Previously we learned…

- Inductive Biases in Convolutional Networks
  - Locality Principle
  - Spatial Invariance
- Inductive Biases in Graph Neural Networks
  - Permutation Invariance

Recurrent Neural Networks

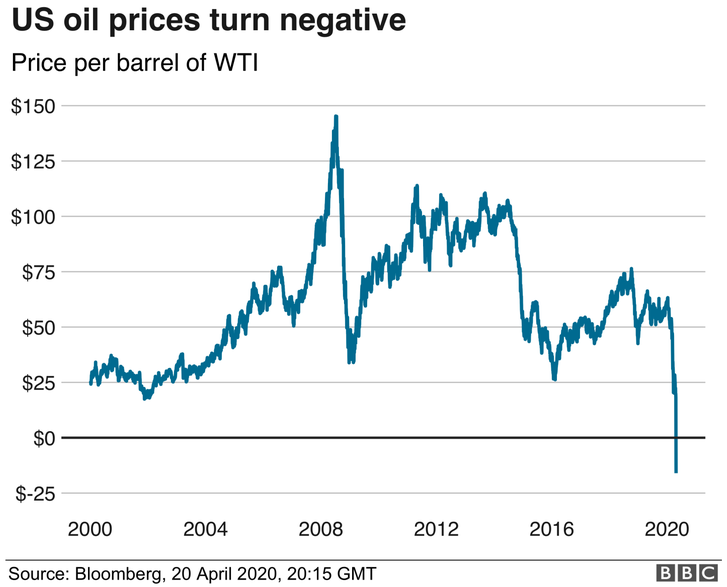
Principles of Deep Learning (AI502/IE408/IE511)

Sungbin Lim (UNIST AIGS & IE)

Contact: ai502deeplearning@gmail.com

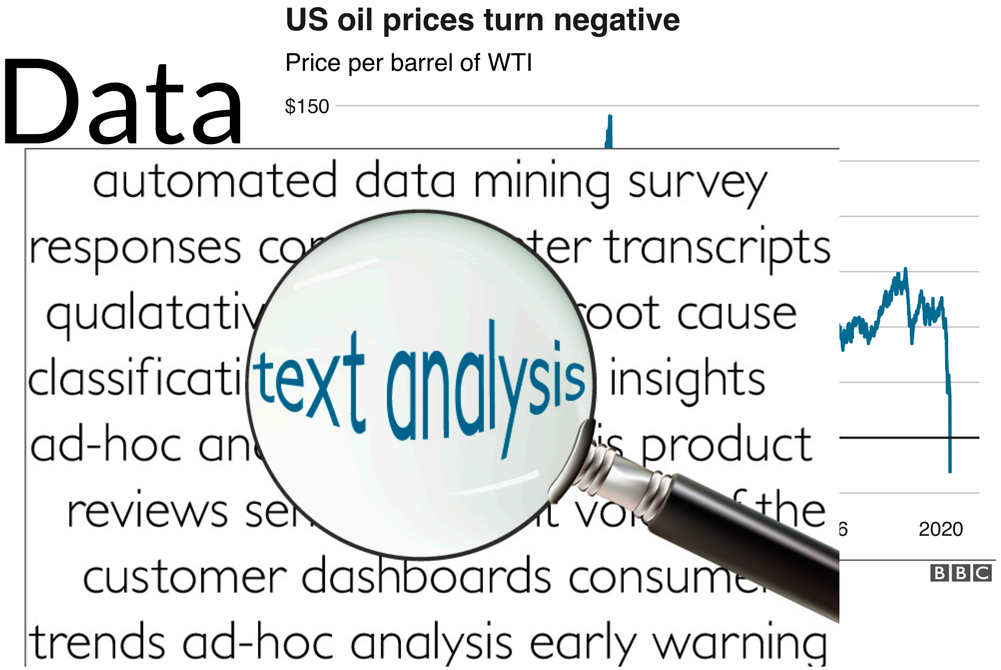
Understanding Sequential Data

- $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sequential data
  - time-series data



Understanding Sequential Data

- $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sequential data
  - time-series data
  - text data



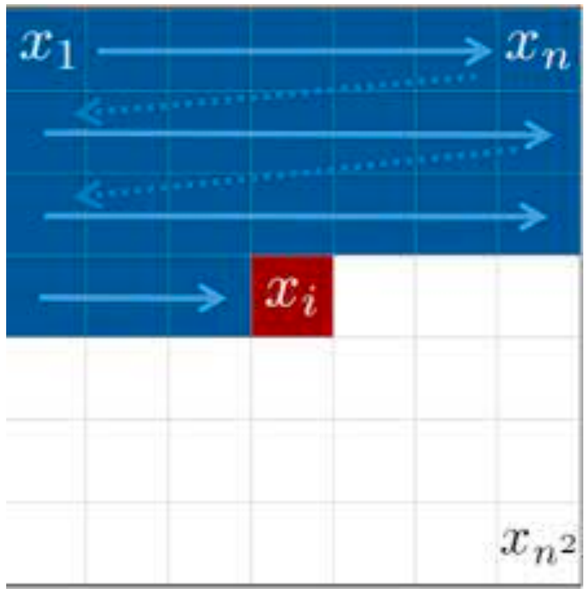
Understanding Sequential Data

- $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sequential data
  - time-series data
  - text data
  - reinforcement learning / planning

Understanding Sequential Data

- $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sequential data
  - time-series data
  - text data
  - reinforcement learning / planning
  - image fames in video
- Music, speech, text, videos, dialogue are all sequential

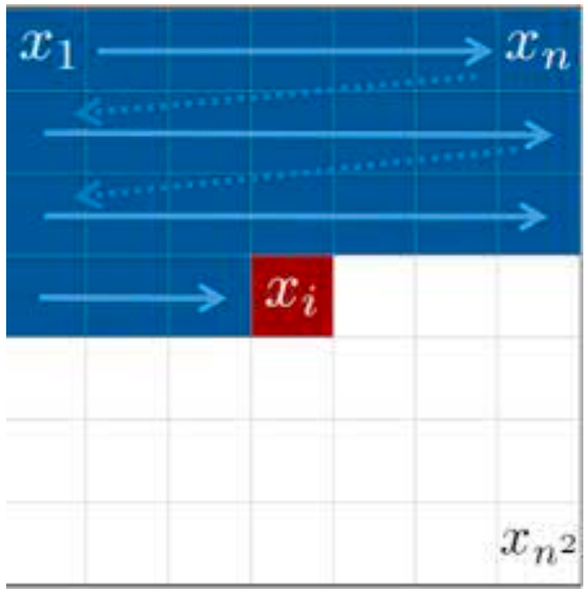
Aren’t image data sequential?



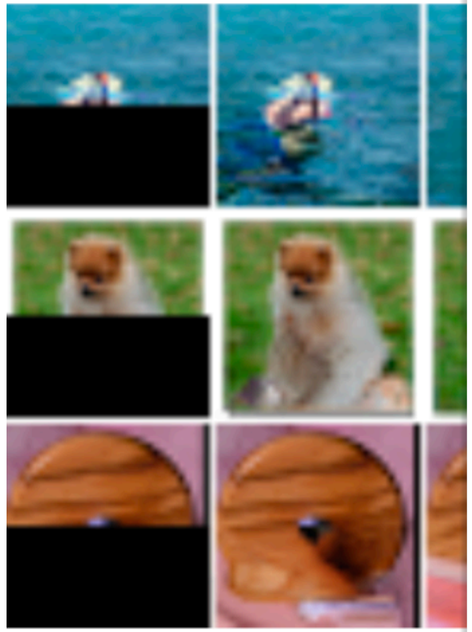
Pixel RNN

Input Generation Ground-truth

Aren’t image data sequential?

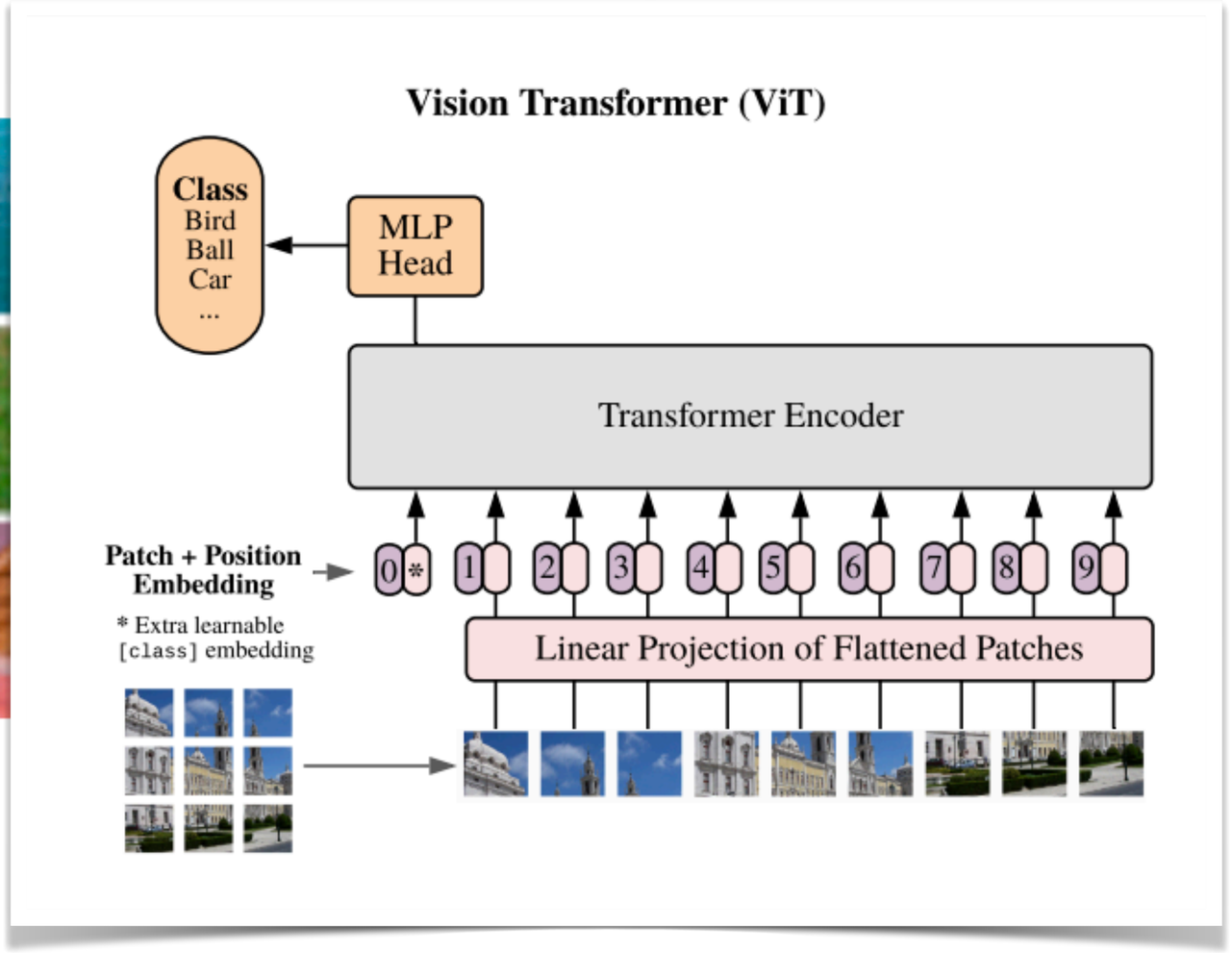


Pixel RNN



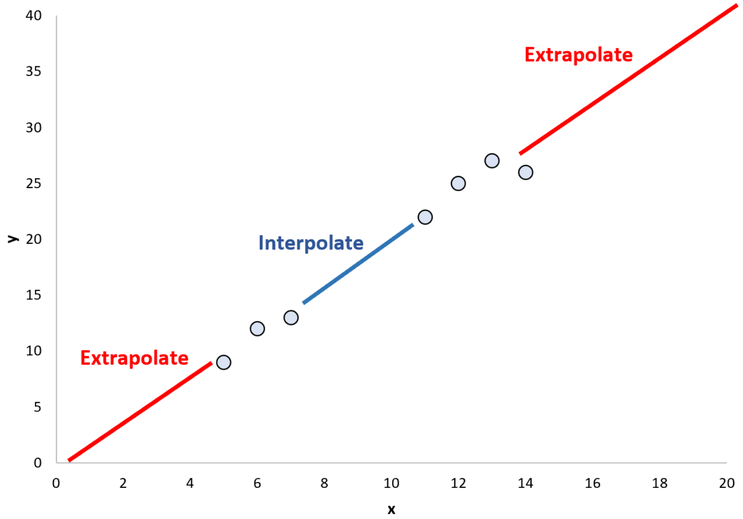
Input

sequential model based approaches draw attention from ML community



Why sequential patterns are challenging?

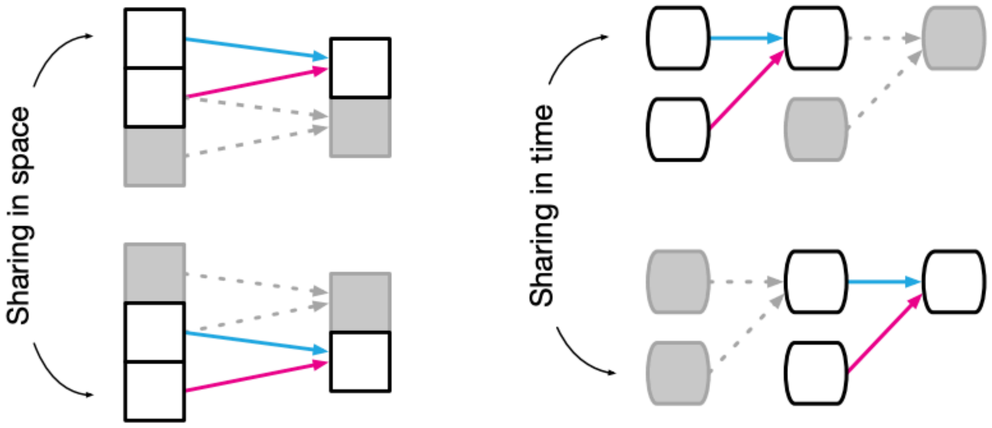
- Sequence length is not fixed
  - variable length → MLP(x), CNN(o)
- Extrapolation vs Interpolation
  - hindsight is easier than foresight (ex. stock price)
- Sequential data are **non-i.i.d**.
  - sequence order is crucial
  - if we permute them, they would make little sense
  - *dog* bites *man* vs *man* bites *dog*



it is possible… huh?

Invariance in Sequential Data

- ***Sequentiality***
  - inputs should be used sequentially → to deal with **non-i.i.d.** data
- ***Temporal Invariance***
  - translation invariance in sequence order



Convolutional layer Sequential Models

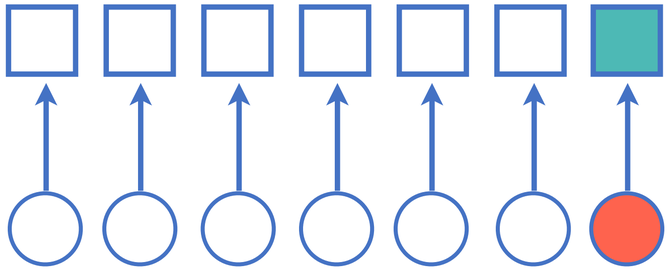
this is also called stationary property according to statistician

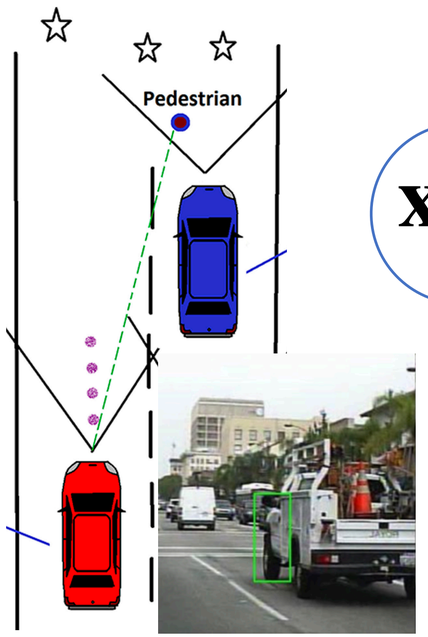
Conditionally Independent Model

- Batch prediction via single-input model

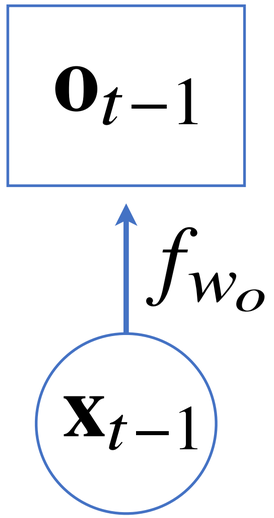
$$\mathbf{o}_t = f_{w_o}(\mathbf{x}_t)$$

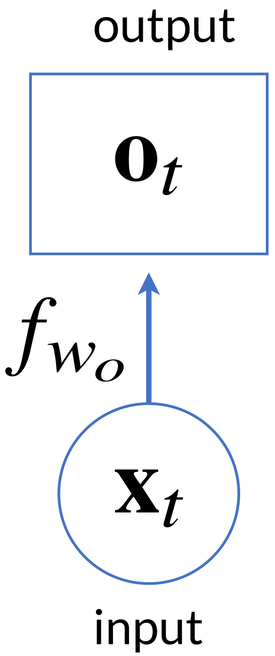
This approach cannot deal with *sequentiality*





Pedestrian occlusion



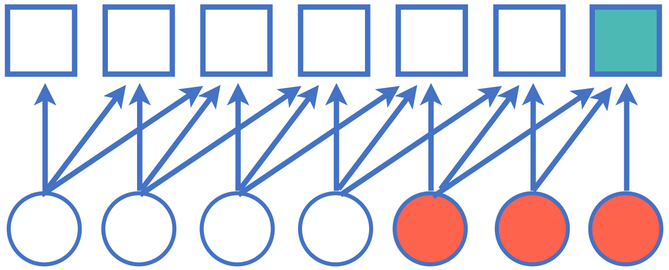


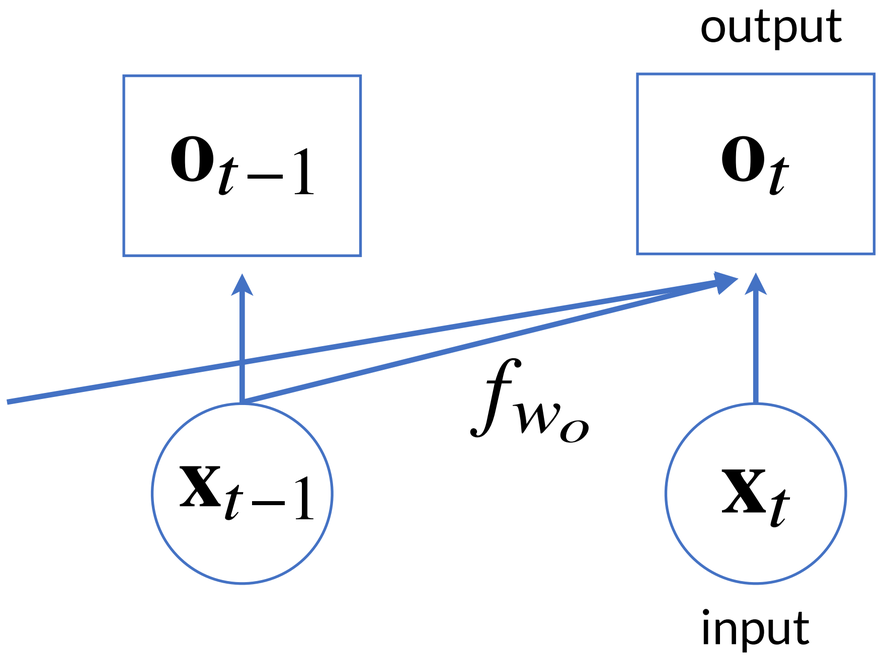
Autoregressive Model

- Autoregressive Model with fixed length

$$\mathbf{o}_t = f_{w_o}(\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-\ell})$$

AR model utilizes fixed-length inputs
hence cannot use the *long-term memory*





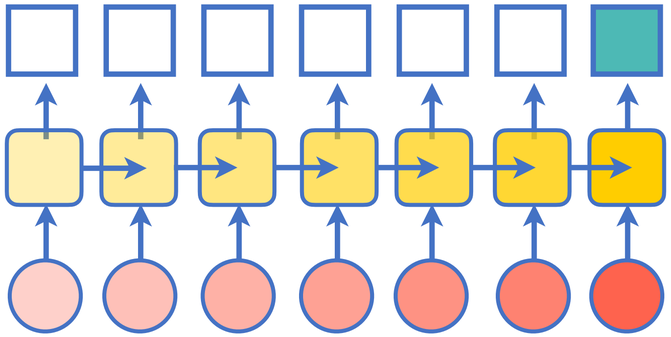
Latent Autoregressive Model

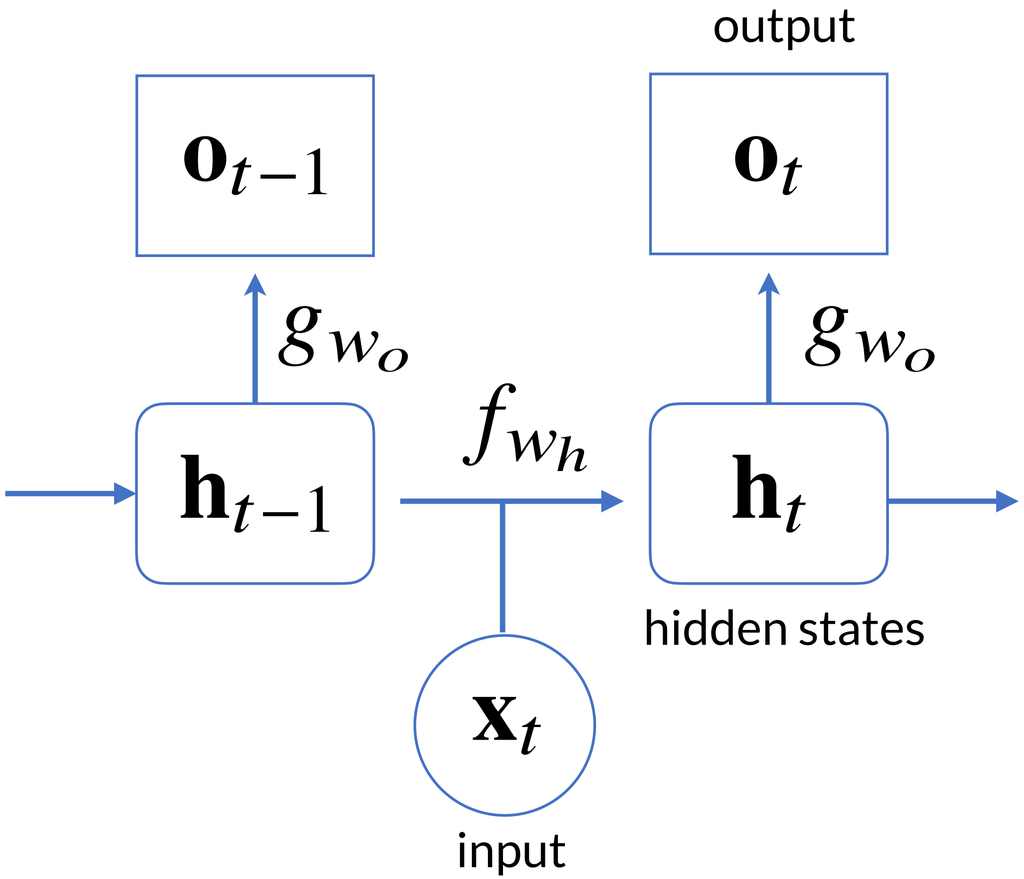
- Latent Autoregressive Model

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

$$\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

Latent AR model can deal with variable-length sequence





Recurrent Neural Network

- Latent Autoregressive Model

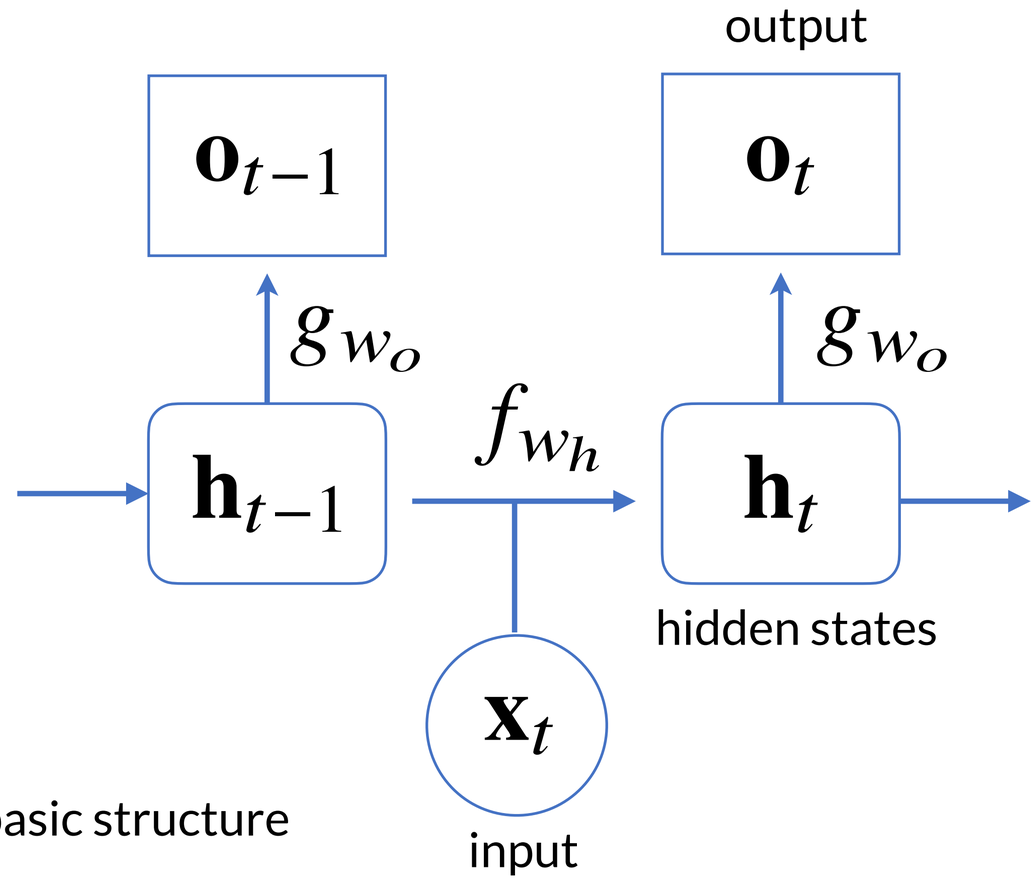
$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

$$\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

- Recurrent Neural Network

$$\mathbf{h}_t = \phi(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + b_h)$$

$$\mathbf{o}_t = W_{oh}\mathbf{h}_t + b_o$$



this is a basic structure of RNN

Recurrent Neural Network

- Latent Autoregressive Model

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

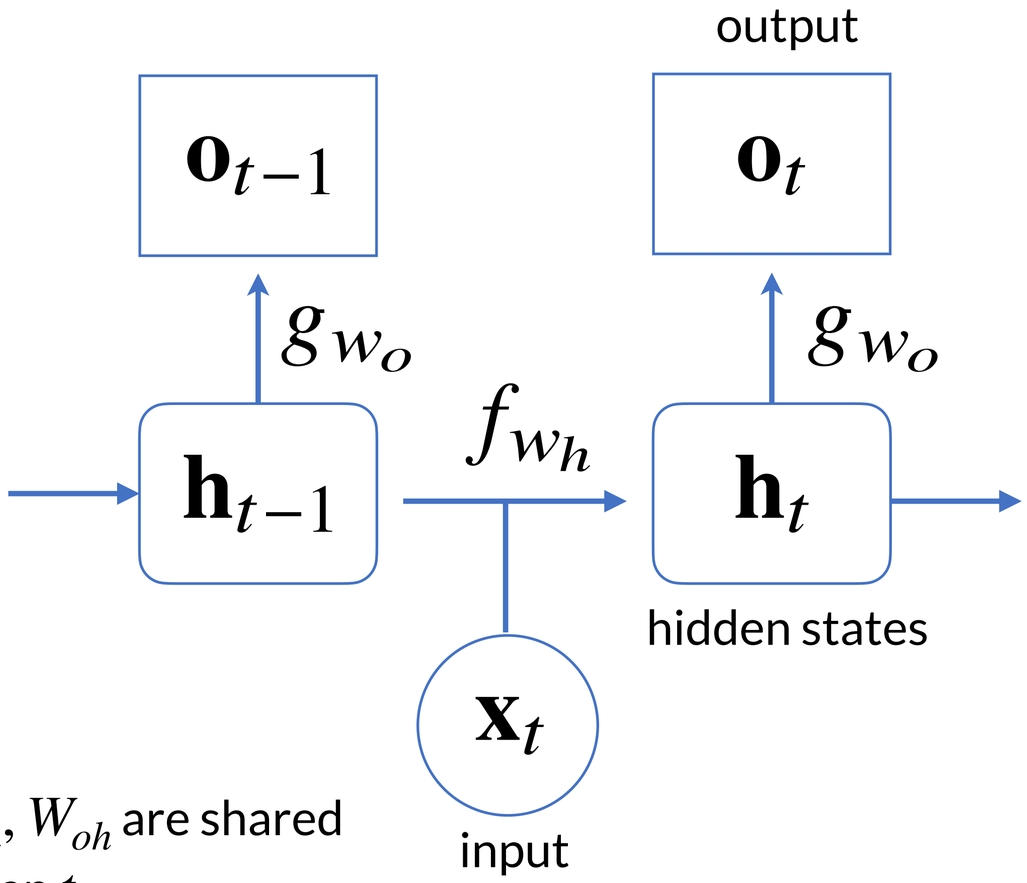
$$\mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

- Recurrent Neural Network

$$\mathbf{h}_t = \phi(W_{hx}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + b_h)$$

$$\mathbf{o}_t = W_{oh}\mathbf{h}_t + b_o$$

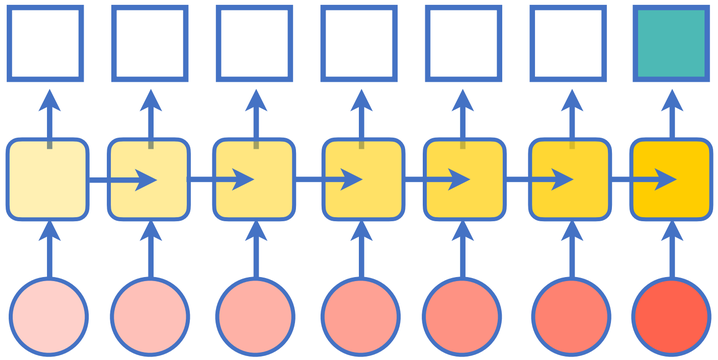
W_{hx}, W_{hh}, W_{oh} are shared among step t



Backpropagation Through Time

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

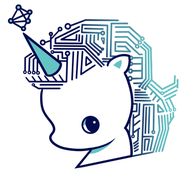
$$L(x, y, w_h, w_o) = \sum_{t=1}^{T} \ell(y_t, o_t)$$



Backpropagation Through Time

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

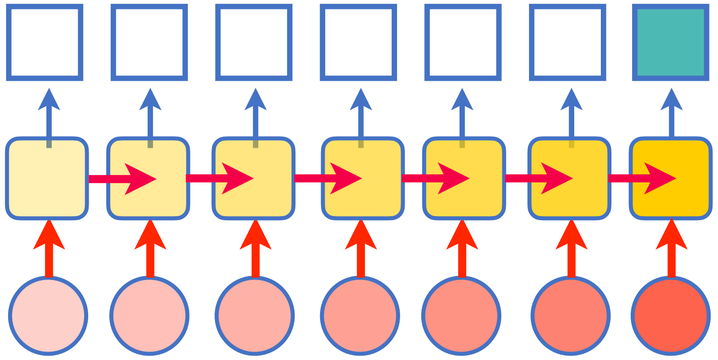
$$L(x, y, w_h, w_o) = \sum_{t=1}^{T} \ell(y_t, o_t)$$



apply the chain rule for
total derivatives

$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^{T} \partial_{w_h} \ell(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} \ell(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$

$$\frac{\mathrm{d} f(y(x), x)}{\mathrm{d} x} = \frac{\partial f}{\partial x}\frac{\partial y}{\partial x} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial x}$$



Backpropagation Through Time

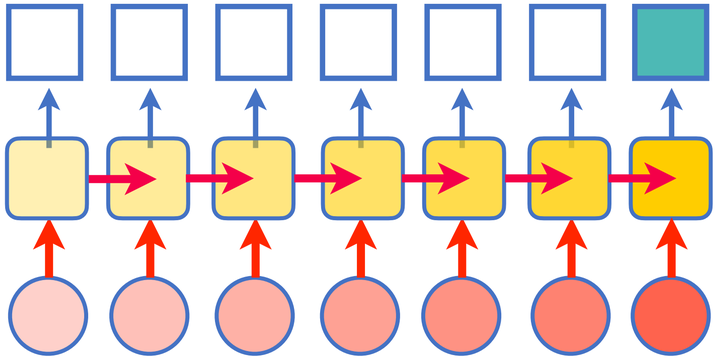
$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

$$L(x, y, w_h, w_o) = \sum_{t=1}^{T} \ell(y_t, o_t)$$

we need to compute ∂_{w_h} recurrently!

$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^{T} \partial_{w_h} \ell(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} \ell(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$

$$\partial_w h_t = (\partial_w f)(x_t, h_{t-1}, w) + (\partial_h f)(x_t, h_{t-1}, w) \partial_w h_{t-1}$$

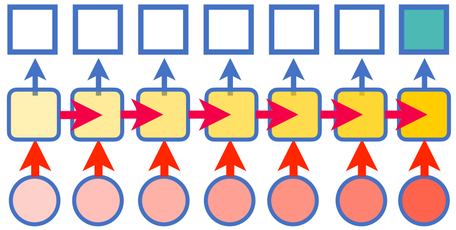


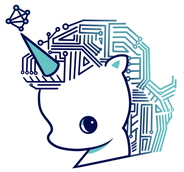
Backpropagation Through Time

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

$$L(x, y, w_h, w_o) = \sum_{t=1}^{T} \ell(y_t, o_t)$$

$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^{T} \partial_{w_h} \ell(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} \ell(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$





this is called BPTT

$$\partial_{w_h} h_t = \partial_{w_h} f(x_t, h_{t-1}, w_h) + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \partial_{h_{j-1}} f(x_j, h_{j-1}, w_h) \right) \partial_{w_h} f(x_i, h_{i-1}, w_h)$$

Backpropagation Through Time

$$\mathbf{h}_t = f_{w_h}(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad \mathbf{o}_t = g_{w_o}(\mathbf{h}_t)$$

$$L(x, y, w_h, w_o) = \sum_{t=1}^{T} \ell(y_t, o_t)$$

$$\partial_{w_h} L(x, y, w_h, w_o) = \sum_{t=1}^{T} \partial_{w_h} \ell(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} \ell(y_t, o_t) \partial_{h_t} g(h_t, w_h) [\partial_{w_h} h_t]$$

what will happen if $1 < \|\partial_{h_{j-1}} f\|$ occur?

$$\partial_{w_h} h_t = \partial_{w_h} f(x_t, h_{t-1}, w_h) + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \partial_{h_{j-1}} f(x_j, h_{j-1}, w_h) \right) \partial_{w_h} f(x_i, h_{i-1}, w_h)$$

Full BPTT is *not* good

- Full BPTT is *very slow* and gradients can blow up
  - minimal changes in the initial conditions lead to disproportionate changes in the outcome
  - **truncation** is needed for numerical stability
- Two solutions
  - truncated BPTT (Jaeger, 2005): truncate the sum after τ steps
  - randomized truncation (Tallec & Ollivier, 2018)

Truncated BPTT

- Approximation of the true gradient
  - terminating the sum after τ steps
  - use `detach` in PyTorch
  - in practice, this works quite well
- Model focuses primarily on short-term influence rather than long-term one

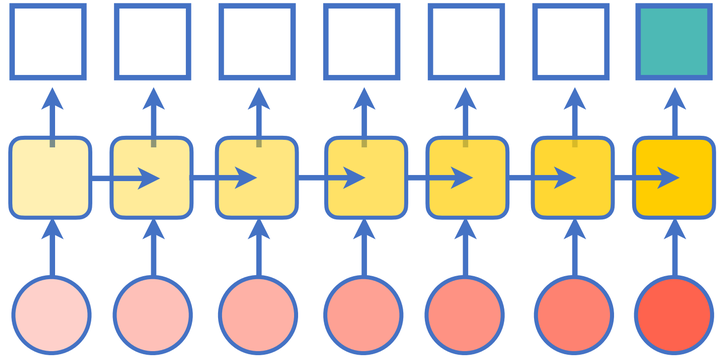
```
# non-truncated
for t in range(T):
    out = model(out)
out.backward()

# truncated to the last K timesteps
for t in range(T):
    out = model(out)
    if T - t == K:
        out.detach()
out.backward()
```

$$\partial_{w_h} h_t = \partial_{w_h} f(x_t, h_{t-1}, w_h) + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} \partial_{h_{j-1}} f(x_j, h_{j-1}, w_h) \right) \partial_{w_h} f(x_i, h_{i-1}, w_h)$$

$i = t - k$

this is easy-to-implement

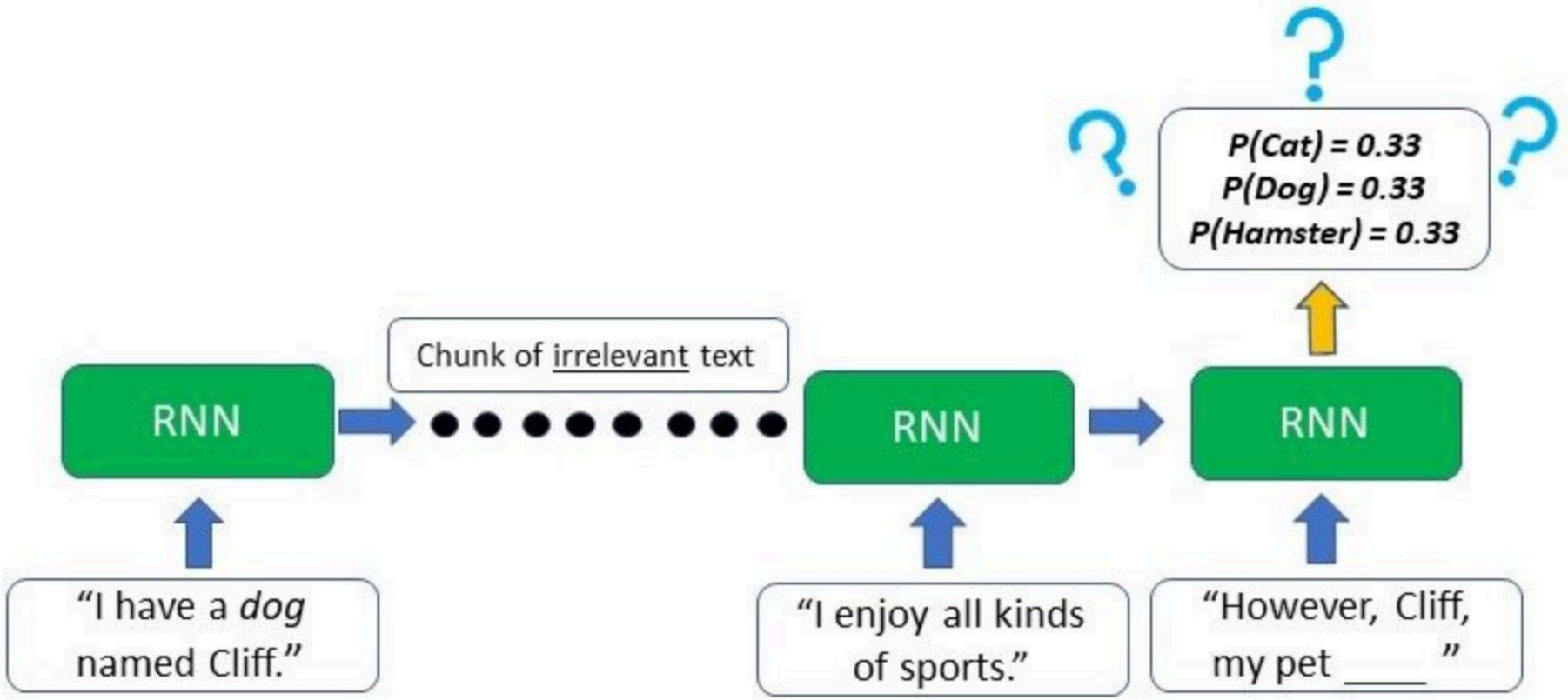
Training tips for RNN



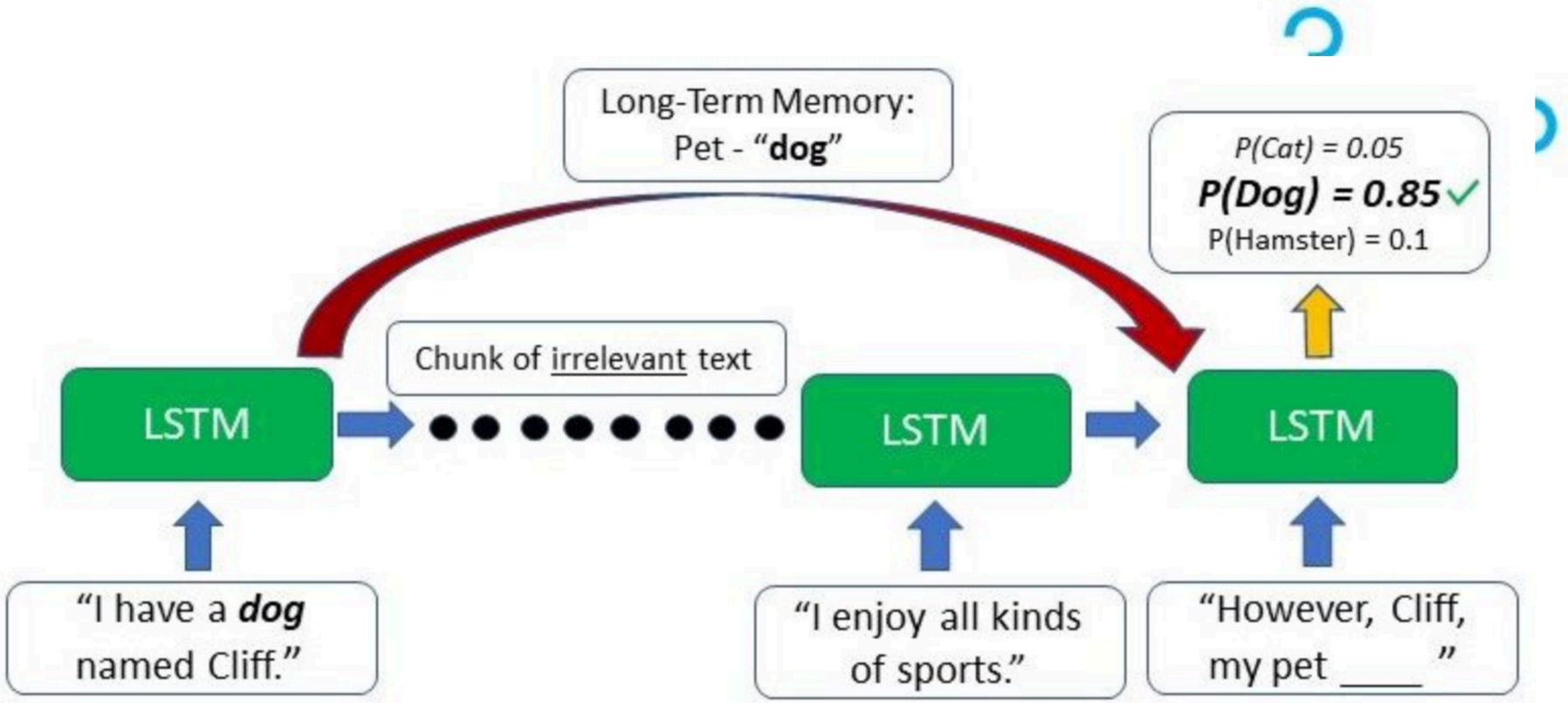
- We need to take care of the following changes
  - long products of matrices can lead to vanishing or divergent gradients
  - sequential partitioning leads to better models
  - we usually utilize ***gradient clipping*** before updating the parameters
- Gradient clipping
  - if gradient explodes, neural network training often diverges and is unstable owing to massive spikes in the loss
  - shrinking learning rate can be an option, but gradient clipping is more useful

$$\mathbf{g} \leftarrow \min\left(1, \frac{\xi}{\|\mathbf{g}\|}\right)\mathbf{g}$$

Beyond simple RNN



Beyond simple RNN



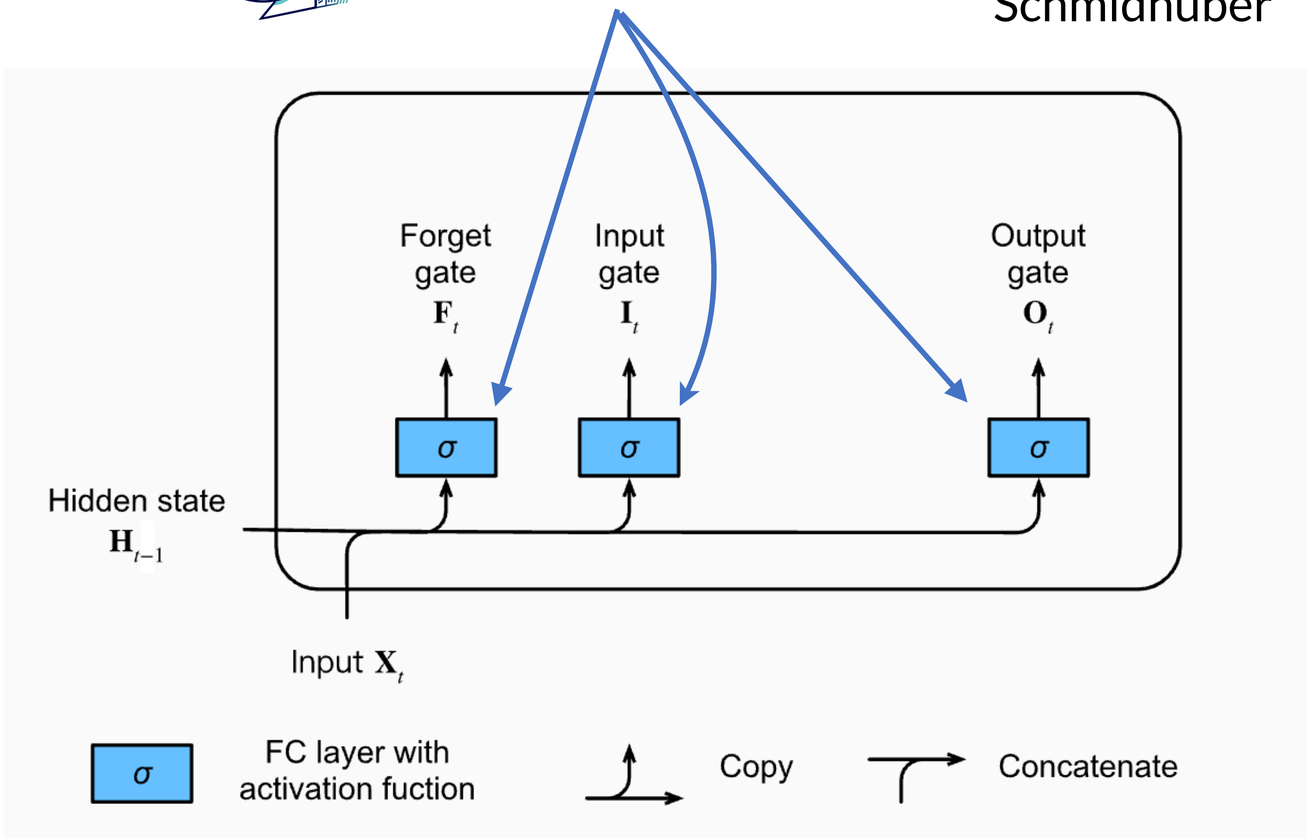
Long-Short Term Memory (LSTM)

Schmidhuber

we use sigmoid here

- Input, Forget, and Output gates
  - **input** gate is needed to decide when to read data
  - **forget** gate reset contents
  - **output** gate read out the entries

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$
$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$



Long-Short Term Memory (LSTM)

- **Candidate** Memory cell

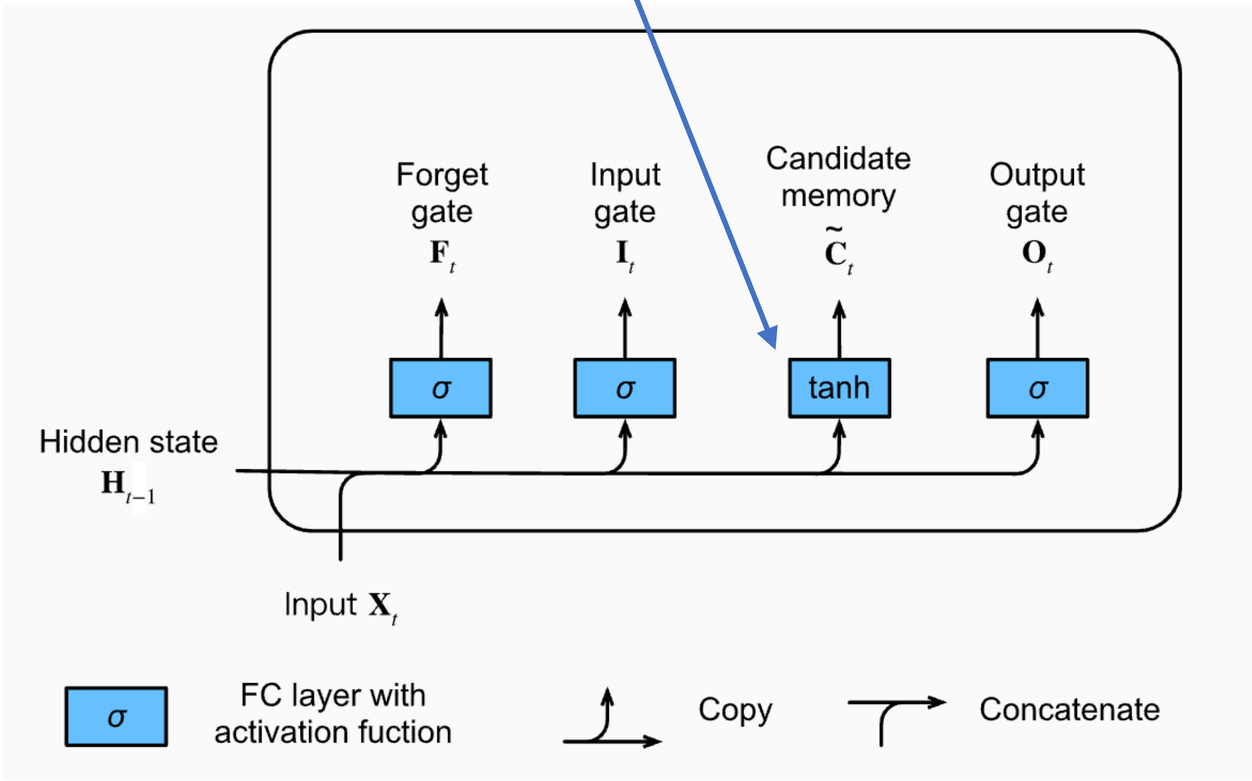
this is tanh activation

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c).$$

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$



Long-Short Term Memory (LSTM)

this alleviate the vanishing gradient problem

- **Memory** cell
  - take new data via $\mathbf{I}_t$ and $\tilde{\mathbf{C}}_t$ and retain old memory cell $\mathbf{C}_{t-1}$ using $\mathbf{F}_t$

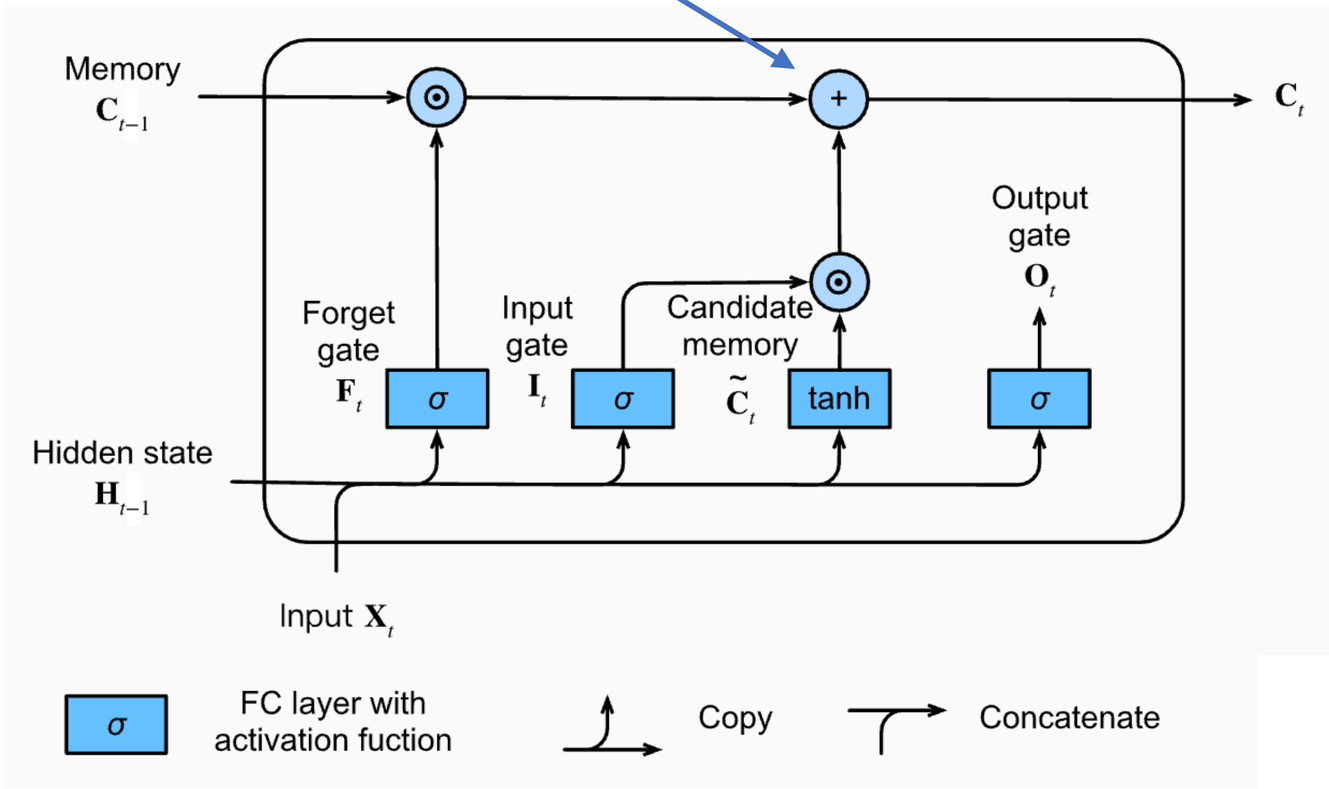
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c).$$

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$



Long-Short Term Memory (LSTM)

this captures time dependency

- **Hidden** states
  - gated version of memory

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t).$$

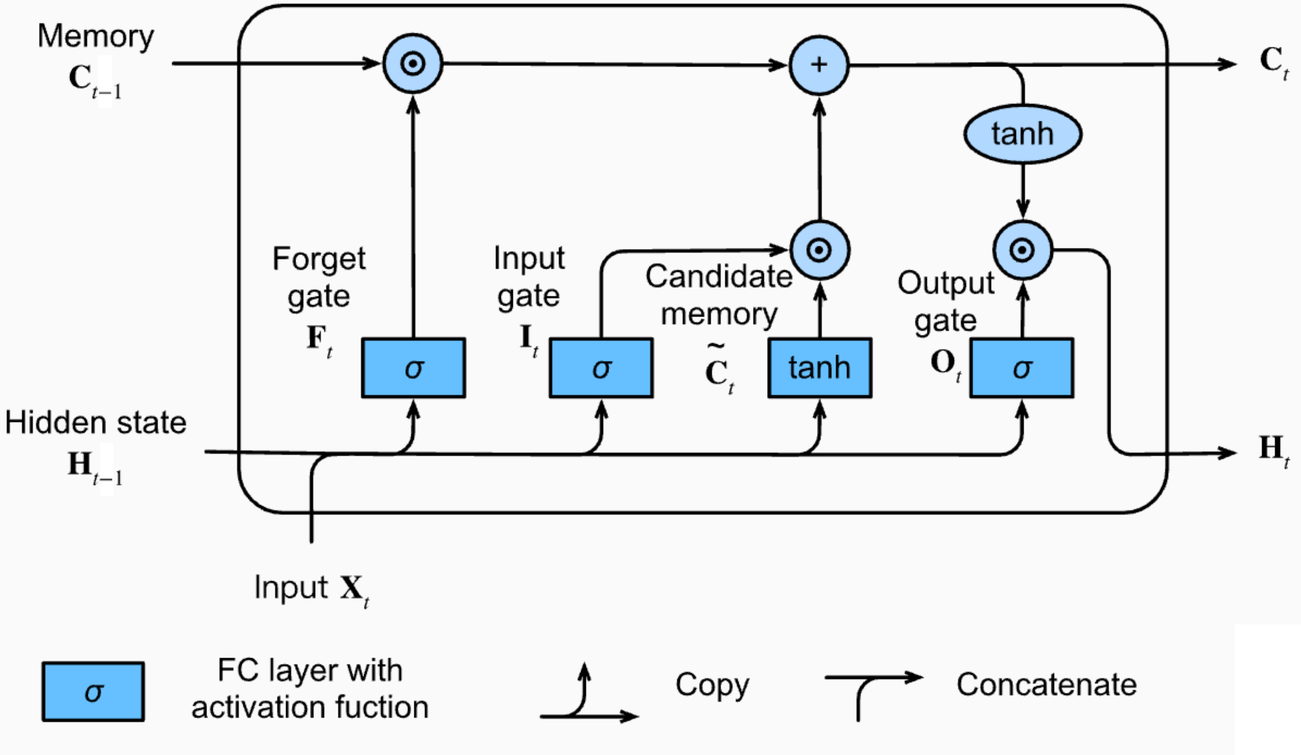
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t.$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \mathbf{b}_c).$$

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \mathbf{b}_i),$$

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \mathbf{b}_f),$$

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \mathbf{b}_o),$$



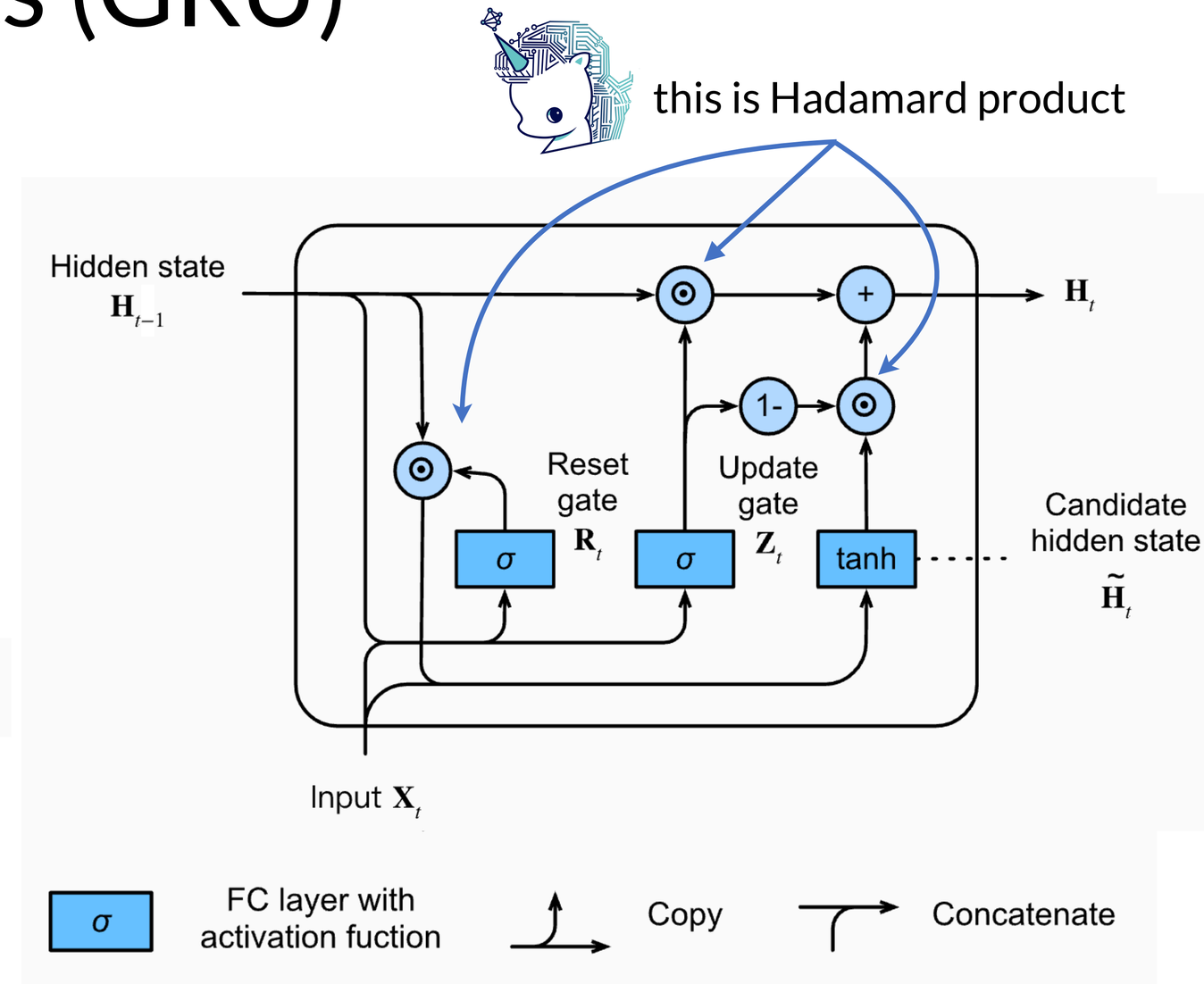
Gated Recurrent Units (GRU)

- Update Gates in Action
  - update variable allows us to control ***new state*** is just a copy of the old state

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t.$$

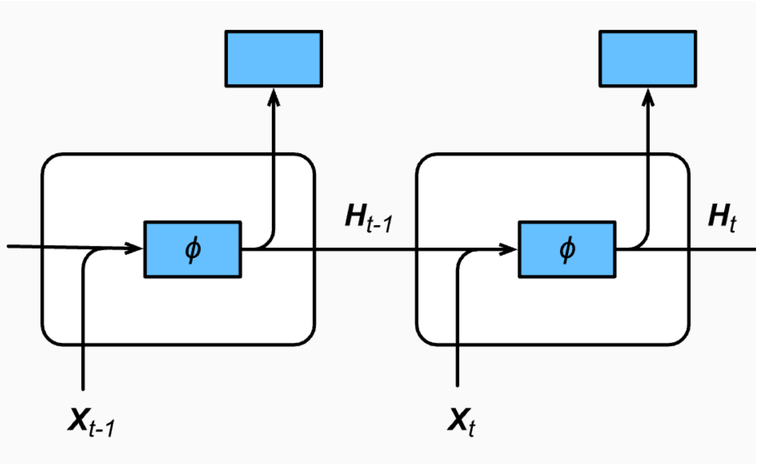
$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h).$$

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r),$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z).$$



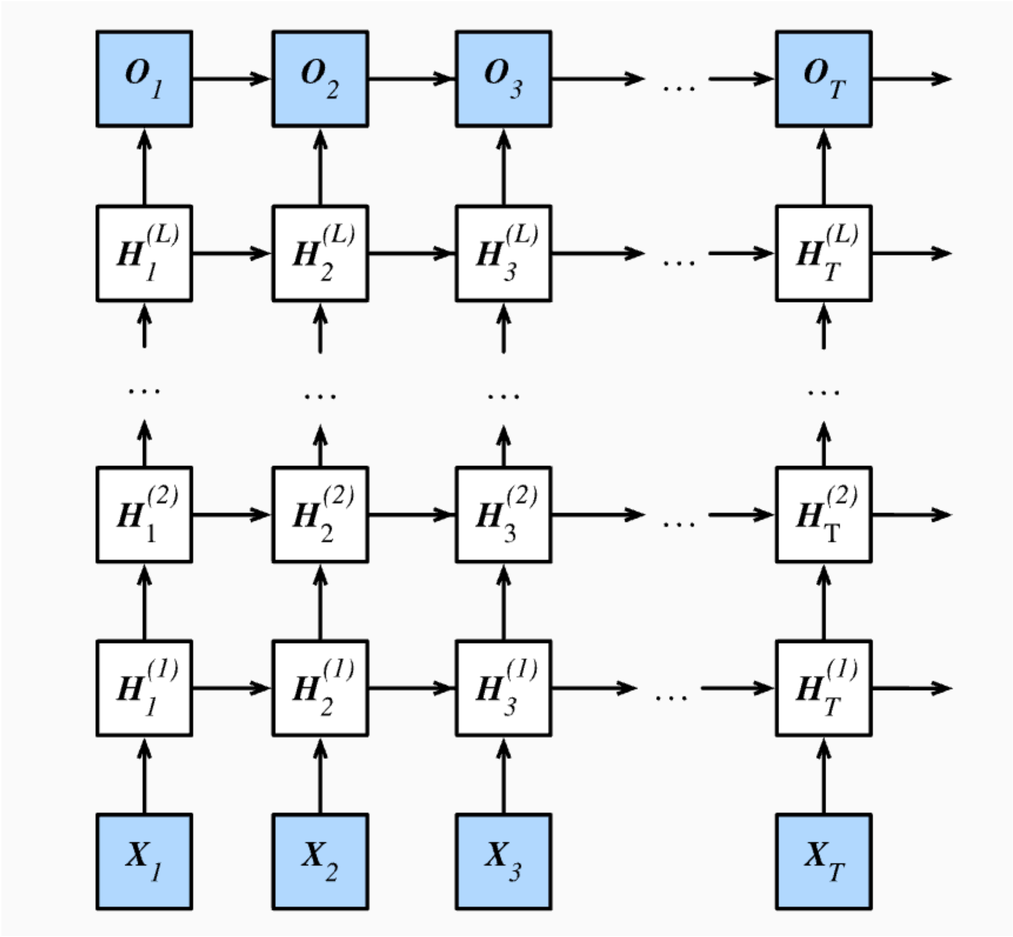
Deep RNN

- We can add more layers in RNN
  - PyTorch provides simple option to implement multiple layers



$$\mathbf{H}_t^{(1)} = f_1\left(\mathbf{X}_t, \mathbf{H}_{t-1}^{(1)}\right),$$
$$\mathbf{H}_t^{(l)} = f_l\left(\mathbf{H}_t^{(l-1)}, \mathbf{H}_{t-1}^{(l)}\right).$$

GRU, LSTM are also possible



An architecture of deep RNN

Bidirectional RNN

- We often need sequential models which can deal with information of the subsequent data
- It is possible to have a mechanism of ***bidirectional*** path in RNN

entity recognition should care about context

1. I am _____
2. I am _____ very hungry.
3. I am _____ very hungry, I could eat half a pig.

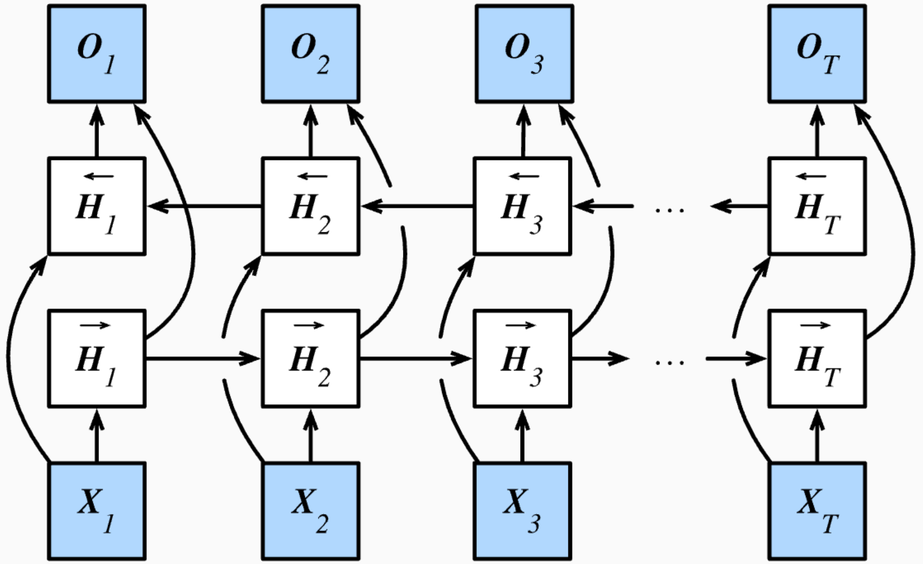


Fig. 9.4.2 Architecture of a bidirectional recurrent neural network.

Bidirectional RNN

- We often need sequential models which can deal with information of the subsequent data
- It is possible to have a mechanism of ***bidirectional*** path in RNN

$$\overrightarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}),$$
$$\overleftarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}).$$

we use concatenation for combining both hidden variables

```
1. I am _____
2. I am _____ very hungry.
3. I am _____ very hungry, I could eat half a pig.
```

concat

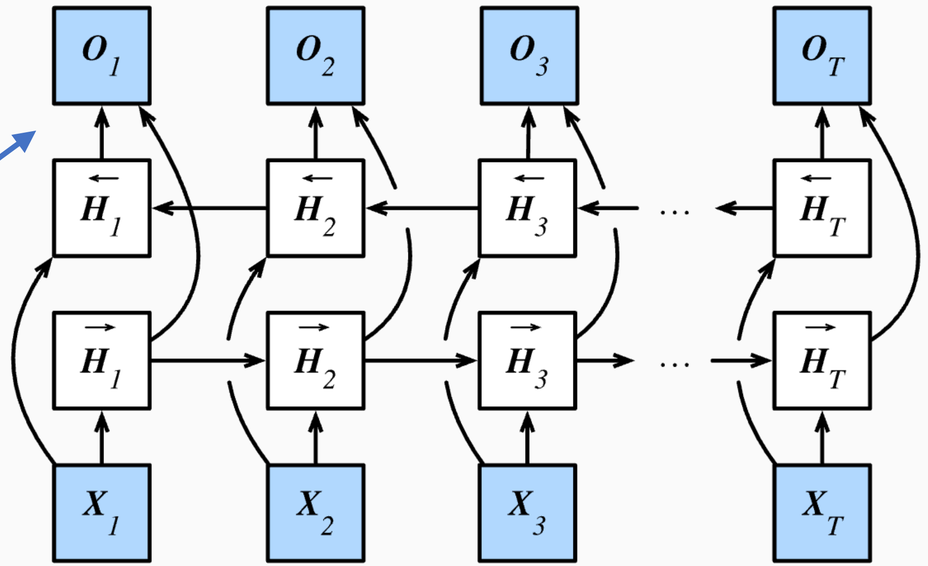
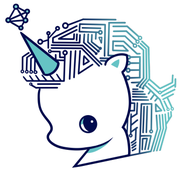


Fig. 9.4.2 Architecture of a bidirectional recurrent neural network.

When we use bidirectional RNN?

- We use bidirectional RNN (or LSTM) when we need information from both ends of the sequence
  - we use information from both future and past observation
  - if we apply bidirectional RNN naively, we would get bad performance (+ exceedingly ***slow*** training)
- In practice, bidirectional layers are used carefully
  - etc) filling in missing words, annotating tokens, translation



these are restricted in NLP

Q&A