Topics in Social Data Science

Week 4

Artificial Neural Networks 3

Convolutional- and Recurrent Neural Networks

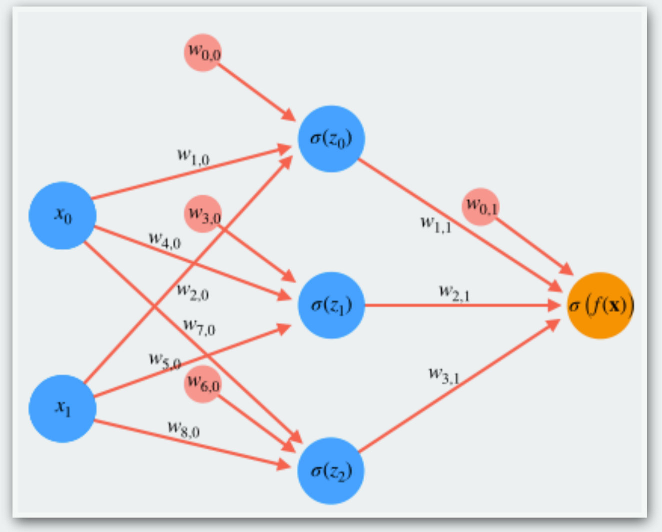
Overview of today + tomorrow

- CS231N lectures
- Nielsen Chapter 6, CS231N course notes, Goodfellow Chapter 10, Karpathy blog post
- My lecture (CNNs and RNNs)
- Exercises in Python

Quick recap…

Find the gradients with **Backpropagation** … this week

(1) The model



Input ⟶ Neural network function ⟶ **Output**

Datapoints, **X**

Parameters: Weights, **W**

Prediction, $\tilde{\mathbf{y}}$

(2) Its performance

Input ⟶ Cost function ⟶ **Output**

Weights, **W**

Many training examples

One number, **the cost**

$$
\begin{aligned}
C(\mathbf{W}) &= \frac{1}{N}\sum_i (\tilde{y}_i - y_i)^2 \\
&= (0.96-1)^2 \\
&+ (0.10-0)^2 \\
&+ (0.04-0)^2 \\
&+ \ldots \\
&+ (0.70-1)^2 \\
&+ (0.02-0)^2 \\
&+ (0.99-1)^2
\end{aligned}
$$

(3) The cost function gradient in **W**

$$
-\nabla C(\mathbf{W}) = \begin{bmatrix} -0.23 \\ 1.32 \\ 0.20 \\ 0.38 \\ -1.23 \\ 0.01 \\ 1.20 \\ -2.12 \\ 0.73 \\ 2.17 \\ 0.54 \\ -0.23 \\ -0.93 \\ 0.45 \end{bmatrix}
$$

(4) Updating **W**

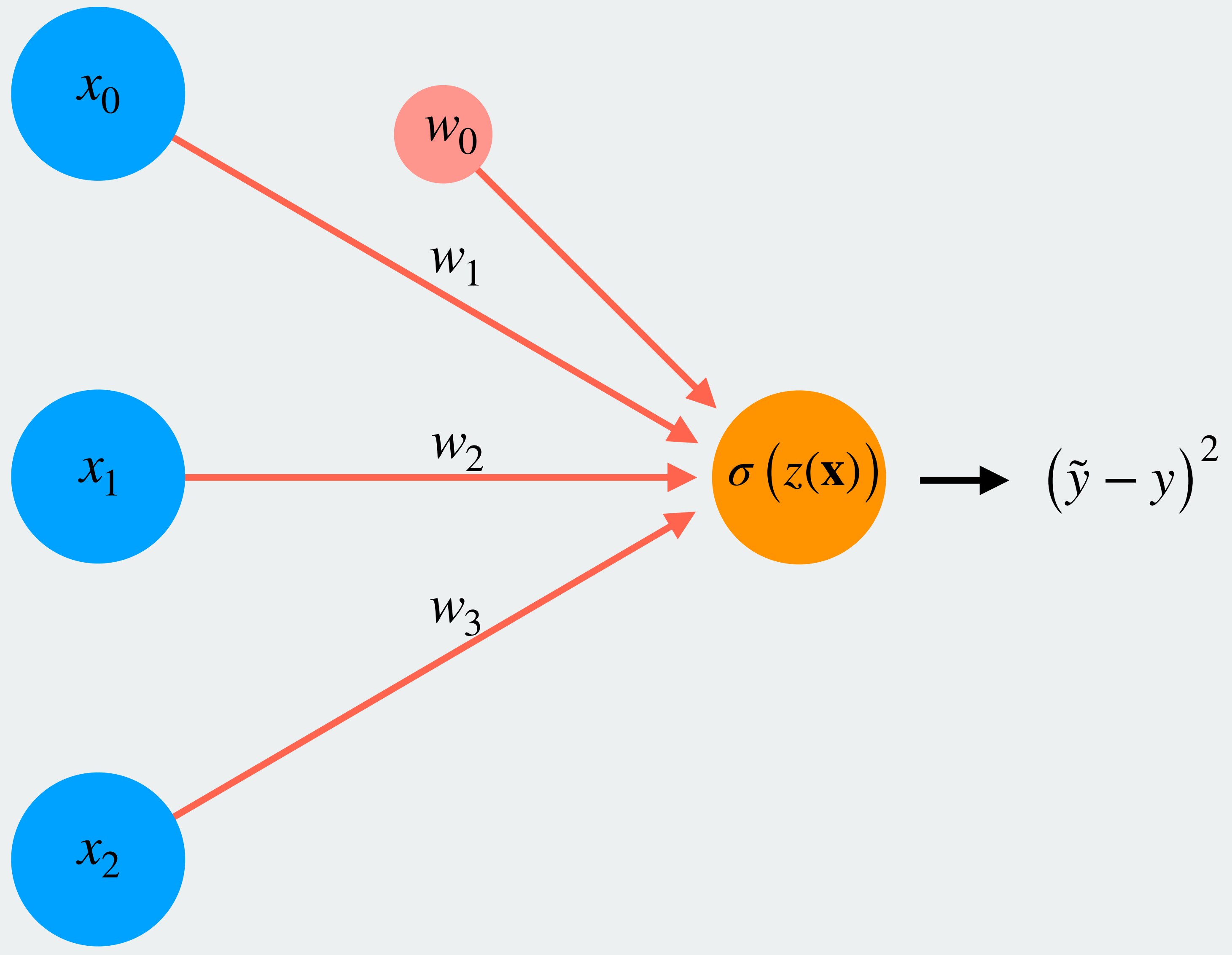
$$
\mathbf{W} = \mathbf{W}^{\text{old}} + r\left(-\nabla C(\mathbf{W}^{\text{old}})\right)
$$

(5) Repeat 3 and 4

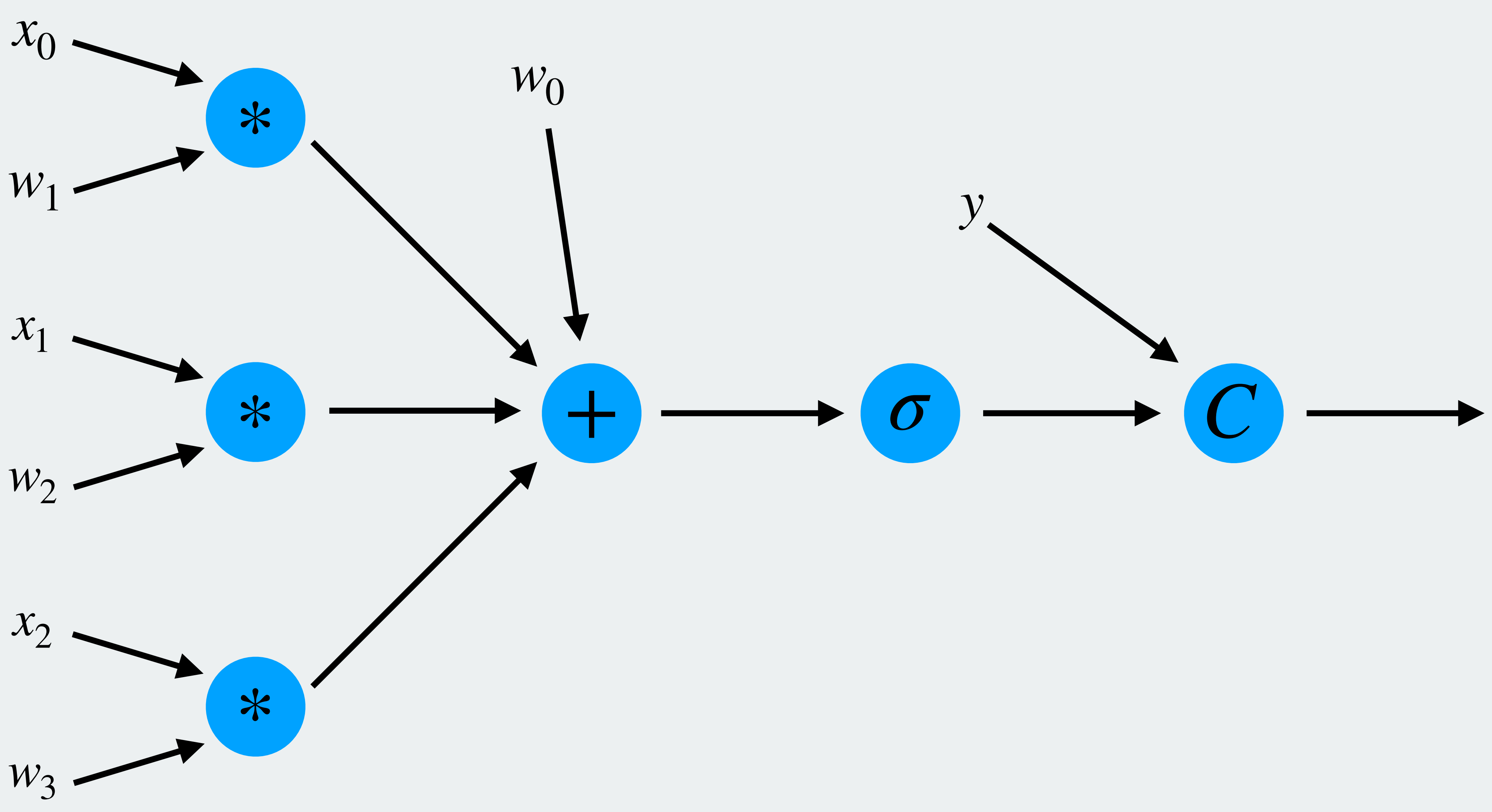
r is usually called the *learning rate*

Quick recap…

Neural network

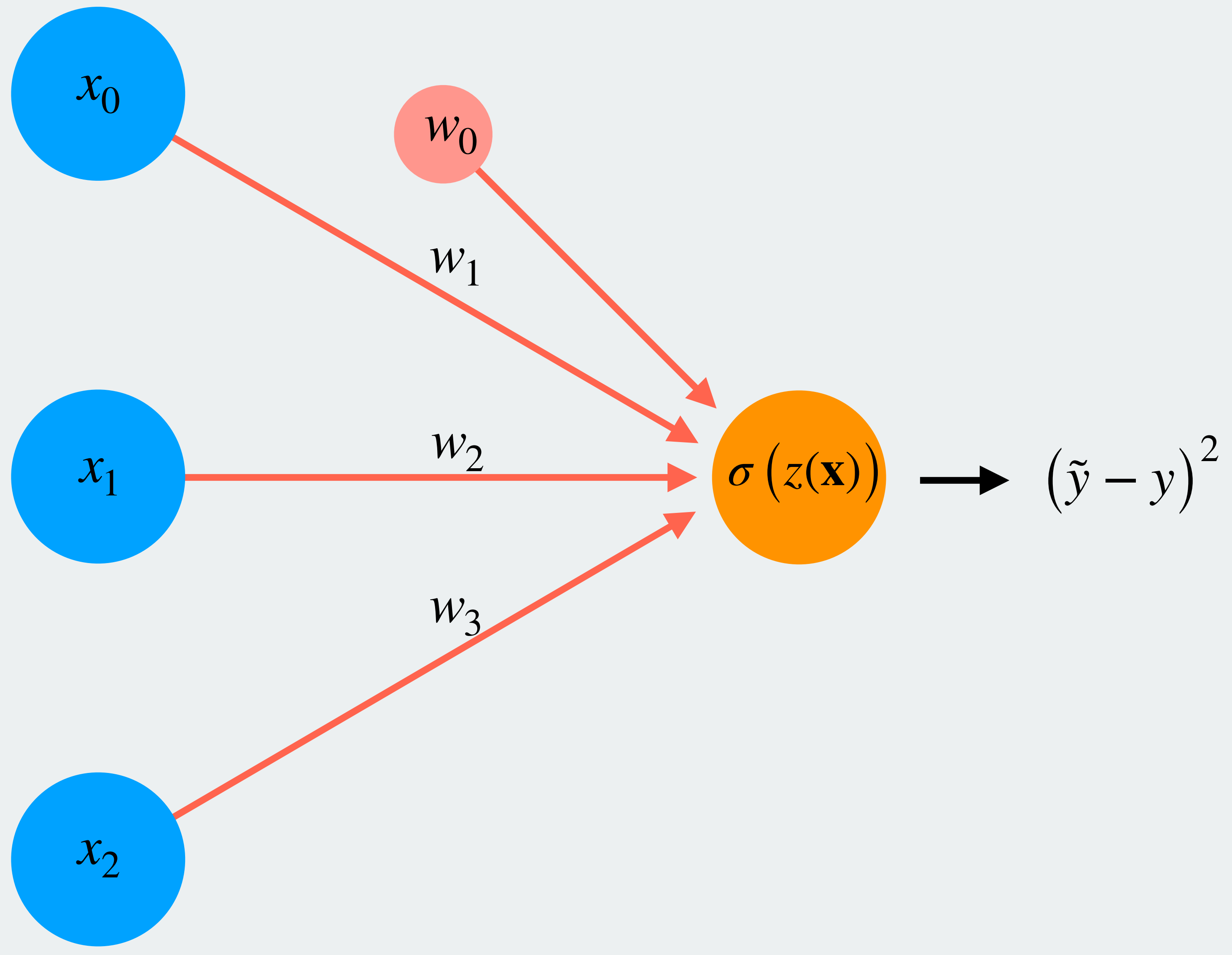


Computational graph

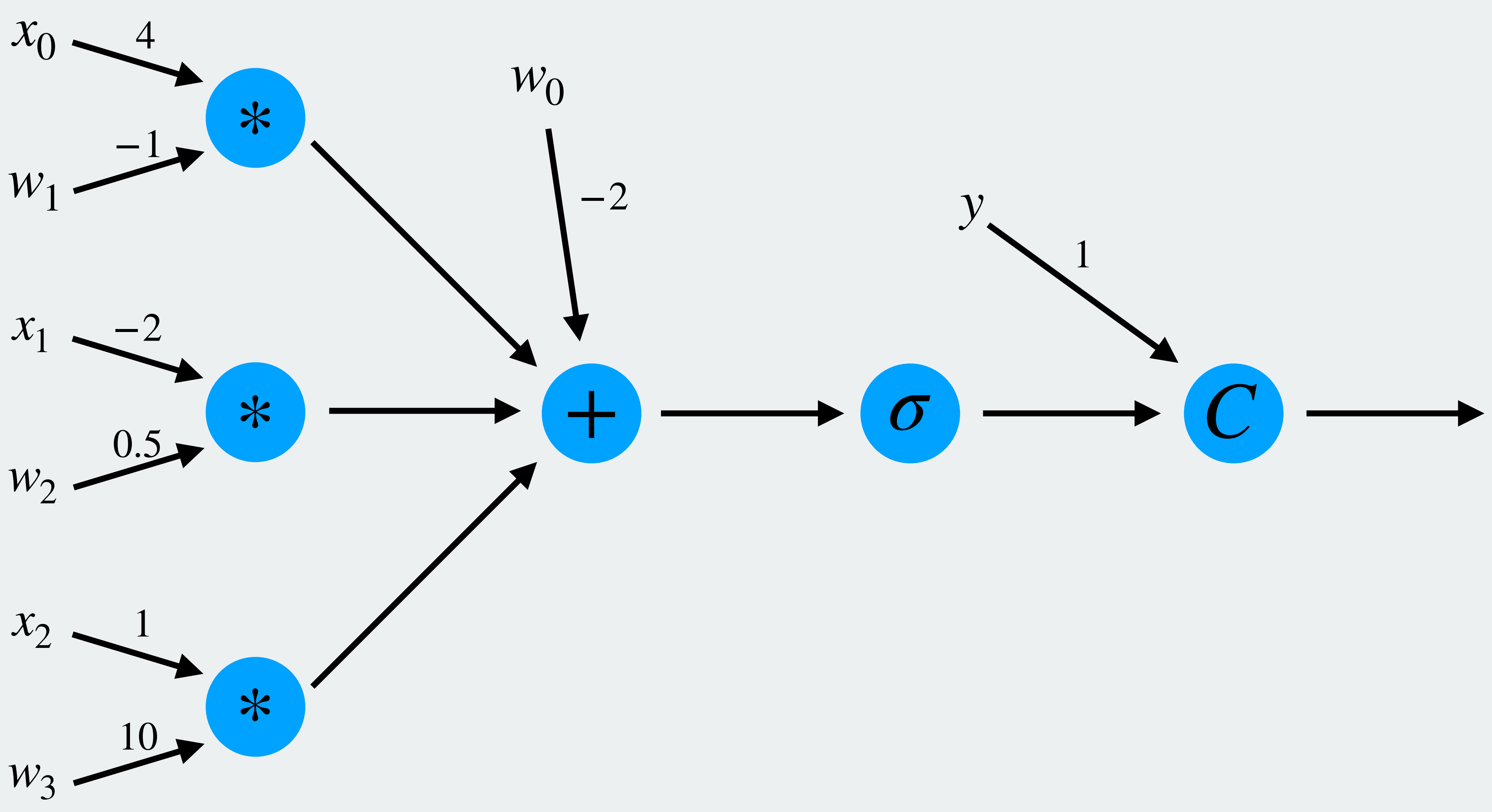


Quick recap…

Neural network

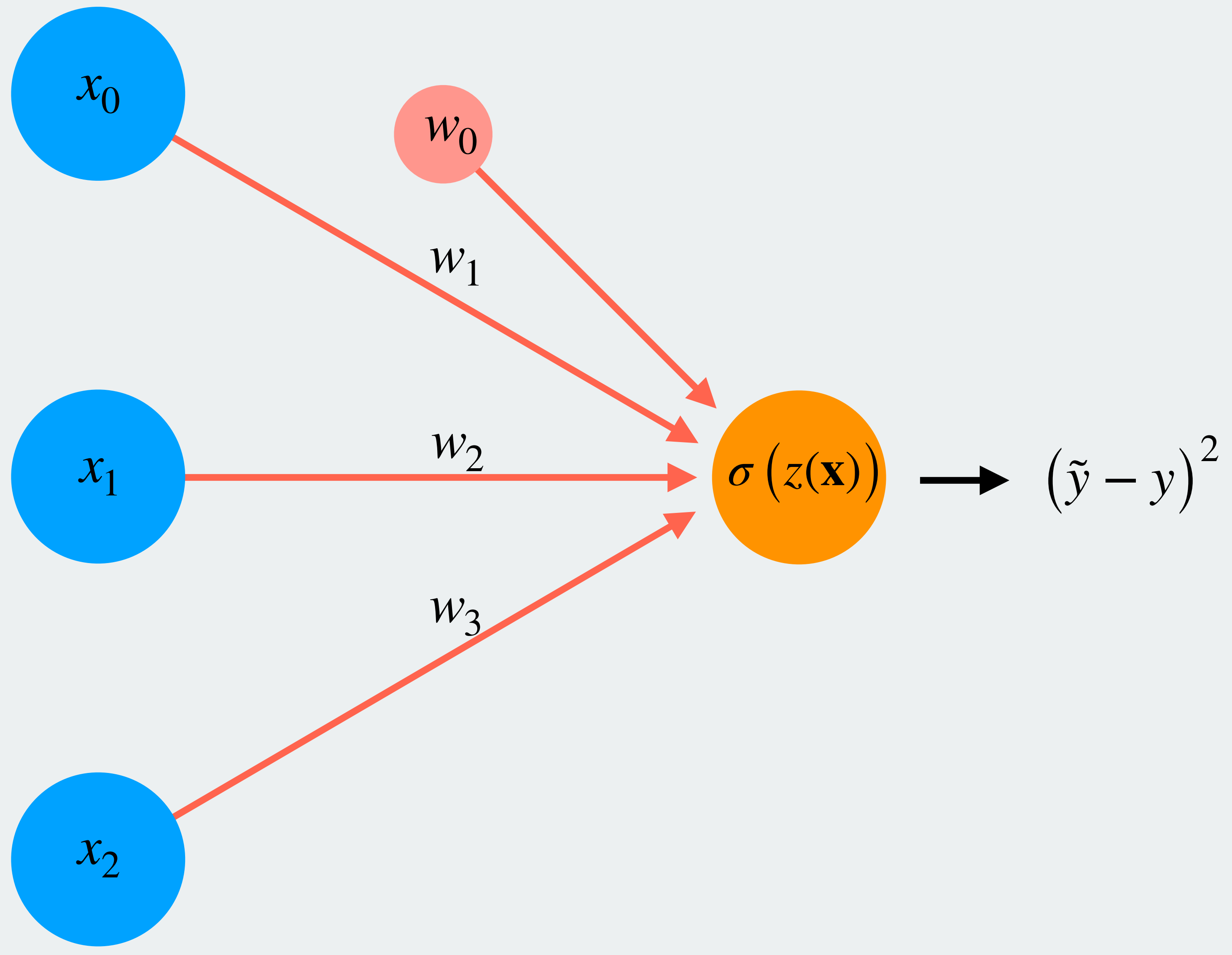


Computational graph



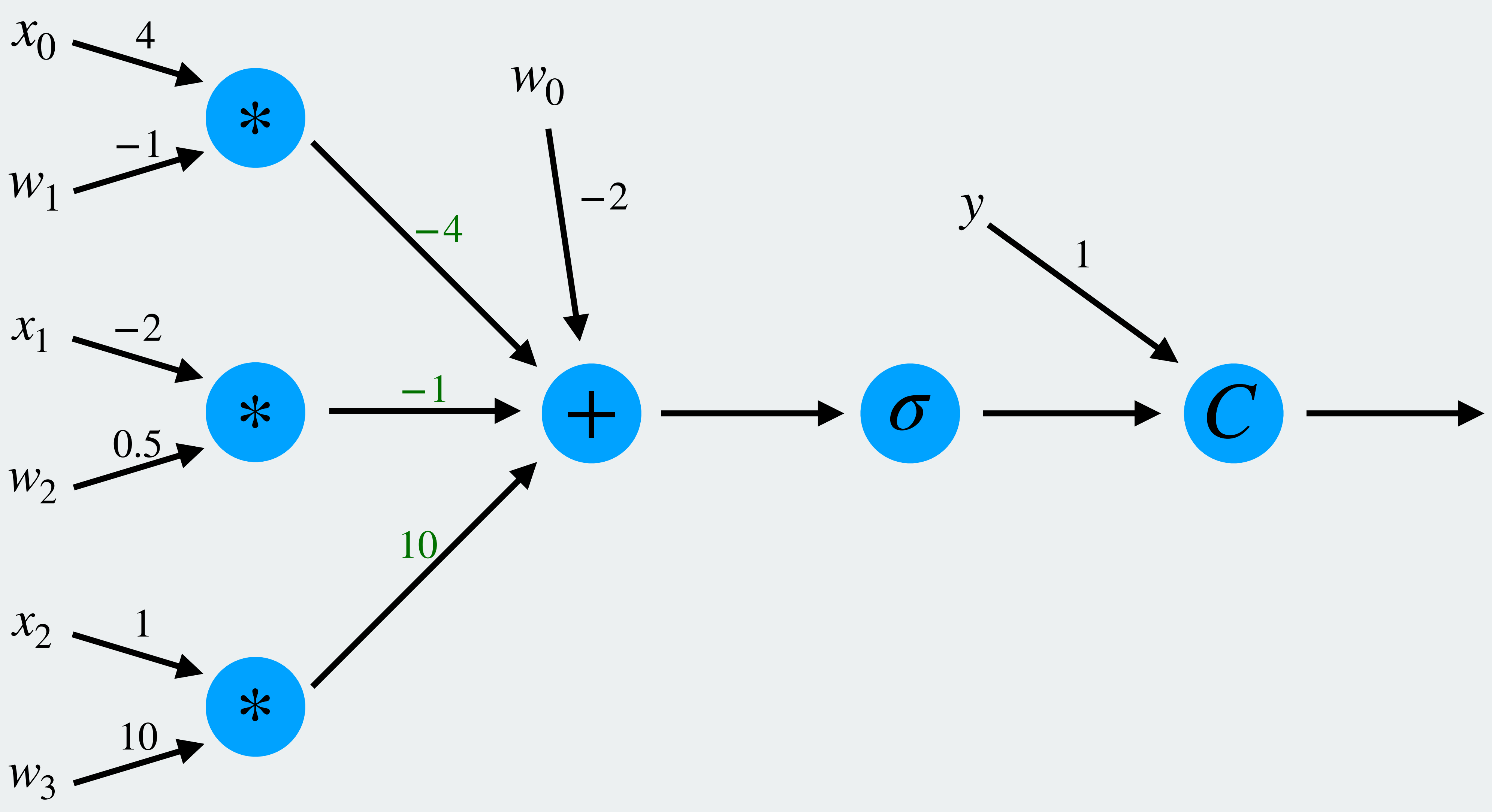
Quick recap…

Neural network



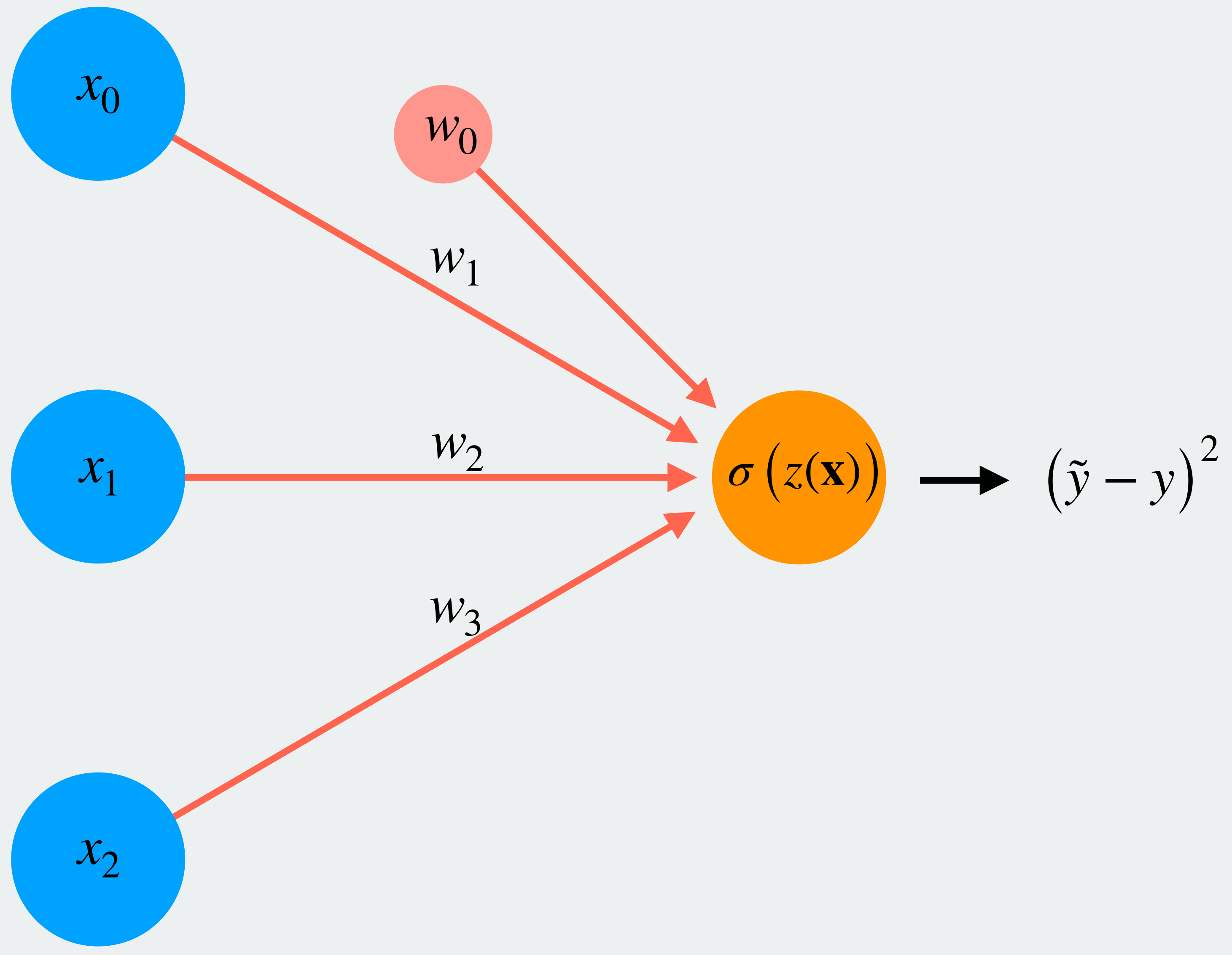
Computational graph

Forward pass

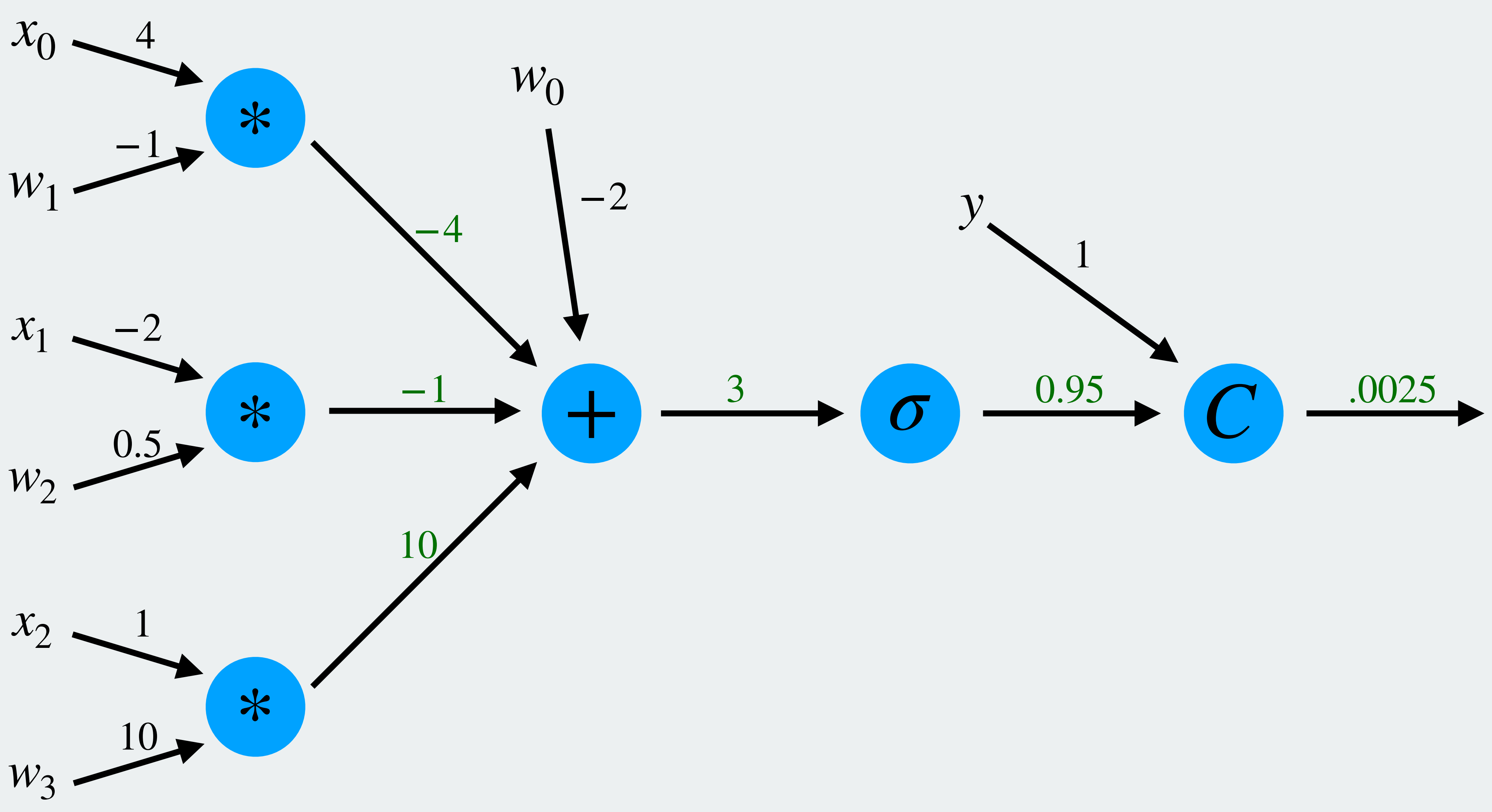


Quick recap…

Neural network → **Computational graph**

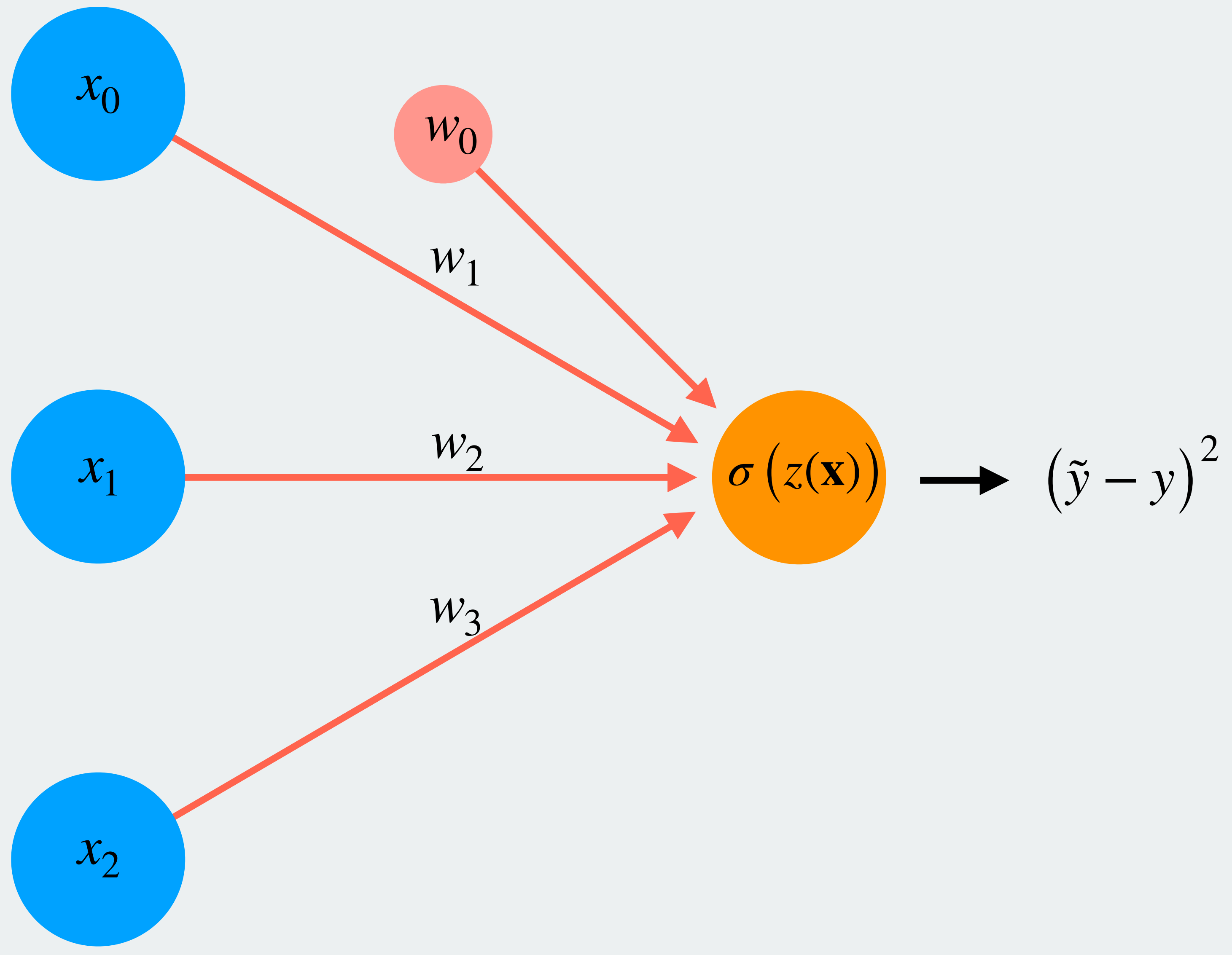


Forward pass



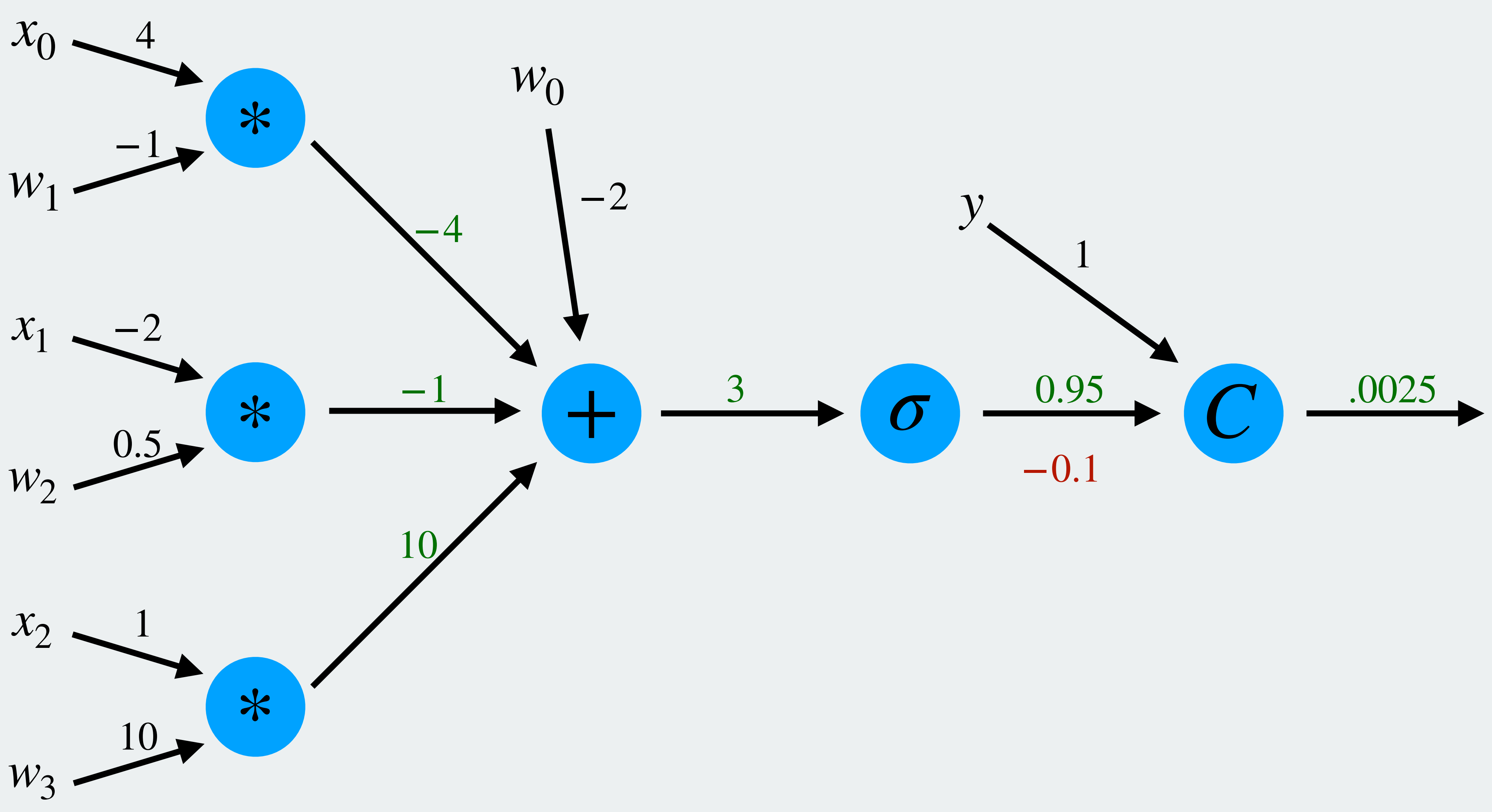
Quick recap…

Neural network

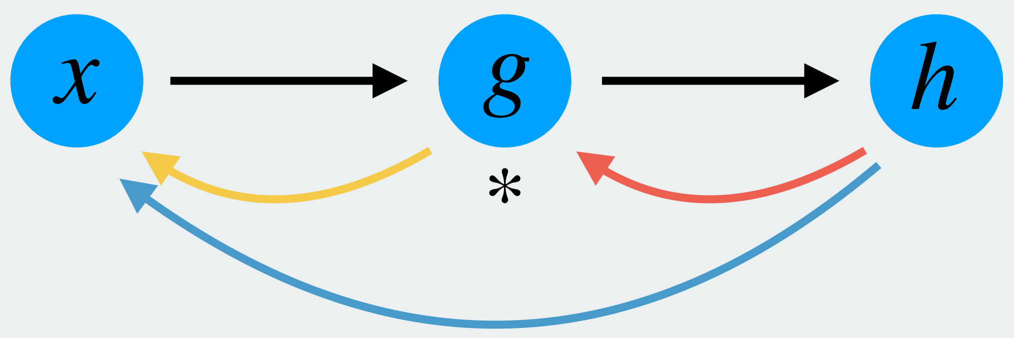


Computational graph

Backward pass



$h(g(x))$

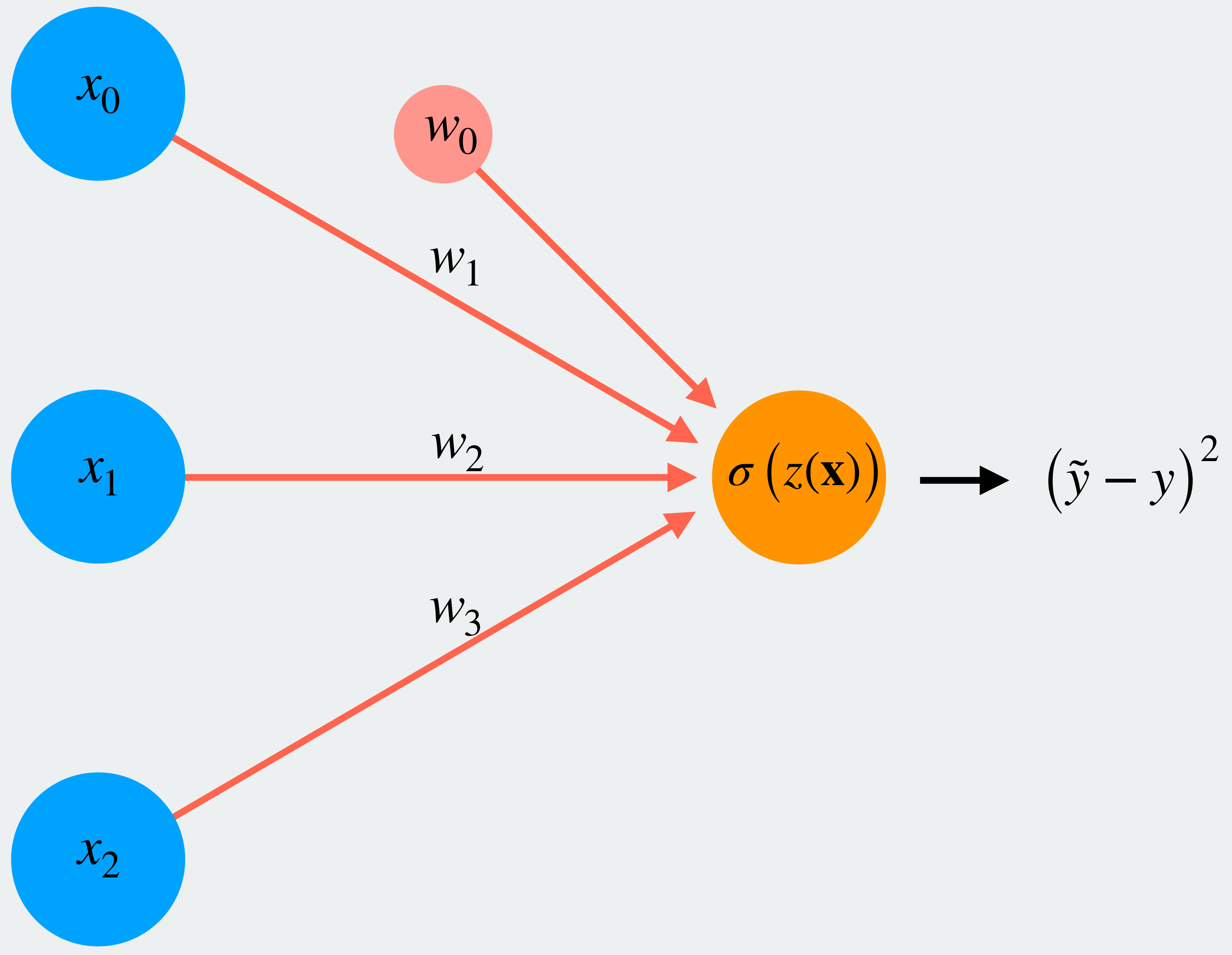


Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

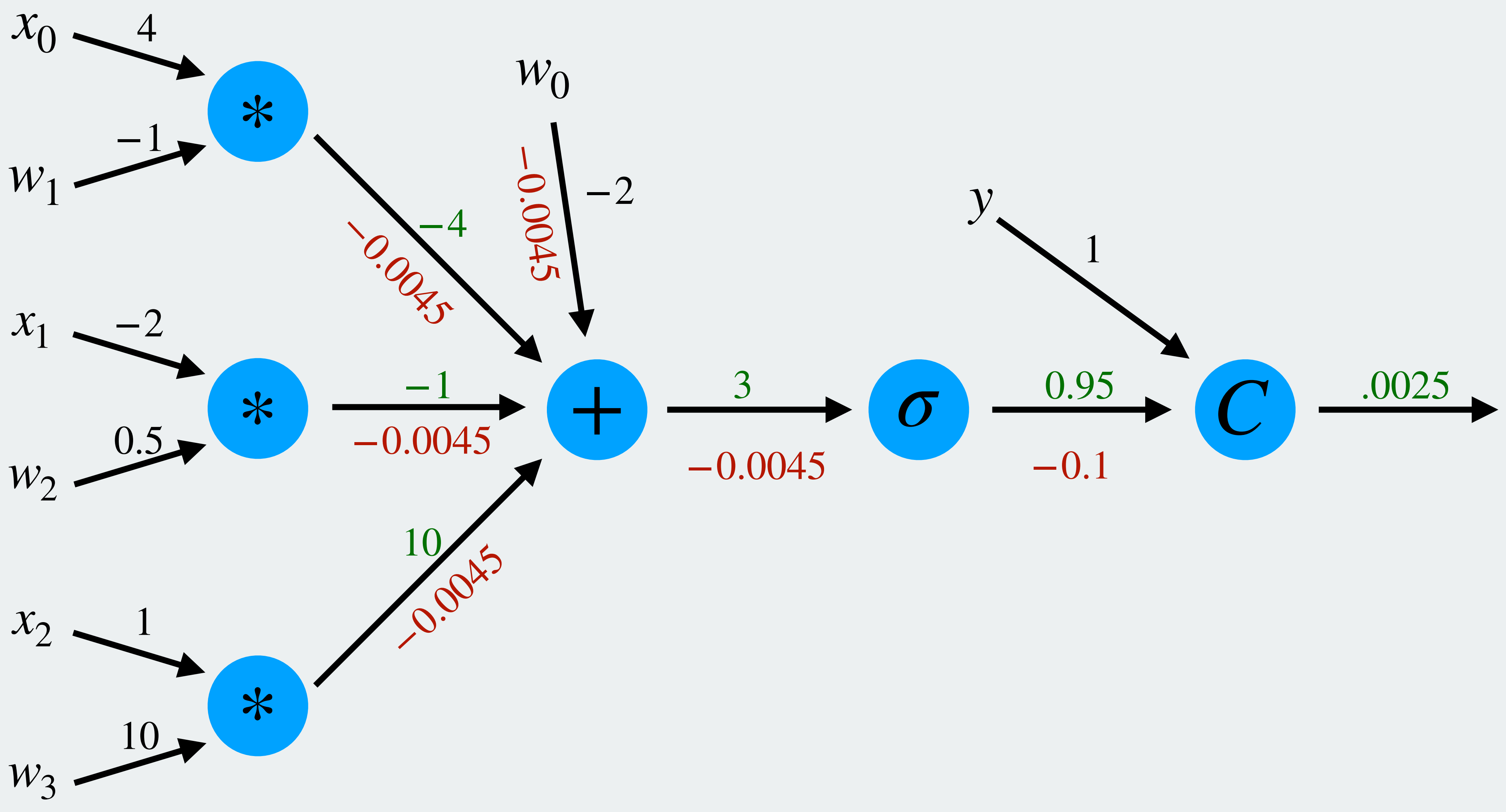
Quick recap…

Neural network

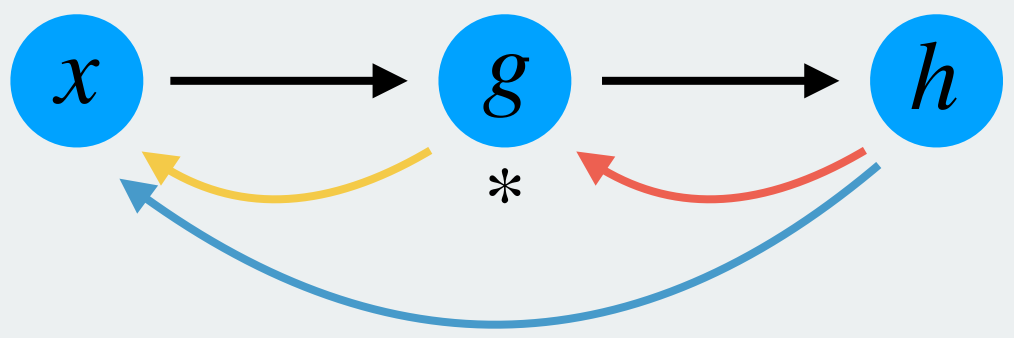


Computational graph

Backward pass



$h\left(g\left(x\right)\right)$

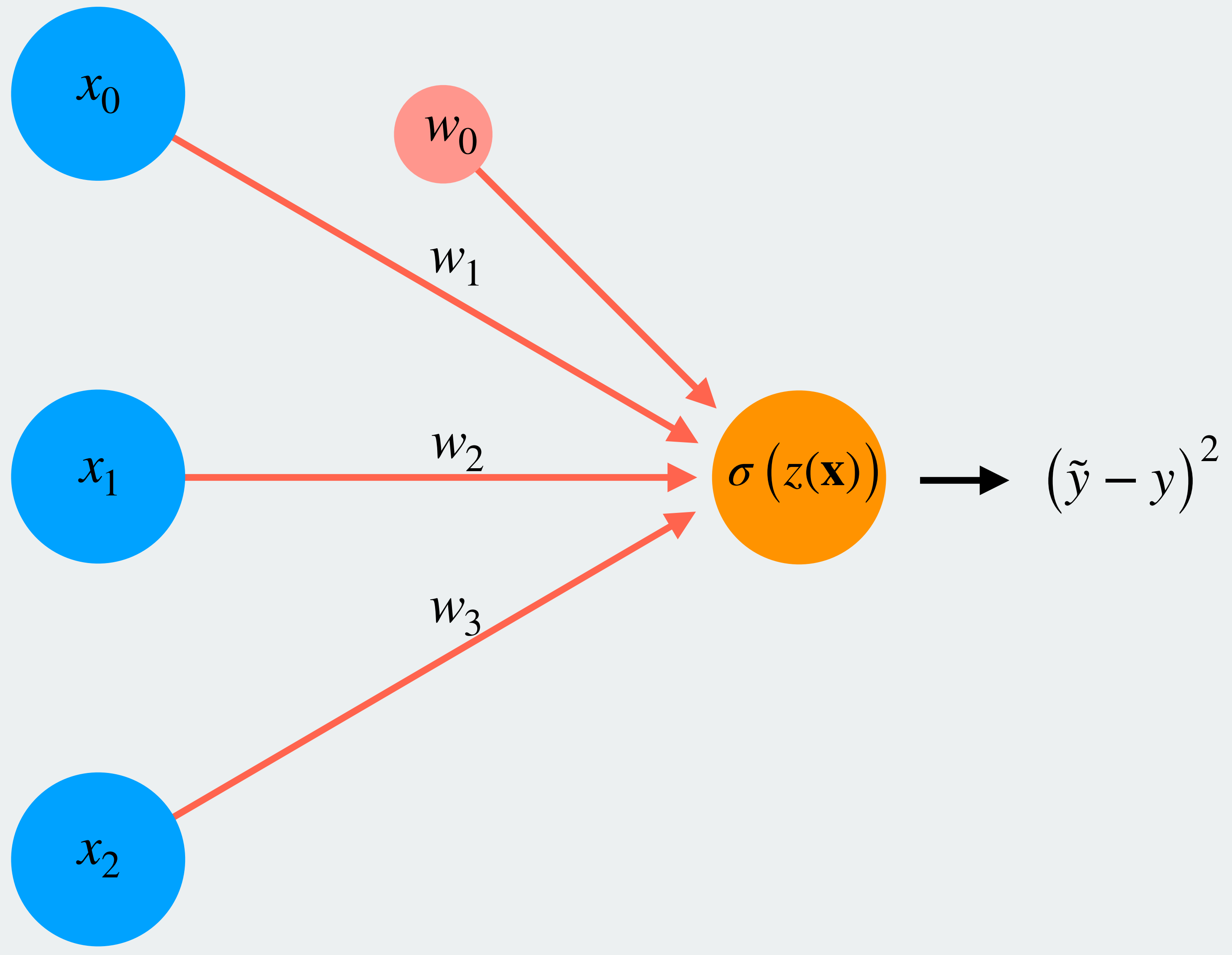


Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

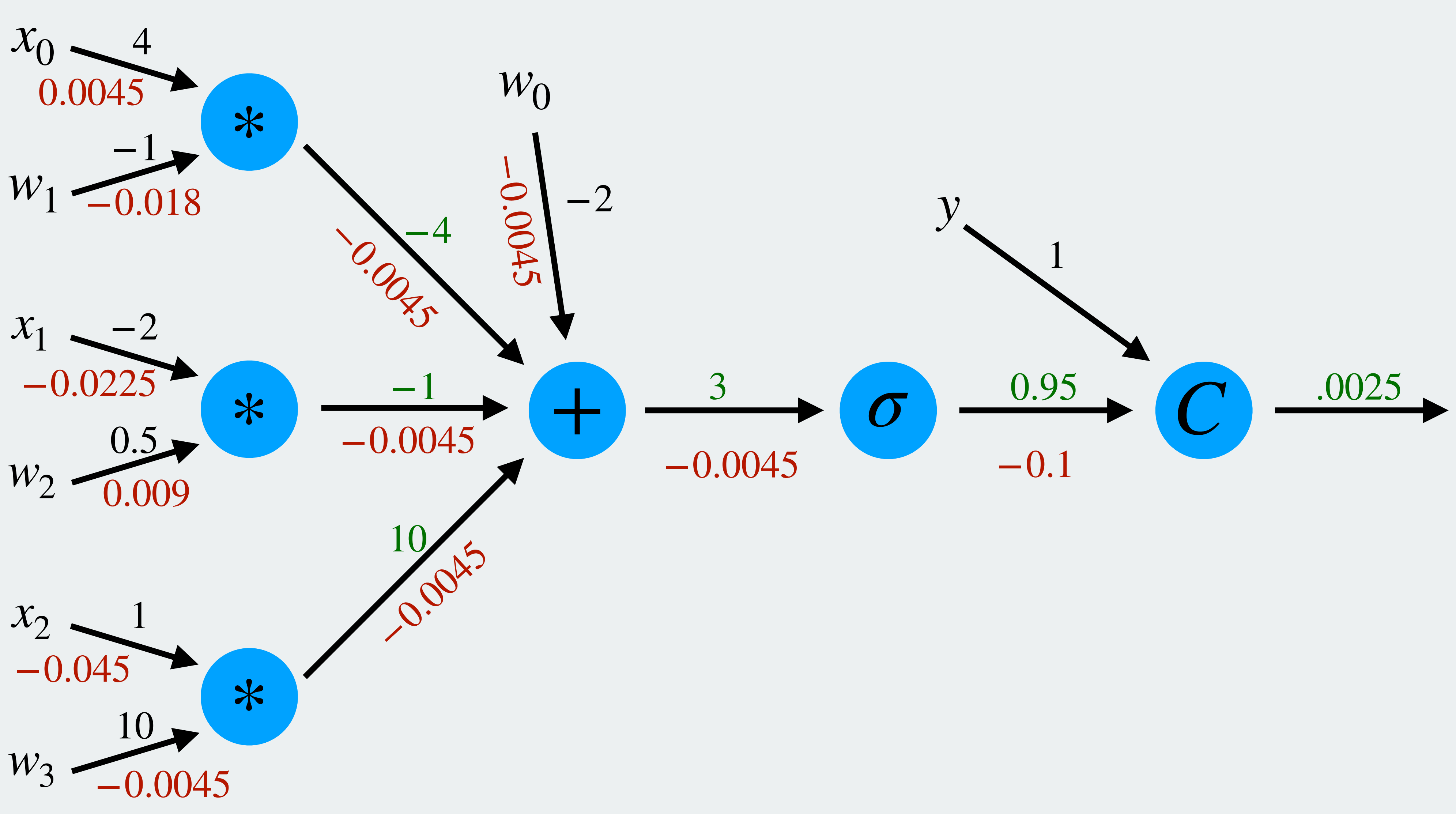
Quick recap…

Neural network

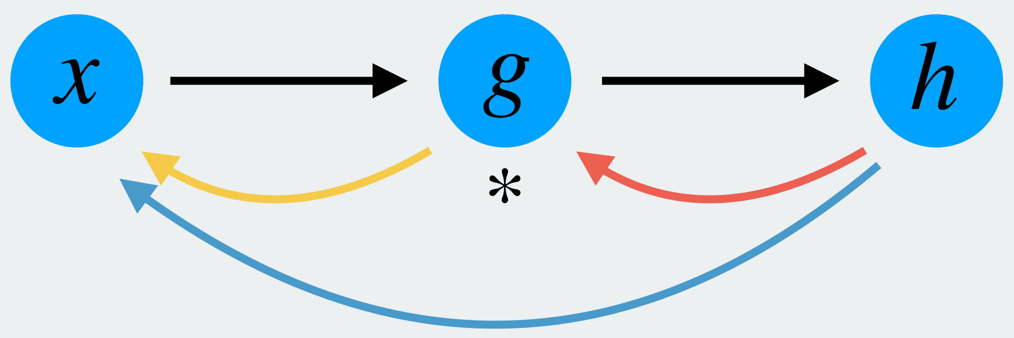


Computational graph

Backward pass



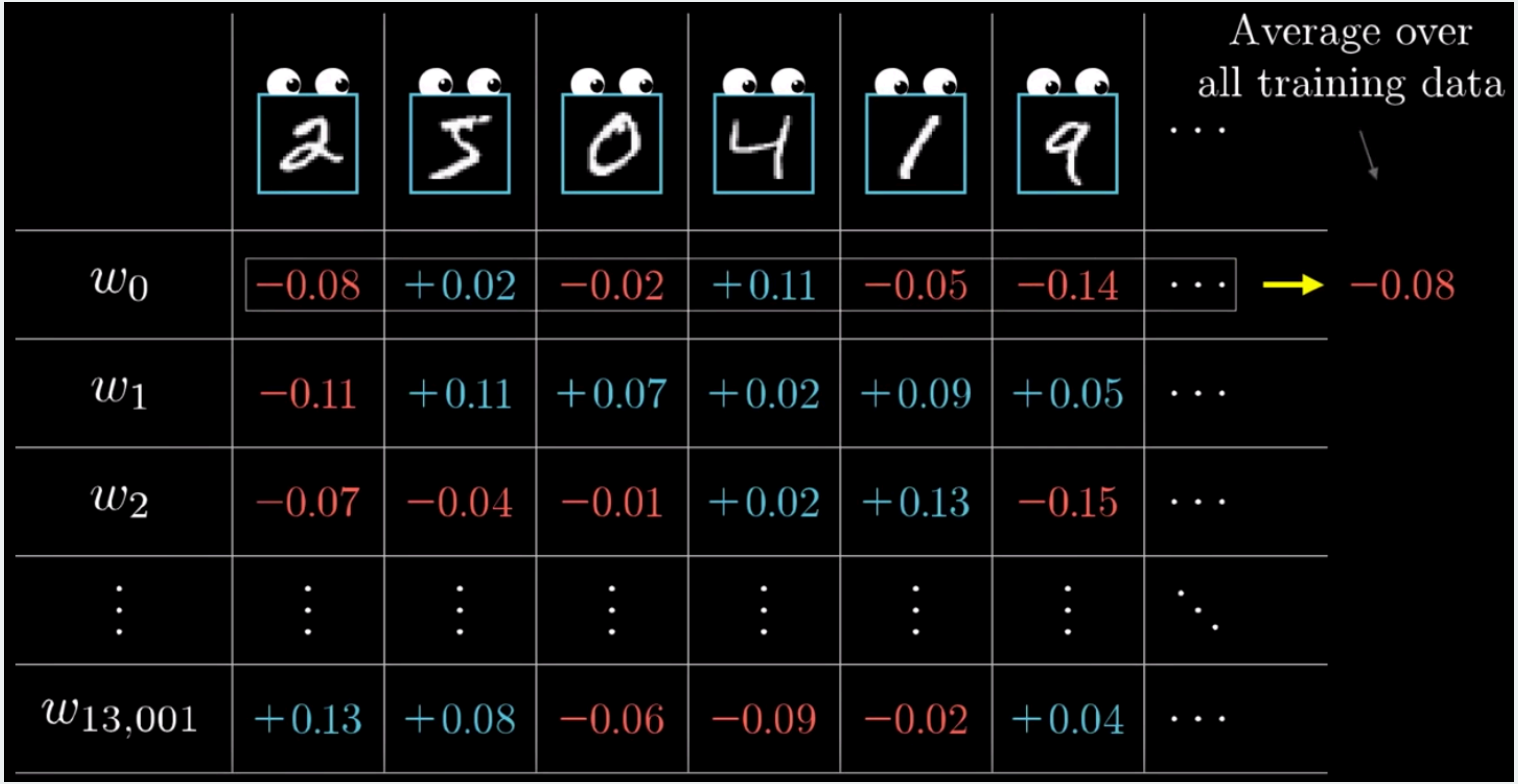
$h(g(x))$



Chain rule says:

$$\frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{dx}$$

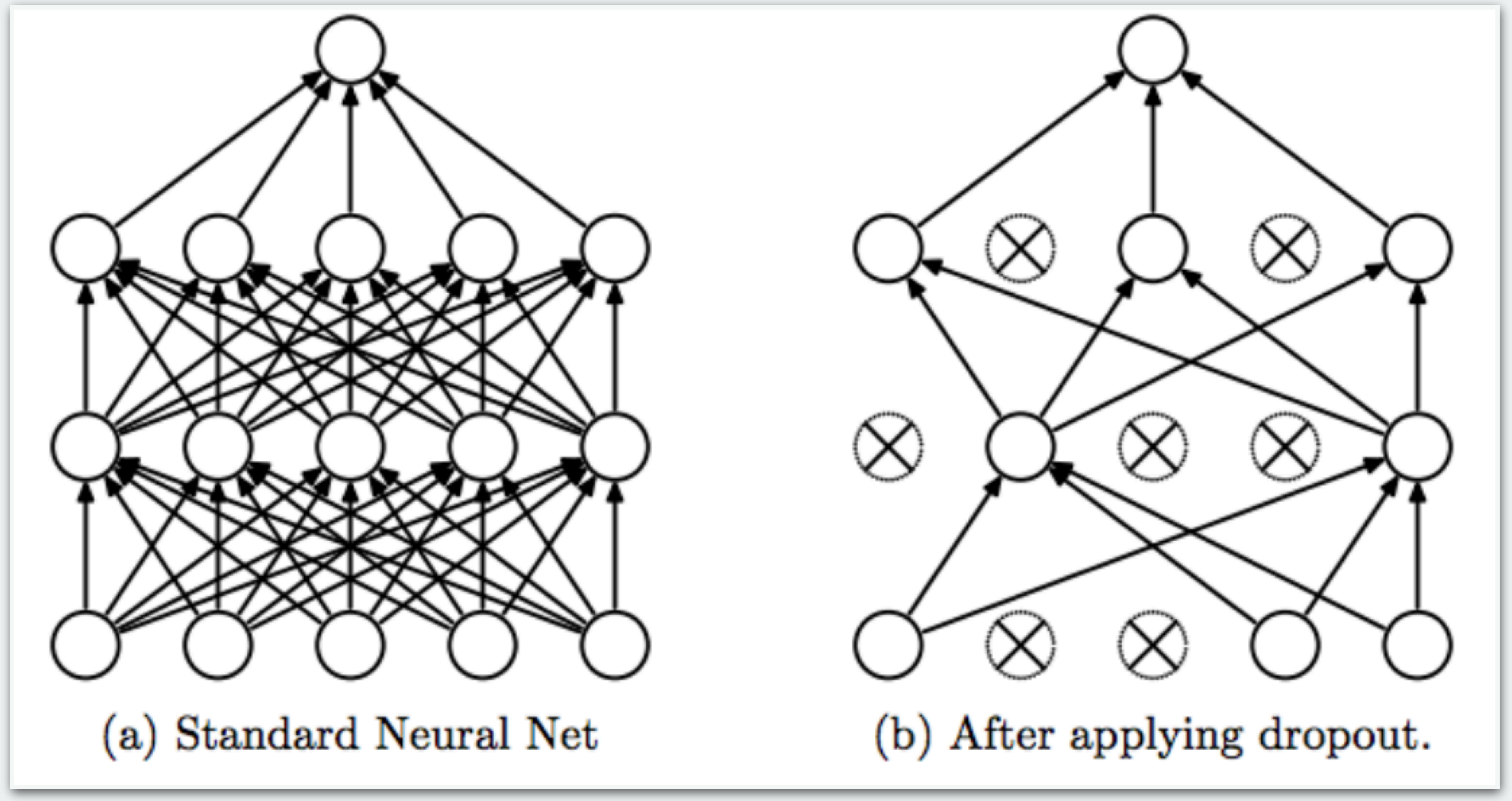
Quick recap…


Average over
all training data
w_0 −0.08 +0.02 −0.02 +0.11 −0.05 −0.14 ⋯ → −0.08
w_1 −0.11 +0.11 +0.07 +0.02 +0.09 +0.05 ⋯
w_2 −0.07 −0.04 −0.01 +0.02 +0.13 −0.15 ⋯
$w_{13,001}$ +0.13 +0.08 −0.06 −0.09 −0.02 +0.04 ⋯

Quick recap…

Dropout:

“In each SGD step, randomly ignore a fraction *p* of neurons”



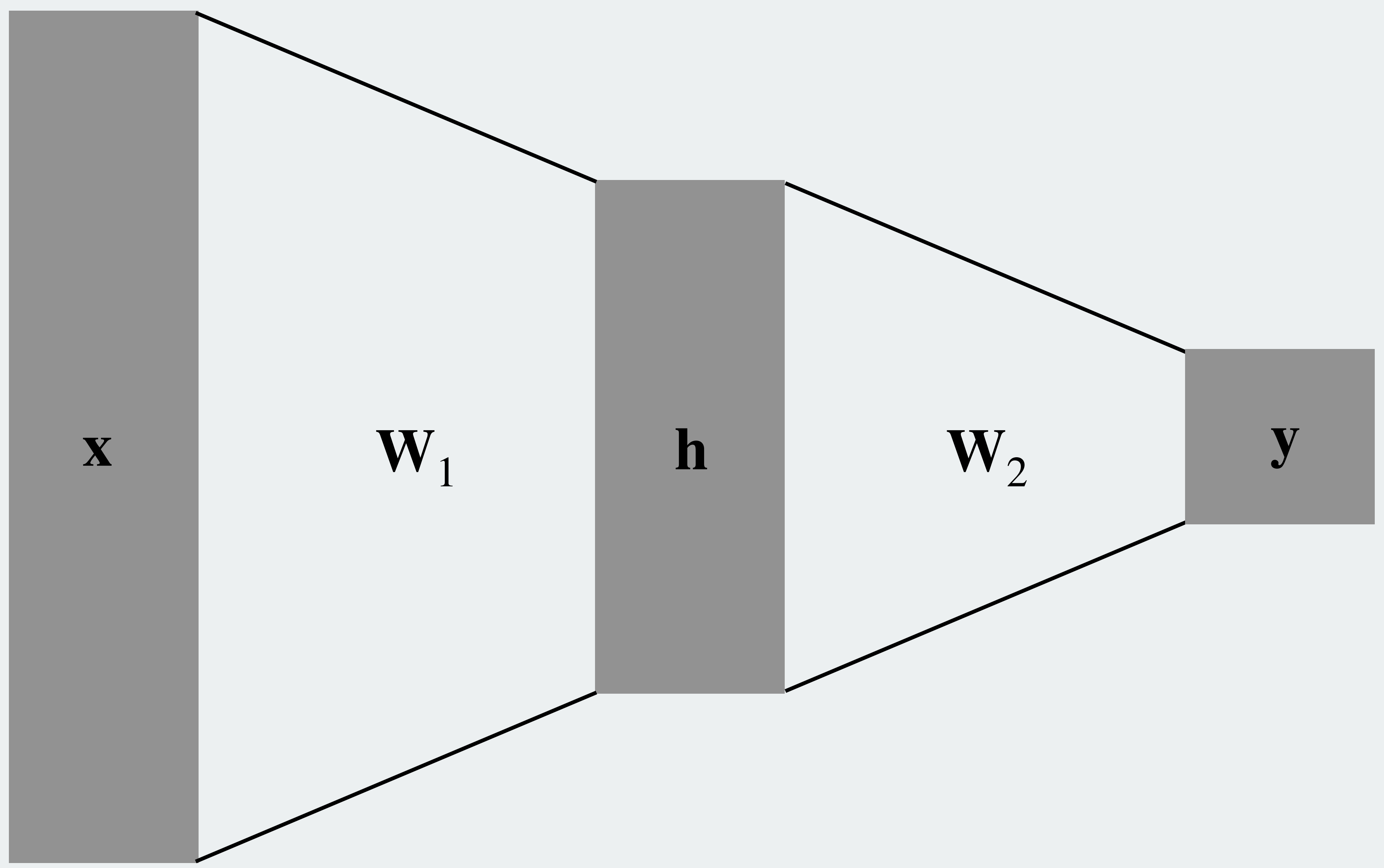
Srivastava, Nitish, et al. ”Dropout: a simple way to prevent neural networks from overfitting”, JMLR 2014

- Can select *p* in wide range. Typical is 0.2 – 0.8, dependent on size of ANN
- Can apply only in specific layers. It is typical to only do dropout in a designated “dropout layer” somewhere close to output.

Convolutional Neural Networks

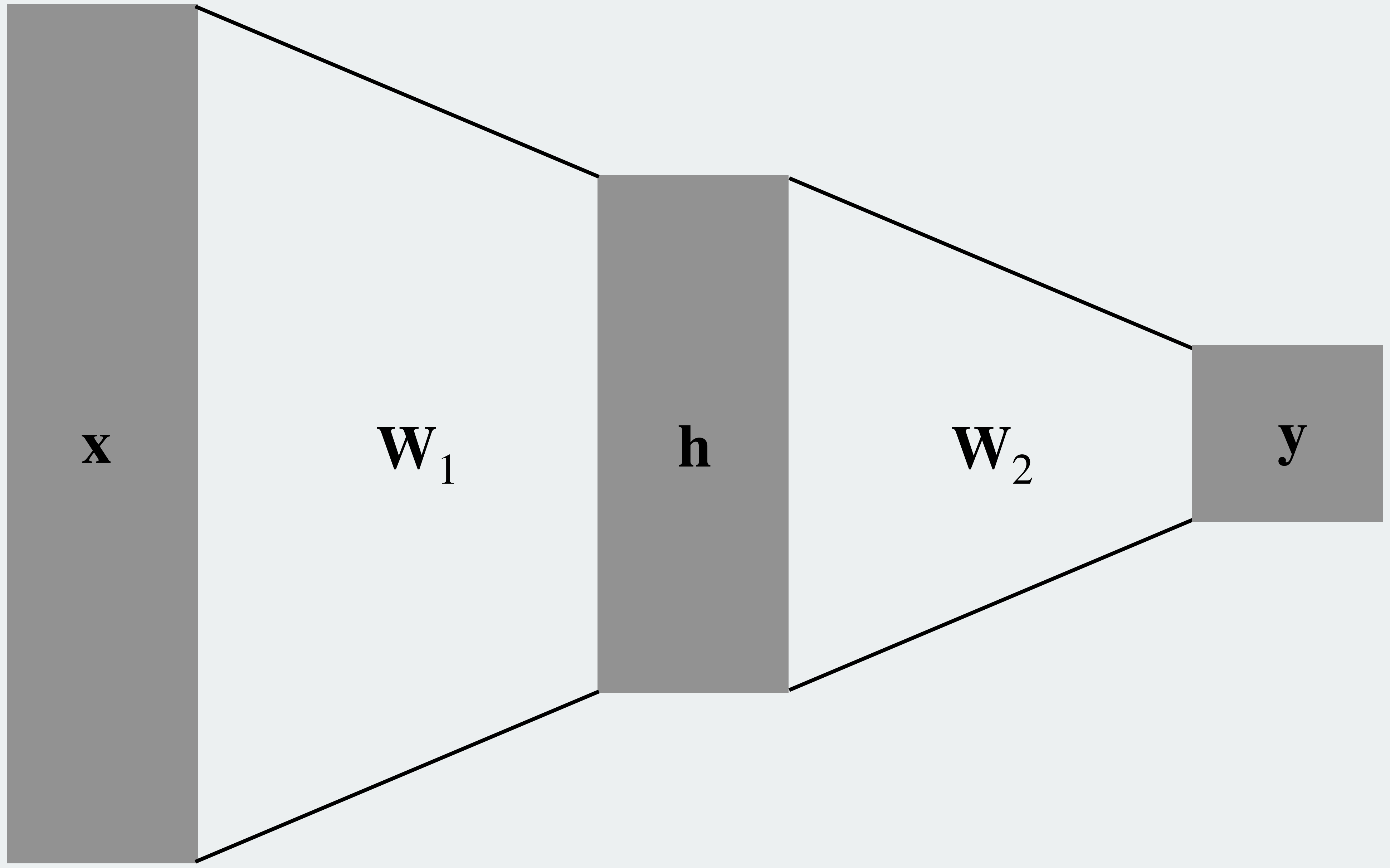
THE neural network architecture to use for image data

The problem with vanilla neural networks

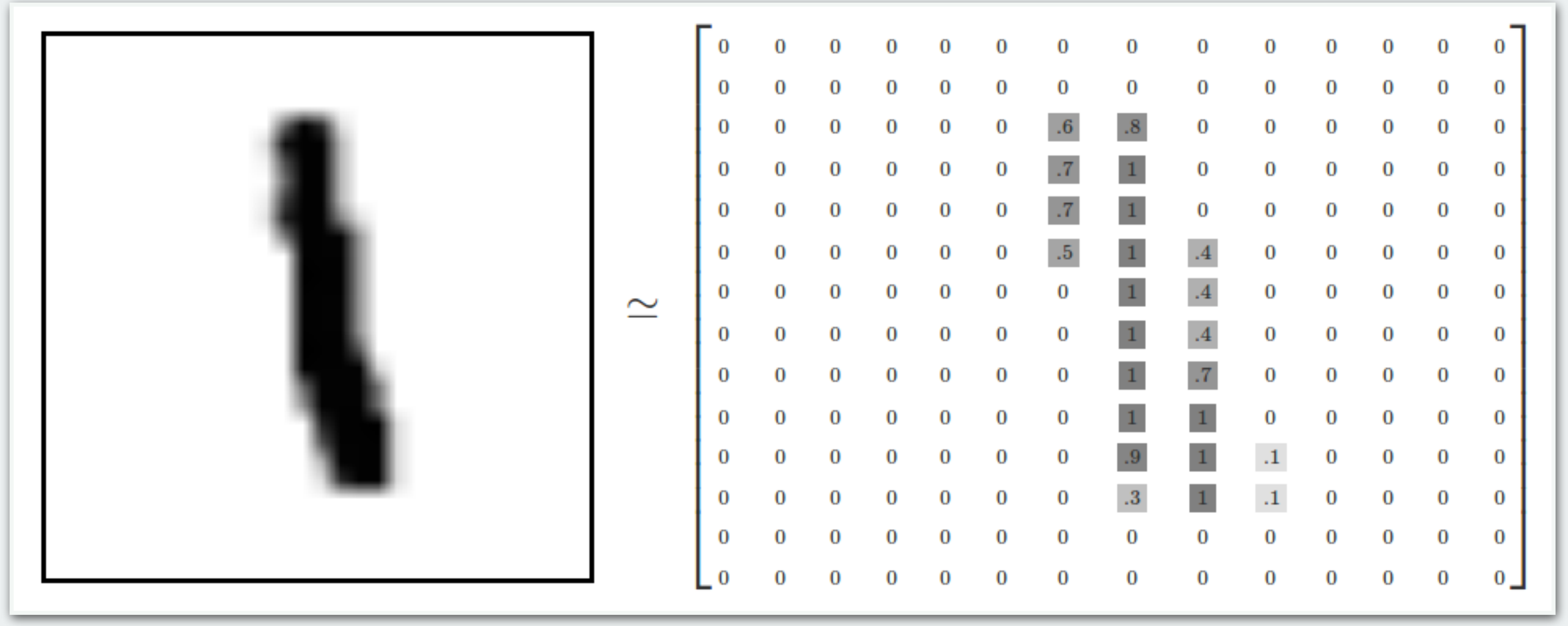


- Single operation on whole input
- Each neuron reacts to specific inputs

The problem with vanilla neural networks

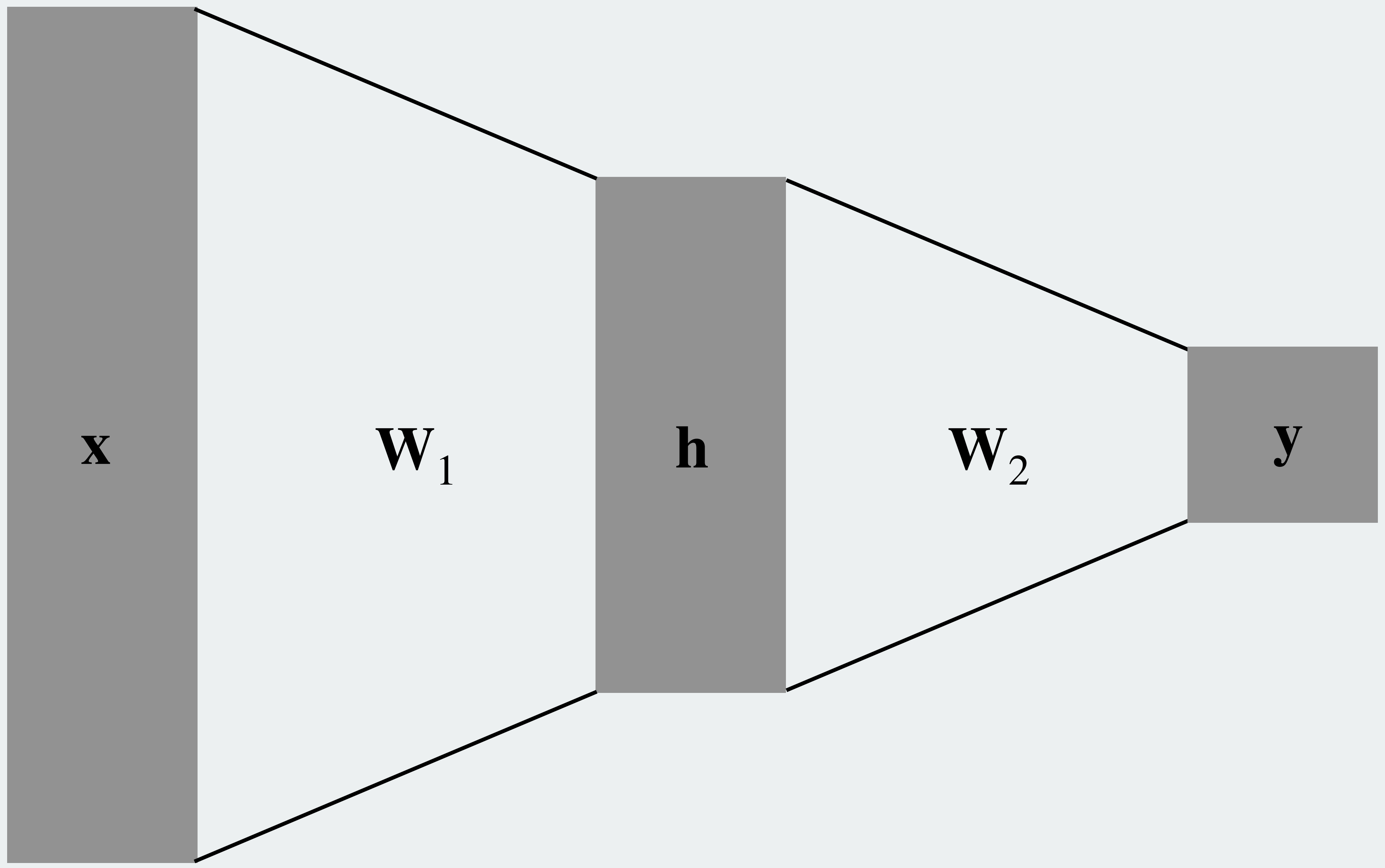


- Single operation on whole input
- Each neuron reacts to specific inputs
- Bad for images: objects move around
- No attention to spatial adjacency



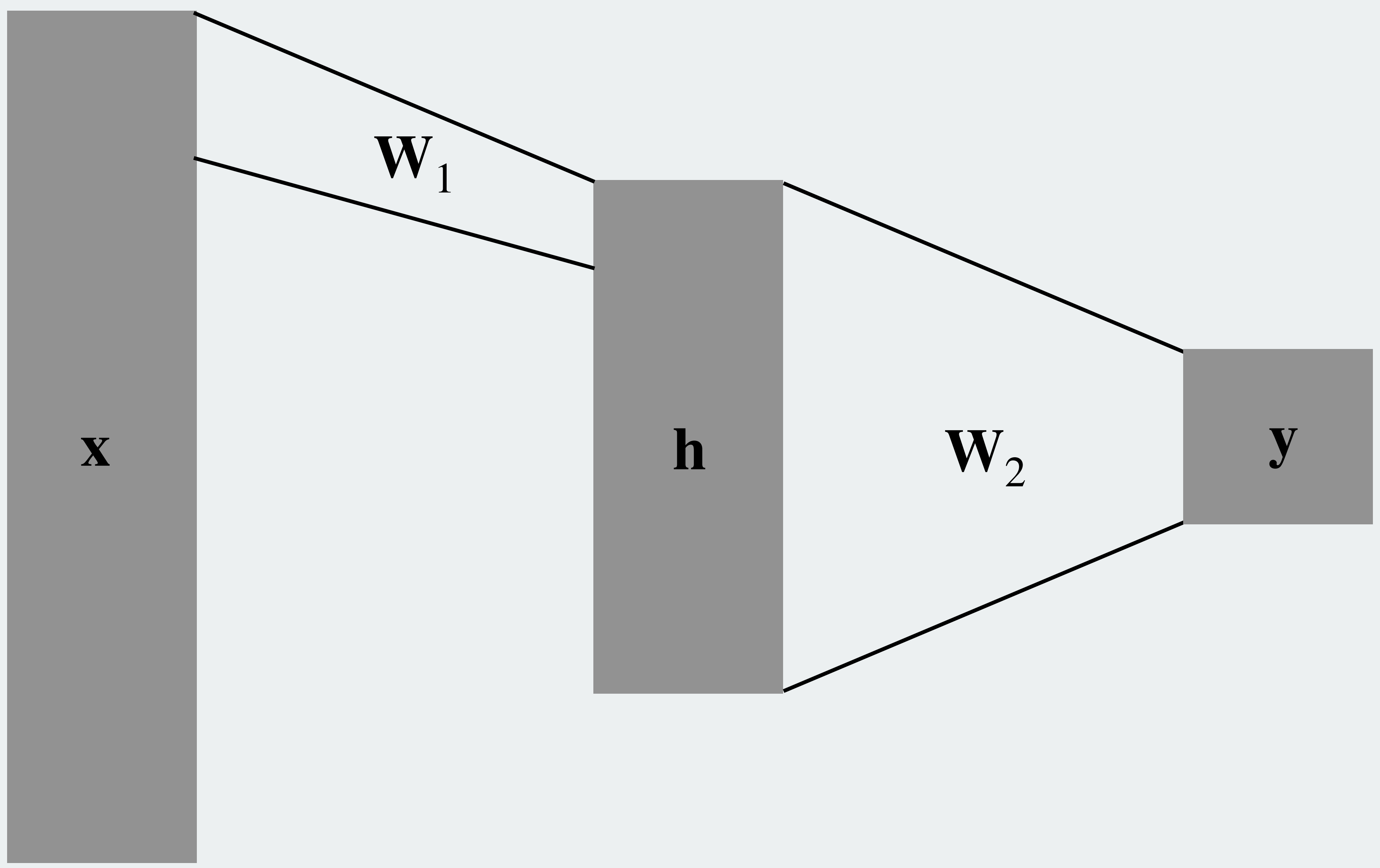
The problem with vanilla neural networks

> Solution: “slide” the weight matrix over the input



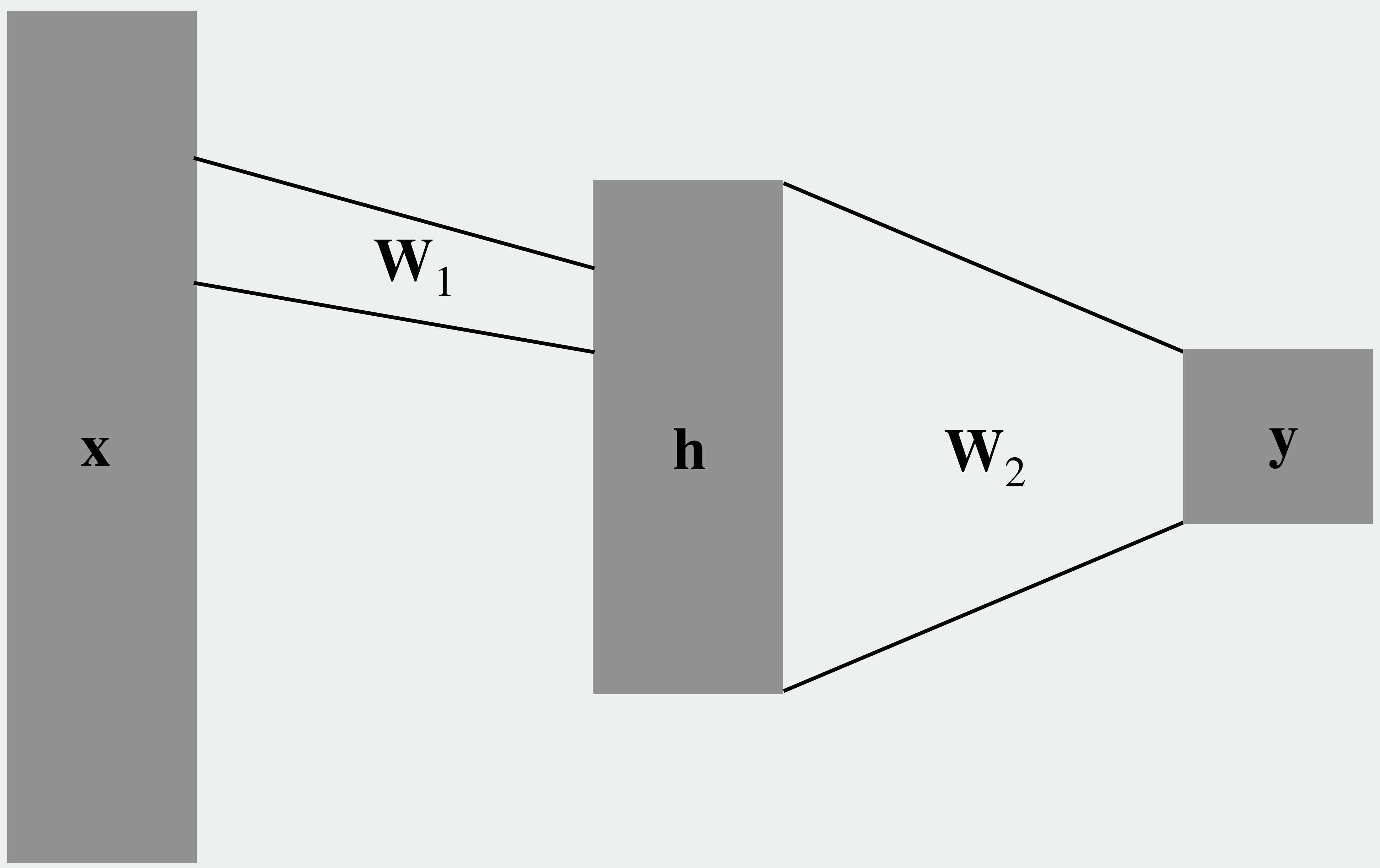
The problem with vanilla neural networks

> Solution: “slide” the weight matrix over the input



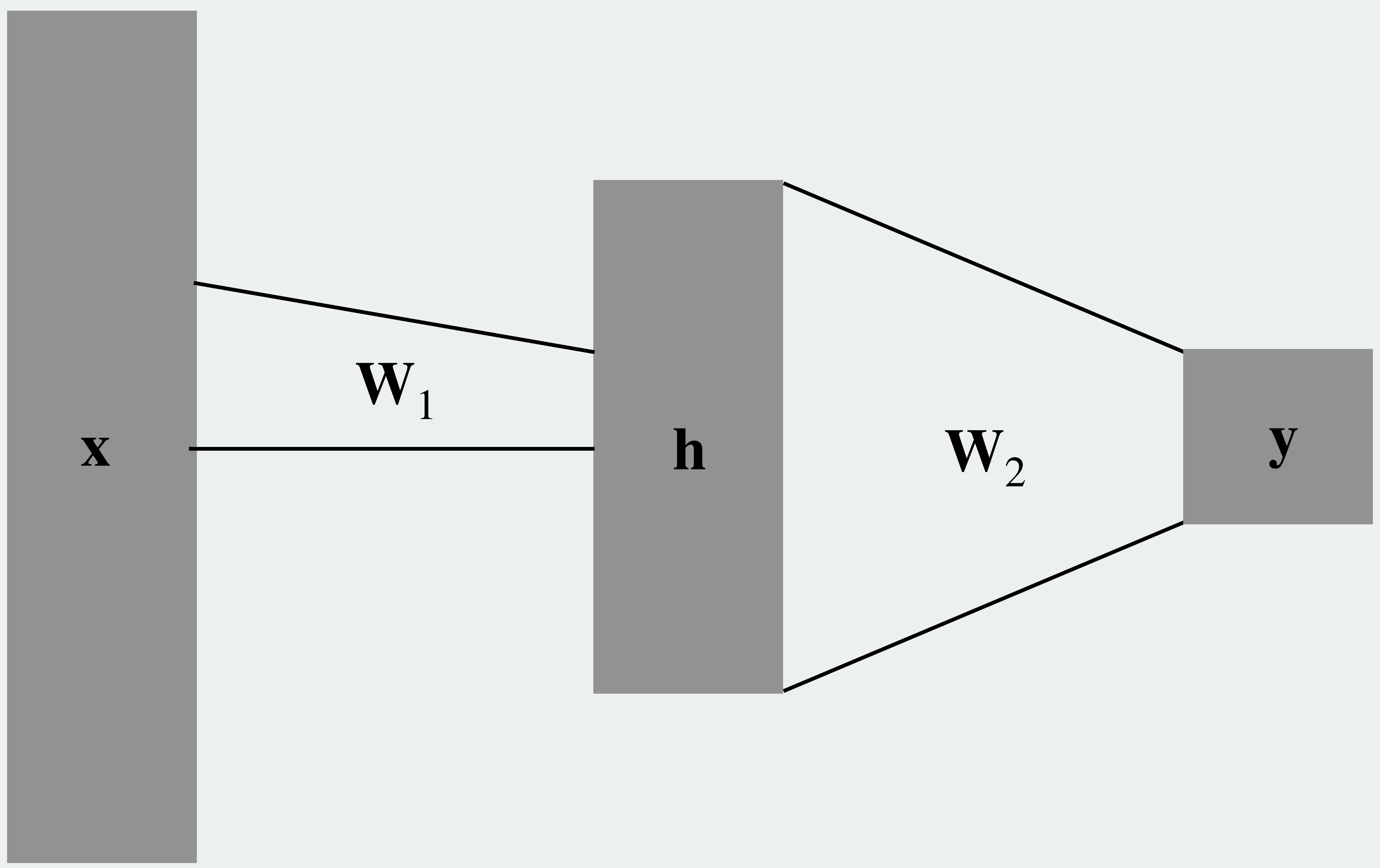
The problem with vanilla neural networks

> Solution: “slide” the weight matrix over the input



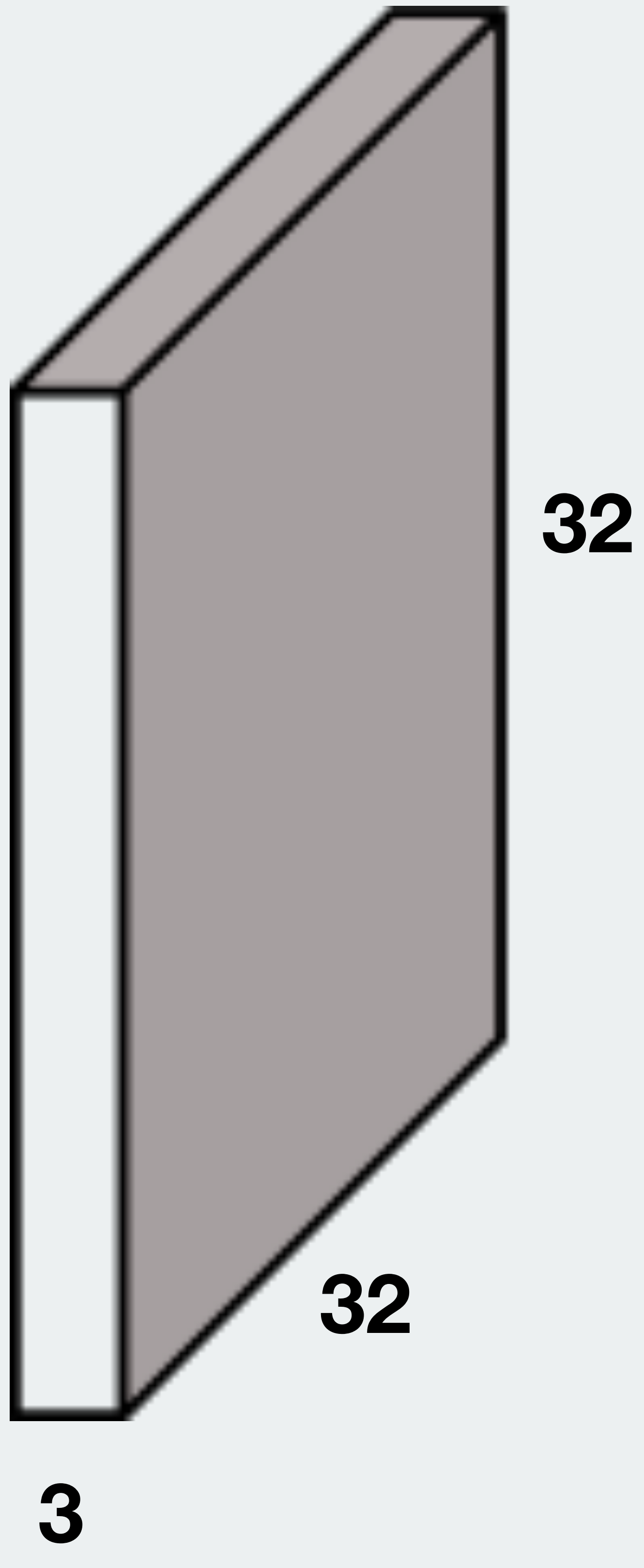
The problem with vanilla neural networks

> Solution: “slide” the weight matrix over the input

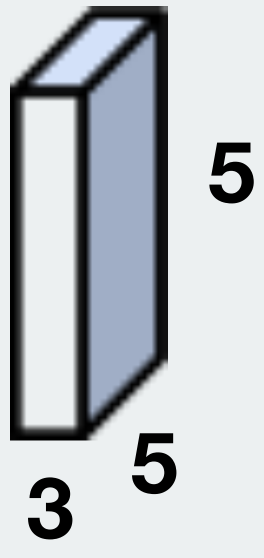


Convolutional neural networks

Input *image*
32 x 32 x 3

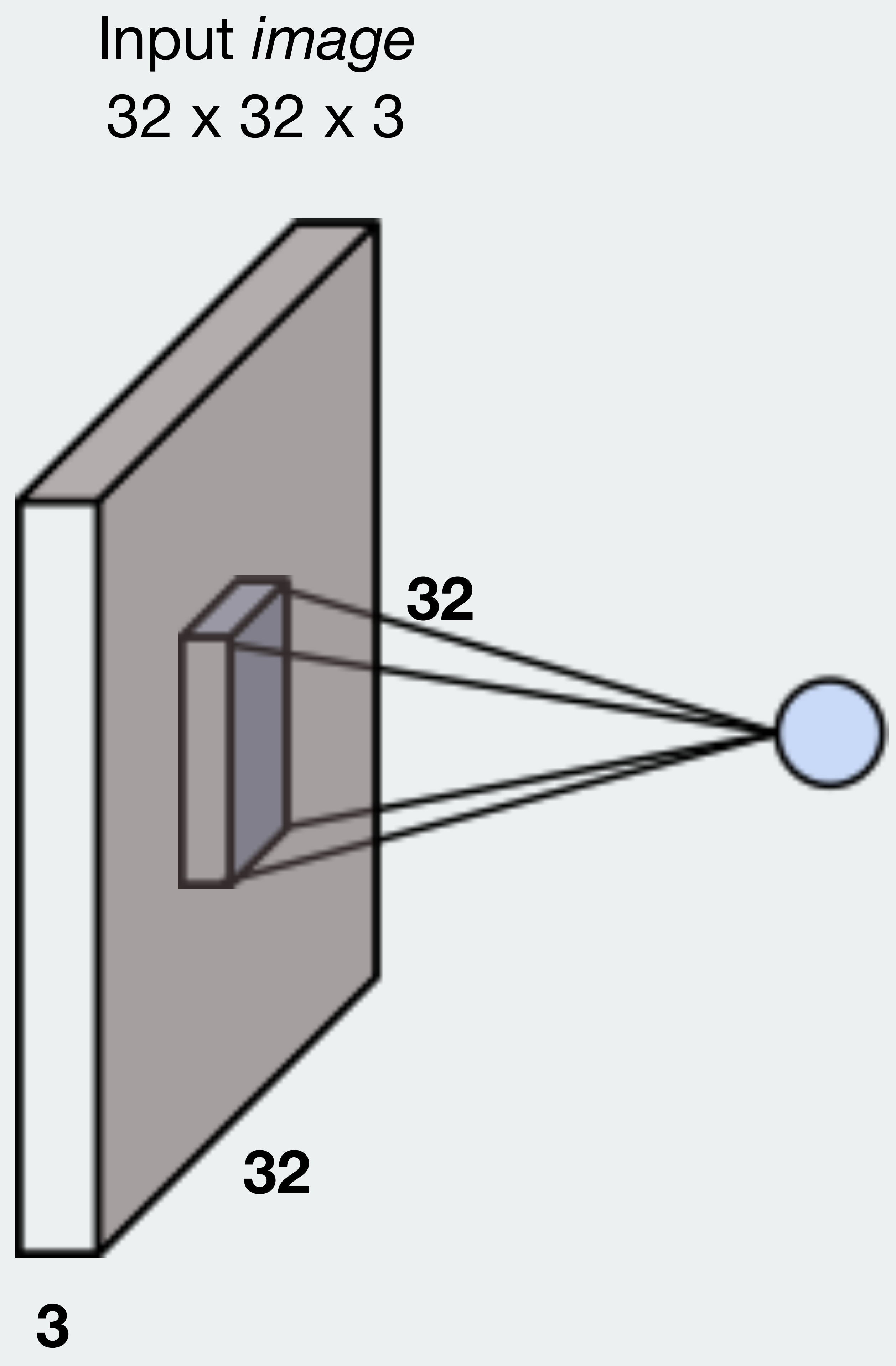


filter
5 x 5 x 3



Convolutional neural networks

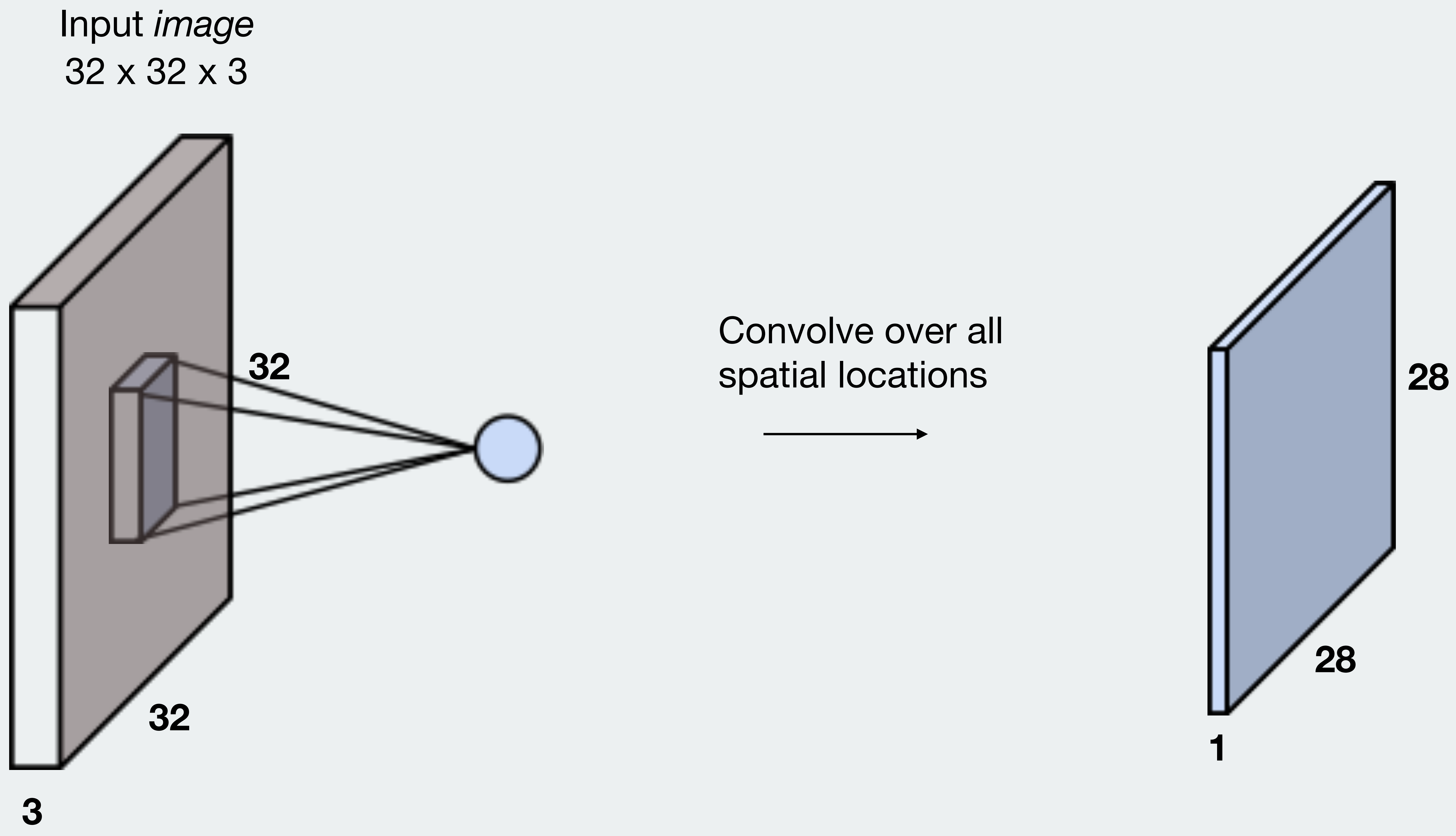
> Convolve the filter across the input image to computing dot products



1 number: the result of taking the dot product between the filter and one chunk of the image (5*5*3 = 75-dimensional dot product + bias)

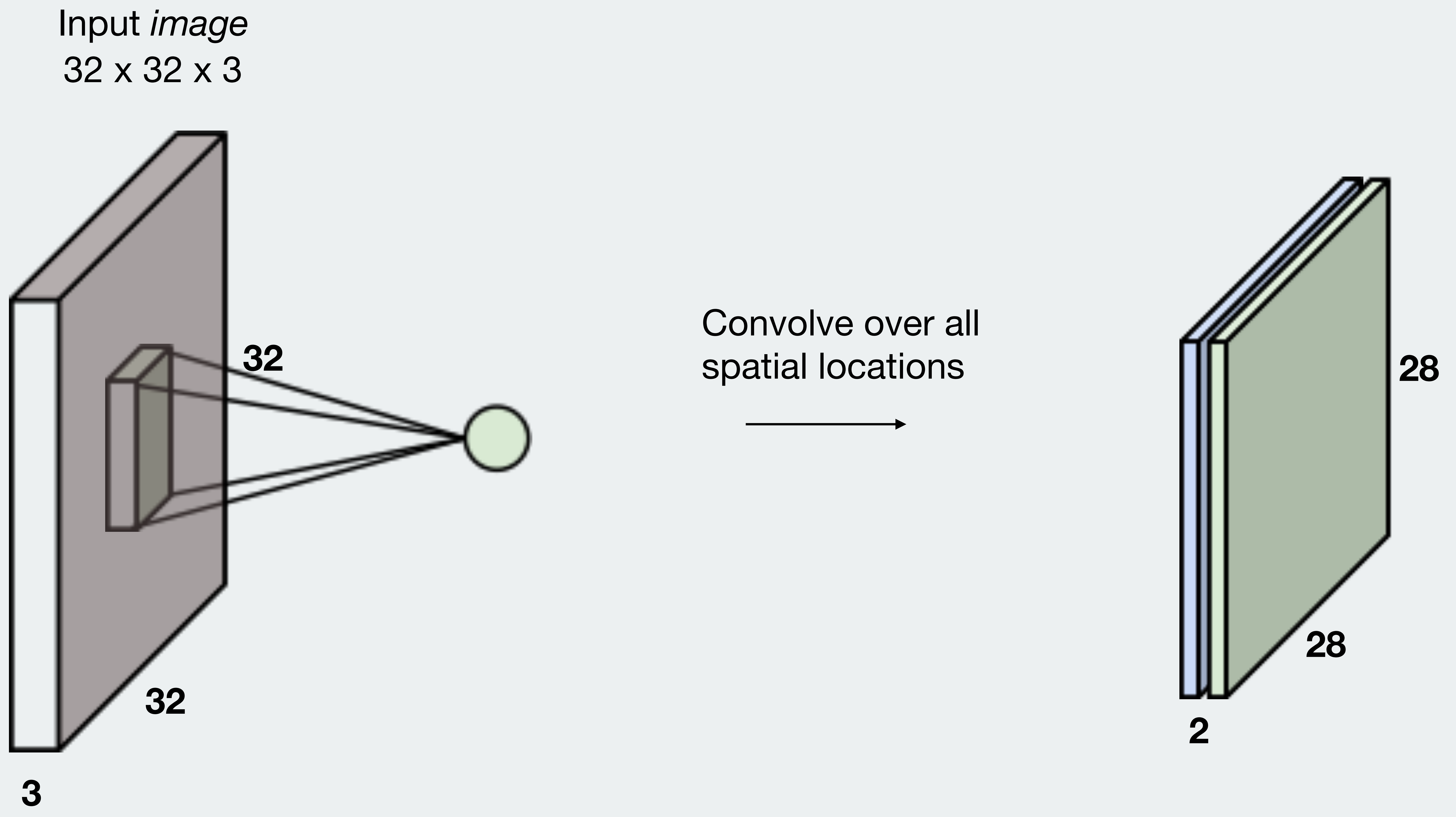
Convolutional neural networks

> Convolution by 1 filter produces new *activation map* of depth 1



Convolutional neural networks

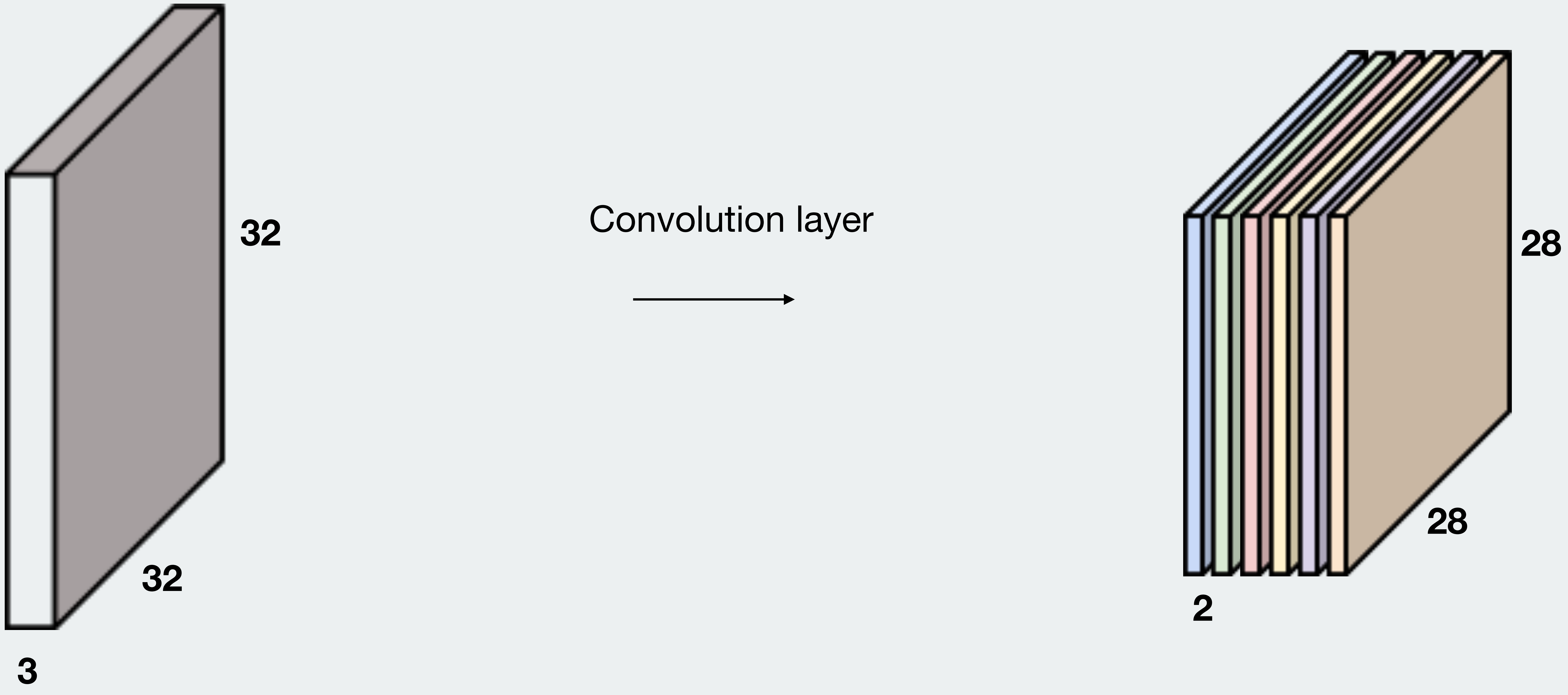
> Convolution by 2 filters produces new *activation map* of depth 2



Convolutional neural networks

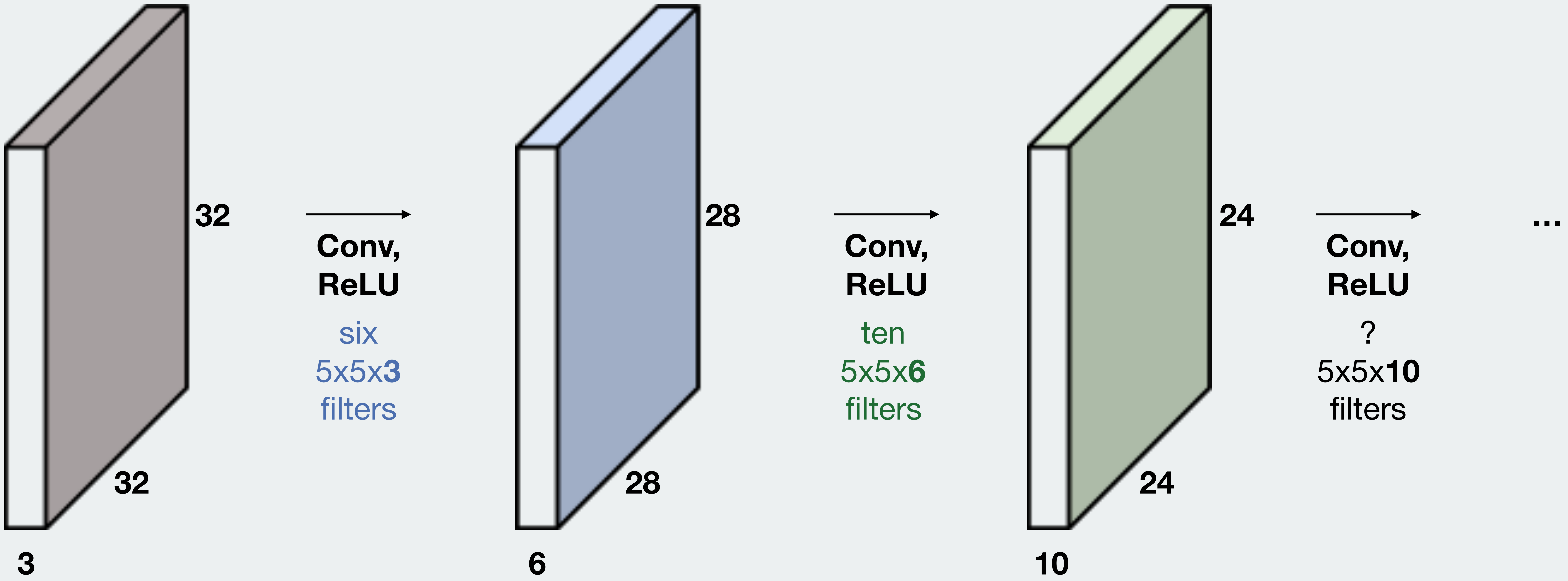
> Convolution by *n* filters produces new *activation map* of depth *n*

Input *image*
32 x 32 x 3



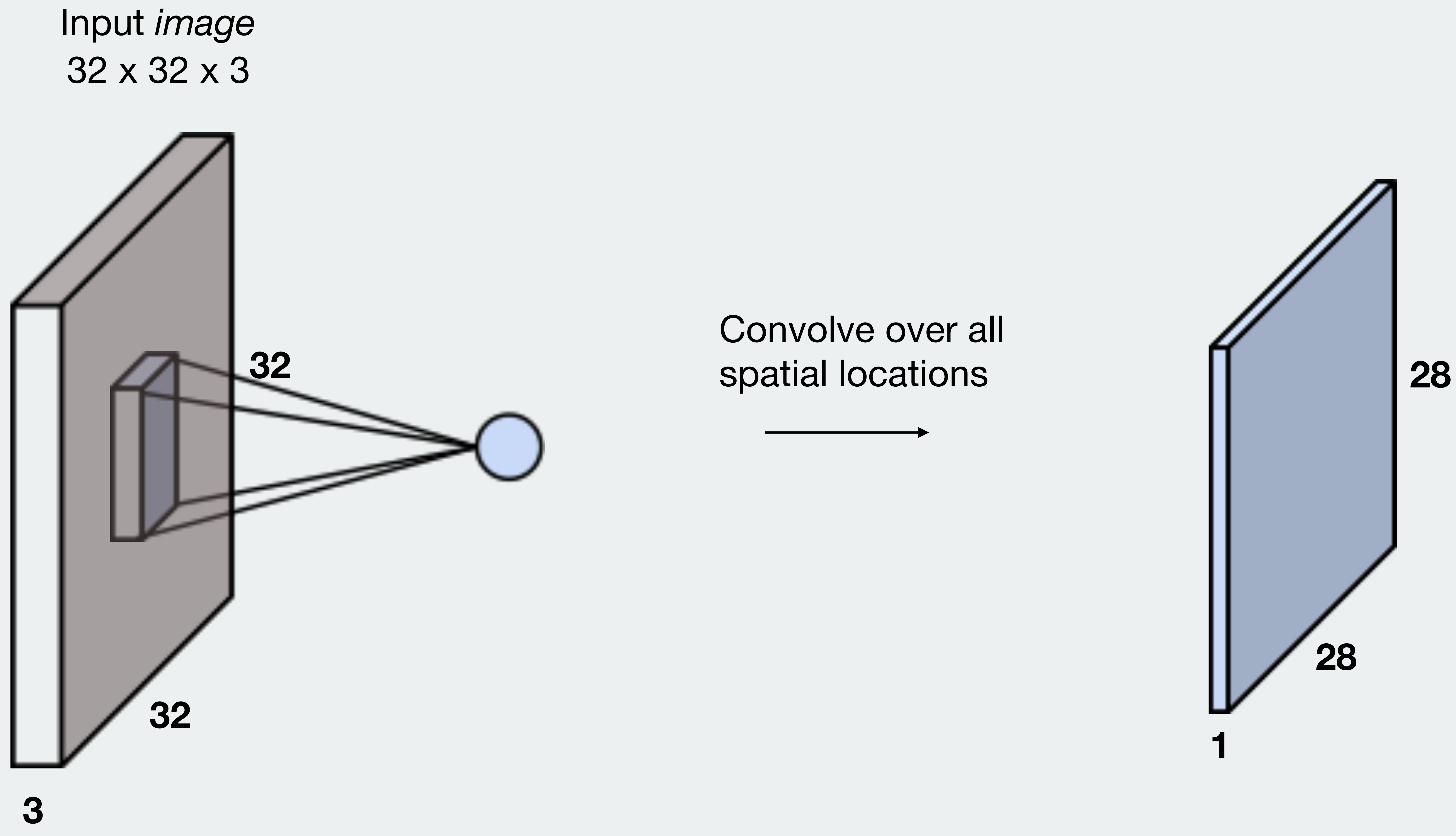
Convolutional neural networks

> Stack these operations



Convolutional neural networks

> Dimensions

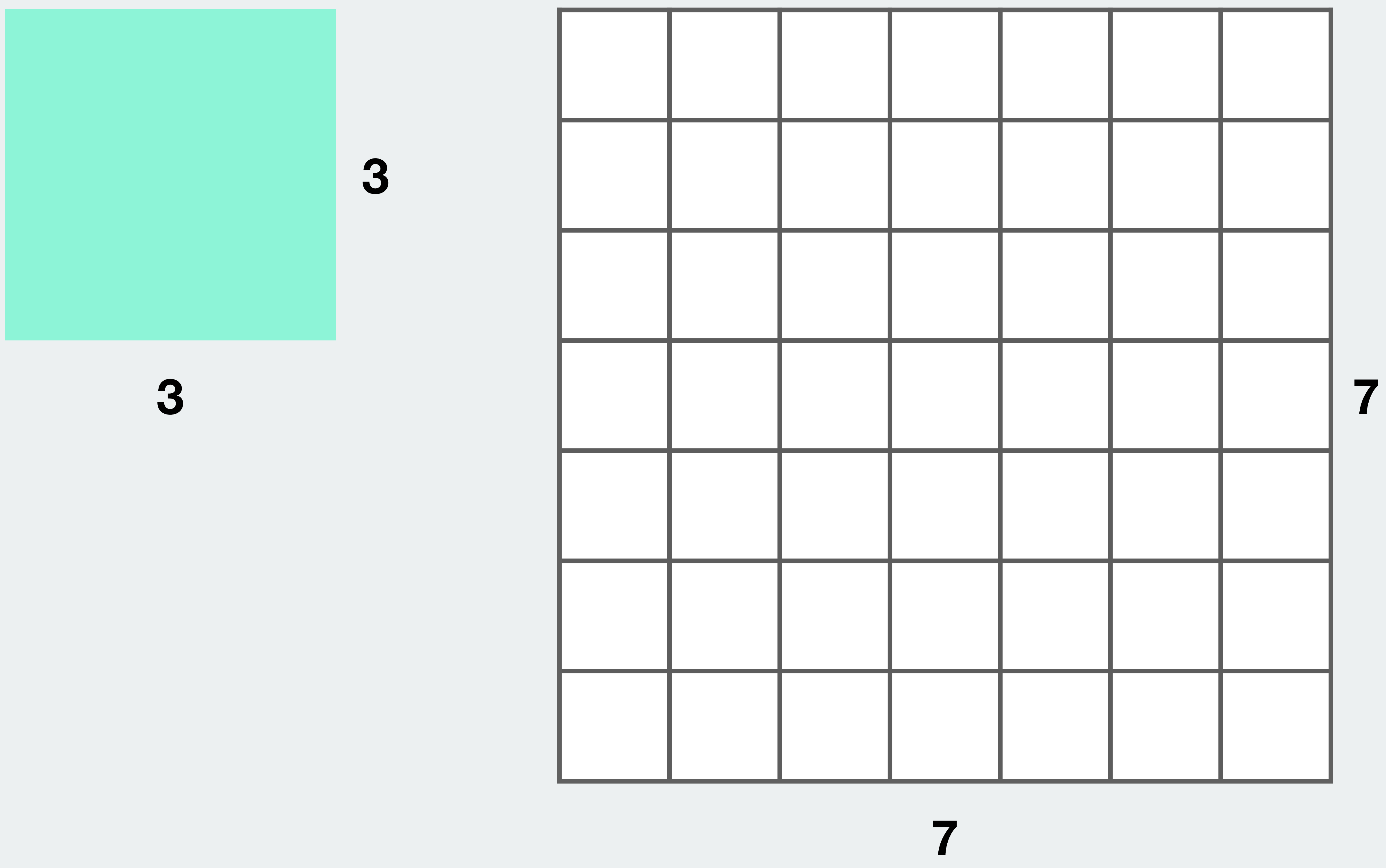


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

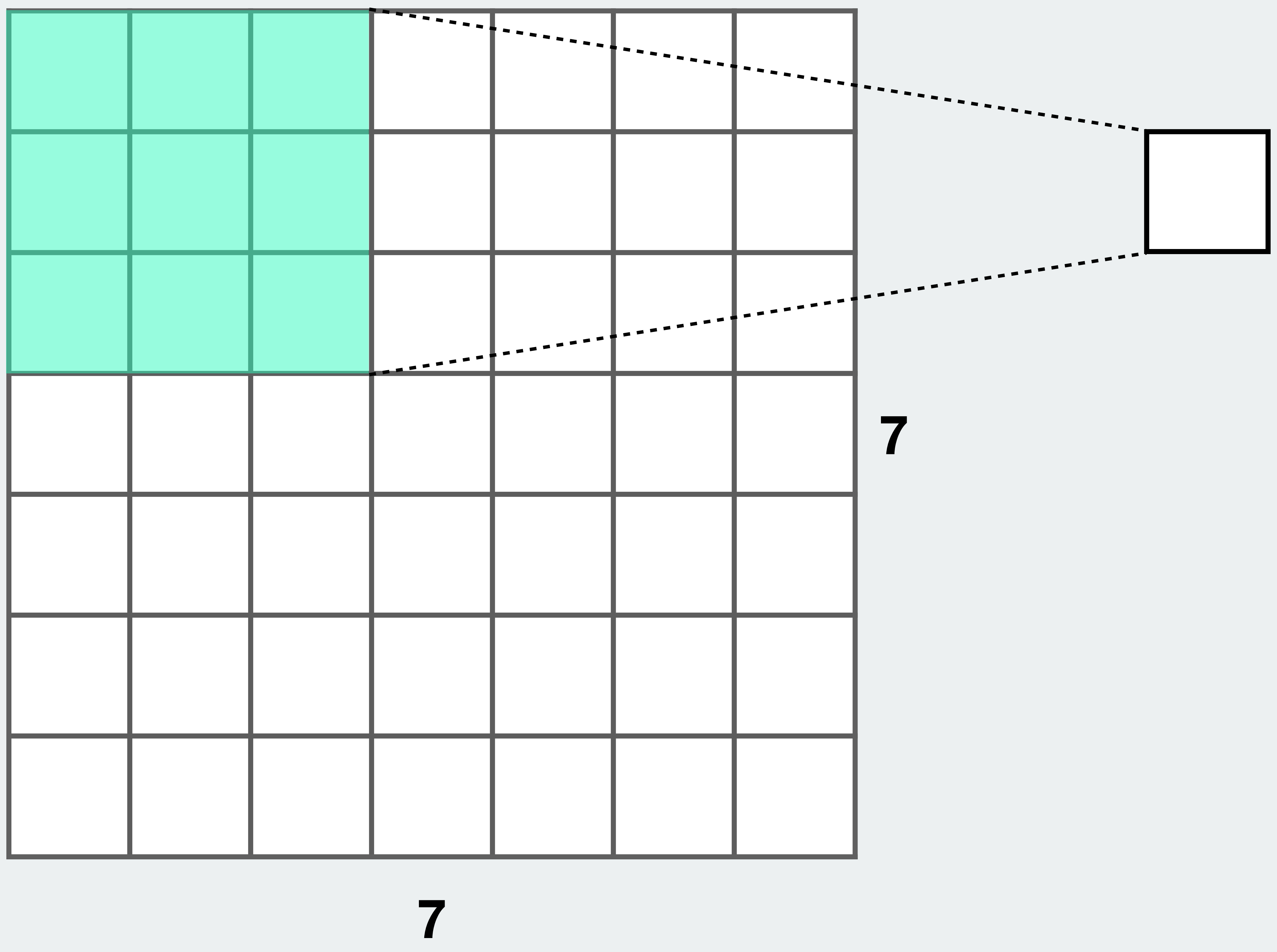


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

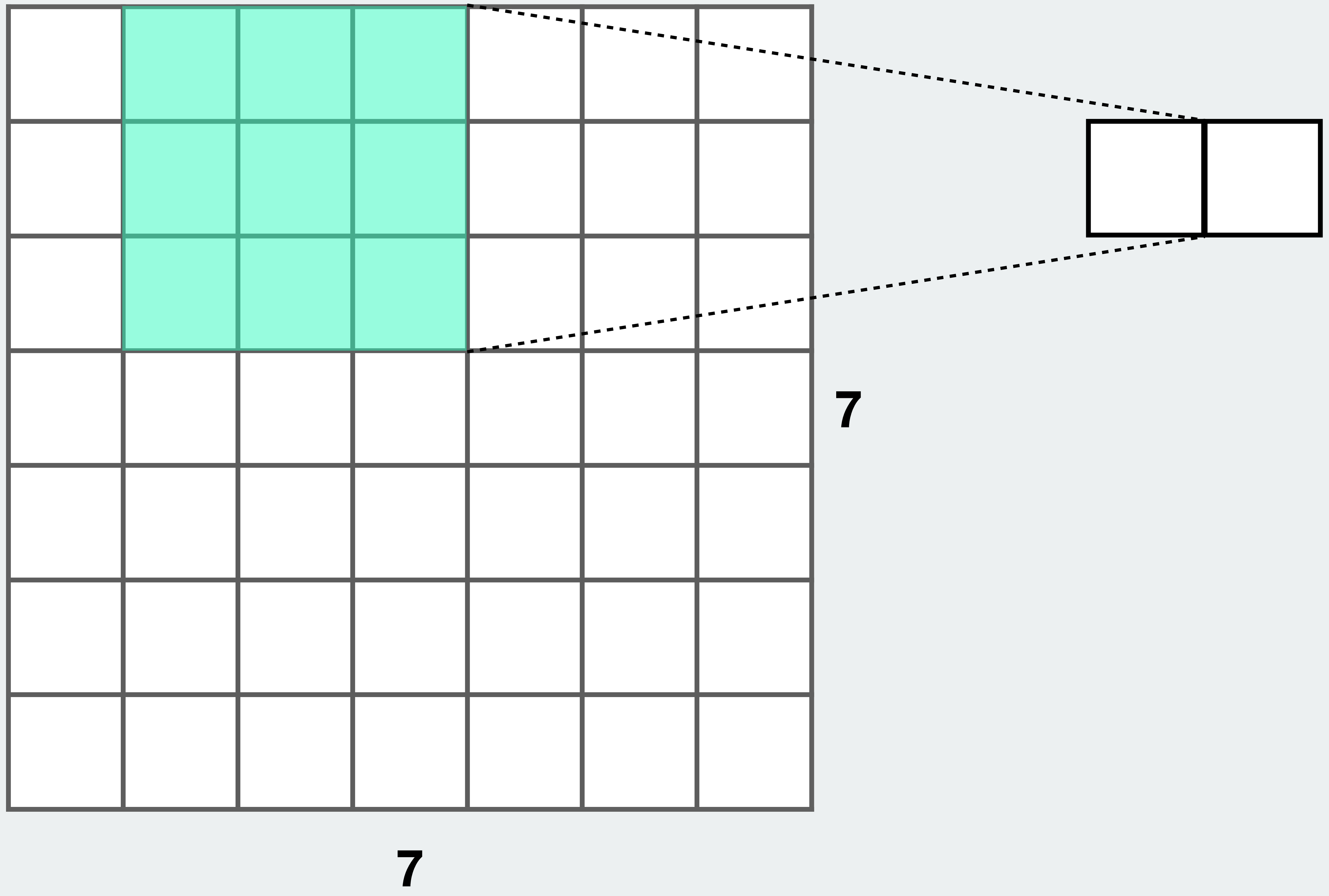


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

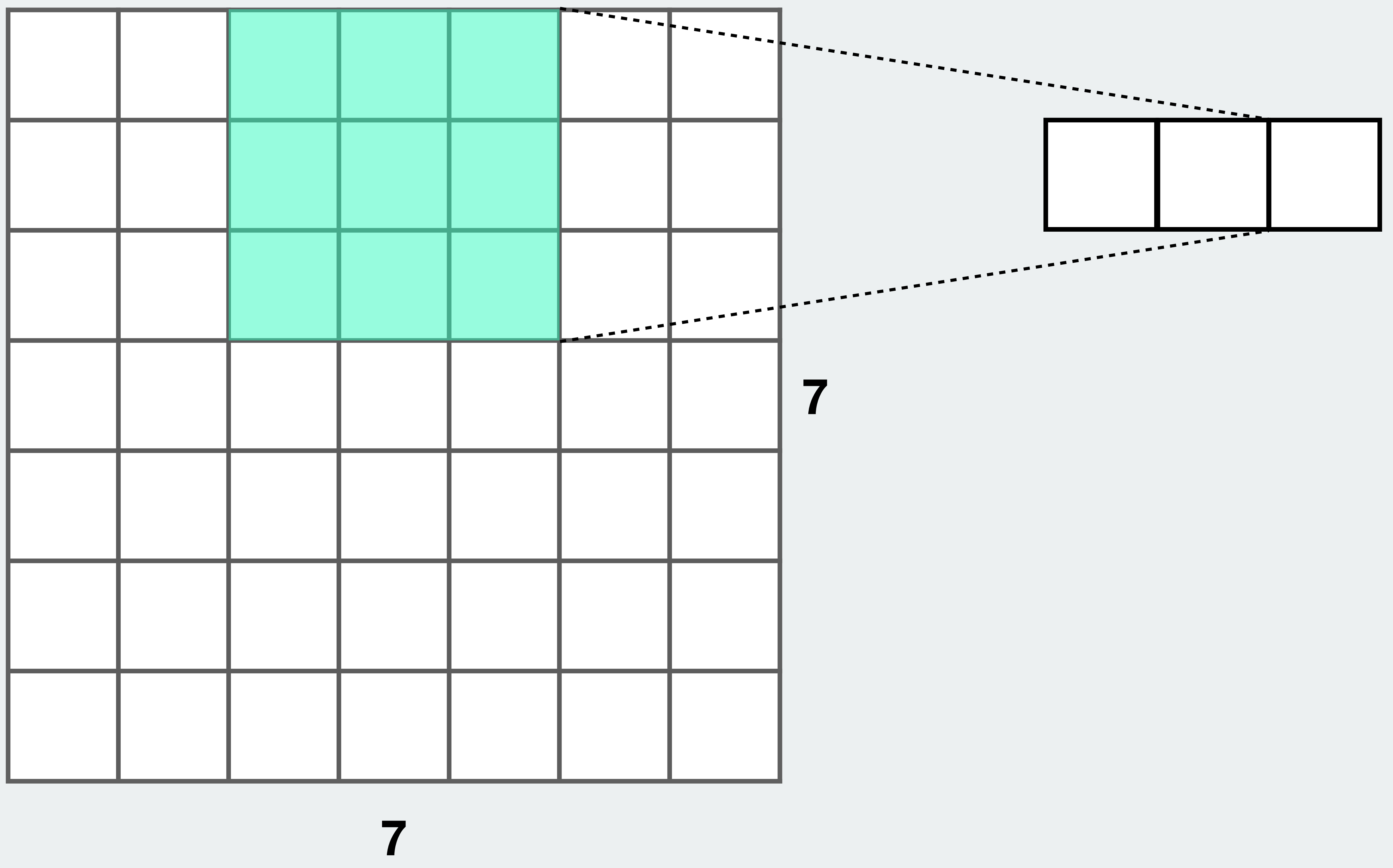


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

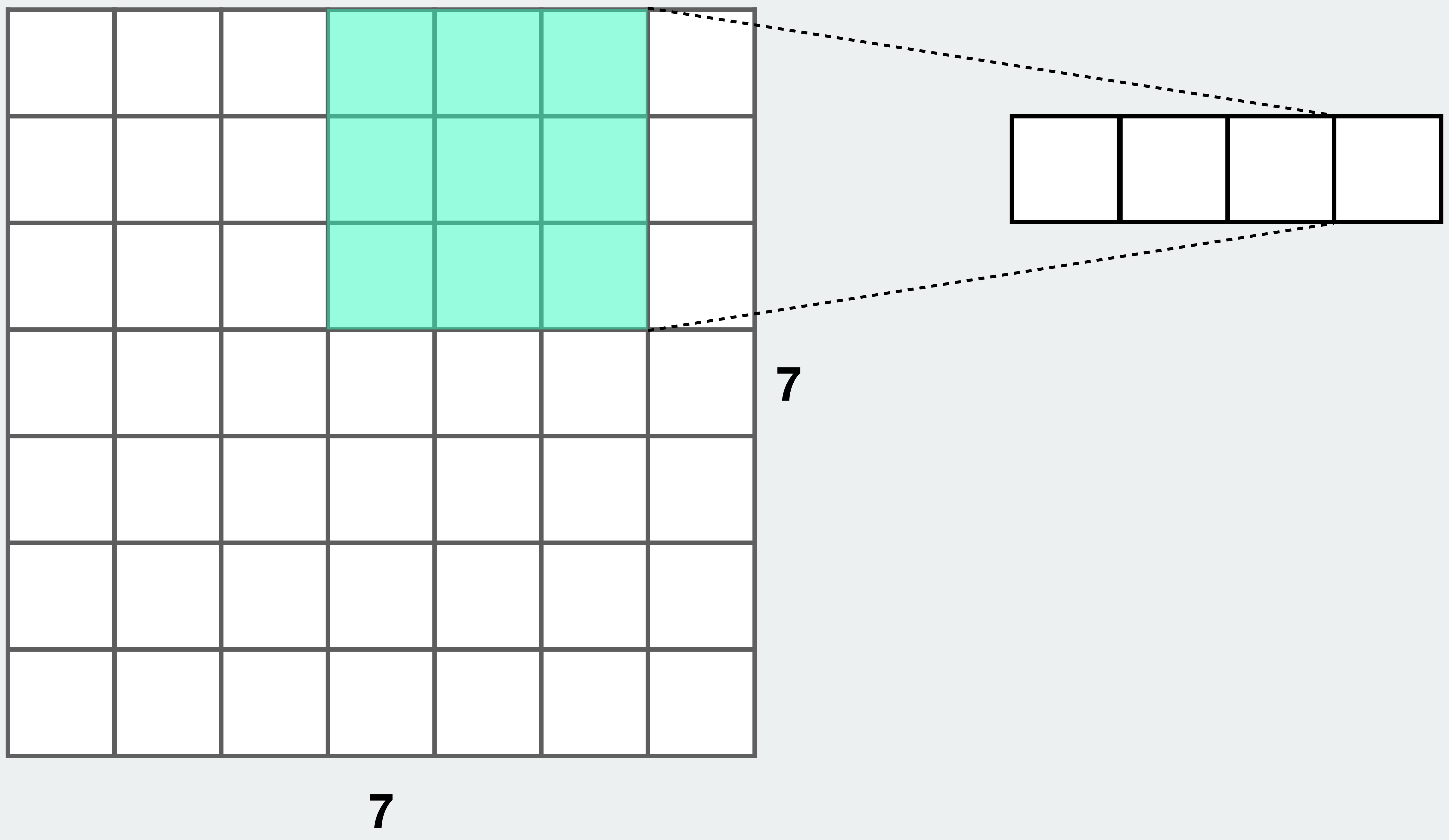


Convolutional neural networks

> Dimensions

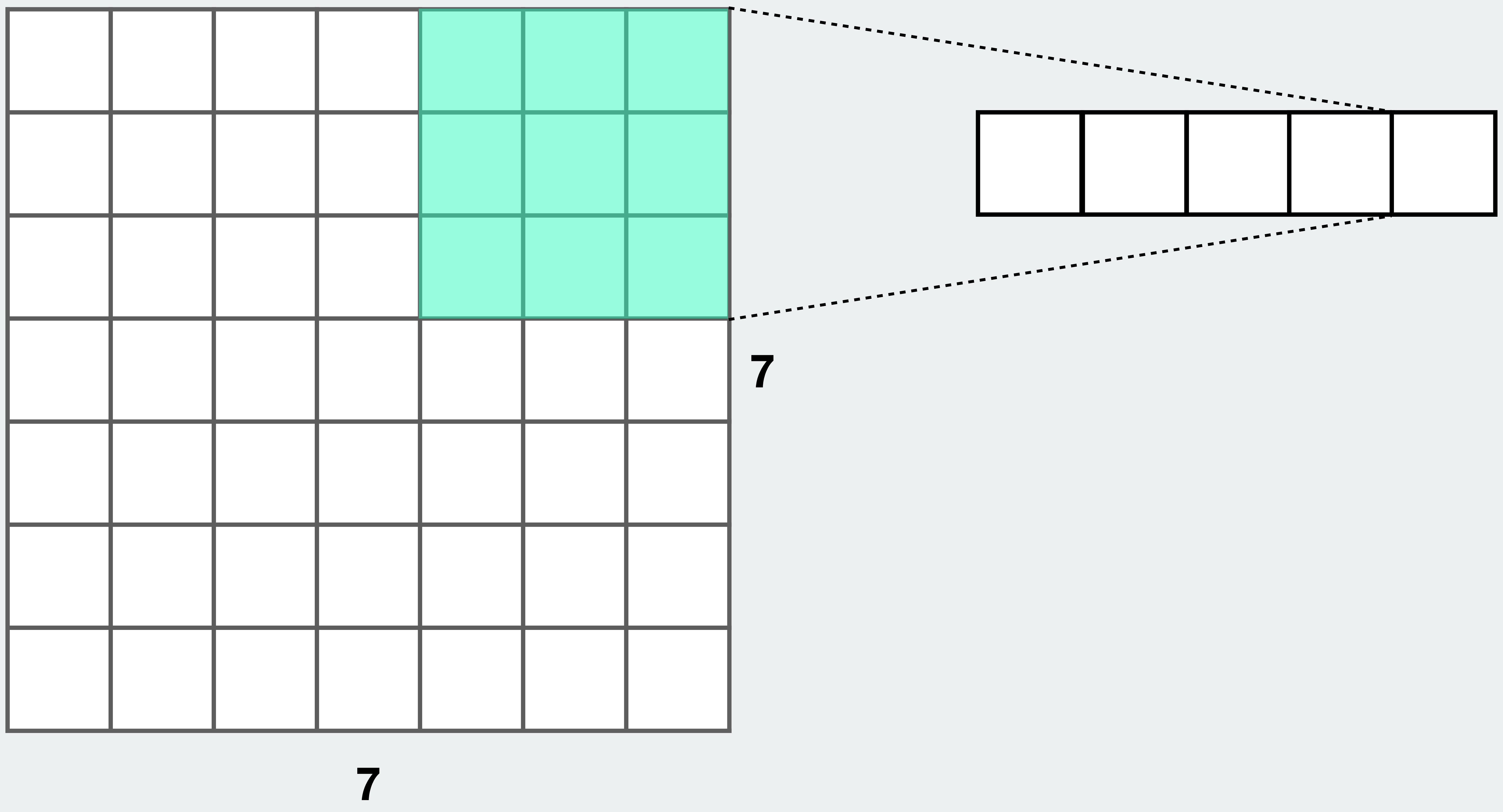
Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?



Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

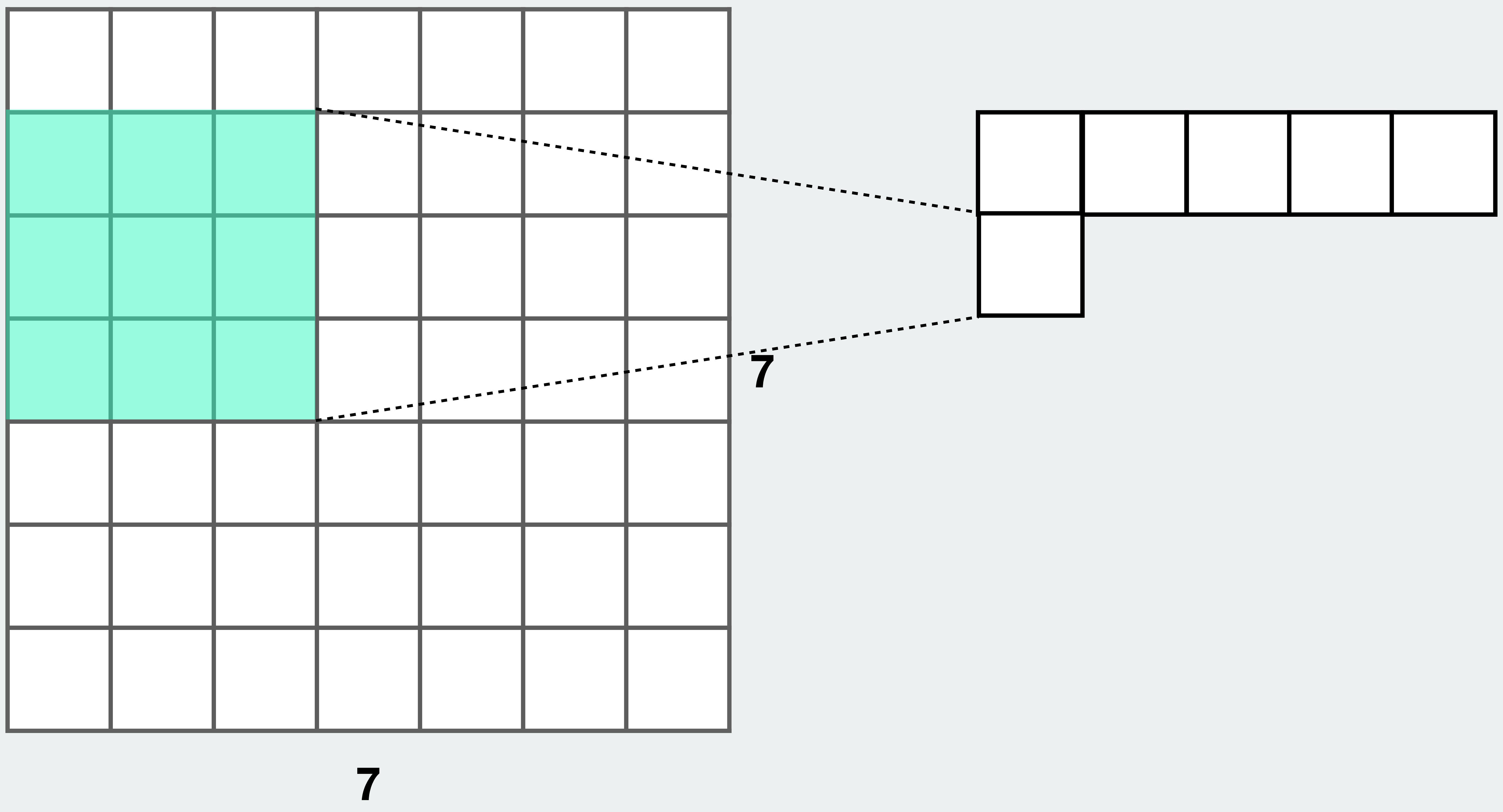


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

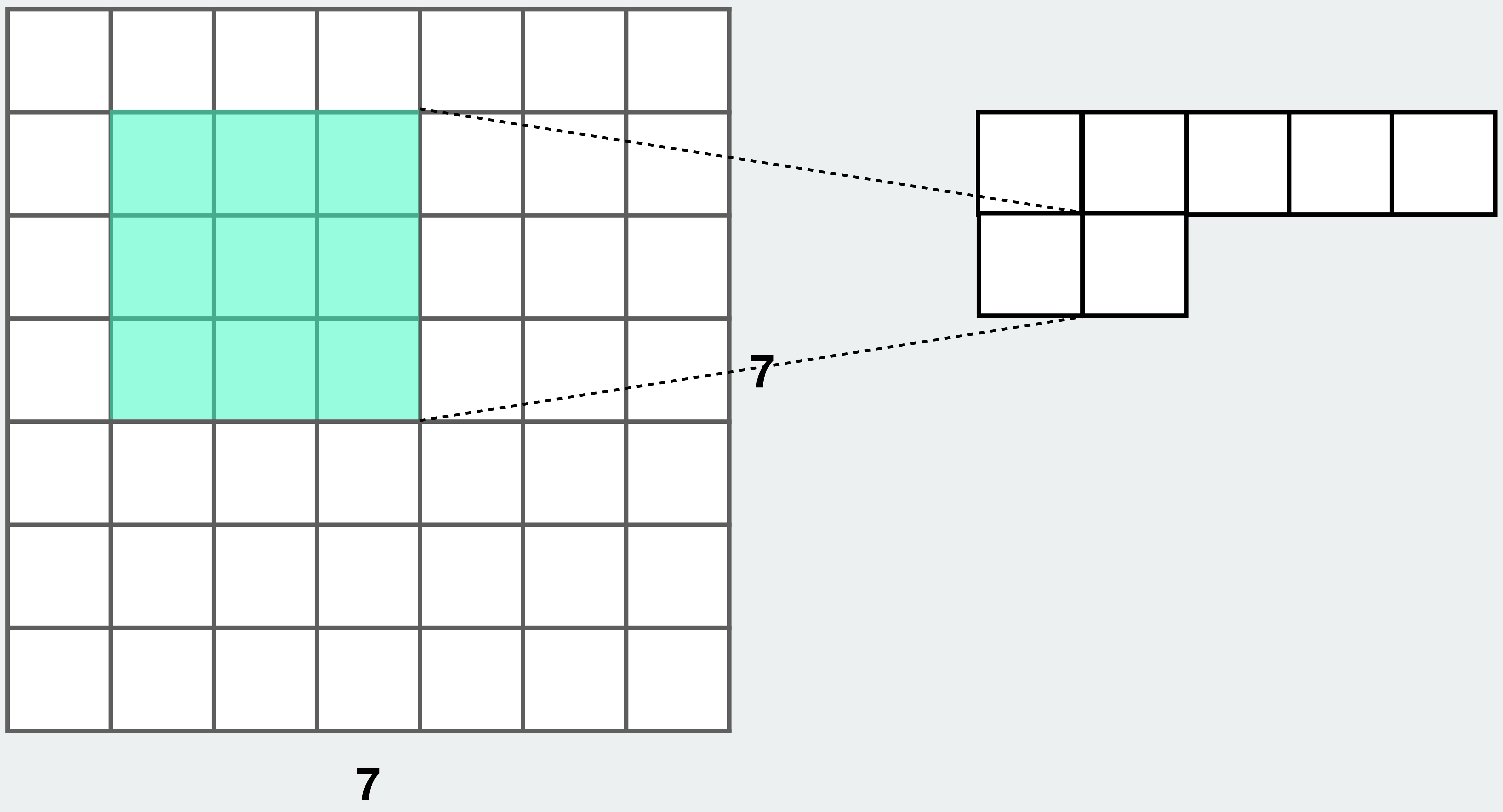


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?



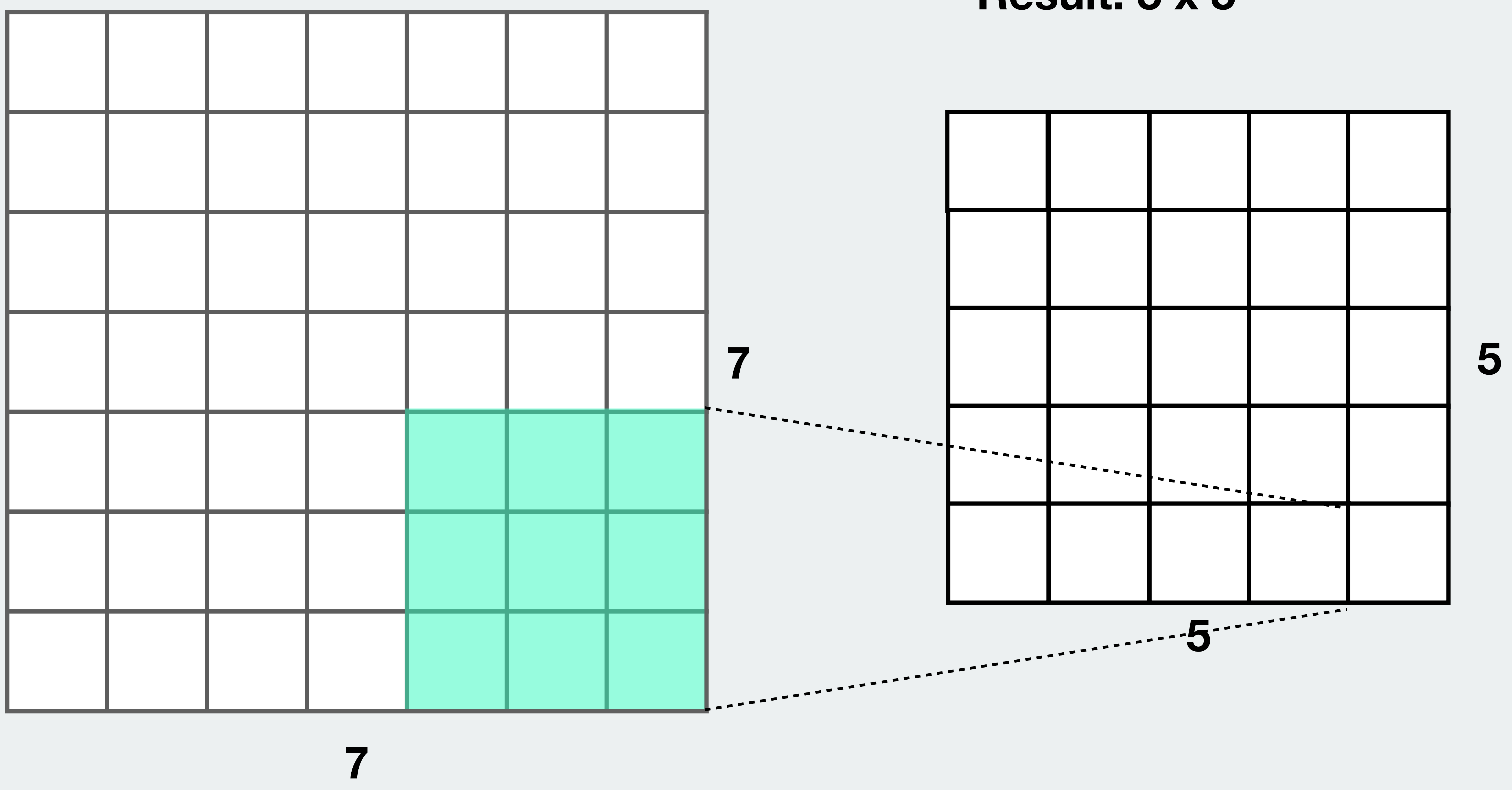
Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What is the dimensions of the resulting activation map?

Result: 5 x 5

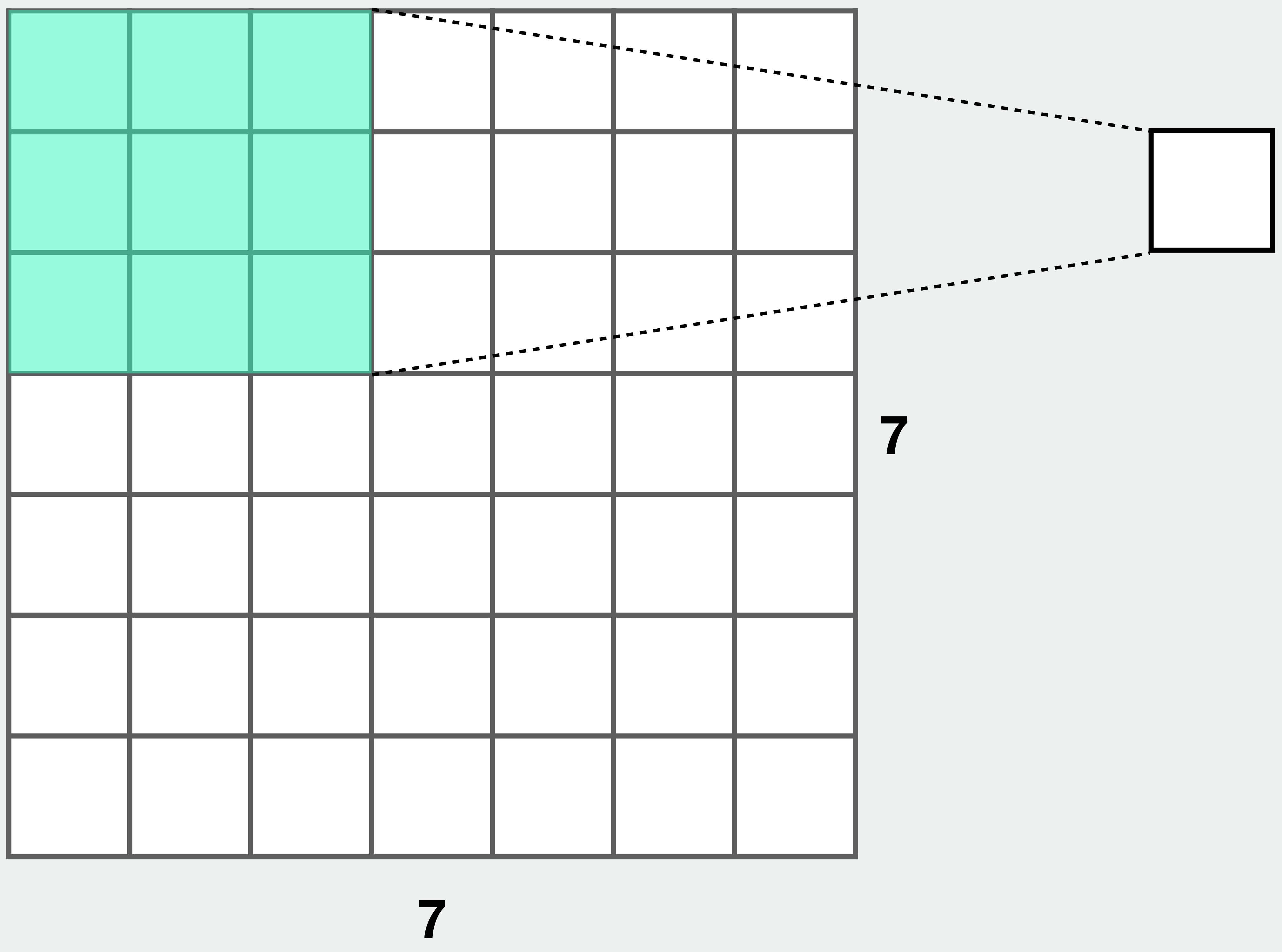


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What if we use stride 2?

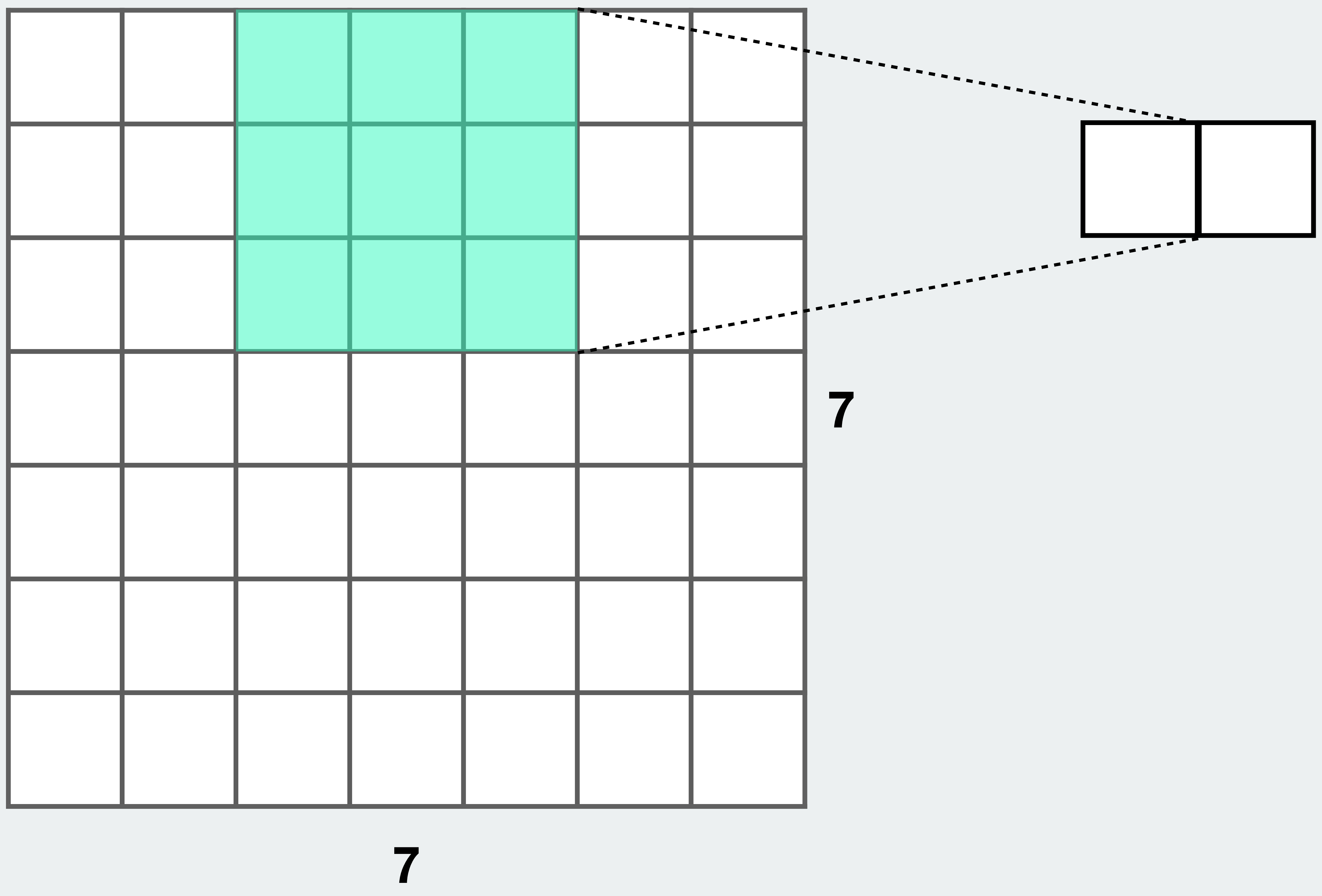


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What if we use stride 2?

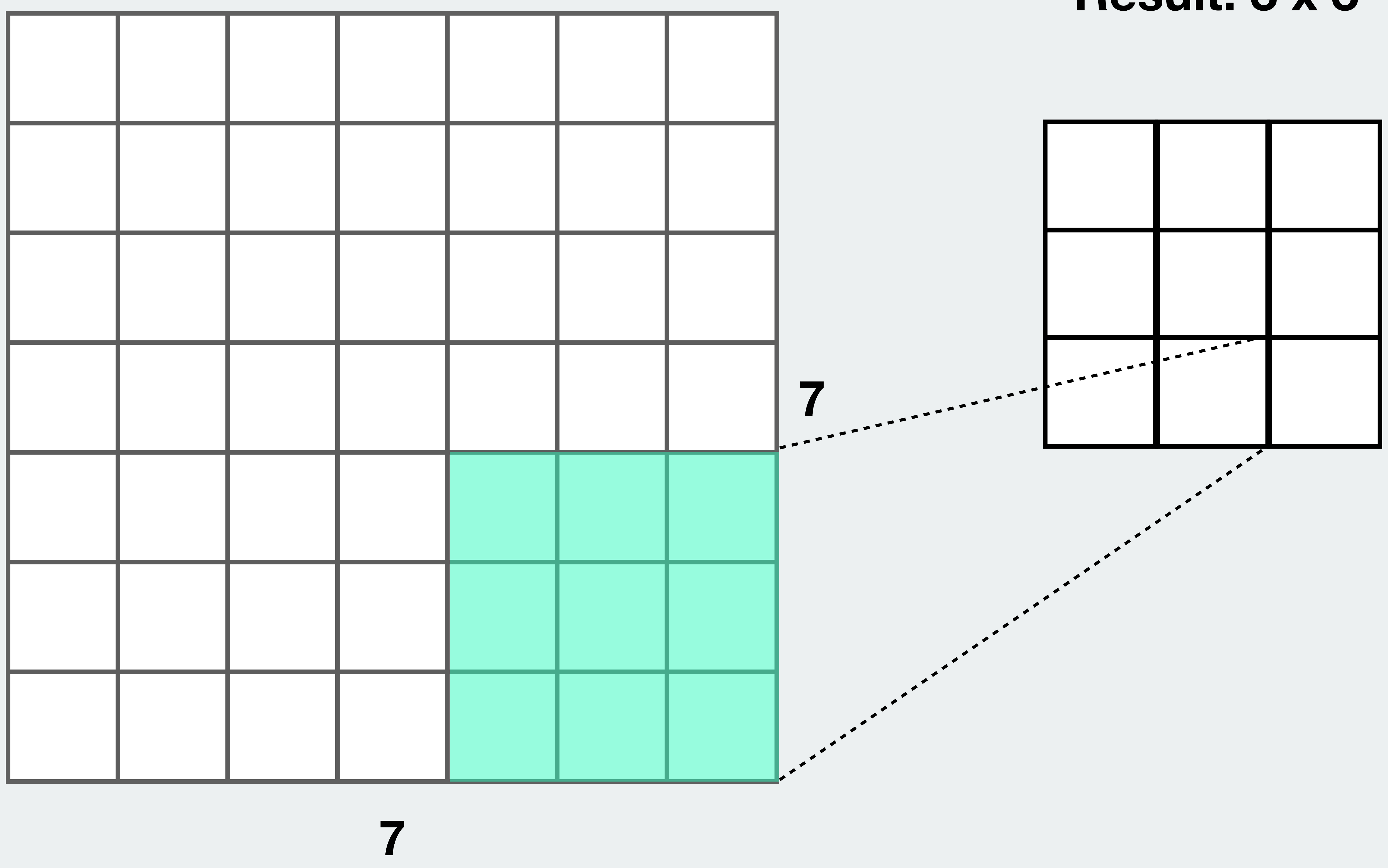


Convolutional neural networks
> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: What if we use stride 2?

Result: 3 x 3

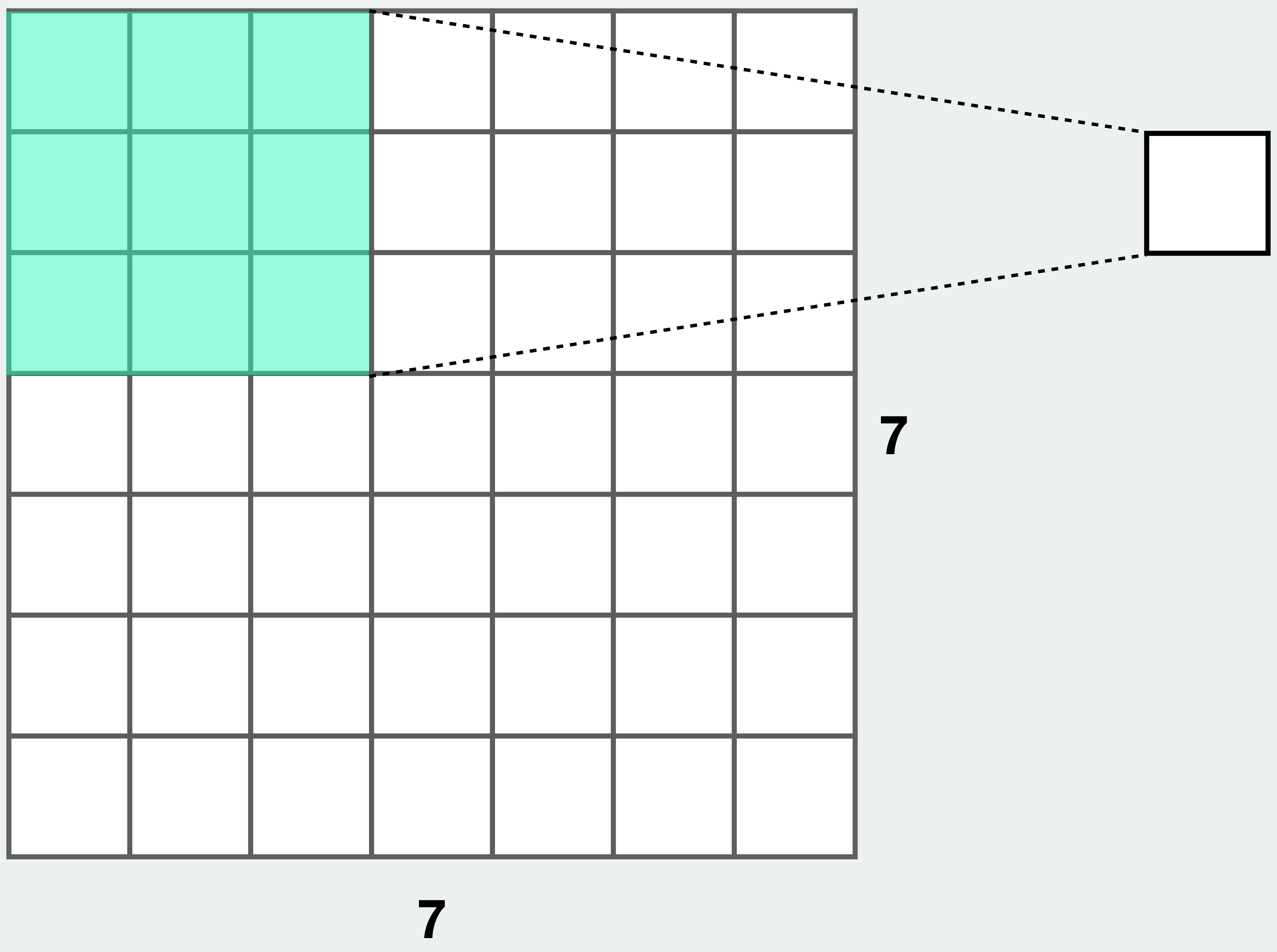


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: Stride 3?

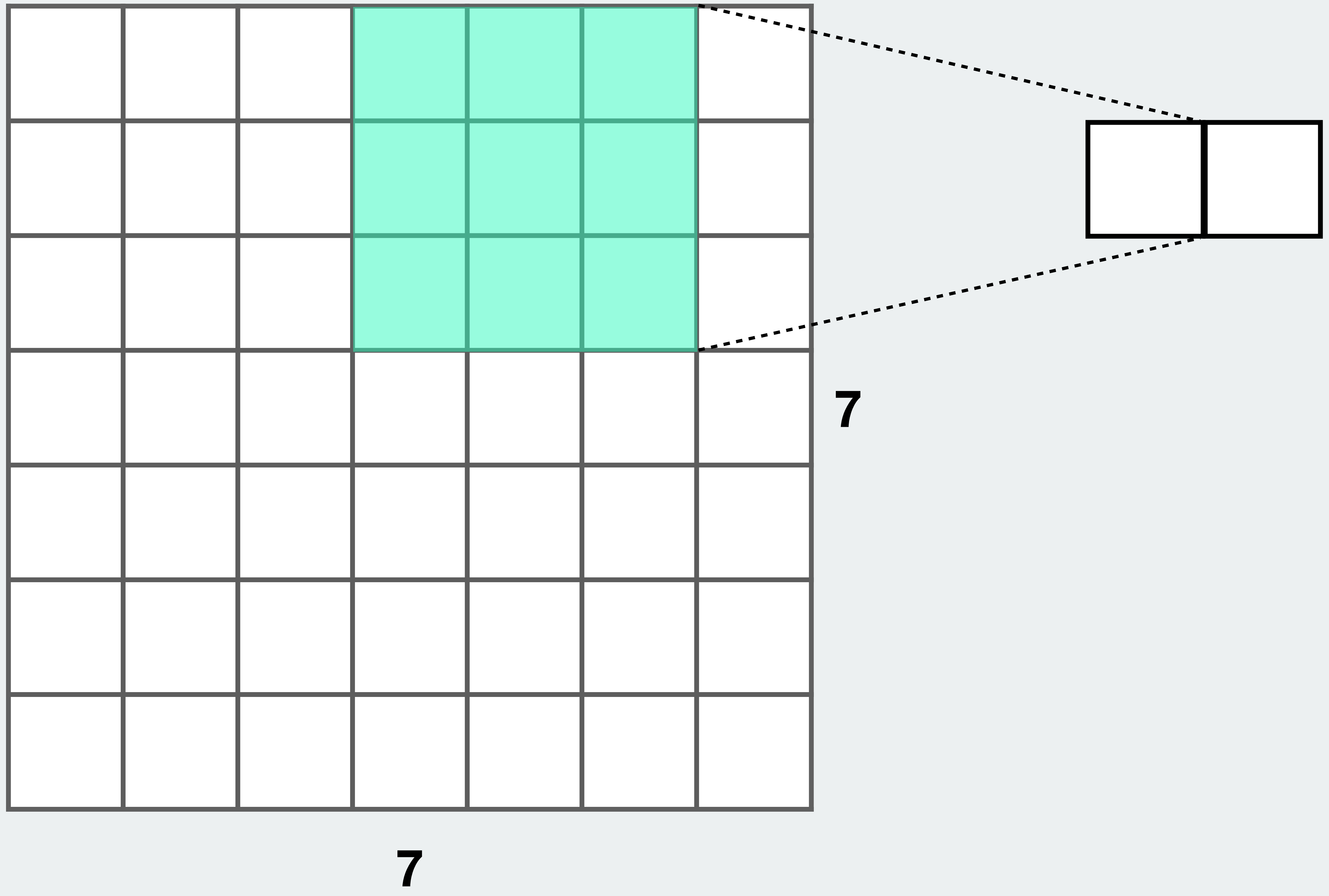


Convolutional neural networks

> Dimensions

Example: 7 x 7 input
3 x 3 filter

Question: Stride 3?

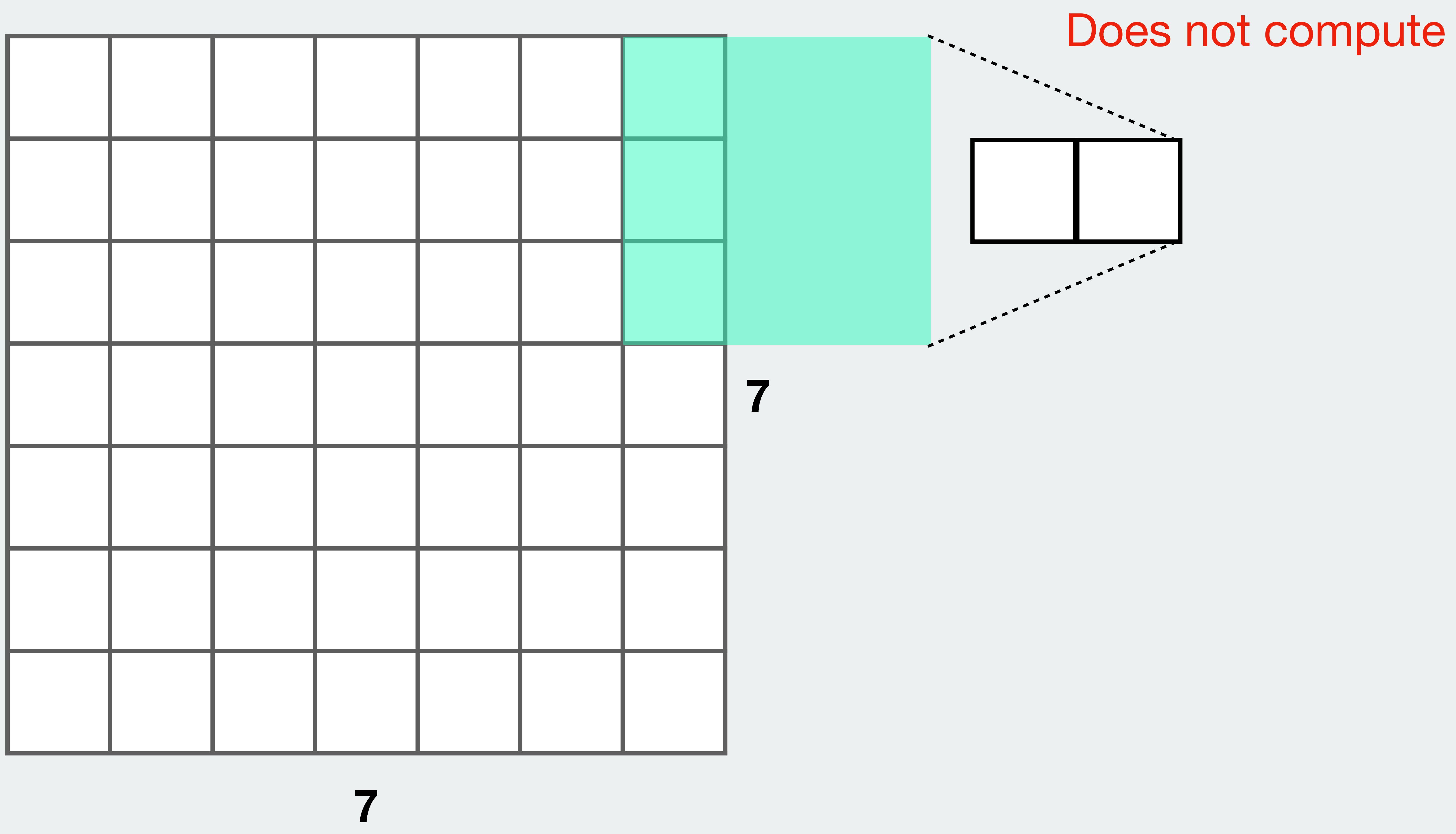


Convolutional neural networks

> Dimensions

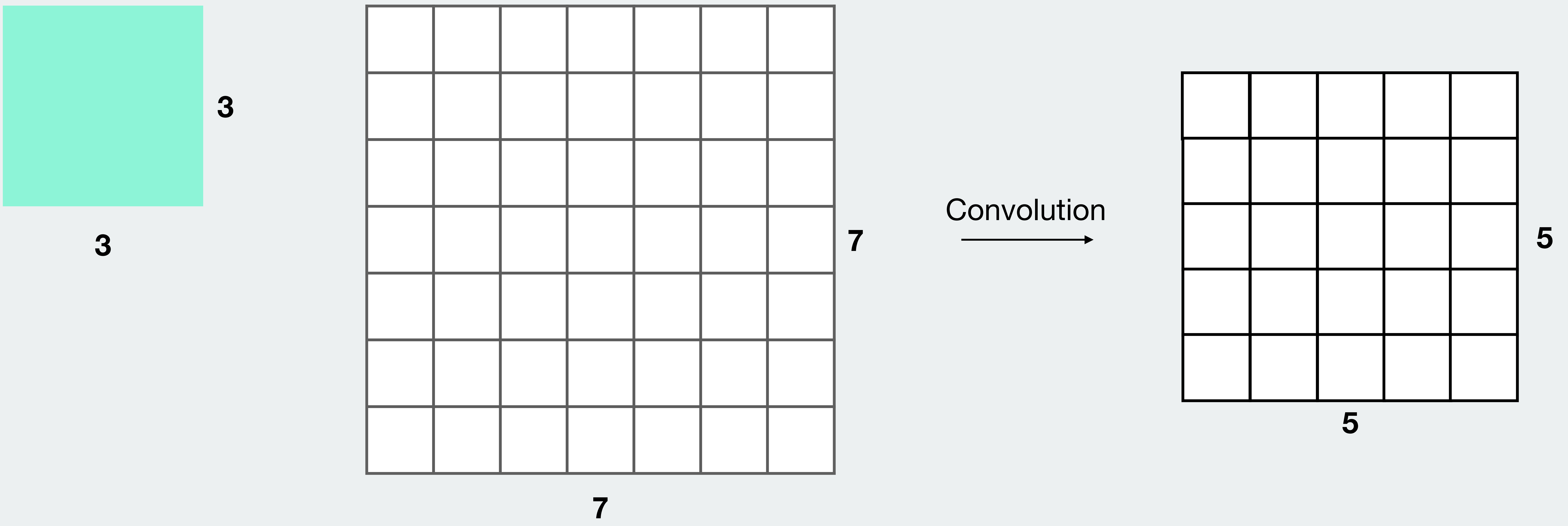
Example: 7 x 7 input
3 x 3 filter

Question: Stride 3?



Problem: The image *shrinks*

7 x 7 => **5 x 5**



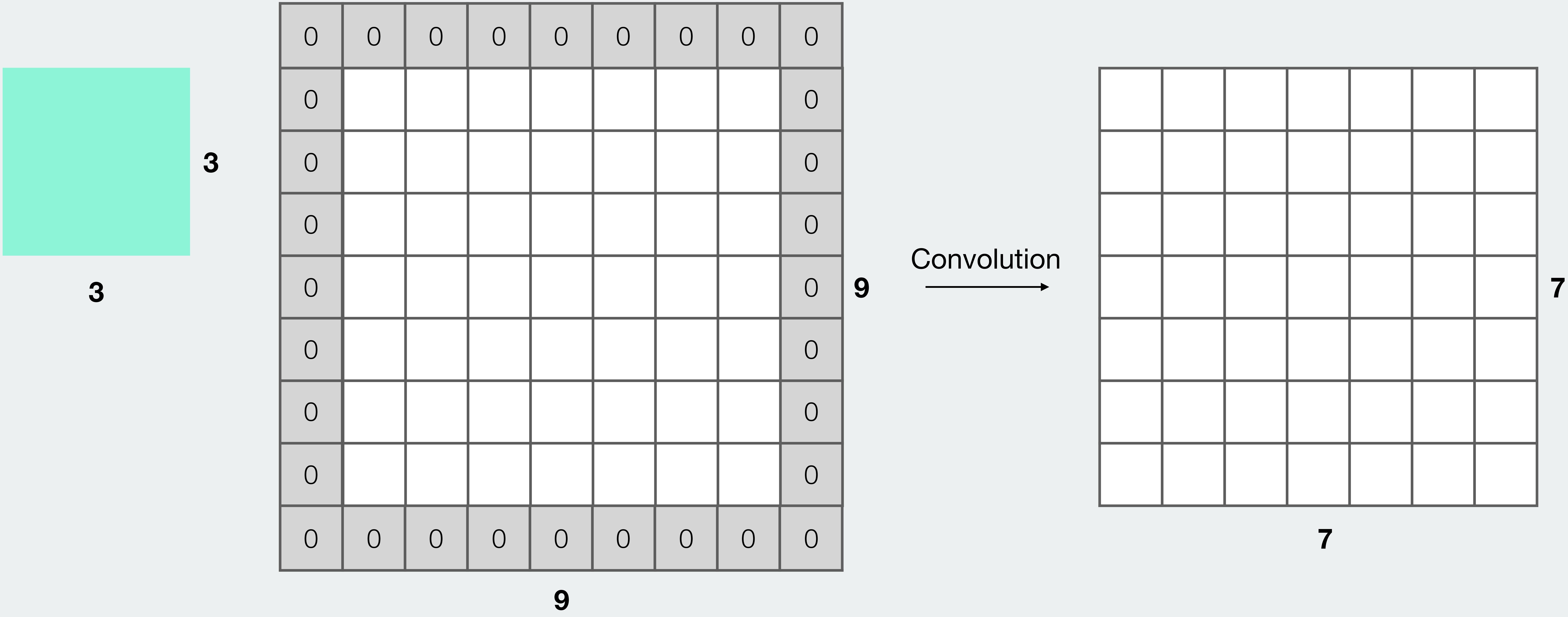
Convolutional neural networks

> Dimensions

Problem: The image *shrinks*

Solution: *Padding!*

7 x 7 => **5 x 5**

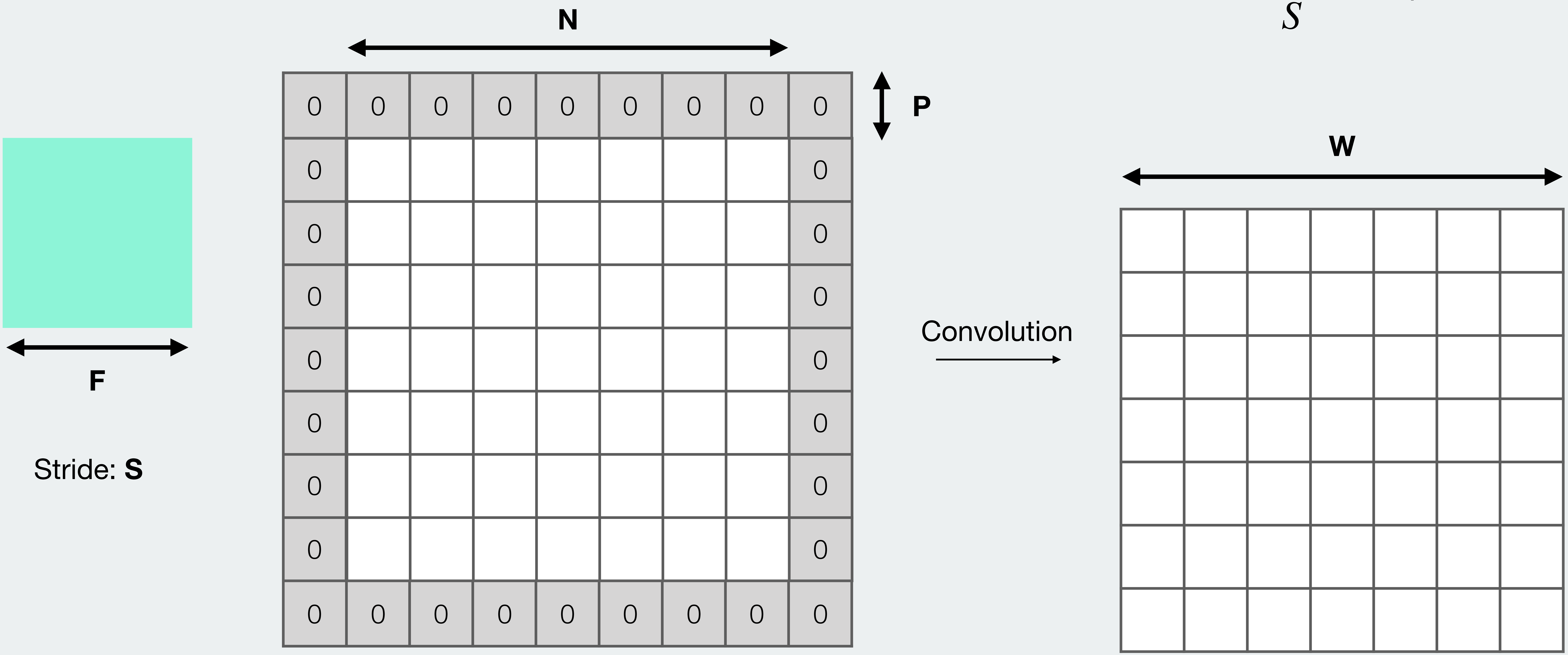


Convolutional neural networks

> Dimensions

General formula

$$W = \frac{N + 2P - F}{S} + 1$$



Animations

Given: 128 x 128 x 3 input

Ten 5 x 5 x 3 filters

Padding: 2

Stride: 1

Question 1: What are the output dimensions?

Question 2: What is the number of parameters?

Question 3: What if $F = N + 2P$?

Given: 128 x 128 x 3 input

Ten 5 x 5 x 3 filters

Padding: 2

Stride: 1

Question 1: What are the output dimensions?

Question 2: What is the number of parameters?

Question 3: What if $F = N + 2P$?

Answer 1:

$$W = \frac{128 + 2 \cdot 2 - 5}{1} + 1 = 128$$

Answer 2:

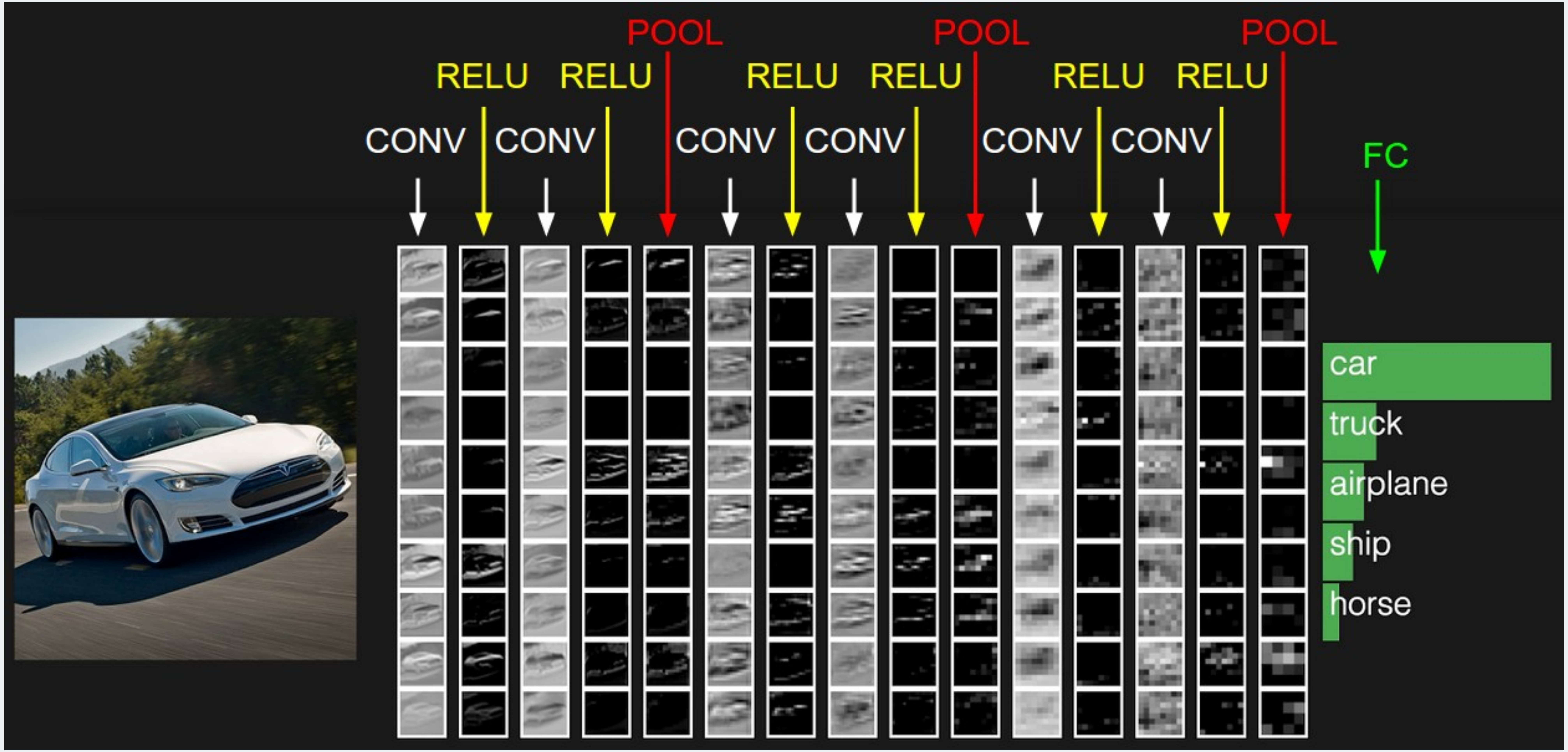
$$5 \cdot 5 \cdot 3 \cdot 10 + 10 = 760$$

Answer 3:

Then it's just a VNN!

Convolutional neural networks

> Example of a bigger network

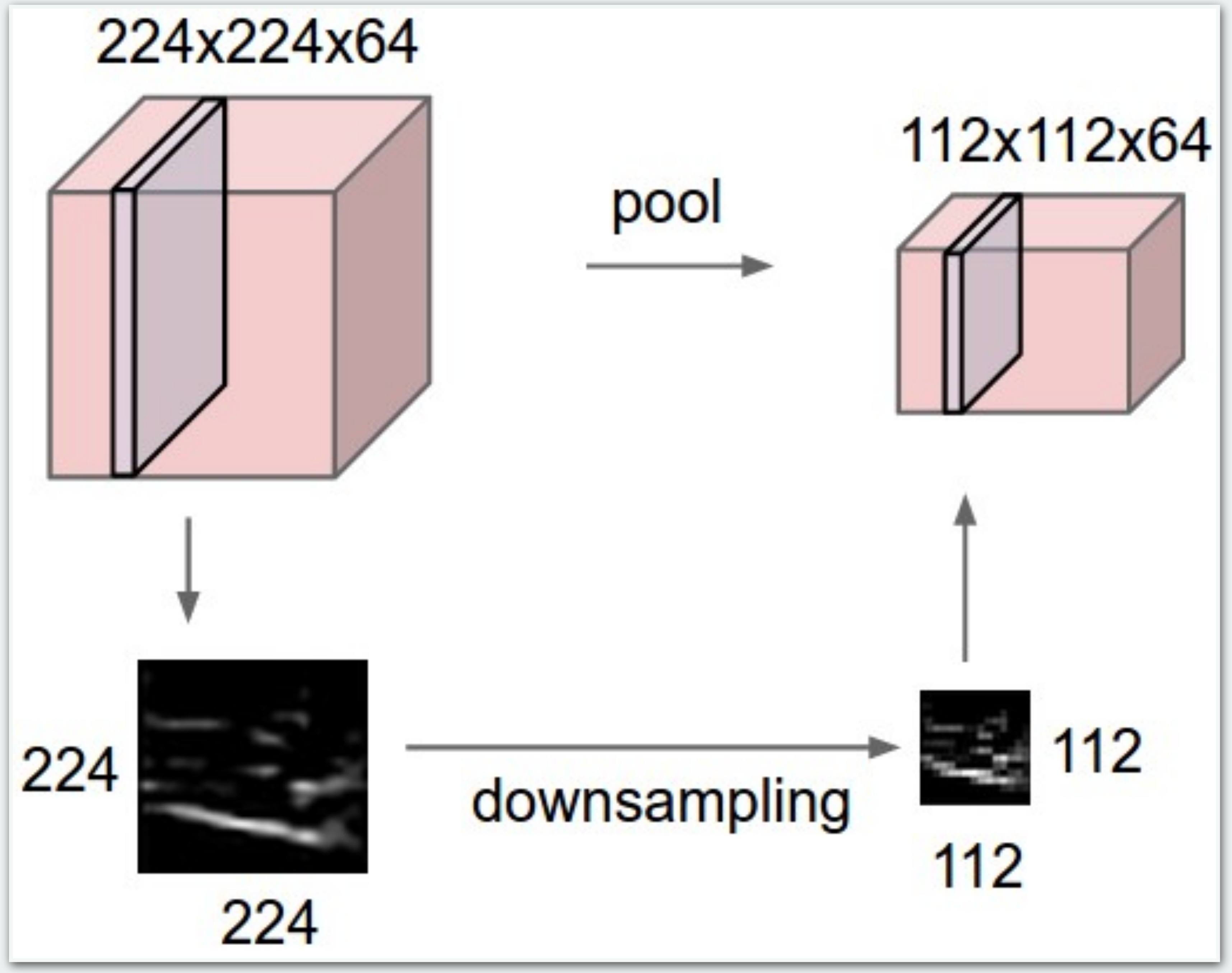


http://cs231n.github.io/convolutional-networks/

Convolutional neural networks

> Pooling

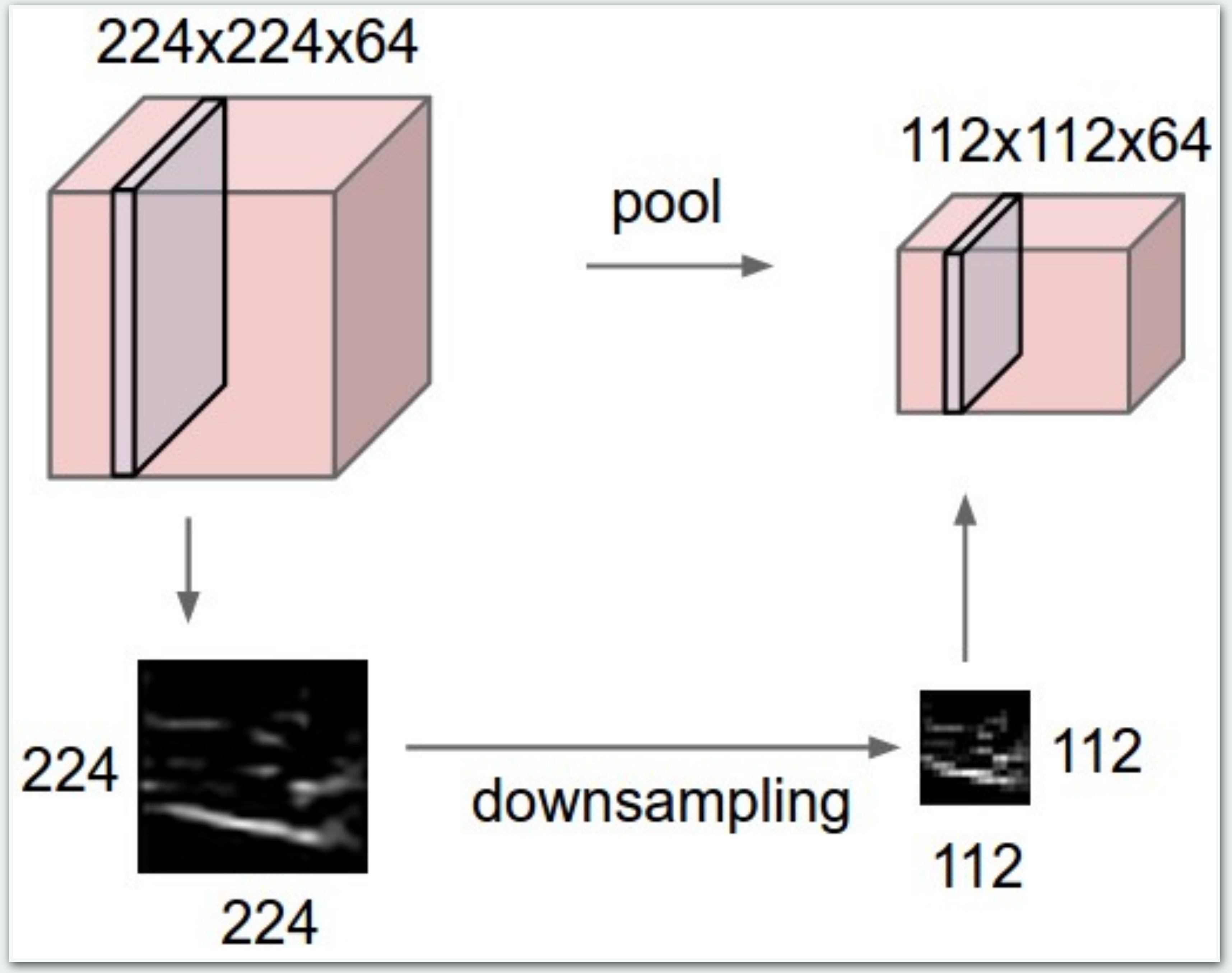
- Method used for downsampling
- Reduces number of parameters and computations
- Lowers width and height of volume by an integer factor
- Preserved depth

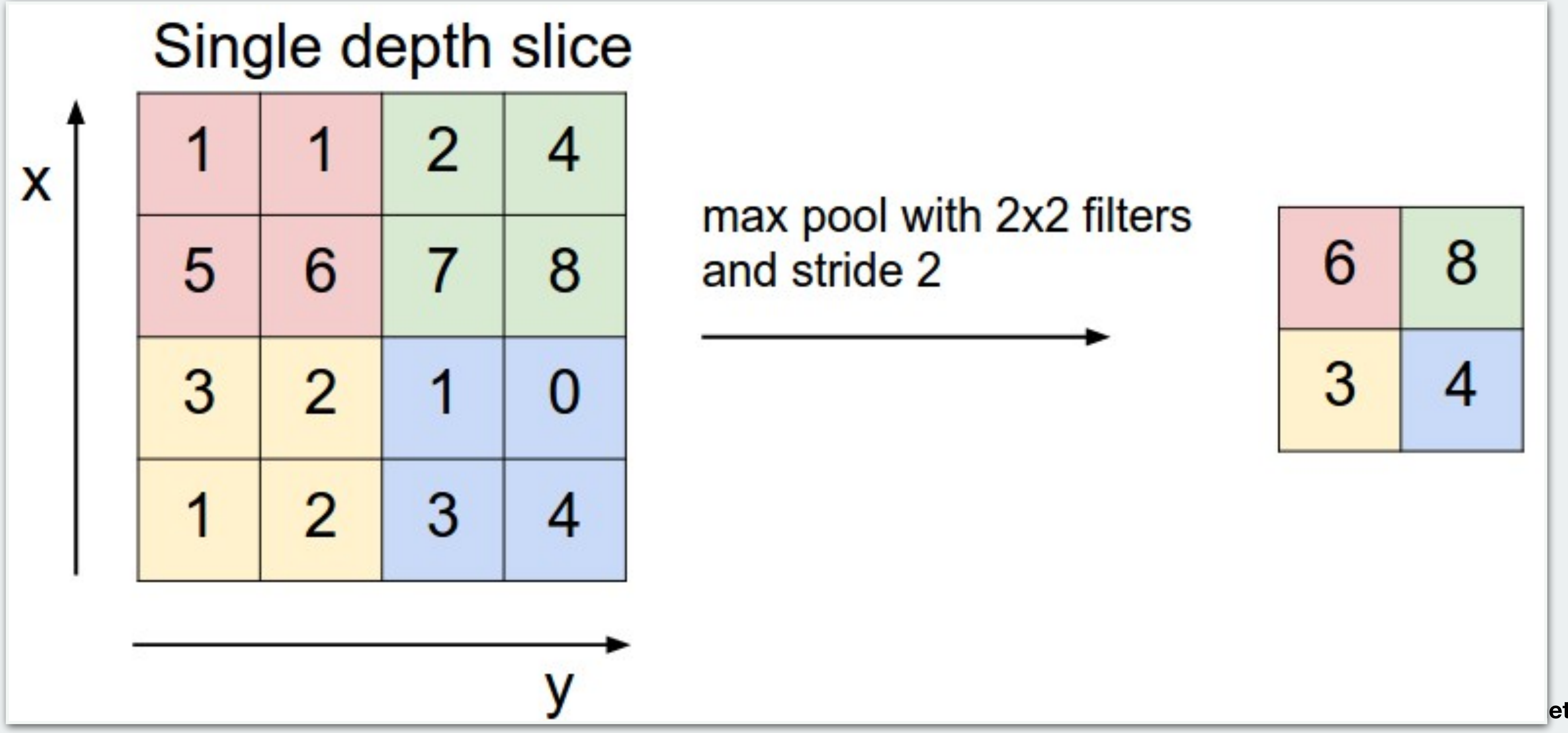


http://cs231n.github.io/convolutional-networks/

Convolutional neural networks
> Pooling

- Method used for downsampling
- Reduces number of parameters and computations
- Lowers width and height of volume by an integer factor
- Preserved depth





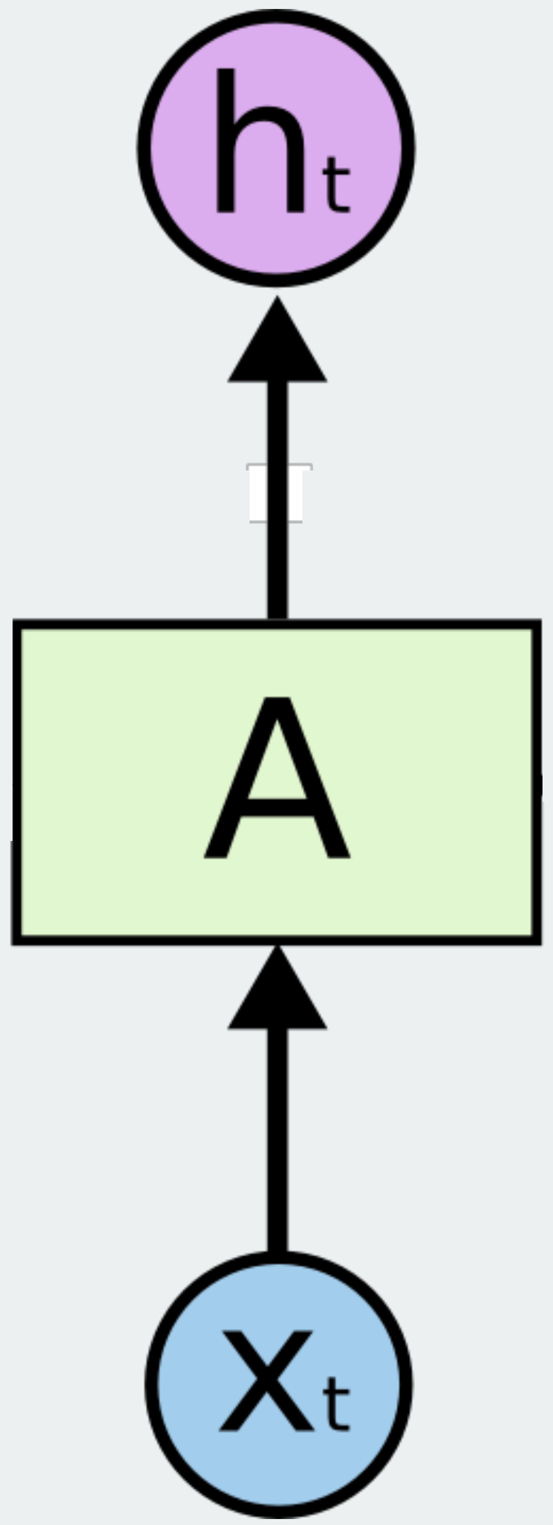
etworks/

Recurrent Neural Networks

THE neural network architecture to use for sequential data

The **problem** with **all** feed forward neural networks

- Input and output must be of hard-assigned dimensions
- The network makes a fixed number of computations
- If input is sequence-like (video, sound, etc.) the network is ignorant to the order of samples

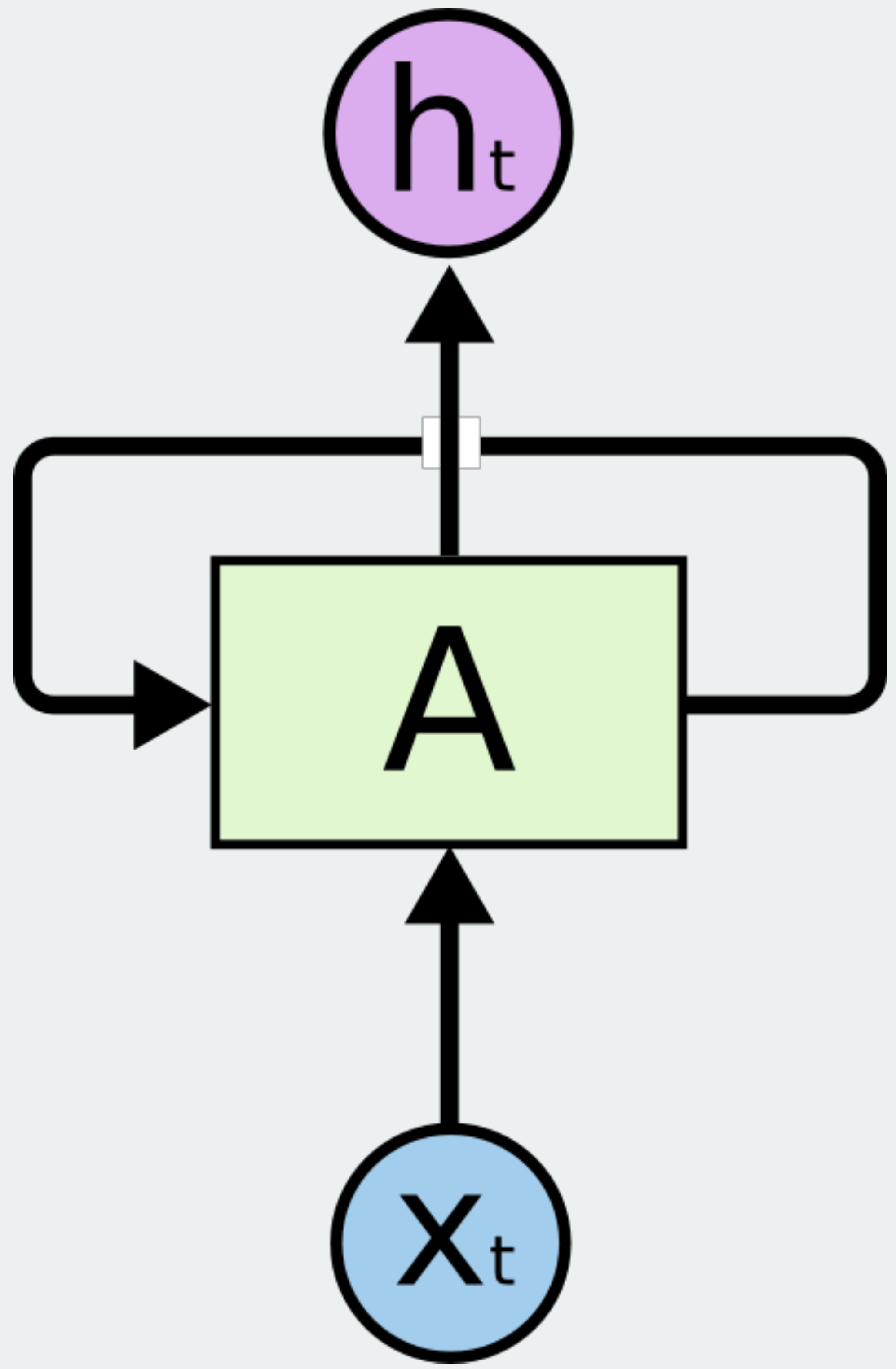


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The **problem** with **all** feed forward neural networks

> **Solution:** *Recurrence!*

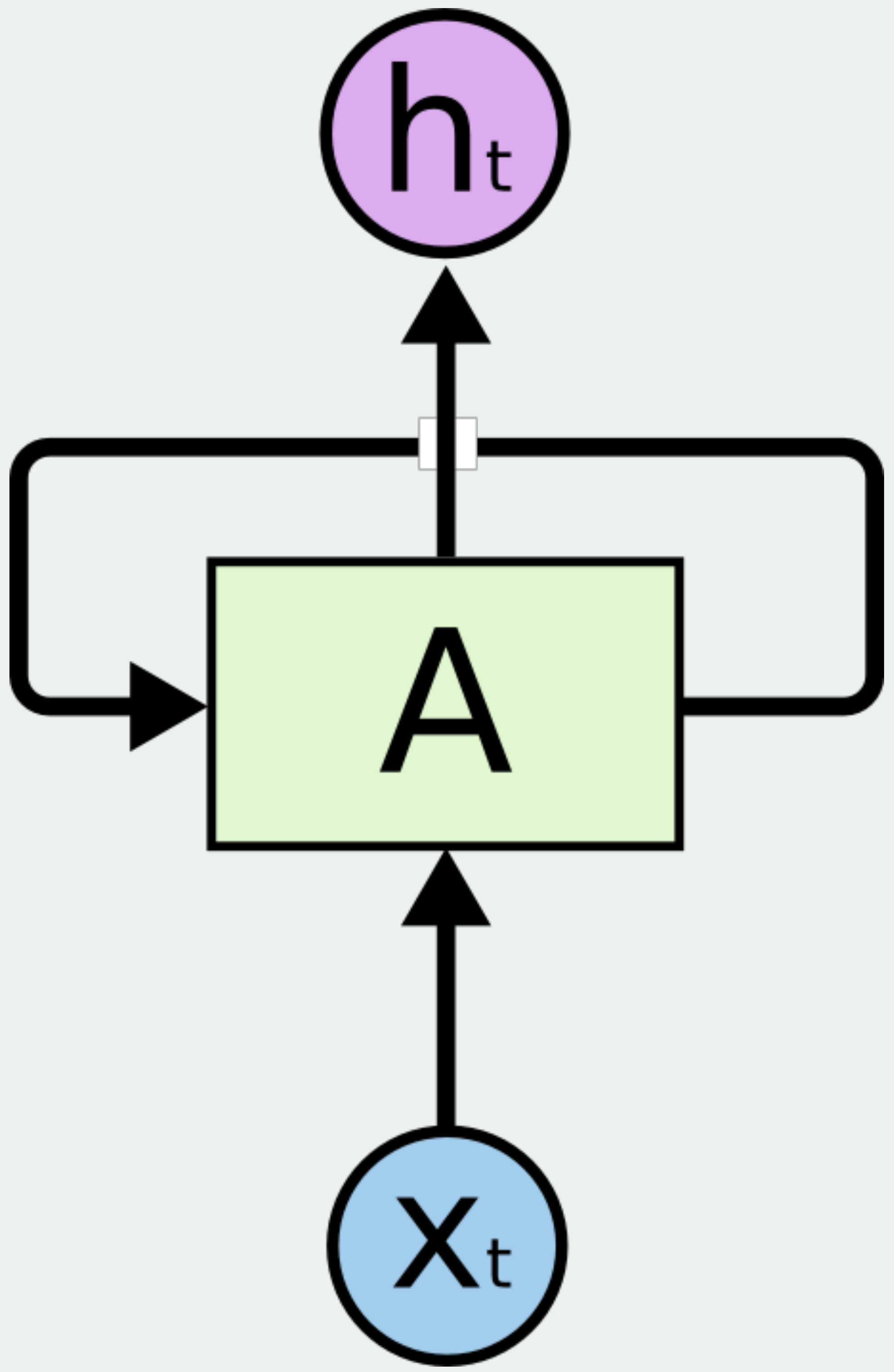
- Input and output must be of hard-assigned dimensions
- The network makes a fixed number of computations
- If input is sequence-like (video, sound, etc.) the network is ignorant to the order of samples



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks

> Fundamental idea



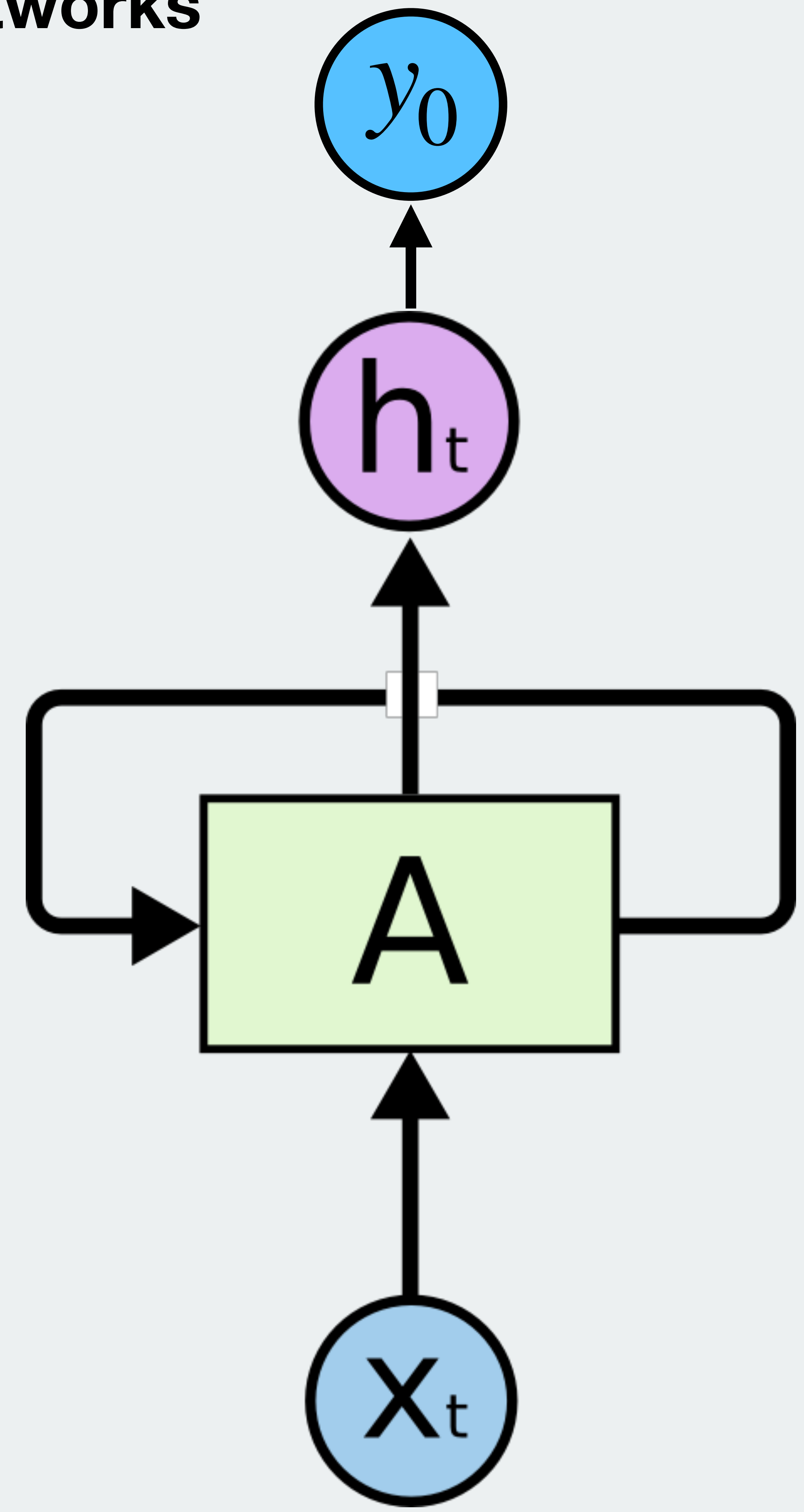
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: W is the same in each iteration

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks

> Fundamental idea



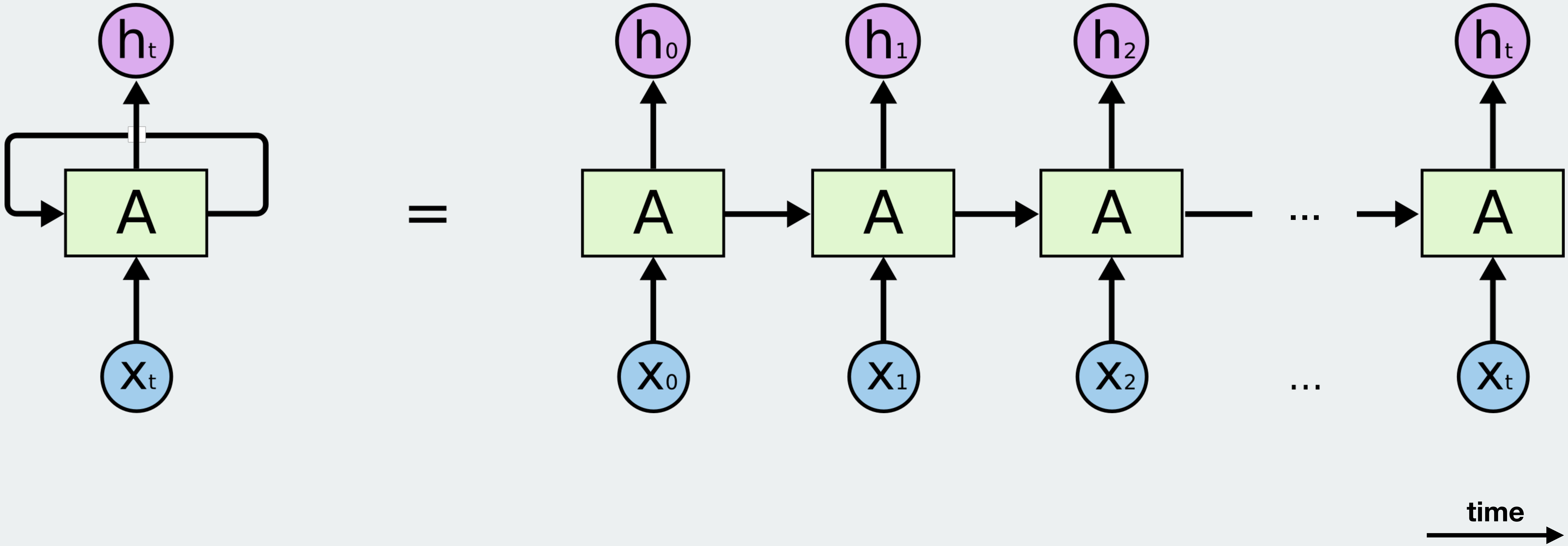
$$h_t = f_W(h_{t-1}, x_t)$$

$$y_t = W_{hy} h_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks

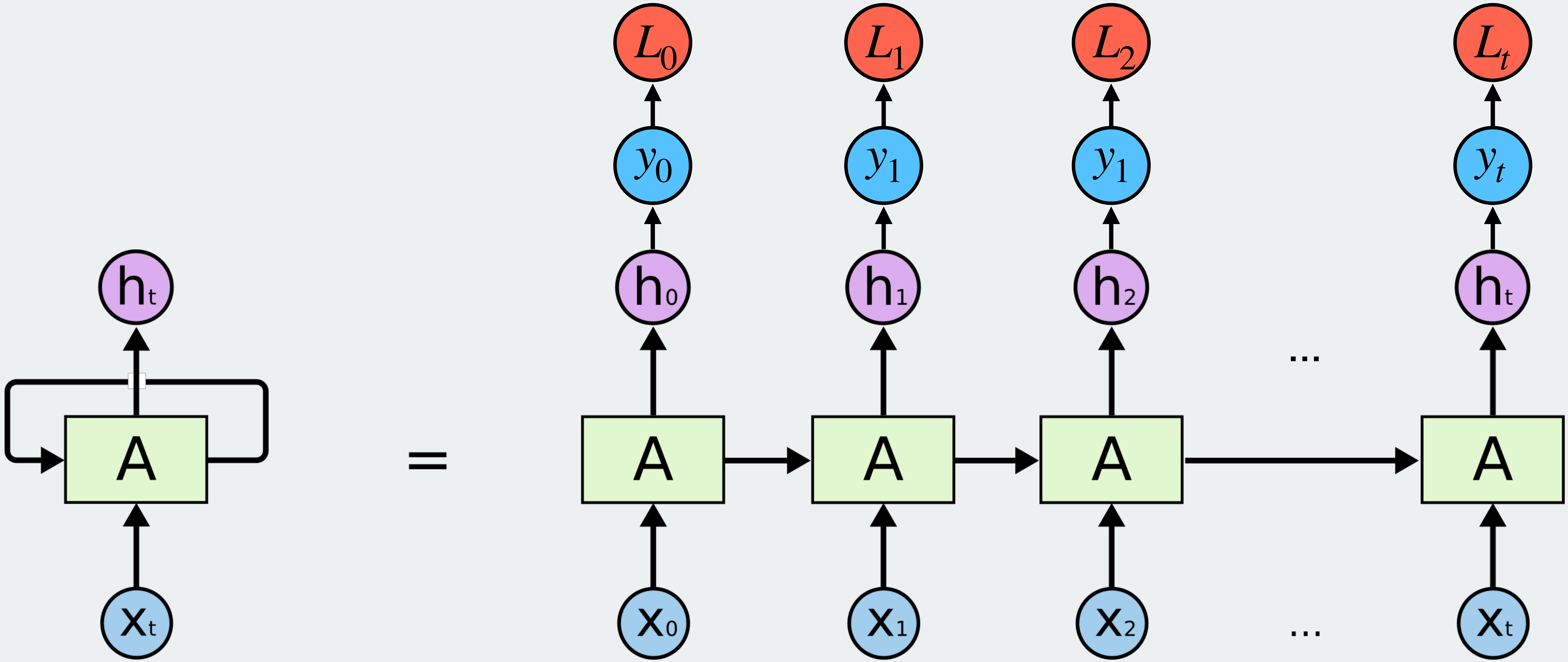
> Unrolled in time



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks

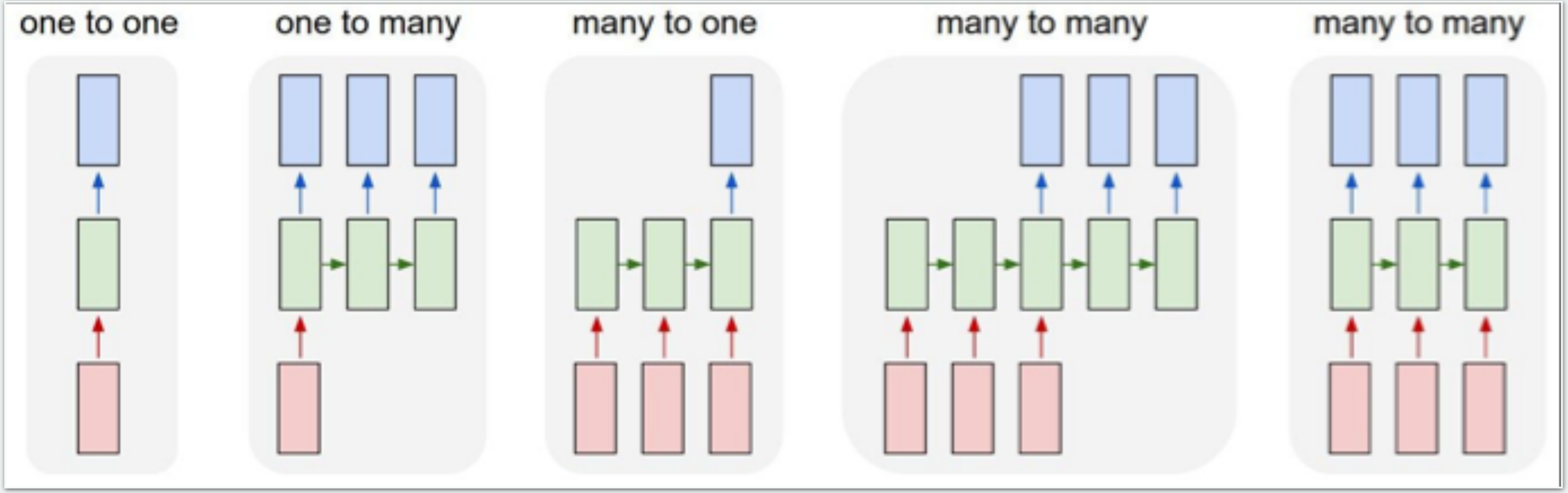
> Backpropagation



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent neural networks

> Ways to process sequential data

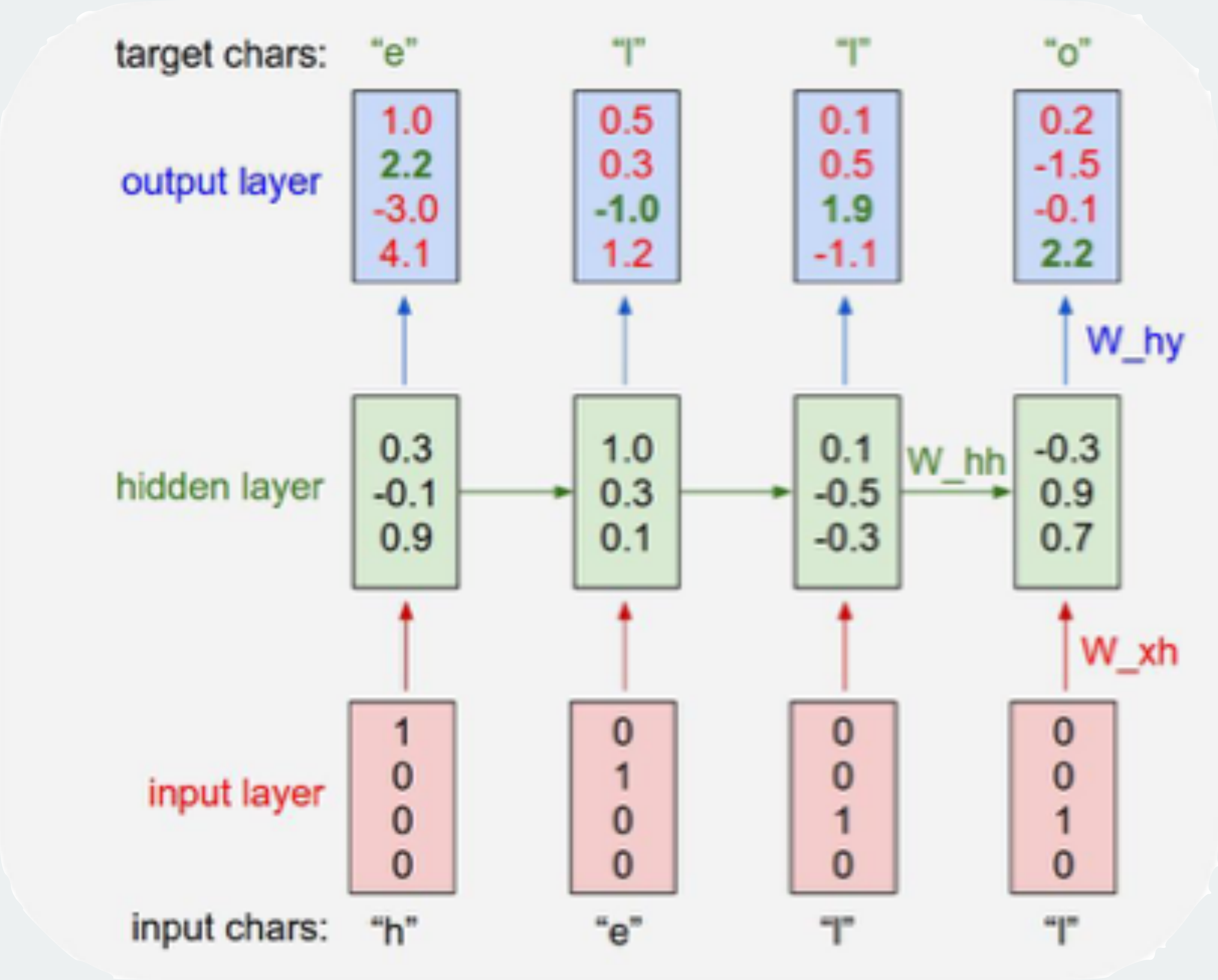


http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

> Example: predicting next character

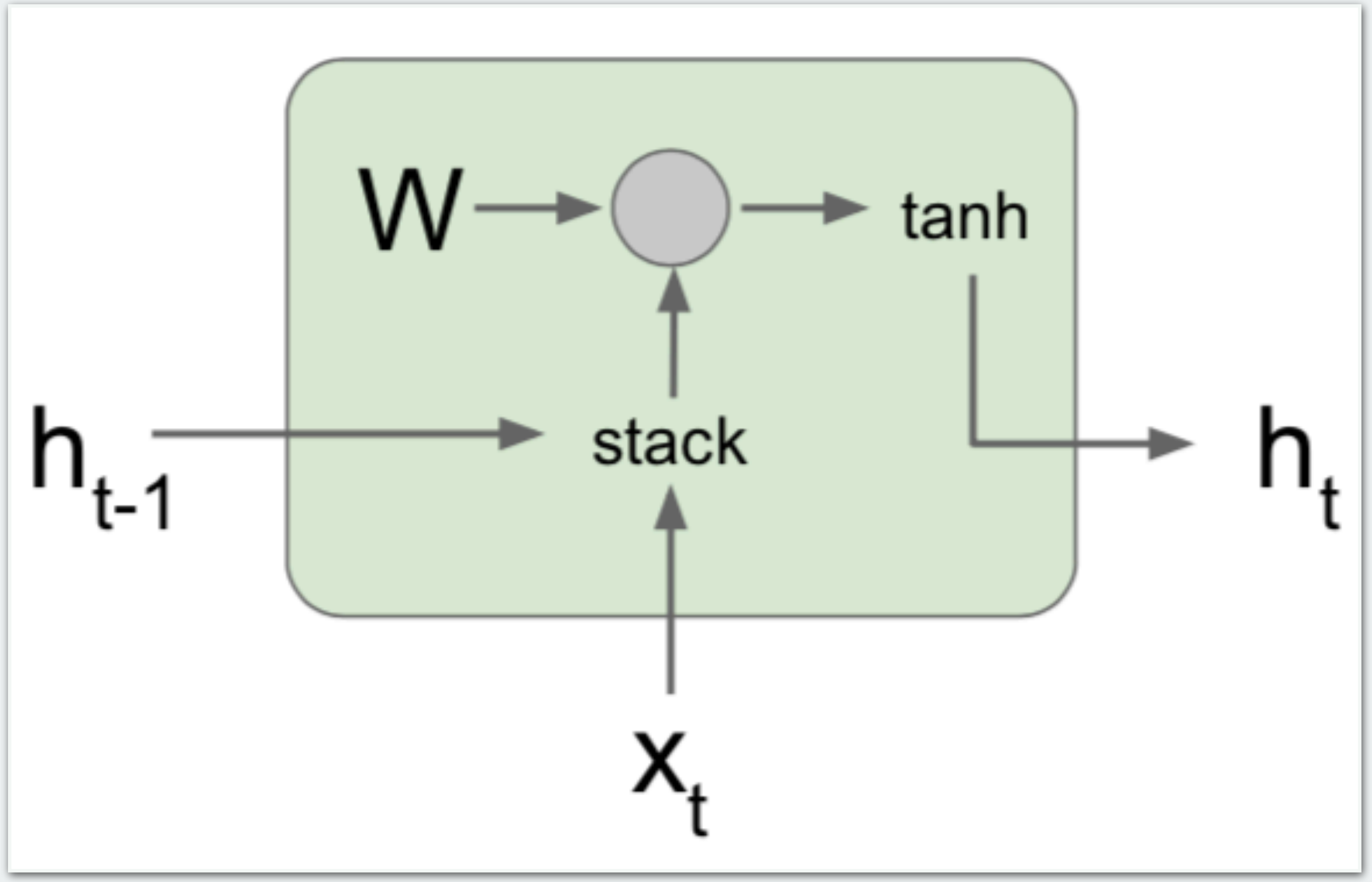
training sequence**: “hello”**



http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

> Vanilla RNN, architecture



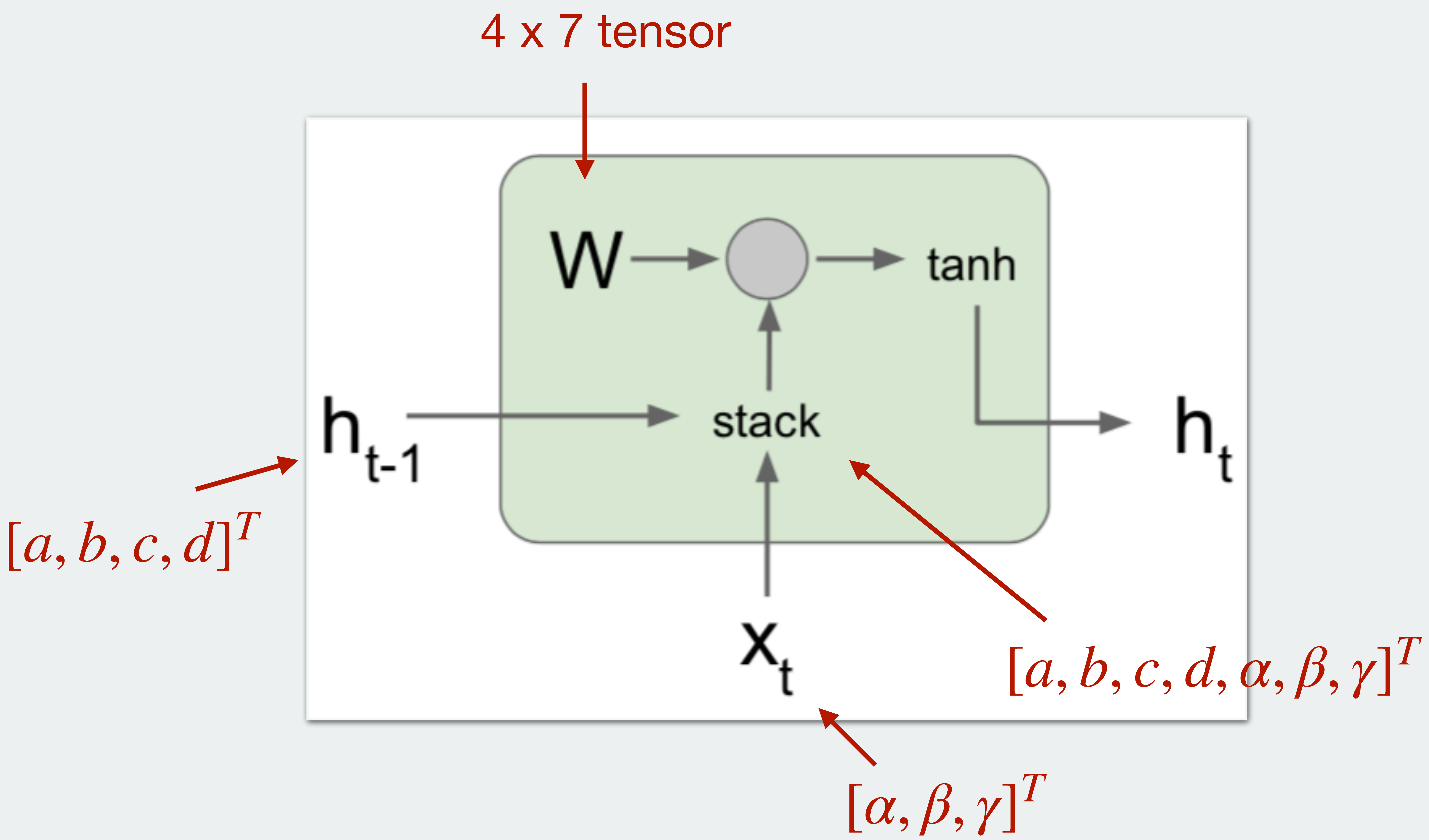
http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

> Vanilla RNN, architecture

4: because dotting (h, t) onto it should result in a new vector with 4 elements

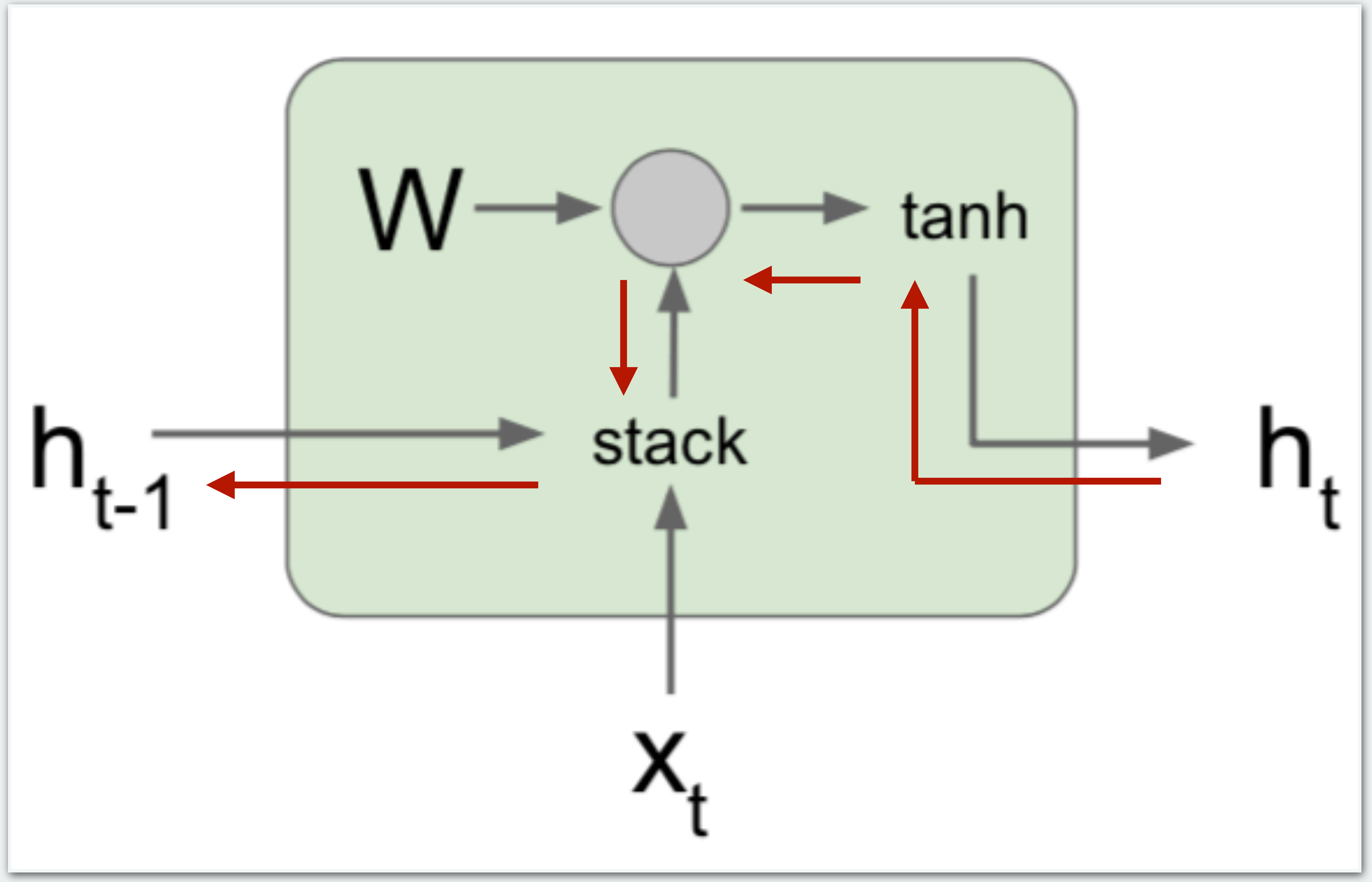
7: because the (h, t) vector we are dotting onto it has 7 elements in it



http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

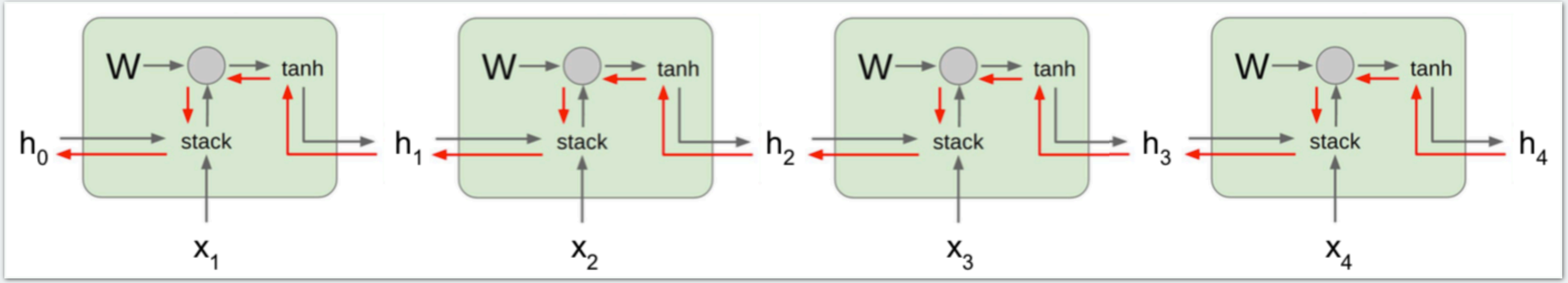
> Vanilla RNN, backpropagation



http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

> Vanilla RNN, backpropagation



Problem: Repeated multiplications by **W** during backpropagation

Leads to: Exploding/vanishing gradients

Solution: Gradient clipping (solves exploding gradients), or **change architecture**

http://cs231n.stanford.edu/syllabus.html

Long Short Term Memory (LSTM)

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

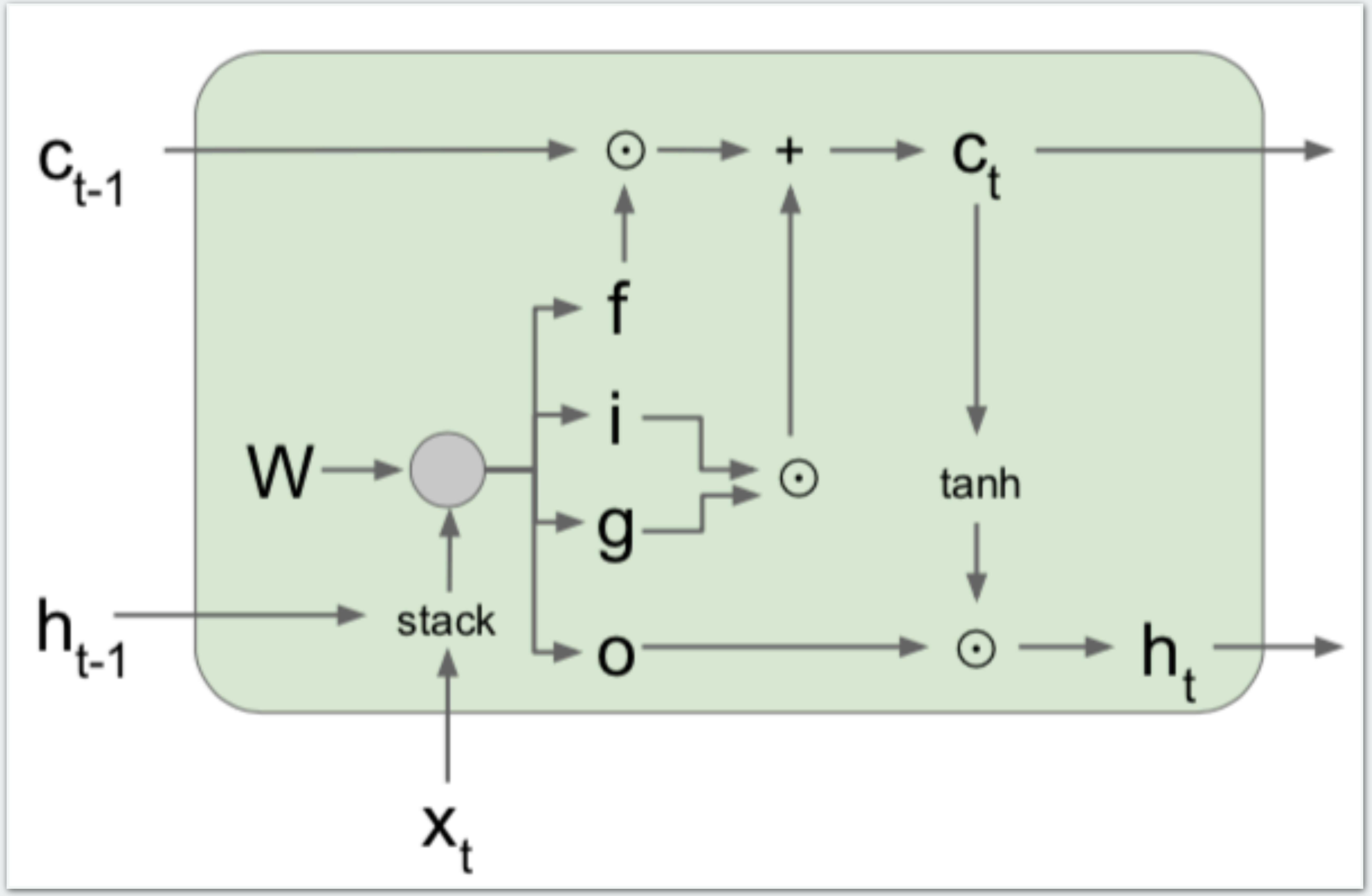
Vanilla RNN

$$h_t = \tanh\left(W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \right)$$

http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks

> Vanilla RNN vs. LSTM



Long Short Term Memory (LSTM)

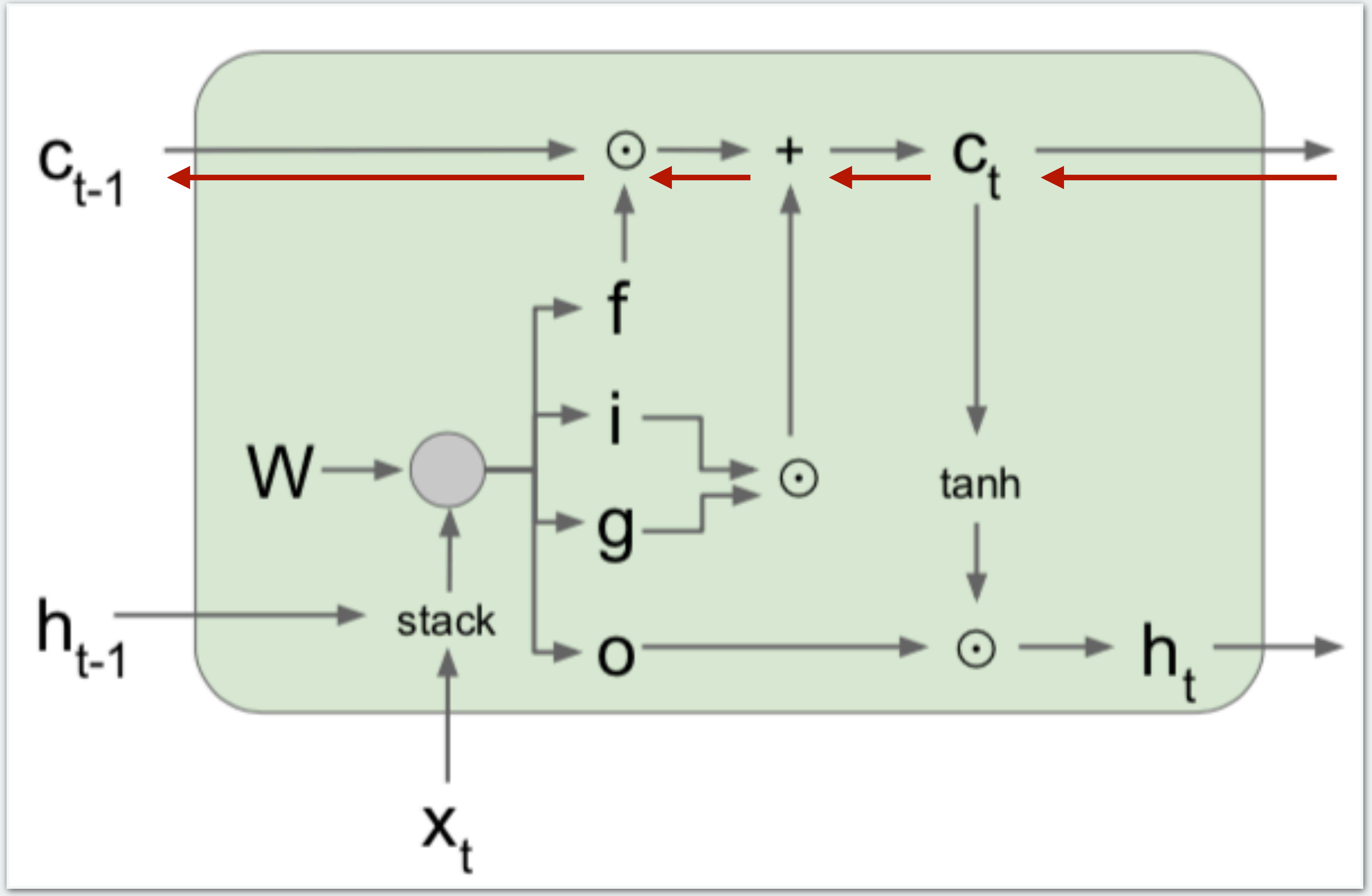
$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/syllabus.html

"Gradient highway": solves vanishing/exploding gradient problems



Long Short Term Memory (LSTM)

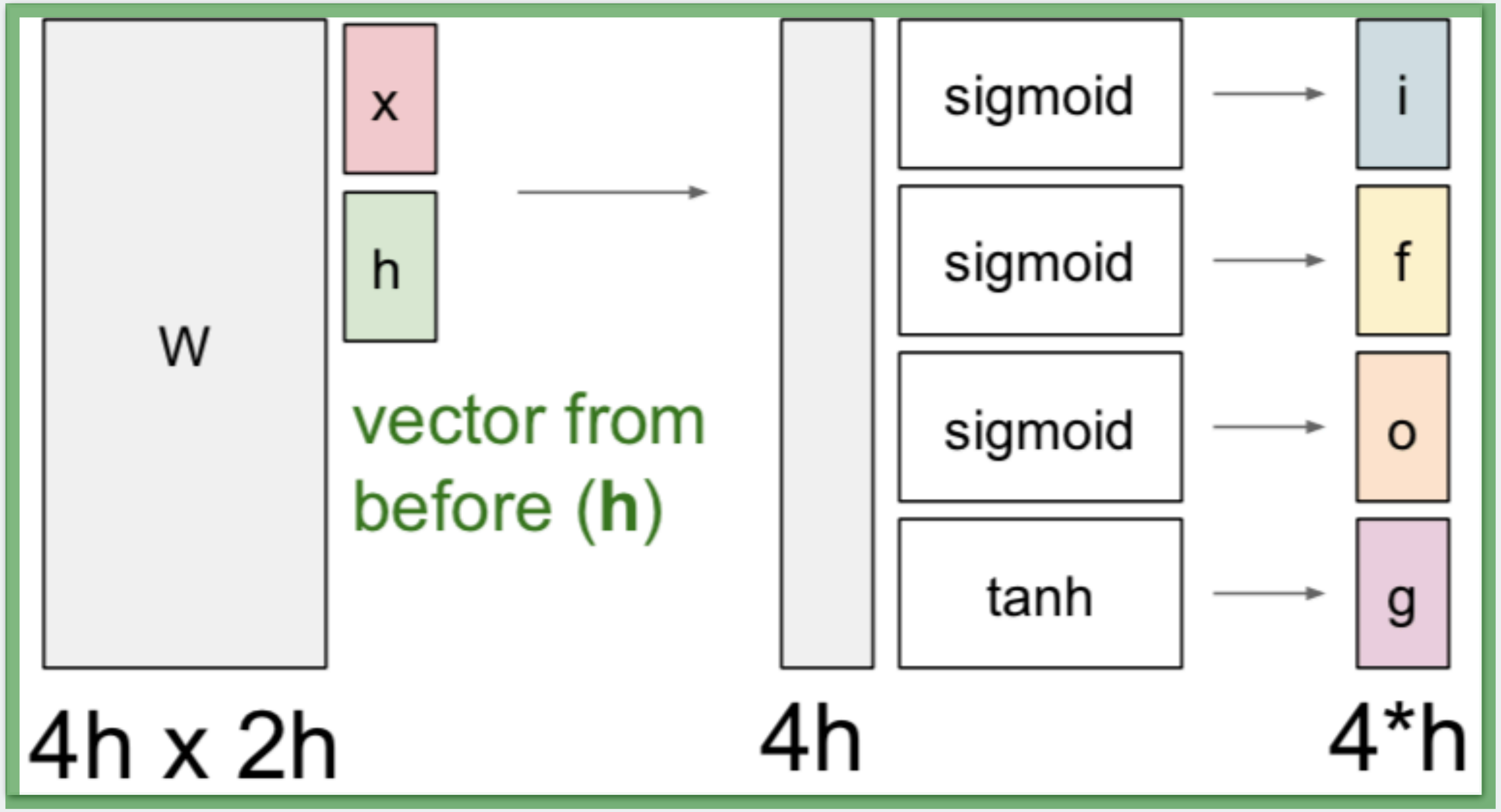
$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/syllabus.html

Recurrent neural networks
> Vanilla RNN vs. LSTM



Long Short Term Memory (LSTM)

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

http://cs231n.stanford.edu/syllabus.html