# Lecture 16:
# Recurrent Neural Networks

Subhransu Maji, Chuang Gan and TAs
Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

# Visual Question Answering


Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.


Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

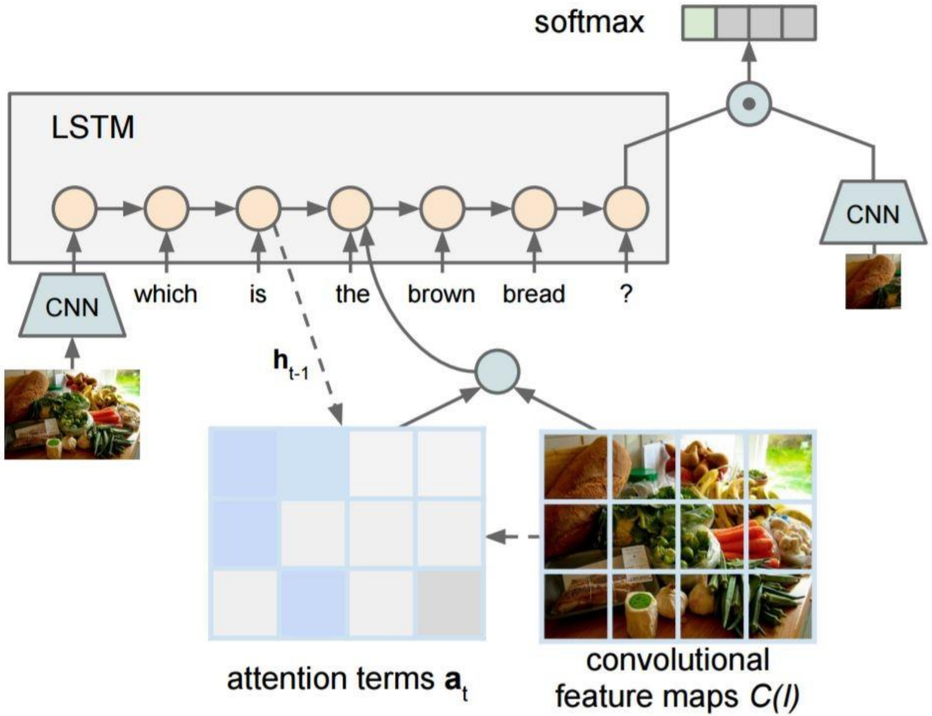
A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

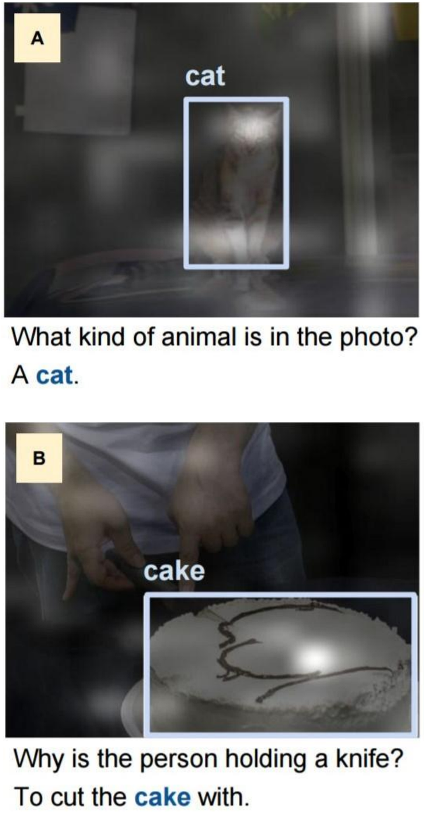
Q: Who is under the umbrella?

A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

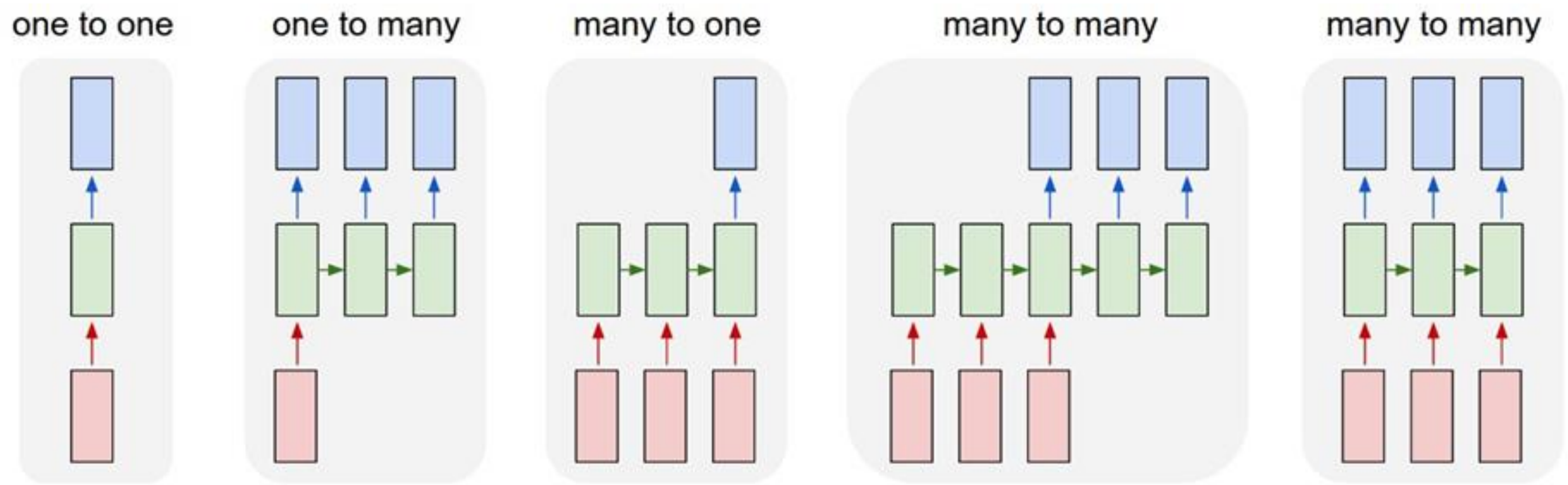
# Visual Question Answering: RNNs with Attention



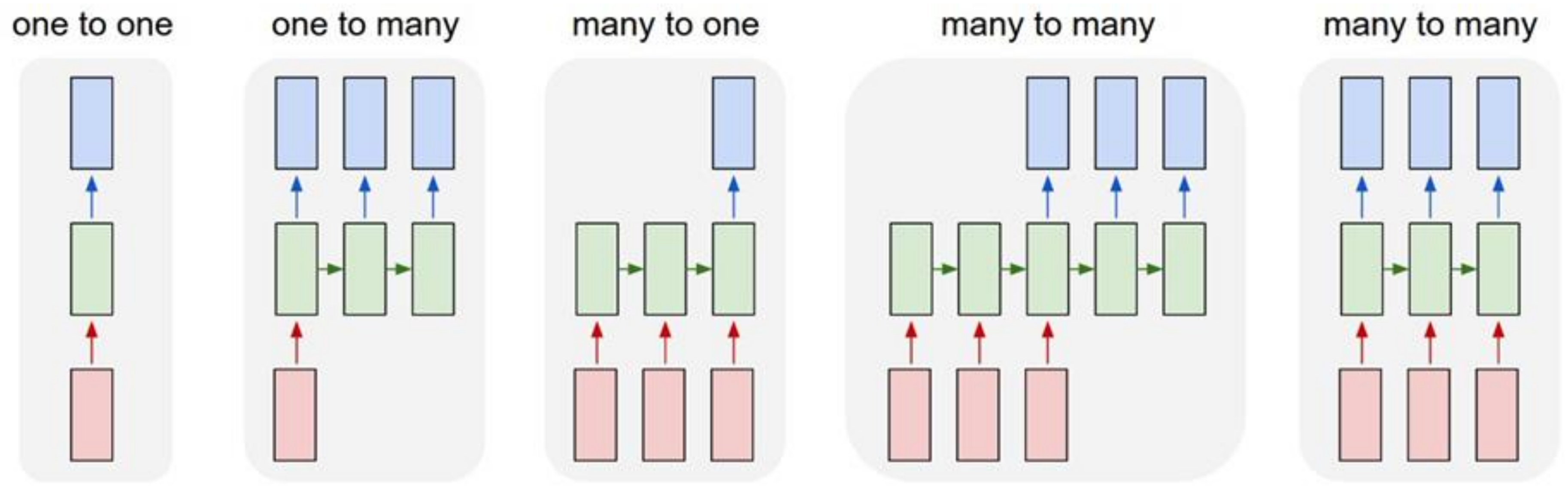

Zhu et al, “Visual 7W: Grounded Question Answering in Images”, CVPR 2016

# Recurrent Networks offer a lot of flexibility:



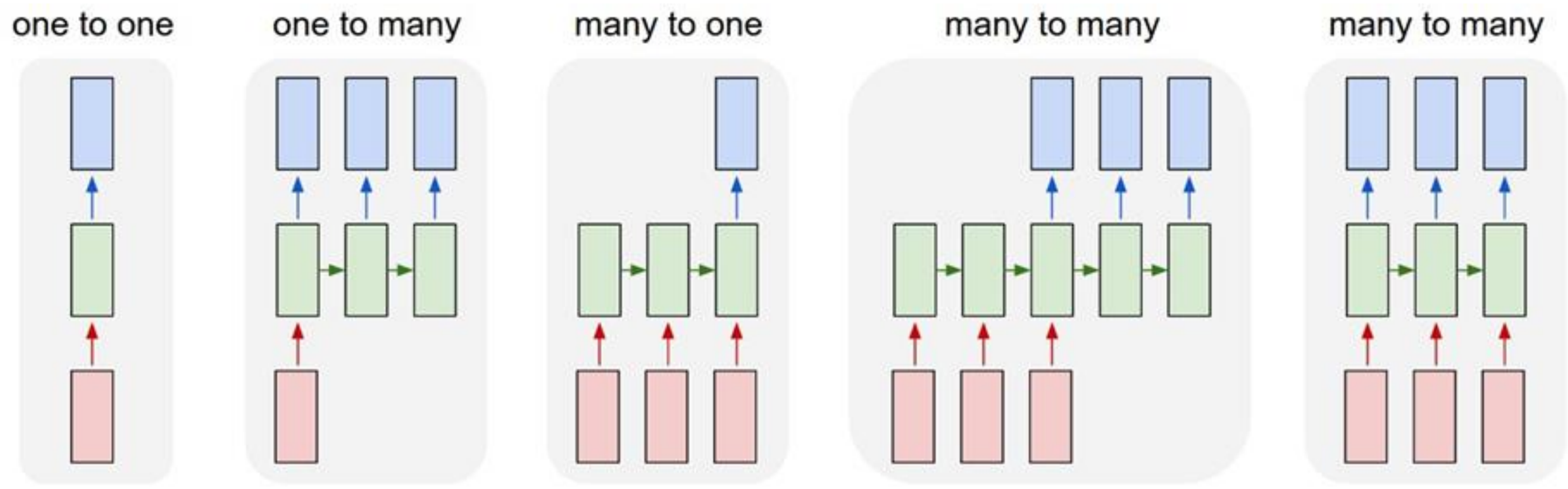

**Vanilla Neural Networks**

# Recurrent Networks offer a lot of flexibility:



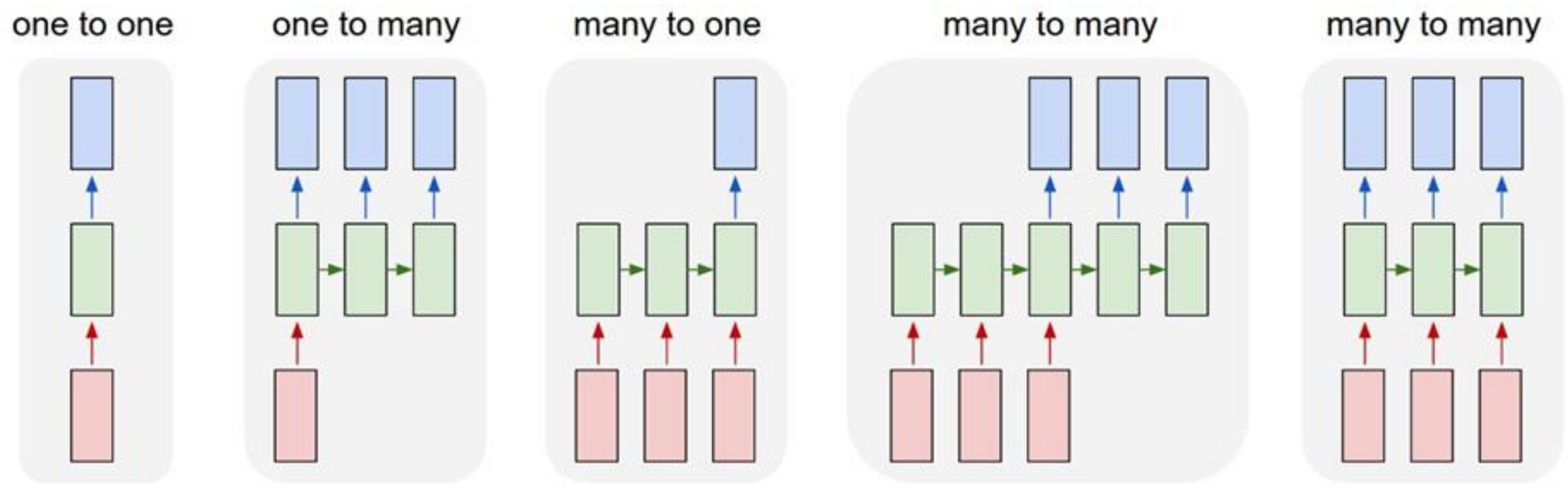

e.g. **Image Captioning**
image -> sequence of words

# Recurrent Networks offer a lot of flexibility:



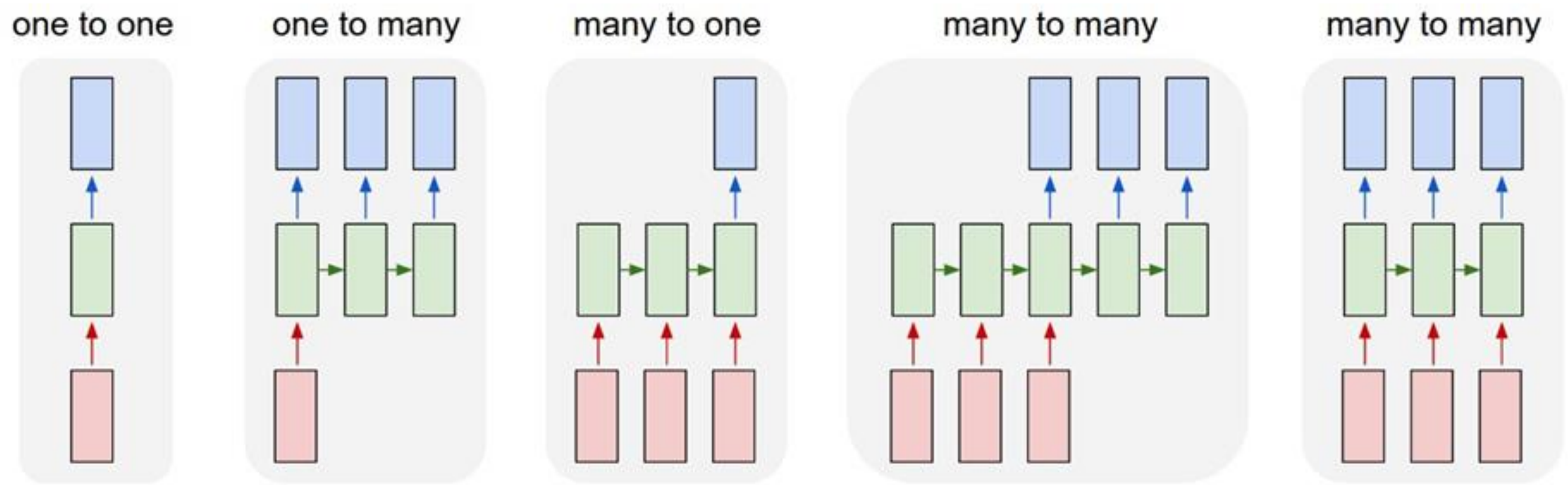

e.g. **Sentiment Classification**
sequence of words -> sentiment

# Recurrent Networks offer a lot of flexibility:



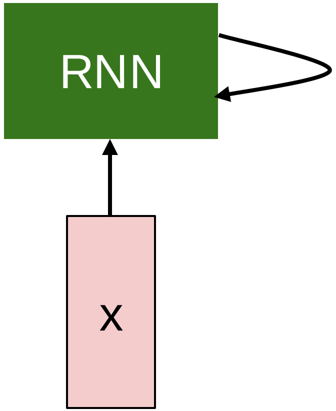

e.g. **Machine Translation**
seq of words -> seq of words

# Recurrent Networks offer a lot of flexibility:

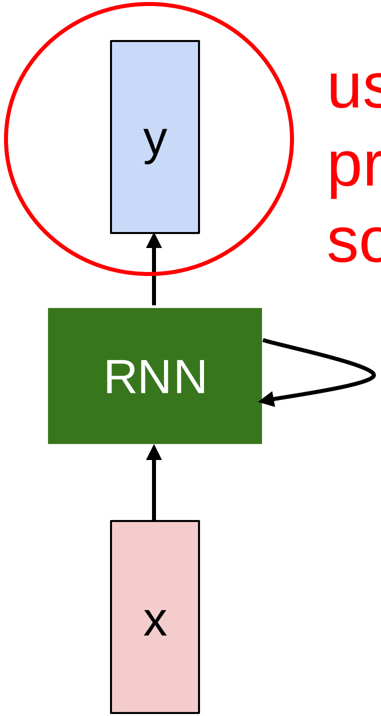



e.g. **Video classification on frame level**

# Recurrent Neural Network



# Recurrent Neural Network




usually want to predict a vector at some time steps

# Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

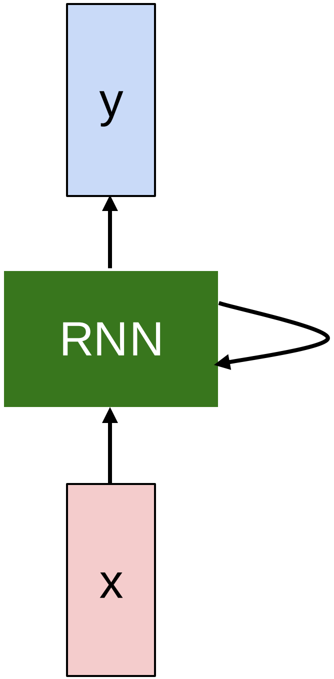
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

input vector at some time step

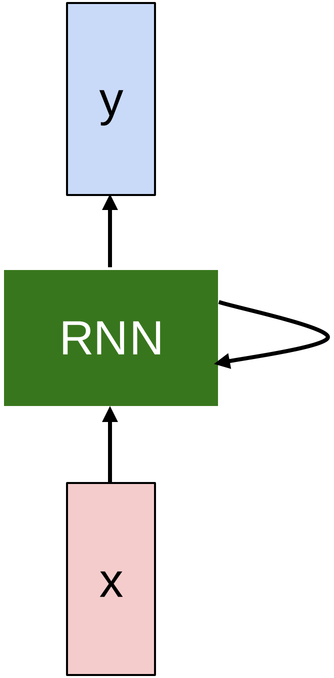
some function with parameters W



# Recurrent Neural Network

We can process a sequence of vectors **x** by applying a recurrence formula at every time step:

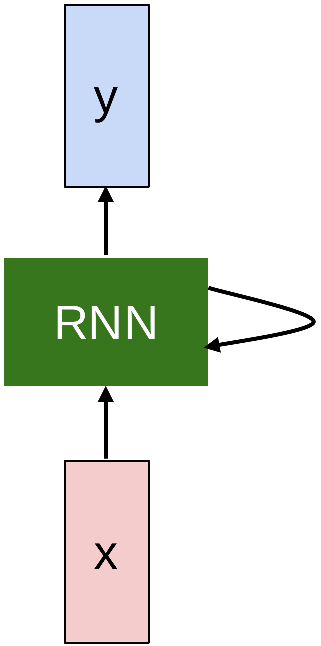
$$h_t = f_W(h_{t-1}, x_t)$$




Notice: the same function and the same set of parameters are used at every time step.

# (Vanilla) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:




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

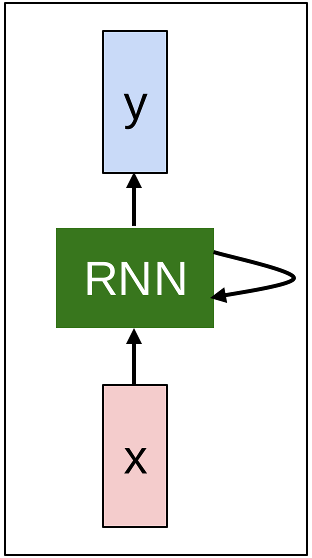
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

**Character-level language model example**

Vocabulary:
[h,e,l,o]

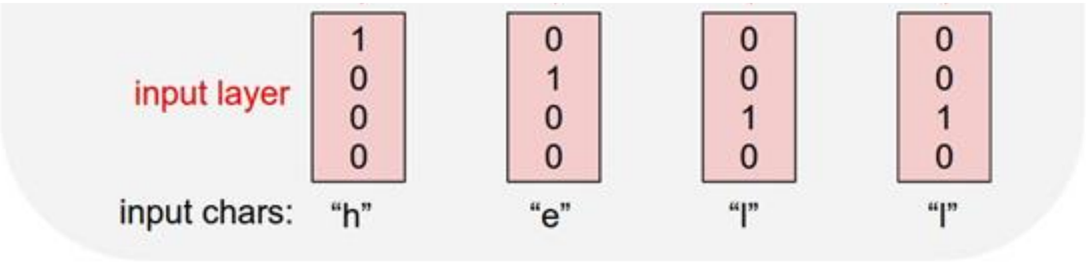
Example training sequence:
**“hello”**



## Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
**“hello”**

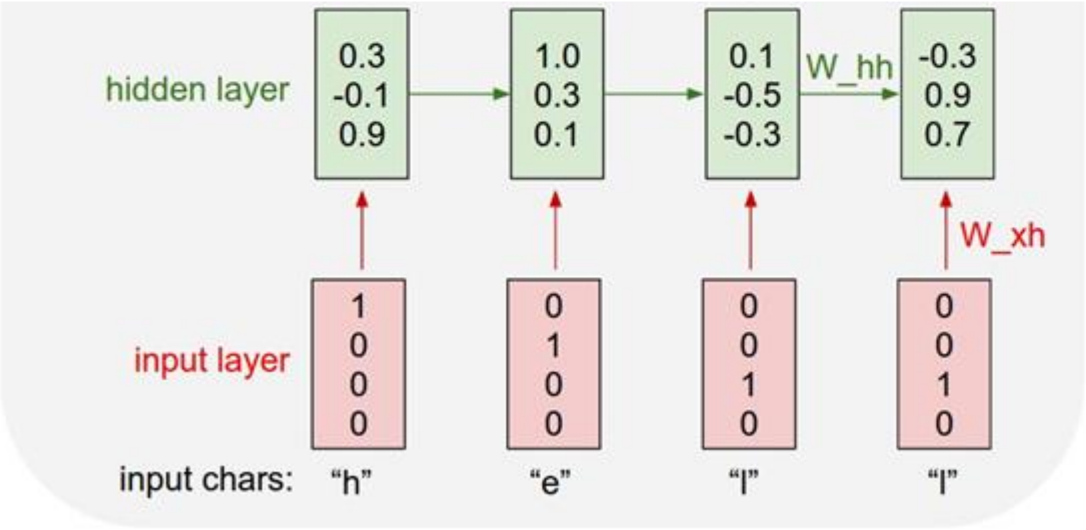


# Character-level language model example

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary:
[h,e,l,o]

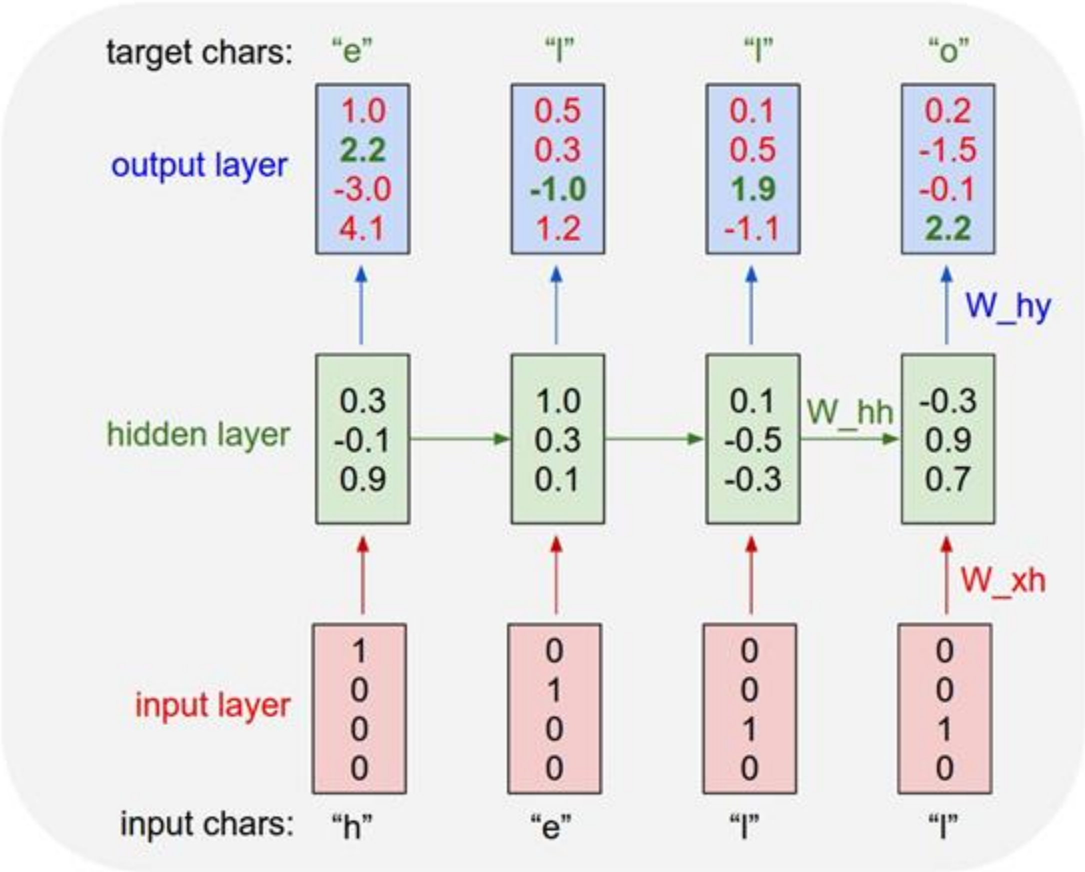
Example training sequence:
**“hello”**



**Character-level language model example**

Vocabulary:
[h,e,l,o]

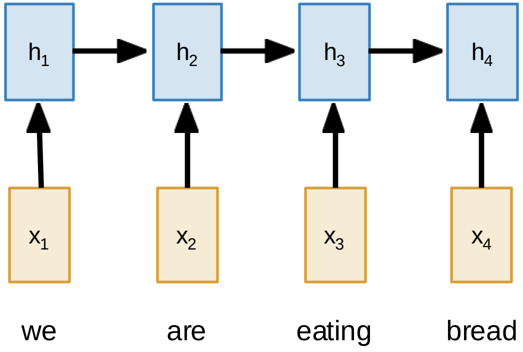
Example training sequence:
**“hello”**



# Sequence to Sequence with RNNs

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

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

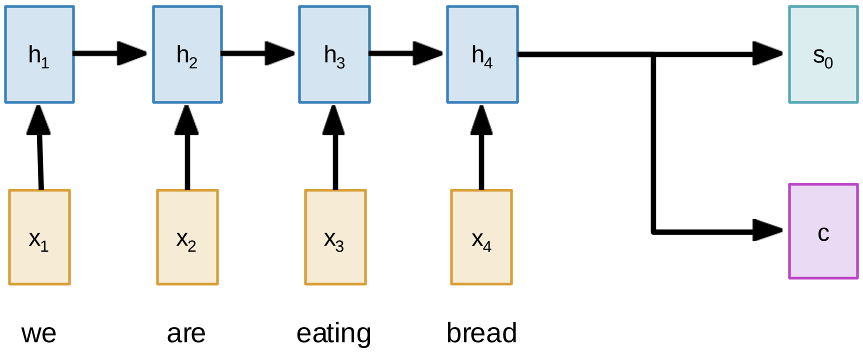


# Sequence to Sequence with RNNs

**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 $c=h_T$)



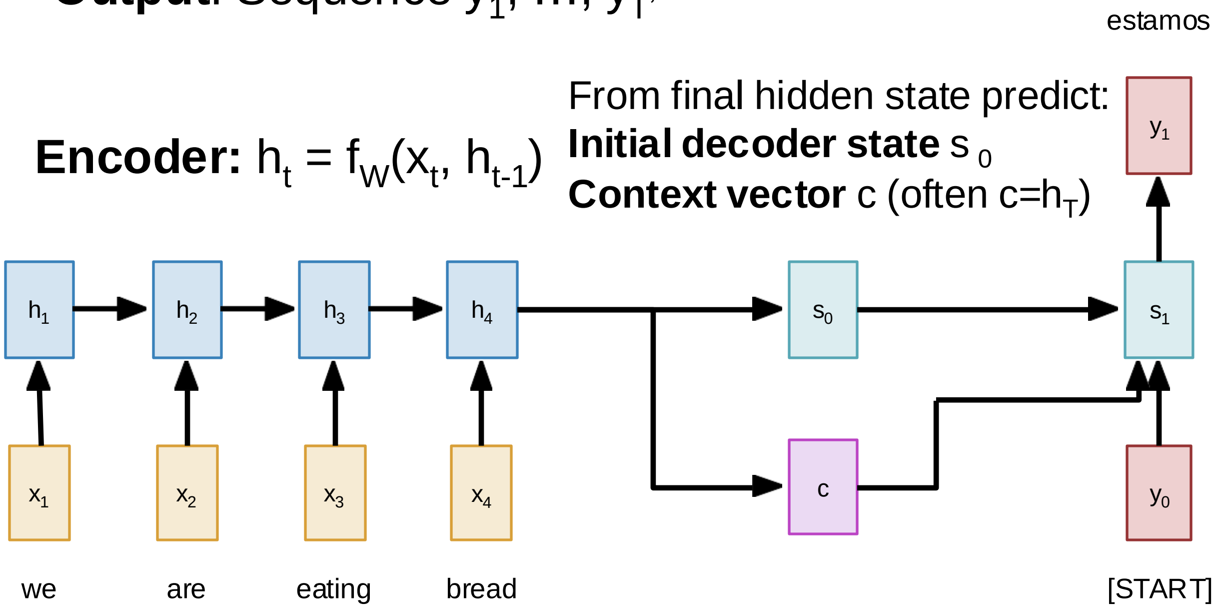
# Sequence to Sequence with RNNs

**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 $c=h_T$)



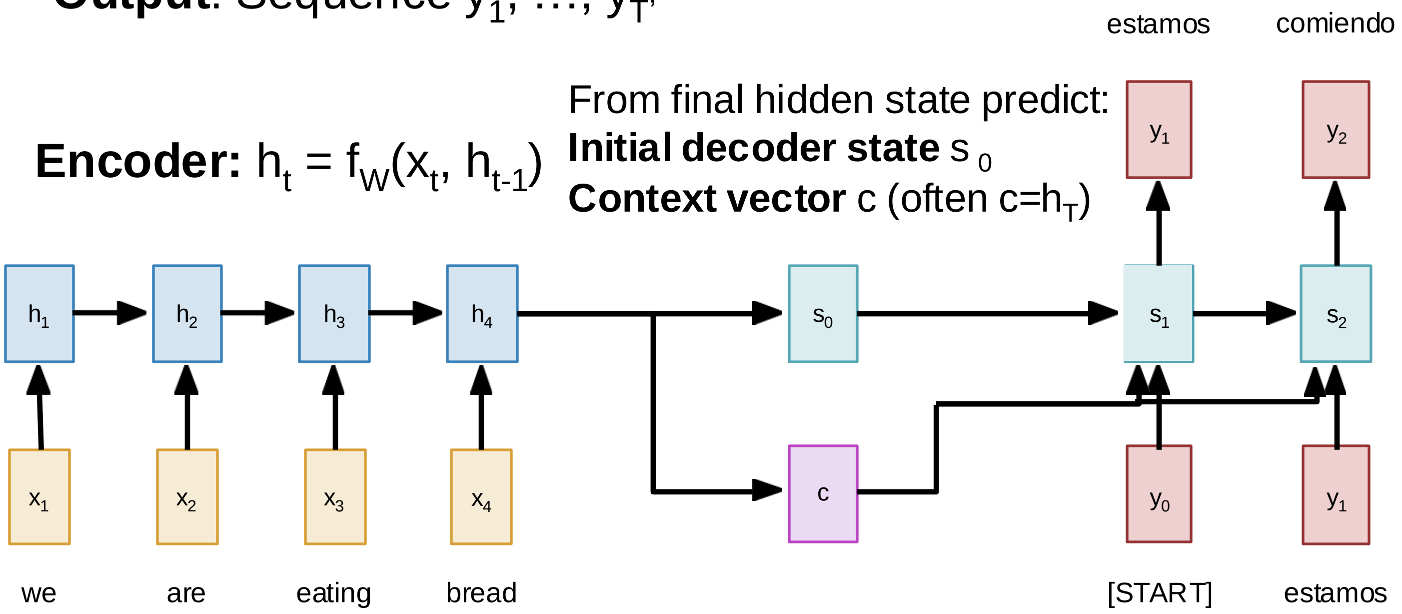
# Sequence to Sequence with RNNs

**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 $c=h_T$)



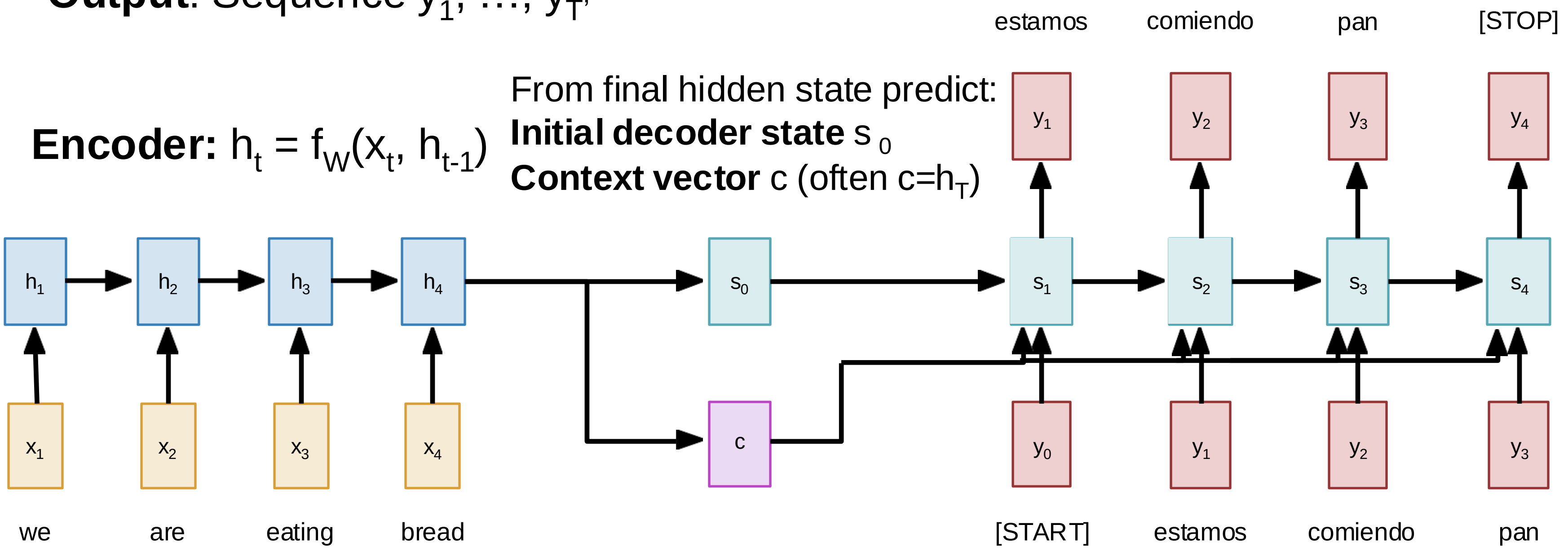
# Sequence to Sequence with RNNs

**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 $c=h_T$)



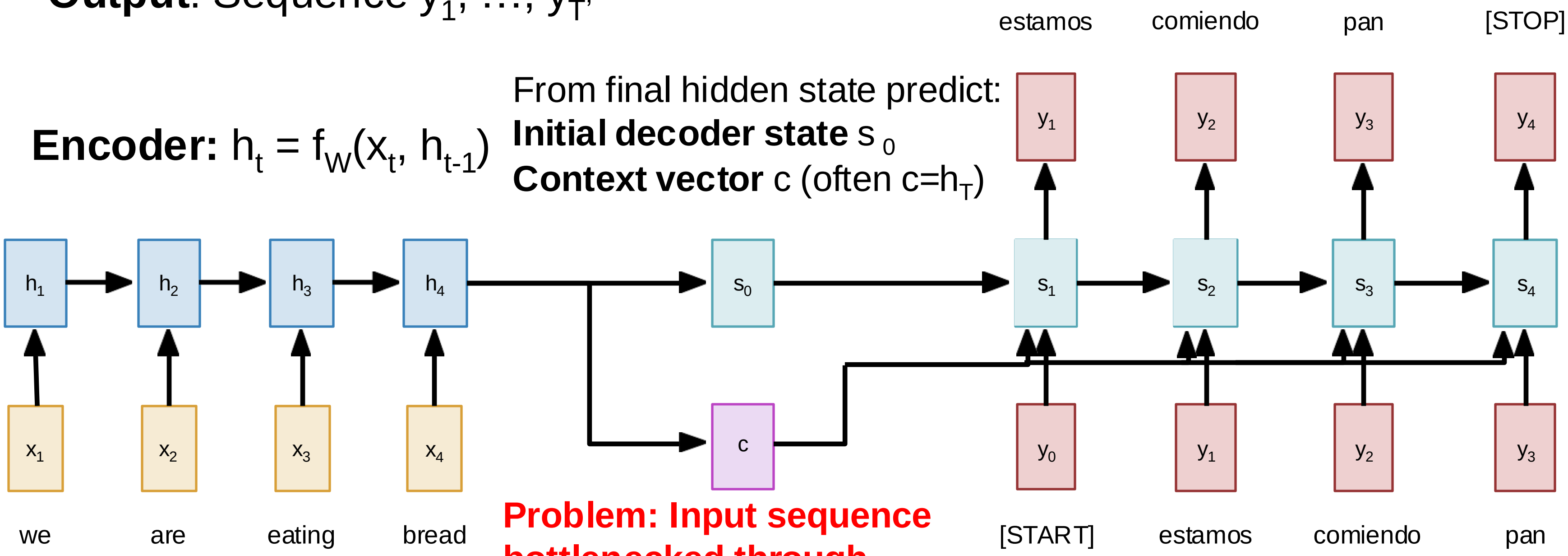
# Sequence to Sequence with RNNs

**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 $c=h_T$)




**Problem: Input sequence bottlenecked through fixed-sized vector. What if**

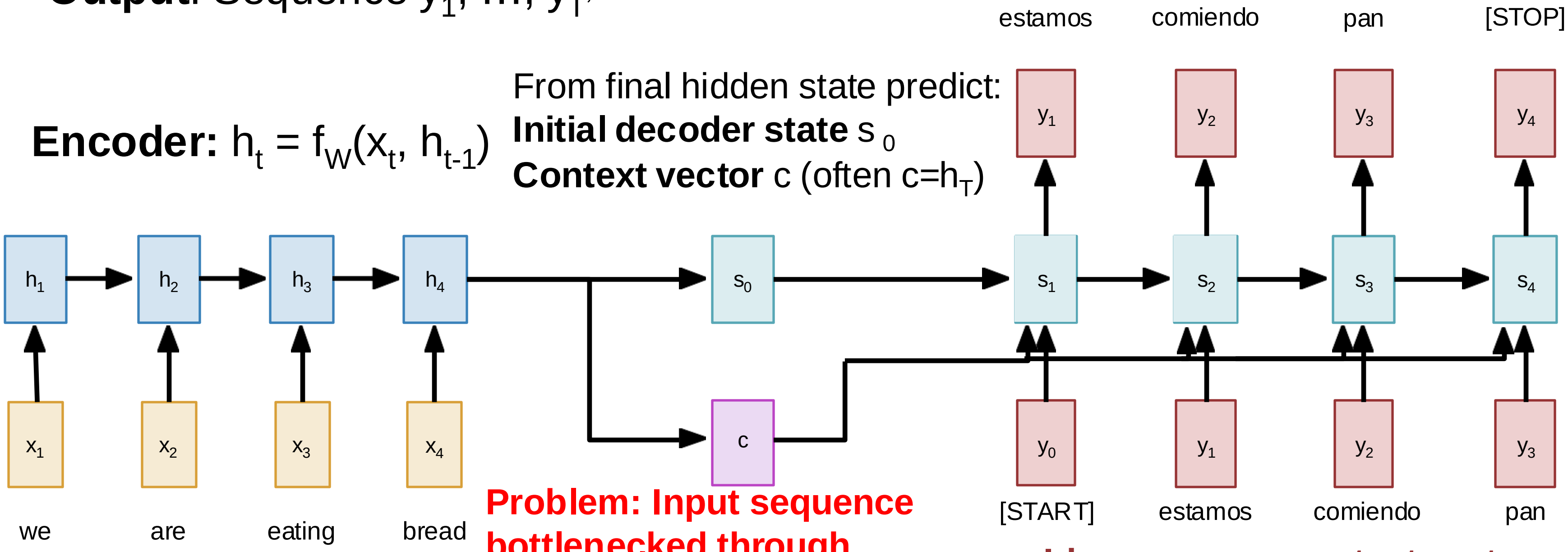
# Sequence to Sequence with RNNs

**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 $c=h_T$)




**Problem: Input sequence bottlenecked through fixed-sized vector. What if**

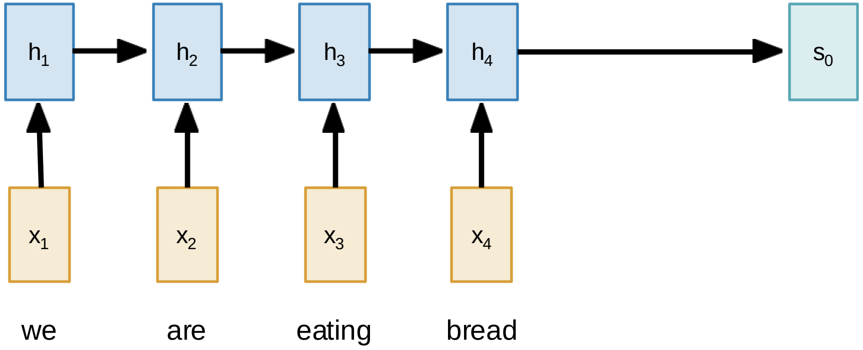
**Idea: use new context vector at each step of decoder!**

# Sequence to Sequence with RNNs and Attention

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

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

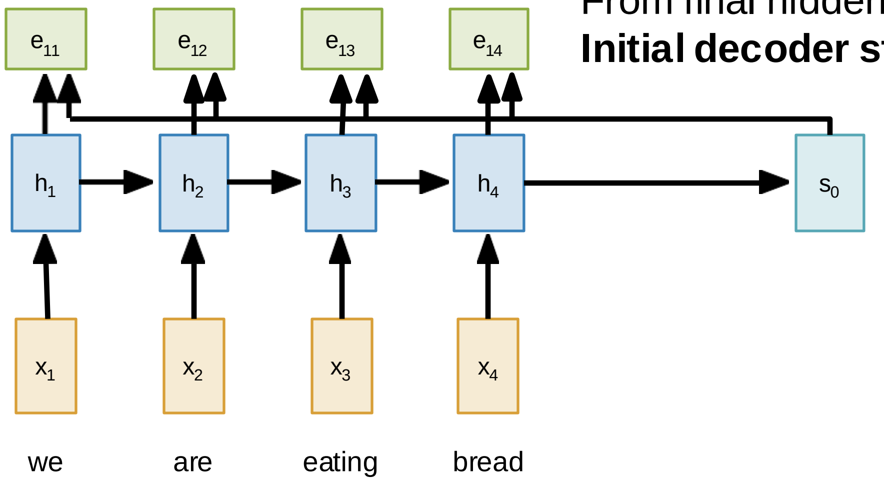
From final hidden state:
**Initial decoder state** $s_0$



# Sequence to Sequence with RNNs and Attention

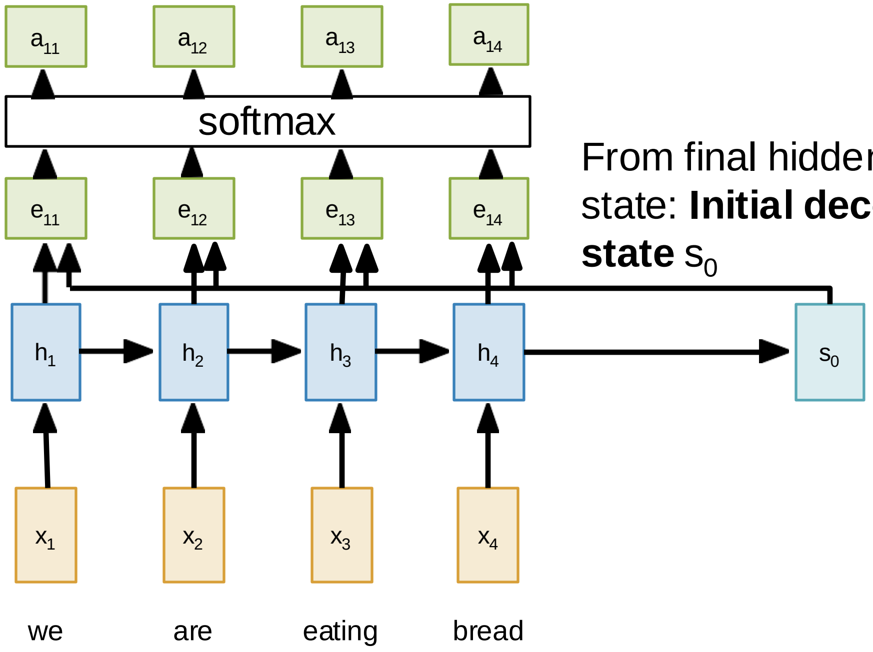
Compute (scalar) **alignment scores**

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




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

# Sequence to Sequence with RNNs and Attention

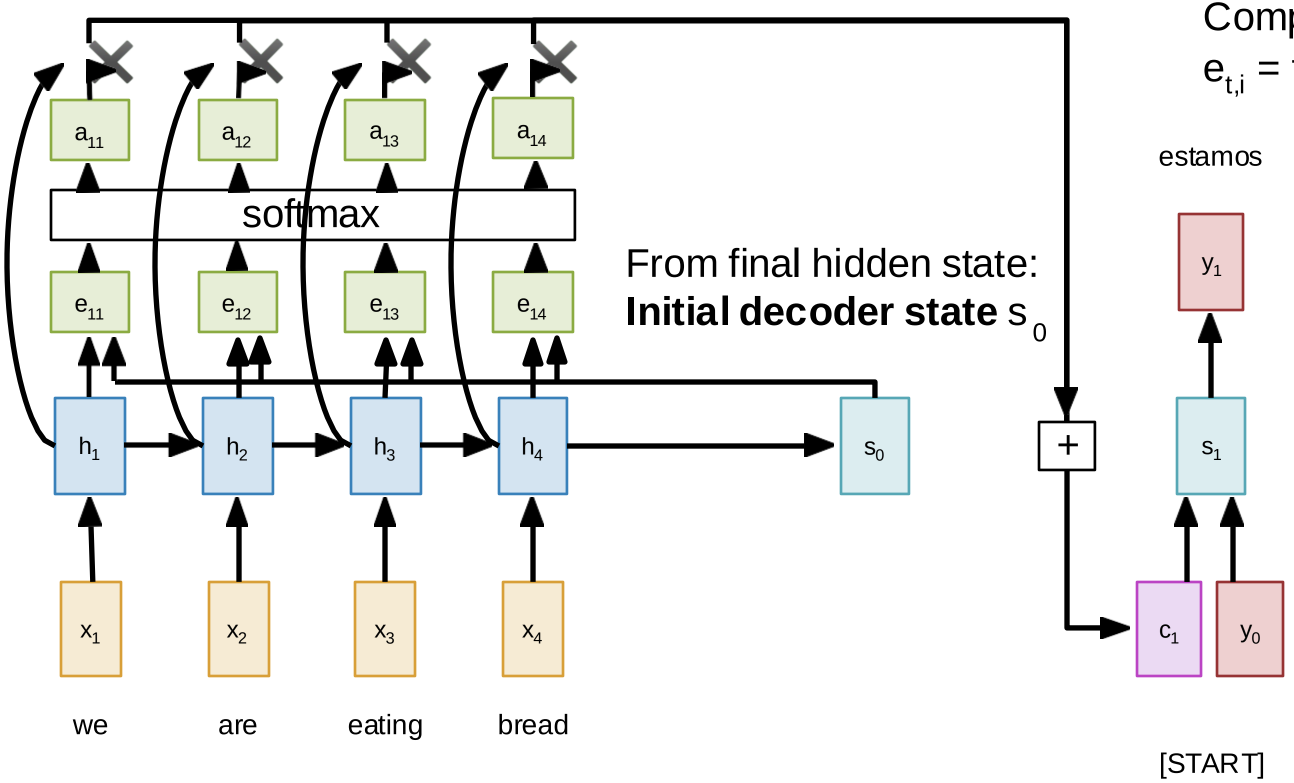



From final hidden state: **Initial decoder state** $s_0$

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

# Sequence to Sequence with RNNs and Attention



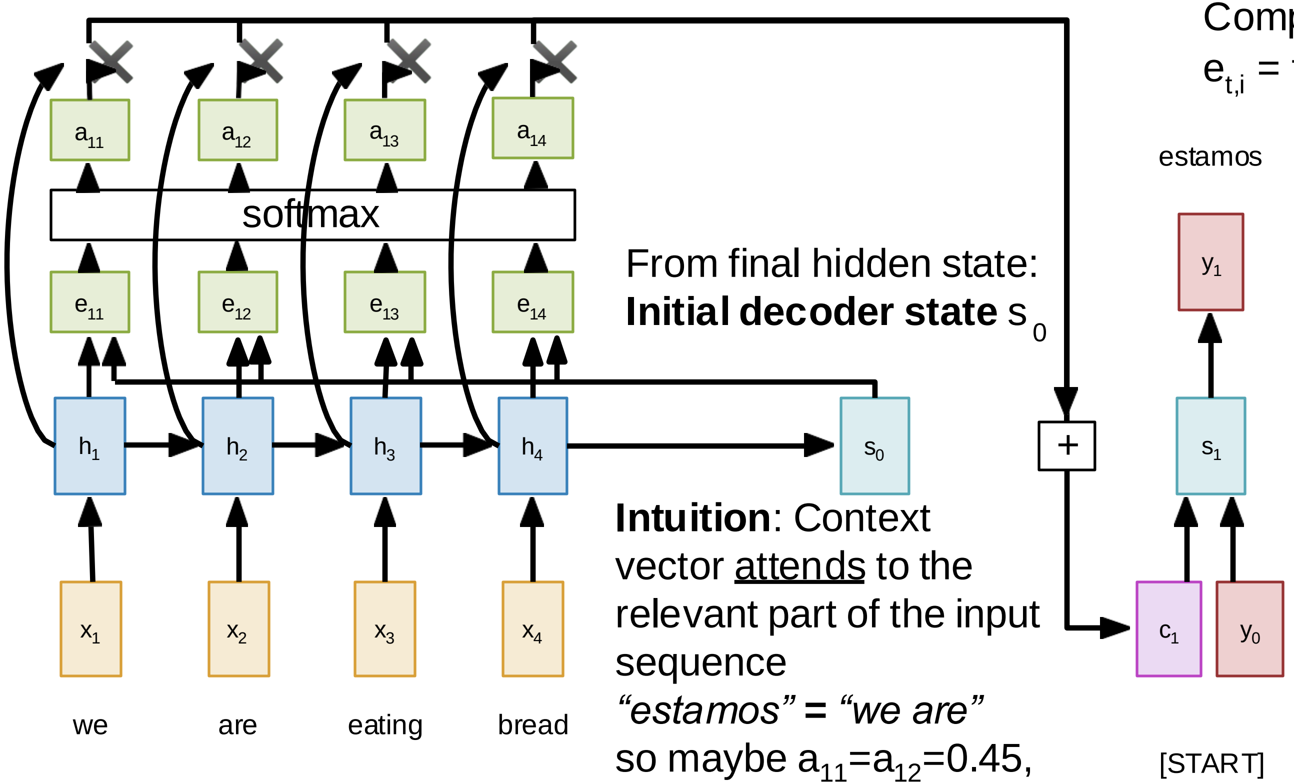

From final hidden state:
**Initial decoder state $s_0$**

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

# Sequence to Sequence with RNNs and Attention




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

**Intuition**: Context vector <u>attends</u> to the relevant part of the input sequence
*"estamos"* **=** *"we are"*
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

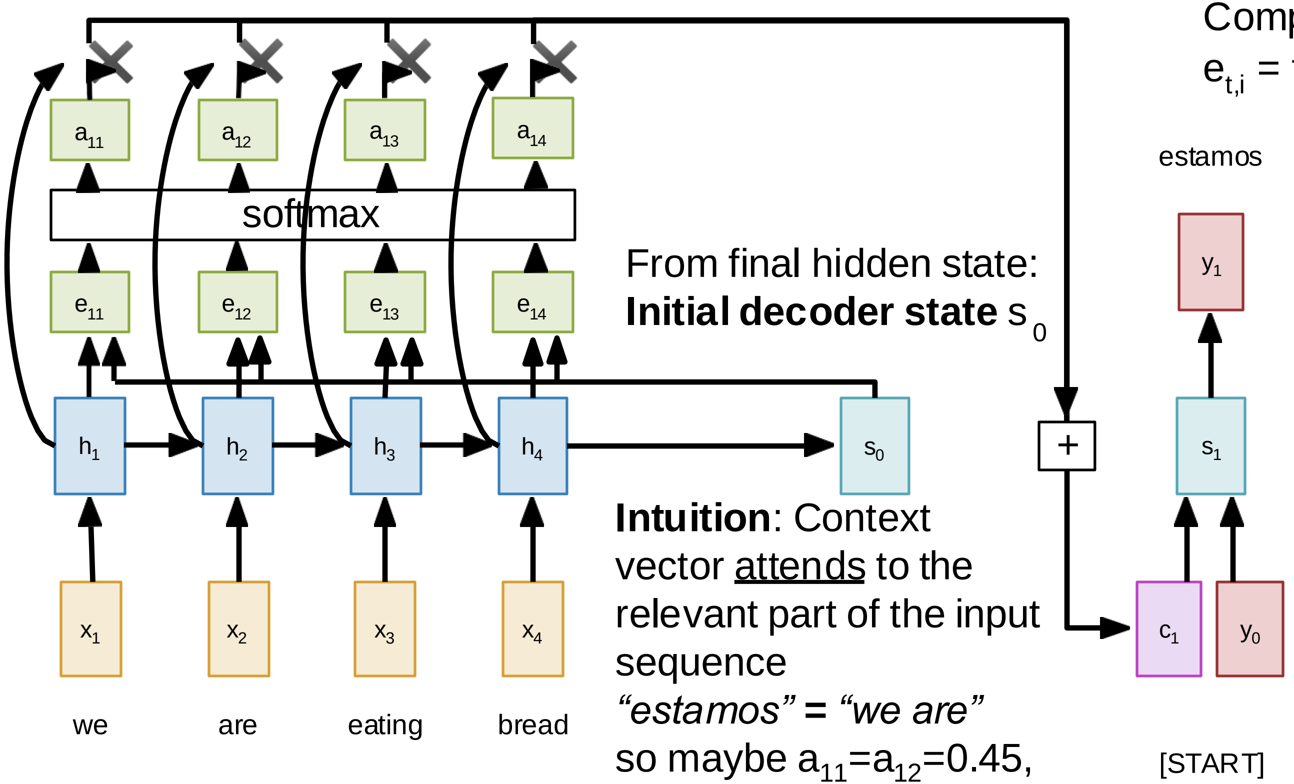
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ ($f_{att}$ is an MLP)

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

# 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** $s_0$

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

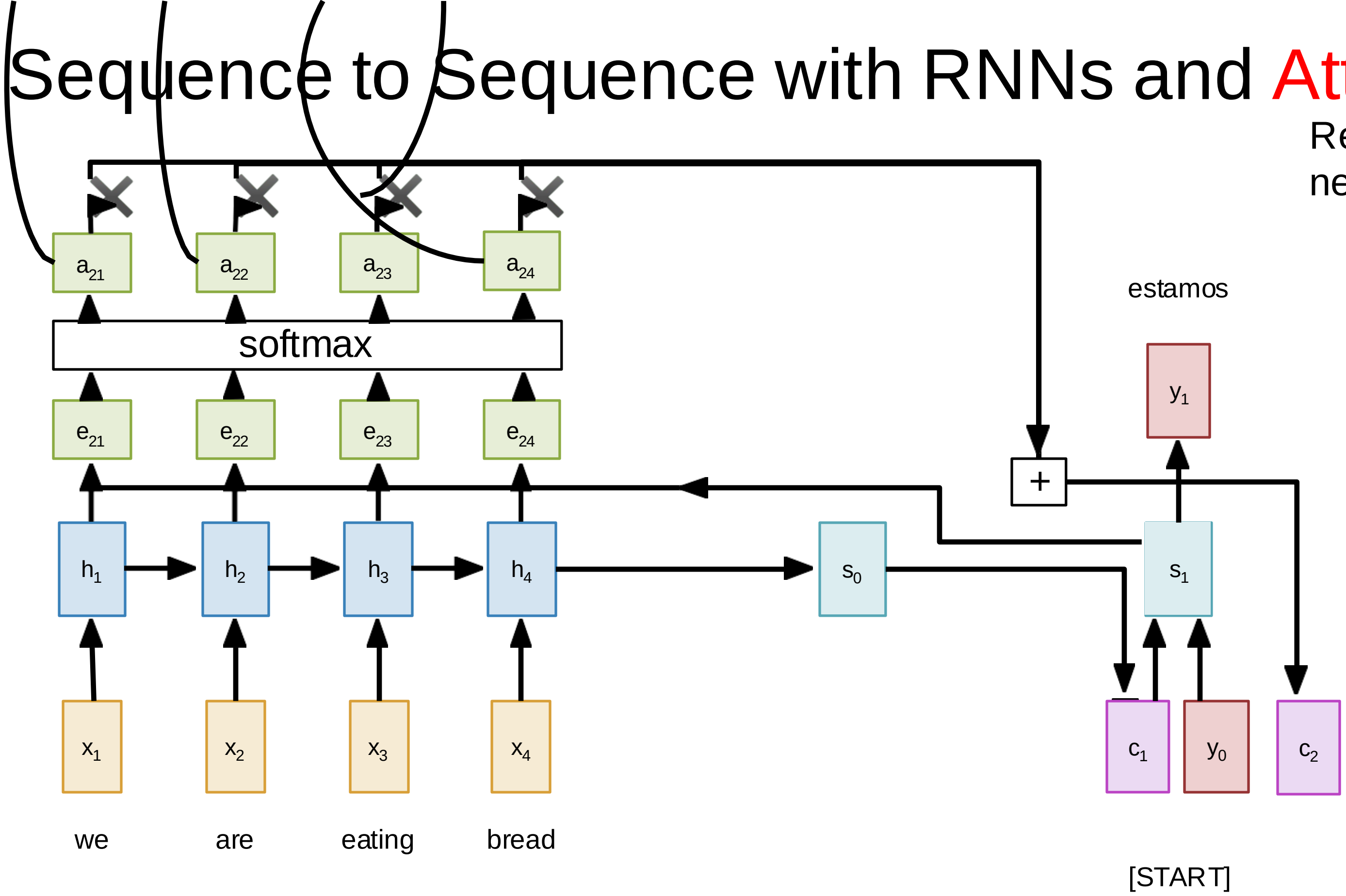
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

**Intuition**: Context vector attends to the relevant part of the input sequence
*"estamos"* = *"we are"*
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

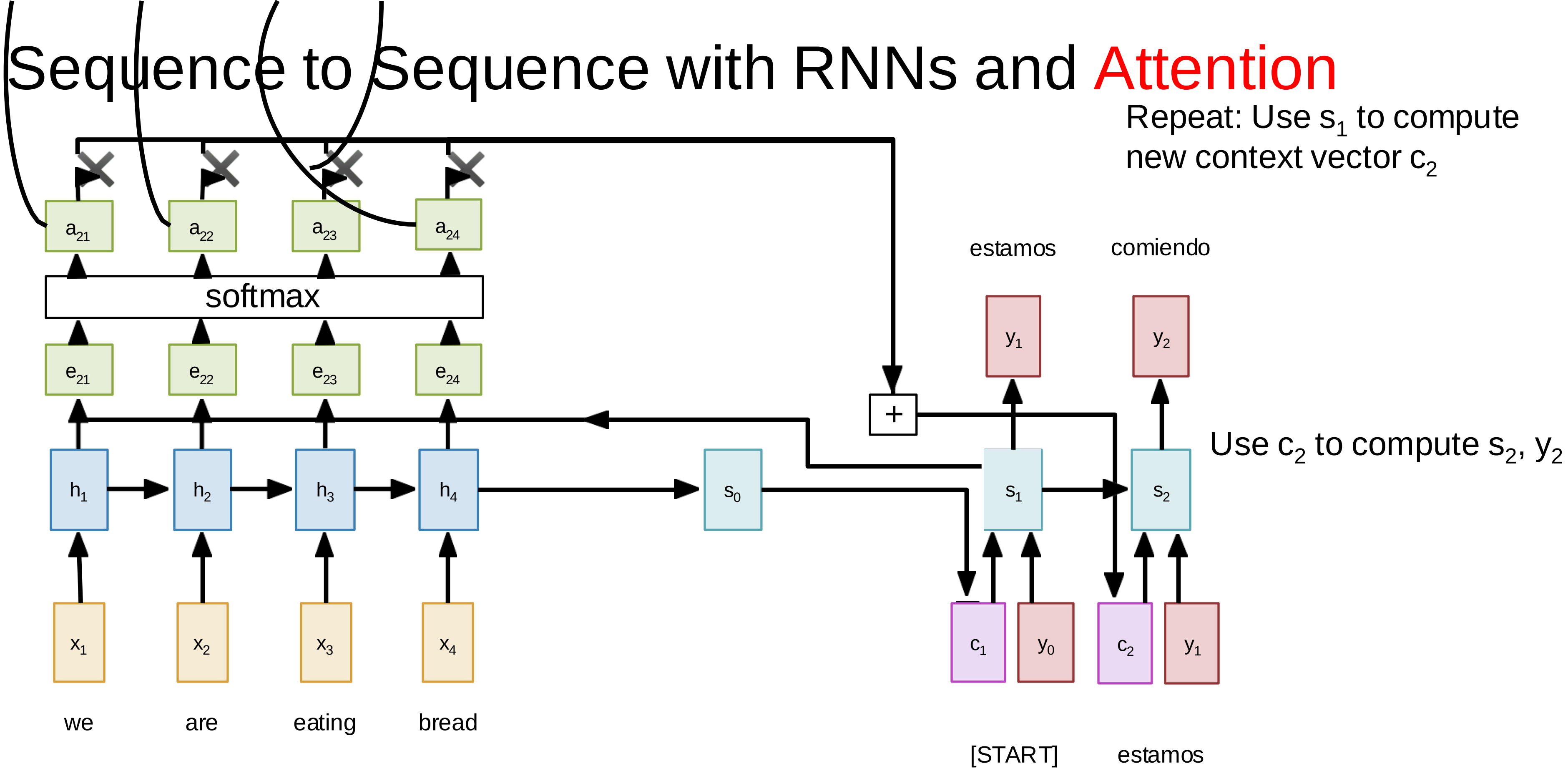
**This is all differentiable! No supervision on attention weights – backprop through everything**

# Sequence to Sequence with RNNs and Attention

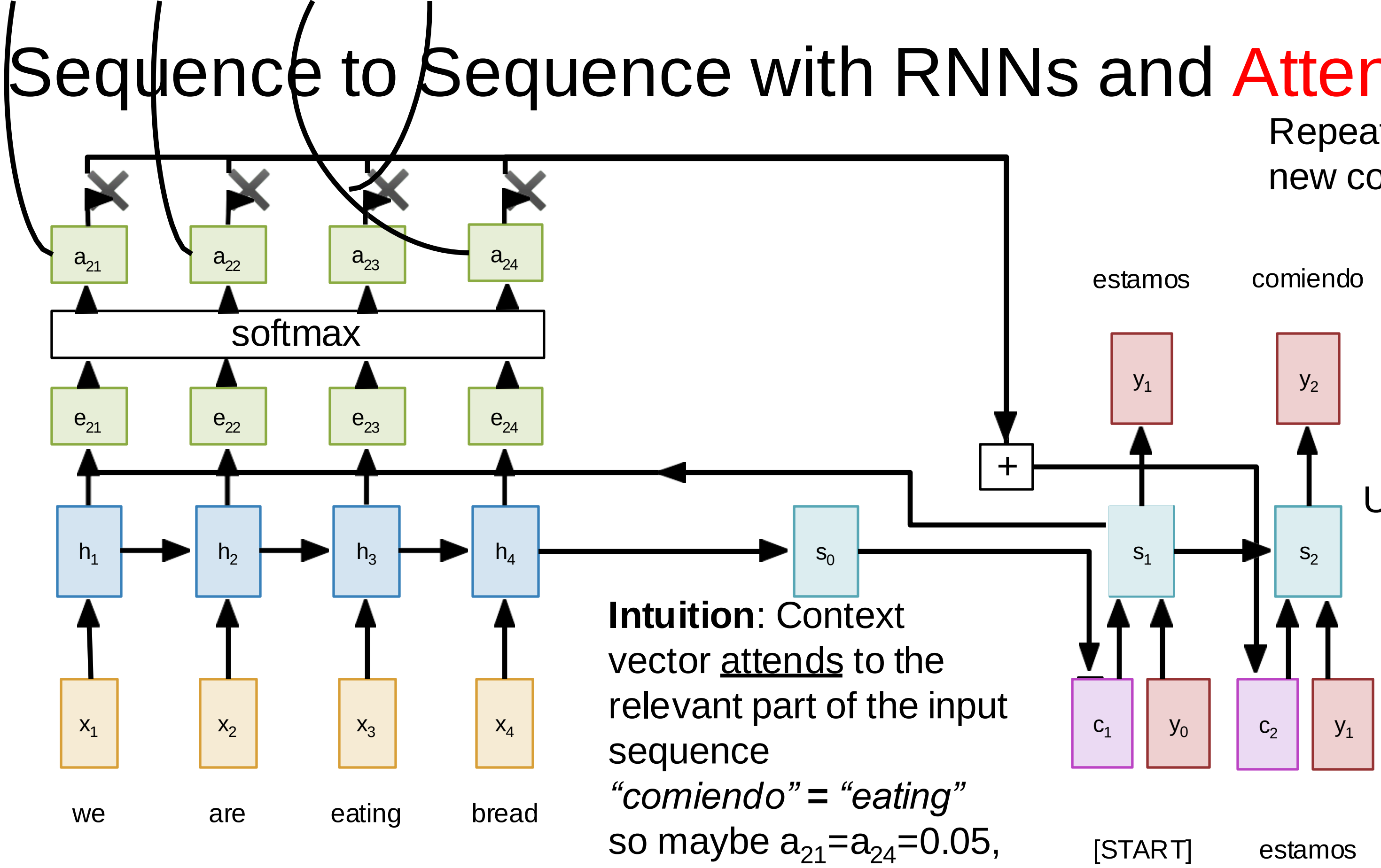
Repeat: Use $s_1$ to compute new context vector $c_2$



# Sequence to Sequence with RNNs and Attention



# 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 <u>attends</u> to the relevant part of the input sequence

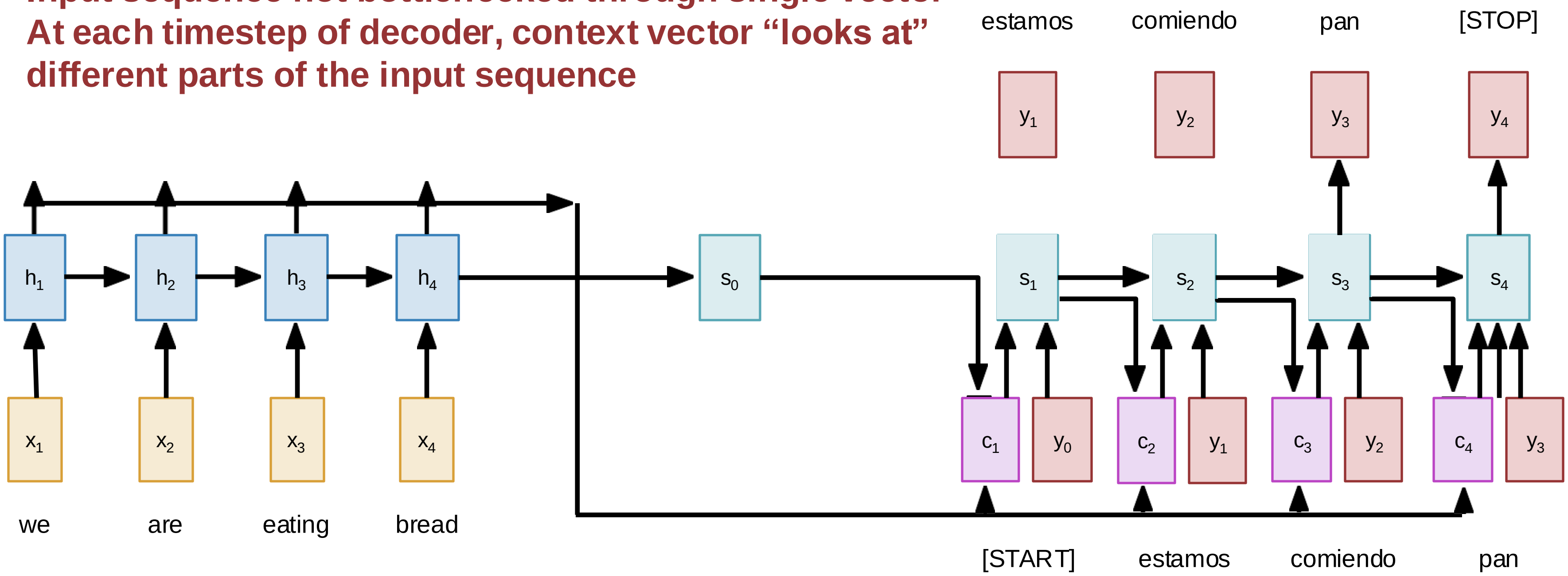
*"comiendo"* **=** *"eating"*

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

# Sequence to Sequence with RNNs and Attention

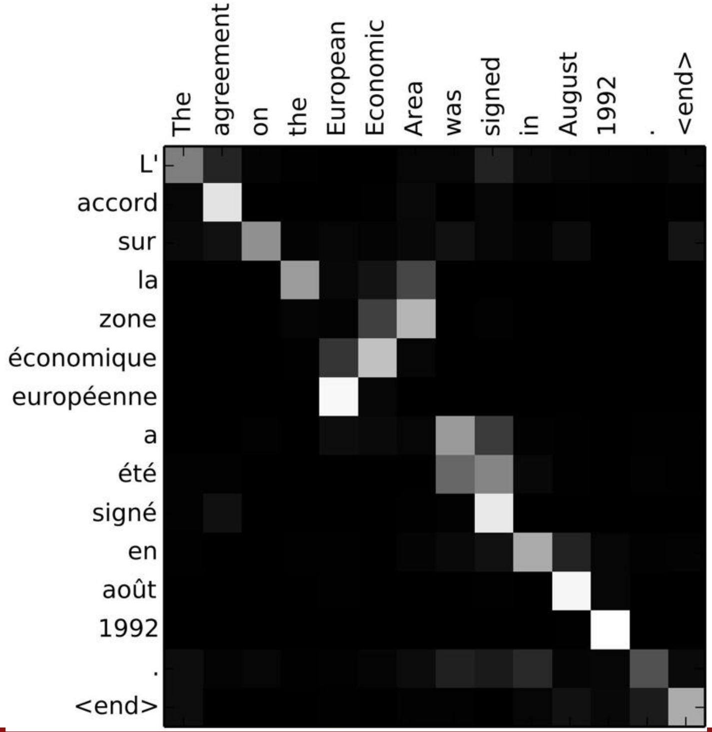
**Use a different context vector in each timestep of decoder**

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



# Sequence to Sequence with RNNs and Attention

Visualize attention weights $a_{t,i}$




**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."

# Sequence to Sequence with RNNs and Attention

**Example**: English to French translation

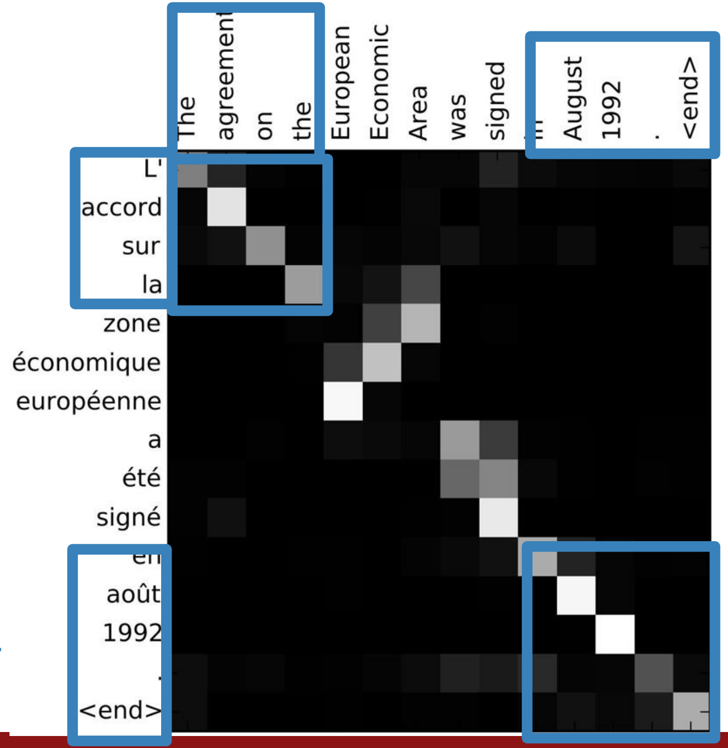
**Input**: "**The agreement on the** European Economic Area was signed **in August 1992**."

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

Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

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."

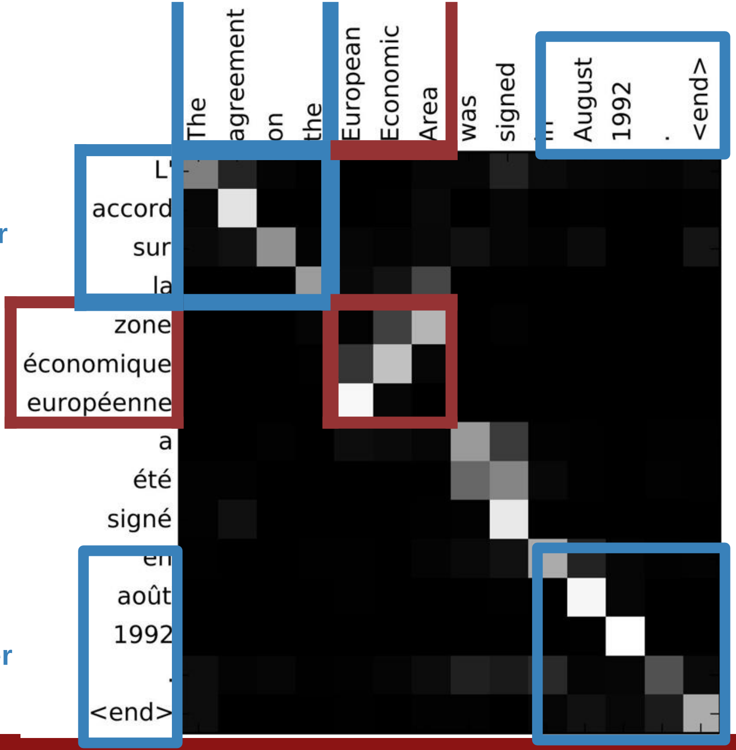
**Output**: "L'accord sur la zone économique européenne a été signé en août 1992."

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$

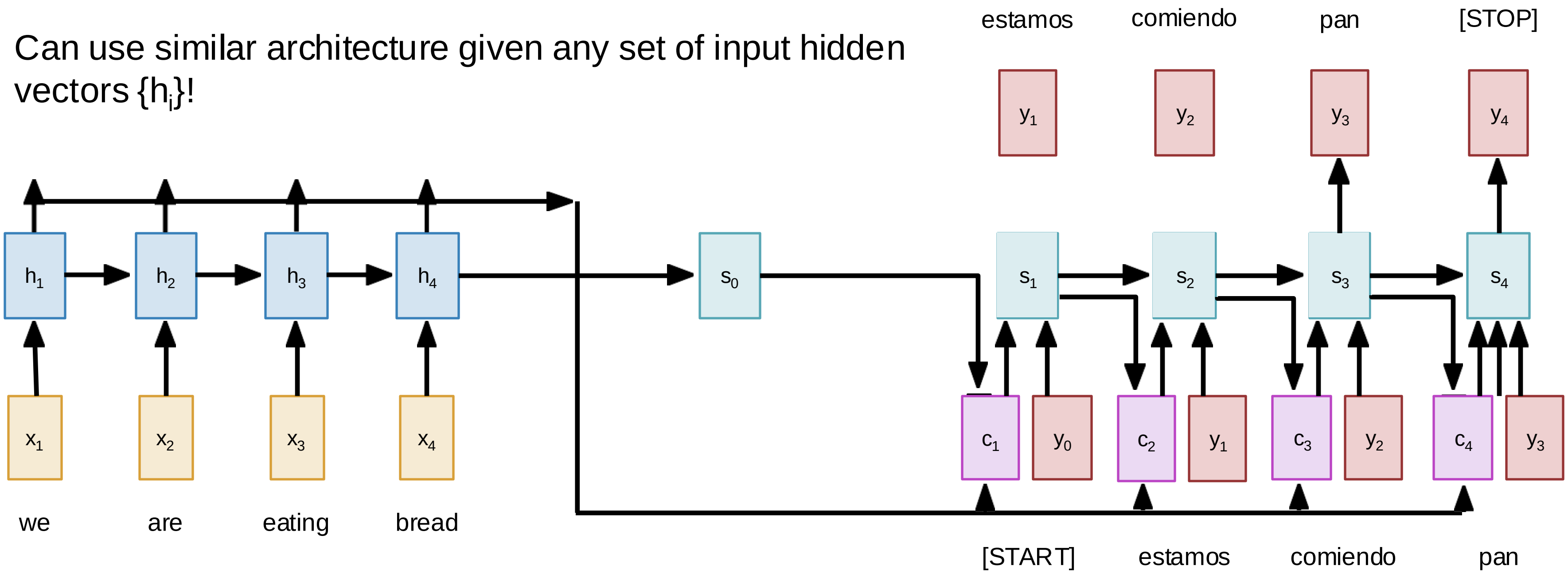



Subhransu Maji, Chuang Gan and TAs
Some slides kindly provided by Fei-Fei Li, Jiajun Wu, Erik Learned-Miller

# 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 $\{h_i\}$!



# 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/simpler architectures are a hot topic of current research
- Better understanding (both theoretical and empirical) is needed.