Introduction to Deep Learning

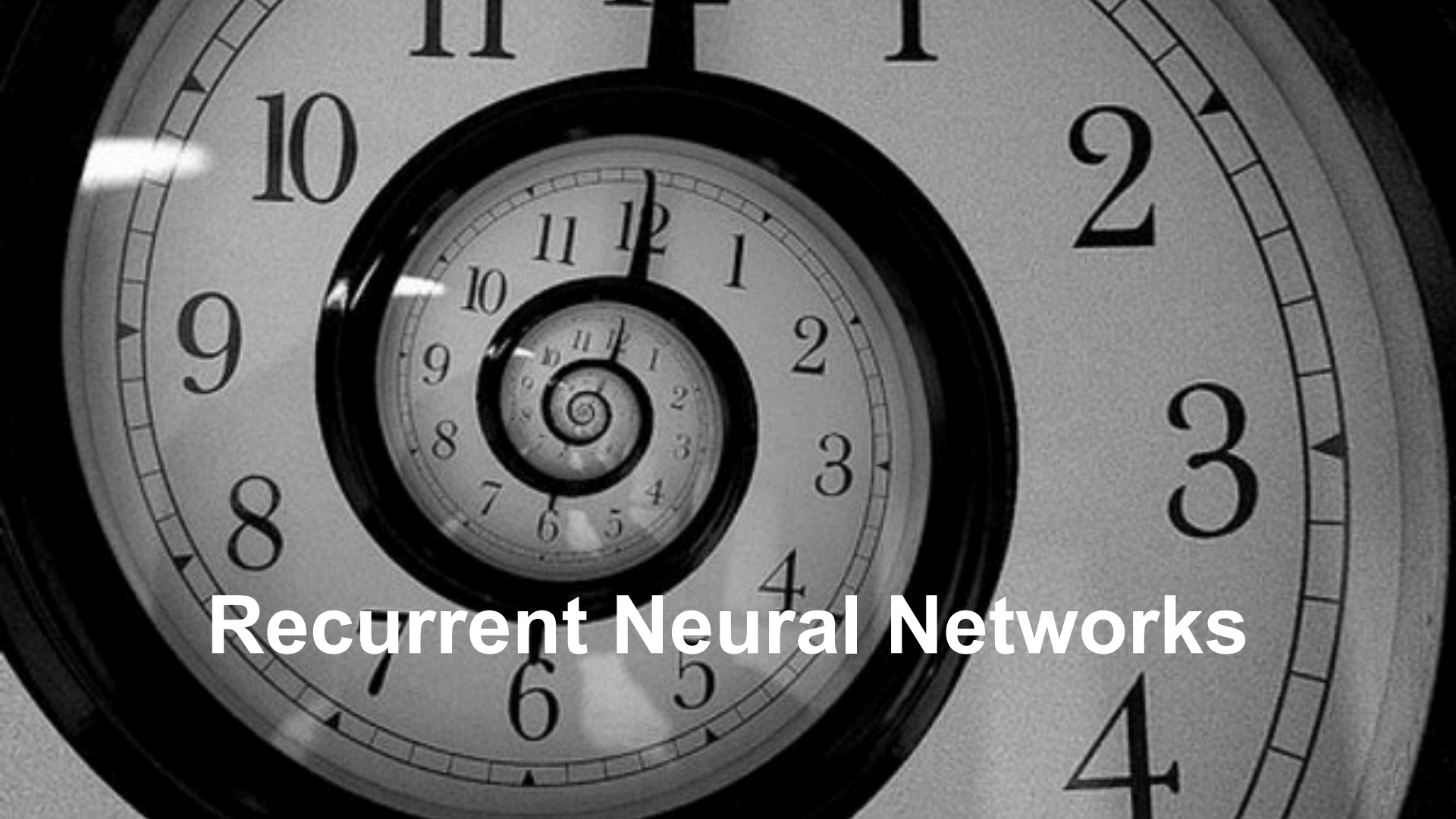
19. Recurrent Neural Networks

STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

aws

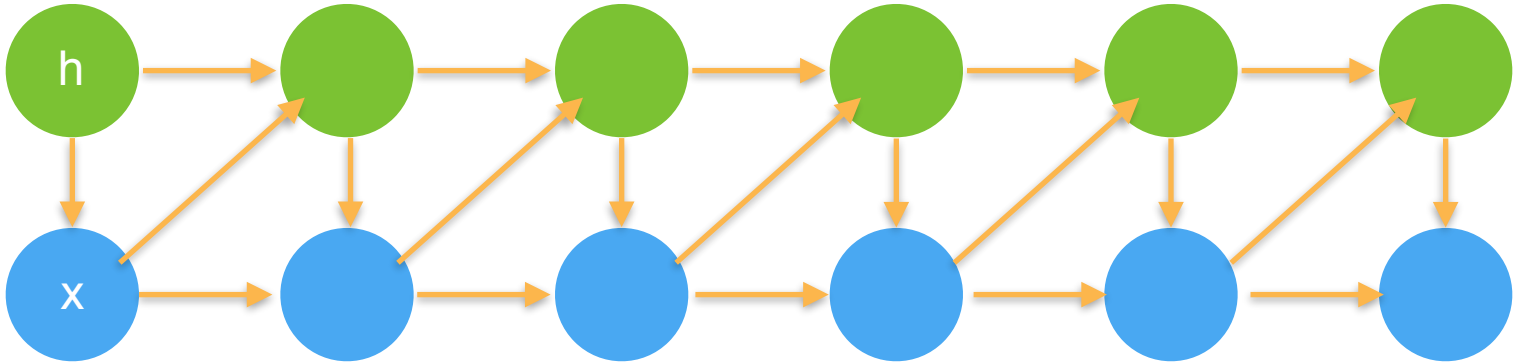


Recurrent Neural Networks

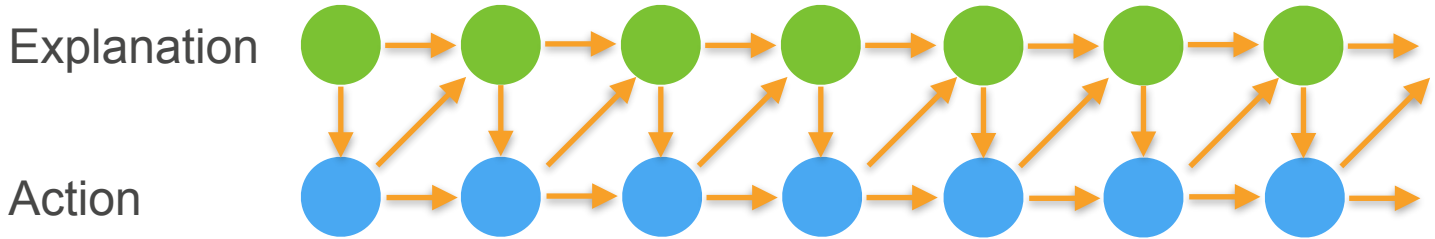
Latent Variable Autoregressive Models

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

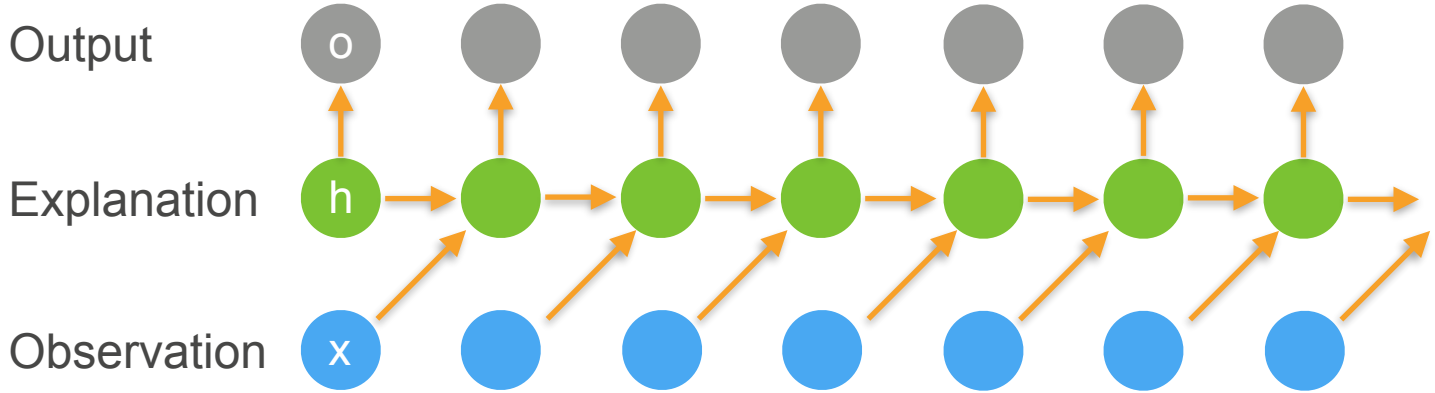
$p(h_t \mid h_{t-1}, x_{t-1})$ and $p(x_t \mid h_t, x_{t-1})$



Recurrent Neural Networks (with hidden state)



Recurrent Neural Networks (with hidden state)



- Hidden State update

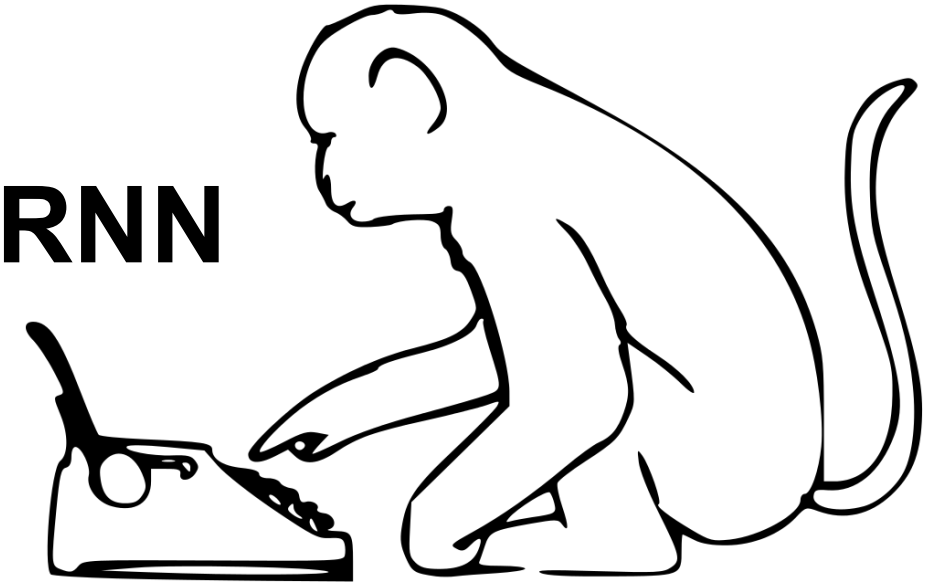
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

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

Code …

Implementing an RNN Language Model



Input Encoding

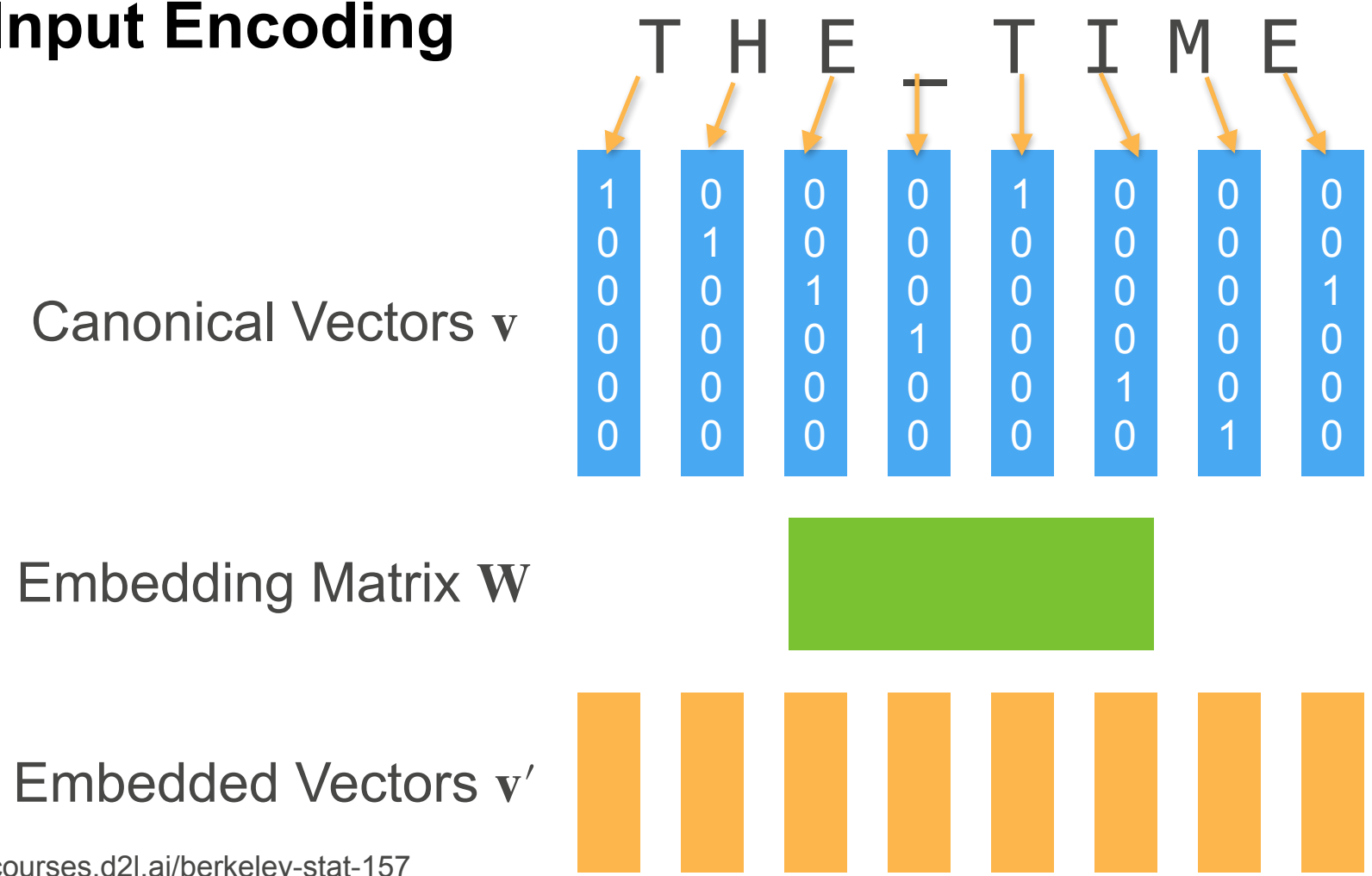
- Need to map input tokens to vectors
  - Pick granularity (words, characters, subwords)
  - Map to indicator vectors

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

- Multiply by embedding matrix W

Input Encoding



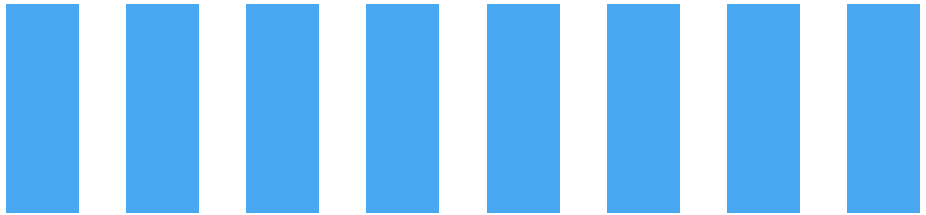
RNN with hidden state mechanics

- 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

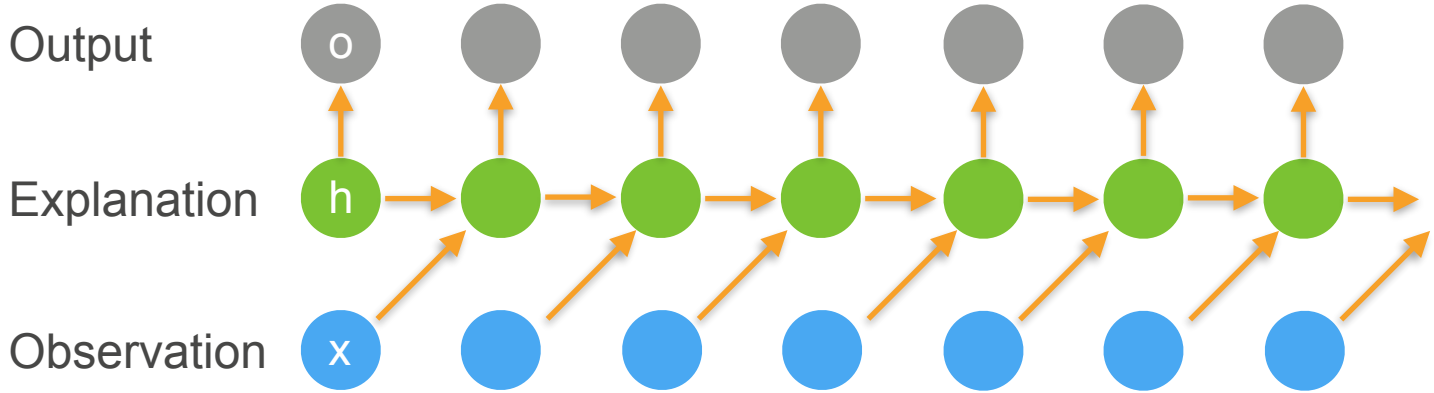
$$-\sum_{t=1}^{T} \log p(y_t | \text{model}) \quad \text{vs.} \quad \pi := -\frac{1}{T}\sum_{t=1}^{T} \log p(y_t | \text{model})$$

- 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

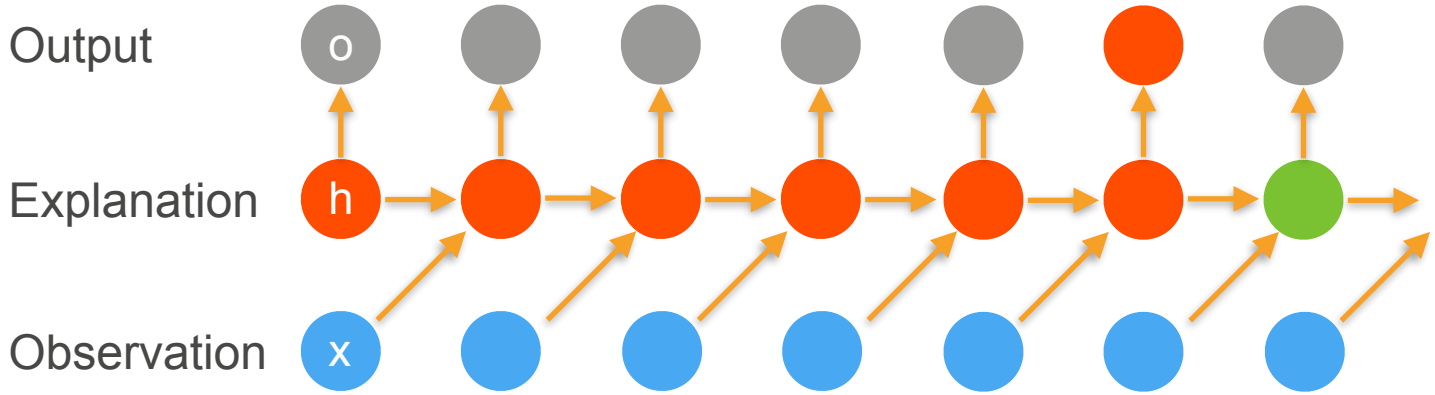
Unstable (divergence)

expensive

Latent State Gradient $\partial_w h_t$

- Gradient Recursion

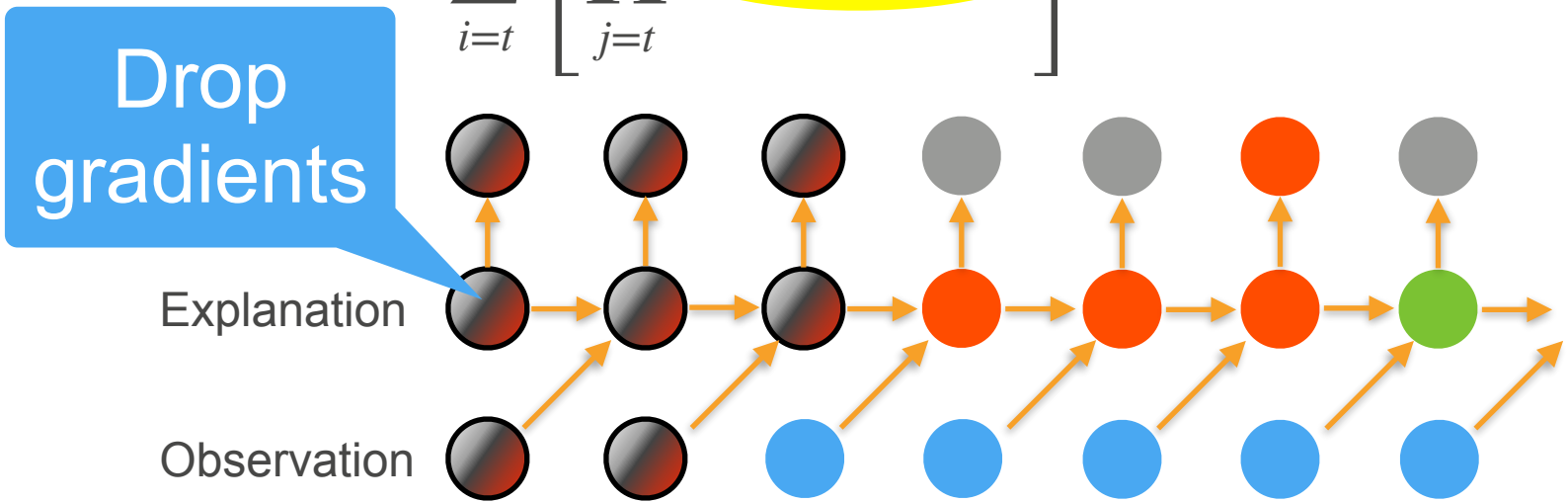
$$\partial_w h_t = \sum_{i=t}^{1} \left[\prod_{j=t}^{i} \partial_h f(x_j, h_{j-1}, w) \right] \partial_w f(x_i, h_{i-1}, w)$$



Latent State Gradient $\partial_w h_t$

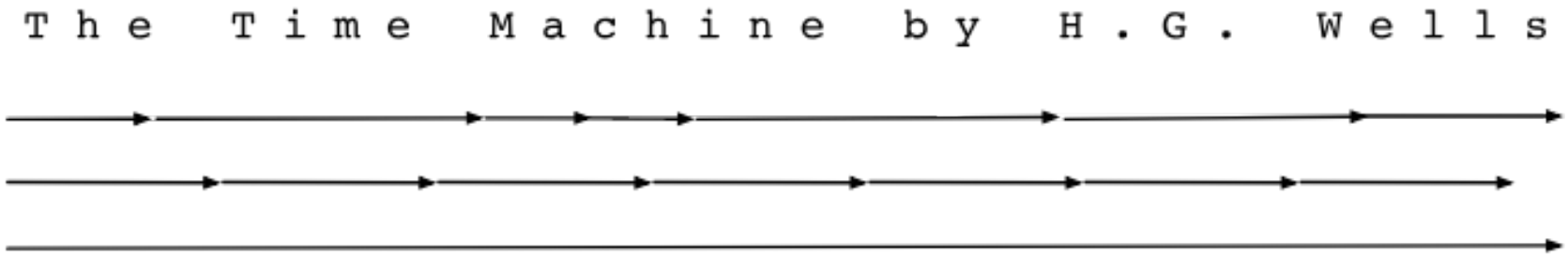
- Gradient Recursion

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



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)

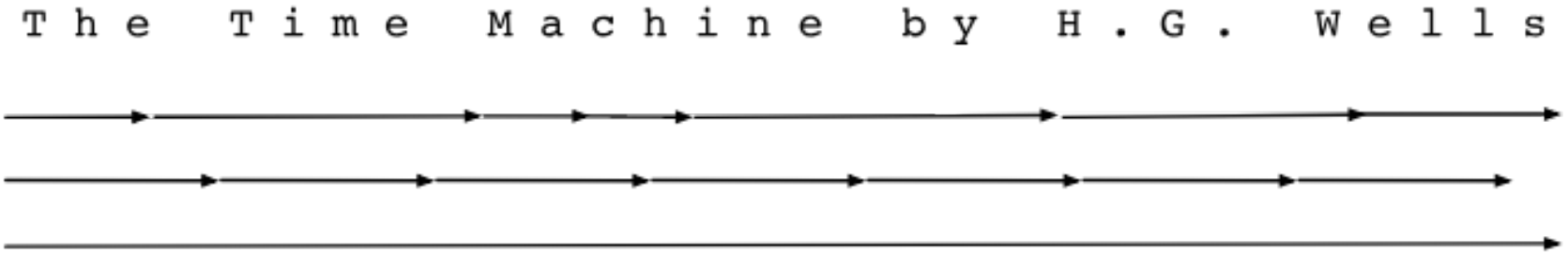


Truncated BPTT

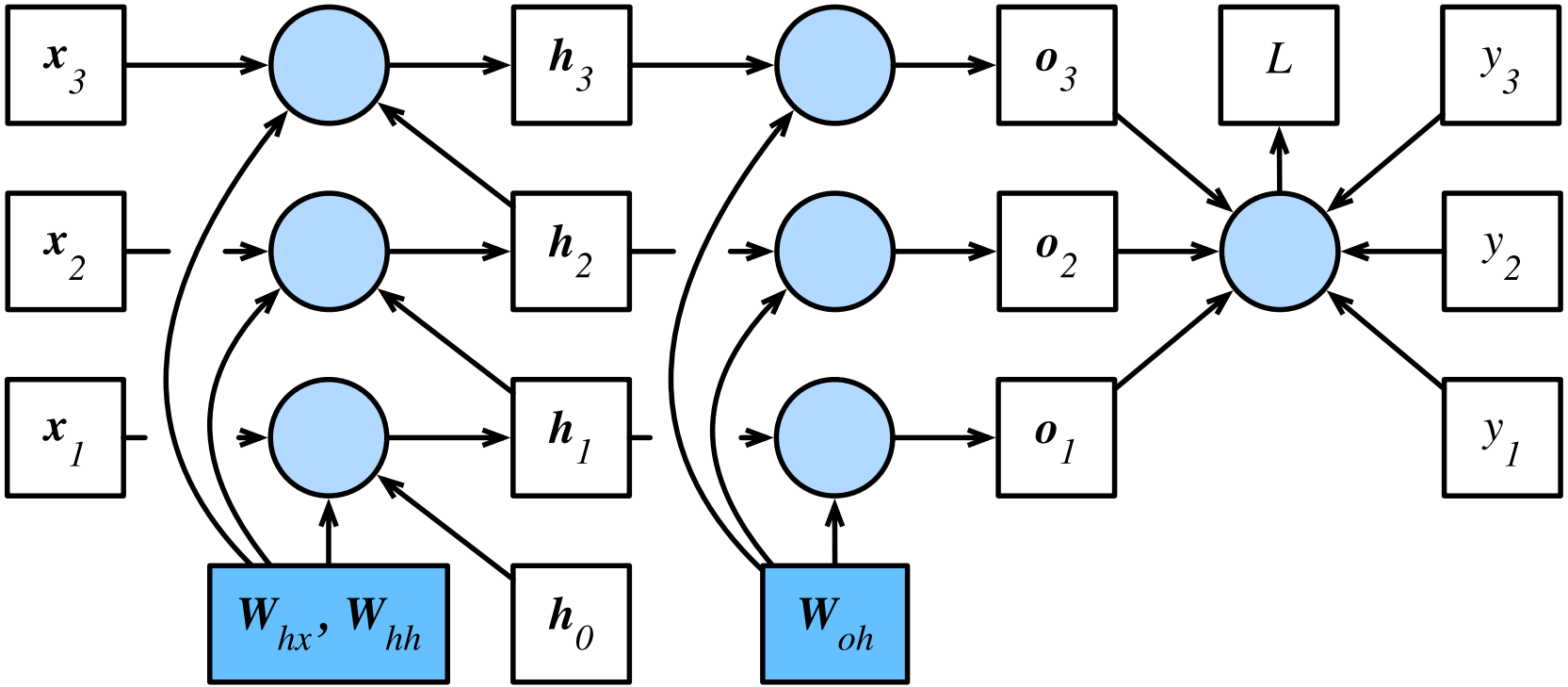
- Random variable instead of simple truncation

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

- 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

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

- State update gradient

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

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

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

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

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

Drop gradients

Truncation in practice

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

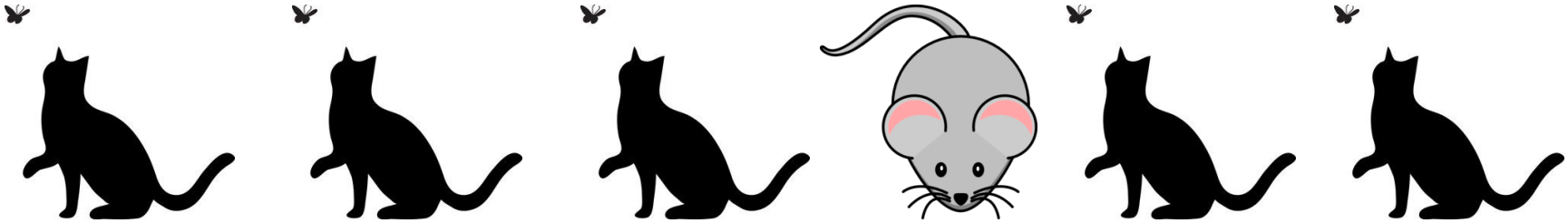
```
for s in state:
    s.detach()
```

- 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

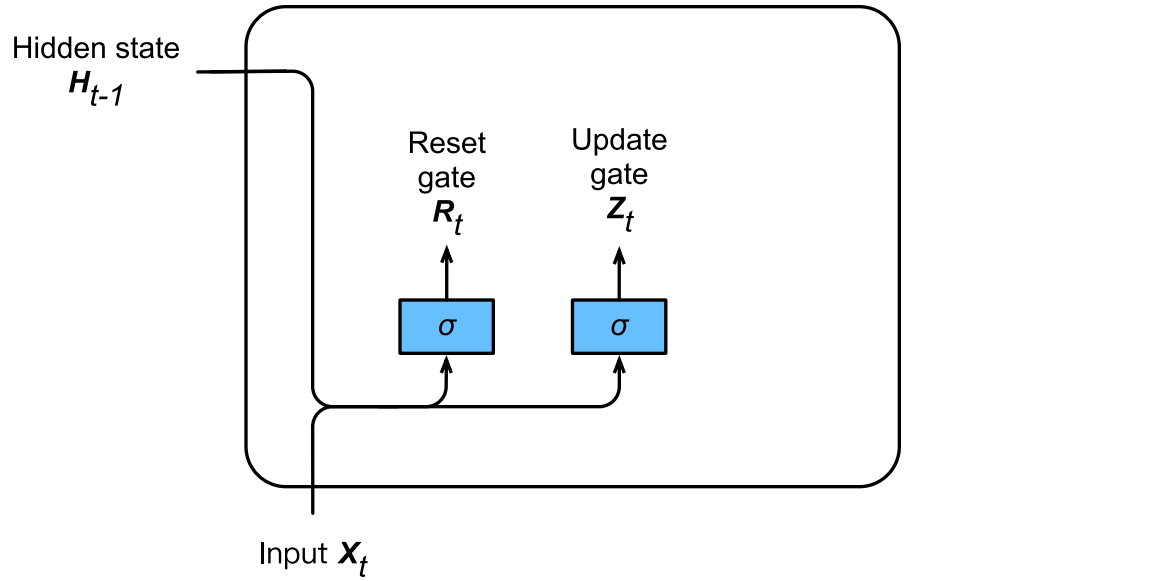
- Not all observations are equally relevant



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

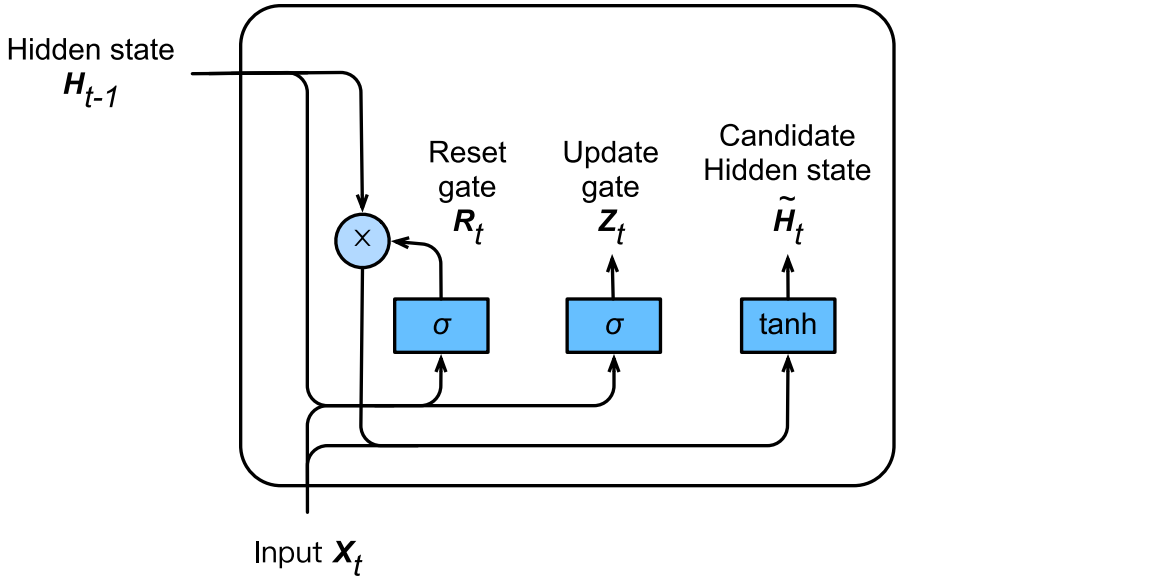
Gating

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



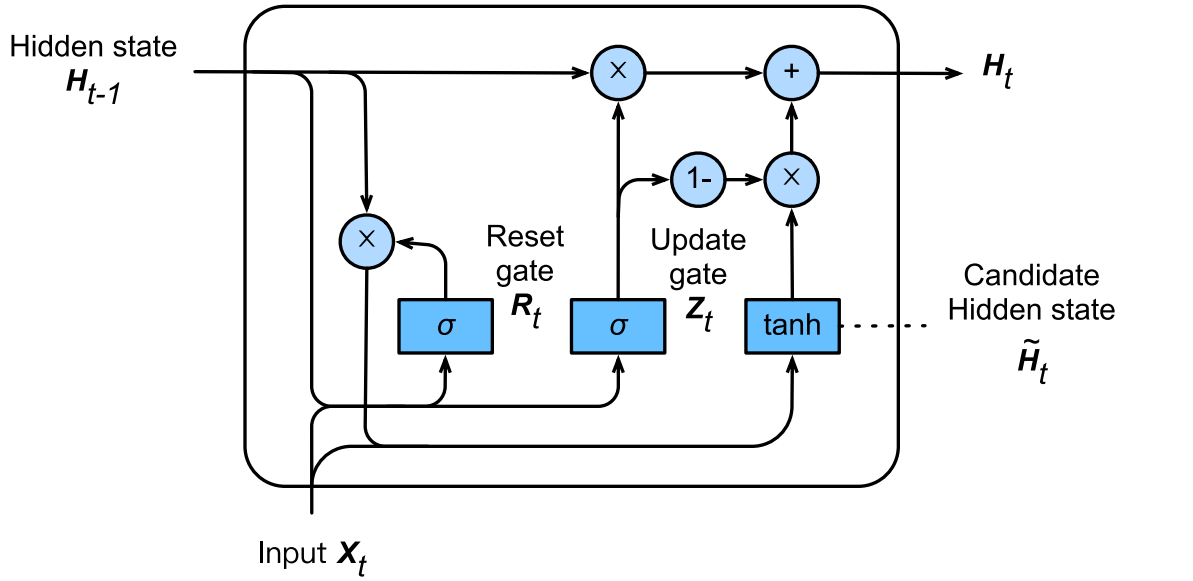
Candidate Hidden State

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



Hidden State

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



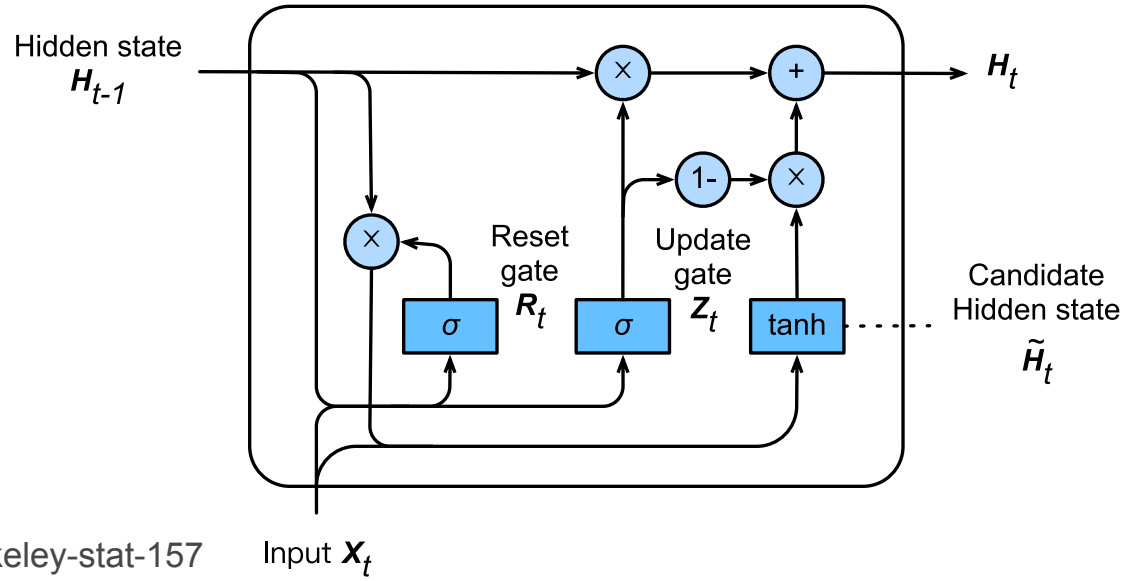
Summary

$$\boldsymbol{R}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hr} + \boldsymbol{b}_r),$$

$$\boldsymbol{Z}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xz} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hz} + \boldsymbol{b}_z)$$

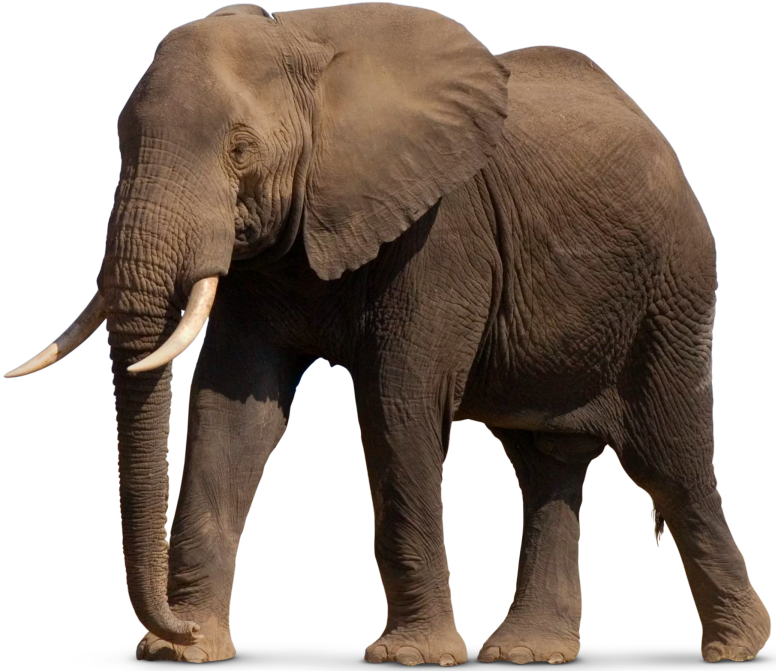
$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \left(\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}\right) \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

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

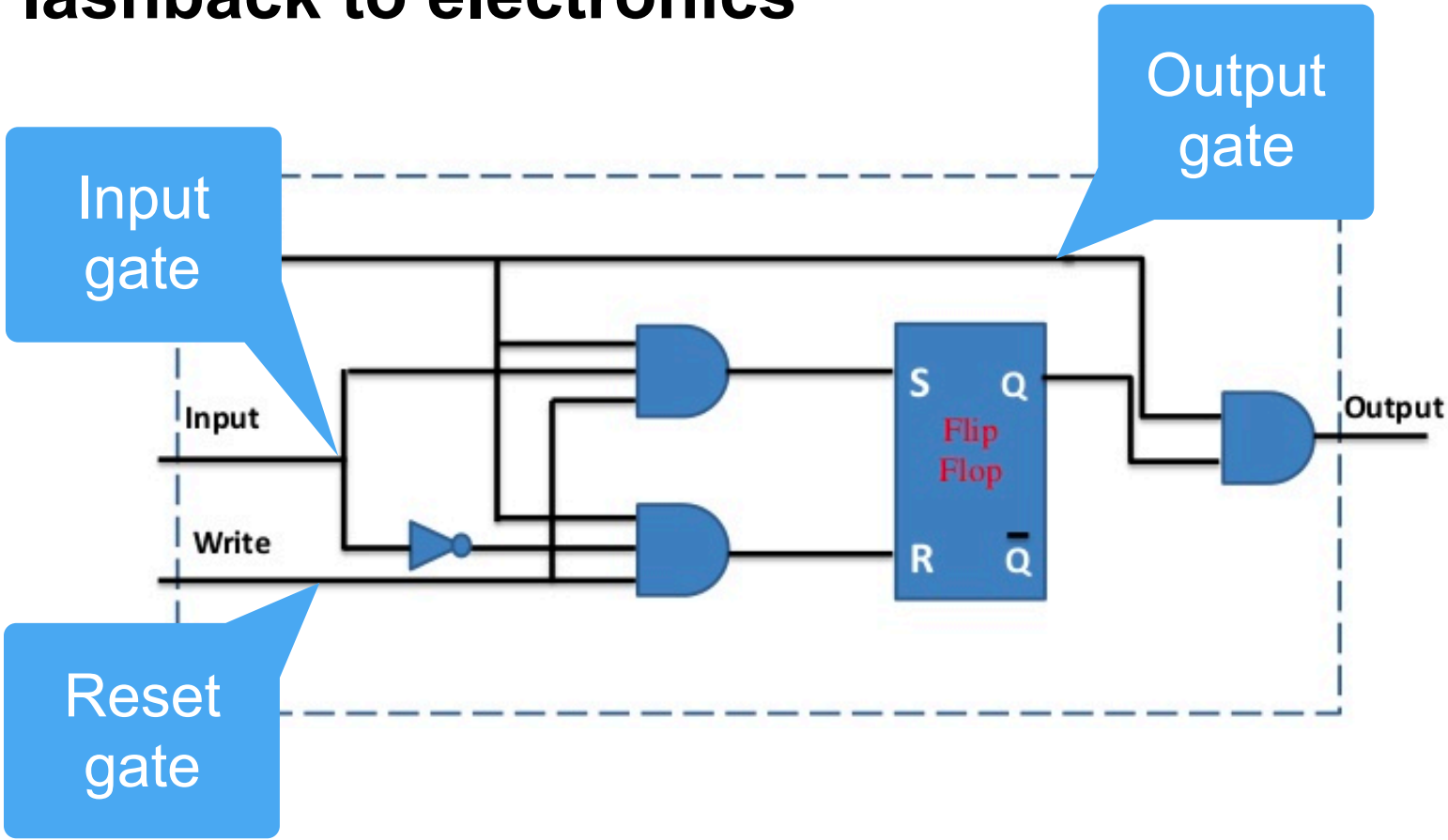


Code …

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

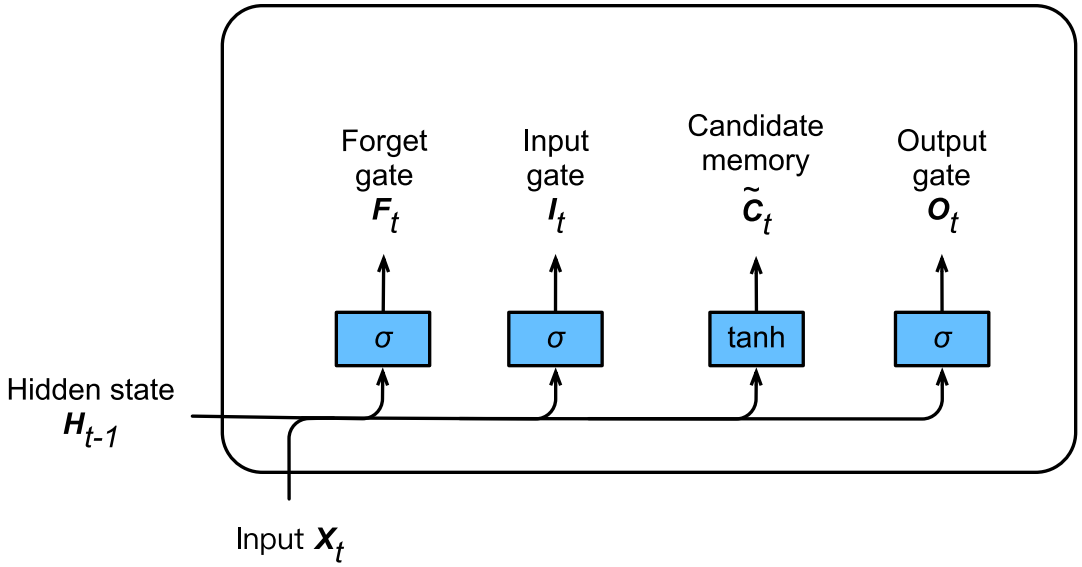
- **Hidden state** and **Memory cell**

Gates

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

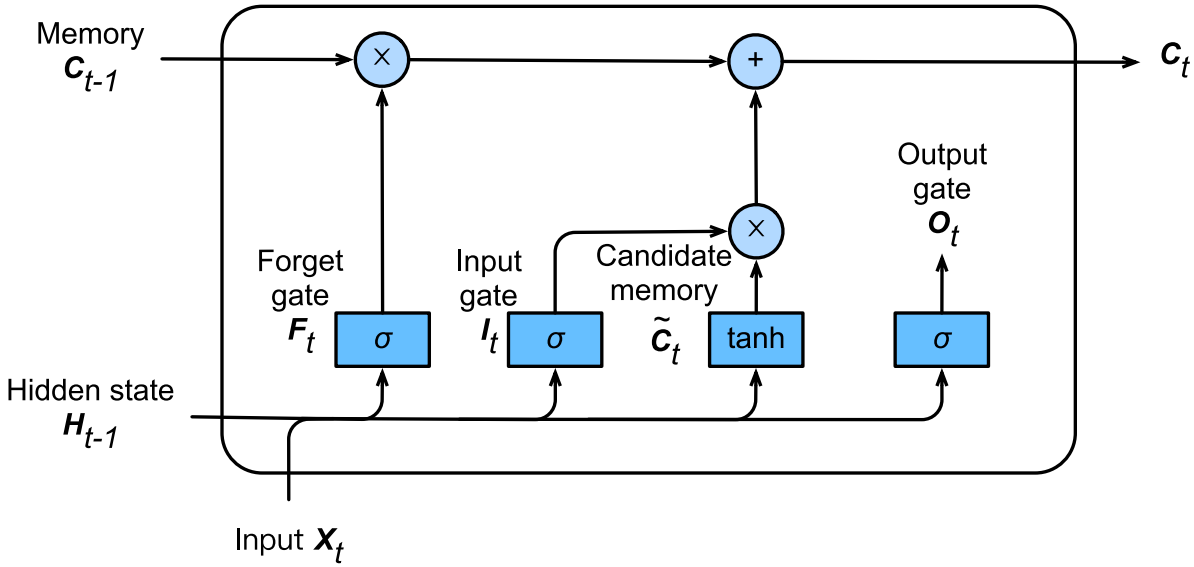
$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f)$$

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



Candidate Memory Cell

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



σ FC layer with activation fuction

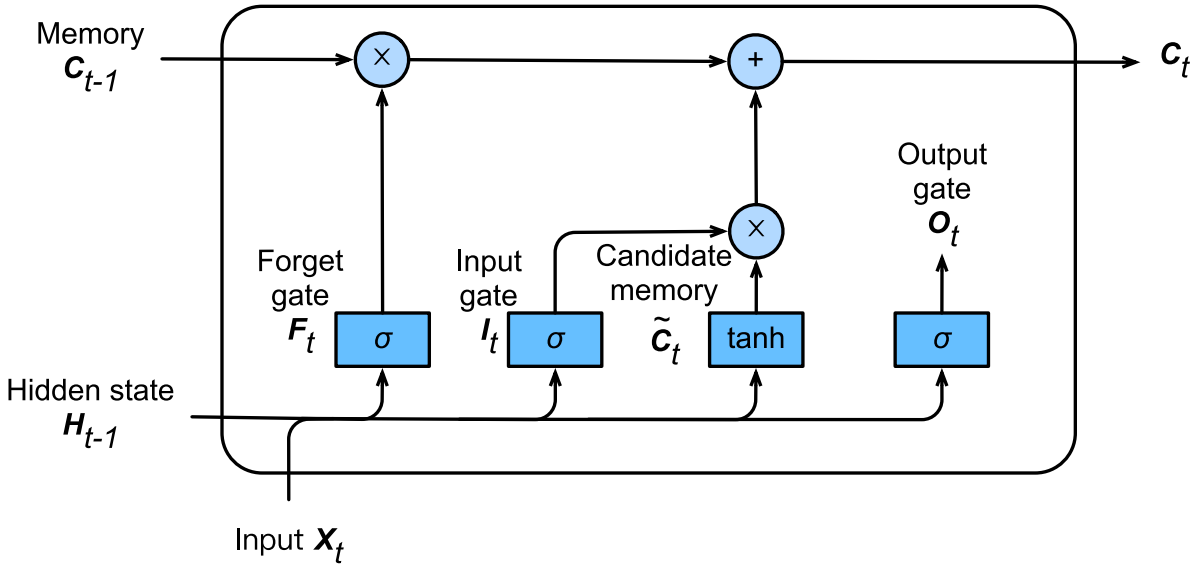
Element-wise Operator

Copy

Concatenate

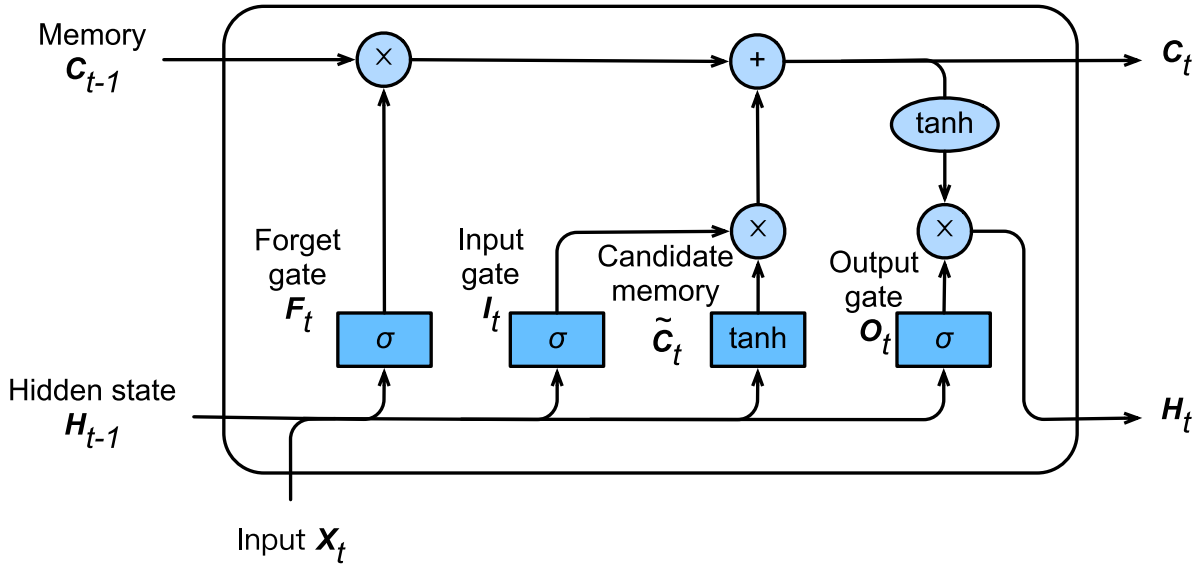
Memory Cell

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



Hidden State / Output

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



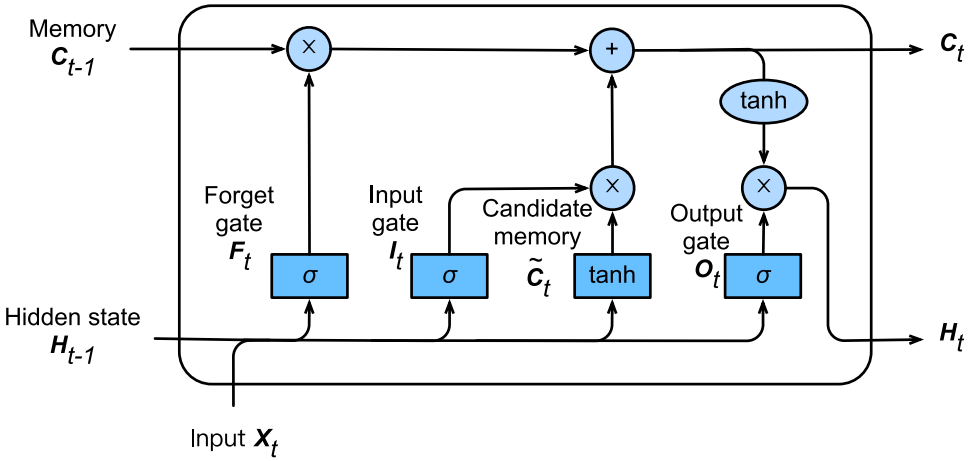
σ FC layer with activation fuction

Element-wise Operator

Copy

Concatenate

Hidden State / Output



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

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

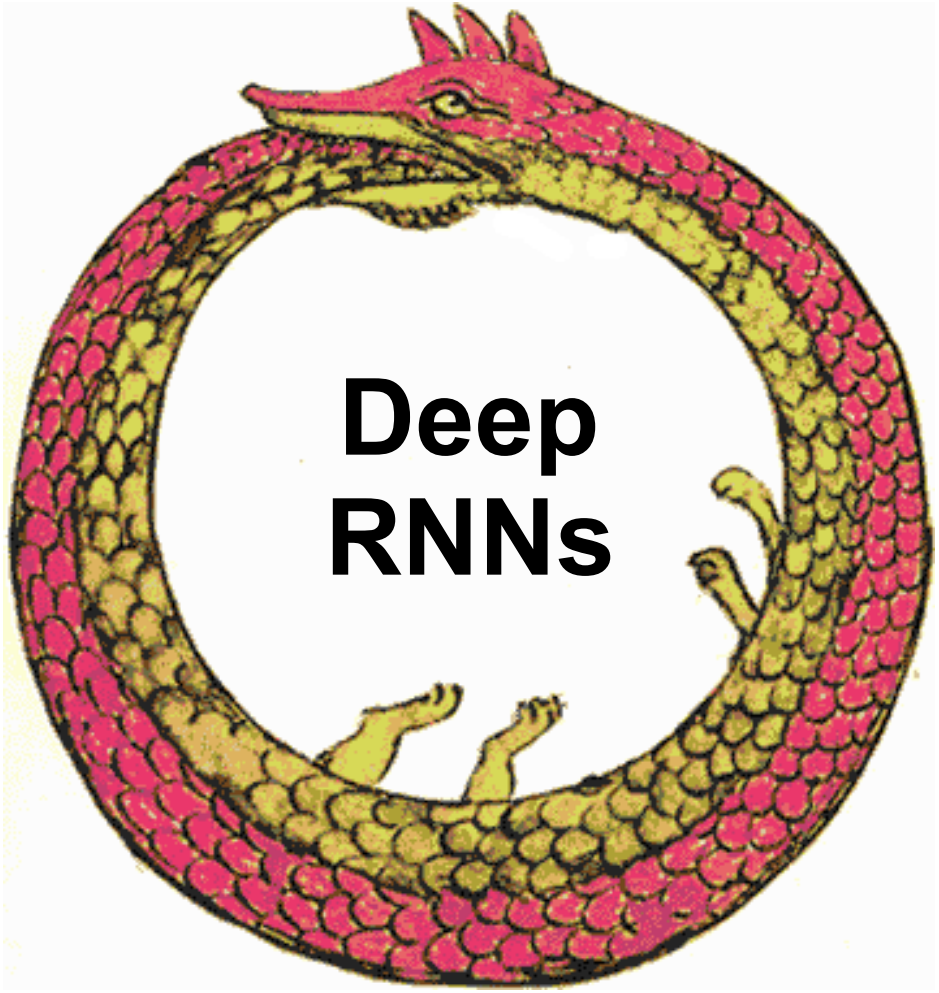
$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$$

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

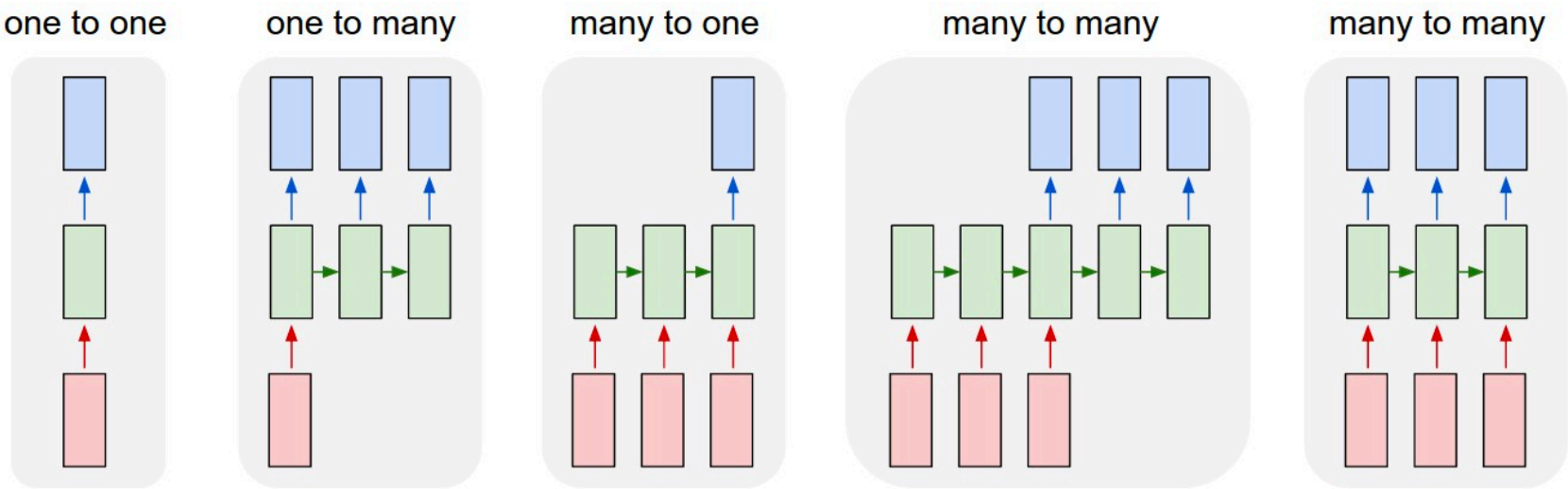
$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

$$H_t = O_t \odot \tanh(C_t)$$

Code …


Deep
RNNs

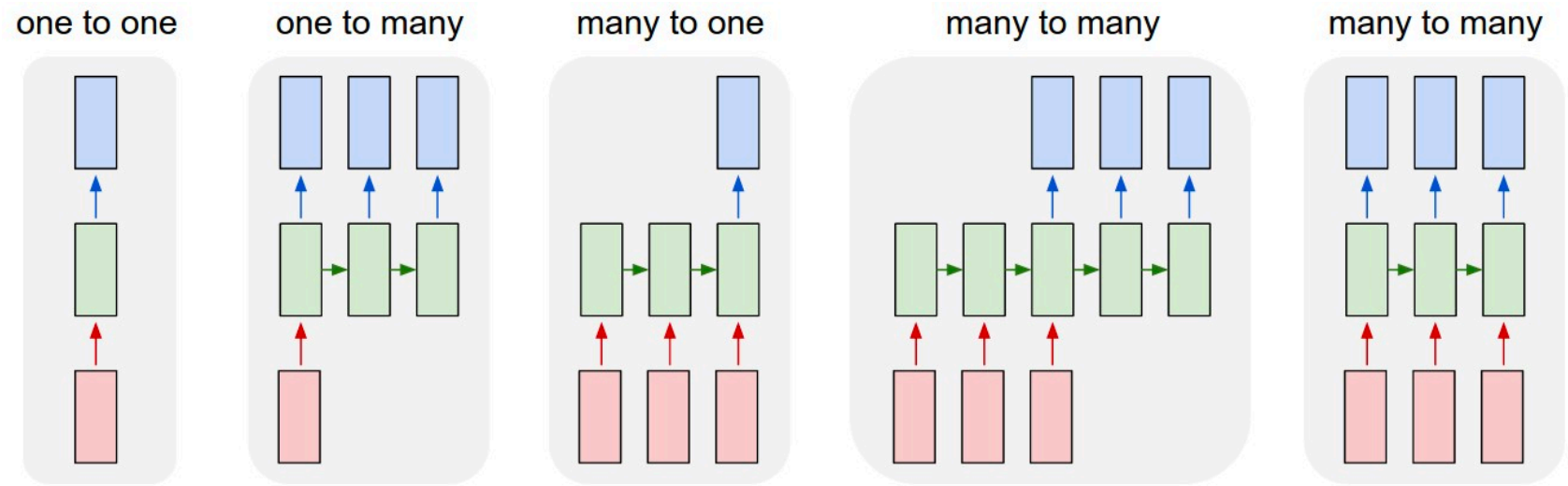
Using RNNs



- Encode sequence
- Decode sequence
- Do both

(image courtesy of karpathy.github.io)

Using RNNs



Poetry Generation

Sentiment Analysis

Document Classification

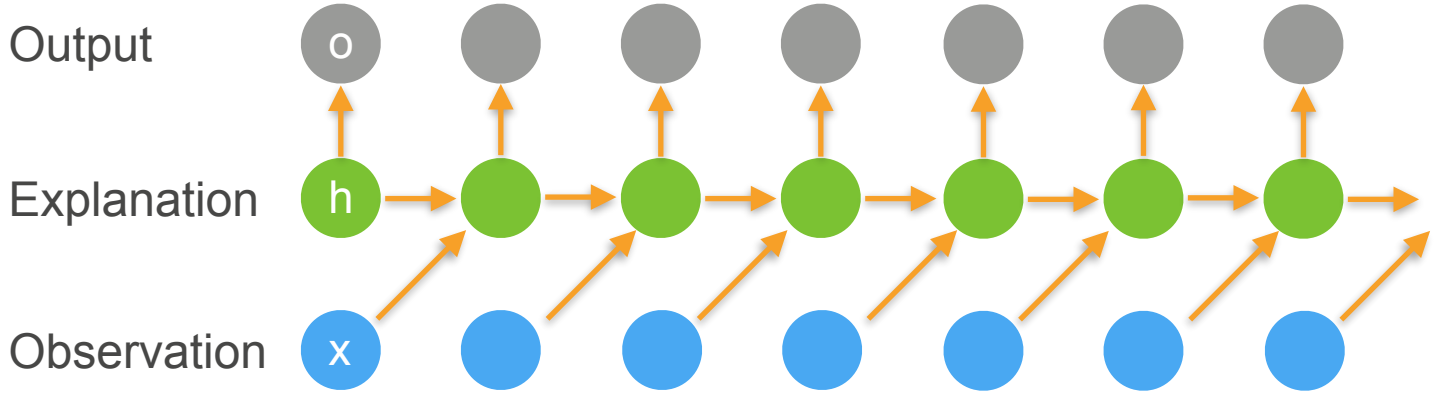
Question Answering

Machine Translation

Named Entity Tagging

(image courtesy of karpathy.github.io)

Recall - Recurrent Neural Networks



- Hidden State update

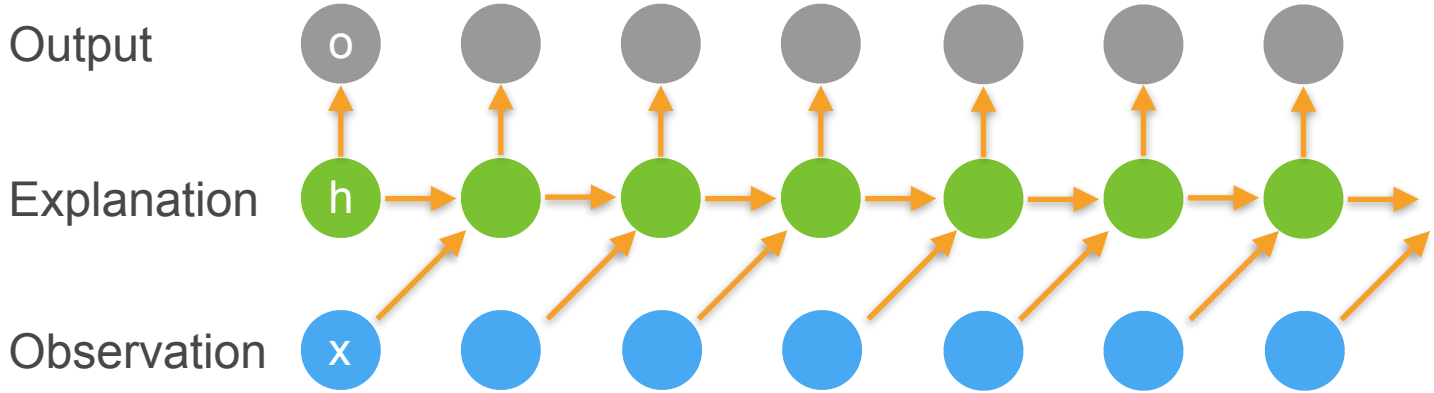
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

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

How to make more nonlinear?

Plan A - Nonlinearity in the units



- 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

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

Replace with MLP?

Plan A - Nonlinearity in the units

- Keeps the structure of the latent space
- More complex gradients (very costly)
- E.g. Zoph et al, 2018 learned cells with ~40 units (slow and expensive - nobody uses them in practice)

- Hidden State update

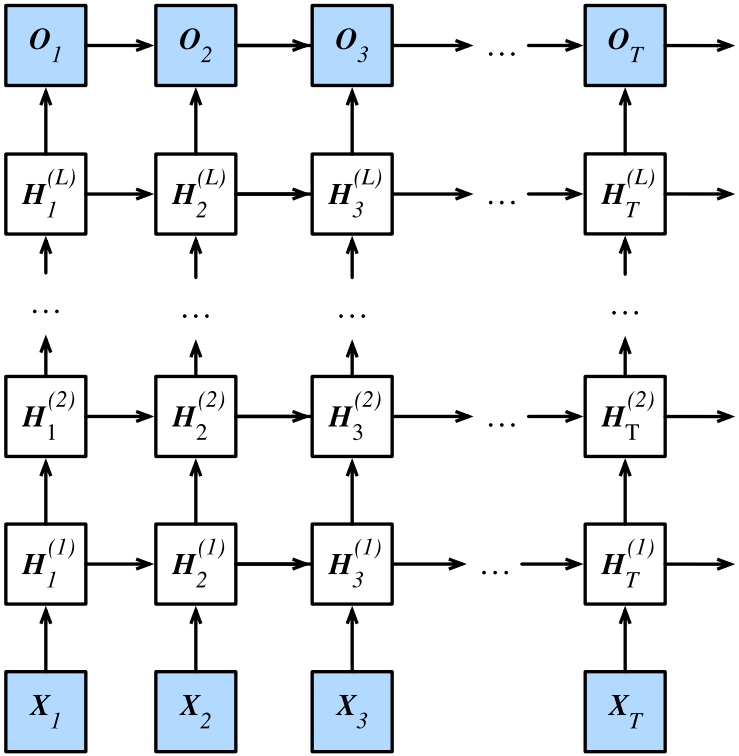
$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_{t-1} + \mathbf{b}_h)$$

- Observation update

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

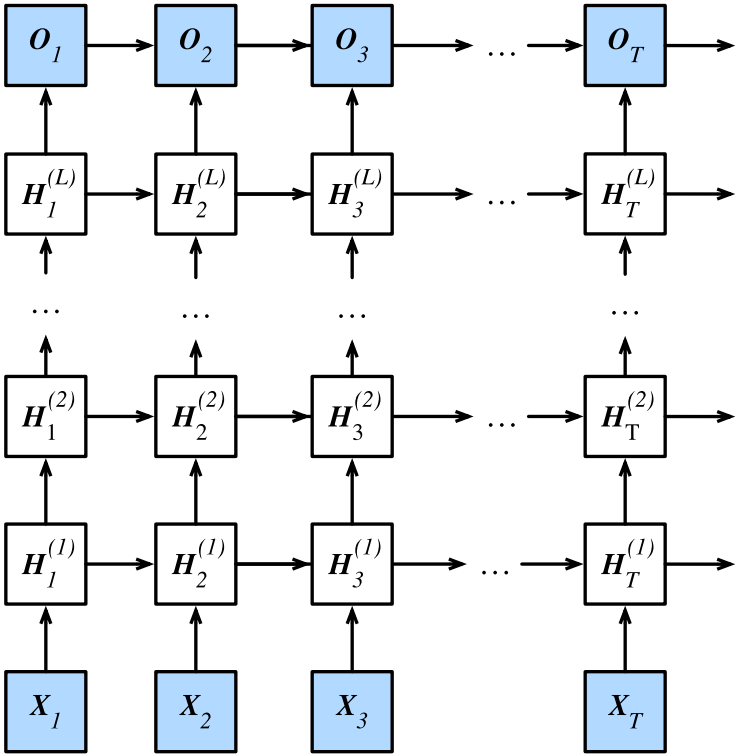
Replace with MLP?

Plan B - We go deeper



- Shallow RNN
  - Input
  - Hidden layer
  - Output
- Deep RNN
  - Input
  - **Hidden layer**
  - **Hidden layer**
  - **…**
  - Output

Plan B - We go deeper



$$\mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{X}_t)$$
$$\mathbf{O}_t = g(\mathbf{H}_t)$$

$$\mathbf{H}_t^1 = f(\mathbf{H}_{t-1}^1, \mathbf{X}_t)$$
$$\mathbf{H}_t^j = f(\mathbf{H}_{t-1}^j, \mathbf{H}_t^{j-1})$$
$$\mathbf{O}_t = g(\mathbf{H}_t^L)$$

Code …

Bidirectional RNNs

The Future Matters

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

The Future Matters

```
I am happy.
I am not   very hungry,
I am very  very hungry, I could eat half a pig.
```

The Future Matters

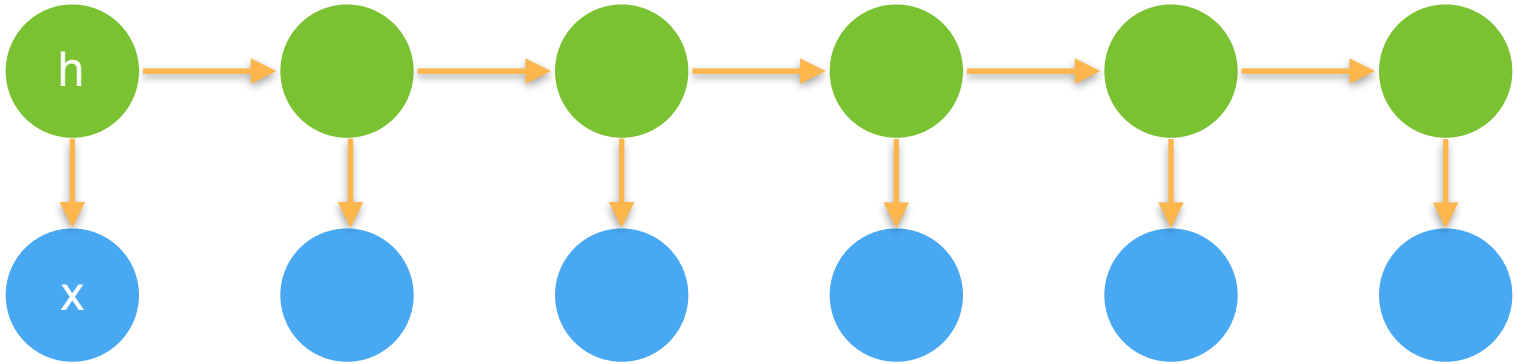
```
I am happy.
I am not   very hungry,
I am very  very hungry, I could eat half a pig.
```

- Very different words to fill in, depending on past and **future** context of a word.
- RNNs so far only look at the past
- In interpolation (fill in) we can use the future, too.

Flashback - Graphical Models

- Hidden Markov Model

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



- Can model sequence jointly and solve by dynamic programming

Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 \mid h_1) \prod_{i=2}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

Dynamic programming

$$p(x) = \sum_h p(h_1)p(x_1 \mid h_1) \prod_{i=2}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

$$= \sum_{h_2,\ldots h_T} \underbrace{\left[\sum_{h_1} p(h_1)p(x_1 \mid h_1)p(h_2 \mid h_1) \right]}_{=:\pi_2(h_2)} p(x_2 \mid h_2) \prod_{i=2}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

$$= \sum_{h_3,\ldots h_T} \underbrace{\left[\sum_{h_2} \pi_2(h_2)p(x_2 \mid h_2)p(h_3 \mid h_2) \right]}_{=:\pi_3(h_3)} p(x_3 \mid h_3) \prod_{i=3}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 \mid h_1)\prod_{i=2}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

- Forward pass

$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t \mid h_t)p(h_{t+1} \mid h_1)$$

Dynamic programming

$$p(x) = \sum_{h} \prod_{i=1}^{T-1} p(h_t \mid h_{t-1}) p(x_t \mid h_t) \cdot p(h_T \mid h_{T-1}) p(x_T \mid h_T)$$

$$= \sum_{h_1, \ldots h_{T-1}} \prod_{i=1}^{T-1} p(h_t \mid h_{t-1}) p(x_t \mid h_t) \cdot \underbrace{\left[\sum_{h_T} p(h_T \mid h_{T-1}) p(x_T \mid h_T) \right]}_{=:\rho_{T-1}(h_{T-1})}$$

$$= \sum_{h_1, \ldots h_{T-2}} \prod_{i=1}^{T-2} p(h_t \mid h_{t-1}) p(x_t \mid h_t) \cdot \underbrace{\left[\sum_{h_{T-1}} p(h_{T-1} \mid h_{T-2}) p(x_{T-1} \mid h_{T-1}) \right]}_{=:\rho_{T-2}(h_{T-2})}$$

Dynamic programming

- Joint probability

$$p(x, h) = p(h_1)p(x_1 \mid h_1)\prod_{i=2}^{T} p(h_t \mid h_{t-1})p(x_t \mid h_t)$$

- Forward pass

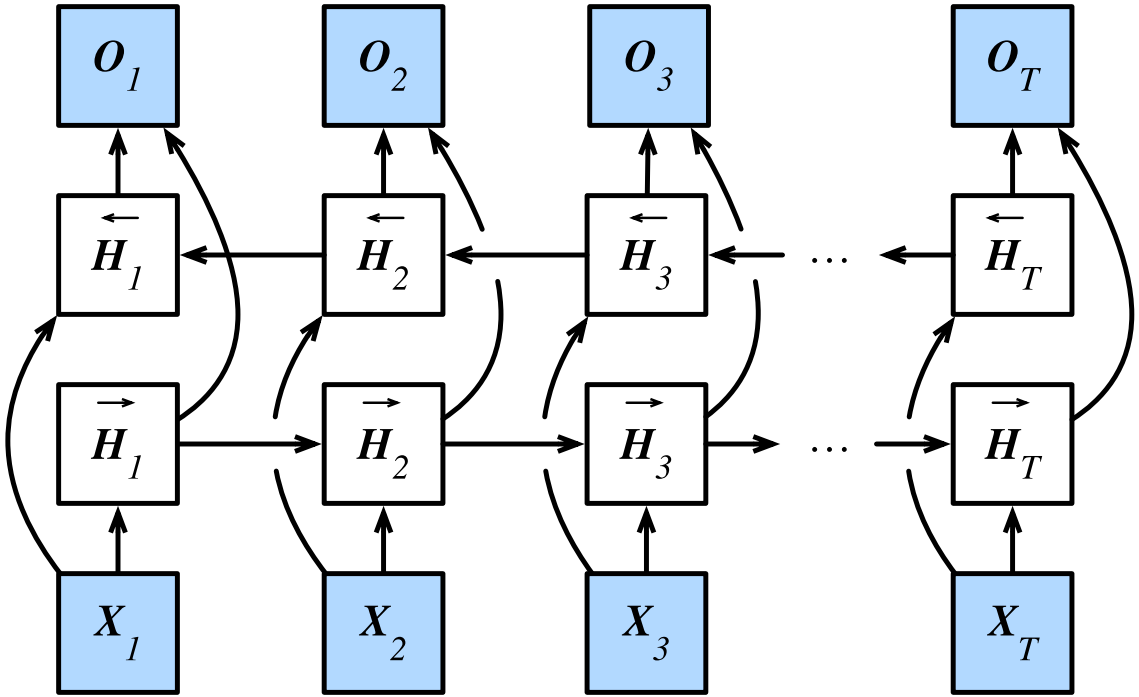
$$\pi_{t+1}(h_{t+1}) = \sum_{h_t} \pi_t(h_t)p(x_t \mid h_t)p(h_{t+1} \mid h_1)$$

- Backward pass

$$p(x_j \mid x_{-j}) \propto \sum_{h_j} \pi_j(h_j)\rho_j(h_j)p(x_j \mid h_j)$$

Can we do this with RNNs, too?

Bidirectional RNN



- One RNN forward
- Another one backward
- Combine both hidden states for output generation

```
epoch 600, perplexity 1.016867, time 0.15 sec
 - travellerer cumplph peouguninininininin suppepepepepepepepepepe
 - time travellererer  fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
 - traveller hime of copspepepep smefsffff''''''''''''''''''''
 - time travellerer prefifififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
 - travellererererererererererererererererererererererererer
 - time travellererererererererererererererererererererererererer
```

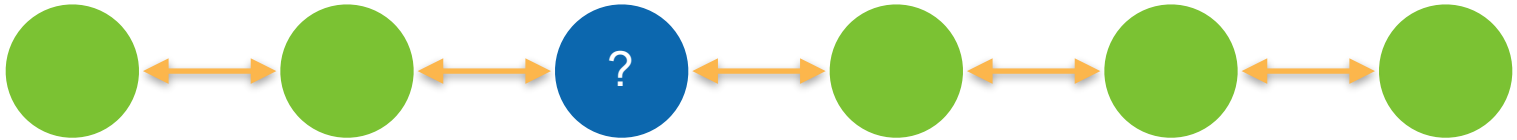
This does not work for sequence generation

```
epoch 600, perplexity 1.016867, time 0.15 sec
 - travellerer cumplph peouguninininin suppepepepepepepepepepe
 - time travellererer  fuf this shanatatatatatatatatatatatatatata
epoch 800, perplexity 1.007069, time 0.15 sec
 - traveller hime of copspepepep smefsffff''''''''''''''''''''
 - time travellerer prefifififididididididididididididididididi
epoch 1000, perplexity 1.001932, time 0.15 sec
 - travellererererererererererererererererererererererererer
 - time travellererererererererererererererererererererererererer
```

Why?

Reasons

- Training time



- Test time

Next lecture

Can still use it to **encode** the sequence