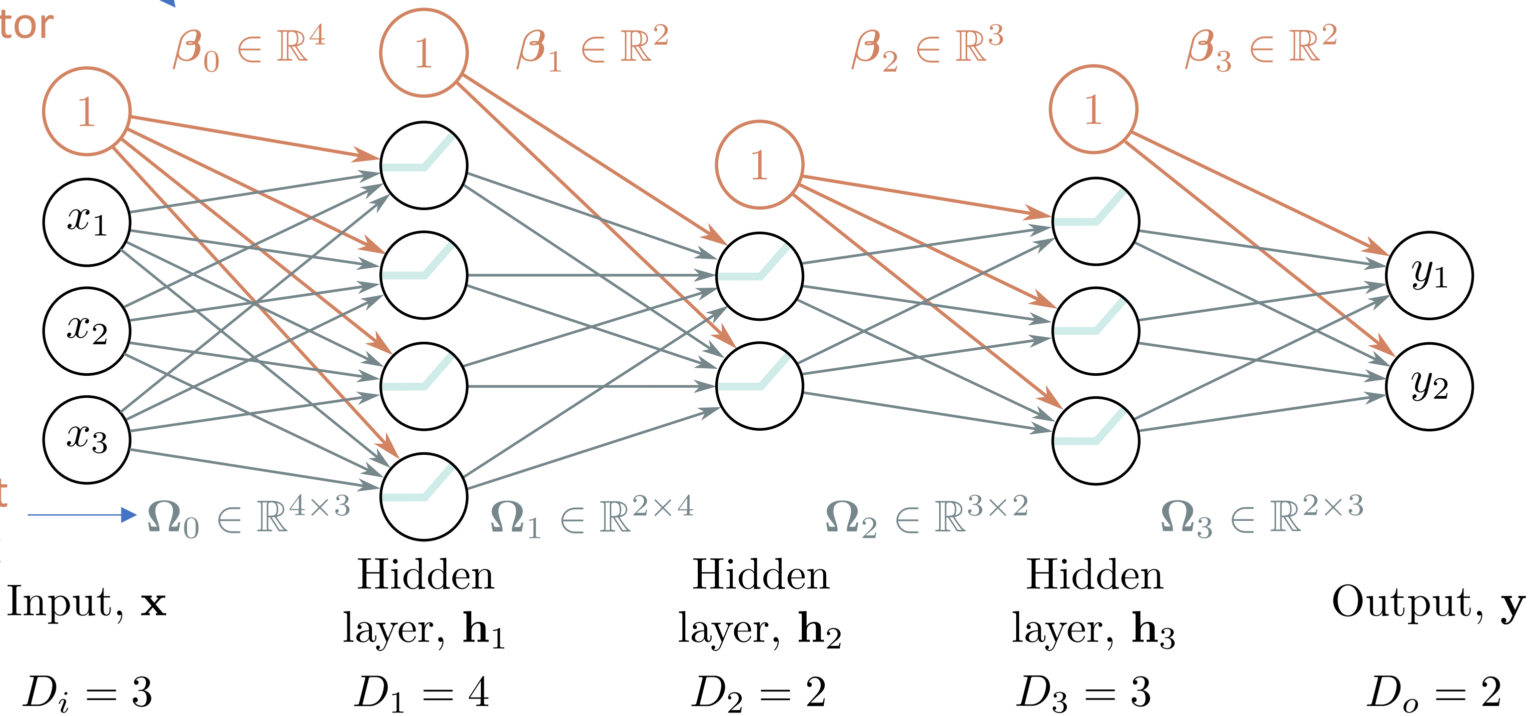
EECS 230 Deep Learning
Lecture 4: Back Propagation

Some slides from O. Veksler, Y. Boykov, A. Ng, Y. LeCun, G. Hinton, A. Ranzato, R. Fergus

How to train neural networks?

- Training == learning weight and bias

Bias vector



Weight matrix

Example of Multi Layer Perceptron (MLP)

Optimization via Gradient Descent

Optimization of continuous differentiable functions

- How to minimize a function of a single variable

$$f(x) = (x-5)^2$$

- Take derivative and set it to 0

$$\frac{d}{dx} f(x) = 0$$

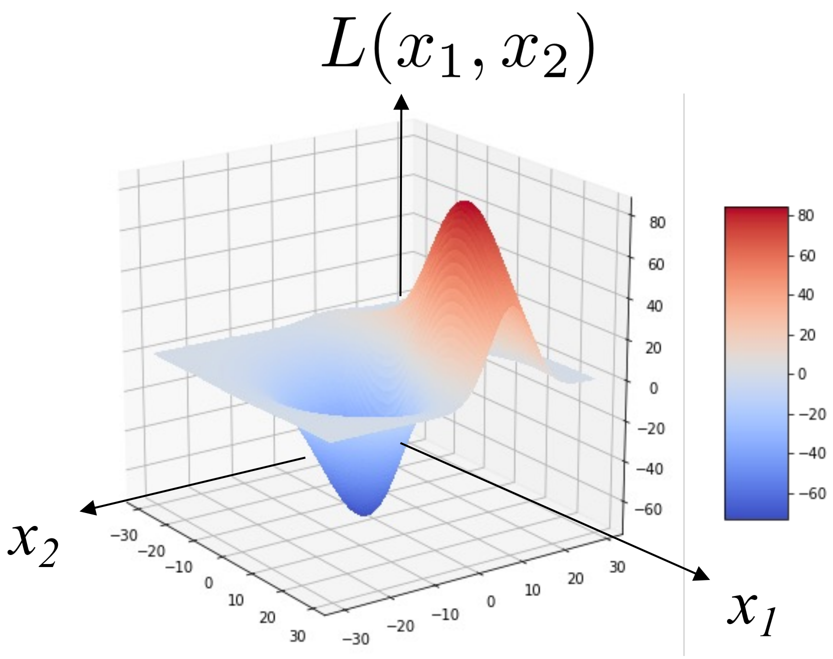
- May find a closed form solution

$$\frac{d}{dx} f(x) \;=\; 2(x-5) \;=\; 0 \qquad \Rightarrow \qquad x = 5$$

UCMERCED

Differentiation

3D plot



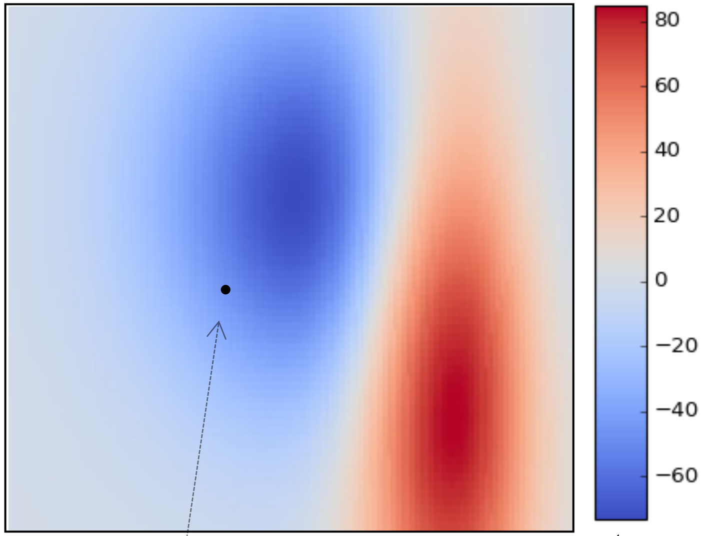
What is "**slope**" of $L(x_1,x_2)$ at a given point $\mathbf{x}=(x_1,x_2)$?

Differentiation

"heat-map" visualization of L

domain of $L(x_1,x_2)$ in R^2

x_2



x_1

range of $L(x_1,x_2)$

What is "**slope**" of $L(x_1,x_2)$ at a given point $\mathbf{x}=(x_1,x_2)$?

Differentiation

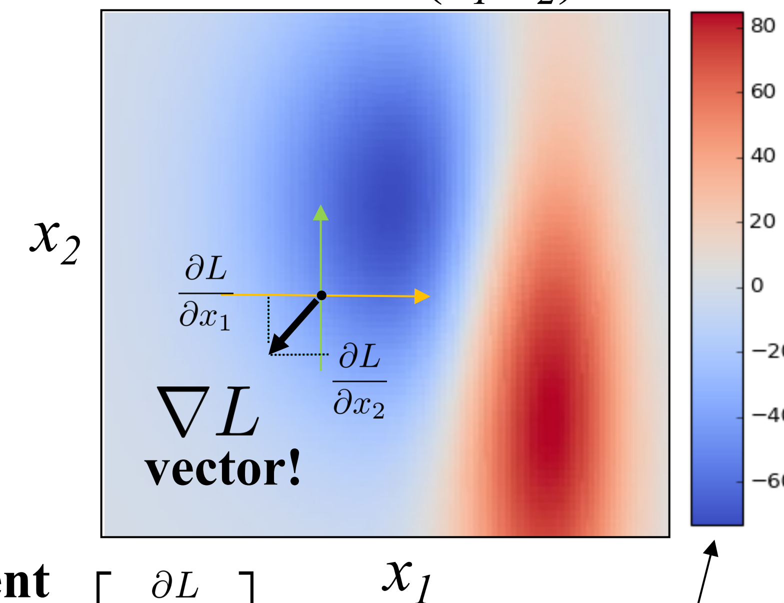
"heat-map" visualization of L

"partial" derivatives

$$\frac{\partial L}{\partial x_1} = \lim_{\epsilon \to 0} \left(\frac{L(x_1 + \epsilon, x_2) - L(x_1, x_2)}{\epsilon} \right)$$

$$\frac{\partial L}{\partial x_2} = \lim_{\epsilon \to 0} \left(\frac{L(x_1, x_2 + \epsilon) - L(x_1, x_2)}{\epsilon} \right)$$

domain of $L(x_1,x_2)$ in R^2



gradient

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix}$$

range of $L(x_1,x_2)$

direction of the steepest ascent at point x=(x_1,x_2)

Differentiation

The most common optimization method for continuous differentiable (multi-variate) functions:

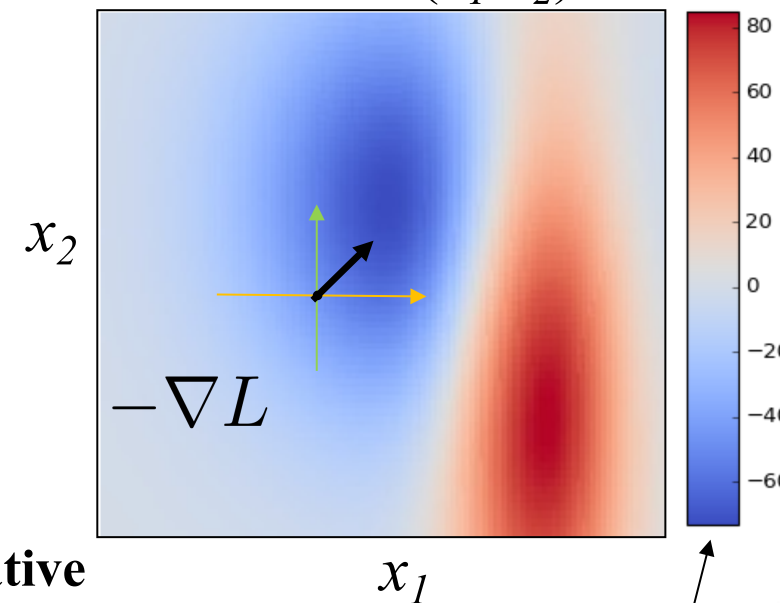
gradient descent

take a step $\mathbf{x}' = \mathbf{x} - \alpha \nabla L$

towards lower values of the function

"heat-map" visualization of L

domain of $L(x_1,x_2)$ in R^2



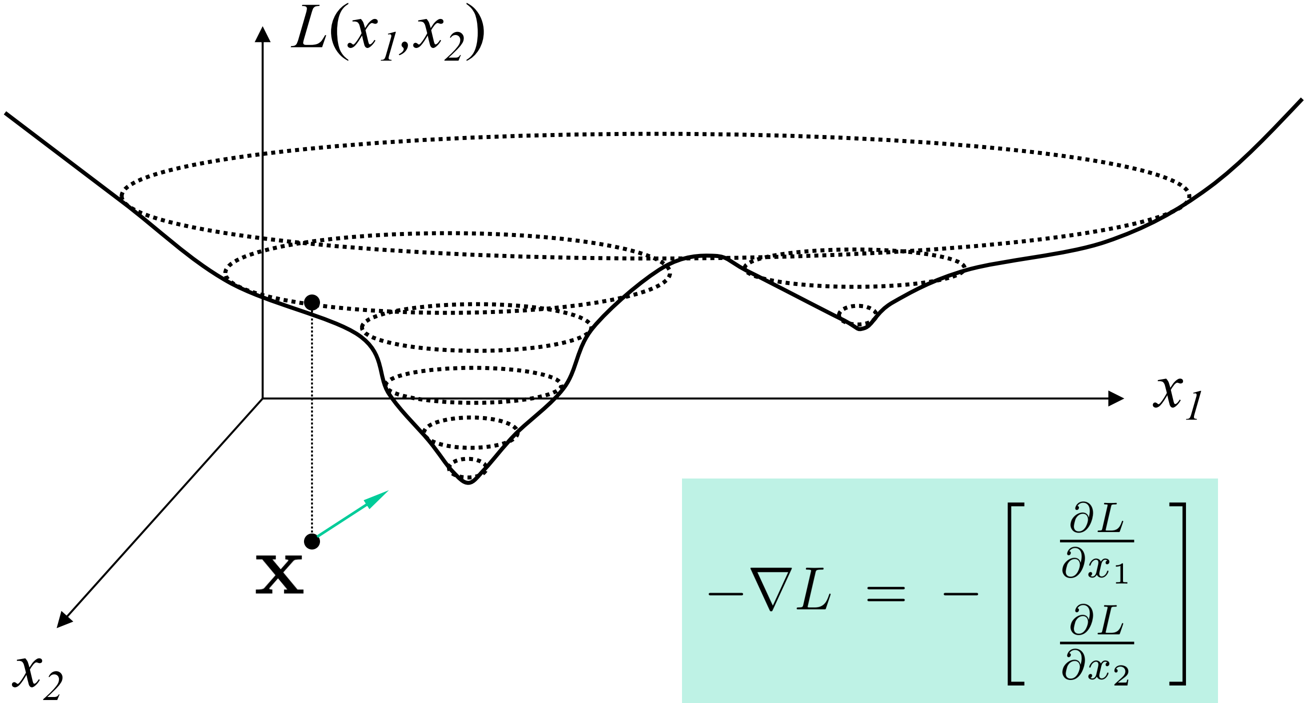
negative gradient

range of $L(x_1,x_2)$

direction of the steepest descent at point x=(x_1,x_2)

Gradient Descent

Example: for a function of two variables



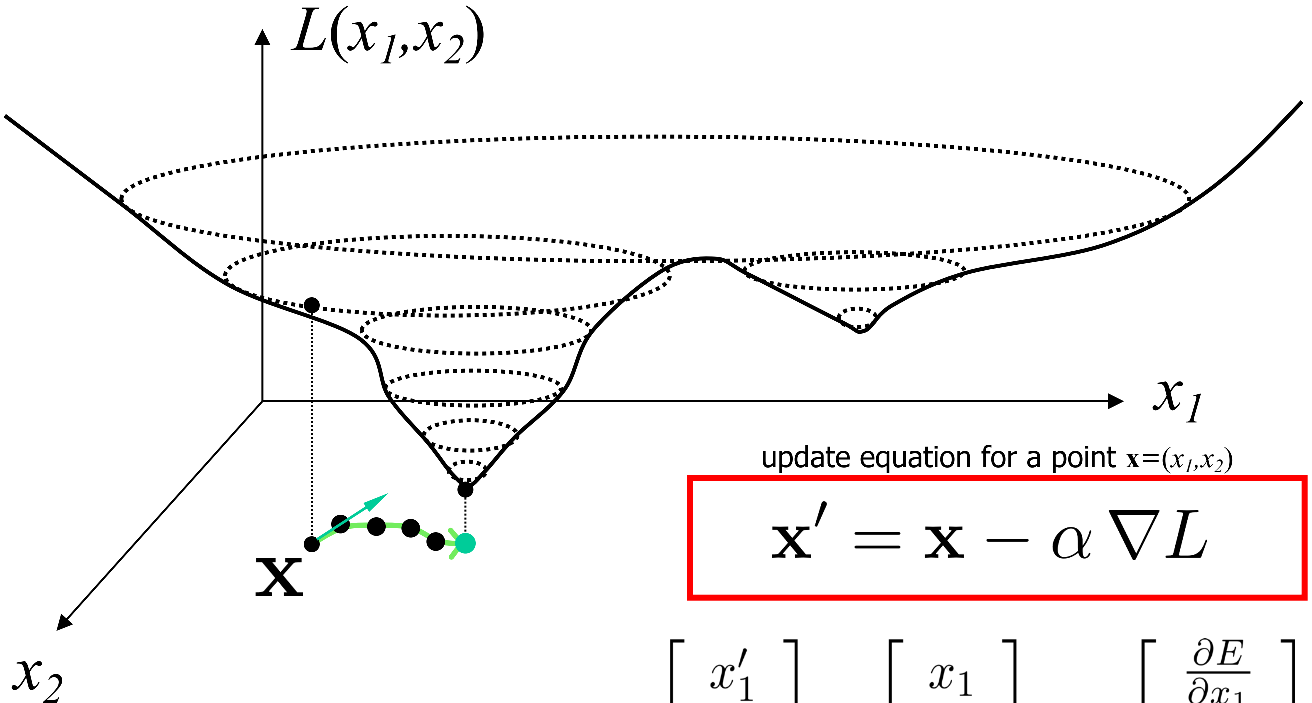
$$-\nabla L = -\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix}$$

- direction of (negative) gradient at point $\mathbf{x}=(x_1,x_2)$ is direction of the steepest descent towards lower values of function L

- magnitude of gradient at $\mathbf{x}=(x_1,x_2)$ gives the value of the slope

Gradient Descent

Example: for a function of two variables



update equation for a point $\mathbf{x}=(x_1,x_2)$

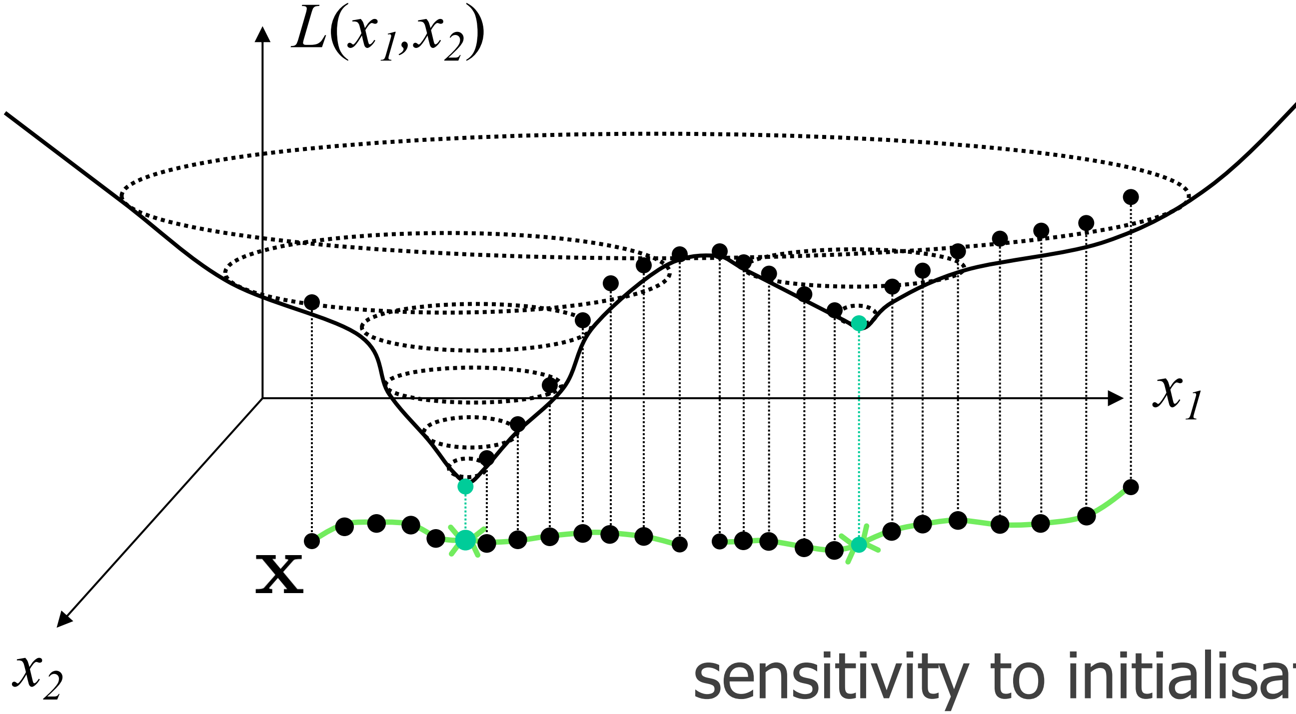
$$\boxed{\mathbf{x}' = \mathbf{x} - \alpha \nabla L}$$

$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \end{bmatrix}$$

Stop at a local minima where $\nabla L = \vec{0}$

Gradient Descent

Example: for a function of two variables

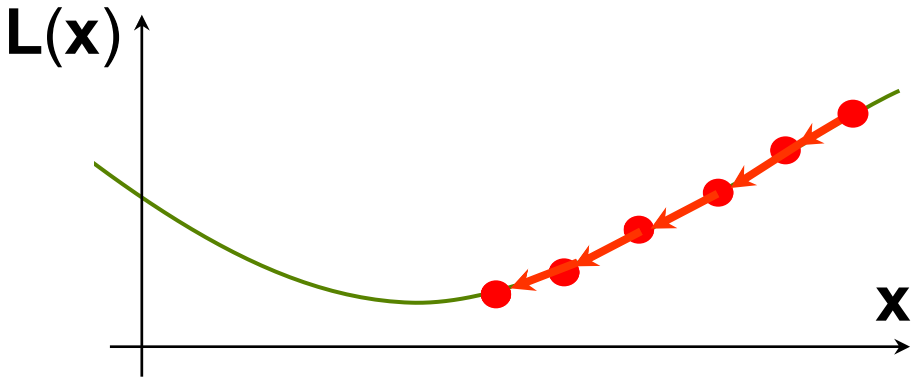


sensitivity to initialisation !!

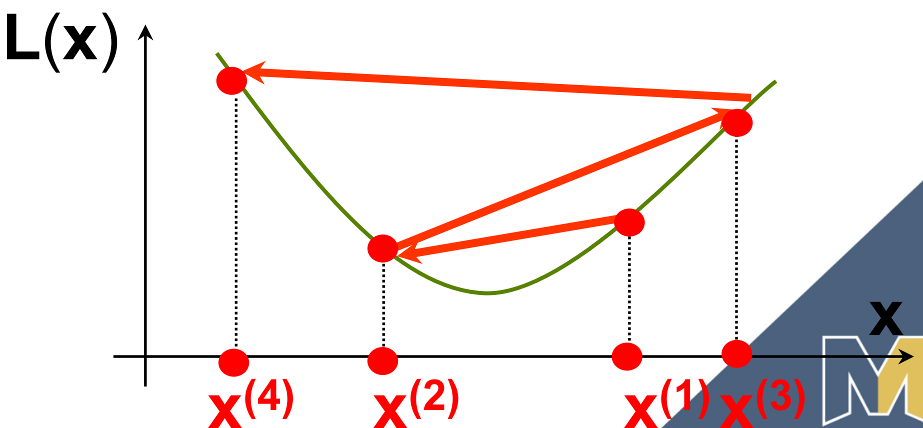
How to Set Learning Rate α?

$$\boxed{\mathbf{x}' = \mathbf{x} - \alpha \nabla L}$$

- If α too small, too many iterations to converge



- If α too large, may overshoot the local minimum and possibly never even converge



UCMERCED

Variable Learning Rate

If desired, can change learning rate α at each iteration

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose α, ε

while $\alpha\|\nabla \mathbf{L}(\mathbf{x}^{(k)})\| > \varepsilon$

$\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha\, \nabla \mathbf{L}(\mathbf{x}^{(k)})$

$\quad \mathbf{k} = \mathbf{k} + 1$

fixed α
gradient descent

$\rightarrow$

$\mathbf{k} = 1$

$\mathbf{x}^{(1)}$ = any initial guess

choose ε

while $\alpha\|\nabla \mathbf{L}(\mathbf{x}^{(k)})\| > \varepsilon$

$\quad$ choose $\alpha^{(k)}$

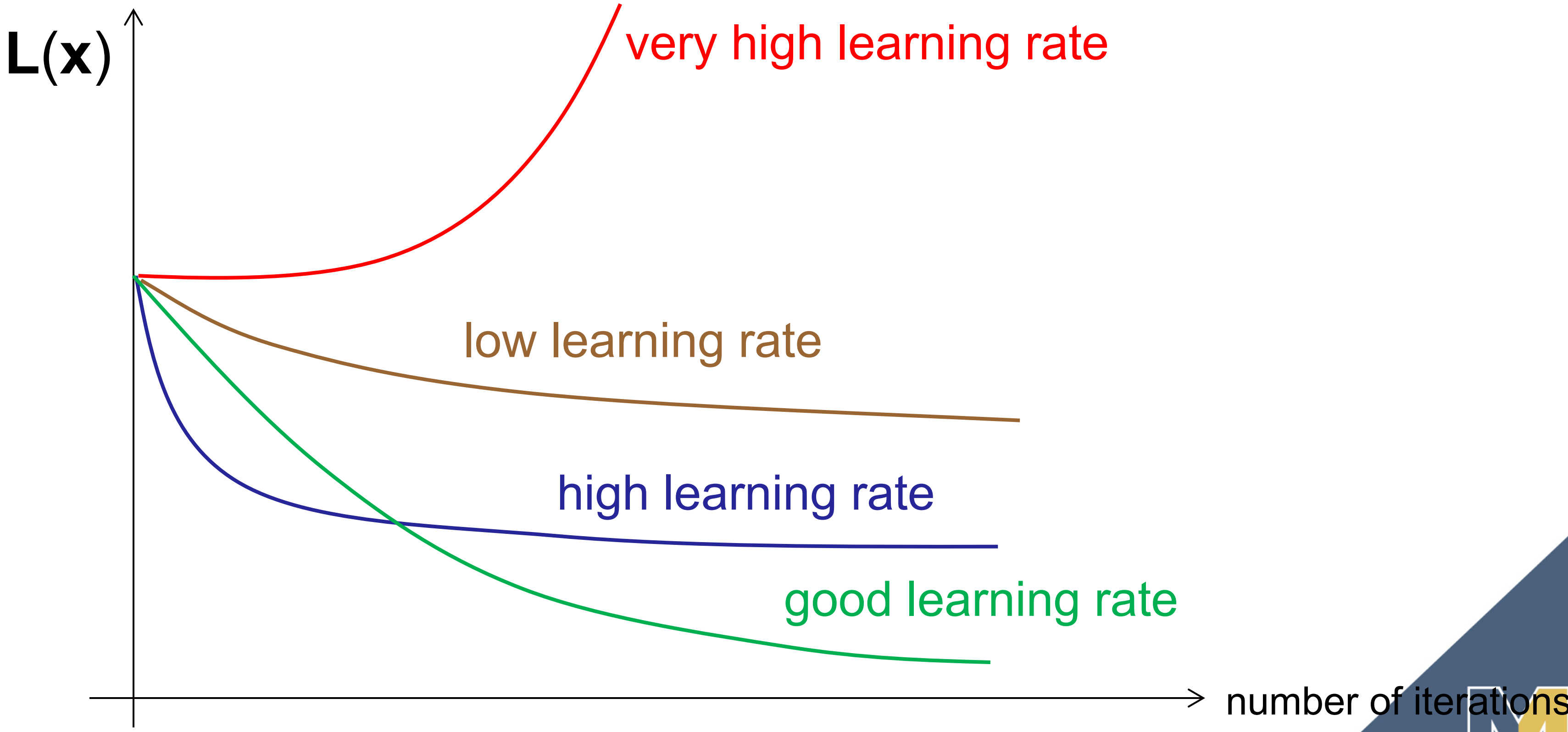
$\quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha^{(k)}\, \nabla \mathbf{L}(\mathbf{x}^{(k)})$

$\quad \mathbf{k} = \mathbf{k} + 1$

variable α
gradient descent

Learning Rate

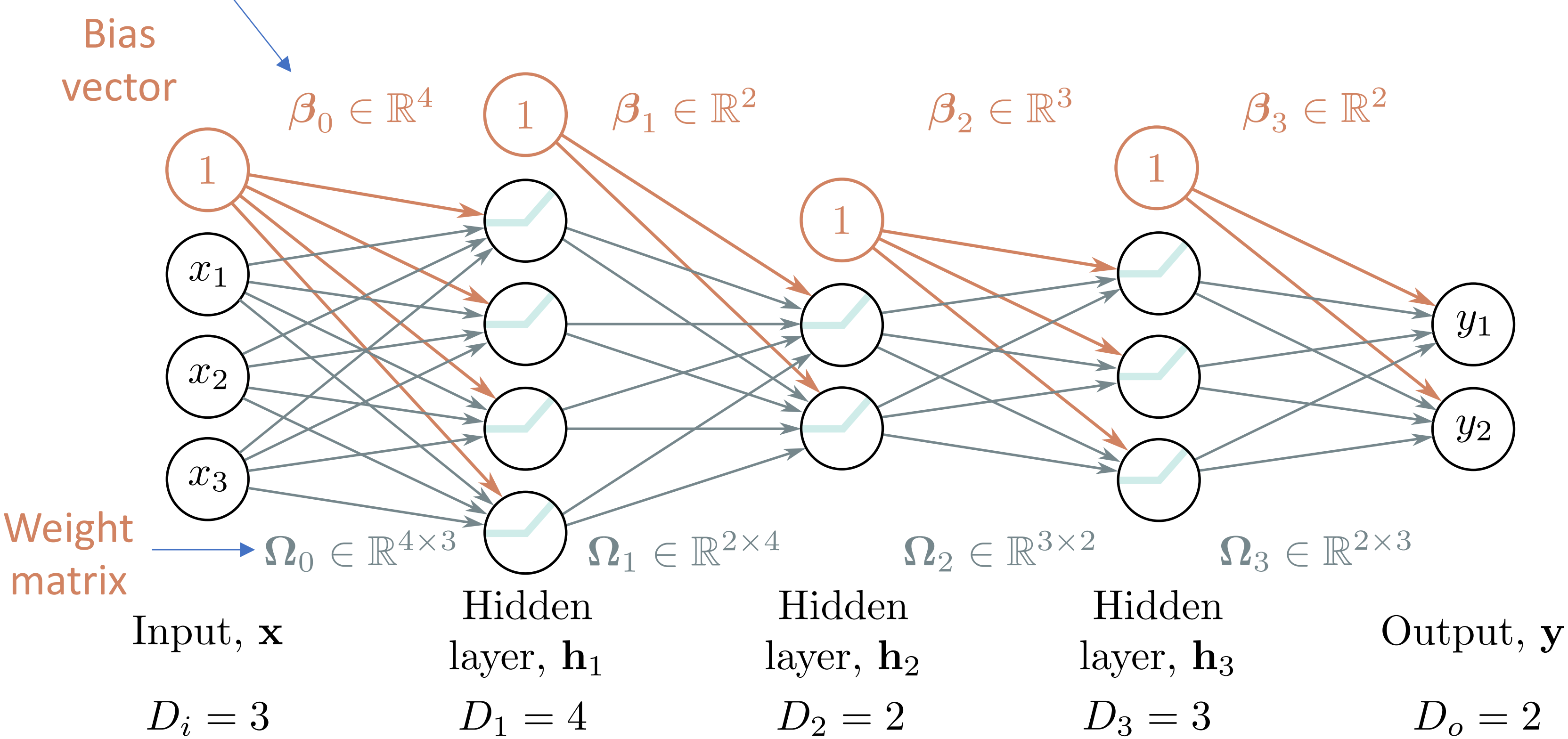
- Monitor learning rate by looking at how fast the objective function decreases



Derivative and Back Propagation

How to take derivatives w.r.t. weights?

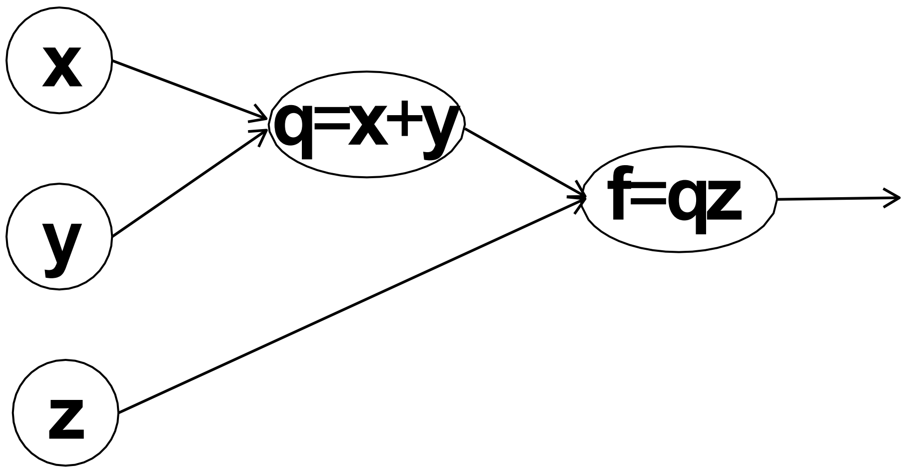
- Training == learning weight and bias



Example of Multi Layer Perceptron (MLP)

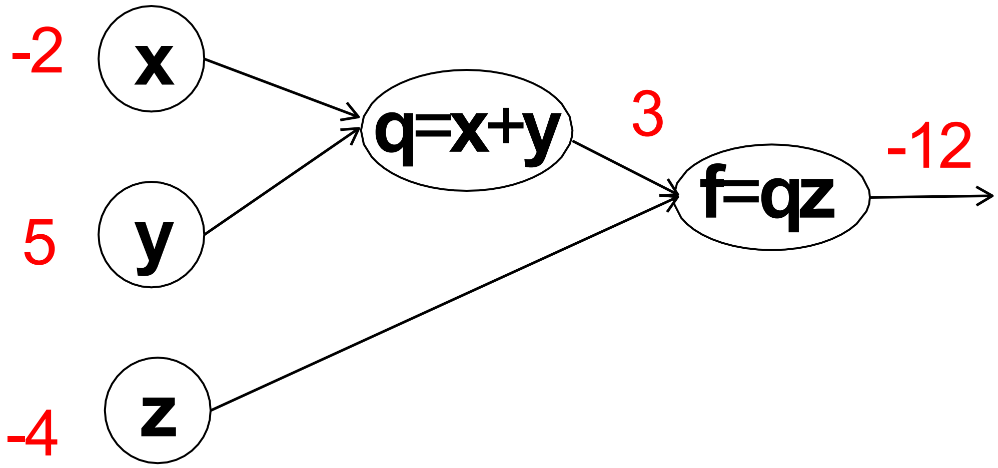
Computing Derivatives: Small Example

- Small network **f**(**x**,**y**,**z**) = (**x**+**y**)**z**
- Rewrite using
  - **q** = **x** + **y**
- **f**(**x**,**y**,**z**) = **qz**
- each node does one operation



Computing Derivatives: Small Example

- Small network $\mathbf{f(x,y,z) = (x+y)z}$
- Rewrite using
  - $\mathbf{q = x + y}$
  - $\mathbf{f(x,y,z) = qz}$
- Example of computing $\mathbf{f(-2,5,-4)}$

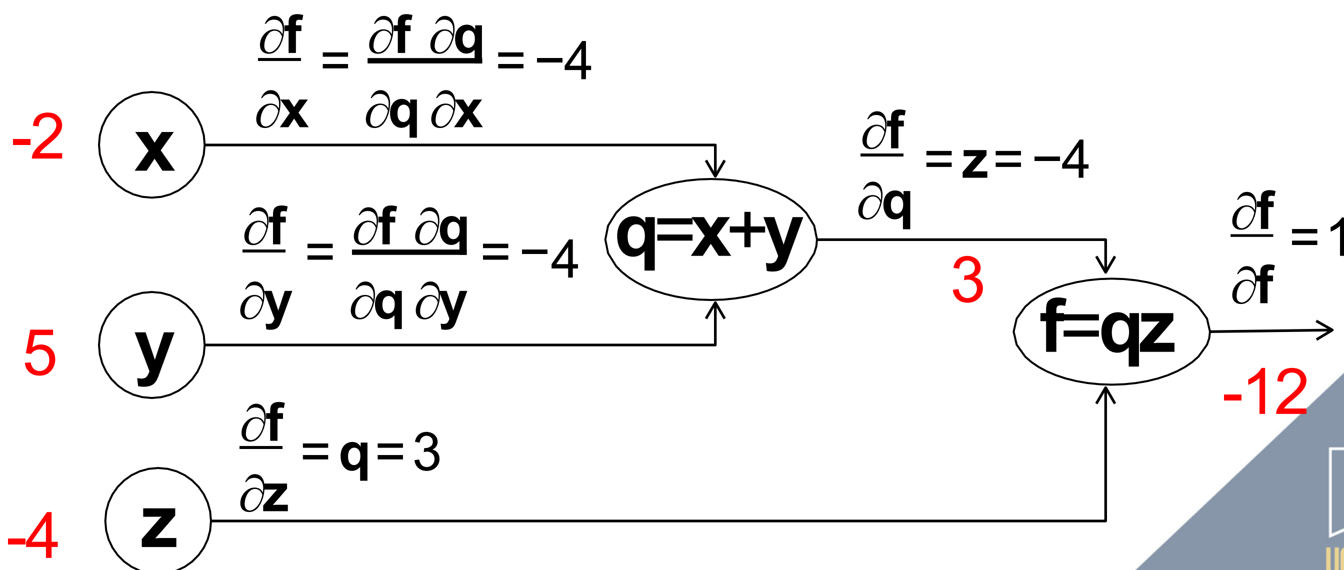


Computing Derivatives: Small Example

- Small network $\mathbf{f(x,y,z) = (x+y)z}$
- Rewrite using $\mathbf{q = x + y} \Rightarrow \mathbf{f(x,y,z) = qz}$
- Want $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}, \frac{\partial \mathbf{f}}{\partial \mathbf{y}}, \frac{\partial \mathbf{f}}{\partial \mathbf{z}}$
- Compute $\frac{\partial \mathbf{f}}{\partial}$ from the end backwards
  - for each edge, with respect to the main variable at edge origin
  - using chain rule with respect to the variable at edge end, if needed

chain rule for $\mathbf{f(y(x))}$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}}$$



Computing Derivatives: Chain of Chain Rule

- Compute $\frac{\partial d}{\partial}$ from the end backwards
  - for each edge, with respect to the main variable at edge origin
  - using chain rule with respect to the variable at edge end, if needed

direction of computation

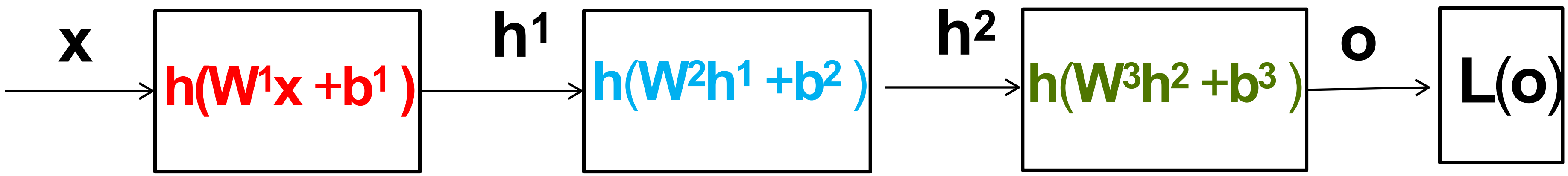
$$\xrightarrow{a} \boxed{b = h(a)} \longrightarrow \boxed{c = h(b)} \longrightarrow \boxed{d = h(c)} \xrightarrow{d}$$

$$\frac{\partial d}{\partial a} = \frac{\partial d}{\partial b}\frac{\partial b}{\partial a} \qquad \frac{\partial d}{\partial b} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial b} \qquad \frac{\partial d}{\partial c}$$

prev local   prev local   local

$$\boxed{\text{example: if } h(c) = c^2, \text{ then } \frac{\partial d}{\partial c} = \frac{\partial h}{\partial c} = 2c}$$

Computing Derivatives Backwards

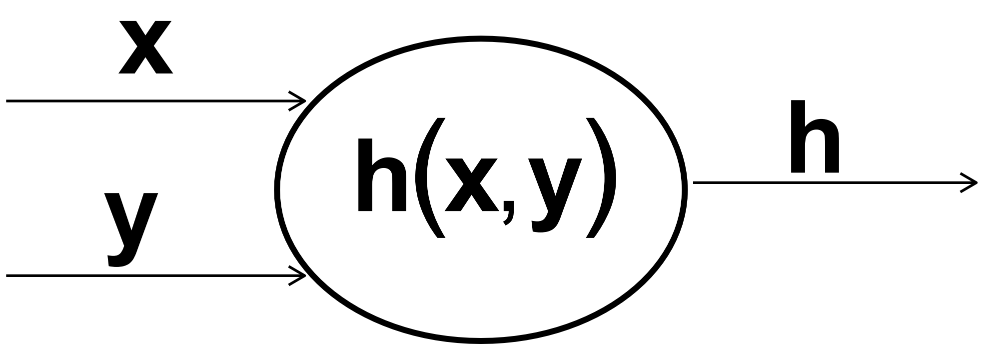


- Have loss function $\mathbf{L}(\mathbf{o})$
- Need derivatives for all $\frac{\partial \mathbf{L}}{\partial \mathbf{w}}, \frac{\partial \mathbf{L}}{\partial \mathbf{b}}$
- Will compute derivatives from end to front, backwards
- On the way will also compute intermediate derivatives $\frac{\partial \mathbf{L}}{\partial \mathbf{h}}$

UCMERCED

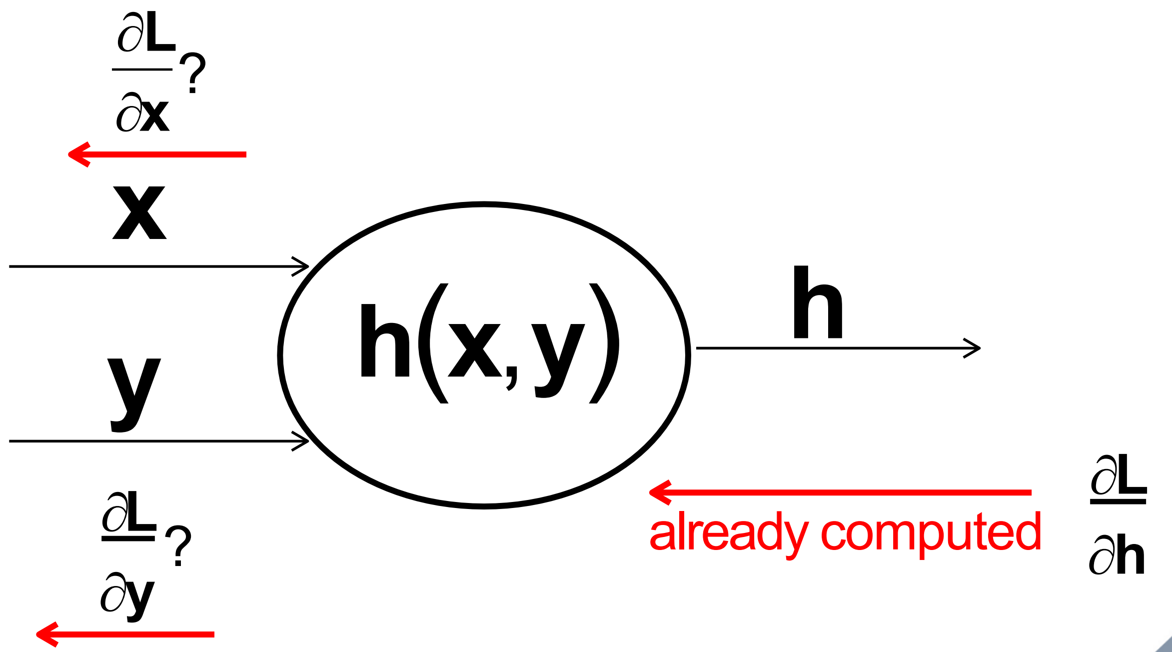
Computing Derivatives: Look at One Node

- Simplified view at a network node
  - inputs **x**,**y** come in
  - node computes some function **h**(**x**,**y**)



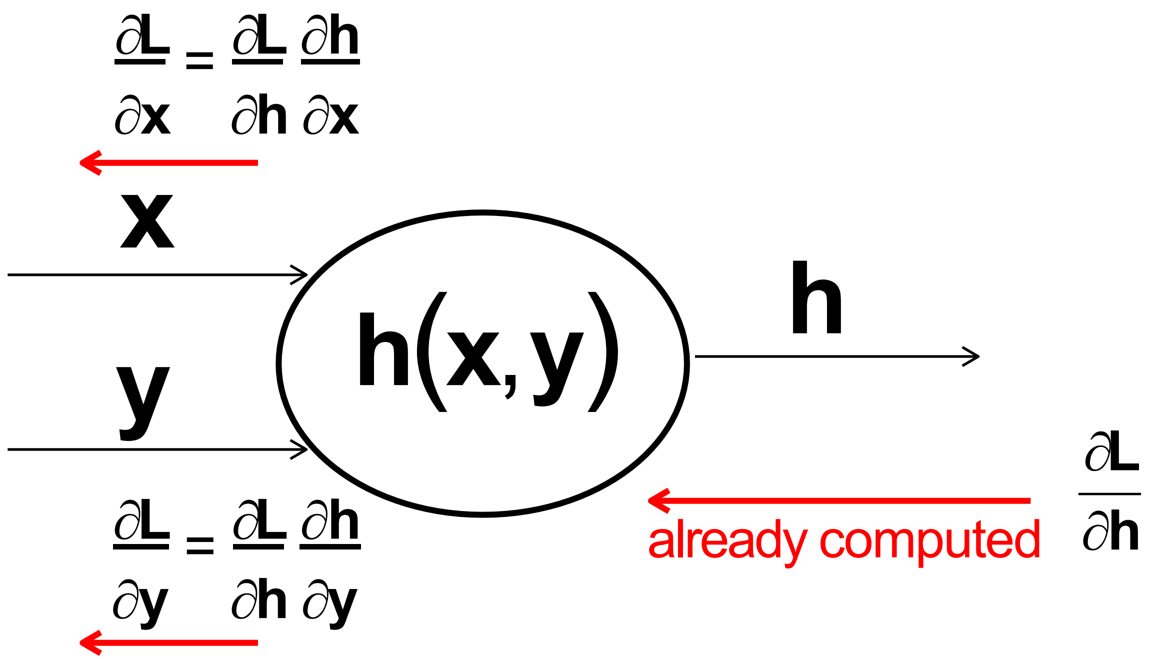
Computing Derivatives: Look at One Node

- At each network node
  - inputs **x**,**y** come in
  - nodes computes activation function **h**(**x**,**y**)
- Have loss function **L**(·)



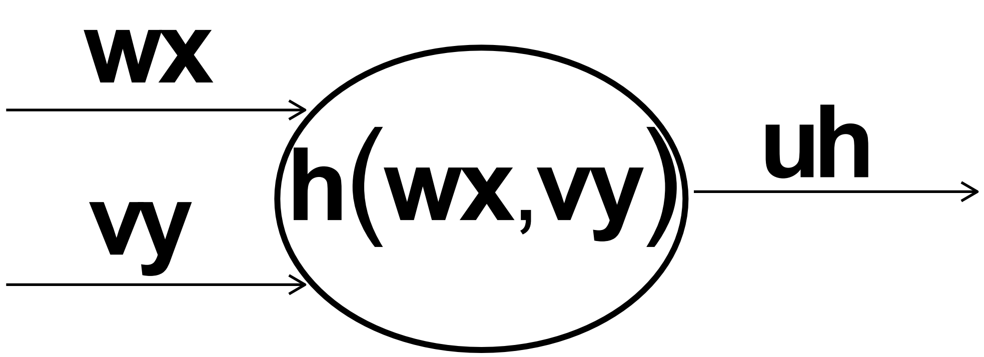
Computing Derivatives: Look at One Node

- Need $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial y}$
- Easy to compute local node derivatives $\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}$

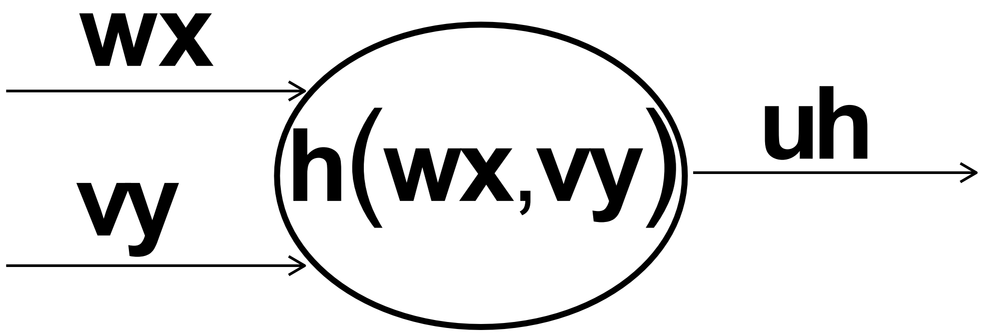


Computing Derivatives: Look at One Node

- More complete view at a network node
  - inputs **x**,**y** come in, get multiplied by weight **w** and **v**
  - node computes function **h**(**wx**,**vy**)
  - node output **h** gets multiplied by **u**

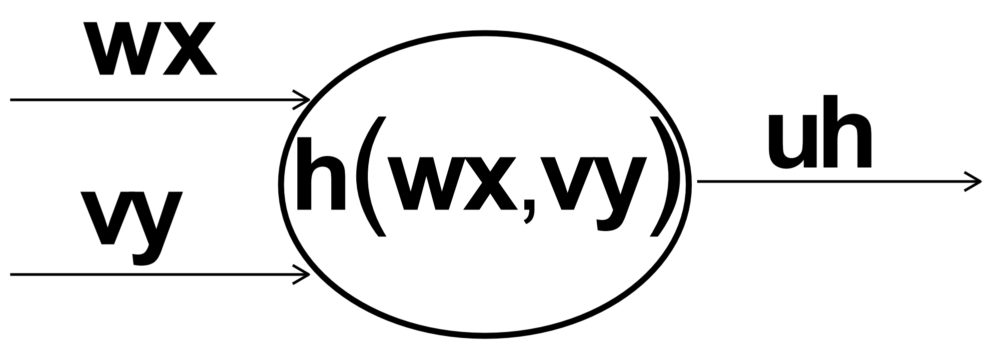


Computing Derivatives: Look at One Node



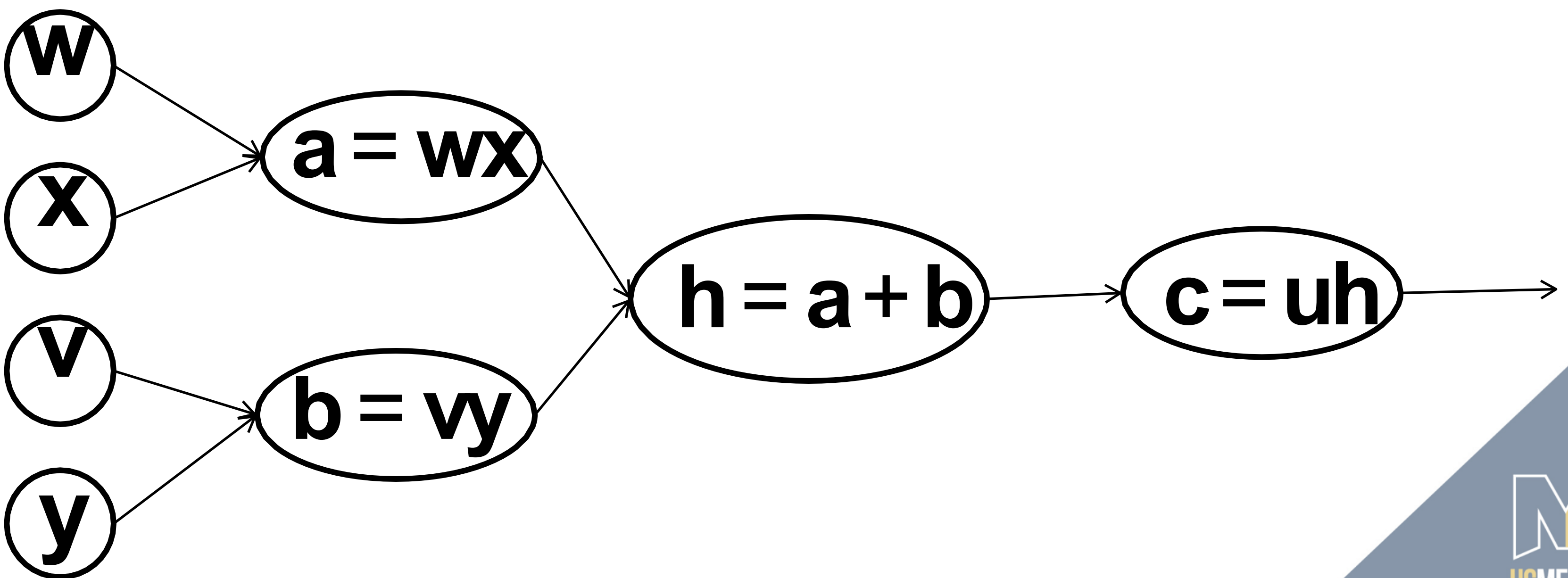
- To be concrete, let **h**(**i**,**j**) = **i** + **j**

Computing Derivatives: Look at One Node

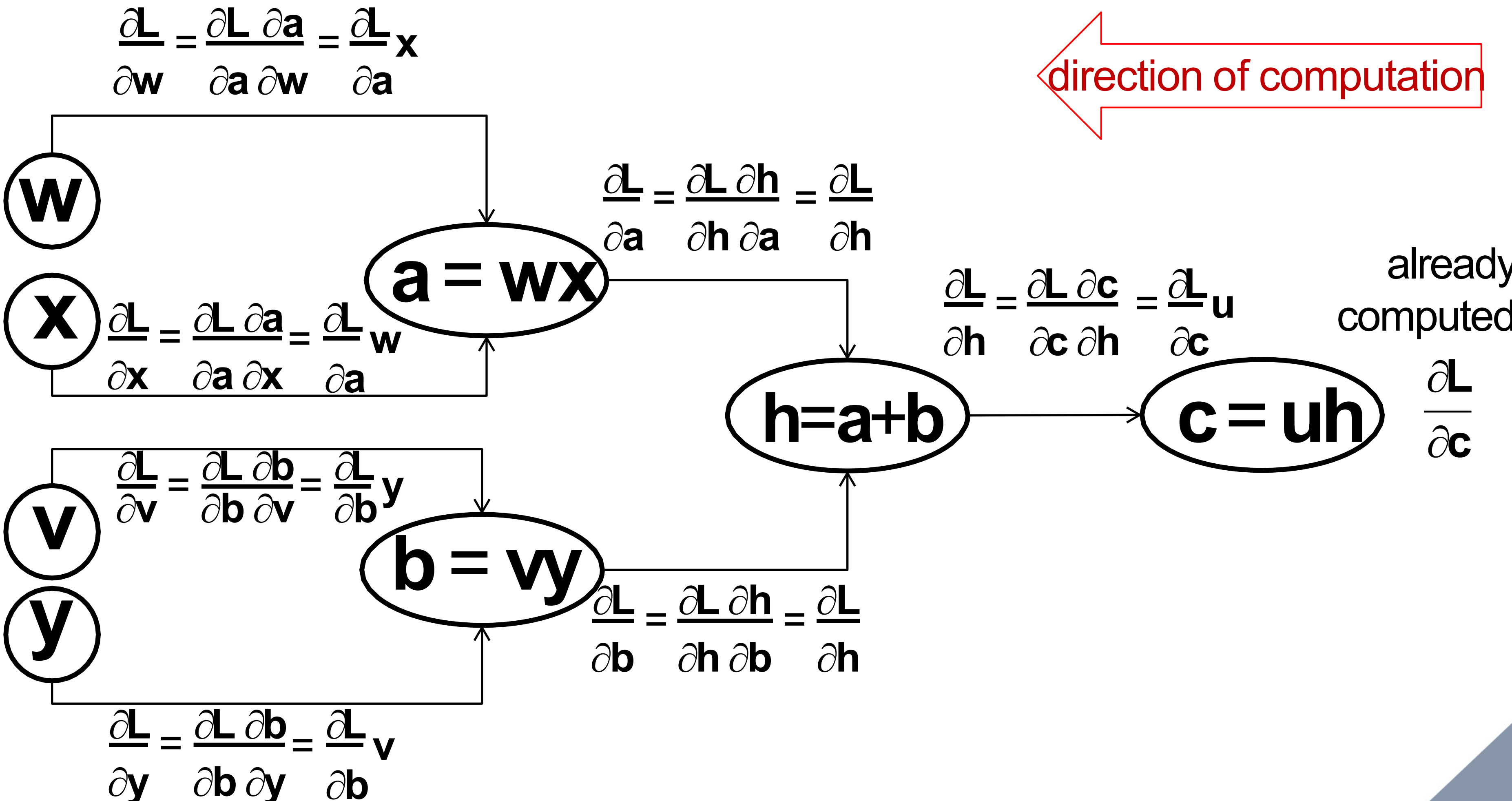


- h(i,j) = i + j

- Break into more computational nodes
  - all computation happens inside nodes, not on edges

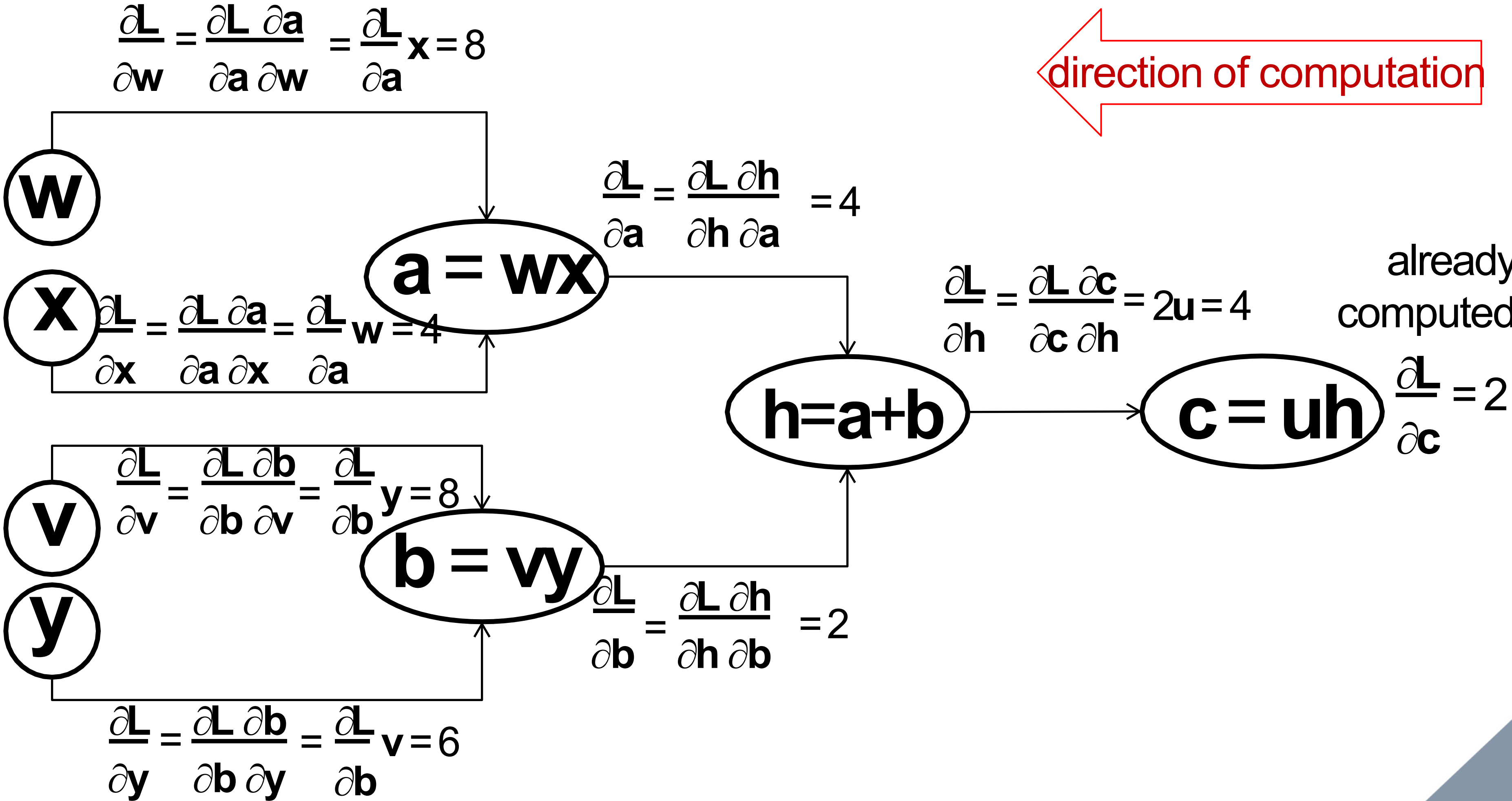


Computing Derivatives: Look at One Node



- Some of these partial derivatives are intermediate
  - their values will not be used for gradient descent

Computing Derivatives: Look at One Node



- Example when **w** = 1, **x** = 2, **v** = 3, **y** = 4, u = 2, $\frac{\partial L}{\partial c} = 2$

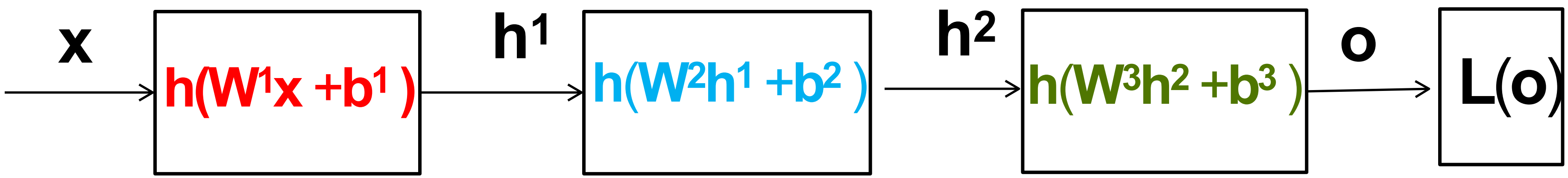
Computing Derivatives: Staging Computation

- Each node is responsible for one function
- To compute exp(1/**x**)

$$\xrightarrow{\mathbf{x}} \left(\mathbf{h} = 1/\mathbf{x} \right) \longrightarrow \left(\mathbf{g} = \exp(\mathbf{h}) \right) \longrightarrow$$

Computing Derivatives: Vector Notation

- Inputs outputs are often vectors



- **h**(a) is a function from $\mathbf{R}^n$ to $\mathbf{R}^m$
- Chain rule generalizes to vector functions

Computing Derivatives: Vector Notation

- Let $\mathbf{f}(\mathbf{x})$: $\mathbf{R}^{\mathbf{n}} \rightarrow \mathbf{R}^{\mathbf{m}}$,
  - $\mathbf{x}$ is $\mathbf{n}$-dimensional vector and output $\mathbf{f}(\mathbf{x})$ is $\mathbf{m}$-dimensional vector
- Jacobian matrix
  - has $\mathbf{m}$ rows and $\mathbf{n}$ columns
  - has $\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}_j}$ in row $\mathbf{i}$, column $\mathbf{j}$

Computing Derivatives: Vector Notation

- $\mathbf{f}(\mathbf{x}): \mathbf{R}^n \rightarrow \mathbf{R}^m$ and $\mathbf{g}(\mathbf{x}): \mathbf{R}^k \rightarrow \mathbf{R}^n$

- $\mathbf{f}(\mathbf{g}(\mathbf{x})): \mathbf{R}^k \rightarrow \mathbf{R}^m$

- Chain rule for vector functions

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}$$

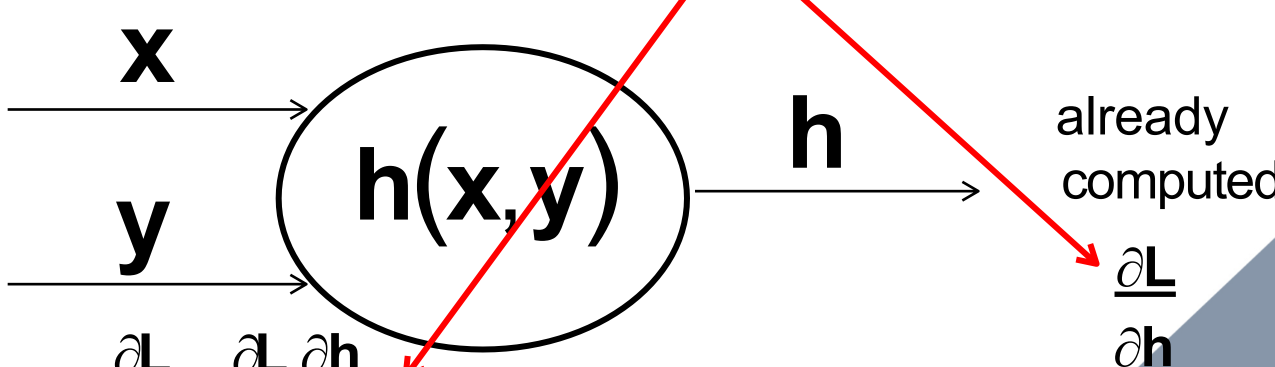
Jacobian matrices

Vector Notation: Look at One Node

- **h**, **x**, **y** are vectors
- already computed Jacobian $\frac{\partial \mathbf{L}}{\partial \mathbf{h}}$
- Need Jacobians $\frac{\partial \mathbf{L}}{\partial \mathbf{x}}, \frac{\partial \mathbf{L}}{\partial \mathbf{y}}$
- Easy to compute local node Jacobians $\frac{\partial \mathbf{h}}{\partial \mathbf{x}}, \frac{\partial \mathbf{h}}{\partial \mathbf{y}}$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$$

Jacobian matrices

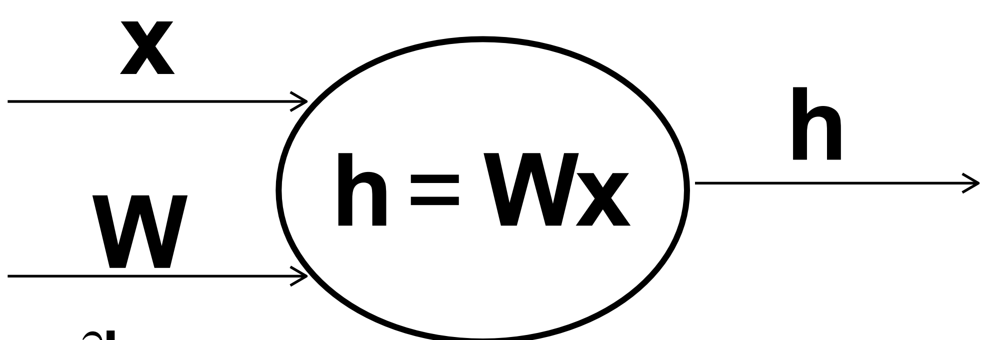


$$\frac{\partial \mathbf{L}}{\partial \mathbf{y}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{y}}$$

Vector Notation: Look at One Node

- Can apply to matrices (and tensors) as well
- But first vectorize matrix (or tensor)
- Say $\mathbf{W}$ is 10 x 5, stretch into 50x1 vector
- Still denote Jacobian by $\dfrac{\partial \mathbf{h}}{\partial \mathbf{W}}$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$$



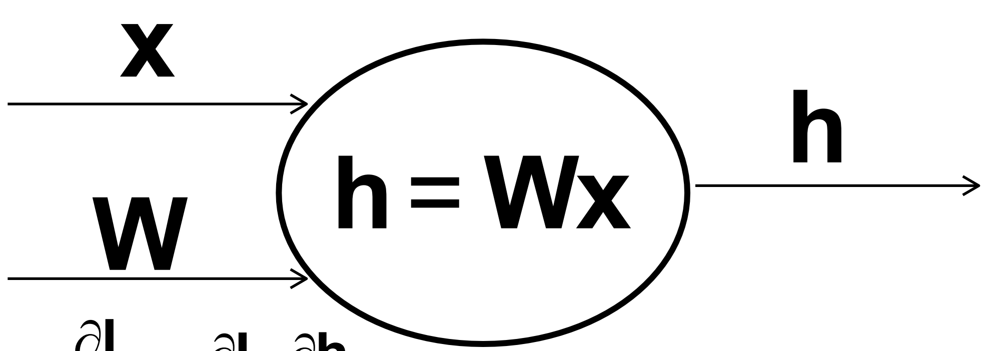
already computed $\dfrac{\partial \mathbf{L}}{\partial \mathbf{h}}$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{W}}$$

Vector Notation: Look at One Node

- Easy to compute local node Jacobians $\dfrac{\partial \mathbf{h}}{\partial \mathbf{x}}, \dfrac{\partial \mathbf{h}}{\partial \mathbf{W}}$
- But they can get very large (although sparse)
- Say $\mathbf{h}$ is 1000 x 1, $\mathbf{W}$ is 1000 x 500, then $\dfrac{\partial \mathbf{h}}{\partial \mathbf{W}}$ is 1000 x 500,000

$$\frac{\partial \mathbf{L}}{\partial \mathbf{x}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$$



already computed $\dfrac{\partial \mathbf{L}}{\partial \mathbf{h}}$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{W}} = \frac{\partial \mathbf{L}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{W}}$$

Summary

- ❑ Gradient Descent Optimization
- ❑ Chain rule of derivatives
- ❑ Back propagation

Next

- ❑ Advanced optimization methods
- ❑ Network regularization
- ❑ Practical tricks for training neural networks