Sequence to Sequence Models

Antoine Bosselut

EPFL

Announcements

- **Class Online:** March 11th.
  - IC Boost Day
  - Lectures will be pre-recorded and posted online

Section Outline

- **Mitigating Vanishing Gradients:** LSTMs, GRUs
- **Sequence-to-sequence models:** Overview, Examples, Training
- **Sequence-to-sequence shortcomings:** Long-range dependencies, Temporal bottleneck
- **Improvements:** Attention mechanisms

Last Week Recap

- Recurrent neural networks can **theoretically** learn to model an **unbounded context length**
  - no increase in model size because weights are shared across time steps
- Practically, however, **vanishing gradients** stop vanilla RNNs from learning useful **long-range dependencies**

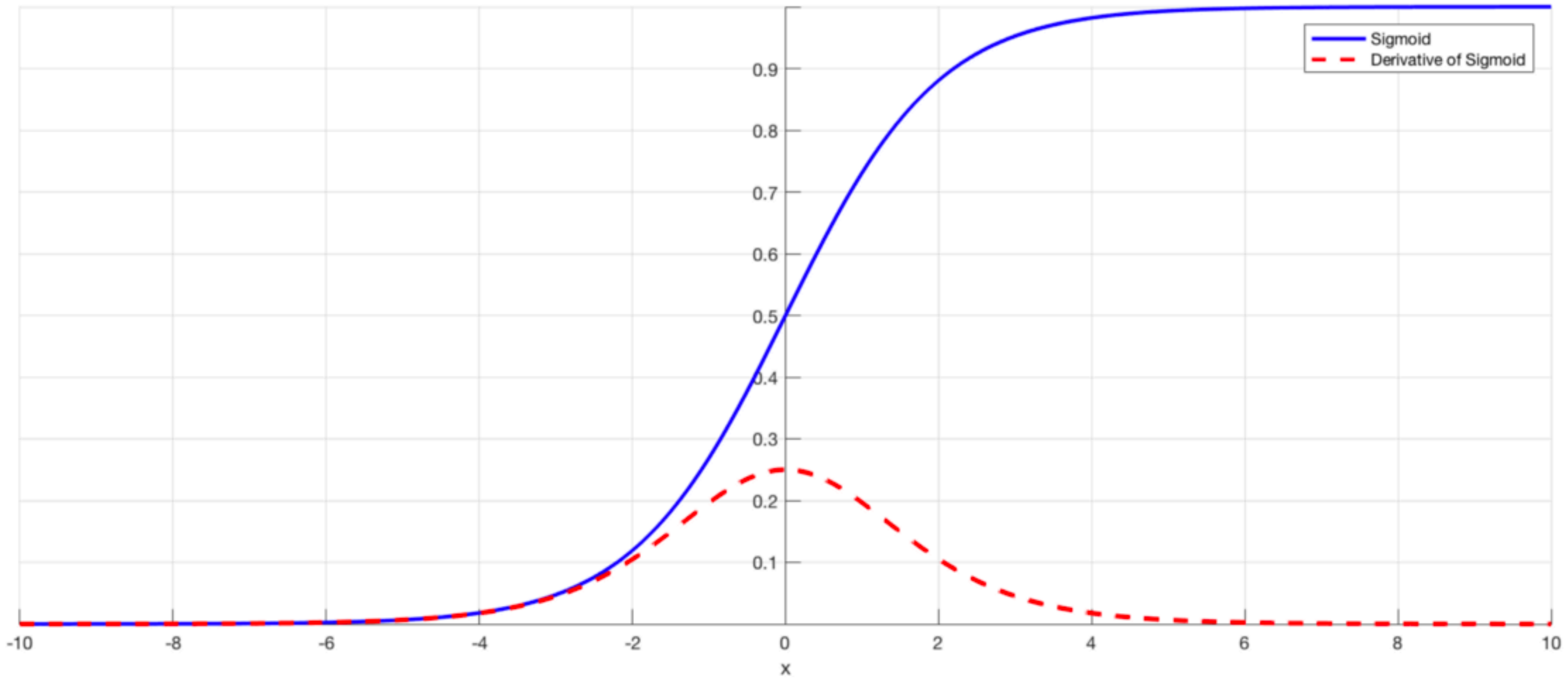
Vanishing Gradients

- **Learning Problem:** Long unrolled networks will crush gradients that backpropagate to earlier time steps

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$u = W_{hx}x_t + W_{hh}h_{t-1} + b_h$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = W_{hh}\frac{\partial \sigma(u)}{\partial u}$$

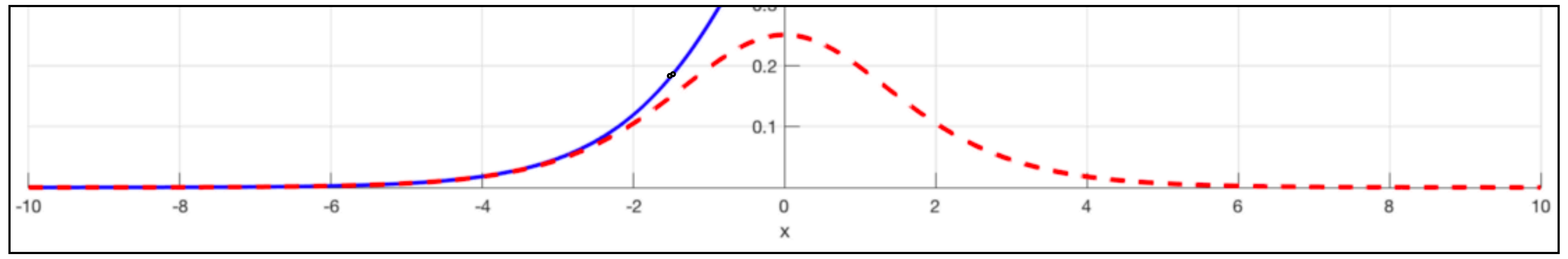


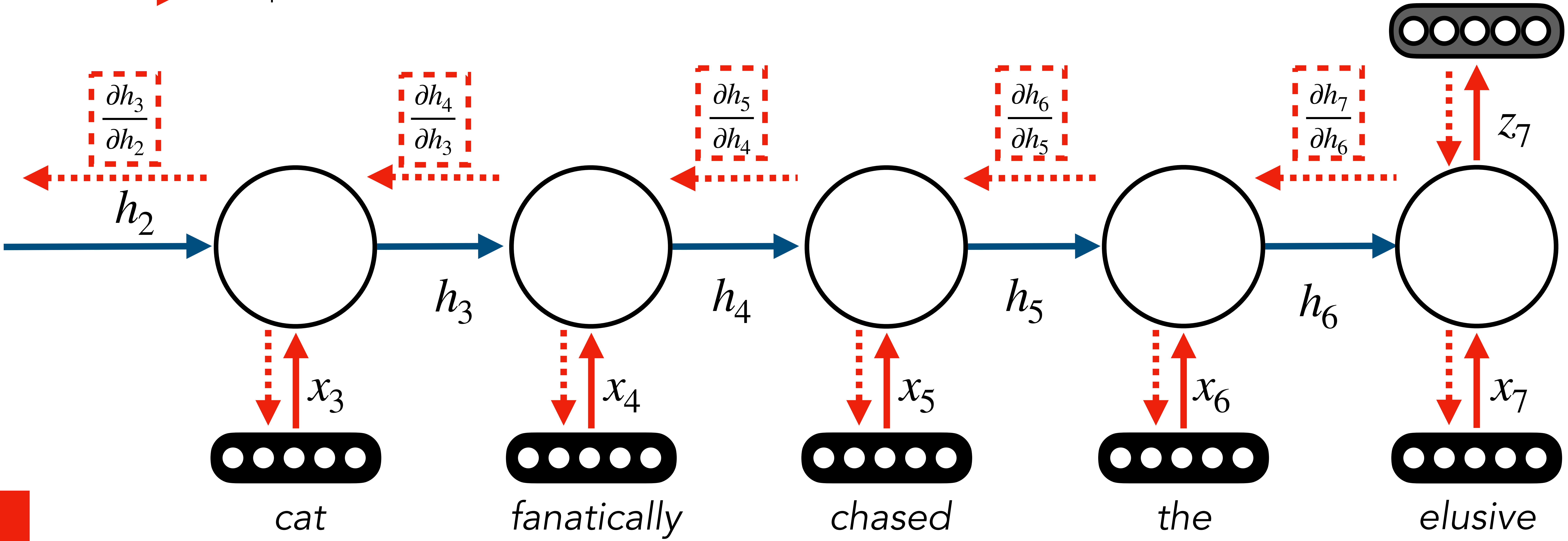
Vanishing Gradients

Gradient flow

Output flow

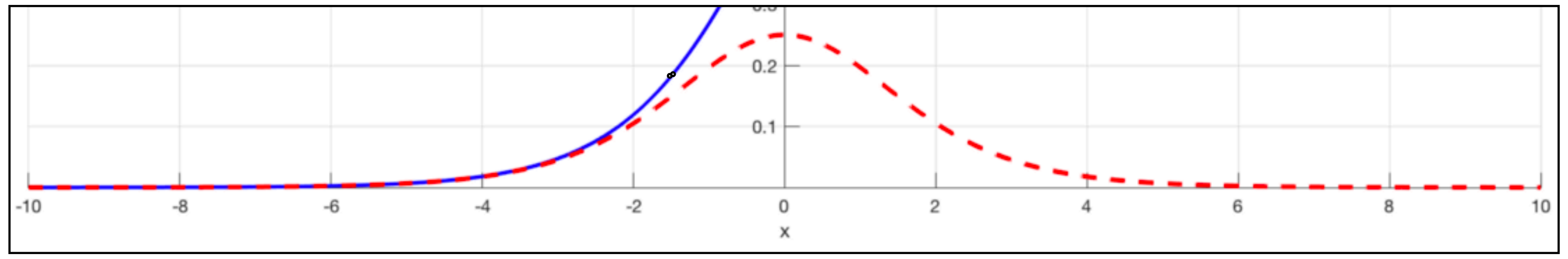
$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = W_{hh}\frac{\partial \sigma(u)}{\partial u}$$

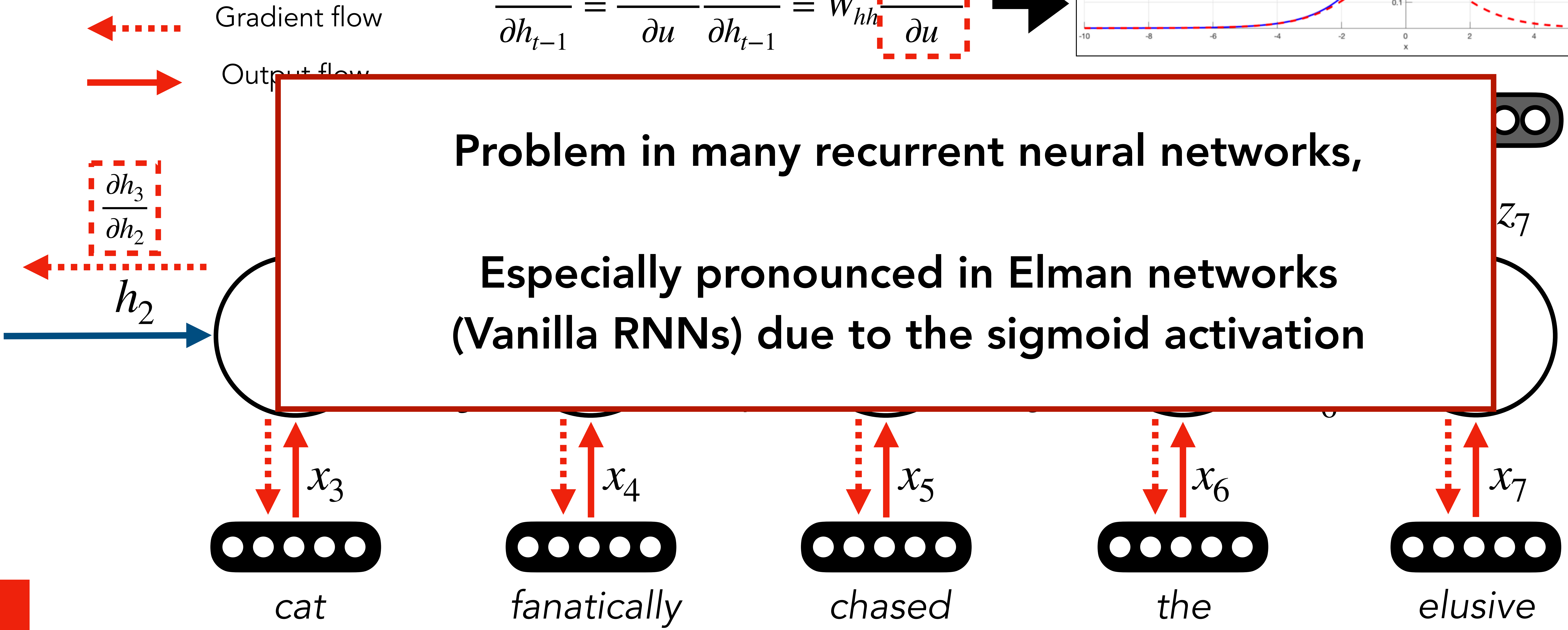




Vanishing Gradients

$$\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial \sigma(u)}{\partial u}\frac{\partial u}{\partial h_{t-1}} = W_{hh}\frac{\partial \sigma(u)}{\partial u}$$





Problem in many recurrent neural networks,

Especially pronounced in Elman networks (Vanilla RNNs) due to the sigmoid activation

Gated Recurrent Neural Networks

- Use gates to avoid dampening gradient signal every time step

$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

Elman Network

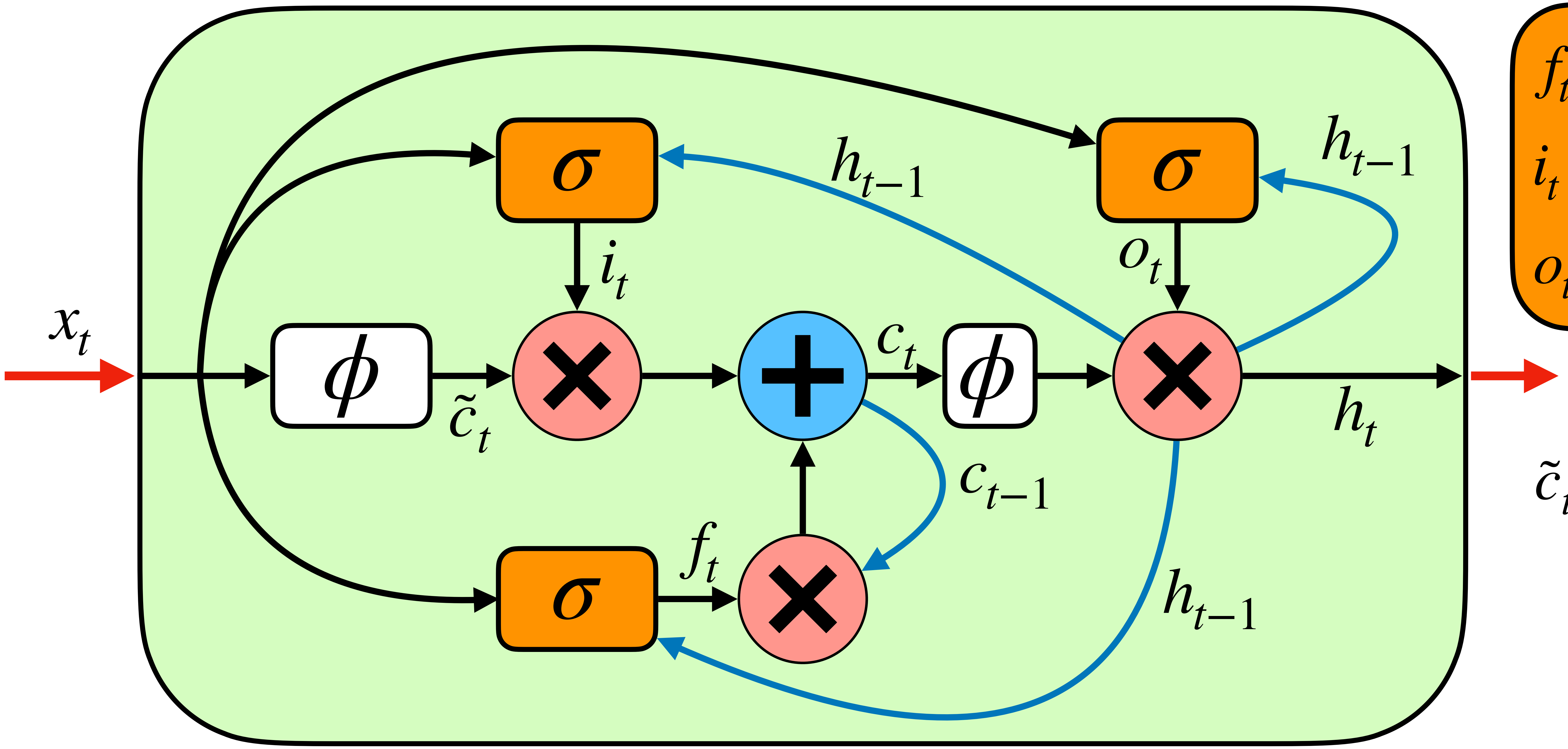
$$h_t = h_{t-1} \odot \mathbf{f} + \mathbf{func}(x_t)$$

Gated Network Abstraction

- Gate value **f** computes how much information from previous hidden state moves to the next time step —> 0 < **f** < 1

- Because h_{t-1} is no longer inside the activation function, it is not automatically constrained, reducing vanishing gradients!

Long Short Term Memory (LSTM)



Gates:

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$
$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$

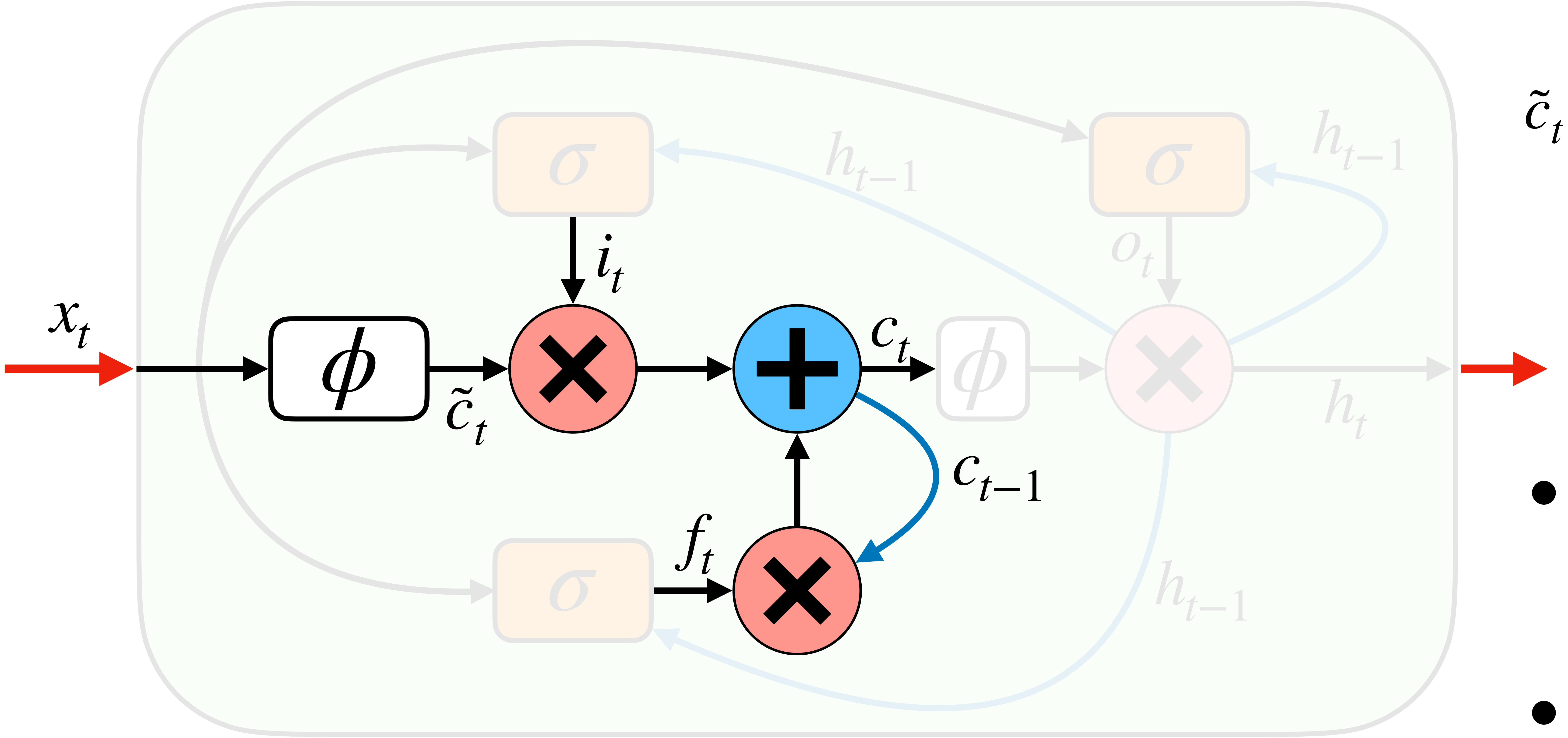
$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

$$h_t = o_t \times \phi(c_t)$$

(Hochreiter and Schmidhuber, 1997)

Cell State

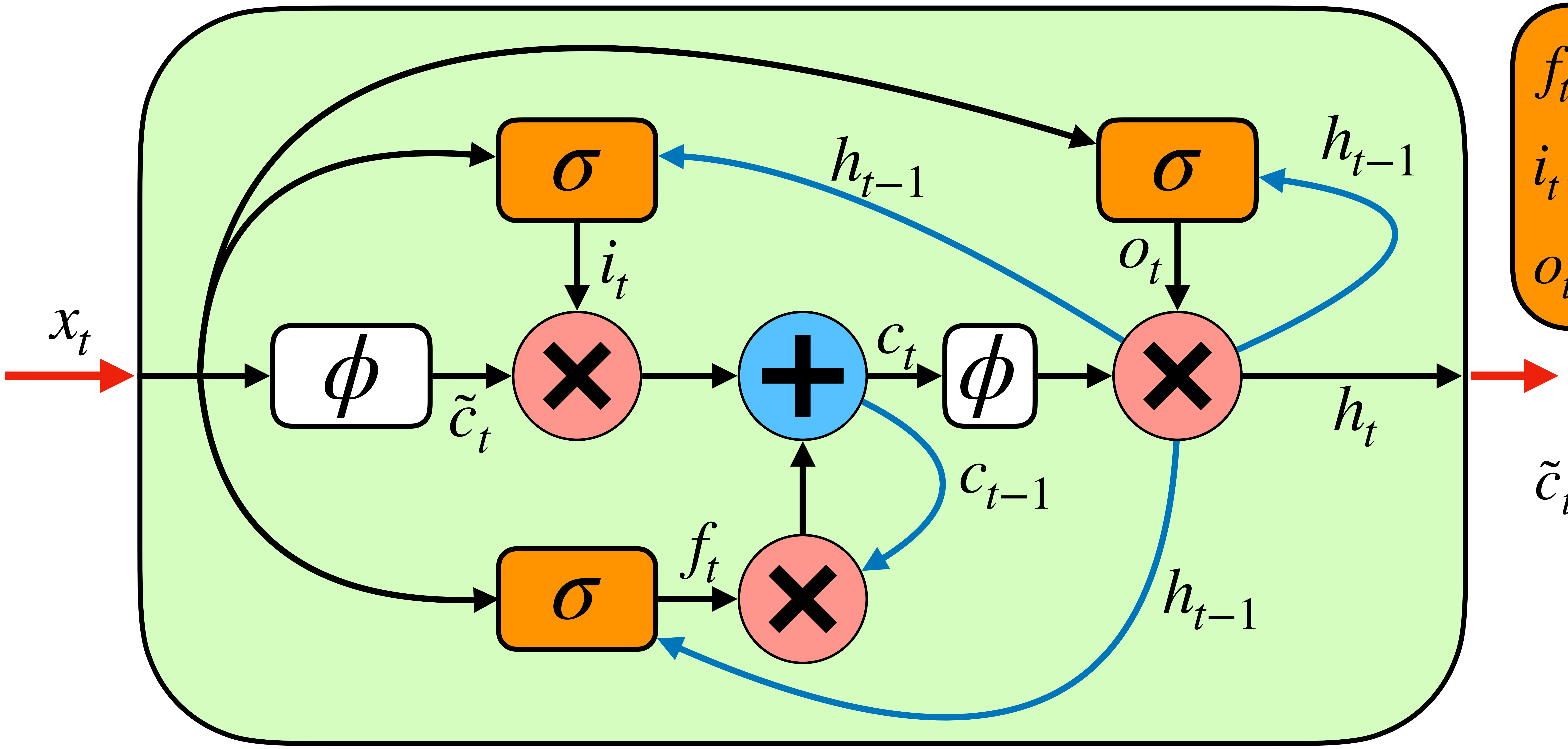


$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$

$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$

- Hidden state h_{t-1} is now short-term memory
- Cell state c_t tracks longer-term dependencies

Long Short Term Memory (LSTM)



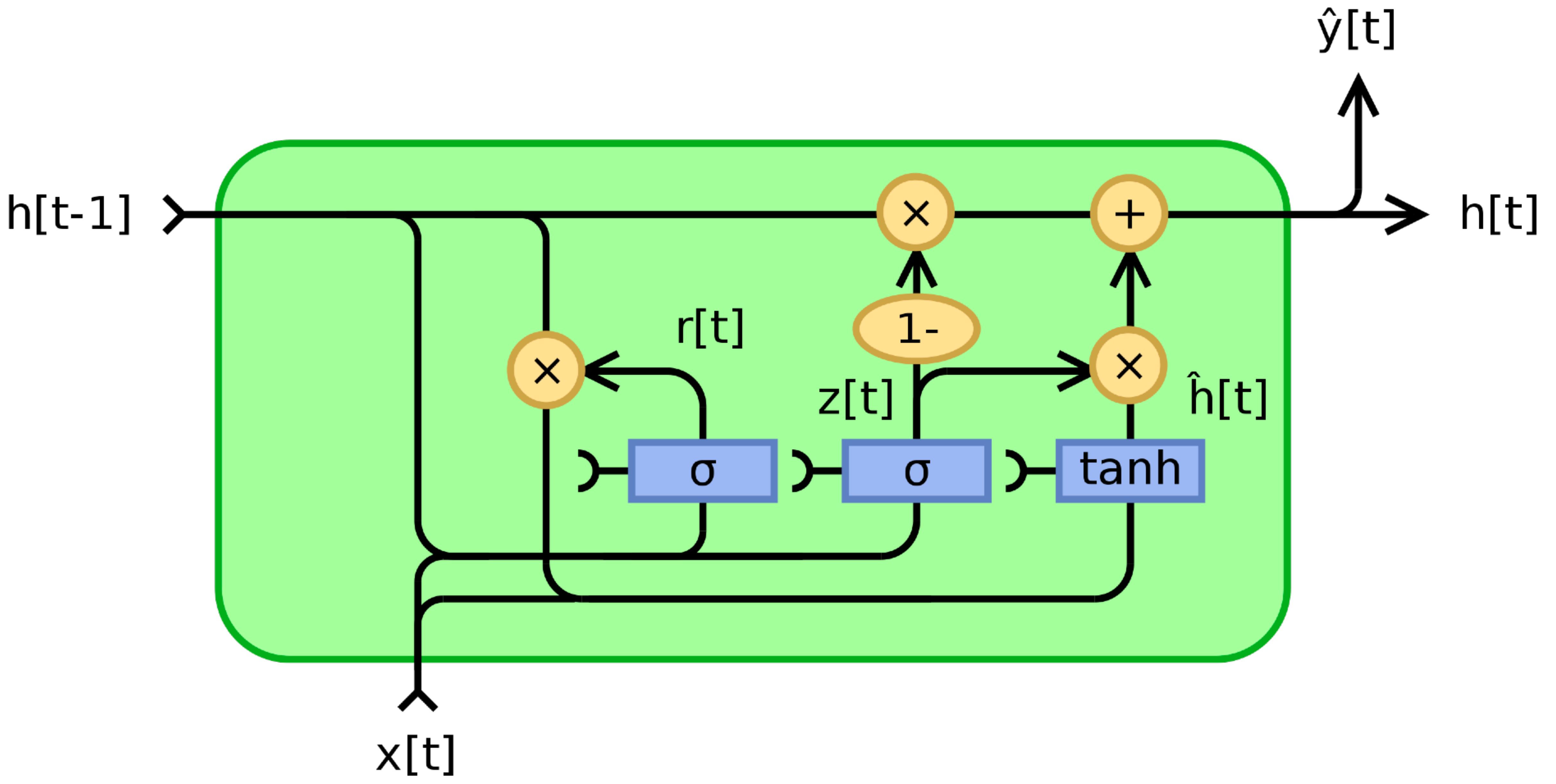
Gates:

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$
$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$

$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$
$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$
$$h_t = o_t \times \phi(c_t)$$

Gated Recurrent Unit (GRU)

- **z** is update gate (used to update hidden state), **r** is reset gate (used to reset hidden state)
- The single hidden state and simpler update gate gives simpler mixing algorithm than in LSTMs



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

Gated Recurrent Unit (GRU)

- Also uses gates to avoid dampening gradient signal every time step

$$h_t = (1 - \mathbf{z}) \odot h_{t-1} + \mathbf{z} \odot \textbf{func}(x_t, h_{t-1}) \qquad\qquad h_t = h_{t-1} \odot \mathbf{f} + \textbf{func}(x_t)$$

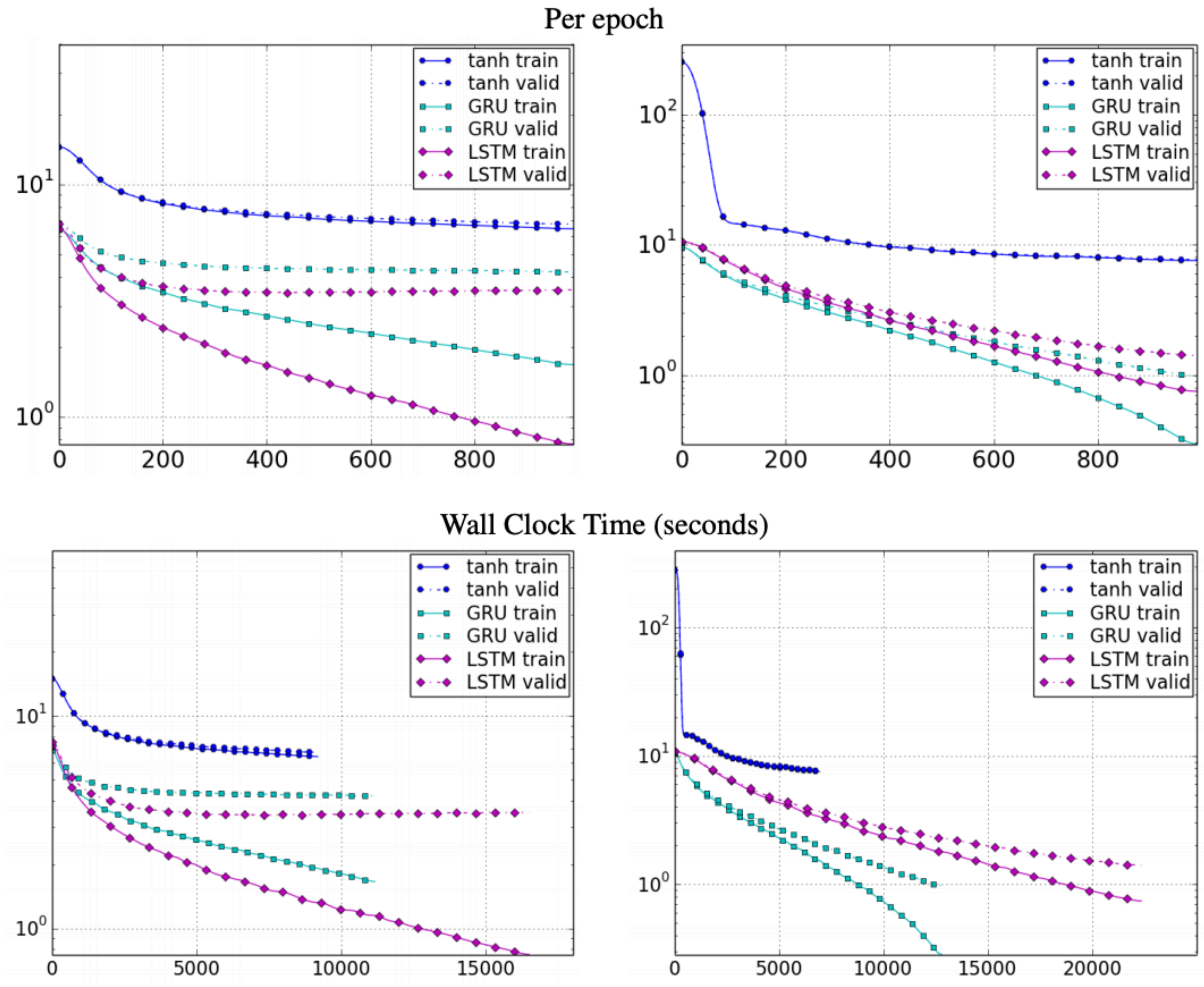
GRU **LSTM**

- Works similarly to LSTM
  - Typically faster to train and sometimes works better than LSTMs
  - Theoretically less powerful (for example, it can't count)

Which is better?

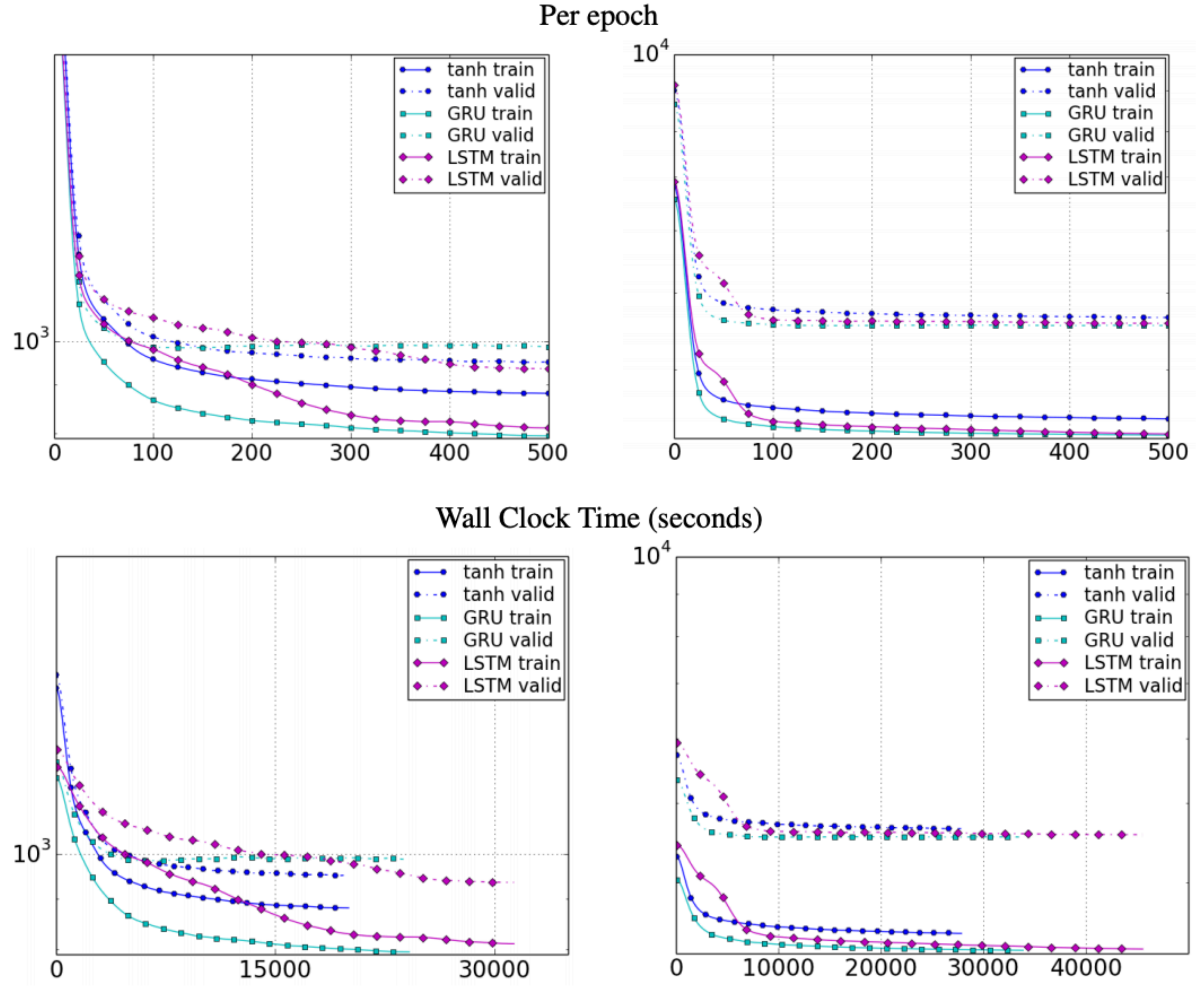
Negative Loglikelihood

Speech Signal Modeling



(a) Ubisoft Dataset A (b) Ubisoft Dataset B

Music Modeling



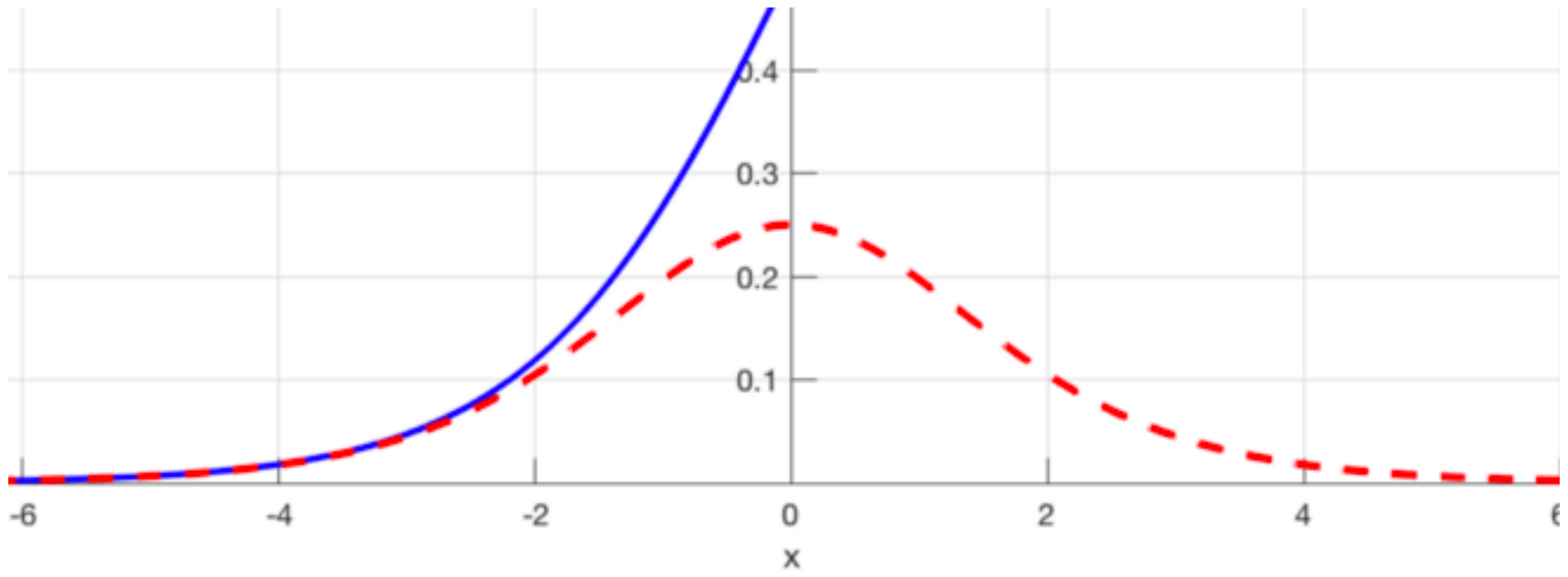
(a) Nottingham Dataset (b) MuseData Dataset

(Chung et al, 2014)

Question

What are the advantages of using LSTMs and GRUs?

Vanishing Gradients?

Recurrent Neural Networks	Long Short Term Memory
State maintained by hidden state feedback	State maintained by cell value
$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$	$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$
Gradient systemically squashed by sigmoid	Gradient set by value of forget gate
	$\frac{\partial c_t}{\partial c_{t-1}} = f_t$
	Can still vanish, but only if forget gate closes!

Question

What’s a disadvantage of using a LSTM or GRU?

Question

What's a disadvantage of using a LSTM or GRU?

More parameters!

$$f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right)$$
$$i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right)$$
$$o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right)$$
$$\tilde{c}_t = \phi\left(W_{cx}x_t + W_{ch}h_{t-1} + b_c\right)$$
$$c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$$
$$h_t = o_t \times \phi(c_t)$$

$$z_t = \sigma\left(W_{zh}h_t + b_z\right)$$
$$h_t = \sigma\left(W_{hx}x_t + W_{hh}h_{t-1} + b_h\right)$$

Question

Could there be better recurrent architectures than GRUs and LSTMs?

Optimal Architectures?

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	**0.92135**	**0.47483**	0.08968
MUT2	0.89735	**0.47324**	0.09036
MUT3	0.90728	0.46478	**0.09161**

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

(Jozefowicz et al, 2015)

Recap

- Recurrent neural networks can **theoretically** learn to model an **unbounded context length**
    - no increase in model size because weights are shared across time steps
- Practically, however, **vanishing gradients** stop vanilla RNNs from learning useful **long-range dependencies**
- LSTMs and GRUs are variants of recurrent networks that mitigate the vanishing gradient problem
    - used for for **many sequence-to-sequence tasks (up next!)**

References

- Elman, J.L. (1990). Finding Structure in Time. *Cogn. Sci., 14*, 179-211.
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*, 1735-1780.
- Cho, K., Merrienboer, B.V., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. *Conference on Empirical Methods in Natural Language Processing.*
- Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., & Schmidhuber, J. (2015). LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems, 28*, 2222-2232.

Question

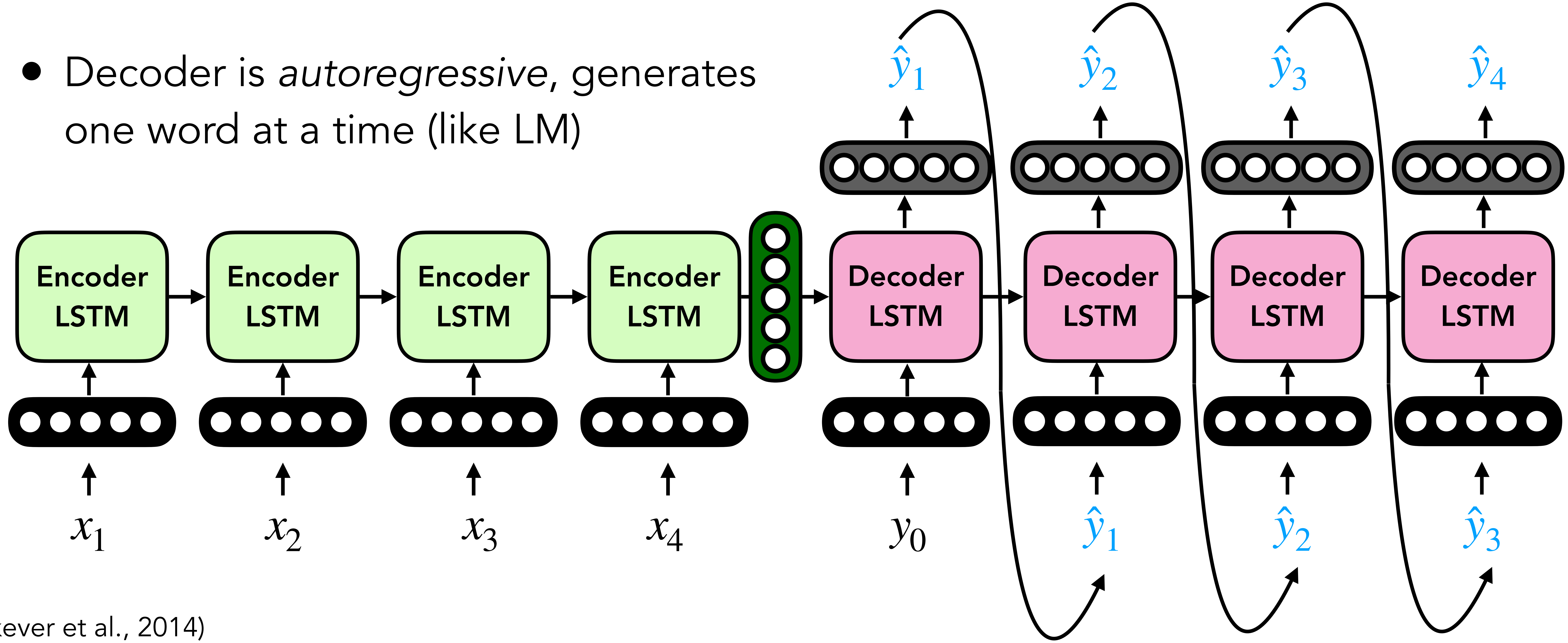
What could we do if the sequence we're encoding and the sequence we want to generate have different properties?

Example: Machine Translation

Encoder-Decoder Models

- Encode a sequence fully with one model (**encoder**) and use its representation to seed a second model that decodes another sequence (**decoder**)

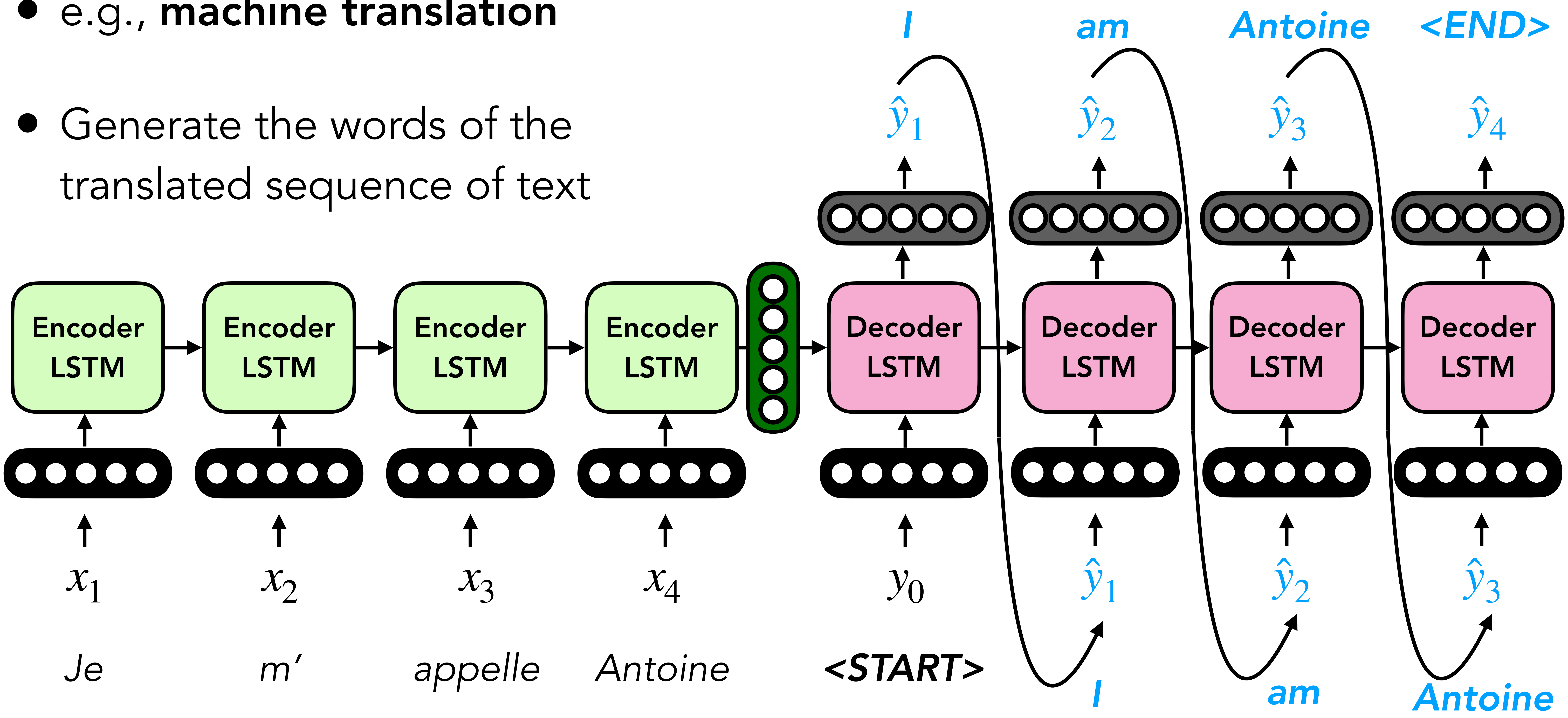
- Decoder is *autoregressive*, generates one word at a time (like LM)



(Sutskever et al., 2014)

Encoder-Decoder Models

- e.g., **machine translation**
- Generate the words of the translated sequence of text



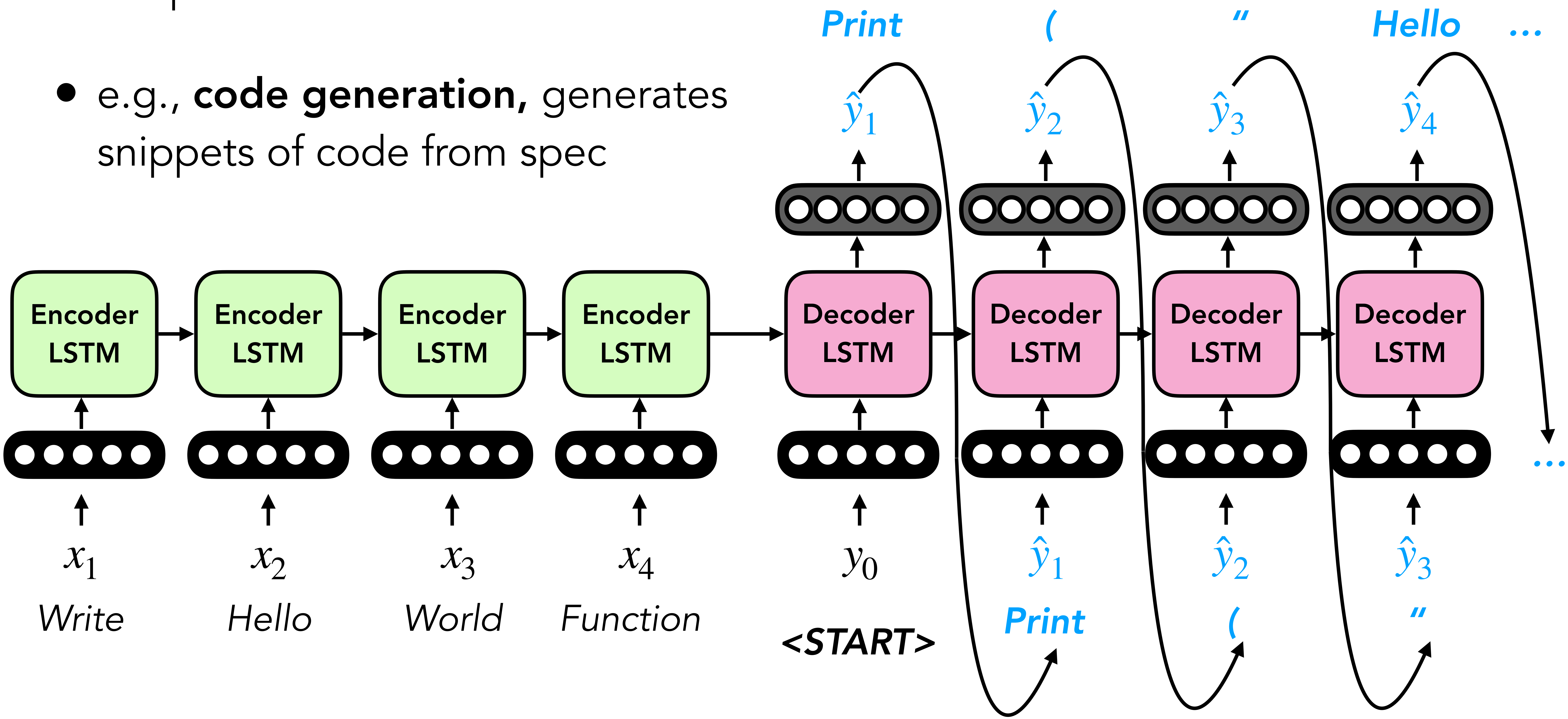
(Sutskever et al., 2014)

Question

What other tasks might have this property ?

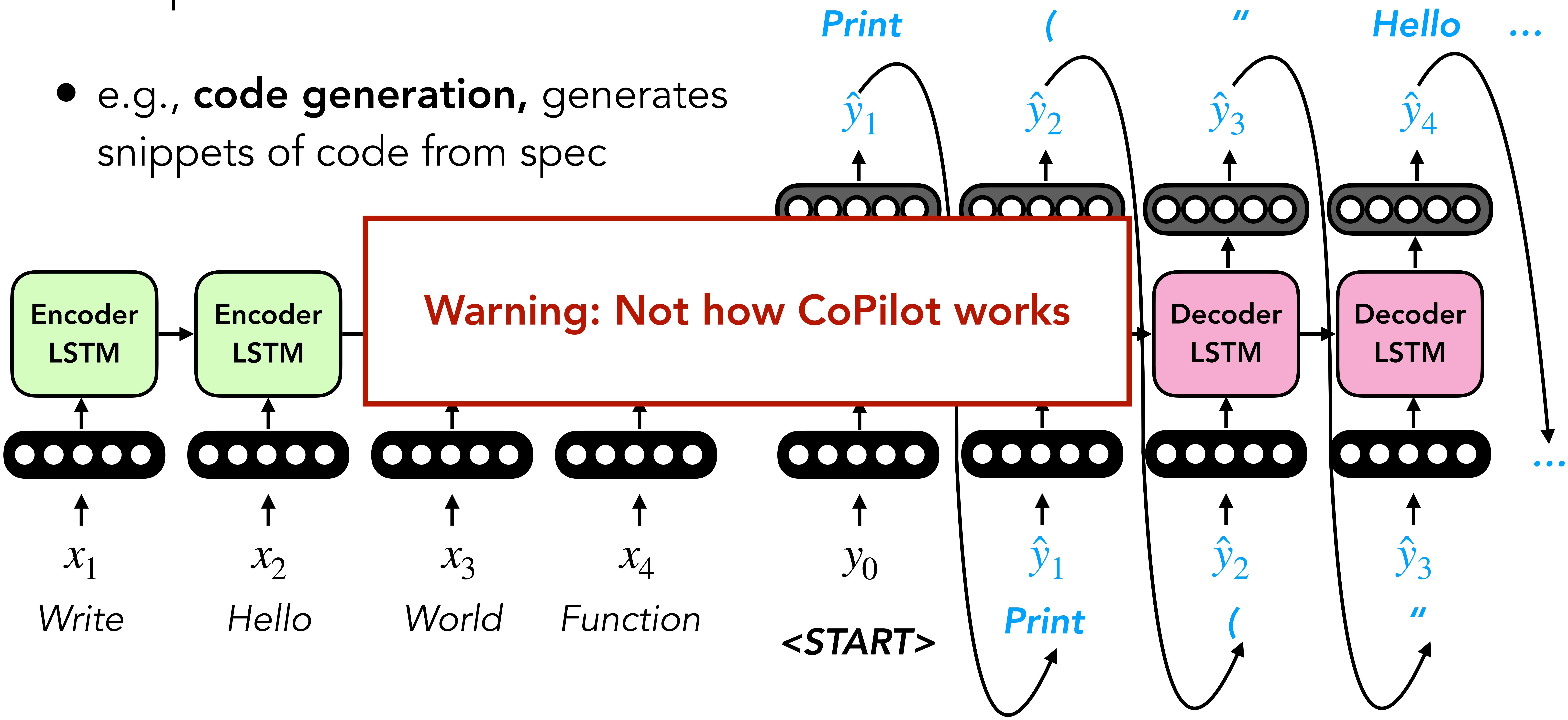
Encoder-Decoder Models

- Output can be other forms of text
- e.g., **code generation,** generates snippets of code from spec



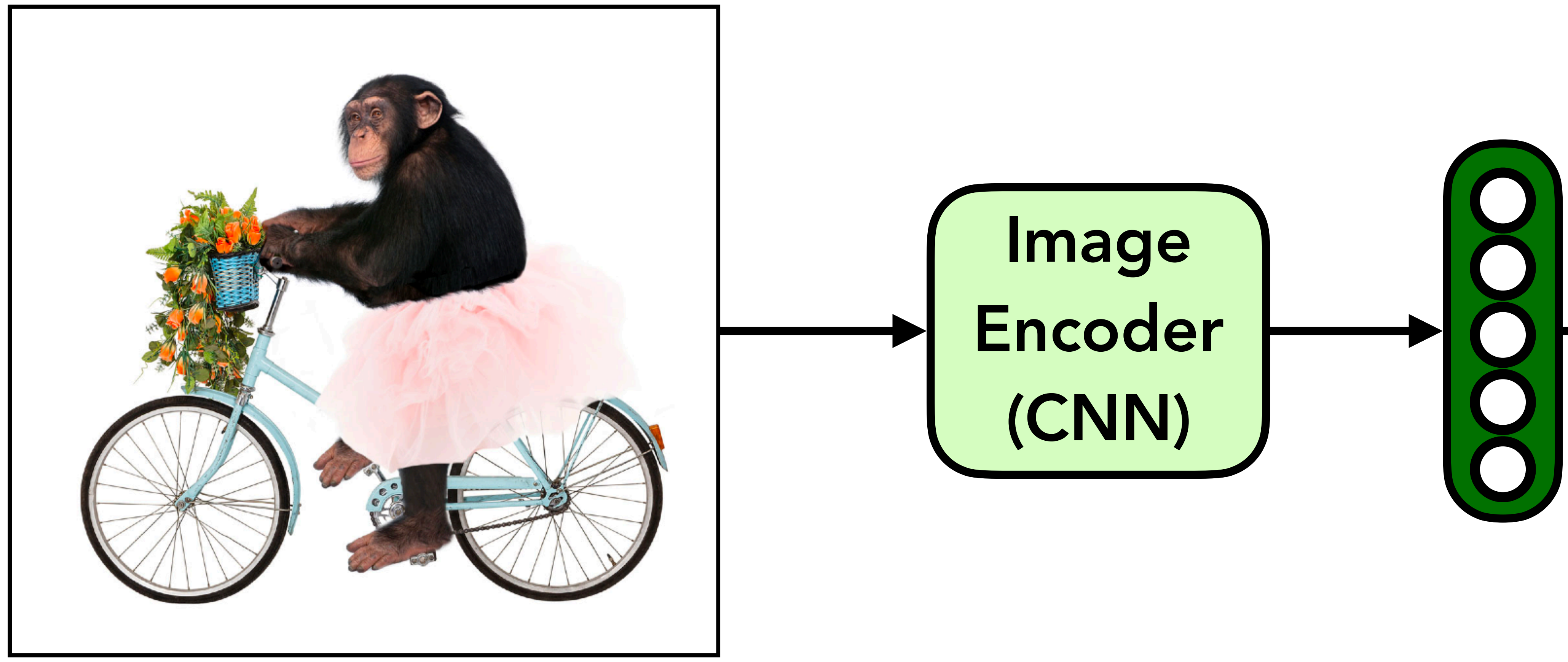
Encoder-Decoder Models

- Output can be other forms of text
- e.g., **code generation,** generates snippets of code from spec



Encoder-Decoder Models

- Input doesn't need to be text
- e.g., **image captioning**



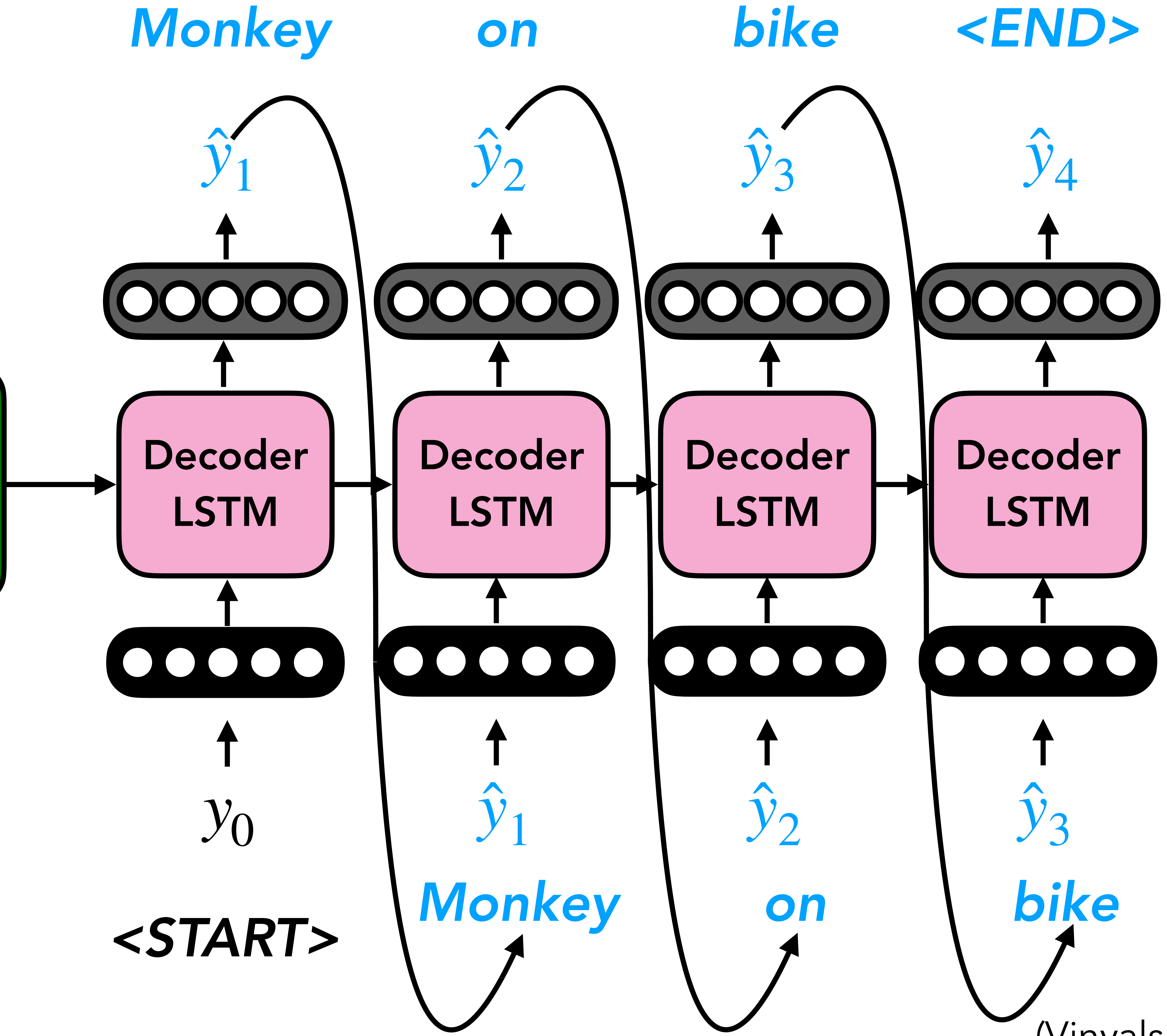


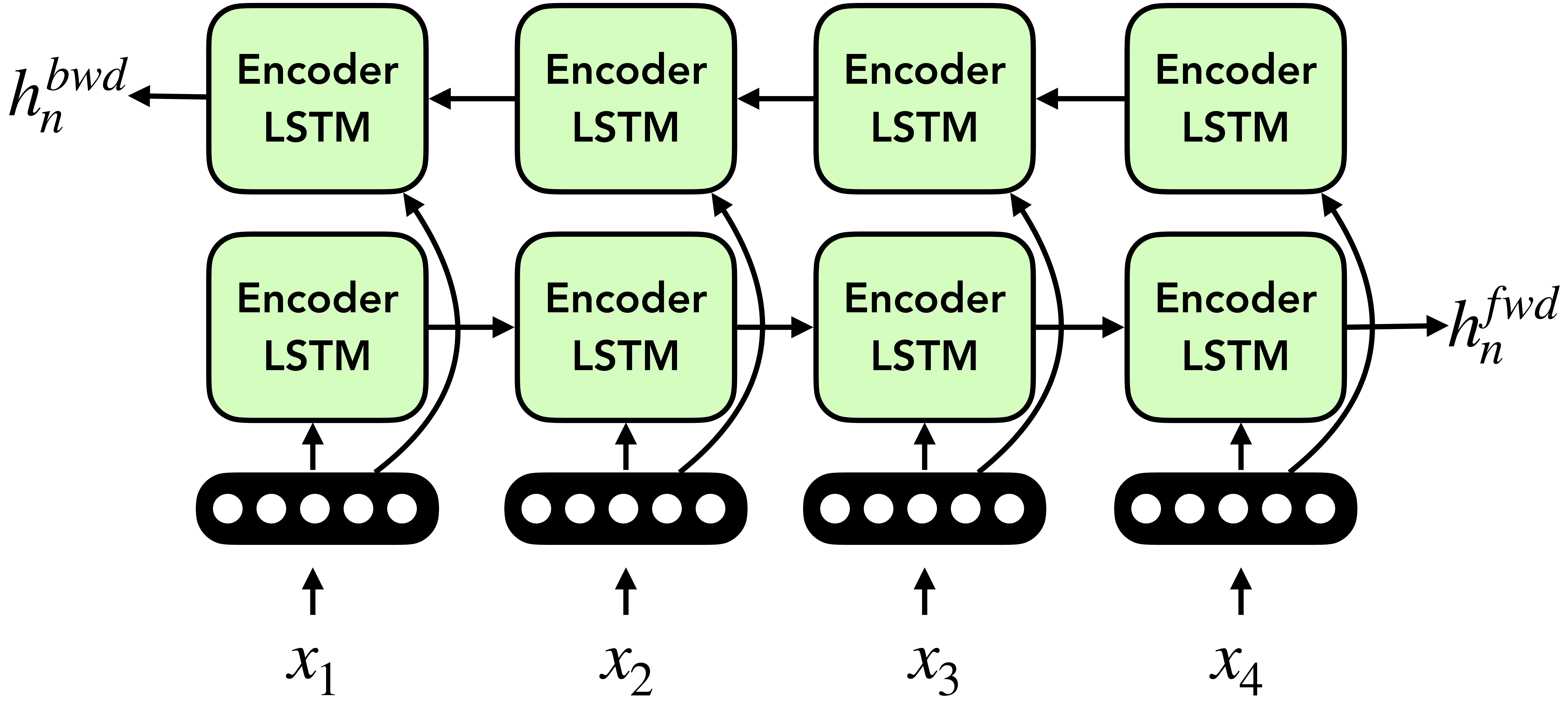
Photo credit: J Hovenstine Studios

- Generate words of image description

(Vinyals et al., 2014)

Bidirectional Encoders

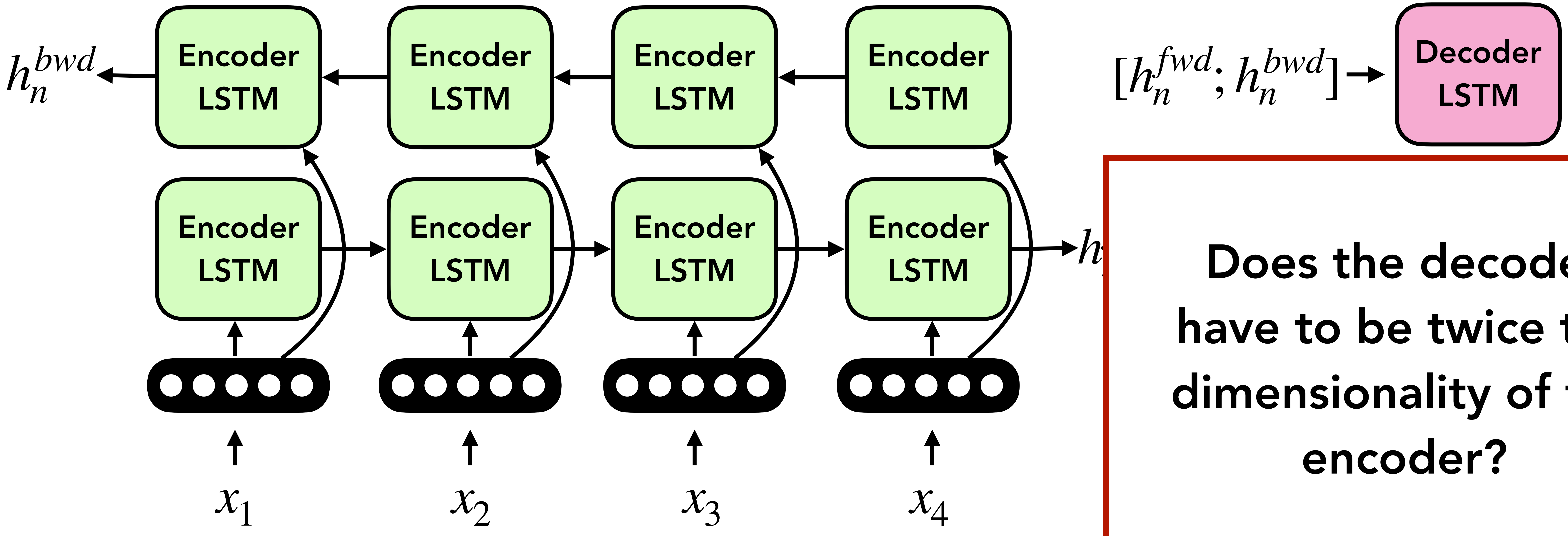
- Decoder needs to be unidirectional (autoregressive models can't know the future…)
- Encoder sequence representation augmented by encoding in both directions



(Schuster and Paliwal, 1997)

Bidirectional Encoders

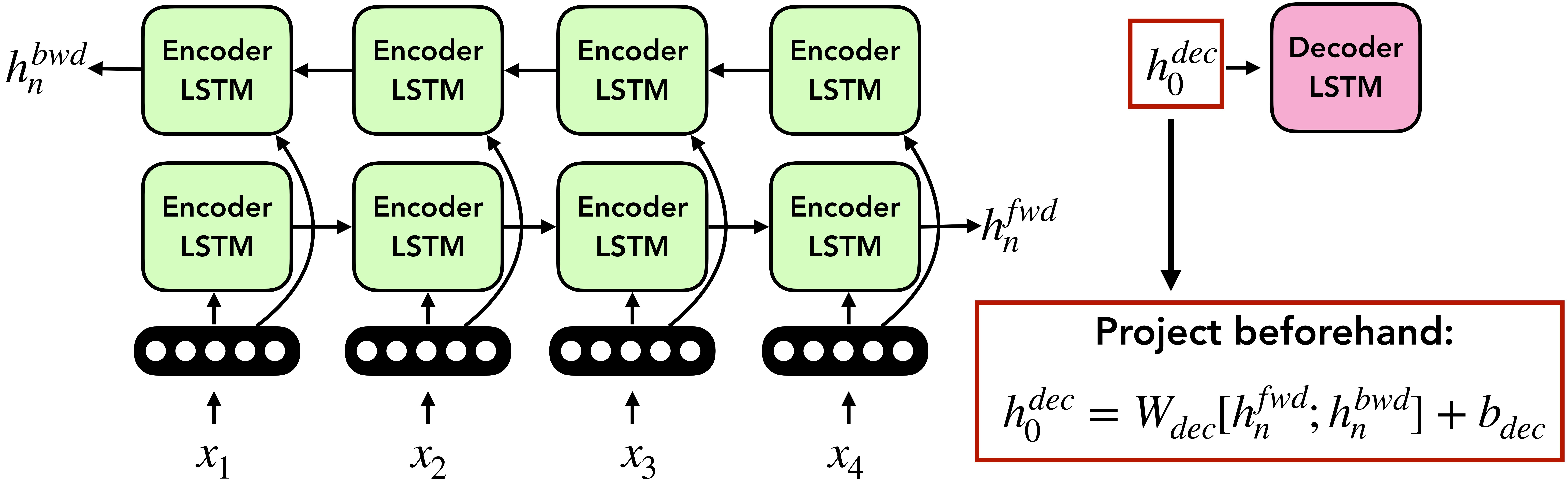
- Decoder needs to be unidirectional (autoregressive models can't know the future…)
- Encoder sequence representation augmented by encoding in both directions



Does the decoder have to be twice the dimensionality of the encoder?

Bidirectional Encoders

- Decoder needs to be unidirectional (autoregressive models can't know the future…)
- Encoder sequence representation augmented by encoding in both directions



Training Encoder-Decoder Models

- With a language model, we had practically unlimited data!
  - We were only learning which words followed others, so any text would do!

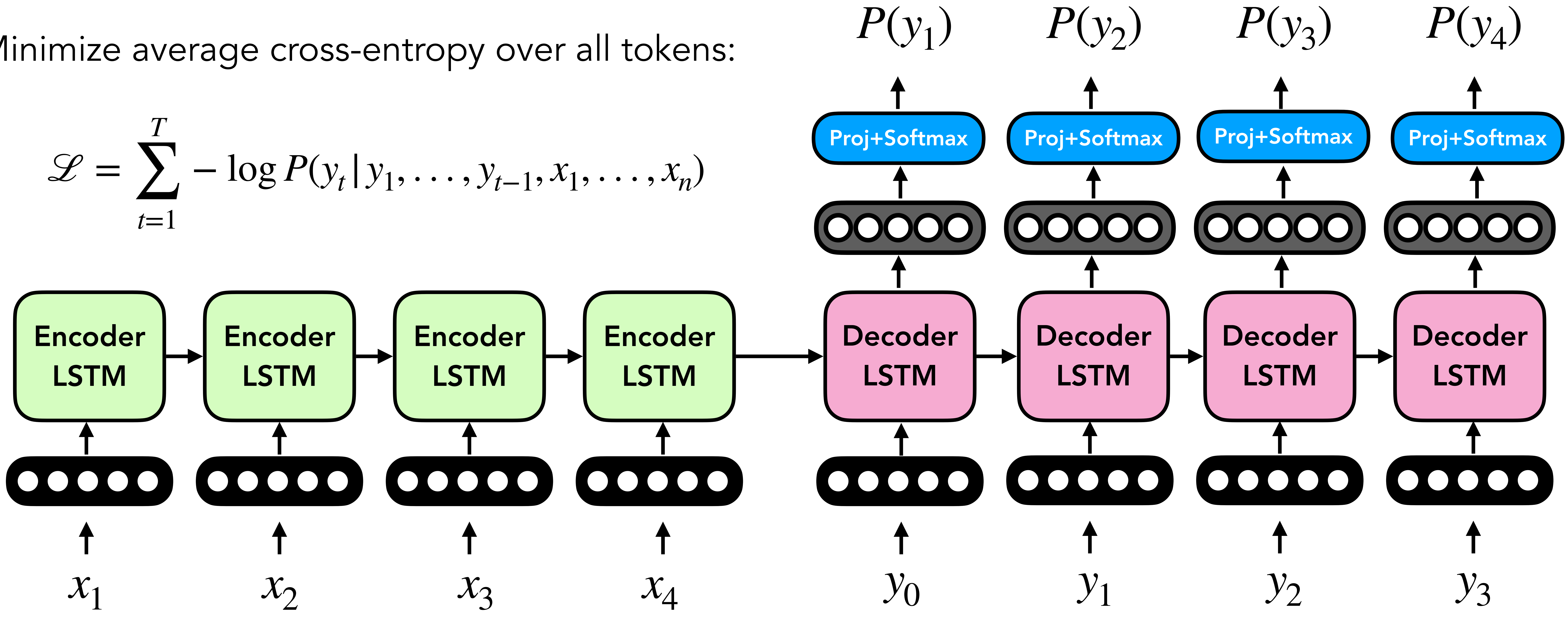
- With encoder-decoder models, we are learning which sequences align with others
  - **Machine Translation**: Need paired data across languages (sentences that have the same meaning)
  - **Image Captioning**: Need paired image-text data (images and their description)
  - **Code Generation**: Need paired code-text data (e.g., code and their comments)
  - And so on… for other tasks!

Training Encoder-Decoder Models

Similar to training a language model!

Minimize average cross-entropy over all tokens:

$$\mathscr{L} = \sum_{t=1}^{T} -\log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$



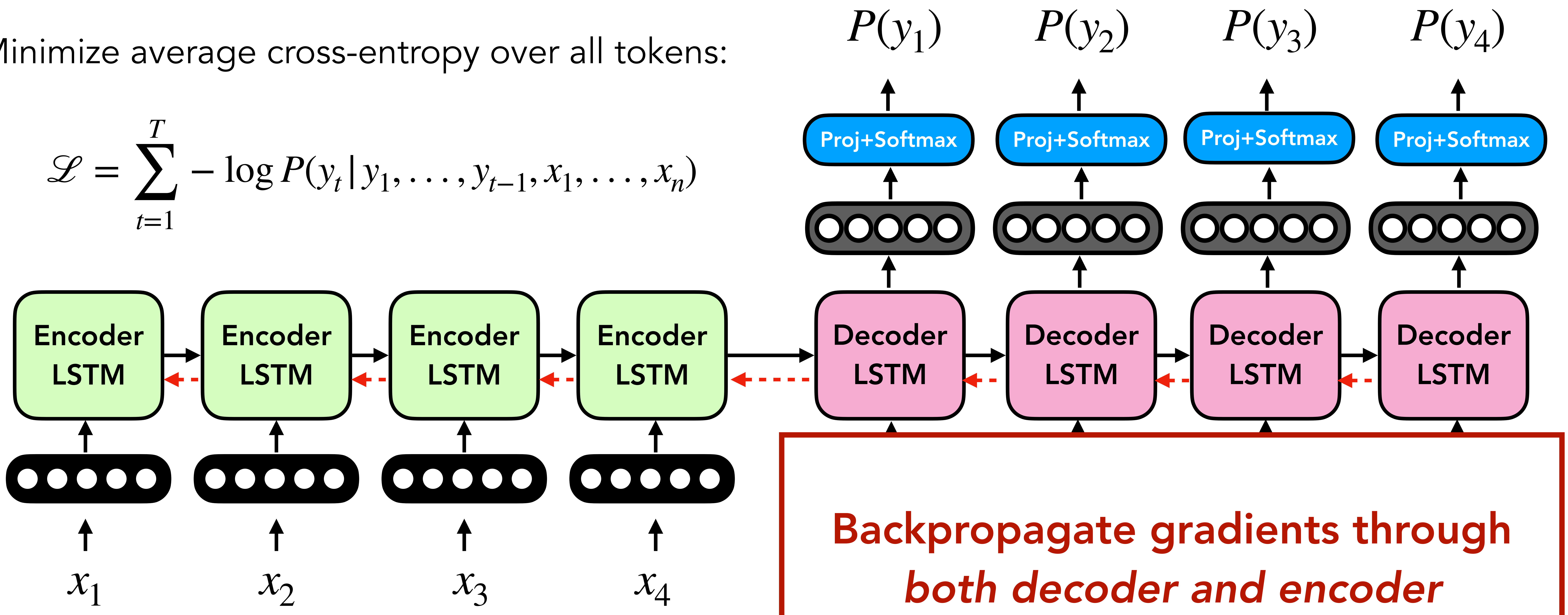
Note: Change in notation — Encoder inputs are now subscripted by *n* and decoder outputs subscripted by *t*

Training Encoder-Decoder Models

Similar to training a language model!

Minimize average cross-entropy over all tokens:

$$\mathscr{L} = \sum_{t=1}^{T} -\log P(y_t \mid y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)$$



Backpropagate gradients through *both decoder and encoder*

Training Encoder-Decoder Models

- With a language model, we had practically unlimited data!
  - We were only learning which words followed others, so any text would do!
- With encoder-d quences align with others
  - **Machine Translat** t have the same meaning)
  - **Image Captioning**: Need paired image-text data (images and their description)
  - **Code Generation**: Need paired code-text data (e.g., code and their comments)
  - And so on… for other tasks!

Warning: Paired data can be much more challenging to find in the wild

“you can’t cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!”

— Ray Mooney (NLP professor at UT Austin)

Issue with Recurrent Models

- State represented as a single vector —> massive compression of information
- At every step, it must be re-computed, making it challenging to learn long-range dependencies without ignoring immediate ones

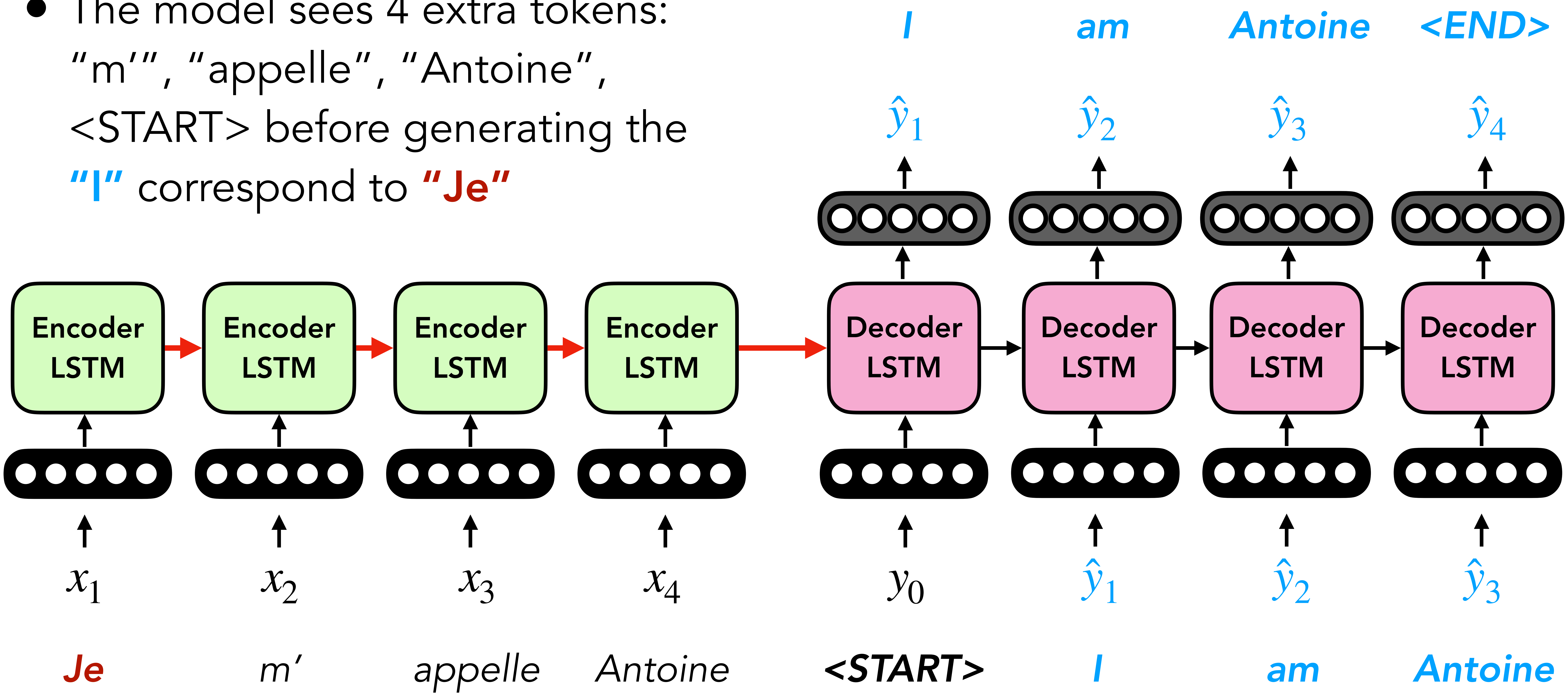
They tuned, discussed for a moment, then struck up a lively ***jig****. Everyone joined in, turning the courtyard into an even more chaotic scene, people now* ***dancing*** *in circles,* ***swinging*** *and* ***spinning*** *in circles, everyone making up their own* ***dance steps****. I felt my feet tapping, my body wanting to move. Aside from writing, I 've always loved* **dancing** *.*

- Nearby words should affect each other more than farther ones, but RNNs make it challenging to learn **any** long-range interactions

LAMBADA dataset, 2016

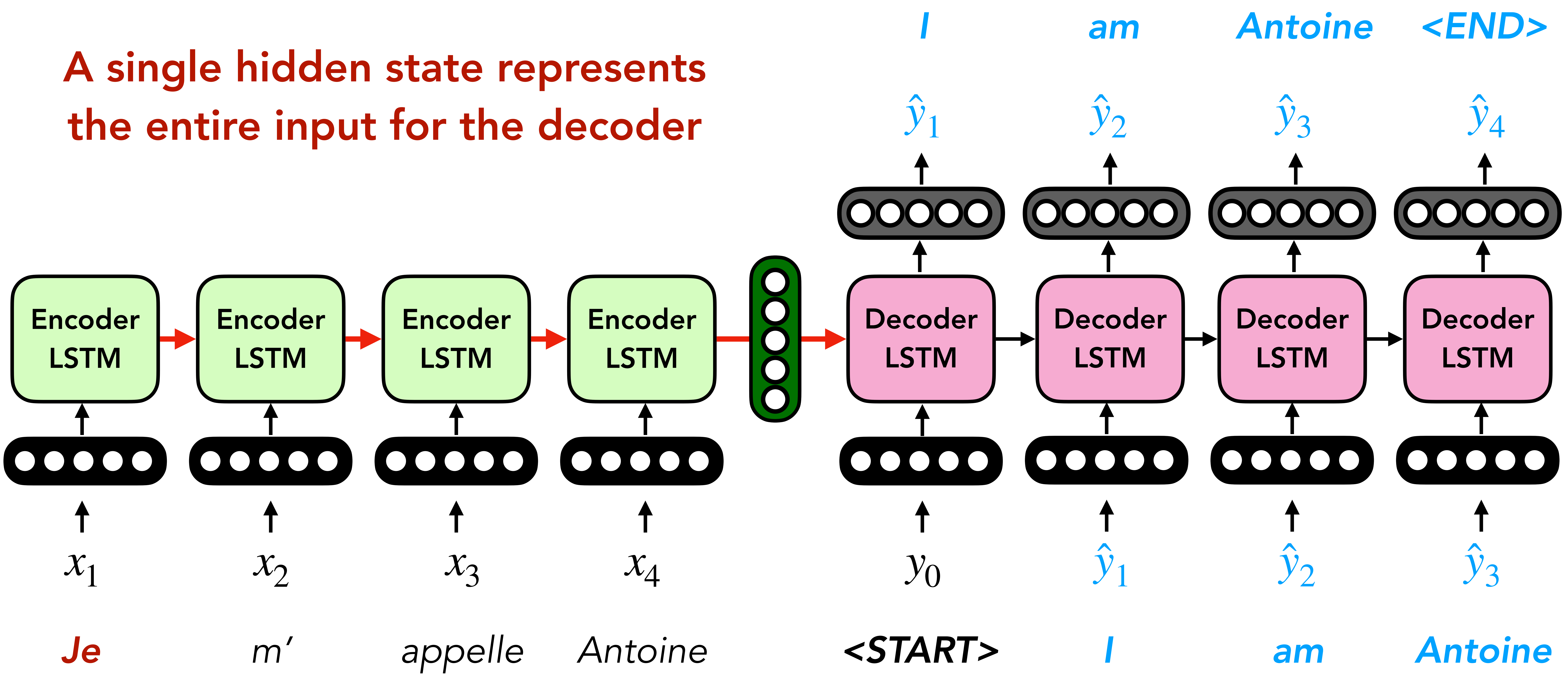
Toy Example

- The model sees 4 extra tokens: "m'", "appelle", "Antoine", <START> before generating the **"I"** correspond to **"Je"**



(Sutskever et al., 2014)

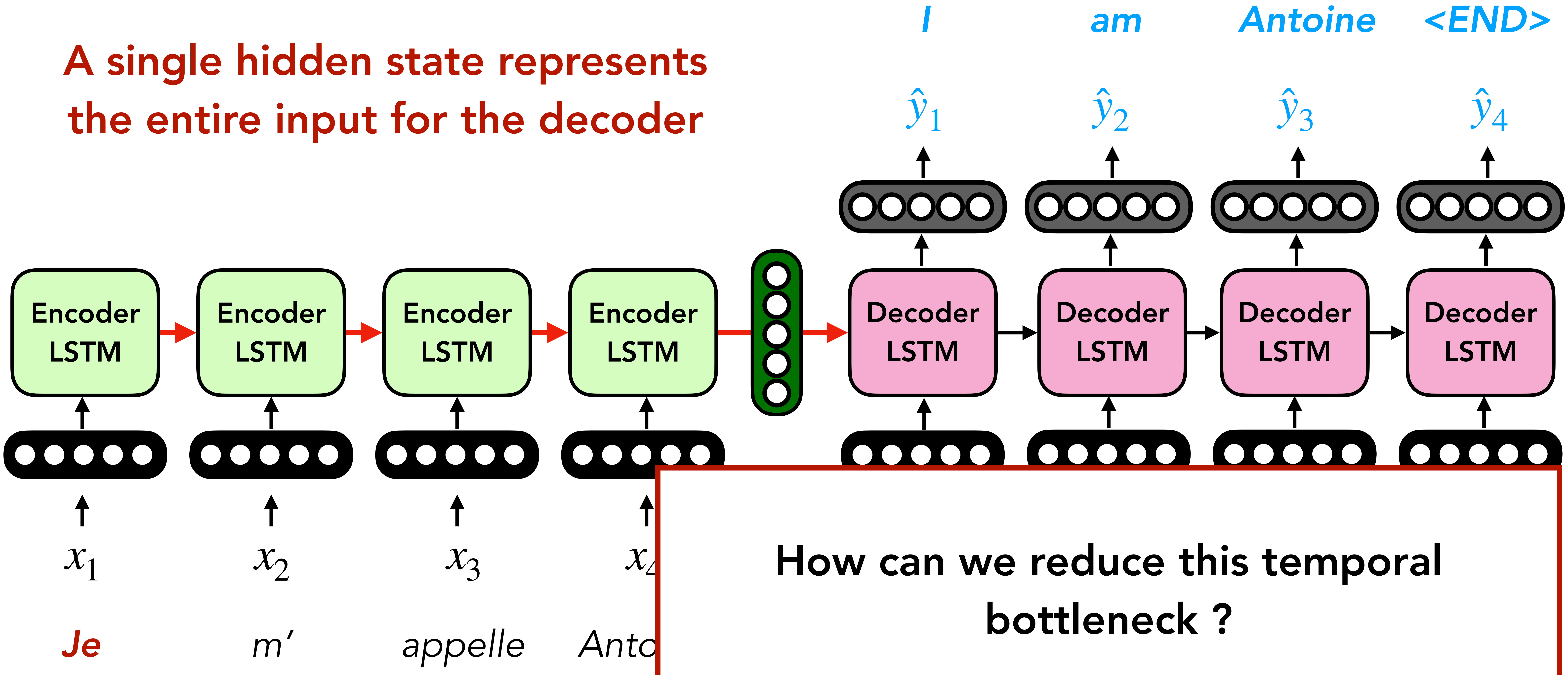
Toy Example



(Sutskever et al., 2014)

Toy Example

A single hidden state represents the entire input for the decoder



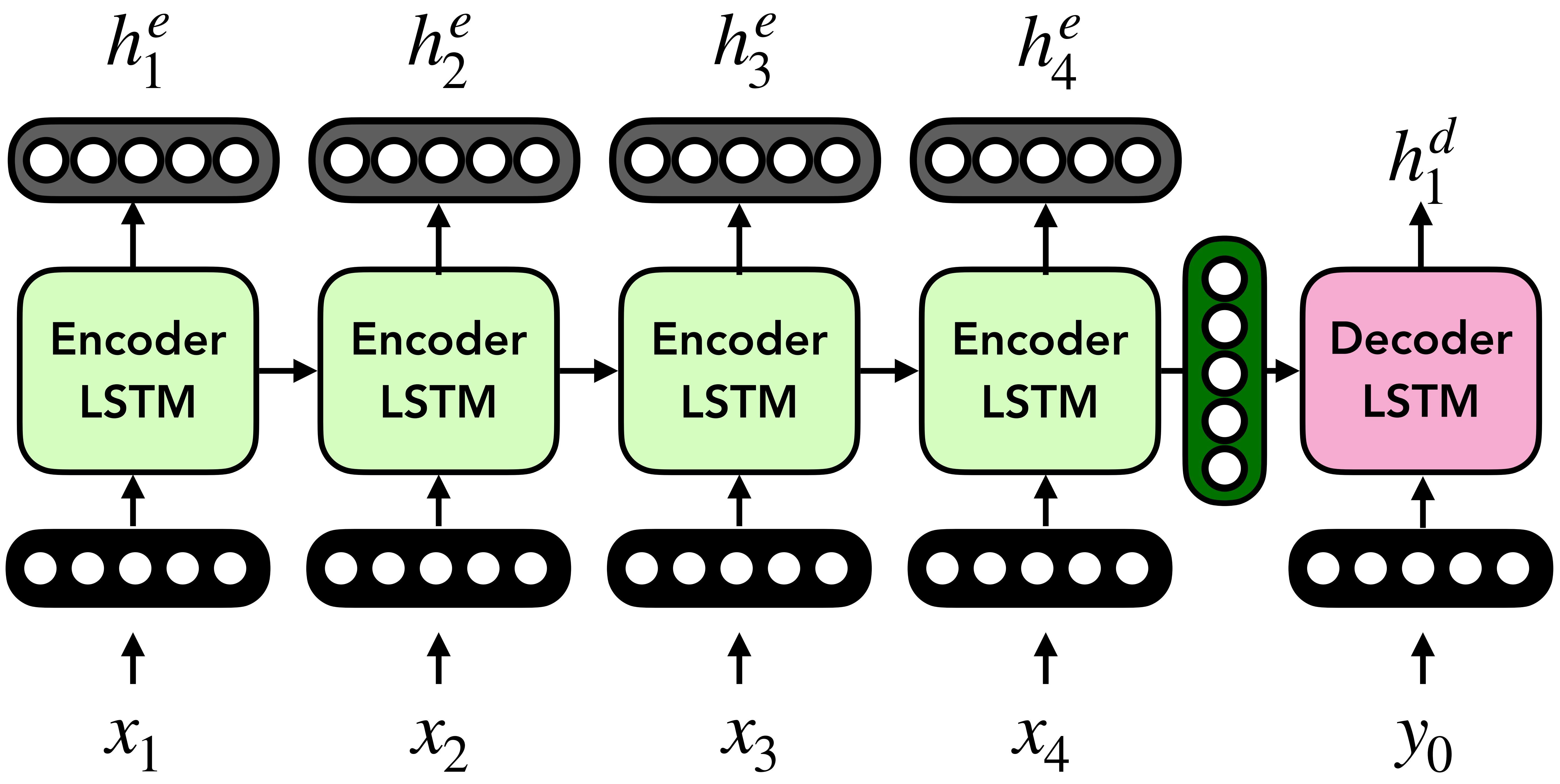
How can we reduce this temporal bottleneck ?

(Sutskever et al., 2014)

Solution

Attention!

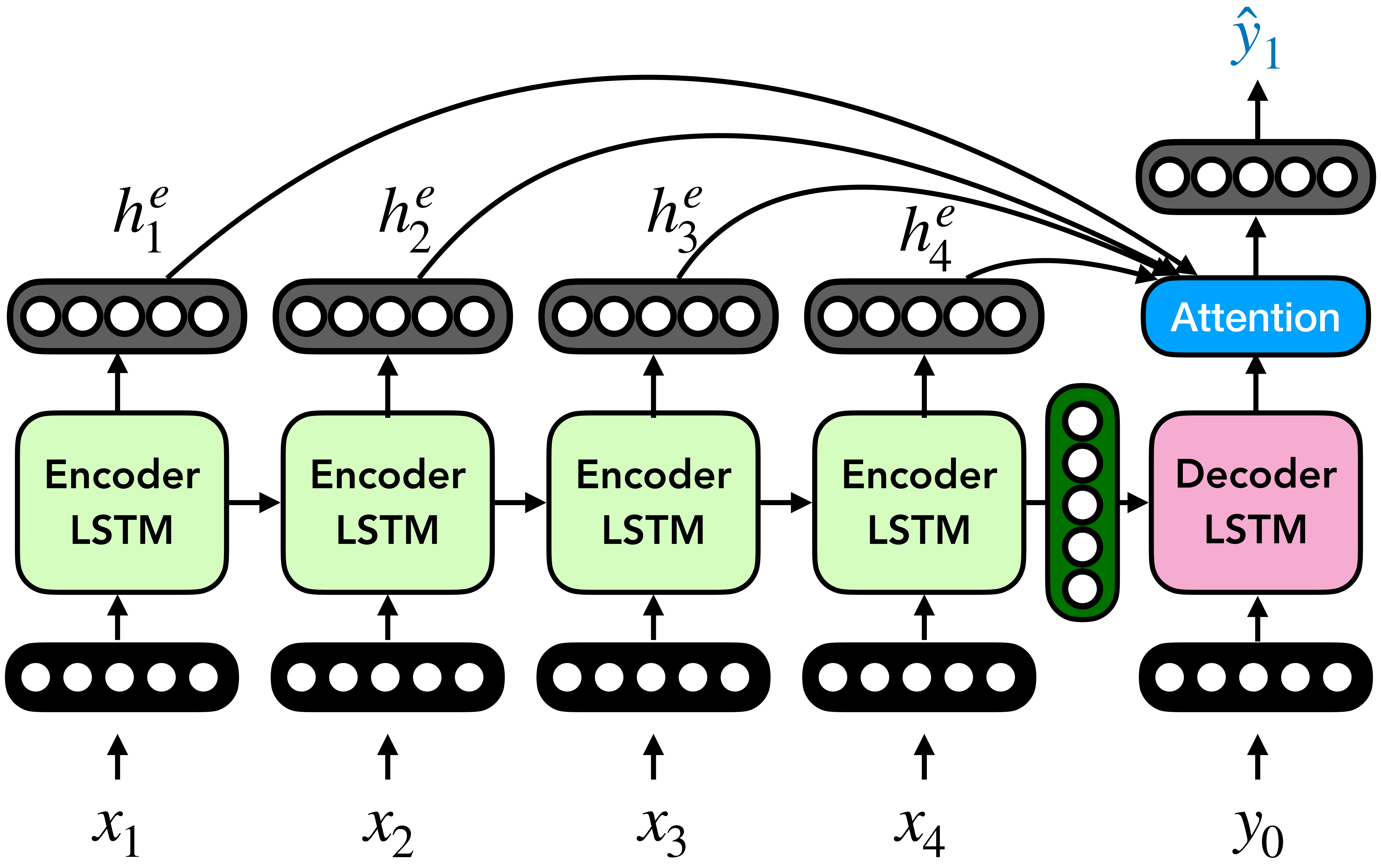
Attentive Encoder-Decoder Models



- **Recall:** At each encoder time step, there is an output of the RNN!

(Bahdanau et al., 2015)

Attentive Encoder-Decoder Models



- **Recall:** At each encoder time step, there is an output h_n^e of the RNN!
- **Idea:** Use the output of the Decoder LSTM to compute an **attention** (i.e., a mixture) over all the h_n^e outputs of the encoder LSTM
- **Intuition:** focus on different parts of the input at each time step

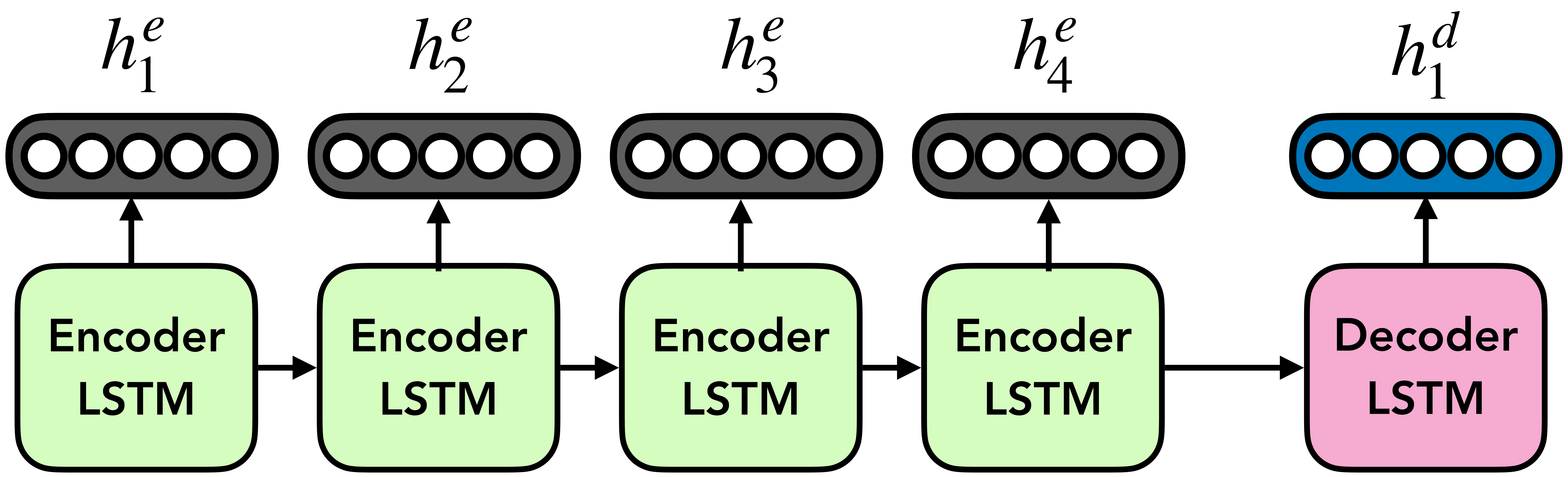
(Bahdanau et al., 2015)

What is attention?

- Attention is a **weighted average over a set of inputs**

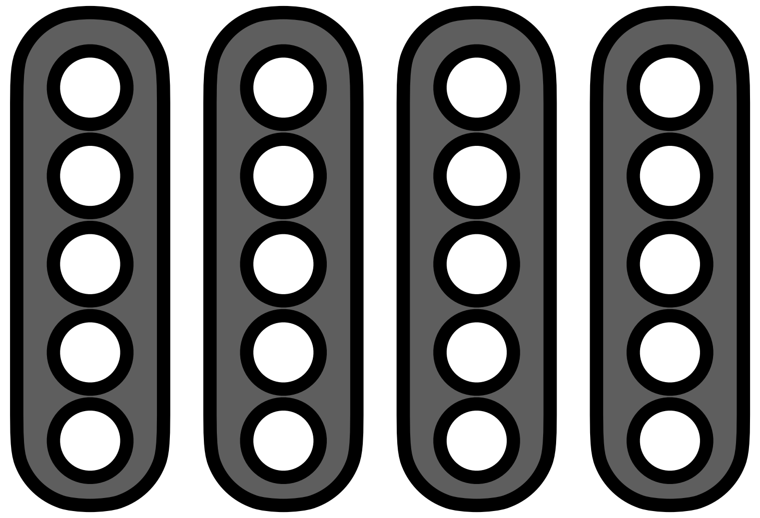
h_n^e = encoder output hidden states

- How should we compute this weighted average?



Attention Function

- **Compute** pairwise similarity between each encoder hidden state and decoder hidden state ("idea of what to decode")



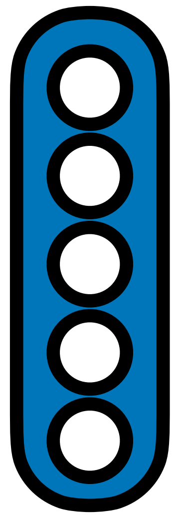
h_1^e h_2^e h_3^e h_4^e

h_n^e = encoder output hidden states

Also known as a "keys"

h_t^d = decoder output hidden state

Also known as a "query"



Attention Function

- **Compute** pairwise similarity between each encoder hidden state and decoder hidden state ("idea of what to decode")

h_n^e = encoder output hidden states

Also known as a "keys"

h_t^d = decoder output hidden state

Also known as a "query"

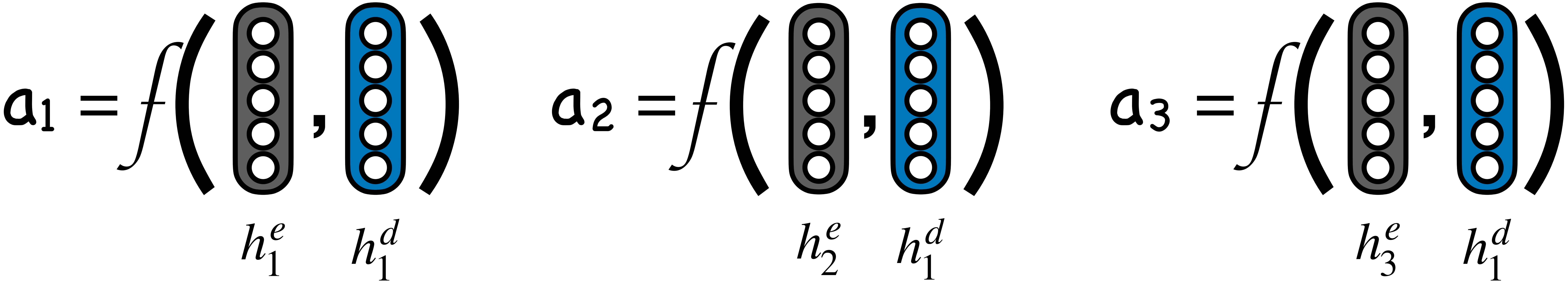
- **We have a single query vector for multiple key vectors**

Attention Function

Attention Function	Formula
Multiplicative	$a = h^e \mathbf{W} h^d$
Linear	$a = v^T \phi(\mathbf{W}[h^e; h^d])$
Scaled Dot Product	$a = \frac{(\mathbf{W}h^e)^T(\mathbf{U}h^d)}{\sqrt{d}}$

Attention Function

- **Compute** pairwise similarity between each encoder hidden state and decoder hidden state ("idea of what to decode")

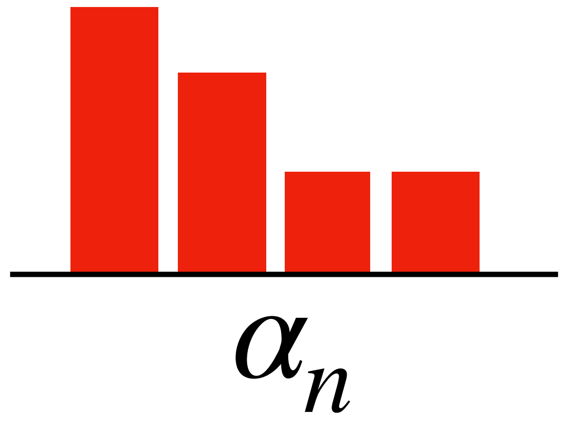


- **Convert** pairwise similarity scores to probability **distribution** (using softmax!) over encoder hidden states and compute weighted average:

Softmax!

$$\alpha_n = \frac{e^{a_n}}{\sum_j e^{a_j}}$$

→

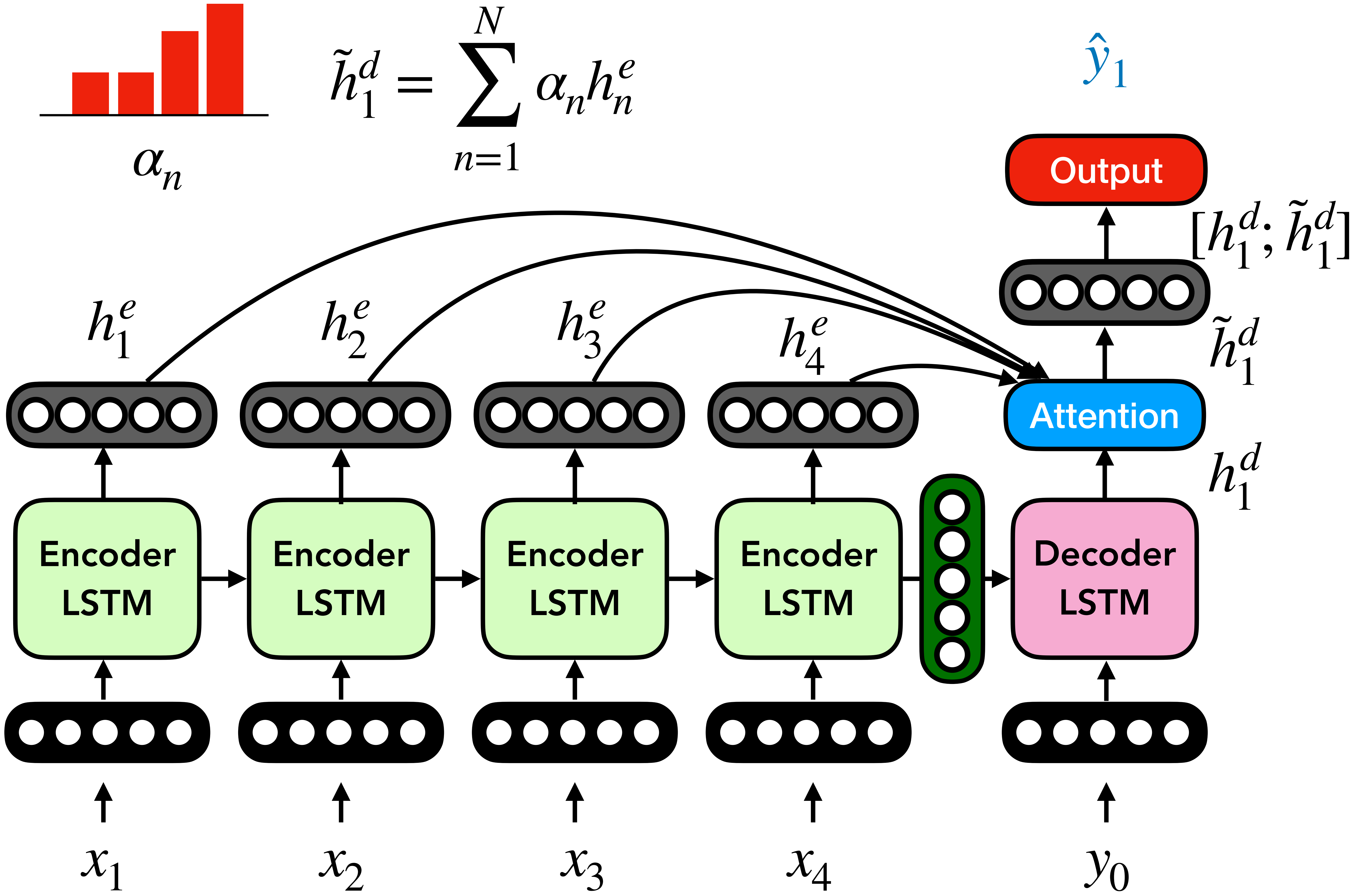


→

$$\tilde{h}_1^d = \sum_{n=1}^{N} \alpha_n h_n^e$$

Here h_n^e is known as the "value"

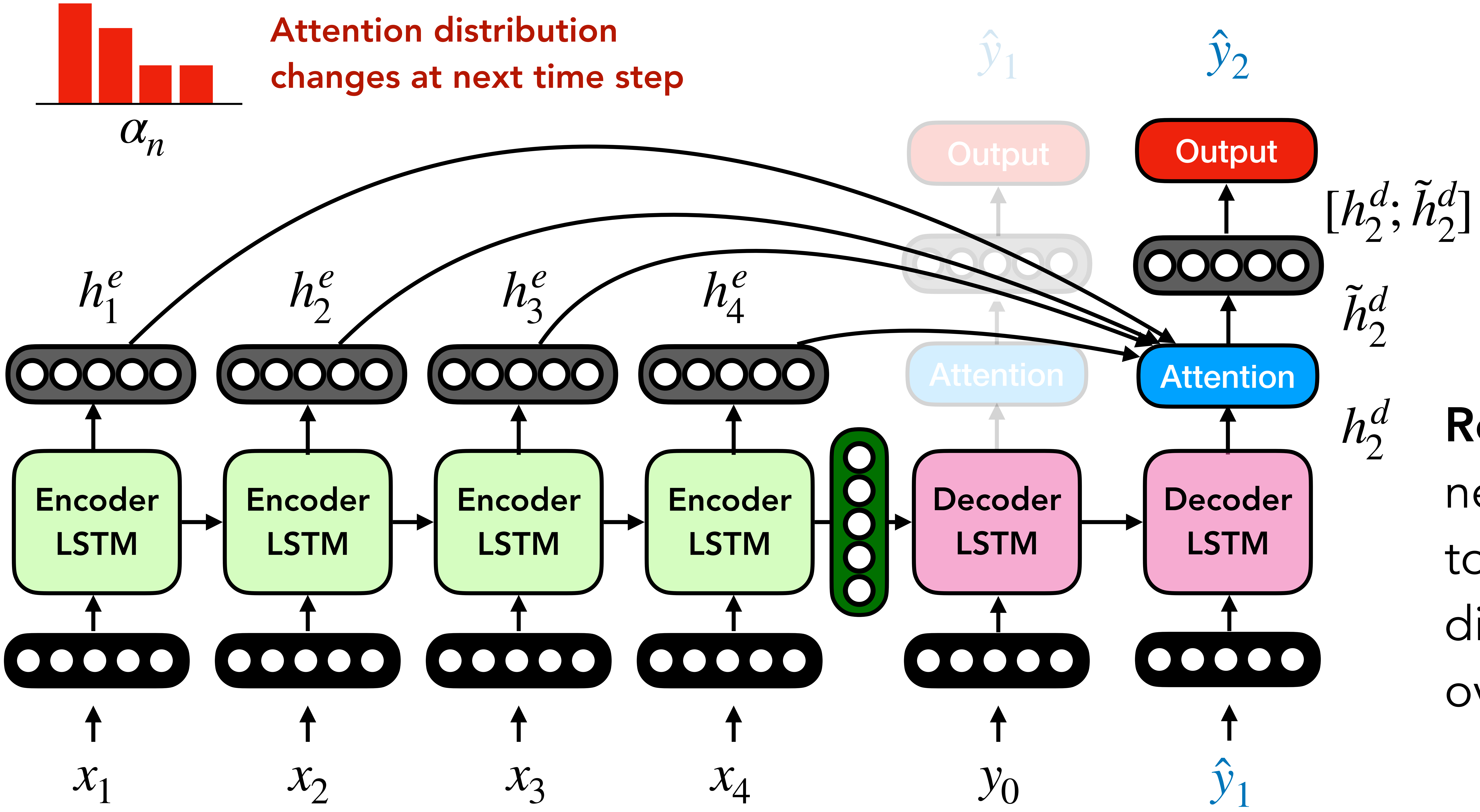
Attentive Encoder-Decoder Models



- **Intuition:** $\tilde{h}_1^d$ contains information about encoder hidden states that got **high** attention
- Typically, $\tilde{h}_1^d$ is concatenated (or composed in some other manner) with h_1^d (the original decoder state) before being passed to the **output** layer
- **Output** layer predicts the most likely output token $\hat{y}_1$

(Bahdanau et al., 2015)

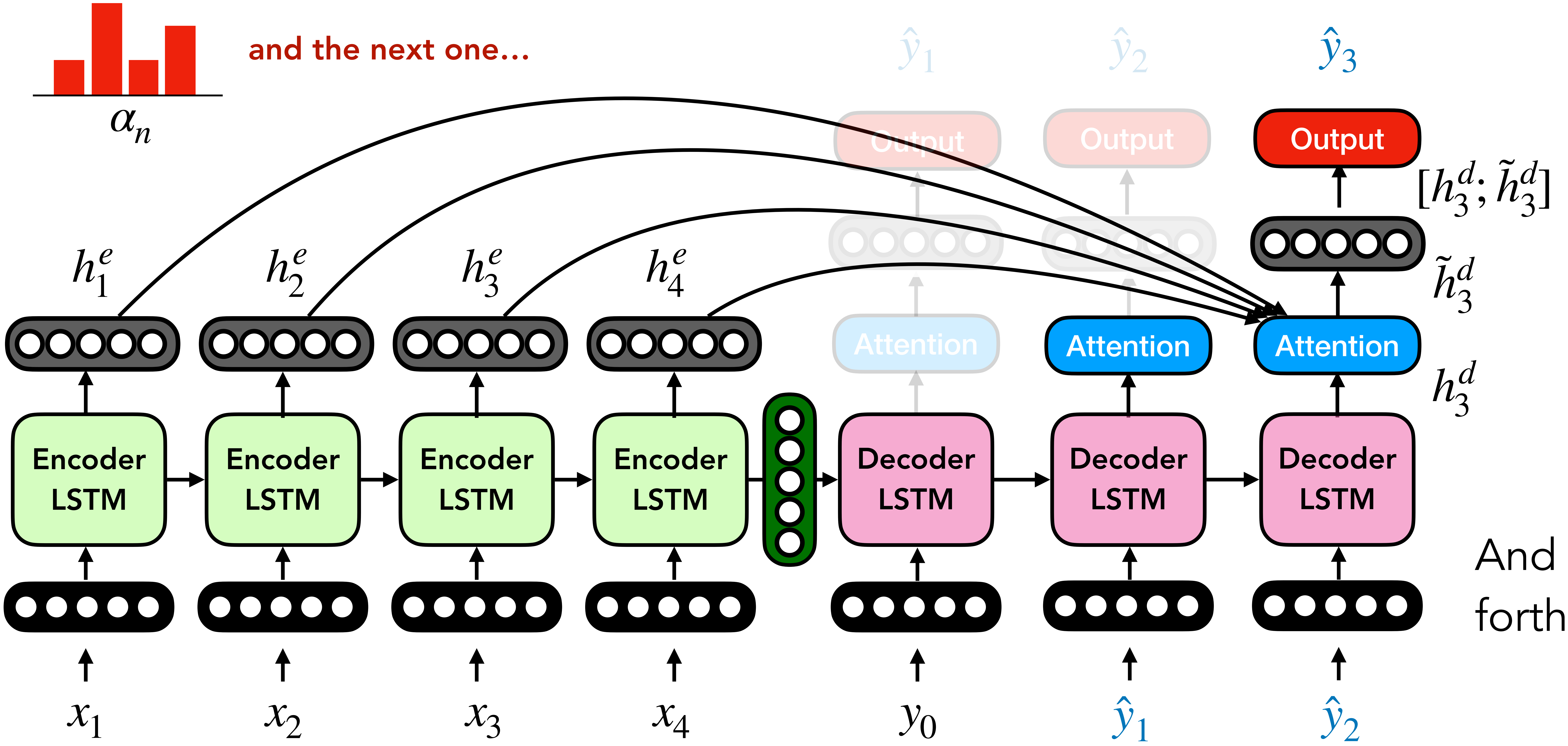
Attentive Encoder-Decoder Models



Repeat in next time step to get new distribution over states

(Bahdanau et al., 2015)

Attentive Encoder-Decoder Models

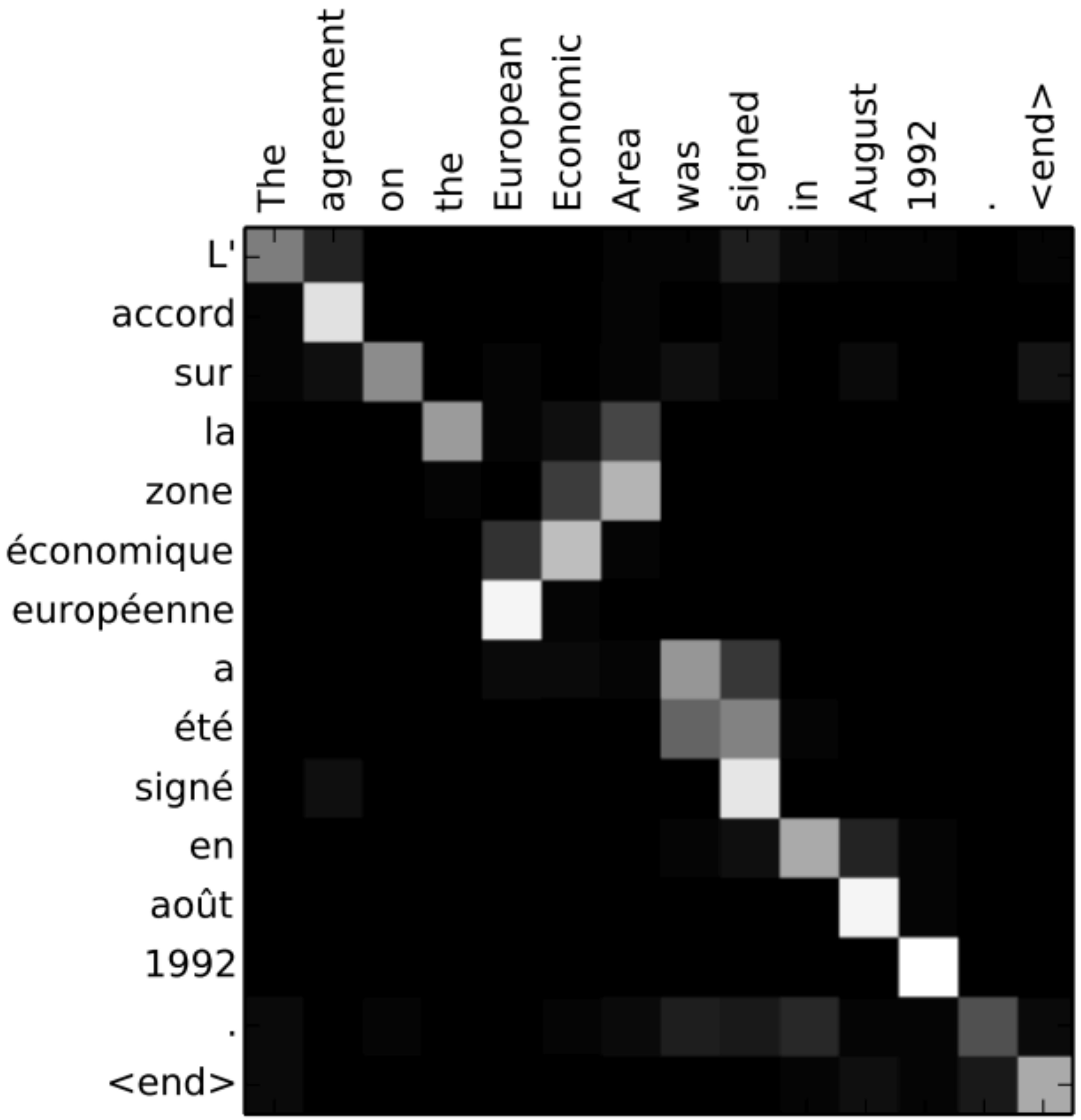


And so forth…

(Bahdanau et al., 2015)

Interpretability?

- **Main Idea:** Attention can be visualised based on the score given to each encoder hidden state
- What is focused on when each word is generated?
- Training with attention gives us implicit alignment for free!



Question

How does attention address the temporal bottleneck in sequence to sequence models?

Direct connections between decoder states and encoder hidden states

Attention Recap

- **Main Idea:** Decoder computes a weighted sum of encoder outputs
  - Compute pairwise score between each encoder hidden state and initial decoder hidden state ("idea of what to decode")
- Many possible functions for computing scores (dot product, bilinear, etc.)
- **Temporal Bottleneck Fixed! Direct link** between decoder and encoder states
  - Helps with vanishing gradients!
- **Interpretability** allows us to investigate model behavior!
- Attention is **agnostic** to the type of RNN used in the encoder and decoder!

Question

In what range can an attention weight fall ?

[0, 1]

References

- Sutskever, I., Vinyals, O., & Le, Q.V. (2014). Sequence to Sequence Learning with Neural Networks. *NIPS*.
- Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2014). Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3156-3164.
- Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q.N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., & Fernández, R. (2016). The LAMBADA dataset: Word prediction requiring a broad discourse context. *ArXiv, abs/1606.06031*.
- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR, abs/1409.0473*.