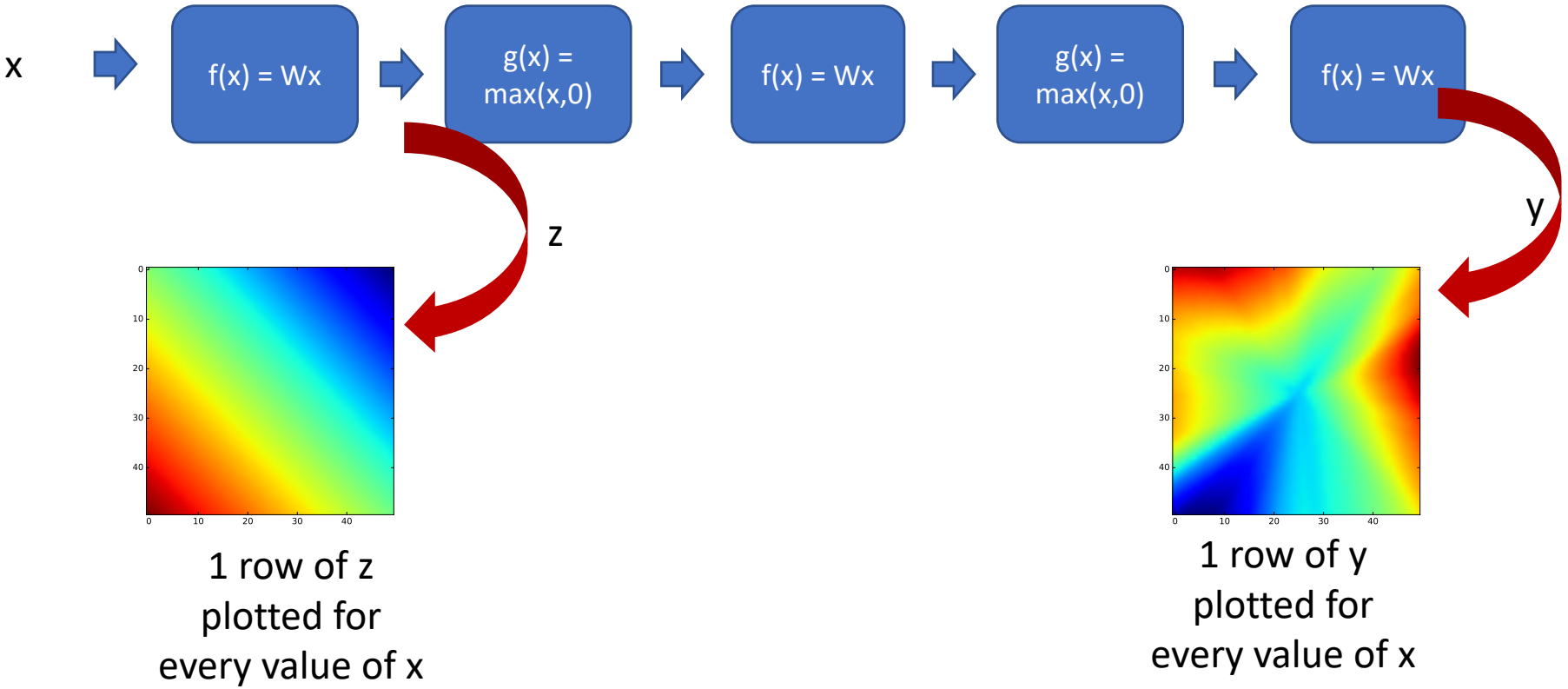
Convolutional network training

Multilayer perceptrons

- Key idea: build complex functions by composing simple functions



Linear function + translation invariance = *convolution*

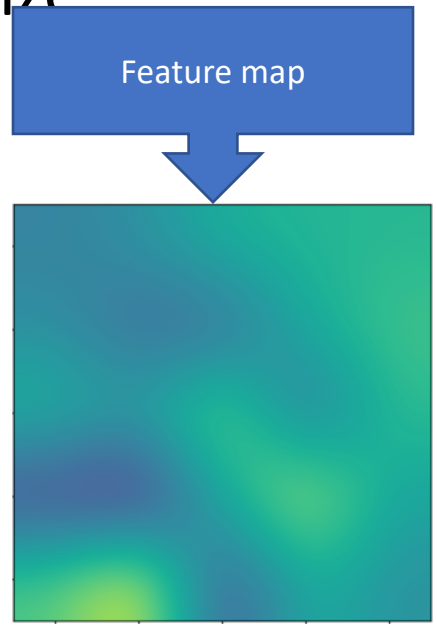
- Local connectivity determines kernel size

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

Linear function + translation invariance = *convolution*

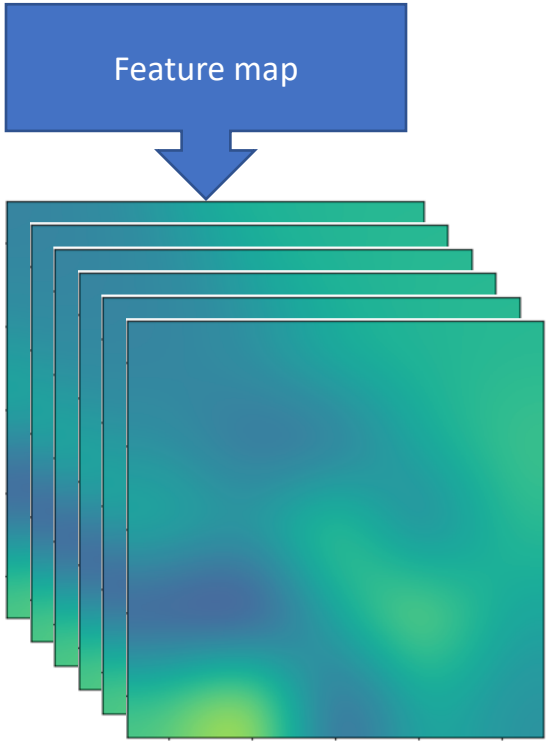
- Local connectivity determines kernel size

5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2

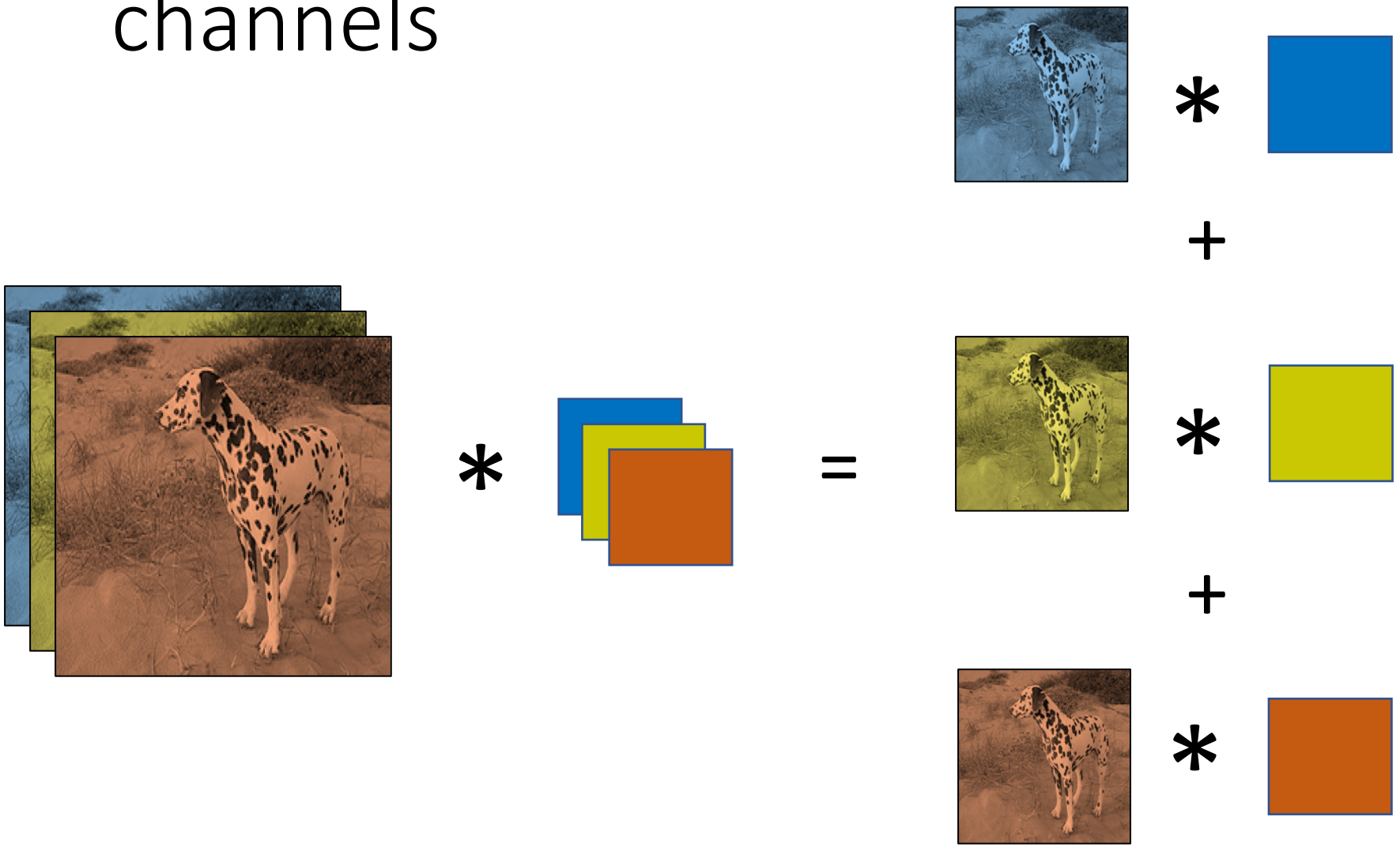


Convolution with multiple filters

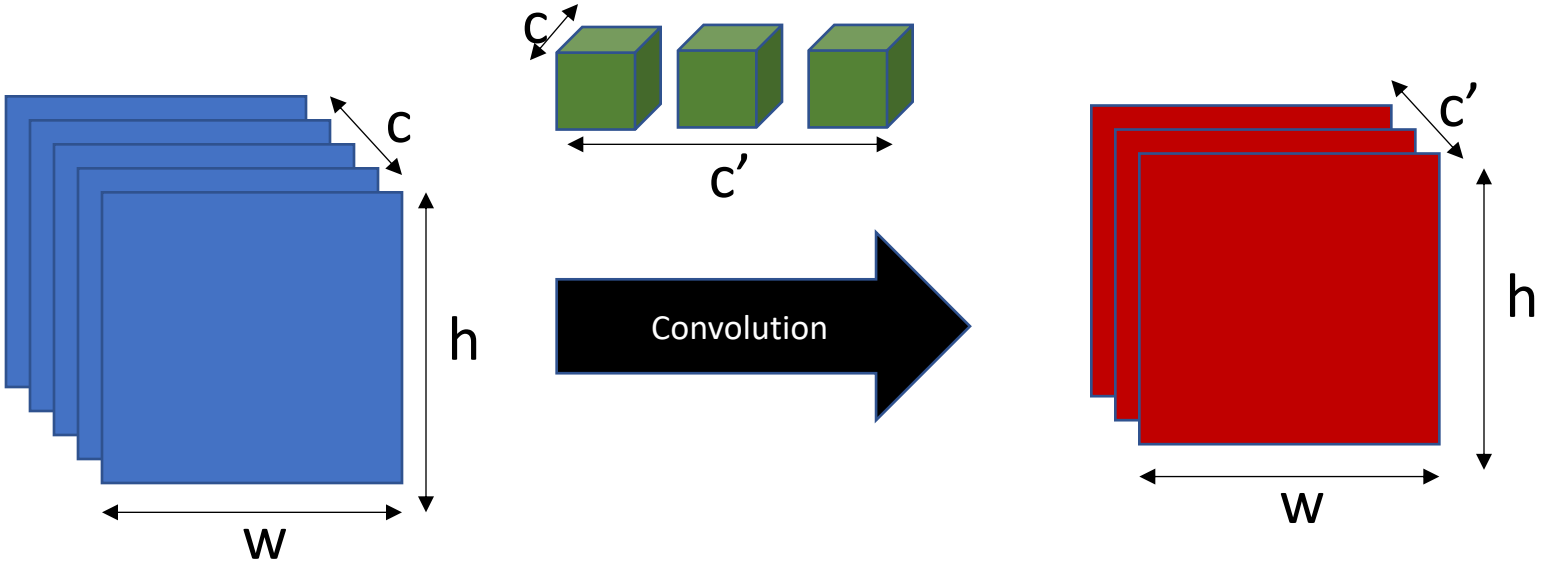
5.4	0.1	3.6
1.8	2.3	4.5
1.1	3.4	7.2



Convolution over multiple channels



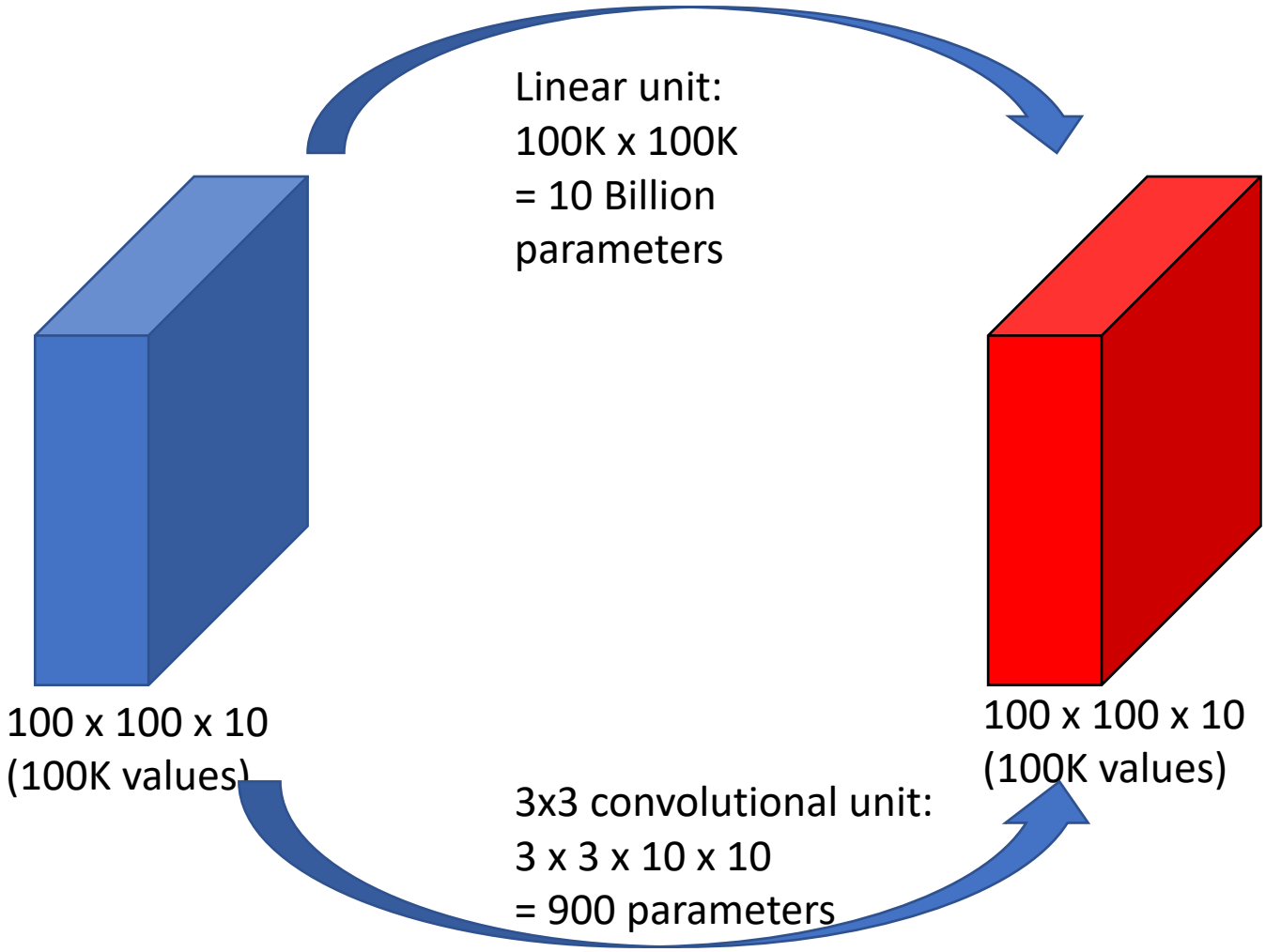
Convolution as a primitive



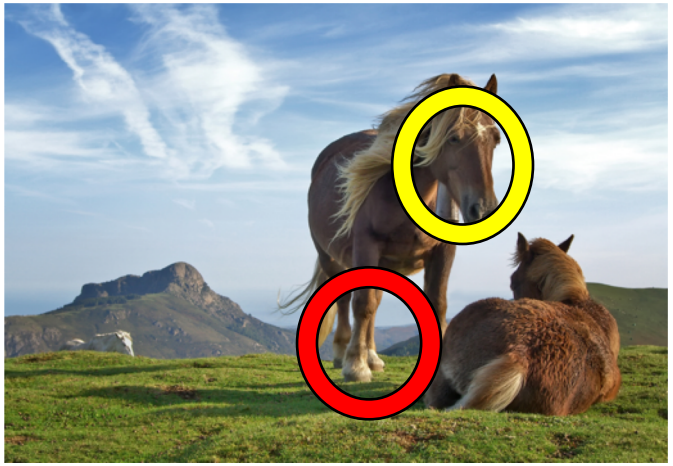
The convolution unit

- Each convolutional unit takes *a collection of feature maps* as input, and produces *a collection of feature maps* as output
- Parameters: Filters (+bias)
- If c_{in} input feature maps and c_{out} output feature maps
  - Each filter is k x k x c_{in}
  - There are c_{out} such filters
- Other hyperparameters: padding

Convolution vs Linear unit

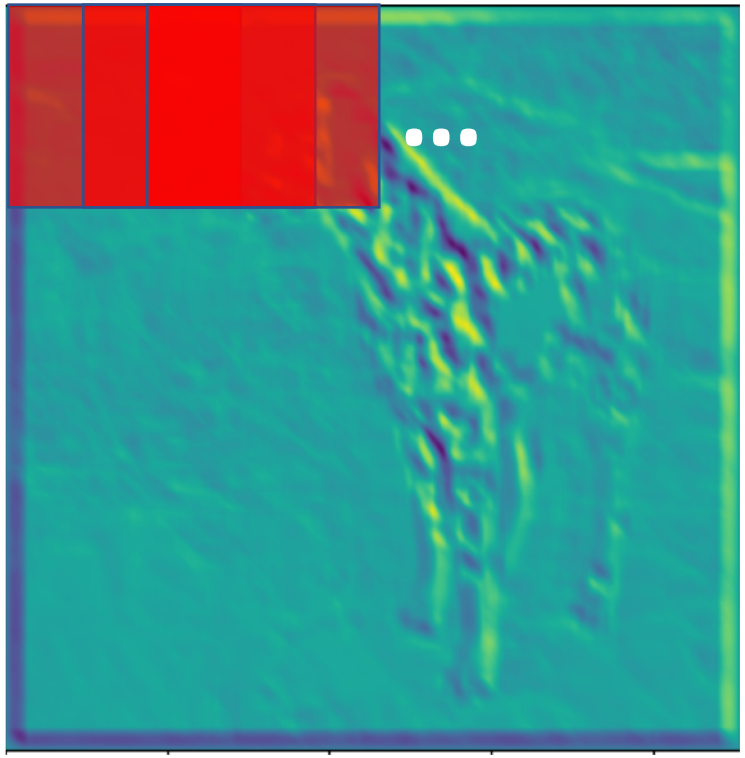


Invariance to distortions

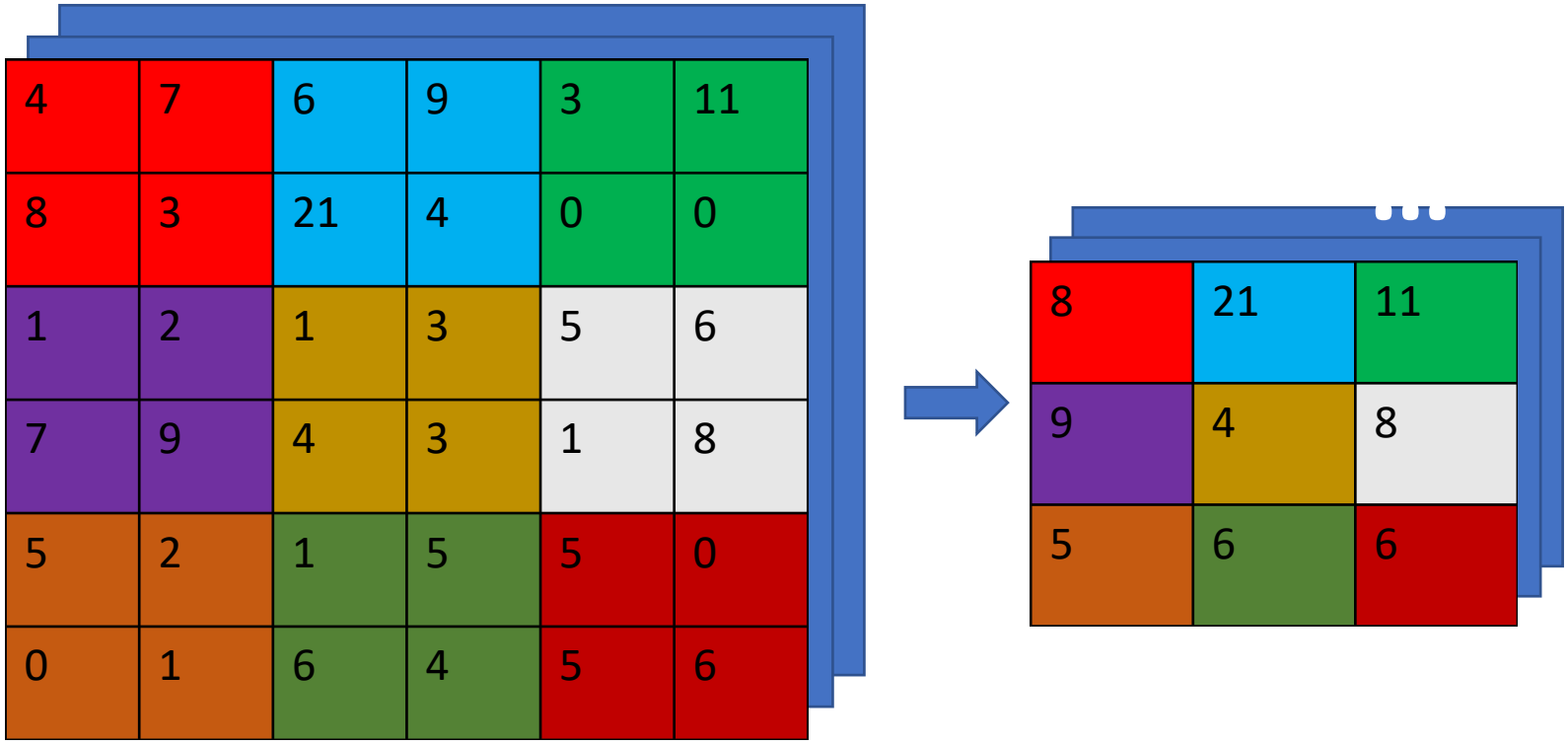


Invariance to distortions: Pooling

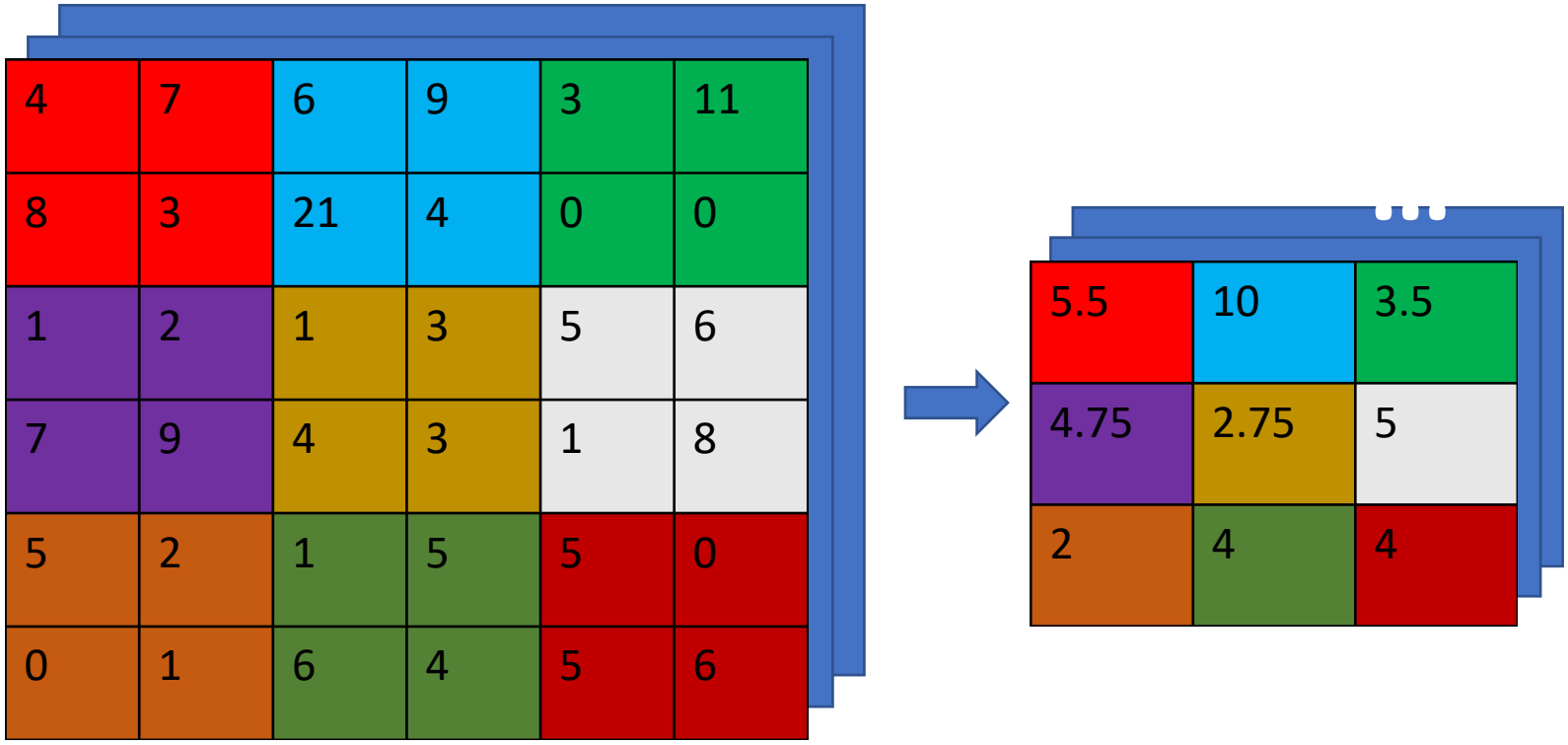
- Each window is reduced to a single value, example max or average



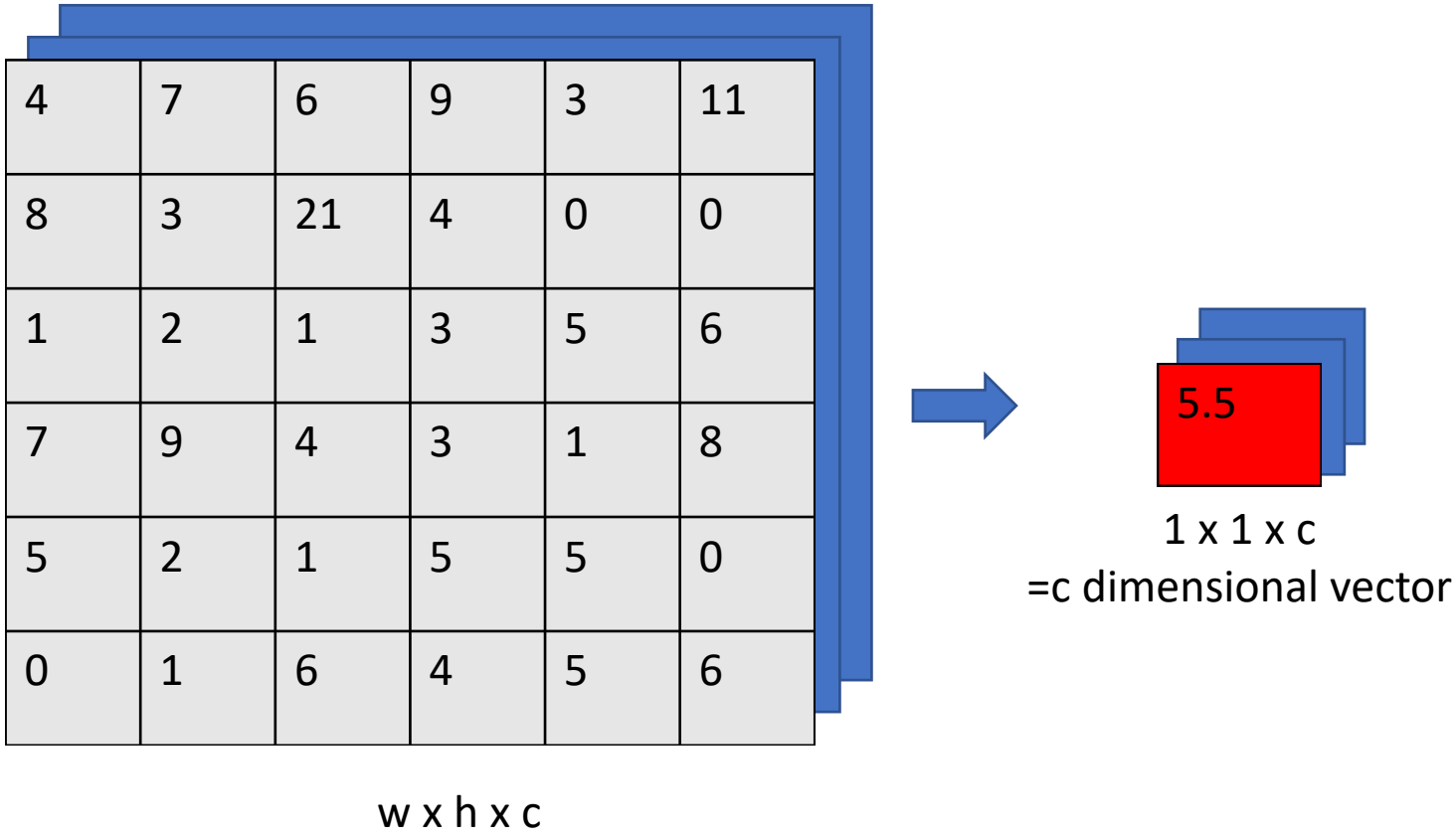
Invariance to distortions: Max Pooling



Invariance to distortions: Average Pooling



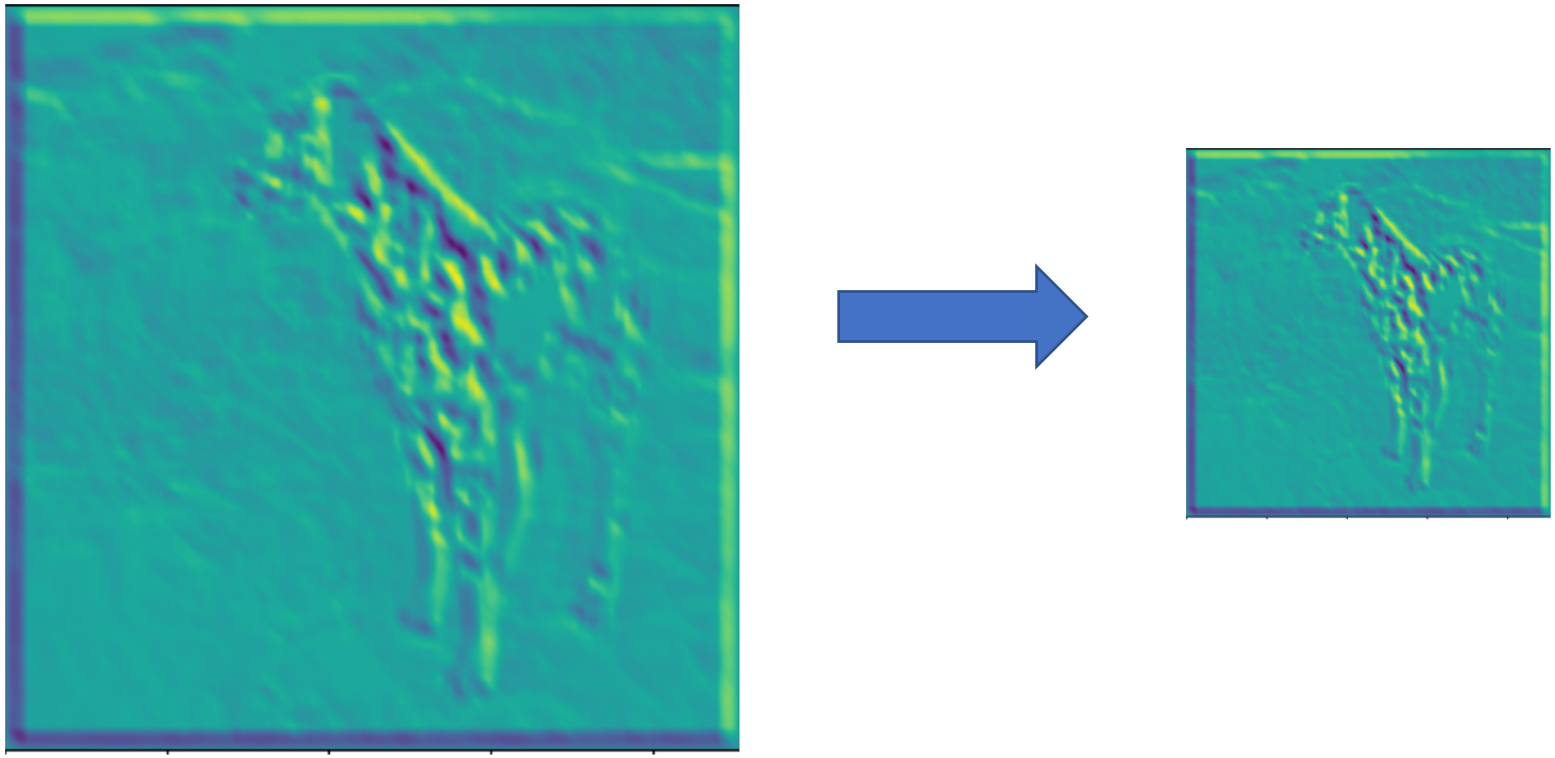
Global average pooling



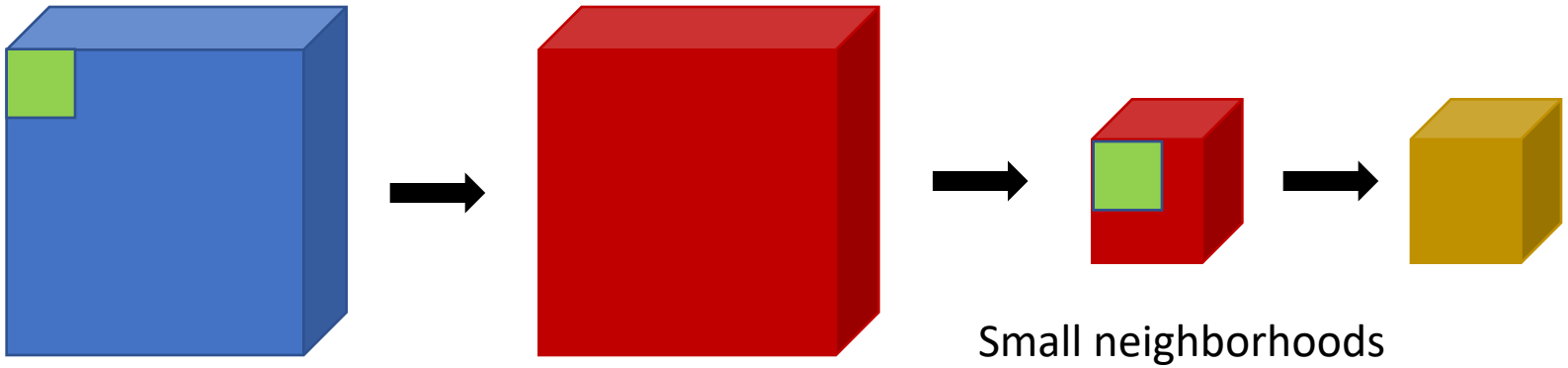
The pooling unit

- Each pooling unit takes *a collection of feature maps* as input and produces *a collection of feature maps* as output
- Output feature maps are usually smaller in height / width
- Parameters: None

Invariance to distortions: Subsampling



Convolution subsampling convolution



Small neighborhoods on subsampled feature map = large neighborhood on original image

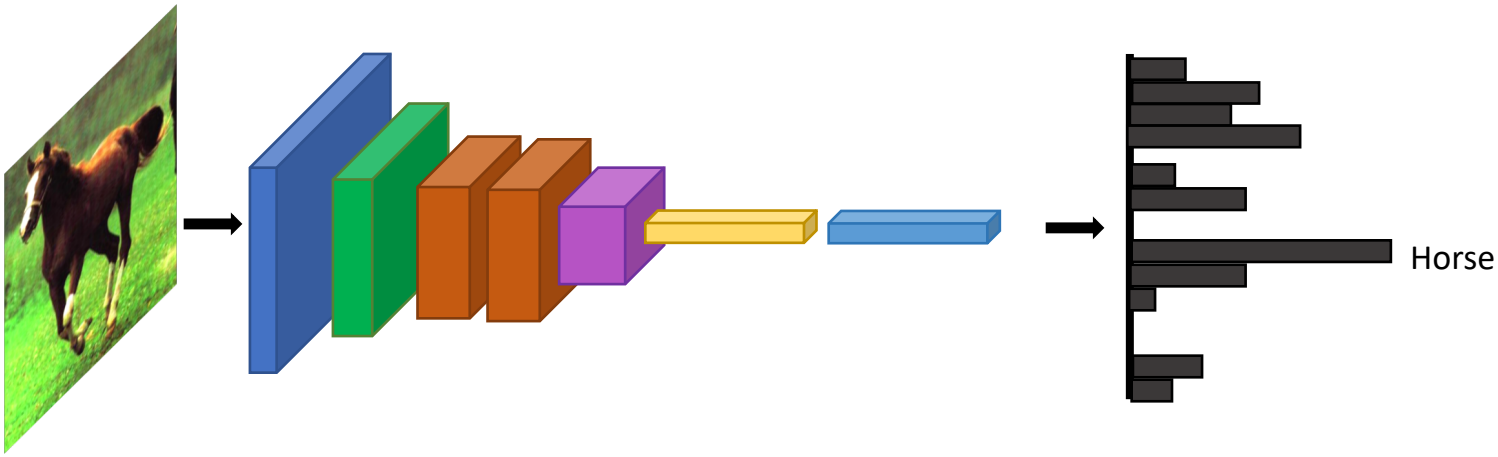
Convolution subsampling convolution

- Convolution in earlier steps detects *more local* patterns *less resilient* to distortion
- Convolution in later steps detects *more global* patterns *more resilient* to distortion
- Subsampling allows capture of *larger, more invariant* patterns

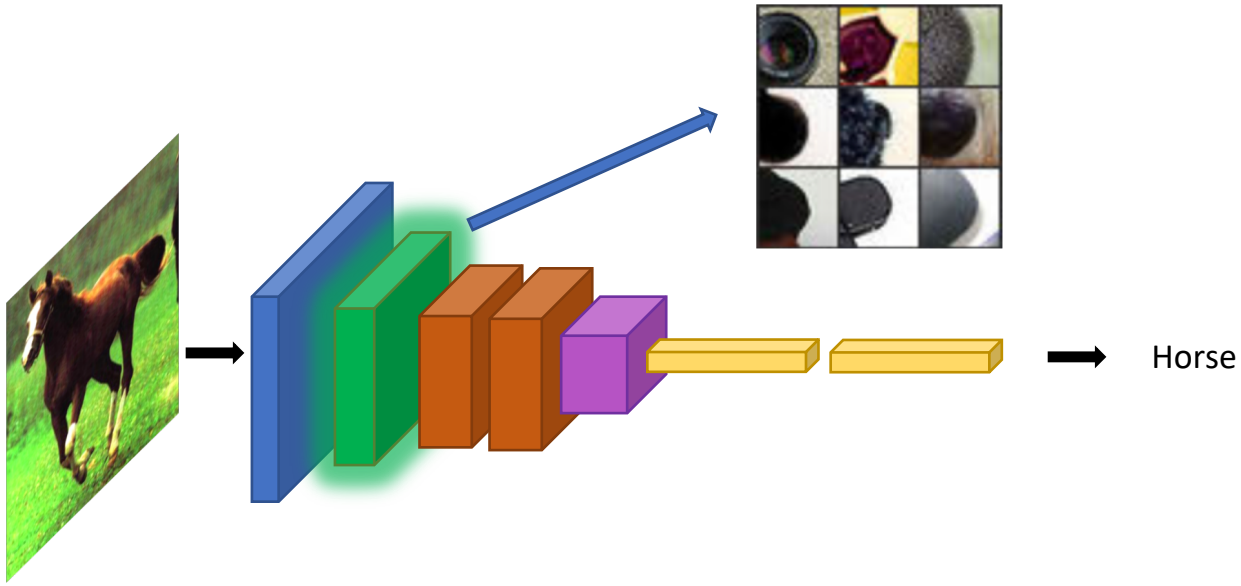
Strided convolution

- Convolution with stride s = standard convolution + subsampling by picking 1 value every s values
- Example: convolution with stride 2 = standard convolution + subsampling by a factor of 2

Convolutional networks

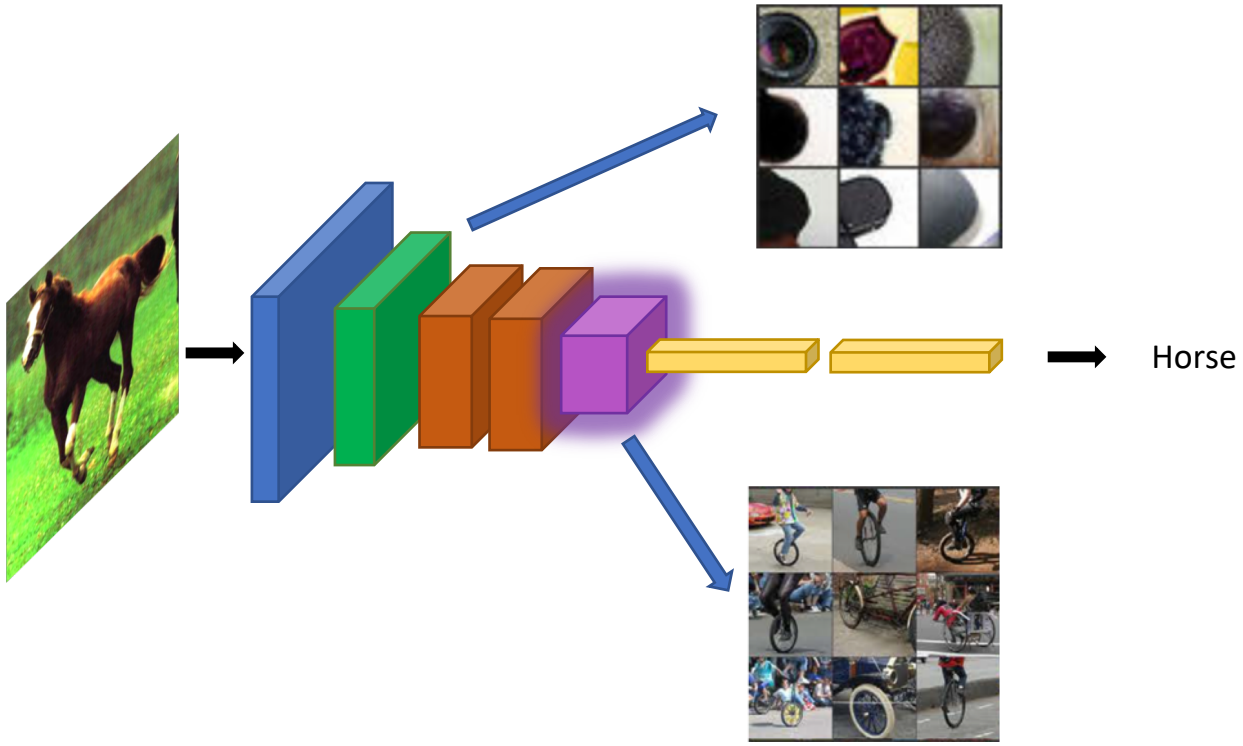


Convolutional networks



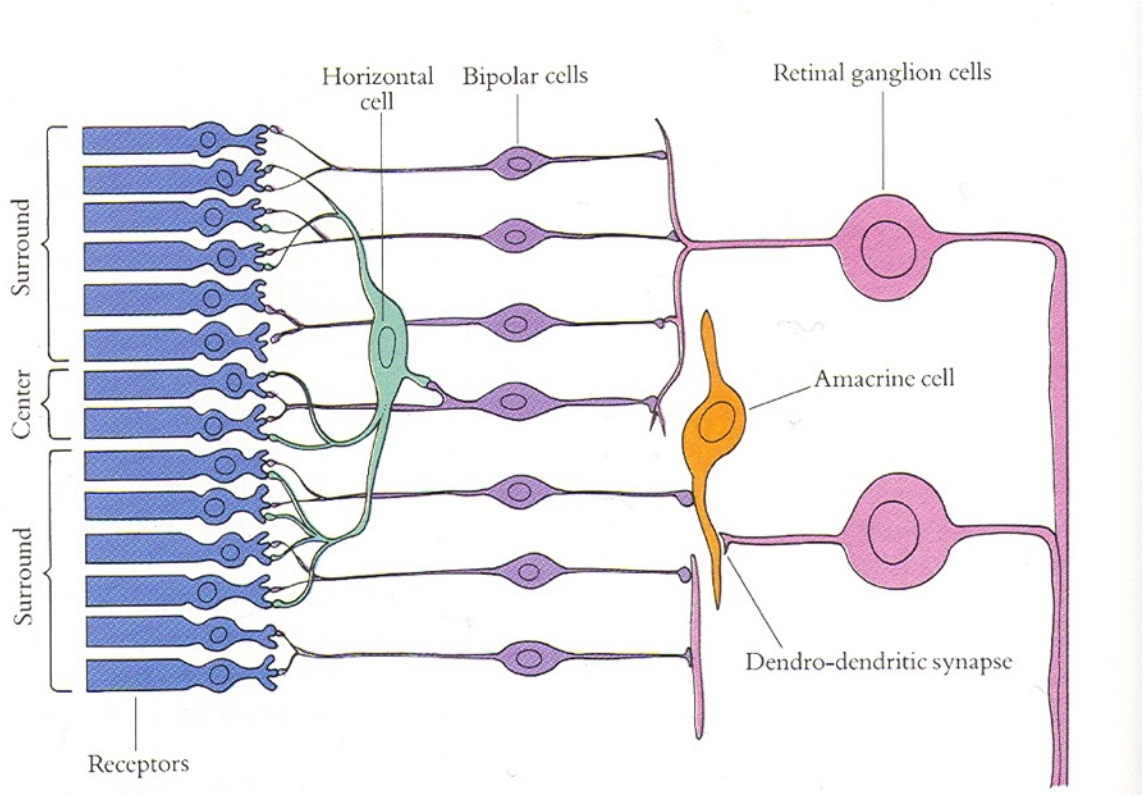
Visualizations from : M. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In ECCV 2014.

Convolutional networks



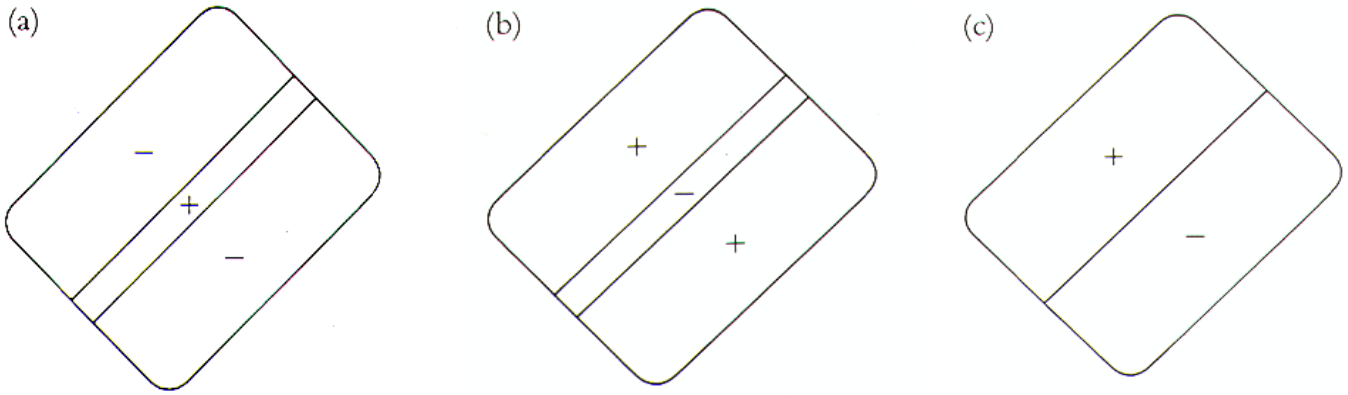
Visualizations from : M. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *ECCV* 2014.

Convolutional Networks and the Brain



Slide credit: Jitendra Malik

Receptive fields of simple cells (discovered by Hubel & Wiesel)



Slide credit: Jitendra Malik

Convolutional networks

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86.11 (1998): 2278-2324.

Convolutional networks

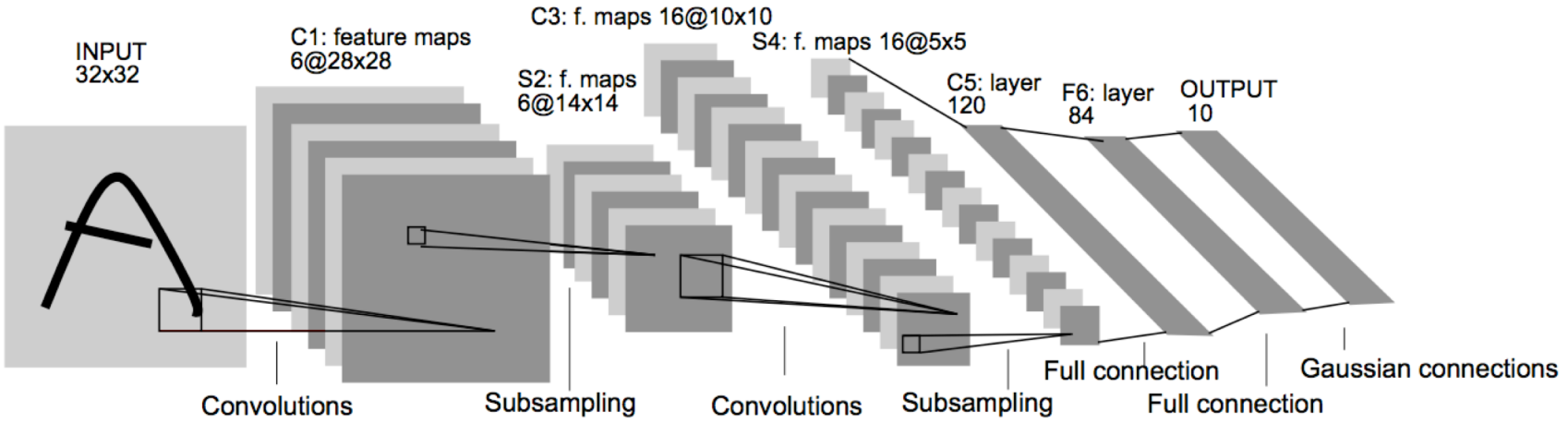
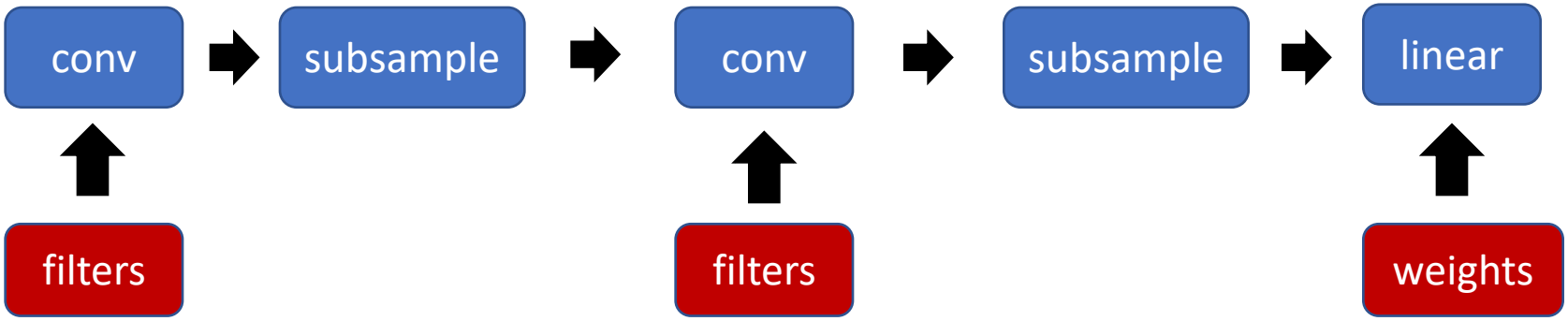


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

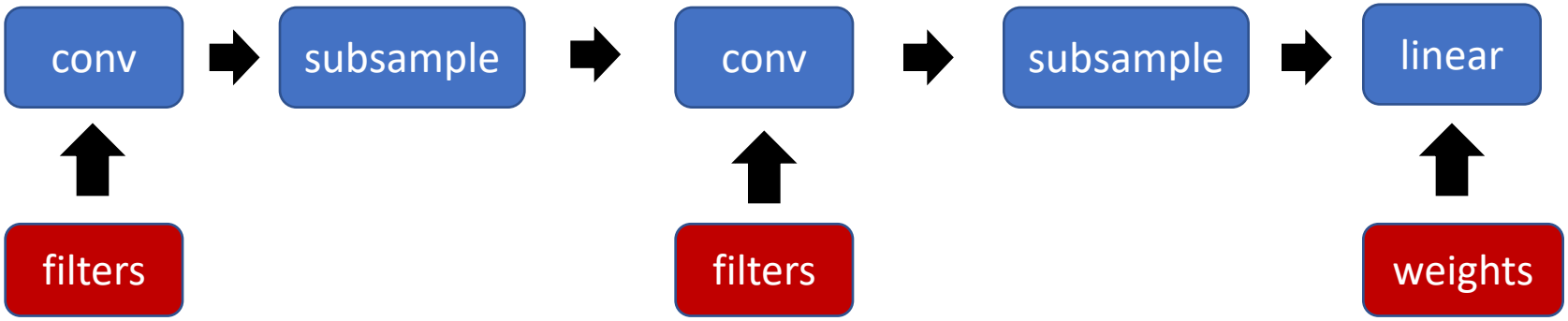
Convolutional networks



Last time

- Linear classifiers on pixels bad, need non-linear classifiers
- Multi-layer perceptrons overparametrized
- Reduce parameters by local connections and shift invariance => Convolution
- Intersperse subsampling to capture ever larger deformations
- Stick a final classifier

Convolutional networks



Empirical Risk Minimization

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} L(h(x_i; \boldsymbol{\theta}), y_i)$$

Convolutional network

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \lambda \frac{1}{N} \sum_{i=1}^{N} \nabla L(h(x_i; \boldsymbol{\theta}), y_i)$$

Gradient descent update

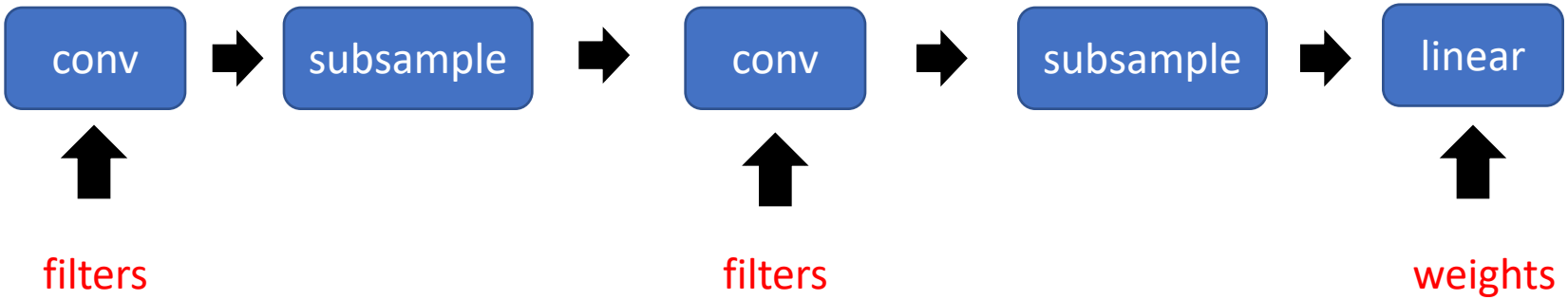
Computing the gradient of the loss

$$\nabla L(h(x;\boldsymbol{\theta}), y)$$

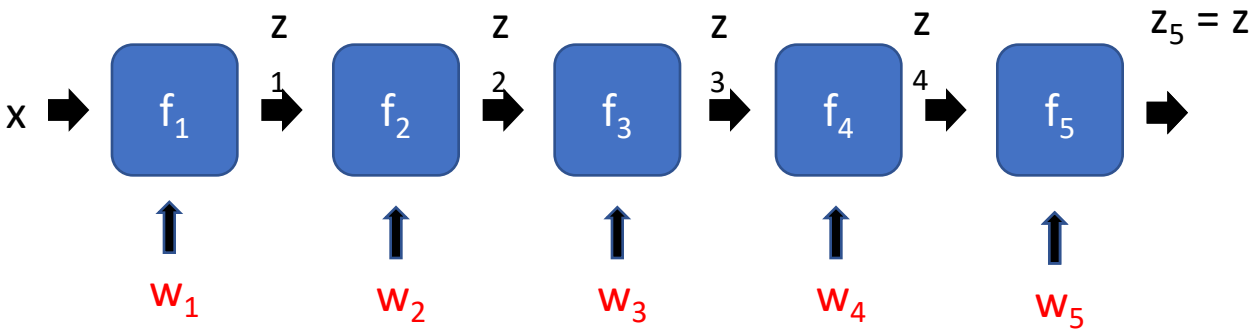
$$z = h(x;\boldsymbol{\theta})$$

$$\nabla_{\boldsymbol{\theta}} L(z, y) = \frac{\partial L(z, y)}{\partial z} \frac{\partial z}{\partial \boldsymbol{\theta}}$$

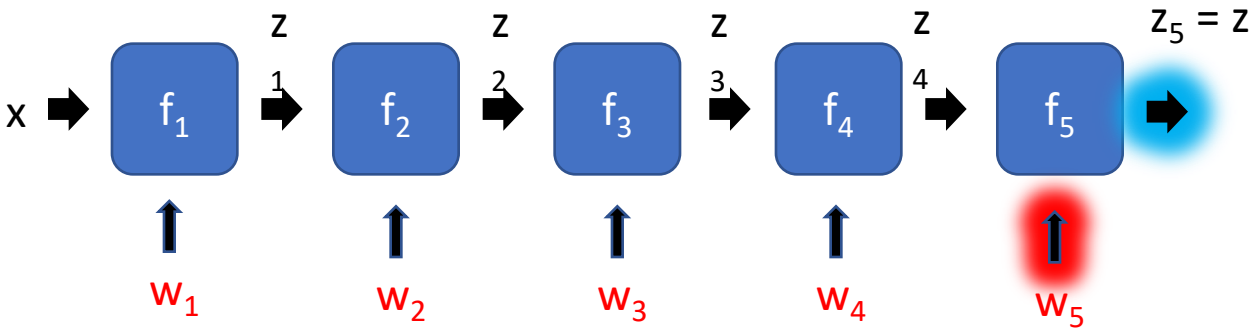
Convolutional networks



The gradient of convnets

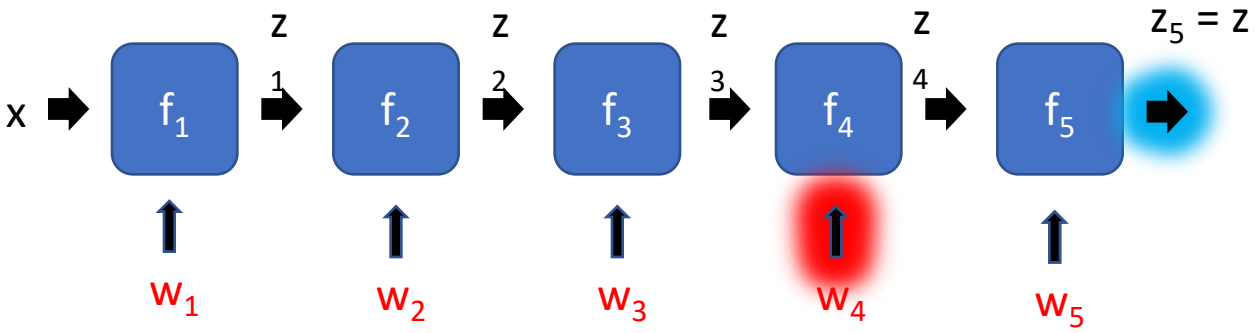


The gradient of convnets



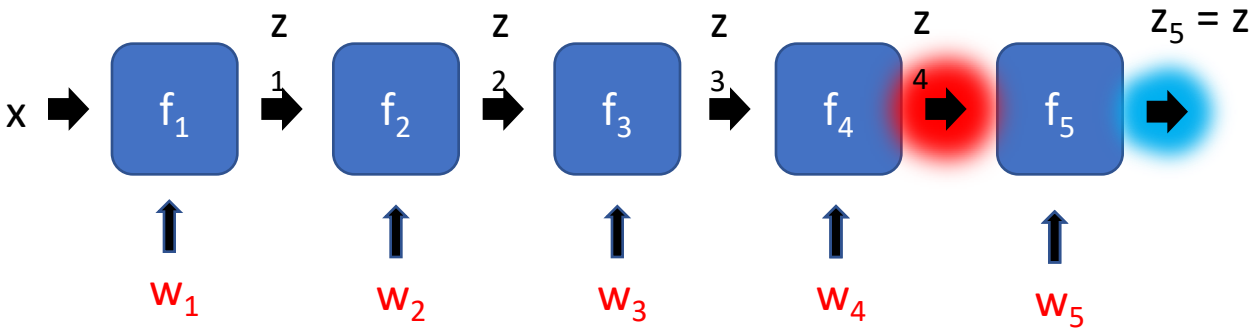
$$\frac{\partial z}{\partial w_5}$$

The gradient of convnets



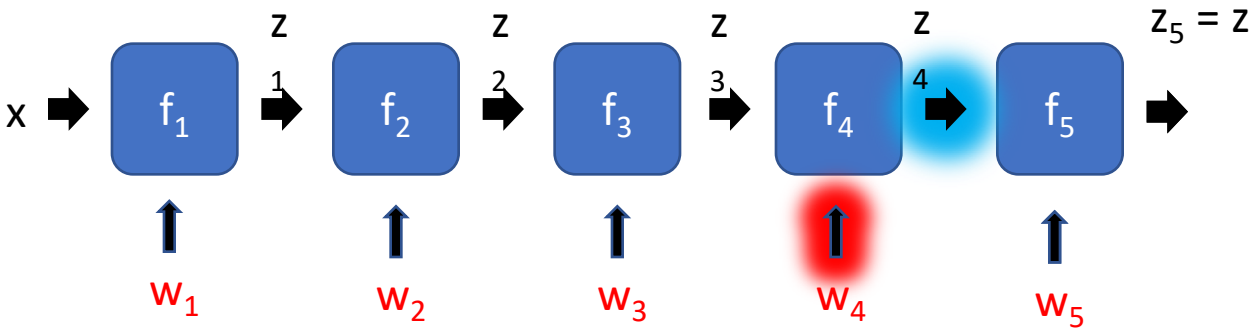
$$\frac{\partial z}{\partial w_4}$$

The gradient of convnets



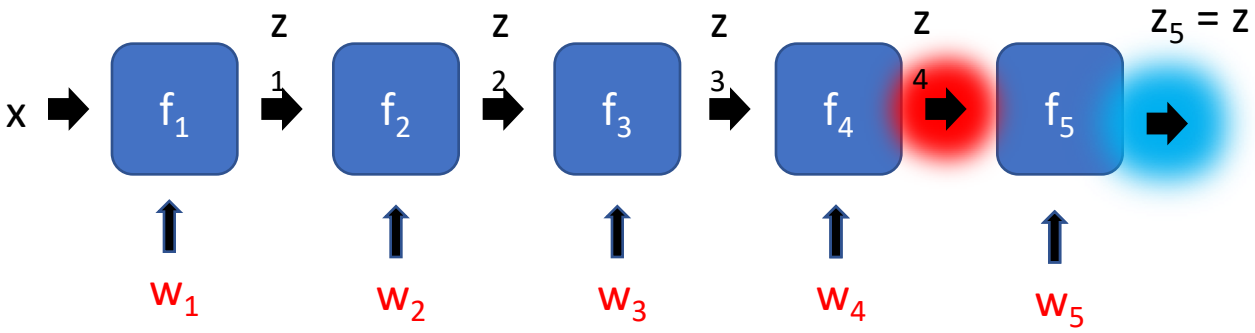
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial w_4}$$

The gradient of convnets



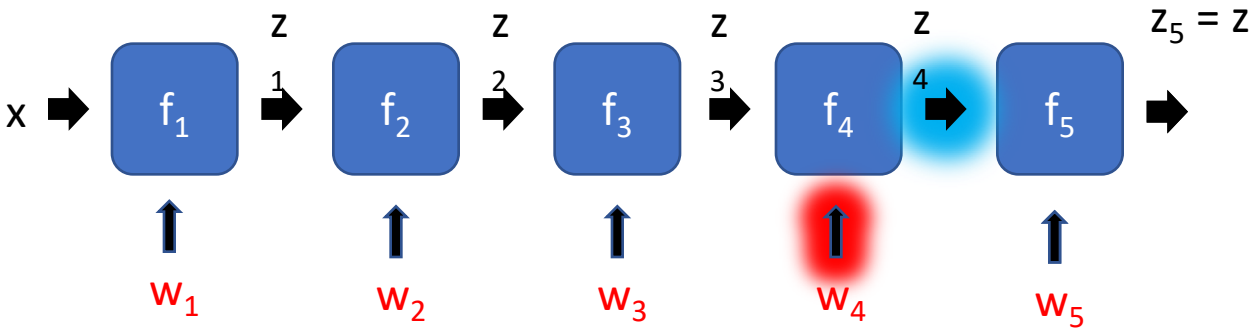
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial w_4}$$

The gradient of convnets



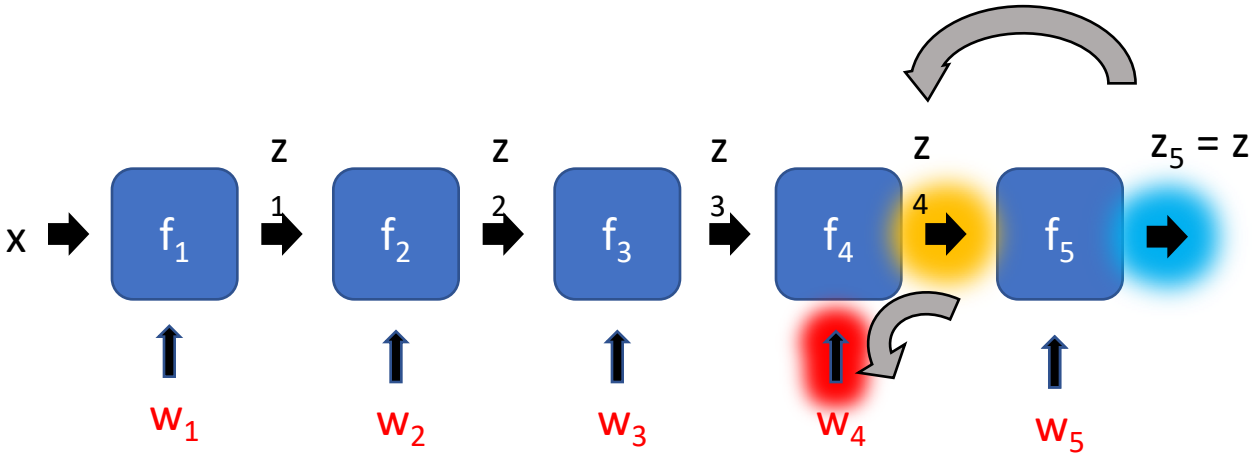
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4}\frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

The gradient of convnets



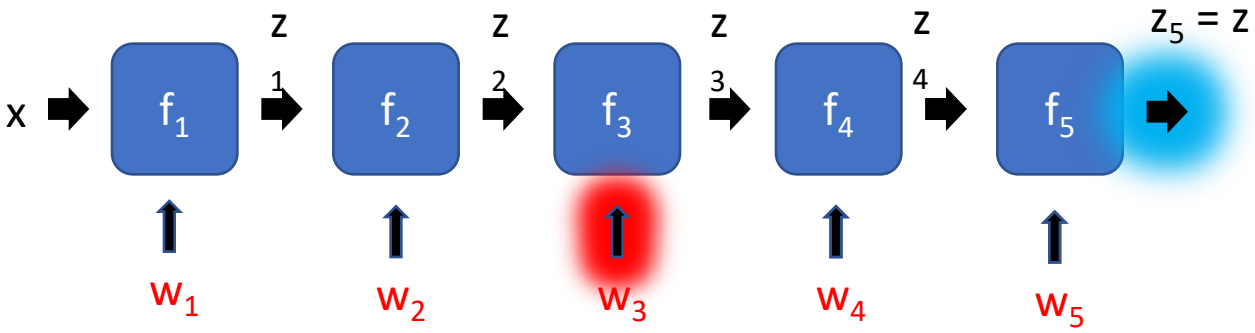
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4}\frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

The gradient of convnets



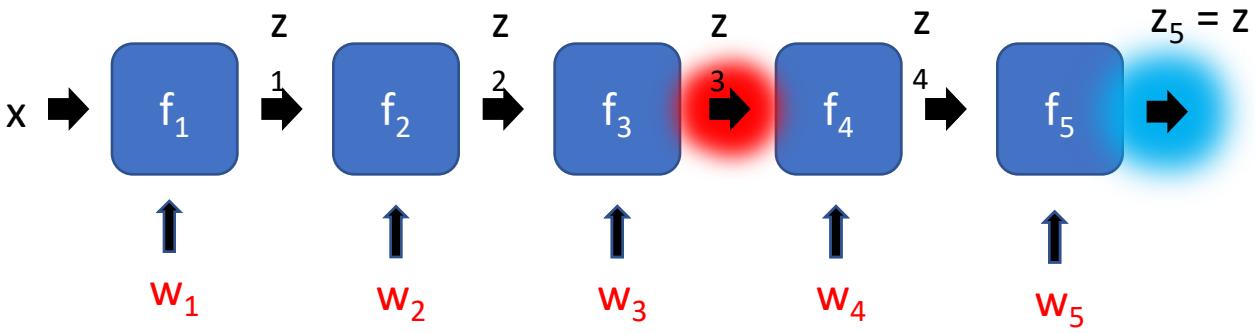
$$\frac{\partial z}{\partial w_4} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial w_4} = \frac{\partial f_5(z_4, w_5)}{\partial z_4}\frac{\partial f_4(z_3, w_4)}{\partial w_4}$$

The gradient of convnets



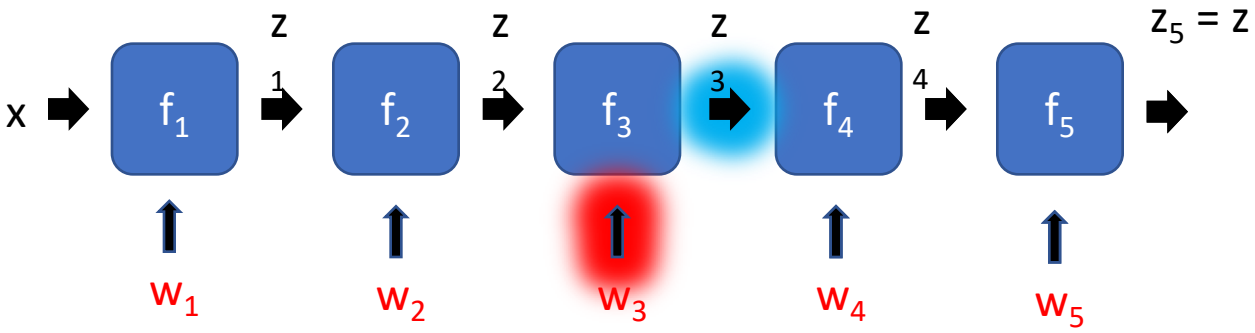
$$\frac{\partial z}{\partial w_3}$$

The gradient of convnets



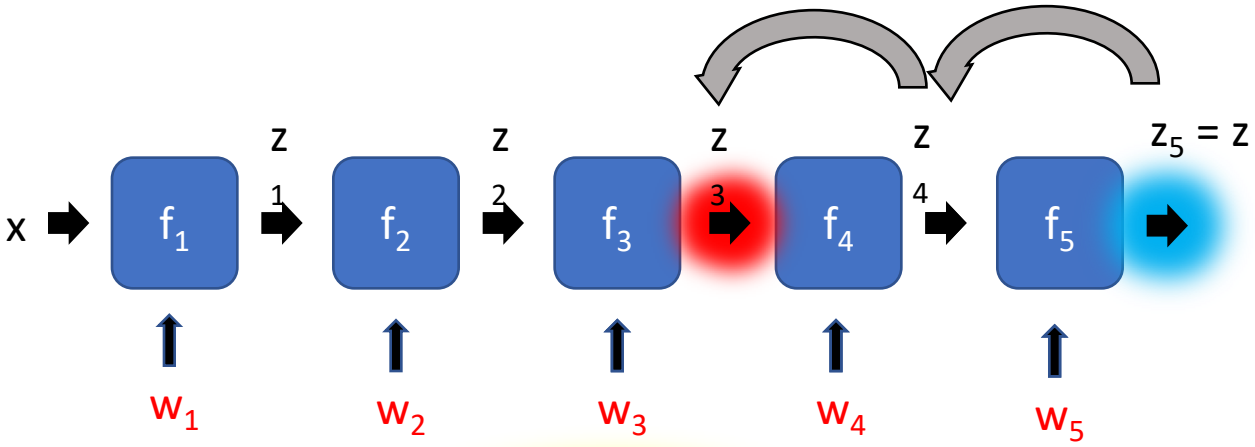
$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

The gradient of convnets



$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

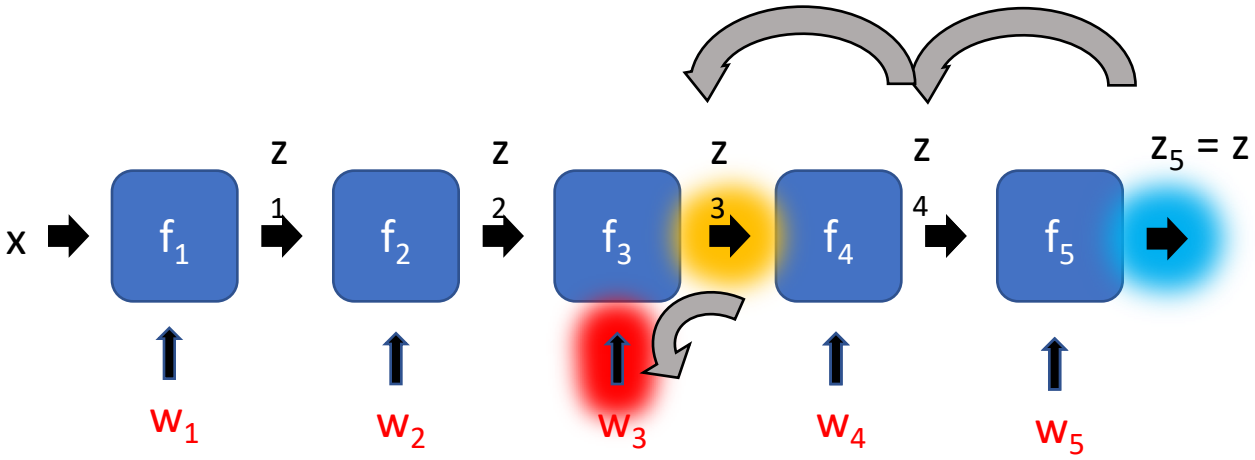
The gradient of convnets



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

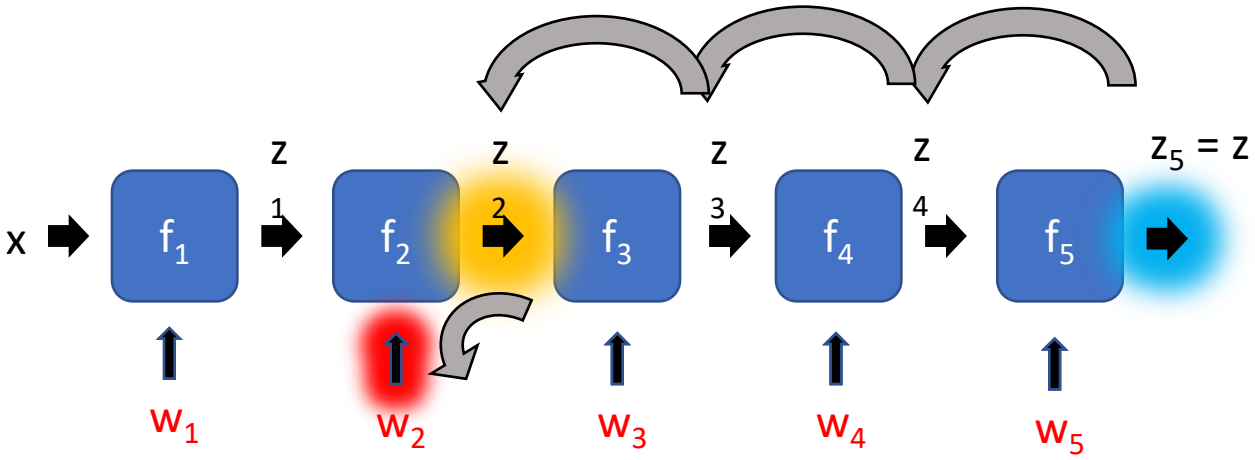
The gradient of convnets



$$\frac{\partial z}{\partial z_3} = \frac{\partial z}{\partial z_4}\frac{\partial z_4}{\partial z_3}$$

$$\frac{\partial z}{\partial w_3} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial w_3}$$

The gradient of convnets

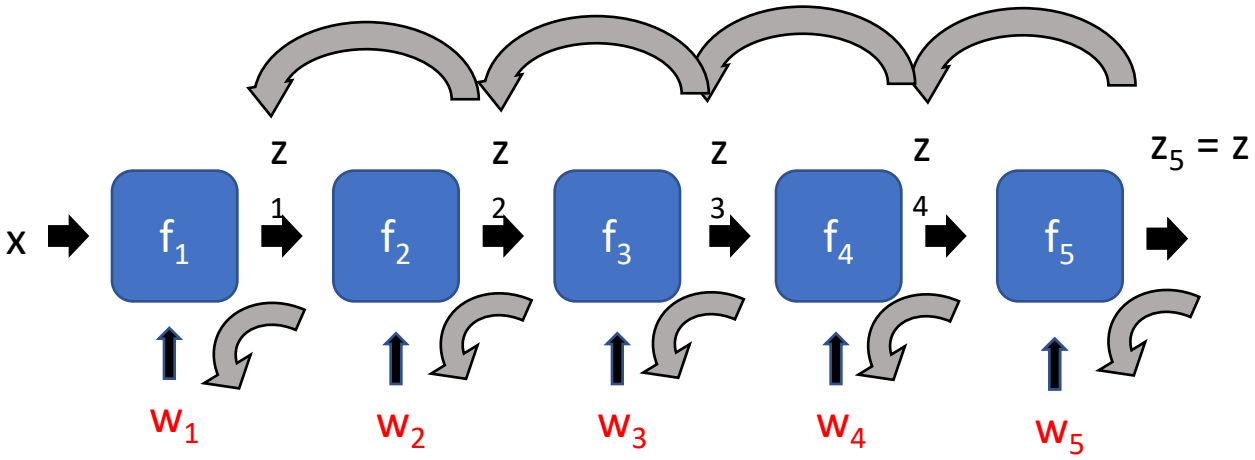


$$\frac{\partial z}{\partial z_2} = \frac{\partial z}{\partial z_3}\frac{\partial z_3}{\partial z_2}$$

$$\frac{\partial z}{\partial w_2} = \frac{\partial z}{\partial z_2}\frac{\partial z_2}{\partial w_2}$$

Recurrence going backward!!

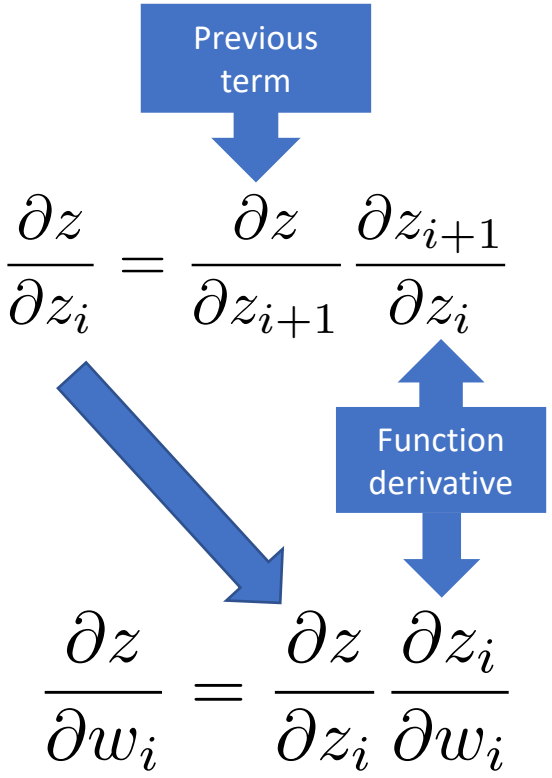
The gradient of convnets



Backpropagation

Backpropagation for a sequence of functions

$$z_i = f_i(z_{i-1}, w_i)$$
$$z_0 = x$$
$$z = z_n$$



Backpropagation for a sequence of functions

$$z_i = f_i(z_{i-1}, w_i) \qquad z_0 = x \qquad z = z_n$$

- Assume we can compute partial derivatives of each function

$$\frac{\partial z_i}{\partial z_{i-1}} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial z_{i-1}} \qquad \frac{\partial z_i}{\partial w_i} = \frac{\partial f_i(z_{i-1}, w_i)}{\partial w_i}$$

- Use $g(z_i)$ to store gradient of z w.r.t z_i, $g(w_i)$ for w_i
- Calculate g_i by iterating backwards

$$g(z_n) = \frac{\partial z}{\partial z_n} = 1 \qquad g(z_{i-1}) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial z_{i-1}} = g(z_i)\frac{\partial z_i}{\partial z_{i-1}}$$

- Use gi to compute gradient of parameters

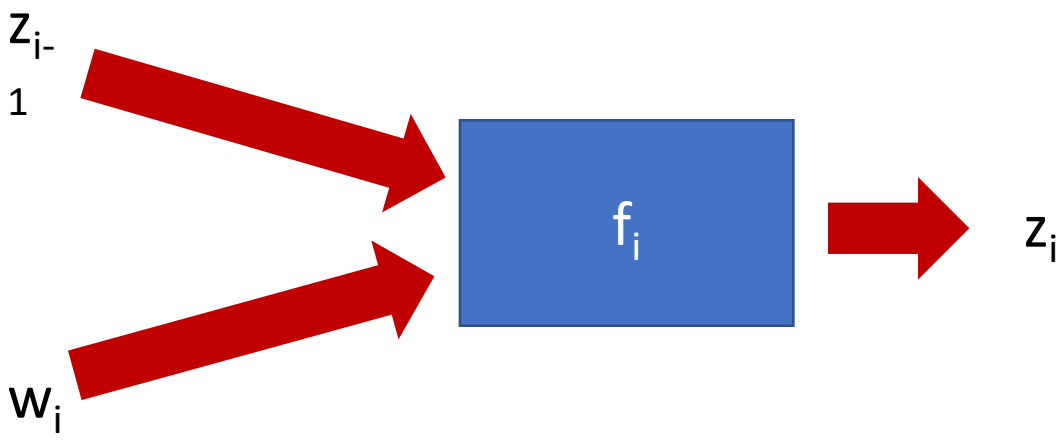
$$g(w_i) = \frac{\partial z}{\partial z_i}\frac{\partial z_i}{\partial w_i} = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions

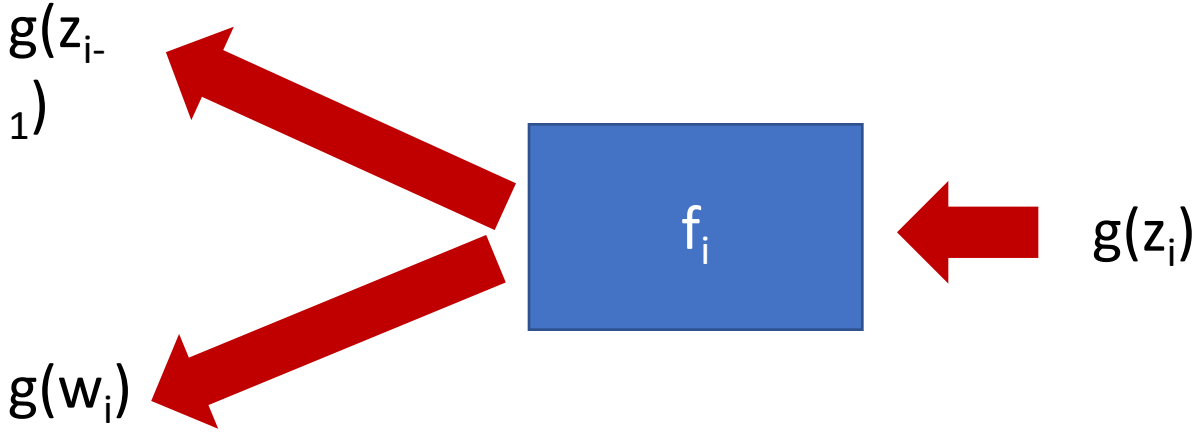
- Each “function” has a “forward” and “backward” module
- Forward module for f_i
  - takes z_{i-1} and weight w_i as input
  - produces z_i as output
- Backward module for f_i
  - takes $g(z_i)$ as input
  - produces $g(z_{i-1})$ and $g(w_i)$ as output

$$g(z_{i-1}) = g(z_i)\frac{\partial z_i}{\partial z_{i-1}} \qquad g(w_i) = g(z_i)\frac{\partial z_i}{\partial w_i}$$

Backpropagation for a sequence of functions



Backpropagation for a sequence of functions



Chain rule for vectors

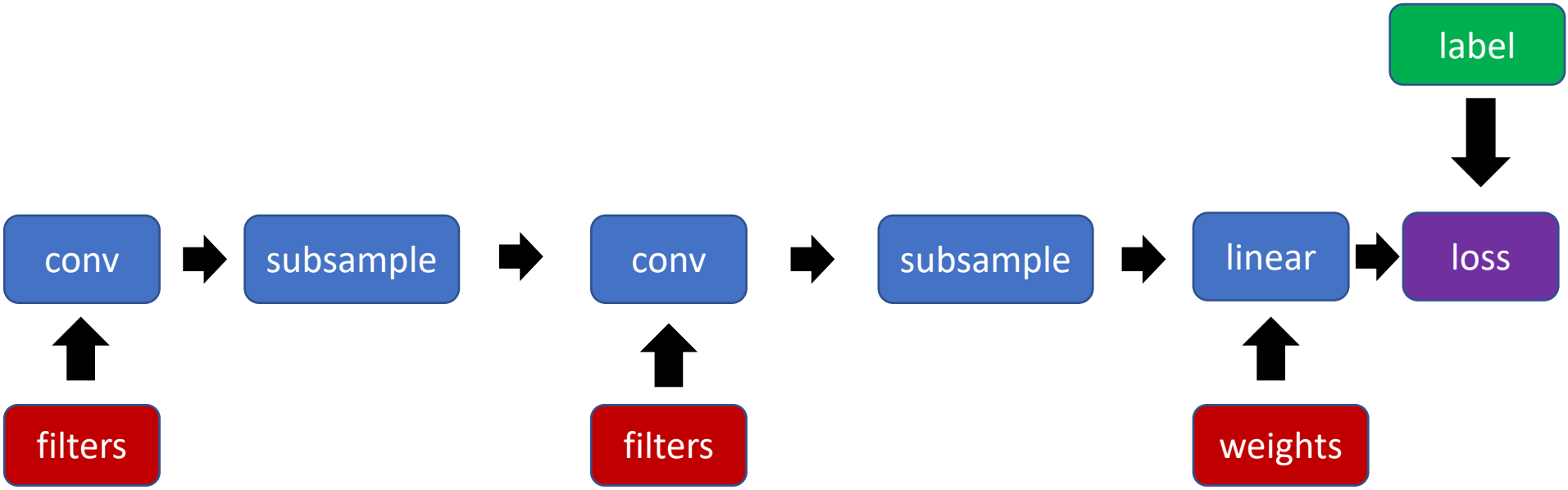
$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial c}\frac{\partial c}{\partial b} \qquad \frac{\partial a_i}{\partial b_j} = \sum_k \frac{\partial a_i}{\partial c_k}\frac{\partial c_k}{\partial b_j}$$

$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}}(i,j) = \frac{\partial a_i}{\partial b_j}$$

Jacobian

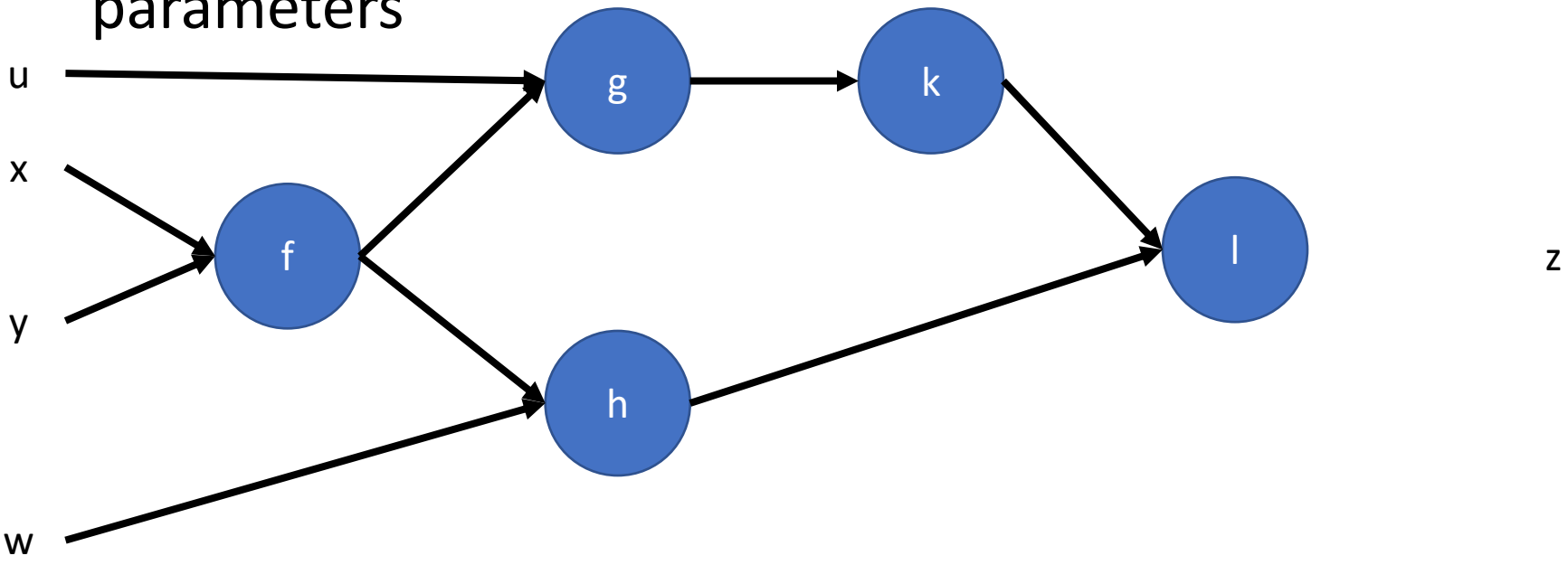
$$\frac{\partial \mathbf{a}}{\partial \mathbf{b}} = \frac{\partial \mathbf{a}}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \mathbf{b}}$$

Loss as a function



Beyond sequences: computation graphs

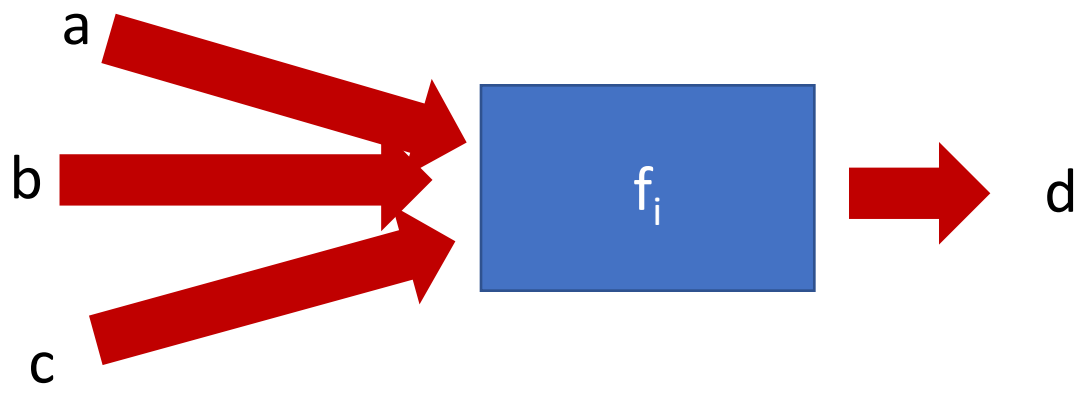
- Arbitrary *graphs* of functions
- No distinction between intermediate outputs and parameters



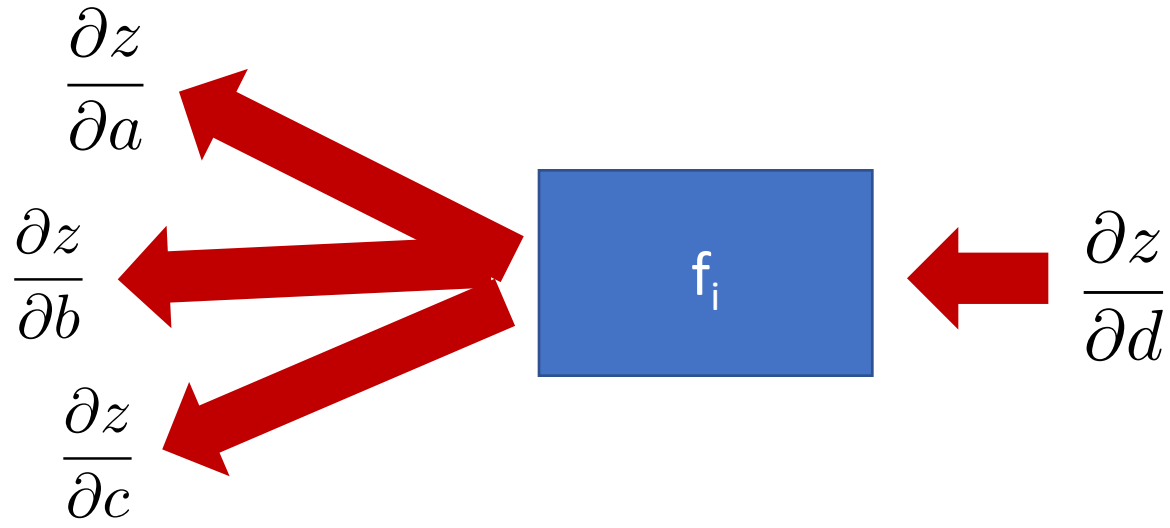
Computation graph - Functions

- Each node implements two functions
  - A “forward”
    - Computes output given input
  - A “backward”
    - Computes derivative of z w.r.t input, given derivative of z w.r.t output

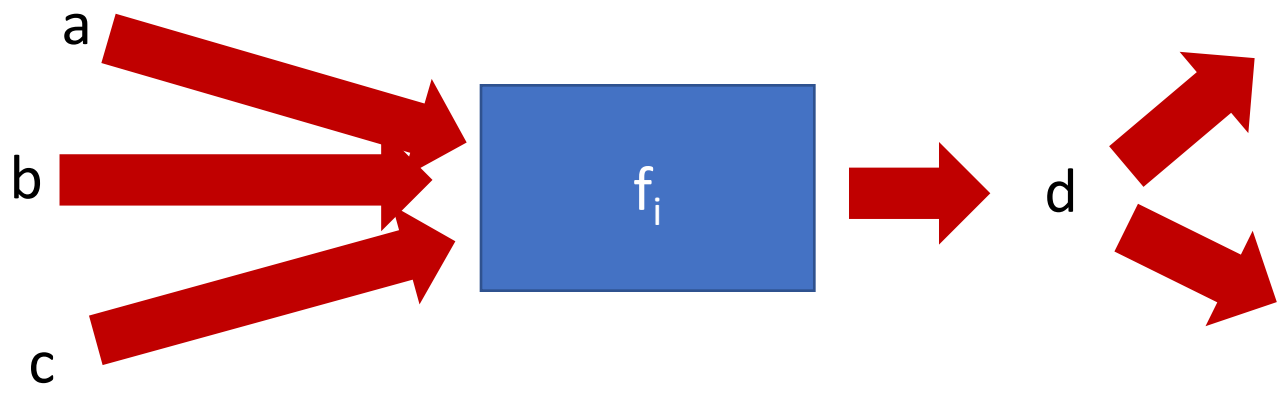
Computation graphs



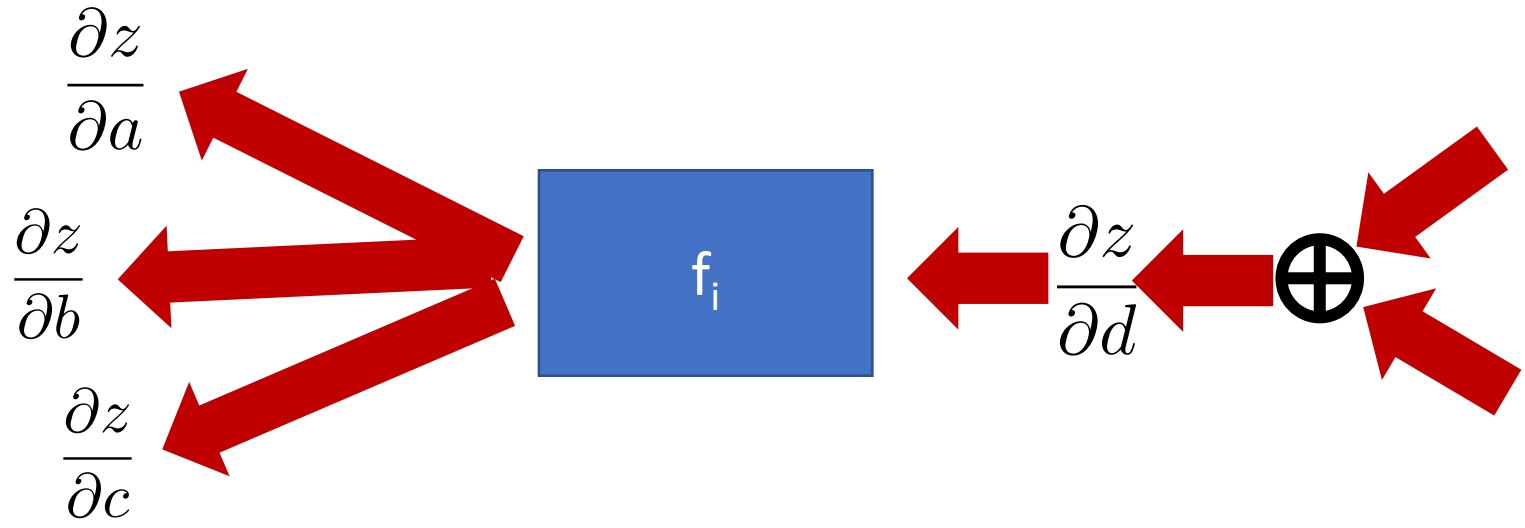
Computation graphs



Computation graphs



Computation graphs



Neural network frameworks