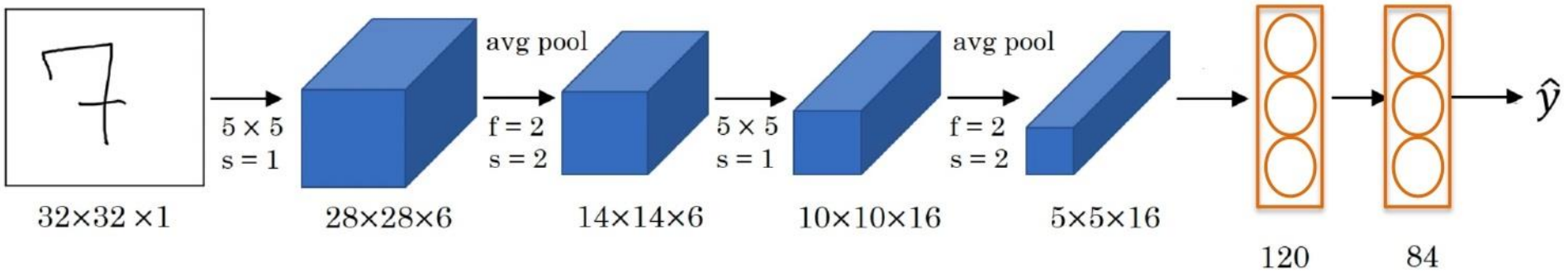
# Lecture 10 Recap

# LeNet

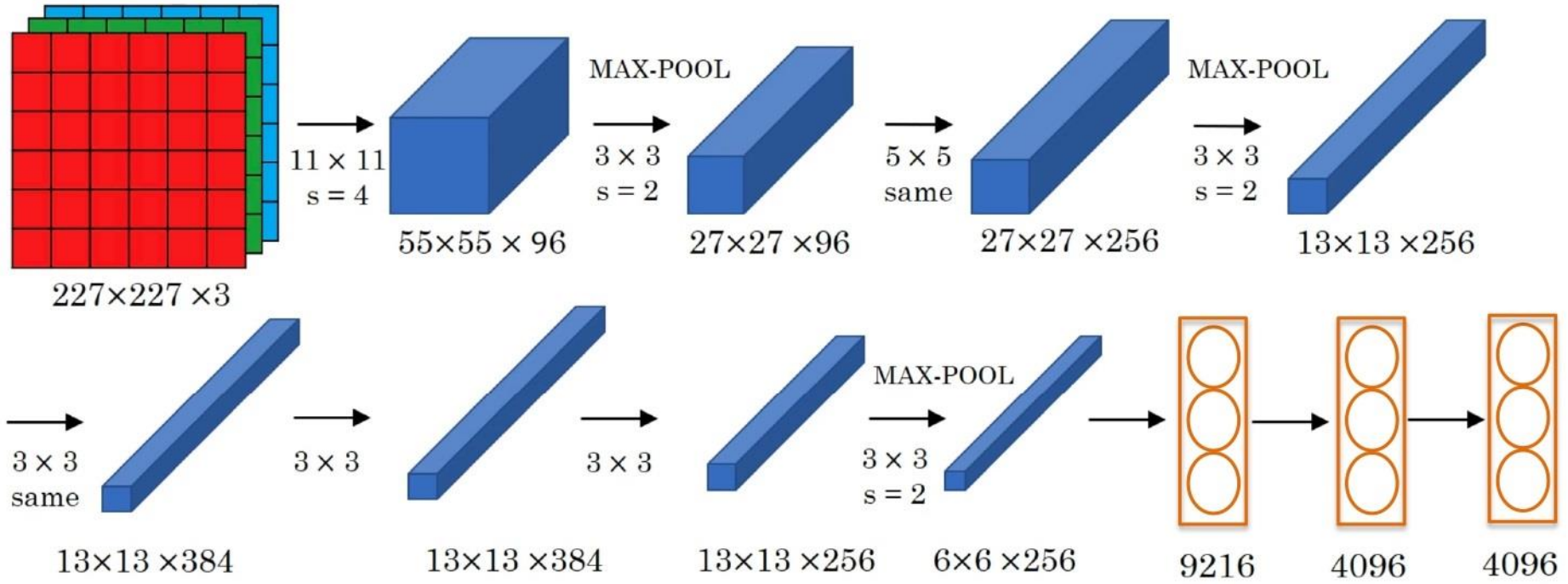
- Digit recognition: 10 classes

60k parameters




- Conv -> Pool -> Conv -> Pool -> Conv -> FC
- As we go deeper: Width, Height ↓ Number of Filters ↑

# AlexNet




- Softmax for 1000 classes

[Krizhevsky et al., ANIPS'12] AlexNet

# VGGNet

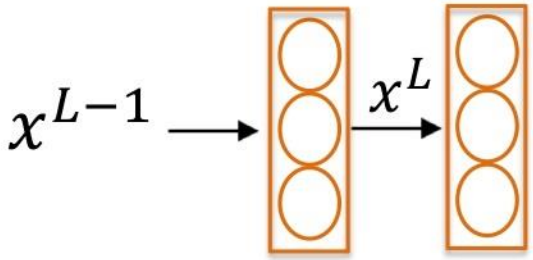
- Striving for **simplicity**
  - Conv -> Pool -> Conv -> Pool -> Conv -> FC
  - Conv=3x3, s=1, same; Maxpool=2x2, s=2
- As we go deeper: Width, Height ↓ Number of Filters ↑
- Called VGG-16: 16 layers that have weights

  138M parameters
- Large but simplicity makes it appealing

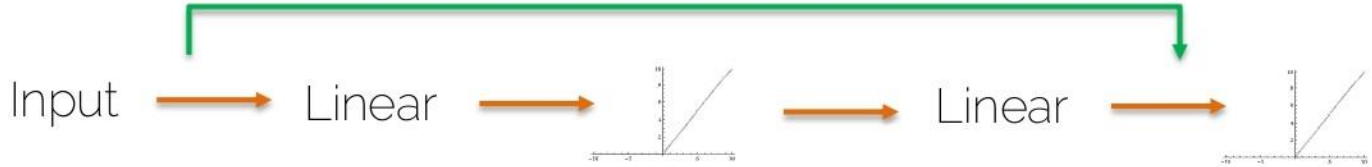
[Simonyan et al., ICLR'15] VGGNet

# Residual Block

- Two layers

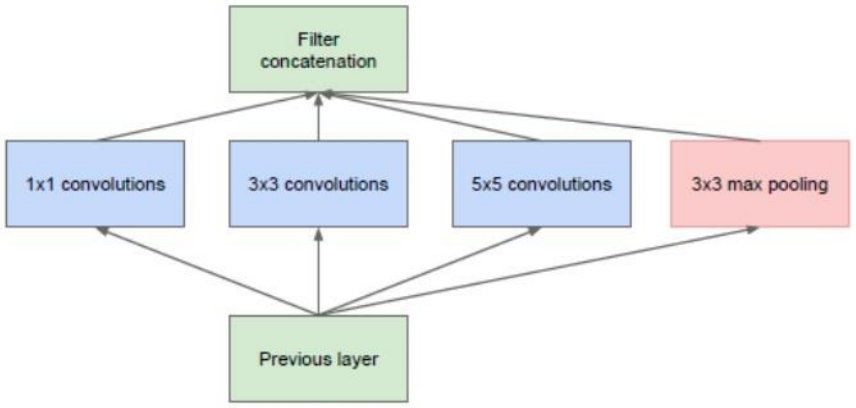



$$x^{L+1} = f(W^{L+1}x^{L} + b^{L+1} + x^{L-1})$$

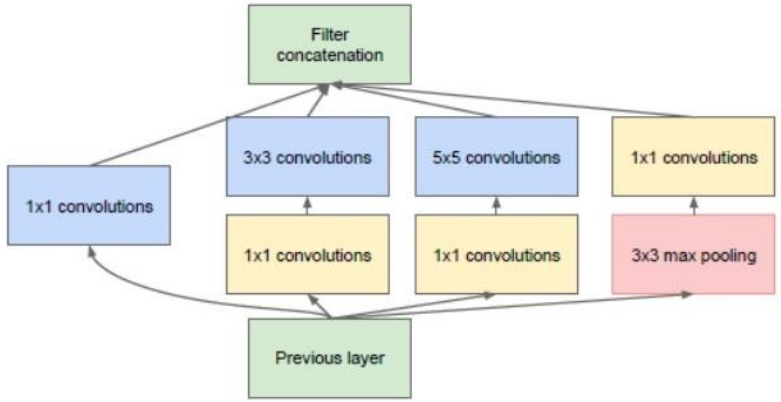



$$x^{L+1} = f(W^{L+1}x^{L} + b^{L+1})$$

# Inception Layer




(a) Inception module, naïve version




(b) Inception module with dimensionality reduction

[Szegedy et al., CVPR'15] GoogleNet

# Lecture 11

# Transfer Learning

# Transfer Learning

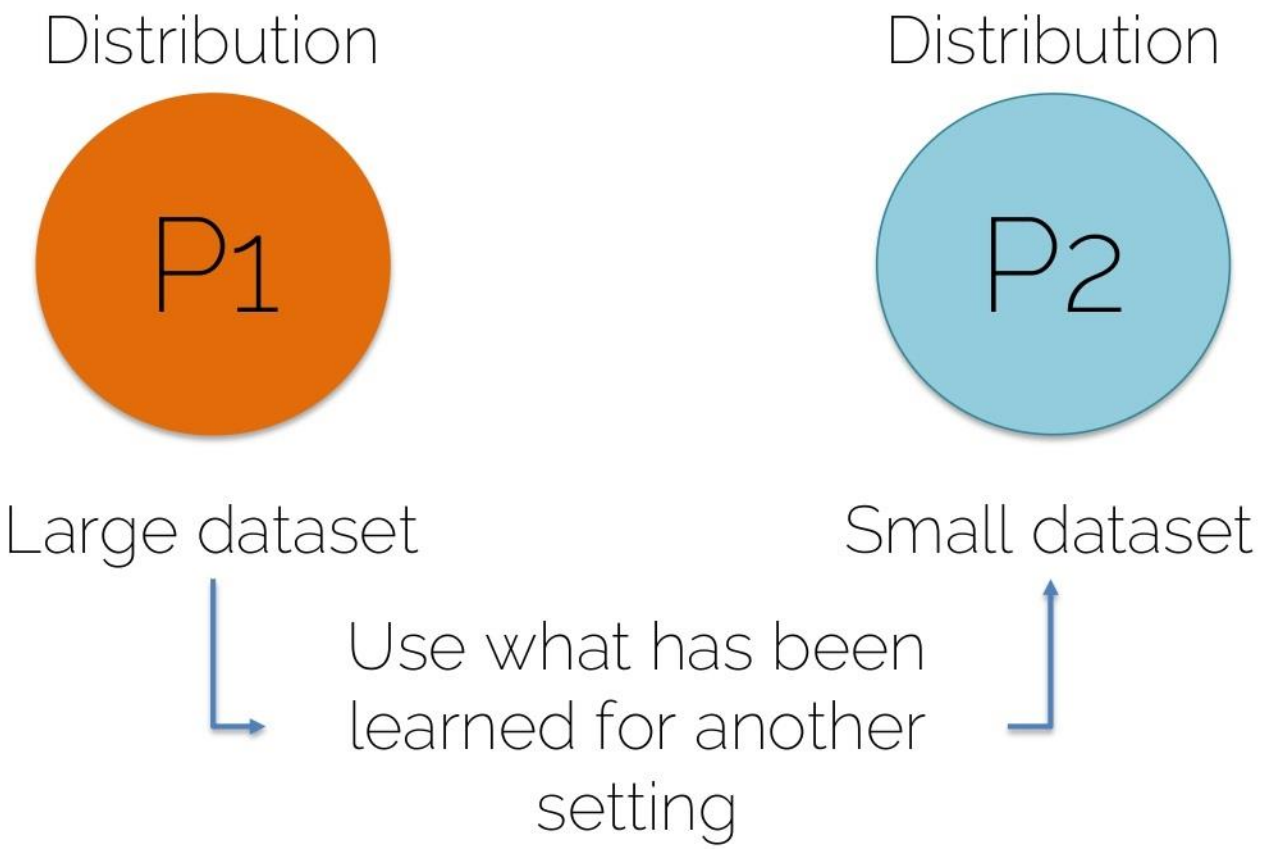
- Training your own model can be difficult with limited data and other resources

e.g.,

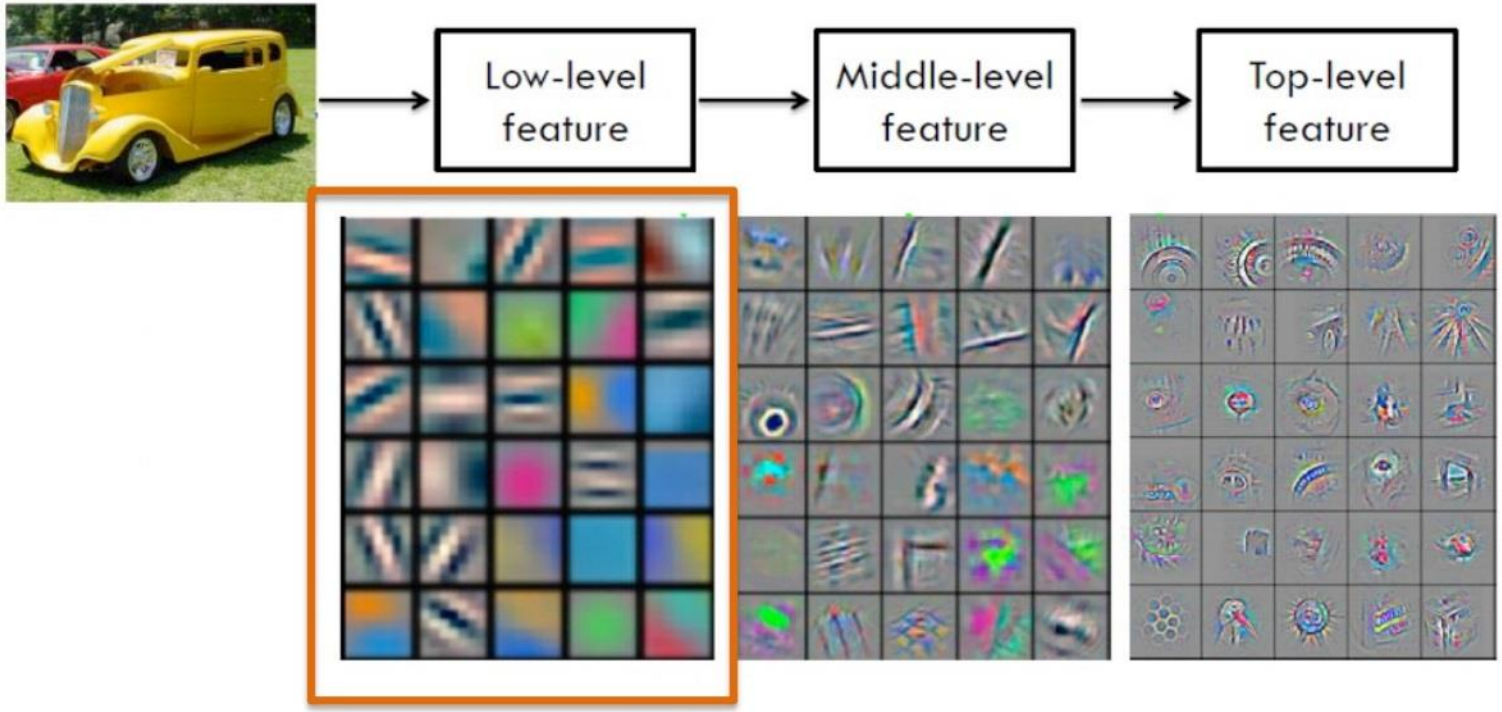
- It is a laborious task to manually annotate your own training dataset

→ Why not reuse already pre-trained models?

# Transfer Learning



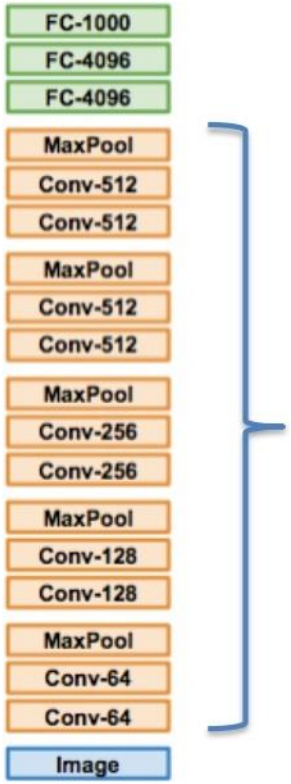
# Transfer Learning for Images




[Zeiler al., ECCV'14] Visualizing and Understanding Convolutional Networks

# Transfer Learning

Trained on ImageNet




Feature extraction

[Donahue et al., ICML'14] DeCAF,
[Razavian et al., CVPRW'14] CNN Features off-the-shelf

# Transfer Learning

Trained on ImageNet


Decision layers

Parts of an object (wheel, window)

Simple geometrical shapes (circles, etc)

Edges

[Donahue et al., ICML'14] DeCAF,
[Razavian et al., CVPRW'14] CNN Features off-the-shelf

# Transfer Learning

Trained on ImageNet


TRAIN

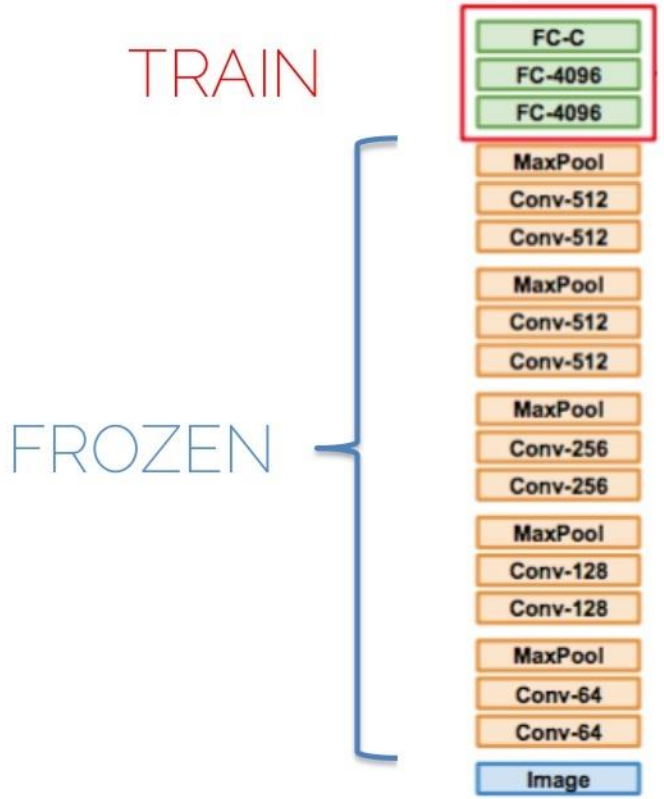

New dataset with C classes

FROZEN

[Donahue et al., ICML'14] DeCAF,
[Razavian et al., CVPRW'14] CNN Features off-the-shelf

# Transfer Learning

If the dataset is big enough train more layers with a low learning rate



# When Transfer Learning makes Sense

- When task T1 and T2 have the same input (e.g. an RGB image)
- When you have more data for task T1 than for task T2
- When the low-level features for T1 could be useful to learn T2

# Now you are:

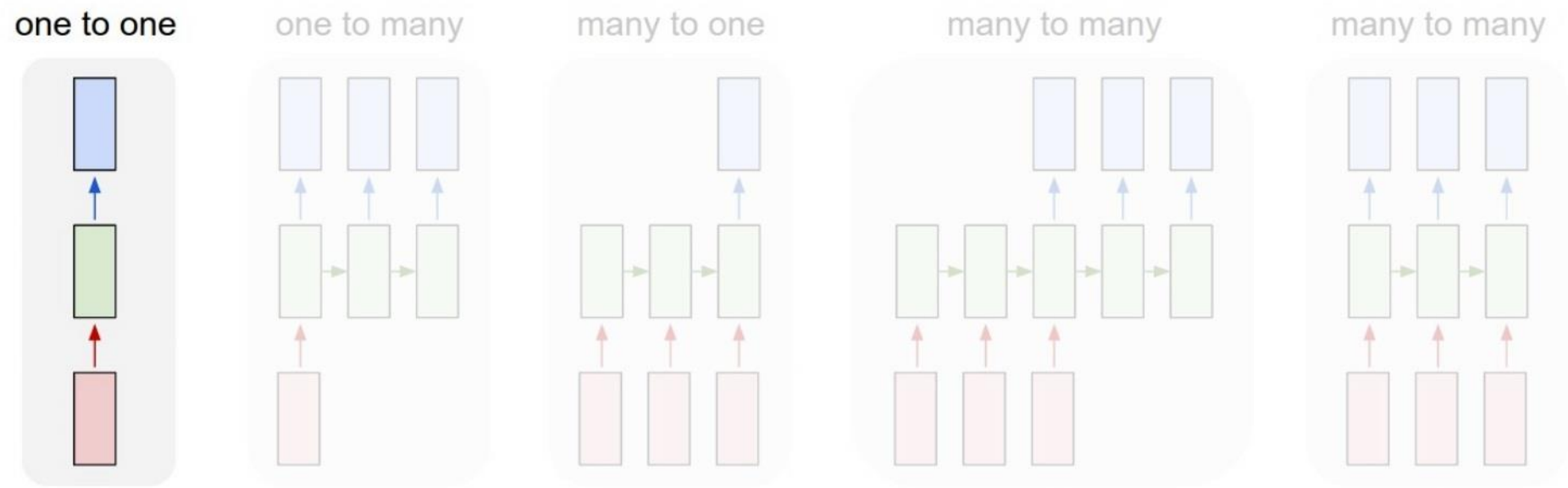
- Ready to perform image classification on any dataset
- Ready to design your own architecture
- Ready to deal with other problems such as semantic segmentation (Fully Convolutional Network)

# Recurrent Neural Networks

# Processing Sequences

- Recurrent neural networks process sequence data
- Input/output can be sequences

# RNNs are Flexible




Classic Neural Networks for Image Classification

# RNNs are Flexible

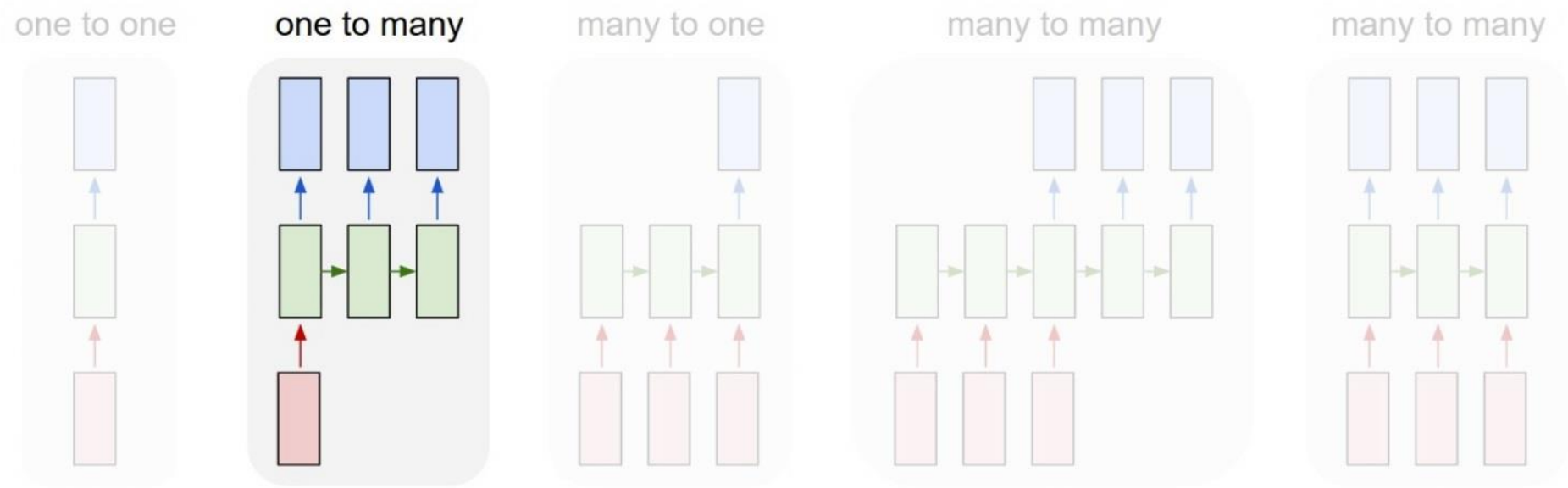
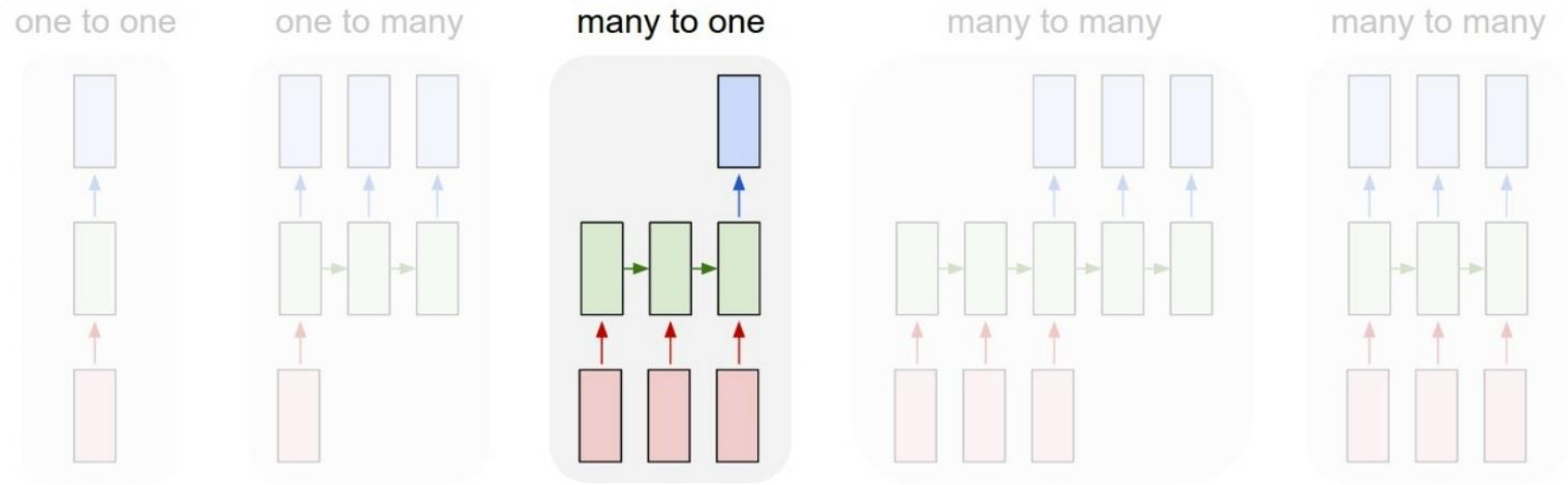


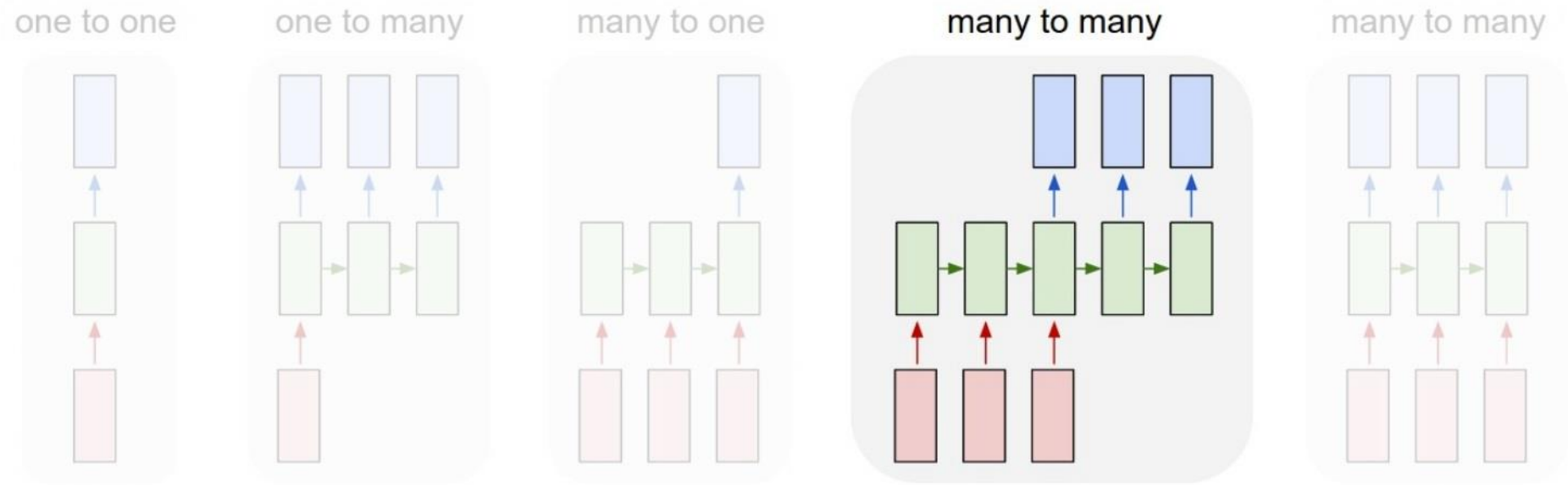

Image captioning

# RNNs are Flexible



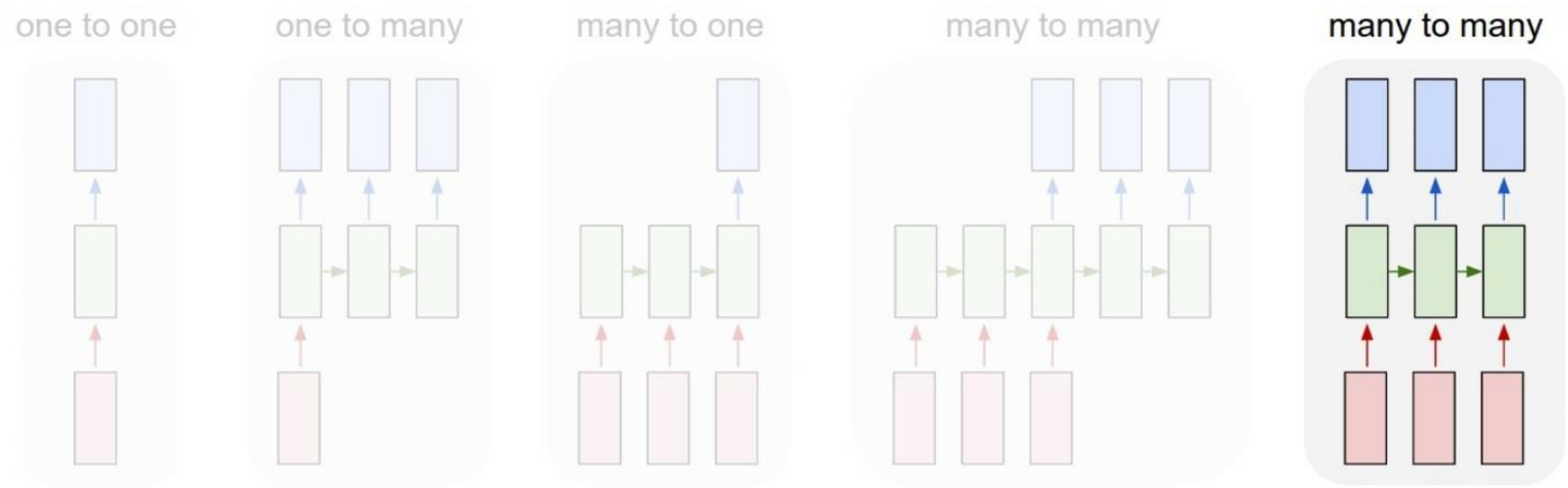

Language recognition

# RNNs are Flexible



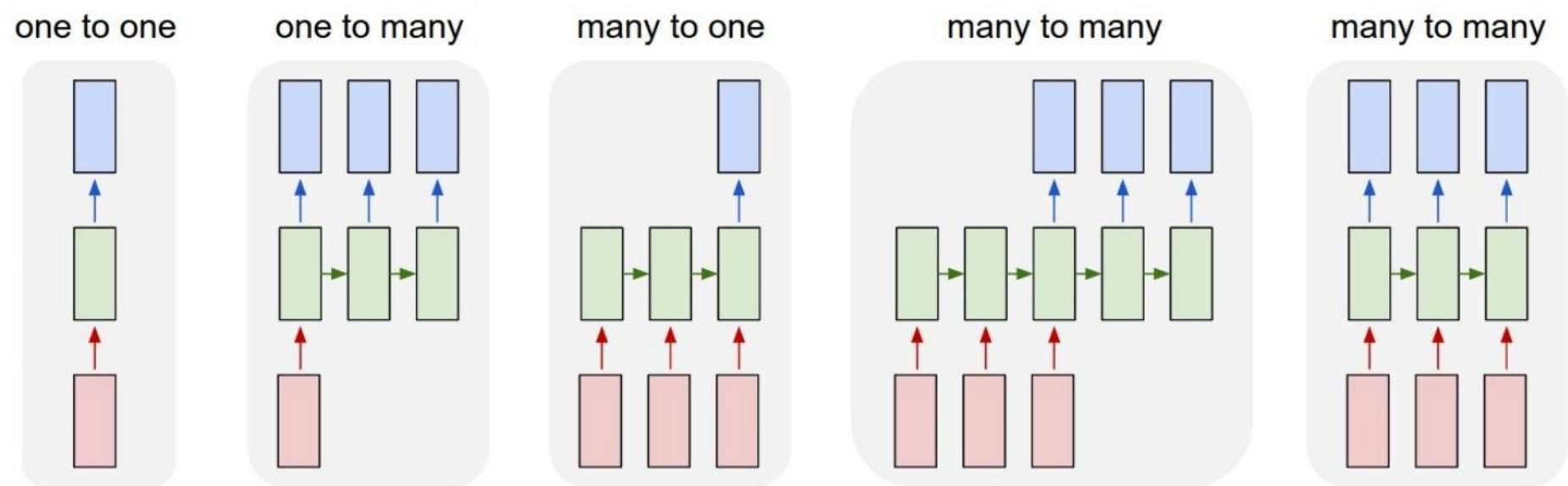

Machine translation

# RNNs are Flexible




Event classification

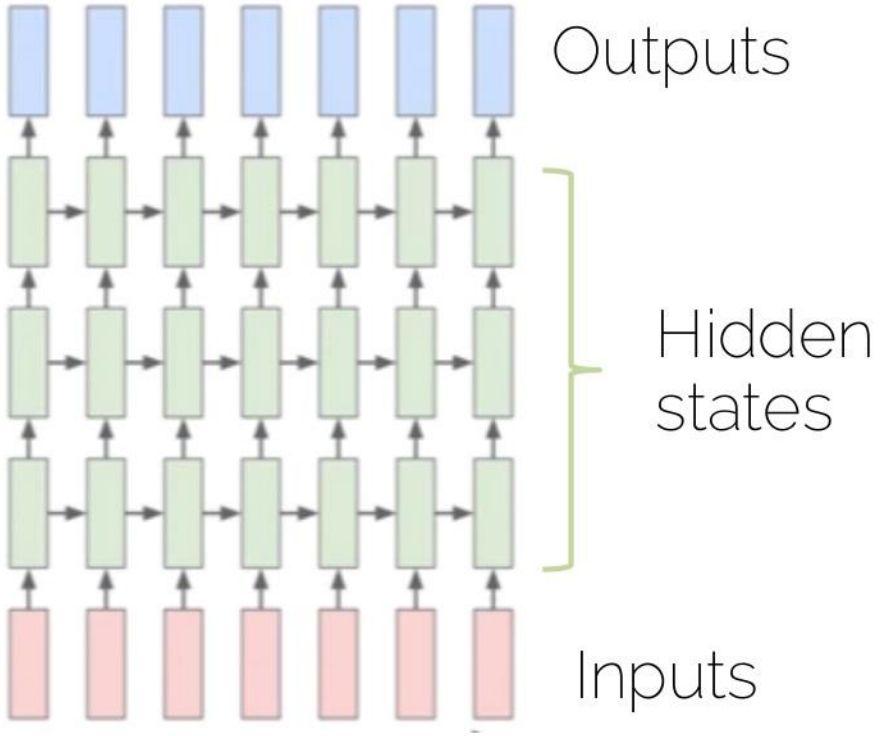
# RNNs are Flexible




Event classification

# Basic Structure of an RNN

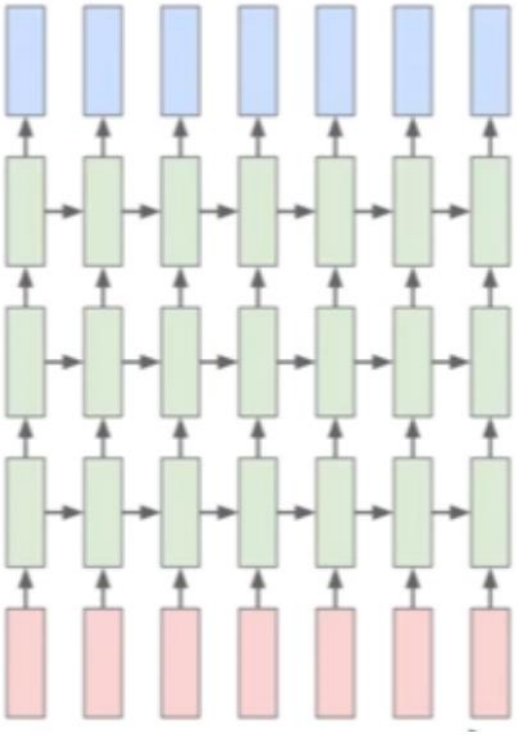
- Multi-layer RNN



# Basic Structure of an RNN

- Multi-layer RNN

The hidden state will have its own internal dynamics

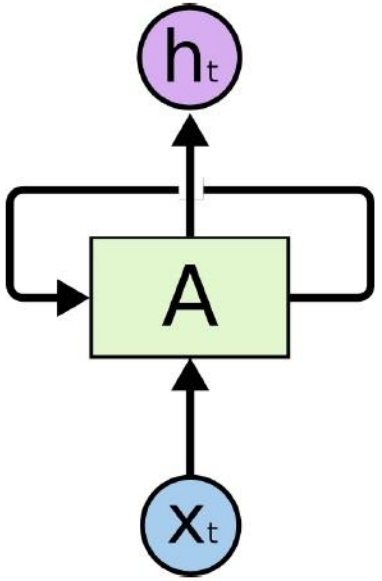


More expressive model!

# Basic Structure of an RNN

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

Hidden state




$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

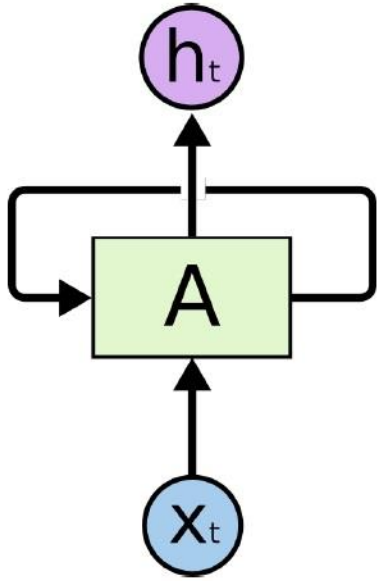
Previous hidden state

input

# Basic Structure of an RNN

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

Hidden state

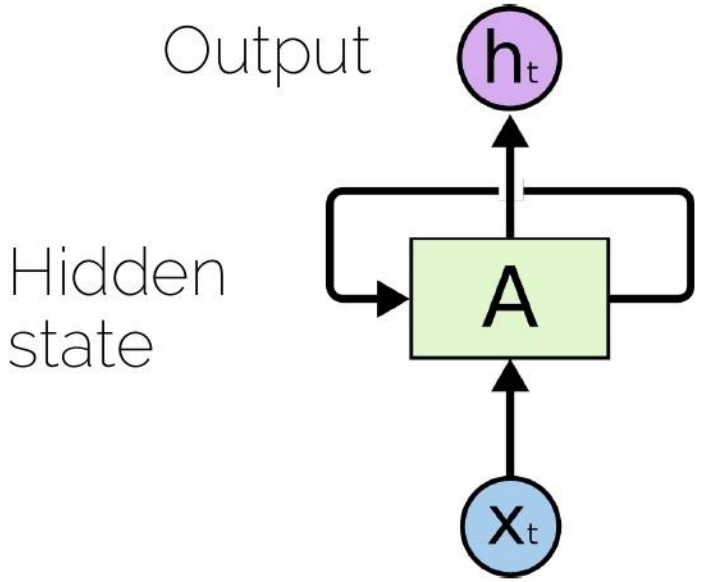



$$\boldsymbol{A}_t = \boldsymbol{\theta}_c \boldsymbol{A}_{t-1} + \boldsymbol{\theta}_x \boldsymbol{x}_t$$

Parameters to be learned

# Basic Structure of an RNN

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



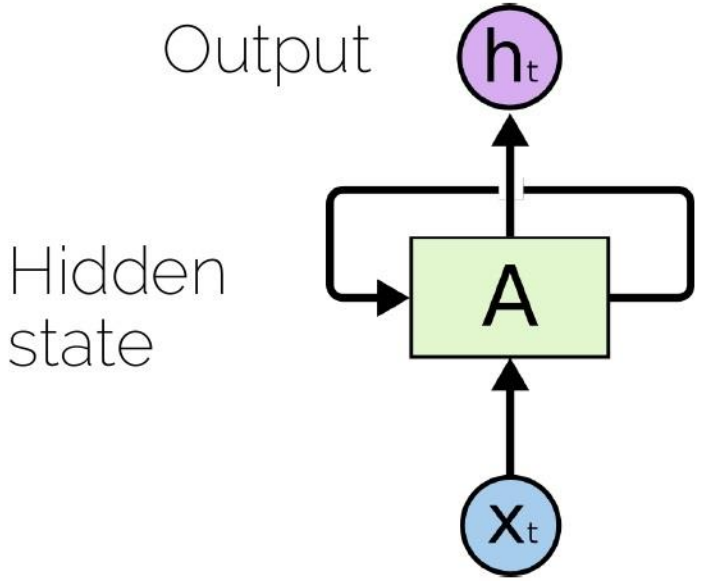

$$\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$$

$$\boldsymbol{h_t = \theta_h A_t}$$

Note: non-linearities ignored for now

# Basic Structure of an RNN

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




$$\boldsymbol{A_t} = \boldsymbol{\theta_c A_{t-1}} + \boldsymbol{\theta_x x_t}$$

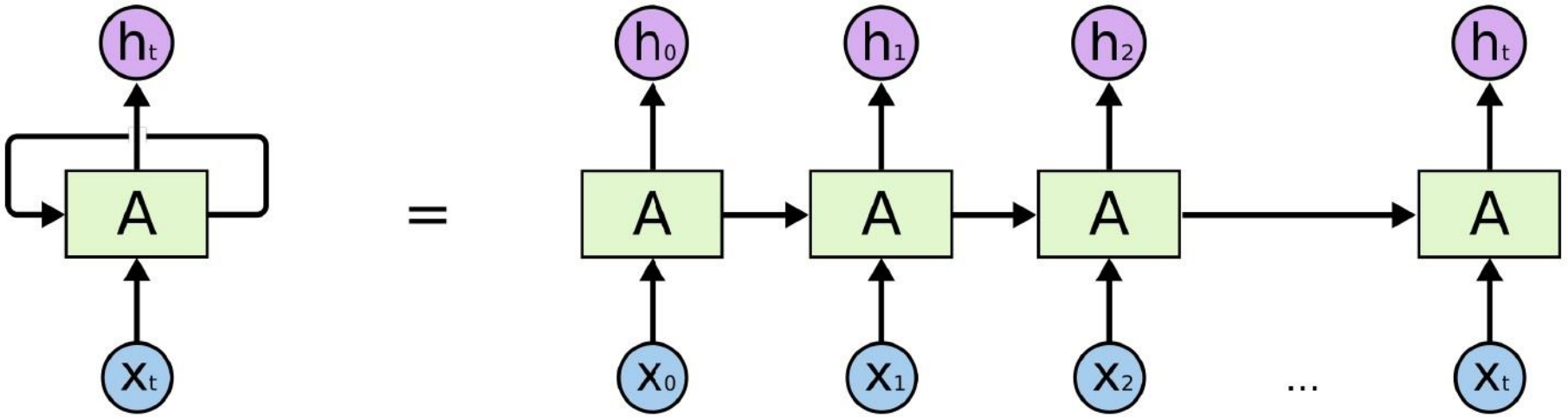
$$\boldsymbol{h_t} = \boldsymbol{\theta_h A_t}$$

Same parameters for each time step = generalization!

# Basic Structure of an RNN

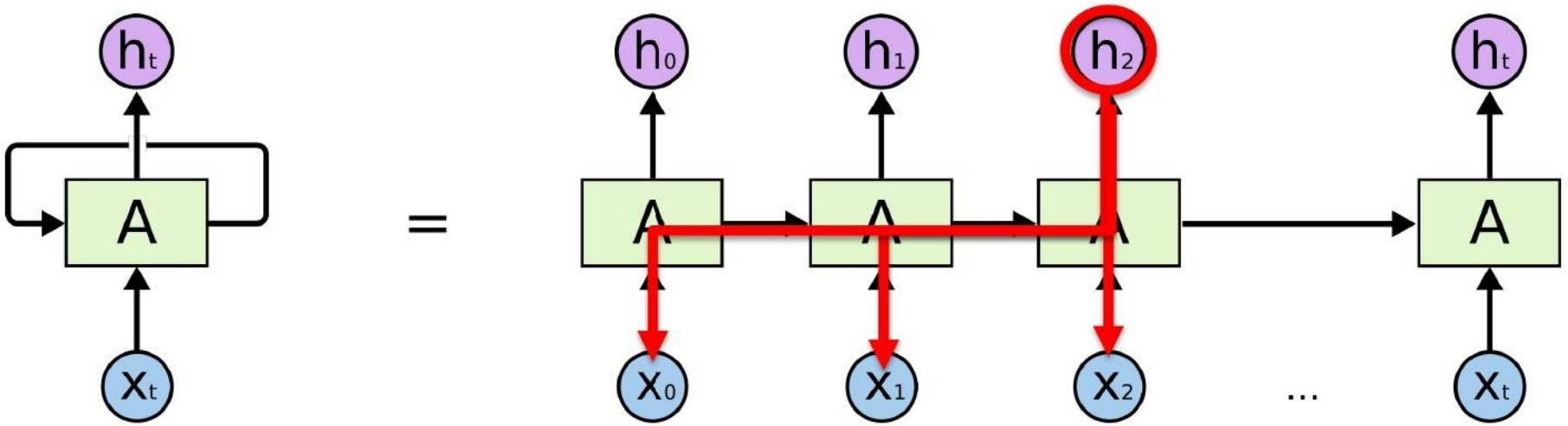
- Unrolling RNNs

Same function for the hidden layers



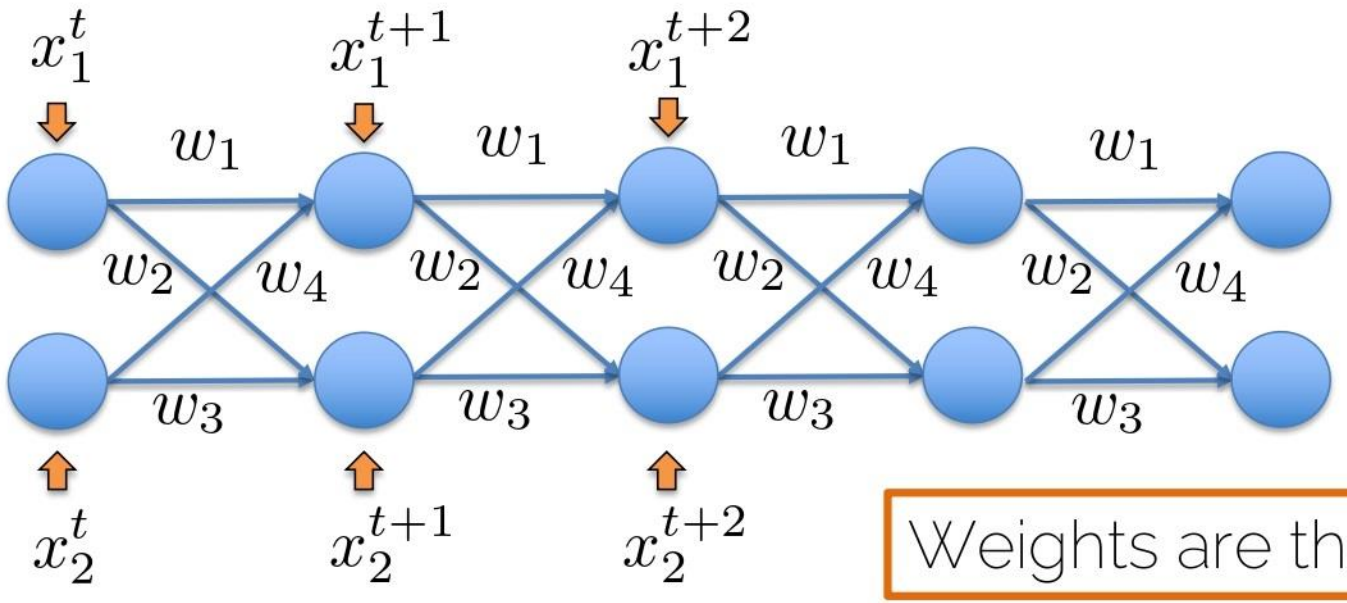
# Basic Structure of an RNN

- Unrolling RNNs



# Basic Structure of an RNN

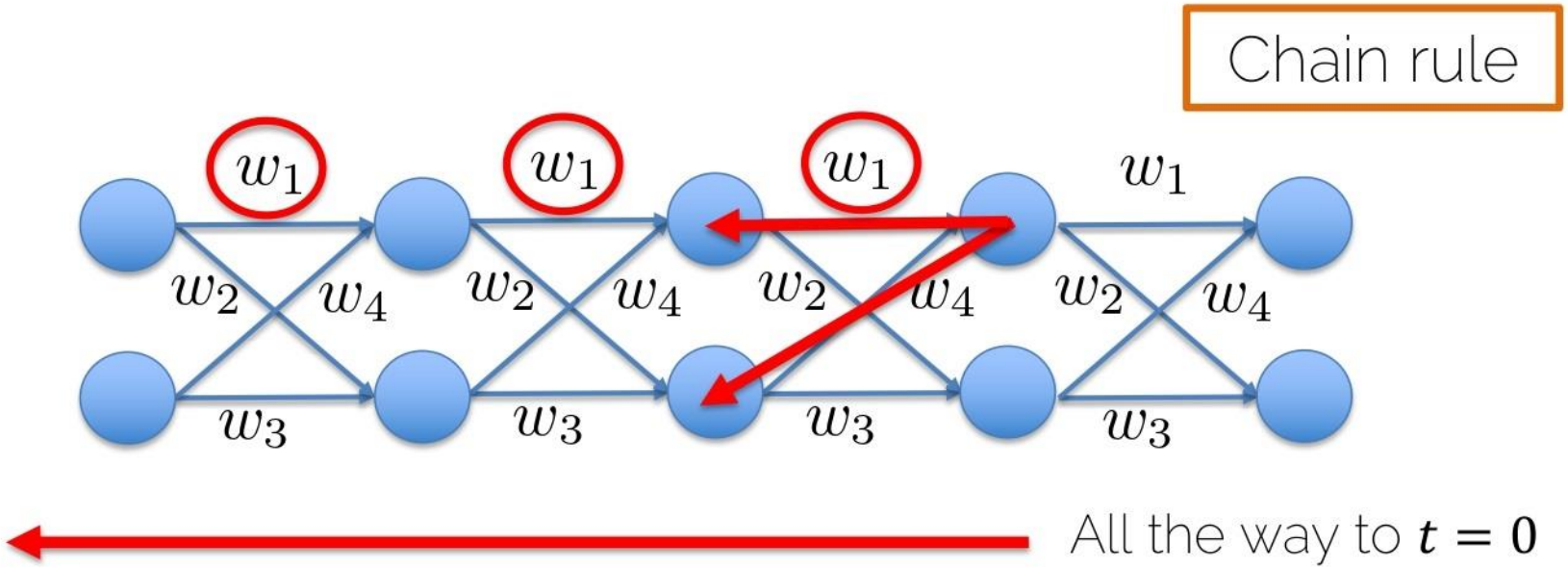
- Unrolling RNNs as feedforward nets




Weights are the same!

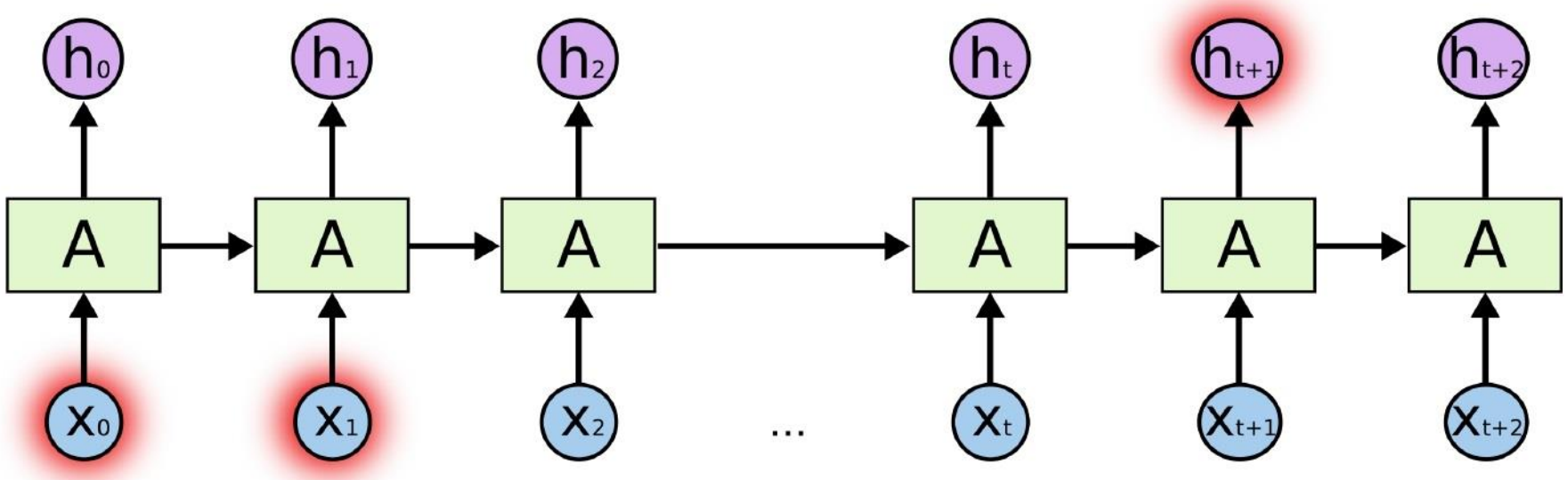
# Backprop through an RNN

- Unrolling RNNs as feedforward nets



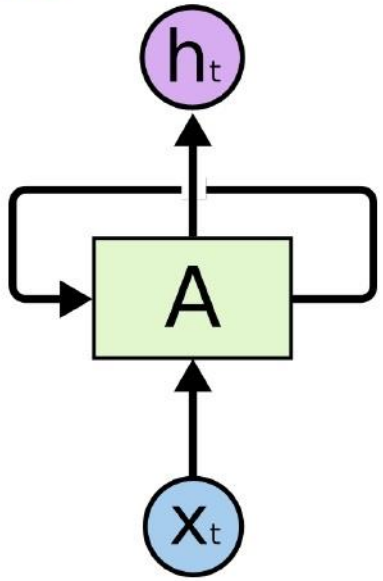

Add the derivatives at different times for each weight

# Long-term Dependencies



# Long-term Dependencies

- Simple recurrence $\boldsymbol{A_t = \theta_c A_{t-1} + \theta_x x_t}$
- Let us forget the input $\boldsymbol{A_t = \theta_c{}^t A_0}$




Same weights are multiplied over and over again

# Long-term Dependencies

- Simple recurrence $\boldsymbol{A_t = \theta_c^{\,t} A_0}$

What happens to small weights?

Vanishing gradient

What happens to large weights?

Exploding gradient

# Long-term Dependencies

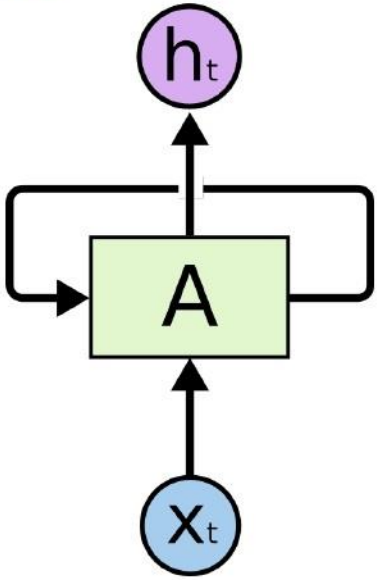
- Simple recurrence $\boldsymbol{A_t = \theta_c^{\,t} A_0}$

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

$$\boldsymbol{\theta = Q \Lambda Q^T}$$

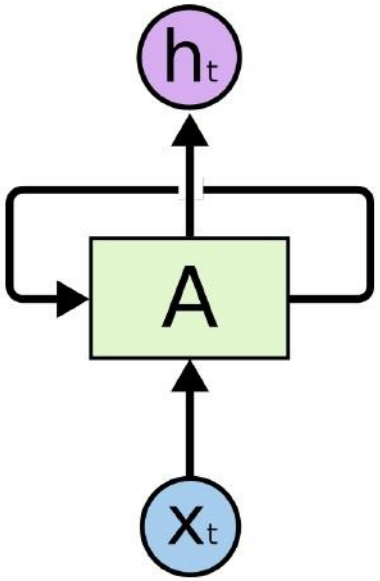
Matrix of eigenvectors

Diagonal of this matrix are the eigenvalues



# Long-term Dependencies

- Simple recurrence $\boldsymbol{A_t = \theta^t A_0}$




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

$$\boldsymbol{\theta = Q \Lambda Q^T}$$

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

$$\boldsymbol{A_t = Q \Lambda^t Q^T A_0}$$

# Long-term Dependencies

- Simple recurrence $\boldsymbol{A_t = Q\Lambda^t Q^T A_0}$

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

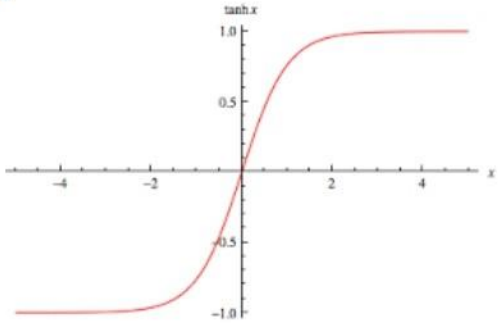
- Simple recurrence $\boldsymbol{A_t = \theta_c^{\,t} A_0}$

Let us just make a matrix with eigenvalues = 1

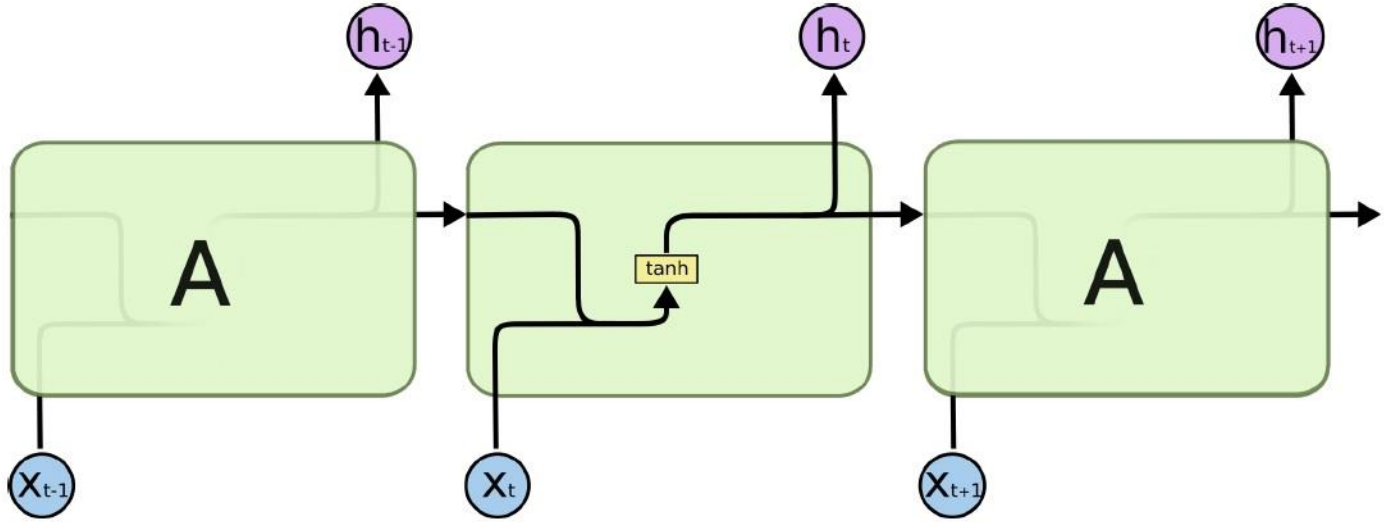
Allow the **cell** to maintain its "*state*"

# Vanishing Gradient

- 1. From the weights $\boldsymbol{A_t = \theta_c{}^t A_0}$




- 2. From the activation functions ($\boldsymbol{tanh}$)

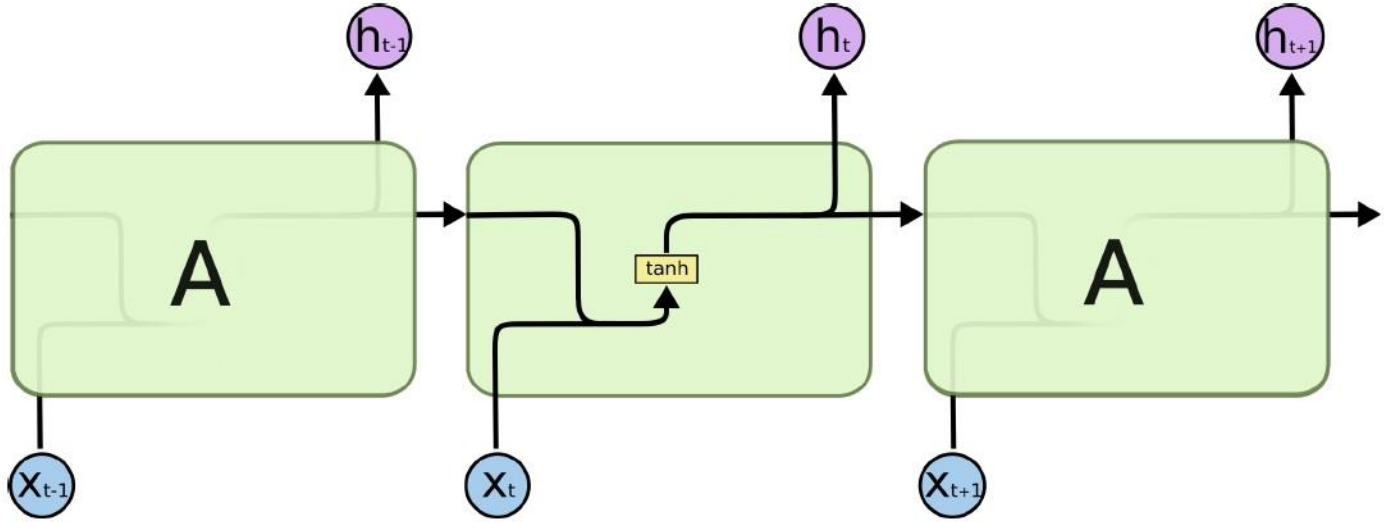


# Vanishing Gradient

- 1. From the weights $\boldsymbol{A_t = \theta^t A_0}$

  1

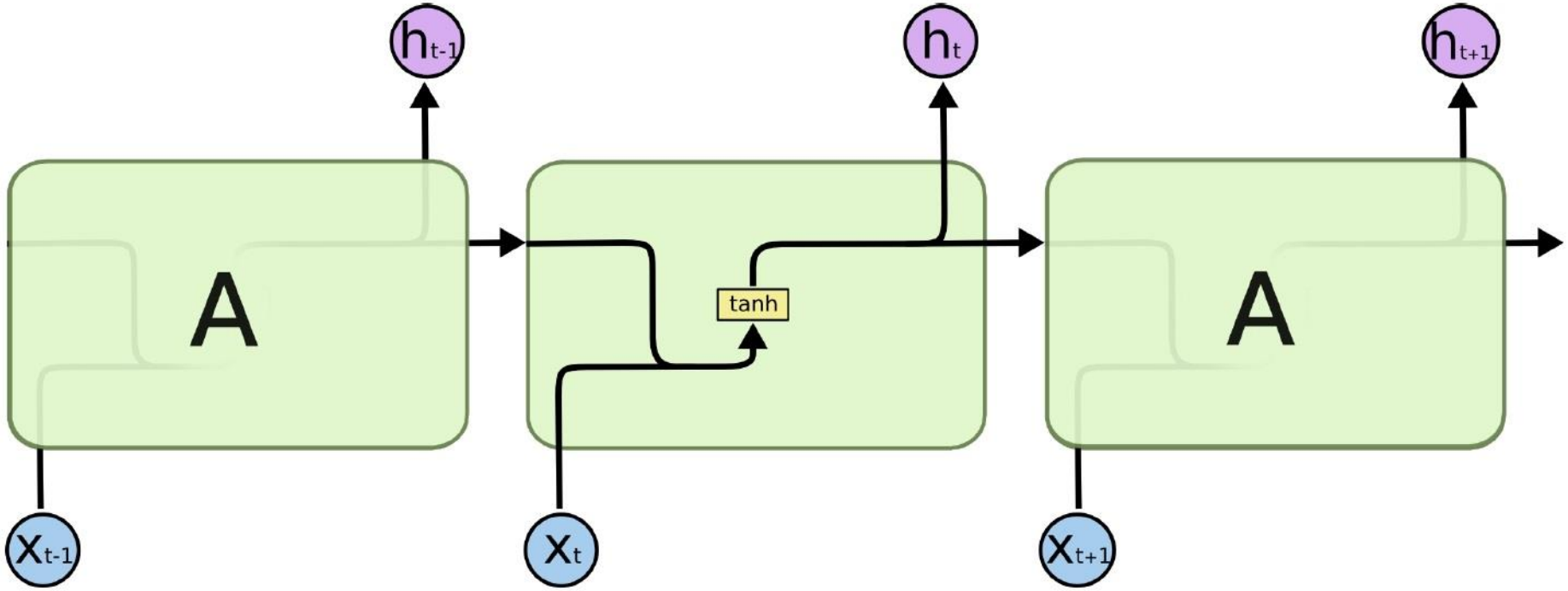
- 2. From the activation functions ($\boldsymbol{tanh}$) ?



# Long Short Term Memory

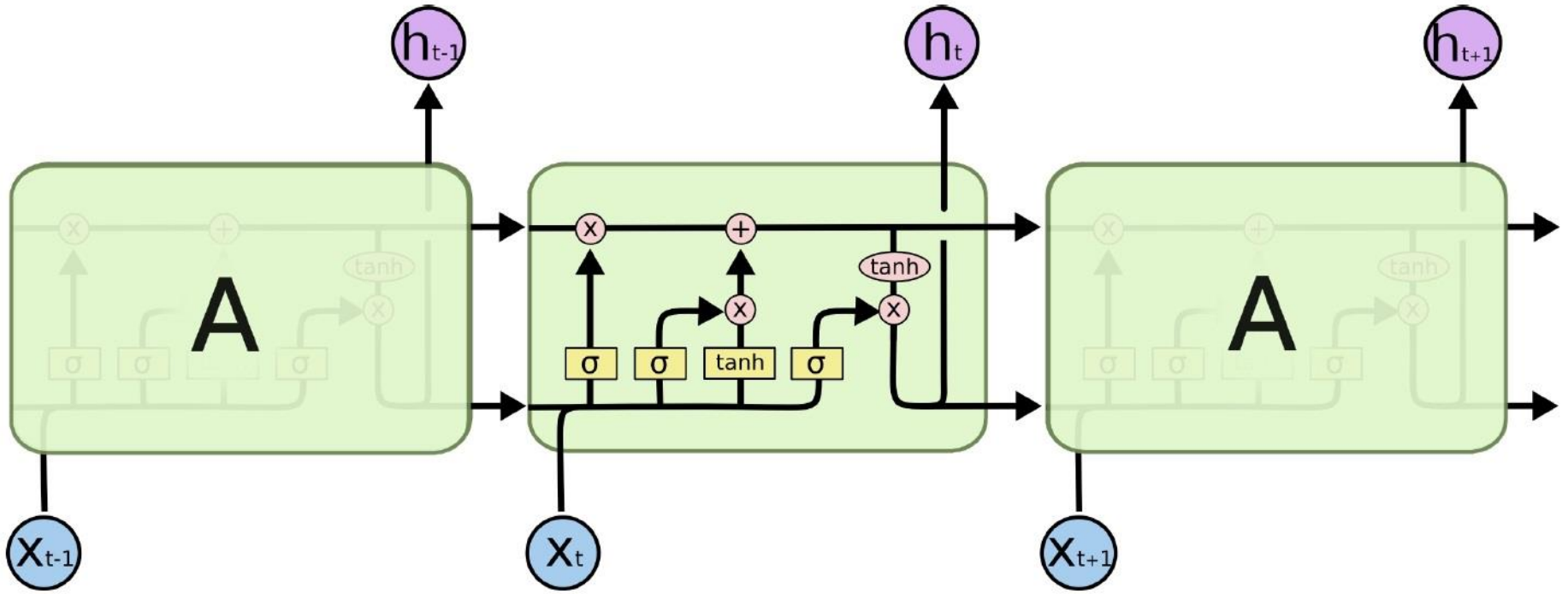
# Long-Short Term Memory Units

Simple RNN has **tanh** as non-linearity



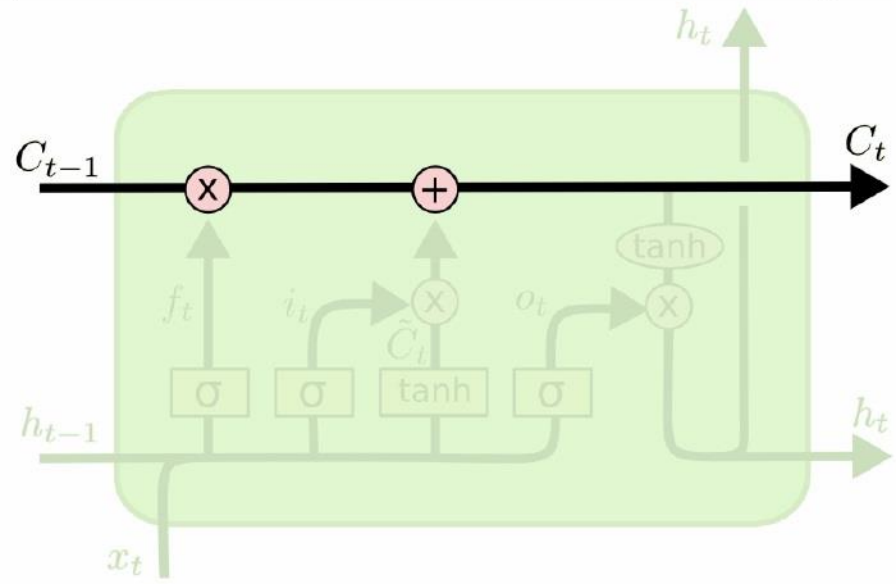
# Long-Short Term Memory Units

LSTM



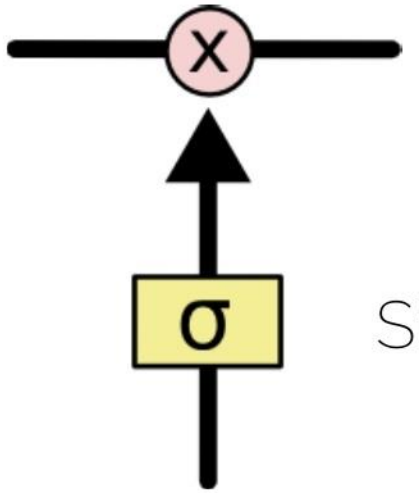
# Long-Short Term Memory Units

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



# Long-Short Term Memory Units

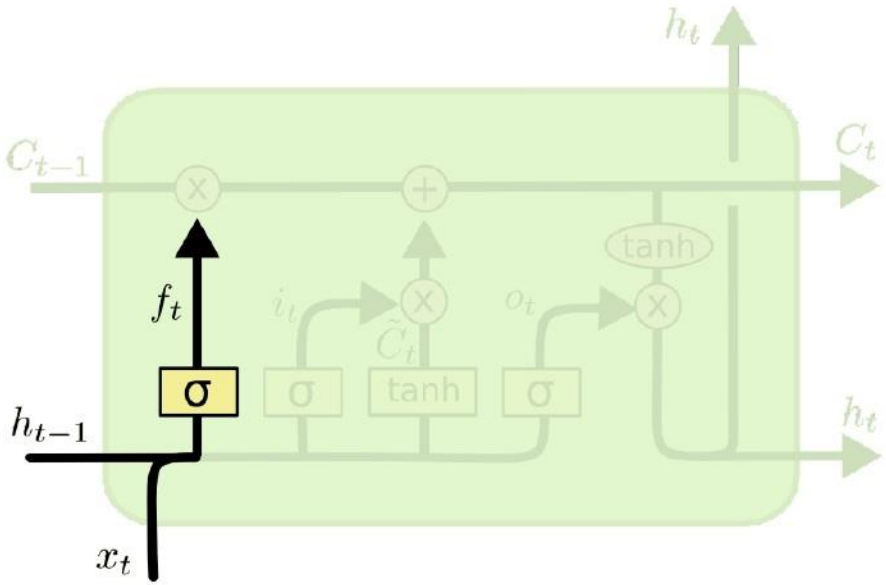
- Key ingredients
- Cell = transports the information through the unit
- Gate = remove or add information to the cell state




Sigmoid

# LSTM: Step by Step

- Forget gate $\boldsymbol{f_t = sigm(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)}$

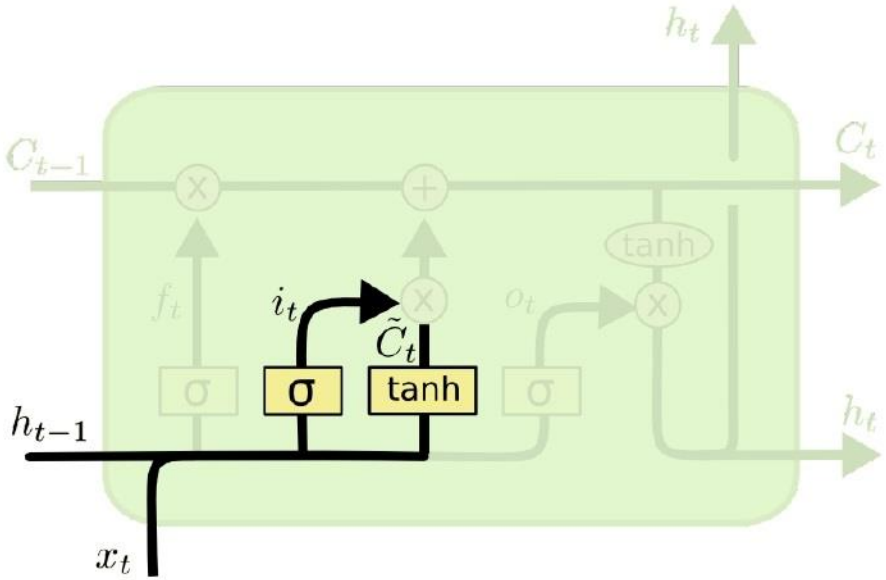



Decides when to erase the cell state

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

# LSTM: Step by Step

- Input gate $\boldsymbol{i_t = sigm(\theta_{xi} x_t + \theta_{hi} h_{t-1} + b_i)}$

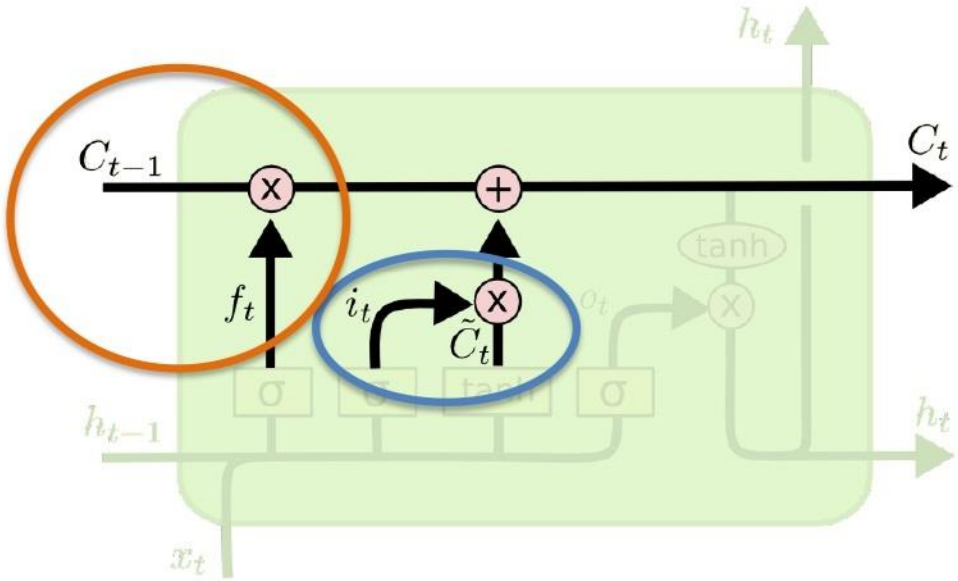



Decides which values will be updated

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

# LSTM: Step by Step

- Element-wise operations



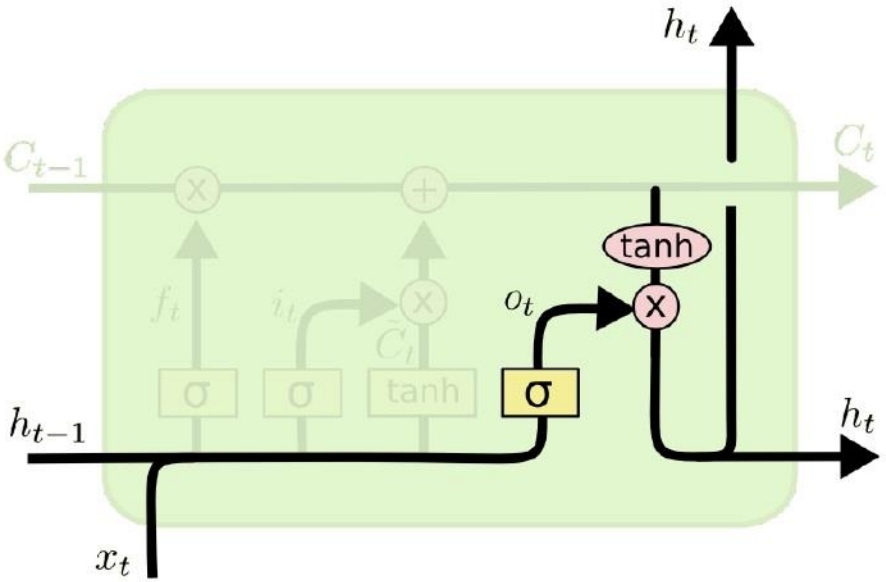

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

Previous states

Current state

# LSTM: Step by Step

- Output gate $\boldsymbol{h_t = o_t \odot \tanh(C_t)}$




Decides which values will be outputted

Output from a $\boldsymbol{\tanh(-1, 1)}$

# LSTM: Step by Step

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

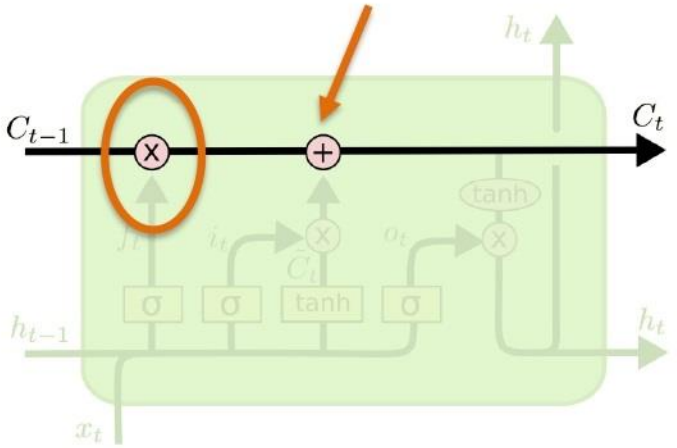
# LSTM: Step by Step

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

Learned through backpropagation

# LSTM: Vanishing Gradients?

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




- Cell $\boldsymbol{C_t} = \boldsymbol{f_t} \odot \boldsymbol{C_{t-1}} + \boldsymbol{i_t} \odot \boldsymbol{g_t}$

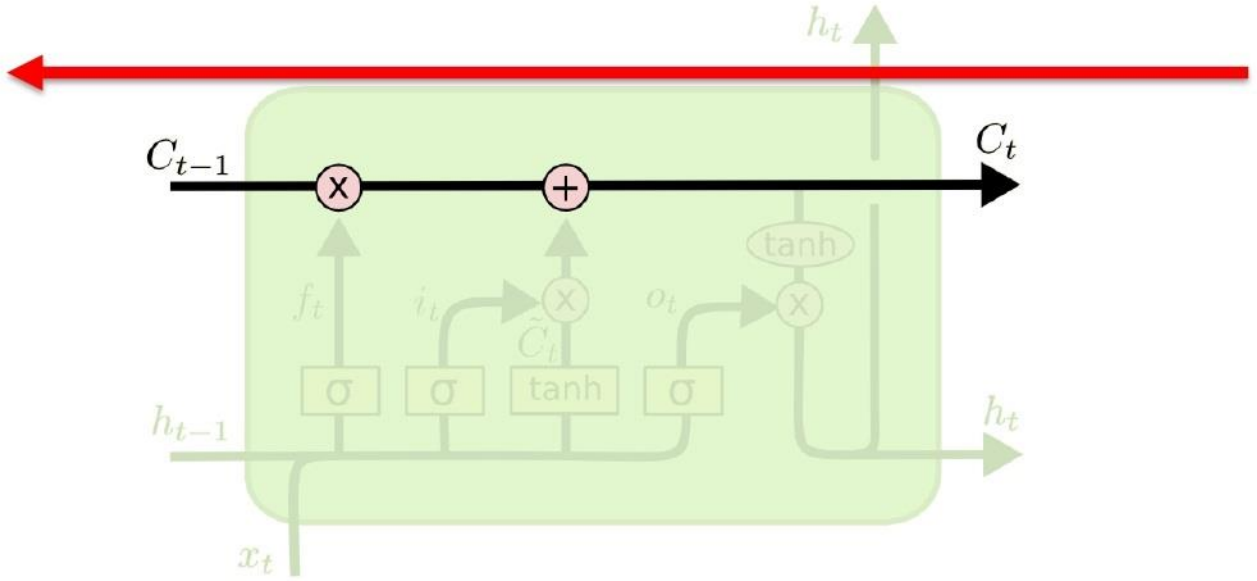
1 for important information

weights

Identity function

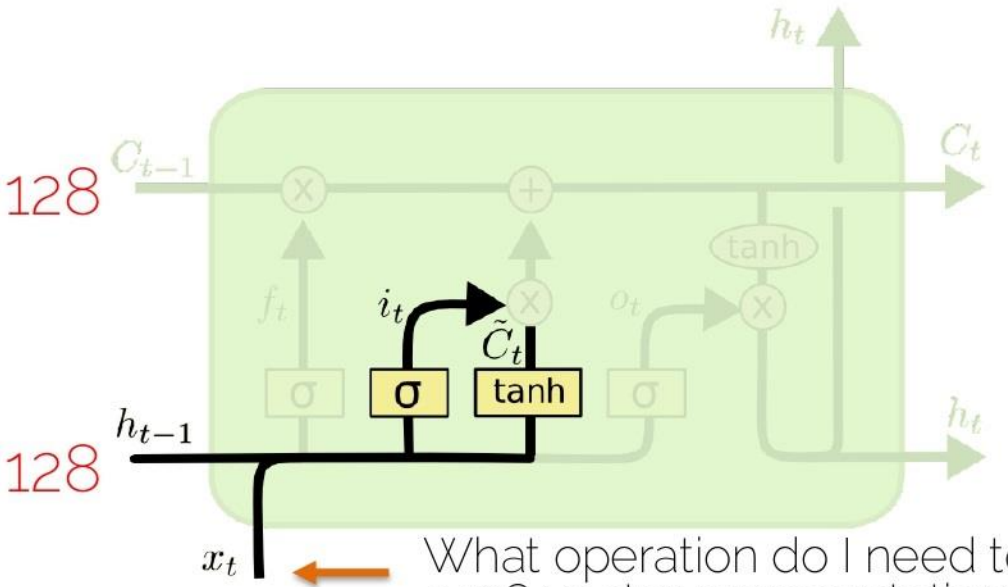
# LSTM

- Highway for the gradient to flow



# LSTM: Dimensions

- Cell update $\overset{128}{\boldsymbol{g}_t} = tanh(\boldsymbol{\theta}_{xg}\overset{128}{\boldsymbol{x}_t} + \boldsymbol{\theta}_{hg}\overset{128}{\boldsymbol{h}_{t-1}} + \boldsymbol{b}_g)$




When coding an LSTM, we have to define the size of the hidden state

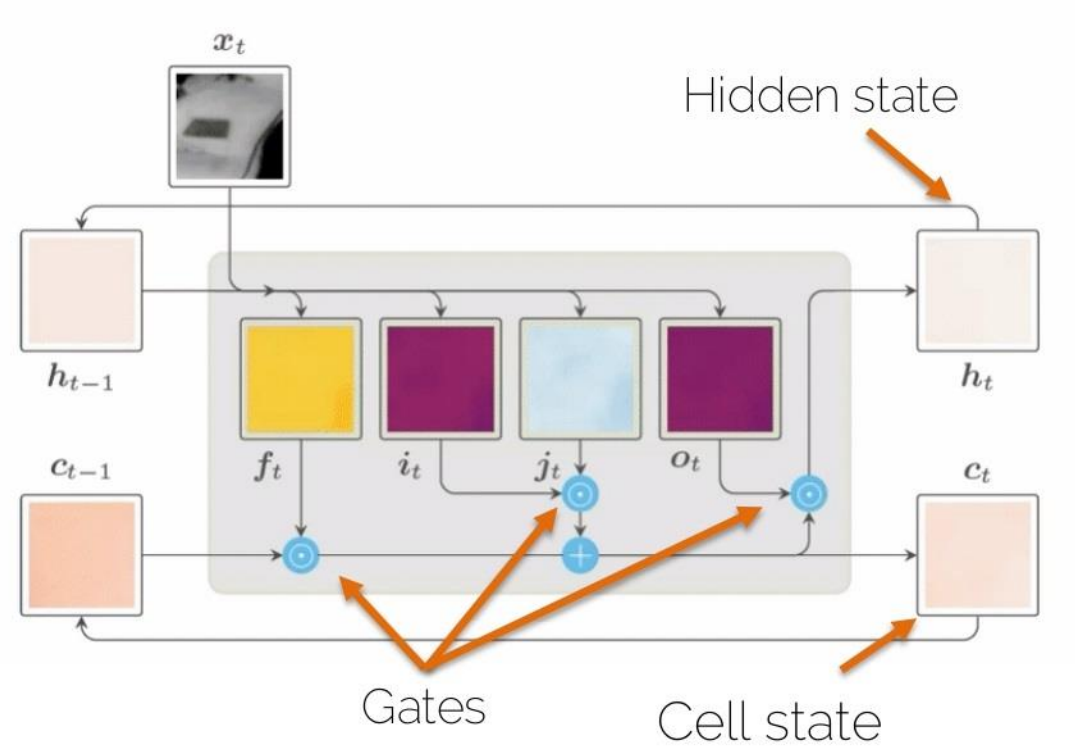
Dimensions need to match

What operation do I need to do to my input to get a 128 vector representation?

# General LSTM Units

- Input, states, and gates not limited to 1st-order tensors
- Gate functions can consist of FC and CNN layers

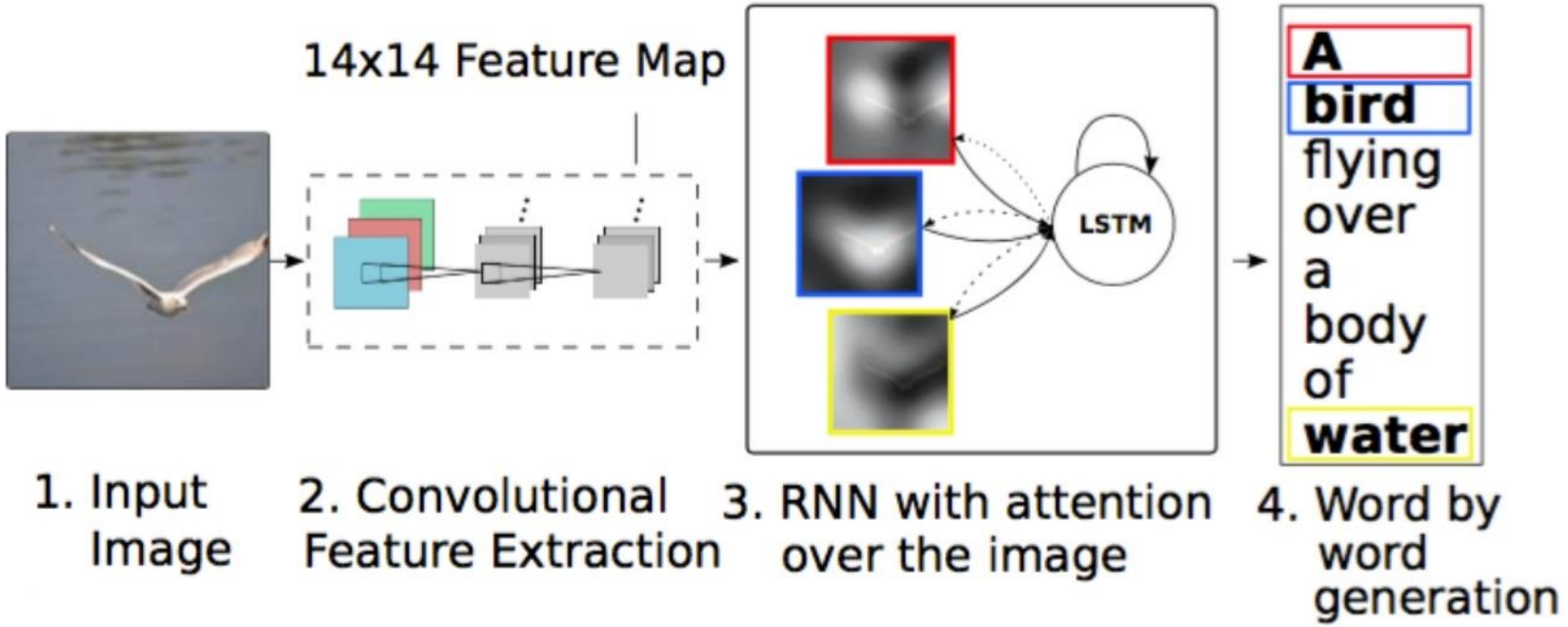
# ConvLSTM for Video Sequences




- Input, hidden, and cell states are higher order tensors (i.e. images)
- Gates have CNN instead of FC layers

# RNNs in Computer Vision

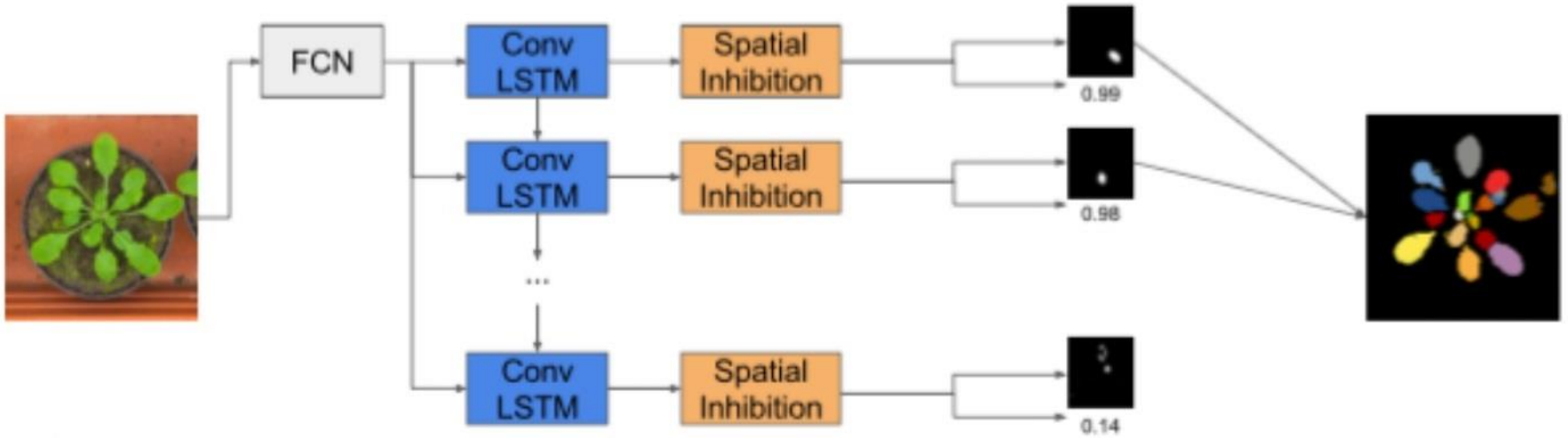
- Caption generation




[Xu et al., PMLR'15] Neural Image Caption Generation

# RNNs in Computer Vision

- Instance segmentation

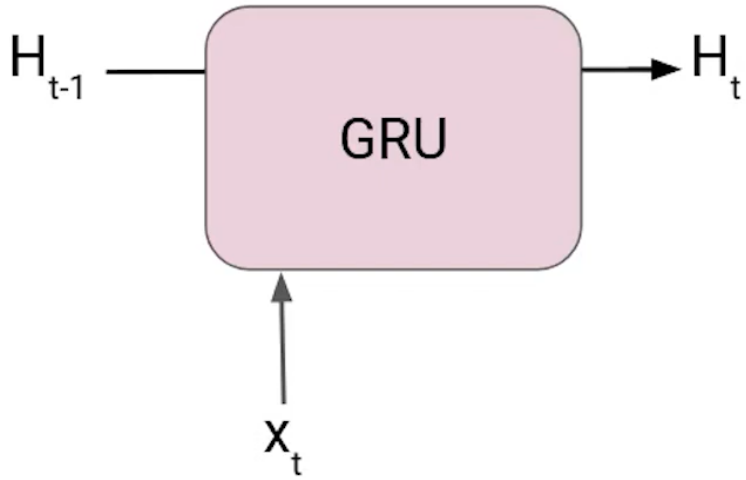



[Romera-Paredes et al., ECCV'16] Recurrent Instance Segmentation

# GRU

Gated Recurrent Unit:

- Proposed in 2014
- Simpler alternative to LSTM (fewer gates)
- No separate cell state ($C_t$)
- Faster training



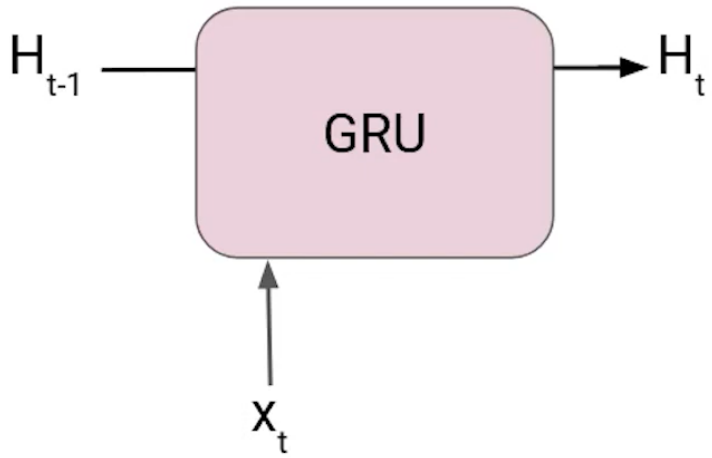
# GRU

- Reset Gate (short-term memory)

  $r_t = \sigma (x_t * U_r + H_{t-1} * W_r)$

- Update Gate (long-term memory)

  $u_t = \sigma (x_t * U_u + H_{t-1} * W_u)$



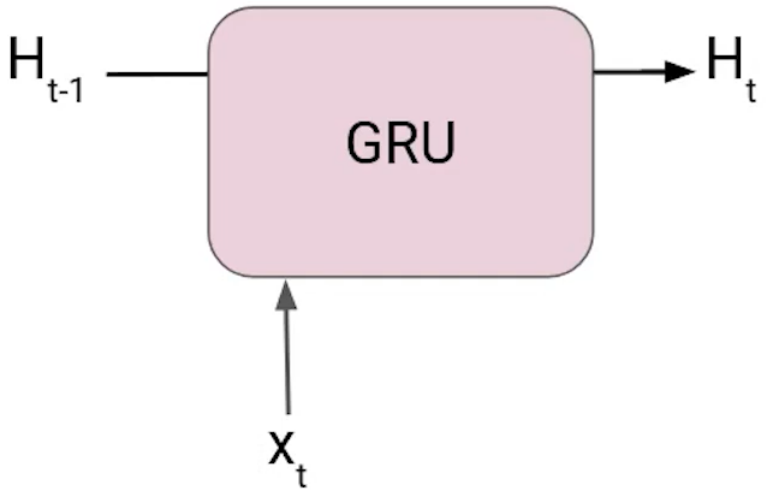
# GRU

- Find candidate hidden state using new info

$$\hat{H}_t = \tanh( x_t * U_g + (r_t \circ H_{t-1}) * W_g )$$

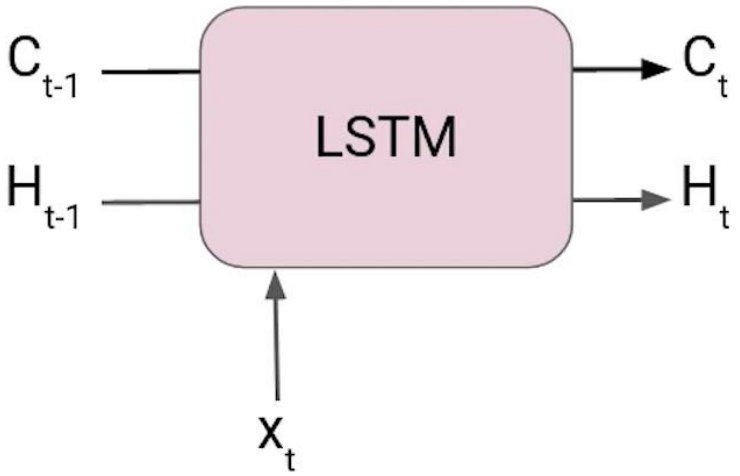
- Update the hidden state using the update gate

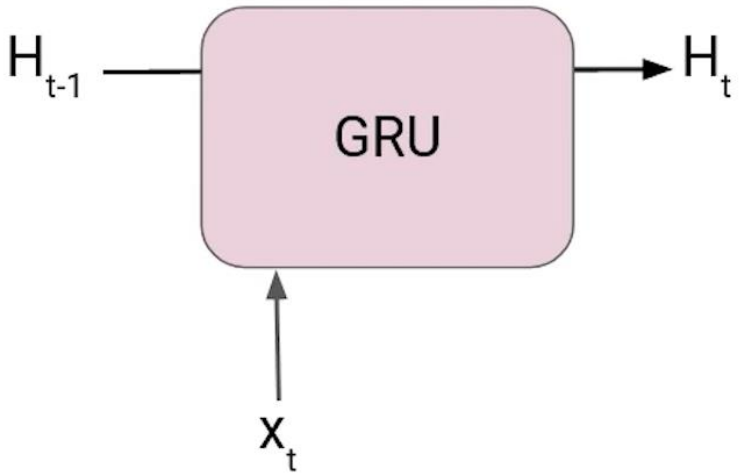
$$H_t = u_t \circ H_{t-1} + (1-u_t) \circ \hat{H}_t$$




In case, you are interested to know more about GRU I suggest you read this Paper.

# GRU




- 3 Gates
- Input, Output, and Forget
- Separate cell state $C_t$ and Hidden state $H_t$

- 2 Gates
- Update and Reset
- Single Hidden state $H_t$