CS5670: Computer Vision

Noah Snavely

Lecture 25: Backprop and convnets

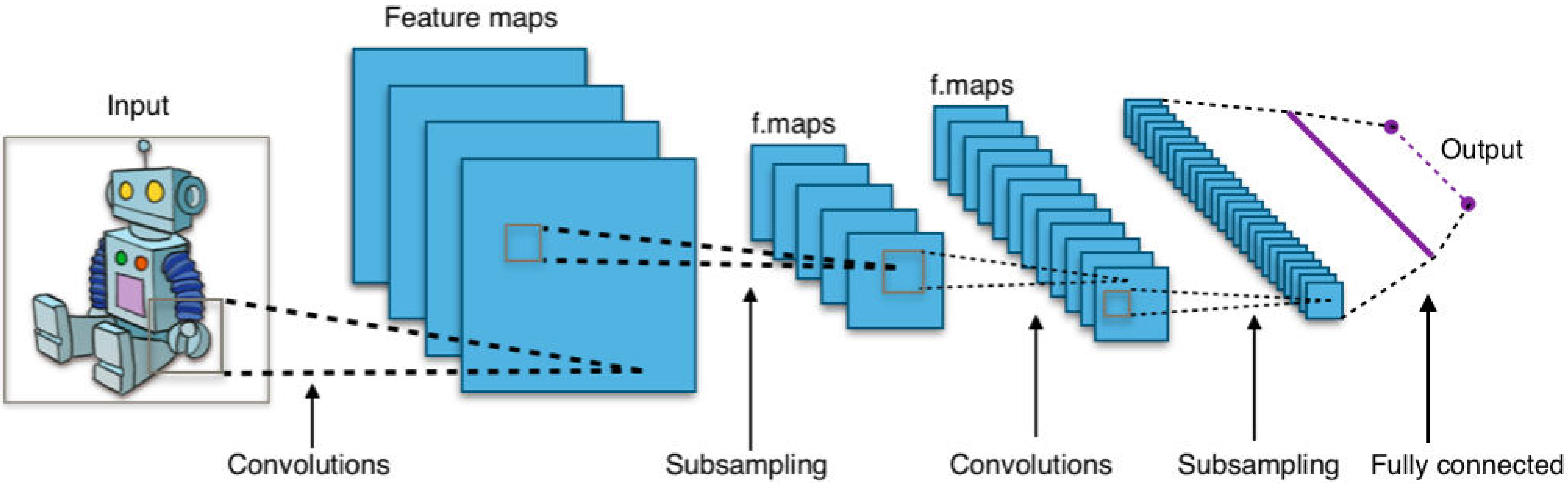
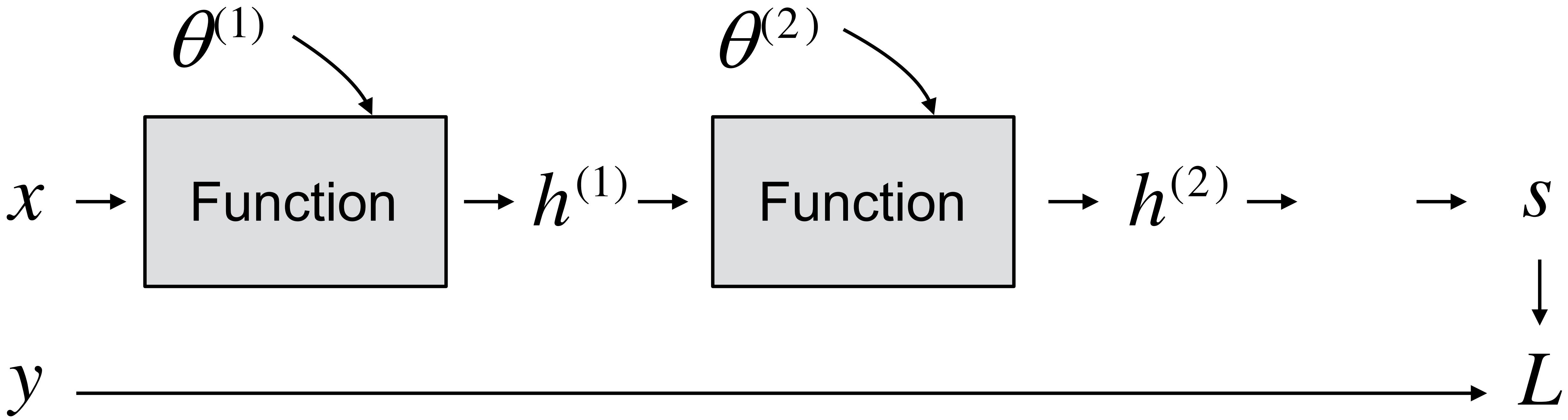


Image credit: Aphex34, [CC BY-SA 4.0 (http://creativecommons.org/licenses/by-sa/4.0)]

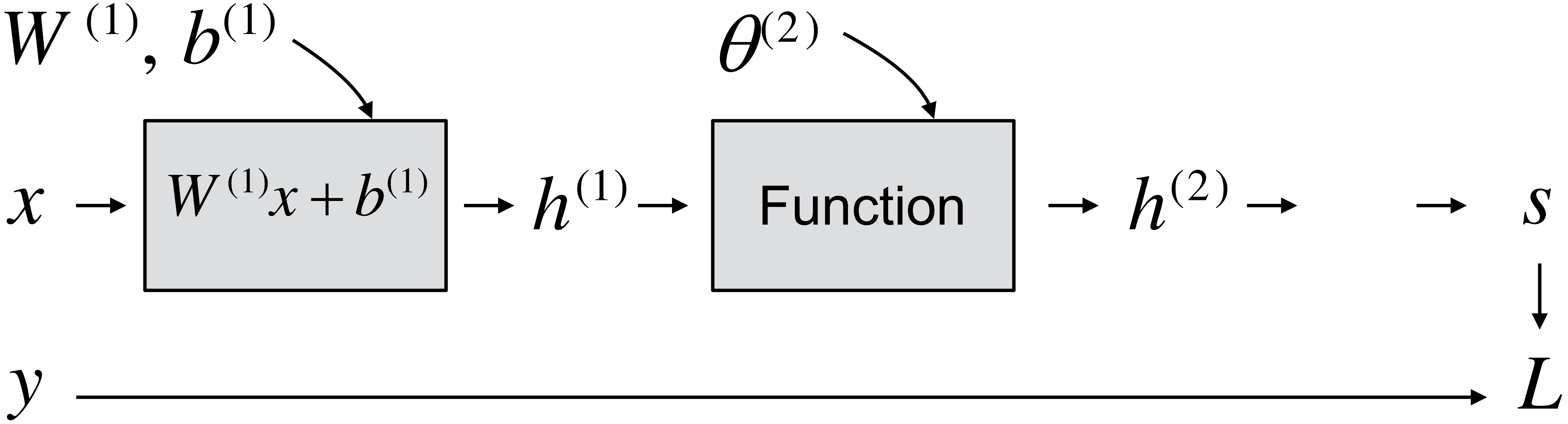
Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/

Review: Setup



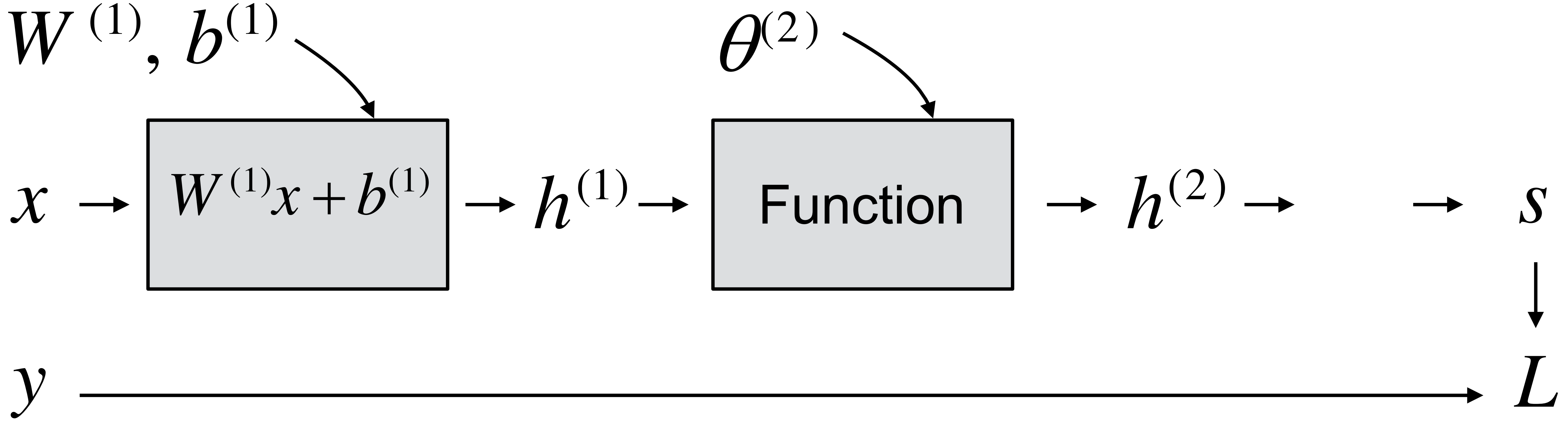
- **Goal:** Find a value for parameters ($\theta^{(1)}, \theta^{(2)}$, …), so that the loss (L) is small

Review: Setup



Toy Example:

Review: Setup



$W^{(1)}, b^{(1)}$

$\theta^{(2)}$

x

$W^{(1)}x + b^{(1)}$

$h^{(1)}$

Function

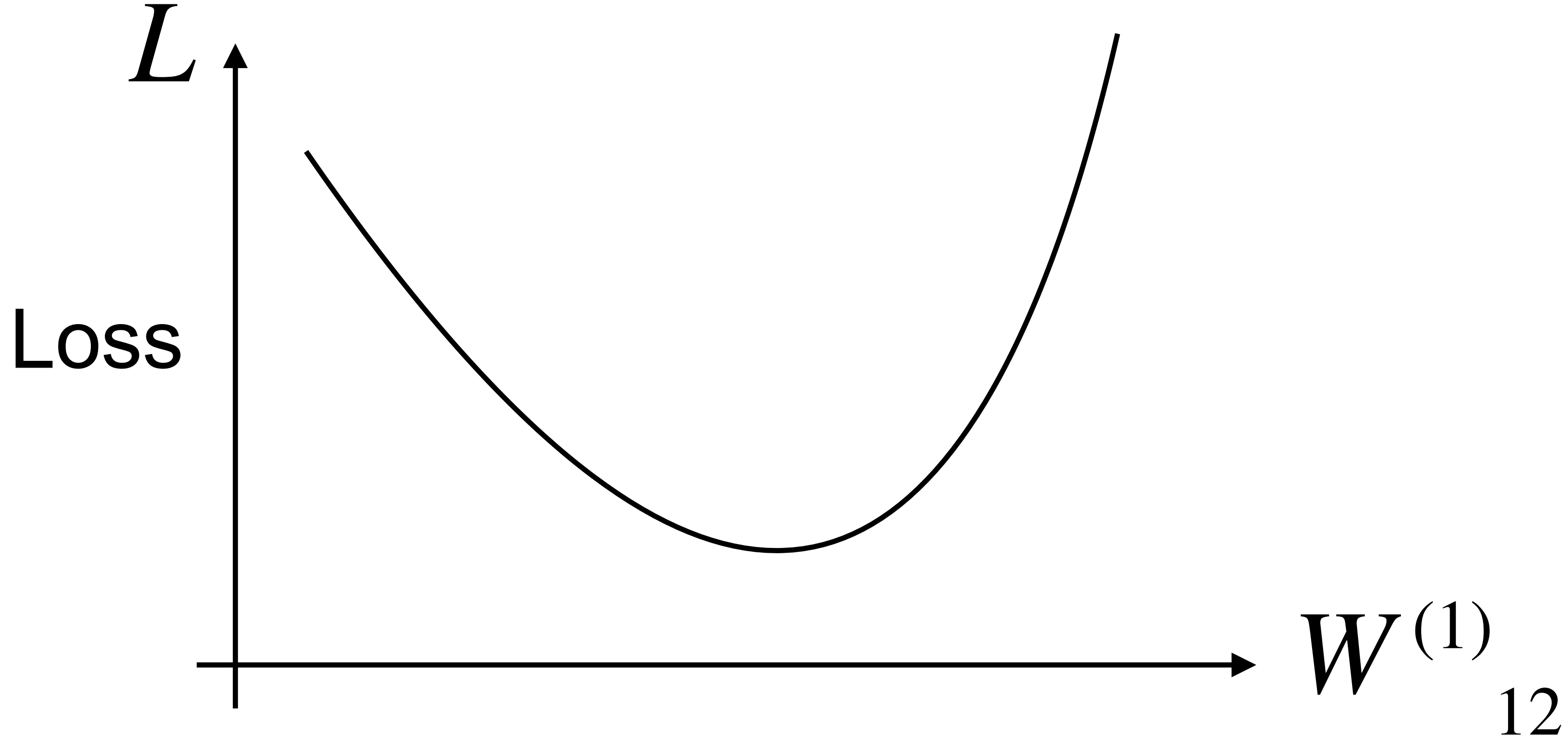
$h^{(2)}$

s

y

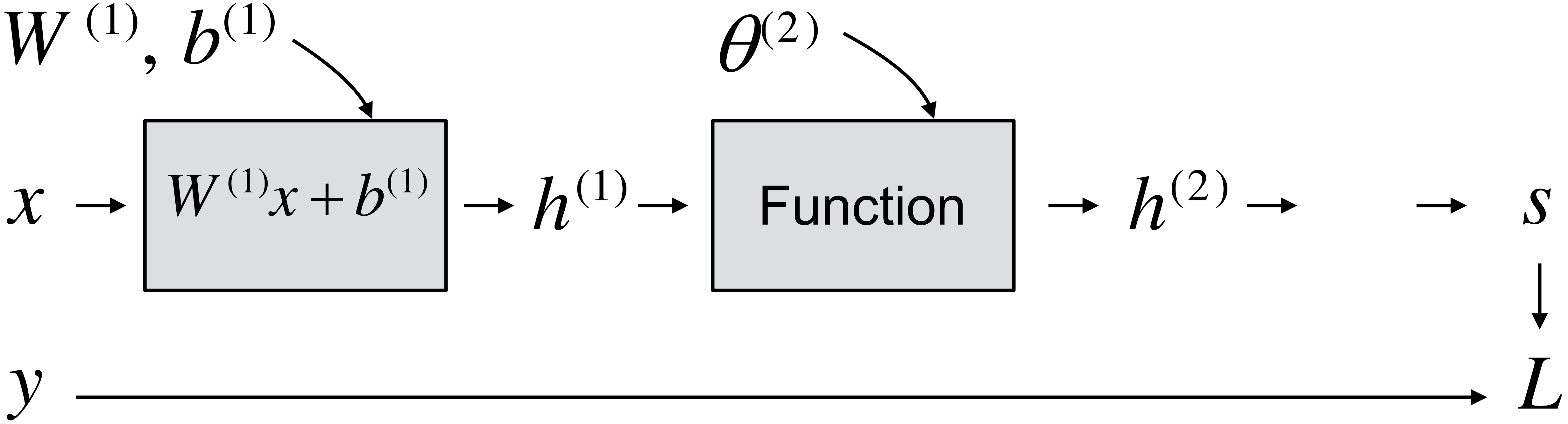
L

Toy Example:

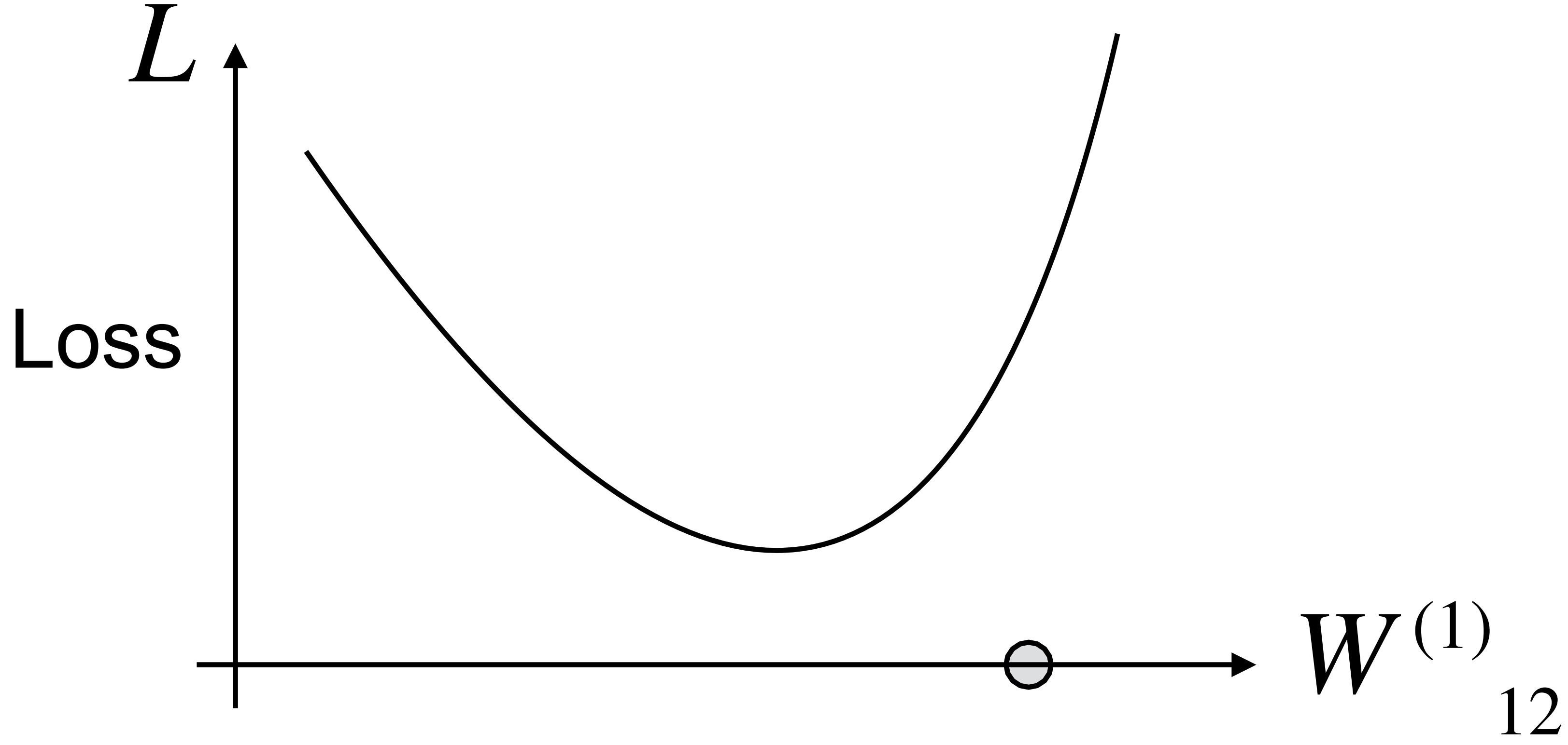


A weight somewhere in the network

Review: Setup

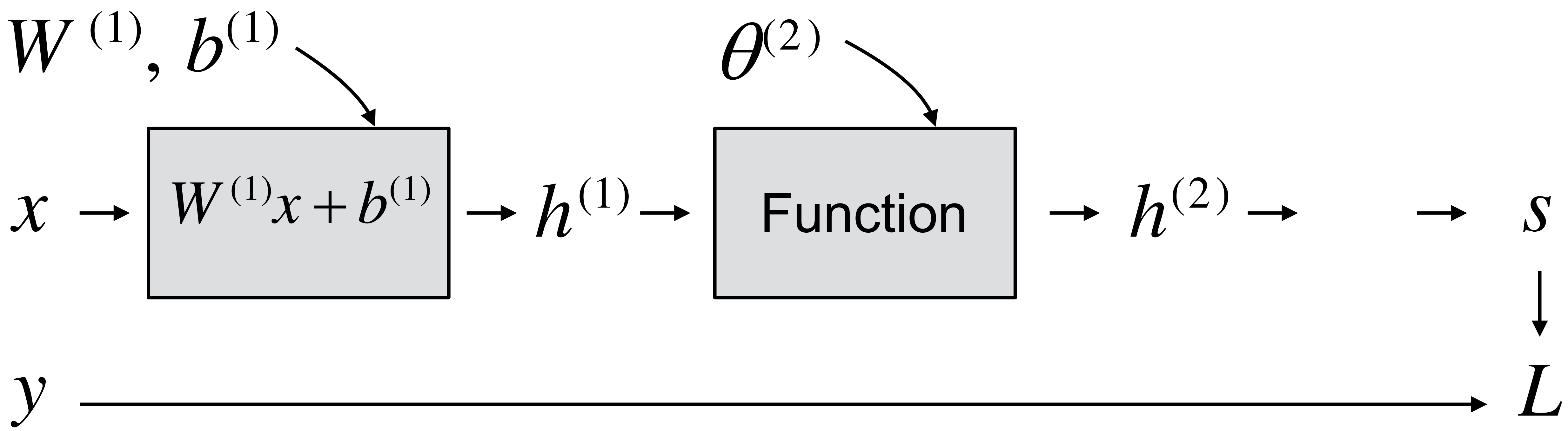


Toy Example:



A weight somewhere in the network

Review: Setup



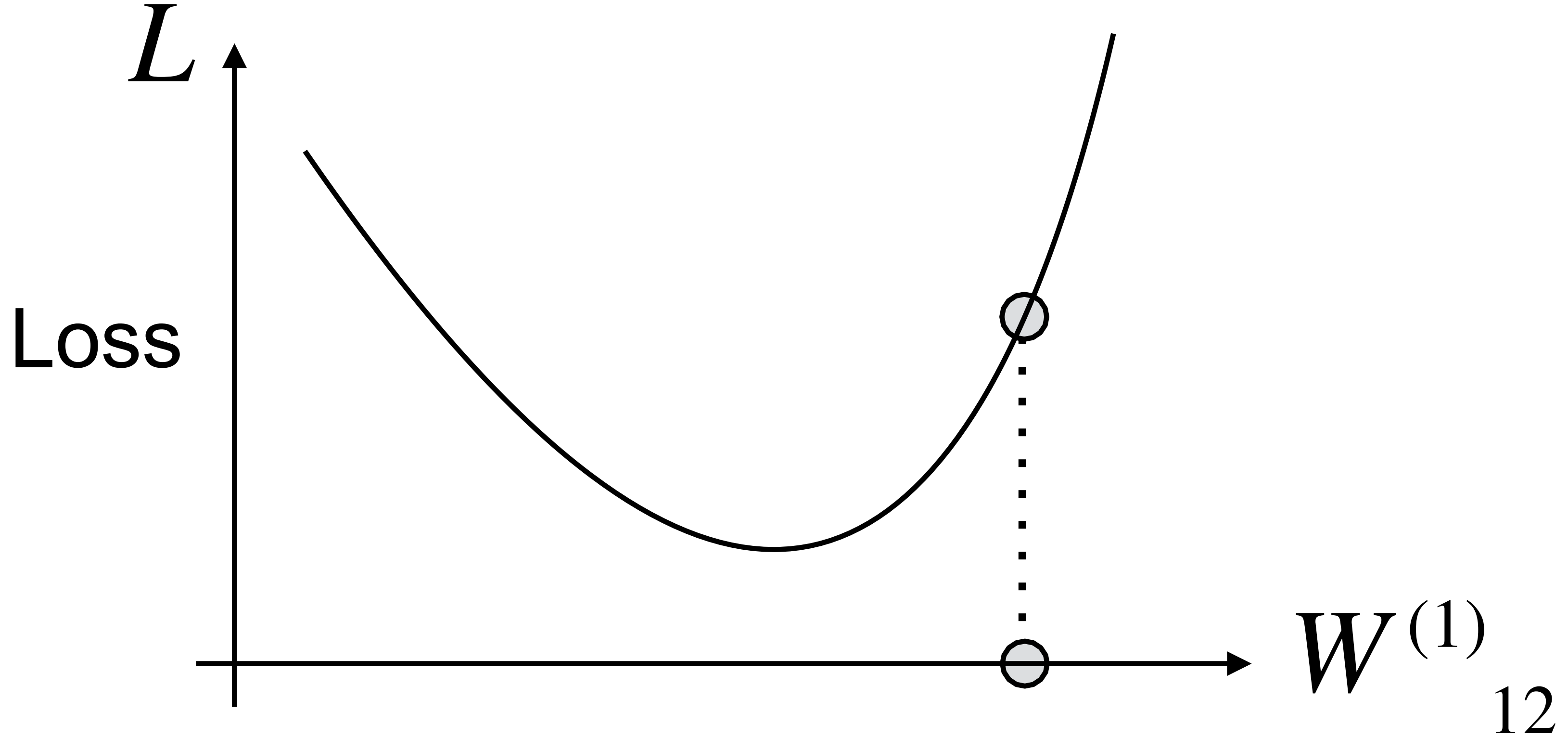
$W^{(1)}, b^{(1)}$

$\theta^{(2)}$

$x \rightarrow \boxed{W^{(1)}x + b^{(1)}} \rightarrow h^{(1)} \rightarrow$ Function $\rightarrow h^{(2)} \rightarrow \quad \rightarrow s$

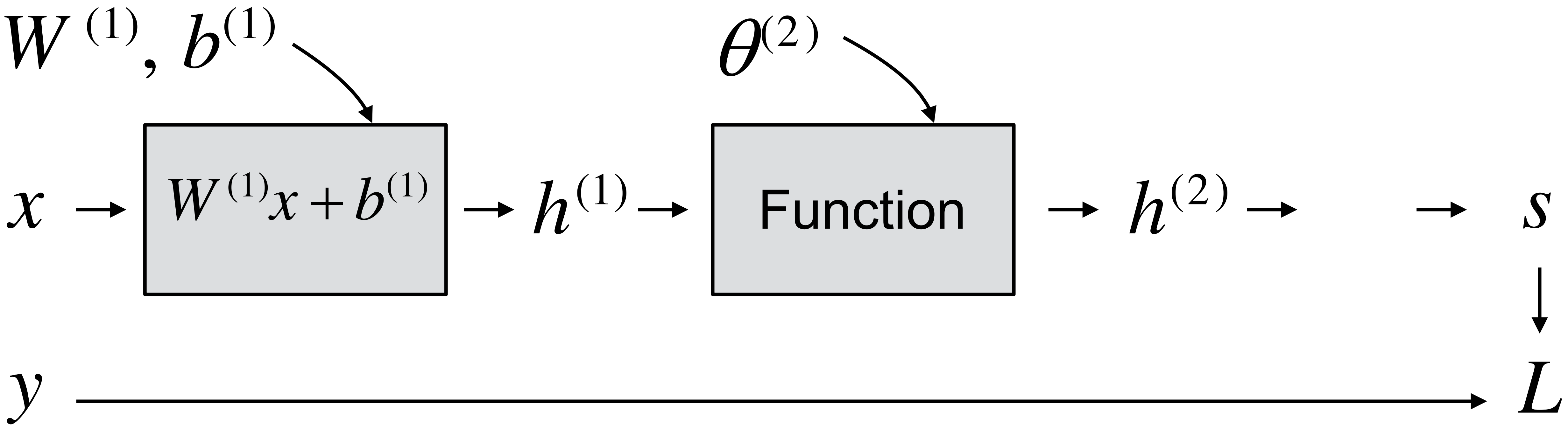
$y \rightarrow L$

Toy Example:

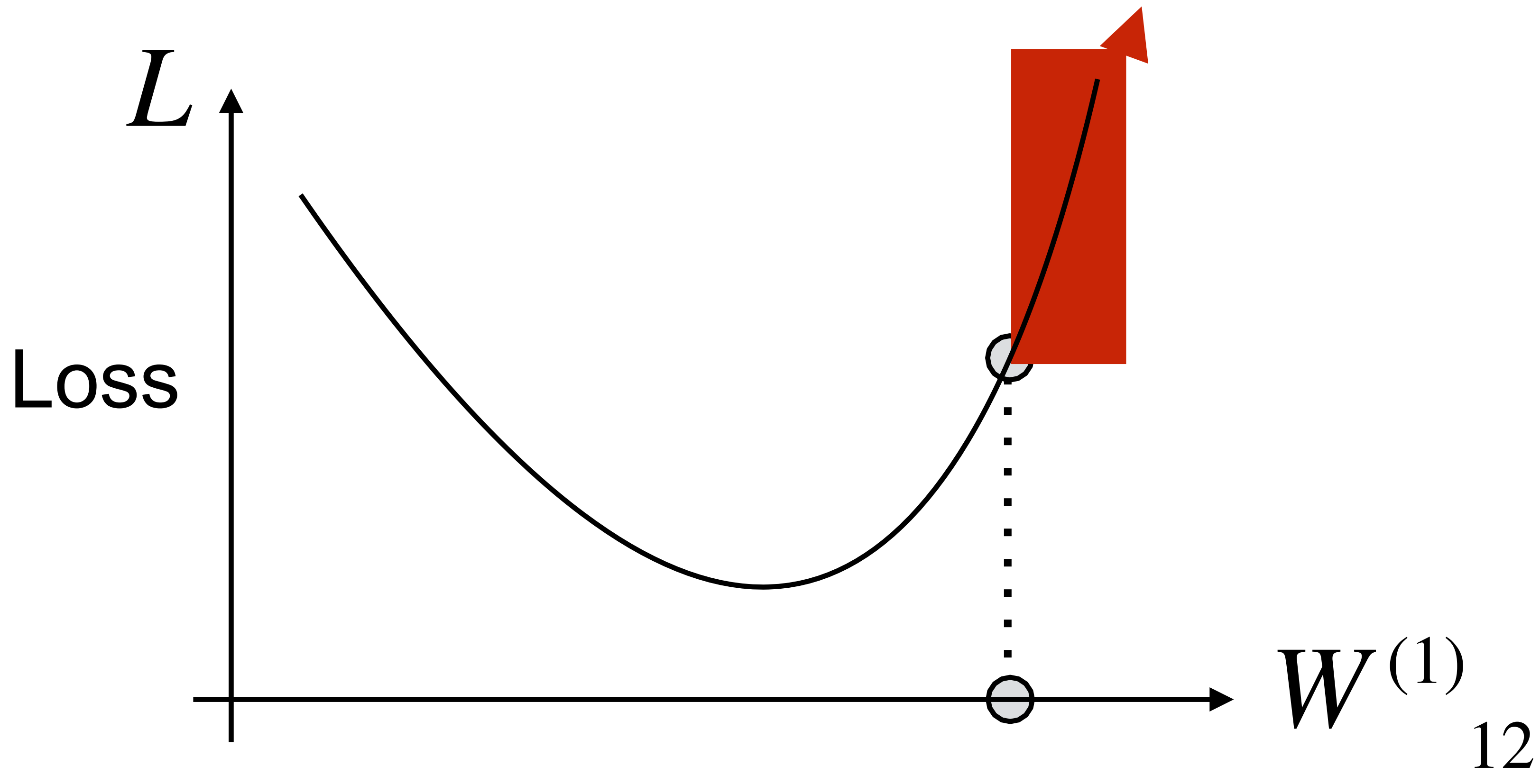


A weight somewhere in the network

Review: Setup

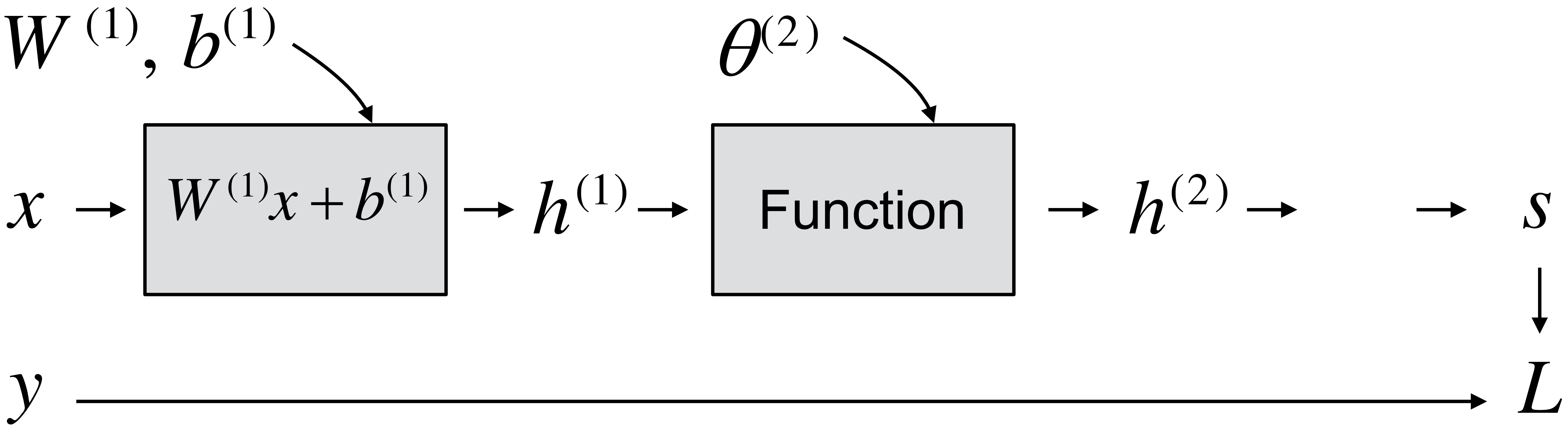


Toy Example:

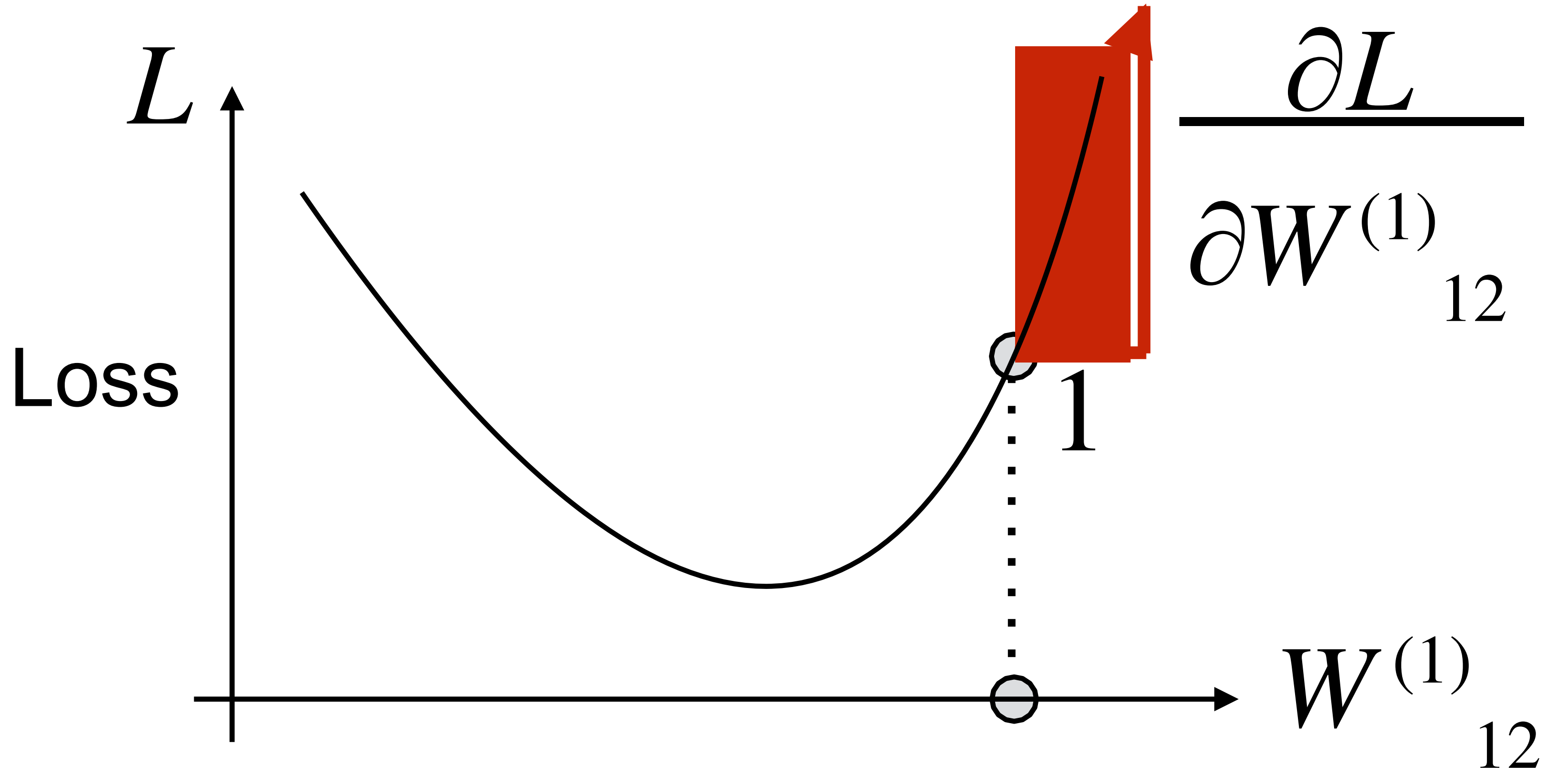


A weight somewhere in the network

Review: Setup

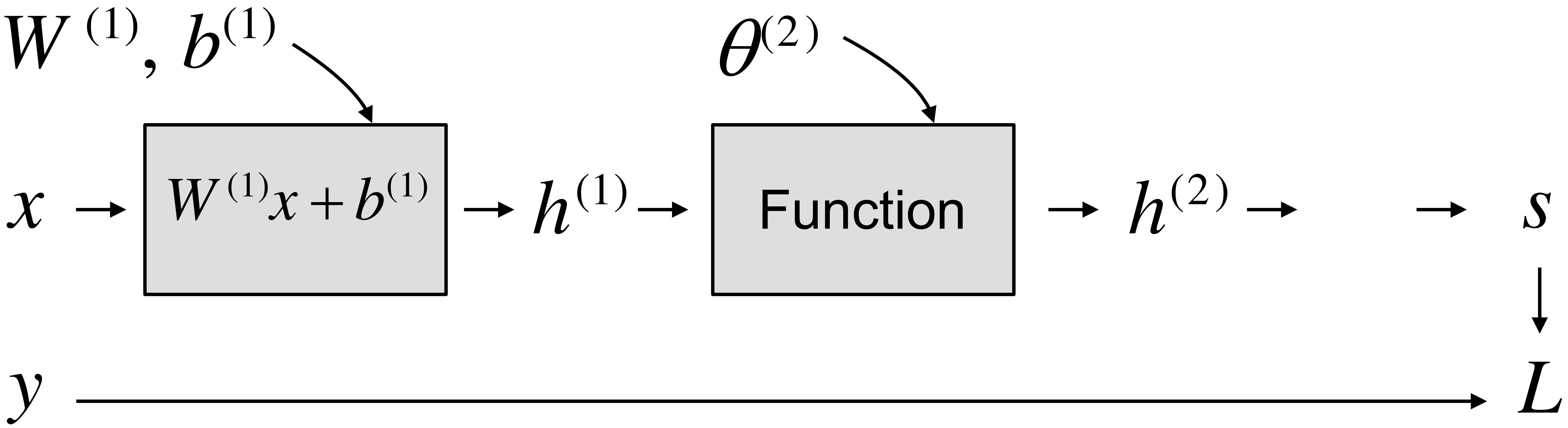


Toy Example:



A weight somewhere in the network

Review: Setup



$W^{(1)}, b^{(1)}$

$\theta^{(2)}$

x

$W^{(1)}x + b^{(1)}$

$h^{(1)}$

Function

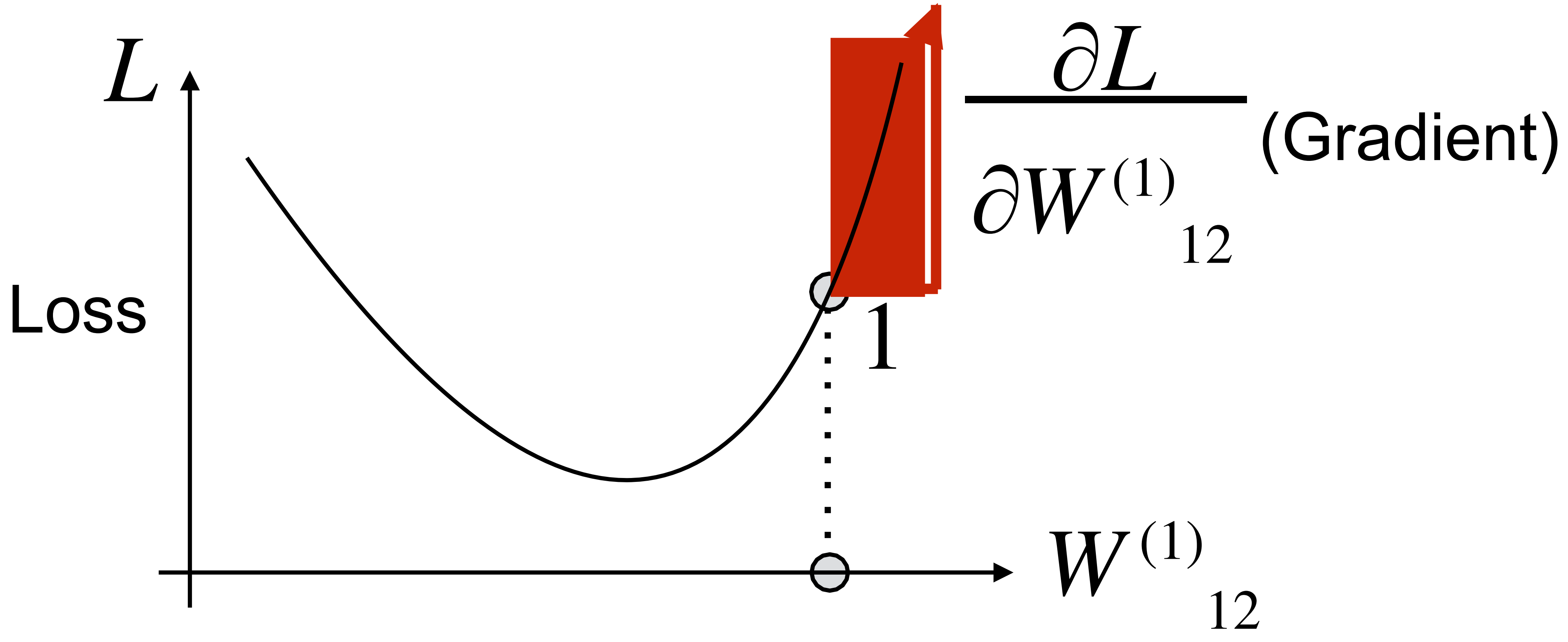
$h^{(2)}$

s

y

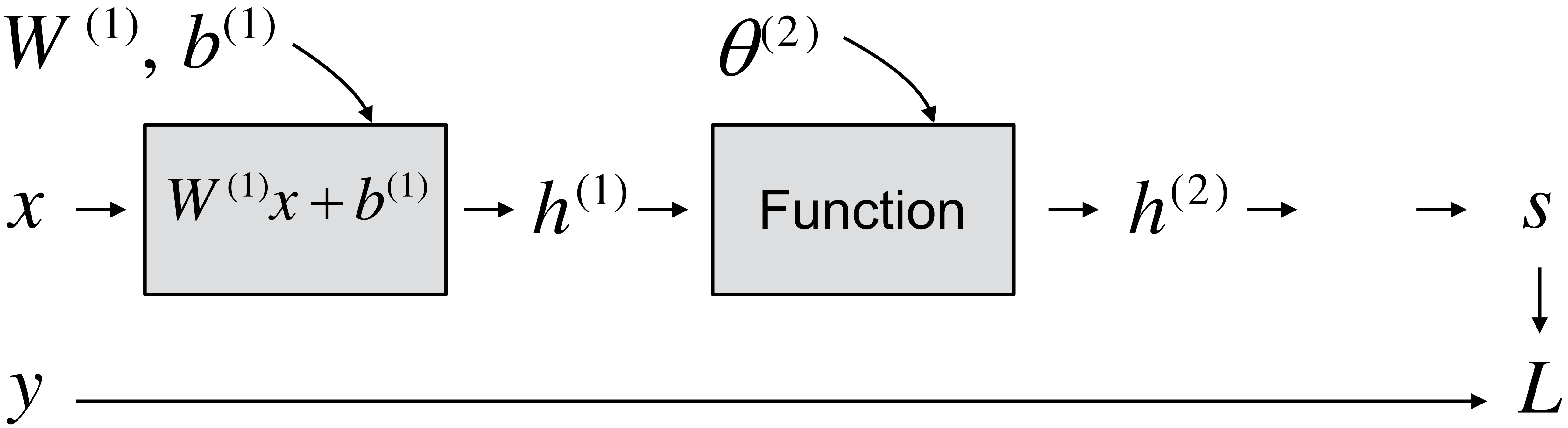
L

Toy Example:

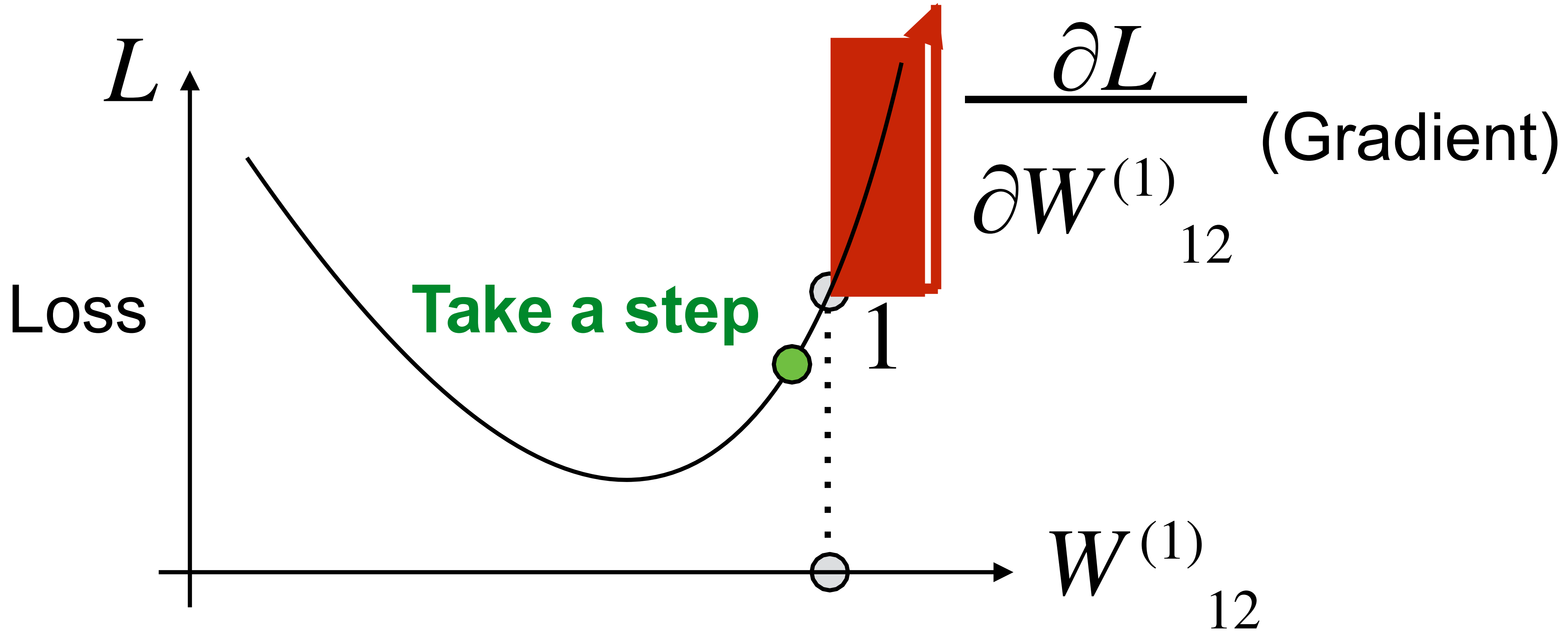


A weight somewhere in the network

Review: Setup

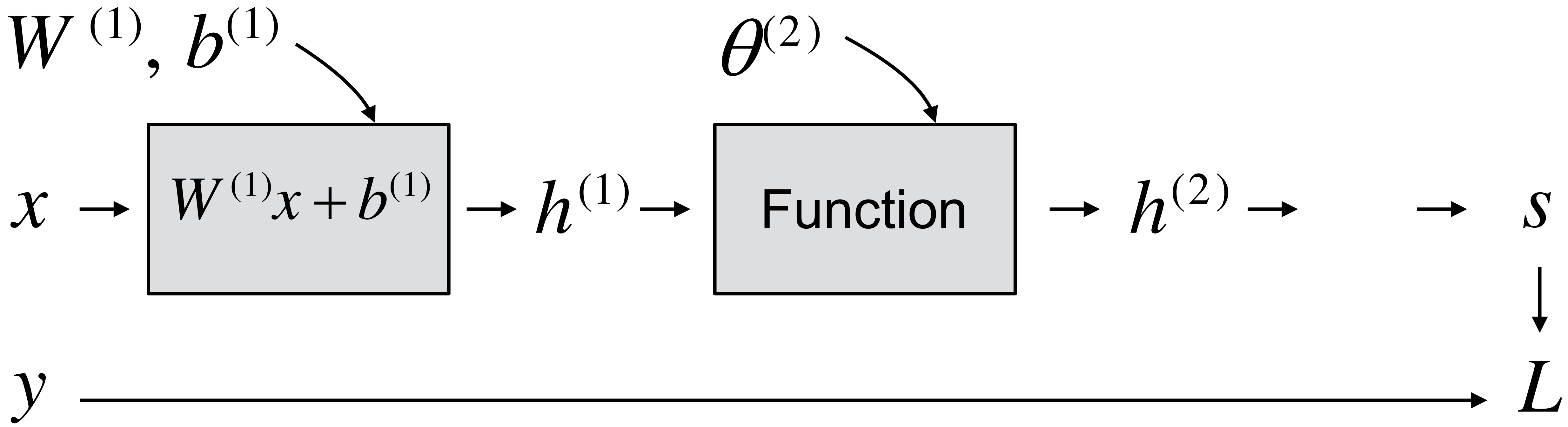


Toy Example:



A weight somewhere in the network

Review: Setup



$W^{(1)}, b^{(1)}$

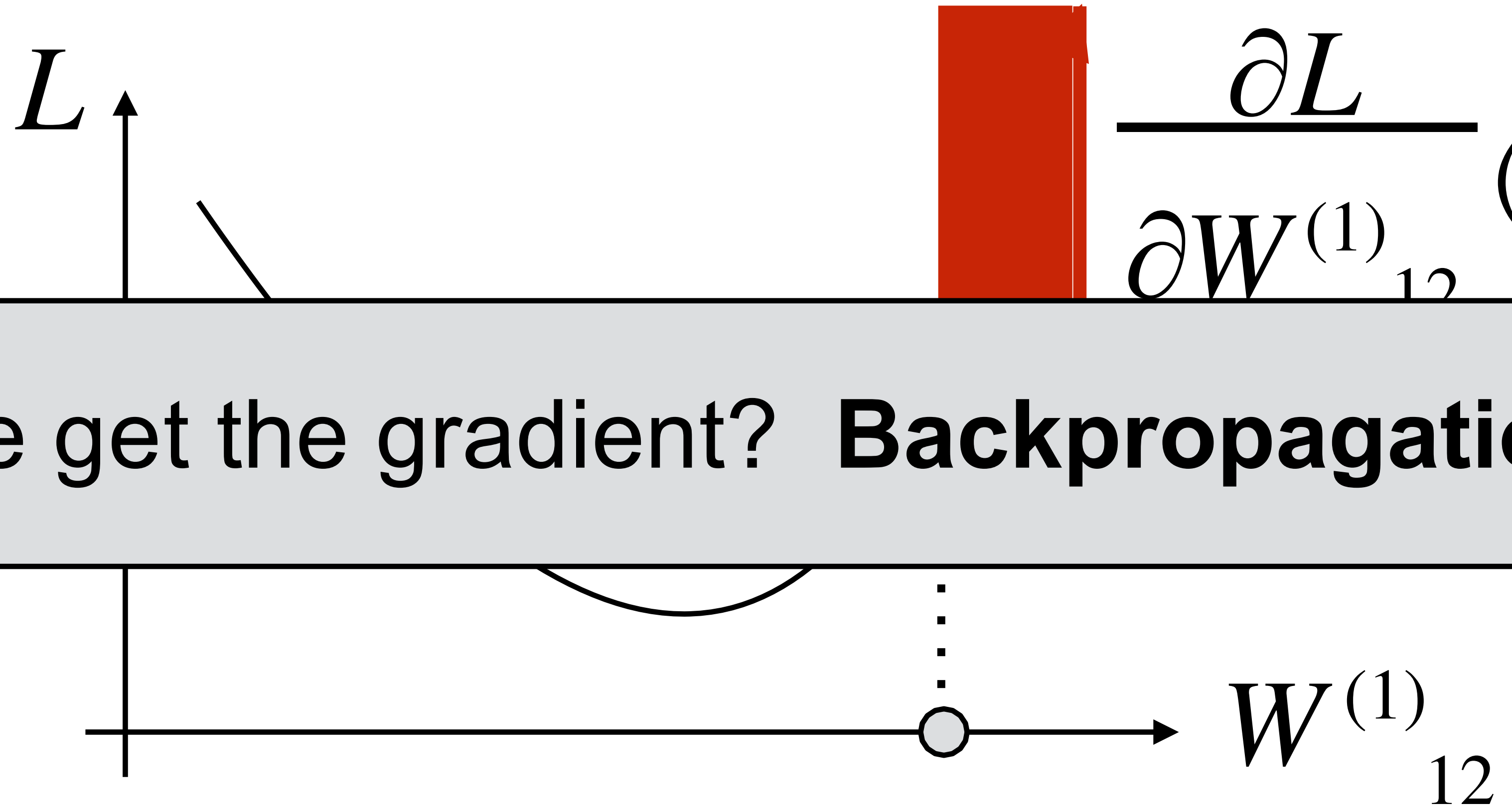
$\theta^{(2)}$

$x \rightarrow \boxed{W^{(1)}x + b^{(1)}} \rightarrow h^{(1)} \rightarrow \boxed{\text{Function}} \rightarrow h^{(2)} \rightarrow \quad \rightarrow s$

$s \downarrow L$

$y \longrightarrow L$

Toy Example:



$\frac{\partial L}{\partial W^{(1)}_{12}}$ (Gradient)

How do we get the gradient? **Backpropagation**

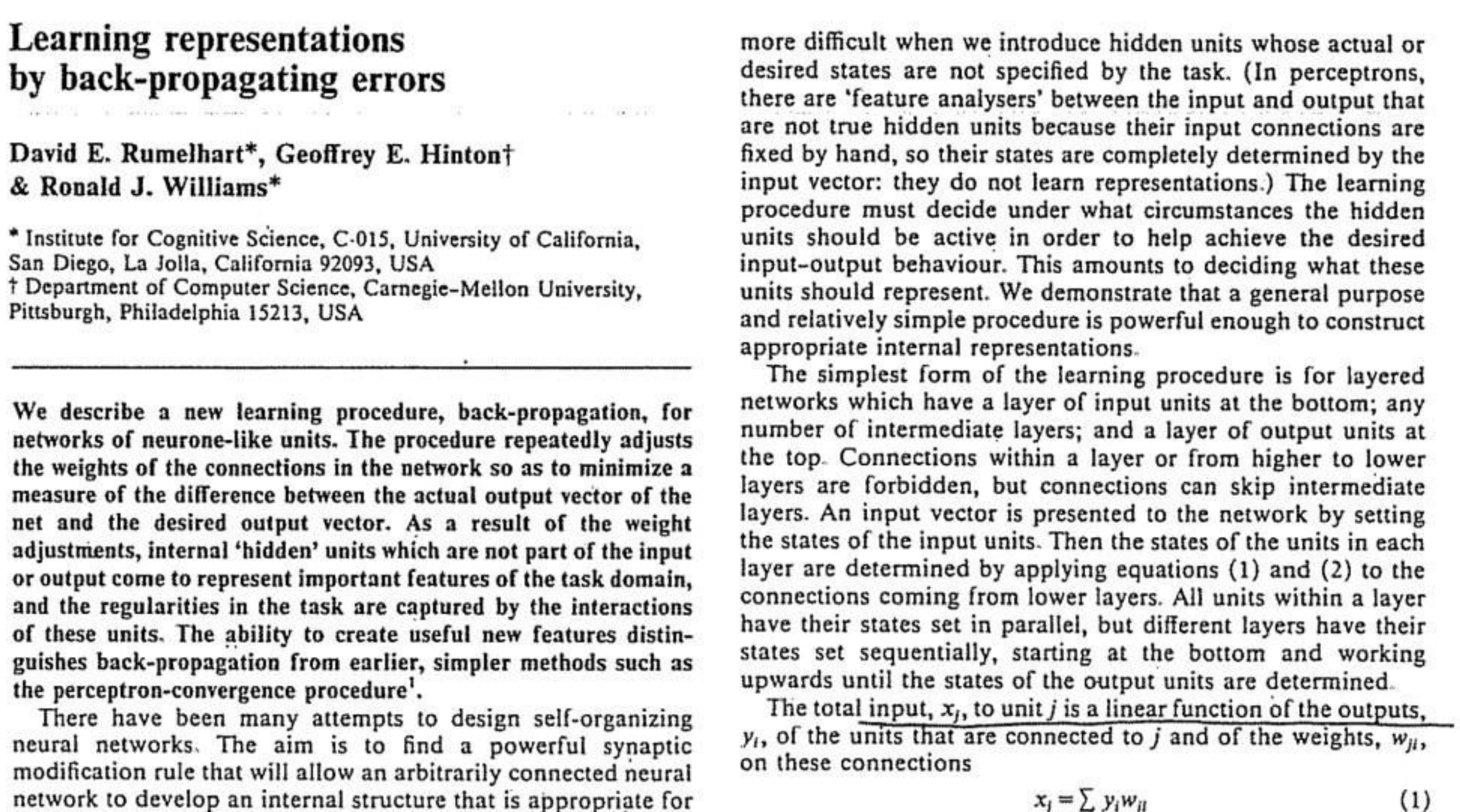
A weight somewhere in the network

Backprop

It’s just the chain rule

Backpropagation

[Rumelhart, Hinton, Williams. Nature 1986]



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

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

Chain rule recap

I hope everyone remembers the chain rule:

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

Chain rule recap

I hope everyone remembers the chain rule:

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

Forward propagation: $x \longrightarrow h \longrightarrow$

Backward propagation: $\frac{\partial L}{\partial x} \longleftarrow \frac{\partial L}{\partial h} \longleftarrow$

Chain rule recap

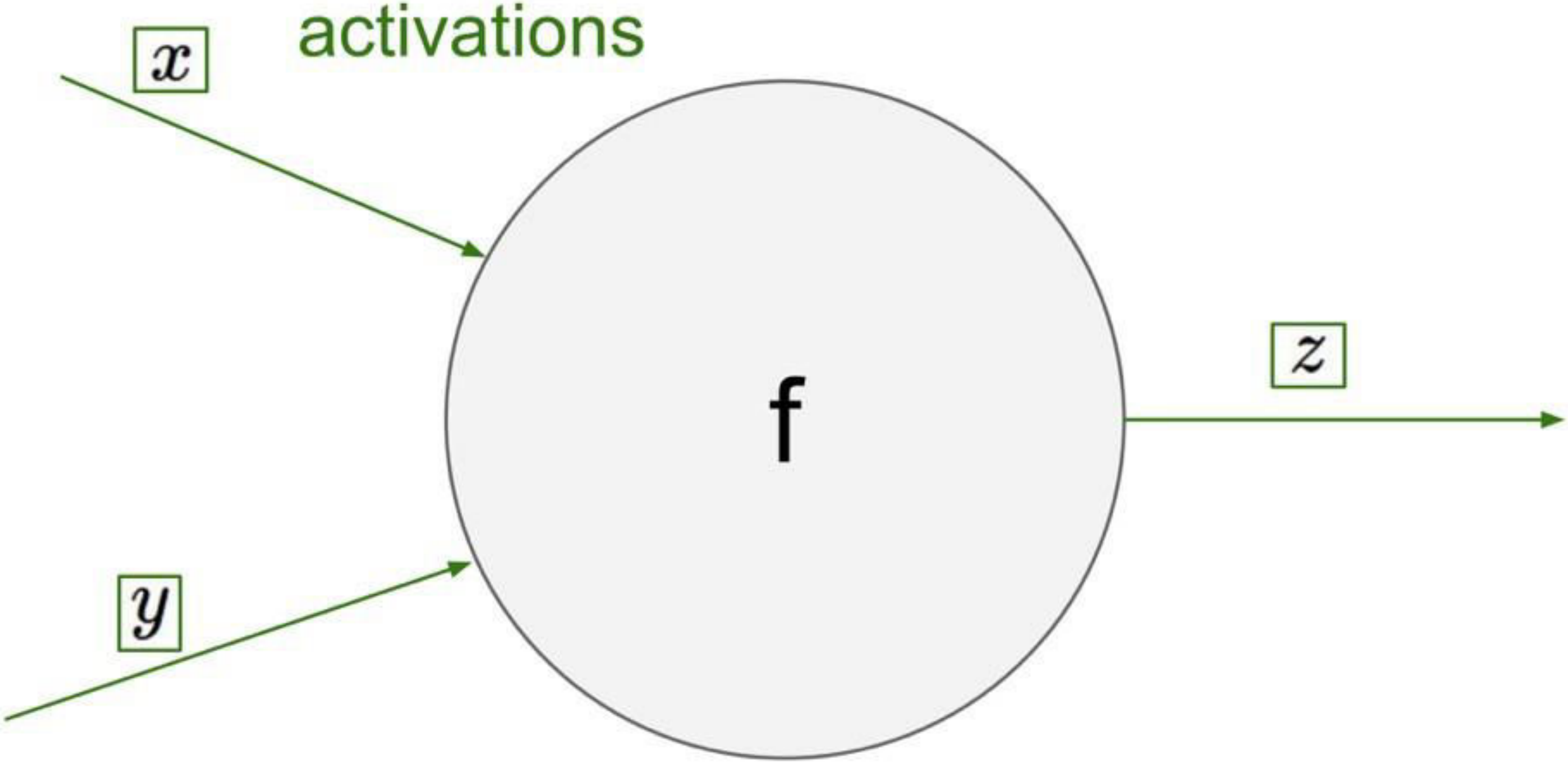
I hope everyone remembers the chain rule:

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

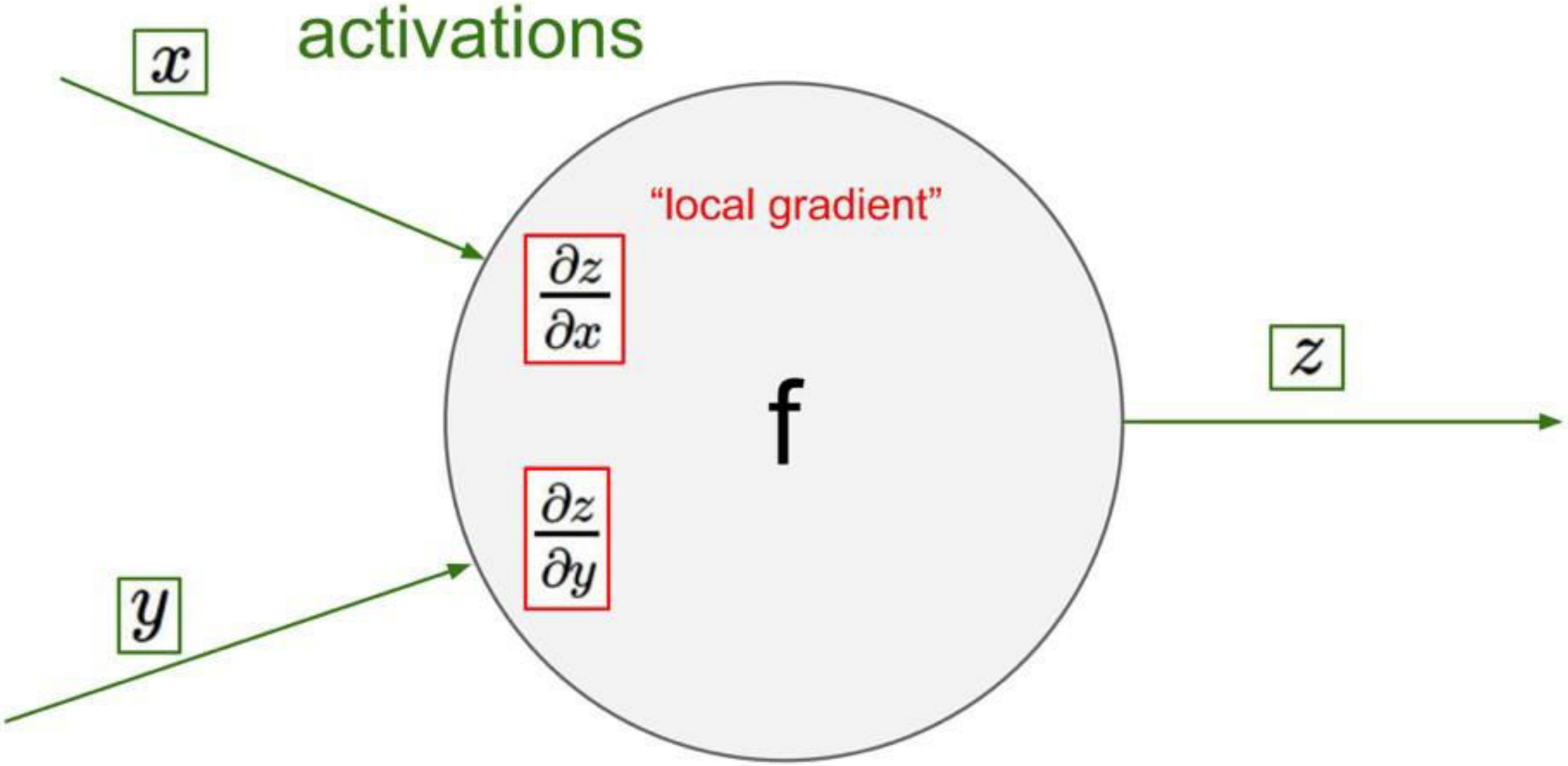
Forward propagation: $x \longrightarrow h \longrightarrow$

Backward propagation: $\frac{\partial L}{\partial x} \longleftarrow \frac{\partial L}{\partial h} \longleftarrow$

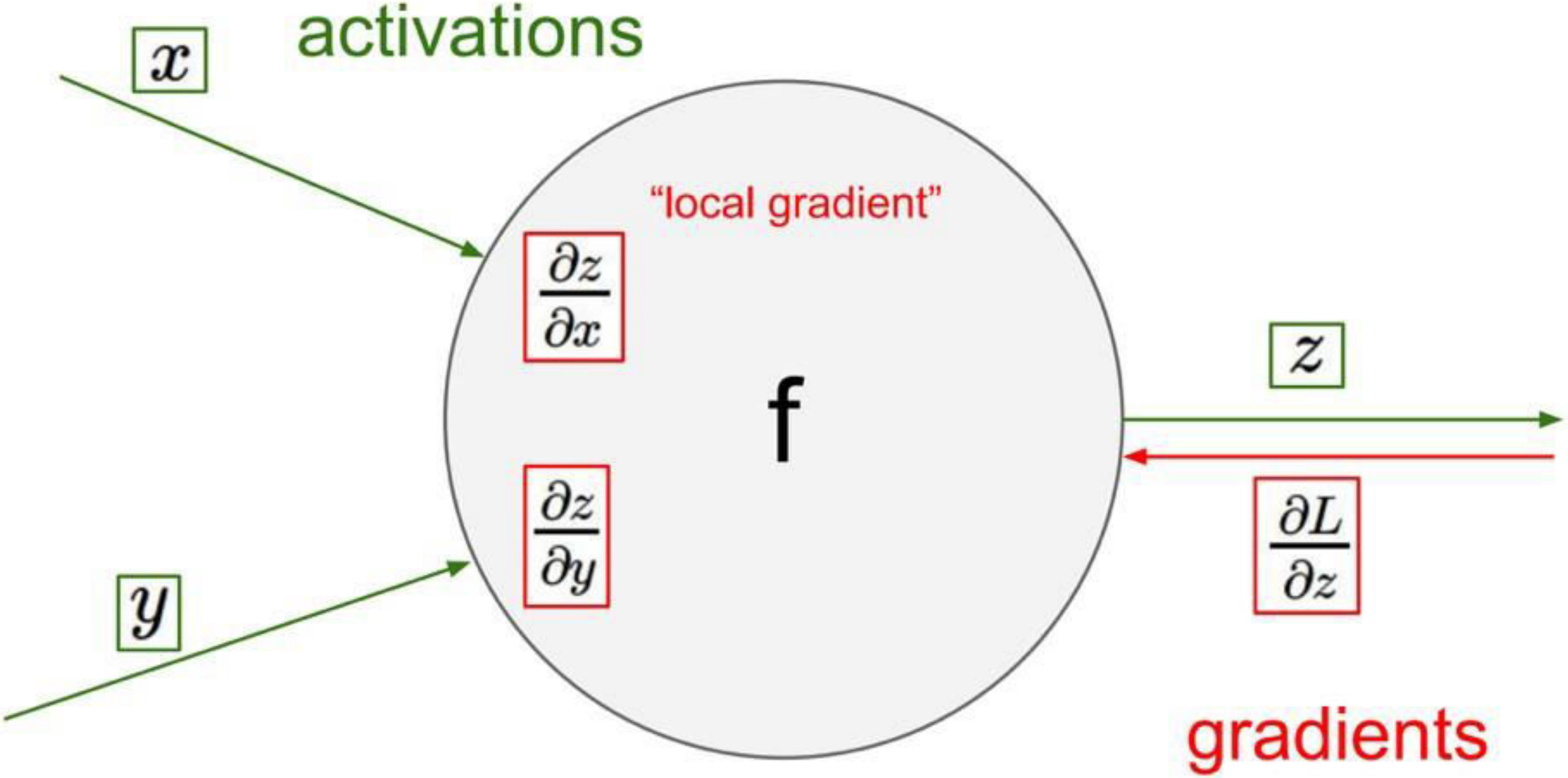
(extends easily to multi-dimensional x and y)



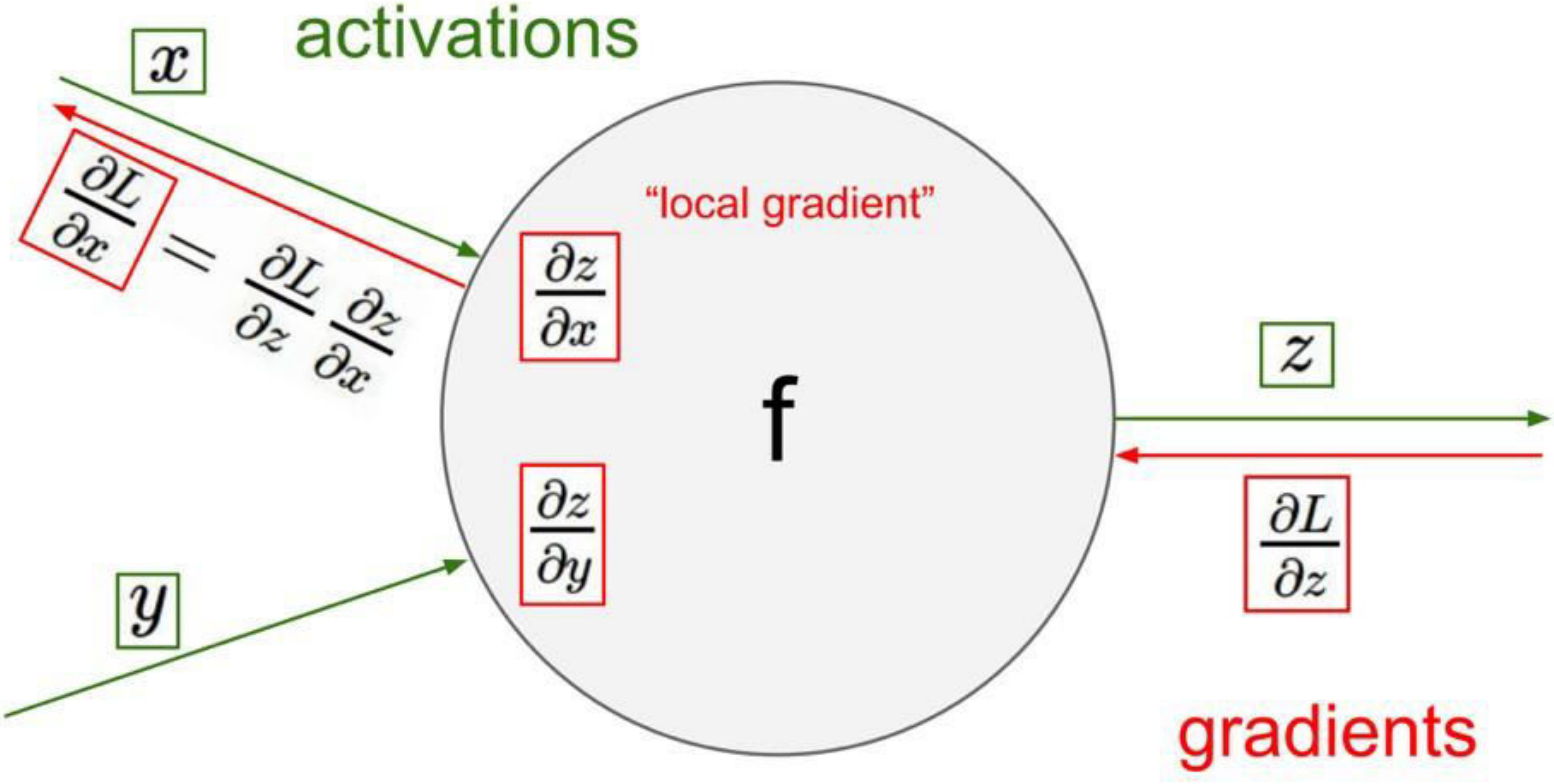
Slide from Karpathy 2016



Slide from Karpathy 2016



Slide from Karpathy 2016



activations

x

$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x}$

"local gradient"

$\frac{\partial z}{\partial x}$

f

z

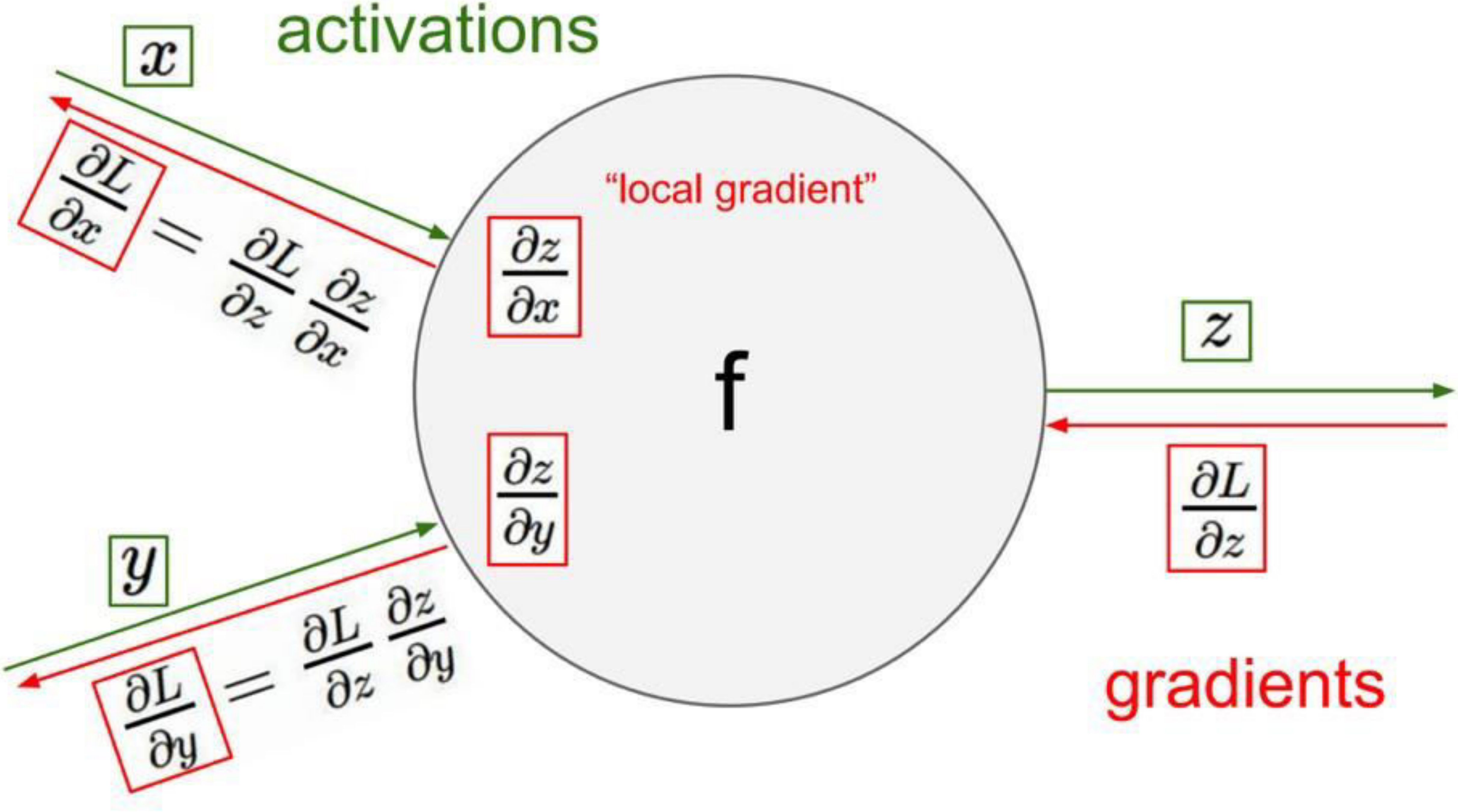
$\frac{\partial z}{\partial y}$

$\frac{\partial L}{\partial z}$

y

gradients

Slide from Karpathy 2016



activations

x

y

"local gradient"

$\frac{\partial z}{\partial x}$

$\frac{\partial z}{\partial y}$

f

z

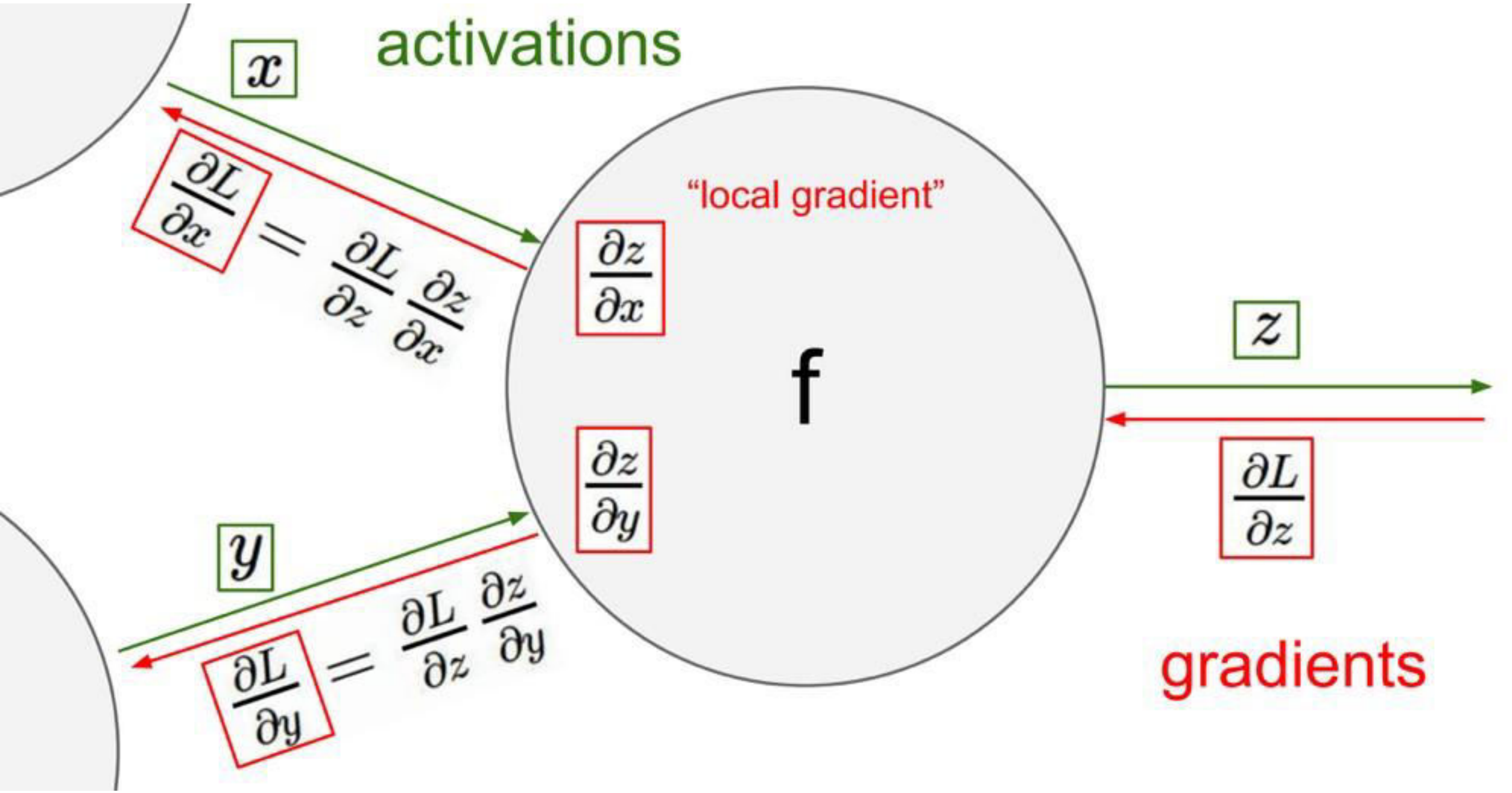
$\frac{\partial L}{\partial z}$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$$

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

gradients

Slide from Karpathy 2016



activations

x

$\boxed{\frac{\partial L}{\partial x}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$

"local gradient"

$\boxed{\frac{\partial z}{\partial x}}$

f

$\boxed{\frac{\partial z}{\partial y}}$

z

$\boxed{\frac{\partial L}{\partial z}}$

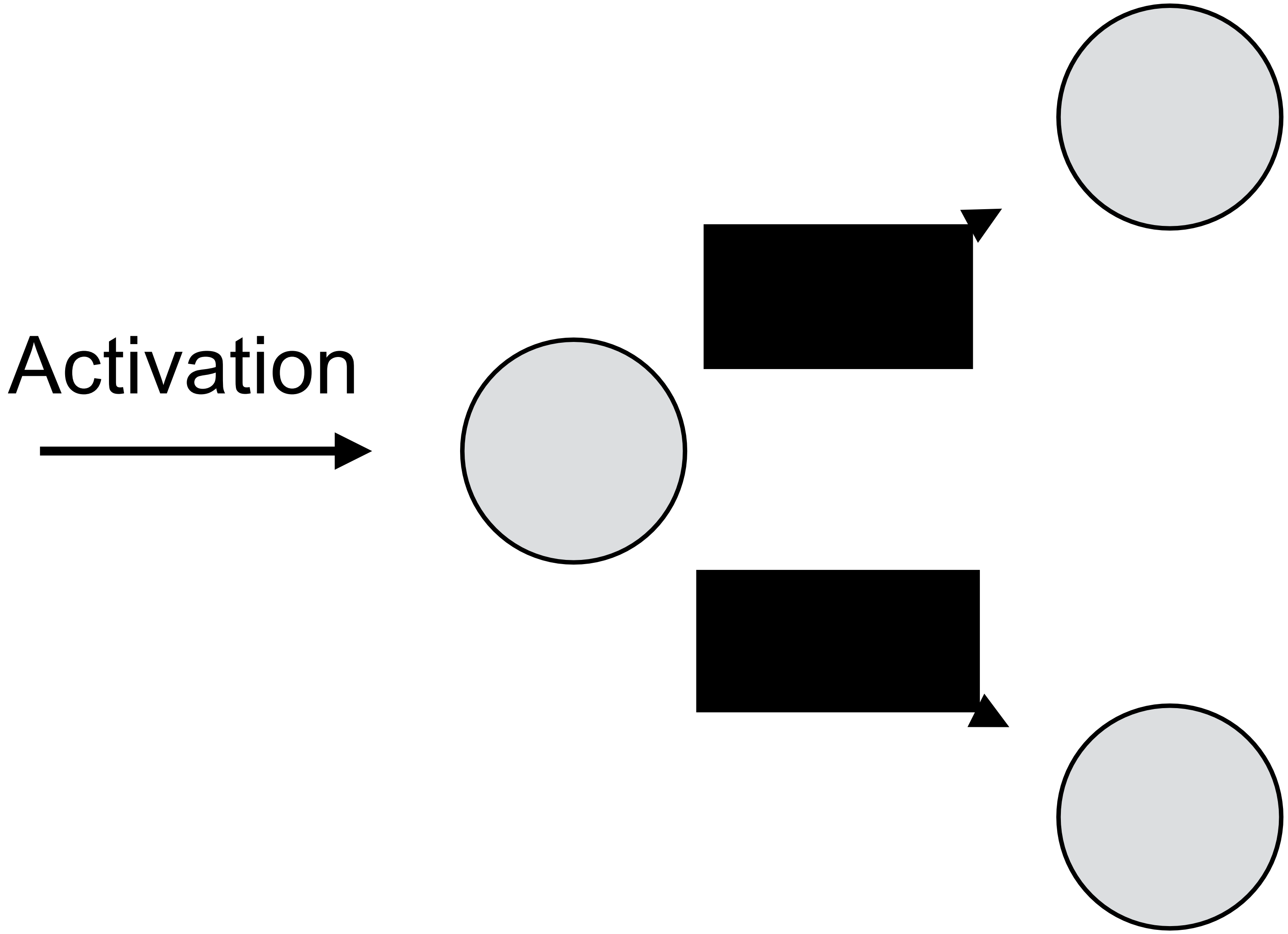
y

$\boxed{\frac{\partial L}{\partial y}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

gradients

Slide from Karpathy 2016

Gradients add at branches

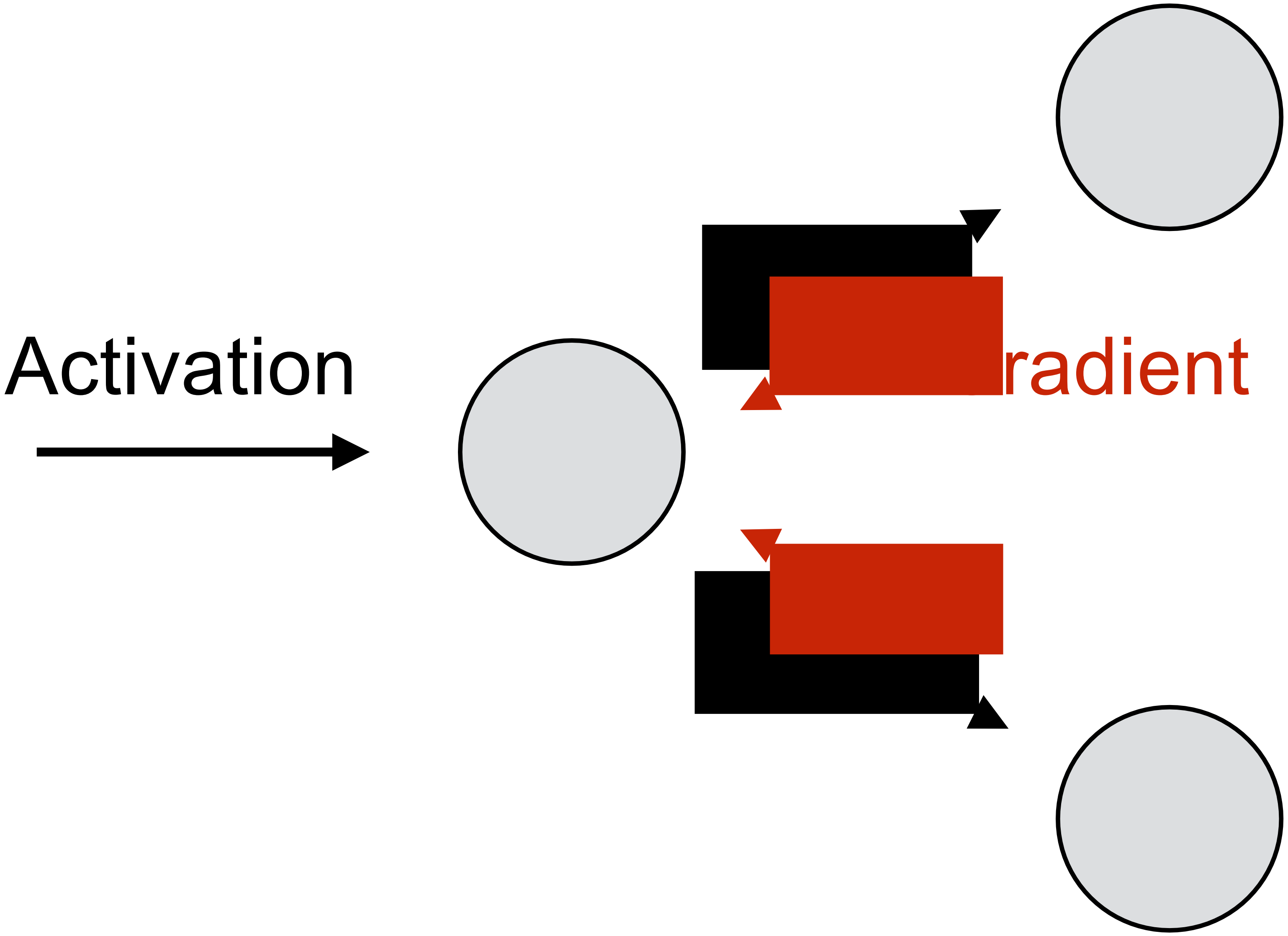


Activation

[redacted]

[redacted]

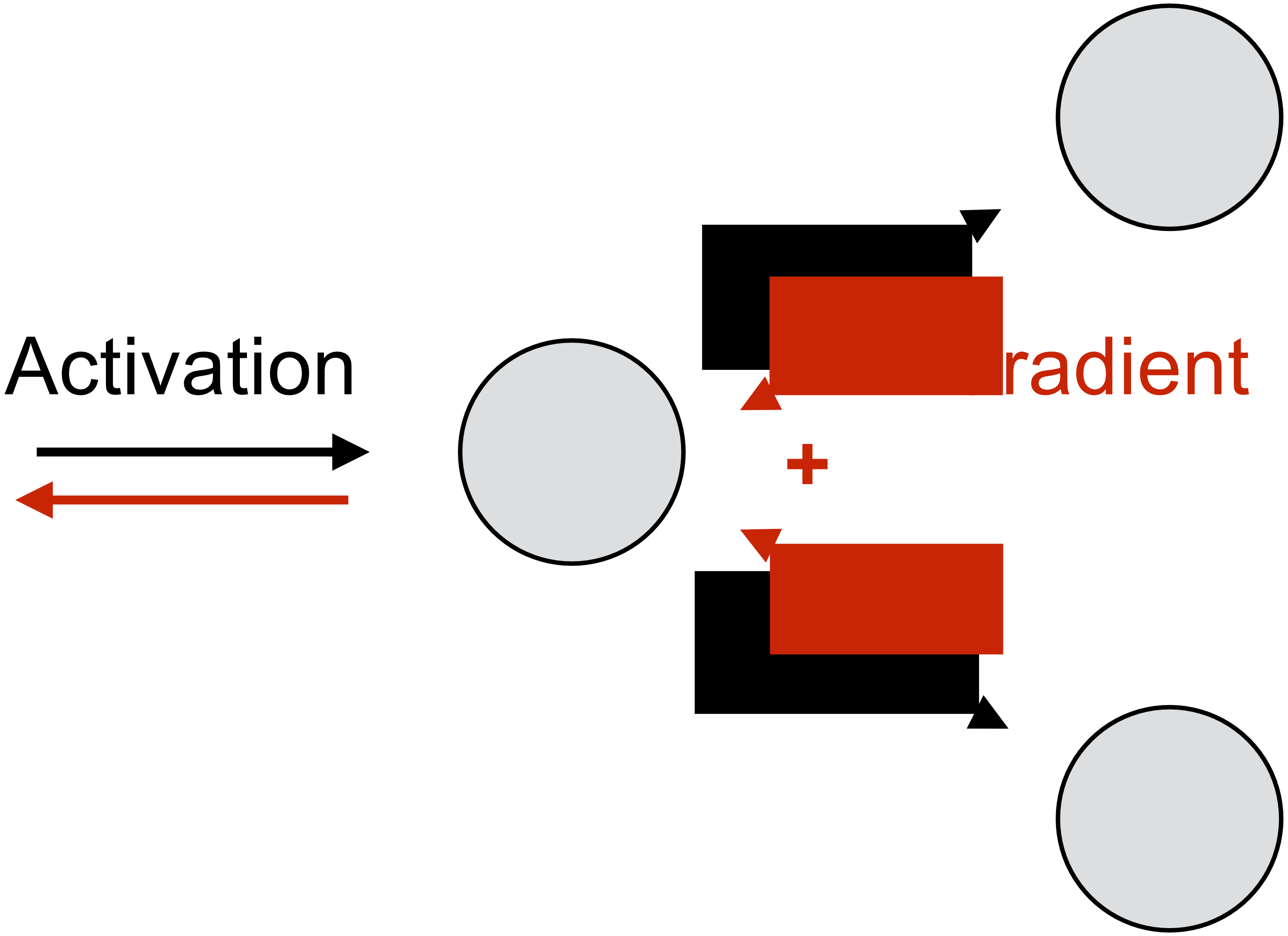
Gradients add at branches



Activation

radient

Gradients add at branches

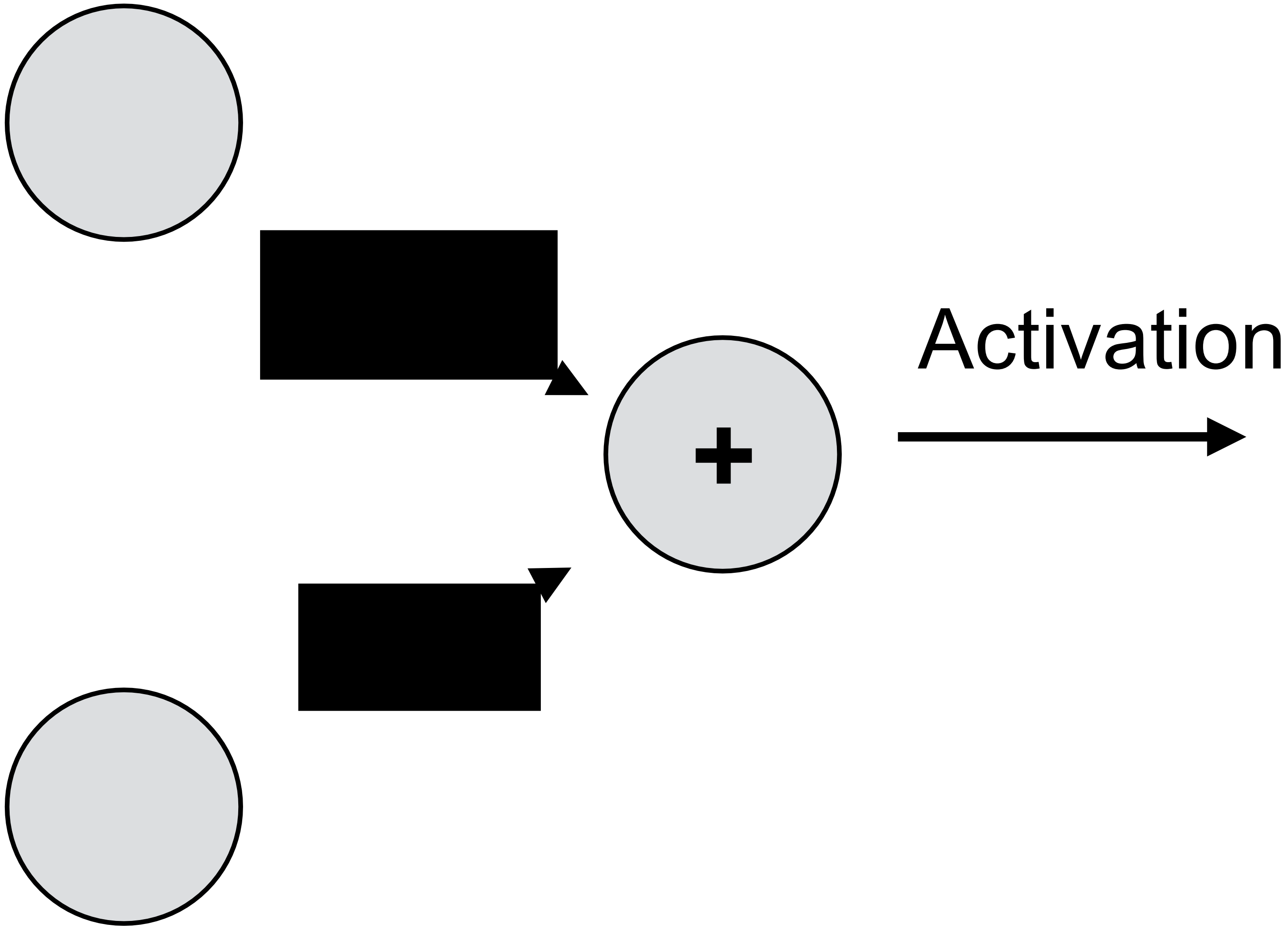


Activation

radient

+

Gradients copy through sums

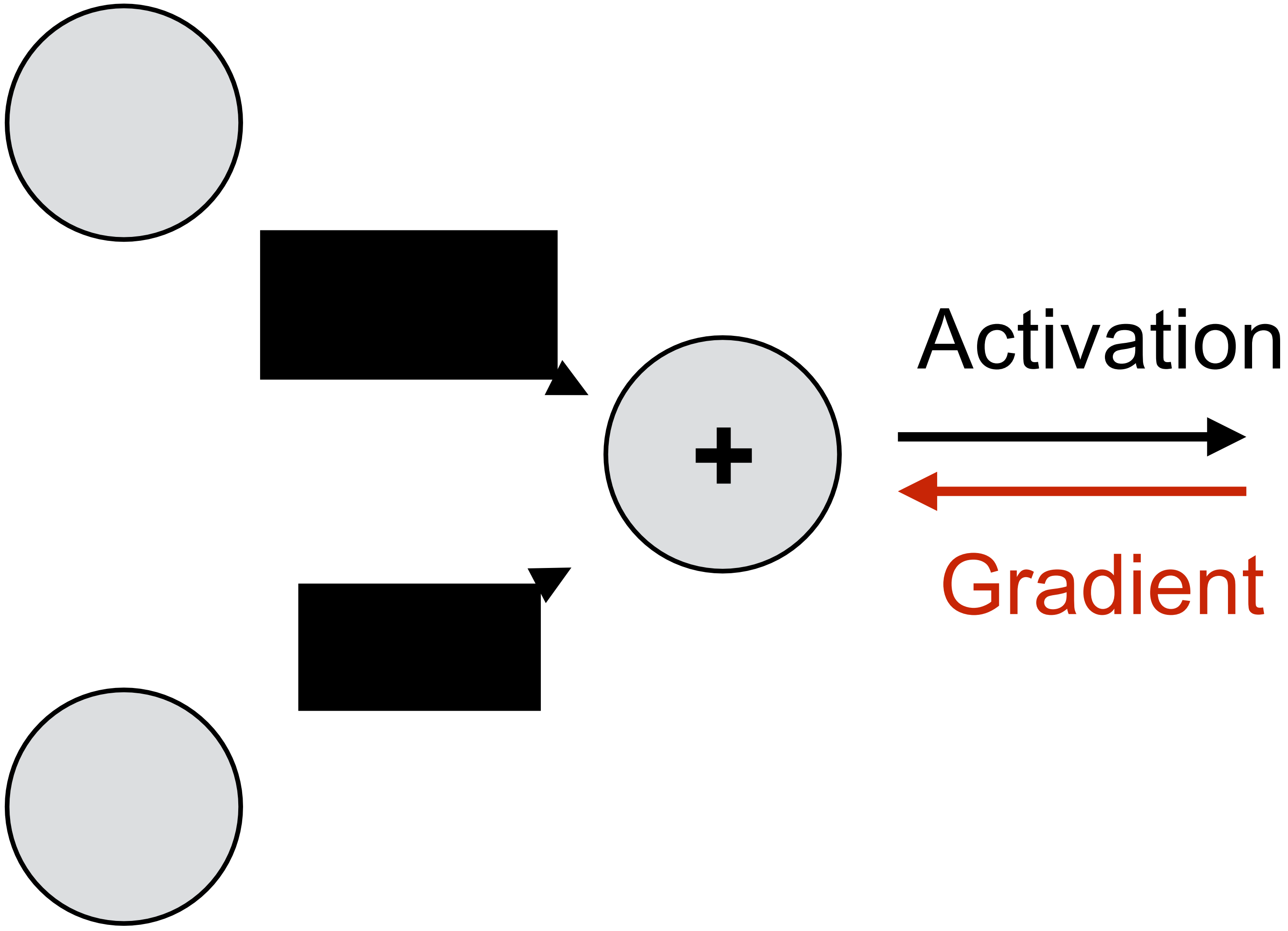


[redacted]

[redacted]

Activation

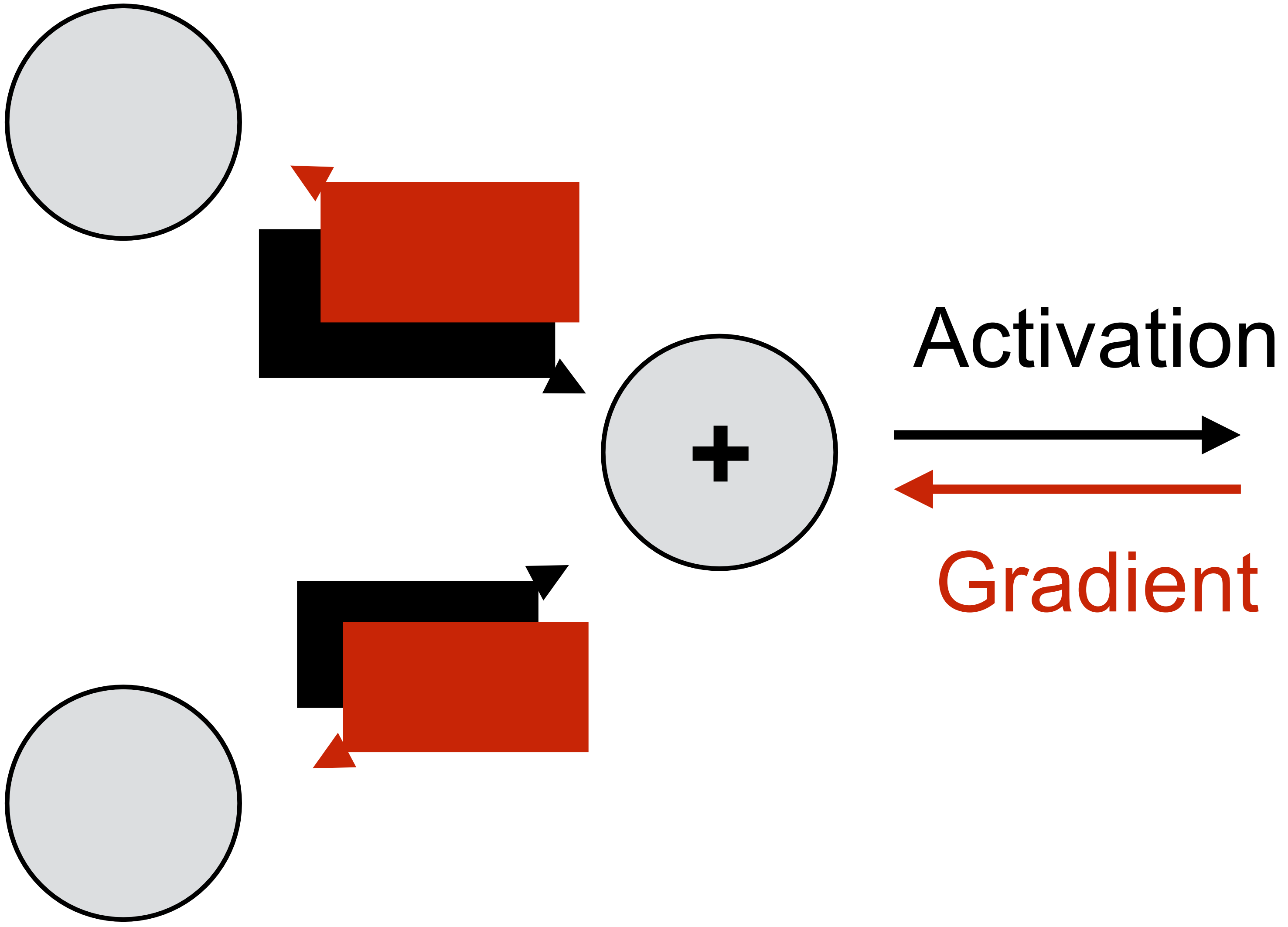
Gradients copy through sums



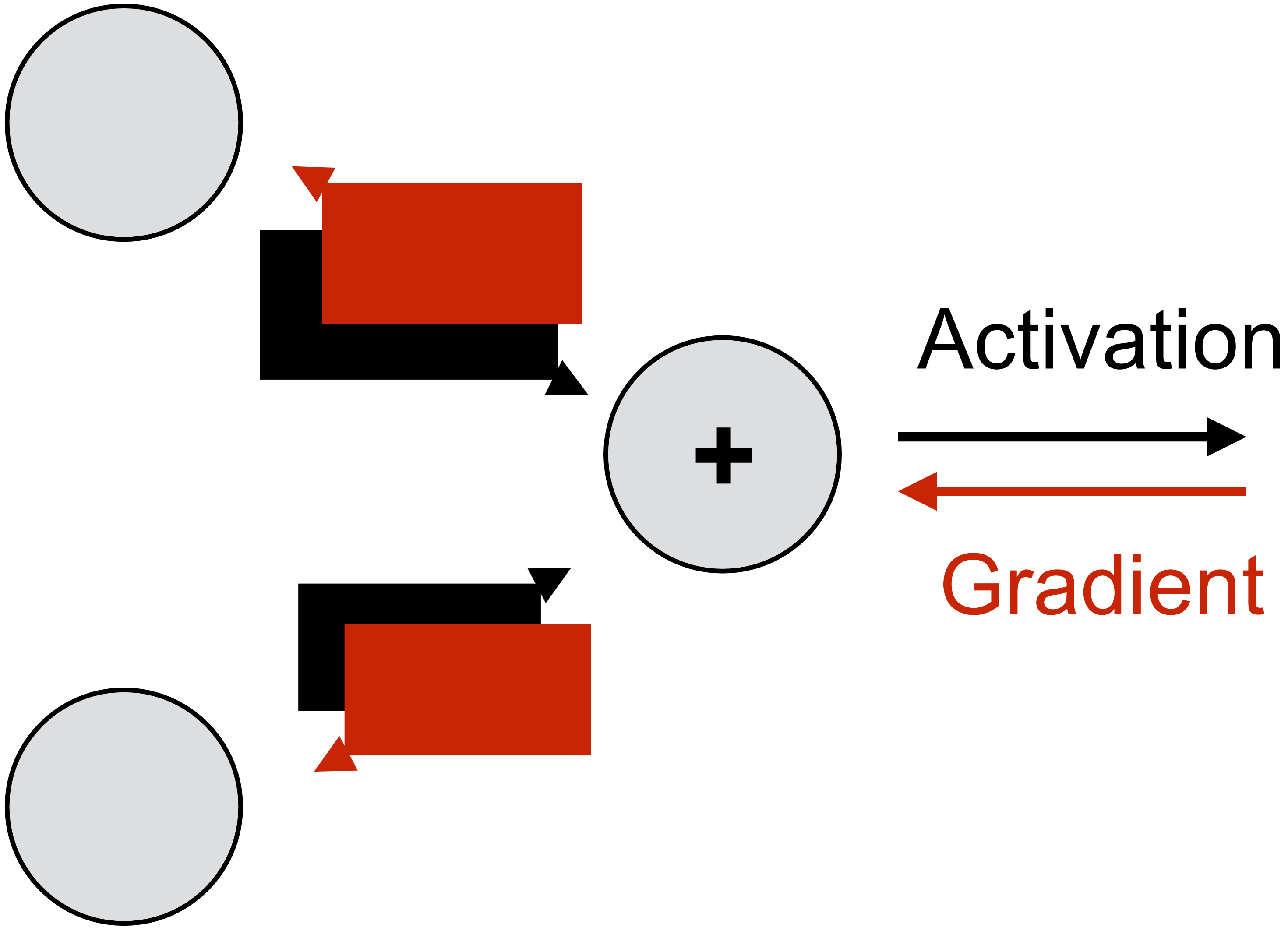
[redacted]

[redacted]

Gradients copy through sums

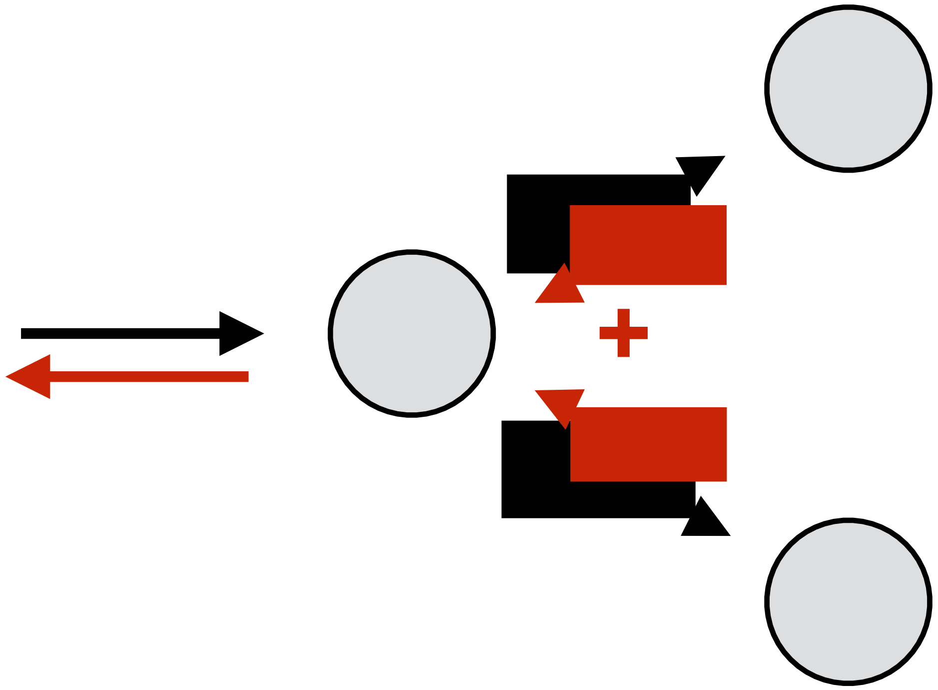


Gradients copy through sums

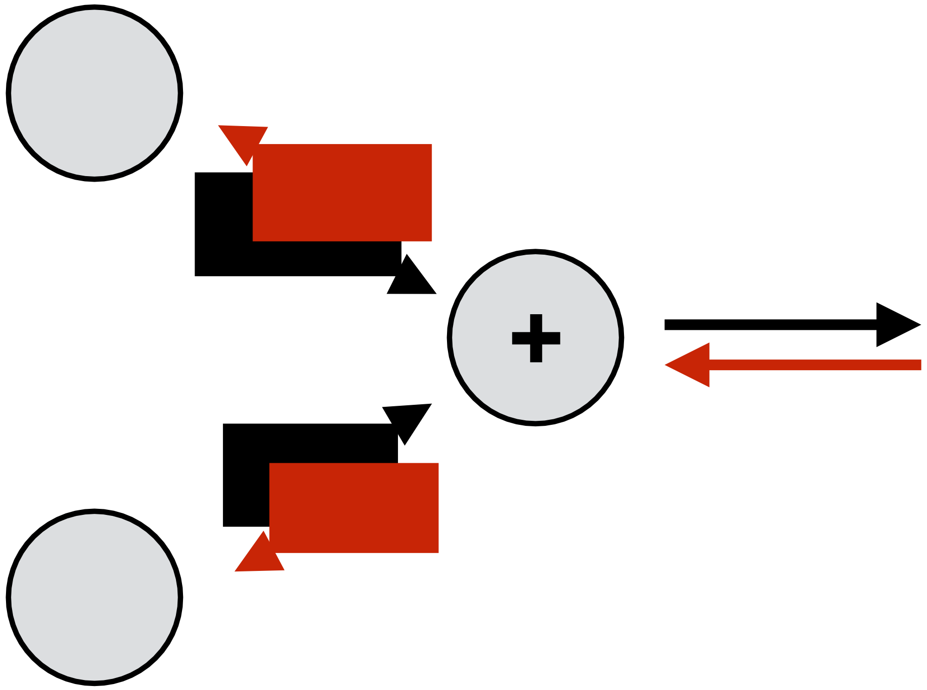


The gradient flows through both branches at “full strength”

Symmetry between forward and backward

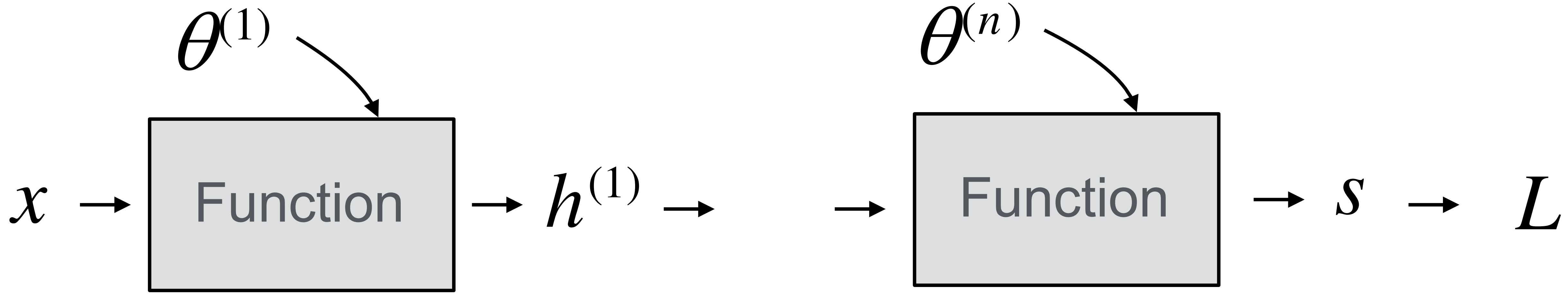


Forward: copy
Backward: add

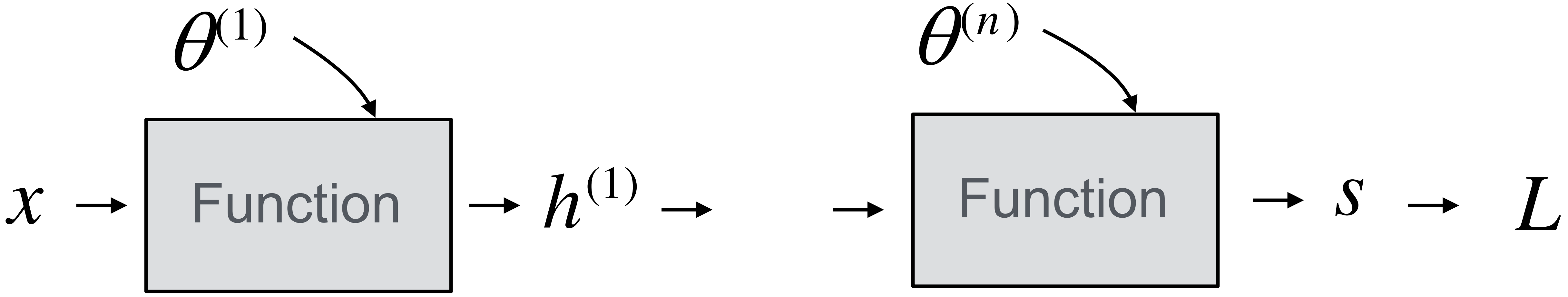


Forward: add
Backward: copy

Forward Propagation:

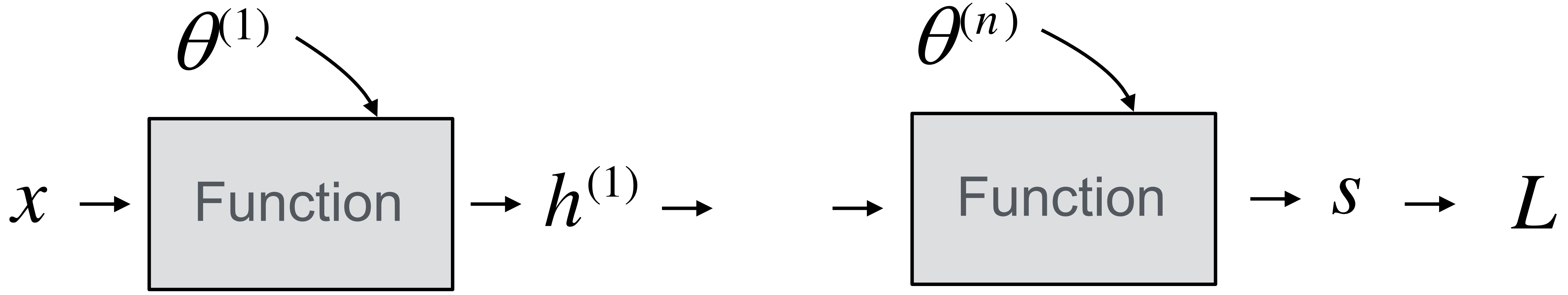


Forward Propagation:



Backward Propagation:

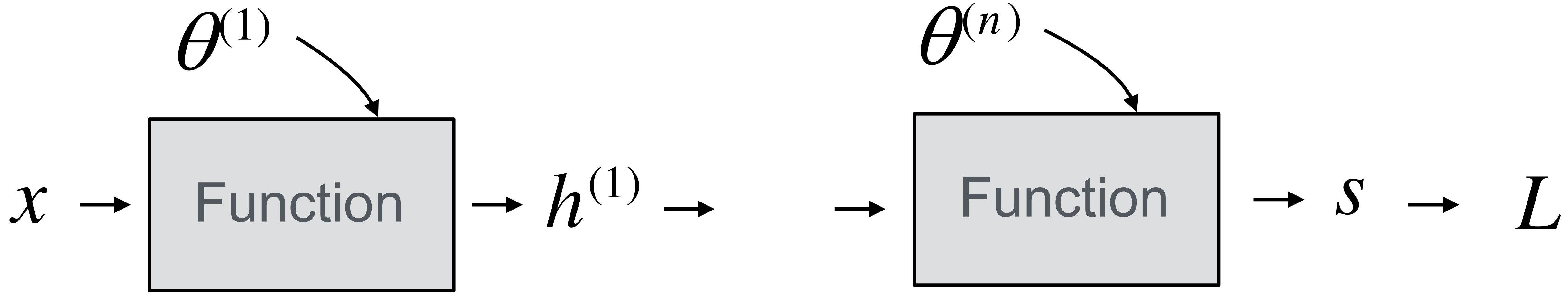
Forward Propagation:



Backward Propagation:

L

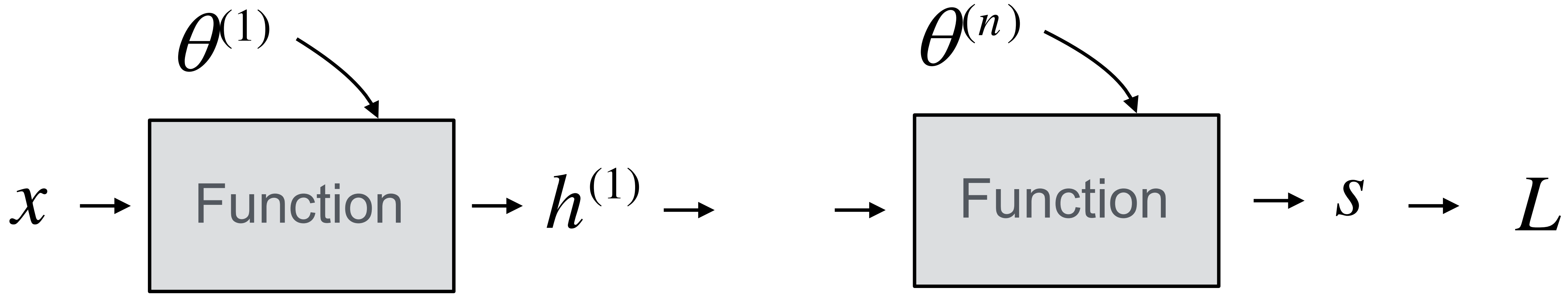
Forward Propagation:



Backward Propagation:

$$\frac{\partial L}{\partial s} \leftarrow L$$

Forward Propagation:

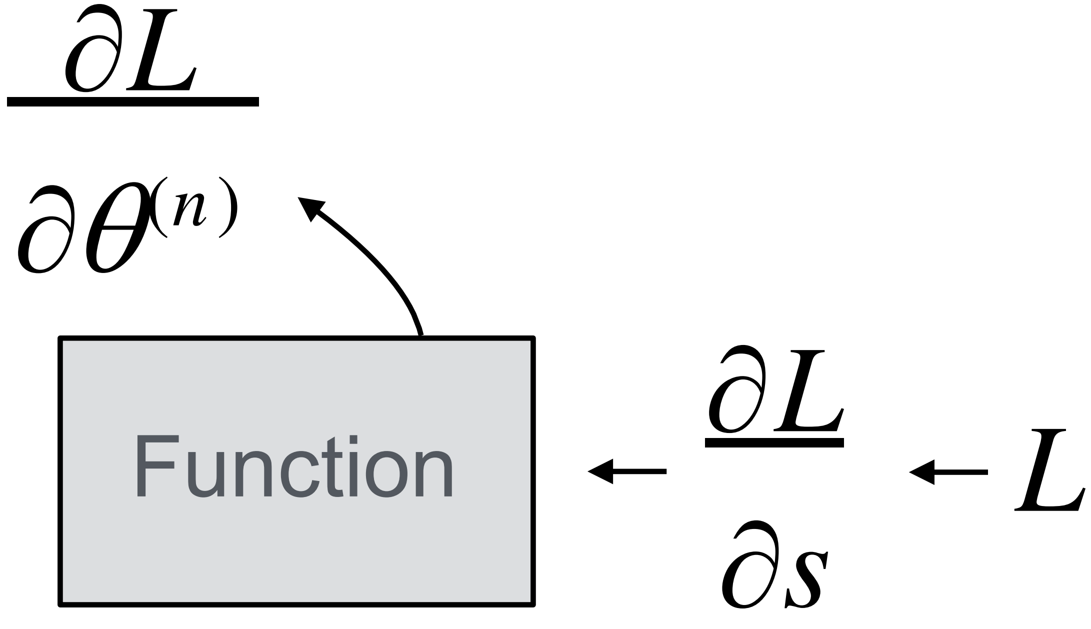

$\theta^{(1)}$

$\theta^{(n)}$

$x \rightarrow$ Function $\rightarrow h^{(1)} \rightarrow$

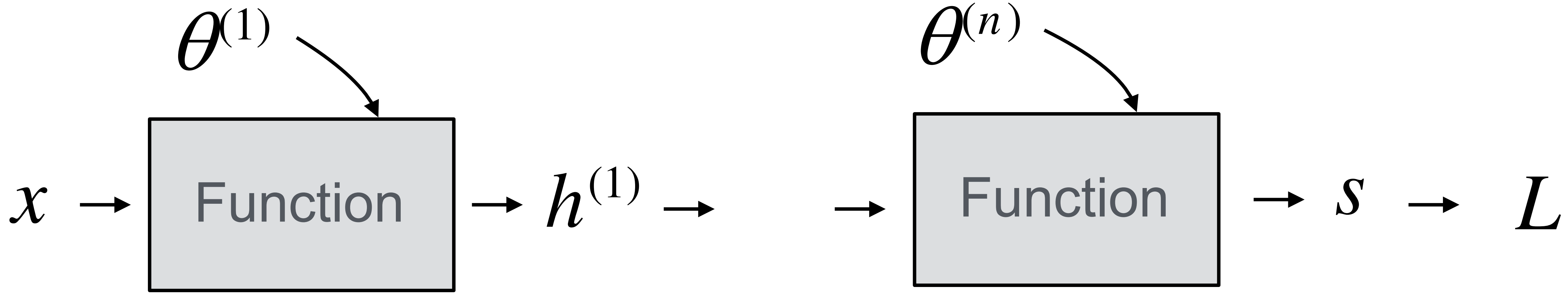
$\rightarrow$ Function $\rightarrow s \rightarrow L$

Backward Propagation:

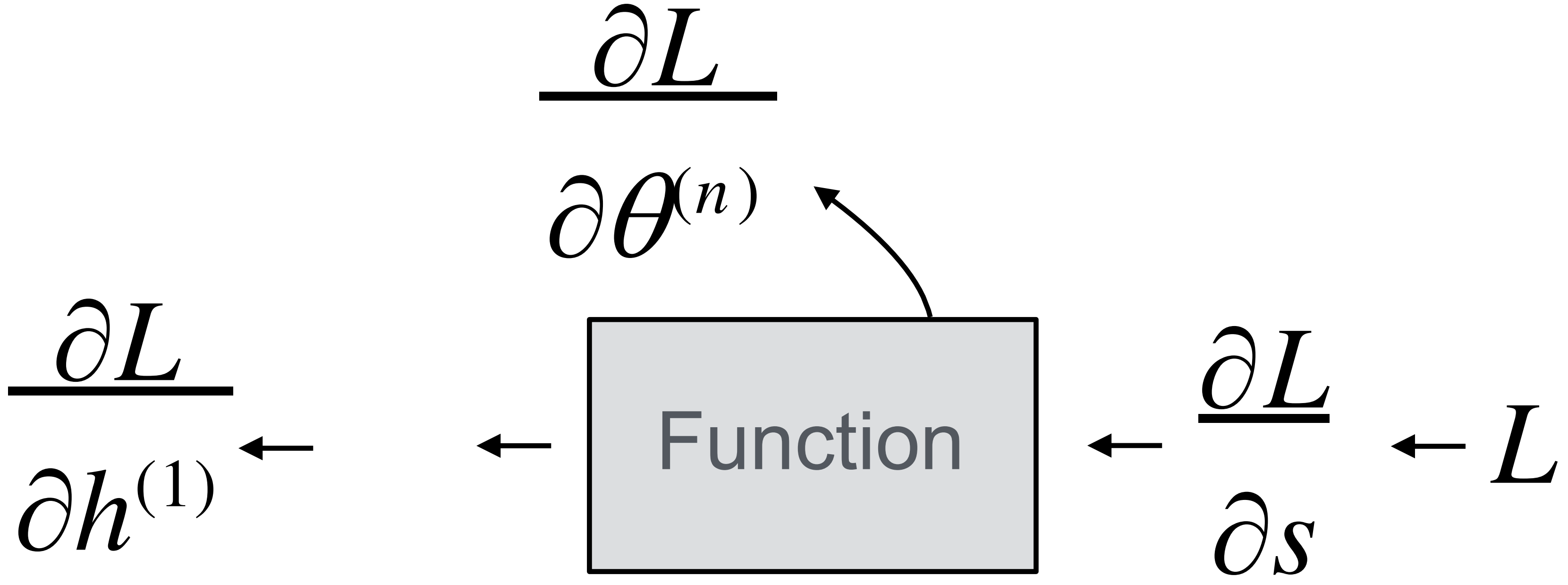

$\frac{\partial L}{\partial \theta^{(n)}}$

Function $\leftarrow \frac{\partial L}{\partial s} \leftarrow L$

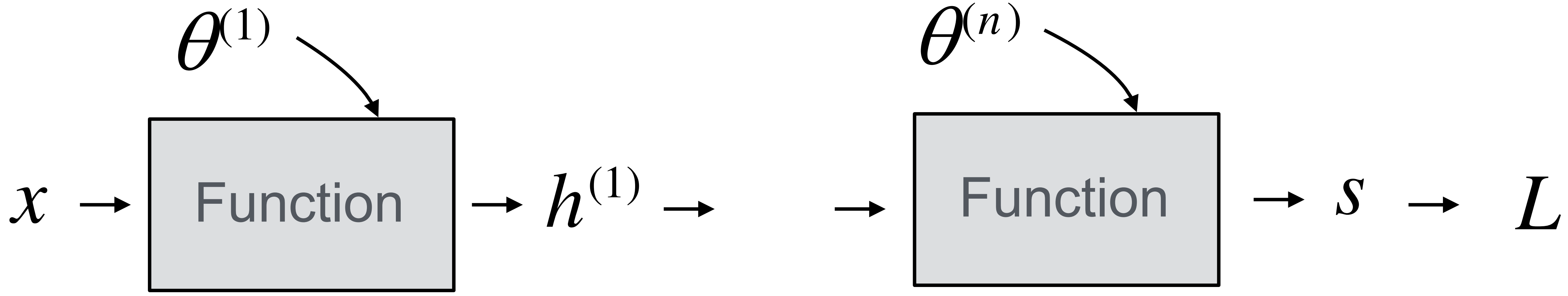
Forward Propagation:



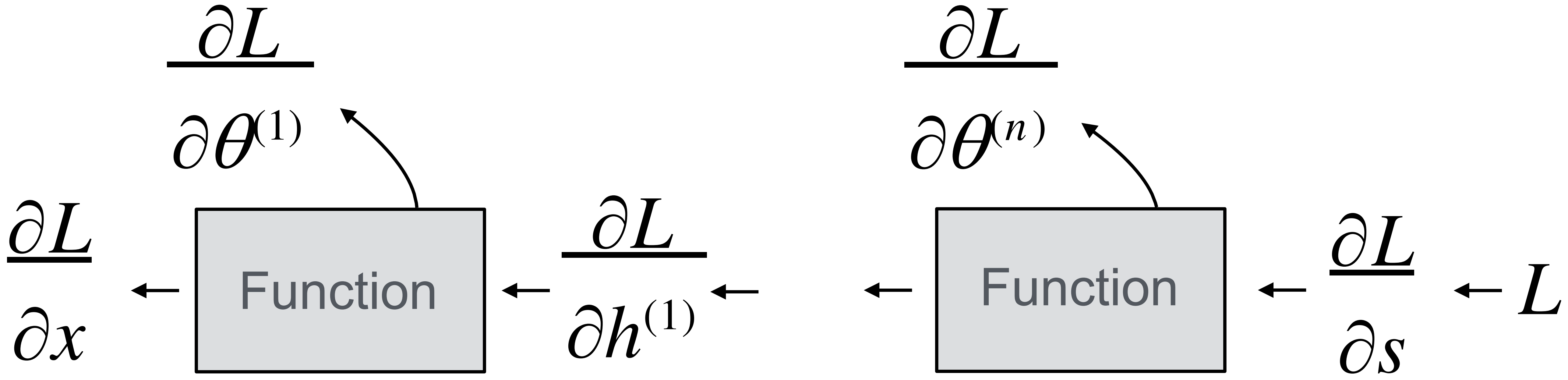
Backward Propagation:



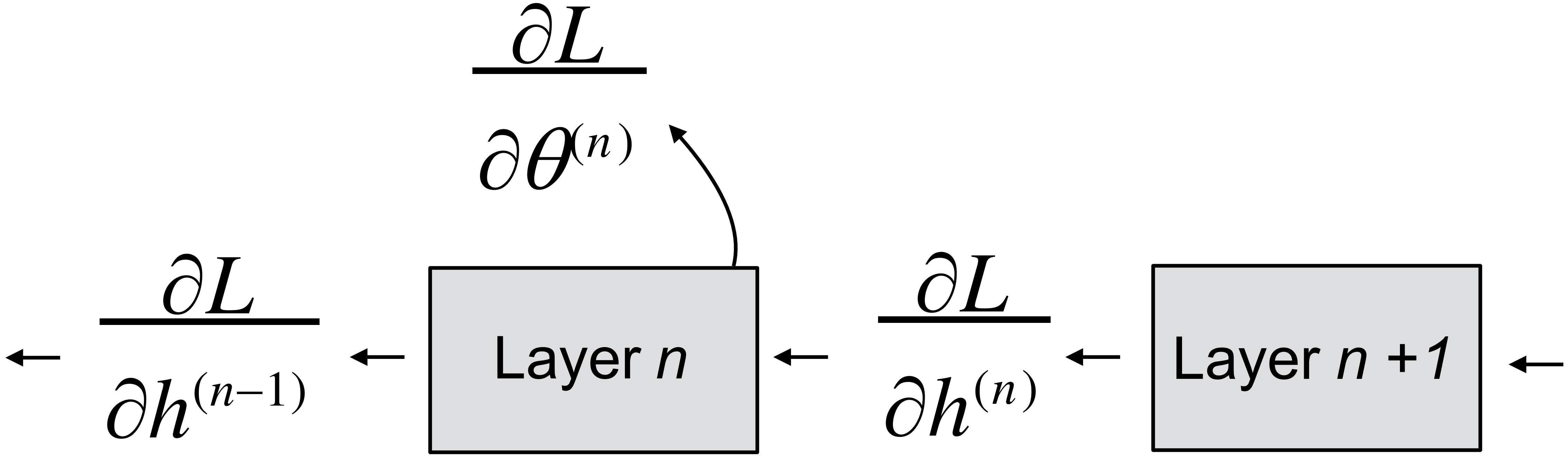
Forward Propagation:



Backward Propagation:



What to do for each layer



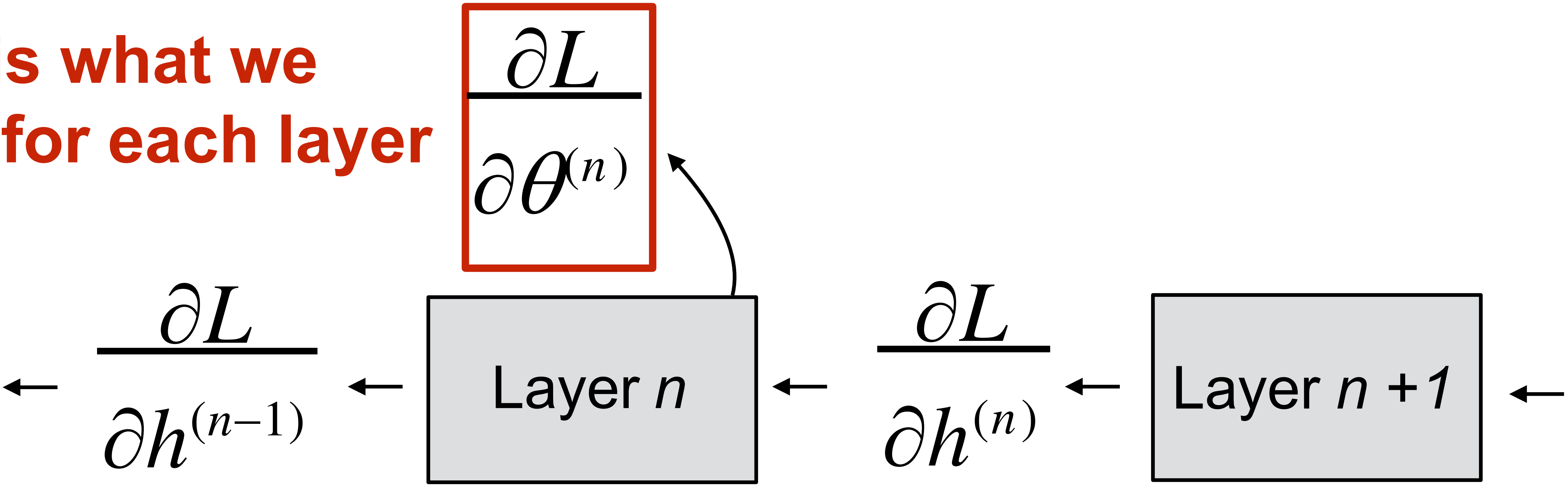
$\frac{\partial L}{\partial \theta^{(n)}}$

$\frac{\partial L}{\partial h^{(n-1)}}$

Layer *n*

$\frac{\partial L}{\partial h^{(n)}}$

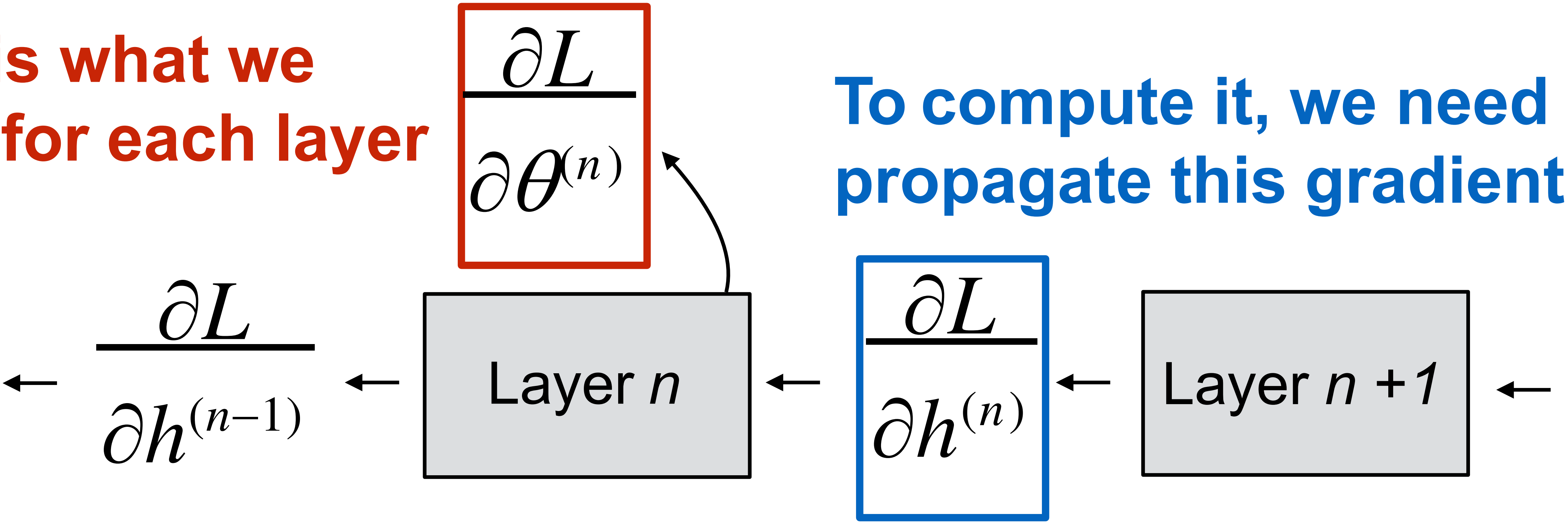
Layer *n* +1



This is what we want for each layer

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}}$$

$$\leftarrow \frac{\partial L}{\partial h^{(n-1)}} \leftarrow \text{Layer } n \leftarrow \frac{\partial L}{\partial h^{(n)}} \leftarrow \text{Layer } n+1 \leftarrow$$



This is what we want for each layer

$$\frac{\partial L}{\partial \theta^{(n)}}$$

To compute it, we need to propagate this gradient

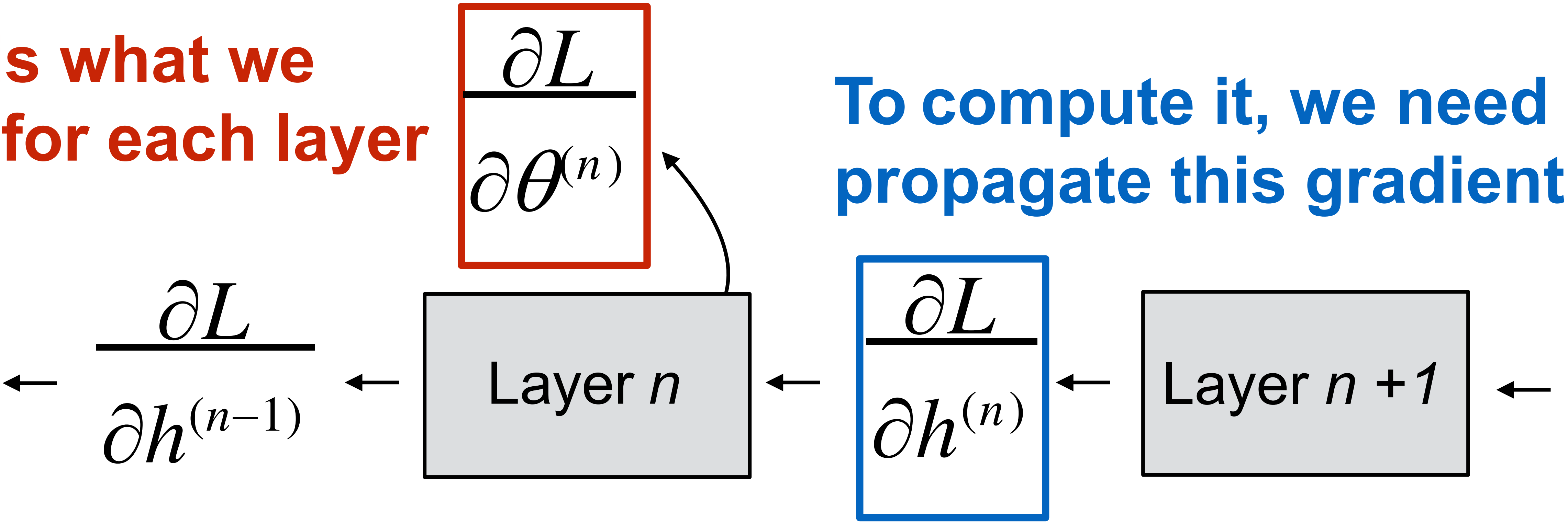
$$\frac{\partial L}{\partial h^{(n)}}$$

$$\frac{\partial L}{\partial h^{(n-1)}}$$

This is what we want for each layer

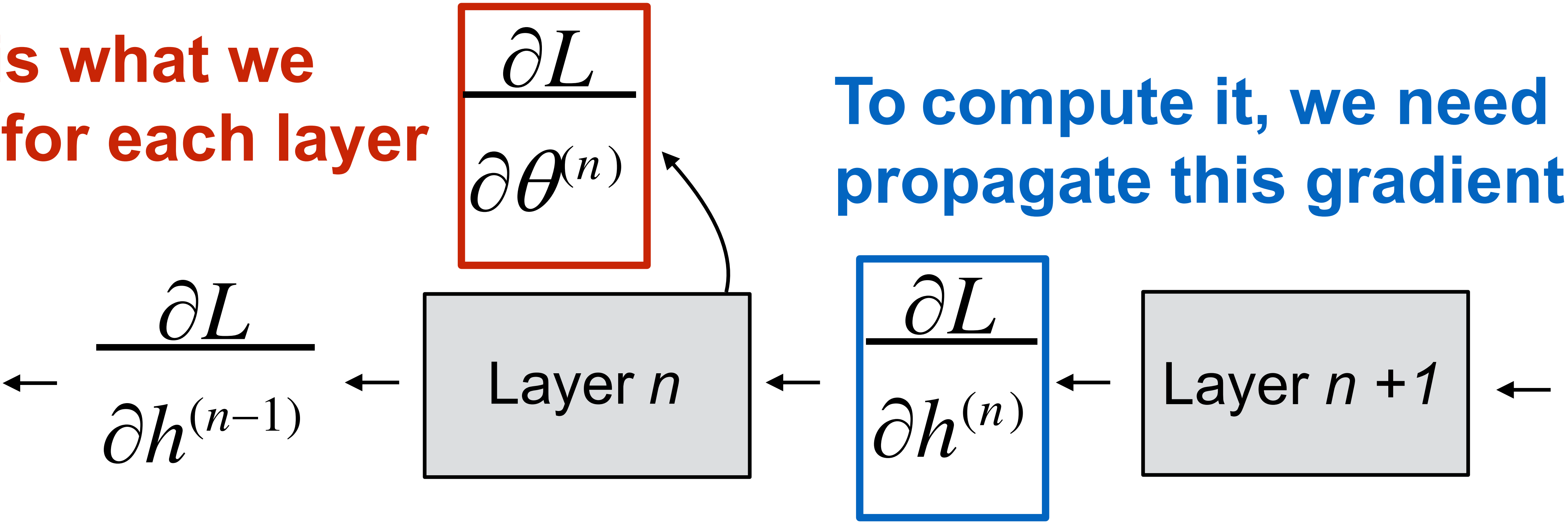
$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}}$$

To compute it, we need to propagate this gradient



$$\leftarrow \frac{\partial L}{\partial h^{(n-1)}} \leftarrow \text{Layer } n \leftarrow \boxed{\frac{\partial L}{\partial h^{(n)}}} \leftarrow \text{Layer } n+1 \leftarrow$$

For each layer:



This is what we want for each layer

$\boxed{\frac{\partial L}{\partial \theta^{(n)}}}$

To compute it, we need to propagate this gradient

$\leftarrow \frac{\partial L}{\partial h^{(n-1)}} \leftarrow$ Layer n $\leftarrow \boxed{\frac{\partial L}{\partial h^{(n)}}} \leftarrow$ Layer $n+1$ $\leftarrow$

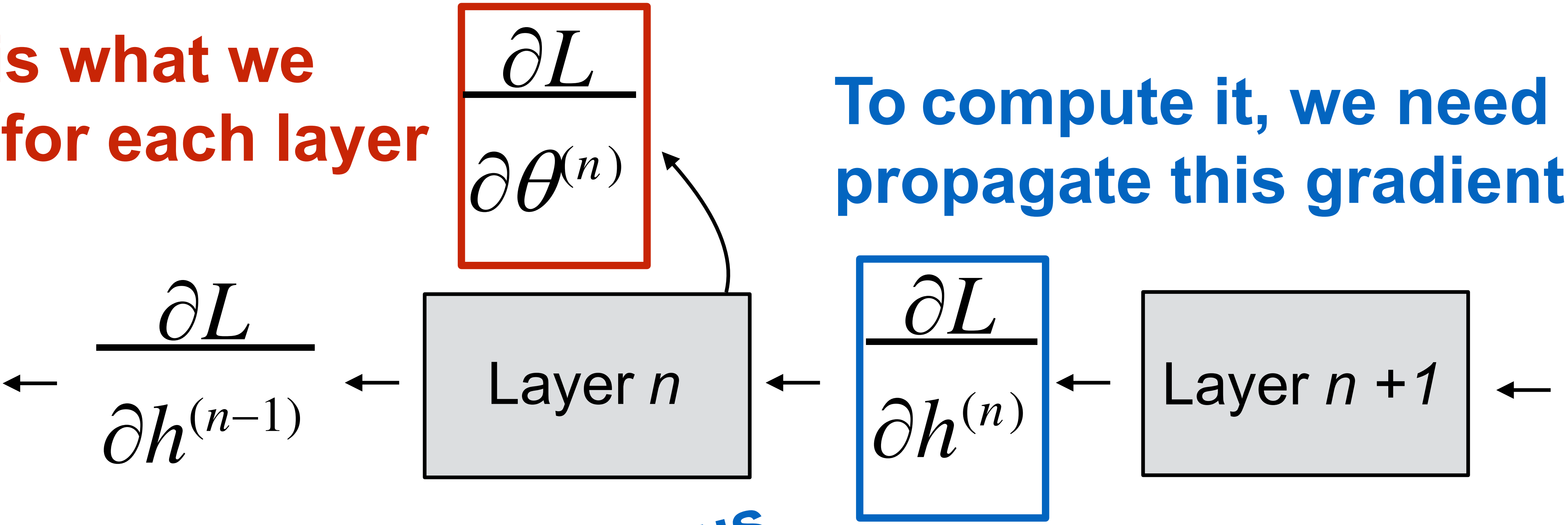
For each layer:

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \frac{\partial L}{\partial h^{(n)}} \cdot \frac{\partial h^{(n)}}{\partial \theta^{(n)}}$$

What we want

This is what we want for each layer

To compute it, we need to propagate this gradient



[redacted]

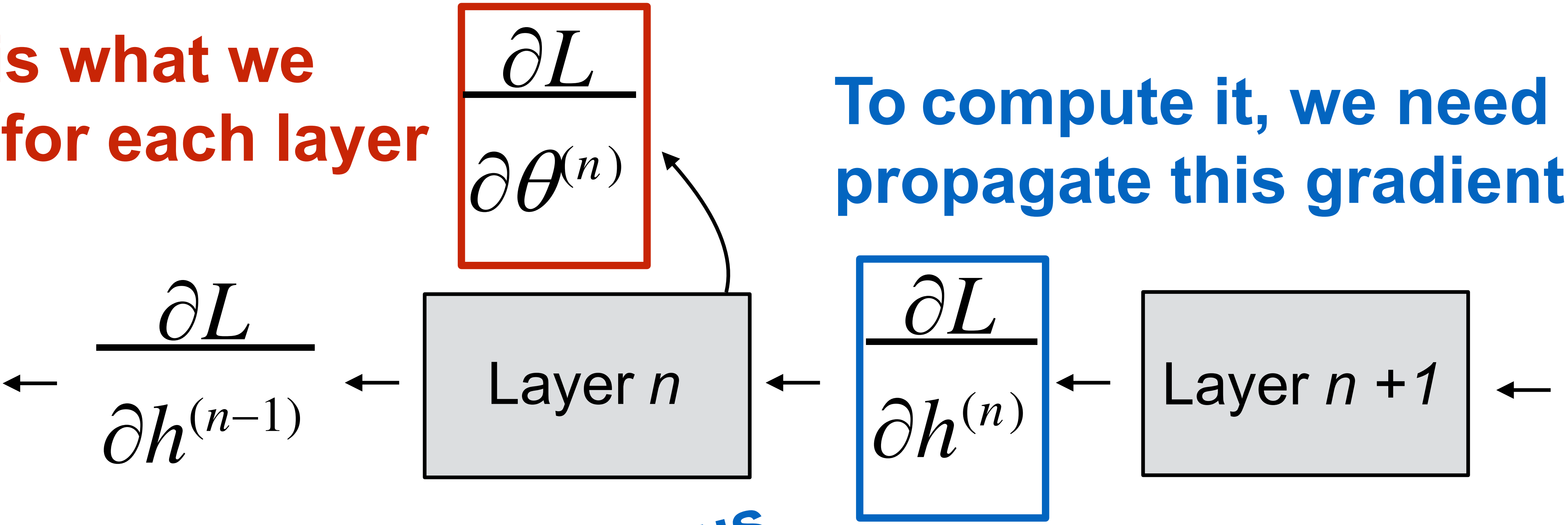
For each layer:

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \frac{\partial h^{(n)}}{\partial \theta^{(n)}}$$

What we want

This is what we want for each layer

To compute it, we need to propagate this gradient



For each layer: [redacted]

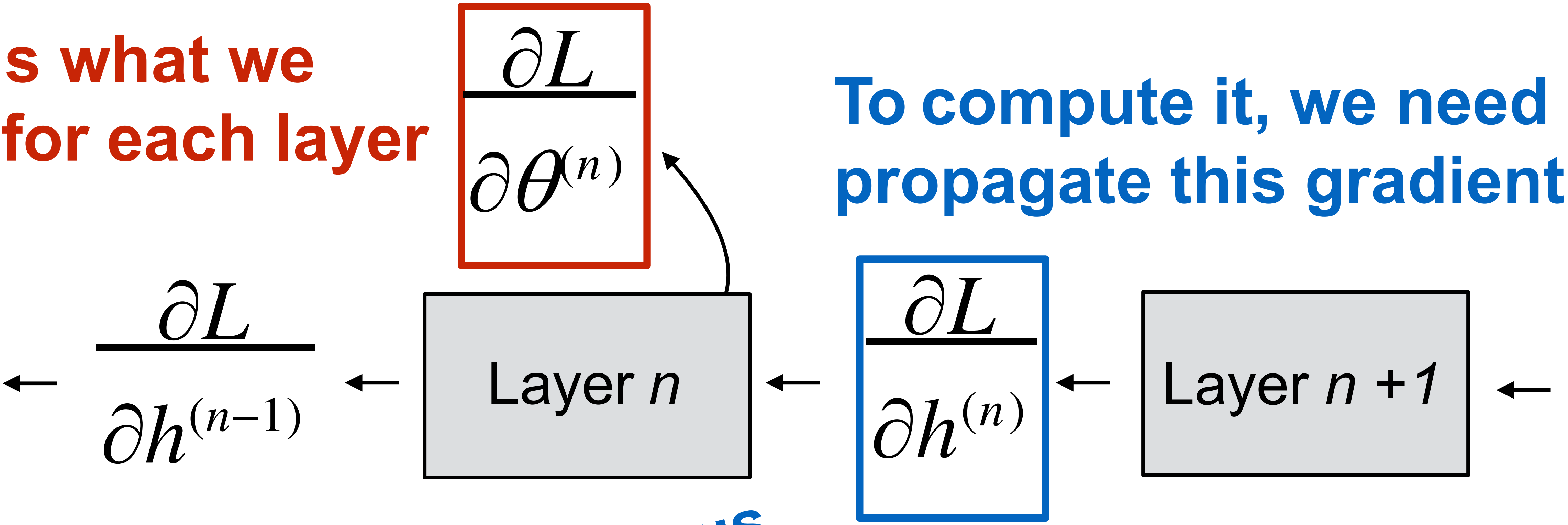
$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \boxed{\frac{\partial h^{(n)}}{\partial \theta^{(n)}}}$$

What we want

[redacted]

This is just the local gradient of layer *n*

This is what we want for each layer



To compute it, we need to propagate this gradient

For each layer: [redacted]

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \boxed{\frac{\partial h^{(n)}}{\partial \theta^{(n)}}}$$

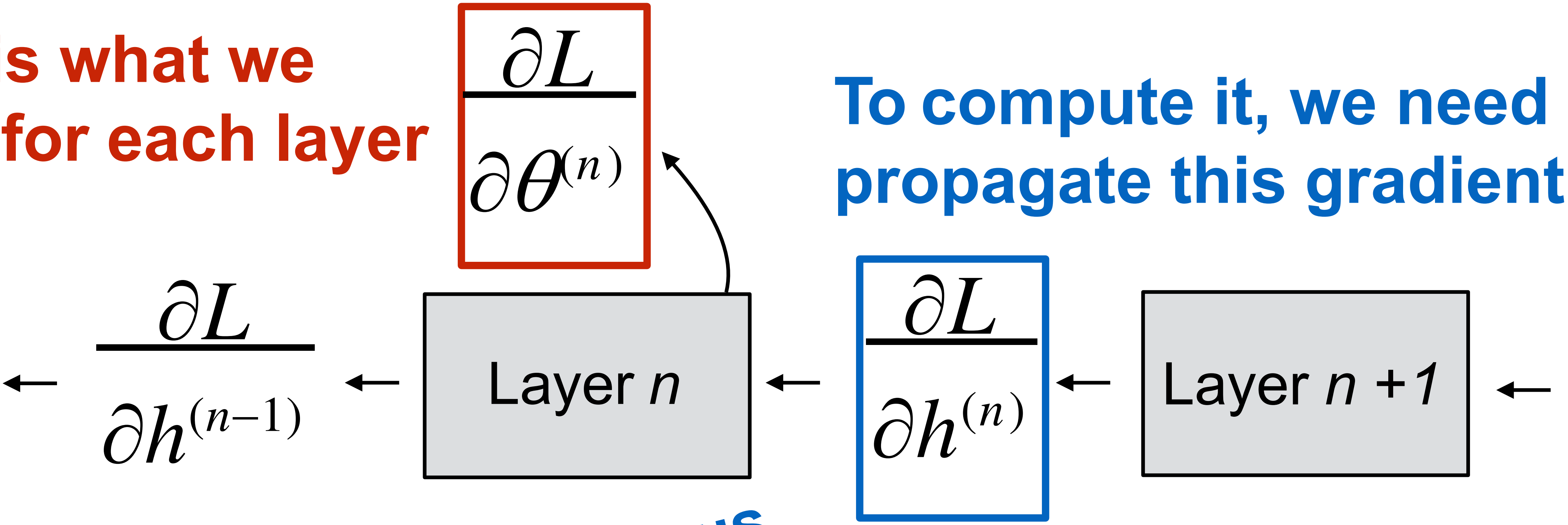
What we want

[redacted]

This is just the local gradient of layer *n*

$$\frac{\partial L}{\partial h^{(n-1)}} = \frac{\partial L}{\partial h^{(n)}} \cdot \frac{\partial h^{(n)}}{\partial h^{(n-1)}}$$

This is what we want for each layer



To compute it, we need to propagate this gradient

[redacted]

[redacted]

For each layer:

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \boxed{\frac{\partial h^{(n)}}{\partial \theta^{(n)}}} \qquad \frac{\partial L}{\partial h^{(n-1)}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \frac{\partial h^{(n)}}{\partial h^{(n-1)}}$$

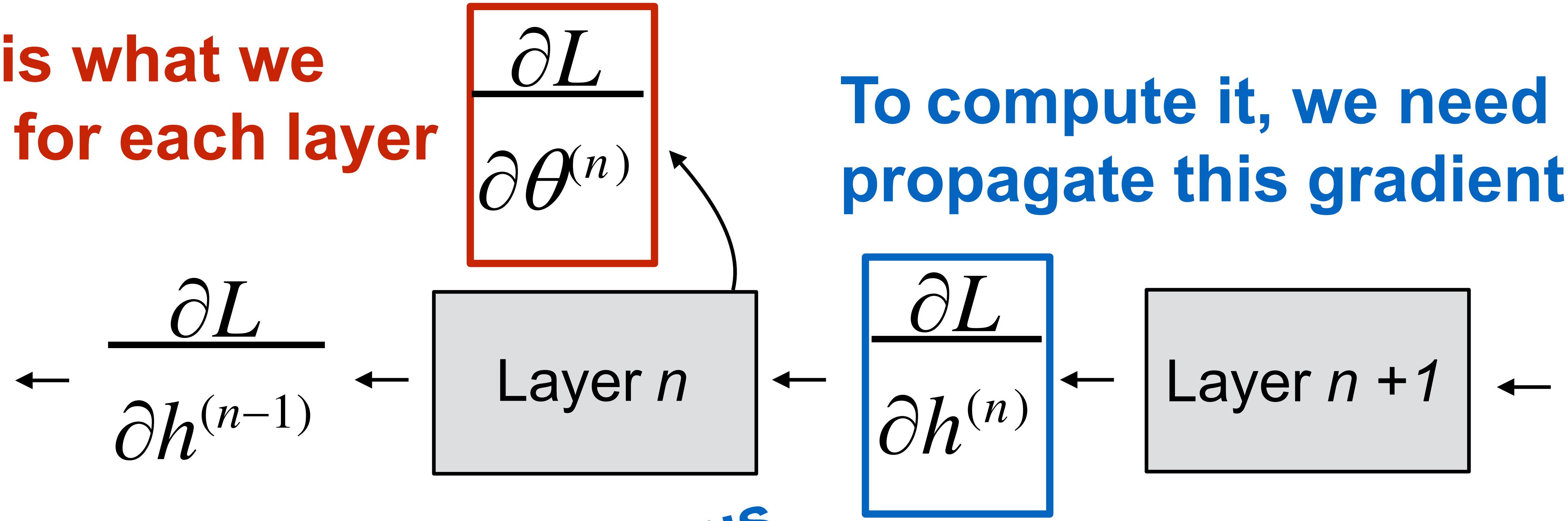
What we want

[redacted]

This is just the local gradient of layer *n*

This is what we want for each layer

To compute it, we need to propagate this gradient



For each layer: [redacted] [redacted]

$$\boxed{\frac{\partial L}{\partial \theta^{(n)}}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \boxed{\frac{\partial h^{(n)}}{\partial \theta^{(n)}}} \qquad \frac{\partial L}{\partial h^{(n-1)}} = \boxed{\frac{\partial L}{\partial h^{(n)}}} \cdot \boxed{\frac{\partial h^{(n)}}{\partial h^{(n-1)}}}$$

What we want

[redacted] [redacted]

This is just the local gradient of layer *n*

Summary

For each layer, we compute:

$$\left[\text{Propagated gradient to the left}\right] = \left[\text{Propagated gradient from right}\right] \cdot \left[\text{Local gradient}\right]$$

Summary

For each layer, we compute:

$$\left[\text{Propagated gradient to the left}\right] = \left[\text{Propagated gradient from right}\right] \cdot \underset{\substack{\Big\uparrow \\ \text{(Can compute immediately)}}}{\left[\text{Local gradient}\right]}$$

Summary

For each layer, we compute:

$$\left[\text{Propagated gradient to the left}\right] = \left[\text{Propagated gradient from right}\right]\cdot\left[\text{Local gradient}\right]$$

(Received during backprop)

(Can compute immediately)

30s cat picture break

http://stylonica.com/cat-pictures/

Backprop in N-dimensions

just add more subscripts and more summations

Backprop in N-dimensions

just add more subscripts and more summations

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

x, h scalars
(L is always scalar)

Backprop in N-dimensions

just add more subscripts and more summations

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

x, h scalars
(L is always scalar)

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

x, h 1D arrays (vectors)

Backprop in N-dimensions

just add more subscripts and more summations

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

x, h scalars
(L is always scalar)

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

x, h 1D arrays (vectors)

$$\frac{\partial L}{\partial x_{ab}} = \sum_i \sum_j \frac{\partial L}{\partial h_{ij}}\frac{\partial h_{ij}}{\partial x_{ab}}$$

x, h 2D arrays

Backprop in N-dimensions

just add more subscripts and more summations

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

x, h scalars
(L is always scalar)

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

x, h 1D arrays (vectors)

$$\frac{\partial L}{\partial x_{ab}} = \sum_i \sum_j \frac{\partial L}{\partial h_{ij}}\frac{\partial h_{ij}}{\partial x_{ab}}$$

x, h 2D arrays

$$\frac{\partial L}{\partial x_{abc}} = \sum_i \sum_j \sum_k \frac{\partial L}{\partial h_{ijk}}\frac{\partial h_{ijk}}{\partial x_{abc}}$$

x, h 3D arrays

Examples

Example: Mean Subtraction

(for a single input)

Example: Mean Subtraction

(for a single input)

- Example layer: mean subtraction:

Example: Mean Subtraction

(for a single input)

- Example layer: mean subtraction:

$$h_i = x_i - \frac{1}{D} \sum_k x_k$$

Example: Mean Subtraction

(for a single input)

- Example layer: mean subtraction:

$$h_i = x_i - \frac{1}{D} \sum_k x_k$$

(here, “i” and “k” are channels)

Example: Mean Subtraction

(for a single input)

- Example layer: mean subtraction:

$$h_i = x_i - \frac{1}{D}\sum_k x_k$$

(here, “i” and “k” are channels)

- Always start with the chain rule (this one is for 1D):

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

Example: Mean Subtraction

(for a single input)

- Example layer: mean subtraction:

$$h_i = x_i - \frac{1}{D}\sum_k x_k$$

(here, "i" and "k" are channels)

- Always start with the chain rule (this one is for 1D):

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

- **Note:** Be very careful with your subscripts! Introduce new variables and don't re-use letters.

Example: Mean Subtraction

(for a single input)

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer:

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

(backprop aka chain rule)

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

(backprop aka chain rule)

$$= \sum_i \frac{\partial L}{\partial h_i}\left(\delta_{ij} - \frac{1}{D}\right)$$

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

Example: Mean Subtraction
(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

(backprop aka chain rule)

$$= \sum_i \frac{\partial L}{\partial h_i}\left(\delta_{ij} - \frac{1}{D}\right)$$

$$= \sum_i \frac{\partial L}{\partial h_i}\delta_{ij} - \frac{1}{D}\sum_i \frac{\partial L}{\partial h_i}$$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

(backprop aka chain rule)

$$= \sum_i \frac{\partial L}{\partial h_i}\left(\delta_{ij} - \frac{1}{D}\right)$$

$$= \sum_i \frac{\partial L}{\partial h_i}\delta_{ij} - \frac{1}{D}\sum_i \frac{\partial L}{\partial h_i}$$

$$= \frac{\partial L}{h} - \frac{1}{D}\sum_i \frac{\partial L}{h}$$

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Taking the derivative of the layer: $\frac{\partial h_i}{\partial x_j} = \delta_{ij} - \frac{1}{D}$

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

$$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial h_i}\frac{\partial h_i}{\partial x_j}$$

(backprop aka chain rule)

$$= \sum_i \frac{\partial L}{\partial h_i}\left(\delta_{ij} - \frac{1}{D}\right)$$

$$= \sum_i \frac{\partial L}{\partial h_i}\delta_{ij} - \frac{1}{D}\sum_i \frac{\partial L}{\partial h_i}$$

$$= \frac{\partial L}{h} - \frac{1}{D}\sum_i \frac{\partial L}{h}$$

Done!

Example: Mean Subtraction

(for a single input)

$$h_i = x_i - \frac{1}{D}\sum_k x_k$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial h_i} - \frac{1}{D}\sum_k \frac{\partial L}{\partial h_k}$$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Backward: $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial h_i} - \frac{1}{D}\sum_k \frac{\partial L}{\partial h_k}$

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Backward: $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial h_i} - \frac{1}{D}\sum_k \frac{\partial L}{\partial h_k}$

- In this case, they're identical operations!

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$
- Backward: $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial h_i} - \frac{1}{D}\sum_k \frac{\partial L}{\partial h_k}$

- In this case, they’re identical operations!
- Usually the forwards pass and backwards pass are similar **but not the same**.

Example: Mean Subtraction

(for a single input)

- Forward: $$h_i = x_i - \frac{1}{D}\sum_k x_k$$
- Backward: $$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial h_i} - \frac{1}{D}\sum_k \frac{\partial L}{\partial h_k}$$

- In this case, they're identical operations!

- Usually the forwards pass and backwards pass are similar **but not the same**.

- Derive it by hand, and check it numerically

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$

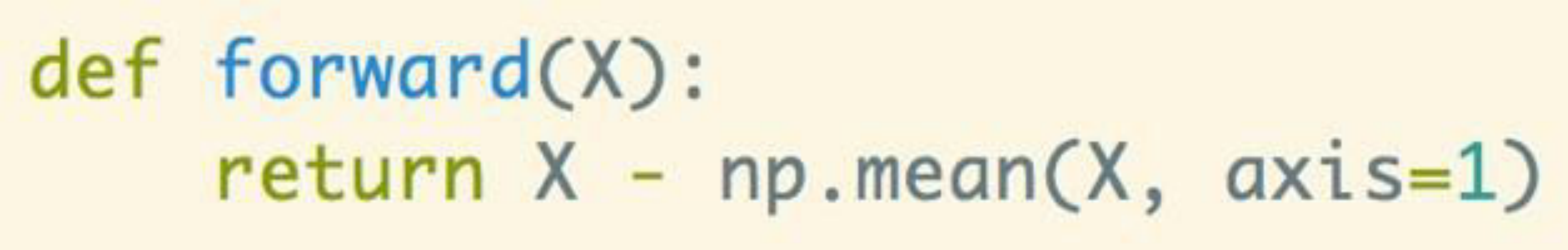
Let’s code this up in NumPy:

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$

Let’s code this up in NumPy:



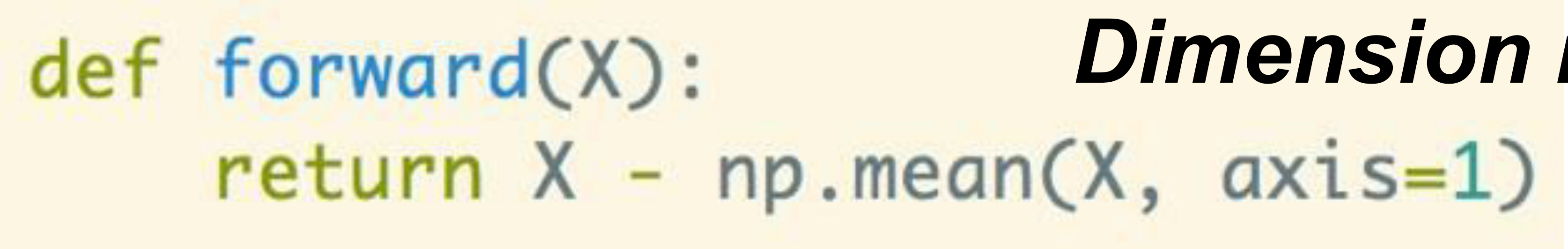
```
def forward(X):
    return X - np.mean(X, axis=1)
```

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$

Let’s code this up in NumPy:



```
def forward(X):
    return X - np.mean(X, axis=1)
```

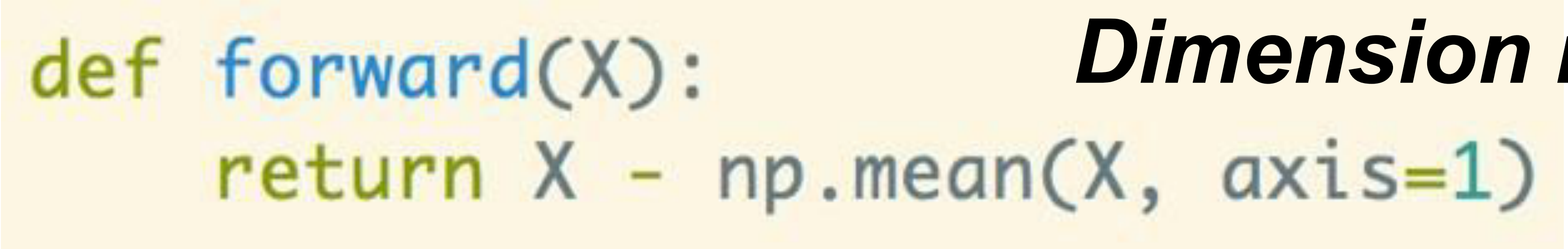
Dimension mismatch

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D} \sum_k x_k$

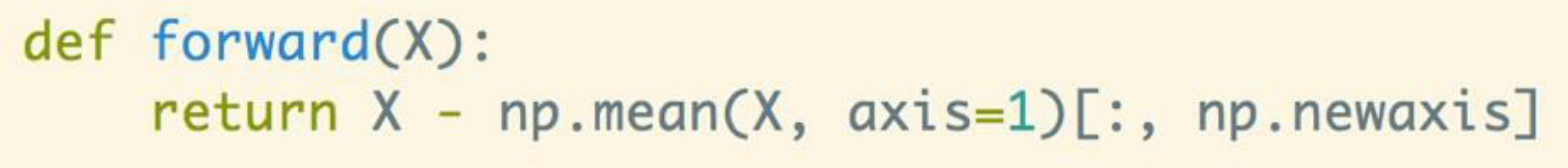
Let’s code this up in NumPy:



```
def forward(X):
    return X - np.mean(X, axis=1)
```

Dimension mismatch

You need to broadcast properly:



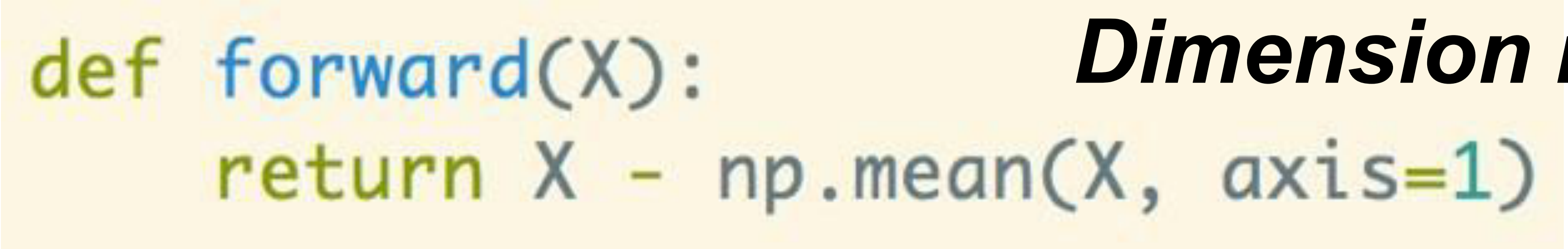
```
def forward(X):
    return X - np.mean(X, axis=1)[:, np.newaxis]
```

Example: Mean Subtraction

(for a single input)

- Forward: $h_i = x_i - \frac{1}{D}\sum_k x_k$

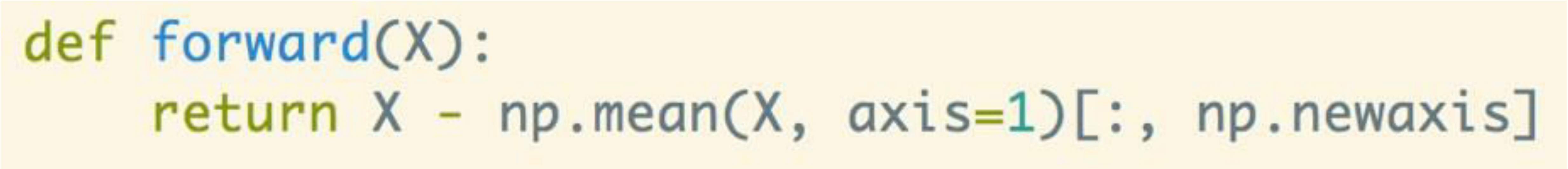
Let’s code this up in NumPy:



```
def forward(X):
    return X - np.mean(X, axis=1)
```

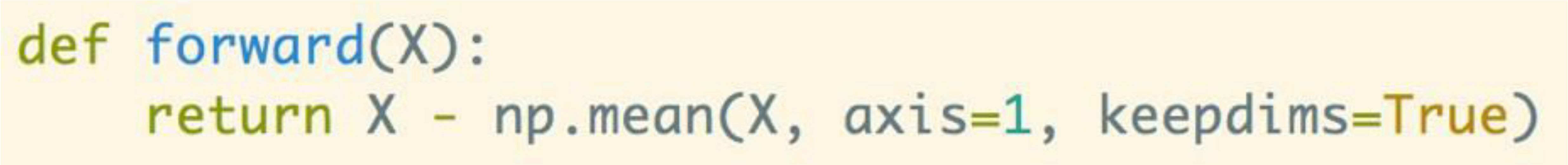
Dimension mismatch

You need to broadcast properly:



```
def forward(X):
    return X - np.mean(X, axis=1)[:, np.newaxis]
```

This also works:

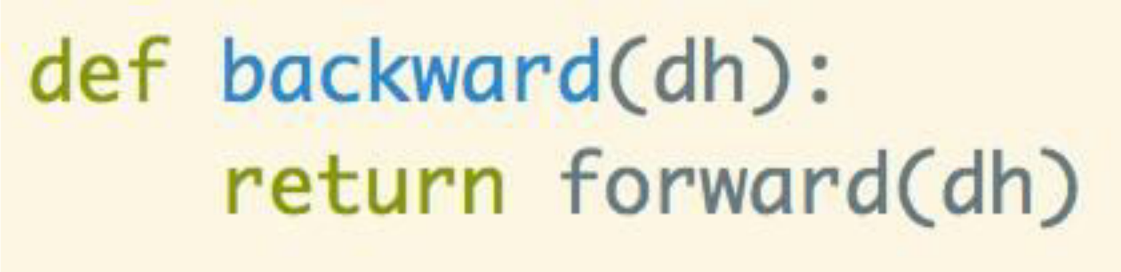


```
def forward(X):
    return X - np.mean(X, axis=1, keepdims=True)
```

Example: Mean Subtraction

(for a single input)

The backward pass is easy:



```
def backward(dh):
    return forward(dh)
```

(Remember they're usually not the same)

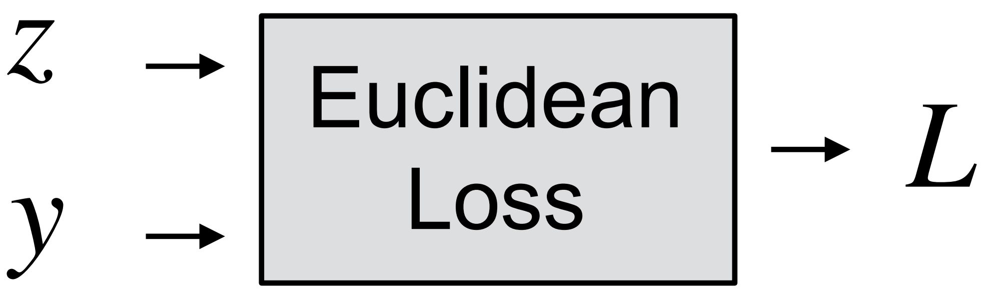
Example: Euclidean Loss

Example: Euclidean Loss

- Euclidean loss layer:

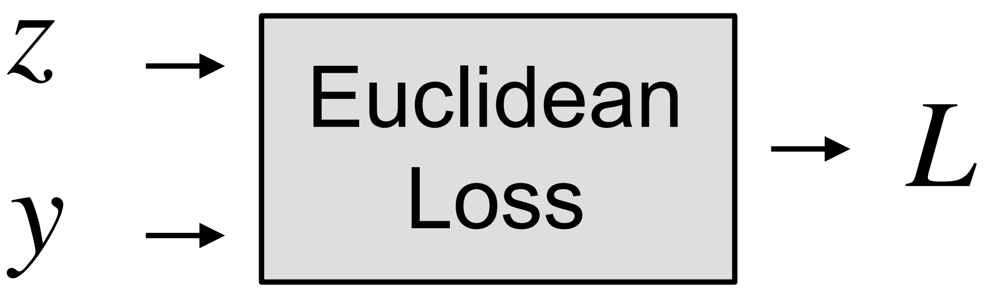
Example: Euclidean Loss

- Euclidean loss layer:



Example: Euclidean Loss

- Euclidean loss layer:



z →

y →

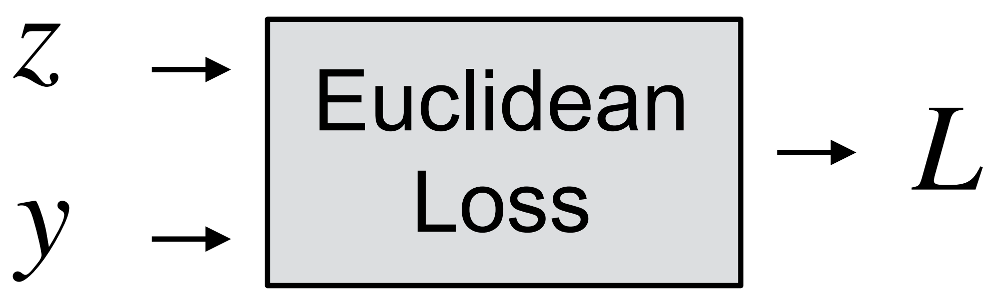
Euclidean Loss

→ L

$$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

Example: Euclidean Loss

- Euclidean loss layer:



z

y

Euclidean Loss

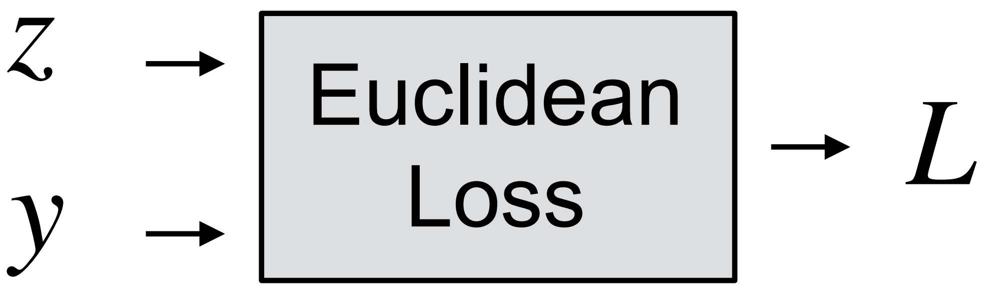
L

$$L_i = \frac{1}{2} \sum_j (z_{i,j} - y_{i,j})^2$$

("i" is the batch index,
"j" is the channel)

Example: Euclidean Loss

- Euclidean loss layer:



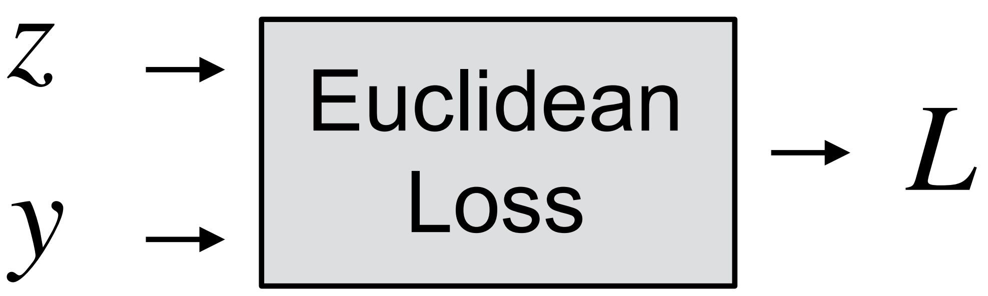
$$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

(“i” is the batch index,
“j” is the channel)

- The total loss is the average over N examples:

Example: Euclidean Loss

- Euclidean loss layer:



$$L_i = \frac{1}{2} \sum_j (z_{i,j} - y_{i,j})^2$$

("i" is the batch index,
"j" is the channel)

- The total loss is the average over N examples:

$$L = \frac{1}{N} \sum_i L_i$$

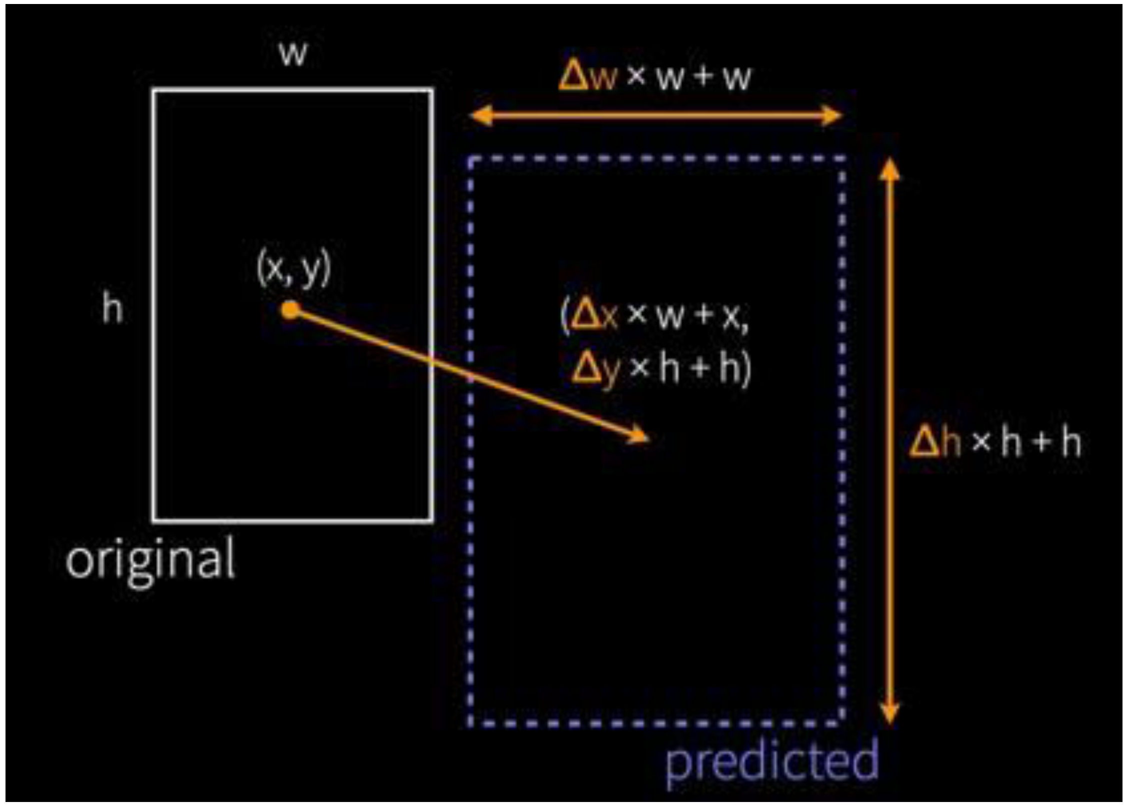
Example: Euclidean Loss

Example: Euclidean Loss

- Used for **regression**, e.g. predicting an adjustment to box coordinates when detecting objects:

Example: Euclidean Loss

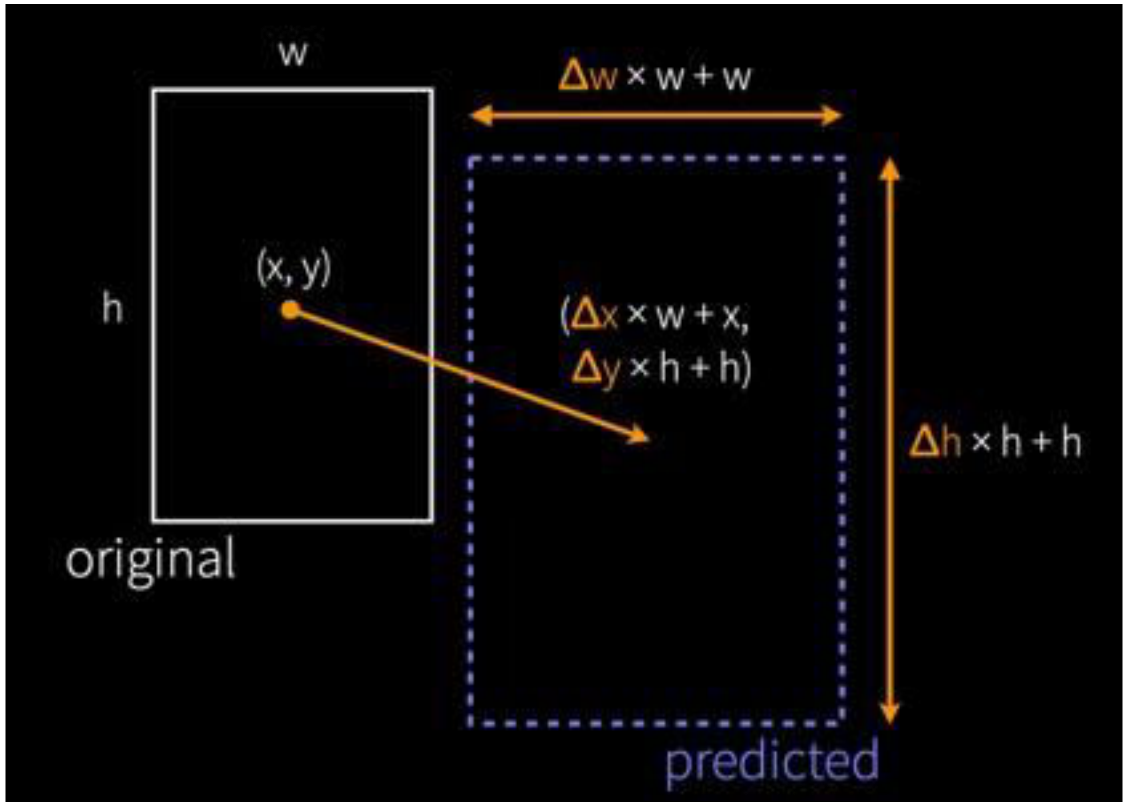
- Used for **regression**, e.g. predicting an adjustment to box coordinates when detecting objects:



Bounding box regression from the R-CNN object detector [Girshick 2014]

Example: Euclidean Loss

- Used for **regression**, e.g. predicting an adjustment to box coordinates when detecting objects:



Bounding box regression from the R-CNN object detector [Girshick 2014]

- **Note:** Can be unstable and other losses often work better. Alternatives: L1 distance (instead of L2), discretizing into category bins and using softmax

Example: Euclidean Loss

Example: Euclidean Loss

- Forward: $$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

Example: Euclidean Loss

- Forward: $L_i = \frac{1}{2} \sum_{j} (z_{i,j} - y_{i,j})^2$

- Backward:

Example: Euclidean Loss

- Forward: $$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward: $$\frac{\partial L_i}{\partial z_{i,j}} = z_{i,j} - y_{i,j}$$

Example: Euclidean Loss

- Forward: $$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward: $$\frac{\partial L_i}{\partial z_{i,j}} = z_{i,j} - y_{i,j}$$

$$\frac{\partial L_i}{\partial y_{i,j}} = y_{i,j} - z_{i,j}$$

Example: Euclidean Loss

- Forward: $$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward: $$\frac{\partial L_i}{\partial z_{i,j}} = z_{i,j} - y_{i,j}$$

$$\frac{\partial L_i}{\partial y_{i,j}} = y_{i,j} - z_{i,j}$$

- **Q:** If you scale the loss by *C*, what happens to gradient computed in the backwards pass?

Example: Euclidean Loss

- Forward: $$L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$
- Backward: $$\frac{\partial \boxed{L_i}}{\partial z_{i,j}} = z_{i,j} - y_{i,j}$$

$$\frac{\partial L_i}{\partial y_{i,j}} = y_{i,j} - z_{i,j}$$

(note that this is with respect to Li, not L)

- **Q:** If you scale the loss by *C*, what happens to gradient computed in the backwards pass?

Example: Euclidean Loss

Example: Euclidean Loss

- Forward pass, for a batch of N inputs:

Example: Euclidean Loss

- Forward pass, for a batch of N inputs:

$$L = \frac{1}{N}\sum_i L_i \qquad L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

Example: Euclidean Loss

- Forward pass, for a batch of N inputs:

$$L = \frac{1}{N}\sum_i L_i \qquad L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward pass:

Example: Euclidean Loss

- Forward pass, for a batch of N inputs:

$$L = \frac{1}{N}\sum_i L_i \qquad L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward pass:

$$\frac{\partial L}{\partial x_{i,j}} = \frac{z_{i,j} - y_{i,j}}{N} \qquad \frac{\partial L}{\partial y_{i,j}} = \frac{y_{i,j} - z_{i,j}}{N}$$

Example: Euclidean Loss

- Forward pass, for a batch of N inputs:

$$L = \frac{1}{N}\sum_i L_i \qquad L_i = \frac{1}{2}\sum_j (z_{i,j} - y_{i,j})^2$$

- Backward pass:

$$\frac{\partial L}{\partial x_{i,j}} = \frac{z_{i,j} - y_{i,j}}{N} \qquad \frac{\partial L}{\partial y_{i,j}} = \frac{y_{i,j} - z_{i,j}}{N}$$

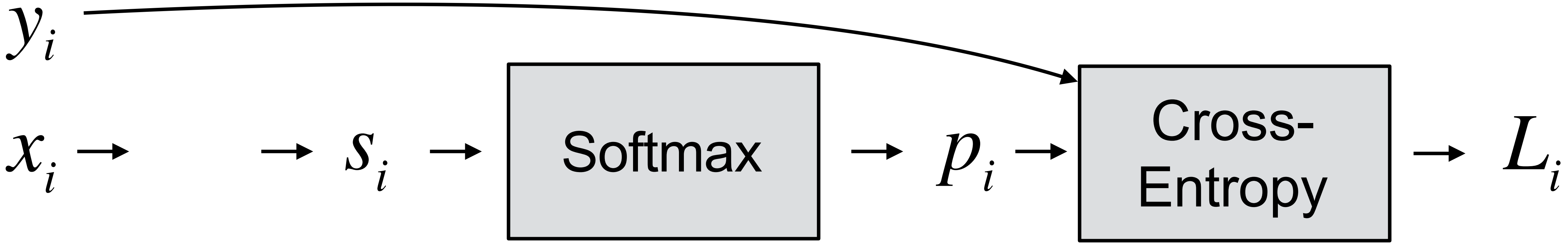
(You should be able to derive this)

Example: Softmax (for N inputs)

Remember Softmax?
It’s a loss function for predicting categories?

Example: Softmax (for N inputs)

Remember Softmax?
It’s a loss function for predicting categories?

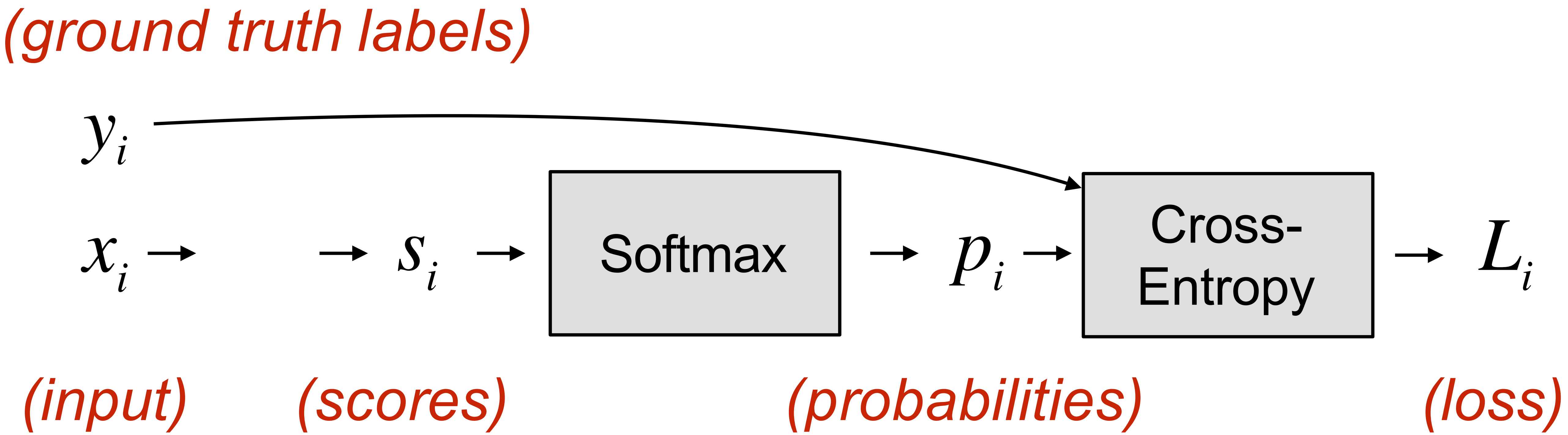


y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

Example: Softmax (for N inputs)

Remember Softmax?
It’s a loss function for predicting categories?



(ground truth labels)

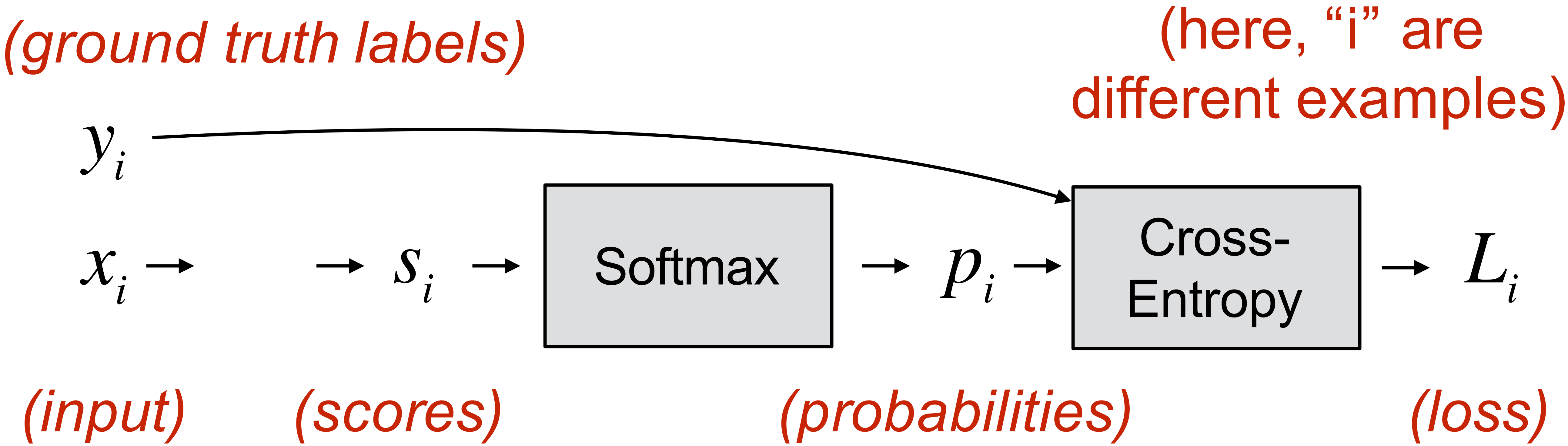
y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

(input) *(scores)* *(probabilities)* *(loss)*

Example: Softmax (for N inputs)

Remember Softmax?
It’s a loss function for predicting categories?



(ground truth labels)

(here, “i” are different examples)

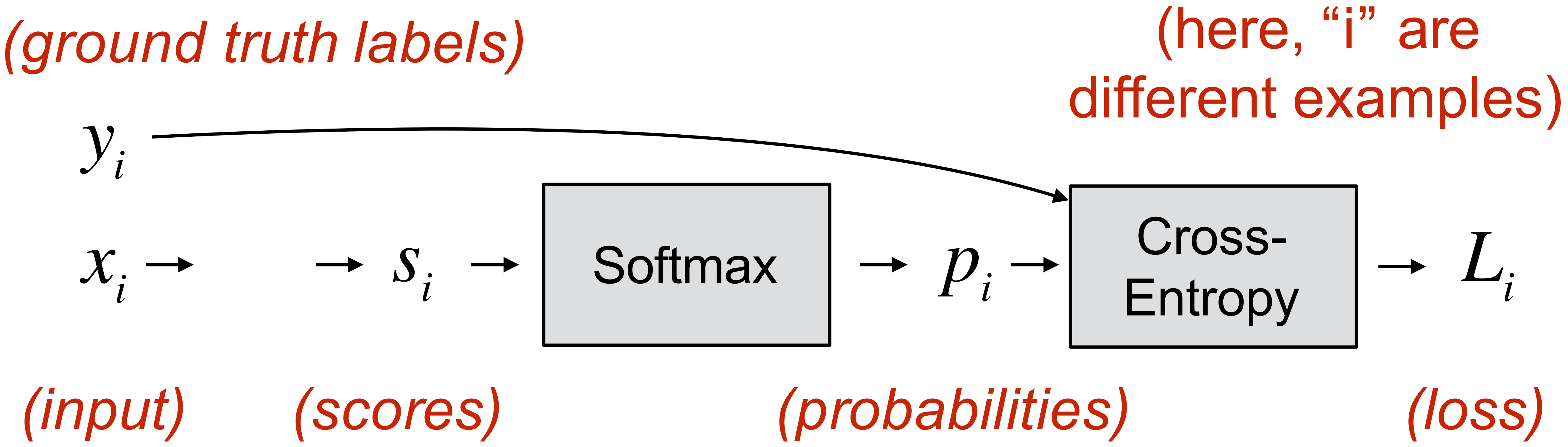
y_i

$x_i \rightarrow \qquad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

(input) *(scores)* *(probabilities)* *(loss)*

Example: Softmax (for N inputs)

Remember Softmax?
It’s a loss function for predicting categories?



(ground truth labels)

(here, “i” are different examples)

y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

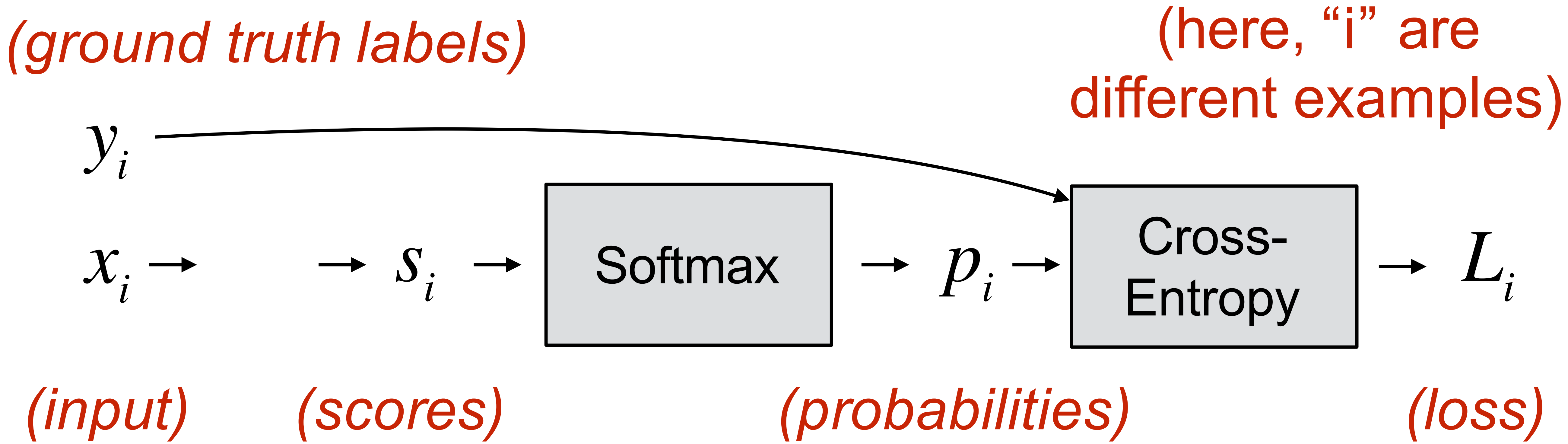
(input) *(scores)* *(probabilities)* *(loss)*

$$p_{i,j} = \frac{e^{s_{i,j}}}{\sum_k e^{s_{i,k}}}$$

(Softmax)

Example: Softmax (for N inputs)

Remember Softmax?
It's a loss function for predicting categories?



$$p_{i,j} = \frac{e^{s_{i,j}}}{\sum_k e^{s_{i,k}}}$$

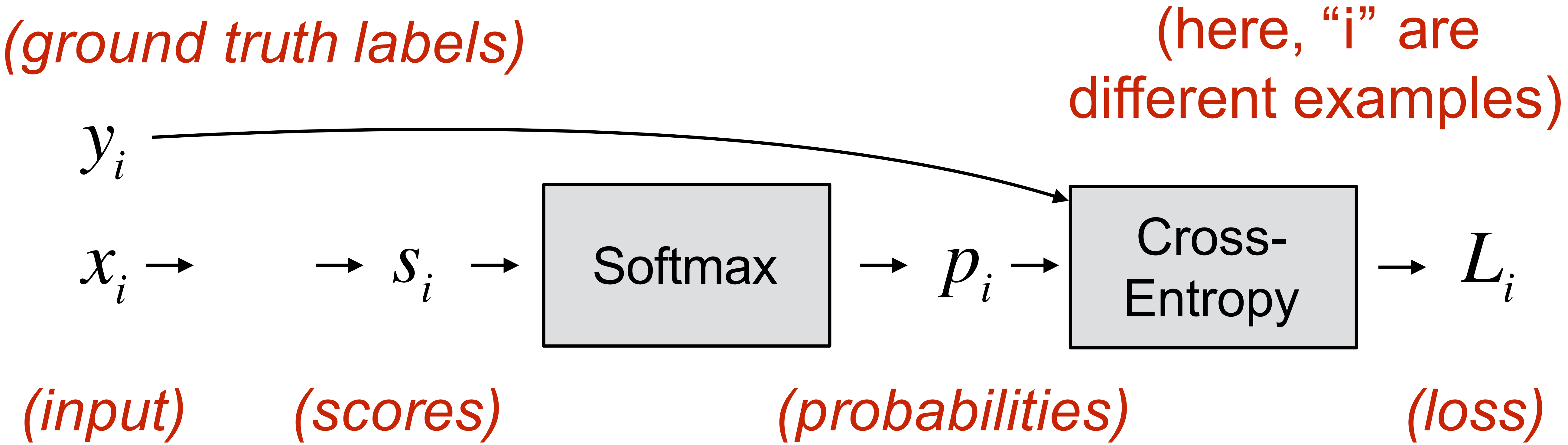
(Softmax)

$$L_i = -\log p_{i,y_i}$$

(Cross-entropy)

Example: Softmax (for N inputs)

Remember Softmax?
It's a loss function for predicting categories?



(ground truth labels)

(here, "i" are different examples)

y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

(input) *(scores)* *(probabilities)* *(loss)*

$$p_{i,j} = \frac{e^{s_{i,j}}}{\sum_k e^{s_{i,k}}}$$

(Softmax)

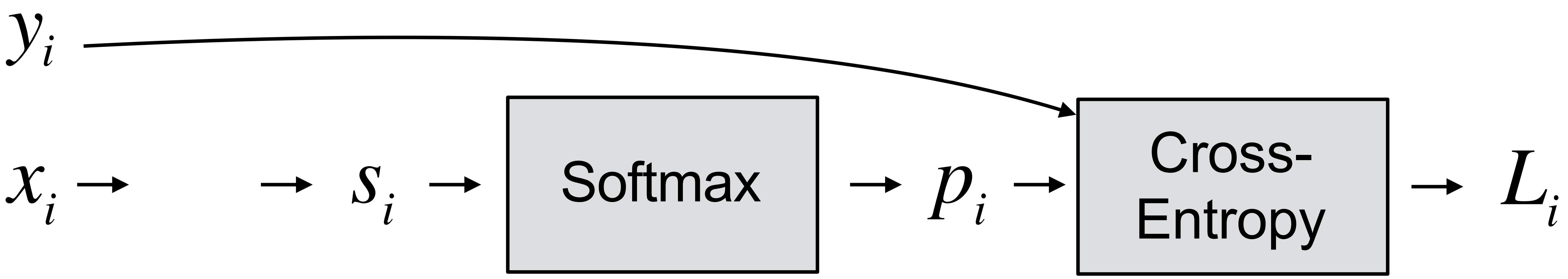
$$L_i = -\log p_{i,y_i}$$

(Cross-entropy)

$$L = \frac{1}{N} \sum_i L_i$$

(Avg. over examples)

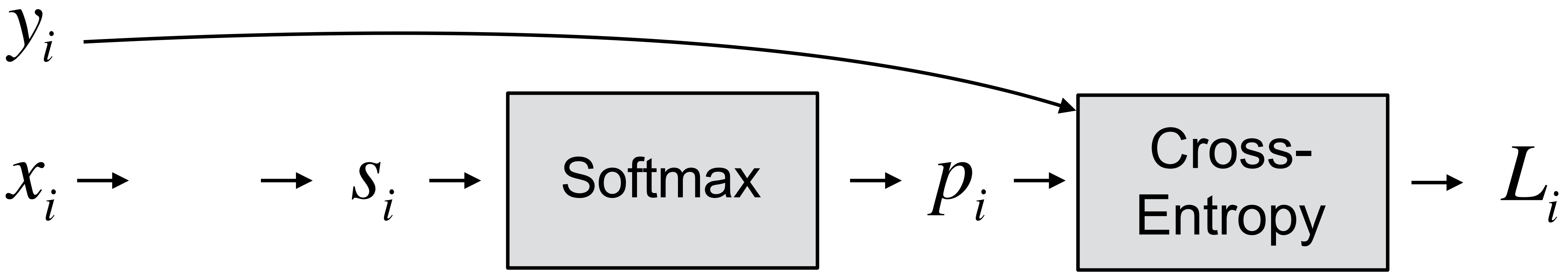
Example: Softmax (for N inputs)



y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

Example: Softmax (for N inputs)

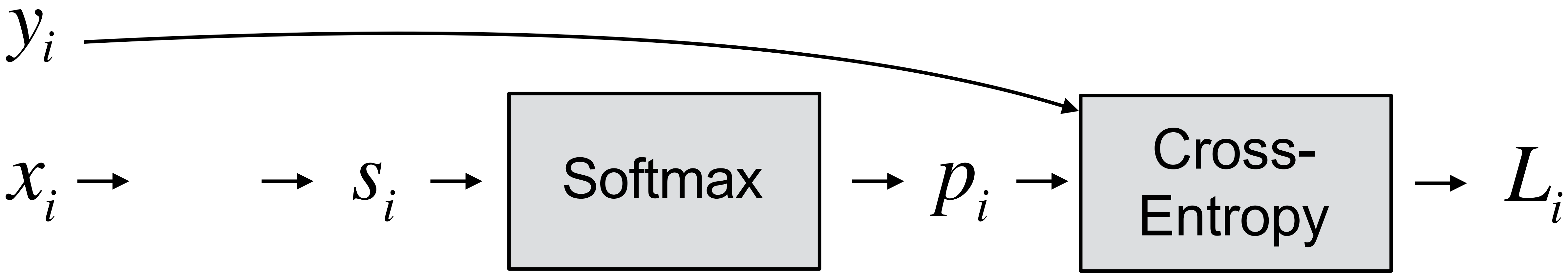


y_i

$x_i \rightarrow \quad \rightarrow s_i \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

Derivative: $\dfrac{\partial L}{\partial s_{i,j}} = \dfrac{p_{i,j} - t_{i,j}}{N}$

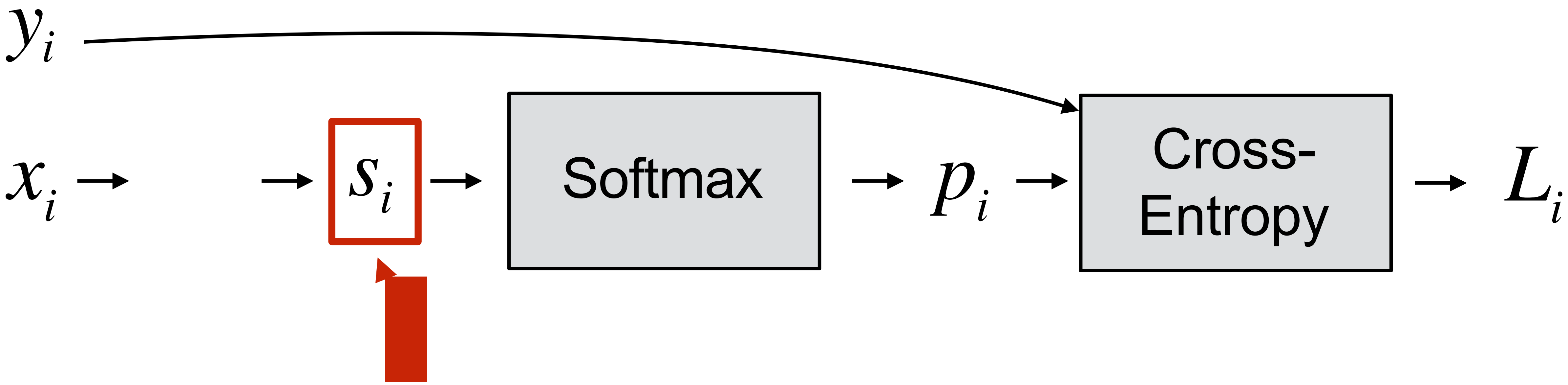
Example: Softmax (for N inputs)



Derivative: $\frac{\partial L}{\partial s_{i,j}} = \frac{p_{i,j} - t_{i,j}}{N}$ where $t_i = [0 \ldots 1 \ldots 0]$

(Entry y_i set to 1)

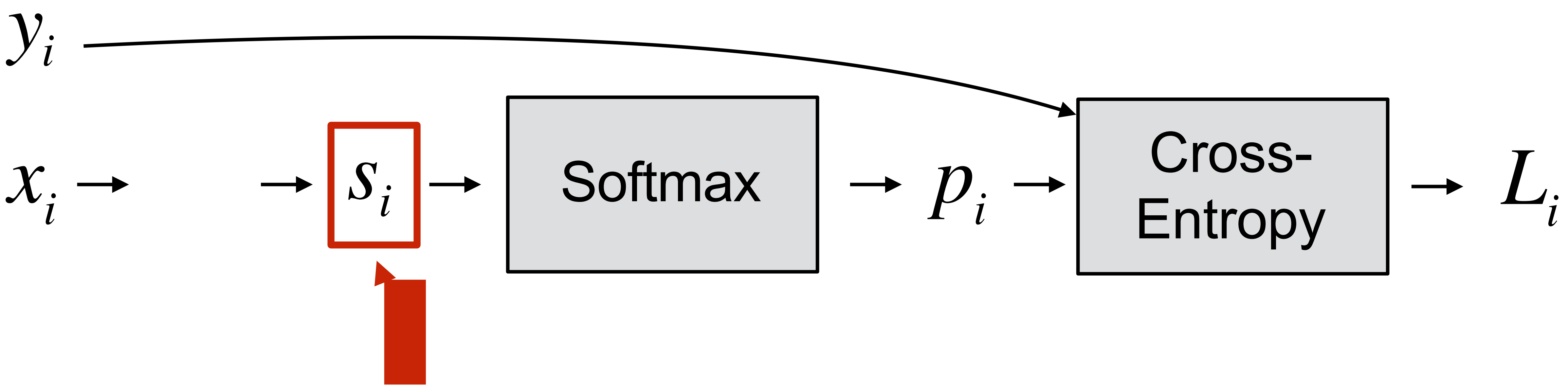
Example: Softmax (for N inputs)



Derivative: $\boxed{\frac{\partial L}{\partial s_{i,j}}} = \frac{p_{i,j} - t_{i,j}}{N}$ where $t_i = [0 \ldots 1 \ldots 0]$

(Entry y_i set to 1)

Example: Softmax (for N inputs)



y_i

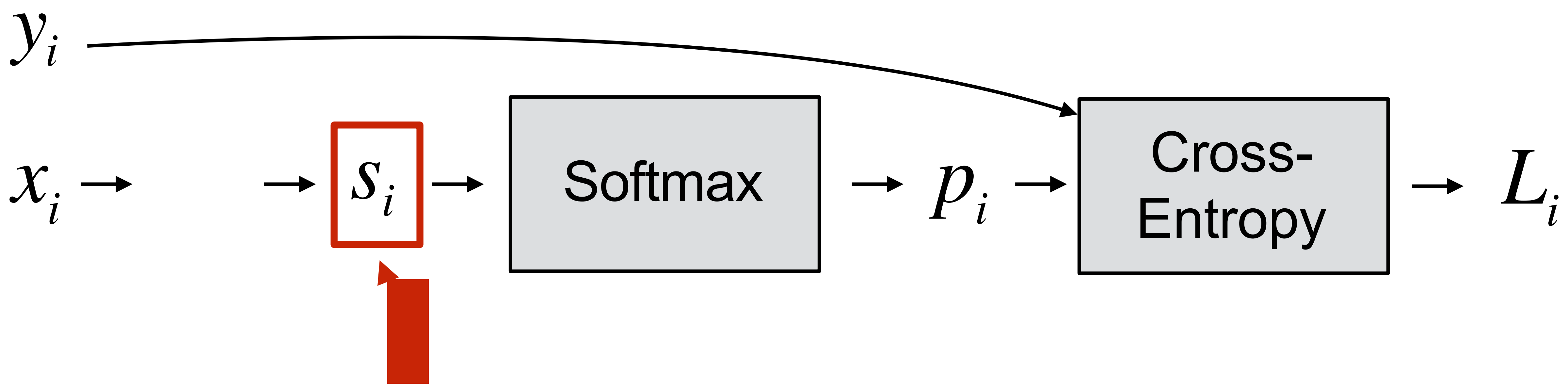
$x_i \rightarrow \qquad \rightarrow \boxed{s_i} \rightarrow$ Softmax $\rightarrow p_i \rightarrow$ Cross-Entropy $\rightarrow L_i$

Derivative: $\boxed{\dfrac{\partial L}{\partial s_{i,j}}} = \dfrac{p_{i,j} - t_{i,j}}{N}$ where $t_i = [0 \ldots 1 \ldots 0]$

(Entry y_i set to 1)

(You will derive this in PA5)

Example: Softmax (for N inputs)



Derivative: $\boxed{\dfrac{\partial L}{\partial s_{i,j}}} = \dfrac{p_{i,j} - t_{i,j}}{N}$ where $t_i = [0 \; \ldots \; 1 \; \ldots \; 0]$

(Entry y_i set to 1)

(You will derive this in PA5)

Now we can continue backpropagating to the layer before “f”

What about the weights?

To get the derivative of the weights, use the chain rule again!

What about the weights?

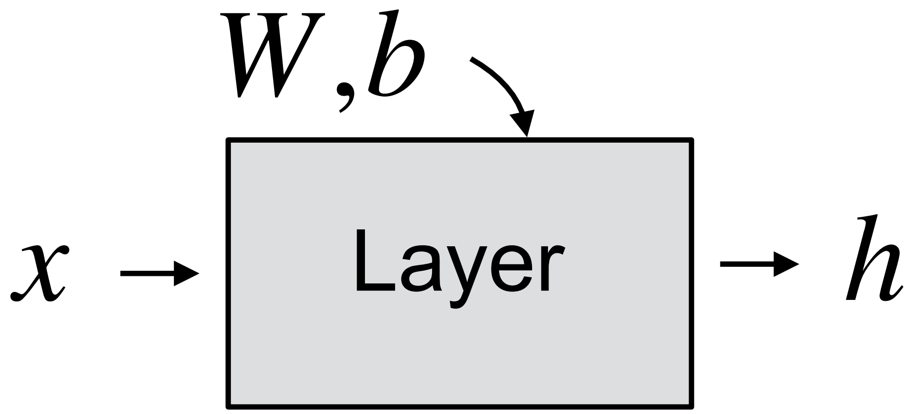
To get the derivative of the weights, use the chain rule again!

Example: 2D weights, 1D bias, 1D hidden activations:

What about the weights?

To get the derivative of the weights, use the chain rule again!

Example: 2D weights, 1D bias, 1D hidden activations:

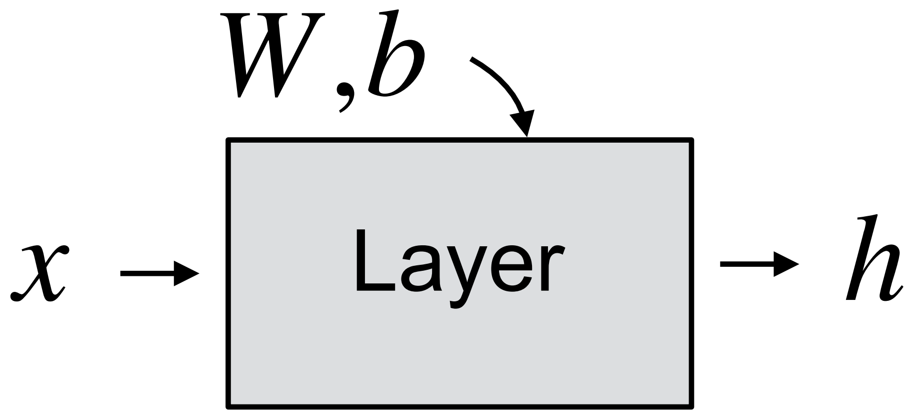


$$h = h(x; W)$$

What about the weights?

To get the derivative of the weights, use the chain rule again!

Example: 2D weights, 1D bias, 1D hidden activations:



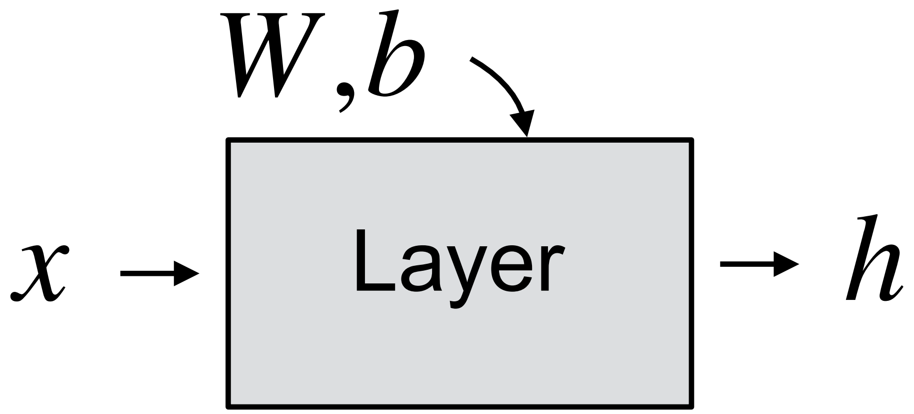
$$h = h(x; W)$$

$$\frac{\partial L}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial h_k}{\partial W_{ij}}$$

What about the weights?

To get the derivative of the weights, use the chain rule again!

Example: 2D weights, 1D bias, 1D hidden activations:



W, b

$x \rightarrow$ Layer $\rightarrow h$

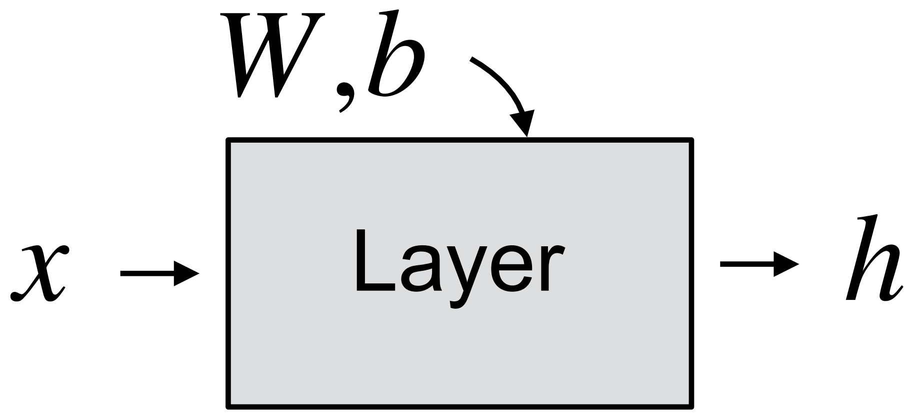
$$h = h(x; W)$$

$$\frac{\partial L}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial h_k}{\partial W_{ij}} \qquad \frac{\partial L}{\partial b_i} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial h_k}{\partial b_i}$$

What about the weights?

To get the derivative of the weights, use the chain rule again!

Example: 2D weights, 1D bias, 1D hidden activations:



$$h = h(x; W)$$

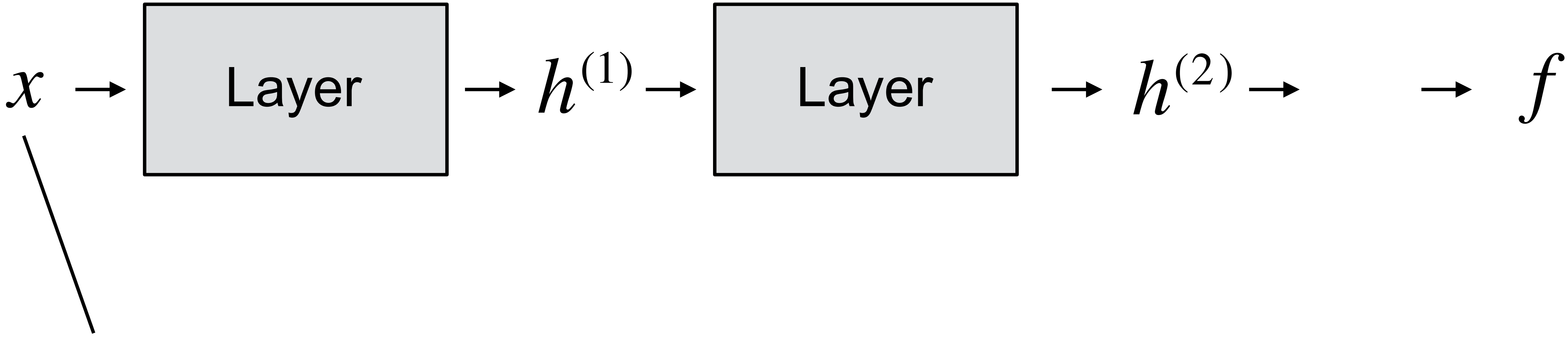
$$\frac{\partial L}{\partial W_{ij}} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial h_k}{\partial W_{ij}} \qquad \frac{\partial L}{\partial b_i} = \sum_k \frac{\partial L}{\partial h_k} \frac{\partial h_k}{\partial b_i}$$

(the number of subscripts and summations changes depending on your layer and parameter sizes)

ConvNets

They’re just neural networks with
3D activations and weight sharing

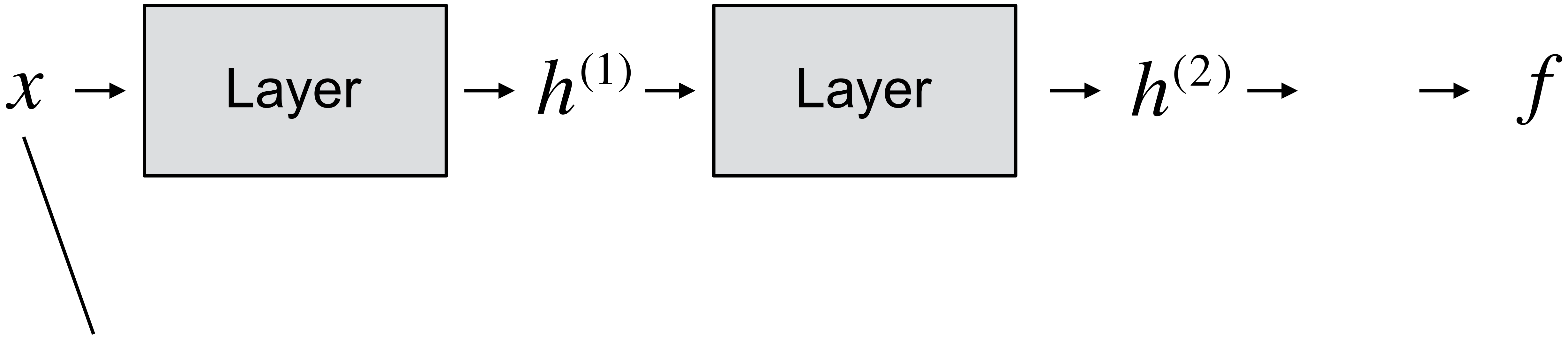
What shape should the activations have?



$x \rightarrow$ Layer $\rightarrow h^{(1)} \rightarrow$ Layer $\rightarrow h^{(2)} \rightarrow \quad \rightarrow f$

- The input is an image, which is 3D (RGB channel, height, width)

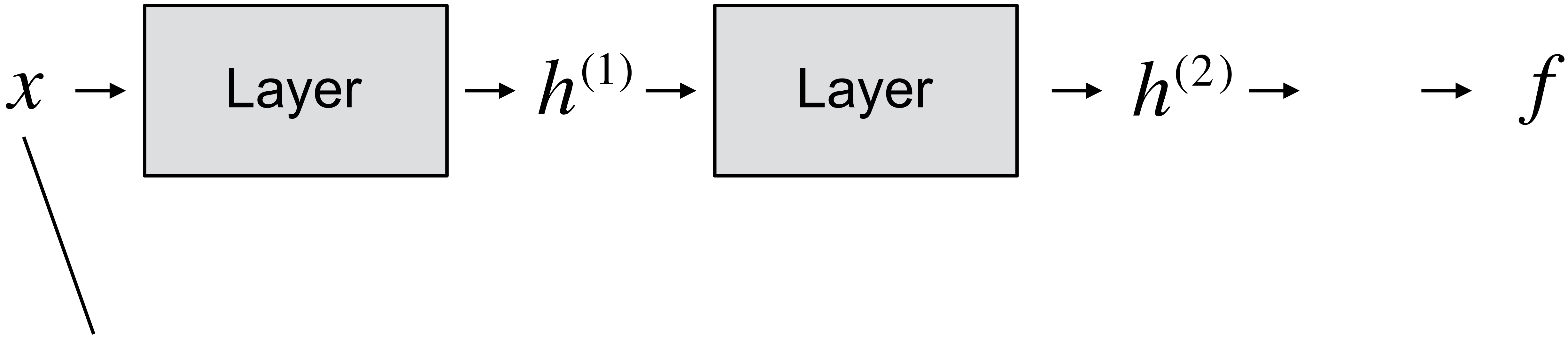
What shape should the activations have?



-The input is an image, which is 3D (RGB channel, height, width)

-We could flatten it to a 1D vector, but then we lose structure

What shape should the activations have?



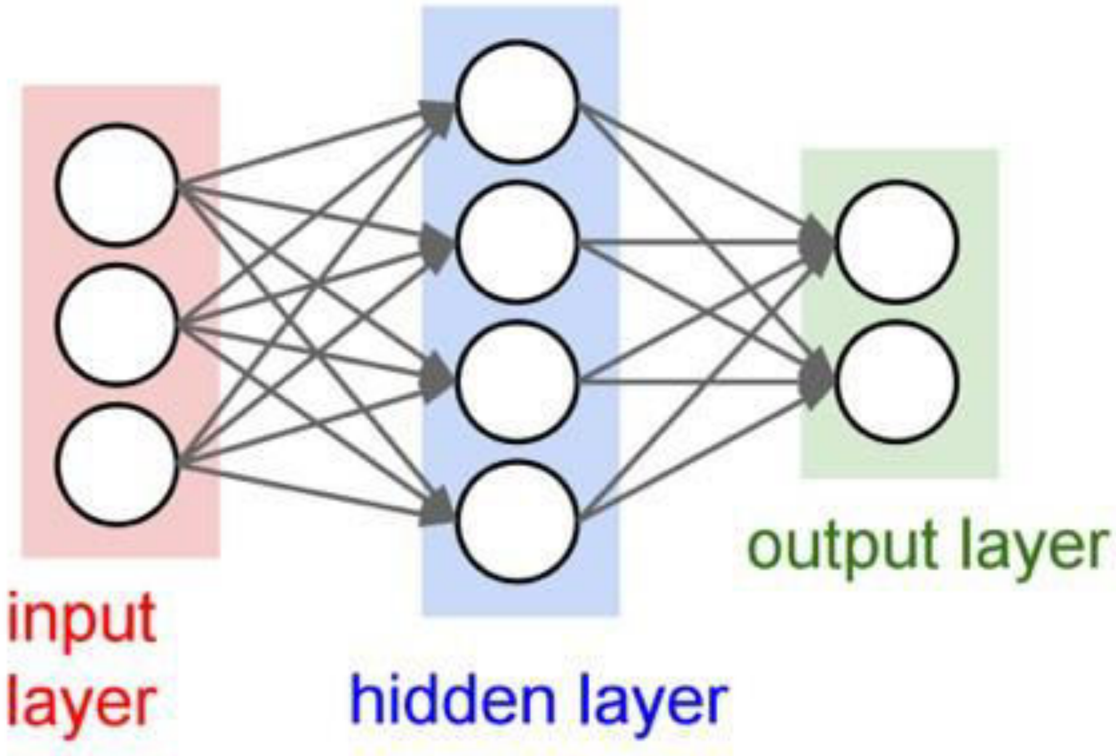
-The input is an image, which is 3D (RGB channel, height, width)

-We could flatten it to a 1D vector, but then we lose structure

- What about keeping everything in 3D?

3D Activations

before:

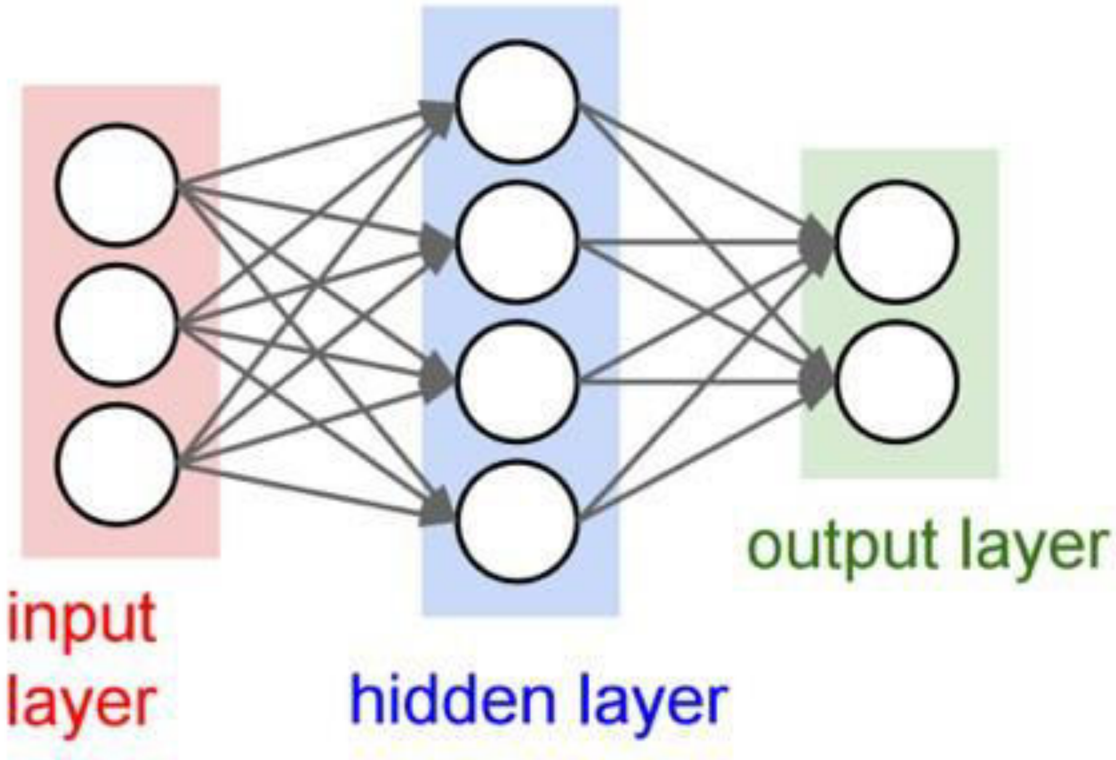


(1D vectors)

Figure: Andrej Karpathy

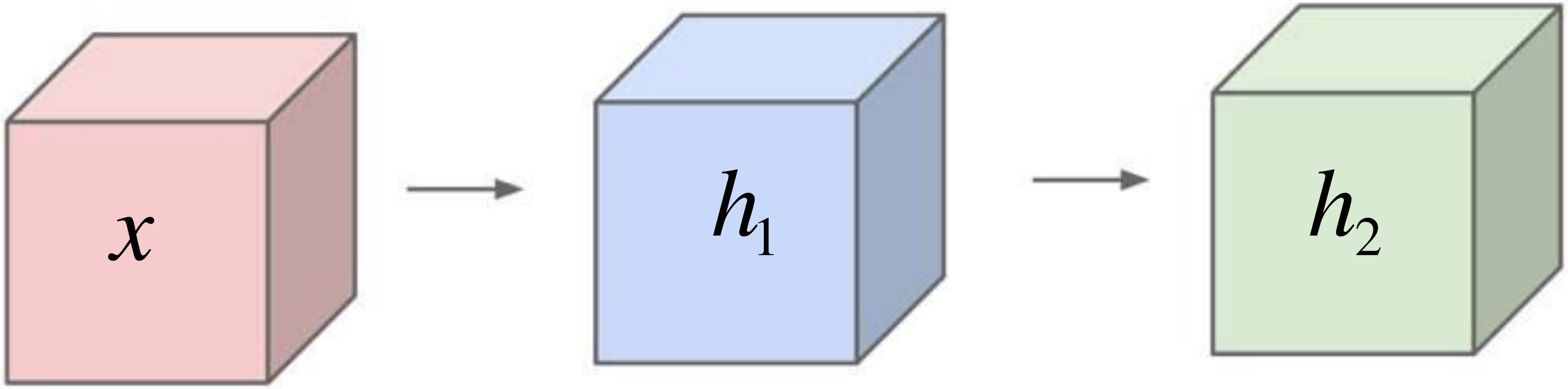
3D Activations

before:



(1D vectors)

now:



(3D arrays)

Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**

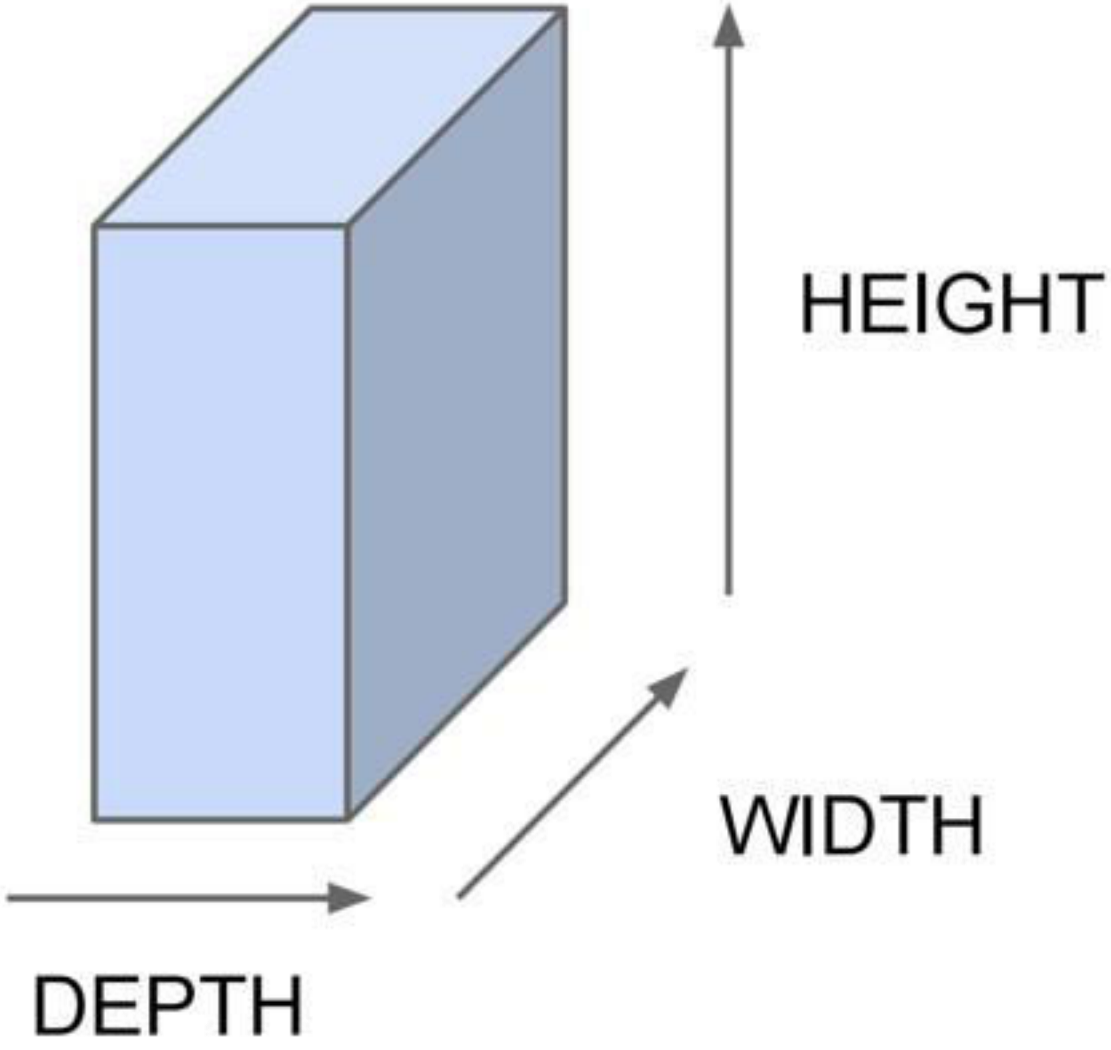
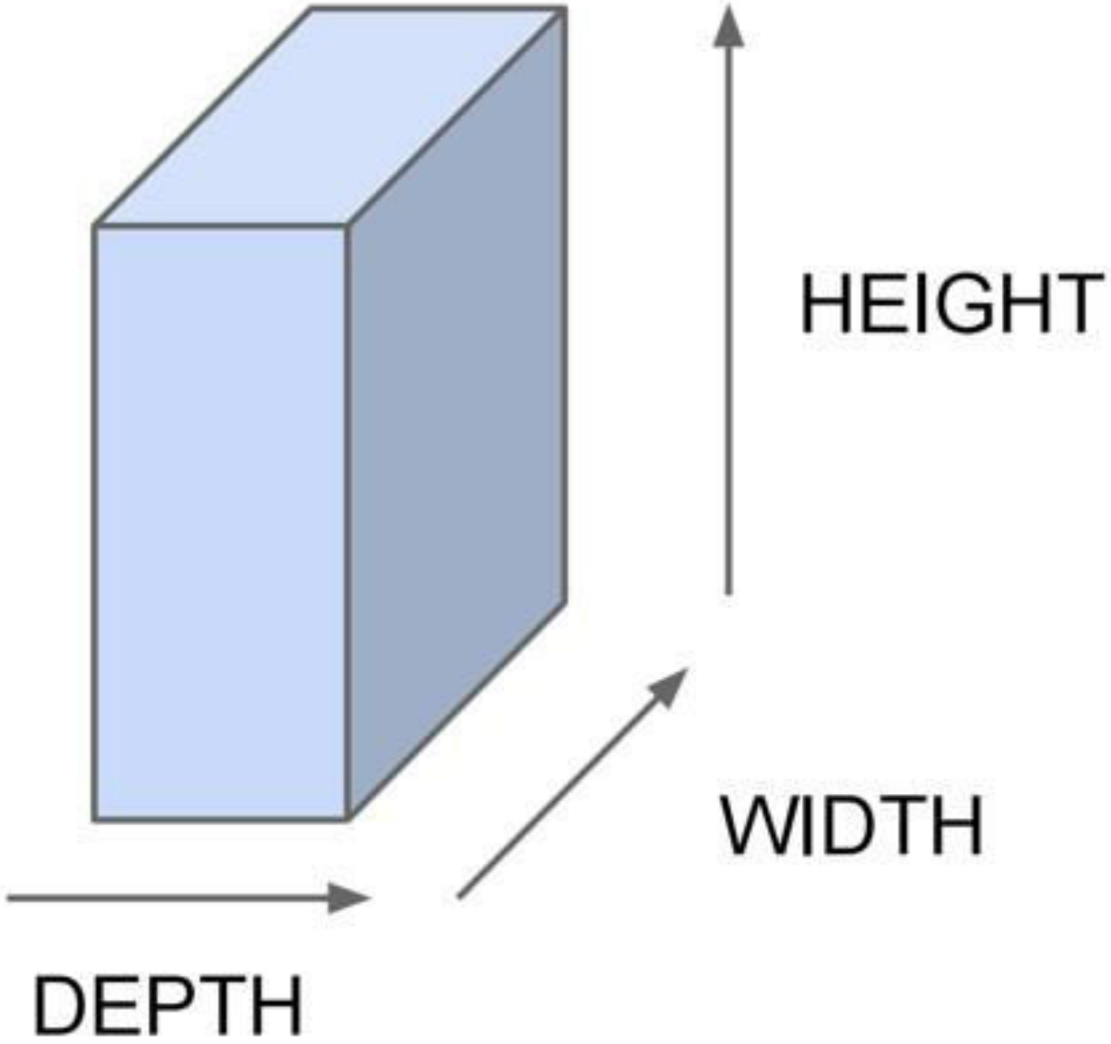


Figure: Andrej Karpathy

3D Activations

All Neural Net activations arranged in **3 dimensions:**



For example, a CIFAR-10 image is a 3x32x32 volume
(3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:

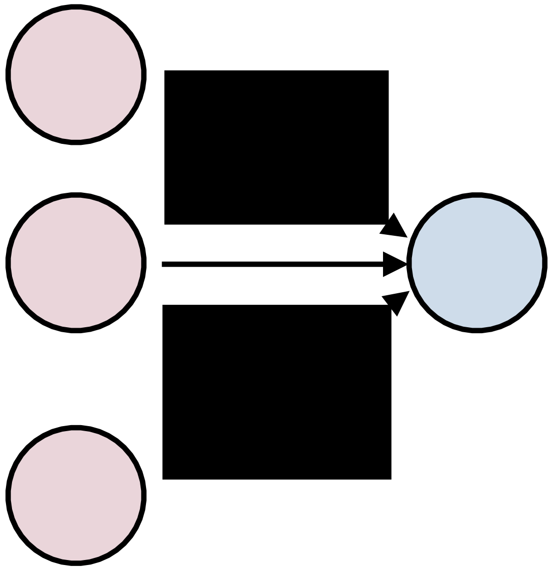
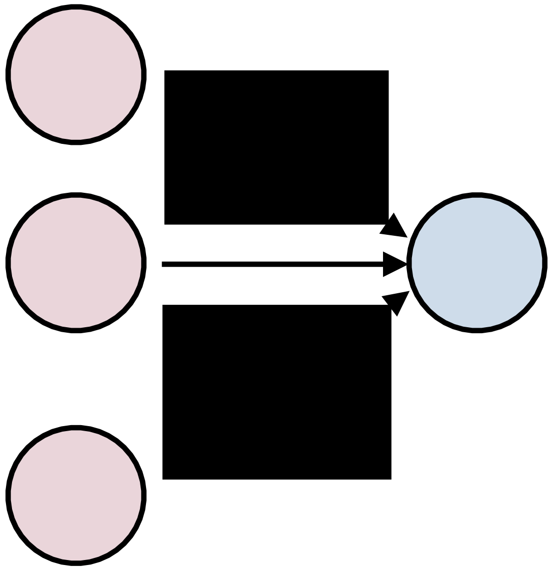


Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

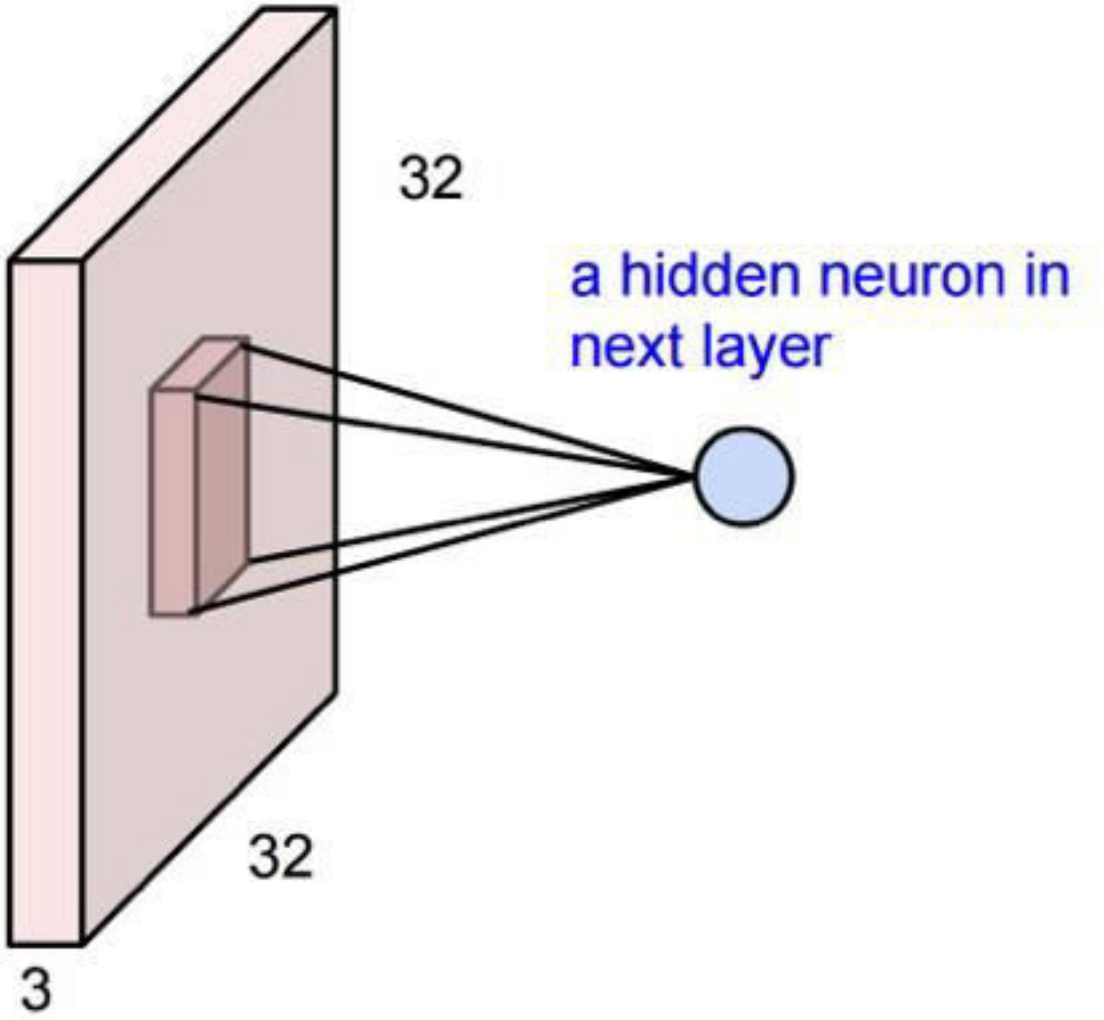
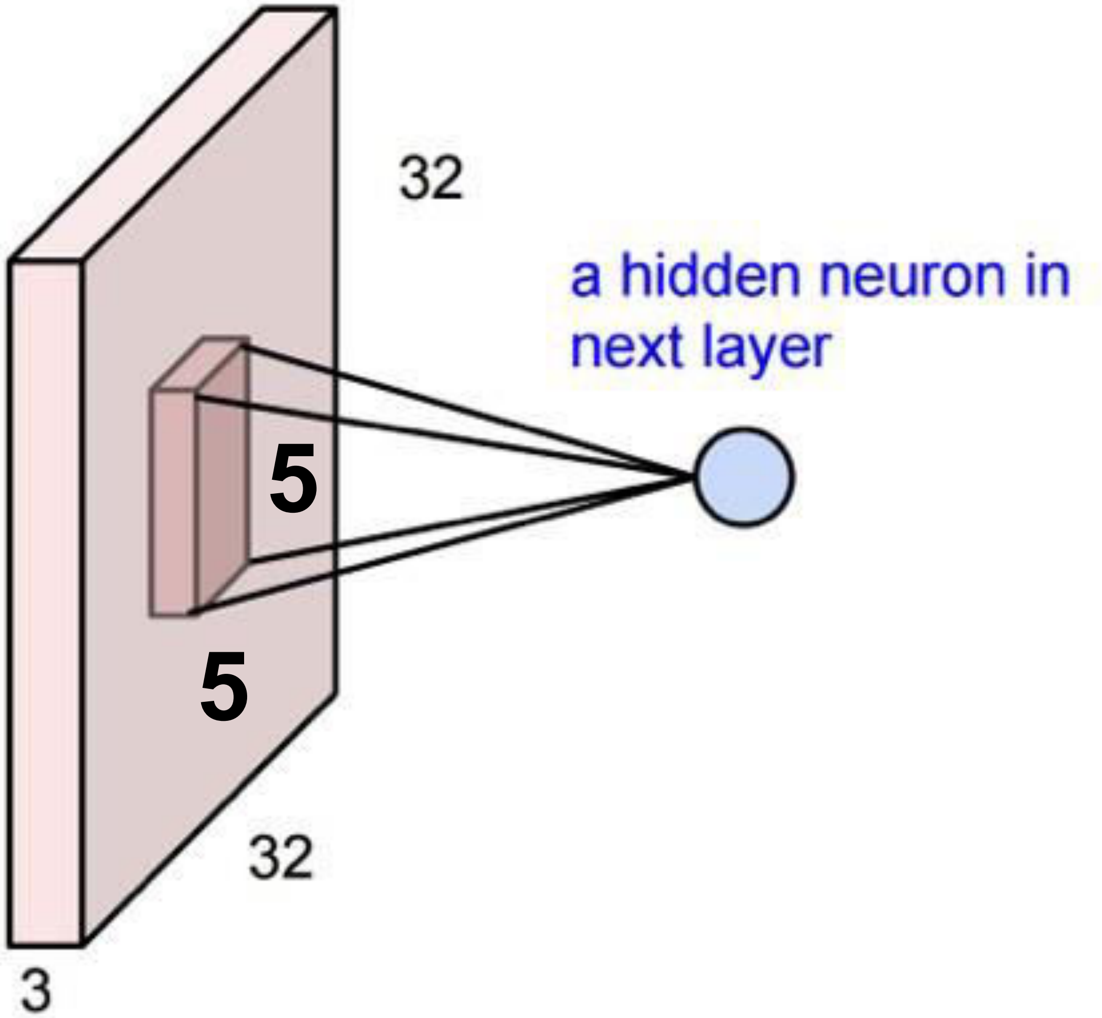


Figure: Andrej Karpathy

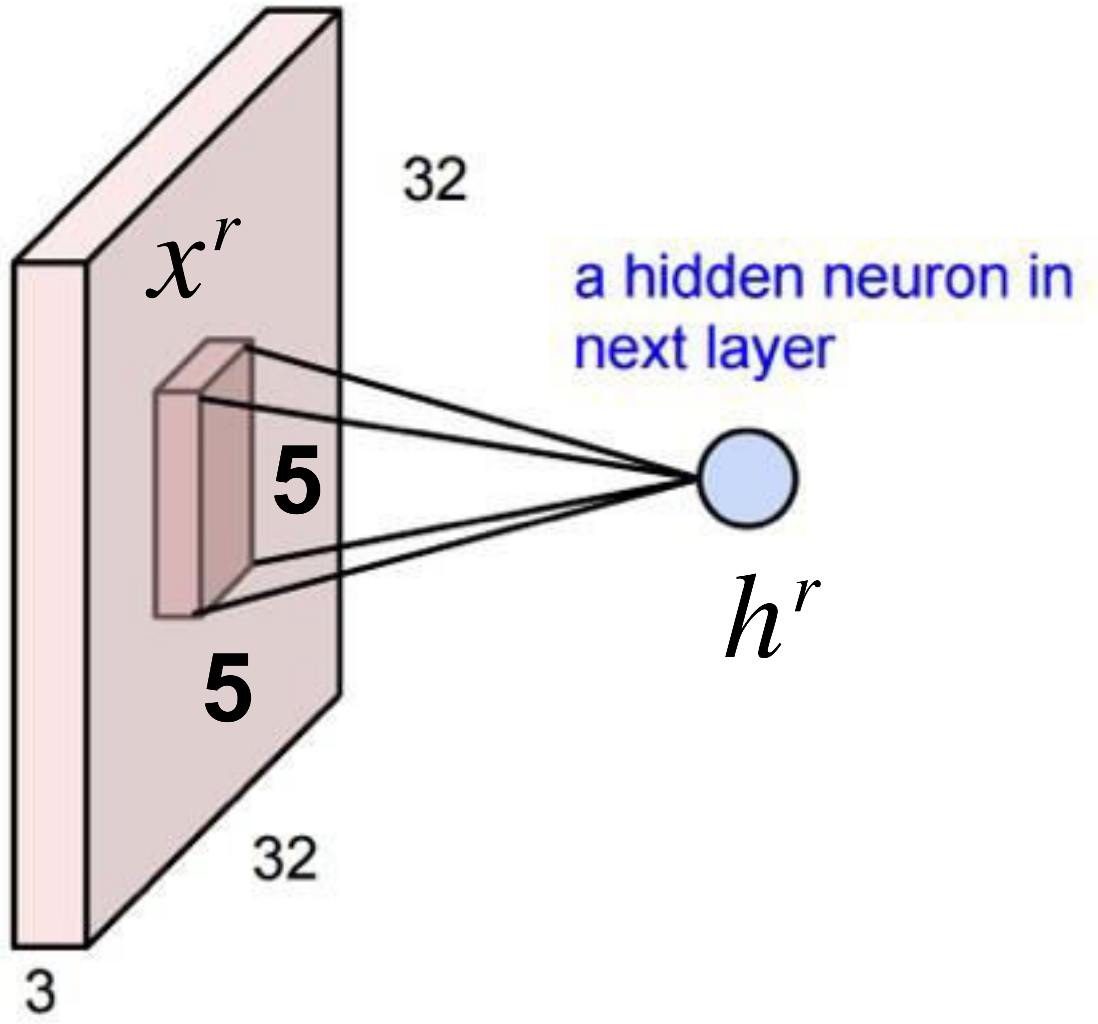
3D Activations



- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

Figure: Andrej Karpathy

3D Activations

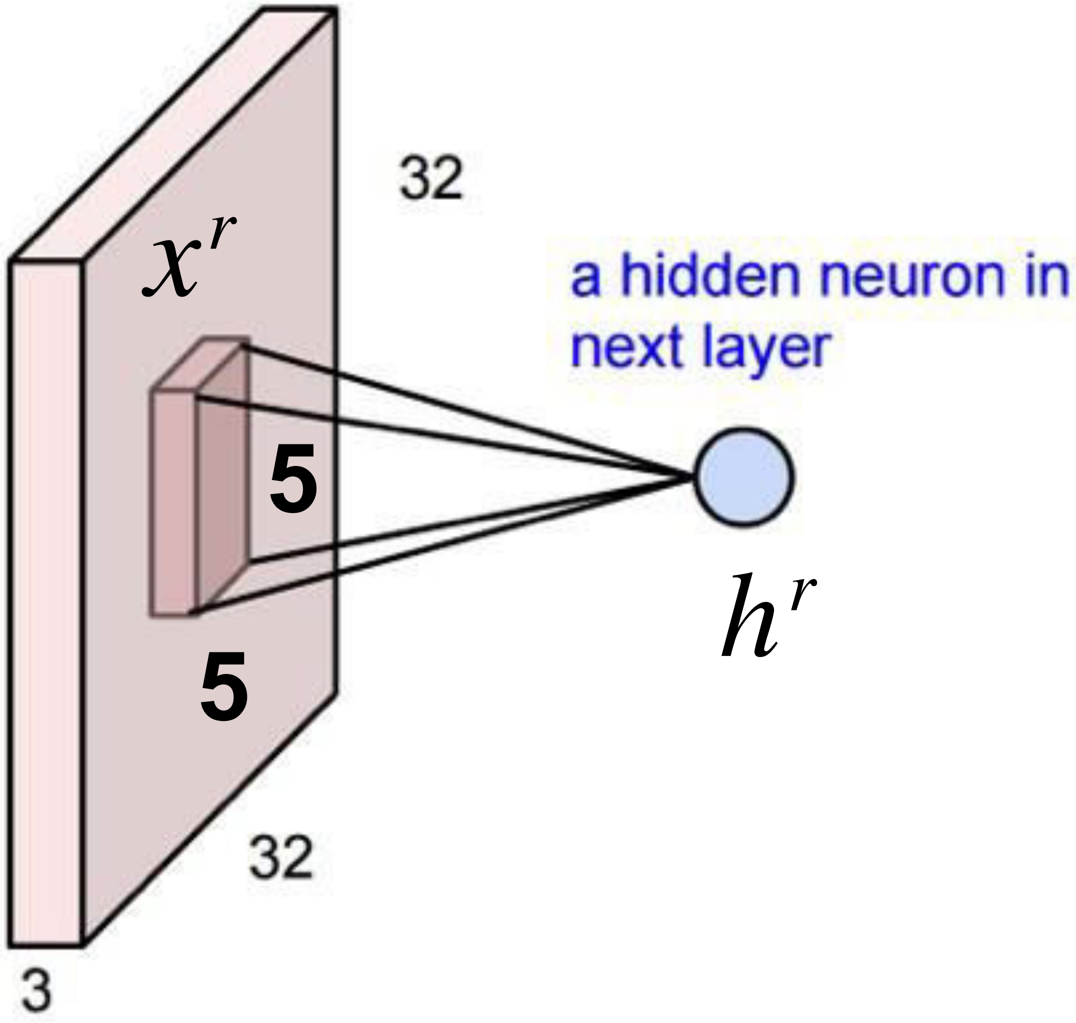


Example: consider the region of the input “x^r”

With output neuron h^r

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

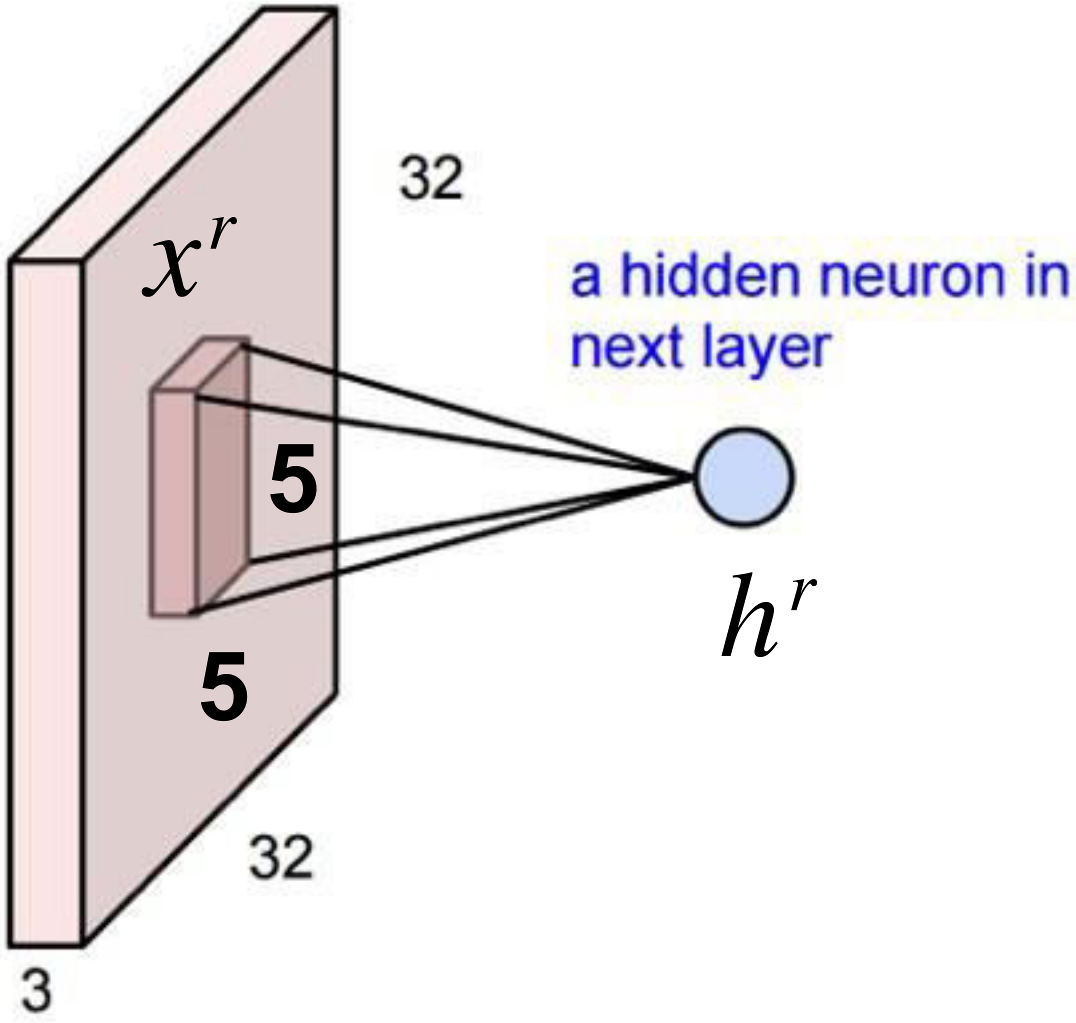
With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Figure: Andrej Karpathy

3D Activations



Example: consider the region of the input "x^r"

With output neuron h^r

Then the output is:

$$h^r = \sum_{ijk} x^r_{\ ijk} W_{ijk} + b$$

Sum over 3 axes

Figure: Andrej Karpathy

3D Activations

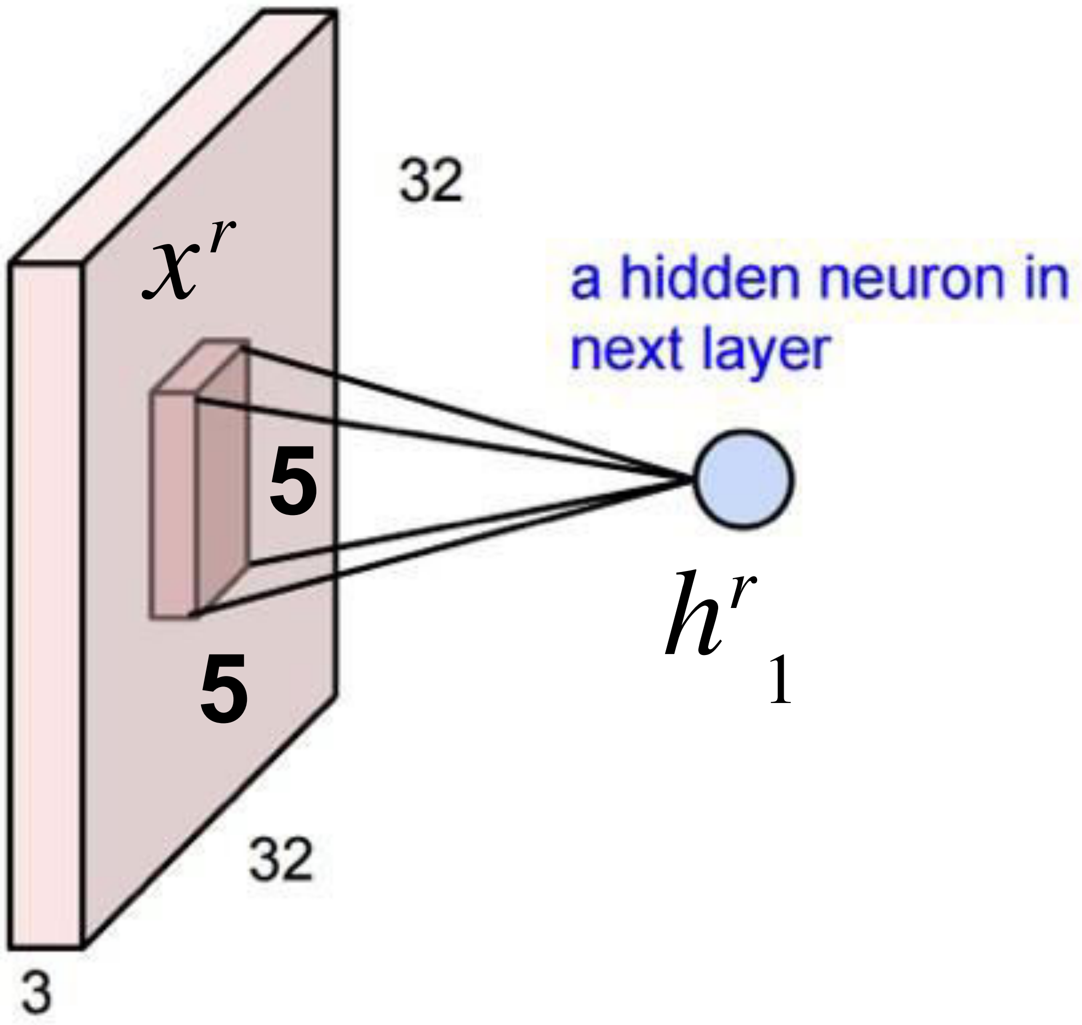


Figure: Andrej Karpathy

3D Activations

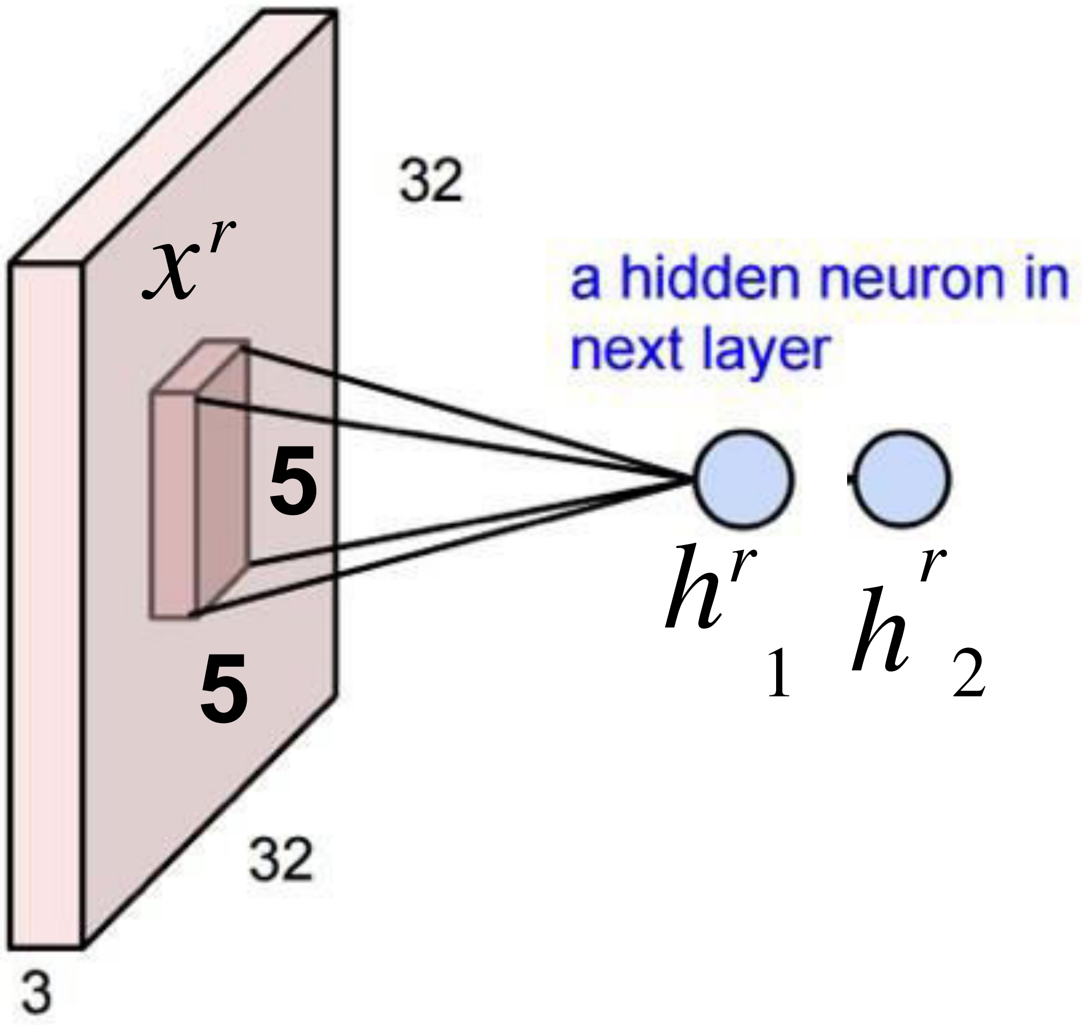
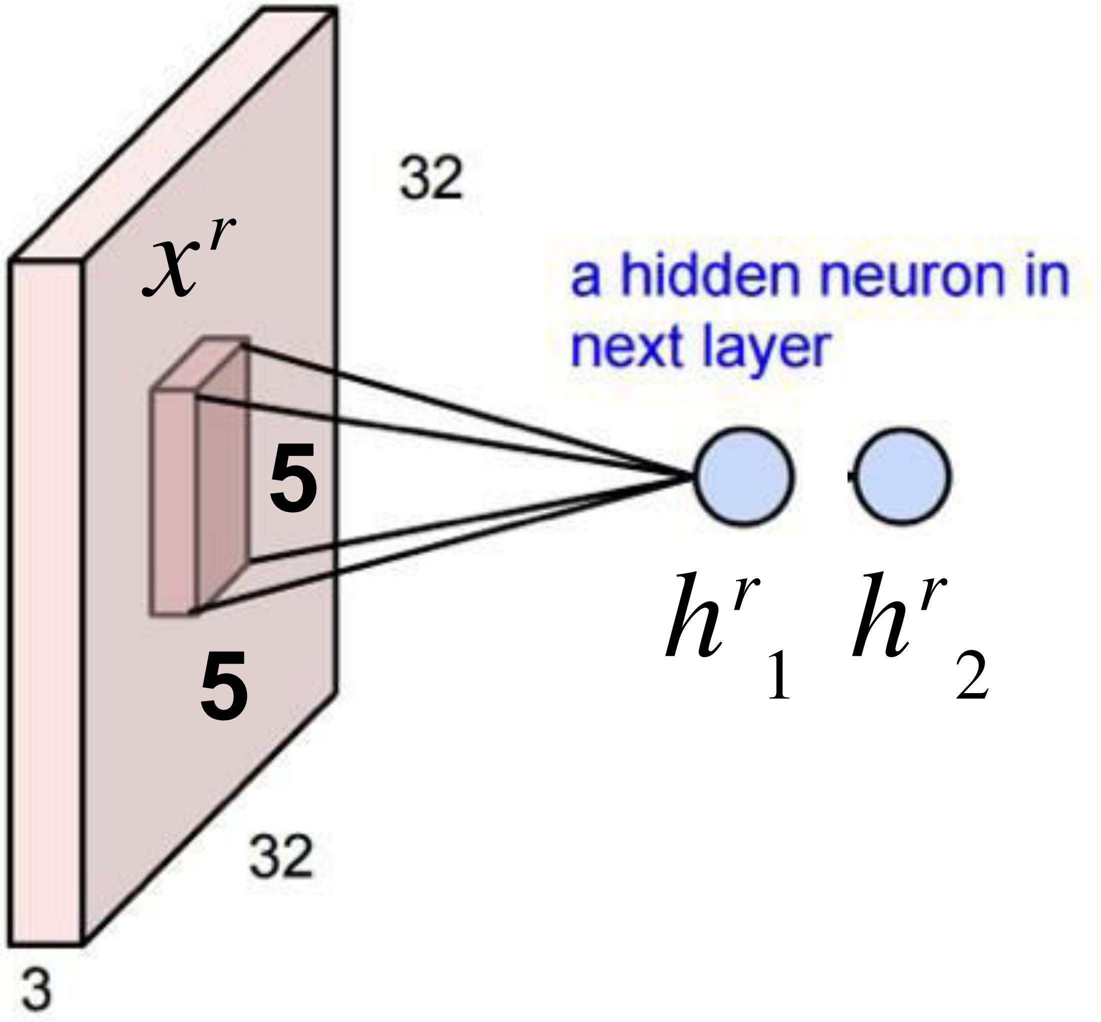


Figure: Andrej Karpathy

3D Activations



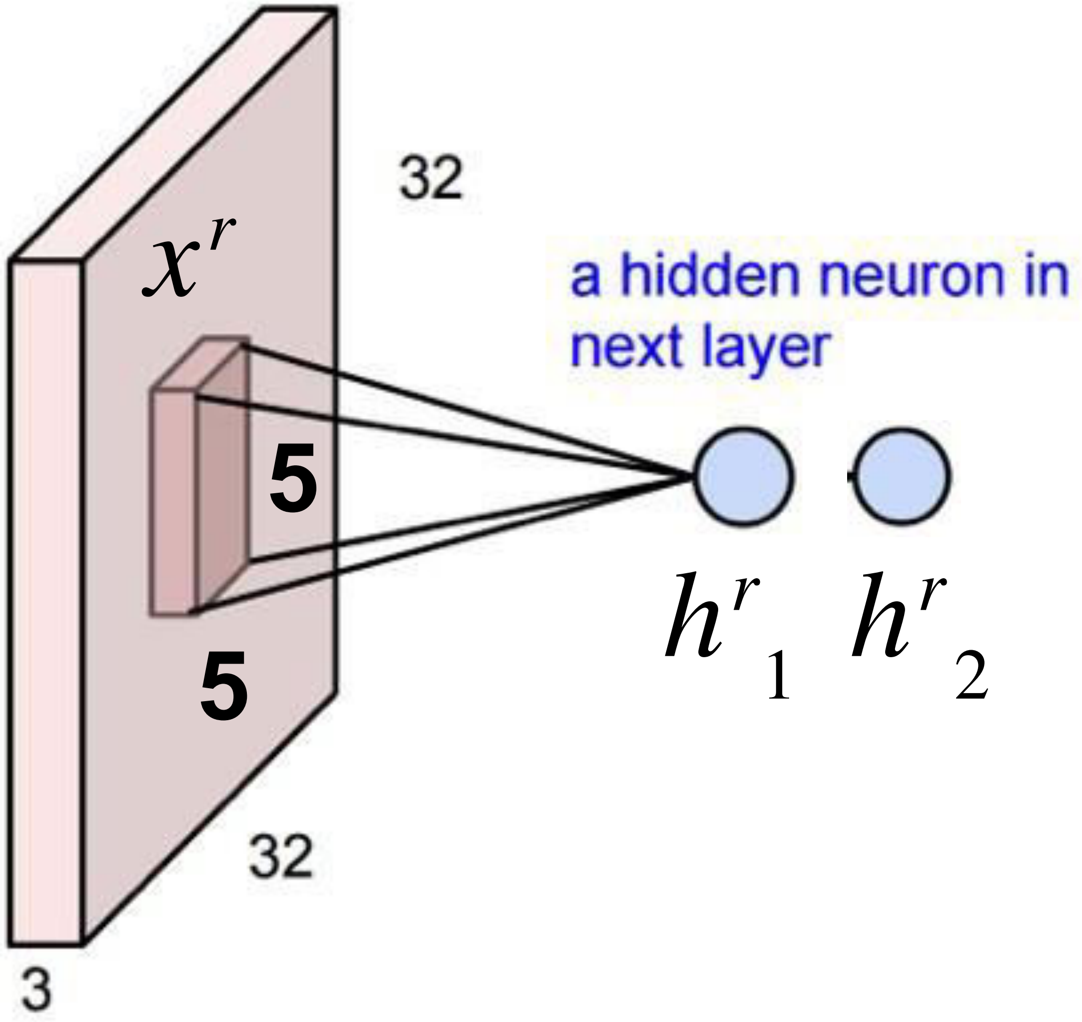
With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations



With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{1}ijk} + b_{\boxed{1}}$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{\boxed{2}ijk} + b_{\boxed{2}}$$

Figure: Andrej Karpathy

3D Activations

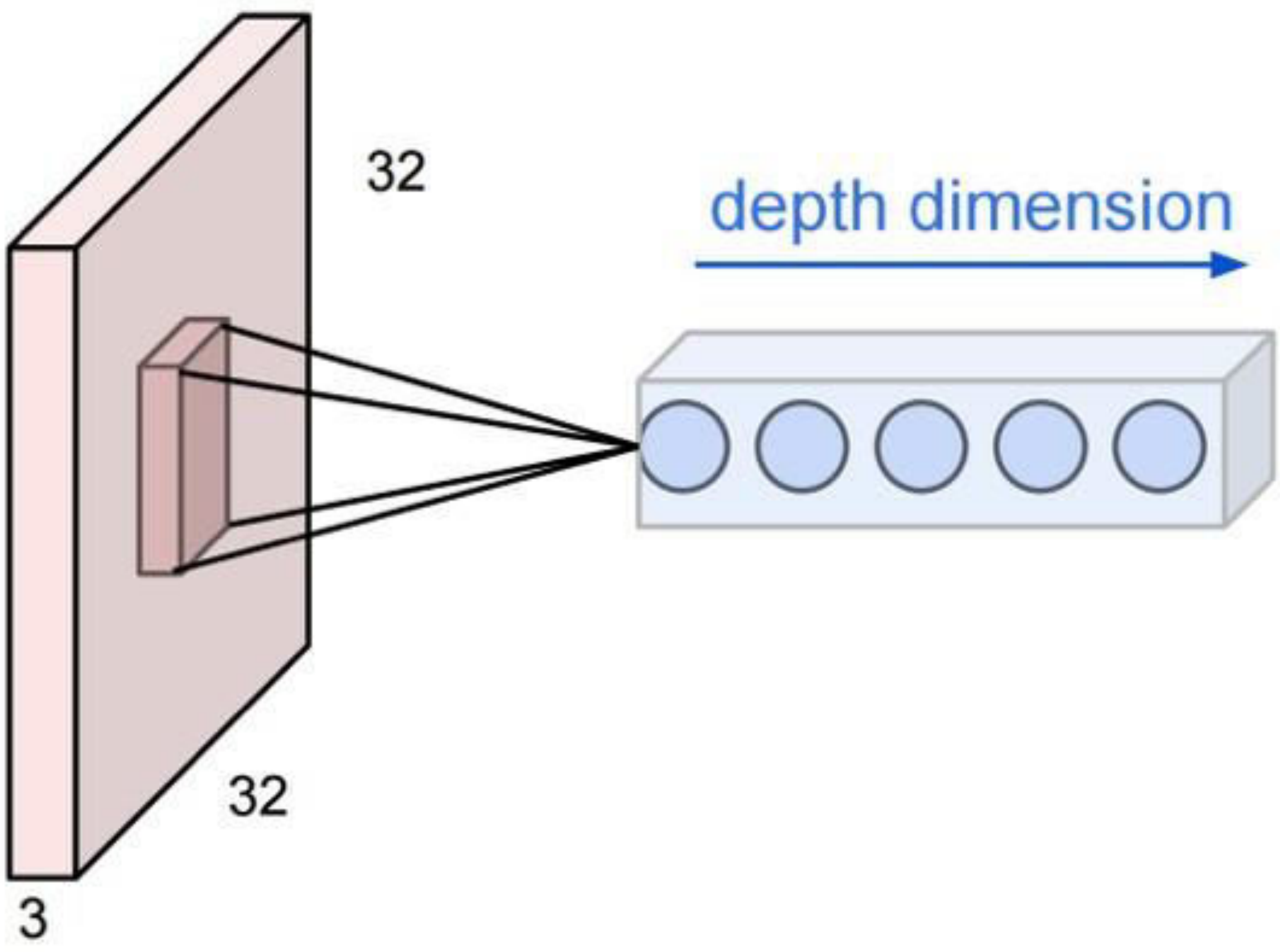
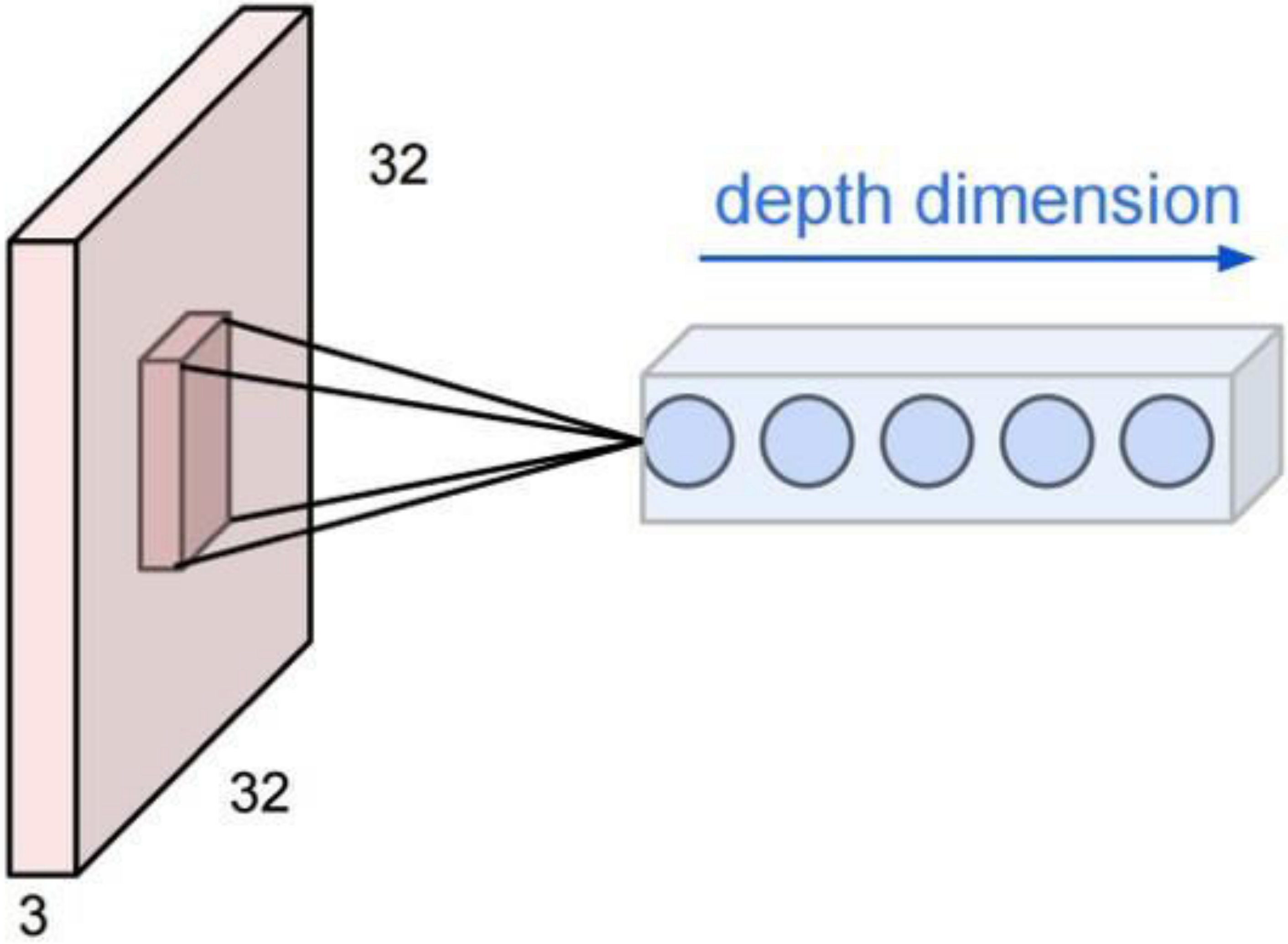


Figure: Andrej Karpathy

3D Activations

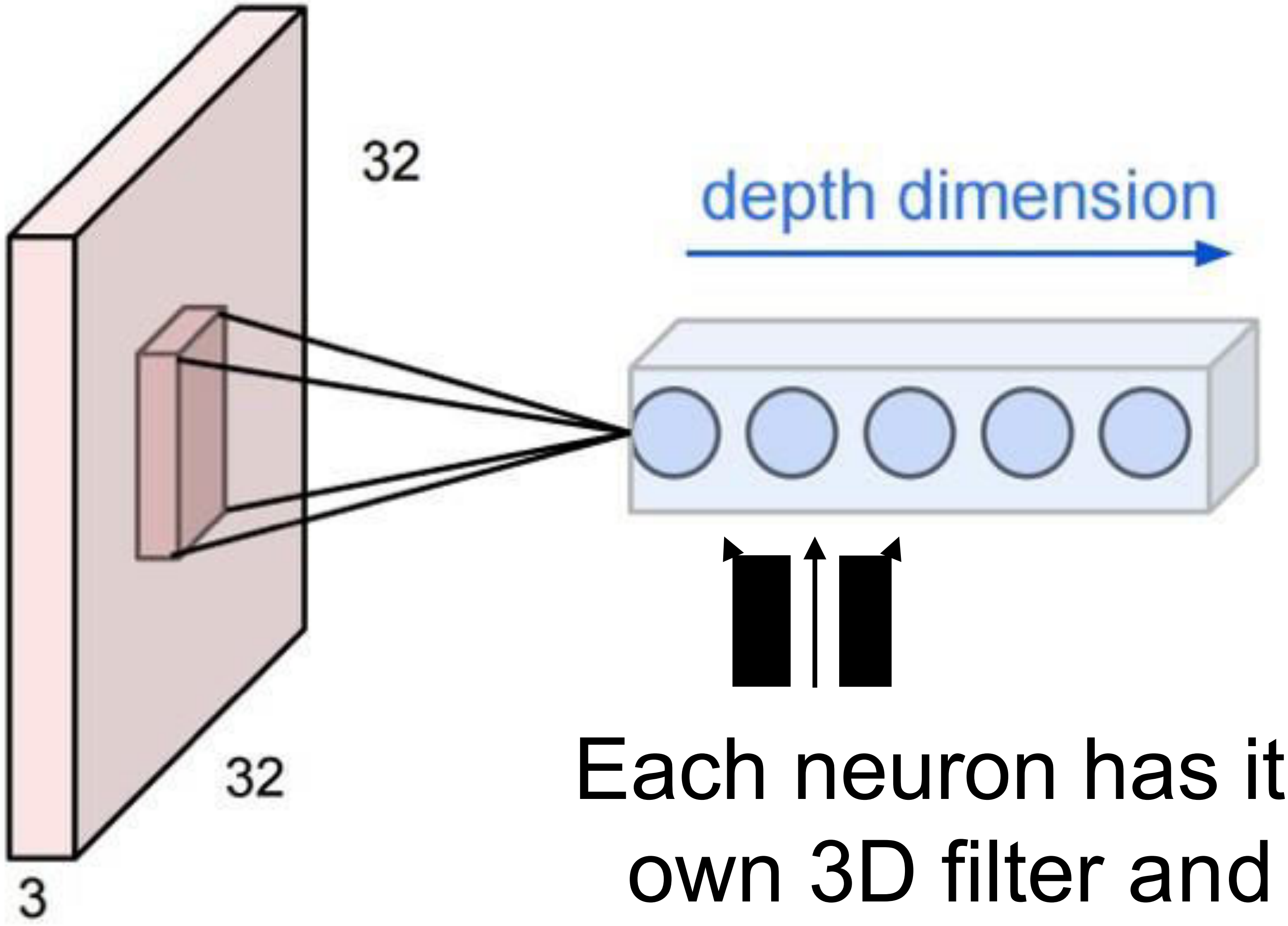


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations



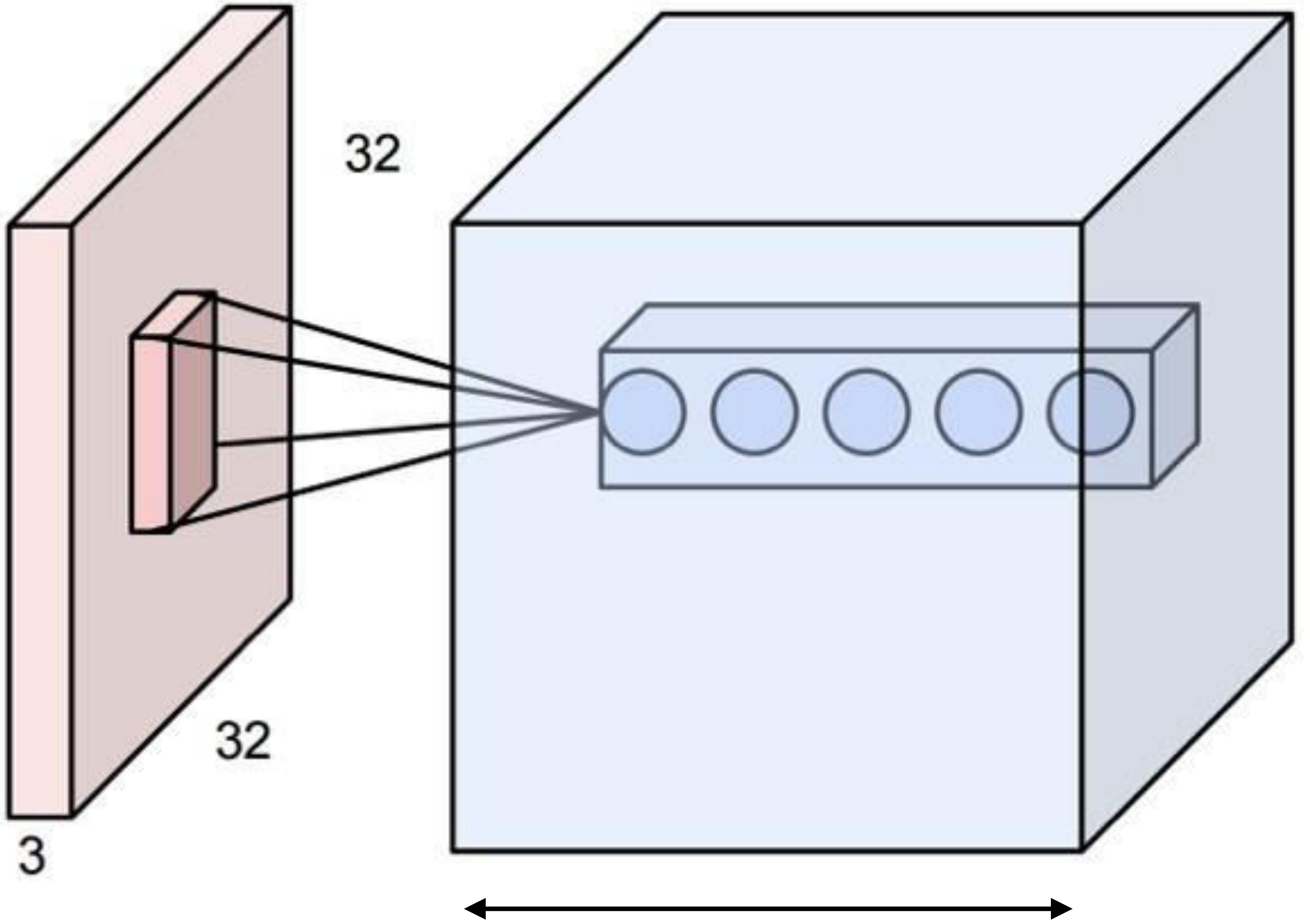
We can keep adding more outputs

These form a column in the output volume:
[depth x 1 x 1]

Each neuron has its own 3D filter and own (scalar) bias

Figure: Andrej Karpathy

3D Activations

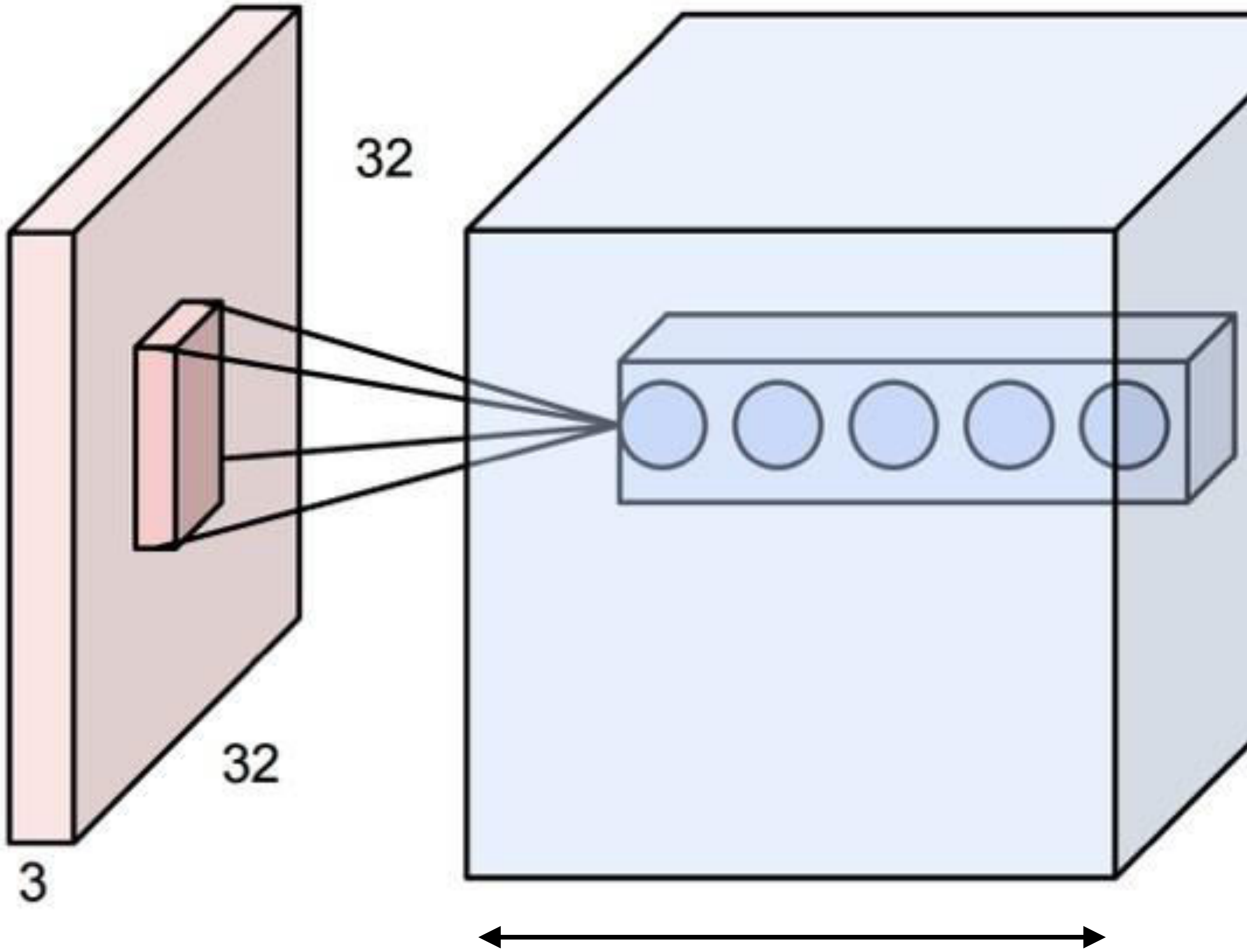


Now repeat this across the input

D sets of weights (also called filters)

Figure: Andrej Karpathy

3D Activations



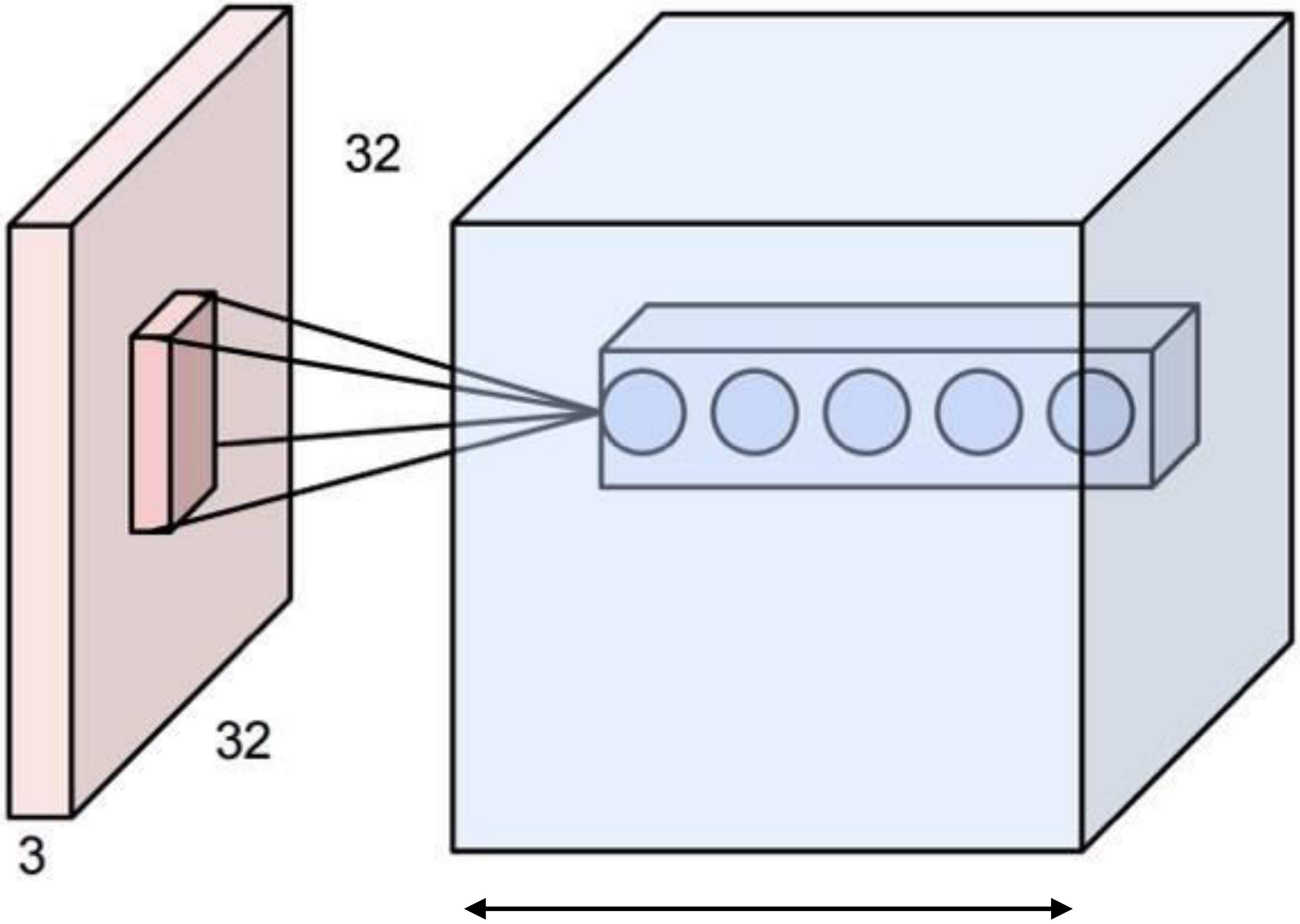
D sets of weights
(also called filters)

Now repeat this
across the input

Weight sharing:
Each filter shares
the same weights
(but each depth
index has its own
set of weights)

Figure: Andrej Karpathy

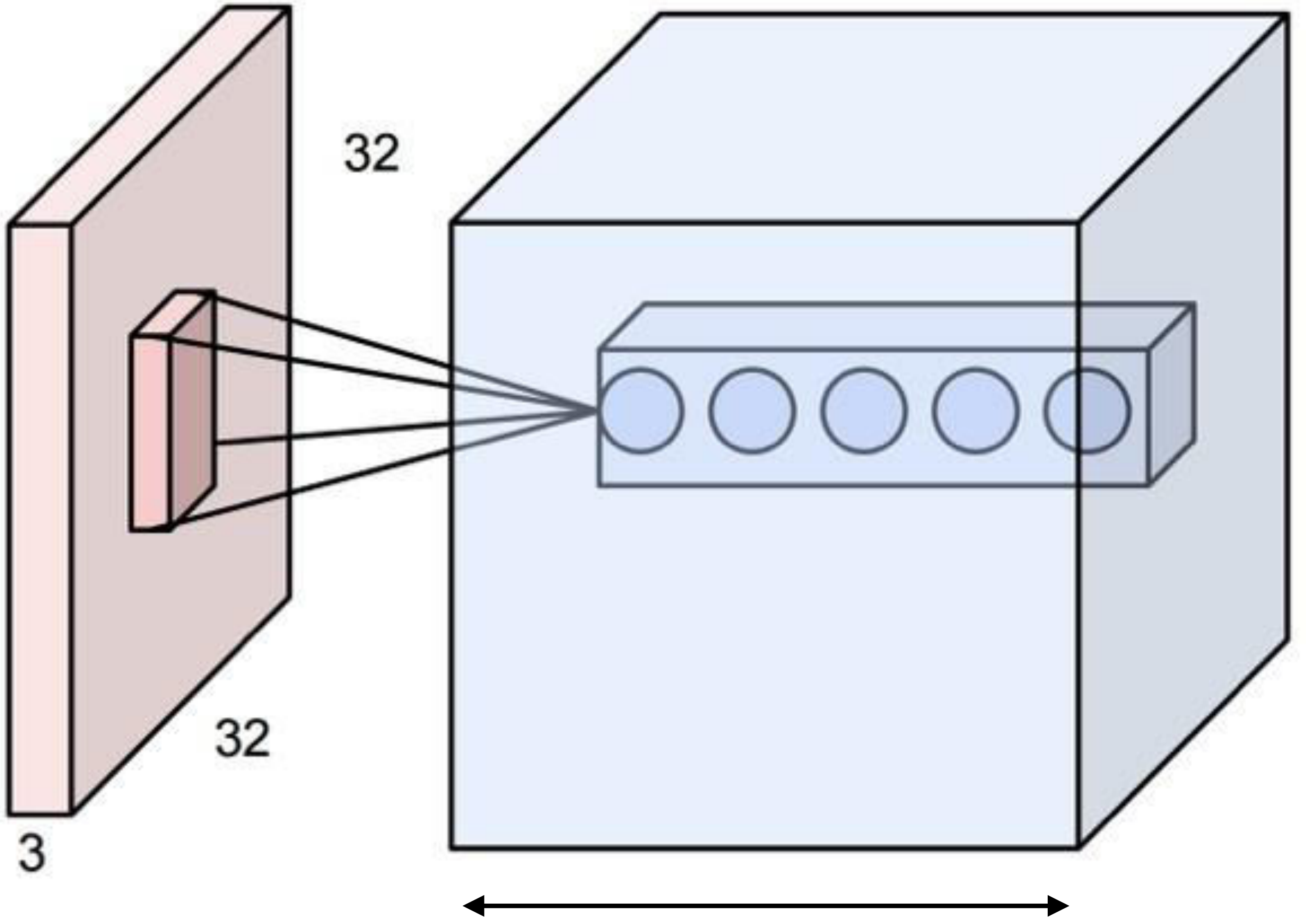
3D Activations



D sets of weights
(also called filters)

Figure: Andrej Karpathy

3D Activations

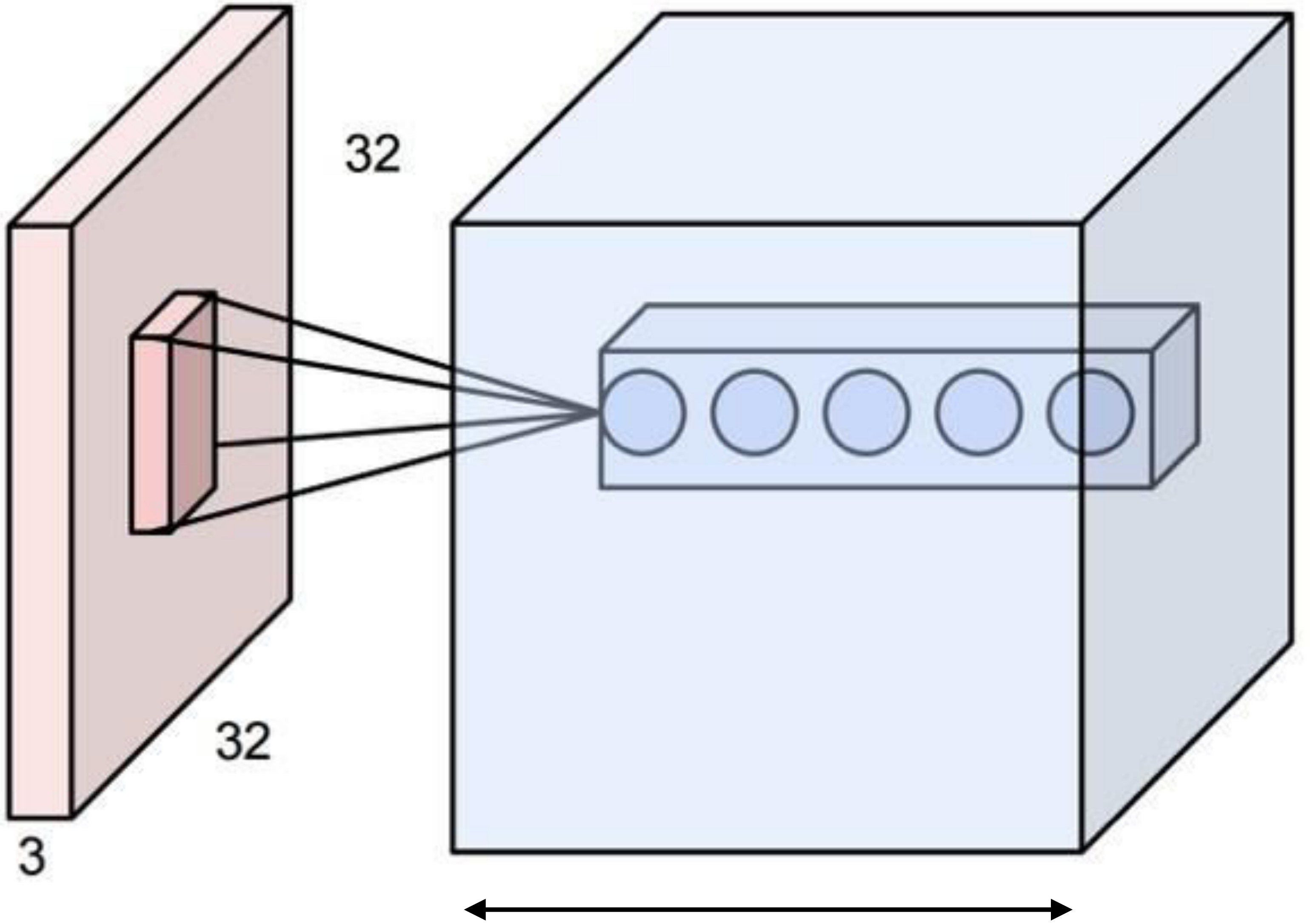


D sets of weights
(also called filters)

With weight
sharing,
this is called
convolution

Figure: Andrej Karpathy

3D Activations



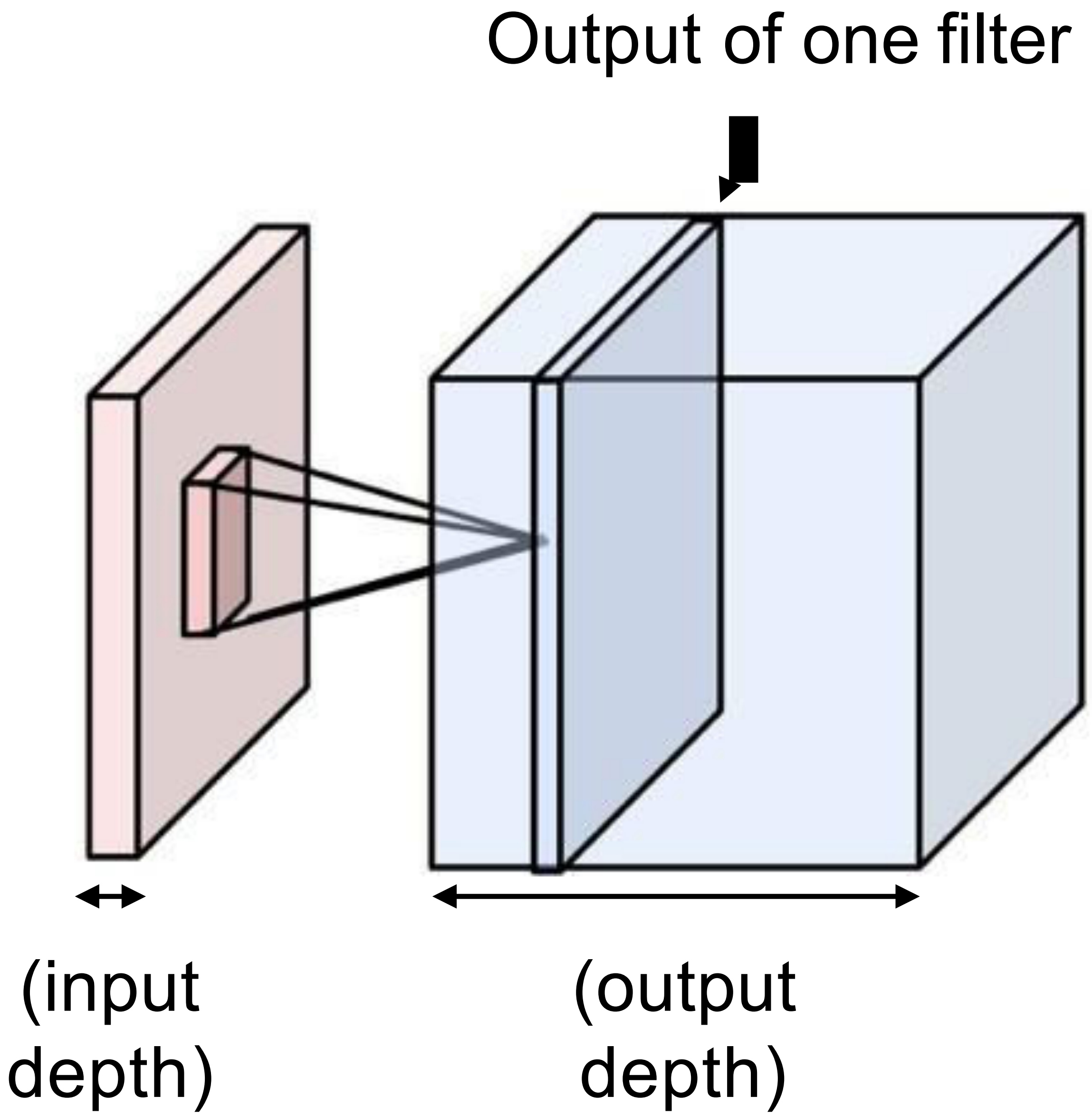
D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Without weight sharing, this is called a **locally connected layer**

Figure: Andrej Karpathy

3D Activations

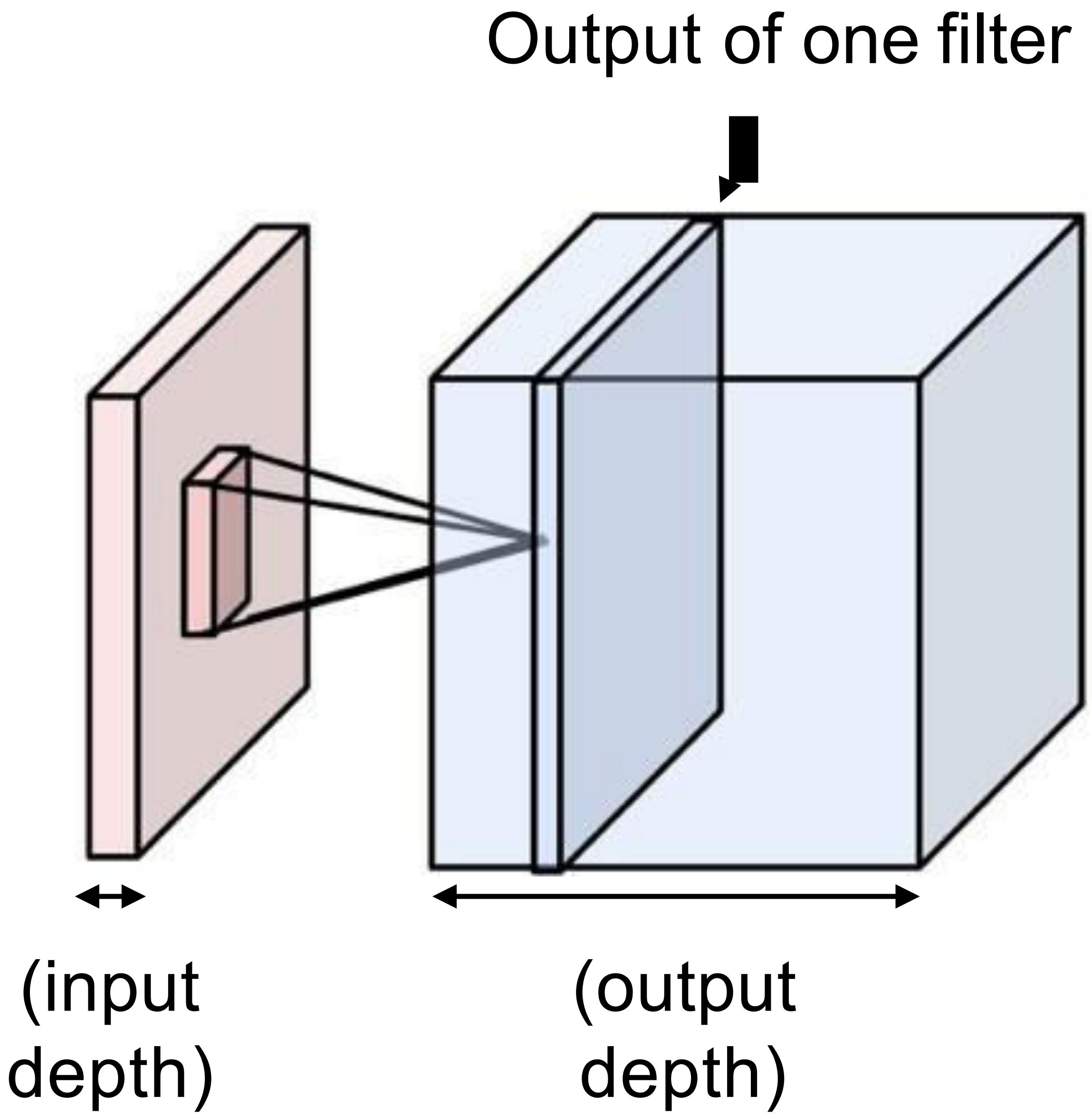


One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

3D Activations



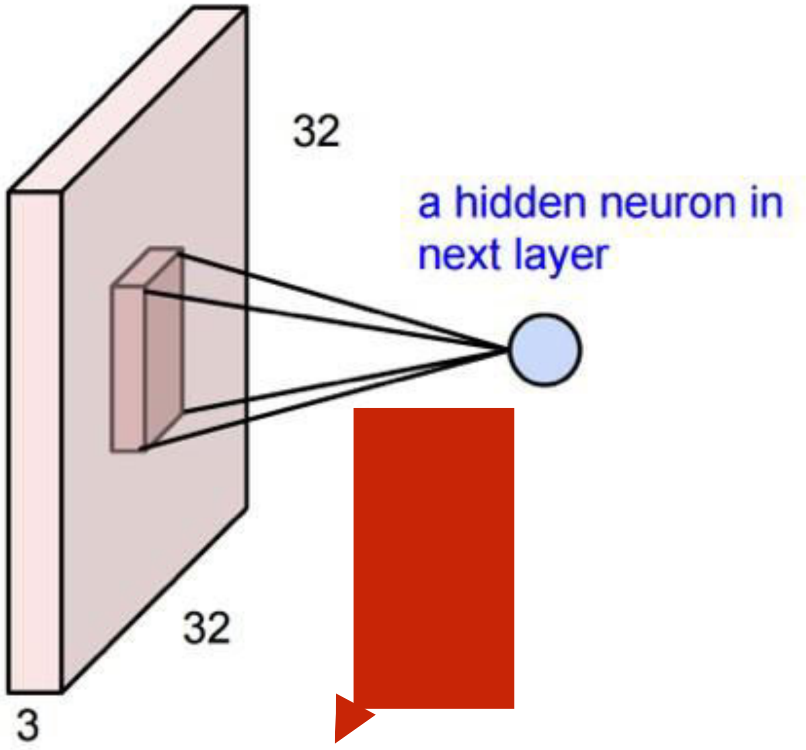
One set of weights gives one slice in the output

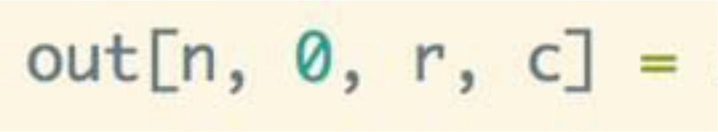
To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

3D Activations

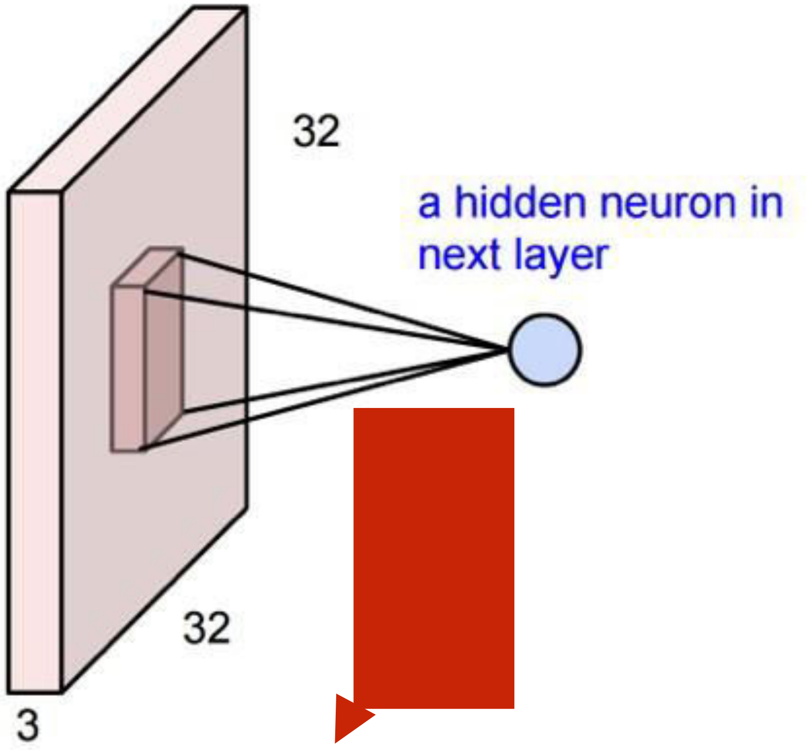




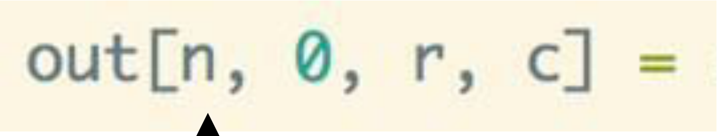
```
out[n, 0, r, c] =
```

Let’s code this up in NumPy

3D Activations



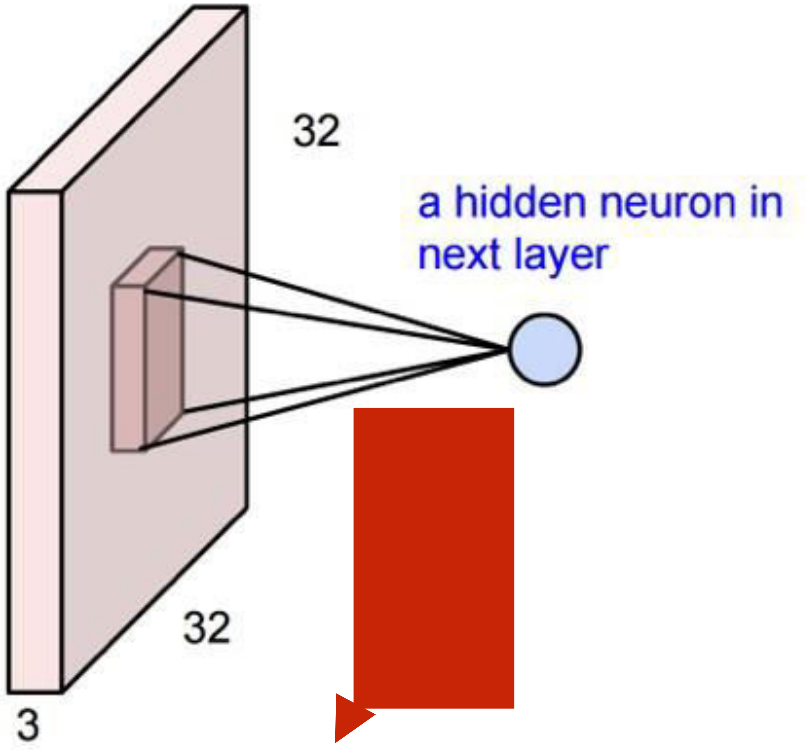
Let's code this up in NumPy



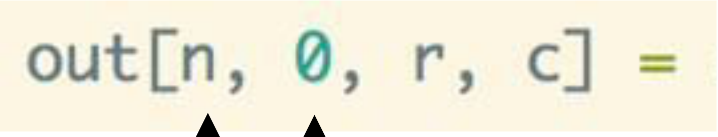
```
out[n, 0, r, c] =
```

nth example

3D Activations



Let’s code this up in NumPy

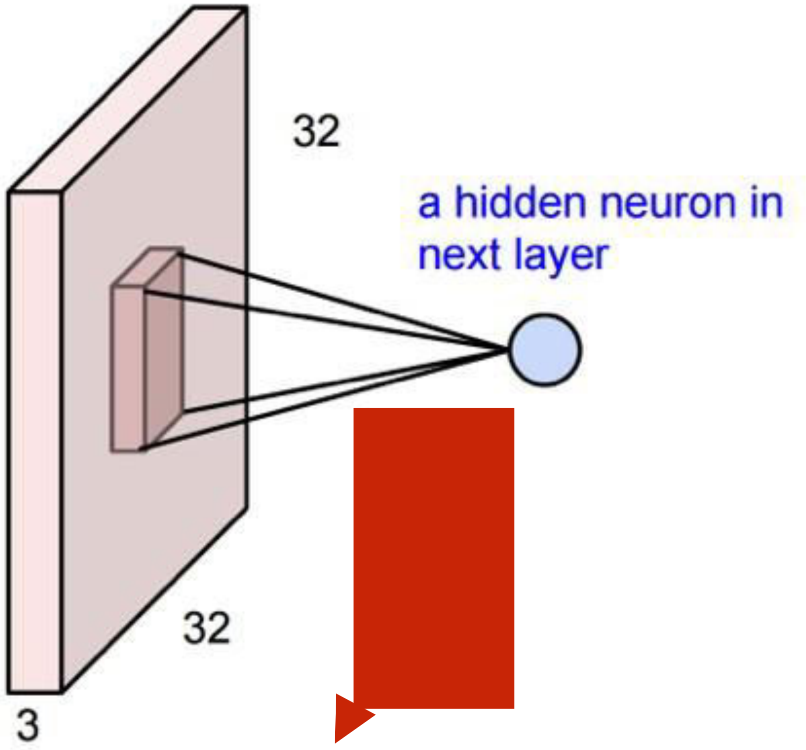


```
out[n, 0, r, c] =
```

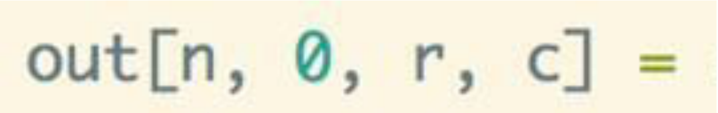
first filter

nth example

3D Activations



Let’s code this up in NumPy



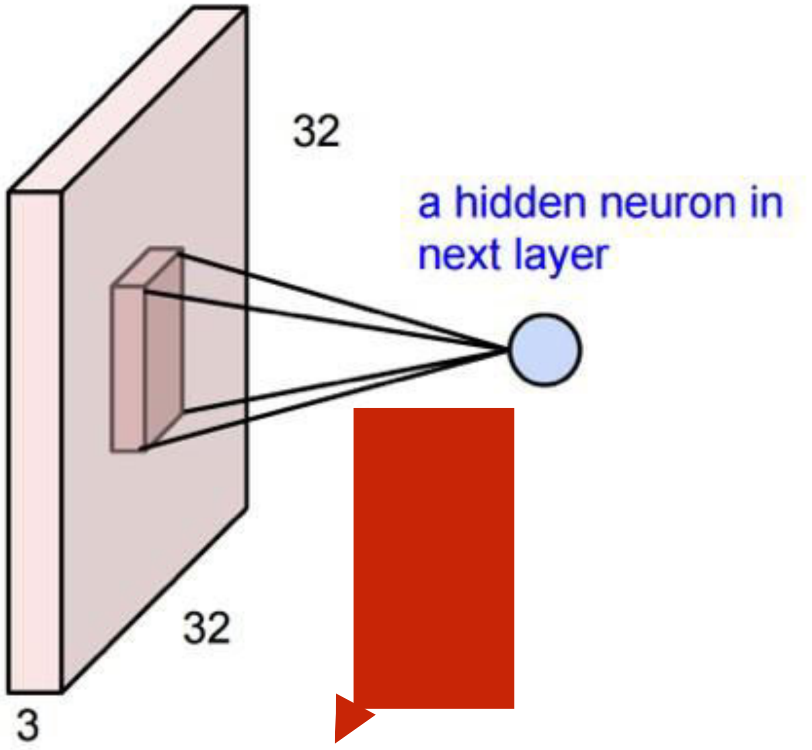
```
out[n, 0, r, c] =
```

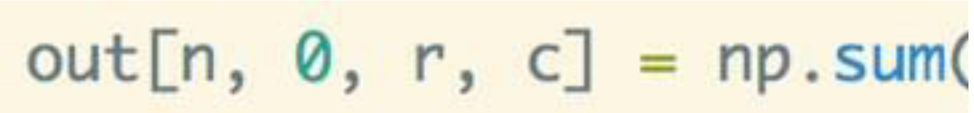
output position

first filter

nth example

3D Activations





```
out[n, 0, r, c] = np.sum(
```

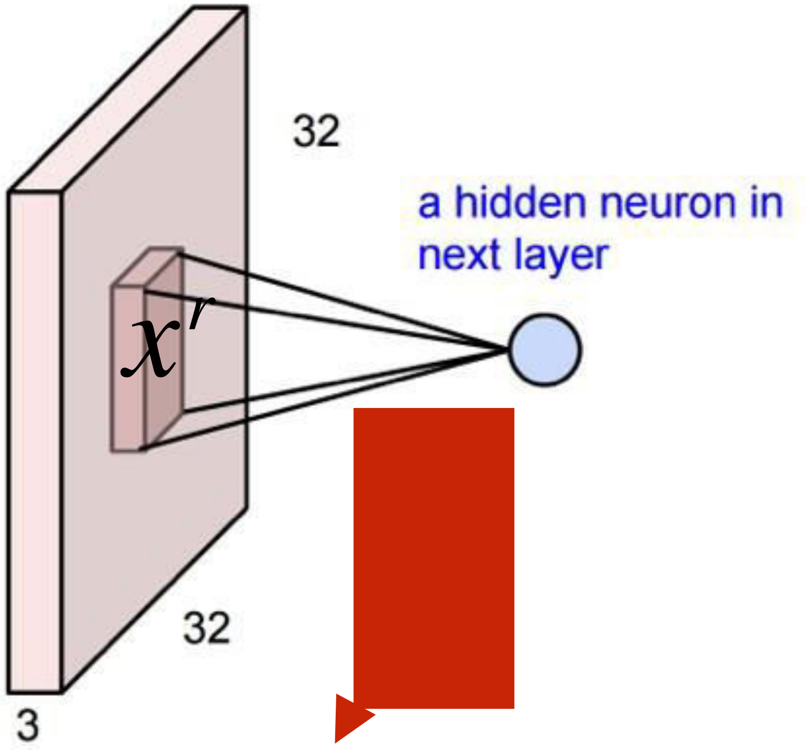
output position

first filter

nth example

Let’s code this up in NumPy

3D Activations



Let’s code this up in NumPy

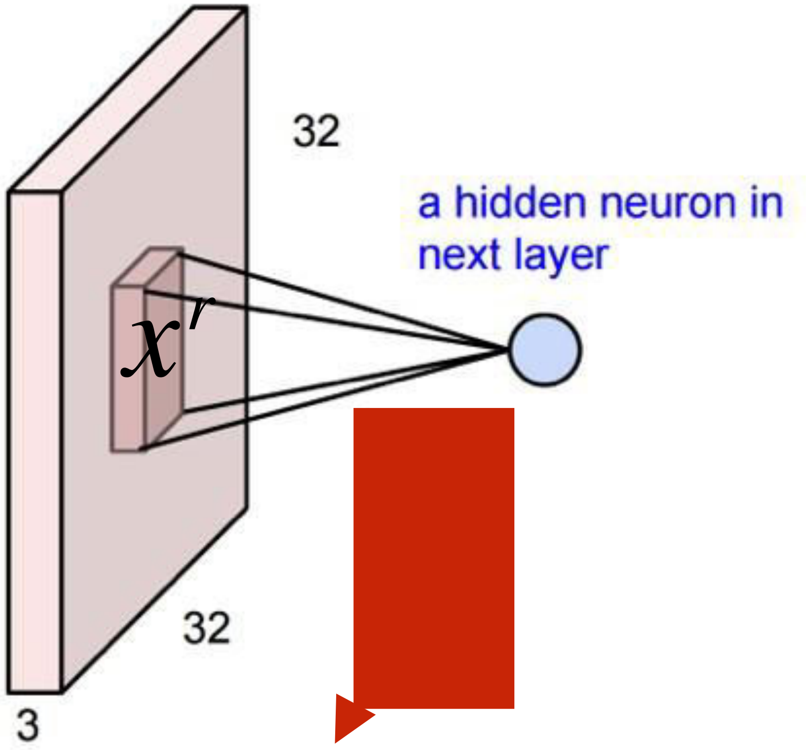
```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1]
```

output position

first filter

nth example

3D Activations



Let’s code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1]
```

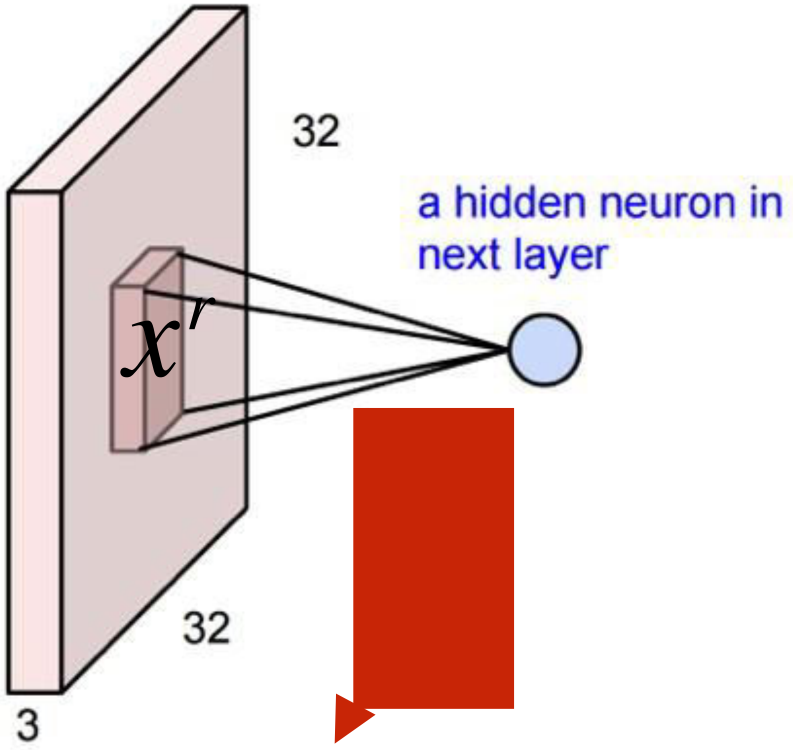
output position

first filter

n^{th} example

n^{th} example

3D Activations



Let's code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1]
```

output position

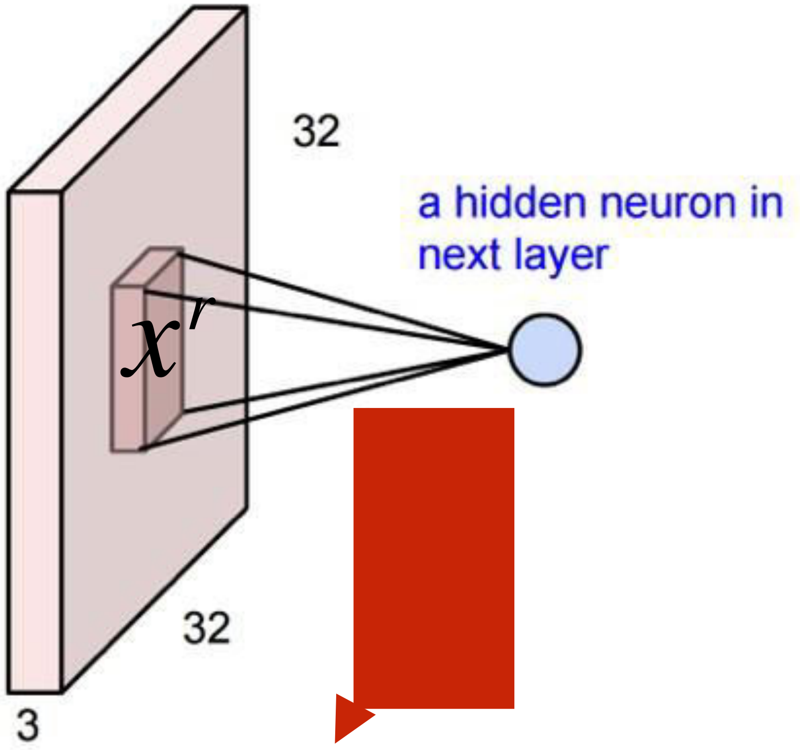
first filter

all input channels

nth example

nth example

3D Activations



Let's code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1]
```

output position

input region

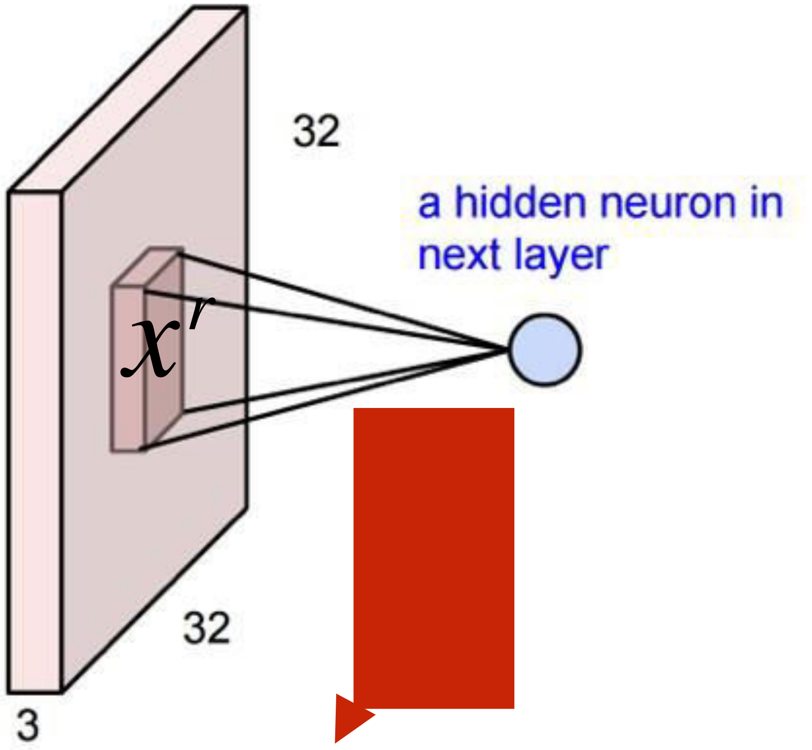
first filter

all input channels

nth example

nth example

3D Activations



Let's code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1] * W[0, :, :, :]) + b[0]
```

output position

first filter

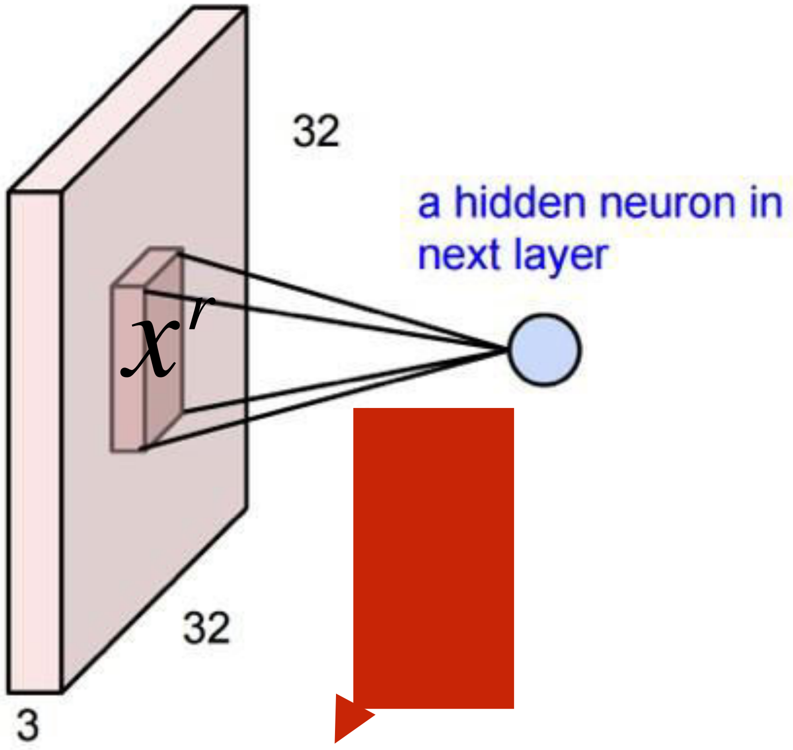
nth example

input region

all input channels

nth example

3D Activations



Let’s code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1] * W[0, :, :, :]) + b[0]
```

output position

first filter

nth example

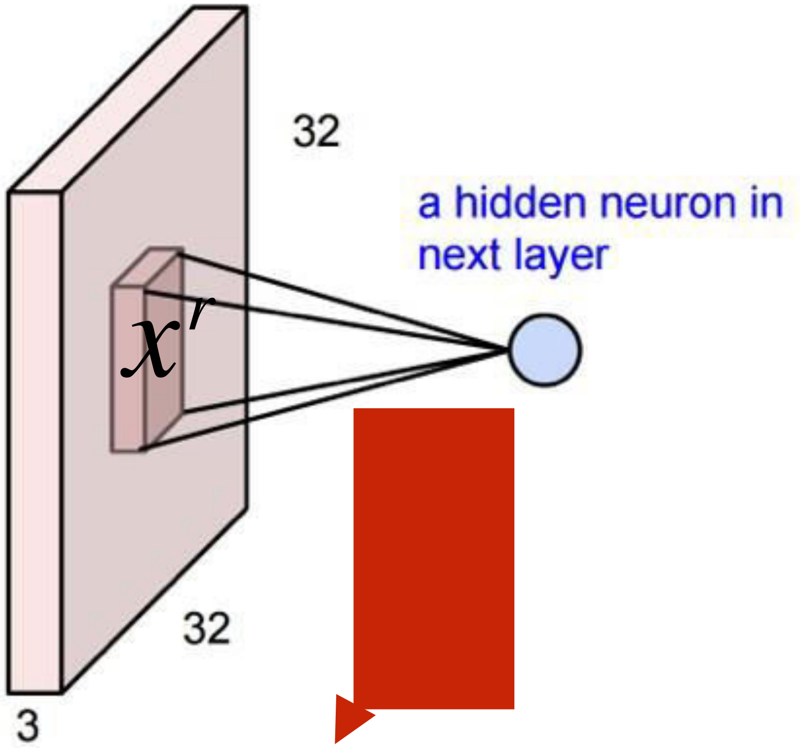
input region

all input channels

nth example

first filter

3D Activations



Let’s code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1] * W[0, :, :, :]) + b[0]
```

nth example

first filter

output position

nth example

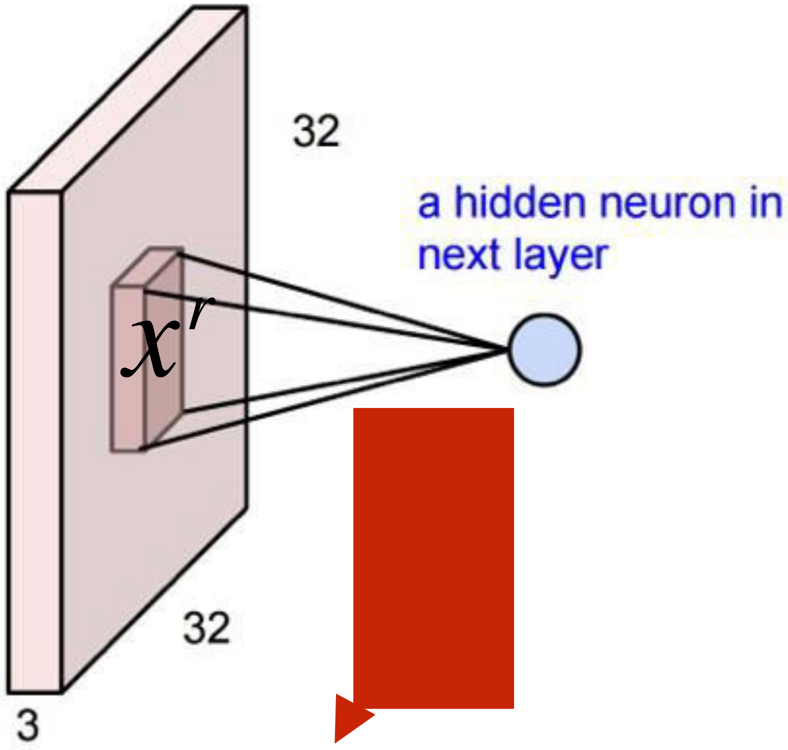
all input channels

input region

first filter

all channels

3D Activations



Let’s code this up in NumPy

```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1] * W[0, :, :, :]) + b[0]
```

nth example

first filter

output position

nth example

all input channels

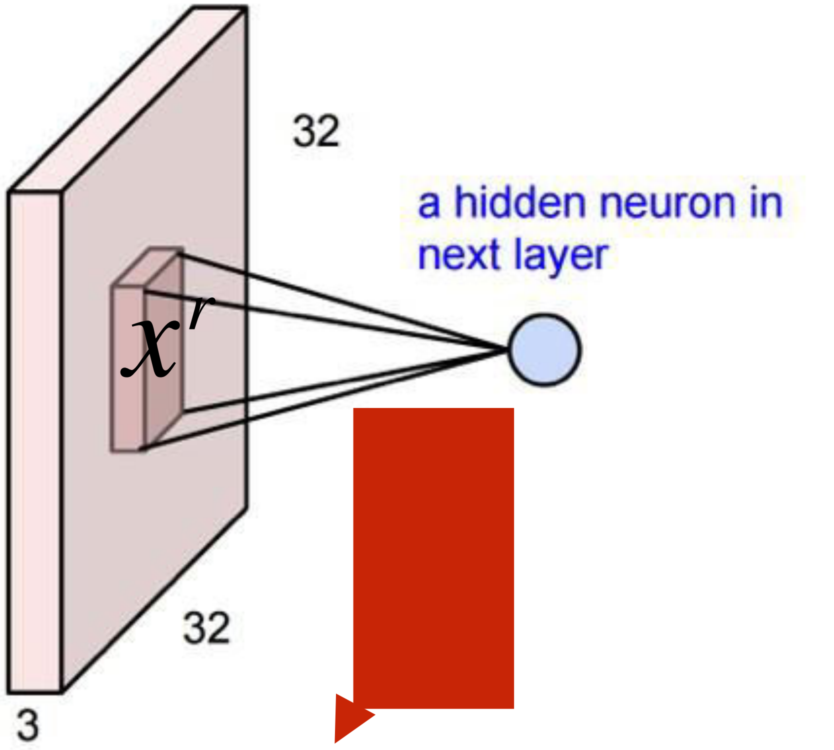
input region

first filter

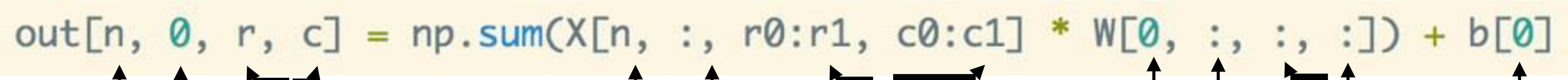
all channels

all positions

3D Activations



Let's code this up in NumPy



```
out[n, 0, r, c] = np.sum(X[n, :, r0:r1, c0:c1] * W[0, :, :, :]) + b[0]
```

nth example

first filter

output position

nth example

all input channels

input region

first filter

all channels

all positions

bias

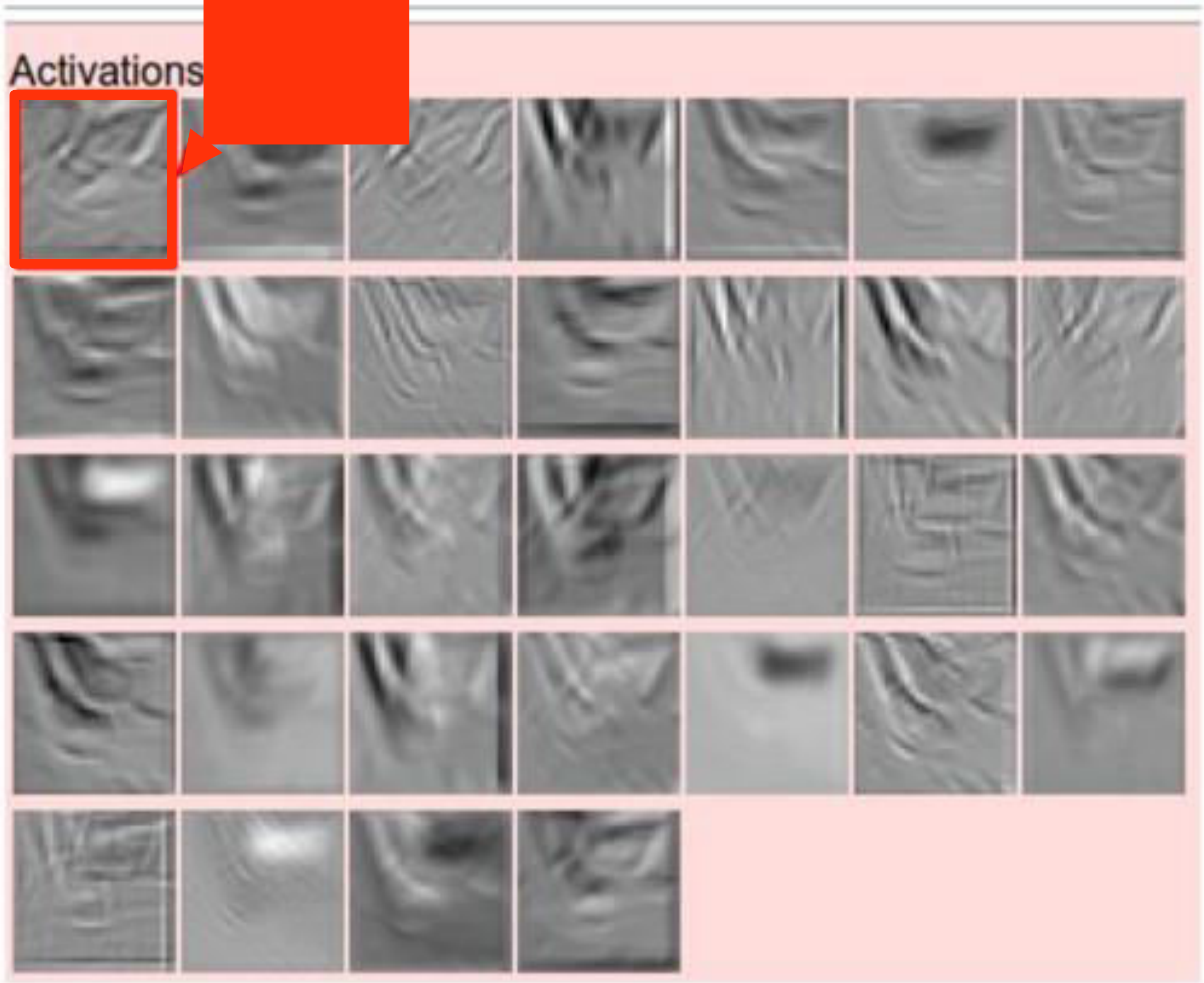
3D Activations

We can unravel the 3D cube and show each layer separately:

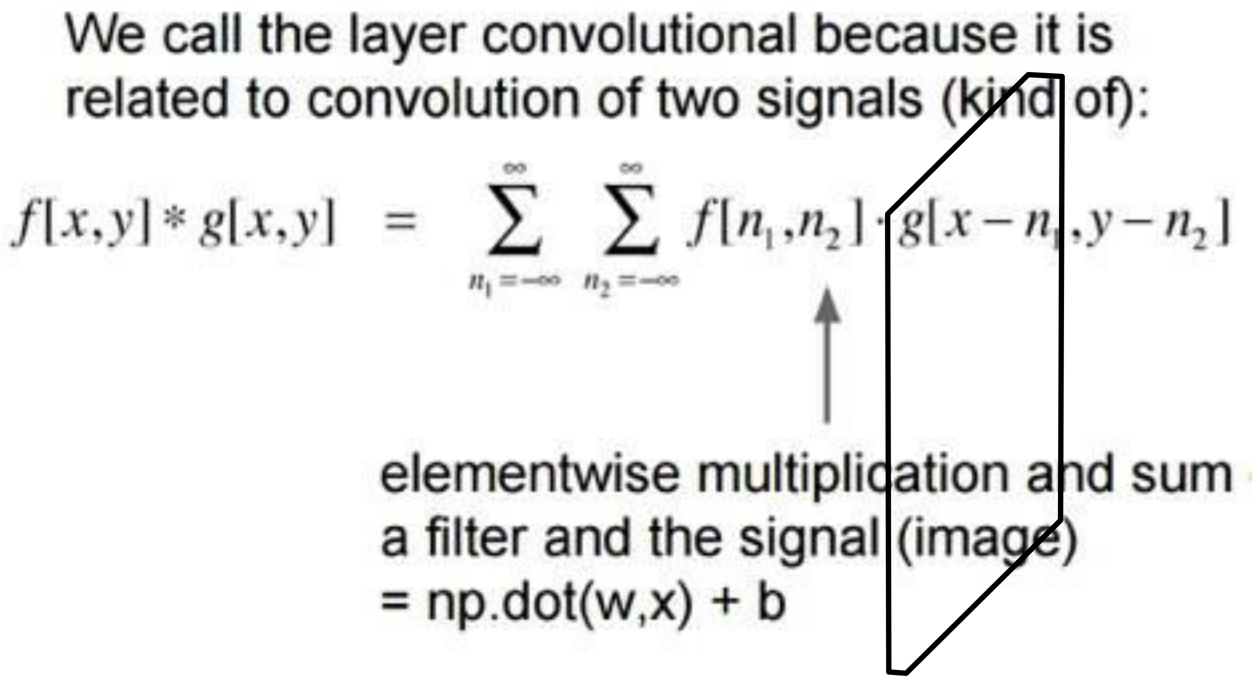
(Input)

one filter = one depth slice (or activation map)

(32 filters, each 3x5x5)



Can call the neurons "filters"



We call the layer convolutional because it is related to convolution of two signals (kind of):

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum a filter and the signal (image)

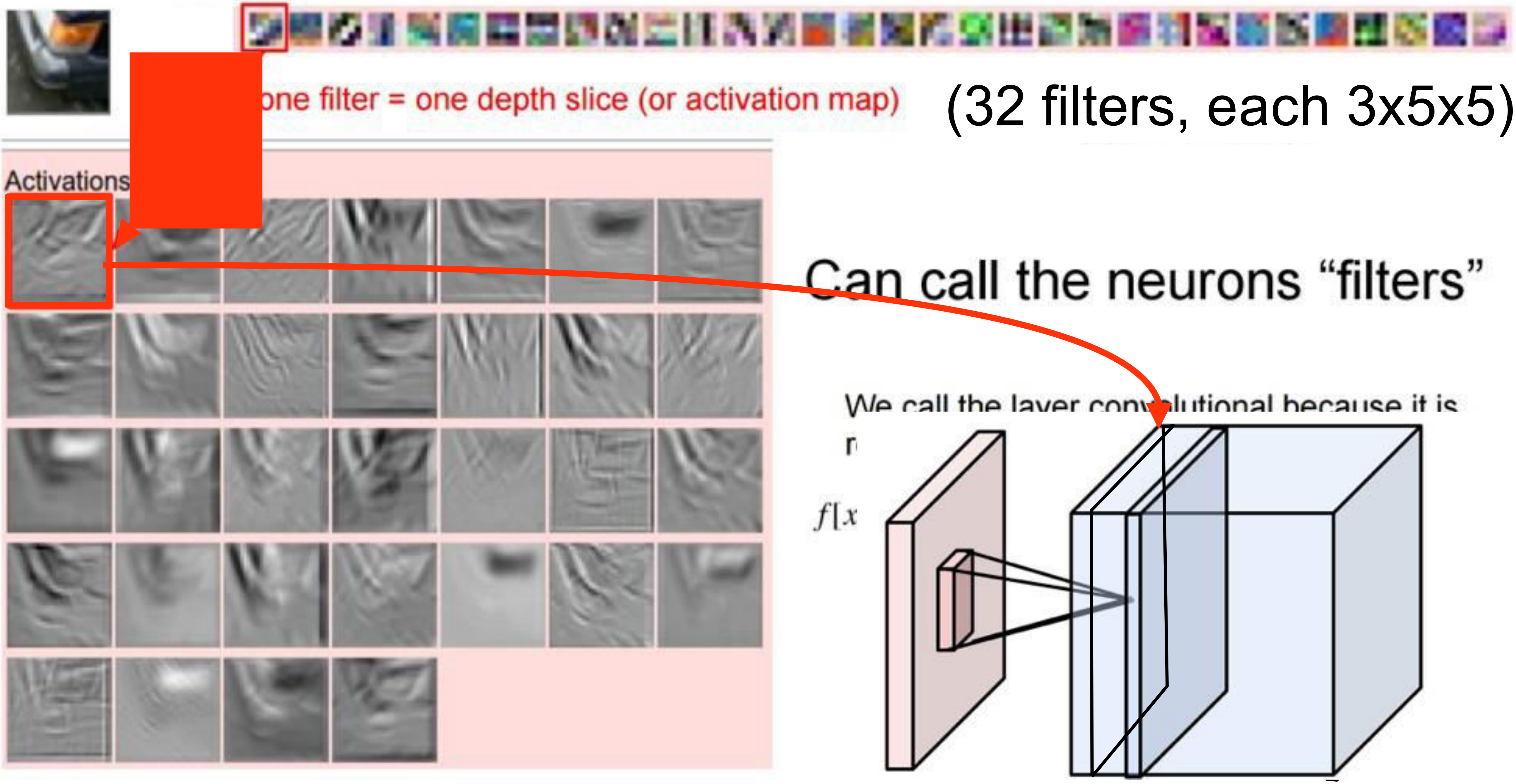
= np.dot(w,x) + b

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



(32 filters, each 3x5x5)

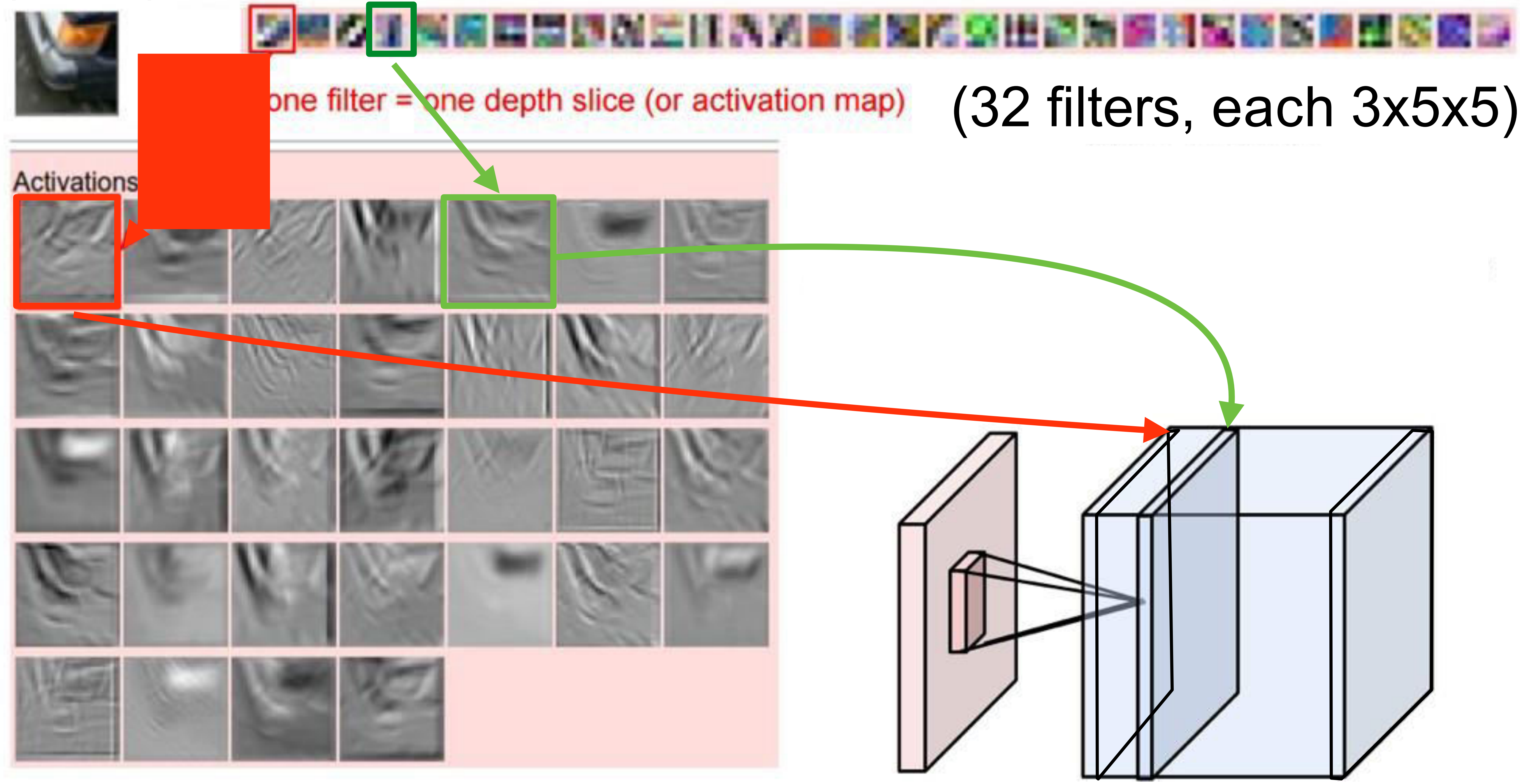
Can call the neurons “filters”

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



[redacted]

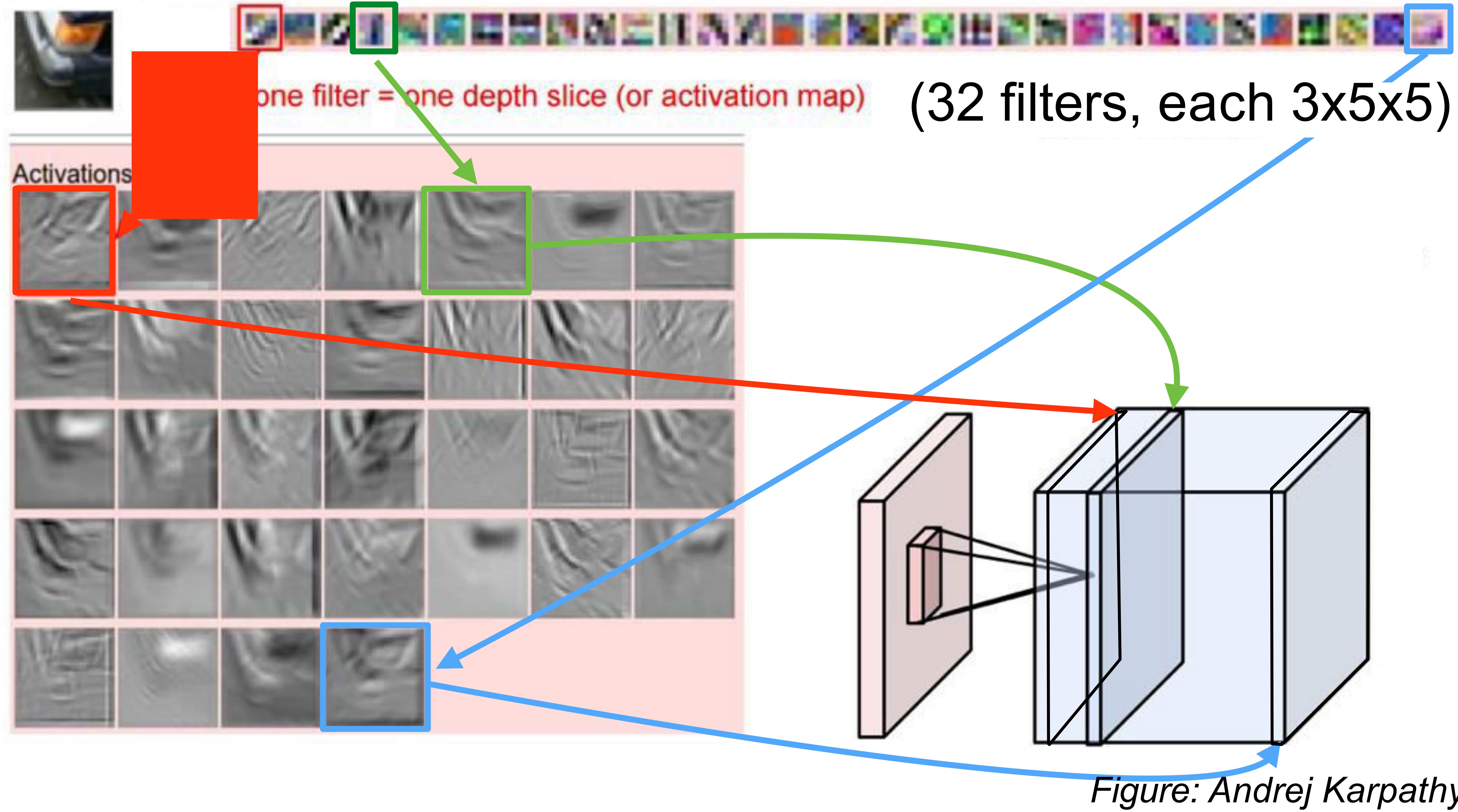
(32 filters, each 3x5x5)

Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

(Input)



(32 filters, each 3x5x5)

Figure: Andrej Karpathy

Questions?