Attention

Sequential data

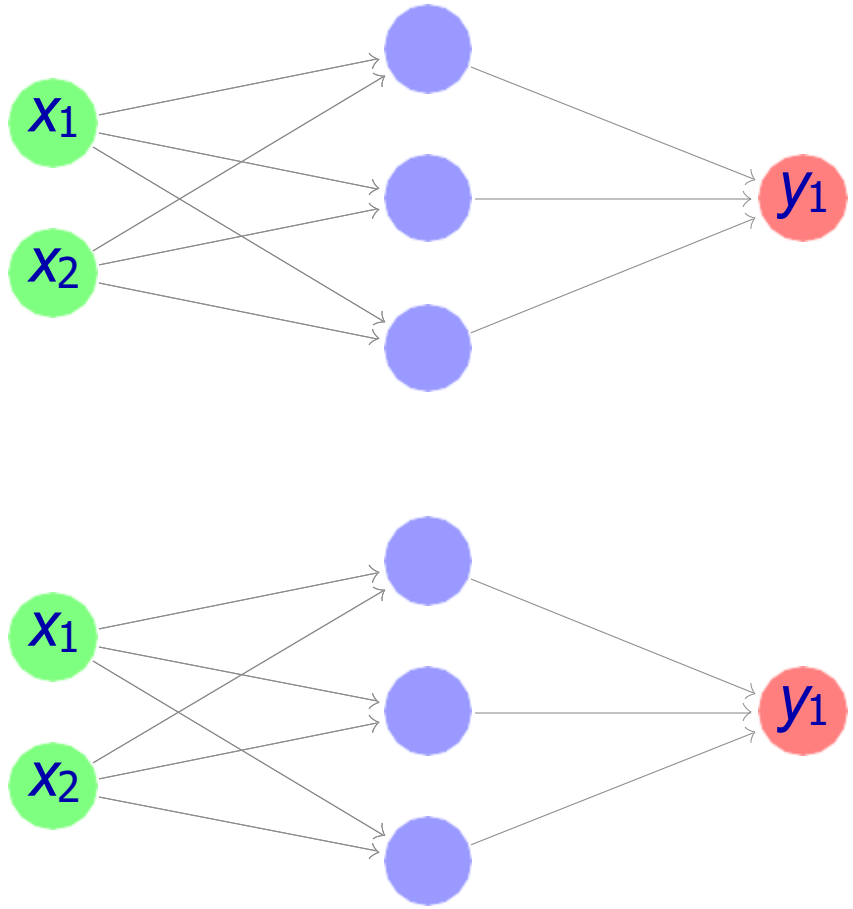
Recurrent neural networks (RNNs) are often used for handling sequential data.

They introduced first in 1986 (Rumelhart et al 1986).

Sequential data usually involves variable length inputs.

Parameter sharing

Parameter sharing makes it possible to extend and apply the model to examples of different lengths and generalize across them.



Recurrent neural network

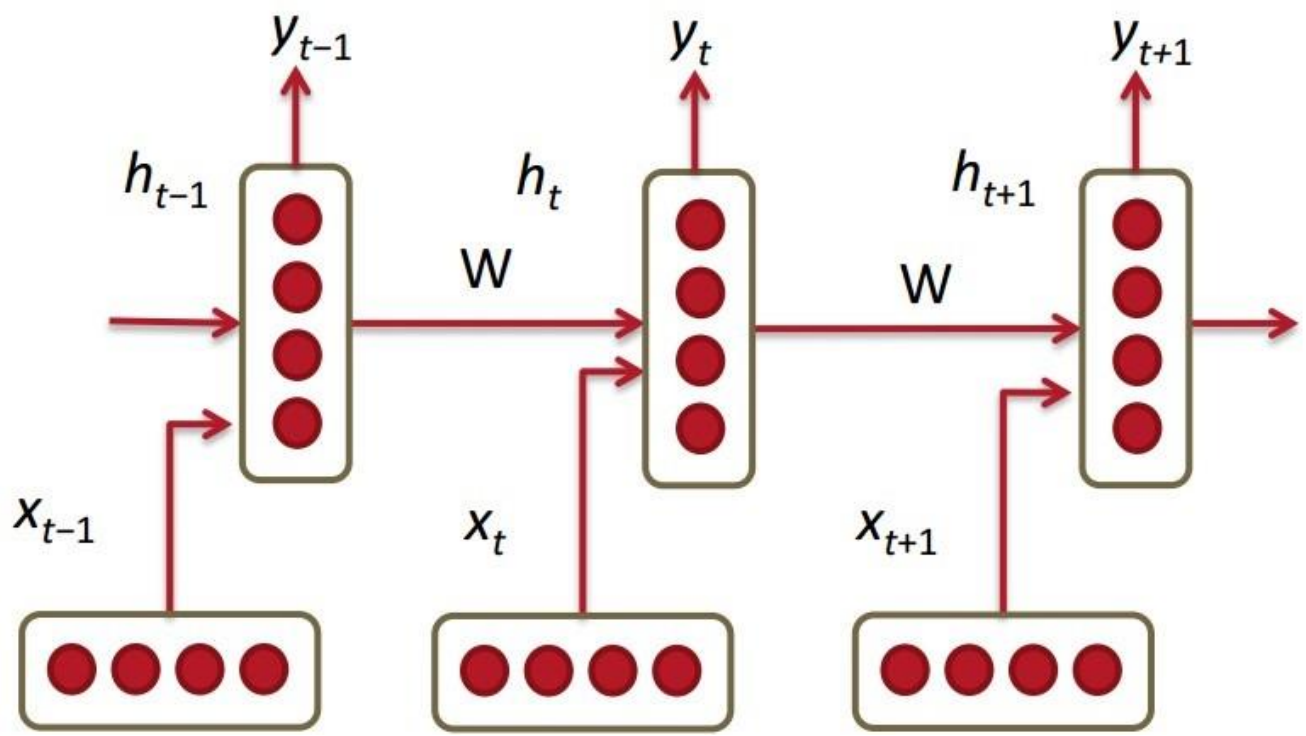
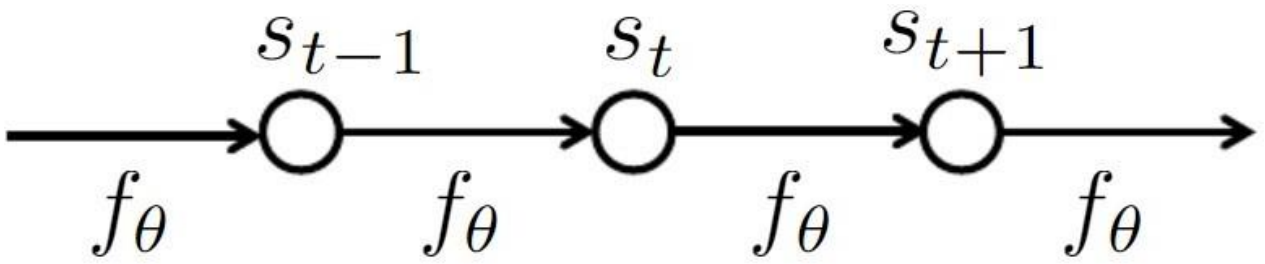


Figure: Richard Socher

Dynamic systems

The classical form of a dynamical system:

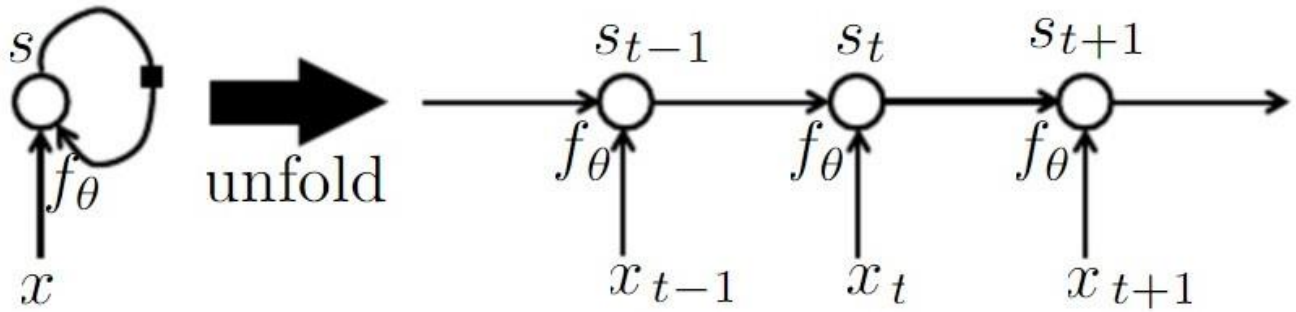
$$s_t = f_\theta(s_{t-1})$$



Dynamic systems

Now consider a dynamic system with an external signal x

$$s_t = f_\theta(s_{t-1}, x_t)$$



The state contains information about the whole past sequence.

$$s_t = g_t(x_t, x_{t-1}, x_{t-s}, \ldots, x_2, x_1)$$

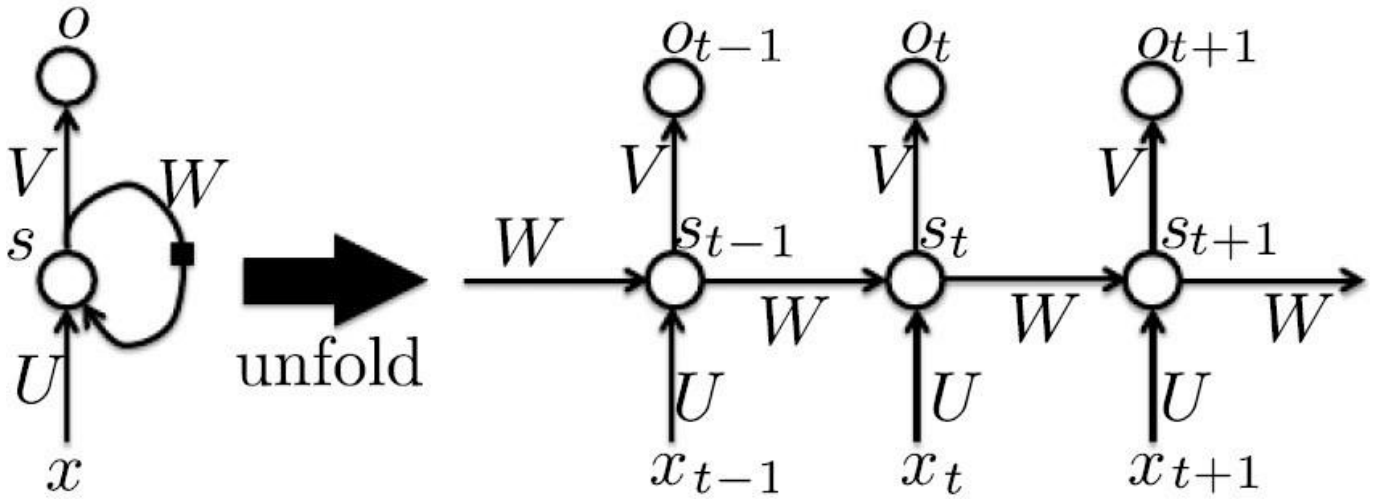
Parameter sharing

We can think of s_t as a summary of the past sequence of inputs up to t.

If we define a different function g_t for each possible sequence length, we would not get any generalization.

If the same parameters are used for any sequence length allowing much better generalization properties.

Recurrent Neural Networks



$$\mathbf{a}_t = \mathbf{b} + W\mathbf{s}_{t-1} + U\mathbf{x}_t$$

$$\mathbf{s}_t = \tanh(\mathbf{a}_t)$$

$$\mathbf{o}_t = \mathbf{c} + V\mathbf{s}_t$$

$$\mathbf{p}_t = \mathrm{softmax}(\mathbf{o}_t)$$

Computing the Gradient in a Recurrent Neural Network

Using the generalized back-propagation algorithm one can obtain the so-called Back-Propagation Through Time (BPTT) algorithm.

Exploding or Vanishing Product of Jacobians

In recurrent nets (also in very deep nets), the final output is the composition of a large number of non-linear transformations.

Even if each of these non-linear transformations is smooth. Their composition might not be.

The derivatives through the whole composition will tend to be either very small or very large.

LSTM, Gated RNN

The Long-Short-Term-Memory (LSTM) algorithm was proposed in 1997 (Hochreiter and Schmidhuber, 1997).

Several variants of the LSTM are found in the literature: Hochreiter and Schmidhuber 1997

Graves, 2012 Graves et al., 2013

Sutskever et al., 2014

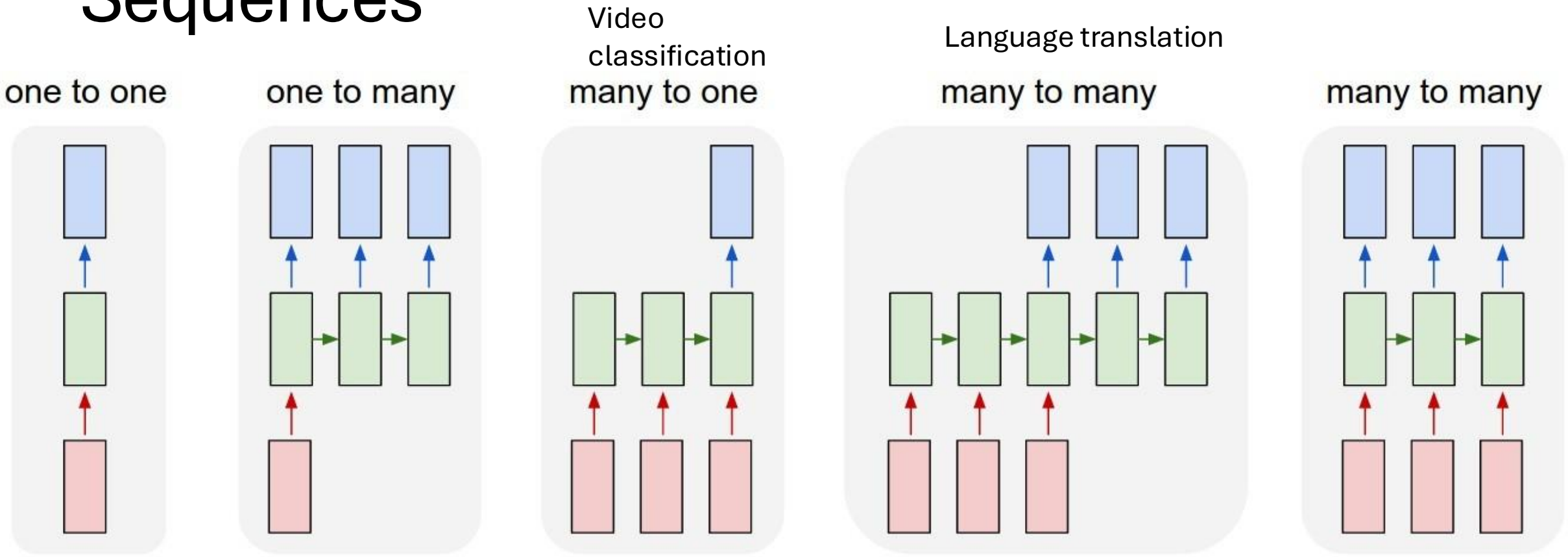
Recent work on gated RNNs, Gated Recurrent Units (GRU) was proposed in 2014

Cho et al., 2014

Chung et al., 2014, 2015 Jozefowicz et al., 2015

Chrupala et al., 2015

Recurrent Neural Networks: Process Sequences

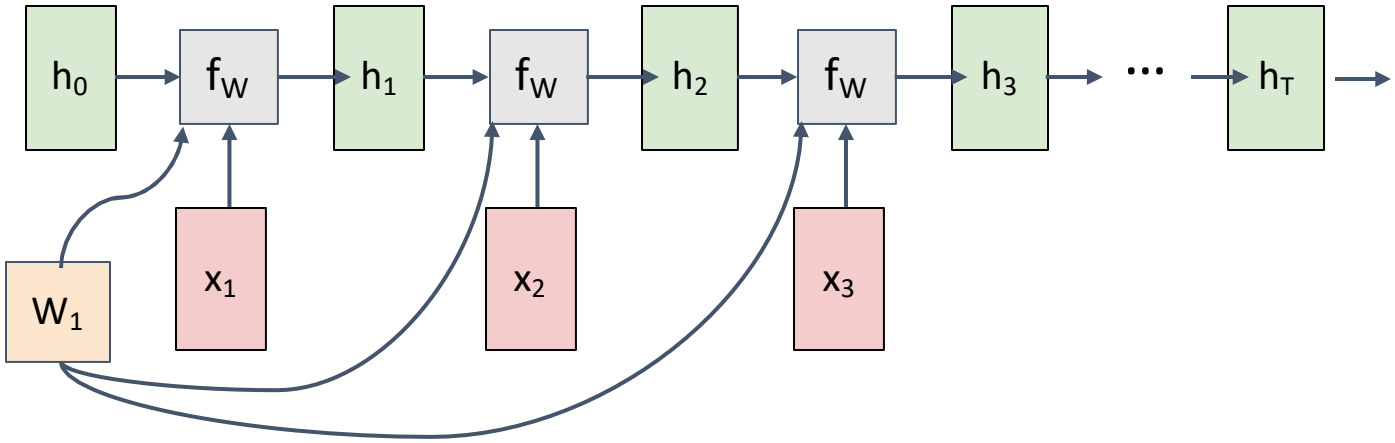


Caption

e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

Sequence to Sequence (seq2seq) (Many to one) + (One to many)

Many to one: Encode input sequence in a single vector

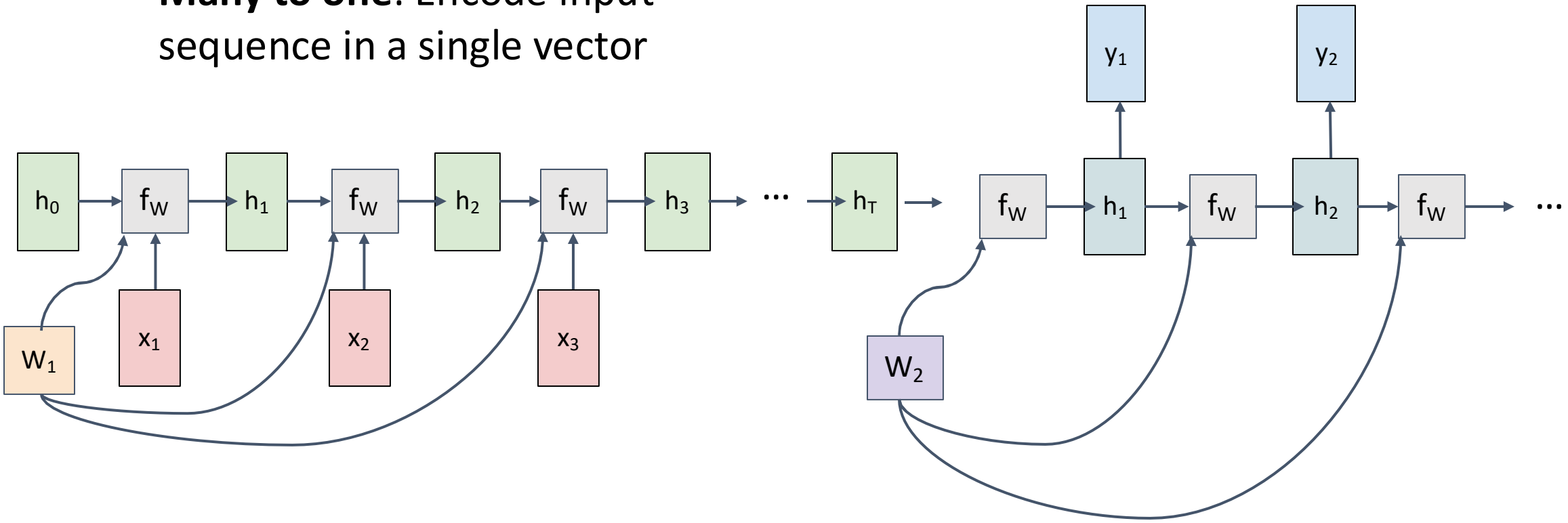


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence to Sequence (seq2seq) (Many to one) + (One to many)

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector



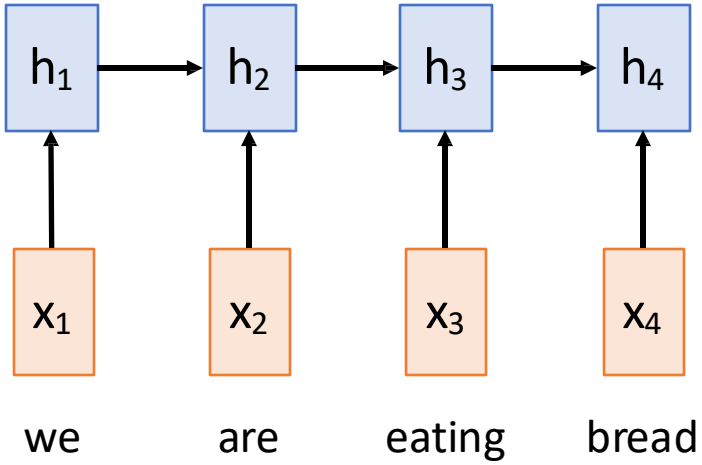
Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

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



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

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

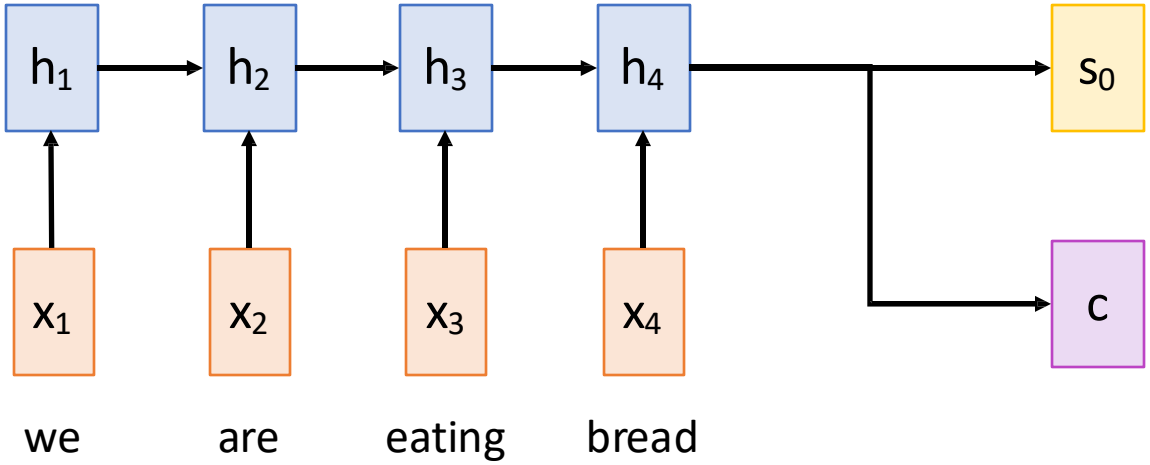
Output: Sequence $y_1, \ldots, y_{T'}$

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

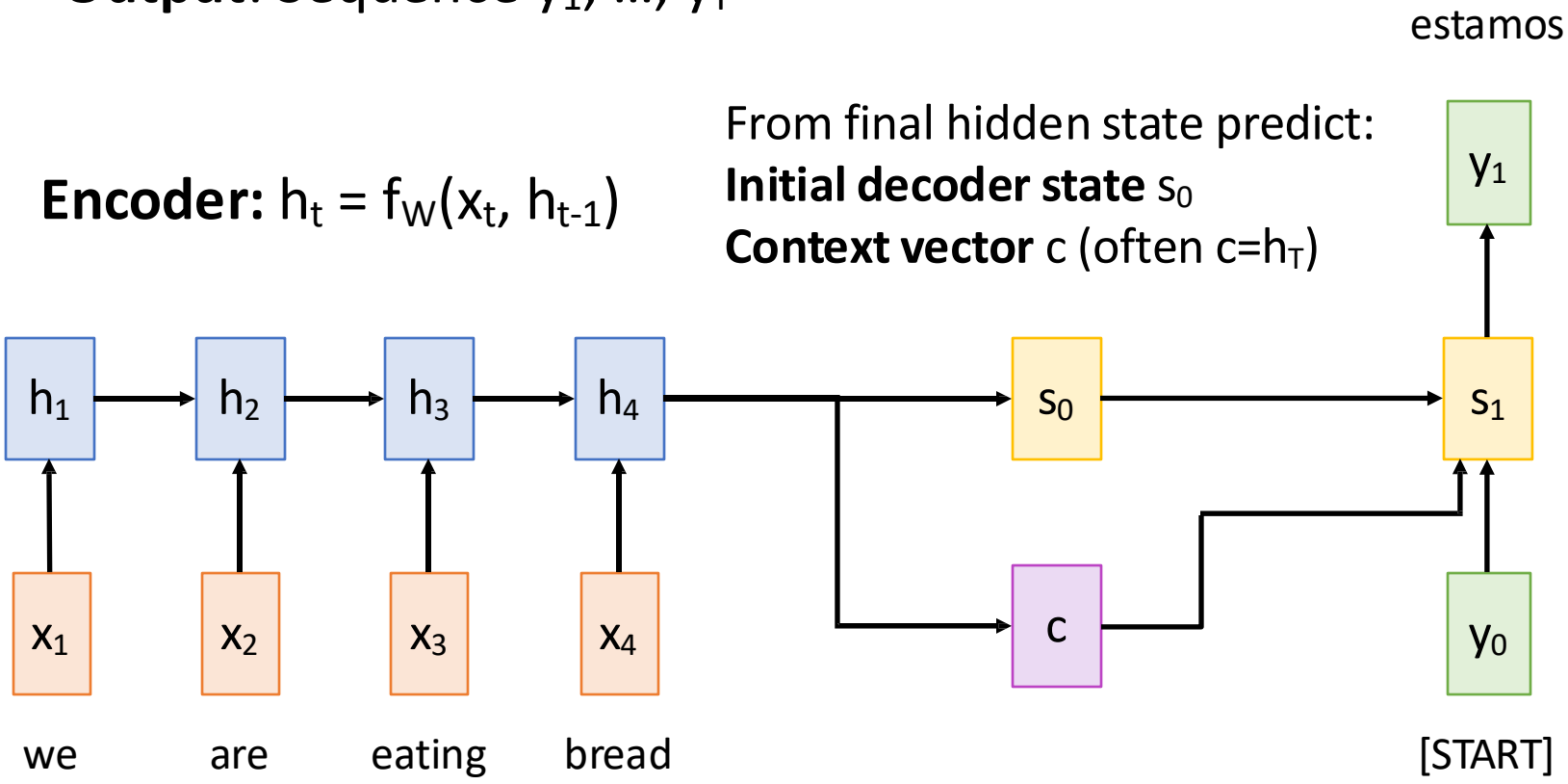
Input: Sequence $x_1, \ldots x_T$

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

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

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

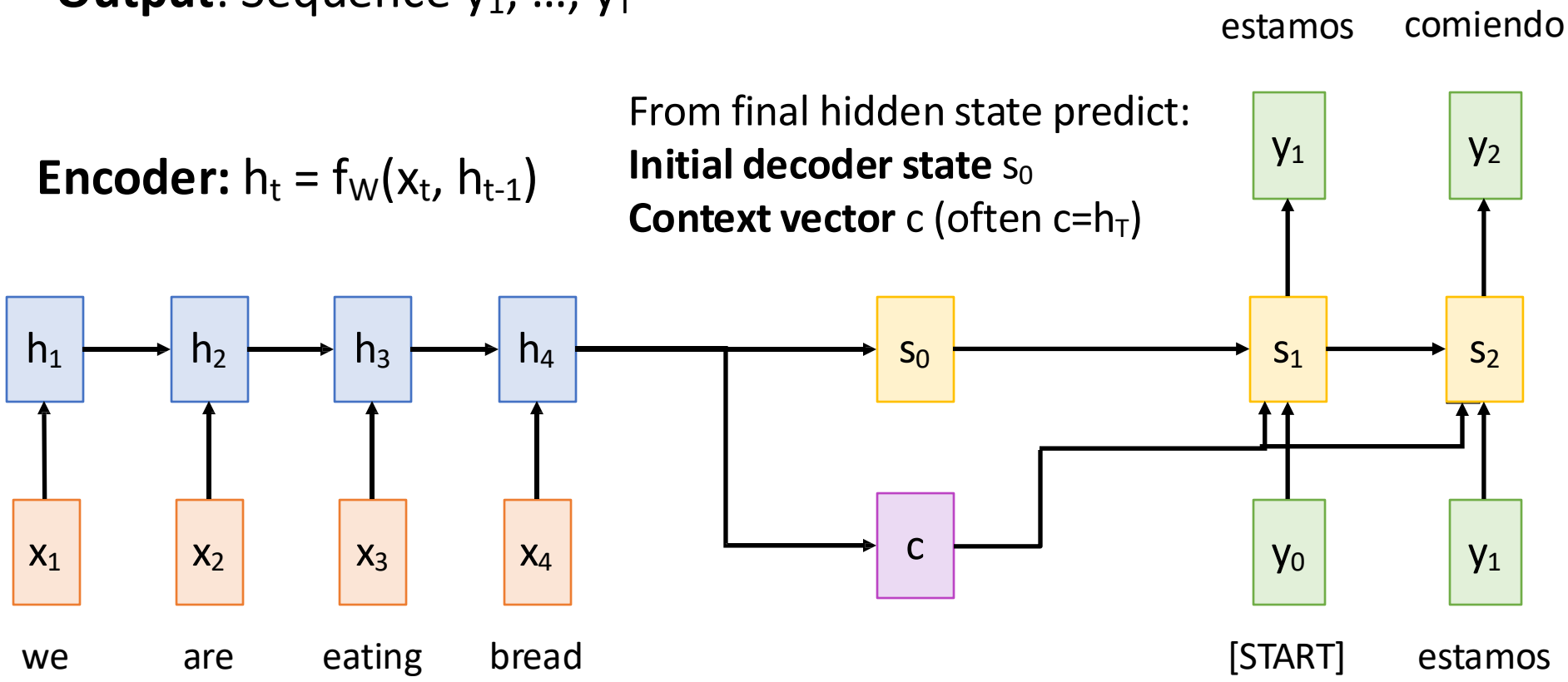
Input: Sequence $x_1, \ldots x_T$

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

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

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

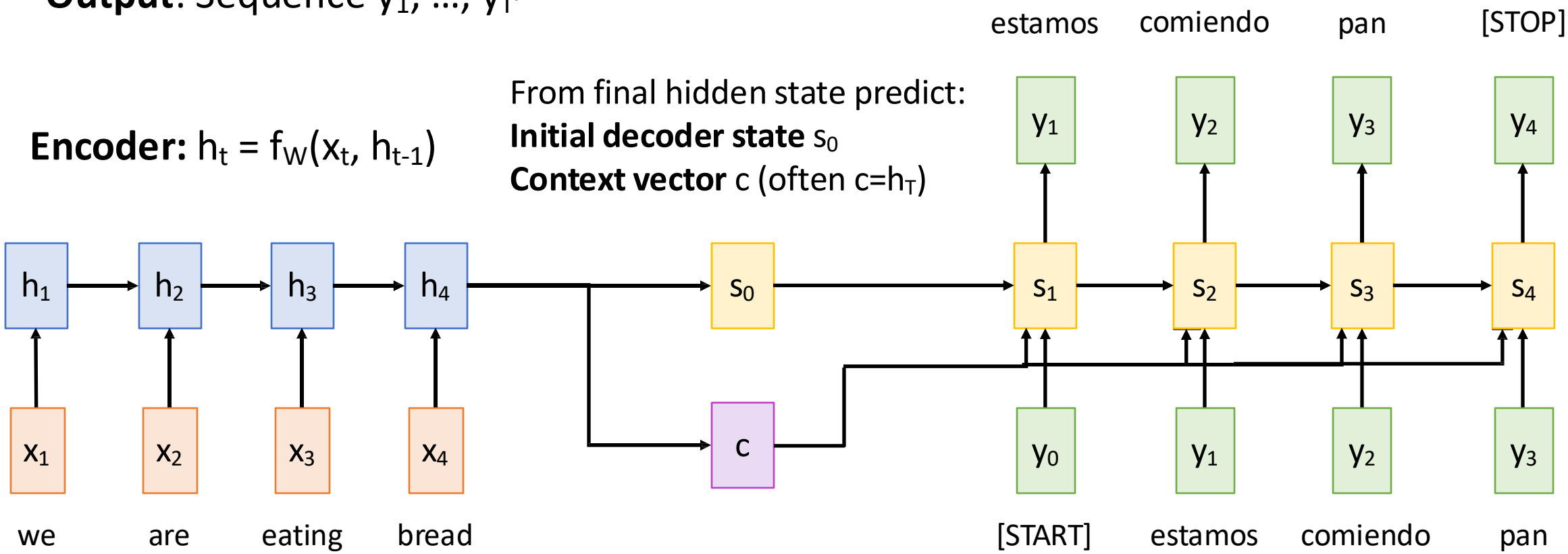
Input: Sequence $x_1, \ldots x_T$

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

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

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

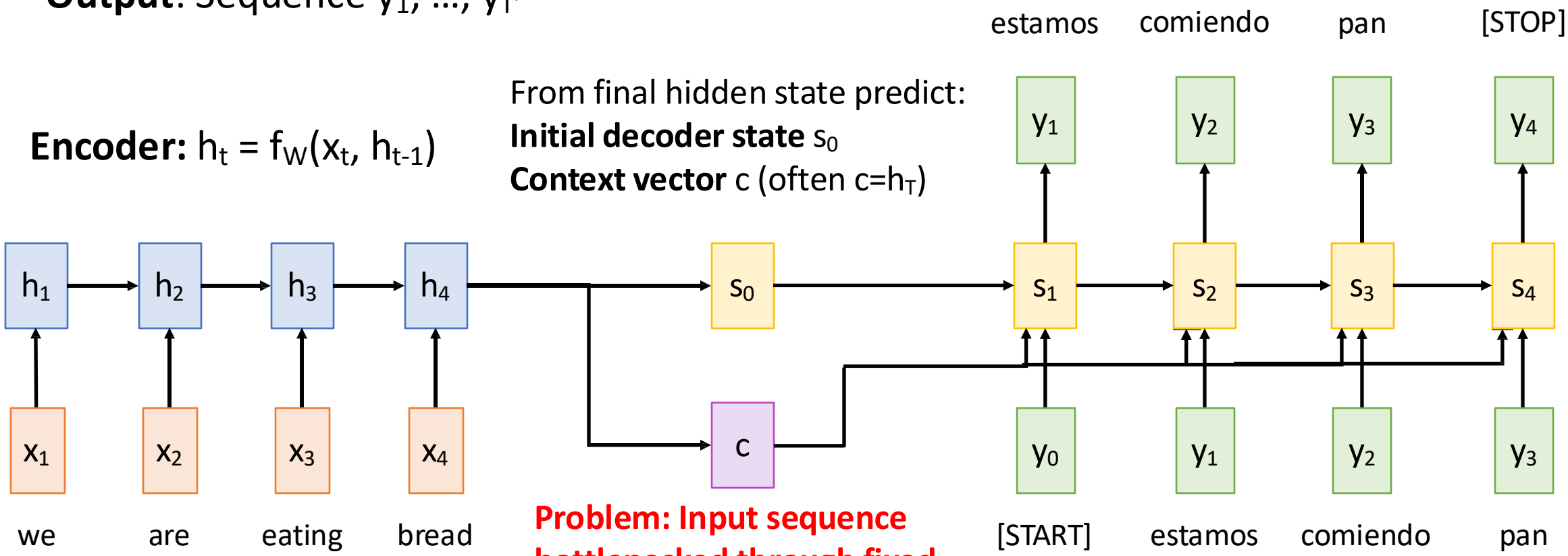
Input: Sequence $x_1, \ldots x_T$

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

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

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?

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

Sequence-to-Sequence with RNNs

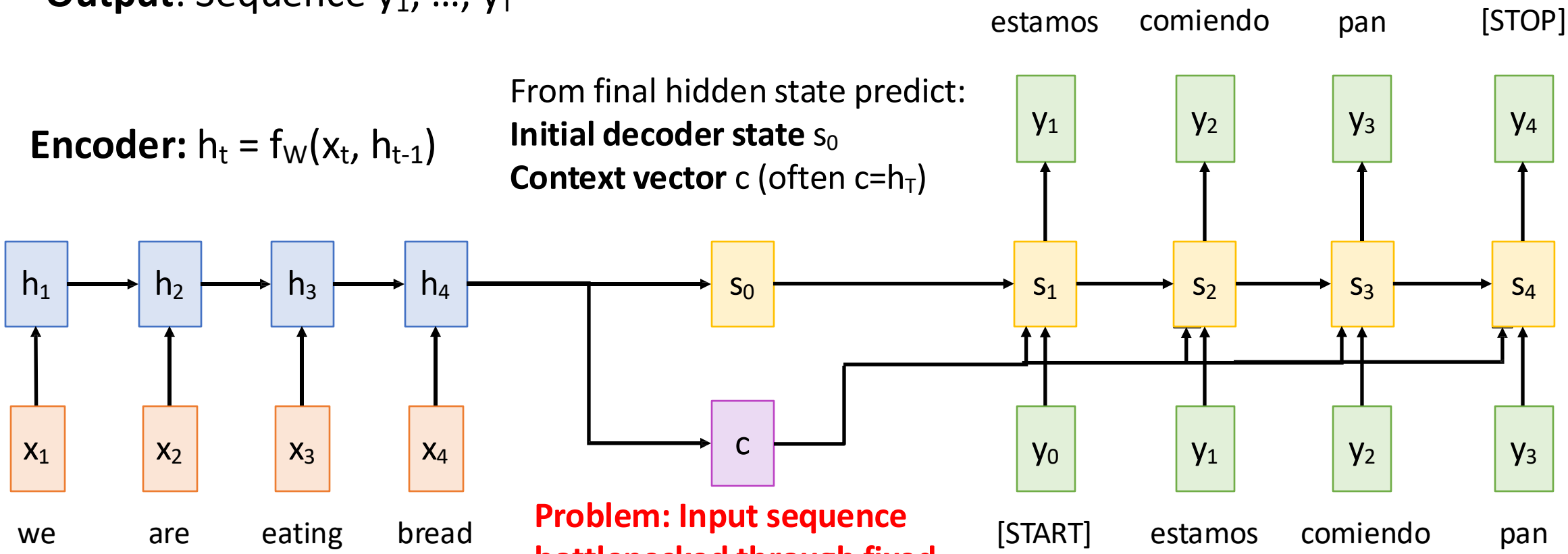
Input: Sequence $x_1, \ldots x_T$

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

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

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", NeurIPS 2014

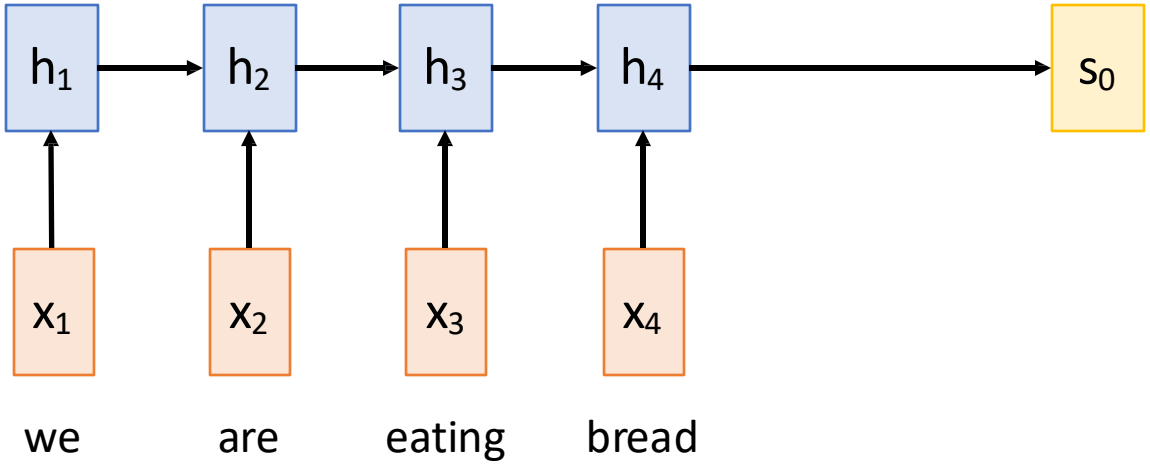
Sequence-to-Sequence with RNNs and Attention

Input: Sequence $x_1, \ldots x_T$

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

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

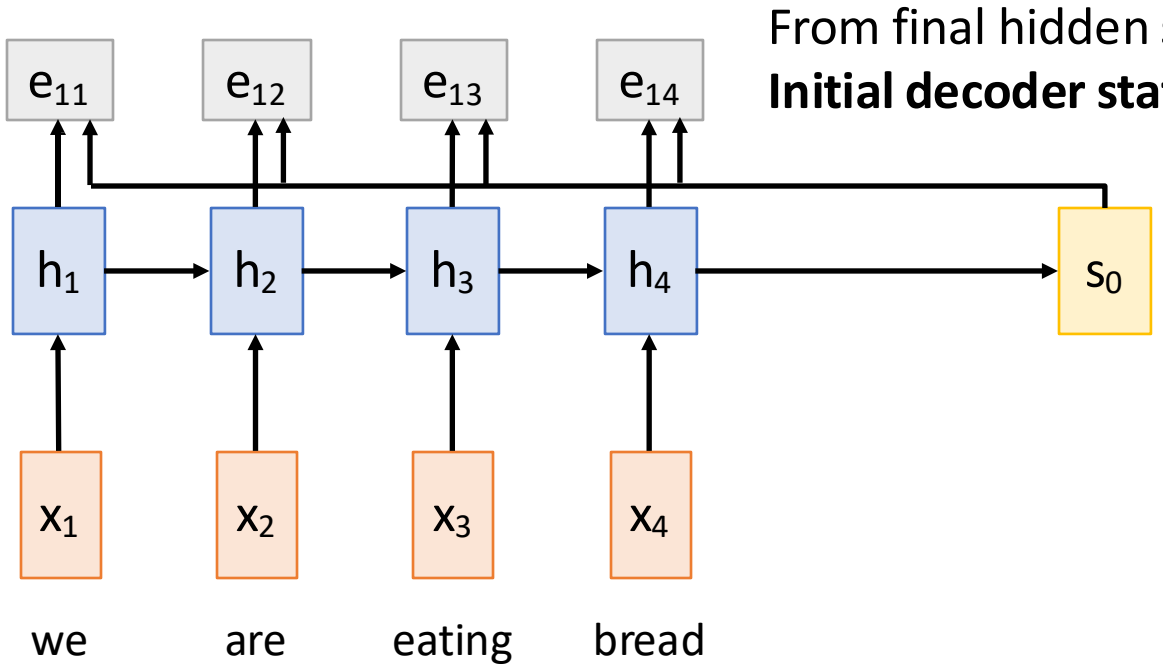
From final hidden state:
Initial decoder state s_0



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

Sequence-to-Sequence with RNNs and Attention

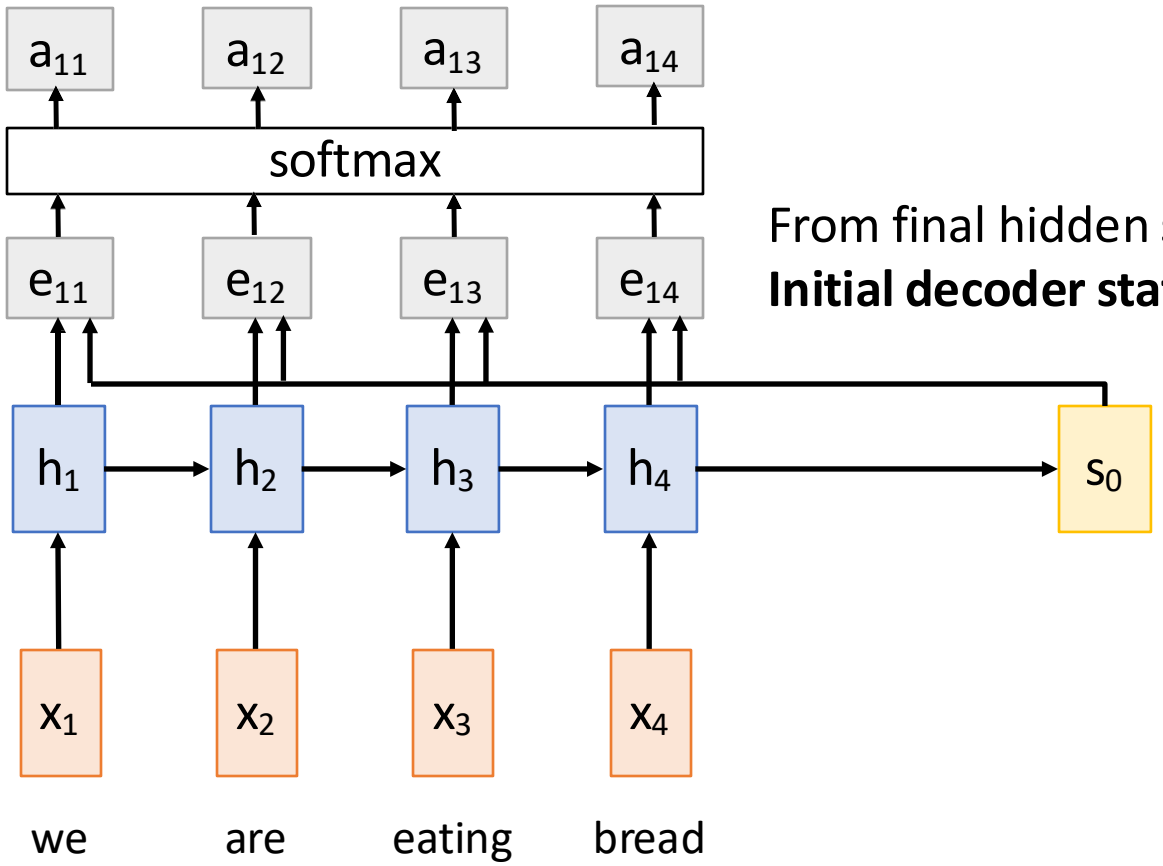
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)



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



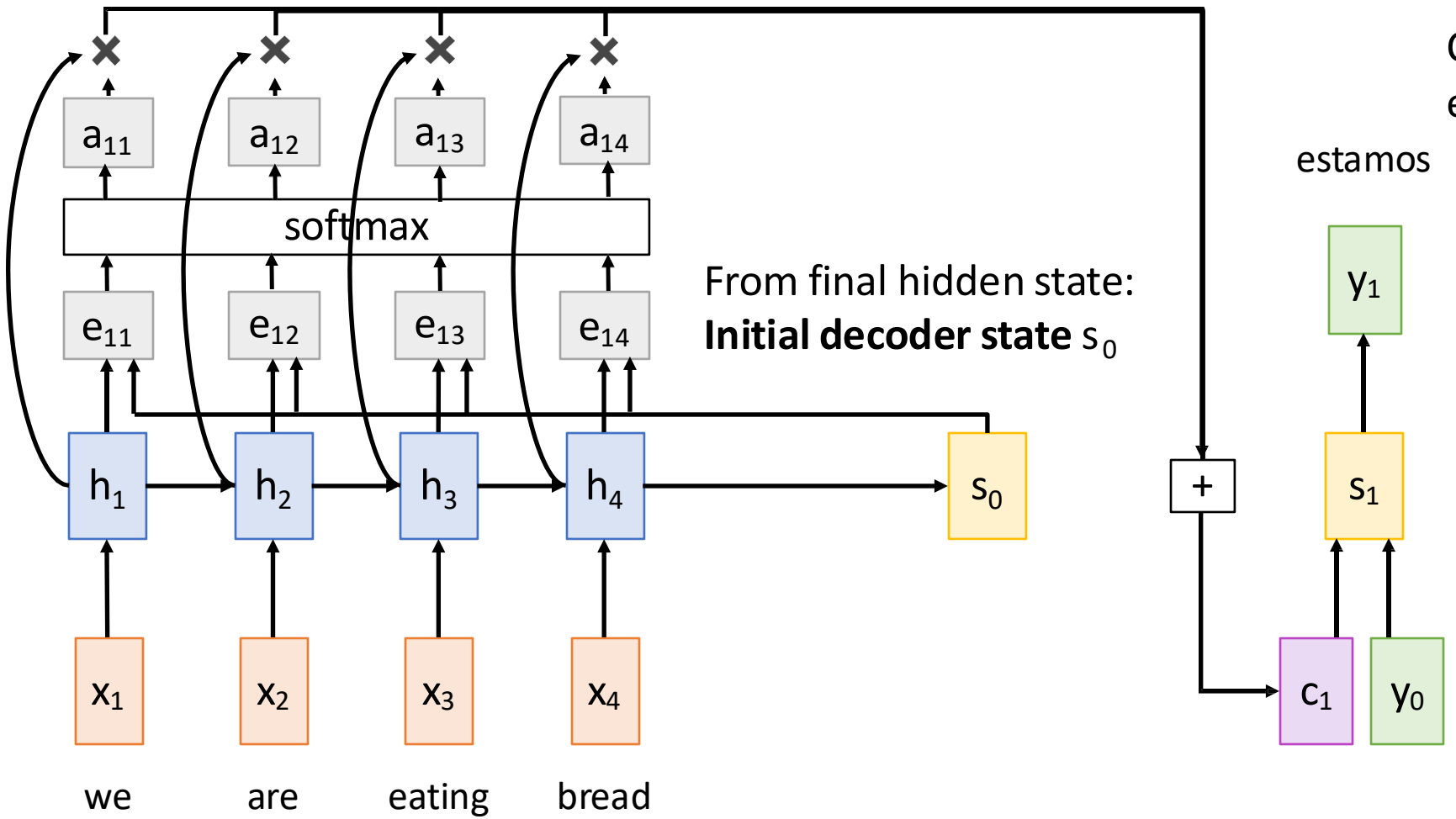
From final hidden state:
Initial decoder state s_0

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

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)

Normalize alignment scores to get **attention weights**

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

Compute context vector as linear combination of hidden states

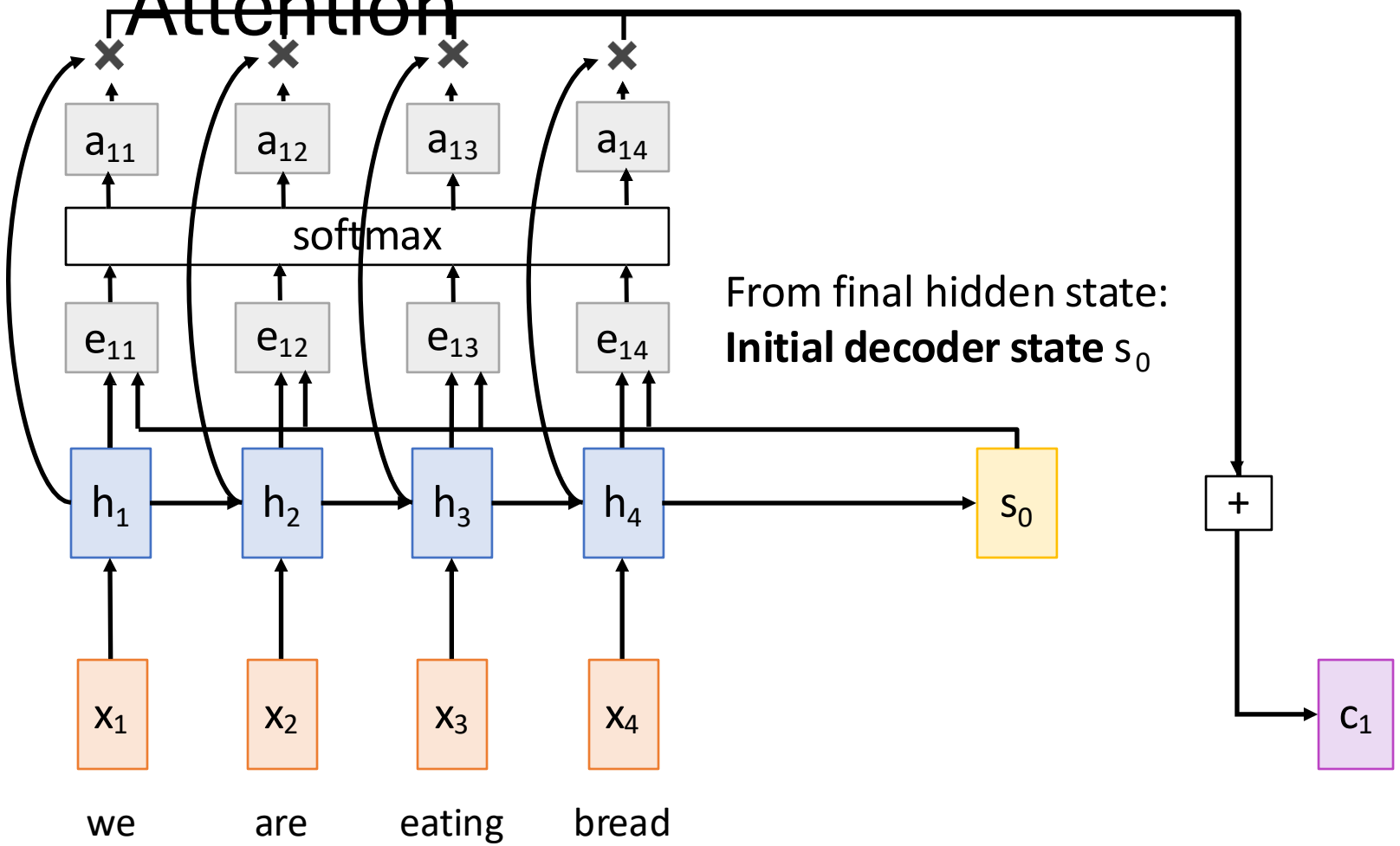
$c_t = \sum_i a_{t,i} h_i$

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

This is all differentiable! Do not supervise 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



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**

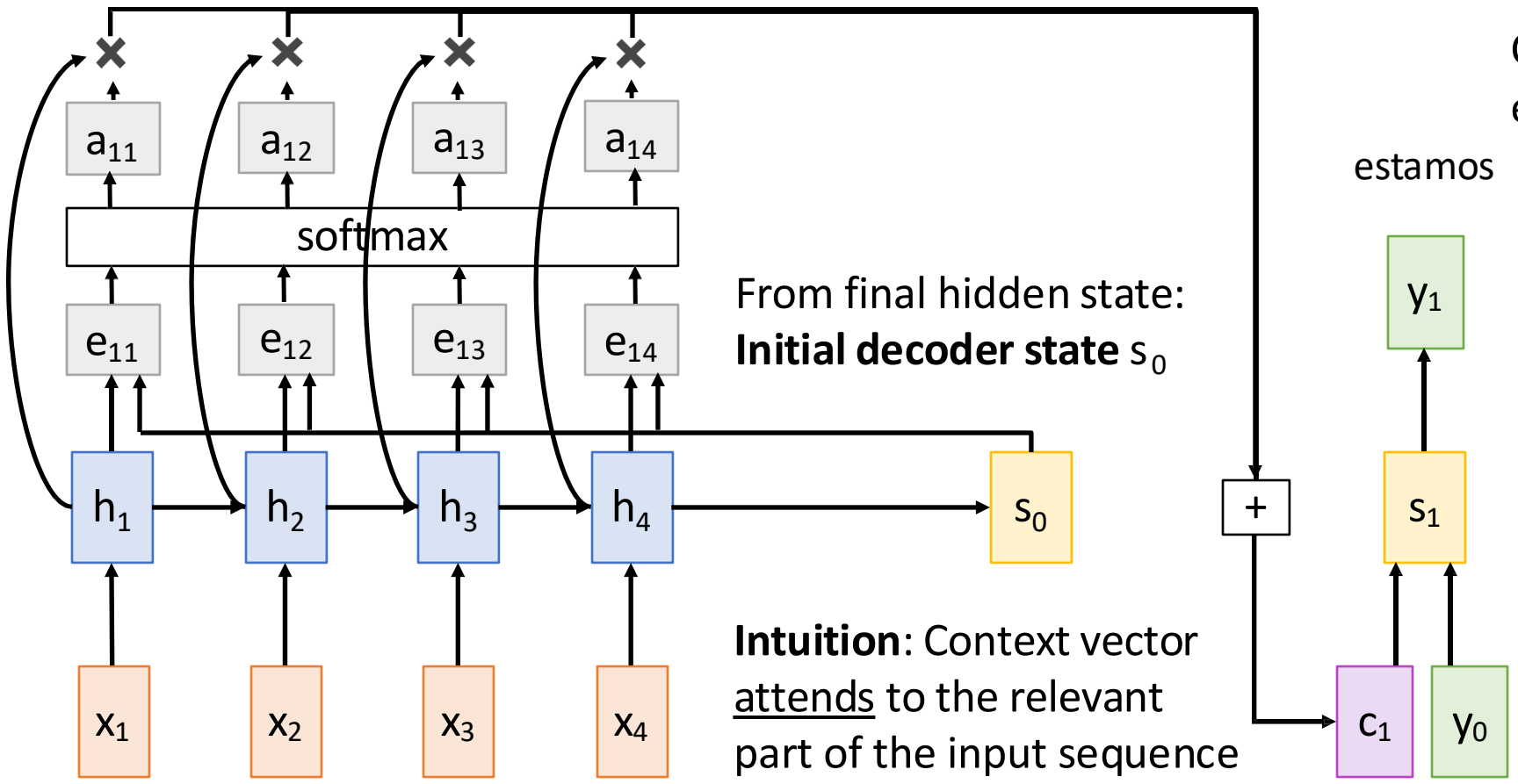
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 0$

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

Intuition: Context vector attends 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

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

Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

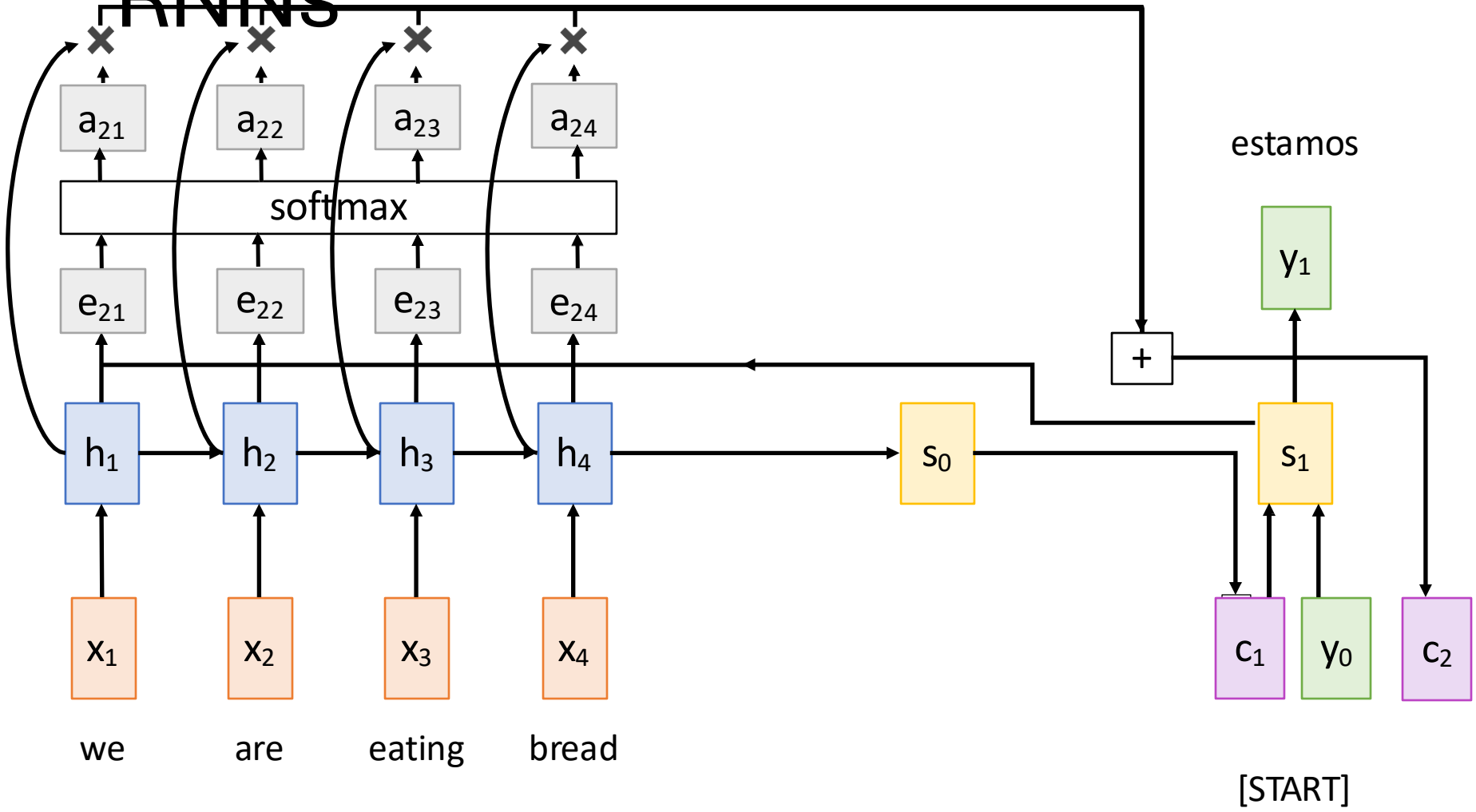
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! Do not supervise 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

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

Attention

- The basic idea behind the attention mechanism is directing the focus on important factors when processing data.

Attention

- The basic idea behind the attention mechanism is directing the focus on important factors when processing data.
- Attention is a fancy name for weighted average.

Attention

- The basic idea behind the attention mechanism is directing the focus on important factors when processing data.

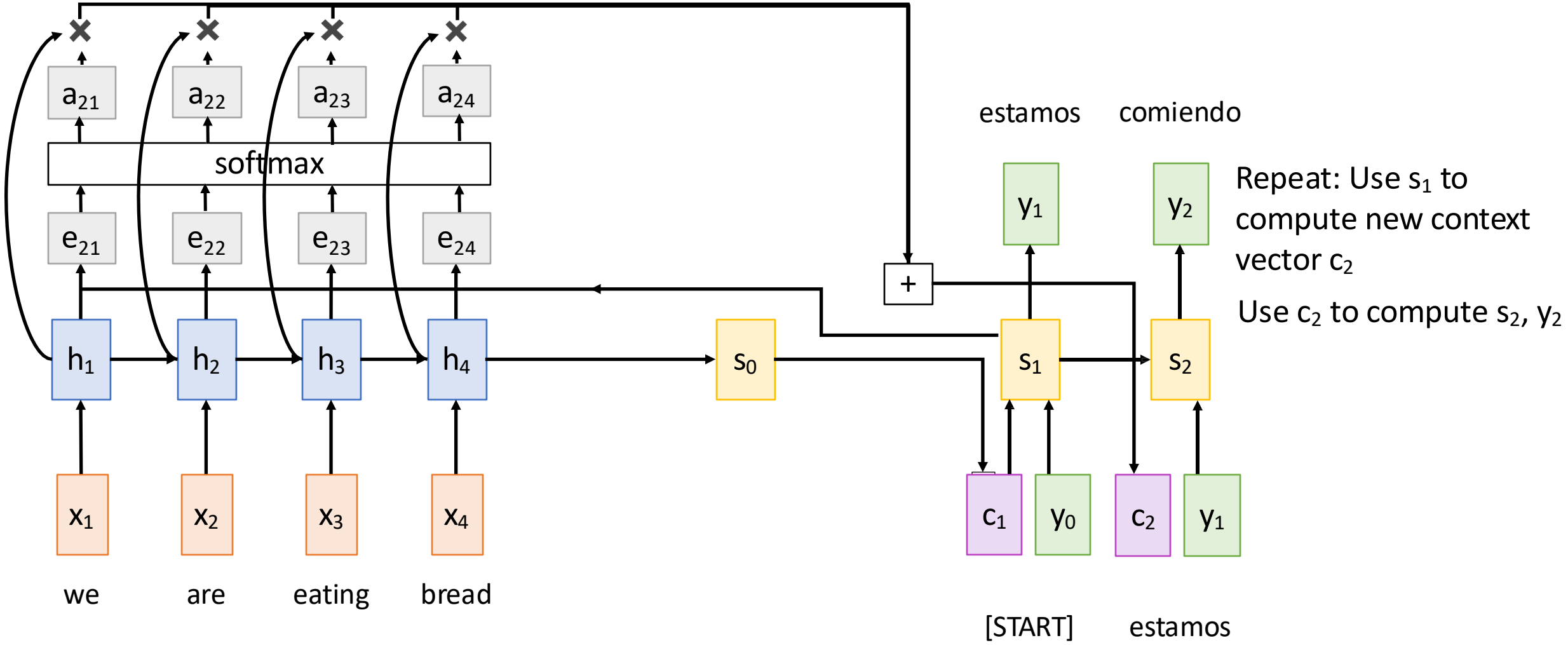
- Attention is a fancy name for weighted average.

- Bahdanau et al., 2014

and

- Luong et al., 2015

Sequence-to-Sequence with RNNs and Attention

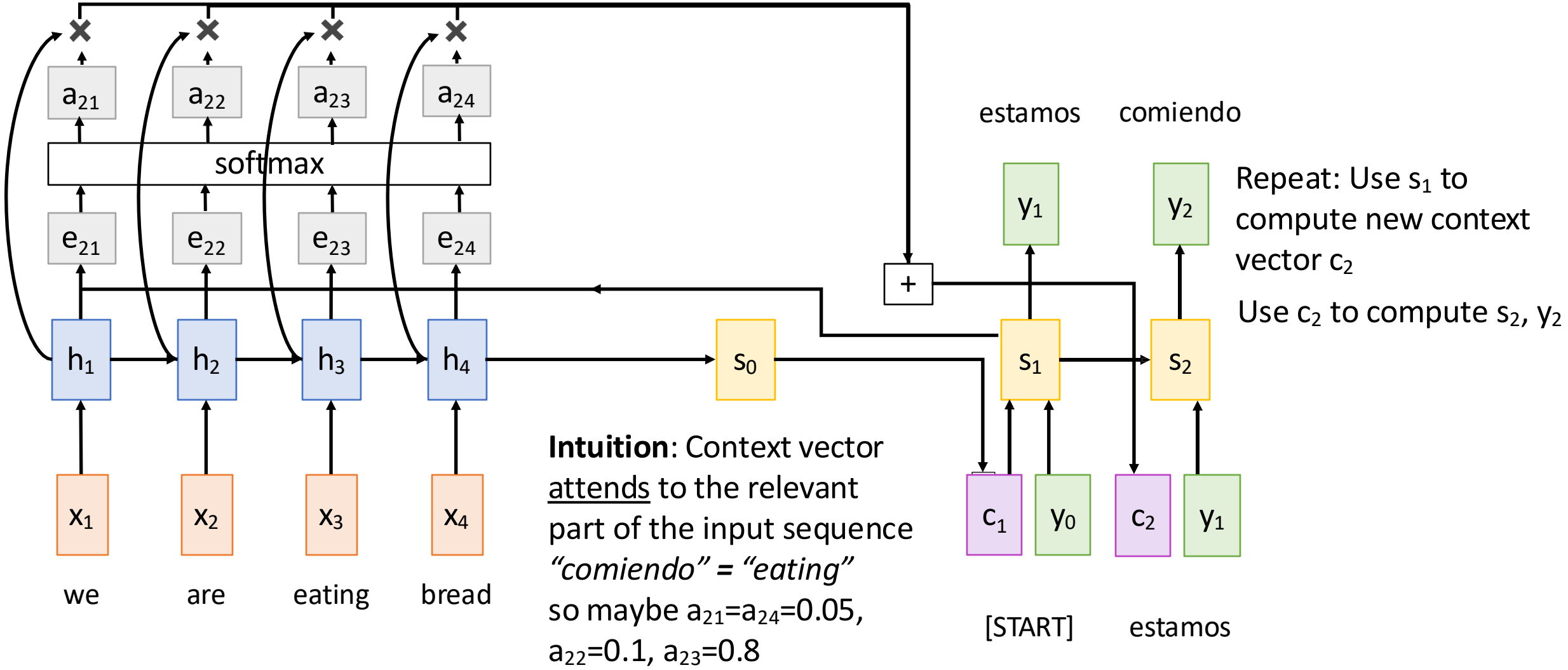


Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

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

Sequence-to-Sequence with RNNs and Attention



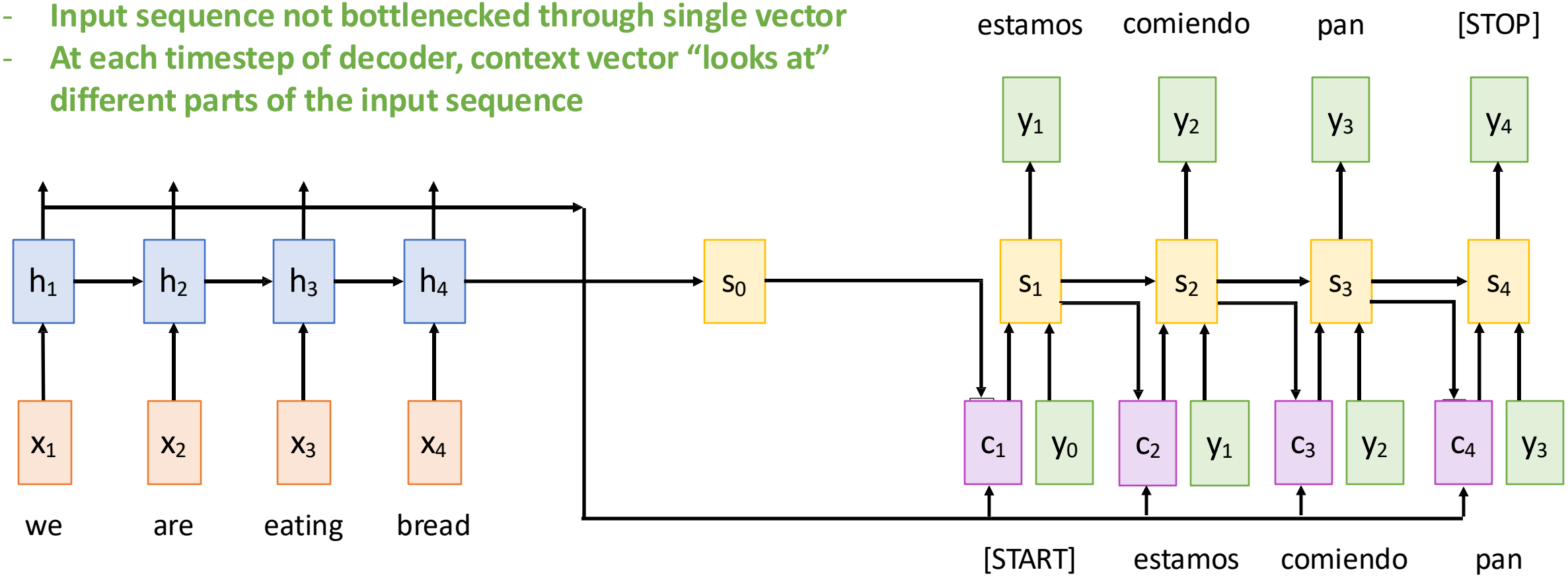
Intuition: Context vector attends to the relevant part of the input sequence
"comiendo" = "eating"
so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

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

Sequence-to-Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

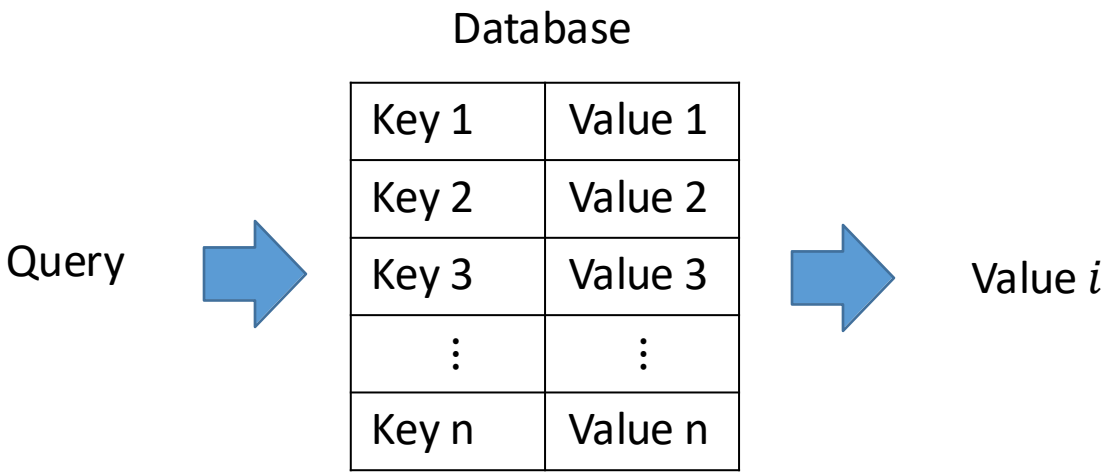
- **Input sequence not bottlenecked through single vector**
- **At each timestep of decoder, context vector "looks at" different parts of the input sequence**



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

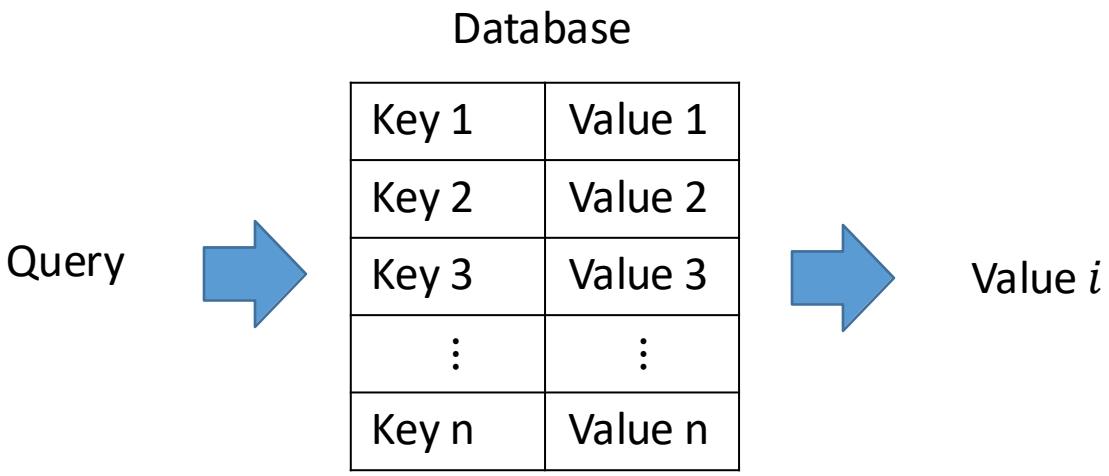
Generalized definition

- Mimics the retrieval of a **value** v_i for a **query** q based on **key** k_i in database.



Generalized definition

- Mimics the retrieval of a **value** v_i for a **query** q based on **key** k_i in database.



$$attention(q, k, v) = \sum_i similarity(q, k_i) \times v_i$$

- To calculate the Attention of a target word with respect to the input word
- Use the Query of the target and the Key of the input
- Calculate a matching score
- These matching scores act as the weights of the Value vectors.

Query	Key	Value
The	The	The
early	early	early
bird	bird	bird
catches	catches	catches
the	the	the
worm	worm	worm

Query	Key	Value
e_{The}	e_{The}	e_{The}
e_{early}	e_{early}	e_{early}
e_{bird}	e_{bird}	e_{bird}
e_{catches}	e_{catches}	e_{catches}
e_{the}	e_{the}	e_{the}
e_{worm}	e_{worm}	e_{worm}

Query	Key	Value
$W_Q^T e_{\text{The}}$	$W_K^T e_{\text{The}}$	$W_V^T e_{\text{The}}$
$W_Q^T e_{\text{early}}$	$W_K^T e_{\text{early}}$	$W_V^T e_{\text{early}}$
$W_Q^T e_{\text{bird}}$	$W_K^T e_{\text{bird}}$	$W_V^T e_{\text{bird}}$
$W_Q^T e_{\text{catches}}$	$W_K^T e_{\text{catches}}$	$W_V^T e_{\text{catches}}$
$W_Q^T e_{\text{the}}$	$W_K^T e_{\text{the}}$	$W_V^T e_{\text{the}}$
$W_Q^T e_{\text{worm}}$	$W_K^T e_{\text{worm}}$	$W_V^T e_{\text{worm}}$

Calculating Attention for the Word "bird"

Objective:

- Calculate the attention weights for the word "**bird**" within the sentence context.

Procedure:

- Focus on the word "**bird**" as our point of interest.

Attention Calculation:

- Calculate a similarity between the query vector of "bird" and the key vectors of other words in the sentence.

"The", "early", "bird", "catches", "the", "worm".

Generalized definition (example)

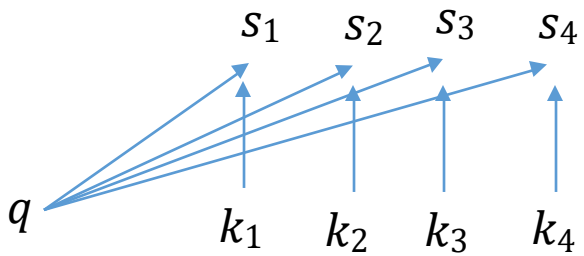
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{The}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{early}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{catches}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{the}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{worm}} \rangle)$$

Generalized definition (example)

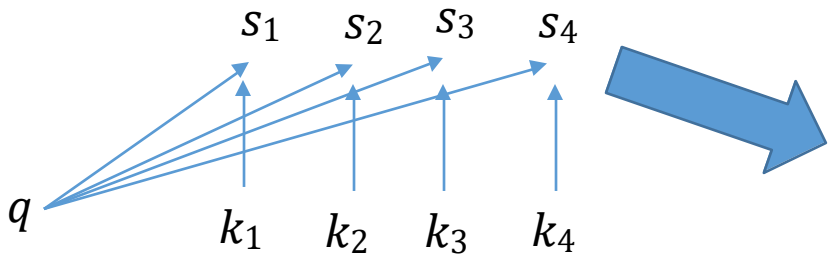
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{The}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{early}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{catches}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{the}} \rangle)$$
$$\text{softmax}(\langle W_Q^T e_{\text{bird}}, W_K^T e_{\text{worm}} \rangle)$$

$$v_{\text{bird}} = a_1 v_{\text{The}} + a_2 v_{\text{early}} + a_3 v_{\text{catches}} + \dots$$

Neural architecture



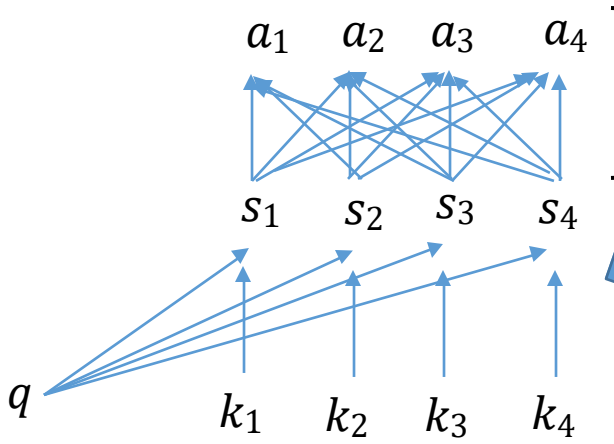
Neural architectu



$$s_i = similarity(k_i, q) = \begin{cases} q^T k_i \\ q^T k_i / \sqrt{p} \\ q^T W W k_i \\ \vdots \end{cases}$$

(p is the dimensionality of k)

Neural architectu

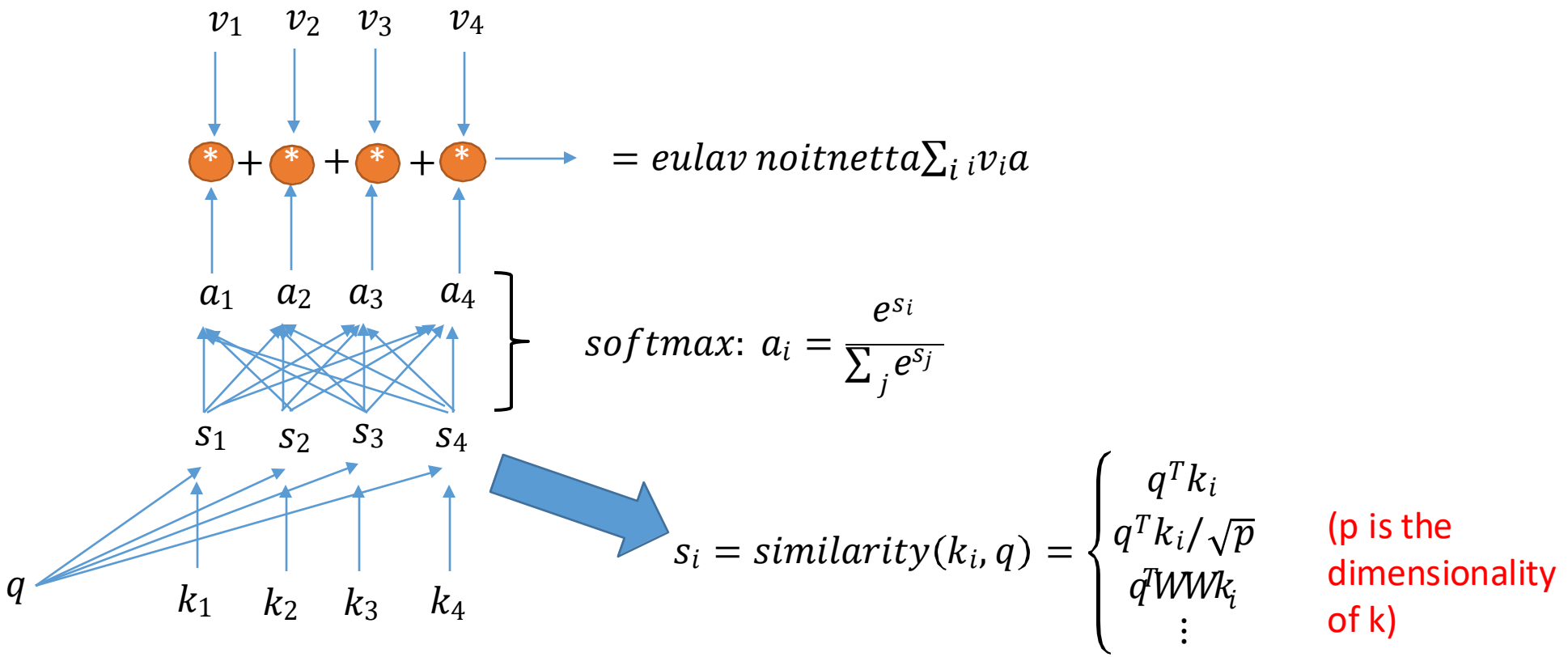


$$softmax:\ a_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

$$s_i = similarity(k_i, q) = \begin{cases} q^T k_i \\ q^T k_i / \sqrt{p} \\ q^T W W k_i \\ \vdots \end{cases}$$

(p is the dimensionality of k)

Neural architecture



Matrix Forms:

- Words in sequence: $X = [x_1, ..., x_n] \in \mathbb{R}^{d \times n}$
- Queries: $Q = [q_1, ..., q_n] \in \mathbb{R}^{p \times n}$
- Keys: $K = [k_1, ..., k_n] \in \mathbb{R}^{p \times n}$
- Values: $V = [v_1, ..., v_n] \in \mathbb{R}^{r \times n}$

Projection into Subspaces (Post Definitions):

- Queries: $q_i = W_Q x_i$
- Keys: $k_i = W_K x_i$
- Values: $v_i = W_V x_i$

Matrix Form

Attention Computation:

- $Z := \text{attention}(Q, K, V) = V\text{softmax}\left(\frac{1}{\sqrt{p}}\, \mathrm{K}^{\mathrm{T}}\mathrm{Q}\right)$