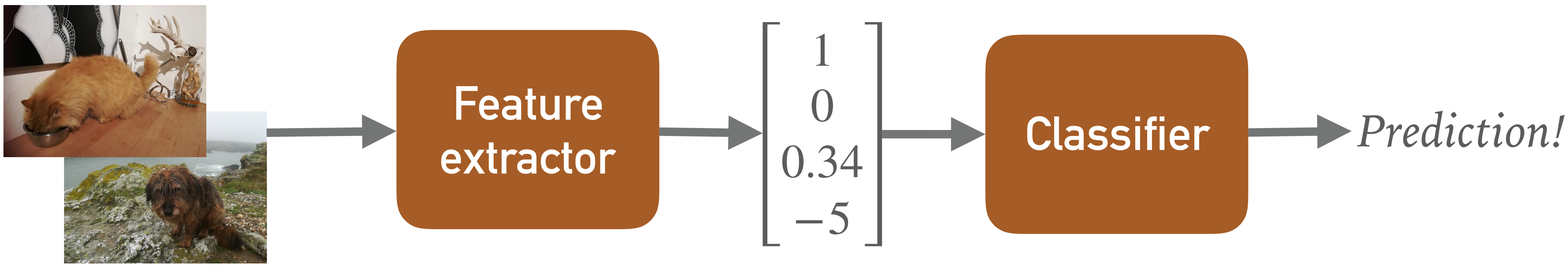
INTRODUCTION À L'APPRENTISSAGE AUTOMATIQUE

Lecture 3 - Polytech

Caio Corro

PRE-DEEP LEARNING ERA

The « old school » machine learning pipeline

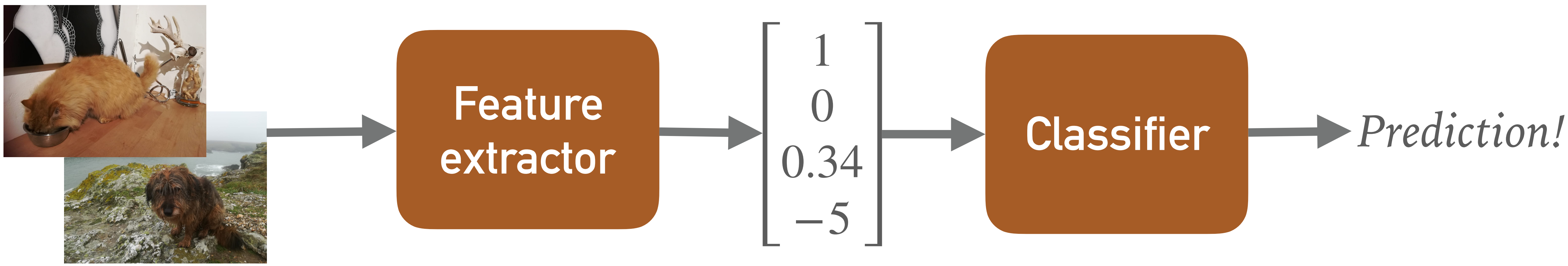


Feature extraction

- Problem dependent
  - Images : SIFT features, invariant to translation, scaling, etc.
  - Text : Stemming, lemmatisation
- Automatic or manual
- Raw data (sometimes…)

PRE-DEEP LEARNING ERA

The « old school » machine learning pipeline



Example of classifiers

- Decision Tree:
  - Make a decision considering a limited number of features
  - Use conjunction of features to make a prediction
- K-nearest neighbors:
  - All features are used and considered equals
- Perceptron/linear classifier:
  - Weight features so they are more or less important to make a decision

DEEP LEARNING

The deep learning « pipeline »

Neural Network

Prediction!

What's the difference?

- No (or limited) feature extraction: use raw data as input!
- Complicated classifier: a neural network is (really) big non-convex function

Neural architecture design

- What kind of parameterized mathematical functions?
  - Image input: Convolutions? or others.
  - Text input: Recurrent neural networks? or others.
- How many parameters?
- How many layers?

Equivariant to translation

Take into account the sequential nature of the input

BUILDING NEURAL NETWORKS

Architecture design

Neural network = complicated parameterized function

- Inductive bias: take into account the data properties to design the architectures
- Time complexity/speed
- Mathematical properties for efficient training:
  differentiability, prevent vanishing/exploding gradient, …

Parameter optimization

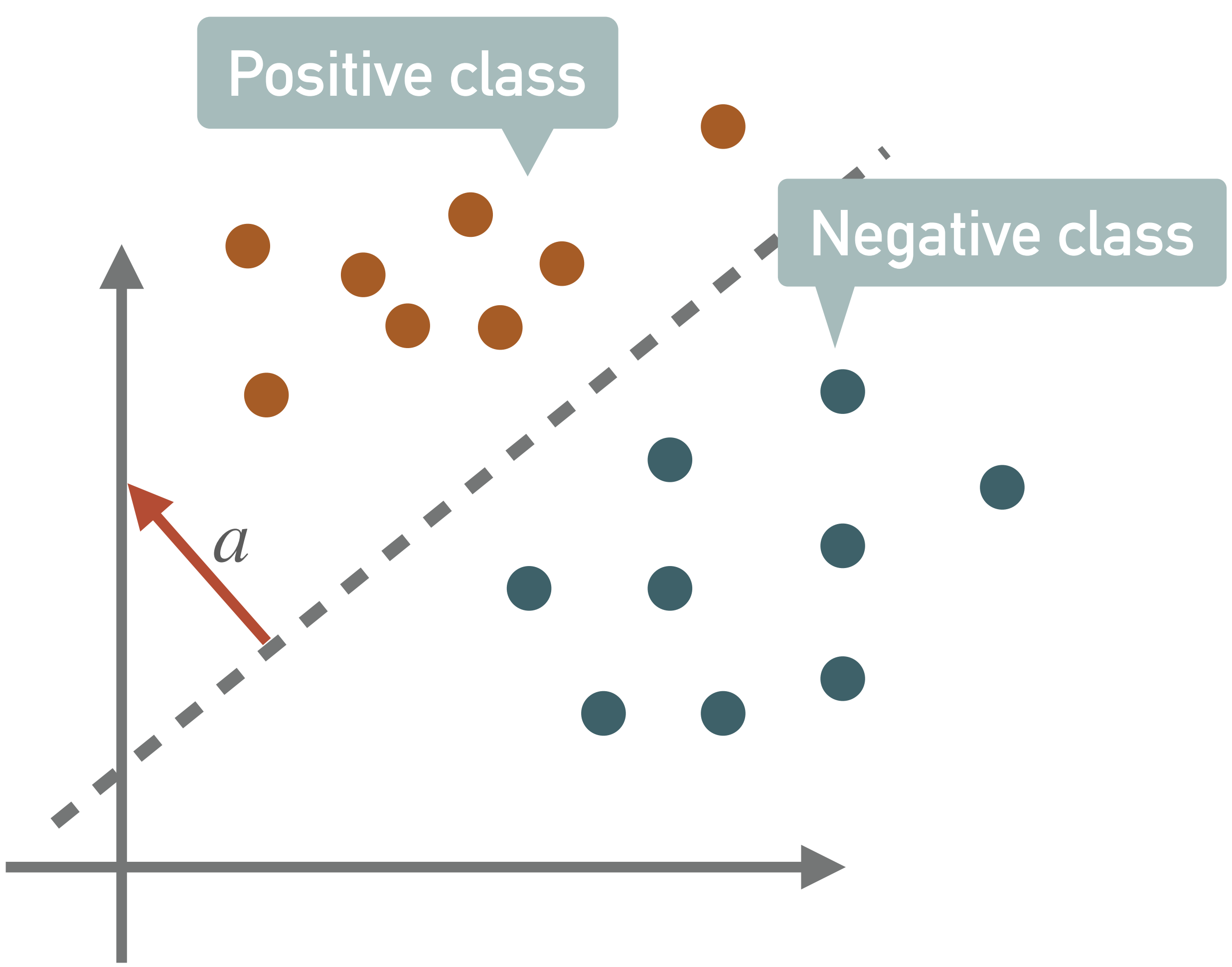
- Efficient optimization algorithms (i.e. first order gradient-based methods)
- Prevent overfitting
- Parallelized training

LINEAR CLASSIFICATION

BINARY LINEAR CLASSIFIER: DEFINITION

Classification function

- In general: $f_\theta : \mathcal{X} \to \mathcal{Y}$
- Binary case: $f_\theta : \mathbb{R}^n \to \{-1, 1\}$



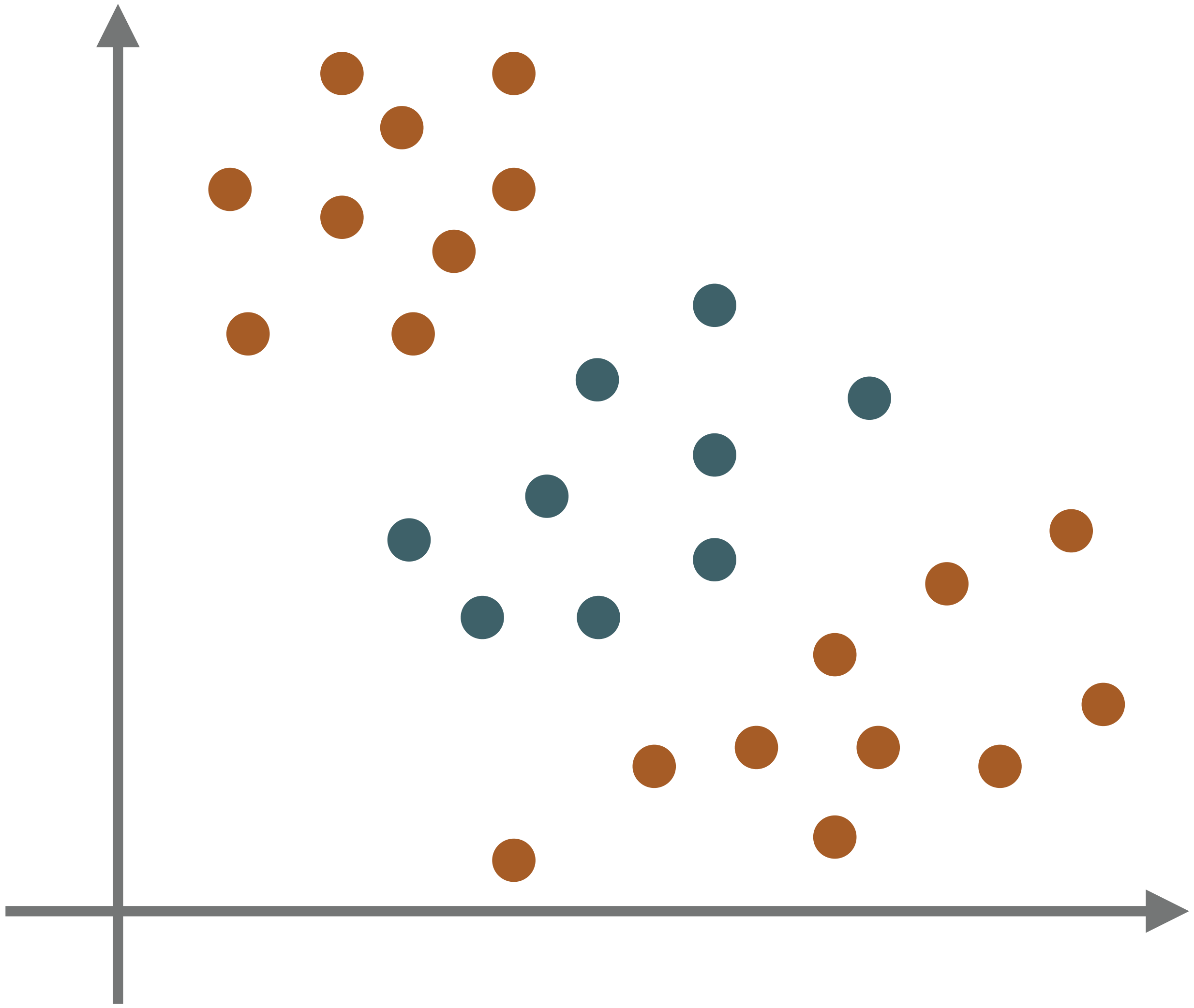
Perceptron

- Let the parameters be $\theta = \{a, b\}$
- Classification function:

$$f_\theta(x) = \begin{cases} -1 & \mathit{if} \quad a^\top x + b \leq 0, \\ 1 & \mathit{if} \quad a^\top x + b > 0. \end{cases}$$

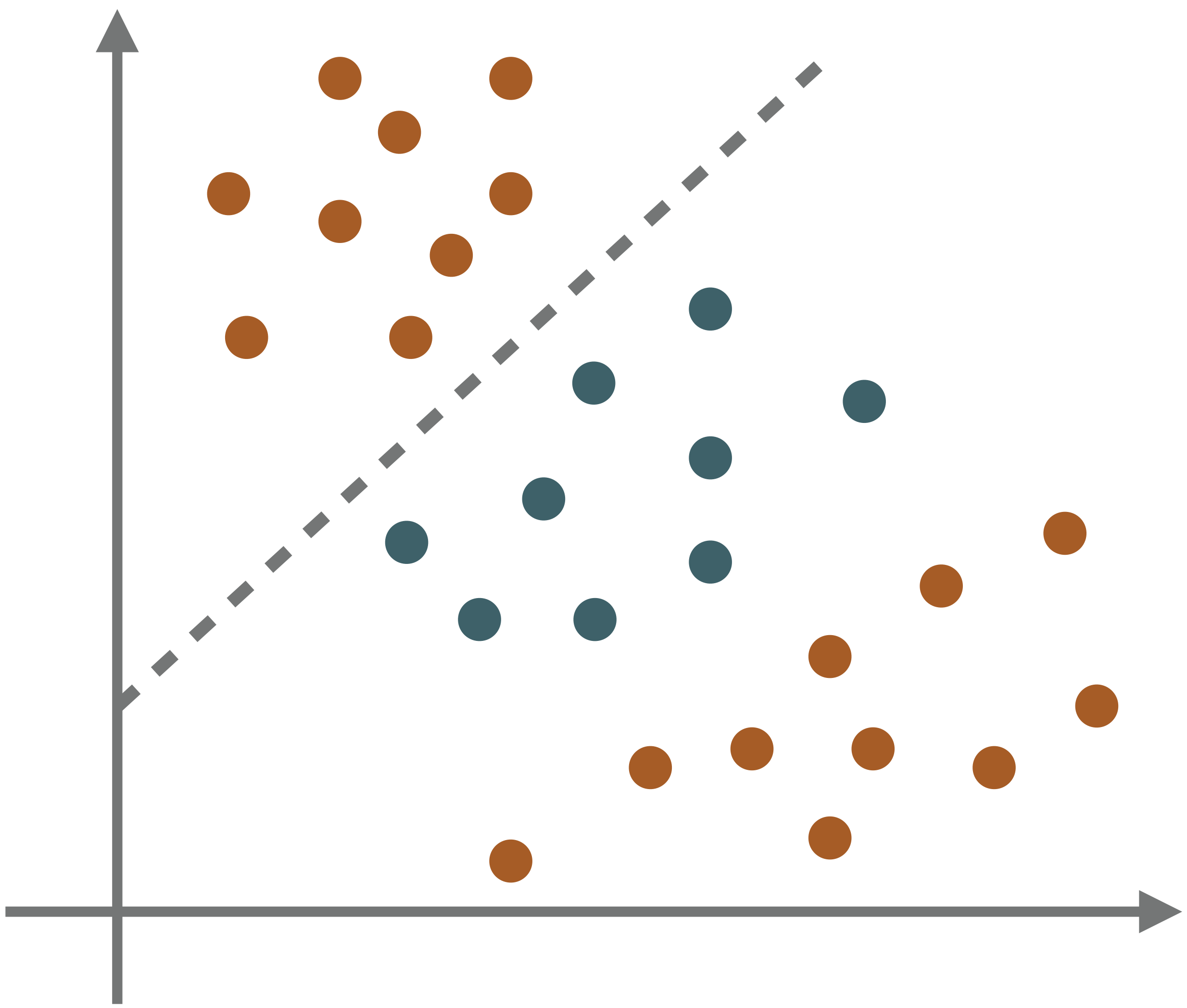
PROBLEMATIC CASES

- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



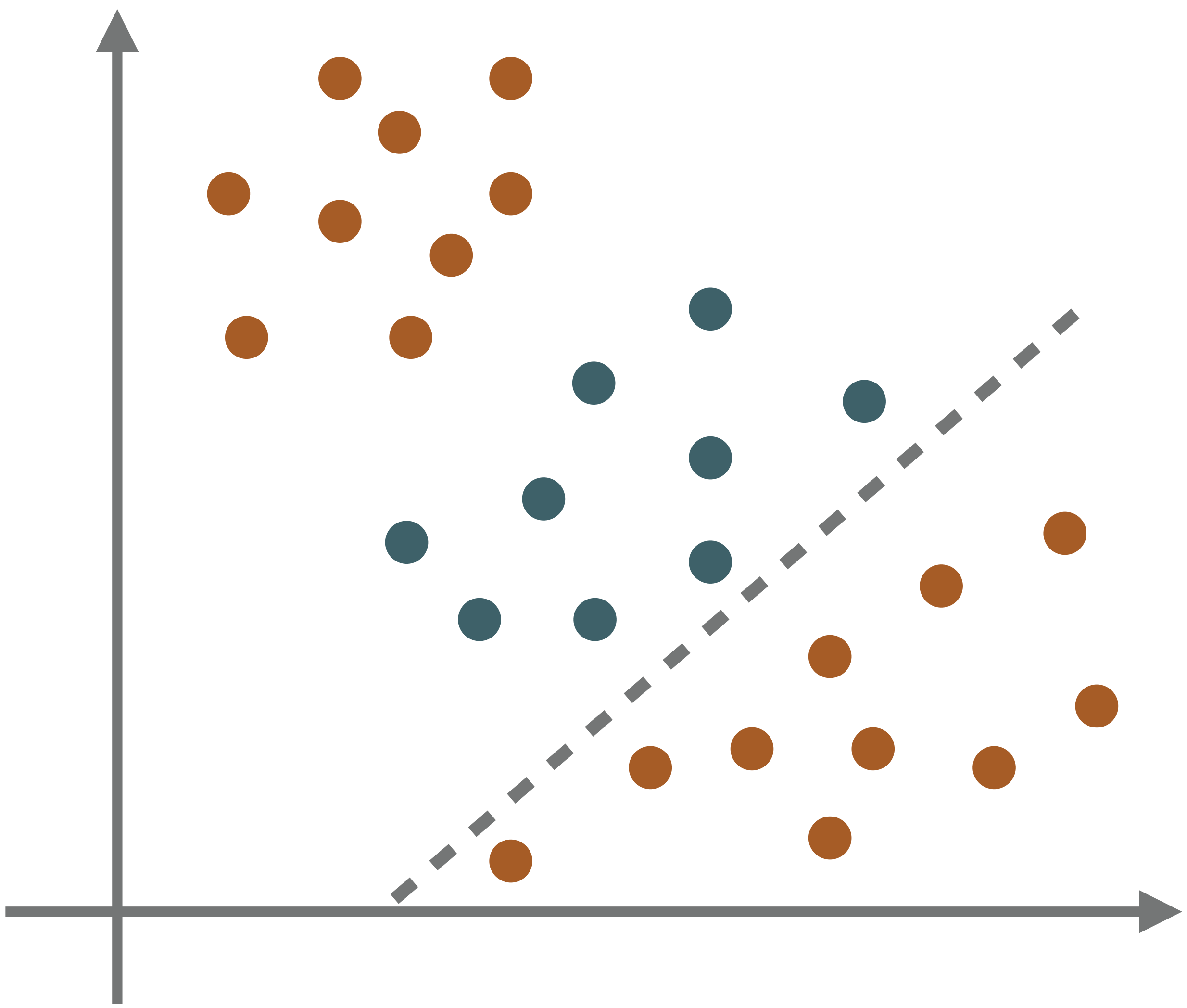
PROBLEMATIC CASES

- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



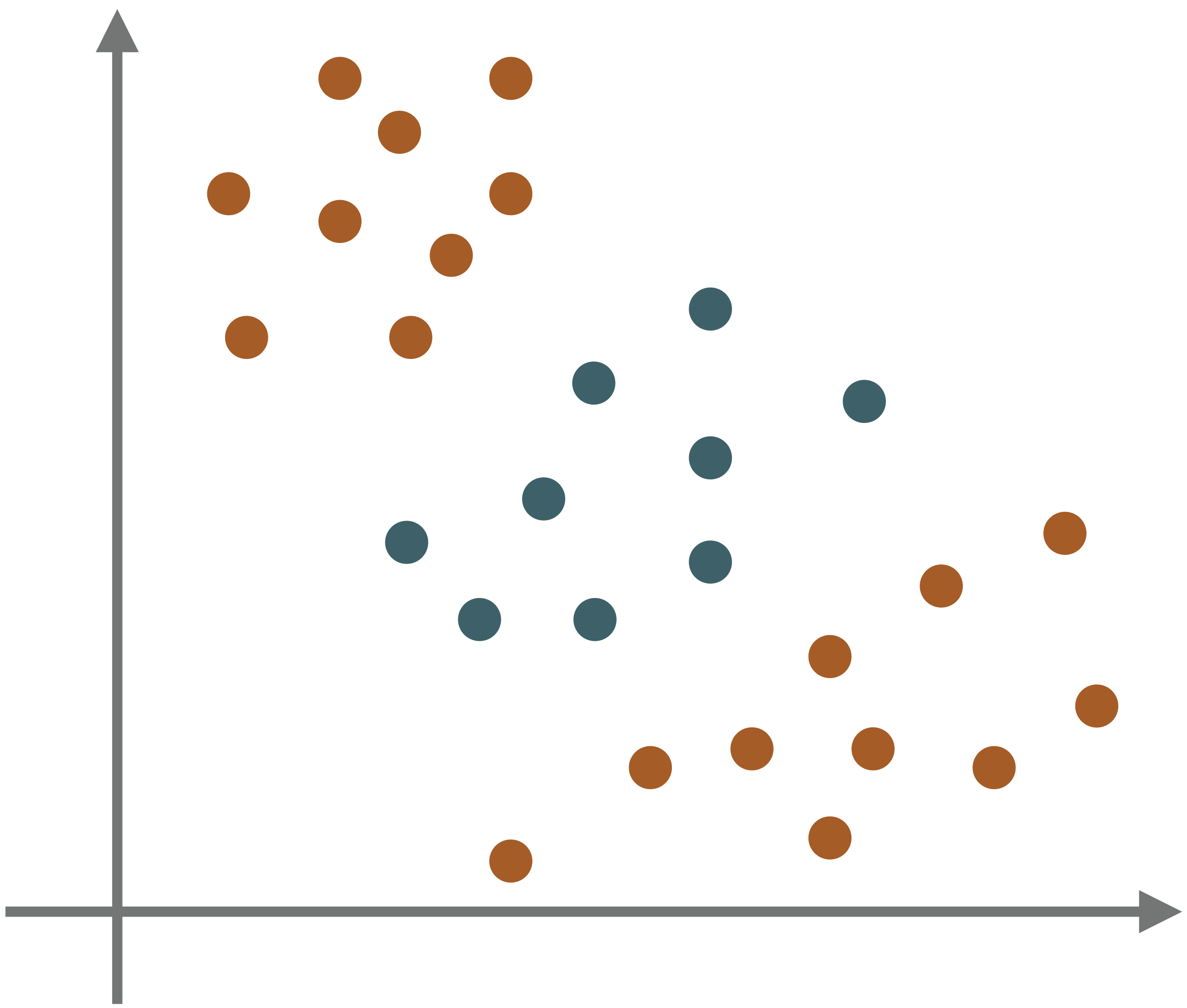
PROBLEMATIC CASES

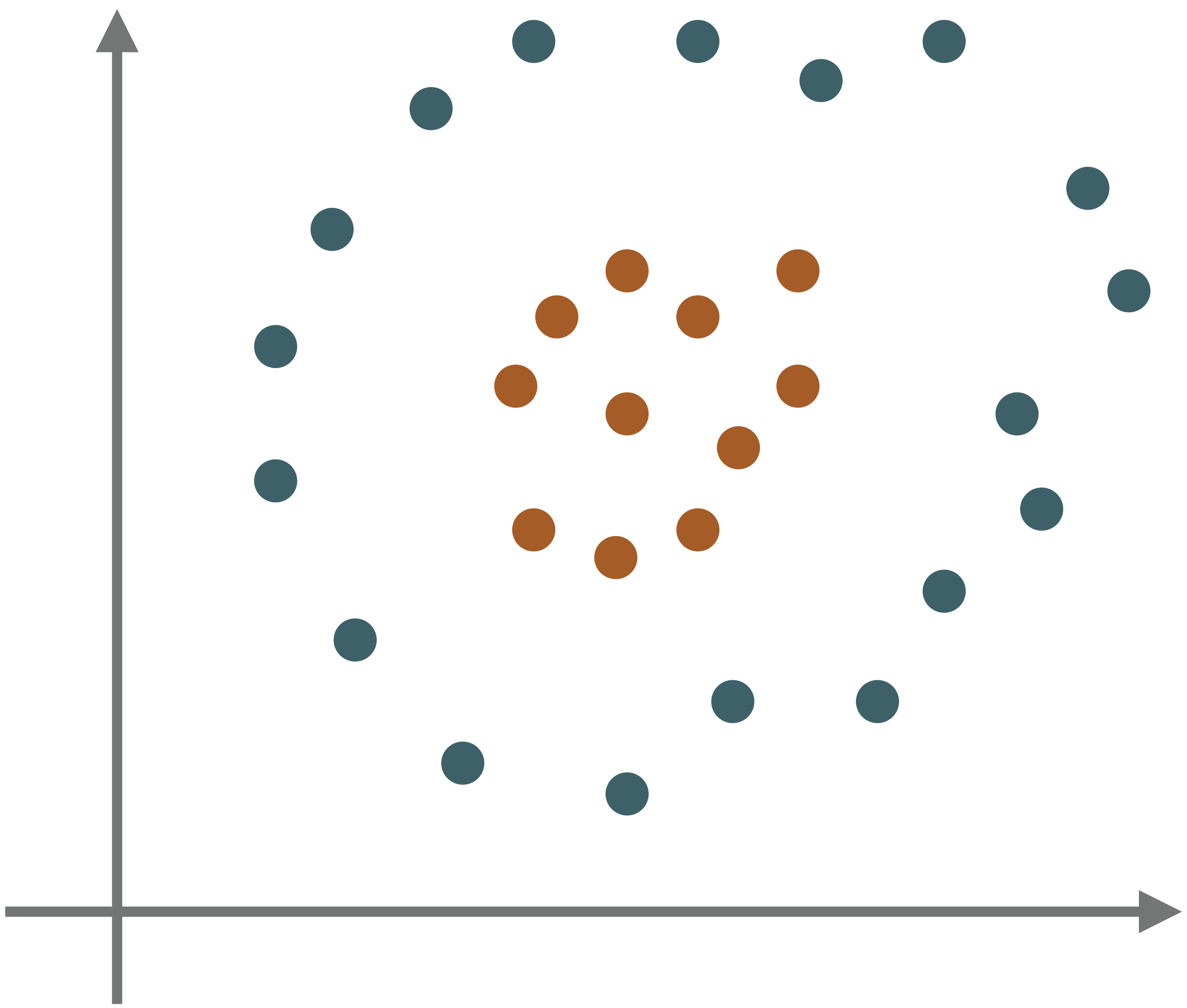
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**



PROBLEMATIC CASES

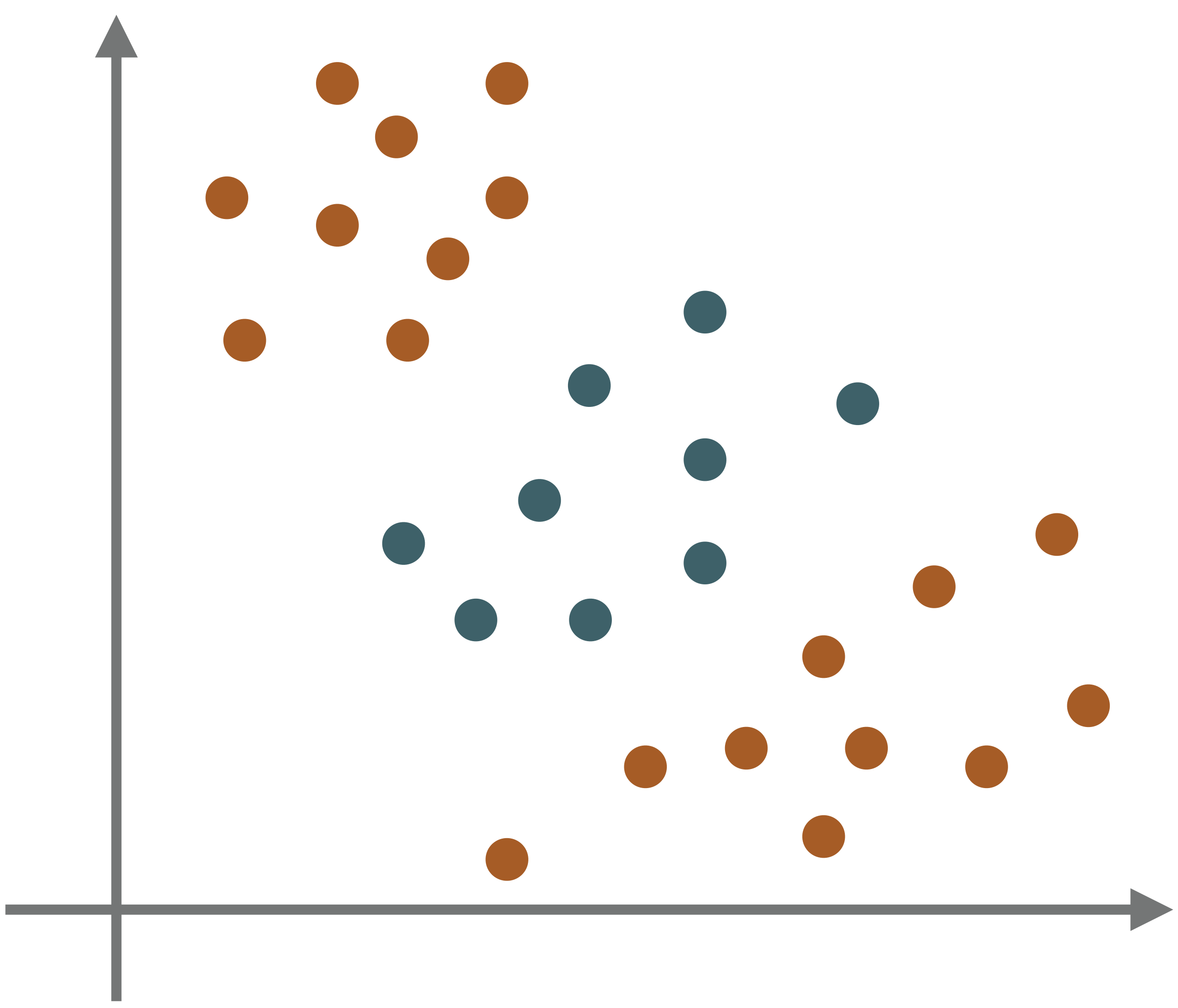
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**

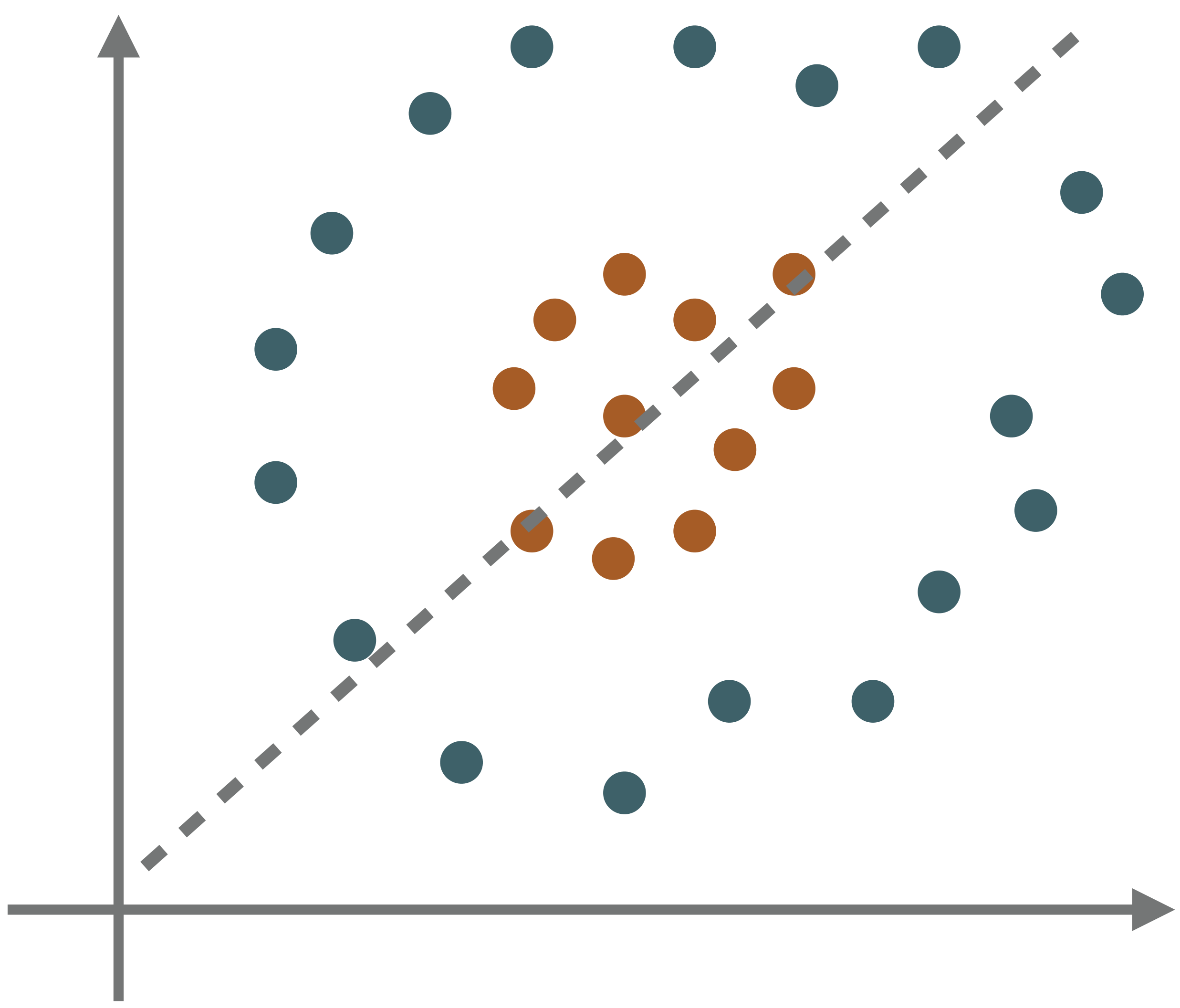




PROBLEMATIC CASES

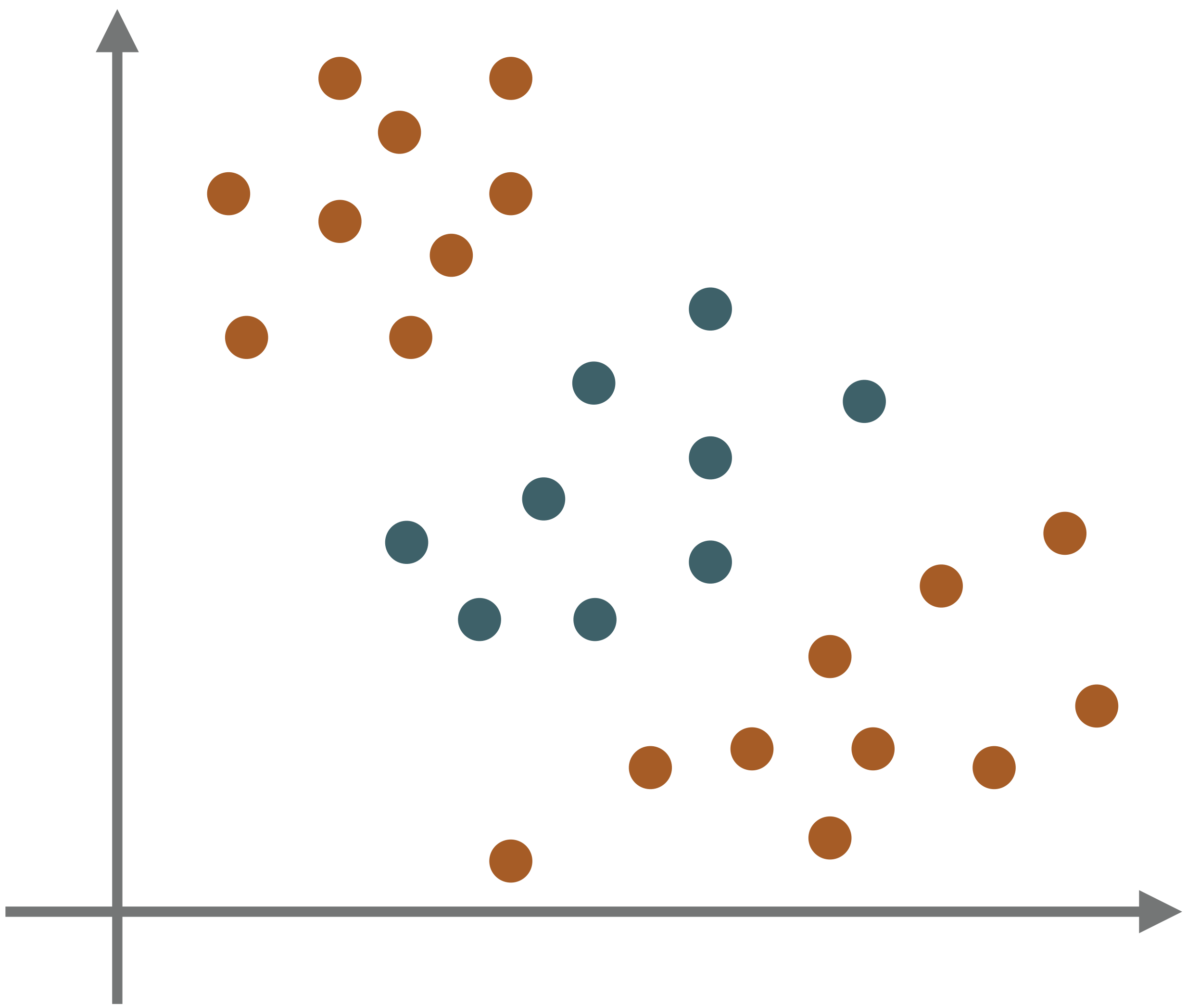
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**

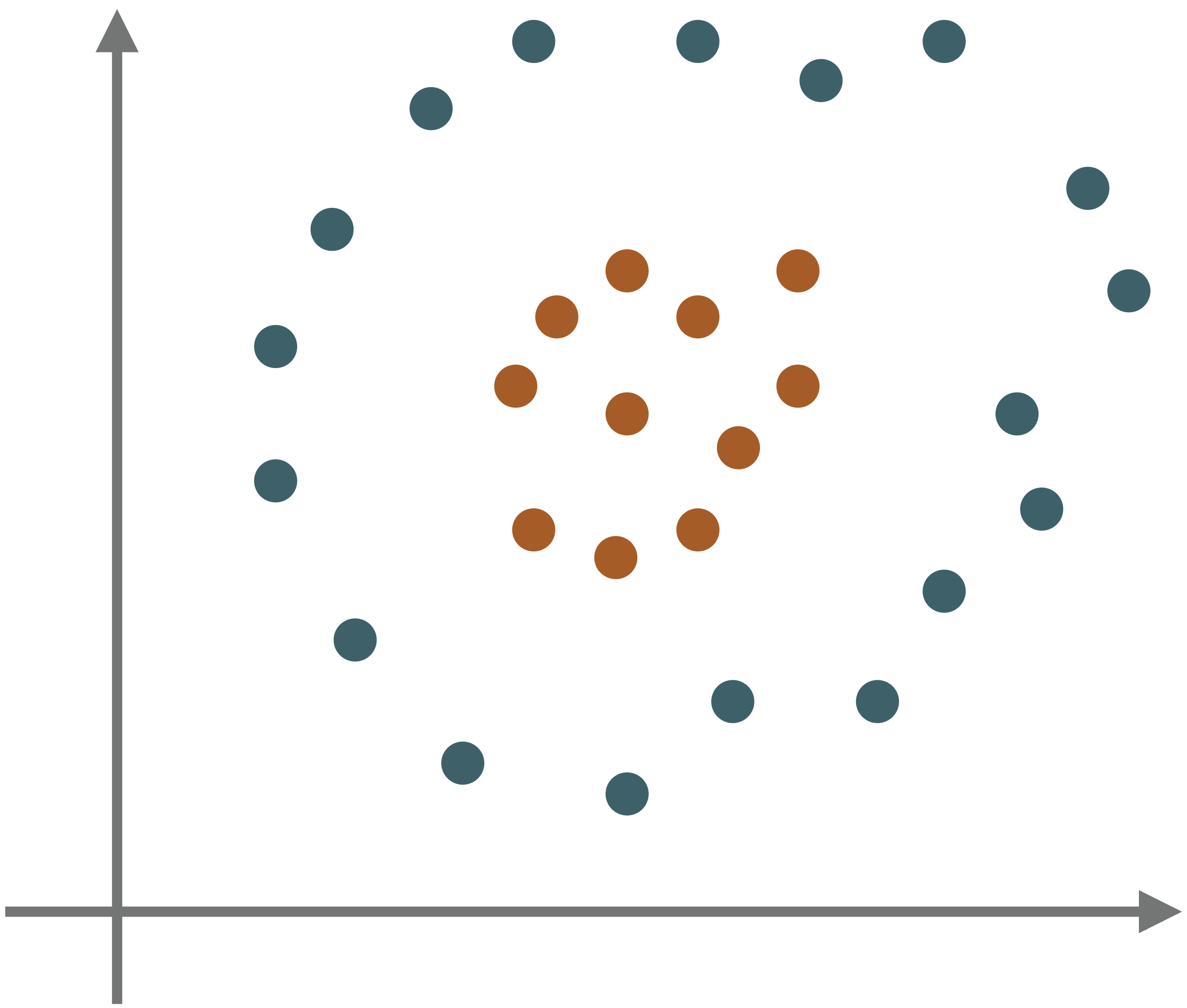




PROBLEMATIC CASES

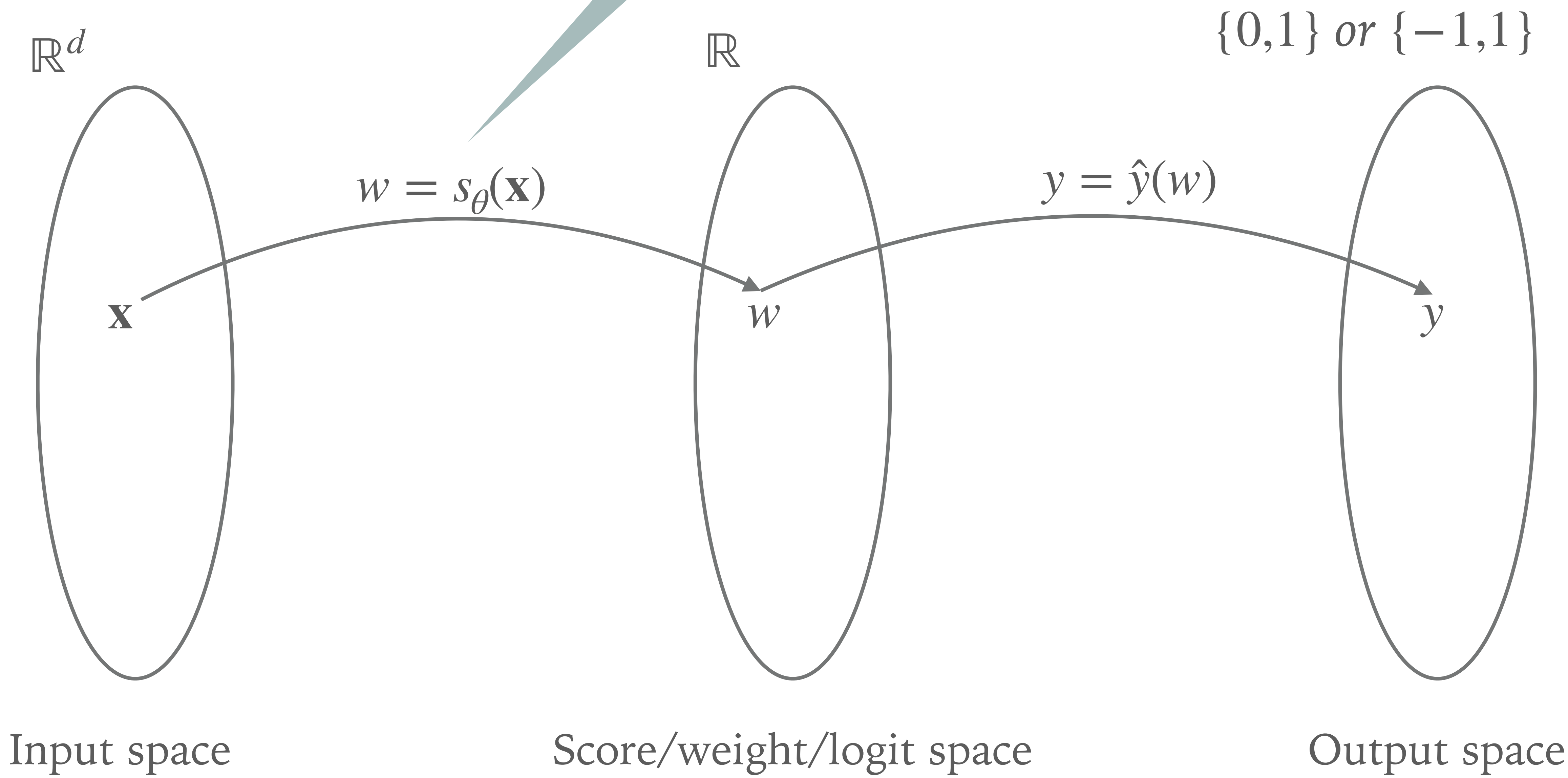
- Can we always find a hyperplane that separate classes? **NO**
- Can we characterize formally in which cases we can? **YES**





BINARY CLASSIFICATION

Can we replace the scoring function by something "more complicated"?



MULTI-LAYER PERCEPTRON

MAIN IDEA

Classifier

Parameterized function $f_\theta : \mathscr{X} \to \mathscr{Y}$

Feature space

Score/output space

Parameters

How to deal with non-separable inputs?

- Manually transform the inputs :(
- Learn automatically a transformation?

Intuition behind multi-layer perceptrons

- Compute « latent » hidden representations so that classes are linearly separable
- Use non-linear activation units so the transformation is not convex

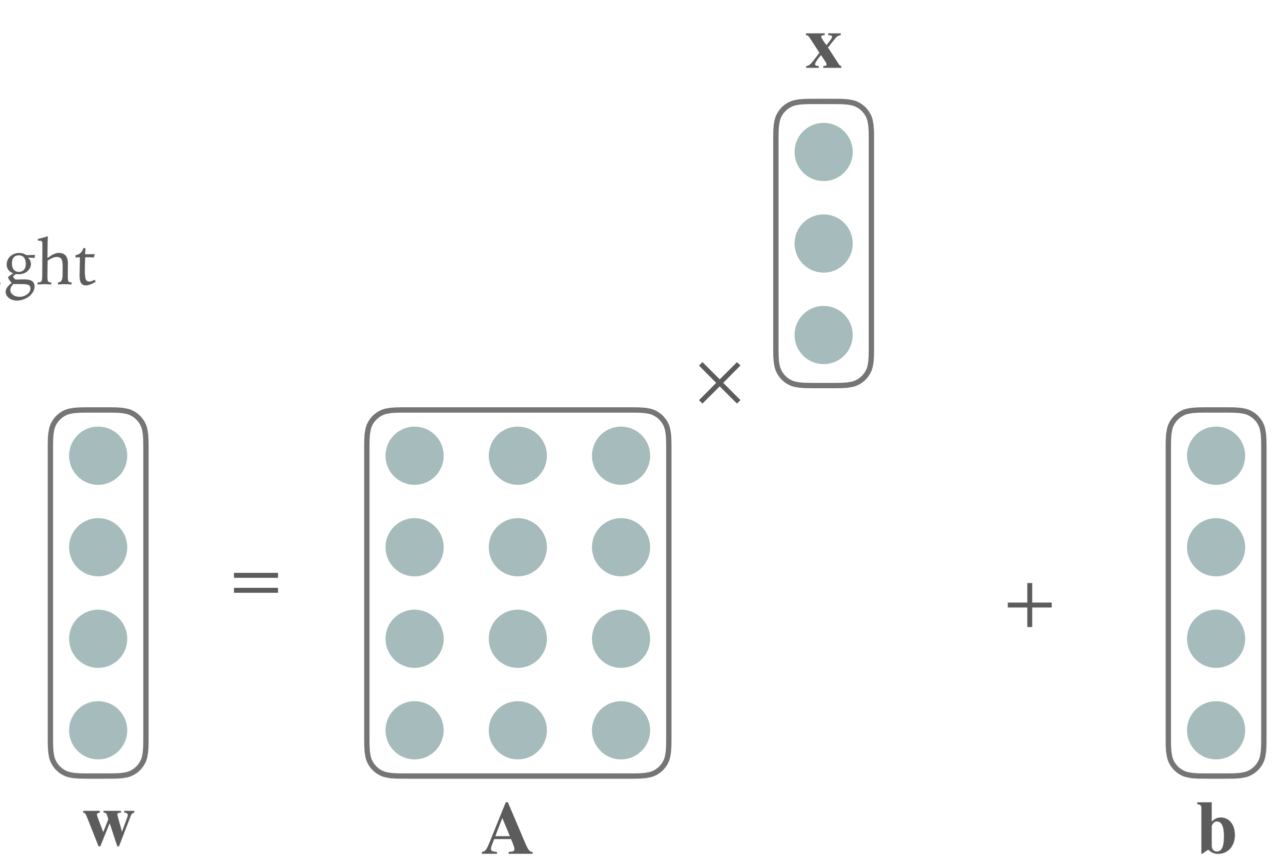
LINEAR CLASSIFIER FOR MULTI-CLASS CLASSIFICATION

Problem

- Input: features
- Output: 1-in-k prediction

Linear classifier $\mathbf{w} = \mathbf{A}\mathbf{x} + \mathbf{b}$

- Input dim: 3
- Output dim: k=4 classes
- Prediction: class with maximum weight



MULTILAYER PERCEPTRON 1/2

$$\mathbf{z}^{(1)} = \sigma\left(\mathbf{A}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

First hidden layer

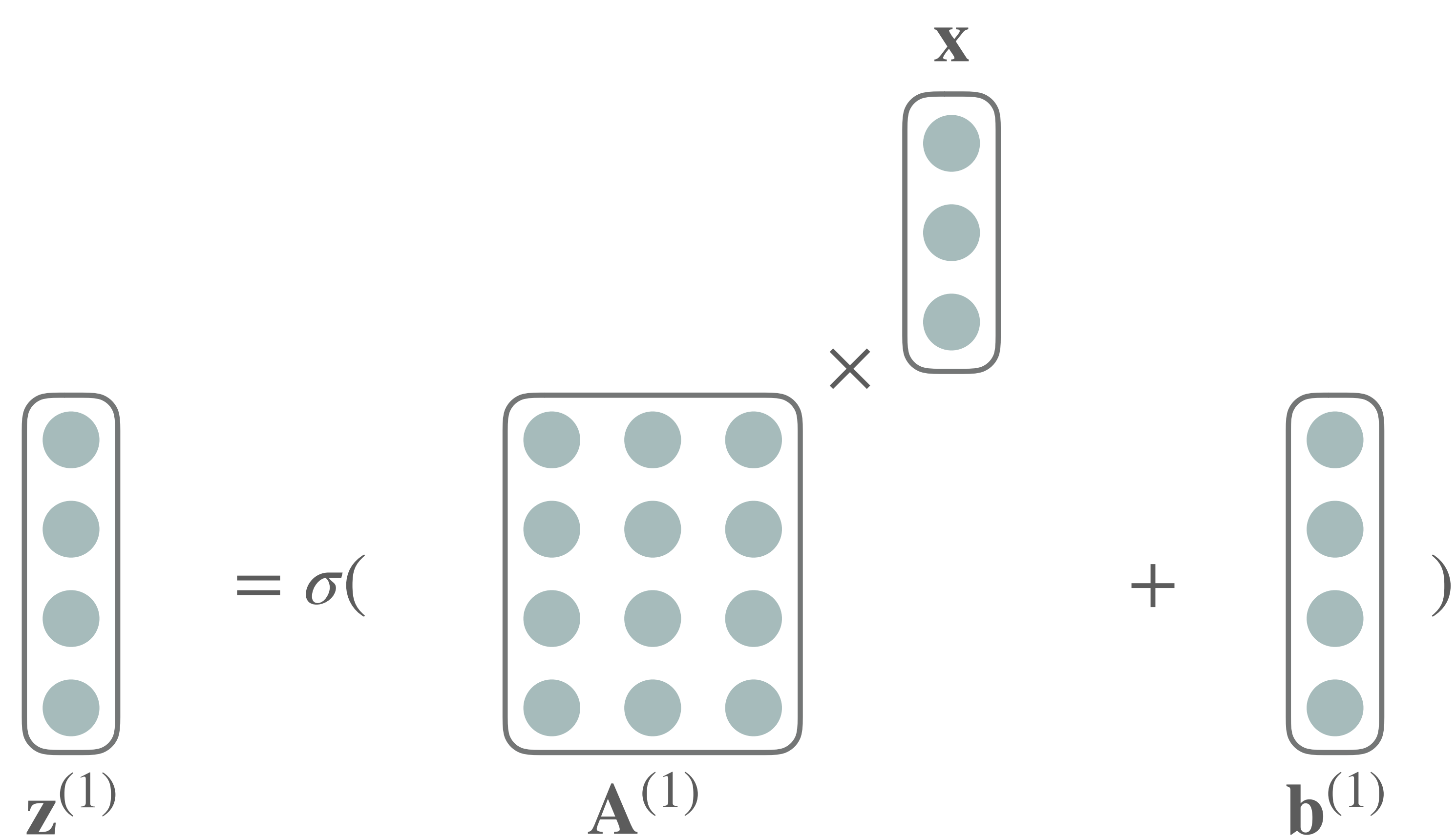
$$\mathbf{z}^{(2)} = \sigma\left(\mathbf{A}^{(2)}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}\right)$$

Second hidden layer

$$\mathbf{w} = \mathbf{A}^{(3)}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

Output projection

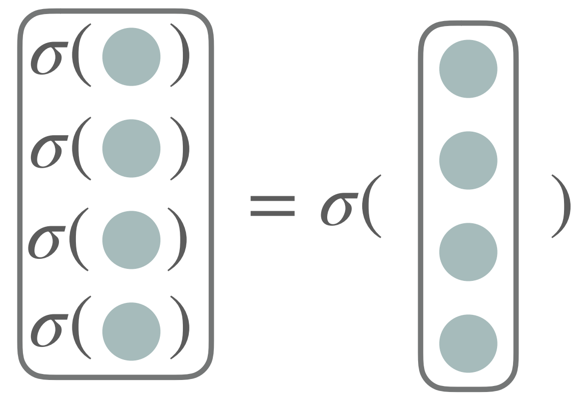
- $\mathbf{x}$: input features
- $\mathbf{z}^{(\mathbf{i})}$: hidden representations
- $\mathbf{w}$: output logits
- $\theta = \{\mathbf{A}^{(1)}, \mathbf{b}^{(1)}, \dots\}$: trainable parameters
- σ : piecewise non-linear activation function



NON-LINEAR ACTIVATION FUNCTIONS 1/2

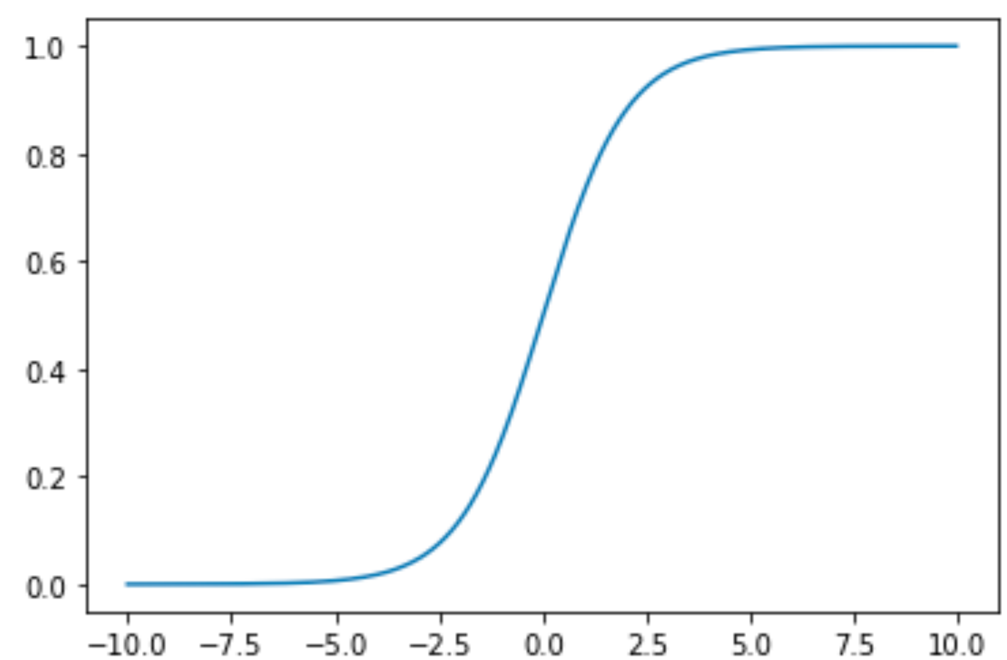
Main idea

- Apply a non-linear transformation
- Piecewise (so its fast to compute)
- There are many possibilities (I'll just present 3 of them)



Sigmoid

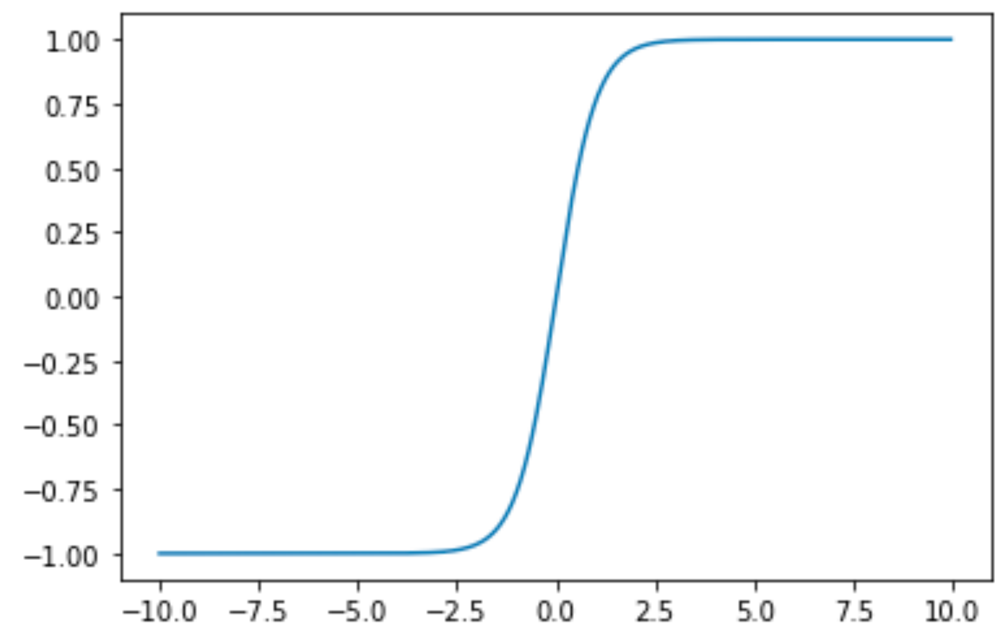
$$\sigma(u) = \frac{\exp(u)}{1 + \exp(u)} = \frac{1}{1 + \exp(-u)}$$



NON-LINEAR ACTIVATION FUNCTIONS 2/2

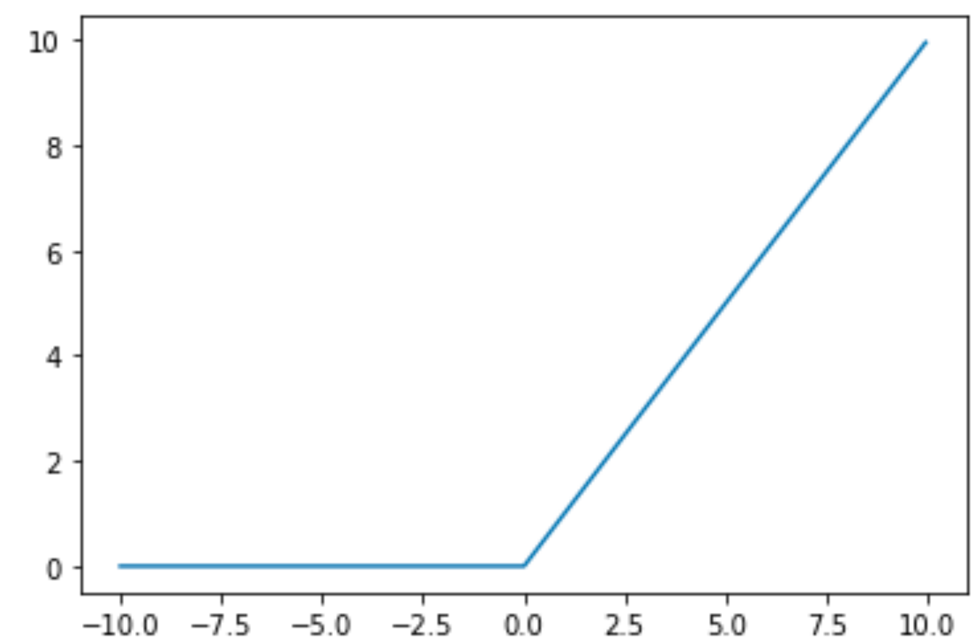
Hyperbolic tangent (tanh)

$$\tanh(u) = \frac{\exp(2u) - 1}{\exp(2u) + 1}$$



Rectified Linear Unit (relu)

$$\text{relu}(u) = \max(0,u)$$



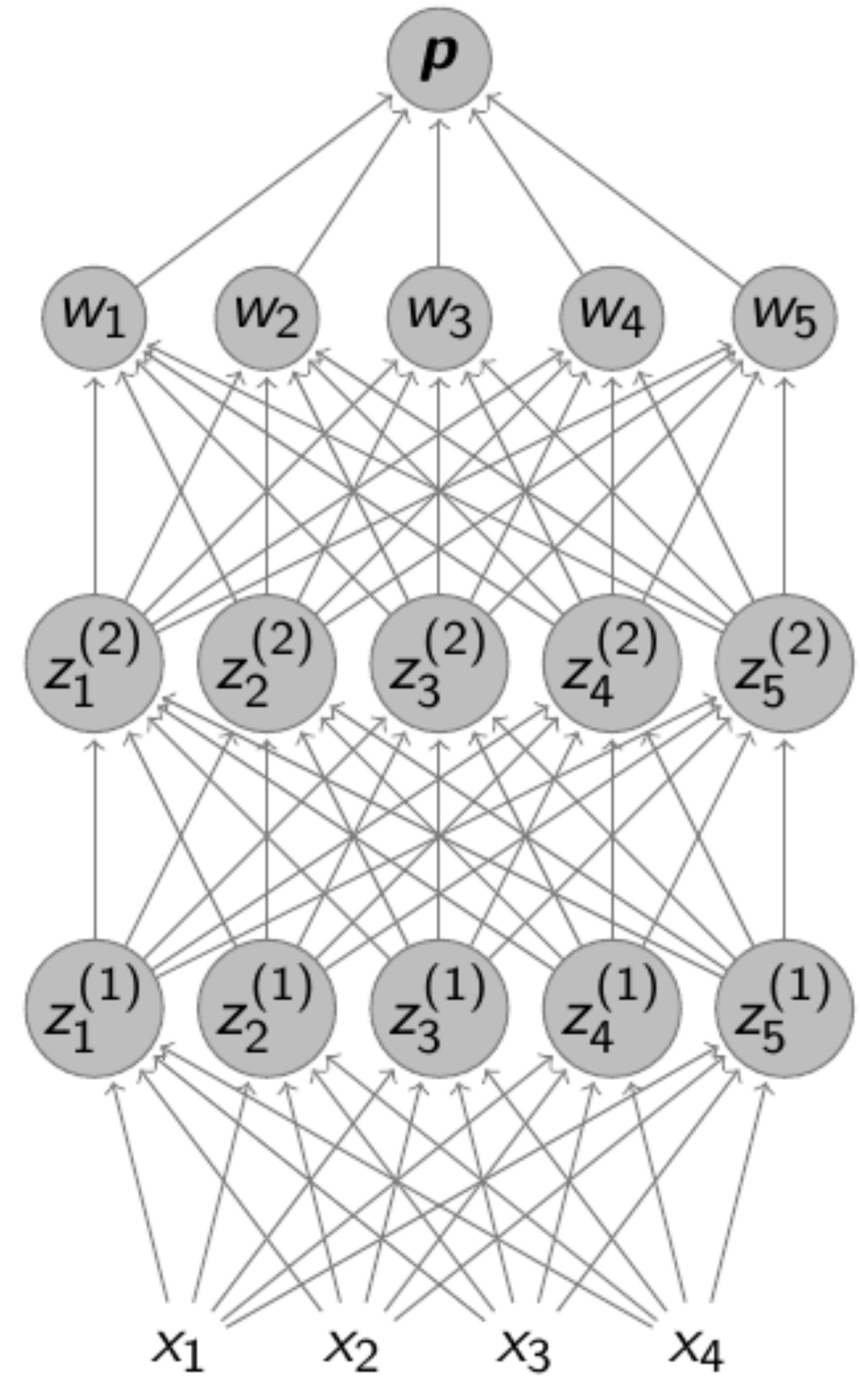
- $\boldsymbol{x}$: input features
- $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}$: hidden representation
- $\boldsymbol{w}$: output logits or class weights
- $\boldsymbol{p}$: probability distribution over classes
- $\theta = \{\boldsymbol{A}^{(1)}, \boldsymbol{b}^{(1)}, \ldots\}$: parameters
- σ: non-linear activation function

$$\boldsymbol{z}^{(1)} = \sigma\left(\boldsymbol{A}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{z}^{(2)} = \sigma\left(\boldsymbol{A}^{(2)}\boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

$$\boldsymbol{w} = \sigma\left(\boldsymbol{A}^{(3)}\boldsymbol{z}^{(2)} + \boldsymbol{b}^{(3)}\right)$$

$$\boldsymbol{p} = \text{Softmax}(\boldsymbol{w}) \quad \text{i.e.} \quad p_i = \frac{\exp(w_i)}{\sum_j \exp(w_j)}$$



Graphical or mathematical representation?

- Use a graphical representation only if required
- Alway prefer the mathematical description!

Code example!

PREDICTION FUNCTION

Vocabulary issue

The term "prediction function" can refer to both the "full model" or only the function that transforms the class weights/logits/scores to an actual output. :(

DO NOT CONFUSE

- The (non-linear) activation function (inside the neural network)
- The function that transforms weights/logits/scores into an output (at the output of the neural network)

NEURAL ARCHITECTURES: A REALLY QUICK OVERVIEW

NEURAL ARCHITECTURE DESIGN

Neural network = complicated parameterized function

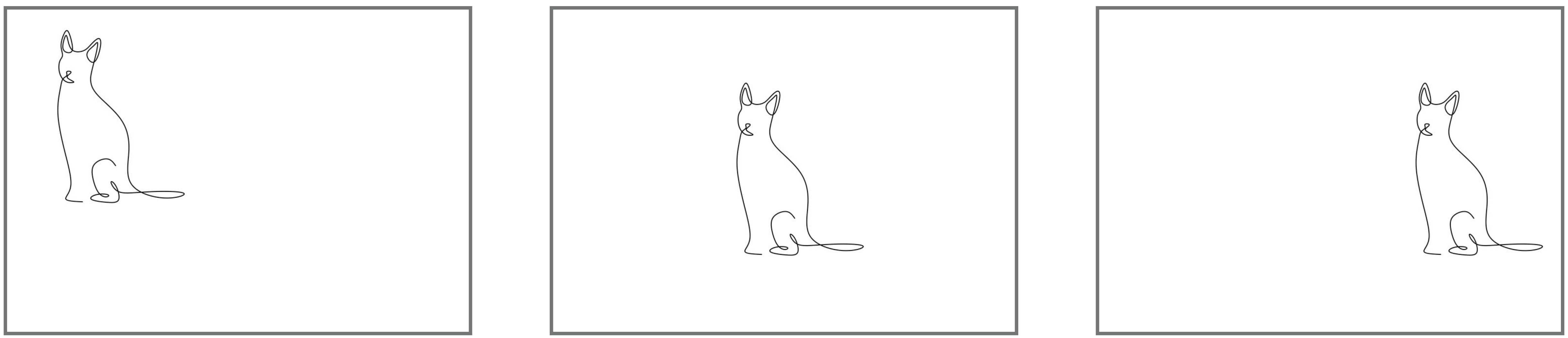
- Inductive bias: take into account the data to design the architectures
- Time complexity/speed
- Mathematical properties for efficient training:
  differentiability, prevent vanishing/exploding gradients

CONVOLUTIONAL NEURAL NETWORKS (CNN)

Intuition

No matter where the cat is in the picture, it is a cat

=> we want to encode this fact in the neural architecture!

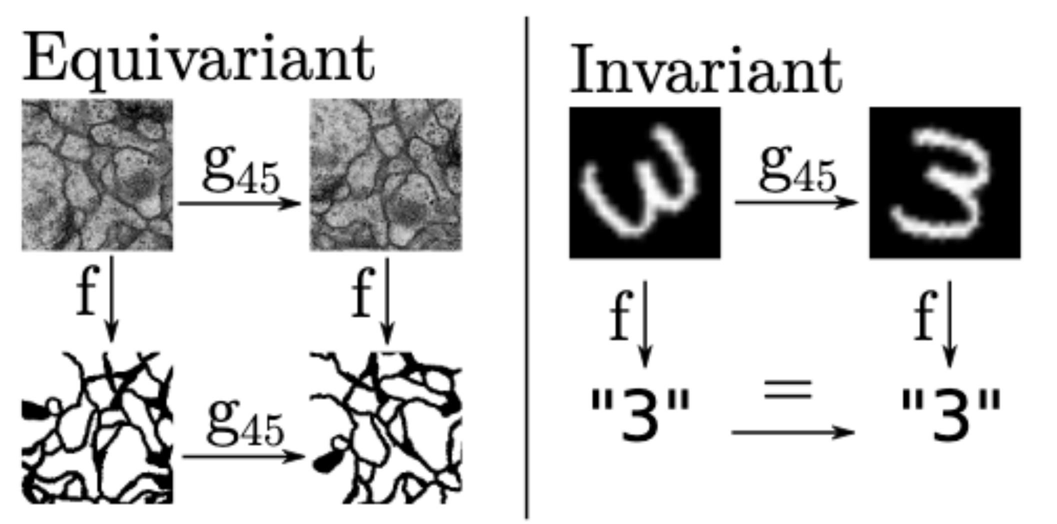


Equivariant function

If we apply a transformation on the input,
the output will be transformed in the « same » way

Invariant function

If we apply a transformation on the input,
the output will remain the same



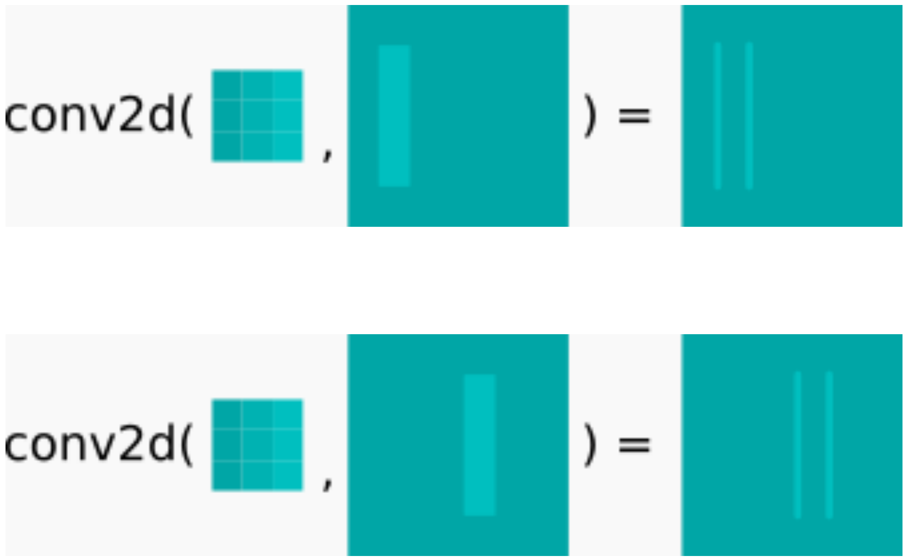
EQUIVARIANT CONVOLUTIONS IN COMPUTER VISION

Translation equivariant convolution

Preserves the « translation structure »

- ➤ If the input is transposed
- ➤ The output is also transposed

+ pooling will make the model invariant



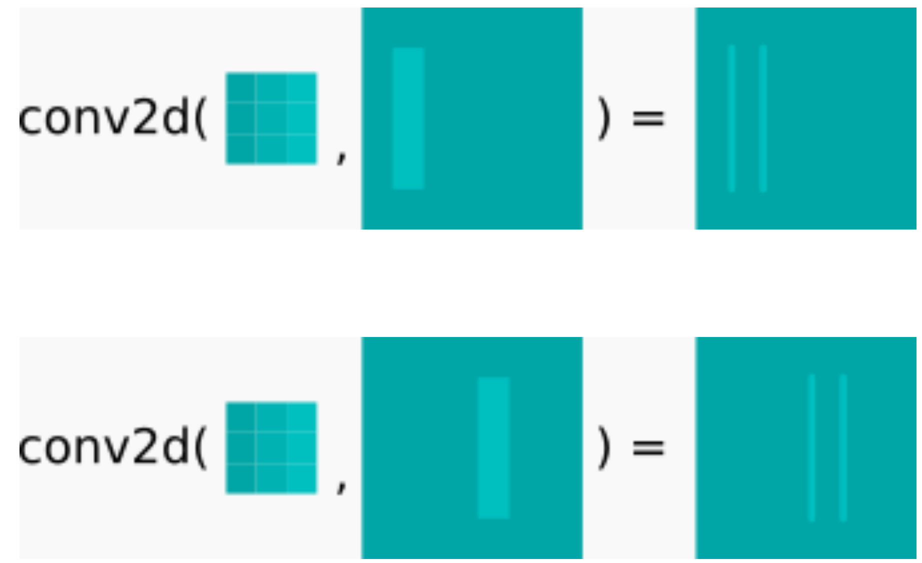
EQUIVARIANT CONVOLUTIONS IN COMPUTER VISION

Translation equivariant convolution

Preserves the « translation structure »

- ➤ If the input is transposed
- ➤ The output is also transposed

+ pooling will make the model invariant

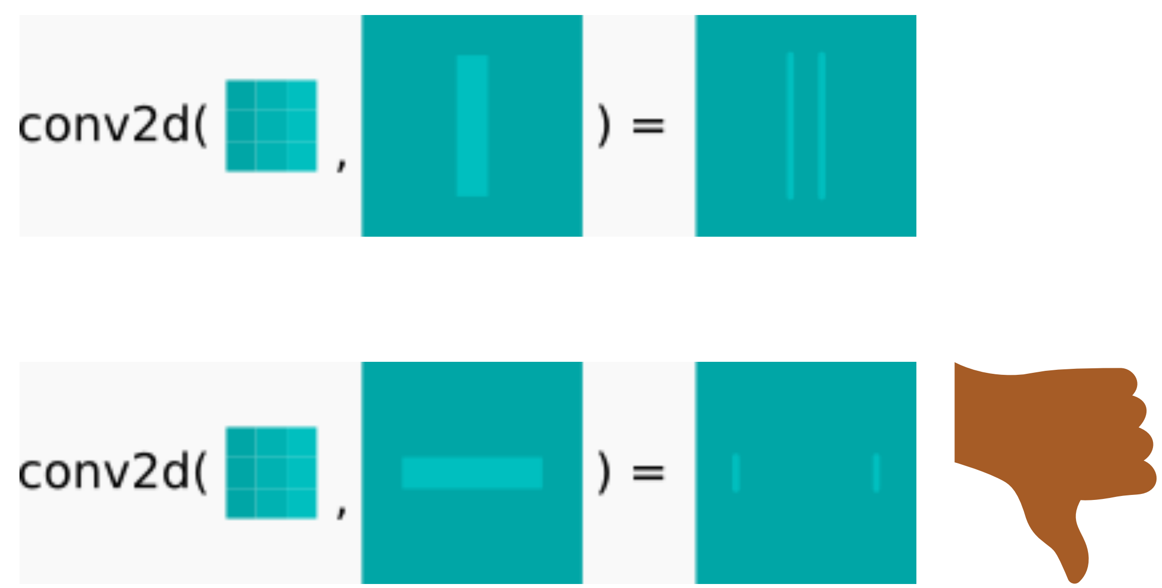


Rotation equivariant convolution

Preserves the « rotation structure »

- ➤ If the input is rotated
- ➤ The output is also rotated

Standard convolution **<u>is not</u>** rotation equivariant



GROUP CONVOLUTIONS

[Cohen and Weiling, 2016]

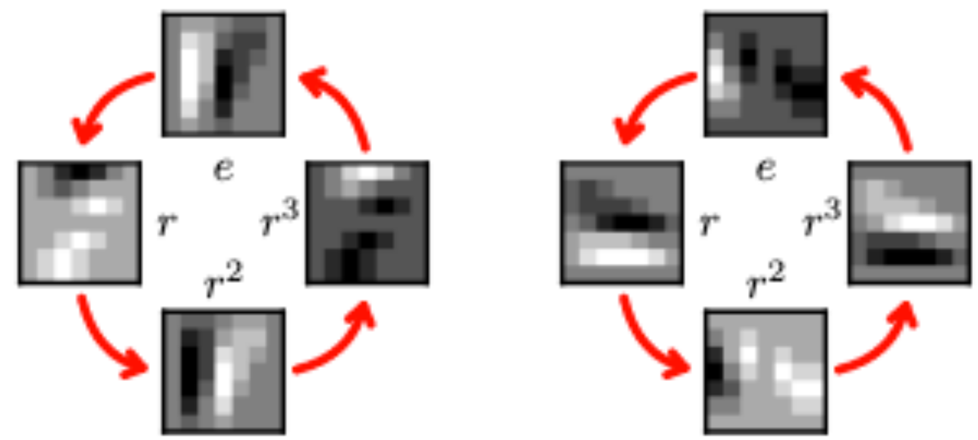


Figure 1. A p4 feature map and its rotation by r.

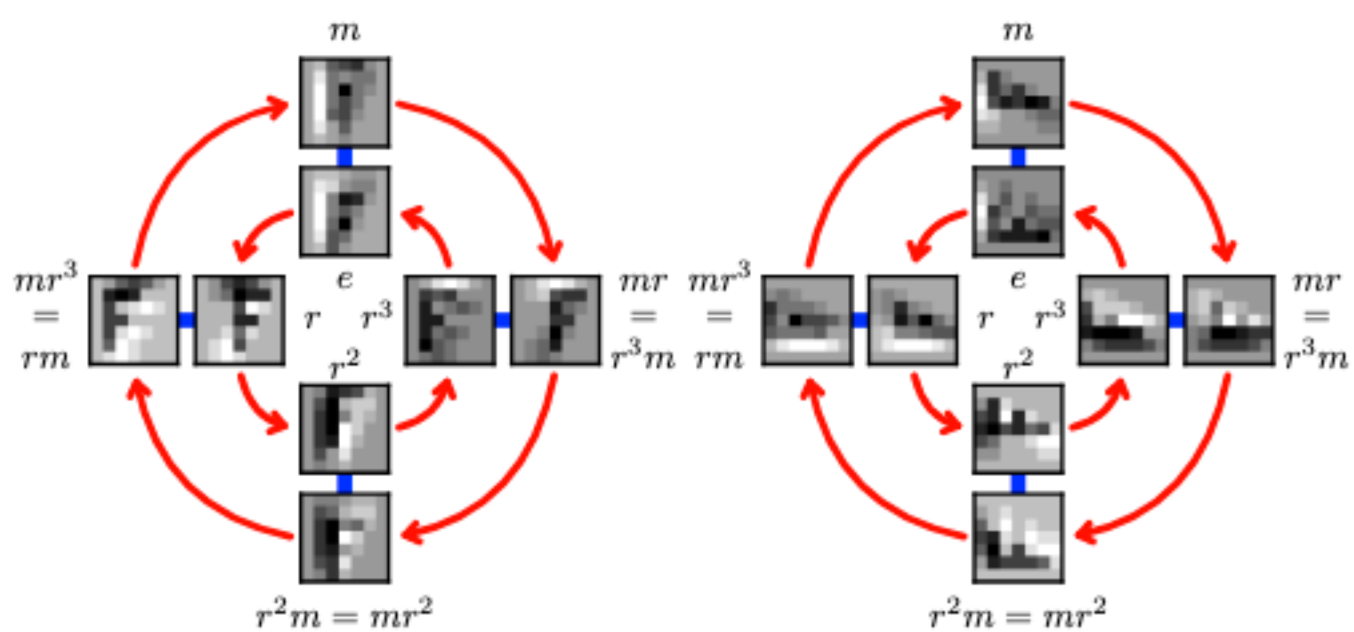
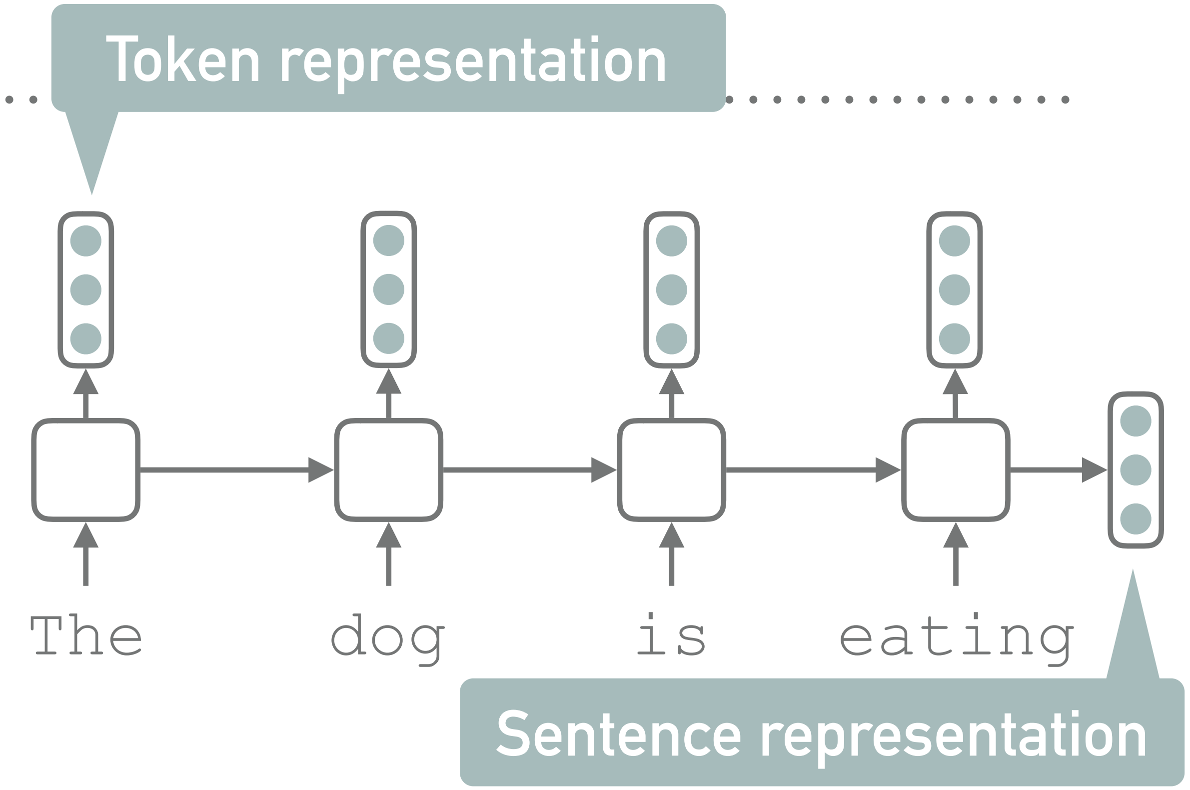


Figure 2. A p4m feature map and its rotation by r.

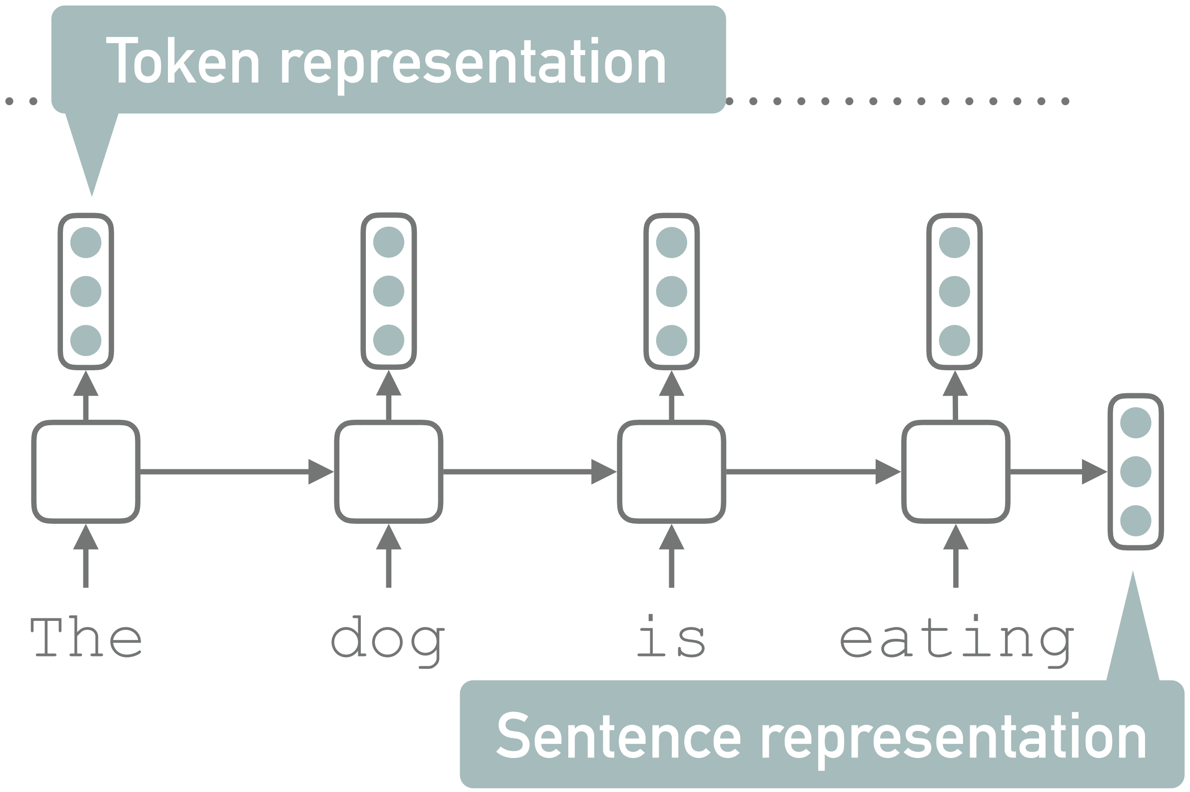
RECURRENT NEURAL NETWORKS

Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



RECURRENT NEURAL NETWORKS



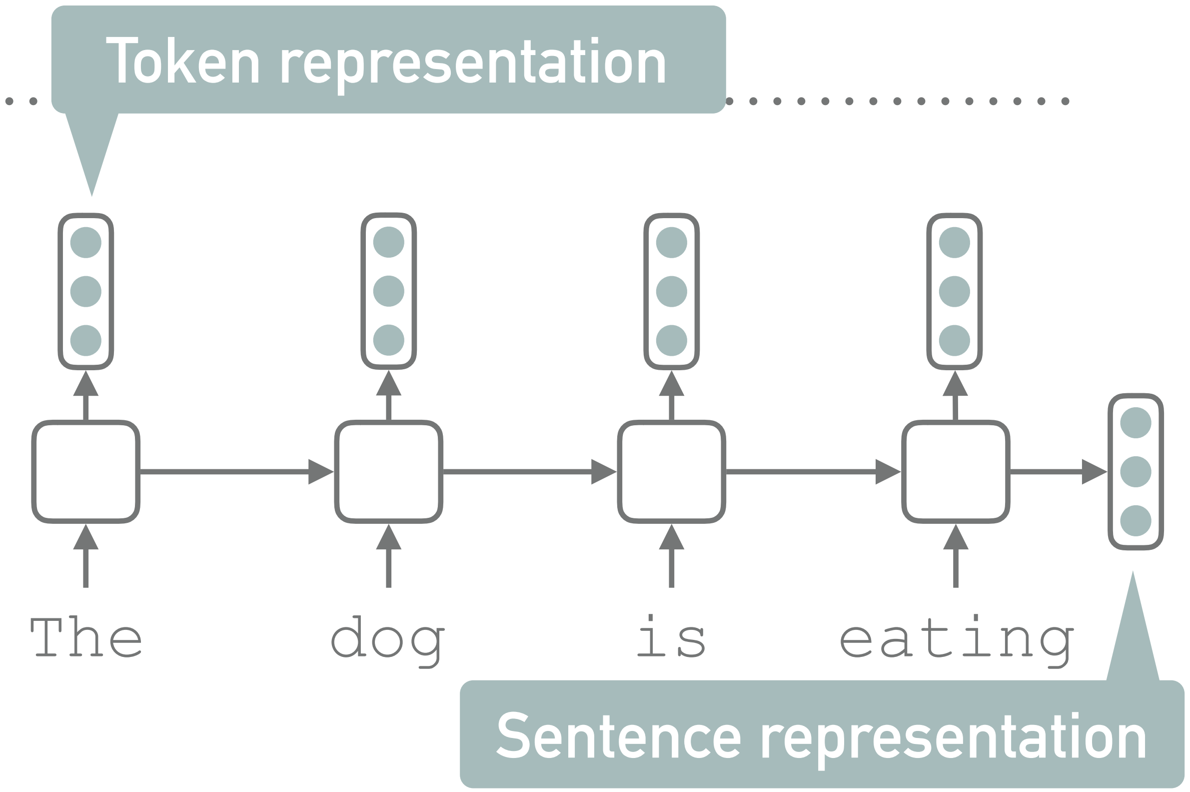
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input

Intuition

Use two RNNs with **different trainable parameters**

RECURRENT NEURAL NETWORKS

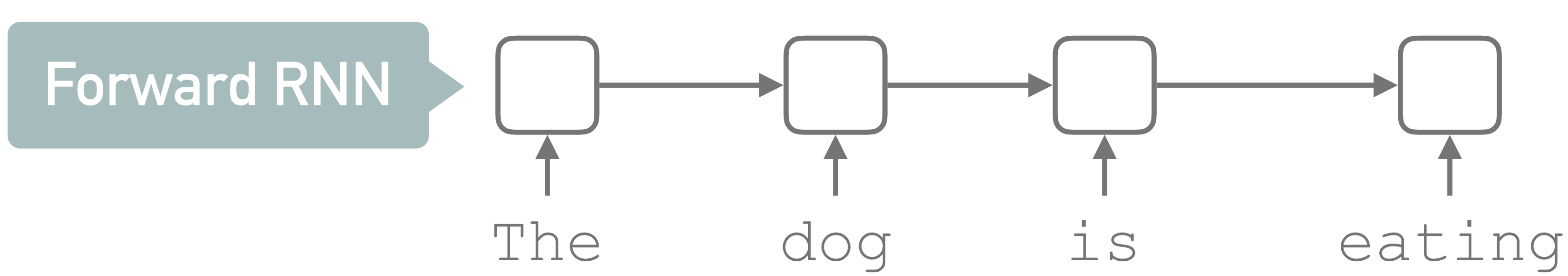


Recurrent neural networks

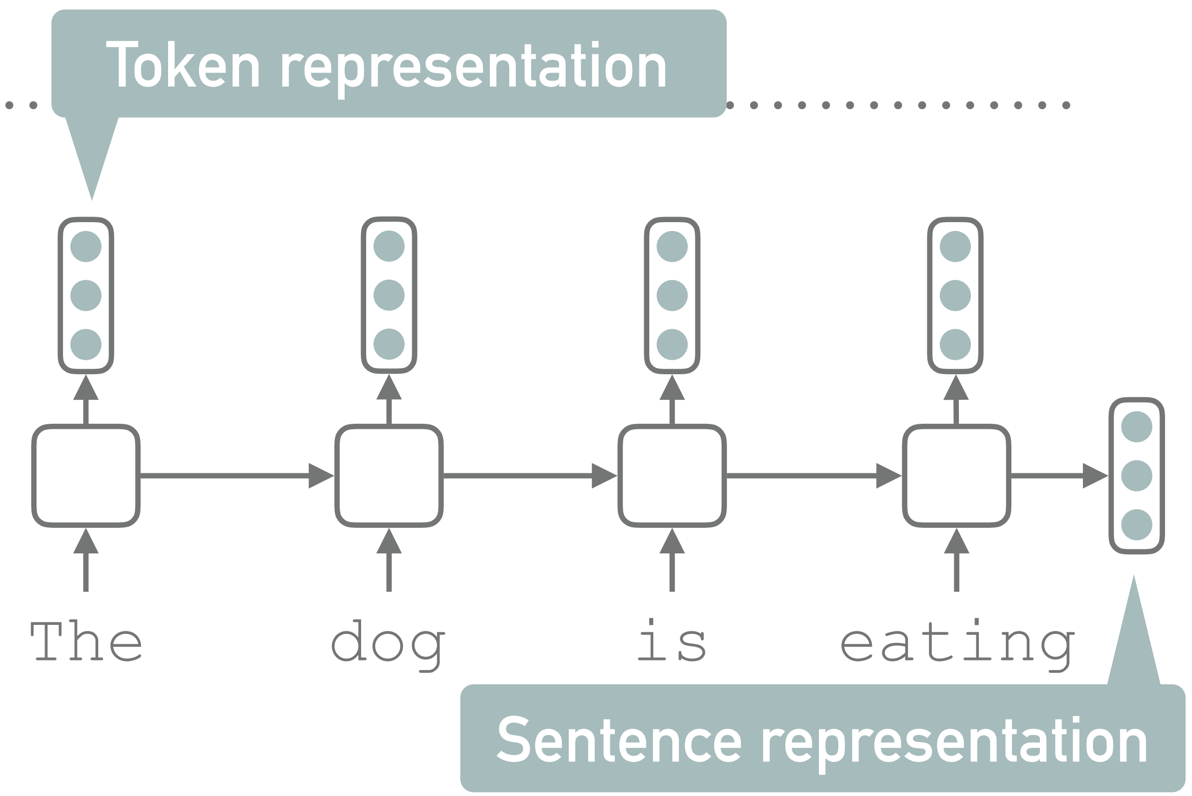
- Inputs are fed sequentially
- State representation updated at each input

Intuition

Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

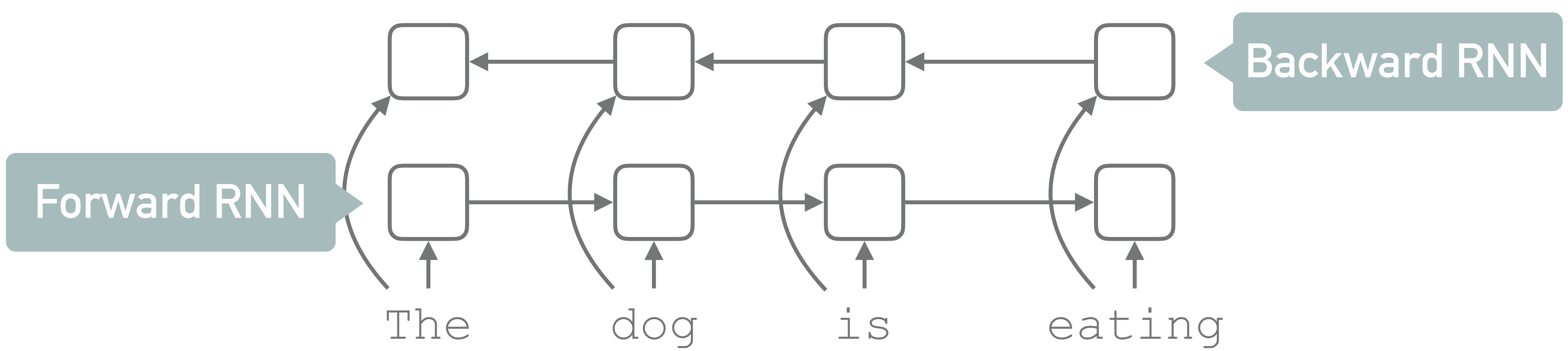


Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input

Intuition

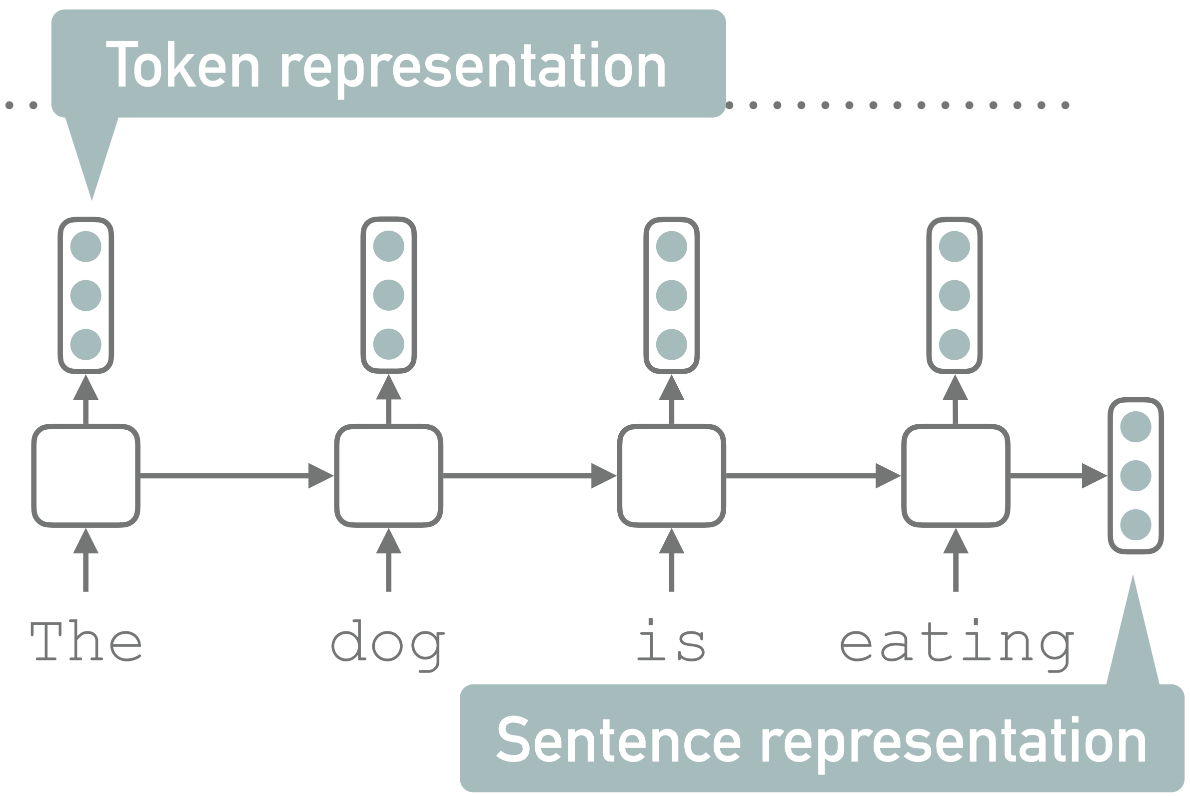
Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

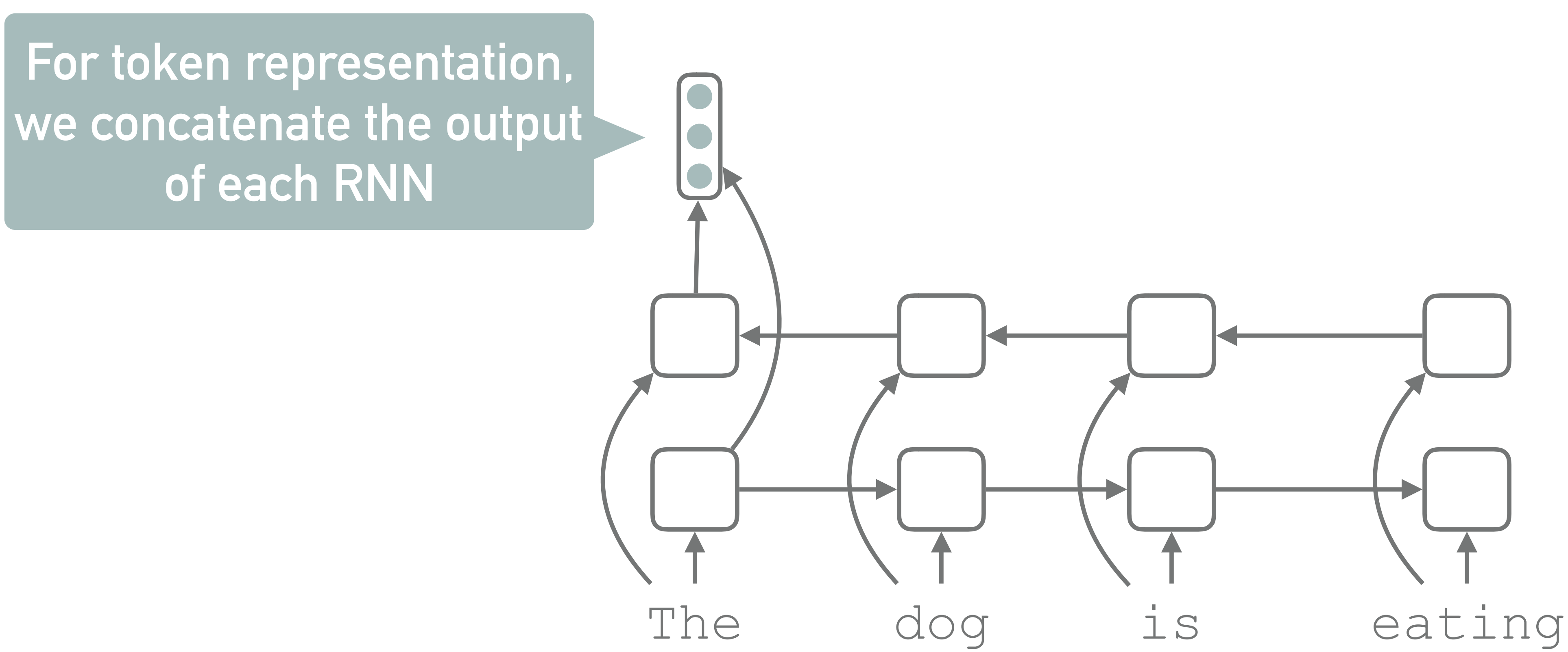
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input



Intuition

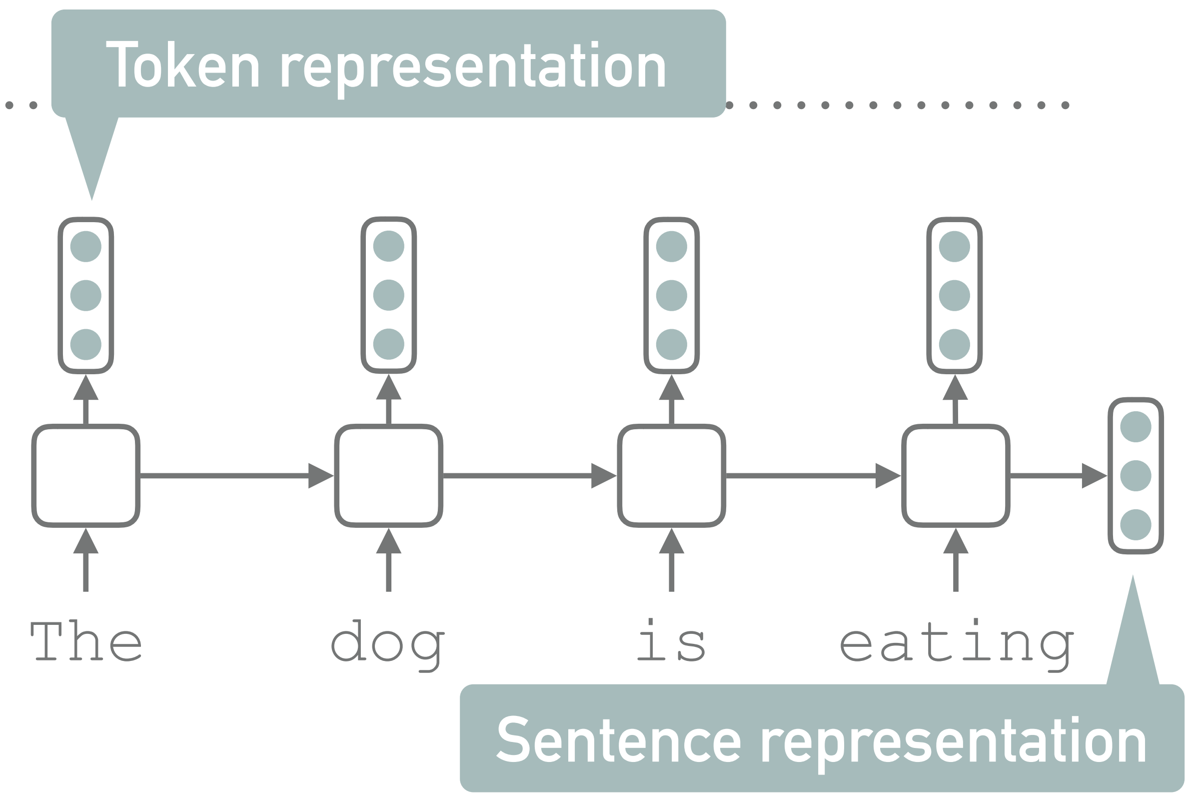
Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

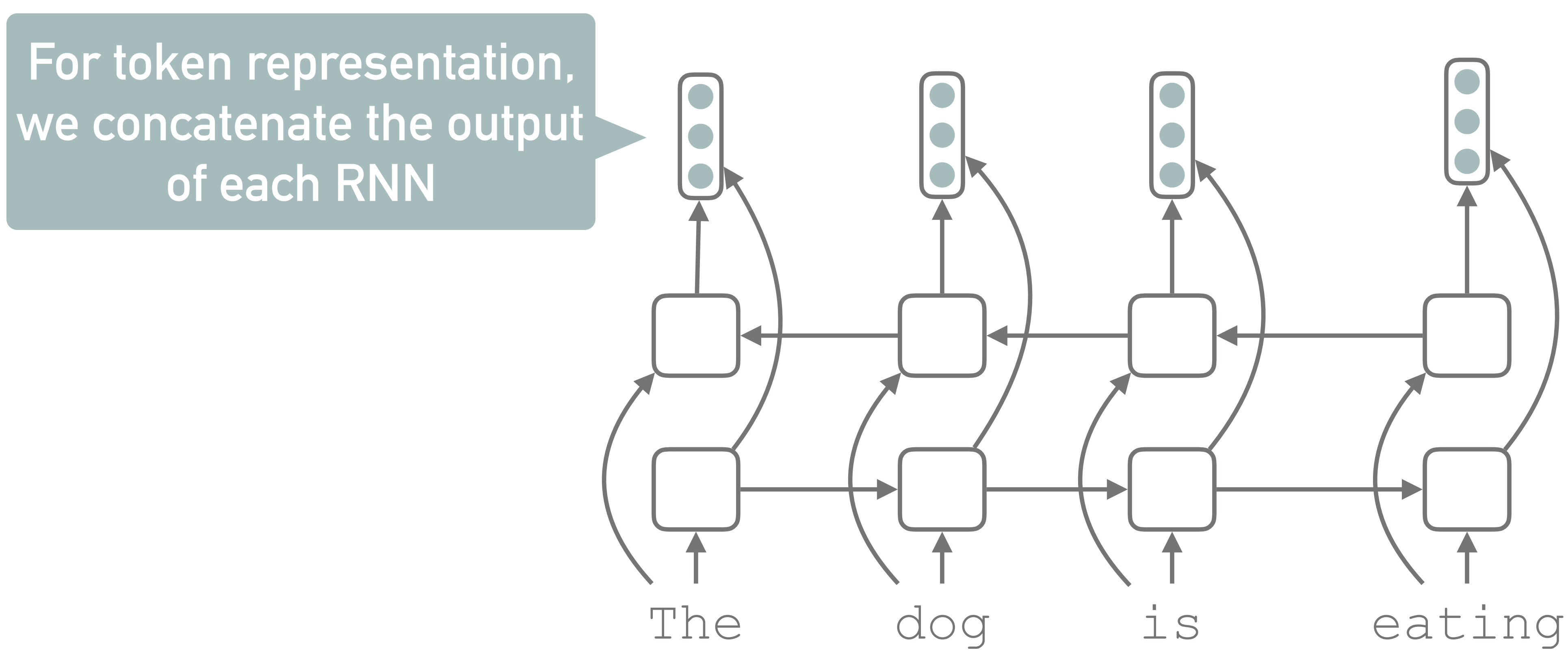
Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input

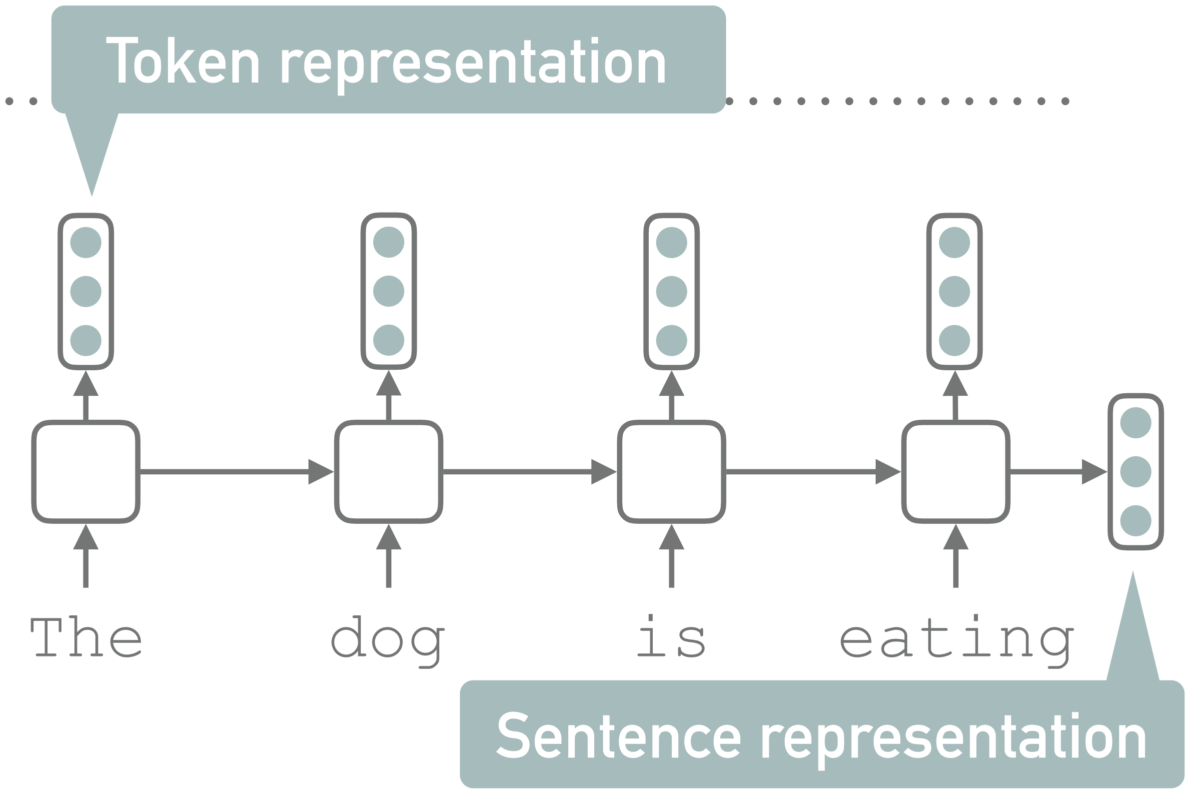


Intuition

Use two RNNs with **different trainable parameters**



RECURRENT NEURAL NETWORKS

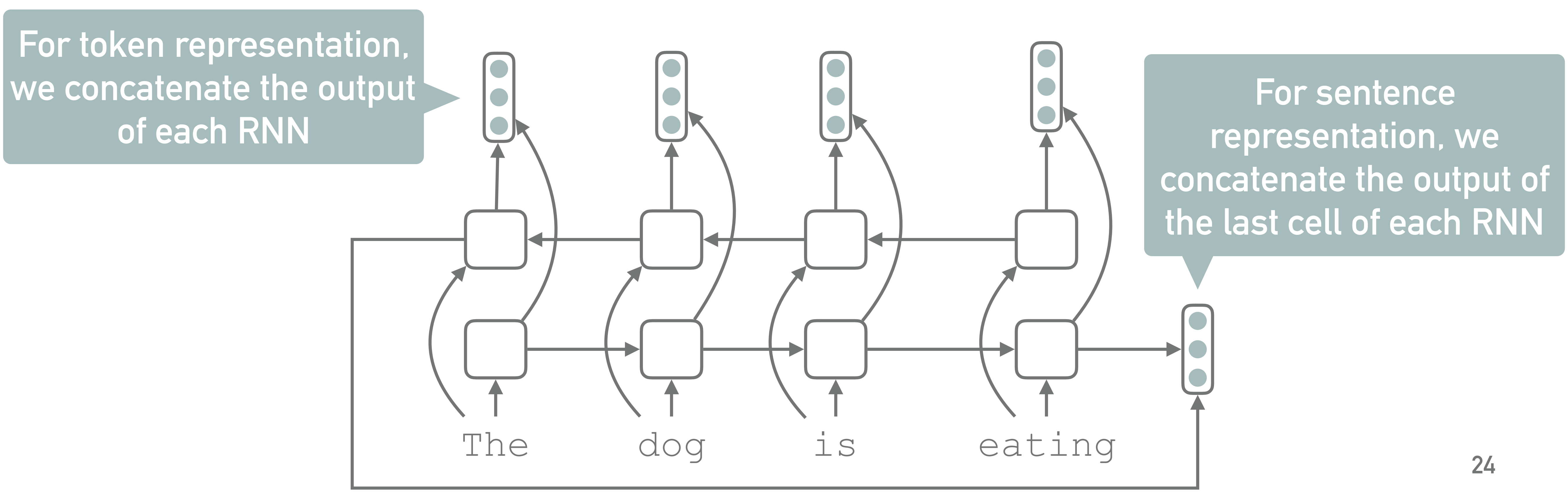


Recurrent neural networks

- Inputs are fed sequentially
- State representation updated at each input

Intuition

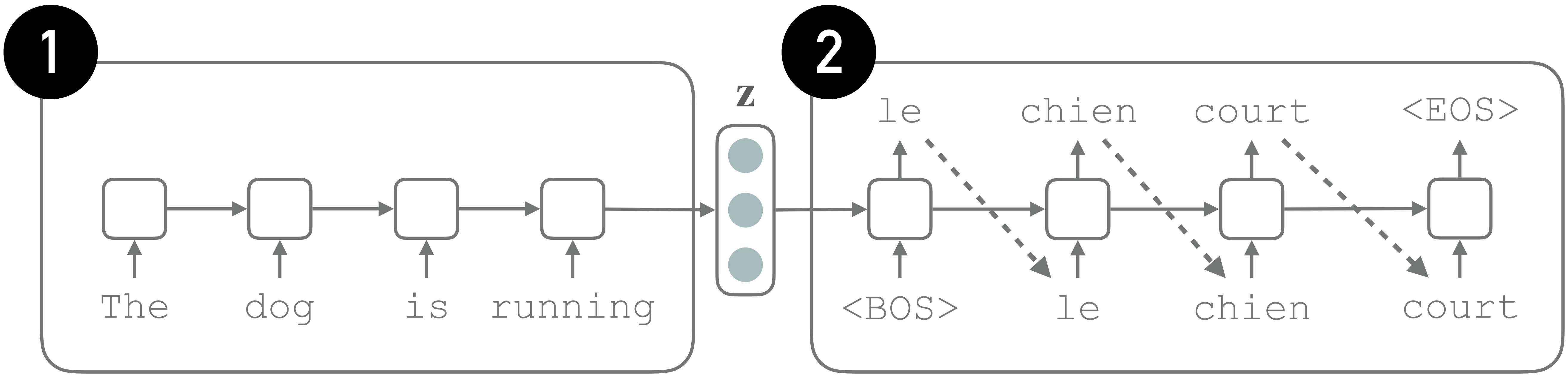
Use two RNNs with **different trainable parameters**



SEQUENCE TO SEQUENCE (SEQ2SEQ)

Intuition

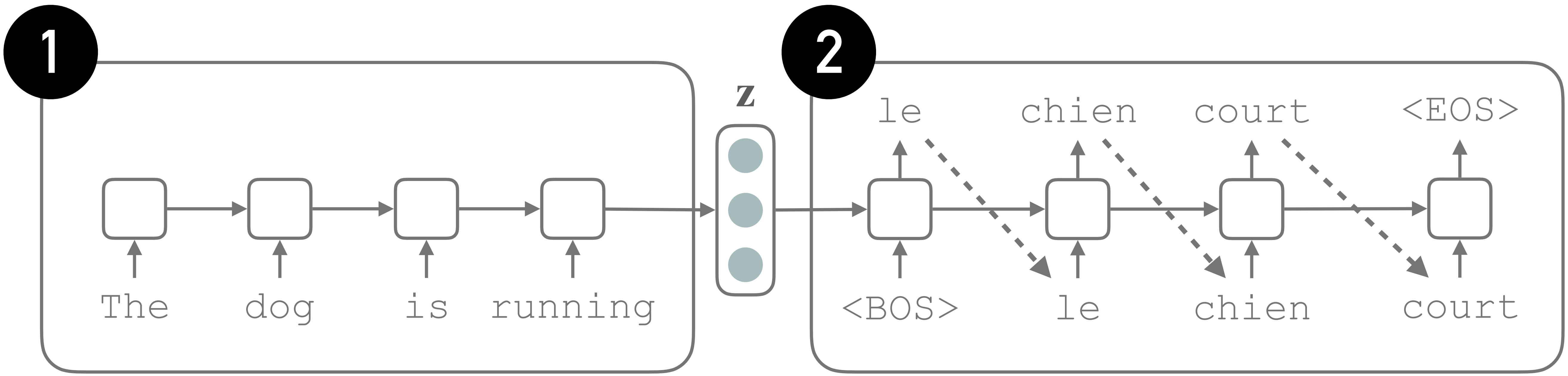
1. **Encoder:** encode the input sentence into a fixed size vector (sentence embedding)
2. **Decoder:** generate the translation auto-regressively (word by word) conditioned on the input sentence embedding



SEQUENCE TO SEQUENCE (SEQ2SEQ)

Intuition

1. **Encoder:** encode the input sentence into a fixed size vector (sentence embedding)
2. **Decoder:** generate the translation auto-regressively (word by word) conditioned on the input sentence embedding



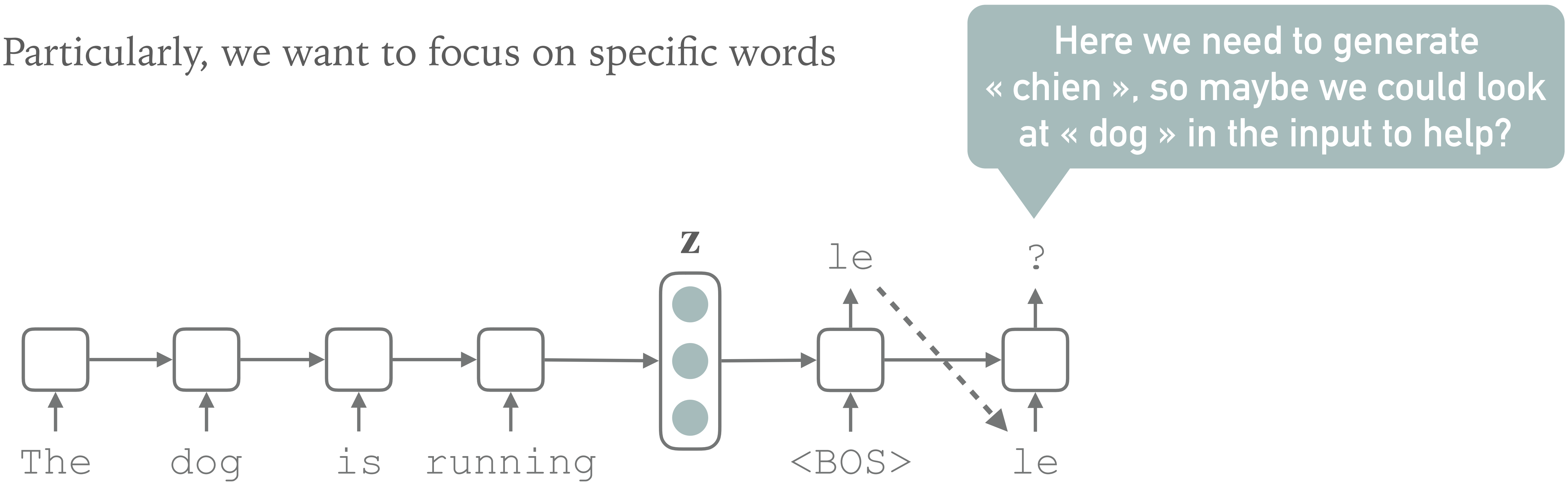
The sentence embedding is a bottleneck, everything must be encoded inside!

SEQ2SEQ WITH ATTENTION

[Bahdanau et al., 2014]

Intuition

- During decoding, we want to « look » at the input sentence
- Particularly, we want to focus on specific words

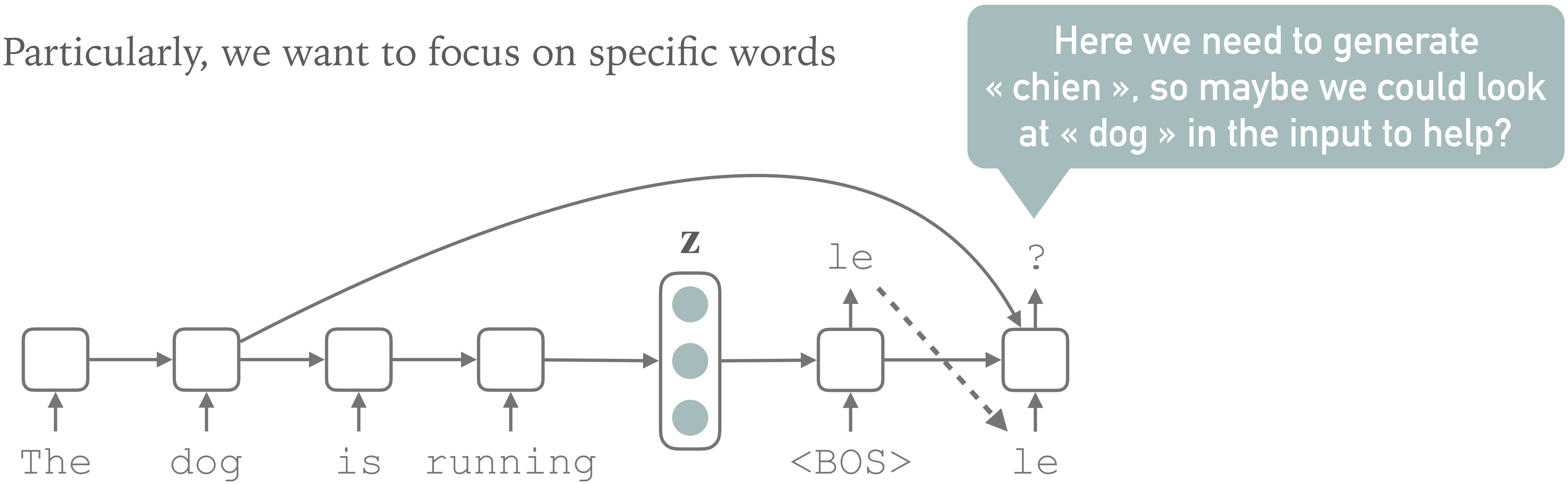


SEQ2SEQ WITH ATTENTION

[Bahdanau et al., 2014]

Intuition

- During decoding, we want to « look » at the input sentence
- Particularly, we want to focus on specific words



Attention mechanism

We had a « module » that wil learn to look at a word from the input

SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

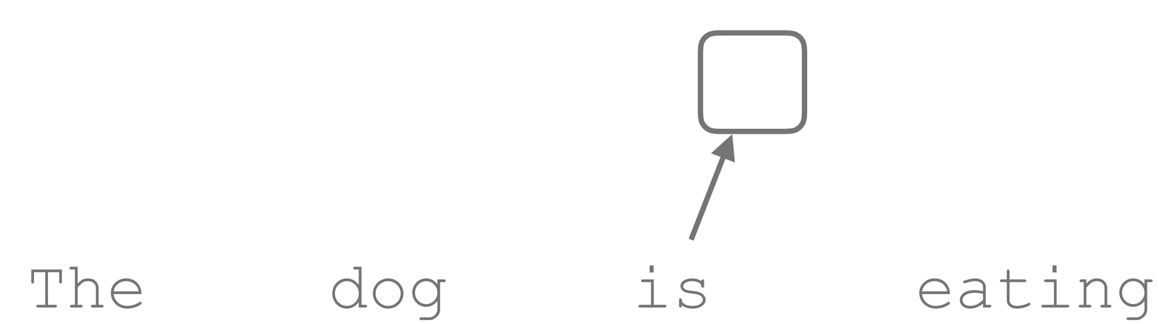
- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to

```
The      dog      is      eating
```

SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

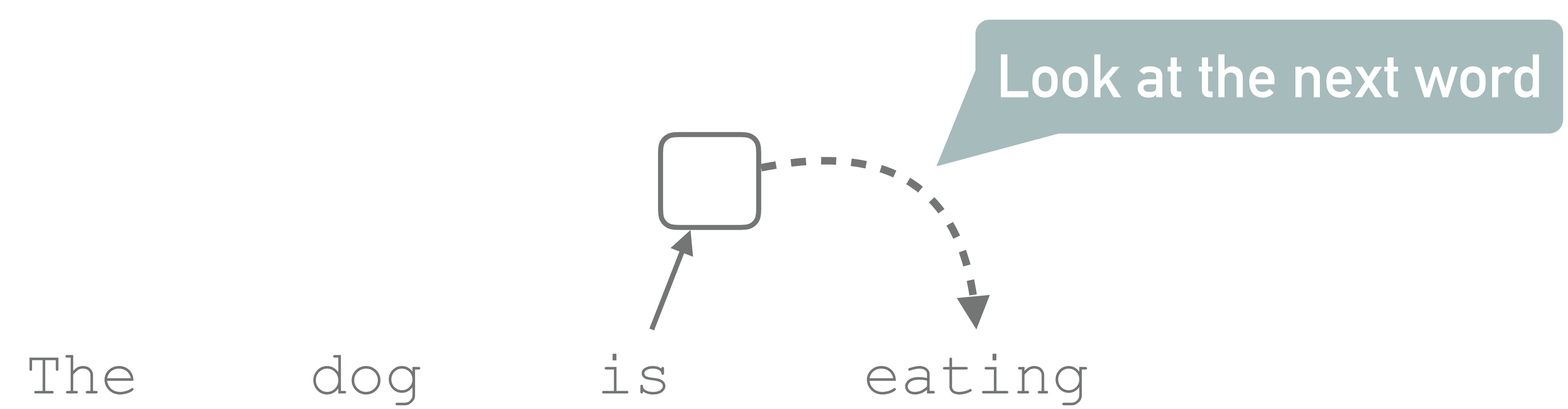
- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

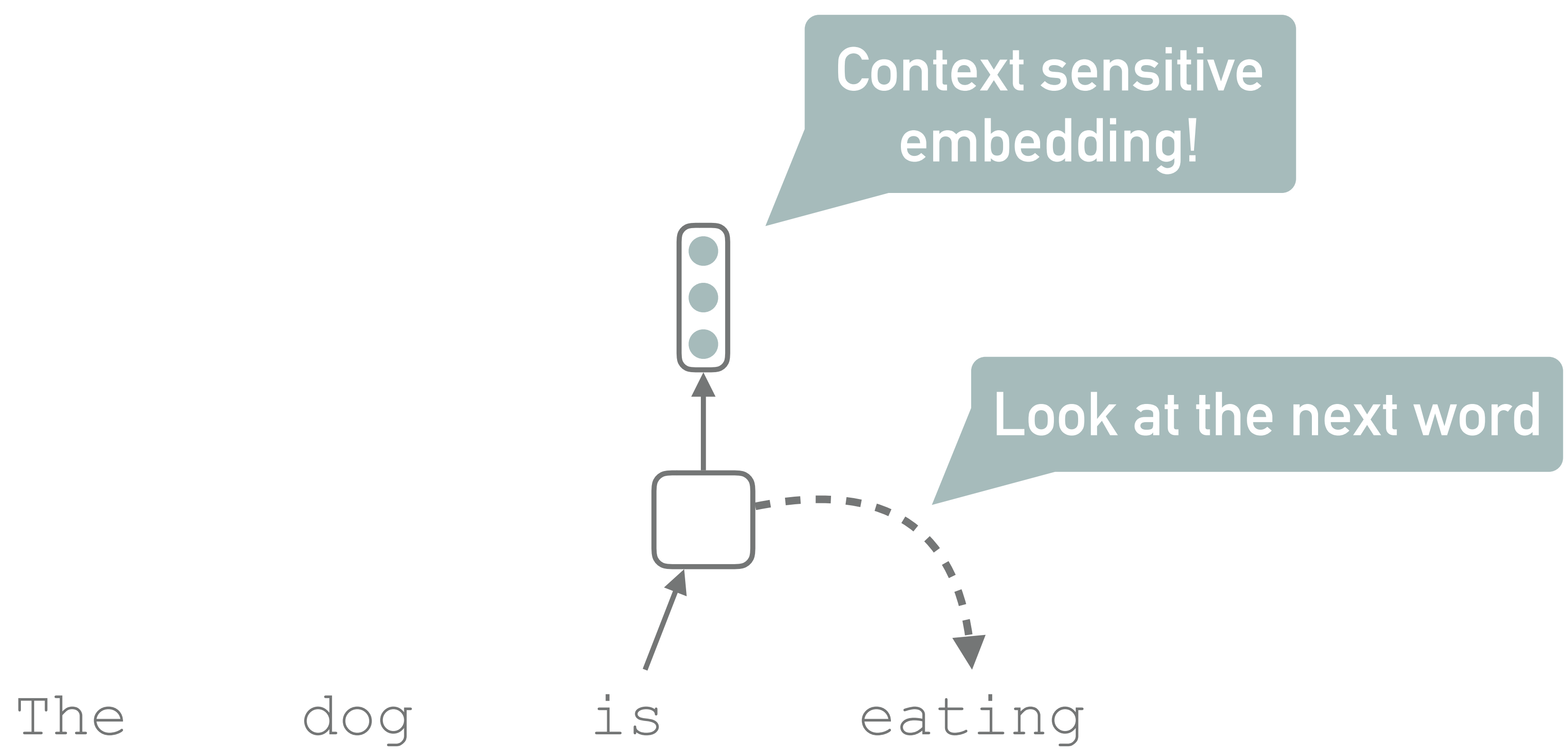
- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

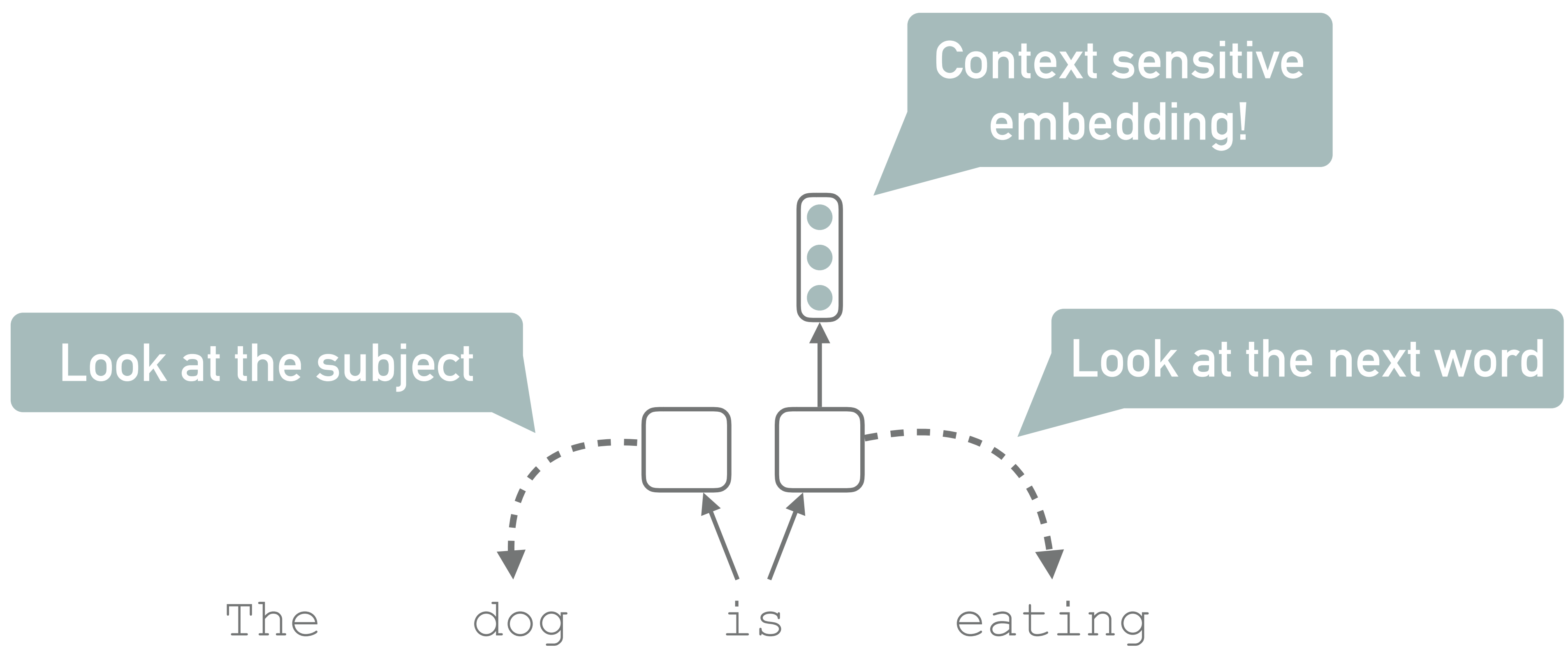
- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

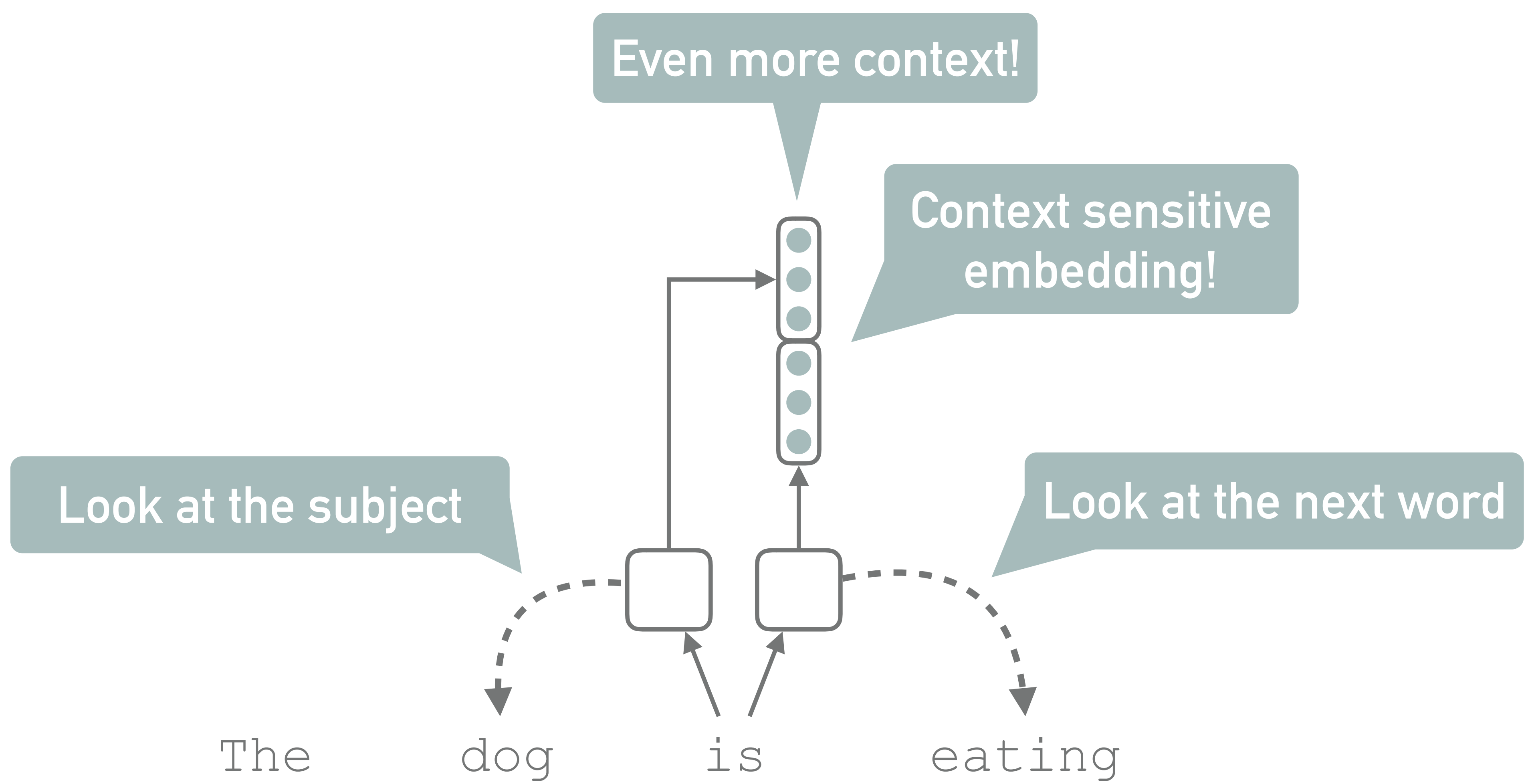
- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to
- Combine several attention modules to attend to several words



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

- Based on "heads" that, for a given input, look at other
- The model learns which word a given head must attend to
- Combine several attention modules to attend to several words



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

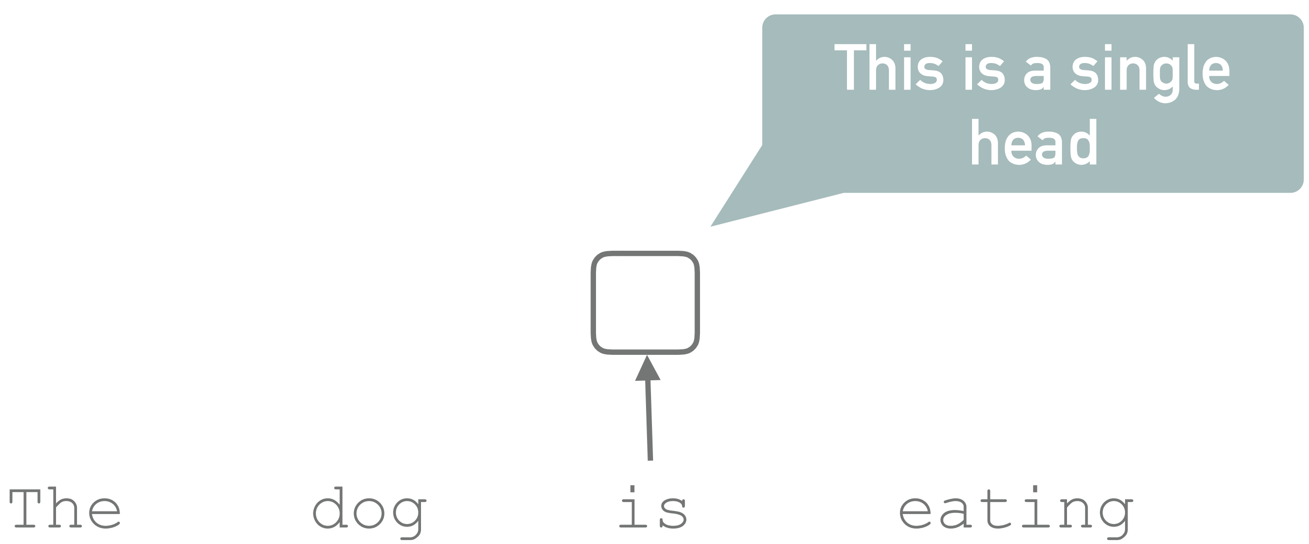
- A head is applied to a given position and try to combine with another word
- Each head is applied to each position in the sentence
- We can use efficient batch matrix multiplication instead of loops

The dog is eating

SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

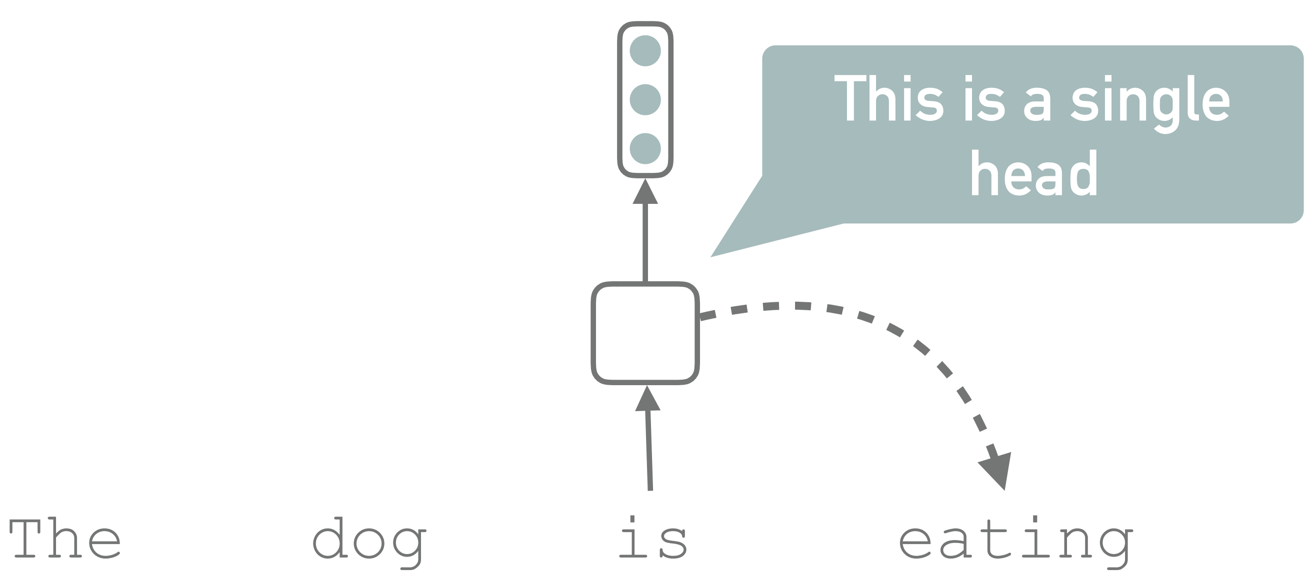
- A head is applied to a given position and try to combine with another word
- Each head is applied to each position in the sentence
- We can use efficient batch matrix multiplication instead of loops



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

- A head is applied to a given position and try to combine with another word
- Each head is applied to each position in the sentence
- We can use efficient batch matrix multiplication instead of loops



SELF-ATTENTIVE NEURAL NETWORKS / TRANSFORMERS

[Vaswani et al., 2017]

- A head is applied to a given position and try to combine with another word
- Each head is applied to each position in the sentence
- We can use efficient batch matrix multiplication instead of loops

GPU PARALLELIZATION

[Vaswani et al., 2017]

Intuition

- No recurrence: use attention only!
- Use many attention layers to be able to learn complex patterns

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Pros

- Easily parallelizable on GPU, very fast in practice
- Direct access to long range dependencies

Cons

- Harder to optimize than plain LSTMs

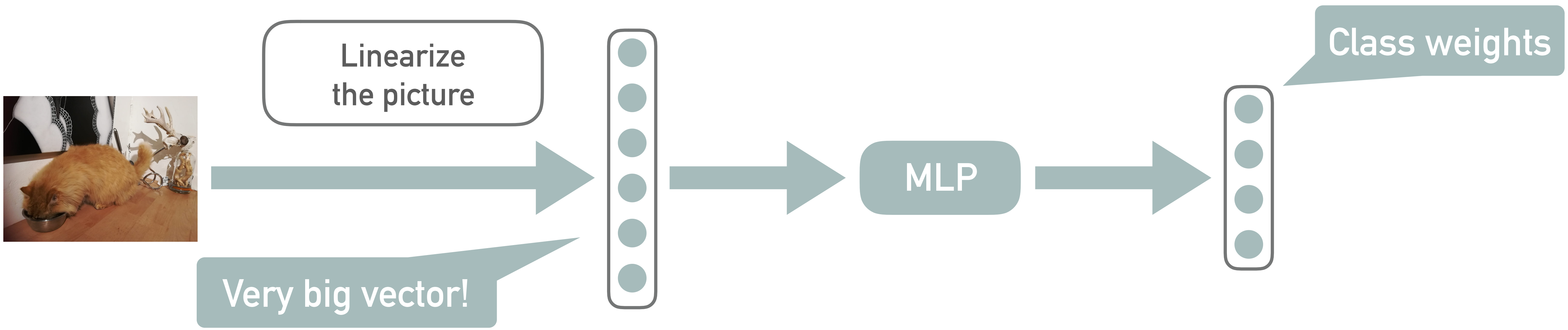
TAKEAWAY

You need to understand the problem you try to solve in order to build good neural architecture

CONVOLUTIONAL NEURAL NETWORKS

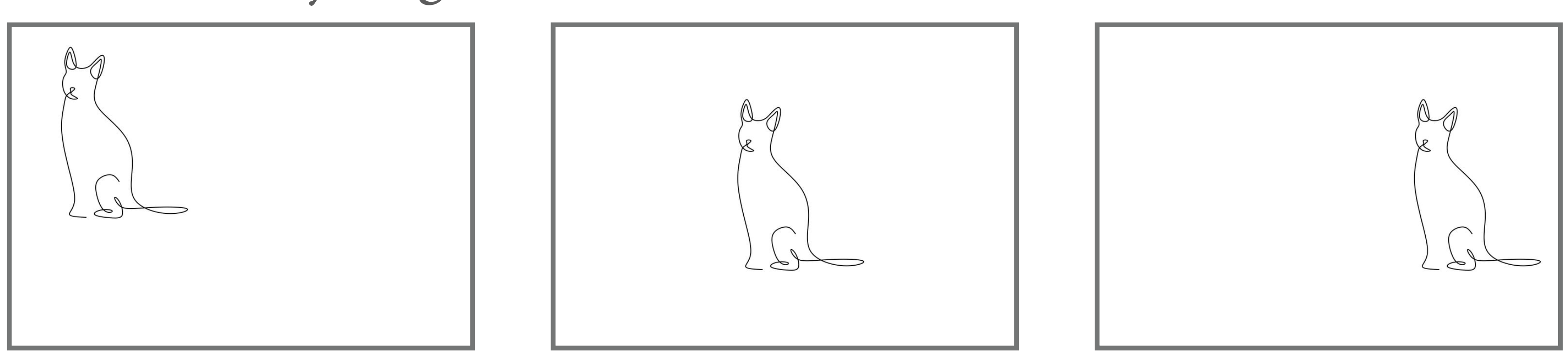
CONVOLUTIONAL NEURAL NETWORKS

Computer vision with a small MLP



Main idea behing convolutions

- No matter where the cat is in the picture, it is a cat
  => we want to encode this fact in the neural architecture!
- If we use a MLP for image inputs, if the input size is large, then the number of parameters will be very large



FILTERS AND CONVOLUTIONS

Assume a signal in 1 dimension

- A filter is a vector of fixed size
- A filter is applied to each position of the signal (convolved) to compute a transformation of the input signal

Input signal

This is a given input, in theory size is not fixed, a convolution can be applied on arbitrary size inputs

2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

FILTERS AND CONVOLUTIONS

Assume a signal in 1 dimension

- A filter is a vector of fixed size
- A filter is applied to each position of the signal (convolved) to compute a transformation of the input signal

Input signal

This is a given input, in theory size is not fixed, a convolution can be applied on arbitrary size inputs

2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

Simple filter of dimension 3

-1	2	-3
a_1	a_2	a_3

- The size of the filter is fixed
- In practice, the values in the filter are learned => parameters of the model
- Can have an additional bias/intercep term

FILTERS AND CONVOLUTIONS

Input signal

2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6$

FILTERS AND CONVOLUTIONS

Input signal

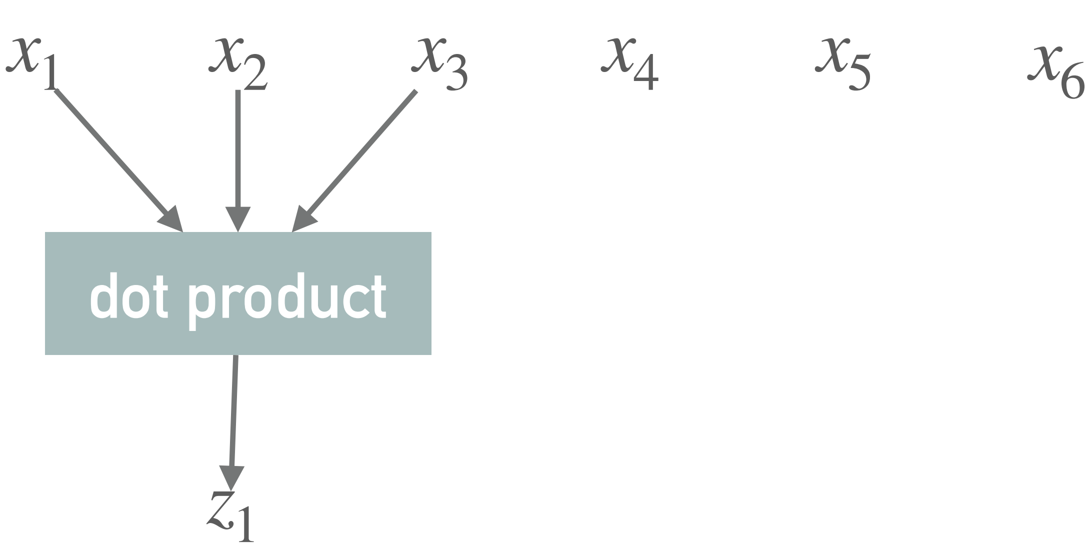
2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

FILTERS AND CONVOLUTIONS

Input signal

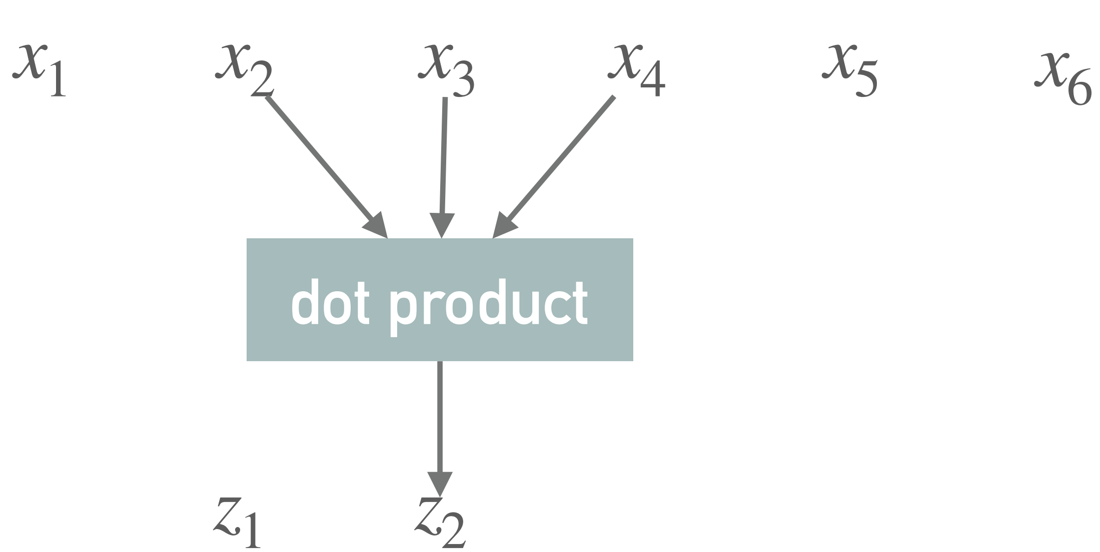
2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

FILTERS AND CONVOLUTIONS

Input signal

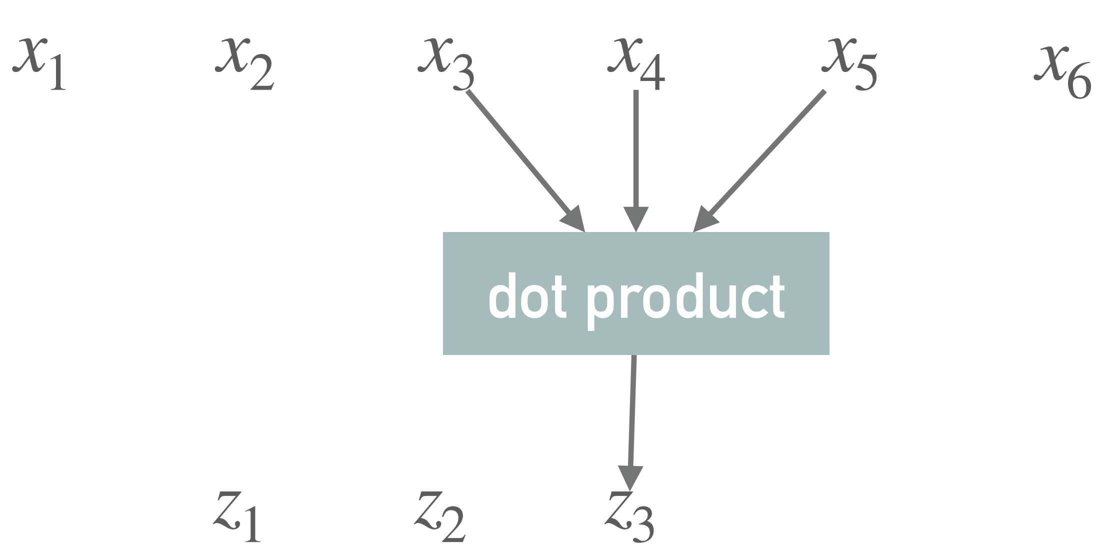
2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_3 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

FILTERS AND CONVOLUTIONS

Input signal

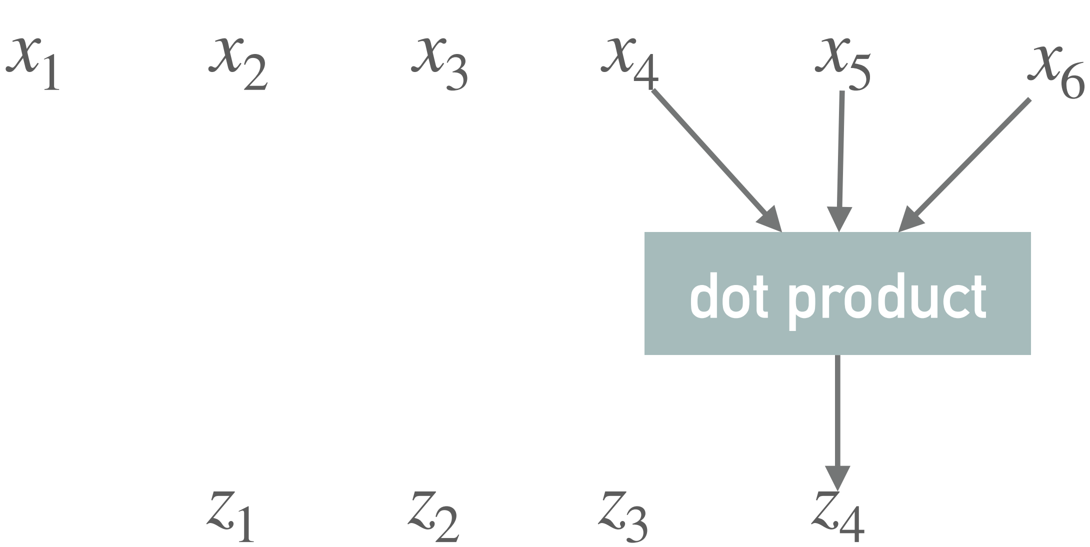
2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_3 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

$$z_4 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

FILTERS AND CONVOLUTIONS

Input signal

2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Filter

-1	2	-3
a_1	a_2	a_3

Convolution

Apply the filter on the input signal using a sliding window

x_1 x_2 x_3 x_4 x_5 x_6

z_1 z_2 z_3 z_4

$$z_1 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_3 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

$$z_4 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

Output is "shorter" than input :(

PADDING

Motivation

We want the output to have the same size as the input

Unpadded input signal

2	-5	10	3	-2	1
x_1	x_2	x_3	x_4	x_5	x_6

Padded input signal

- Pad the signal at the left and right of the input signal
- Default value for padding is 0

Pad of size 1 on both sides

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

PADDING

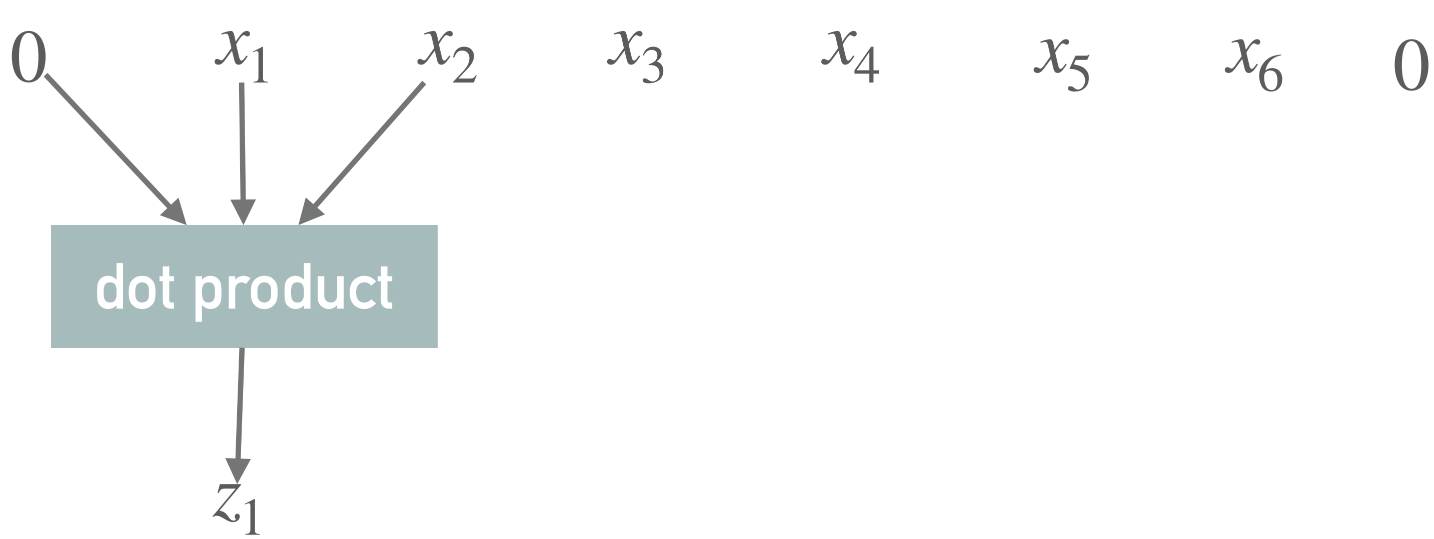
Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$



PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

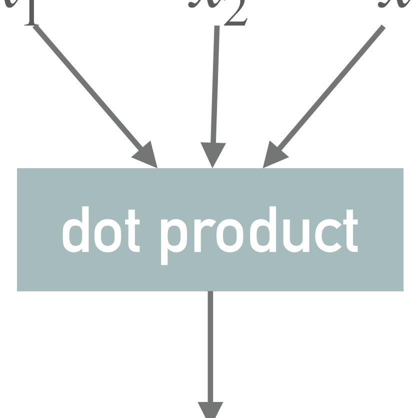
Convolution

Apply the filter on the input signal using a sliding window

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$

$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$



$z_1 \quad z_2$

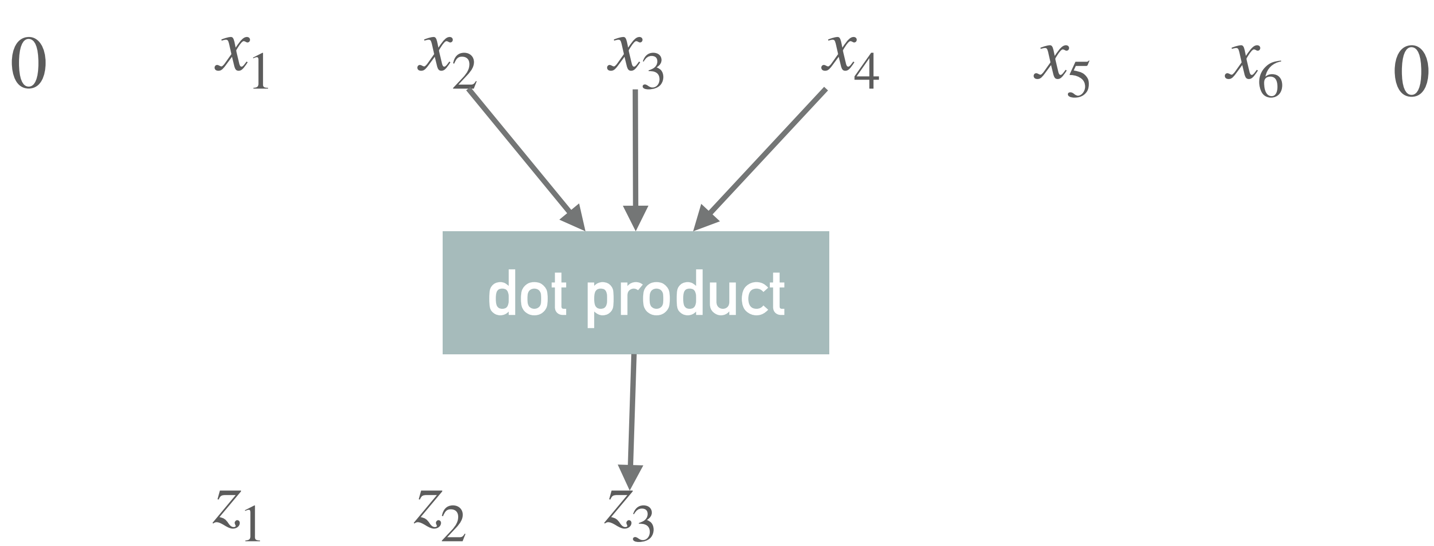
PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$

$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

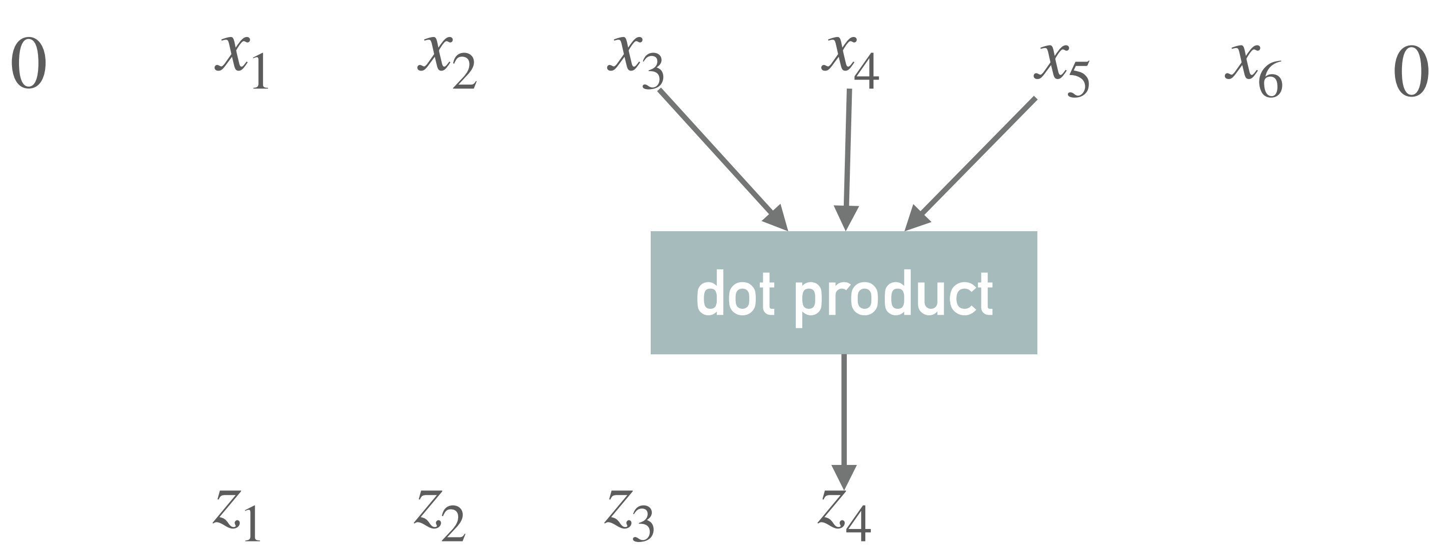
PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$

$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

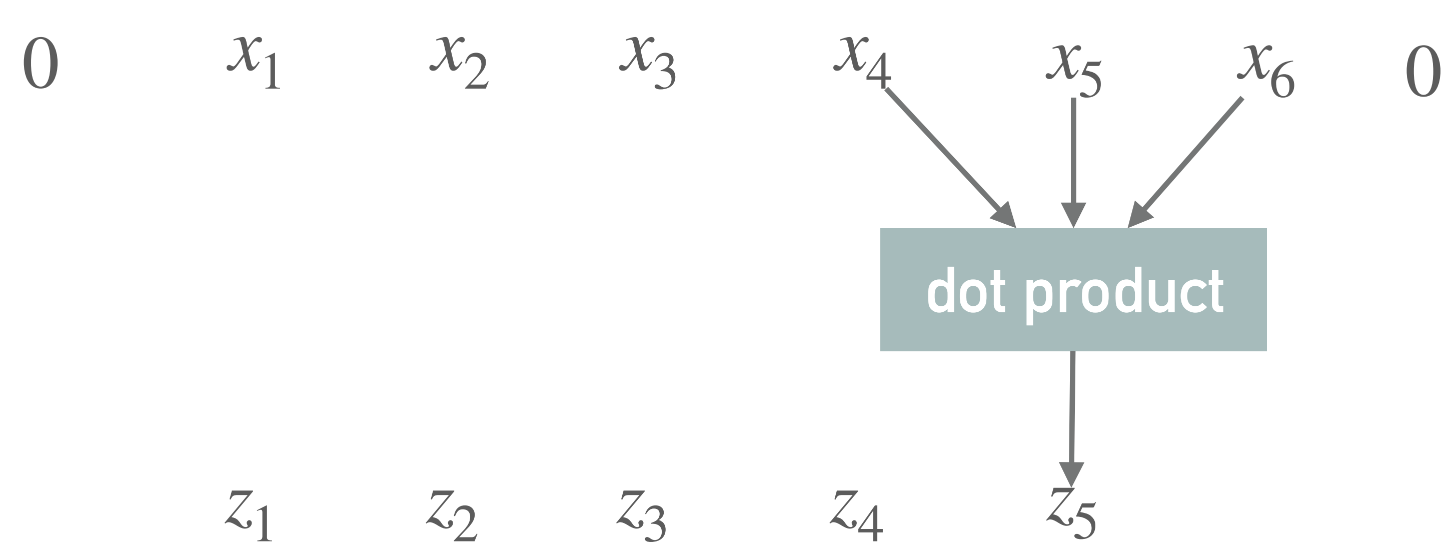
PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$

$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

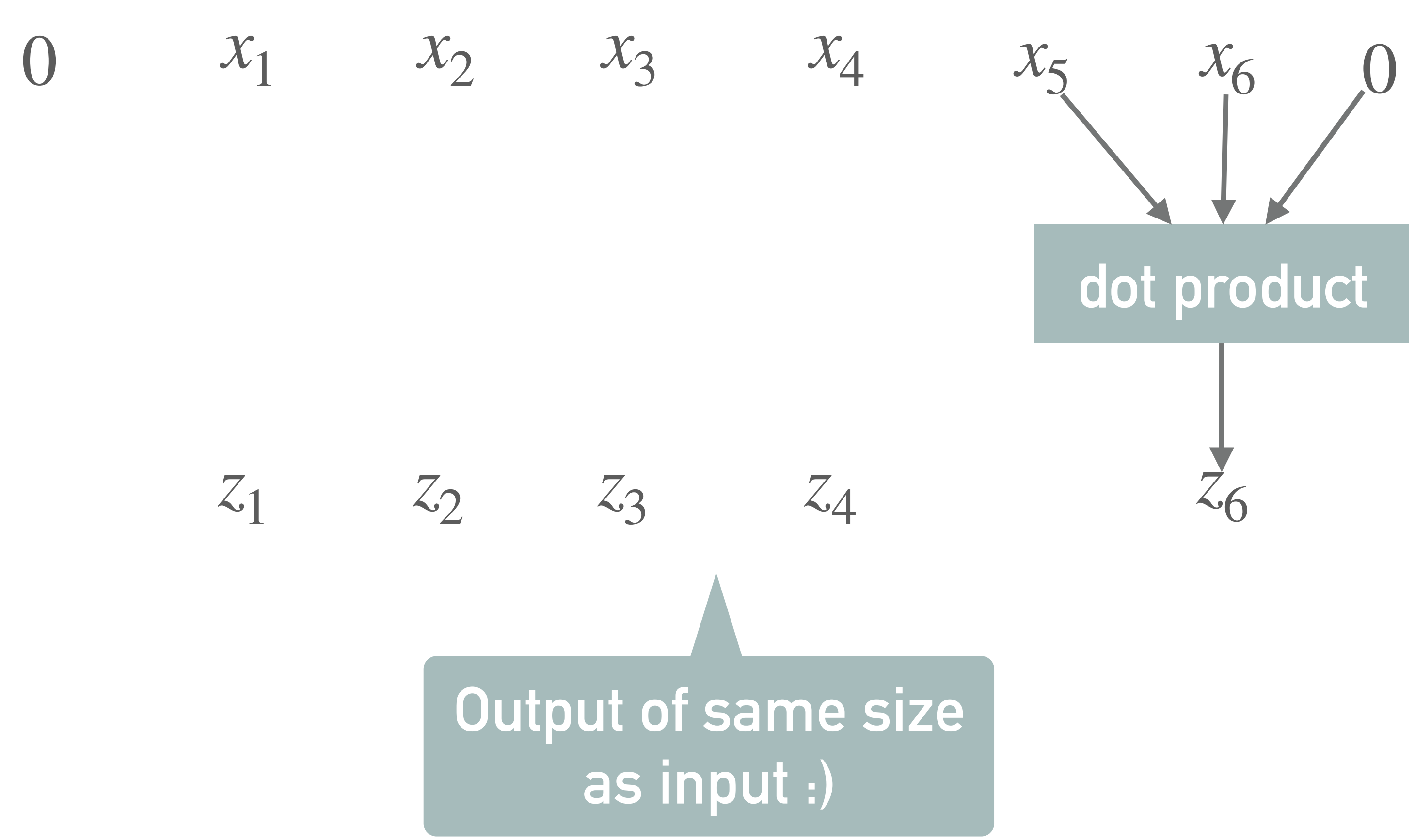
PADDING

Padded input signal

0	2	-5	10	3	-2	1	0
	x_1	x_2	x_3	x_4	x_5	x_6	

Convolution

Apply the filter on the input signal using a sliding window



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$

$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

PADDING

Filter

-1	2	-3	8	-5
a_1	a_2	a_3	a_4	a_5

If the filter is "larger",
we may want to increase padding

Padded input signal (pad size=2)

0	0	2	-5	10	3	-2	1	0	0
		x_1	x_2	x_3	x_4	x_5	x_6		

PADDING

Filter

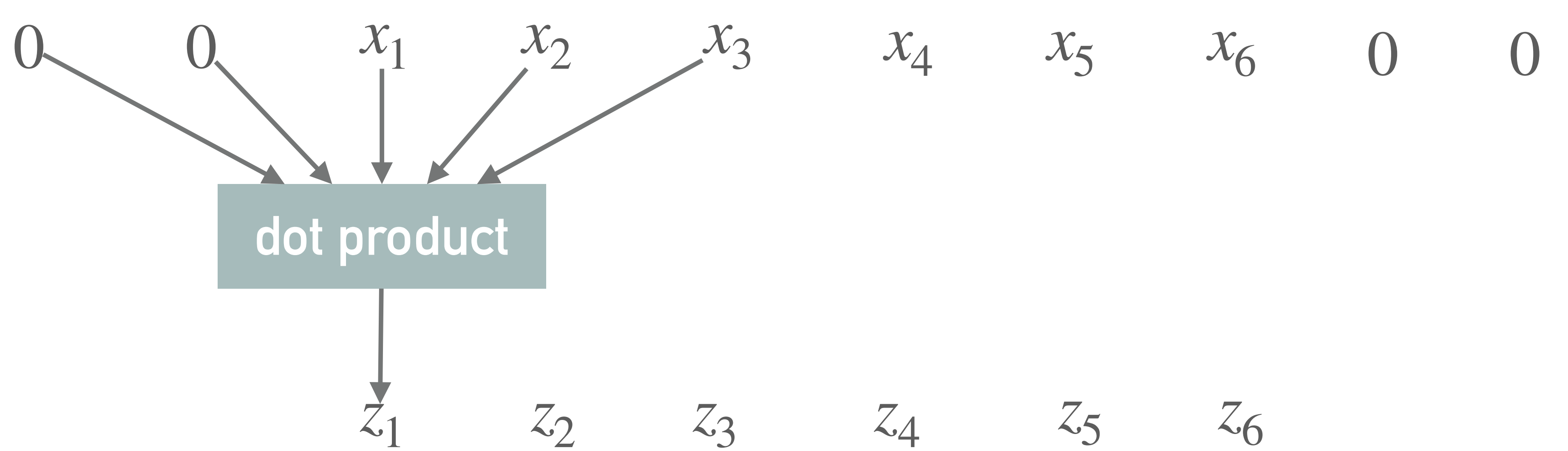
-1	2	-3	8	-5
a_1	a_2	a_3	a_4	a_5

If the filter is "larger",
we may want to increase padding

Padded input signal (pad size=2)

0	0	2	-5	10	3	-2	1	0	0
		x_1	x_2	x_3	x_4	x_5	x_6		

Convolution



$$z_1 = a_1 \times x_0 + a_2 \times 0 + a_3 \times x_1 + a_4 \times x_2 + a_5 \times x_3 + b$$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

STRIDE

Definition

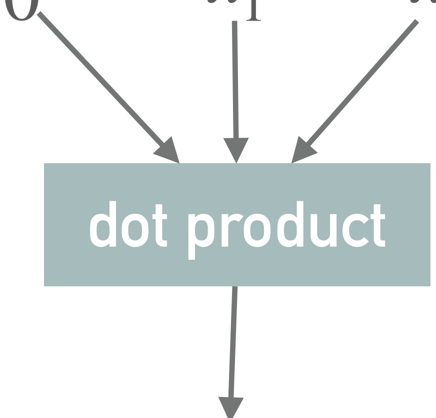
The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$



z_1

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

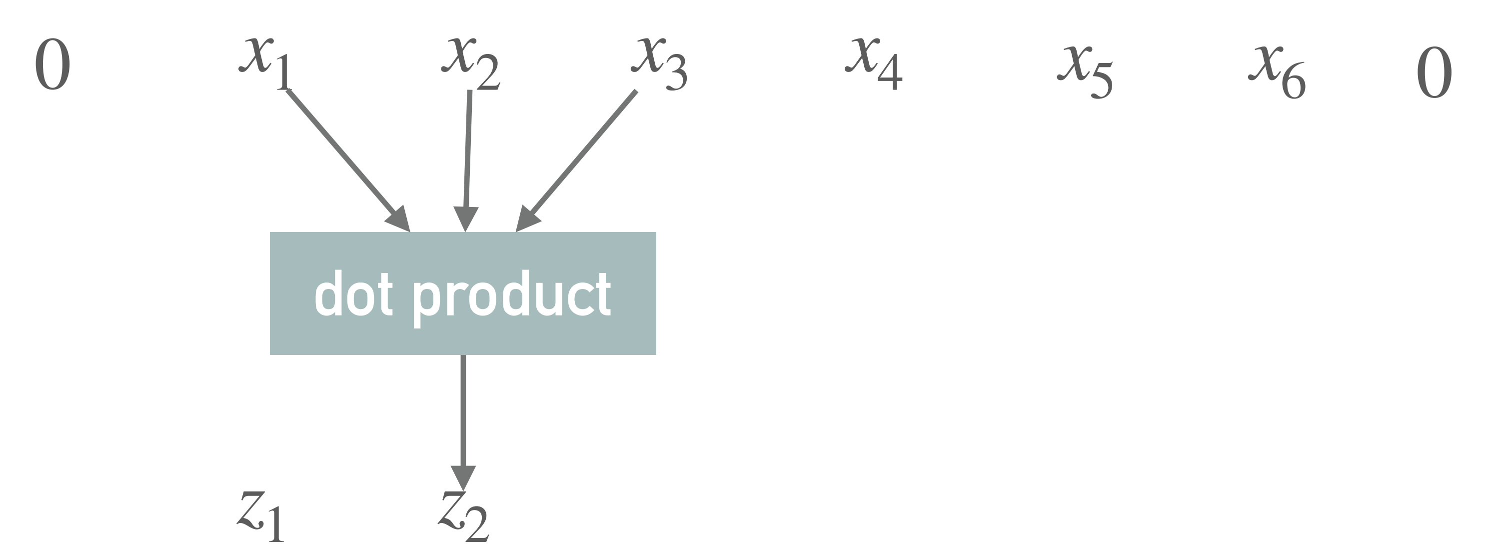
STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

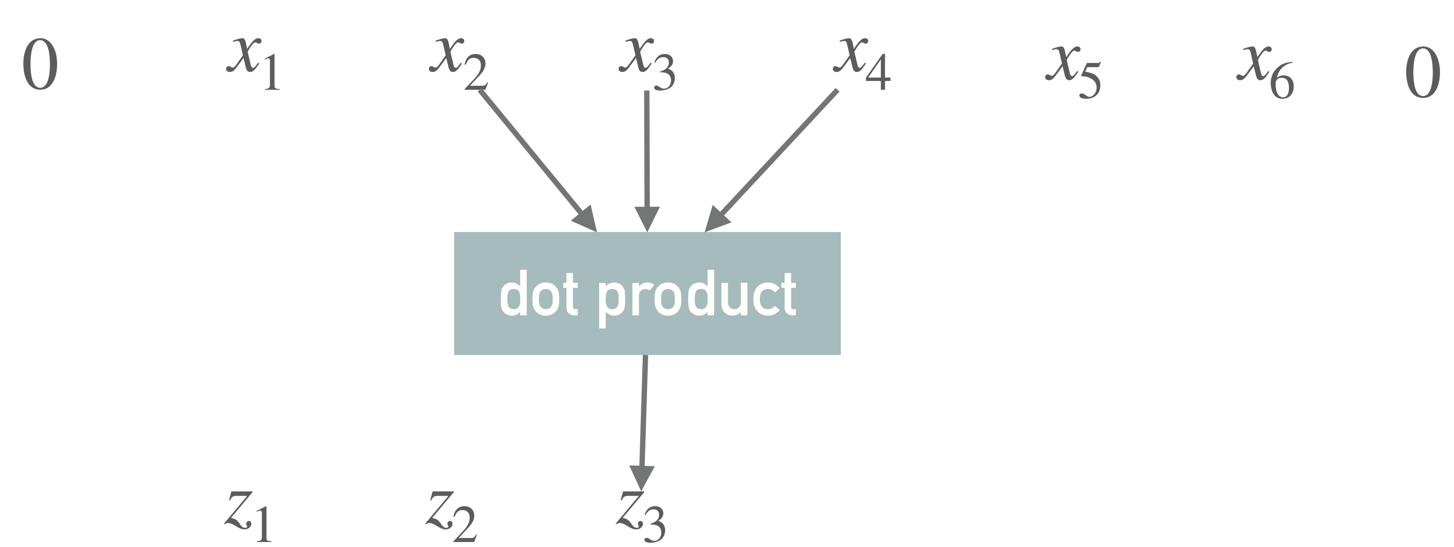
STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

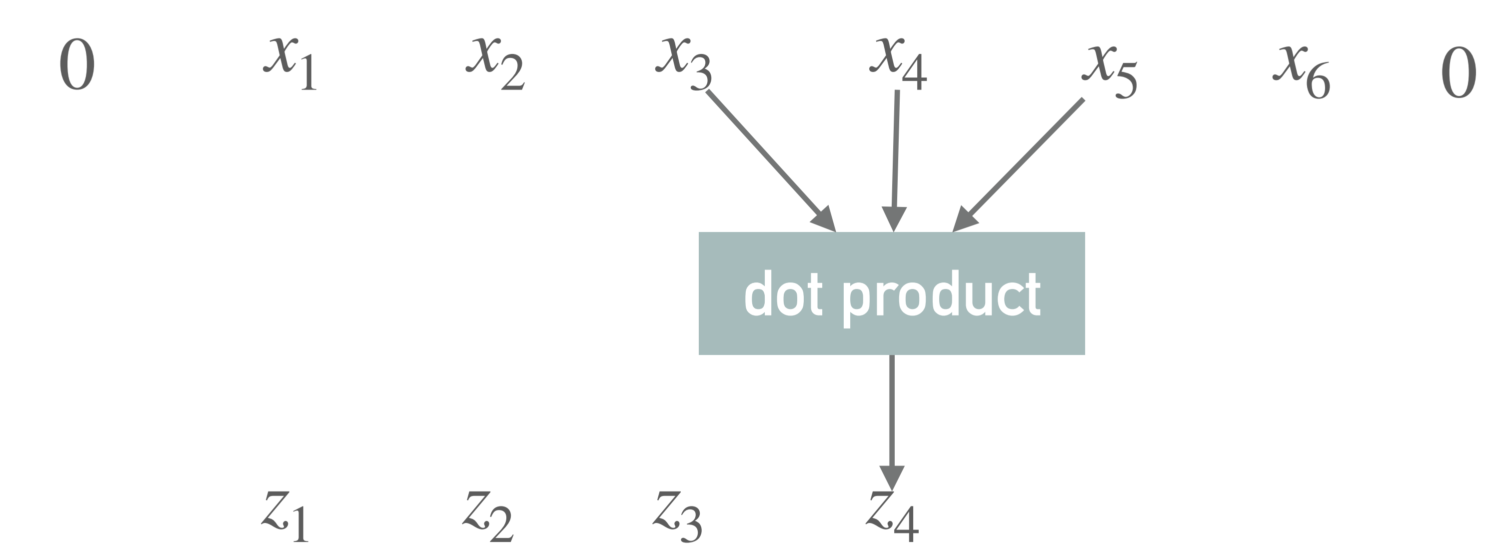
STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

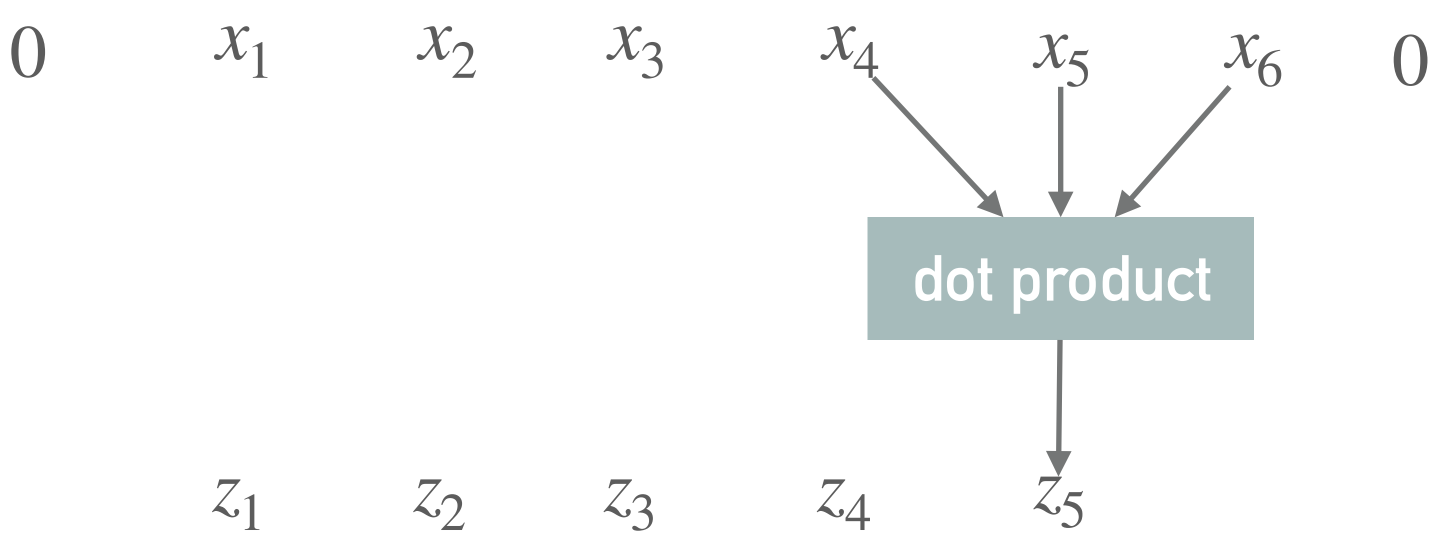
STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

Stride of 2

0 x_1 x_2 x_3 x_4 x_5 x_6 0

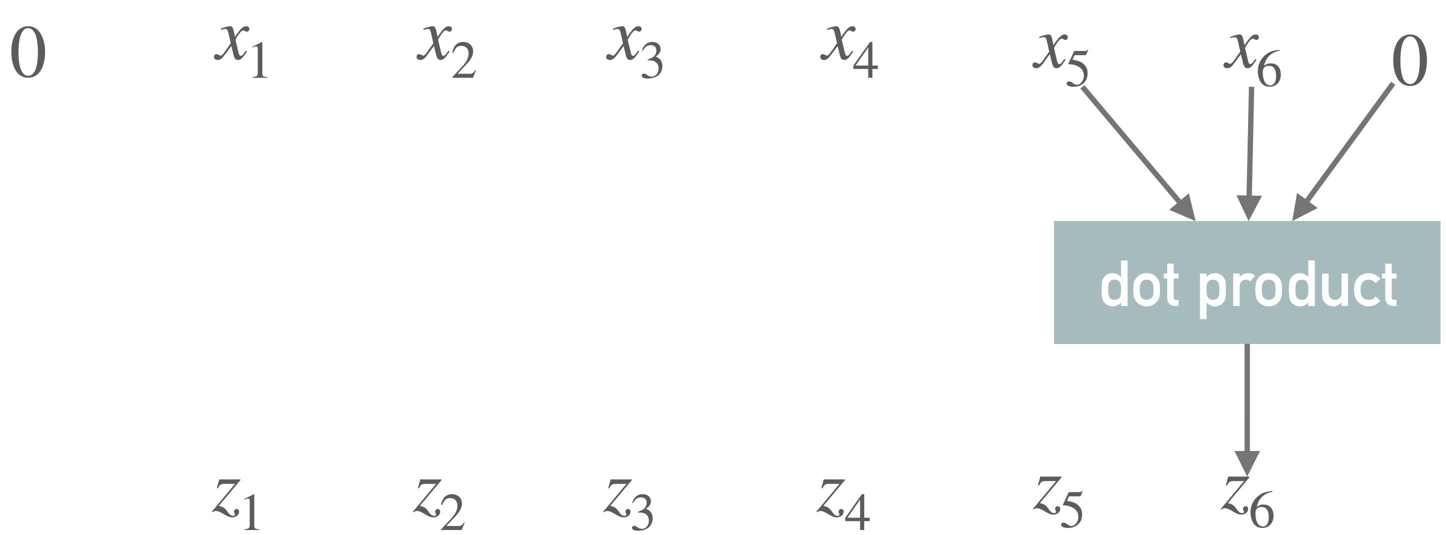
STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1



$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

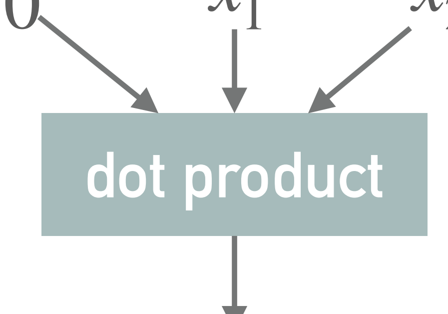
$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$



z_1

$$z_1 = a_1 \times 0 + a_2 \times x_1 + a_3 \times x_2 + b$$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

dot product

$z_1 \quad z_2$

$$z_1 = a_1 \times 0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

dot product

$z_1 \quad z_2 \quad z_3$

$$z_1 = a_1 \times 0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_3 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

STRIDE

Definition

The **stride** is the number of positions you move the filter between two applications.

=> the larger the stride, the smaller the output will be!

Stride of 1

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6$

$$z_1 = a_1 \times x_0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + b$$
$$z_3 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_4 = a_1 \times x_3 + a_2 \times x_4 + a_3 \times x_5 + b$$
$$z_5 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$
$$z_6 = a_1 \times x_5 + a_2 \times x_6 + a_3 \times x_0 + b$$

Stride of 2

$0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad 0$

$z_1 \quad z_2 \quad z_3$

$$z_1 = a_1 \times 0 + a_2 \times x_1 + a_3 \times x_2 + b$$
$$z_2 = a_1 \times x_2 + a_2 \times x_3 + a_3 \times x_4 + b$$
$$z_3 = a_1 \times x_4 + a_2 \times x_5 + a_3 \times x_6 + b$$

POOLING

Objective

Reduce ("compress") the representation.

Main idea

- Compute **max** or **average/mean** over a fixed window
- No parameter for pooling layers
- Usually no padding
- As for filter, we need to define the size and stride of the pooling operation

Window/filter of size 2, stride of 2

x_1 x_2 x_3 x_4 x_5 x_6

POOLING

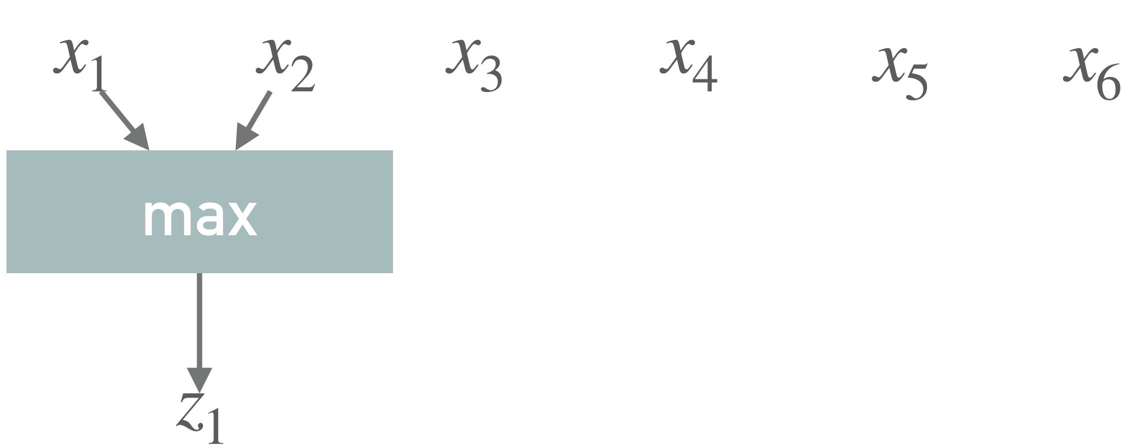
Objective

Reduce ("compress") the representation.

Main idea

- Compute **max** or **average/mean** over a fixed window
- No parameter for pooling layers
- Usually no padding
- As for filter, we need to define the size and stride of the pooling operation

Window/filter of size 2, stride of 2



$$z_1 = \max(x_1, x_2)$$

POOLING

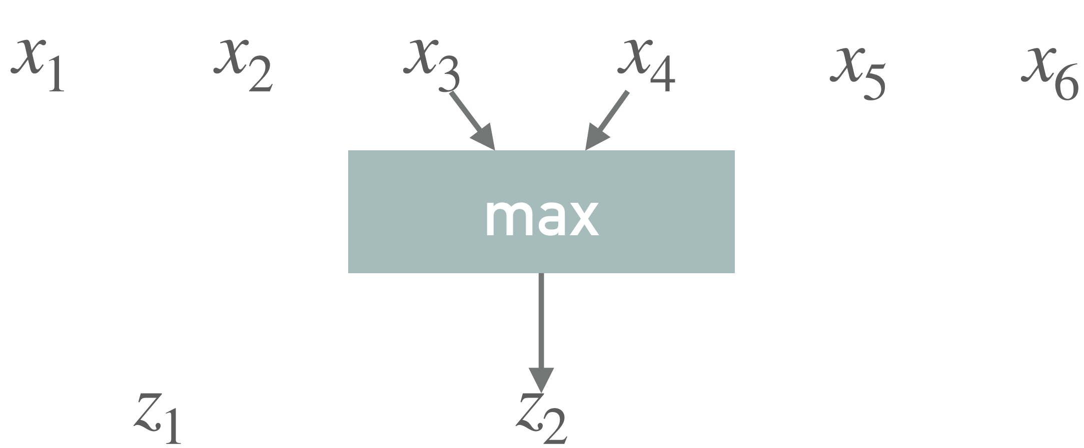
Objective

Reduce ("compress") the representation.

Main idea

- Compute **max** or **average/mean** over a fixed window
- No parameter for pooling layers
- Usually no padding
- As for filter, we need to define the size and stride of the pooling operation

Window/filter of size 2, stride of 2



$$z_1 = \max(x_1, x_2)$$

$$z_2 = \max(x_3, x_4)$$

POOLING

Objective

Reduce ("compress") the representation.

Main idea

- Compute **max** or **average/mean** over a fixed window
- No parameter for pooling layers
- Usually no padding
- As for filter, we need to define the size and stride of the pooling operation

Window/filter of size 2, stride of 2

$$z_1 = \max(x_1, x_2)$$

$$z_2 = \max(x_3, x_4)$$

$$z_3 = \max(x_5, x_6)$$

POOLING

Objective

Reduce ("compress") the representation.

Main idea

- Compute **max** or **average/mean** over a fixed window
- No parameter for pooling layers
- Usually no padding
- As for filter, we need to define the size and stride of the pooling operation

Window/filter of size 2, stride of 2

x_1 x_2 x_3 x_4 x_5 x_6

z_1 z_2 z_3

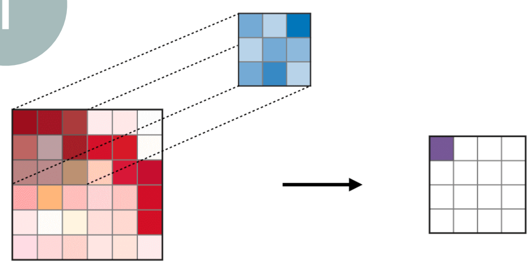
$$z_1 = \max(x_1, x_2)$$

$$z_2 = \max(x_3, x_4)$$

$$z_3 = \max(x_5, x_6)$$

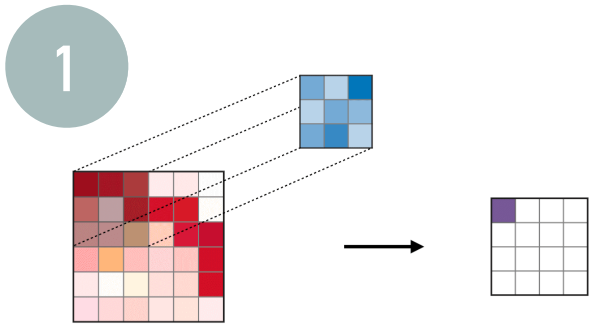
2 DIMENSION CONVOLUTIONS, STRIDE=1

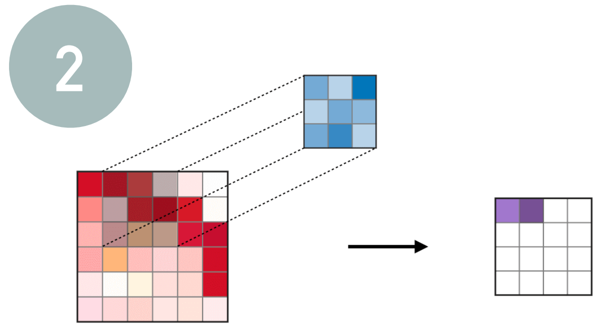
1



Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

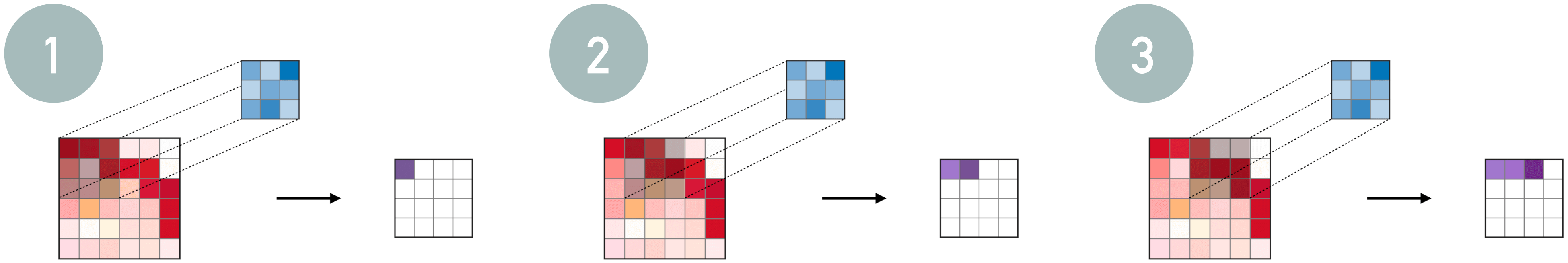
2 DIMENSION CONVOLUTIONS, STRIDE=1





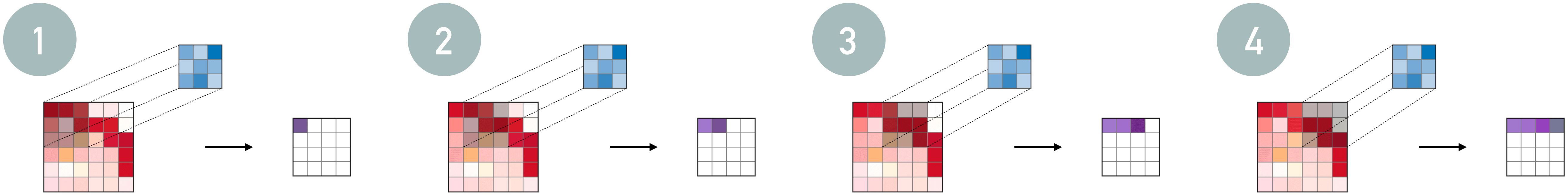
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



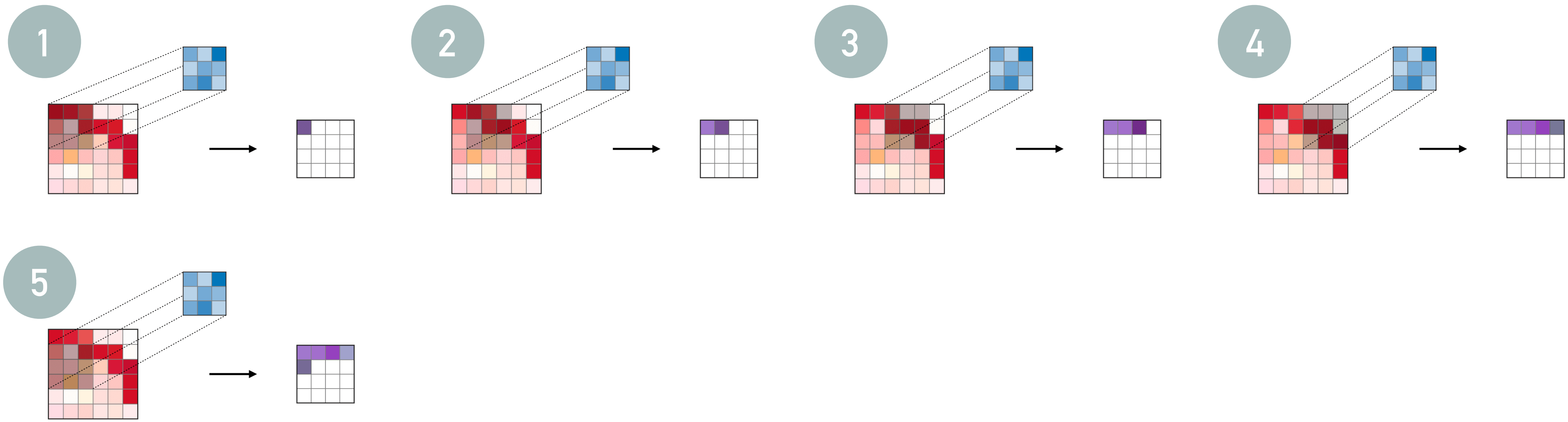
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



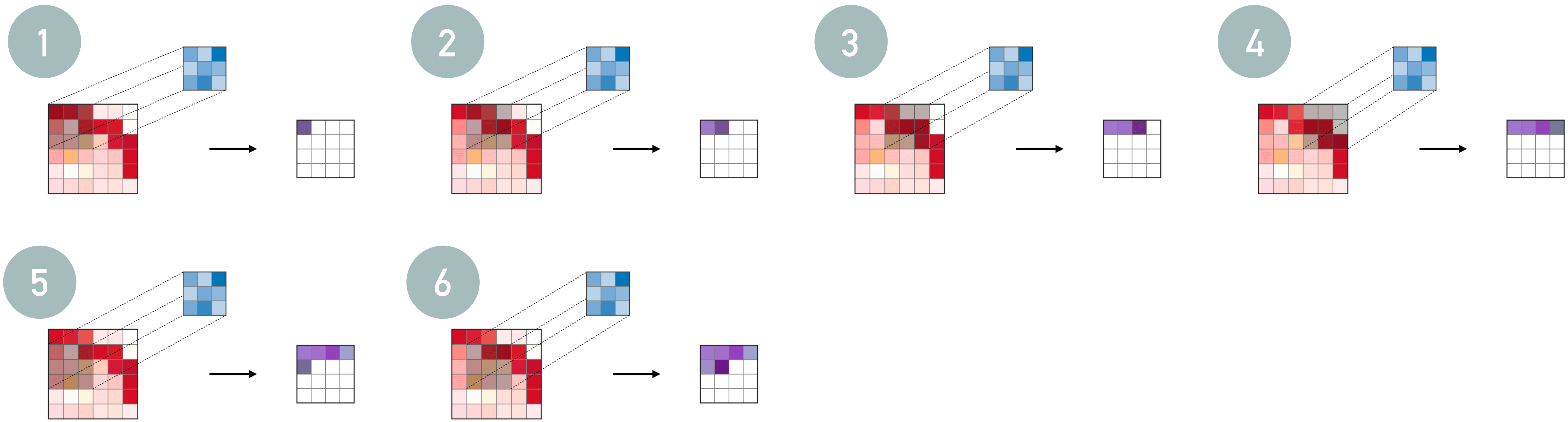
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



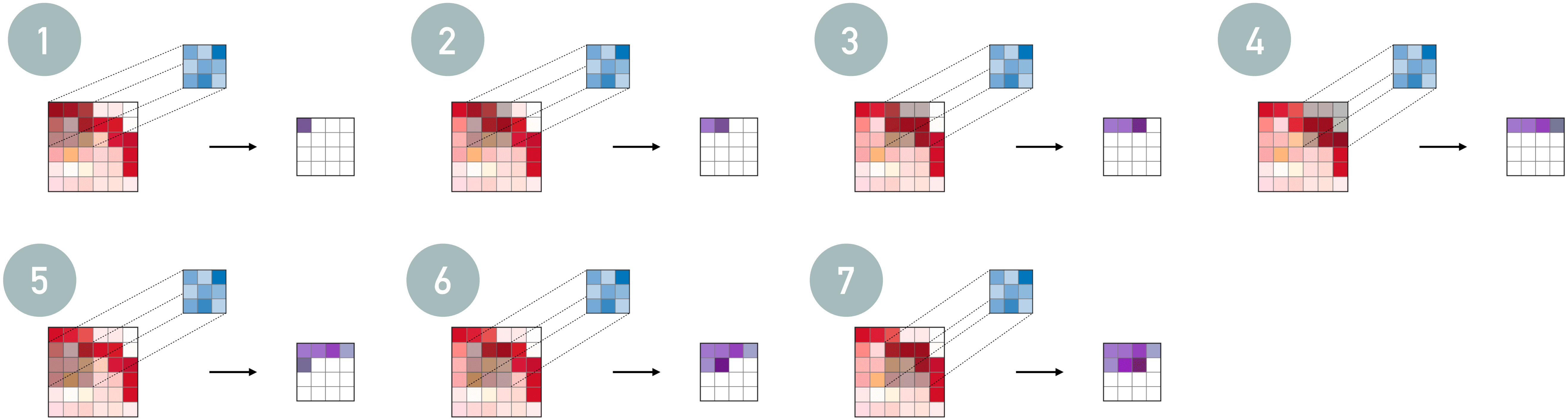
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



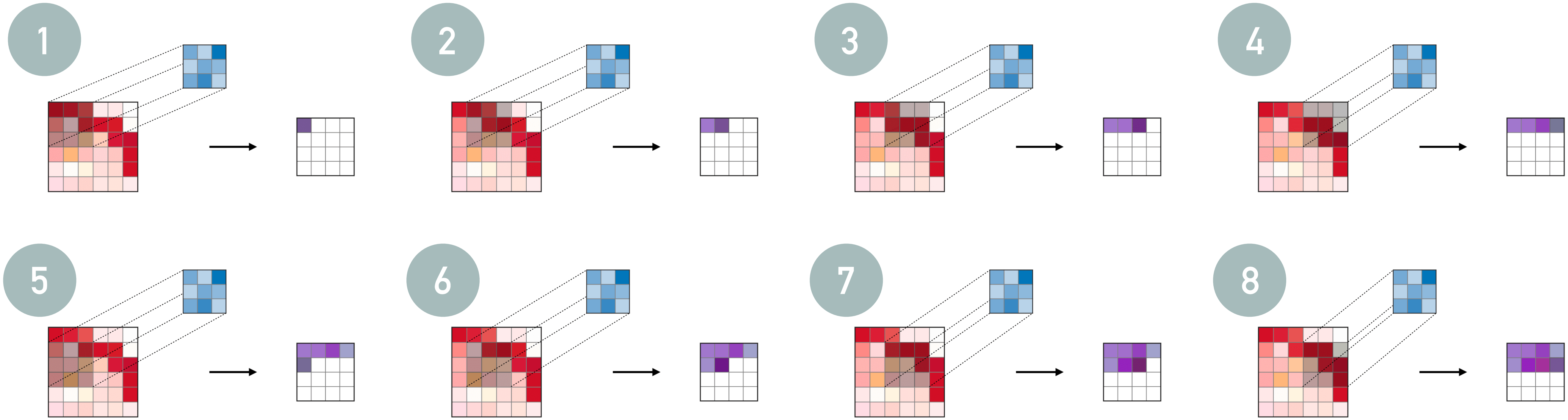
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



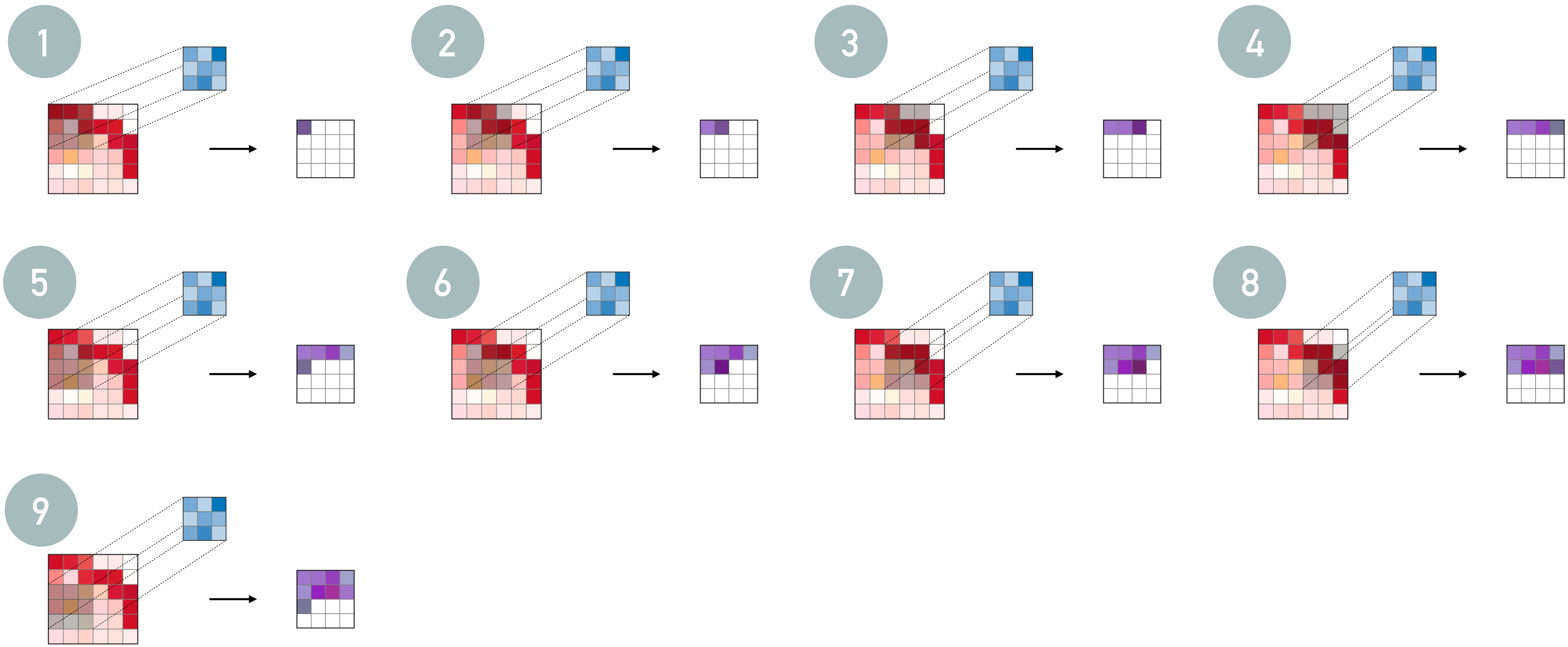
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



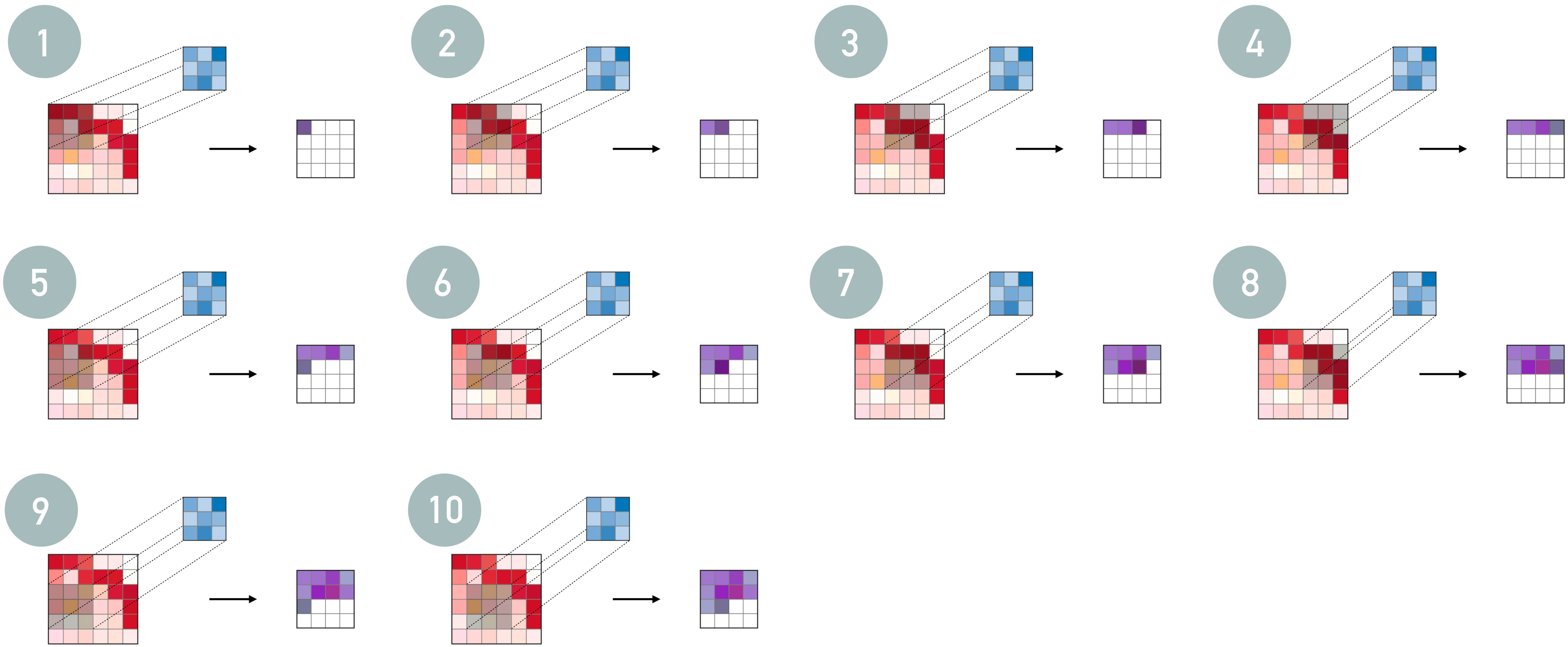
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



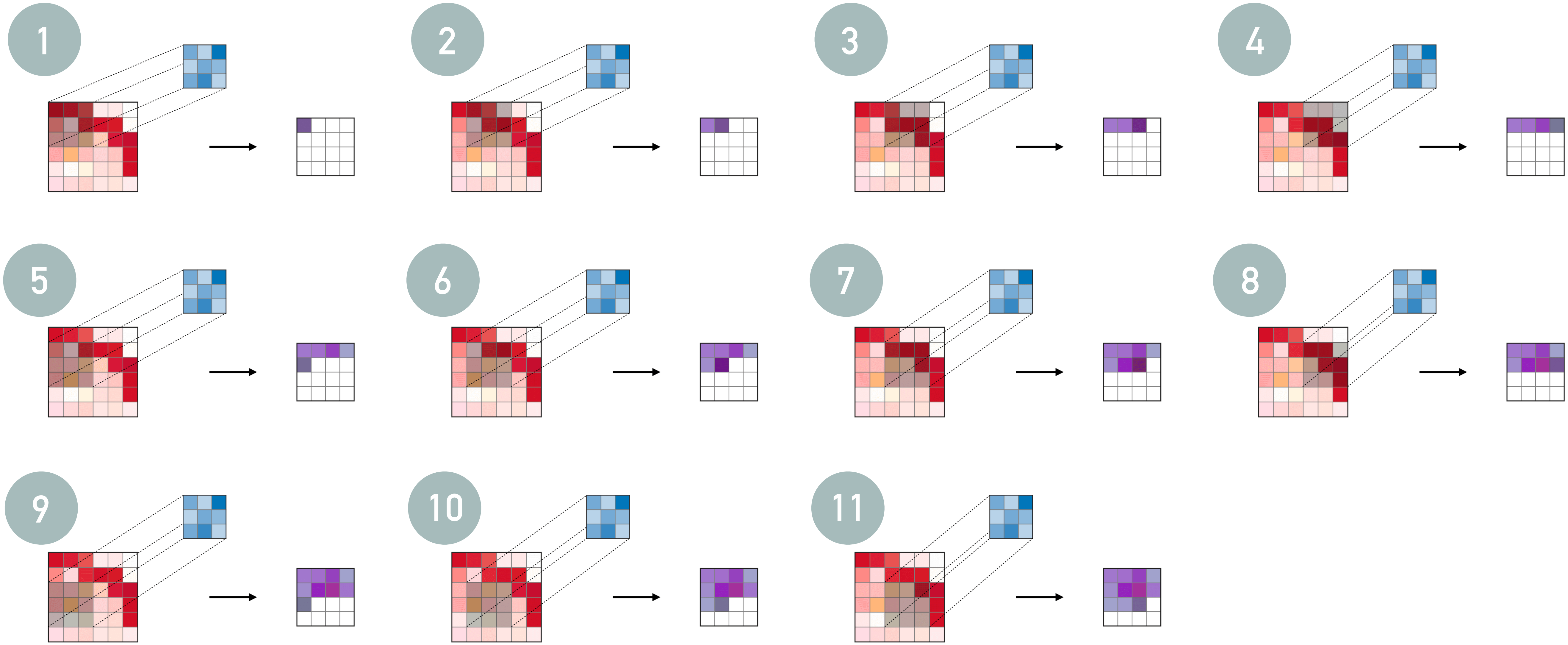
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



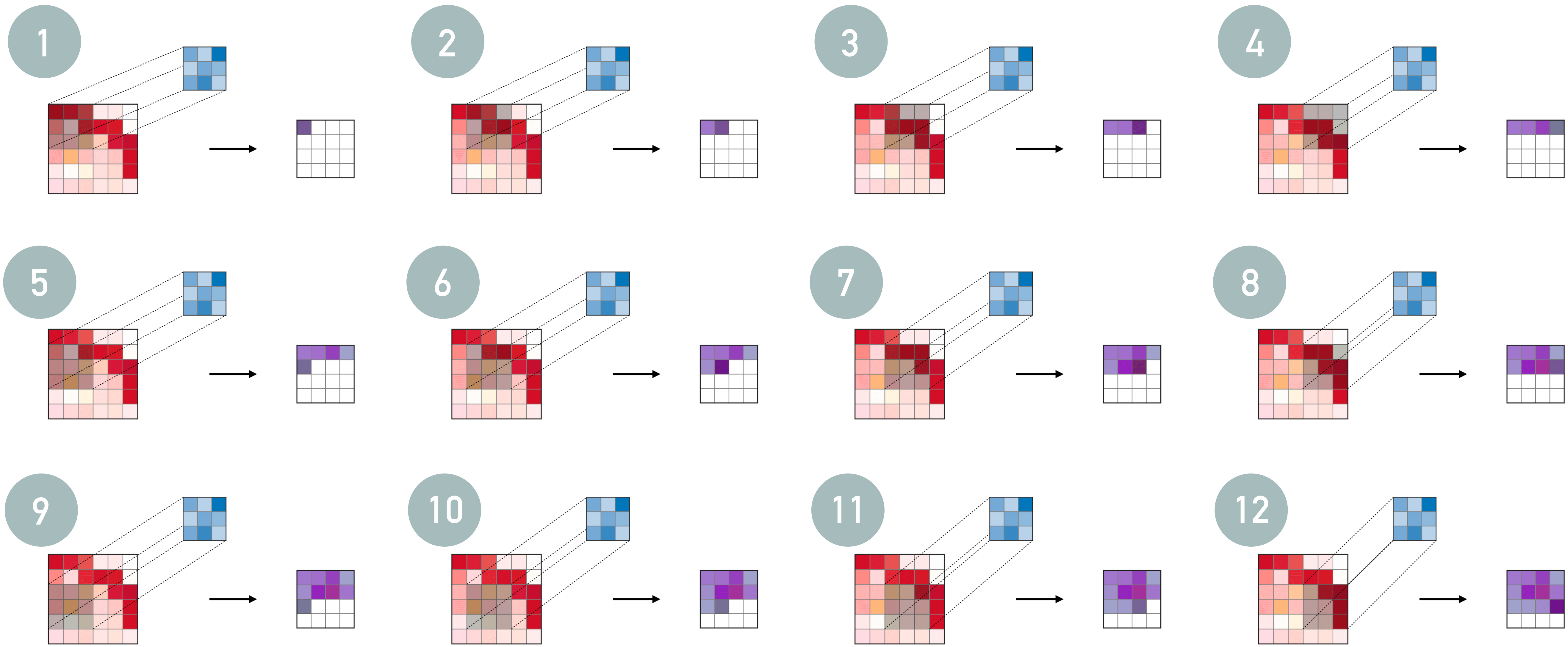
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



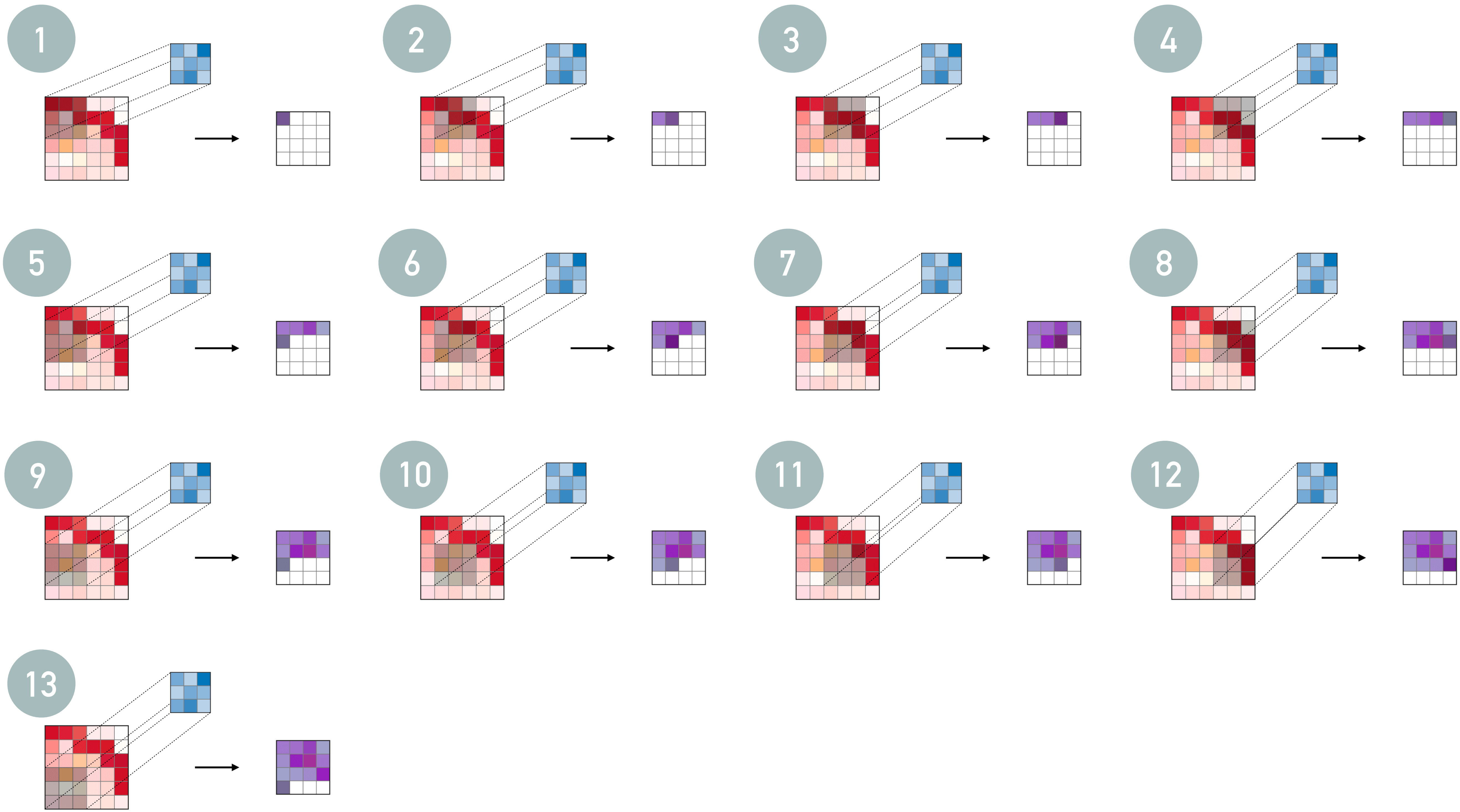
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



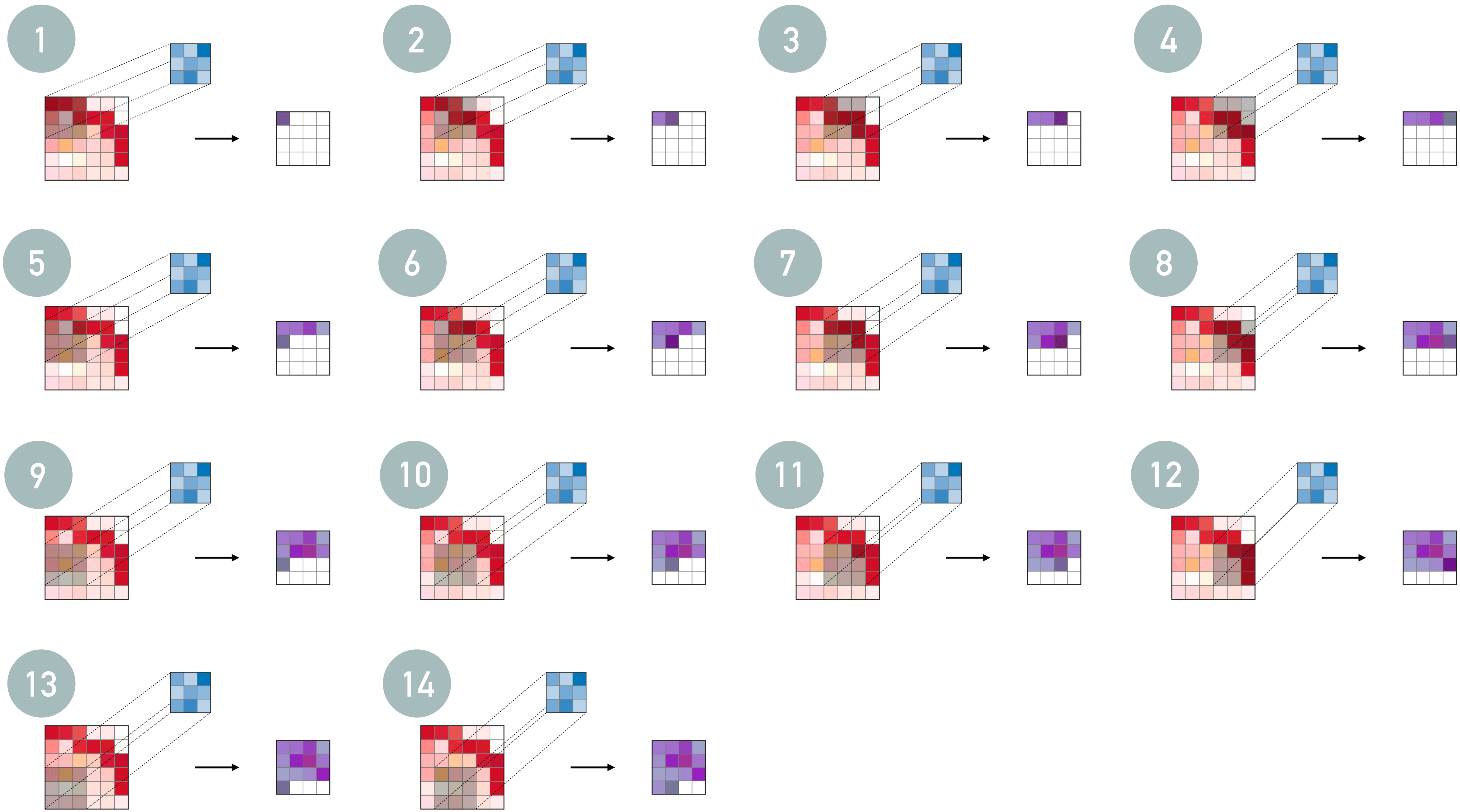
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



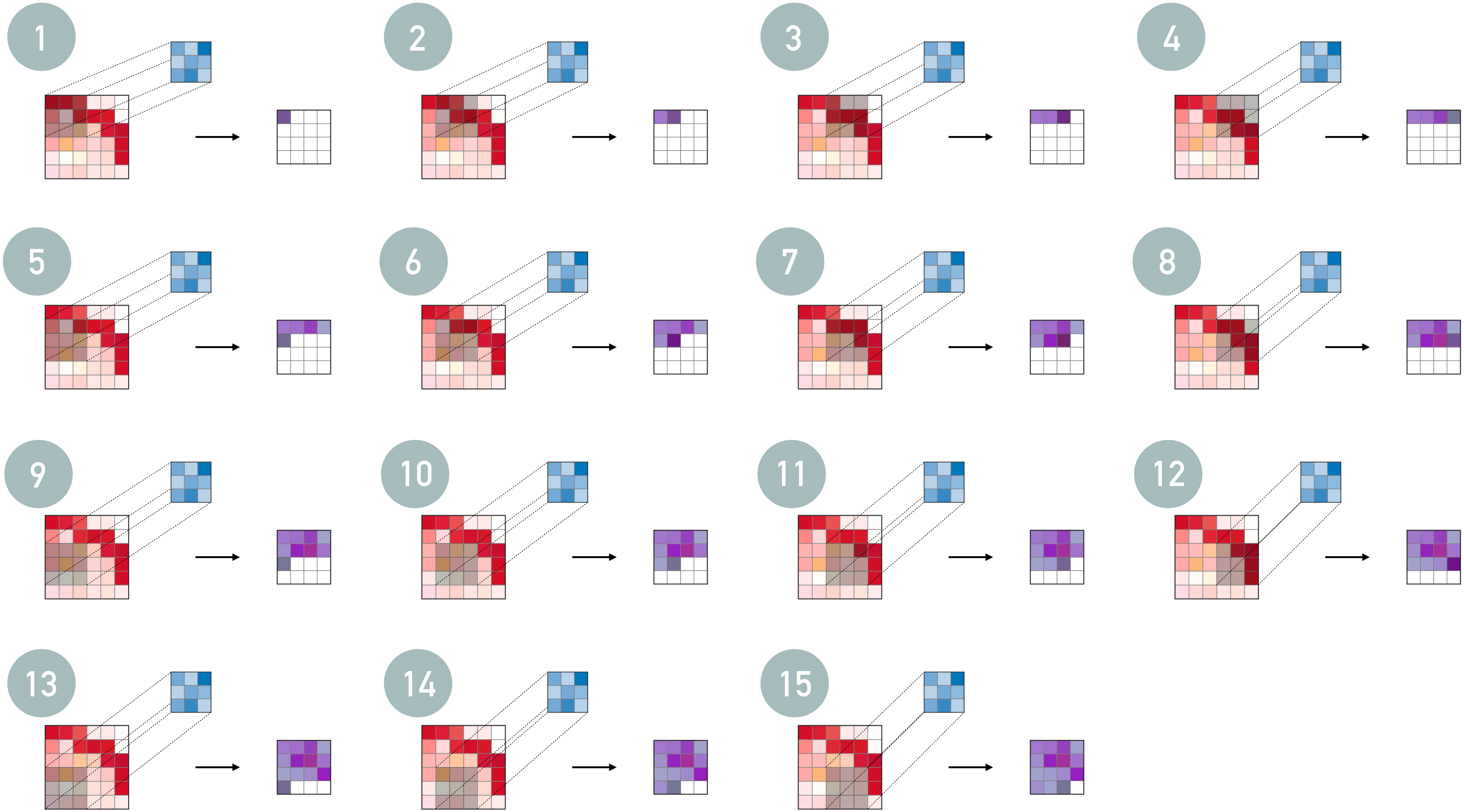
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



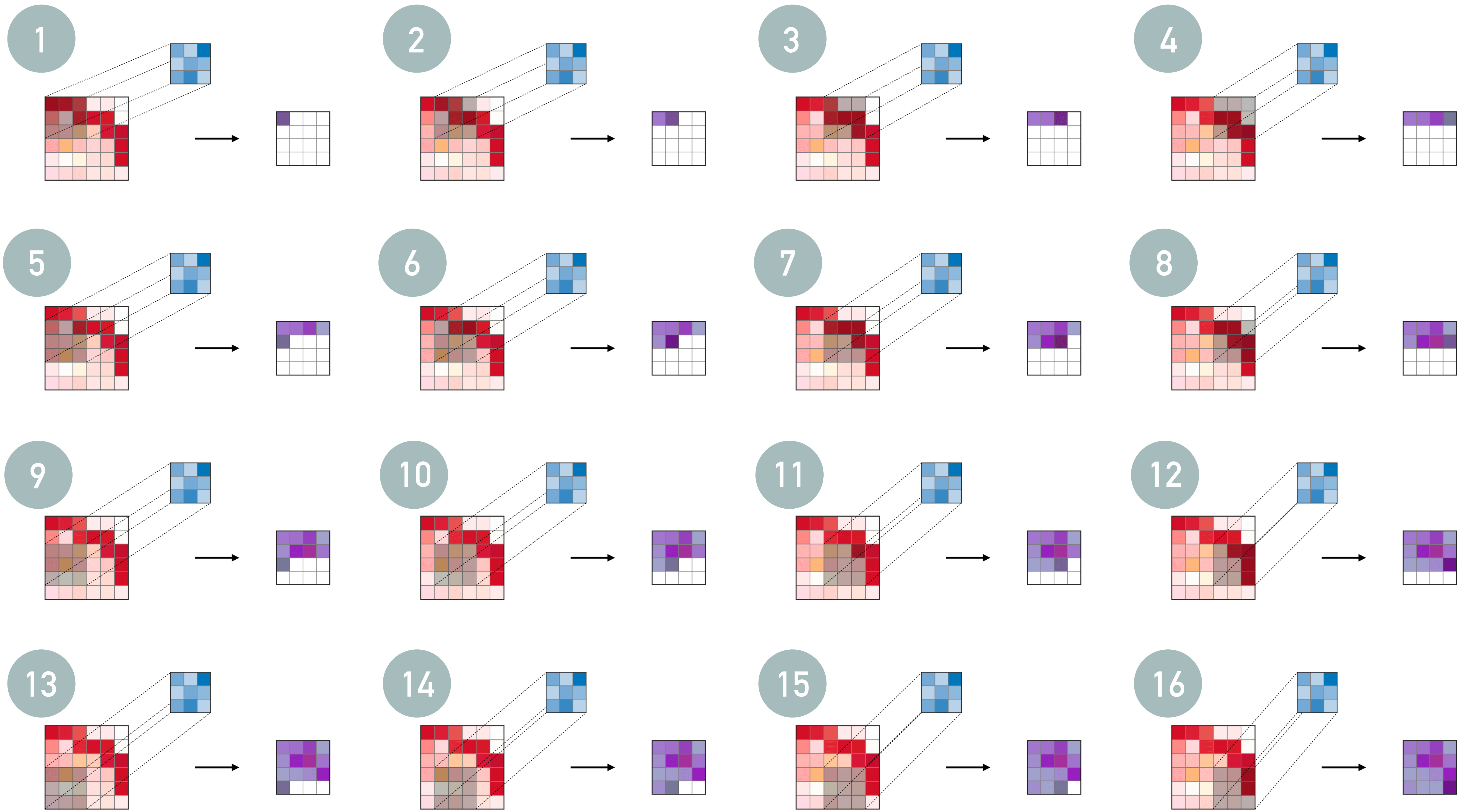
Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

2 DIMENSION CONVOLUTIONS, STRIDE=1



Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

INPUT DEPTH, CHANNELS

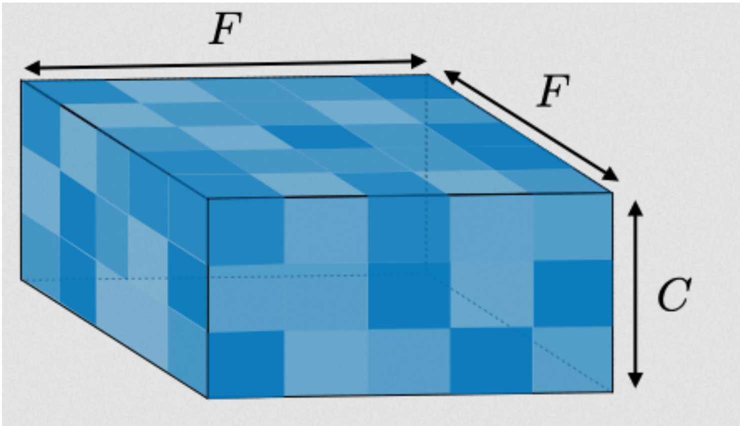
Example of input dimensions

Imaged have third dimension called **channel** to encode colors:

- Grayscale picture: 100 x 100 x 1
- Coloured picture : 100 x 100 x 3 (last dimension is RGB)

3D filter

- F: size of the filter
- C: number of channels in the input



The output associated with an application of this filter will be of size O x O x 1

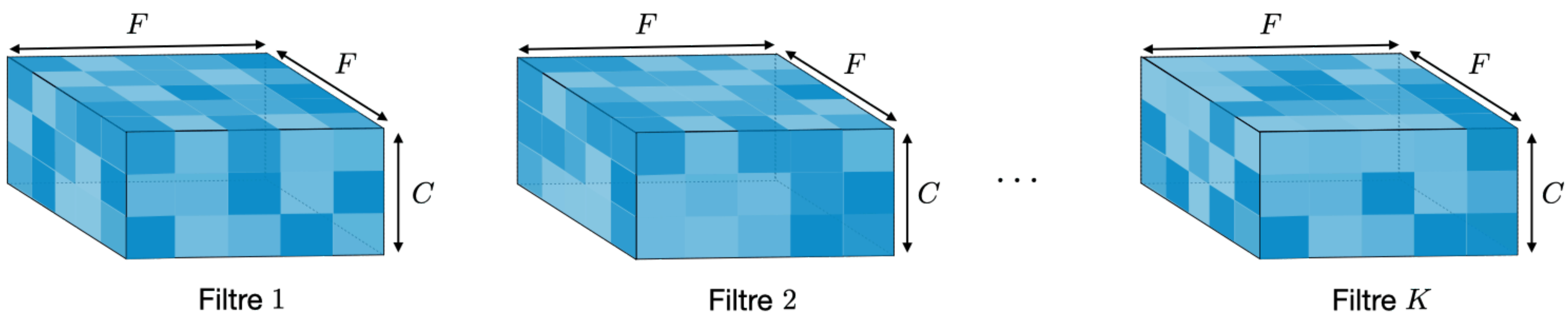
MULTIPLE FILTERS AND OUTPUT CHANNELS

Multiple filters

In practice, we use multiple filters:

- F: size of the filters (in theory we could have filters of different sizes)
- C: number of channels in the input
- K: number of filters

The output associated with an application of this filter will be of size O x O x K

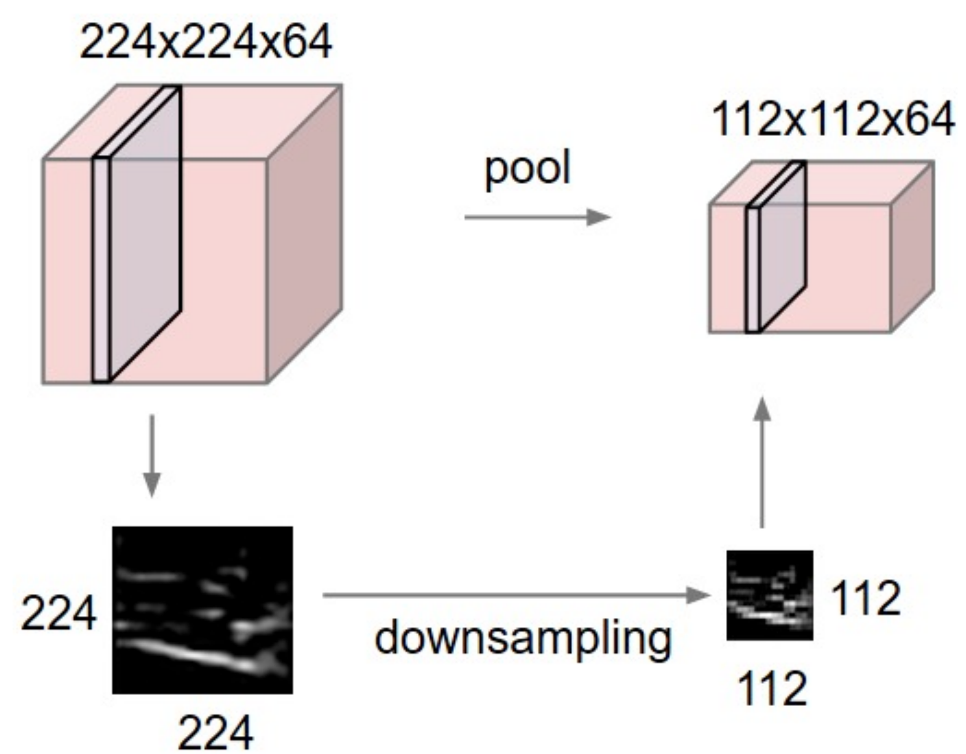


Warning

- Each filter have its own set of parameters
- They must be initialized randomly and "differently" to avoid symmetries

Illustrations from https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels

POOLING IN 2D



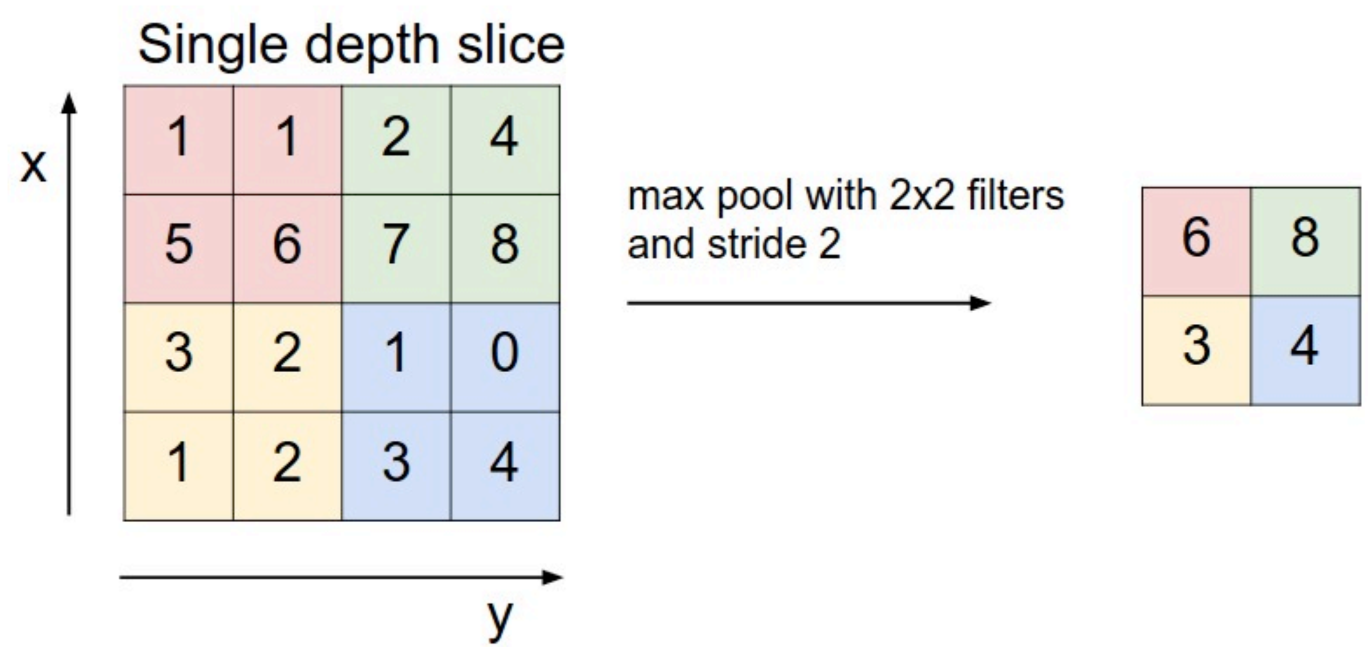


Illustration from https://cs231n.github.io/convolutional-networks/

FULL ARCHITECTURE

- ➤ Apply non-linear activation function after convolution layers
- ➤ At the end of the convolutional architecture,
  linearize the hidden representations and use it a input of a MLP

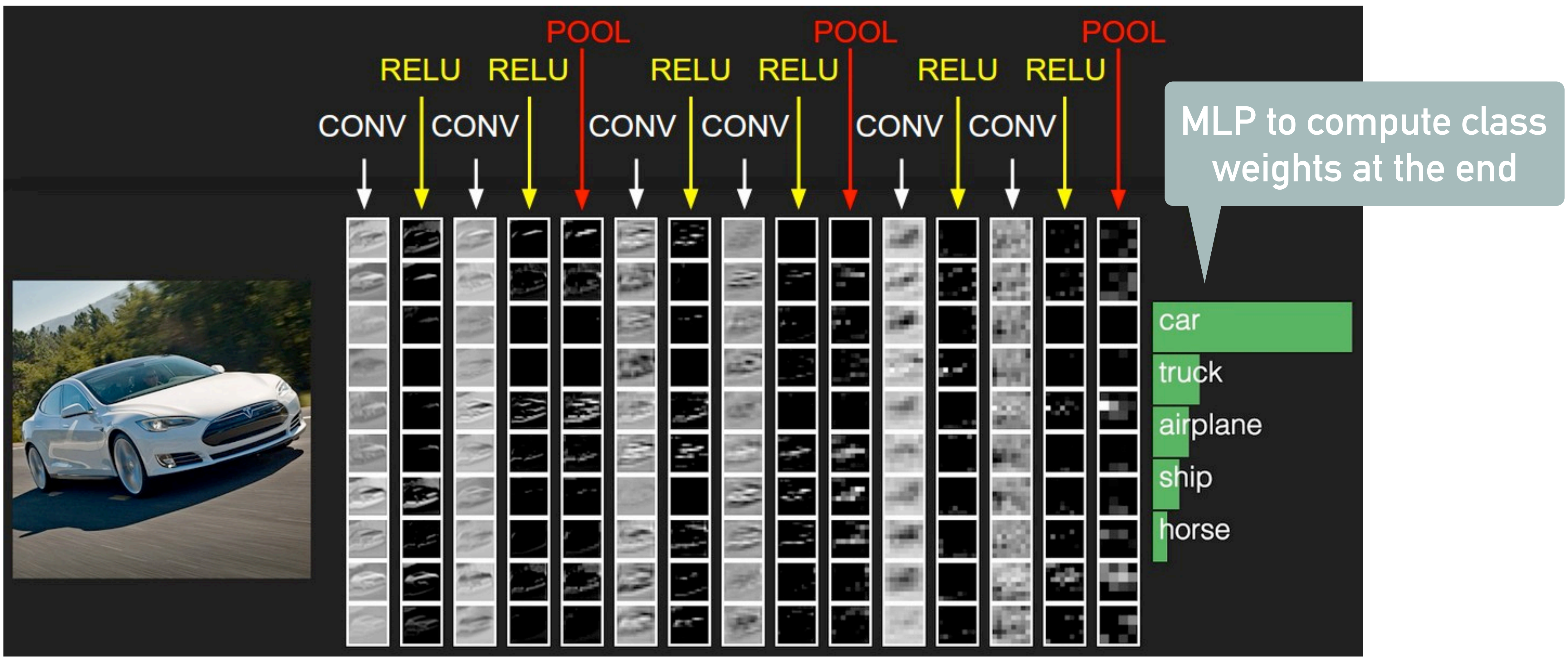


Illustration from https://cs231n.github.io/convolutional-networks/

DATA AUGMENTATION

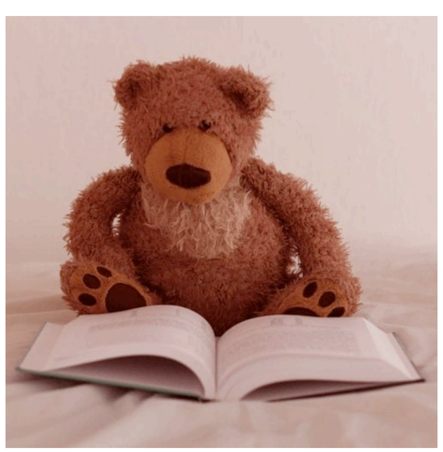
- Convolutions are equivariant to translation, but not to other transformations
- To learn equivariance/invariance to other transformations, just randomly modify the input while training

Original image

Flip

Rotation

Random crop



Color shift

Noise addition

Information loss

Contrast change

Illustration from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks

NEURAL NETWORK TRAINING

GRADIENT-BASED TRAINING

Neural network

Feature space

Parameterized function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ Output space

Parameters

Training

- Labeled example: features + « gold » answer
- Train set: $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
- Find parameters θ so that $f_\theta(x^{(i)}) \simeq y^{(i)}, \forall i$

End-to-end training

- In the old days: layer per layer training (in some kind of generative model)
- Nowadays: Train all parameters at the same time
  (+ unsupervised pretraining in some cases)

Testing / evaluation

- Test if the model <u>generalizes</u> to unseen data (i.e. disjoint set from the train set)

LOSS FUNCTION

Intuition

- Compare the output with the gold output (i.e. the expected output)
- The loss must be minimized (& bounded below by 0)
- Must be related to the evaluation function, but often slightly different

Learning objective

$$\theta^* = \text{argmin}_\theta \quad \frac{1}{n} \quad \sum_{i=1}^{n} \quad l(\ y^{(i)}, f_\theta(\mathbf{x}^{(i)})\ )$$

- <u>Modern machine learning is optimization</u>

In the course notations, this should be the output of the score function

GRADIENT DESCENT

Problem

Solve: $\min_{\theta} g(\theta)$

Intuition

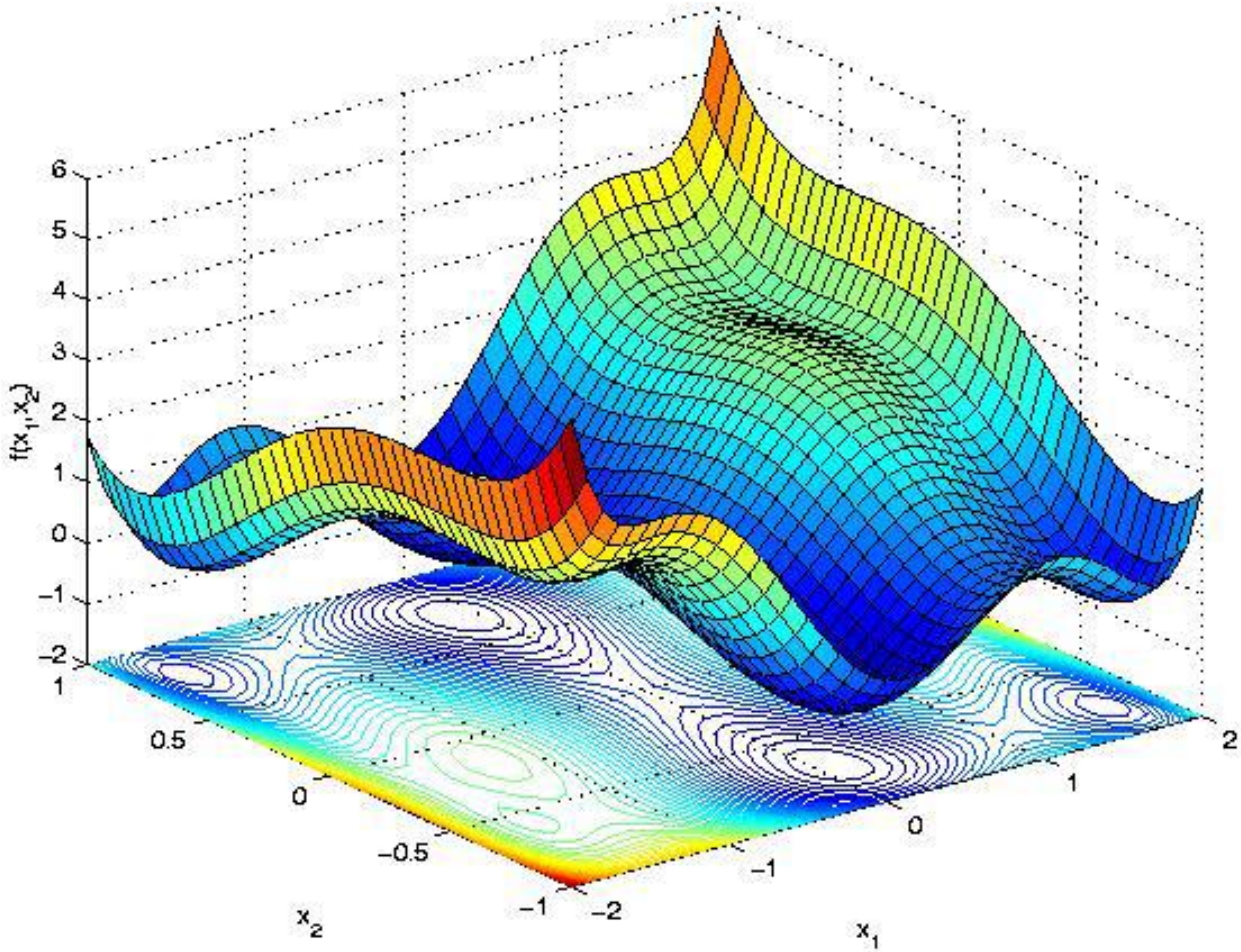
- All you can compute: evaluate the function and its gradient at a given point
- You can use gradient information to see in which direction the function is decreasing
- Therefore: just make a small step in this direction!
- In this course we won't differentiate between gradient and sub-gradient
- In deep learning, it is usual to rely on stochastic gradient descent with "large" minibatch size

Formally

- Choose an initial point randomly: $\theta^{(0)}$
- Make T iterations/steps: $\theta^{(t+1)} = \theta^{(t)} - \eta \times \nabla_{\theta} g(\theta)$

Stepsize

NON-CONVEX FUNCTION ILLUSTRATION



Many local minima! Do we care? NO

TRAIN/DEV/TEST

Parameters vs. hyper-parameters

- Parameters: the parameters of the function, which are learned during training
- Hyper-parameters: the parameters of the training algorithm and the neural architecture choice (number of layers, hidden representation dimensions, …)

Three datasets

- Train set:
  Used to compute the objective and its gradient
- Development / validation set:
  Used during training to choose hyper-parameters and to know when to stop training
- Test set:
  Used to evaluate the model!