Lecture 13

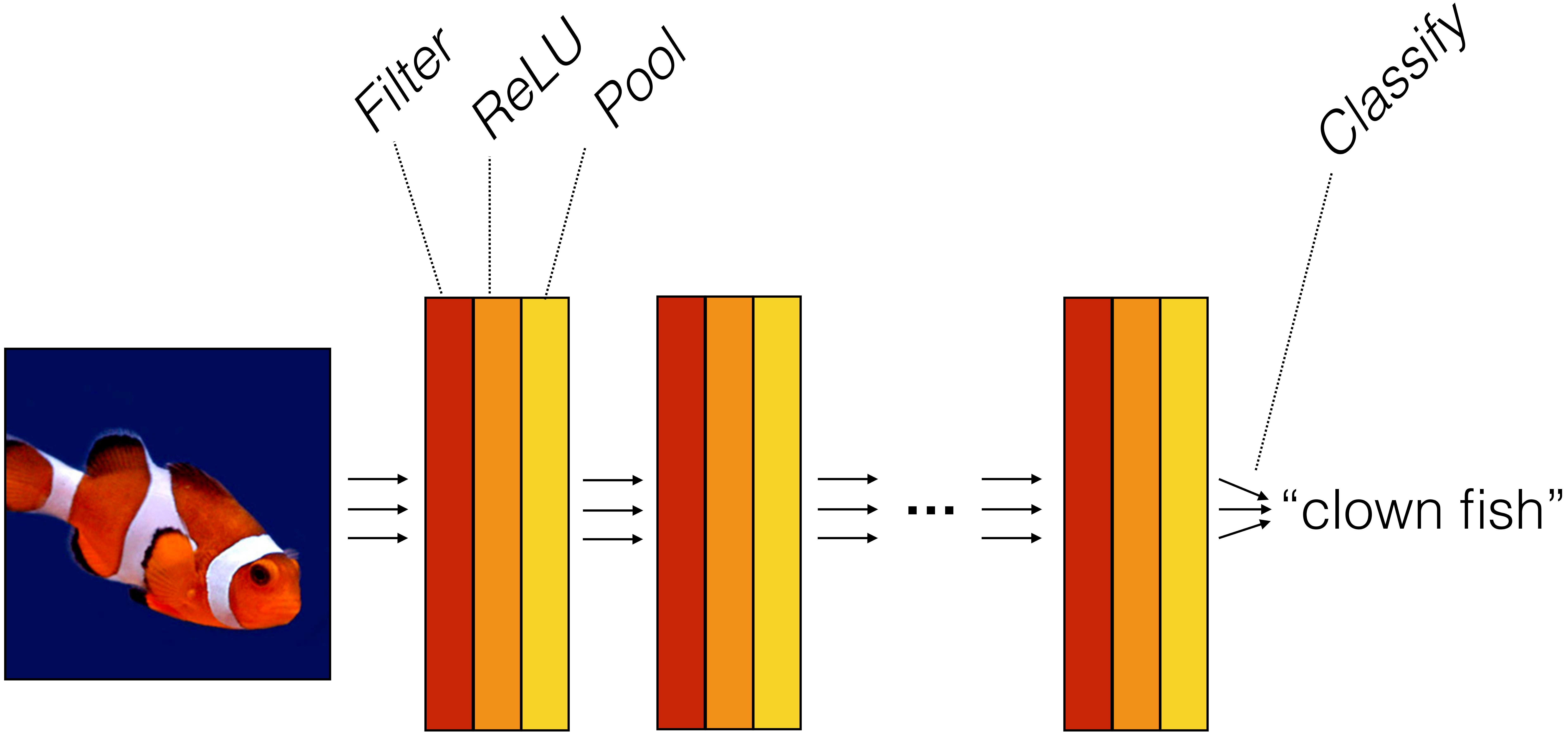
Stochastic gradient descent and backpropagation

6.869/6.819 Advances in Computer Vision

Fall 2019

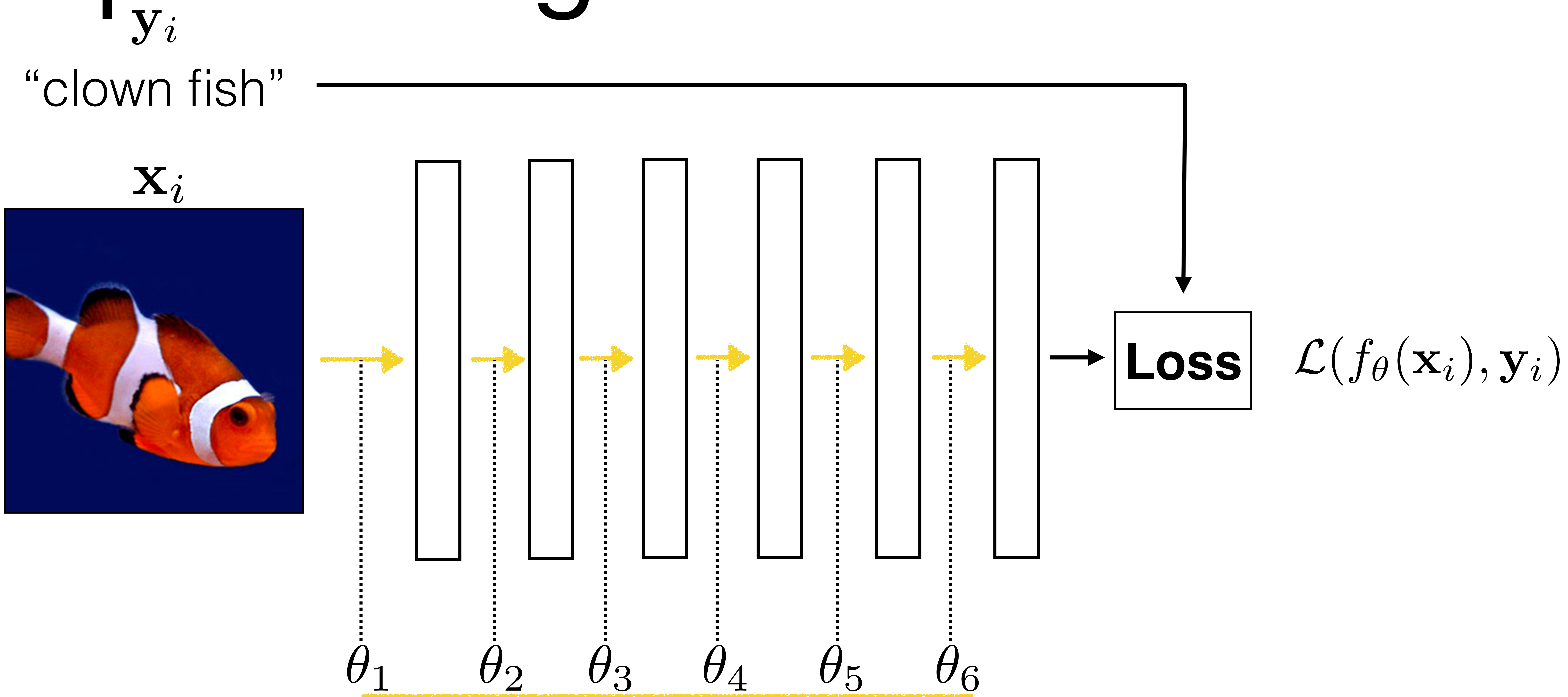
Bill Freeman, Antonio Torralba, Phillip Isola

Computation in a neural net



$$f(\mathbf{x}) = f_L(\dots f_2(f_1(\mathbf{x})))$$

Deep learning



$\mathbf{y}_i$

“clown fish”

$\mathbf{x}_i$

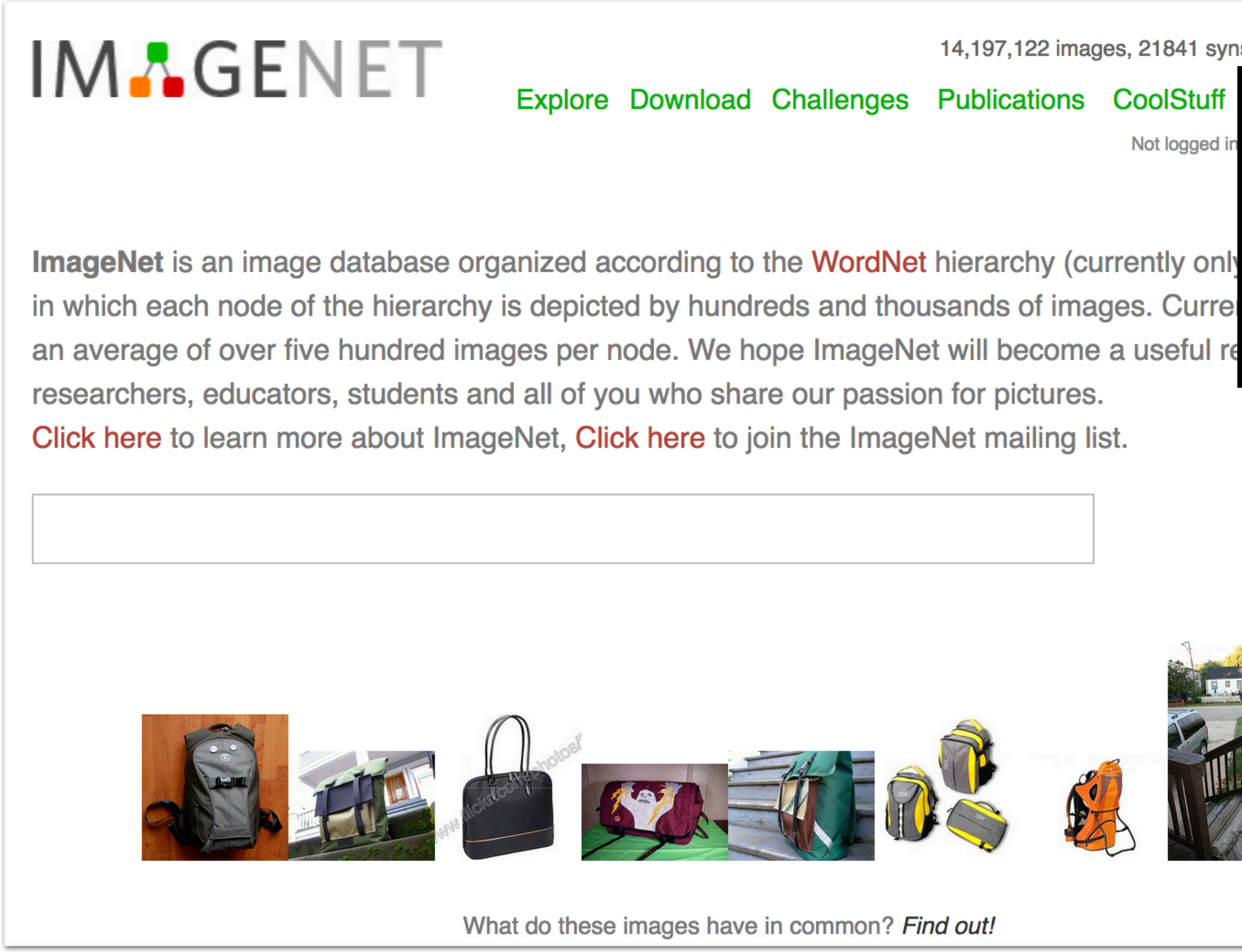
Loss

$\mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)$

θ_1 θ_2 θ_3 θ_4 θ_5 θ_6

$$\boxed{\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}$$

ImageNet classification and Neural nets



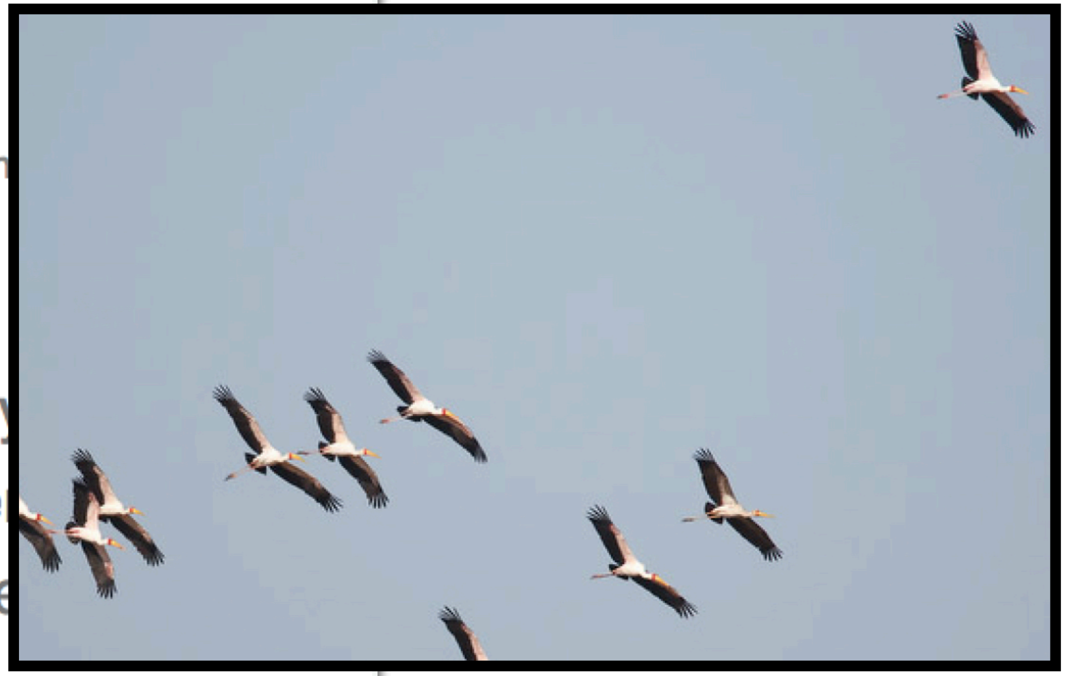
14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications CoolStuff

Not logged i

ImageNet is an image database organized according to the WordNet hierarchy (currently onl
in which each node of the hierarchy is depicted by hundreds and thousands of images. Curre
an average of over five hundred images per node. We hope ImageNet will become a useful re
researchers, educators, students and all of you who share our passion for pictures.
Click here to learn more about ImageNet, Click here to join the ImageNet mailing list.

What do these images have in common? *Find out!*



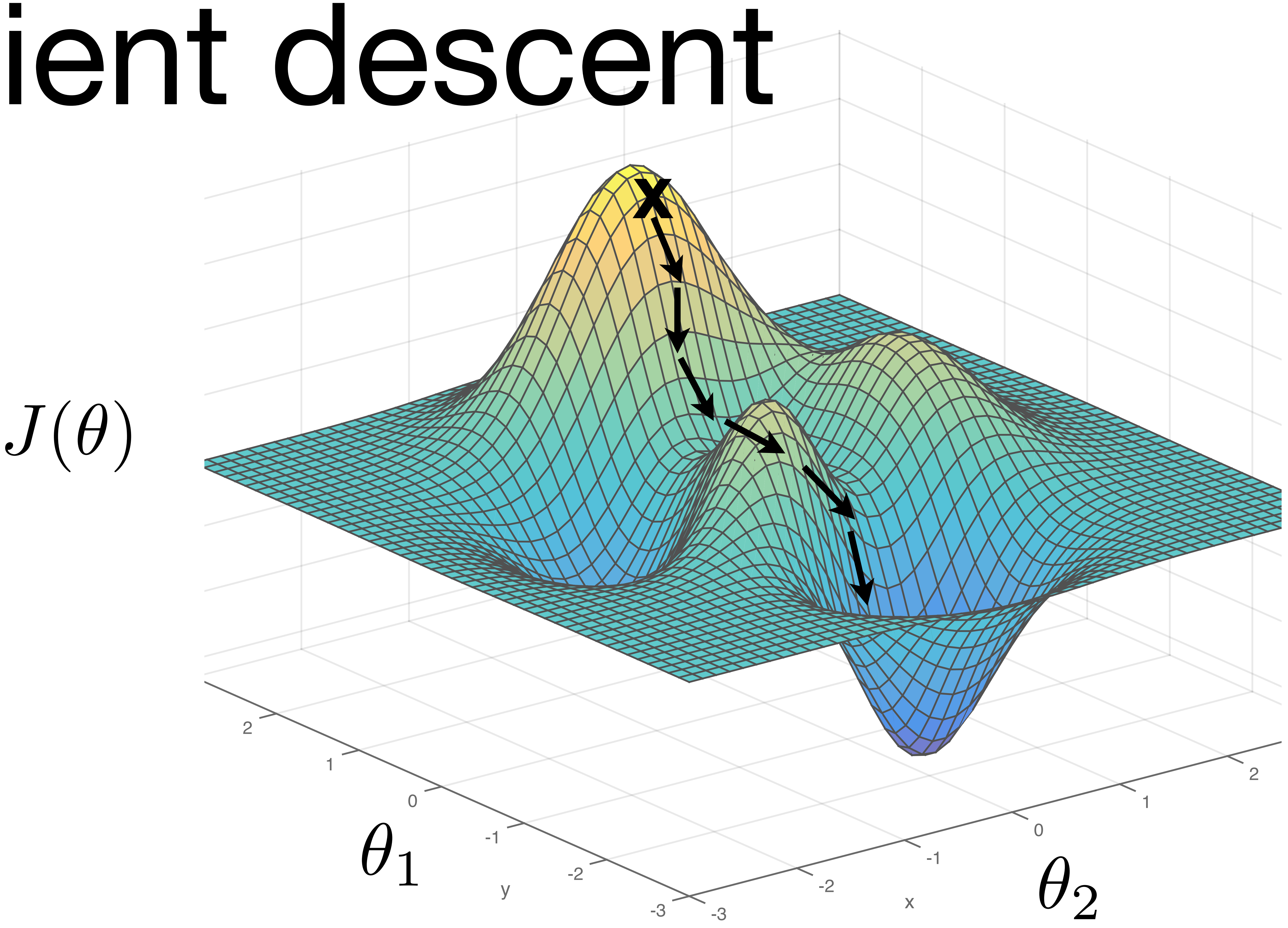
$$\xrightarrow{f}$$

"Birds"

Gradient descent

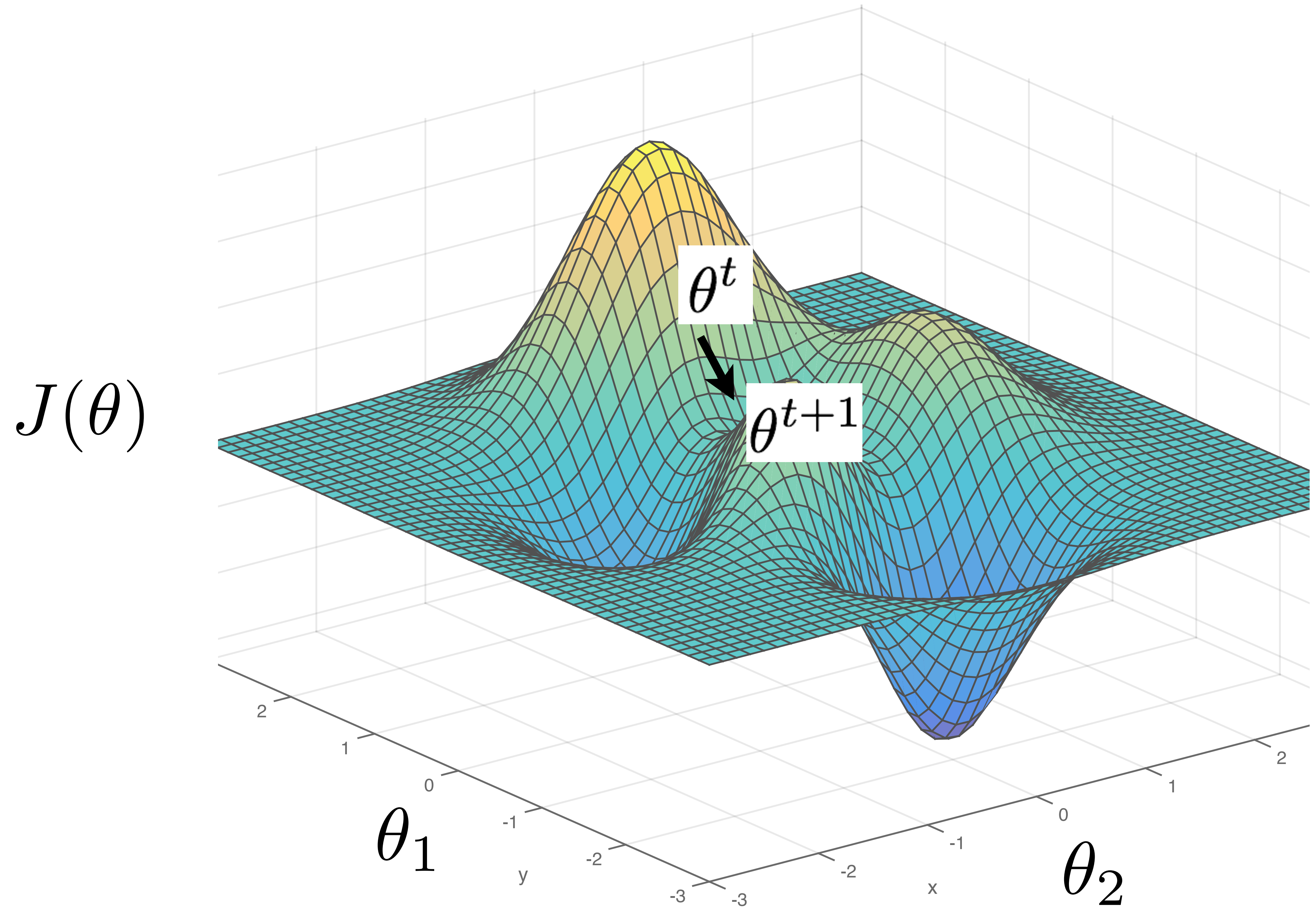
$$\boxed{\theta^* = \arg\min_{\theta} \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}}$$

Gradient descent



$$\boxed{\theta^* = \arg\min_{\theta} J(\theta)}$$

Gradient descent



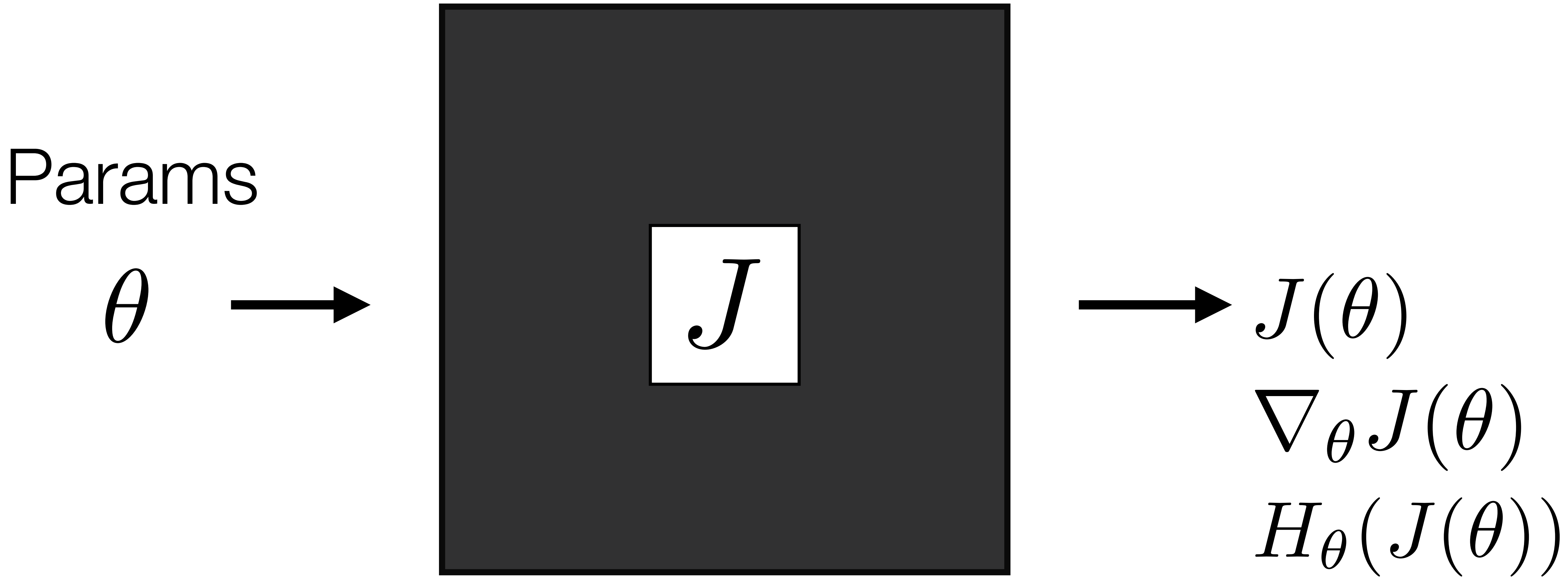
$$\boxed{\theta^* = \arg\min_\theta \underbrace{\sum_{i=1}^{N} \mathcal{L}(f_\theta(\mathbf{x}_i), \mathbf{y}_i)}_{J(\theta)}}$$

One iteration of gradient descent:

$$\theta^{t+1} = \theta^t - \eta_t \left. \frac{\partial J(\theta)}{\partial \theta} \right|_{\theta=\theta^t}$$

learning rate

Optimization



Params

θ

J

$J(\theta)$

$\nabla_\theta J(\theta)$

$H_\theta(J(\theta))$

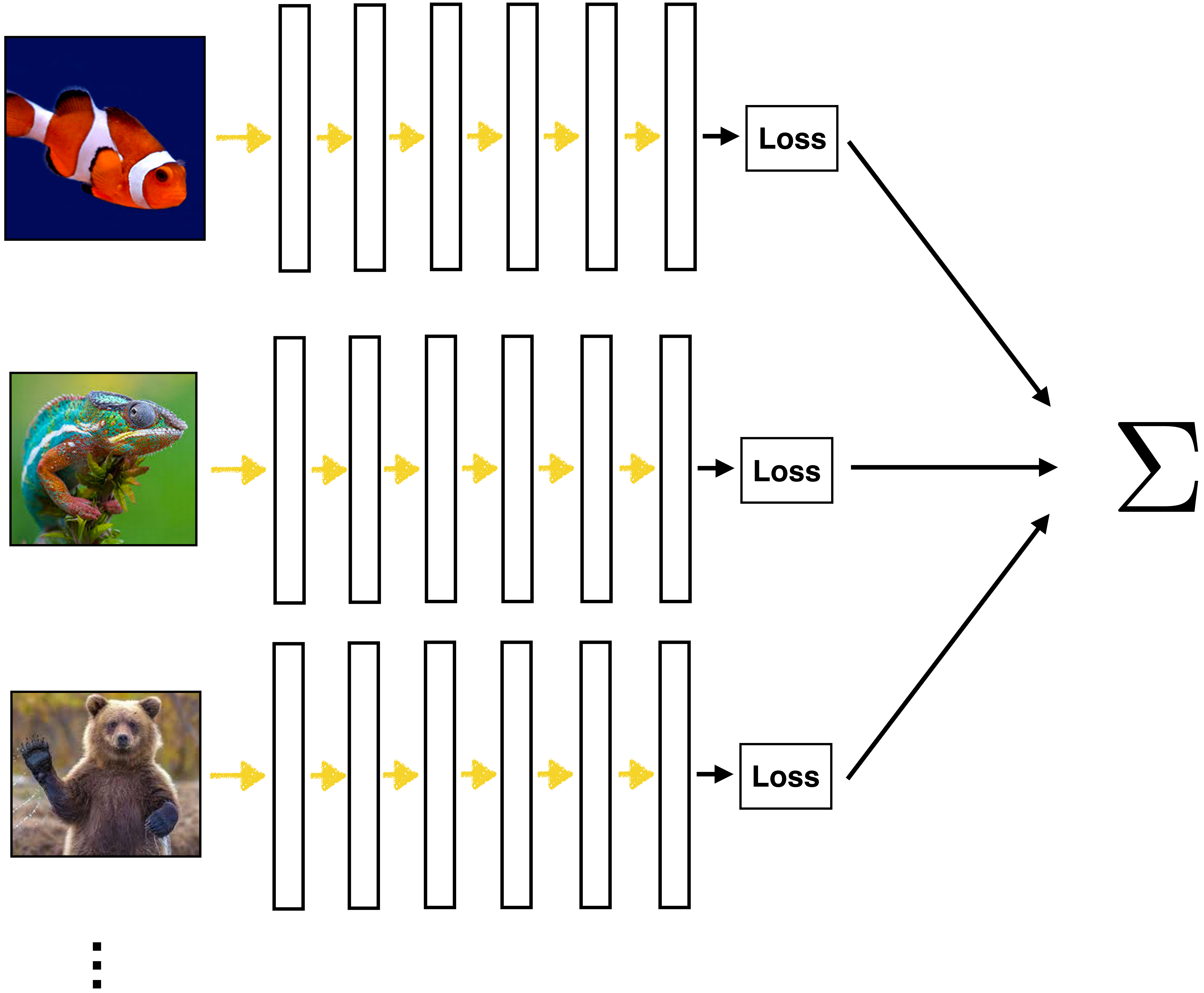
$$\boxed{\theta^* = \arg\min_\theta J(\theta)}$$

- What’s the knowledge we have about J?
  - We can evaluate $J(\theta)$ ⟵ Black box optimization
  - We can evaluate $J(\theta)$ and $\underbrace{\nabla_\theta J(\theta)}_{}$ ⟵ First order optimization

    Gradient
  - We can evaluate $J(\theta)$, $\nabla_\theta J(\theta)$, and $H_\theta(J(\theta))$ ⟵ Second order optimization

    Hessian

Batch (parallel) processing



Stochastic gradient descent (SGD)

- Want to minimize overall loss function **J**, which is sum of individual losses over each example.
- In Stochastic gradient descent, compute gradient on sub-set (batch) of data.

  If batchsize=1 then θ is updated after each example.

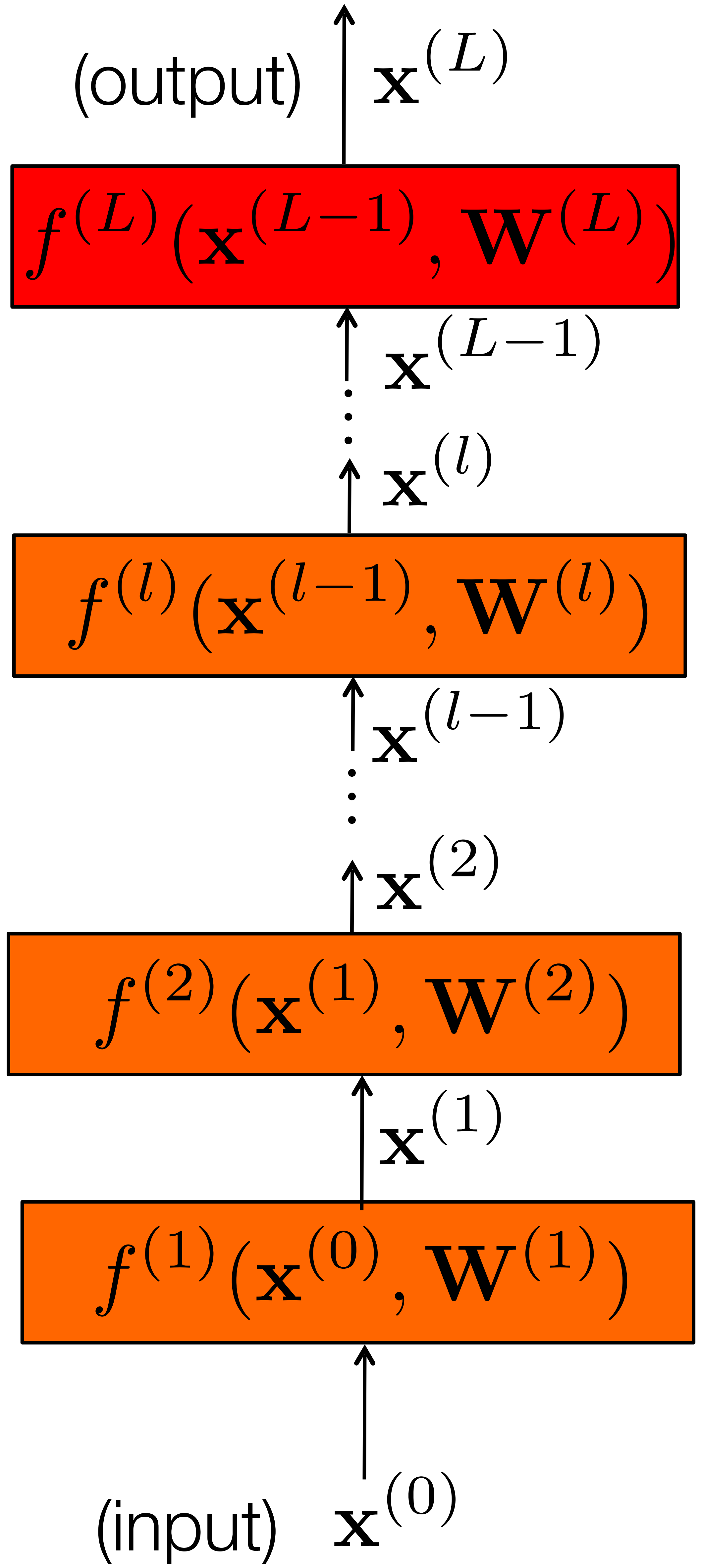
  If batchsize=N (full set) then this is standard gradient descent.
- Gradient direction is noisy, relative to average over all examples (standard gradient descent).
- Advantages
  - Faster: approximate total gradient with small sample
  - Implicit regularizer
- Disadvantages
  - High variance, unstable updates

Forward pass

- Consider model with L layers. Layer l has vector of weights $\mathbf{W}^{(l)}$
- **Forward pass**: takes input $\mathbf{x}^{(l-1)}$ and passes it through each layer $f^{(l)}$:

$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$

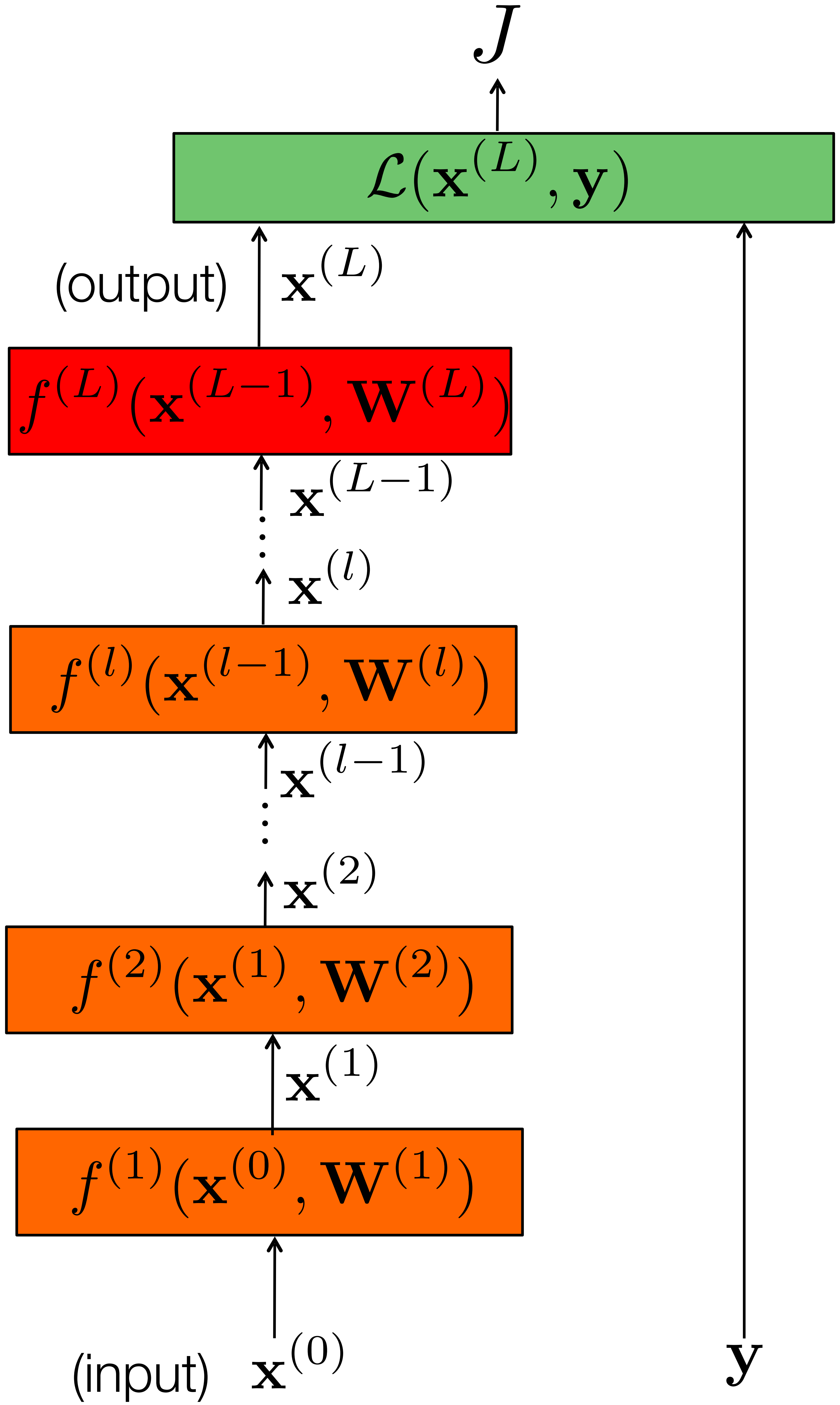
- Output of layer l is $\mathbf{x}^{(l)}$.
- Network output (top layer) is $\mathbf{x}^{(L)}$.



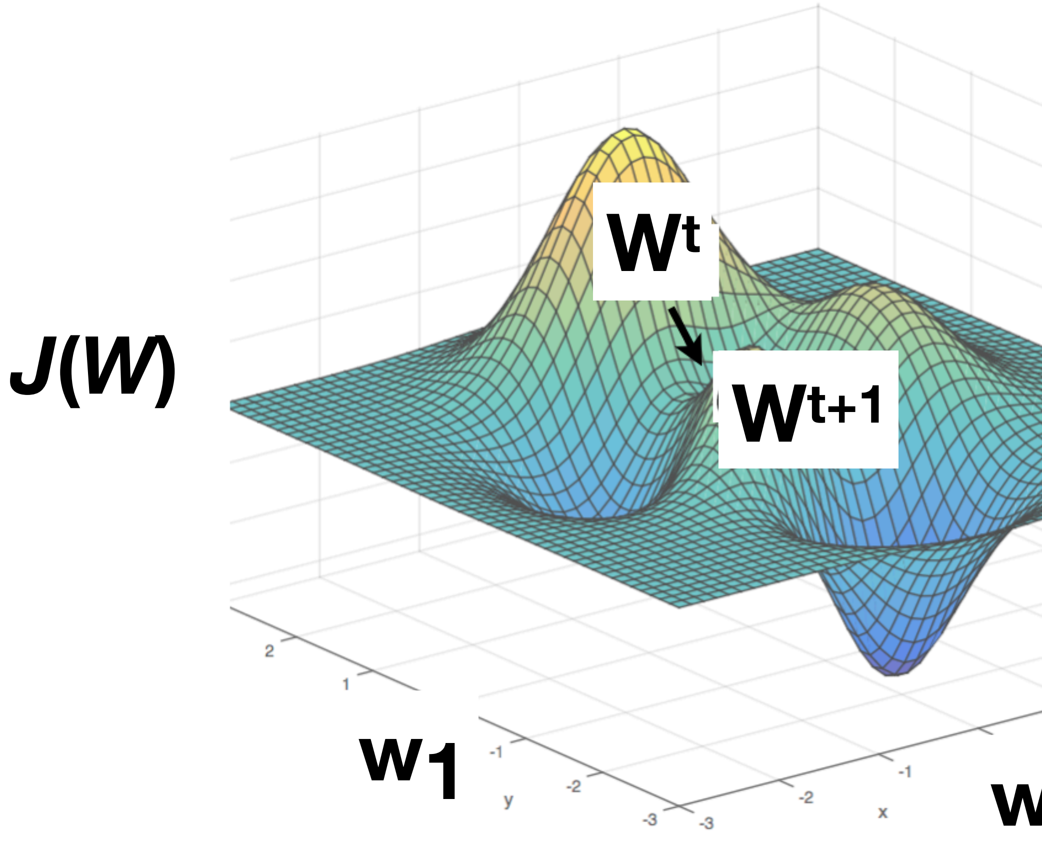
Forward pass

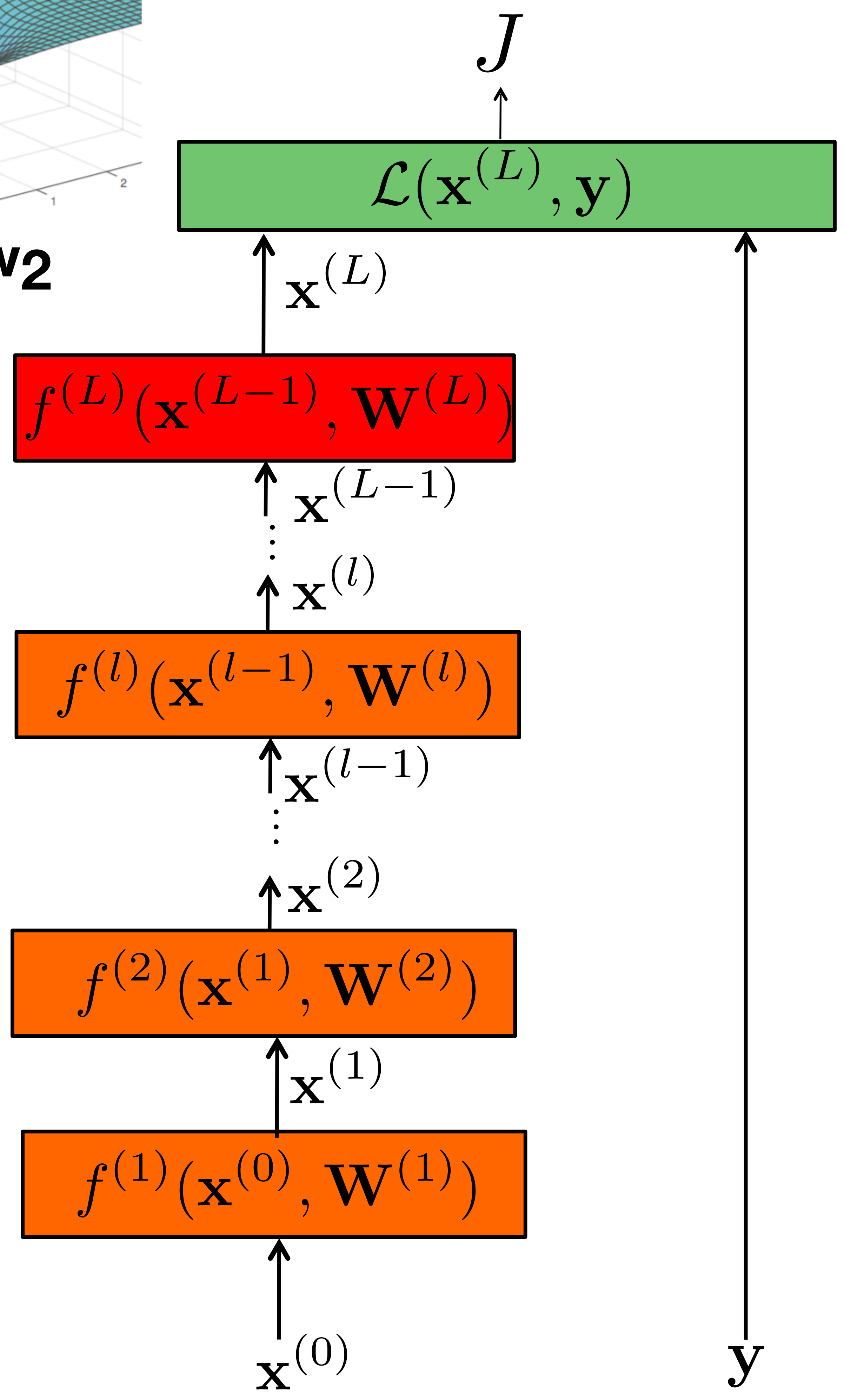
- Consider model with L layers. Layer l has vector of weights $\mathbf{W}^{(l)}$
- **Forward pass**: takes input $\mathbf{x}^{(l-1)}$ and passes it through each layer $f^{(l)}$:

  $$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$
- Output of layer l is $\mathbf{x}^{(l)}$.
- Network output (top layer) is $\mathbf{x}^{(L)}$.
- **Loss function** $\mathcal{L}$ compares $\mathbf{x}^{(L)}$ to $\mathbf{y}$.
- Overall loss is the sum of the cost over all training examples: $$J = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i^{(L)}, \mathbf{y}_i)$$



Gradient descent





- We need to compute gradients of the cost with respect to model parameters $\mathbf{W}^{(l)}$.
- By design, each layer is differentiable with respect to its parameters and input.

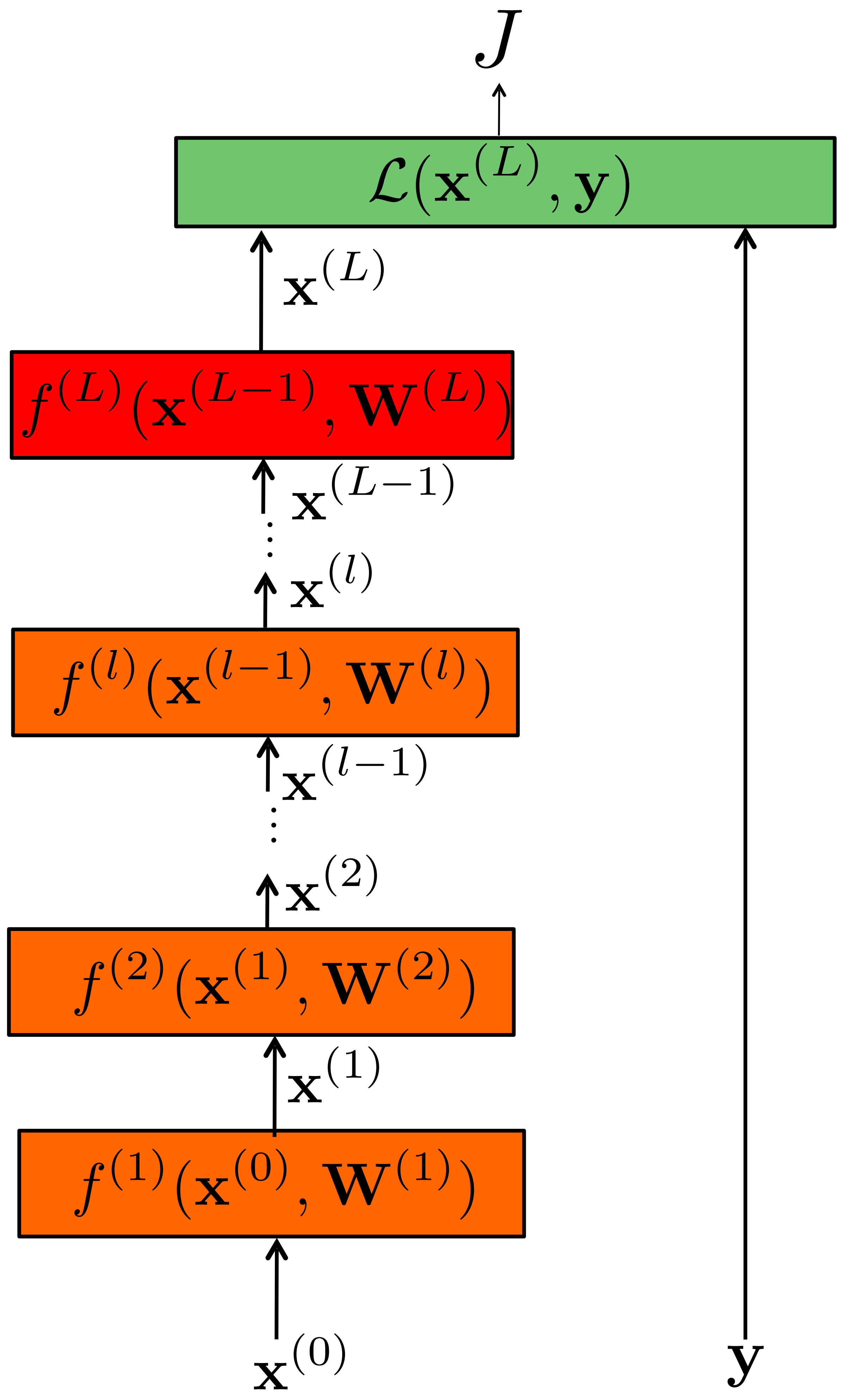
Computing gradients

To compute the gradients, we could start by writing the full energy J as a function of the network parameters.

$$J(\mathbf{W}) = \sum_{i=1} \mathcal{L}(f^{(L)}(\dots f^{(2)}(f^{(1)}(\mathbf{x}_i^{(0)}, \mathbf{W}^{(1)}), \mathbf{W}^{(2)}), \dots \mathbf{W}^{(L)}), \mathbf{y}_i)$$

And then compute the partial derivatives…

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$$



instead, we can use the chain rule to derive a compact algorithm: **backpropagation**

(

Matrix calculus

- $\mathbf{x}$ column vector of size $[n \times 1]$:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

- We now define a function on vector $\mathbf{x}$: $\mathbf{y} = f(\mathbf{x})$
- If y is a scalar, then

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{pmatrix}$$

The derivative of y is a row vector of size $[1 \times n]$

- If $\mathbf{y}$ is a vector $[m \times 1]$, then (*Jacobian formulation*):

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

The derivative of y is a matrix of size $[m \times n]$

(m rows and n columns)

Matrix calculus

- If y is a scalar and $\mathbf{X}$ is a matrix of size $[n \times m]$, then

$$\frac{\partial y}{\partial \mathbf{X}} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{n1}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial y}{\partial x_{1m}} & \frac{\partial y}{\partial x_{2m}} & \cdots & \frac{\partial y}{\partial x_{nm}} \end{pmatrix}$$

The output is a matrix of size $[m \times n]$

Matrix calculus

- Chain rule:

For the function: $h(\mathbf{x}) = f(g(\mathbf{x}))$

Its derivative is: $h'(\mathbf{x}) = f'(g(\mathbf{x}))g'(\mathbf{x})$

and writing $\mathbf{z} = f(\mathbf{u})$, and $\mathbf{u} = g(\mathbf{x})$:

$$\underset{\underset{[m \times n]}{\uparrow}}{\left.\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}}} = \underset{\underset{[m \times p]}{\uparrow}}{\left.\frac{\partial \mathbf{z}}{\partial \mathbf{u}}\right|_{\mathbf{u}=g(\mathbf{a})}} \cdot \underset{\underset{[p \times n]}{\uparrow}}{\left.\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}}}$$

with $p =$ length of vector $\mathbf{u}$ $= |\mathbf{u}|$, $m = |\mathbf{z}|$, and $n = |\mathbf{x}|$

Example, if $|\mathbf{z}| = 1\,, |\mathbf{u}| = 2\,, |\mathbf{x}| = 4$

$h'(\mathbf{x}) =$

Matrix calculus

- Chain rule:

For the function: $h(\mathbf{x}) = f^{(n)}(f^{(n-1)}(\dots f^{(1)}(\mathbf{x})))$

With $\mathbf{u}^{(1)} = f^{(1)}(\mathbf{x})$

$\mathbf{u}^{(i)} = f^{(i)}(\mathbf{u}^{(i-1)})$

$\mathbf{z} = \mathbf{u}^{(n)} = f^{(n)}(\mathbf{u}^{(n-1)})$

The derivative becomes a product of matrices:

$$\left.\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}} = \left.\frac{\partial \mathbf{z}}{\partial \mathbf{u}^{(n-1)}}\right|_{\mathbf{u}^{(n-1)}=f^{(n-1)}(\mathbf{u}^{n-2})} \cdot \left.\frac{\partial \mathbf{u}^{(n-1)}}{\partial \mathbf{u}^{(n-2)}}\right|_{\mathbf{u}^{(n-2)}=f^{(n-2)}(\mathbf{u}^{n-3})} \cdots \left.\frac{\partial \mathbf{u}^{(2)}}{\partial \mathbf{u}^{(1)}}\right|_{\mathbf{u}^{(1)}=f^{(1)}(\mathbf{a})} \cdot \left.\frac{\partial \mathbf{u}^{(1)}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{a}}$$

(exercise: check that all the matrix dimensions work out fine)

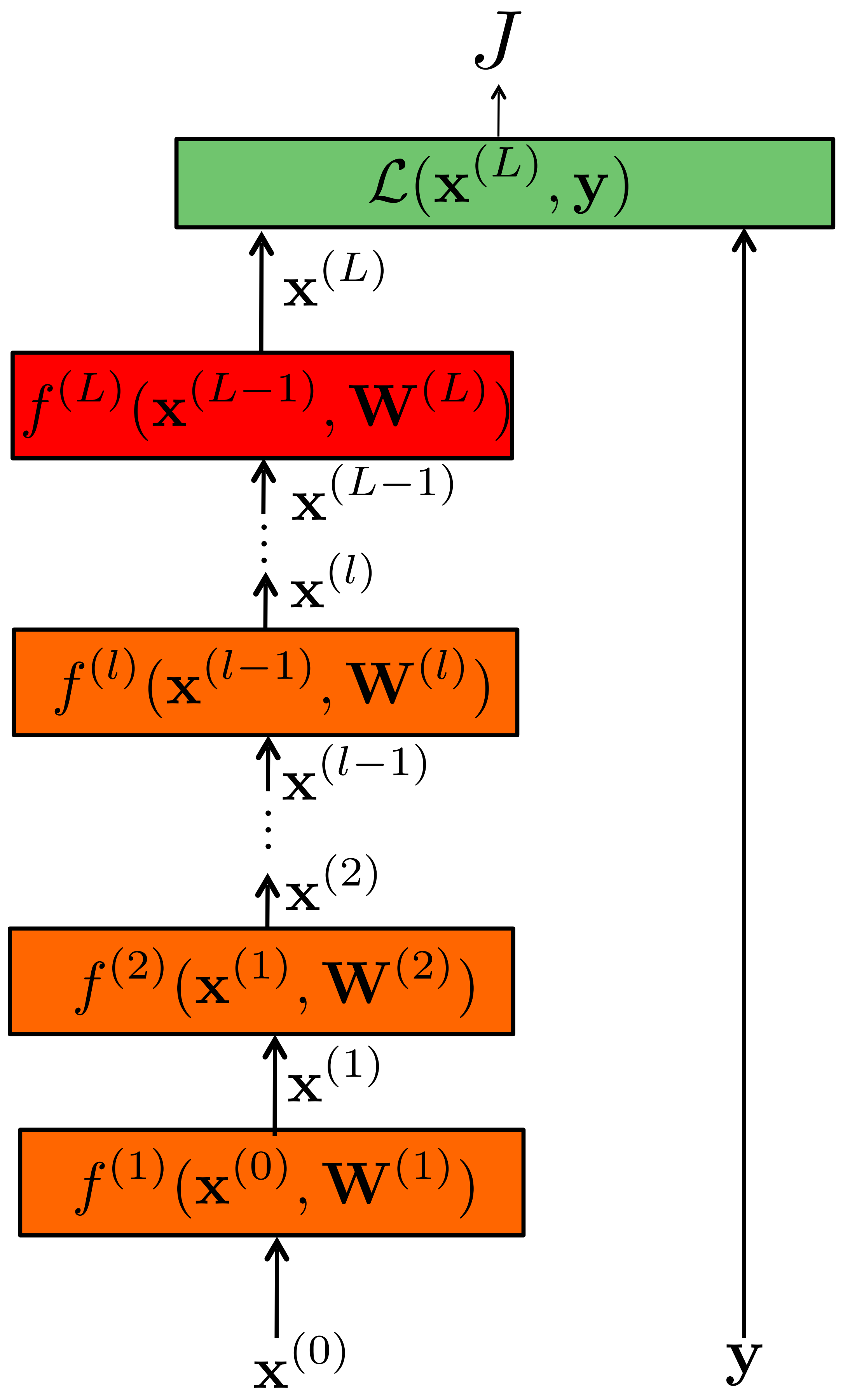
)

Computing gradients

To compute the gradients, we could start by writing the full energy J as a function of the network parameters.

$$J(\mathbf{W}) = \sum_{i=1} \mathcal{L}(f^{(L)}(\dots f^{(2)}(f^{(1)}(\mathbf{x}_i^{(0)}, \mathbf{W}^{(1)}), \mathbf{W}^{(2)}), \dots \mathbf{W}^{(L)}), \mathbf{y}_i)$$

And then compute the partial derivatives… instead, we can use the chain rule to derive a compact algorithm: **backpropagation**



Computing gradients

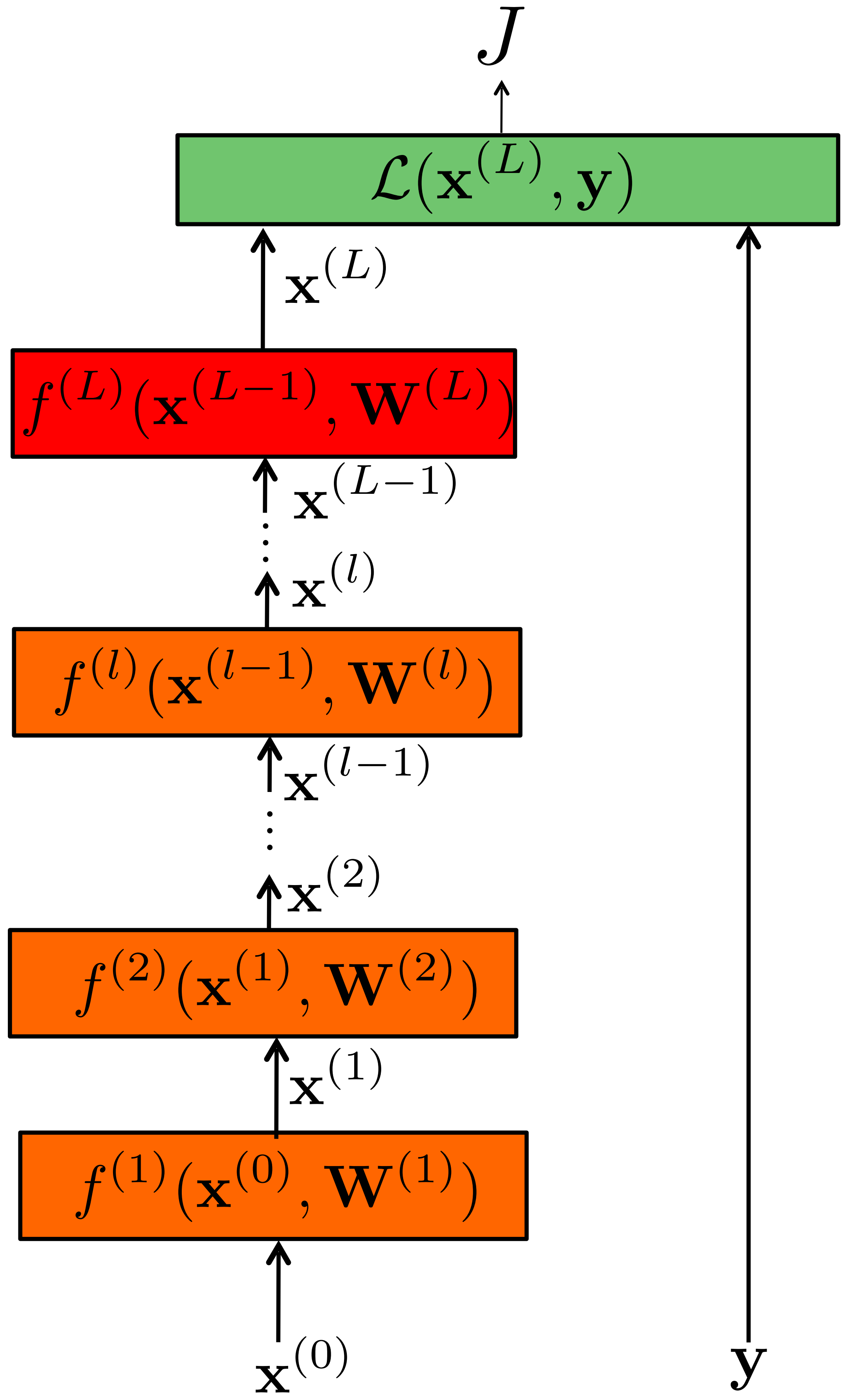
The loss J is the sum of the losses associated to each training example $\{\mathbf{x}_i^{(0)}, \mathbf{y}_i\}$

$$J(\mathbf{W}) = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i^{(L)}, \mathbf{y}_i; \mathbf{W})$$

Its gradient with respect to each of the network's parameters w is:

$$\frac{\partial J(\mathbf{W})}{\partial w} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}(\mathbf{x}_i^{(L)}, \mathbf{y}_i; \mathbf{W})}{\partial w}$$

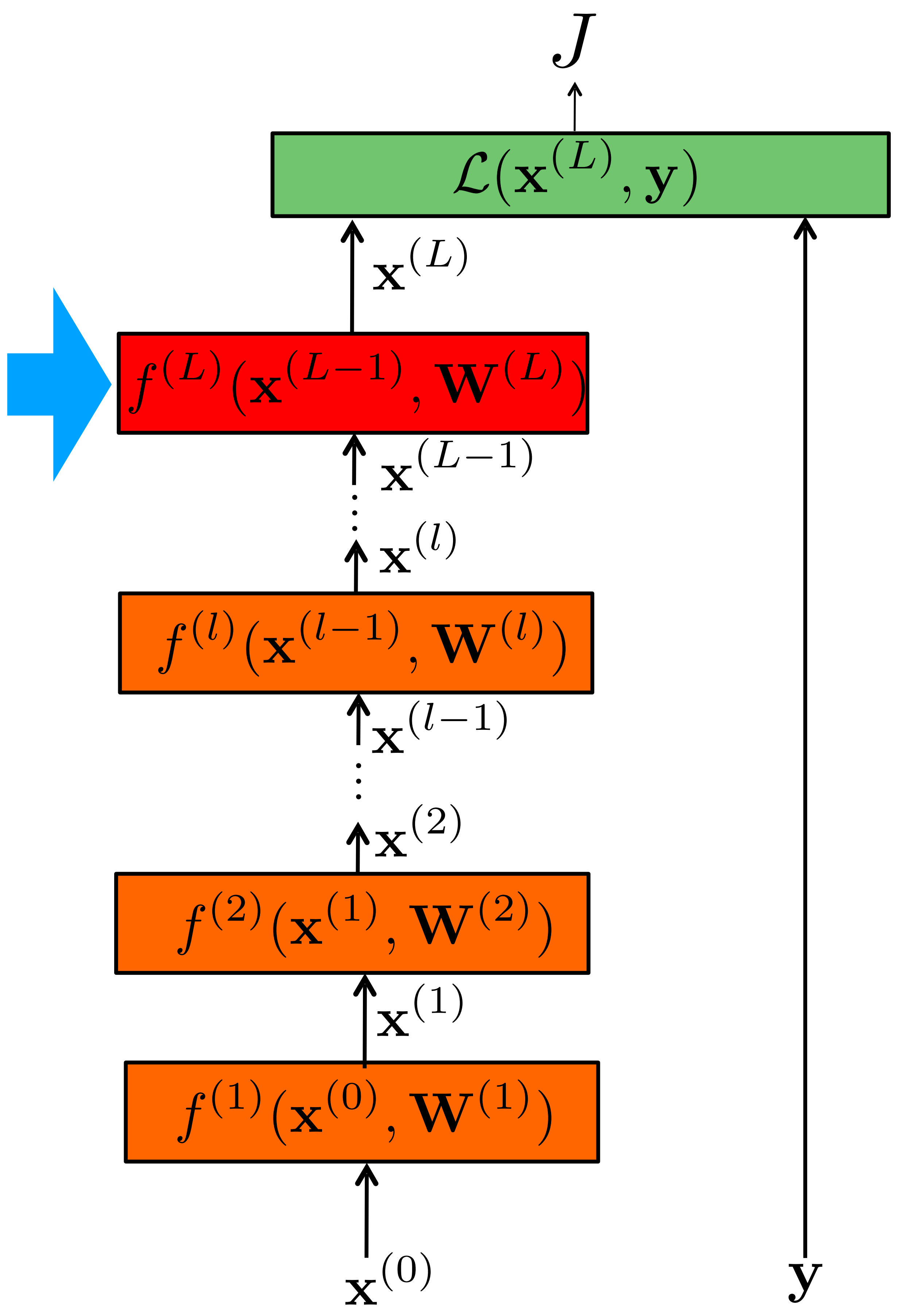
is how much J varies when the parameter w is varied.



Computing gradients

We could write the loss function to get the gradients as:

$$\mathcal{L}(\mathbf{x}^{(L)}, \mathbf{y}; \mathbf{W}) = \mathcal{L}(f^{(L)}(\mathbf{x}^{(L-1)}, \mathbf{W}^{(L)}), \mathbf{y})$$



If we compute the gradient with respect to the parameters of the last layer (output layer) $W^{(L)}$, using the **chain rule**:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \cdot \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \cdot \frac{\partial f^{(L)}(\mathbf{x}^{(L-1)}, \mathbf{W}^{(L)})}{\partial \mathbf{W}^{(L)}}$$

How much the loss changes when we change $W^{(L)}$?

The change is the product between how much the loss changes when we change the output of the last layer and how much the output changes when we change the layer parameters.

Computing gradients: cost layer

If we compute the gradient with respect to the parameters of the last layer (output layer) $W^{(L)}$, using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \cdot \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \cdot \frac{\partial f^{(L)}(\mathbf{x}^{(L-1)}, \mathbf{W}^{(L)})}{\partial \mathbf{W}^{(L)}}$$

For example, for an Euclidean loss:

$$\mathcal{L}(\mathbf{x}^{(L)}, \mathbf{y}) = \frac{1}{2} \left\| \mathbf{x}^{(L)} - \mathbf{y} \right\|_2^2$$

The gradient is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} = \mathbf{x}^{(L)} - \mathbf{y}$$

Will depend on the layer structure and non-linearity.

Computing gradients: layer l

We could write the full loss function to get the gradients:

$$\mathcal{L}(\mathbf{x}^{(L)}, \mathbf{y}; \mathbf{W}) = \mathcal{L}(f^{(L)}(\ldots f^{(2)}(f^{(1)}(\mathbf{x}^{(0)}, \mathbf{W}^{(1)}), \mathbf{W}^{(2)}), \ldots \mathbf{W}^{(L)}), \mathbf{y})$$

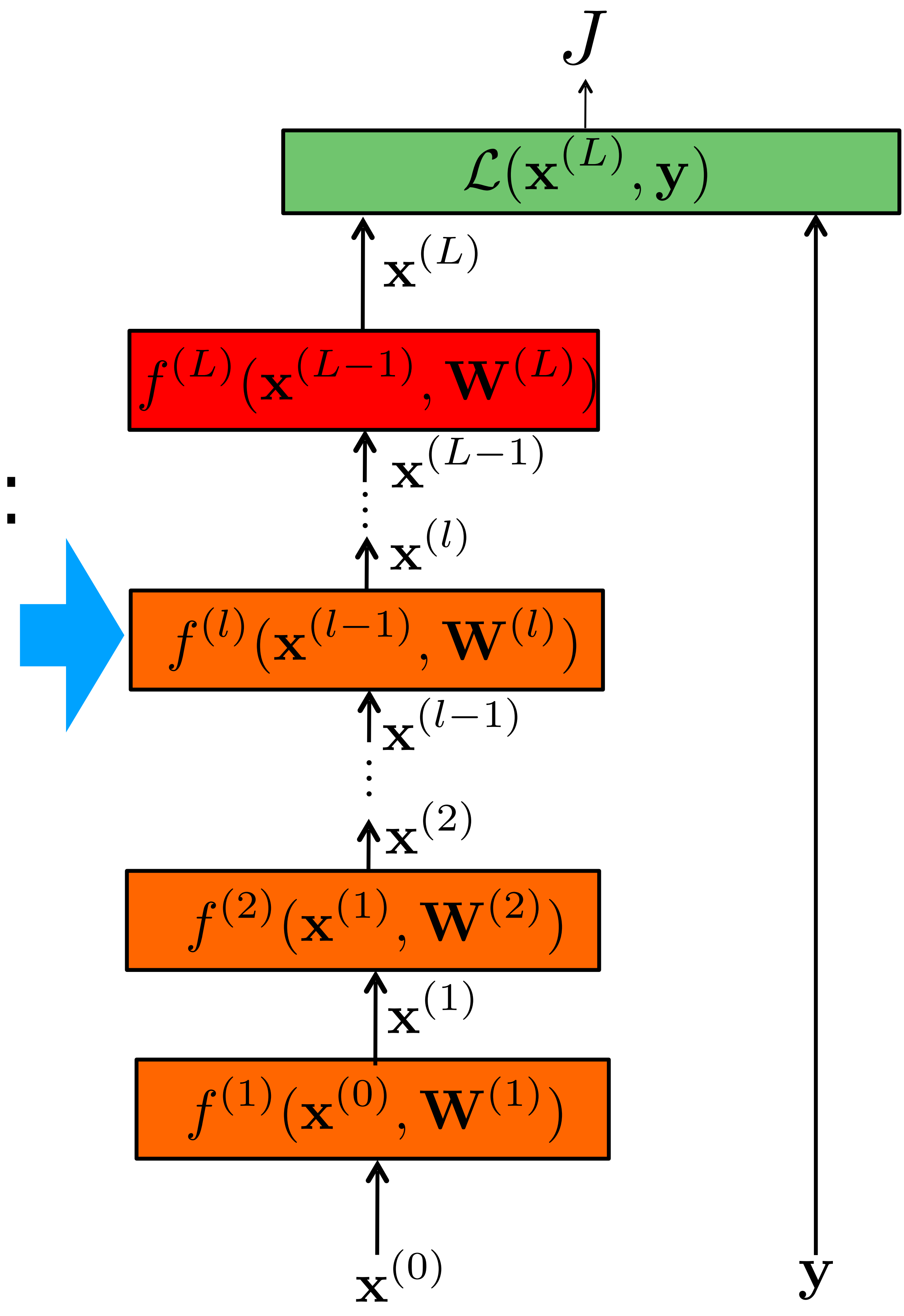
If we compute the gradient with respect to $\mathbf{W}^{(l)}$, using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \cdot \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{x}^{(L-1)}} \cdot \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{x}^{(L-2)}} \cdots \frac{\partial \mathbf{x}^{(l+1)}}{\partial \mathbf{x}^{(l)}}}_{\dfrac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}} \cdot \underset{\uparrow}{\frac{\partial \mathbf{x}^{(l)}}{\partial \mathbf{W}^{(l)}}}$$

And this can be computed iteratively!

$$\frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{W}^{(l)}}$$

This is easy.



Backpropagation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l+1)}} \longrightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \longrightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$$

And this can be computed iteratively. We start at the top (L) and we can compute the gradient at layer l-1

If we have the value of $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$ we can compute the gradient at the layer below as:

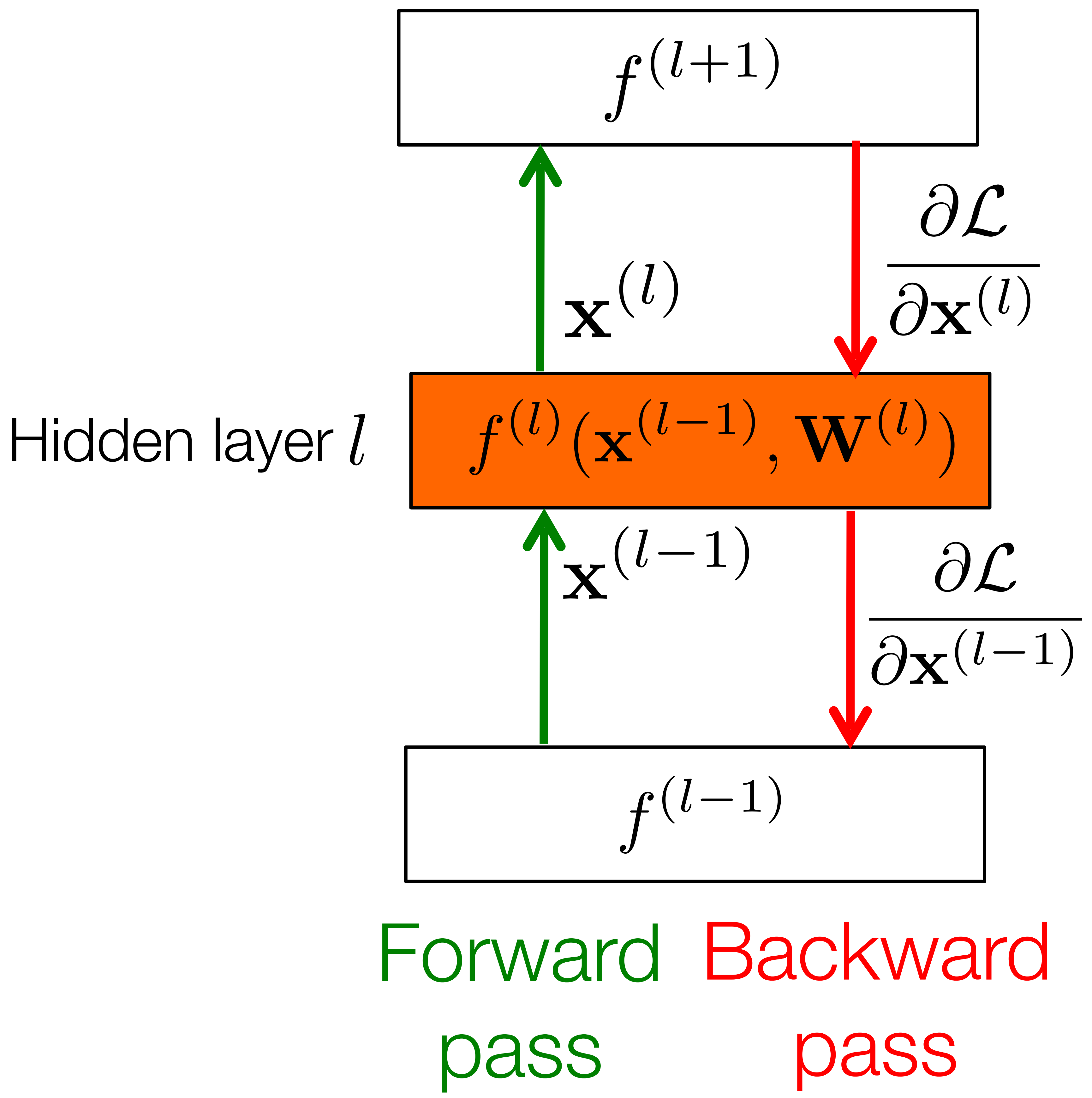
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial \mathbf{x}^{(l)}}{\partial \mathbf{x}^{(l-1)}}$$

Gradient layer l-1

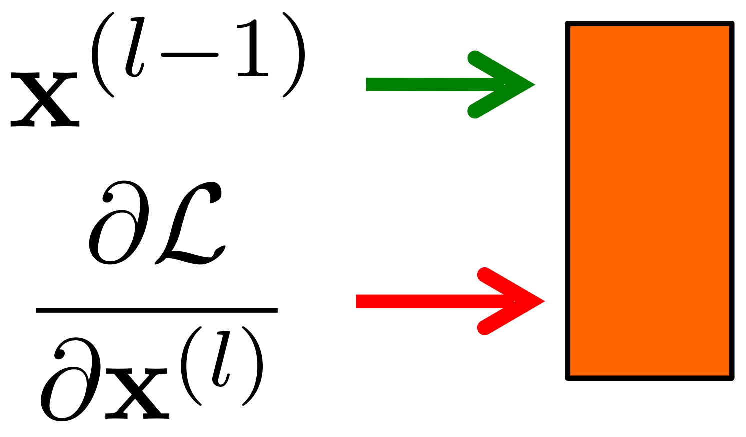
Gradient layer l

$$\frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

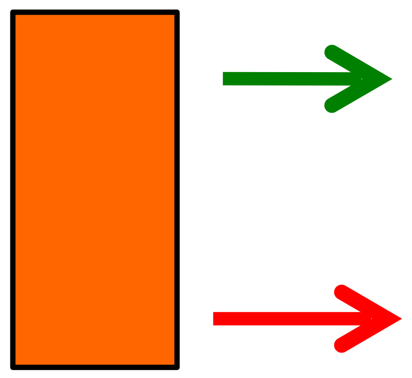
Backpropagation — Goal: to update parameters of layer l



- Layer l has two inputs (during training)



- We compute the outputs



$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

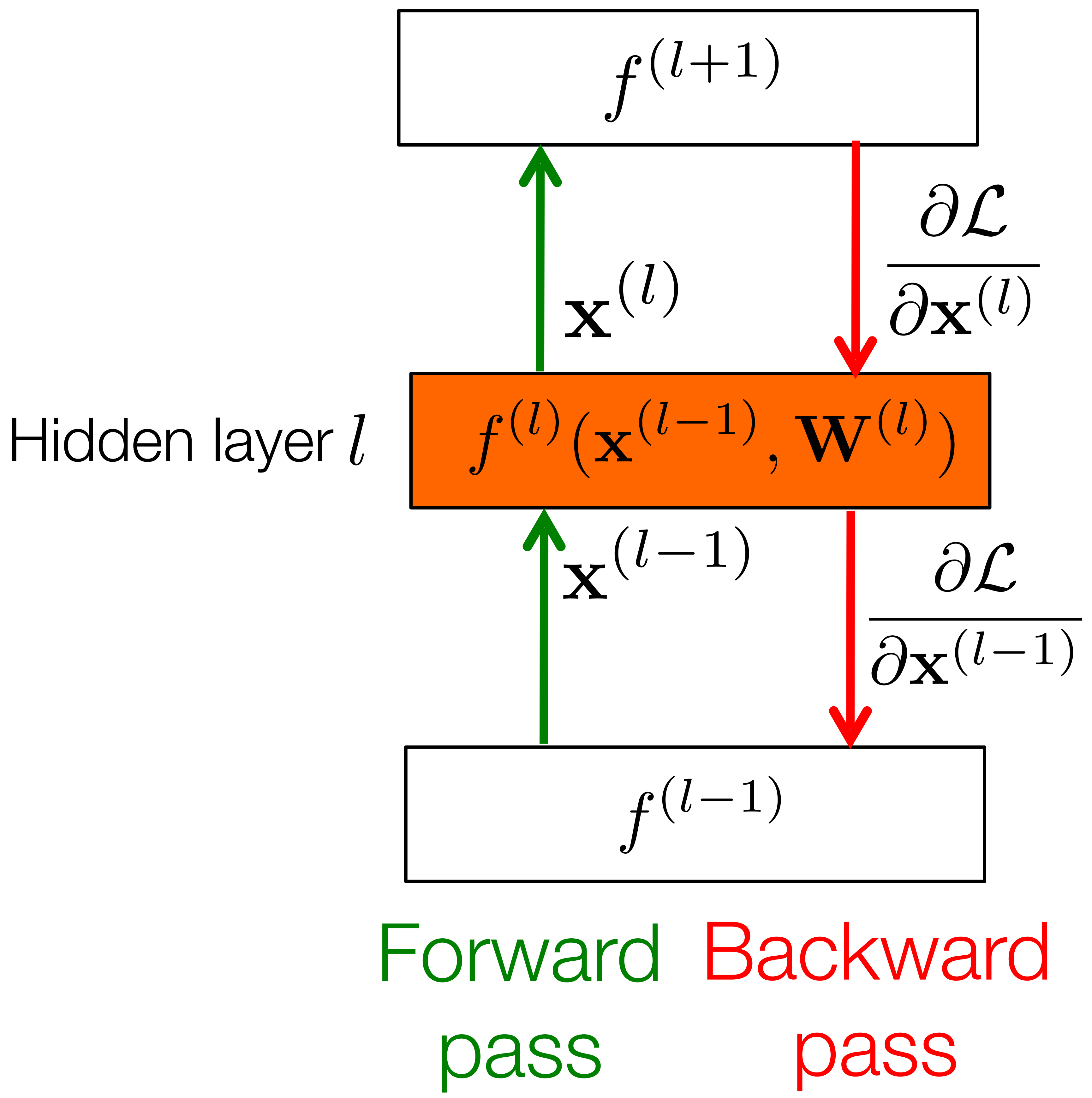
- To compute the output, we need:

$$\frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

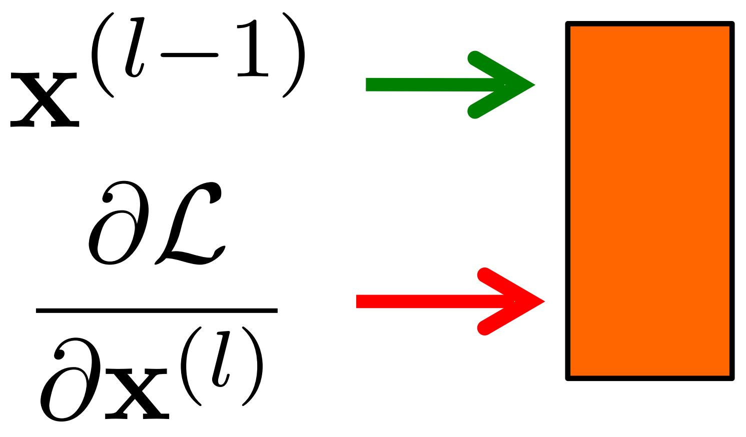
- To compute the weight update, we need:

$$\frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{W}^{(l)}}$$

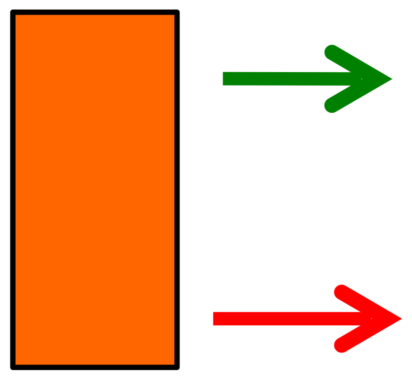
Backpropagation — Goal: to update parameters of layer l



- Layer l has two inputs (during training)



- We compute the outputs



$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

- The weight update equation is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{W}^{(l)}}$$

$$\boxed{\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} + \eta \left(\frac{\partial J}{\partial \mathbf{W}^{(l)}} \right)^{T}}$$

(sum over all training examples to get J)

Backpropagation Summary

- Forward pass: for each training example, compute the outputs for all layers:

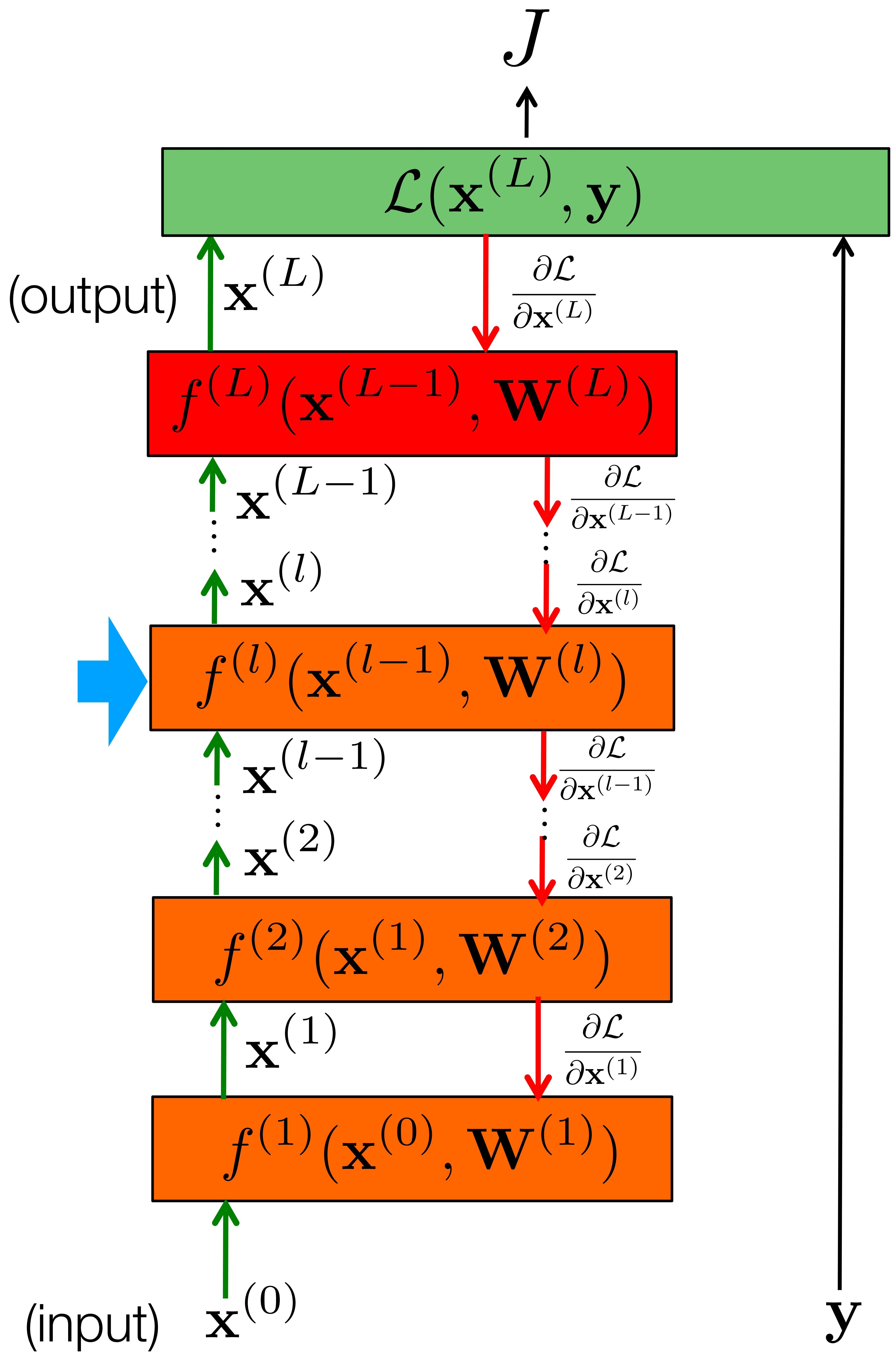
$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})$$

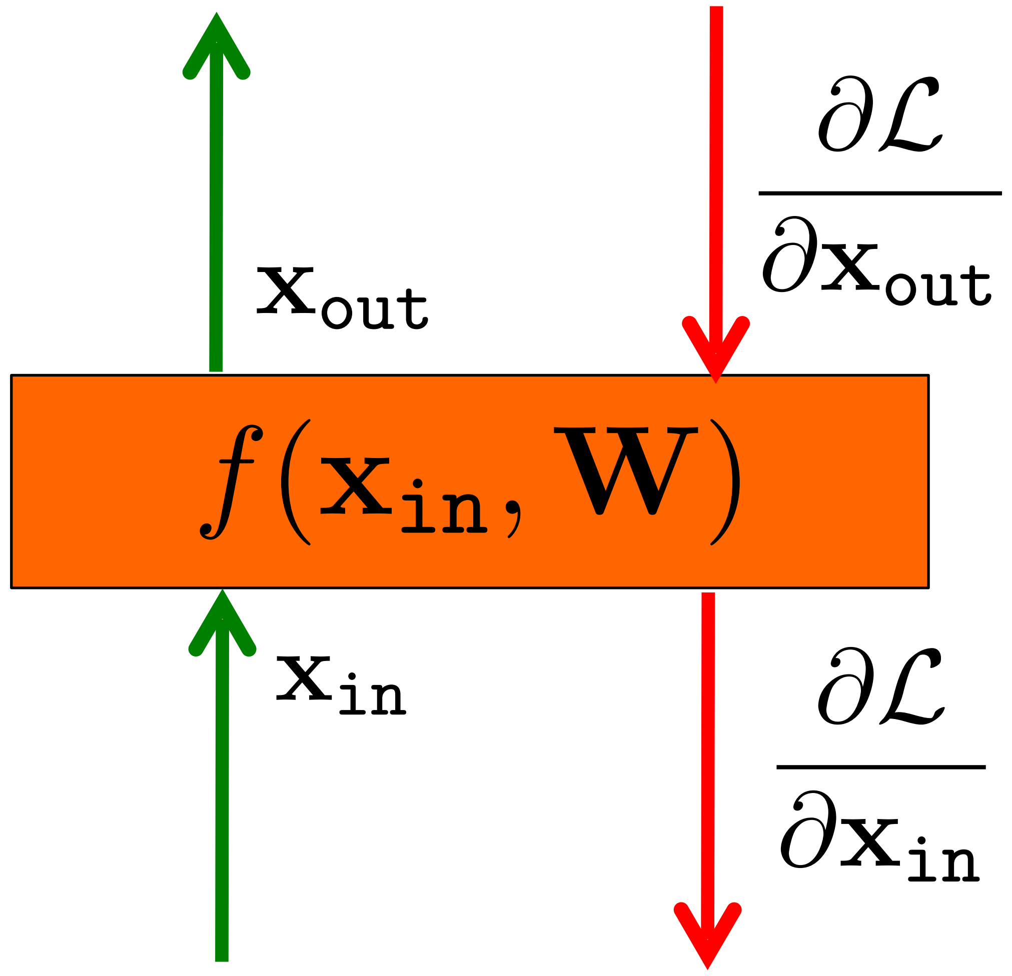
- Backwards pass: compute loss derivatives iteratively from top to bottom:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{x}^{(l-1)}}$$

- Compute gradients w.r.t. weights, and update weights:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \cdot \frac{\partial f^{(l)}(\mathbf{x}^{(l-1)}, \mathbf{W}^{(l)})}{\partial \mathbf{W}^{(l)}}$$





Linear Module

- Forward propagation: $\mathbf{x}_{\mathtt{out}} = f(\mathbf{x}_{\mathtt{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\mathtt{in}}$

With W being a matrix of size $|x_{\text{out}}| \times |x_{\text{in}}|$

- Backprop to input:

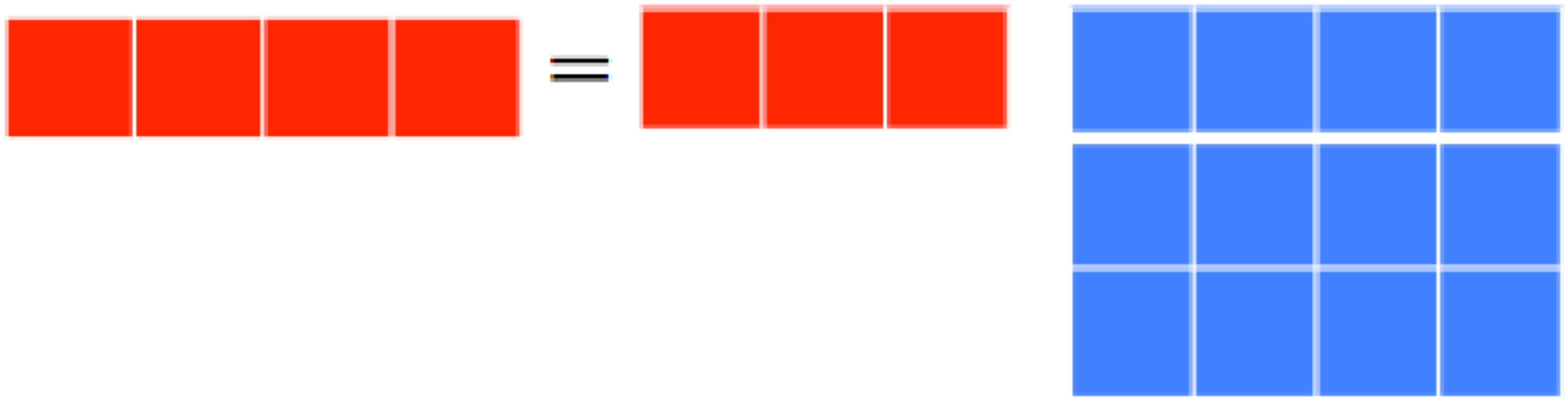
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}}} \cdot \frac{\partial f(\mathbf{x}_{\mathtt{in}}, \mathbf{W})}{\partial \mathbf{x}_{\mathtt{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}}} \cdot \frac{\partial \mathbf{x}_{\mathtt{out}}}{\partial \mathbf{x}_{\mathtt{in}}}$$

If we look at the j component of output x_{out}, with respect to the i component of the input, x_{in}:

$$\frac{\partial \mathbf{x}_{\mathtt{out}_i}}{\partial \mathbf{x}_{\mathtt{in}_j}} = \mathbf{W}_{ij} \longrightarrow \frac{\partial f(\mathbf{x}_{\mathtt{in}}, \mathbf{W})}{\partial \mathbf{x}_{\mathtt{in}}} = \mathbf{W}$$

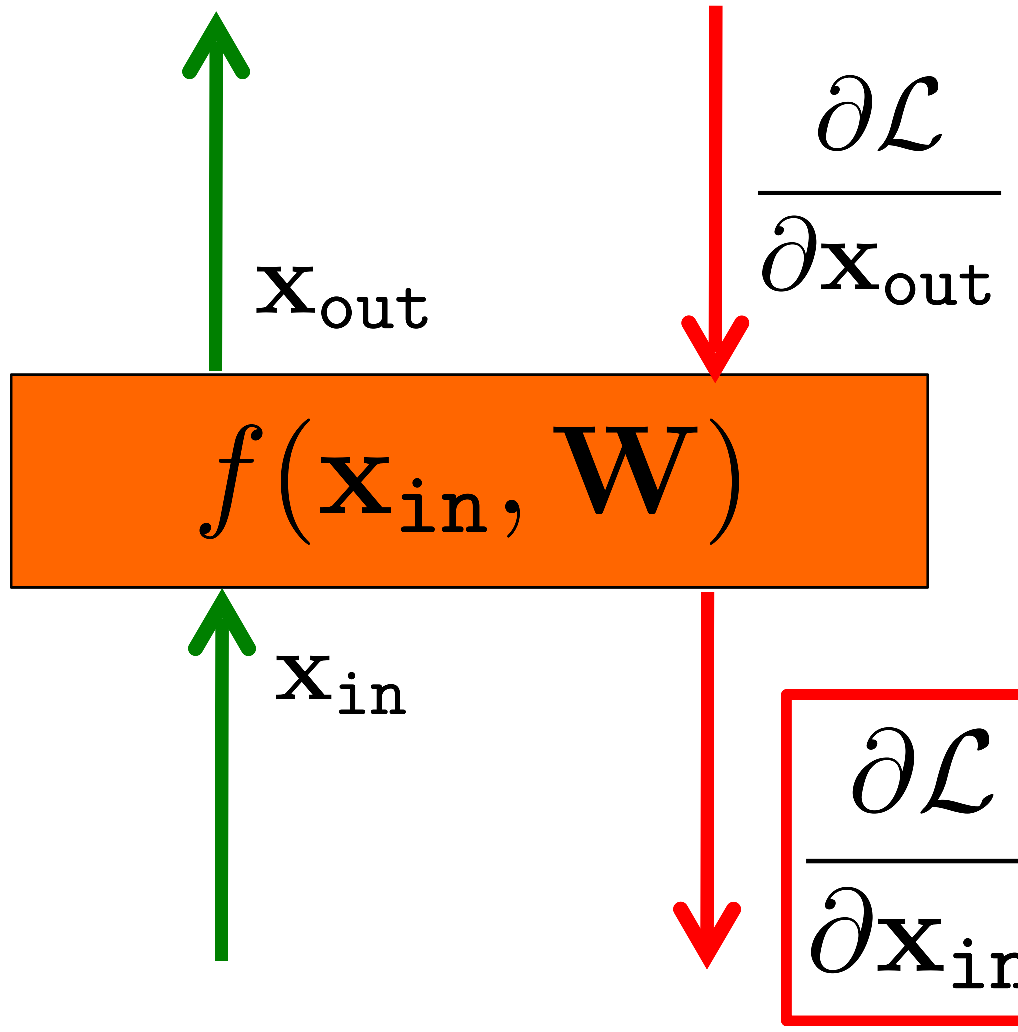
Therefore:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}}} \cdot \mathbf{W}}$$

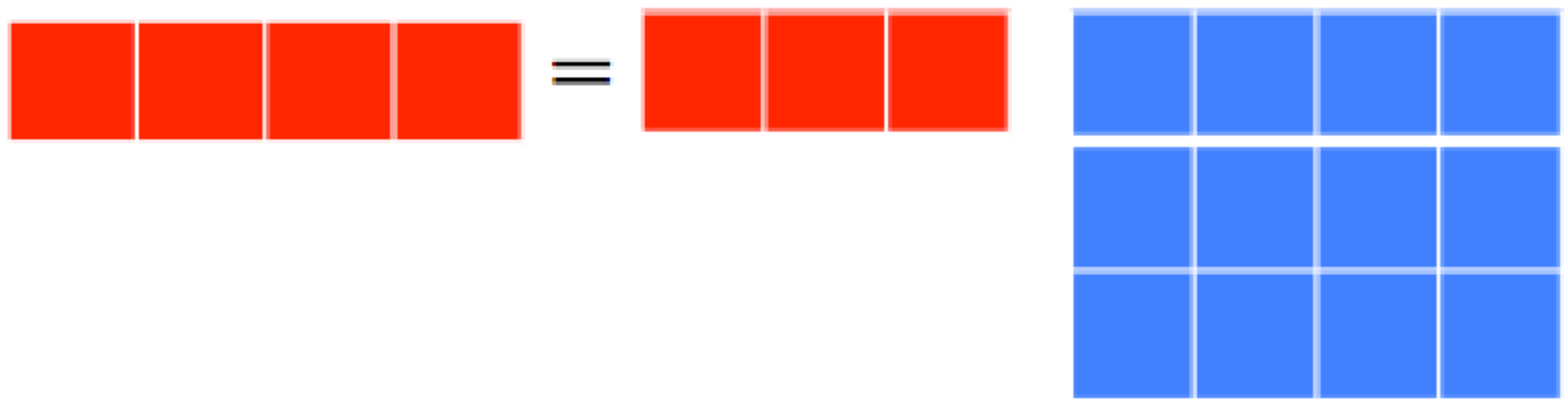


Linear Module

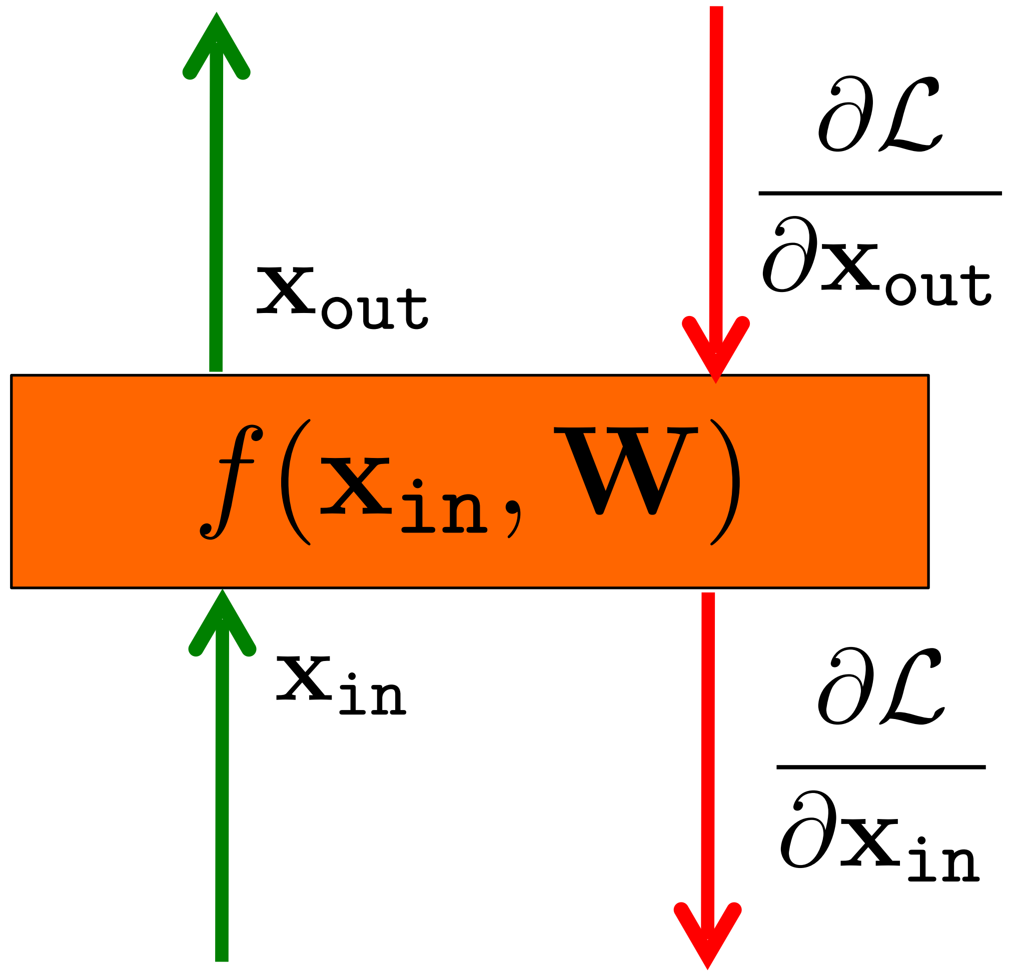
- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to input:



$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \mathbf{W}}$$



Now let's see how we use the set of outputs to compute the weights update equation (backprop to the weights).



Linear Module

- Forward propagation: $\mathbf{x}_{\text{out}} = f(\mathbf{x}_{\text{in}}, \mathbf{W}) = \mathbf{W}\mathbf{x}_{\text{in}}$
- Backprop to weights:

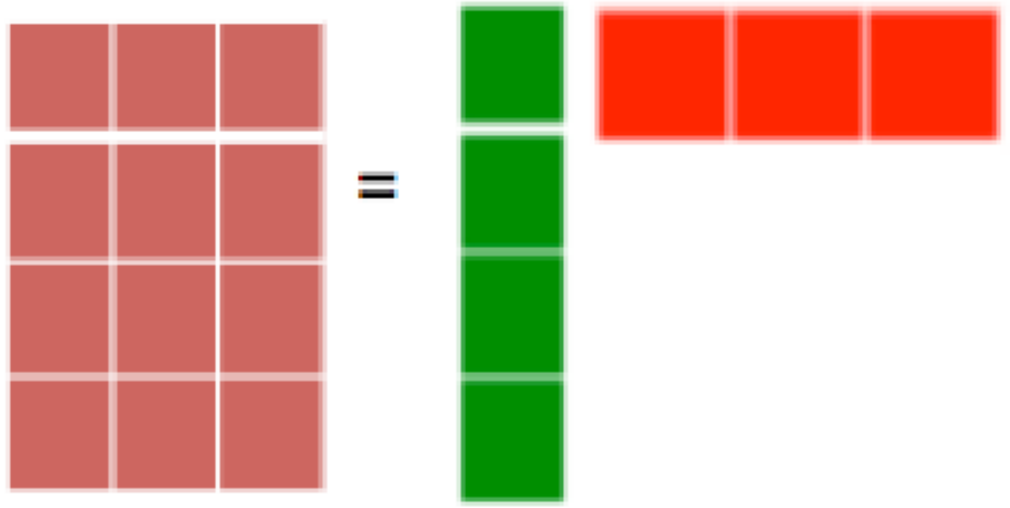
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial f(\mathbf{x}_{\text{in}}, \mathbf{W})}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}} \cdot \frac{\partial \mathbf{x}_{\text{out}}}{\partial \mathbf{W}}$$

If we look at how the parameter W_{ij} changes the cost, only the i component of the output will change, therefore:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}_i}} \cdot \frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{W}_{ij}} \underset{\uparrow}{=} \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}_i}} \cdot \mathbf{x}_{\text{in}_j}$$

$$\frac{\partial \mathbf{x}_{\text{out}_i}}{\partial \mathbf{W}_{ij}} = \mathbf{x}_{\text{in}_j}$$

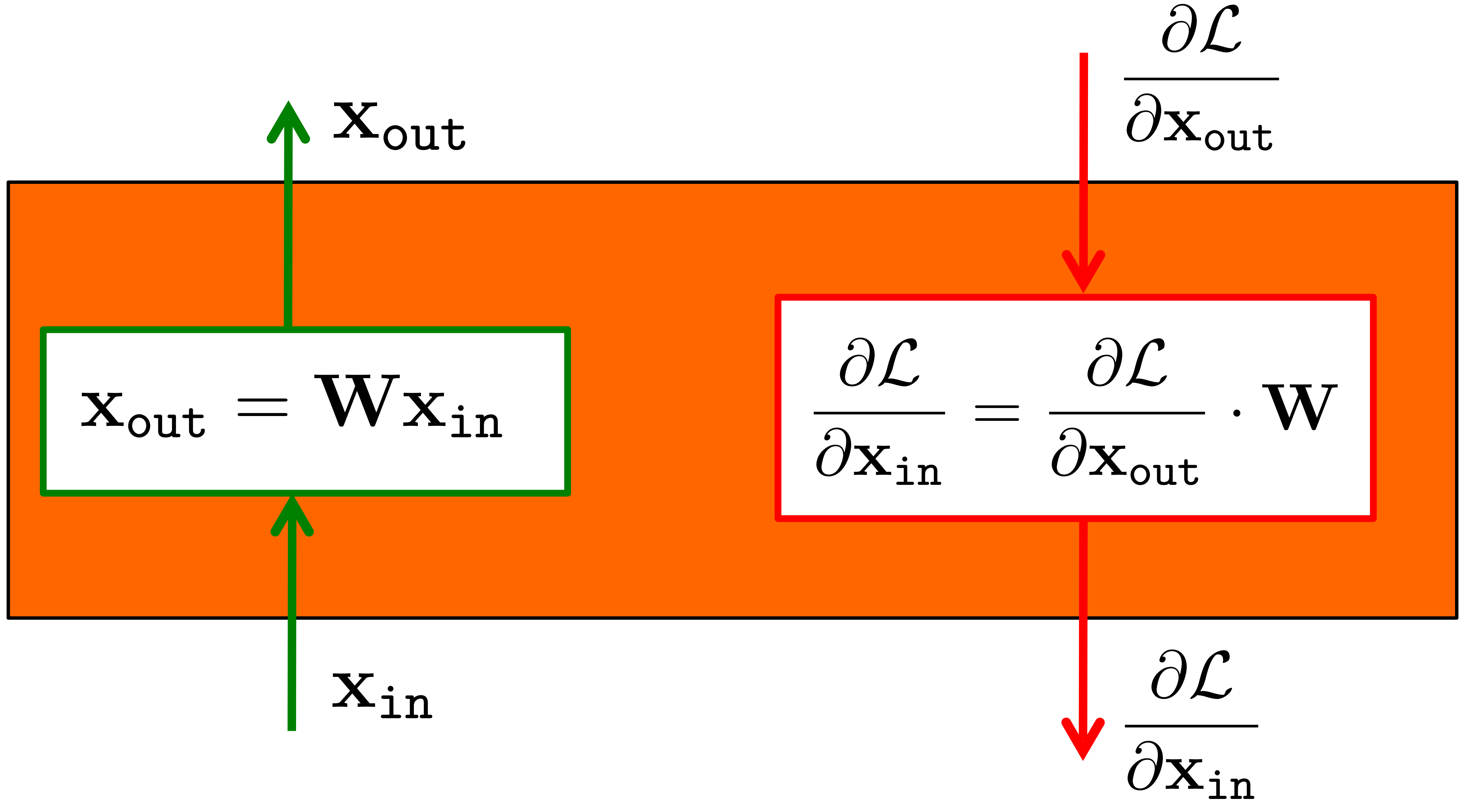
$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{x}_{\text{in}} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\text{out}}}}$$



And now we can update the weights (by summing over all the training examples):

$$\boxed{\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}}\right)^{T} \quad \text{(sum over all training examples to get J)}}$$

Linear Module



$$\mathbf{x}_{\texttt{out}} = \mathbf{W}\mathbf{x}_{\texttt{in}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{in}}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}}} \cdot \mathbf{W}$$

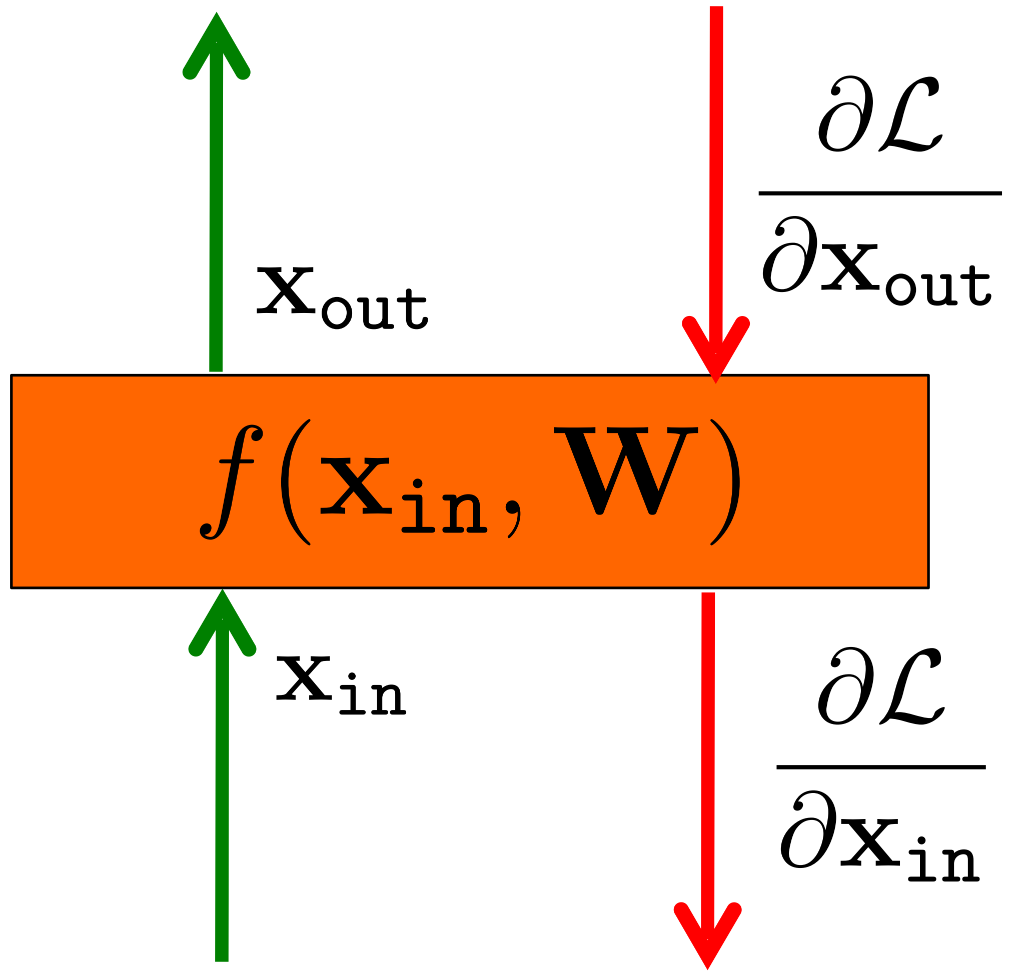
$$\longrightarrow \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \mathbf{x}_{\texttt{in}} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}}}}$$

Weight updates:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

$$J = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i) \longrightarrow \frac{\partial J}{\partial \mathbf{W}} = \sum_{i=1}^{N} \mathbf{x}_{\texttt{in}} \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}}} \bigg|_{\mathbf{x}_i, \mathbf{y}_i}$$

Sum over N training pairs



Pointwise function

- Forward propagation:

$$\mathbf{x}_{\mathtt{out}_i} = h(\mathbf{x}_{\mathtt{in}_i} + b_i)$$

h = an arbitrary function, b_i is a bias term.

- Backprop to input: $$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{in}_i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}_i}} \cdot \frac{\partial \mathbf{x}_{\mathtt{out}_i}}{\partial \mathbf{x}_{\mathtt{in}_i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}_i}} \cdot h'(\mathbf{x}_{\mathtt{in}_i} + b_i)$$

- Backprop to bias: $$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}_i}} \cdot \frac{\partial \mathbf{x}_{\mathtt{out}_i}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\mathtt{out}_i}} \cdot h'(\mathbf{x}_{\mathtt{in}_i} + b_i)$$

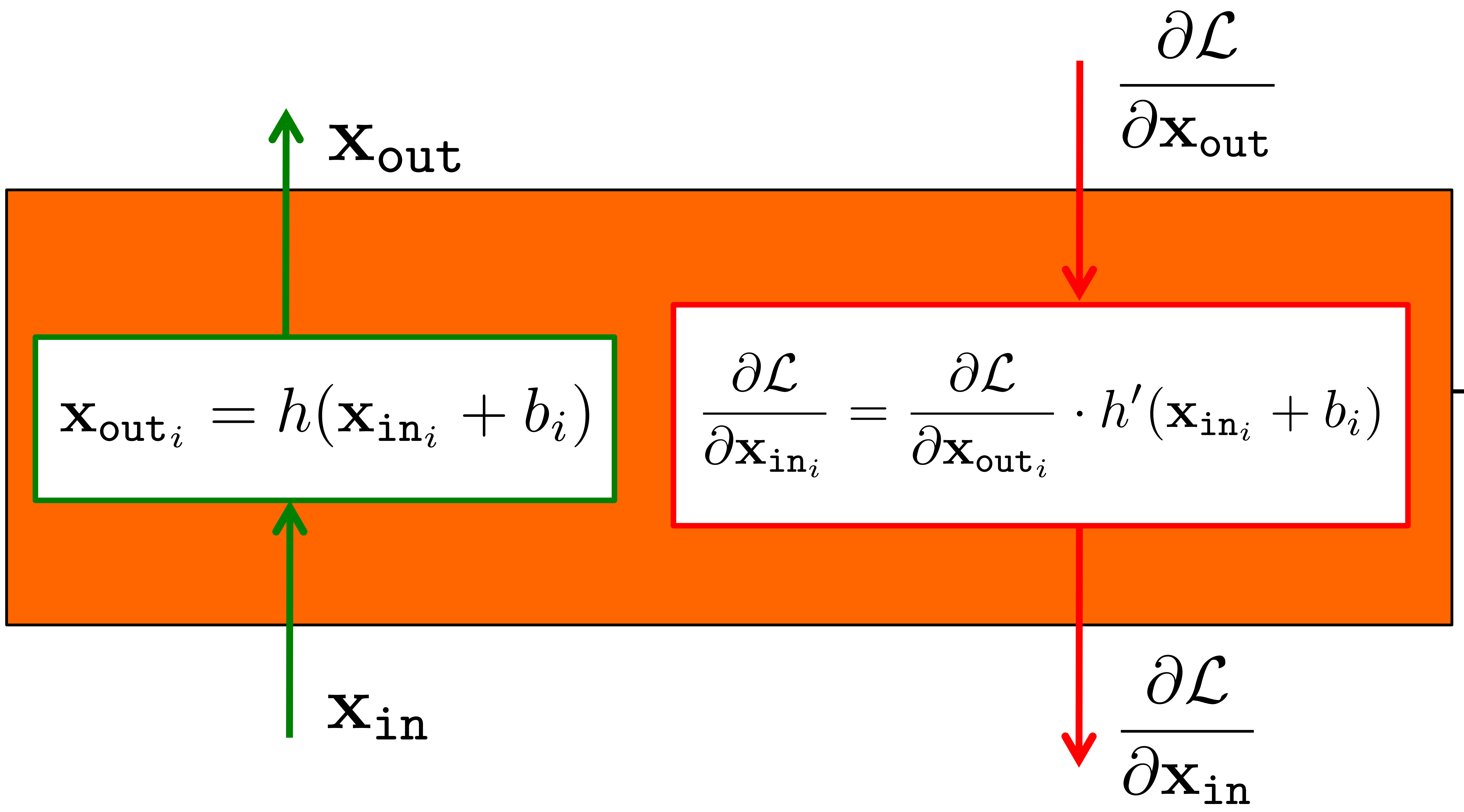
We use this last expression to update the bias.

Some useful derivatives:

For hyperbolic tangent: $\tanh'(x) = 1 - \tanh^2(x)$

For ReLU: $h(x) = \max(0, x), \quad h'(x) = \mathbb{1}(x \geq 0)$

Pointwise function



$$\mathbf{x}_{\texttt{out}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}}}$$

$$\mathbf{x}_{\texttt{out}_i} = h(\mathbf{x}_{\texttt{in}_i} + b_i)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{in}_i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}_i}} \cdot h'(\mathbf{x}_{\texttt{in}_i} + b_i)$$

$$\mathbf{x}_{\texttt{in}}$$

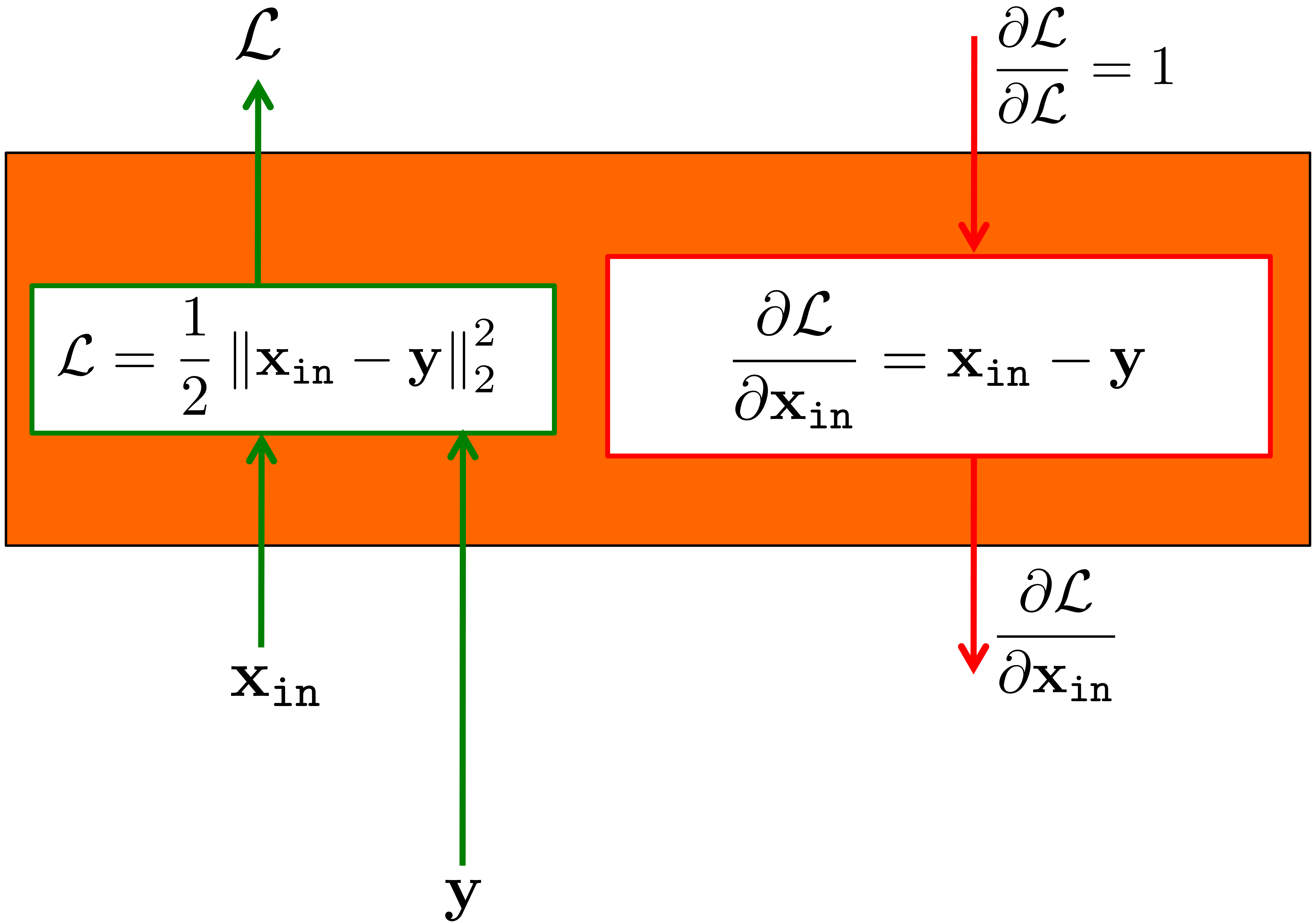
$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{in}}}$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{\texttt{out}_i}} \cdot h'(\mathbf{x}_{\texttt{in}_i} + b_i)$$

Bias updates:

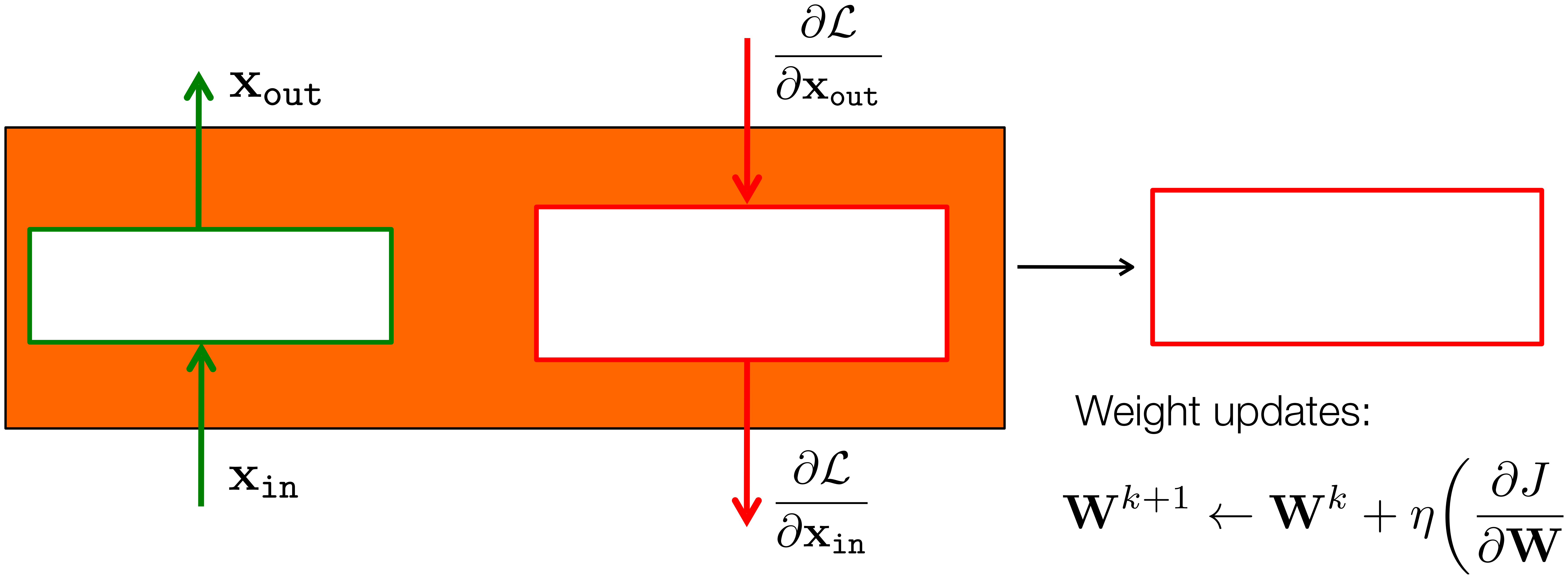
$$b_i^{k+1} \leftarrow b_i^k + \eta \frac{\partial J}{\partial b_i}$$

Euclidean cost module



Homework: Convolution Module

Assume the input x_{in} and output x_{out} are 1D signals of the same length N.
The convolution kernel is w_i, and has length $M < N$



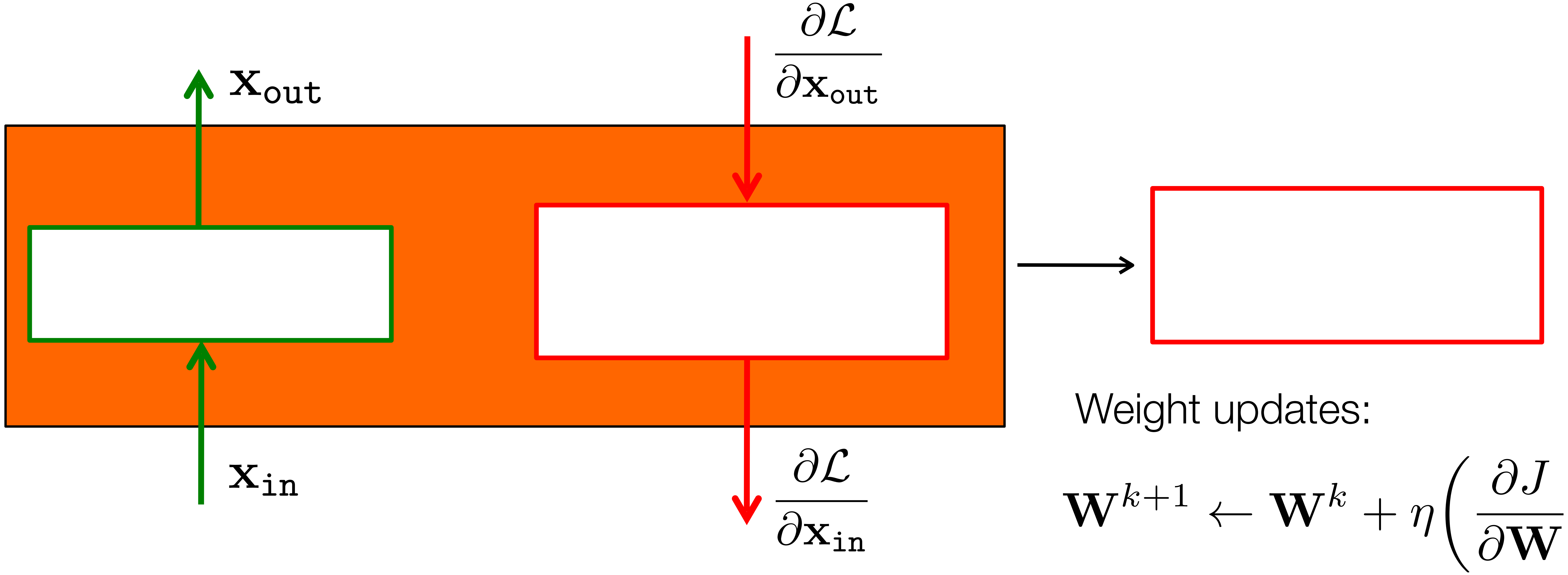
Weight updates:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

Homework: max pooling module

Assume the input x_{in} and output x_{out} are 1D signals of different lengths.

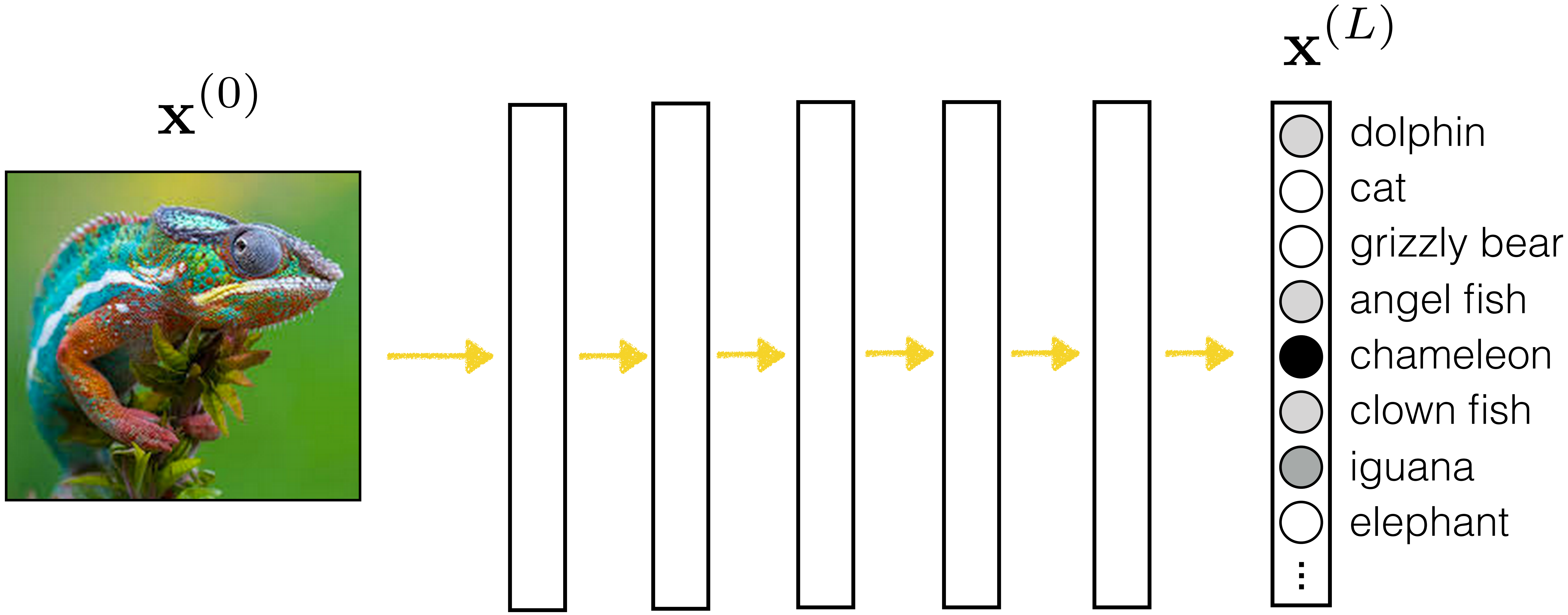


Weight updates:

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^{k} + \eta \left(\frac{\partial J}{\partial \mathbf{W}} \right)^{T}$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

Unit visualization via backprop



$\mathbf{x}^{(0)}$

$\mathbf{x}^{(L)}$

dolphin
cat
grizzly bear
angel fish
chameleon
clown fish
iguana
elephant
⋮

$$\frac{\partial x_j^{(L)}}{\partial \mathbf{x}^{(0)}} = \frac{\partial x_j^{(L)}}{\partial \mathbf{x}^{(L-1)}} \cdot \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{x}^{(L-2)}} \cdots \frac{\partial \mathbf{x}^{(2)}}{\partial \mathbf{x}^{(1)}} \cdot \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}^{(0)}}$$

↖ How much the “chameleon” score is increased or decreased by changing the image pixels.

Unit visualization via backprop

$$\arg\max_{\mathbf{x}^{(0)}} x_j^{(L)}$$

$$\mathbf{x}^{(0)^{k+1}} \leftarrow \mathbf{x}^{(0)^{k}} + \eta \frac{\partial x_j^{(L)}}{\partial \mathbf{x}^{(0)}}$$

Unit visualization via backprop

Make an image that maximizes the "cat" output neuron:

$$\arg\max_{\mathbf{x}^{(0)}} x_j^{(L)} + \lambda R(\mathbf{x}^{(0)})$$

$$\mathbf{x}^{(0)^{k+1}} \leftarrow \mathbf{x}^{(0)^{k}} + \eta \frac{\partial(x_j^{(L)} + \lambda R(\mathbf{x}^{(0)}))}{\partial \mathbf{x}^{(0)}}$$

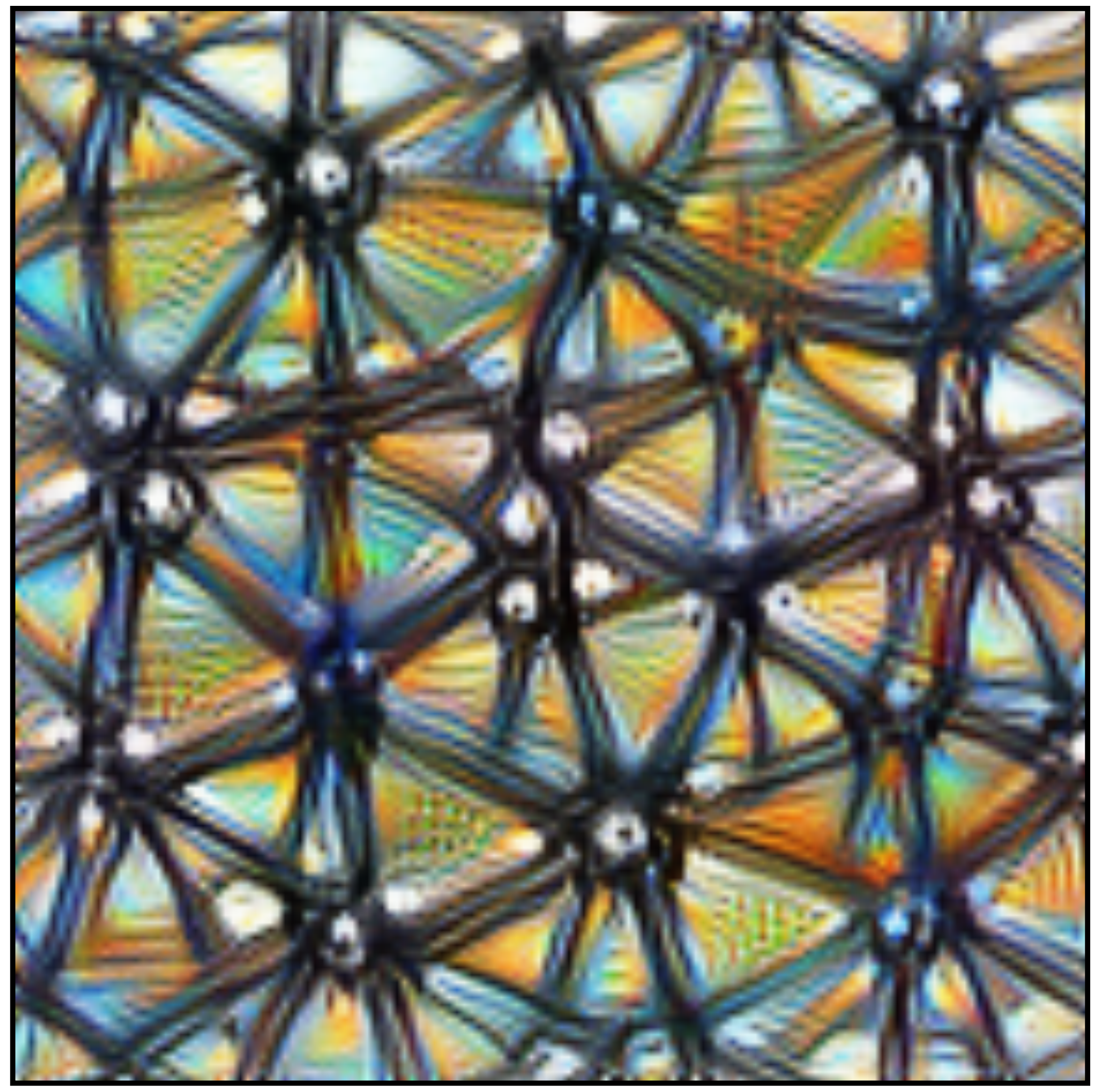
[https://distill.pub/2017/feature-visualization/]

Unit visualization via backprop

$$\arg\max_{\mathbf{x}^{(0)}} x_j^{(L)} + \lambda R(\mathbf{x}^{(0)})$$

$$\mathbf{x}^{(0)^{k+1}} \leftarrow \mathbf{x}^{(0)^{k}} + \eta \frac{\partial(x_j^{(L)} + \lambda R(\mathbf{x}^{(0)}))}{\partial \mathbf{x}^{(0)}}$$

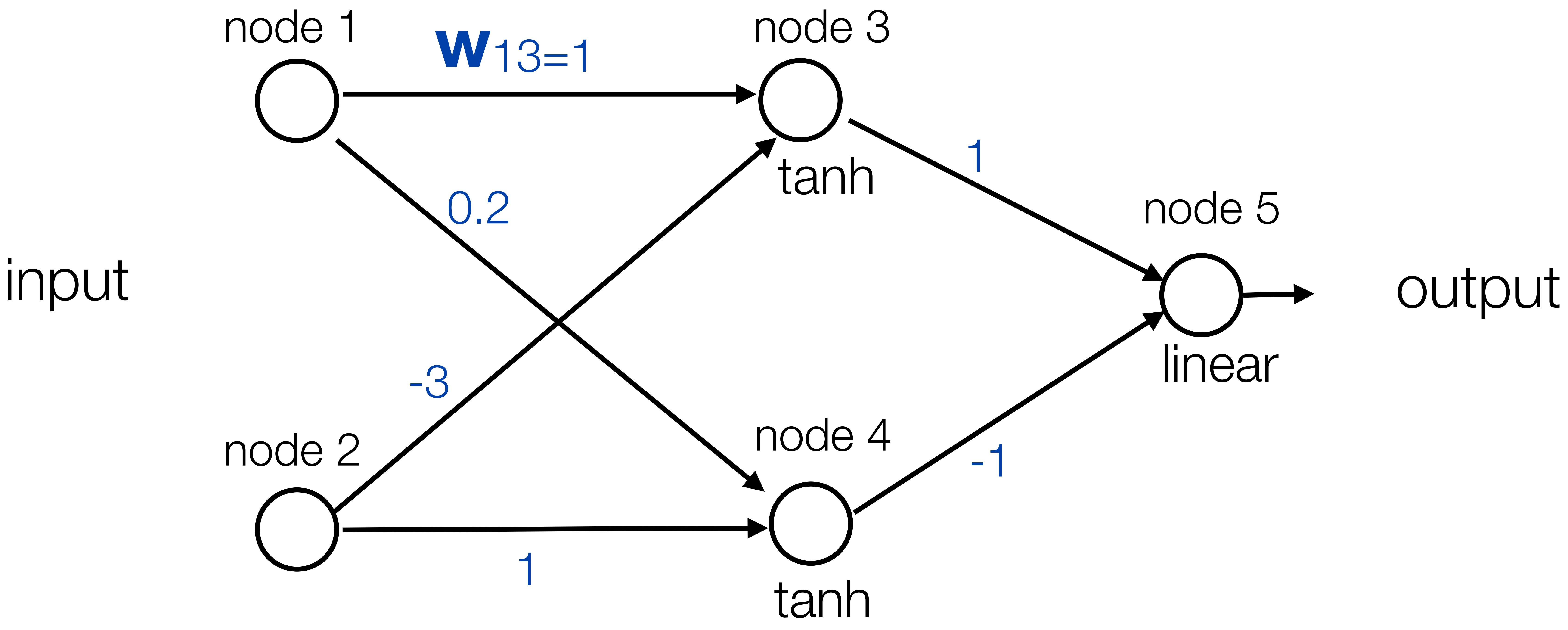
Make an image that maximizes the value of a random neuron in the middle of the network:



[https://distill.pub/2017/feature-visualization/]

“Deep dream” [https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html]

Backpropagation example



Learning rate $\eta = -0.2$ (because we used positive increments)

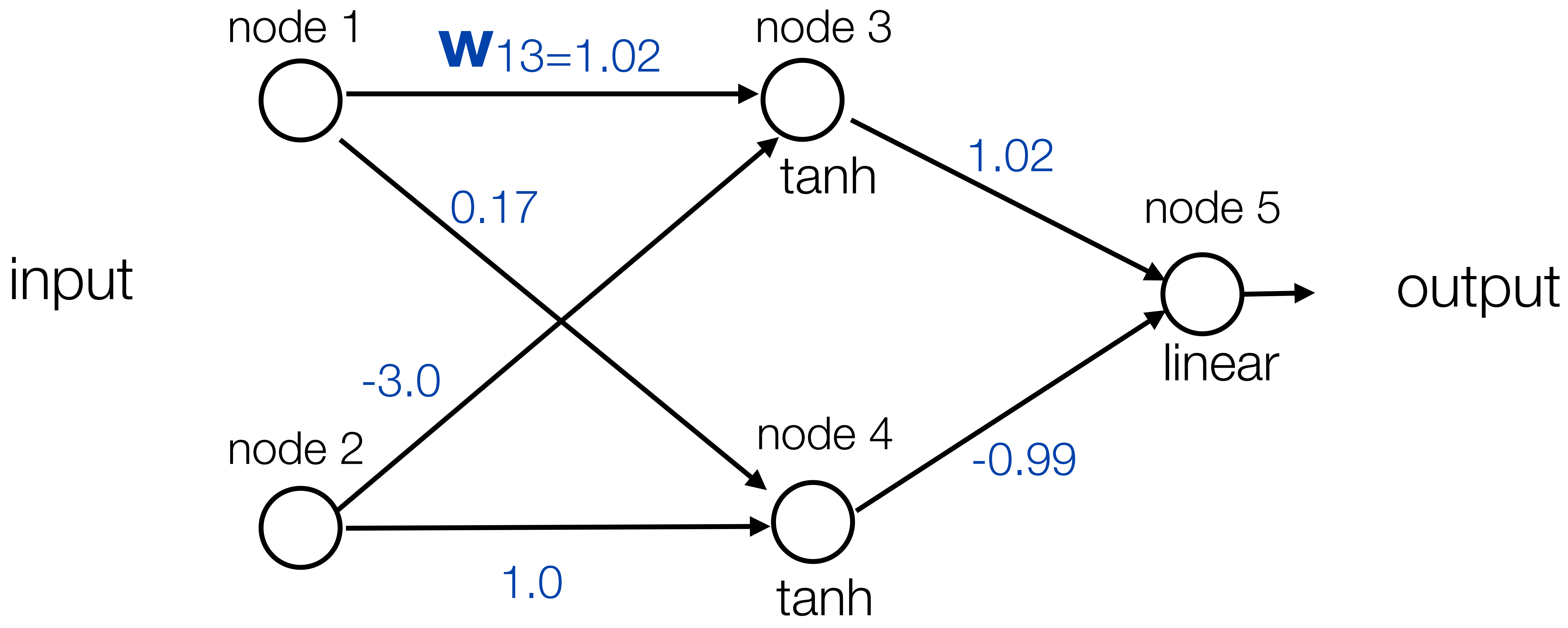
Euclidean loss

Training data:

input		desired output
node 1	node 2	node 5
1.0	0.1	0.5

Exercise: run one iteration of back propagation

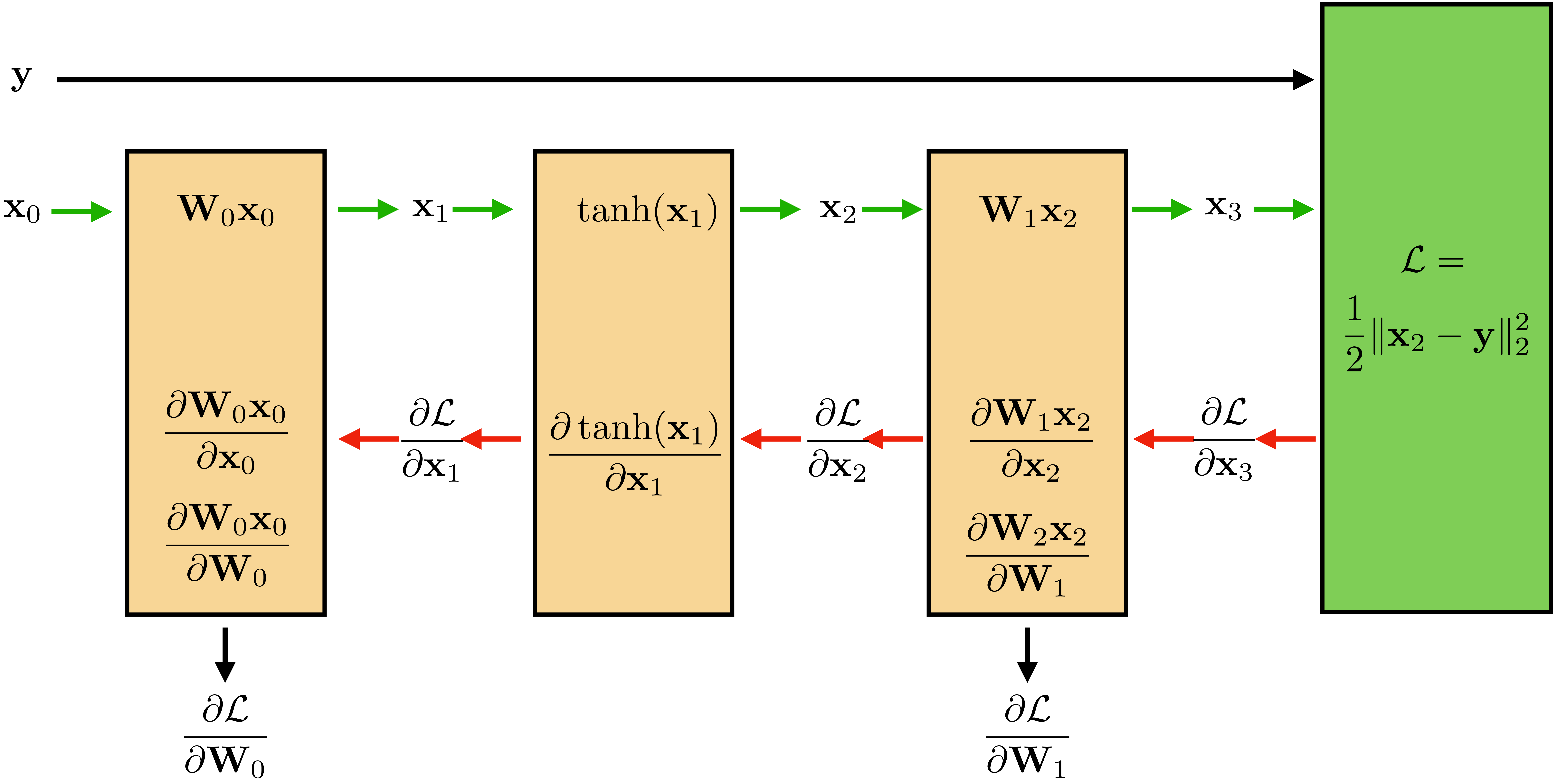
Backpropagation example



After one iteration (rounding to two digits)

Step by step solution

First, let's rewrite the network using the modular block notation:



$\mathbf{y}$

$\mathbf{x}_0$

$\mathbf{W}_0\mathbf{x}_0$

$\mathbf{x}_1$

$\tanh(\mathbf{x}_1)$

$\mathbf{x}_2$

$\mathbf{W}_1\mathbf{x}_2$

$\mathbf{x}_3$

$\mathcal{L} = \frac{1}{2}\|\mathbf{x}_2 - \mathbf{y}\|_2^2$

$\frac{\partial \mathbf{W}_0\mathbf{x}_0}{\partial \mathbf{x}_0}$

$\frac{\partial \mathbf{W}_0\mathbf{x}_0}{\partial \mathbf{W}_0}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}$

$\frac{\partial \tanh(\mathbf{x}_1)}{\partial \mathbf{x}_1}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}$

$\frac{\partial \mathbf{W}_1\mathbf{x}_2}{\partial \mathbf{x}_2}$

$\frac{\partial \mathbf{W}_2\mathbf{x}_2}{\partial \mathbf{W}_1}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0}$

$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}$

We need to compute all these terms simply so we can find the weight updates at the bottom.

Our goal is to perform the following two updates:

$$\mathbf{W}_0^{k+1} = \mathbf{W}_0^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} \right)^T$$

$$\mathbf{W}_1^{k+1} = \mathbf{W}_1^k + \eta \left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} \right)^T$$

where W^k are the weights at some iteration k of gradient descent given by the first slide:

$$\mathbf{W}_0^k = \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} \qquad \mathbf{W}_1^k = \begin{pmatrix} 1 & -1 \end{pmatrix}$$

First we compute the derivative of the loss with respect to the output:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}} = \mathbf{x}_3 - \mathbf{y}$$

Now, by the chain rule, we can derive equations, working *backwards*, for each remaining term we need:

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} = \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}}\mathbf{W}_1$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}\frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}\frac{\partial \tanh(\mathbf{x}_1)}{\partial \mathbf{x}_1} = \boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}}(1 - \tanh^2(\mathbf{x}_1))$$

ending up with our two gradients needed for the weight update:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}\frac{\partial \mathbf{x}_1}{\partial \mathbf{W}_0} = \mathbf{x}_0\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\frac{\partial \mathbf{x}_3}{\partial \mathbf{W}_1} = \mathbf{x}_2\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}}$$

Notice the ordering of the two terms being multiplied here. The notation hides the details but you can write out all the indices to see that this is the correct ordering — or just check that the dimensions work out.

The values for input vector x_0 and target y are also given by the first slide:

$$\mathbf{x}_0 = \begin{pmatrix} 1.0 \\ 0.1 \end{pmatrix} \qquad \mathbf{y} = 0.5$$

Finally, we simply plug these values into our equations and compute the numerical updates:

Forward pass:

$$\mathbf{x}_1 = \mathbf{W}_0 \mathbf{x}_0 = \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

$$\mathbf{x}_2 = \tanh(\mathbf{x}_1) = \begin{pmatrix} 0.604 \\ 0.291 \end{pmatrix}$$

$$\mathbf{x}_3 = \mathbf{W}_1 \mathbf{x}_2 = \begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 0.604 \\ 0.291 \end{pmatrix} = 0.313$$

$$\mathcal{L} = \frac{1}{2}(\mathbf{x}_3 - \mathbf{y})^2 = 0.017$$

Backward pass:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} = \mathbf{x}_3 - \mathbf{y} = -0.1869$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_2} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_3}\mathbf{W}_1 = -0.1869\begin{pmatrix}1 & -1\end{pmatrix} = \begin{pmatrix}-0.1869 & 0.1869\end{pmatrix}$$

diagonal matrix because tanh is a pointwise operation

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_2}(1 - \tanh^2(\mathbf{x}_1)) = \begin{pmatrix}-0.1869 & 0.1869\end{pmatrix}\begin{pmatrix}1 - \tanh^2(0.7) & 0 \\ 0 & 1 - \tanh^2(0.3)\end{pmatrix} = \begin{pmatrix}-0.1186 & 0.171\end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0} = \mathbf{x}_0\frac{\partial \mathcal{L}}{\partial \mathbf{x}_1} = \begin{pmatrix}1.0 \\ 0.1\end{pmatrix}\begin{pmatrix}-0.1186 & 0.171\end{pmatrix} = \begin{pmatrix}-0.1186 & 0.171 \\ -0.01186 & 0.0171\end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \mathbf{x}_2\frac{\partial \mathcal{L}}{\partial \mathbf{x}_3} = \begin{pmatrix}0.604 \\ 0.291\end{pmatrix}\begin{pmatrix}-0.1186\end{pmatrix} = \begin{pmatrix}-0.113 \\ -0.054\end{pmatrix}$$

Gradient updates:

$$\begin{aligned}\mathbf{W}_0^{k+1} &= \mathbf{W}_0^k + \eta\left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_0}\right)^T \\ &= \begin{pmatrix} 1 & -3 \\ 0.2 & 1 \end{pmatrix} - 0.2\begin{pmatrix} -0.1186 & 0.171 \\ -0.01186 & 0.0171 \end{pmatrix} \\ &= \begin{pmatrix} 1.02 & -3.0 \\ 0.17 & 1.0 \end{pmatrix}\end{aligned}$$

$$\begin{aligned}\mathbf{W}_1^{k+1} &= \mathbf{W}_1^k + \eta\left(\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1}\right)^T \\ &= \begin{pmatrix} 1 & -1 \end{pmatrix} - 0.2\begin{pmatrix} -0.113 & -0.054 \end{pmatrix} \\ &= \begin{pmatrix} 1.02 & -0.989 \end{pmatrix}\end{aligned}$$