Northeastern

CS 7140:
ADVANCED MACHINE LEARNING

Langevin Monte Carlo

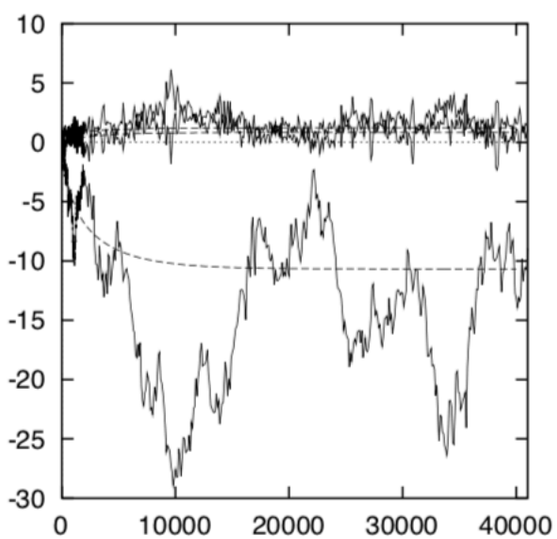
- Monte Carlo: evaluate the integral of a distribution

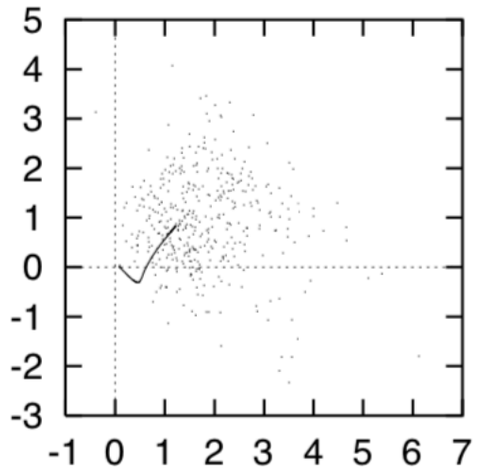
$$\mathbb{E}[f(w)] \approx \frac{1}{R}\sum_{r} f(w^{(r)})$$

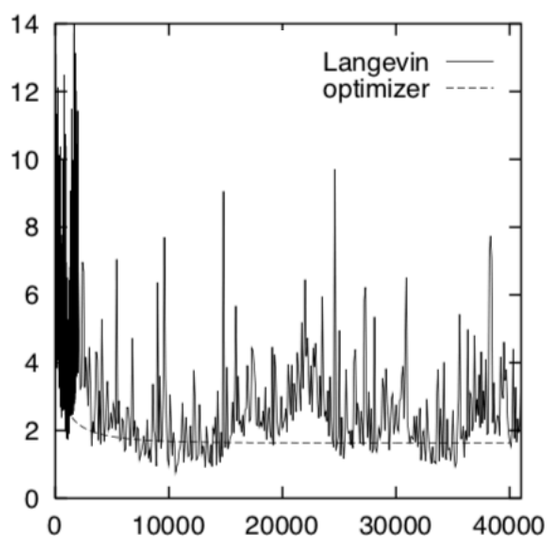
- Langevin: gradient descent with added noise $p \sim \mathcal{N}(0,1)$

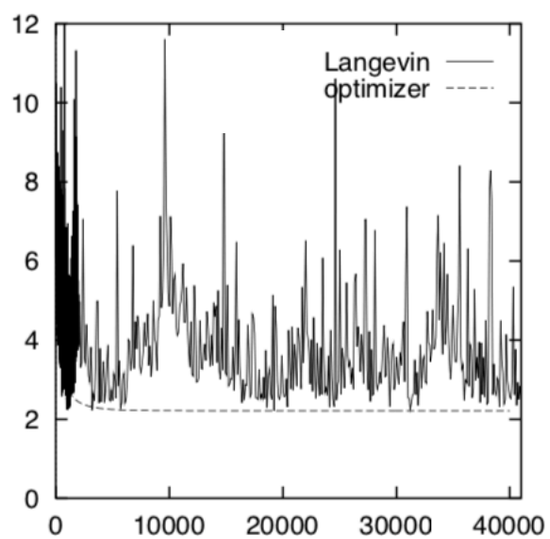
$\Delta w = \frac{1}{2}\epsilon^2 g + \epsilon p$ accept or reject depending on $J(w)$, g is the gradient

- ϵ controls the step size, too *large*, moves may be rejected; too *small*, progress around the state space is slow









Gaussian Approximation

- Taylor expansion $J(w) \approx J(w_0) + \frac{1}{2}(w - w_0)^\top A(w - w_0) + \cdots$

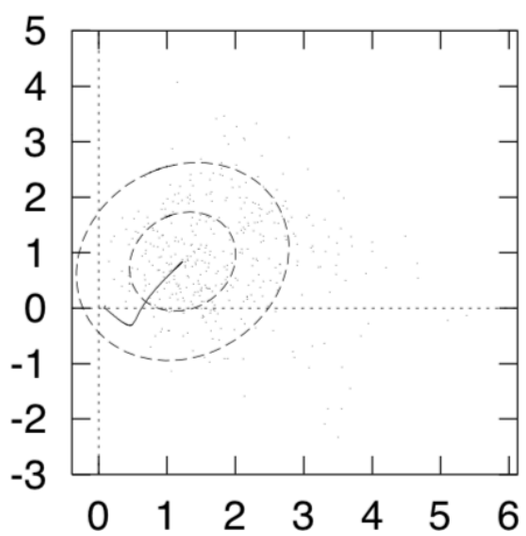
  where $A_{ij} \equiv \frac{\partial^2}{\partial w_i w_j} J(w)$ is the *Hessian* matrix

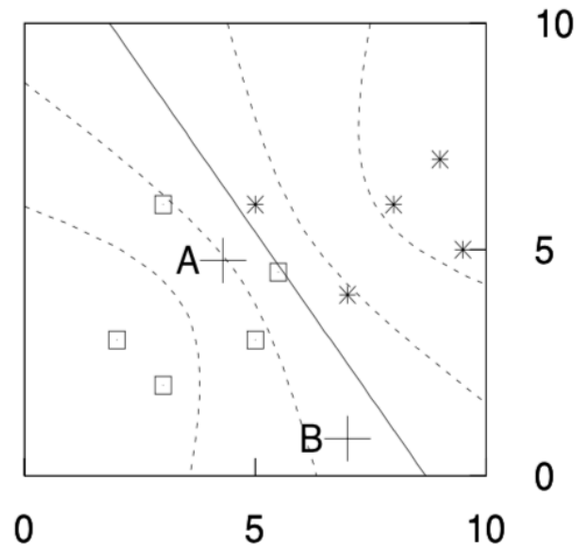
- Approximate the posterior with a Gaussian

  $$Q(w; w_0, A) = [\det A/2\pi]^{1/2} \exp\Big[-\frac{1}{2}(w - w_0)^\top A(w - w_0)\Big]$$

- Compute the integral using Gaussian posterior

  $$\Delta w = w - w_0 \qquad P(w \mid D, \alpha) \approx (1/Z_Q)\exp(-\frac{1}{2}\Delta w^\top A \Delta w)$$





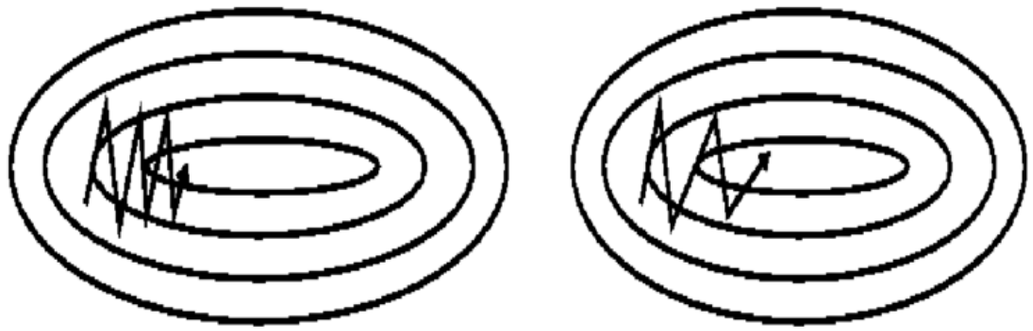
Accelerating SGD

- Momentum

physical interpretation: update velocity

$$w = w - \alpha \tilde{\nabla}_w \mathcal{L}$$

$$v_t = \alpha v_{t-1} - \epsilon \, \tilde{\nabla}_w \, L(w_t)$$
$$w_{t+1} = w_t + v_t$$



SGD *without* momentum SGD *with* momentum

- Nesterov Momentum

stronger theoretical guarantee

$$v_t = \alpha v_{t-1} - \epsilon \, \tilde{\nabla}_w \, L(w_t + \alpha v_{t-1})$$
$$w_{t+1} = w_t + v_t$$

Adaptive Learning Rate

Cost is sensitive to learning rate only in some directions in parameter space

Use a separate learning rate for each parameter

- AdaGrad

gradient

approximate Hessian

update

$$g = \tilde{\nabla}_w L(w_t)$$

$$r_t = r_{t-1} + g \odot g$$

$$w_{t+1} = w_t - \frac{\epsilon}{\delta + \sqrt{r}} \odot g$$

- RMSProp

gradient

approximate Hessian

update

$$g = \tilde{\nabla}_w L(w_t)$$

$$r_t = \rho r_{t-1} + (1 - \rho) g \odot g$$

$$w_{t+1} = w_t - \frac{\epsilon}{\sqrt{\delta + r}} \odot g$$

STRUCTURED PREDICTION

Structured Prediction

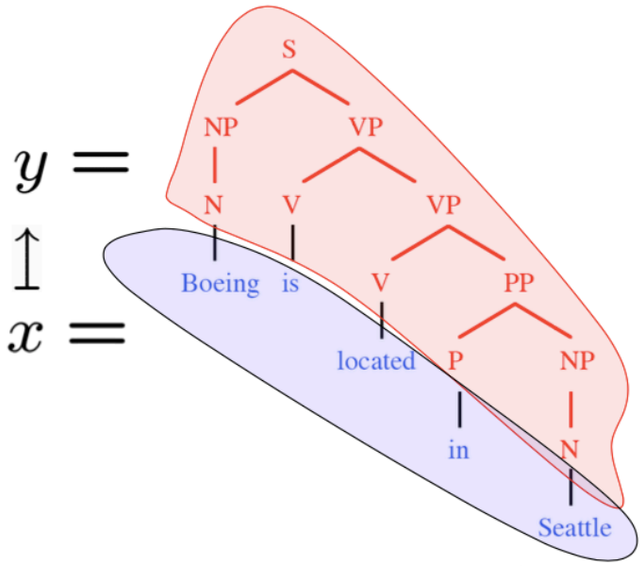
- A prediction function $f : \mathcal{X} \longrightarrow \mathcal{Y}$, where the output $\mathcal{Y}$ domain is structured

  $y^\star = f(x) = \text{argmax}_y g(x, y)$ MAP inference

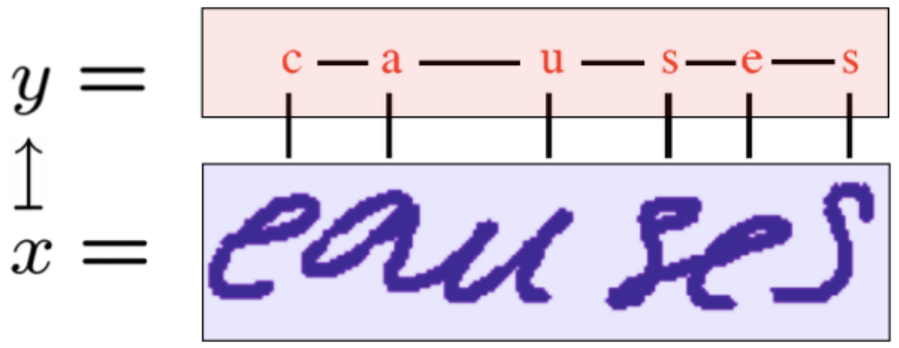
- $g(x, y) = p(y \mid x)$ when the model is probabilistic

- Many MAP inference problems are NP Hard: combinatorial number of possibilities

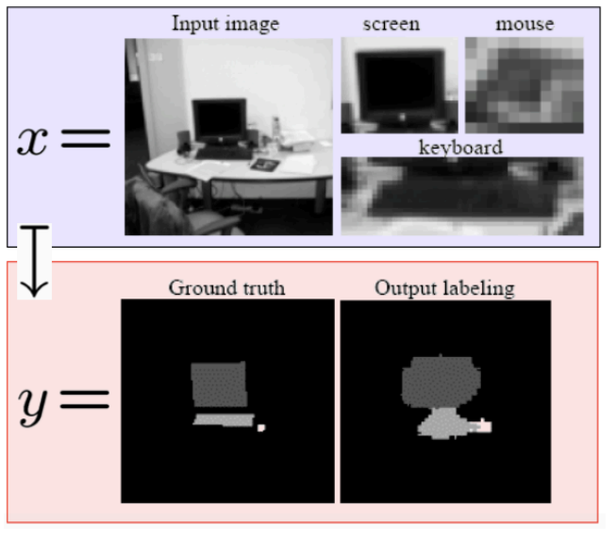
Examples Applications



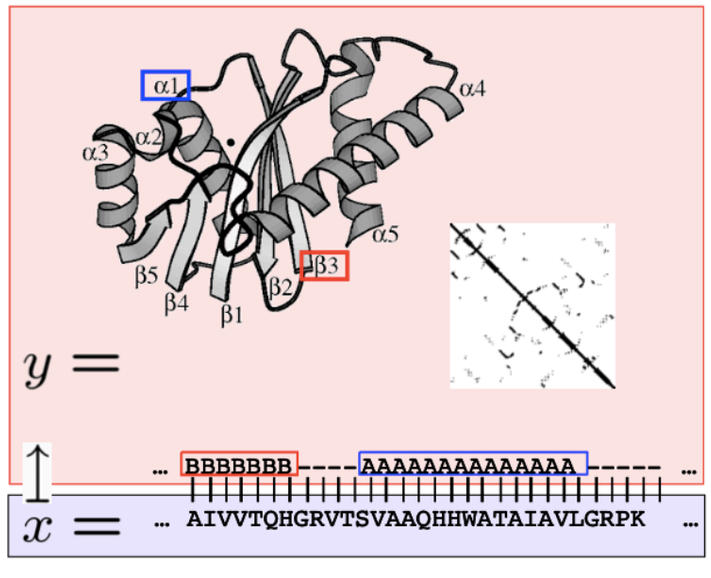
natural language parsing



hand-written text recognition



visual scene understanding



protein structure prediction

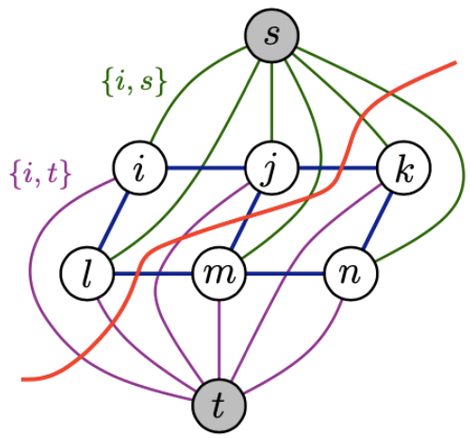
Graph Cuts

- Image foreground segmentation:
  - foreground 1, background 0, $C(x_i, x_j)$ intensity contrast

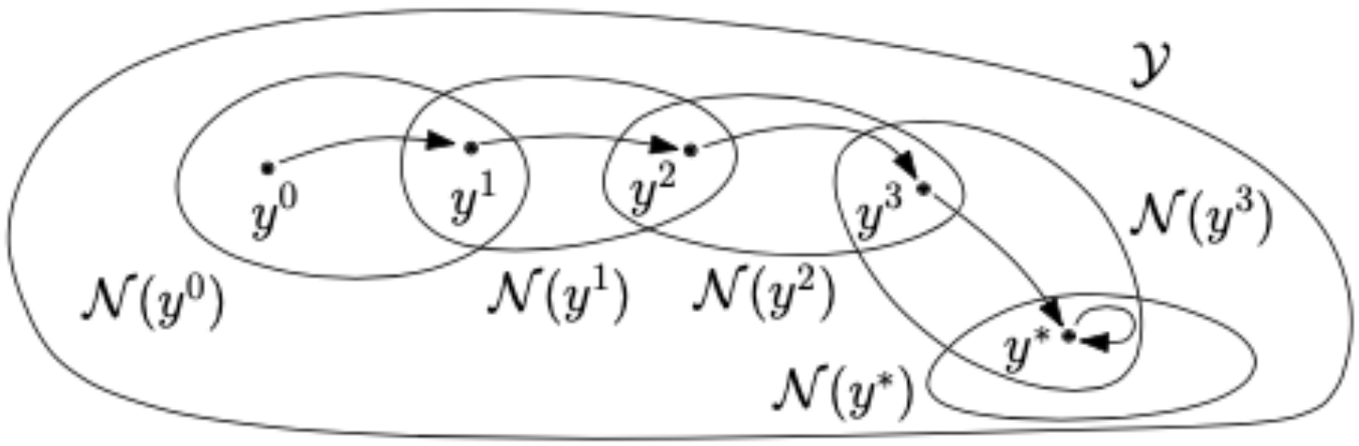
$$g(x, y; w) = \sum_{i \in V} \log p(y_i \mid x_i) + w \sum_{(i,j) \in E} C(x_i, x_j)[y_i \neq y_j]$$

  - find a global maximizer with graph cuts method

- Giving up Optimality
  - computational hard
  - model uncertainty



Local Search



- Improve the solution by iteratively solving $\text{argmax}_{y \in N_s(y^t)} g(x, y)$ within the *neighborhood* of current solution

- Terminate when no further improvement can be made

- Optimality cannot be guaranteed. Approximation algorithms offer worst-case a priori guarantees.

Branch and Bound

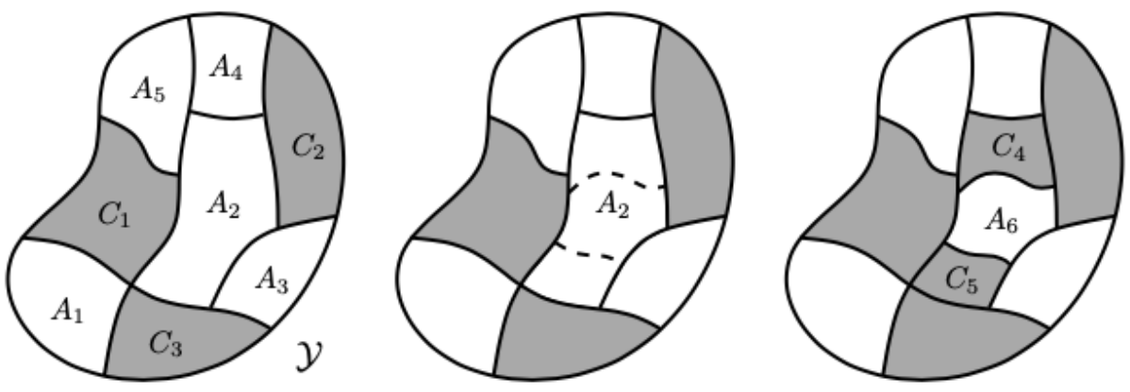
- Worst Case: Implicit enumeration of $\mathcal{Y}$ and is guaranteed to find the optimal solution

- Divide-and-conquer algorithm

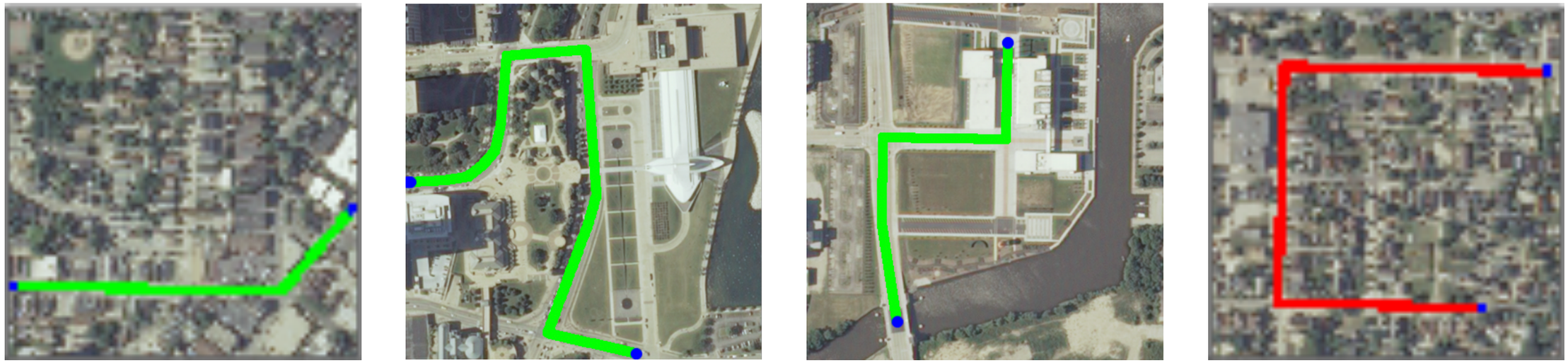
  - Partition y into *active* and *closed* nodes.

    - Closed nodes do not contain a better solution

    - Move nodes from active to closed



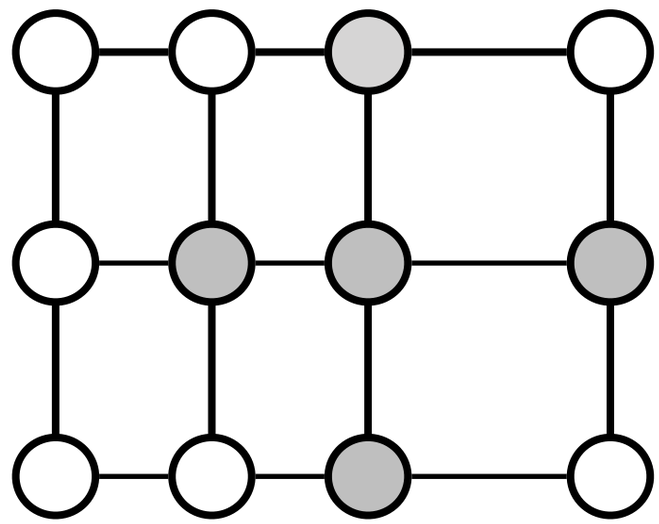
Learning to Plan



- Learning the cost function with the intended property, e.g. preferring roads and avoiding vegetation
- Update the cost function as search continues
- Use A* for prediction and loss-augmented prediction

Gradient-Based Optimization

- Lagrangian Relaxation
  - Objective $J(x)$, equality constraint $x = c$
  - Lagrangian: $J(x) + \lambda(x - c)$

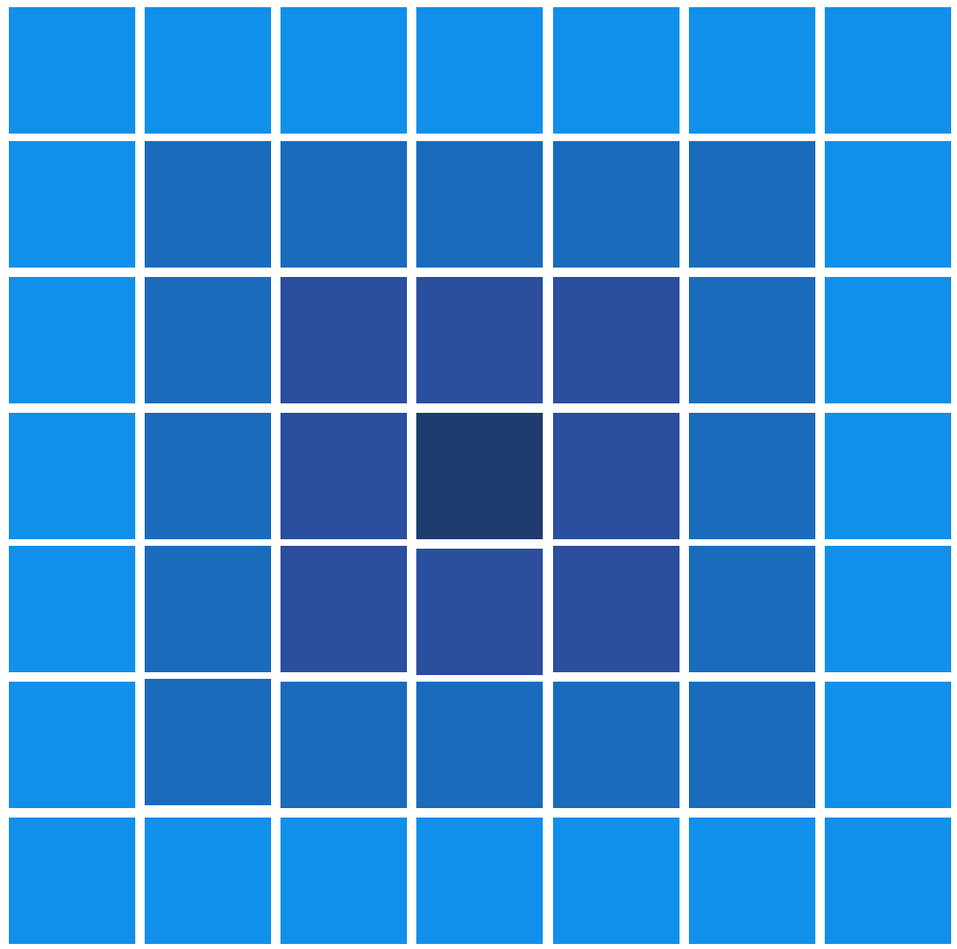


- MAP-MRF:
  - $$E(y;\theta,\mu) = \sum_{m,n} \theta_{mn}\mu(y_m, y_n) + \sum_n \theta_n \mu(y_n)$$
  - constraints: $$\sum_n \mu_n(y_n) = 1, \quad \sum_n \mu_{m,n}(y_m, y_n) = \mu_m(y_m)$$

DEEP STRUCTURED PREDICTION

Spatial Structure

- to reduce the amount of learning, we can exploit the ***spatial structure*** of image data



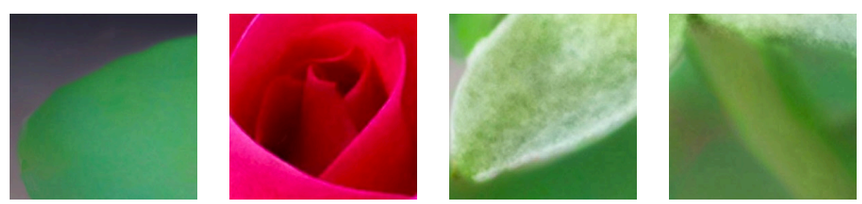
Locality

nearby areas tend to contain stronger patterns

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

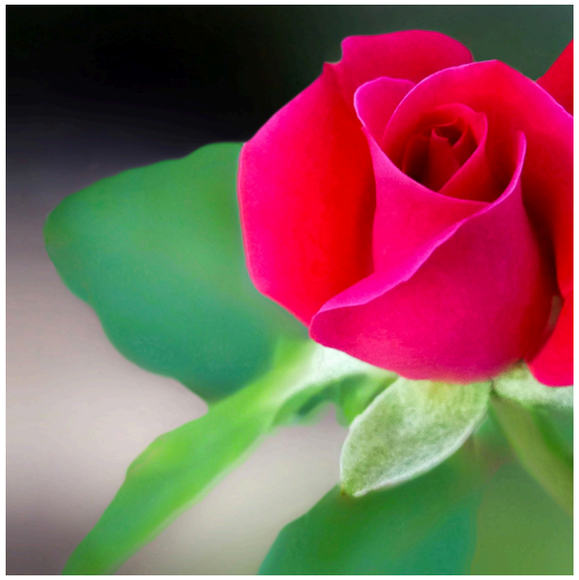
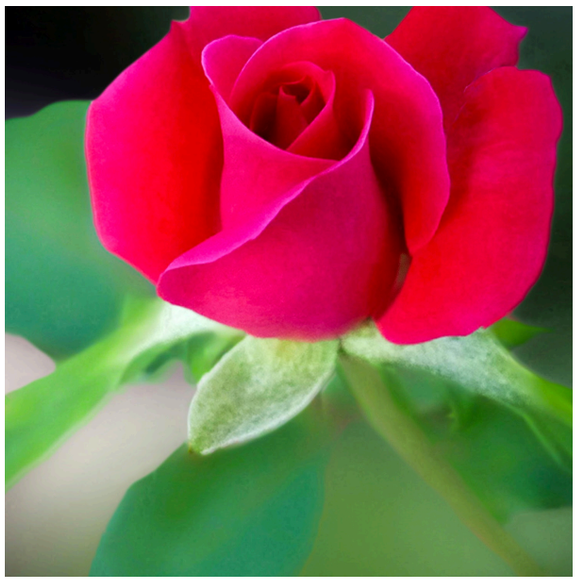
nearby ***patches*** tend to share characteristics
and are combined in particular ways

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



Translation Invariance

relative (rather than absolute) positions are relevant

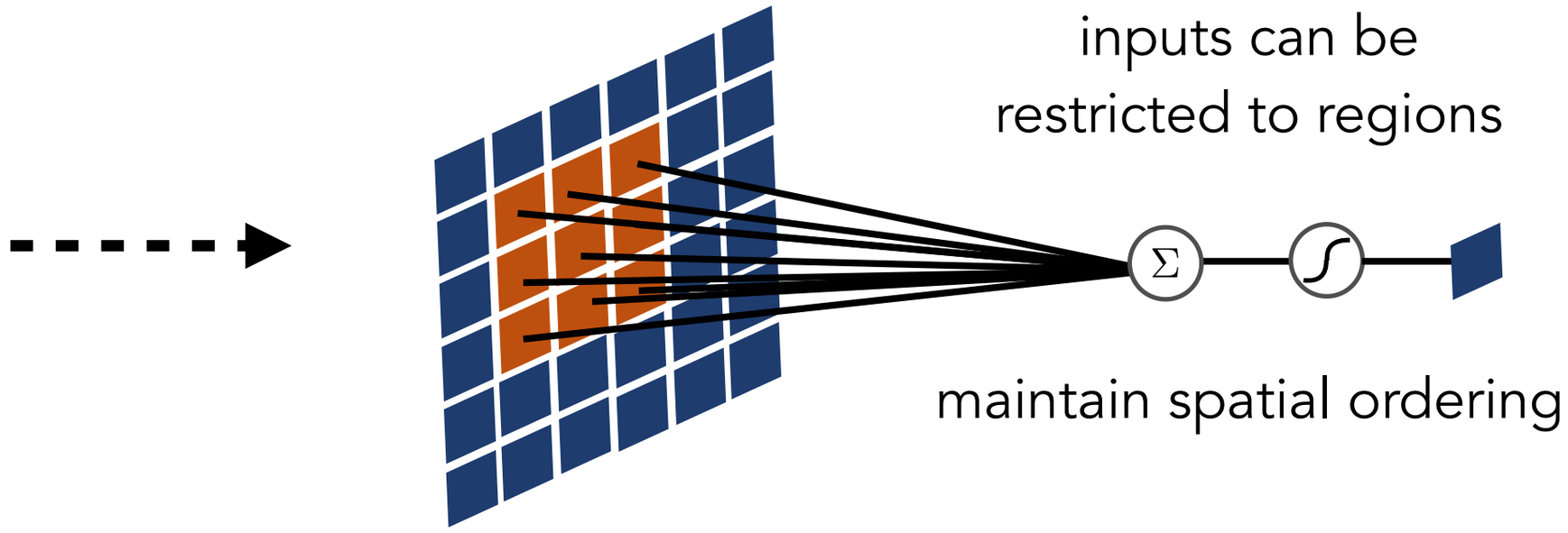





Rose is independent of absolute location of its pixels

Inductive Biases

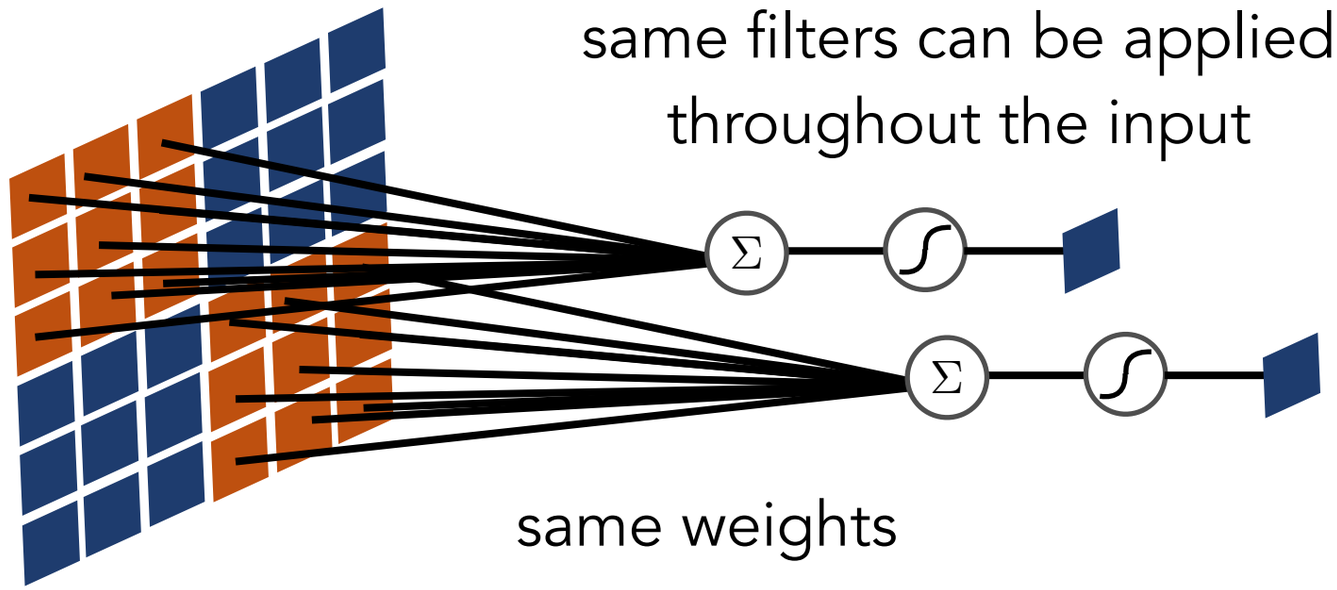
locality

nearby areas tend to contain stronger patterns



translation invariance

relative positions are relevant

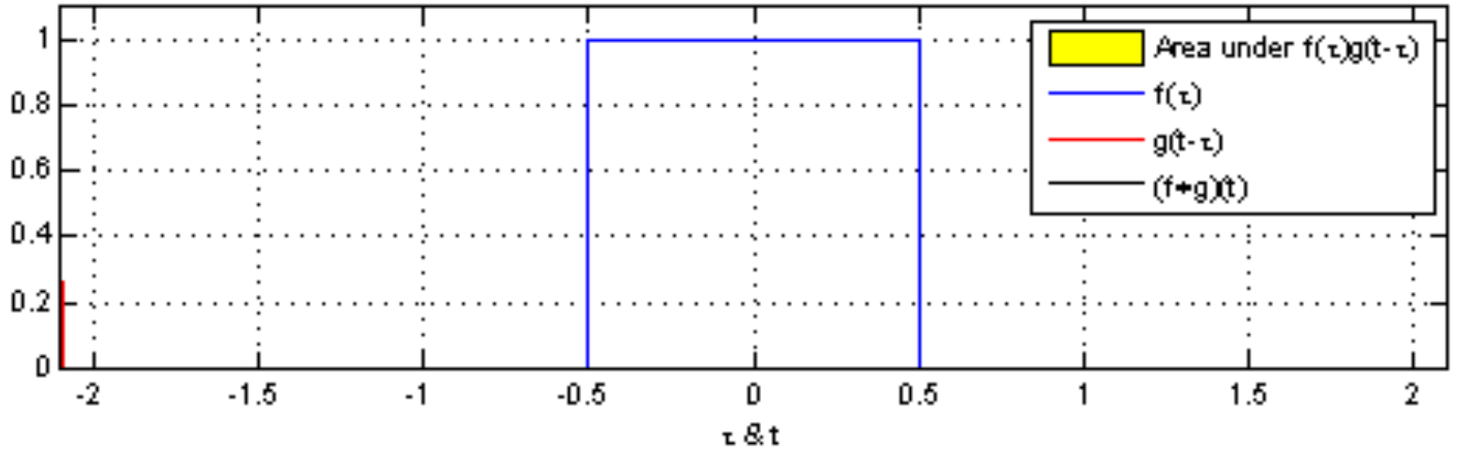


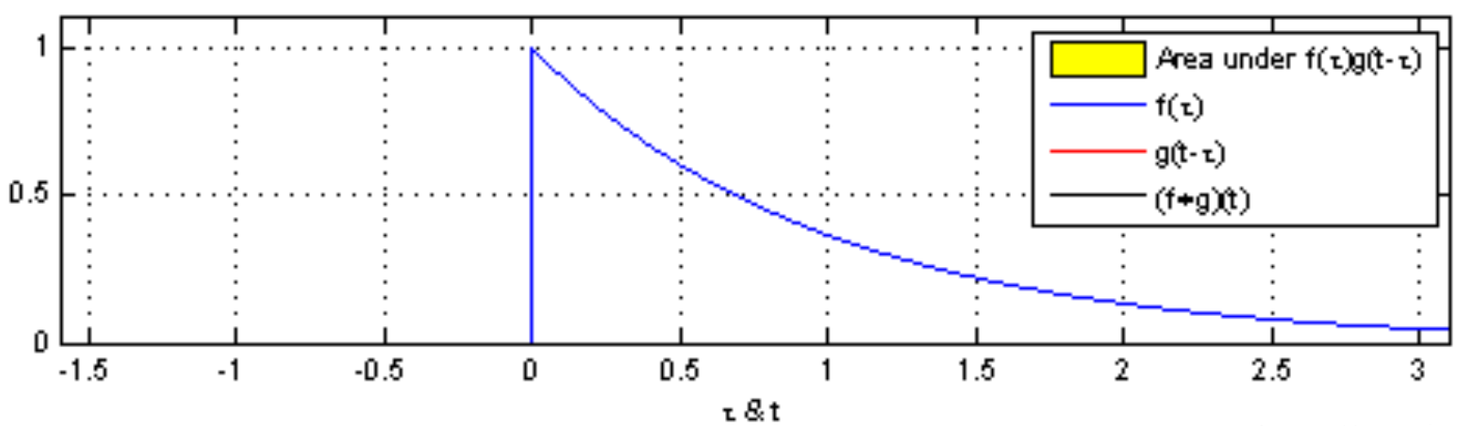
Convolution

- A integral: the amount of overlap of one function as it is shifted over another function

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = (g * f)(t)$$

filter

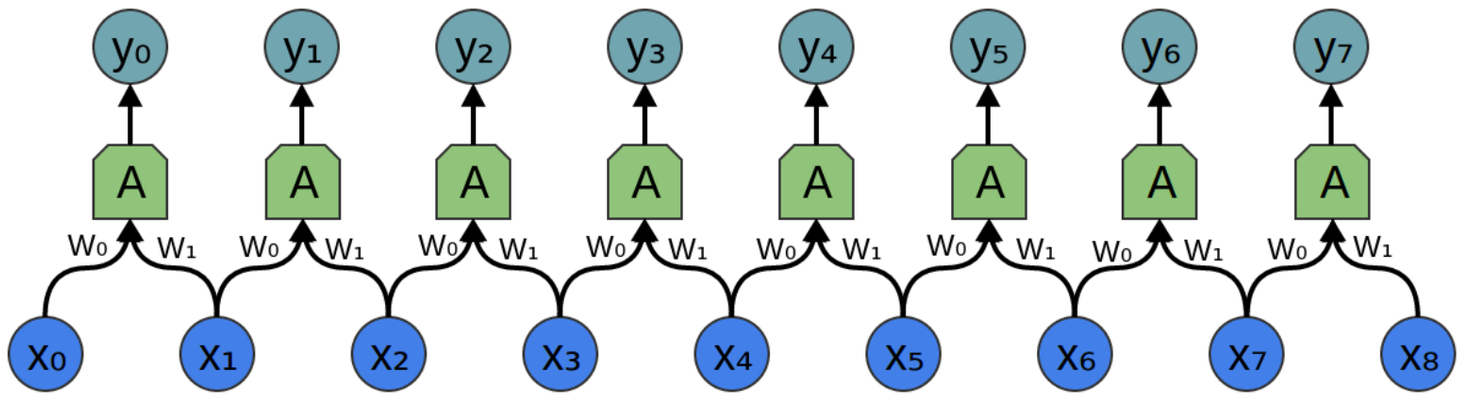




https://en.wikipedia.org/wiki/Convolution

Convolution

- 1D discrete $$(f * g)(t) = \sum_i f(i)g(t-i)$$
- 2D discrete $$(f * g)(x,y) = \sum_i \sum_j f(i,j)g(x-i,y-j)$$
- 2D cross correlation $$(f * g)(x,y) = \sum_i \sum_j f(i,j)g(x+i,y+j)$$



allow weights sharing, reduce parameter size

Convolutional Filter

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

$$(f * g)(x, y) = \sum_i \sum_j f(i, j) g(x - i, y - j)$$

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

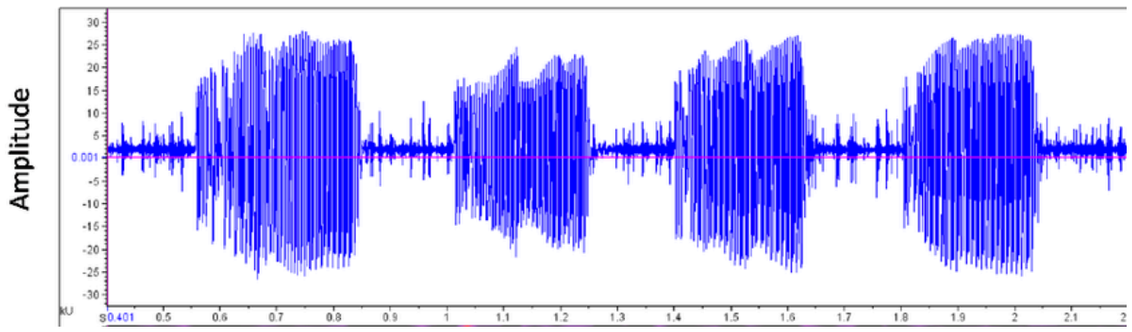
measures degree of filter feature at input location

→ *feature map*

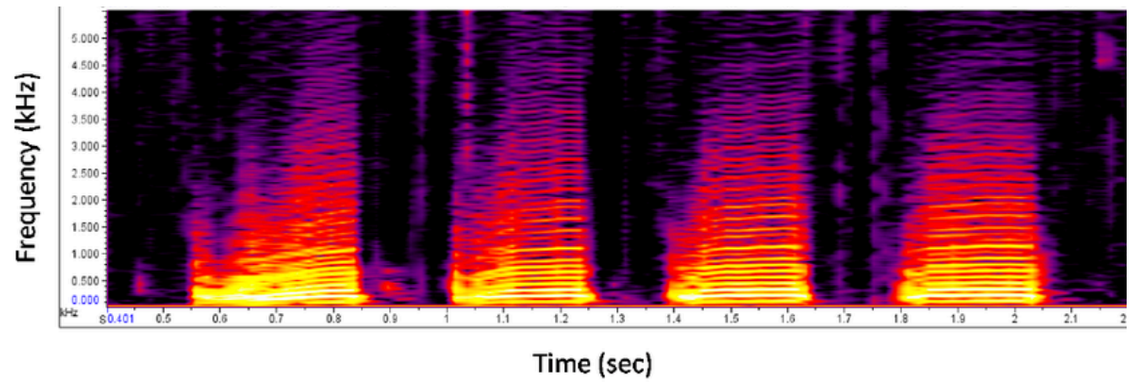
http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html

Convolution Applications

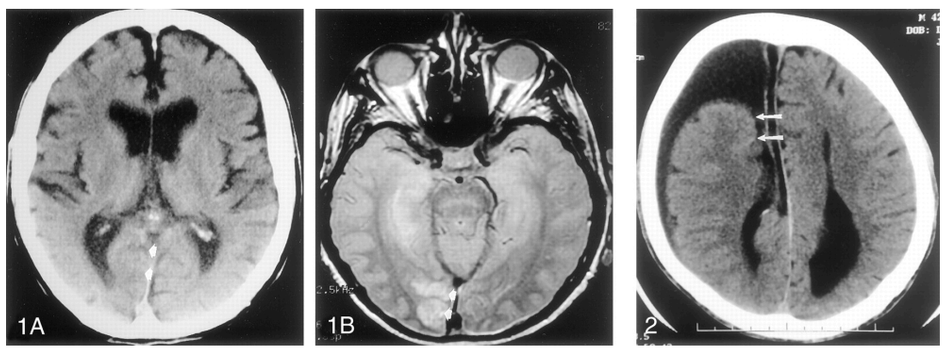
- 1 D Conv



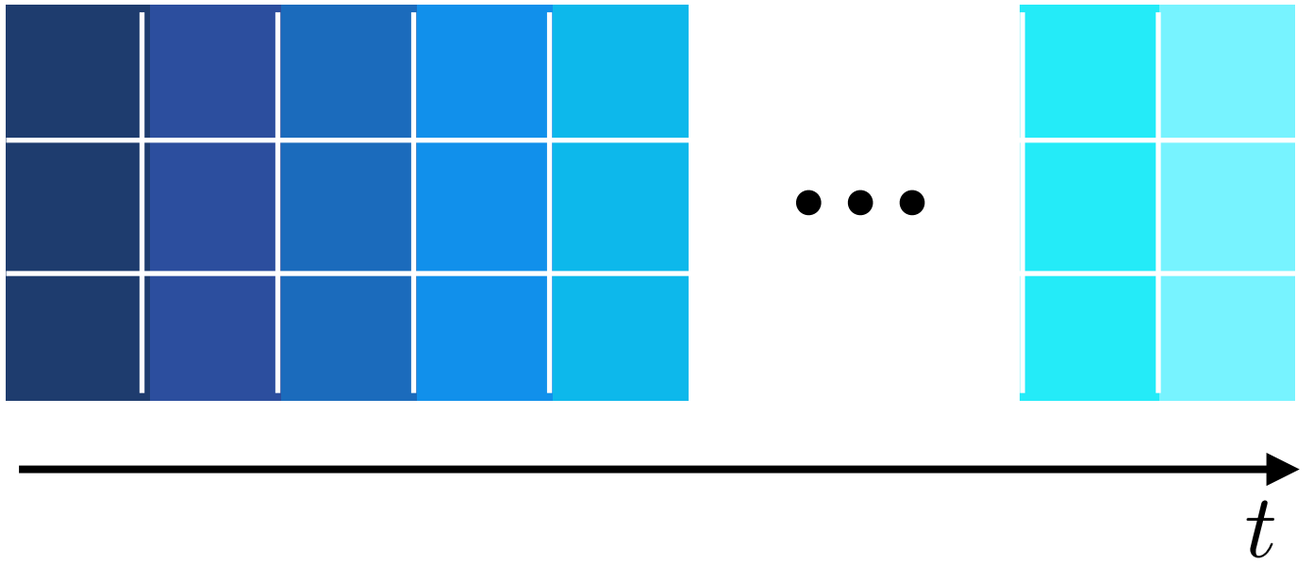
- 2 D Conv



- 3 D Conv



Temporal Structure



sequence data *also* tends to obey

locality: nearby regions tend to form stronger patterns

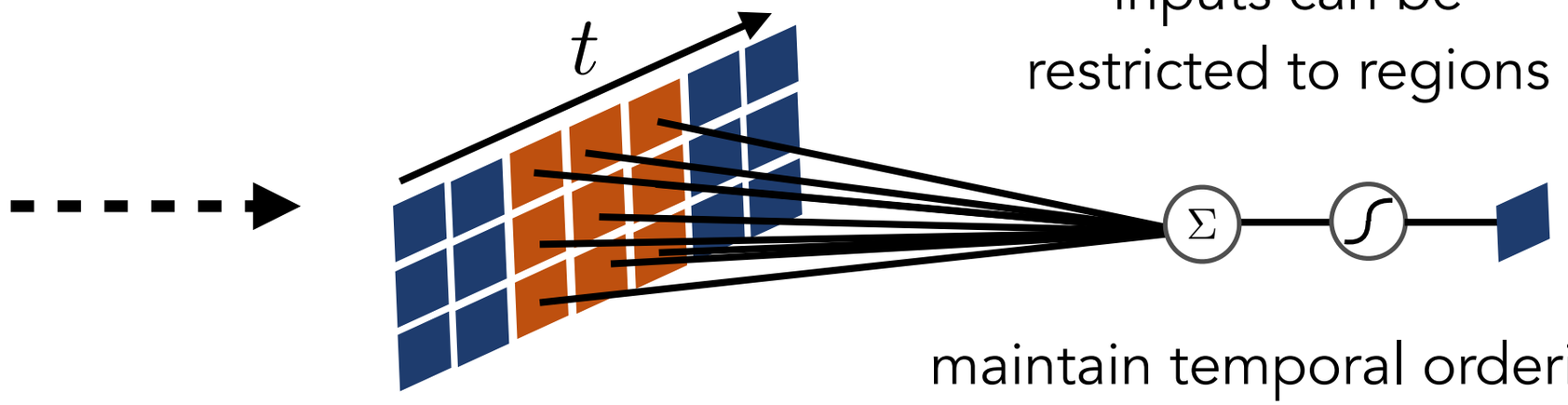
translation invariance: patterns are relative rather than absolute

but has a *single* axis on which extended patterns occur

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

locality

nearby areas tend to contain stronger patterns

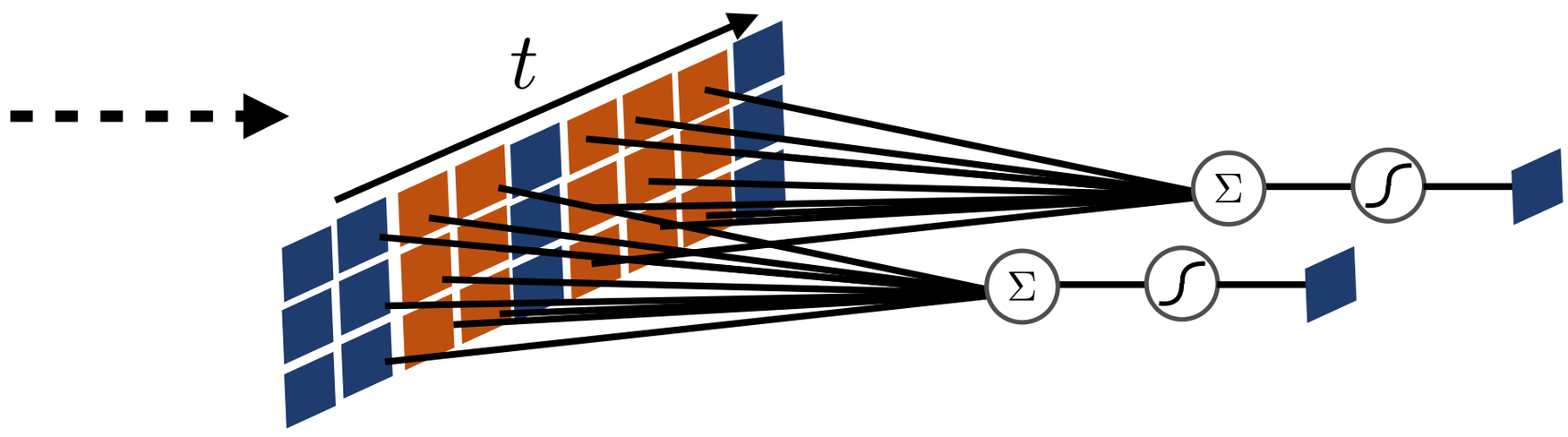


inputs can be restricted to regions

maintain temporal ordering

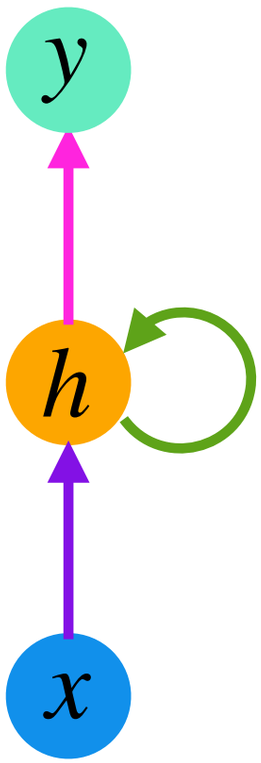
translation invariance

relative positions are relevant



same weights can be applied throughout the input

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



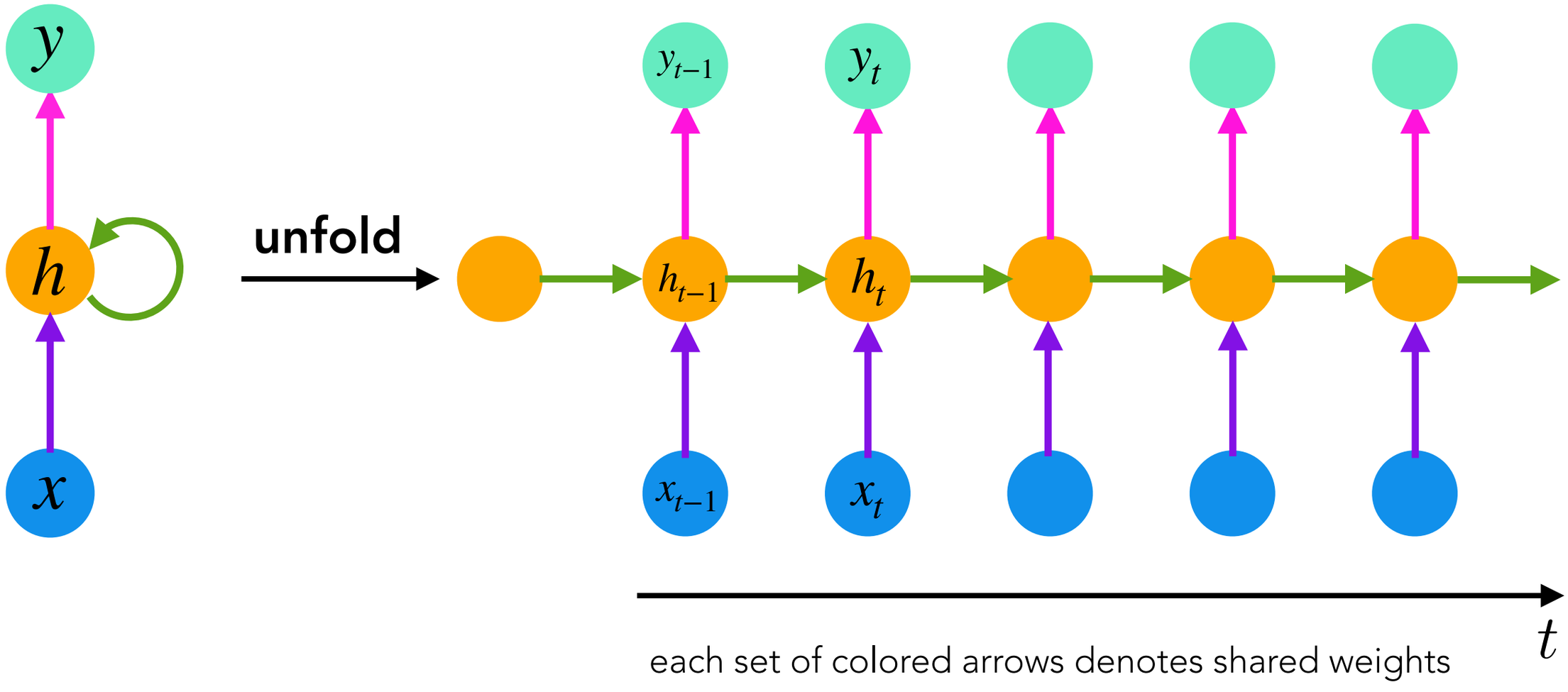
Hidden State

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

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

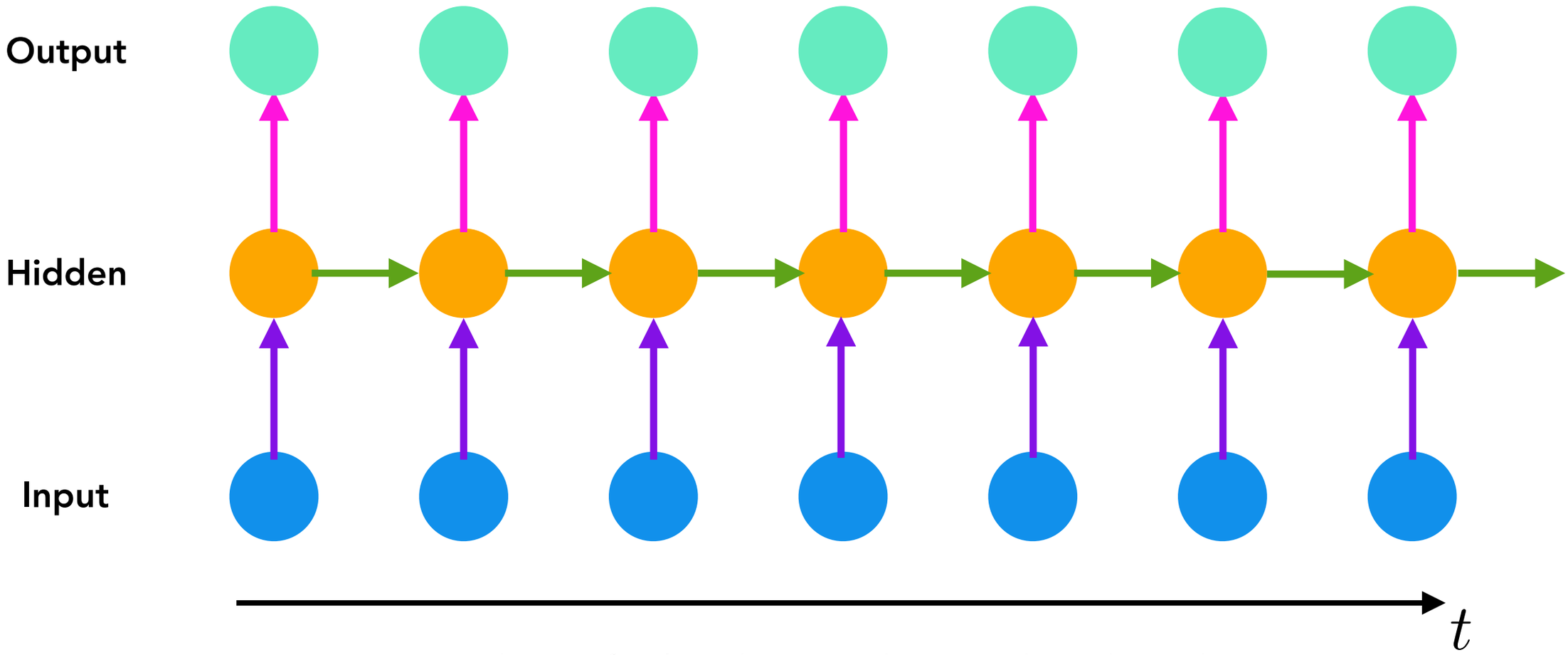
unfold a recurrent neural network



each set of colored arrows denotes shared weights

mirror the sequential structure of the data,

we can process the data sequentially



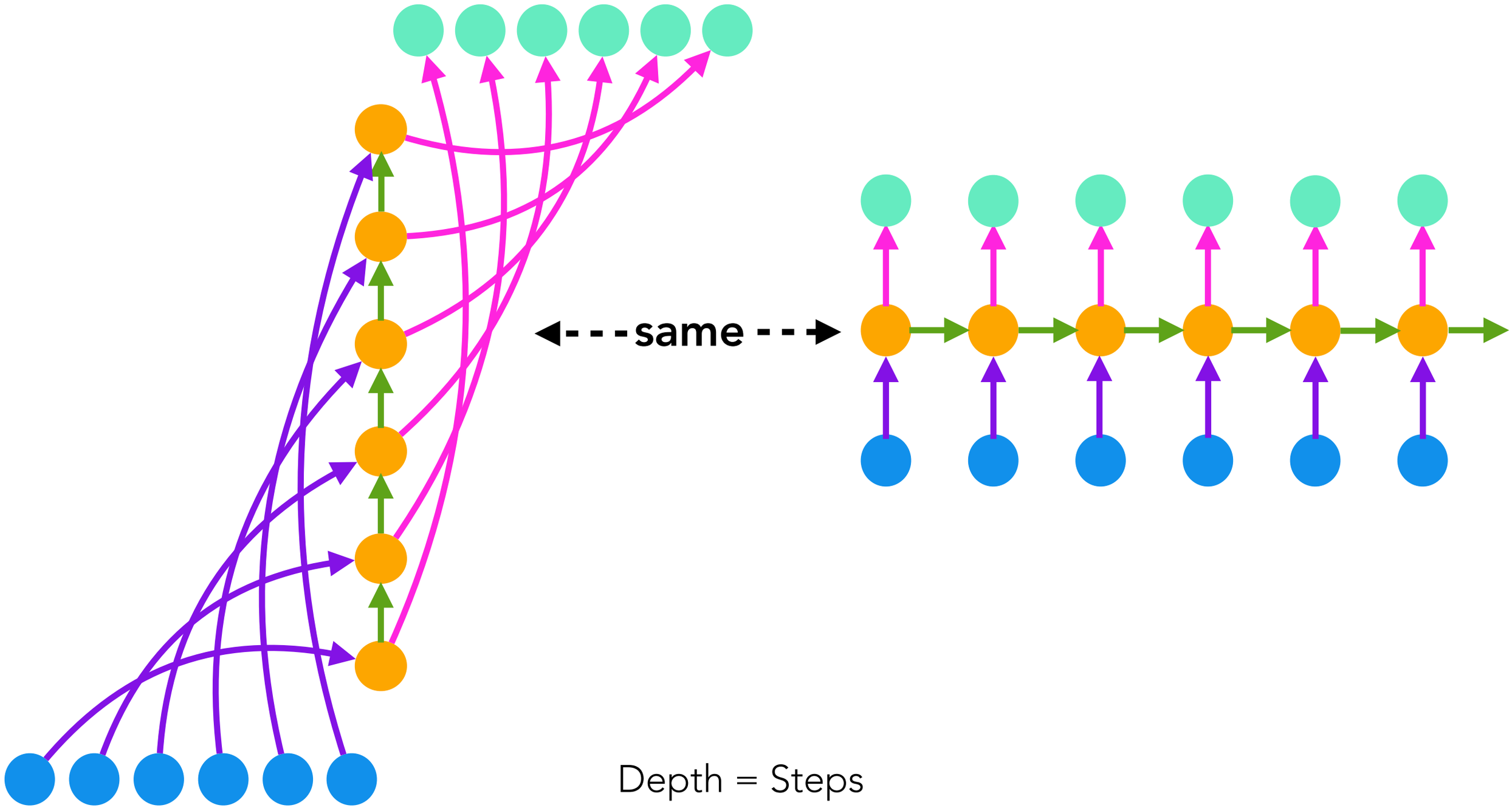
each set of colored arrows denotes shared weights

maintain an *internal representation* during processing

→ potentially *infinite* effective input window

→ fixed number of parameters

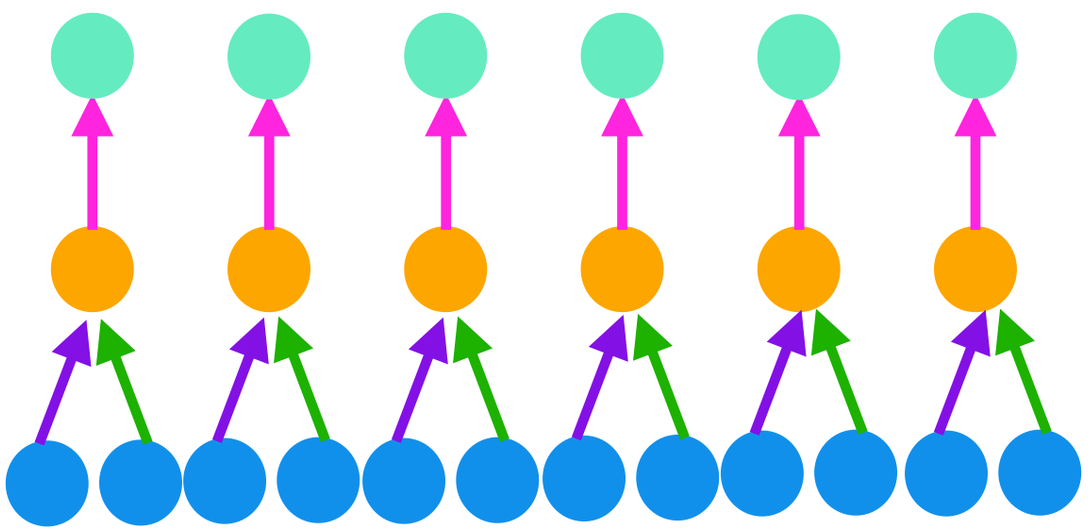
basic recurrent networks are also a ***special case***
of standard neural networks with *skip connections* and *shared weights*



Weight Sharing

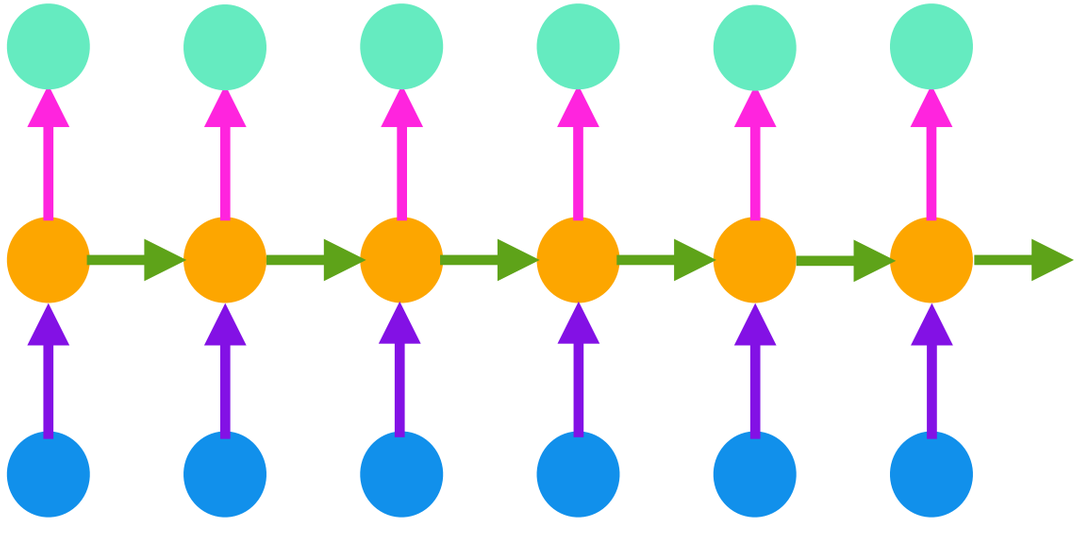
CNN

sharing weights across filters



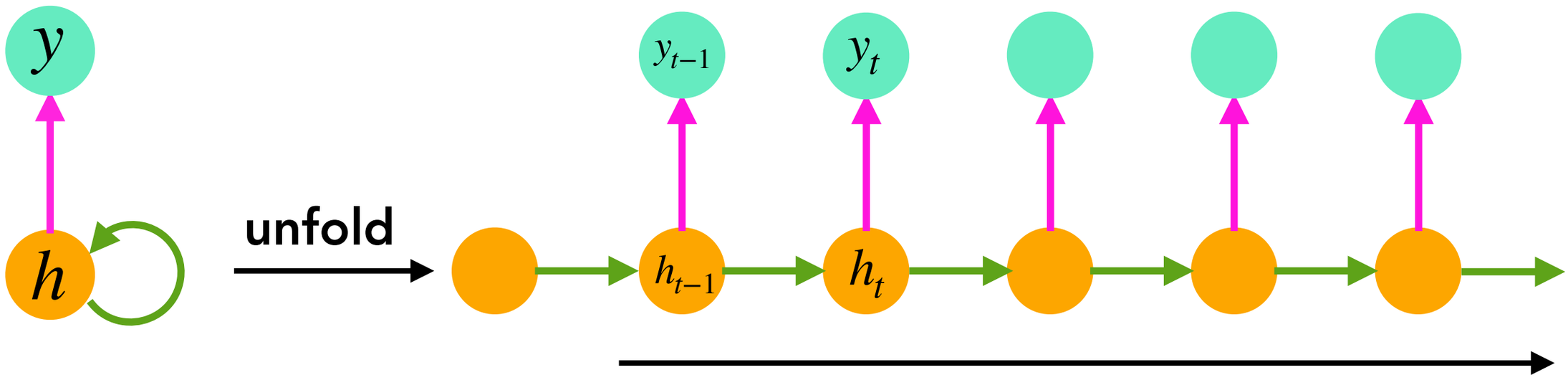
RNN

sharing weights across transitions



Probabilistic Interpretation

- Consider a simple RNN without inputs

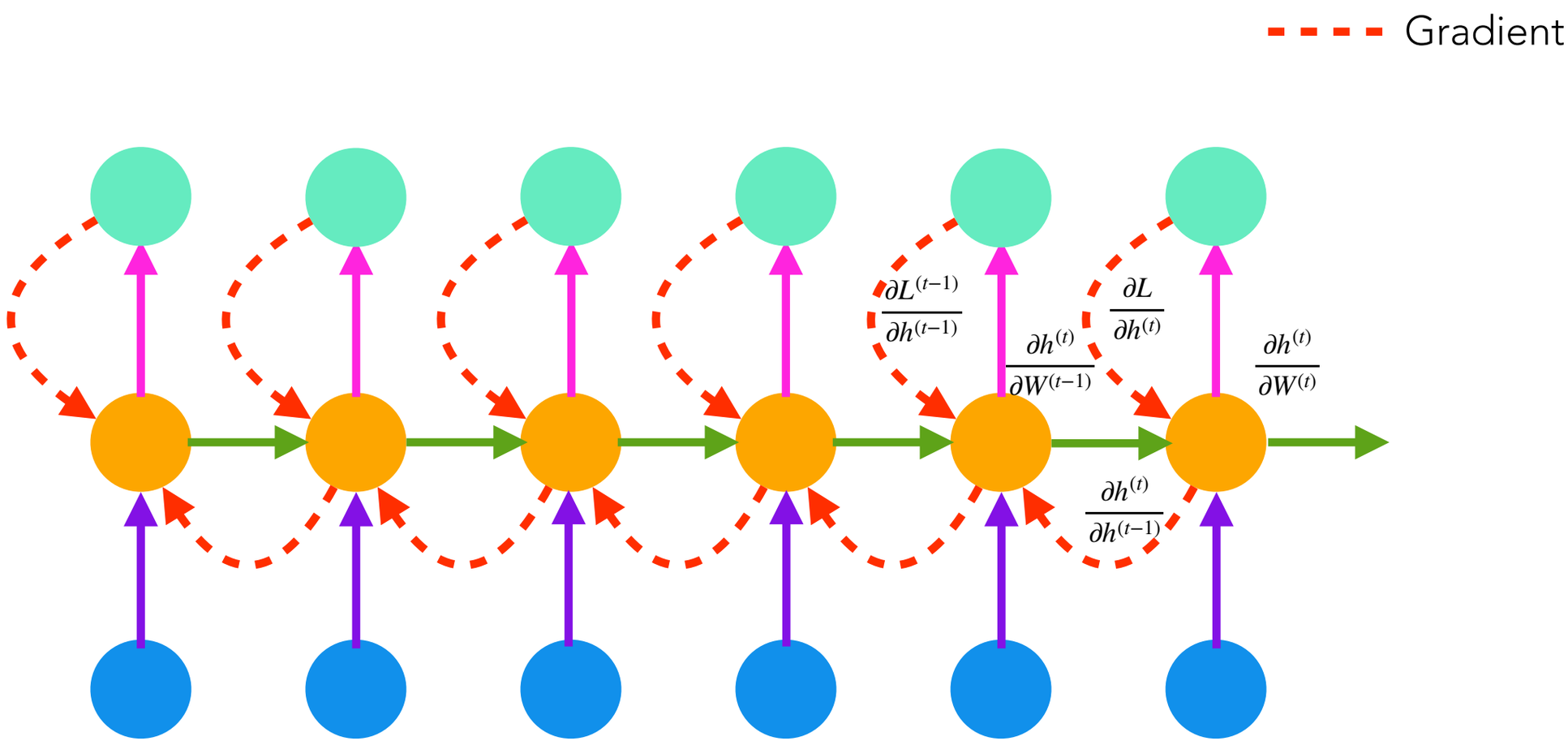


$$p(y_1, y_2, \cdots, y_T) = \prod_{t=1}^{T} p(y_t \mid y_{t-1}, \cdots, y_1)$$

$$p(y_t \mid y_{t-1}, \cdots, y_1) = \text{RNN}(h_t)$$ deterministic hidden states

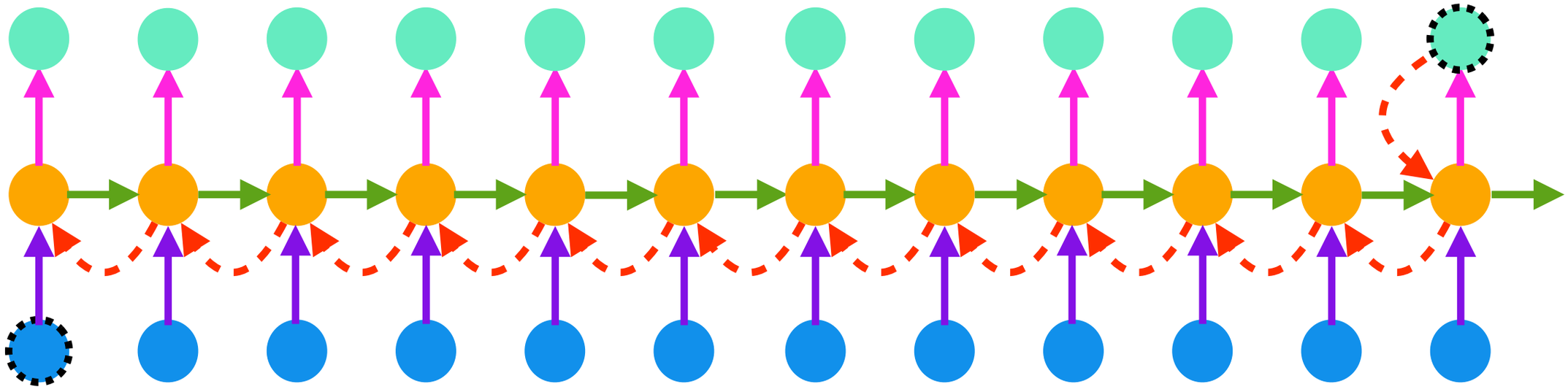
backpropagation

resulting in ***backpropagation through time (BPTT)***



Introducing dummy variable $W^{(t)}$,which are copies of W but only uses at time step t.

primary difficulty of training RNNs involves
propagating information over long horizons

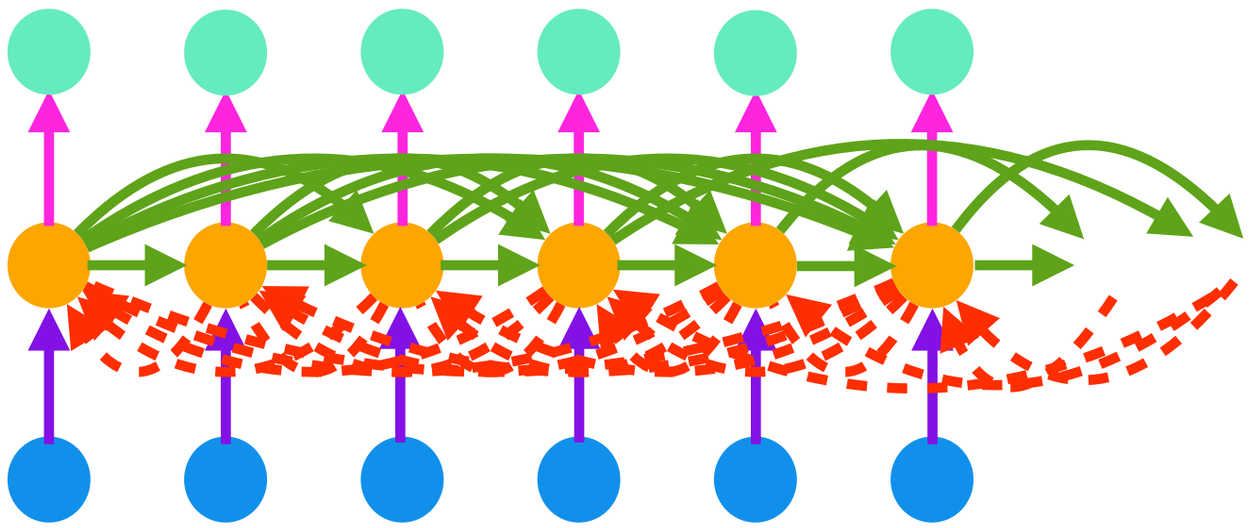


e.g. input at one step is predictive of output at *much later* step

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

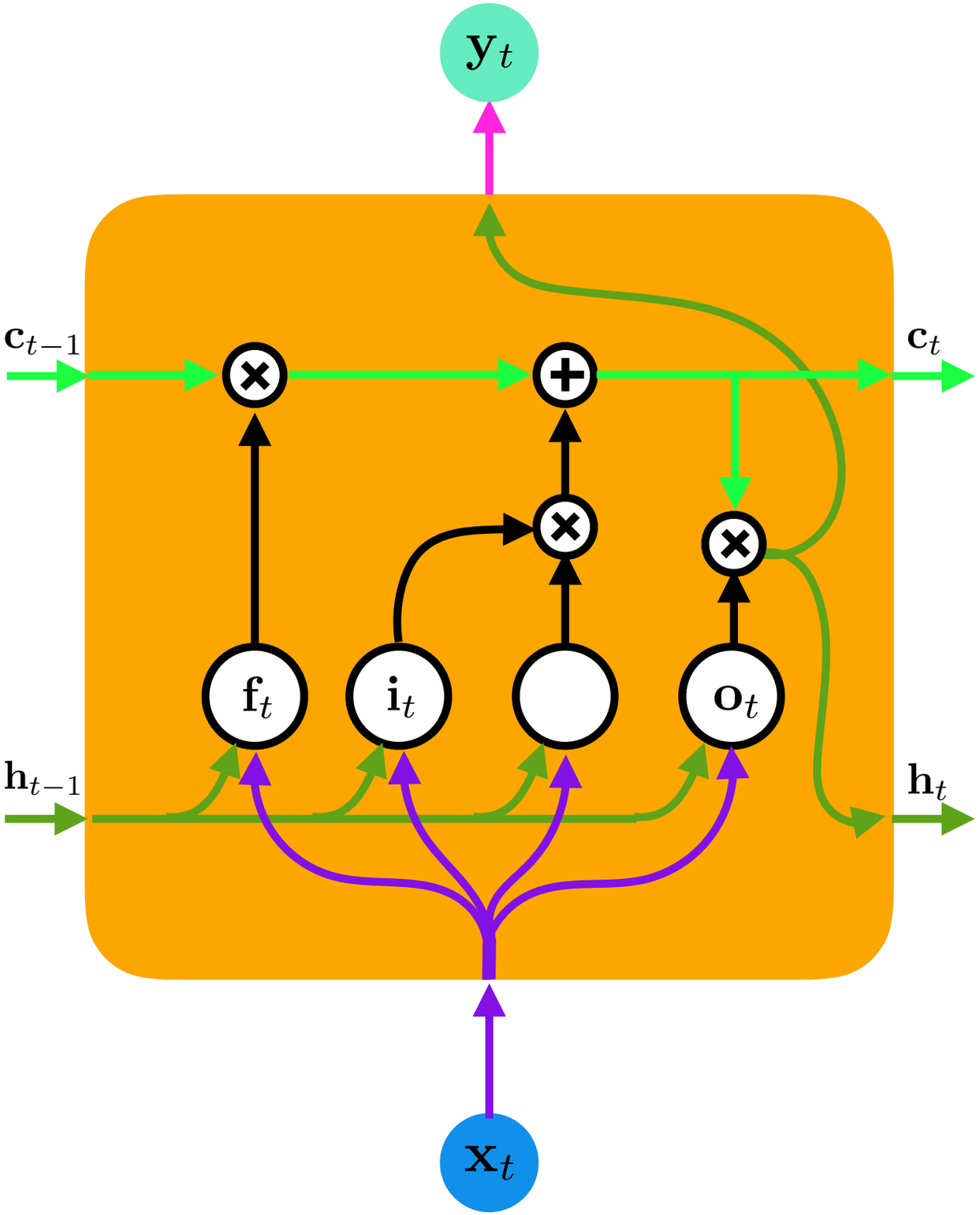
information, gradients can propagate more easily

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

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

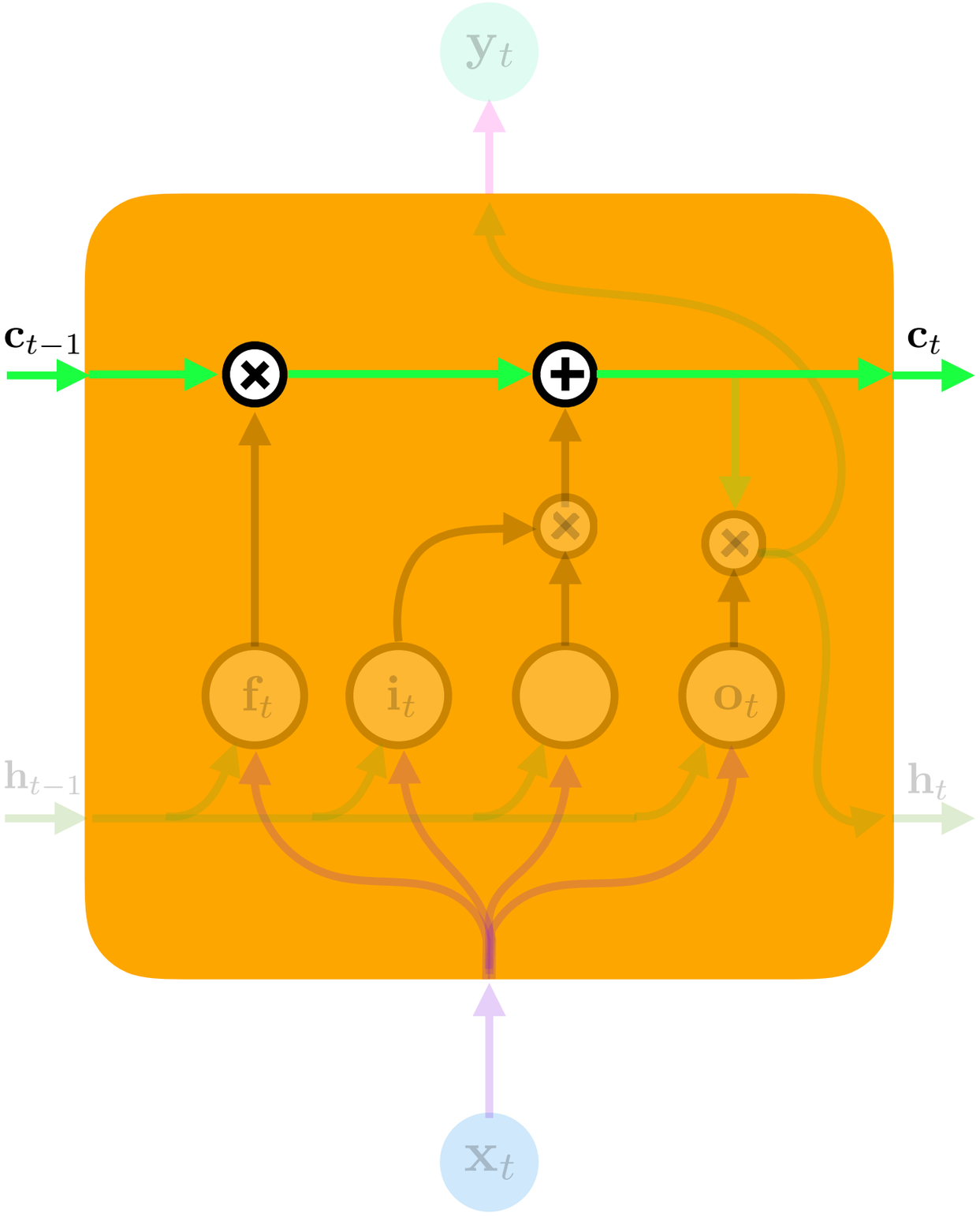
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

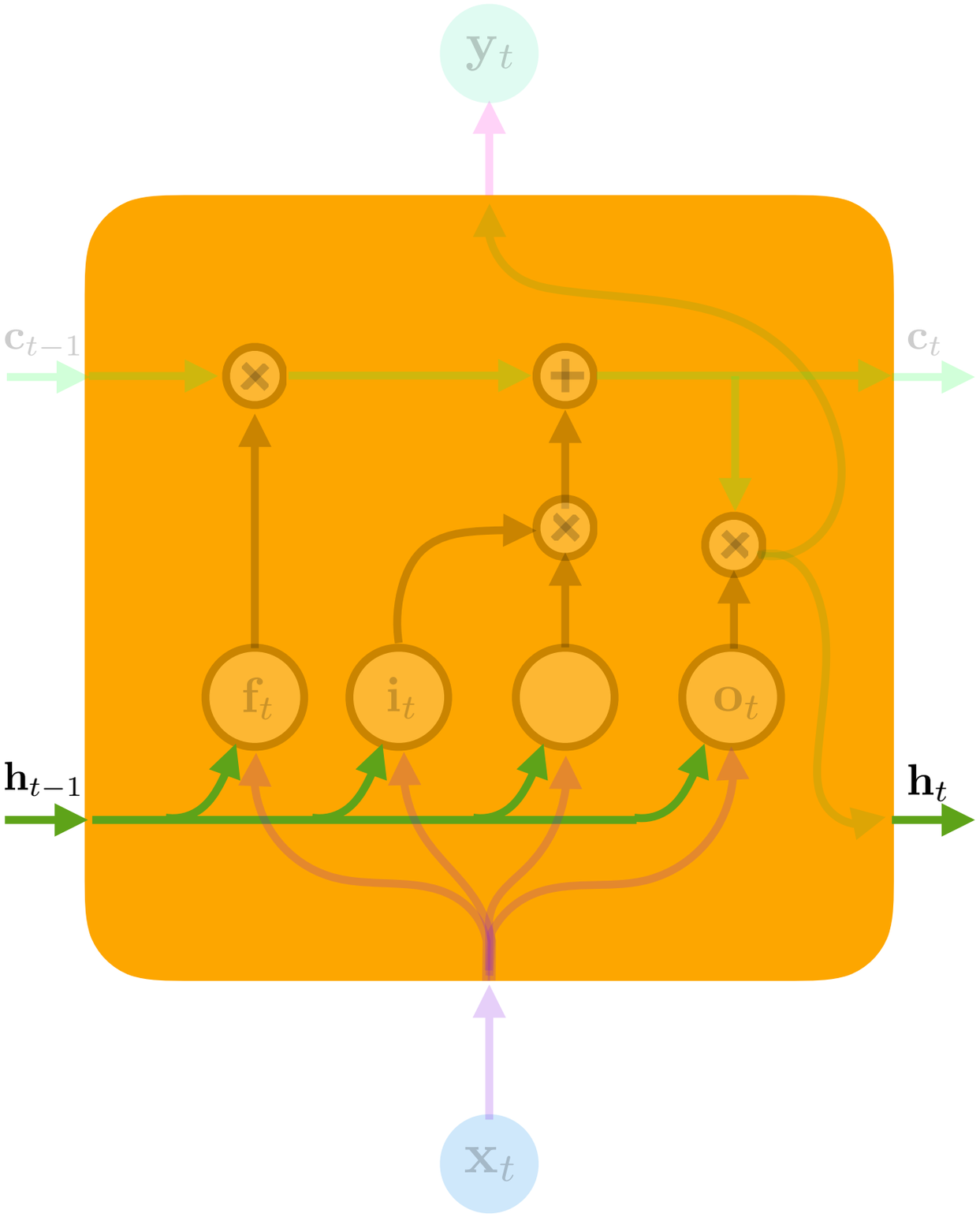
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

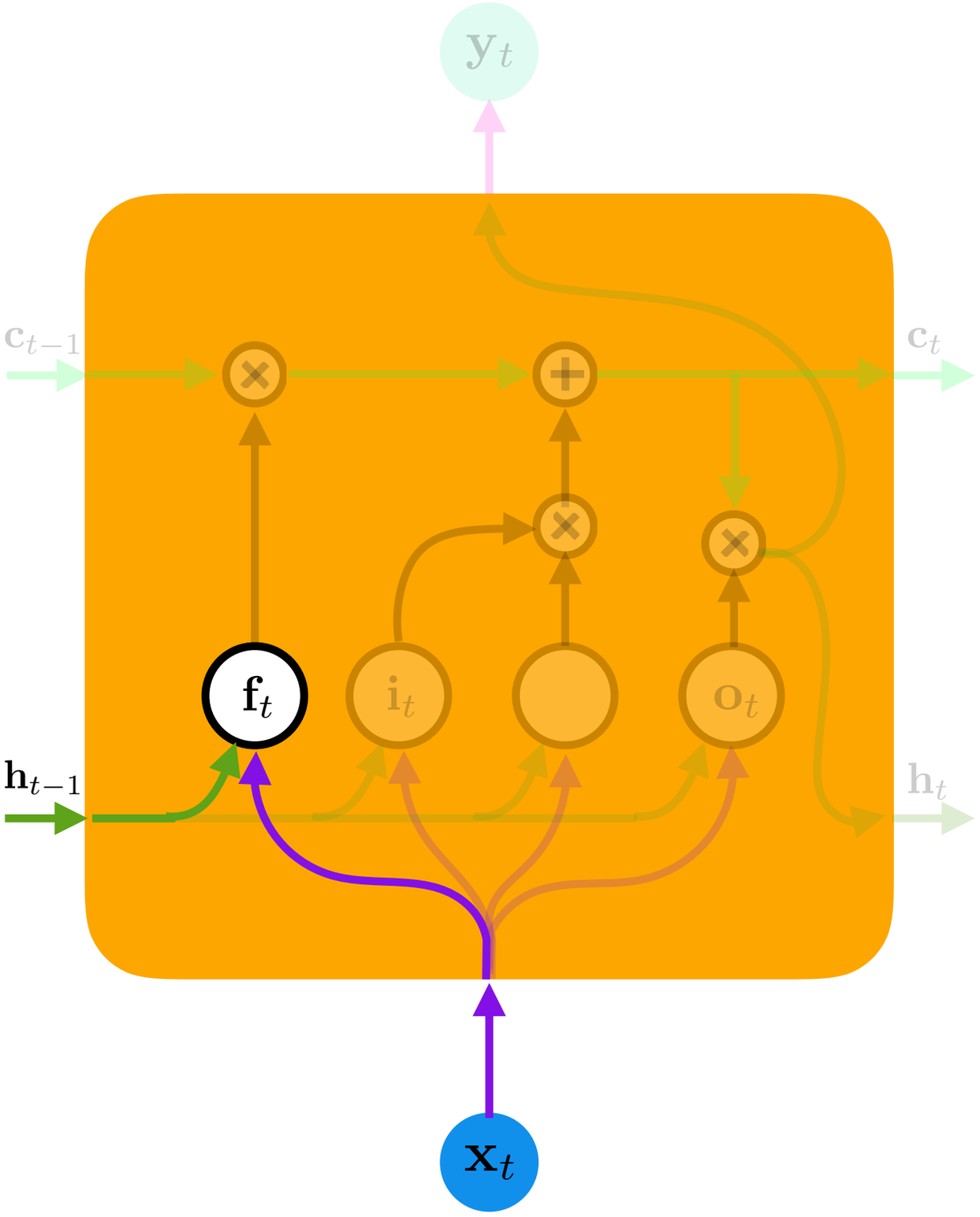
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

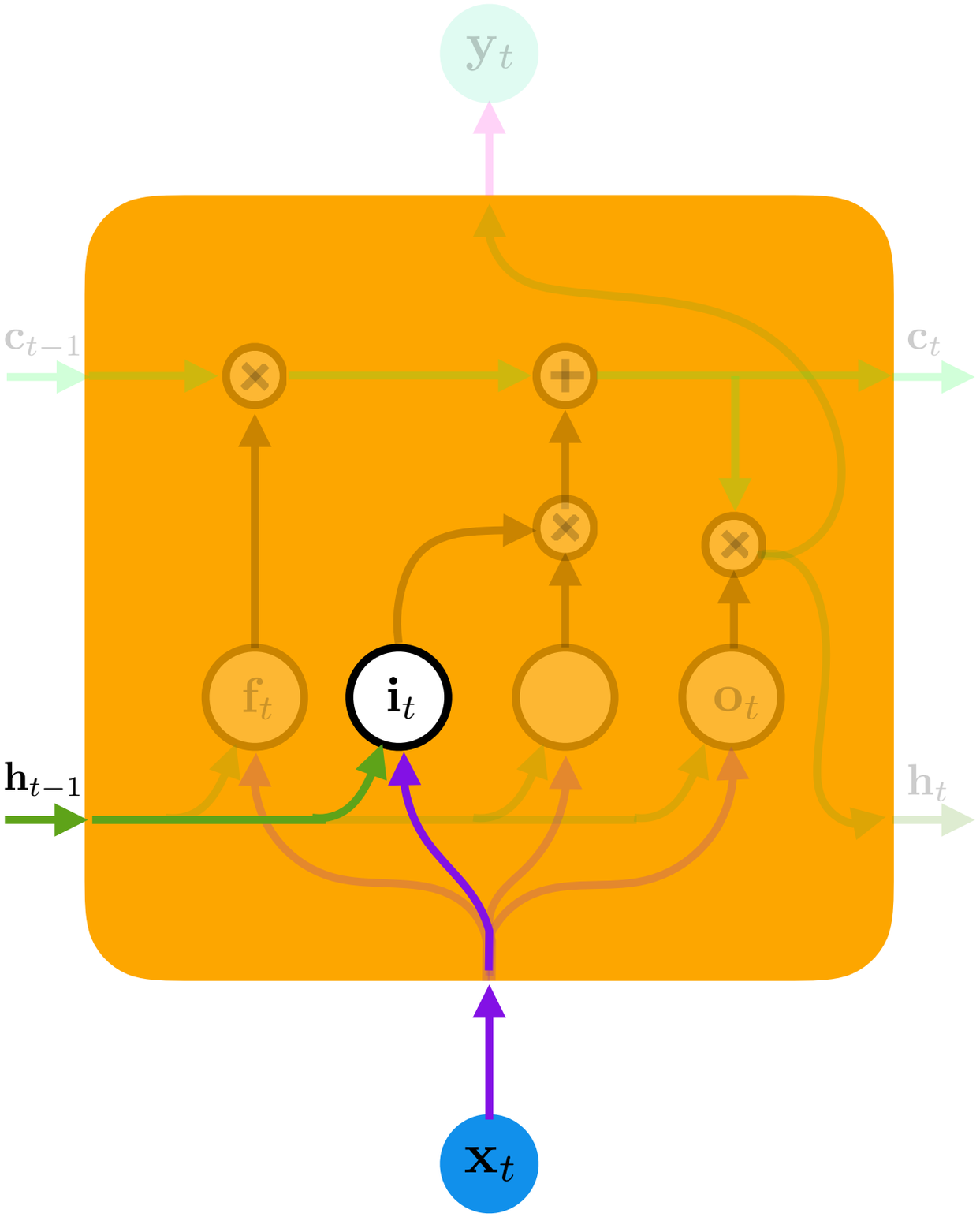
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal\mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

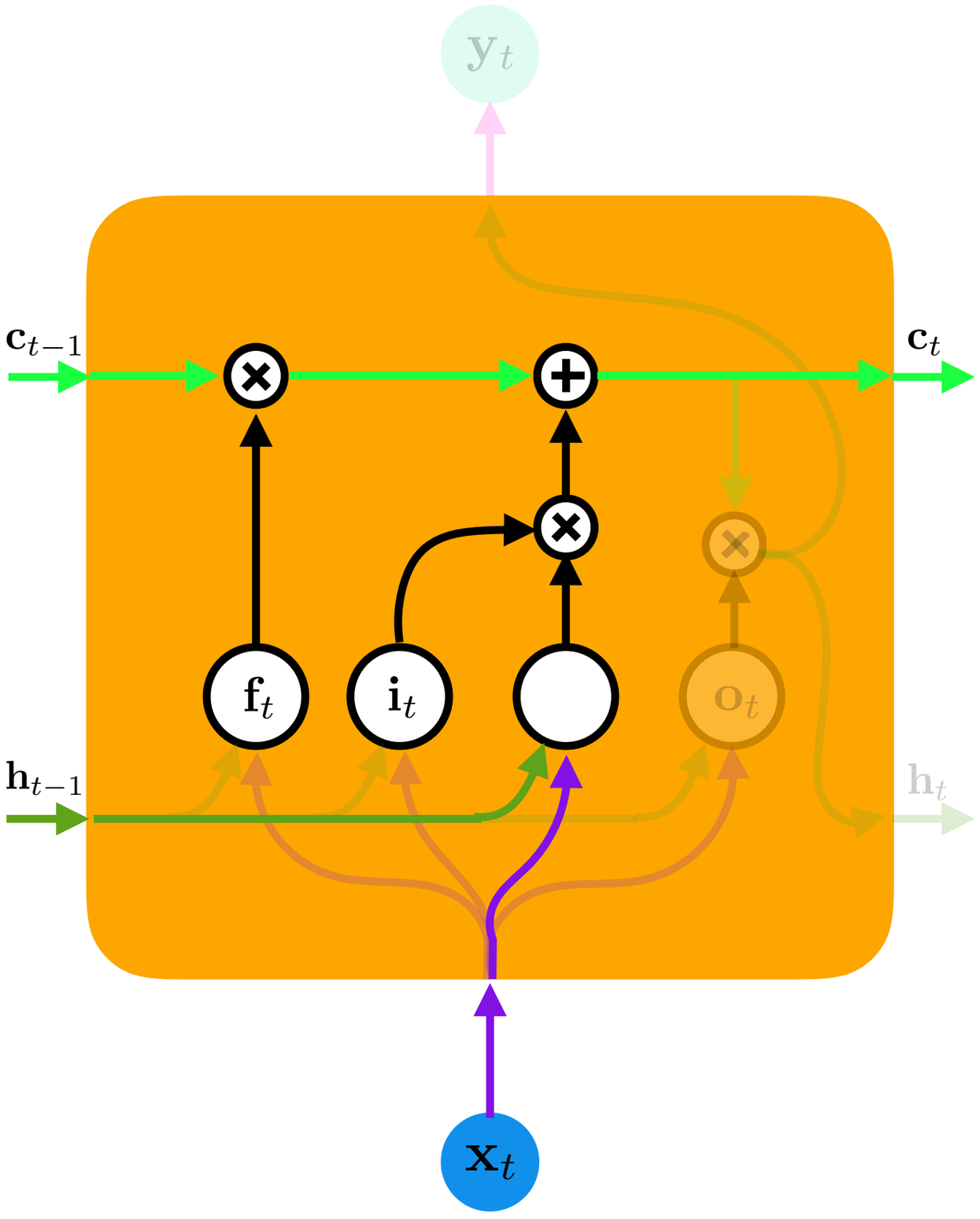
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

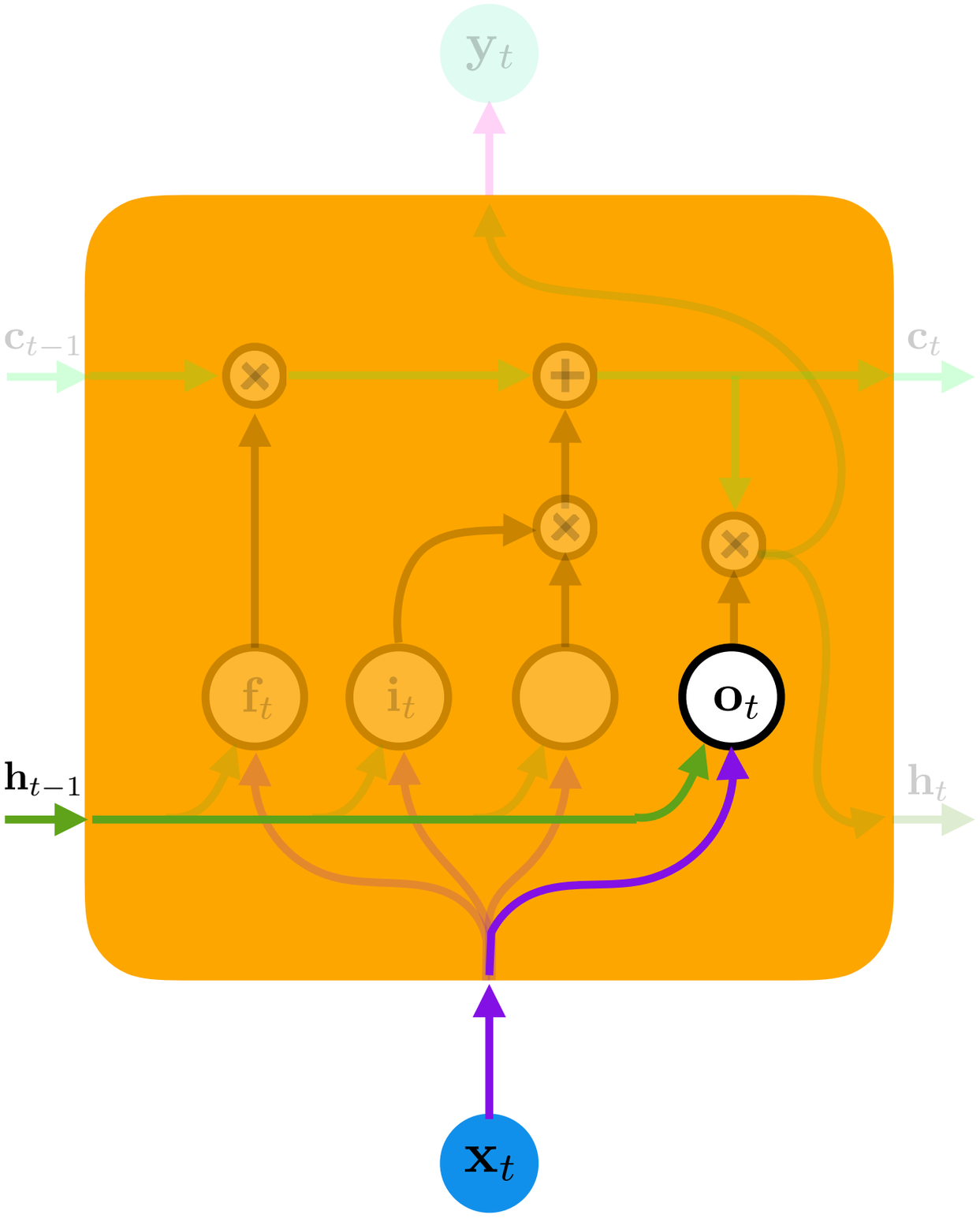
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

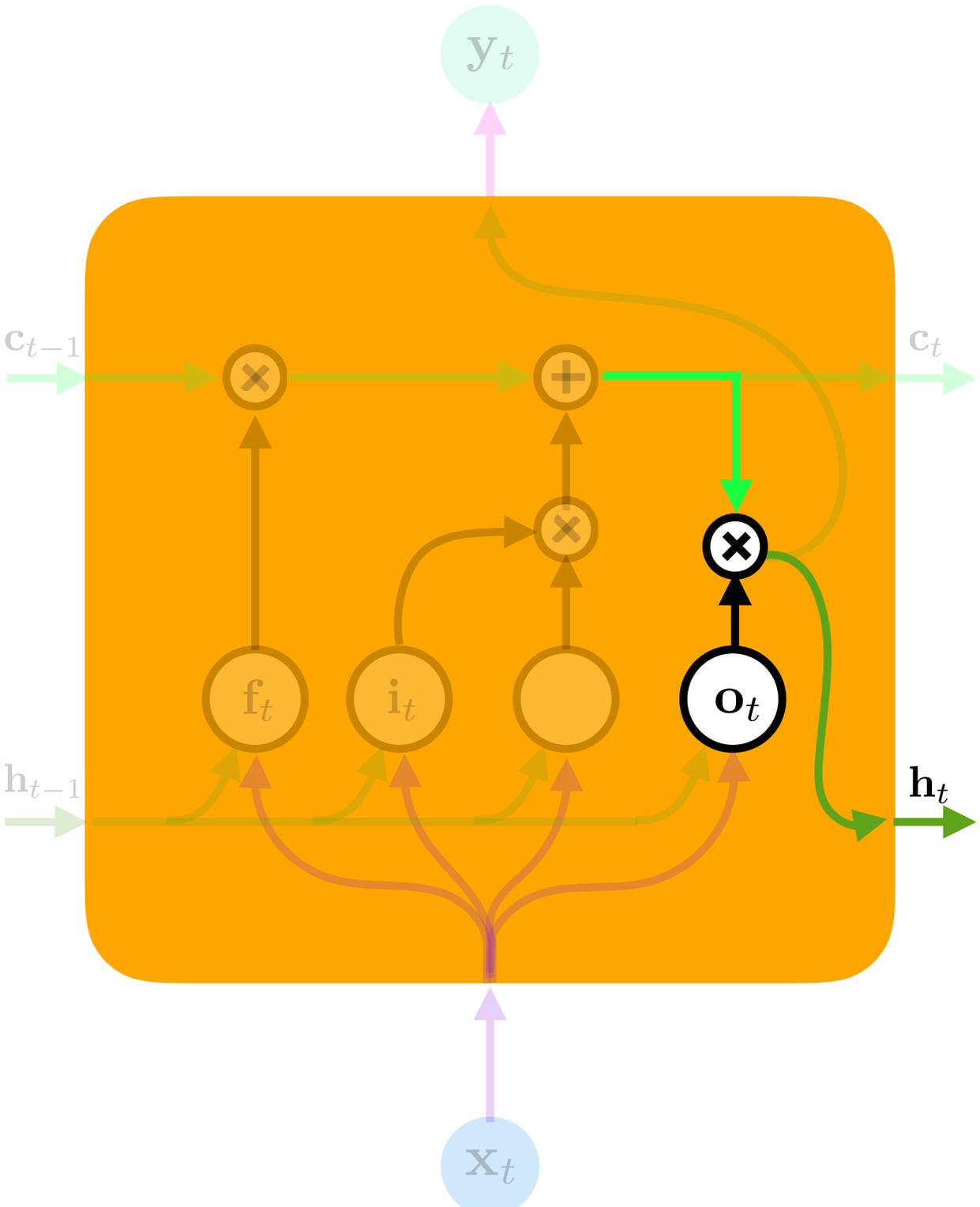
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

Hidden State

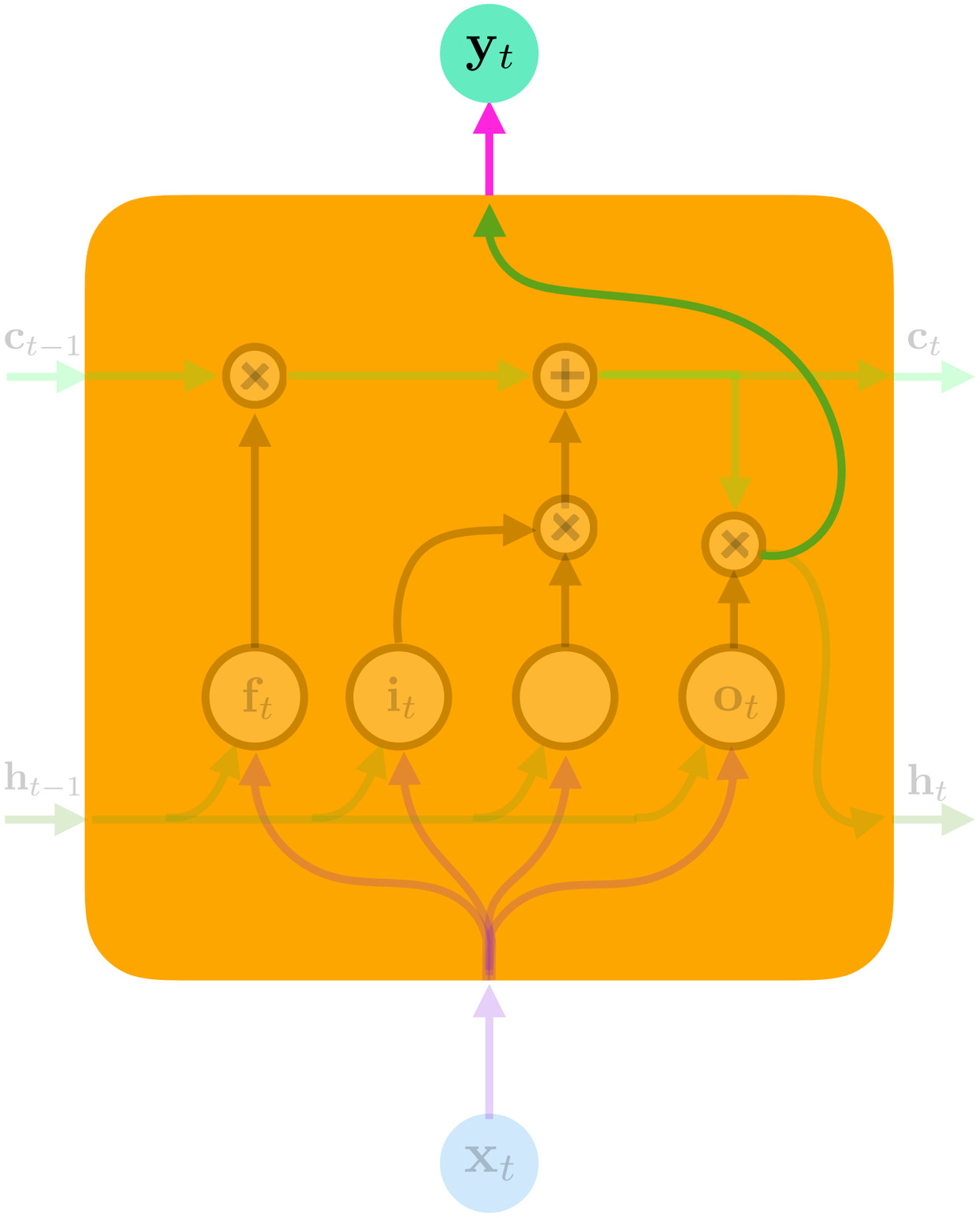
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

add trainable ***memory*** to the network

read from and *write* to "***cell***" state



Long Short-Term Memory (LSTM)

Forget Gate

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

Input Gate

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

Cell State

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_\mathbf{c}^\intercal[\mathbf{h}_{t-1}, \mathbf{x}_t])$$

Output Gate

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

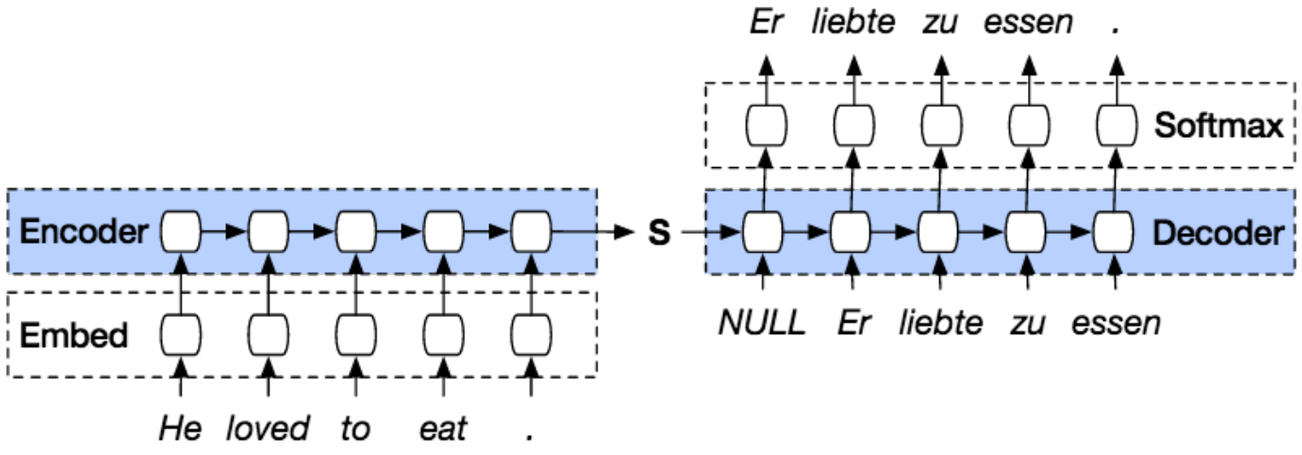
Hidden State

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

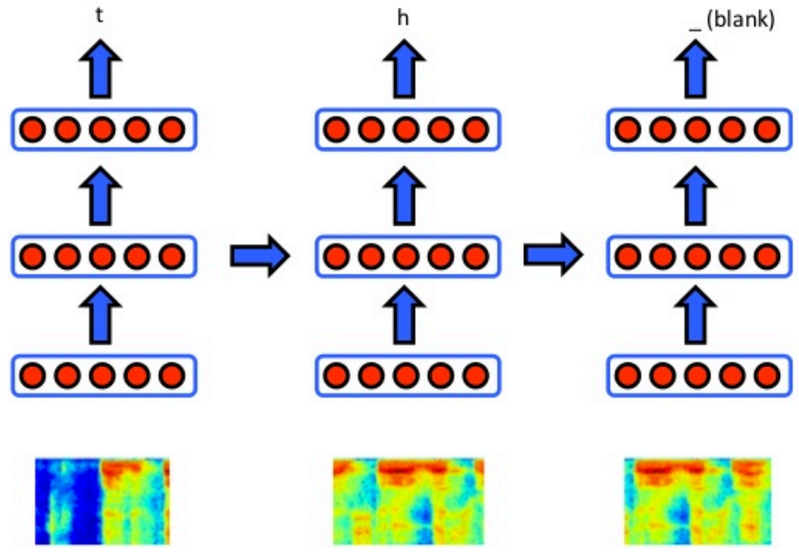
Output

$$\mathbf{y}_t = \sigma(\mathbf{W}_\mathbf{y}^\intercal \mathbf{h}_t)$$

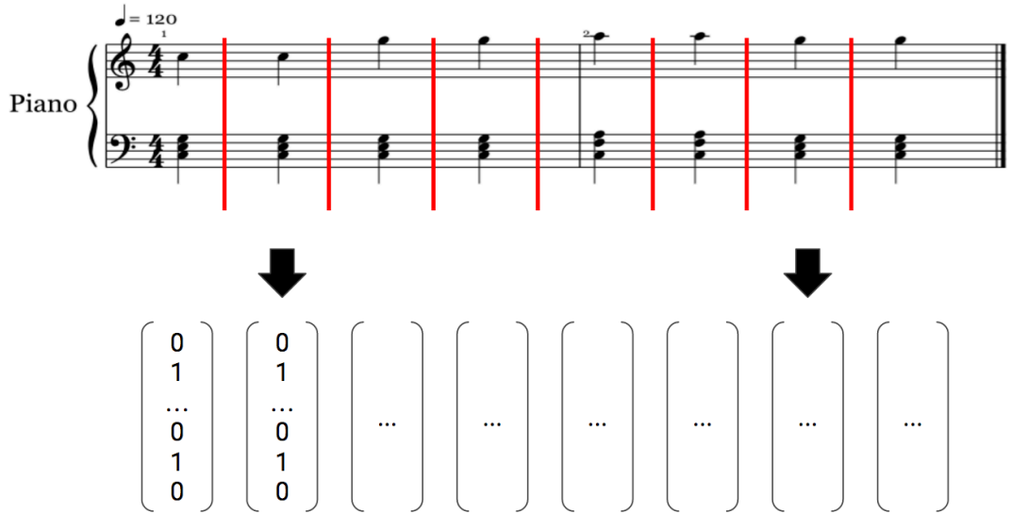
RNN Applications



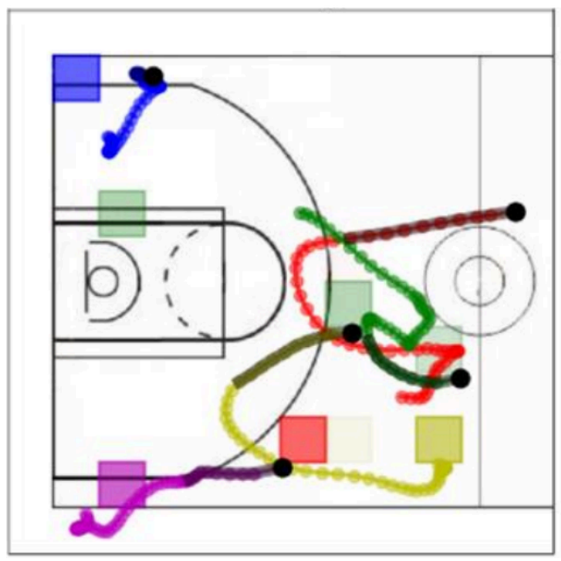
machine translation



speech recognition



music composition



trajectory generation