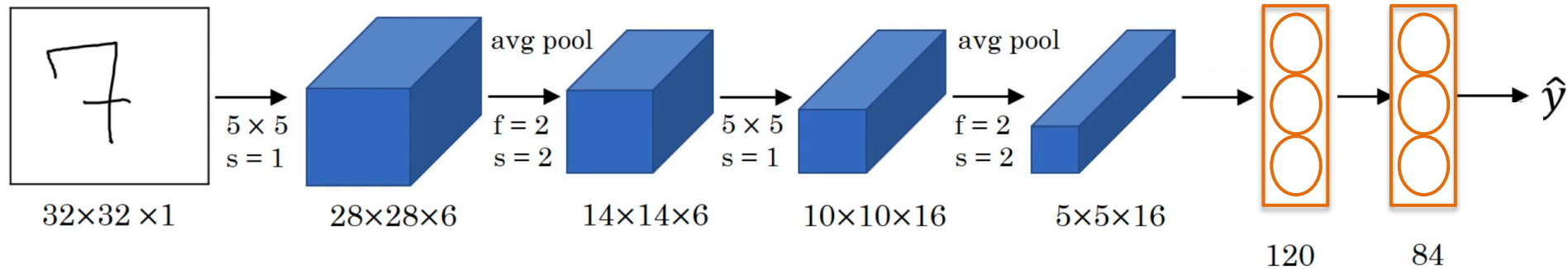


Lecture 10 Recap

LeNet

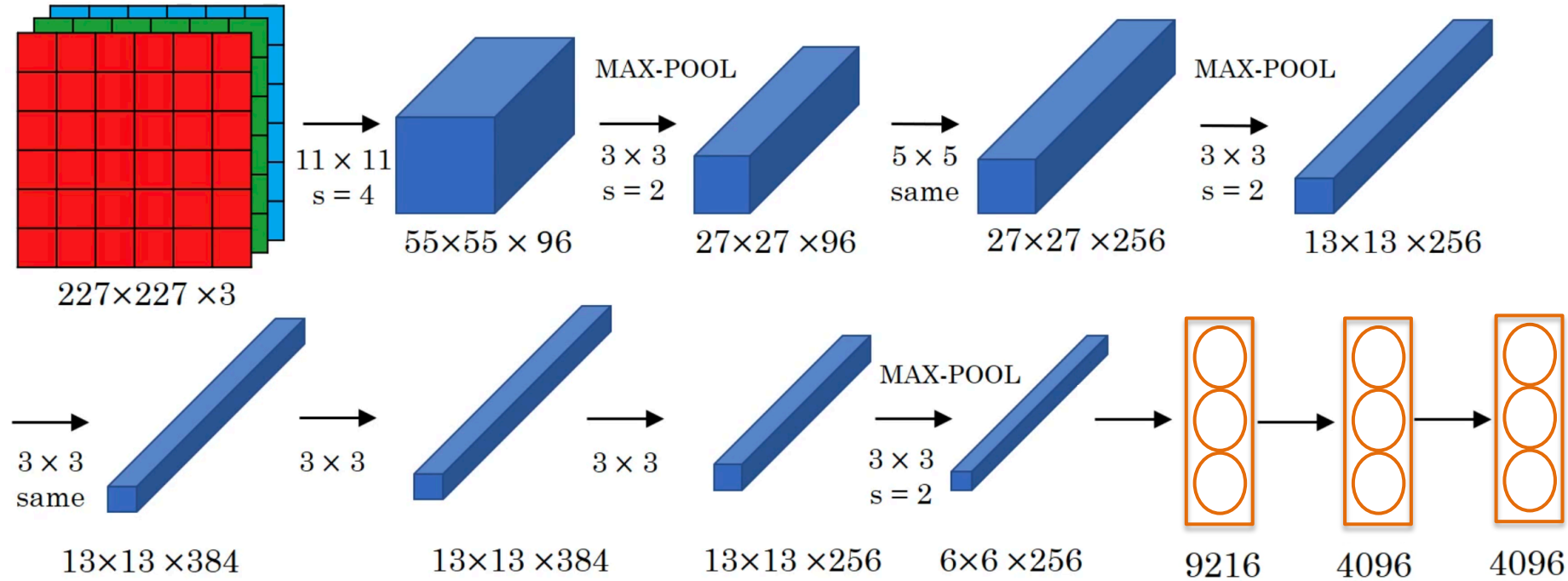
- Digit recognition: 10 classes

60k parameters



- Conv \rightarrow Pool \rightarrow Conv \rightarrow Pool \rightarrow Conv \rightarrow FC
- As we go deeper: Width, Height \downarrow Number of Filters \uparrow



AlexNet



- Softmax for 1000 classes

[Krizhevsky et al., ANIPS'12] AlexNet

VGGNet

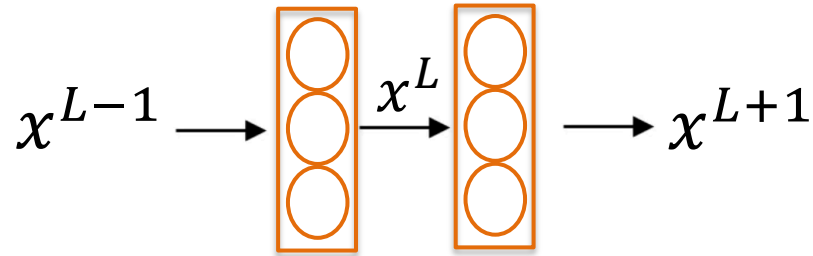
- Striving for **simplicity**
 - Conv \rightarrow Pool \rightarrow Conv \rightarrow Pool \rightarrow Conv \rightarrow FC
 - Conv=3x3, s=1, same; Maxpool=2x2, s=2
- As we go deeper: Width, Height  Number of Filters 
- Called VGG-16: 16 layers that have weights

138M parameters
- Large but simplicity makes it appealing

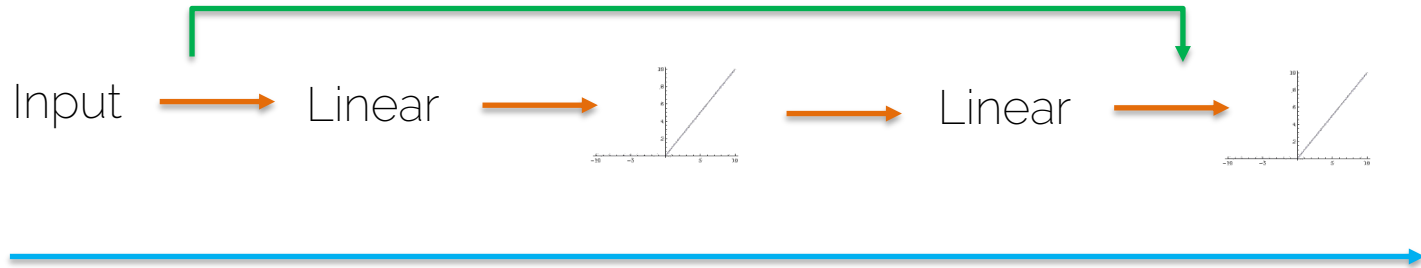
[Simonyan et al., ICLR'15] VGGNet

Residual Block

- Two layers

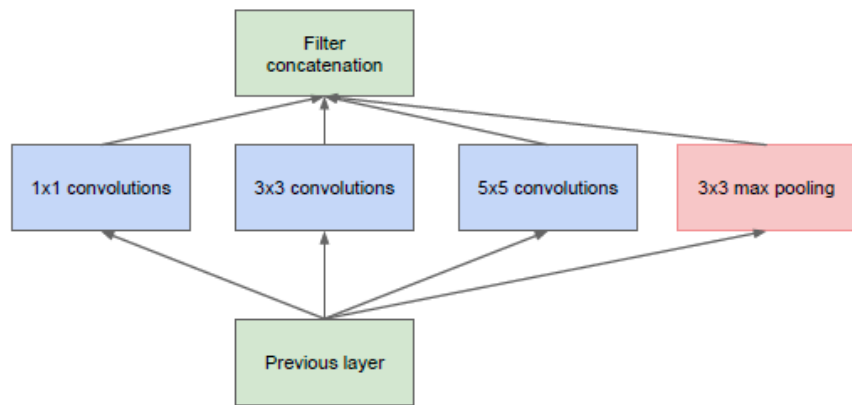


$$x^{L+1} = f(W^{L+1}x^L + b^{L+1} + x^{L-1})$$

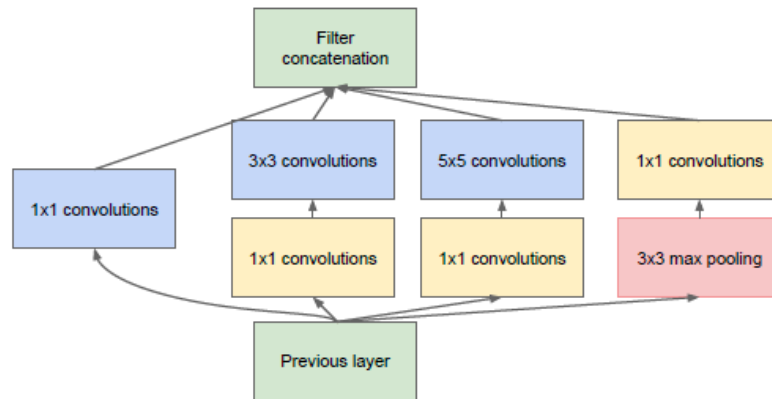


$$x^{L+1} = f(W^{L+1}x^L + b^{L+1})$$

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?

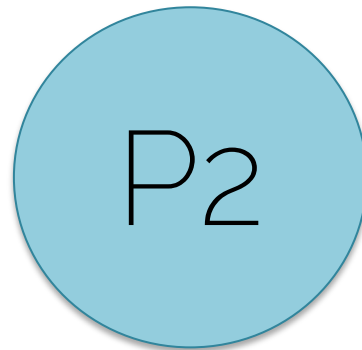
Transfer Learning

Distribution



Large dataset

Distribution



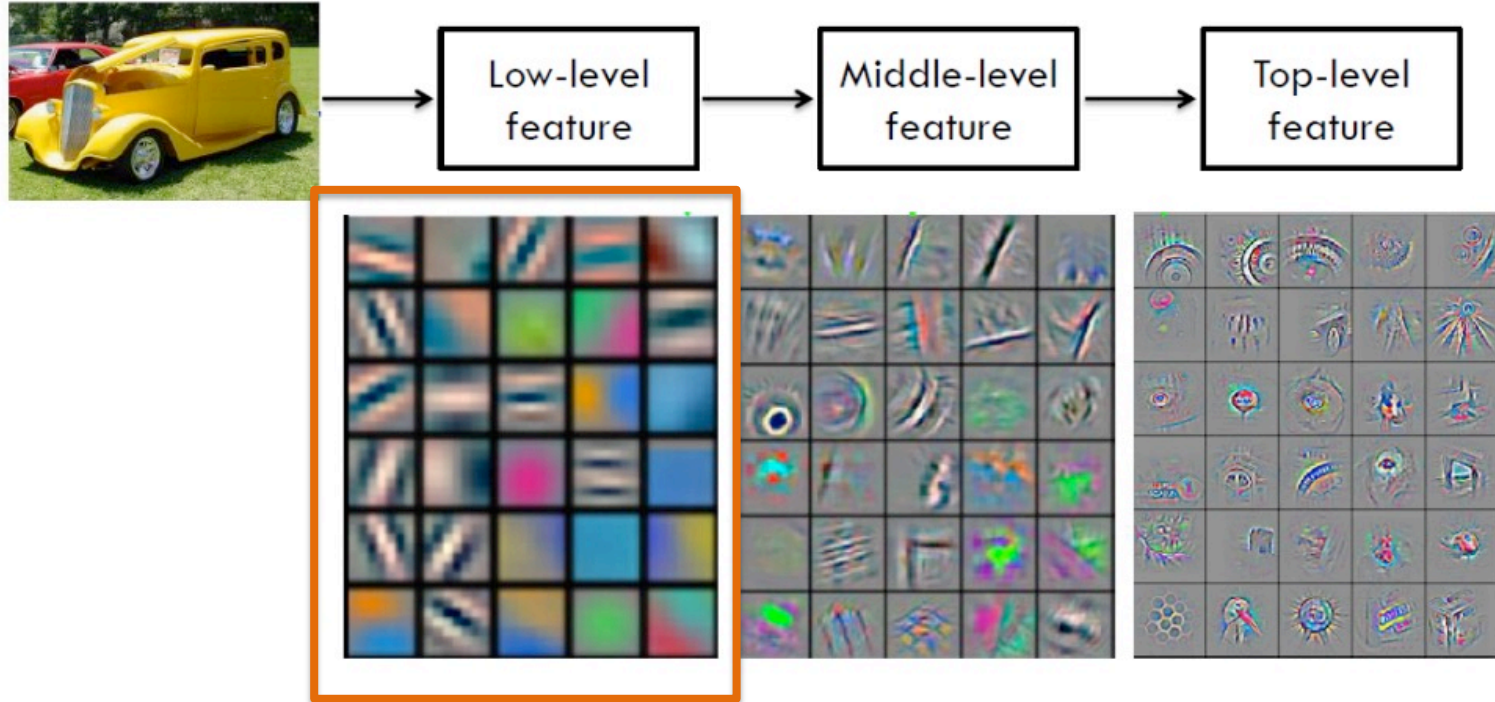
Small dataset



Use what has been
learned for another
setting



Transfer Learning for Images



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

Trained on
ImageNet

Transfer Learning

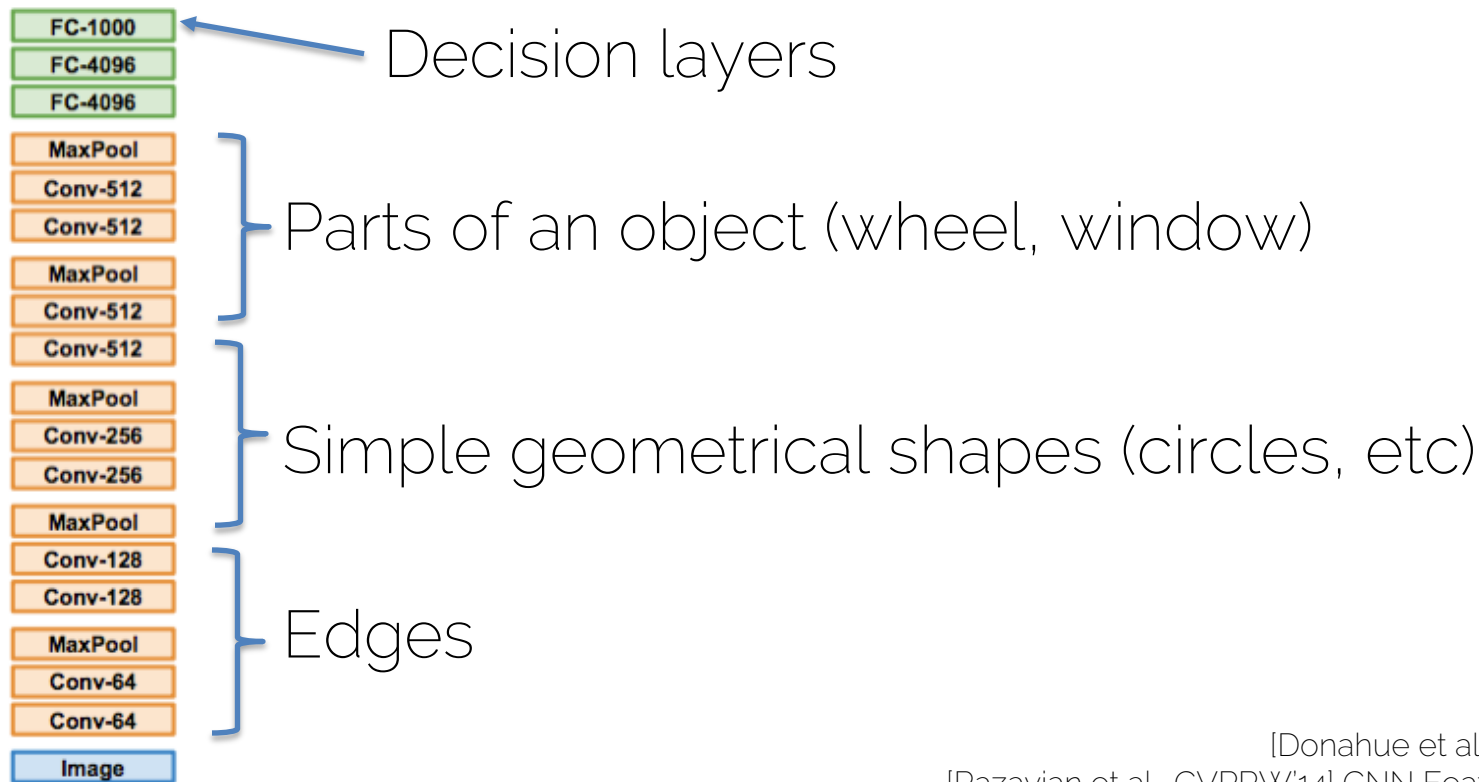


Feature
extraction

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

Trained on
ImageNet

Transfer Learning



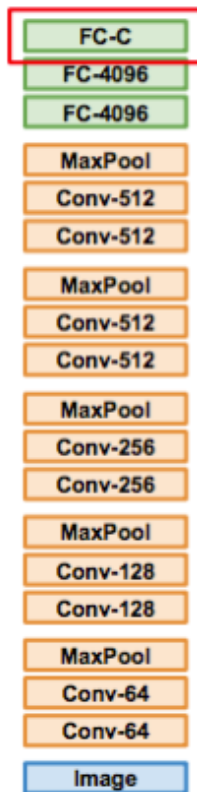
[Donahue et al., ICML'14] DeCAF,
[Razavian et al., CVPRW'14] CNN Features off-the-shelf

Trained on
ImageNet

Transfer Learning



TRAIN



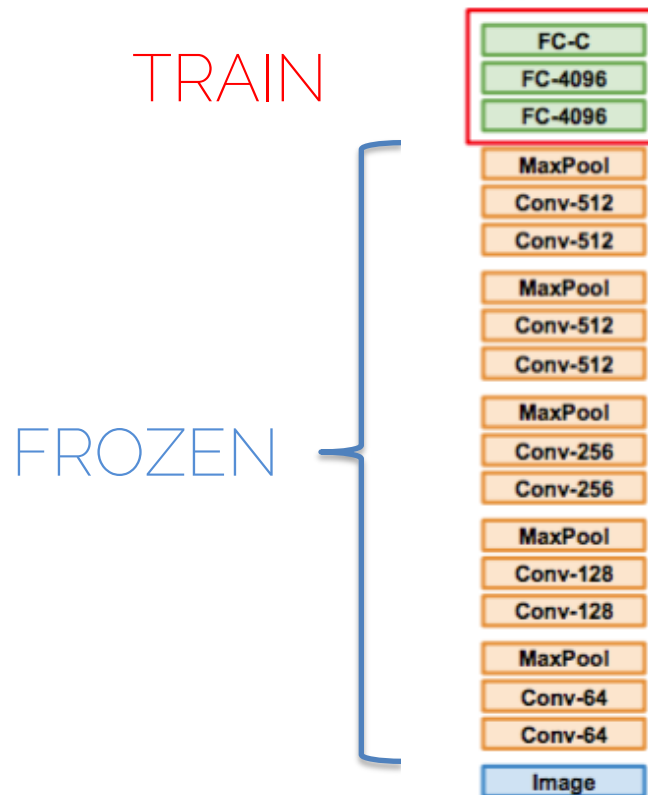
New dataset
with C classes

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 T_1 and T_2 have the same input (e.g. an RGB image)
- When you have more data for task T_1 than for task T_2
- When the low-level features for T_1 could be useful to learn T_2

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)

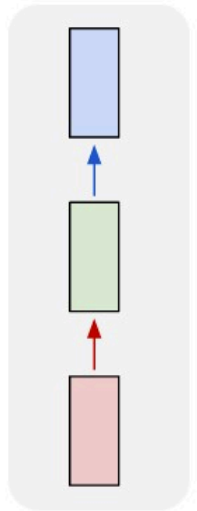
Recurrent Neural Networks

Processing Sequences

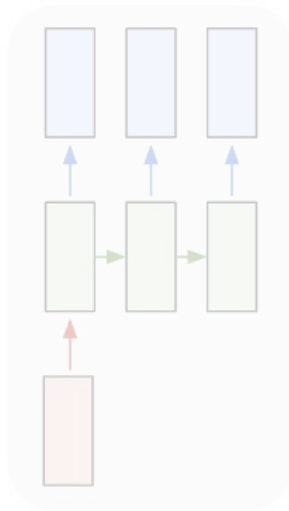
- Recurrent neural networks process sequence data
- Input/output can be sequences

RNNs are Flexible

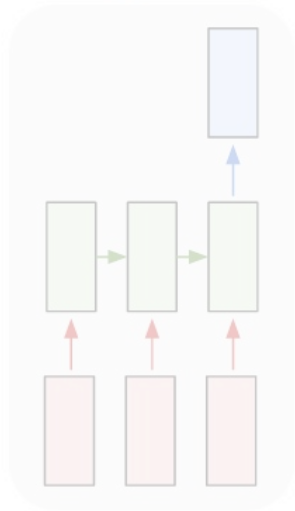
one to one



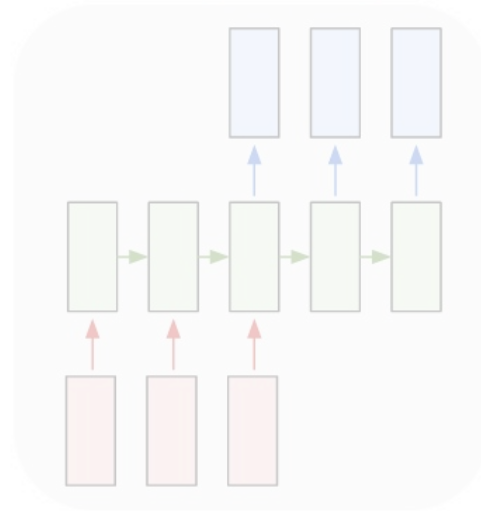
one to many



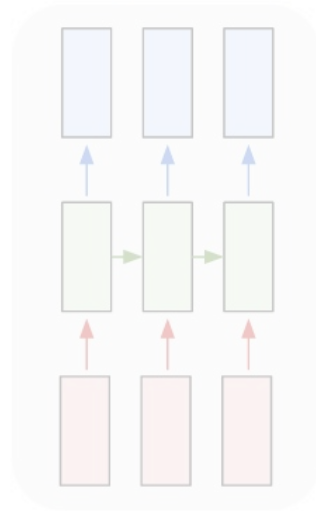
many to one



many to many



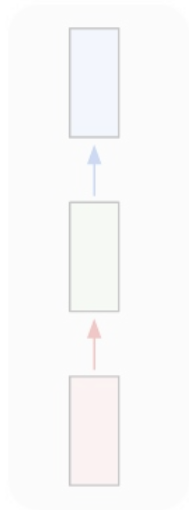
many to many



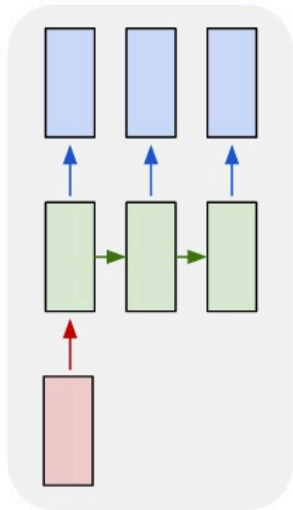
Classic Neural Networks for Image Classification

RNNs are Flexible

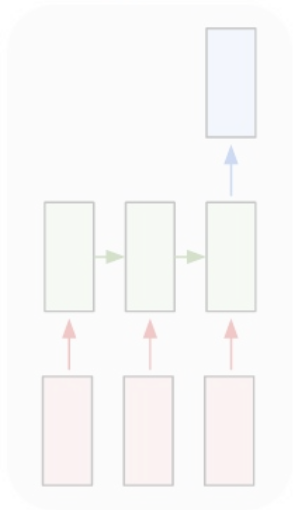
one to one



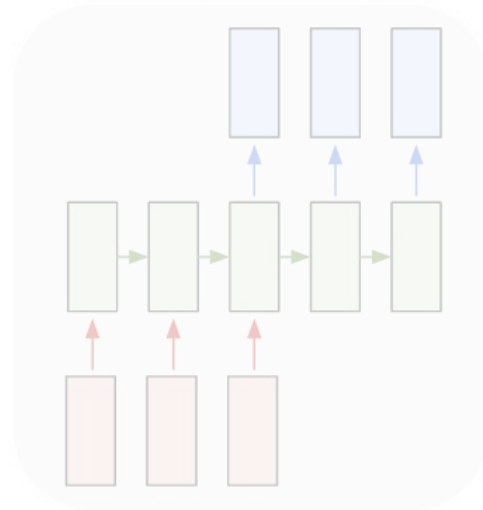
one to many



many to one



many to many



many to many

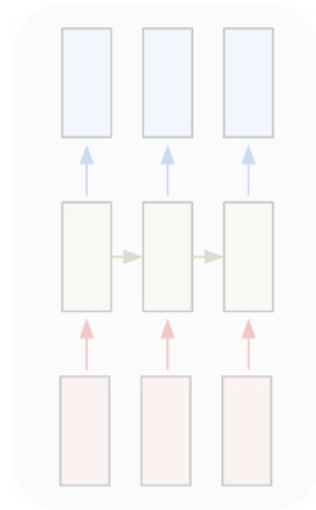
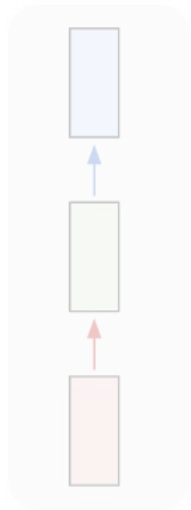


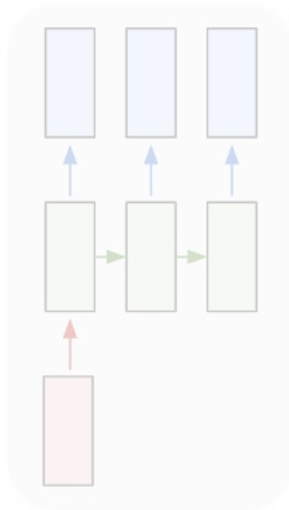
Image captioning

RNNs are Flexible

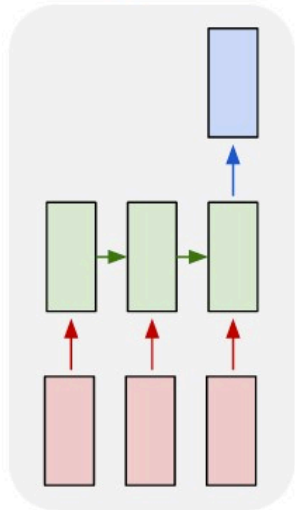
one to one



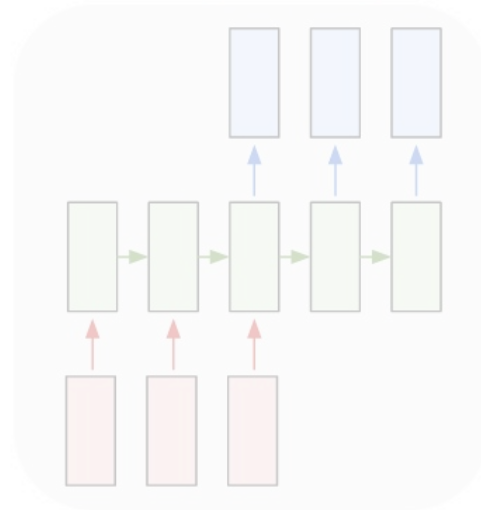
one to many



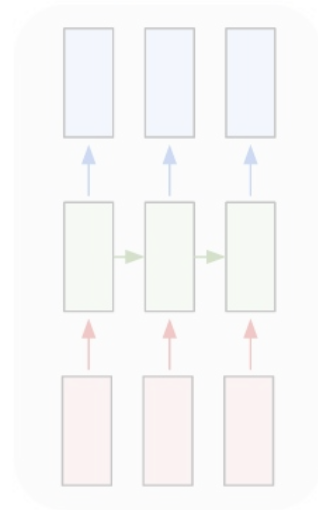
many to one



many to many



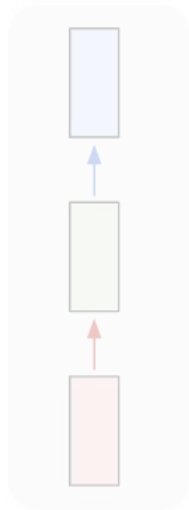
many to many



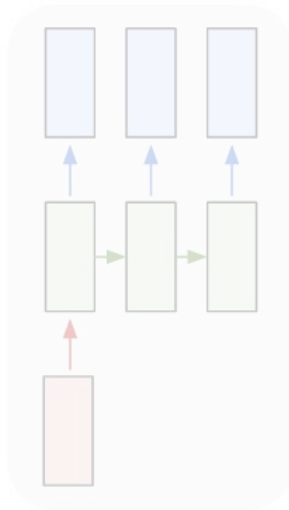
Language recognition

RNNs are Flexible

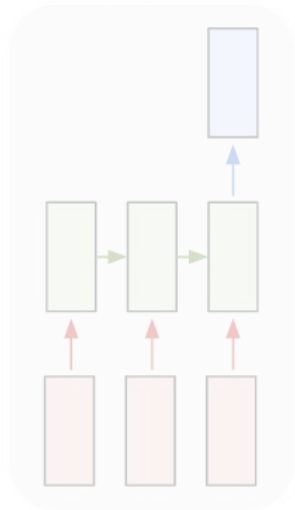
one to one



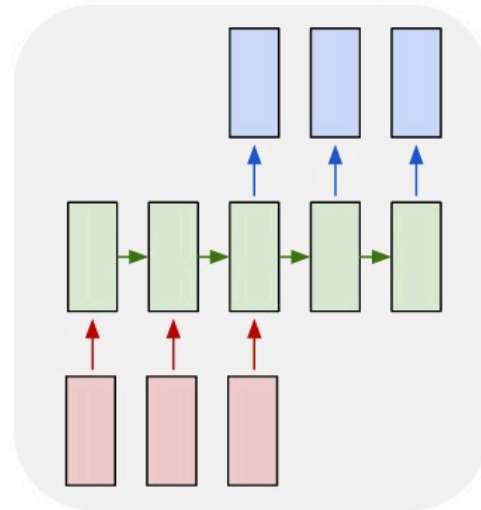
one to many



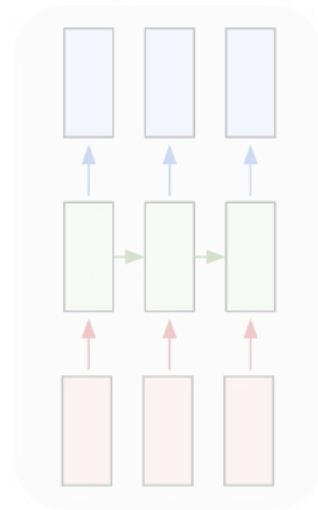
many to one



many to many



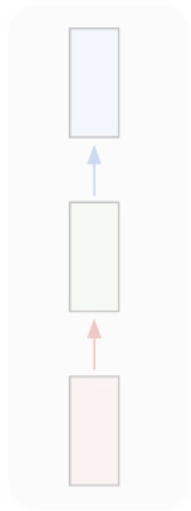
many to many



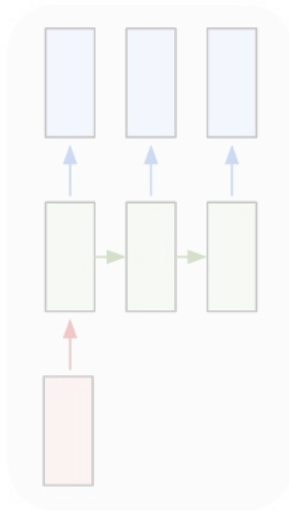
Machine translation

RNNs are Flexible

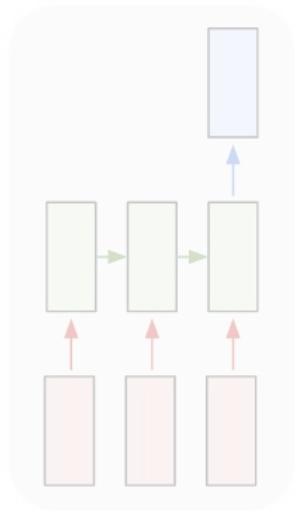
one to one



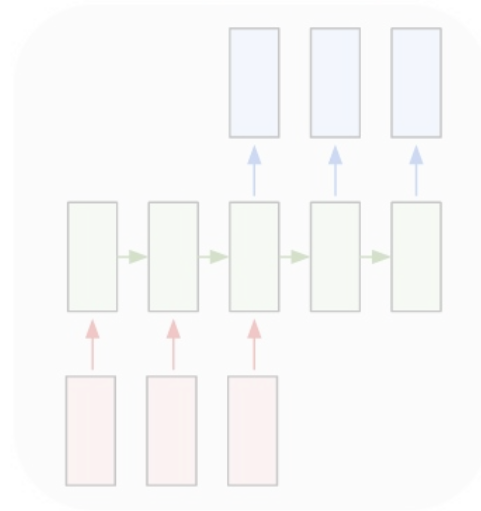
one to many



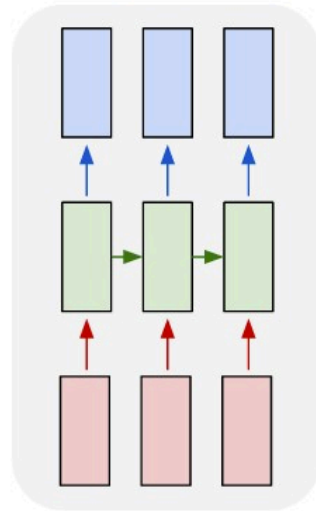
many to one



many to many



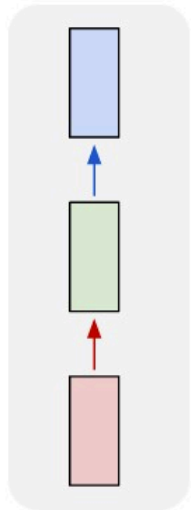
many to many



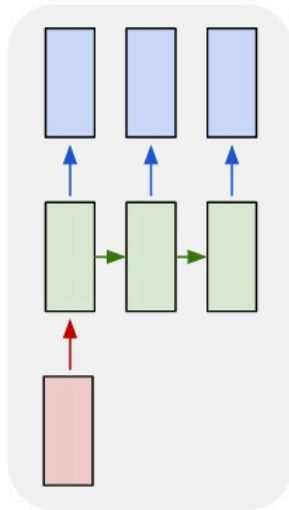
Event classification

RNNs are Flexible

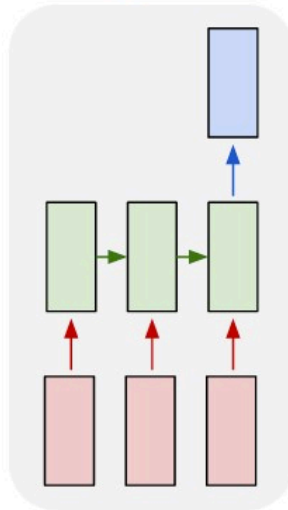
one to one



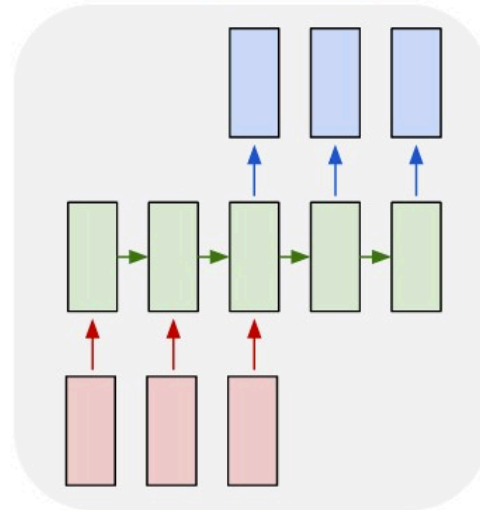
one to many



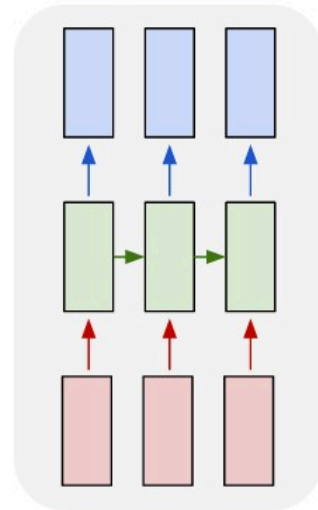
many to one



many to many



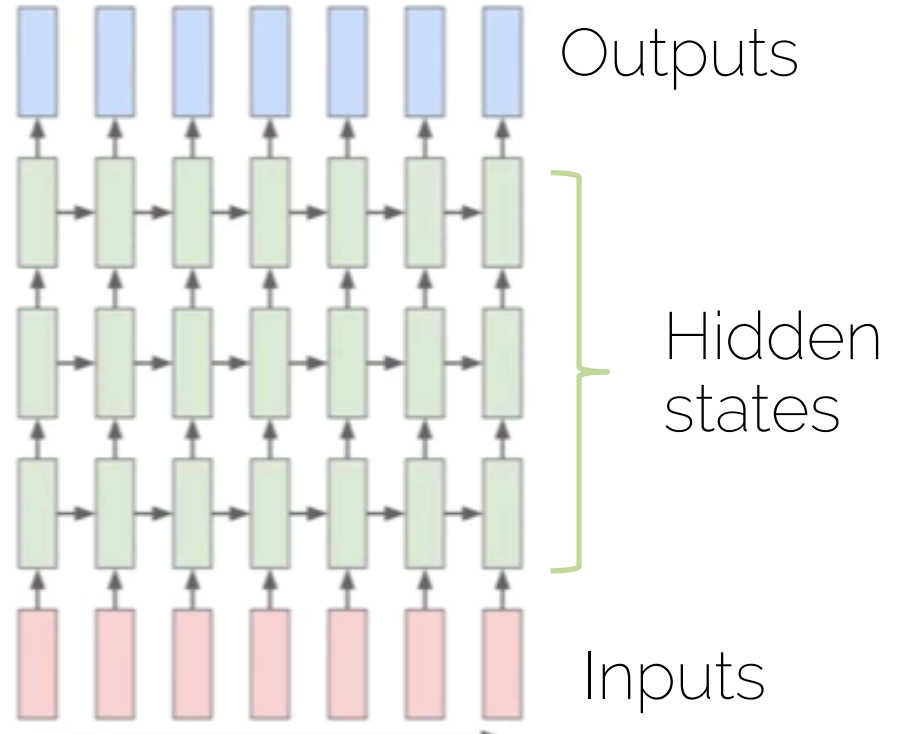
many to many



Event classification

Basic Structure of an RNN

- Multi-layer RNN



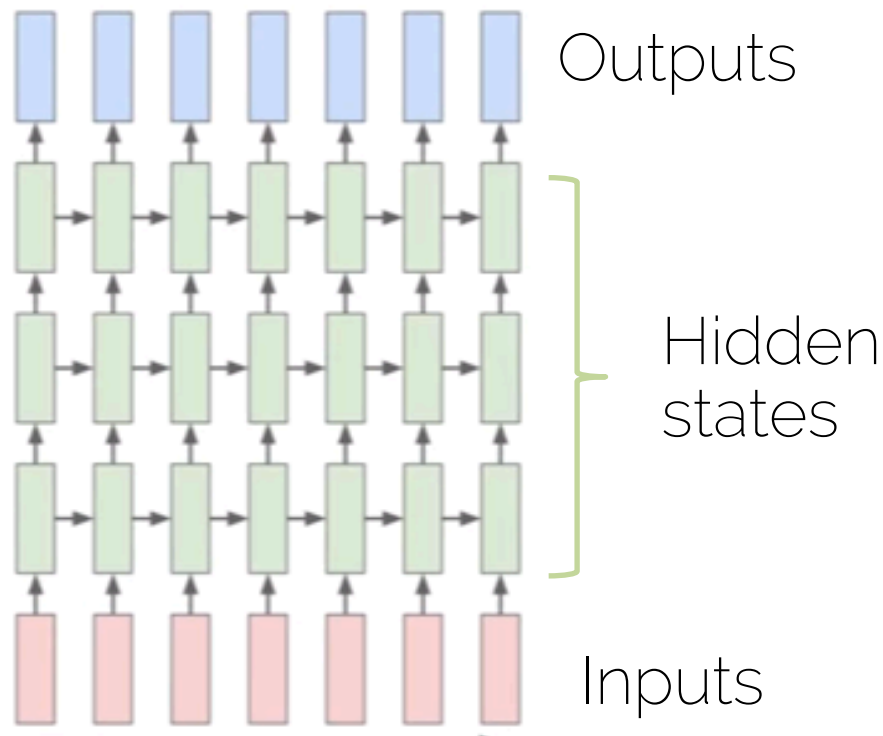
Basic Structure of an RNN

- Multi-layer RNN

The hidden state
will have its own
internal dynamics

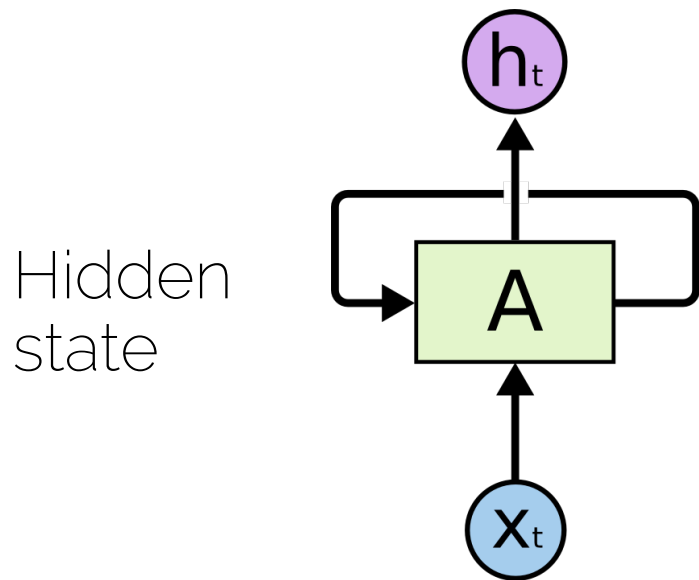


More expressive
model!



Basic Structure of an RNN

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



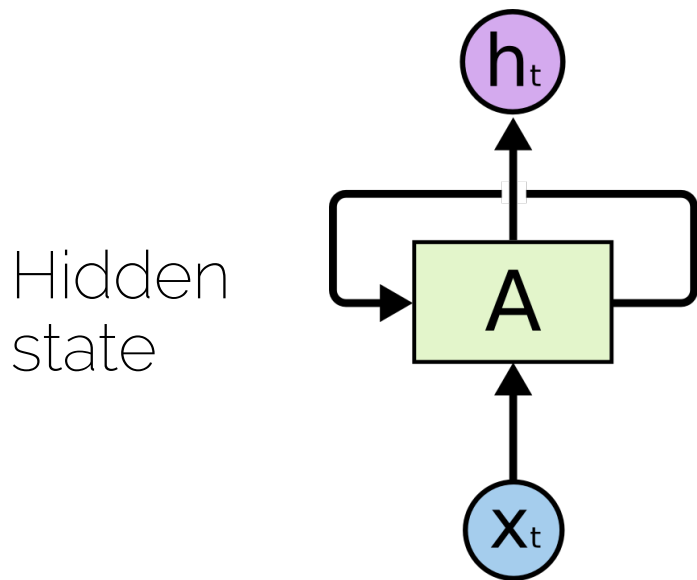
$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

Previous
hidden
state

input

Basic Structure of an RNN

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

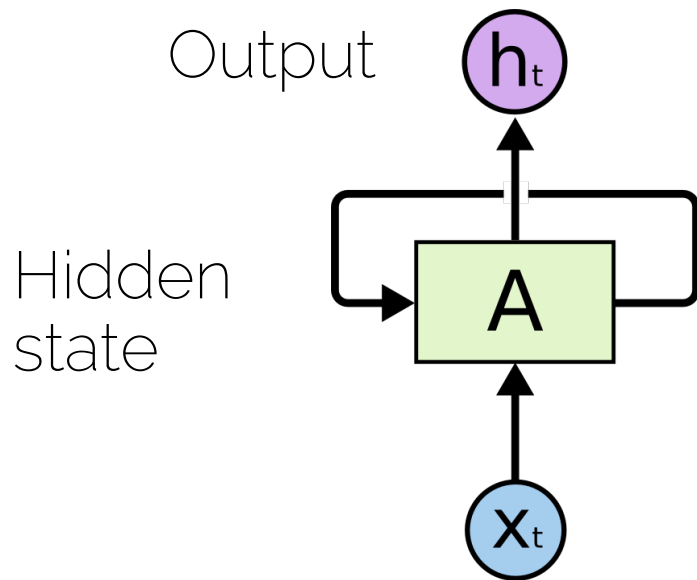


$$\mathbf{A}_t = \boldsymbol{\theta}_c \mathbf{A}_{t-1} + \boldsymbol{\theta}_x \mathbf{x}_t$$

Parameters to be learned

Basic Structure of an RNN

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



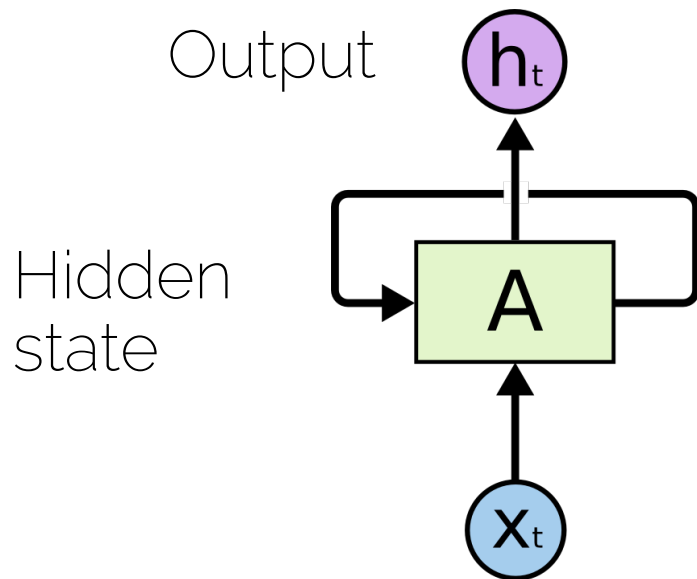
$$A_t = \theta_c A_{t-1} + \theta_x x_t$$

$$h_t = \theta_h A_t$$

Note: non-linearities
ignored for now

Basic Structure of an RNN

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



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

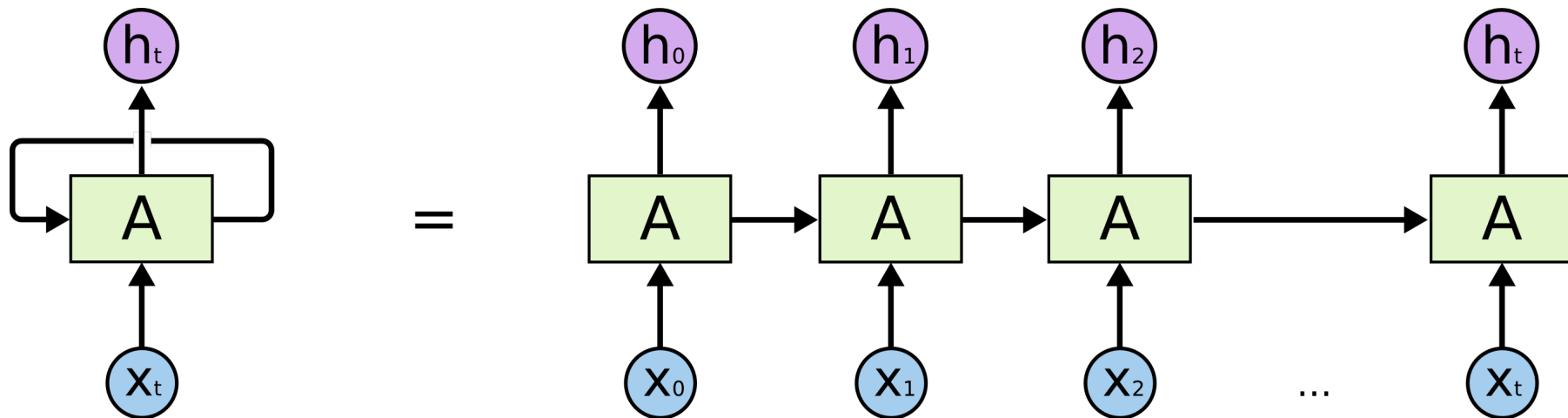
$$h_t = \theta_h A_t$$

Same parameters for each time step = generalization!

Basic Structure of an RNN

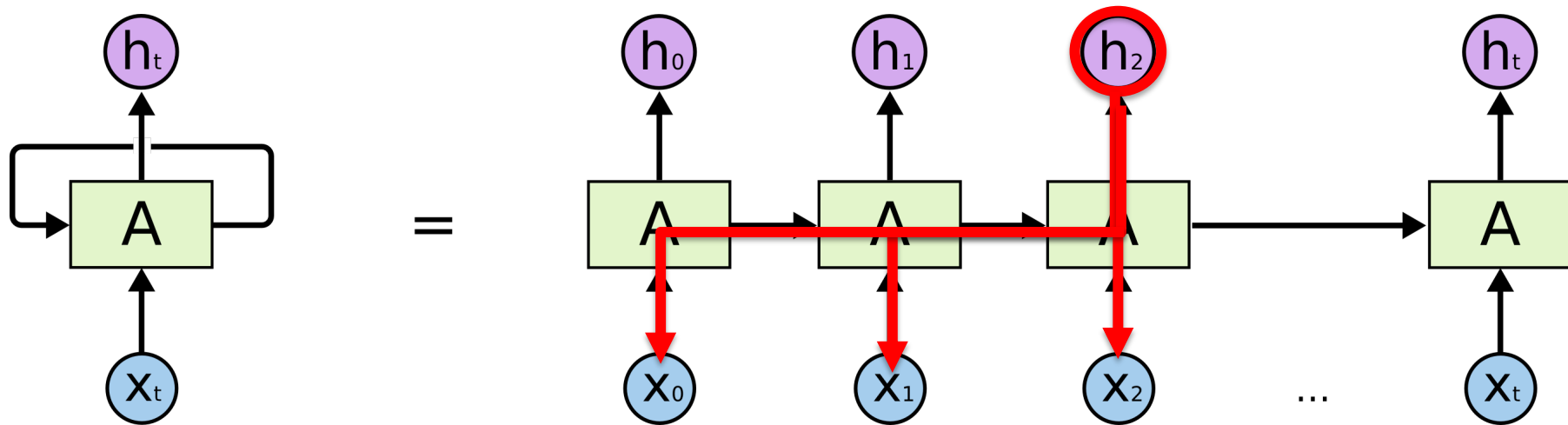
- Unrolling RNNs

Same function for the hidden layers



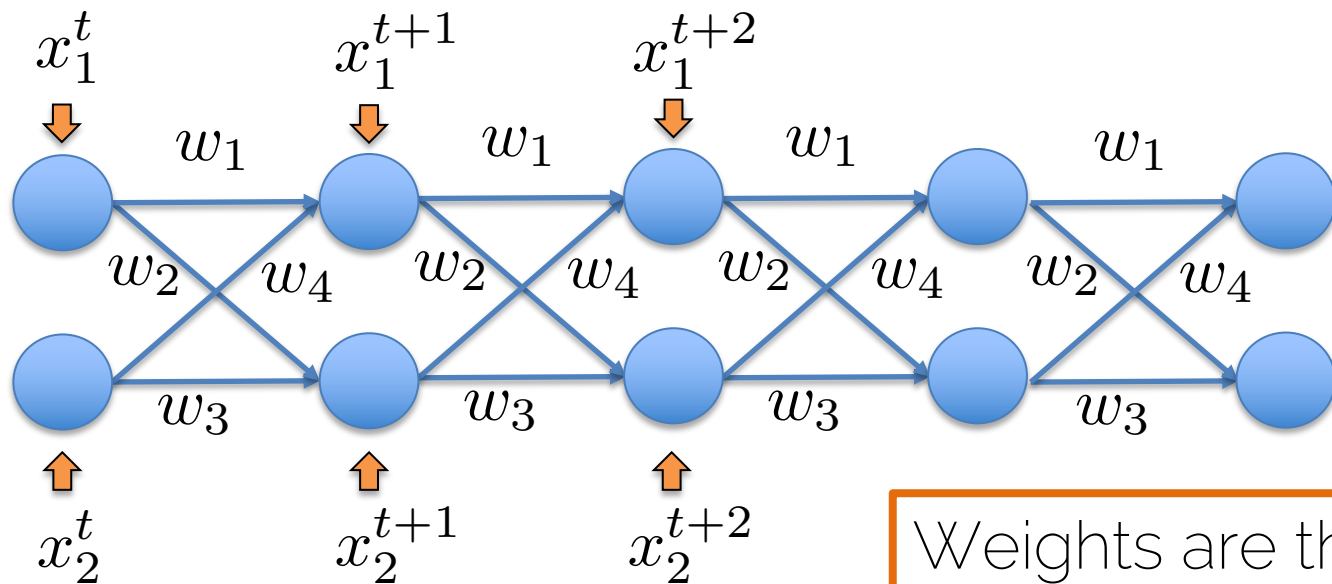
Basic Structure of an RNN

- Unrolling RNNs



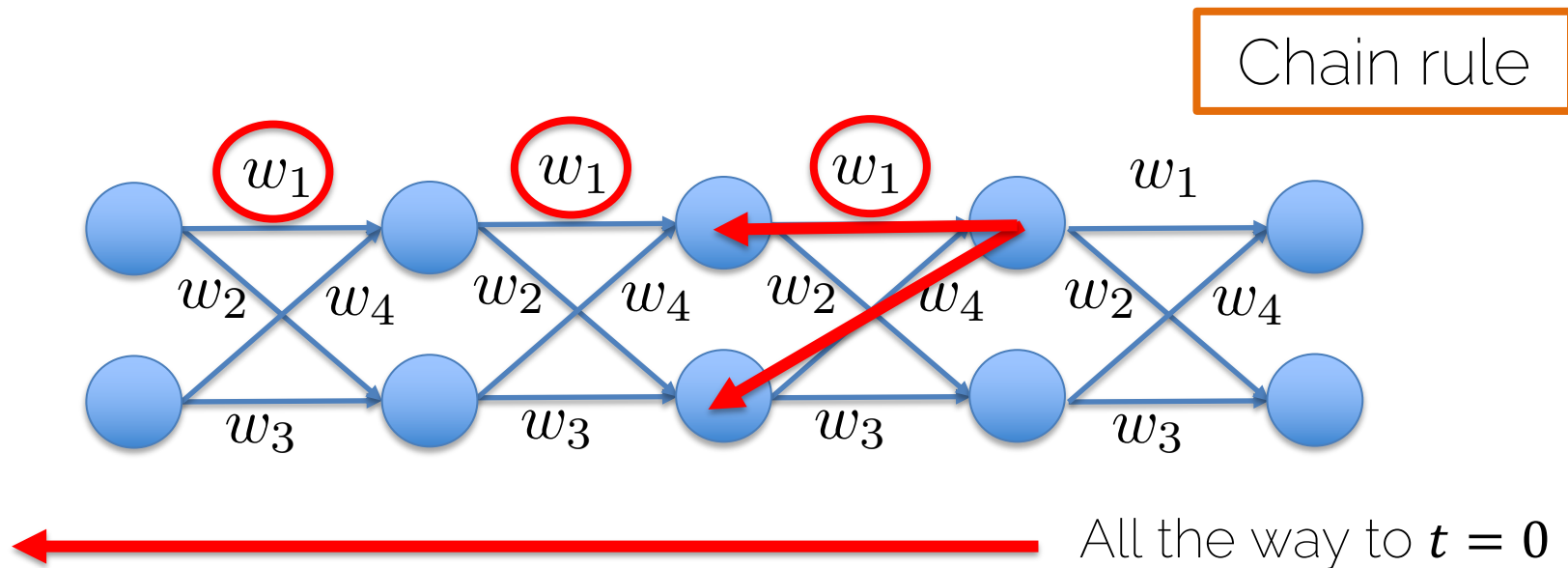
Basic Structure of an RNN

- Unrolling RNNs as feedforward nets



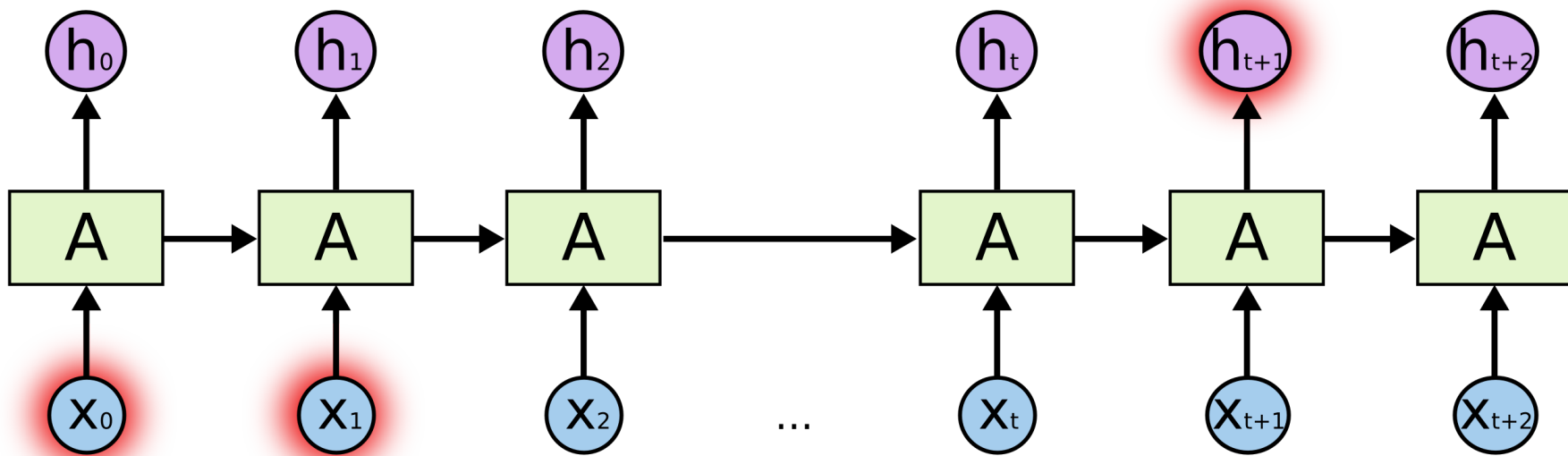
Backprop through an RNN

- Unrolling RNNs as feedforward nets



Add the derivatives at different times for each weight

Long-term Dependencies



I moved to Germany ...

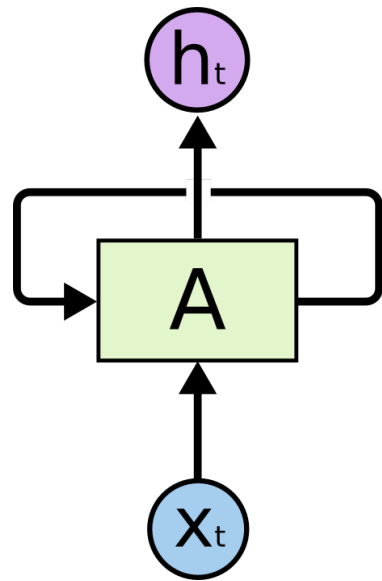
so I speak German fluently.

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 $\mathbf{A}_t = \boldsymbol{\theta}_c^t \mathbf{A}_0$



Same weights are
multiplied over and over
again



Long-term Dependencies

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

What happens to small weights?

Vanishing gradient

What happens to large weights?

Exploding gradient

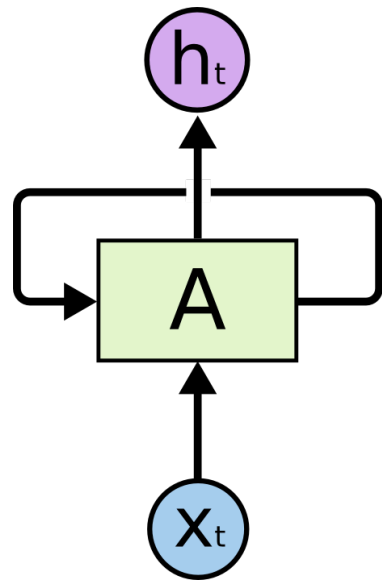
Long-term Dependencies

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

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

Matrix of
eigenvectors

Diagonal of this
matrix are the
eigenvalues



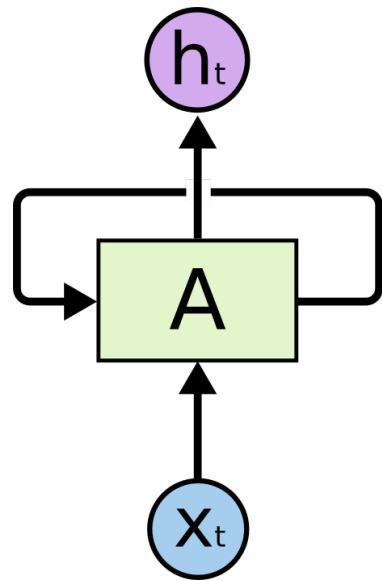
Long-term Dependencies

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

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

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

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



Long-term Dependencies

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

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 $\mathbf{A}_t = \boldsymbol{\theta}_c^t \mathbf{A}_0$

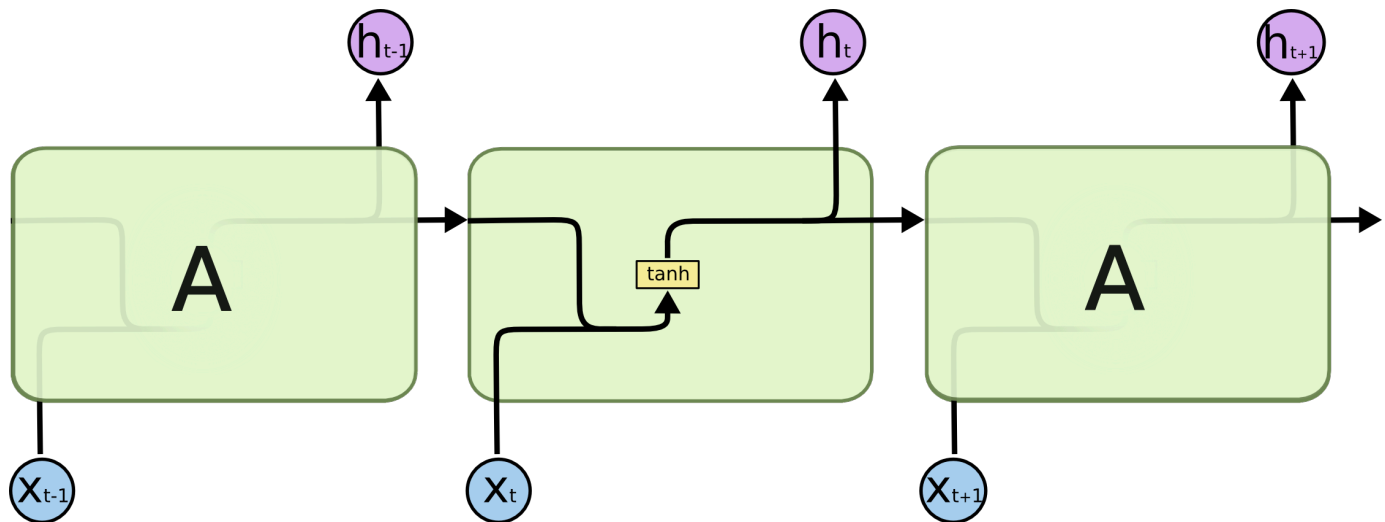
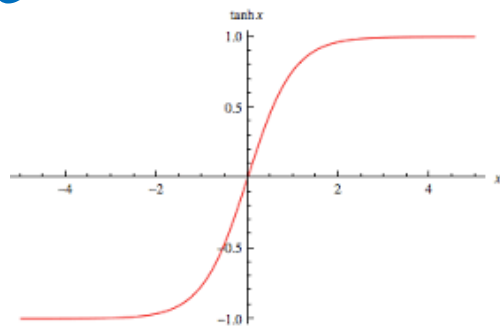


Let us just make a matrix with eigenvalues = 1

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

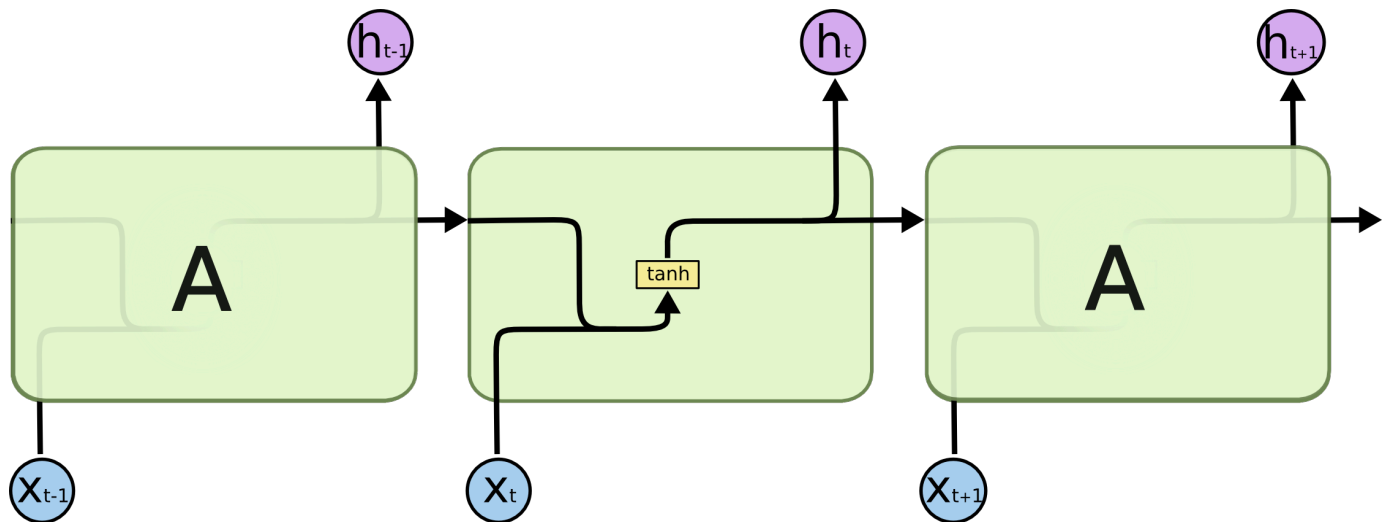
Vanishing Gradient

- 1. From the weights $\mathbf{A}_t = \boldsymbol{\theta}_c^t \mathbf{A}_0$
- 2. From the activation functions (*tanh*)



Vanishing Gradient

- 1. From the weights $\mathbf{A}_t = \cancel{\theta^t} \mathbf{A}_0$
1
- 2. From the activation functions (*tanh*) ?

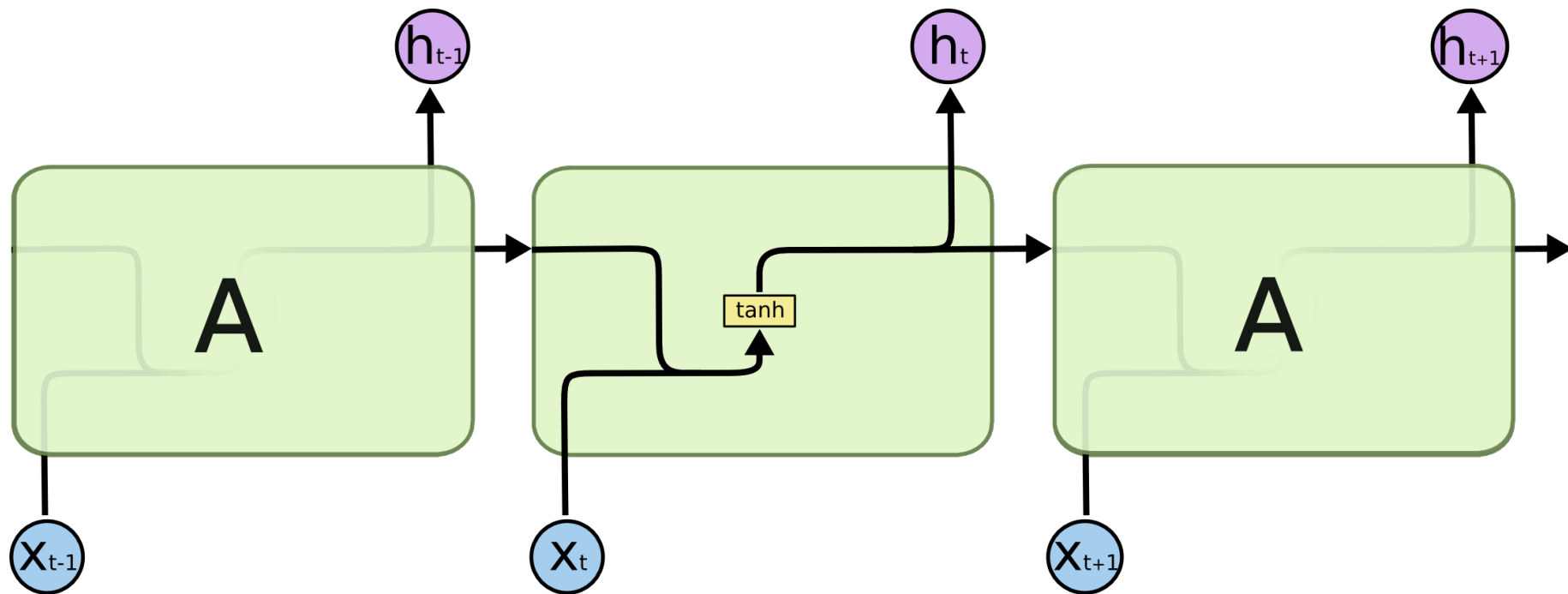


Long Short Term Memory

[Hochreiter et al., Neural Computation'97] Long Short-Term Memory

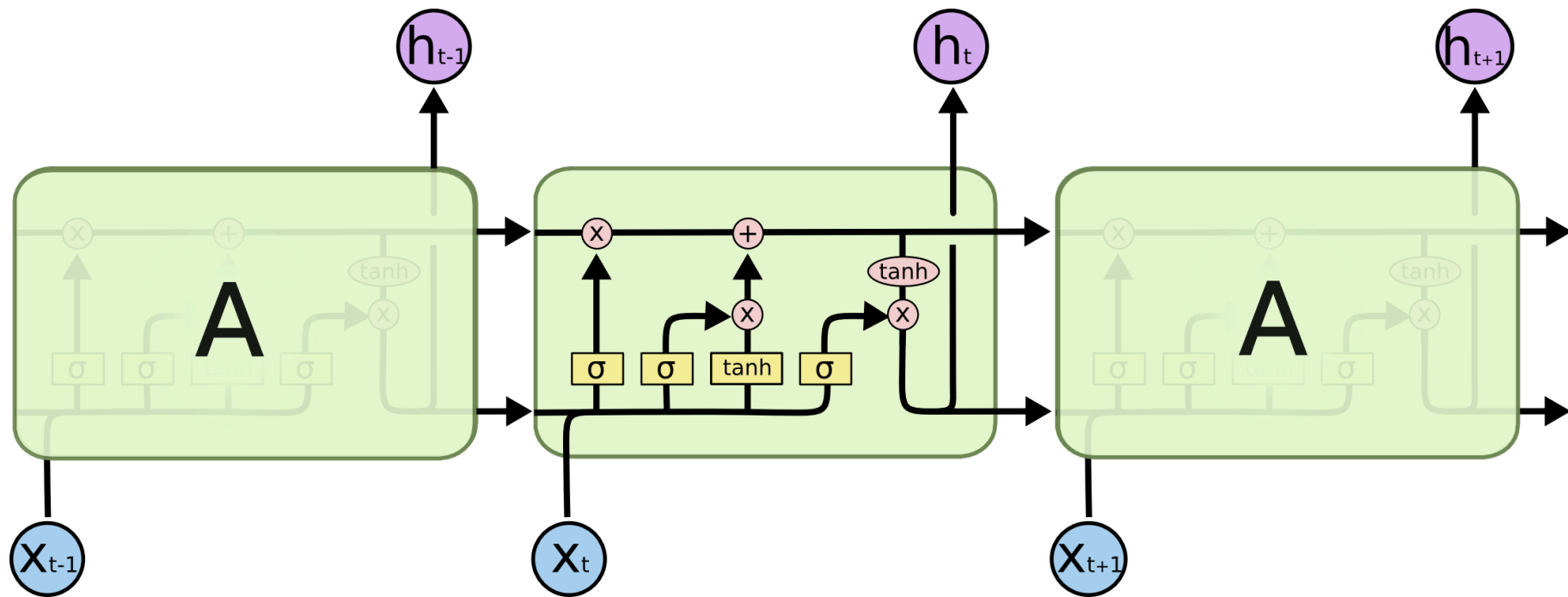
Long-Short Term Memory Units

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



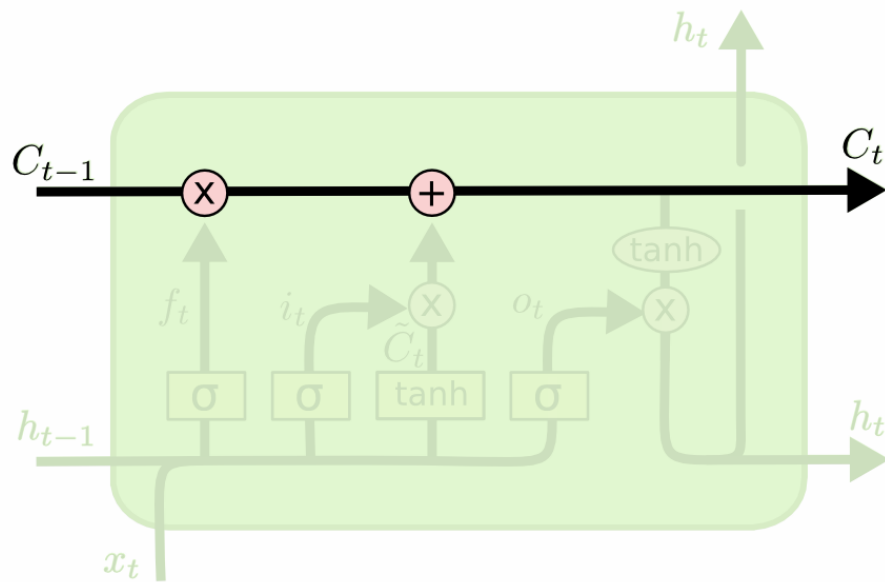
Long-Short Term Memory Units

LSTM



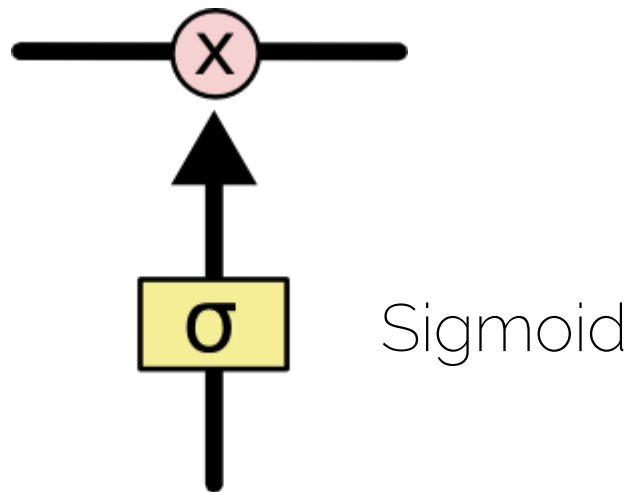
Long-Short Term Memory Units

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



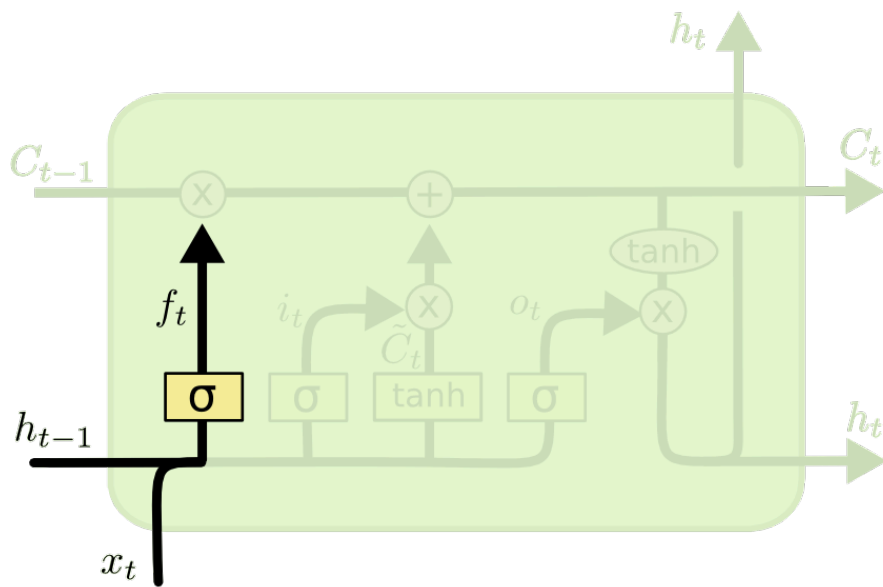
Long-Short Term Memory Units

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



LSTM: Step by Step

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

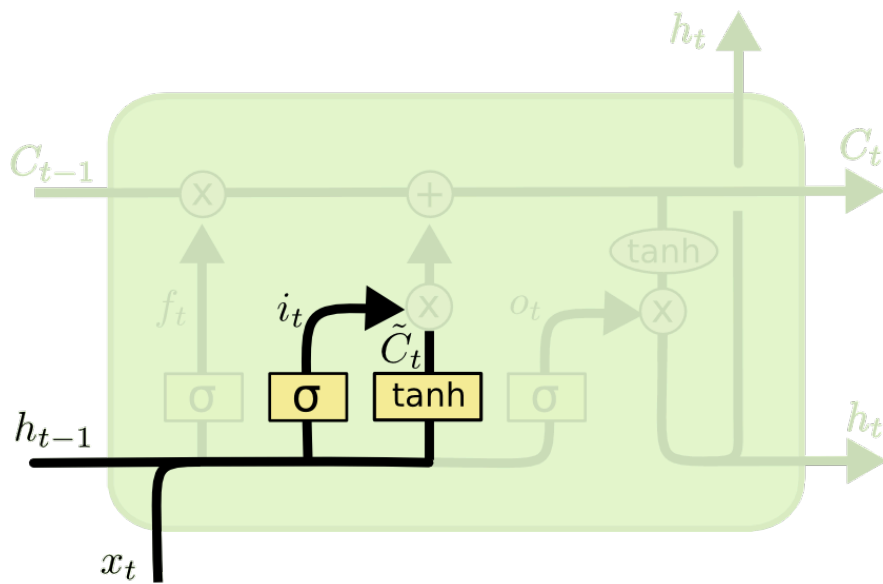


Decides when to erase the cell state

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

LSTM: Step by Step

- Input gate $\mathbf{i}_t = \text{sigm}(\boldsymbol{\theta}_{xi}\mathbf{x}_t + \boldsymbol{\theta}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$

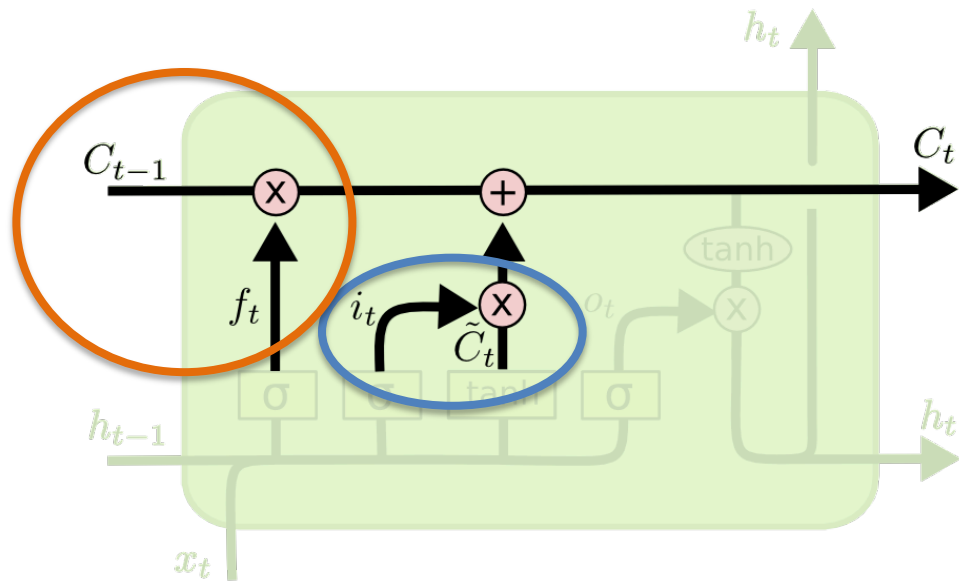


Decides which values will be updated

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

LSTM: Step by Step

- Element-wise operations



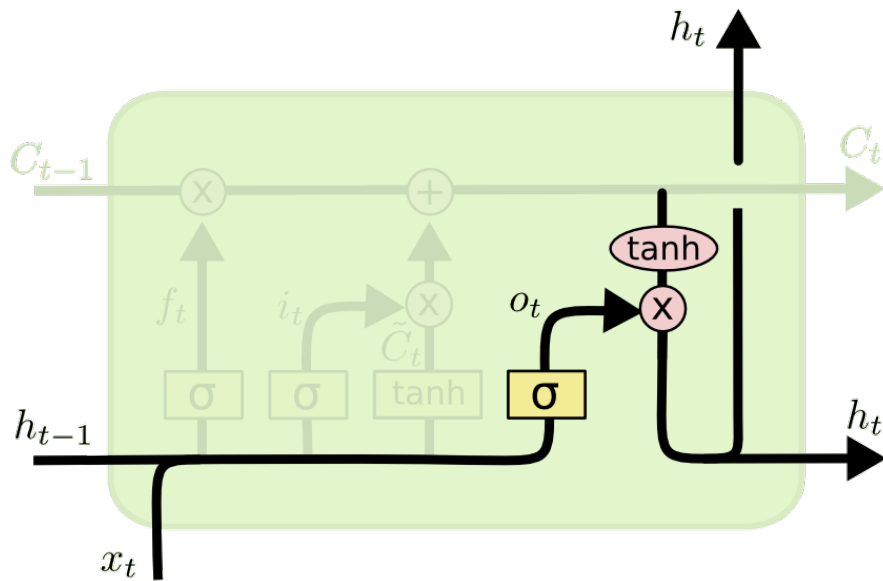
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Previous
states

Current
state

LSTM: Step by Step

- Output gate $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$



Decides which values will be outputted

Output from a $\tanh (-1, 1)$

LSTM: Step by Step

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

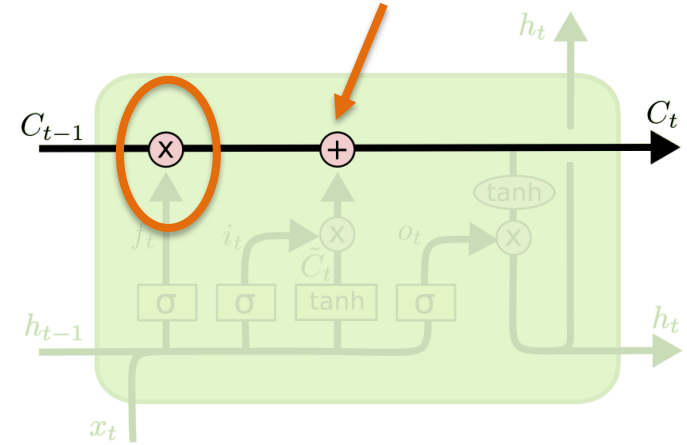
LSTM: Step by Step

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

Learned through
backpropagation

LSTM: Vanishing Gradients?

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



1 for important
information

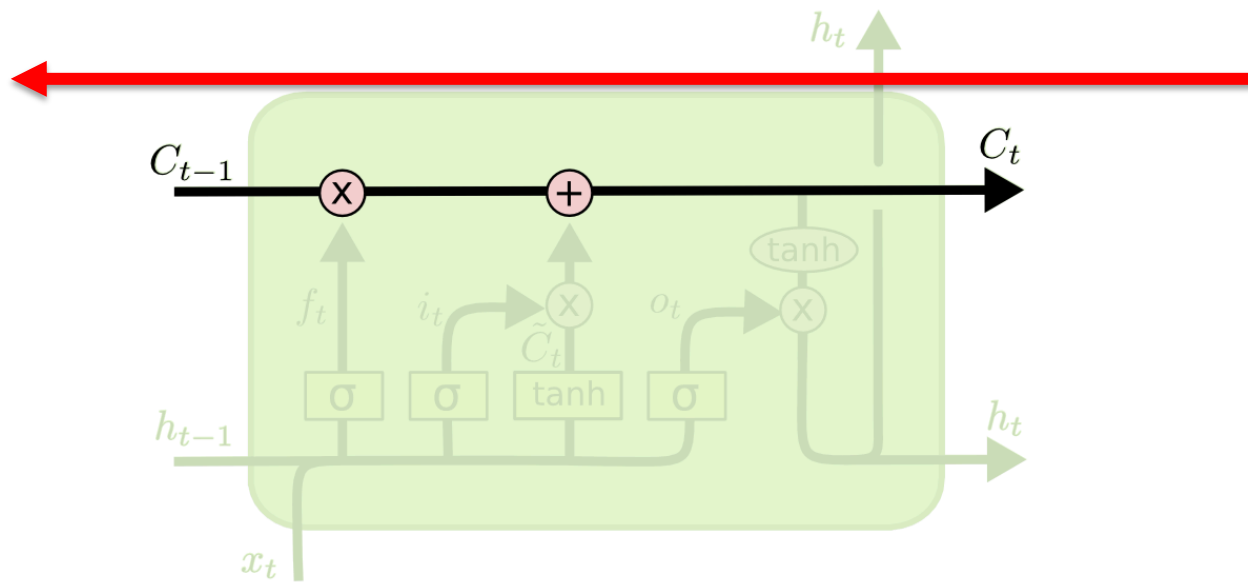
- Cell
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

weights

Identity function

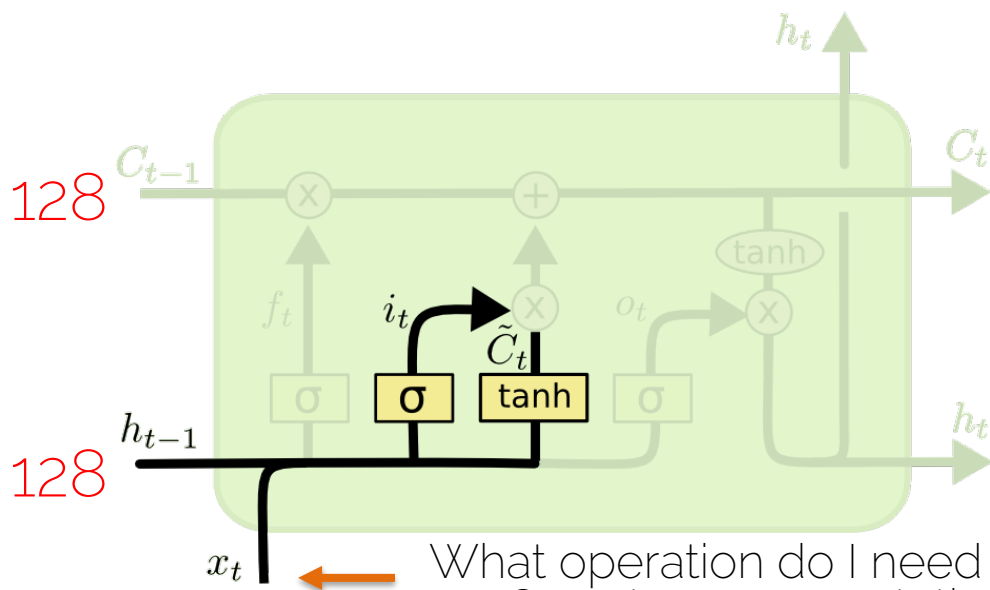
LSTM

- Highway for the gradient to flow



LSTM: Dimensions

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



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

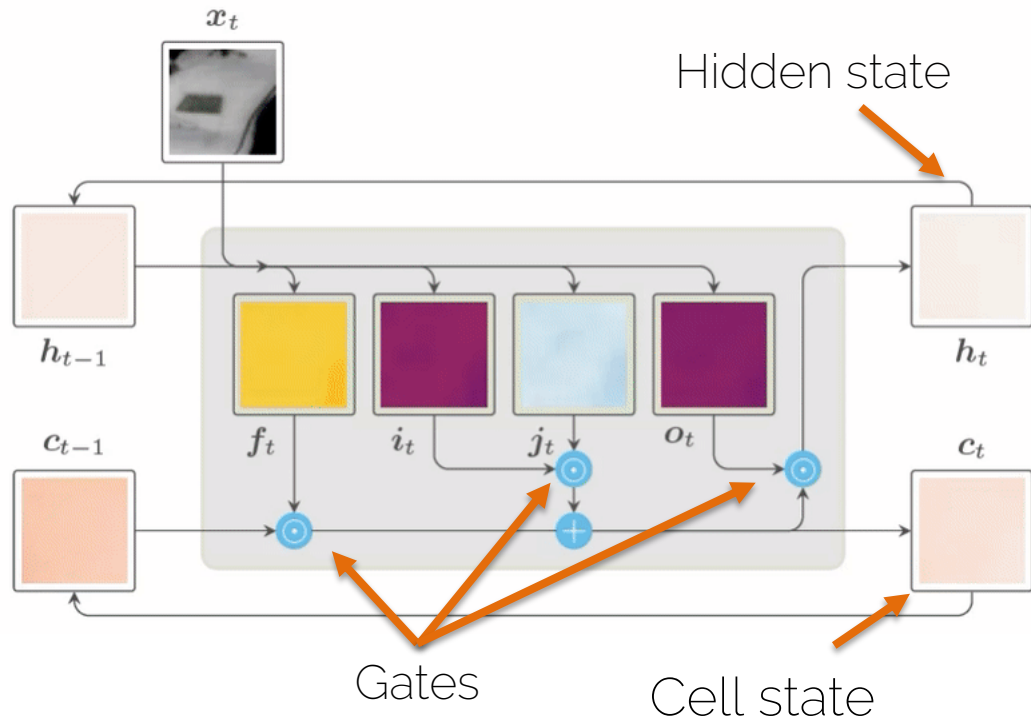
Dimensions need to match

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

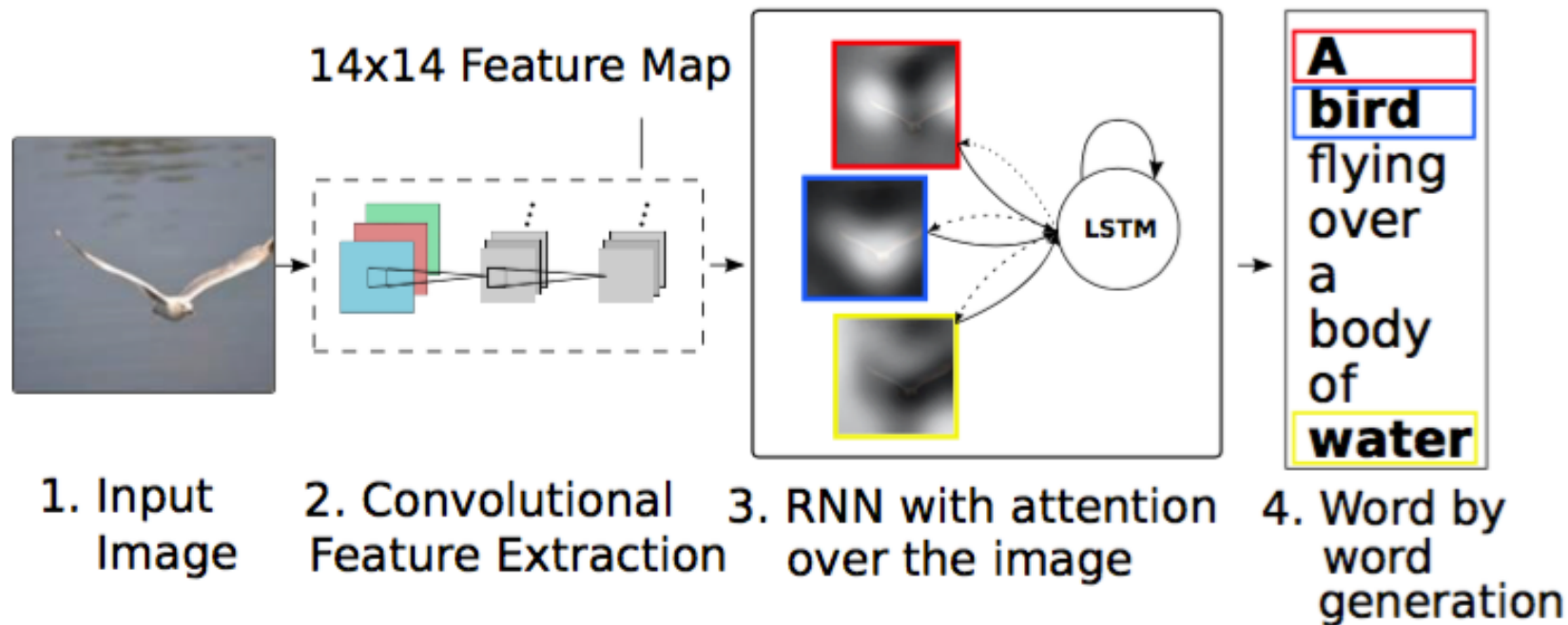
ConvLSTM for Video Sequences



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

RNNs in Computer Vision

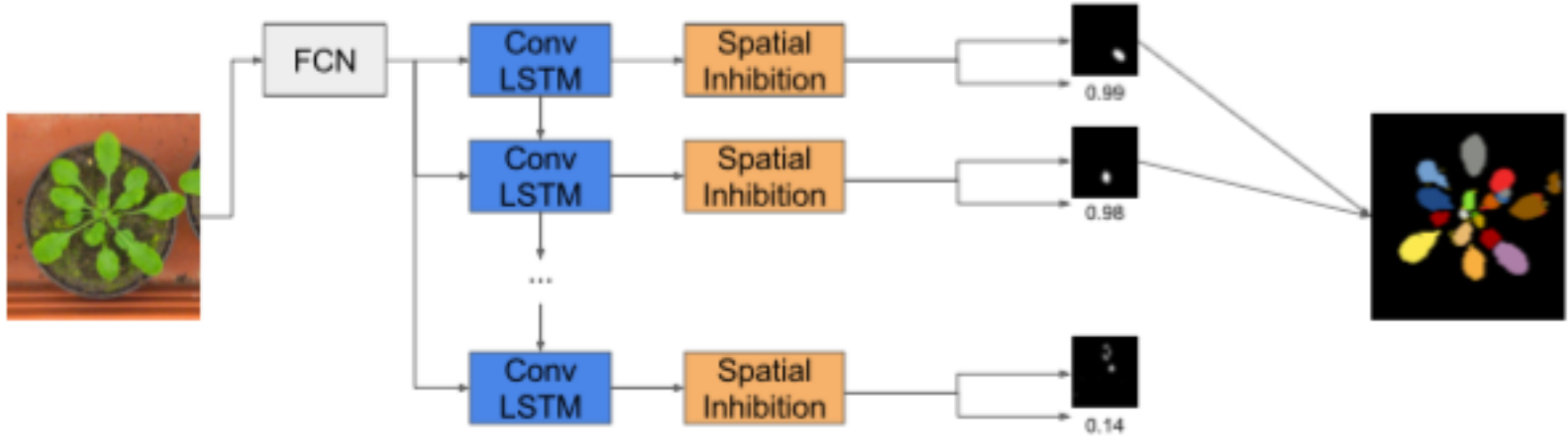
- Caption generation



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

RNNs in Computer Vision

- Instance segmentation



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

See you next time!