Practical Reinforcement Learning

Episode 6.5

Recurrent neural networks

deep learning recap 2

Yandex
Data Factory

LAMBDA

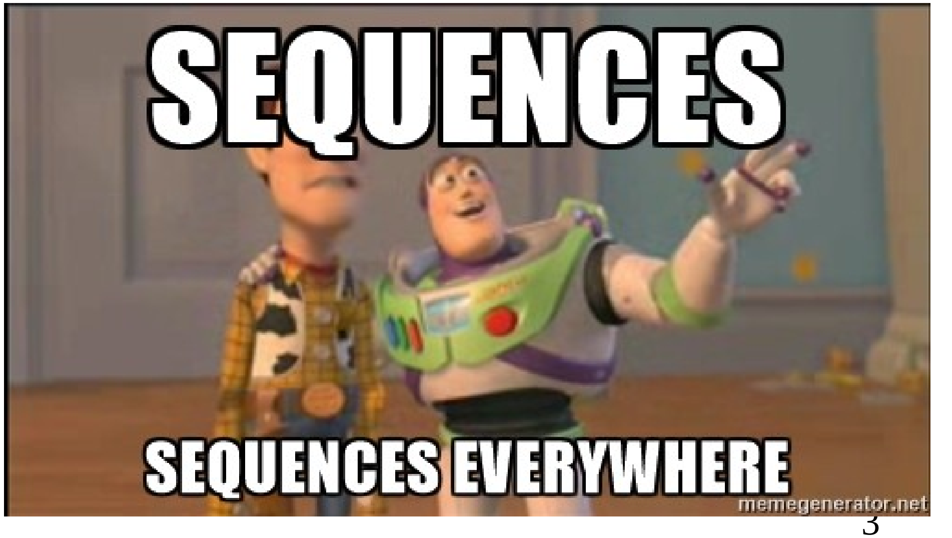
British Hedgehog
Preservation Society

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



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
- ...

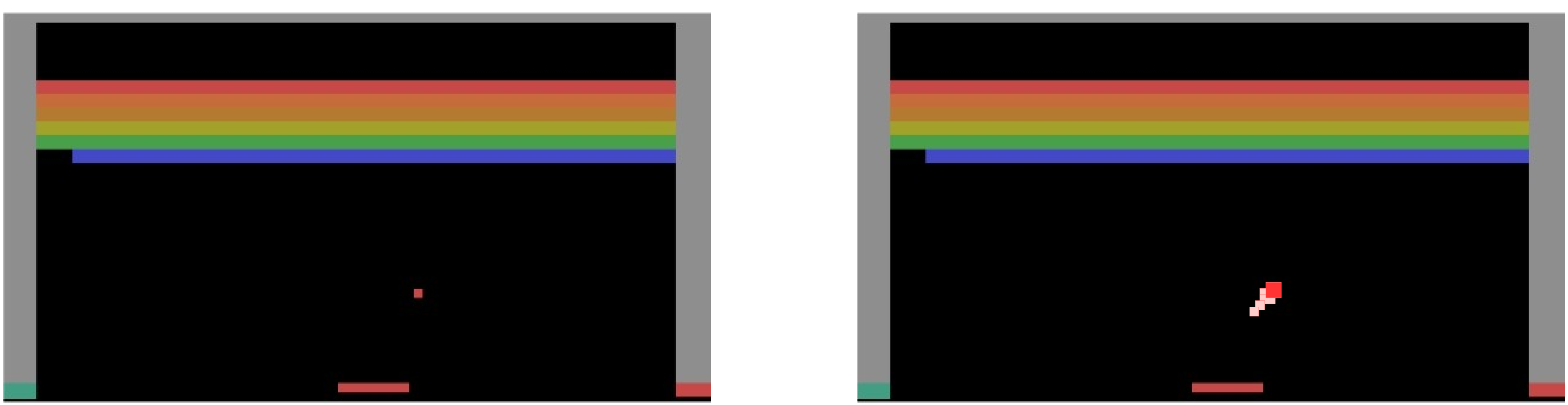
Time series @pomdp

- Agent observation is incomplete

  Field of view, hidden variables

- Infer state from sequence of observations!

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



Natural language as time series

Data:

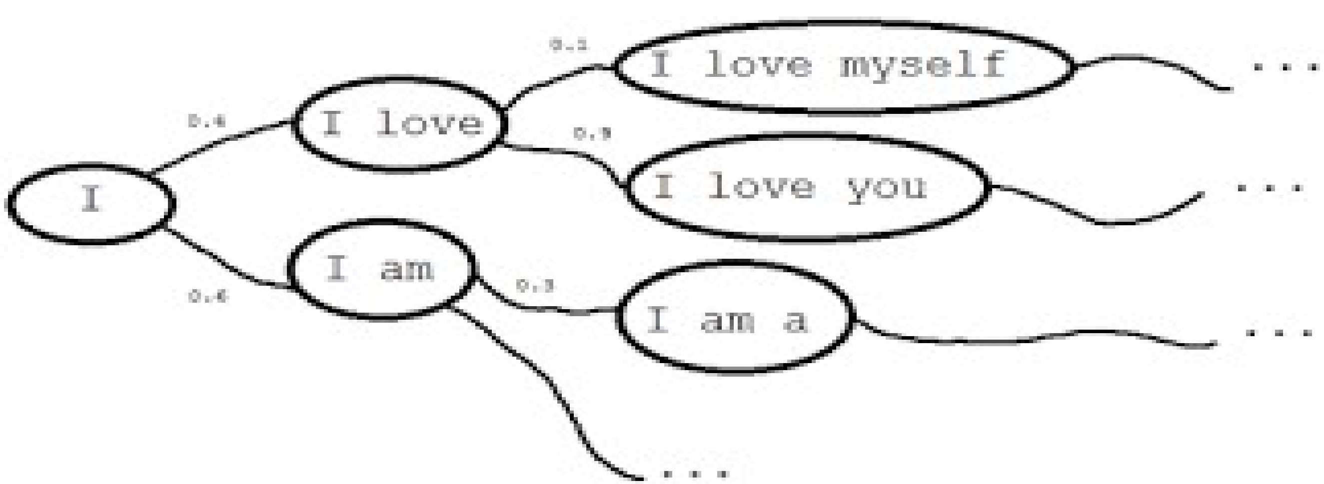
- Literature
- Conversation
- Tweets
- Book scans
- Speech

Language model

Objective:

- Learn P(text)

$$P(text)=P(w_{0,}w_{1,}\ldots,w_n)=P(w_0)\cdot P(w_1|w_0)\cdot P(w_2|w_1w_0)\cdot\ldots\cdot P(w_n|\ldots)$$



Language model

Why learning it?

- Detect languages as P(text|language)
- Sentiment analysis P(text|happy)
- Any text analysis you can imagine
- Generate texts!
  - Cool article http://bit.ly/1K610Ie
  - Generating clickbait: http://bit.ly/21cZM70

Language model

- Actual distribution

$$P(text) = P(w_{0,} w_{1,} \ldots, w_n) = P(w_0) \cdot P(w_1 | w_0) \cdot P(w_2 | w_1 w_0) \cdot \ldots \cdot P(w_n | \ldots)$$

- Bag of words assumption (independent words)

$$P(text) = P(w_{0,} w_{1,} \ldots, w_n) = P(w_0) \cdot P(w_1) \cdot P(w_2) \cdot \ldots \cdot P(w_n)$$

- **Anything better?**

Language model

- Actual distribution

$$P(text) = P(w_{0,} w_{1,} \ldots, w_n) = P(w_0) \cdot P(w_1 | w_0) \cdot P(w_2 | w_1 w_0) \cdot \ldots \cdot P(w_n | \ldots)$$

- Bag of words assumption (independent words)

$$P(text) = P(w_{0,} w_{1,} \ldots, w_n) = P(w_0) \cdot P(w_1) \cdot P(w_2) \cdot \ldots \cdot P(w_n)$$

- Markov assumption

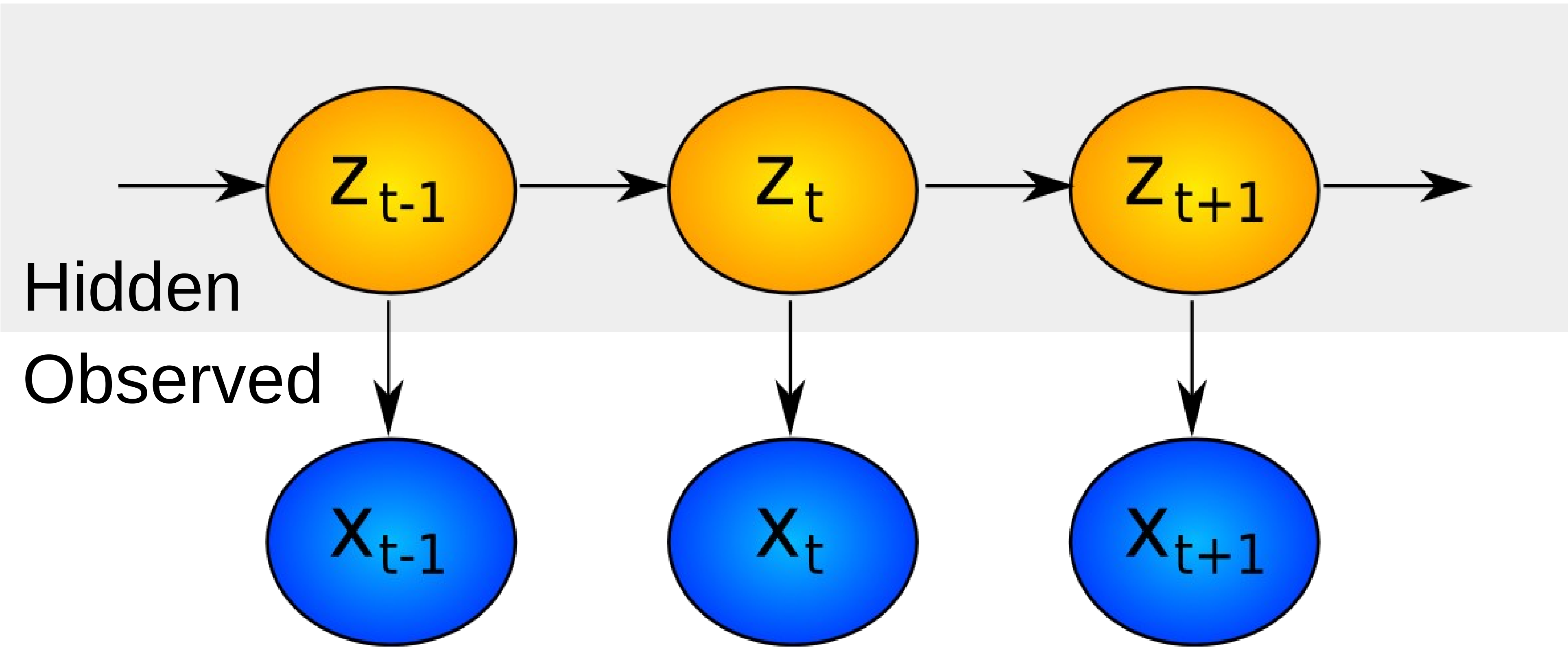
$$P(text) = P(w_{0,} w_{1,} \ldots, w_n) = P(w_0) \cdot P(w_1 | w\, 0) \cdot P(w_2 | w_1) \cdot \ldots \cdot P(w_n | w_{n-1})$$

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

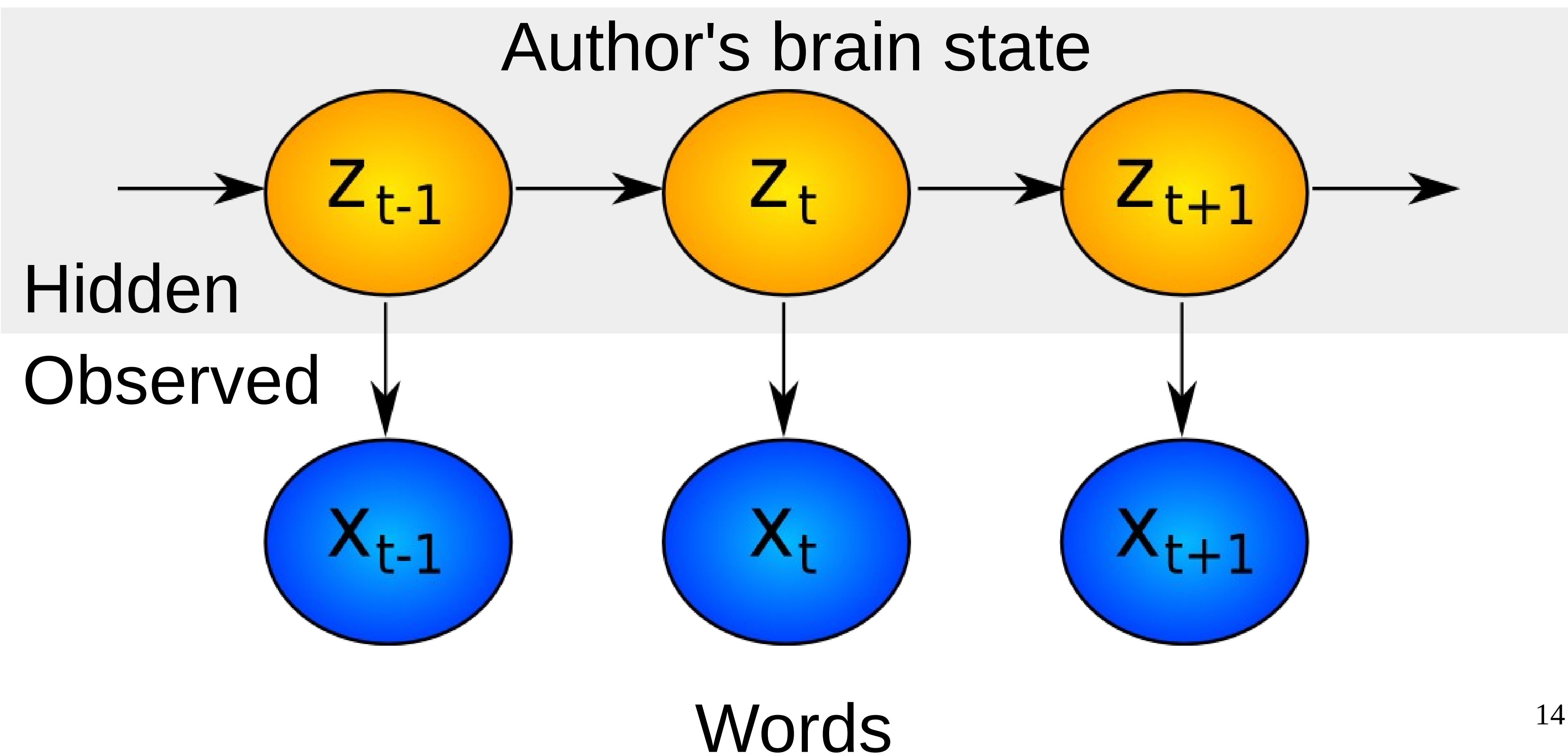
Can we learn* arbitrarily long dependencies?

* without infinitely many parameters

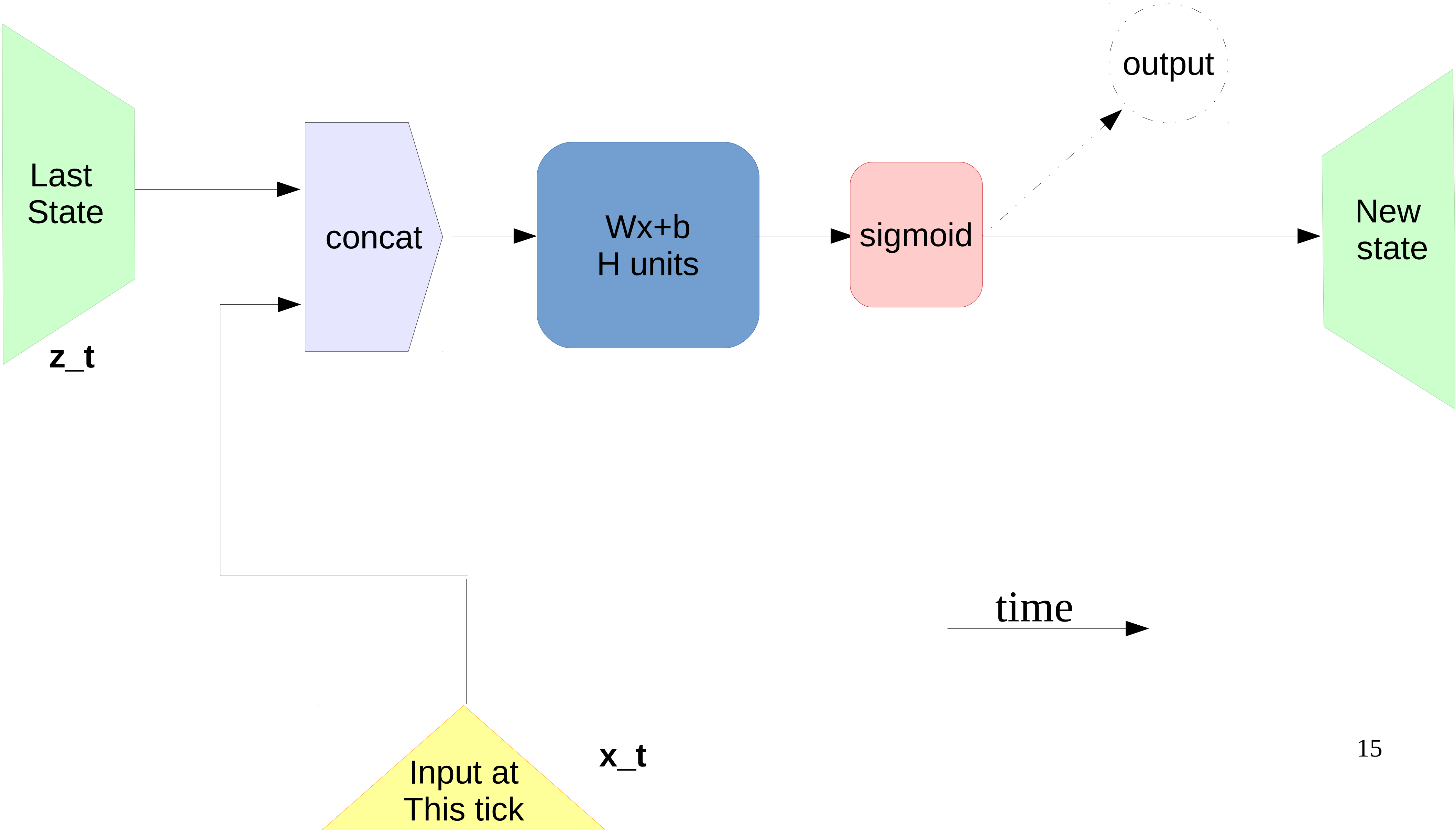
Hidden Markov Models: what's hidden



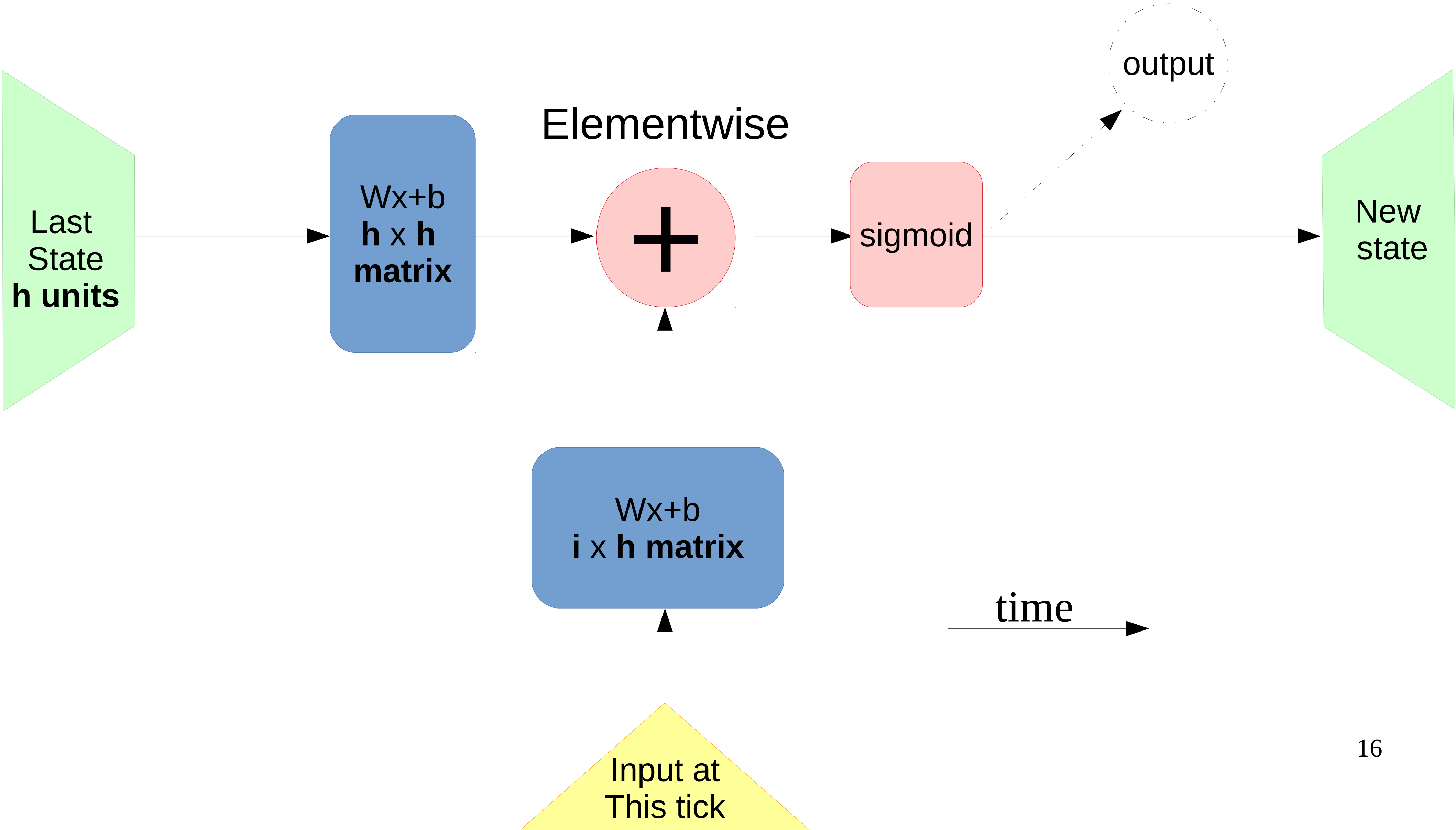
Hidden Markov Models: what is hidden



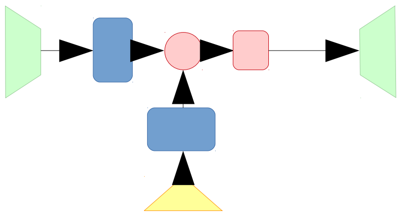
Recurrent neural network: one step



Recurrent neural network: one step

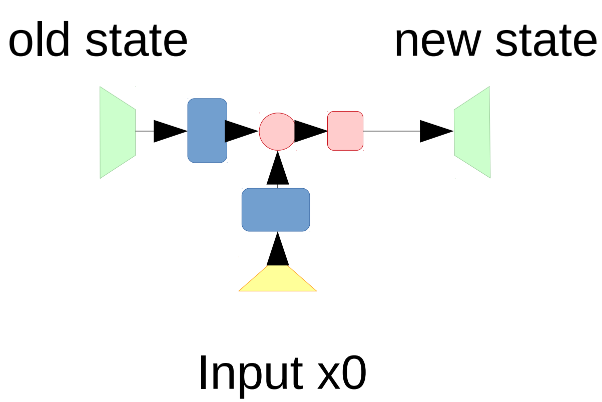


Recurrent neural network

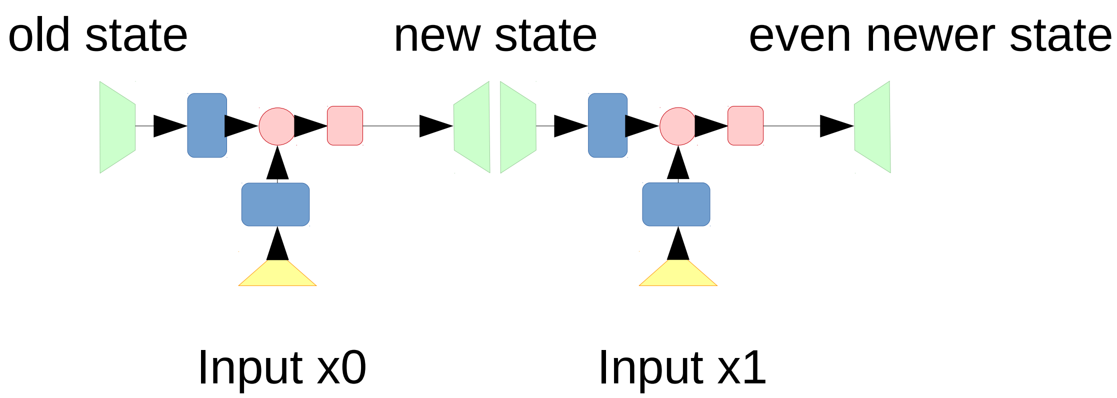


Zoom-out
of previous slide

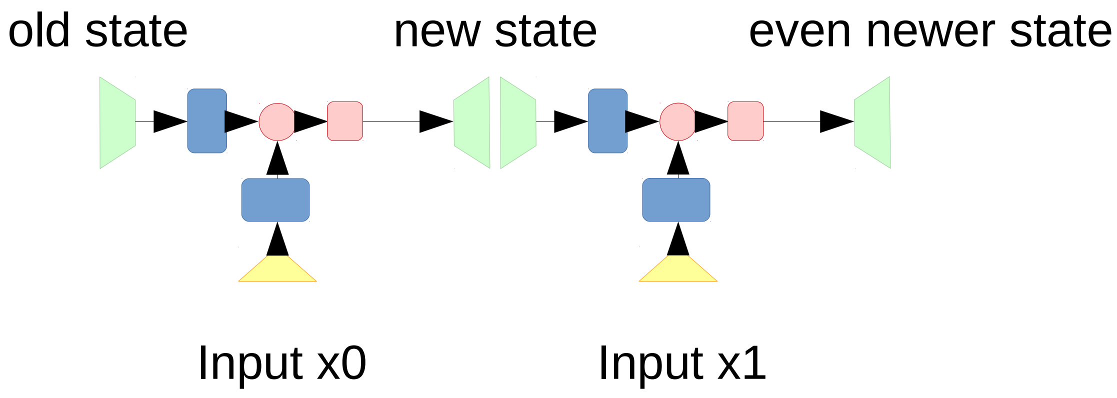
Recurrent neural network



Recurrent neural network

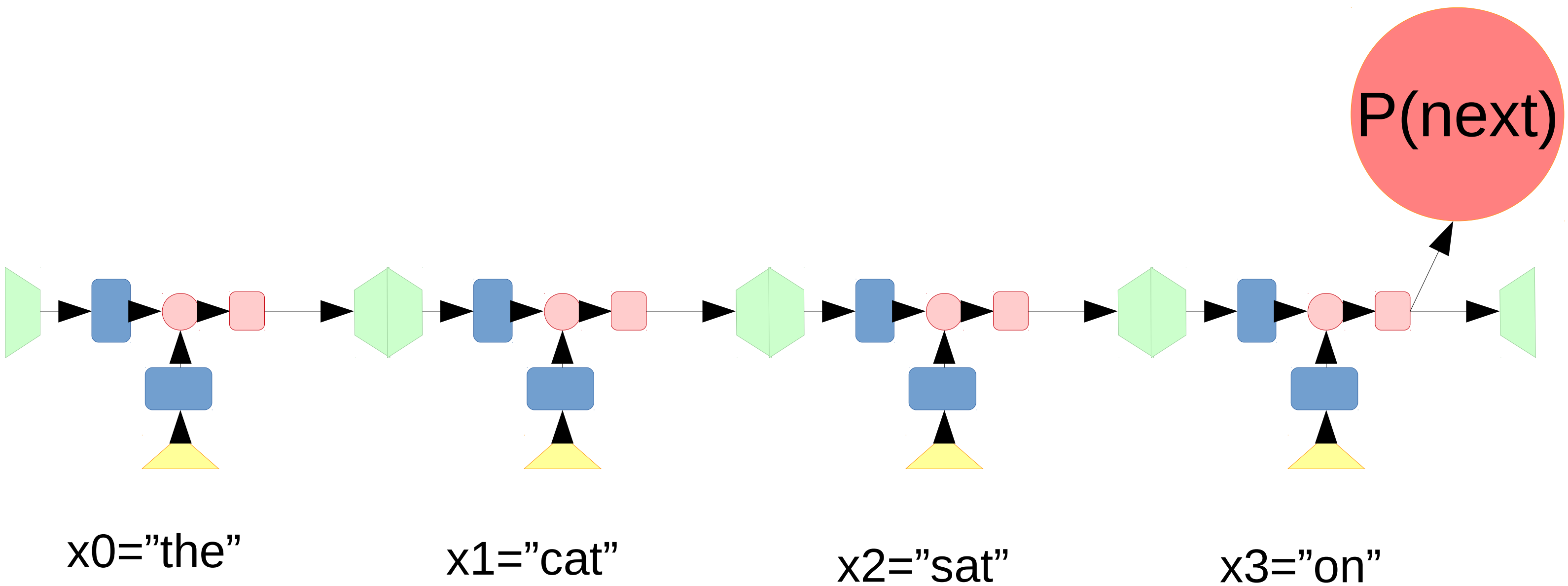


Recurrent neural network

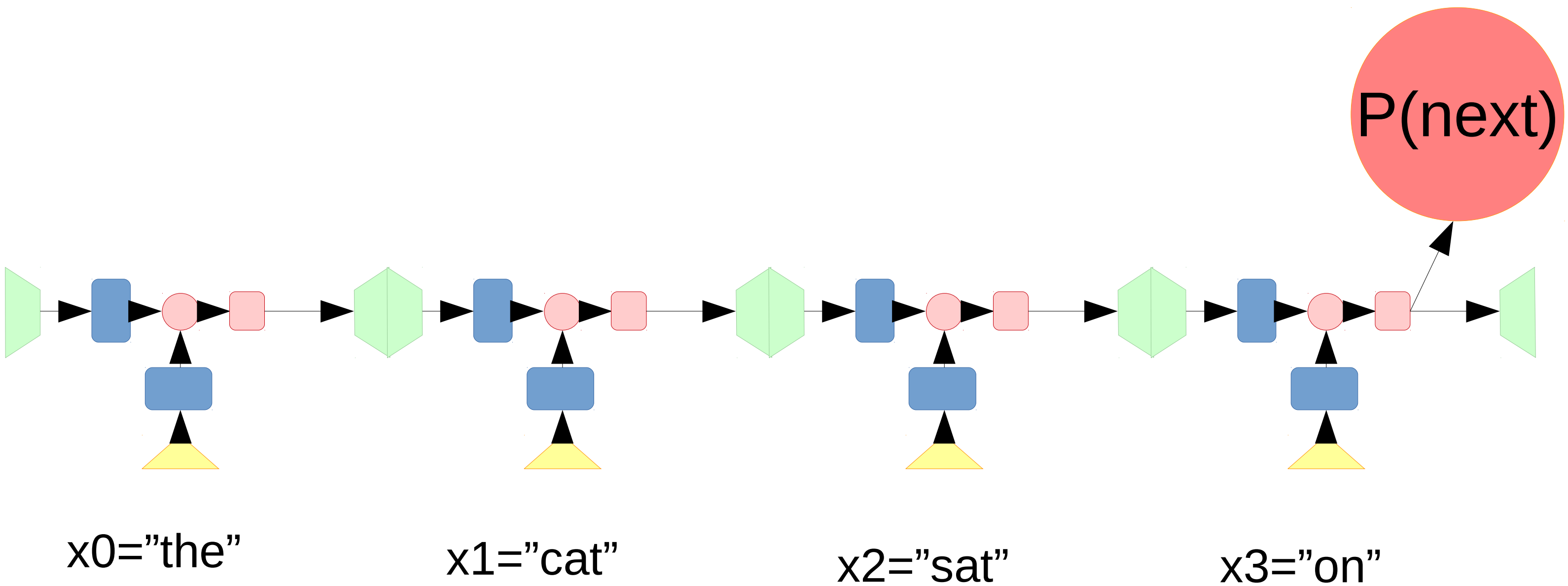


We use **same weight matrices** for all steps

Recurrent neural network

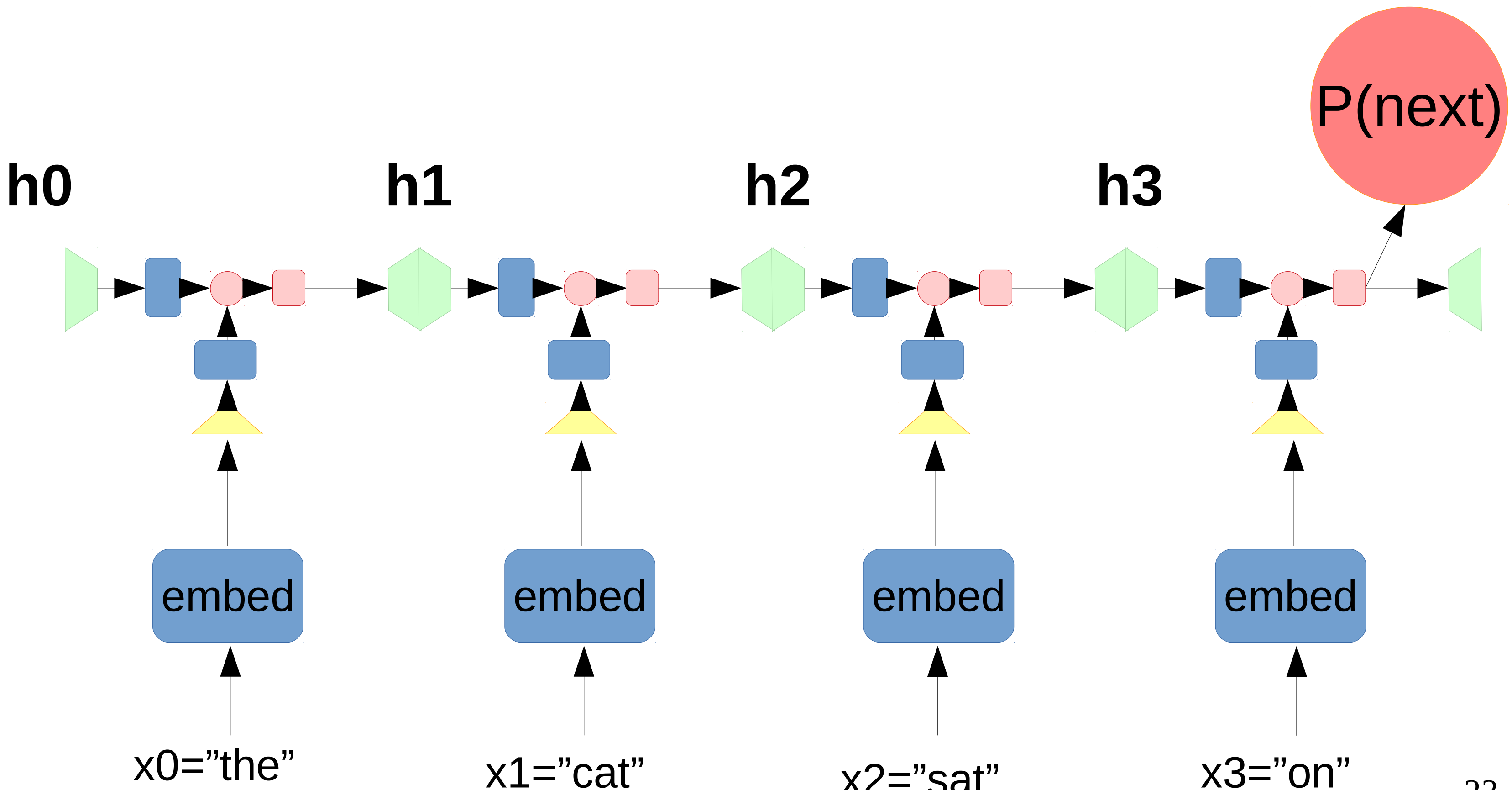


Recurrent neural network



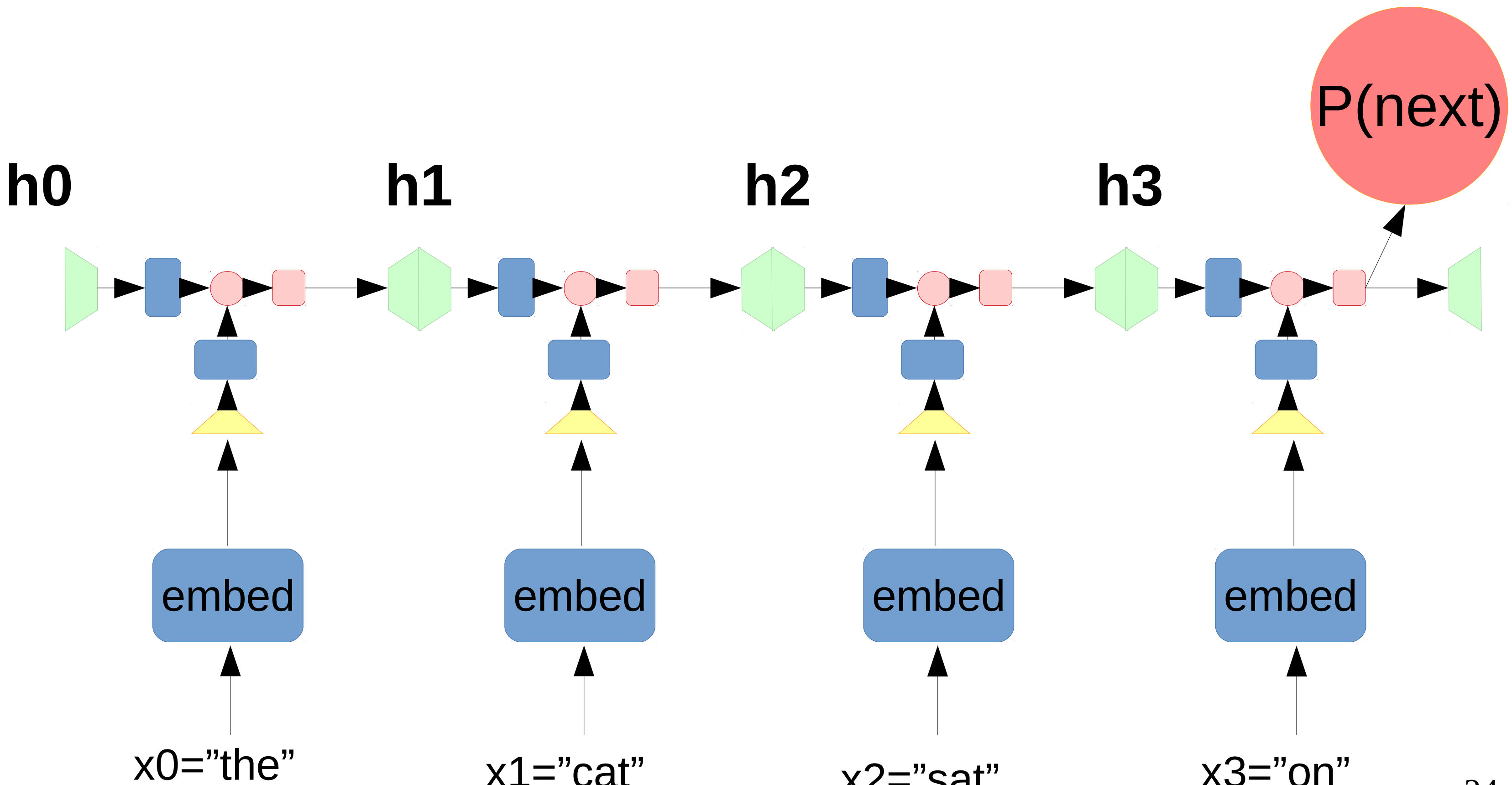
How can we represent words?

Recurrent neural network



Recurrent neural network

$$h_0 = \bar{0}$$
$$h_1 = \sigma(W_{hid} \cdot h_0 + W_{inp} \cdot x_0 + b)$$
$$h_2 = ?$$



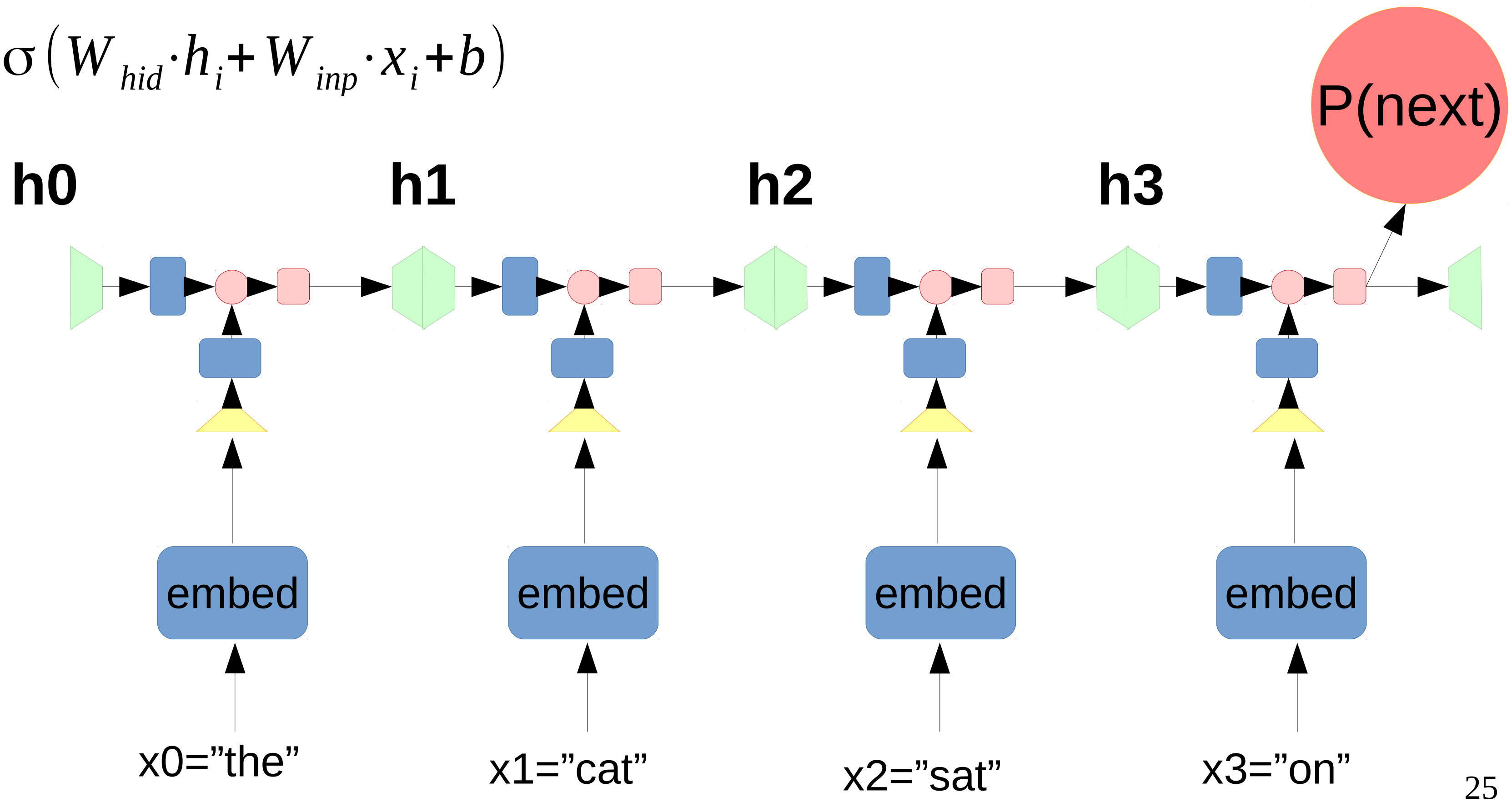
Recurrent neural network

$$h_0 = \bar{0}$$

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

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

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



Recurrent neural network

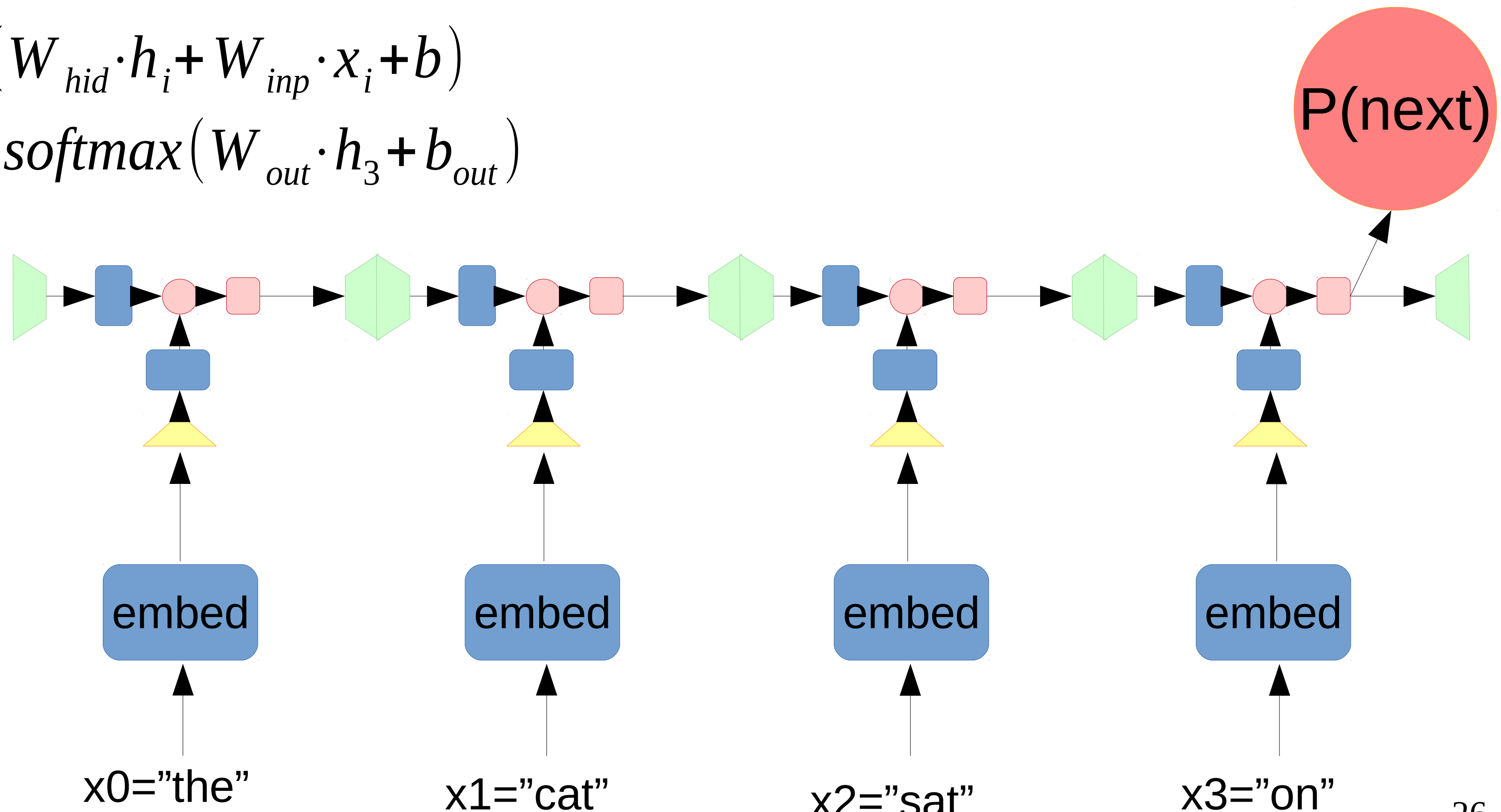
$$h_0 = \bar{0}$$

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

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

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

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

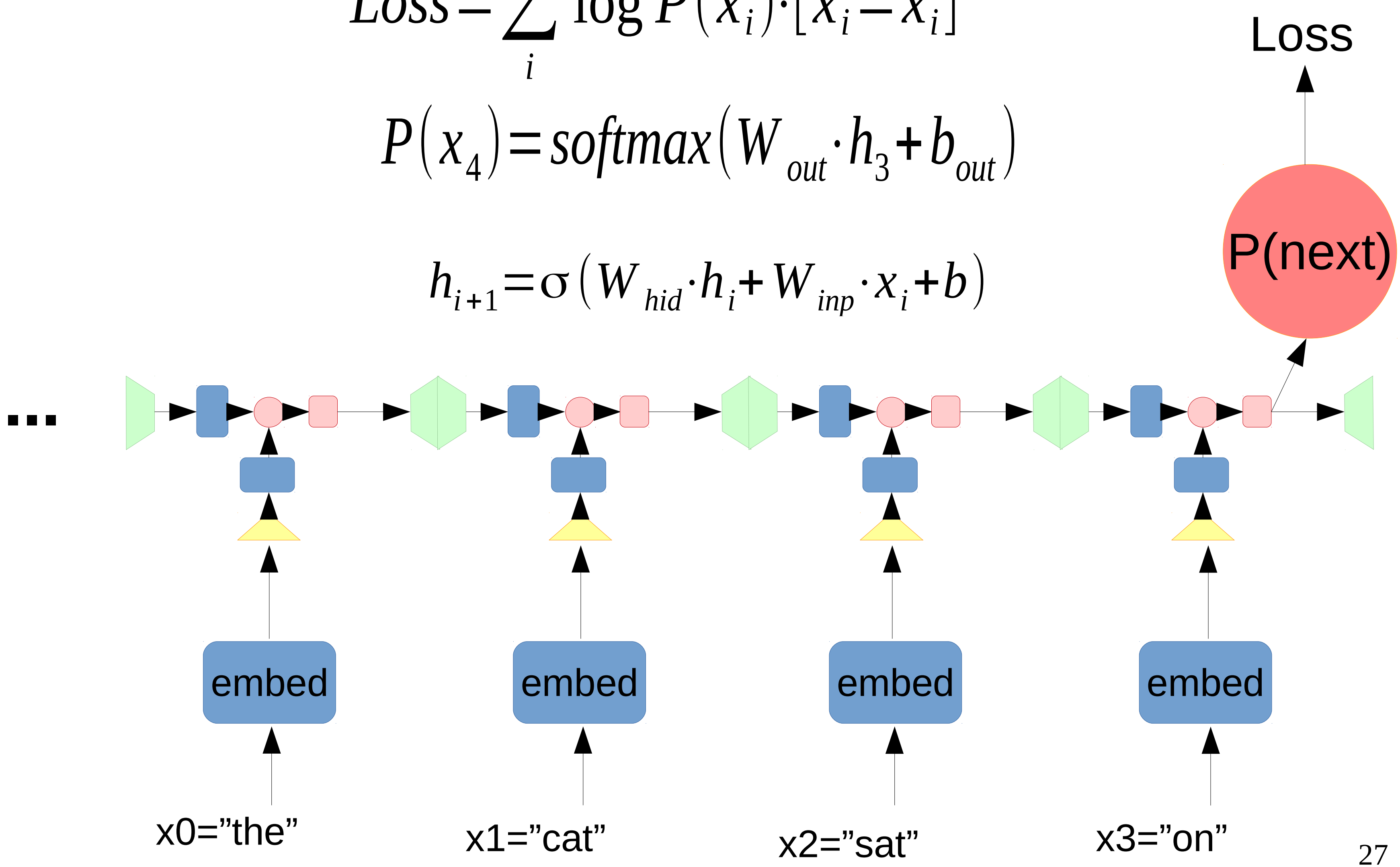


Recurrent neural network

$$Loss = \sum_i \log P(x_i) \cdot [x_i = x_i]$$

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

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



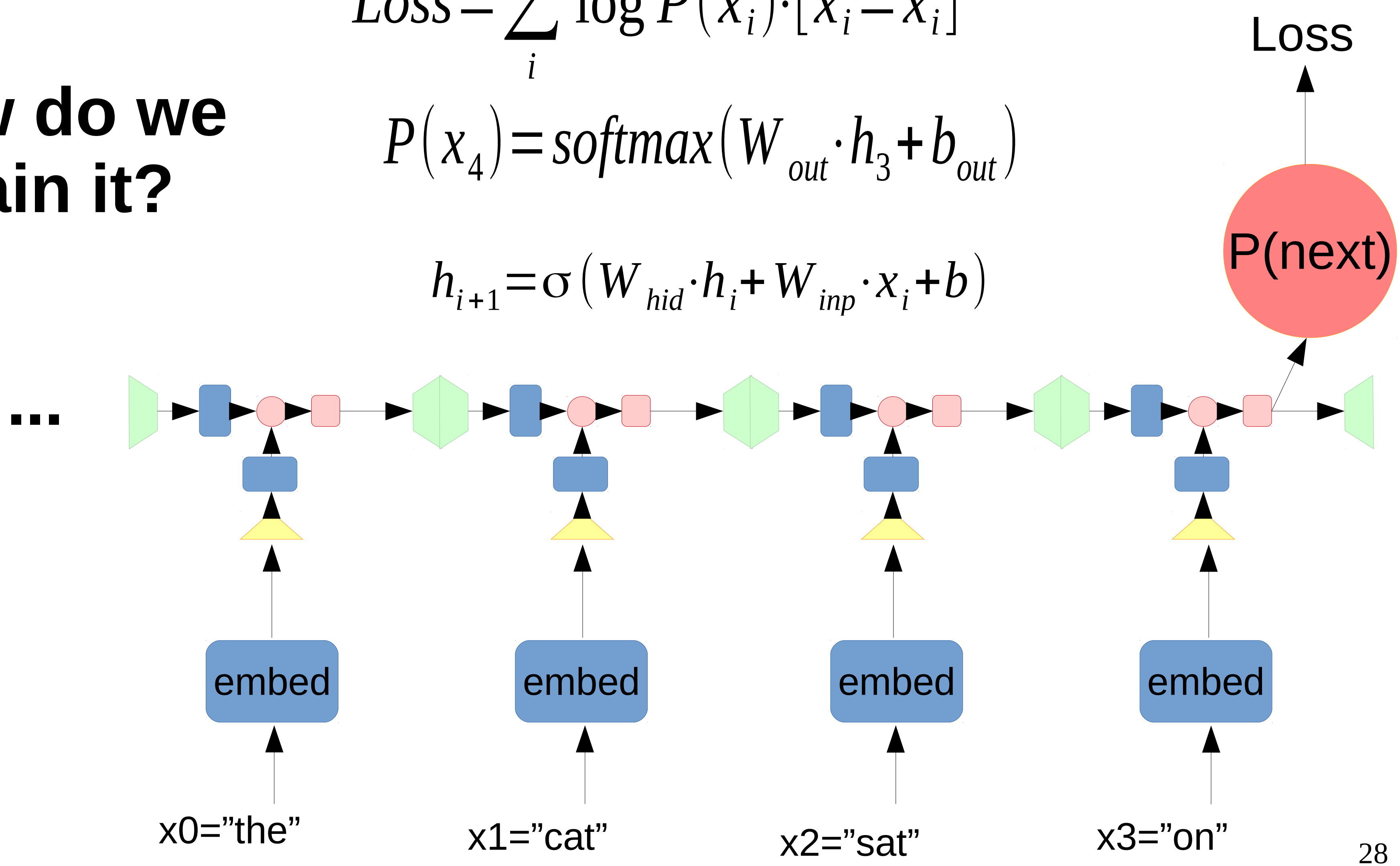
Recurrent neural network

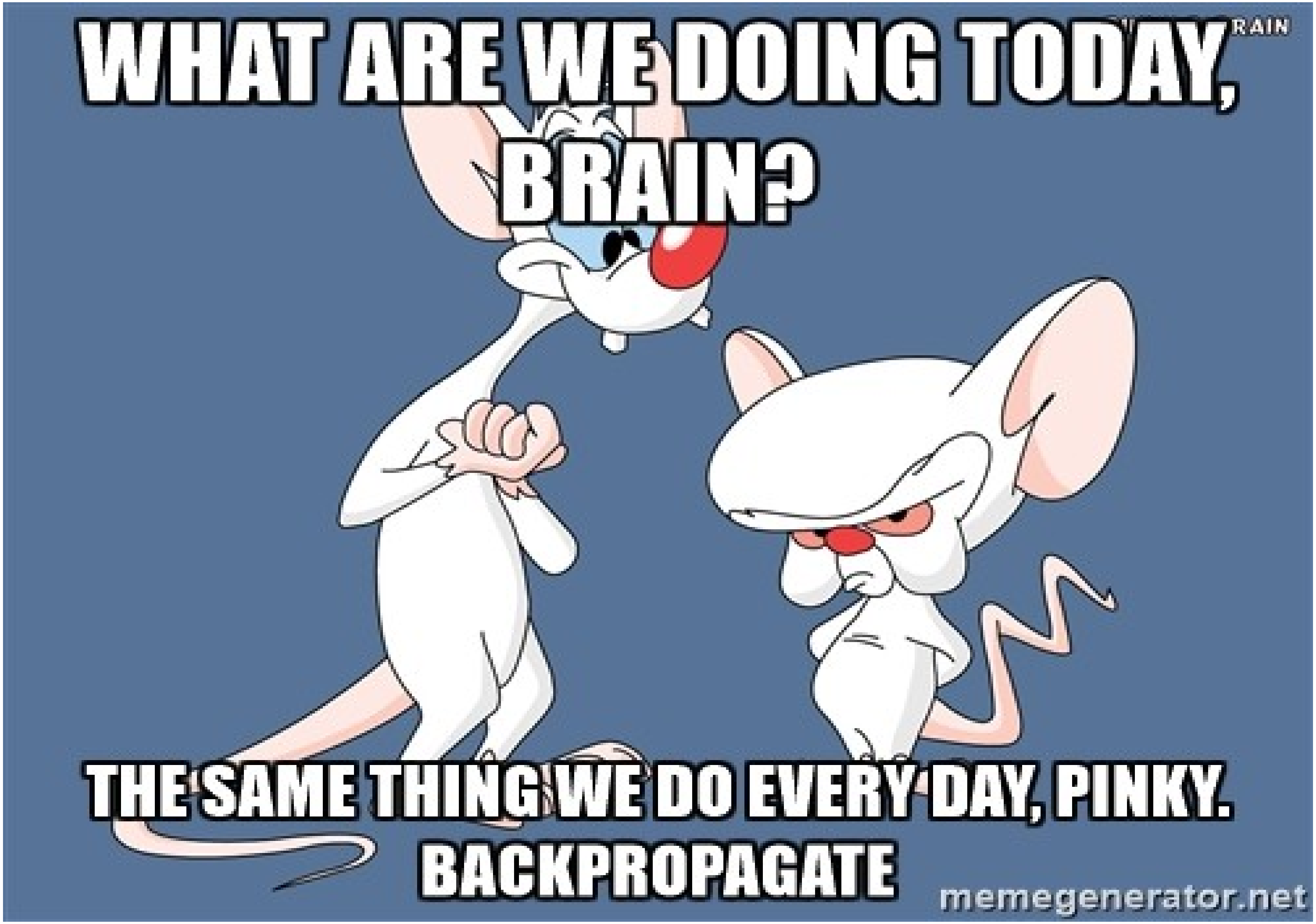
$$Loss = \sum_i \log P(x_i) \cdot [x_i = x_i]$$

How do we train it?

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

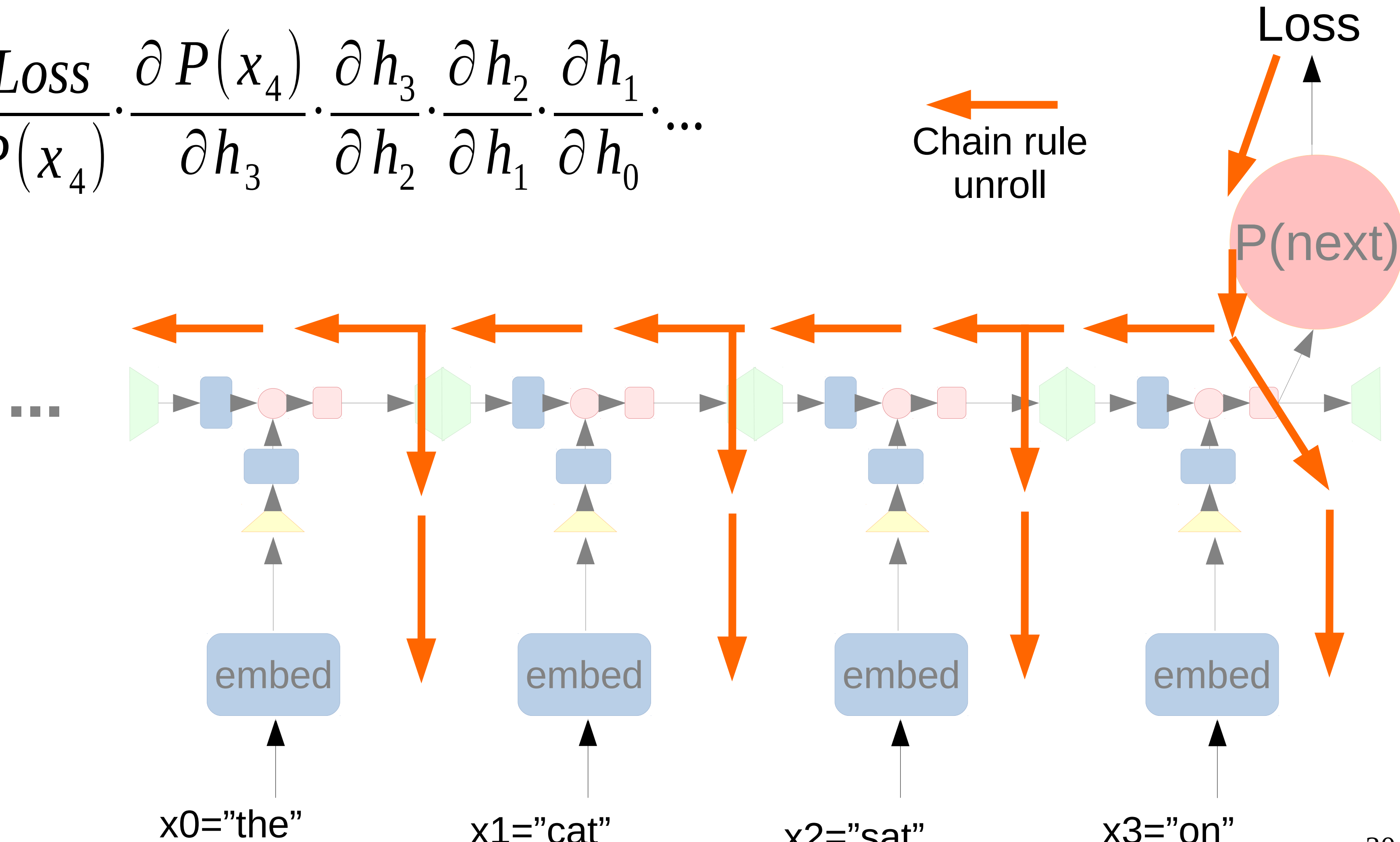
$$h_{i+1} = \sigma(W_{hid} \cdot h_i + W_{inp} \cdot x_i + b)$$




WHAT ARE WE DOING TODAY, BRAIN?
THE SAME THING WE DO EVERY DAY, PINKY.
BACKPROPAGATE
memegenerator.net

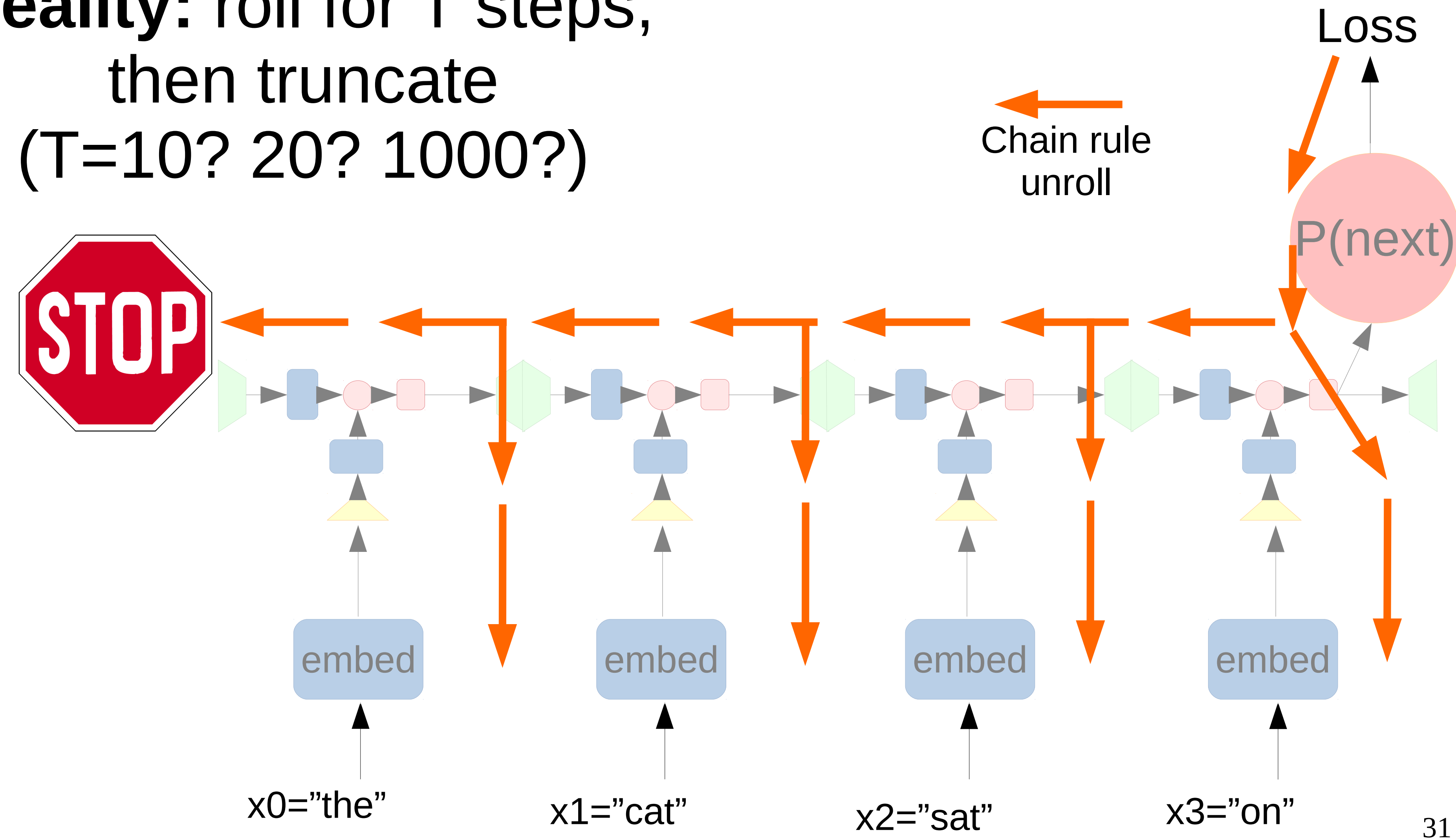
Backpropagation through time

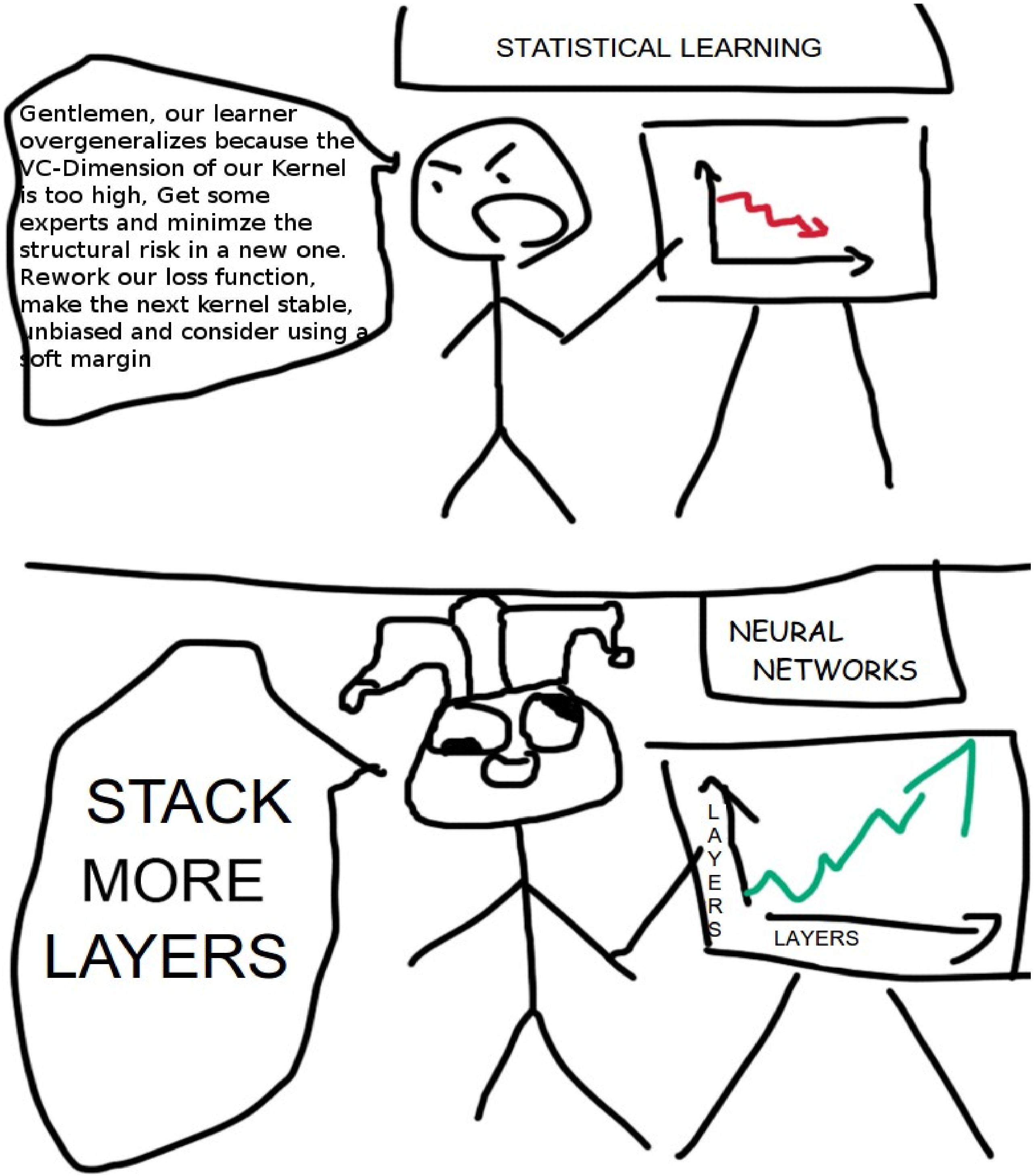
$$\frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0} \cdot \ldots$$



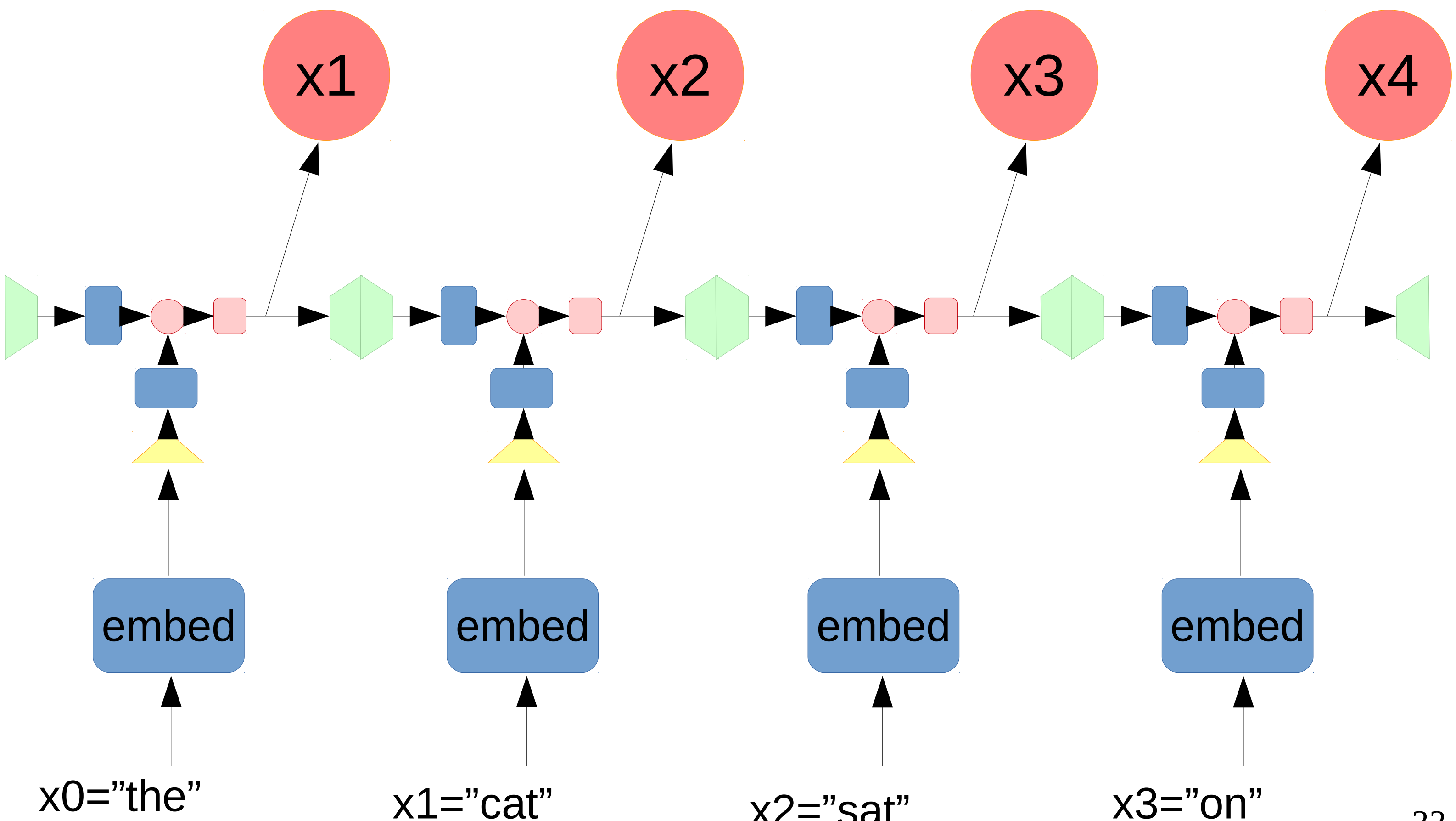
Truncated BPTT

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



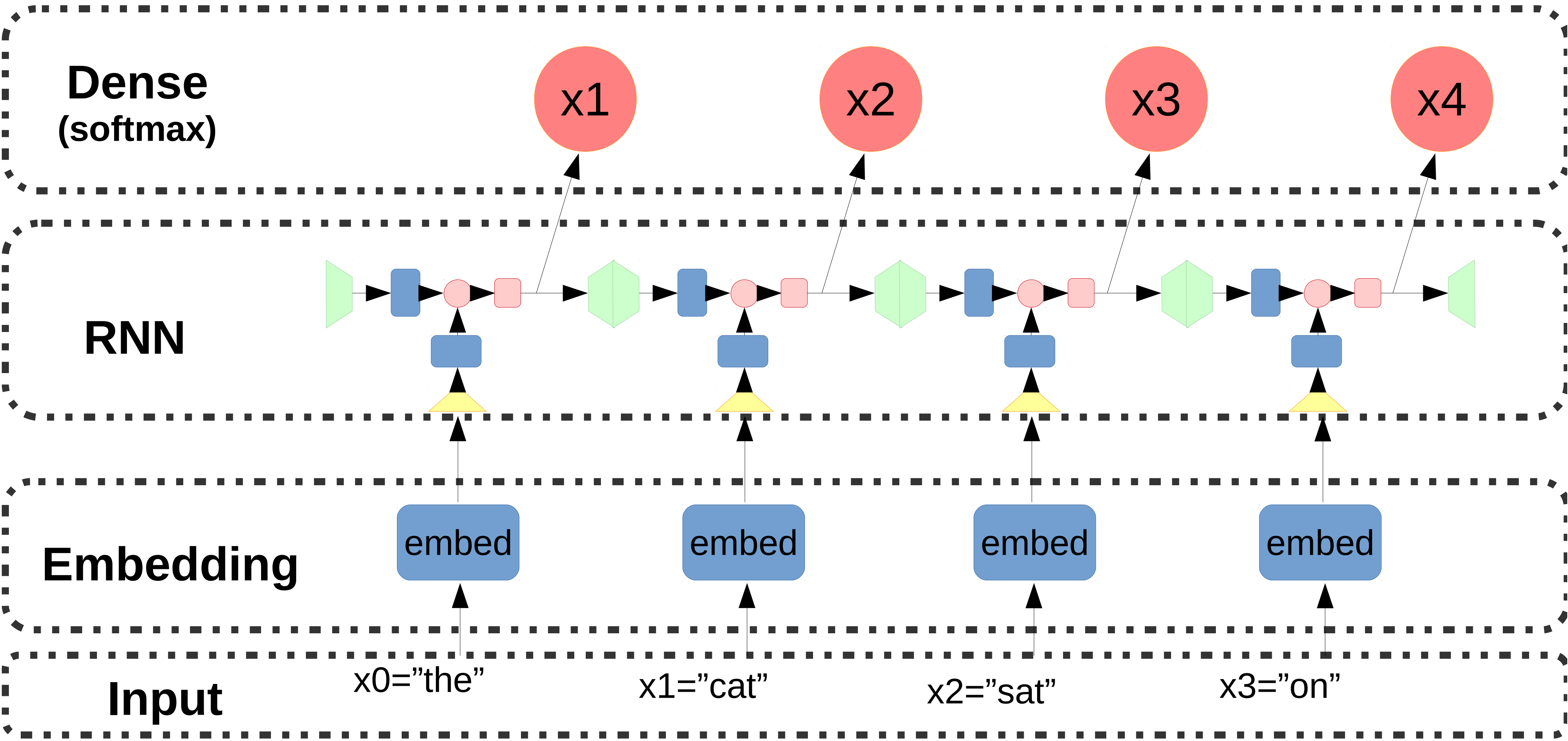

STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS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

What is layer, again?

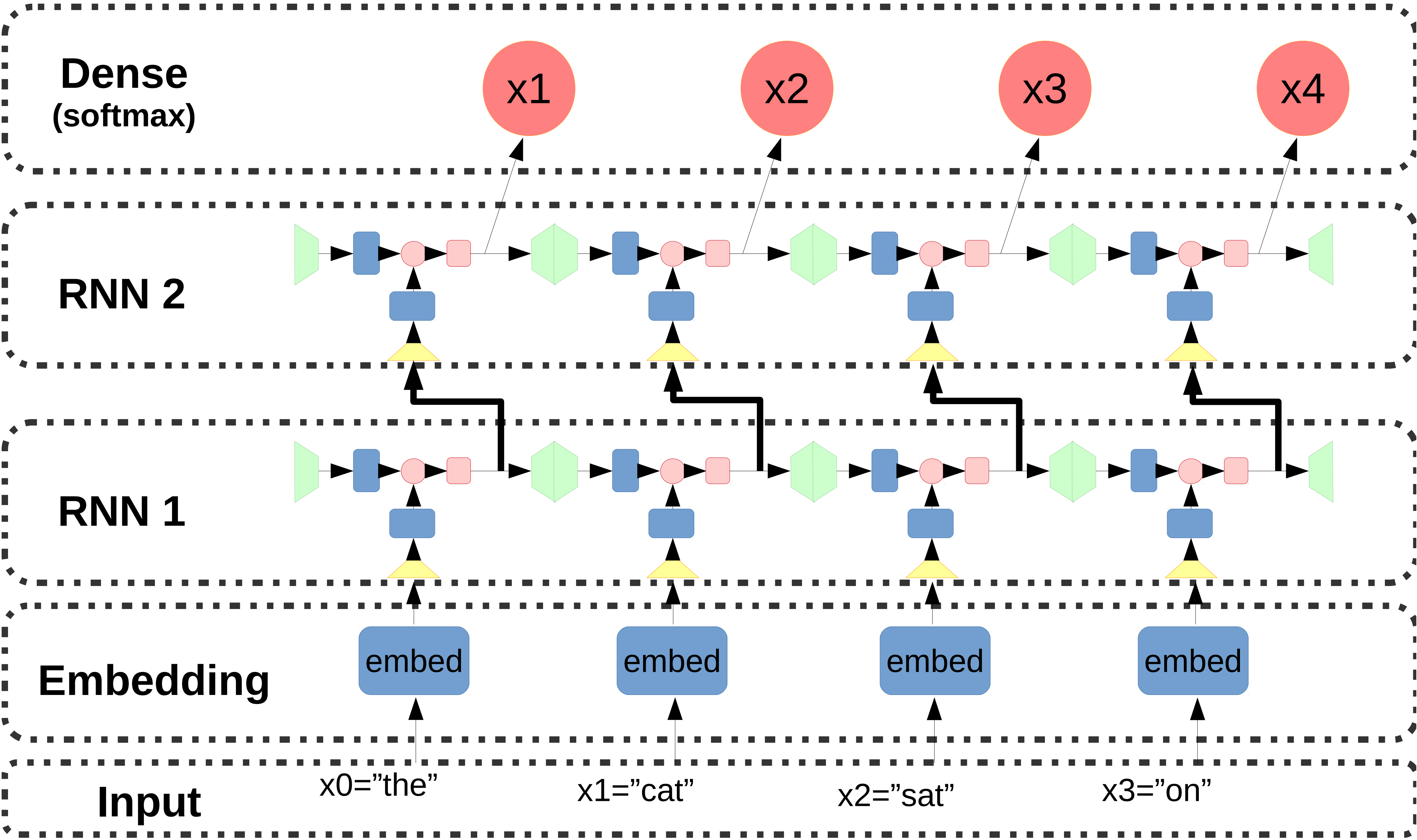


Layers

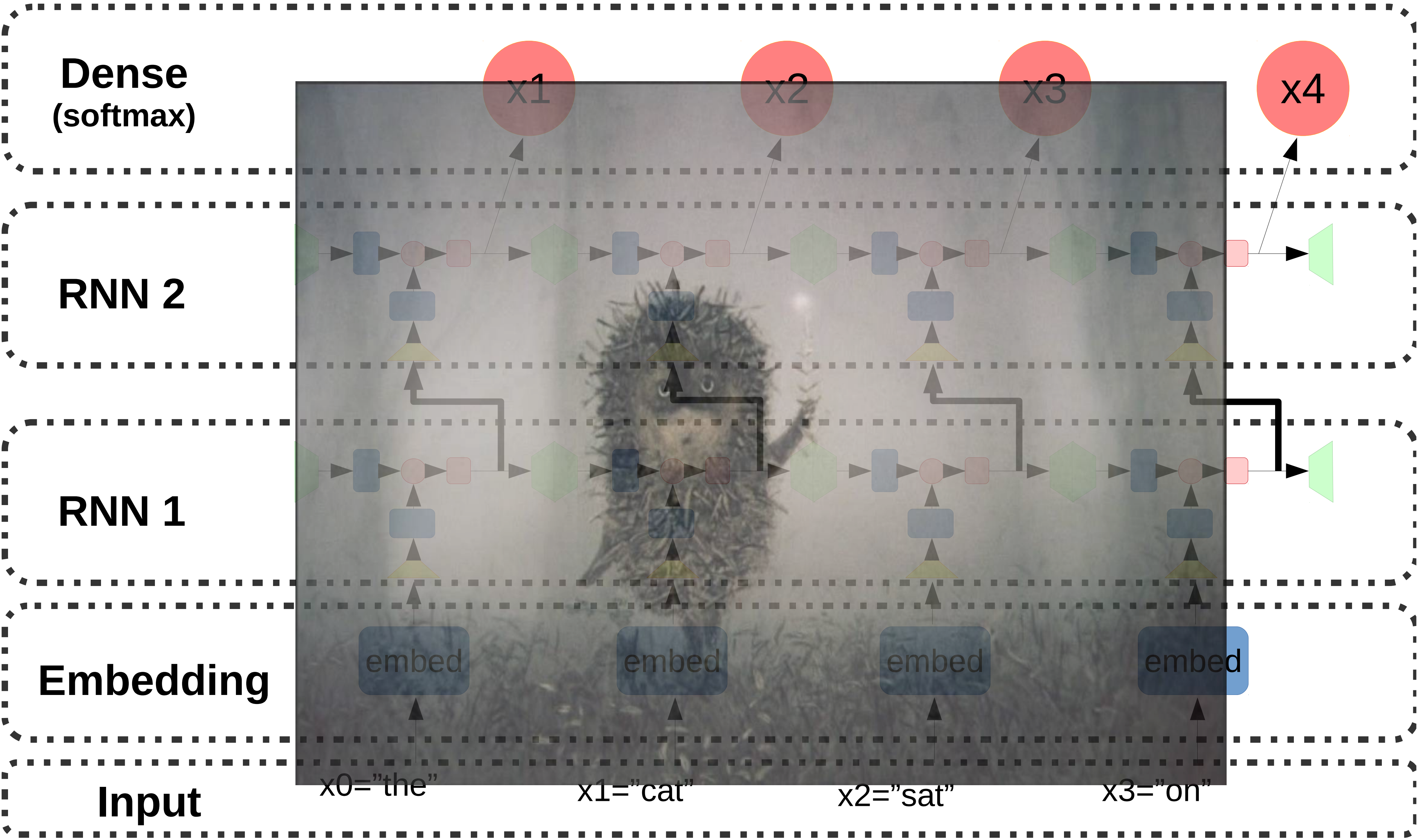
Where to stick more layers?

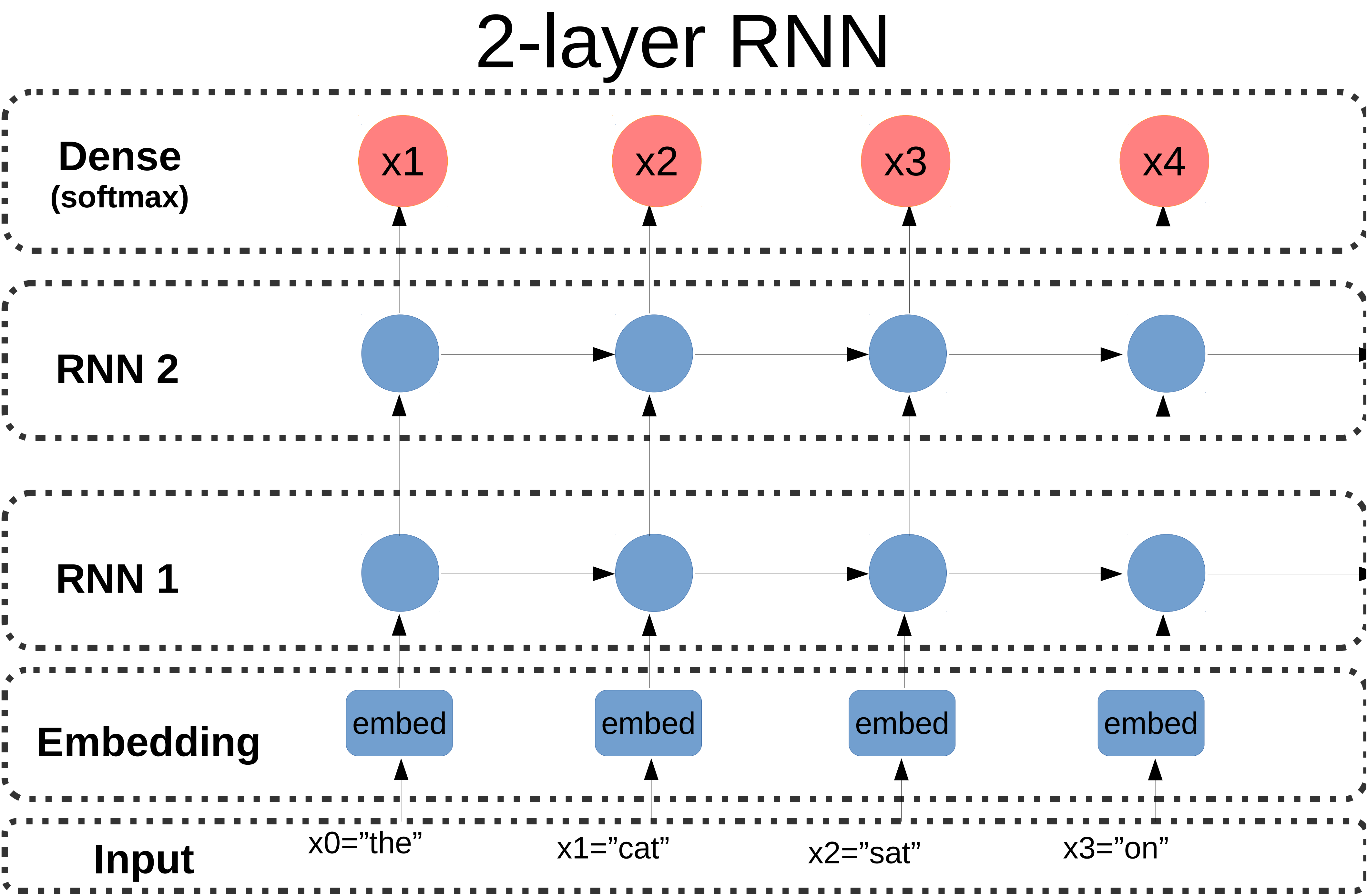


More layers

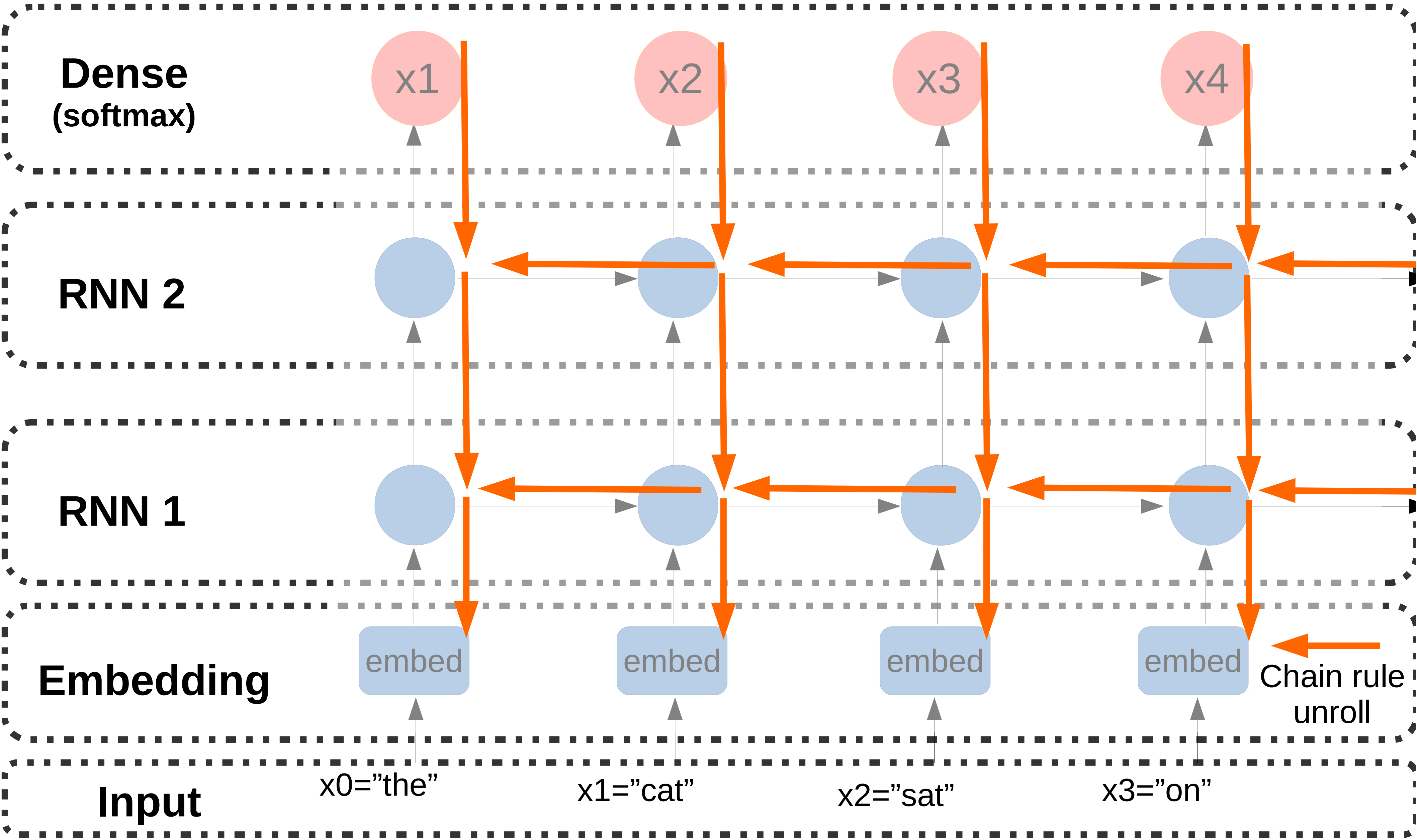

Dense
(softmax)
x1
x2
x3
x4
RNN 2
RNN 1
Embedding
embed
embed
embed
embed
Input
x0="the"
x1="cat"
x2="sat"
x3="on"

Too f**king complicated




2-layer RNN
Dense
(softmax)
x1
x2
x3
x4
RNN 2
RNN 1
Embedding
embed
embed
embed
embed
Input
x0=”the”
x1=”cat”
x2=”sat”
x3=”on”

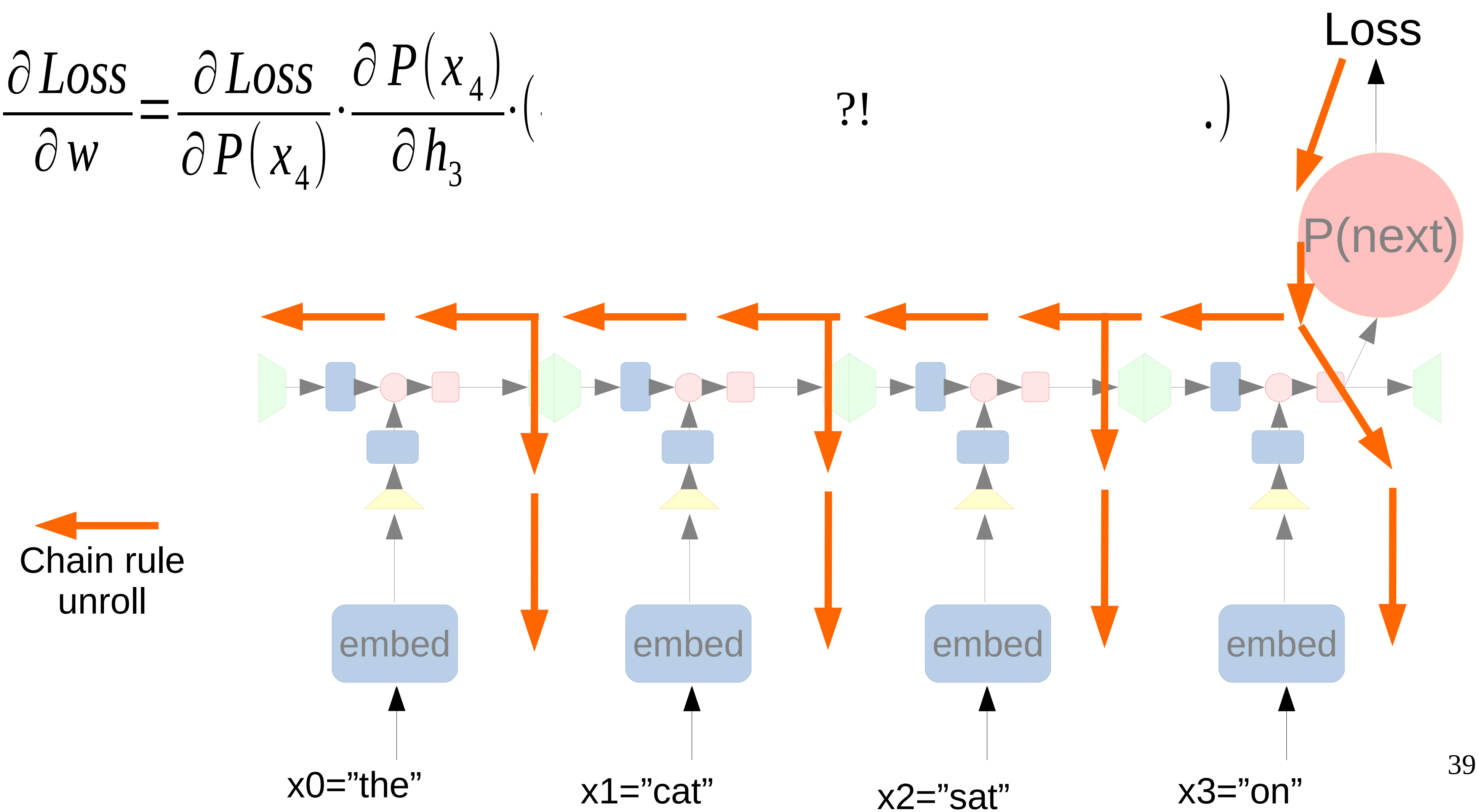
BPTT again


Dense
(softmax)
x1
x2
x3
x4
RNN 2
RNN 1
Embedding
embed
embed
embed
embed
Chain rule
unroll
Input
x0="the"
x1="cat"
x2="sat"
x3="on"

BPTT Again

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

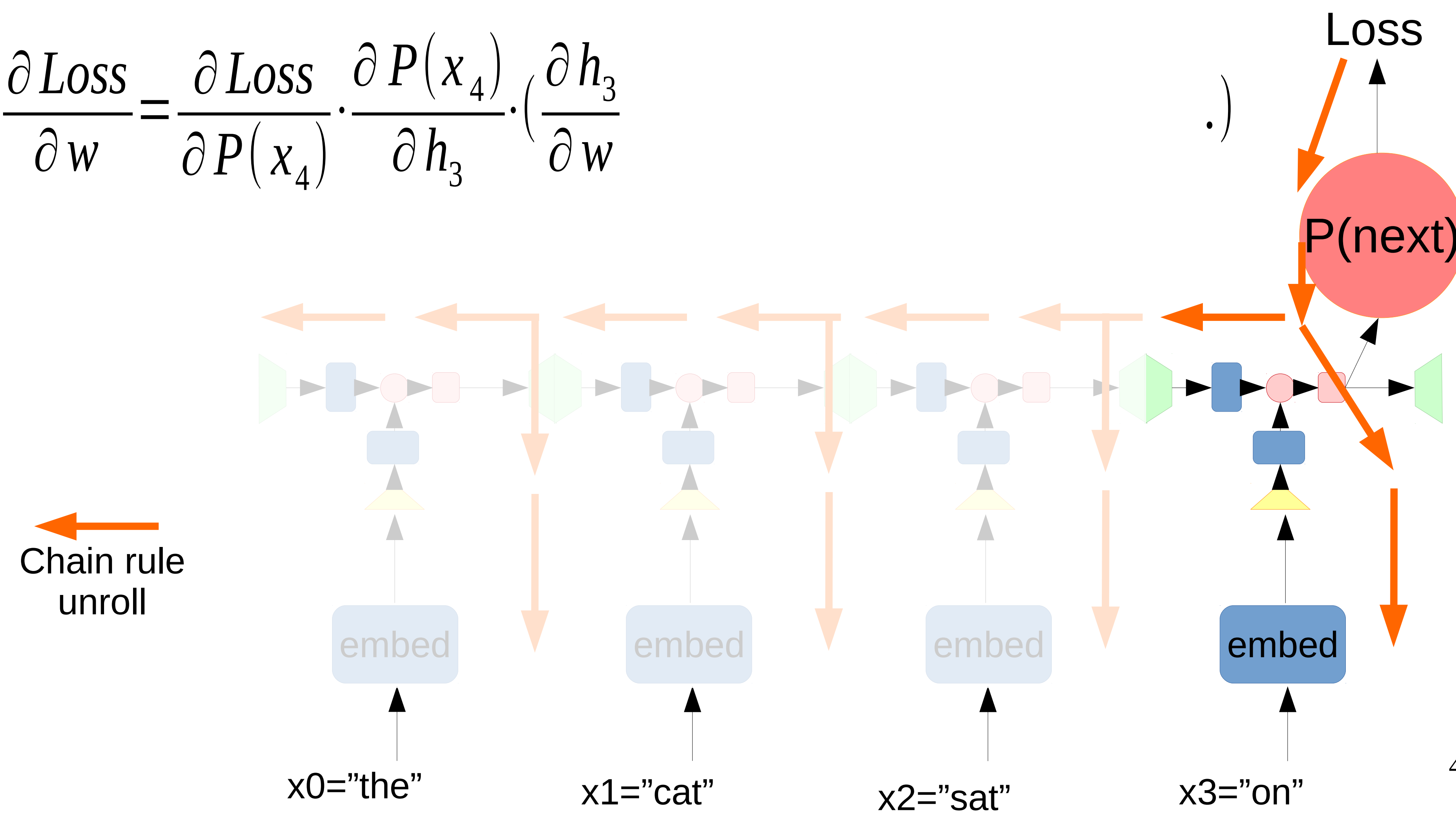
$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot (\qquad ?! \qquad .)$$



BPTT Again

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

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

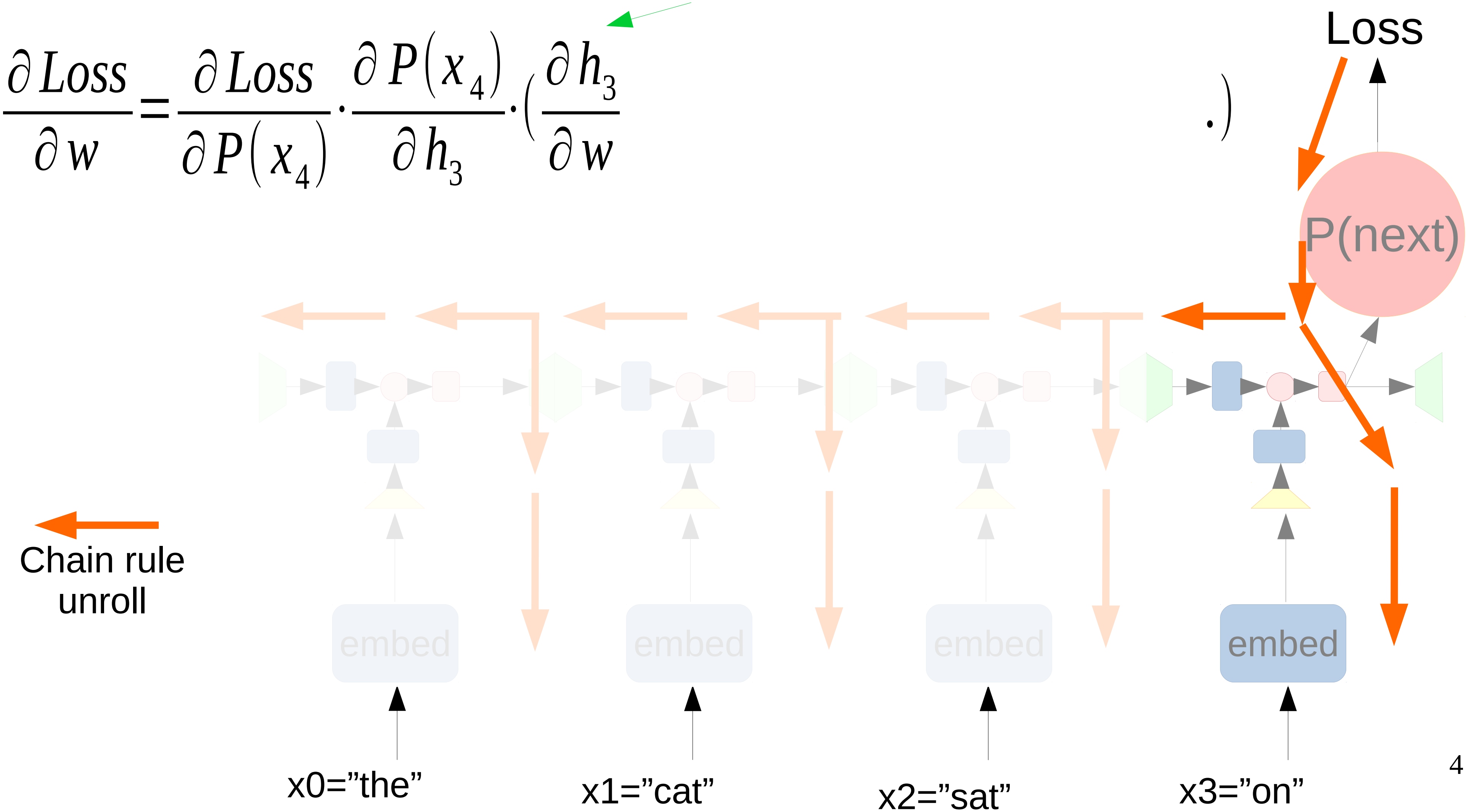


BPTT Again

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

consider h2 constant

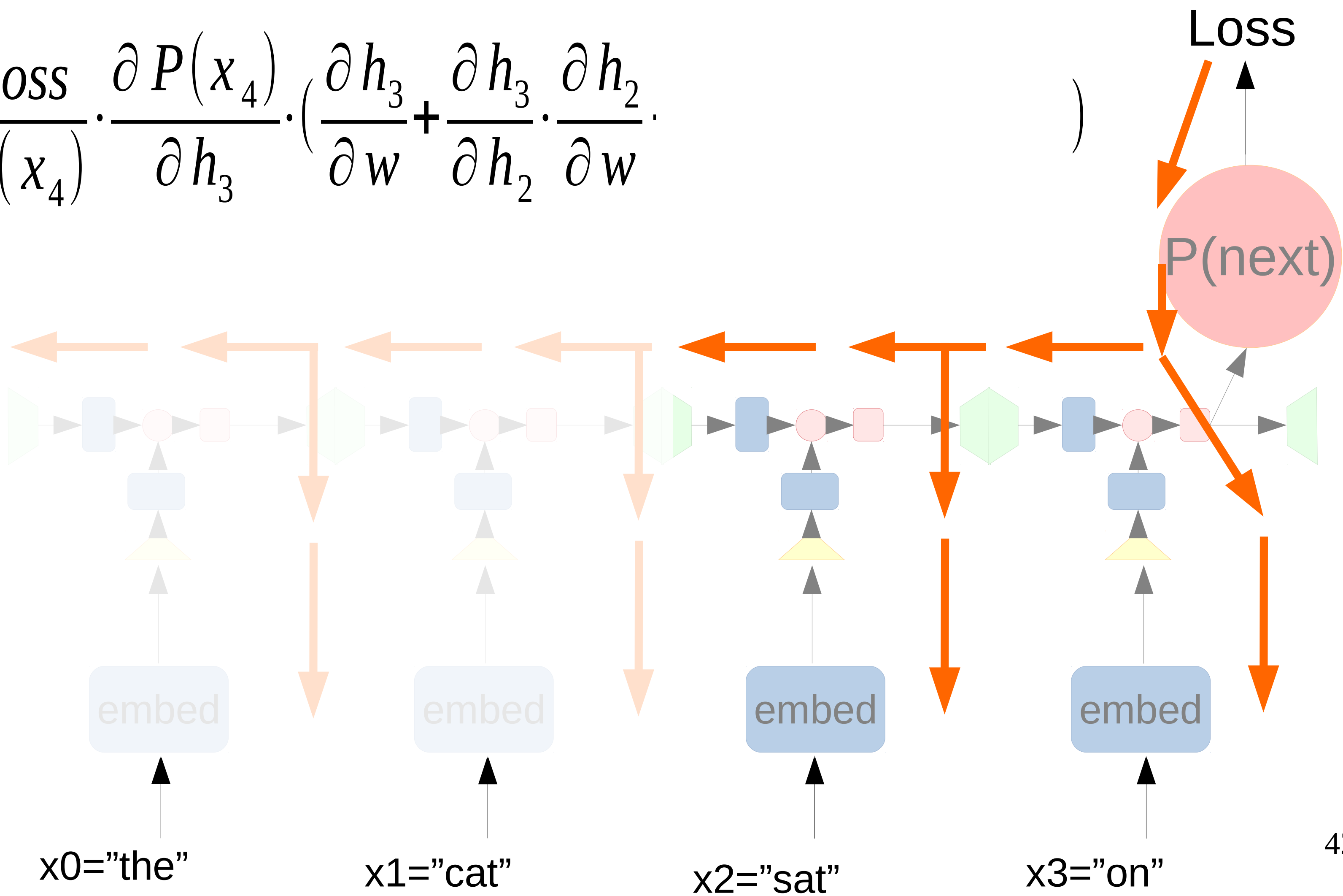
$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} \qquad \cdot\right)$$



BPTT Again

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

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

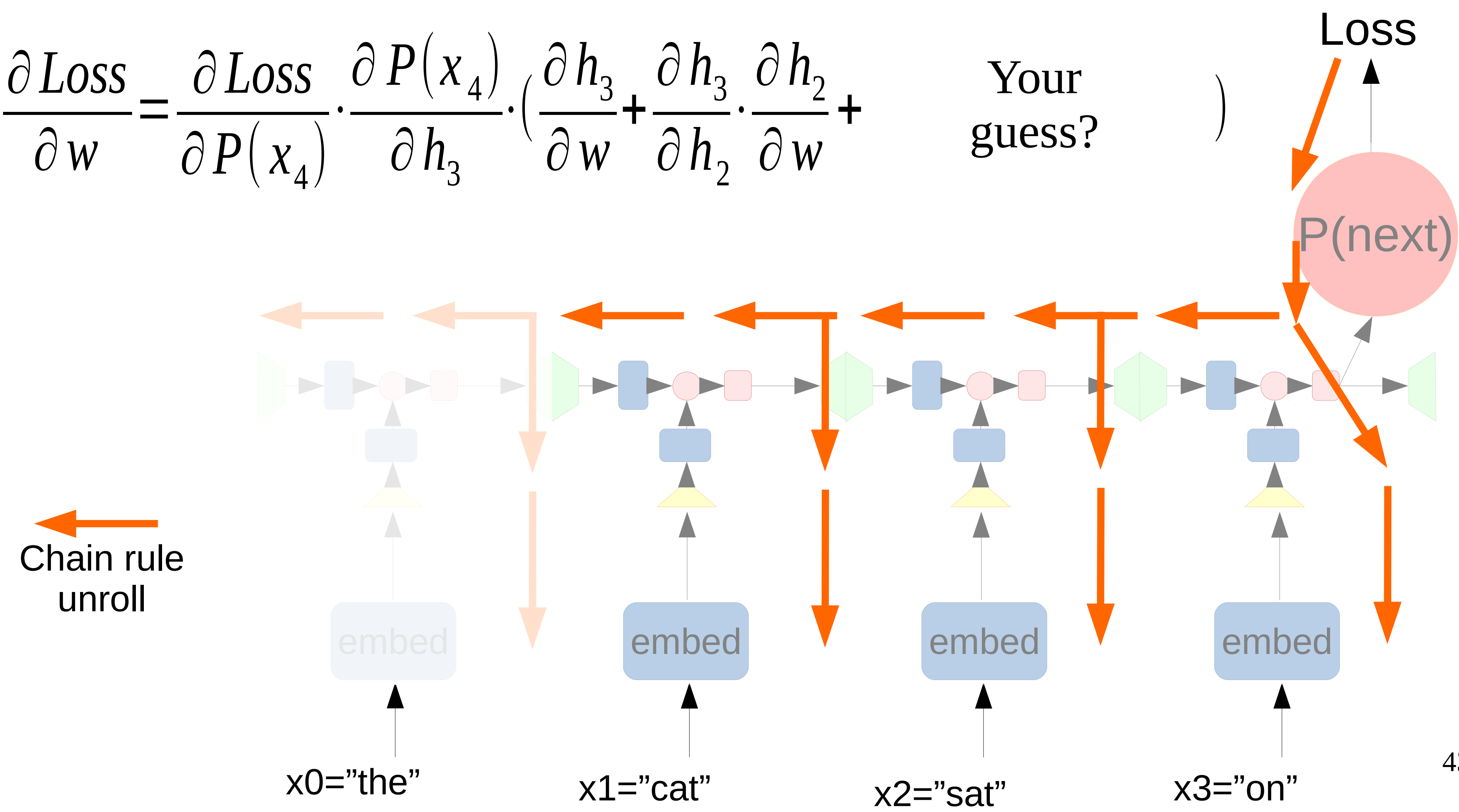


Chain rule unroll

BPTT Again

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

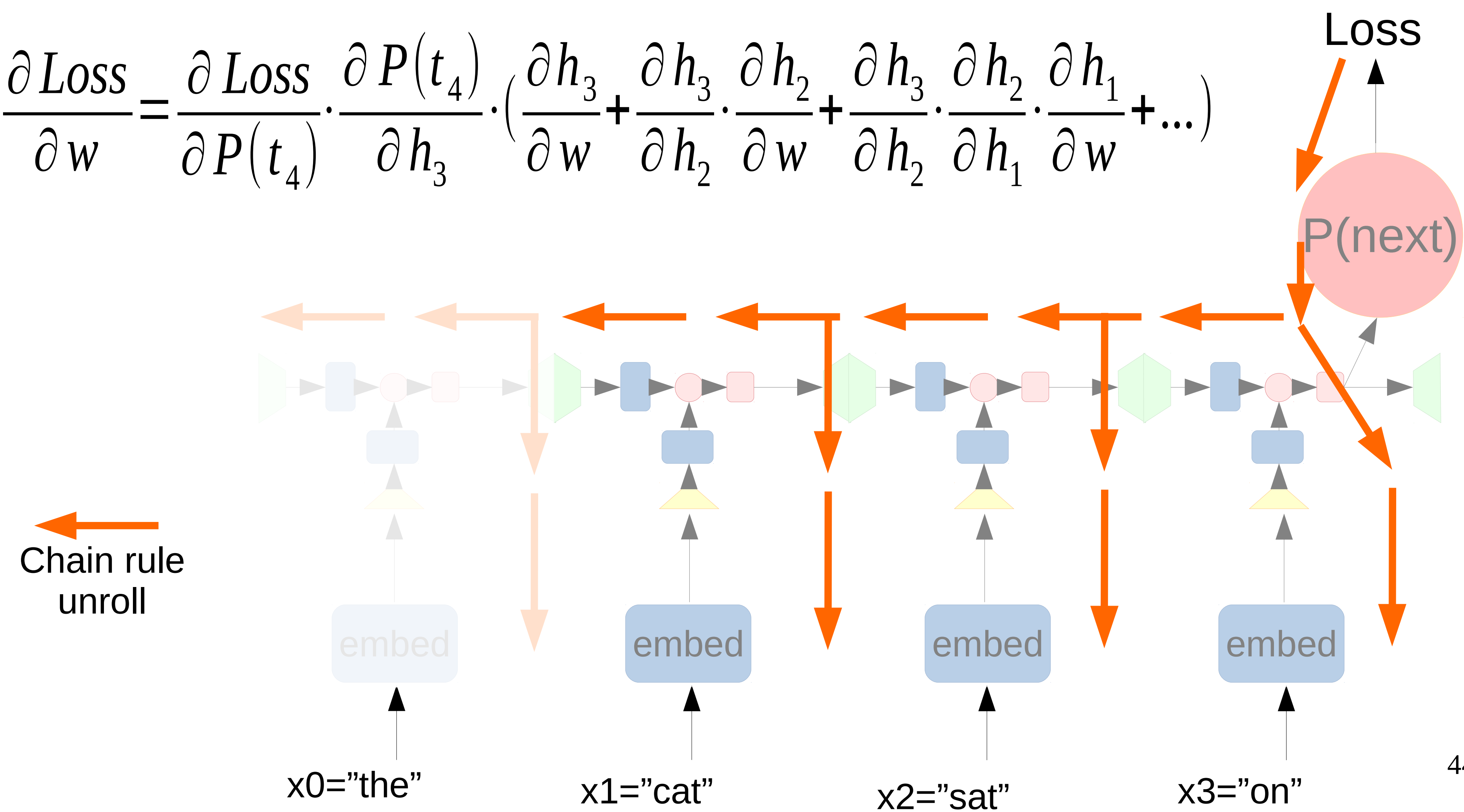
$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} + \quad \text{Your guess?} \quad \right)$$



BPTT Again

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

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



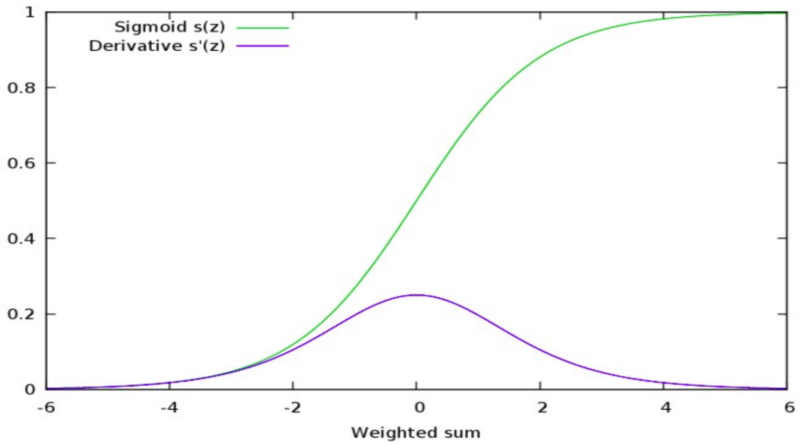
Gradient explosion and vanishing

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

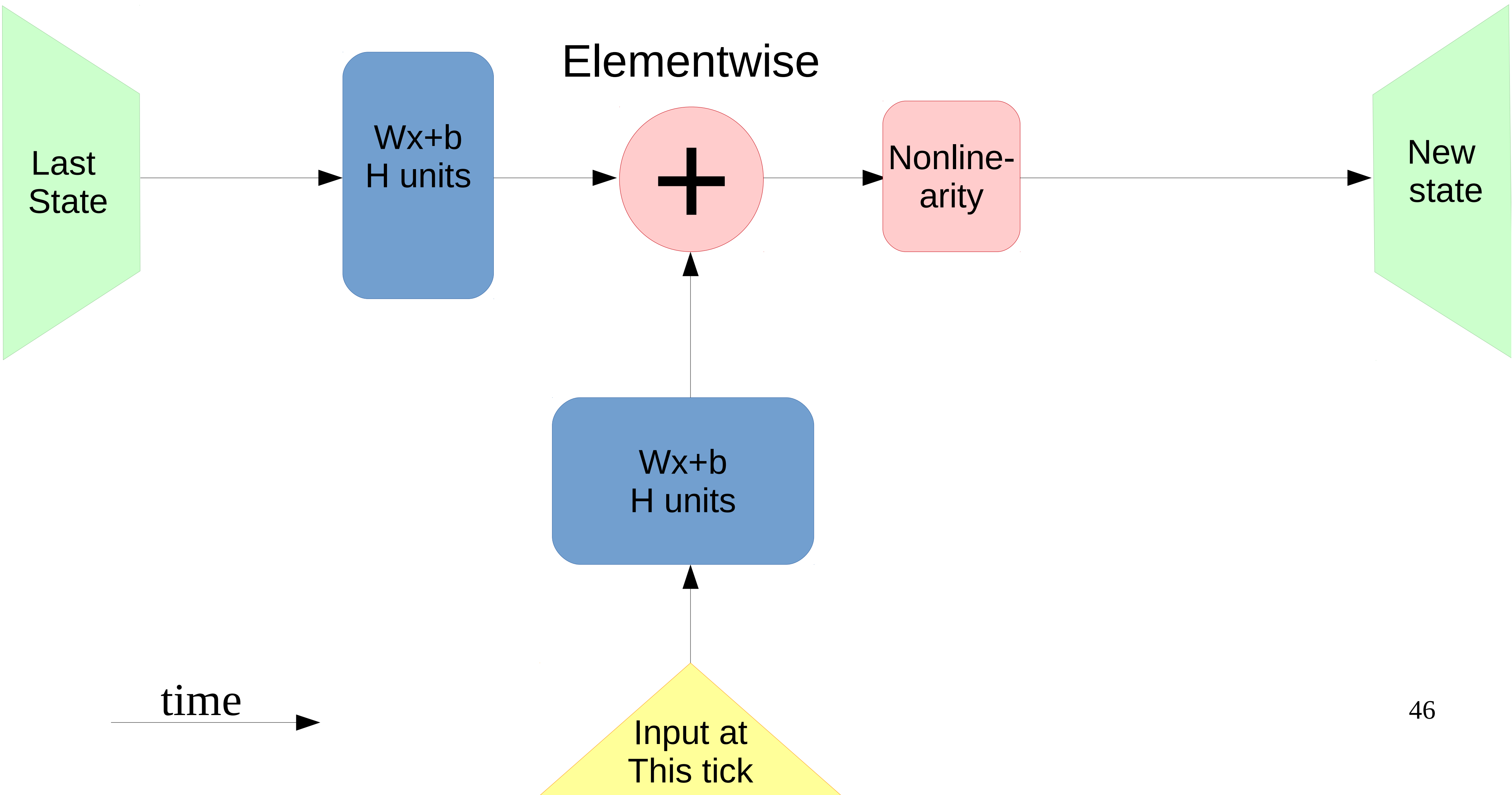
$$\frac{\partial Loss}{\partial w} = \frac{\partial Loss}{\partial P(x_4)} \cdot \frac{\partial P(x_4)}{\partial h_3} \cdot \left(\frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial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

- Many sigmoids near 0 or 1
  - Gradients → 0
  - Not training for long-term dependencies

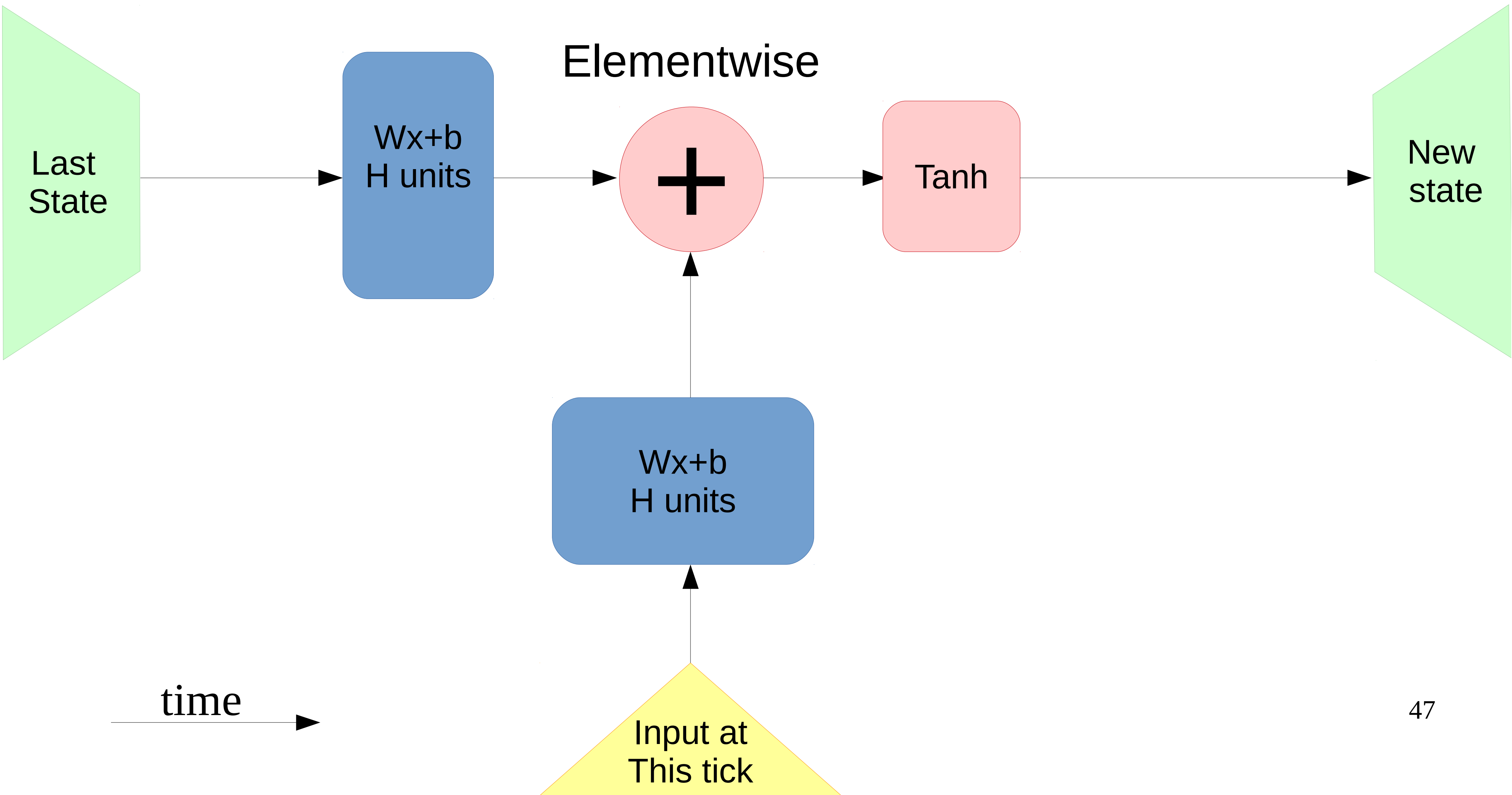
- Many nonzero values
  - Derivative stacks to >1
  - Gradients → inf
  - Weights → shit



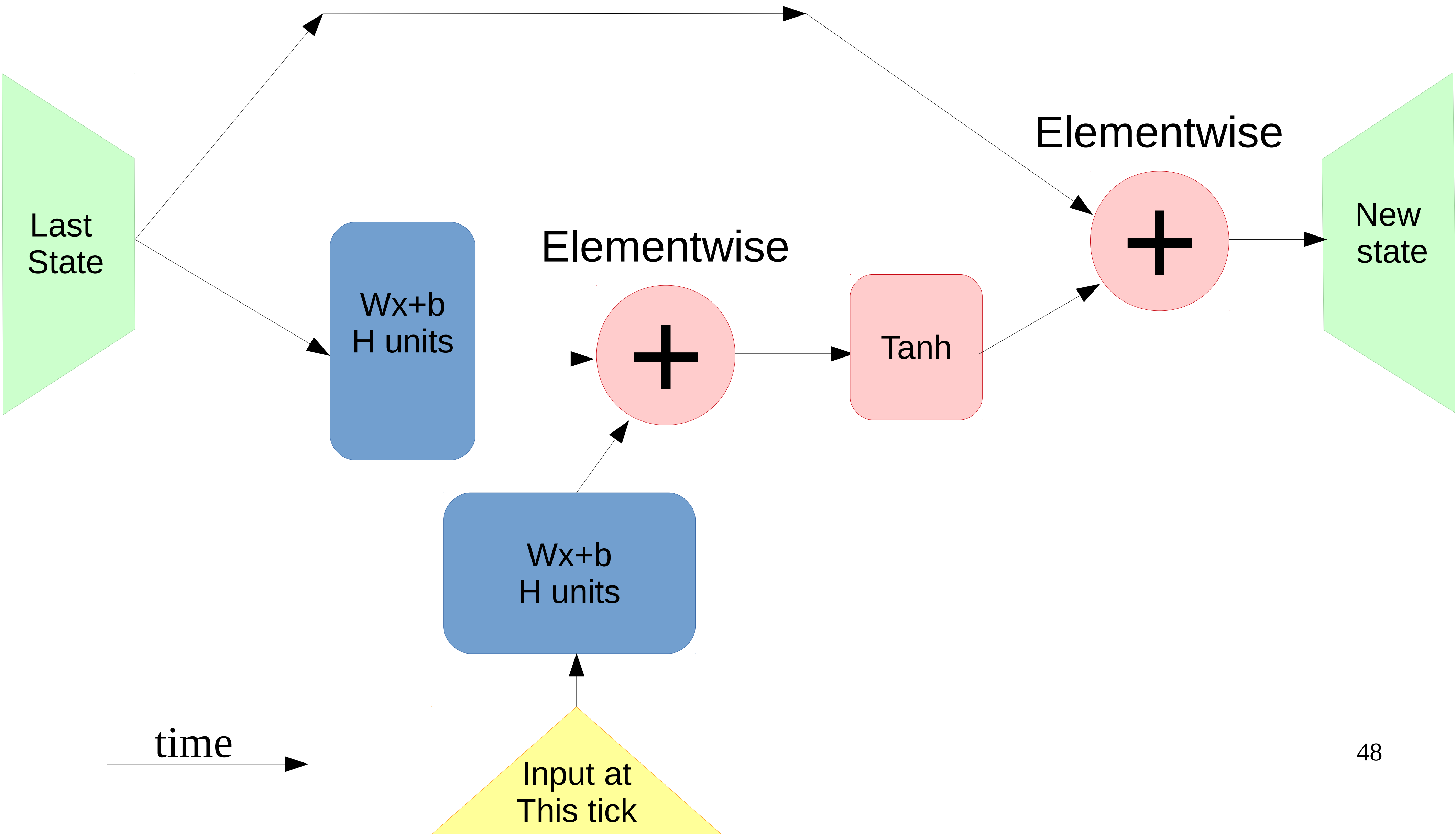
RNN step


Last
State
Wx+b
H units
Elementwise
+
Nonline-
arity
New
state
Wx+b
H units
Input at
This tick
time

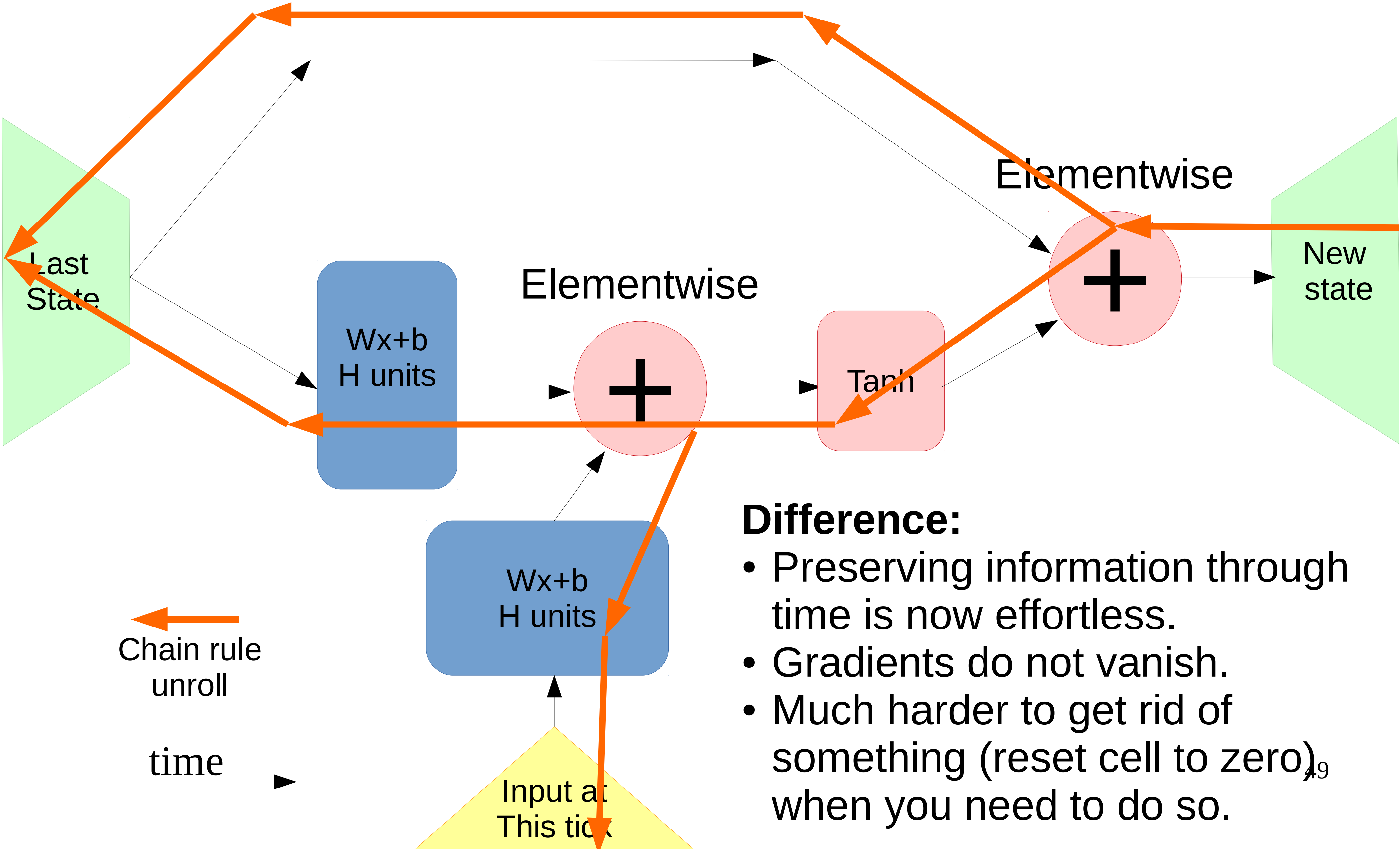
RNN step


Last
State
Wx+b
H units
Elementwise
+
Tanh
New
state
Wx+b
H units
Input at
This tick
time

Residual RNN step



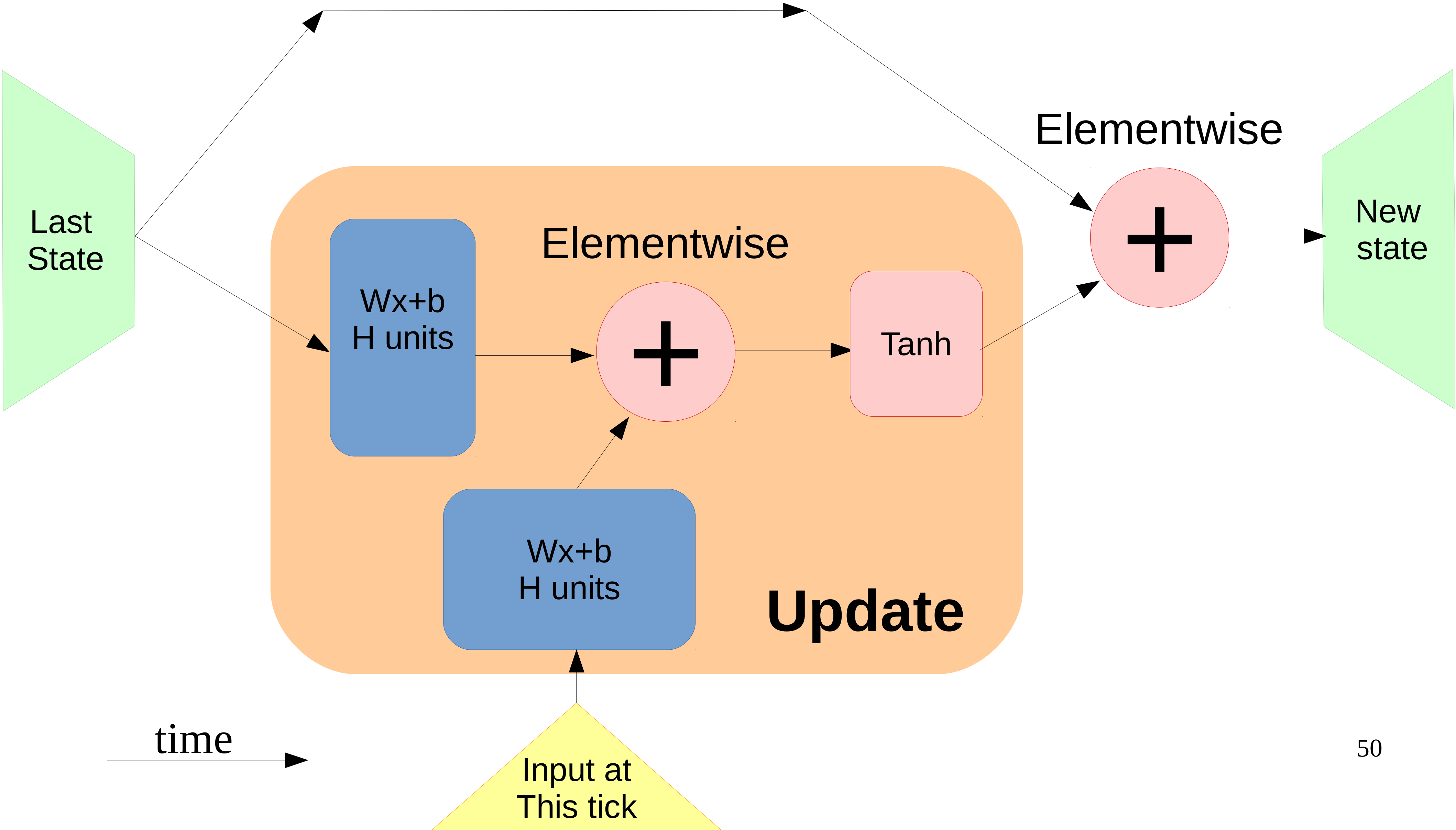
Residual RNN step



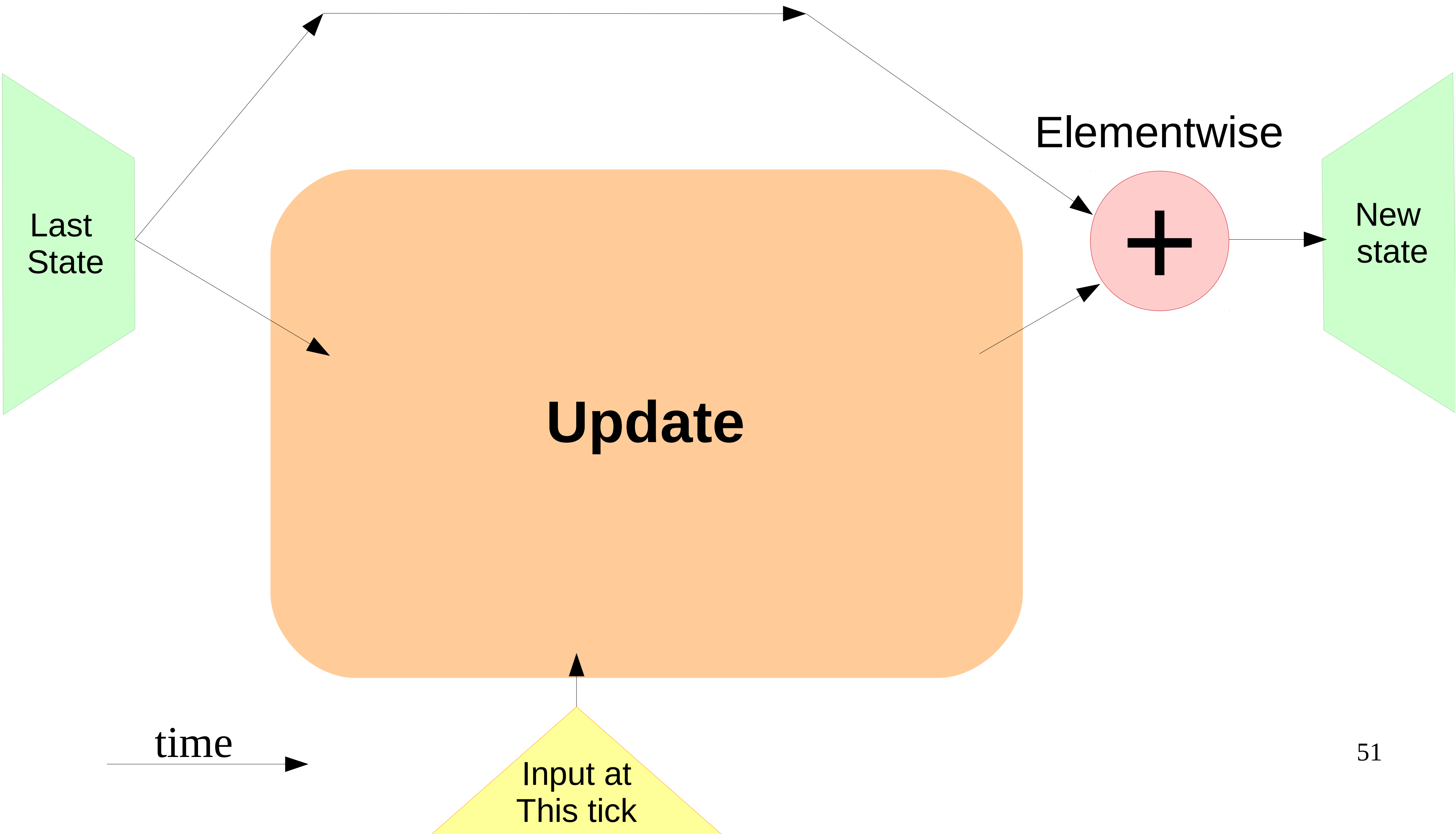
Difference:

- Preserving information through time is now effortless.
- Gradients do not vanish.
- Much harder to get rid of something (reset cell to zero) when you need to do so.

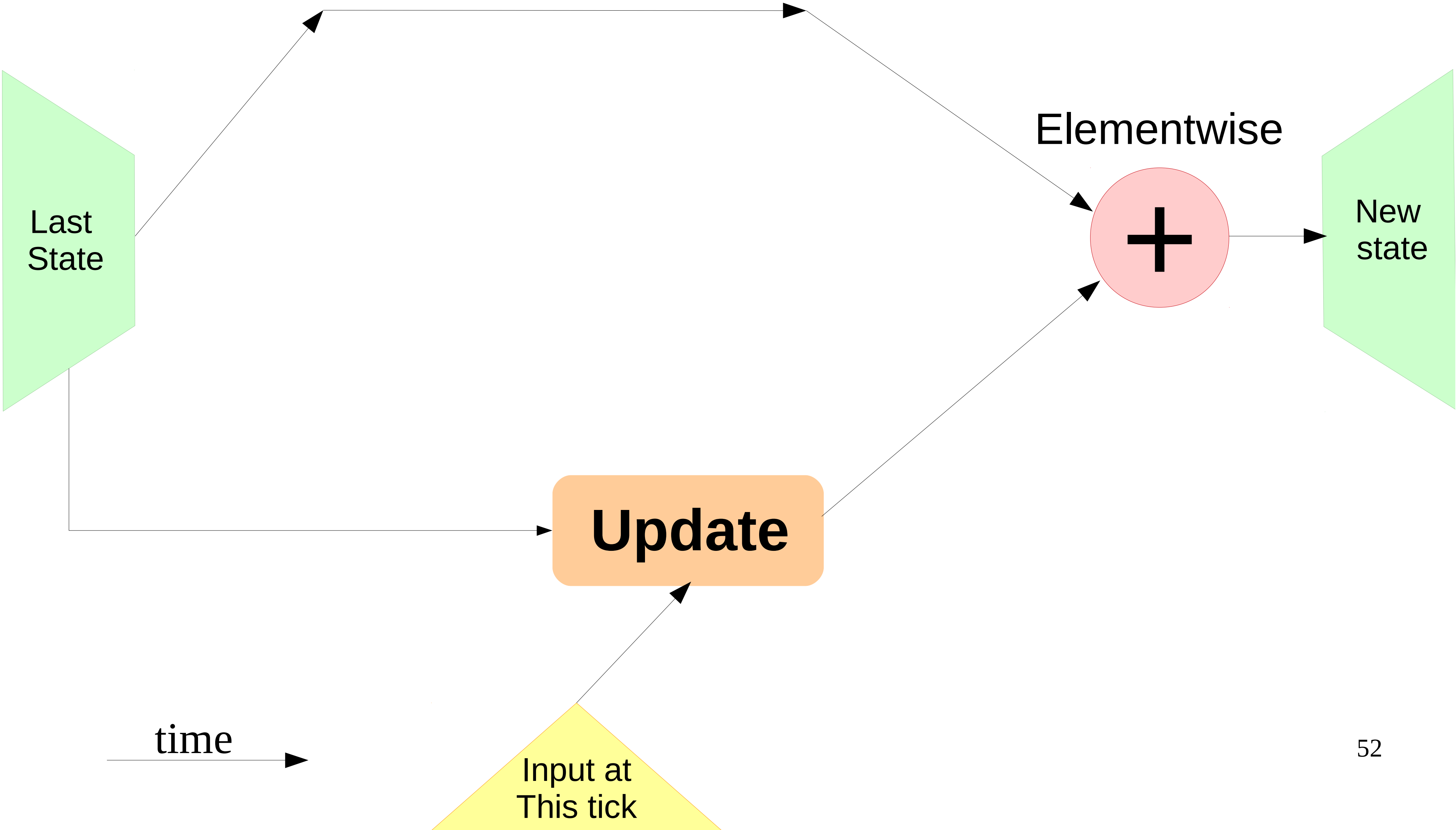
Residual RNN step



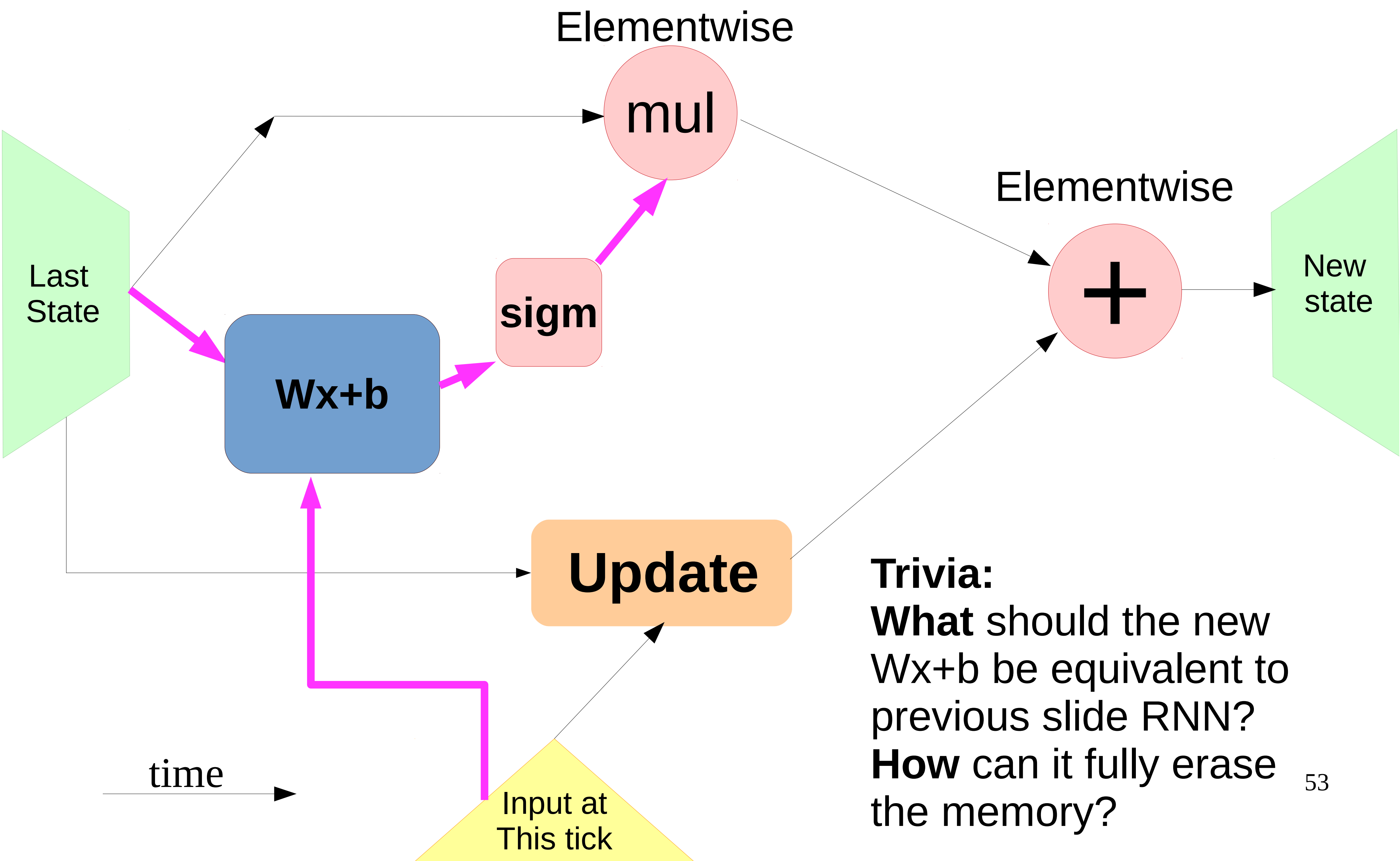
Residual RNN step



Residual RNN step

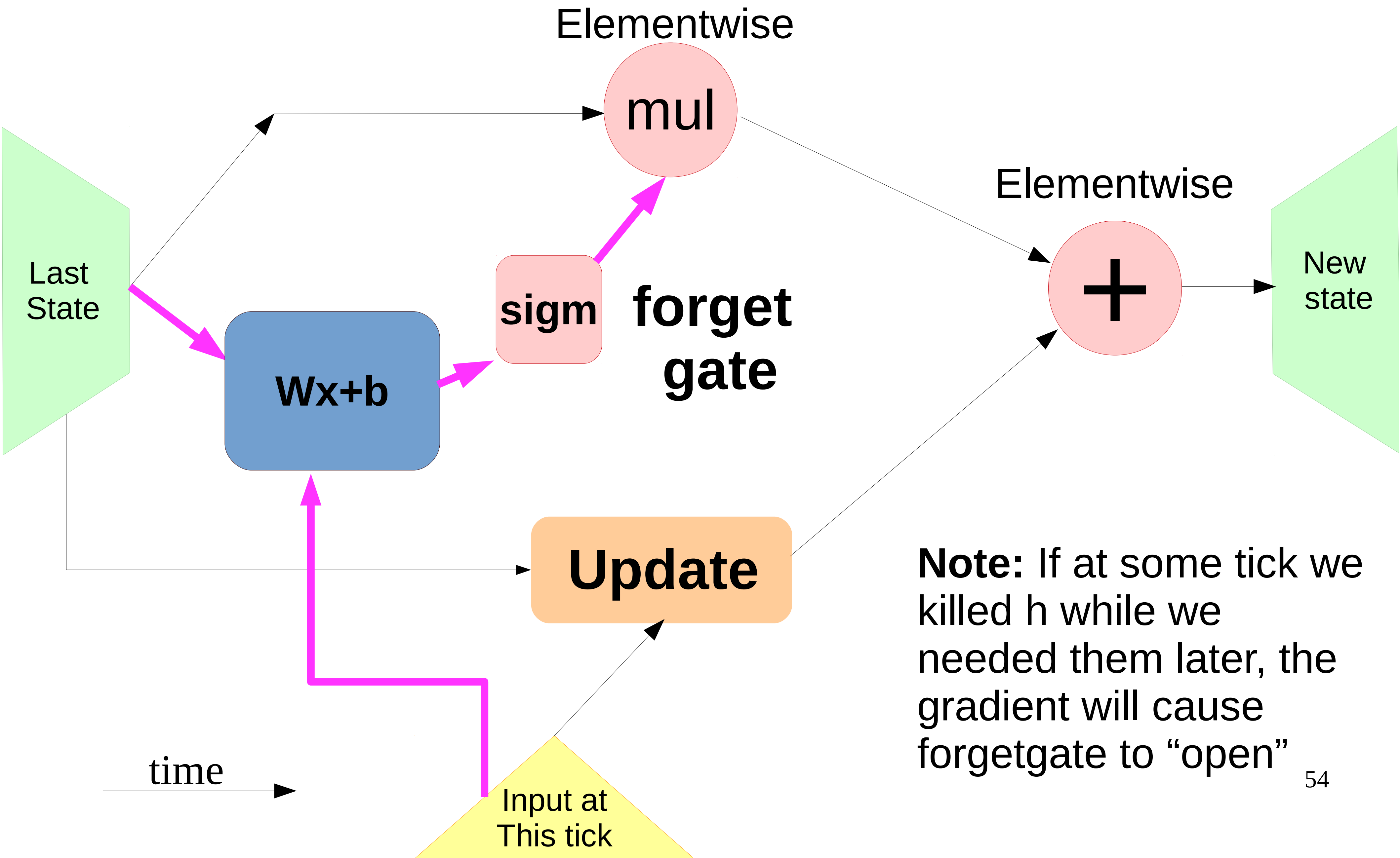


Residual RNN step



Trivia:
What should the new Wx+b be equivalent to previous slide RNN?
How can it fully erase the memory?

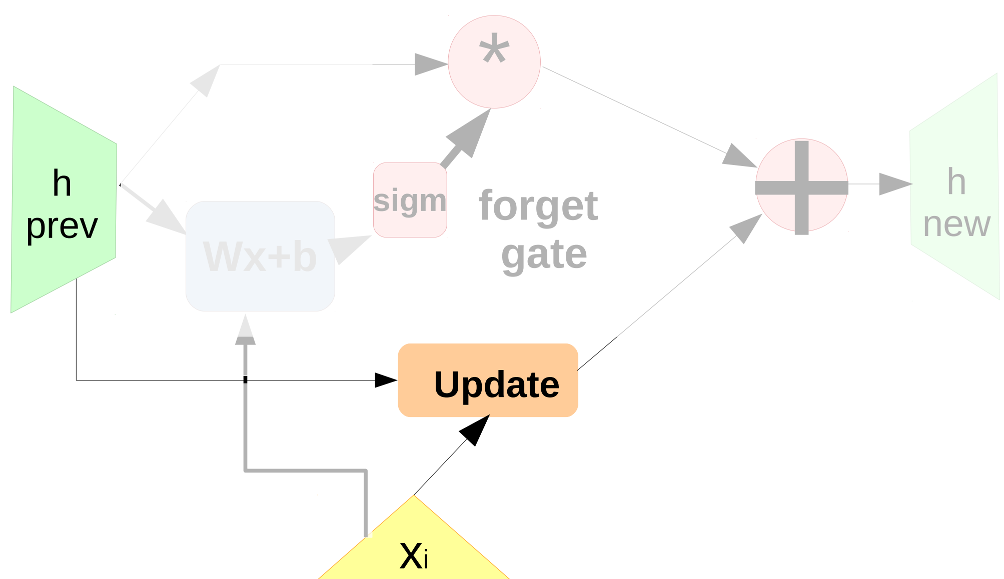
Residual RNN step



Note: If at some tick we killed h while we needed them later, the gradient will cause forgetgate to “open”

What we drew

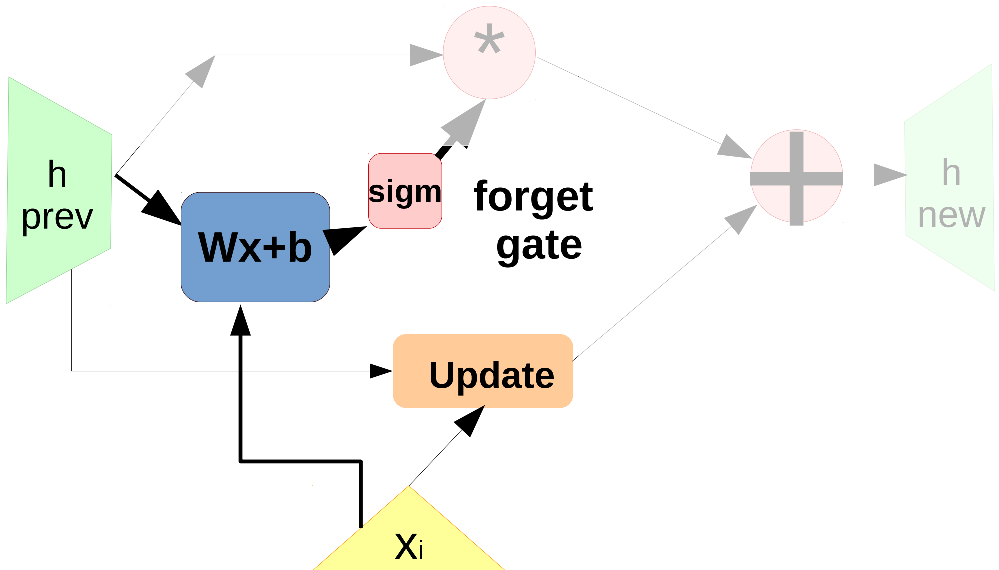
$$update(x_i, h_{i-1}) = \tanh\left(W_{hid}^{update} \cdot h_{i-1} + W_{inp}^{update} \cdot x_i + b^{update}\right)$$



What we drew

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

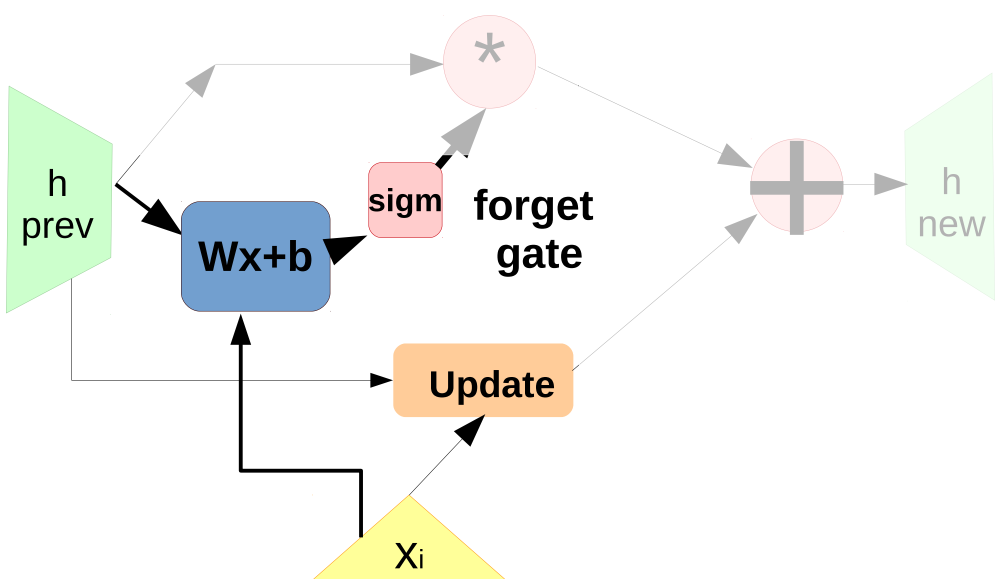
$$forget(x_i, h_{i-1}) = \sigma(W_{hid}^{forget} \cdot h_{i-1} + W_{inp}^{forget} \cdot x_i + b^{forget})$$



What we drew

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

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



How to compute h_new?

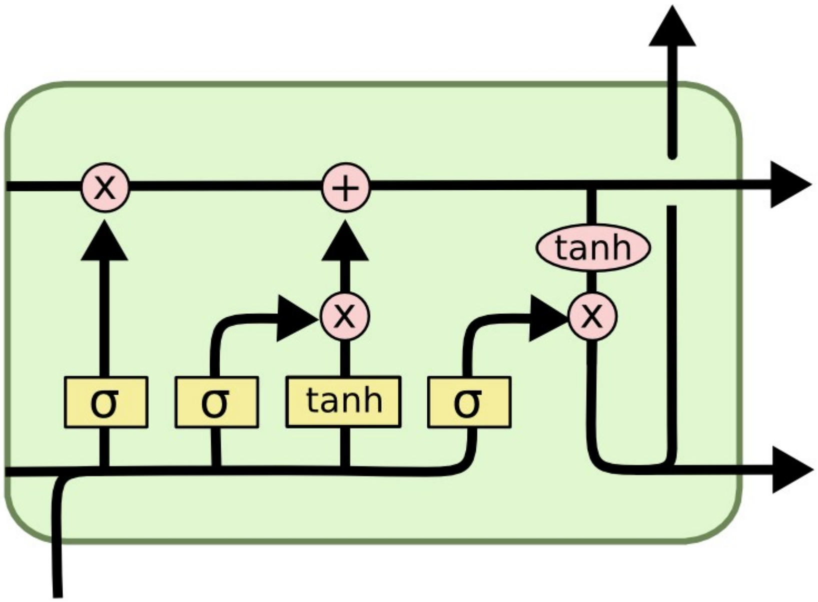
What we drew

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

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

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

LSTM



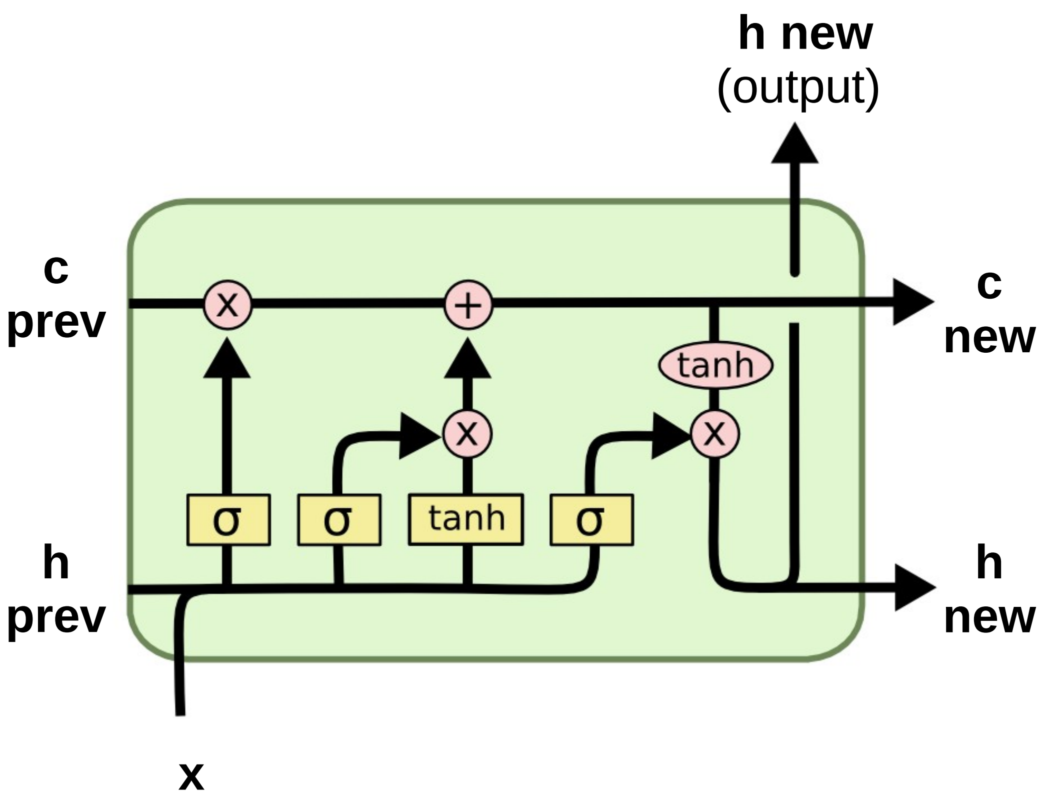
2 hidden states:

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

4 blocks:

- Update
- Forget gate
- Input gate
- Output gate

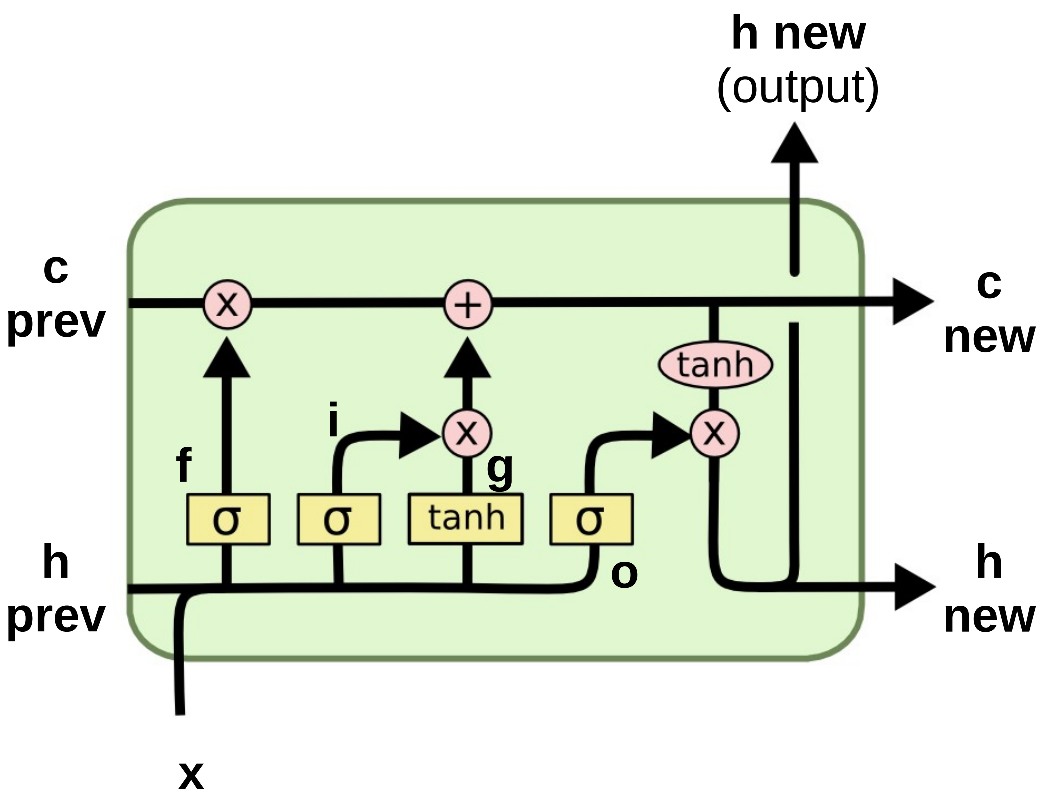
LSTM



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

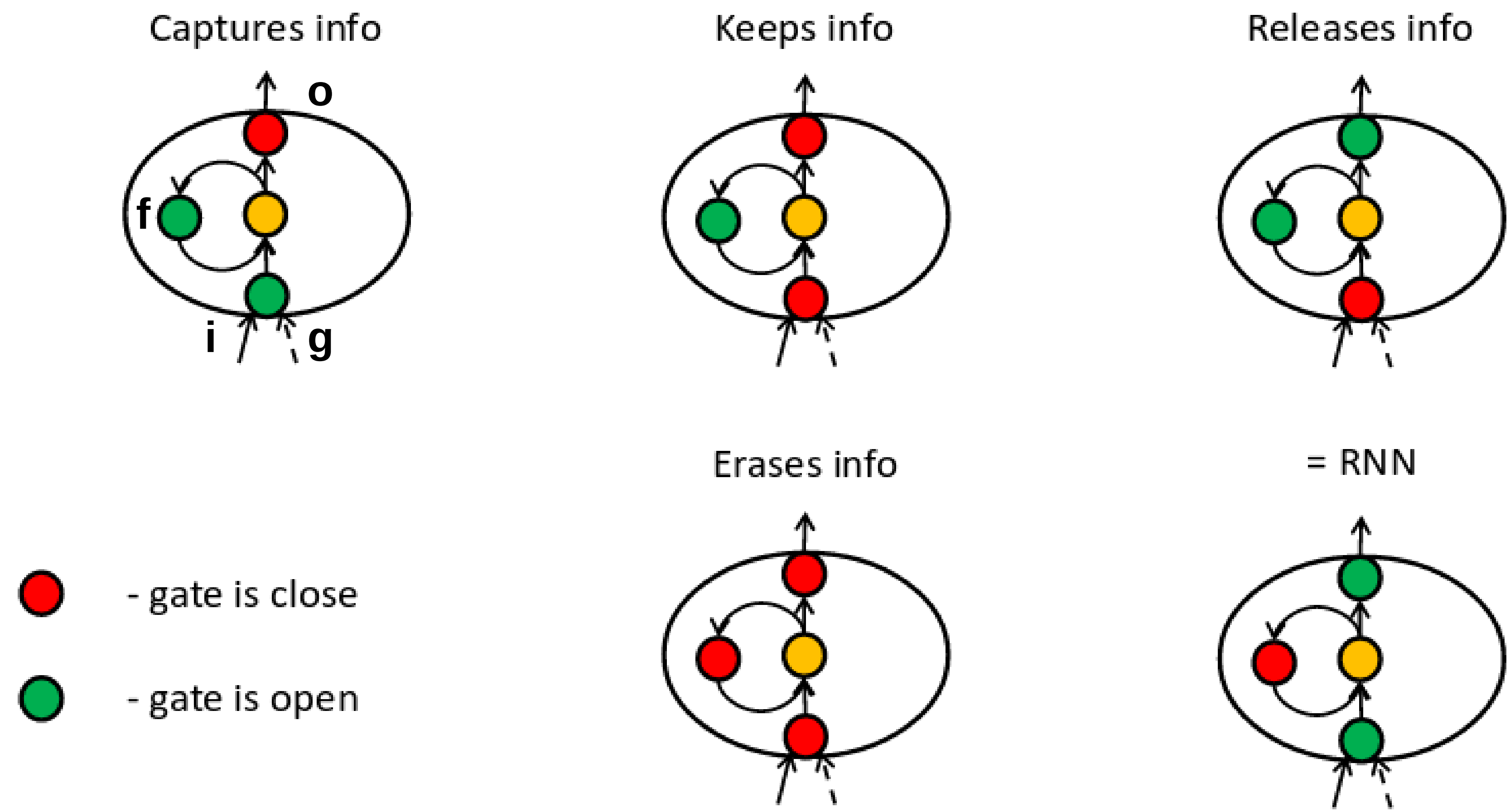
Where are the gates?

LSTM



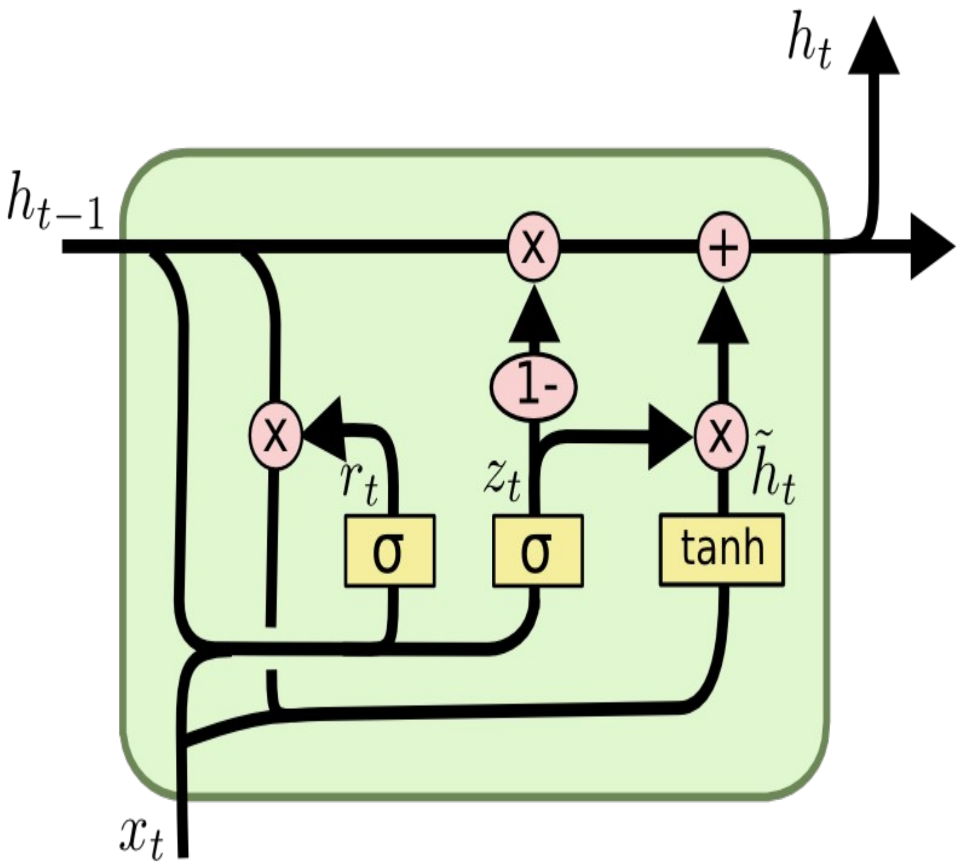
$$i_t = Sigm(\theta_{xi} x_t + \theta_{hi} h_{t-1} + b_i)$$
$$f_t = Sigm(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)$$
$$o_t = Sigm(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$$
$$g_t = Tanh(\theta_{xg} x_t + \theta_{hg} h_{t-1} + b_g)$$
$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_t$$
$$h_t = o_t \otimes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

LSTM: not a monster



[Pictures: E Lobacheva, D Vetrov]

GRU



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

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

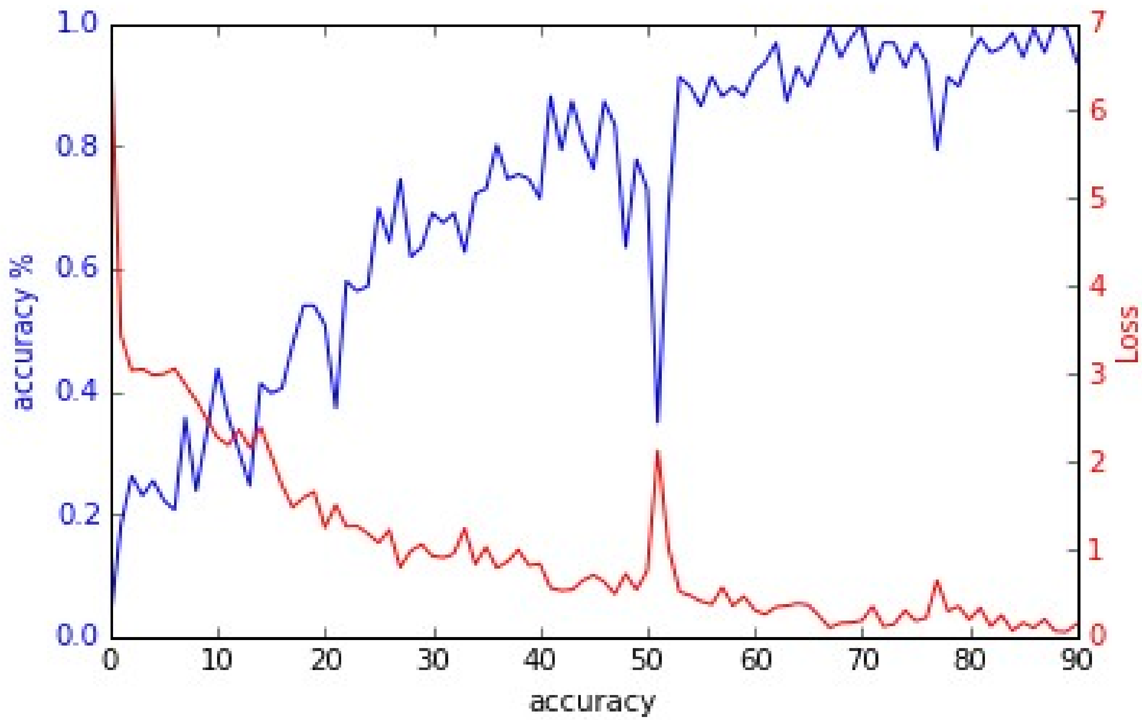
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

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

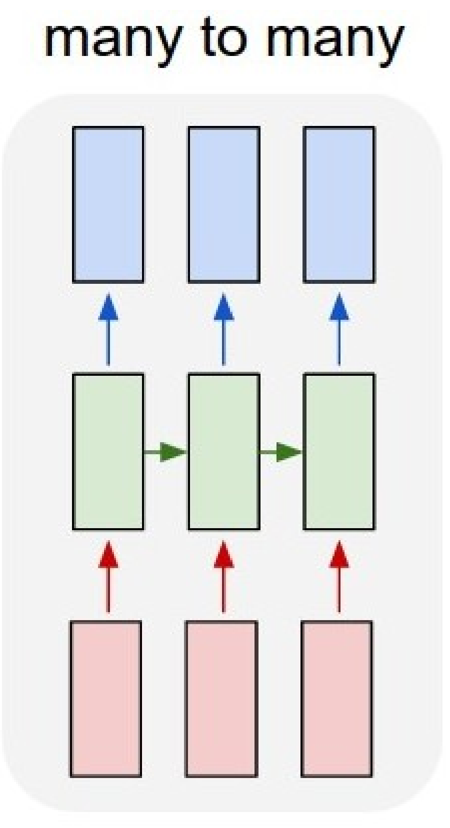
Medium:

- Grammatically coherent text
- Resembling particular author

Hard:

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

Recurrent Architectures: regular

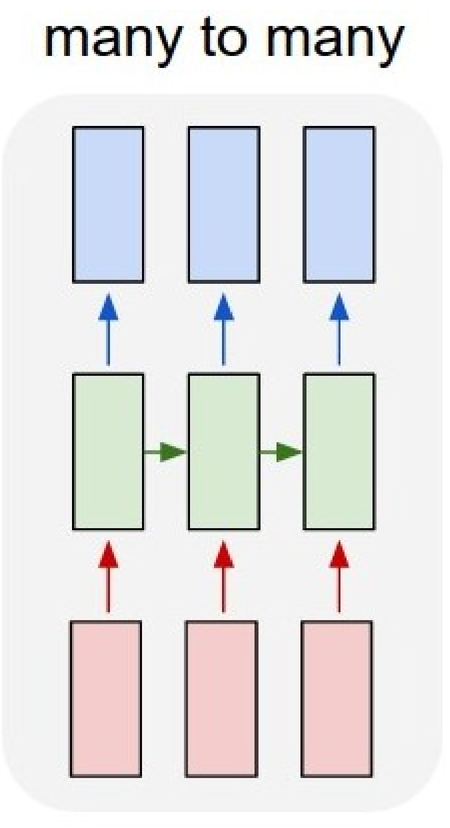


- Read sequence
- Predict sequence of answers at each tick

Tasks:
- Language model
- POS Tagging

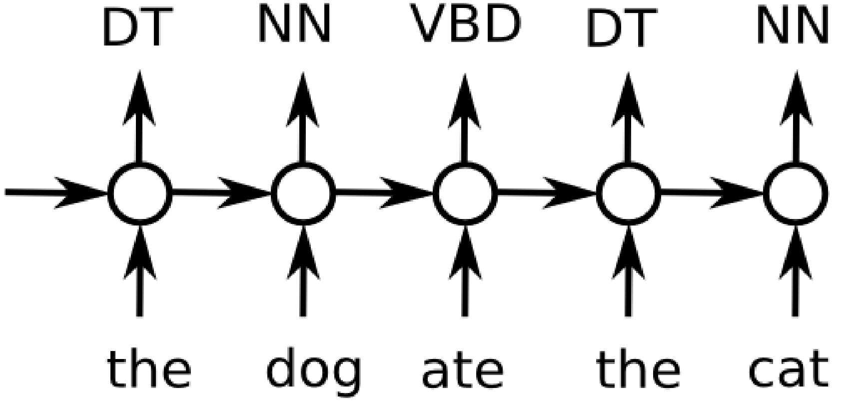
How to implement?
- See last week

Recurrent Architectures: regular



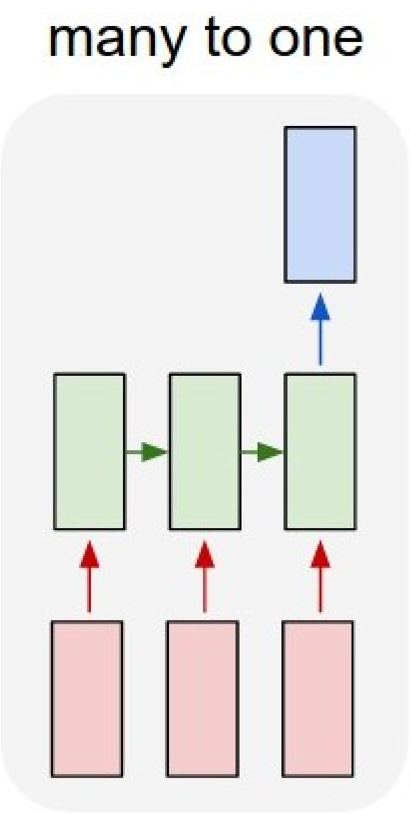
- Read sequence
- Predict sequence of answers at each tick

POS tagging



Why RNN?

Recurrent Architectures: Encoder

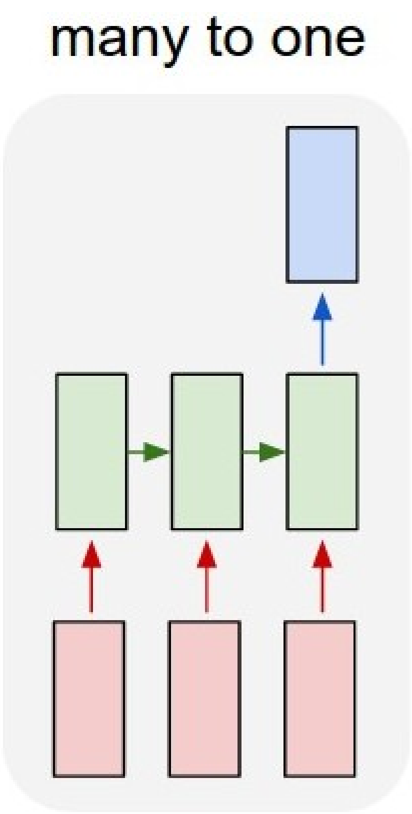


Encoder
- Read sequence
- Predict once

Tasks:
- **?!**

How to implement?
- **?!**

Recurrent Architectures: Encoder

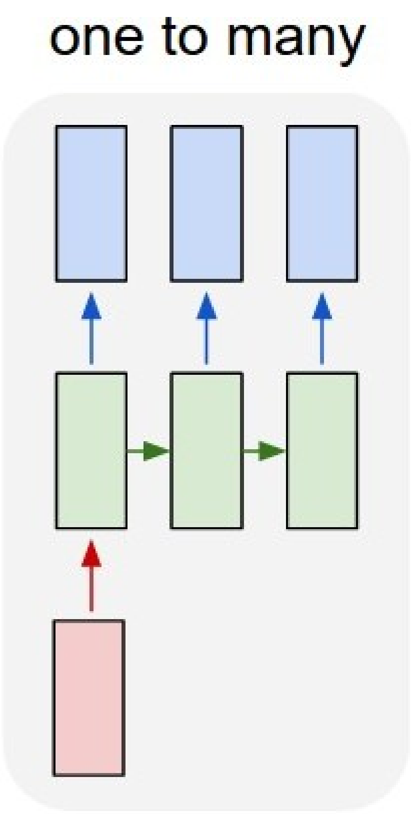


Encoder
- Read sequence
- Predict once

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

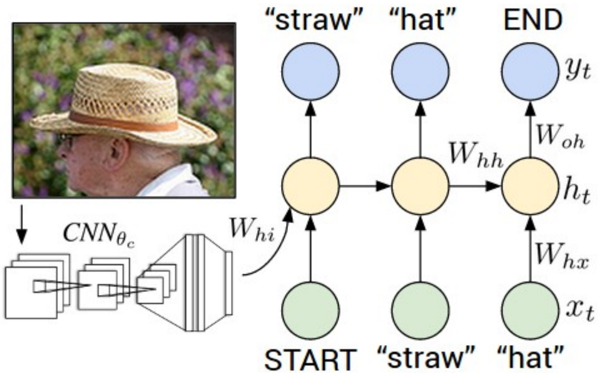
How to implement?
- Take last/max/mean over time

Recurrent Architectures: Decoder



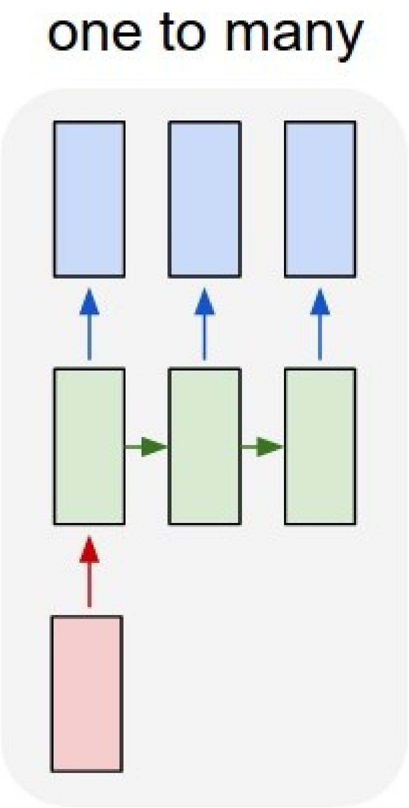
Decoder
- Take one state
- Generate sequence

Tasks:
- Image captioning



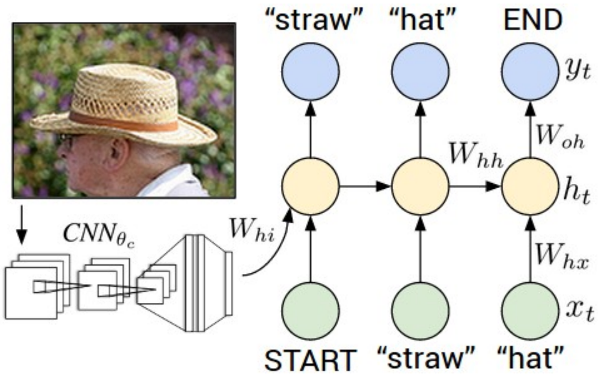
How to implement?
- **?!**

Recurrent Architectures: Decoder



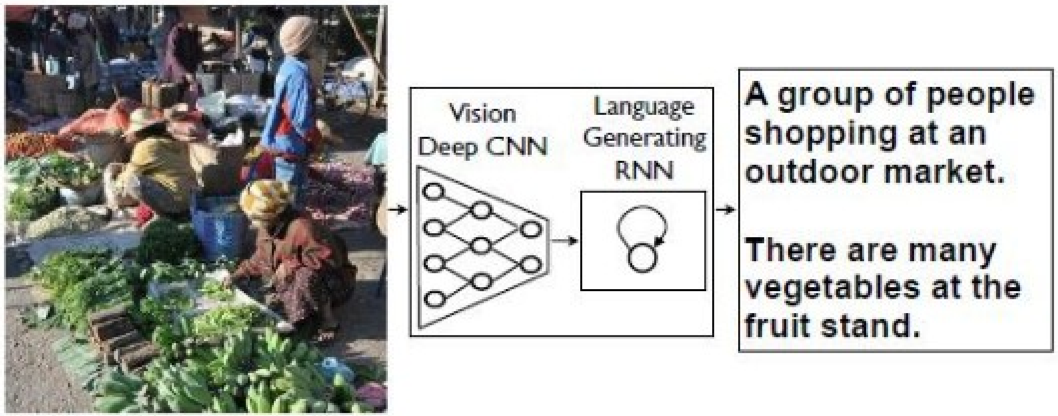
Decoder
- Take one state
- Generate sequence

Tasks:
- Image captioning



How to implement?
- First state init (instead of zeros)
- Input at each tick

Image captioning



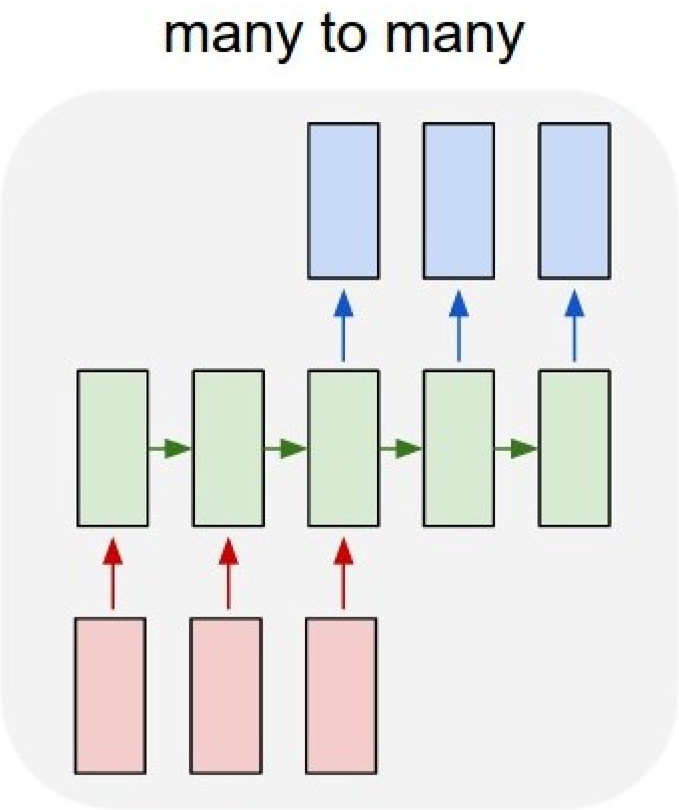
- Demo - http://stanford.io/2esMxOq
- Upload your image - http://bit.ly/2eAoueP

Seq2seq

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

Example: **Machine translation**

Seq2seq

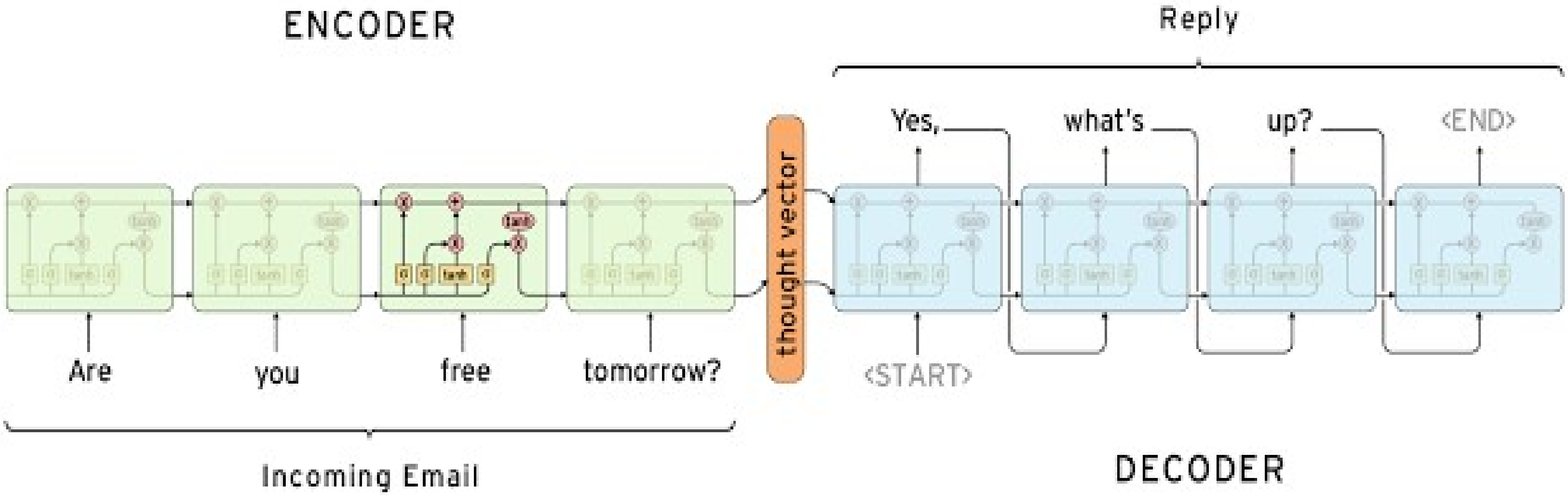


Idea:

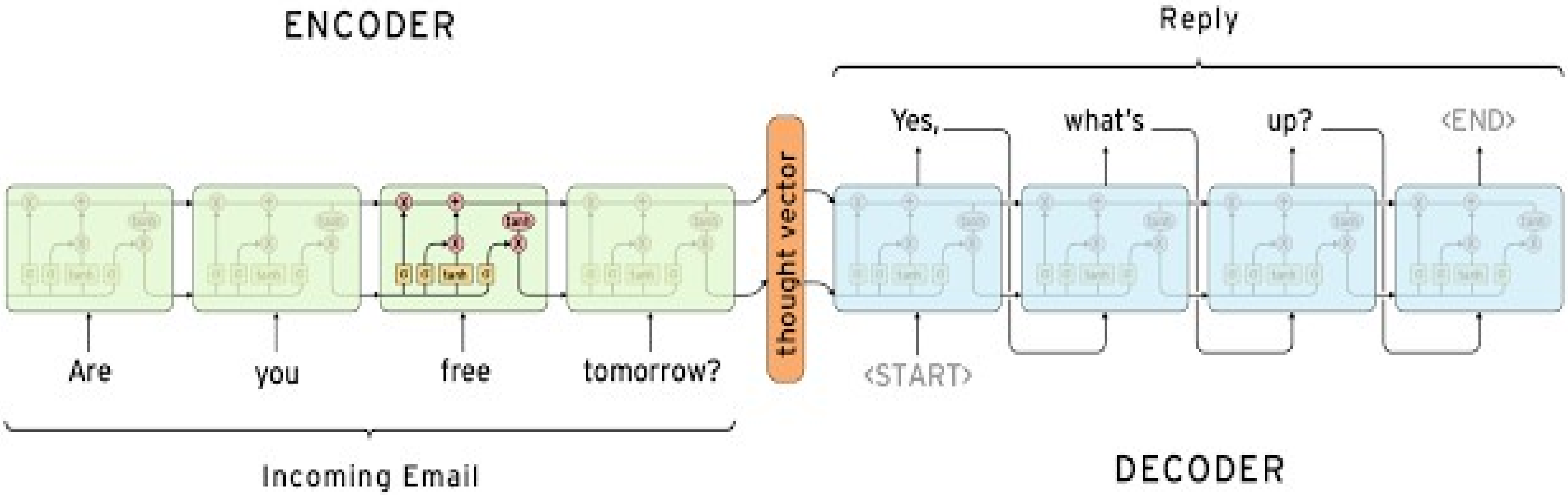
- first read (encode) the sequence
- then generate new one out of the encoded vector

How to implement that?

Seq2seq: encoder-decoder

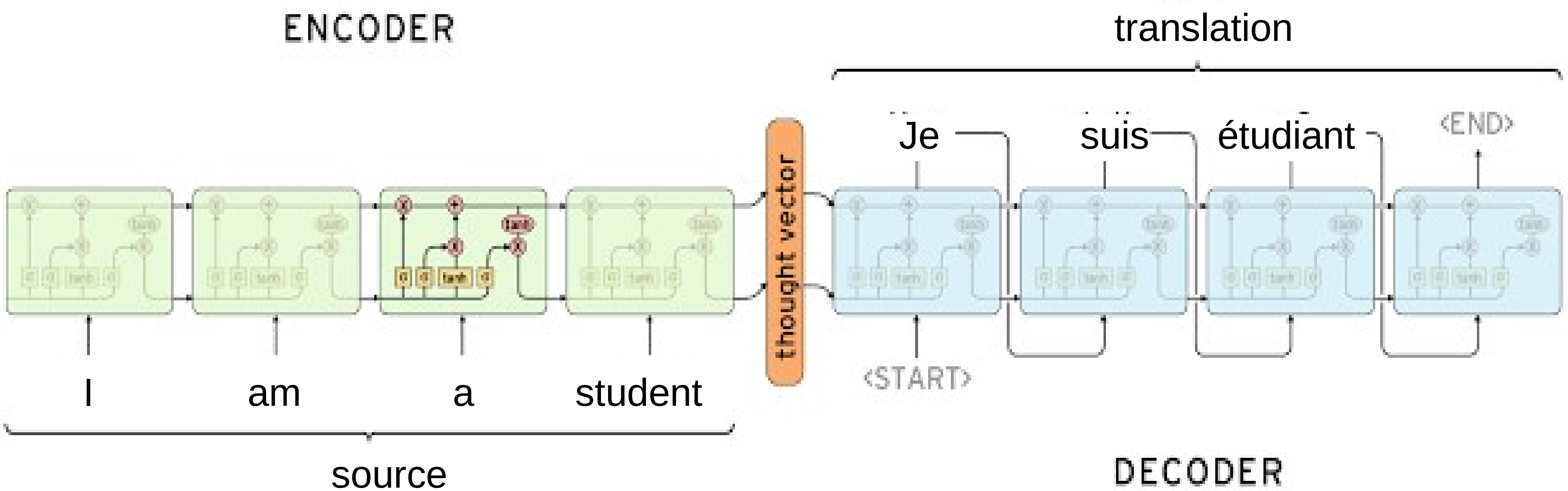


Seq2seq: Conversation model



Exactly the same

Seq2seq: Machine translation



Nuff

Coding time!