Neural Machine Translation

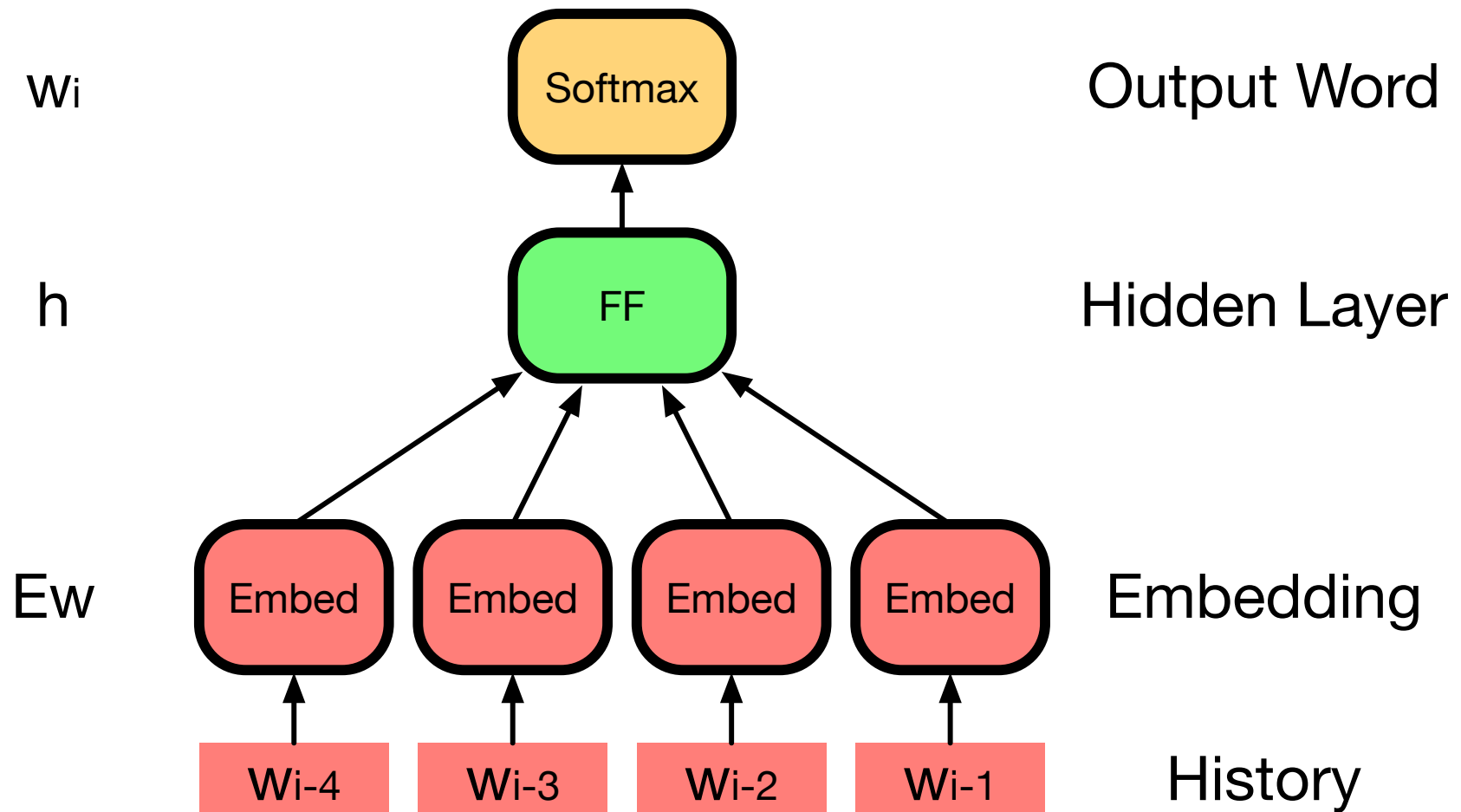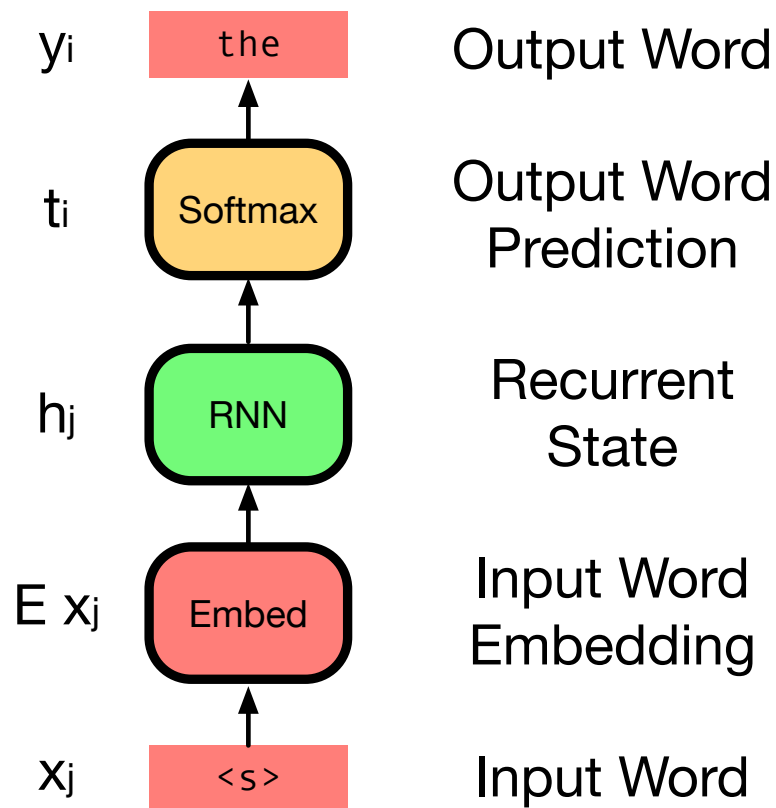Philipp Koehn

3 October 2023

Language Models

- Modeling variants

- feed-forward neural network

  - recurrent neural network

  - long short term memory neural network

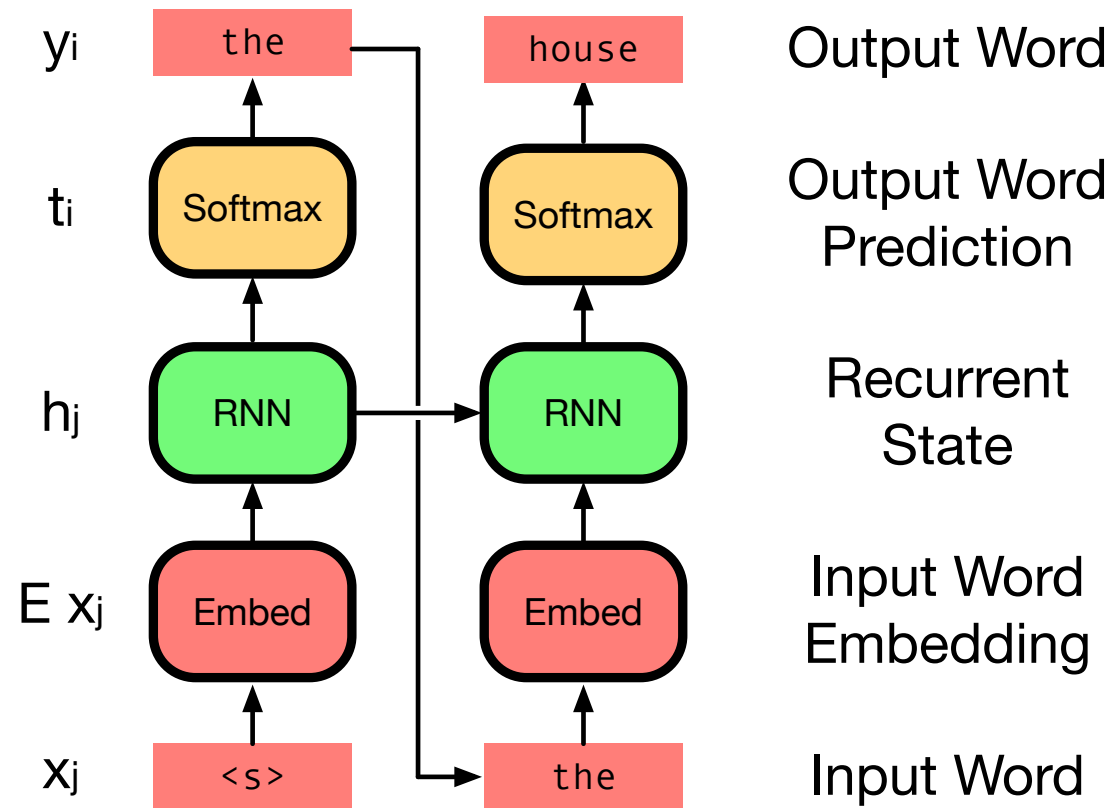- May include input context

Philipp Koehn

Feed Forward Neural Language Model

w_i **Softmax**   Output Word

h **FF**   Hidden Layer

E_w Embed   Embed   Embed   Embed   Embedding

w_{i-4} w_{i-3} w_{i-2} w_{i-1} History

Recurrent Neural Language Model

Predict the first word of a sentence

Recurrent Neural Language Model

y_i the   house   Output Word

t_i Softmax   Softmax   Output Word Prediction

h_j RNN   RNN   Recurrent State

$E\,x_j$ Embed   Embed   Input Word Embedding

x_j <s>   the   Input Word
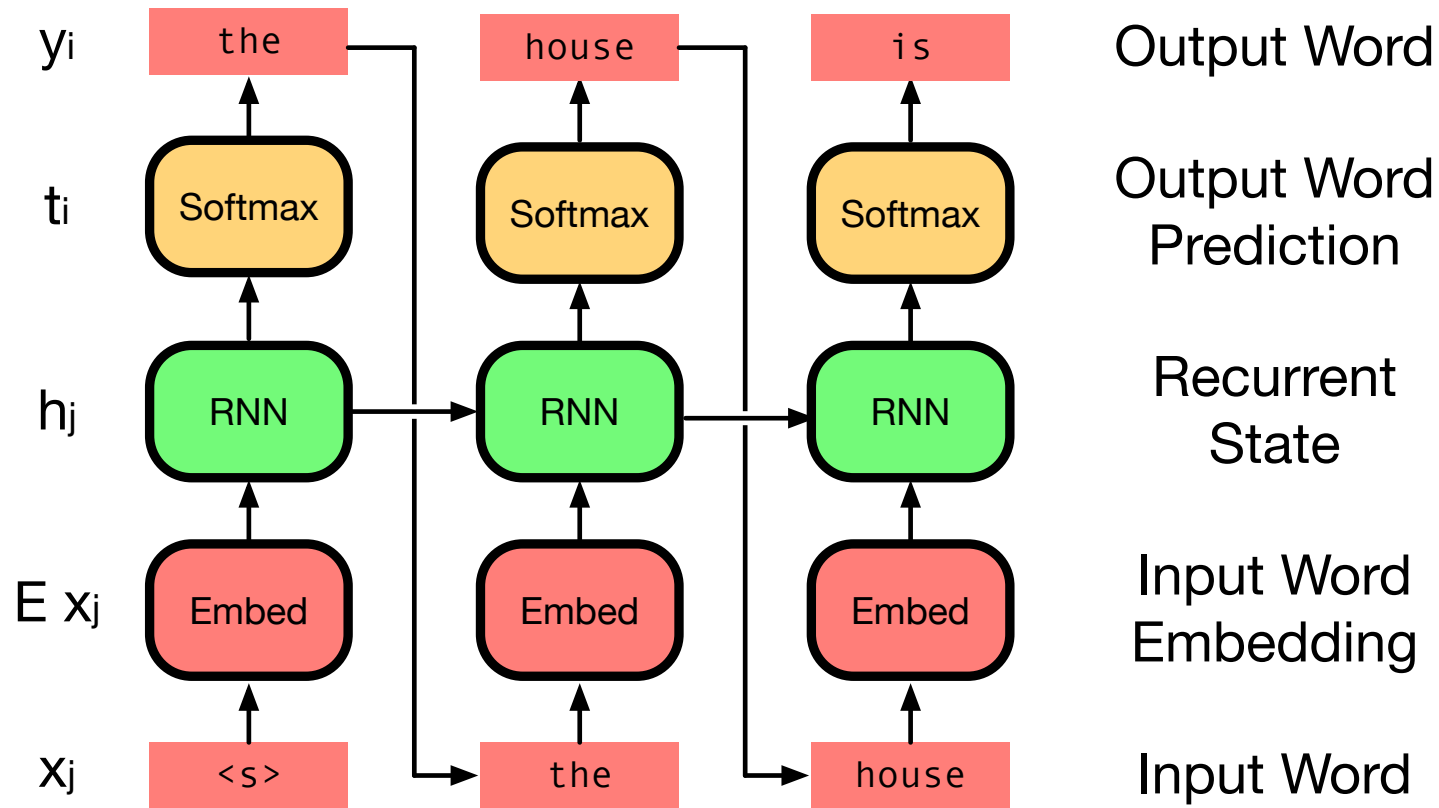
Predict the second word of a sentence

Re-use hidden state from first word prediction

Recurrent Neural Language Model

y_i | the   house   is   Output Word

t_i Softmax   Softmax   Softmax   Output Word Prediction

h_j RNN   RNN   RNN   Recurrent State

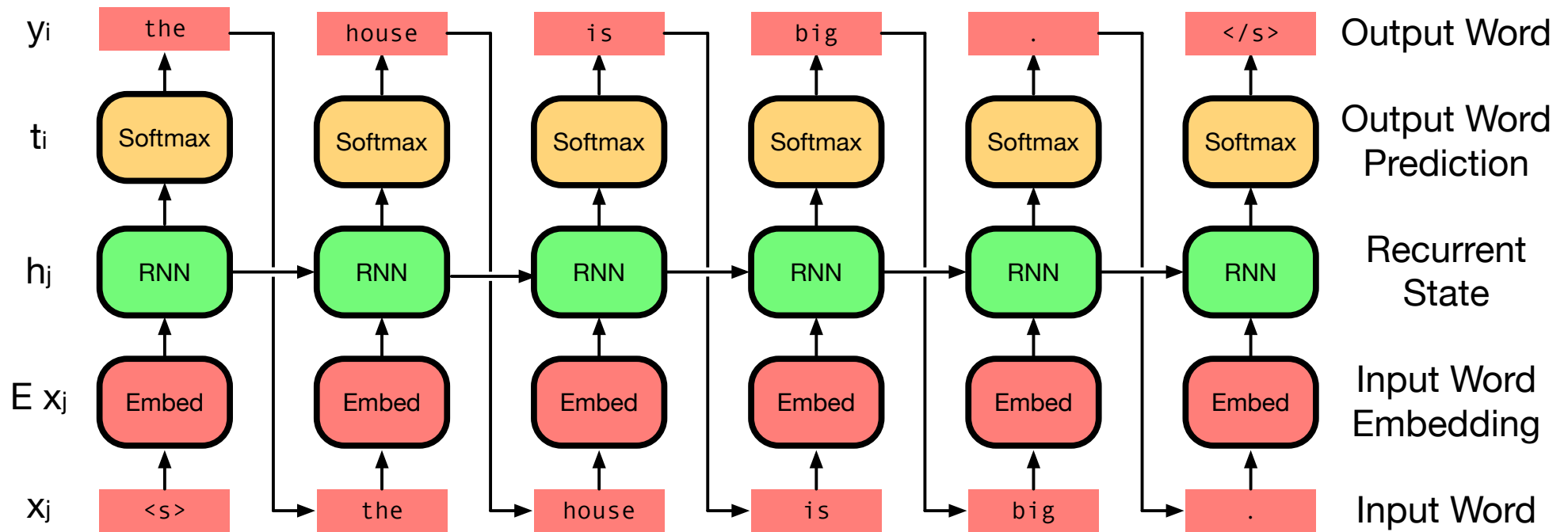$E\, x_j$ Embed   Embed   Embed   Input Word Embedding

x_j <s>   the   house   Input Word

Predict the third word of a sentence

... and so on
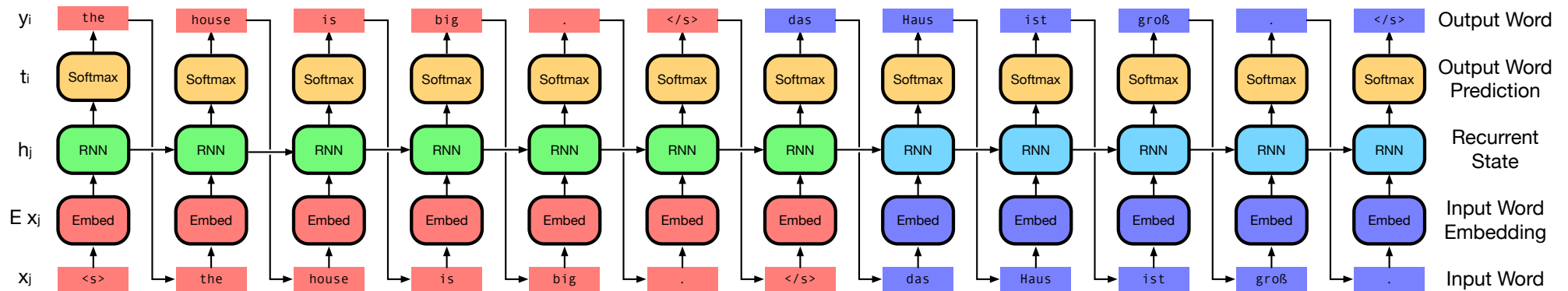
Recurrent Neural Language Model

Recurrent Neural Translation Model

- We predicted the words of a sentence

- Why not also predict their translations?

Encoder-Decoder Model

- Obviously madness
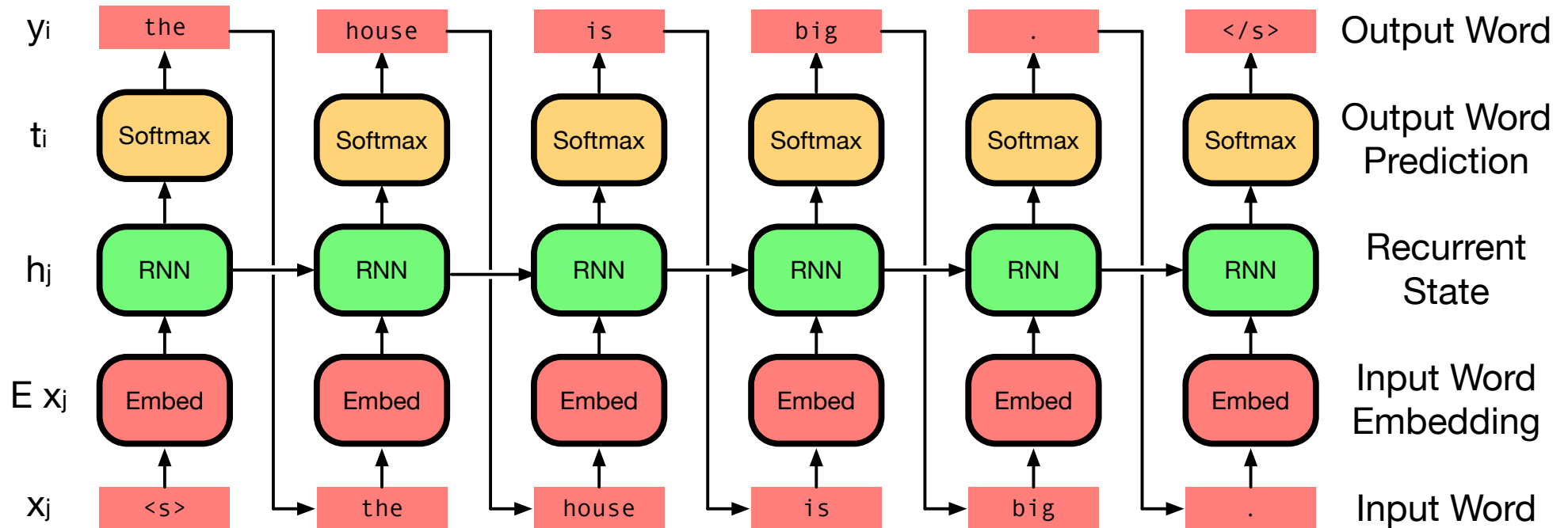
- Proposed by Google (Sutskever et al. 2014)

- Alignment of input words to output words

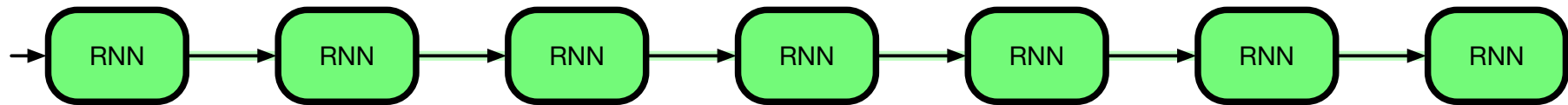$\Rightarrow$ Solution: attention mechanism

neural translation model
with attention

Input Encoding

- Inspiration: recurrent neural network language model on the input side

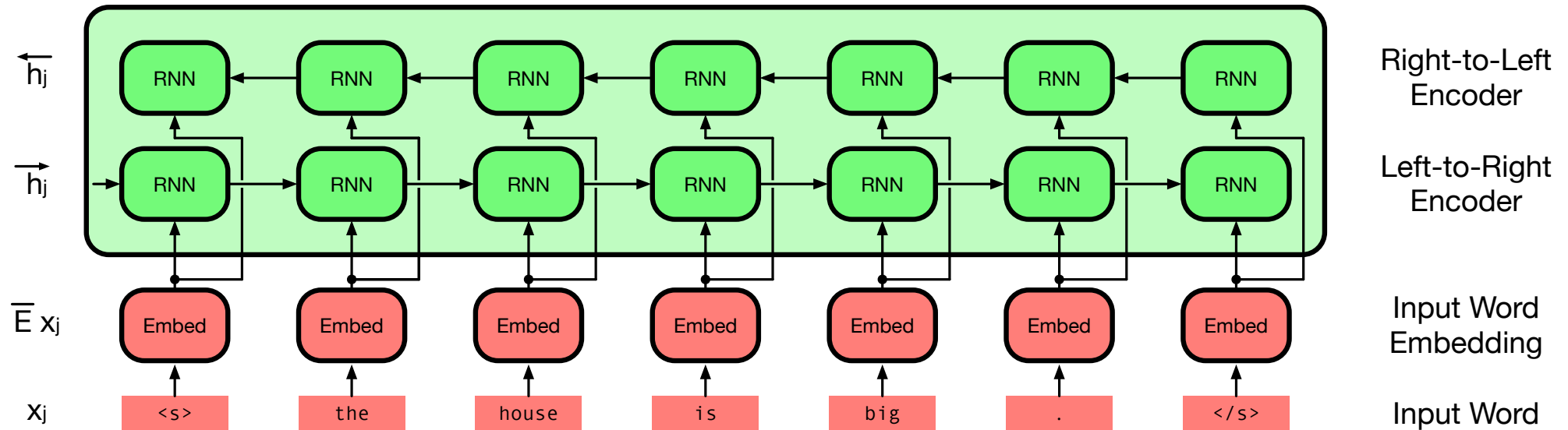- This gives us the hidden states

- These encode left context for each word

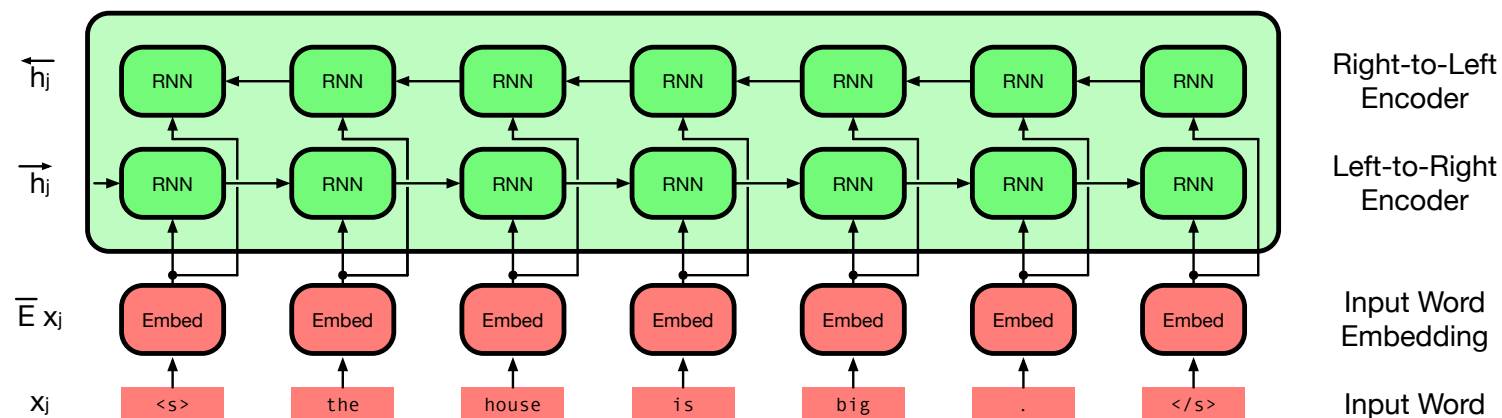- Same process in reverse: right context for each word

Input Encoder

- Input encoder: concatenate bidrectional RNN states

- Each word representation includes full left and right sentence context

Encoder: Math

- Input is sequence of words x_j, mapped into embedding space $\bar{E} \, x_j$

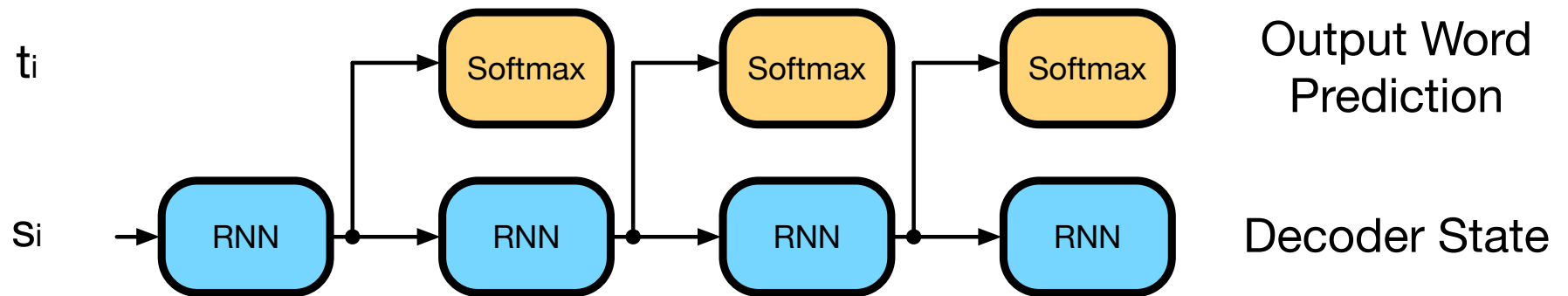- Bidirectional recurrent neural networks

$$\overleftarrow{h_j} = f(\overleftarrow{h_{j+1}}, \bar{E} \, x_j)$$
$$\overrightarrow{h_j} = f(\overrightarrow{h_{j-1}}, \bar{E} \, x_j)$$

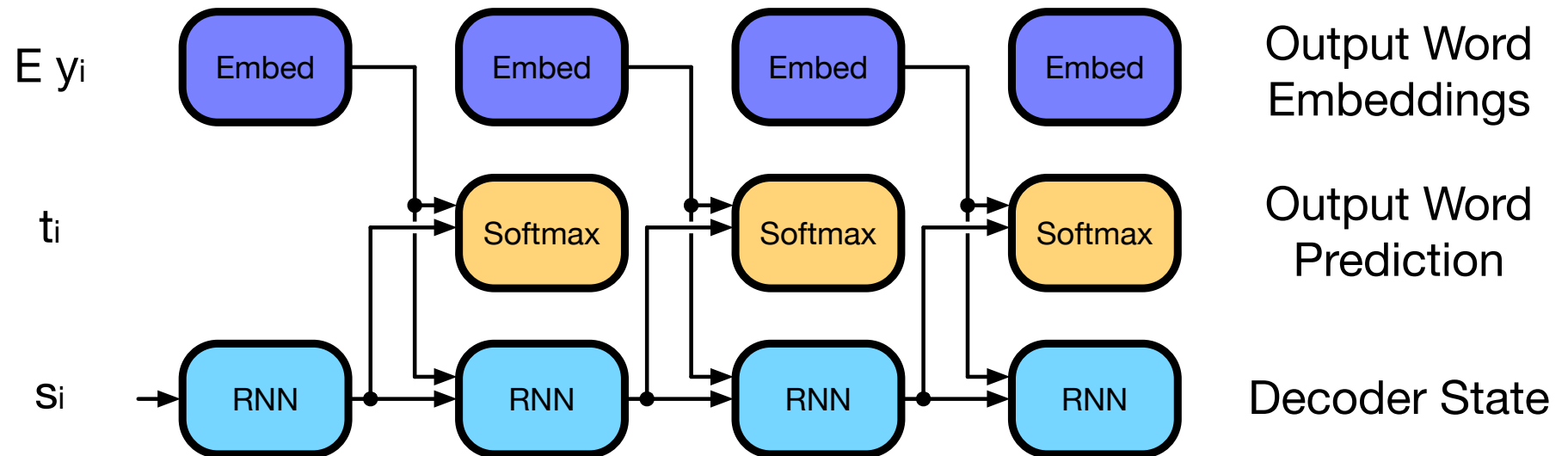- Various choices for the function $f()$: feed-forward layer, GRU, LSTM, ...

Decoder

- We want to have a recurrent neural network predicting output words

t_i Softmax   Softmax   Softmax   Output Word Prediction

s_i RNN   RNN   RNN   RNN   Decoder State
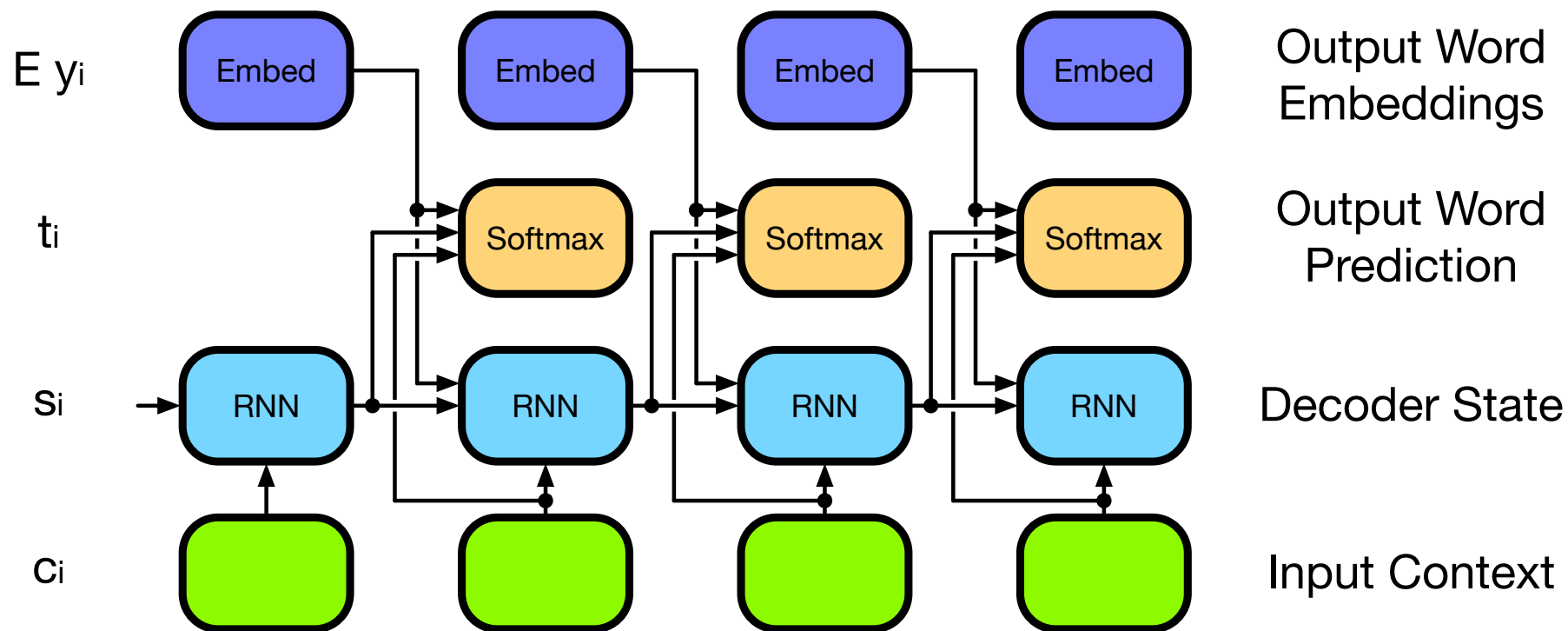
- We want to have a recurrent neural network predicting output words

- We feed decisions on output words back into the decoder state

- We want to have a recurrent neural network predicting output words

		Output Word Embeddings
$E\ y_i$	Embed	
t_i	Softmax	Output Word Prediction
s_i	RNN	Decoder State
c_i		Input Context

- We feed decisions on output words back into the decoder state
- Decoder state is also informed by the input context

E y_i	Embed ... Embed — Output Word Embeddings
y_i	\<s\> ... das — Output Word
t_i	Softmax — Output Word Prediction
s_i	RNN ... RNN — Decoder State
c_i	— Input Context
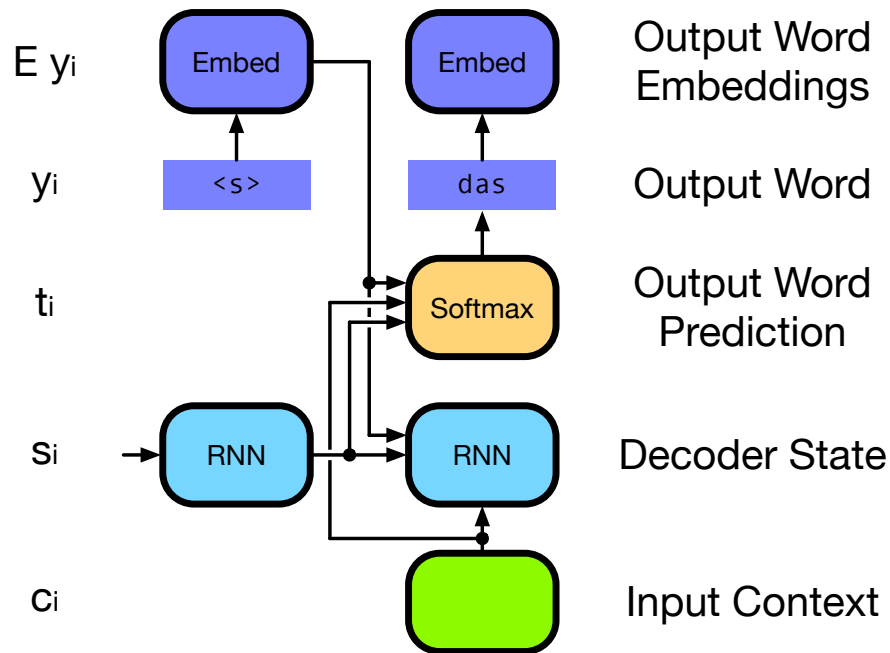
- Decoder is also recurrent neural network over sequence of hidden states s_i
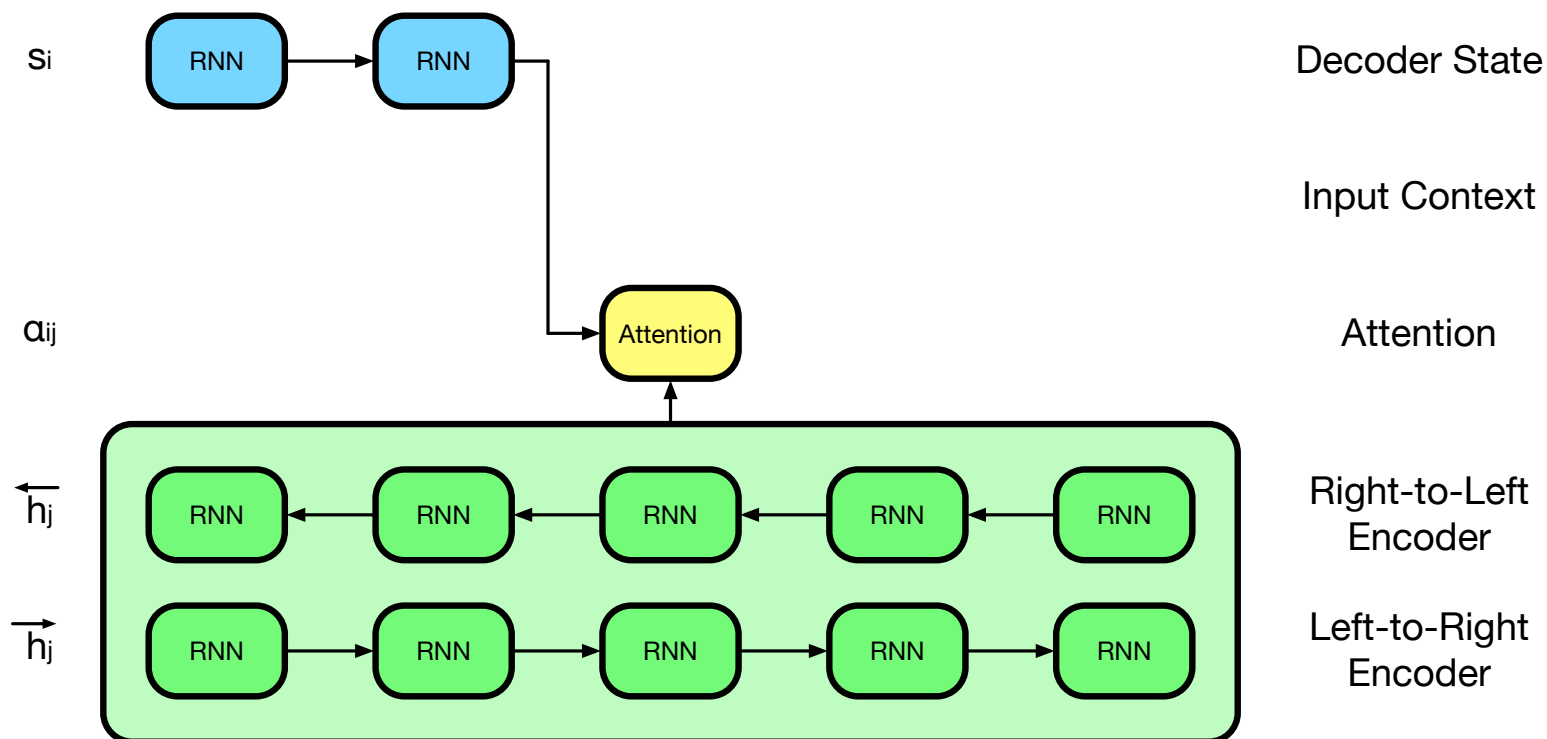
$$s_i = f(s_{i-1}, \ Ey_{-1}, c_i)$$

- Again, various choices for the function $f()$: feed-forward layer, GRU, LSTM, ...

- Output word y_i is selected by computing a vector t_i (same size as vocabulary)

$$t_i = W(Us_{i-1} + VEy_{i-1} + Cc_i)$$
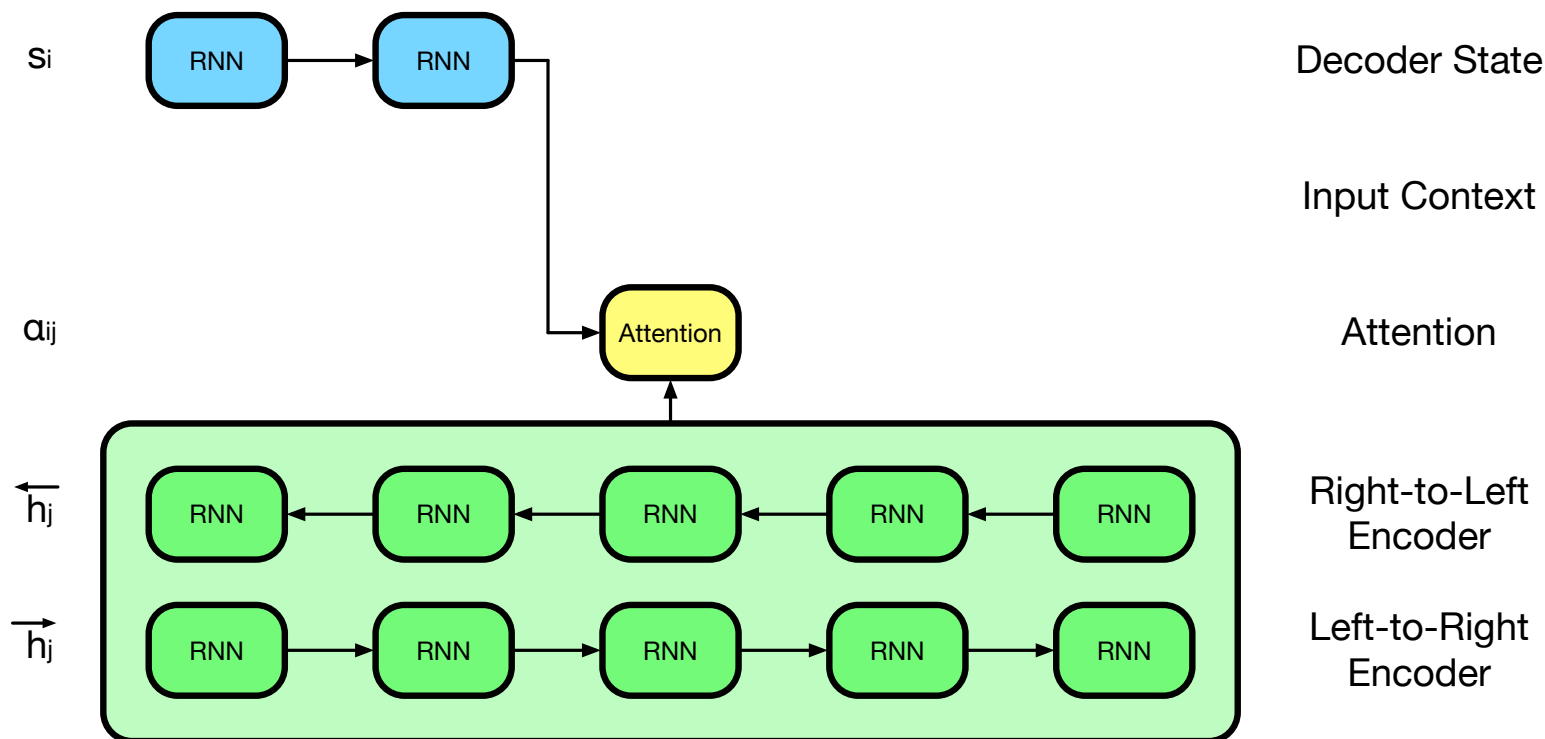
  then finding the highest value in vector t_i

- If we normalize t_i, we can view it as a probability distribution over words

- Ey_i is the embedding of the output word y_i

Attention

- Given what we have generated so far (decoder hidden state)

- ... which words in the input should we pay attention to (encoder states)?

Attention

- Given: – the previous hidden state of the decoder s_{i-1}
  – the representation of input words $h_j = (\overleftarrow{h_j}, \overrightarrow{h_j})$

- Predict an alignment probability $a(s_{i-1}, h_j)$ to each input word j (modeled with with a feed-forward neural network layer)
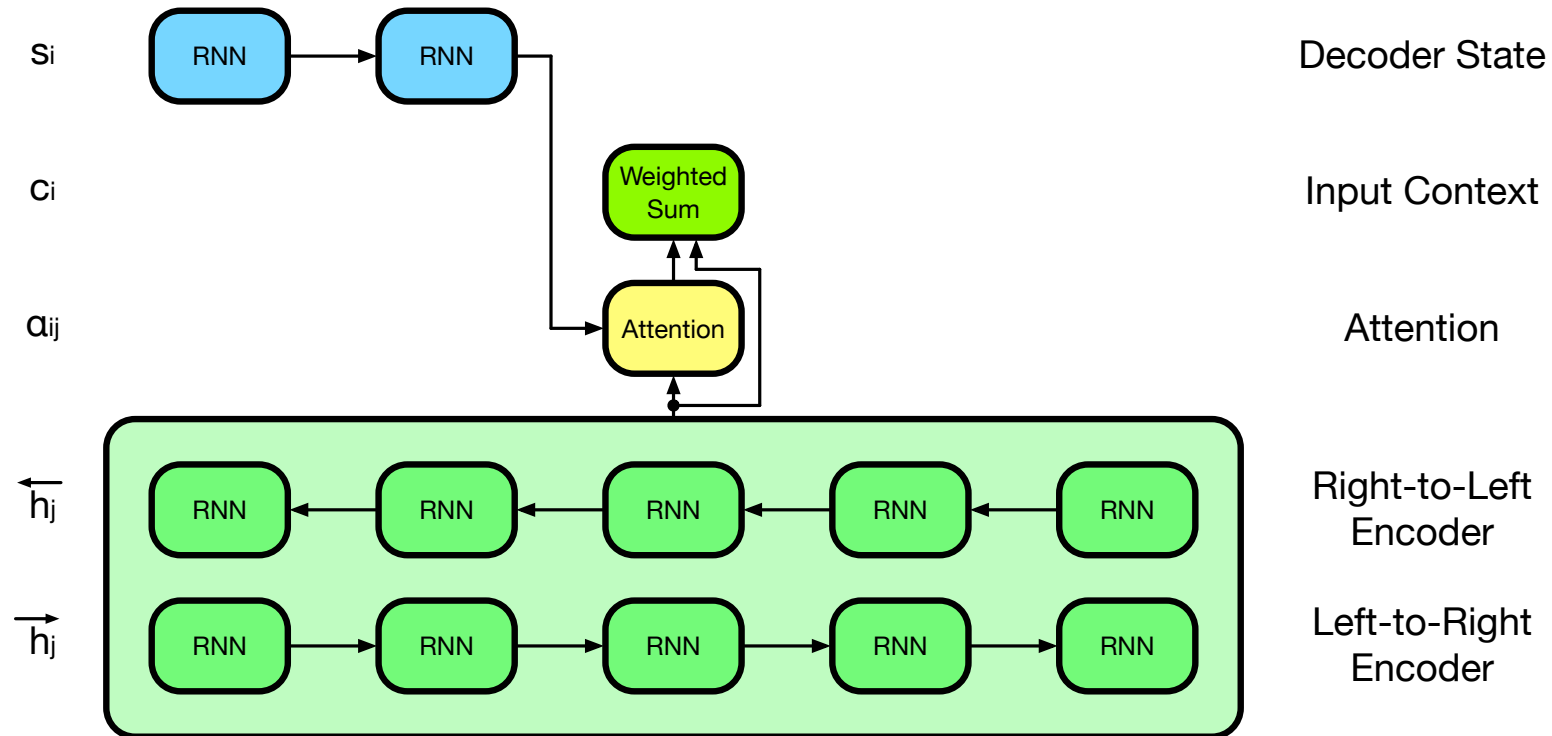
Attention
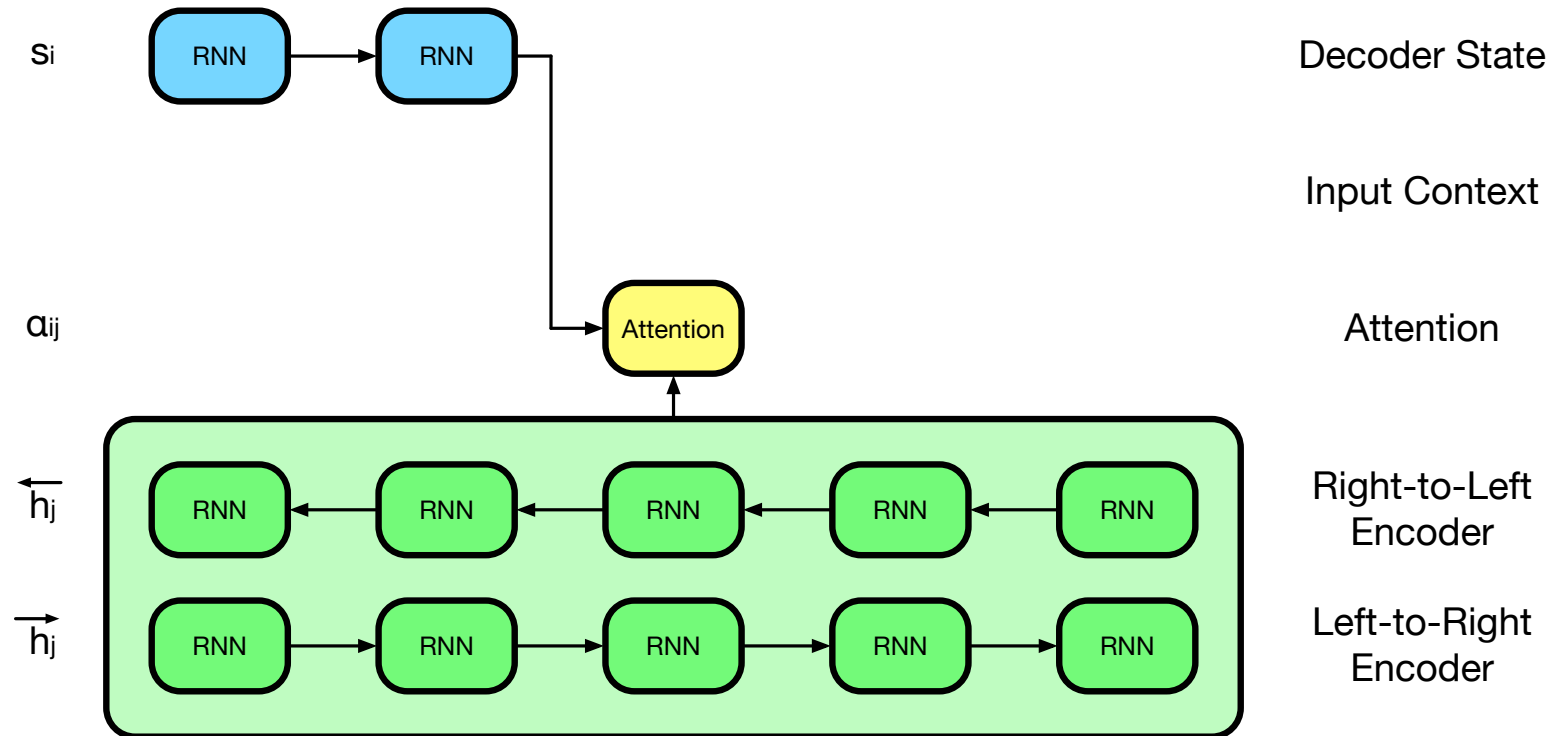
- Normalize attention (softmax)

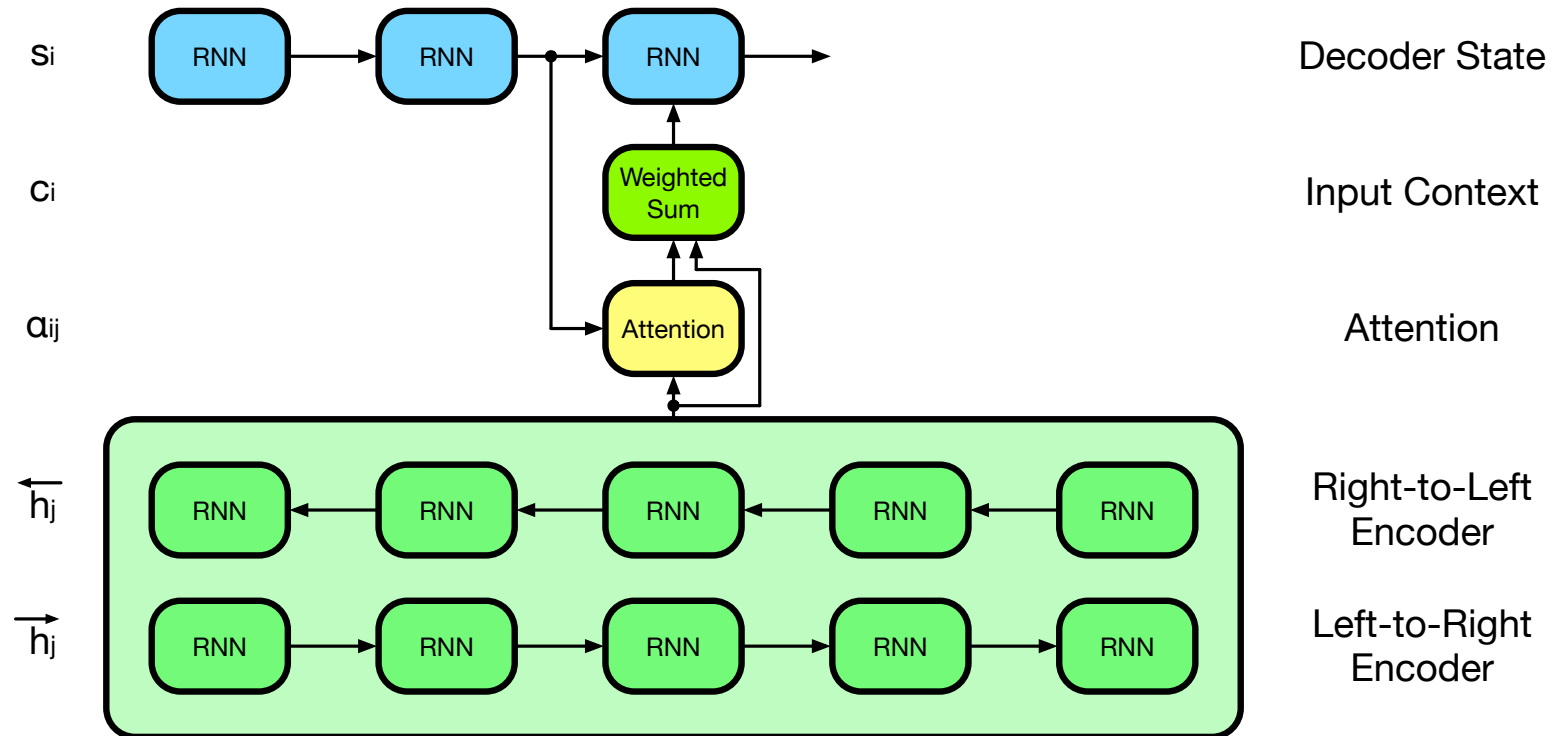$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

- Relevant input context: weigh input words according to attention: $c_i = \sum_j \alpha_{ij} h_j$

- Use context to predict next hidden state and output word

training

Comparing Prediction to Correct Word

y_i	`<s>`		`das`		`Haus`		`ist`	Output Word

$- \log t_i [y_i]$ — Cost Cost Cost — Error

t_i — Softmax Softmax Softmax — Output Word Prediction
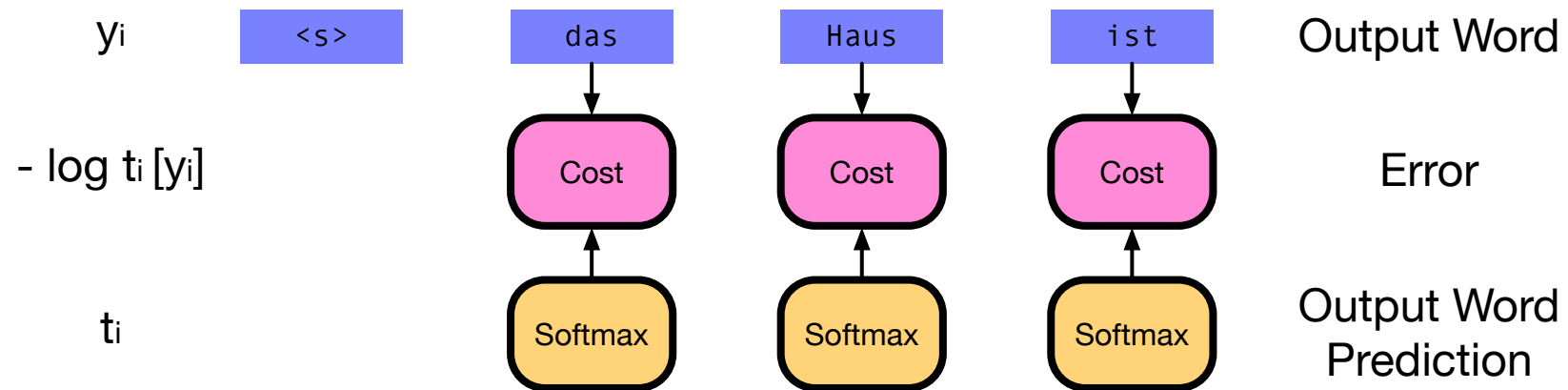
- Current model gives some probability $t_i[y_i]$ to correct word y_i

- We turn this into an error by computing cross-entropy: $-\log t_i[y_i]$

Unrolled Computation Graph

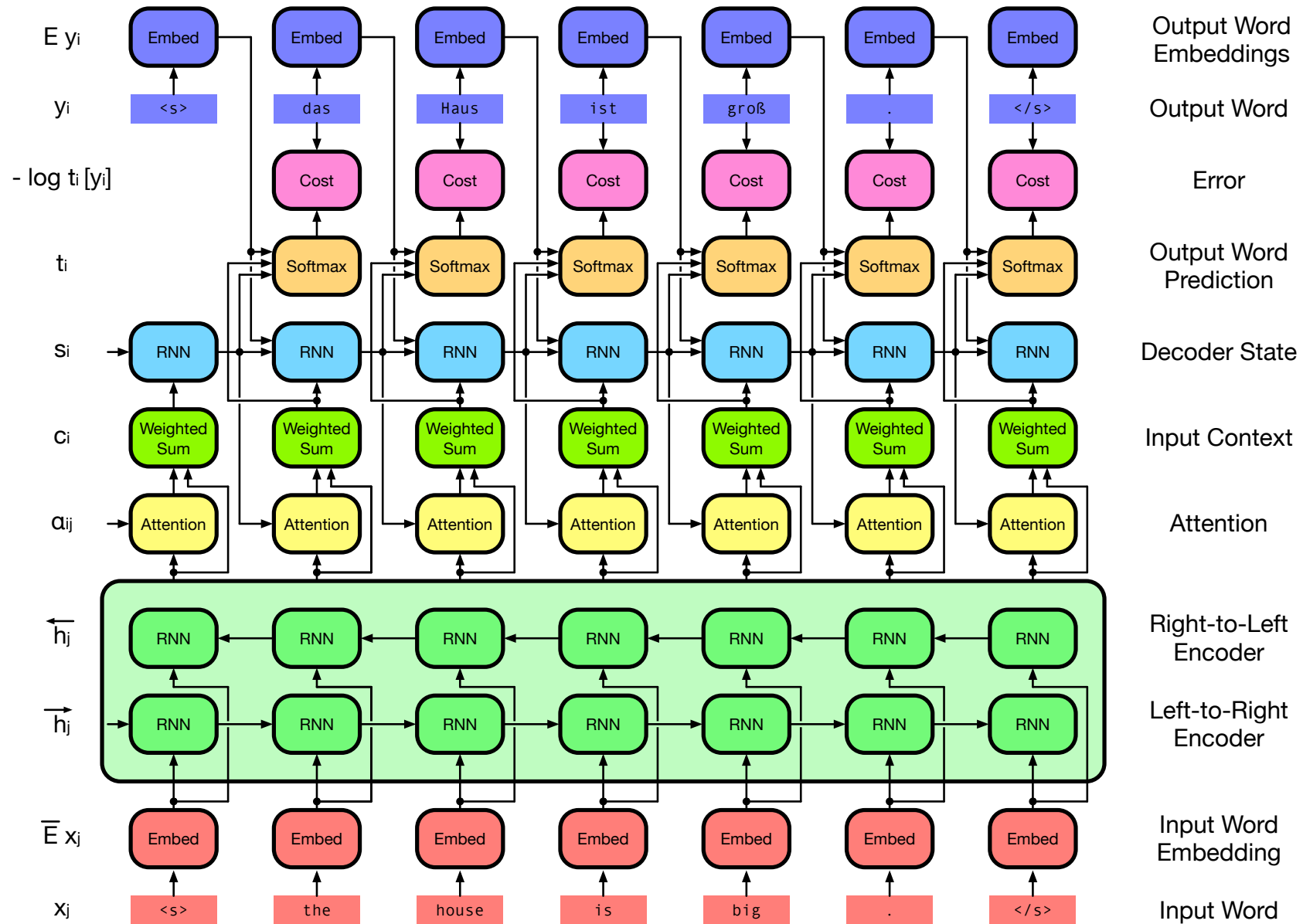|		
$E\,y_i$	Output Word Embeddings	
y_i	Output Word	
$-\log t_i\,[y_i]$	Error	
t_i	Output Word Prediction	
s_i	Decoder State	
c_i	Input Context	
α_{ij}	Attention	
$\overleftarrow{h_j}$	Right-to-Left Encoder	
$\overrightarrow{h_j}$	Left-to-Right Encoder	
$\overline{E}\,x_j$	Input Word Embedding	
x_j	Input Word	

Output words: <s>, das, Haus, ist, groß, ., </s>

Input words: <s>, the, house, is, big, ., </s>

- Encoder and decoder are recurrent neural networks

- We can add additional layers for each step

- Recall shallow and deep language models

Shallow **Deep Stacked** **Deep Transitional**
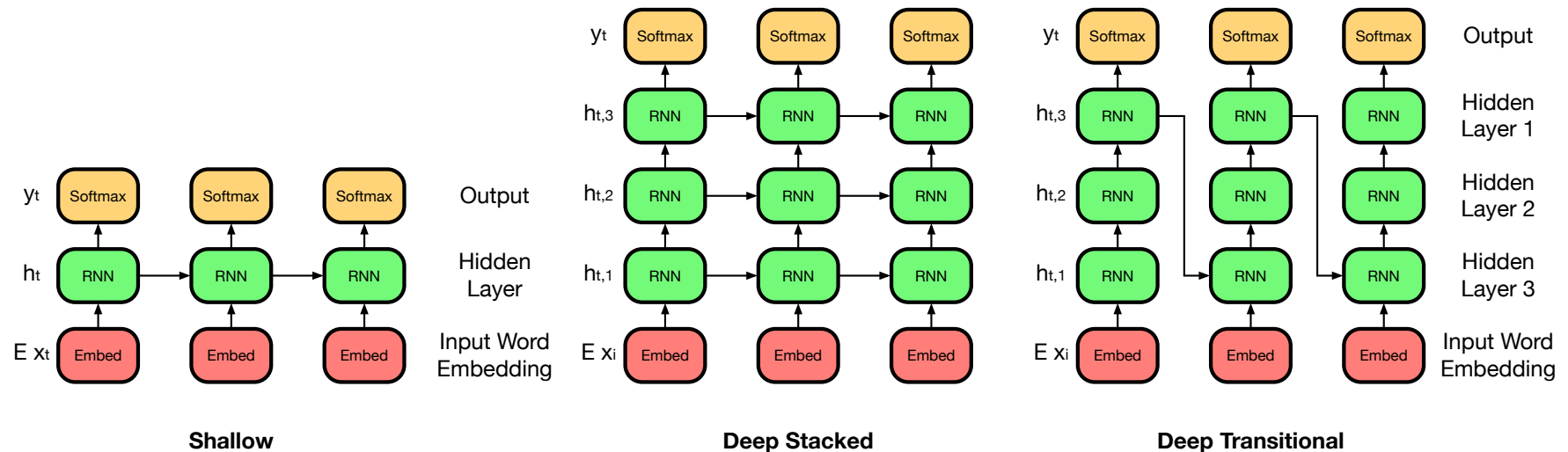
- Adding residual connections (short-cuts through deep layers) help