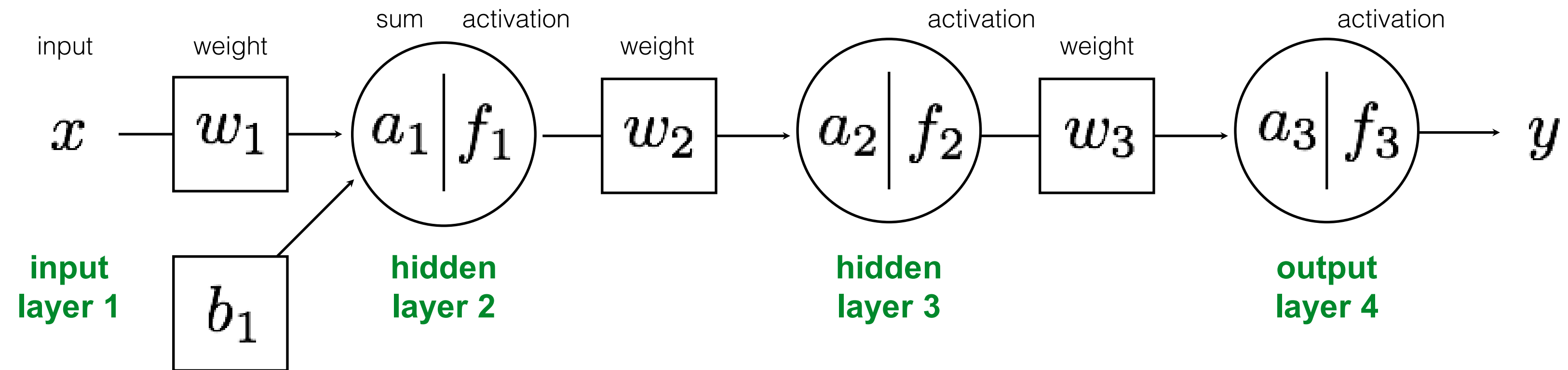
Lecture 3: Gradient Descent

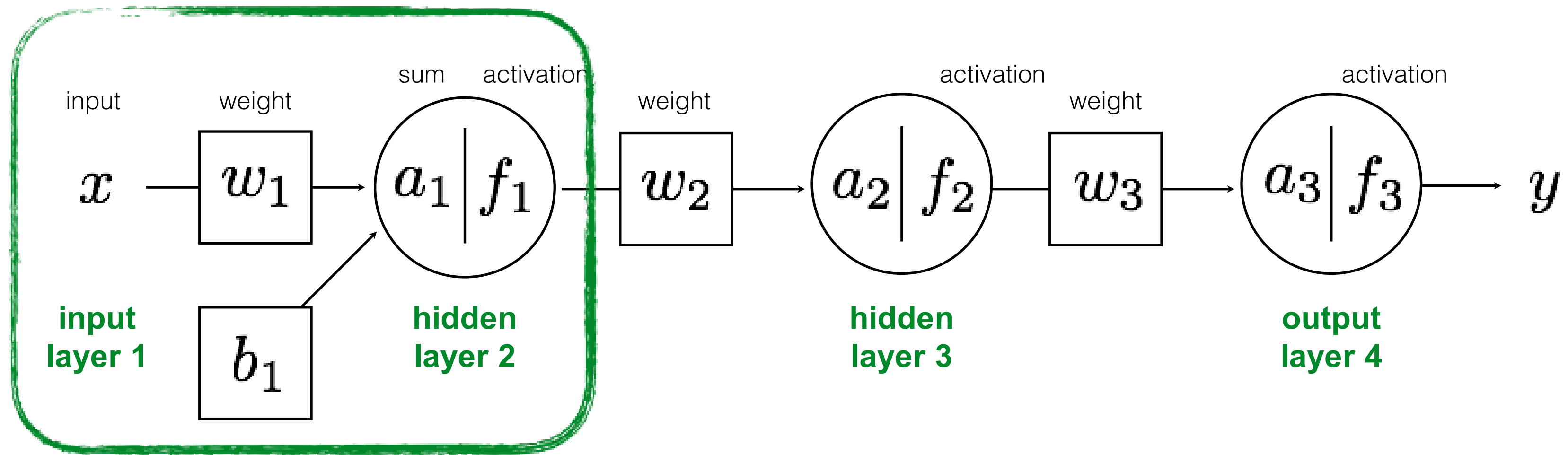


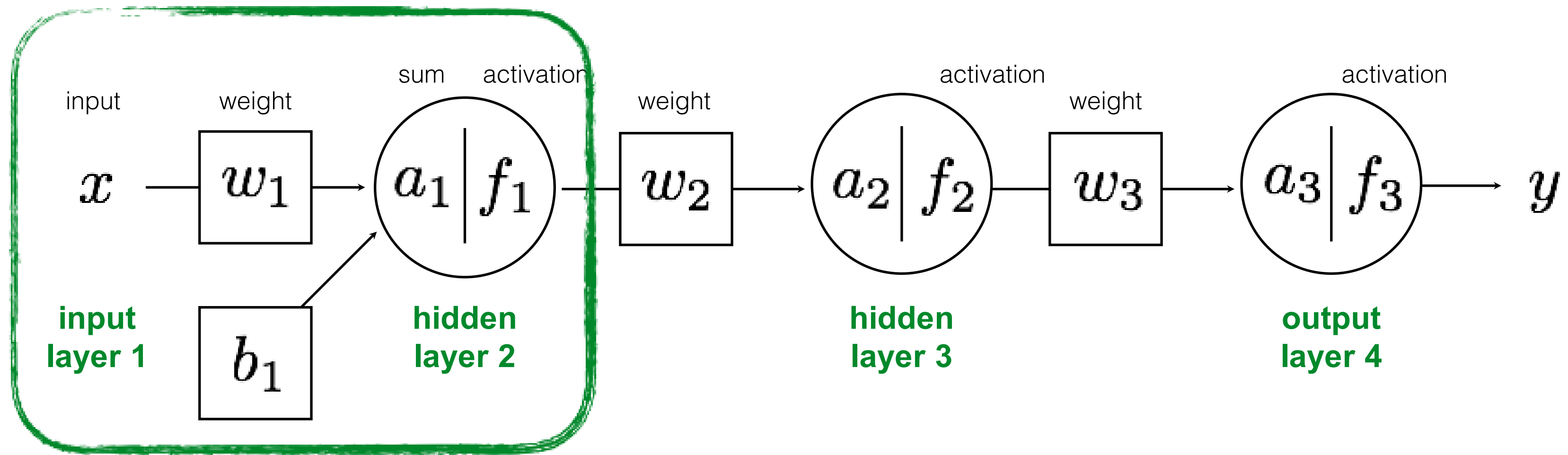
SHEN COMIX



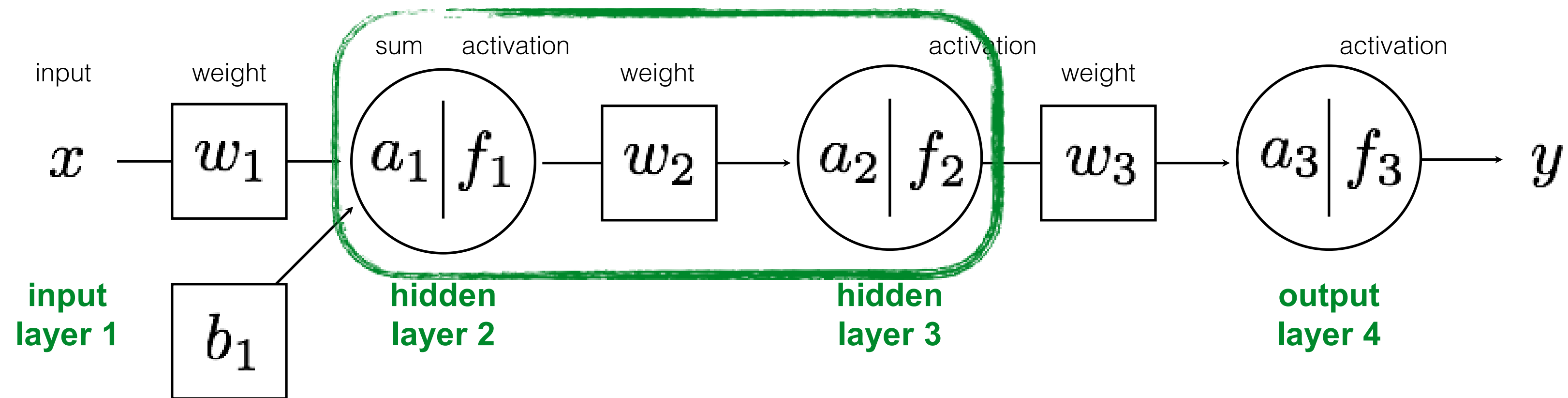
Some slides adapted from Suren Jayasuriya (ASU),
Matt Gormley (CMU)



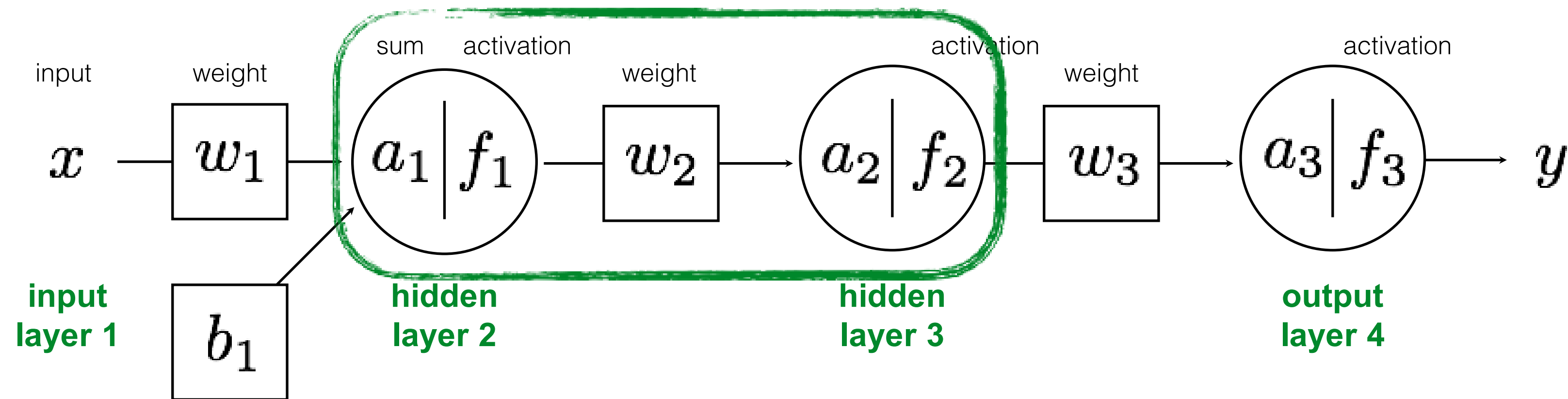




$$a_1 = w_1 \cdot x + b_1$$

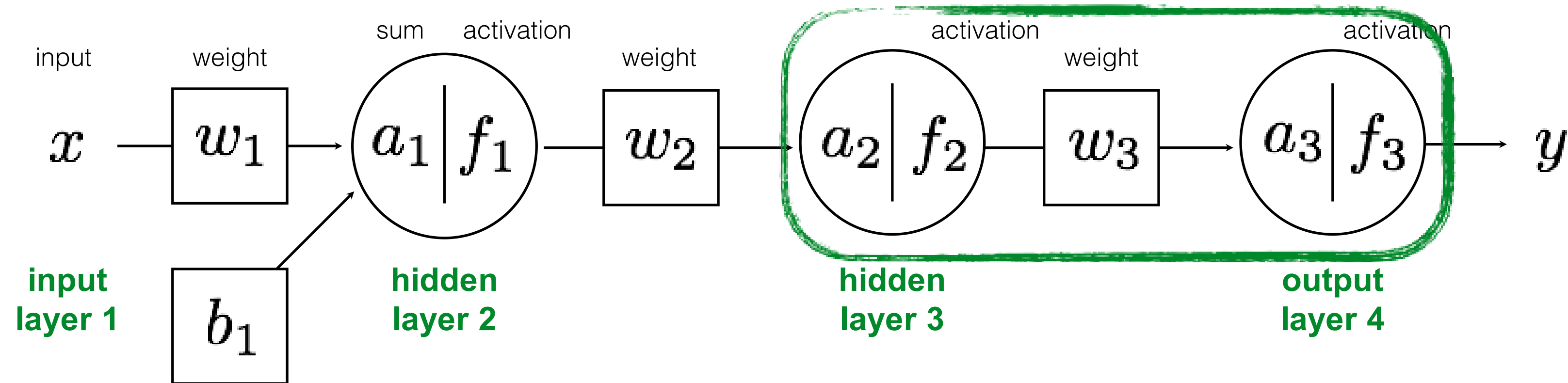


$$a_1 = w_1 \cdot x + b_1$$



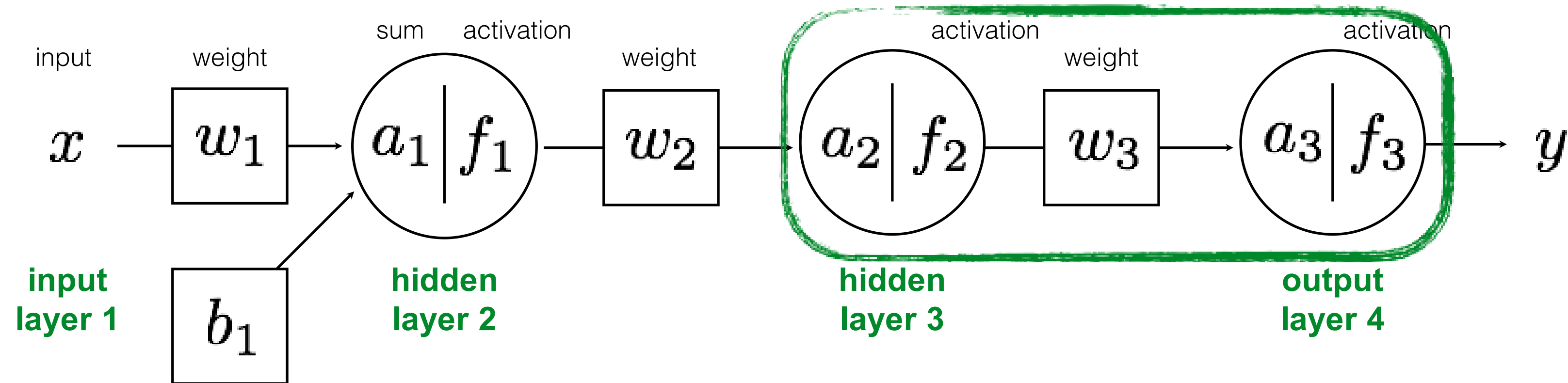
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

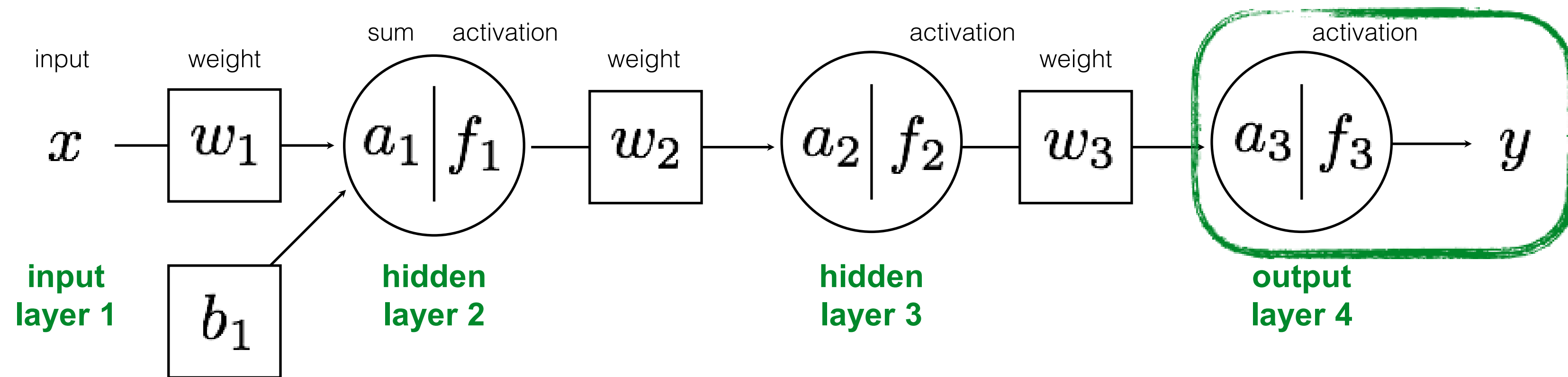
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

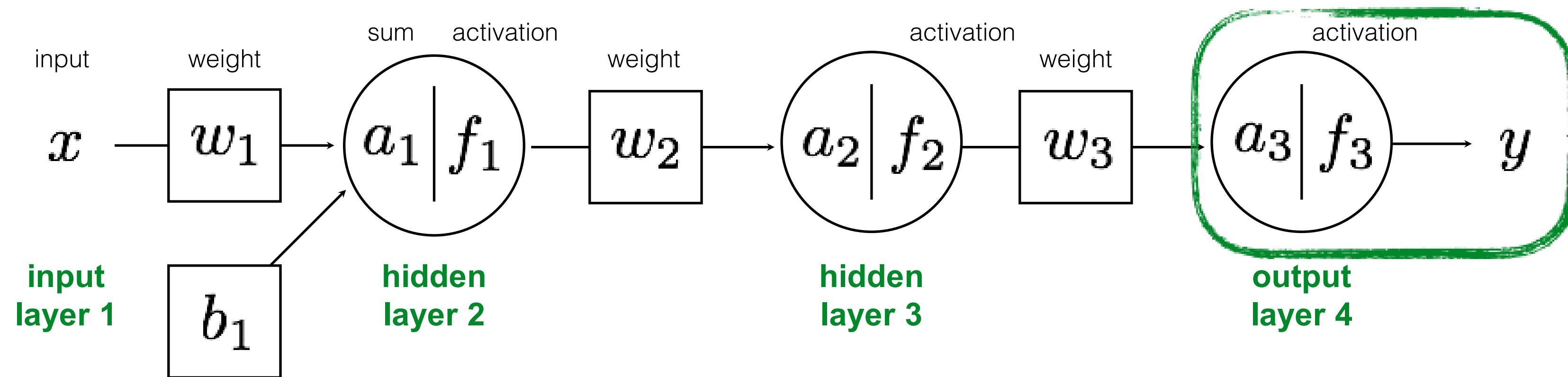
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$



known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function
sometimes has
parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict

a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

b. Compute Loss

2. Update

a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter partial derivatives

b. Gradient update

$$\theta \leftarrow \theta - \eta \nabla \theta$$

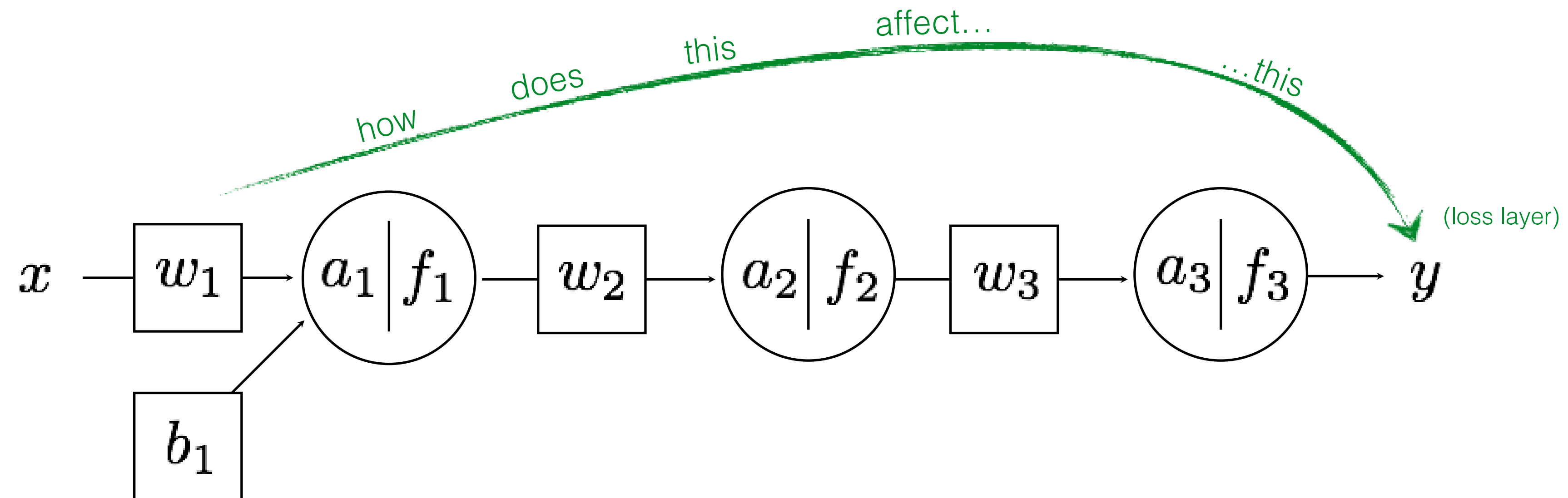
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

Remember,

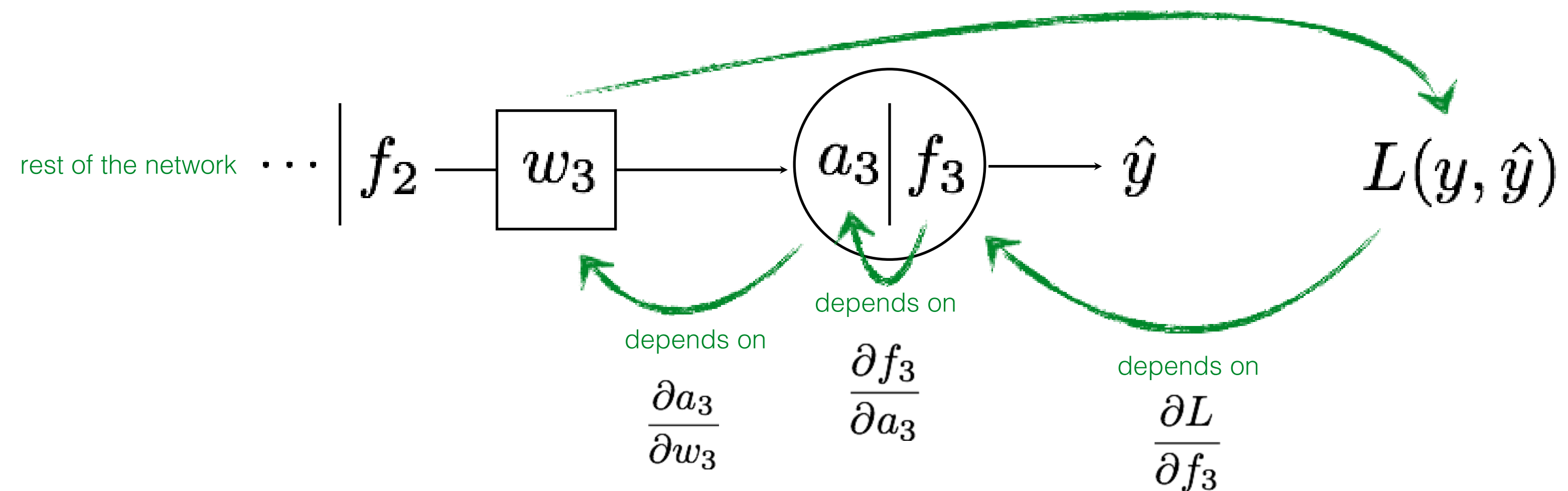
Partial derivative $\frac{\partial L}{\partial w_1}$ describes...

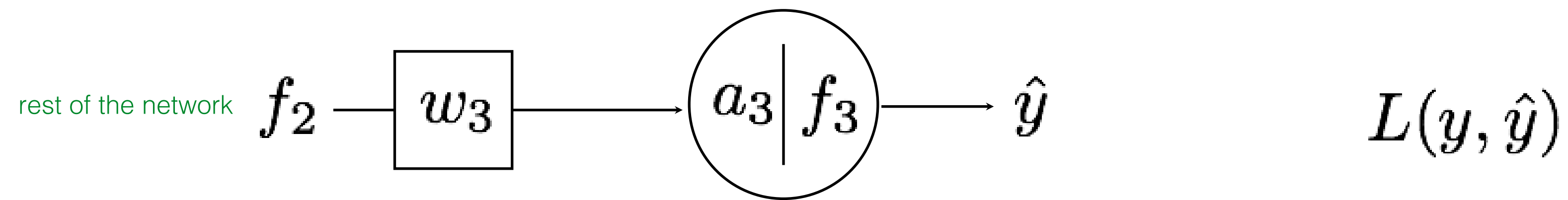


According to the chain rule...

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

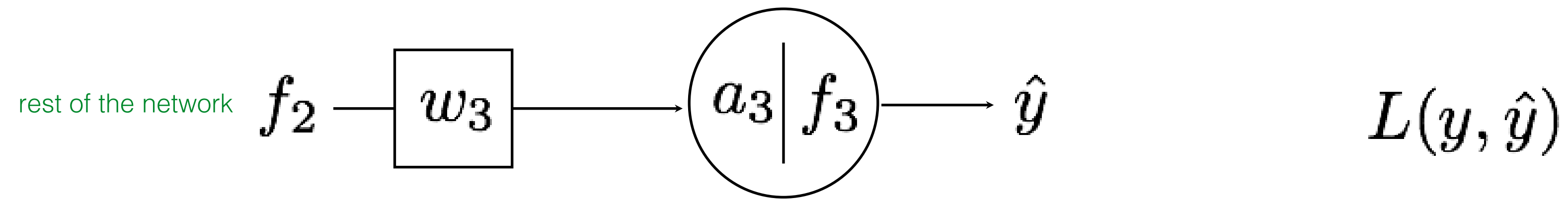
Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$





$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

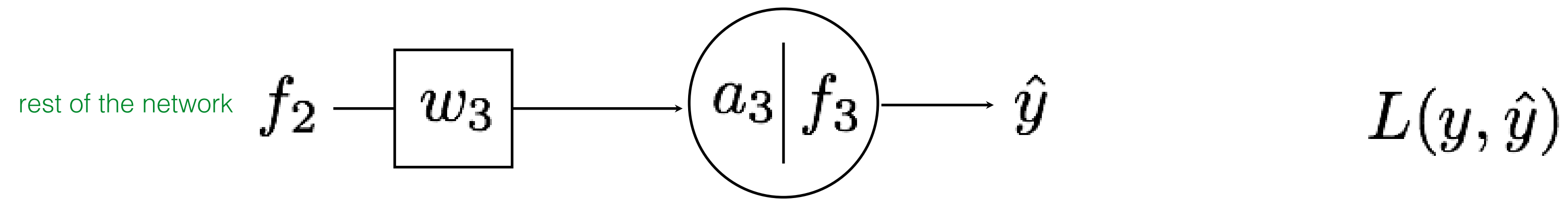
Chain Rule!



$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

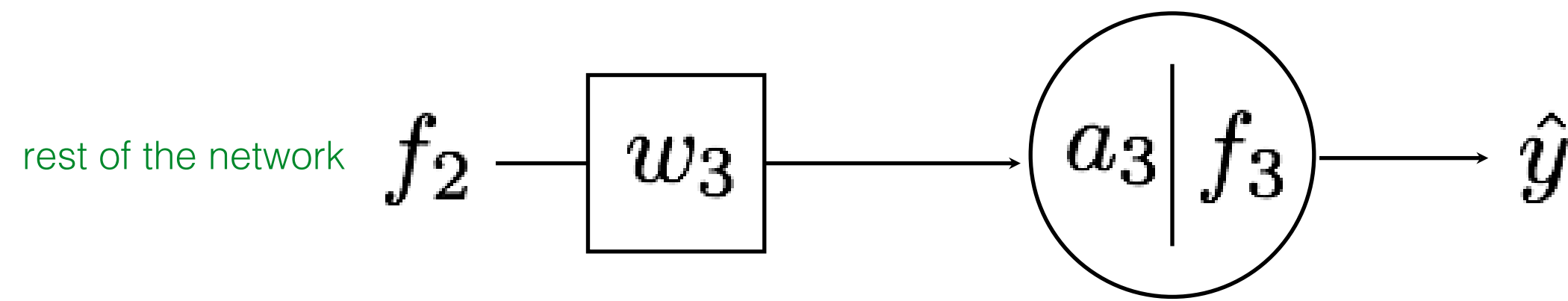
$$= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss

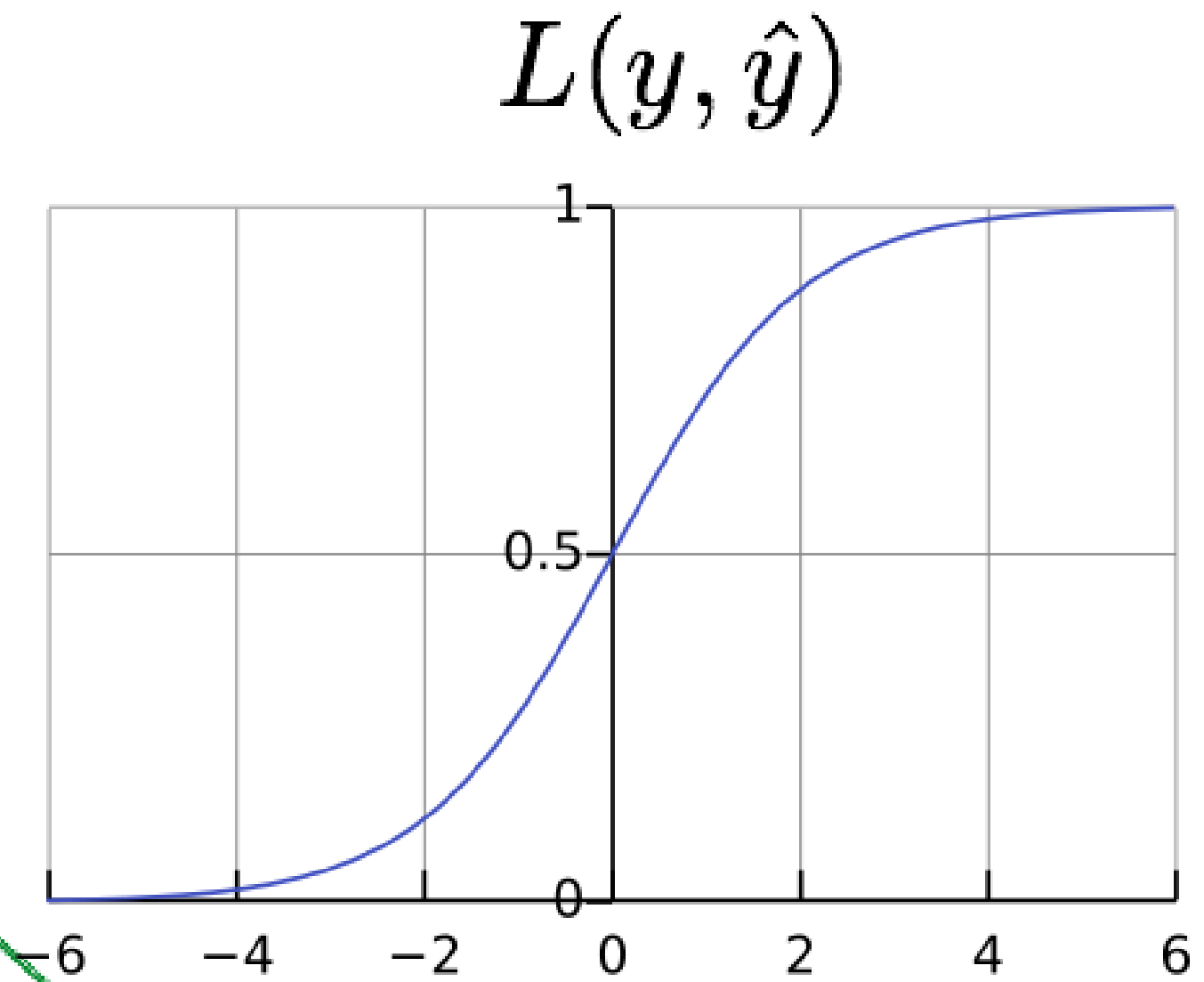


$$\begin{aligned}\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\ &= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}\end{aligned}$$

Let's use a Sigmoid function

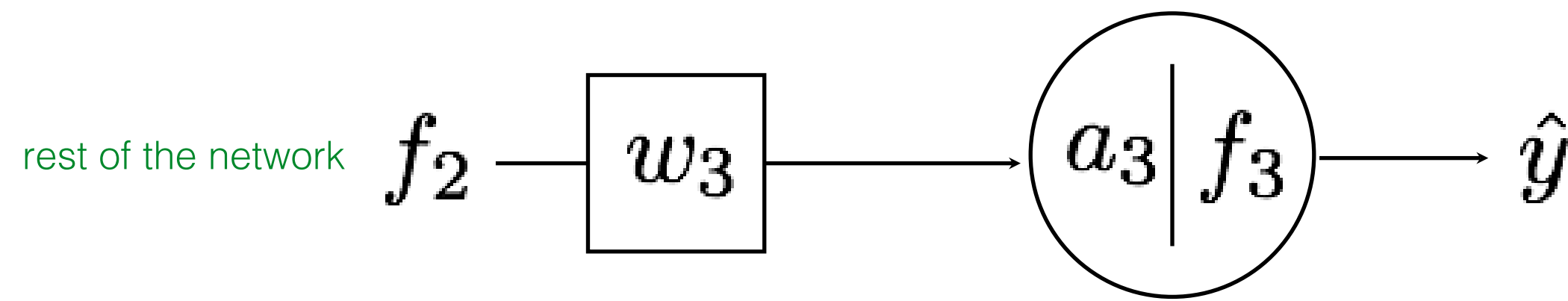


$$\begin{aligned}\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\ &= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}\end{aligned}$$

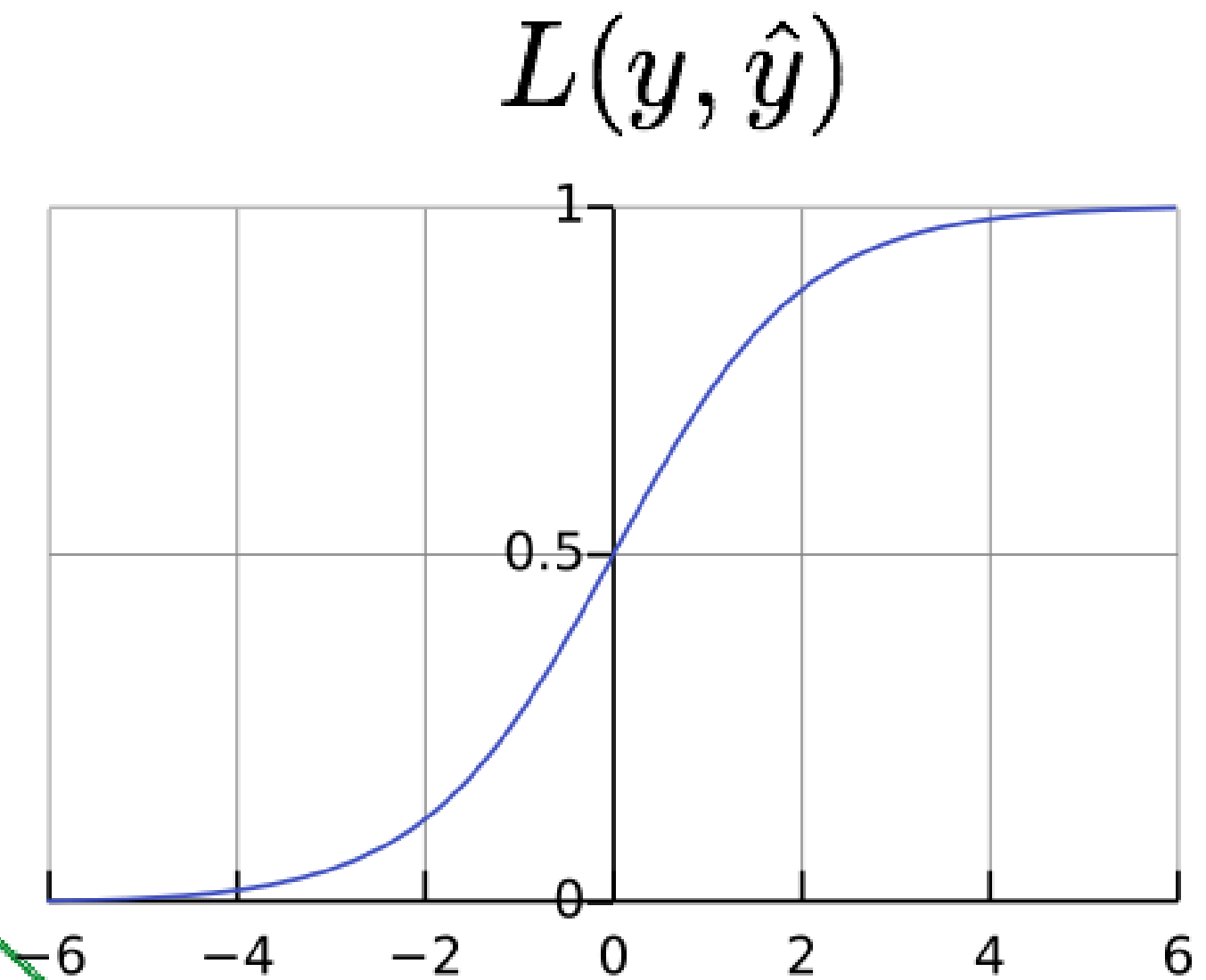


Let's use a Sigmoid function

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



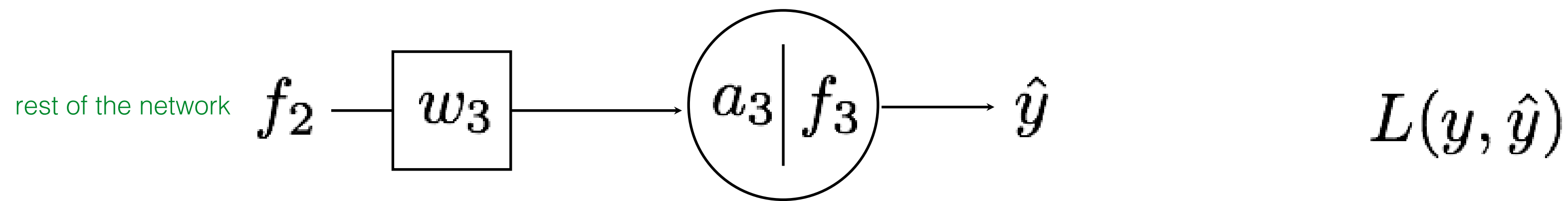
$$\begin{aligned}\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\ &= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}\end{aligned}$$



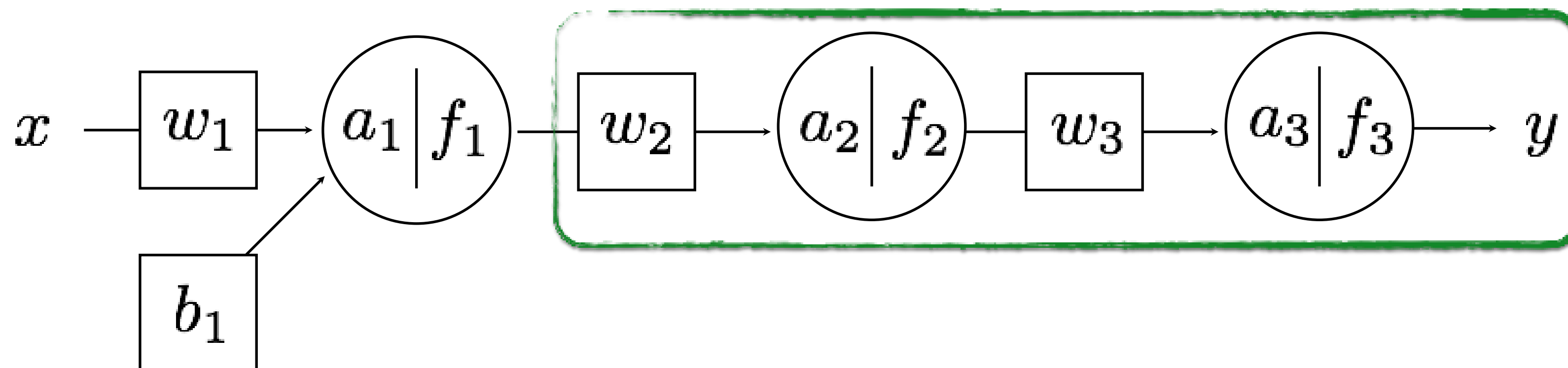
Let's use a Sigmoid function

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

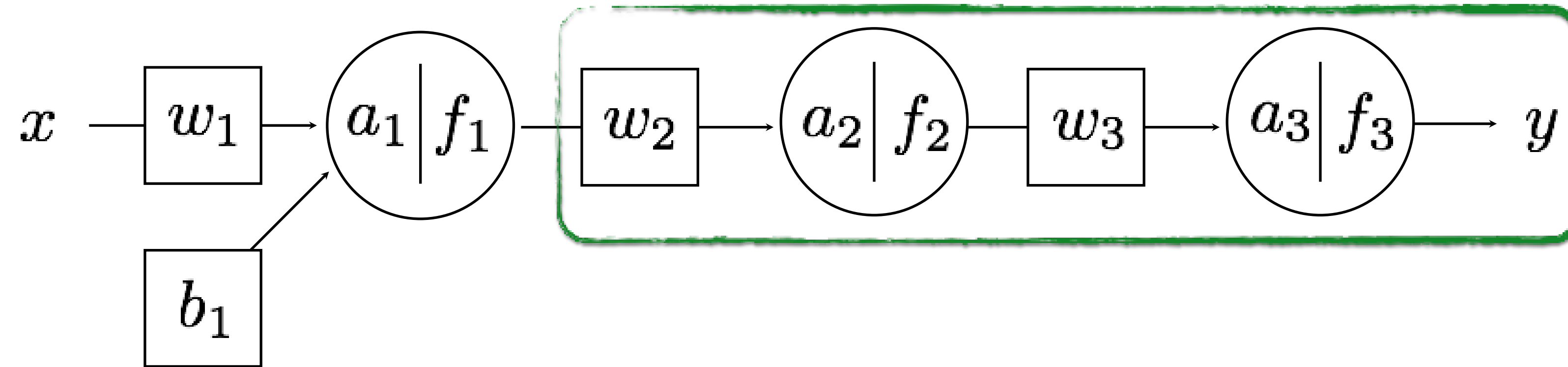
$$\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$



$$\begin{aligned}
 \frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
 &= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
 &= -\eta(y - \hat{y}) f_3(1 - f_3) \frac{\partial a_3}{\partial w_3} \\
 &= -\eta(y - \hat{y}) f_3(1 - f_3) f_2
 \end{aligned}$$



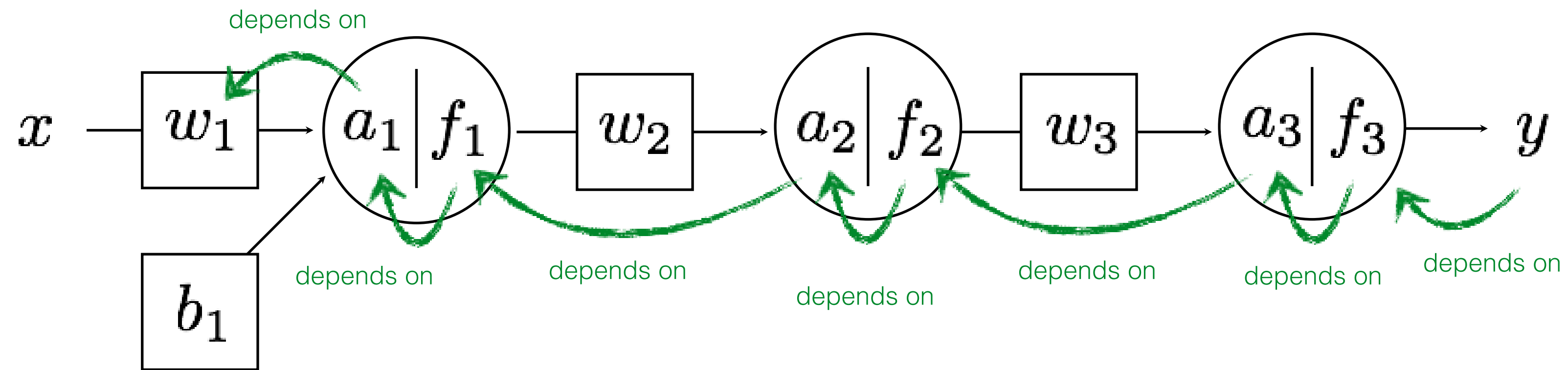
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

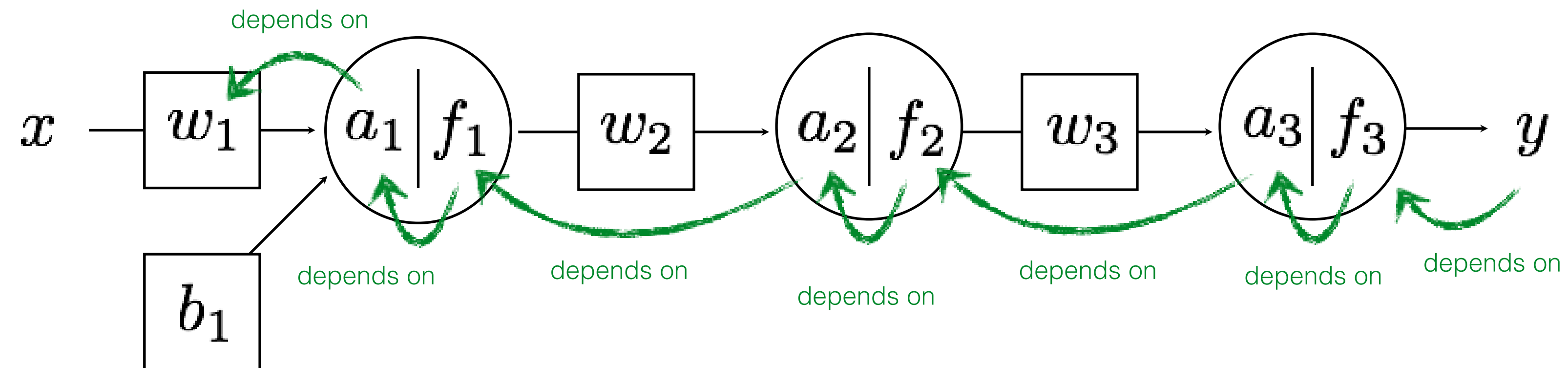
already computed.
re-use (propagate)!

The chain rule says...



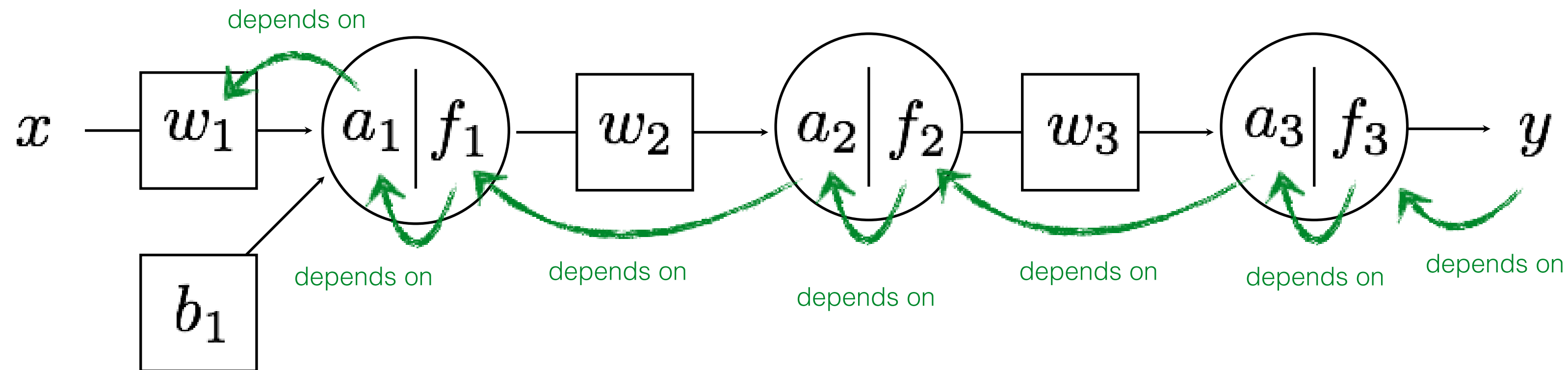
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

The chain rule says...

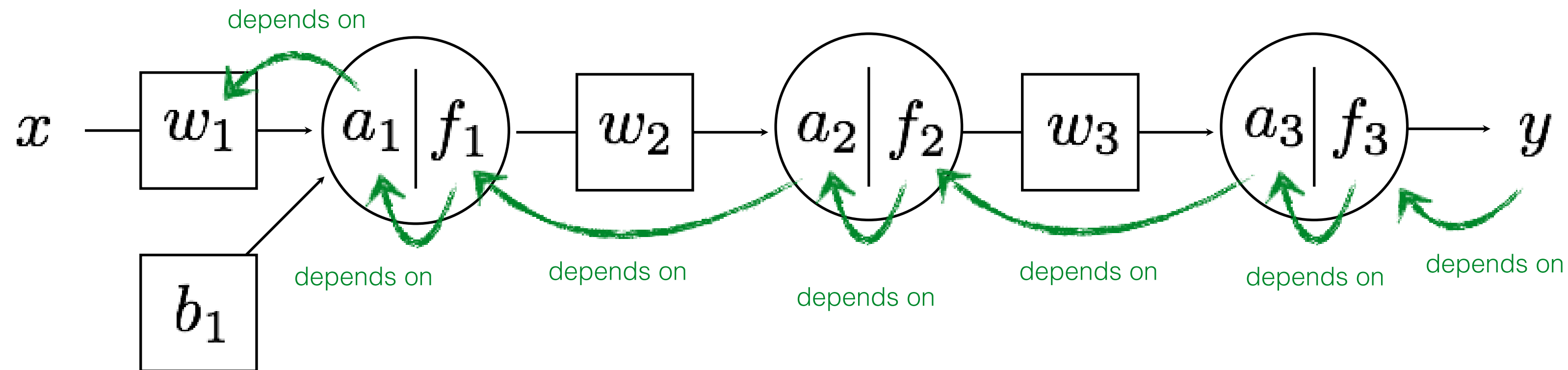


$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!



$$\begin{aligned}
 \frac{\partial \mathcal{L}}{\partial w_3} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
 \frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2} \\
 \frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} \\
 \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}
 \end{aligned}$$

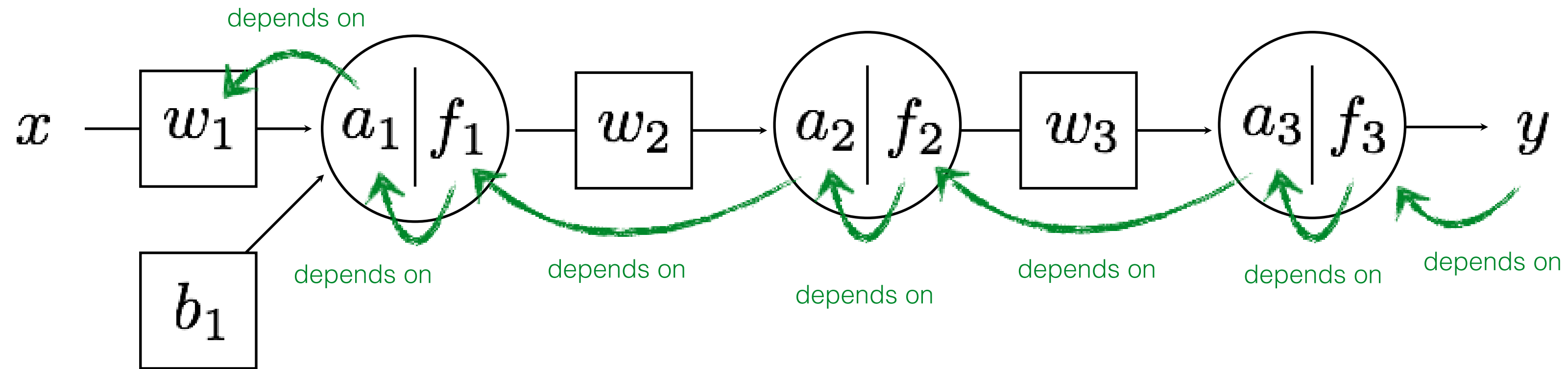


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

a. Forward pass

$$\hat{y} = f_{\text{MLP}}(x_i; \theta)$$

b. Compute Loss

$$\mathcal{L}_i$$

2. Update

a. Back Propagation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_3} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\ \frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2} \\ \frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} \\ \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial f_1} \frac{\partial f_1}{\partial a_1} \frac{\partial a_1}{\partial b}\end{aligned}$$

b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$

$$w_2 = w_2 - \eta \nabla w_2$$

$$w_1 = w_1 - \eta \nabla w_1$$

$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

b. Compute Loss $\mathcal{L}_i$

2. Update

a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of partial derivatives

b. Gradient update $\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$ vector of update equations

some history / controversy ...

[Nature](#) volume 323, pages 533–536 (1986)

[Nobel Prize in Physics 2024](#)

Learning representations by back-propagating errors

David E. Rumelhart*, **Geoffrey E. Hinton†**
& **Ronald J. Williams***

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA



<https://people.idsia.ch/~juergen/who-invented-backpropagation.html>

- An efficient application of the chain rule (first written down by Leibniz, 1676)
- Precursors to backpropagation appeared in optimal control theory since 1950s.
 - LeCun et al 1985 credits 1950s work by Pontryagin.
- Modern backpropagation was first published by Seppo Linnainmaa as "reverse mode of automatic differentiation" (1970) for discrete connected networks of nested differentiable functions.
- Rumelhart "independently developed" backpropagation. He did not cite previous work as he was "unaware" of them.

Why are NNs such good function approximators?

Universal Approximation Theorem:

“A neural network $\hat{f}(x)$ with a single hidden-layer can approximate any continuous real function $f(x)$ to within any arbitrary degree of accuracy ϵ given a sufficient number of neurons in the hidden layer”

Nice Explanation:

<https://www.deep-mind.org/2023/03/26/the-universal-approximation-theorem/>

Why are NNs such good function approximators?

Universal Approximation Theorem (UAT) limitations:

1. Does not specify what the network size should be
2. Only guarantees *function approximation* on the given training data; does not guarantee generalization on new test data
3. Only assures that approximation exists; does not provide insights on how to train the network to achieve that optimal function

“A neural network $\hat{f}(x)$ with a single hidden-layer can approximate any continuous real function $f(x)$ to within any arbitrary degree of accuracy ϵ given a sufficient number of neurons in the hidden layer”

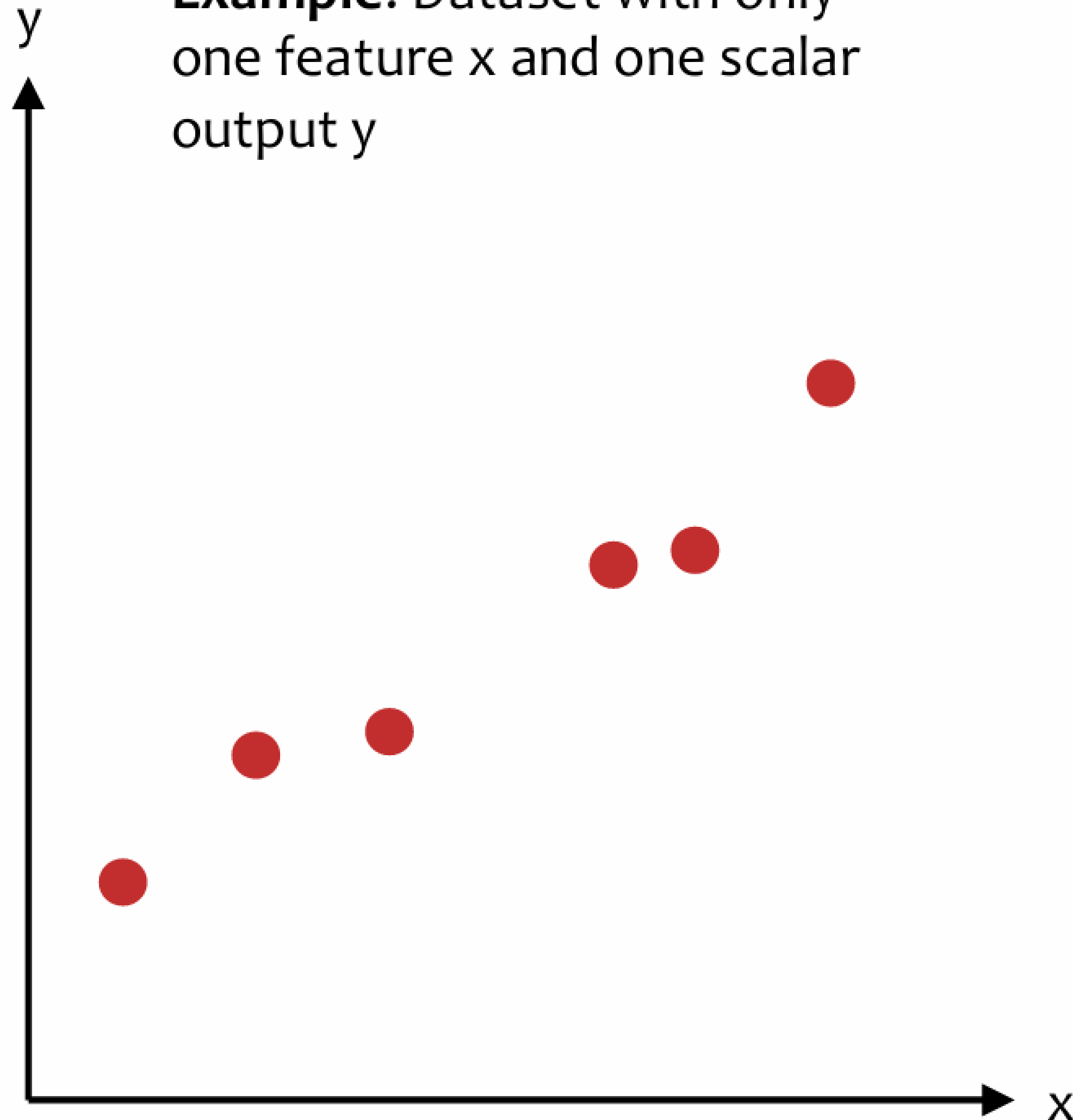
Training the NN to obtain
the “best” weights

The intuition ...

Recap: Linear Regression

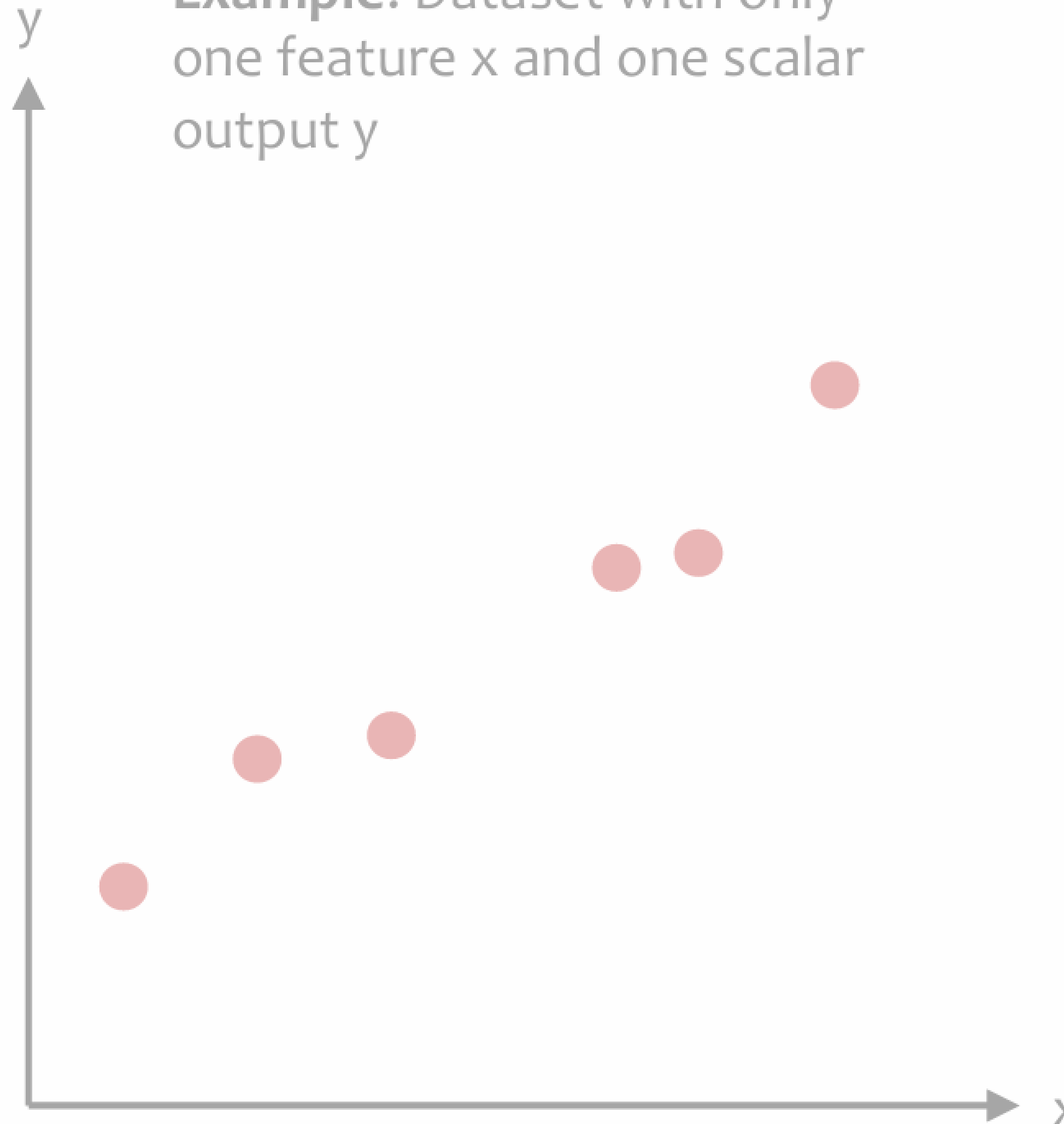
Example: Dataset with only one feature x and one scalar output y

Q: What is the function that best fits these points?

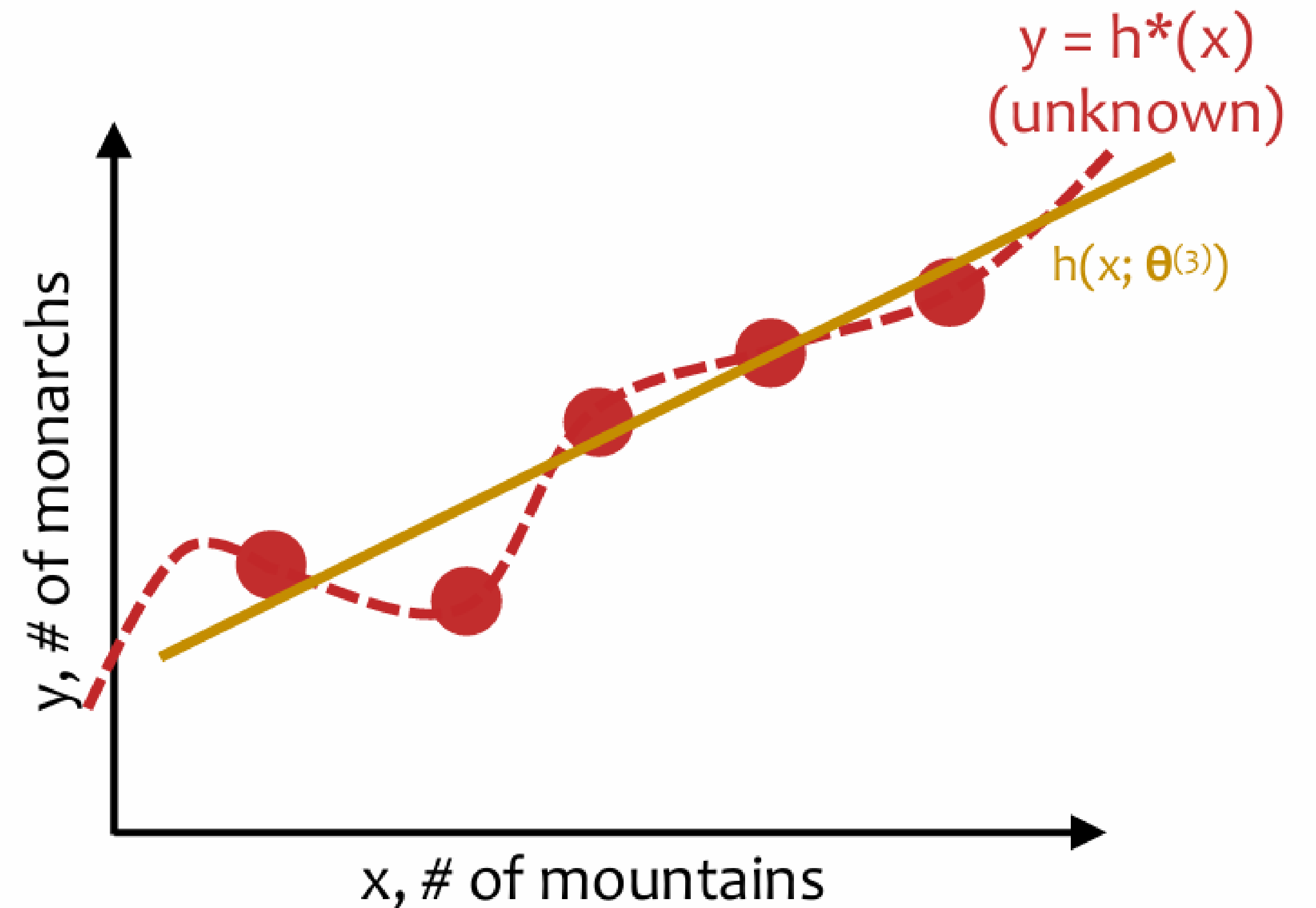


Recap: Linear Regression

Example: Dataset with only one feature x and one scalar output y



Q: What is the function that best fits these points?



Naïve Idea:

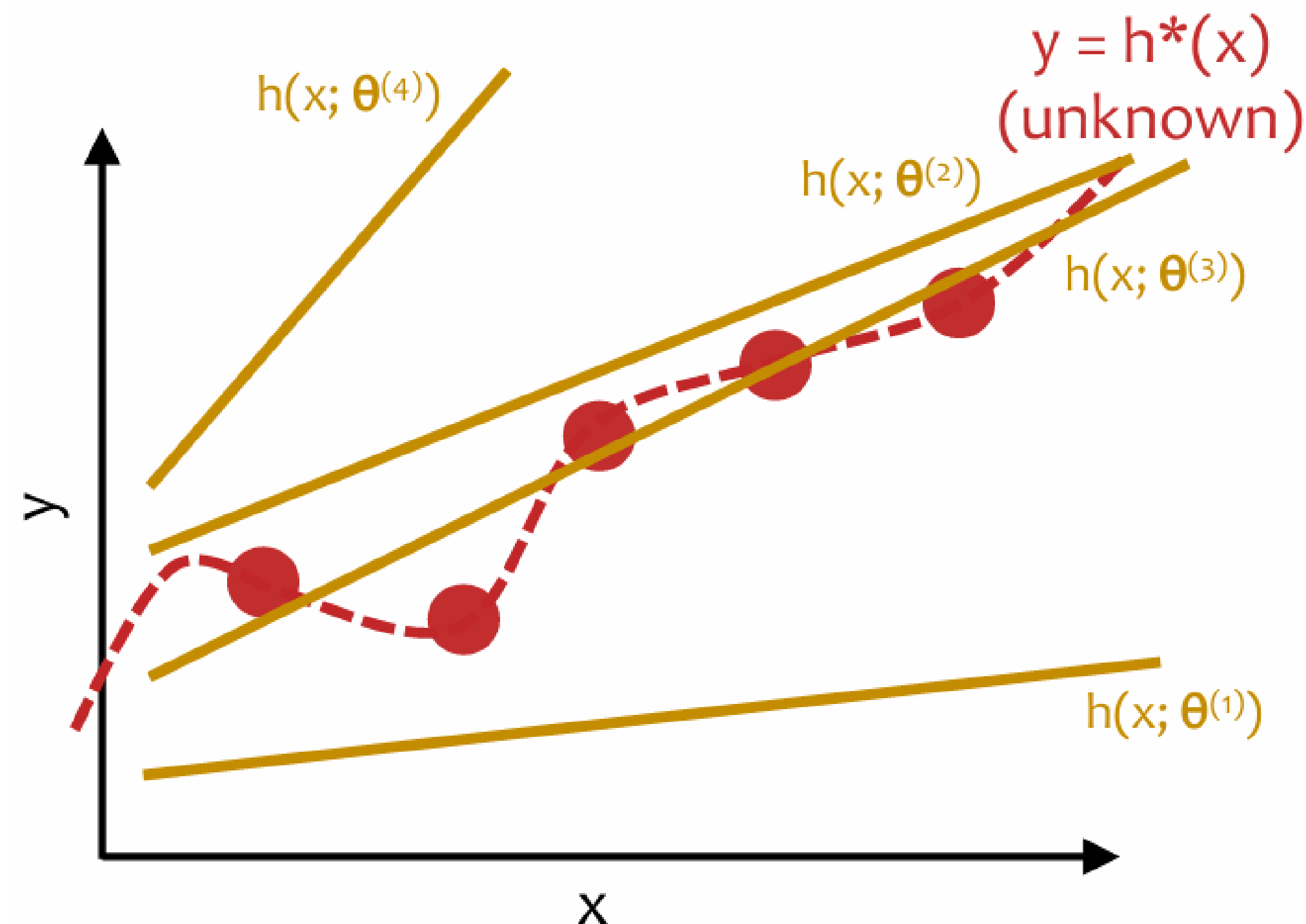
Linear Regression by Random Guessing

!!!

Linear Regression by Rand. Guessing

Optimization Method #0: Random Guessing

1. Pick a random θ
2. Evaluate $J(\theta)$
3. Repeat steps 1 and 2 many times
4. Return θ that gives smallest $J(\theta)$



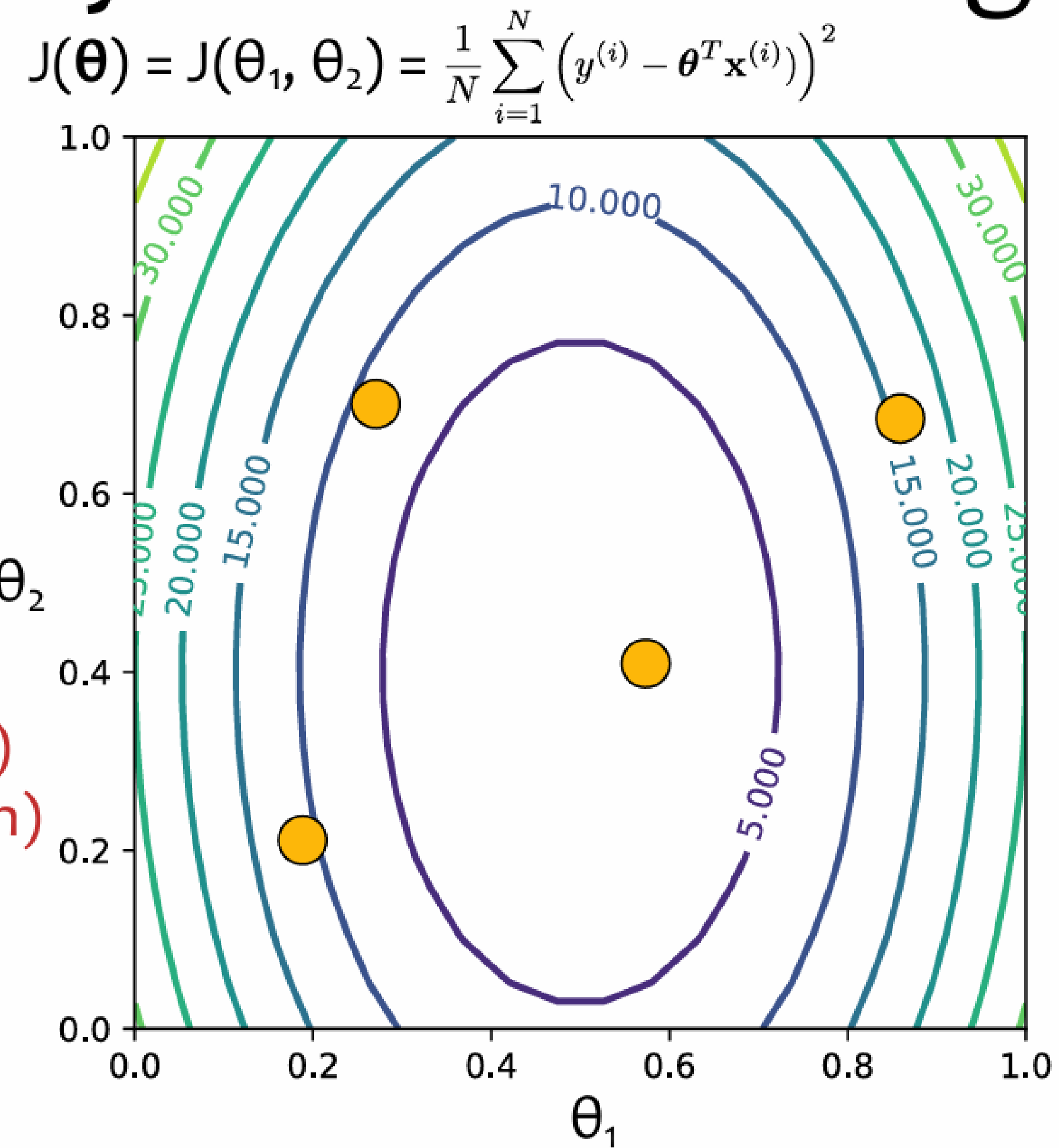
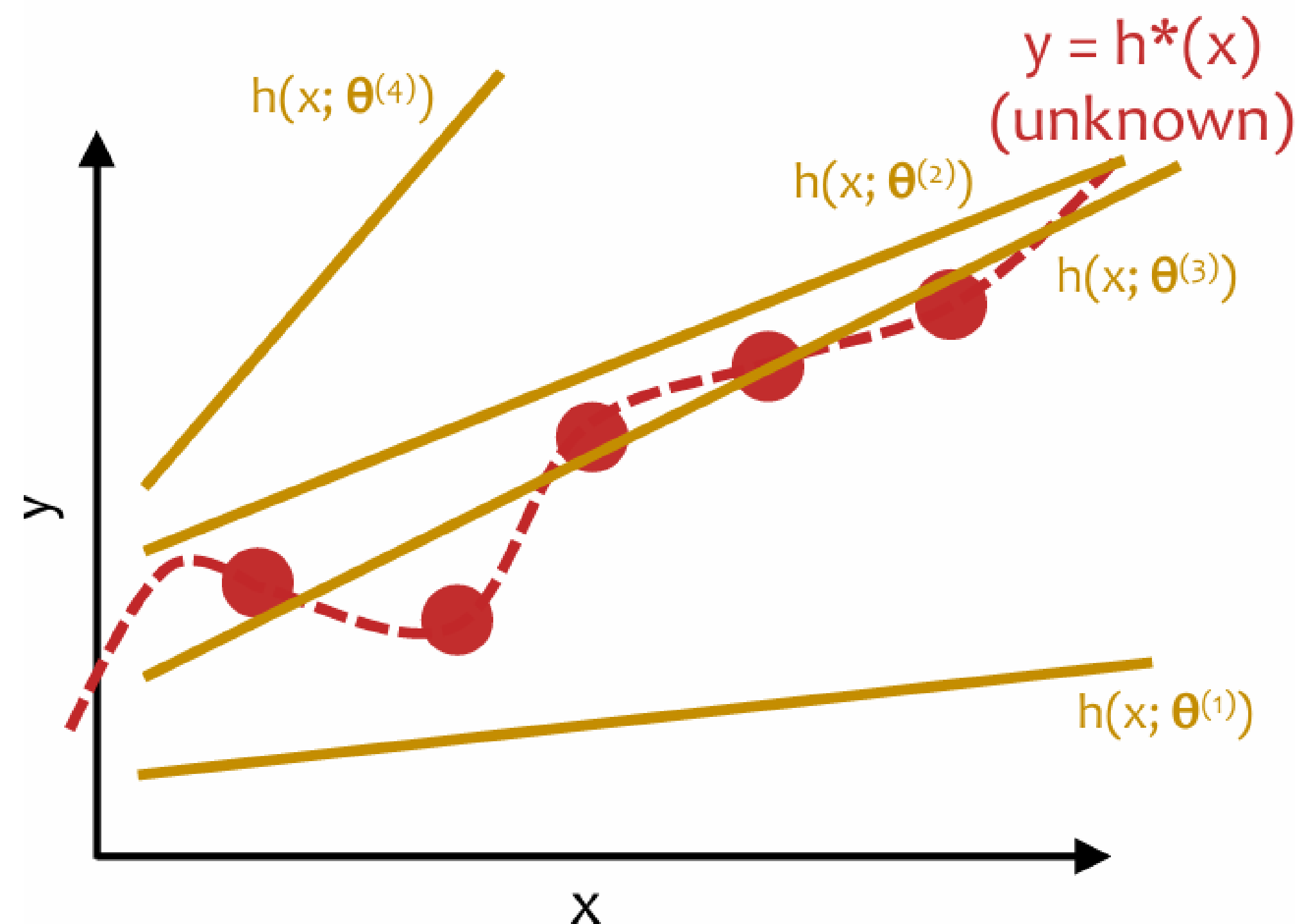
For Linear Regression:

- target function $h^*(x)$ is **unknown**
- only have access to $h^*(x)$ through **training examples** $(x^{(i)}, y^{(i)})$
- want $h(x; \theta^{(t)})$ that **best approximates** $h^*(x)$
- **enable generalization** w/inductive bias that restricts hypothesis class to **linear functions**

Linear Regression by Rand. Guessing

Optimization Method #0: Random Guessing

1. Pick a random θ
2. Evaluate $J(\theta)$
3. Repeat steps 1 and 2 many times
4. Return θ that gives smallest $J(\theta)$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.2	0.2	10.4
2	0.3	0.7	7.2
3	0.6	0.4	1.0
4	0.9	0.7	16.2

Better Idea:

An optimization algorithm called

“Gradient Descent”

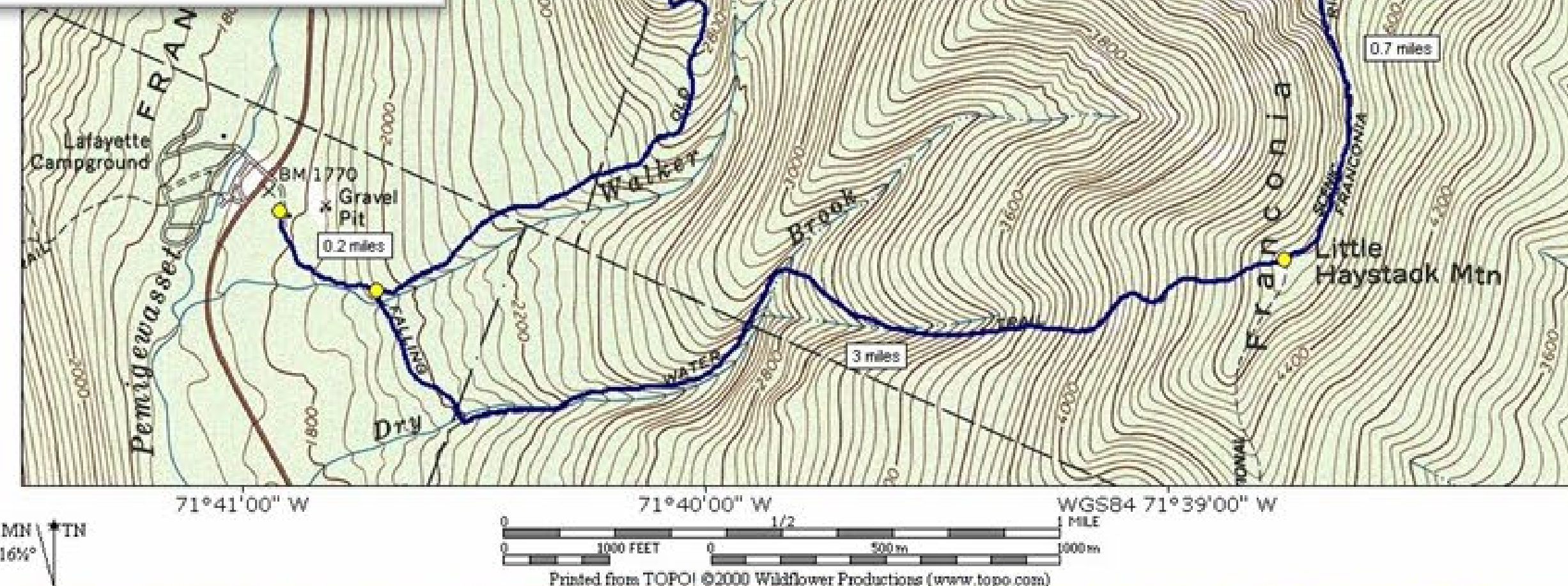
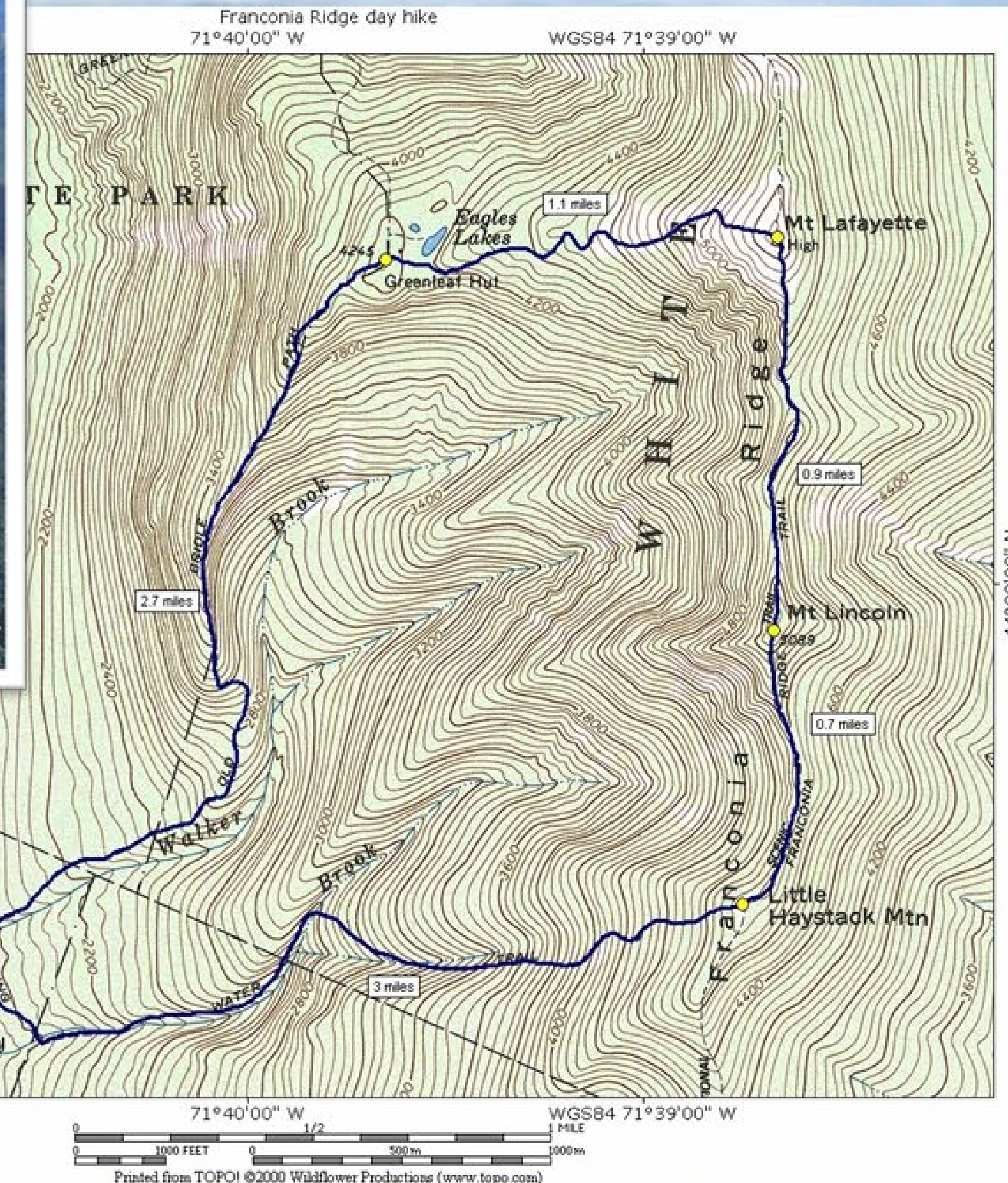
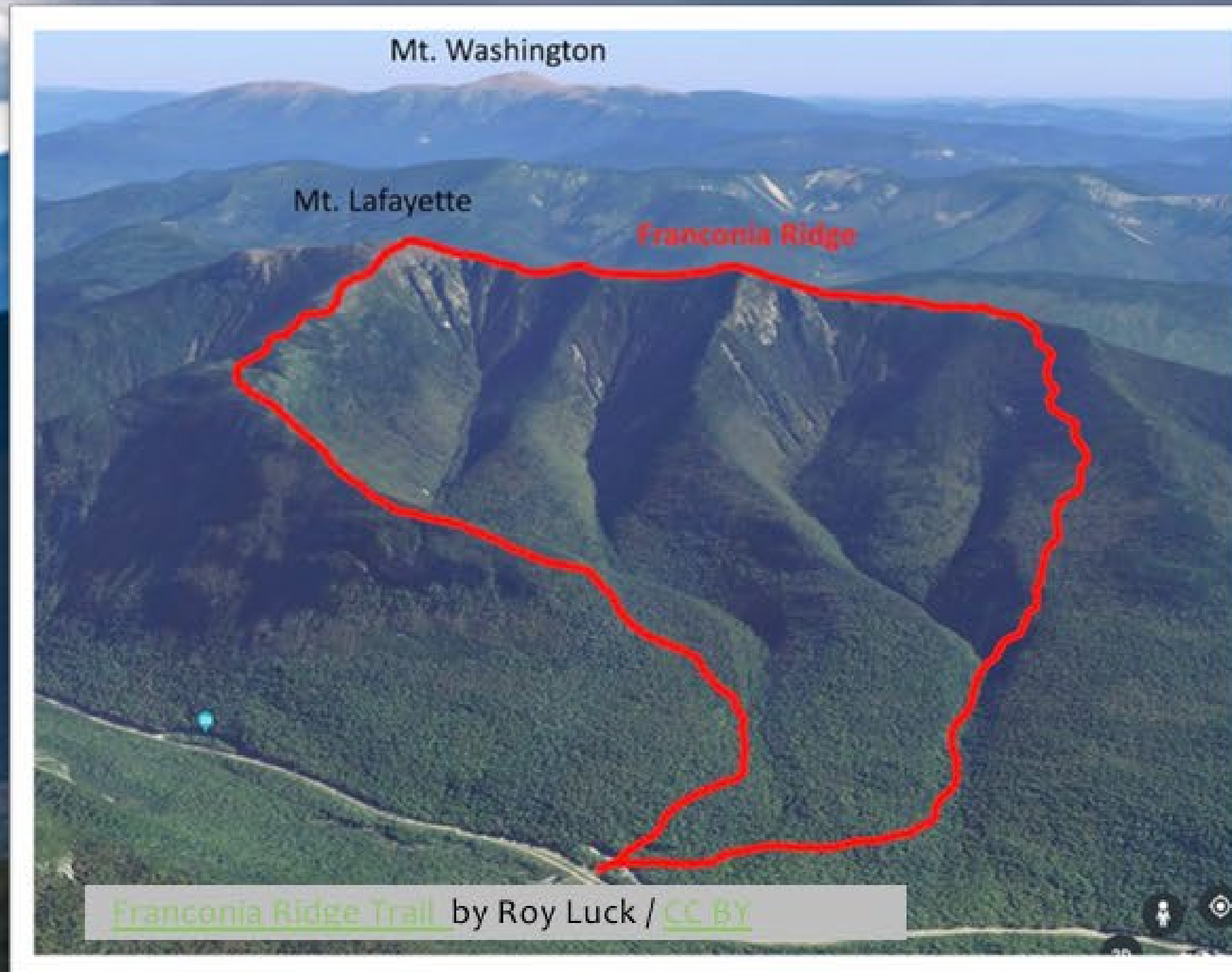
Recall: Gradient of a vector

Def: The gradient of $J : \mathbb{R}^M \rightarrow \mathbb{R}$ is

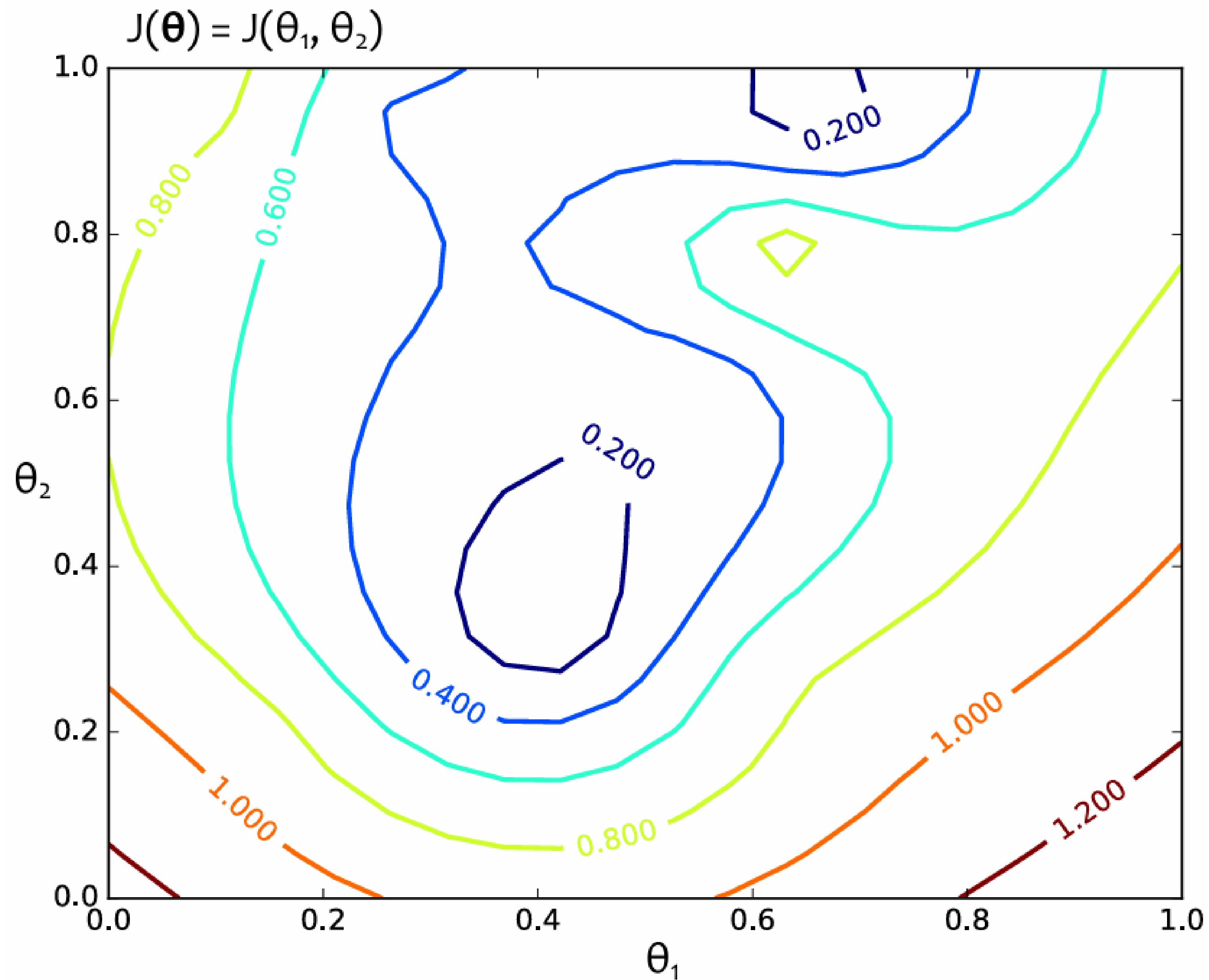
$$\nabla J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_M} \end{bmatrix}$$

Each entry is a first-order partial derivative

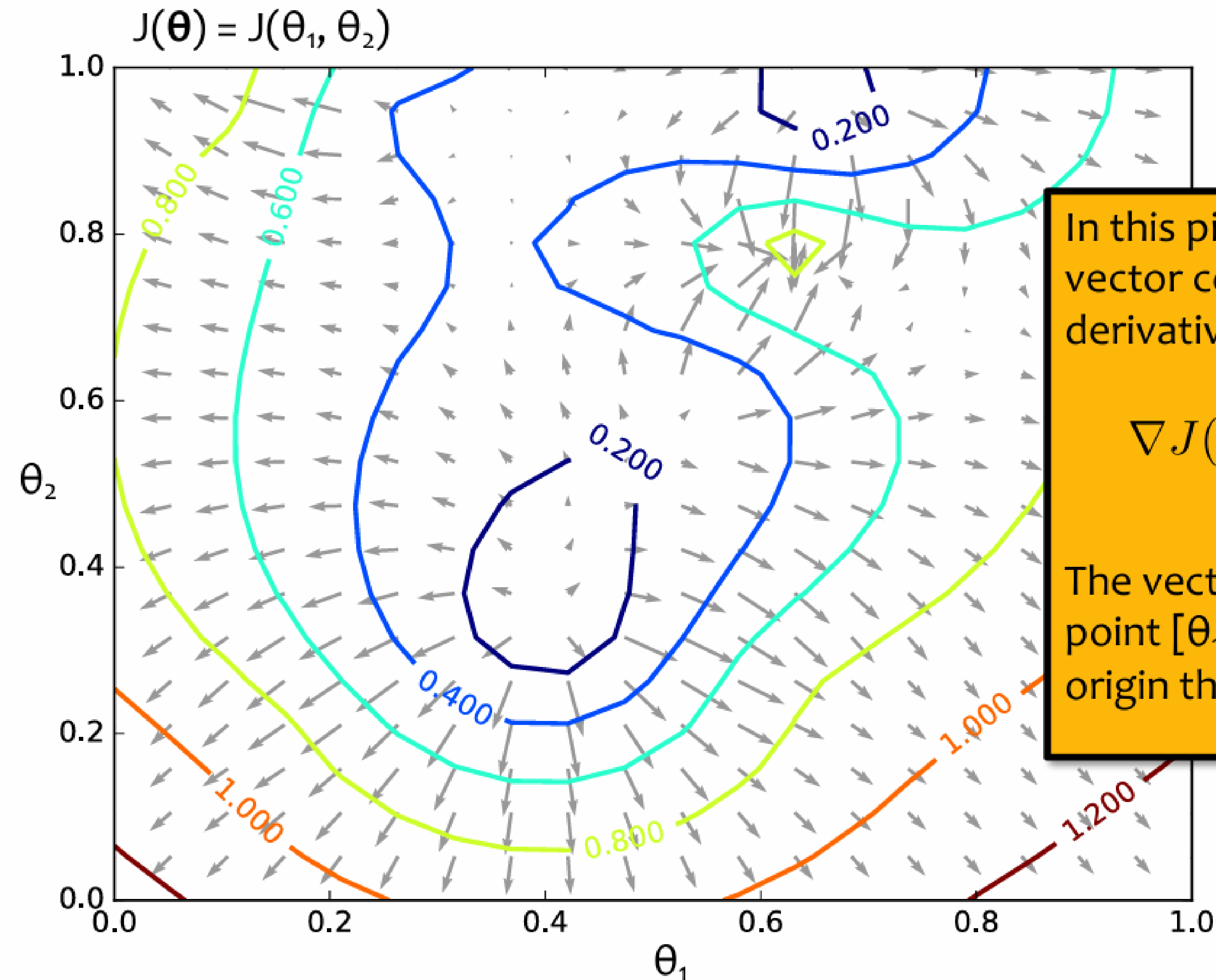
Topographical Maps



Gradients



Gradients



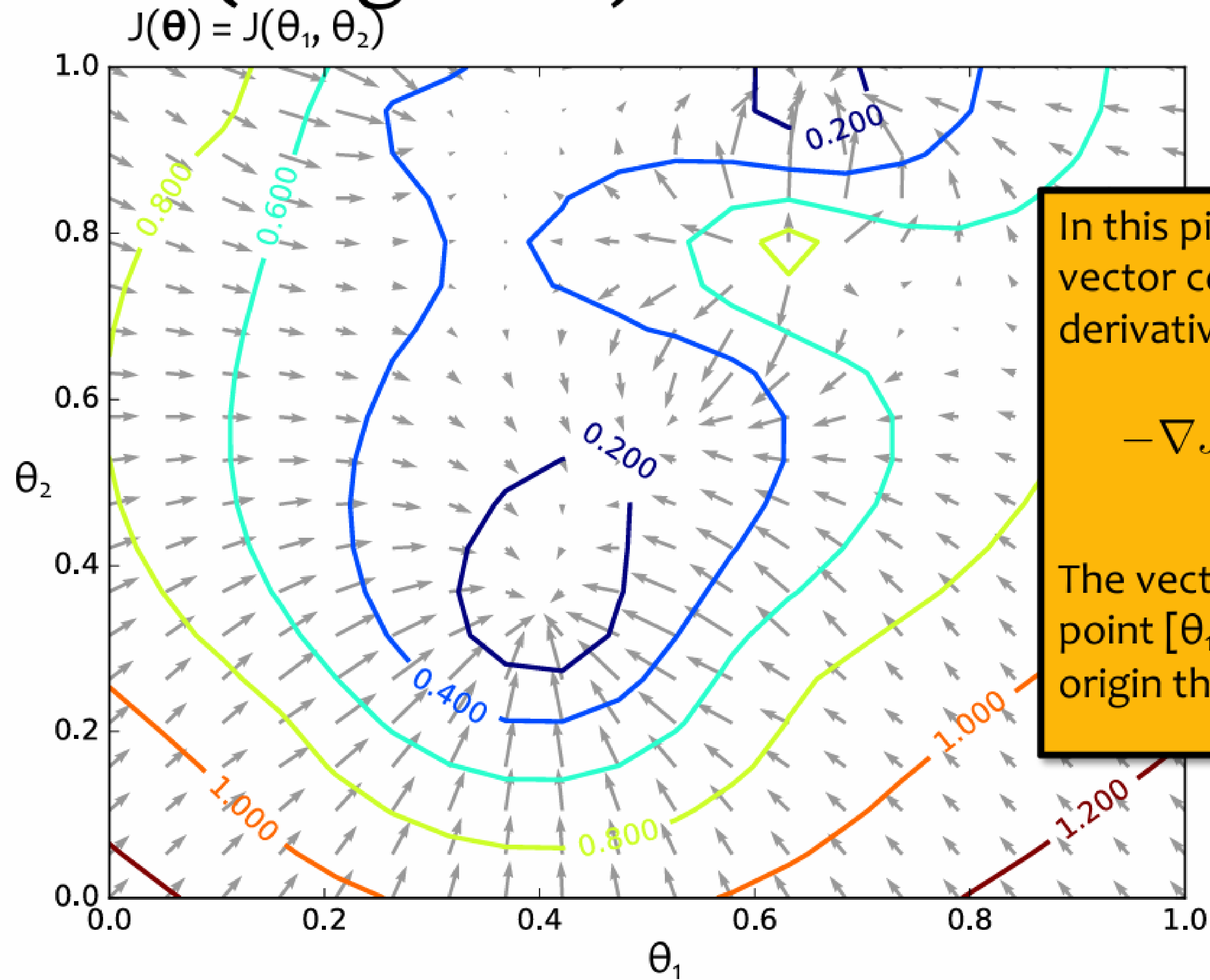
In this picture, each arrow is a 2D vector consisting of two partial derivatives.

$$\nabla J(\theta_1, \theta_2) = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \end{bmatrix}$$

The vector is evaluated at the point $[\theta_1, \theta_2]^T$ and plotted with its origin there as well.

These are the **gradients** that Gradient **Ascent** would follow.

(Negative) Gradients



In this picture, each arrow is a 2D vector consisting of two partial derivatives.

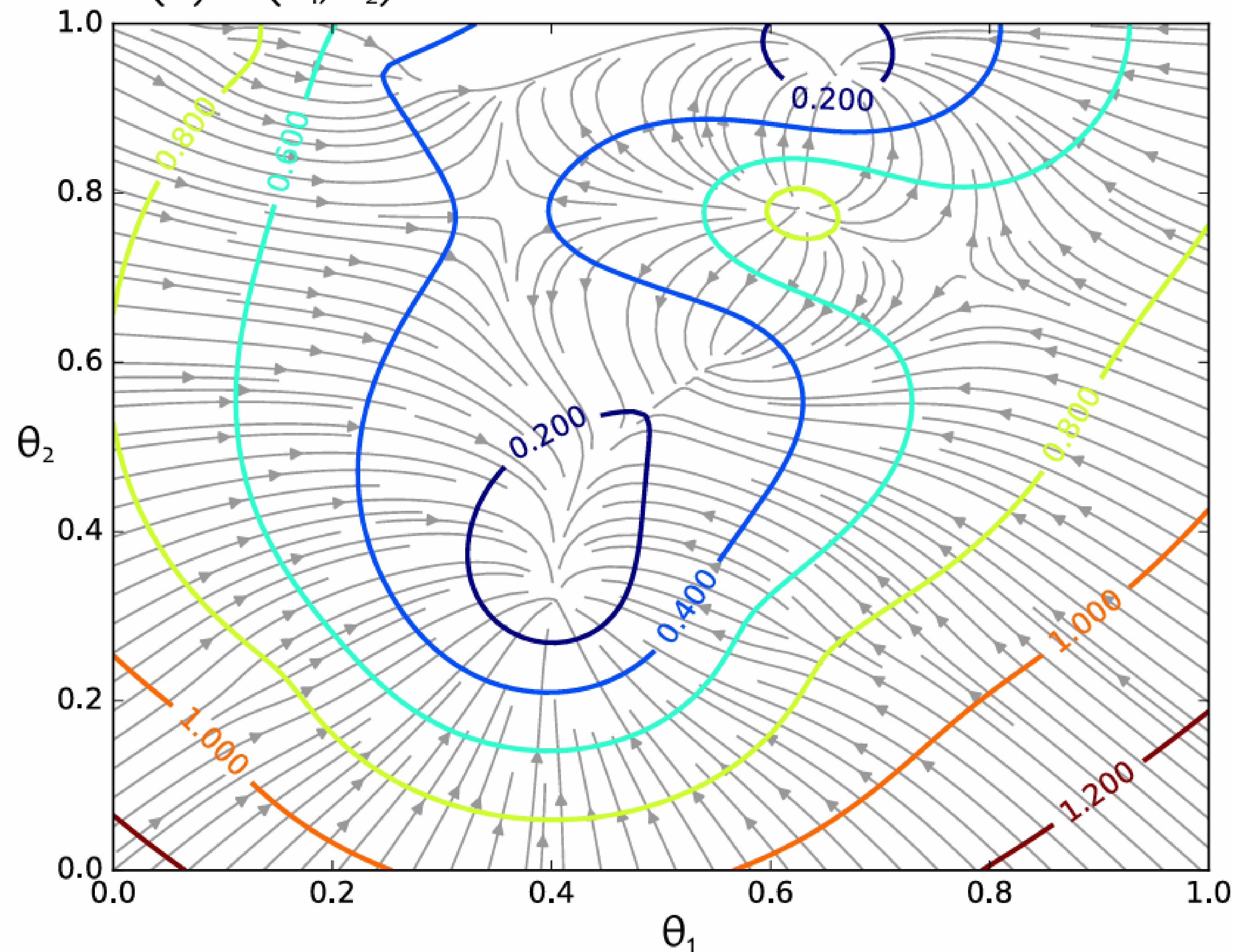
$$-\nabla J(\theta_1, \theta_2) = \begin{bmatrix} -\frac{\partial J}{\partial \theta_1} \\ -\frac{\partial J}{\partial \theta_2} \end{bmatrix}$$

The vector is evaluated at the point $[\theta_1, \theta_2]^T$ and plotted with its origin there as well.

These are the **negative** gradients that Gradient **Descent** would follow.

(Negative) Gradient *Paths*

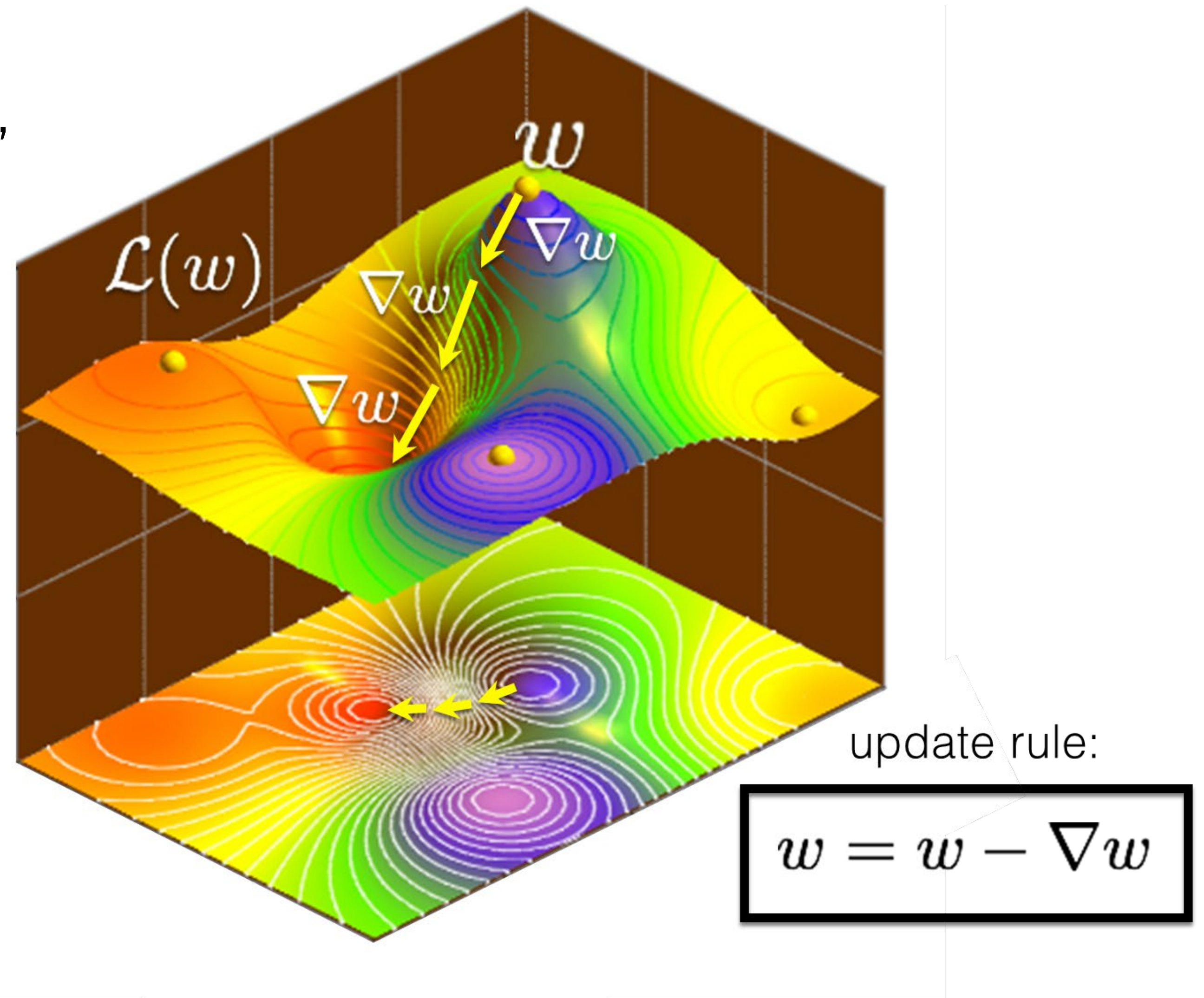
$$J(\theta) = J(\theta_1, \theta_2)$$



Shown are the **paths** that Gradient Descent would follow if it were making **infinitesimally small steps**.

Gradient Descent

- Go down the path of “steepest descent”

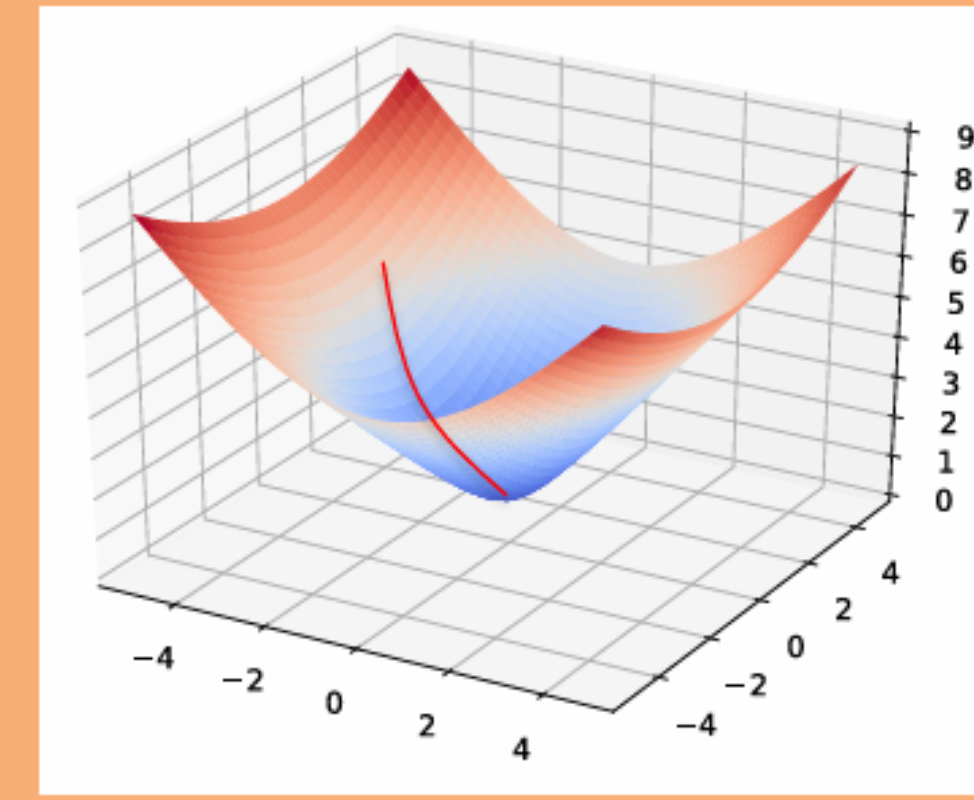


Gradient Descent

- Go down the path of steepest descent

Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )  
2:    $\theta \leftarrow \theta^{(0)}$   
3:   while not converged do  
4:      $\theta \leftarrow \theta - \gamma \nabla_{\theta} J(\theta)$   
5:   return  $\theta$ 
```



In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

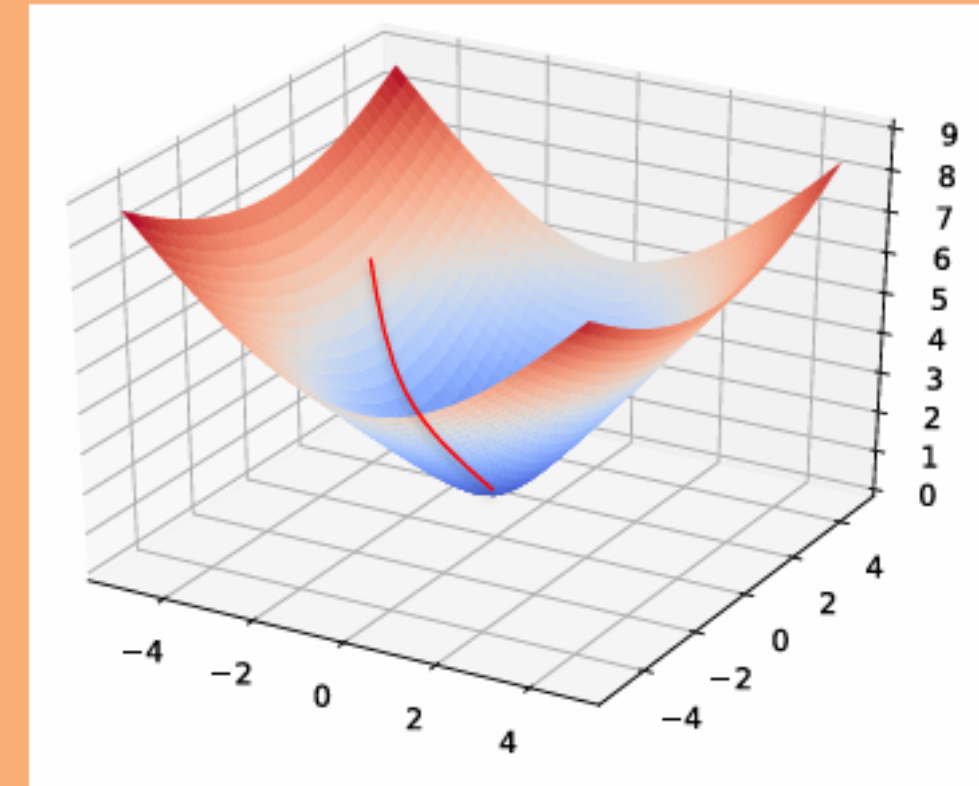
$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \frac{d}{d\theta_1} J(\theta) \\ \frac{d}{d\theta_2} J(\theta) \\ \vdots \\ \frac{d}{d\theta_M} J(\theta) \end{bmatrix}$$

Gradient Descent

- Go down the path of steepest descent

Algorithm 1 Gradient Descent

```
1: procedure GD( $\mathcal{D}$ ,  $\theta^{(0)}$ )  
2:    $\theta \leftarrow \theta^{(0)}$   
3:   while not converged do  
4:      $\theta \leftarrow \theta - \gamma \nabla_{\theta} J(\theta)$   
5:   return  $\theta$ 
```



There are many possible ways to detect **convergence**.
For example, we could check whether the L2 norm of the gradient is below some small tolerance.

$$\|\nabla_{\theta} J(\theta)\|_2 \leq \epsilon$$

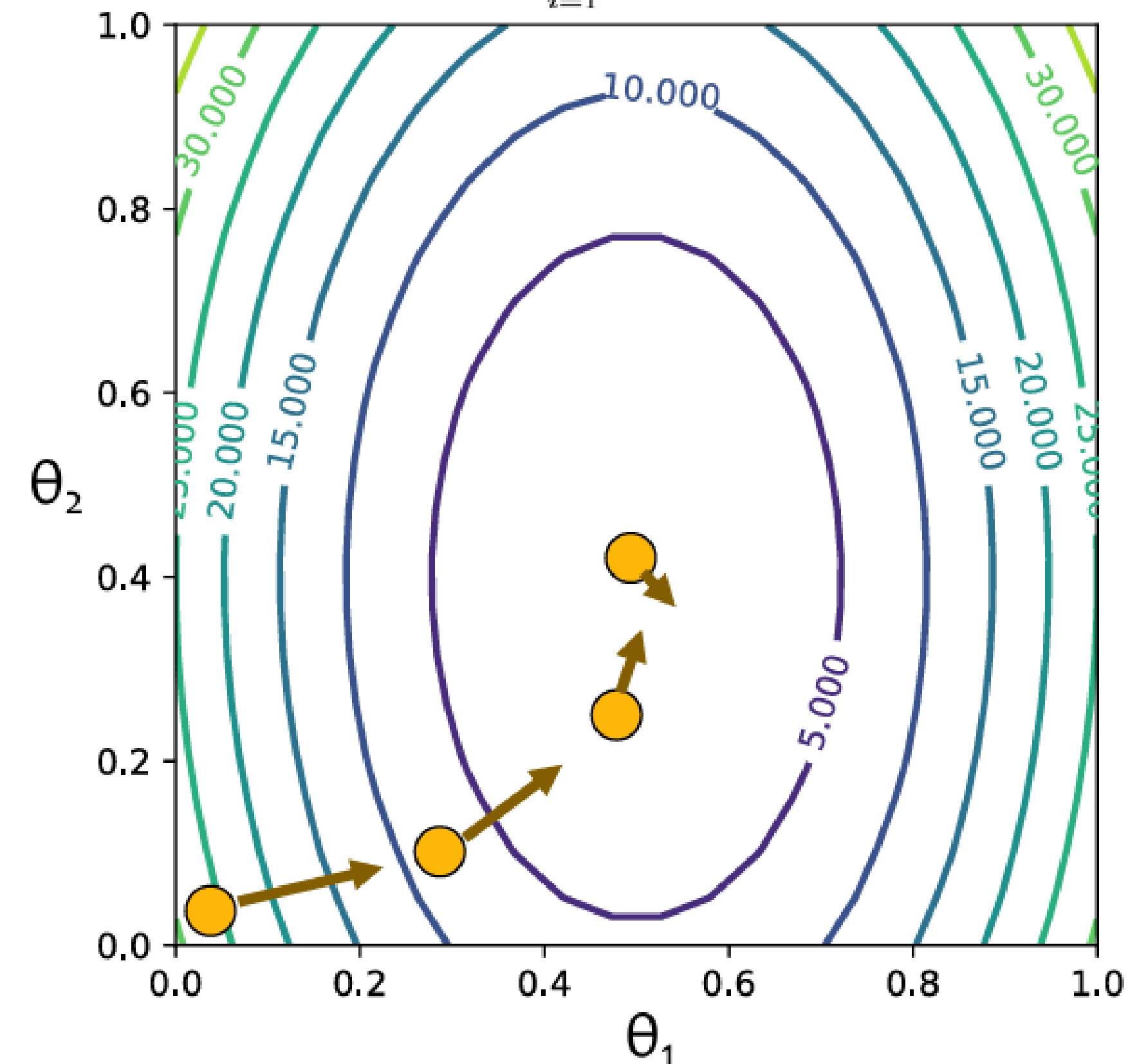
Alternatively we could check that the reduction in the objective function from one iteration to the next is small.

Linear Regression by Gradient Desc.

Optimization Method #1: Gradient Descent

1. Pick a random θ
2. Repeat:
 - a. Evaluate gradient $\nabla J(\theta)$
 - b. Step opposite gradient
3. Return θ that gives smallest $J(\theta)$

$$J(\theta) = J(\theta_1, \theta_2) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2$$



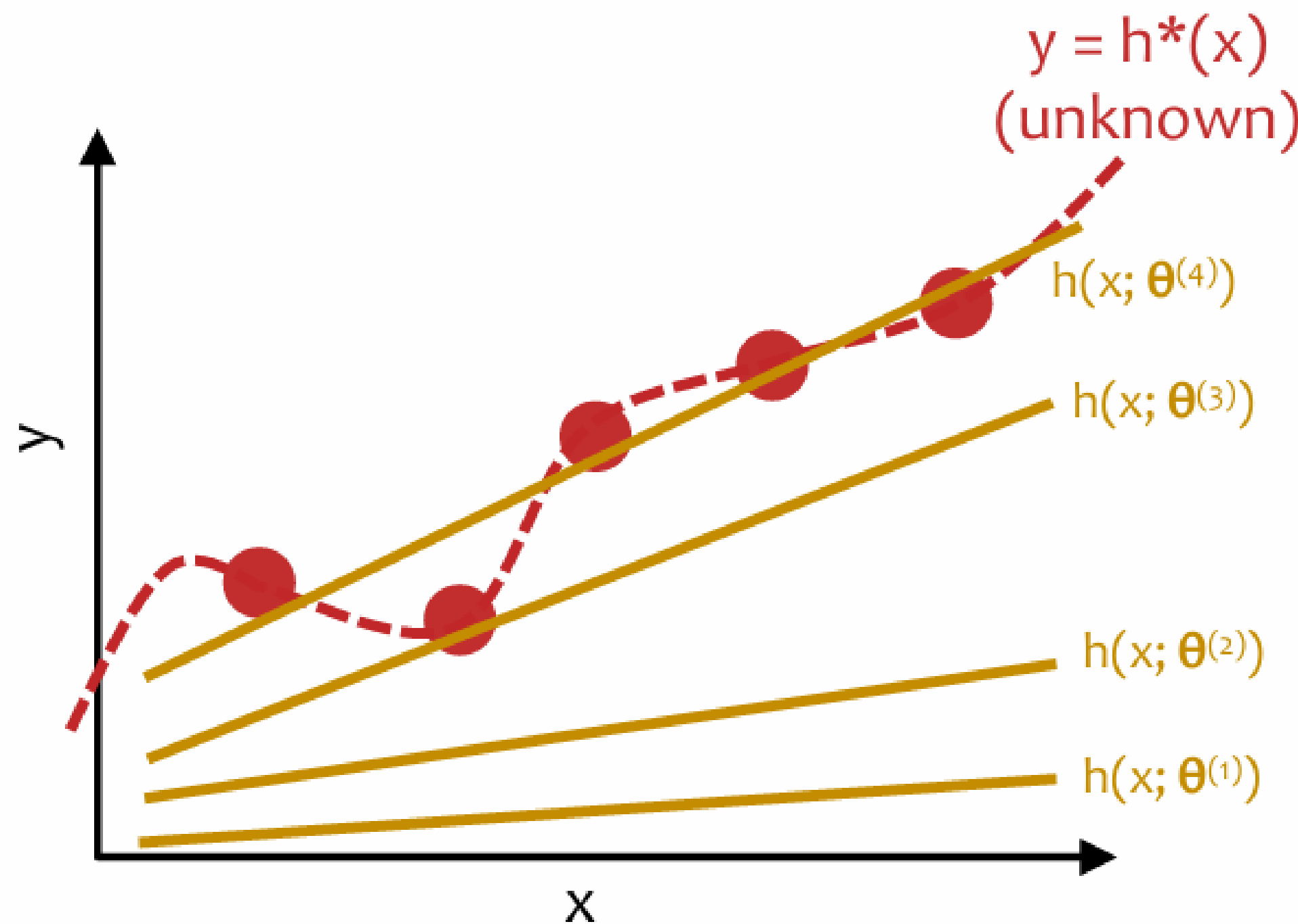
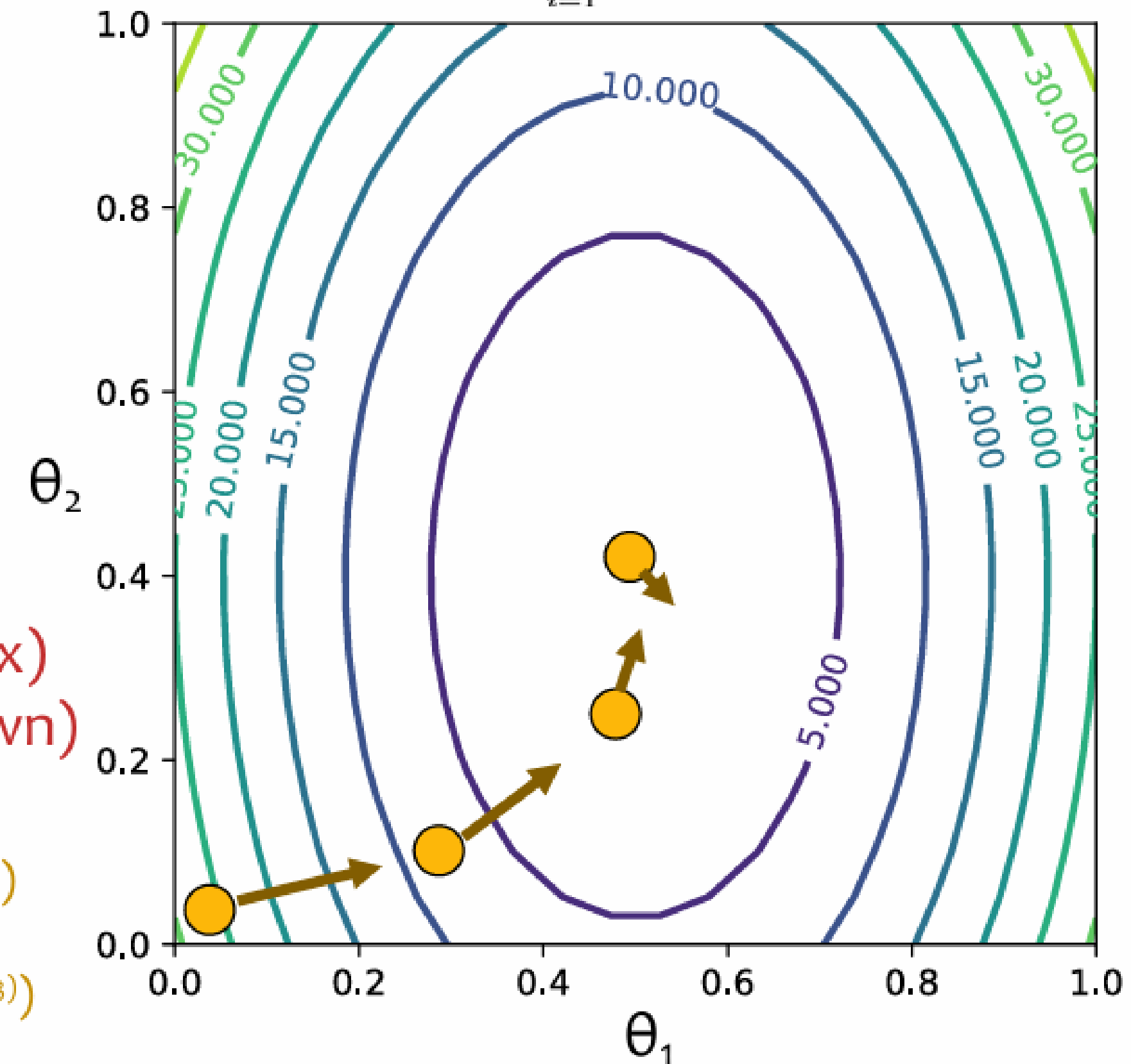
t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Linear Regression by Gradient Desc.

Optimization Method #1: Gradient Descent

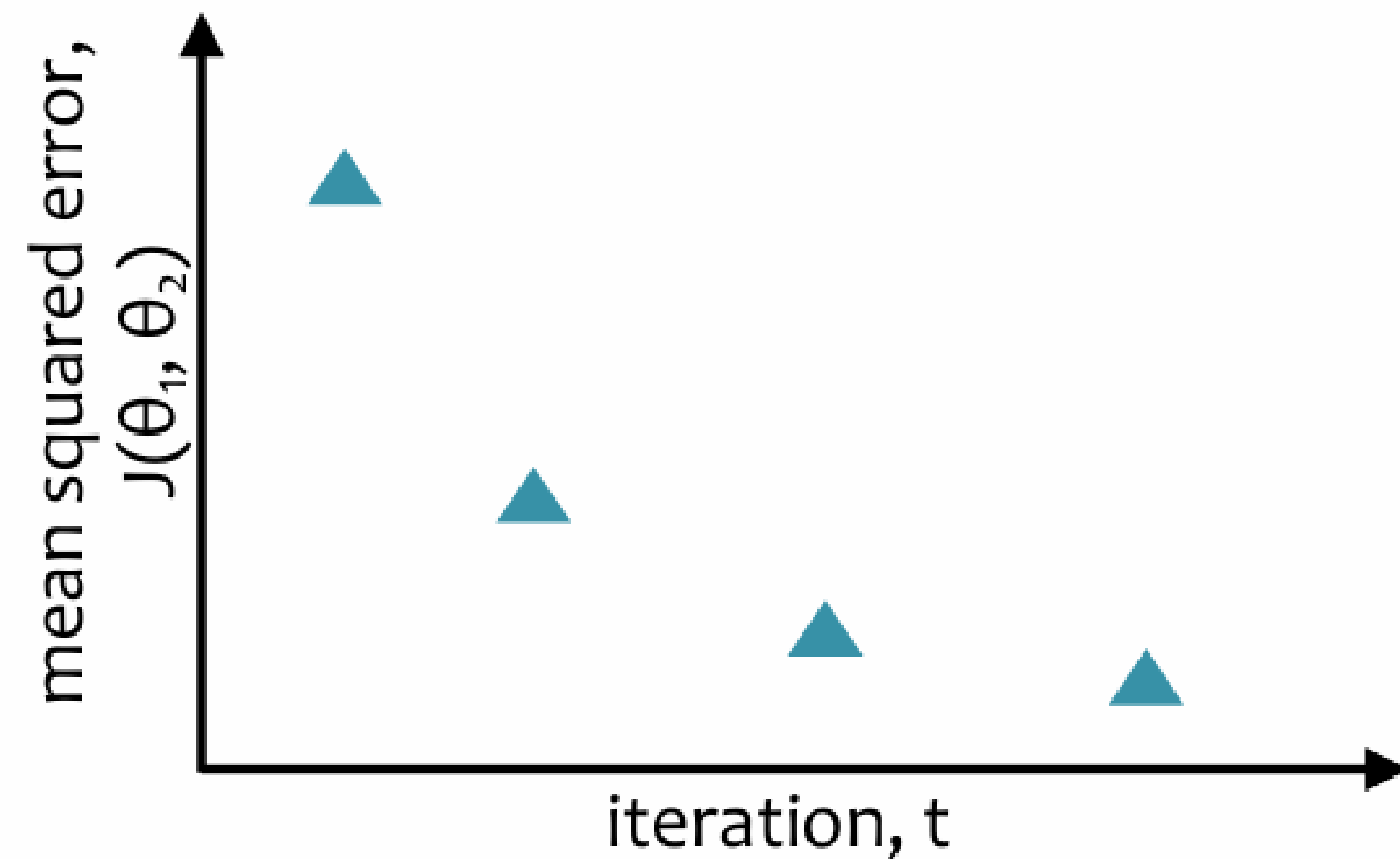
1. Pick a random θ
2. Repeat:
 - a. Evaluate gradient $\nabla J(\theta)$
 - b. Step opposite gradient
3. Return θ that gives smallest $J(\theta)$

$$J(\theta) = J(\theta_1, \theta_2) = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - \theta^T \mathbf{x}^{(i)})^2$$

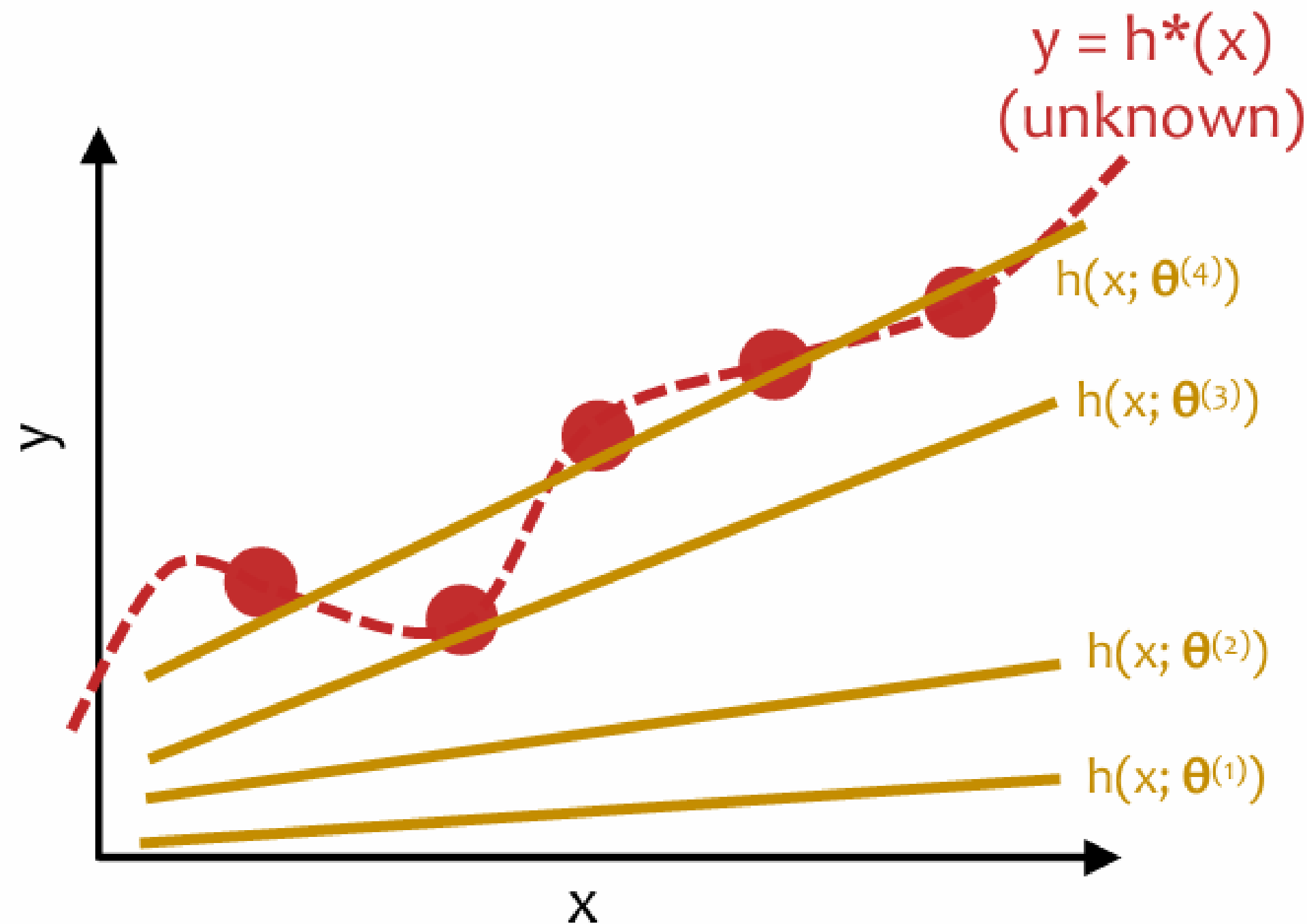
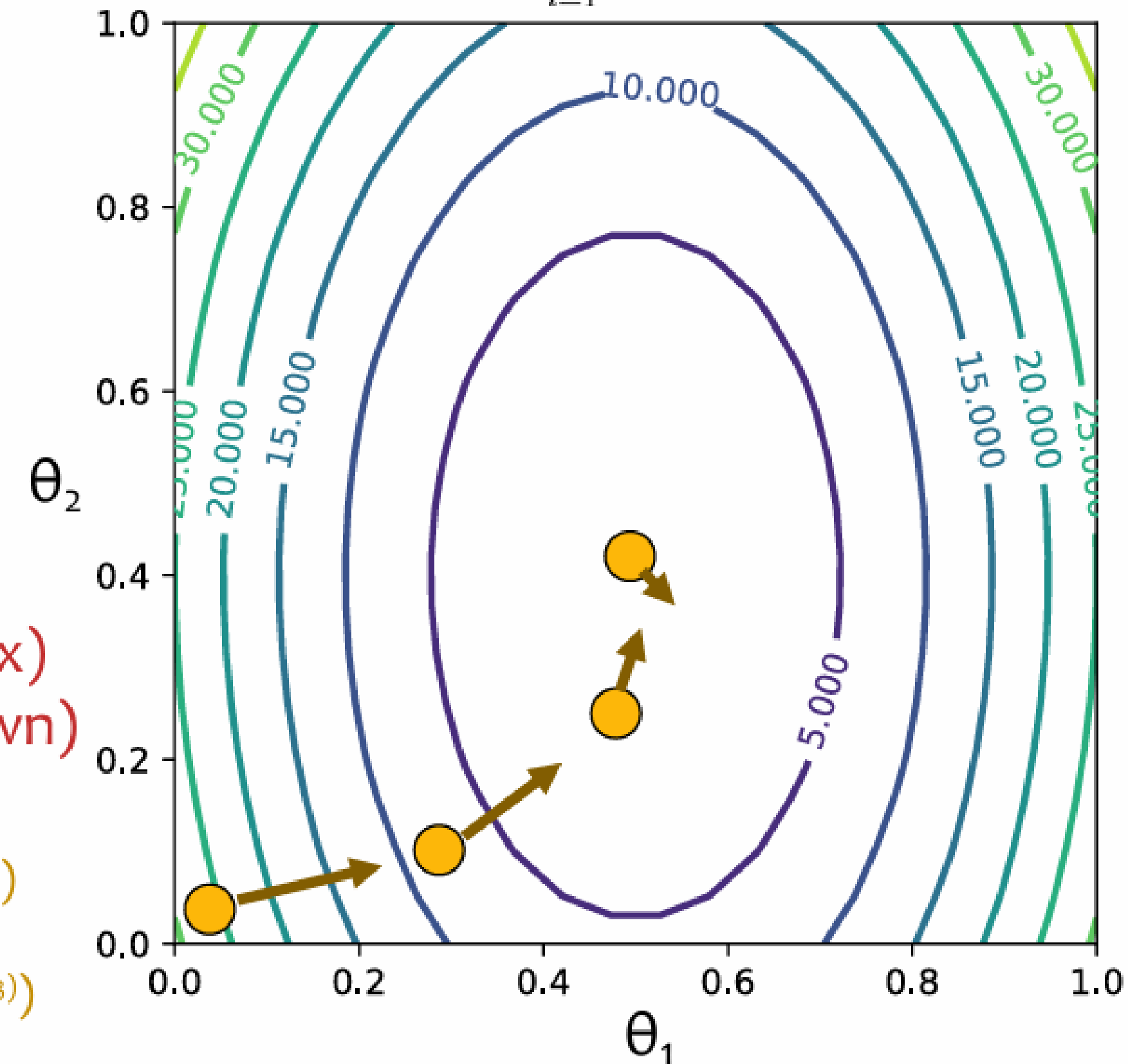


t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Linear Regression by Gradient Desc.



$$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N} \sum_{i=1}^N \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)} \right)^2$$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

a. Forward pass

$$\hat{y} = f_{\text{MLP}}(x_i; \theta)$$

b. Compute Loss

$$\mathcal{L}_i$$

2. Update

a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter partial derivatives

b. Gradient update

$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

vector of parameter update equations

THE UNIVERSITY OF CHICAGO

THE UNIVERSITY OF CHICAGO
CHICAGO, ILLINOIS 60637
TEL: 773-936-3000
WWW.CHICAGO.EDU

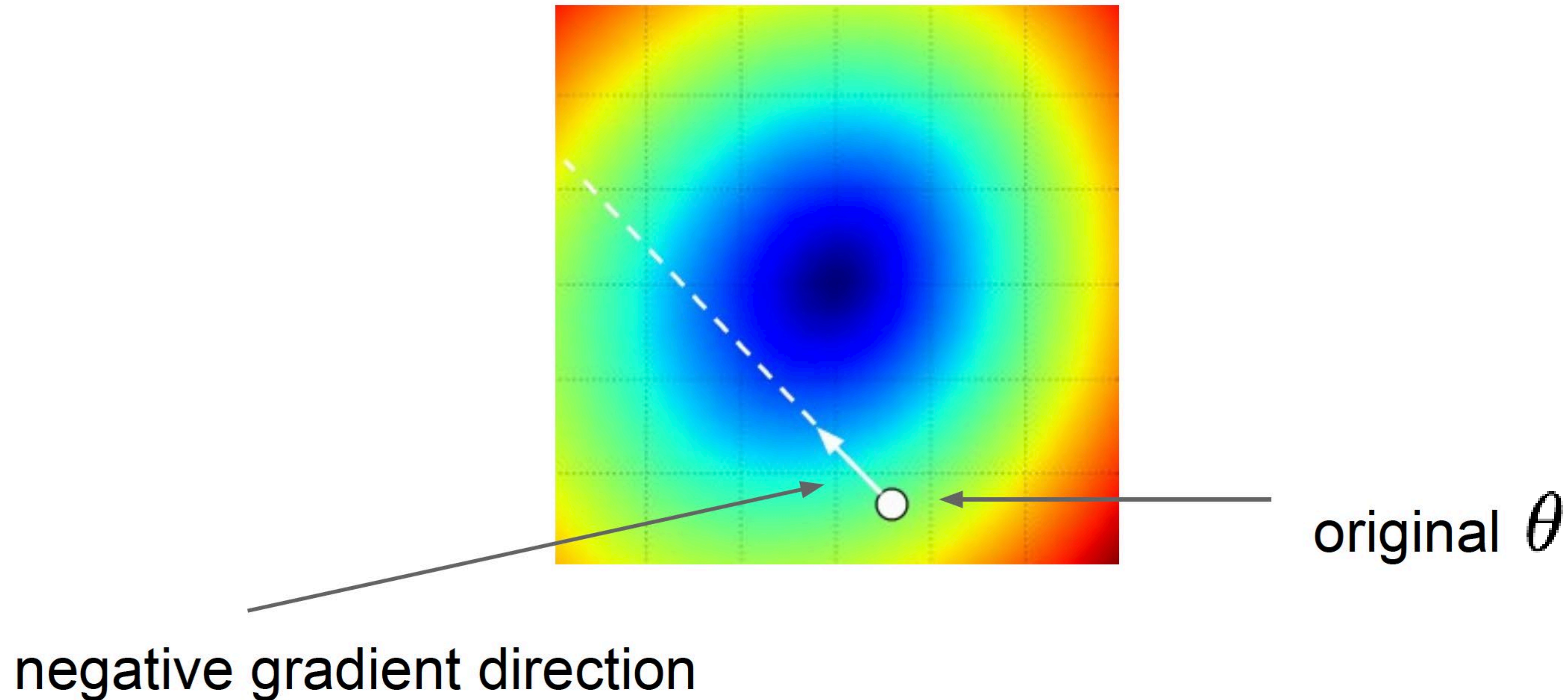


Python Step-by-Step Example

- <https://colab.research.google.com/drive/17dK6cynECzk2ObyCqDk5gKcUyN1kMjSR?usp=sharing>



Learning rates



$$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$$

Step size: learning rate

Too big: will miss the minimum

Too small: slow convergence

Objective Functions for NNs

1. Quadratic Loss:

- the same objective as Linear Regression
- i.e. mean squared error

$$J = \ell_Q(y, y^{(i)}) = \frac{1}{2}(y - y^{(i)})^2$$

$$\frac{dJ}{dy} = y - y^{(i)}$$

2. Binary Cross-Entropy:

- the same objective as Binary Logistic Regression
- i.e. negative log likelihood
- This requires our output y to be a probability in $[0,1]$

$$J = \ell_{CE}(y, y^{(i)}) = -(y^{(i)} \log(y) + (1 - y^{(i)}) \log(1 - y))$$
$$\frac{dJ}{dy} = - \left(y^{(i)} \frac{1}{y} + (1 - y^{(i)}) \frac{1}{y - 1} \right)$$

Objective Functions for NNs

Cross-entropy vs. Quadratic loss

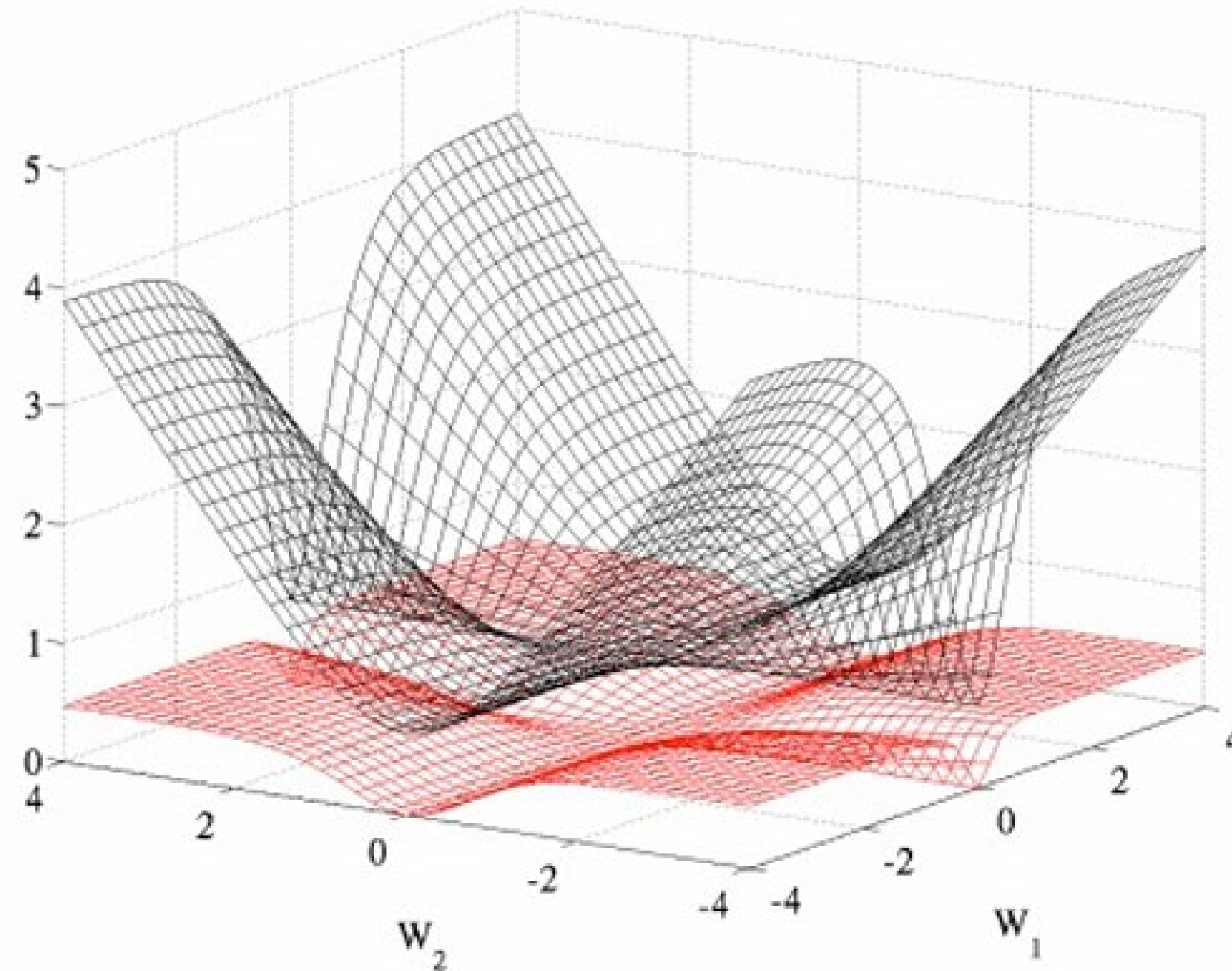
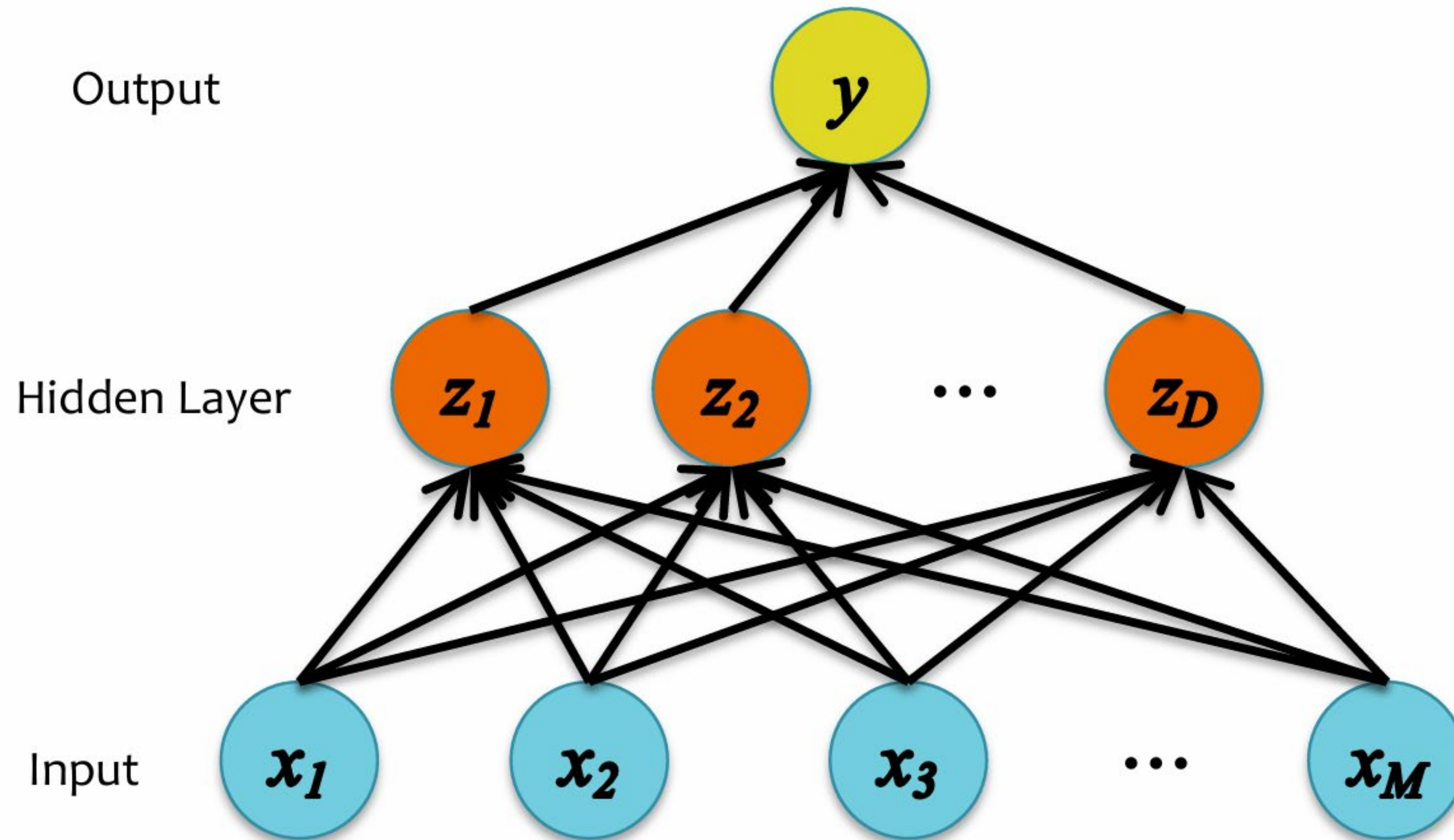


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.

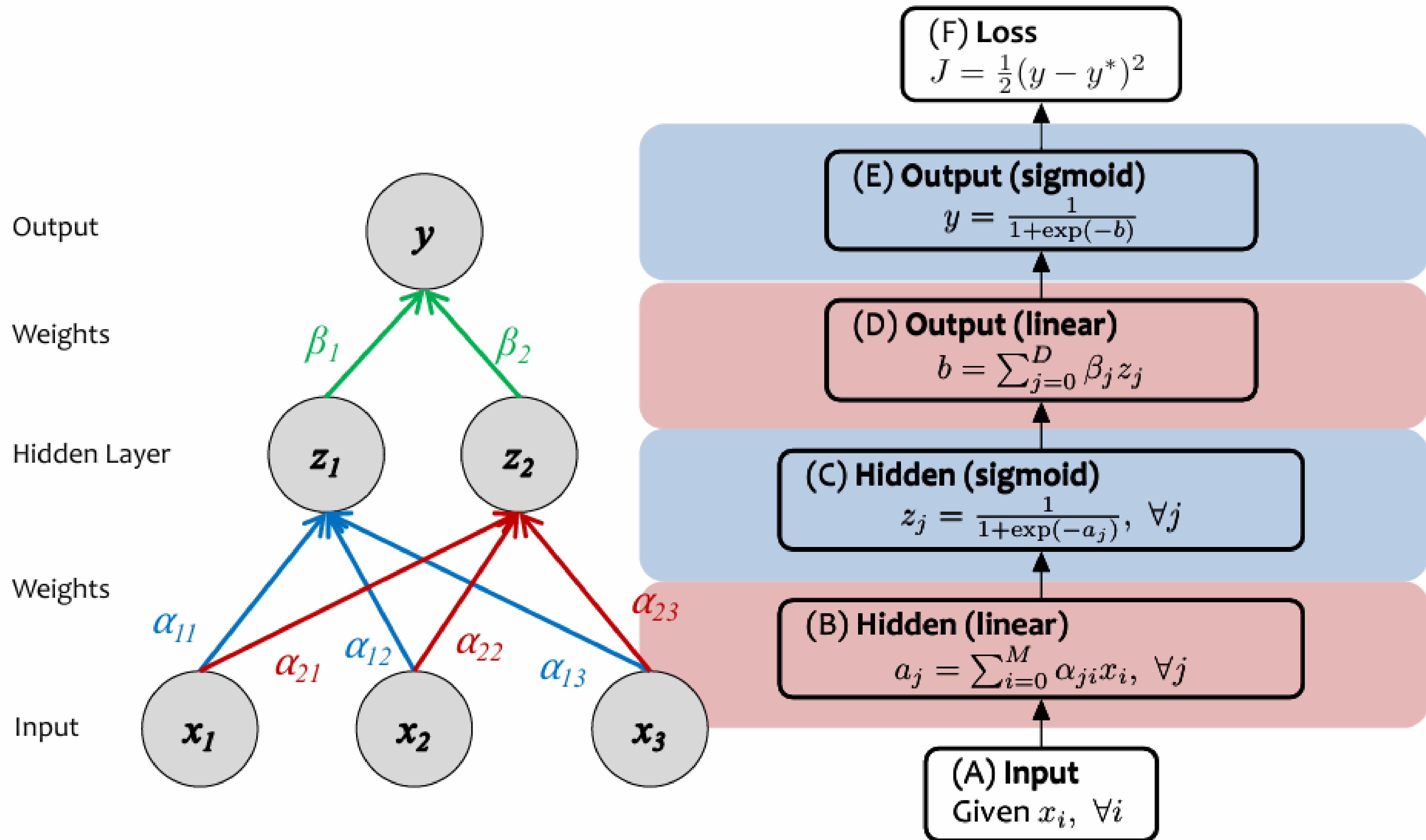
Single Output Neural Network

Let's write the equation



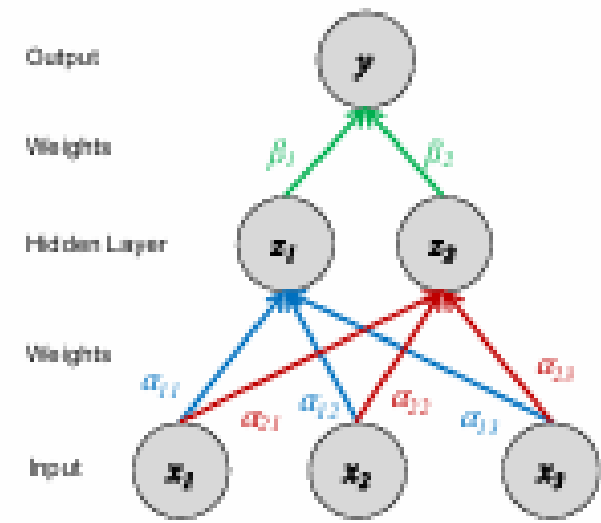
Training

Forward-Computation



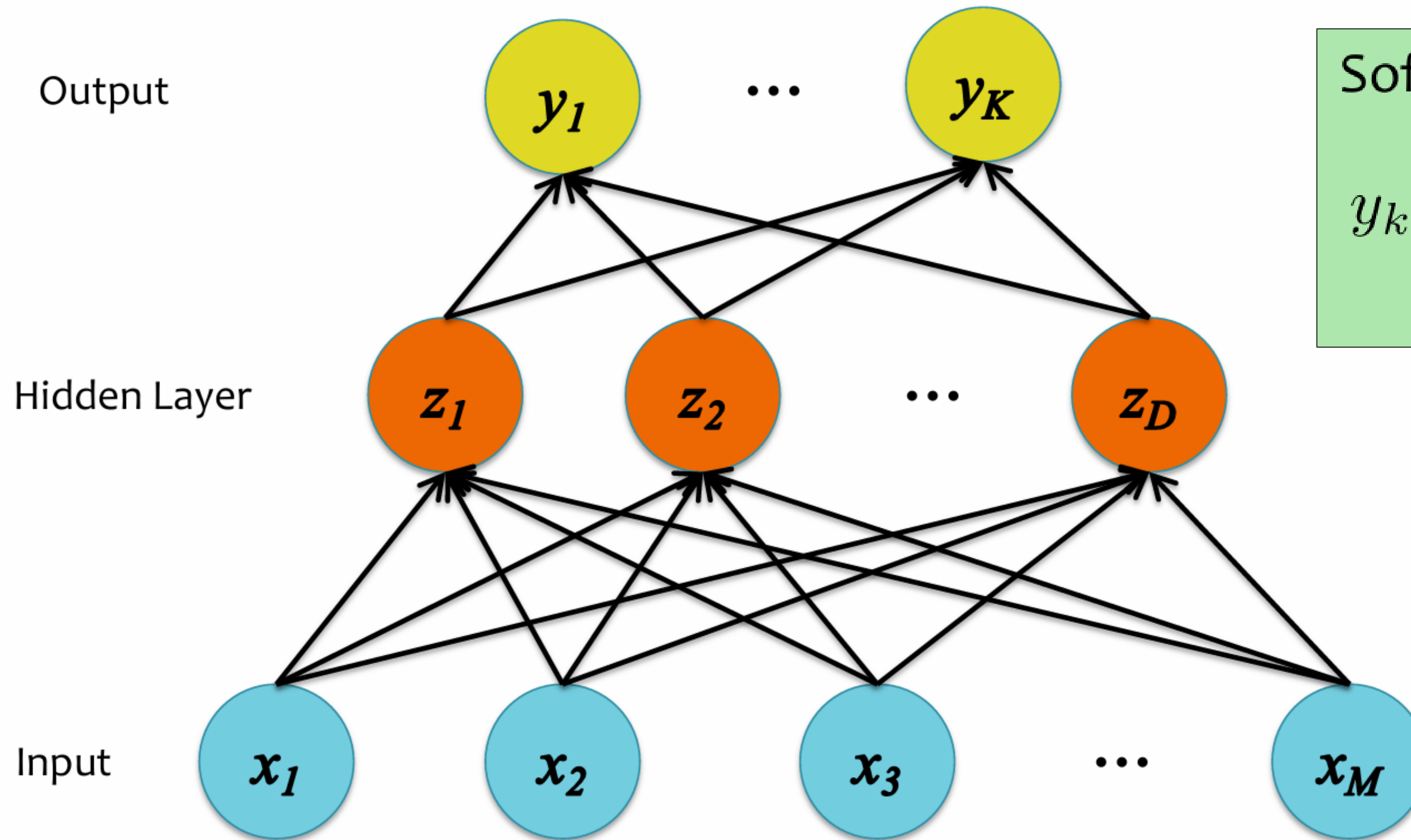
Case 2: Neural Network

Backpropagation



	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$g_y = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$g_b = g_y \frac{\partial y}{\partial b}, \frac{\partial y}{\partial b} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^D \beta_j z_j$	$g_{\beta_j} = g_b \frac{\partial b}{\partial \beta_j}, \frac{\partial b}{\partial \beta_j} = z_j$ $g_{z_j} = g_b \frac{\partial b}{\partial z_j}, \frac{\partial b}{\partial z_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$g_{a_j} = g_{z_j} \frac{\partial z_j}{\partial a_j}, \frac{\partial z_j}{\partial a_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^M \alpha_{ji} x_i$	$g_{\alpha_{ji}} = g_{a_j} \frac{\partial a_j}{\partial \alpha_{ji}}, \frac{\partial a_j}{\partial \alpha_{ji}} = x_i$ $g_{x_i} = \sum_{j=0}^D g_{a_j} \frac{\partial a_j}{\partial x_i}, \frac{\partial a_j}{\partial x_i} = \alpha_{ji}$

Multi-class Output



Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^K \exp(b_l)}$$

Backpropagation

