Convolutional and Recurrent Deep Neural Networks

Marco Frasca
Università degli Studi di Milano

marco.frasca@unimi.it

DeepLife Course
4EU+

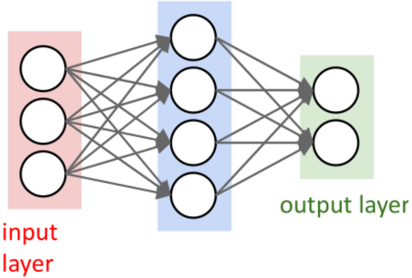
4 March 2024

Deep Learning in Life Sciences

CHARLES UNIVERSITY · SORBONNE UNIVERSITÉ · UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386 · UNIVERSITY OF WARSAW · UNIVERSITÀ DEGLI STUDI DI MILANO · 4eu+ EUROPEAN UNIVERSITY ALLIANCE

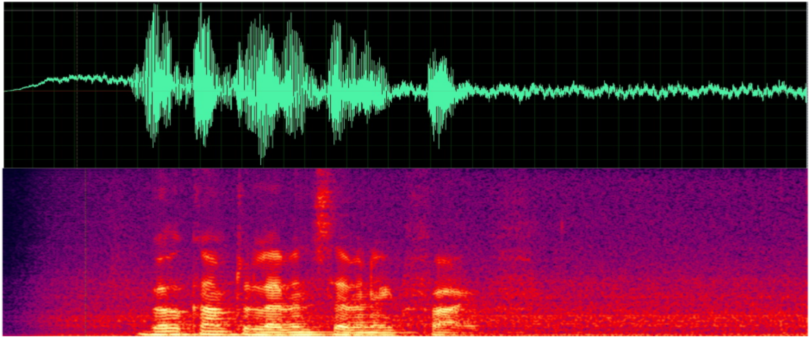
Outline

Recap



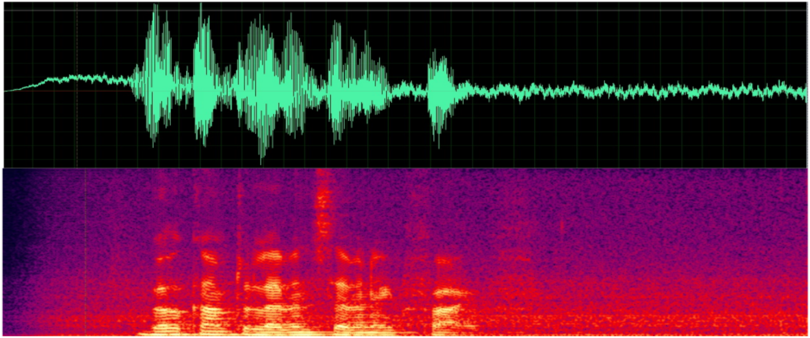
- Multi-layer Perceptron (MLP) is able to detect patterns in input data
- Inputs are numeric vectors
  - E.g. digits, numeric signals etc.

A different problem



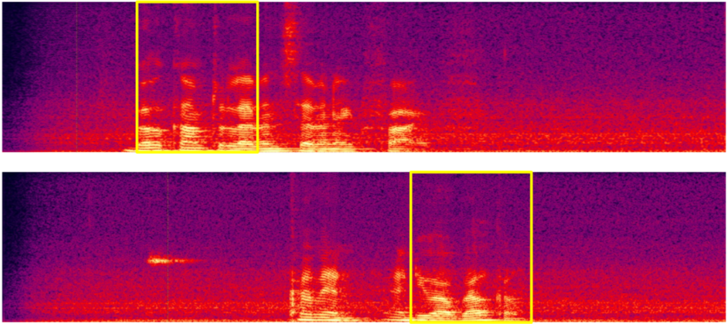
- Given a signal, does it contain a given word? E.g., "welcome"?

A different problem



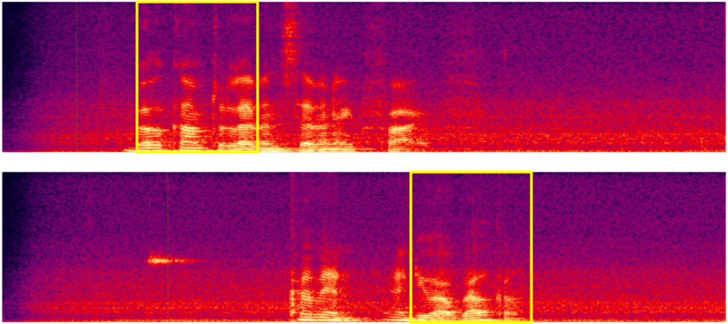
- Given a signal, does it contain a given word? E.g., "welcome"?
- **Solution 1**: MLP which takes the whole time-frequency signal and output 1 if there is a "welcome" in
- Does it work?

Shift Variance Issue



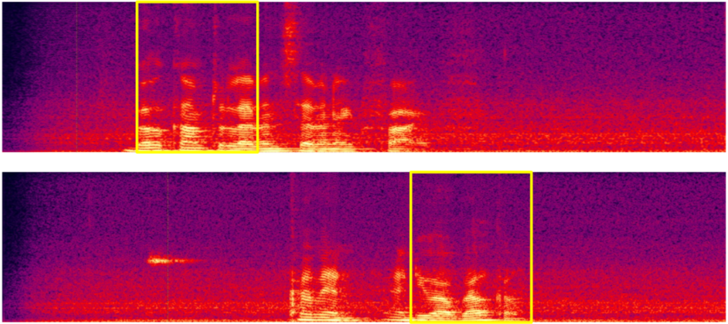
- Shift variance problem: a MLP that finds a “welcome” in the top recording will not find it in the lower one

Shift Variance Issue



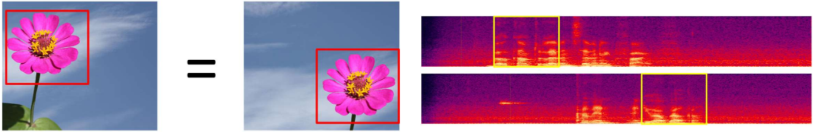
- Shift variance problem: a MLP that finds a "welcome" in the top recording will not find it in the lower one
- Unless trained with both

Shift Variance Issue



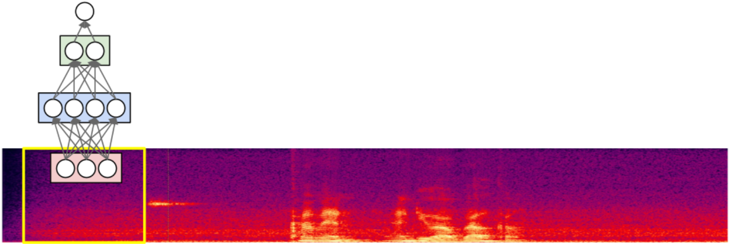
- Shift variance problem: a MLP that finds a “welcome” in the top recording will not find it in the lower one
- Unless trained with both
- This would require a very large network and a large amount of training data to cover every case

Shift Variance Issue



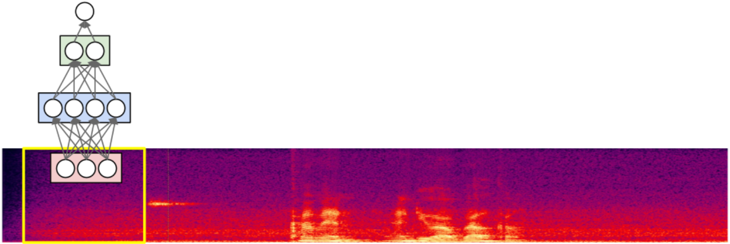
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP won't recognize
- Requirement: Network must be *shift invariant*

Possible solution: scan the input



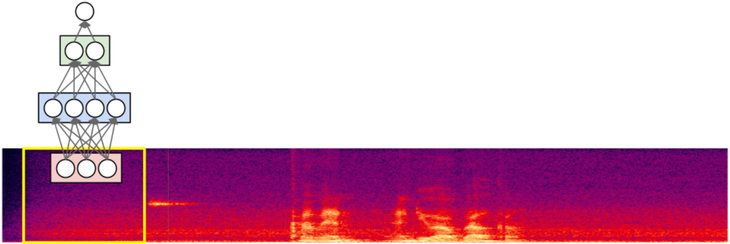
- Split the input in (overlapping) windows

Possible solution: scan the input



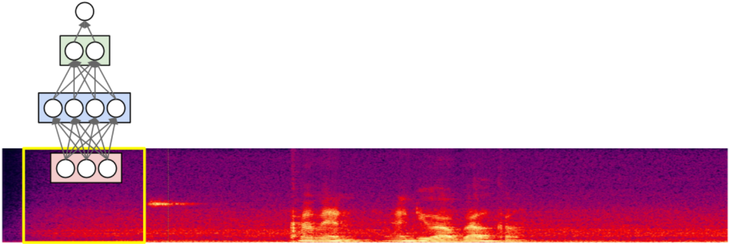
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP

Possible solution: scan the input



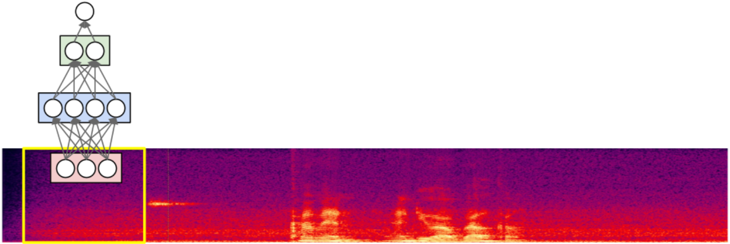
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions?

Possible solution: scan the input



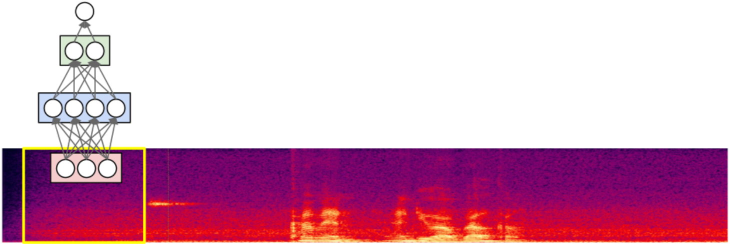
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!

Possible solution: scan the input



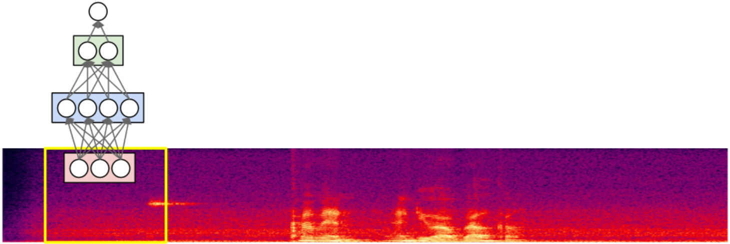
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!
- This is what a Convolutional Layer does!

Possible solution: scan the input



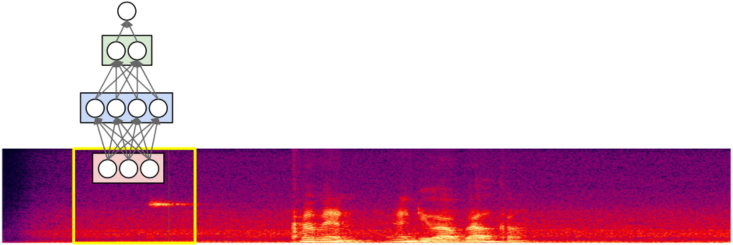
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!
- This is what a Convolutional Layer does!
- Nice thing: the MLP scanning each window is the same! The parameters (connection weights) are thereby shared

Possible solution: scan the input



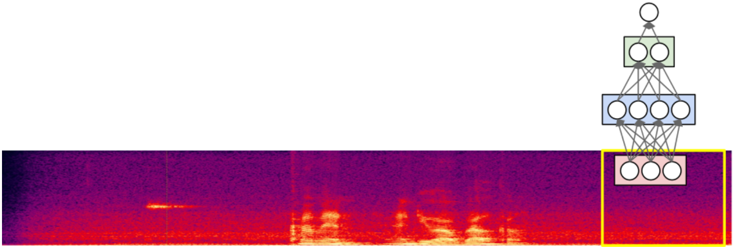
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!
- This is what a Convolutional Layer does!
- Nice thing: the MLP scanning each window is the same! The parameters (connection weights) are thereby shared

Possible solution: scan the input



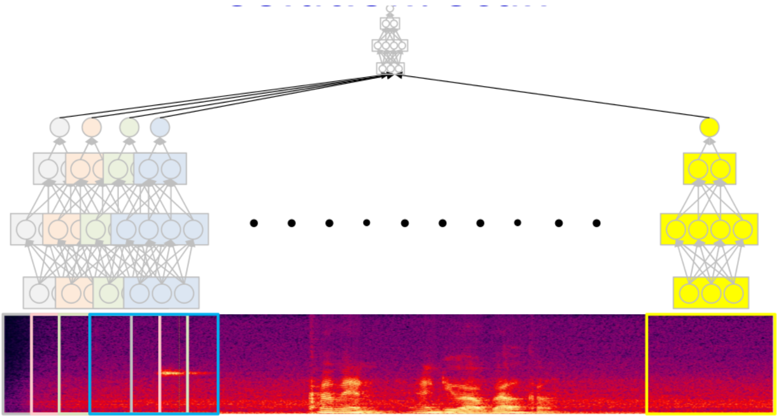
- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!
- This is what a Convolutional Layer does!
- Nice thing: the MLP scanning each window is the same! The parameters (connection weights) are thereby shared

Possible solution: scan the input



- Split the input in (overlapping) windows
- Scan for the target word: each window is input to the same "welcome-detector" MLP
- What to do on each window to take into account even overlapping portions? Operate a convolution on each window!
- This is similar to what Convolutional Layer does
- Nice thing: the MLP scanning each window is the same! The parameters (connection weights) are thereby shared

Combining the Windows



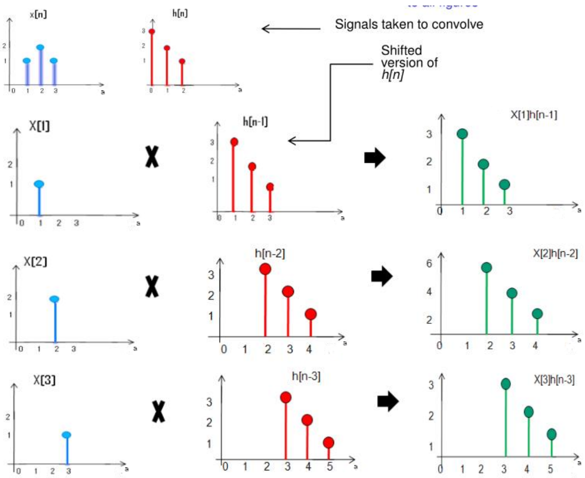
"Does welcome occur in this recording?"

- Maximum of all the outputs (Equivalent of Boolean OR)
- Or a proper softmax/logistic
  - Adjacent windows can combine their evidence
- Or even an MLP

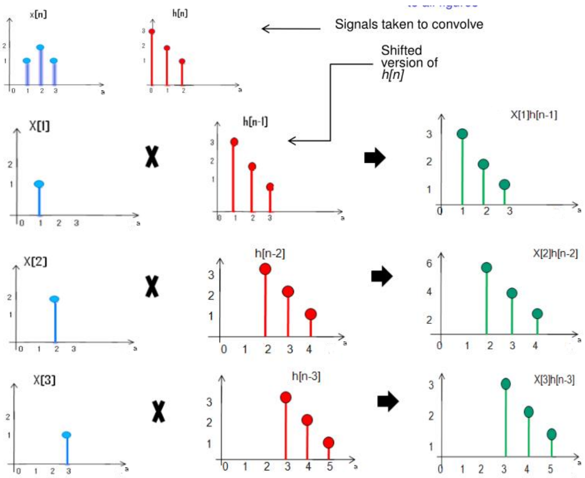
2

Outline

Convolution https://www.slideserve.com/sylvie/discrete-convolution-of-two-signals-powerpoint-ppt-presentation

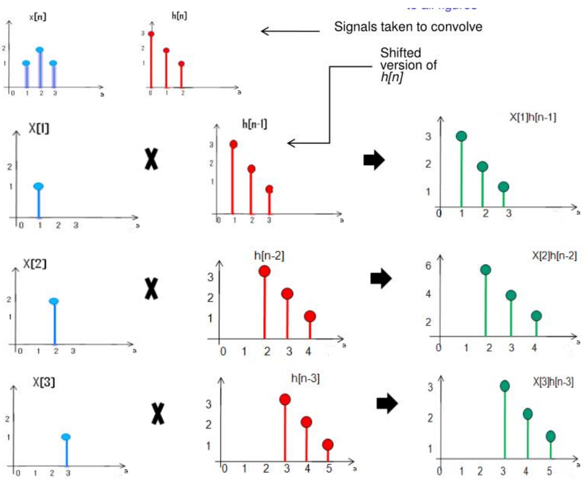


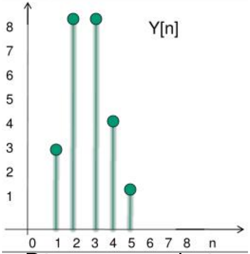
- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x



- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x
- Typically h is shifted of a constant (1, 2, or 3 in the figure)

Convolution https://www.slideserve.com/sylvie/discrete-convolution-of-two-signals-powerpoint-ppt-presentation

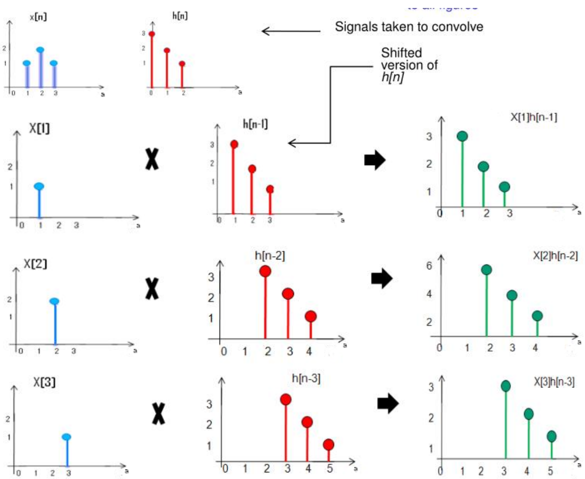


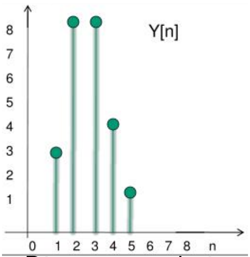


– Dicrete convolution: a new function Y of n

$Y(n) := (x * h)(n) := \sum_{a \in A} x(a)h(n-a)$

- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x
- Typically h is shifted of a constant (1, 2, or 3 in the figure)





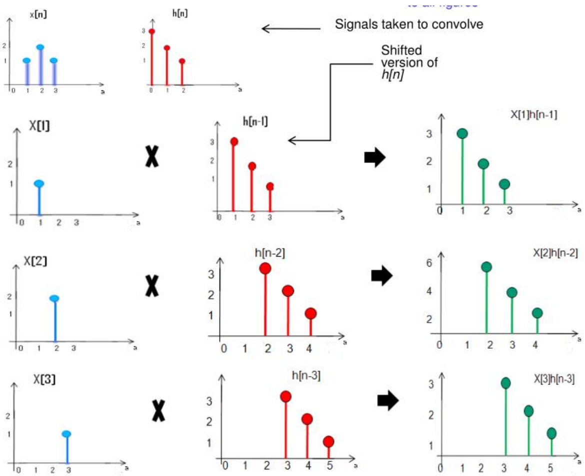
– Dicrete convolution: a new function Y of n

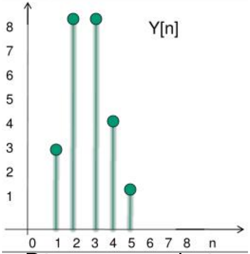
$Y(n) := (x * h)(n) := \sum_{a \in A} x(a)h(n-a)$

– In the example $A = \{1, 2, 3\}$

- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x
- Typically h is shifted of a constant (1, 2, or 3 in the figure)

Convolution https://www.slideserve.com/sylvie/discrete-convolution-of-two-signals-powerpoint-ppt-presentation





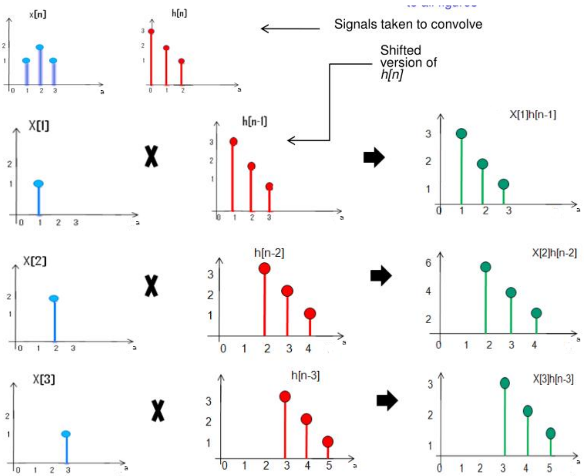
– Dicrete convolution: a new function Y of n $Y(n) := (x * h)(n) := \sum_{a \in A} x(a)h(n-a)$

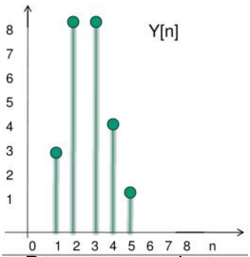
– In the example $A = \{1, 2, 3\}$

– $Y(1) = \sum_{a \in A} x(a)h(1-a) = 1 \cdot 3 + 2 \cdot 0 + 1 \cdot 0 = 3$

- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x
- Typically h is shifted of a constant (1, 2, or 3 in the figure)

Convolution https://www.slideserve.com/sylvie/discrete-convolution-of-two-signals-powerpoint-ppt-presentation





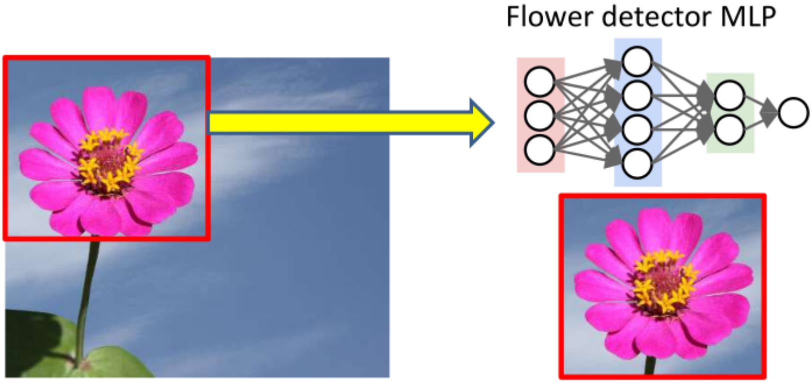
- Dicrete convolution: a new function Y of n $Y(n) := (x * h)(n) := \sum_{a \in A} x(a)h(n-a)$
- In the example $A = \{1, 2, 3\}$
- $Y(1) = \sum_{a \in A} x(a)h(1-a) = 1 \cdot 3 + 2 \cdot 0 + 1 \cdot 0 = 3$
- $Y(2) = ?$

- A convolution is a composition of two real-valued functions x and h to get a smoothed version of x
- Typically h is shifted of a constant (1, 2, or 3 in the figure)

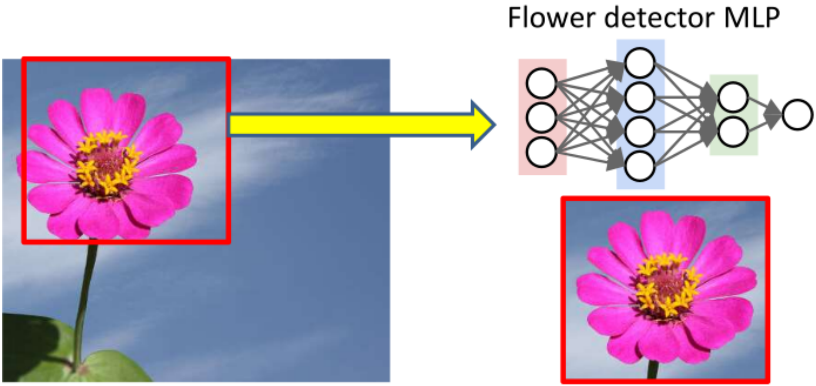
Convolution in Conv layers

- In the case of a convolutional Neural Nets (CNN), the "scanning MLP" operates a convolution
- The input values in a window are the h values
- The connection weights of the sliding window (to be learned) represent the x values

2D (Discrete) Convolution

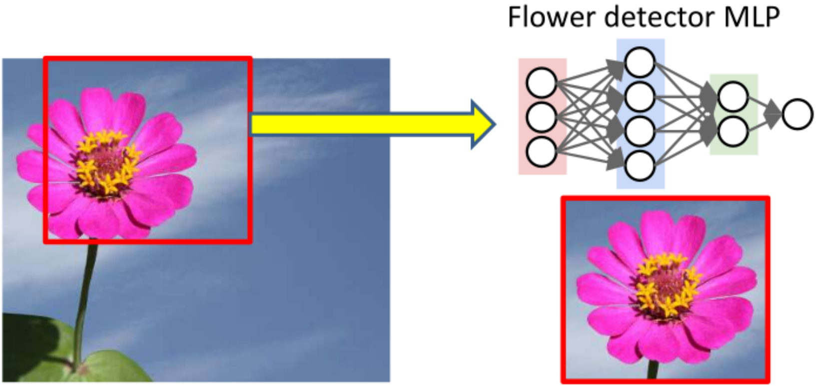


For images our window moves in 2 directions!



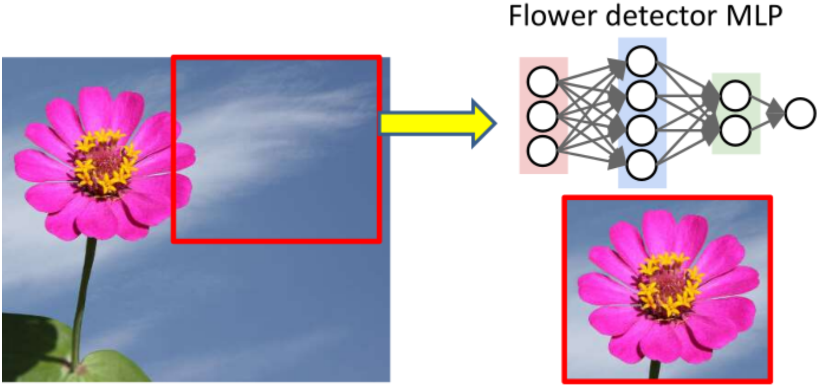
For images our window moves in 2 directions!

2D Convolution



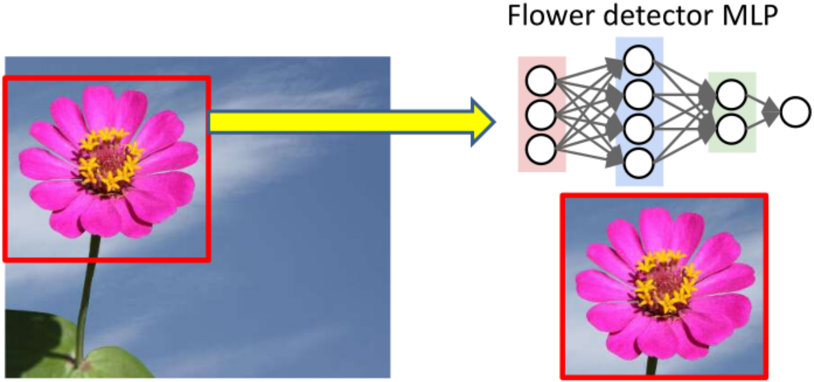
For images our window moves in 2 directions!

2D Convolution

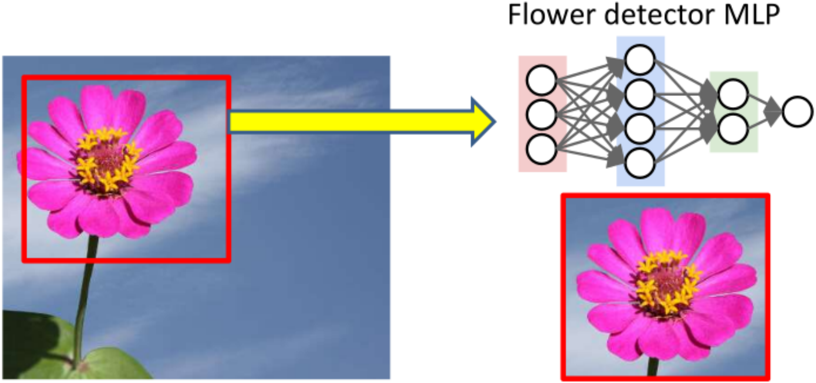


For images our window moves in 2 directions!

2D Convolution

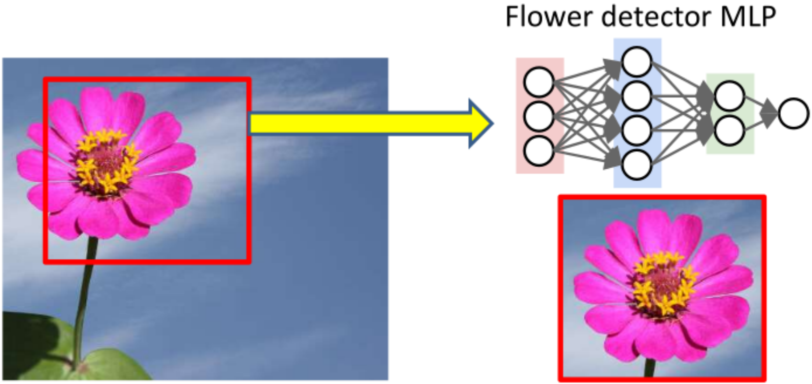


For images our window moves in 2 directions!



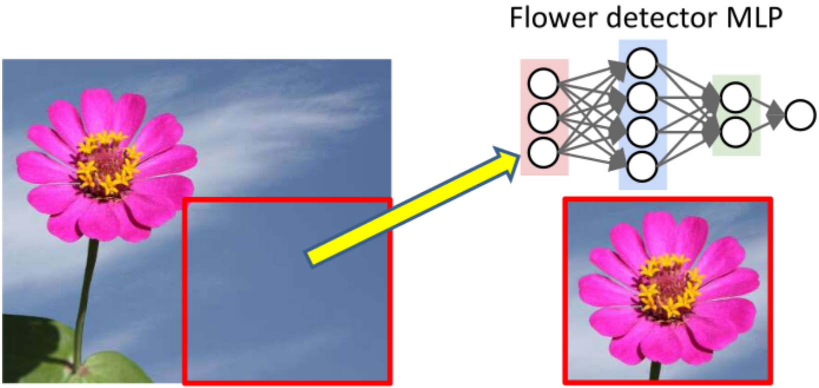
For images our window moves in 2 directions!

2D Convolution



For images our window moves in 2 directions!

2D Convolution



For images our window moves in 2 directions!

Shared parameters of a Convolutional Layer

RECAP:

- Backing to the "imaginary" welcome-detector scanning-MLP, it operates the same job in each window

Shared parameters of a Convolutional Layer

RECAP:

- Backing to the “imaginary” welcome-detector scanning-MLP, it operates the same job in each window
- We can thereby use the same MLP for all inputs windows!

RECAP:

- Backing to the "imaginary" welcome-detector scanning-MLP, it operates the same job in each window
- We can thereby use the same MLP for all inputs windows!
  - Shared parameters!

RECAP:

- Backing to the "imaginary" welcome-detector scanning-MLP, it operates the same job in each window
- We can thereby use the same MLP for all inputs windows!
  - Shared parameters!
- A Convolutional layer has much less parameters than a fully-connected one!

Discrete Convolution: 1D case

In a convolutional layer, with little abuse of terminology, x (g in the figure) is often referred to as a "filter", or a "kernel" or a "filter kernel"

f	4	3	2	-5	3	5	2	5	5	6
g	1/3	1/3	1/3	→						

'Slide' **filter kernel** from left to right; at each position, compute a single value in the output data

Which value this operation outputs?

Discrete Convolution: 1D case

In a convolutional layer, with little abuse of terminology, x (g in the figure) is often referred to as a "filter", or a "kernel" or a "filter kernel"

f	4	3	2	-5	3	5	2	5	5	6
g	1/3	1/3	1/3							

'Slide' **filter kernel** from left to right; at each position, compute a single value in the output data

Which value this operation outputs? Exactly the result of the convolution with the input!

Discrete Convolution: 1D case

f	**4**	**3**	**2**	**-5**	**3**	**5**	**2**	**5**	**5**	**6**
g	**1/3**	**1/3**	**1/3**							
$f * g$		**3**								

$$4 \cdot \frac{1}{3} + 3 \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} = 3$$

Discrete Convolution: 1D case

f	4	3	2	-5	3	5	2	5	5	6
g		1/3	1/3	1/3						
$f * g$		3	0							

$$3 \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} + (-5) \cdot \frac{1}{3} = 0$$

Discrete Convolution: 1D case

f	4	3	2	-5	3	5	2	5	5	6
g								1/3	1/3	1/3
$f * g$		3	0	0	1	10/3	4	4	16/3	

$$5 \cdot \frac{1}{3} + 5 \cdot \frac{1}{3} + 6 \cdot \frac{1}{3} = \frac{16}{3}$$

The step used to slide the window is called a STRIDE

Discrete Convolution: 1D case

4	3	2	-5	3	5	2	5	5	6

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

What to do at boundaries?

Discrete Convolution: 1D case

4	3	2	-5	3	5	2	5	5	6

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

What to do at boundaries?

Possible Solution: 0-Padding

0	4	3	2	-5	3	5	2	5	5	6	0

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

$$0 \cdot \frac{1}{3} + 4 \cdot \frac{1}{3} + 3 \cdot \frac{1}{3} = \frac{7}{3}$$

What to do at boundaries?

Pad (often 0's)

7/3	3	0	0	1	10/3	4	4	16/3	11/3

Discrete Convolution: 1D case

4	3	2	-5	3	5	2	5	5	6

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

What to do at boundaries?

Possible Solution: 0-Padding

0	4	3	2	-5	3	5	2	5	5	6	0

1/3	1/3	1/3

??	3	0	0	1	10/3	4	4	16/3	??

$$0 \cdot \frac{1}{3} + 4 \cdot \frac{1}{3} + 3 \cdot \frac{1}{3} = \frac{7}{3}$$

What to do at boundaries?

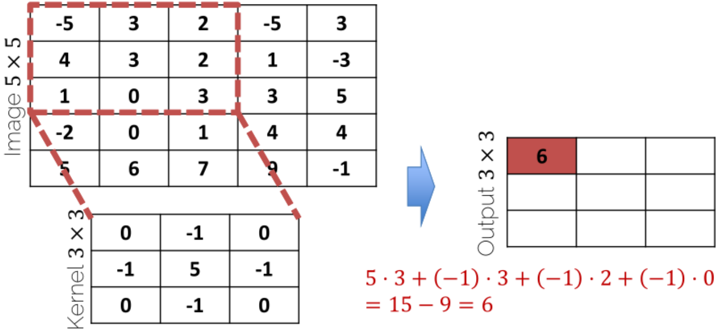
Pad (often 0's)

7/3	3	0	0	1	10/3	4	4	16/3	11/3

Padding allows the output channel to keep the size of the input

Discrete Convolution: 2D case

- In 2D the sliding window moves in two directions
- The stride can be different in the two directions



$$5 \cdot 3 + (-1) \cdot 3 + (-1) \cdot 2 + (-1) \cdot 0 + (-1) \cdot 4$$
$$= 15 - 9 = 6$$

Discrete Convolution: 2D case

Image 5 × 5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3 × 3

0	-1	0
-1	5	-1
0	-1	0

Output 3 × 3

6	1	

$$5 \cdot 2 + (-1) \cdot 2 + (-1) \cdot 1 + (-1) \cdot 3 + (-1) \cdot 3$$
$$= 10 - 9 = 1$$

Discrete Convolution: 2D case

Image 5×5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3×3

0	-1	0
-1	5	-1
0	-1	0

Output 3×3

6	1	8

$$5 \cdot 1 + (-1) \cdot (-5) + (-1) \cdot (-3) + (-1) \cdot 3$$
$$+(-1) \cdot 2$$
$$= 5 + 3 = 8$$

Discrete Convolution: 2D case

Image 5×5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

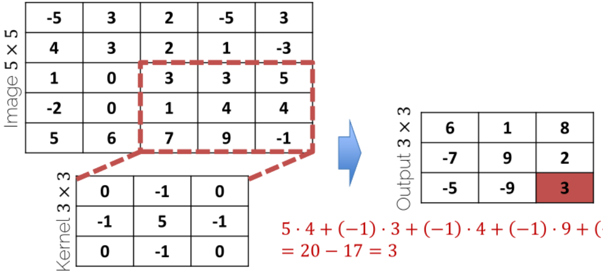
Kernel 3×3

0	-1	0
-1	5	-1
0	-1	0

Output 3×3

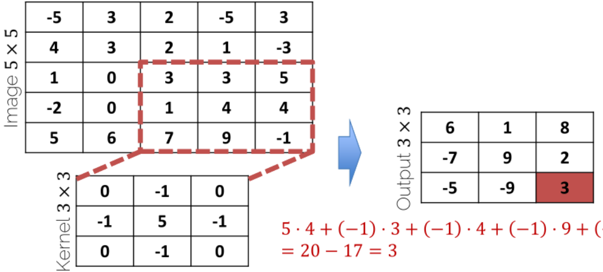
6	1	8
-7		

$$5 \cdot 0 + (-1) \cdot 3 + (-1) \cdot 0 + (-1) \cdot 1 + (-1) \cdot 3$$
$$= 0 - 7 = -7$$



$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1$$
$$= 20 - 17 = 3$$

- the kernel size in the example is 3×3 or simply 3 by shortening



$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1$$
$$= 20 - 17 = 3$$

- the kernel size in the example is 3×3 or simply 3 by shortening
- The output is also known as a feature map

Discrete Convolution: 2D case

Image 5 × 5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3 × 3

0	-1	0
-1	5	-1
0	-1	0

Output 3 × 3

6	1	8
-7	9	2
-5	-9	3

$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1$$
$$= 20 - 17 = 3$$

- the kernel size in the example is 3×3 or simply 3 by shortening
- The output is also known as a feature map
- The feature map size is 3×3 as well in this example

Discrete Convolution: 2D case

Image 5 × 5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

Kernel 3 × 3

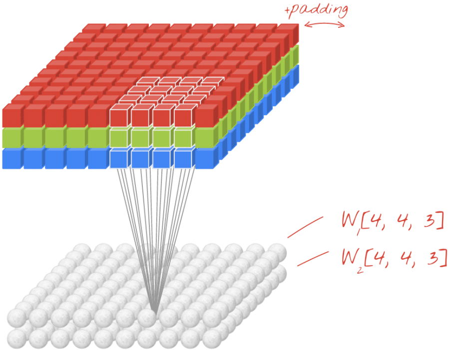
0	-1	0
-1	5	-1
0	-1	0

Output 3 × 3

6	1	8
-7	9	2
-5	-9	3

$$5 \cdot 4 + (-1) \cdot 3 + (-1) \cdot 4 + (-1) \cdot 9 + (-1) \cdot 1$$
$$= 20 - 17 = 3$$

- the kernel size in the example is 3×3 or simply 3 by shortening
- The output is also known as a feature map
- The feature map size is 3×3 as well in this example
- Even in this case we could 0-pad the input grid for the feature map to have the same size of the input

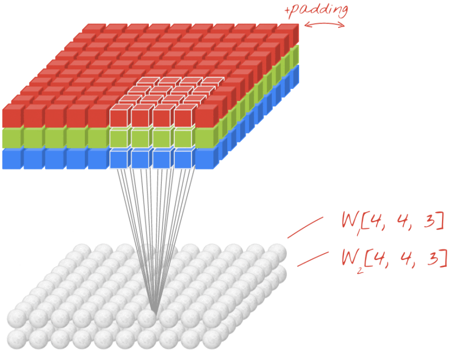


Source: https://stats.stackexchange.com/questions/240926/how-are-convolutional-layers-connected-in-theano

$W_1[4, 4, 3]$
$W_2[4, 4, 3]$

$W[\underbrace{4, 4}_{\text{filter size}}, \underbrace{3}_{\text{input channels}}, \underbrace{2}_{\text{output channels}}]$

- In 1D,2D, or 3D cases, the number of filters used is the number of output channels (regardless of the number of input maps!)



Source: https://stats.stackexchange.com/questions/240926/how-are-convolutional-layers-connected-in-theano

$W_1[4, 4, 3]$
$W_2[4, 4, 3]$

$W[4, 4, 3, 2]$

filter size — input channels — output channels

- In 1D,2D, or 3D cases, the number of filters used is the number of output channels (regardless of the number of input maps!)
- Each feature map produced by a filter is an output channel

- Suppose we use a 2D filter of size k. How many parameters?

Convolutional Layer

- Suppose we use a 2D filter of size k. How many parameters? Exactly k^2 weights

Convolutional Layer

- Suppose we use a 2D filter of size k. How many parameters? Exactly k^2 weights
- Suppose to use N distinct filters scanning the input image, we need Nk^2 weights

- Suppose we use a 2D filter of size k. How many parameters? Exactly k^2 weights
- Suppose to use N distinct filters scanning the input image, we need Nk^2 weights
  - In this case, the feature map (output) has a depth N

Convolutional Layer

- Suppose we use a 2D filter of size k. How many parameters? Exactly k^2 weights
- Suppose to use N distinct filters scanning the input image, we need Nk^2 weights
  - In this case, the feature map (output) has a depth N
  - In other words, each filter produces an output channel, which are stacked atop each other

Outline

Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

A nice property which can come from using more convolutional layers in sequence is the following:

- Suppose we want to use a window of 9×9 pixels

Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

A nice property which can come from using more convolutional layers in sequence is the following:

- Suppose we want to use a window of 9×9 pixels
- We can use a 3×3 window in the first convolutional layer

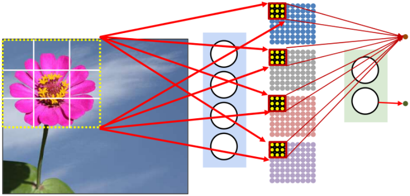
Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

A nice property which can come from using more convolutional layers in sequence is the following:

- Suppose we want to use a window of 9×9 pixels
- We can use a 3×3 window in the first convolutional layer
- Then we scan the output feature map with another 3×3 window



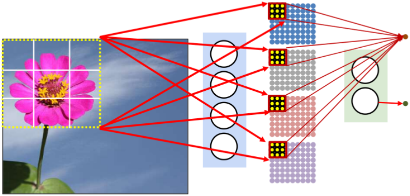
Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

A nice property which can come from using more convolutional layers in sequence is the following:

- Suppose we want to use a window of 9×9 pixels
- We can use a 3×3 window in the first convolutional layer
- Then we scan the output feature map with another 3×3 window



- Each neuron in the second convolutional layer is performing the equivalent of a convolution of a 9×9 filter!

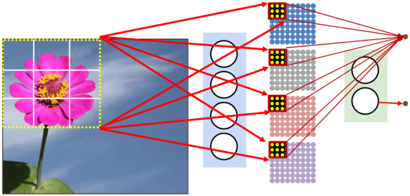
Distributing the Convolution

Nothing prevent us to use more convolutional layers in sequence!

- At each level we extract novel (more general) features

A nice property which can come from using more convolutional layers in sequence is the following:

- Suppose we want to use a window of 9×9 pixels
- We can use a 3×3 window in the first convolutional layer
- Then we scan the output feature map with another 3×3 window



- Each neuron in the second convolutional layer is performing the equivalent of a convolution of a 9×9 filter!
- Total parameters per filter: 9 in the first layer, and 9 in the second layer. 18 vs 81!!

- We can distribute the convolution over mulitple layers

Distributing the Convolution

- We can distribute the convolution over mulitple layers
- Even tens of layers!

- We can distribute the convolution over mulitple layers
- Even tens of layers!
- Distributing is convenient for three reasons:
  1. Distribution forces hierarchical representations (more generalizable)
     - Each convolutional layer learns novel and more general features from the previous one

Distributing the Convolution

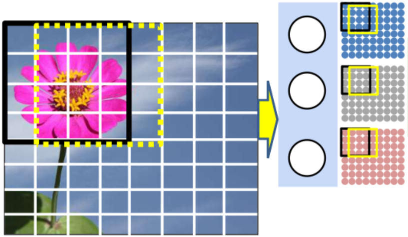
- We can distribute the convolution over mulitple layers
- Even tens of layers!
- Distributing is convenient for three reasons:
  1. Distribution forces hierarchical representations (more generalizable)
     - Each convolutional layer learns novel and more general features from the previous one
  2. Fewer computations (we ca resuse computations in overlapping windows)

Distributing the Convolution

- We can distribute the convolution over mulitple layers
- Even tens of layers!
- Distributing is convenient for three reasons:
  1. Distribution forces hierarchical representations (more generalizable)
     - Each convolutional layer learns novel and more general features from the previous one
  2. Fewer computations (we ca resuse computations in overlapping windows)
  3. Less parameters!

Advantage of Distributed Convolution

Overlapping windows in the second conv layer share most computations: computed at the first conv layer!! (small squares in the pic)



Padding

Main types of Padding:

- **VALID**: no padding applied

Main types of Padding:

- **VALID**: no padding applied
- **SAME**: Input map size = Output map size
  - $P = \left\lfloor \frac{F-1}{2} \right\rfloor$, $F :=$ filter size

Main types of Padding:

- **VALID**: no padding applied
- **SAME**: Input map size = Output map size
  - $P = \left\lfloor \frac{F-1}{2} \right\rfloor$, $F :=$ filter size
- Pay attention to use valid convolution: Shrink the feature map, but can loose information at borders!

Padding

Main types of Padding:

- **VALID**: no padding applied
- **SAME**: Input map size $=$ Output map size
  - $P = \left\lfloor \frac{F-1}{2} \right\rfloor$, $F :=$ filter size
- Pay attention to use valid convolution: Shrink the feature map, but can loose information at borders!
- Its usage depends on the semantic of the input!

Pooling

- Pooling neurons replace the weighted sum of the inputs with a pooling function $f(x_1, \ldots, x_k) = pool(x_1, \ldots, x_k)$
  - It is introduced to help making the representation approximately invariant to small translations of the input

Pooling

- Pooling neurons replace the weighted sum of the inputs with a pooling function $f(x_1, \ldots, x_k) = pool(x_1, \ldots, x_k)$
  - It is introduced to help making the representation approximately invariant to small translations of the input
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change

Pooling

- Pooling neurons replace the weighted sum of the inputs with a pooling function $f(x_1, \ldots, x_k) = pool(x_1, \ldots, x_k)$
  - It is introduced to help making the representation approximately invariant to small translations of the input
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change

Conv Layer = 'Feature Extraction'

- Computes a features in a given region

Pooling

- Pooling neurons replace the weighted sum of the inputs with a pooling function $f(x_1, \ldots, x_k) = pool(x_1, \ldots, x_k)$
  - It is introduced to help making the representation approximately invariant to small translations of the input
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change

Conv Layer = 'Feature Extraction'

- Computes a features in a given region

Pooling Layer = 'Feature Selection'

- Picks the most relevant activations in a region

Pooling

- Pooling neurons replace the weighted sum of the inputs with a pooling function $f(x_1, \ldots, x_k) = pool(x_1, \ldots, x_k)$
  - It is introduced to help making the representation approximately invariant to small translations of the input
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change

Conv Layer = 'Feature Extraction'

- Computes a features in a given region

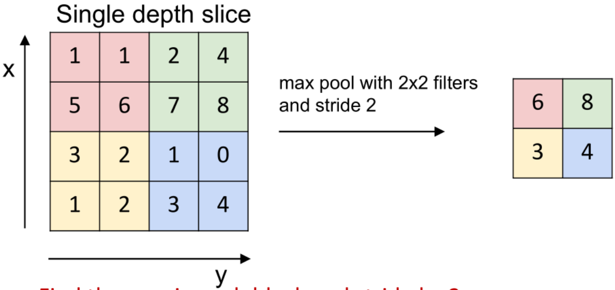
Pooling Layer = 'Feature Selection'

- Picks the most relevant activations in a region

For these reasons pooling layers usually follow convolutional layers

Max/Mean Pooling

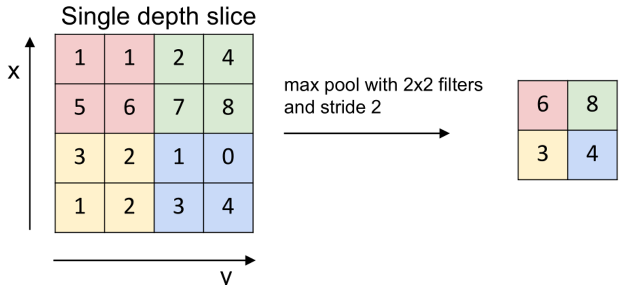
Source:https://www.geeksforgeeks.org/introduction-convolution-neural-network/



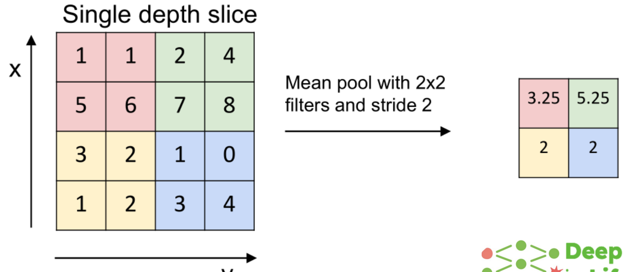
- Find the max in each block and stride by 2

Max/Mean Pooling

Source:https://www.geeksforgeeks.org/introduction-convolution-neural-network/

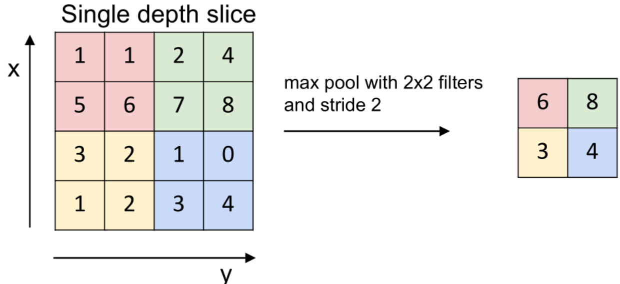


- Find the max in each block and stride by 2

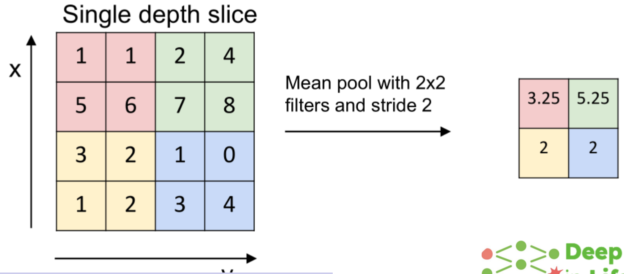


Max/Mean Pooling

Source:https://www.geeksforgeeks.org/introduction-convolution-neural-network/



- Find the max in each block and stride by 2

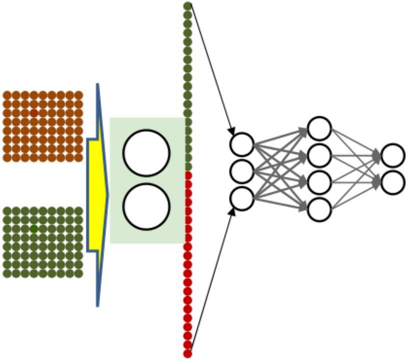


An $N \times N$ picture compressed by a $F \times F$ pooling filter extent with stride S results in an output map of size $\left\lceil \frac{(N-F)}{S} \right\rceil + 1$

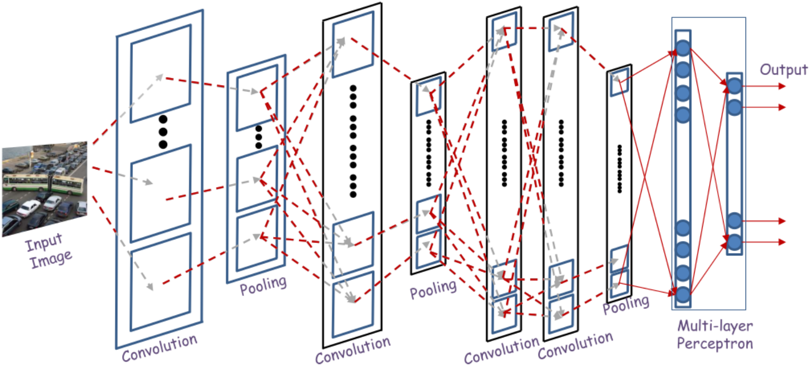
– Typically do not zero pad

Terminology

- The operation of vectorizing the output of a convolutional layer (e.g., to feed the next MLP) is called Flattening



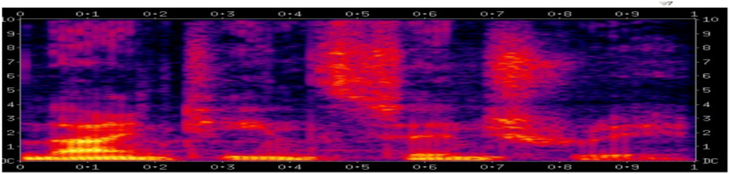
Typical CNN



Input Image
Convolution
Pooling
Convolution
Pooling
Convolution
Convolution
Pooling
Multi-layer Perceptron
Output

A problem variant

What did they say?



- Speech Recognition
  – Analyze a series of spectral vectors, determine what was said
- Note: Inputs are sequences of vectors. Output is a classification result

A problem variant 2

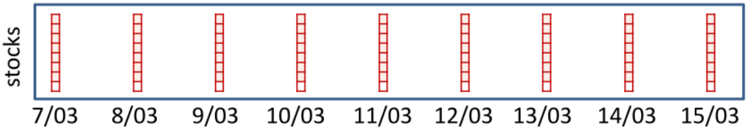
What are they talking about?

The Steelers, meanwhile, continue to struggle to make stops on defense. They've allowed, on average, 30 points a game, and have shown no signs of improving anytime soon.

- Text analysis
  - E.g. analyze document, identify topic
    - Input series of words, output classification output
  - E.g. read English, output French
    - Input series of words, output series of words

A problem variant 3

Should we invest?



Note: Inputs are sequences of vectors. Output may be scalar or vector

- Should I invest, vs. should I not invest in X?
- Decision must be taken considering how things have fared over time

Past/time dependency problems

- In all the considered problems, the output depends not just on the current input, but also on the past
  - Need to consider a sequence of inputs (scalar or vectors)

Past/time dependency problems

- In all the considered problems, the output depends not just on the current input, but also on the past
  - Need to consider a sequence of inputs (scalar or vectors)
- Must produce one or more outputs that take into account prensent and past

Long-term trends

- Longer-term trends are often a characteristic of the problem
  - E.g., weekly, monthly, or annual trends in the market

Long-term trends

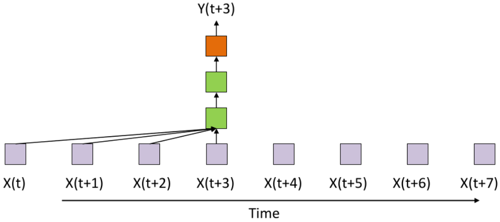
- Longer-term trends are often a characteristic of the problem
  - E.g., weekly, monthly, or annual trends in the market
- Longer historic tends however tend to affect us less than more recent events...

Long-term trends

- Longer-term trends are often a characteristic of the problem
  - E.g., weekly, monthly, or annual trends in the market
- Longer historic tends however tend to affect us less than more recent events...
- **Possibility**: The idea of "memory" could be cast into the inputs

Long-term trends

- Longer-term trends are often a characteristic of the problem
  - E.g., weekly, monthly, or annual trends in the market
- Longer historic tends however tend to affect us less than more recent events...
- **Possibility**: The idea of "memory" could be cast into the inputs
- Problem: Increasing much the input "history" can make the network too complex



A CNN looking at the last few past temporal steps

Infinite dependency

A further problem: what if we need to consider longer (potentially infinite) dependencies?

A further problem: what if we need to consider longer (potentially infinite) dependencies?

- It is possible to realize it by considering previous time output as one of the inputs:

$$Y(t) = f(X(t), Y(t-1))$$

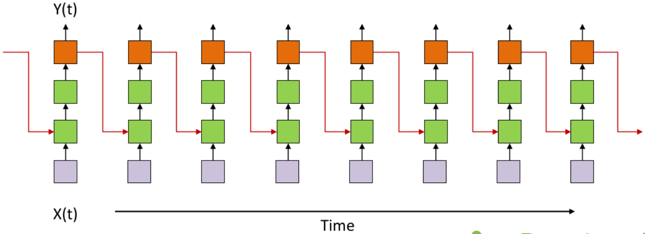
Infinite dependency

A further problem: what if we need to consider longer (potentially infinite) dependencies?

- It is possible to realize it by considering previous time output as one of the inputs:

$$Y(t) = f(X(t), Y(t-1))$$

- $Y(0)$ produces $Y(1)$, which produces $Y(2)$ ... till potentially $Y(\infty)$. If we unroll it, we have something like:



Going beyond: the state-space model

A state-space model use state variables to describe a system

A state-space model use state variables to describe a system

$$h(t) = f(X(t), h(t\text{-}1))$$
$$Y(t) = g(h(t))$$

- $h(t)$ is the state of the network at time t, $Y(t)$ the output at time t

Going beyond: the state-space model

A state-space model use state variables to describe a system

$$h(t) = f(X(t), h(t\text{-}1))$$
$$Y(t) = g(h(t))$$

- $h(t)$ is the state of the network at time t, $Y(t)$ the output at time t
- State summarizes information about the **entire past**

Going beyond: the state-space model

A state-space model use state variables to describe a system

$$h(t) = f(X(t), h(t\text{-}1))$$
$$Y(t) = g(h(t))$$

- $h(t)$ is the state of the network at time t, $Y(t)$ the output at time t
- State summarizes information about the **entire past**
- Model directly embeds the memory in the state

Going beyond: the state-space model

A state-space model use state variables to describe a system

$$h(t) = f(X(t), h(t\text{-}1))$$
$$Y(t) = g(h(t))$$

- $h(t)$ is the state of the network at time t, $Y(t)$ the output at time t
- State summarizes information about the **entire past**
- Model directly embeds the memory in the state
- Need to define the initial state $h(-1)$ (time starts at 0)

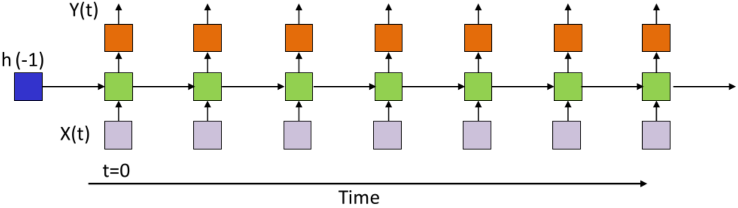
A state-space model use state variables to describe a system

$$h(t) = f(X(t), h(t\text{-}1))$$
$$Y(t) = g(h(t))$$

- $h(t)$ is the state of the network at time t, $Y(t)$ the output at time t
- State summarizes information about the **entire past**
- Model directly embeds the memory in the state
- Need to define the initial state $h(-1)$ (time starts at 0)

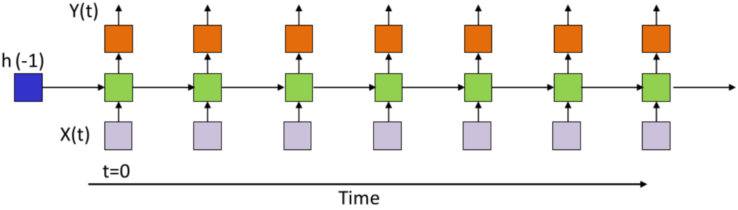
This is a fully recurrent neural network! Or simply a recurrent neural network (RNN)

Recurrent Neural Networks



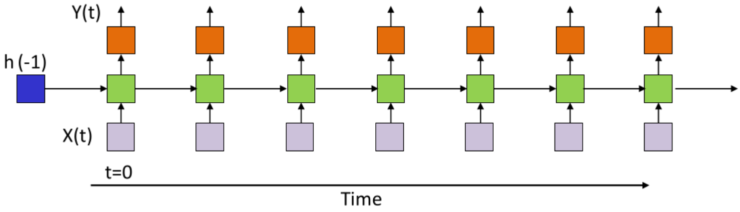
- The state (green) at any time is determined by the input at that time, and the state at the previous time

Recurrent Neural Networks



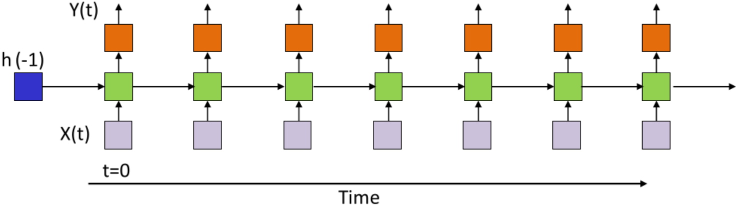
- The state (green) at any time is determined by the input at that time, and the state at the previous time
- An input at $t = 0$ can affect outputs forever!!

Recurrent Neural Networks



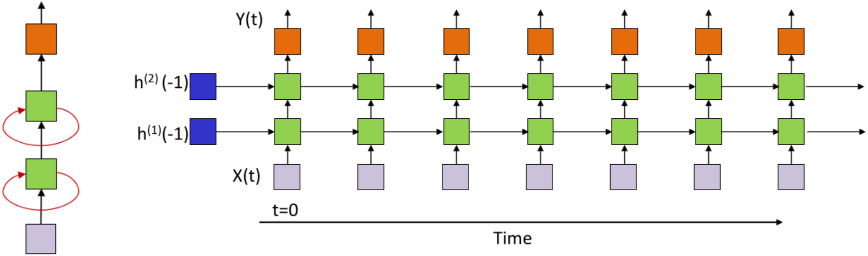
- The state (green) at any time is determined by the input at that time, and the state at the previous time
- An input at $t = 0$ can affect outputs forever!!
- The state can be arbitrarily complex (e.g., a vector/matrix)

Recurrent Neural Networks



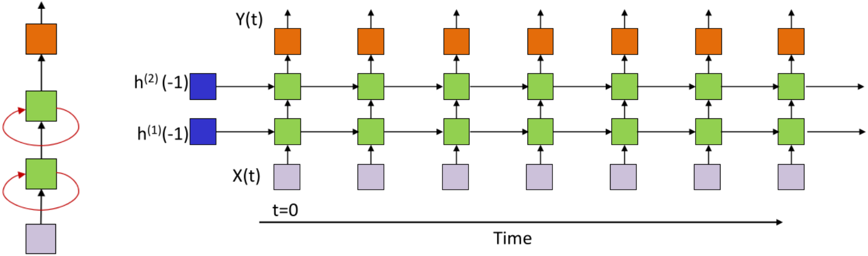
- The state (green) at any time is determined by the input at that time, and the state at the previous time
- An input at $t = 0$ can affect outputs forever!!
- The state can be arbitrarily complex (e.g., a vector/matrix)
- We can add multiple recurrent layers

Rolled view



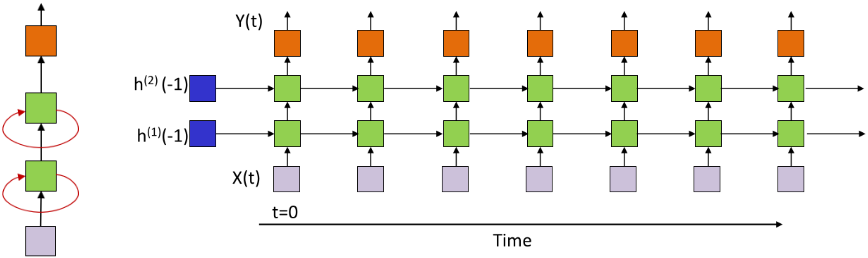
- Loops imply recurrence/memory

Rolled view



- Loops imply recurrence/memory
- The "unrolled" computation is just a giant shared-parameter neural network

Rolled view



- Loops imply recurrence/memory
- The "unrolled" computation is just a giant shared-parameter neural network
- All columns are identical and share parameters

Training RNNs

- Similarly to CNN, network parameters can be trained via gradient-descent (or its variants) using shared-parameter gradient descent rules

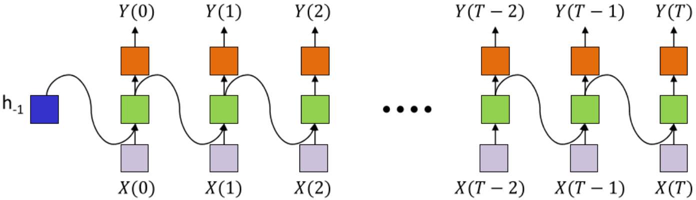
Training RNNs

- Similarly to CNN, network parameters can be trained via gradient-descent (or its variants) using shared-parameter gradient descent rules
- But we need to account now that the inputs (and the outputs) are through time

Training RNNs

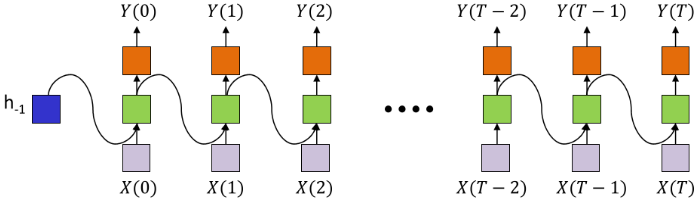
- Similarly to CNN, network parameters can be trained via gradient-descent (or its variants) using shared-parameter gradient descent rules
- But we need to account now that the inputs (and the outputs) are through time
- Back Propagation Through Time (BPTT) !

Back Propagation Through Time - main cases



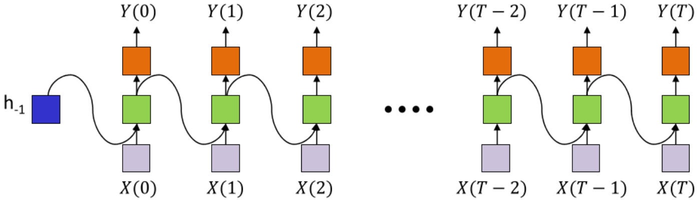
- In general, we have a sequence of $T + 1$ inputs $\boldsymbol{X}(0), \dots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \dots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors

Back Propagation Through Time - main cases



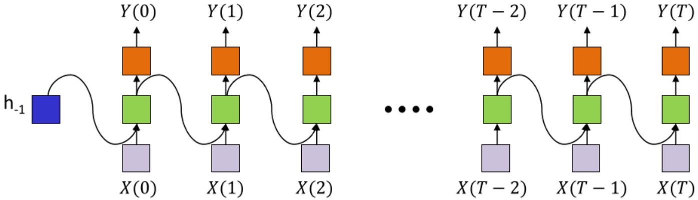
- In general, we have a sequence of $T+1$ inputs $\boldsymbol{X}(0), \ldots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \ldots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors
- **Case 1**: We are interested to all the intermediate outputs (e.g., predicting the trend of an index through the time)

Back Propagation Through Time - main cases



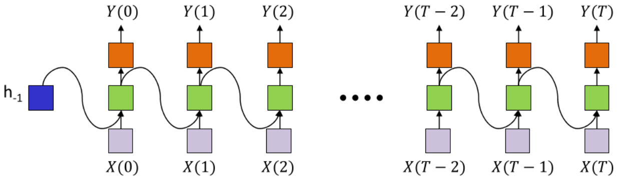
- In general, we have a sequence of $T+1$ inputs $\boldsymbol{X}(0), \ldots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \ldots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors
- **Case 1**: We are interested to all the intermediate outputs (e.g., predicting the trend of an index through the time)
  - We will compute the error between the sequence of desired outputs over time $\boldsymbol{d}(0), \ldots \boldsymbol{d}(T)$ and the corresponding predictions $\boldsymbol{Y}(0), \ldots, \boldsymbol{Y}(T)$

Back Propagation Through Time - main cases

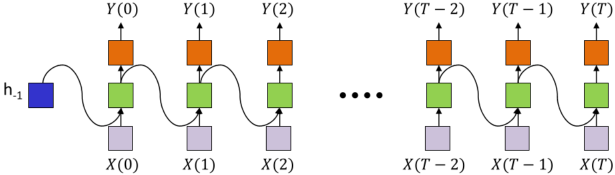


- In general, we have a sequence of $T+1$ inputs $\boldsymbol{X}(0), \dots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \dots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors
- **Case 1**: We are interested to all the intermediate outputs (e.g., predicting the trend of an index through the time)
  - We will compute the error between the sequence of desired outputs over time $\boldsymbol{d}(0), \dots \boldsymbol{d}(T)$ and the corresponding predictions $\boldsymbol{Y}(0), \dots, \boldsymbol{Y}(T)$
  - In principle, this is not just the sum of the errors at individual times

Back Propagation Through Time - main cases



- In general, we have a sequence of $T + 1$ inputs $\boldsymbol{X}(0), \ldots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \ldots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors
- **Case 2**: E.g., we are interested just to the last output $\boldsymbol{Y}(T)$ (e.g., for a final response *success*/*failure*)



- In general, we have a sequence of $T + 1$ inputs $\boldsymbol{X}(0), \ldots, \boldsymbol{X}(T)$
  - and their corresponding outputs $\boldsymbol{Y}(0), \ldots, \boldsymbol{Y}(T)$
  - in a general fashion, they can be vectors
- **Case 2**: E.g., we are interested just to the last output $\boldsymbol{Y}(T)$ (e.g., for a final response *success*/*failure*)
  - We will compute the error between the last desired output $\boldsymbol{d}(T)$ and the corresponding prediction $\boldsymbol{Y}(T)$

The long-term dependency problem

- Even for RNN we have the vanishing/exploding gradient problem
- Mainly if we need to store long temporal dependencies

PATTERN1 [..............................] PATTERN 2

Jane had a quick lunch in the bistro. Then *she*..

Any other pattern of any length can happen between pattern 1 and pattern 2

- RNN will "forget" pattern 1 if intermediate stuff is too long
- "Jane" → the next pronoun referring to her will be "she"

Must know to "remember" for extended periods of time and "recall" when necessary

- Need an alternate way to "remember" stuff

Possible solutions

- Long Short-Term Memory (LSTM) [Hochreiter et al. (1997)]
- Gated Recurrent Unit (GRU)[Cho et al. (2014)]
- More details in the lab session! Next Thursday for UNIMI

CREDITS

Content is inspired and taken from:

- Introduction to Deep Learning,
  https://deeplearning.cs.cmu.edu/F22/index.html
- Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press, 2016. Chapter 9.
  http://www.deeplearningbook.org
- Introduction to Deep Learning (I2DL)
  https://www.3dunderstanding.org/i2dl-w22/, Lecture 11.
- Introduction to Deep Learning,
  https://deeplearning.cs.cmu.edu/F22/index.html, lectures 13, 14.
- Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press, 2016. Chapter 12.
  http://www.deeplearningbook.org
- http://colah.github.io/posts/2015-08-Understanding-LSTMs

REFERENCES

Hochreiter, S., and Schmidhuber, Jurgen. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

Cho, K., van Merrienboer, B., et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". arXiv:1406.1078.