Convolutional Neural Networks

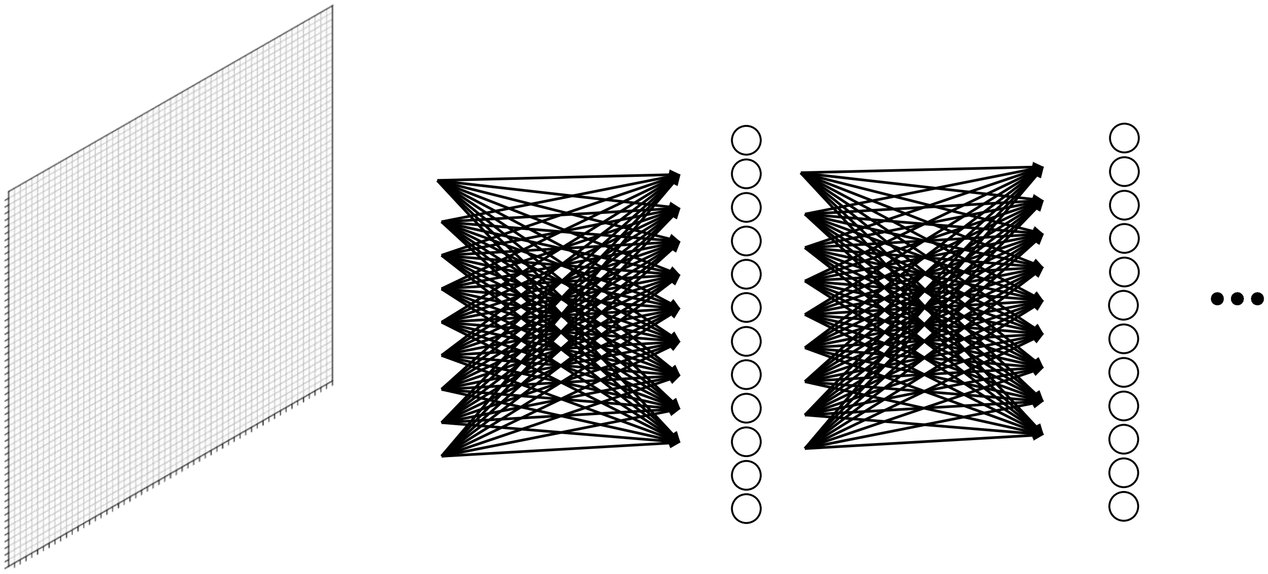
Neural networks inspired by the **visual cortex**

Faculty of Mathematics and Computer Science, University of Bucharest
and
Sparktech Software

Academic Year 2018/2019, 1st Semester

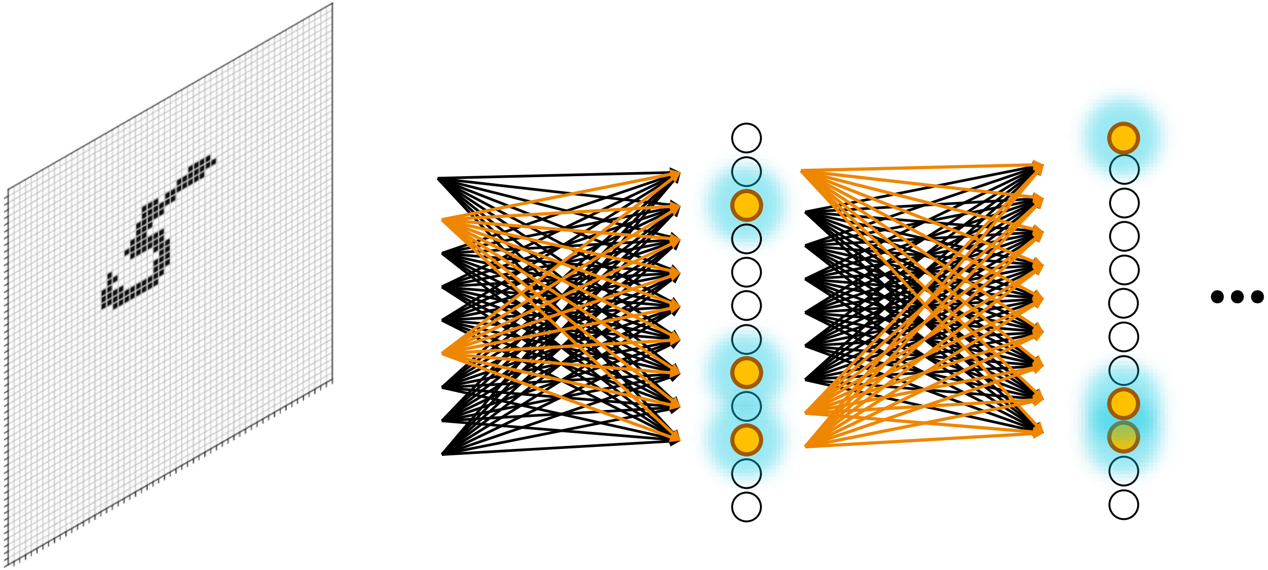
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).



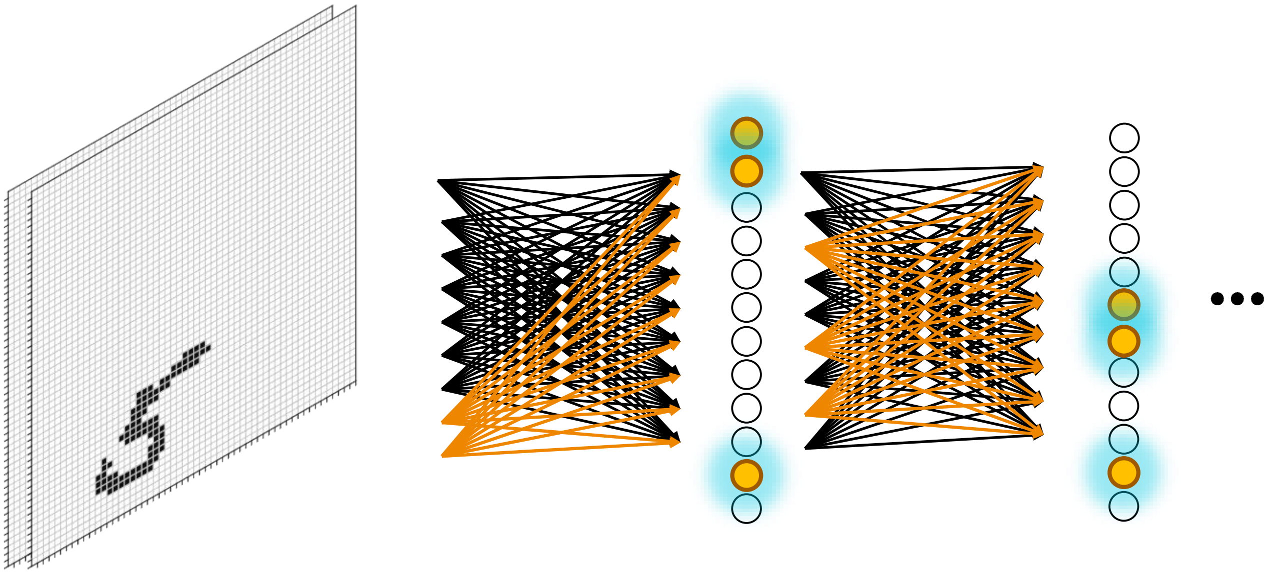
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).



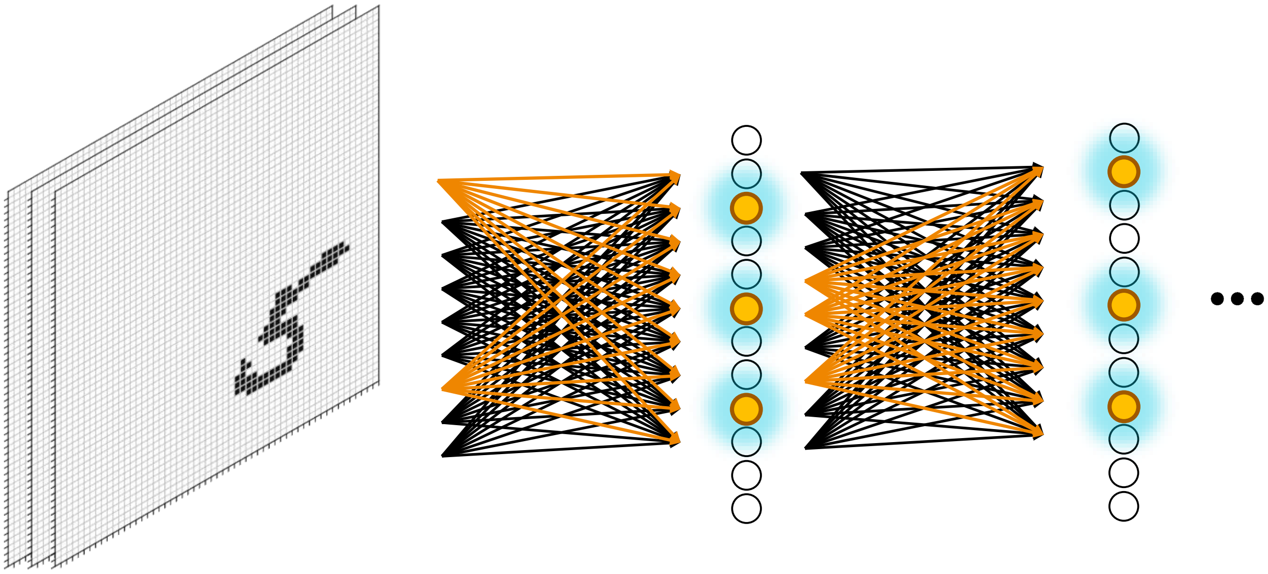
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).



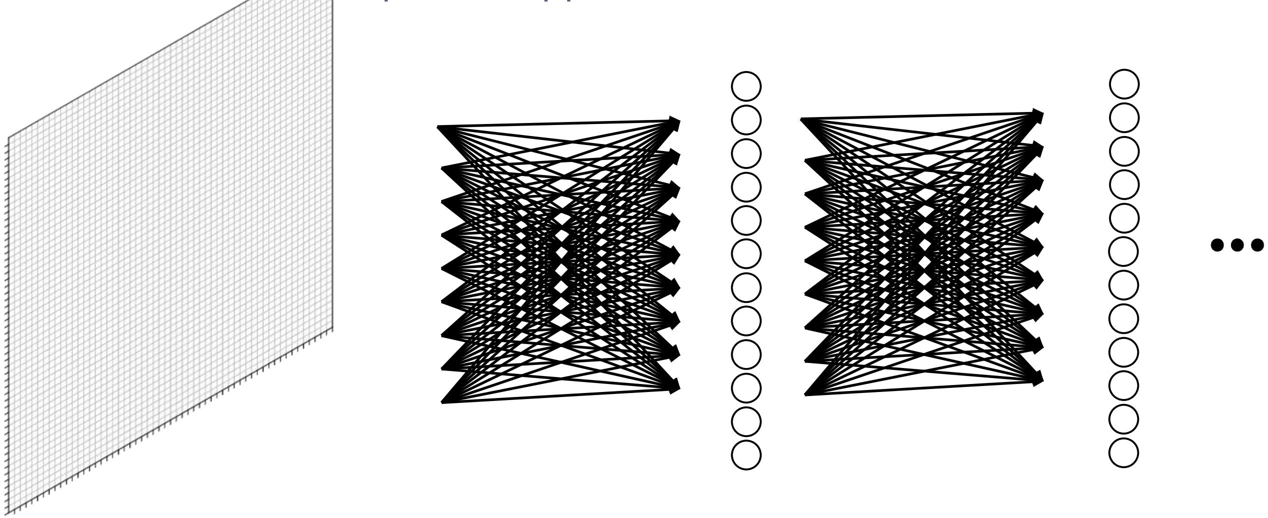
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).



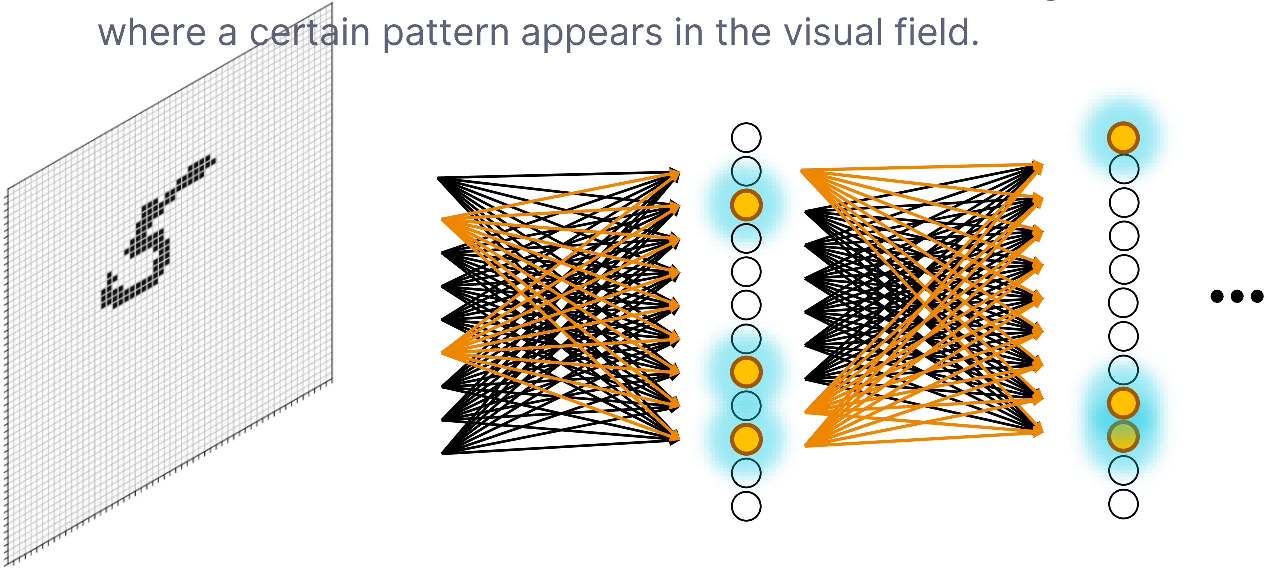
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).
- In the *human visual cortex*, the same neurons tend to get activate, regardless of where a certain pattern appears in the visual field.



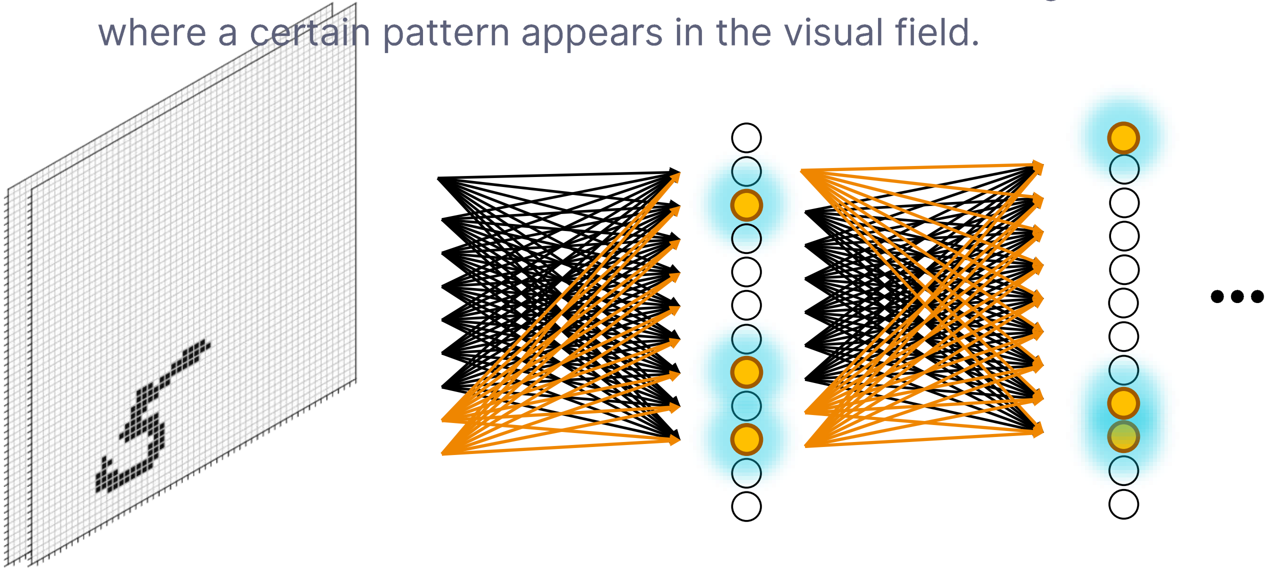
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).
- In the *human visual cortex*, the same neurons tend to get activate, regardless of where a certain pattern appears in the visual field.



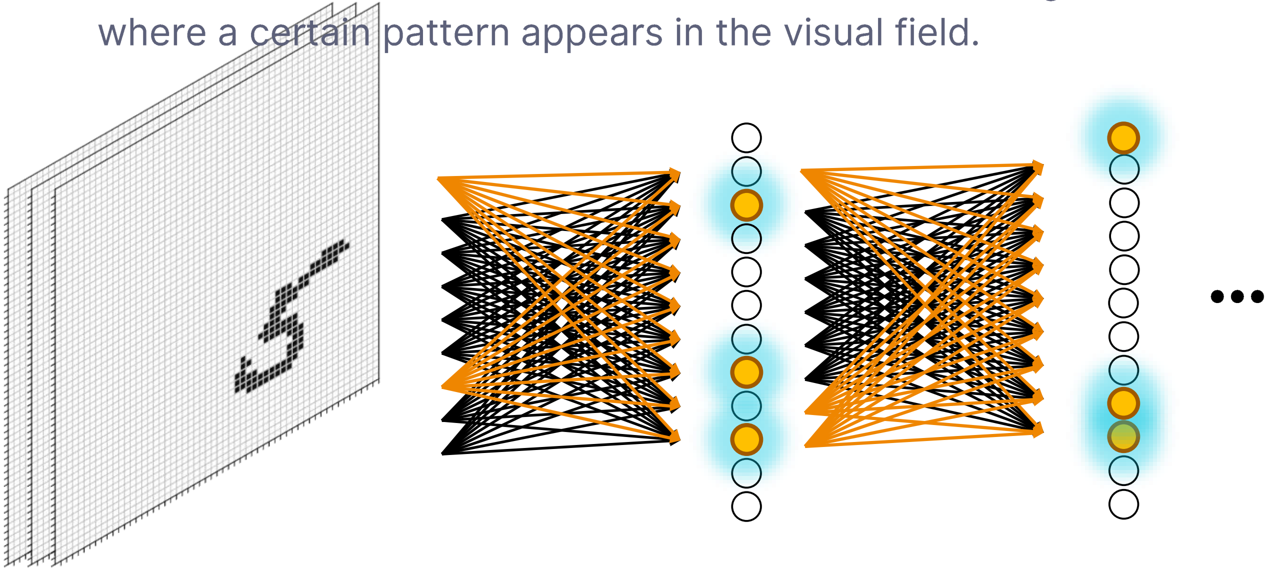
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).
- In the *human visual cortex*, the same neurons tend to get activate, regardless of where a certain pattern appears in the visual field.



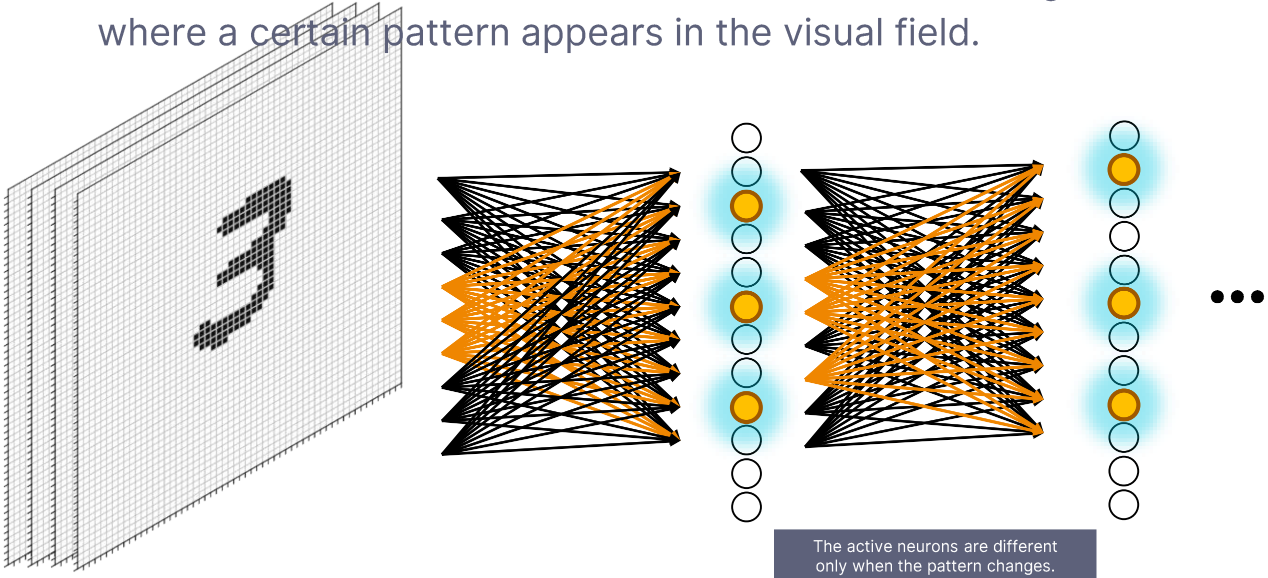
Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).
- In the *human visual cortex*, the same neurons tend to get activate, regardless of where a certain pattern appears in the visual field.



Motivation

- In a *Multilayer Perceptron*, completely different neurons get activated if the same pattern appears in different parts of the input image (even if the MLP correctly identifies the pattern).
- In the *human visual cortex*, the same neurons tend to get activate, regardless of where a certain pattern appears in the visual field.



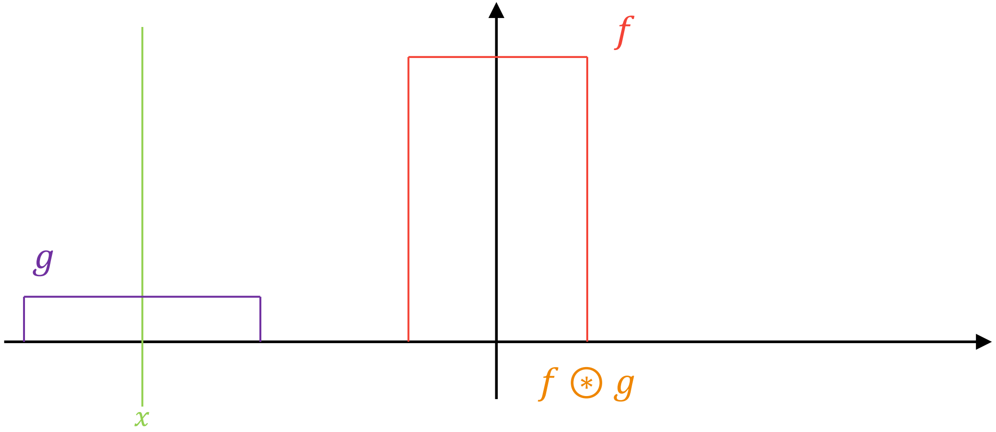
The active neurons are different only when the pattern changes.

What is a convolution?

What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

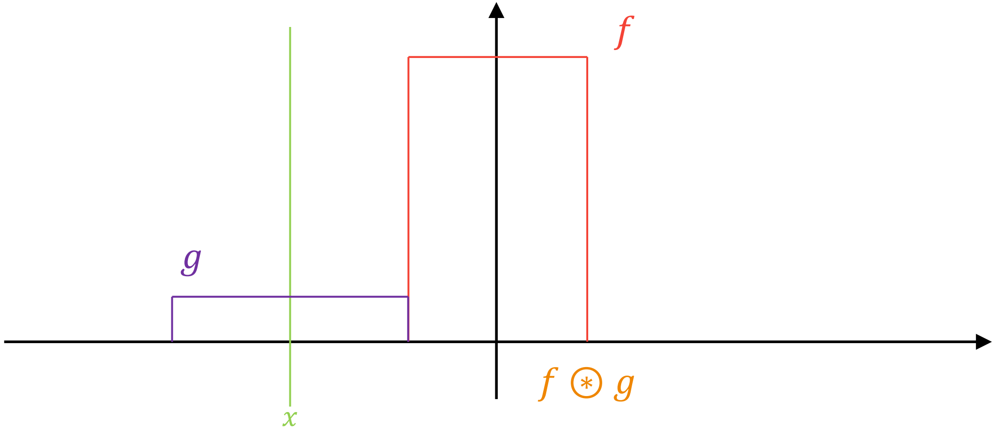
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



What is a convolution?

- Mathematically speaking, a convolution on two functions f and g is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

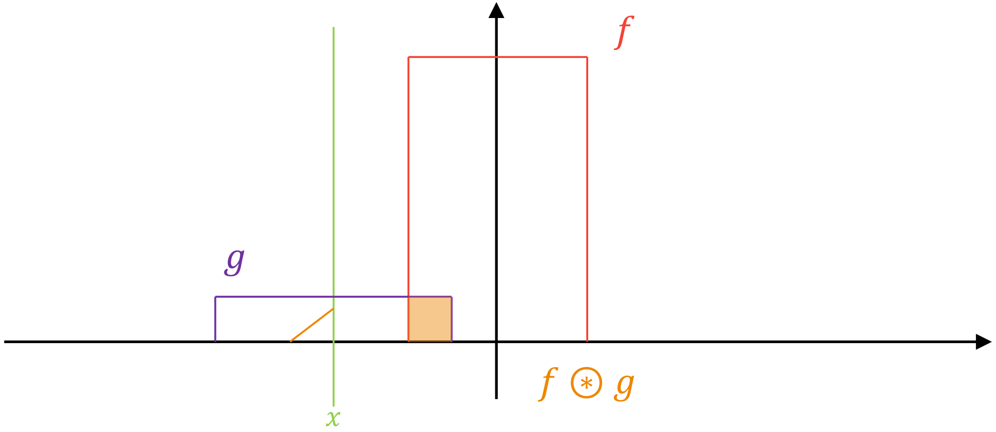
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

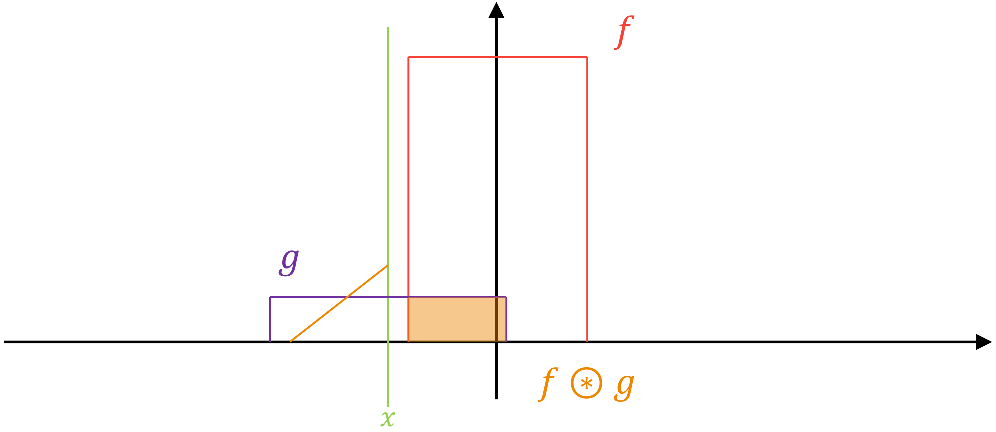
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

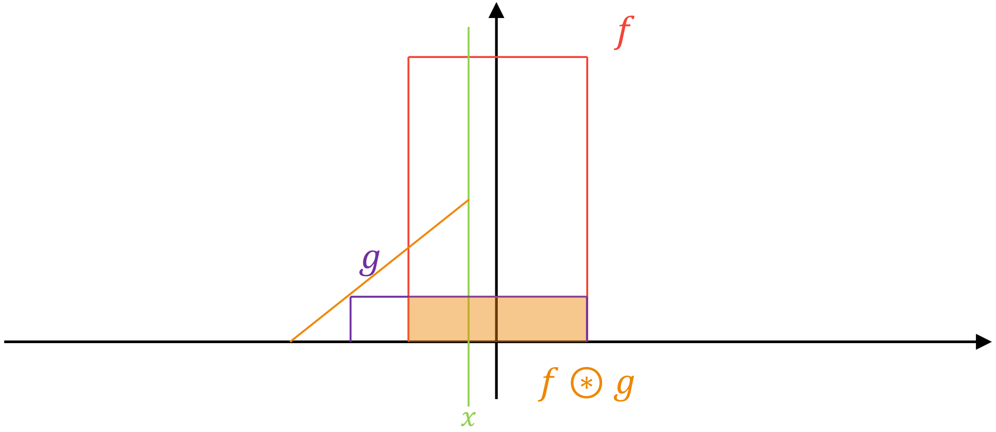
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

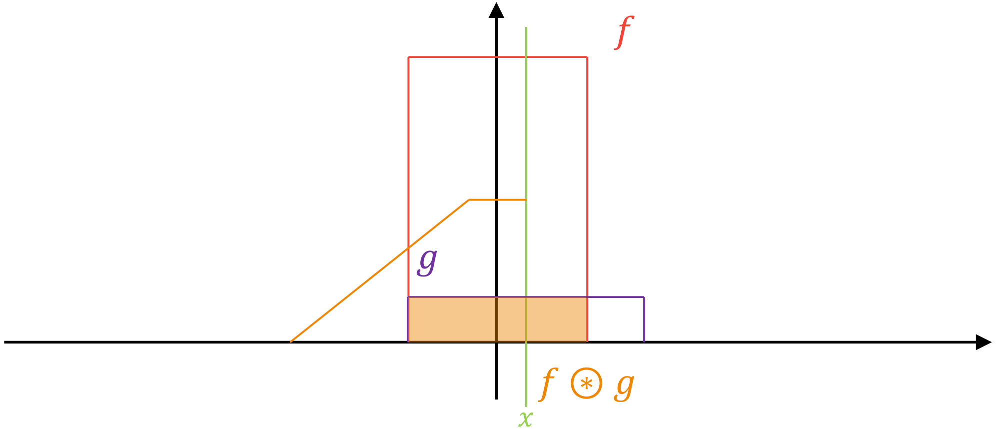
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

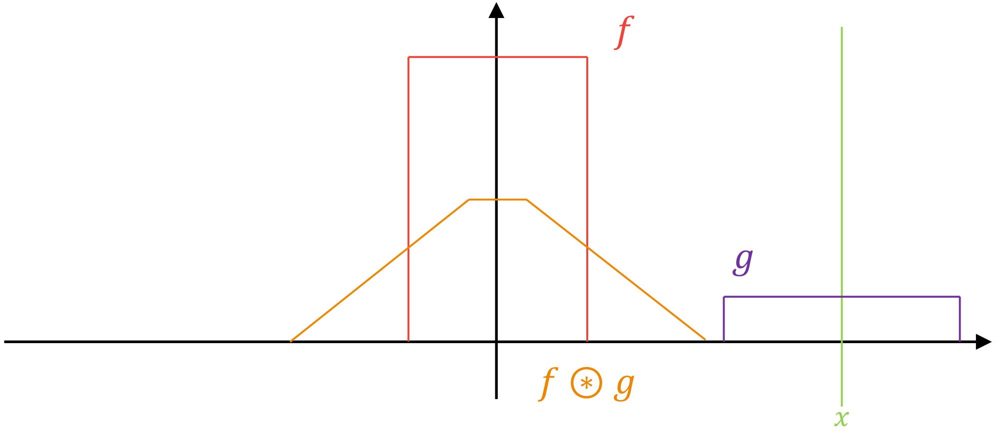
$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



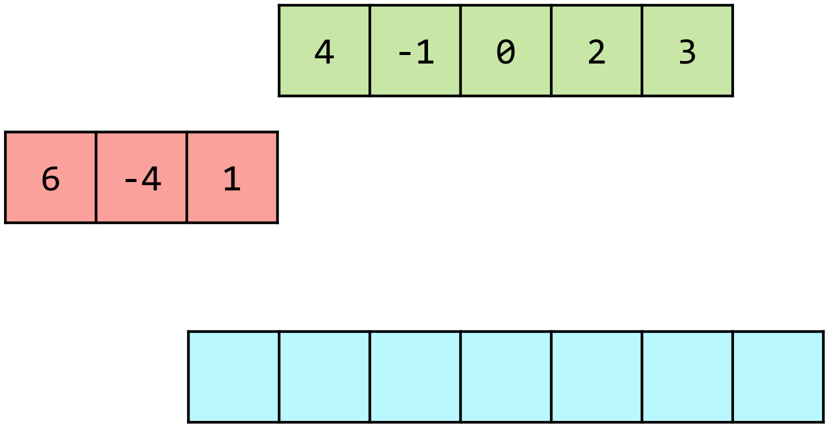
What is a convolution?

- Mathematically speaking, a convolution on two functions *f* and *g* is an operation $\circledast$ which produces a third function $f \circledast g$ that expresses how the shape of one is modified by the other.

$$(f \circledast g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$

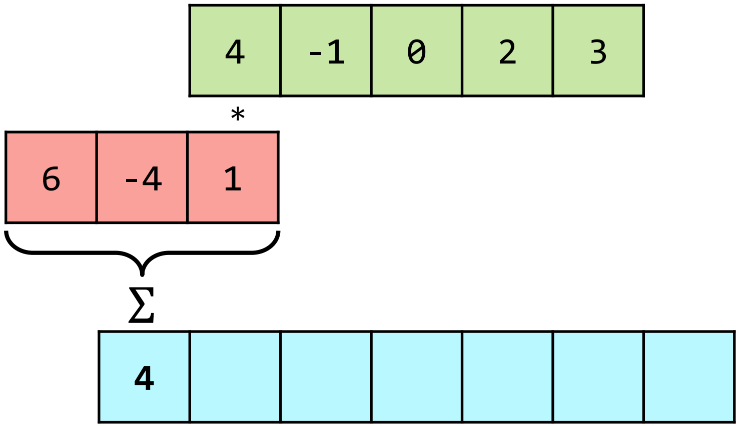


Discrete Convolution



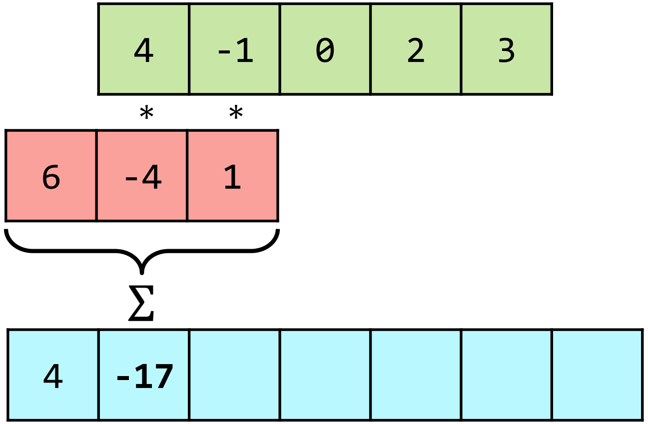
1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

Discrete Convolution



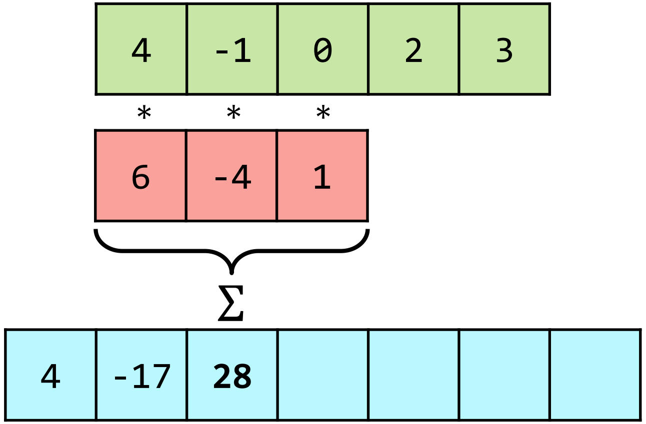
1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

Discrete Convolution



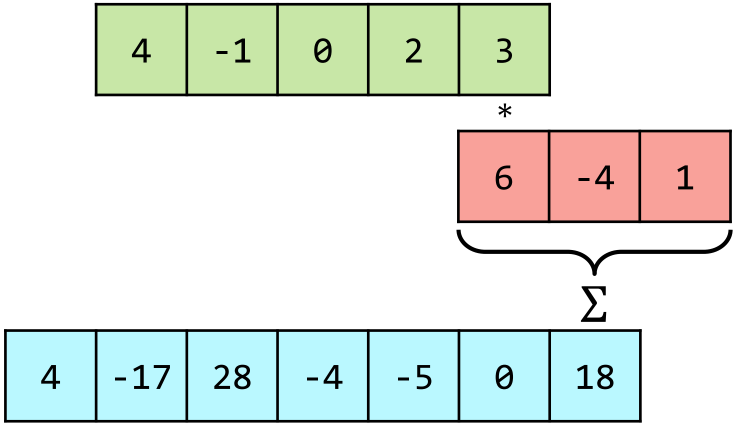
1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

Discrete Convolution



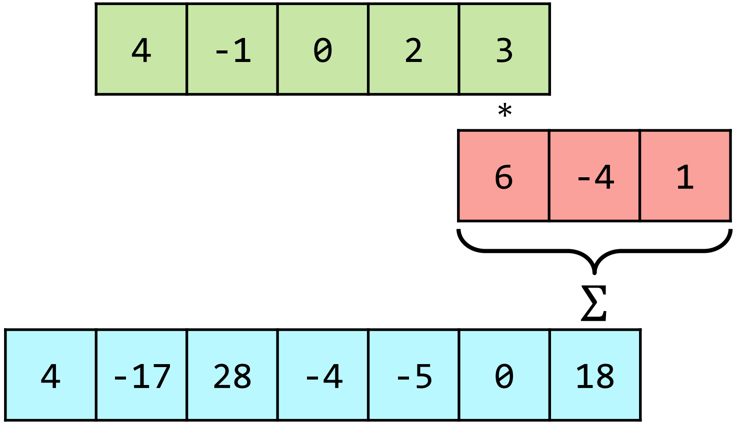
1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

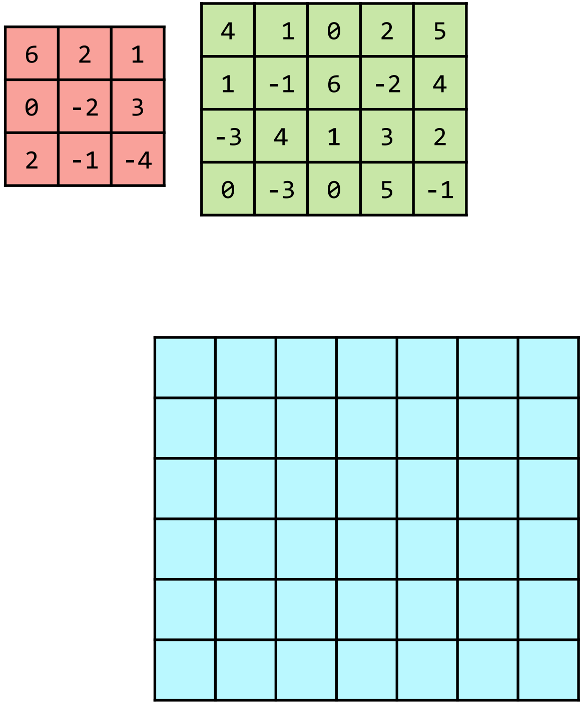
Discrete Convolution



1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

Discrete Convolution

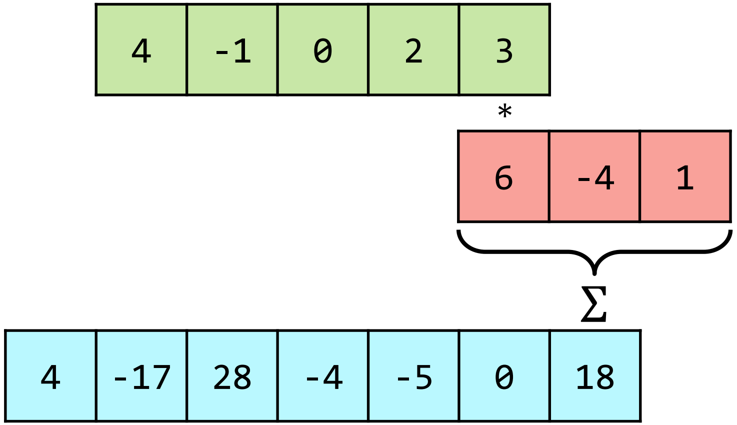


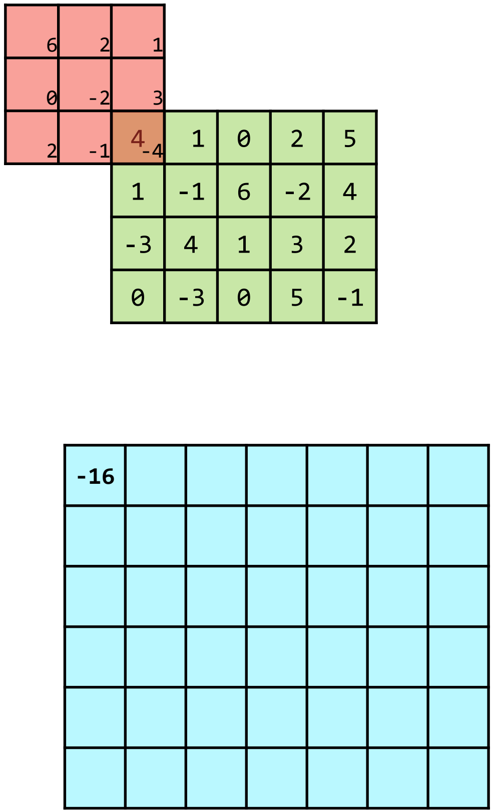


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

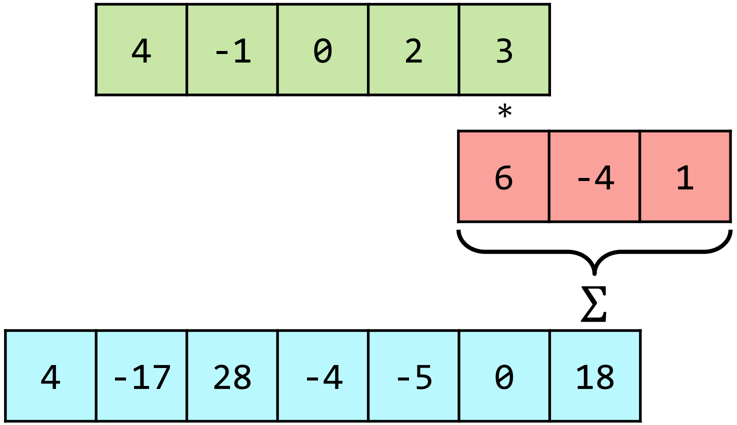


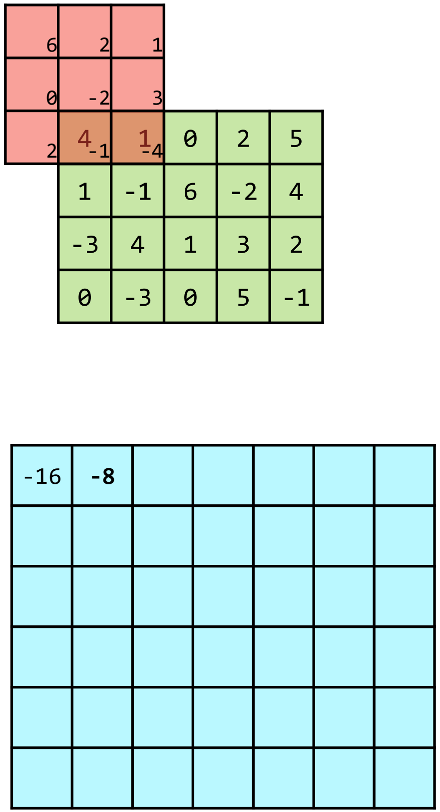


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

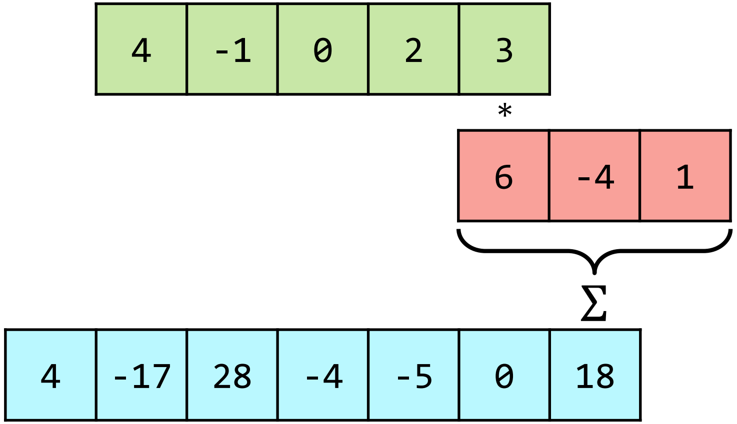


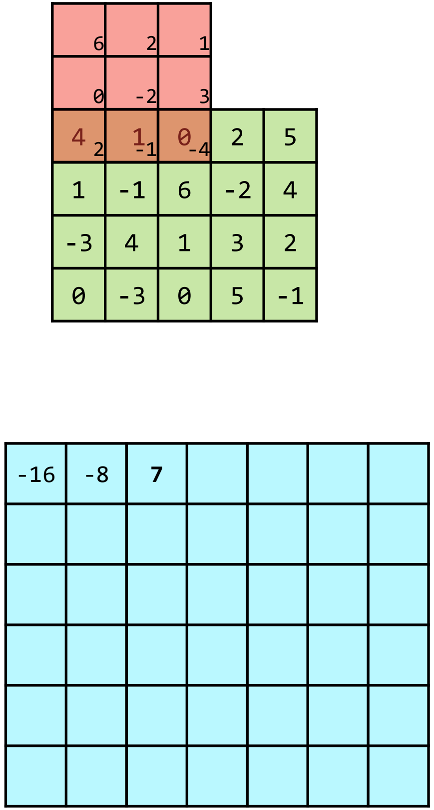


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

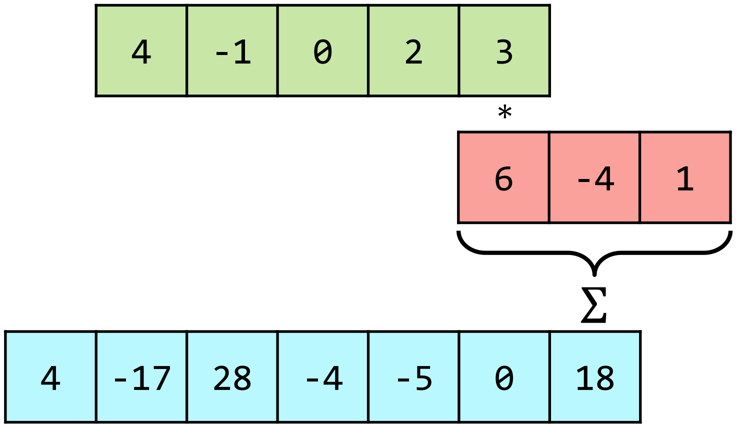


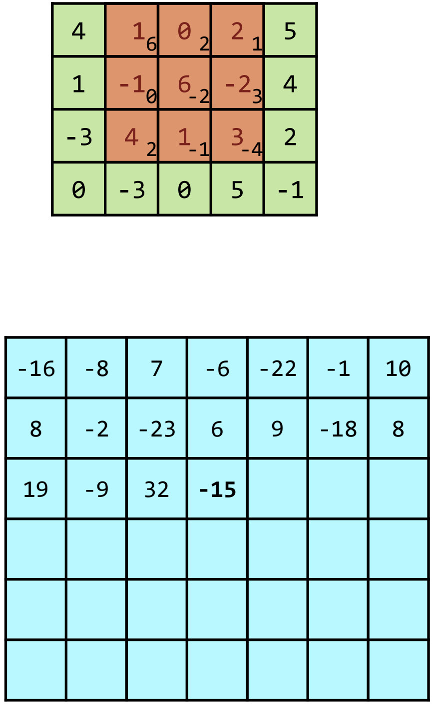


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

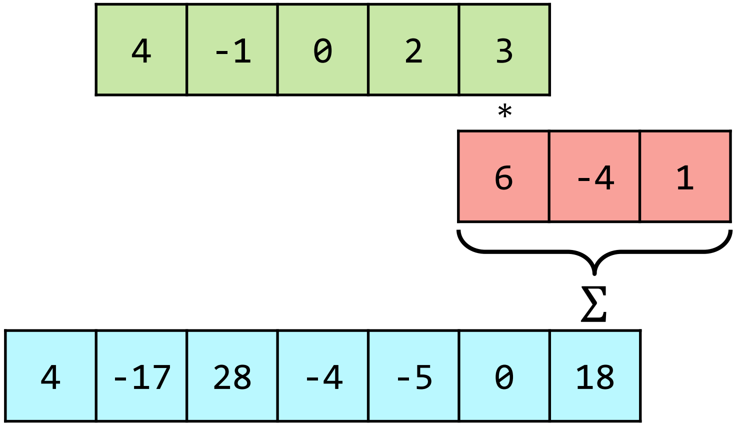


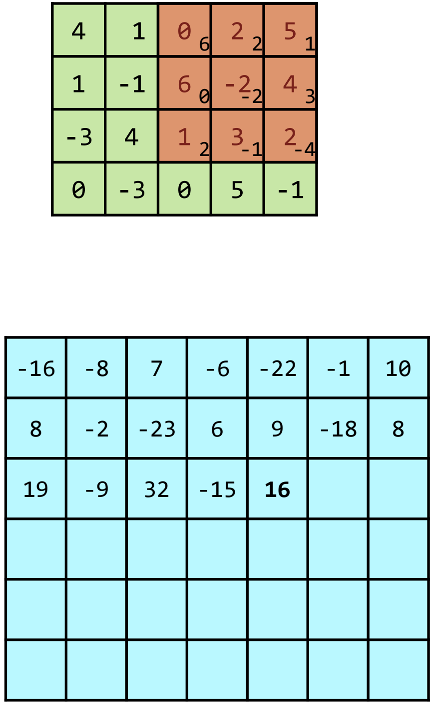


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

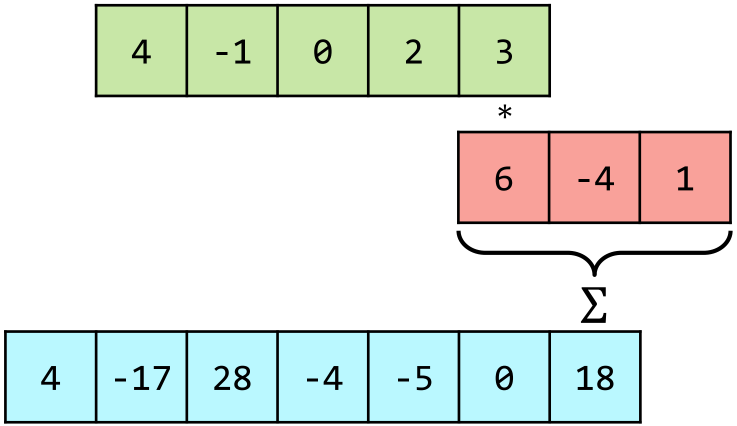


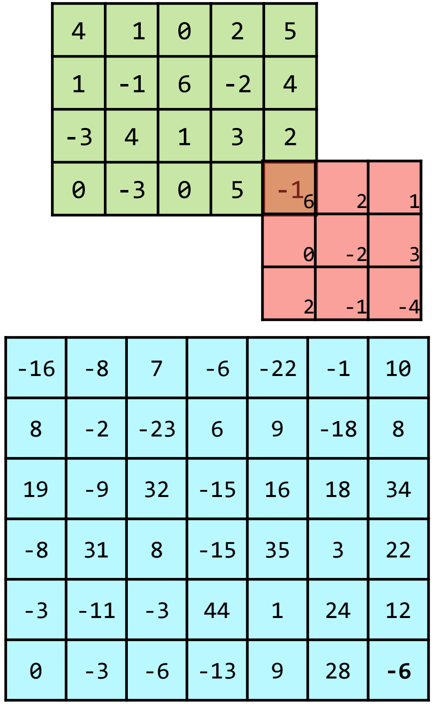


1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution



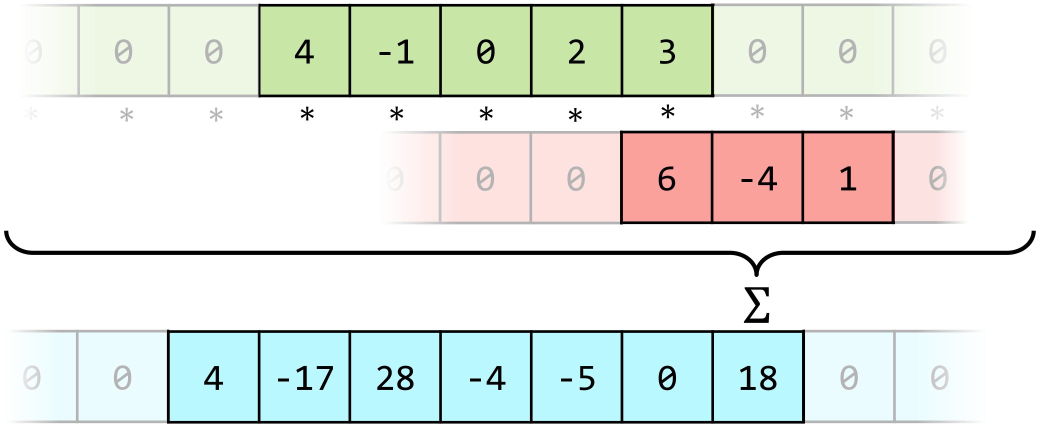


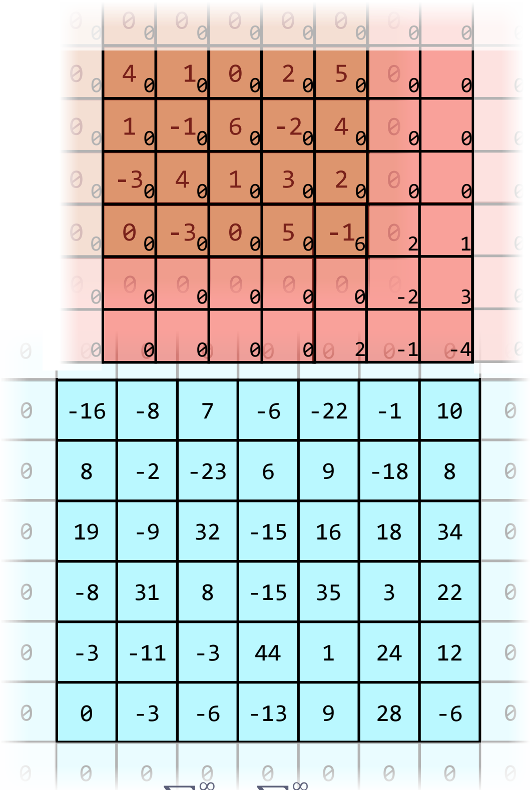
1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Discrete Convolution

The sums are *infinite*, but most of the elements are 0 and we are only interested in the *non-zero part of the result.*





1D: $(f \circledast g)(x) = \sum_{i=-\infty}^{\infty} f(i)g(x-i)$

2D: $(f \circledast g)(x_1, x_2) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(i,j)g(x_1 - i, x_2 - j)$

Convolutions in Image Processing

- Even though convolution is *commutative* ($f \circledast g = g \circledast f$), in practice we usually have a larger "input" matrix (typically an *image*) and a smaller matrix, called a **kernel,** which we "*convolve*" over the image.
- In the field of image processing, the **kernel** is also known as a **filter** or **mask** and it is used for *blurring*, *sharpening*, *edge detection* and other transformations of the input image.

$$\circledast \begin{bmatrix} 0 & -2 & 0 \\ -2 & 9 & -2 \\ 0 & -2 & 0 \end{bmatrix}$$

Sharpen

$$\circledast \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Blur

$$\circledast \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

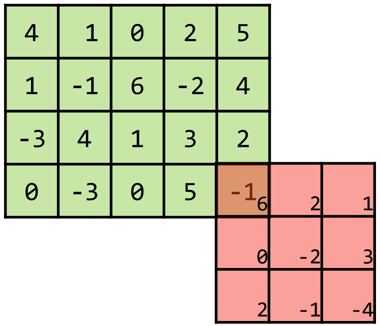
Convolution Parameters

Padding

Kernel matrix can be non-square, but it rarely happens in practice.

- Given an image $X \in \mathbb{R}^{n \times m}$ and a kernel $K \in \mathbb{R}^{k \times k}$, the mathematical definition of convolution will produce a result which is larger in size than X (it will be $(n+k-1) \times (m+k-1)$)

$$\underset{n \times m}{X} \circledast \underset{k \times k}{K} = \underset{(n+k-1)\times(m+k-1)}{R}$$



-16	-8	7	-6	-22	-1	10
8	-2	-23	6	9	-18	8
19	-9	32	-15	16	18	34
-8	31	8	-15	35	3	22
-3	-11	-3	44	1	24	12
0	-3	-6	-13	9	28	-6

Padding

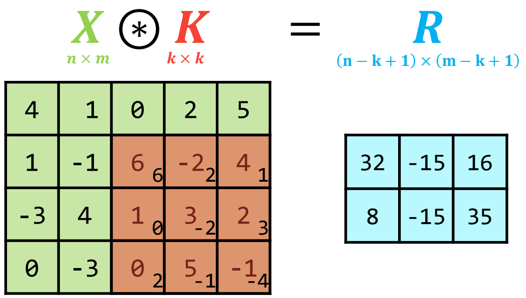
Kernel matrix can be non-square, but it rarely happens in practice.

- Given an image $X \in \mathbb{R}^{n \times m}$ and a kernel $K \in \mathbb{R}^{k \times k}$, the mathematical definition of convolution will produce a result which is larger in size than X (it will be $(n + k - 1) \times (m + k - 1)$)
- The *convolution* used in image processing is only applied to the area in which the two matrices **overlap completely**.
  - This will result in a matrix $R \in \mathbb{R}^{(n-k+1)\times(m-k+1)}$.

Padding

Kernel matrix can be non-square, but it rarely happens in practice.

- Given an image $X \in \mathbb{R}^{n \times m}$ and a kernel $K \in \mathbb{R}^{k \times k}$, the mathematical definition of convolution will produce a result which is larger in size than X (it will be $(n+k-1) \times (m+k-1)$)
- The *convolution* used in image processing is only applied to the area in which the two matrices **overlap completely**.
    - This will result in a matrix $R \in \mathbb{R}^{(n-k+1)\times(m-k+1)}$.



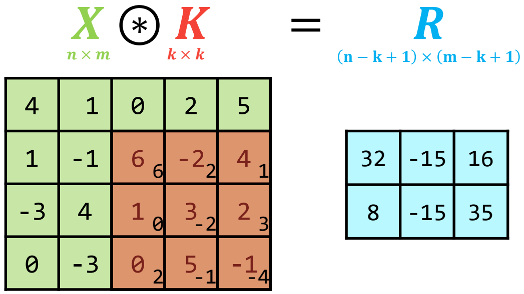
“Valid” Padding

- Basically, “no” padding.
- Sometimes used in practice.

Padding

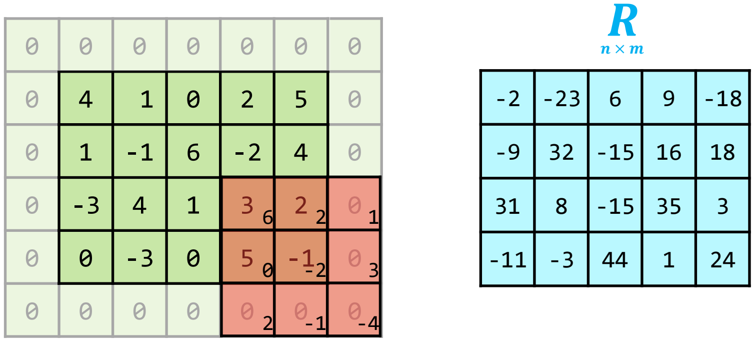
Kernel matrix can be non-square, but it rarely happens in practice.

- Given an image $X \in \mathbb{R}^{n \times m}$ and a kernel $K \in \mathbb{R}^{k \times k}$, the mathematical definition of convolution will produce a result which is larger in size than X (it will be $(n+k-1) \times (m+k-1)$)
- The *convolution* used in image processing is only applied to the area in which the two matrices **overlap completely**.
  - This will result in a matrix $R \in \mathbb{R}^{(n-k+1)\times(m-k+1)}$.
- In practice, we sometimes want the output size to the be same as the input size ($R \in \mathbb{R}^{n \times m}$).
  - This means we need to **pad** the original image with a ${}^{k-1}/_{2}$ border of zeros.



"Valid" Padding

- Basically, "no" padding.
- Sometimes used in practice.



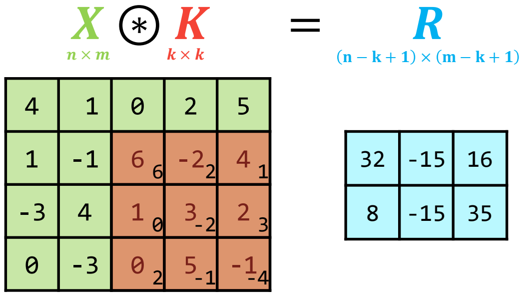
"Same" Padding

- Pad such that the output is the "same" size as the input
- Very common in practice.

Padding

Kernel matrix can be non-square, but it rarely happens in practice.

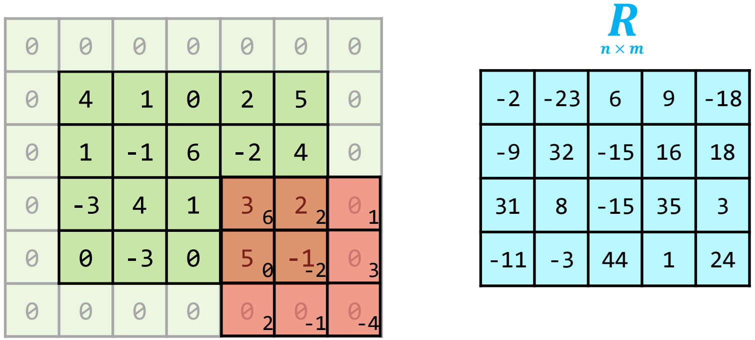
- Given an image $X \in \mathbb{R}^{n \times m}$ and a kernel $K \in \mathbb{R}^{k \times k}$, the mathematical definition of convolution will produce a result which is larger in size than X (it will be $(n+k-1) \times (m+k-1)$)
- The *convolution* used in image processing is only applied to the area in which the two matrices **overlap completely**.
  - This will result in a matrix $R \in \mathbb{R}^{(n-k+1)\times(m-k+1)}$.
- In practice, we sometimes want the output size to the be same as the input size ($R \in \mathbb{R}^{n \times m}$).
  - This means we need to **pad** the original image with a $^{k-1}/_2$ border of zeros.



"Valid" Padding

- Basically, "no" padding.
- Sometimes used in practice.

Kernel size *k* is usually odd. Otherwise we need unsymmetrical padding.

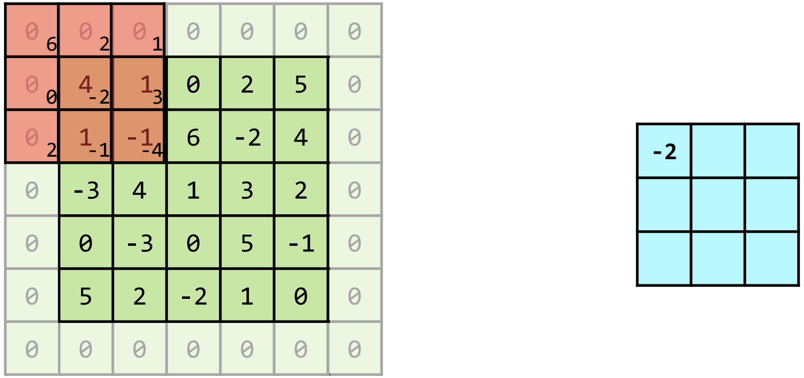


"Same" Padding

- Pad such that the output is the "same" size as the input
- Very common in practice.

Stride

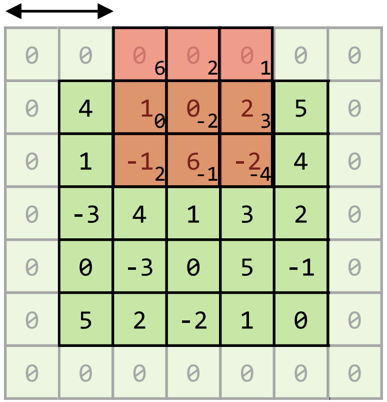
- **Stride** is the amount by which the filter moves when it is convolved over the input image.



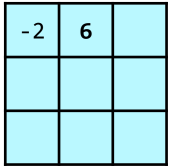
Stride $s = 2$

Stride

- **Stride** is the amount by which the filter moves when it is convolved over the input image.

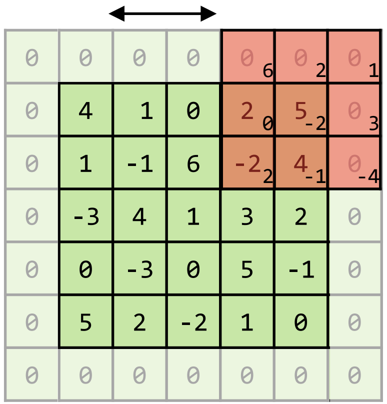


Stride $s = 2$

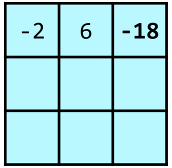


Stride

- **Stride** is the amount by which the filter moves when it is convolved over the input image.

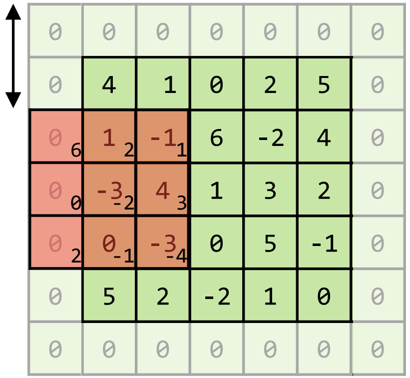


Stride $s = 2$

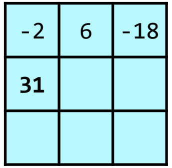


Stride

- **Stride** is the amount by which the filter moves when it is convolved over the input image.

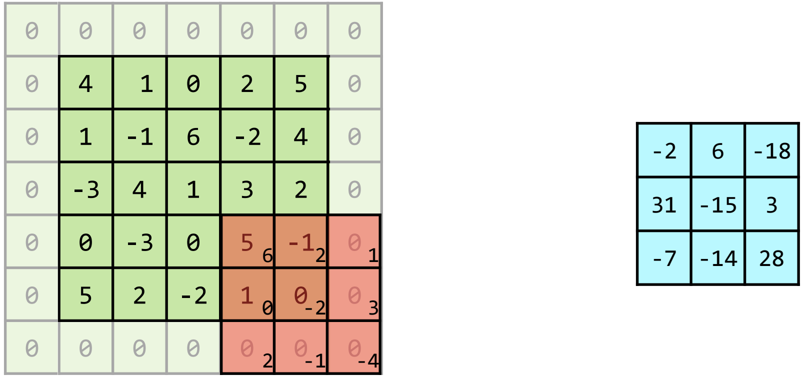


Stride $s = 2$



Stride

- **Stride** is the amount by which the filter moves when it is convolved over the input image.



Stride $s = 2$

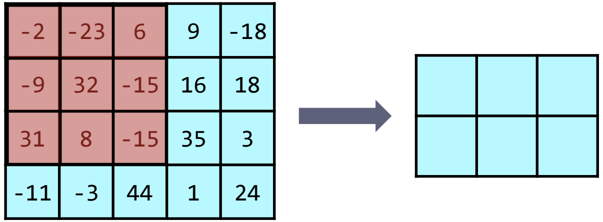
- If we have an image of size $n \times m$, a kernel of size $k \times k$ and we have padding p and stride s, the output will have size:

$$o = \left(\left\lfloor \frac{n-k+2p}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{m-k+2p}{s} \right\rfloor + 1\right)$$

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

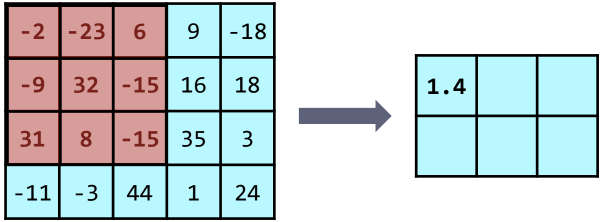


Average Pooling

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

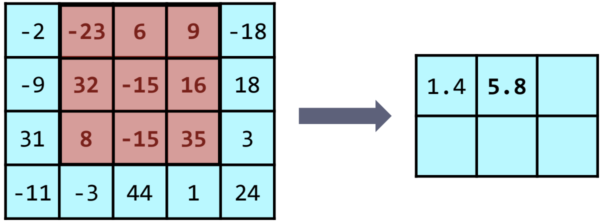


Average Pooling

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$$\text{size} = 3x3, \text{stride} = 1$$



Average Pooling

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

-2	-23	6	9	-18
-9	32	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

→

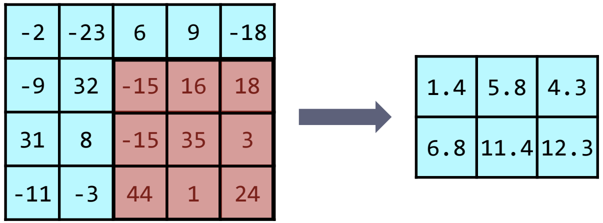
1.4	5.8	4.3
6.8	11.4	**12.3**

Average Pooling

Pooling

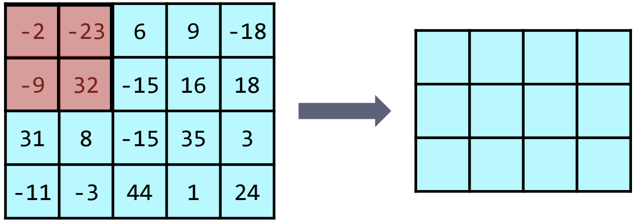
- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$



Average Pooling

$\text{size} = 2x2, \text{stride} = 1$



Max Pooling
Most common in practice

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
  - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
  - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

-2	-23	6	9	-18
-9	32	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

1.4	5.8	4.3
6.8	11.4	12.3

Average Pooling

$\text{size} = 2x2, \text{stride} = 1$

-2	-23	6	9	-18
-9	**32**	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

32			

Max Pooling
Most common in practice

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
    - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
    - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

-2	-23	6	9	-18
-9	32	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

→

1.4	5.8	4.3
6.8	11.4	12.3

Average Pooling

$\text{size} = 2x2, \text{stride} = 1$

-2	-23	6	9	-18
-9	**32**	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

→

32	**32**		

Max Pooling

Most common in practice

Pooling

- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
    - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
    - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$

-2	-23	6	9	-18
-9	32	-15	16	18
31	8	-15	35	3
-11	-3	44	1	24

→

1.4	5.8	4.3
6.8	11.4	12.3

Average Pooling

$\text{size} = 2x2, \text{stride} = 1$

-2	-23	6	9	-18
-9	32	-15	**16**	18
31	8	-15	35	3
-11	-3	44	1	24

→

32	32	**16**	

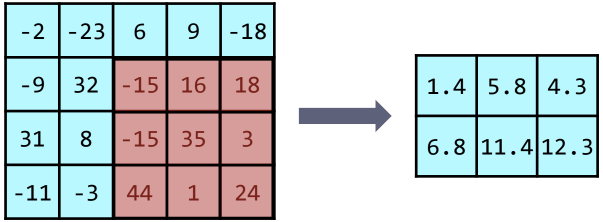
Max Pooling

Most common in practice

Pooling

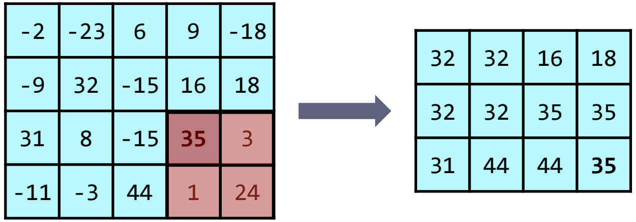
- **Pooling** is an operation which *down-samples* an input image and it is typically applied after a filter convolution.
    - The reasoning is that a filter usually tries *to capture a specific aspect* of the input image (e.g. edge detection) and it does not take as many pixels (features) to “describe” this particular aspect as for the original image.
    - Similarly to a convolution operation, pooling has *size, padding* and *stride*.

$\text{size} = 3x3, \text{stride} = 1$



Average Pooling

$\text{size} = 2x2, \text{stride} = 1$



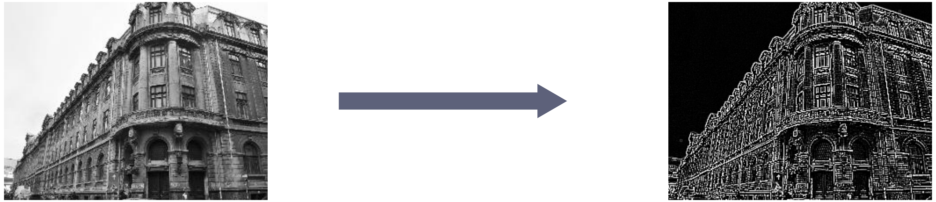
Max Pooling

Most common in practice

Neural Networks with Convolution

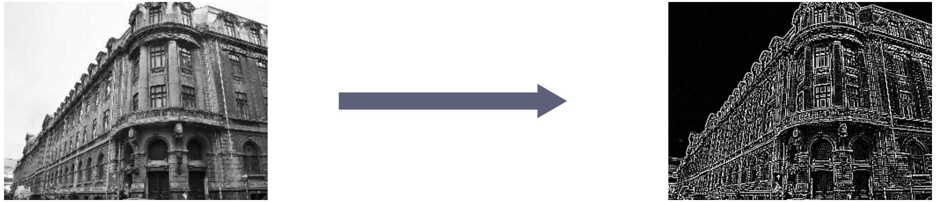
Convolutions as Features

- Convolving a kernel over an input image transforms it into a *new image* with *new characteristics* which were not present in the original image (a different “**representation**” of the input).



Convolutions as Features

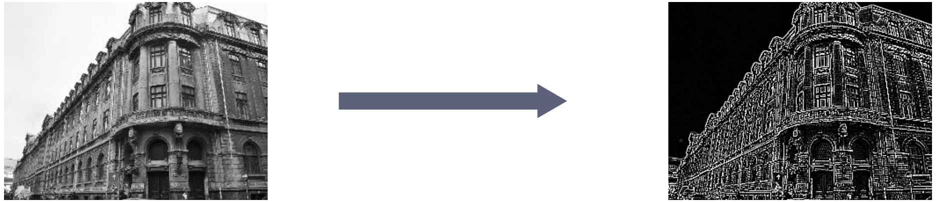
- Convolving a kernel over an input image transforms it into a *new image* with *new characteristics* which were not present in the original image (a different "**representation"** of the input).



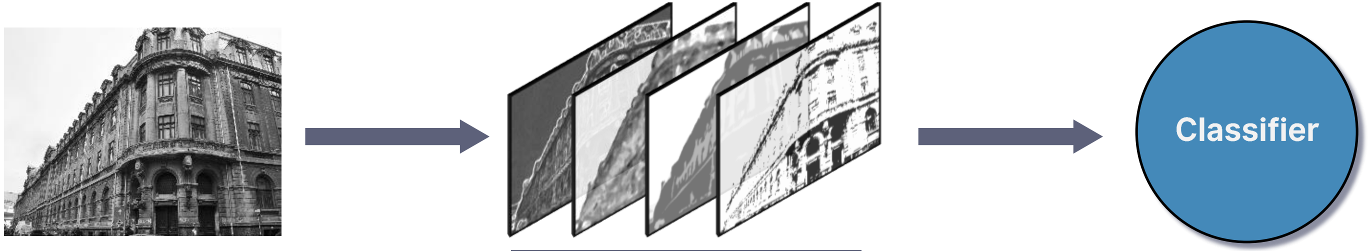
- These *newly obtained features* might contain *better discriminative information* about the labels we are trying to predict.
  - Passing them to a classifier instead of the original pixels could improve performance.

Convolutions as Features

- Convolving a kernel over an input image transforms it into a *new image* with *new characteristics* which were not present in the original image (a different “**representation**” of the input).



- These *newly obtained features* might contain *better discriminative information* about the labels we are trying to predict.
    - Passing them to a classifier instead of the original pixels could improve performance.
- Better still, we could take a couple of different kernels and feed all their outputs to the classifier.



“Representations” of the input.

Convolutional Layer

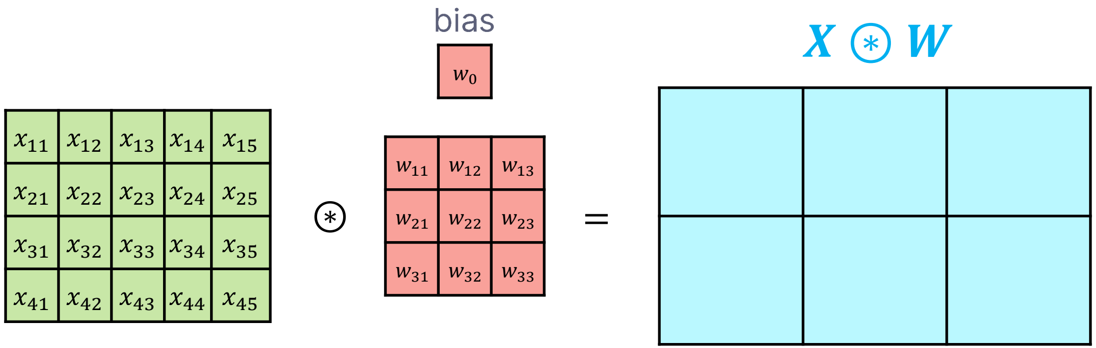
- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.

Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.

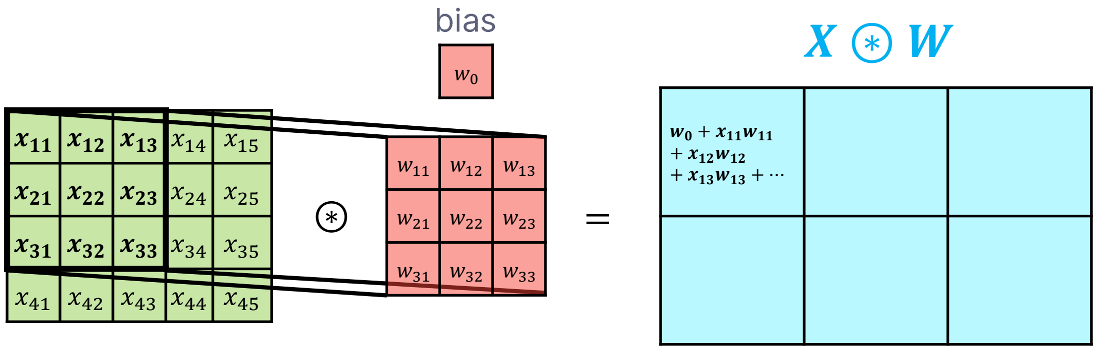
Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.
- A **convolutional layer** is a filter (or a set of filters) whose *weights are learned* through *gradient descent* in the *optimization process*.



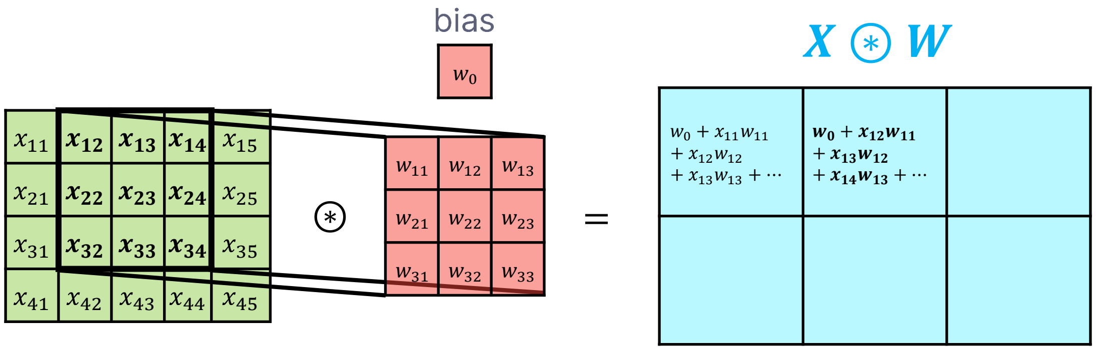
Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.
- A **convolutional layer** is a filter (or a set of filters) whose *weights are lear*ned through *gradient descent* in the *optimization process*.



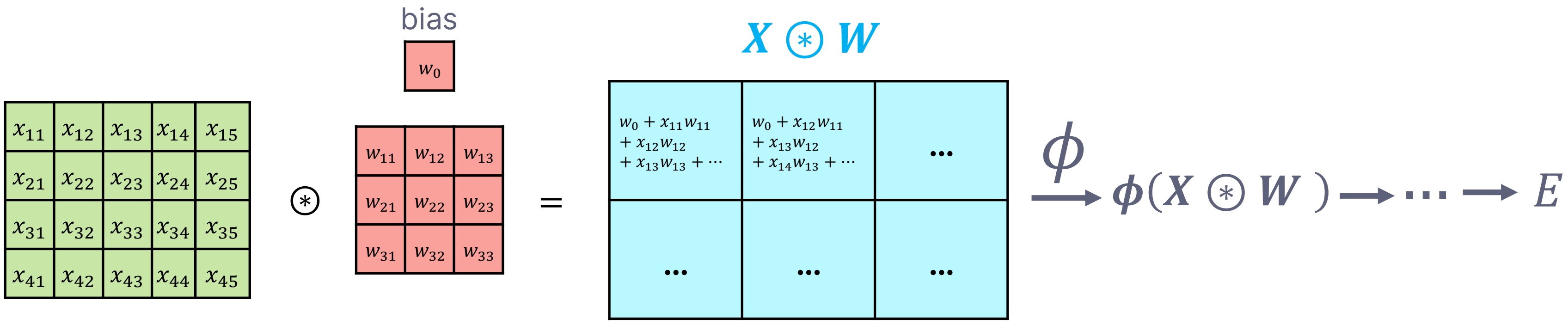
Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.
- A **convolutional layer** is a filter (or a set of filters) whose *weights are learned* through *gradient descent* in the *optimization process*.



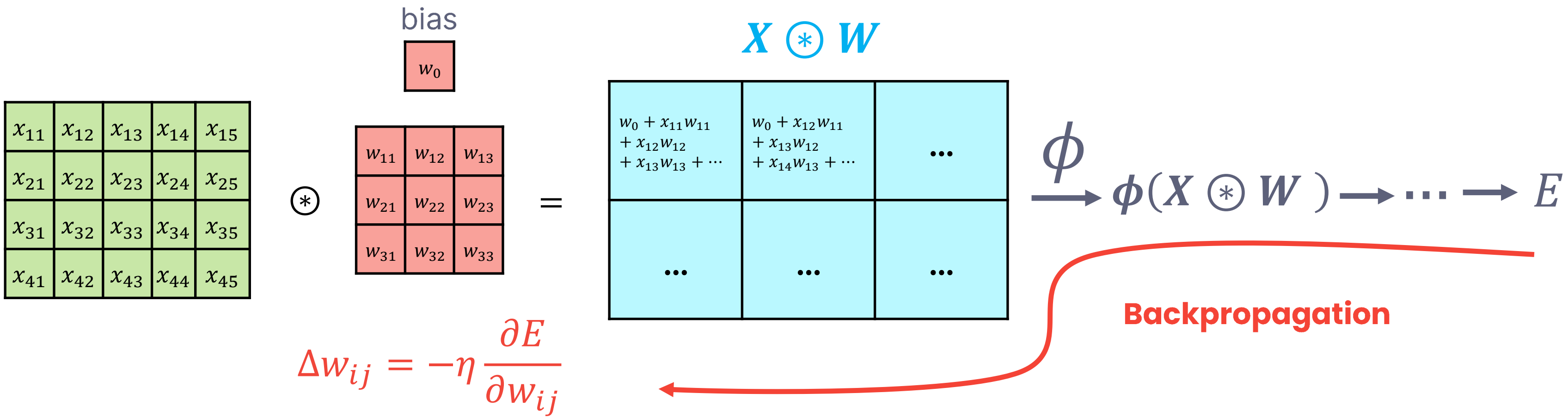
Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.
- A **convolutional layer** is a filter (or a set of filters) whose *weights are learned* through *gradient descent* in the *optimization process*.



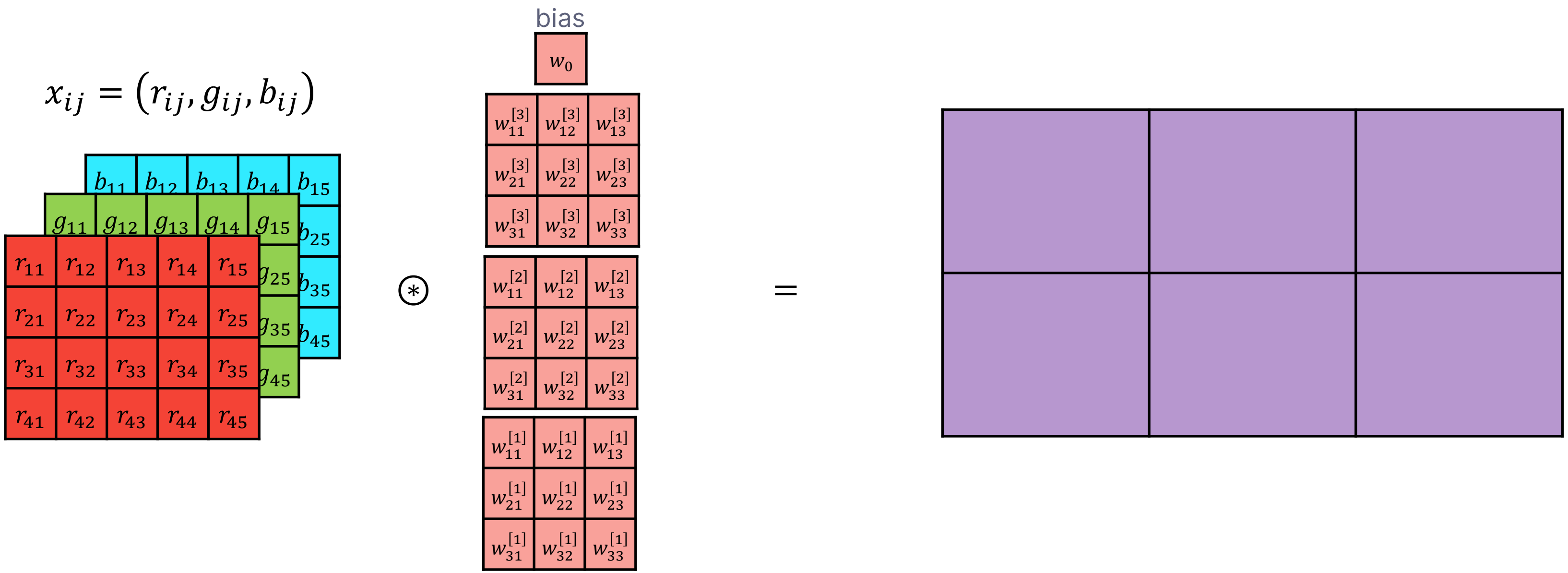
Convolutional Layer

- By choosing proper convolution kernels, we could substantially improve the performance of the predictive algorithm.
  - But in **Deep Learning**, *"choosing the proper convolutions"* should be the task of the learning algorithm itself.
- A **convolutional layer** is a filter (or a set of filters) whose *weights are learned* through *gradient descent* in the *optimization process*.



Multiple Channels

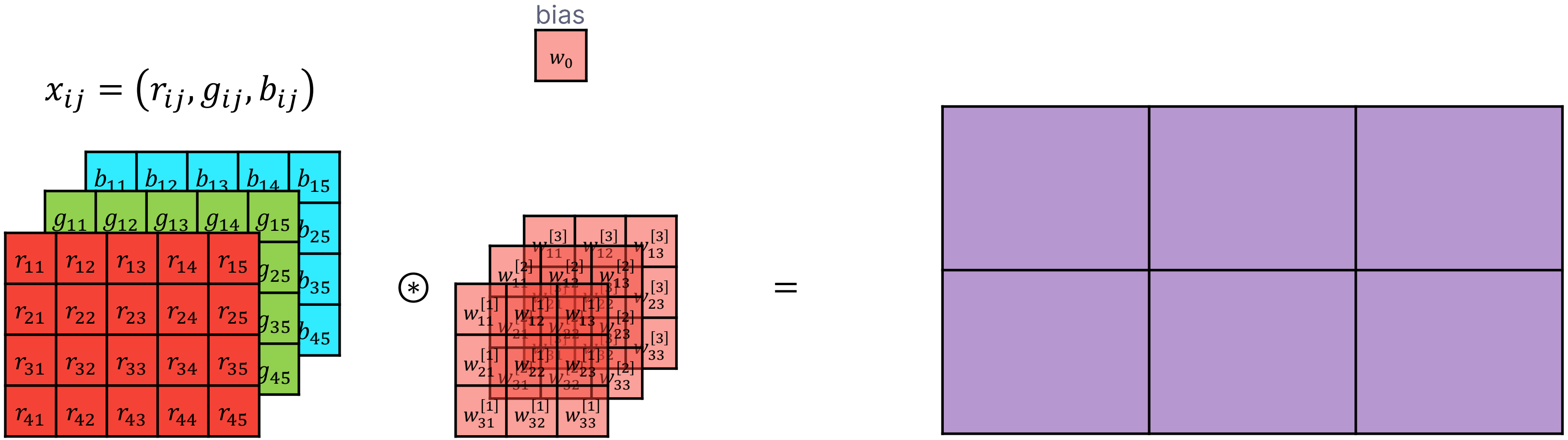
- A color image has 3 channels (each pixel has 3 different values for *red*, *green* and *blue*).
- A filter on a “multi-channel” image needs weights for each channel.



Multiple Channels

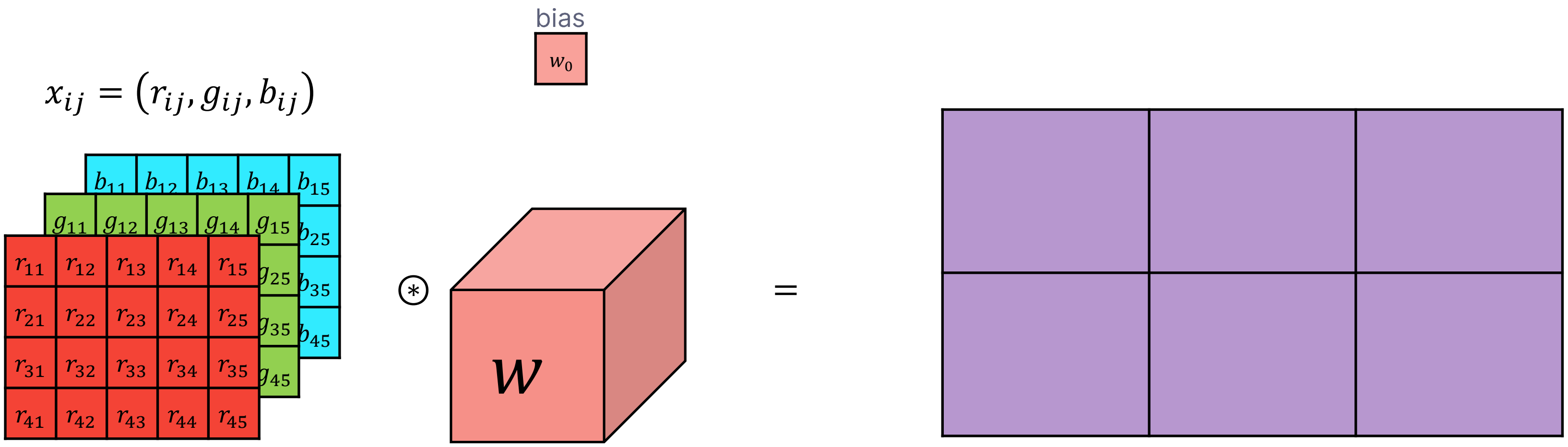
- A color image has 3 channels (each pixel has 3 different values for *red*, *green* and *blue*).
- A filter on a "multi-channel" image needs weights for each channel.

$$x_{ij} = (r_{ij}, g_{ij}, b_{ij})$$



Multiple Channels

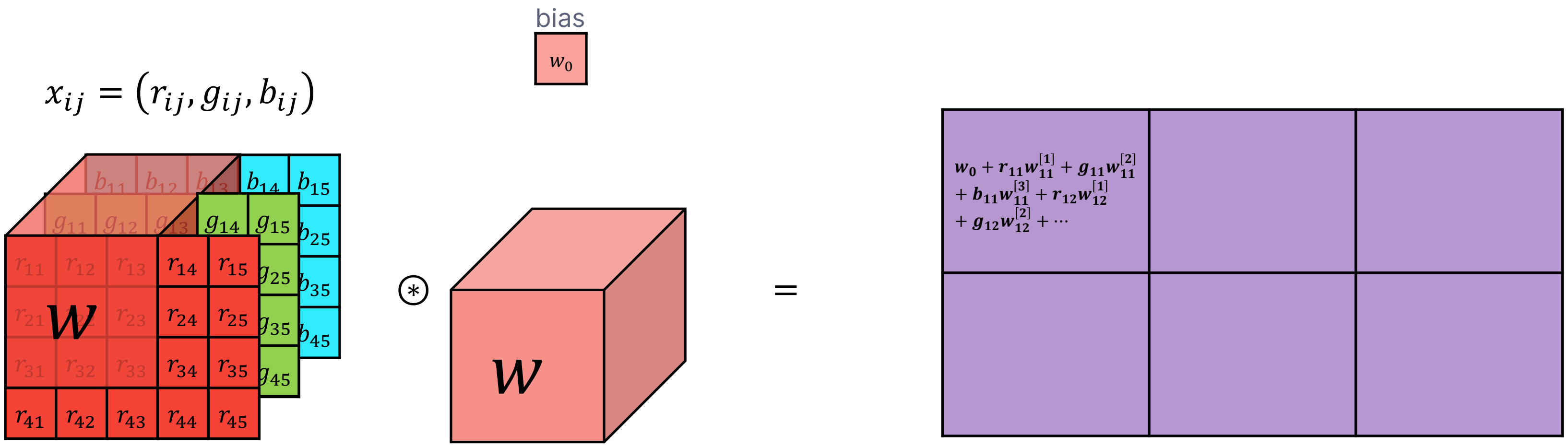
- A color image has 3 channels (each pixel has 3 different values for *red*, *green* and *blue*).
- A filter on a "multi-channel" image needs weights for each channel.



Multiple Channels

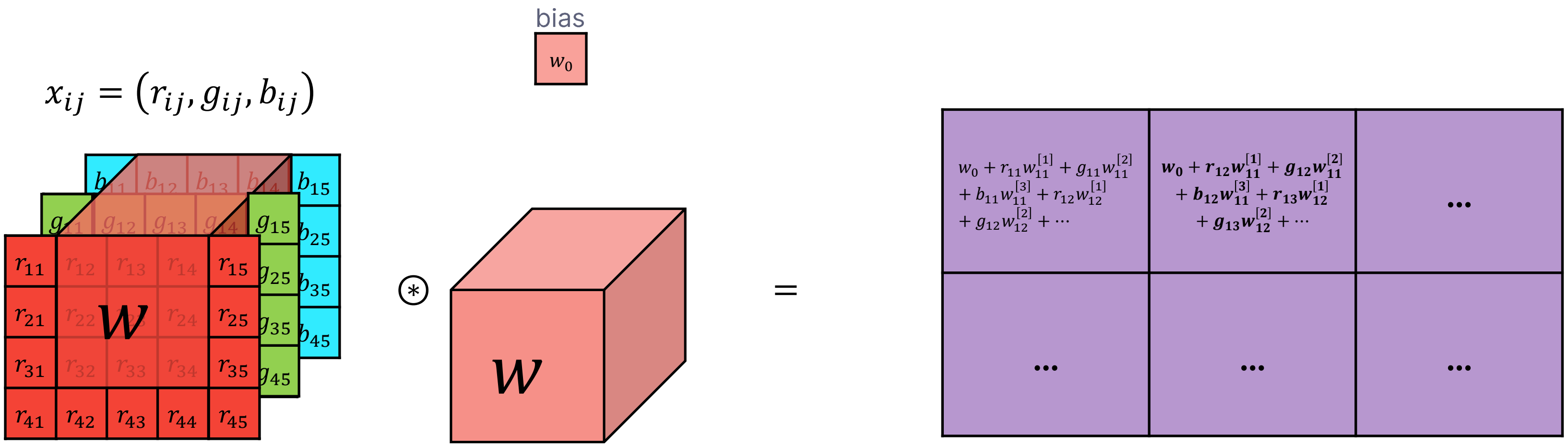
- A color image has 3 channels (each pixel has 3 different values for *red*, *green* and *blue*).
- A filter on a “multi-channel” image needs weights for each channel.

$$x_{ij} = (r_{ij}, g_{ij}, b_{ij})$$



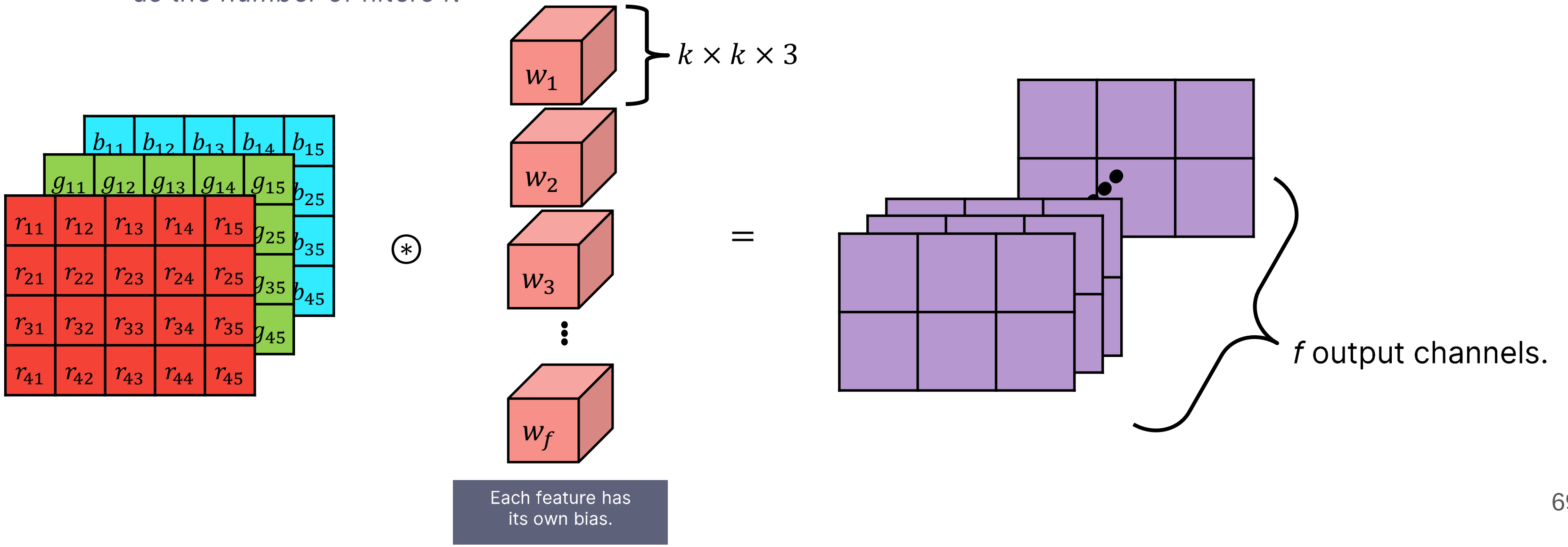
Multiple Channels

- A color image has 3 channels (each pixel has 3 different values for *red*, *green* and *blue*).
- A filter on a “multi-channel” image needs weights for each channel.

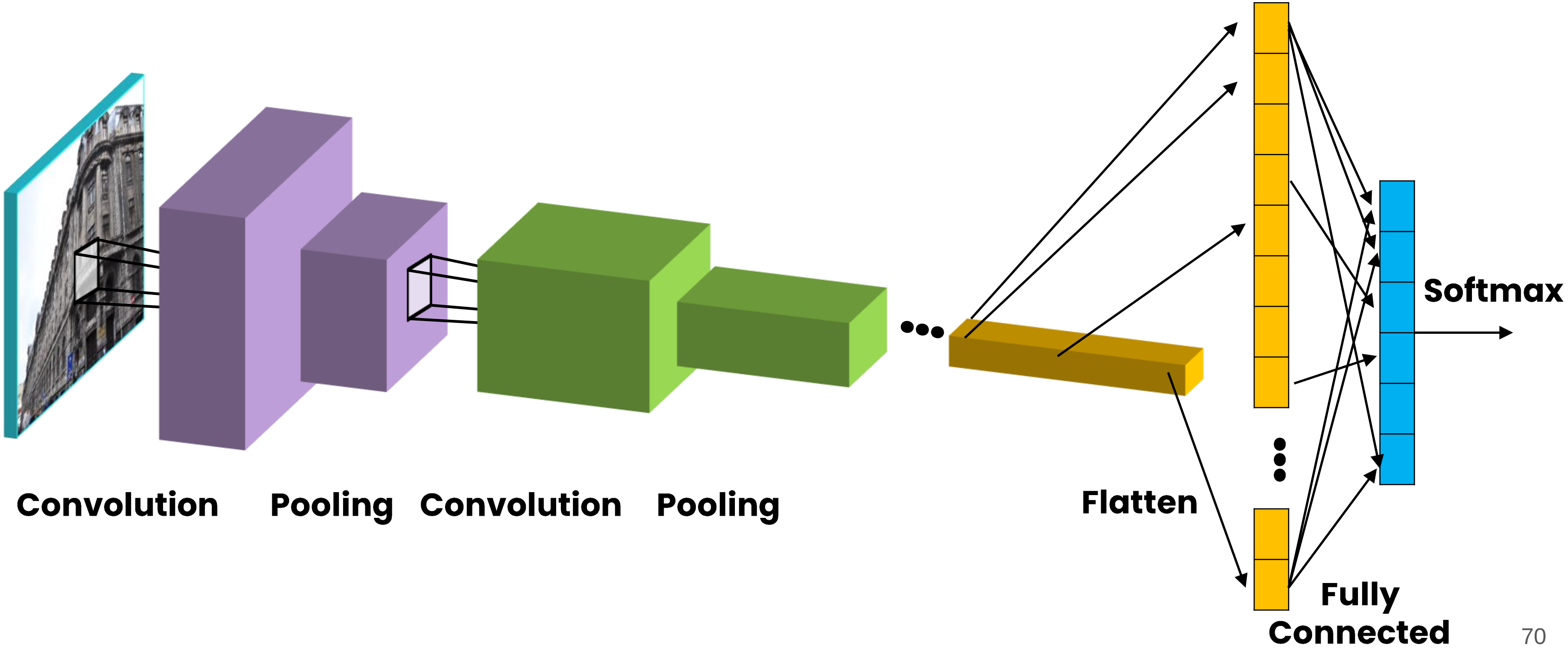


Multiple Channels

- One filter usually captures one feature of the input image, but we want to obtain a *representation with many features*.
  - A **convolutional layer** is usually made up of **multiple filters**.
  - Each filter has the *depth* (number of channels) of the input image and the output has *as many channels as the number of filters f*.



Typical CNN Architecture



Keywords

Convolution

Kernel | Filter | Mask

Padding | Valid Padding | Same Padding

Stride | Pooling | MaxPooling

Convolutional Neural Network (CNN)

Recurrent Neural Networks

Handling **sequences** with neural networks

Faculty of Mathematics and Computer Science, University of Bucharest
and
Sparktech Software

Academic Year 2018/2019, 1st Semester

Motivation

- The problems we have seen so far are “one-to-one” (i.e. one input mapped to one output).
  - Better said, all inputs and all outputs always have the same fixed size.
    - e.g. In the digit recognition problem, all inputs had 28x28 = 724 pixels and all outputs had 10 values (one hot encoding of a digit).

Motivation

- The problems we have seen so far are "one-to-one" (i.e. one input mapped to one output).
    - Better said, all inputs and all outputs always have the same fixed size.
        - e.g. In the digit recognition problem, all inputs had 28x28 = 724 pixels and all outputs had 10 values (one hot encoding of a digit).
- Many real problems involve sequences of data:

Speech Recognition

"What is the weather going to be like tomorrow?"

Image Captioning

"Black and white dog jumps over bar."

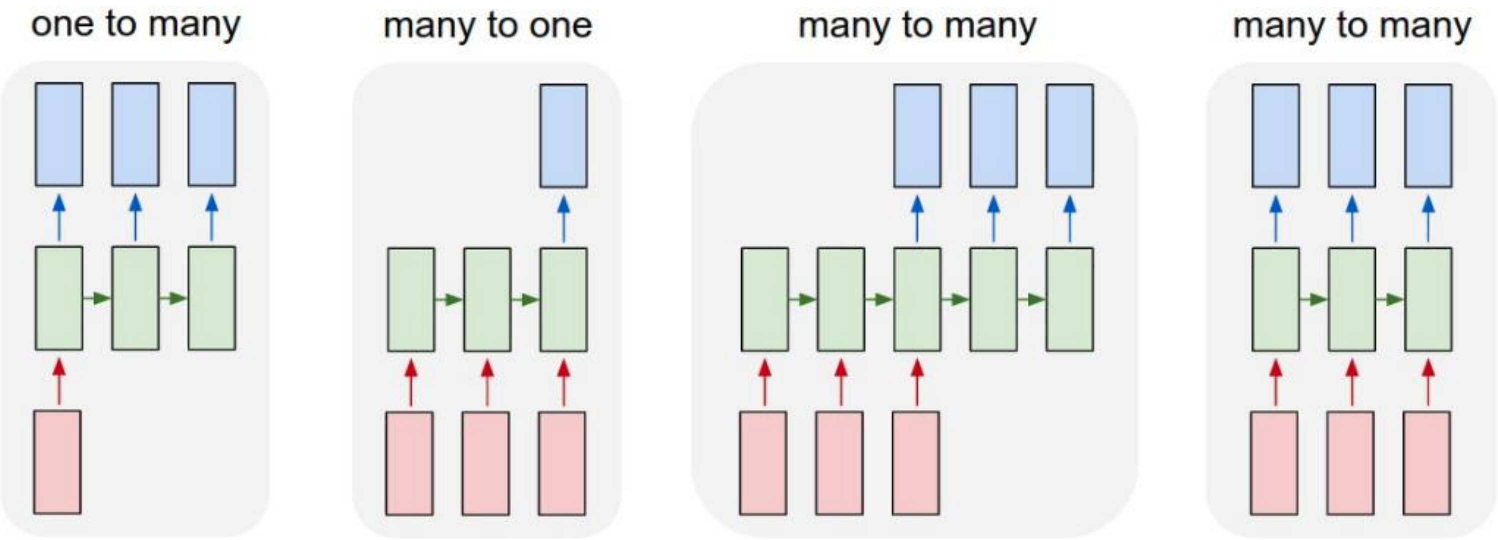
Machine Translation

"I am going to the cinema."

"Ich gehe ins Kino."

Motivation

- Types of problems which involve sequences:



Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Layers

Notation

- One training example is now a **sequence of values:** $\vec{x} \in \mathbb{R}^{m \times n \times T}$
  - $\vec{x}_j^{(i)\langle t \rangle}$ - component *j* of training sample *i* at timestep *t*

Notation

- One training example is now a **sequence of values:** $\vec{x} \in \mathbb{R}^{m \times n \times T}$
  - $\vec{x}_j^{(i)\langle t\rangle}$ - component *j* of training sample *i* at timestep *t*

$\vec{x}^{\langle t\rangle} \in \mathbb{R}^n$

input at "timestep" *t*

$\hat{y}^{\langle t\rangle} \in \mathbb{R}^o$

output at timestep *t*

Notation

- One training example is now a **sequence of values:** $\vec{x} \in \mathbb{R}^{m \times n \times T}$
  - $\vec{x}_j^{(i)\langle t \rangle}$ - component *j* of training sample *i* at timestep *t*

$\vec{x}^{\langle t \rangle} \in \mathbb{R}^n$

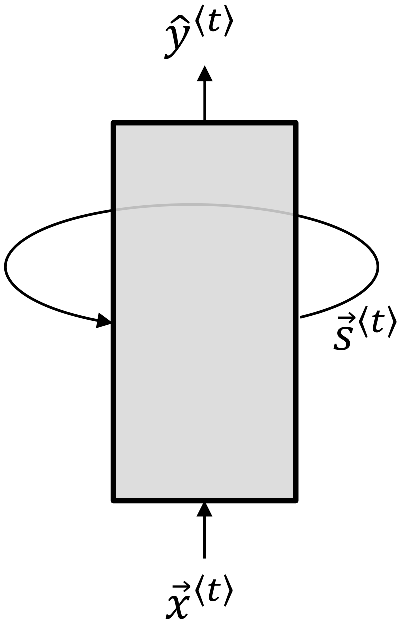
input at "timestep" *t*

$\hat{y}^{\langle t \rangle} \in \mathbb{R}^o$

output at timestep *t*

$\vec{s}^{\langle t \rangle} \in \mathbb{R}^h$

hidden state at "timestep" *t*



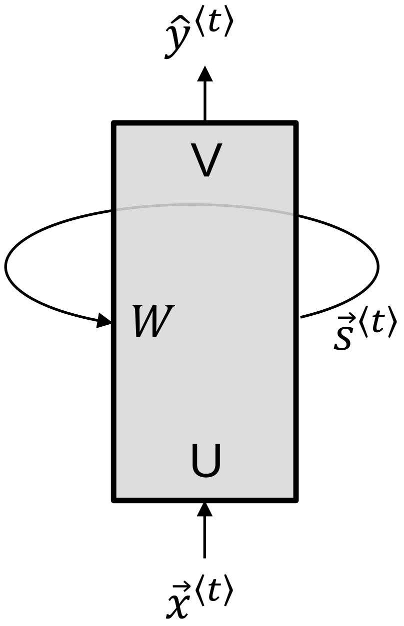
Notation

- One training example is now a **sequence of values:** $\vec{x} \in \mathbb{R}^{m \times n \times T}$
  - $\vec{x}_j^{(i)\langle t \rangle}$ - component *j* of training sample *i* at timestep *t*

$\vec{x}^{\langle t \rangle} \in \mathbb{R}^n$
input at ”timestep” *t*

$\hat{y}^{\langle t \rangle} \in \mathbb{R}^o$
output at timestep *t*

$\vec{s}^{\langle t \rangle} \in \mathbb{R}^h$
hidden state at ”timestep” *t*



Three weight matrices:
- $U \in \mathbb{R}^{h \times n}$
- $W \in \mathbb{R}^{h \times h}$
- $V \in \mathbb{R}^{o \times h}$

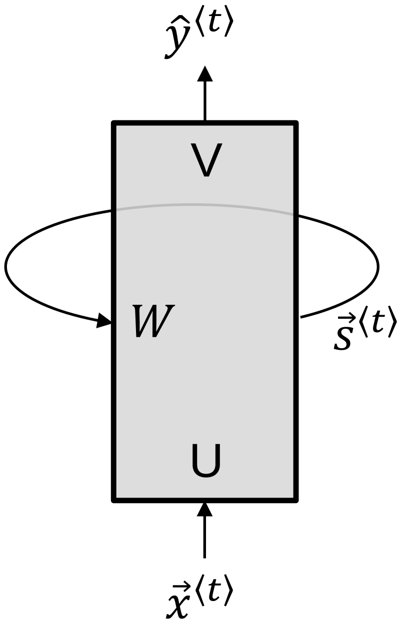
Notation

- One training example is now a **sequence of values:** $\vec{x} \in \mathbb{R}^{m \times n \times T}$
  - $\vec{x}_j^{(i)\langle t \rangle}$ - component *j* of training sample *i* at timestep *t*

$\vec{x}^{\langle t \rangle} \in \mathbb{R}^n$
input at "timestep" *t*

$\hat{y}^{\langle t \rangle} \in \mathbb{R}^o$
output at timestep *t*

$\vec{s}^{\langle t \rangle} \in \mathbb{R}^h$
hidden state at "timestep" *t*



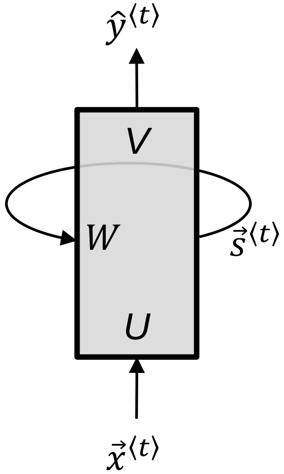
Three weight matrices:

- $U \in \mathbb{R}^{h \times n}$
- $W \in \mathbb{R}^{h \times h}$
- $V \in \mathbb{R}^{o \times h}$

$$\vec{\boldsymbol{s}}^{\langle t \rangle} = \boldsymbol{\phi_1}\left(\boldsymbol{W}\vec{\boldsymbol{s}}^{\langle t-1 \rangle} + \boldsymbol{U}\vec{\boldsymbol{x}}^{\langle t \rangle}\right)$$
$$\hat{\boldsymbol{y}}^{\langle t \rangle} = \boldsymbol{\phi_2}\left(\boldsymbol{V}\vec{\boldsymbol{s}}^{\langle t \rangle}\right)$$

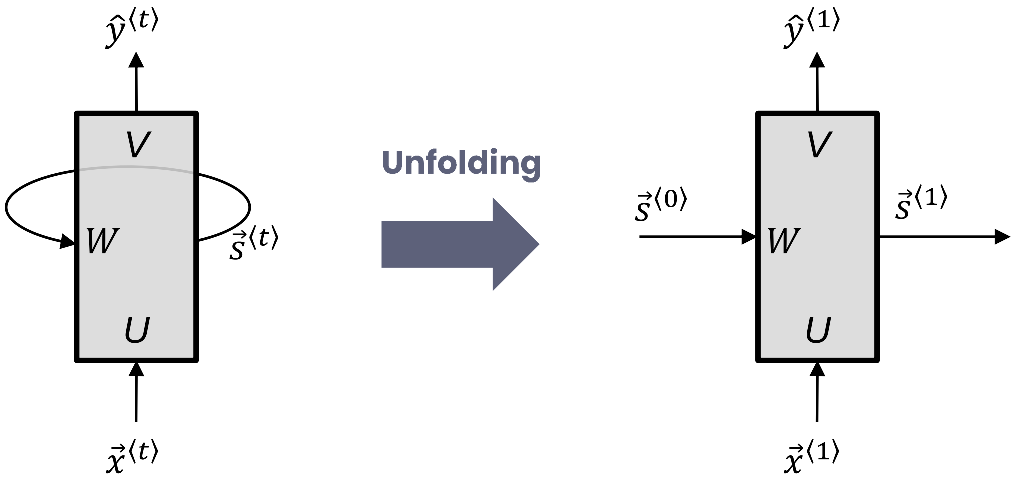
ϕ_1 and ϕ_2 are different activation functions.
Usually ϕ_1 is a tanh and ϕ_2 is a sigmoid (or softmax)

Forward propagation



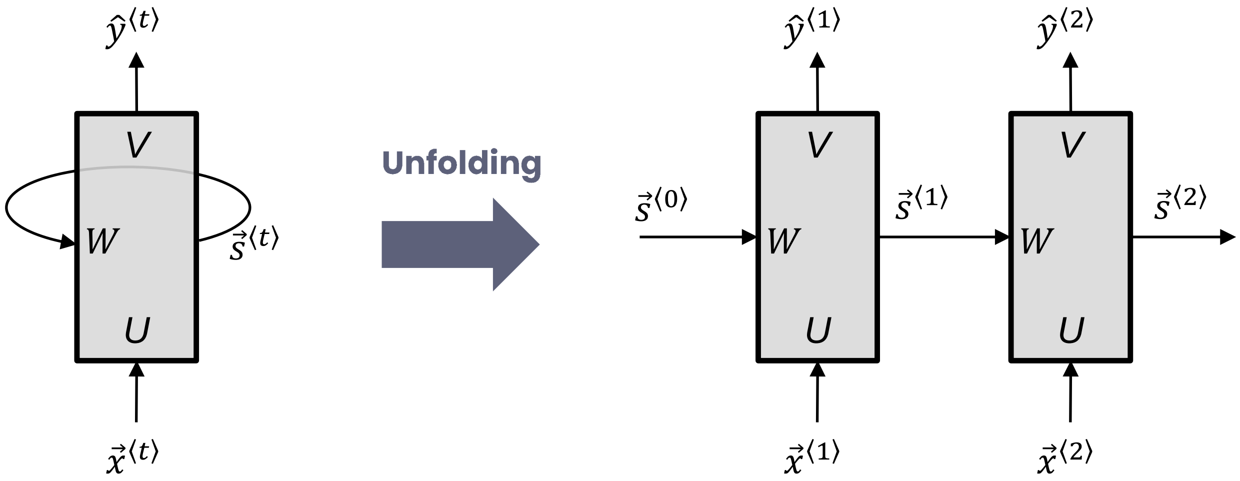
$$\vec{s}^{\langle 0 \rangle} = \vec{0}$$
$$s^{\langle t \rangle} = \phi_1\left(W\vec{s}^{\langle t-1 \rangle} + U\vec{x}^{\langle t \rangle}\right)$$
$$\hat{y}^{\langle t \rangle} = \phi_2\left(V\vec{s}^{\langle t \rangle}\right)$$

Forward propagation



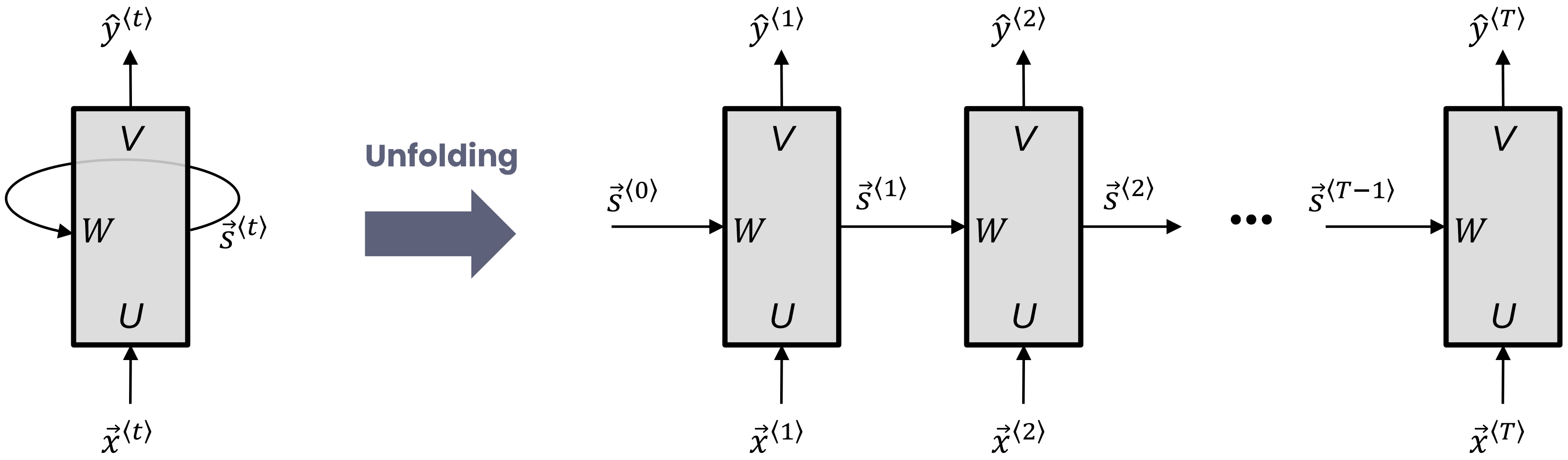
$$\vec{s}^{\langle 0 \rangle} = \vec{0}$$
$$s^{\langle t \rangle} = \phi_1\left(W\vec{s}^{\langle t-1 \rangle} + U\vec{x}^{\langle t \rangle}\right)$$
$$\hat{y}^{\langle t \rangle} = \phi_2\left(V\vec{s}^{\langle t \rangle}\right)$$

Forward propagation



$$\vec{s}^{\langle 0 \rangle} = \vec{0}$$
$$s^{\langle t \rangle} = \phi_1\left(W\vec{s}^{\langle t-1 \rangle} + U\vec{x}^{\langle t \rangle}\right)$$
$$\hat{y}^{\langle t \rangle} = \phi_2\left(V\vec{s}^{\langle t \rangle}\right)$$

Forward propagation



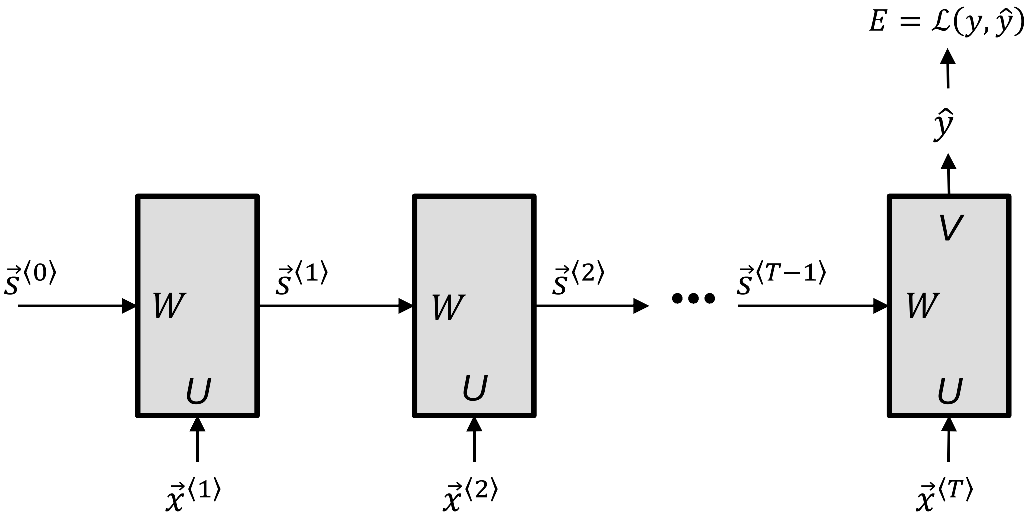
$$\vec{s}^{\langle 0 \rangle} = \vec{0}$$
$$s^{\langle t \rangle} = \phi_1\left(W\vec{s}^{\langle t-1 \rangle} + U\vec{x}^{\langle t \rangle}\right)$$
$$\hat{y}^{\langle t \rangle} = \phi_2\left(V\vec{s}^{\langle t \rangle}\right)$$

An RNN is a network, not just a neuron, even though it is sometimes called a ”cell”.

Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$



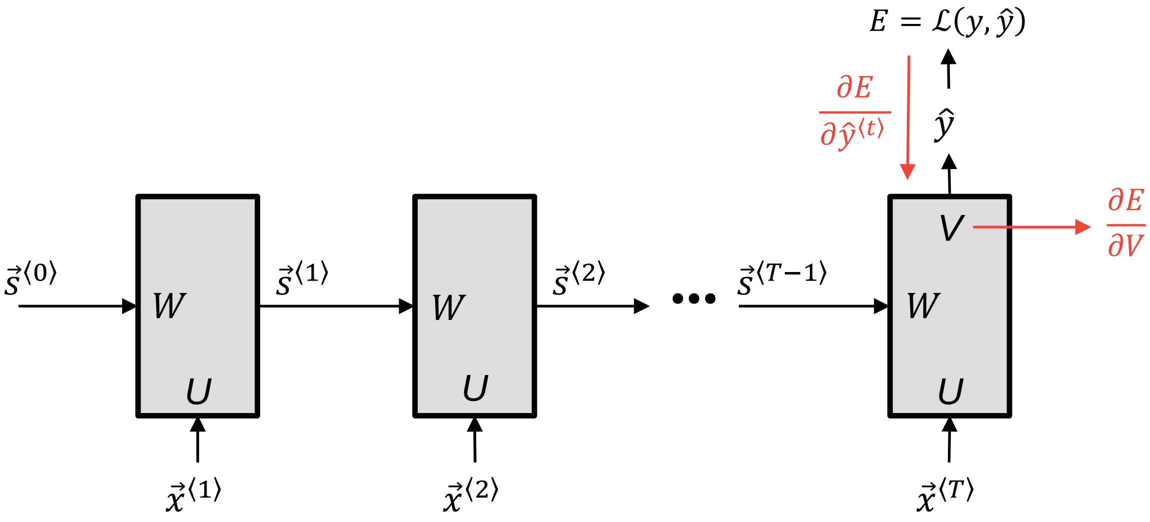
This is a simpler "many-to-one" case but it can be generalized to "many-to-many".

Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$

This is a simpler “many-to-one” case but it can be generalized to “many-to-many”.



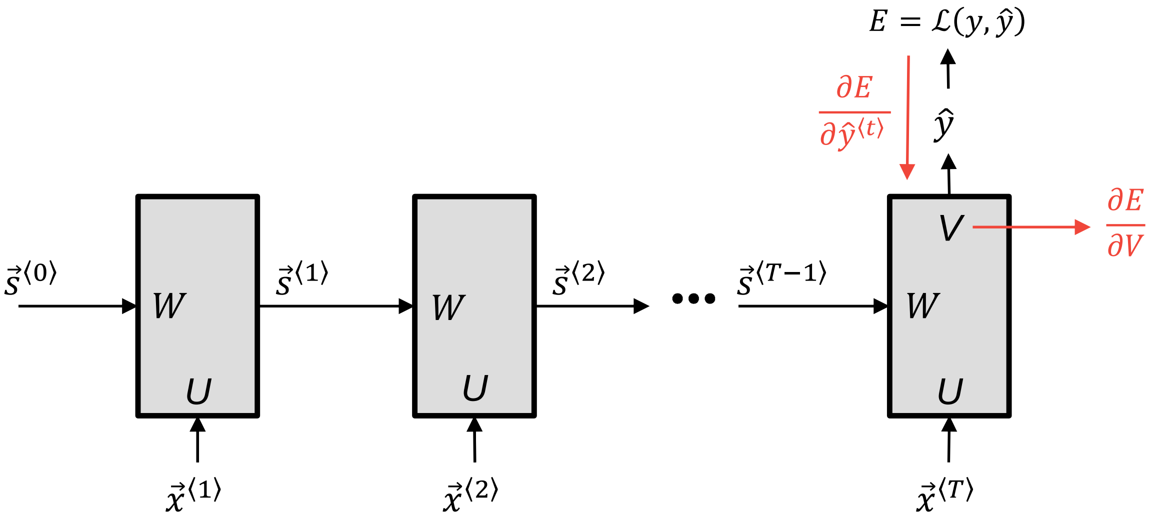
Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$

- ${}^{\partial E}/_{\partial \mathrm{W}}$ depends on ${}^{\partial \vec{s}^{\langle T \rangle}}/_{\partial \mathrm{W}}$, but $\vec{s}^{\langle T \rangle}$ depends on $\vec{s}^{\langle T-1 \rangle}$, which itself depends on W, an so on.
  - Same for ${}^{\partial E}/_{\partial U}$.

This is a simpler "many-to-one" case but it can be generalized to "many-to-many".



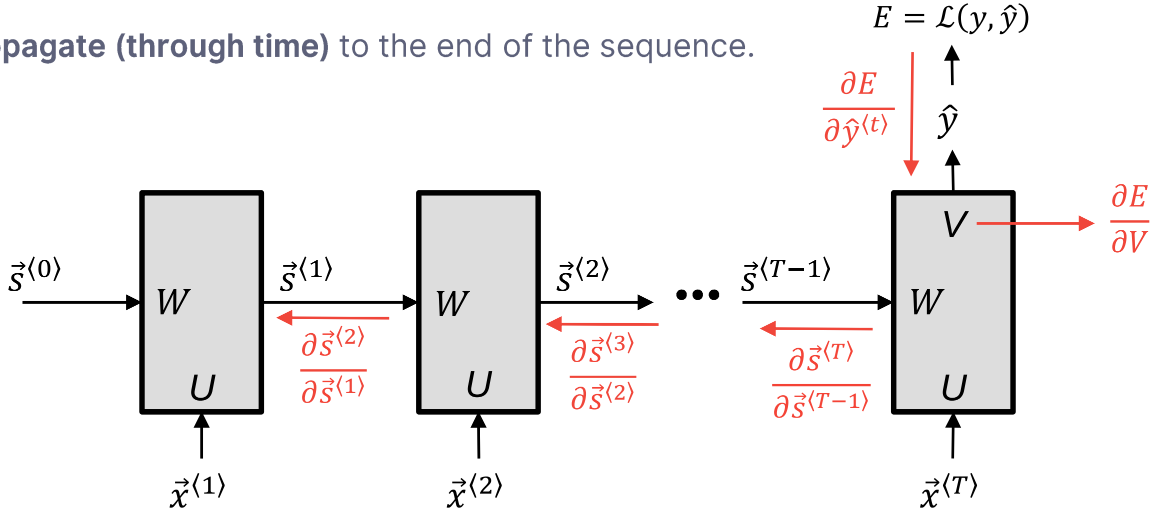
Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$

- ${}^{\partial E}/_{\partial \mathrm{W}}$ depends on ${}^{\partial \vec{s}^{\langle T \rangle}}/_{\partial \mathrm{W}}$, but $\vec{s}^{\langle T \rangle}$ depends on $\vec{s}^{\langle T-1 \rangle}$, which itself depends on W, an so on.
  - Same for ${}^{\partial E}/_{\partial U}$.
  - We need to **backpropagate (through time)** to the end of the sequence.

This is a simpler "many-to-one" case but it can be generalized to "many-to-many".

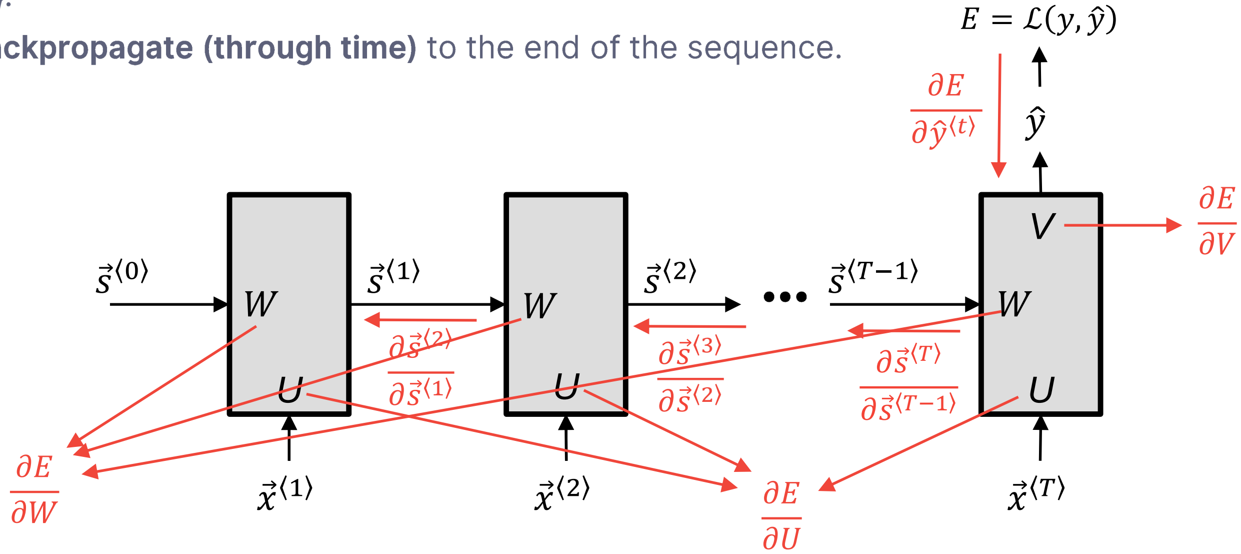


Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$

- ${}^{\partial E}/_{\partial \mathrm{W}}$ depends on ${}^{\partial \vec{s}^{\langle T\rangle}}/_{\partial \mathrm{W}}$, but $\vec{s}^{\langle T\rangle}$ depends on $\vec{s}^{\langle T-1\rangle}$, which itself depends on W, an so on.
    - Same for ${}^{\partial E}/_{\partial U}$.
    - We need to **backpropagate (through time)** to the end of the sequence.



This is a simpler "many-to-one" case but it can be generalized to "many-to-many".

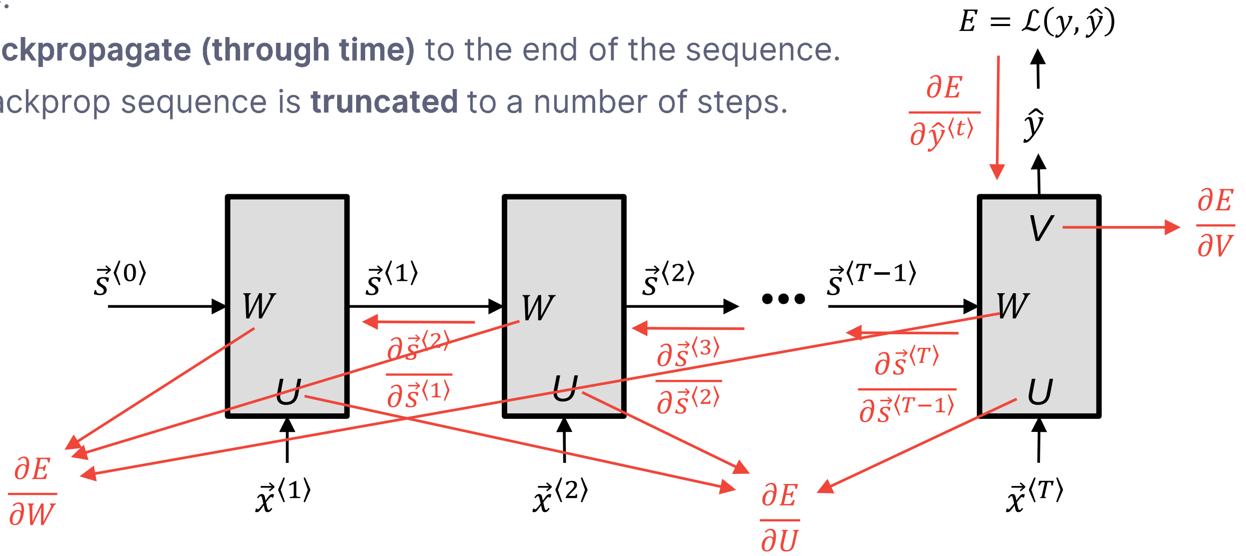
Backpropagation through time (BPTT)

- Like before, weights are updated through **backpropagation**:

$$\Delta \mathrm{V} = -\eta \frac{\partial E}{\partial \mathrm{V}} \qquad \Delta \mathrm{W} = -\eta \frac{\partial E}{\partial \mathrm{W}} \qquad \Delta \mathrm{U} = -\eta \frac{\partial E}{\partial \mathrm{U}}$$

- ${}^{\partial E}/_{\partial \mathrm{W}}$ depends on ${}^{\partial \vec{s}^{\langle T\rangle}}/_{\partial \mathrm{W}}$, but $\vec{s}^{\langle T\rangle}$ depends on $\vec{s}^{\langle T-1\rangle}$, which itself depends on W, an so on.
  - Same for ${}^{\partial E}/_{\partial U}$.
  - We need to **backpropagate (through time)** to the end of the sequence.
  - Usually, the backprop sequence is **truncated** to a number of steps.

This is a simpler “many-to-one” case but it can be generalized to “many-to-many”.



Long Short-Term Memory

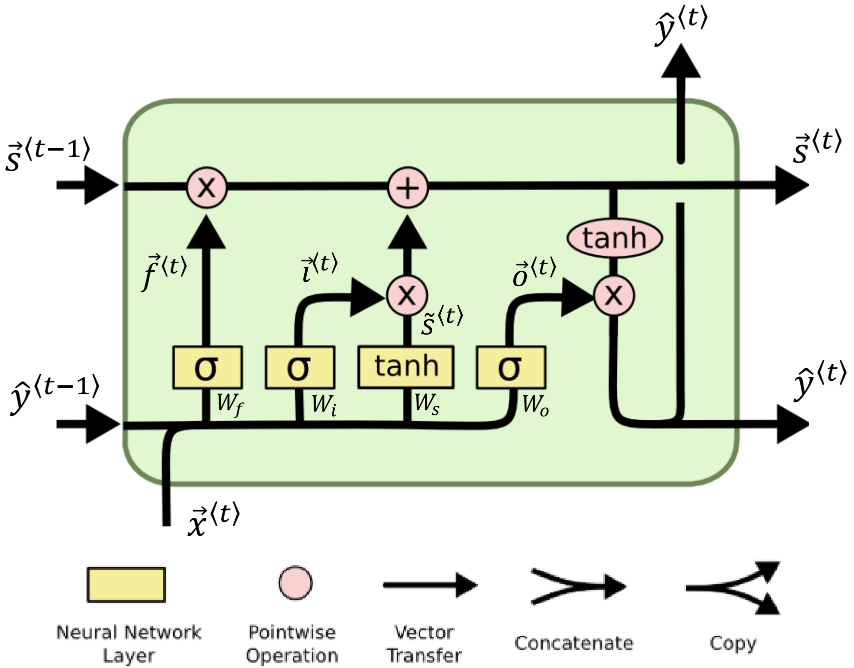
- Because of repeated application of the tanh activation, RNNs suffer from the **vanishing gradient** problem.

Long Short-Term Memory

- Because of repeated application of the tanh activation, RNNs suffer from the **vanishing gradient** problem.
- ReLUs can be used to ameliorate this, but a much better way is to use a different inner structure, called an **LSTM** cell.

Long Short-Term Memory

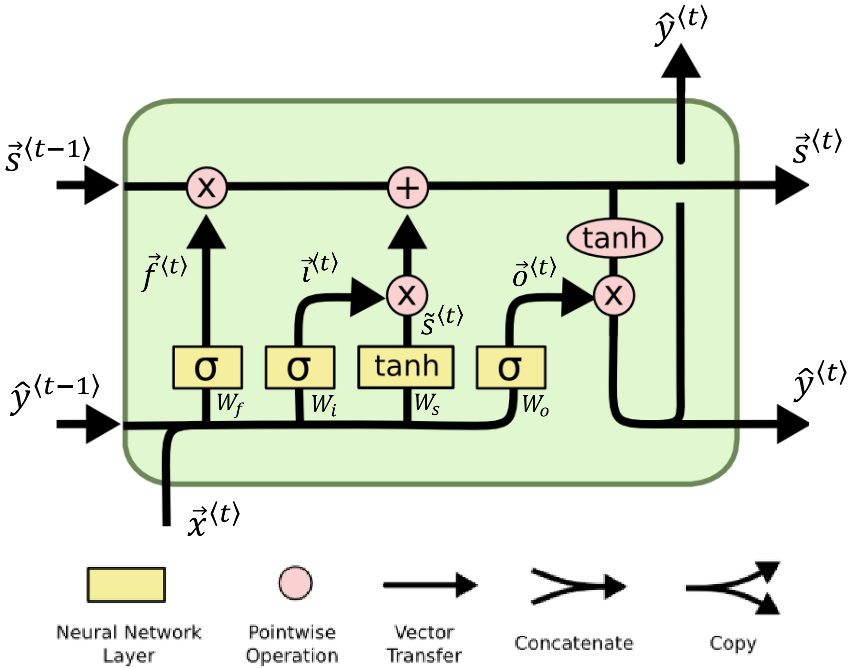
- Because of repeated application of the tanh activation, RNNs suffer from the **vanishing gradient** problem.
- ReLUs can be used to ameliorate this, but a much better way is to use a different inner structure, called an **LSTM** cell.



Visuals are from http://colah.github.io/posts/2015-08-Understanding-LSTMs, but with different notation.

Long Short-Term Memory

- Because of repeated application of the tanh activation, RNNs suffer from the **vanishing gradient** problem.
- ReLUs can be used to ameliorate this, but a much better way is to use a different inner structure, called an **LSTM** cell.



$$\vec{f}^{\langle t \rangle} = \sigma\left(W_f \cdot \left[\hat{y}^{\langle t-1 \rangle}, \vec{x}^{\langle t \rangle}\right] + b_f\right)$$ **"Forget" gate**

$$\vec{\imath}^{\langle t \rangle} = \sigma\left(W_i \cdot \left[\hat{y}^{\langle t-1 \rangle}, \vec{x}^{\langle t \rangle}\right] + b_i\right)$$ **"Input" gate**

$$\tilde{s}^{\langle t \rangle} = \tanh\left(W_s \cdot \left[\hat{y}^{\langle t-1 \rangle}, \vec{x}^{\langle t \rangle}\right] + b_s\right)$$ **State candidates**

$$\vec{s}^{\langle t \rangle} = \vec{f}^{\langle t \rangle} * \vec{s}^{\langle t-1 \rangle} + \vec{\imath}^{\langle t \rangle} * \tilde{s}^{\langle t \rangle}$$ **Hidden state (Memory)**

$$\vec{o}^{\langle t \rangle} = \sigma\left(W_o \cdot \left[\hat{y}^{\langle t-1 \rangle}, \vec{x}^{\langle t \rangle}\right] + b_o\right)$$ **"Output" gate**

$$\hat{y}^{\langle t \rangle} = \vec{o}^{\langle t \rangle} * \tanh\left(\vec{s}^{\langle t \rangle}\right)$$ **Output**

**Visuals are from http://colah.github.io/posts/2015-08-Understanding-LSTMs, but with different notation.*

Keywords

Recurrent Neural Network

RNN

Backpropagation Through Time

BPTT

Truncated BPTT

Long Short-Term Memory

LSTM