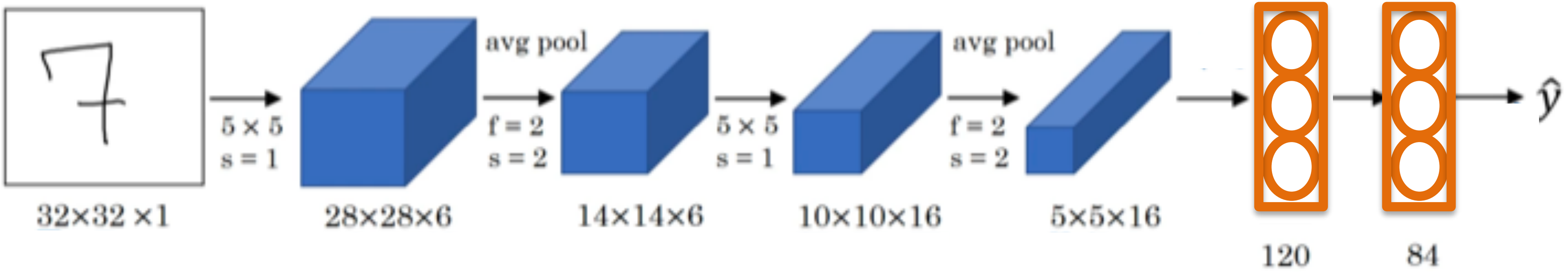
Lecture 10 recap

LeNet

- Digit recognition: 10 classes

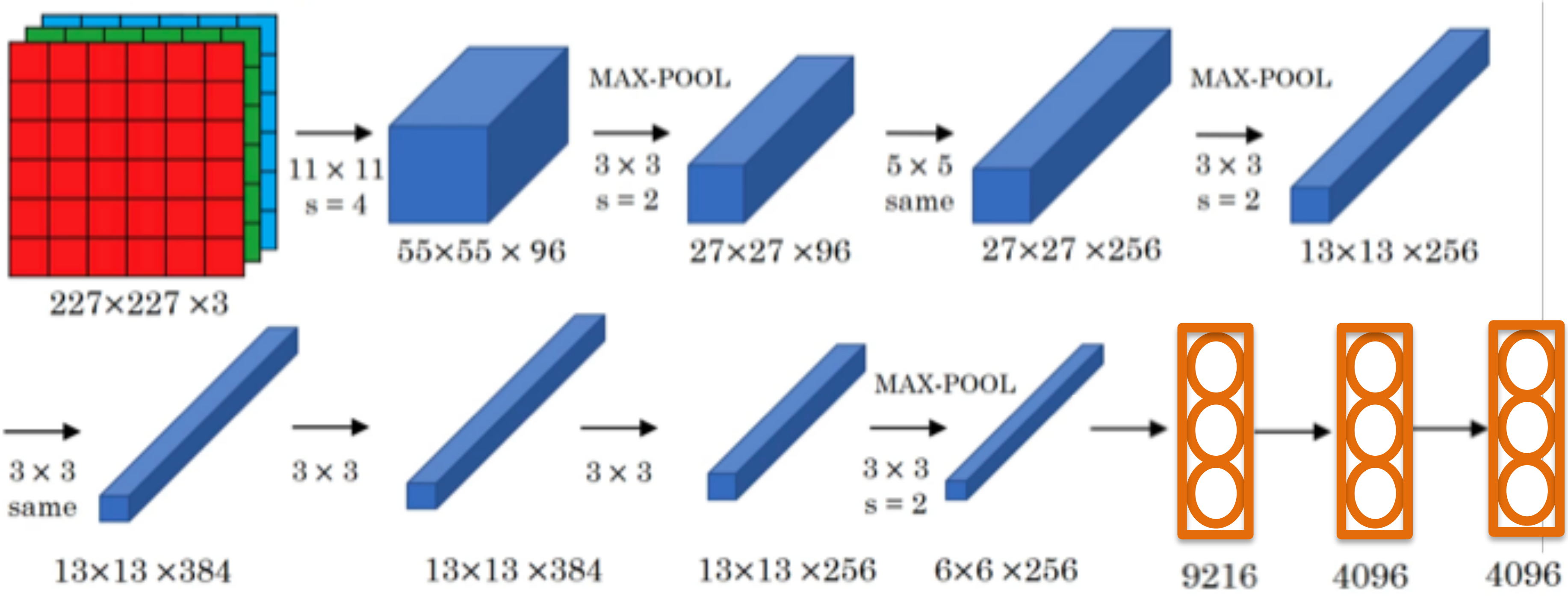
60k parameters



- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ▼ Number of filters ▲

AlexNet

[Krizhevsky et al. 2012]



- Softmax for 1000 classes

VGGNet

[Simonyan and Zisserman 2014]

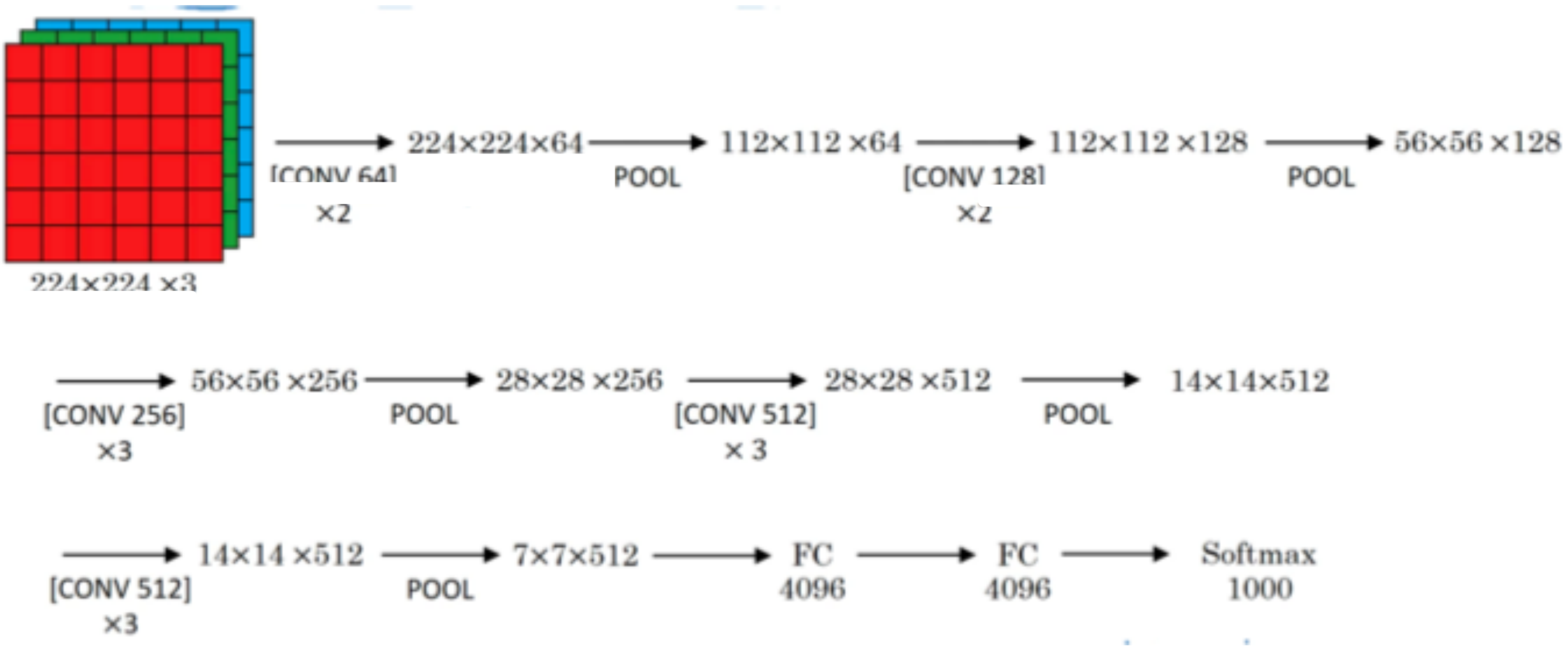
- Striving for simplicity

- CONV = 3x3 filters with stride 1, same convolutions

- MAXPOOL = 2x2 filters with stride 2

VGGNet

Conv=3x3,s=1,same
Maxpool=2x2,s=2



VGGNet

- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, height ▼ Number of filters ▲

- Called VGG-16: 16 layers that have weights

$\boxed{\text{138M parameters}}$

- Large but simplicity makes it appealing

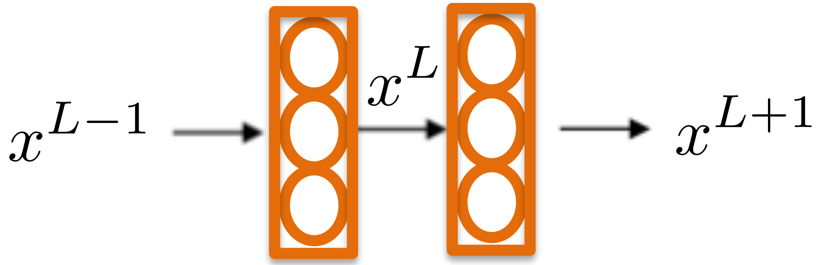
The problem of depth

- As we add more and more layers, training becomes harder
- Vanishing and exploding gradients

- How can we train very deep nets?

Residual block

- Two layers



Input $\rightarrow$ $W^L x^{L-1} + b^L$ $\rightarrow$ $x^L = f(W^L x^{L-1} + b^L)$ $\rightarrow$

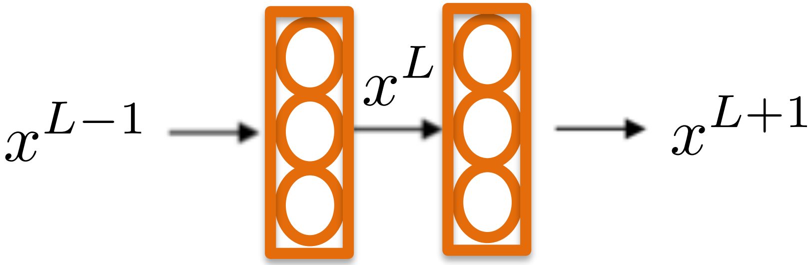
Linear

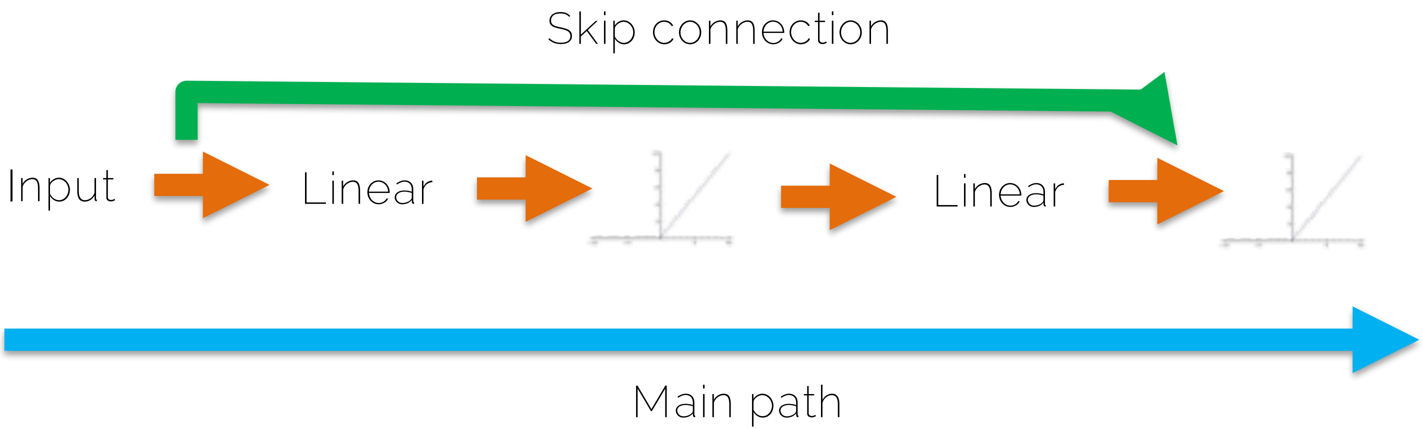
Non-linearity

$\rightarrow$ $x^{L+1} = f(W^{L+1} x^L + b^{L+1})$

Residual block

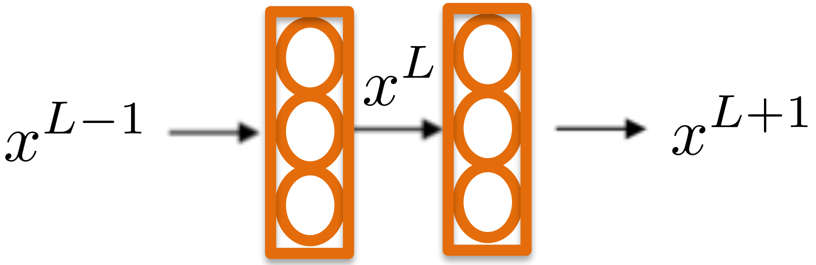
- Two layers



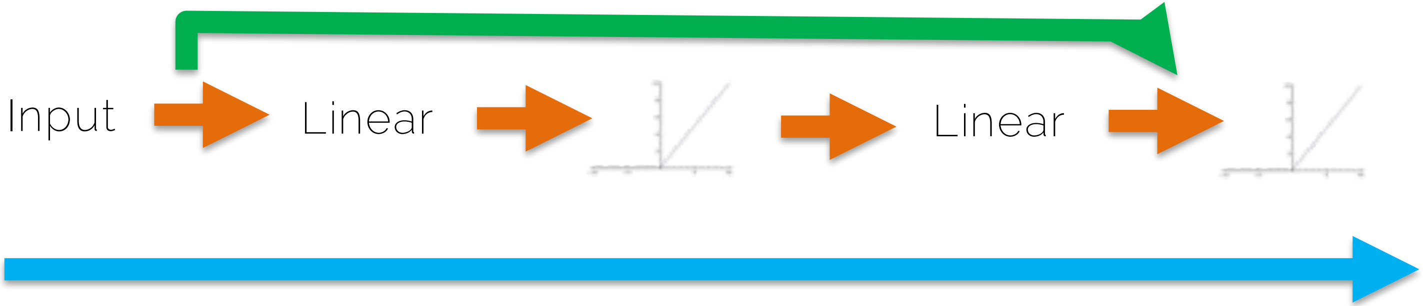


Residual block

- Two layers



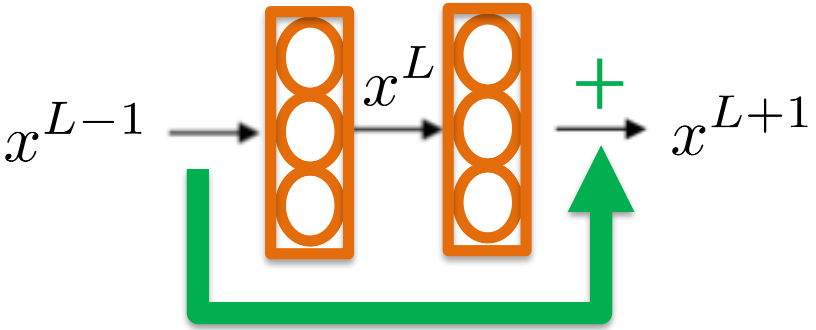
$$x^{L+1} = f(W^{L+1}x^{L} + b^{L+1} + x^{L-1})$$



$$x^{L+1} = f(W^{L+1}x^{L} + b^{L+1})$$

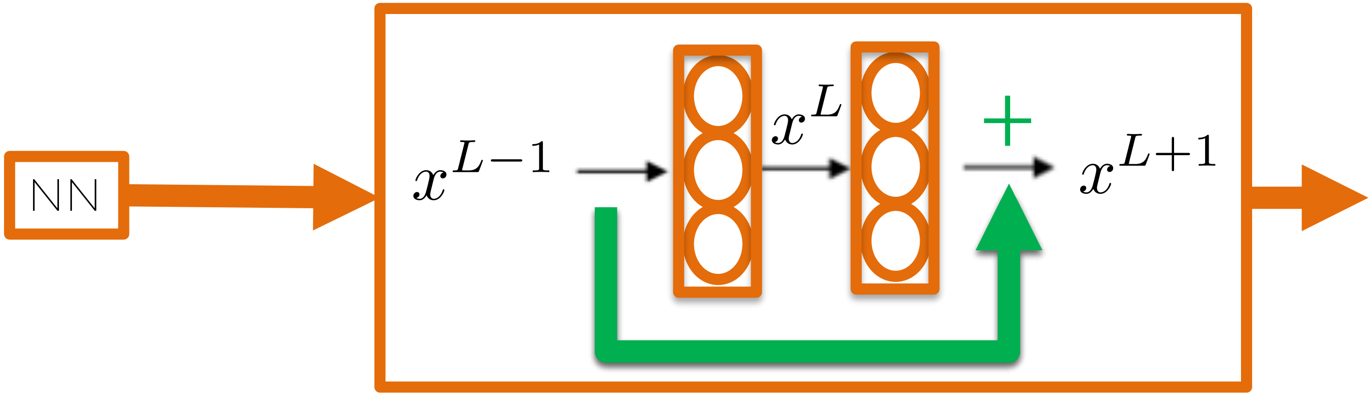
Residual block

- Two layers



- Usually use a same convolution since we need same dimensions
- Otherwise we need to convert the dimensions with a matrix of learned weights or zero padding

Why do ResNets work?



- The identity is easy for the residual block to learn
- Guaranteed it will not hurt performance, can only improve

1x1 convolution

Image 5x5

-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 1x1

2

What is the output size?

1x1 convolution

Image 5x5

-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

-10				

Kernel 1x1

2

$$-5 * 2 = -10$$

1x1 convolution

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	$\boxed{-1}$

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

Kernel 1x1

2

$$-1 * 2 = -2$$

1x1 convolution

Image 5x5

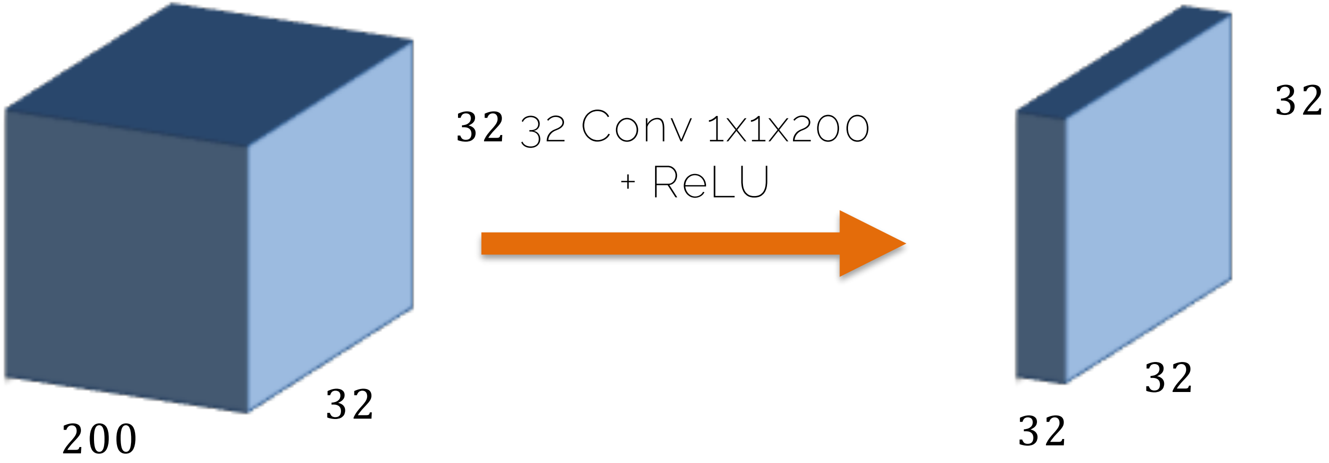
-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

- For 1 kernel or filter, it keeps the dimensions and just scales the input with a number

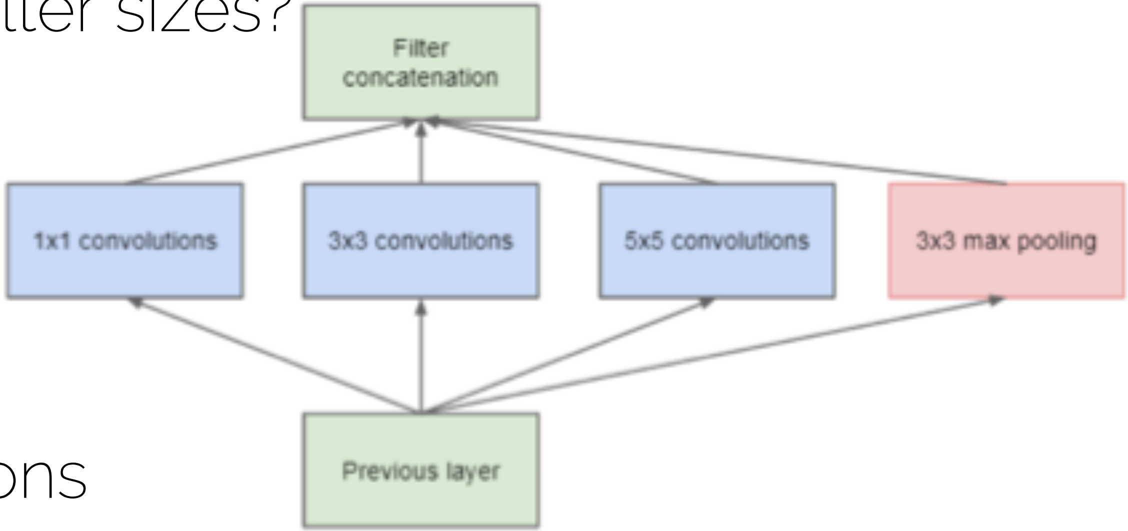
Using 1x1 convolutions

- Use it to shrink the number of channels
- Further adds a non-linearity → one can learn more complex functions

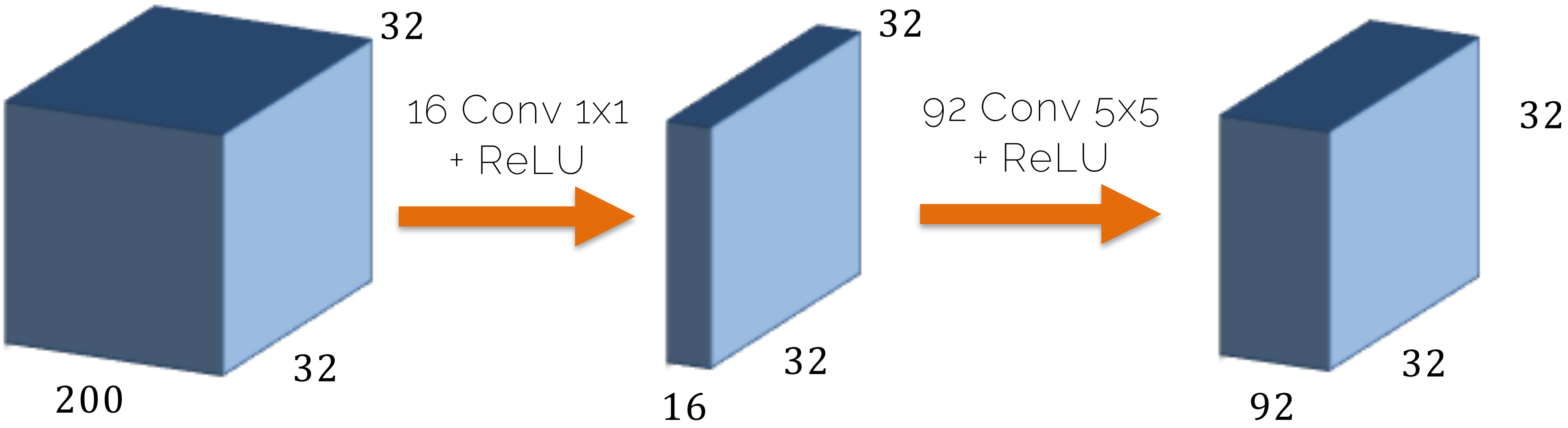


Inception layer

- Tired of choosing filter sizes?
- Use them all!
- All same convolutions
- 3x3 max pooling is with stride 1



Inception layer: computational cost



Multiplications: 1x1x200x32x32x16 5x5x16x32x32x92 ~ 40 million

Reduction of multiplications by 1/10

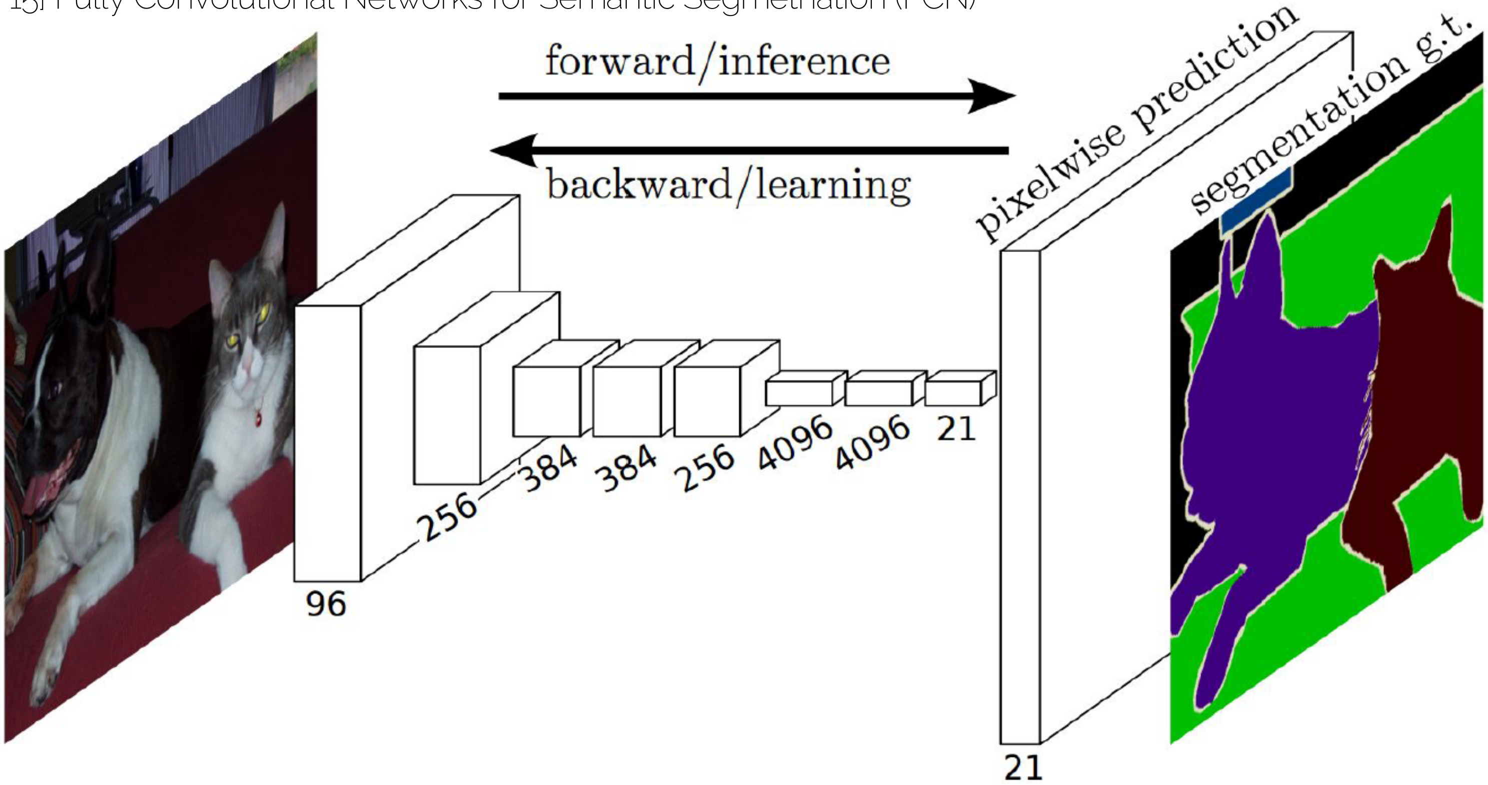
Inception layer

(a) Inception module, naïve version

(b) Inception module with dimensionality reduction

Semantic Segmentation (FCN)

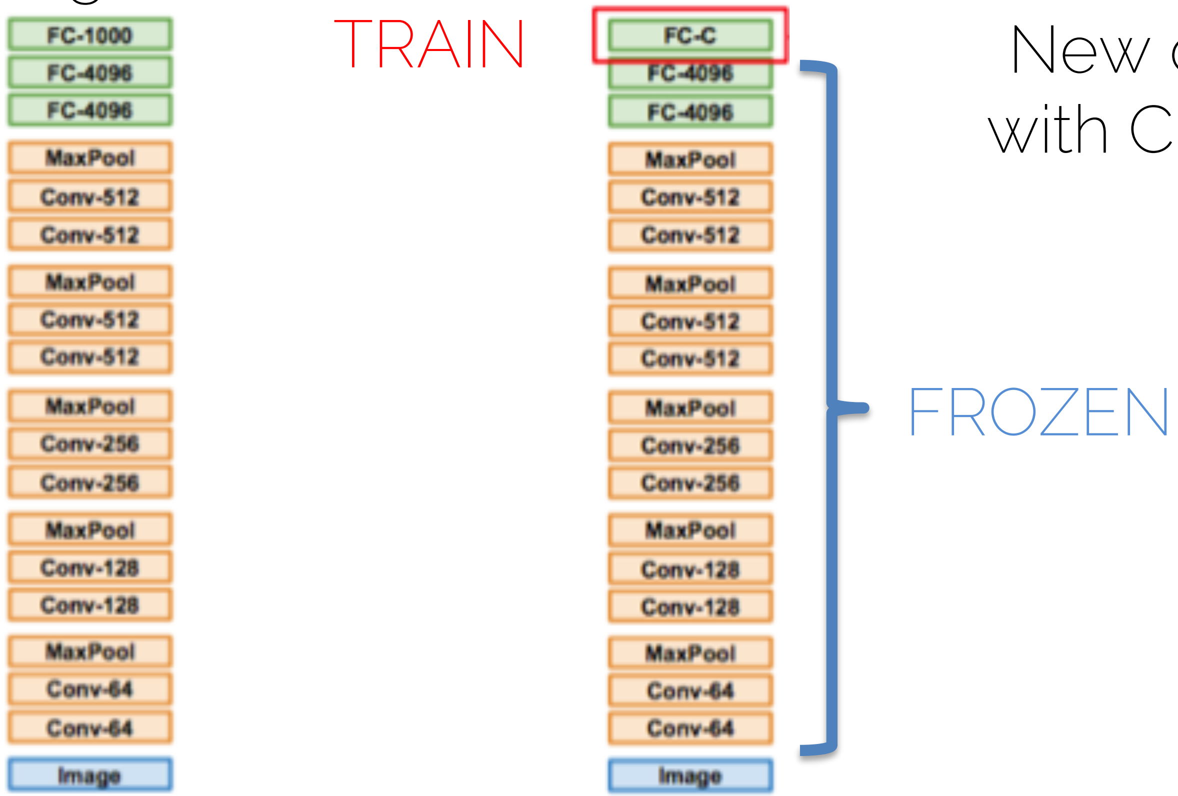
[Long et al. 15] Fully Convolutional Networks for Semantic Segmetnation (FCN)



Transfer learning

Trained on ImageNet

New dataset with C classes



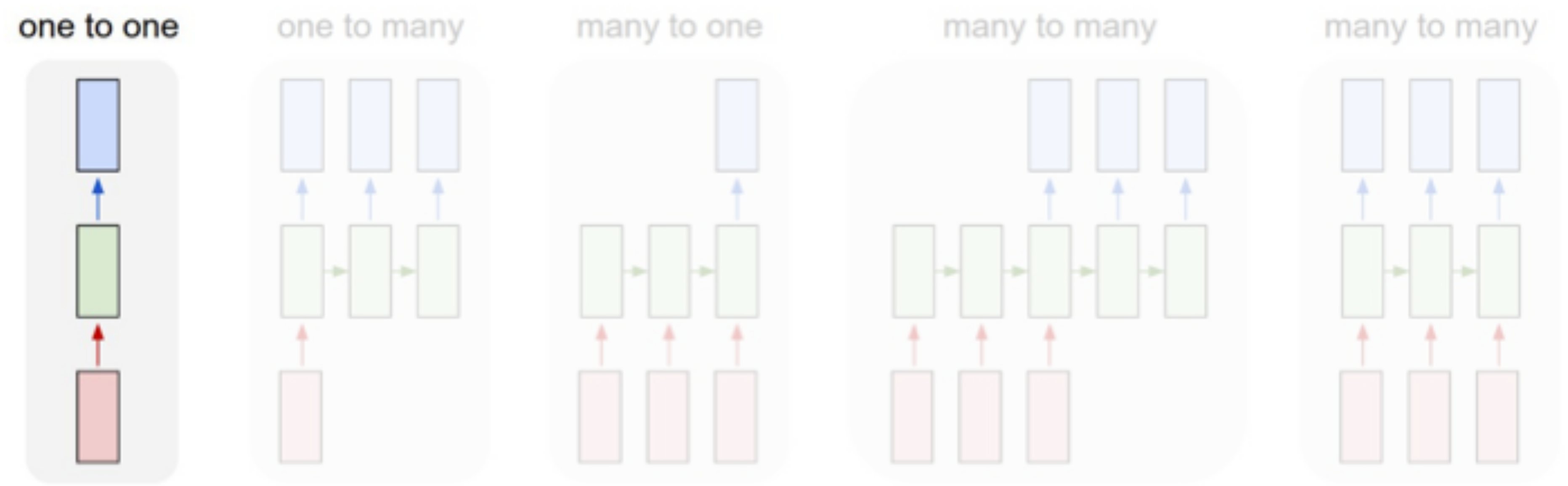
Donahue 2014, Razavian 2014

Now you are:

- Ready to perform image classification on any dataset
- Ready to design your own architecture
- Ready to deal with other problems such as semantic segmentation (Fully Convolutional Network)

Recurrent Neural Networks

RNNs are flexible



Classic Neural Networks for Image Classification

RNNs are flexible

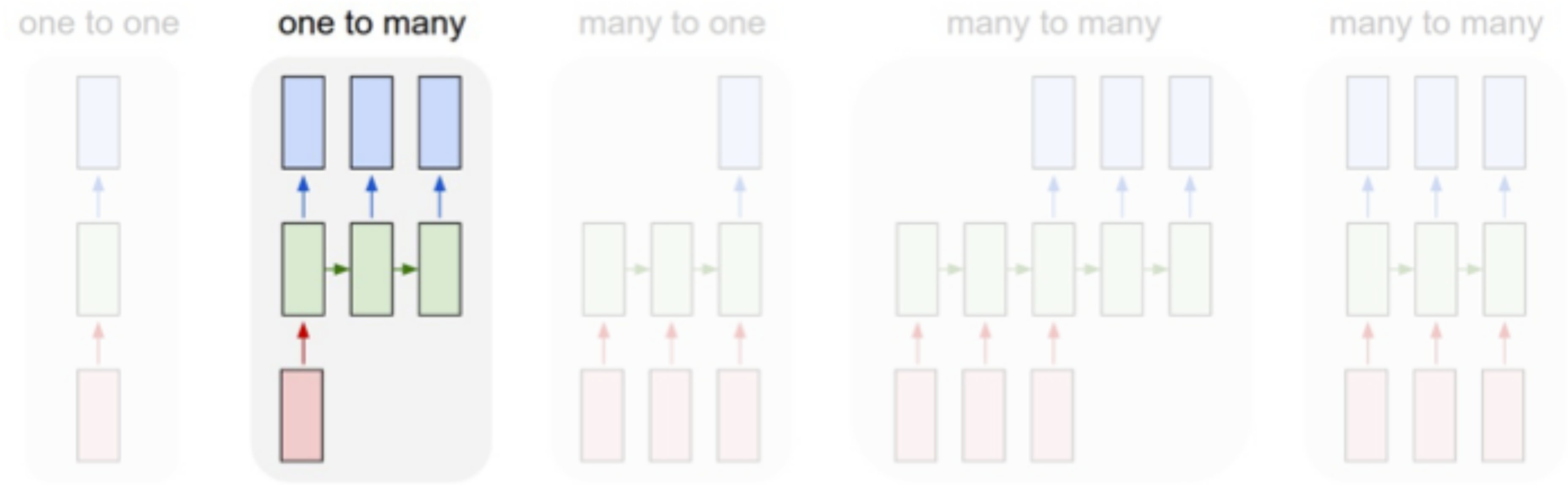
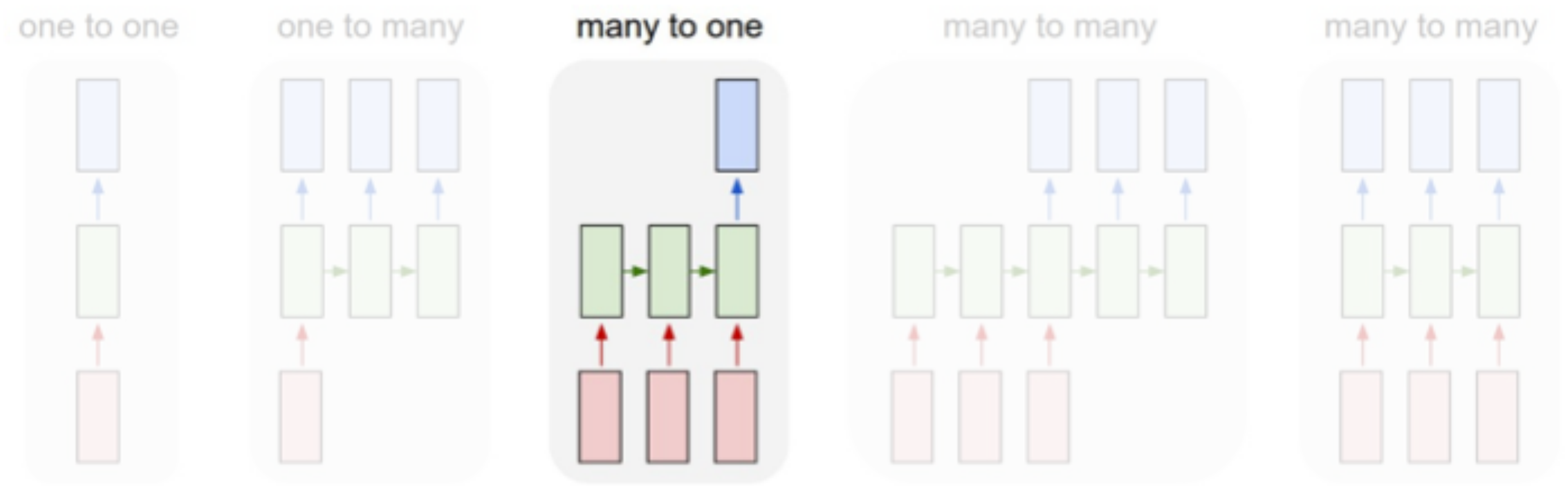


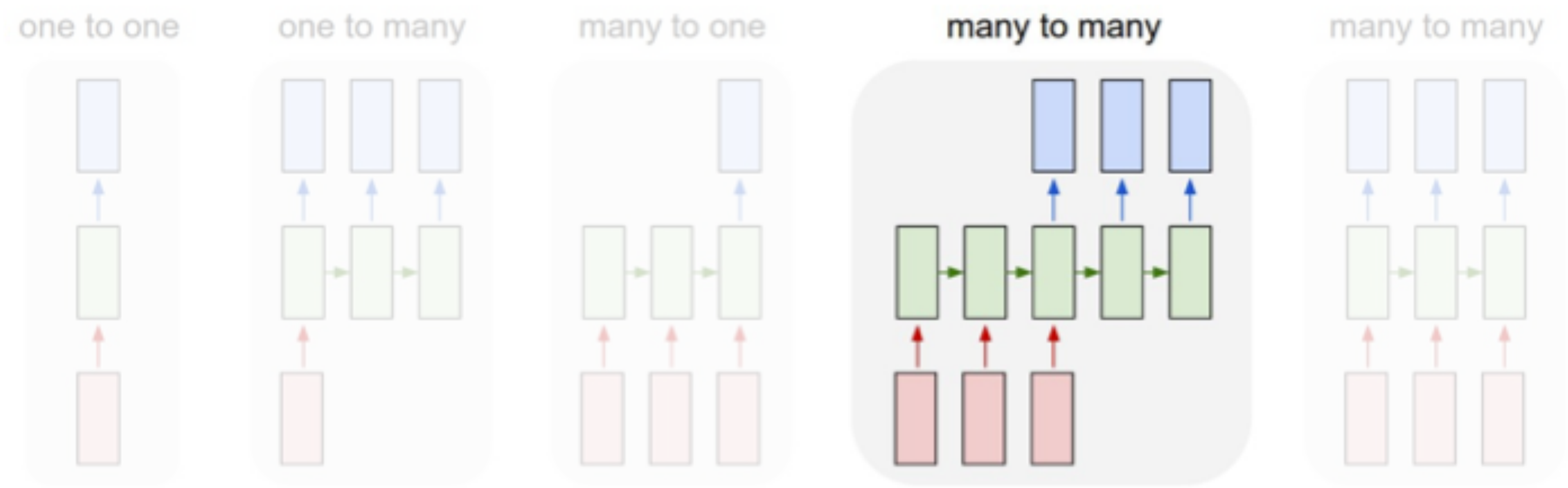
Image captioning

RNNs are flexible



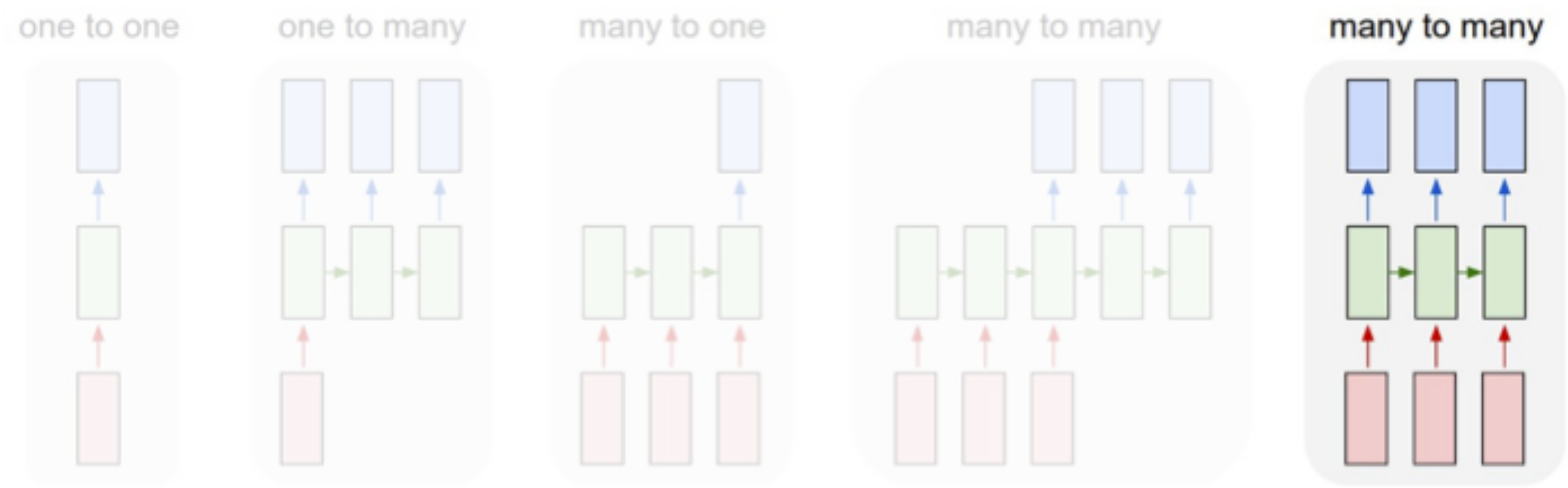
Language recognition

RNNs are flexible



Machine translation

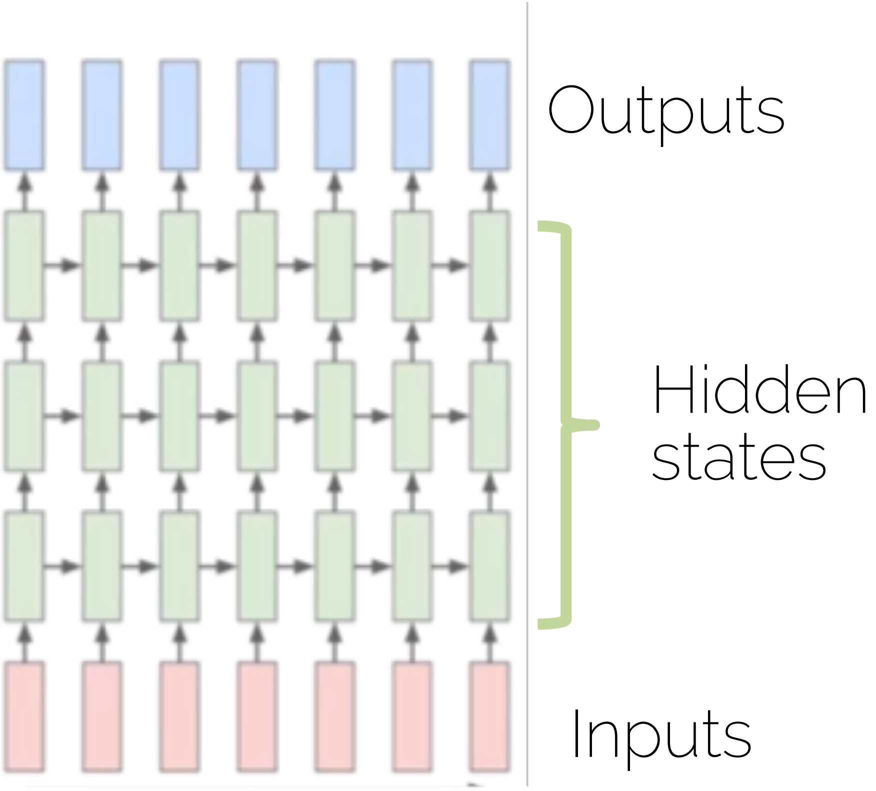
RNNs are flexible



Event classification

Basic structure of a RNN

- Multi-layer RNN

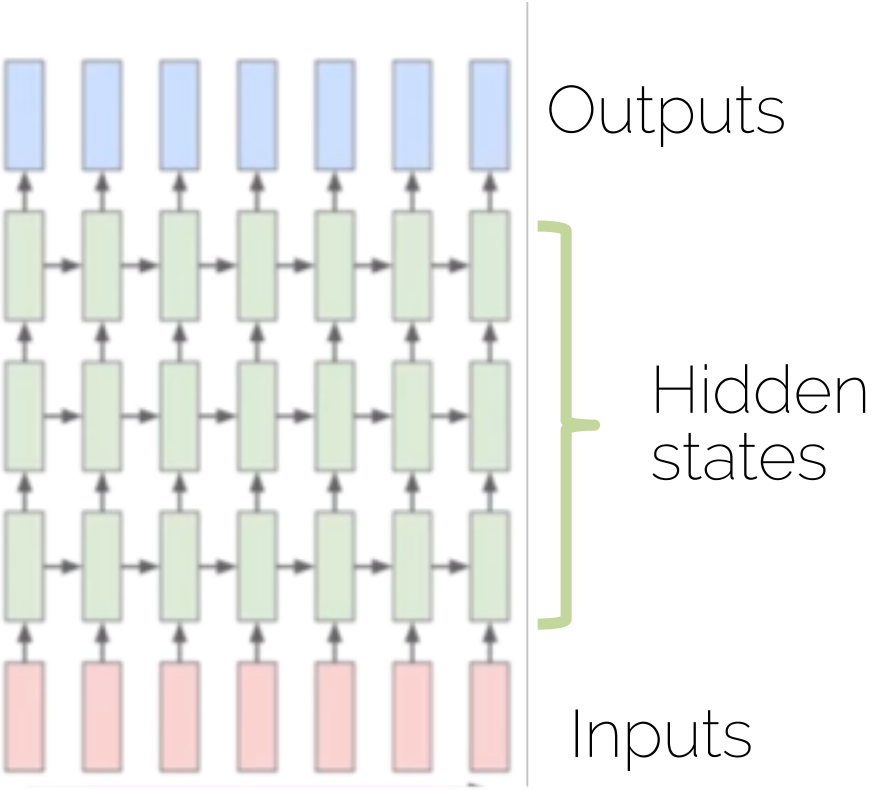


Basic structure of a RNN

- Multi-layer RNN

The hidden state will have its own internal dynamics

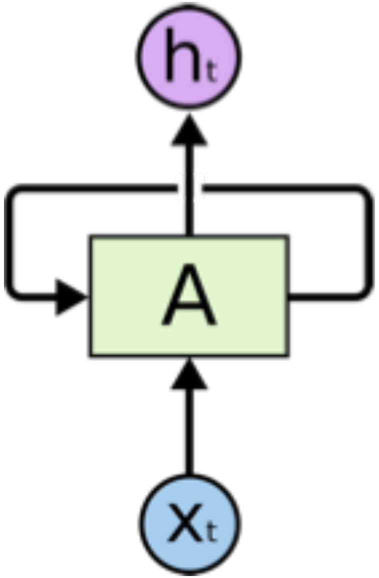
More expressive model!



Basic structure of a RNN

- We want to have notion of “time” or “sequence”

Hidden state



$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

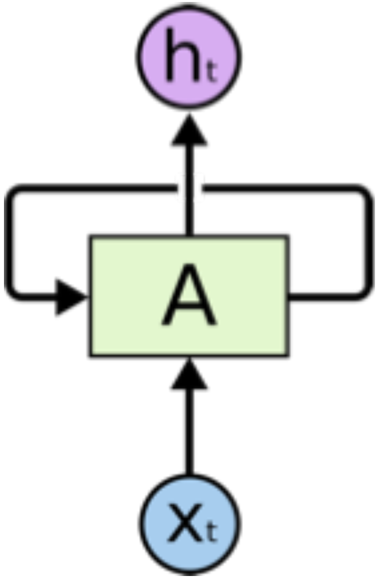
Previous hidden state

input

Basic structure of a RNN

- We want to have notion of "time" or "sequence"

Hidden state



$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

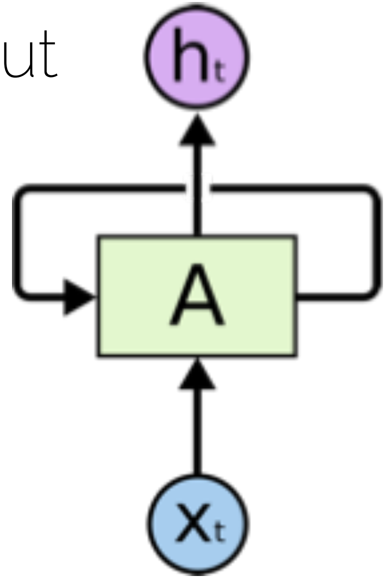
Parameters to be learned

Basic structure of a RNN

- We want to have notion of "time" or "sequence"

Output

Hidden
state



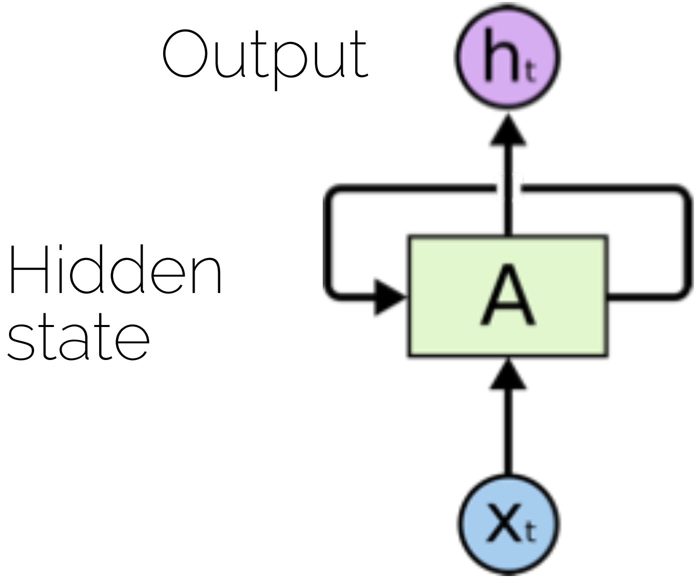
$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

$$\mathbf{h}_t = \boldsymbol{\theta}_h \mathbf{A}_t$$

Note: non-linearities
ignored for now

Basic structure of a RNN

- We want to have notion of “time” or “sequence”



$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

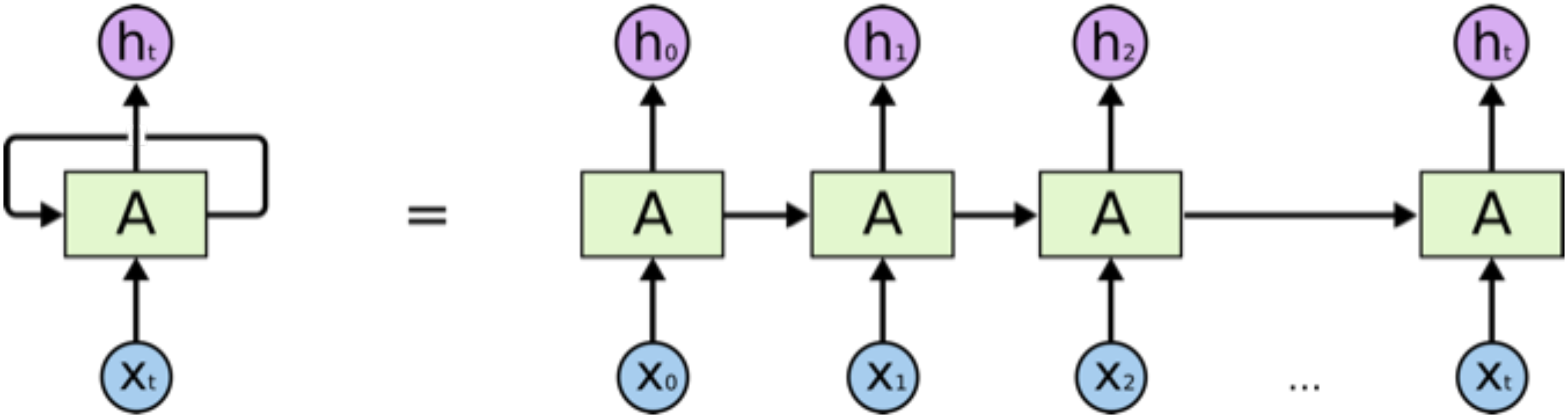
$$\mathbf{h}_t = \boldsymbol{\theta}_h \mathbf{A}_t$$

Same parameters for each time step = generalization!

Basic structure of a RNN

- Unrolling RNNs

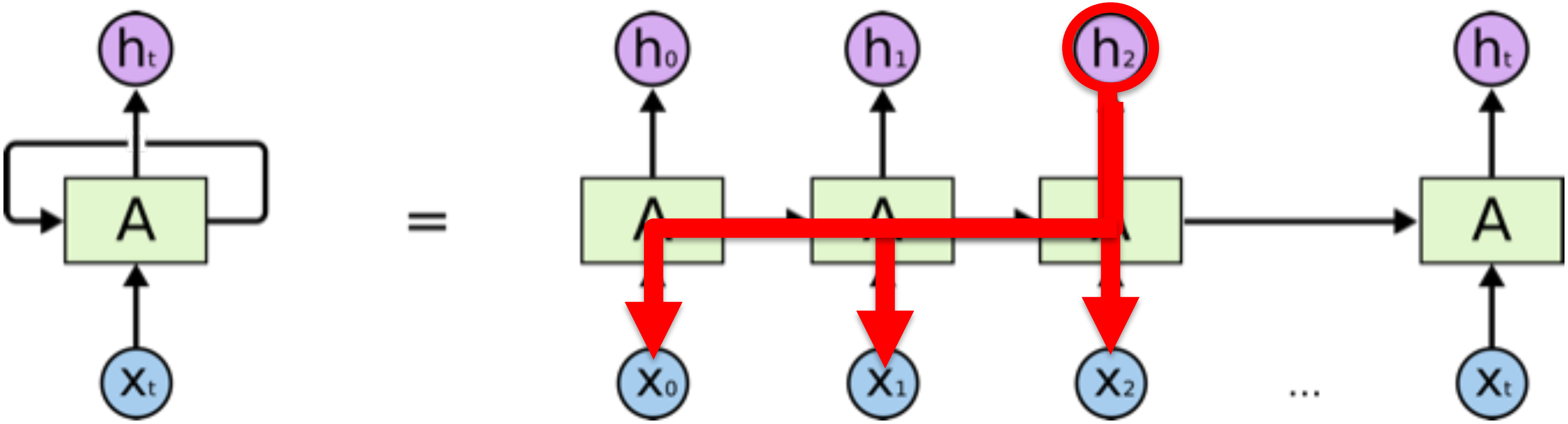
Hidden state is the same



[Christopher Olah] Understanding LSTMs

Basic structure of a RNN

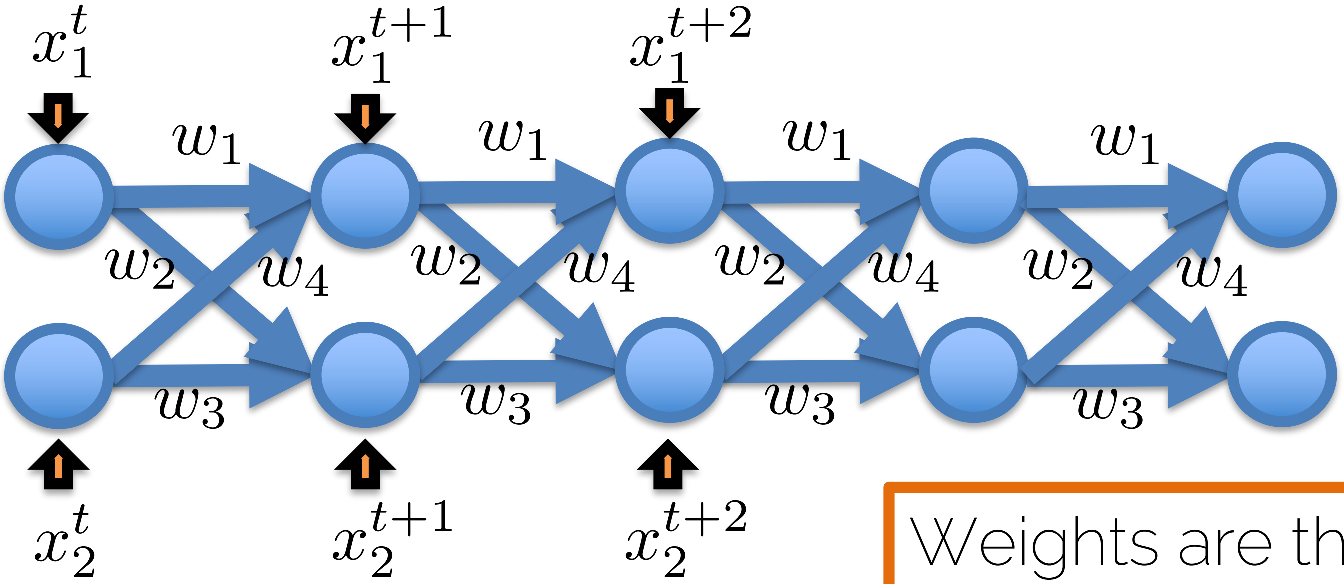
- Unrolling RNNs



[Christopher Olah] Understanding LSTMs

Basic structure of a RNN

- Unrolling RNNs as feedforward nets

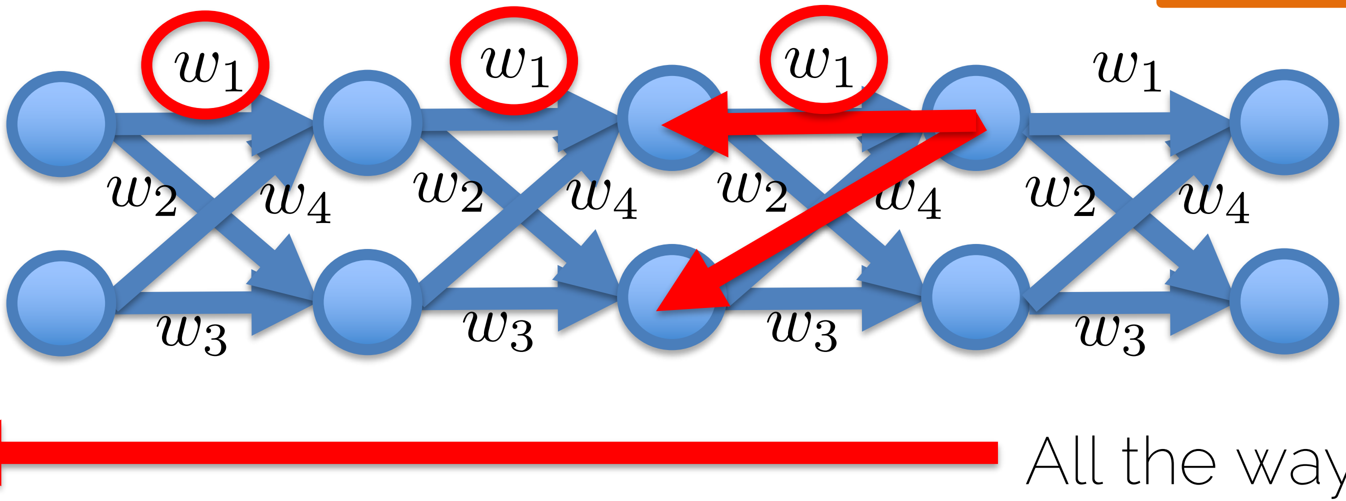


Weights are the same!

Backprop through a RNN

- Unrolling RNNs as feedforward nets

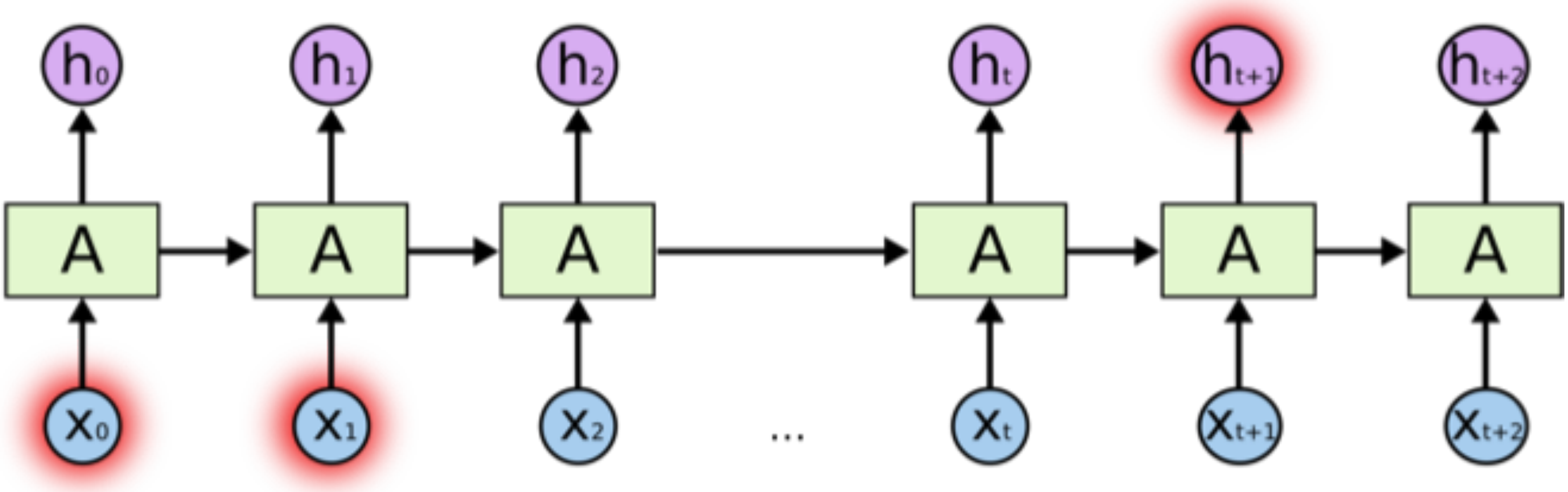
Chain rule



All the way to t=0

Add the derivatives at different times for each weight

Long-term dependencies



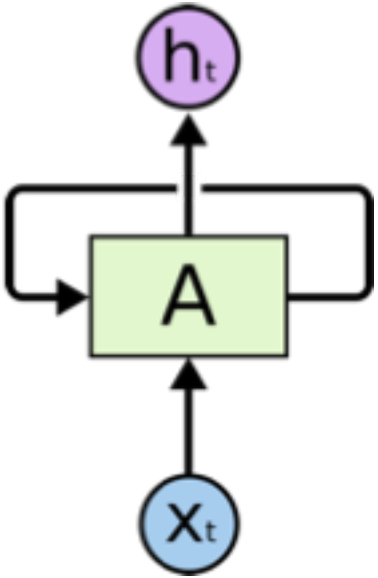
I moved to Germany … so I speak German fluently

Long-term dependencies

- Simple recurrence $\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$

- Let us forget the input $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$

Same weights are multiplied over and over again



Long-term dependencies

- Simple recurrence $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$

What happens to small weights?

Vanishing gradient

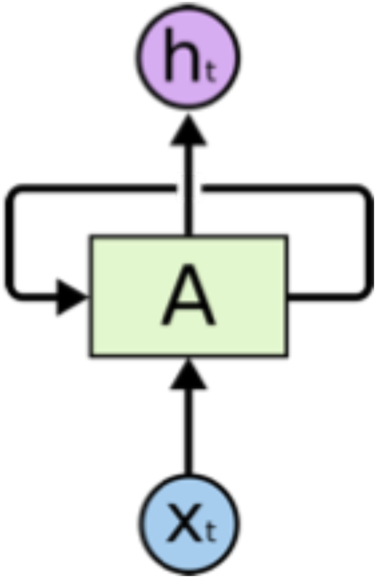
What happens to large weights?

Exploding gradient

Long-term dependencies

- Simple recurrence $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$

- If $\boldsymbol{\theta}$ admits eigendecomposition



$$\boldsymbol{\theta} = \mathbf{Q} \Lambda \mathbf{Q}^\intercal$$

Matrix of eigenvectors

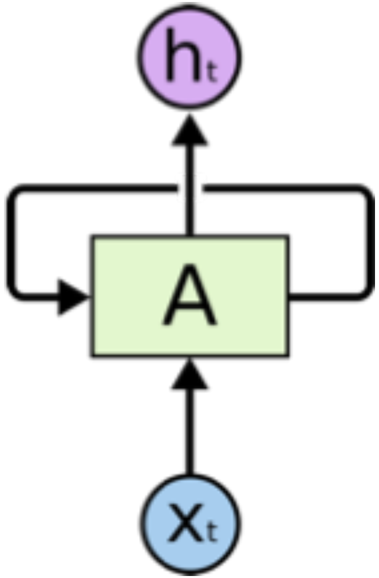
Diagonal of this matrix are the eigenvalues

Long-term dependencies

- Simple recurrence $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$

- If $\boldsymbol{\theta}$ admits eigendecomposition

$$\boldsymbol{\theta} = \mathbf{Q} \Lambda \mathbf{Q}^\intercal$$



- Orthogonal $\boldsymbol{\theta}$ allows us to simplify the recurrence

$$\mathbf{A}_t = \mathbf{Q} \Lambda^t \mathbf{Q}^\intercal \mathbf{A}_0$$

Long-term dependencies

- Simple recurrence $\mathbf{A}_t = \mathbf{Q}\Lambda^t \mathbf{Q}^\intercal \mathbf{A}_0$

What happens to eigenvalues with magnitude less than one?

Vanishing gradient

What happens to eigenvalues with magnitude larger than one?

Exploding gradient

Gradient clipping

Long-term dependencies

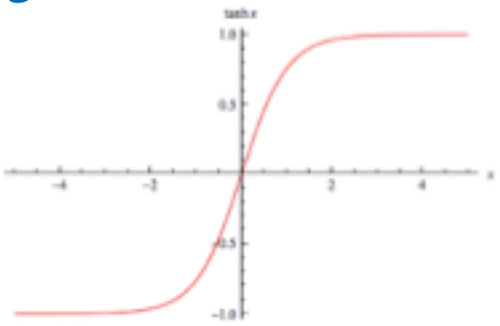
- Simple recurrence $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$

Let us just make a matrix with eigenvalues = 1

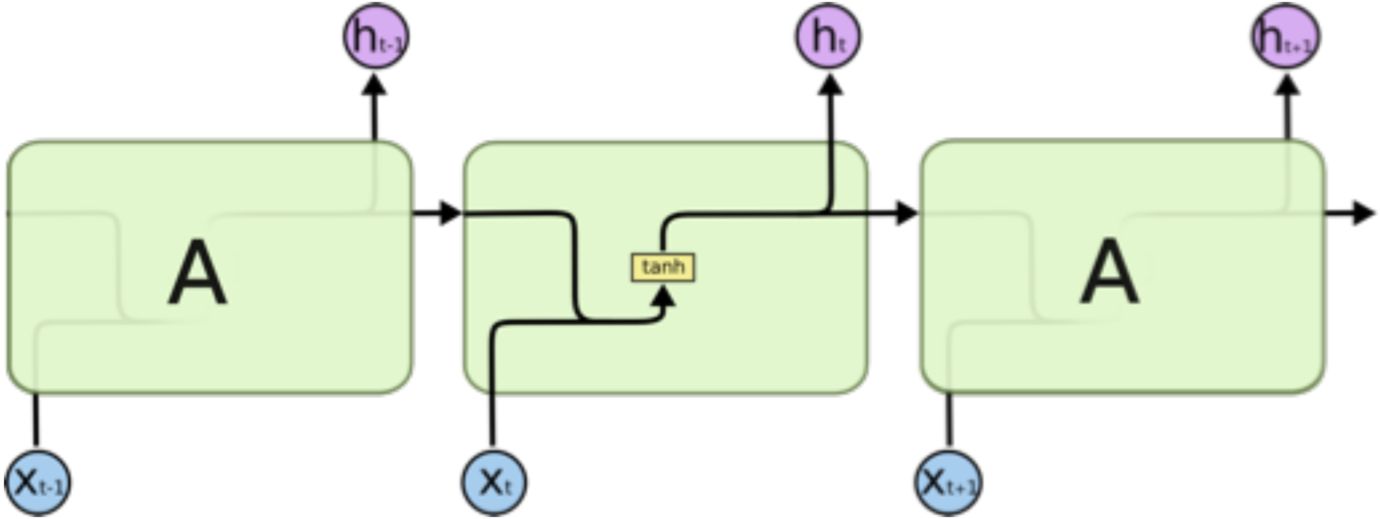
Allow the **cell** to maintain its "state"

Vanishing gradient

- 1. From the weights $\mathbf{A}_t = \boldsymbol{\theta}^t \mathbf{A}_0$



- 2. From the activation functions (tanh)

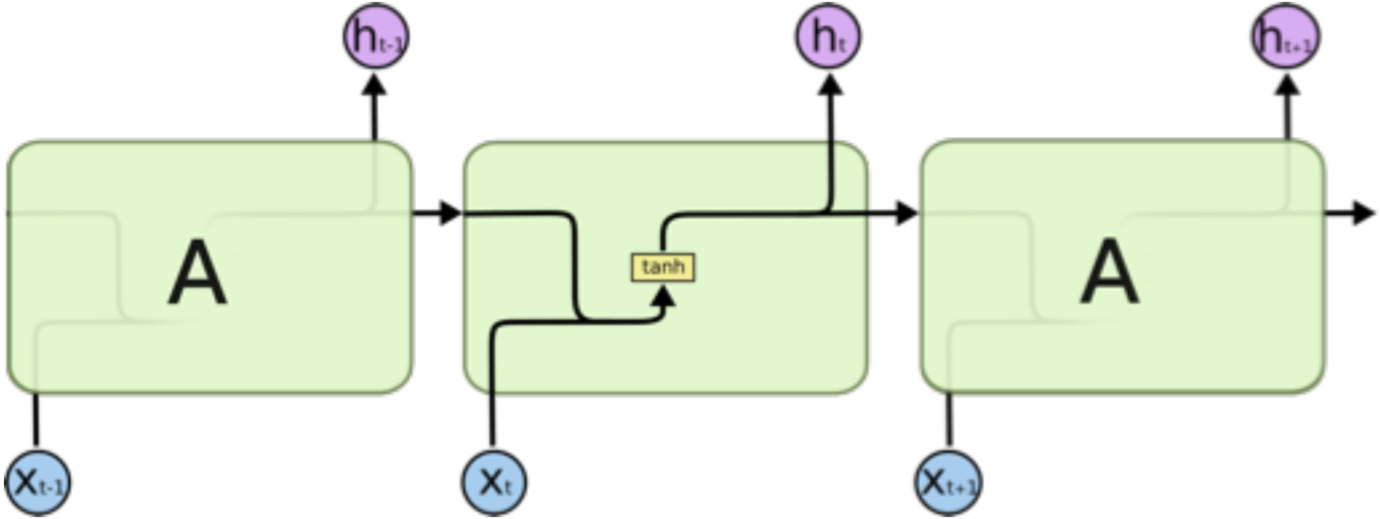


Vanishing gradient

- 1. From the weights $\mathbf{A}_t = \cancel{\boldsymbol{\theta}}^t \mathbf{A}_0$

1

- 2. From the activation functions (~~tanh~~)

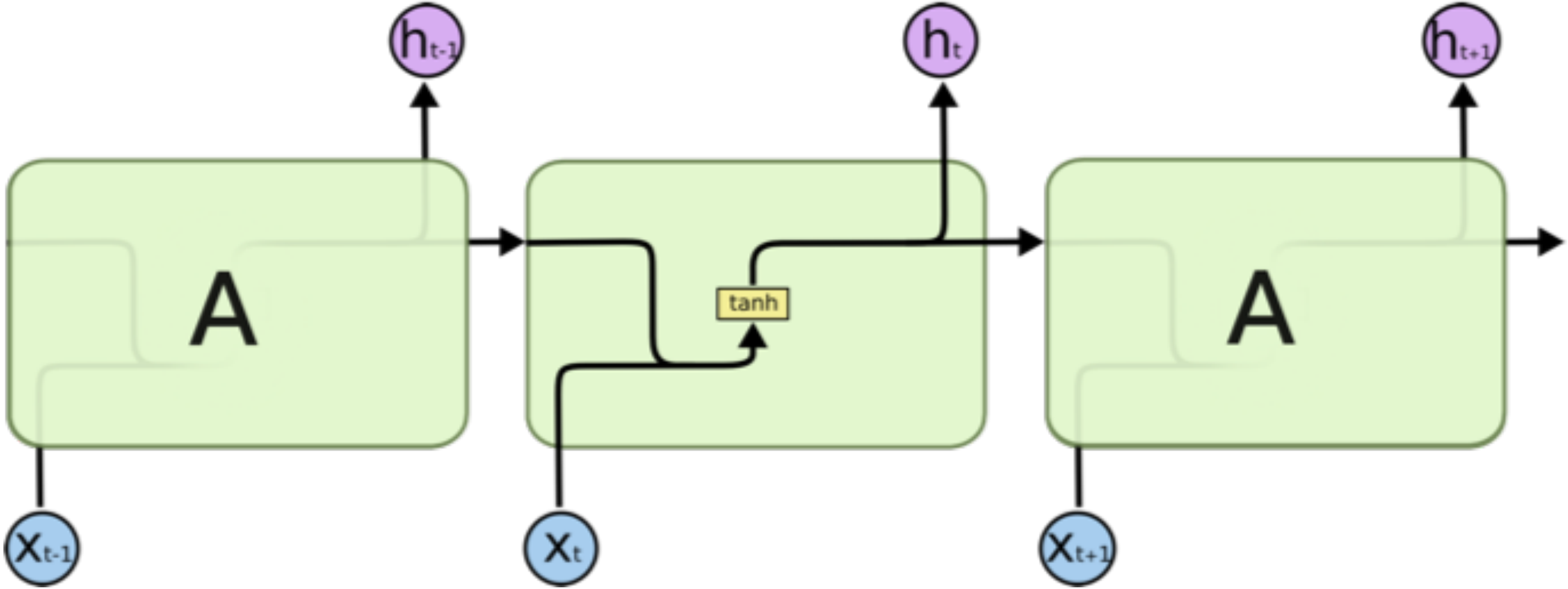


Long Short Term Memory

Hochreiter and Schmidhuber 1997

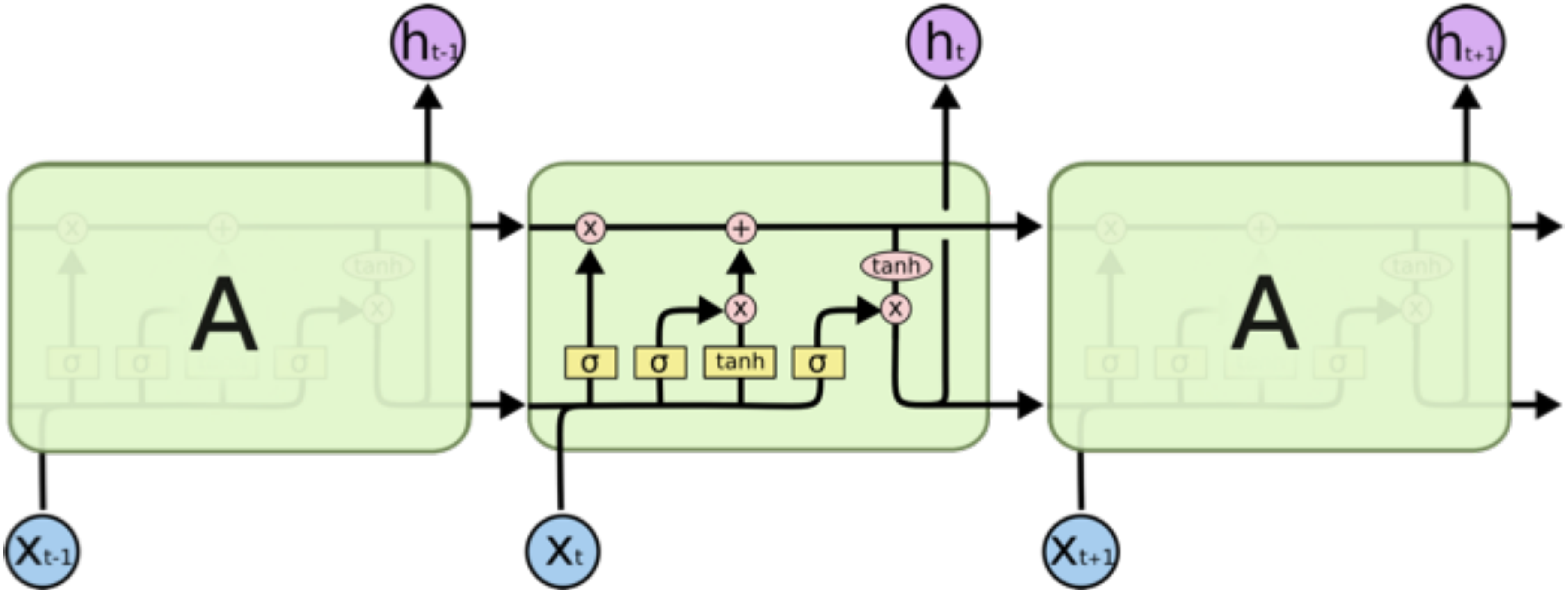
Long-Short Term Memory Units

- Simple RNN has tanh as non-linearity



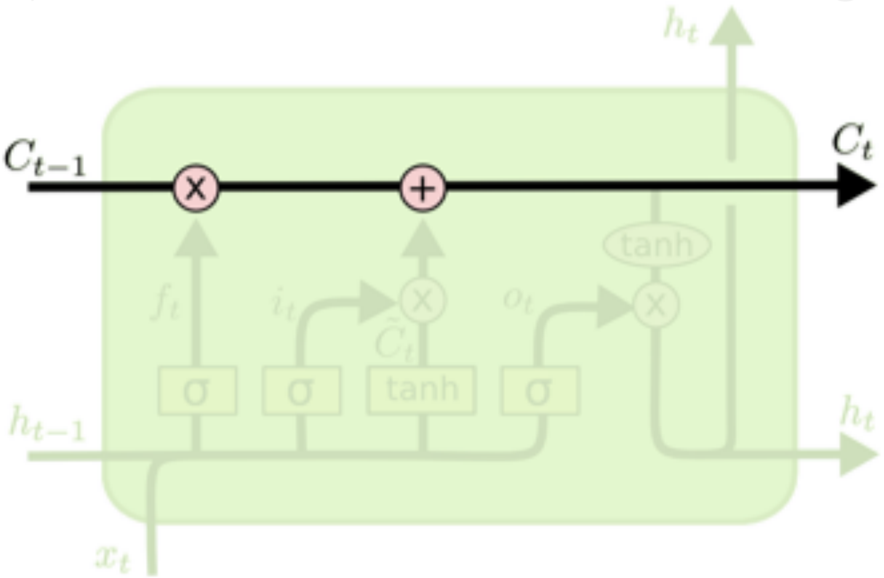
Long-Short Term Memory Units

- LSTM



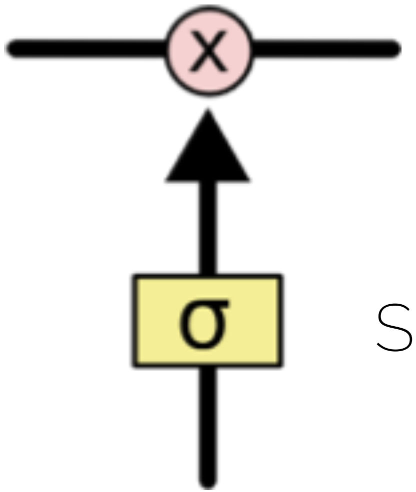
Long-Short Term Memory Units

- Key ingredients
- Cell = transports the information through the unit



Long-Short Term Memory Units

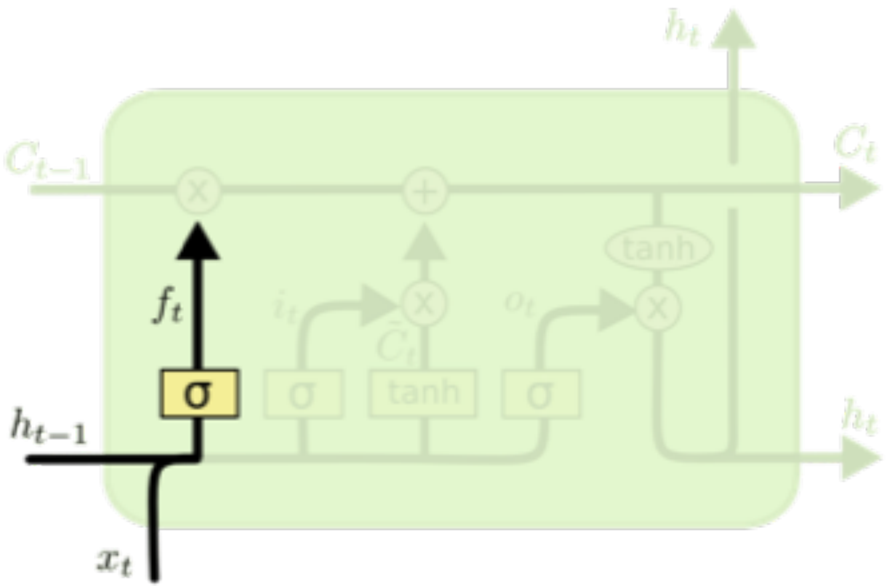
- Key ingredients
- Cell = transports the information through the unit
- Gate = remove or add information to the cell state



Sigmoid

LSTM: step by step

- Forget gate

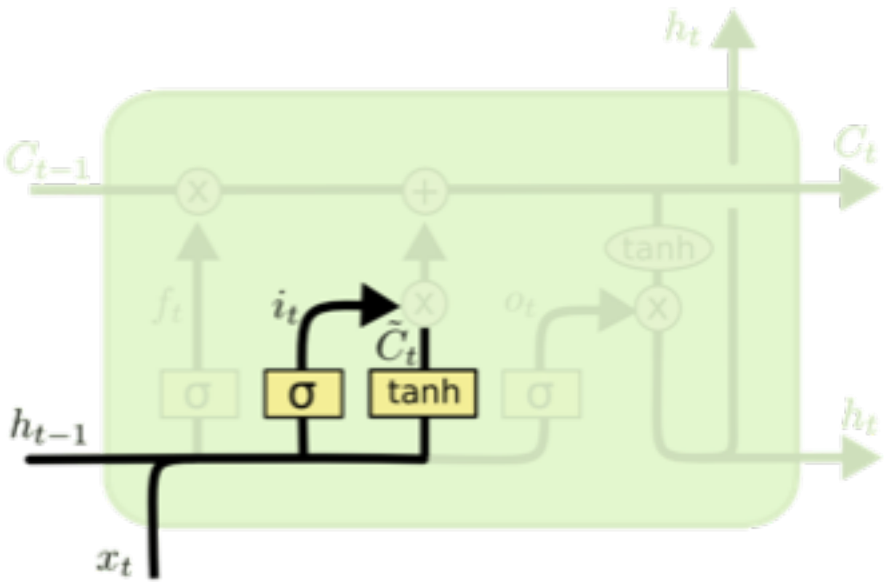


Decides when to erase the cell state

Sigmoid = output between 0 (forget) and 1 (keep)

LSTM: step by step

- Input gate

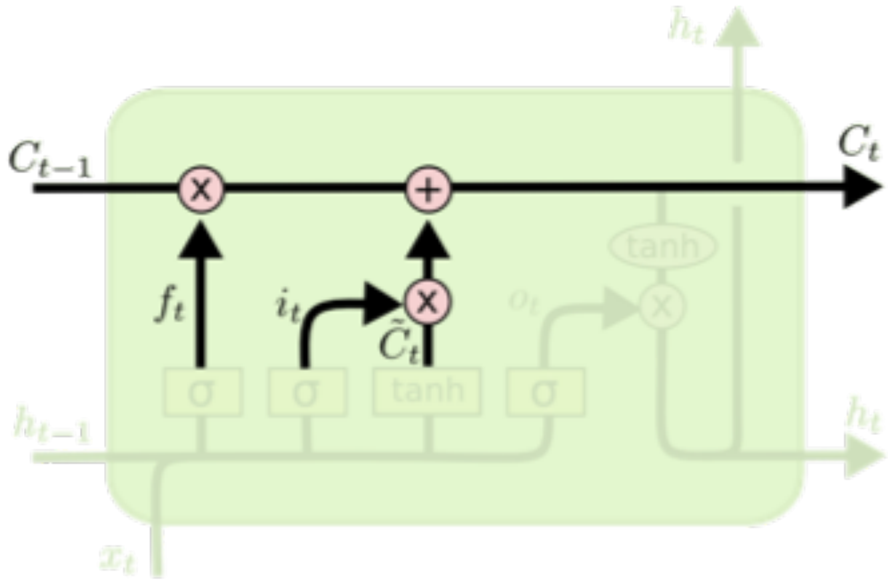


Decides which values will be updated

New cell state, output from a tanh (-1,1)

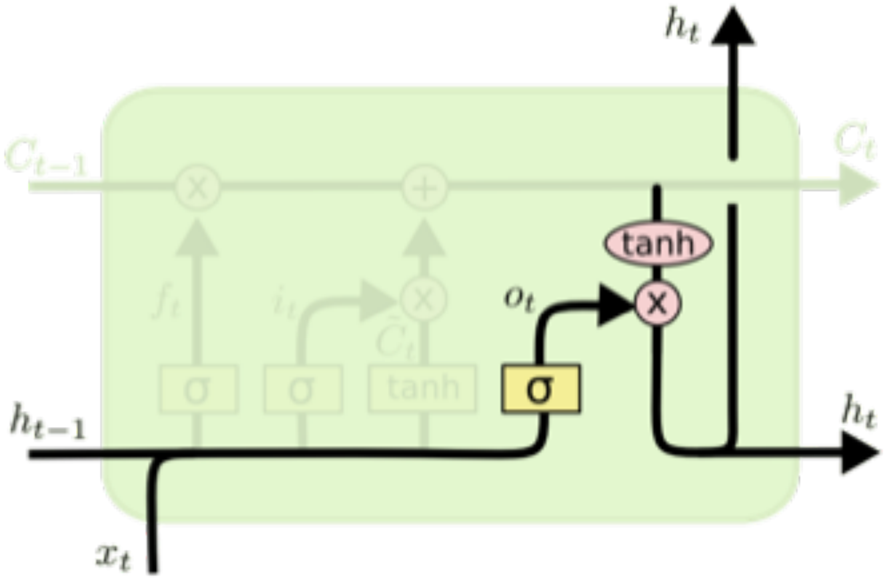
LSTM: step by step

- Element-wise operations



LSTM: step by step

- Output gate



Decides which values will be outputted

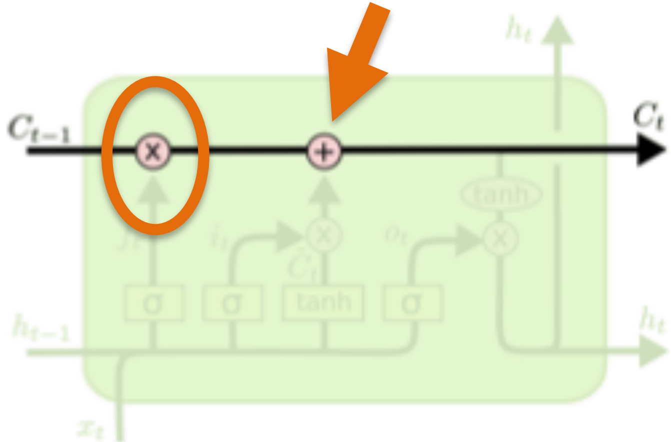
Output from a tanh (-1,1)

LSTM: step by step

- Forget gate $\mathbf{f}_t = Sigm(\boldsymbol{\theta}_{xf}\mathbf{x}_t + \boldsymbol{\theta}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$
- Input gate $\mathbf{i}_t = Sigm(\boldsymbol{\theta}_{xi}\mathbf{x}_t + \boldsymbol{\theta}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$
- Output gate $\mathbf{o}_t = Sigm(\boldsymbol{\theta}_{xo}\mathbf{x}_t + \boldsymbol{\theta}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$
- Cell update $\mathbf{g}_t = Tanh(\boldsymbol{\theta}_{xg}\mathbf{x}_t + \boldsymbol{\theta}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)$
- Cell $\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$
- Output $\mathbf{h}_t = \mathbf{o}_t \odot Tanh(\mathbf{C}_t)$

LSTM: vanishing gradients?

- 1. From the weights
- 2. From the activation functions



1 for important information

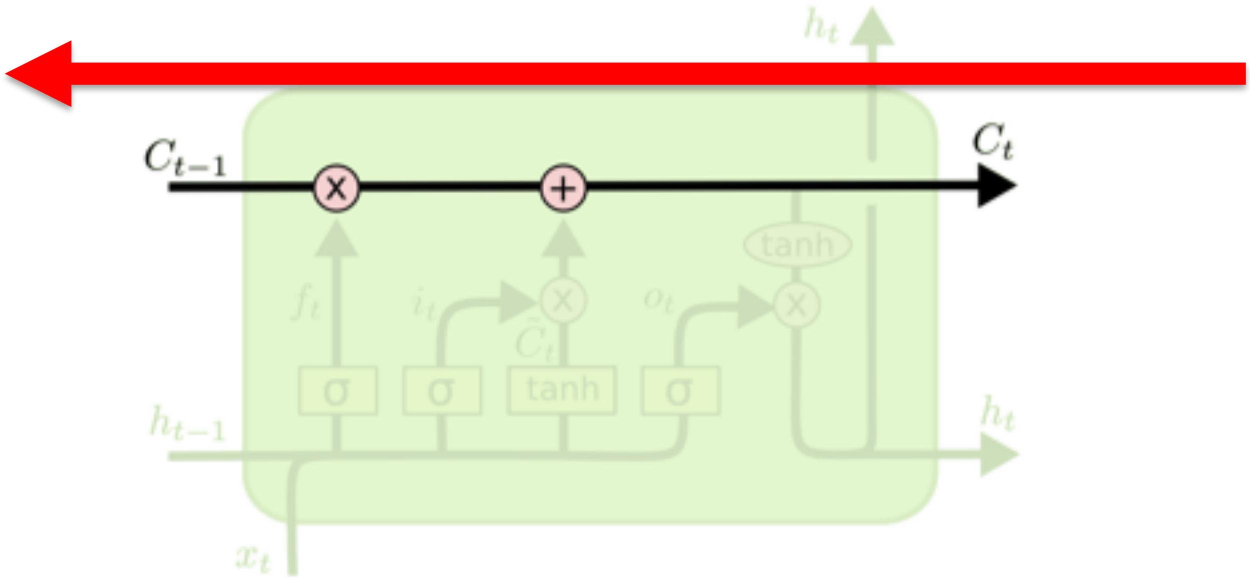
- Cell $$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

weights

Identity function

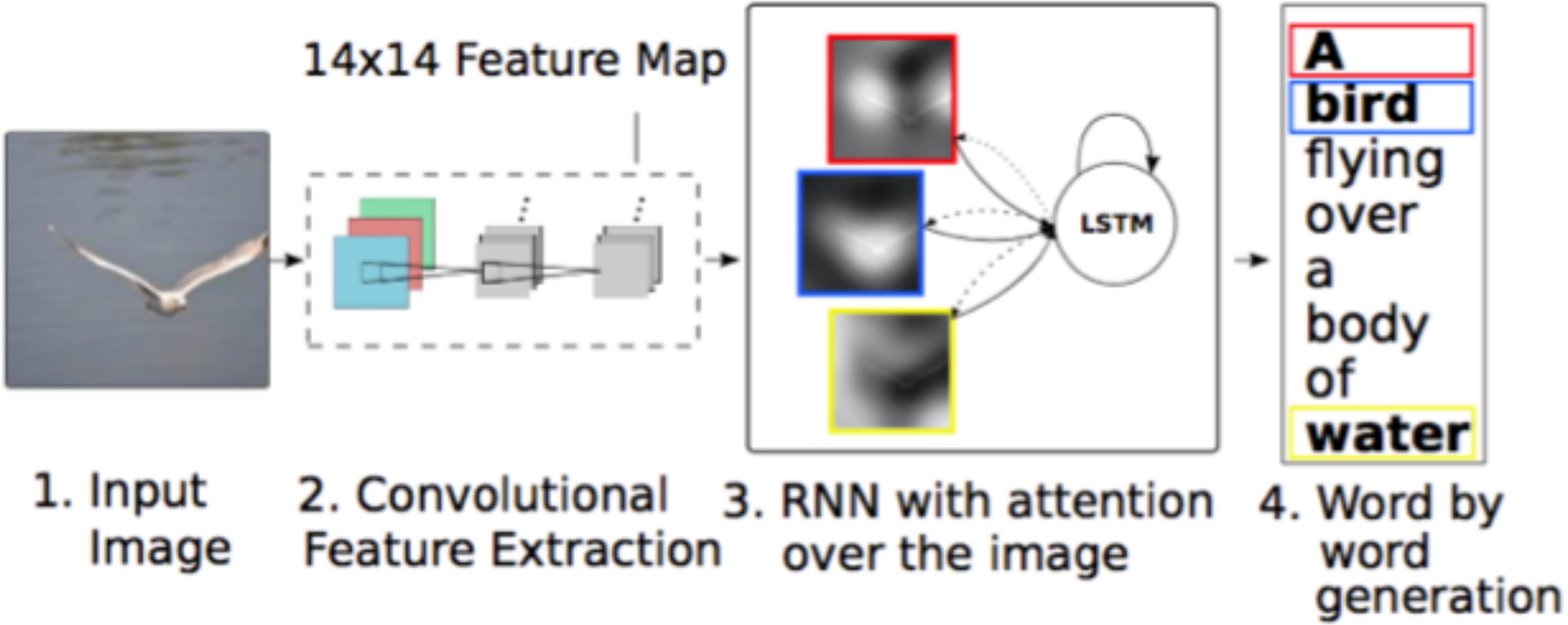
Long-Short Term Memory Units

- Highway for the gradient to flow



RNN's in Computer Vision

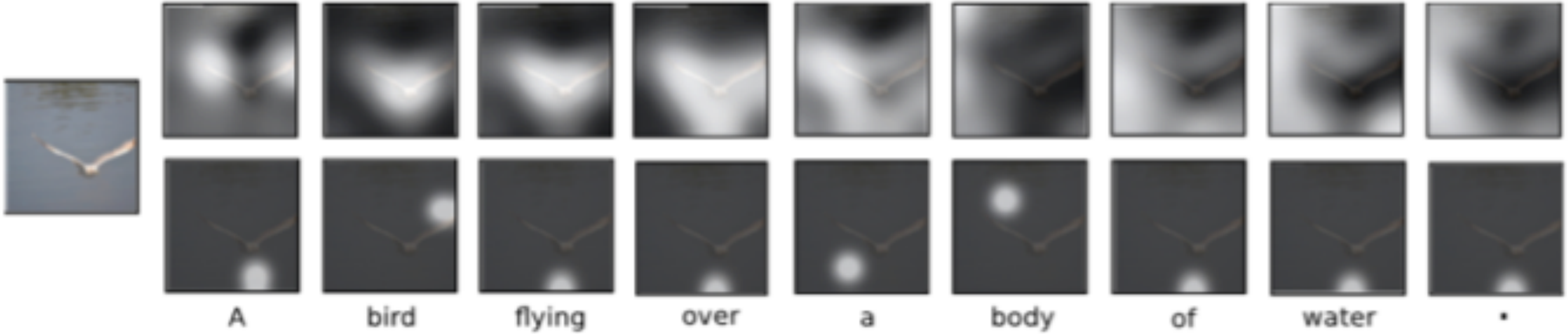
- Caption generation



Xu et al. 2015

RNN's in Computer Vision

- Caption generation

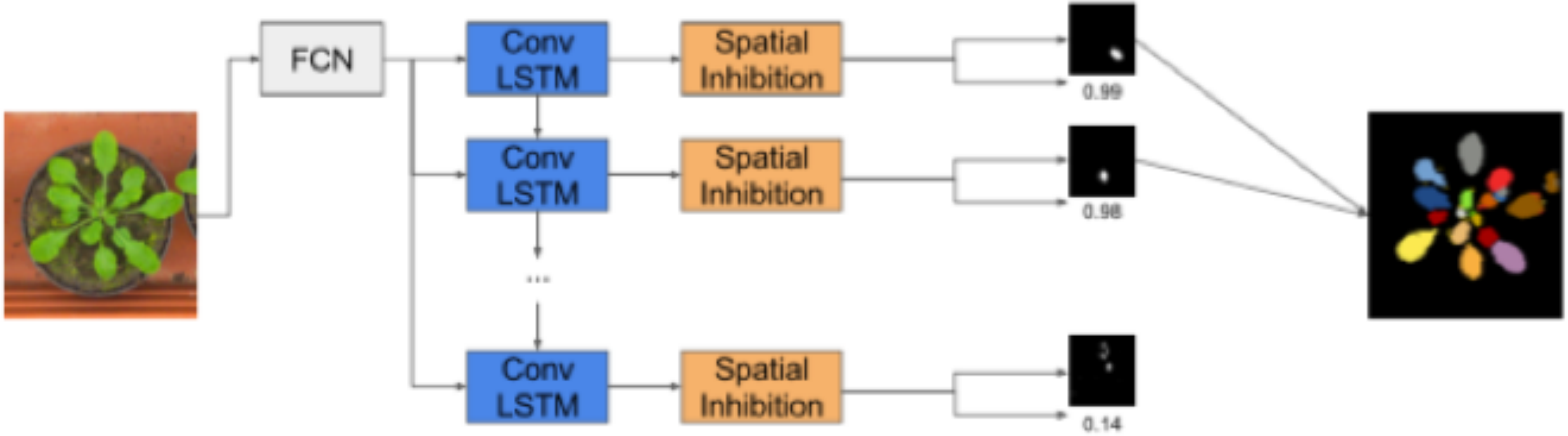


- Focus is shifted to different parts of the image

Xu et al. 2015

RNN's in Computer Vision

- Instance segmentation



Romera-Paredes et al. 2015

Final exam

Final exam

- Multiple choice questions
- Series of questions with free answer
- There can be questions related to the exercises → if you did the exercises it will be easier for you to answer them

Final exam

- Must-know topics:
  - Basics of ML → from linear classifier to NN
  - Optimization schemes (not necessary to know all the formulas, but to have a good understanding of the differences between them and their behavior
  - Backpropagation: concept, math, hint: be fluent at computing backprop by hand
  - Loss functions and activation functions
  - CNN: convolution, backprop
  - RNN, LSTMs

Admin

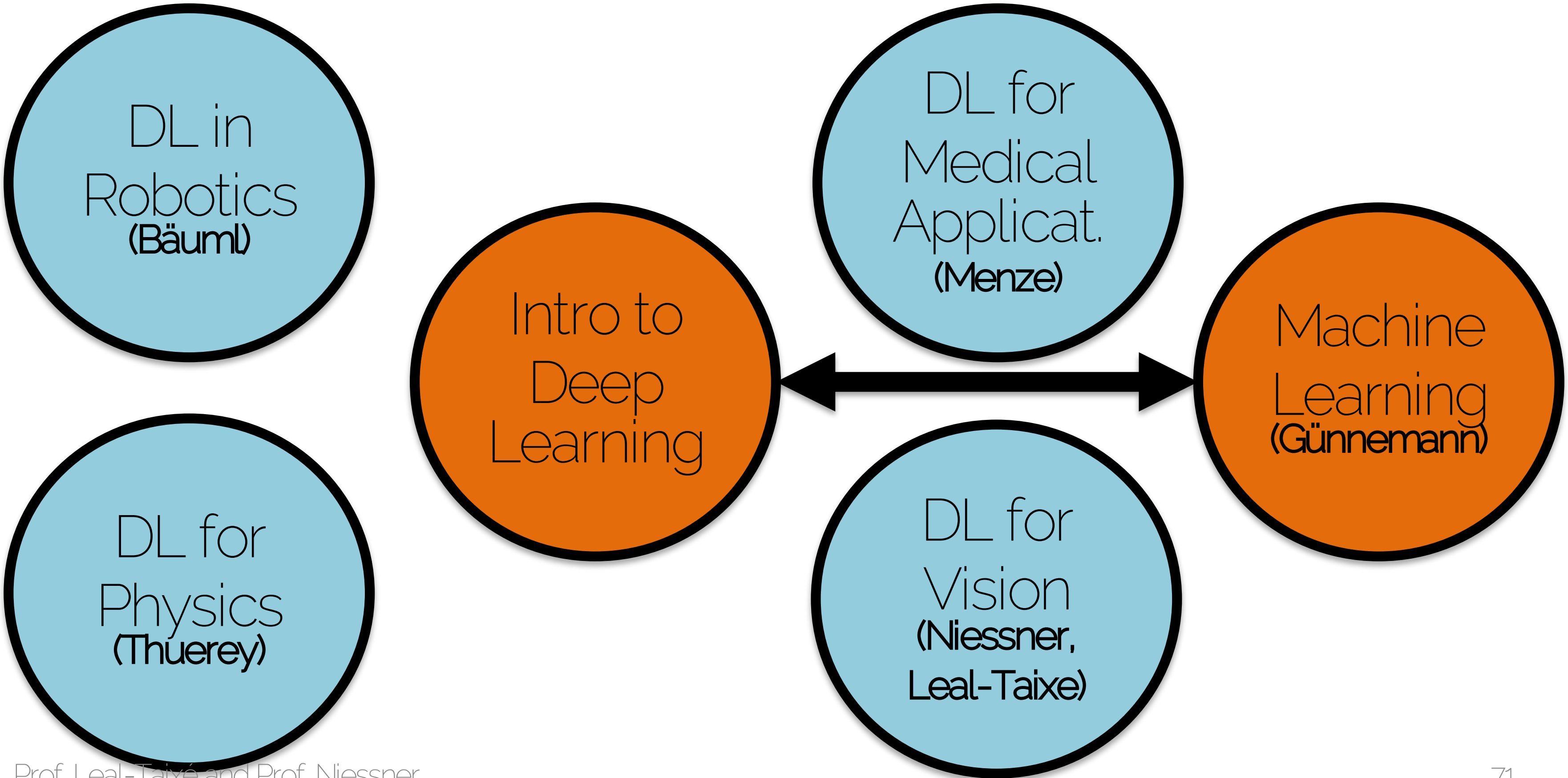
- Exam date: **July 16th at 08:00**
- There will NOT be a retake exam
- No cheat sheet nor calculator during the exam

Next semesters: new DL courses

Deep Learning at TUM

- Keep expanding the courses on Deep Learning
- This Introduction to Deep Learning course is the basis for a series of Advanced DL lectures on different topics
- Advanced topics are typically only for Master students

Deep Learning at TUM



Advanced DL for Computer Vision

- Deep Learning for Vision (WS18/19): syllabus
  - Advanced architectures, e.g. Siamese neural networks
  - Variational Autoencoders
  - Generative models, e.g. GAN,
  - Multi-dimensional CNN
  - Bayesian Deep Learning

Advanced DL for Computer Vision

- Deep Learning for Vision (WS18/19)
  - 2 V + 5 P
  - **Must** have attended the Intro to DL
  - Practical part is a project that will last the whole semester
  - Please do not sign up unless you are willing to spend a lot of time on the project!

Detection, Segmentation and Tracking

- New lecture (Prof. Leal-Taixé, SS19)
  - Must have attended the Intro to DL
  - Common detection and segmentation frameworks (YOLO, Faster-RCNN, Mask-RCNN)
  - Extension to videos → tracking
  - One project that will last the whole semester

Thank you

Niessner

Visual Computing

- 3D scanning
- DL in 3D understanding
- 3D reconstruction

Leal-Taixé

Dynamic Vision and Learning

- Video segmentation
- Object tracking
- Camera localization