

---

# Neural Machine Translation

Philipp Koehn

3 October 2023



# Language Models

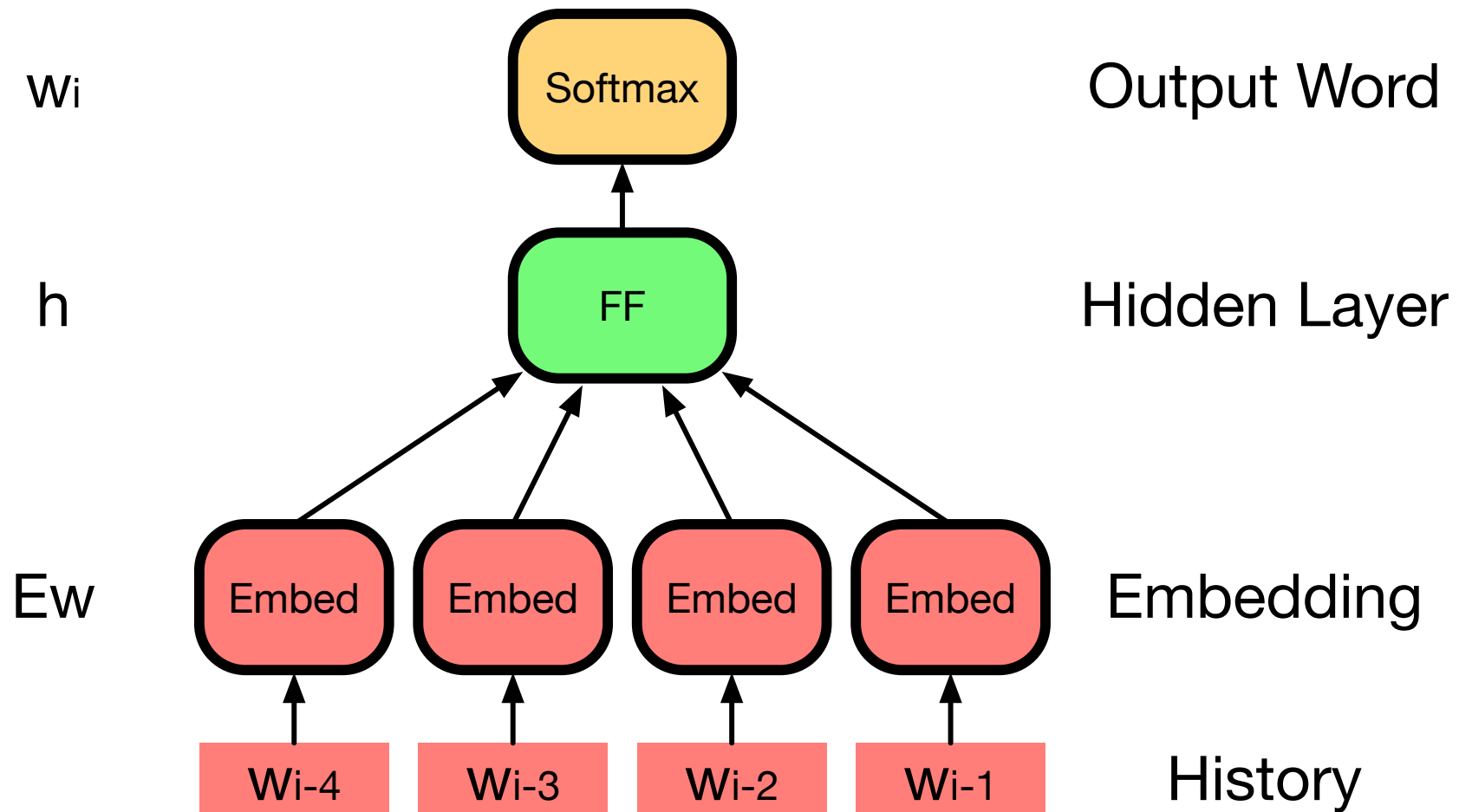


- Modeling variants
  - feed-forward neural network
  - recurrent neural network
  - long short term memory neural network
- May include input context

# Feed Forward Neural Language Model



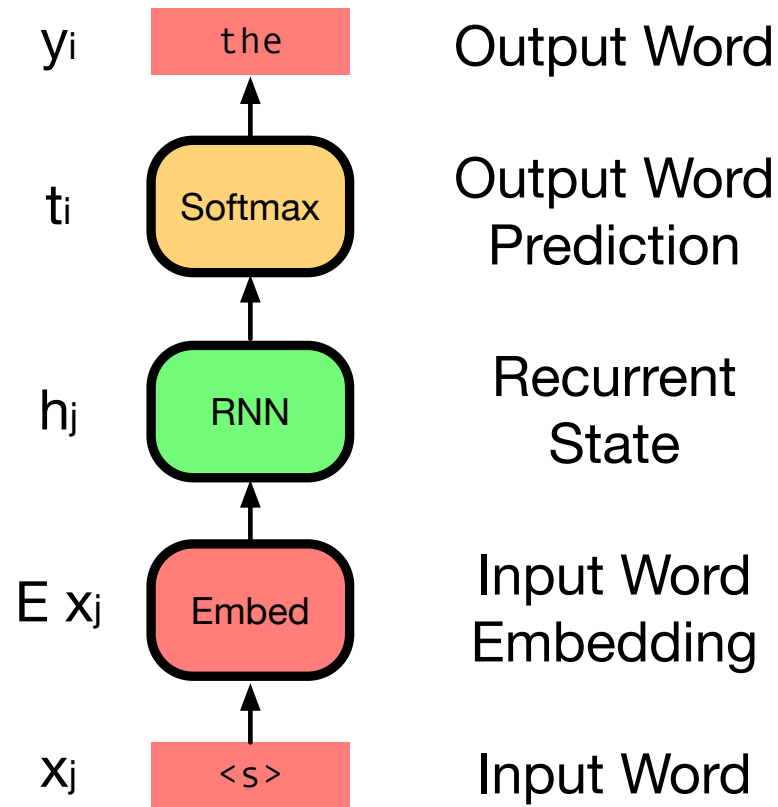
2



# Recurrent Neural Language Model

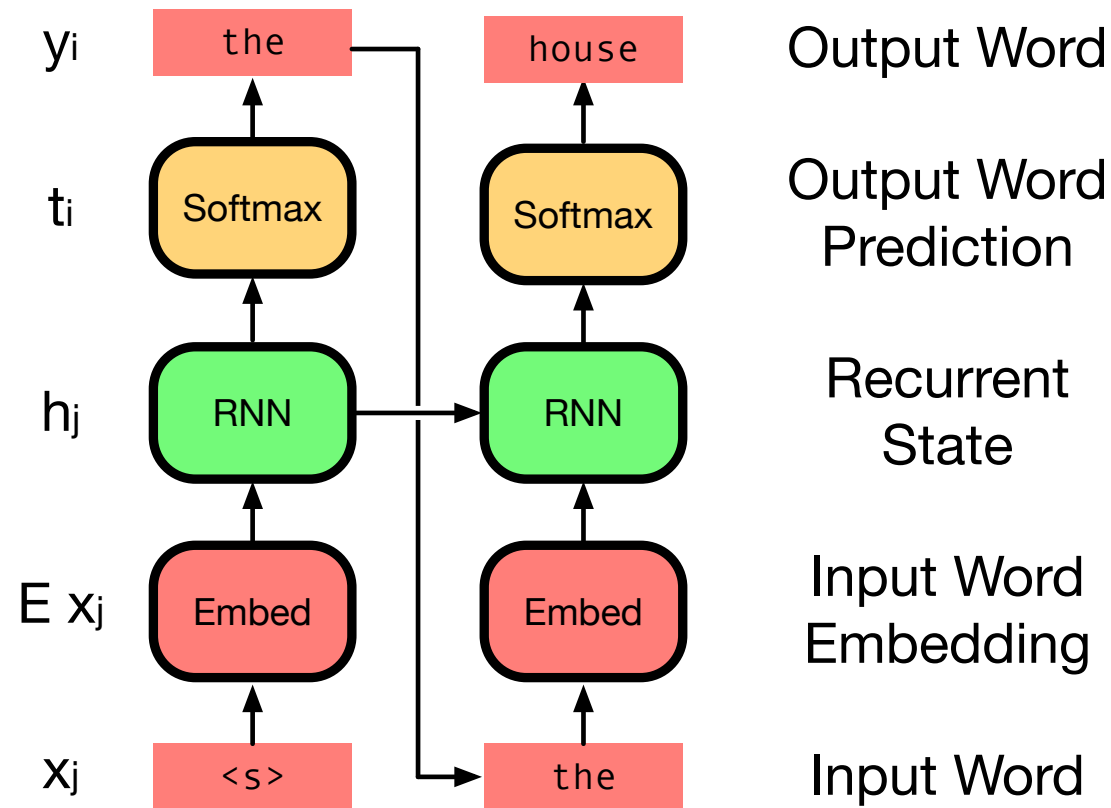


3



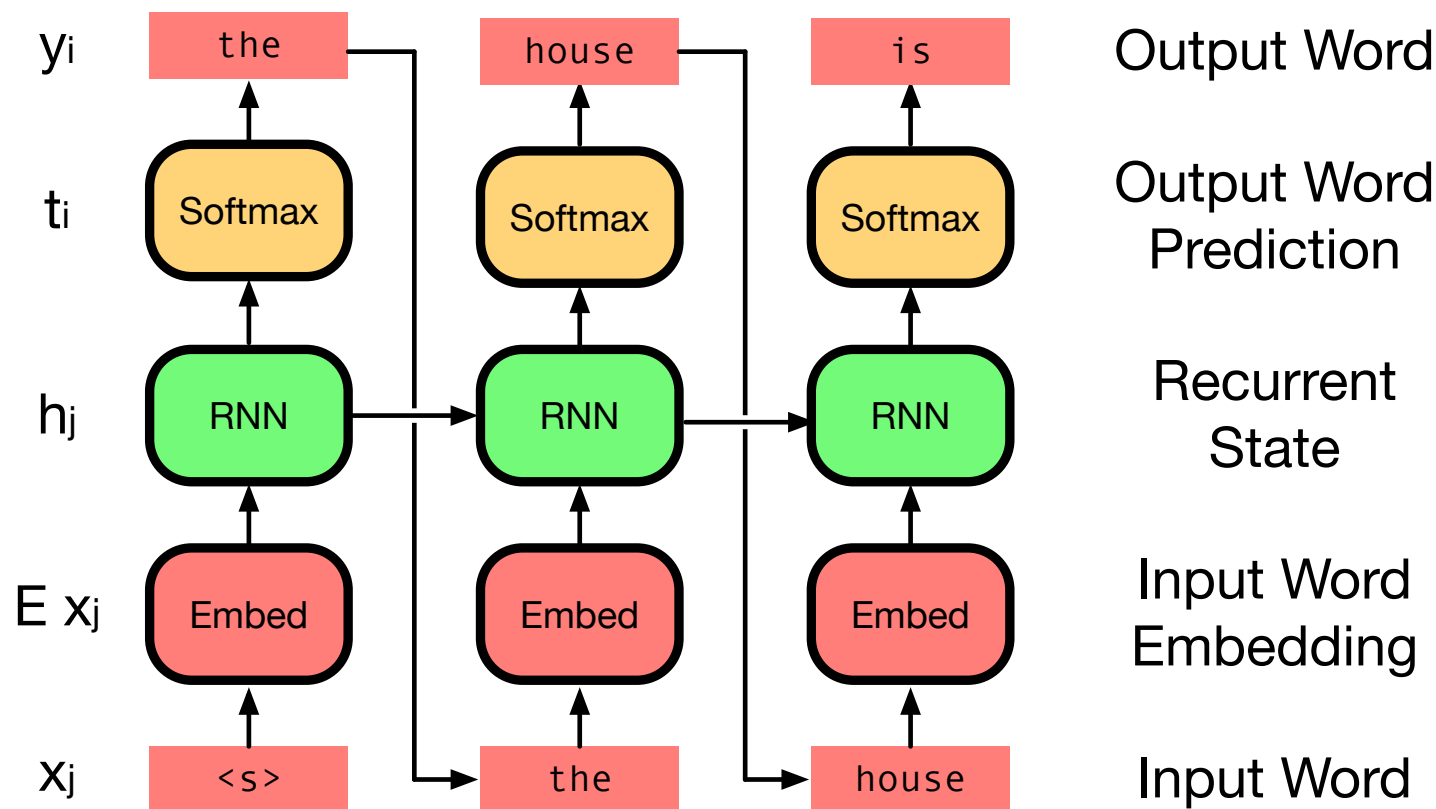
Predict the first word of a sentence

# Recurrent Neural Language Model



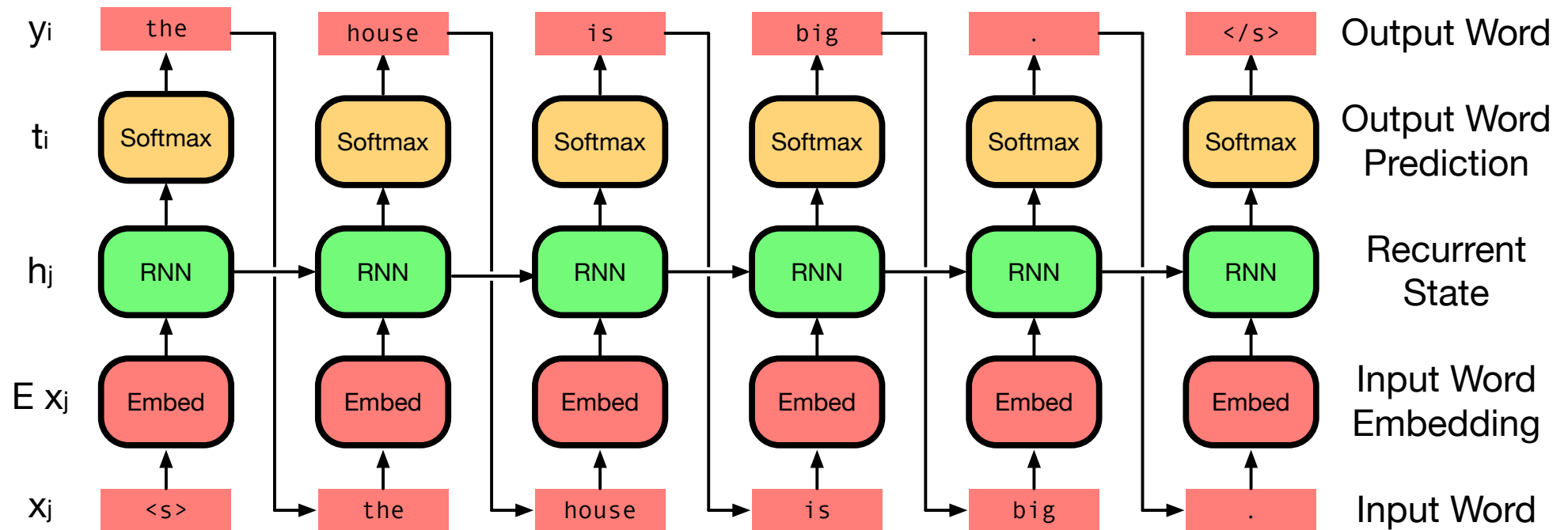
Predict the second word of a sentence  
Re-use hidden state from first word prediction

# Recurrent Neural Language Model



Predict the third word of a sentence  
... and so on

# Recurrent Neural Language Model



# Recurrent Neural Translation Model

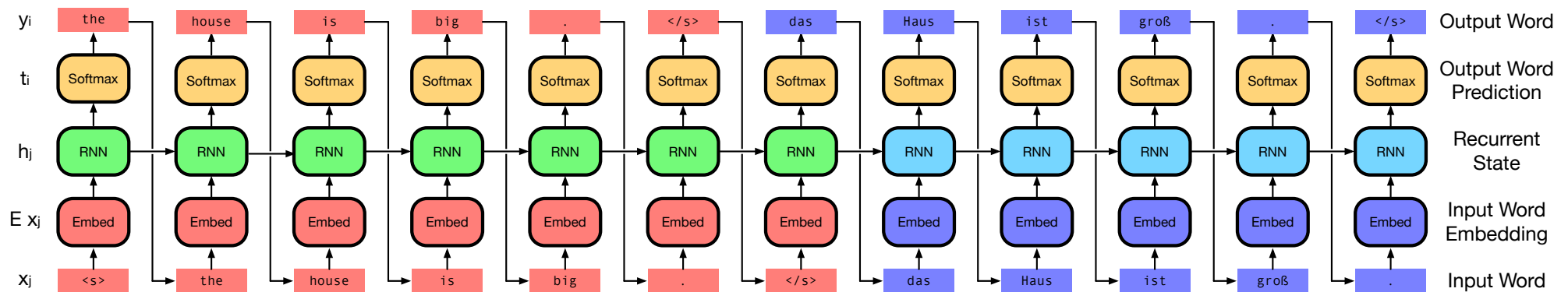


7

- 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)

# What is Missing?

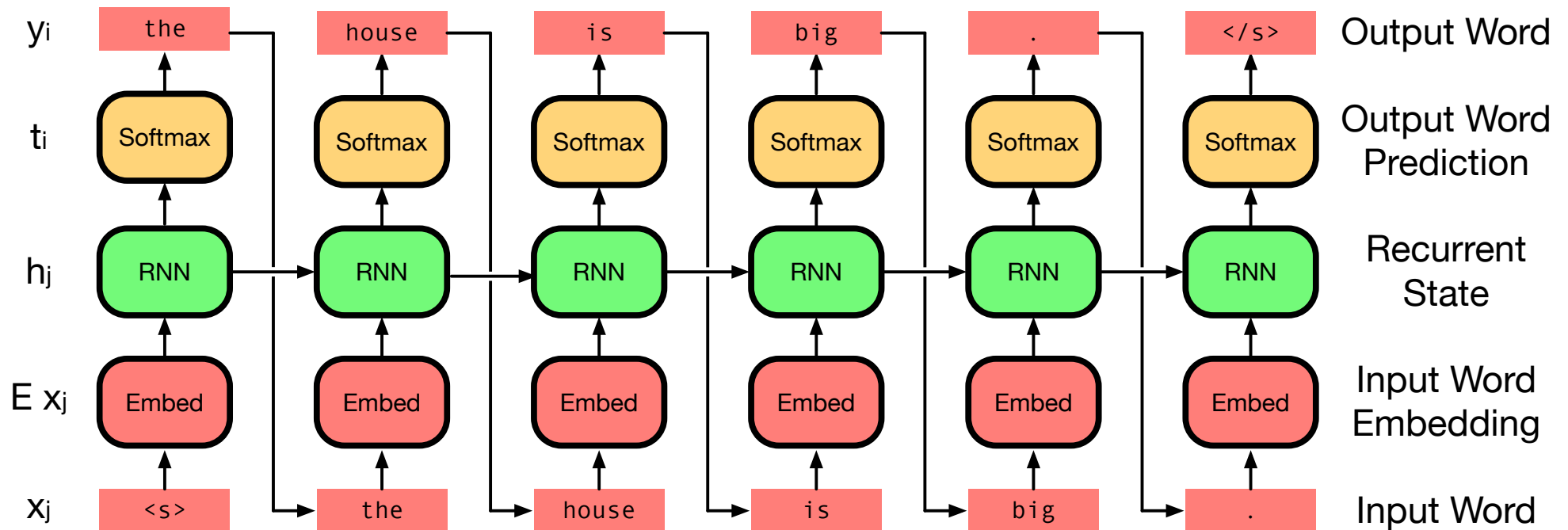


- Alignment of input words to output words

⇒ Solution: attention mechanism

# neural translation model with attention

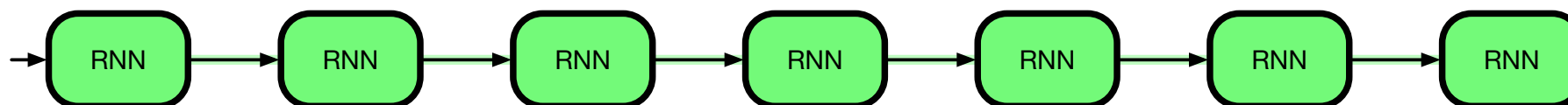
# Input Encoding



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

# Hidden Language Model States

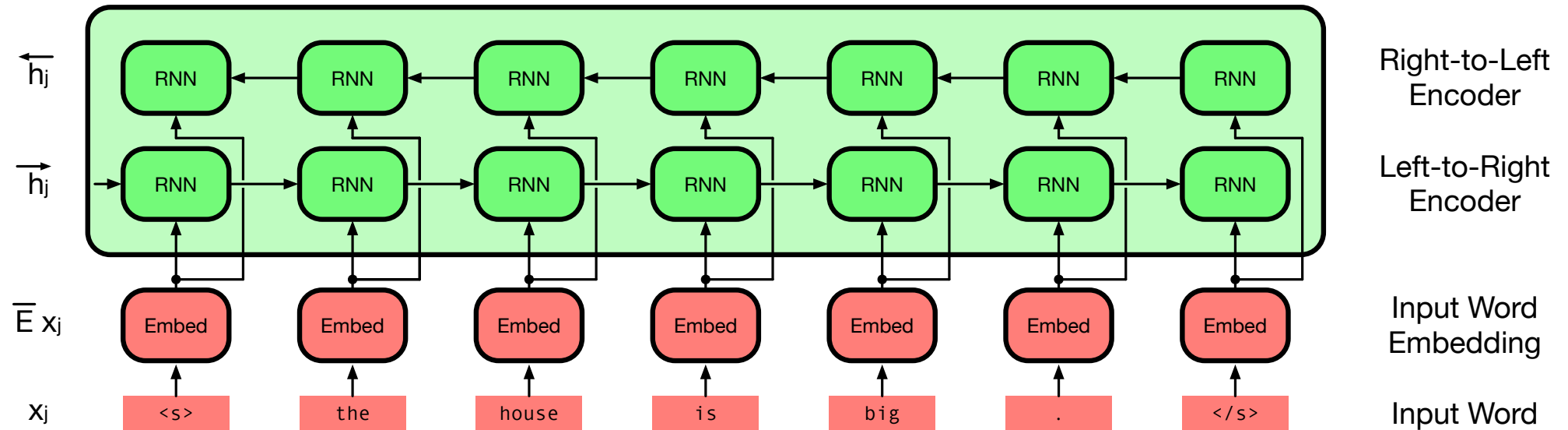
- This gives us the hidden states



- These encode left context for each word
- Same process in reverse: right context for each word

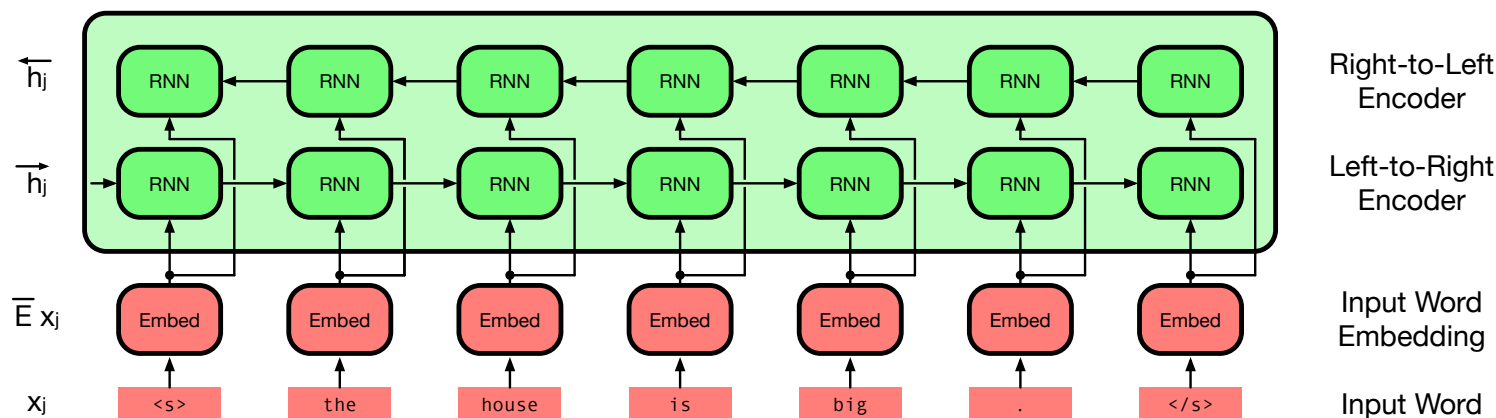


# Input Encoder



- Input encoder: concatenate bidirectional RNN states
- Each word representation includes full left and right sentence context

# Encoder: Math



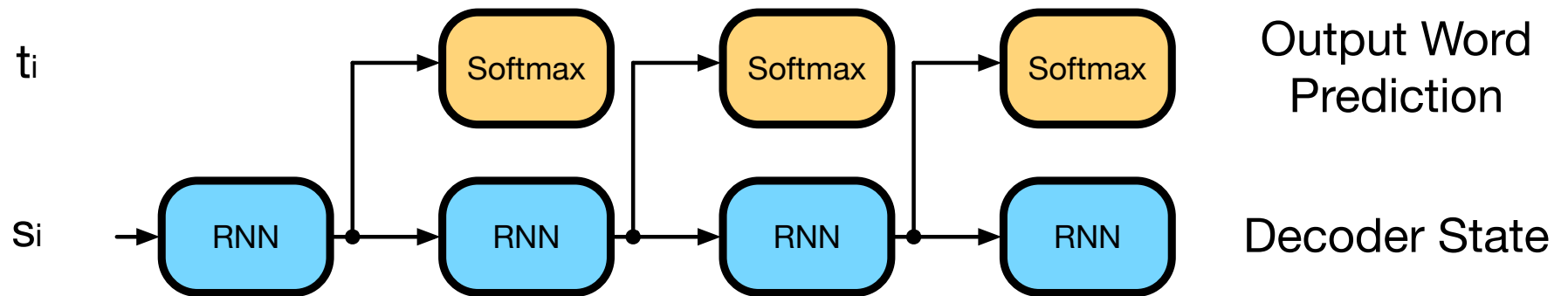
- Input is sequence of words  $x_j$ , mapped into embedding space  $\bar{E} x_j$
- Bidirectional recurrent neural networks

$$\begin{aligned}\overleftarrow{h}_j &= f(\overleftarrow{h}_{j+1}, \bar{E} x_j) \\ \vec{h}_j &= f(\vec{h}_{j-1}, \bar{E} x_j)\end{aligned}$$

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

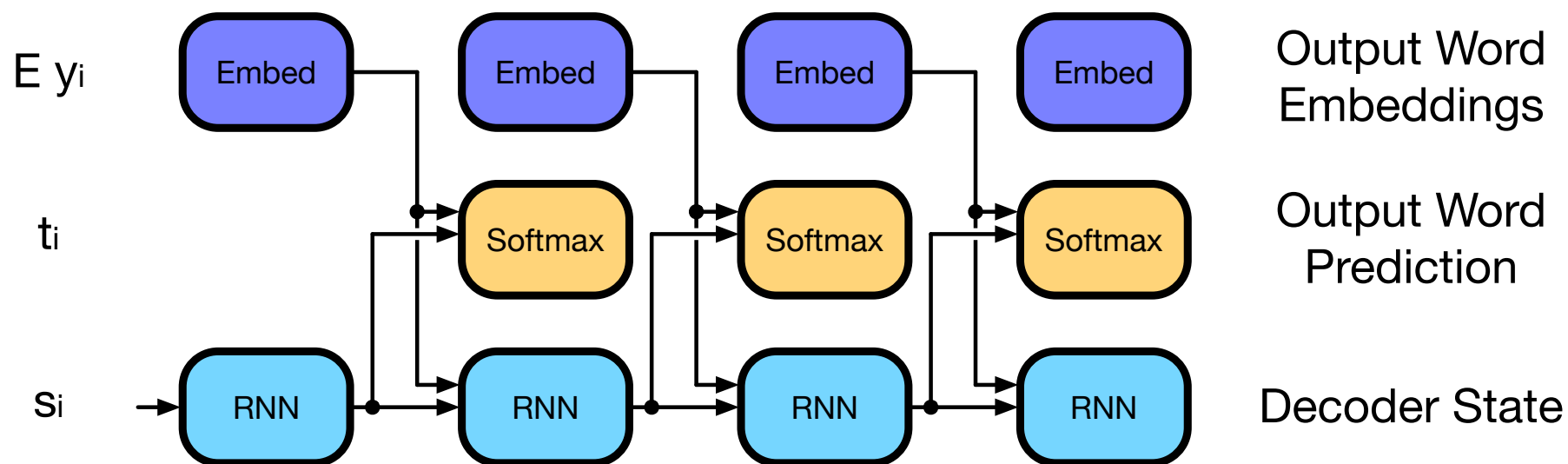
# Decoder

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



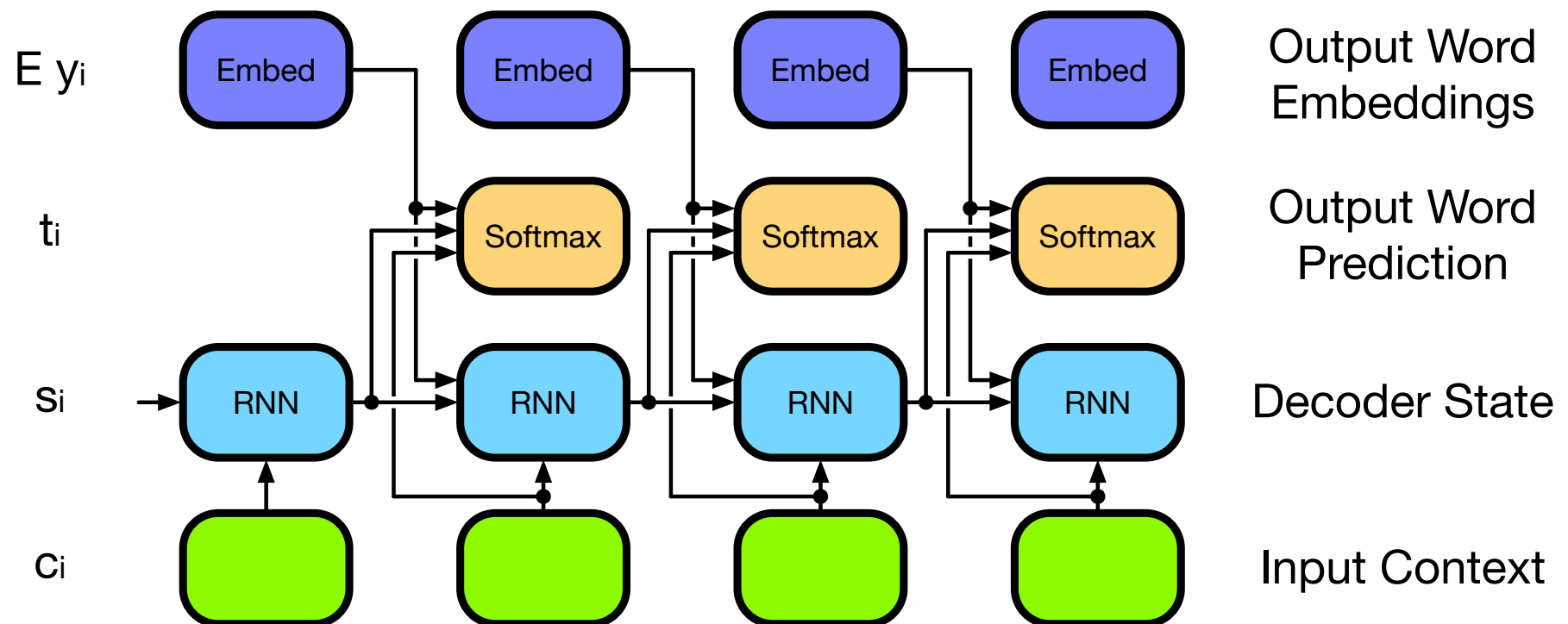


- We want to have a recurrent neural network predicting output words

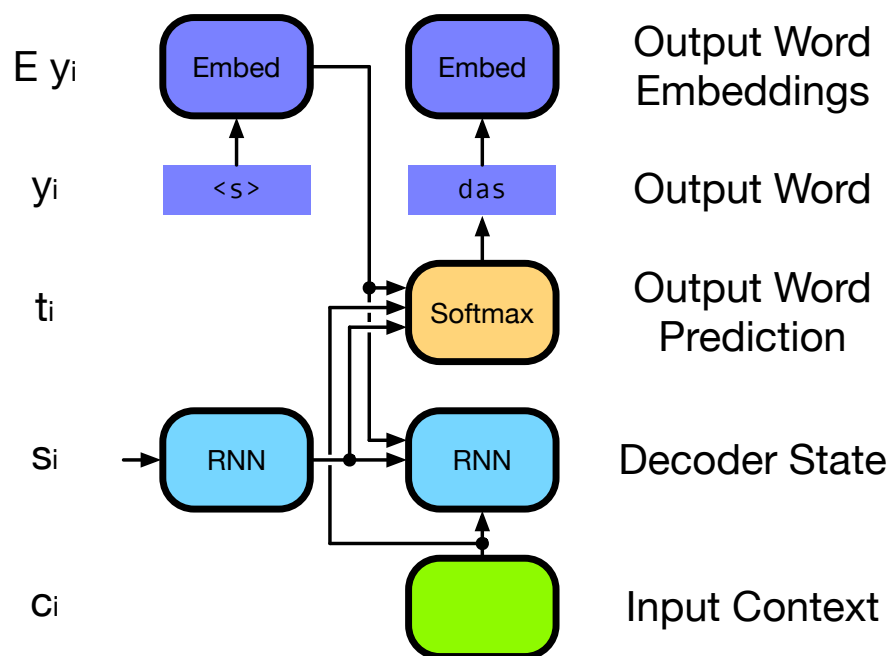


- We feed decisions on output words back into the decoder state

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



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



- Decoder is also recurrent neural network over sequence of hidden states  $s_i$

$$s_i = f(s_{i-1}, E y_{i-1}, c_i)$$

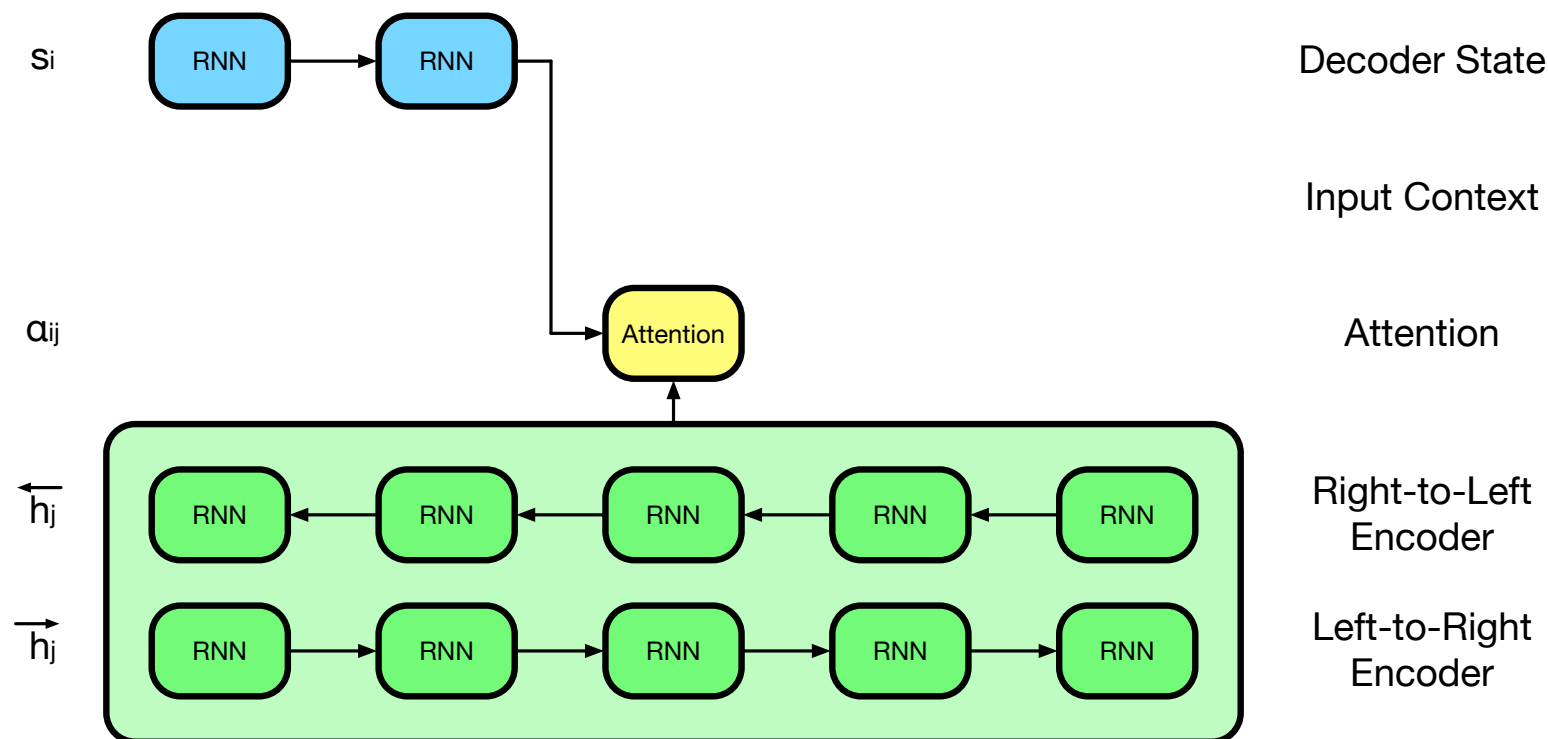
- Again, various choices for the function  $f()$ : feed-forward layer, GRU, LSTM, ...
- Output word  $y_i$  is selected by computing a vector  $t_i$  (same size as vocabulary)

$$t_i = W(U s_{i-1} + V E y_{i-1} + C c_i)$$

then finding the highest value in vector  $t_i$

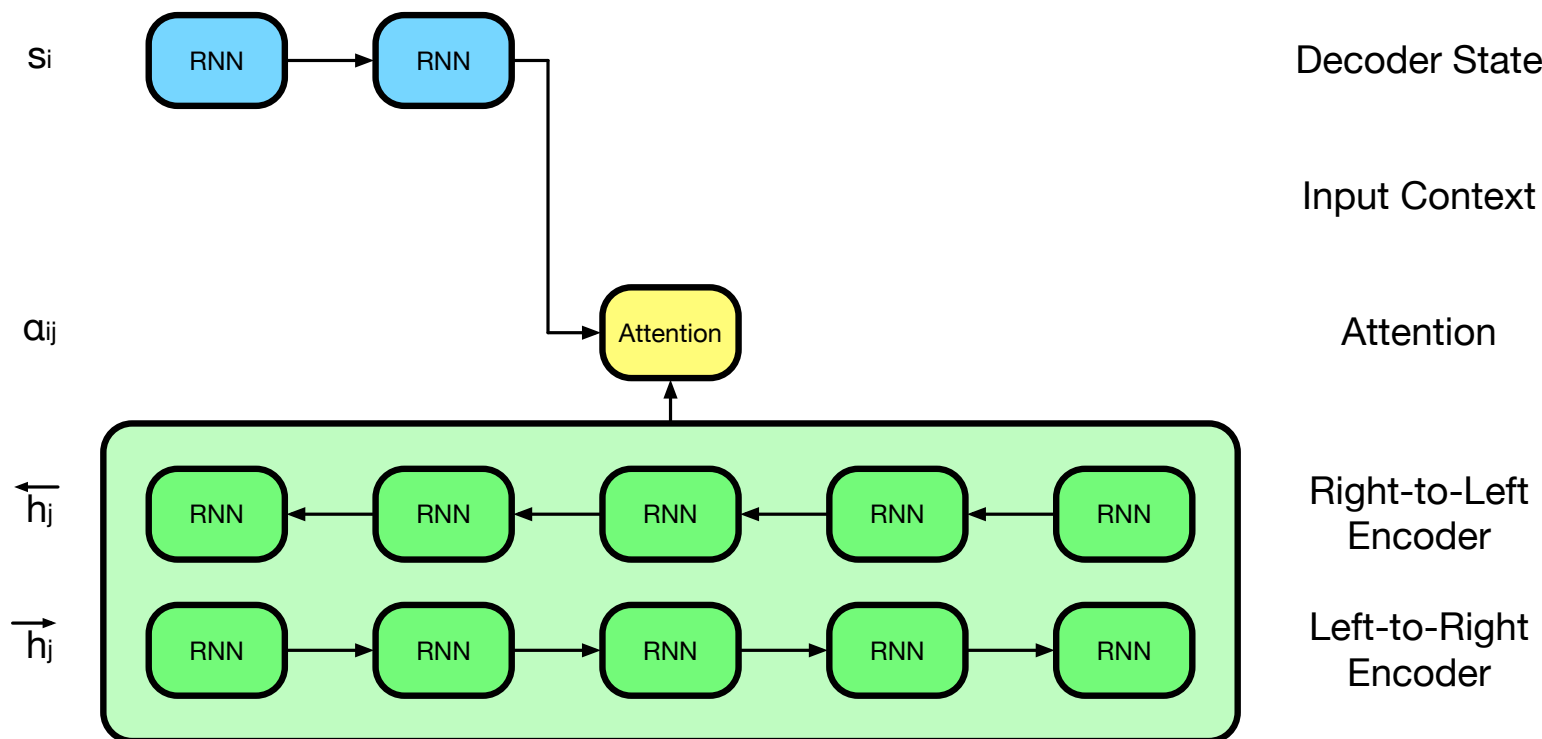
- If we normalize  $t_i$ , we can view it as a probability distribution over words
- $E y_i$  is the embedding of the output word  $y_i$

# Attention



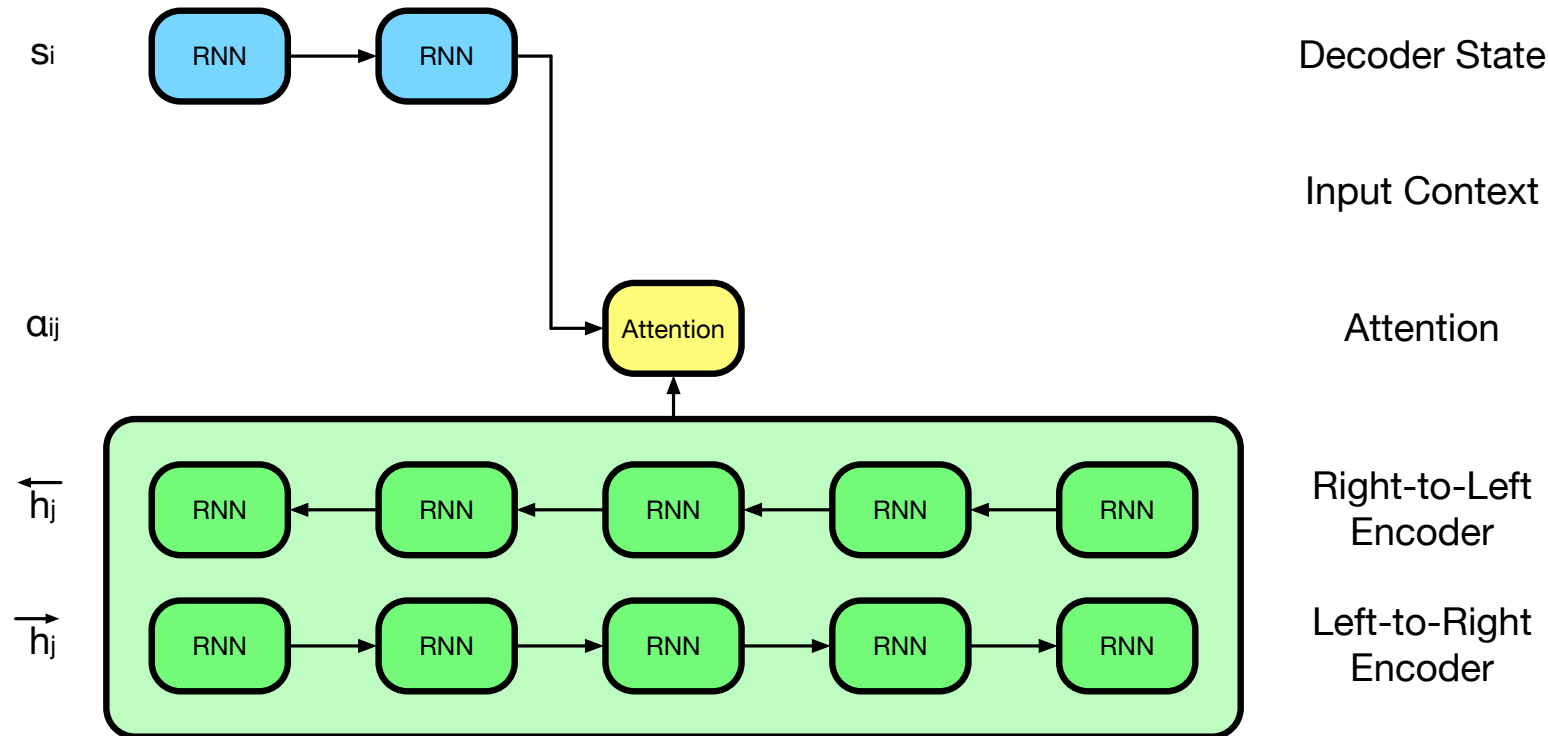
- Given what we have generated so far (decoder hidden state)
- ... which words in the input should we pay attention to (encoder states)?

# 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)

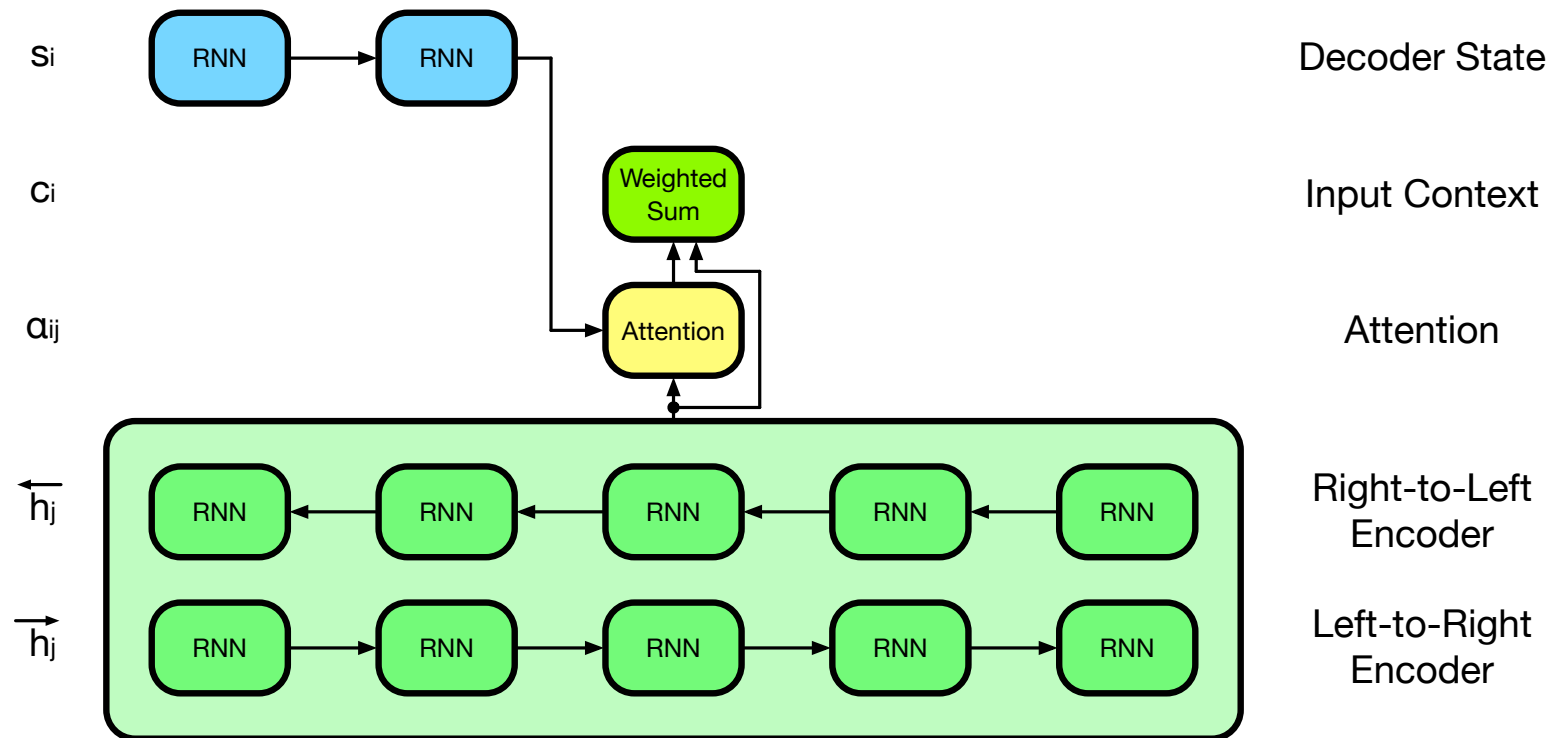
# Attention



- Normalize attention (softmax)

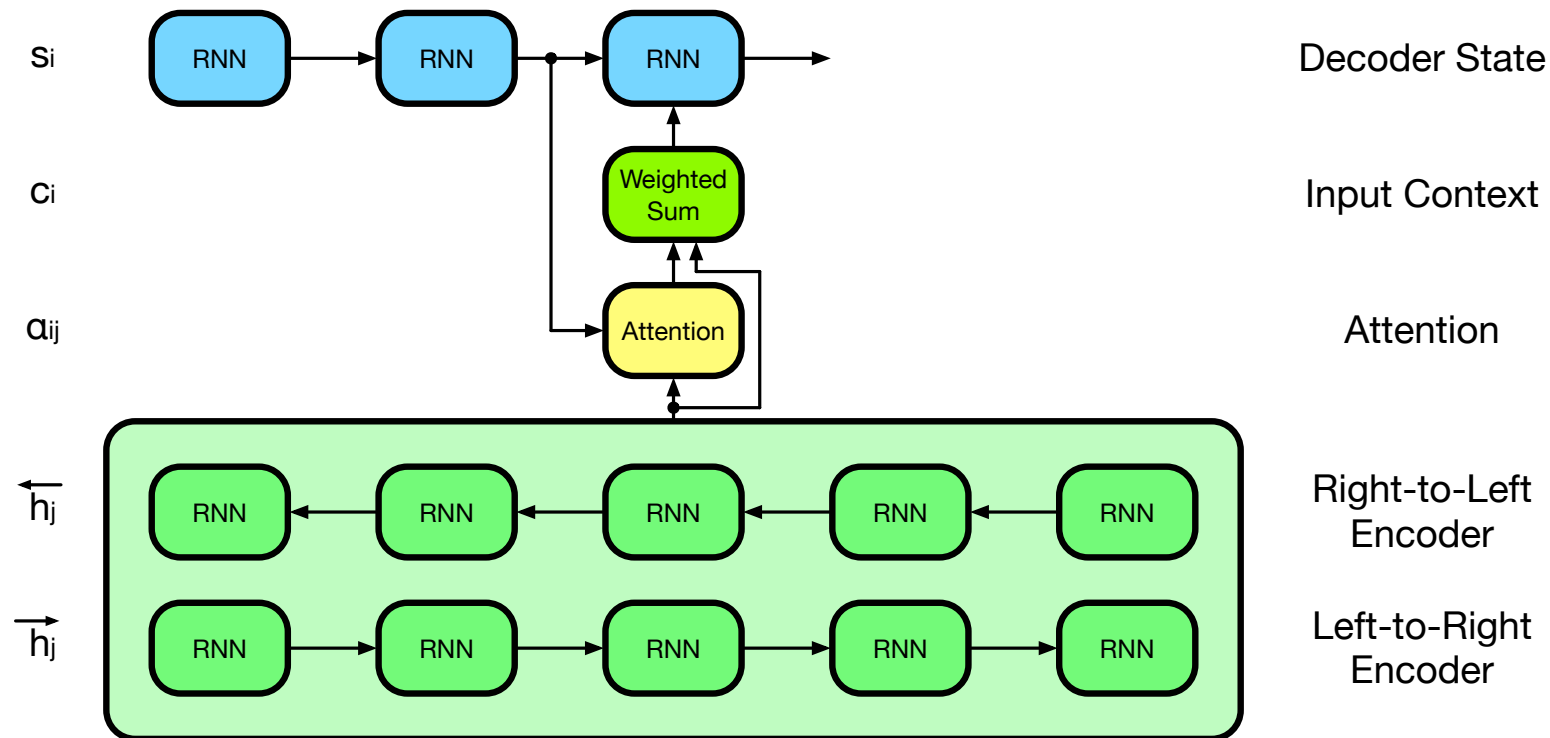
$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

# Attention



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

# Attention

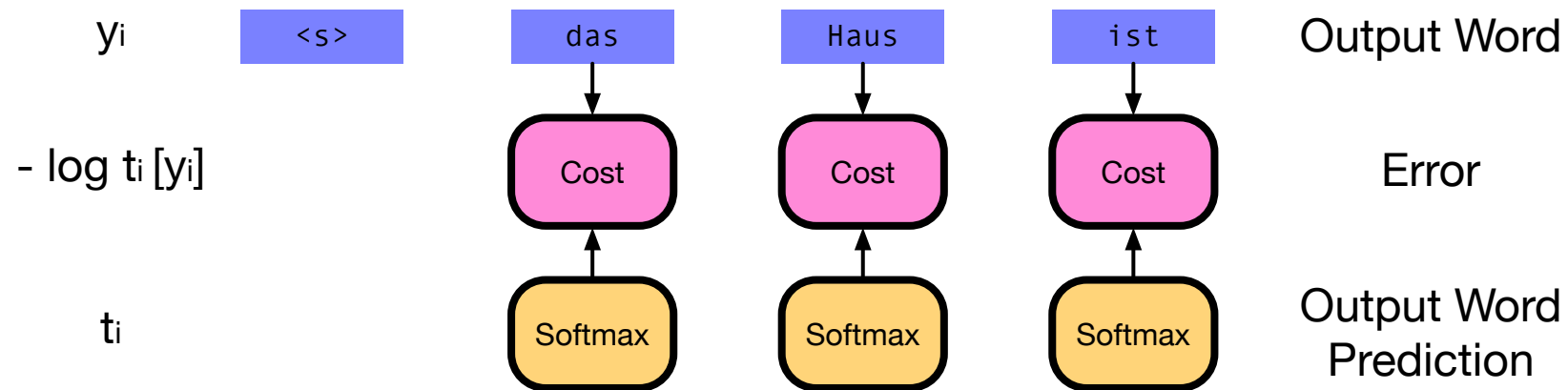


- Use context to predict next hidden state and output word



# training

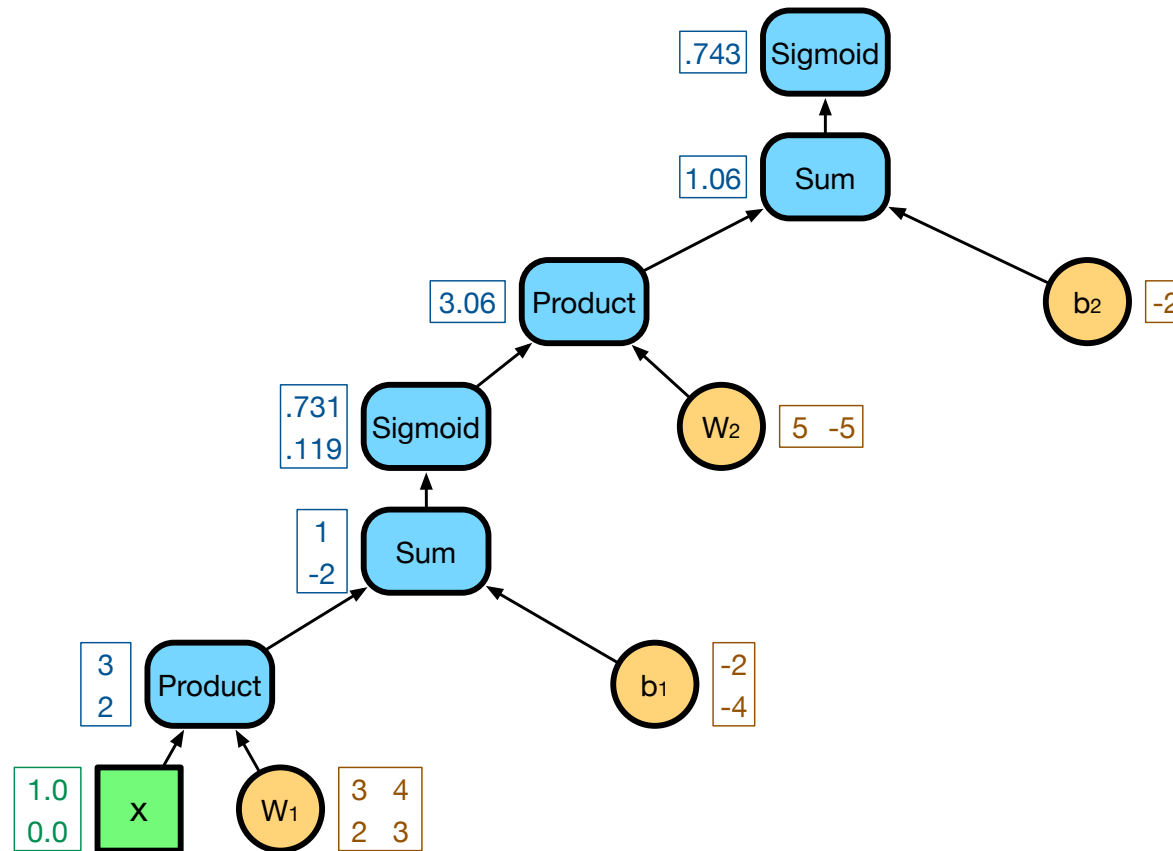
# Comparing Prediction to Correct Word



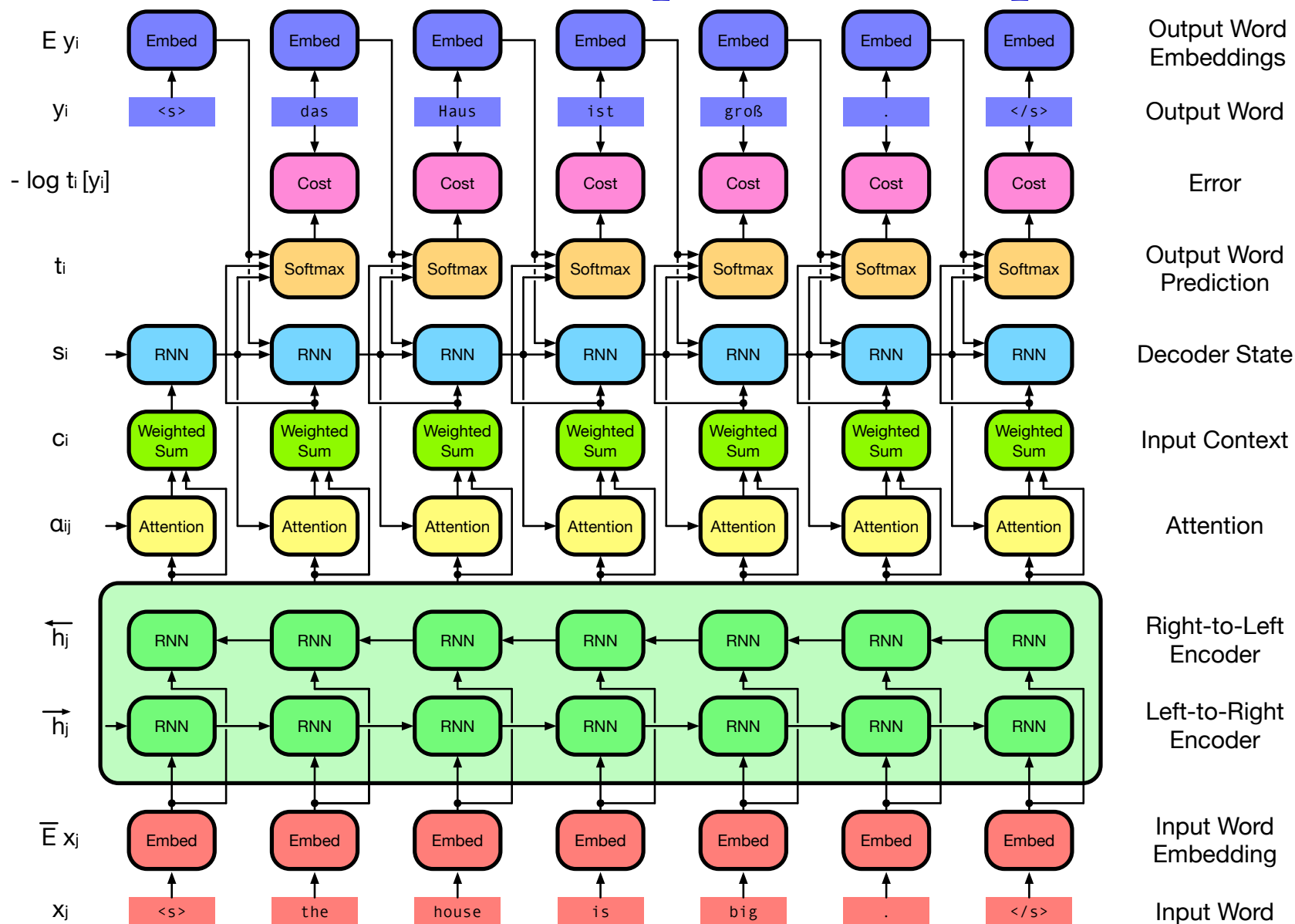
- Current model gives some probability  $t_i[y_i]$  to correct word  $y_i$
- We turn this into an error by computing cross-entropy:  $-\log t_i[y_i]$

# Computation Graph

- Math behind neural machine translation defines a computation graph
- Forward and backward computation to compute gradients for model training



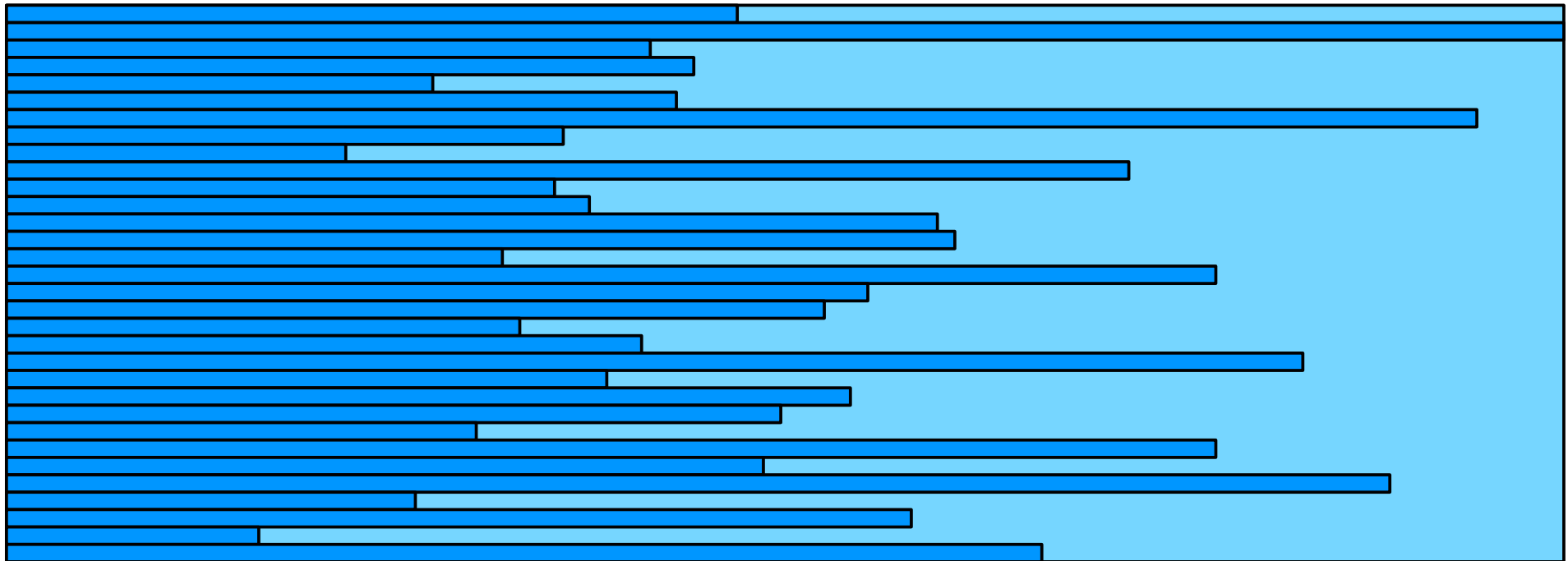
# Unrolled Computation Graph



- Already large degree of parallelism
  - most computations on vectors, matrices
  - efficient implementations for CPU and GPU
- Further parallelism by batching
  - processing several sentence pairs at once
  - scalar operation  $\rightarrow$  vector operation
  - vector operation  $\rightarrow$  matrix operation
  - matrix operation  $\rightarrow$  3d tensor operation
- Typical batch sizes 50–100 sentence pairs

# Batches

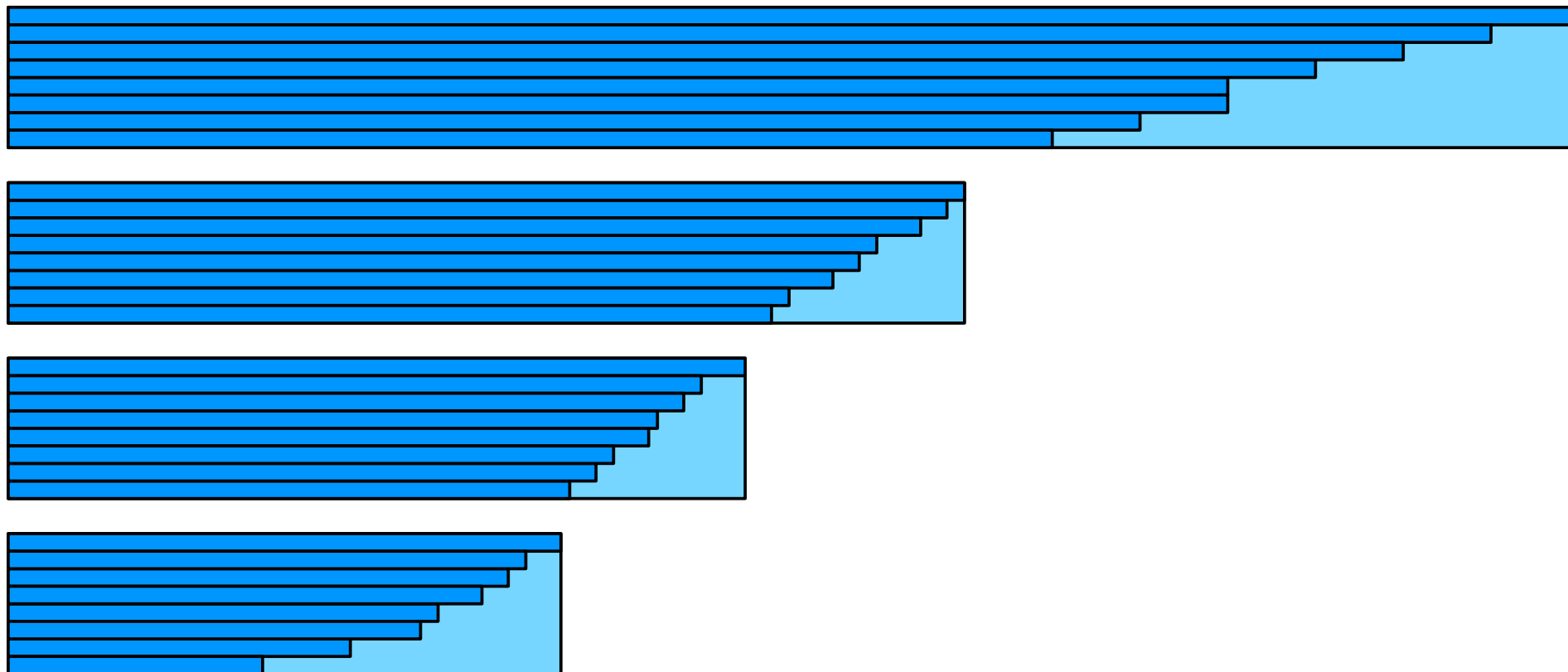
- Sentences have different length
- When batching, fill up unneeded cells in tensors



⇒ A lot of wasted computations

# Mini-Batches

- Sort sentences by length, break up into mini-batches



- Example: Maxi-batch 1600 sentence pairs, mini-batch 80 sentence pairs

# Overall Organization of Training

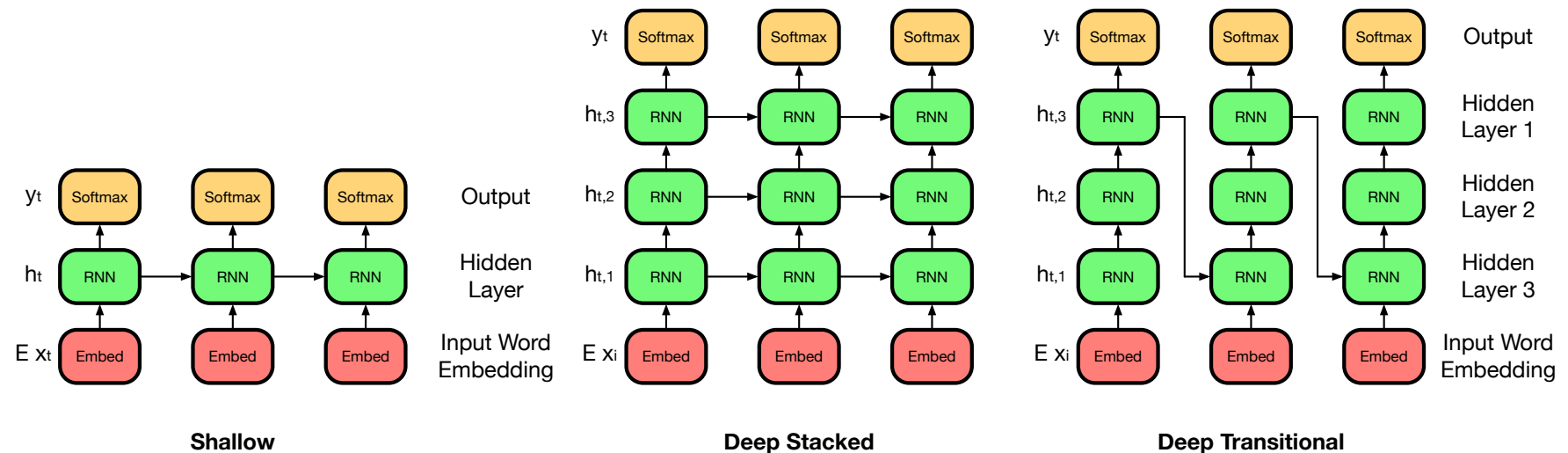
- Shuffle corpus
- Break into maxi-batches
- Break up each maxi-batch into mini-batches
- Process mini-batch, update parameters
- Once done, repeat
- Typically 5-15 epochs needed (passes through entire training corpus)



# deeper models

# Deeper Models

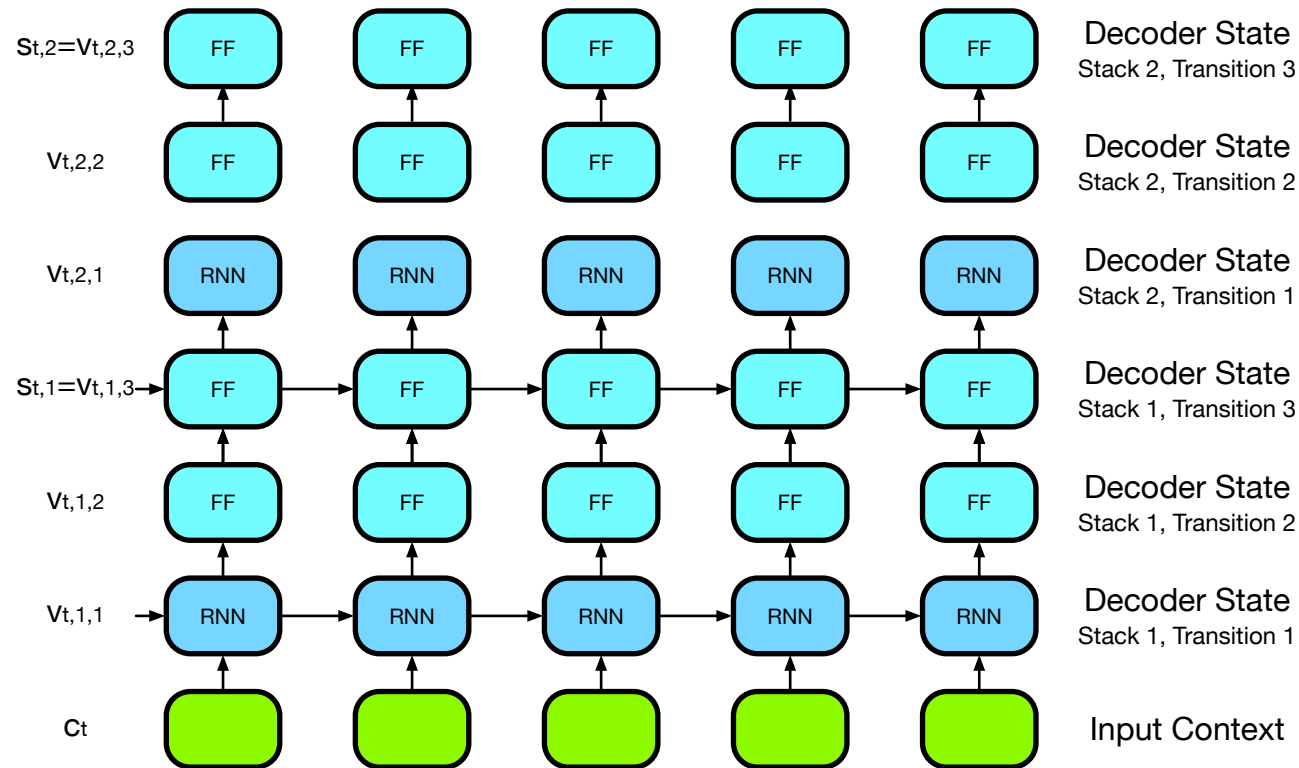
- Encoder and decoder are recurrent neural networks
- We can add additional layers for each step
- Recall shallow and deep language models



- Adding residual connections (short-cuts through deep layers) help

# Deep Decoder

- Two ways of adding layers
  - deep transitions: several layers on path to output
  - deeply stacking recurrent neural networks
- Why not both?



# Deep Encoder

- Previously proposed encoder already has 2 layers
  - left-to-right recurrent network, to encode left context
  - right-to-left recurrent network, to encode right context

⇒ Third way of adding layers

