CS60050 Machine learning

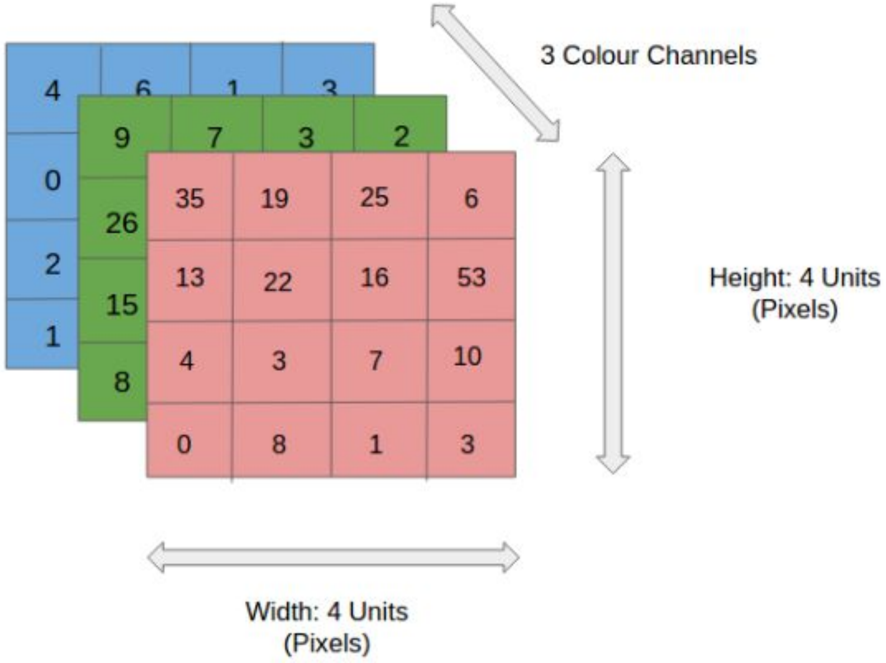
Neural Network Architectures
LSTM, Attention

Sudeshna Sarkar
Somak Aditya

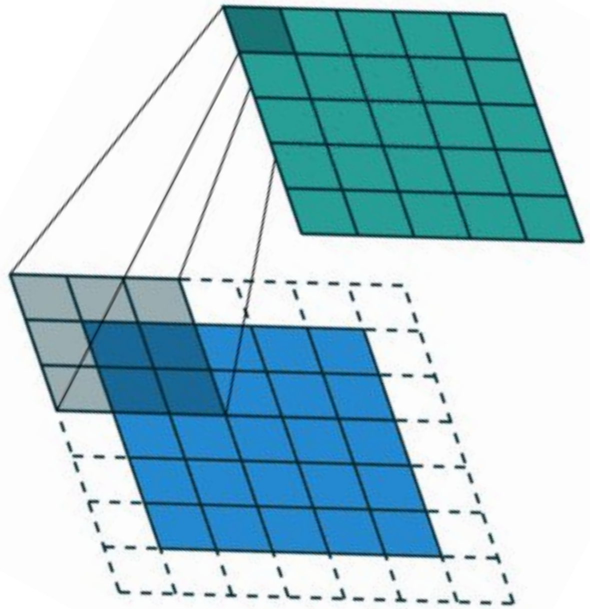
13 November 2024

- Slide Courtesy: Prof. Chris Manning, Prof. Li Fei Fei, Prof. Yunzhu Li, Prof. Ruohan Gao (Stanford Univ)

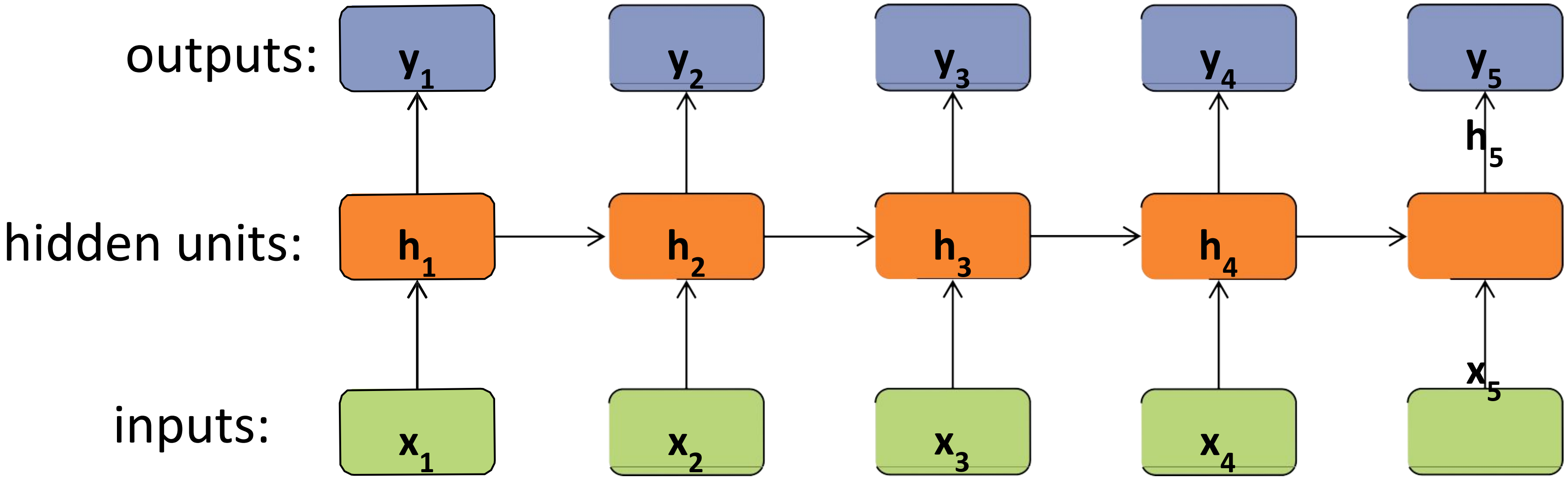
CNN and Convolution



Kernel Size: 3x3x1
Stride: 1
Zero-Padding



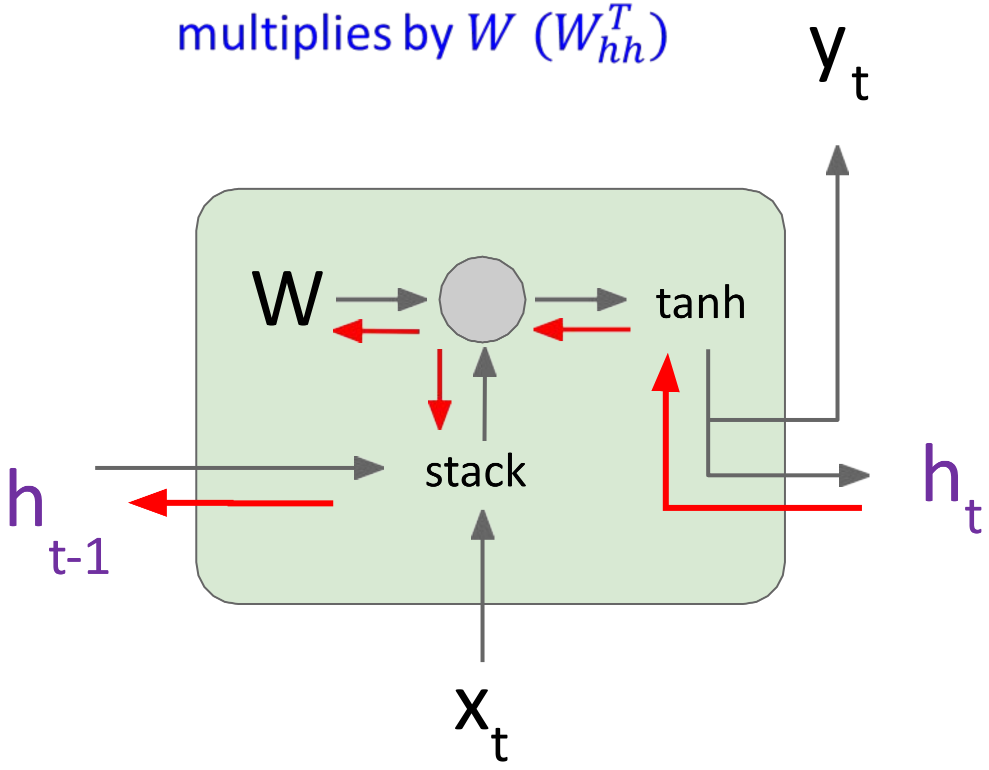
Recurrent Neural Networks (RNNs)



$$h_t = g\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$
$$y_t = W_{hy} h_t + b_y$$

Vanilla RNN Gradient Flow

Backpropagation from h_t to h_{t-1} multiplies by W (W_{hh}^T)



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

RNN 2.0: Long short-term Memory (LSTM)

- Hidden State: holds previous information (Short-term memory)
- Cell State: memory of the network (Long-term memory)

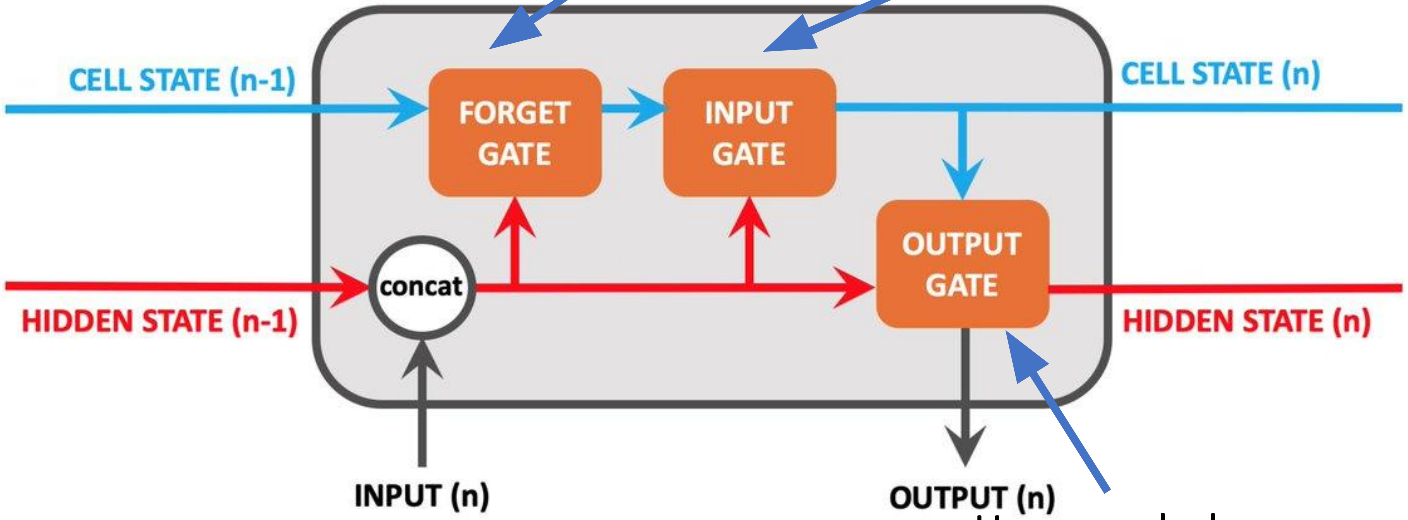
What should be forget in cell state

How much new data should be taken

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

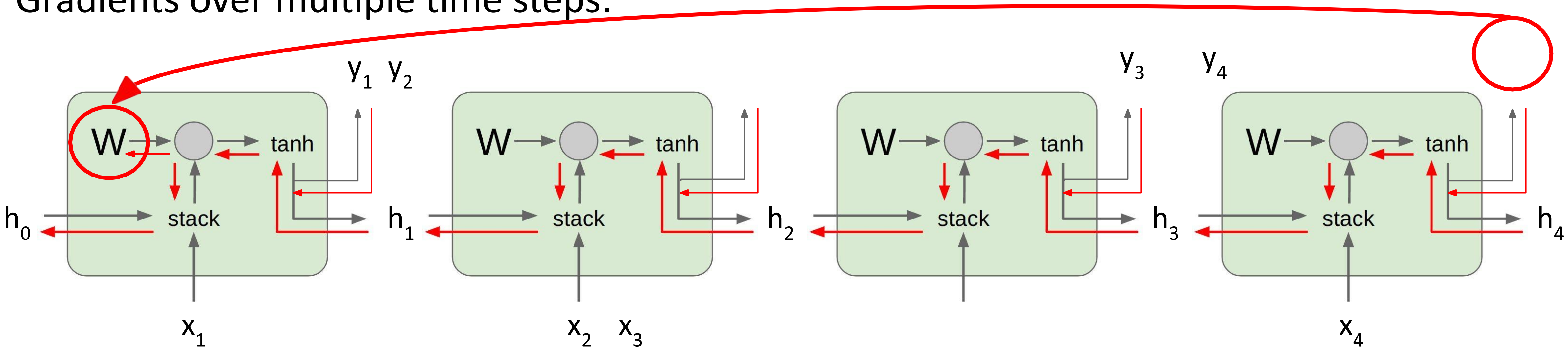
$$h_t = o \odot \tanh(c_t)$$



How much does memory affect output

Revisiting Vanilla RNN Gradient Flow

Gradients over multiple time steps:



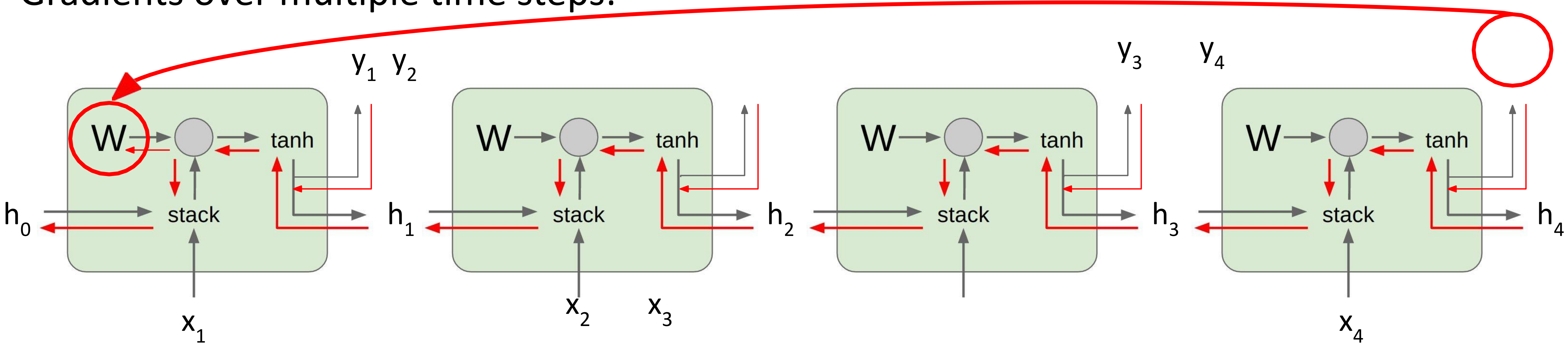
Computing gradient of h_0
Involves many factors of W
and repeated tanh

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Revisiting Vanilla RNN Gradient Flow

Gradients over multiple time steps:



What if we assumed no non-linearity?

Computing gradient of h_0 Involves many factors of W and repeated tanh

Largest singular value > 1: **Exploding gradients** → **Gradient clipping**: Scale gradient if its norm is too big

Largest singular value < 1: **Vanishing gradients** → Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN   LSTM

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Four gates

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

Cell state

$$c_t = f \odot c_{t-1} + i \odot g$$

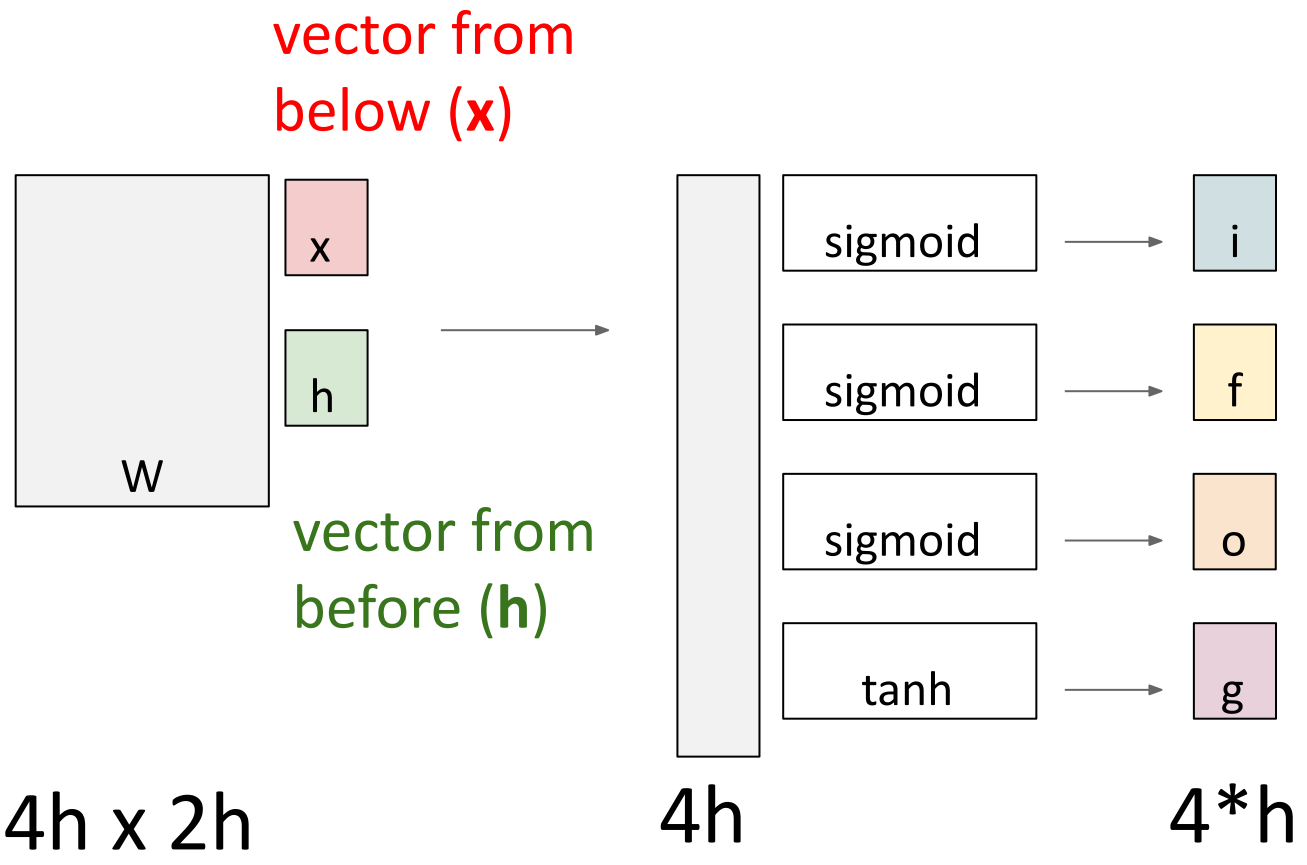
Hidden state

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



g: <u>Gate</u>(?), How much to write to cell

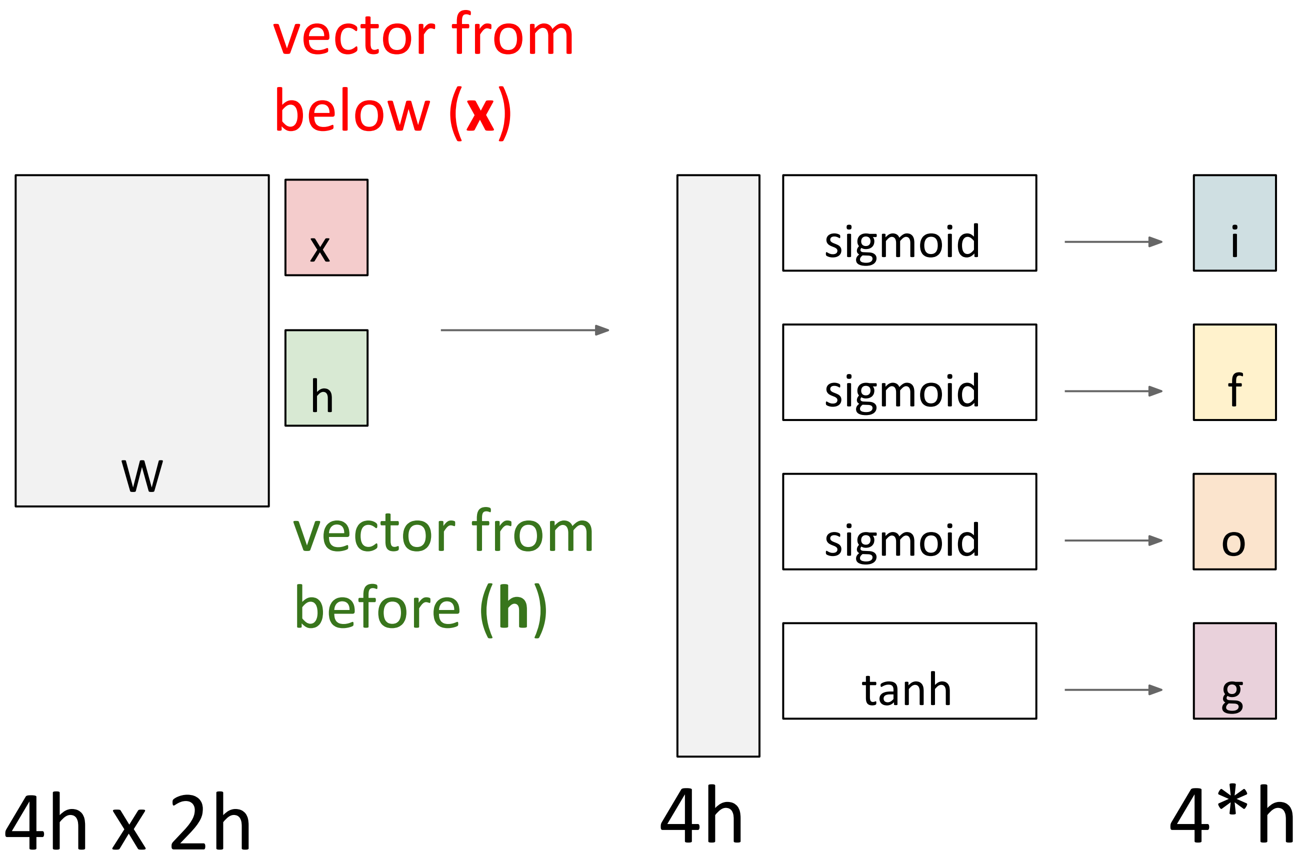
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \boxed{\tanh} \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot \boxed{g}$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



i: Input gate, whether to write to cell

g: Gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \boxed{\tanh} \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot \boxed{g}$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

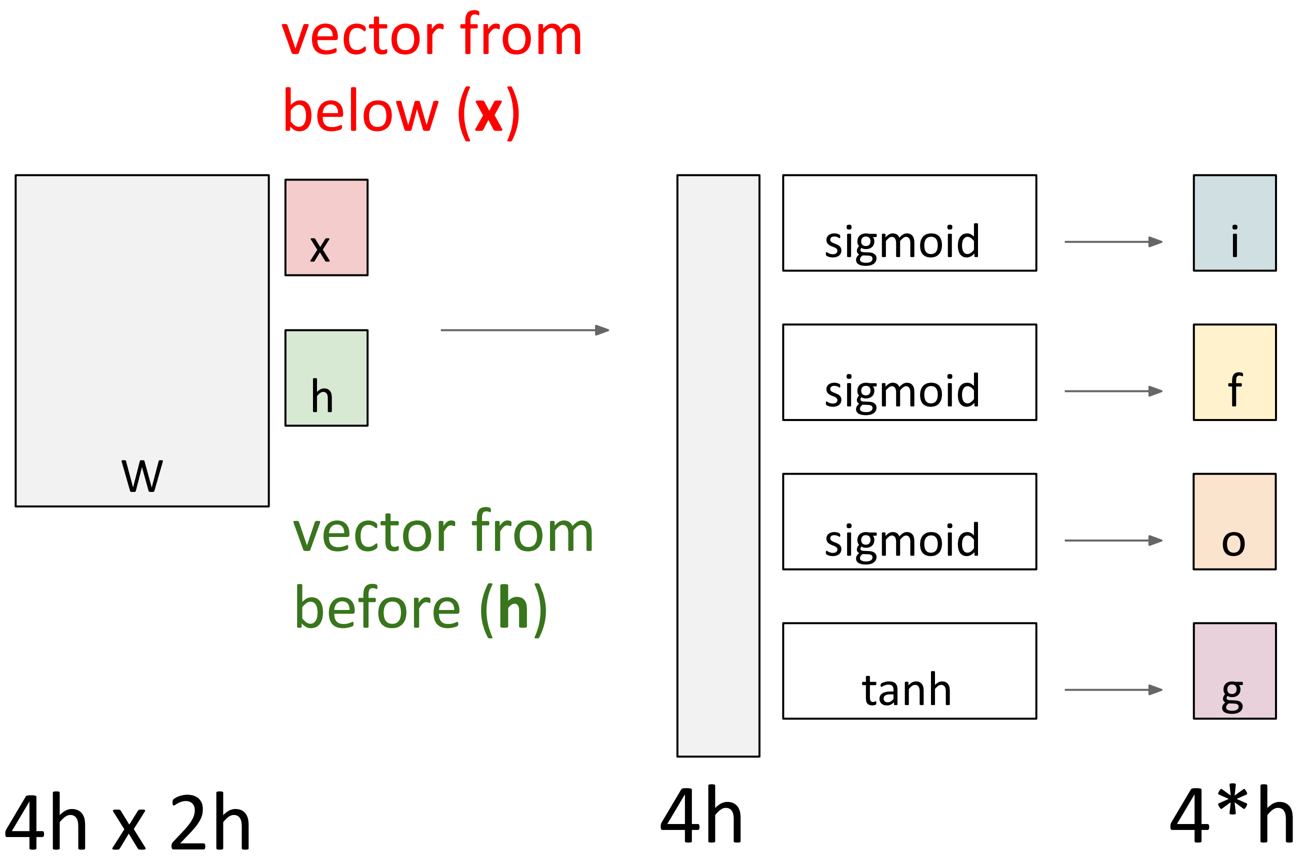
[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

o: Output gate, How much to reveal cell

g: Gate (?), How much to write to cell



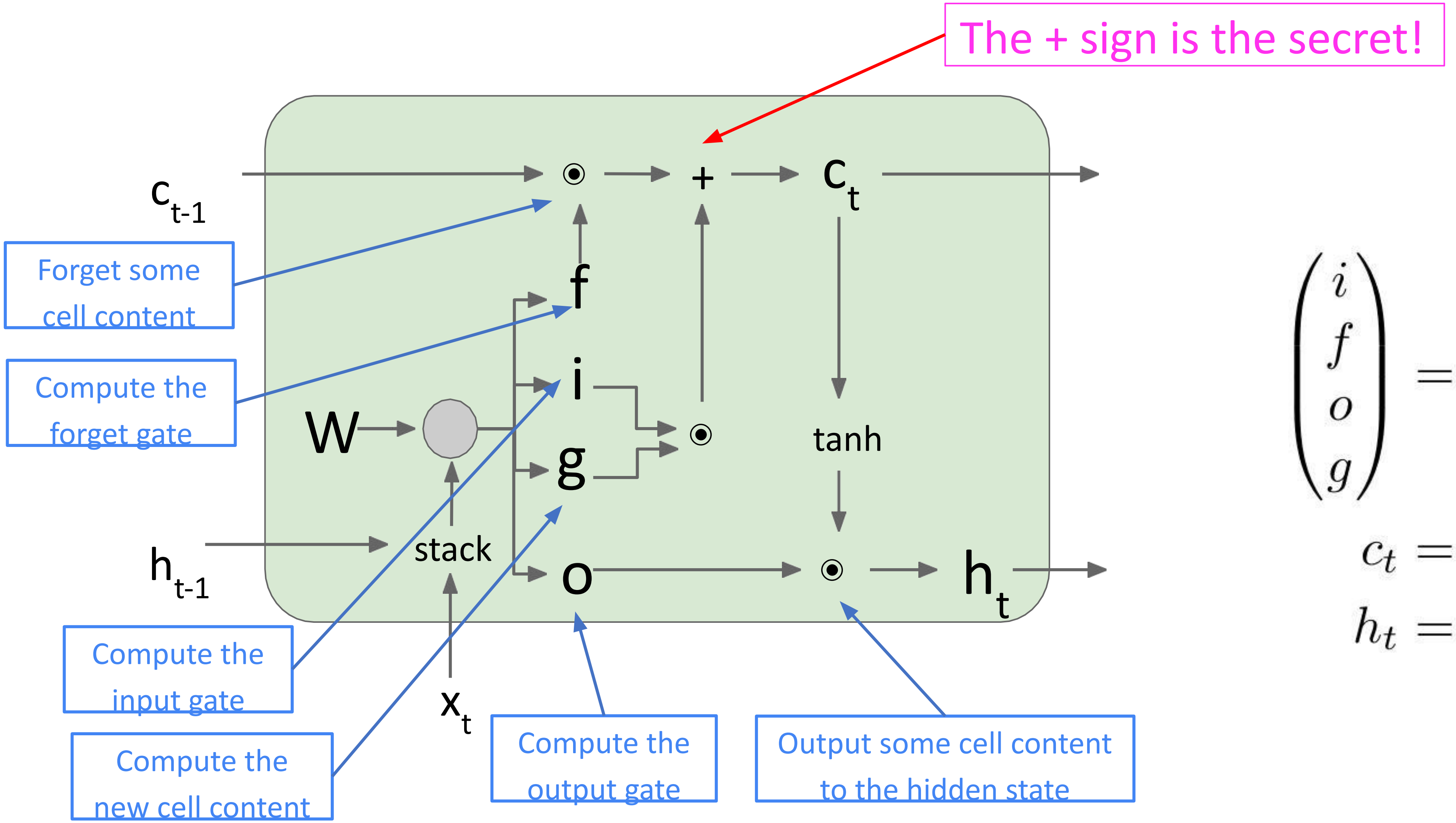
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \boxed{\sigma} \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = \boxed{f} \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

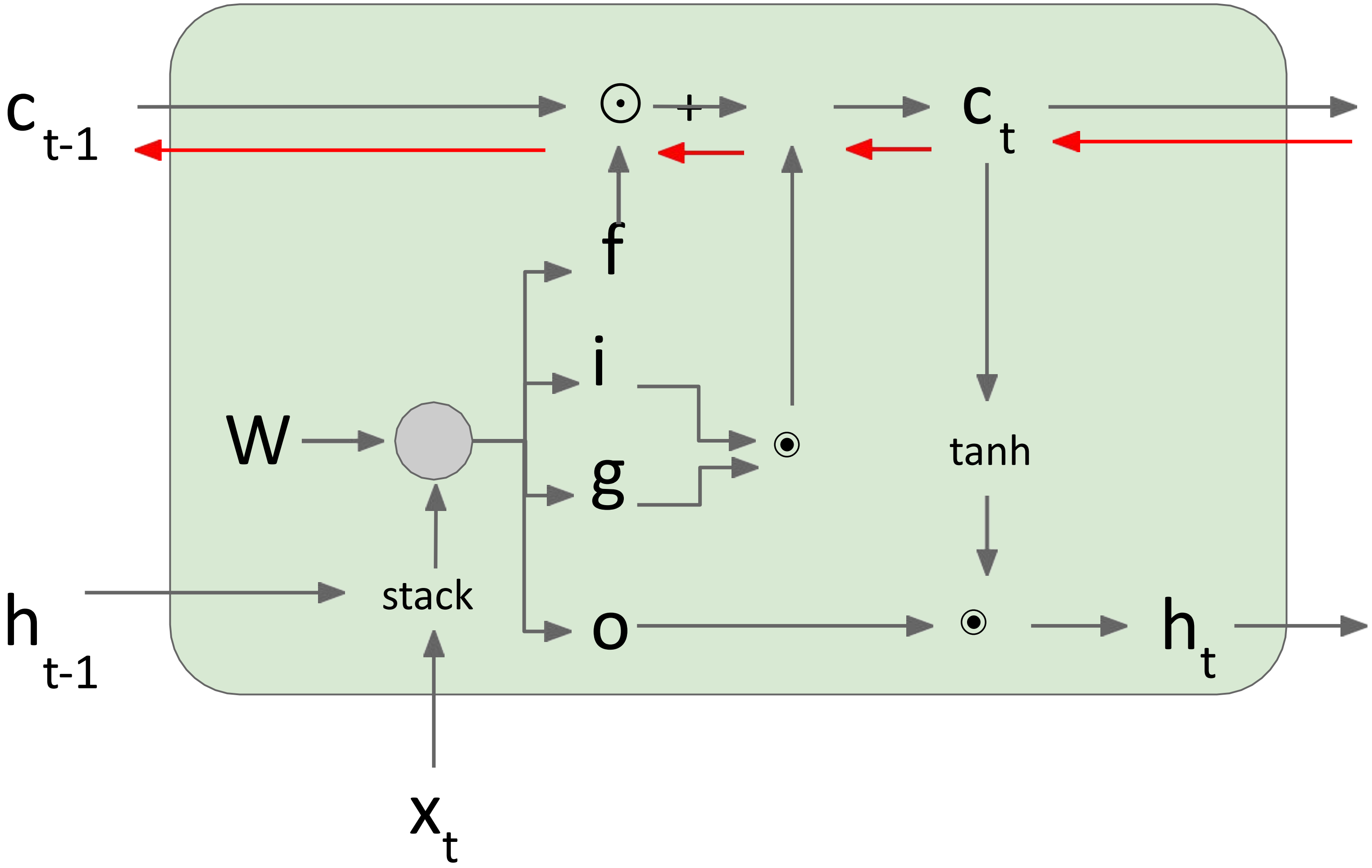


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

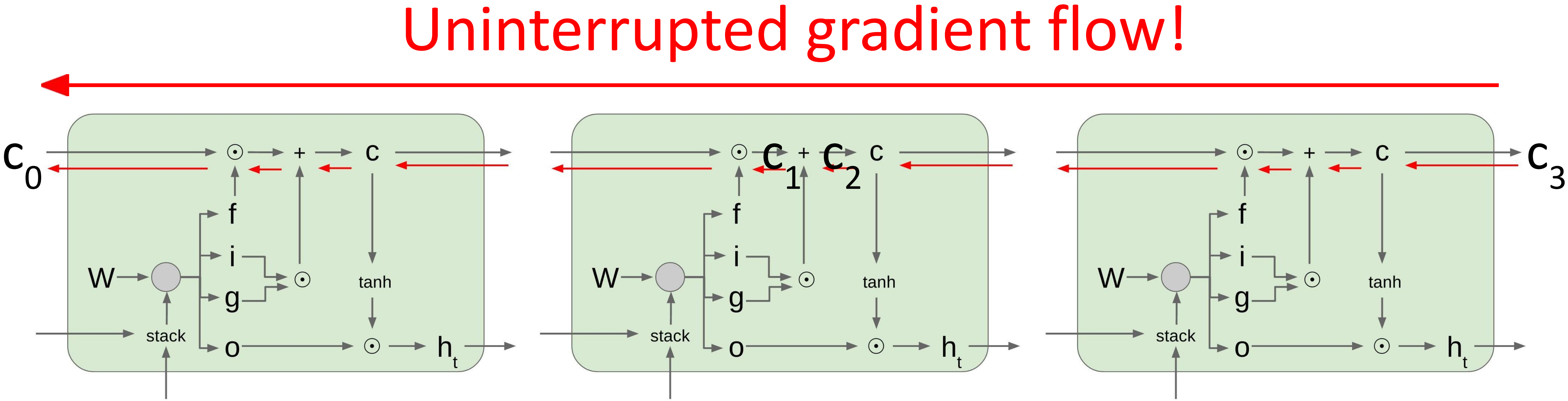
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Do LSTMs solve the vanishing gradient problem?

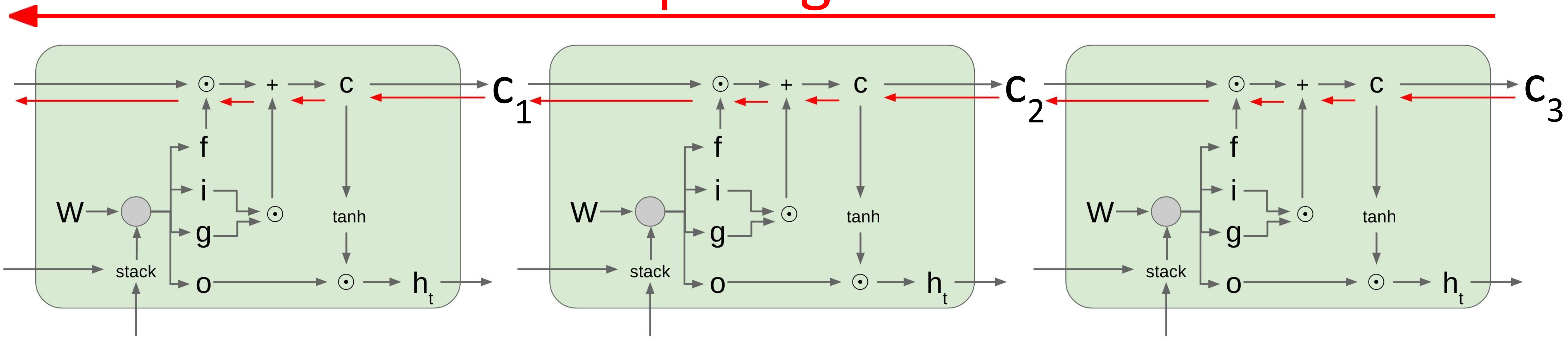
The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

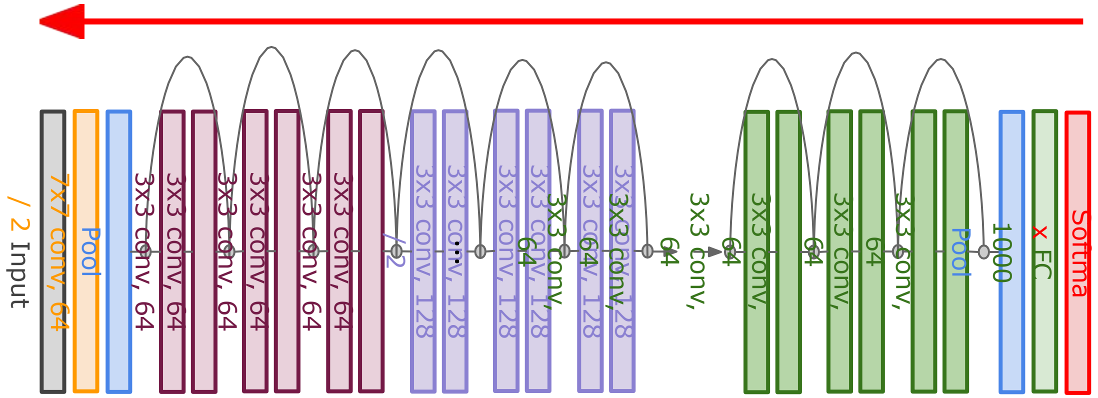
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, “Highway Networks”, ICML DL Workshop 2015

Do LSTMs solve the vanishing gradient problem?

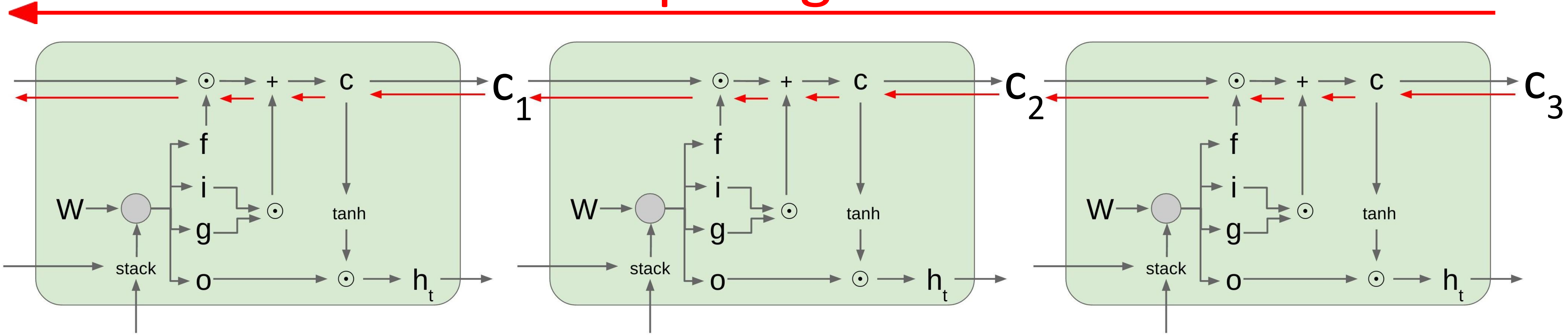
The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

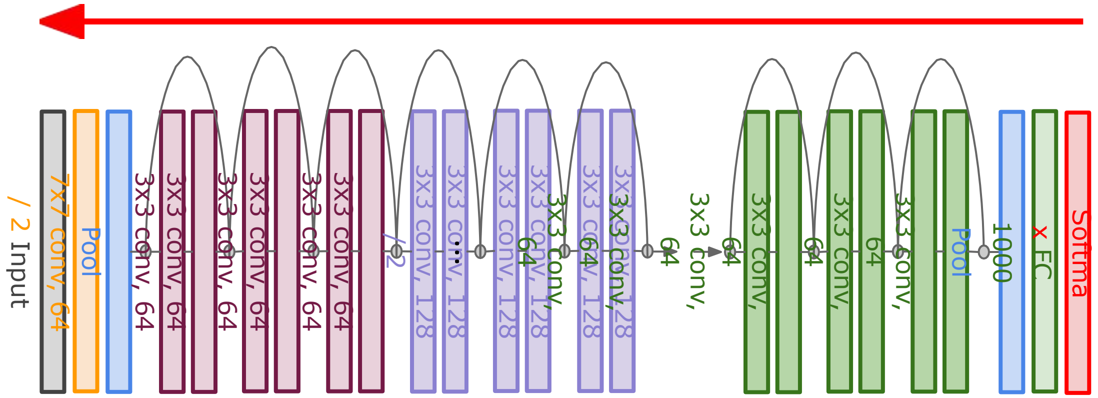
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Long Short Term Memory (LSTM): Gradient Flow

Uninterrupted gradient flow!



Similar to ResNet!



In between:

Highway Networks

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, “Highway Networks”, ICML DL Workshop 2015

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

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:

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

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better understanding (both theoretical and empirical) is needed.

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$
\begin{aligned}
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)\\
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)\\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)\\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

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:

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

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better understanding (both theoretical and empirical) is needed.

Sequence to Sequence with Attention

A Language Modeling perspective and a precursor to Transformers

What is a Language Model?

An LM is

- a probability distribution over sequence of words.
- a way to predict the next word

For a sentence S consisting of m words

$$S = w_1 w_2 w_3 \ldots \ldots . . w_m$$

In Language Model, we assume:

$$P(S) = P(w_1 w_2 w_3 \ldots \ldots . . w_m)$$

$$= P(w_1) \times P(w_2|w_1) \times \cdots \times P(w_m|w_{m-1} \ldots w_1)$$

What is a Language Model?

Using LM, we can find out

- If a sentence S_1 is more likely than another S_2 (conditioned on q, but ignore for now).

For example:

- S_1: Virat Kohli plays cricket for India.
- S_2: plays Kohli cricket for India Virat.
- S_3: Virat Kohli plays plays for India.

Which is more likely?

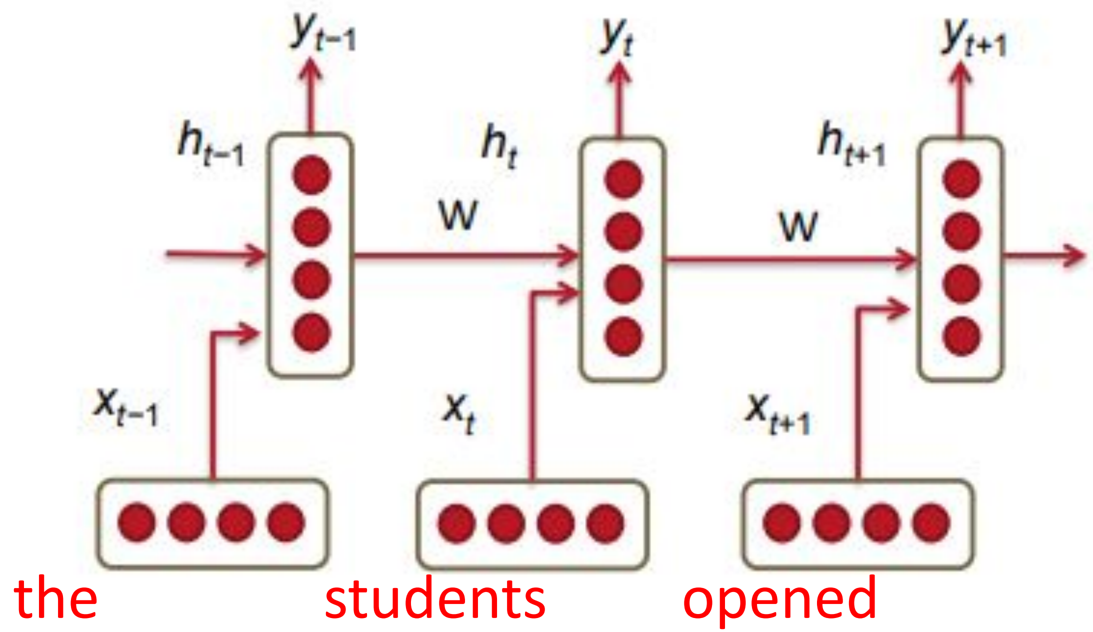
Obviously S_1. Hence our LM should say **$P(S_1) > P(S_2)$** and **$P(S_1) > P(S_3)$**.

Recurrent Neural Networks - LM

Recurrent Neural Networks (RNN)

- Each word depends on all previous words in the "sentence/paragraph".
- ***RNNs add the immediate past to the present.***

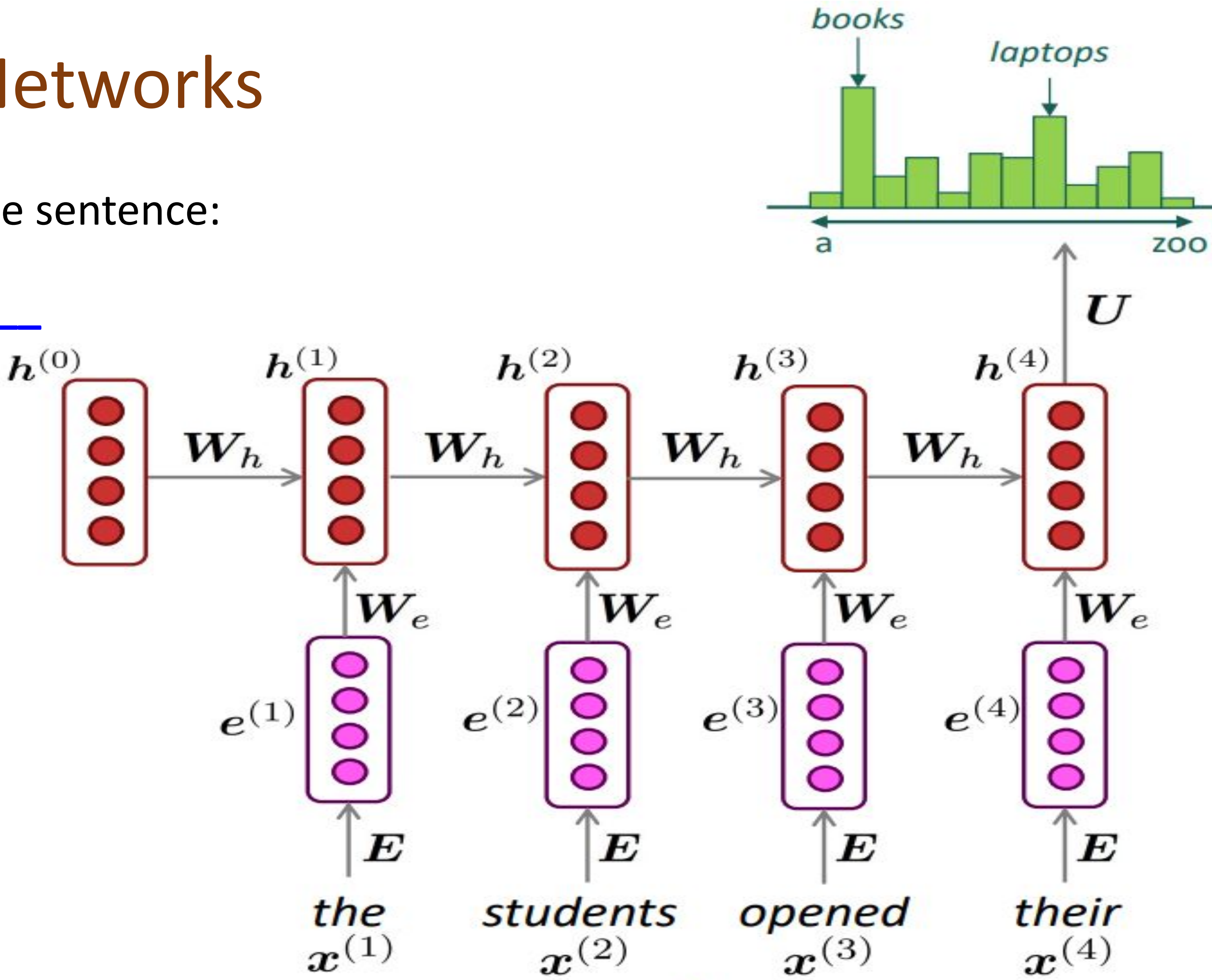
Here, is a simple architecture of RNN:



Recurrent Neural Networks

Working of RNN for the example sentence:

the students opened their _____



Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$\text{perplexity} = \underbrace{\prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)}_{\text{Inverse probability of corpus, according to Language Model}}^{1/T}$$

Normalized by number of words

- This is equal to the exponential of the cross-entropy loss $J(\theta)$:

$$= \prod_{t=1}^{T} \left(\frac{1}{\hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}}} \right)^{1/T} = \exp\left(\frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}^{(t)}_{\boldsymbol{x}_{t+1}} \right) = \exp(J(\theta))$$

Lower perplexity is better!

RNNs greatly improved perplexity over what came before

n-gram model →

Increasingly complex RNNs

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

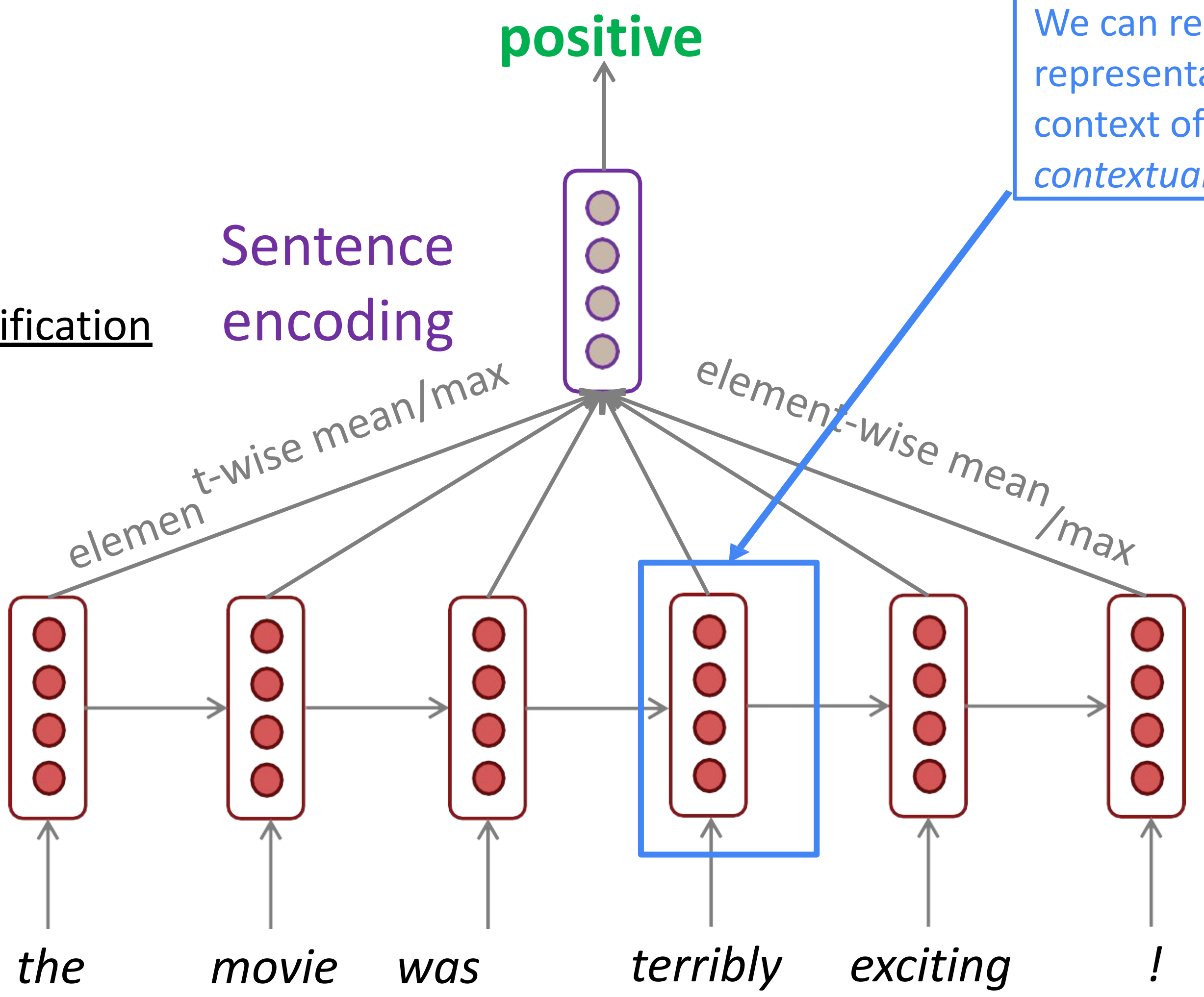
Source: https://research.fb.com/building-an-efficient-neural-language-model-over-a-billion-words/

In Summary …

- **Language Model**: A system that predicts the next word
- Recurrent Neural Network $\neq$ Language Model
  - RNNs can be used for many other things (see later)
- Language Modeling is a traditional subcomponent of many NLP tasks, all those involving generating text or estimating the probability of text.
- In practical tasks:
  - Both left and right context is necessary to model. Bidirectional RNNs
  - Need to remember distant Past. LSTM does NOT FULLY solve this. Attention!

Bidirectional RNNs: motivation

<u>Task: Sentiment</u> <u>Classification</u>



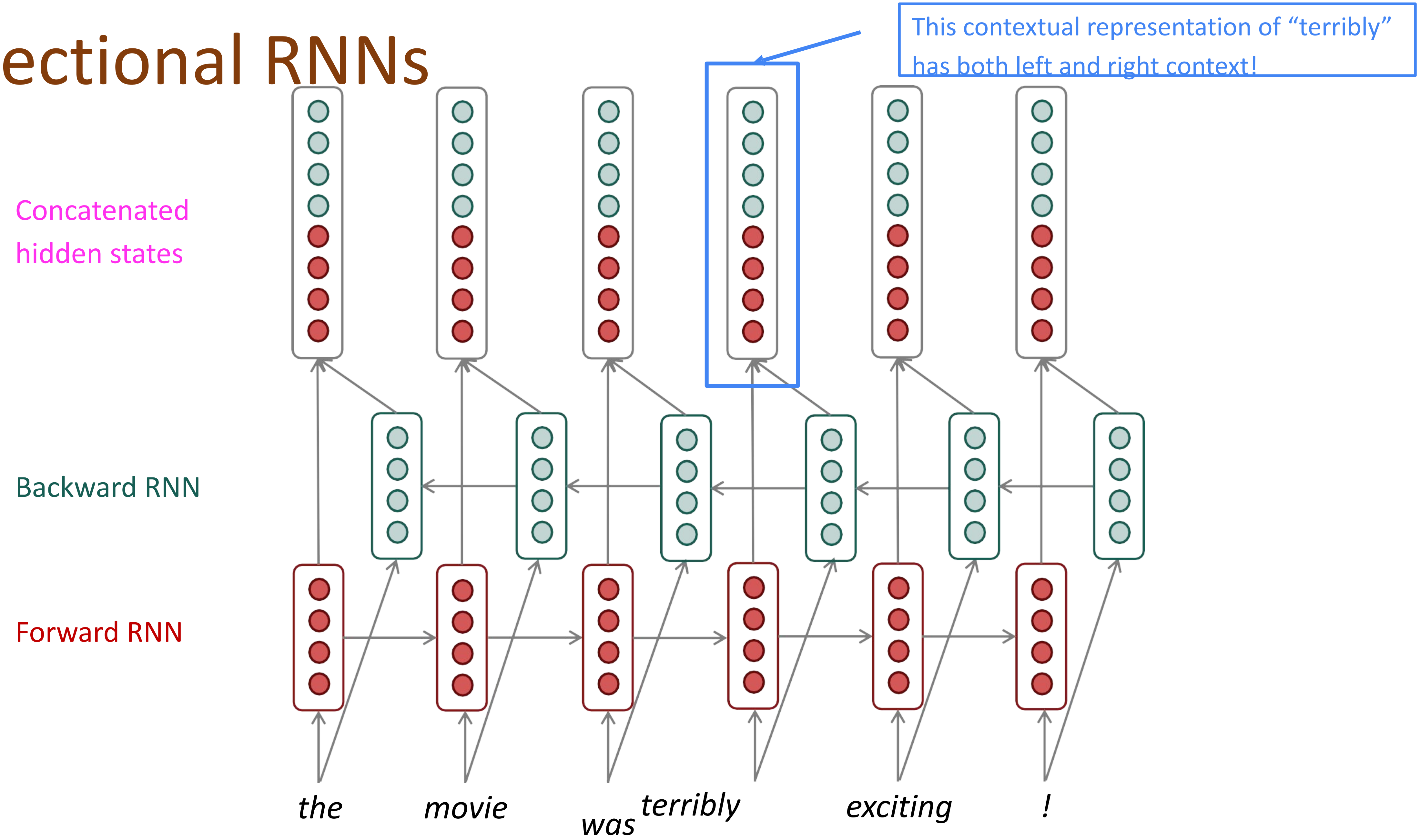
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Bidirectional RNNs



Bidirectional RNNs

On timestep *t*:

This is a general notation to mean "compute one forward step of the RNN" – it could be a simple RNN or LSTM computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

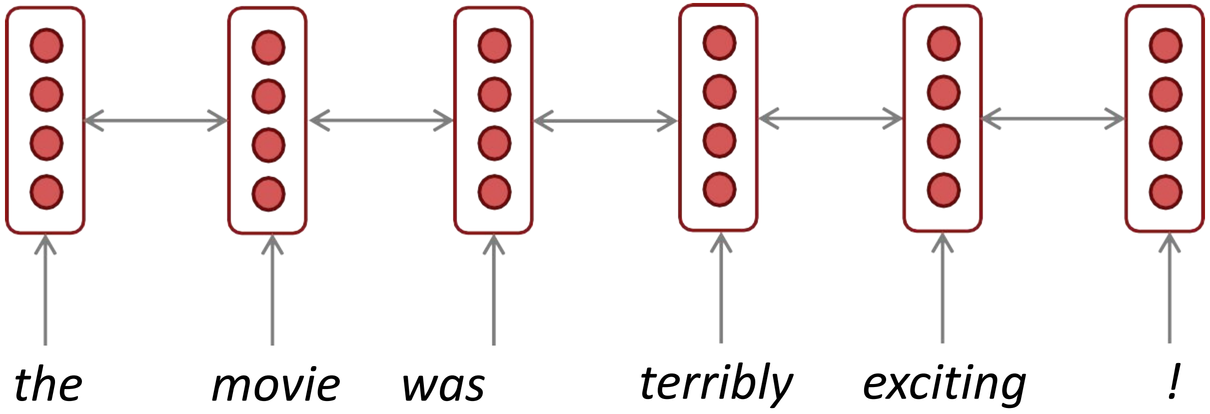
Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNNs: simplified diagram



The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the entire input sequence
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.

- If you do have entire input sequence (e.g., any kind of encoding), bidirectionality is powerful (you should use it by default).

- For example, BERT (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.

Will revisit this in Transformers!

Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)

- We can also make them “deep” in another dimension by applying multiple RNNs – this is a multi-layer RNN.

- This allows the network to compute more complex representations
  - The lower RNNs should compute lower-level features and the higher RNNs should compute higher-level features.

These concepts would be useful in Transformers!

Sequence to Sequence Model

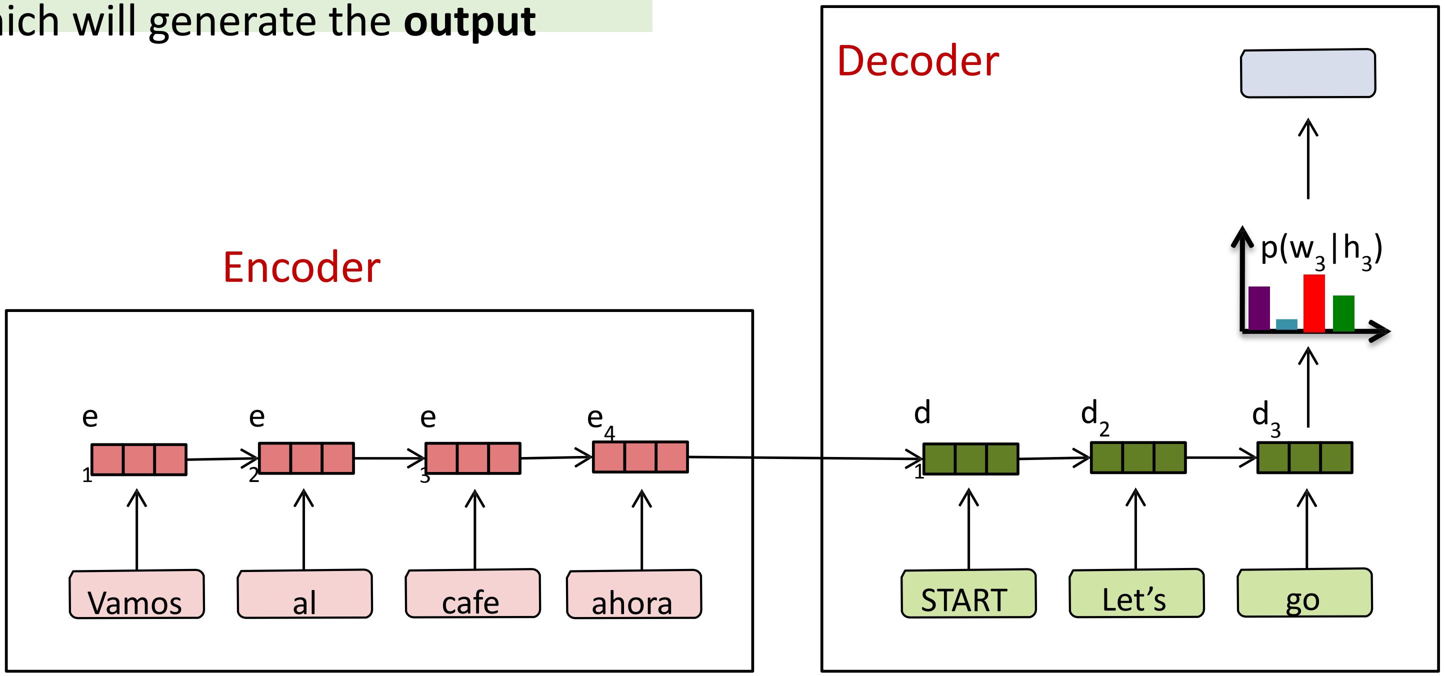
Suppose you want generate a sequence conditioned on another input

Key Idea:

1. Use an **encoder** model to generate a vector representation of the **input**
2. Feed the output of the encoder to a **decoder** which will generate the **output**

Applications:

- translation: Spanish to English
- summarization: article to summary
- speech recognition: speech signal to transcription

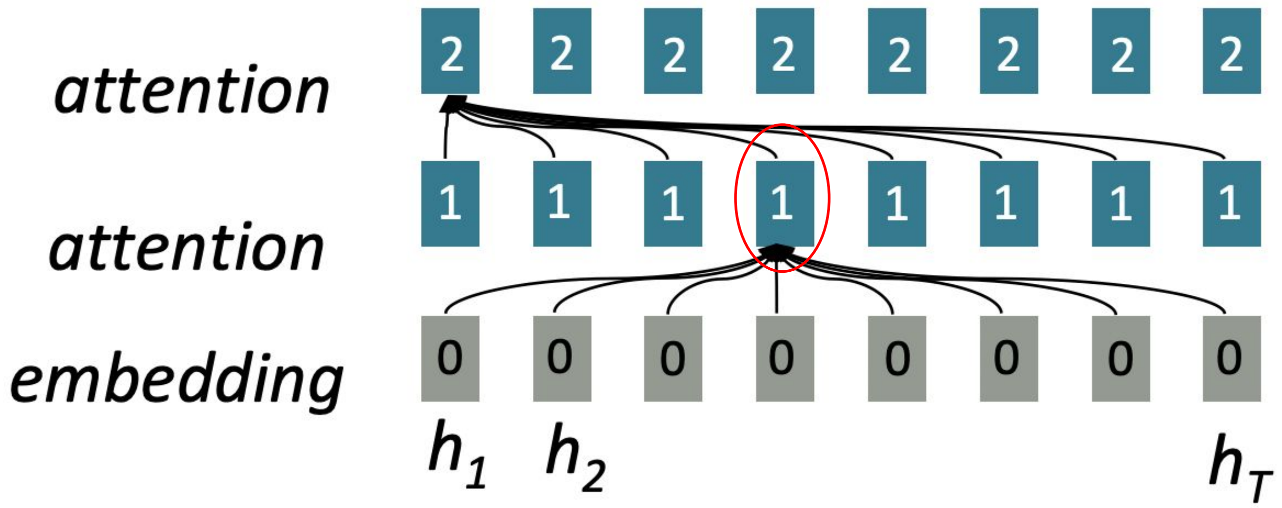


Recurrence to Attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
  - For example, Layer 2 each node j computes

$$\sum_{i=1}^{T} \alpha_i w_{ij} h_i, \text{ s.t. } \Sigma_i \alpha_i = 1$$

- Max. interaction distance: O(1).



Sequence to Sequence with RNNs

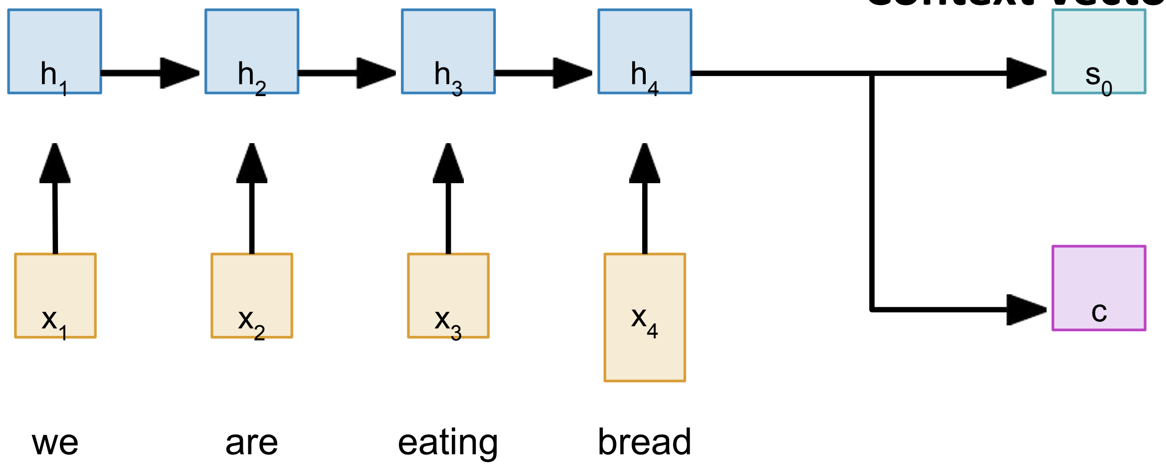
Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

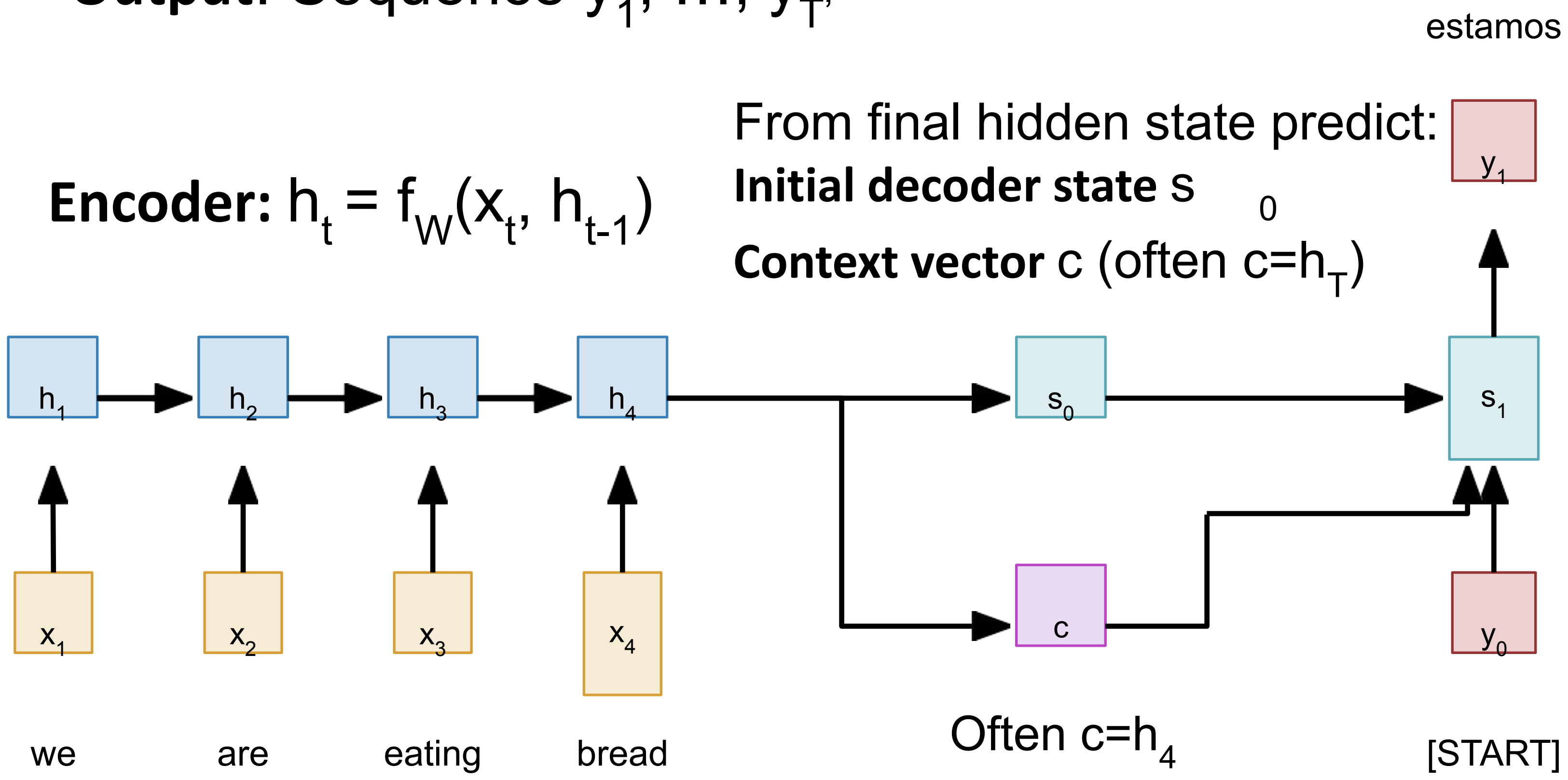
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 201

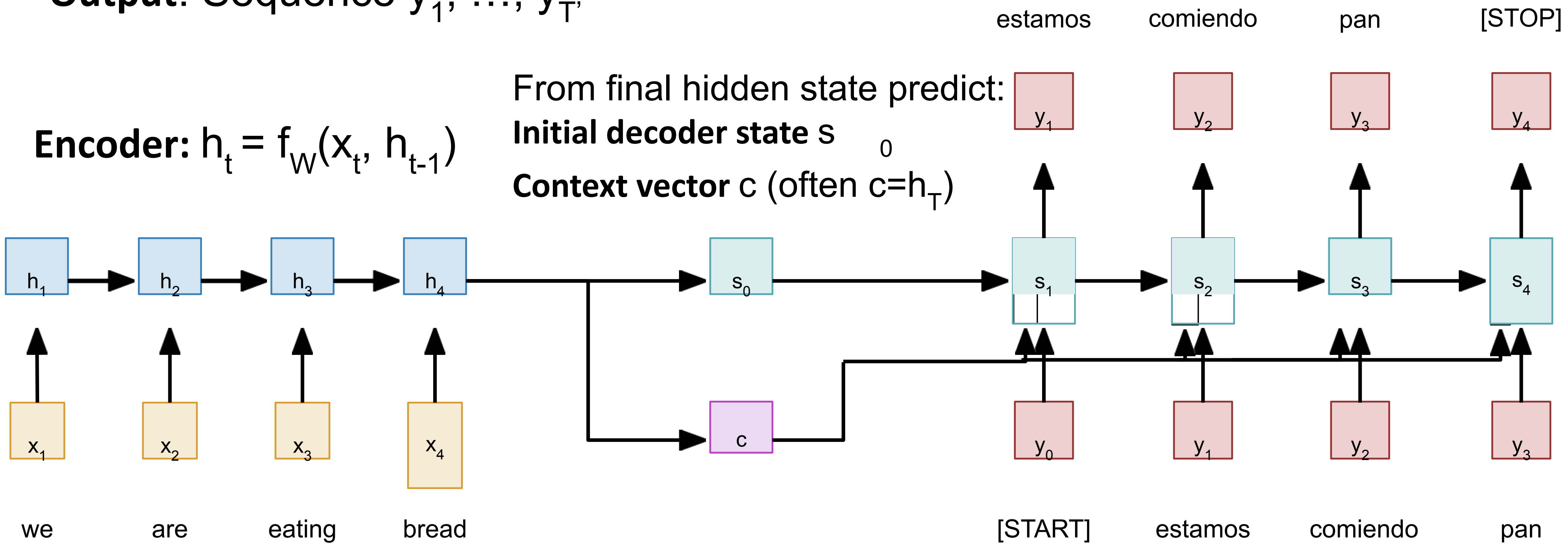
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

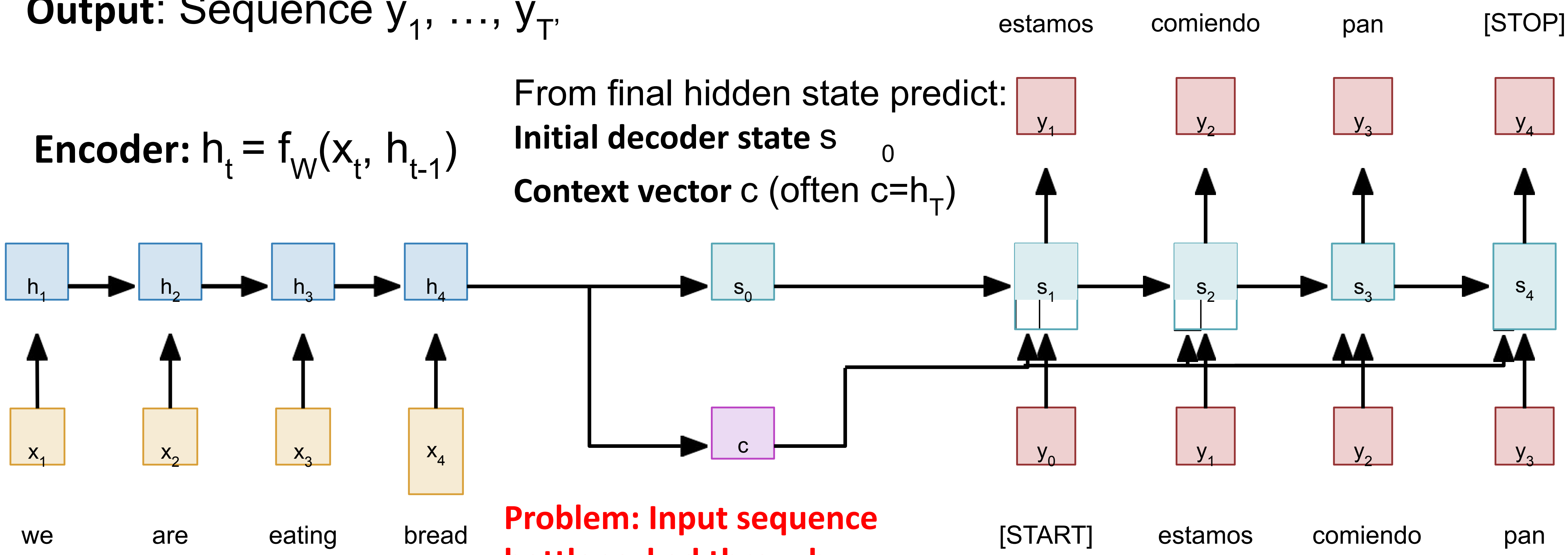
Sequence to Sequence with RNNs

Input: Sequence $x_1, \dots x_T$
Output: Sequence $y_1, \dots, y_{T'}$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if T=1000?

Idea: use new context vector at each step of decoder!

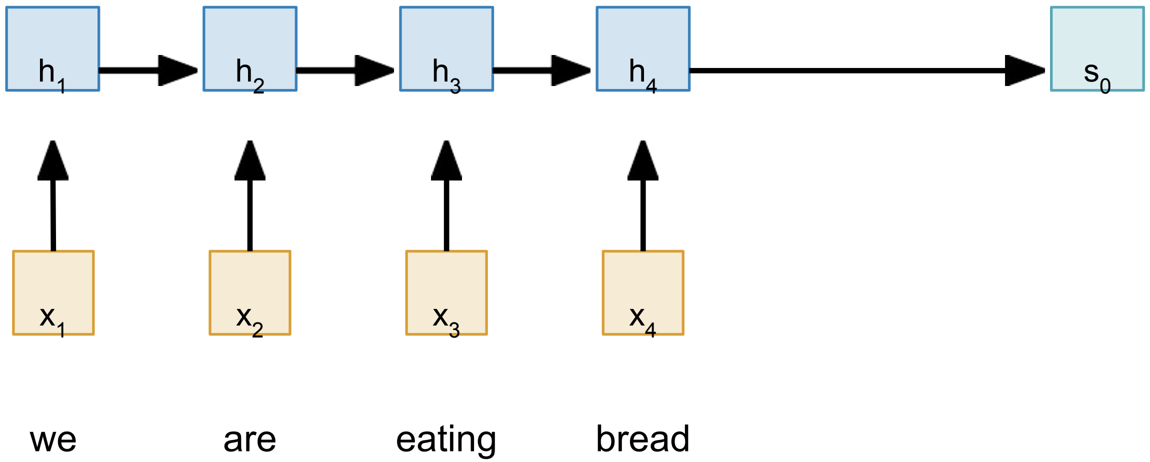
Sutskever et al, "Sequence to sequence learning with neural networks",

Sequence to Sequence with RNNs and Attention

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state:
Initial decoder state s_0



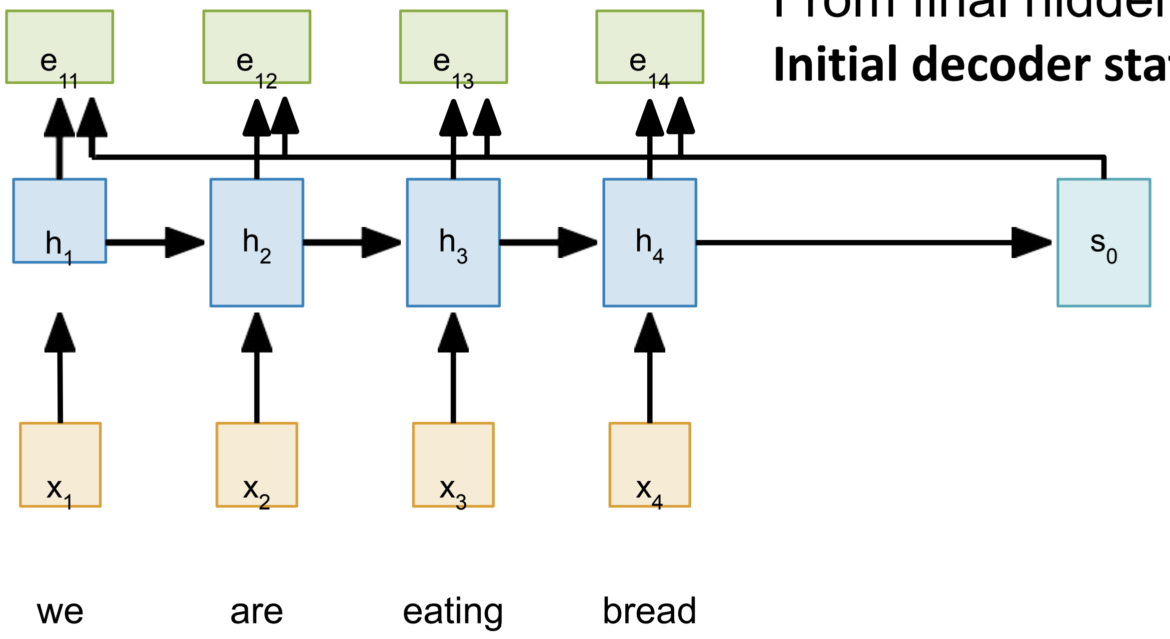
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

Compute (scalar) **alignment scores**

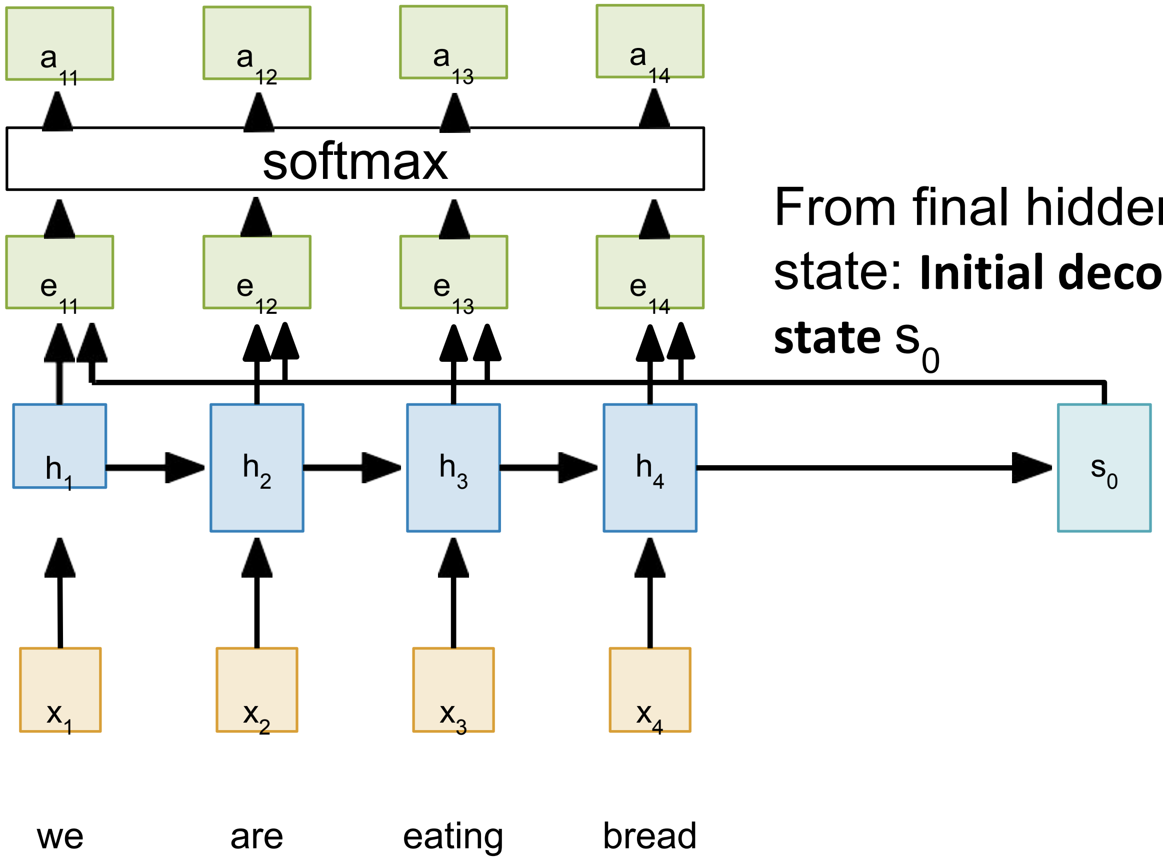
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

From final hidden state:
Initial decoder state s_0



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention



From final hidden state: **Initial decoder state** s_0

Compute (scalar) **alignment scores**

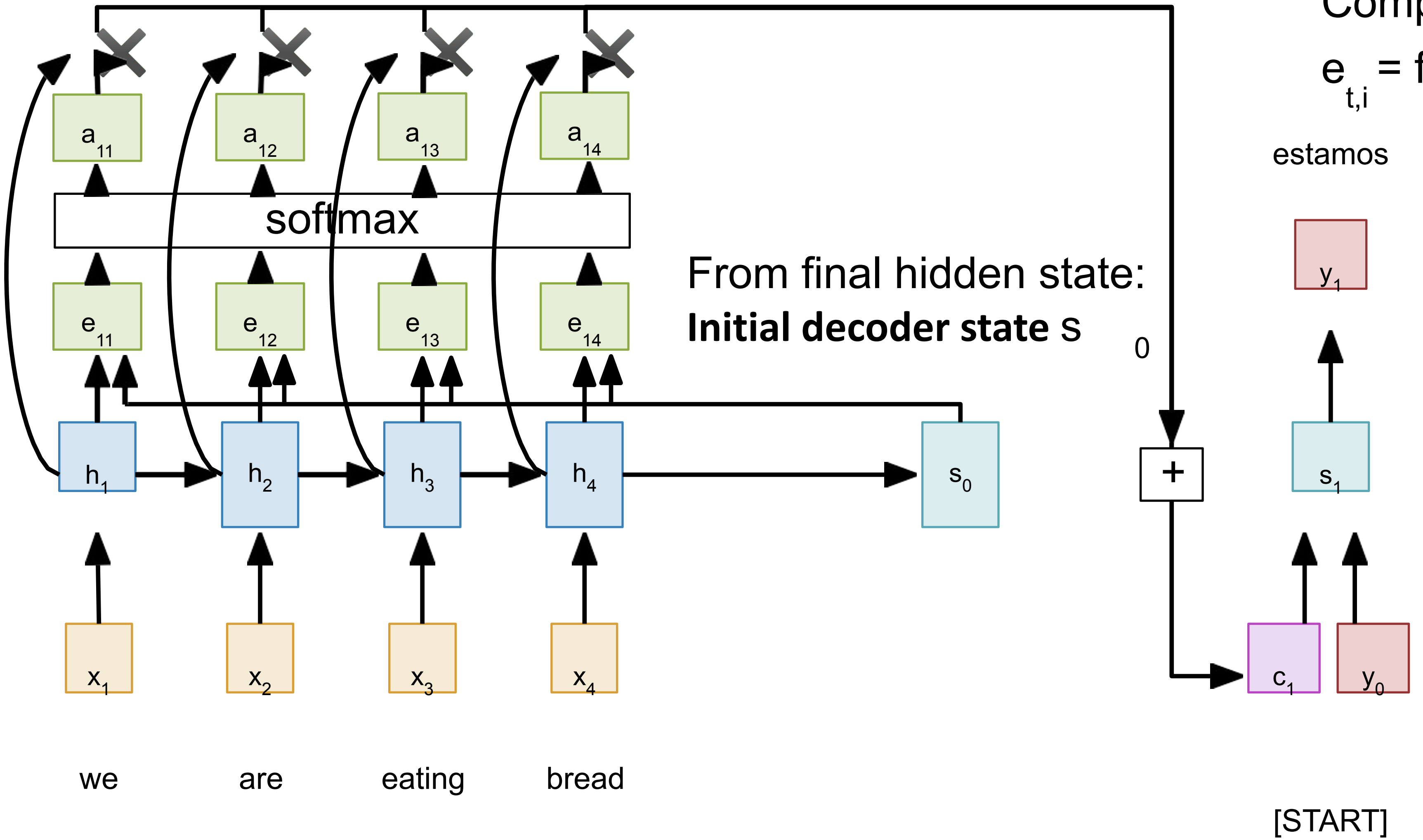
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention



From final hidden state: **Initial decoder state** s_0

Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

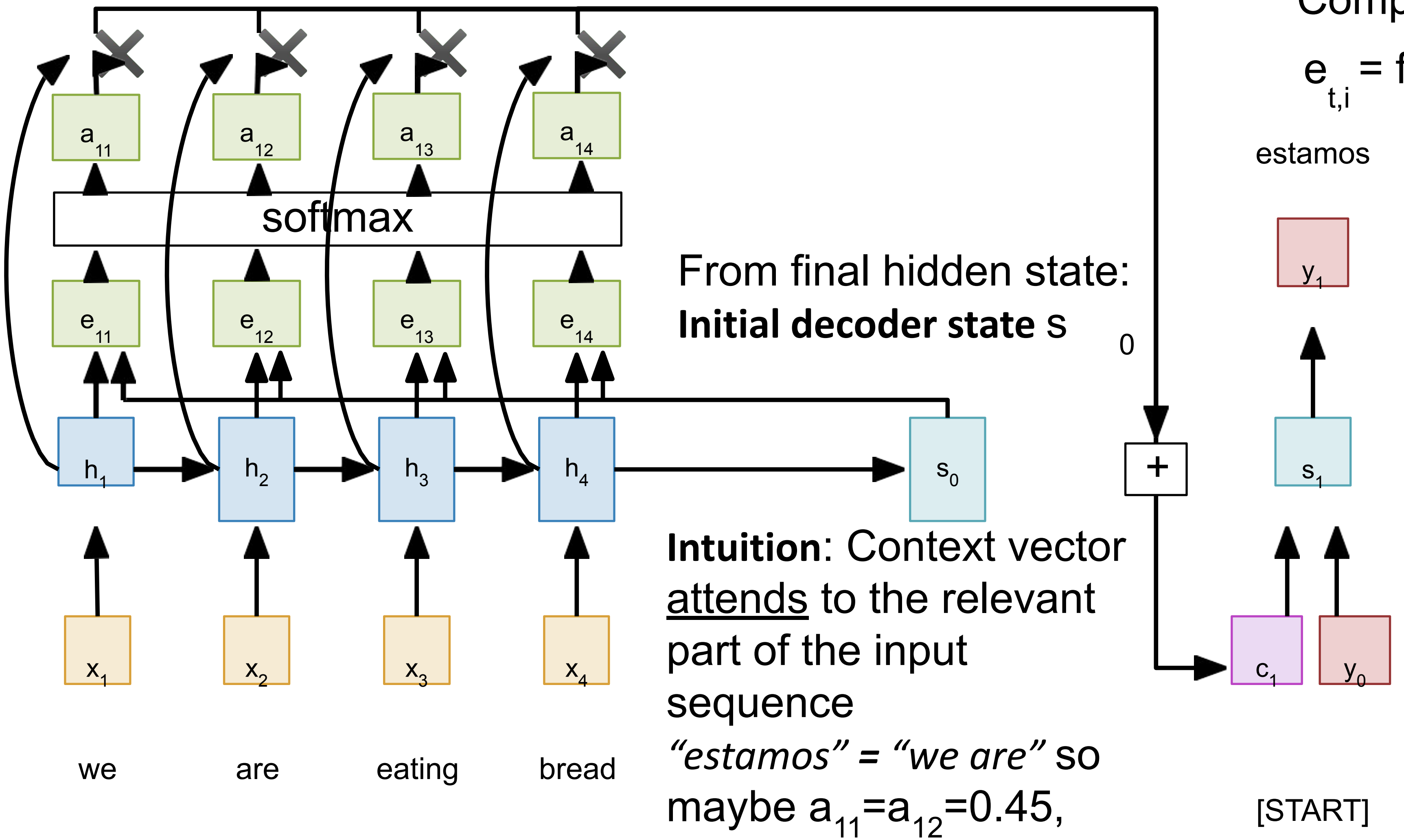
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



From final hidden state: **Initial decoder state** s_0

Compute (scalar) **alignment scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

Intuition: Context vector <u>attends</u> to the relevant part of the input sequence

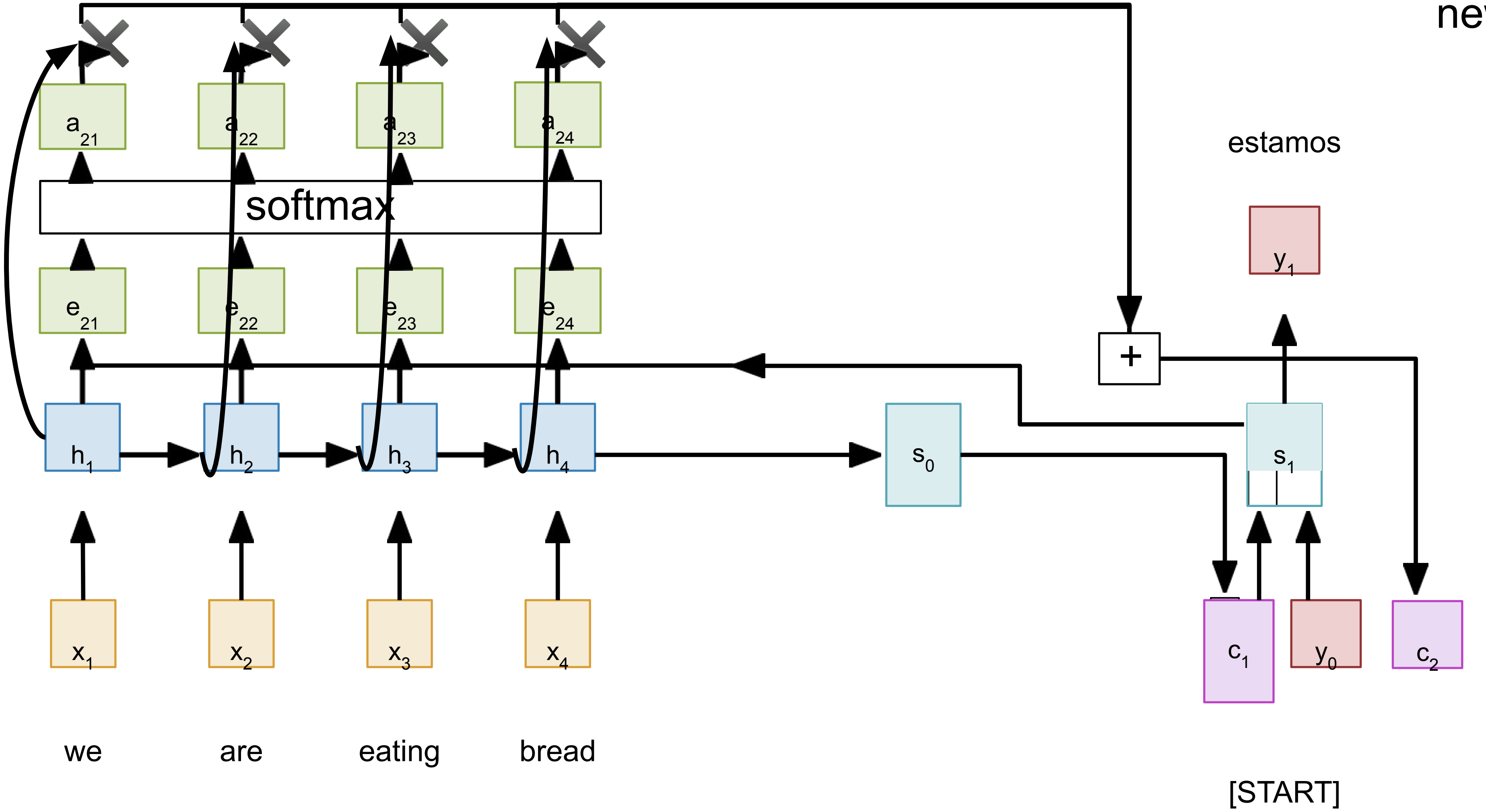
"estamos" = "we are" so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

This is all differentiable! No supervision on attention weights – backprop through everything

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2

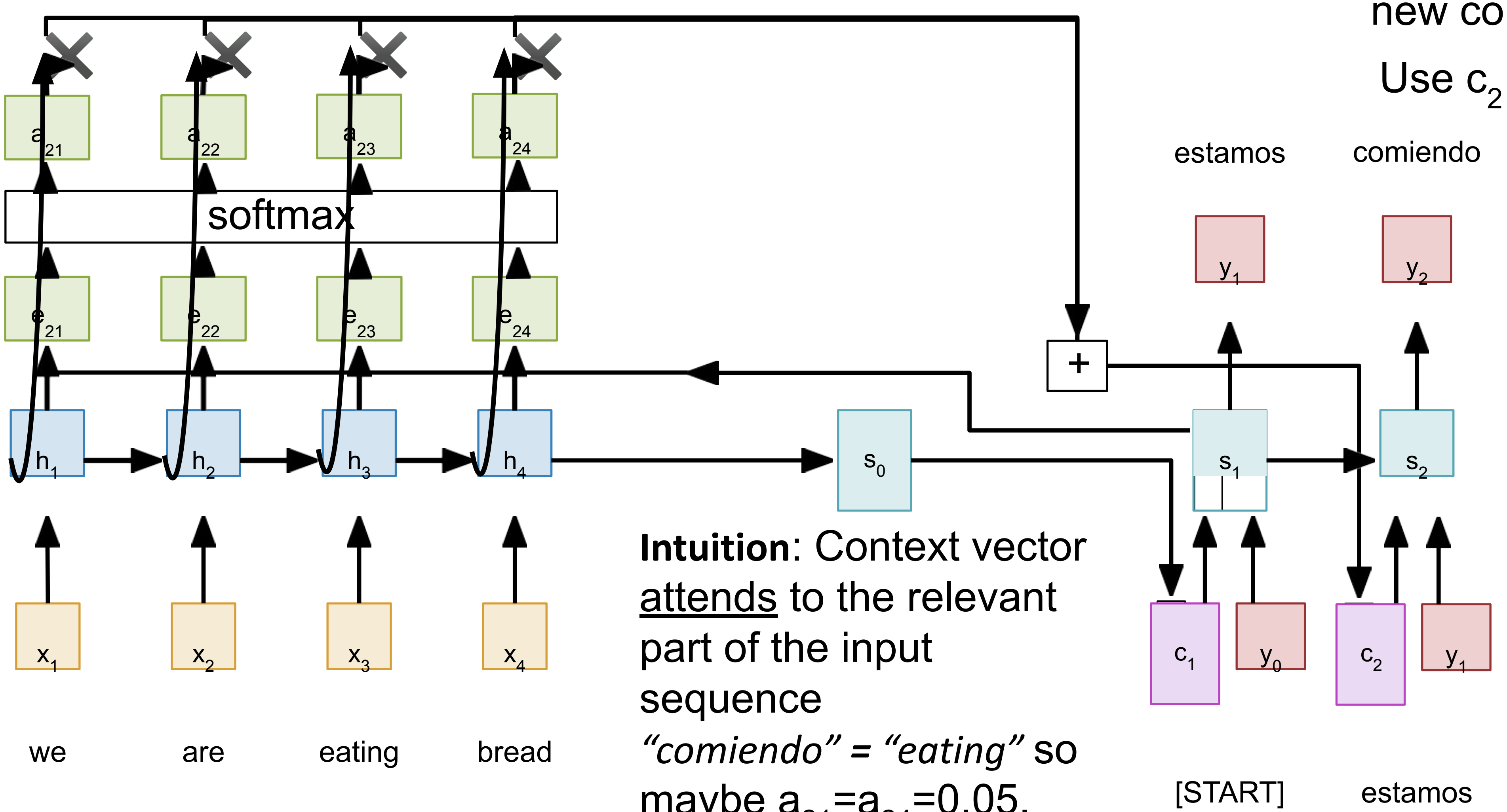


Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2



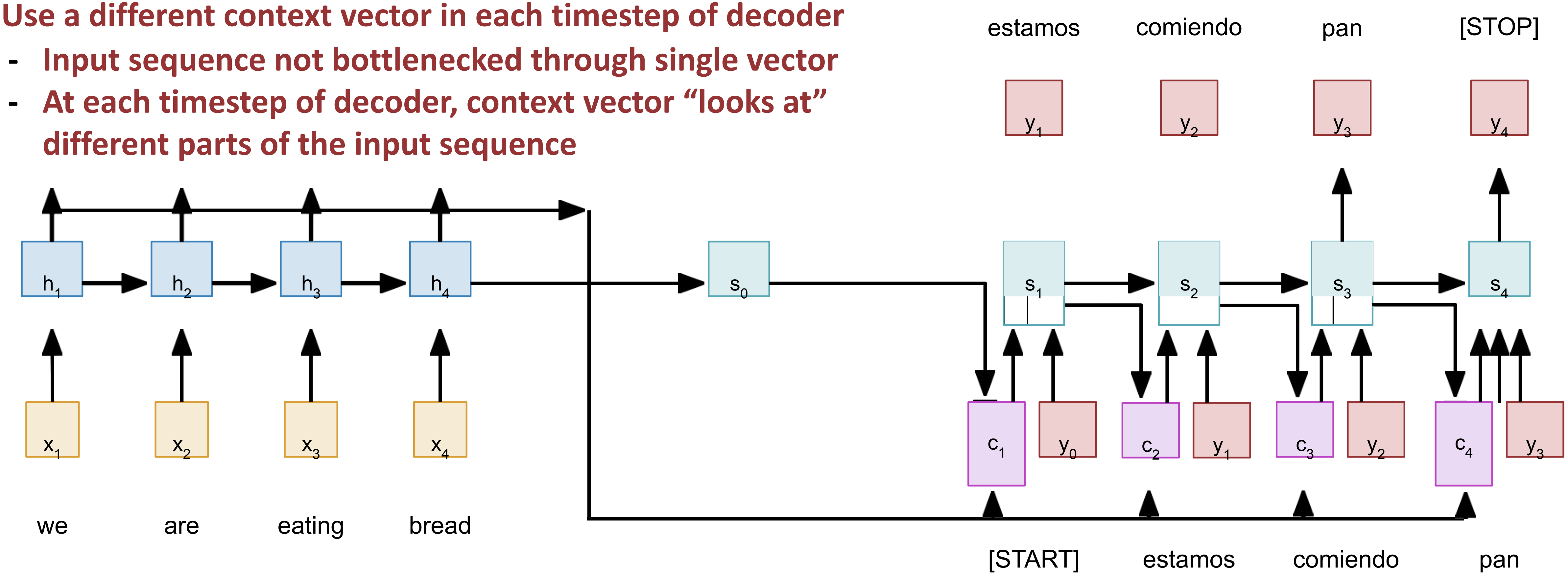
Intuition: Context vector attends to the relevant part of the input sequence

"comiendo" = "eating" so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

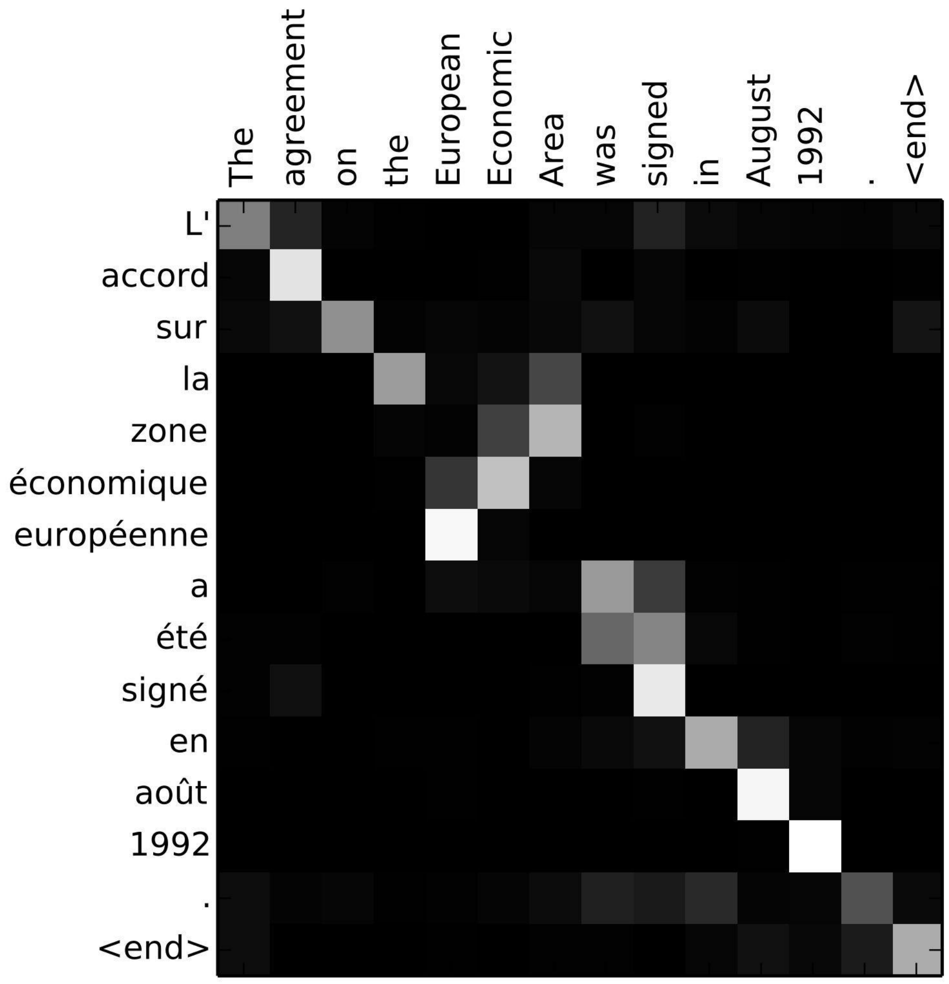
Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



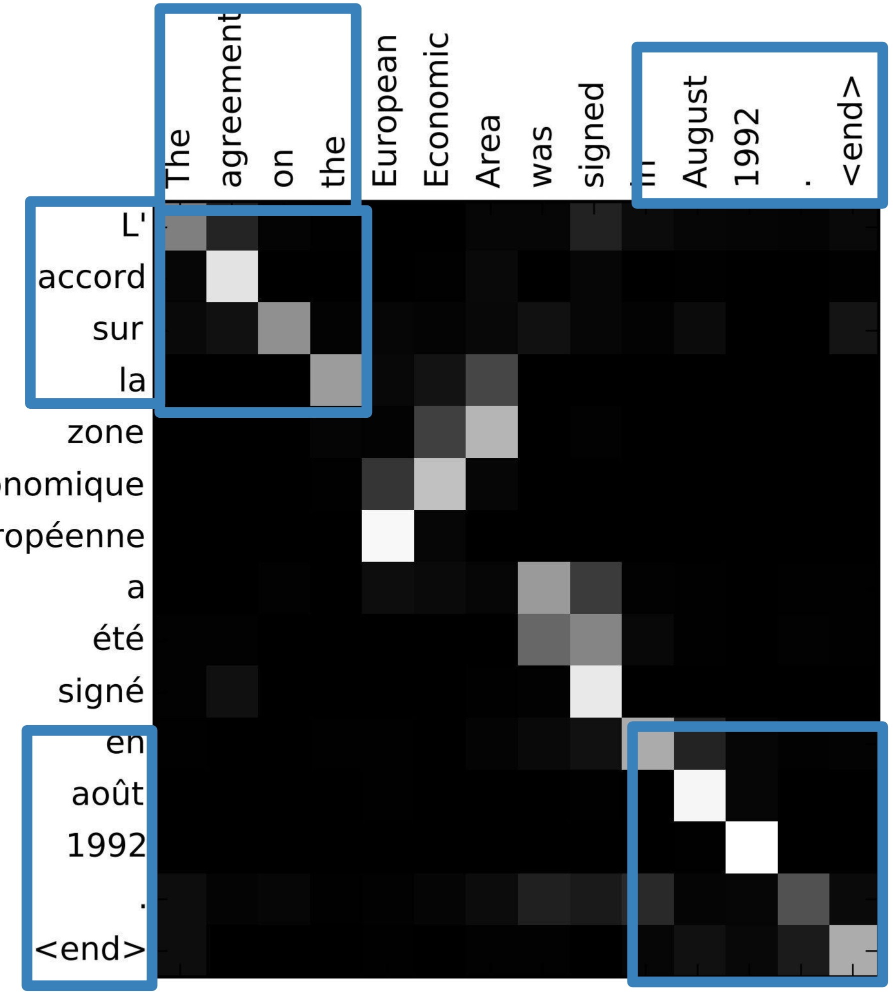
Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed **in August 1992**."

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: "The agreement on the European Economic Area was signed in August 1992."

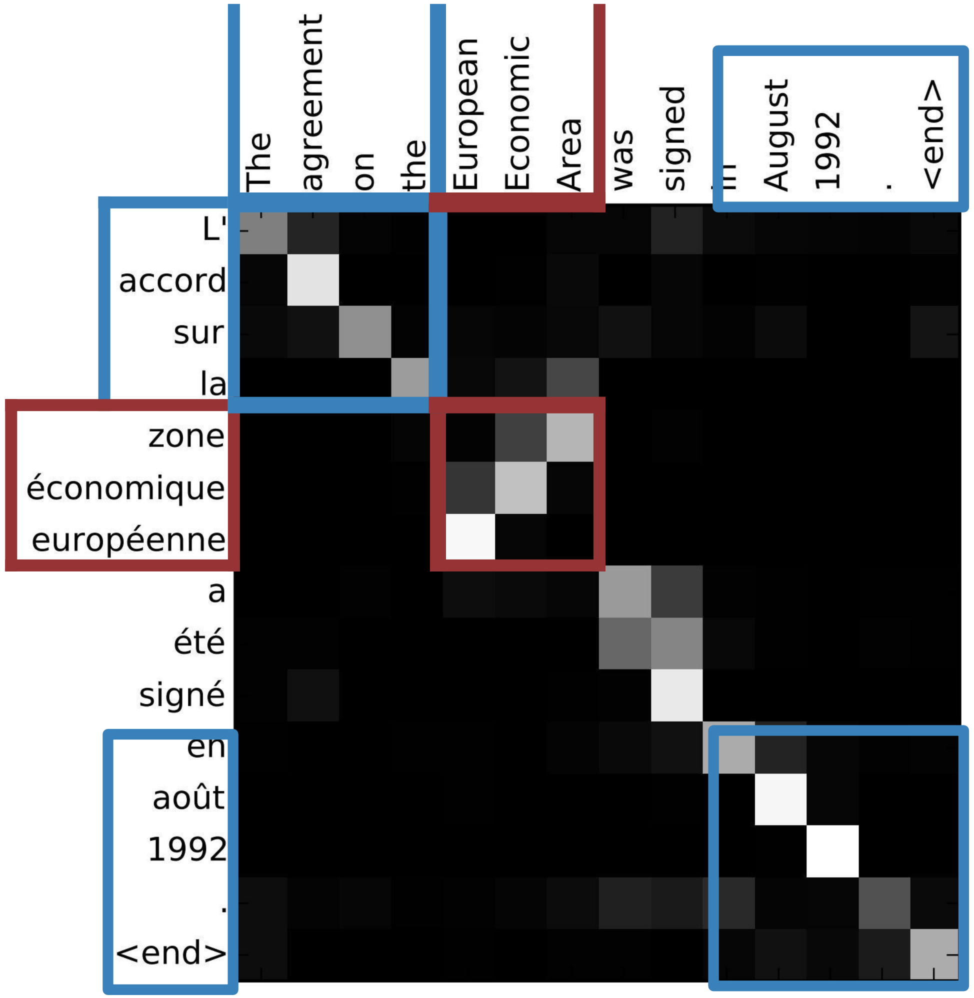
Output: "L'accord sur la zone économique européenne a été signé en août 1992."

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

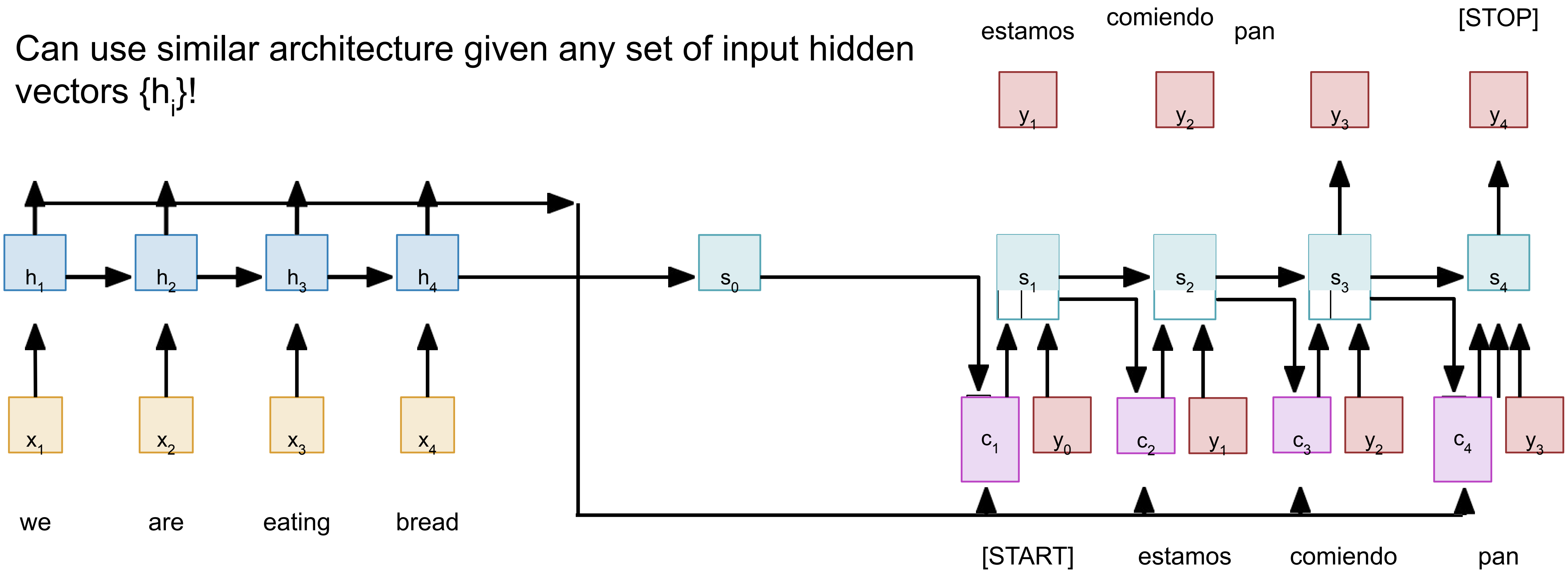
Visualize attention weights $a_{t,i}$



Sequence to Sequence with RNNs and Attention

The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!

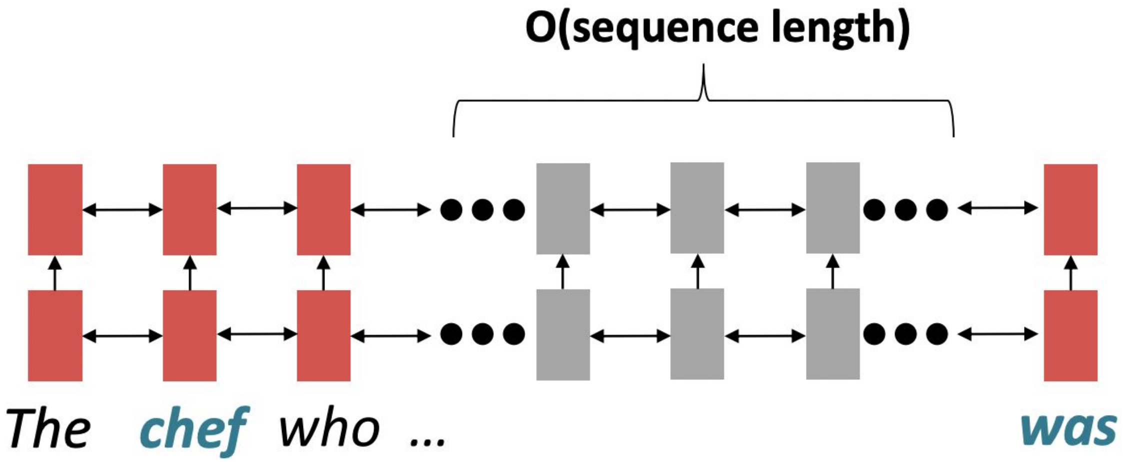


Transformers - Motivation

RNN – Linear Interaction Distance/Non-parallelizable

- RNNs are unrolled “left-to-right”.
- Useful: Nearby words often affect each other’s meanings

- Problem: RNNs take O(sequence length) steps for distant word pairs to interact
- Problem: Linear Order is “baked in”. Not sure that is best.
  - Right-to-left
  - Left-to-right
  - Bi-directional RNNs.

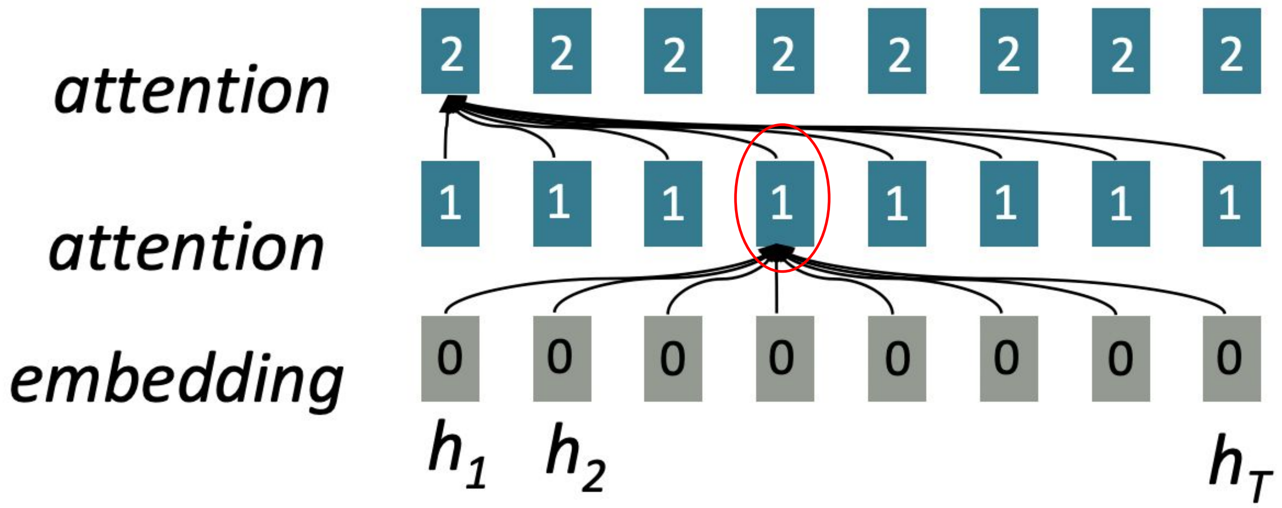


Recurrence to Attention

- Attention treats each word's representation as a query to access and incorporate information from a set of values.
  - For example, Layer 2 each node j computes

$$\sum_{i=1}^{T} \alpha_i w_{ij} h_i, \text{ s.t. } \Sigma_i \alpha_i = 1$$

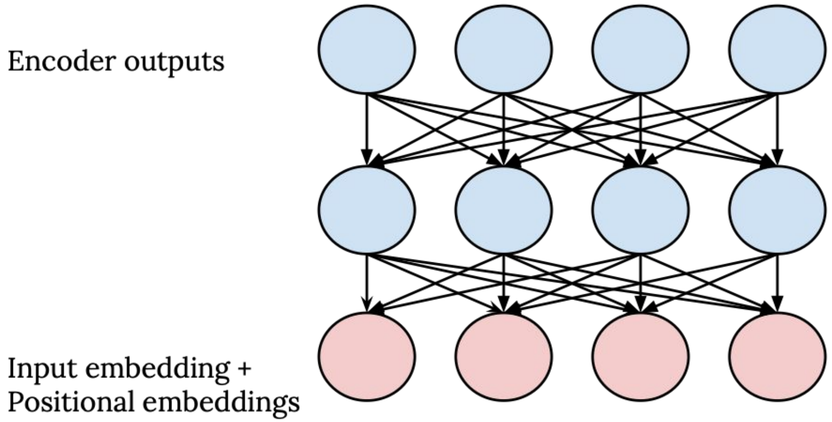
- Max. interaction distance: O(1).



Transformers - Motivation

How can we speed up the encoding process of sequences?

- Remove the recurrent connection (from RNNs)
- Only use attention

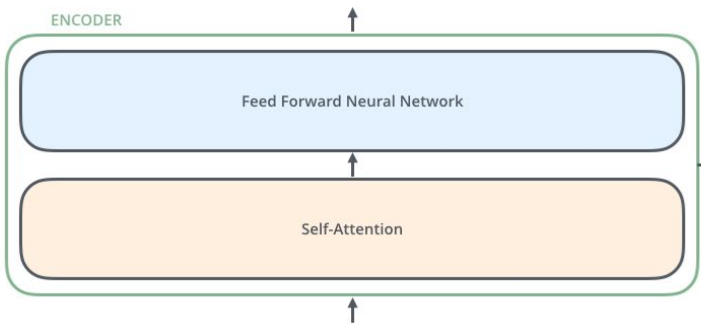


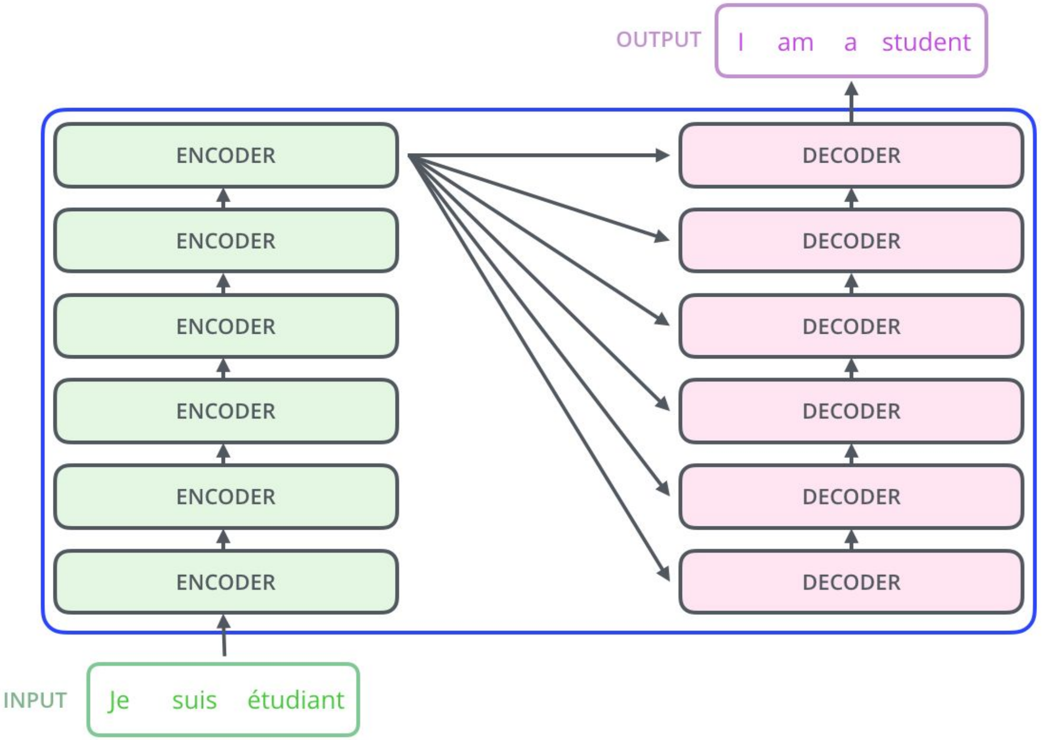
- But No order?
- No nonlinearities. Just weighted average

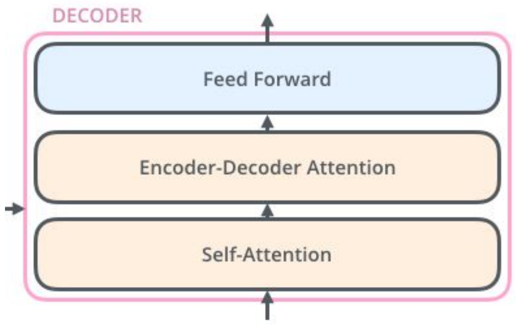
Solution:

- Positional Embeddings (encode positions as vectors)
- Add non-linearities using separate layers
  FFN+BatchNorm

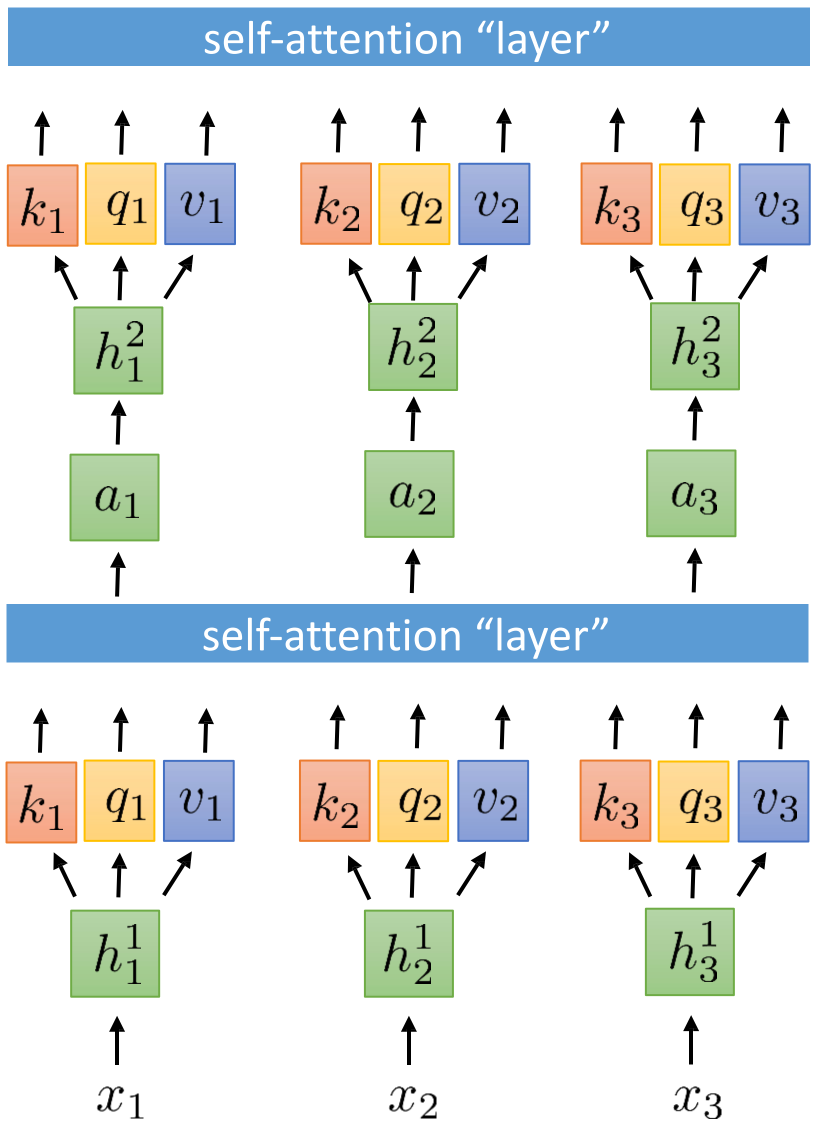
Transformer: Structure







Transformer summary



- Alternate self-attention "layers" with nonlinear position-wise feedforward networks (to get nonlinear transformations)
- Use positional encoding to make the model aware of relative positions of tokens
- Use multi-head attention
- Use masked attention if you want to use the model for decoding.

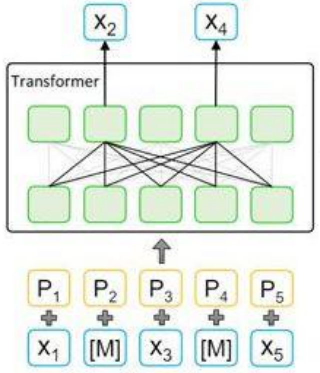
Self-supervised Models (Unsupervised Pretraining)

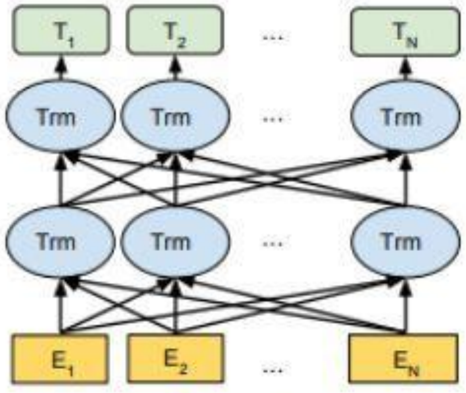
Incorporating context into word embeddings

a watershed idea in NLP

- BERT, 2018: Bidirectional Encoder Representations from Transformers (BERT, 2018)
- GPT

Led to significant improvements on virtually every NLP task.

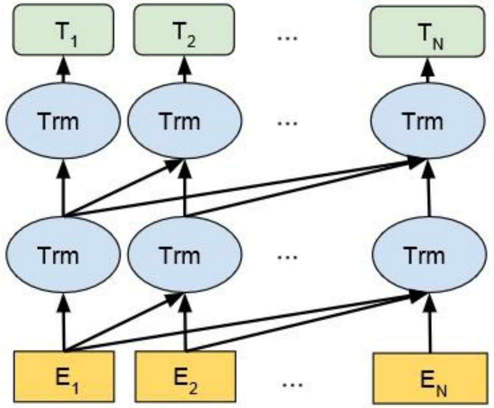




BERT Architecture

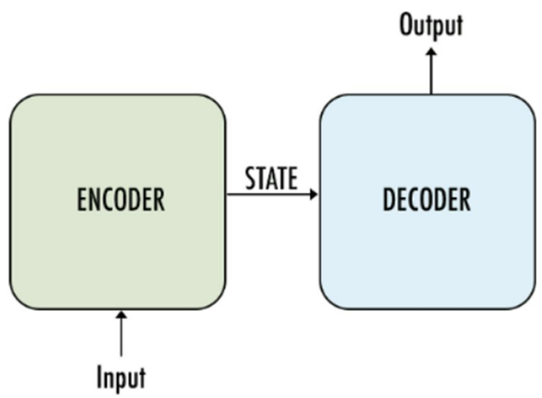
NLP

- Masked language modeling
- Predict the next word
- Next sentence prediction

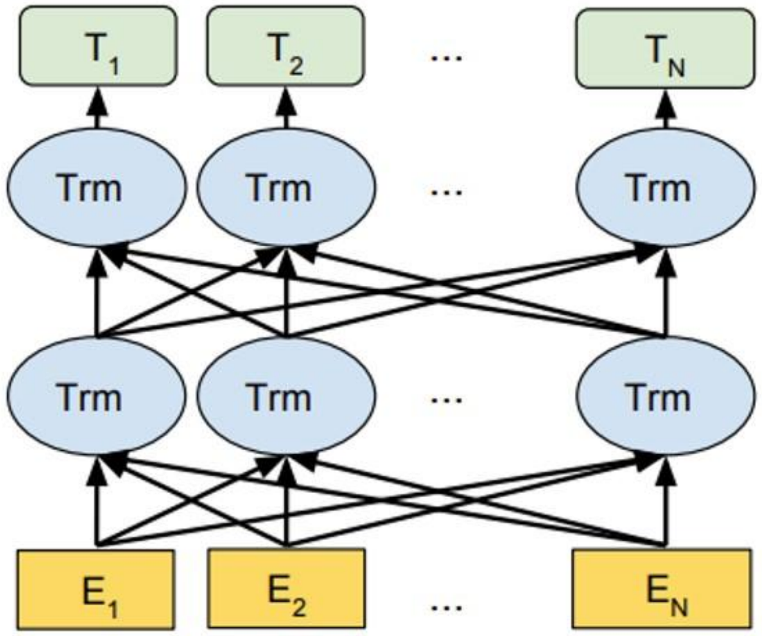


1. Build background knowledge
2. Approximate a form of common sense in AI systems.

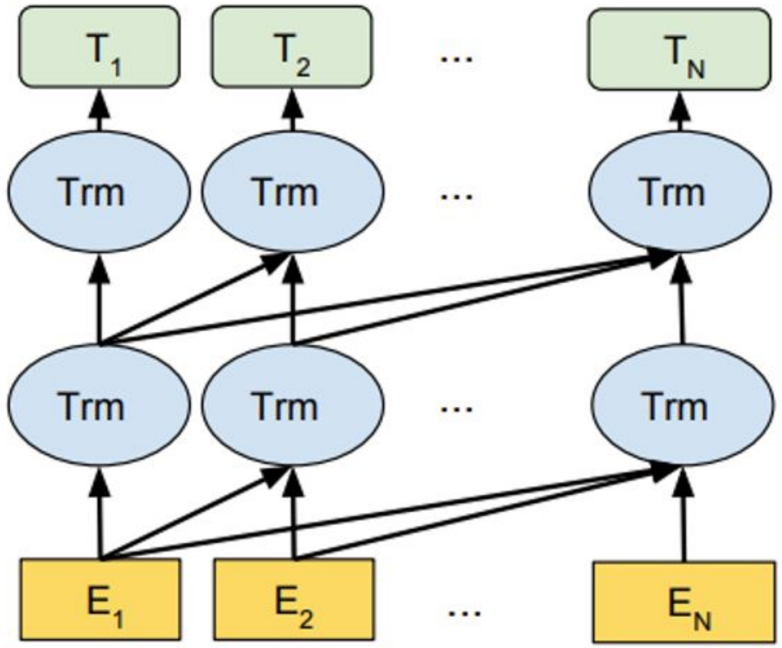
Self-attention encoder decoder



Parametric architectures for sentence denoising: Encoder

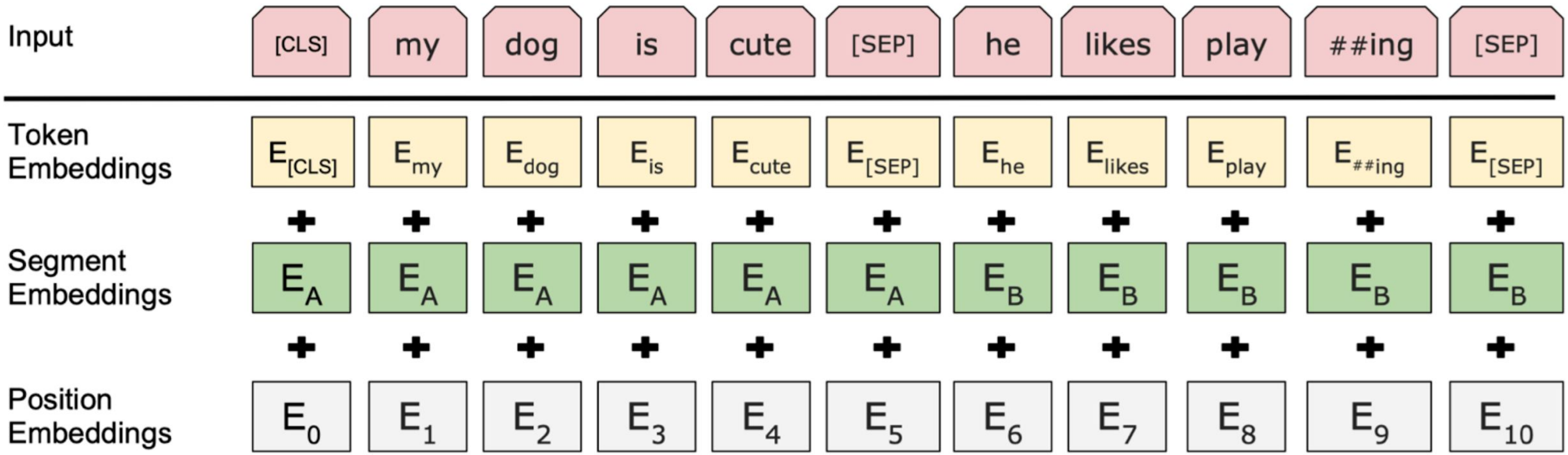


Parametric architectures for text completion: Decoder

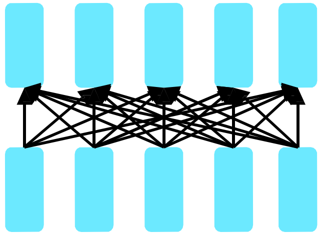


BERT

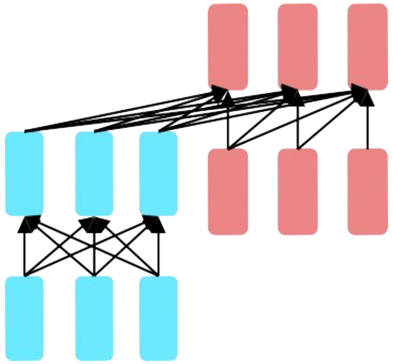
- multi-layer self-attention (Transformer)
- Input: a sentence or a pair of sentences with a separator and subword representation



Large Language Models

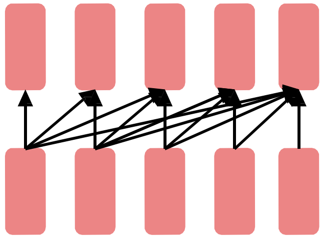


Encoders



Encoder-

Decoders



Decoders

A language model
- Models language
- Assigns probability to a sequence of words

- Encoder only models: BERT, RoBERTa, Electra
- Decoder only models: GPT-n
- Encoder-decoder models: full encoder, autoregressive decoder : T5, BART

Trained by Self-supervised learning on a huge corpus.

Pre-training allows language models to learn robust task-agnostic features

May be followed by

•Supervised fine-tuning for tasks

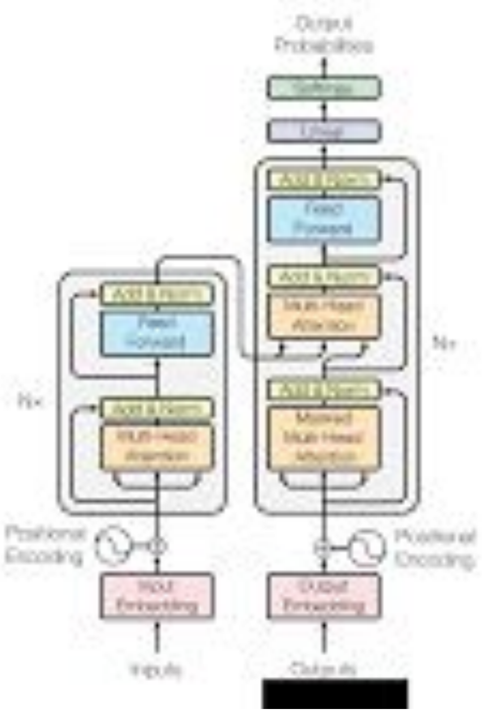
•Reinforcement learning with human feedback

The promise : One single model for many tasks

Three types of LLMs

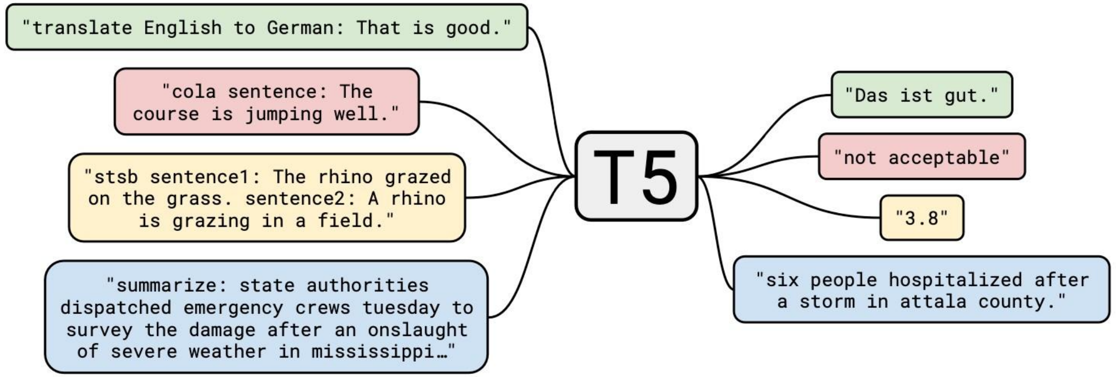
BERT

Encoder

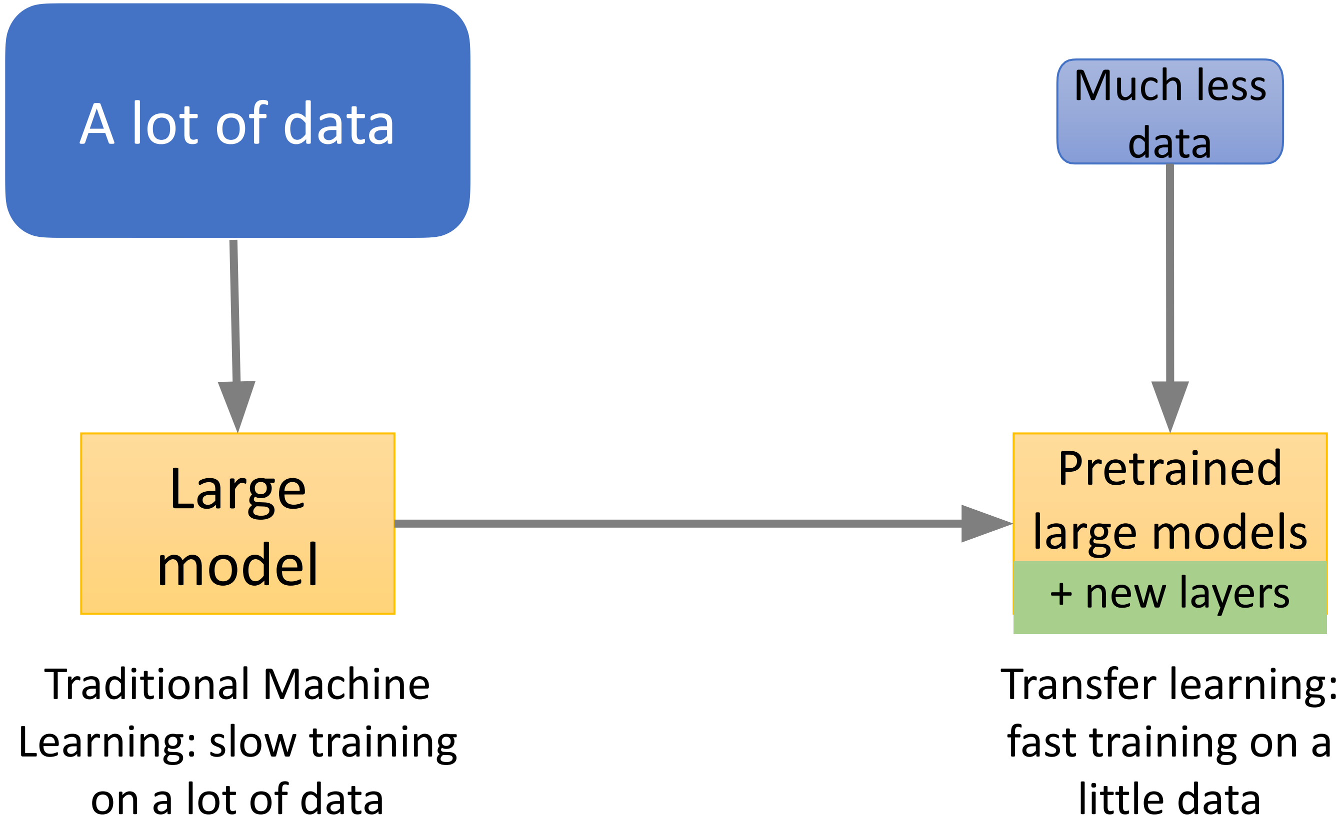


GPT

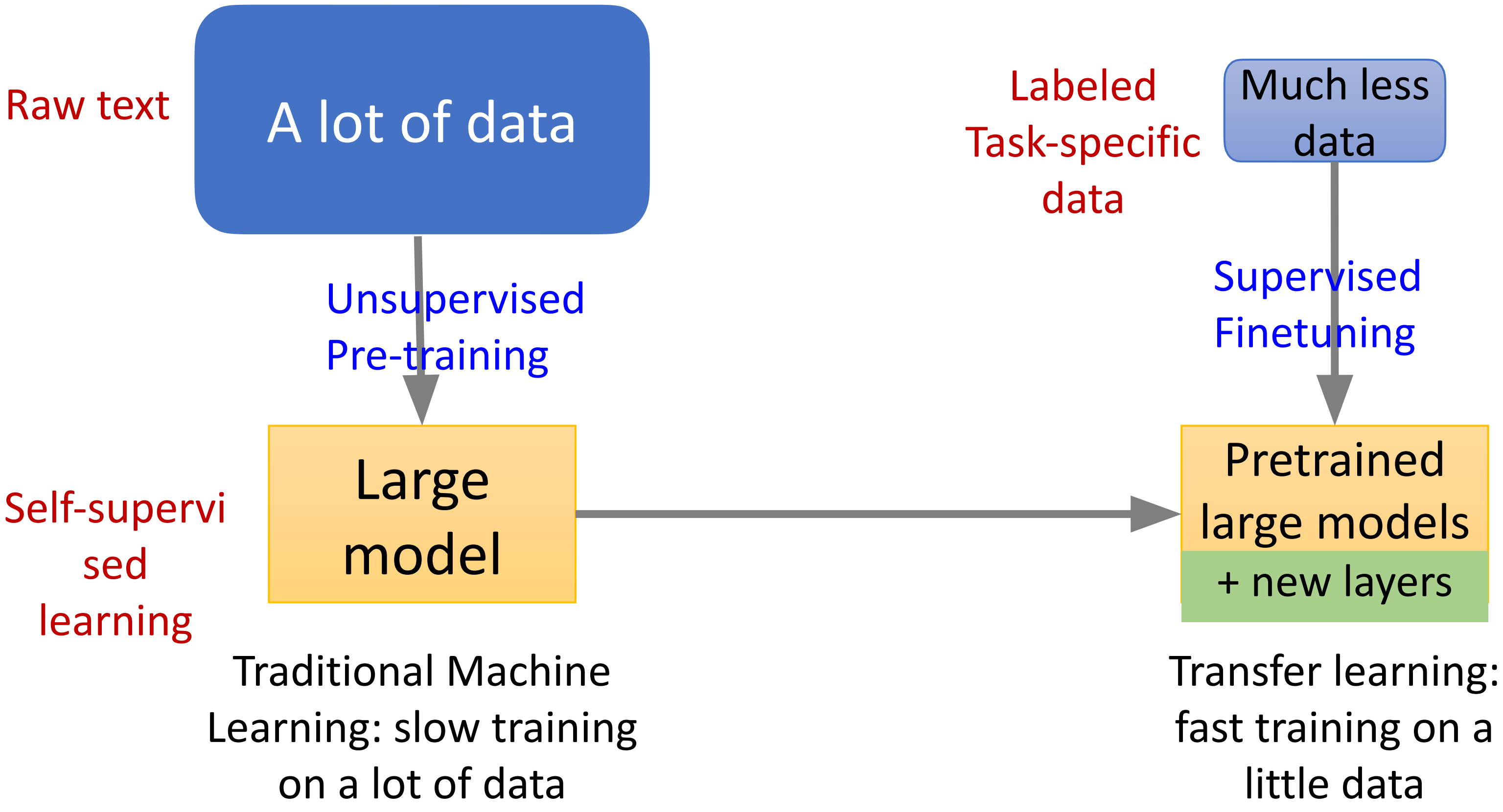
Decoder



Transfer Learning



Transfer Learning in NLP



How are LLMs applied in various tasks/domains?

- Previously
  - By adding task-specific layers on top of LLMs and fine-tuning them on labeled data of such tasks
  - Examples: Text classification, language translation, question-answering

- More recently
  - By prompt engineering/tuning, **without** changing LLM params
  - Design a prompt that elicits the desired output from the LLM
  - Example (English->French translation):

Translate the following English sentence [sentence input] *to French:* [model output]

Slide credit: Yuxiong Wang