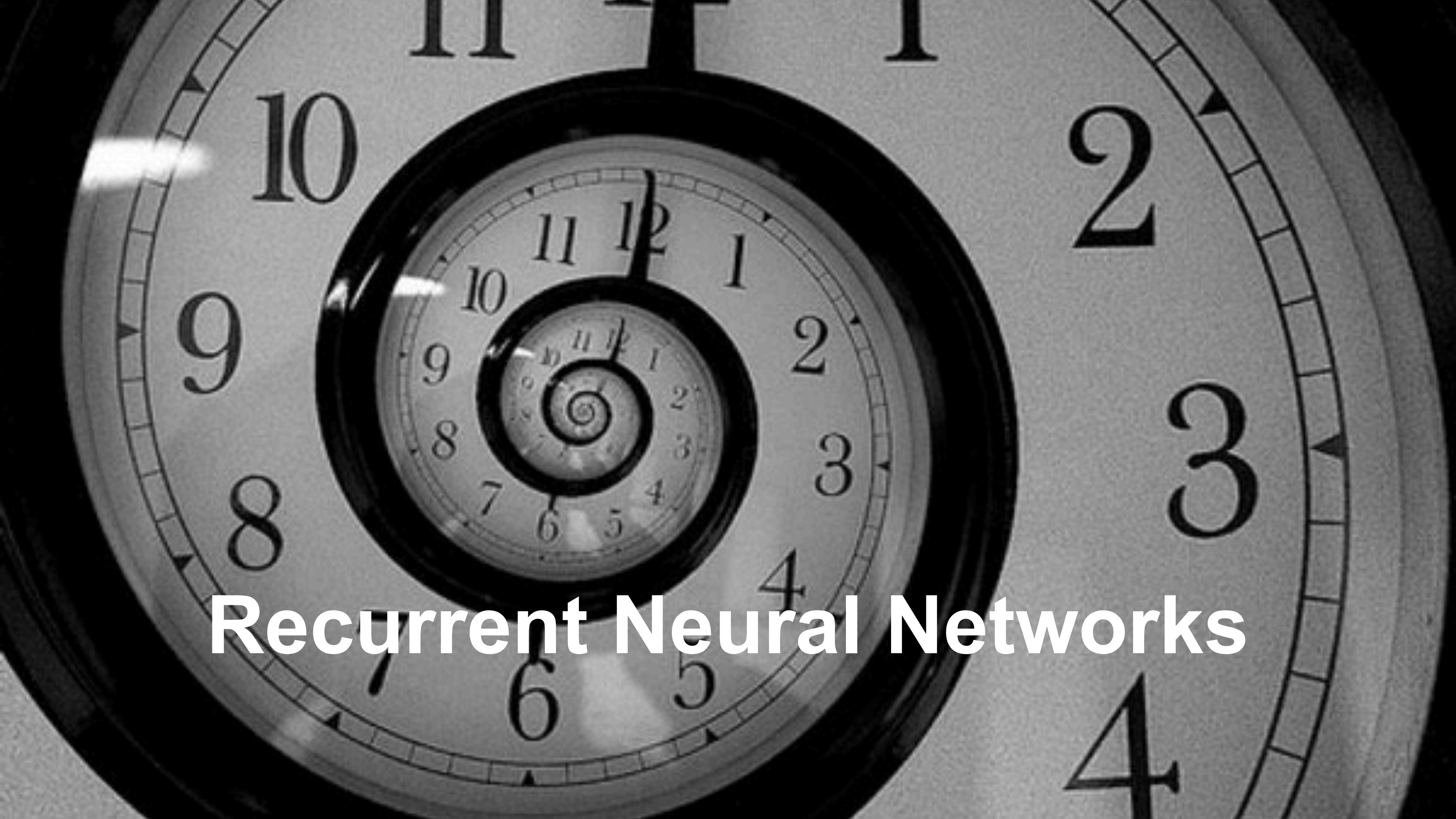
Introduction to Deep Learning

19. Recurrent Neural Networks

STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

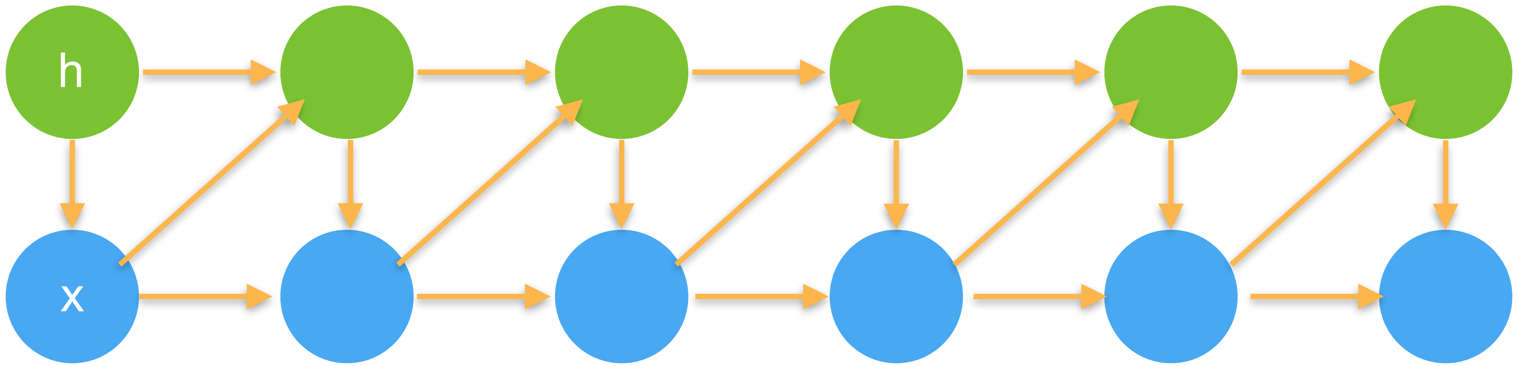


Recurrent Neural Networks

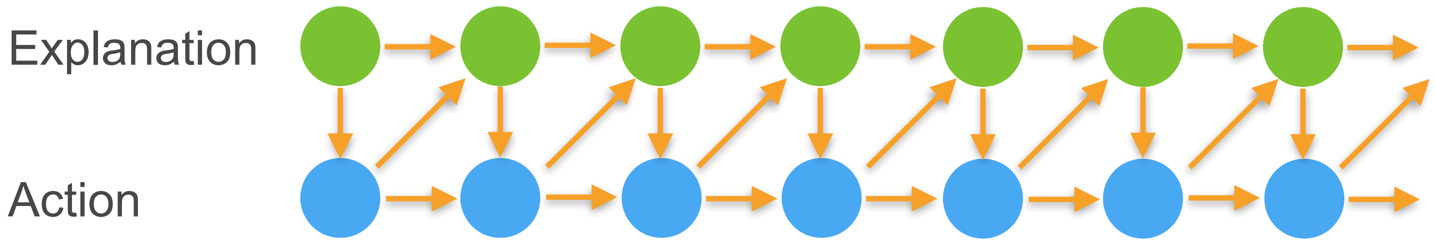
Latent Variable Autoregressive Models

Latent state summarizes all the relevant information about the past. So we get $h_t = f(x_1, \ldots x_{t-1}) = f(h_{t-1}, x_{t-1})$

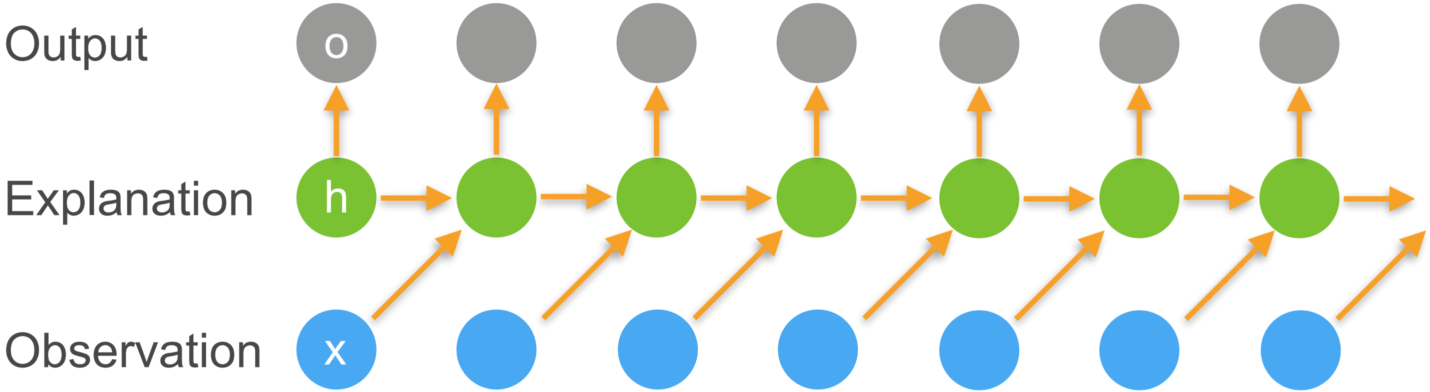
$p(h_t \mid h_{t-1}, x_{t-1})$ and $p(x_t \mid h_t, x_{t-1})$



Recurrent Neural Networks (with hidden state)



Recurrent Neural Networks (with hidden state)



- Hidden State update

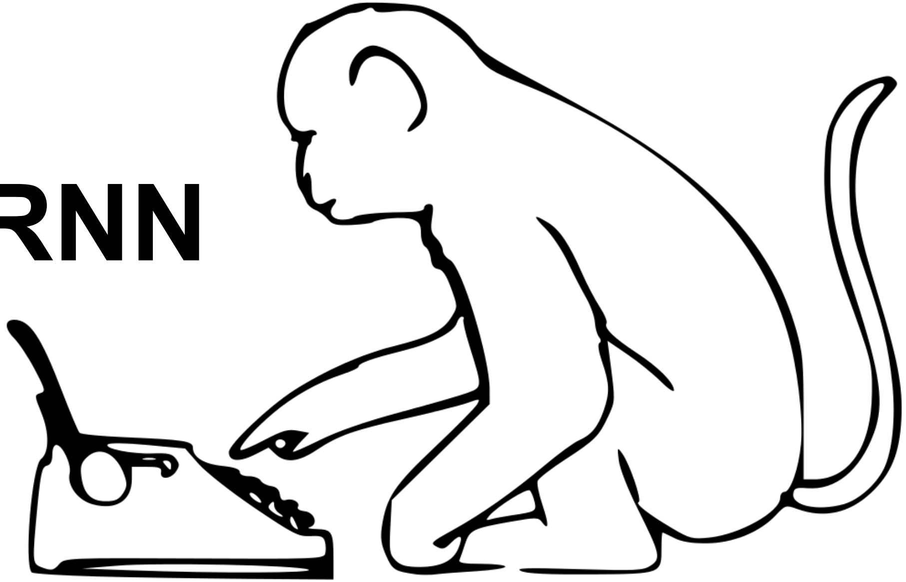
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Code …

Implementing an RNN Language Model



Input Encoding

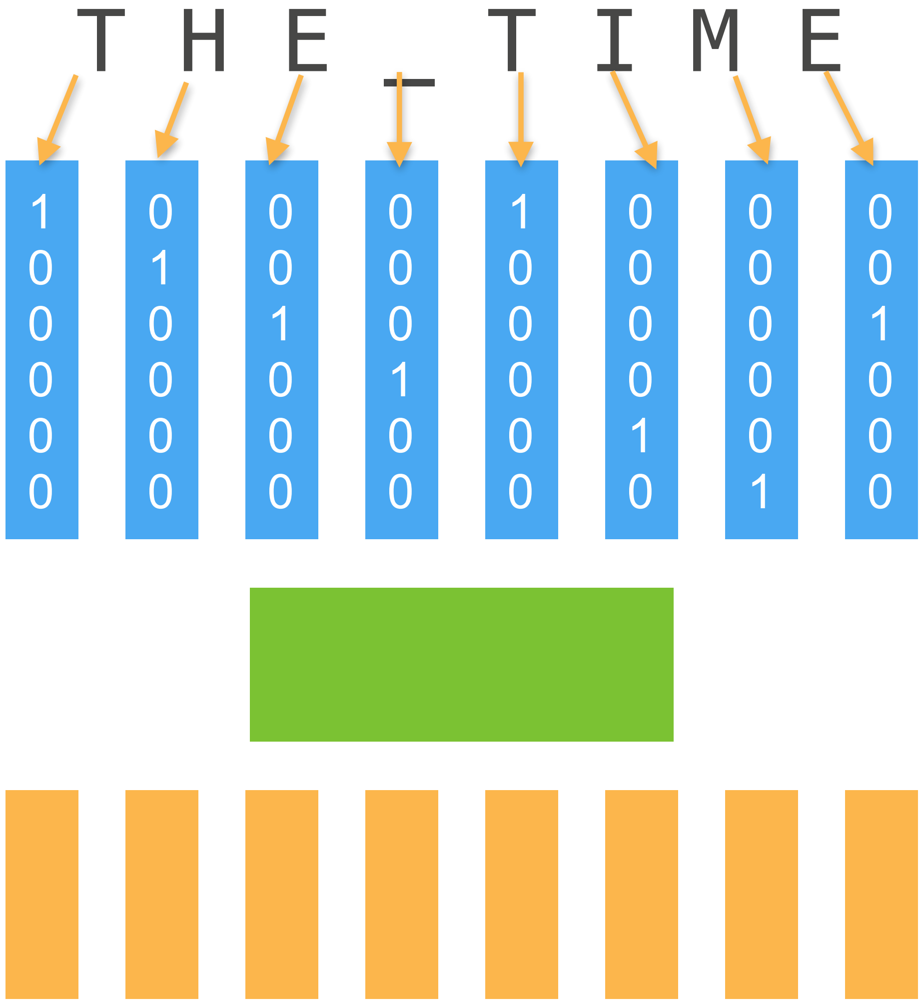
- Need to map input tokens to vectors
  - Pick granularity (words, characters, subwords)
  - Map to indicator vectors

```
nd.one_hot(nd.array([0, 2]), vocab_size)
```

```
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
<NDArray 2x43 @cpu(0)>
```

- Multiply by embedding matrix W

Input Encoding



Canonical Vectors $\mathbf{v}$

Embedding Matrix $\mathbf{W}$

Embedded Vectors $\mathbf{v}'$

RNN with hidden state mechanics

- Input
  vector sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$
- Hidden States
  vector sequence $\mathbf{h}_1, \ldots, \mathbf{h}_T$ where $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$
- Output
  vector sequence $\mathbf{o}_1, \ldots, \mathbf{o}_T$ where $\mathbf{o}_t = g(\mathbf{h}_t)$

Read sequence to generate hidden states, then start generating outputs. Often outputs (symbols) are used as input for next hidden state (and thus output).

Output Decoding

Output Vectors $\mathbf{o}$

Decoding Matrix $\mathbf{W}'$

$$p(y \mid \mathbf{o}) \propto \exp\left(\mathbf{v}_y^\top \mathbf{o}\right) = \exp(\mathbf{o}[y])$$

One-hot decoding

Gradients

- Long chain of dependencies for backprop
  - Need to keep a lot of intermediate values in memory
  - Butterfly effect style dependencies
  - Gradients can vanish or diverge (more on this later)
- Clipping to prevent divergence

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

rescales to gradient of size at most θ

Perplexity

- Typically measure accuracy with log-likelihood
  - This makes outputs of different length incomparable (e.g. bad model on short output has higher likelihood than excellent model on very long output)
  - Normalize log-likelihood to sequence length

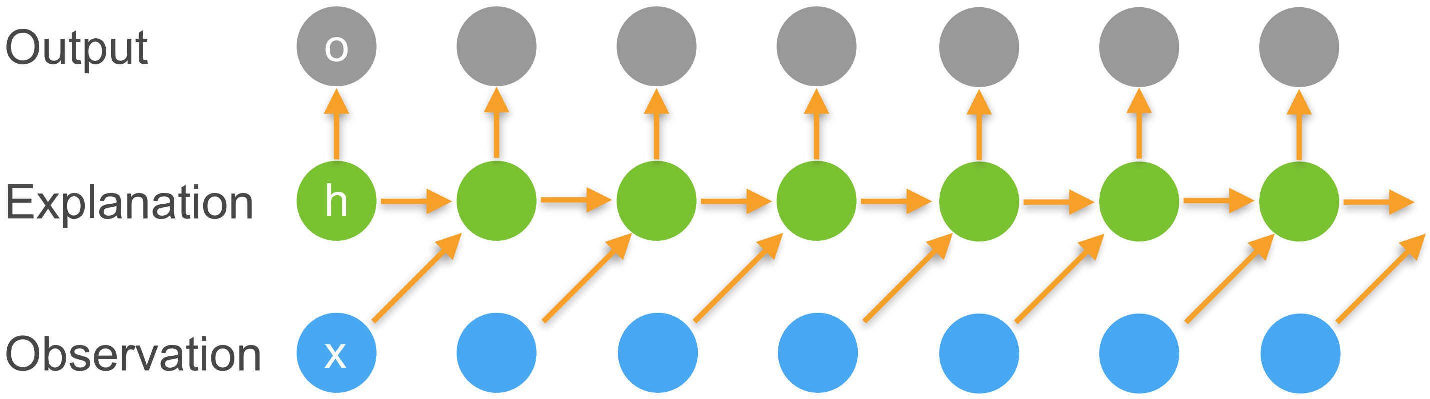
$$-\sum_{t=1}^{T} \log p(y_t | \text{model}) \text{ vs. } \pi := -\frac{1}{T}\sum_{t=1}^{T} \log p(y_t | \text{model})$$

- Perplexity is exponentiated version $\exp(\pi)$ (effectively number of possible choices on average)

Code …

Truncated Backprop Through Time

Recurrent Neural Networks (with hidden state)



- Hidden State update

$$h_t = f(h_{t-1}, x_{t-1}, w)$$

- Observation update

$$o_t = g(h_t, w)$$

Objective function

- RNN generates output which needs to be compared to target labels

$$L(x, y, w) = \sum_{t=1}^{T} l(y_t, o_t)$$

- Gradient

$$\partial_w L = \sum_{t=1}^{T} \partial_w l(y_t, o_t)$$

$$= \sum_{t=1}^{T} \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

Latent State Gradient $\partial_w h_t$

- Objective Function

$$\partial_w L = \sum_{t=1}^{T} \partial_w l(y_t, o_t) = \sum_{t=1}^{T} \partial_{o_t} l(y_t, o_t) \left[\partial_w g(h_t, w) + \partial_{h_t} g(h_t, w) \partial_w h_t \right]$$

- Gradient Recursion

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

$$= \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

Latent State Gradient $\partial_w h_t$

- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$

Too Many Terms

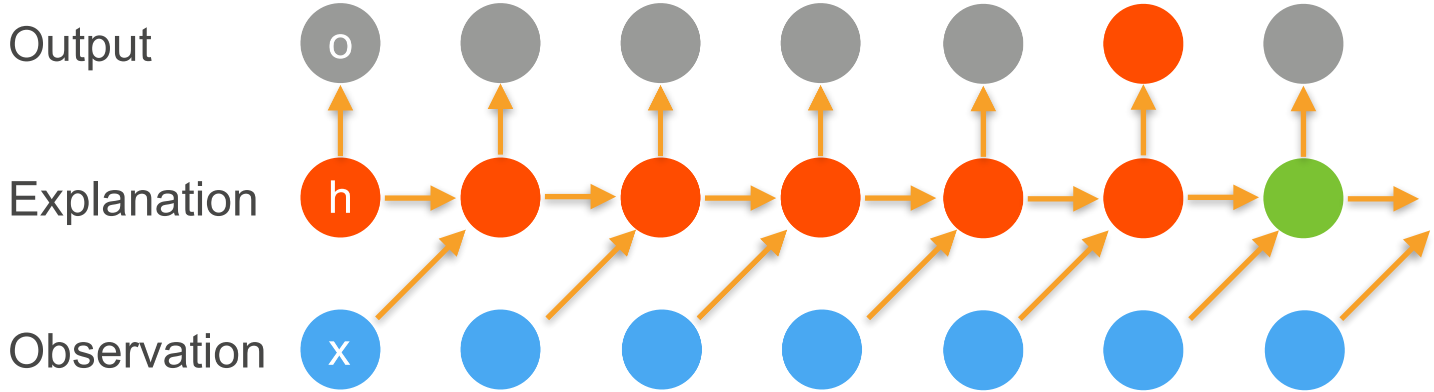
Unstable (divergence)

expensive

Latent State Gradient $\partial_w h_t$

- Gradient Recursion

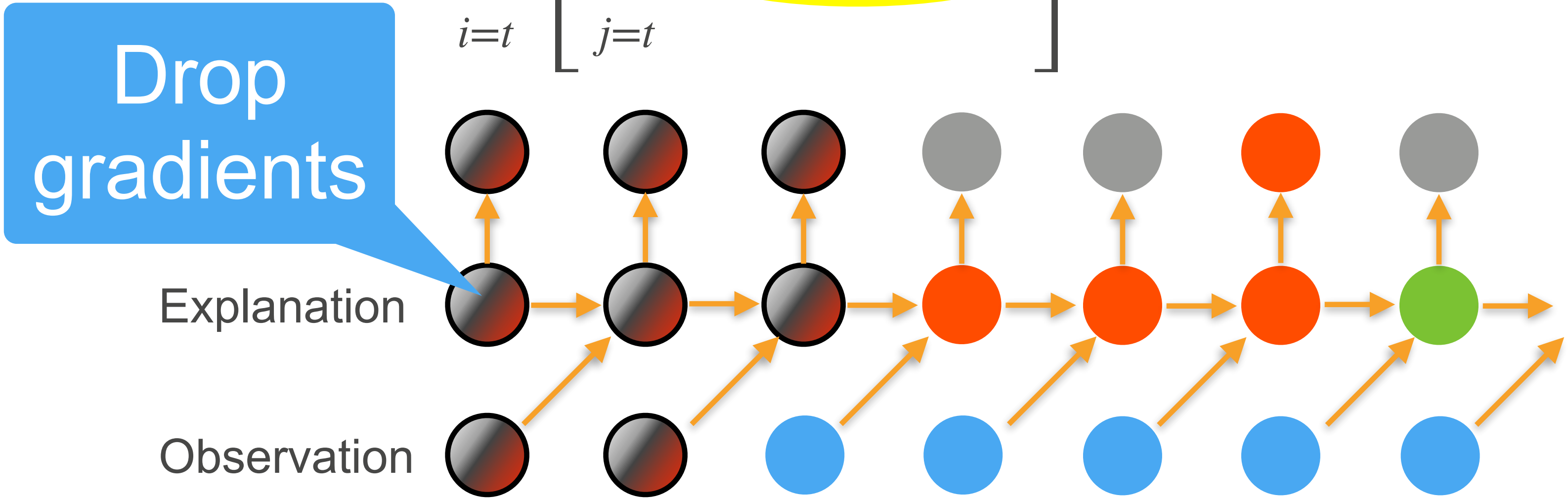
$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



Latent State Gradient $\partial_w h_t$

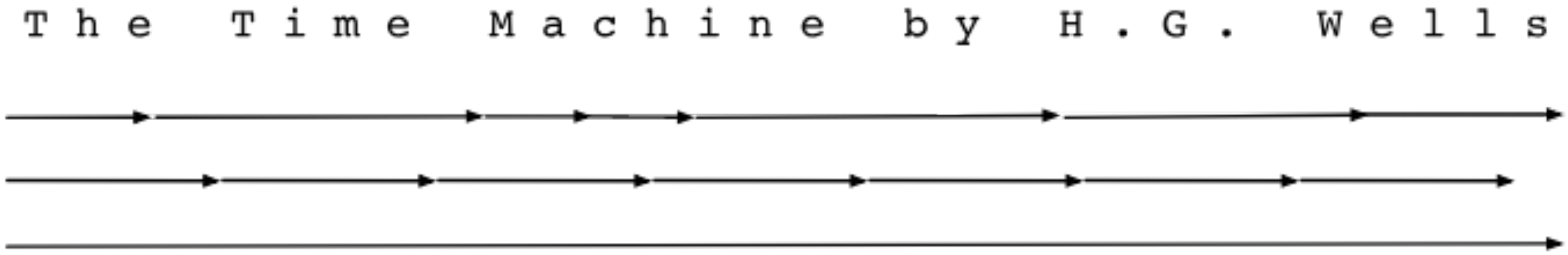
- Gradient Recursion

$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



Truncated BPTT

- Don’t truncate (naive strategy, costly and divergent)
- Truncate at fixed intervals
  (standard approach, is approximation but works well)
- Variable length (Tallec and Olivier, 2015)
  (is exact after reweighting, doesn’t work better in practice)

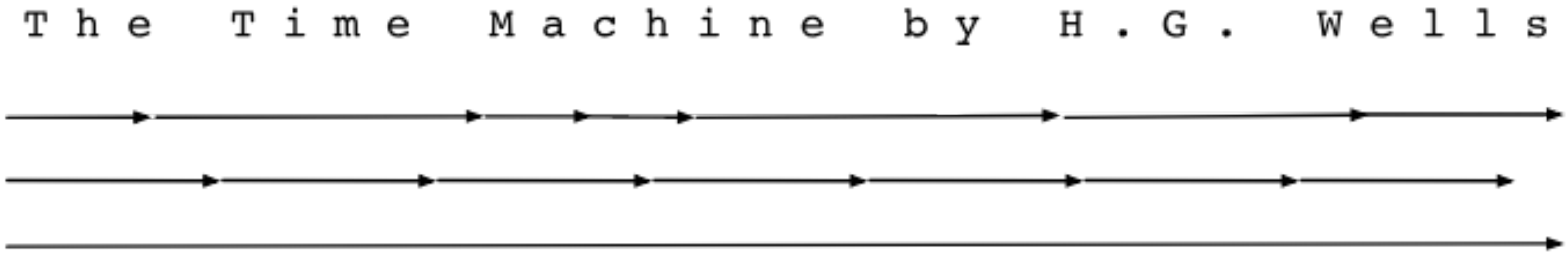


Truncated BPTT

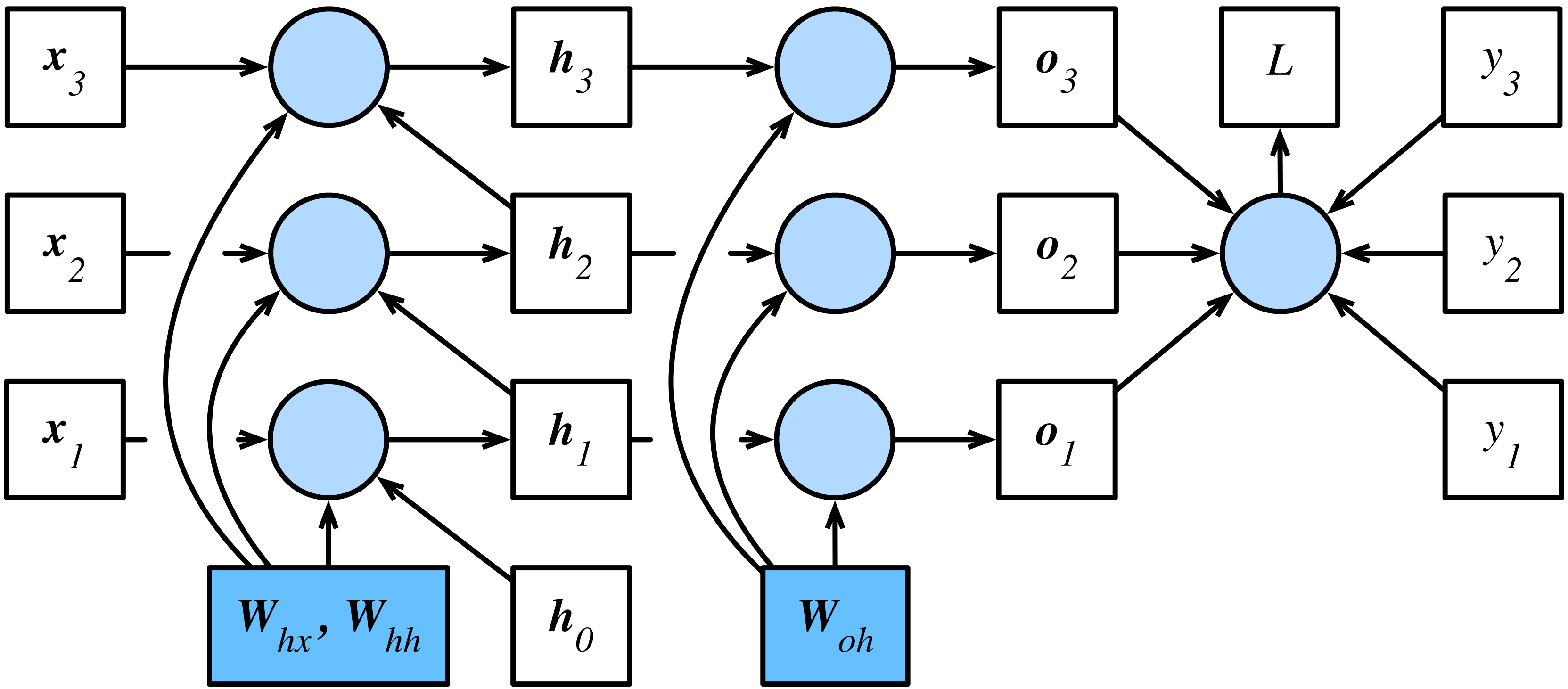
- Random variable instead of simple truncation

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

- Variable length (Tallec and Olivier, 2015)
  (is exact after reweighting, doesn't work better in practice)



Computational Graph



Example in detail

Toy Model

- Linear RNN

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh}\mathbf{h}_t$$

- Output gradient

$$\partial_{\mathbf{W}_{oh}} L = \sum_{t=1}^{T} \operatorname{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{h}_t\right)$$

- State update gradient

$$\partial_{\mathbf{W}_{hh}} L = \sum_{t=1}^{T} \operatorname{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hh}} \mathbf{h}_t\right)$$

$$\partial_{\mathbf{W}_{hx}} L = \sum_{t=1}^{T} \operatorname{prod}\left(\partial_{\mathbf{o}_t} l(\mathbf{o}_t, y_t), \mathbf{W}_{oh}, \partial_{\mathbf{W}_{hx}} \mathbf{h}_t\right)$$

Gradients … continued

- Linear RNN

$$\mathbf{h}_t = \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} \text{ and } \mathbf{o}_t = \mathbf{W}_{oh}\mathbf{h}_t$$

- Recursive update

$$\partial_{\mathbf{h}_t}\mathbf{h}_{t+1} = \mathbf{W}_{hh}^\top \text{ and thus } \partial_{\mathbf{h}_t}\mathbf{h}_T = \left(\mathbf{W}_{hh}^\top\right)^{T-t}$$

- Full recursion

Drop gradients

$$\partial_{\mathbf{W}_{hh}}\mathbf{h}_t = \sum_{j=1}^{t}\left(\mathbf{W}_{hh}^\top\right)^{t-j}\mathbf{h}_j$$

$$\partial_{\mathbf{W}_{hx}}\mathbf{h}_t = \sum_{j=1}^{t}\left(\mathbf{W}_{hh}^\top\right)^{t-j}\mathbf{x}_j\,.$$

Truncation in practice

- Compute forward pass **across** truncation boundaries
- Backprop only until truncation boundary (typically mini batch boundary, too)
- In code

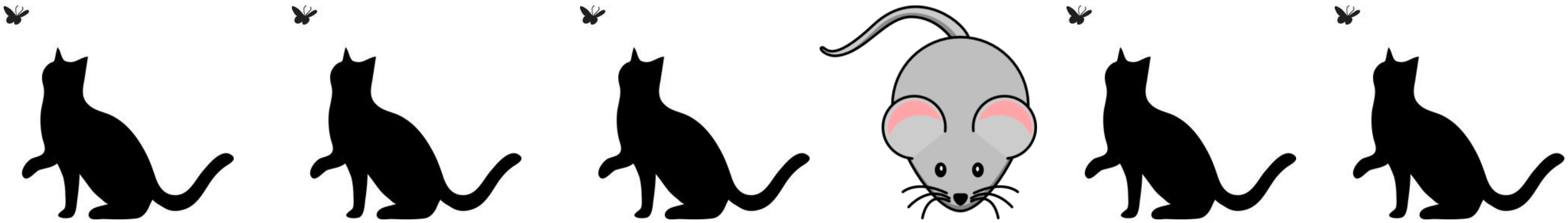
```
for s in state:
    s.detach()
```

- Good reason for why sequential sampling is much more accurate than random - state is carried through.

Gated Recurrent Unit (GRU)

Paying attention to a sequence

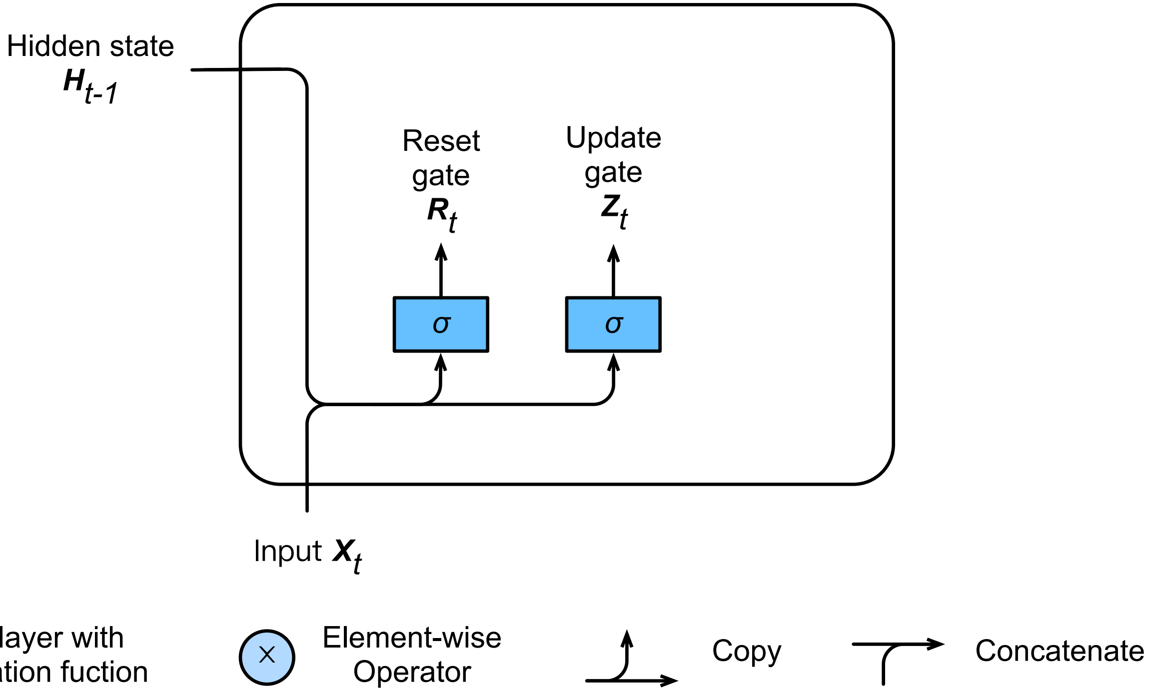
- Not all observations are equally relevant



- Only remember the relevant ones
  - Need mechanism to **pay attention (update gate)**
  - Need mechanism to **forget (reset gate)**

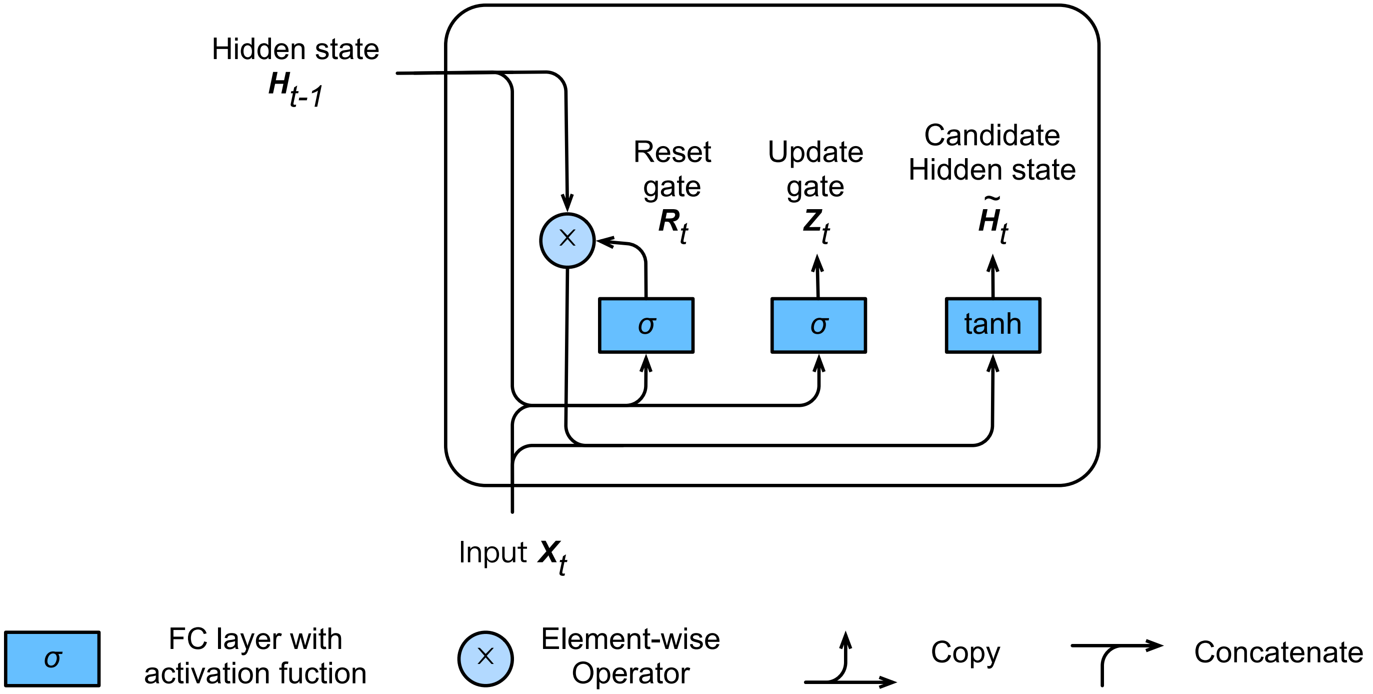
Gating

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$



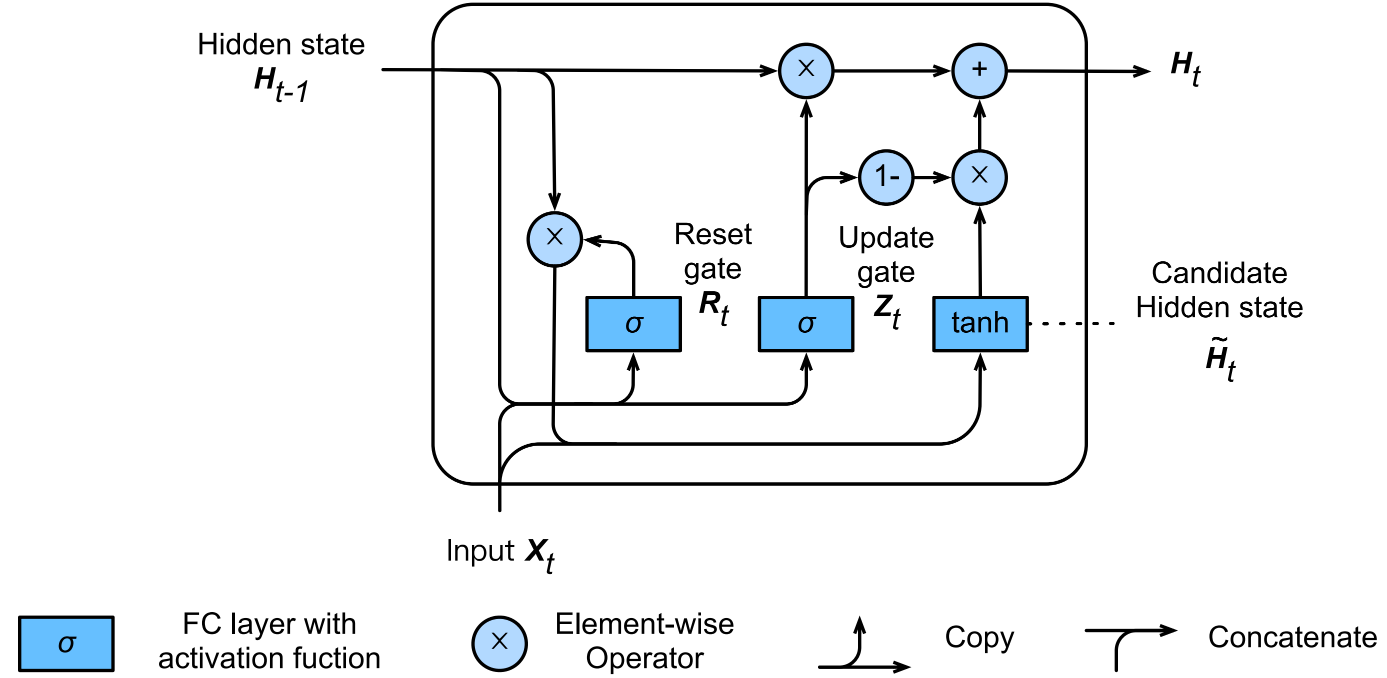
Candidate Hidden State

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



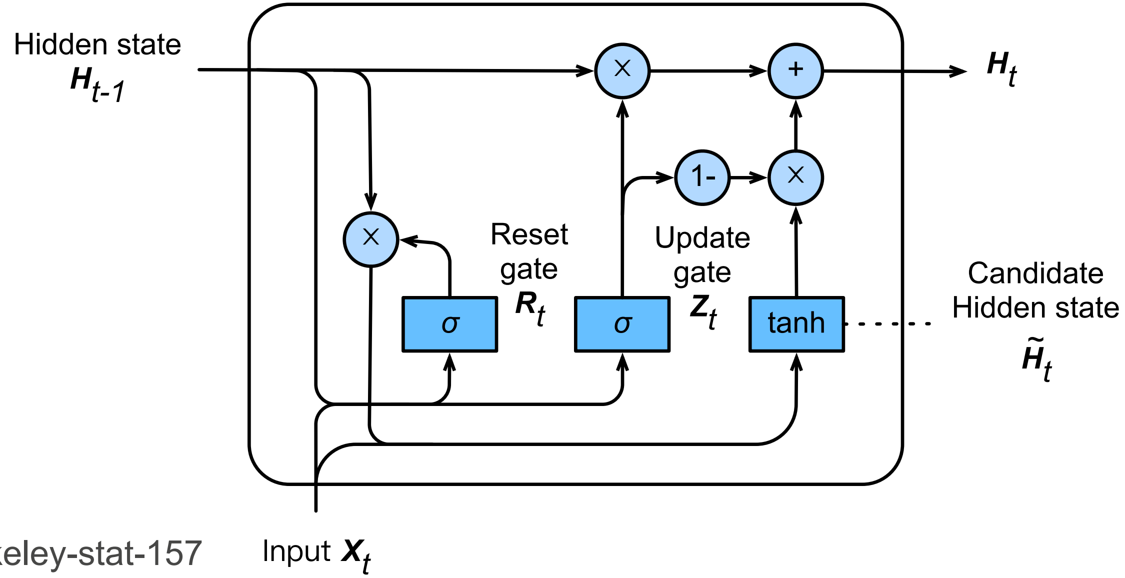
Hidden State

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$



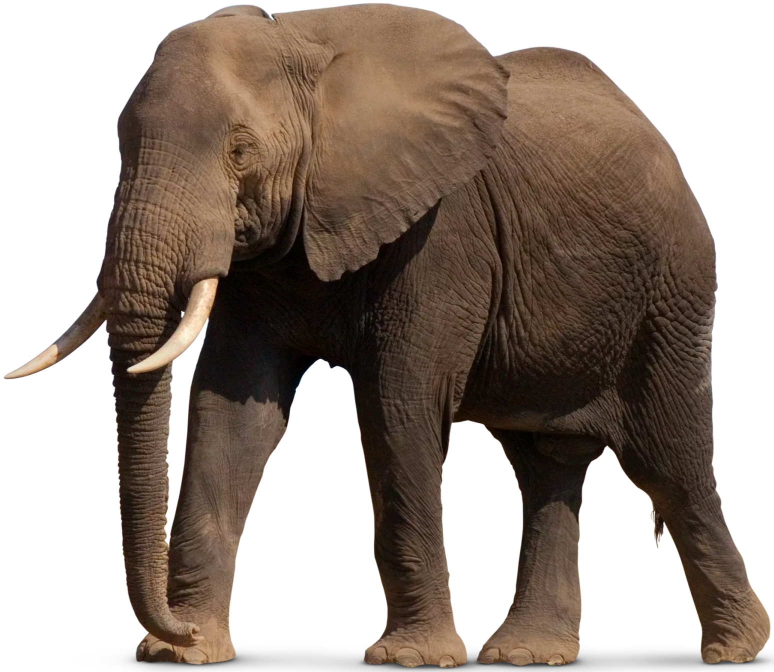
Summary

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r),$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$
$$\tilde{H}_t = \tanh(X_t W_{xh} + \left(R_t \odot H_{t-1}\right) W_{hh} + b_h)$$
$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$$

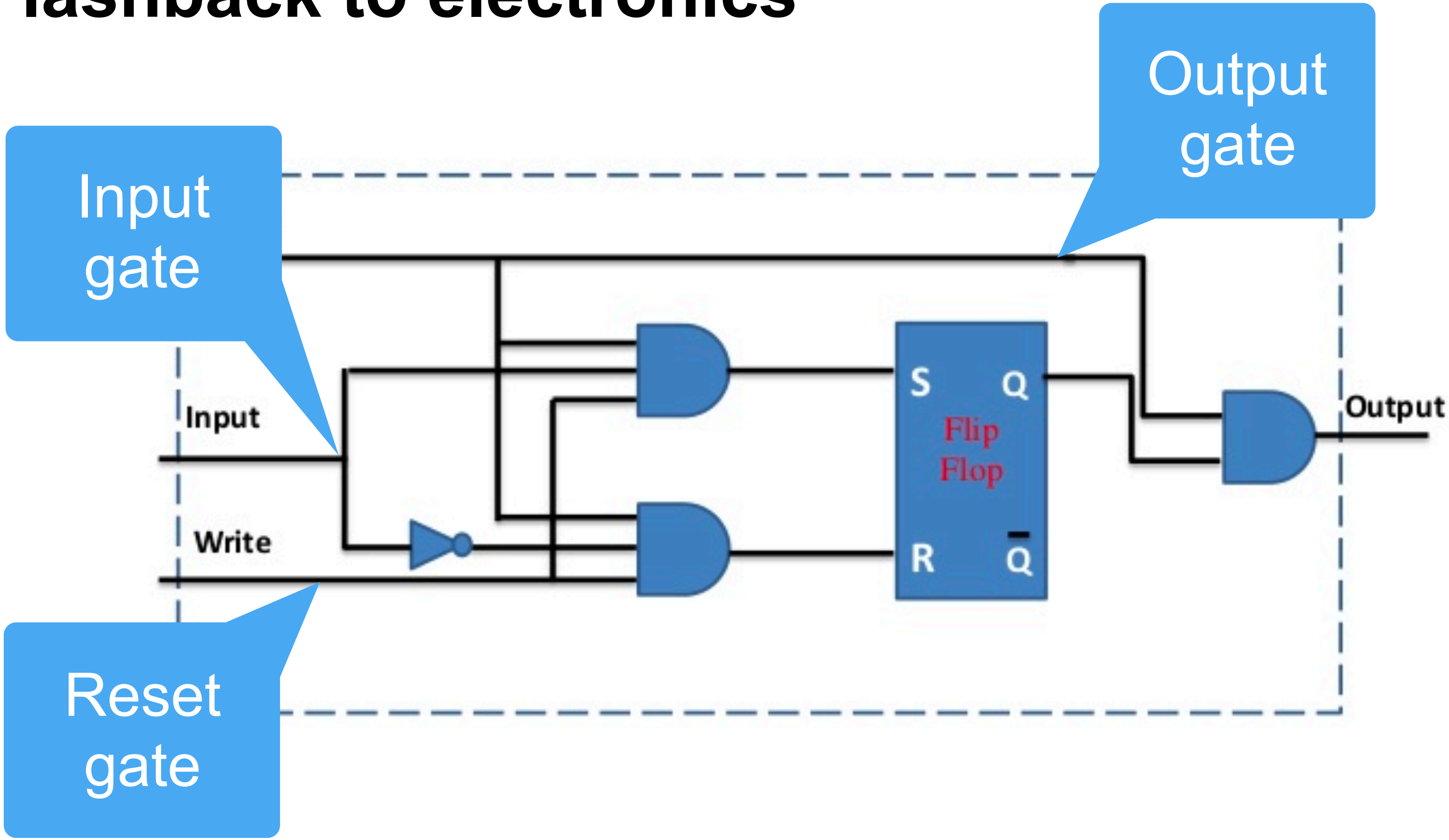


Code …

Long Short Term Memory



Flashback to electronics



Long Short Term Memory

- **Forget gate**
  Shrink values towards zero
- **Input gate**
  Decide whether we should ignore the input data
- **Output gate**
  Decide whether the hidden state is used for the output generated by the LSTM

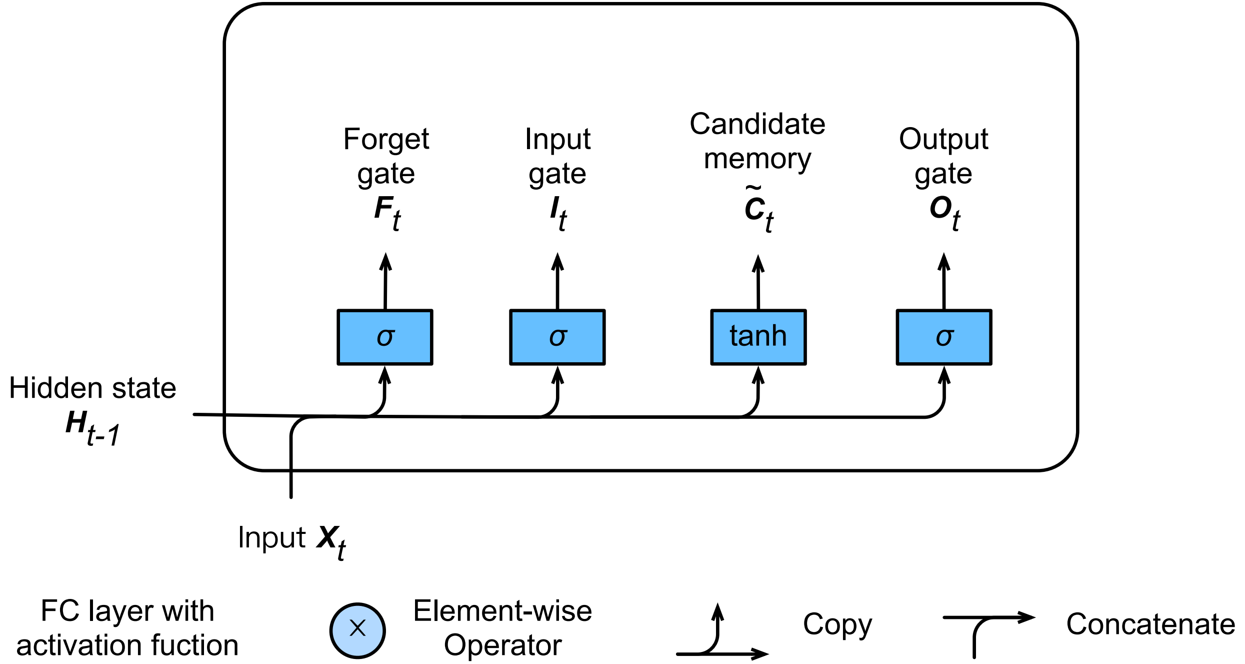
- **Hidden state** and **Memory cell**

Gates

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

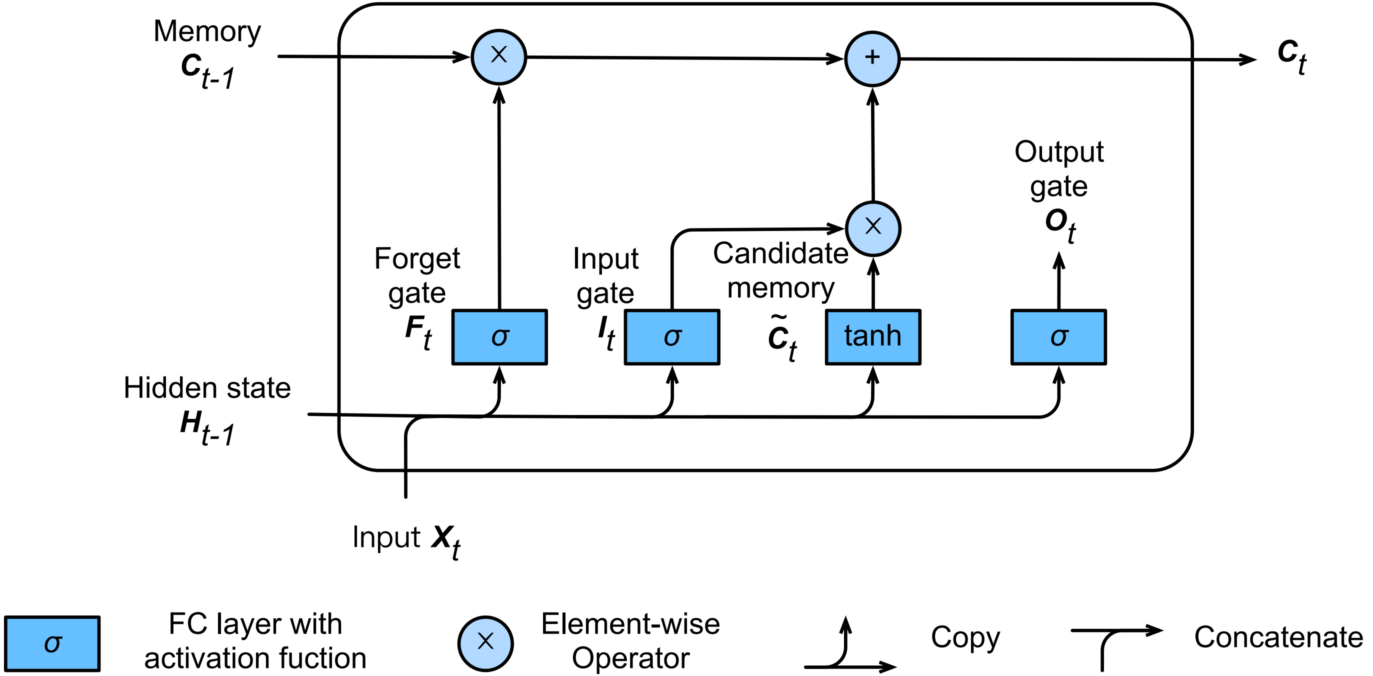
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$



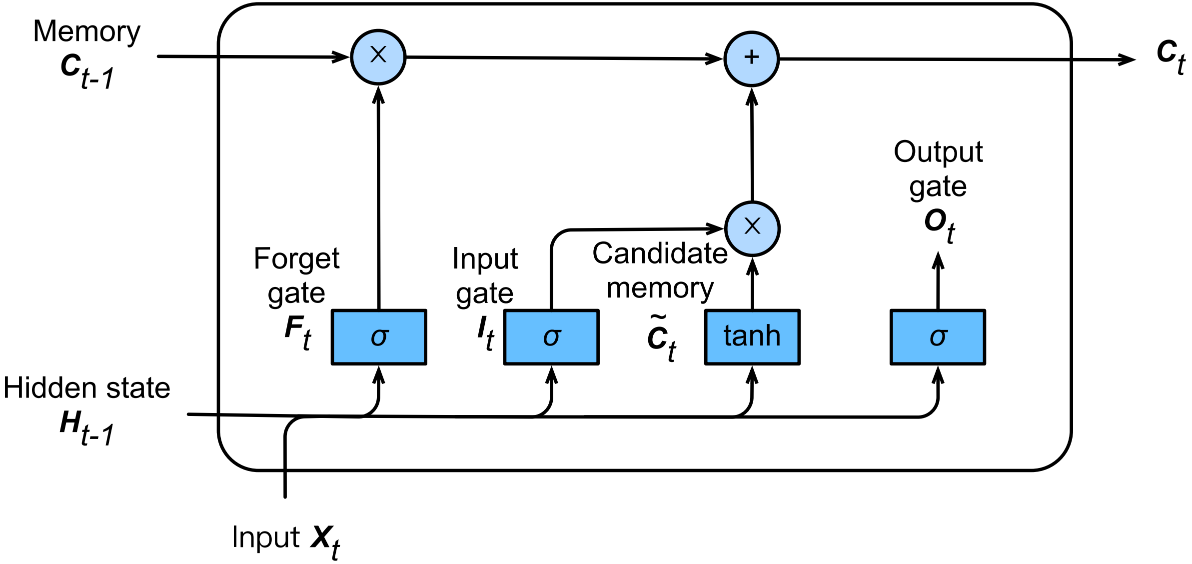
Candidate Memory Cell

$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$



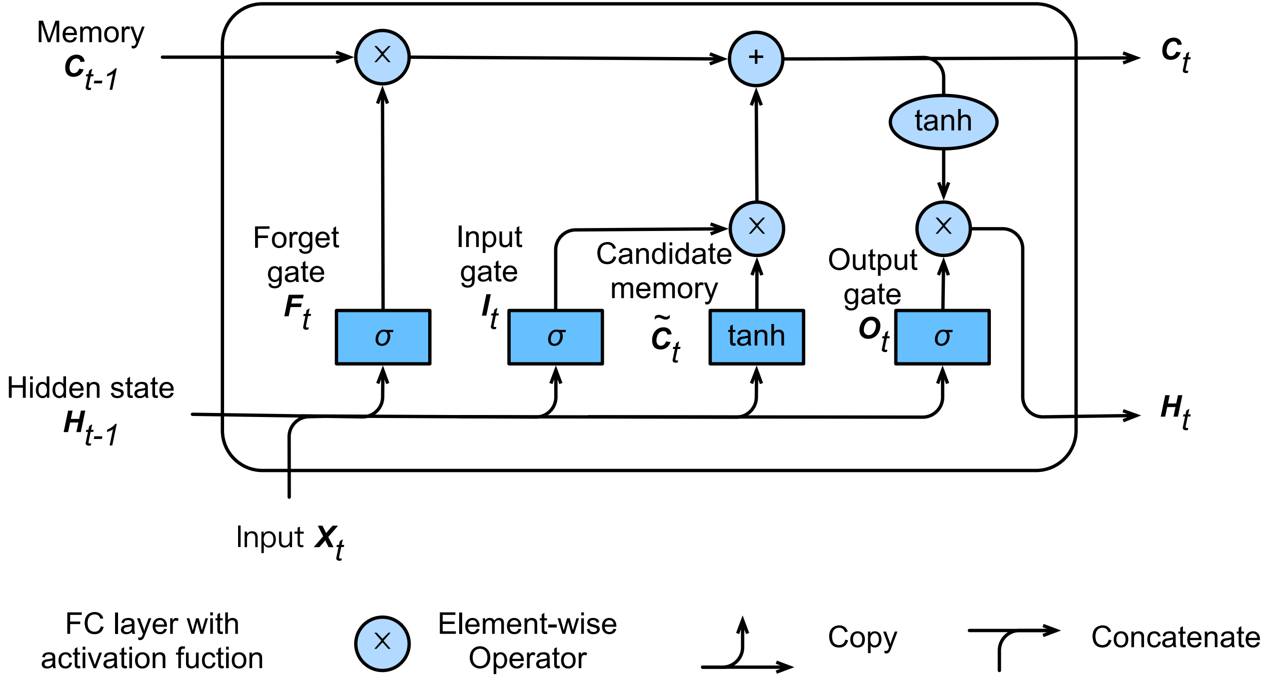
Memory Cell

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

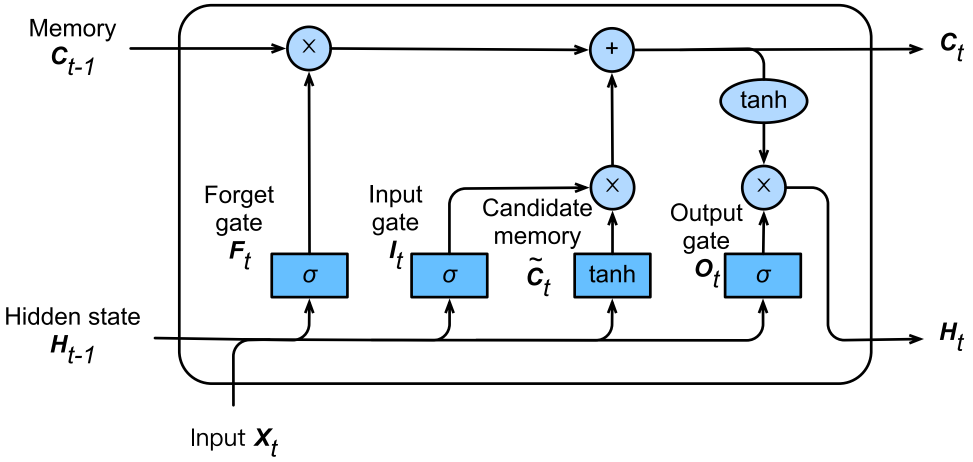


Hidden State / Output

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



Hidden State / Output



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$
$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$$
$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$
$$\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$$
$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$
$$H_t = O_t \odot \tanh(C_t)$$

Code …