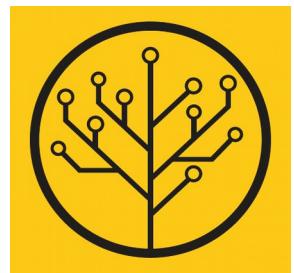


Practical Reinforcement Learning

Episode 6 ¾, spring 2018

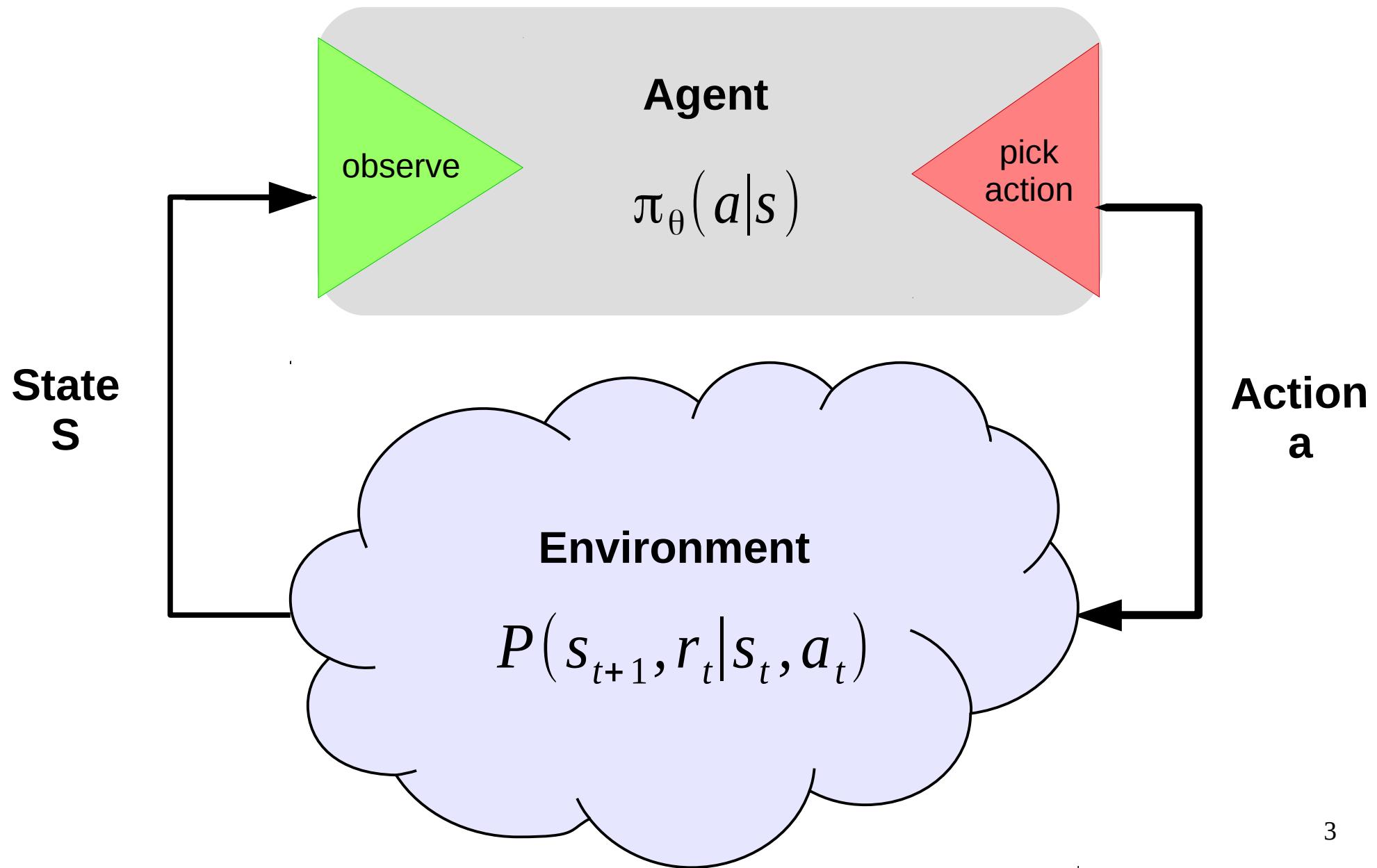
Recurrent neural networks

deep learning recap 2

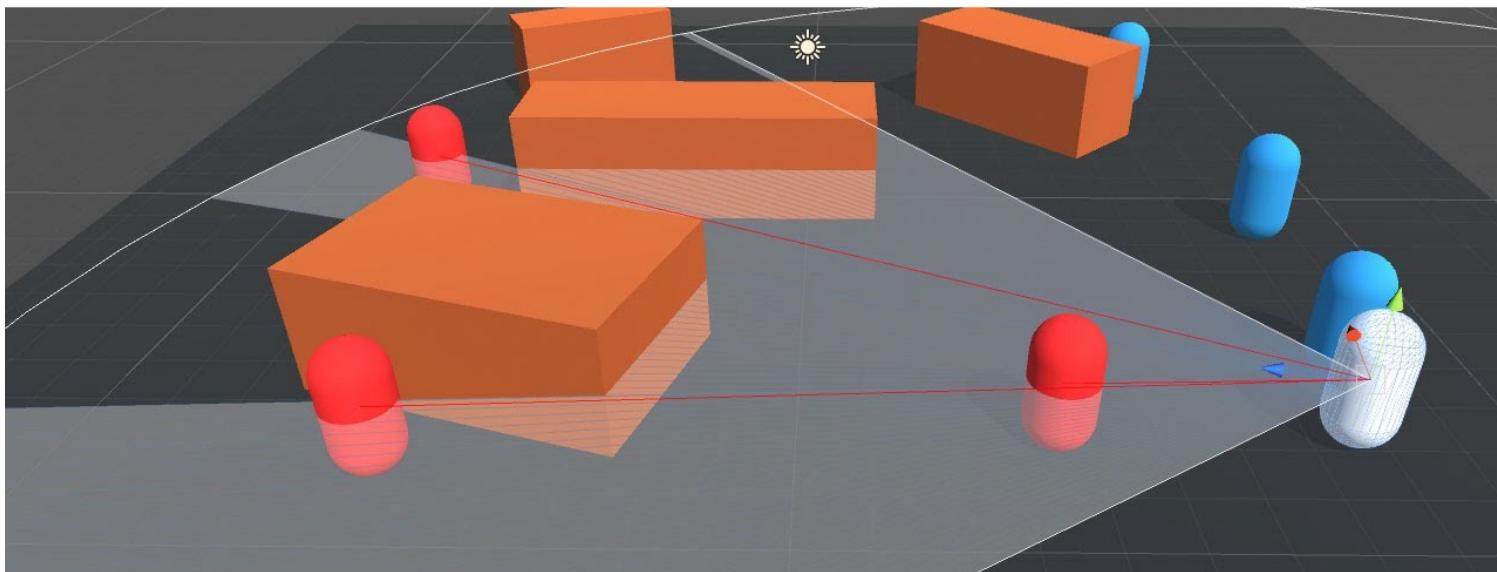
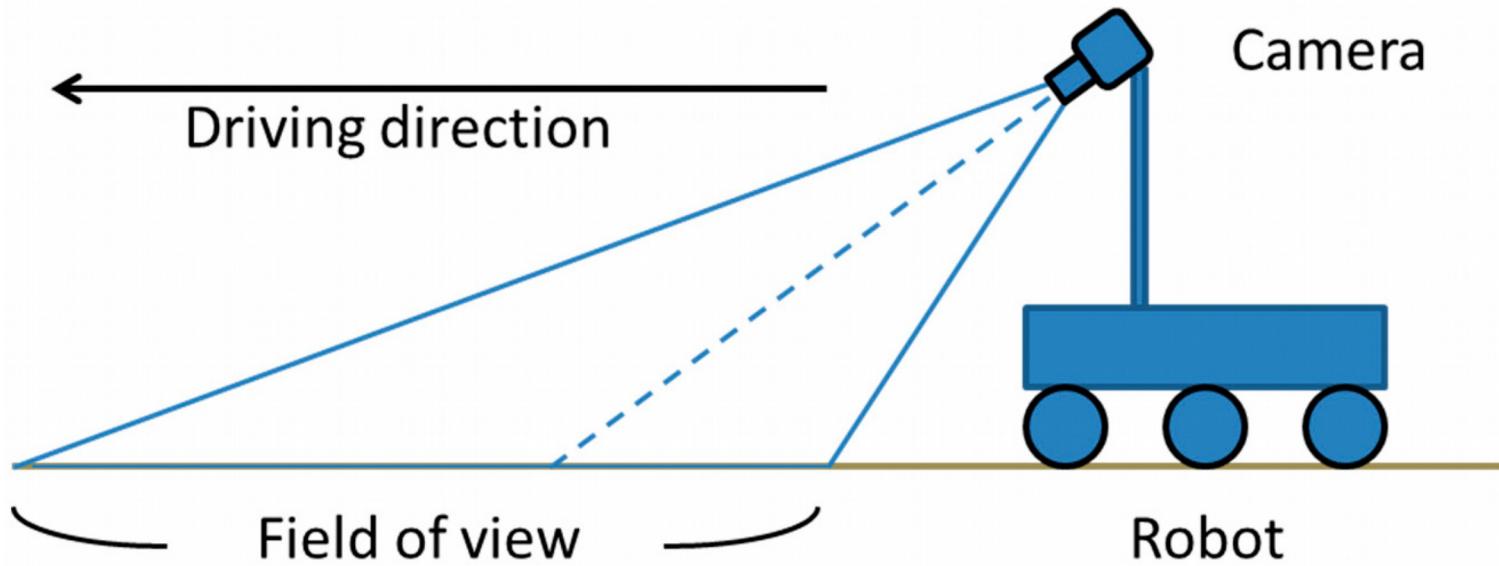


Previously on reinforcement learning...

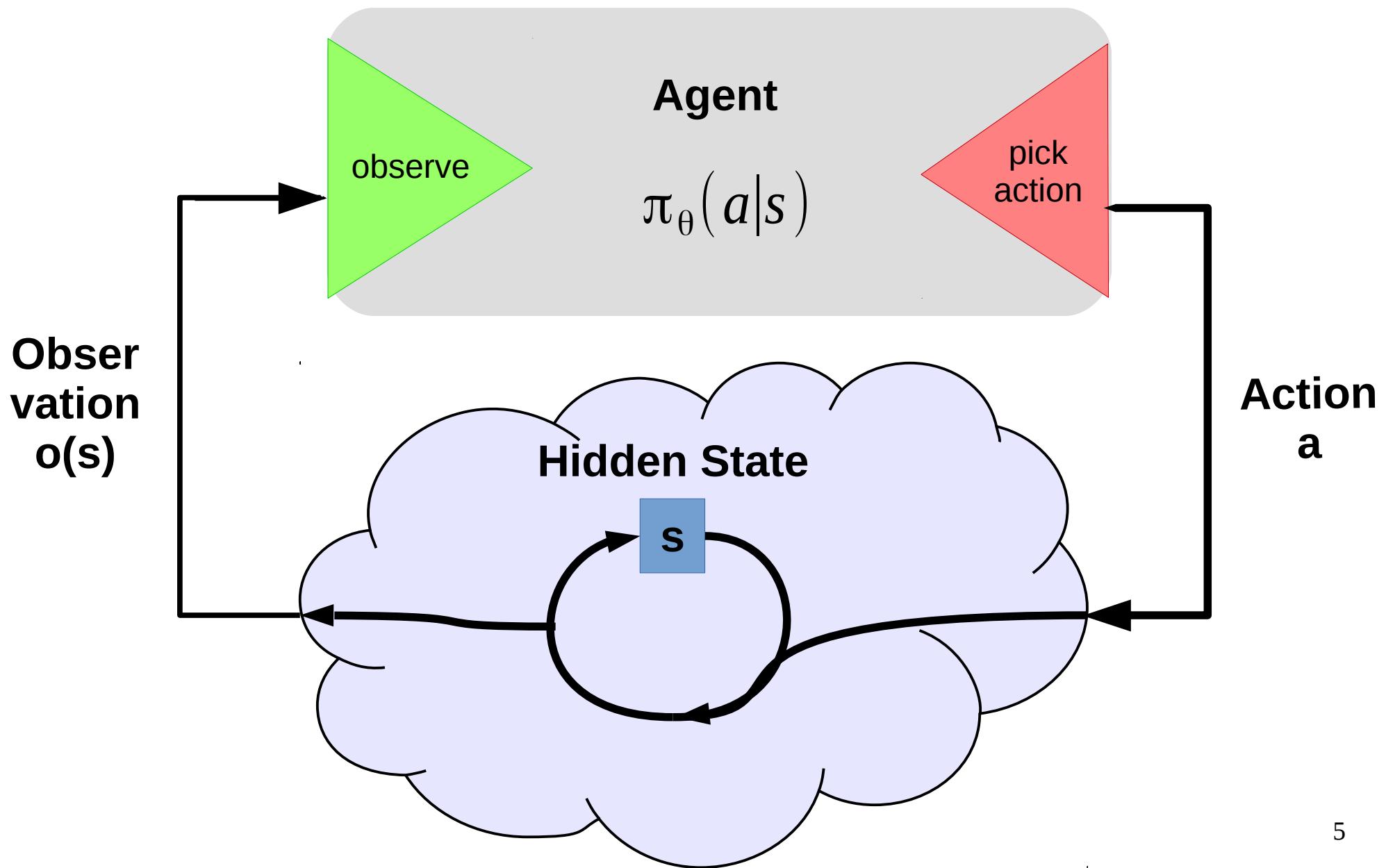
Recap: MDP



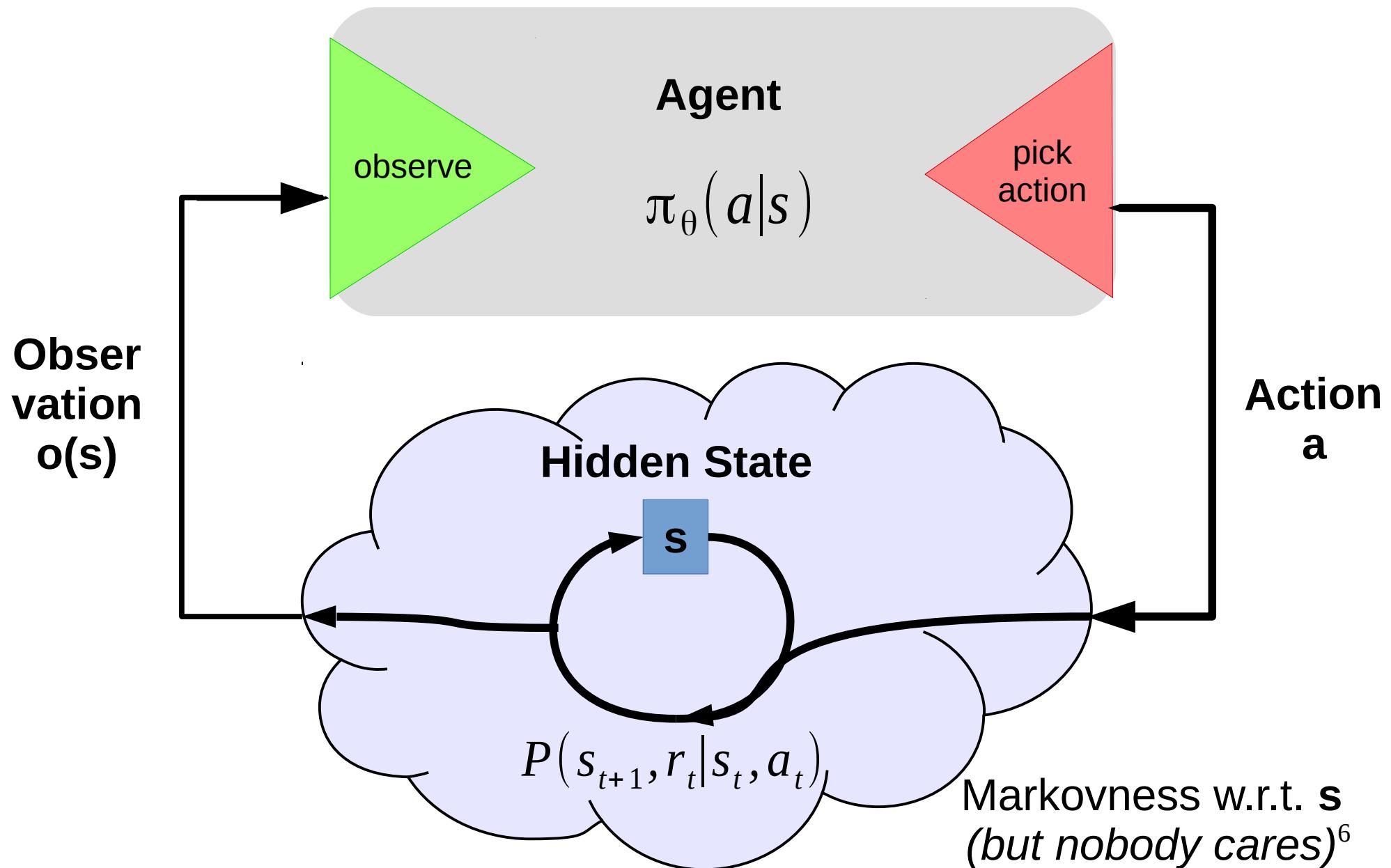
Partially Observable MDP



Partially Observable MDP



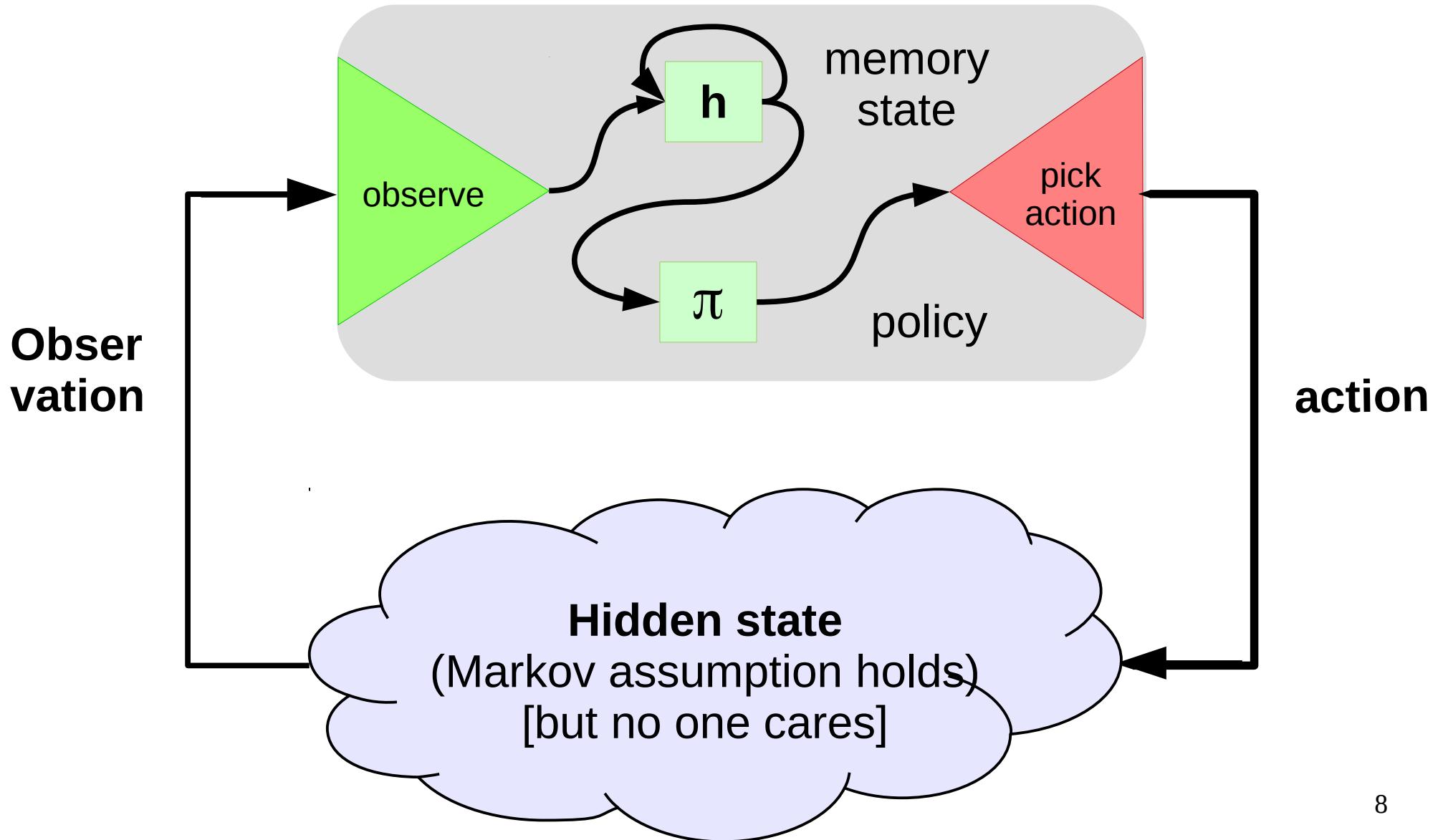
Partially Observable MDP



Partially Observable MDP

**How do we train an agent
in such process?**

Partially observable MDP



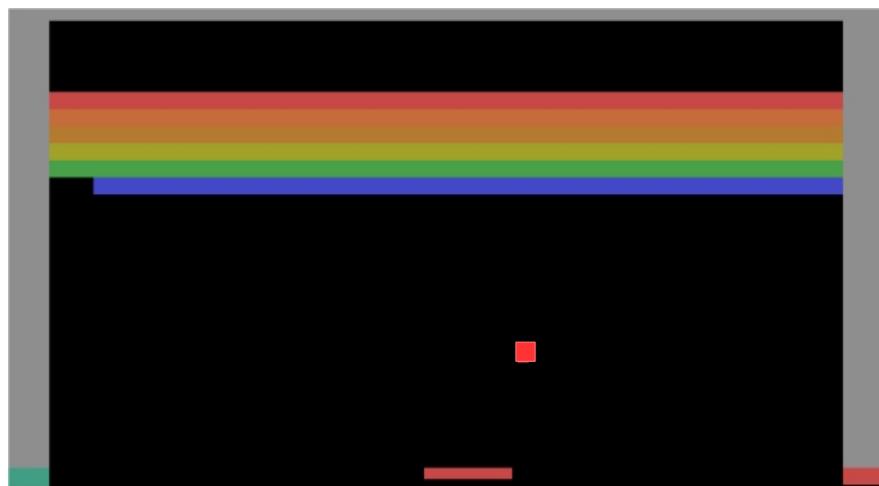
N-gram heuristic

Idea:

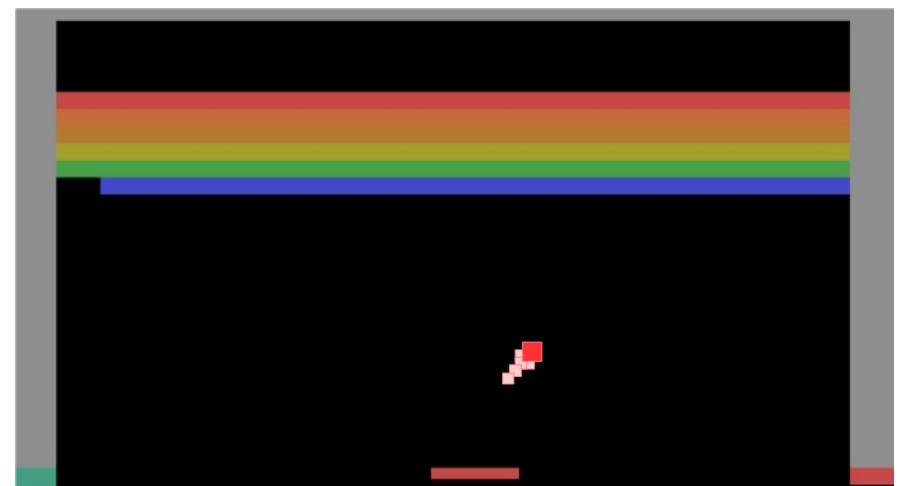
$$s_t \neq o(s_t)$$

$$s_t \approx (o(s_{t-n}), a_{t-n}, \dots, o(s_{t-1}), a_{t-1}, o(s_t))$$

e.g. ball movement in breakout

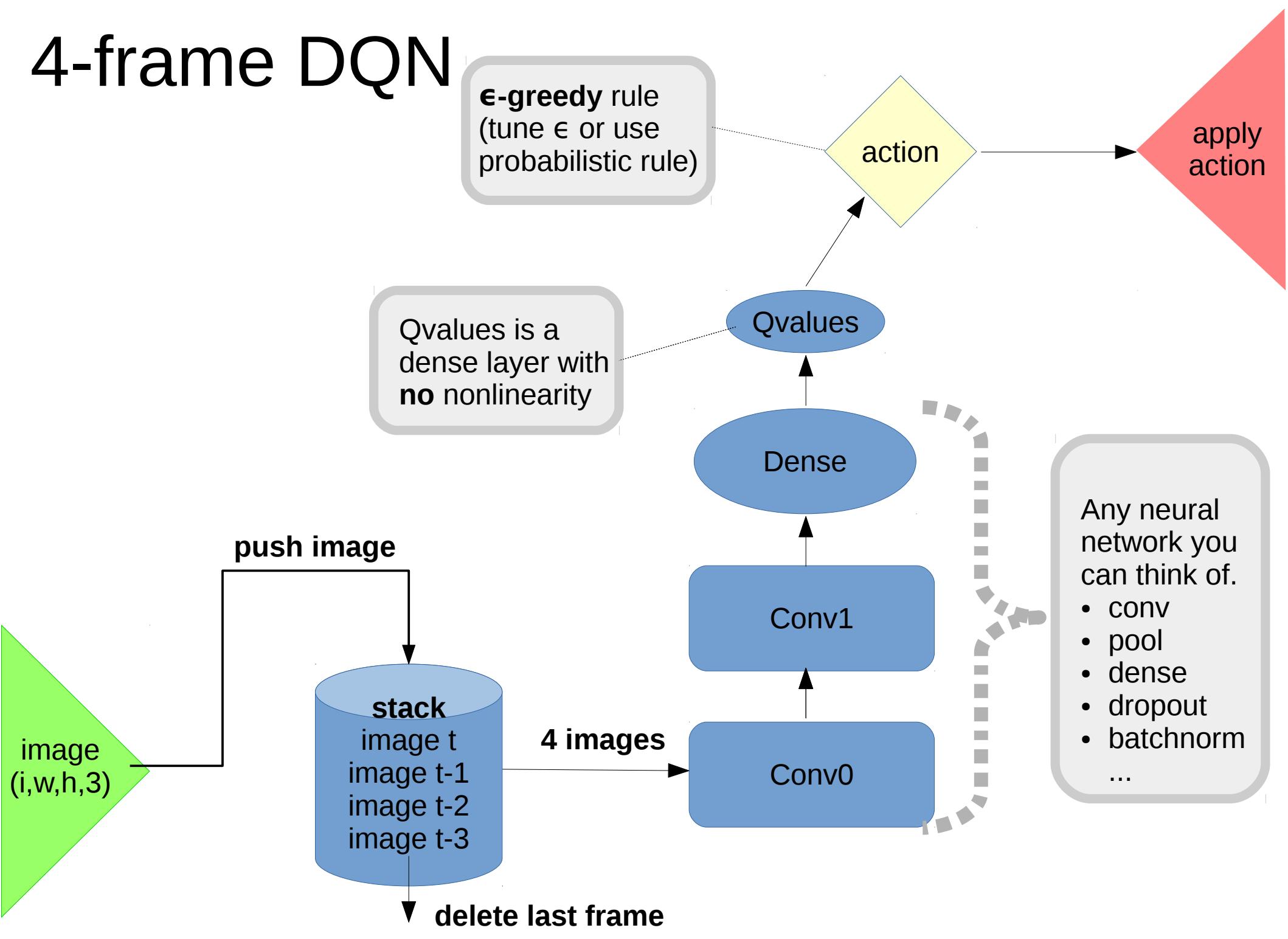


- One frame



- Several frames

4-frame DQN



Can we do better?

- There's a hidden process
- You only see some part of it
- You want to learn hidden states

<Yes, recurrent neural networks>

Sequential data

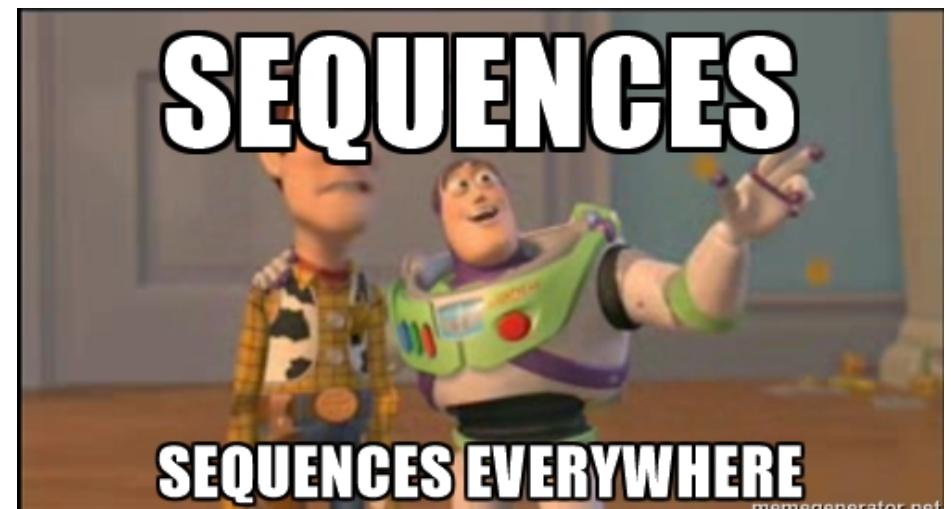
- Time series
 - Financial data analysis
 - Demand prediction
 - Predict vehicle breakdown using sensor data
 - Medical sensors
 - e.g. sugar level

Sequential data

- Time series
 - Financial data analysis
 - Demand prediction
 - Predict vehicle breakdown using sensor data
 - Medical sensors
 - e.g. sugar level

- Sound
 - Speech recognition
 - Text to speech
 - Music generation
 - Music recommendation
 - ...

- Text
 - Generating tweets, poetry
 - Sentiment analysis
 - See last lecture :)
- Spatio-temporal
 - Video
 - Precipitation maps
 - Ultrasonography
 - Robot camera frames



Could go on all day

Time series @finance

Data:

- Stock indices
- Commodities
- Forex

Objectives:

- Portfolio management
- Volatility targeting
- Estimating true value
- ...



Time series @finance

Data:

- Stock indices
- Commodities
- Forex

Objectives:

- ~~Portfolio management~~ ~ trading stuff
- ~~Volatility targeting~~ ~ evaluating risk
- Estimating true value
- ...



Natural language as time series

Data:

- Literature
- Conversation
- Tweets
- Book scans
- Speech

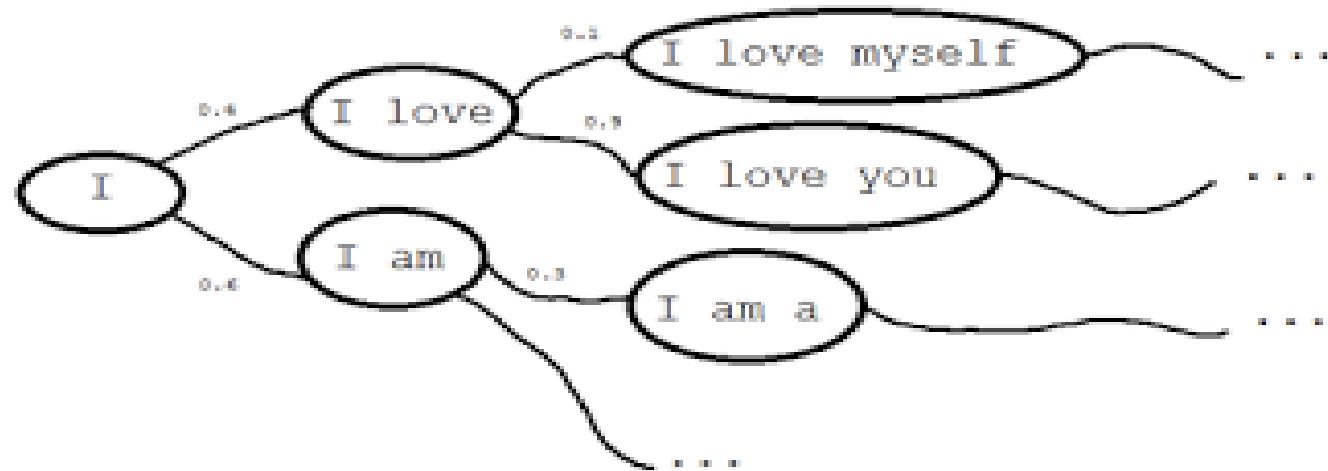


Language model

Objective:

- Learn $P(\text{text})$

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1 w_0) \cdot \dots \cdot P(w_n| \dots)$$



Language model

Why learning it?

- Detect languages as $P(\text{text}|\text{language})$
- Sentiment analysis $P(\text{text}|\text{happy})$
- Any text analysis you can imagine
- Generate texts!
 - Cool article <http://bit.ly/1K610le>
 - Generating clickbait: <http://bit.ly/21cZM70>

Language model

- Actual distribution

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1 w_0) \cdot \dots \cdot P(w_n| \dots)$$

- Bag of words assumption (independent words)

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1) \cdot P(w_2) \cdot \dots \cdot P(w_n)$$

- Anything better?

Language model

- Actual distribution

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1 w_0) \cdot \dots \cdot P(w_n| \dots)$$

- Bag of words assumption (independent words)

$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1) \cdot P(w_2) \cdot \dots \cdot P(w_n)$$

- Markov assumption

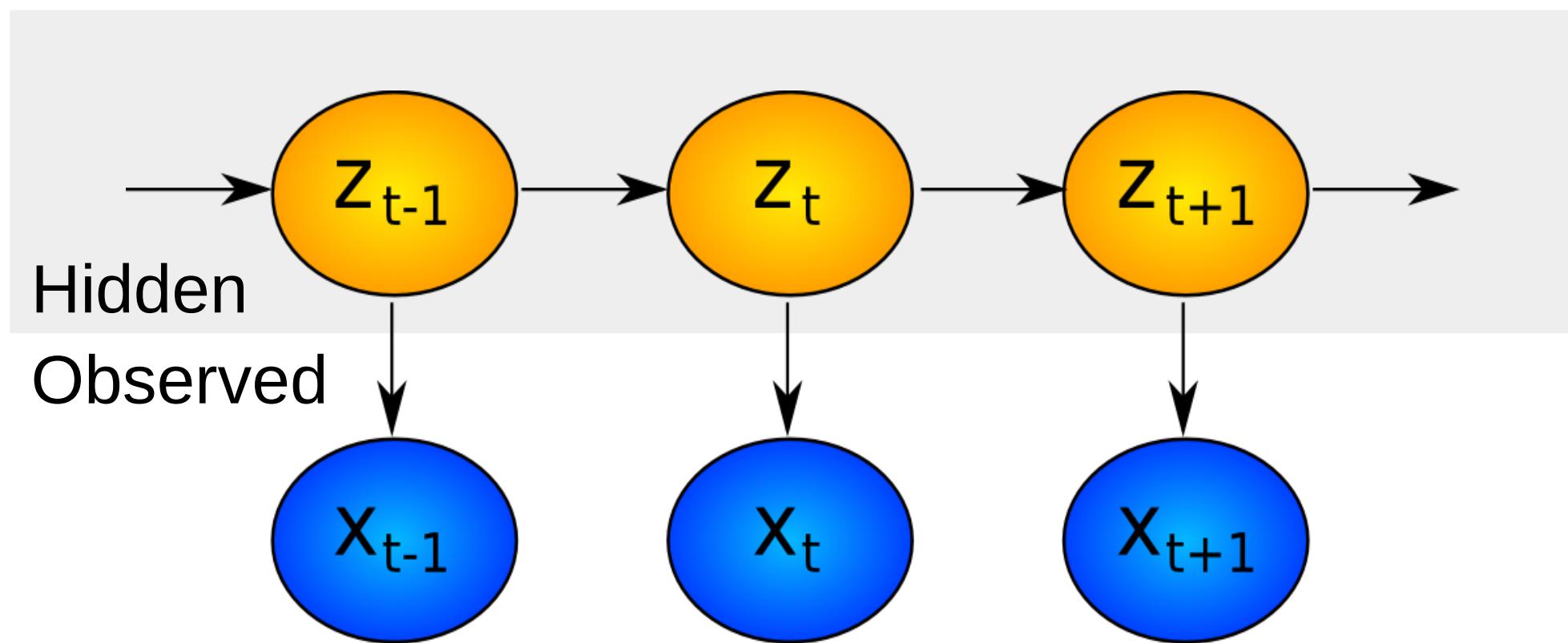
$$P(\text{text}) = P(w_0, w_1, \dots, w_n) = P(w_0) \cdot P(w_1|w_0) \cdot P(w_2|w_1) \cdot \dots \cdot P(w_n|w_{n-1})$$

- also 3-gram, 5-gram, 100-gram

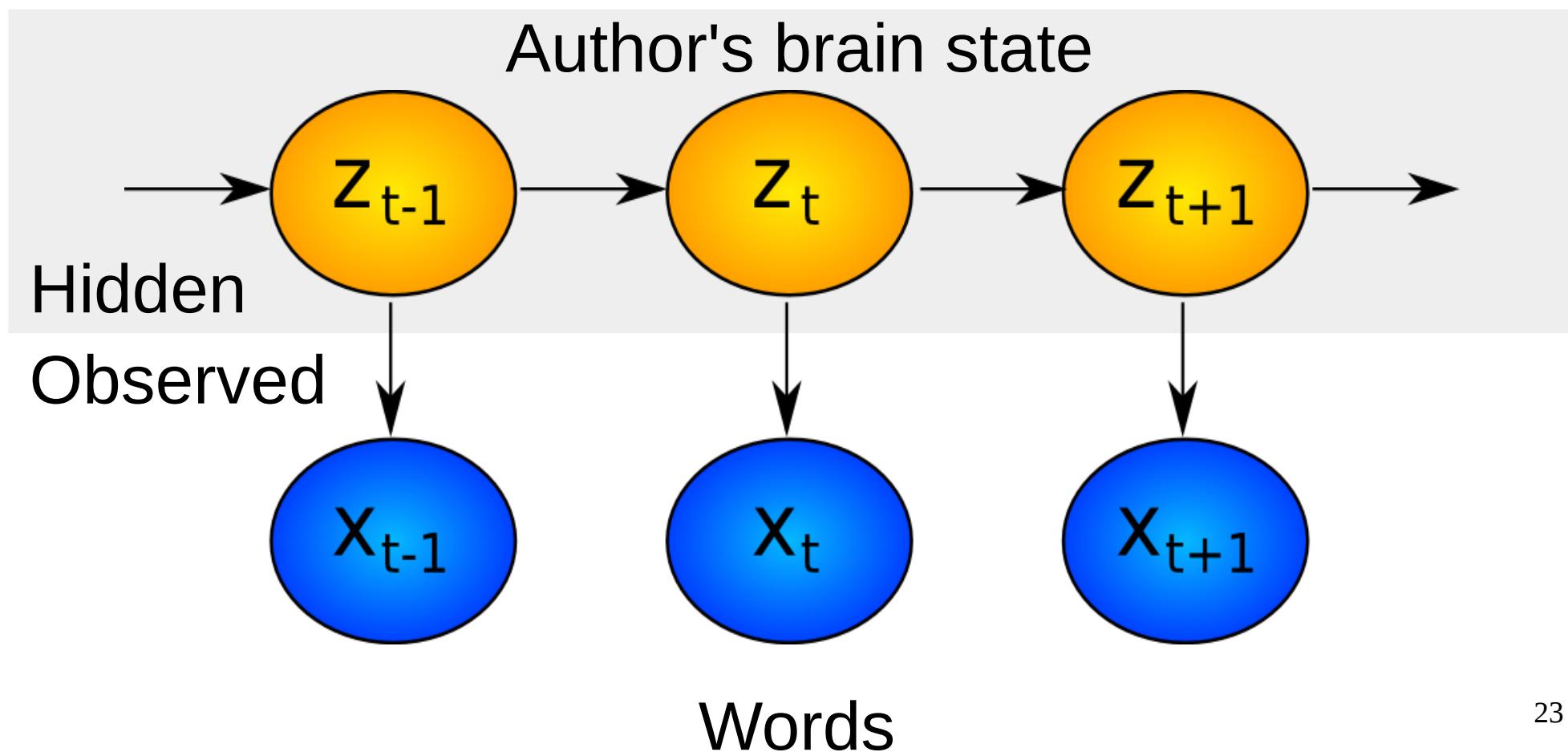
Can we learn* arbitrarily long dependencies?

* without infinitely many parameters

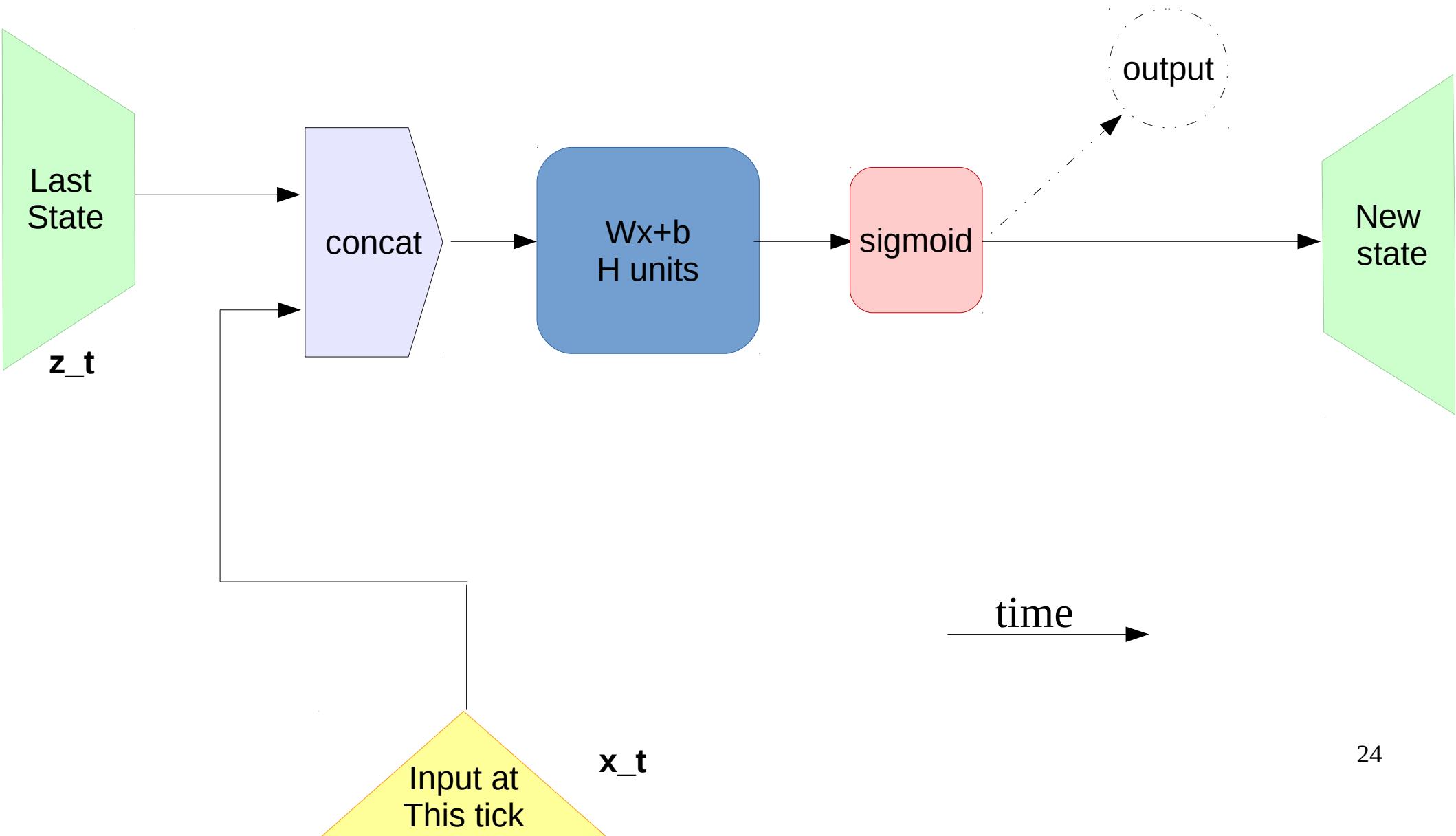
Hidden Markov Models: what's hidden



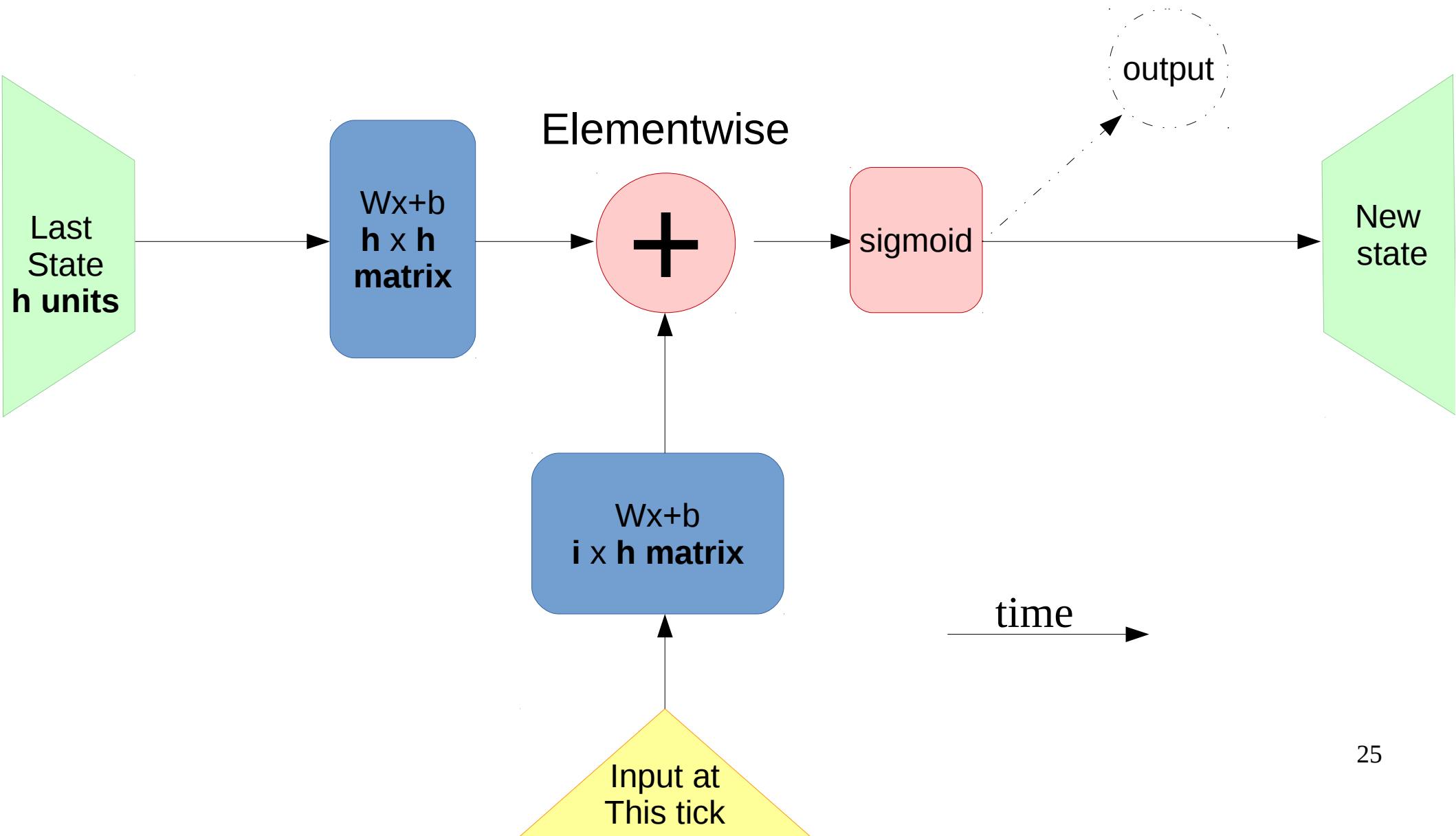
Hidden Markov Models: what is hidden



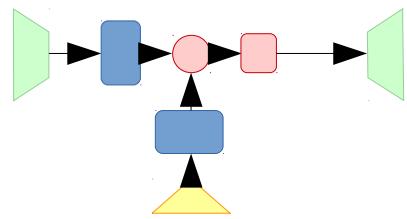
Recurrent neural network: one step



Recurrent neural network: one step

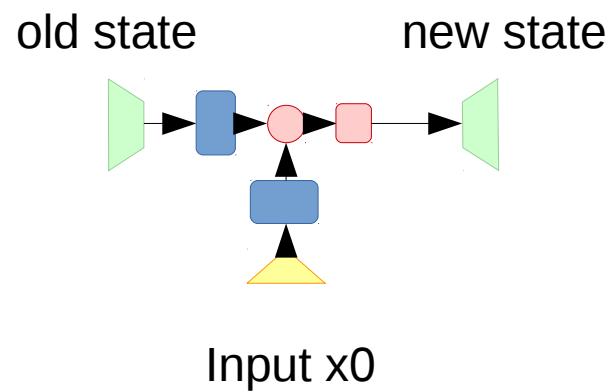


Recurrent neural network

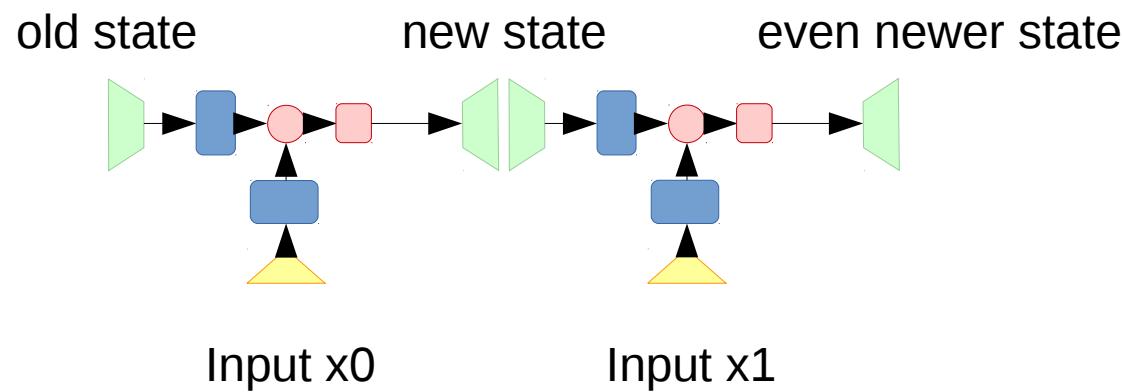


Zoom-out
of previous slide

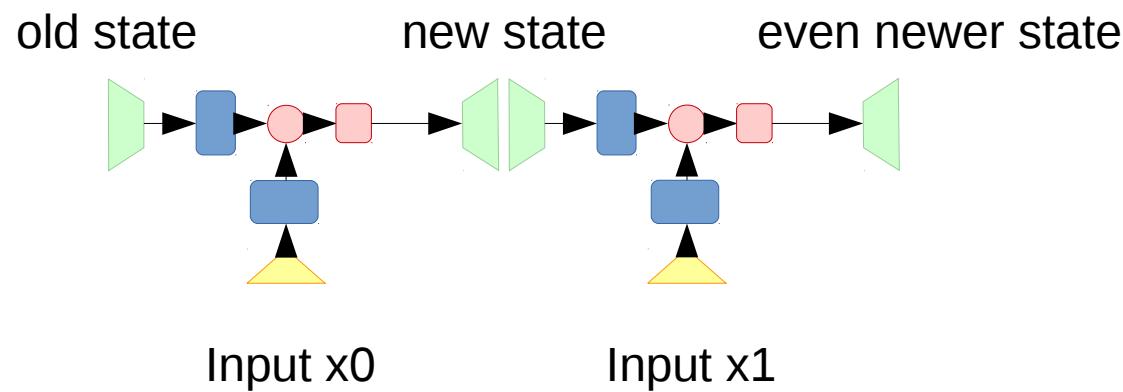
Recurrent neural network



Recurrent neural network

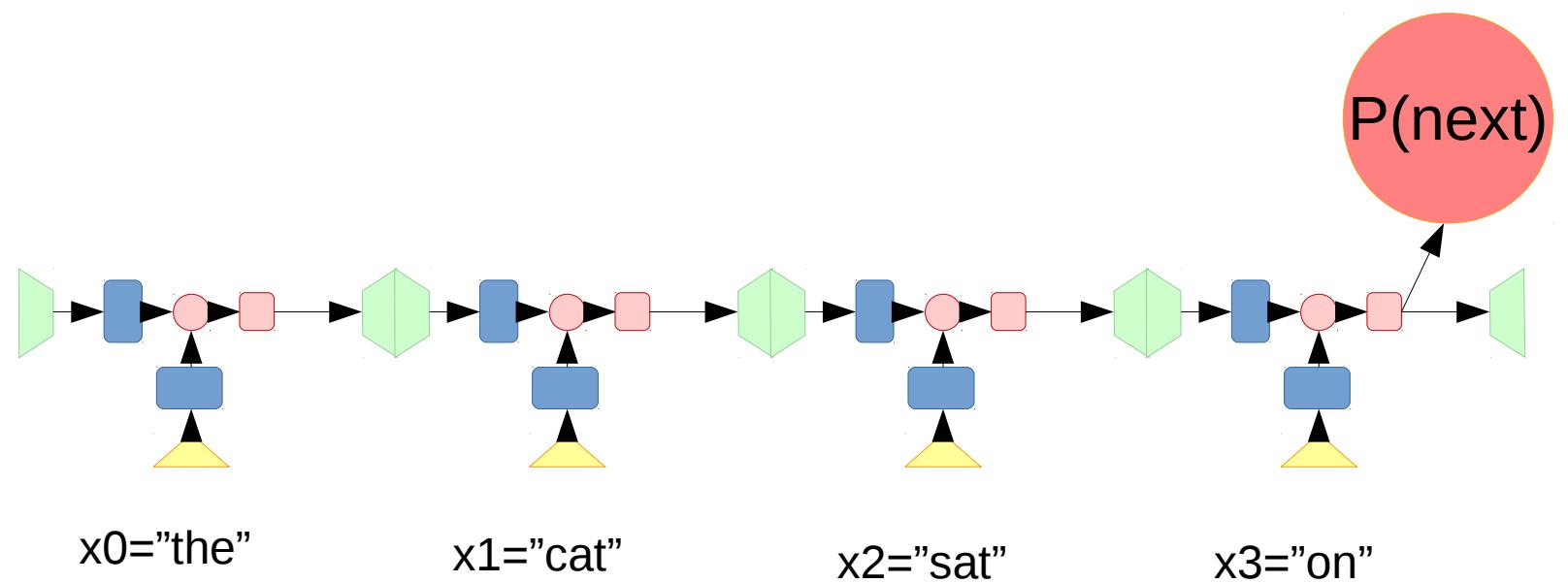


Recurrent neural network

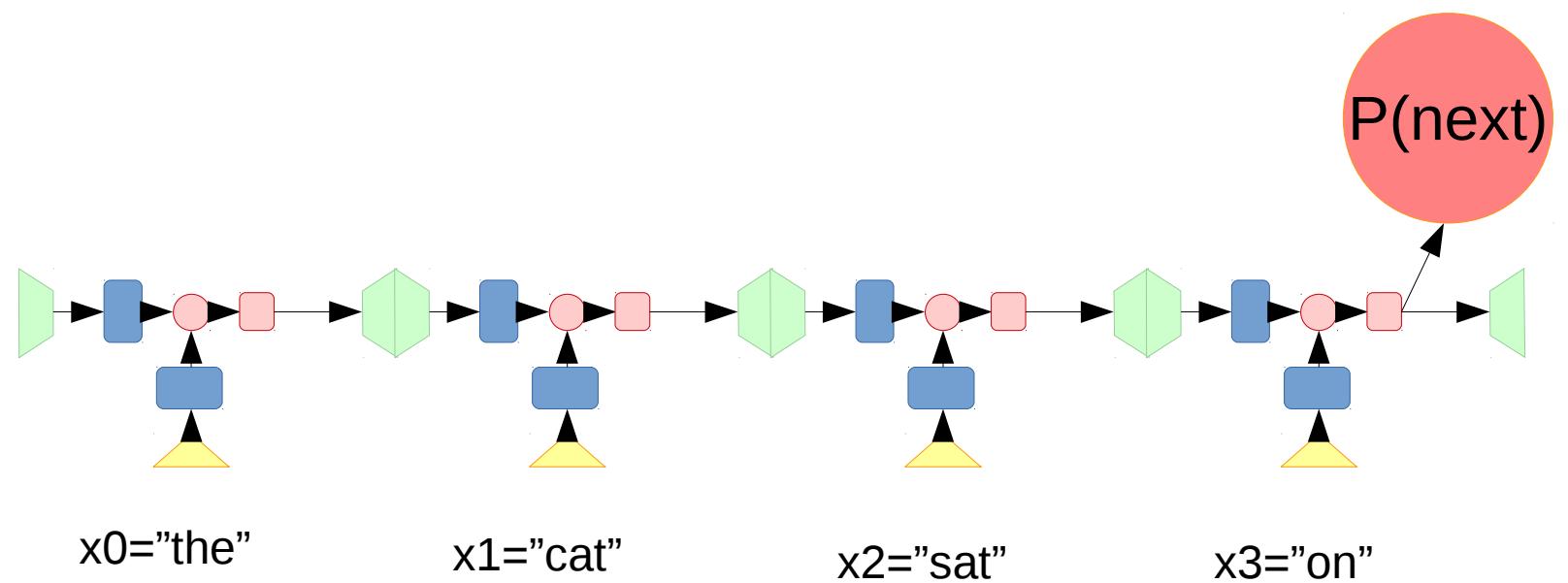


We use **same weight matrices** for all steps

Recurrent neural network

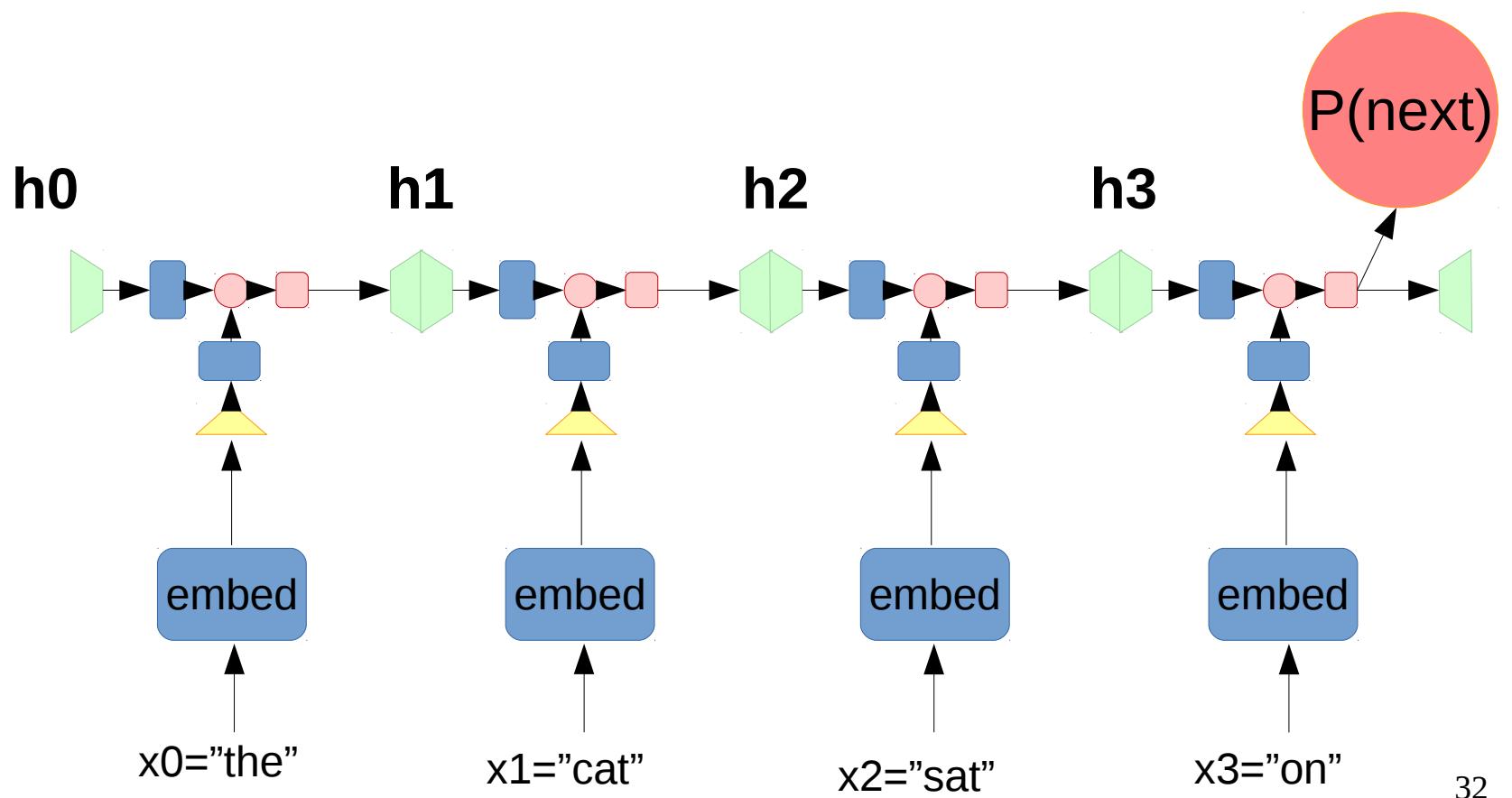


Recurrent neural network



How can we represent words?

Recurrent neural network

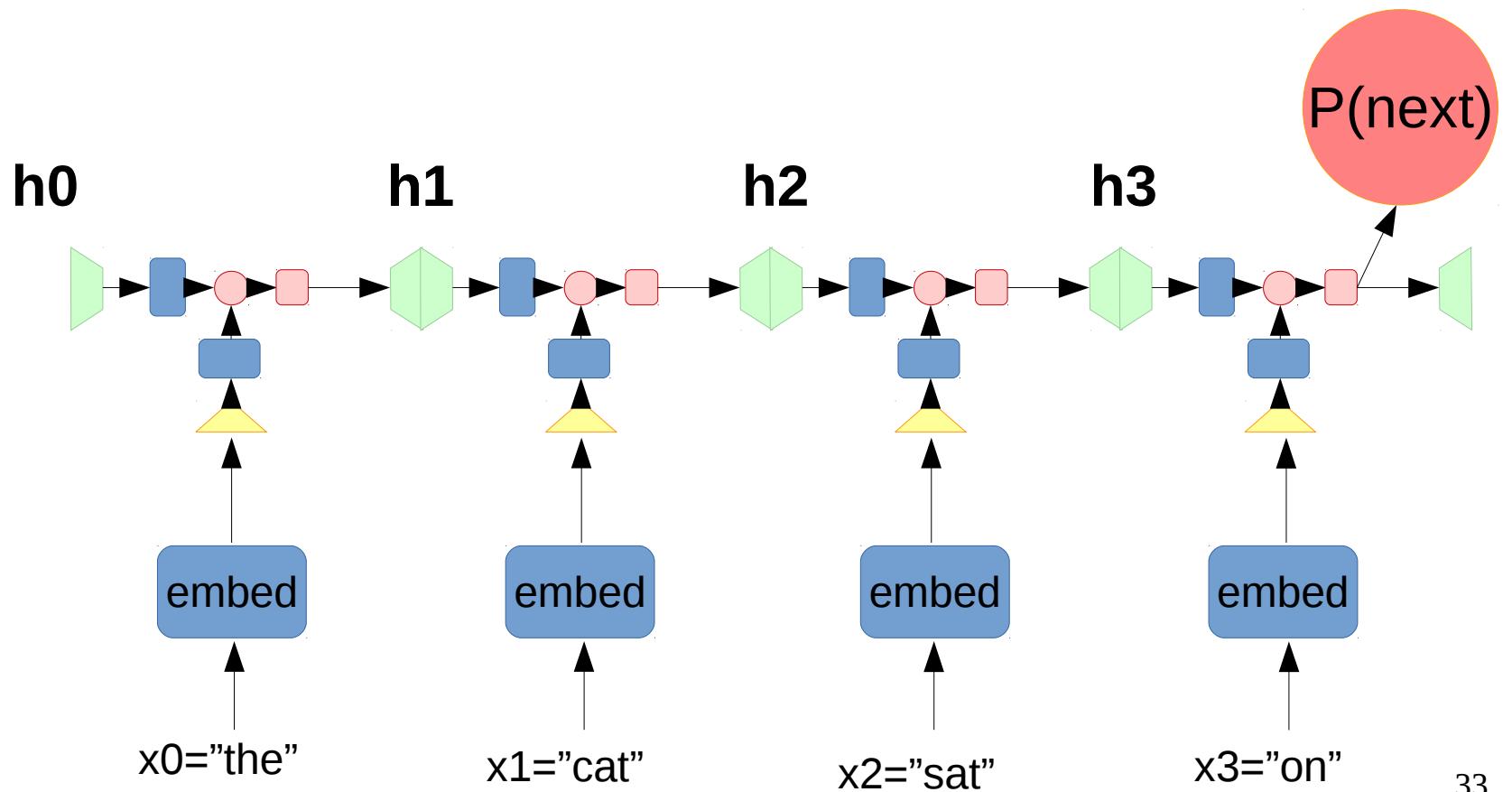


Recurrent neural network

$$h_0 = 0$$

$$h_1 = \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b)$$

$$h_2 = ?$$



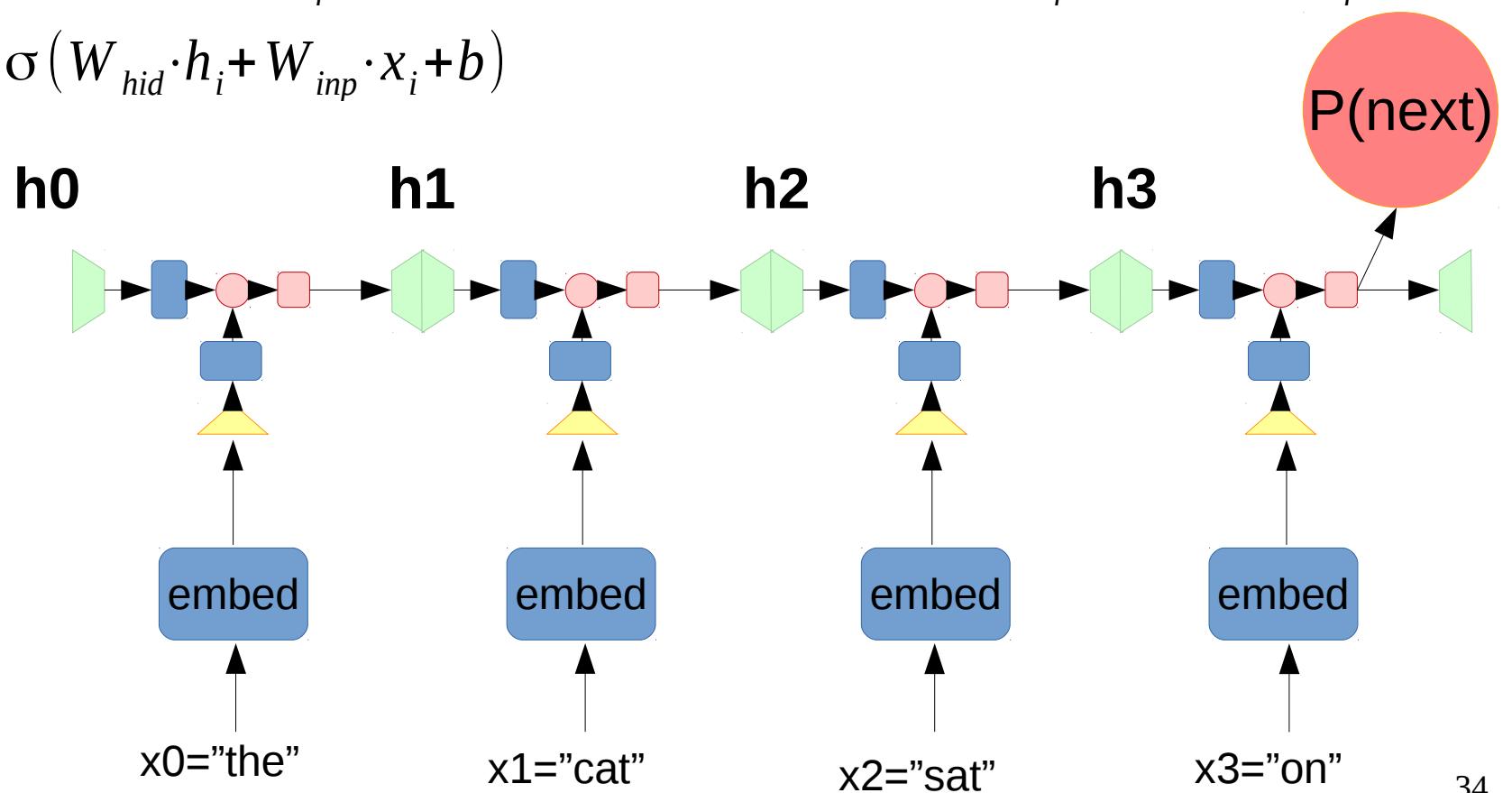
Recurrent neural network

$$h_0 = 0$$

$$h_1 = \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b)$$

$$h_2 = \sigma(W_{hid} \cdot h_1 + W_{inp} \cdot x_1 + b) = \sigma(W_{hid} \cdot \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b) + W_{inp} \cdot x_1 + b)$$

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$



Recurrent neural network

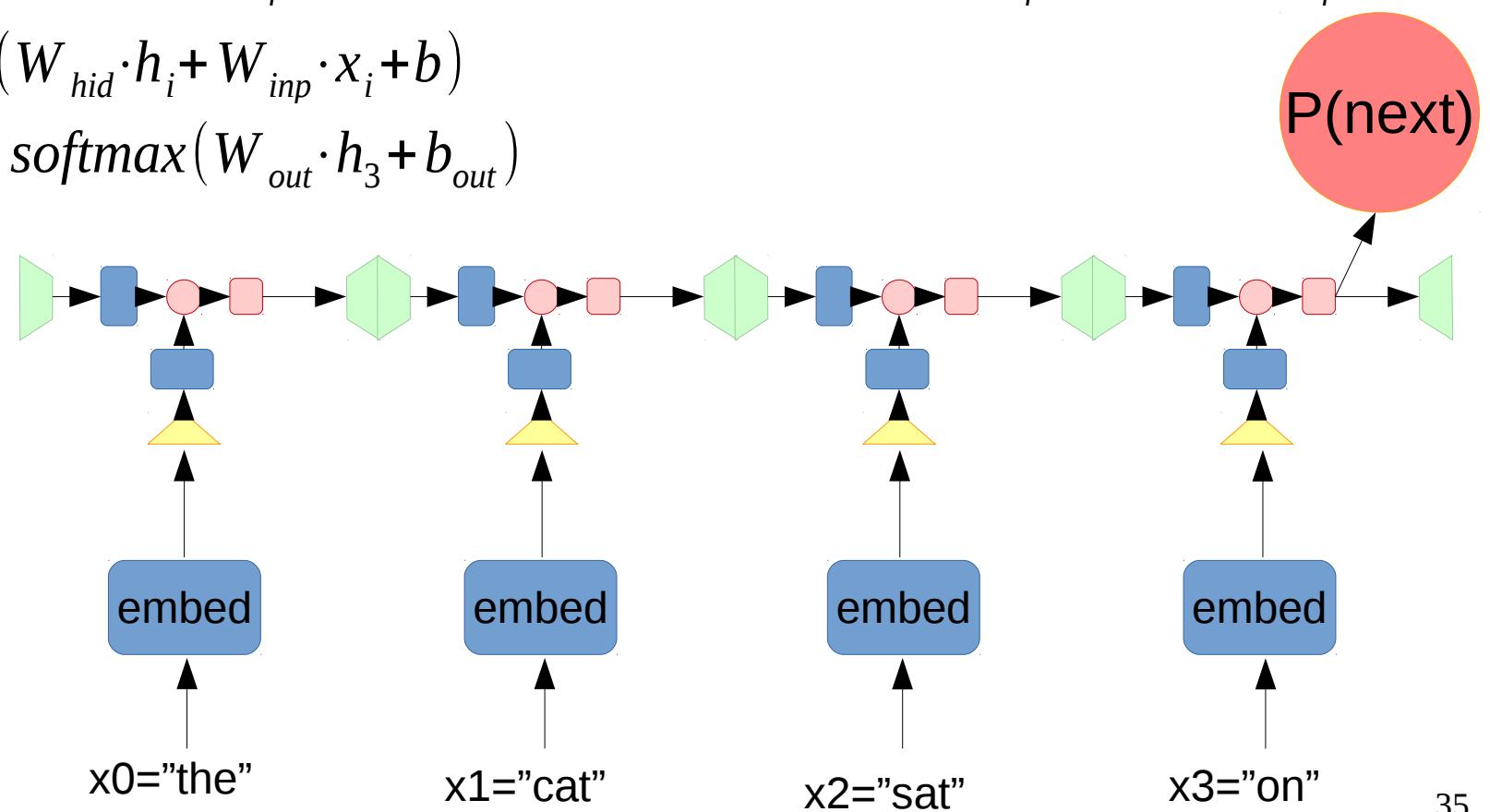
$$h_0 = 0$$

$$h_1 = \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b)$$

$$h_2 = \sigma(W_{hid} \cdot h_1 + W_{inp} \cdot x_1 + b) = \sigma(W_{hid} \cdot \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b) + W_{inp} \cdot x_1 + b)$$

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$P(x_4) = \text{softmax}(W_{out} \cdot h_3 + b_{out})$$

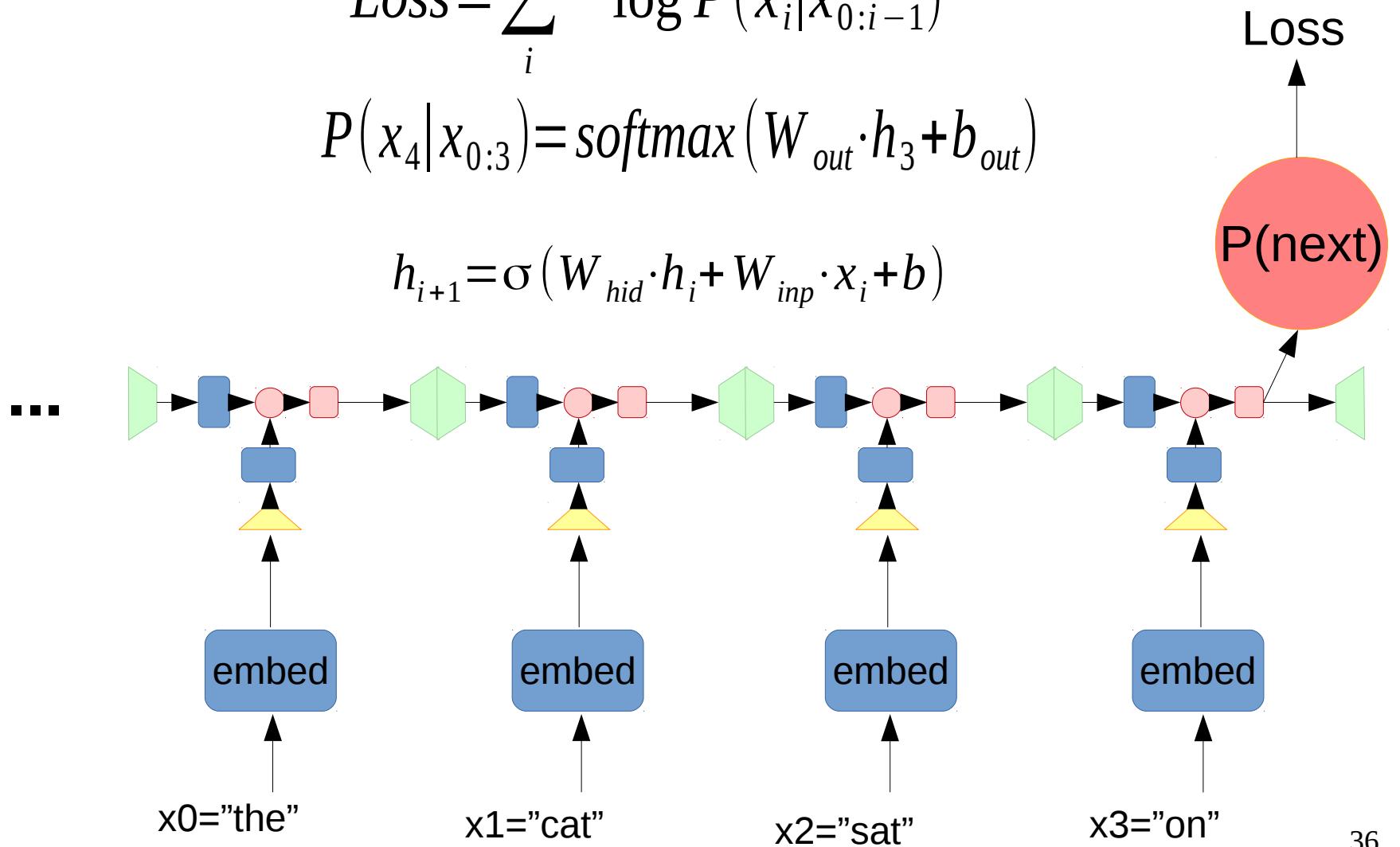


Recurrent neural network

$$Loss = \sum_i -\log P(x_i | x_{0:i-1})$$

$$P(x_4 | x_{0:3}) = \text{softmax}(W_{out} \cdot h_3 + b_{out})$$

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$



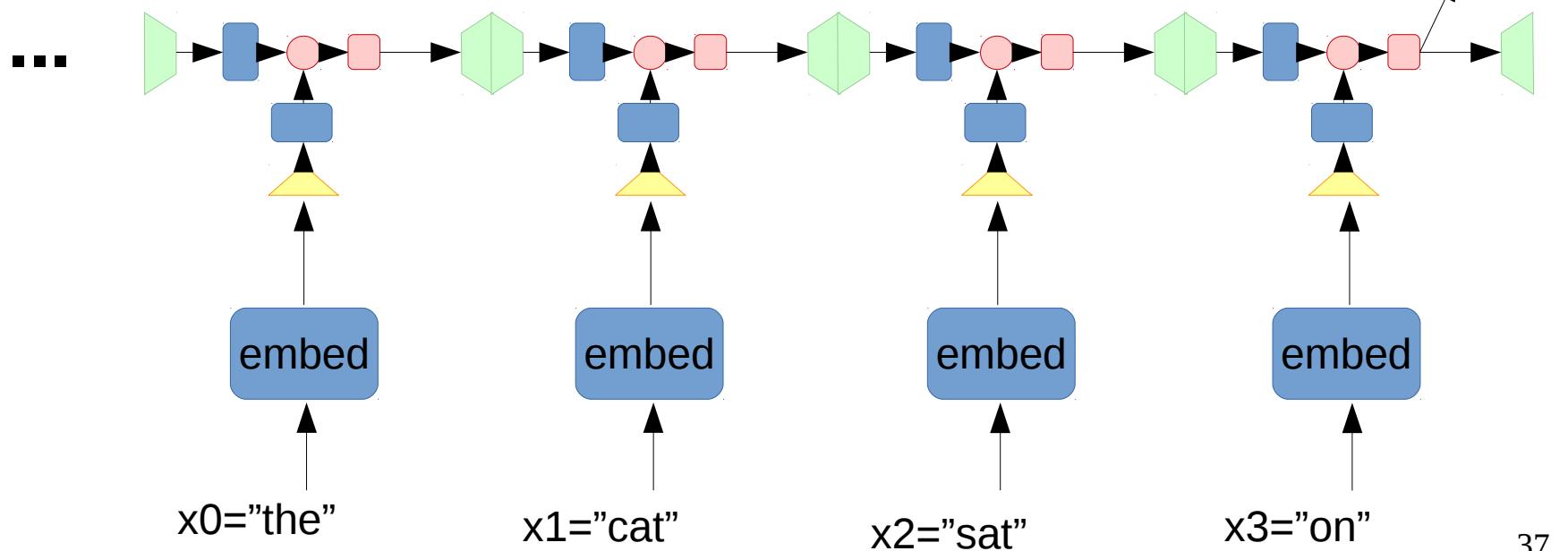
Recurrent neural network

How do we train it?

$$Loss = \sum_i -\log P(x_{i+1} | x_{0:i})$$

$$P(x_4 | x_{0:3}) = \text{softmax}(W_{out} \cdot h_3 + b_{out})$$

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$



WHAT ARE WE DOING TODAY,
RAIN BRAIN?

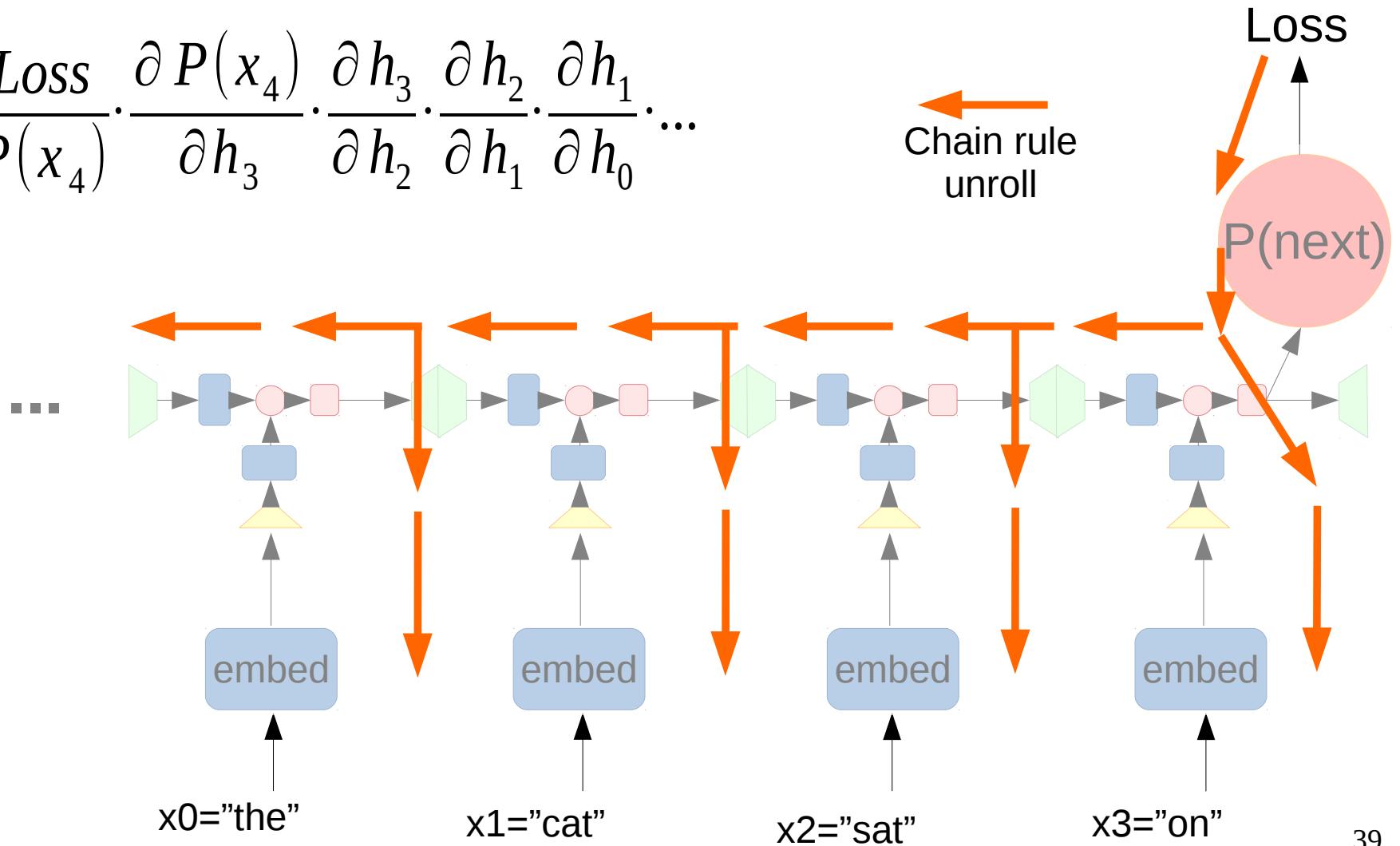


THE SAME THING WE DO EVERY DAY, PINKY.
BACKPROPAGATE

memegenerator.net

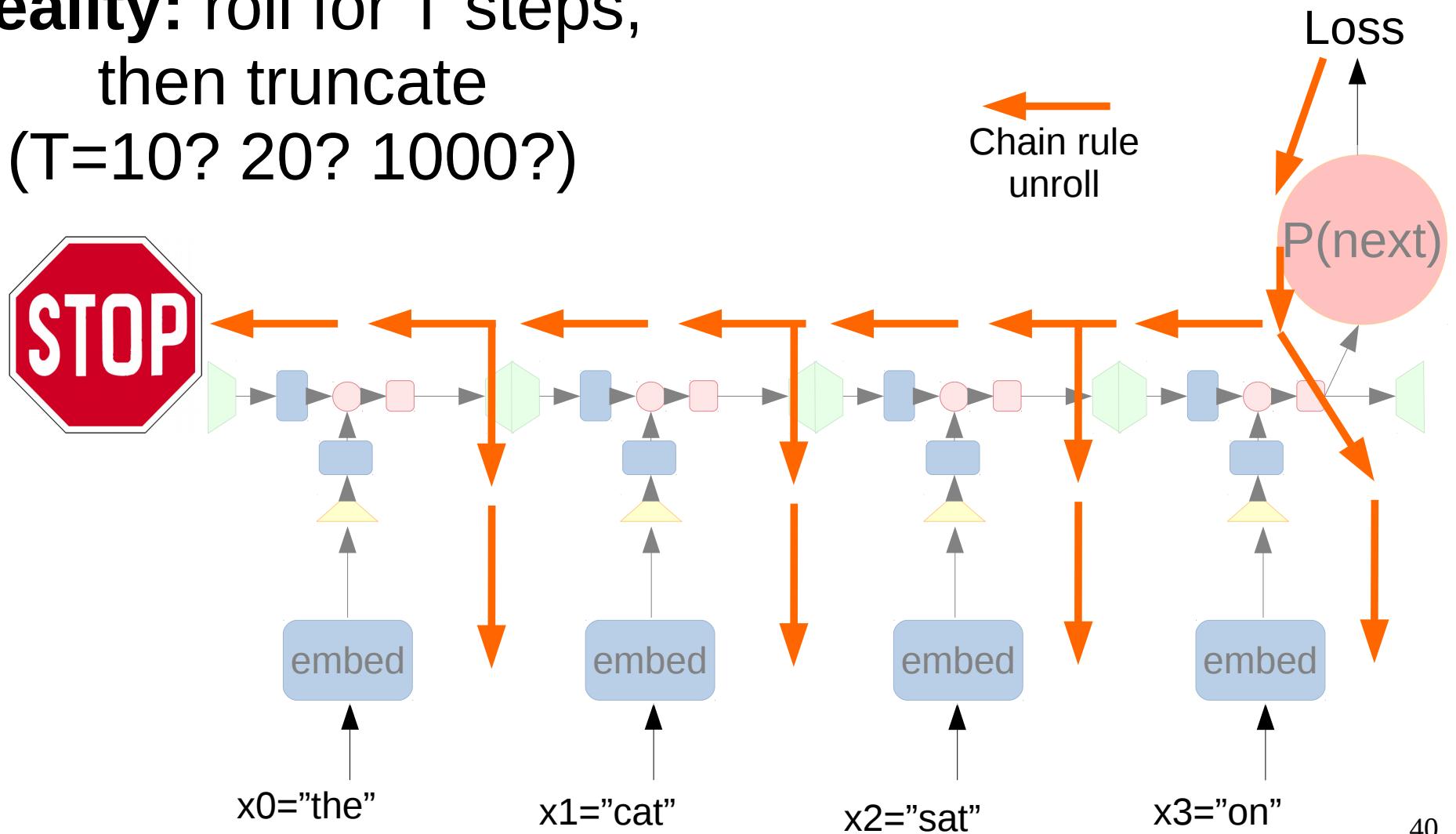
Backpropagation through time

$$\frac{\partial \text{Loss}}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0} \cdot \dots$$



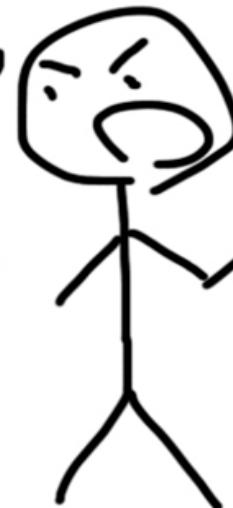
Truncated BPTT

Reality: roll for T steps,
then truncate
($T=10?$ $20?$ $1000?$)



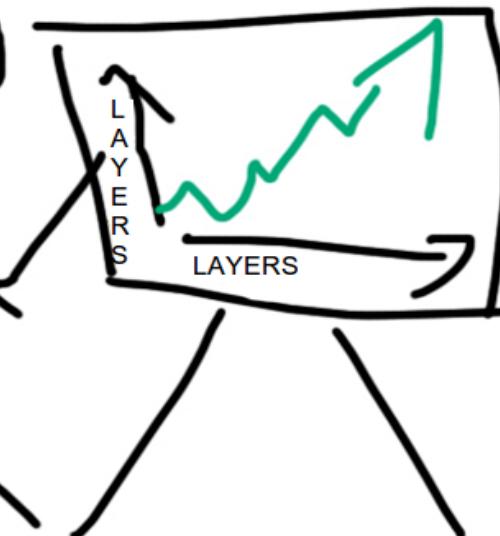
STATISTICAL LEARNING

Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high. Get some experts and minimize the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin

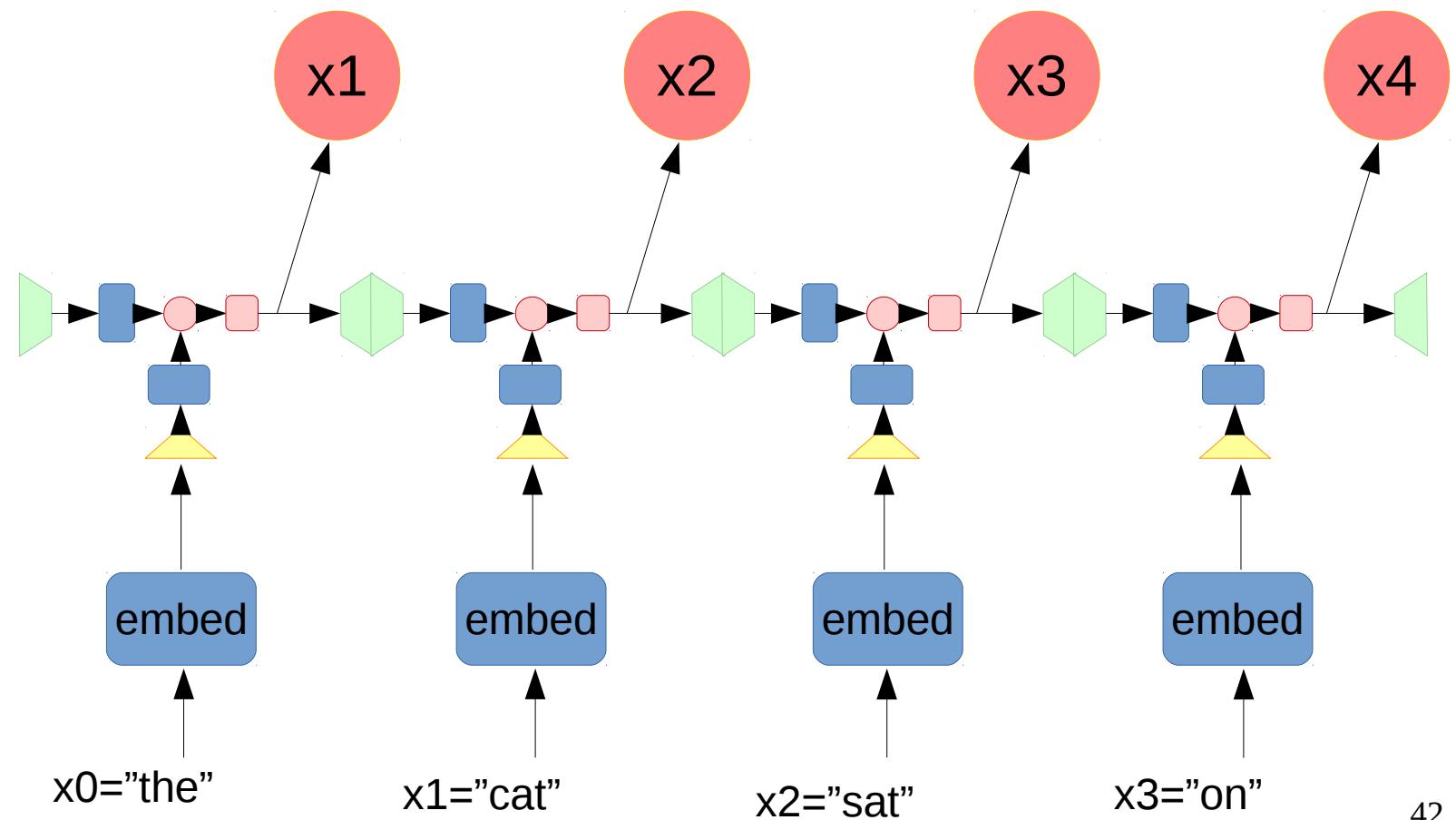


NEURAL NETWORKS

STACK MORE LAYERS

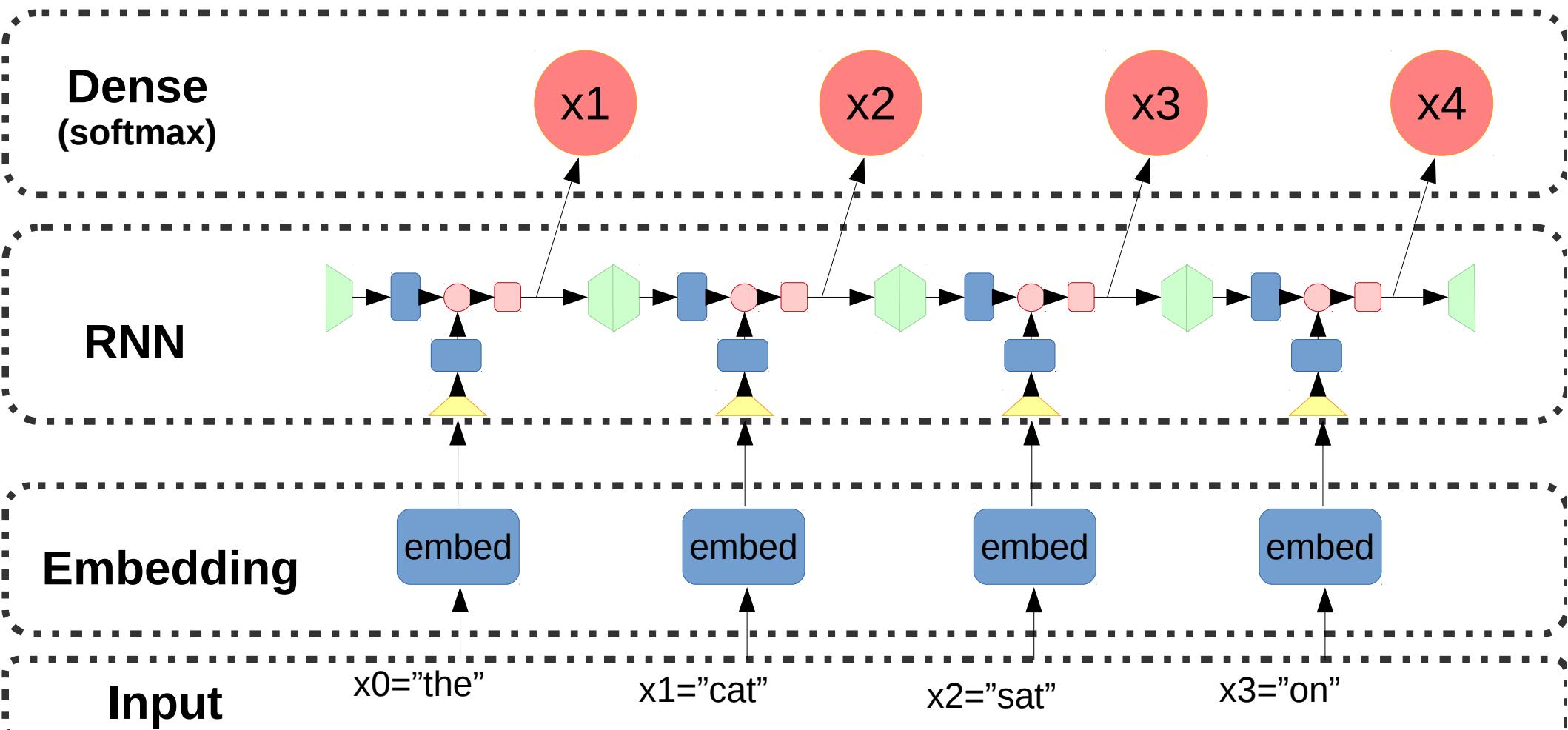


What is layer, again?

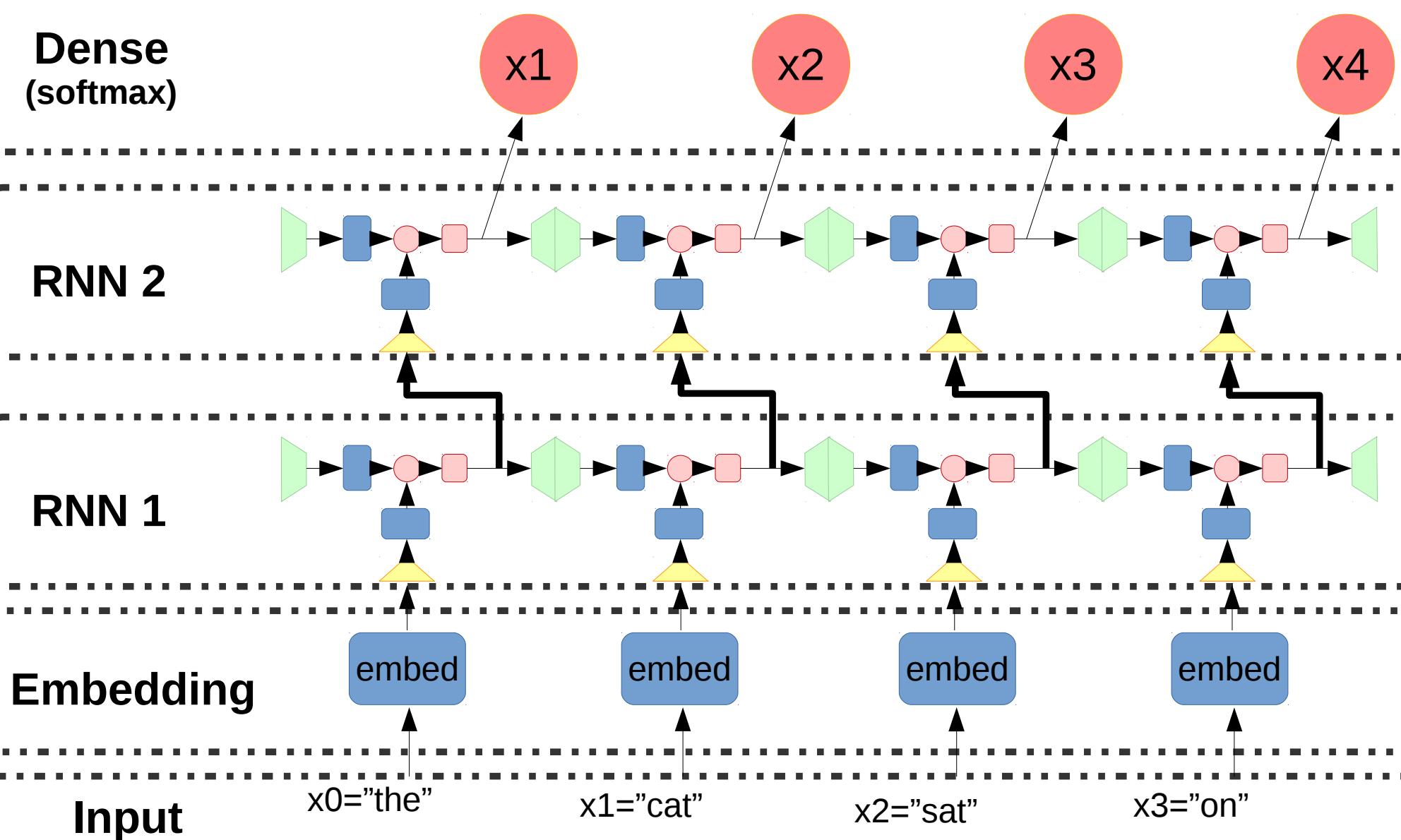


Layers

Where to stick more layers?



More layers



Too f**king complicated

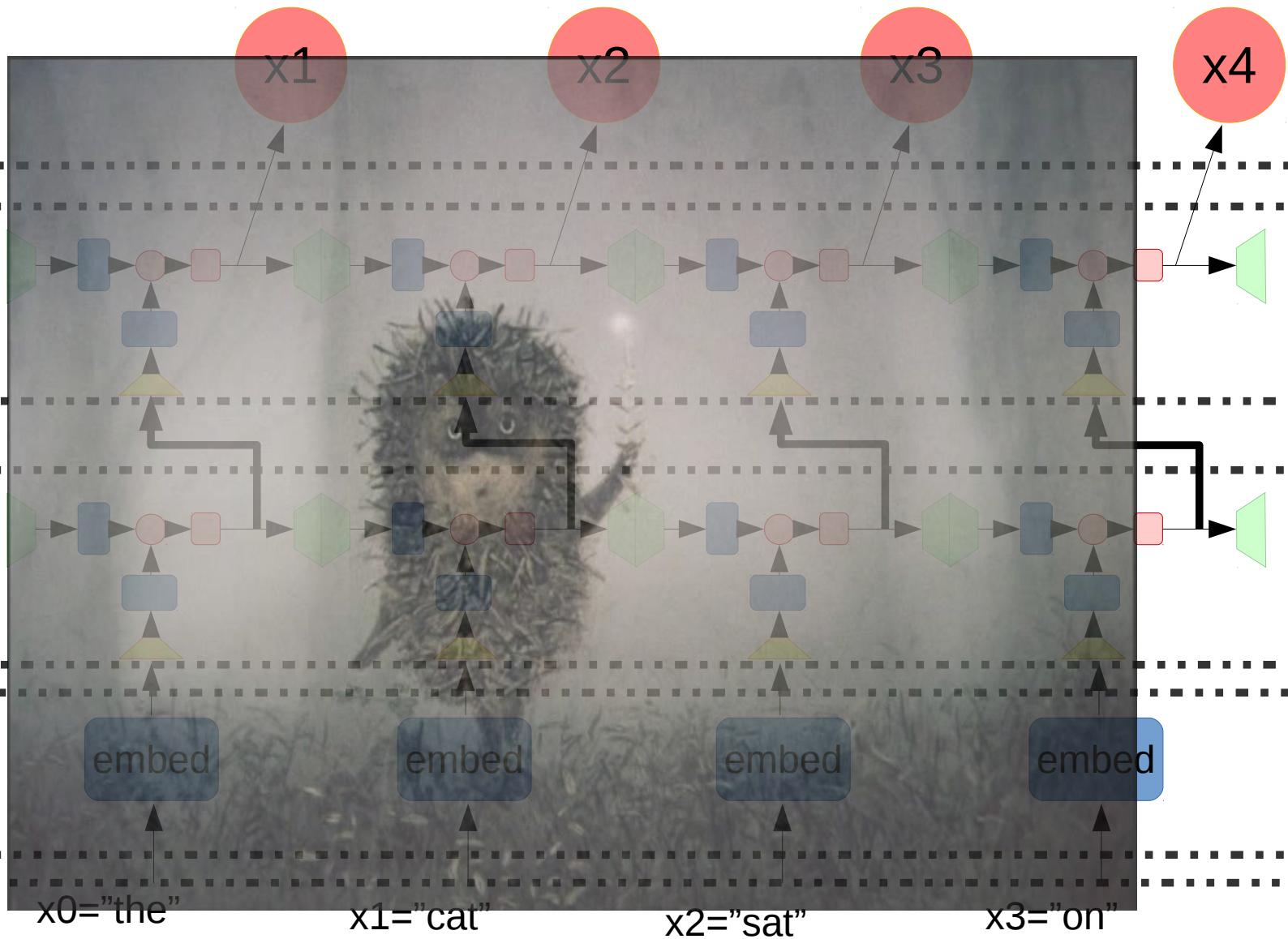
**Dense
(softmax)**

RNN 2

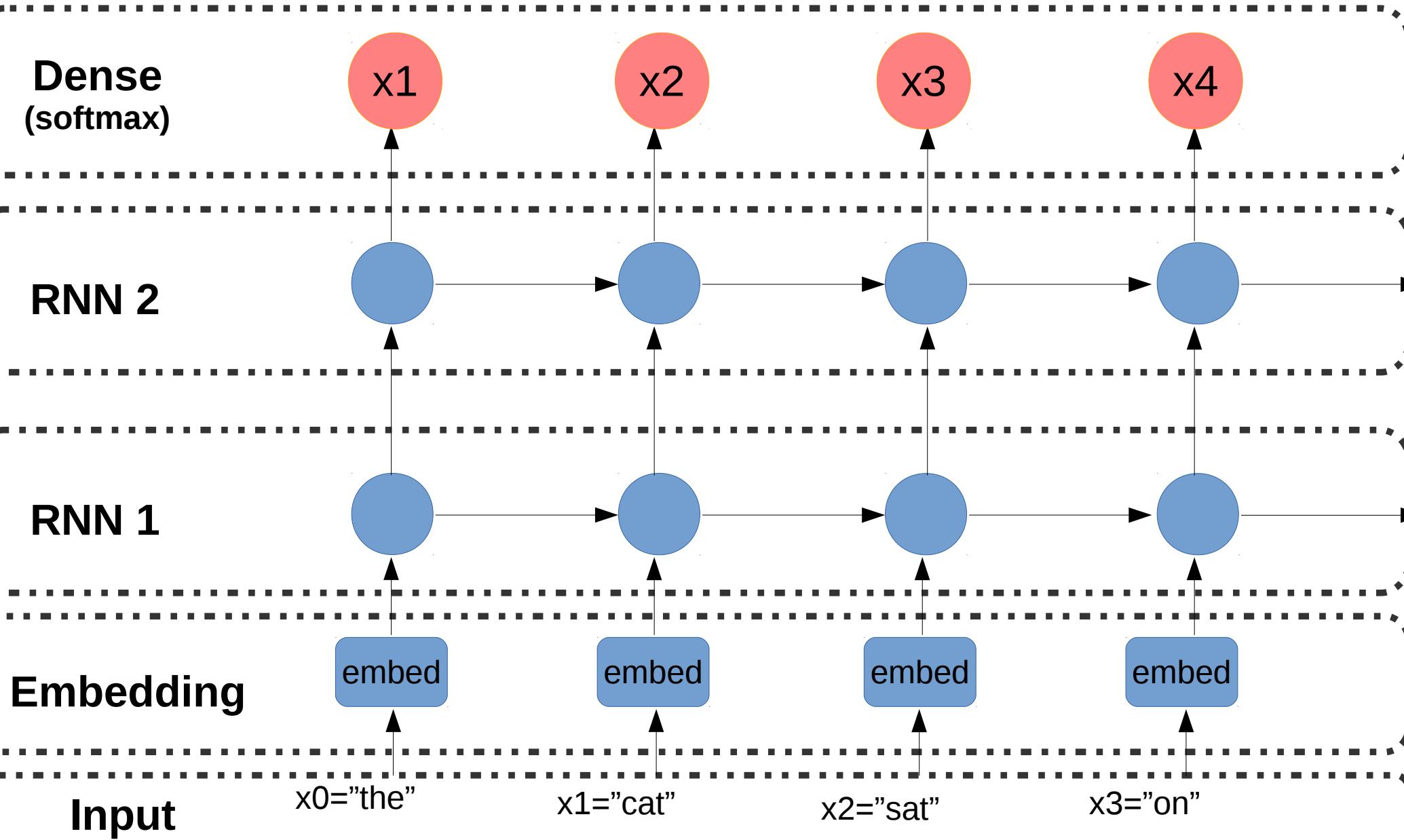
RNN 1

Embedding

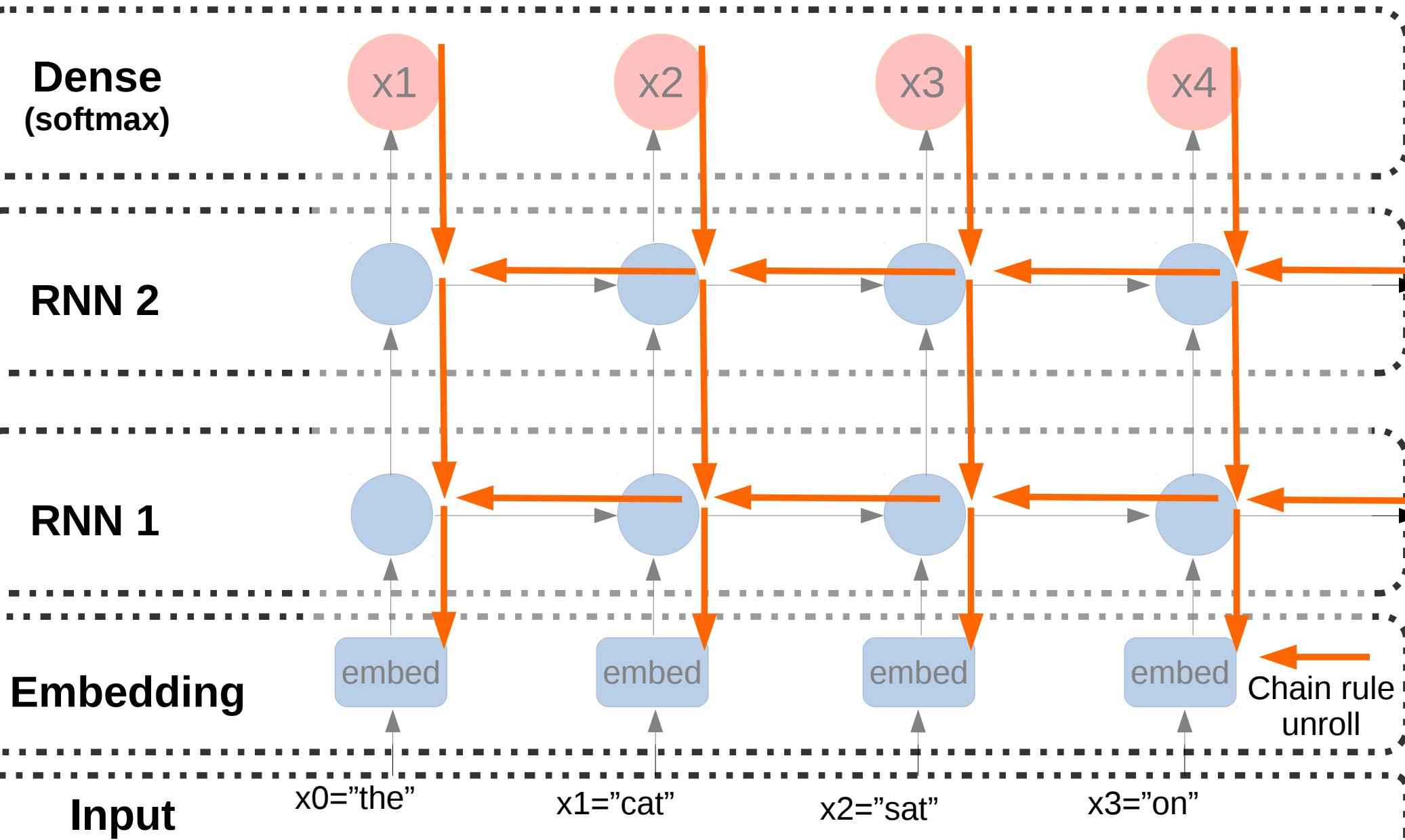
Input



2-layer RNN



BPTT again

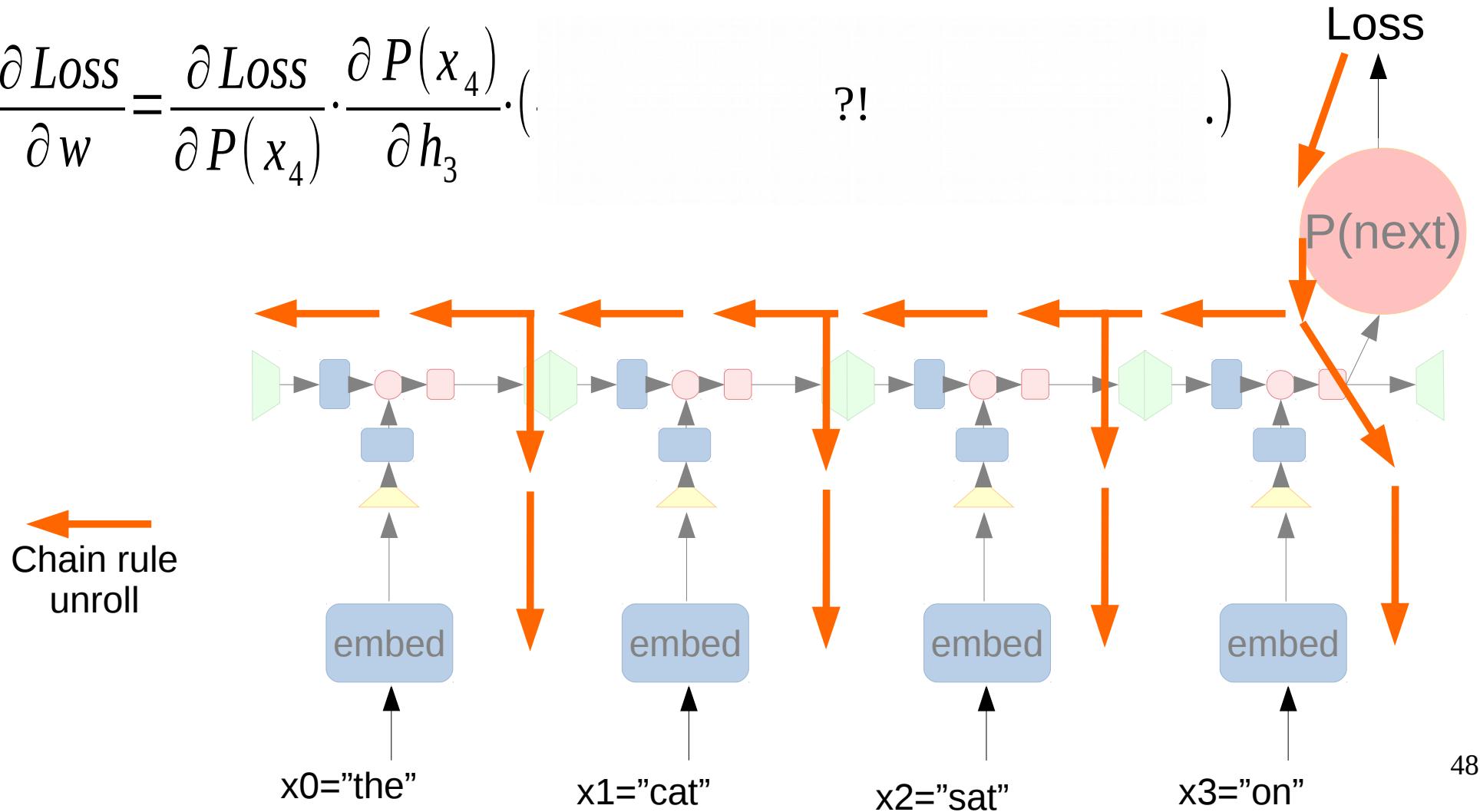


BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot ($$

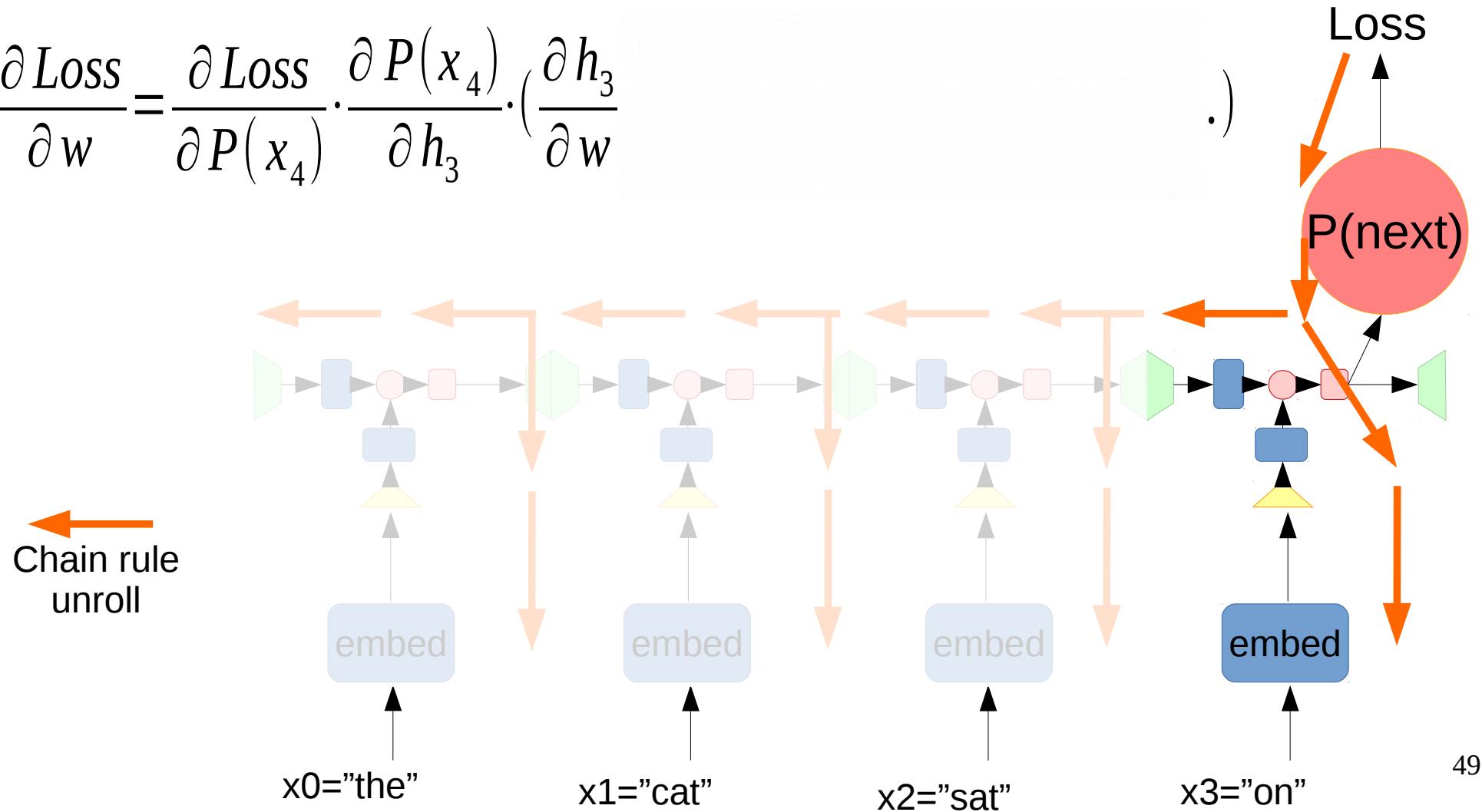
?)



BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} \right)$$

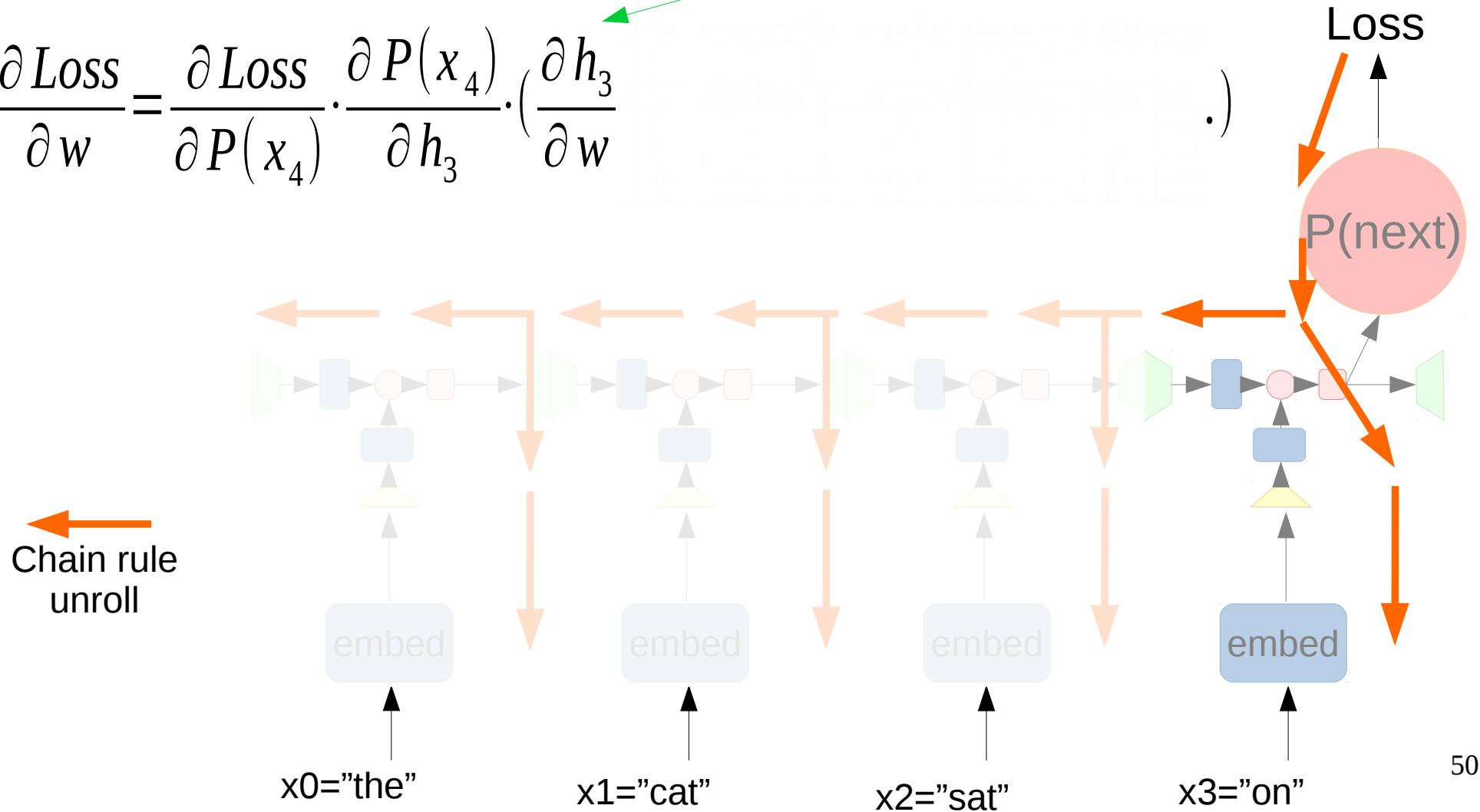


BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

consider h2 constant

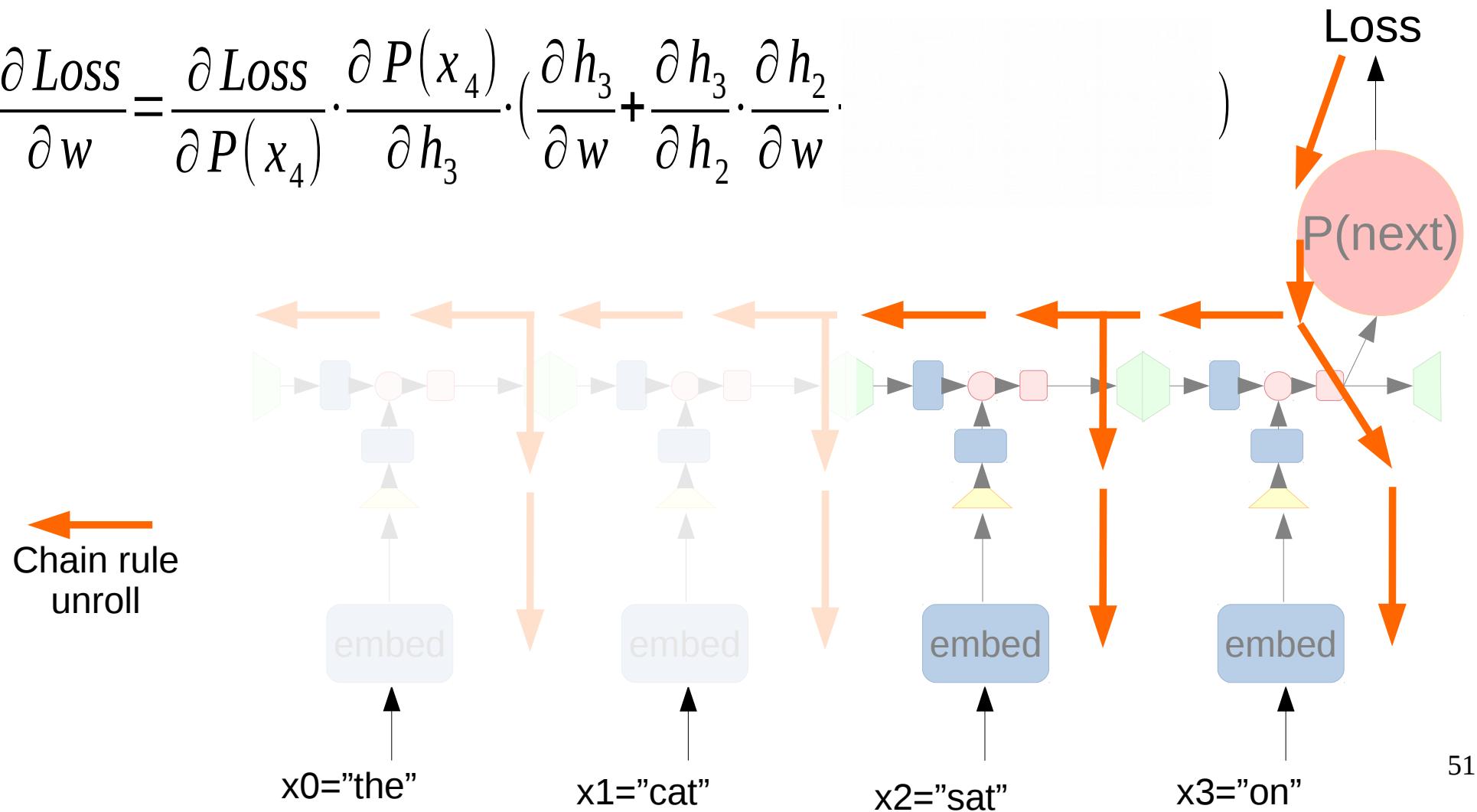
$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} \right)$$



BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} \right)$$

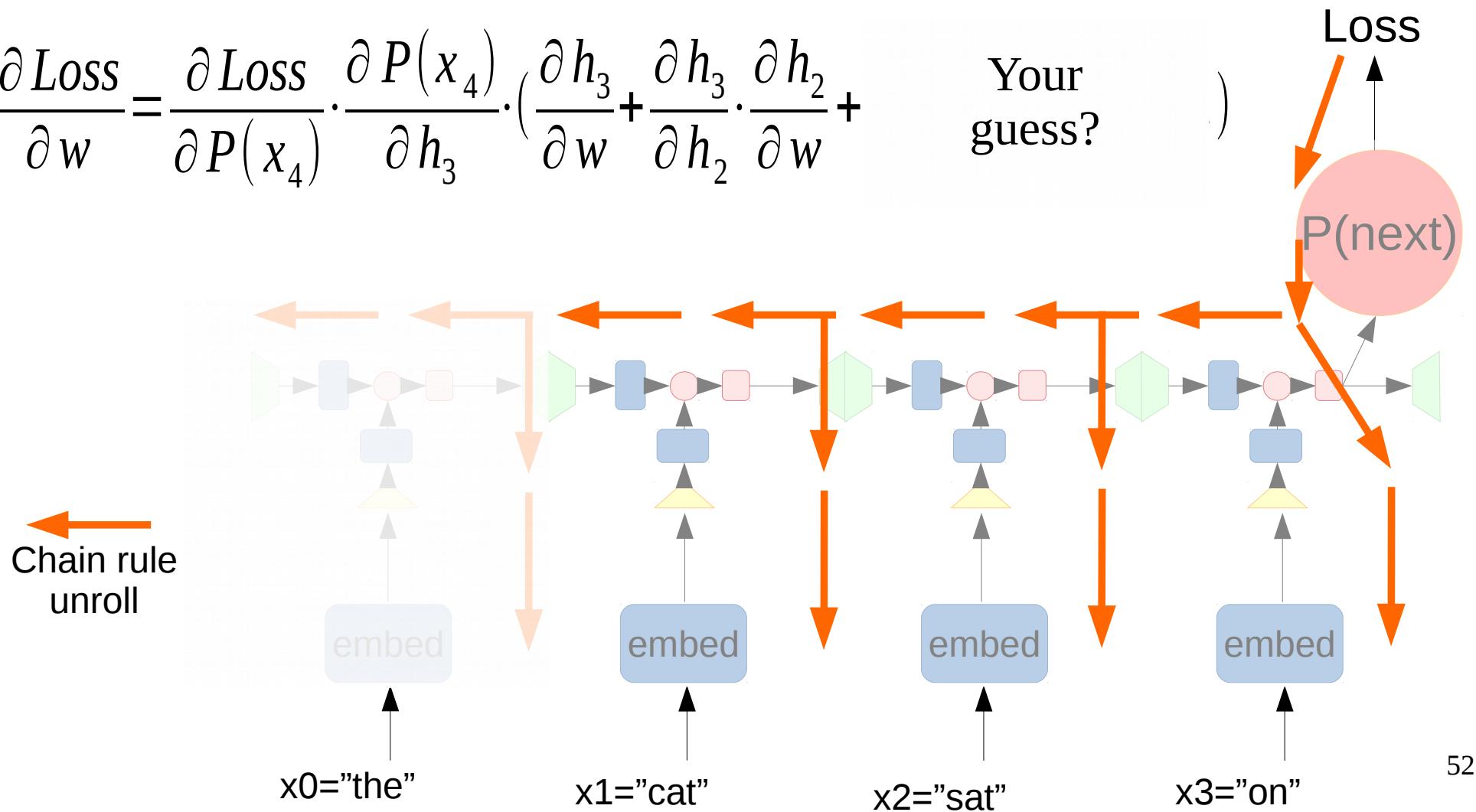


BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} + \dots \right)$$

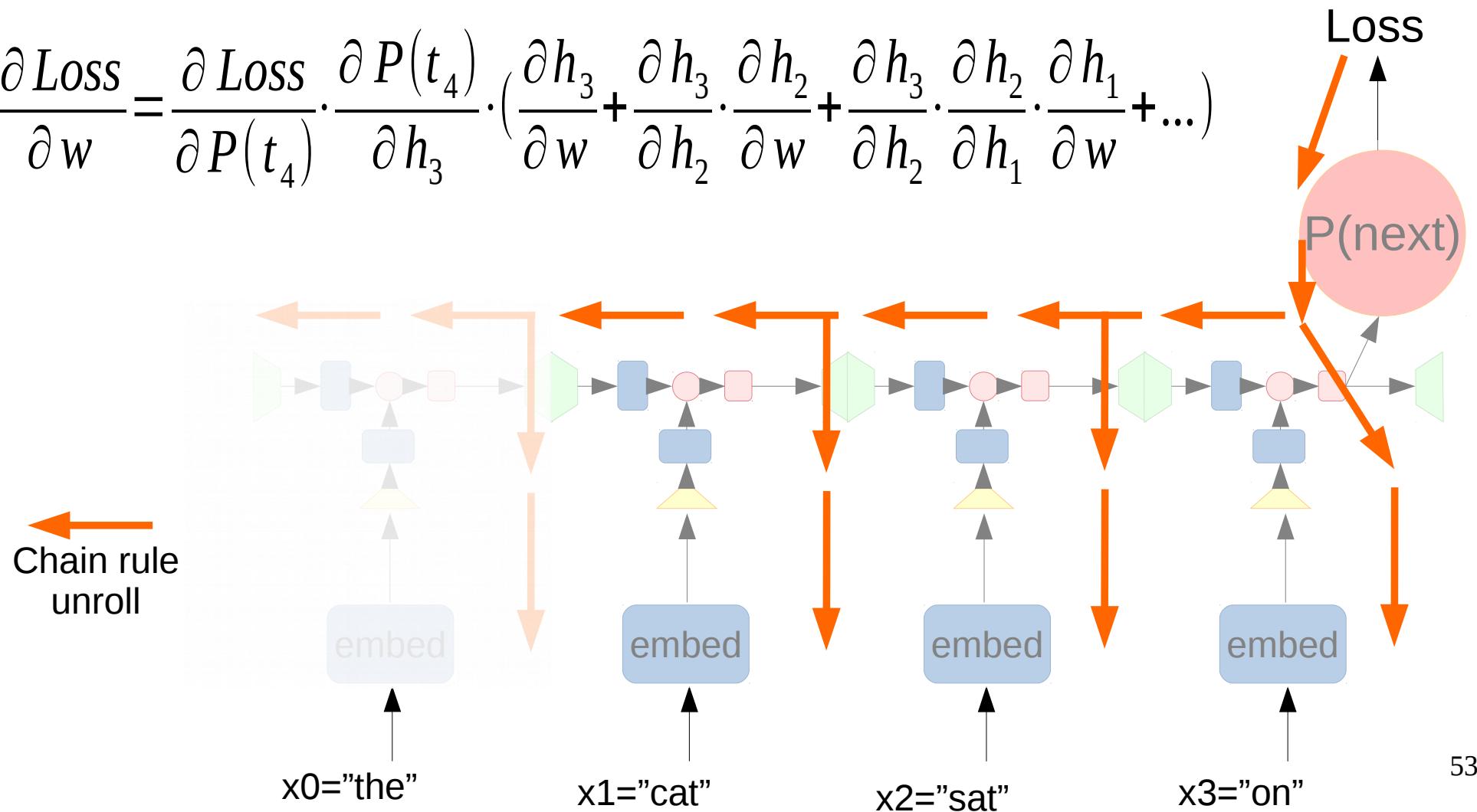
Your
guess?



BPTT Again

$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(t_4)} \cdot \frac{\partial P(t_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial w} + \dots \right)$$

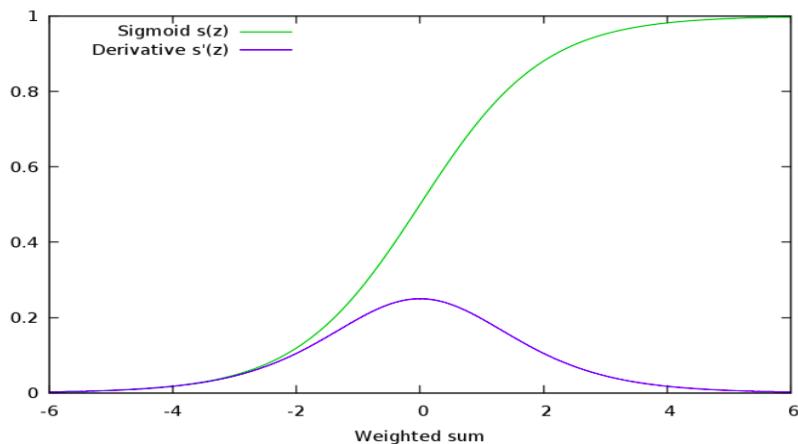


Gradient explosion and vanishing

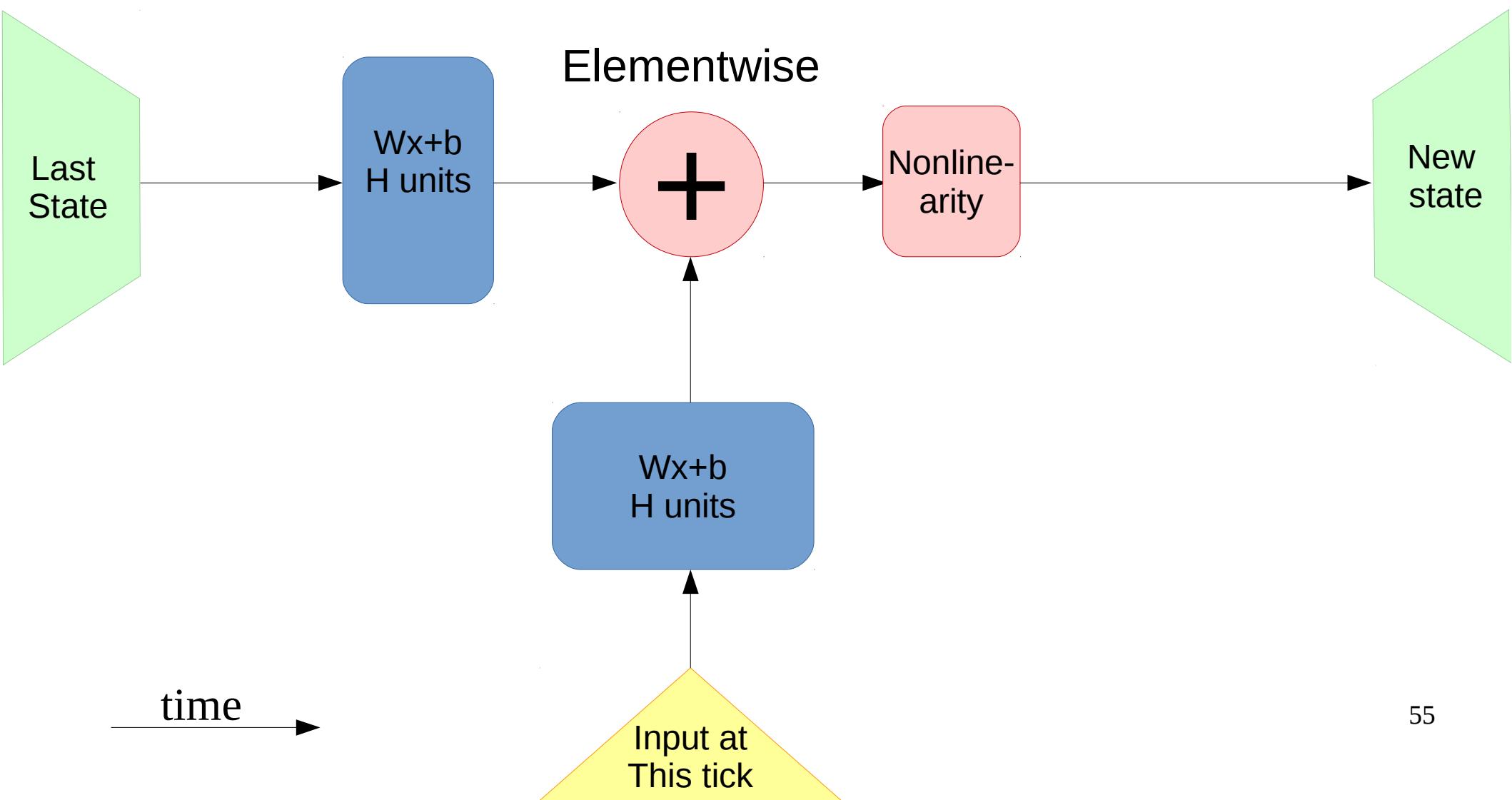
$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$

$$\frac{\partial \text{Loss}}{\partial w} = \frac{\partial \text{Loss}}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial w} + \dots \right)$$

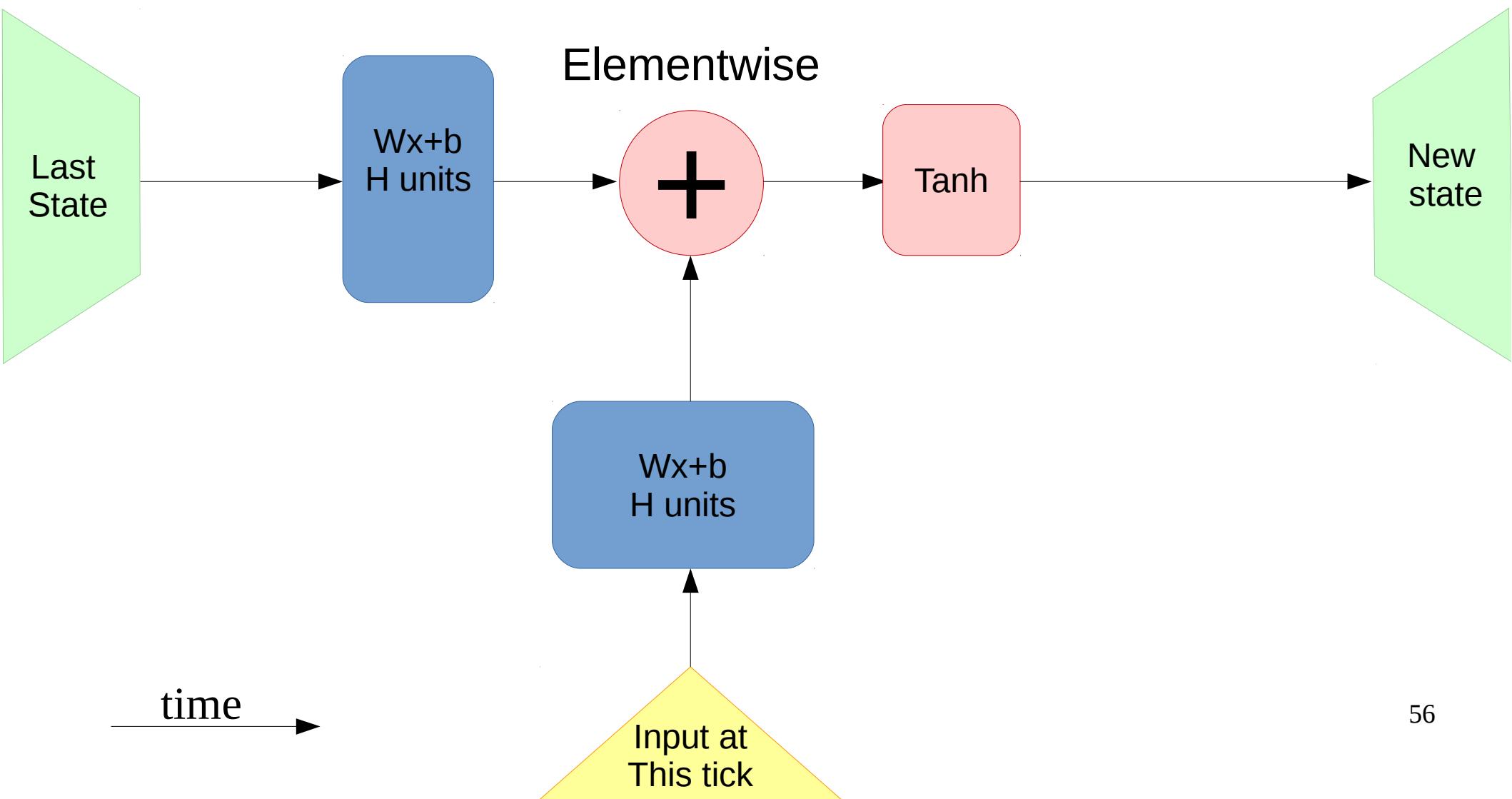
- Many sigmoids near 0 or 1
 - Gradients $\rightarrow 0$
 - Not training for long-term dependencies
- Many nonzero values
 - Derivative stacks to >1
 - Gradients $\rightarrow \infty$
 - Weights \rightarrow shit



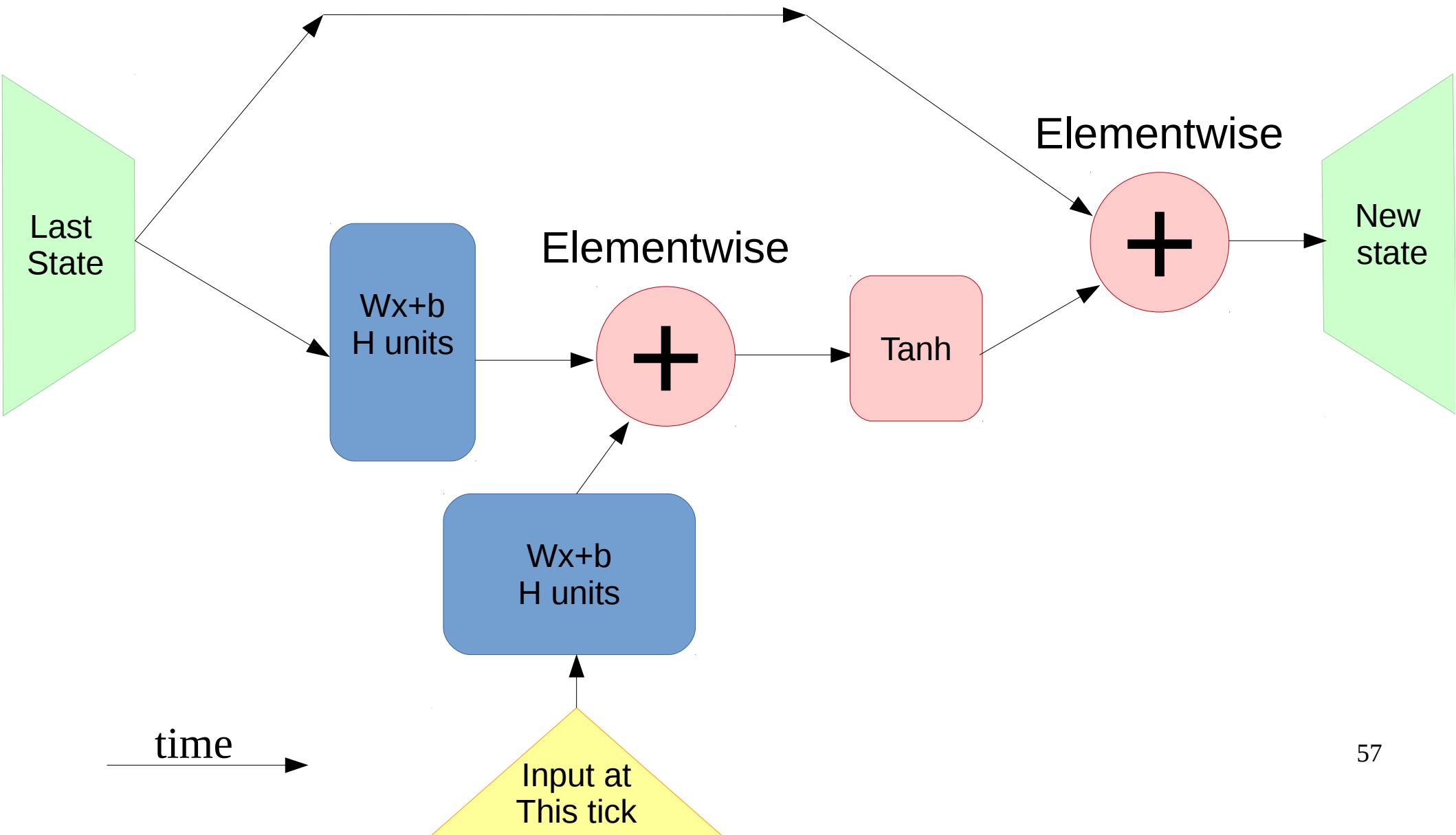
RNN step



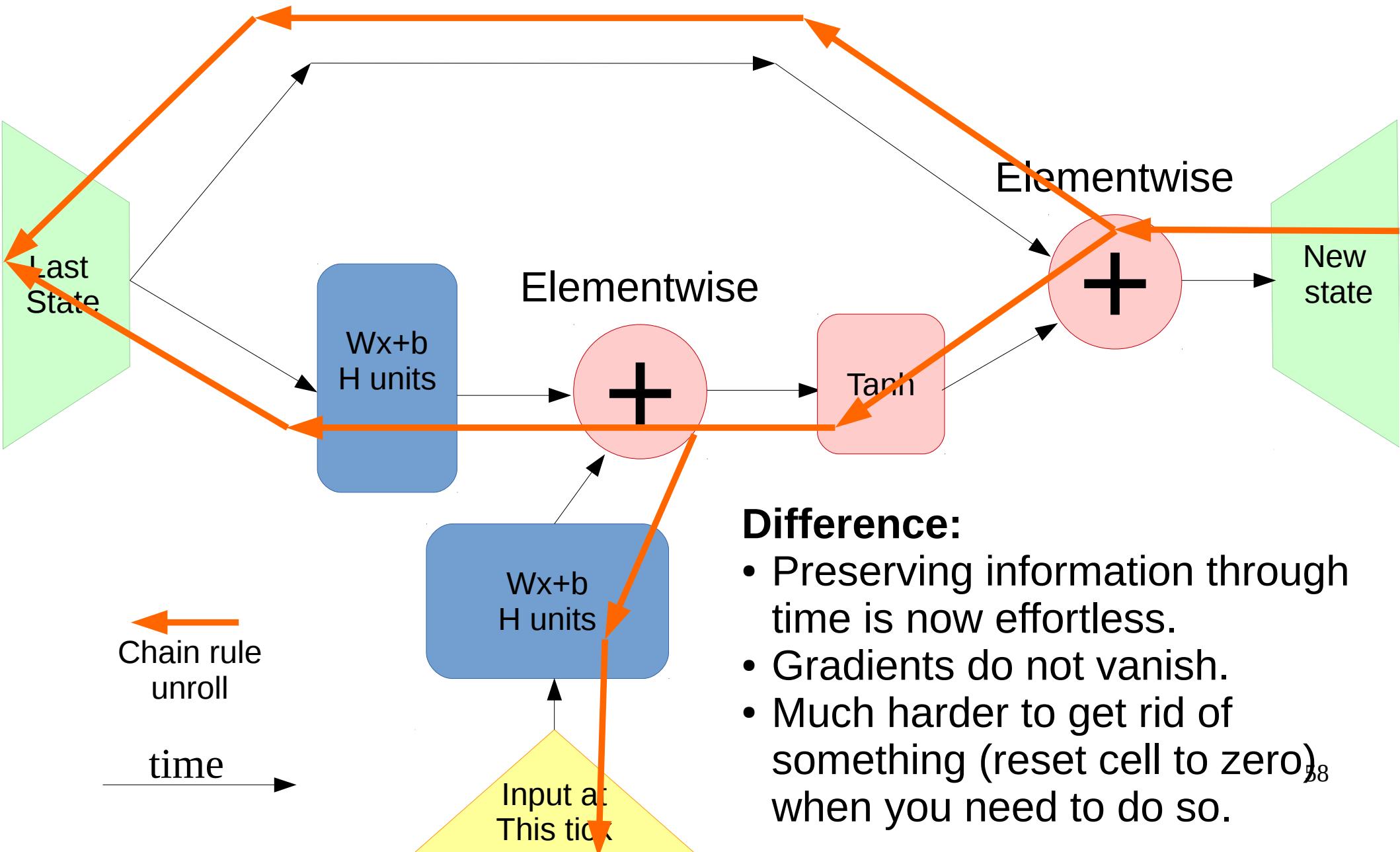
RNN step



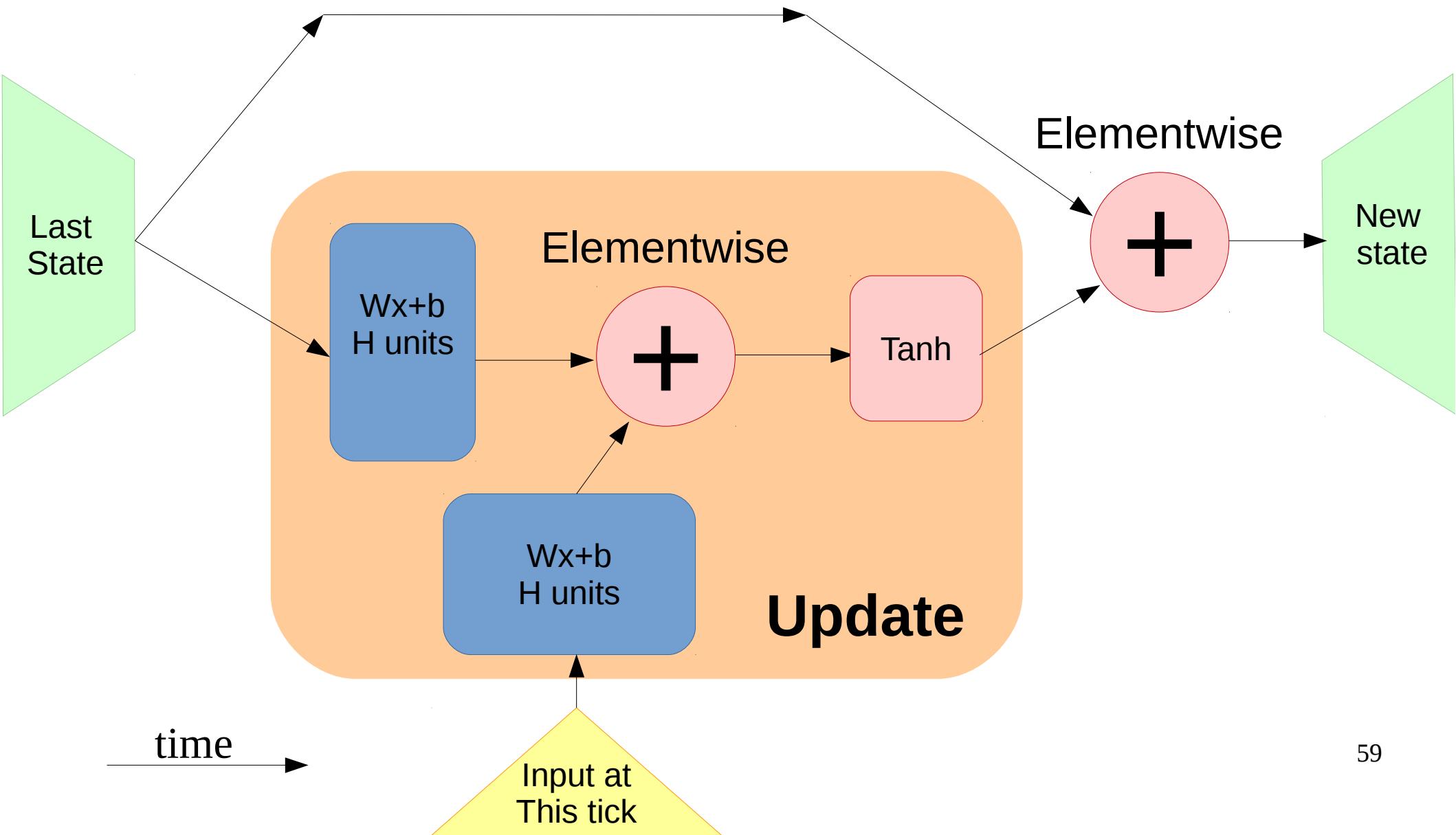
Residual RNN step



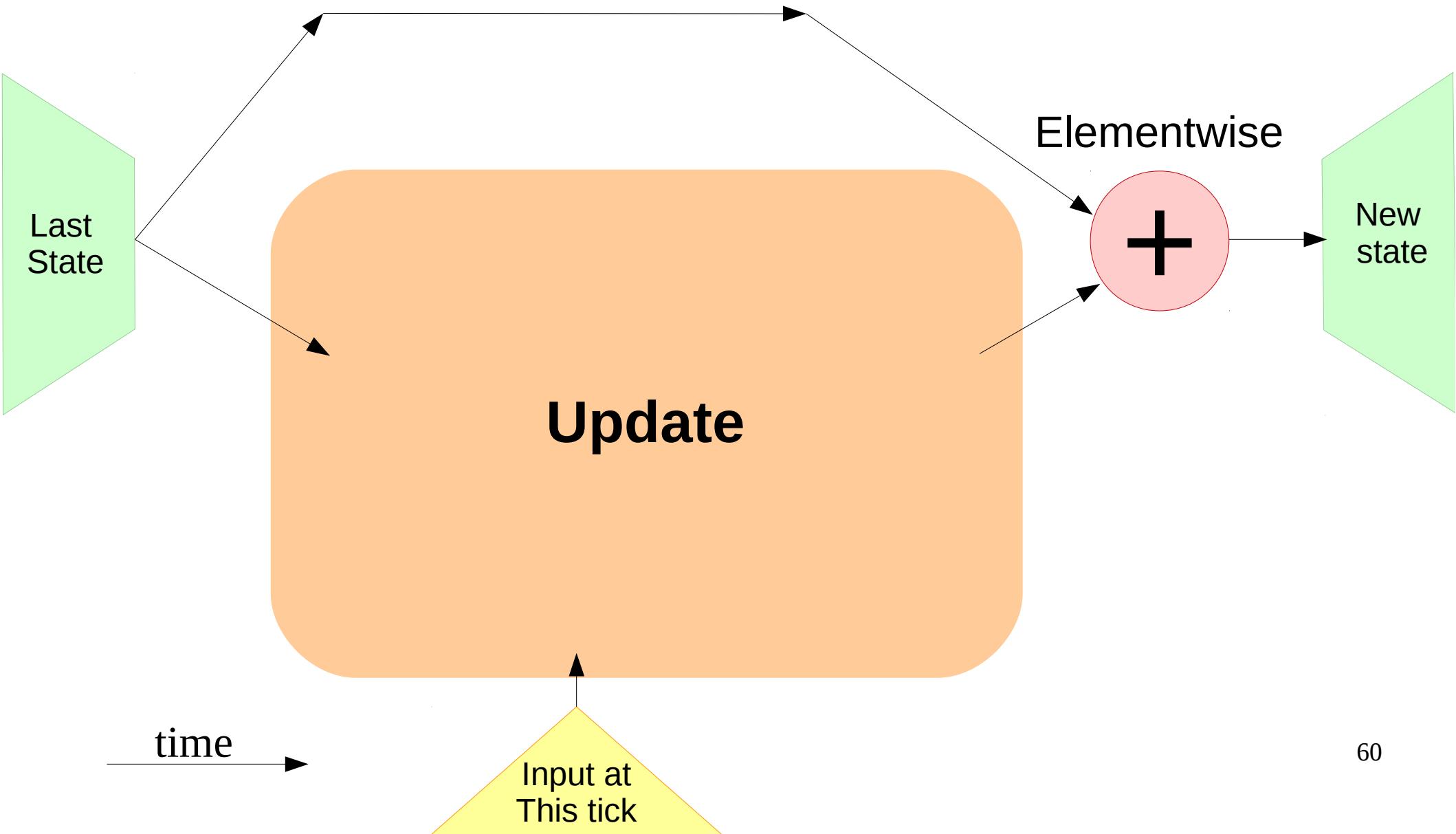
Residual RNN step



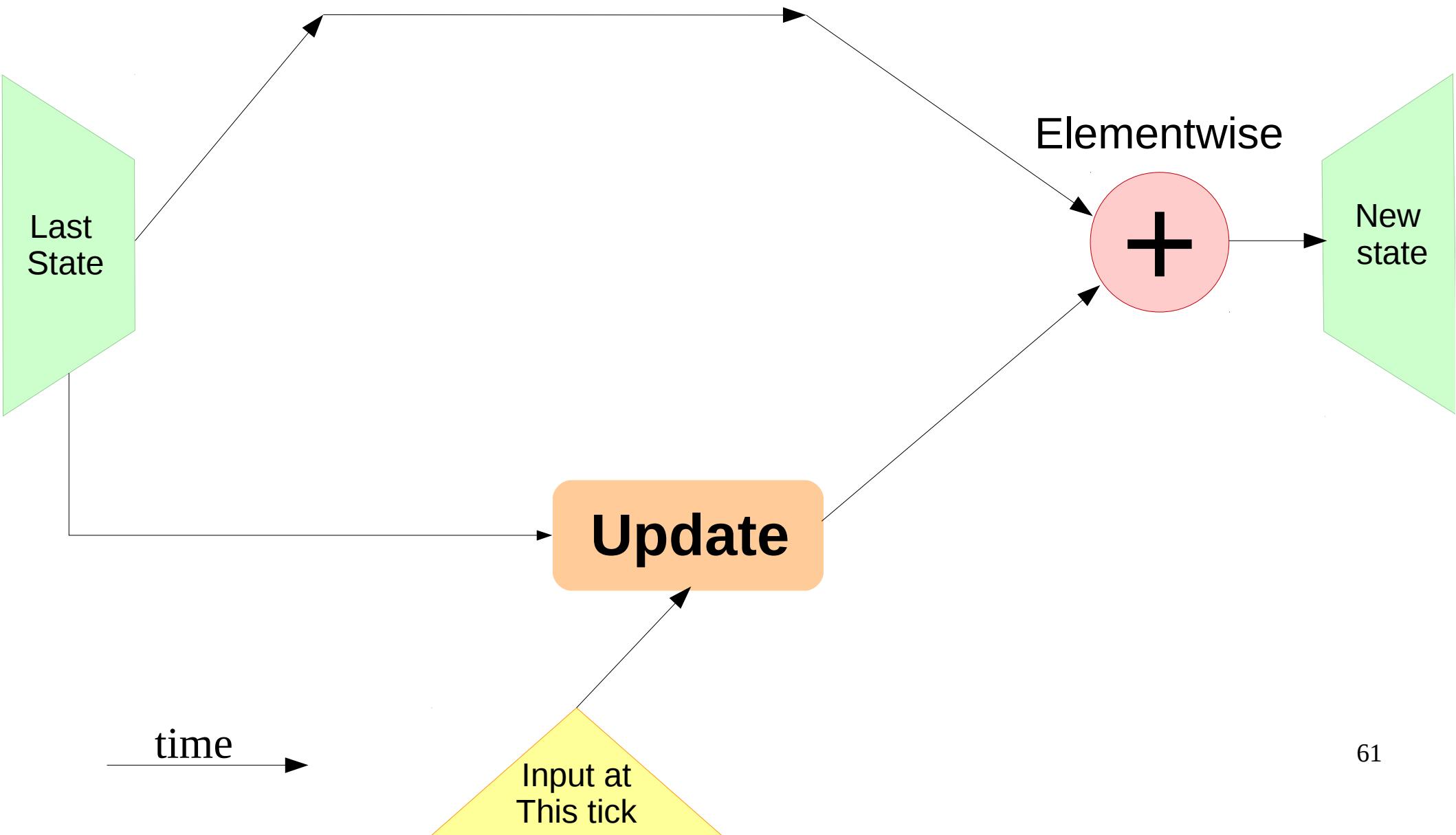
Residual RNN step



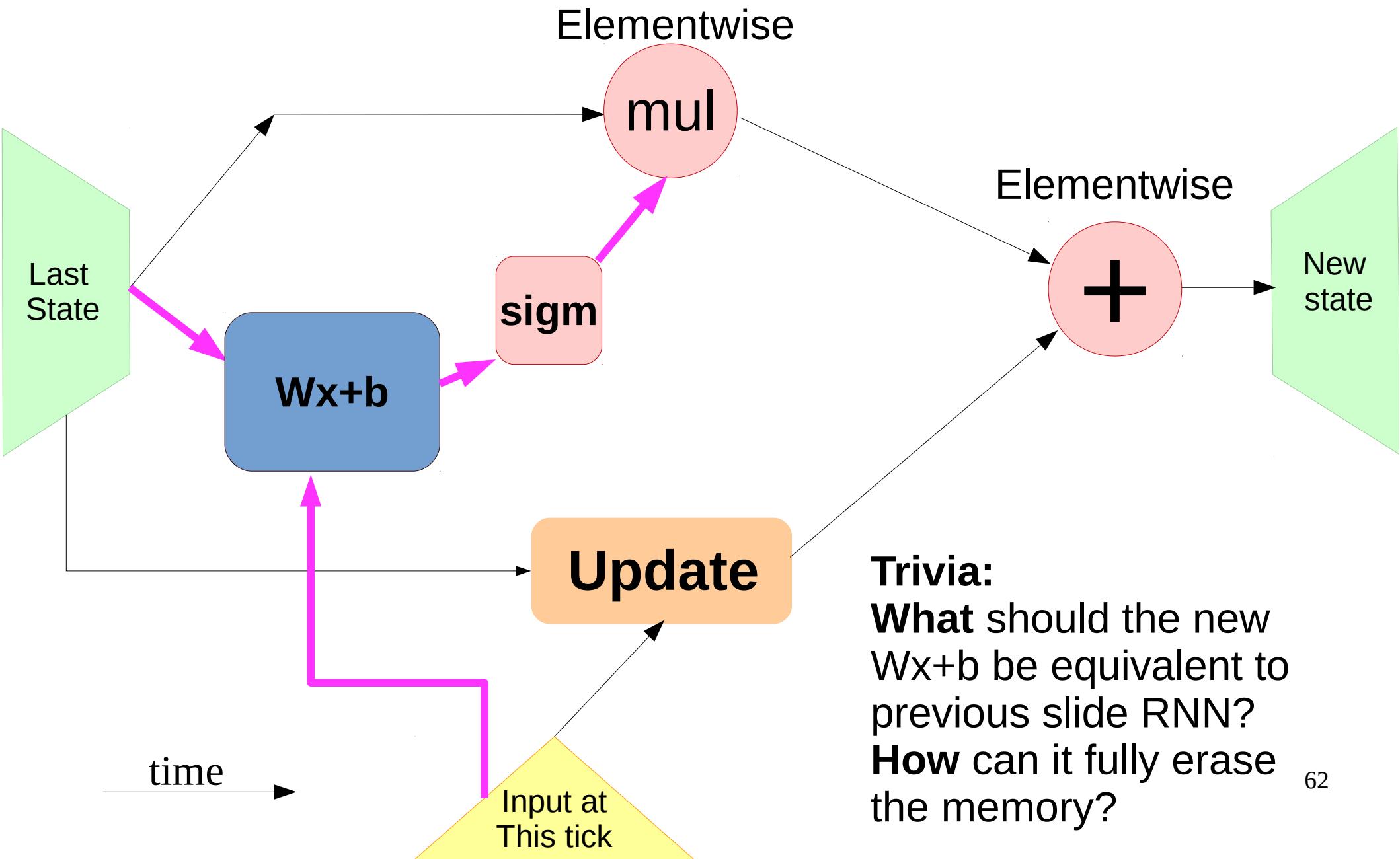
Residual RNN step



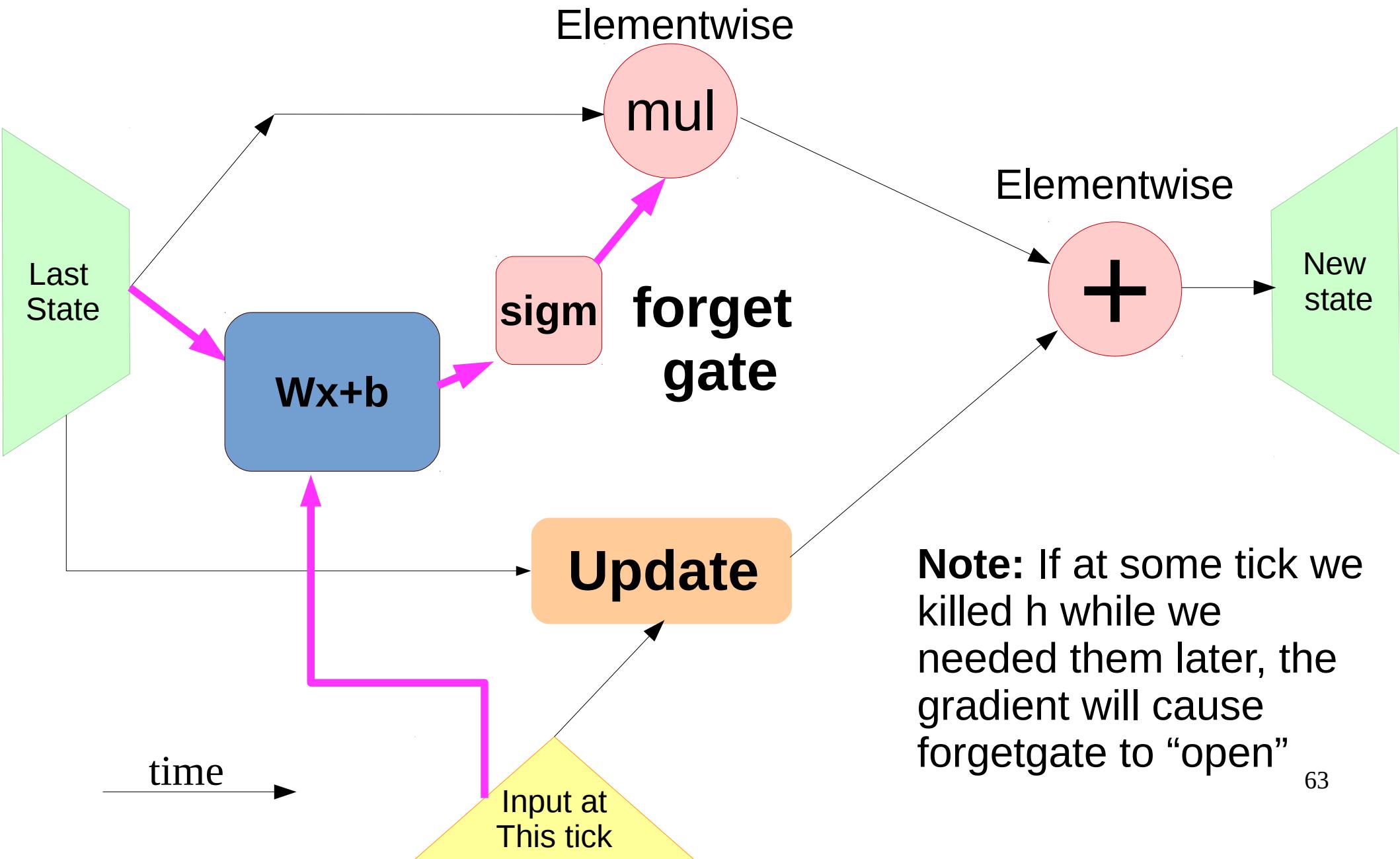
Residual RNN step



Residual RNN step

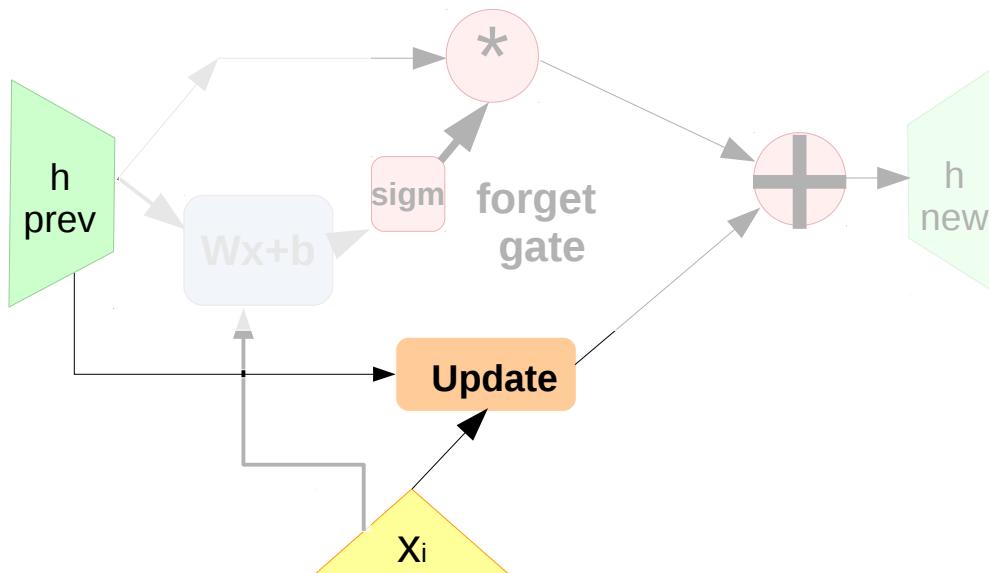


Residual RNN step



What we drew

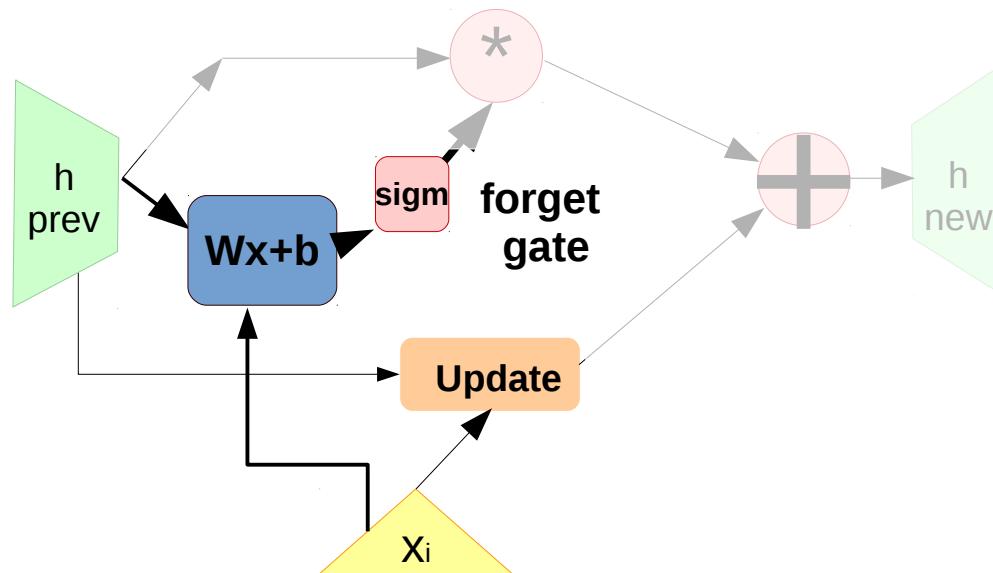
$$update(x_i, h_{i-1}) = \tanh(W_{hid}^{update} \cdot h_{i-1} + W_{inp}^{update} \cdot x_i + b^{update})$$



What we drew

$$update(x_i, h_{i-1}) = \tanh(W_{hid}^{update} \cdot h_{i-1} + W_{inp}^{update} \cdot x_i + b^{update})$$

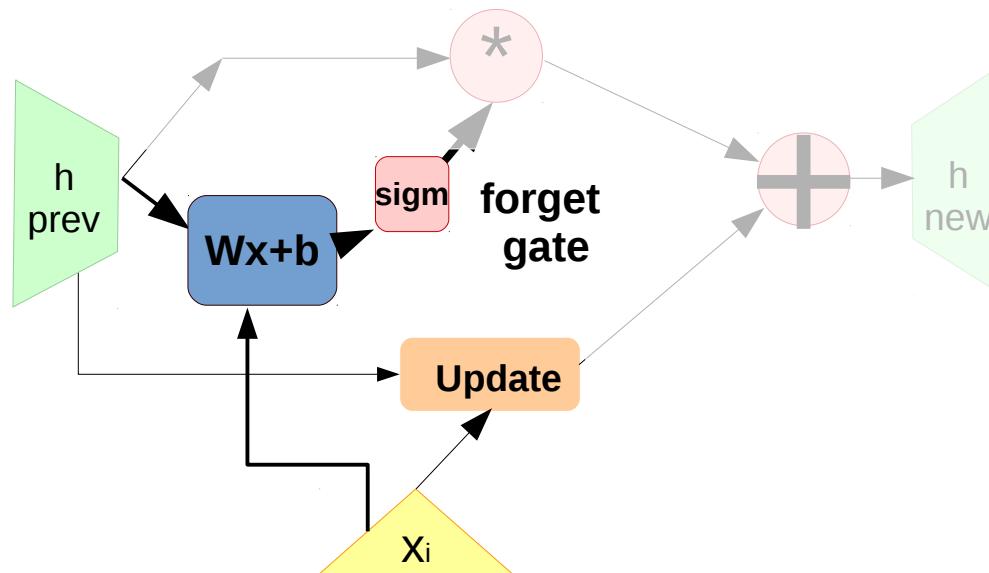
$$forget(x_i, h_{i-1}) = \sigma(W_{hid}^{forget} \cdot h_{i-1} + W_{inp}^{forget} \cdot x_i + b^{forget})$$



What we drew

$$update(x_i, h_{i-1}) = \tanh(W_{hid}^{update} \cdot h_{i-1} + W_{inp}^{update} \cdot x_i + b^{update})$$

$$forget(x_i, h_{i-1}) = \sigma(W_{hid}^{forget} \cdot h_{i-1} + W_{inp}^{forget} \cdot x_i + b^{forget})$$



How to compute h_{new} ?

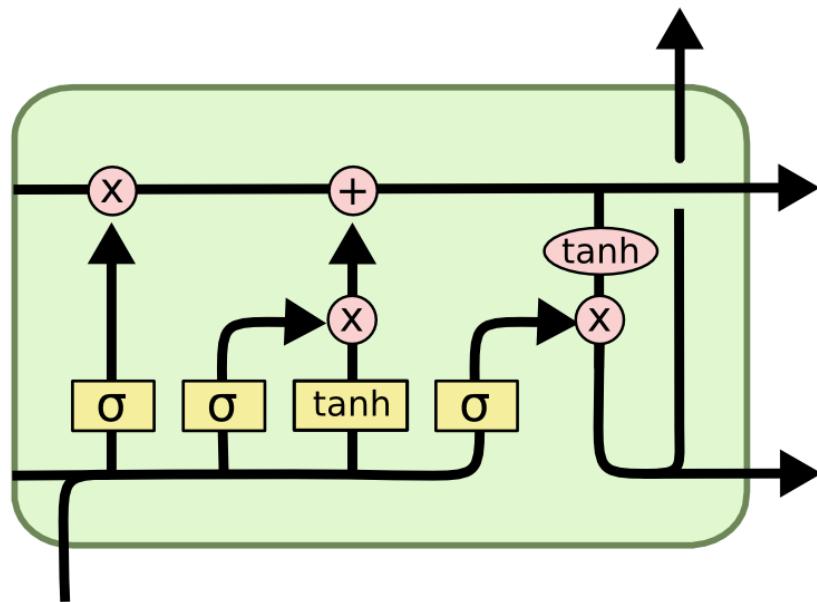
What we drew

$$update(x_i, h_{i-1}) = \tanh(W_{hid}^{update} \cdot h_{i-1} + W_{inp}^{update} \cdot x_i + b^{update})$$

$$forget(x_i, h_{i-1}) = \sigma(W_{hid}^{forget} \cdot h_{i-1} + W_{inp}^{forget} \cdot x_i + b^{forget})$$

$$h_i(x_i, h_{i-1}) = forget(x_i, h_{i-1}) \cdot h_{i-1} + update(x_i, h_{i-1})$$

LSTM



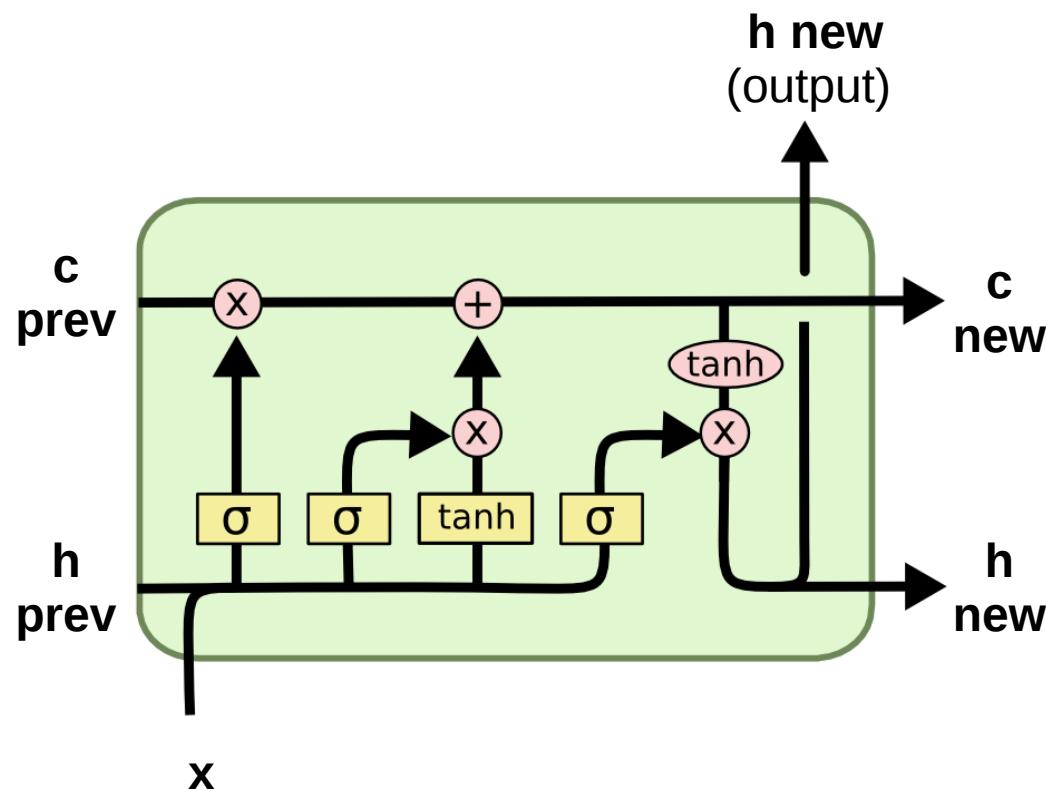
2 hidden states:

- Cell (“private” state)
- Output (“public” state)

4 blocks:

- Update
- Forget gate
- Input gate
- Output gate

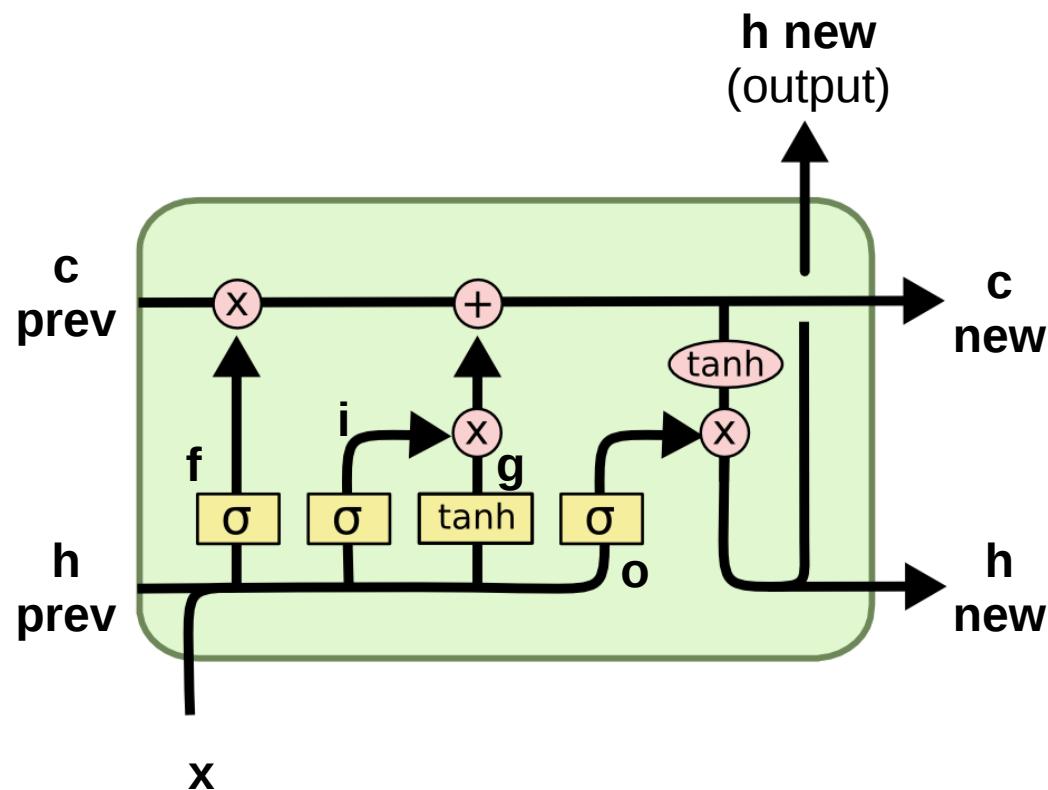
LSTM



$$i_t = \text{Sigm}(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$f_t = \text{Sigm}(\theta_{xf}x_t + \theta_{hf}h_{t-1} + b_f)$$
$$o_t = \text{Sigm}(\theta_{xo}x_t + \theta_{ho}h_{t-1} + b_o)$$
$$g_t = \text{Tanh}(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$
$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t$$
$$h_t = o_t \otimes \text{Tanh}(c_t)$$

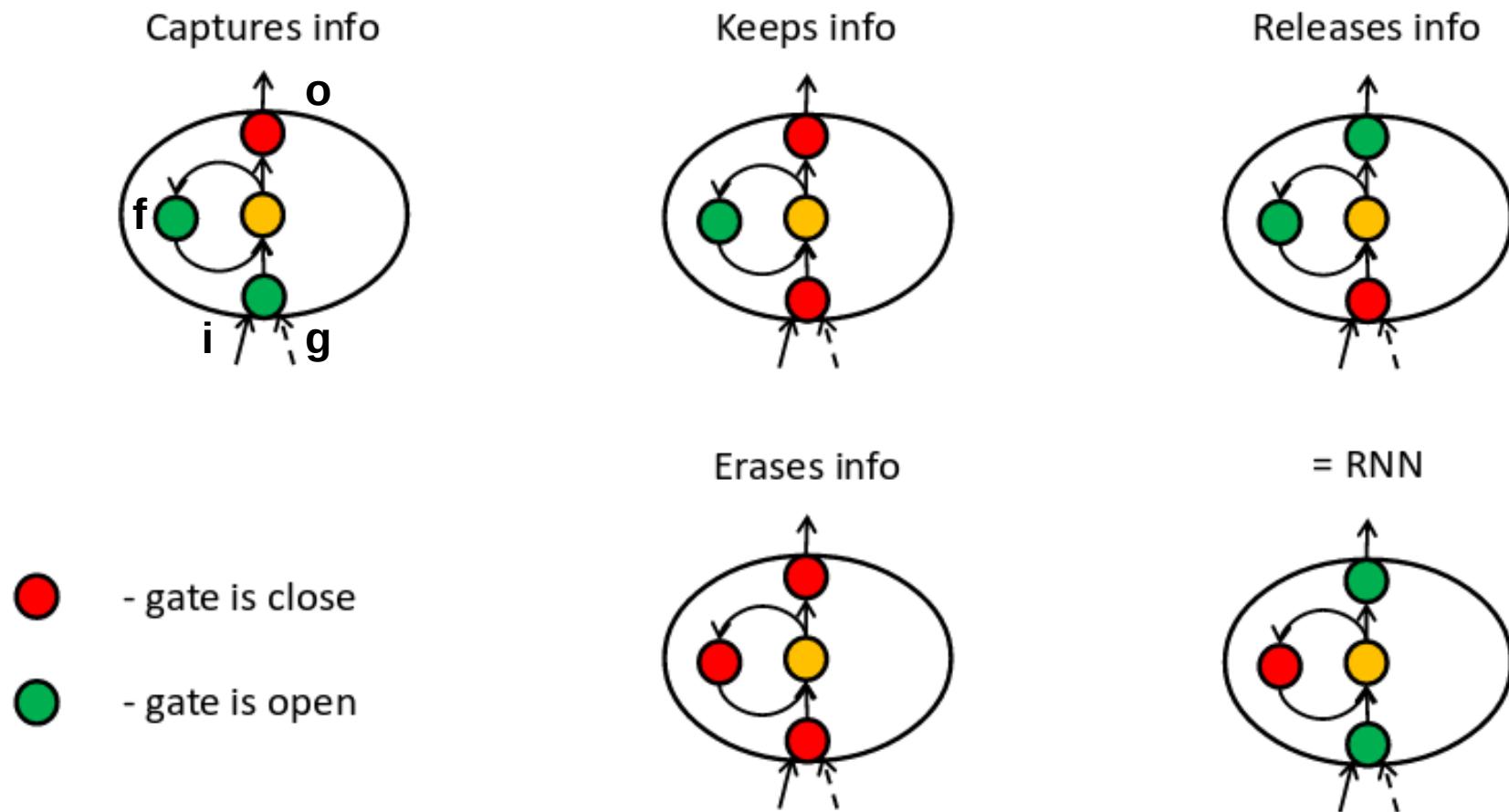
Where are the gates?

LSTM



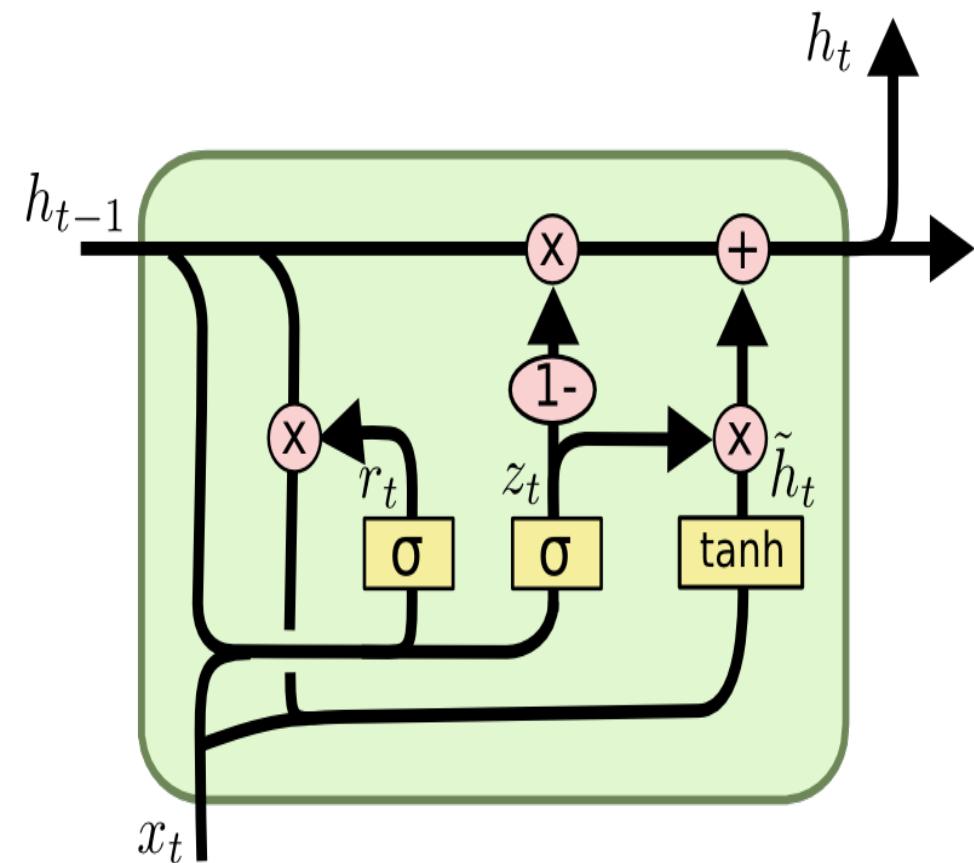
$$\begin{aligned}i_t &= \text{Sigm}(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i) \\f_t &= \text{Sigm}(\theta_{xf}x_t + \theta_{hf}h_{t-1} + b_f) \\o_t &= \text{Sigm}(\theta_{xo}x_t + \theta_{ho}h_{t-1} + b_o) \\g_t &= \text{Tanh}(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g) \\c_t &= f_t \otimes c_{t-1} + i_t \otimes g_t \\h_t &= o_t \otimes \text{Tanh}(c_t)\end{aligned}$$

LSTM: not a monster



[Pictures: E Lobacheva, D Vetrov]

GRU



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

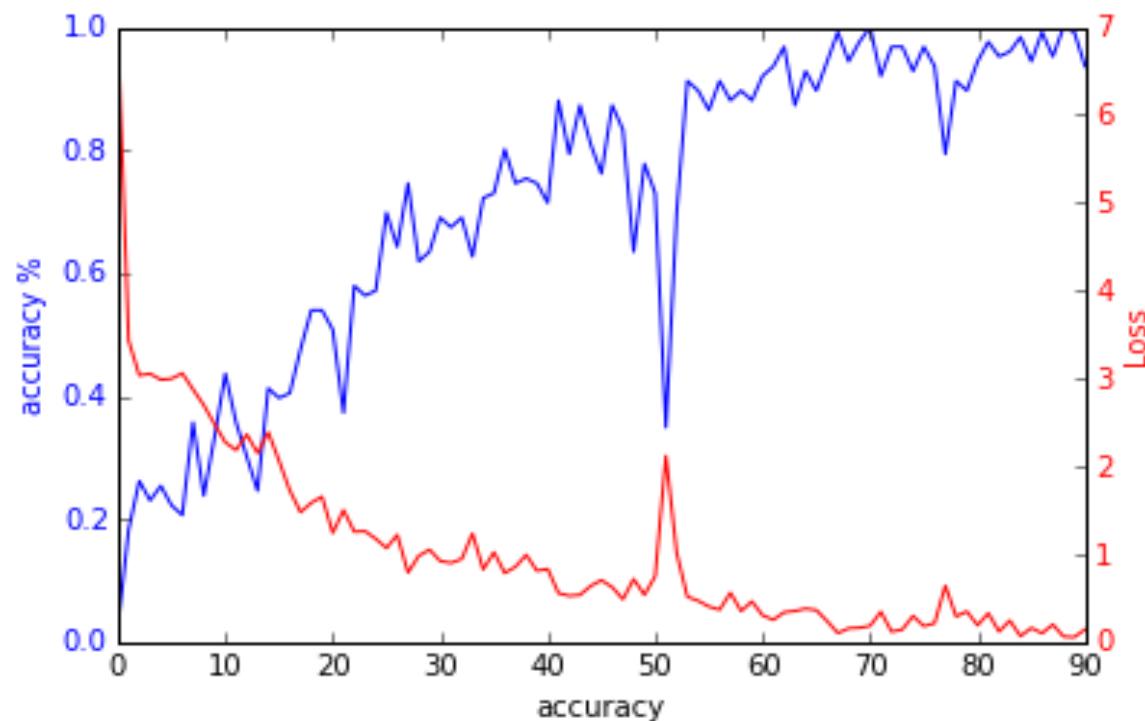
$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Okay, the gradients no longer vanish
except they still do, if only slower

But how do we deal with exploding grads?



Ideas?

Gradient clipping

At each time tick,

- check if grad abs value is more than ... 5?
- If so, clip it
 - large positive is now 5,
 - large negative is now -5
- How large is too large?
 - Reduce clipping threshold until explosions disappear

Gradient clipping

Where do I clip?

- Clip each element of $\delta L/\delta w$
- Clip each element of $\delta h_{i+1}/\delta h_i$
- Clip whole $\delta L/\delta w$ by norm
- If $\left\| \frac{\delta L}{\delta w} \right\| > 5$, scale $\frac{\delta L}{\delta w} / \left\| \frac{\delta L}{\delta w} \right\| \cdot 5$

Generating stuff

Easy:

- Names, small phrases
- Orthographically correct delirium

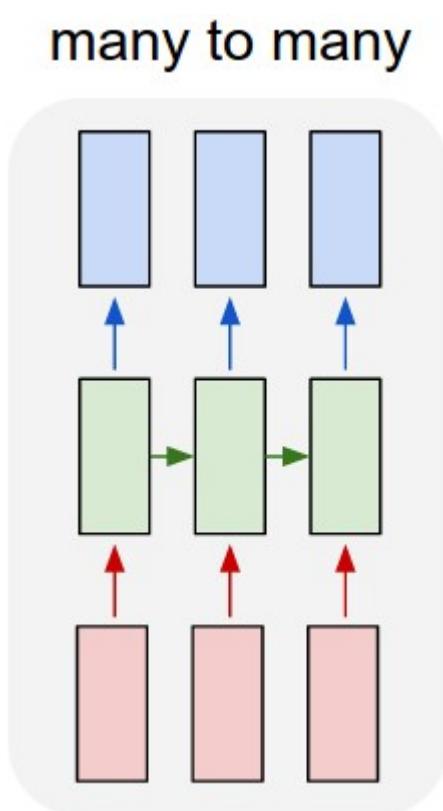
Medium:

- Grammatically coherent text
- Resembling particular author

Hard:

- C/C++ source code
- Music
- Organic molecules
- LaTex articles
- Your course projects

Recurrent Architectures: regular



- Read sequence
- Predict sequence of answers at each tick

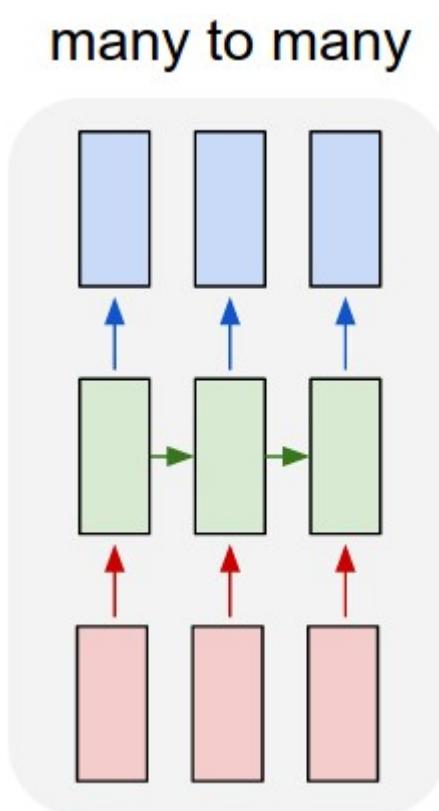
Tasks:

- Language model
- POS Tagging

How to implement?

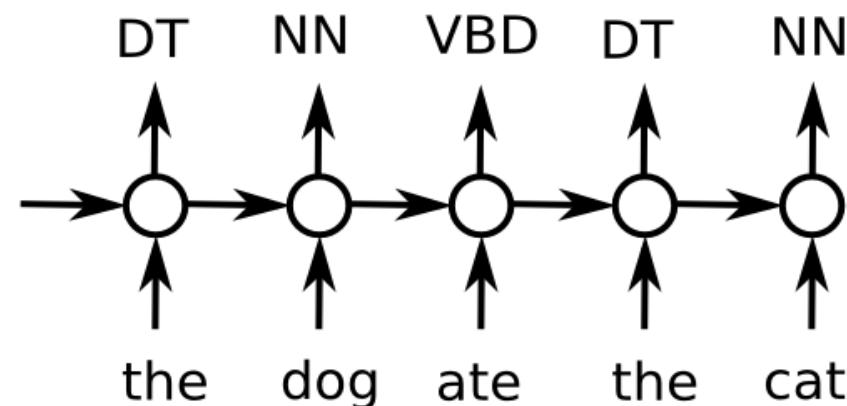
- See last week

Recurrent Architectures: regular



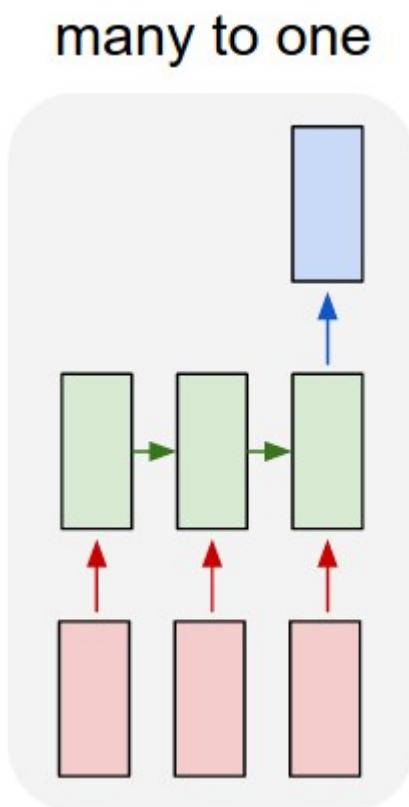
- Read sequence
- Predict sequence of answers at each tick

POS tagging



Why RNN?

Recurrent Architectures: Encoder



Encoder

- Read sequence
- Predict once

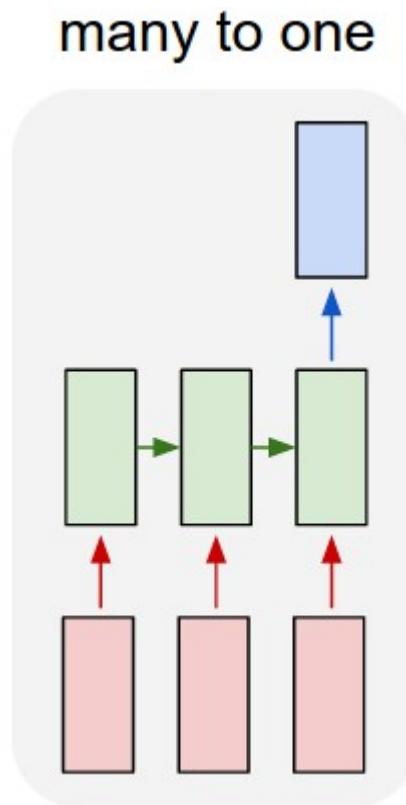
Tasks:

- ?!

How to implement?

- ?!

Recurrent Architectures: Encoder



Encoder

- Read sequence
- Predict once

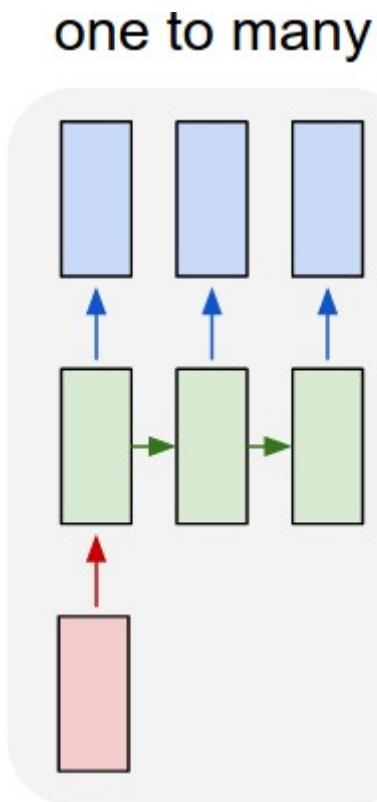
Tasks:

- Sentiment analysis
- Detect age by status
- Filter bad content
- Any text analysis

How to implement?

- Take last/max/mean over time

Recurrent Architectures: Decoder

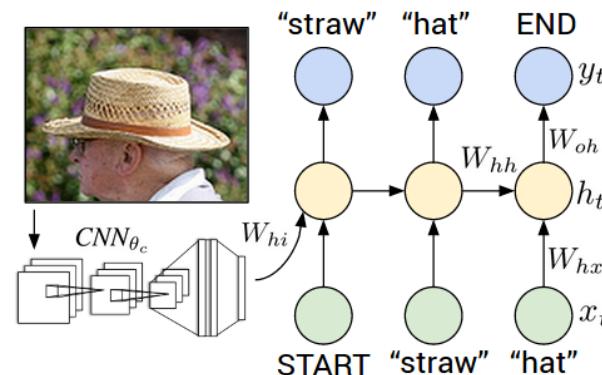


Decoder

- Take one state
- Generate sequence

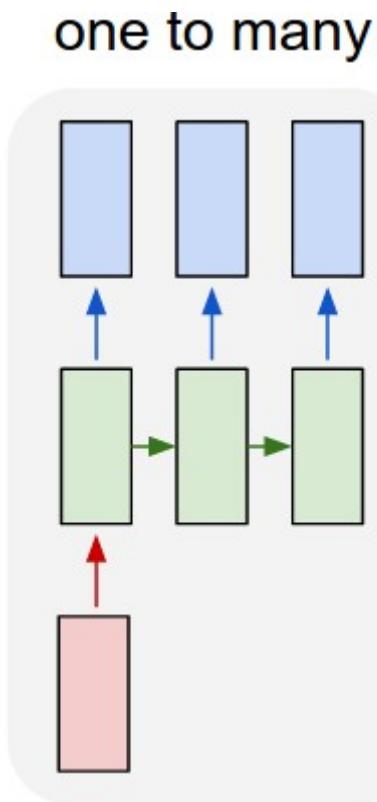
Tasks:

- Image captioning



How to implement?
• ?!

Recurrent Architectures: Decoder

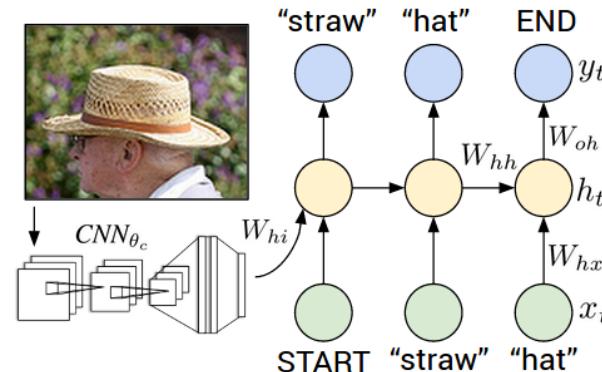


Decoder

- Take one state
- Generate sequence

Tasks:

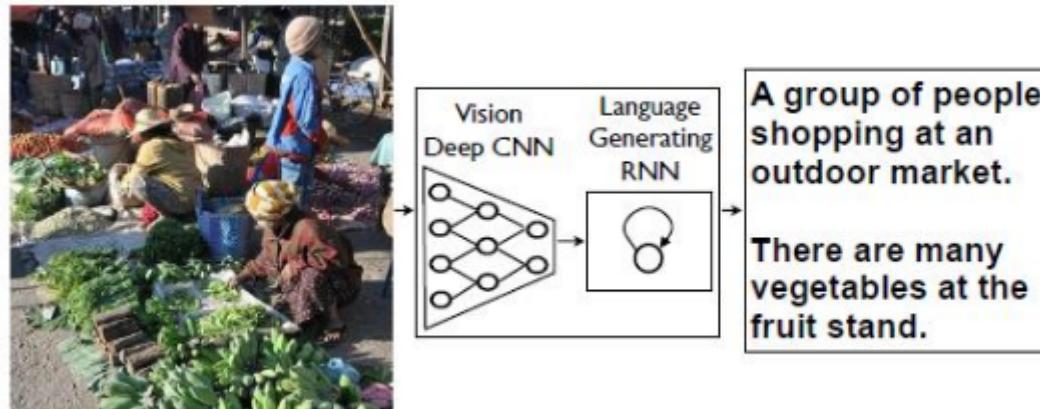
- Image captioning



How to implement?

- First state init (instead of zeros)
- Input at each tick

Image captioning



- Demo - <http://stanford.io/2esMxOq>
- Upload your image - <http://bit.ly/2eAoueP>

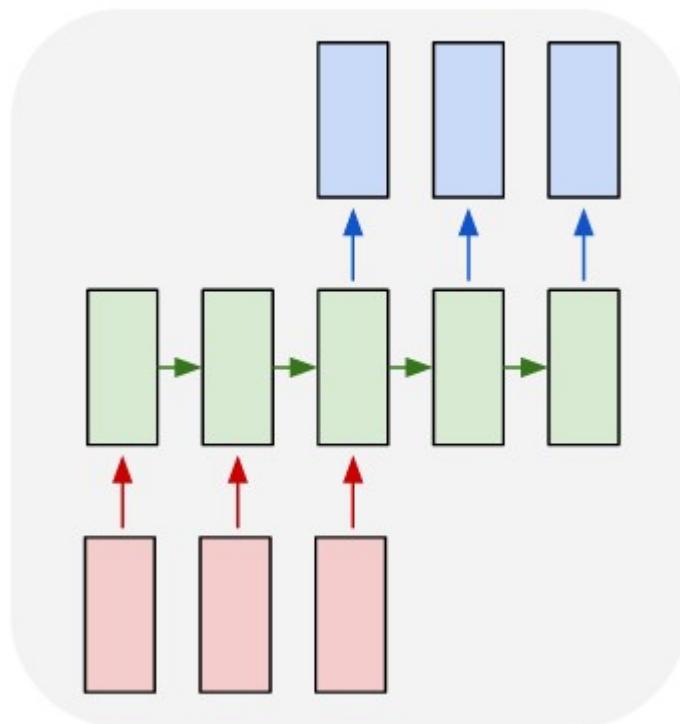
Seq2seq

How do we convert sequence to sequence of different kind/without time synchronization?

Example: Machine translation

Seq2seq

many to many

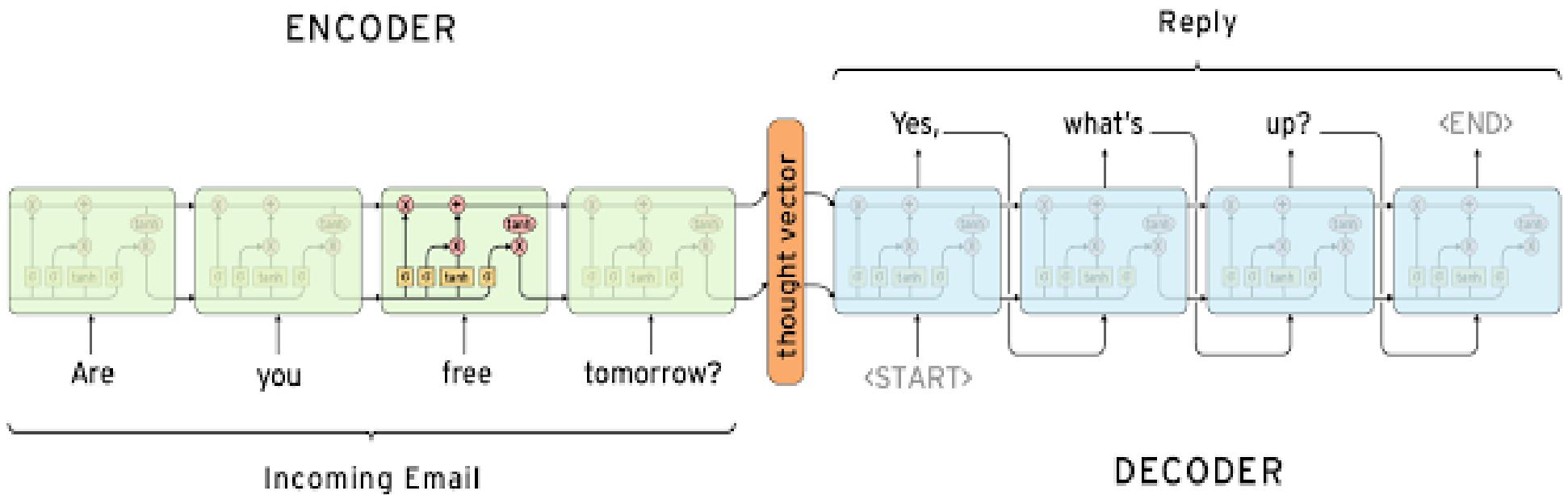


Idea:

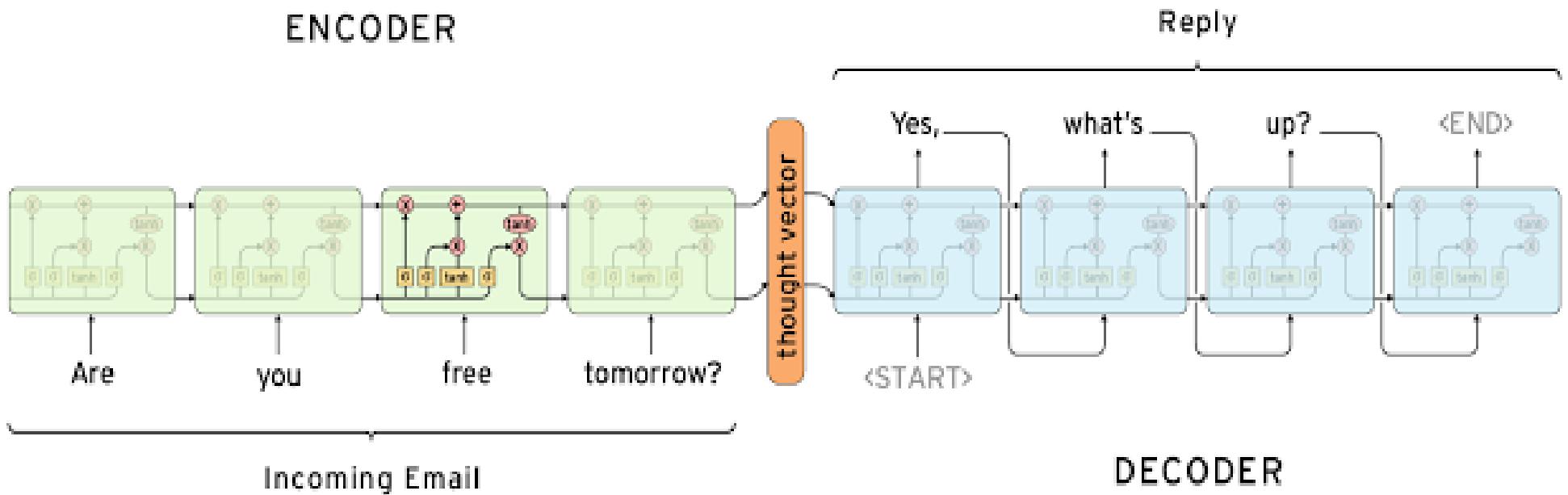
- first read (encode) the sequence
- then generate new one out of the encoded vector

How to implement that?

Seq2seq: encoder-decoder

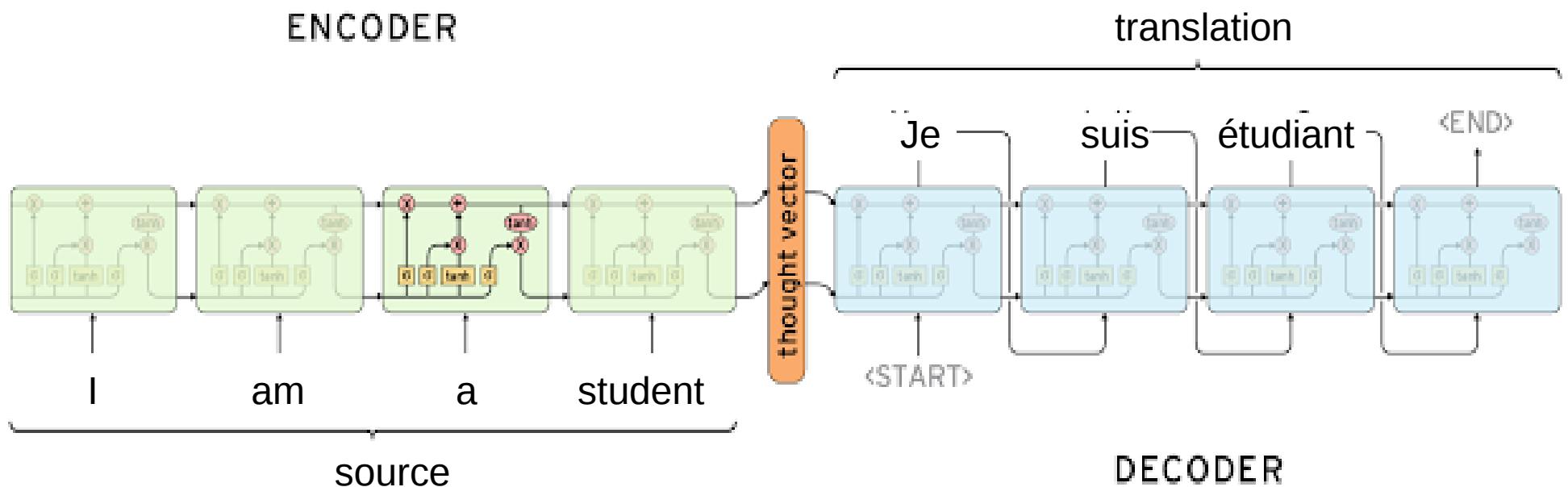


Seq2seq: Conversation model



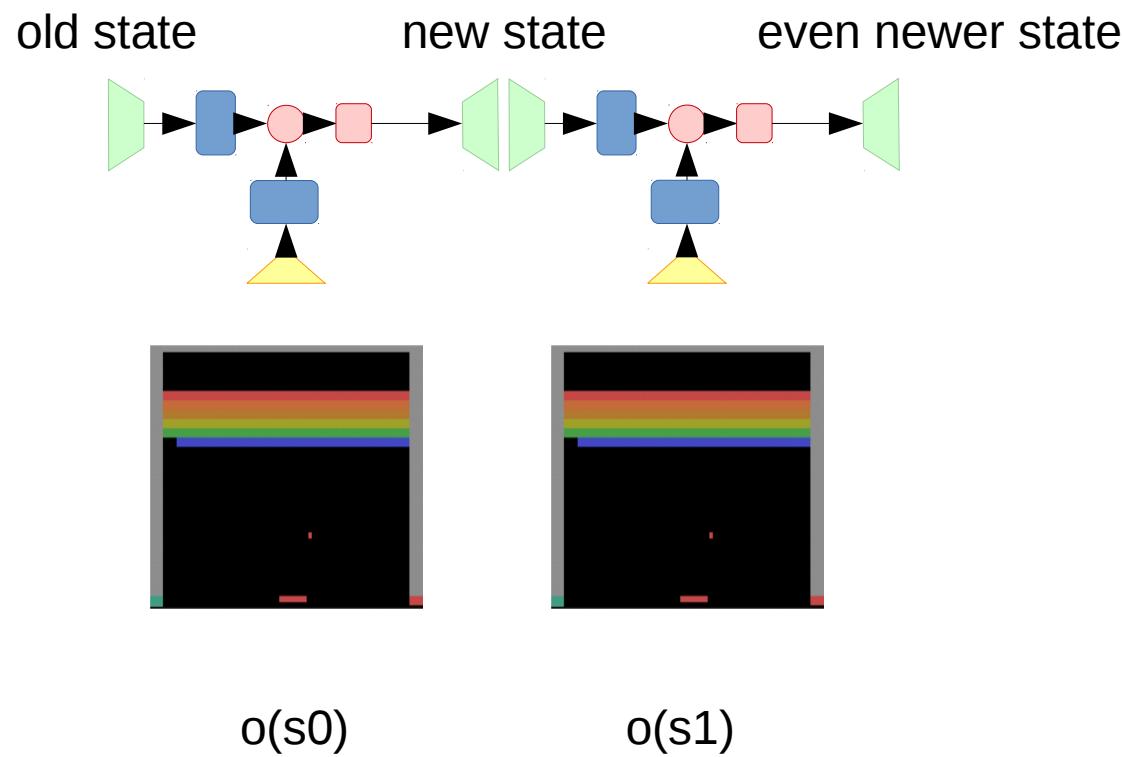
Exactly the same

Seq2seq: Machine translation



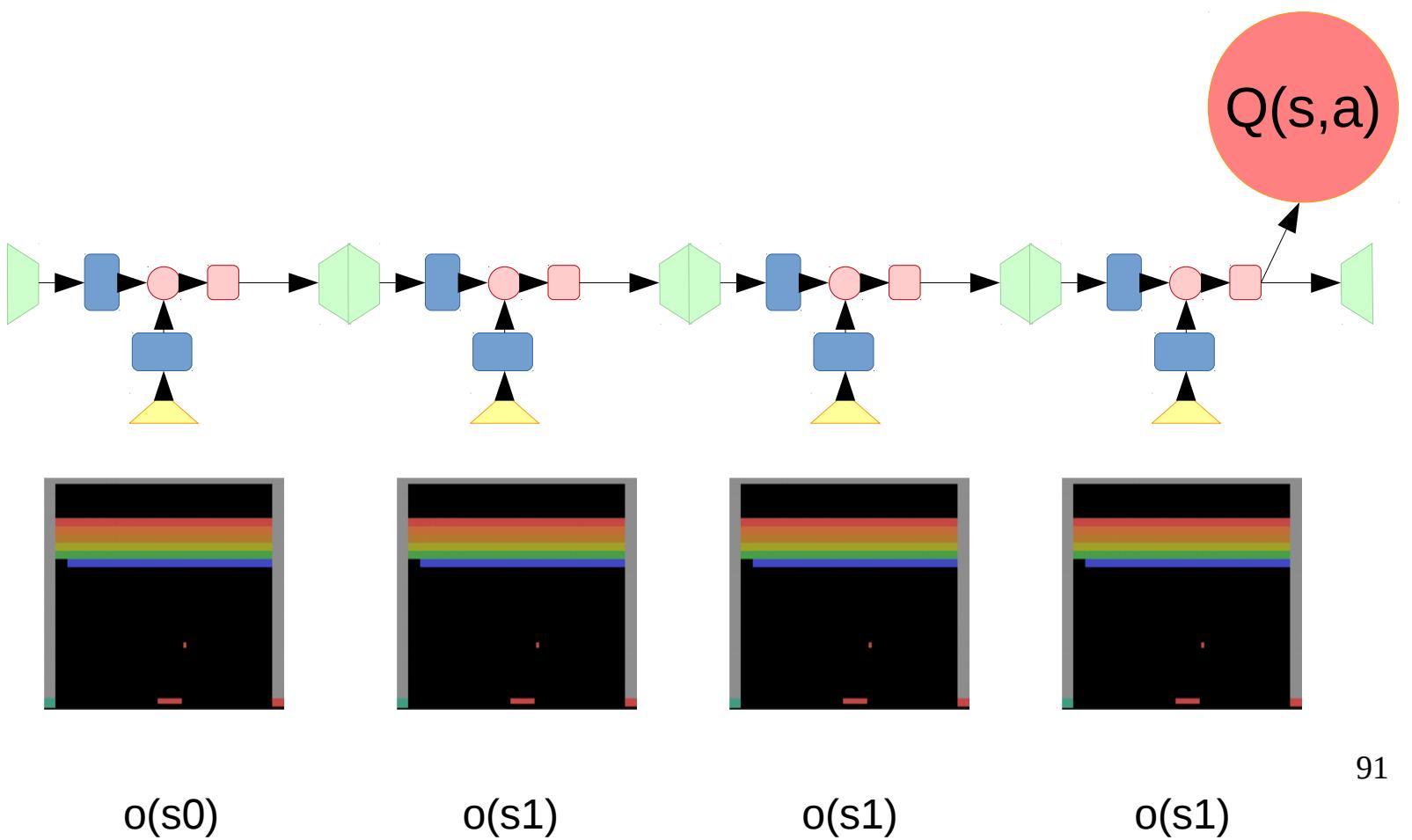
Meanwhile in reinforcement learning

RNN for RL



RNN for RL

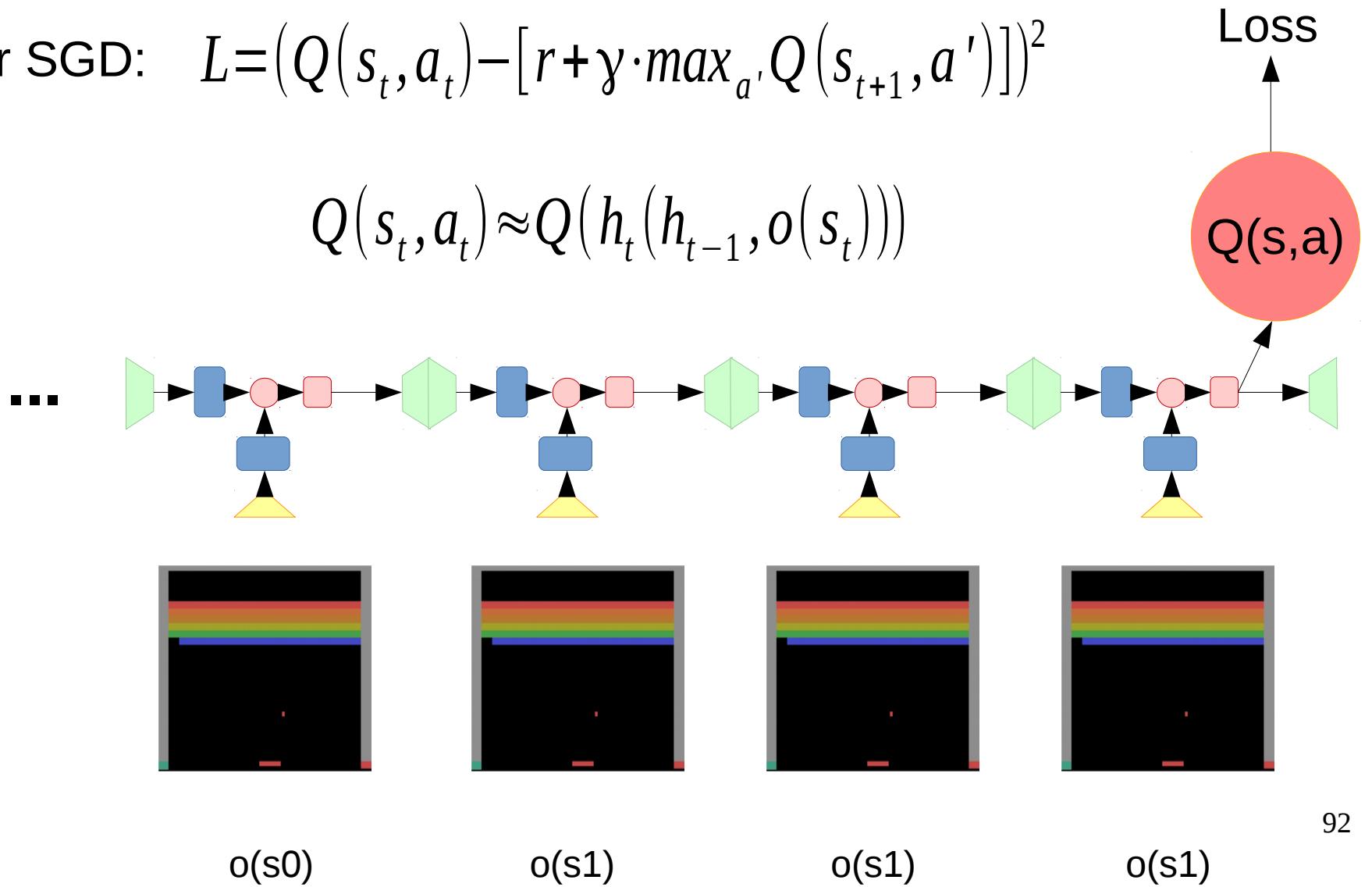
Or $\pi(a|s)$



Deep Recurrent Q-network

const
Loss for SGD: $L = (Q(s_t, a_t) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')])^2$

$$Q(s_t, a_t) \approx Q(h_t(h_{t-1}, o(s_t)))$$



To be continued...

Nuff

Coding time!

