CS378: Natural Language Processing

Lecture 14: Neural Network (Sequence)

TEXAS
The University of Texas at Austin

Eunsol Choi

Slides from Greg Durrett, Yoav Artzi, Yejin Choi.

Neural Network

- Recurrent Neural Network (RNN)
  - Input is a variable length sequence, but output is not.
- Encoder-Decoder model
  - Both input / output is a variable length sequence
  - Attention mechanism
- Alternative approaches to model sequence
  - Convolutional Neural Network
  - Transformer

Readings

- Recurrent Neural Network (RNN)
  - J&M 9
- Encoder-Decoder model
  - J&M 10
- Thursday: Alternative approaches to model sequence
  - Convolutional Neural Network [Voita]
  - Transformer [Vaswani et al, 2017], JM 11.2-5

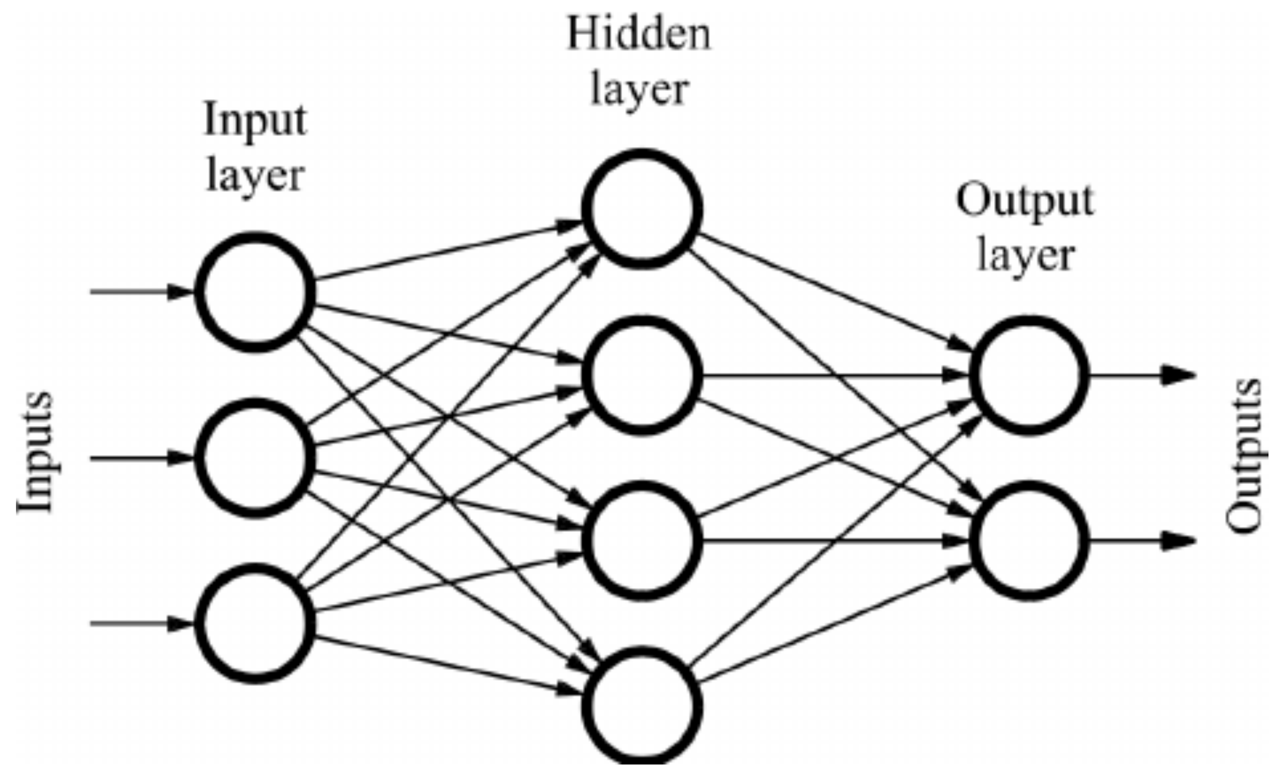
Logistics

- Independent study proposal — I'll return it sometime this week!

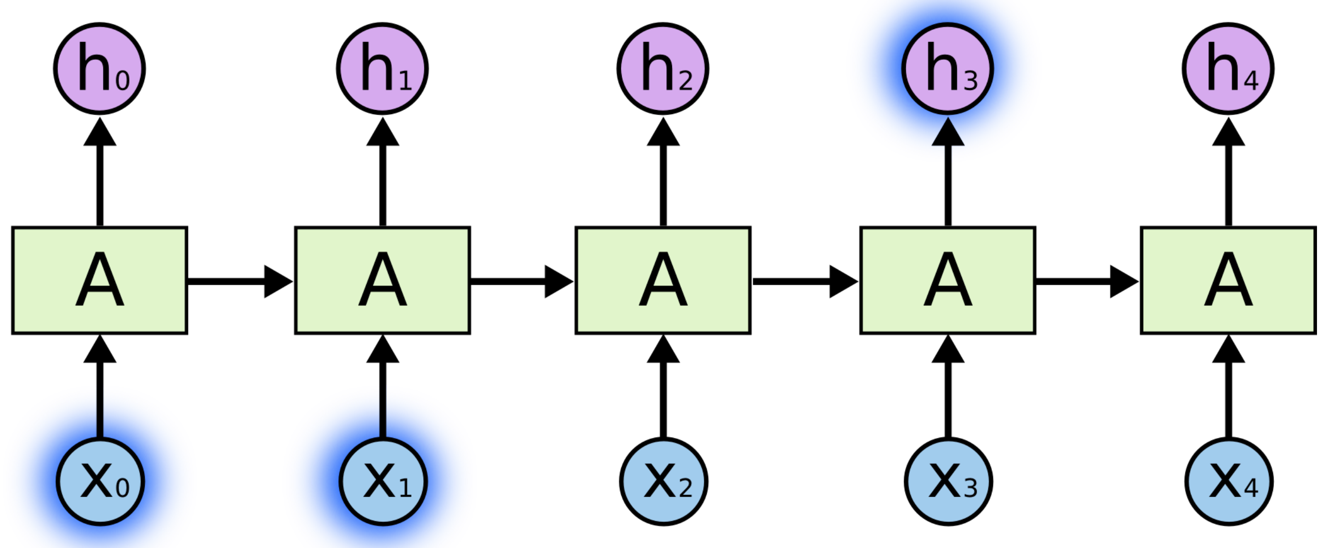
- HW3 intermediate deadline today

Neural Networks in NLP

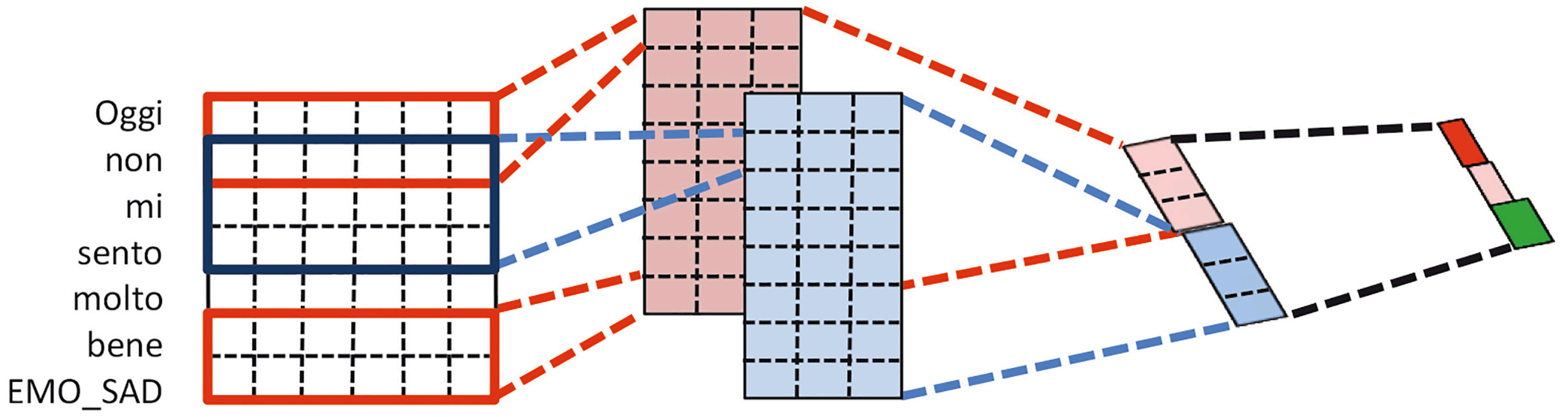
Feed-forward NNs



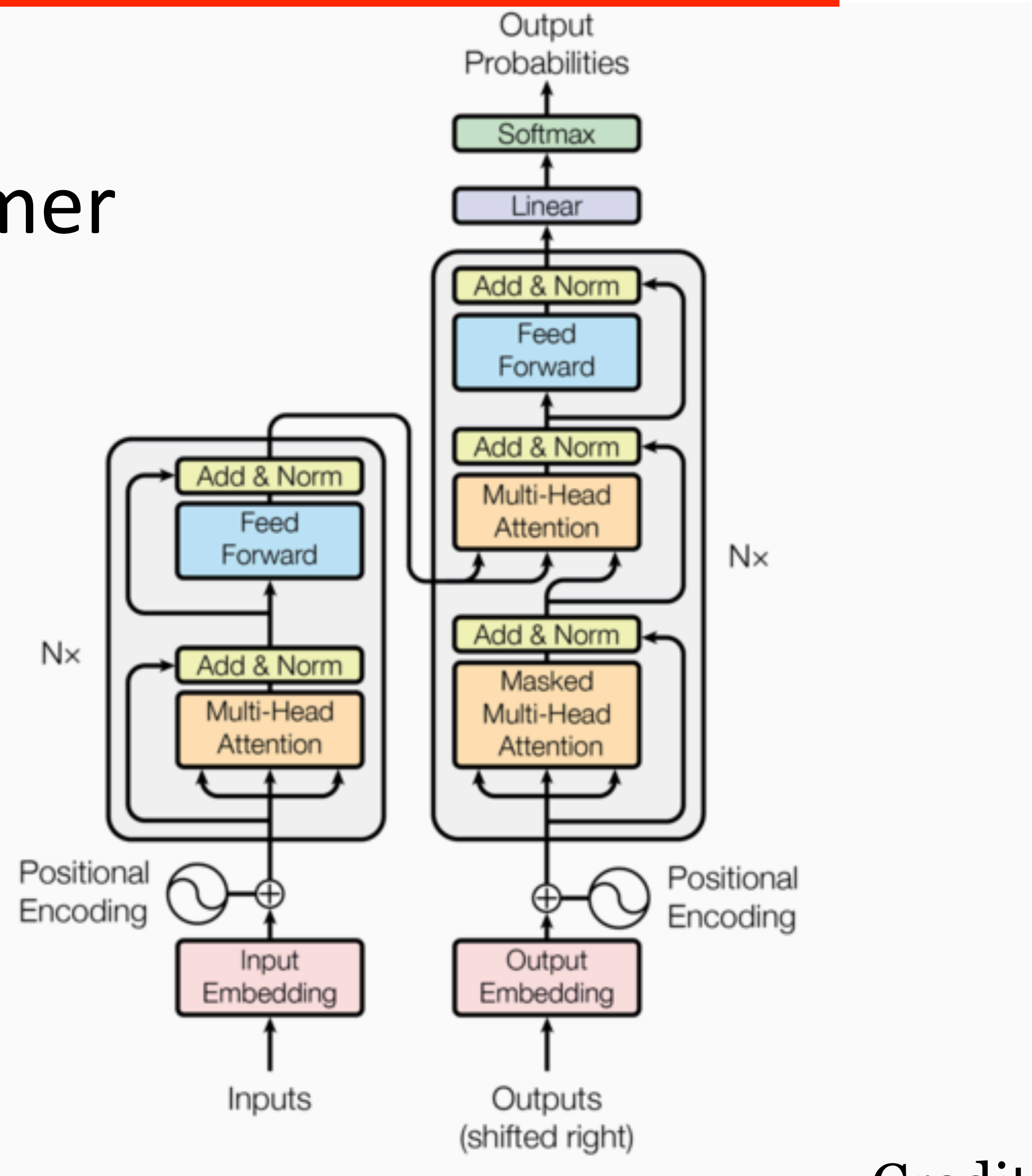
Recurrent NNs



Convolutional NNs



Transformer



Credits: Princeton NLP course

Recap: RNNs

- $S = \mathbb{R}^{d_{hid}}$ - hidden state space $(h_1, h_2 \ldots)$
- $\Sigma = \mathbb{R}^{d_{in}}$ - input state space $(x_1, x_2 \ldots .)$
- $s_0 \in S$ - initial state vector (h_0)
- $R : \mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{hid}} \to \mathbb{R}^{d_{hid}}$ - transition function

- Maps from dense input sequence to dense hidden state representation sequence

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \to h_1, \ldots, h_n$$

- For all $i \in \{1, \ldots, n\}$,
  - $h_i = R\left(h_{i-1}, \boldsymbol{x}_i\right)$
  - Simple definition of R: $R(h_{i-1}, x_i) = \tanh(Wx_i + Vh_{i-1} + b)$

- R is parameterized, where the parameters are shared across all steps.

$$h_4 = R\left(h_3, \boldsymbol{x}_4\right) = \ldots = R(R\left(R\left(R\left(h_0, \boldsymbol{x}_1\right), \boldsymbol{x}_2\right), \boldsymbol{x}_3\right), \boldsymbol{x}_4)$$

Recap: RNNs

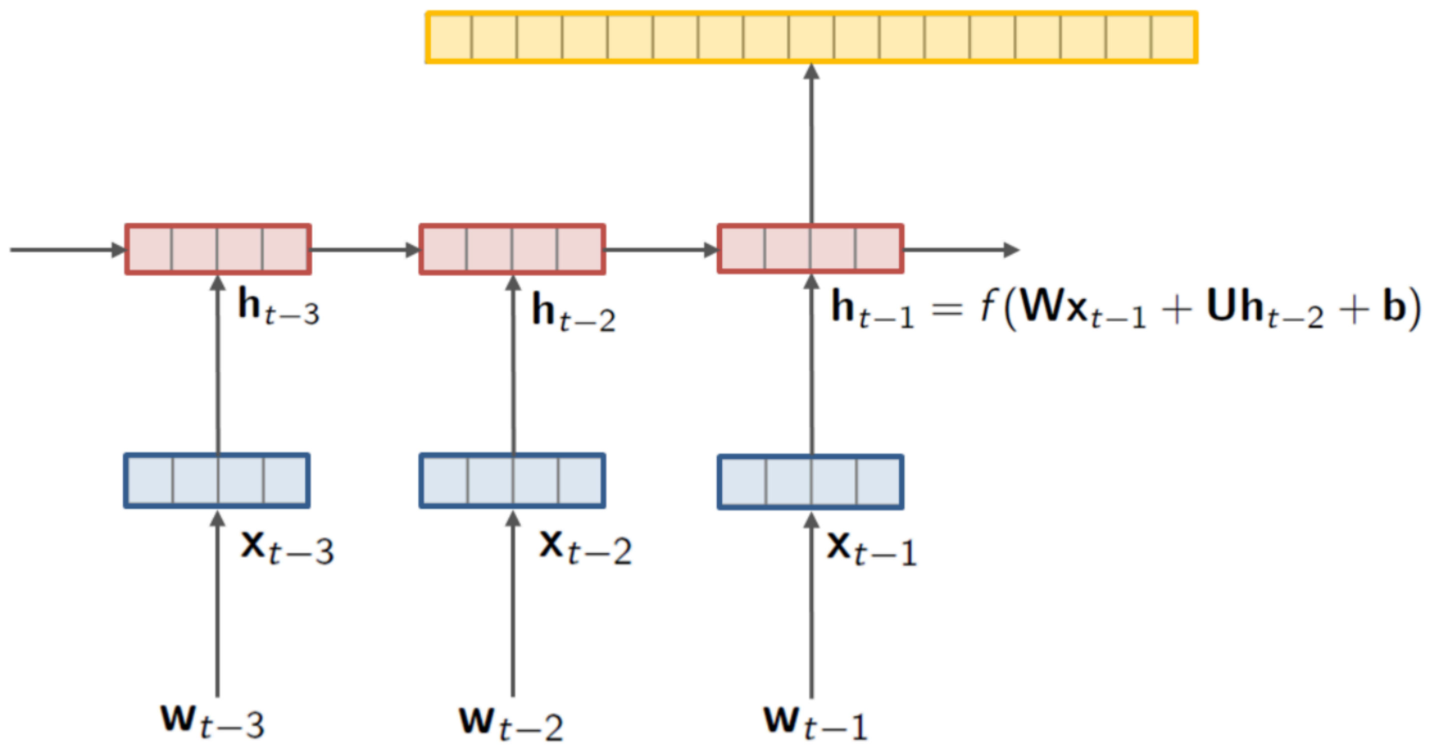
- Hidden states h_i can be used in different ways
- Output function maps hidden state vectors to symbols:

$$O: \mathbb{R}^{d_{hid}} \rightarrow \mathbb{R}^{d_{out}}$$

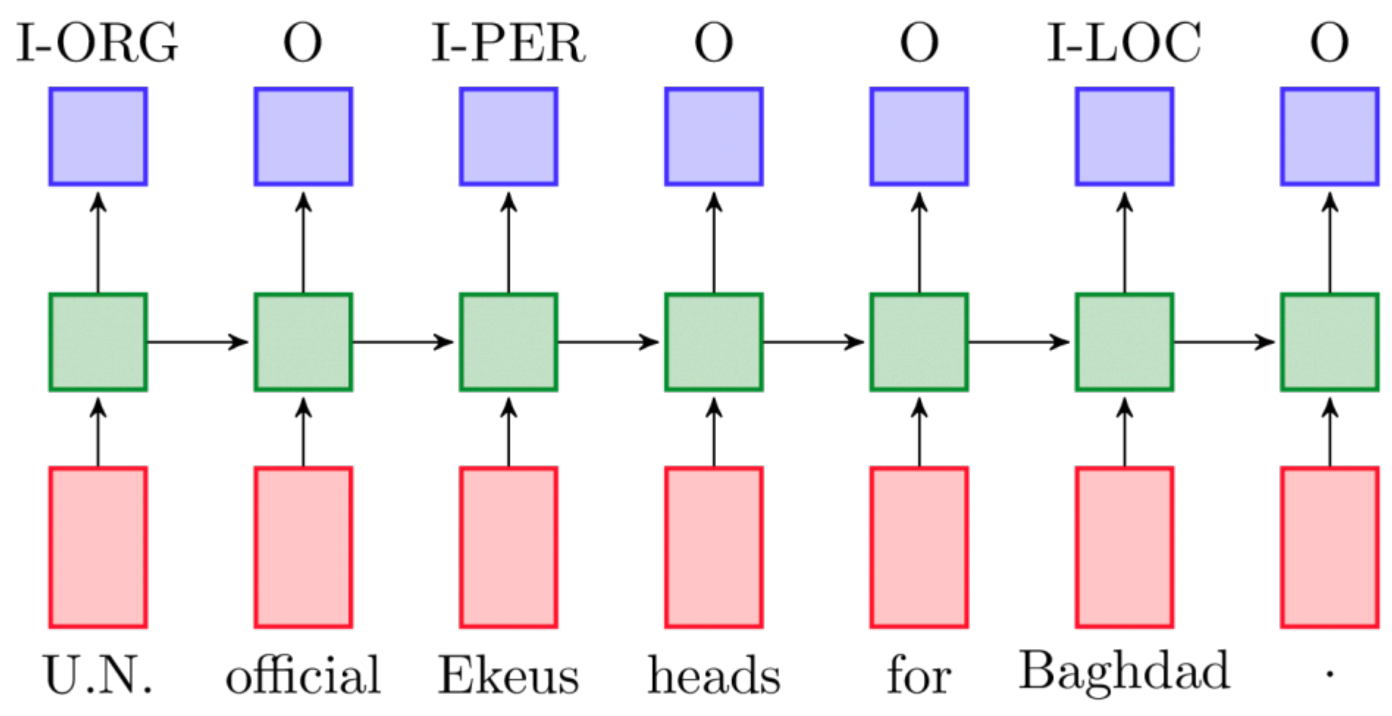
  - For example: single layer + softmax

$$O(h_i) = \text{softmax}(h_i \boldsymbol{W} + \boldsymbol{b})$$

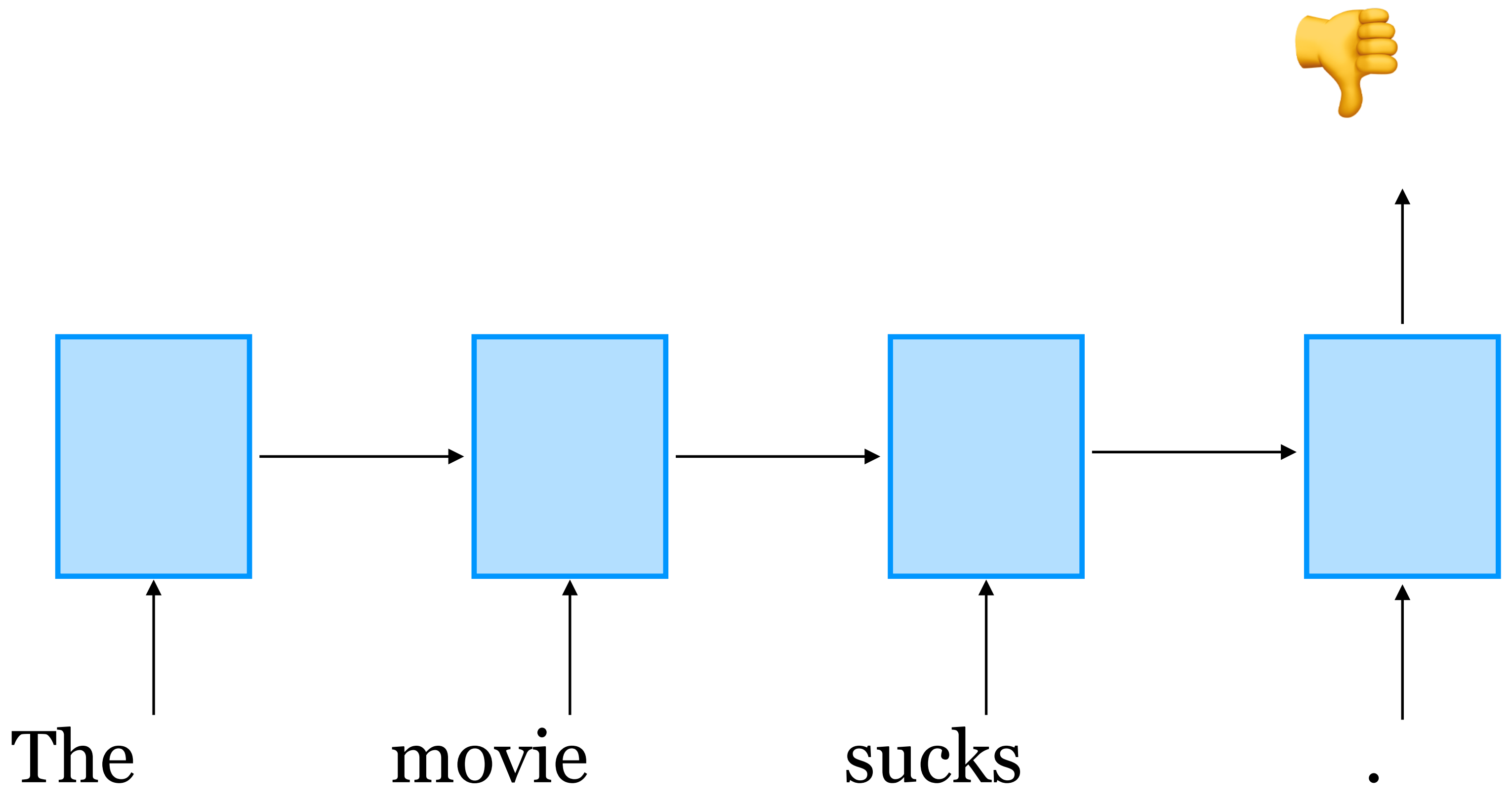
Language modeling



Sequence tagging

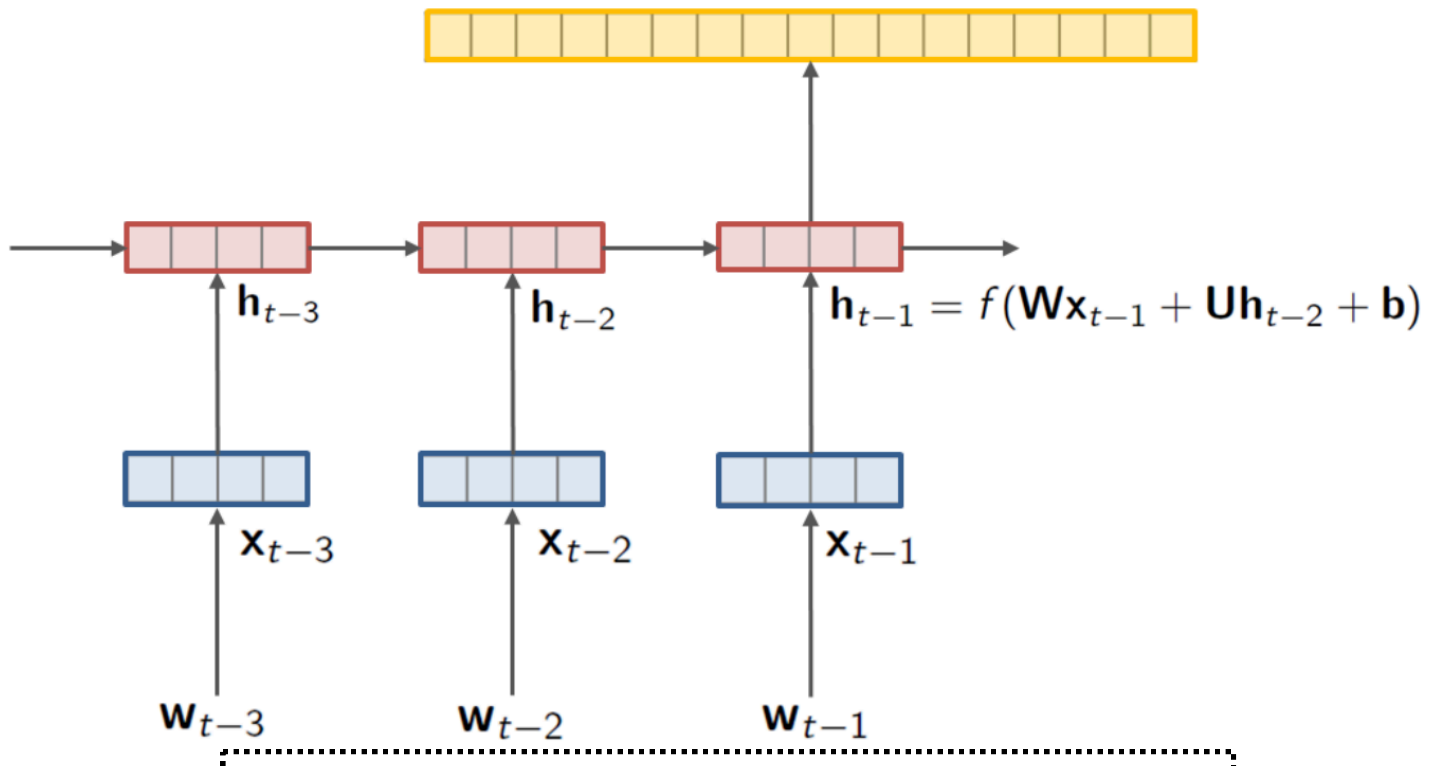


Text classification

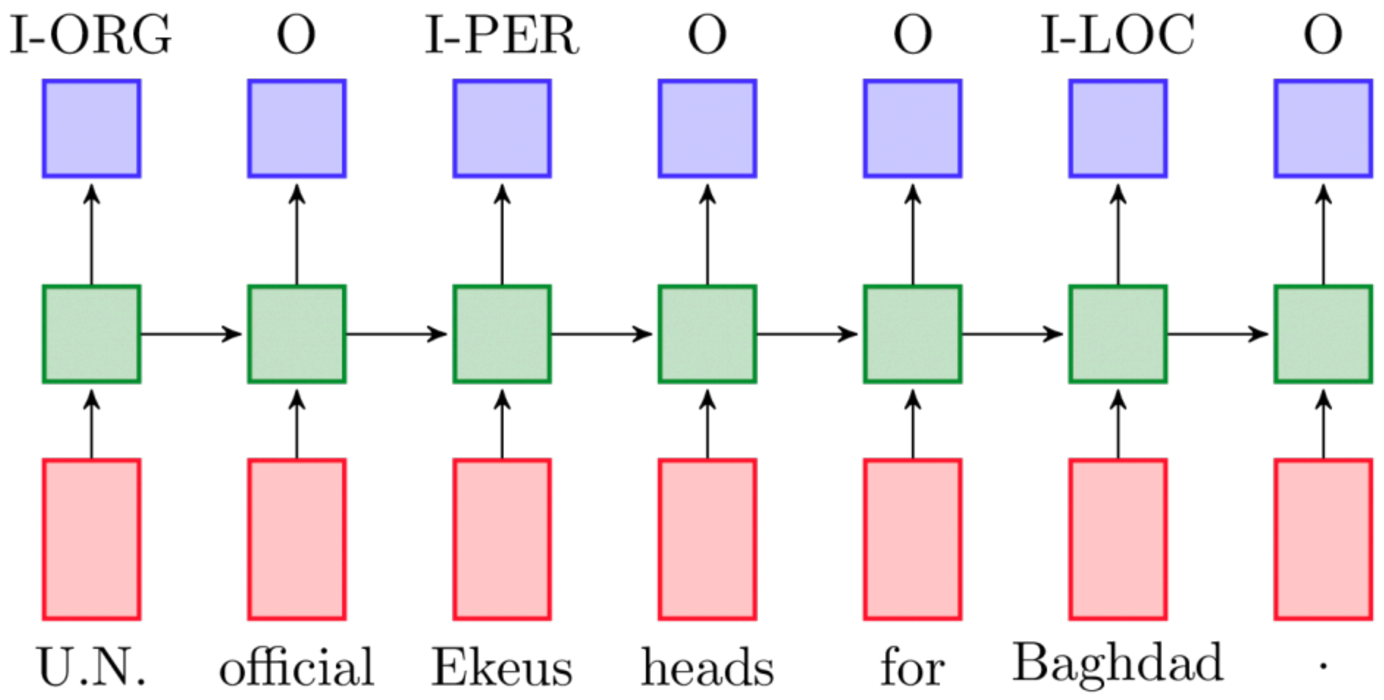


Output functions can be

- Transducer: make some prediction for each element in a sequence

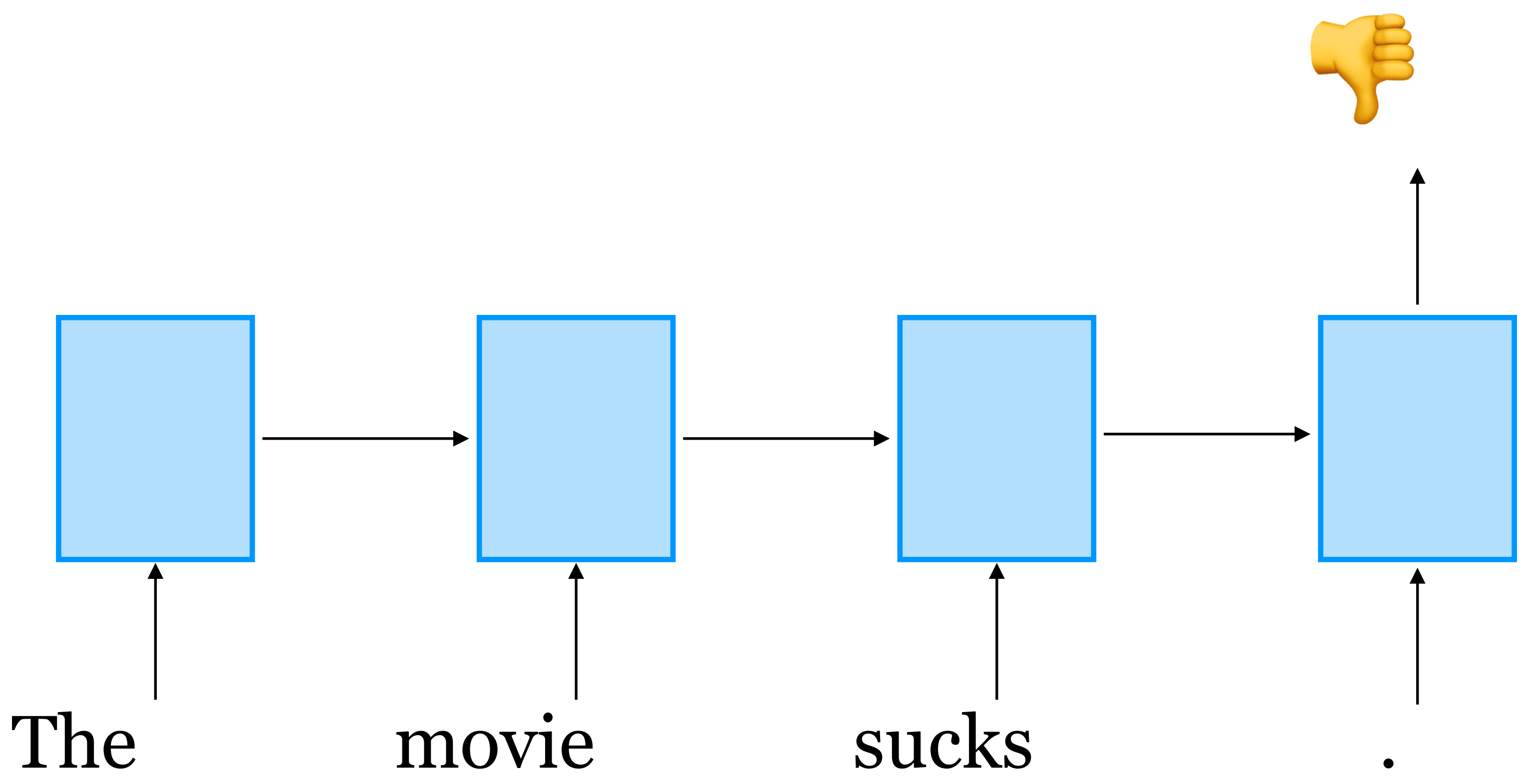


Language modeling



Sequence tagging

- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

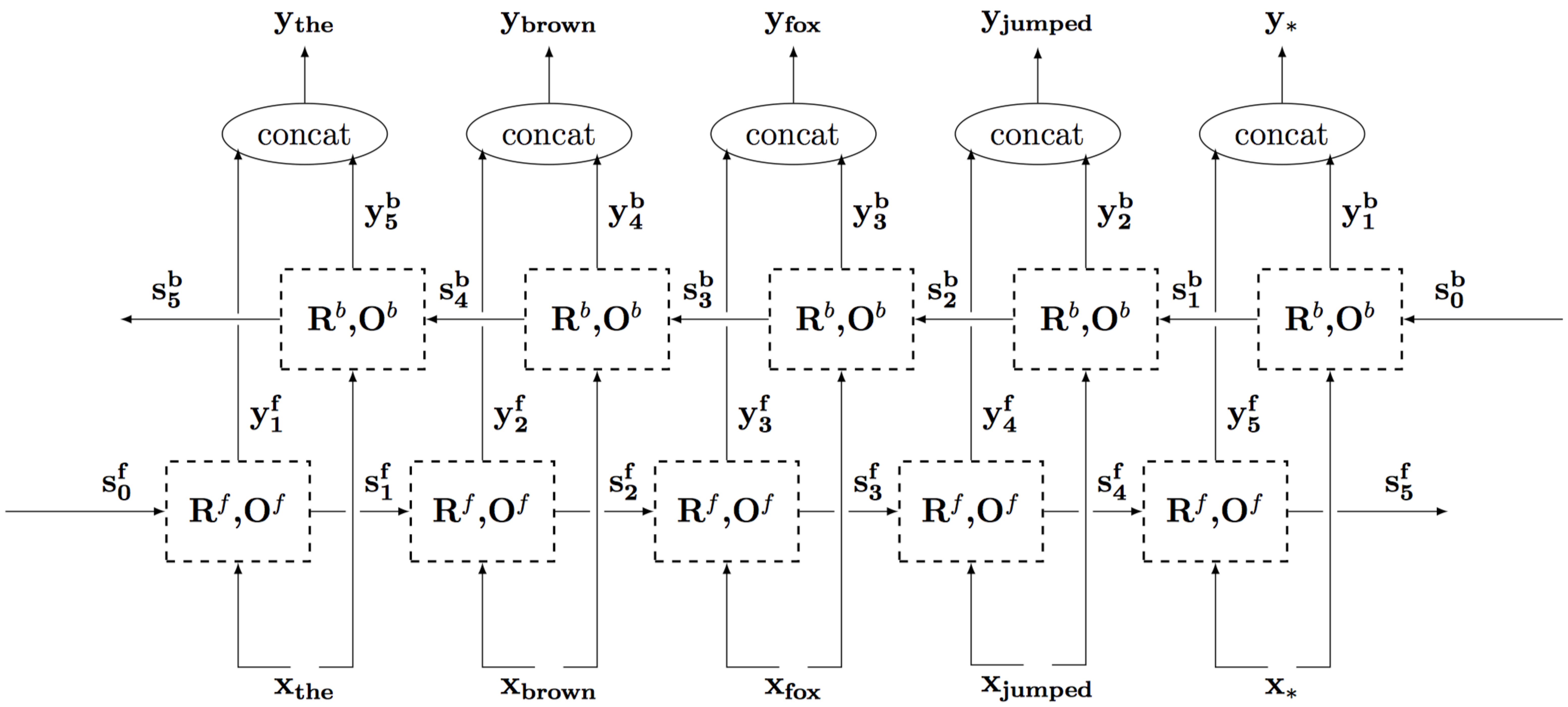


Text classification

Bi-directional RNNs

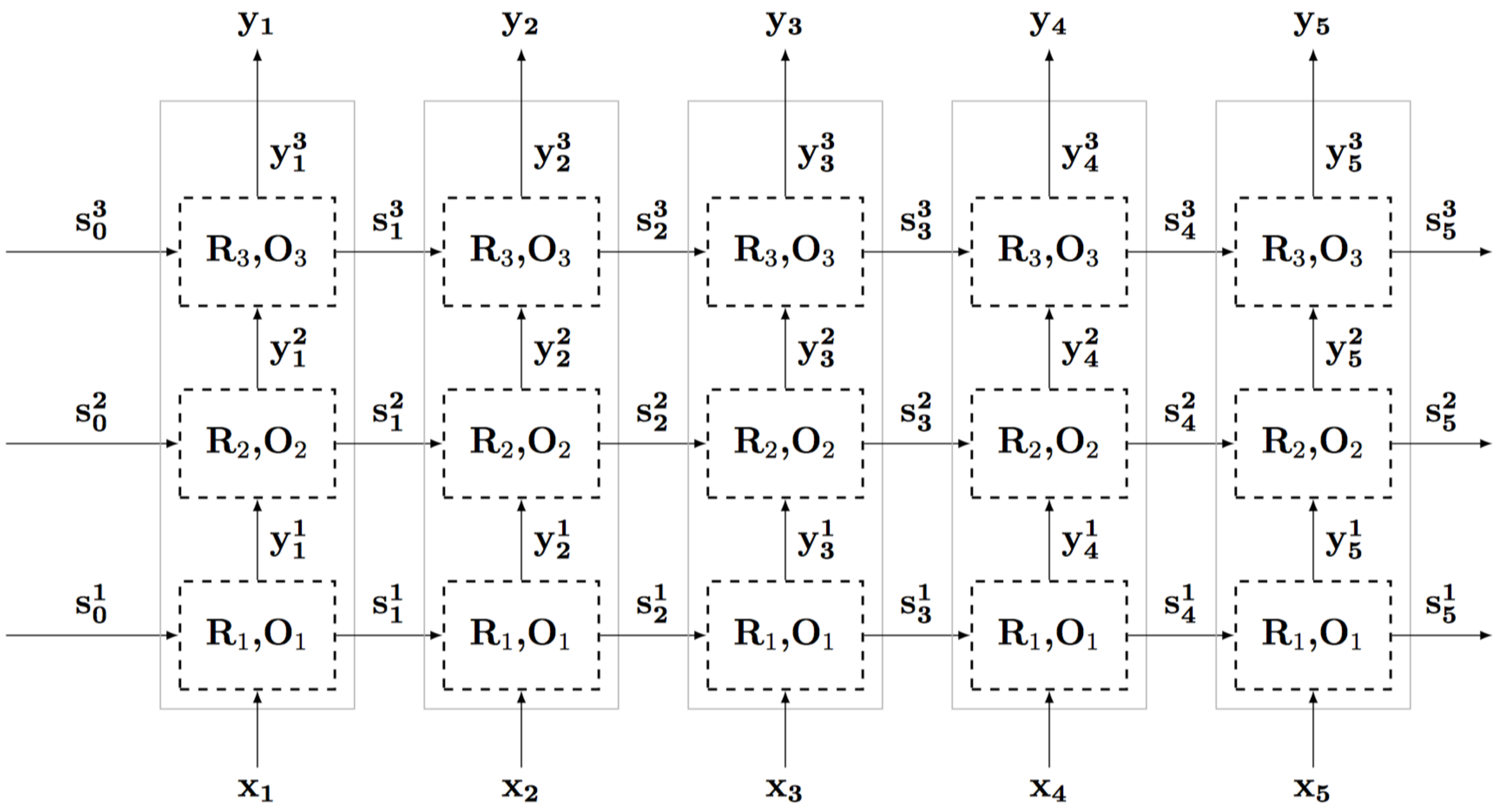
- RNN decisions are based on historical (past) data only.
- How can we account for future input?
- Is it realistic?

$$p(X) = \prod_{i=1}^{n} p(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$$



Deep RNNs

- We can make RNNs deeper (vertically) by stacking them.



Recurrent Neural Language Model

$$P(x_1, x_2, \ldots, x_n) = P(x_1) \times P(x_2 \mid x_1) \times P(w_3 \mid x_1, x_2) \times \ldots \times P(x_n \mid x_1, x_2, \ldots, x_{n-1})$$

$$= P(x_1 \mid \mathbf{h}_0) \times P(x_2 \mid \mathbf{h}_1) \times P(x_3 \mid \mathbf{h}_2) \times \ldots \times P(x_n \mid \mathbf{h}_{n-1})$$

$$x_t = E(w_t)$$

$$R(h_{i-1}, x_i) = \tanh(Wx_i + Vh_{i-1} + b)$$

$$O(h_i) = \mathrm{softmax}(h_i W_o)$$

$$L(\theta) = -\frac{1}{n}\sum_{t=1}^{n} \log O(h_{t-1})(x_t)$$

$$\theta = \{\mathbf{W}, \mathbf{V}, \mathbf{b}, \mathbf{W}_o, \mathbf{E}\}$$

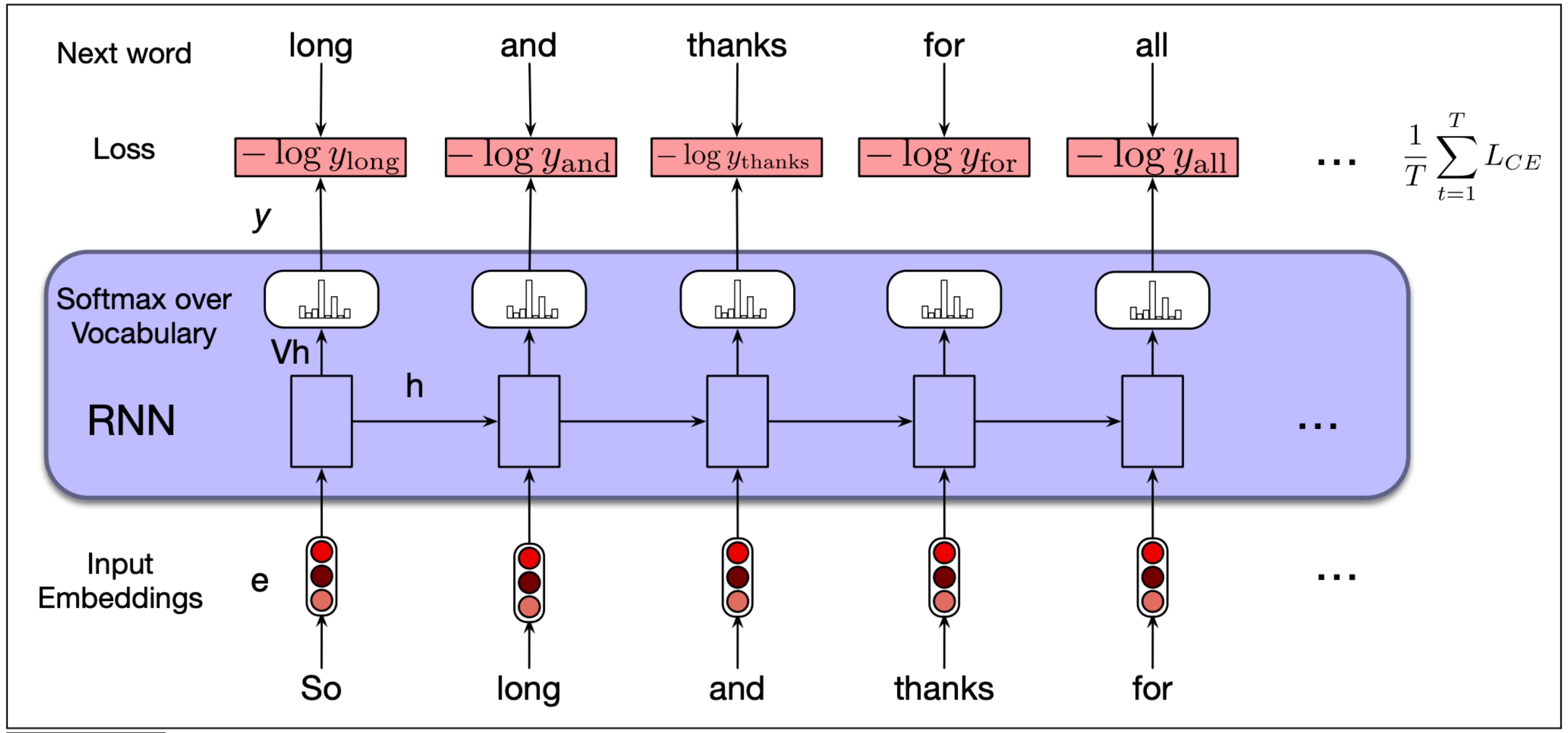
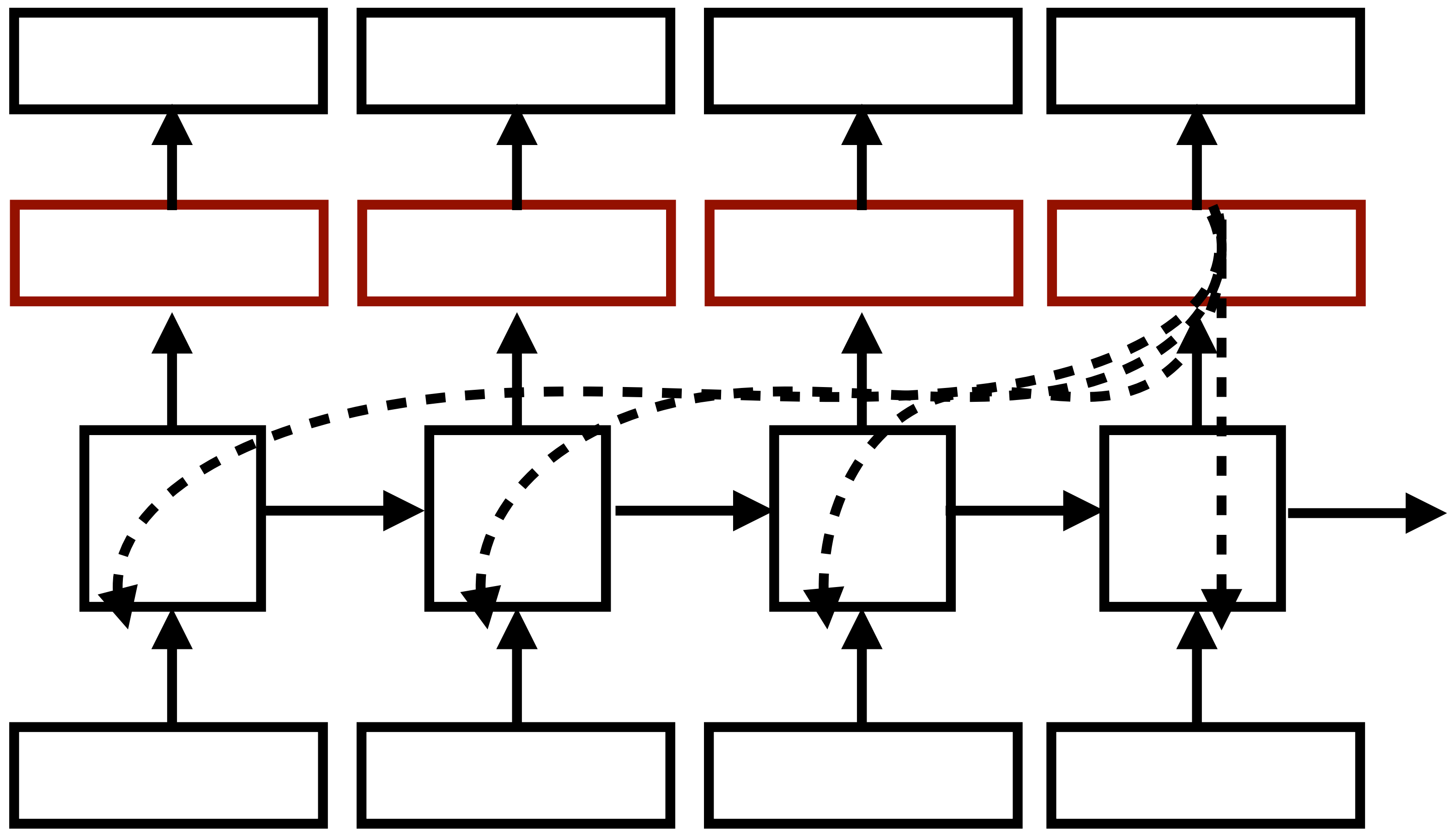


Figure 9.6 Training RNNs as language models.

Training RNNs

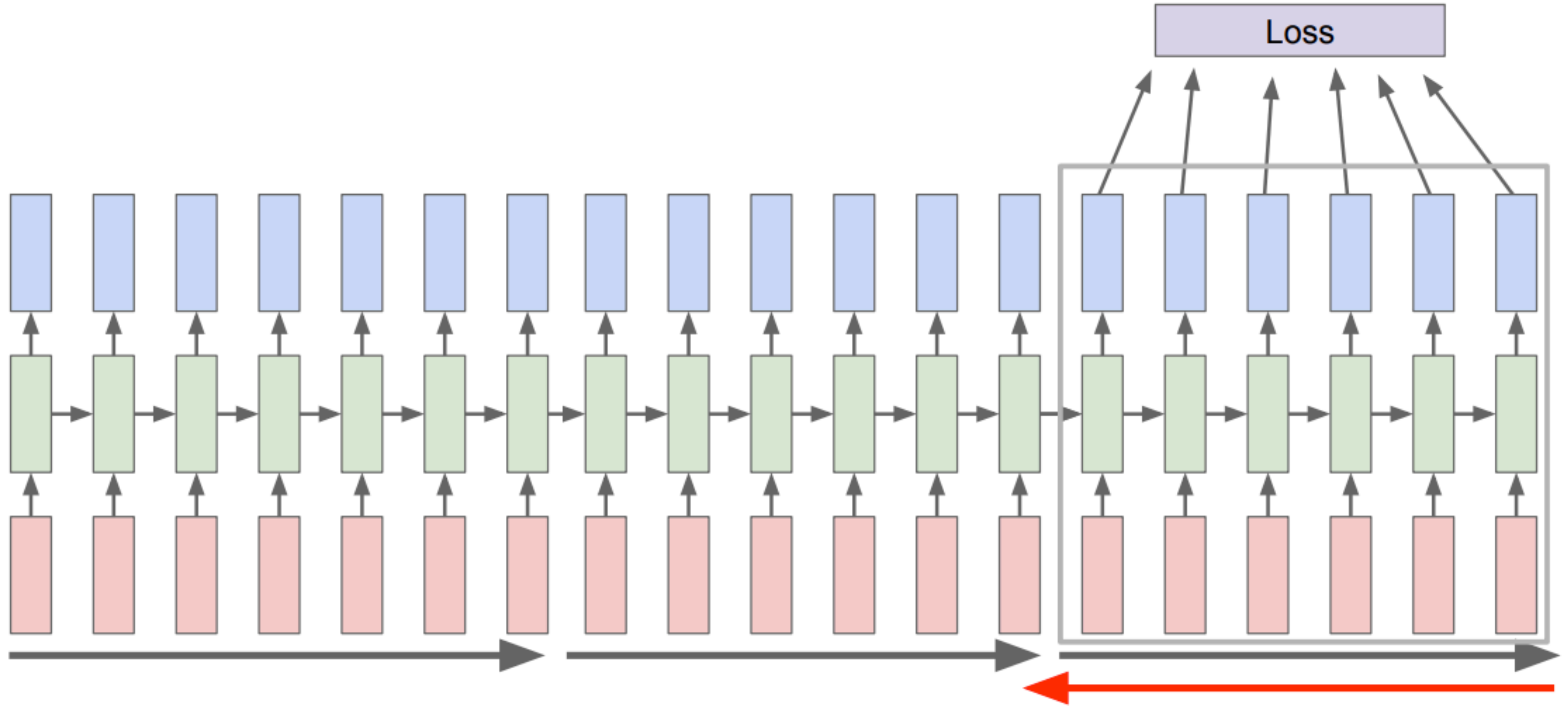


- Loss = negative log likelihood of probability of gold prediction tag
- Training uses stochastic gradient descent and back propagation
- Backpropagation through time (BPTT)
  - Run forward propagation
  - Run backward propagation
- The derivative $\frac{\partial L_t}{\partial \mathbf{V}} = \frac{\partial L_t}{\partial \mathbf{h_{t-1}}} \frac{\partial \mathbf{h_{t-1}}}{\partial V}$

Affected by $\frac{\partial \mathbf{h_{t-2}}}{\partial V}$

[Truncated] Backpropagation through time

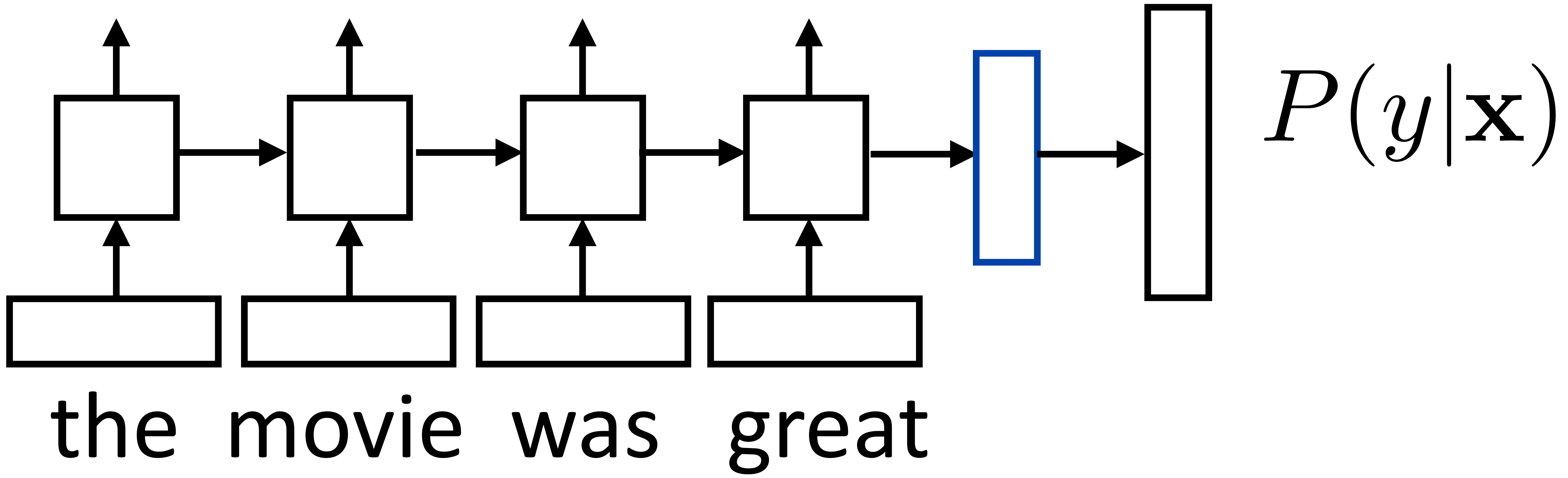
- Backpropagation is very expensive if you handle long sequences



- Run forward and backward through chunks of the sequence instead of whole sequence

- Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Training RNNs



- RNN potentially needs to learn how to "remember" information for a long time!

it was my favorite movie of 2016, though it wasn't without problems -> **+**

Long-term depedencies with RNN

- If the gradients becomes vanishingly small (or explodingly large) over long distances, models cannot easily capture long-term dependencies

  it was my favorite movie of 2016, though it wasn't without problems -> **+**

  - Key signal is from earlier hidden state, but gradient larger at later hidden states
  - Repeated multiplication by V causes problems

$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

- How to fix vanishing gradient problem?
  - LSTMs: Long short term memory networks

Gated Connections

- Designed to fix “vanishing gradient” problem using *gates*

$$\mathbf{h}_t = \mathbf{h}_{t-1} \odot \mathbf{f} + \mathrm{func}(\mathbf{x}_t)$$

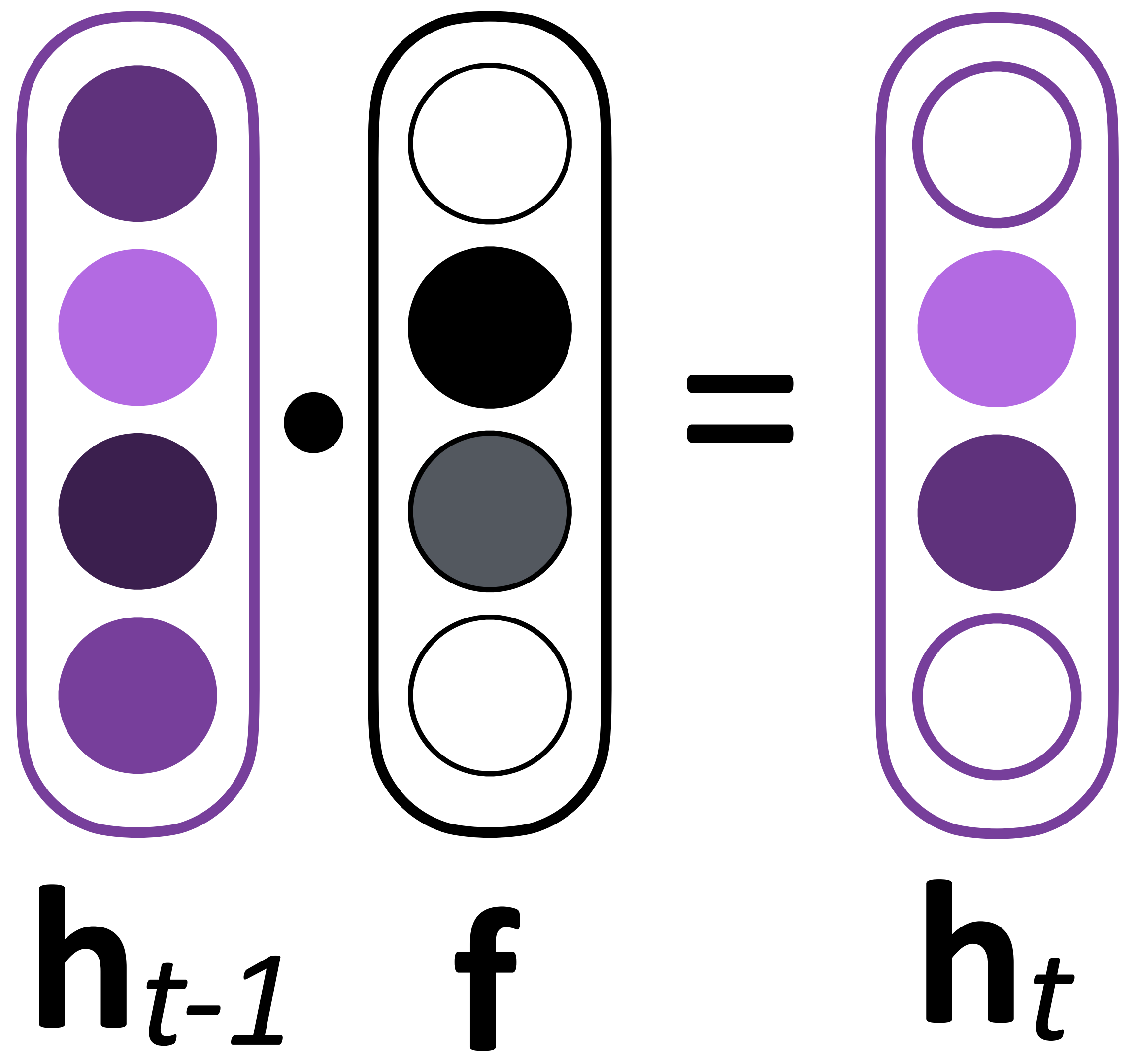
Gated

$$\mathbf{h}_t = \tanh(W\mathbf{x}_t + V\mathbf{h}_{t-1} + \mathbf{b}_h)$$

Elman

- Vector-valued “forget gate” **f** computed based on input and previous hidden state

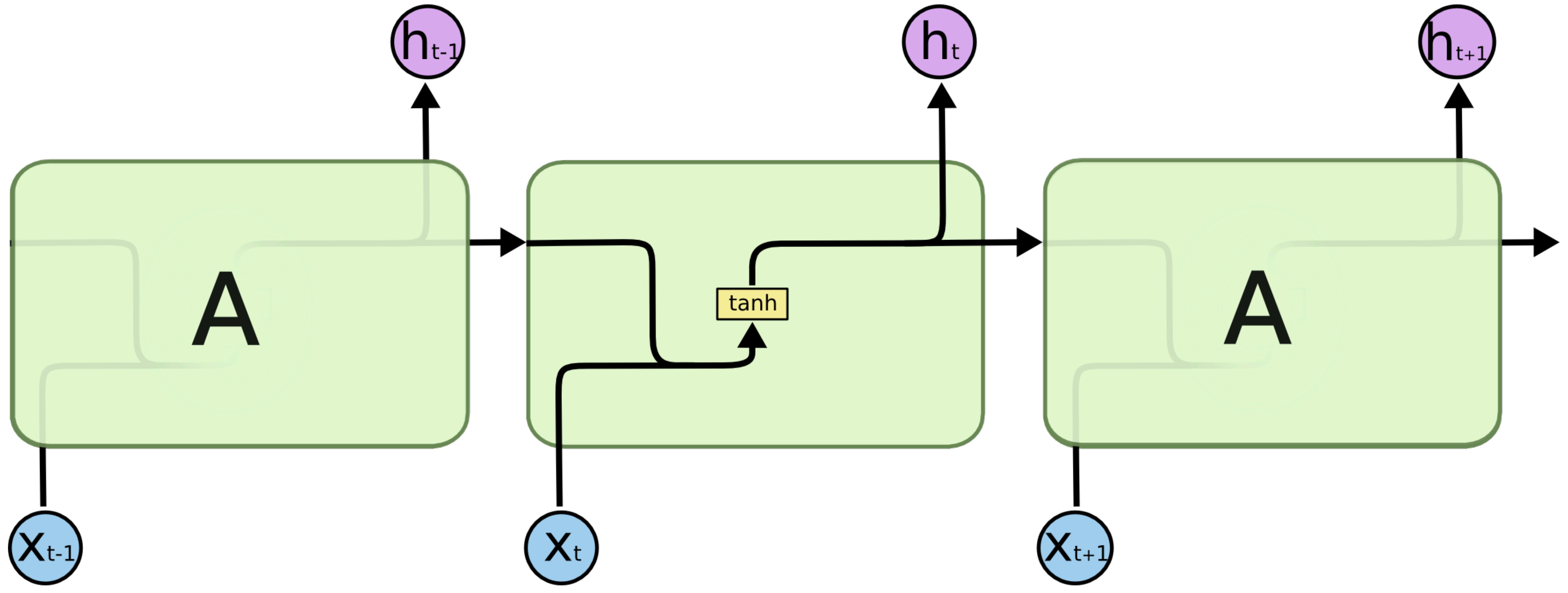
$$\mathbf{f} = \sigma(W^{xf}\mathbf{x}_t + W^{hf}\mathbf{h}_{t-1})$$

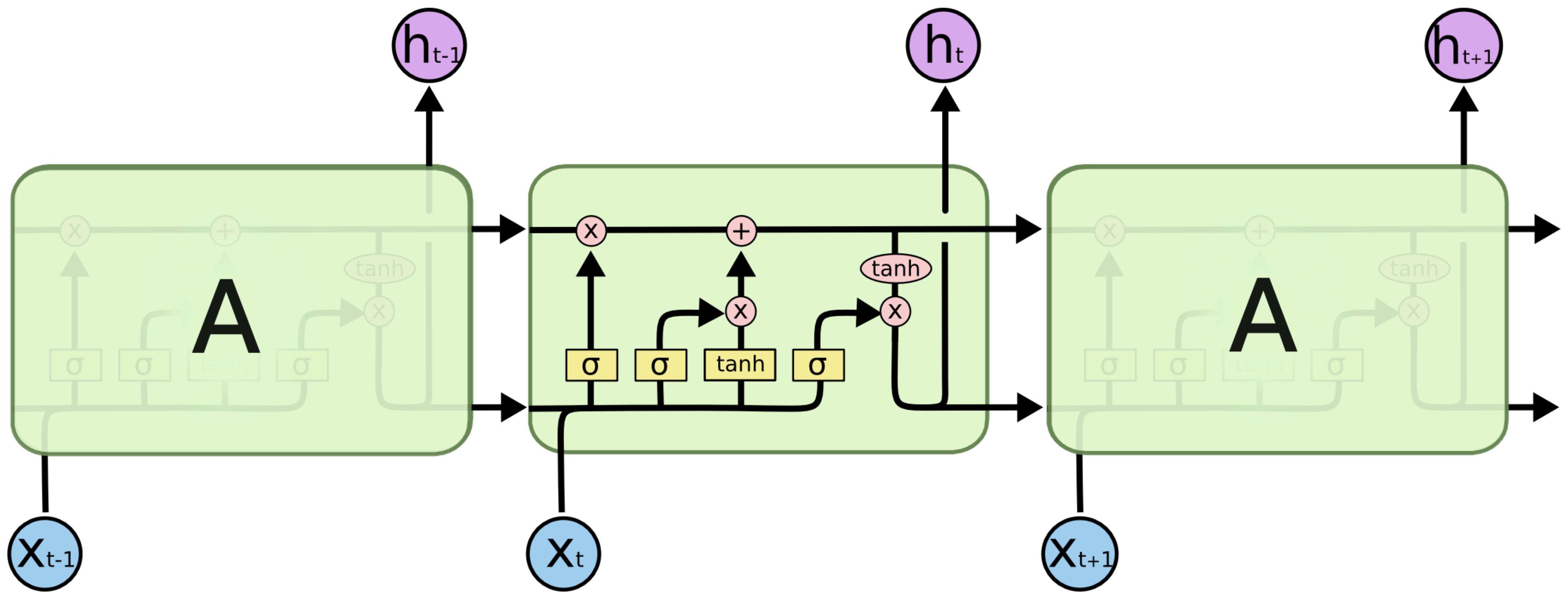


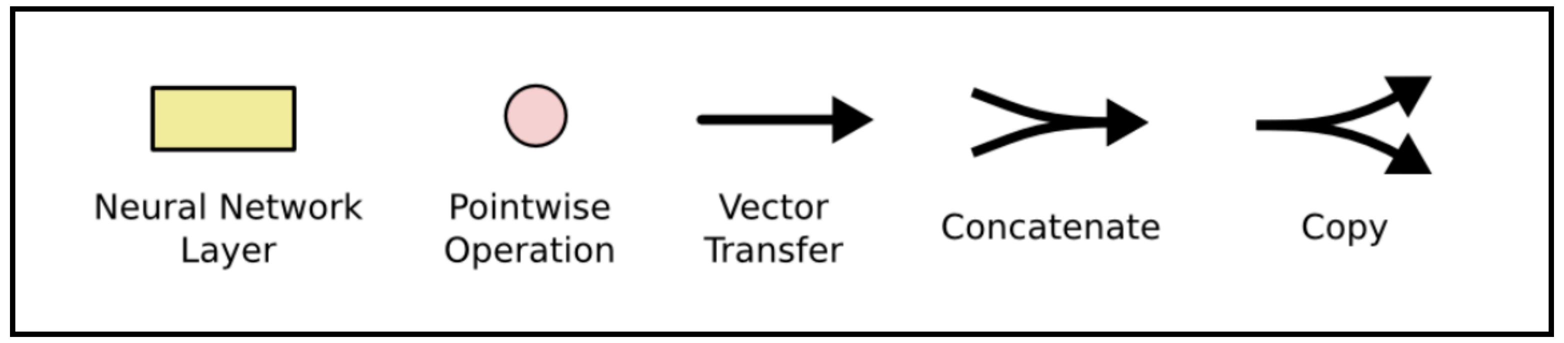
- Sigmoid: elements of **f** are in (0, 1)

- If **f** ≈ **1**, we simply sum up a function of all inputs — gradient doesn’t vanish! More stable without matrix multiply (*V*) as well

Elman RNN vs LSTMs



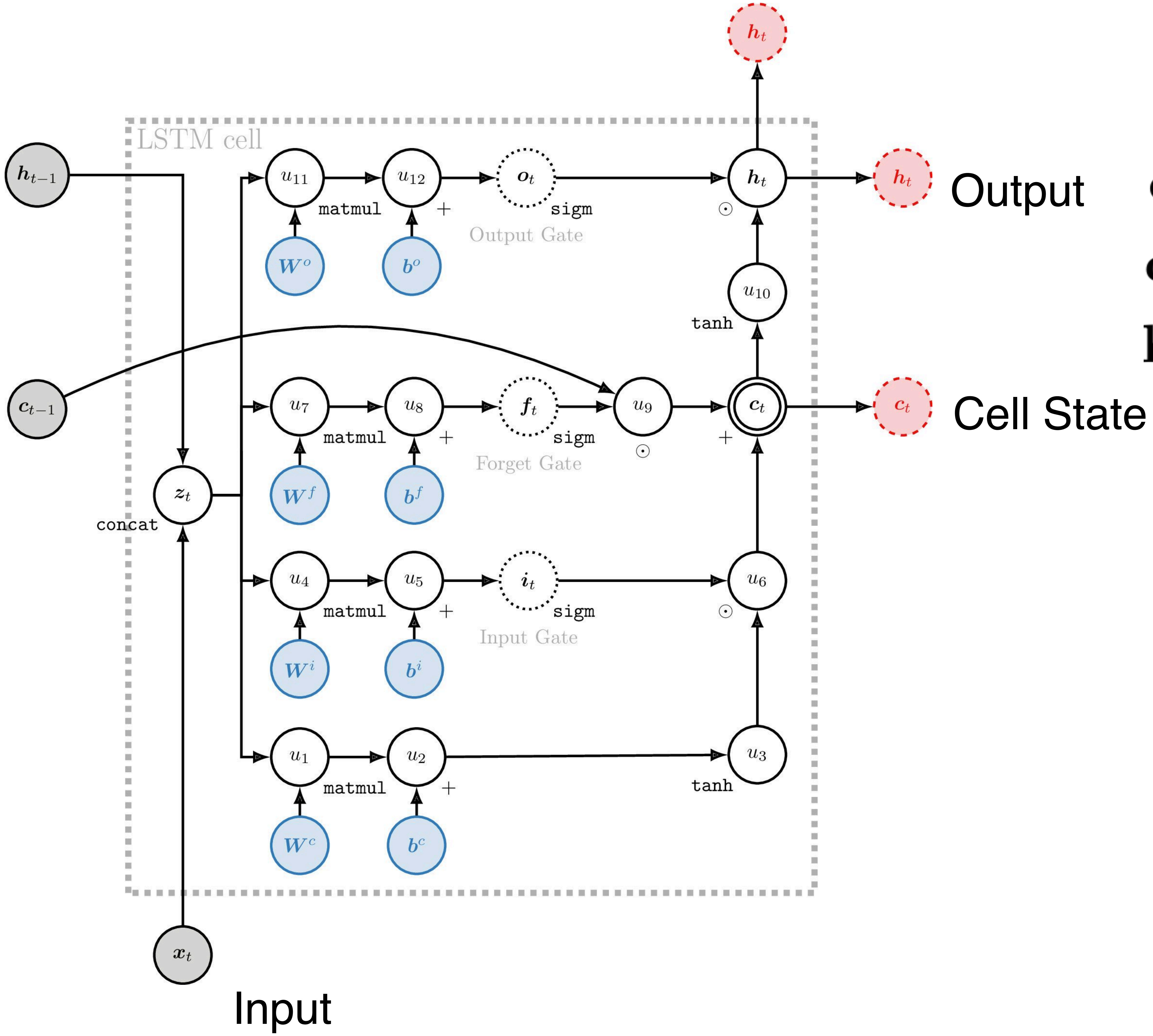




LSTMs

- "Long short-term memory": hidden state is a "short-term" memory
- "Cell" **c** in addition to hidden state **h** cell state —> stores long-term information
  - We write / erase cell c_i after each step
  - We read hidden state h_i from c_i
  - Basic communication flow: **x** -> **c** -> **h ->** output

LSTMs



$$\mathbf{f}_t = \sigma(\mathbf{W}^f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^f)$$

$$\mathbf{i}_t = \sigma(\mathbf{W}^i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^f)$$

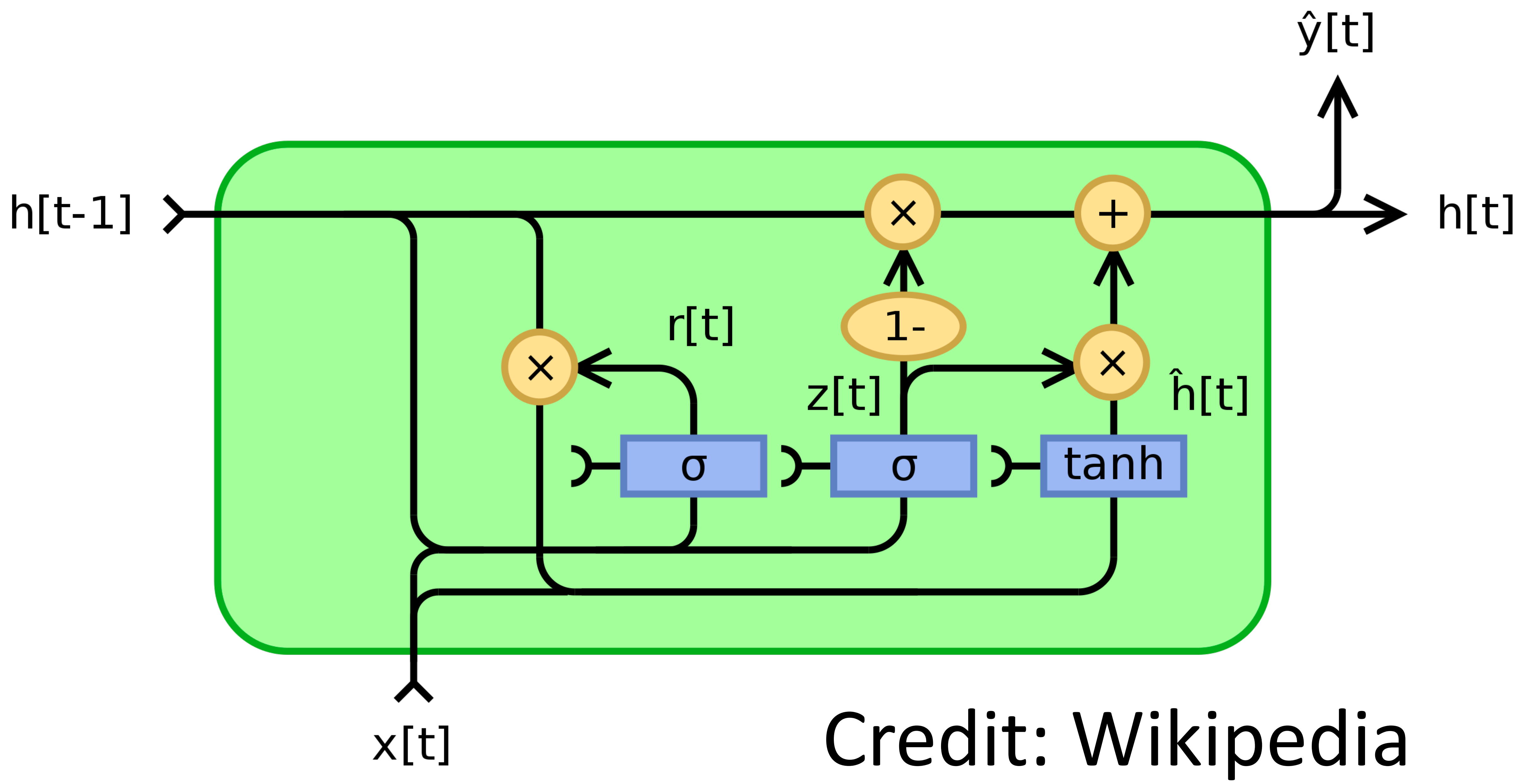
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}^c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^c)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}^o)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Gated Recurrent Unit (GRU)

- **z** is update, **r** is reset
- The single hidden state and simpler update gate gives simpler mixing semantics than in LSTMs
- Faster to train and sometimes works better than LSTMs, often a tossup

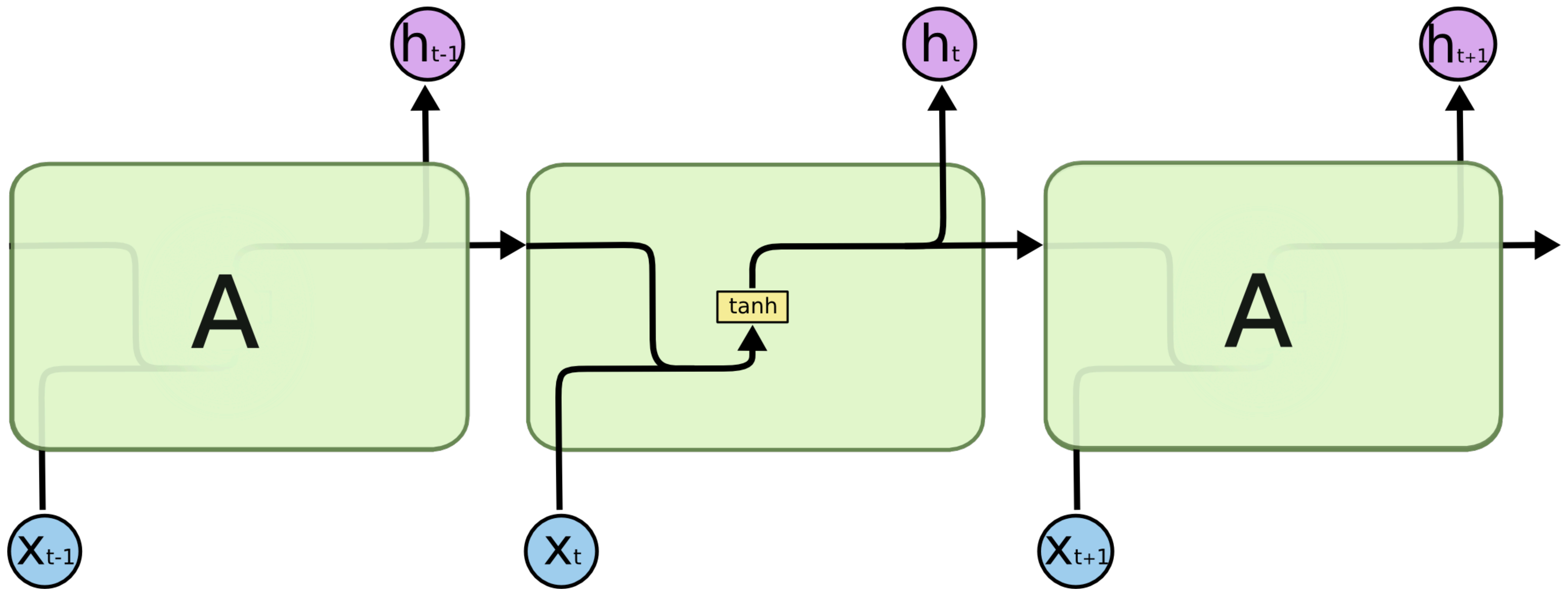


Credit: Wikipedia

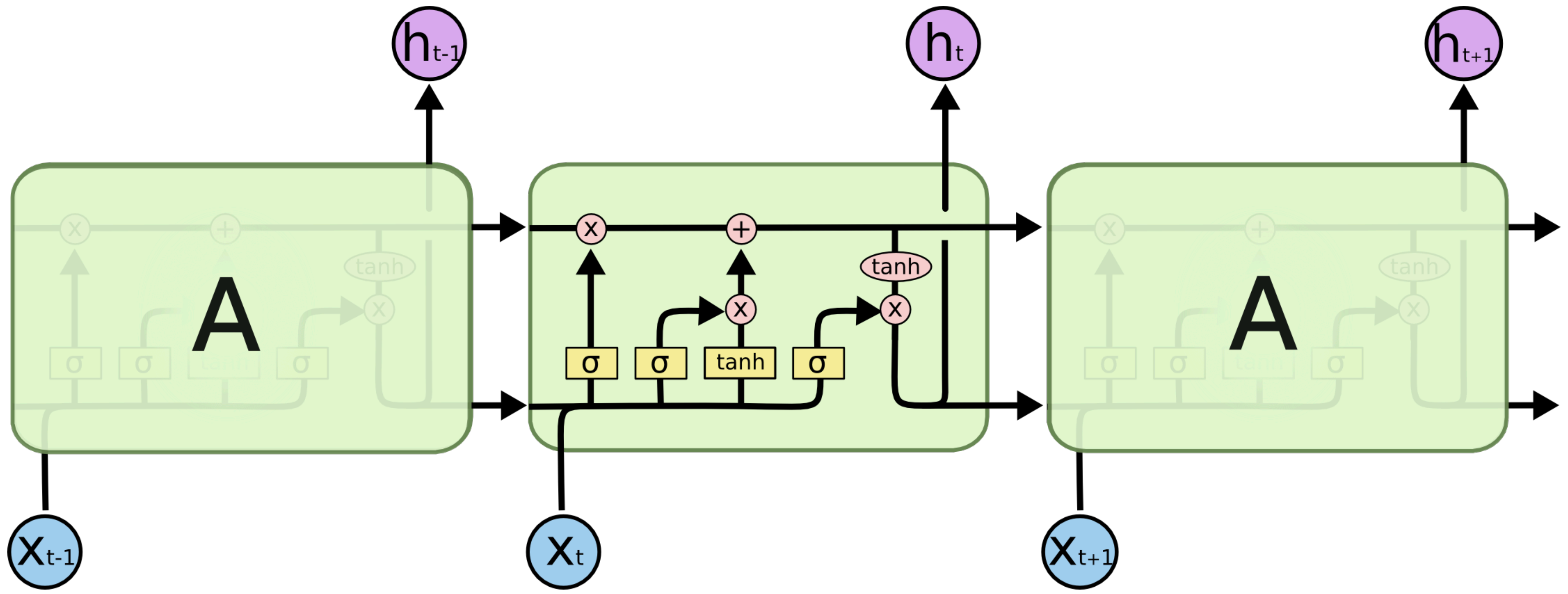
$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)
\end{aligned}
$$

Recap

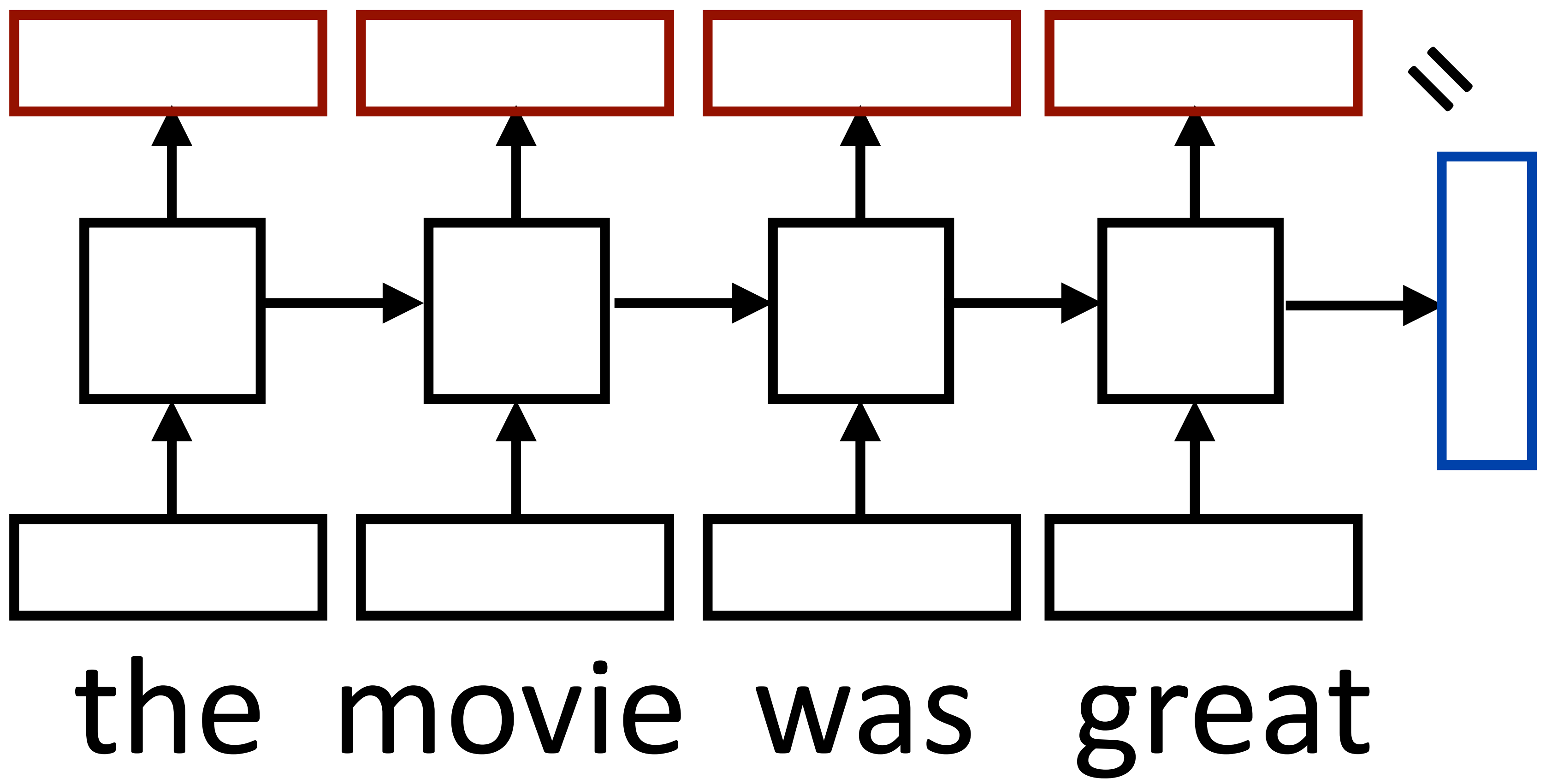
- Recurrent Neural Network
  - An architecture to handle variable length input
  - Preserves the states from previous tokens



- LSTM
  - Gating mechanism to control information flow, to maintain longer term history

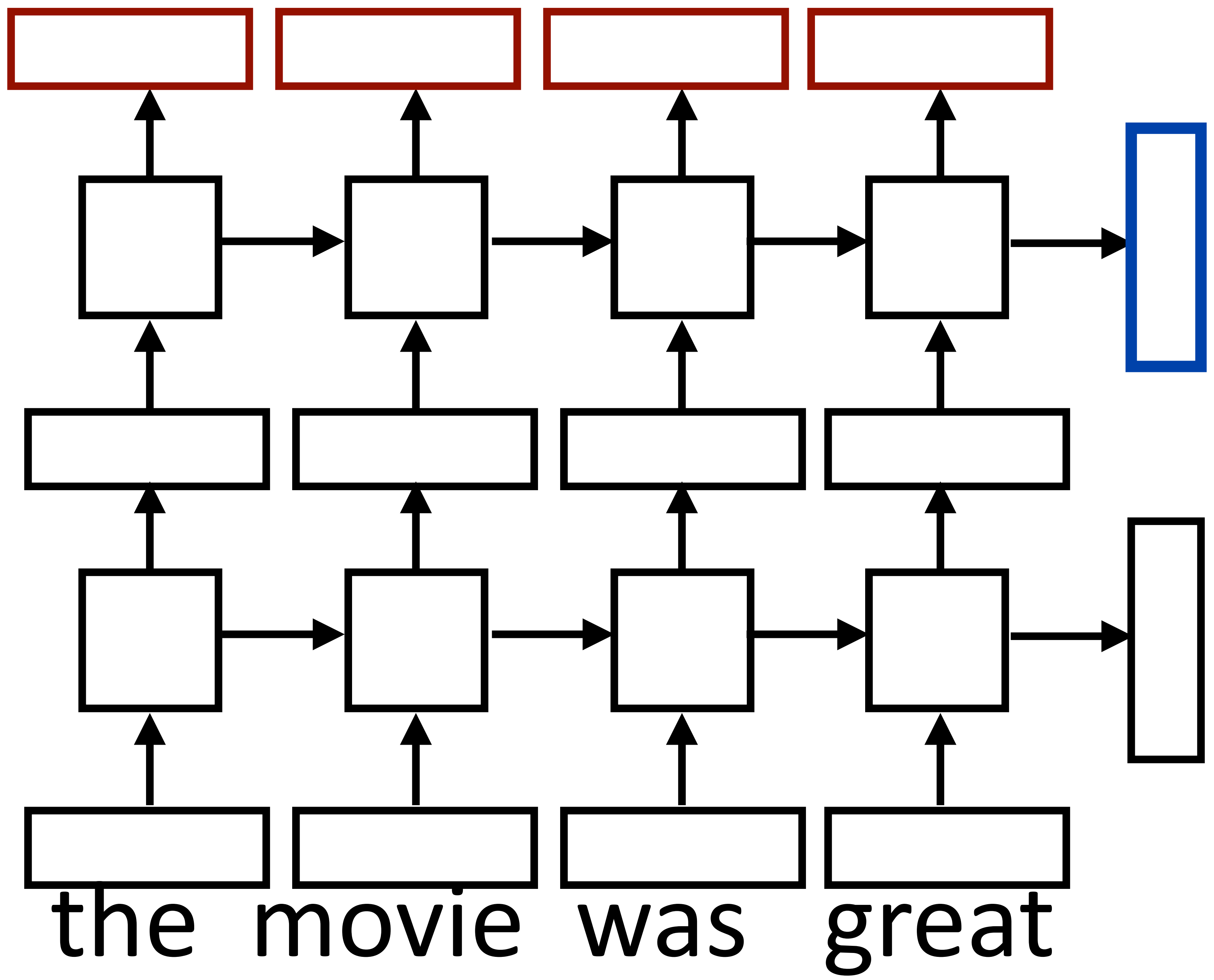


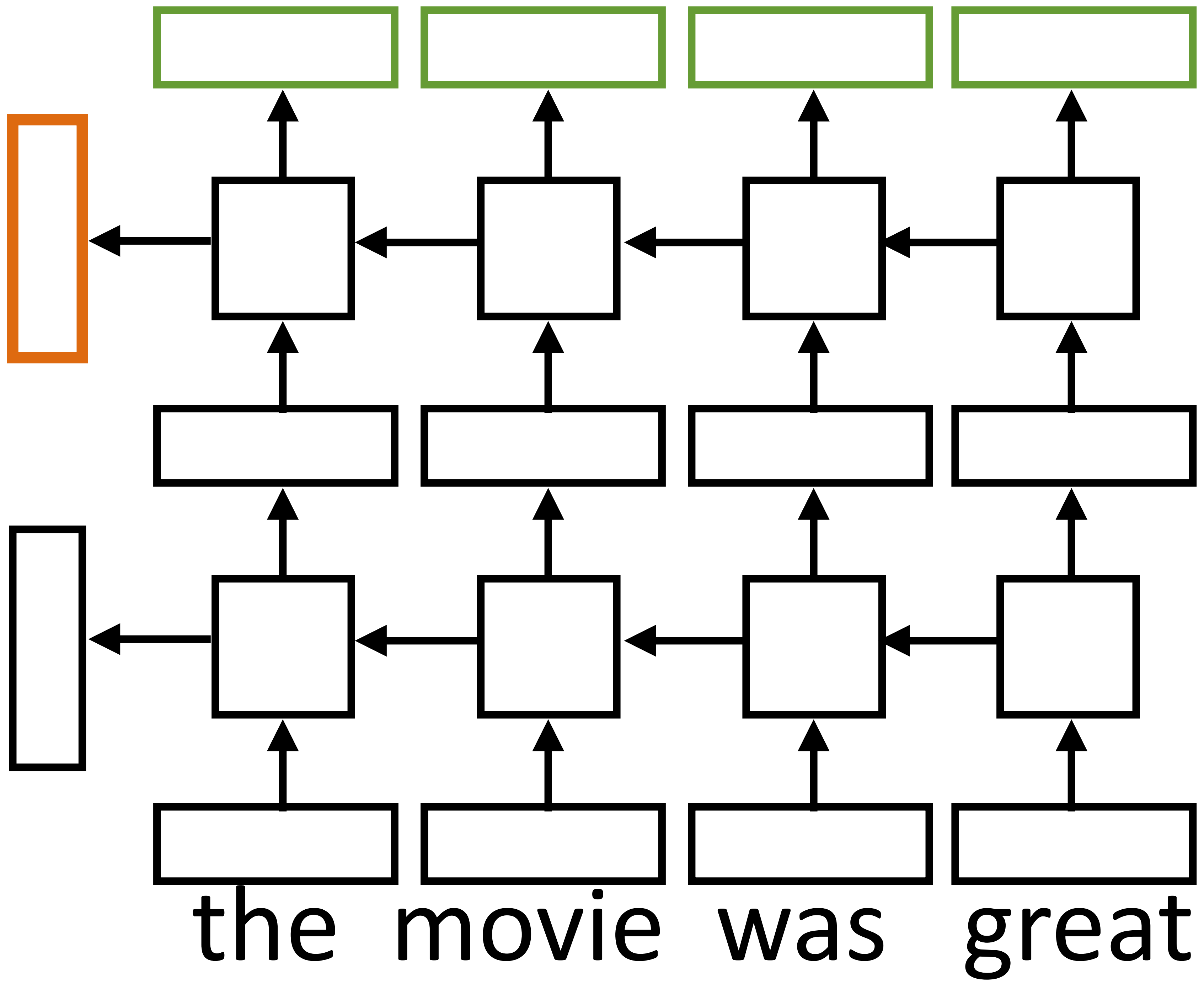
What do RNNs produce?



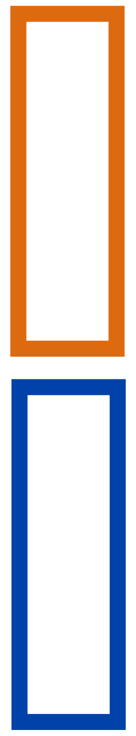
- Encoding of the sentence — can pass this a decoder or make a classification decision about the sentence
- Encoding of each word — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

Multilayer Bidirectional RNN

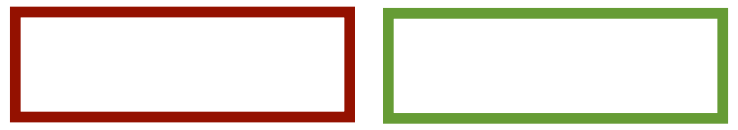




- Sentence classification based on concatenation of both final outputs

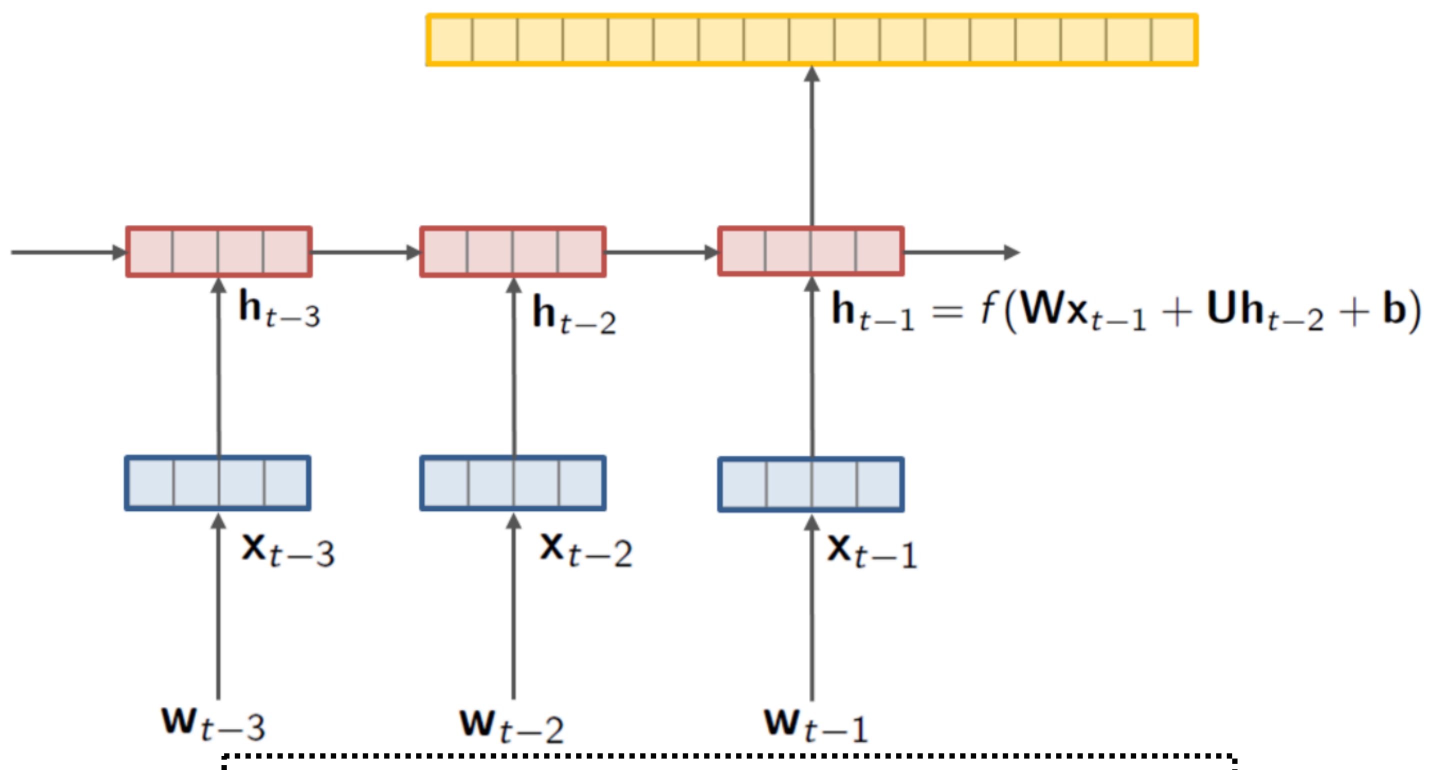


- Token classification based on concatenation of both directions' token representations

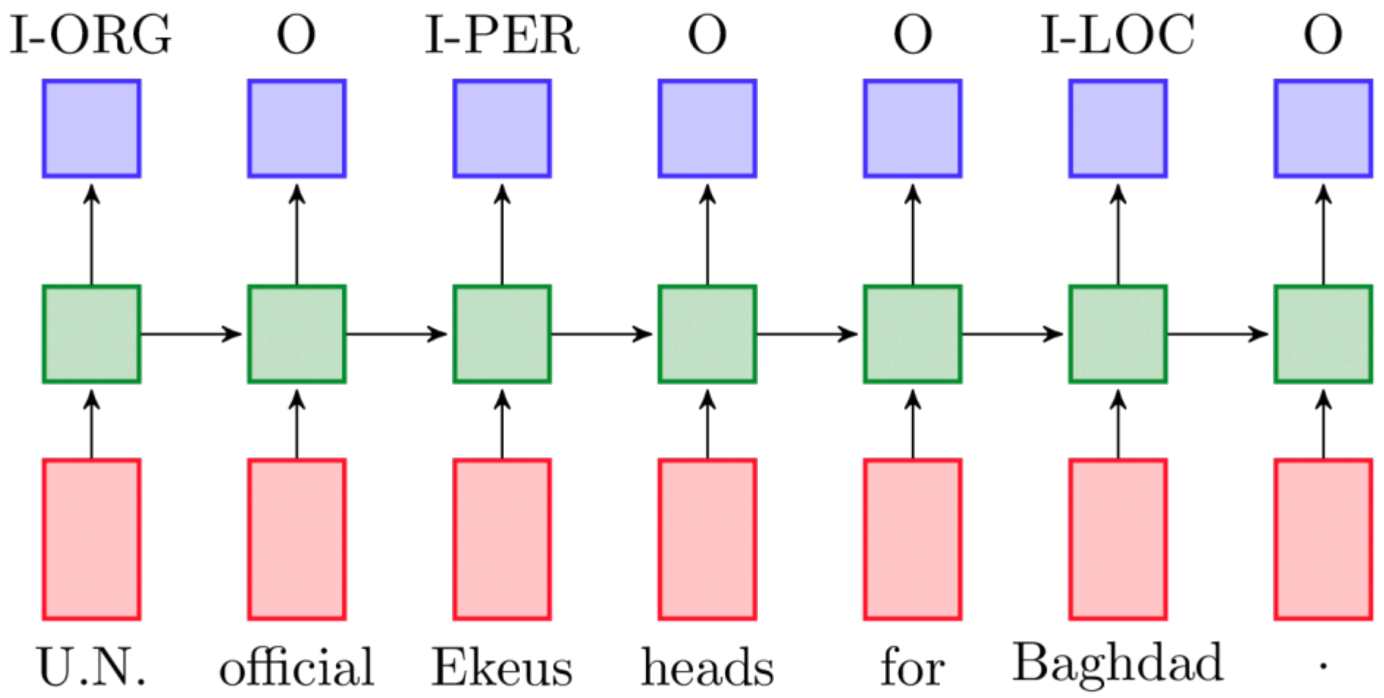


Recap: RNN can be used for…

- Transducer: make some prediction for each element in a sequence



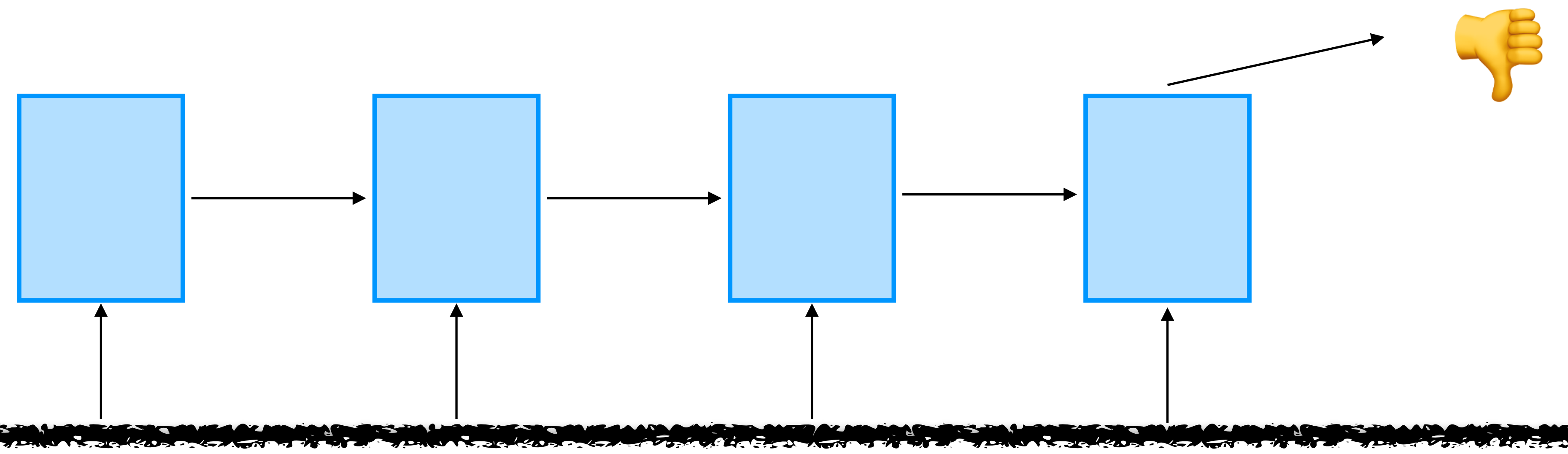
Language modeling



Sequence tagging

- Acceptor/encoder: encode a sequence into a fixed-sized vector and use that for some purpose

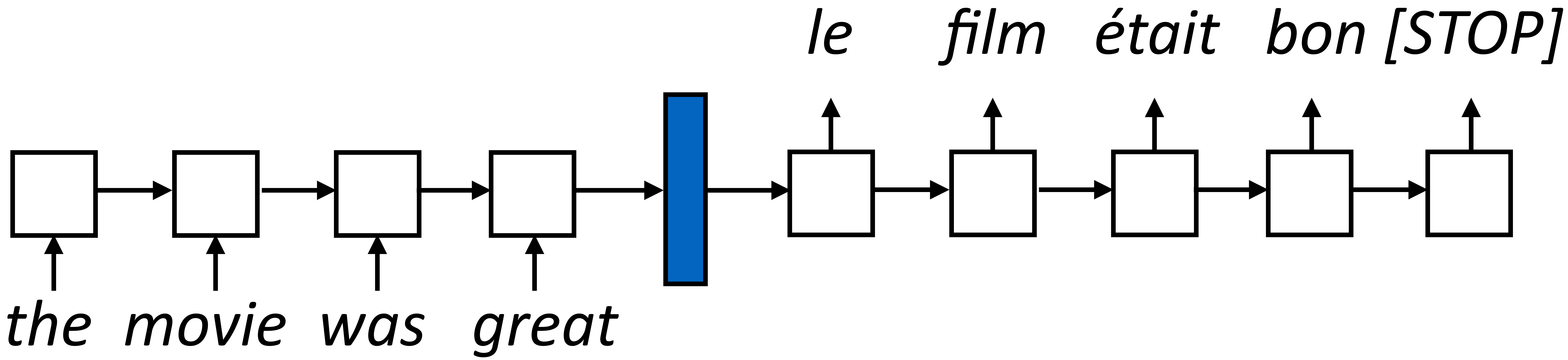
Text classification



How can we use this for machine translation? summarization?

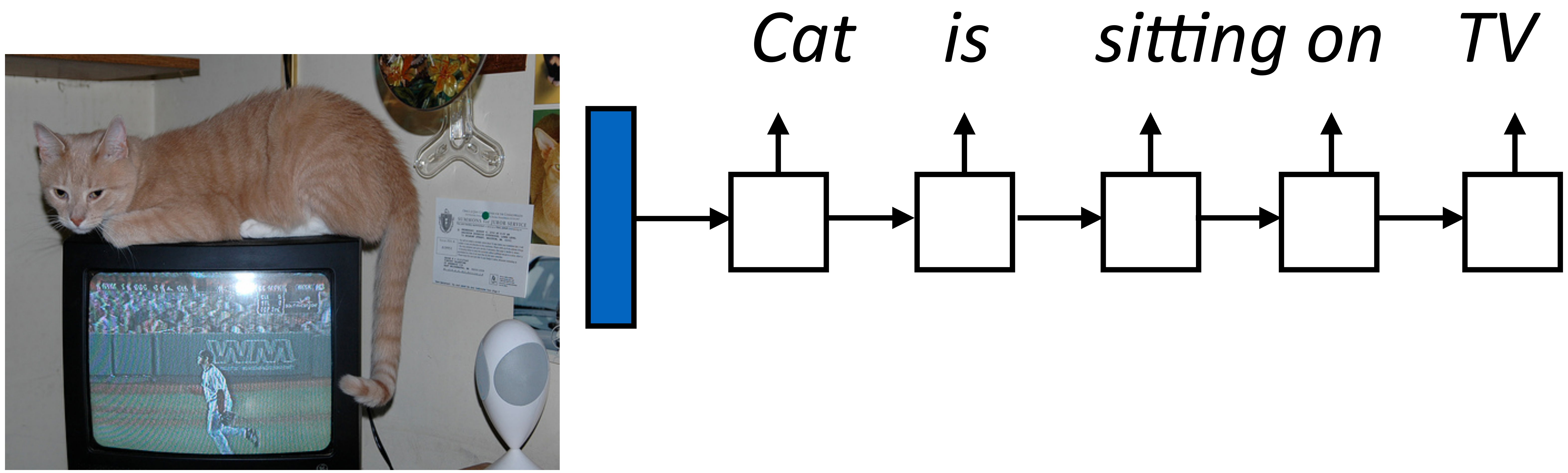
Seq2Seq Model

- Input: a sequence of tokens
- Output: a sequence of tokens (of ***arbitrary*** length)



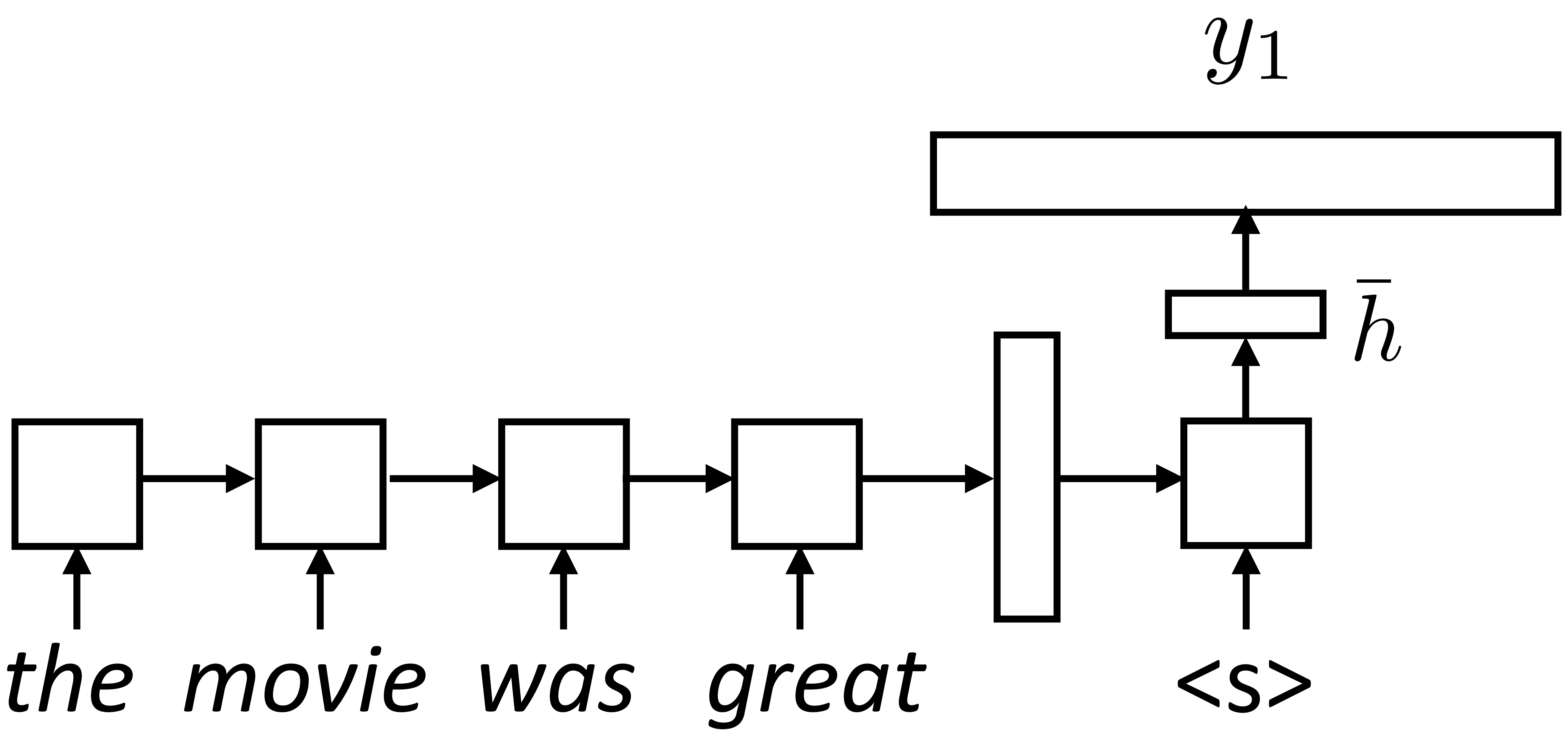
One2Seq

- Input: one item
- Output: a sequence of tokens (of ***arbitrary*** length)



Model

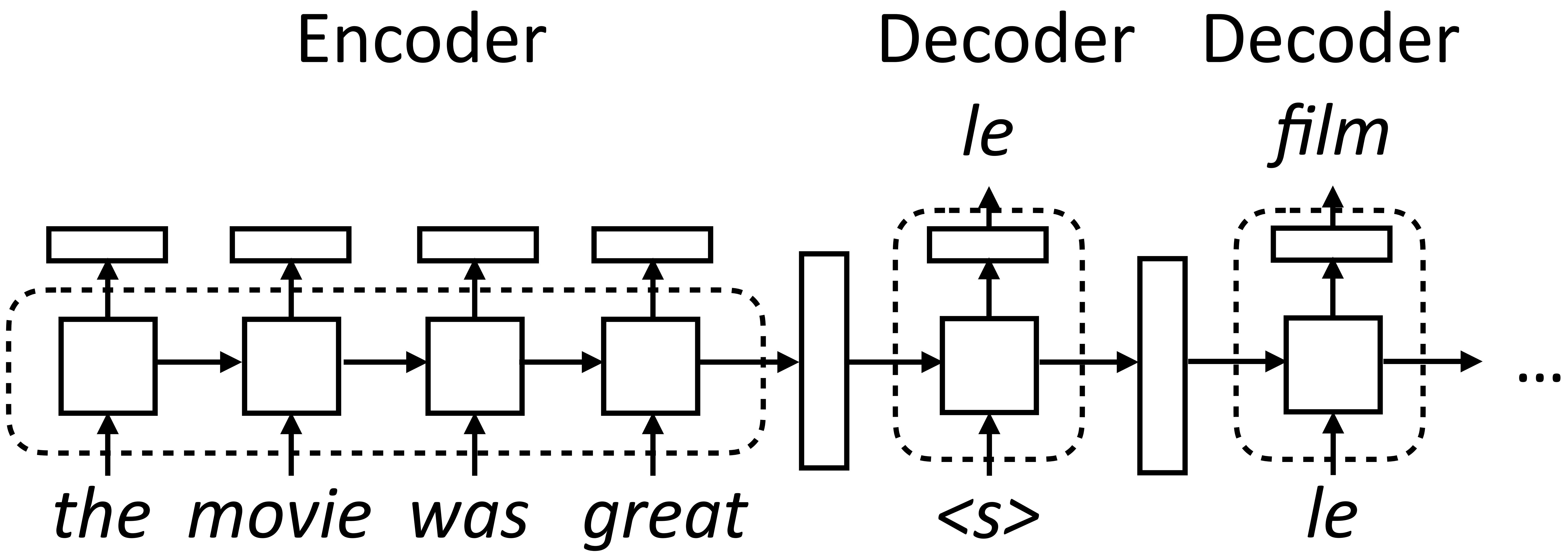
- Generate next word conditioned on previous word as well as hidden state



$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h})$$

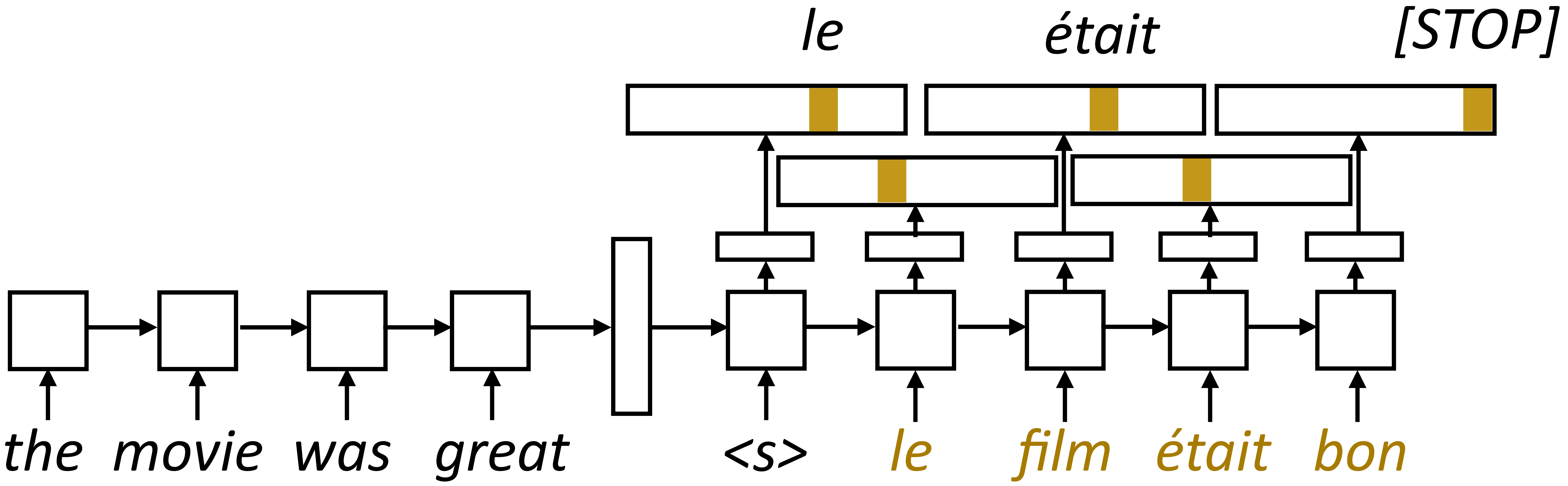
$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$

Implementing seq2seq Models



- Encoder: consumes sequence of tokens, produces a vector. Analogous to encoders for classification/tagging tasks
- Decoder: separate module, single cell.
  - Takes two inputs: hidden state and previous token.
  - Outputs token and a new hidden state.

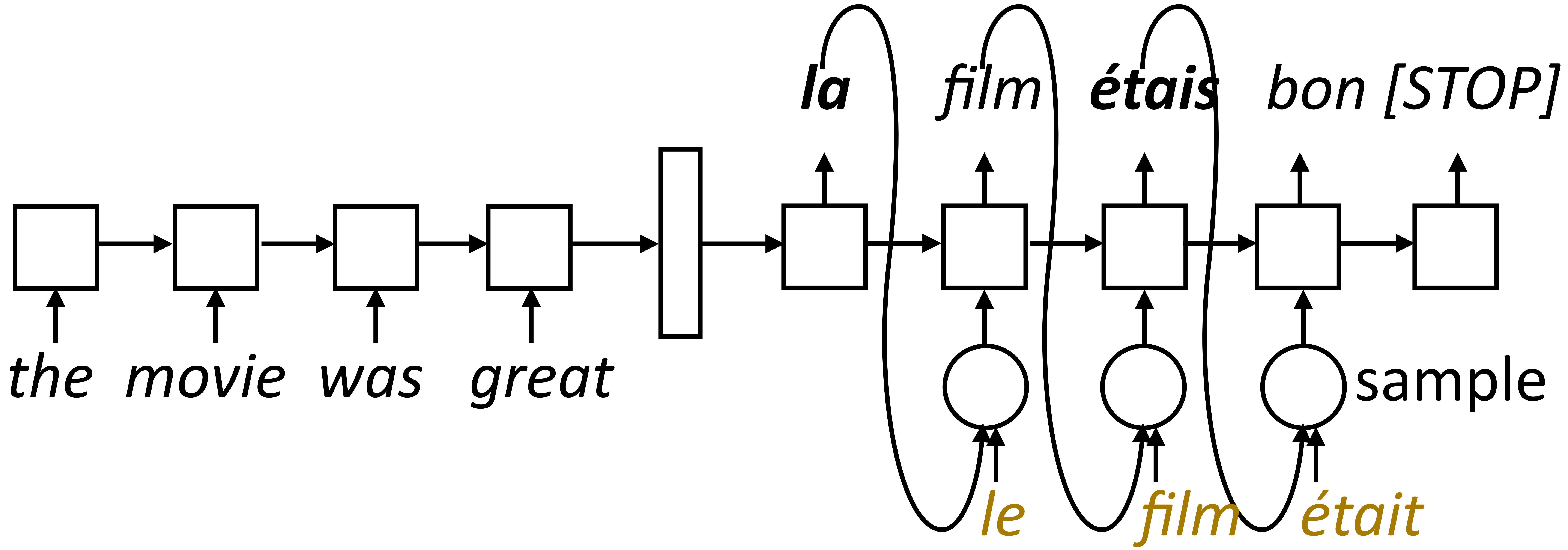
Training



- Objective: maximize $\sum_{(\mathbf{x},\mathbf{y})} \sum_{i=1}^{n} \log P(y_i^* | \mathbf{x}, y_1^*, \ldots, y_{i-1}^*)$
- One loss term for each target-sentence word, feed the correct word regardless of model's prediction (called "teacher forcing")

Training: Scheduled Sampling

- Model needs to do the right thing even with its own predictions

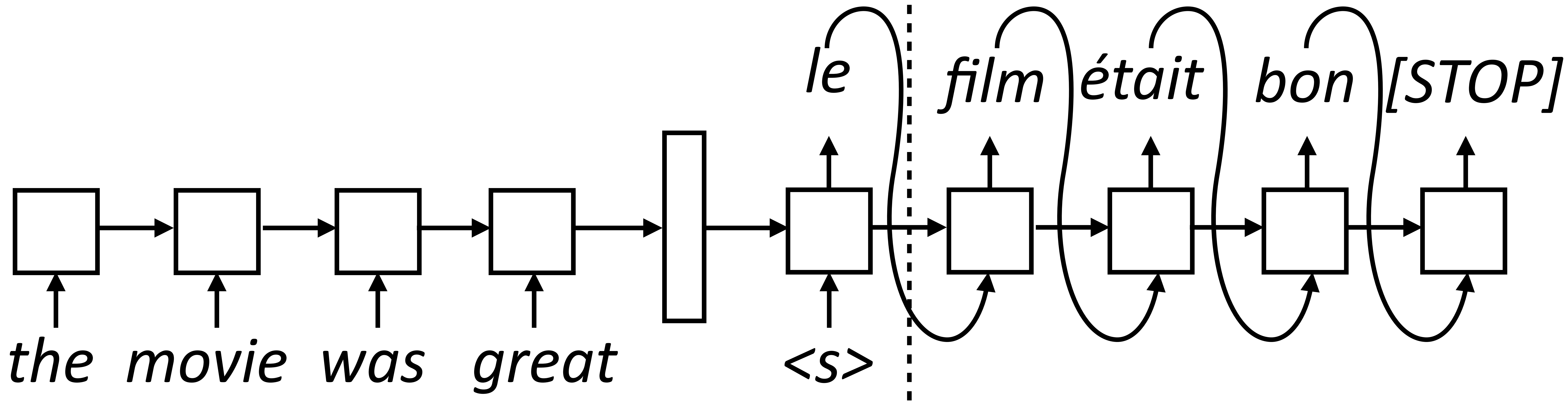


- Scheduled sampling: with probability *p*, take the gold as input, else take the model's prediction
- Starting with *p* = 1 (teacher forcing) and decaying it works best

Bengio et al. (2015)

Inference

- Generate next word conditioned on previous word as well as hidden state



- Need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state
- Decoder is advanced one state at a time until [STOP] is reached

Takeaways

- RNNs can transduce inputs (produce one output for each input) or compress the whole input into a vector

- Useful for a range of tasks with sequential input: sentiment analysis, language modeling, natural language inference, machine translation