

DEEP LEARNING

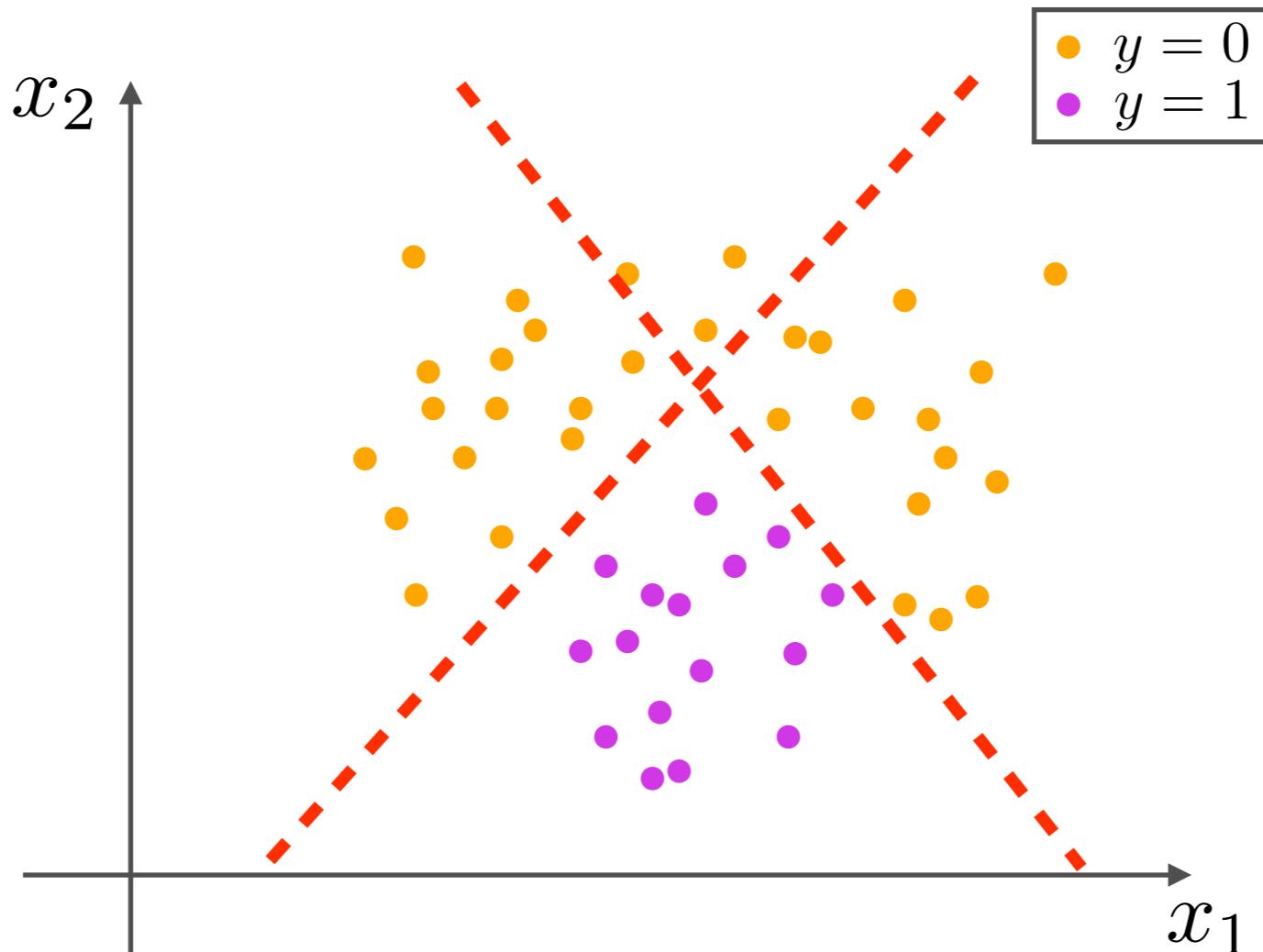
PART TWO - *CONVOLUTIONAL & RECURRENT NETWORKS*

Slides by Joe Marino



REVIEW

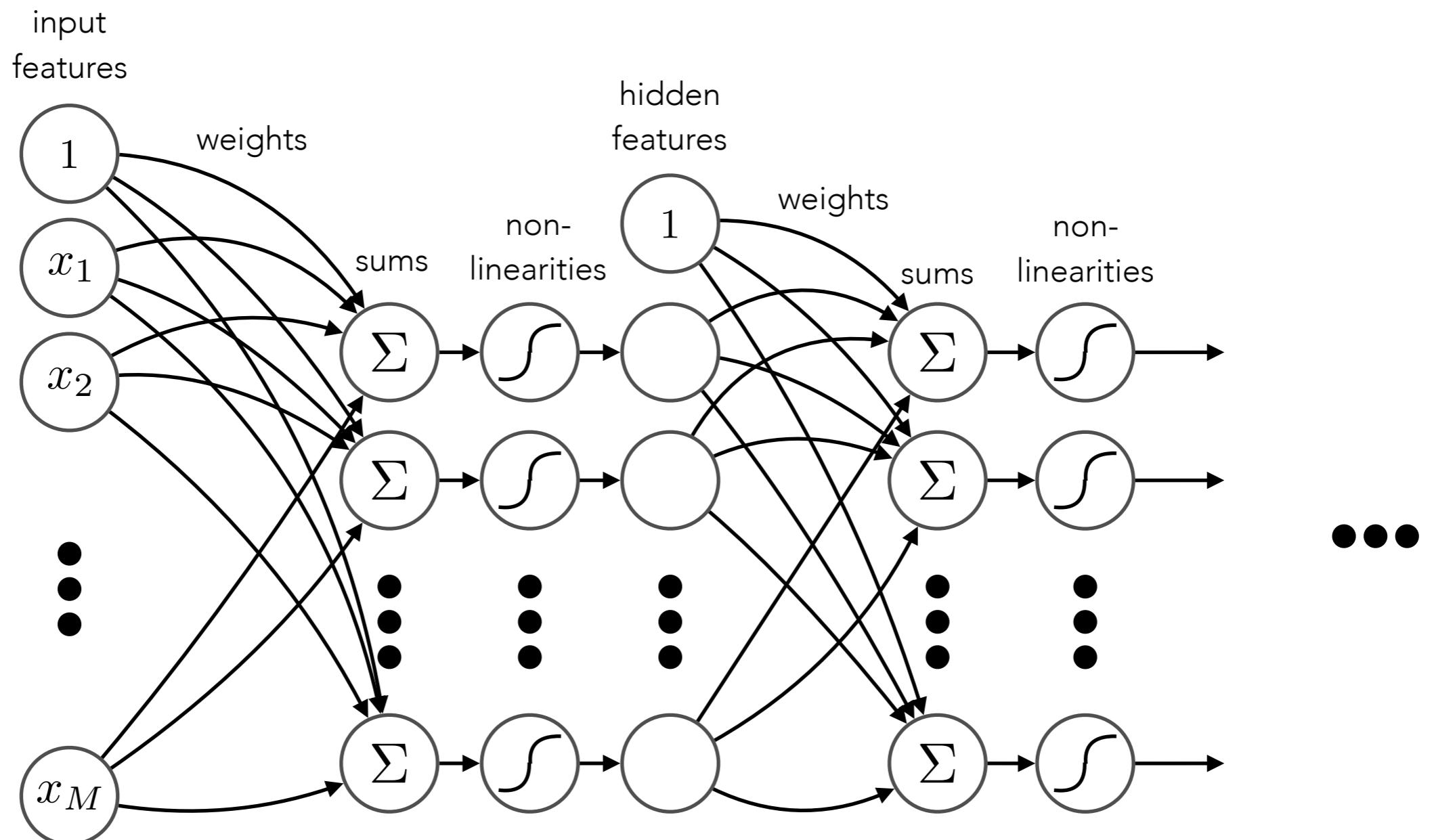
we want to learn **non-linear** decision boundaries



$$\mathbf{x} = (x_1, x_2)$$

we can do this by composing *linear* decision boundaries

neural networks formalize a method for building these composed functions



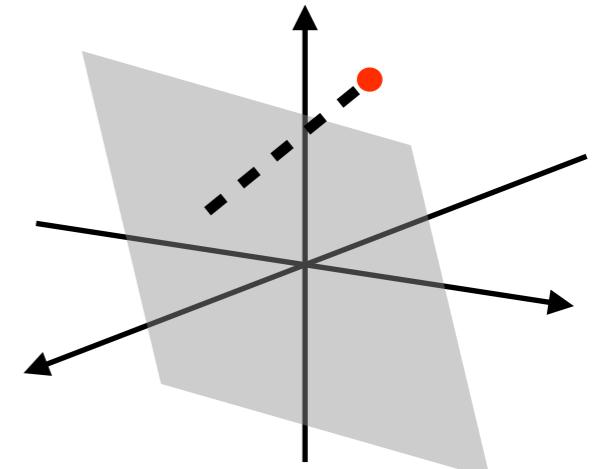
deep networks are *universal function approximators*

a geometric interpretation

the dot product is the shortest distance between a point and a plane

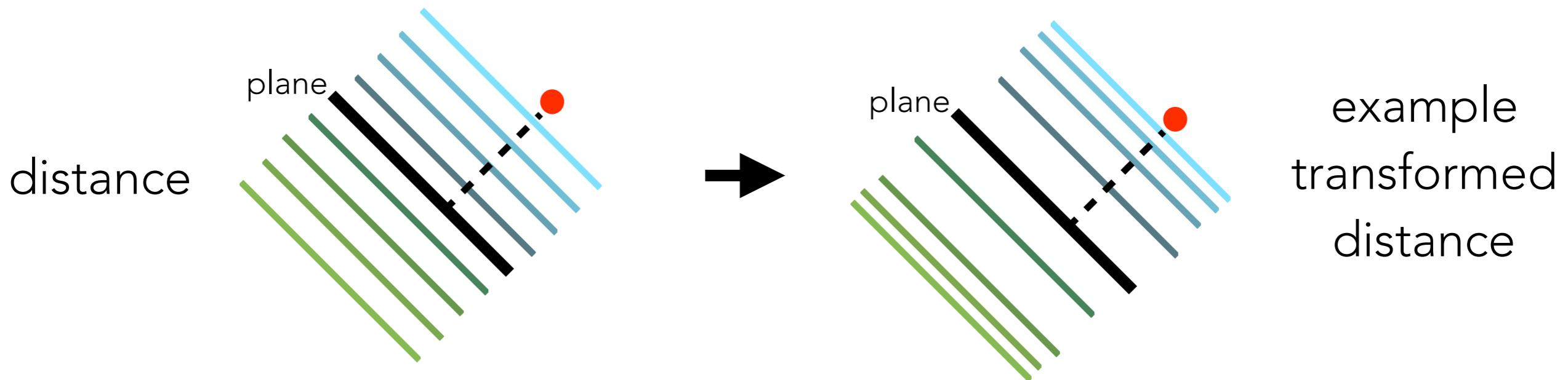
each artificial neuron defines a (hyper)plane:

$$0 = w_0 + w_1x_1 + w_2x_2 + \dots w_Mx_M$$

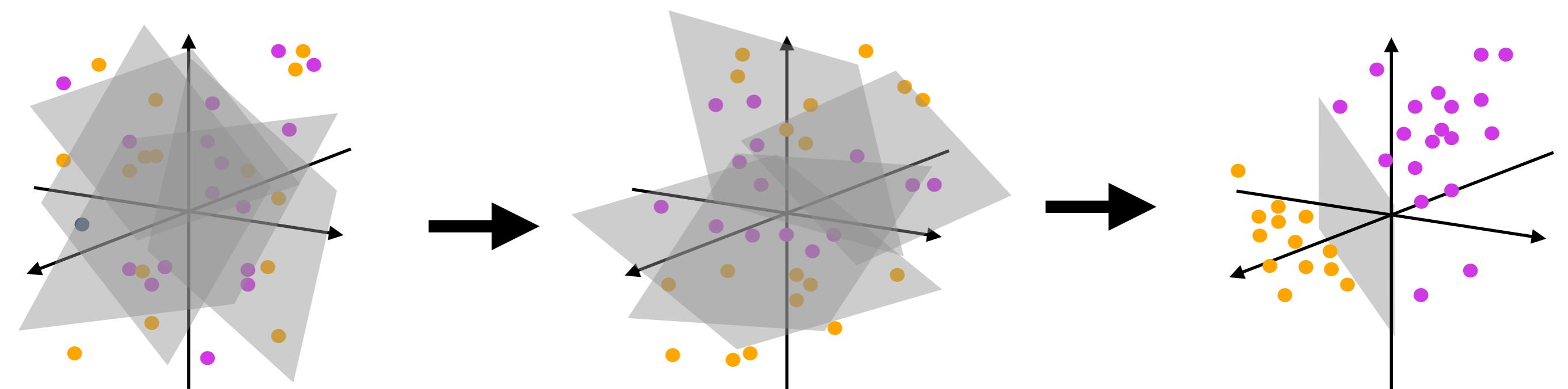


summation: distance from plane to input

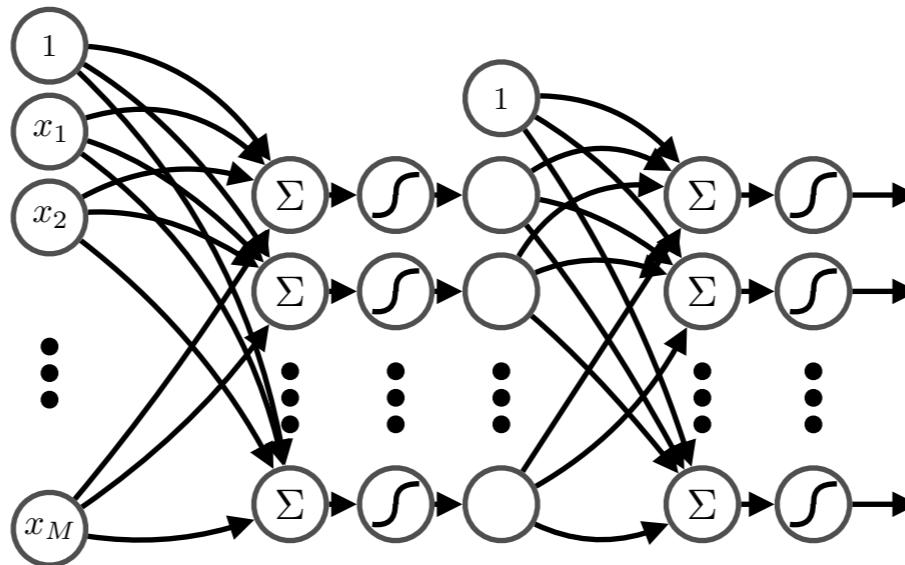
non-linearity: convert distance into non-linear field



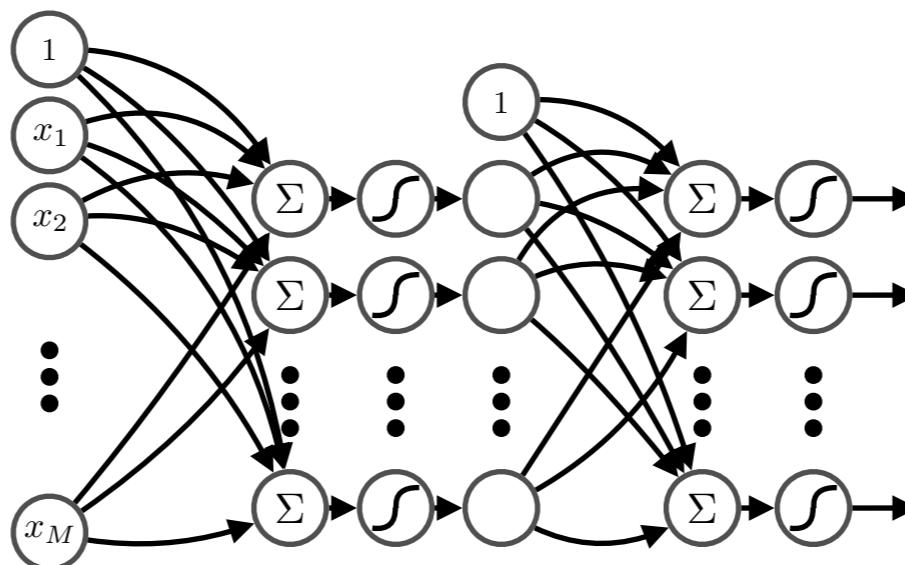
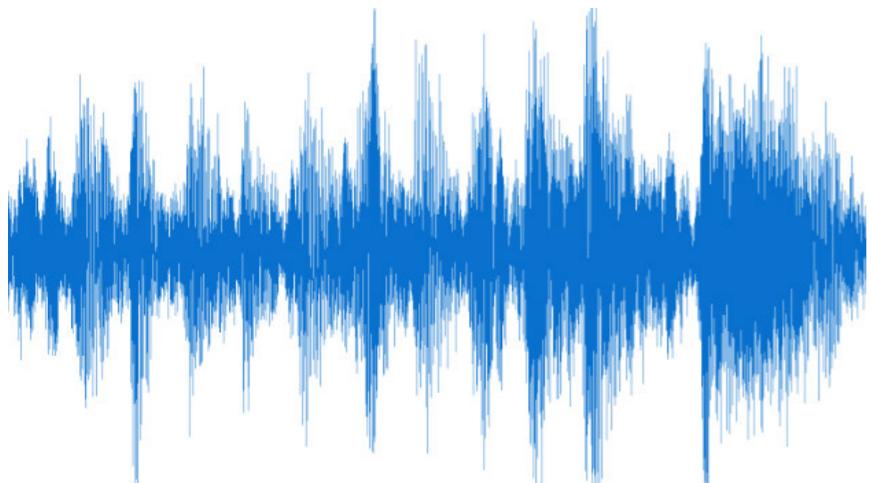
1. cut the space up with hyperplanes
2. evaluate distances of points to hyperplanes
3. non-linearly transform these distances to get new points



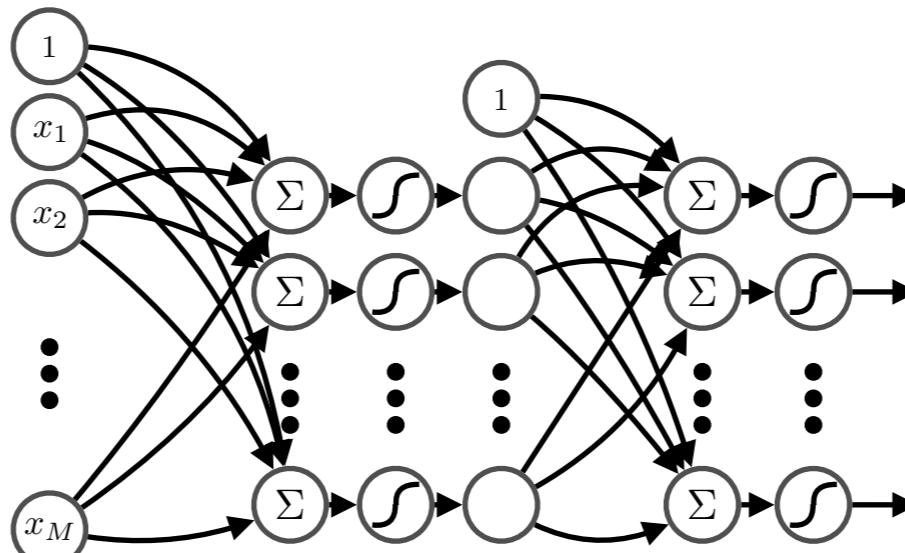
repeat until data have been *linearized*



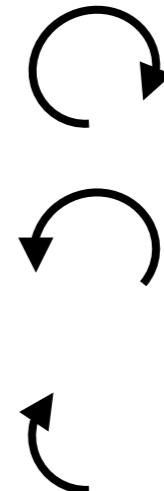
cat



"Alexa, what is the weather going to be like today?"



torques



today



images



audio & text

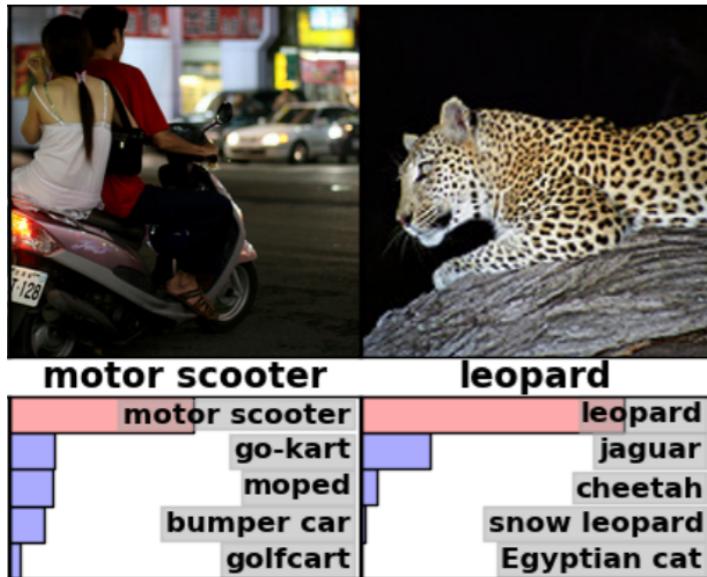


virtual/physical control tasks

to scale deep networks to these domains,
we often need to use ***inductive biases***

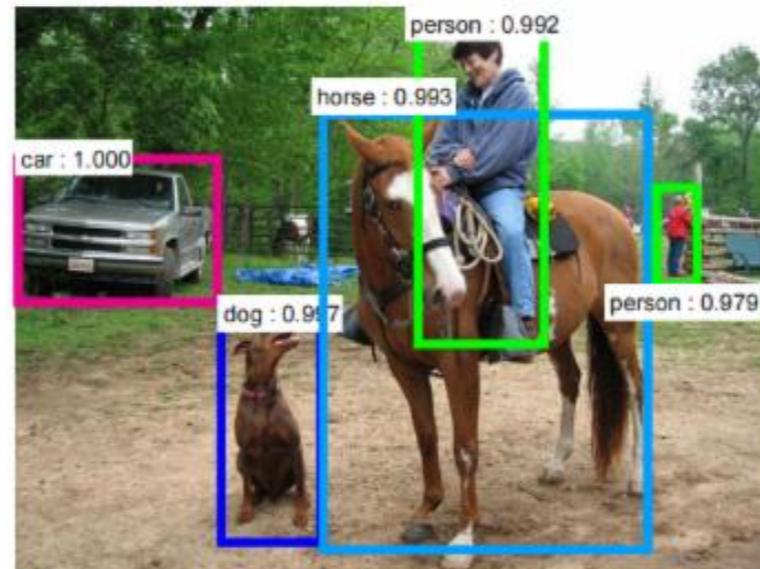
INDUCTIVE BIASES

object recognition



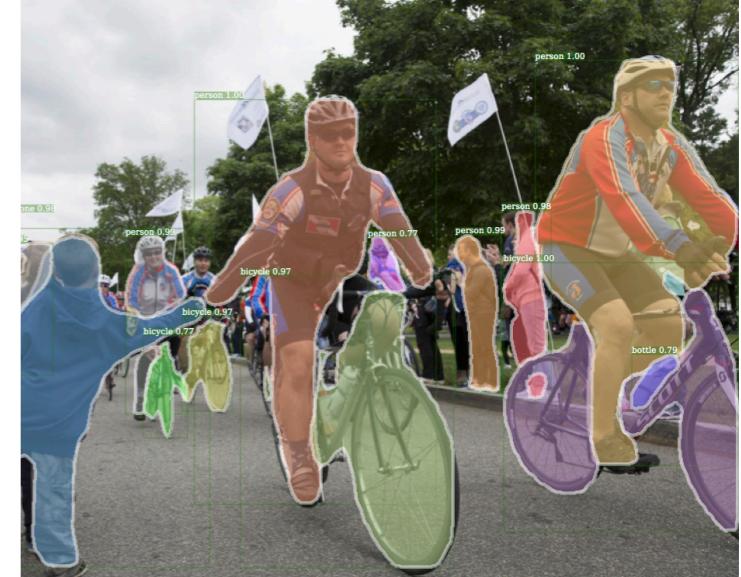
Krizhevsky et al., 2012

object detection



Ren et al., 2016

object segmentation



He et al., 2017

ultimately, we care about solving tasks

text question answering

text translation

Source	Analysts believe the country is unlikely to slide back into full-blown conflict, but recent events have unnerved foreign investors and locals.
PBMT	Les analystes estiment que le pays a peu de chances de retomber dans un conflit total, mais les événements récents ont inquiété les investisseurs étrangers et locaux. 5.0
GNMT	Selon les analystes, il est peu probable que le pays retombe dans un conflit généralisé, mais les événements récents ont attiré des investisseurs étrangers et des habitants locaux. 2.0
Human	Les analystes pensent que le pays ne devrait pas retomber dans un conflit ouvert, mais les récents événements ont ébranlé les investisseurs étrangers et la population locale. 5.0

Wu et al., 2016

```

1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary?      bathroom      1
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel?    hallway 4
7 John moved to the office.
8 Sandra journeyed to the bathroom.
9 Where is Daniel?    hallway 4
10 Mary moved to the hallway.
11 Daniel travelled to the office.
12 Where is Daniel?    office 11
13 John went back to the garden.
14 John moved to the bedroom.
15 Where is Sandra?    bathroom      8
1 Sandra travelled to the office.
2 Sandra went to the bathroom.
3 Where is Sandra?    bathroom      2

```

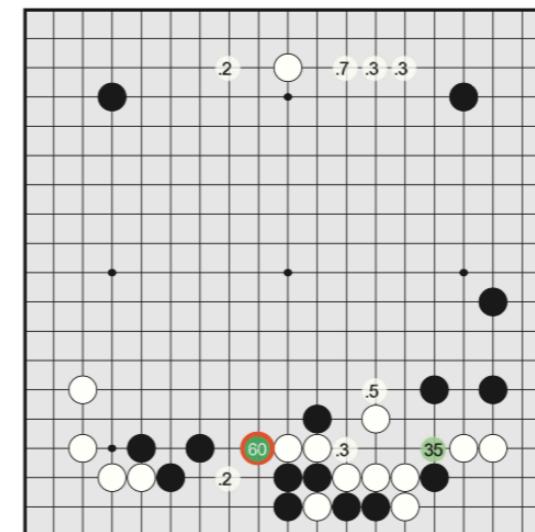
Weston et al., 2015

atari



Minh, et al., 2013

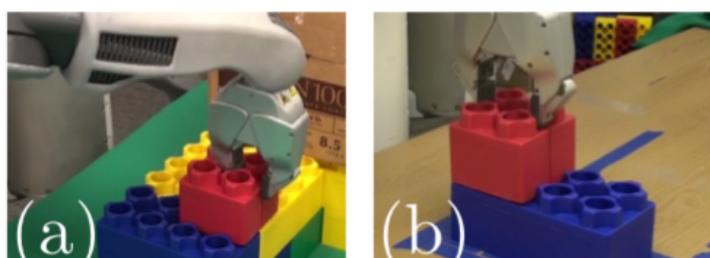
go



Silver, Huang, et al., 2016

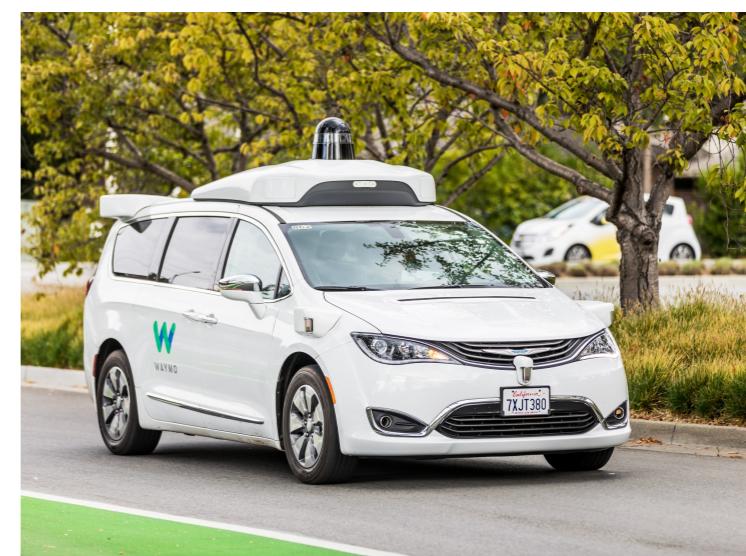
ultimately, we care about *solving tasks*

object manipulation



Levine, Finn, et al., 2016

autonomous driving



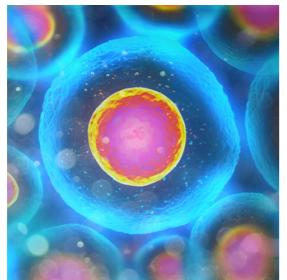
Waymo

survival & reproduction

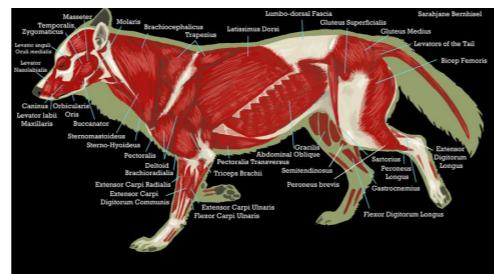


ultimately, we care about solving tasks

cellular signaling,
maintenance



muscle actuation



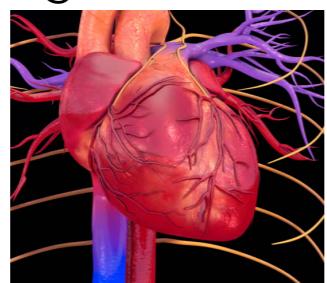
navigation



social/mating
behavior



organ function



vision



hunting,
foraging



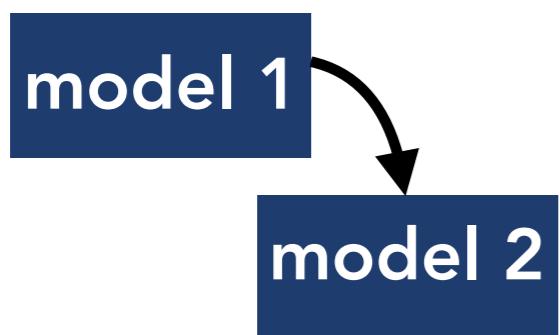
two components for solving any task

priors

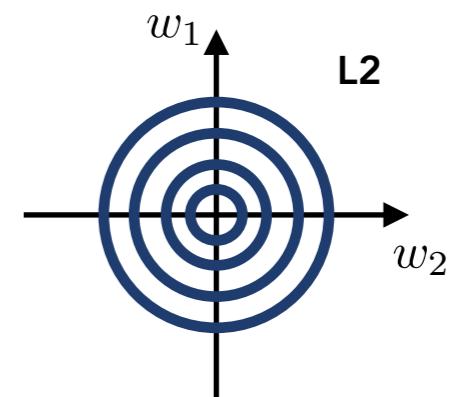


learning

param. values

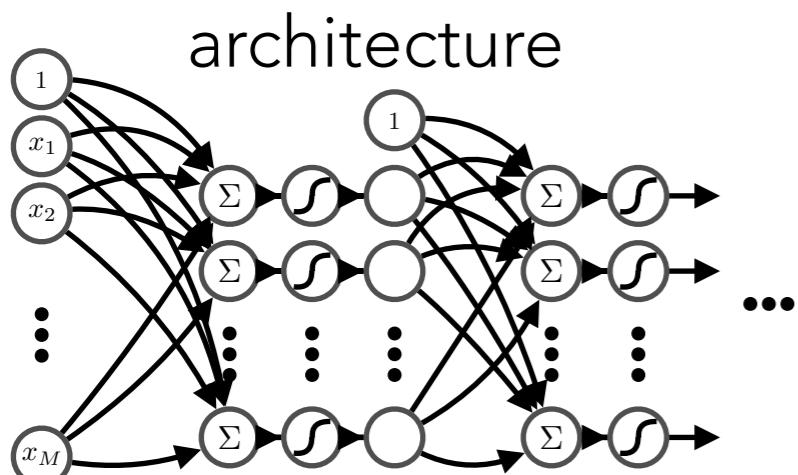


param. constraints

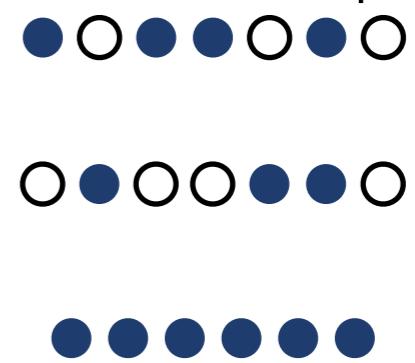


priors

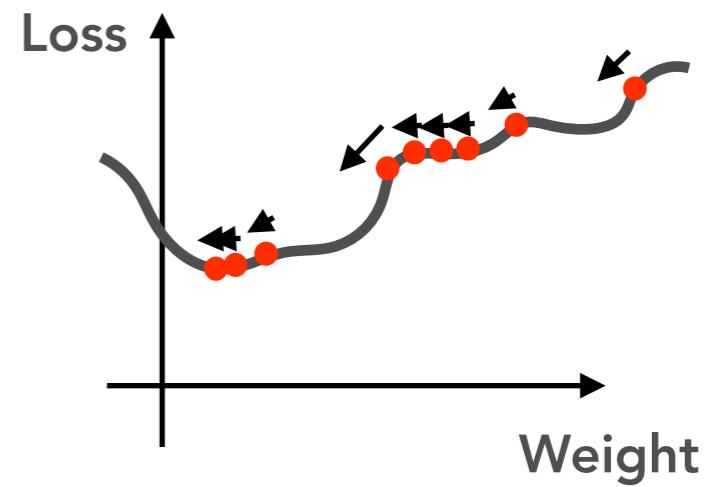
knowledge assumed beforehand



activities, outputs

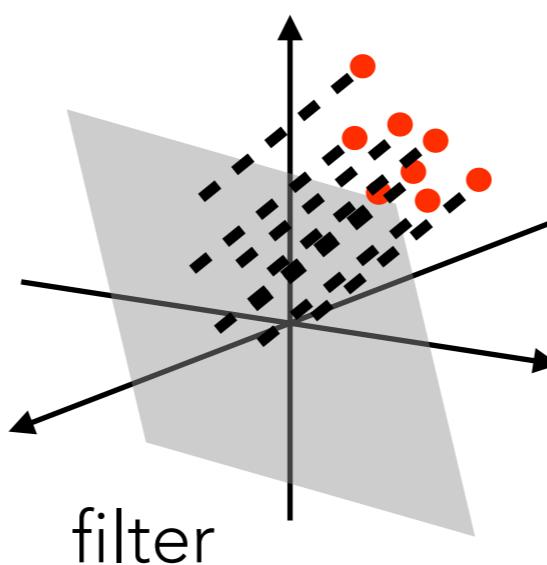


LOSS/ERROR
↓
GRADIENT
↓
IMPROVEMENT



learning

knowledge extracted from data



it's a balance!



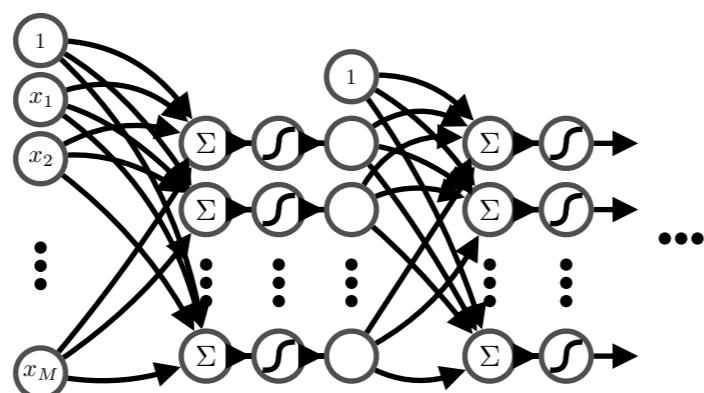
strong priors, minimal learning

- fast/easy to learn and deploy
- may be too rigid, unadaptable

weak priors, much learning

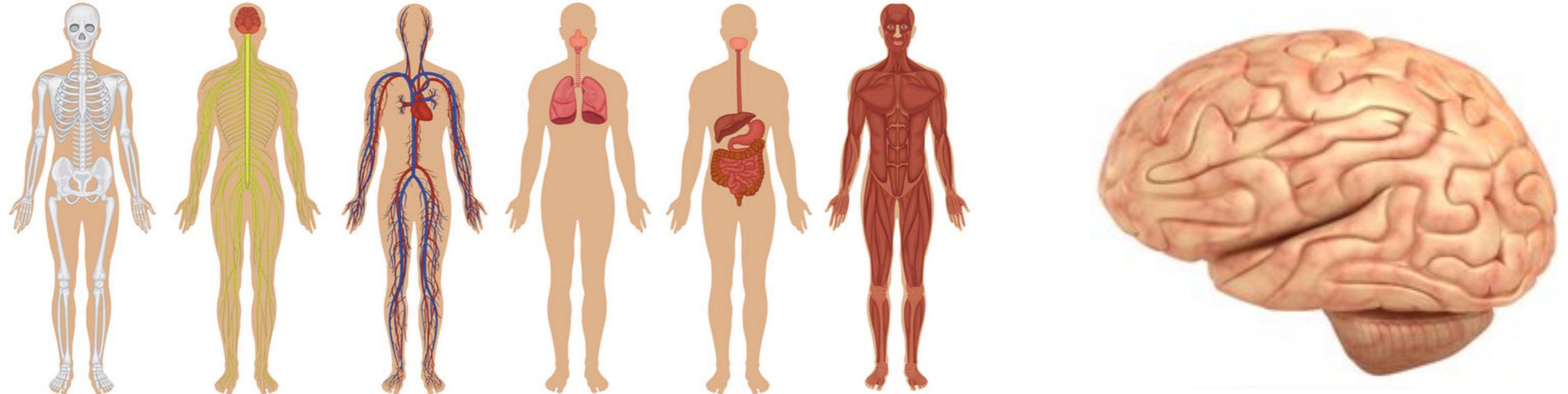
- slow/difficult to learn and deploy
- flexible, adaptable

for a desired level of performance on a task...



choose priors and collect data to obtain a model
that achieves that performance in the minimal amount of time

priors are essential - always have to make some assumptions,
cannot integrate over all possible models



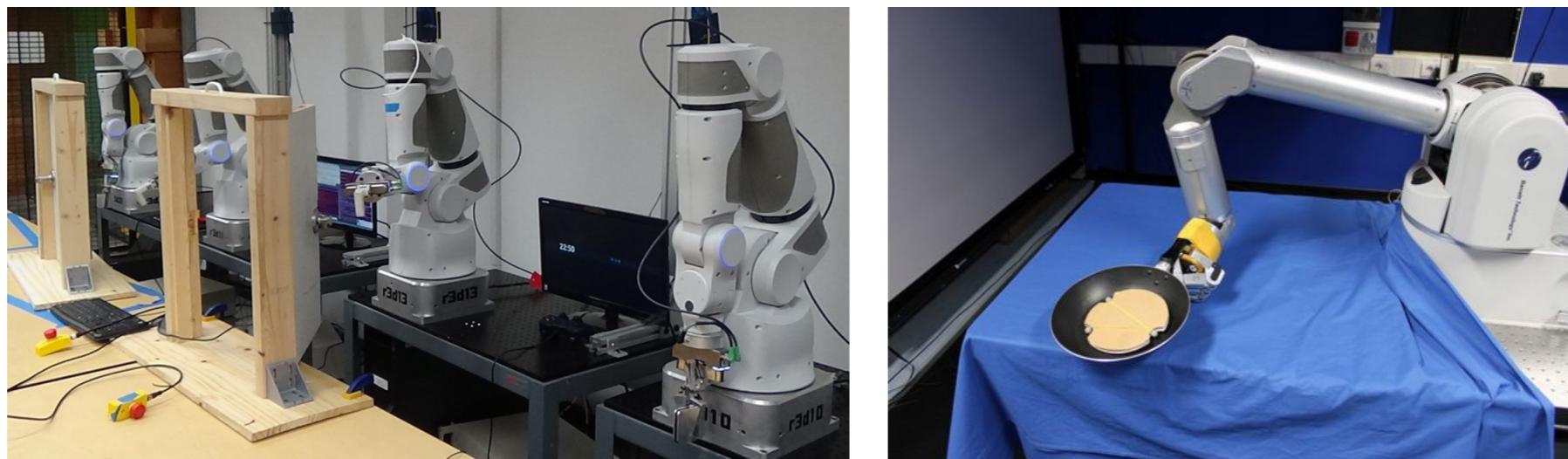
we are all initialized from *evolutionary priors*

humans seem to have a larger capacity for learning than other organisms

up until now, all of our machines have been purely based on priors



for the first time in history, **we can now create machines that also learn**

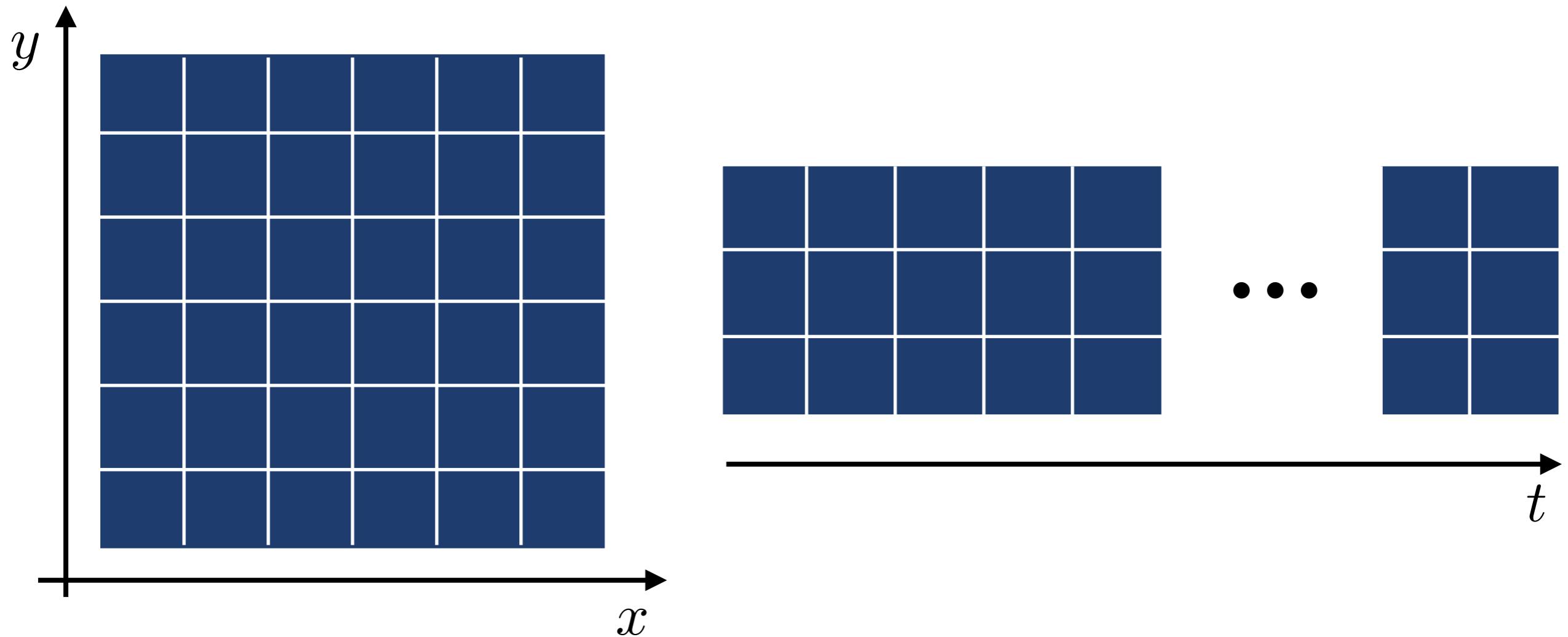


these machines can perform tasks that are impossible to hand-design

...but they are mostly still based on priors!

we can exploit known structure in spatial and sequential data
to impose priors (i.e. inductive biases) on our models

inductive: inferring general laws from examples



this allows us to learn models in complex, high-dimensional domains
while limiting the number of parameters and data examples

CONVOLUTIONAL NEURAL NETWORKS

task: *object recognition*



Yisong

discriminative mapping from image to object identity



images contain all of the information about the binary latent variable Yisong/Not Yisong

extract the relevant information about this latent variable to form conditional probability

inference: $p(\text{Yisong} | \quad)$



notice that images also contain other *nuisance* information, such as pose, lighting, background, etc.

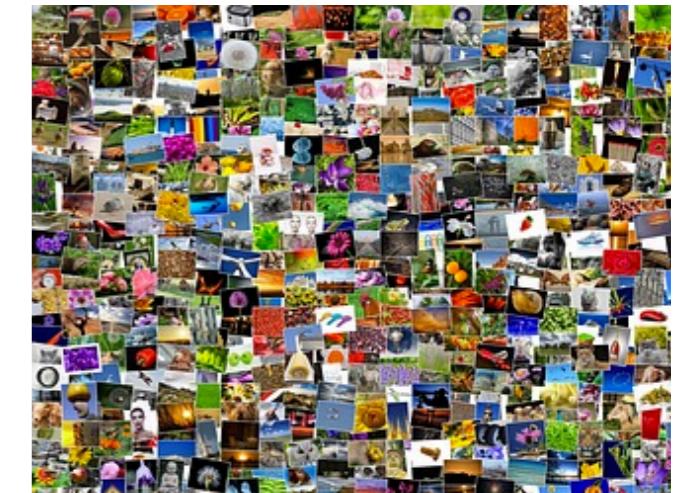
want to be *invariant* to nuisance information

data, label collection

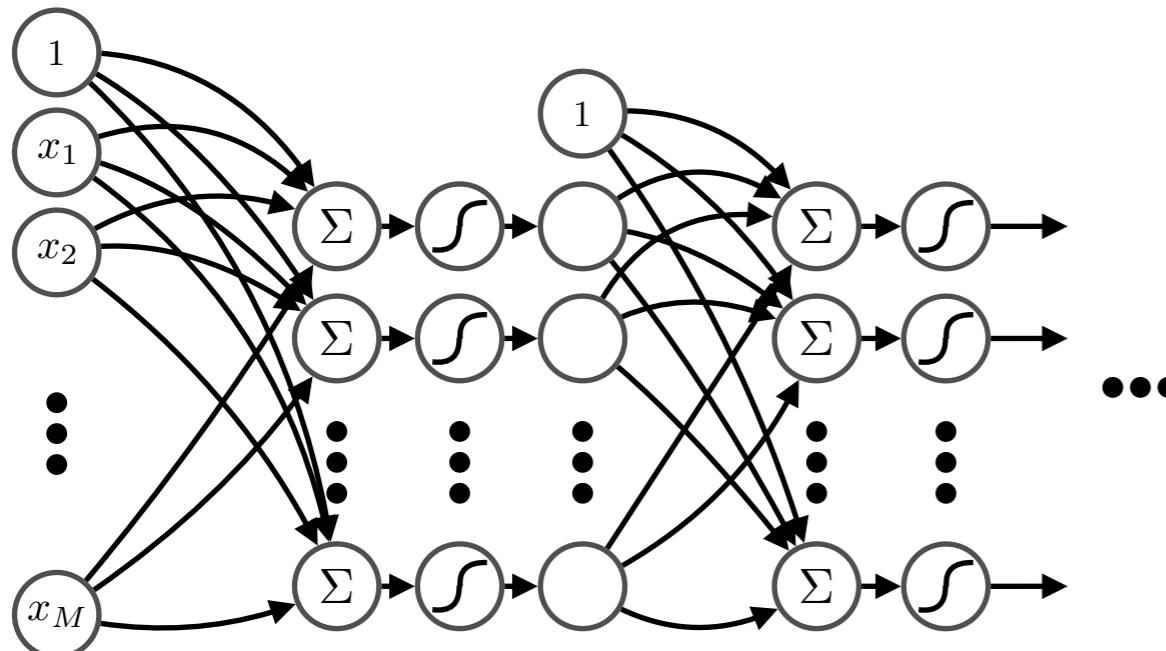
the mapping is too difficult to define by hand,
need to learn from data



Yisong



Not Yisong



then, we need to choose
a model architecture...

standard neural networks require a fixed input size...



fewer parameters,
but unclear patterns



clearer patterns,
but more parameters

convert to grayscale...

15 25 35

x x x

15 25 35

x x x 50 x 50

75 x 75 x 1

100 x 100 x 1

1 1 1 x 1

225 625

1,225 2,500

5,625

5,625

5,625 10,000

22,500

150 x 150 x 1

205 x 205 x 1

280 x 280 x 1

22,500

42,025

78,400

fewer parameters,
but unclear patterns



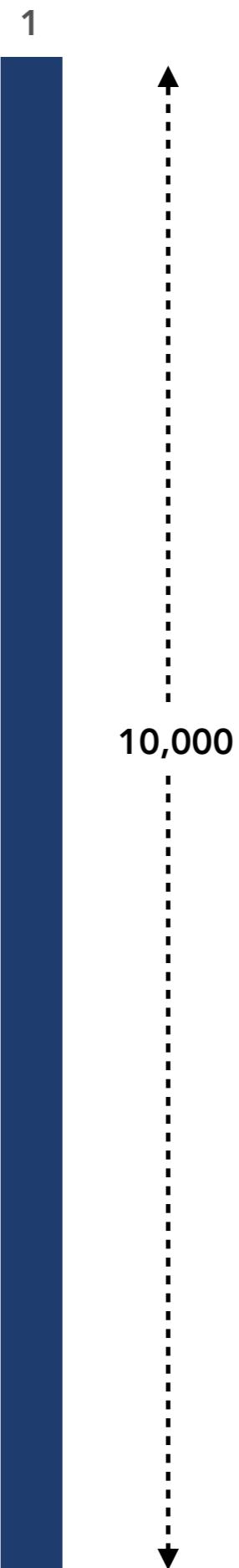
clearer patterns,
but more parameters

$100 \times 100 \times 1$



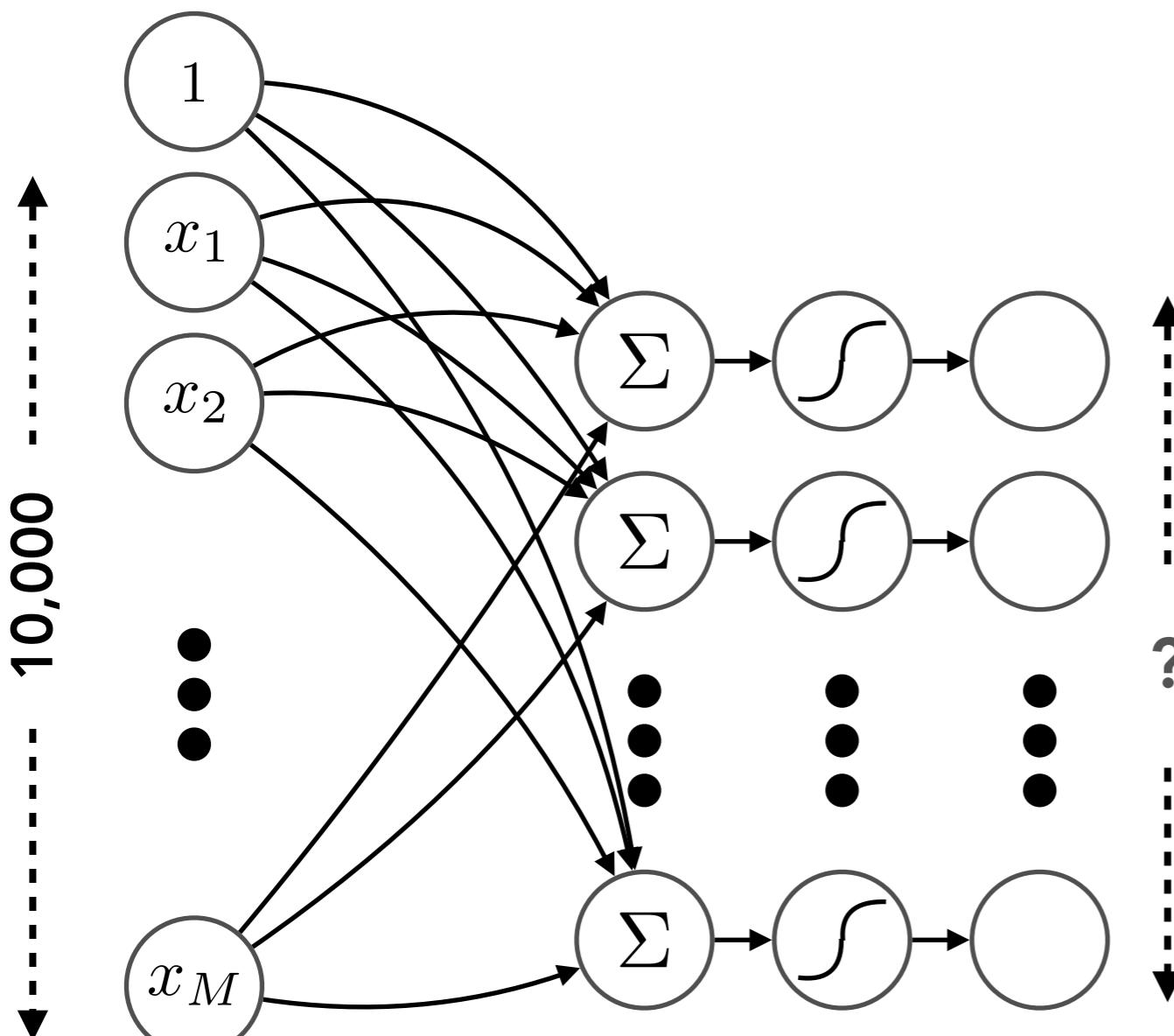
10,000

reshape



how many units do we need?

INPUT

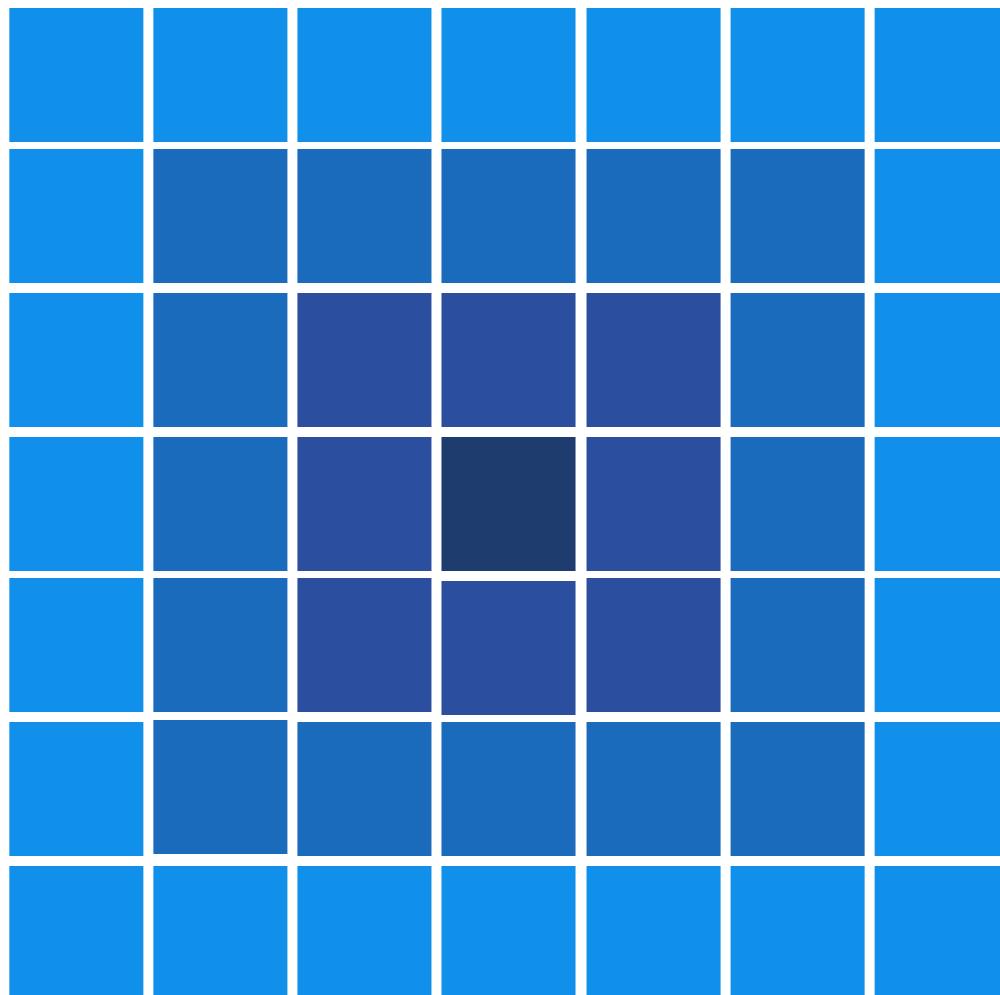


$$\# \text{ units} \times 10,000 = \# \text{ weights}$$

1	----->	10,000
10	----->	100,000
100	----->	1,000,000
1,000	----->	10,000,000
10,000	----->	100,000,000
100,000	----->	1,000,000,000
1,000,000	----->	10,000,000,000

*if we want to recognize even a few basic patterns at each location,
the number of parameters will explode!*

to reduce the amount of learning,
we can introduce *inductive biases*



exploit the ***spatial structure*** of image data

locality

nearby areas tend to contain stronger patterns

*nearby **pixels** tend to be similar and vary
in particular ways*



*nearby **patches** tend to share characteristics
and are combined in particular ways*



*nearby **regions** tend to be found
in particular arrangements*



translation invariance

relative (rather than absolute) positions are relevant



Yisong's identity is independent of absolute location of his pixels

let's convert **locality** and **translation invariance** into *inductive biases*

locality

nearby areas tend
to contain stronger
patterns

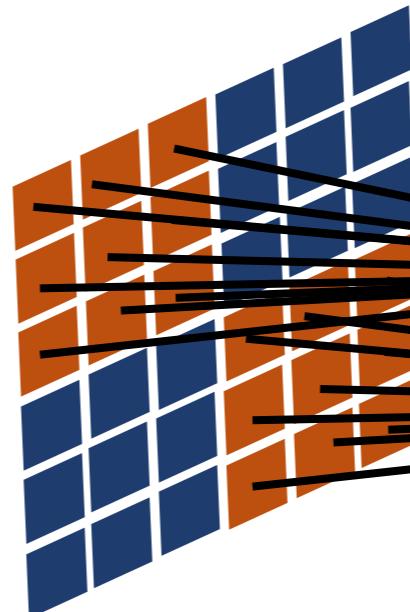


inputs can be
restricted to regions

maintain spatial ordering

translation invariance

relative positions
are relevant

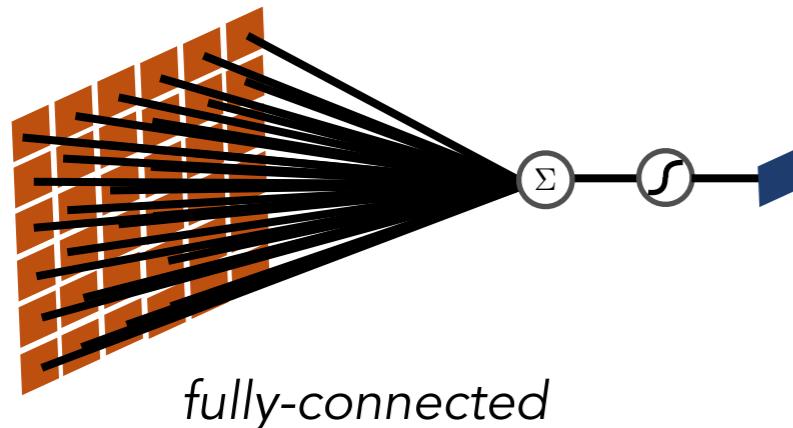


same filters can be applied
throughout the input

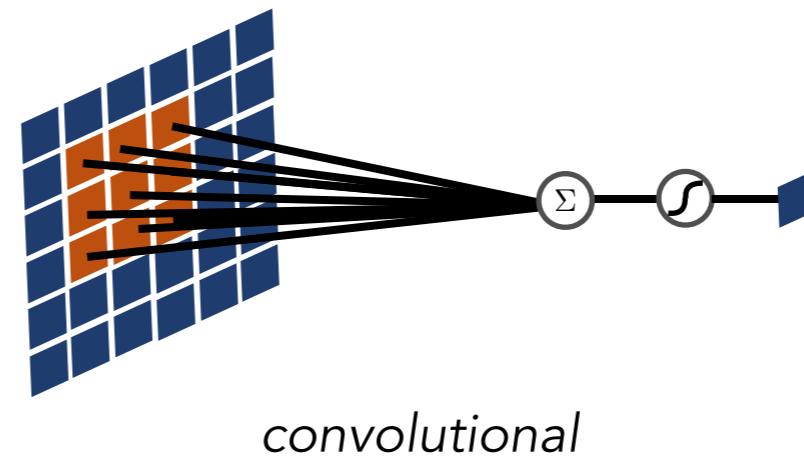
same weights

these are the inductive biases of ***convolutional neural networks***

→ special case of standard (fully-connected) neural networks

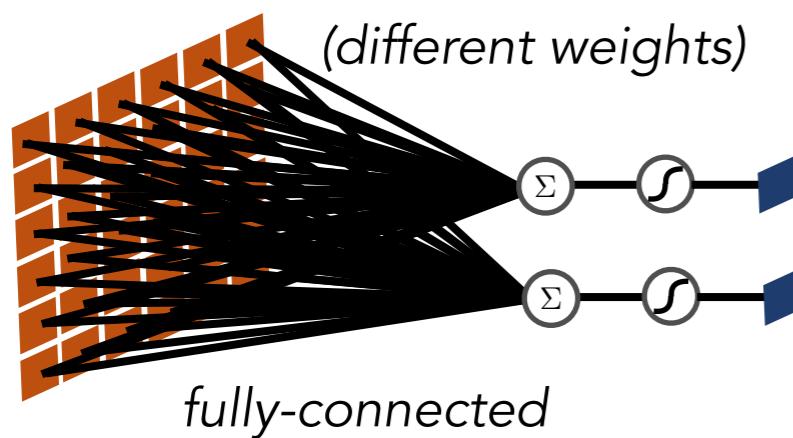


fully-connected

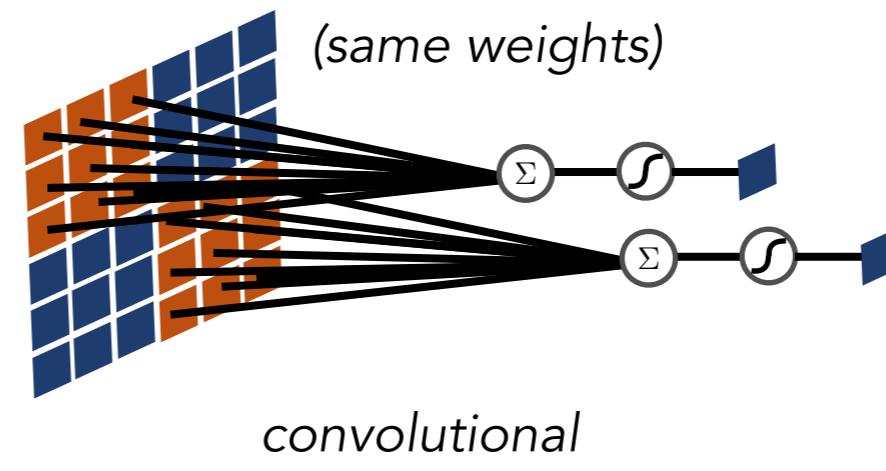


convolutional

weight savings



fully-connected



convolutional

weight savings

these *inductive biases* make the ***number of weights independent of the input size!***

convolve a set of filters with the input

filter weights: $\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

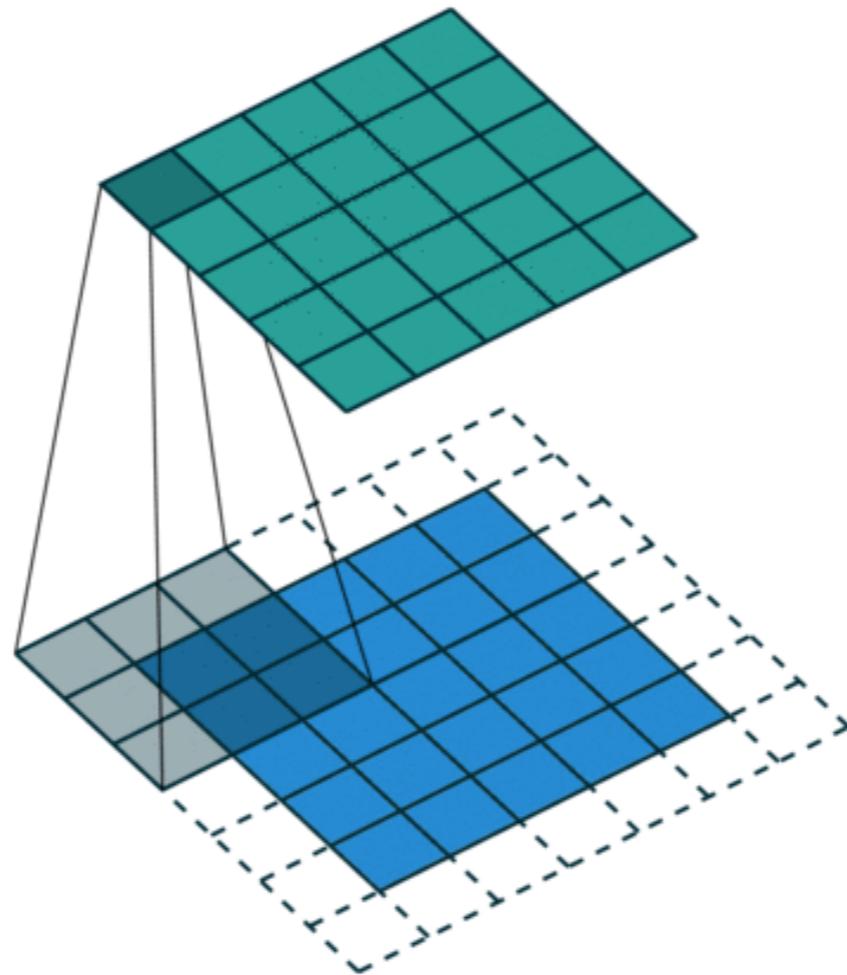
12	12	17
10	17	19
9	6	14

take inner (dot) product of filter and each input location

measures degree of filter feature at input location

→ *feature map*

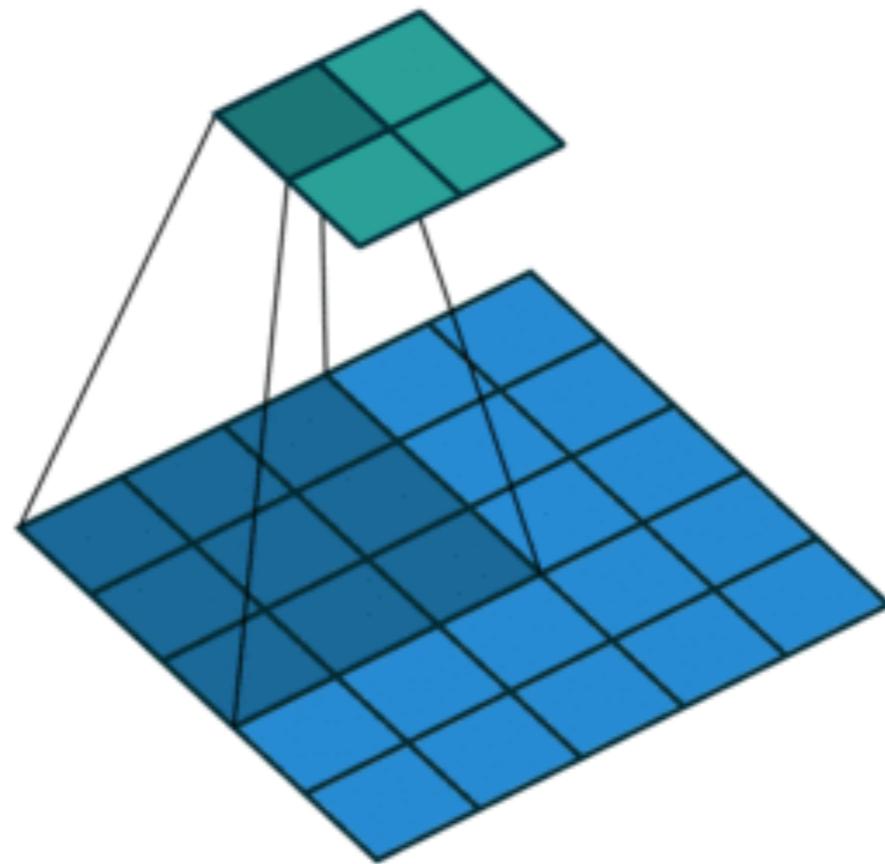
use ***padding*** to preserve spatial size



typically add zeros around the perimeter

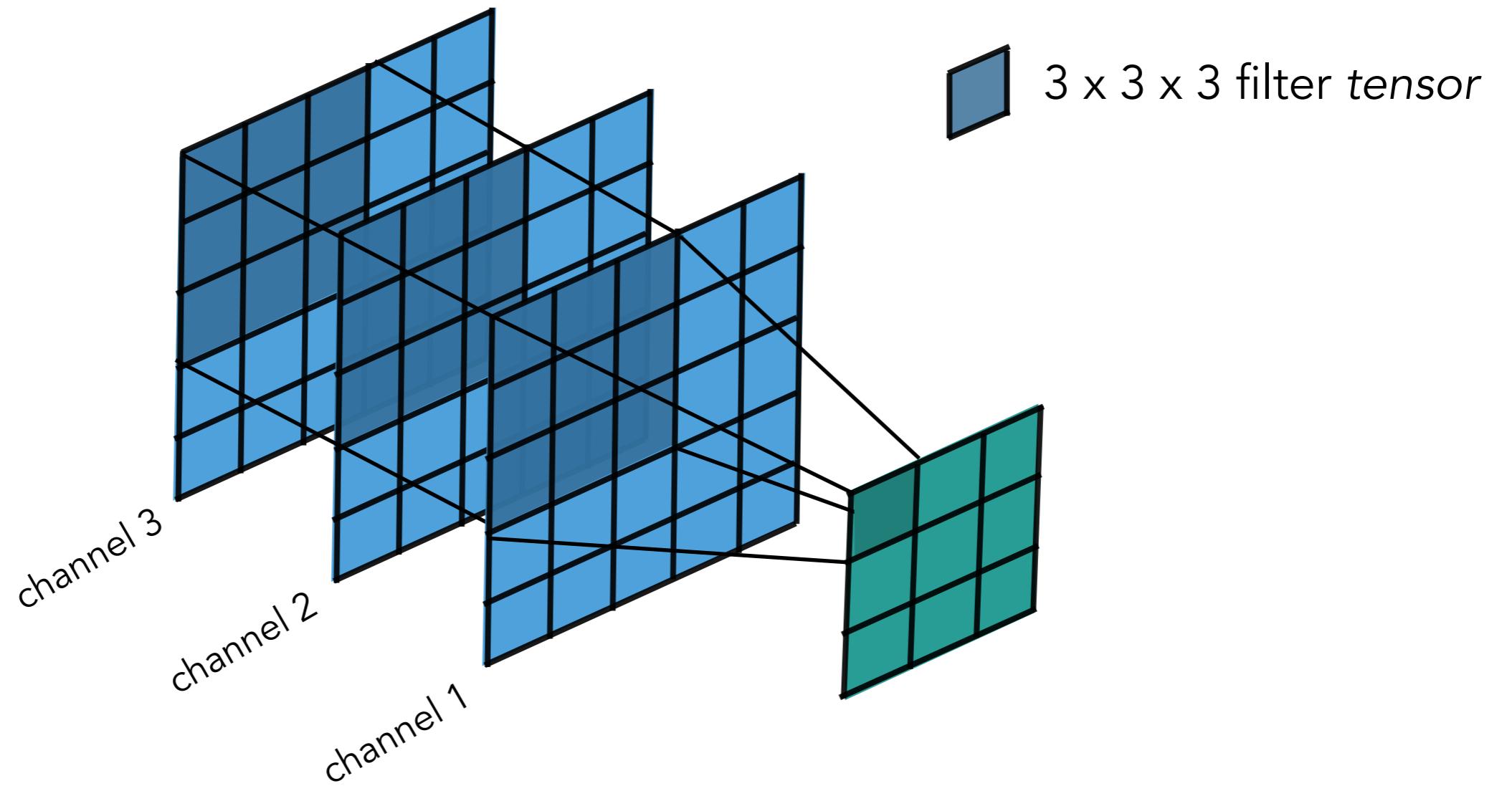
use ***stride*** to downsample the input

$\text{stride} = 2$



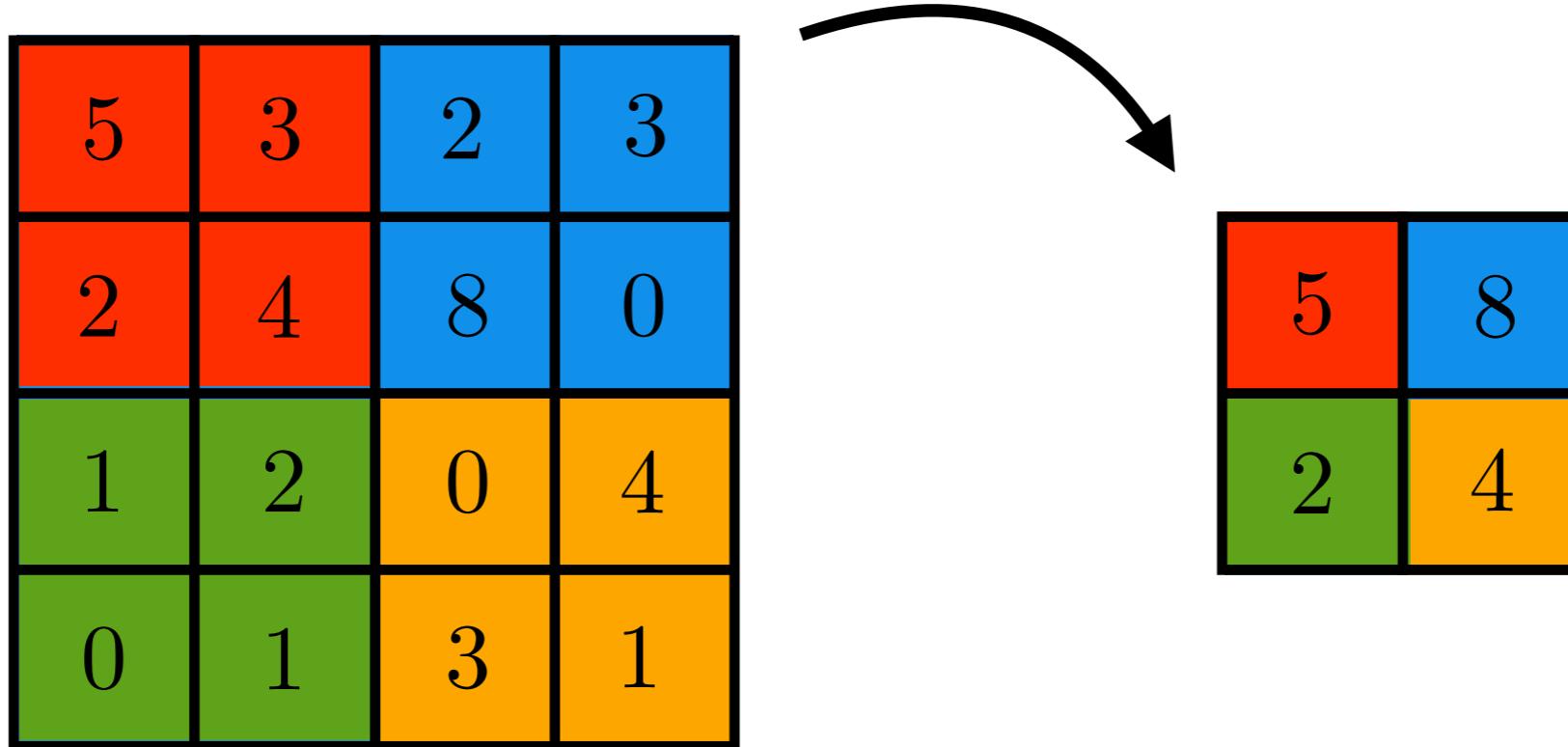
only compute output at some integer interval

filters are applied to all input channels



each filter results in a new output channel

pooling locally aggregates values in each feature map

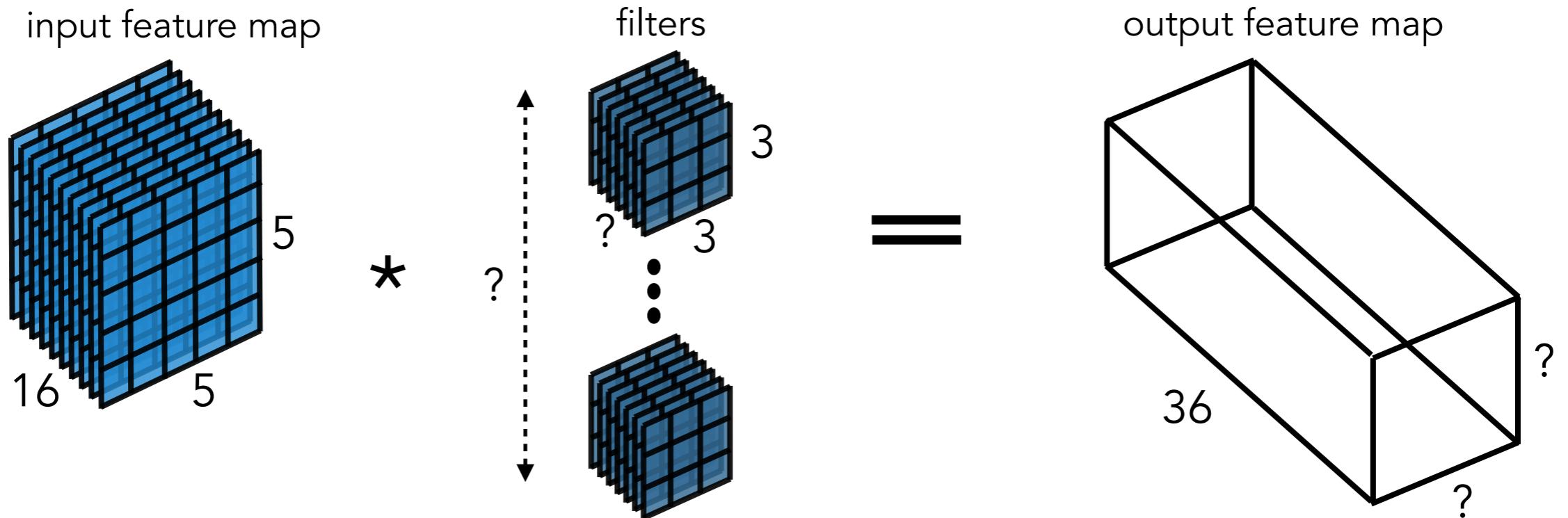


downsampling and invariance

can be applied with *padding* and *stride*

predefined operation: maximum, average, etc.

convolutional pop-quiz



if we use $\text{stride}=1$ and $\text{padding}=0$ then...

how many filters are there? **36**

same as the number of output channels

what size is each filter? **$3 \times 3 \times 16$**

channels match the number of input channels

what is the output filter map size? **$3 \times 3 \times 36$**

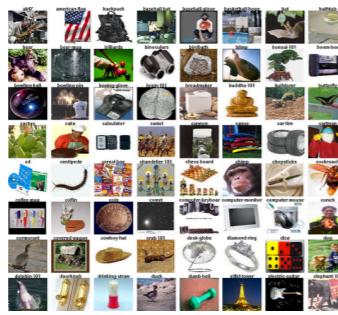
result of only valid convolutions

natural image datasets



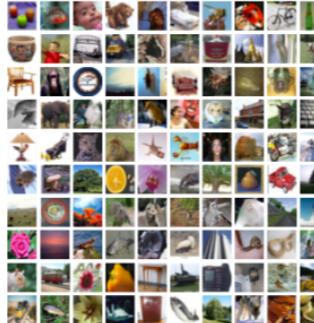
Caltech-101

101 classes,
9,146 images



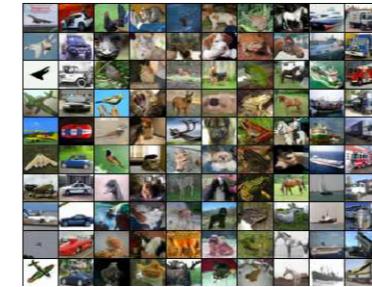
Caltech-256

256 classes,
30,607 images



CIFAR-10

10 classes,
60,000 images



CIFAR-100

100 classes,
60,000 images



ImageNet

Competition

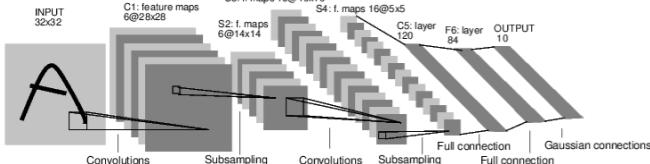
1,000 classes,
1.2 million images

Full

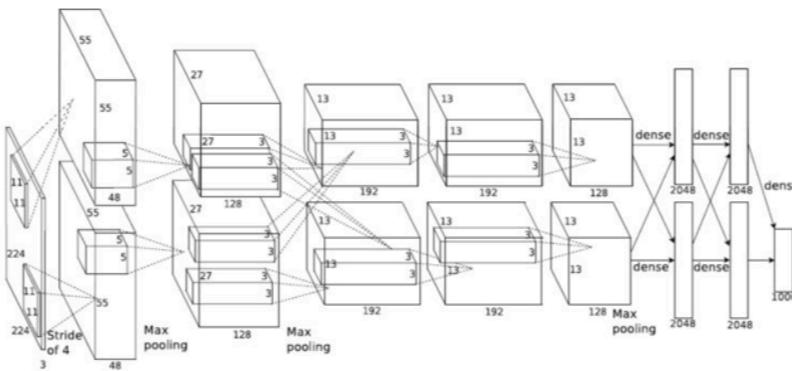
21,841 classes,
14 million images

• • •

convolutional models for classification



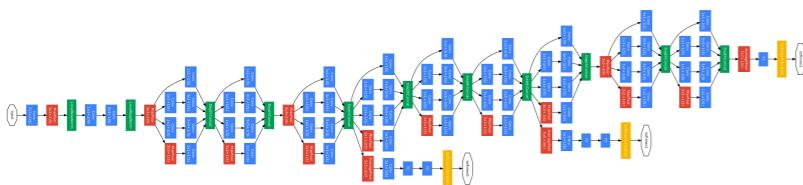
LeNet



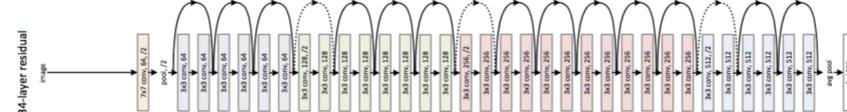
AlexNet



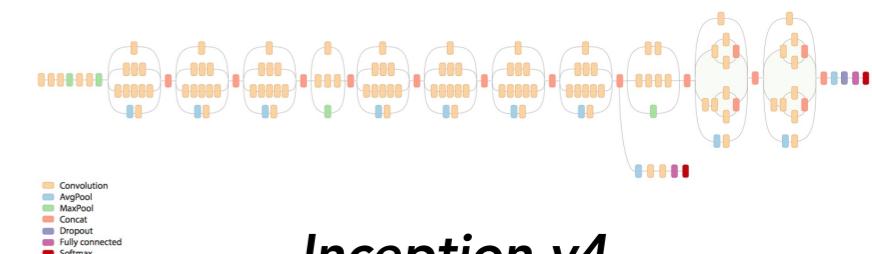
VGG



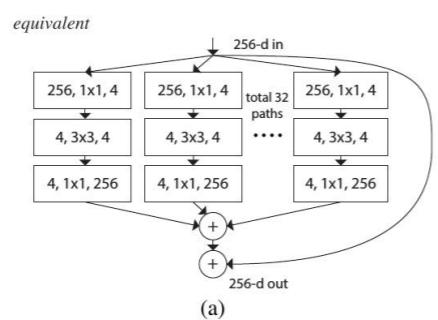
GoogLeNet



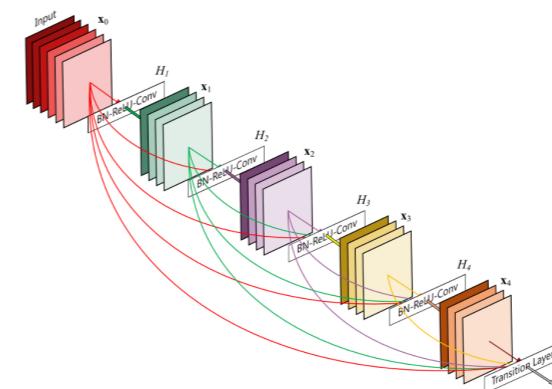
ResNet



Inception v4

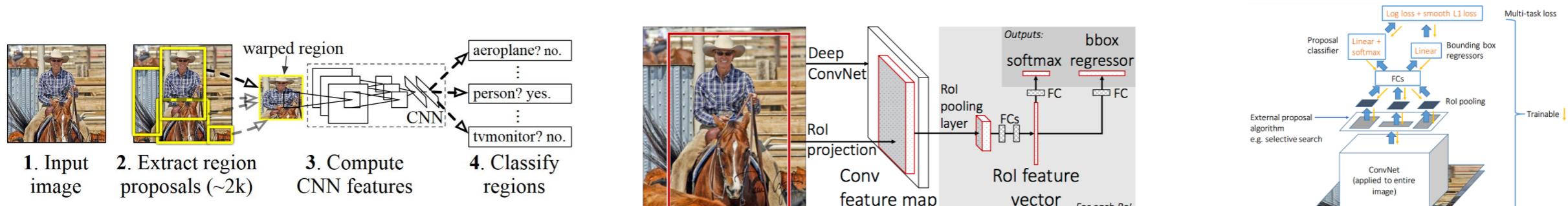


ResNeXt



DenseNet

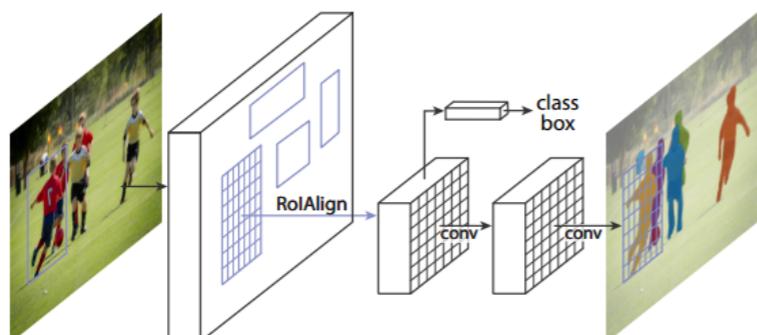
convolutional models for *detection*, *segmentation*, etc.



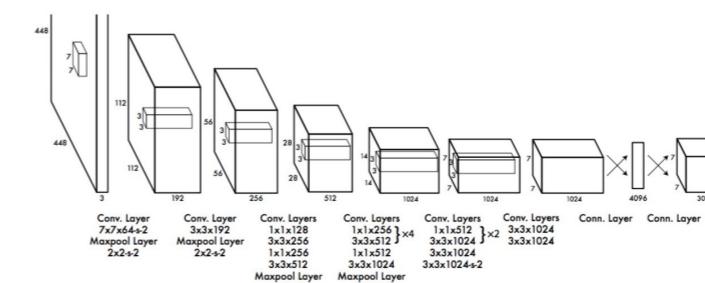
R-CNN

Fast R-CNN

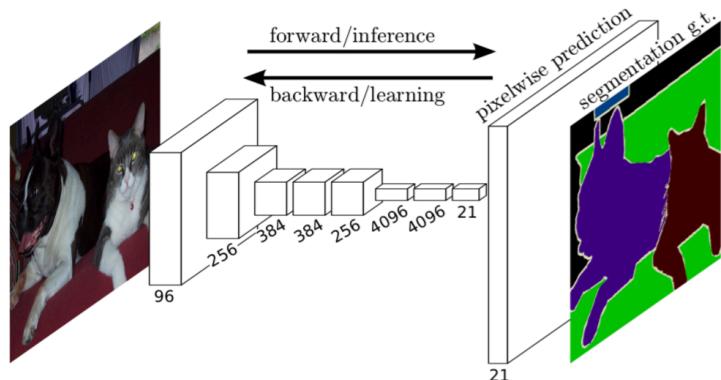
Faster R-CNN



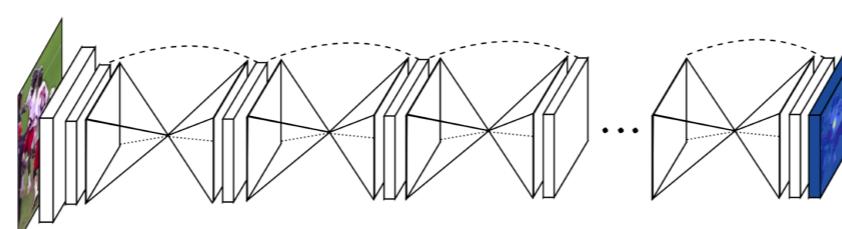
Mask R-CNN



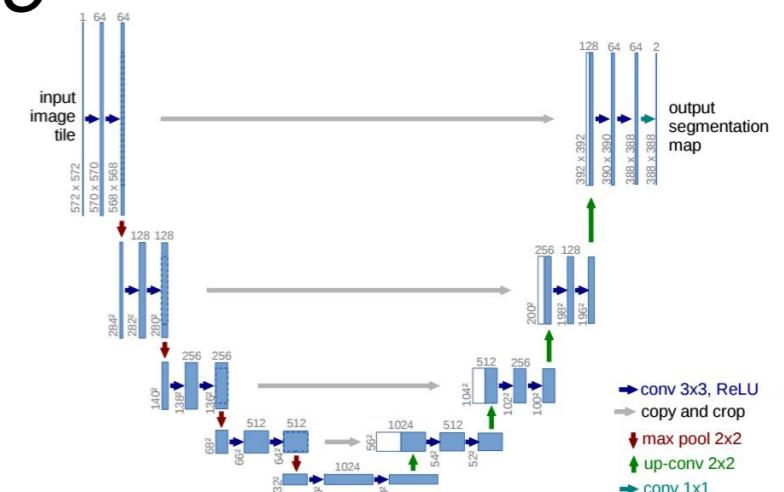
YOLO



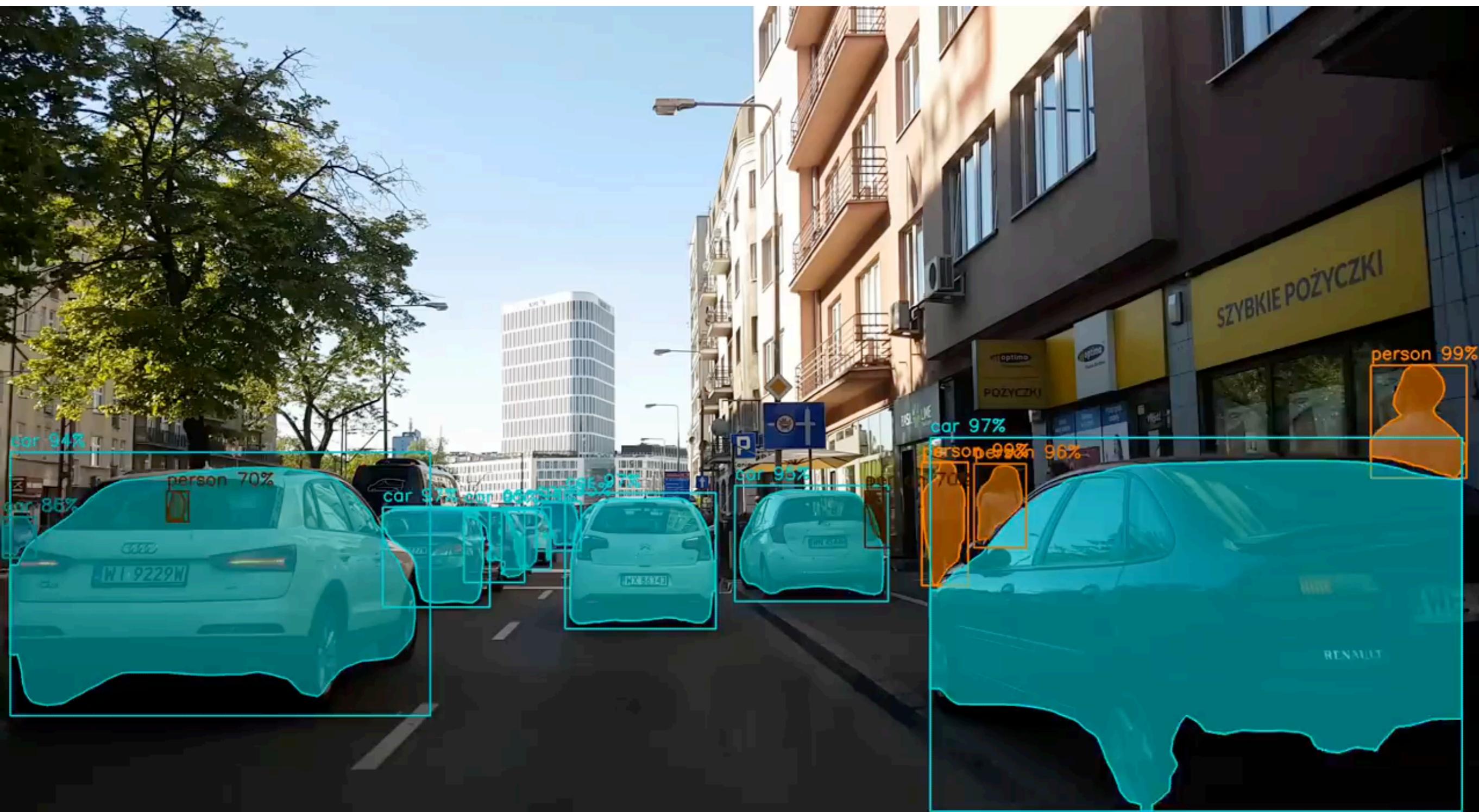
FCN



Hourglass



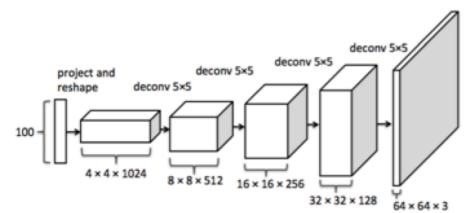
U-Net



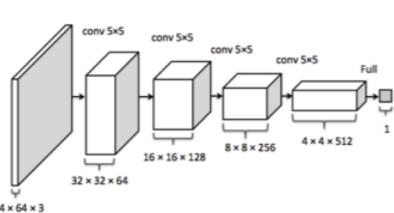
10.4 fps



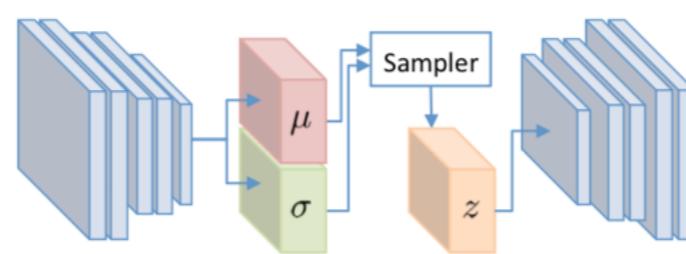
convolutional models for *image* generation



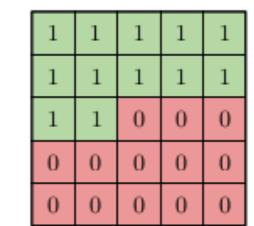
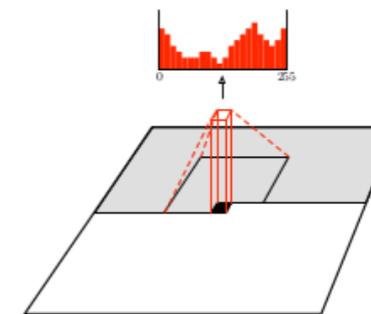
DC-GAN



convolutional VAE



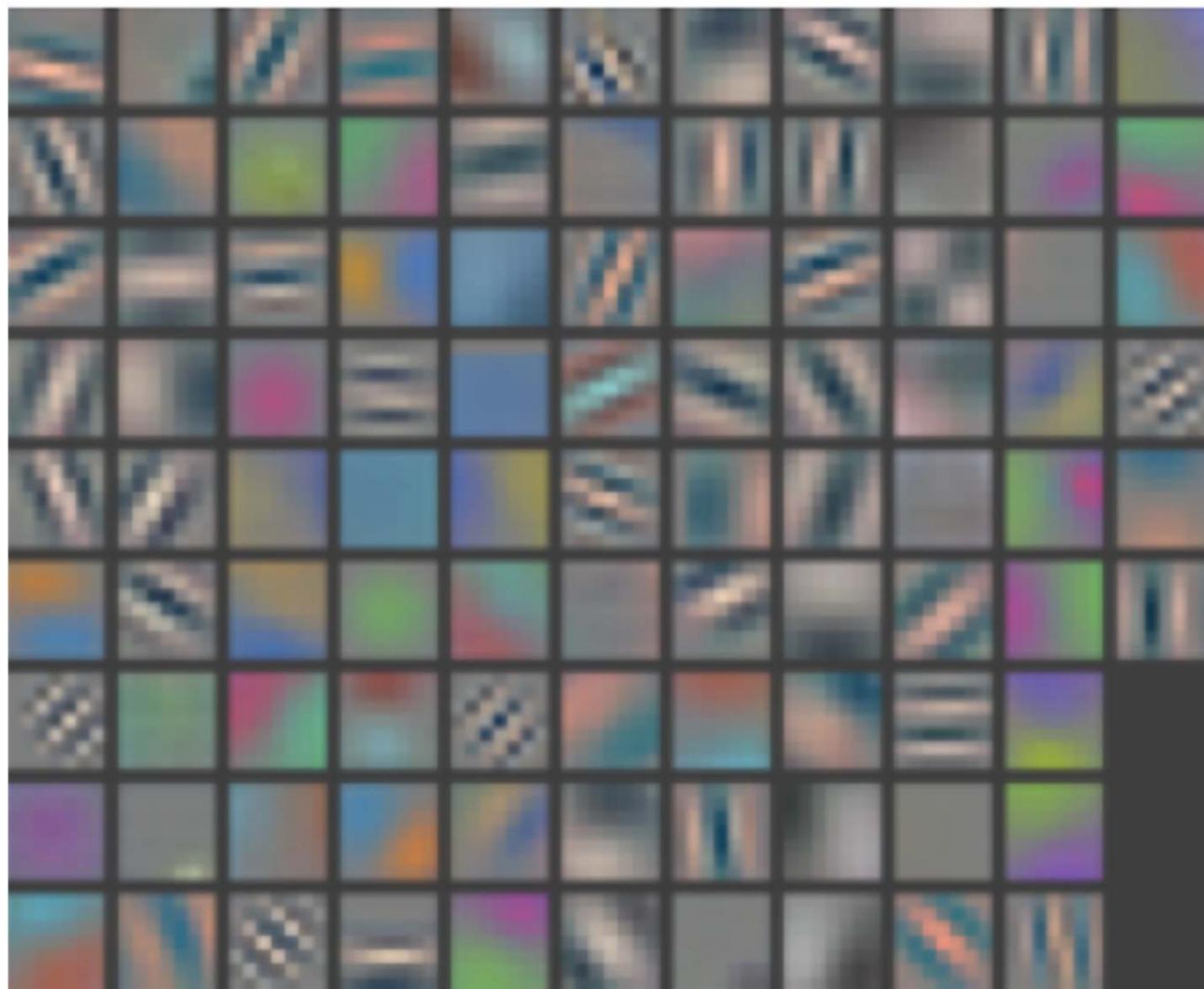
Pixel CNN



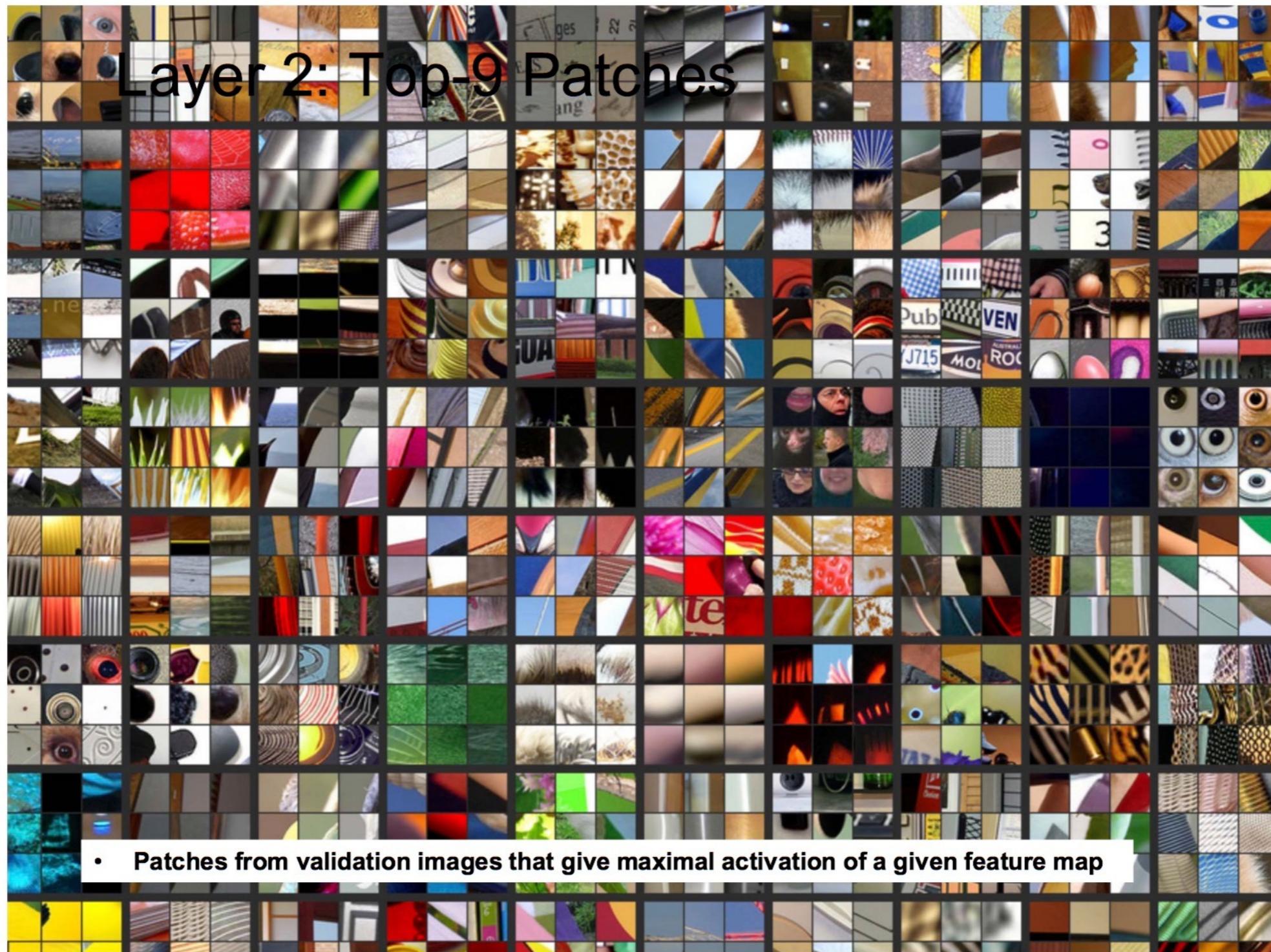
CelebA-HQ
1024 × 1024

Progressive growing

filter visualization



filter visualization



filter visualization



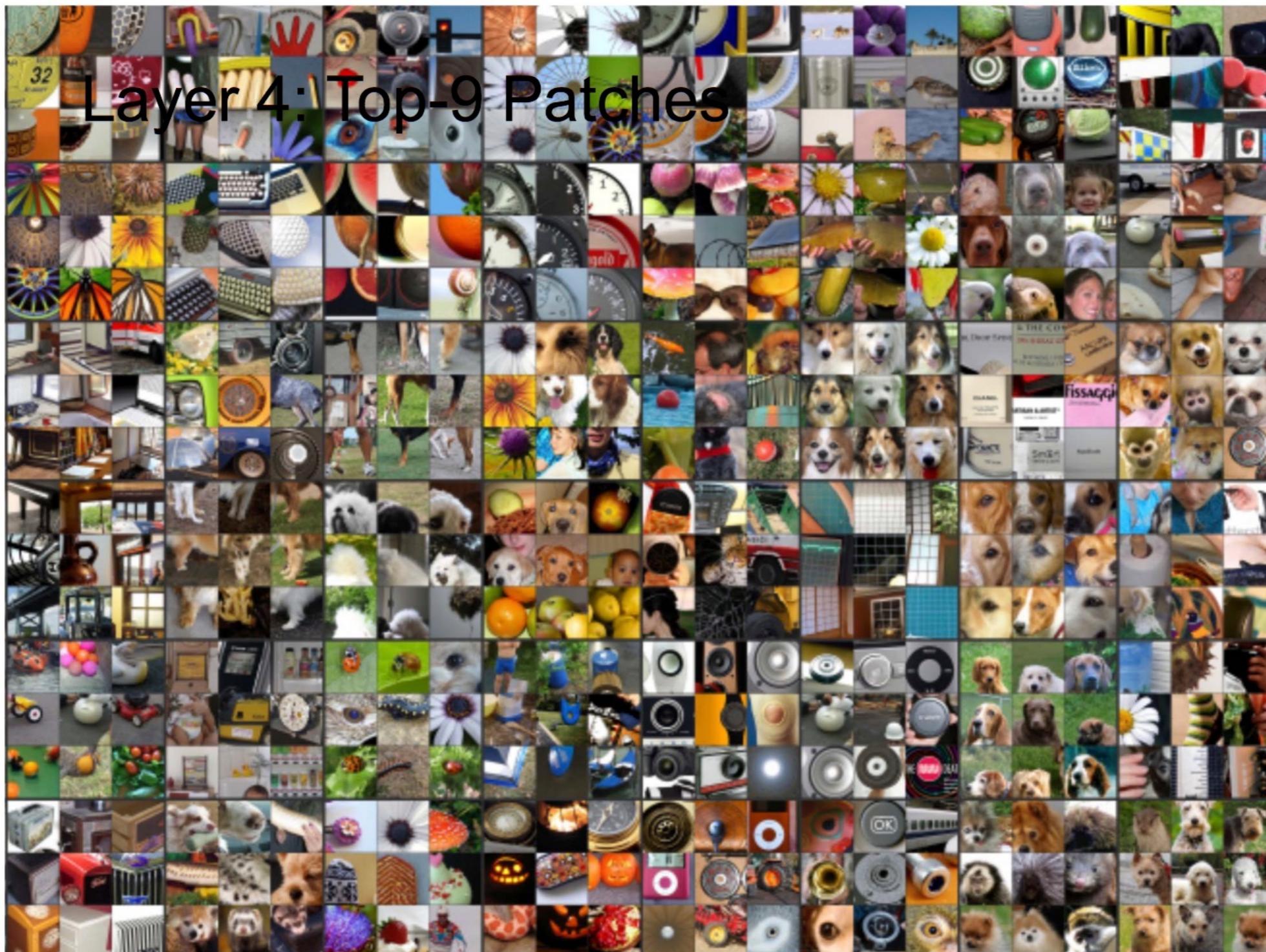
filter visualization



filter visualization



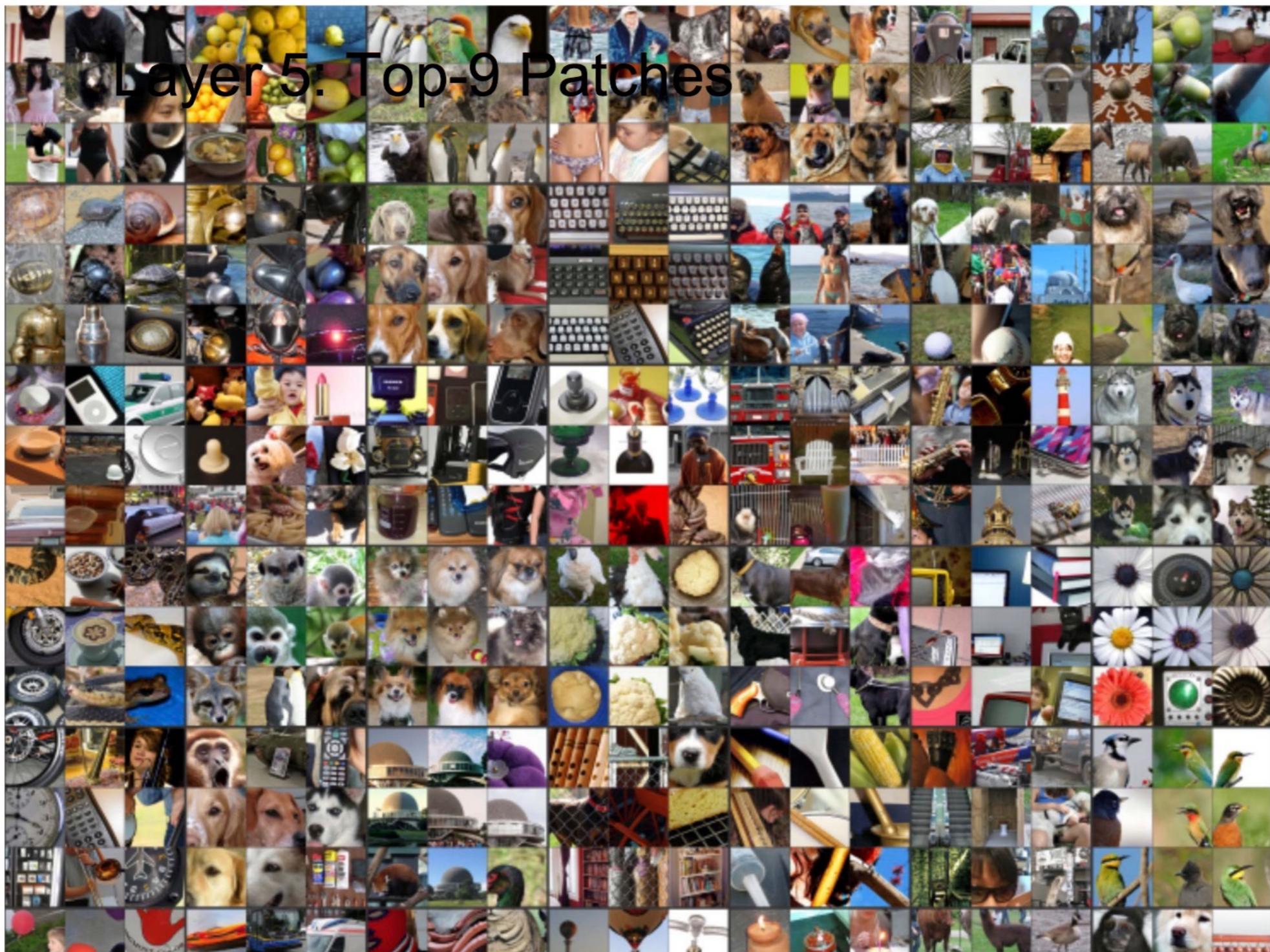
filter visualization



filter visualization



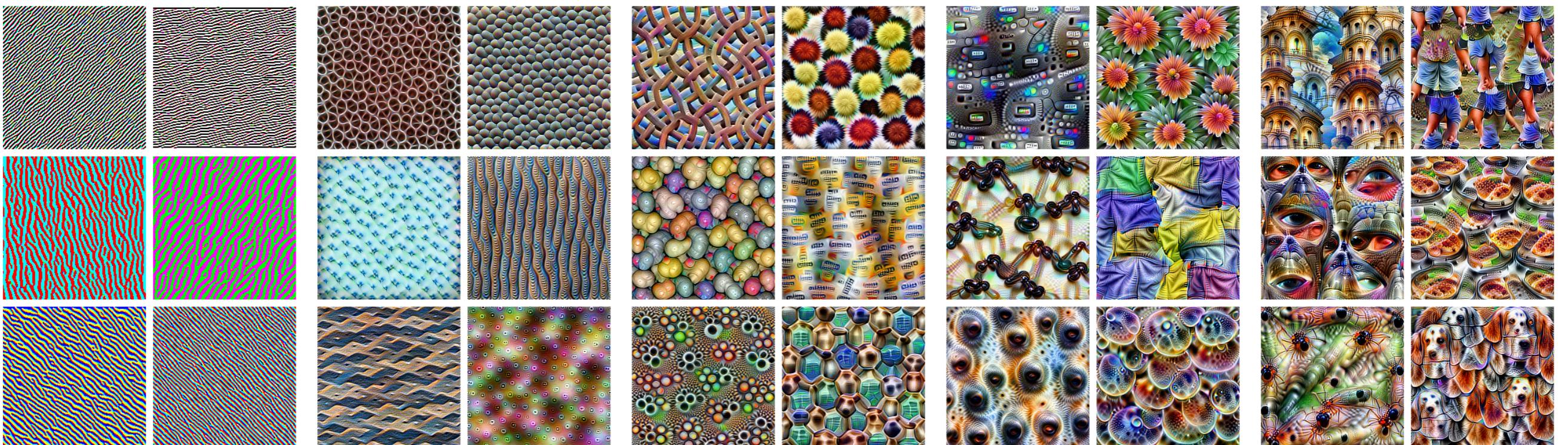
filter visualization



filter visualization



filter visualization



Edges (layer conv2d0)

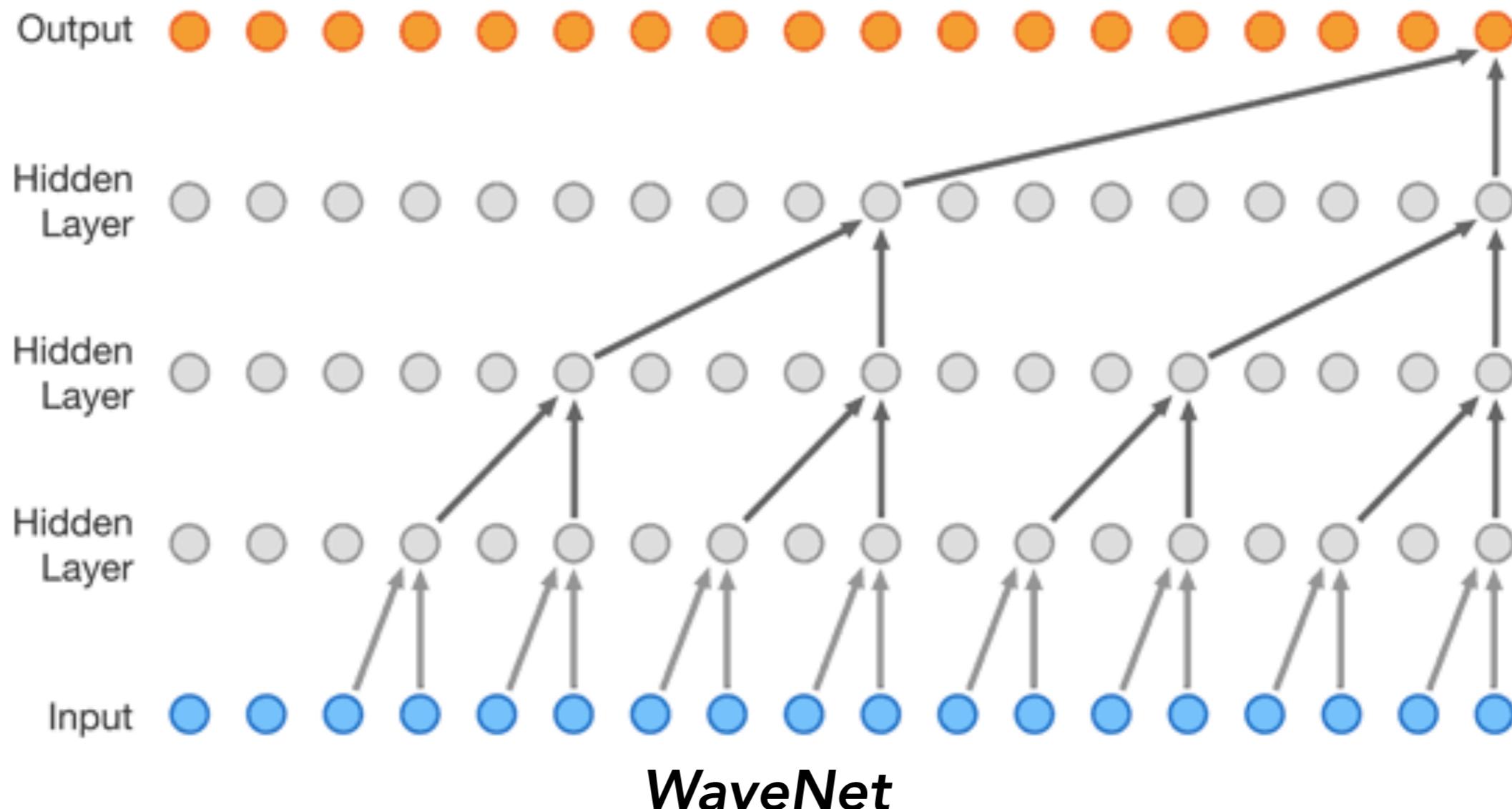
Textures (layer mixed3a)

Patterns (layer mixed4a)

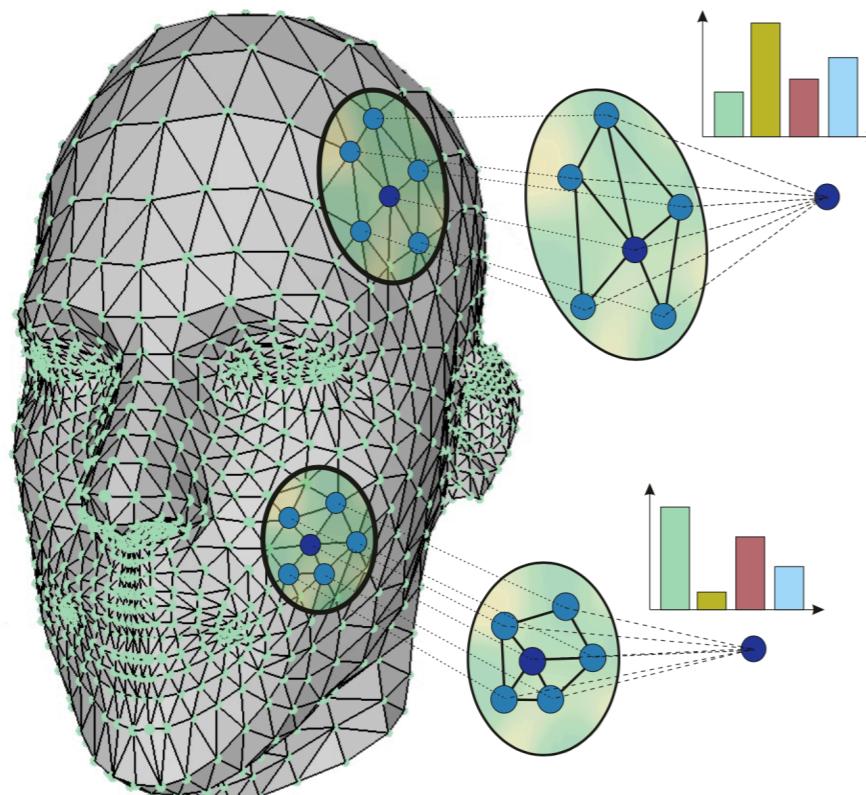
Parts (layers mixed4b & mixed4c)

Objects (layers mixed4d & mixed4e)

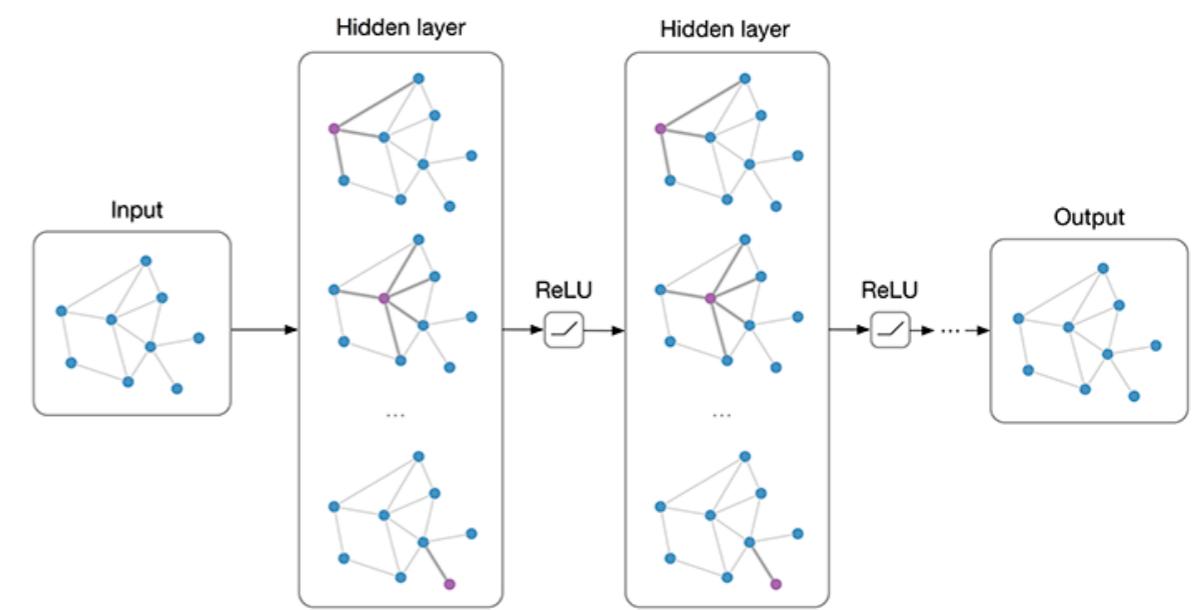
convolutions applied to sequences



convolutions in non-euclidean spaces



Spline CNN



Graph Convolutional Network

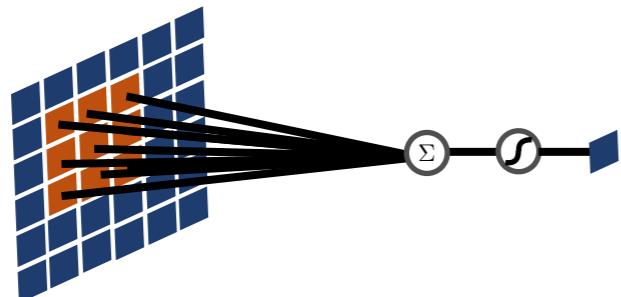
Application to Speech Animation (automated lip-syncing)



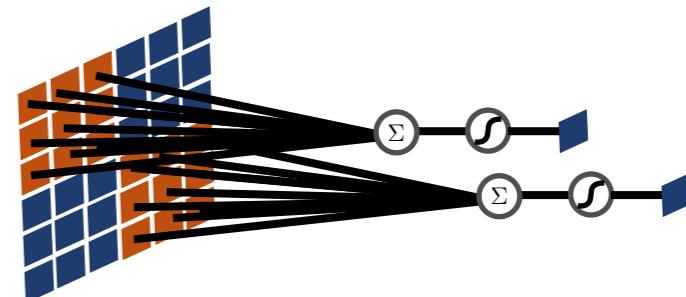
German

recapitulation

we can exploit spatial structure to impose inductive biases on the model



locality



translation invariance

this limits the number of parameters required,
reducing flexibility in reasonable ways

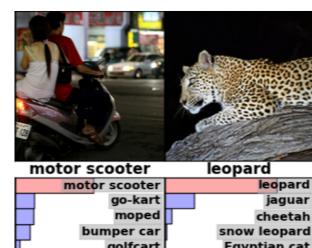
can then scale these models to complex data sets to perform difficult tasks



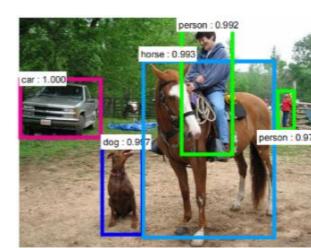
ImageNet



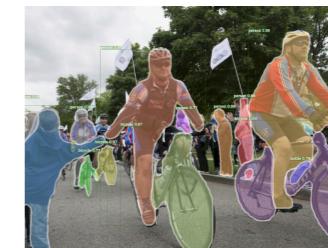
recognition



detection



segmentation

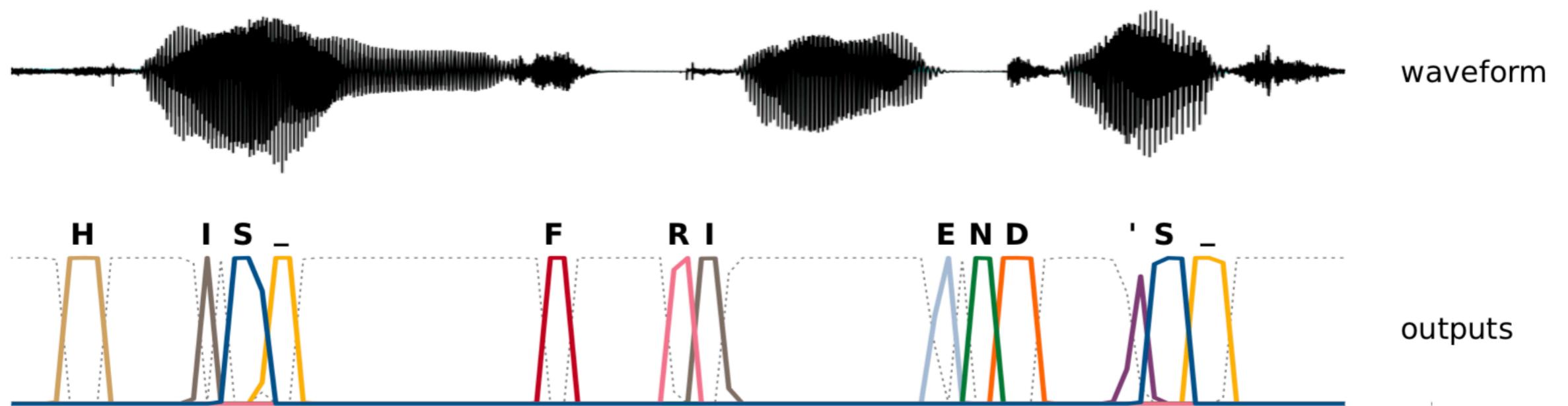


generation



RECURRENT NEURAL NETWORKS

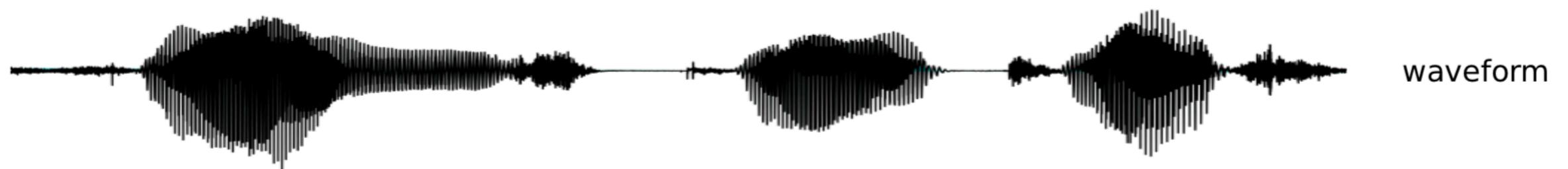
task: speech recognition



Graves & Jaitly, 2014

mapping from input waveform to sequence of characters

the input waveform contains all of the information about the corresponding transcribed text



form a discriminative mapping: $p(\text{text sequence} | \text{---})$

again, there is *nuisance information* in the waveform coming from the speaker's voice characteristics, volume, background, etc.

the mapping is too difficult to define by hand,
need to learn from data

data, label collection



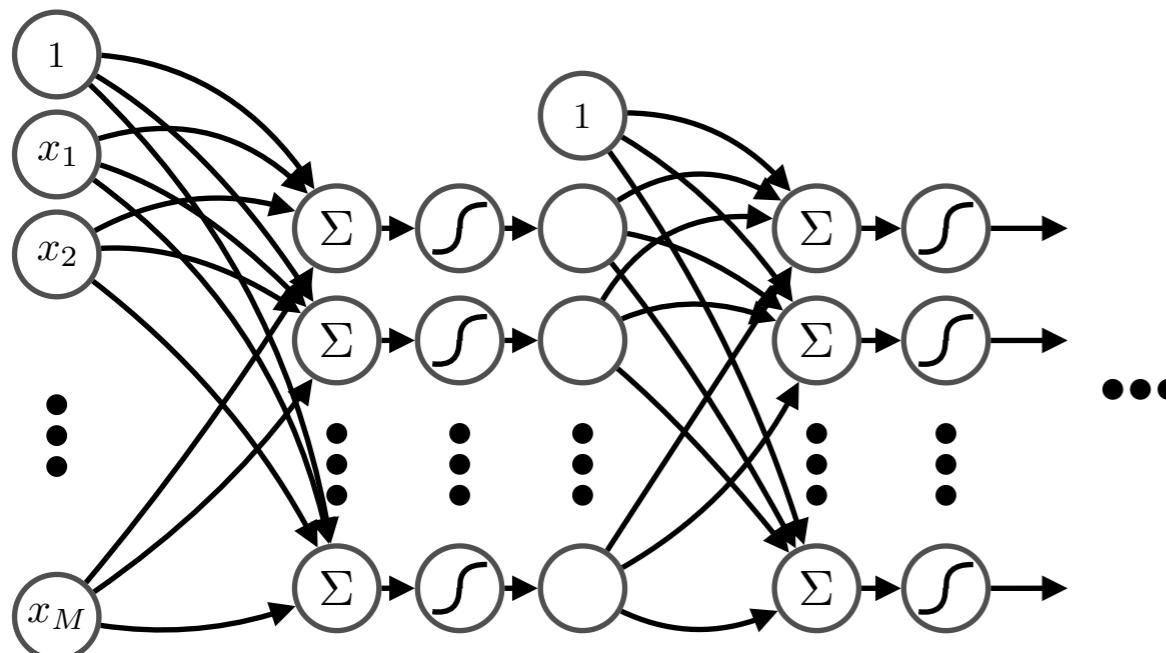
Audio

"OK Google..."

"Hey Siri..."

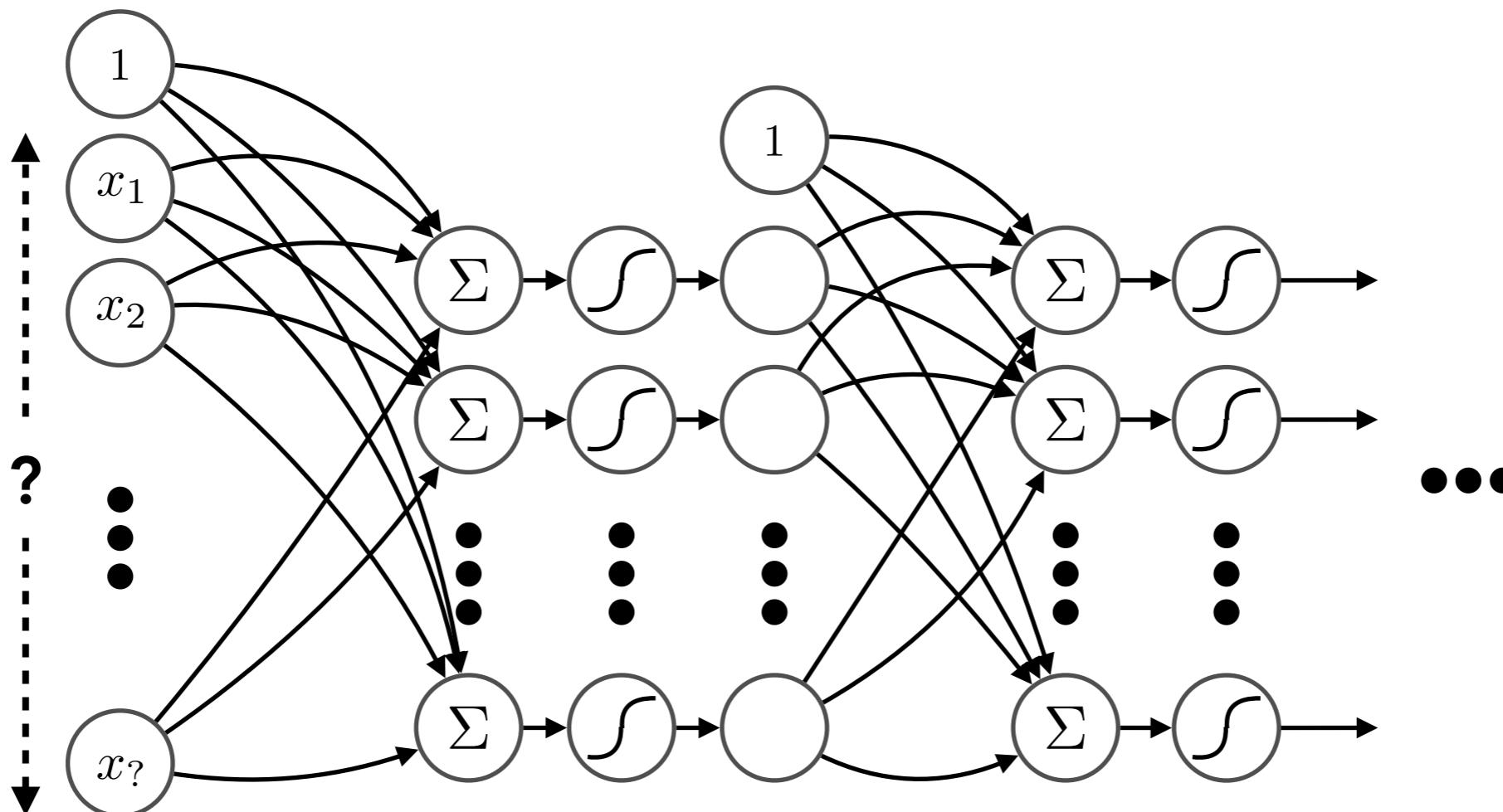
"Yo Alexa..."

Transcriptions



but how do we define the network architecture?

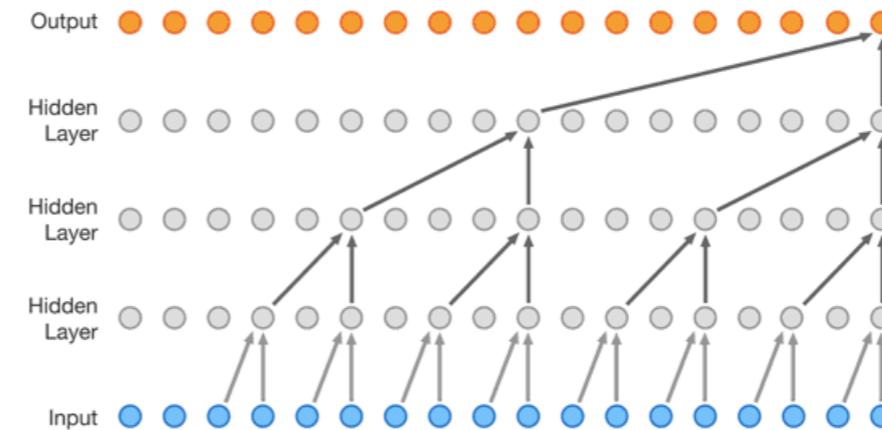
problem: inputs can be of variable size



standard neural networks can only handle data of a fixed input size

wait, but convolutional networks can handle variable input sizes...
can't we just use them?

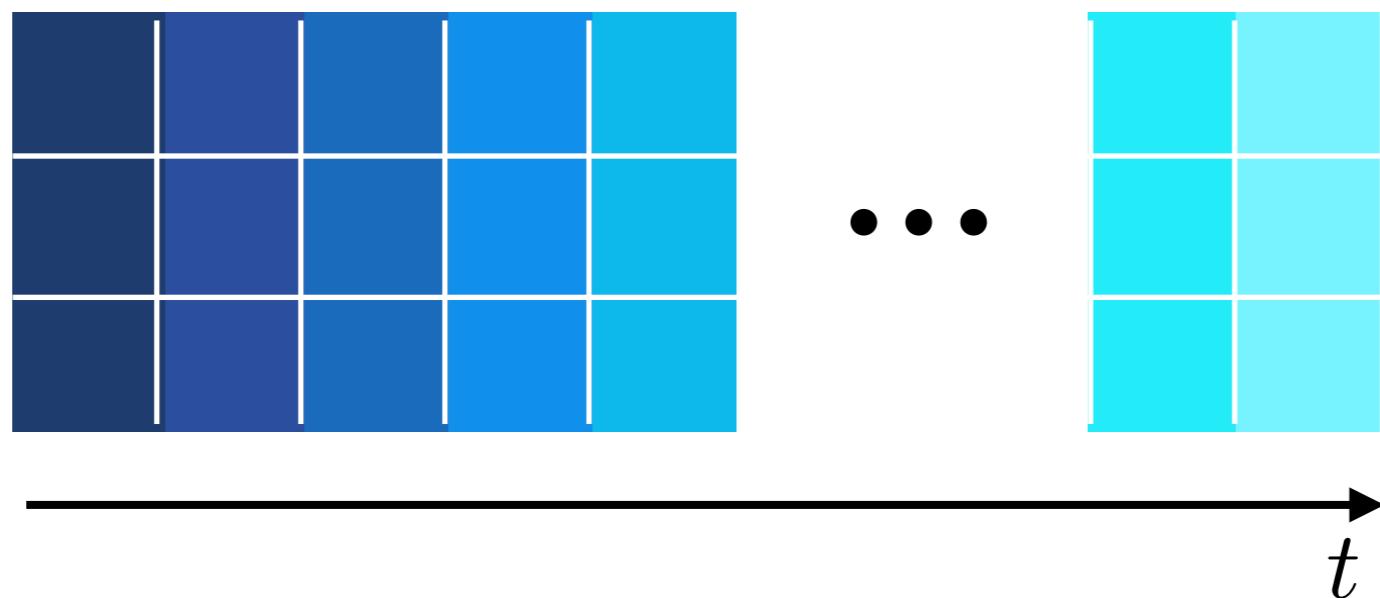
yes, we could



however, this relies on a fixed input window size

we may be able to exploit additional structure in sequence data
to impose better inductive biases

the structure of sequence data



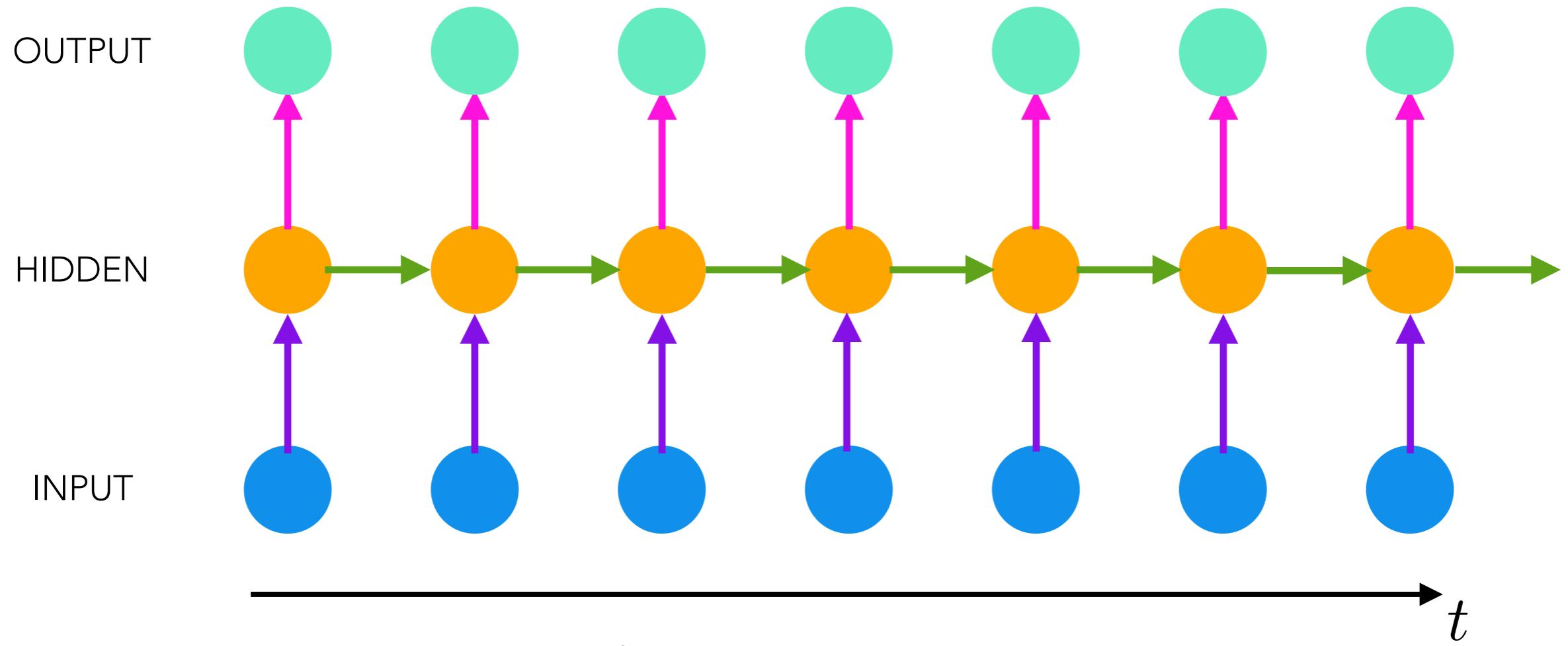
sequence data also tends to obey

locality: nearby regions tend to form stronger patterns

translation invariance: patterns are relative rather than absolute

but has a single axis on which extended patterns occur

to mirror the sequential structure of the data,
we can process the data sequentially

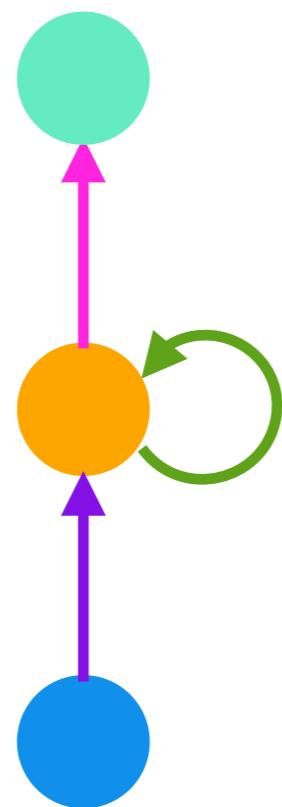


maintain an *internal representation* during processing

→ potentially *infinite* effective input window

→ fixed number of parameters

a **recurrent neural network (RNN)** can be expressed as



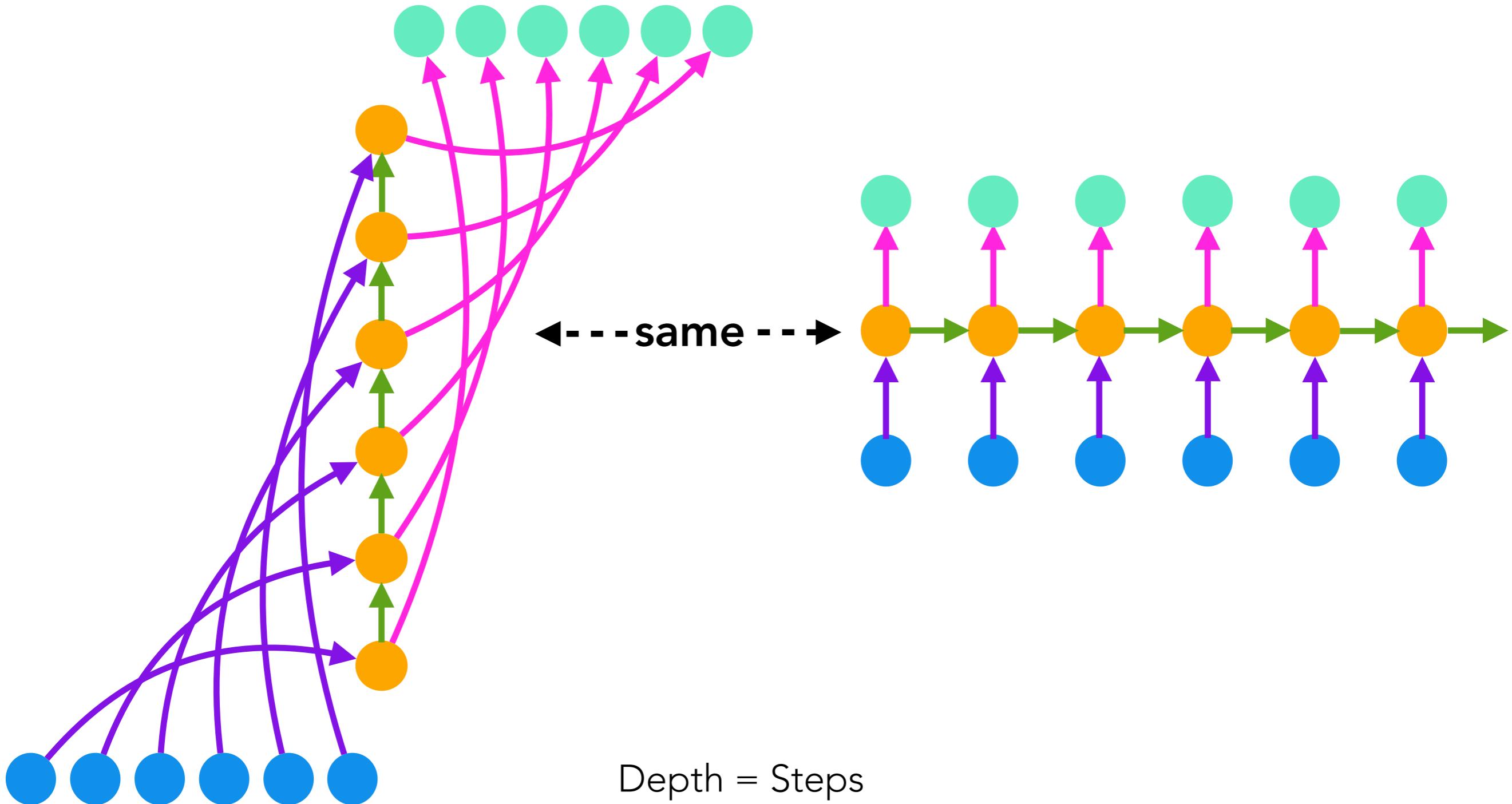
Hidden State

$$\mathbf{h}_t = \sigma(\mathbf{W}_h^\top [\mathbf{h}_{t-1}, \mathbf{x}_t])$$

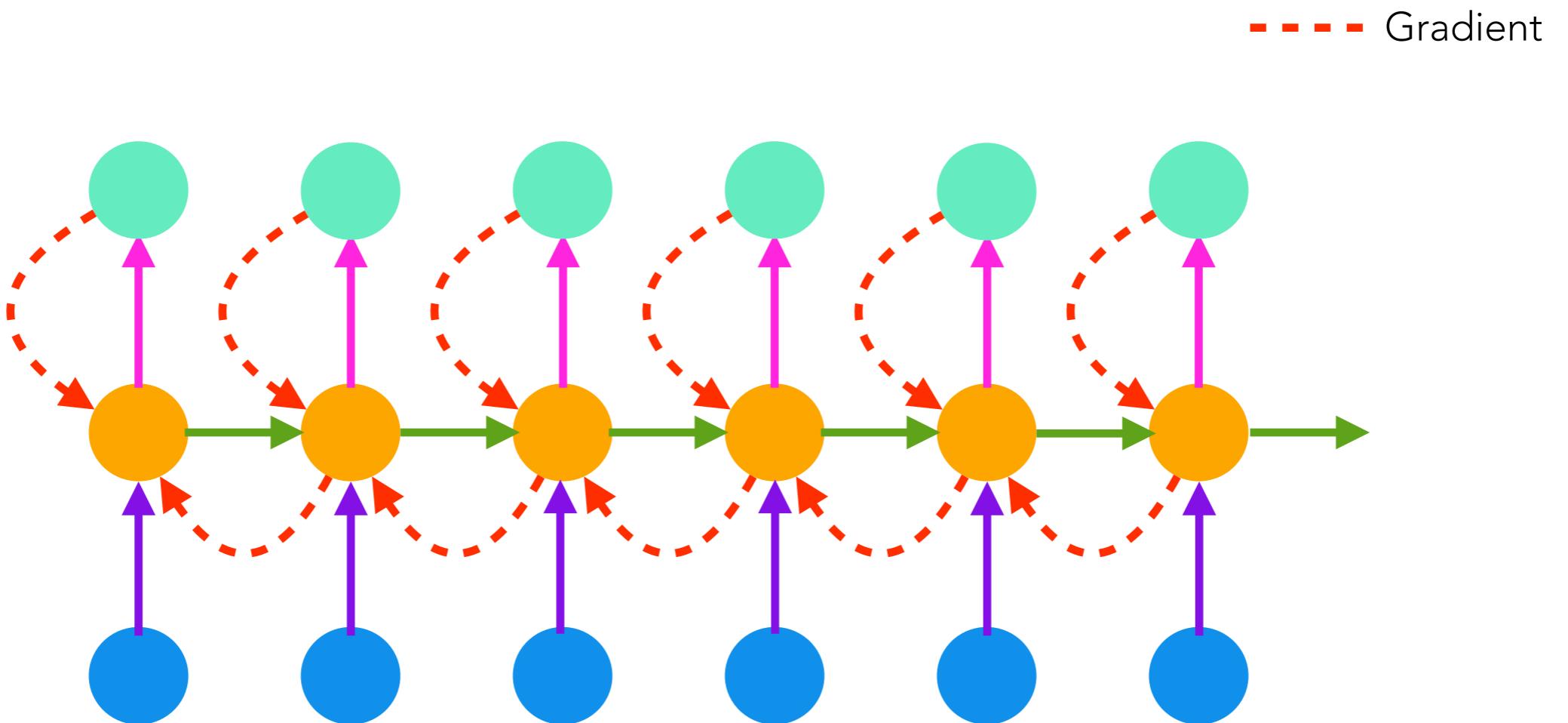
Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_y^\top \mathbf{h}_t)$$

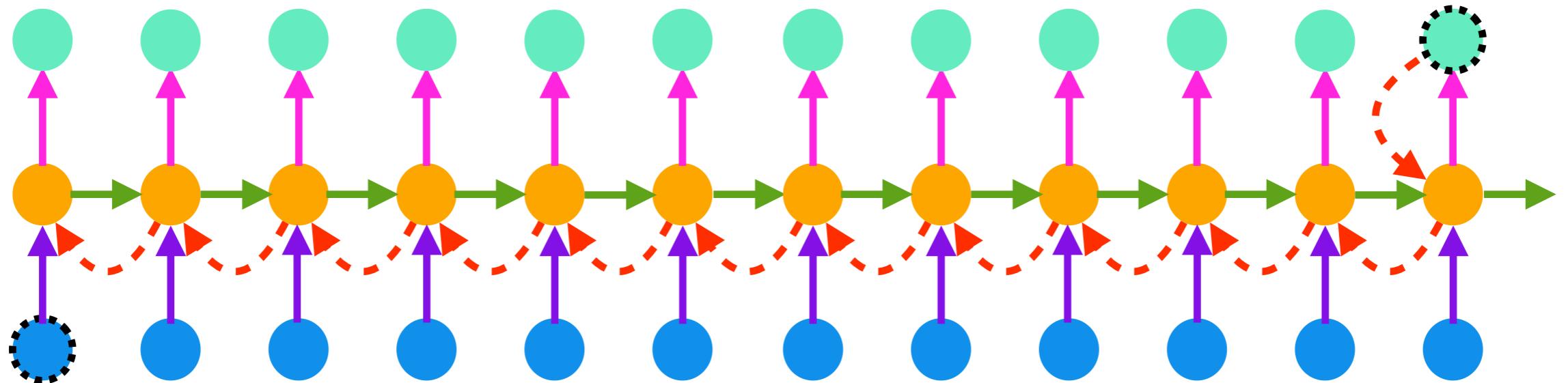
basic recurrent networks are also a **special case**
of standard neural networks with skip connections and shared weights



therefore, we can use standard backpropagation to train,
resulting in ***backpropagation through time (BPTT)***



primary difficulty of training RNNs involves propagating information over long horizons

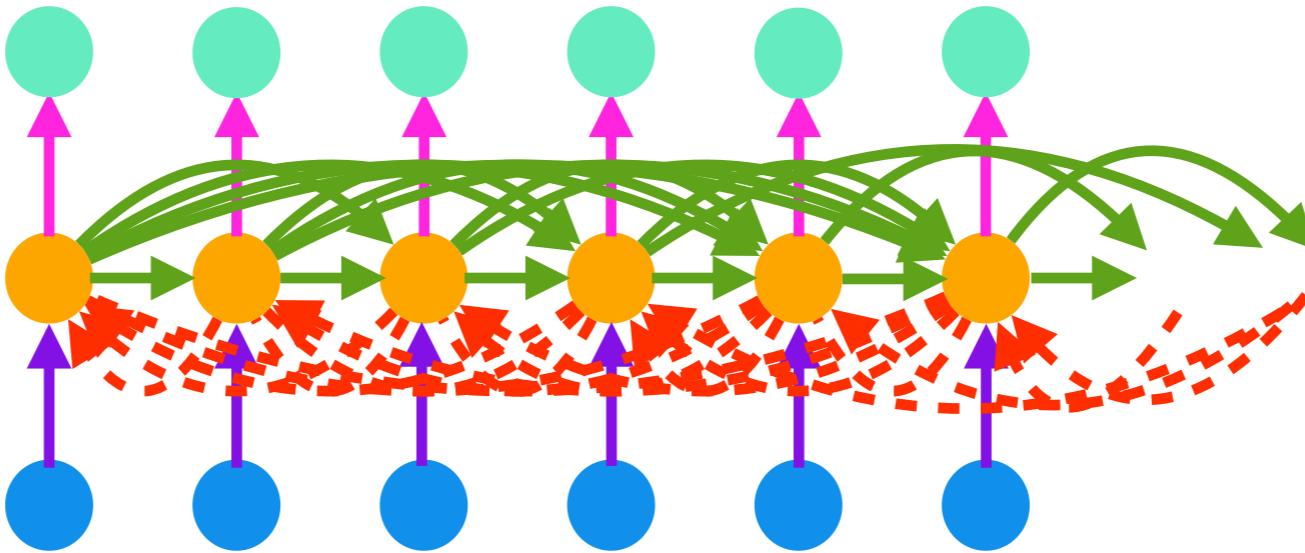


e.g. input at one step is predictive of output at much *later* step

learning extended sequential dependencies
requires propagating gradients over long horizons

- vanishing / exploding gradients
- large memory/computational footprint

naïve attempt to fix information propagation issue



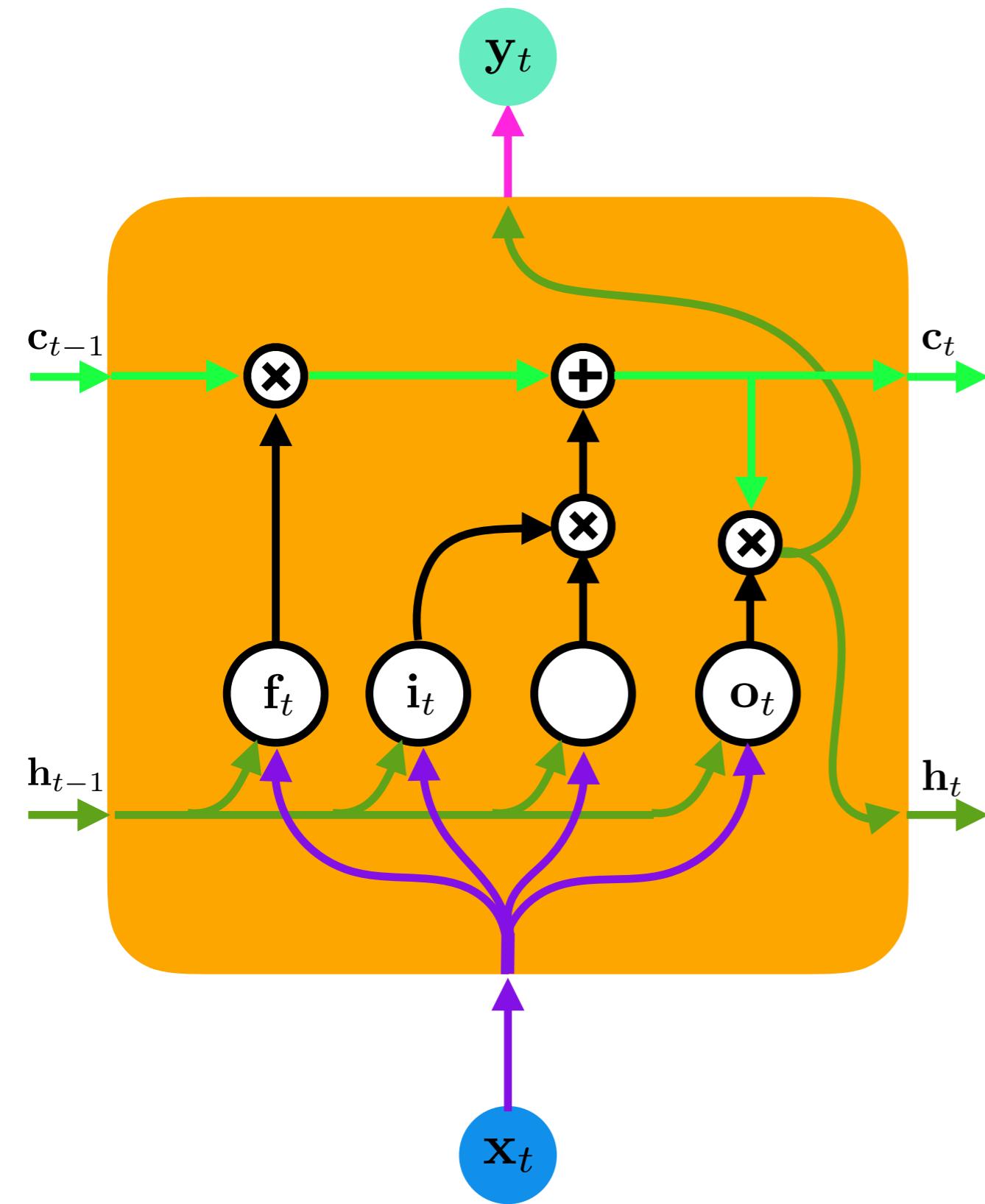
add skip connections across steps

information, gradients can propagate more easily

but...

- increases computation
- must set limit on window size

add trainable ***memory*** to the network
read from and write to “***cell***” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

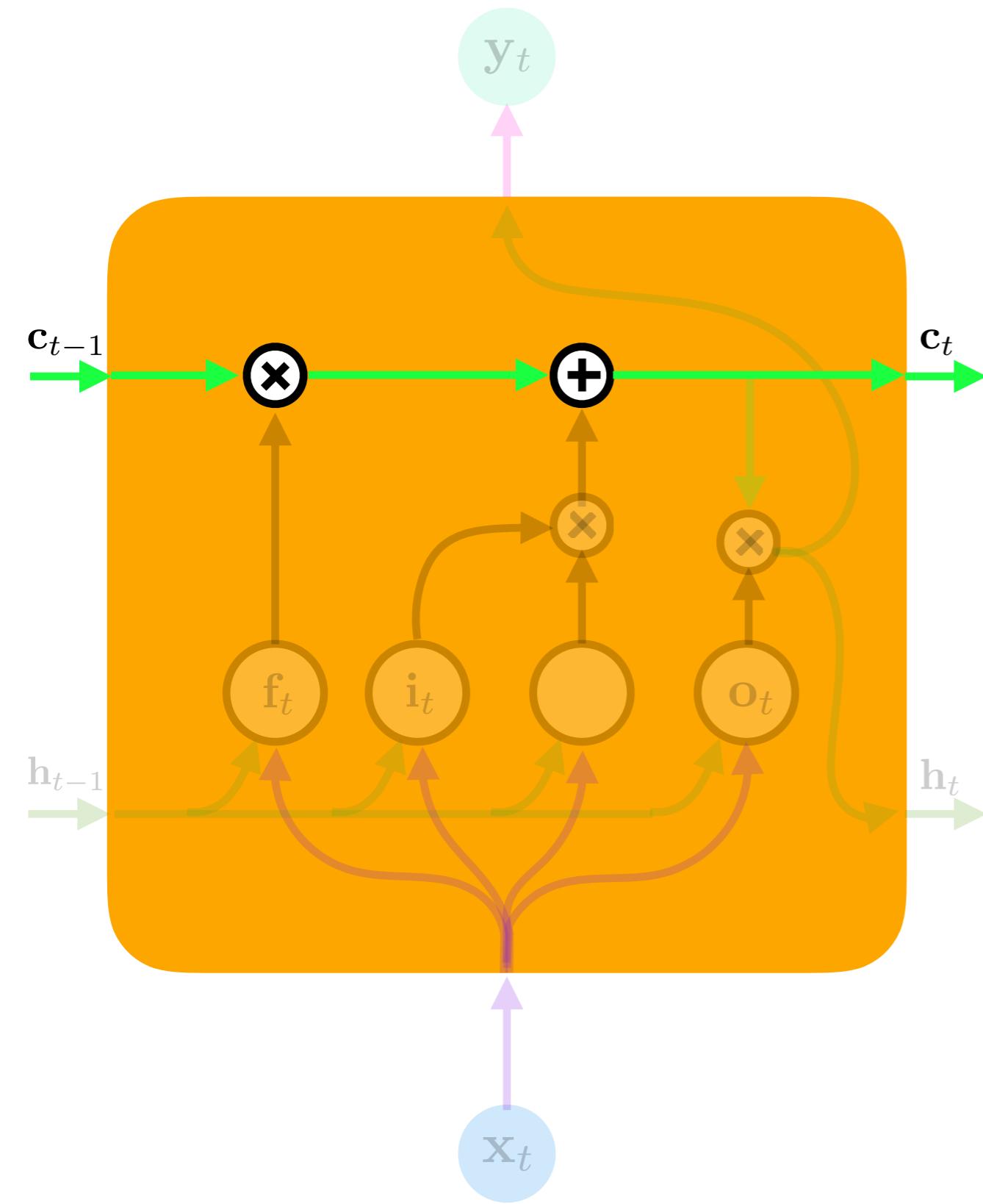
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable **memory** to the network
read from and write to “**cell**” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^T[h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^T[h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^T[h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^T[h_{t-1}, x_t])$$

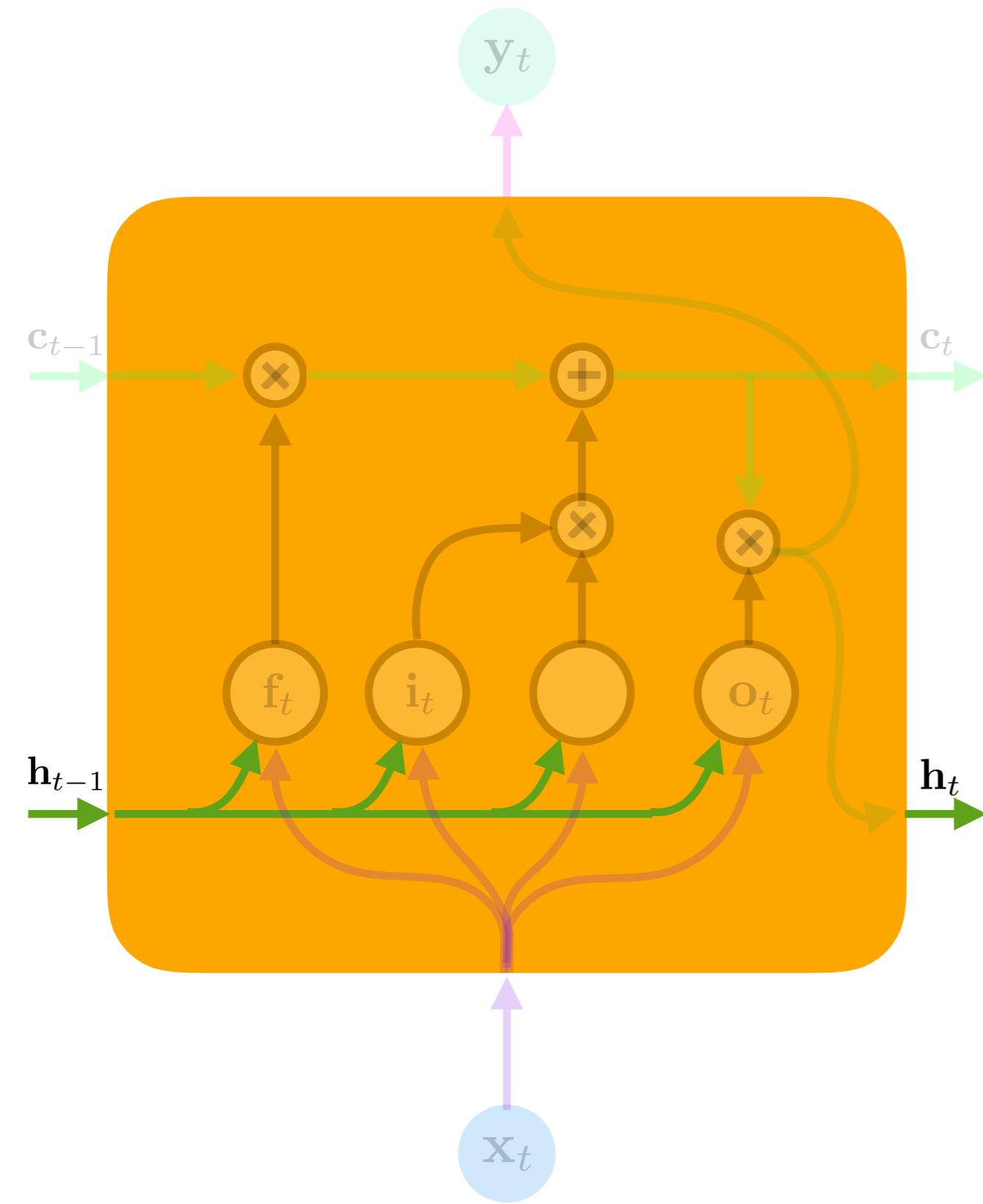
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^T h_t)$$

add trainable ***memory*** to the network
read from and write to “***cell***” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

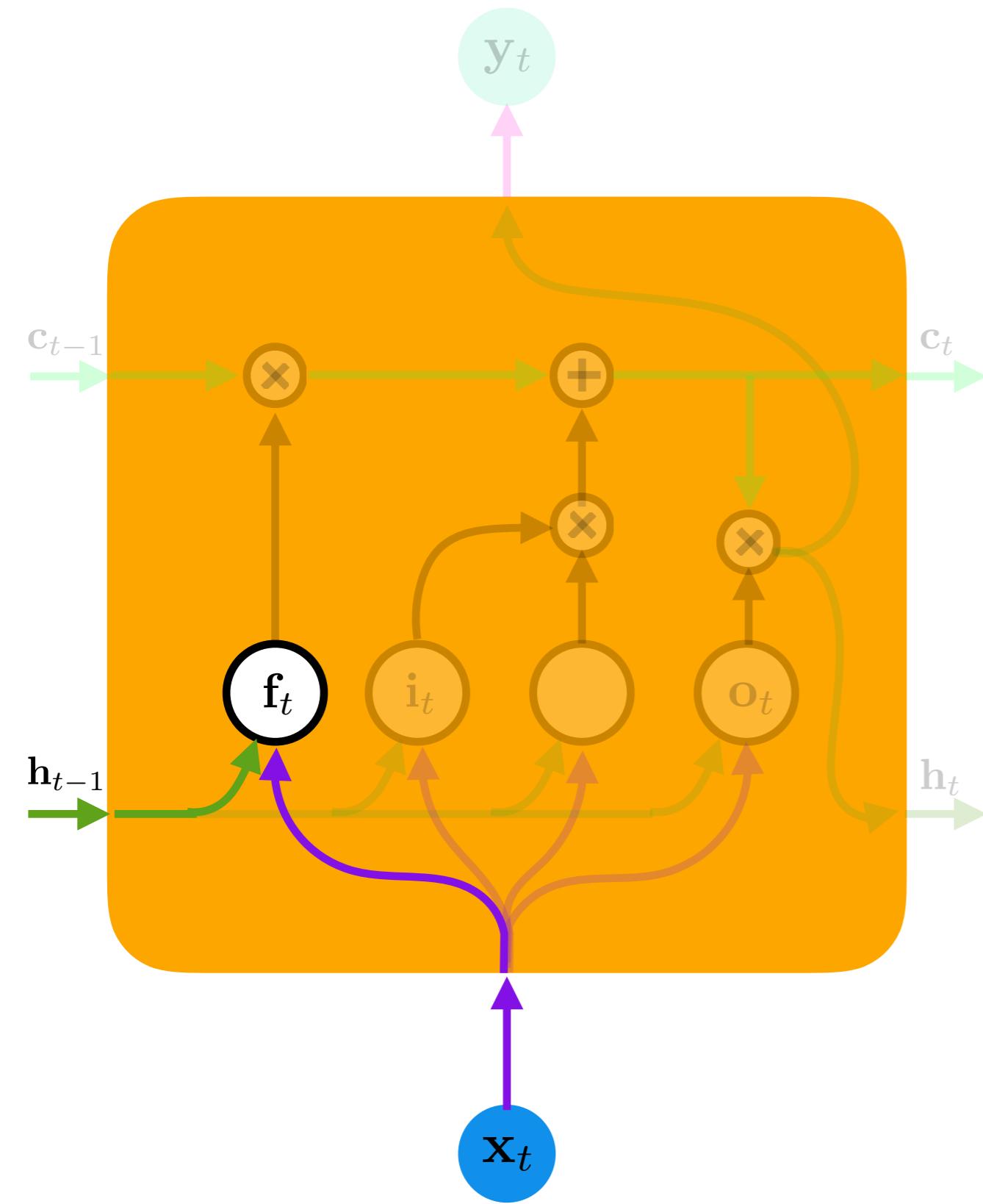
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable **memory** to the network
read from and write to “**cell**” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

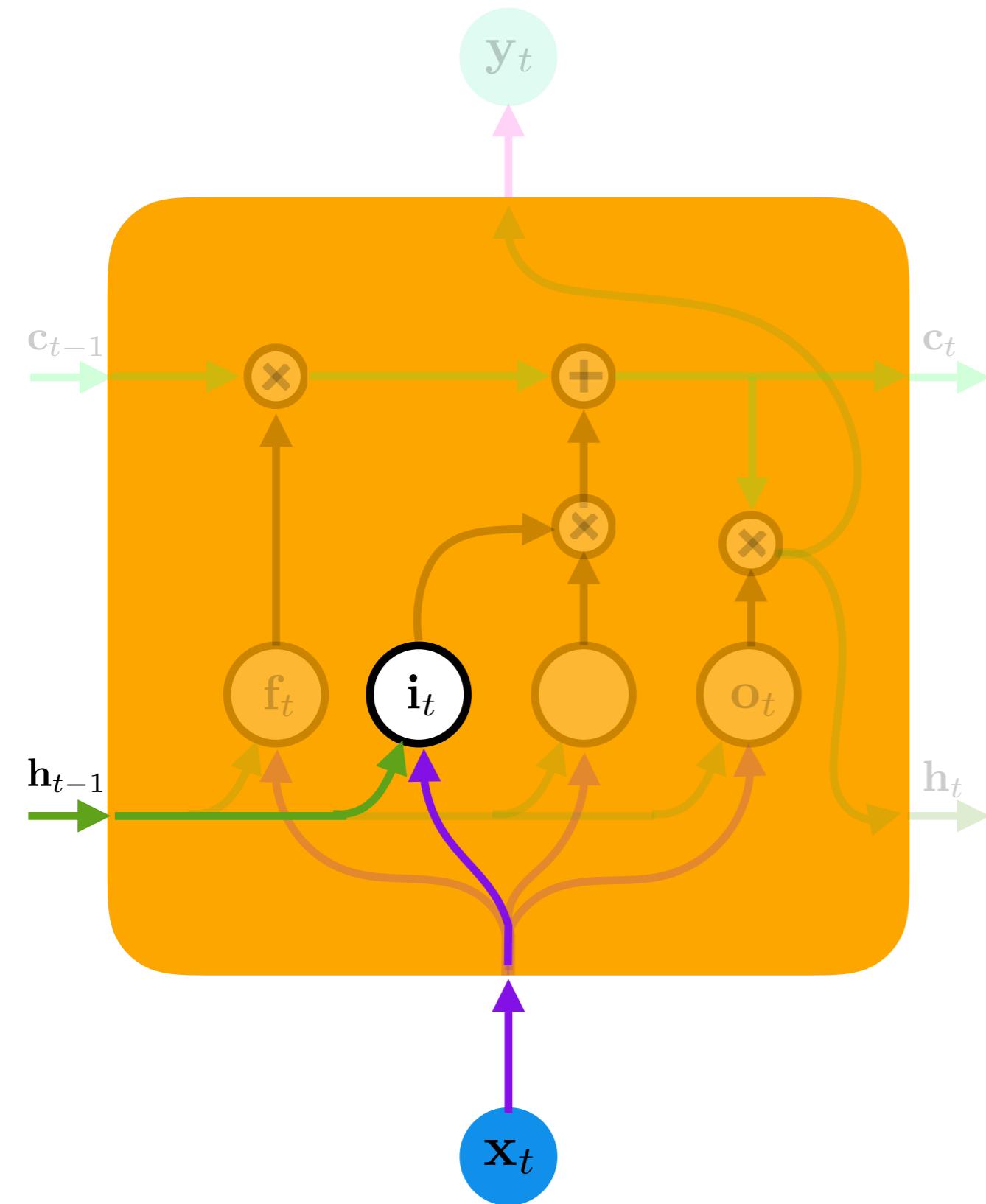
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable ***memory*** to the network
read from and write to “***cell***” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

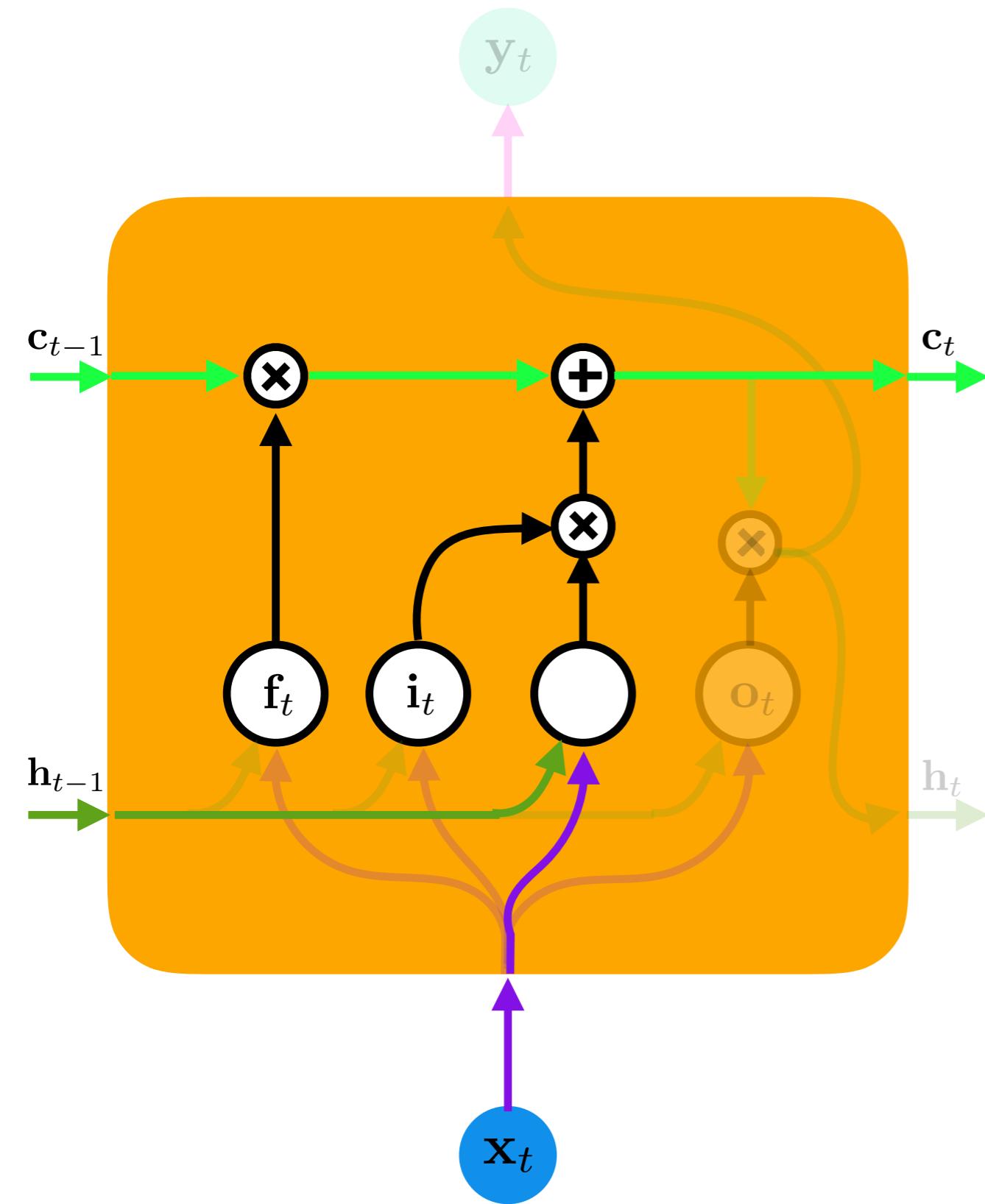
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable **memory** to the network
read from and write to “**cell**” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

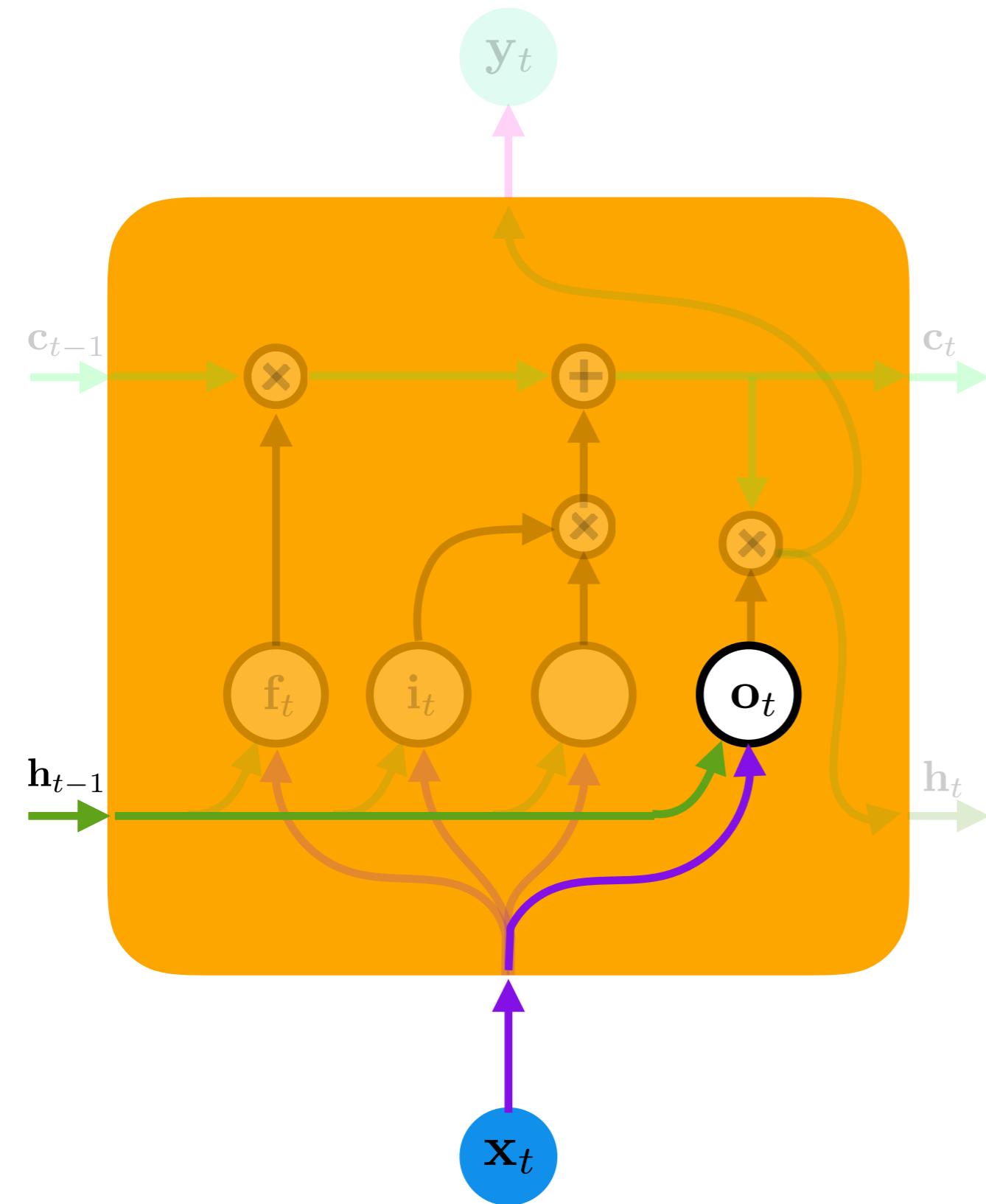
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable **memory** to the network
read from and write to “**cell**” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

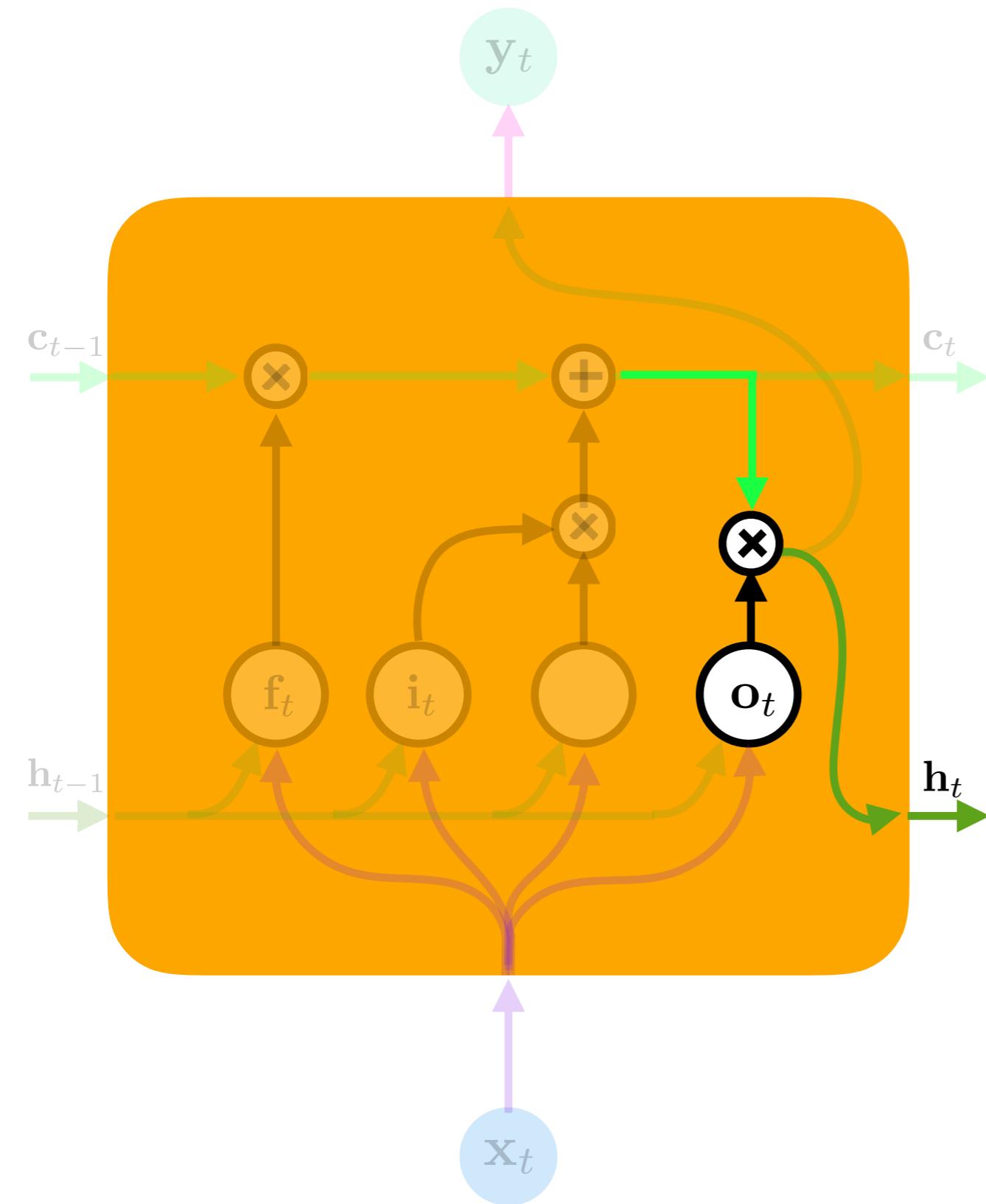
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable ***memory*** to the network
read from and write to “***cell***” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^\top [h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^\top [h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^\top [h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^\top [h_{t-1}, x_t])$$

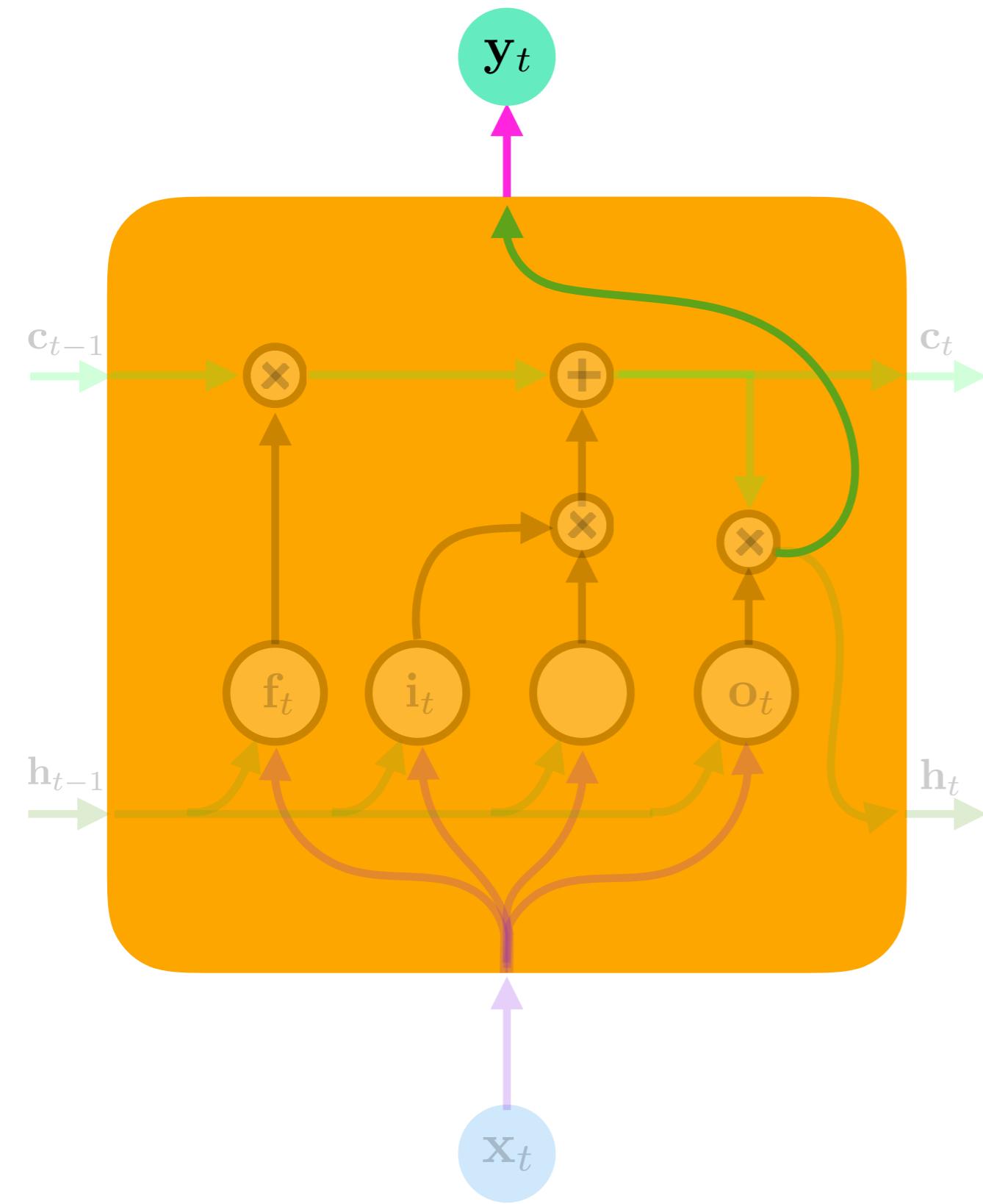
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

Output

$$y_t = \sigma(W_y^\top h_t)$$

add trainable ***memory*** to the network
read from and write to “***cell***” state



Long Short-Term Memory (LSTM)

Forget Gate

$$f_t = \sigma(W_f^T[h_{t-1}, x_t])$$

Input Gate

$$i_t = \sigma(W_i^T[h_{t-1}, x_t])$$

Cell State

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c^T[h_{t-1}, x_t])$$

Output Gate

$$o_t = \sigma(W_o^T[h_{t-1}, x_t])$$

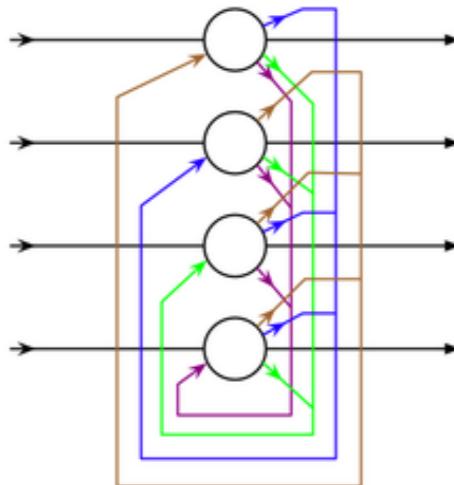
Hidden State

$$h_t = o_t \odot \tanh(c_t)$$

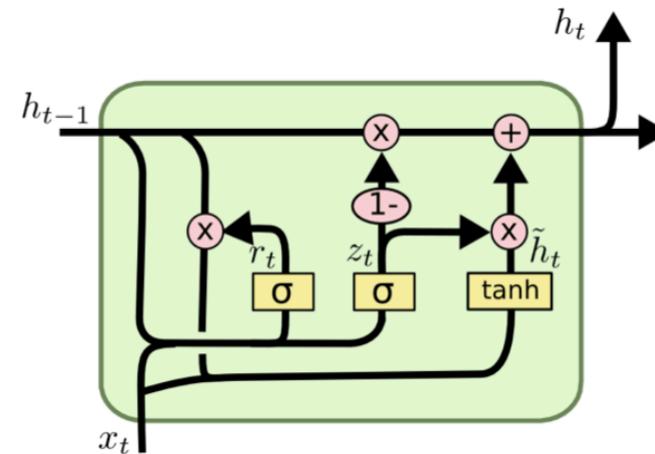
Output

$$y_t = \sigma(W_y^T h_t)$$

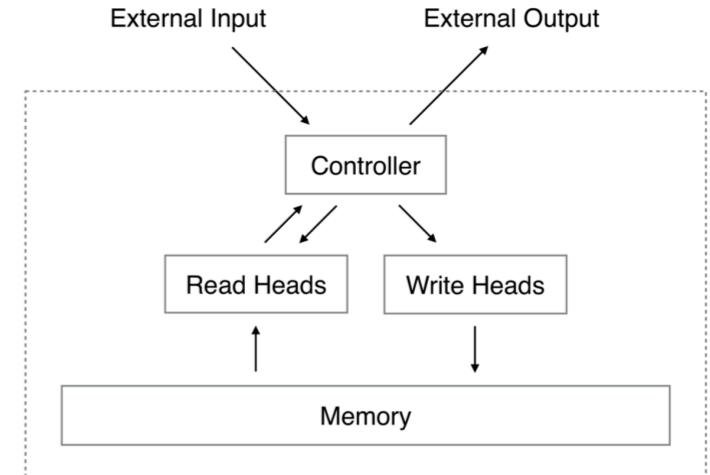
memory networks



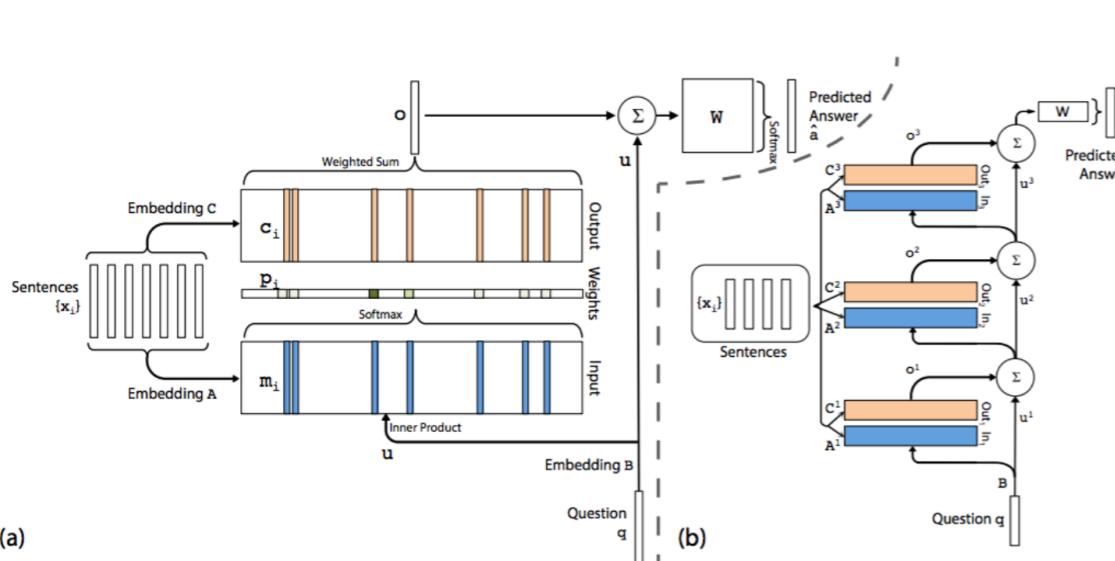
Hopfield Network
Hopfield, 1982



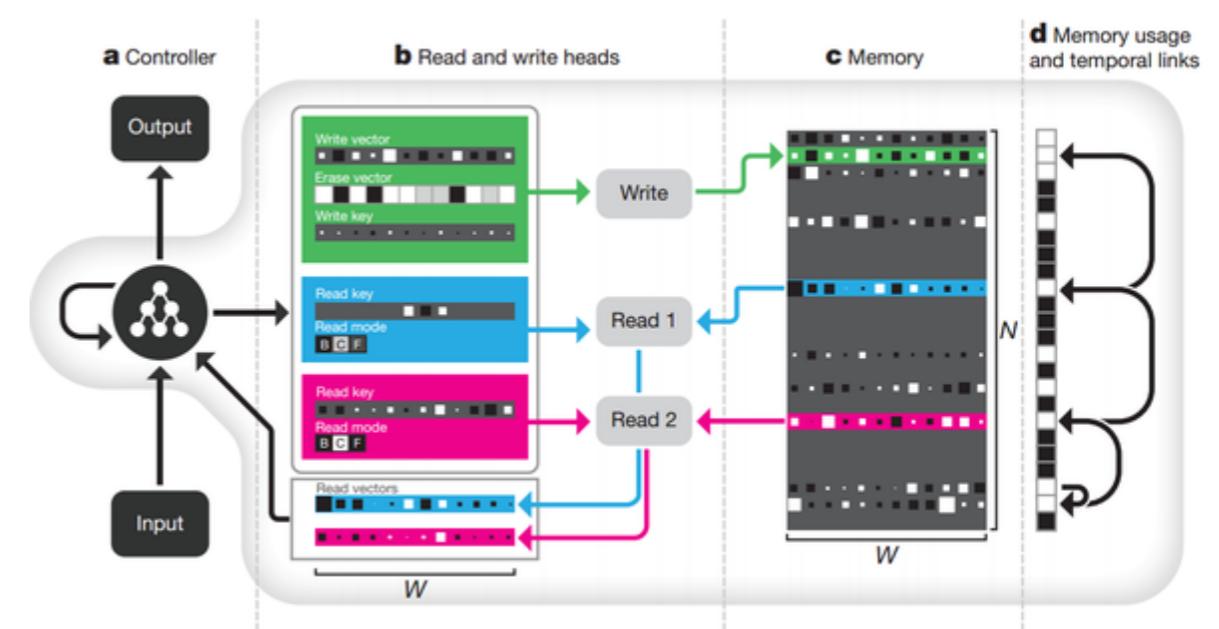
Gated Recurrent Unit (GRU)
Cho et al., 2014



Neural Turing Machine (NTM)
Graves et al., 2014



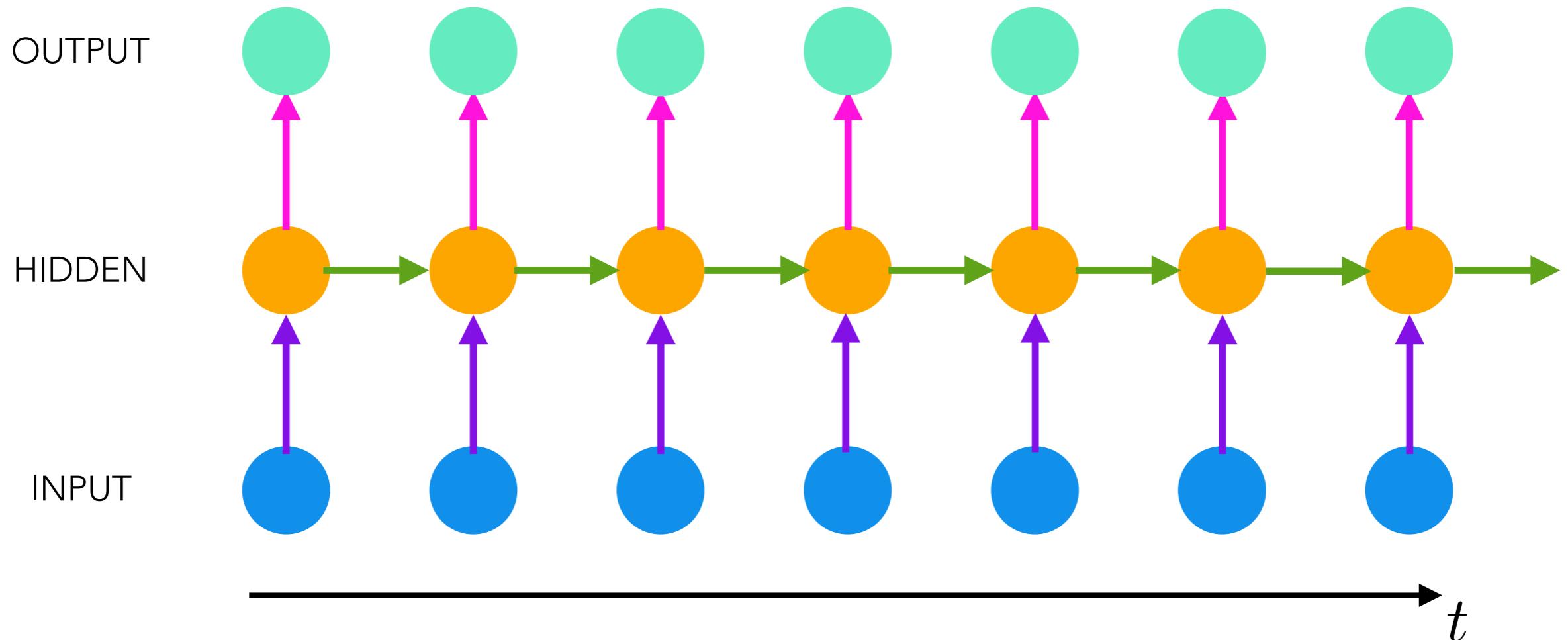
Memory Networks (MemNN)
Weston et al., 2015



Differentiable Neural Computer (DNC)
Graves, Wayne, et al., 2016

bi-directional RNNs

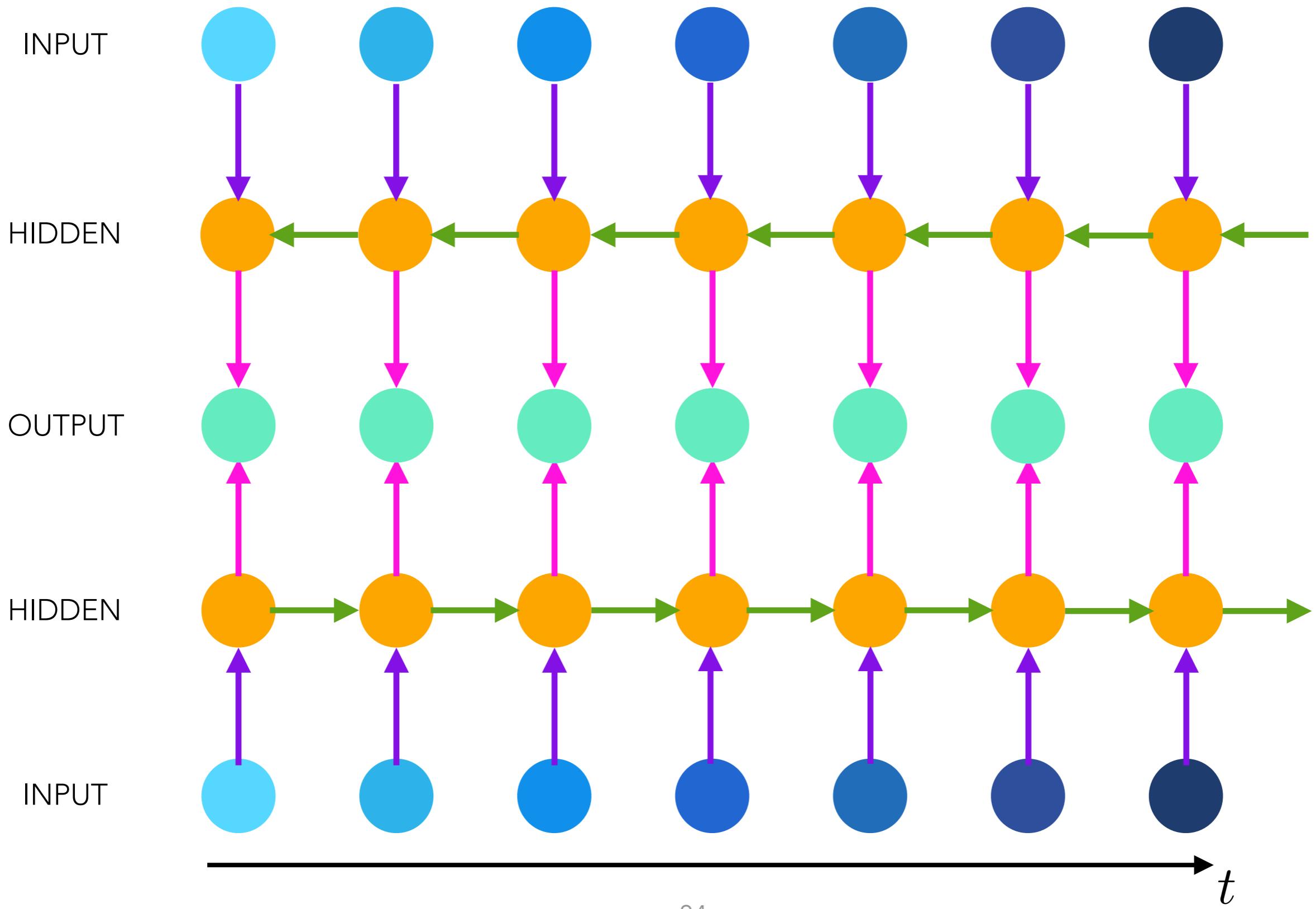
up until now, we have considered the output of the network to only be a function of the preceding inputs (***filtering***)



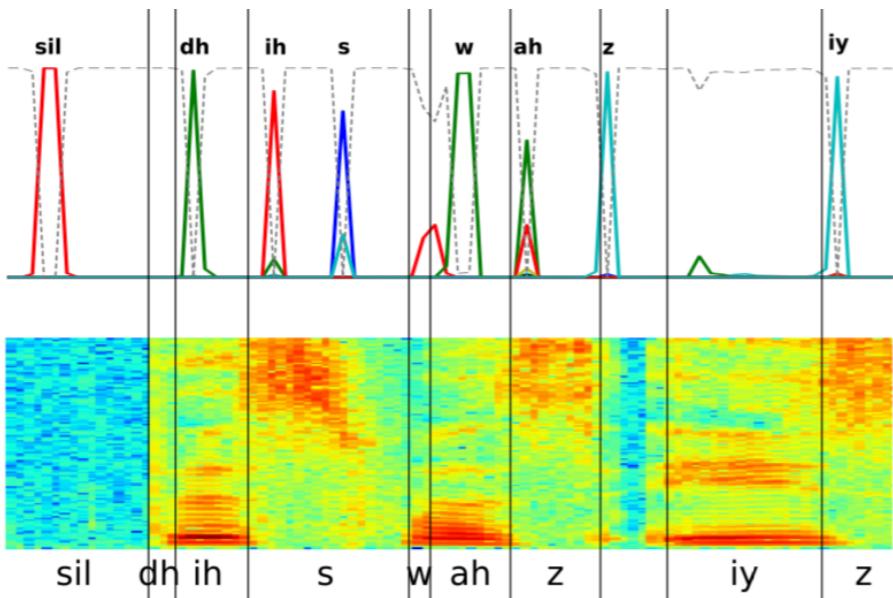
but future inputs may help in determining this output (***smoothing***)

can we make the output a function of both the future and the past inputs?

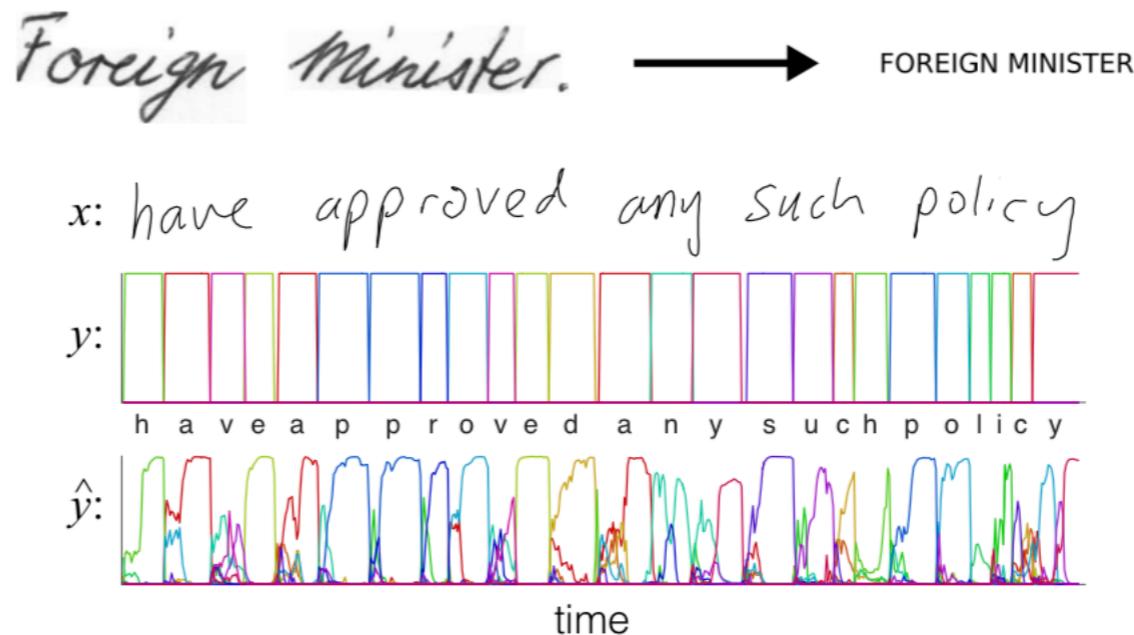
bi-directional RNNs



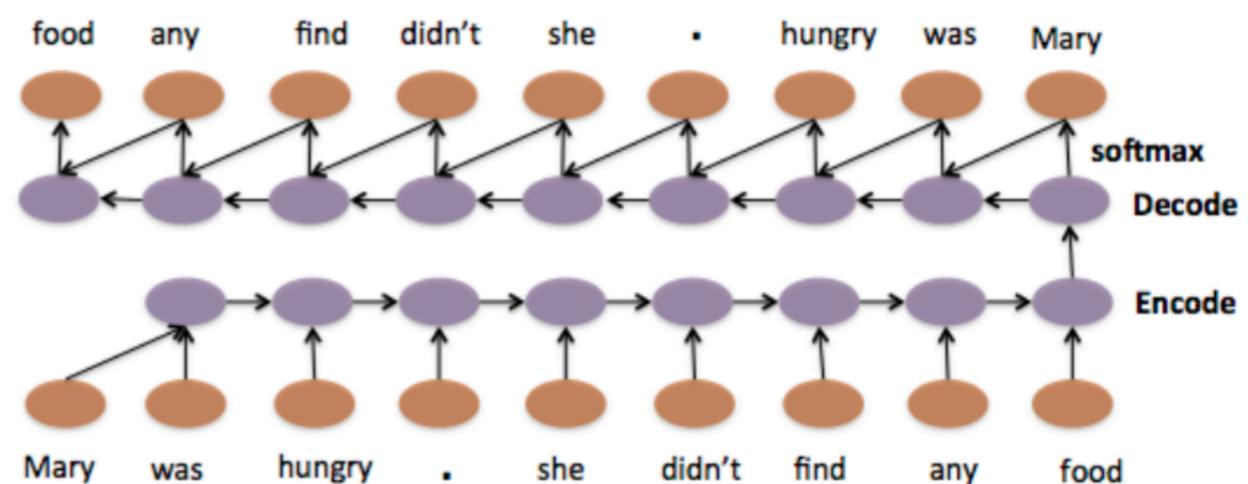
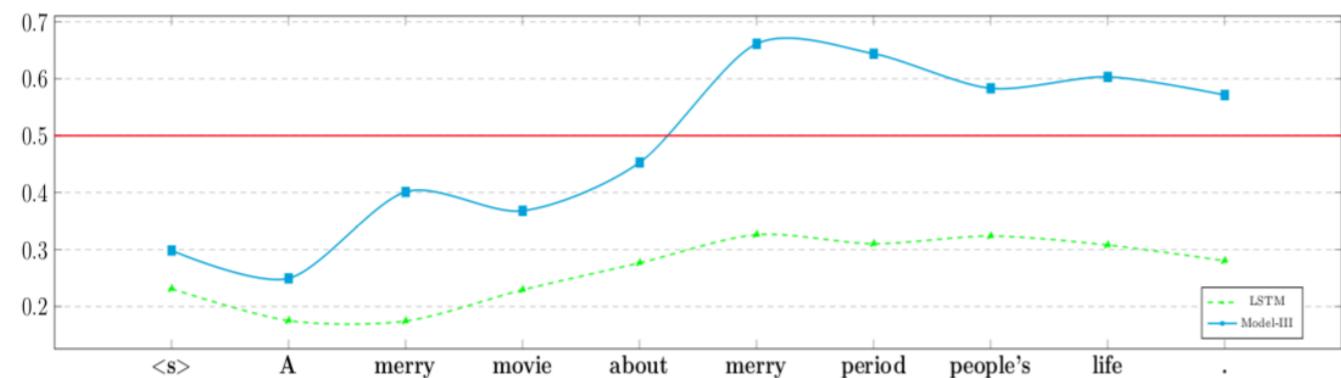
audio classification



handwriting classification

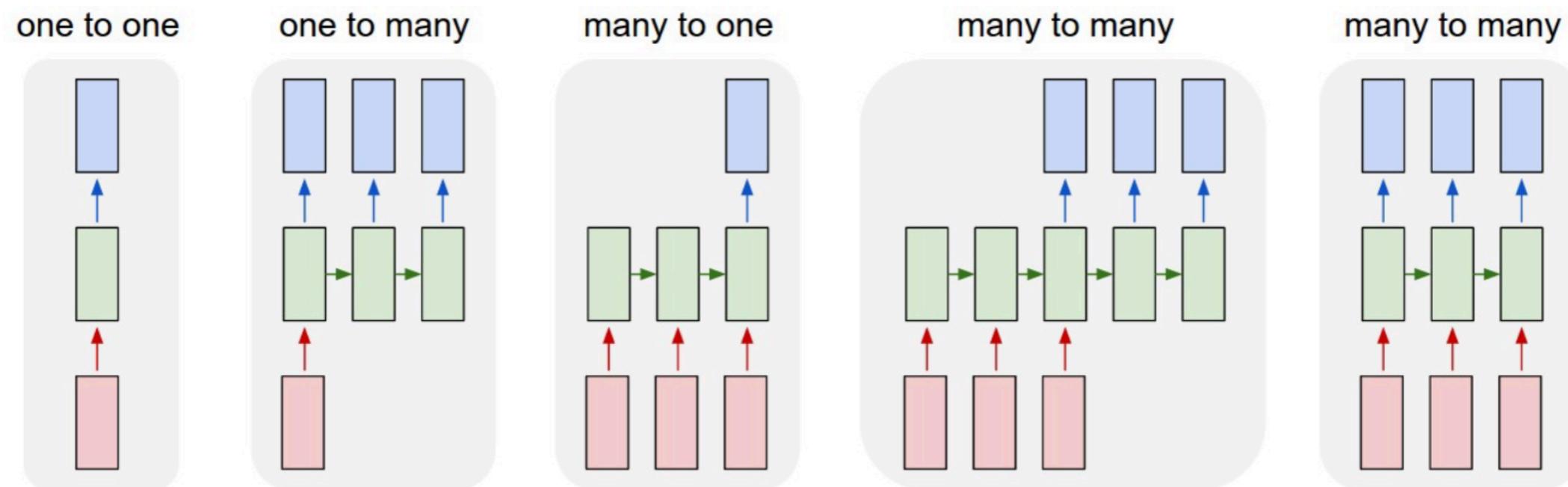


text classification

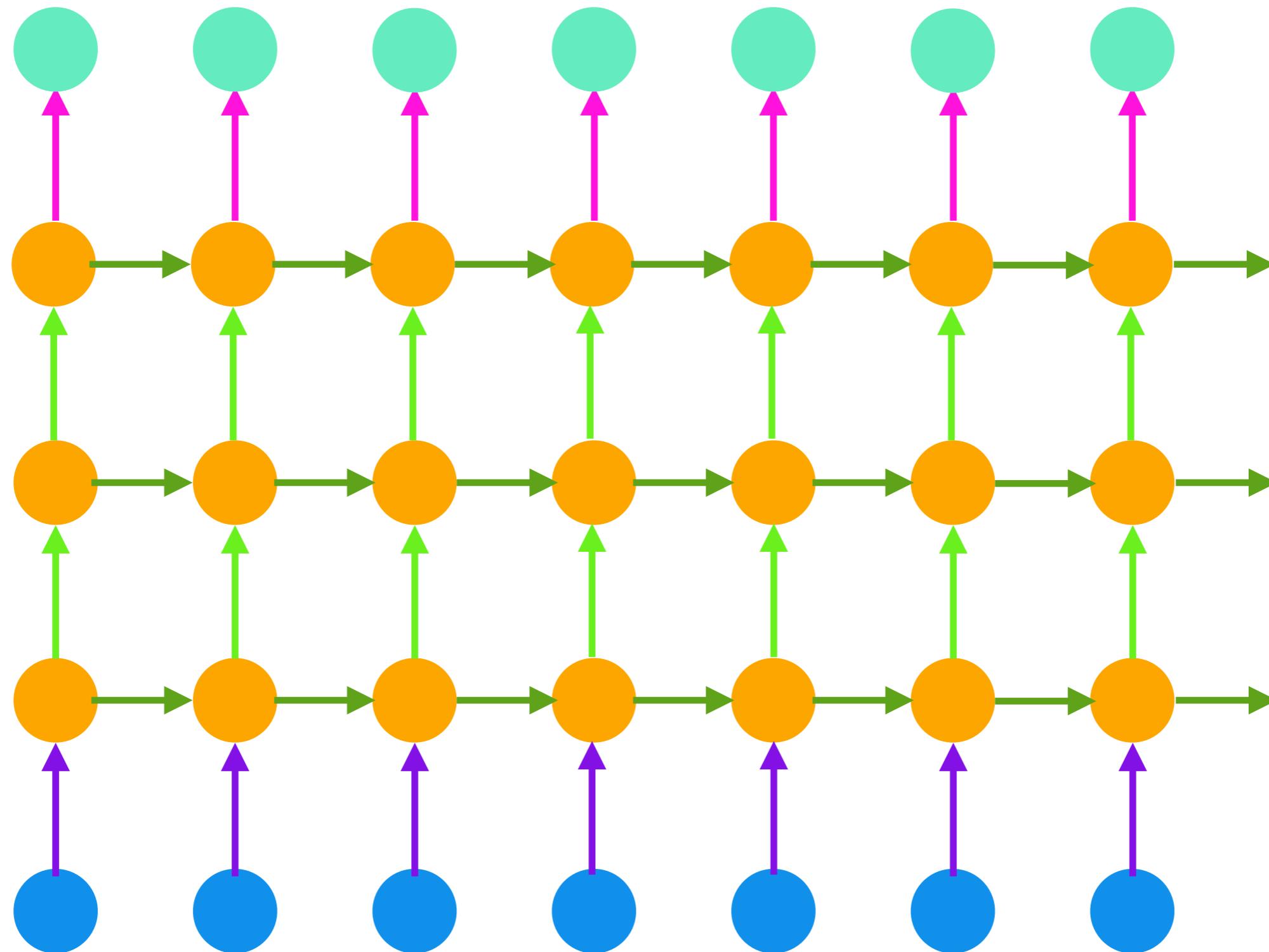


Graves, et al., 2013
Eyolfsdottir, et al., 2017
Others

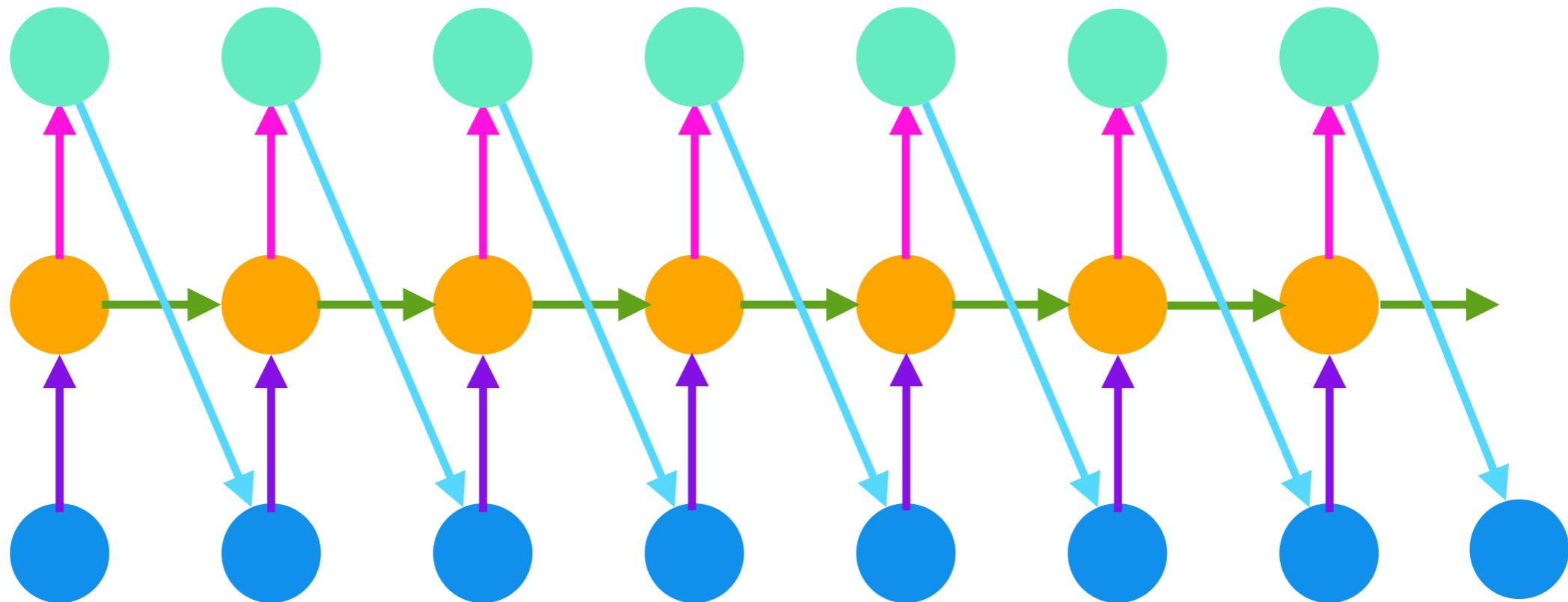
tons of options!



deep recurrent neural networks



auto-regressive generative modeling



output becomes next input

auto-regressive generative language modeling

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

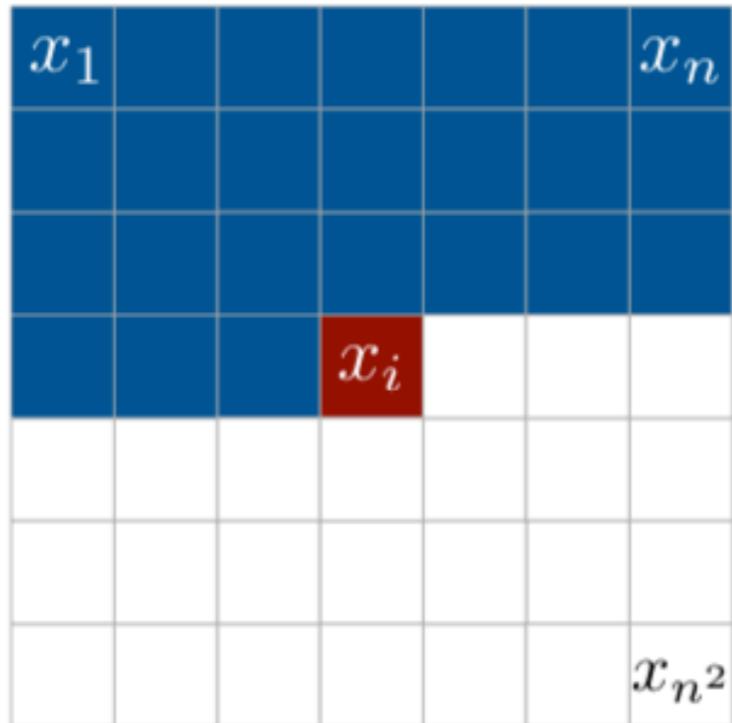
Come, sir, I will make did behold your worship.

VIOLA:

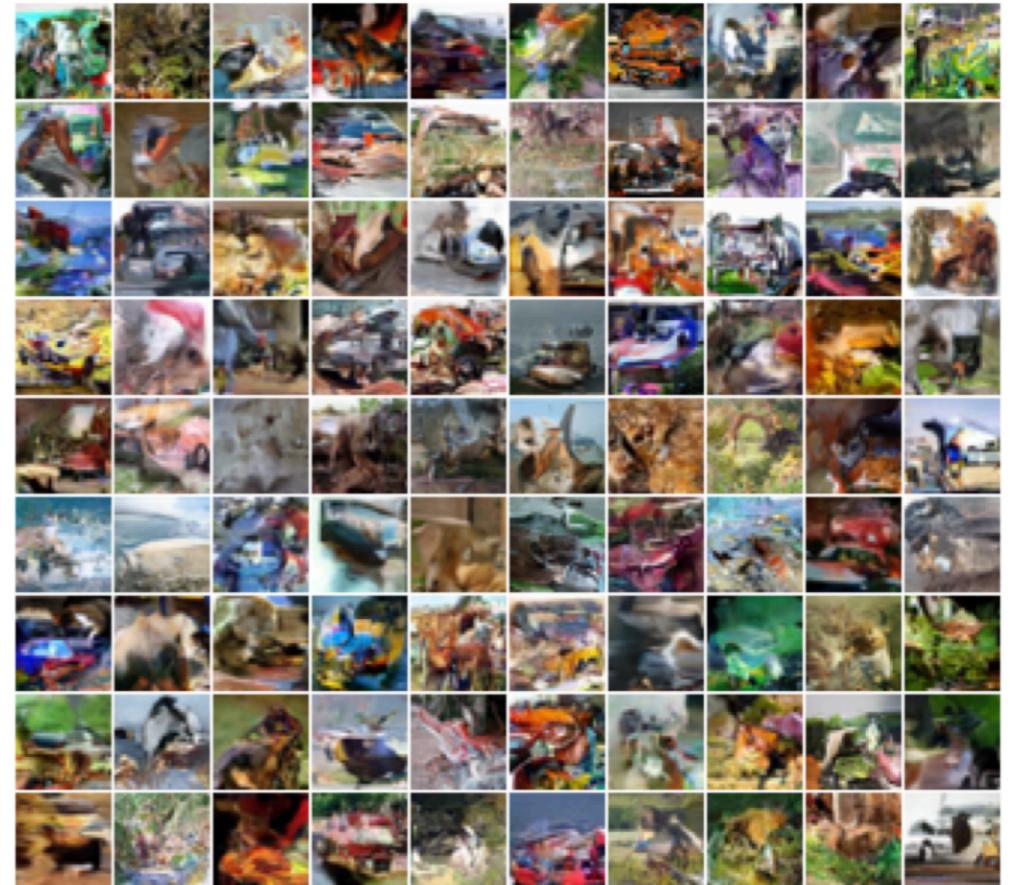
I'll drink it.

Pixel RNN uses recurrent networks to perform
auto-regressive image generation

context

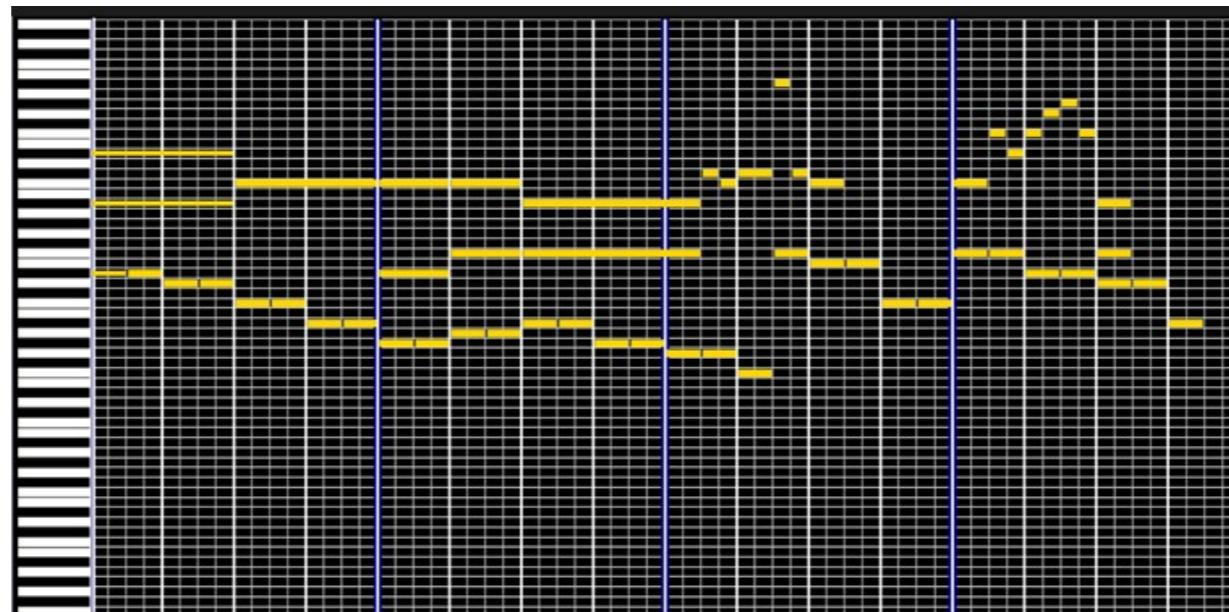


generated samples



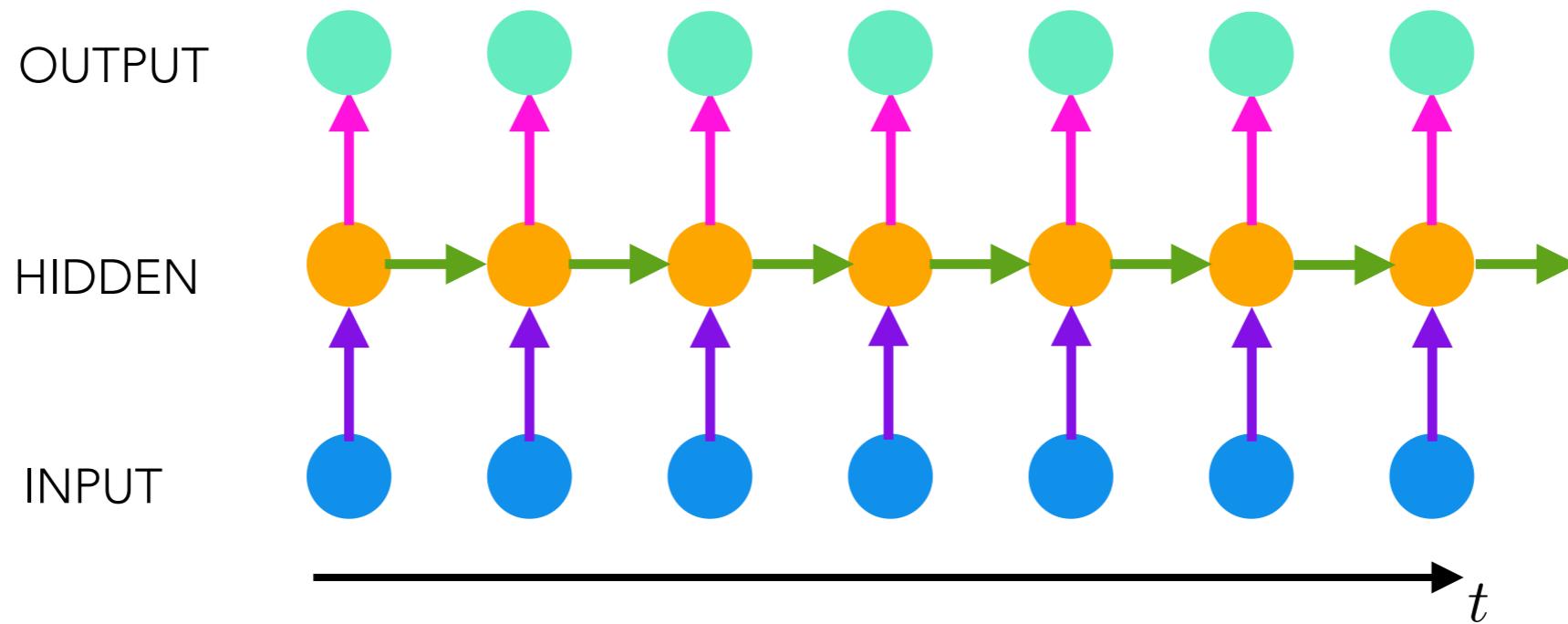
condition the generation of each pixel on a sequence of past pixels

MIDI music generation



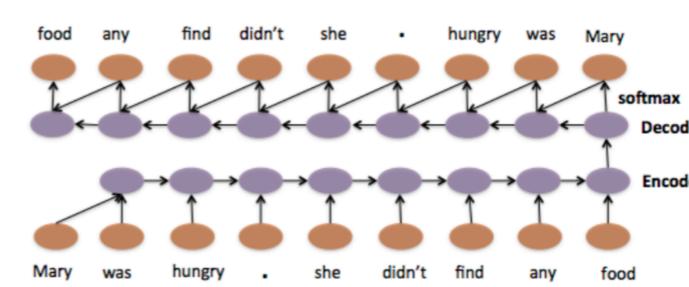
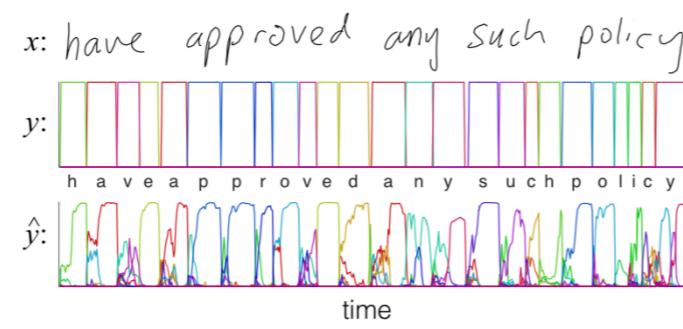
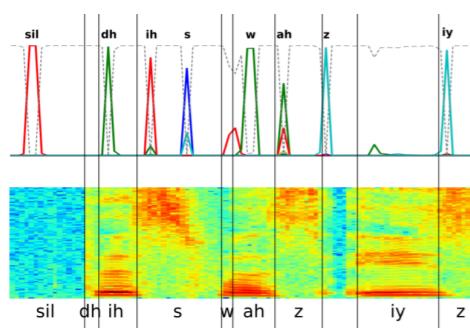
recapitulation

we can exploit sequential structure to impose inductive biases on the model



this limits the number of parameters required,
reducing flexibility in reasonable ways

can then scale these models to complex data sets to perform difficult tasks



RECAP

recapitulation

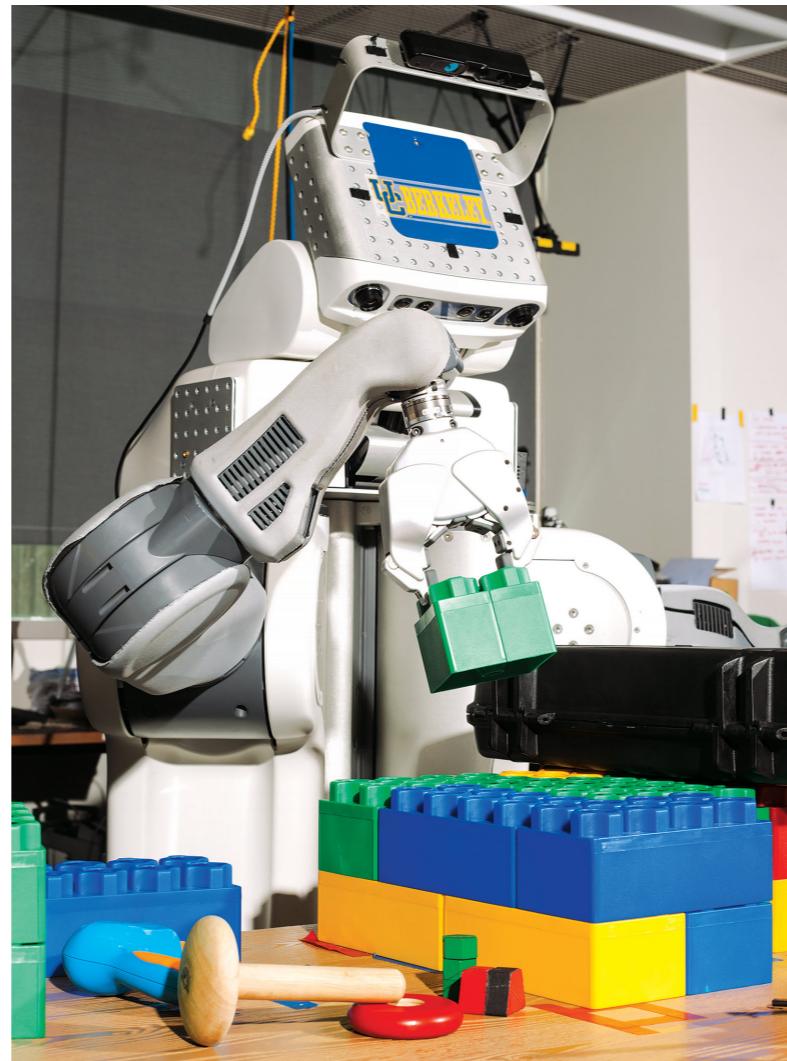
we used additional priors (inductive biases) to
scale deep networks up to handle spatial and sequential data



without these priors, we would need
more parameters and data

we live in a **spatiotemporal** world

we are constantly getting sequences of spatial sensory inputs



embodied intelligent machines need to learn from
spatial and temporal patterns

CNNs and RNNs are building blocks for machines that can use spatiotemporal data to solve tasks

