# Sequence-to-Sequence Models

Kevin Duh
Johns Hopkins University
May 2019

# Outline

1. Problem Definition
2. Recurrent Model with Attention
3. Transformer Model

# Machine Learning Abstractions

# Machine Learning Abstractions

- Training data
  - Input: **x** / Output: **y**
  - Lots of $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ $\mathbf{i}=1,2,\ldots,N$

# Machine Learning Abstractions

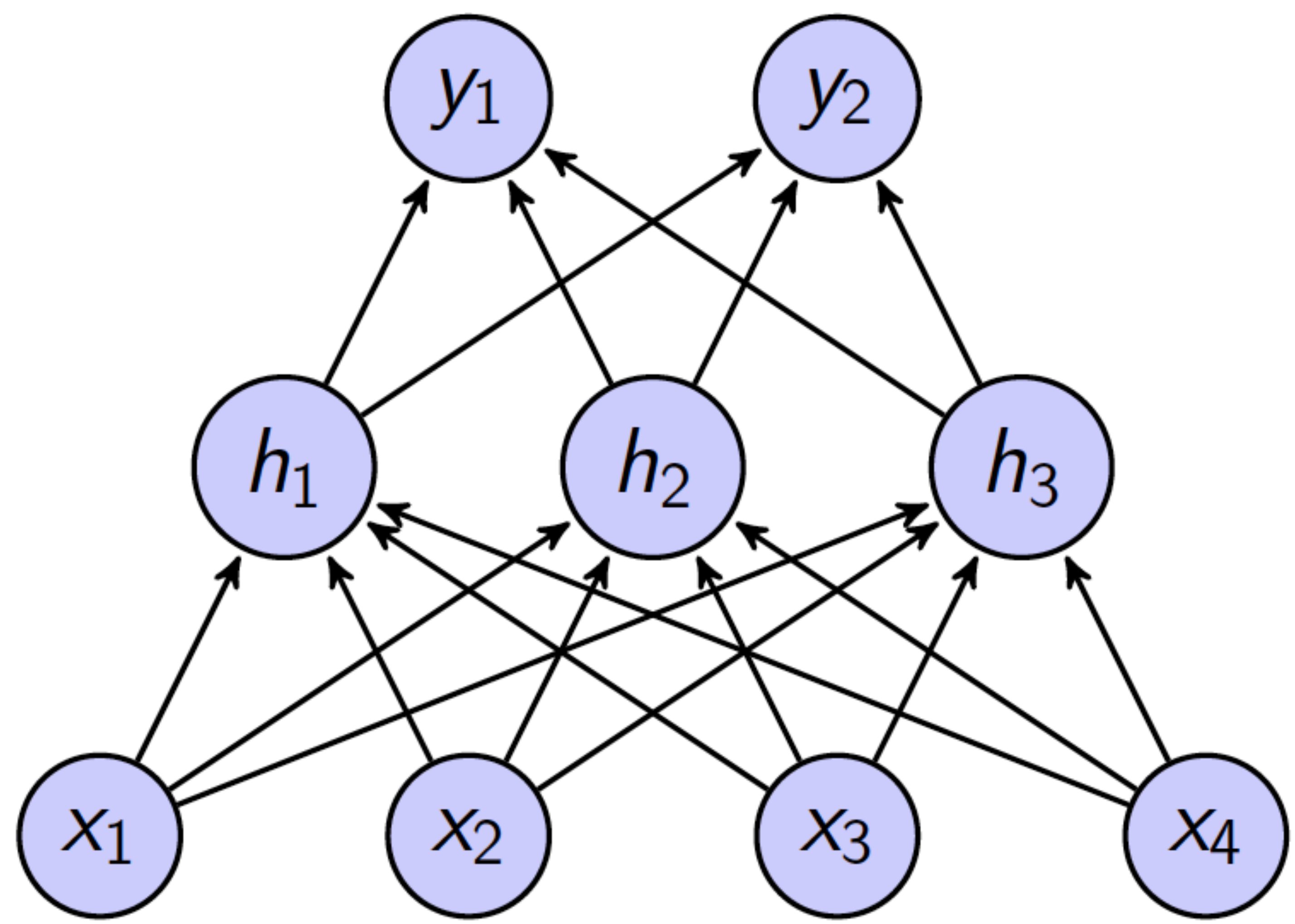
- Training data
  - Input: $\mathbf{x}$ / Output: $\mathbf{y}$
  - Lots of $\{(\mathbf{x}_i,\mathbf{y}_i)\}$ $\mathbf{i}=1,2,\ldots,N$
- Goal: Build model $F(\mathbf{x})$ on training data, generalize to test data: $\mathbf{y}_{\mathbf{prediction}} = F(\mathbf{x}_{\mathbf{test}})$ , $\mathbf{y}_{\mathbf{prediction}}$ vs $\mathbf{y}_{\mathbf{truth}}$

# Machine Learning Abstractions

- Training data
  - Input: $\mathbf{x}$ / Output: $\mathbf{y}$
  - Lots of $\{(\mathbf{x_i},\mathbf{y_i})\}$ $\mathbf{i}$=1,2,…,N
- Goal: Build model F($\mathbf{x}$) on training data, generalize to test data: $\mathbf{y_{prediction}} = F(\mathbf{x_{test}})$ , $\mathbf{y_{prediction}}$ vs $\mathbf{y_{truth}}$
- What is the structure of $\mathbf{x}$ and $\mathbf{y}$?

# Standard classification problem

- **x** is a vector in $R^D$
- **y** is a label from {class1, class2, class3, … classK}
- A neural net for F(**x**):
  - **x**=[$x_1$; $x_2$; $x_3$; $x_4$]
  - **h**=nonlinear(W***x**)
  - **y**=softmax(M***h**)



# Image classification example

Image feature:
**x** = 960x720 256 RGB vector

**y** = {dog, cat, squirrel, alligator, dinosaur}

From: https://commons.wikimedia.org/wiki/File:This_is_a_very_cute_dog.jpg

# More complex problems

# More complex problems

- Complex Input:
  - **x** is a sequence of L vectors/words: $R^{DxL}$
  - **y** is a label from {class1, class2, class3, … classK}
  - Example: mention span to NE type classification

# More complex problems

- Complex Input:
  - **x** is a sequence of L vectors/words: $R^{DxL}$
  - **y** is a label from {class1, class2, class3, … classK}
  - Example: mention span to NE type classification
- Complex Input and Output:
  - **x** is a sequence of L vectors/words
  - **y** is a sequence of J vectors/words

# Sequence Output Example: Image Captioning

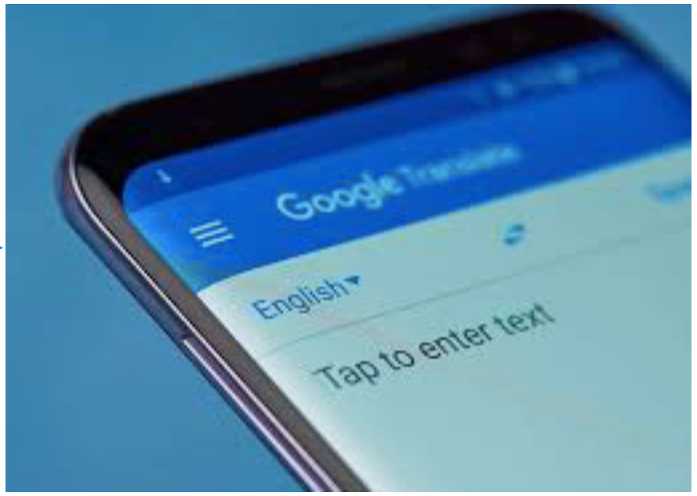
Image feature:
**x** = 960x720 256 RGB vector

**Caption text generation output space:**
**{ all possible English sentences }**

**a cute dog**
**a very cute dog**
**super cute puppy**
**adorable puppy looking at me**
**....**

# Sequence-to-Sequence Example: Machine Translation

**das Haus ist gross**




**the house is big**

# Sequence-to-Sequence Example: Named Entity Recognition

# Handling sequences

# Handling sequences

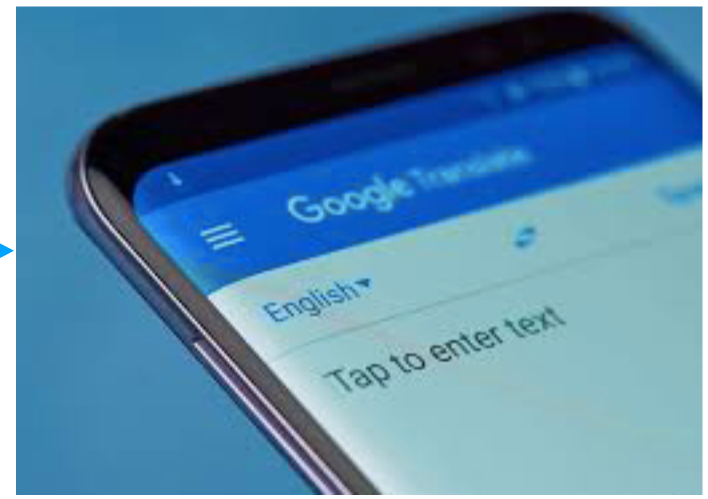
- For sequence input:
  - We need an “encoder” to convert arbitrary length input to some fixed-length hidden representation
  - Without this, may be hard to apply matrix operations

# Handling sequences

- For sequence input:
  - We need an “encoder” to convert arbitrary length input to some fixed-length hidden representation
  - Without this, may be hard to apply matrix operations
- For sequence output:
  - We need a “decoder” to generate arbitrary length output
  - One method: generate one word at a time, until special <stop> token

# Example:
# Machine Translation

**das Haus ist gross**

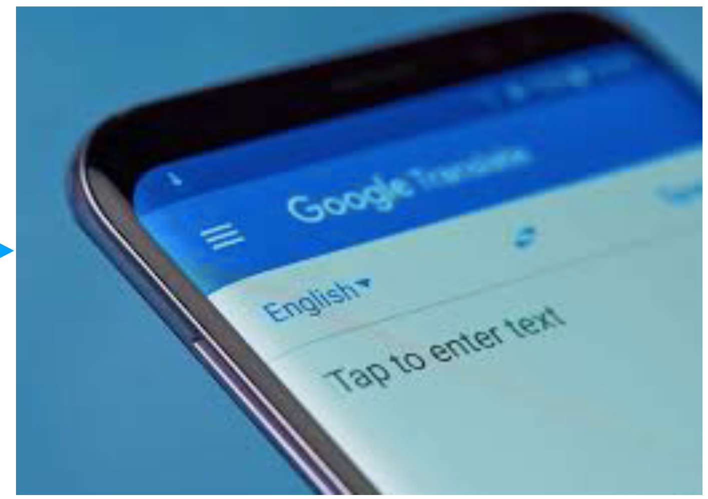



**the house is big**

**das Haus ist gross**

# Example:
# Machine Translation

das Haus ist gross




the house is big

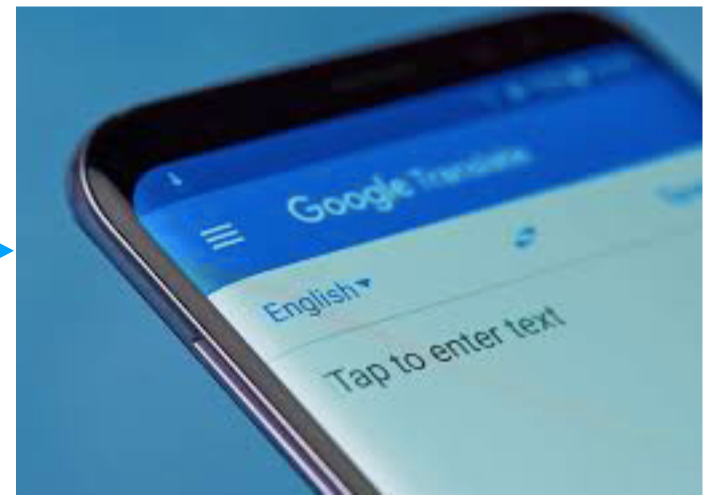
"Sentence Vector"

Encoder

das Haus ist gross

# Example: Machine Translation

das Haus ist gross




the house is big

“Sentence Vector”

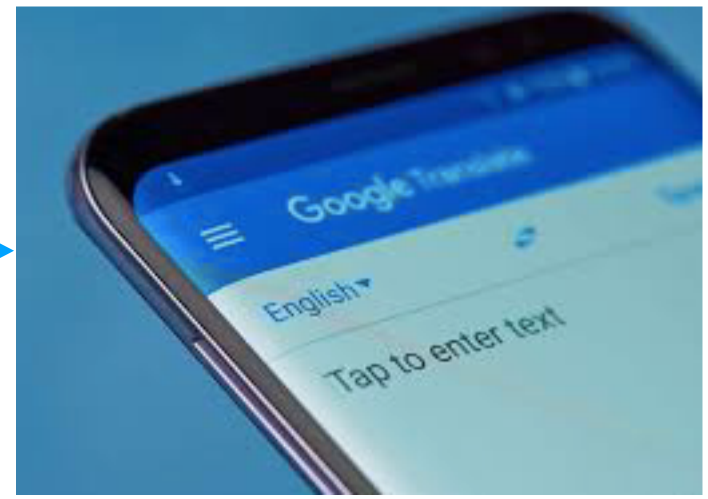
Encoder

Decoder

das Haus ist gross

# Example: Machine Translation

das Haus ist gross




the house is big

step 1: the

“Sentence Vector”

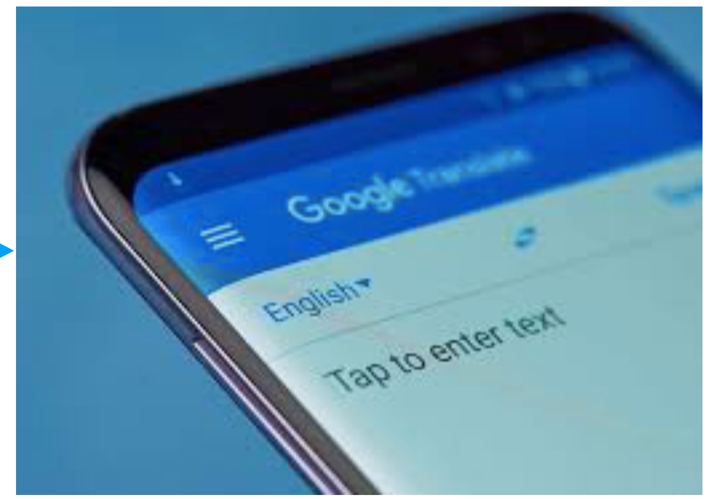
Encoder

Decoder

das Haus ist gross

# Example: Machine Translation

das Haus ist gross




the house is big

"Sentence Vector"

Encoder

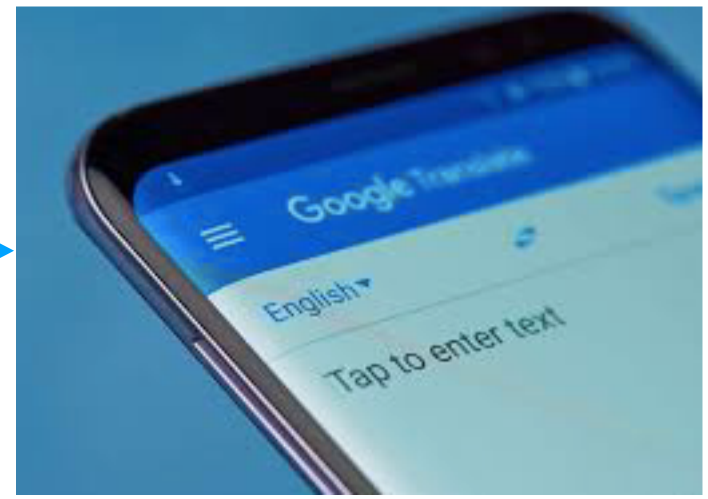
Decoder

das Haus ist gross

step 1: the
step 2: house

# Example:
# Machine Translation

das Haus ist gross




the house is big

das Haus ist gross

Encoder

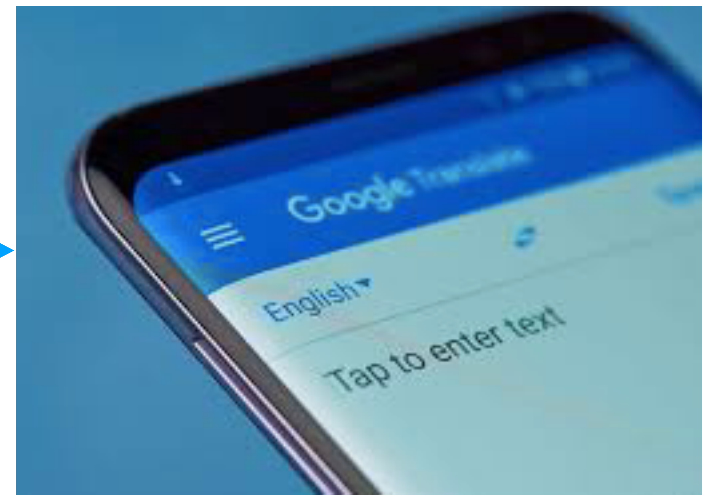
“Sentence Vector”

Decoder

step 1: the
step 2: house
step 3: is

# Example:
# Machine Translation

das Haus ist gross




the house is big

“Sentence Vector”

Encoder

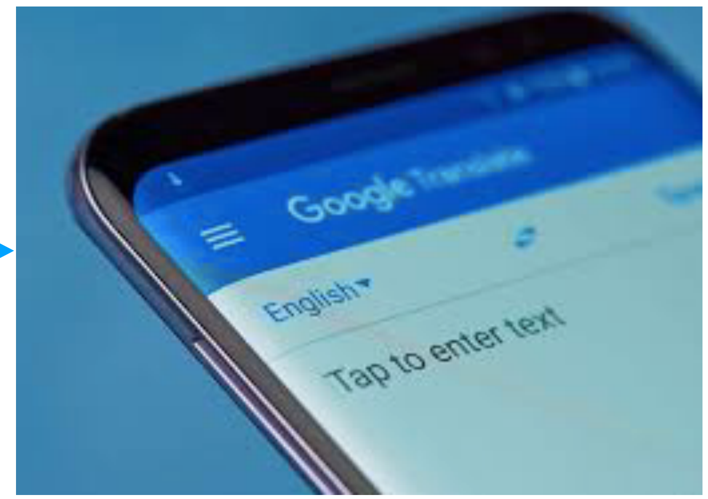
das Haus ist gross

Decoder

step 1: the
step 2: house
step 3: is
step 4: big

# Example:
# Machine Translation

das Haus ist gross




the house is big

“Sentence Vector”

Encoder

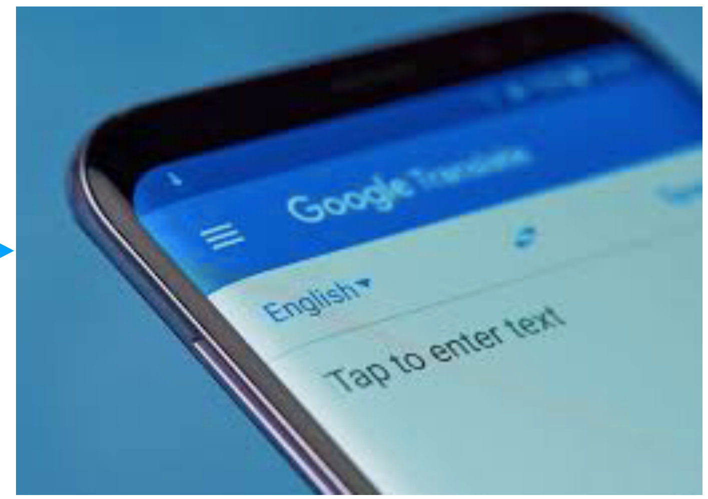
das Haus ist gross

Decoder

step 1: the
step 2: house
step 3: is
step 4: big
step 5: <stop>

# Example:
# Machine Translation

das Haus ist gross




the house is big

“Sentence Vector”

Encoder

Decoder

das Haus ist gross

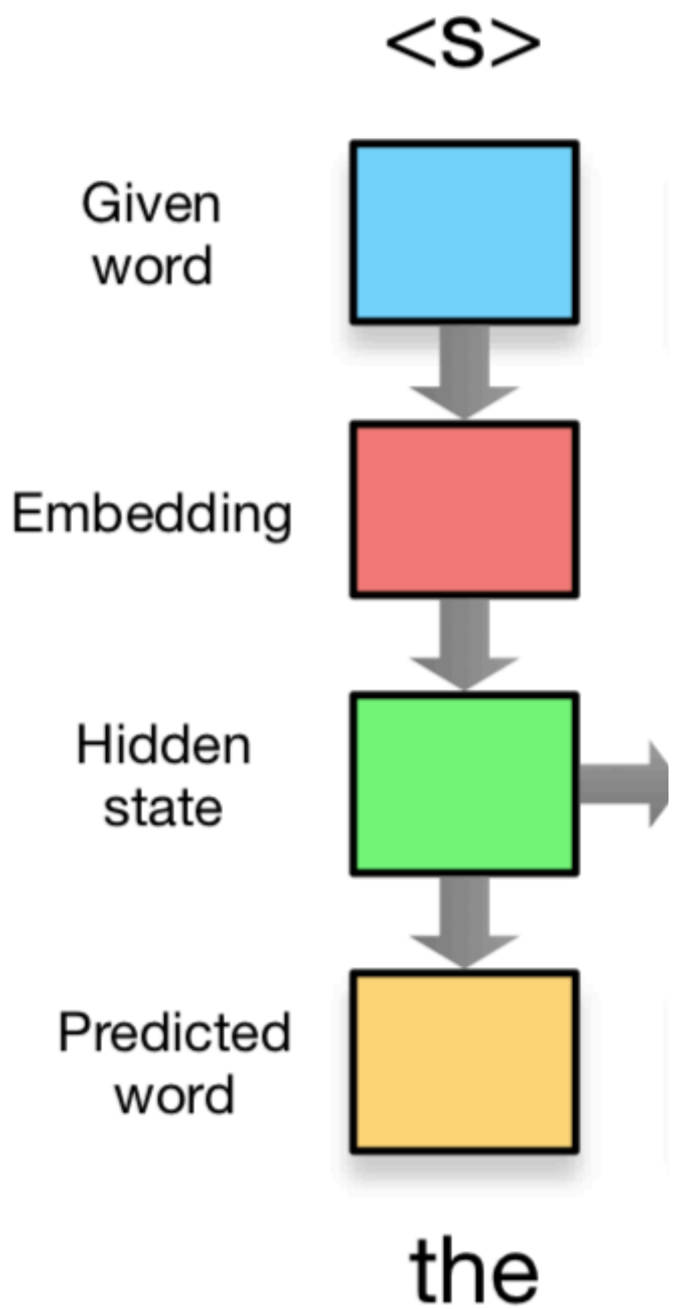
step 1: the
step 2: house
step 3: is
step 4: big
step 5: <stop>

Each step applies a
softmax over all vocab

# Outline

1. Problem Definition
2. Recurrent Model with Attention
3. Transformer Model

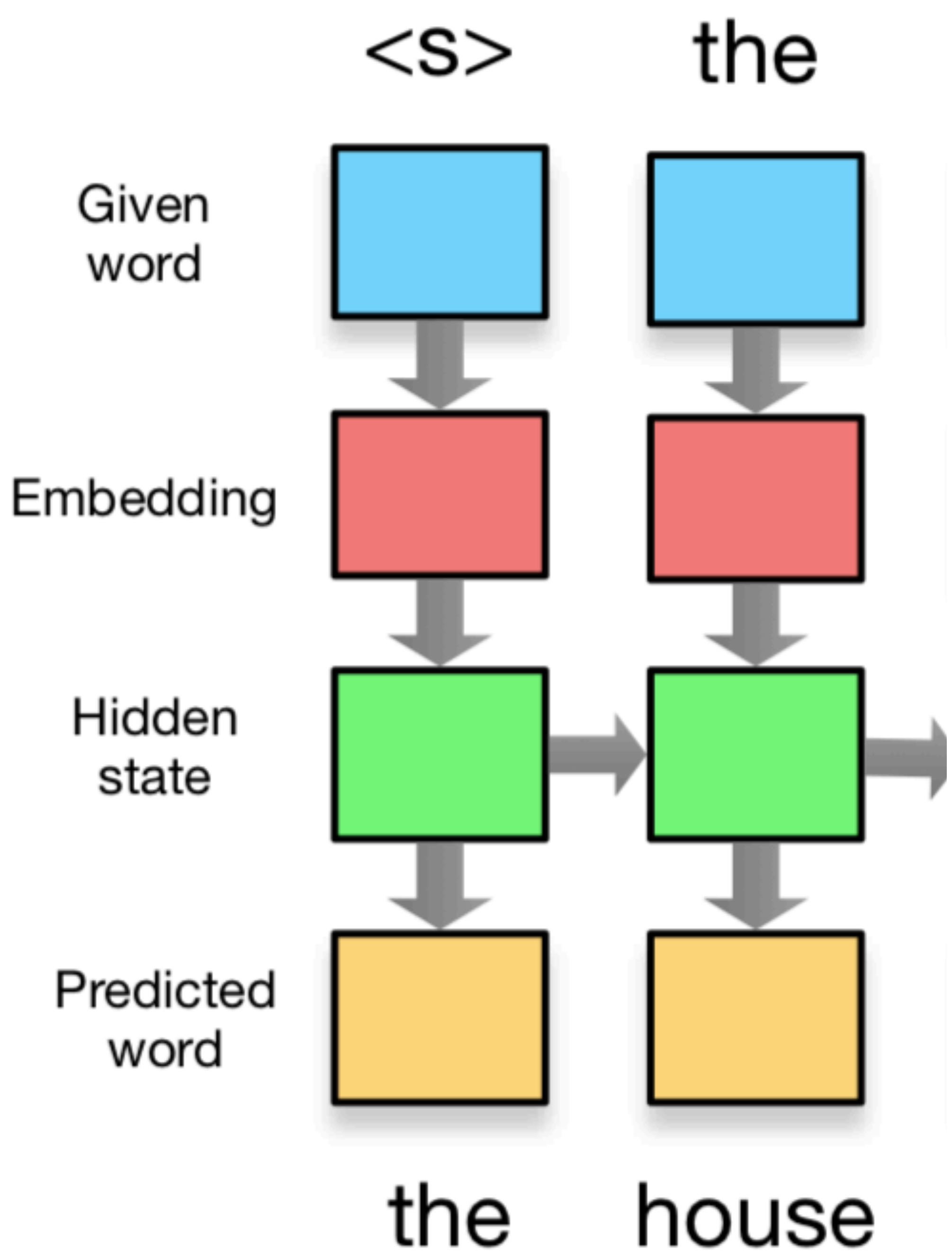
# Sequence modeling with a recurrent network




the house is big .

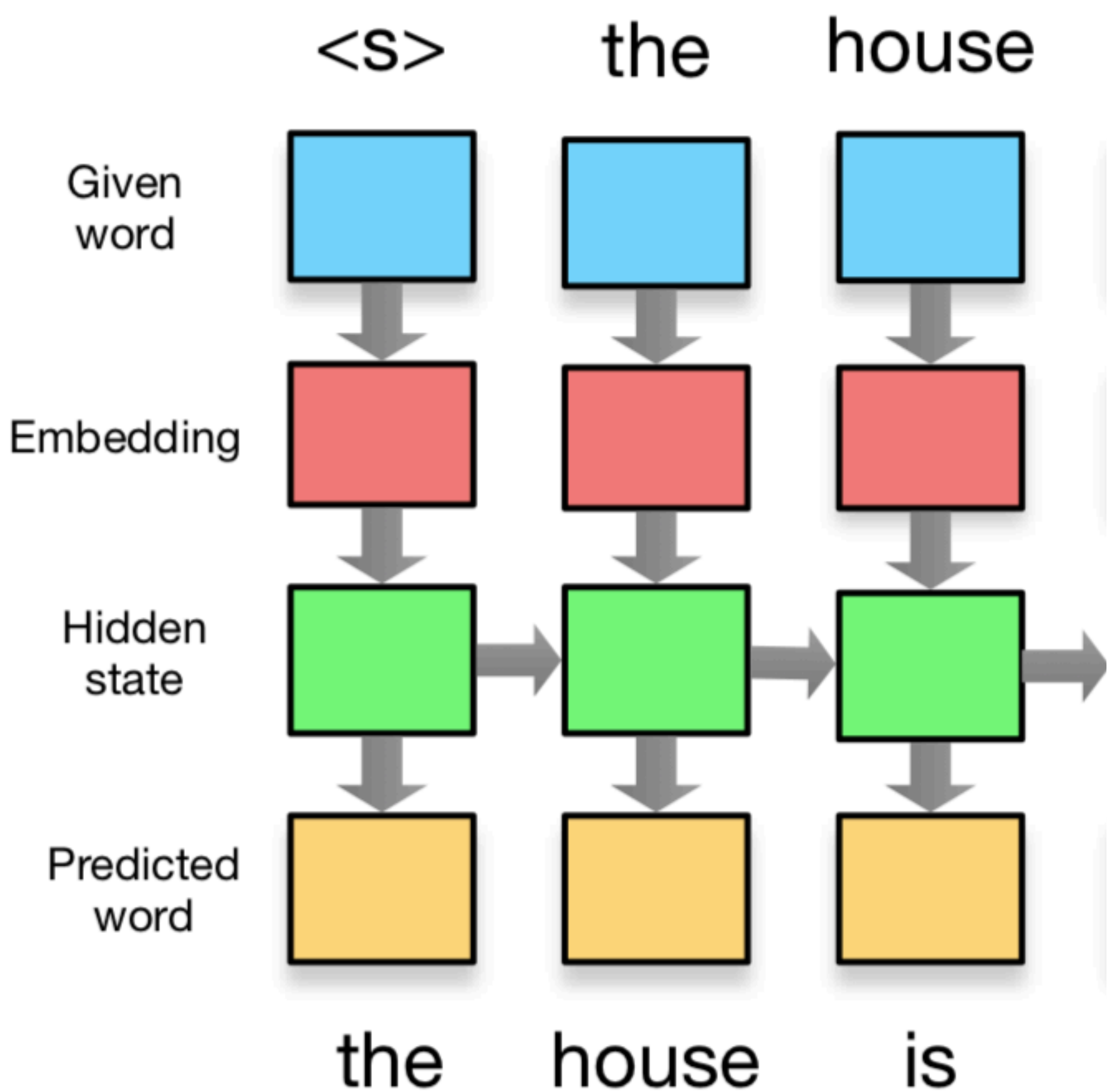
**The following animations courtesy of Philipp Koehn: http://mt-class.org/jhu**

# Sequence modeling with a recurrent network



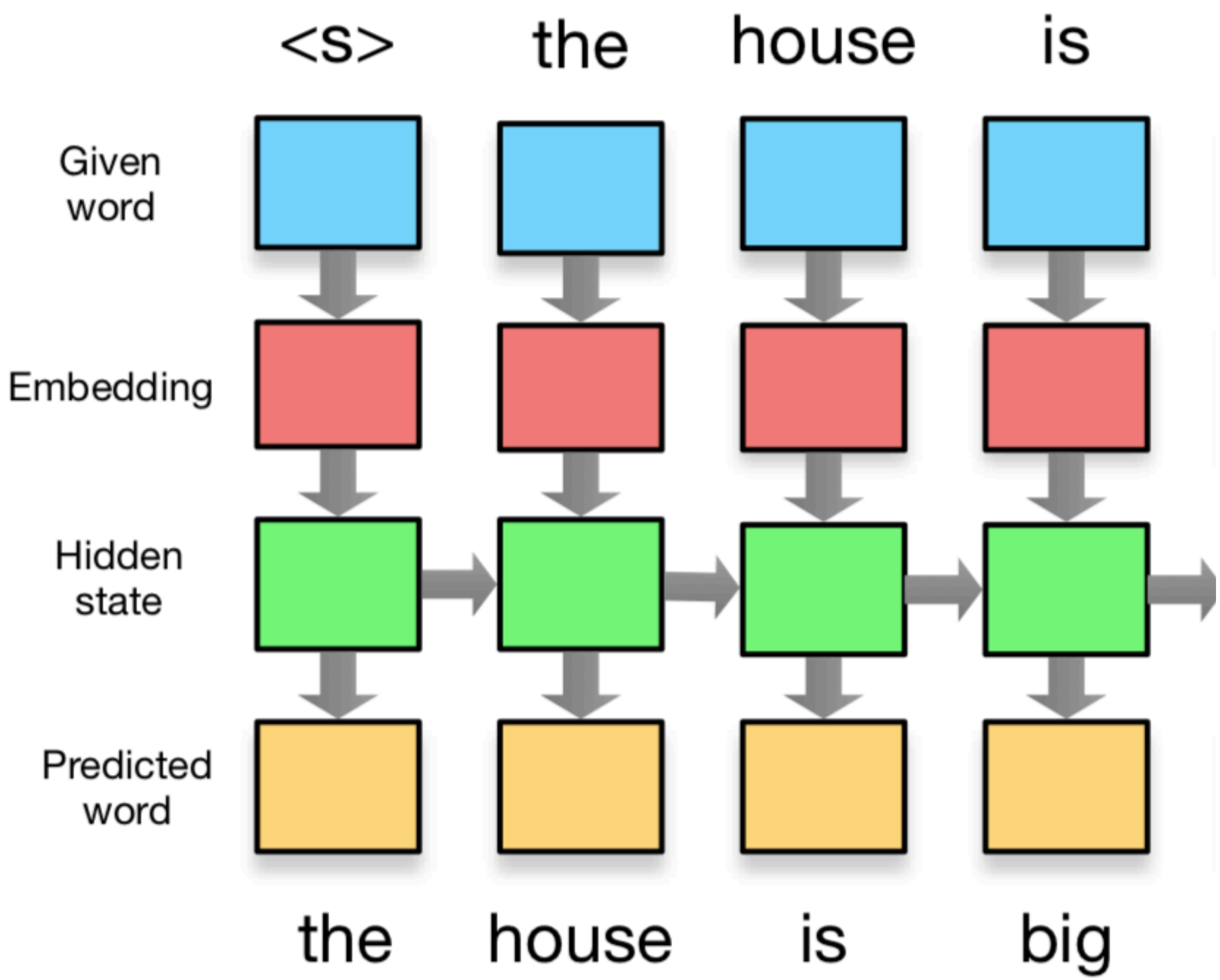

**the house is big .**

# Sequence modeling with a recurrent network



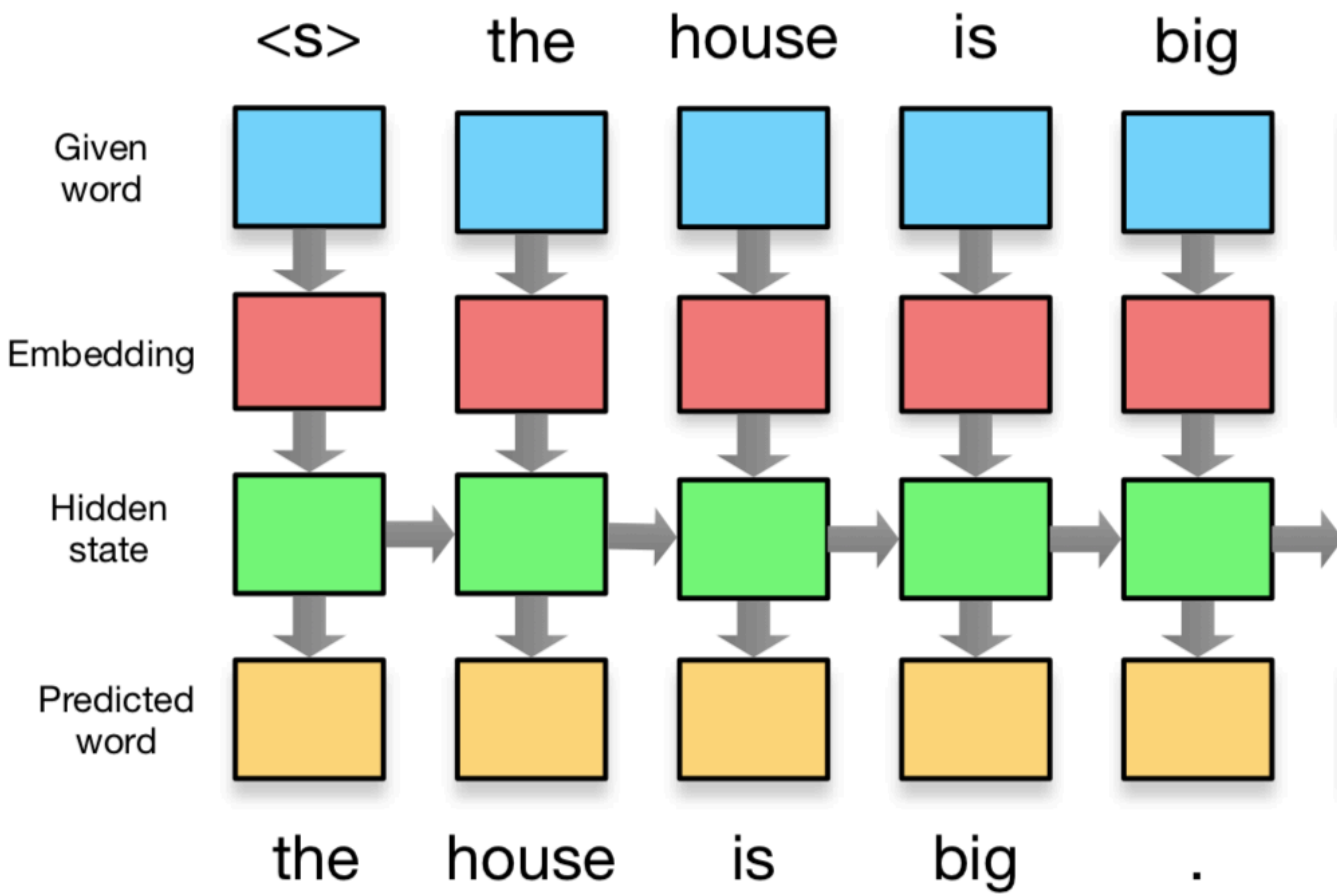

**the house is big .**

# Sequence modeling with a recurrent network



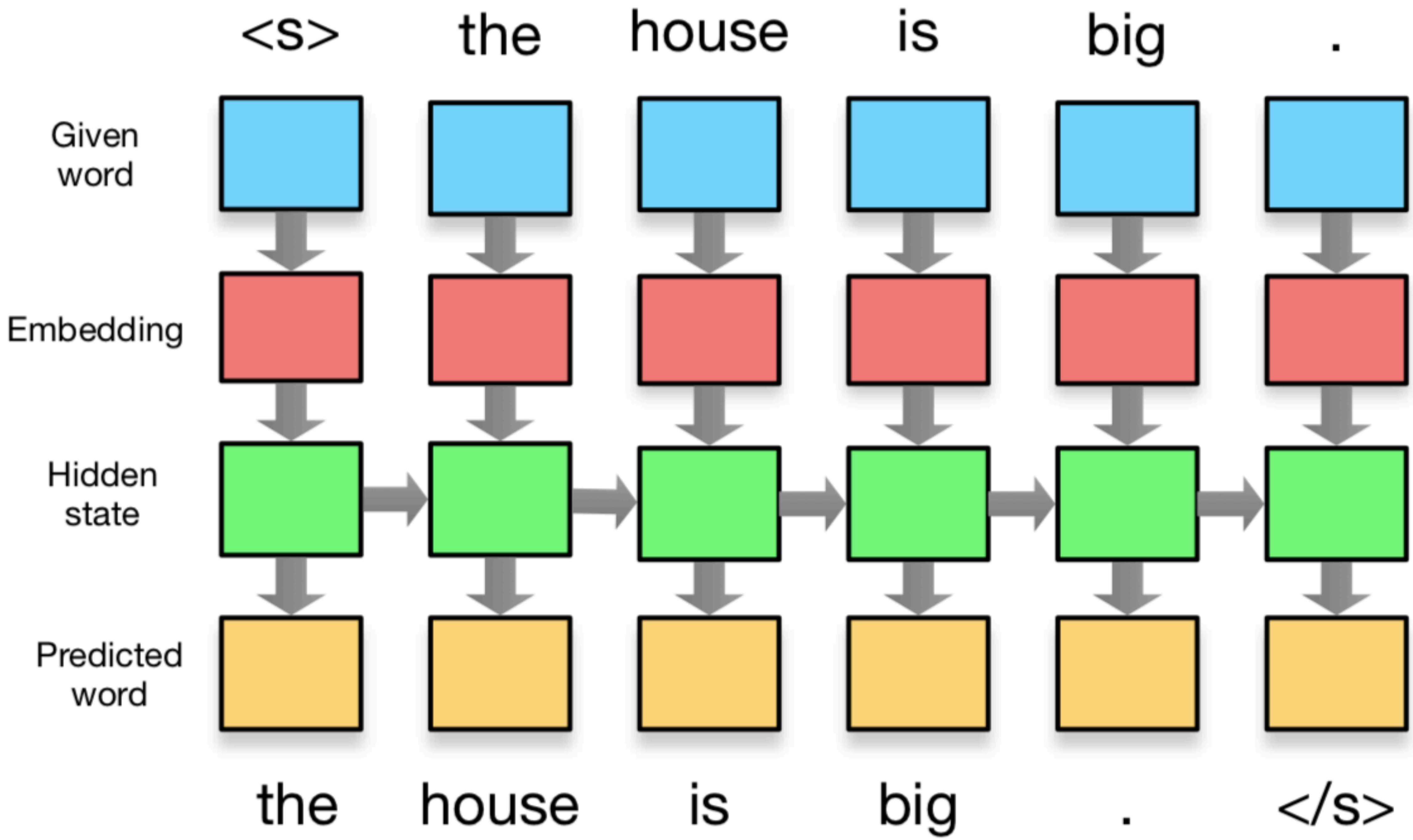

**the house is big .**

# Sequence modeling with a recurrent network




**the house is big .**

# Sequence modeling with a recurrent network

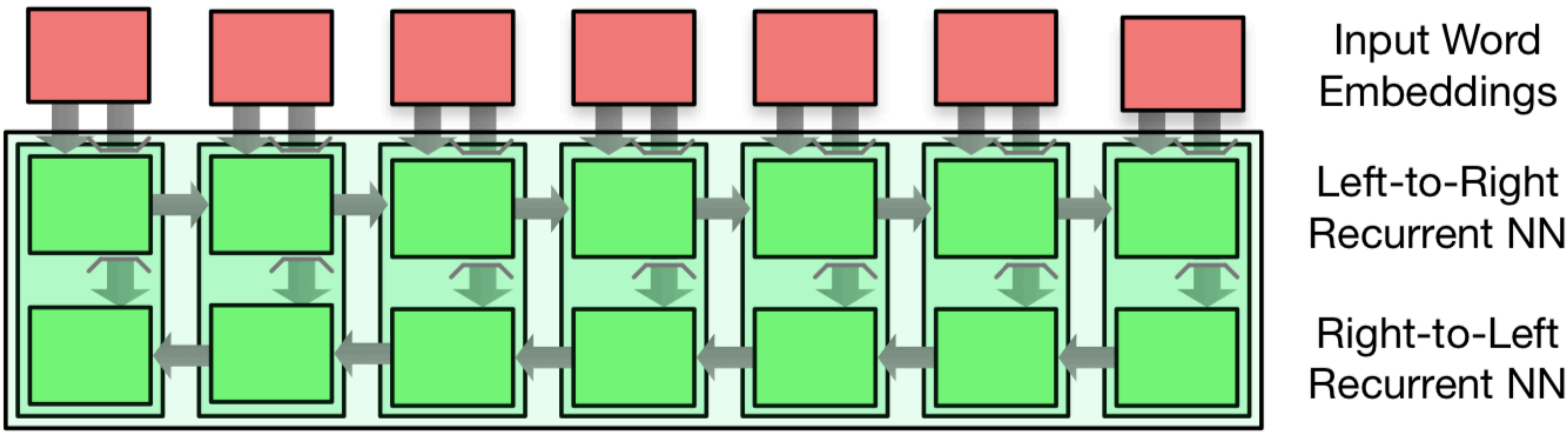



**the house is big .**

# Recurrent models for sequence-to-sequence problems

- We can use these models for both input and output
- For output, there is the constraint of left-to-right generation
- For input, we are provided the whole sentence at once, we can do both left-to-right and right-to-left modeling
- The recurrent units may be based on LSTM, GRU, etc.

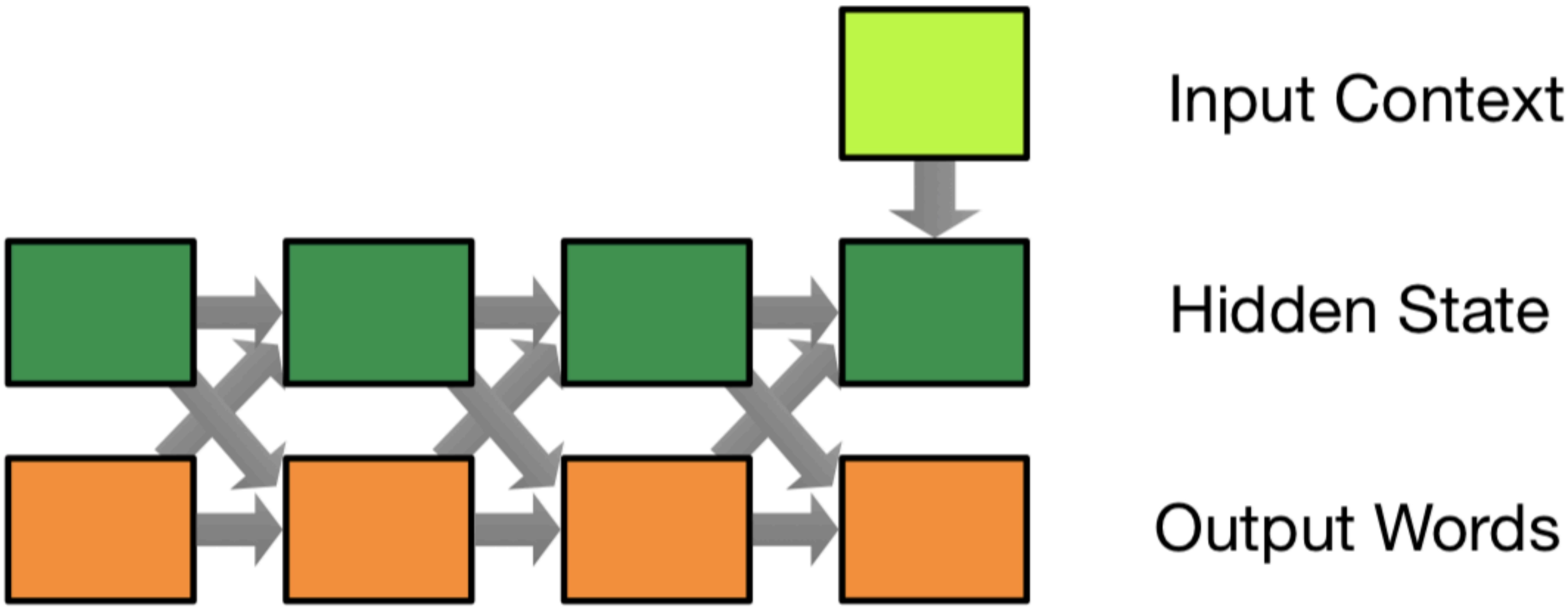
# Bidirectional Encoder for Input Sequence




**Word embedding: word meaning in isolation**
**Hidden state of each Recurrent Neural Net (RNN): word meaning in this sentence**

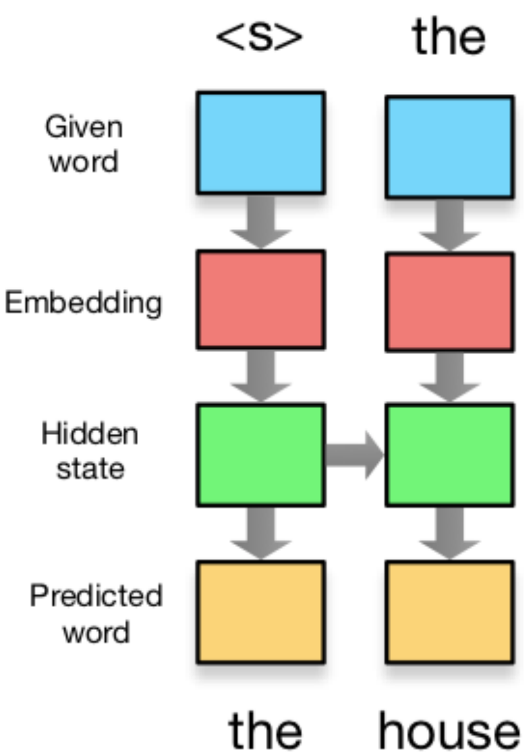
$$\overleftarrow{h_j} = f(\overleftarrow{h_{j+1}}, \bar{E}\, x_j)$$
$$\overrightarrow{h_j} = f(\overrightarrow{h_{j-1}}, \bar{E}\, x_j)$$

# Left-to-Right Decoder

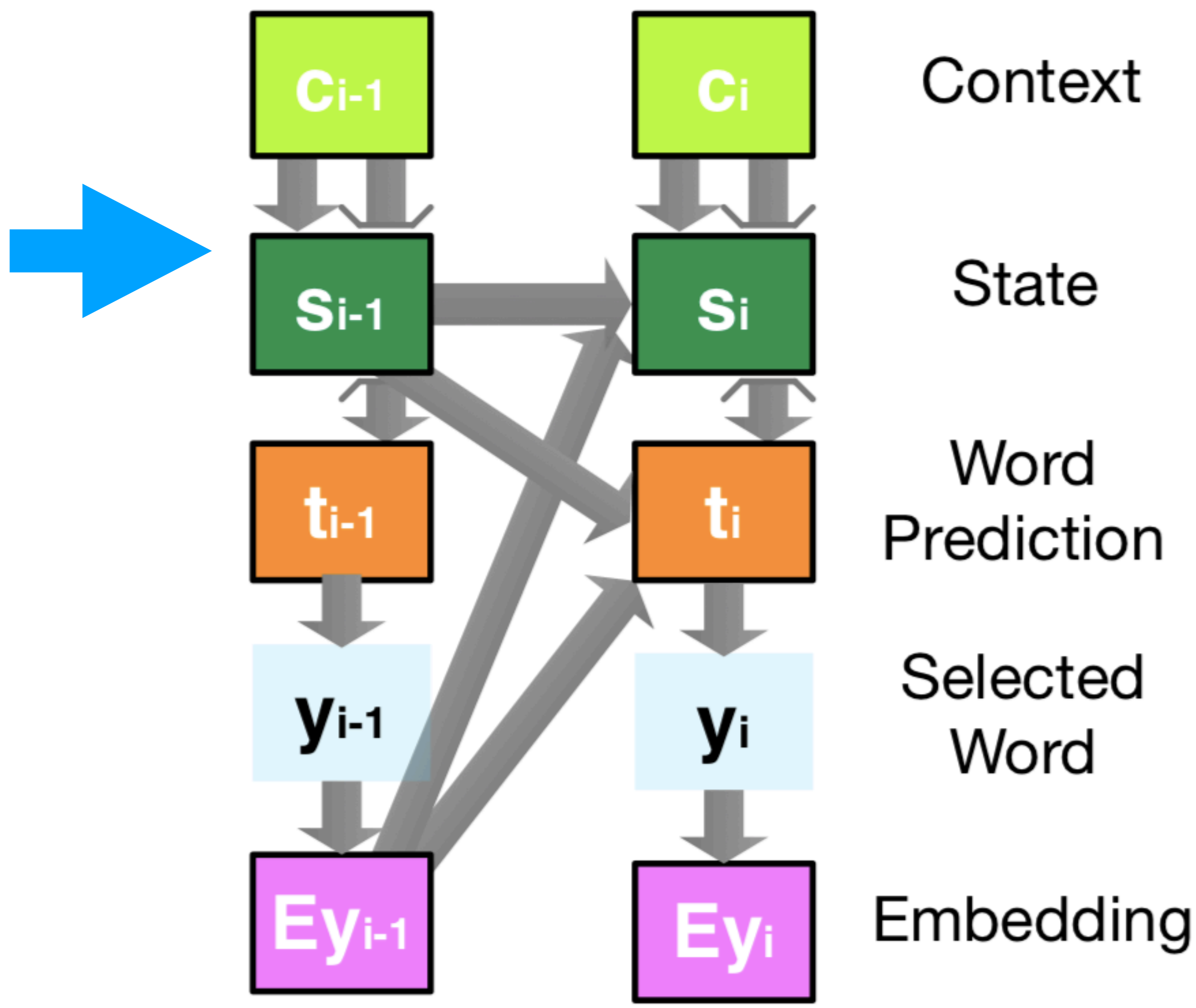



- Input context comes from encoder
- Each output is informed by current hidden state and previous output word
- Hidden state is updated at every step

# In detail: each step




(simplified view)




**Context contains information from encoder/input**

$$s_i = f(s_{i-1},\ Ey_{-1}, c_i)$$

$$t_i = W(Us_{i-1} + VEy_{i-1} + Cc_i)$$

# What connects the encoder and decoder

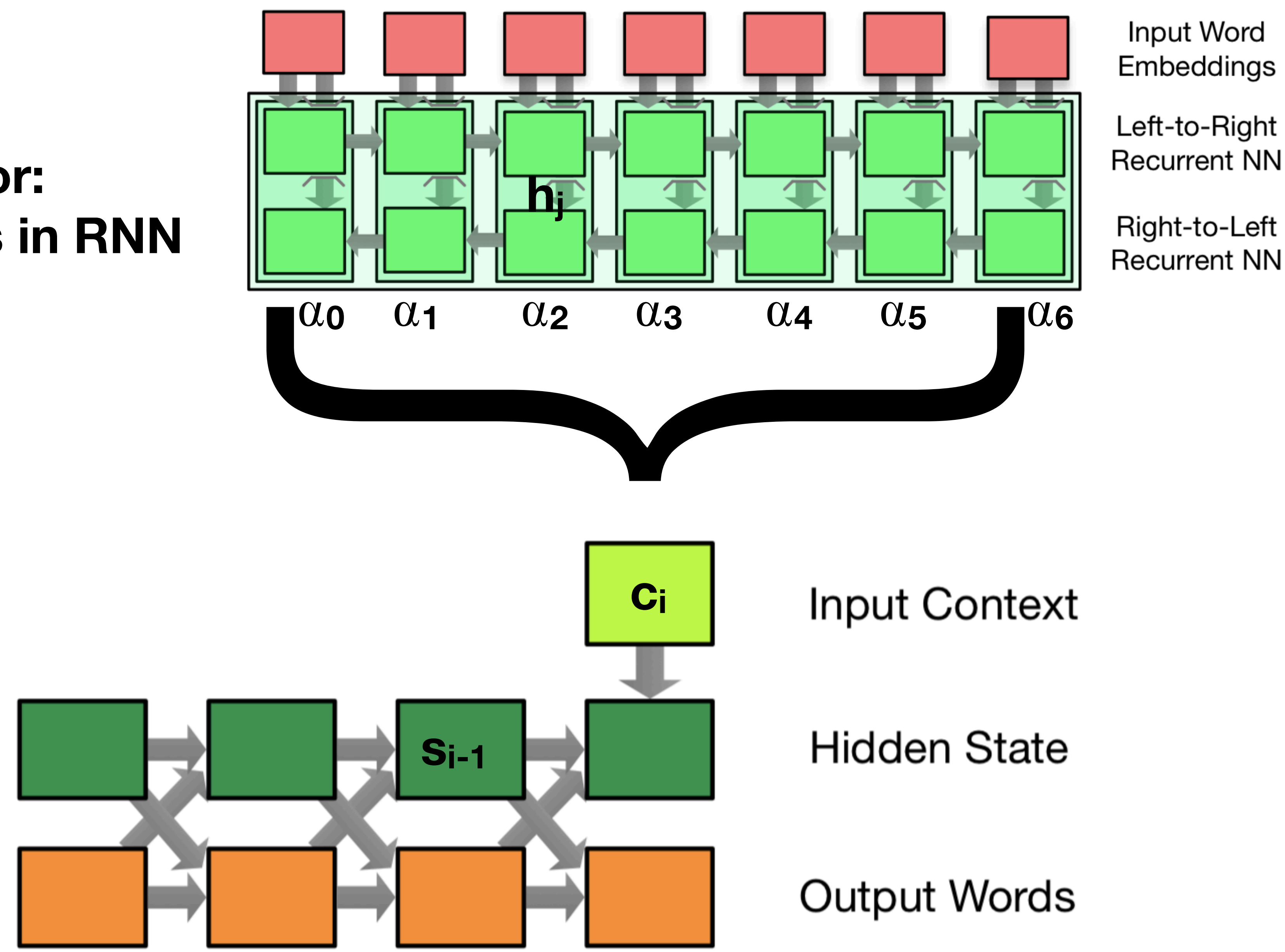
**Input context is a fixed-dim vector: weighted average of all L vectors in RNN**

**How to compute weighting? Attention mechanism:**

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

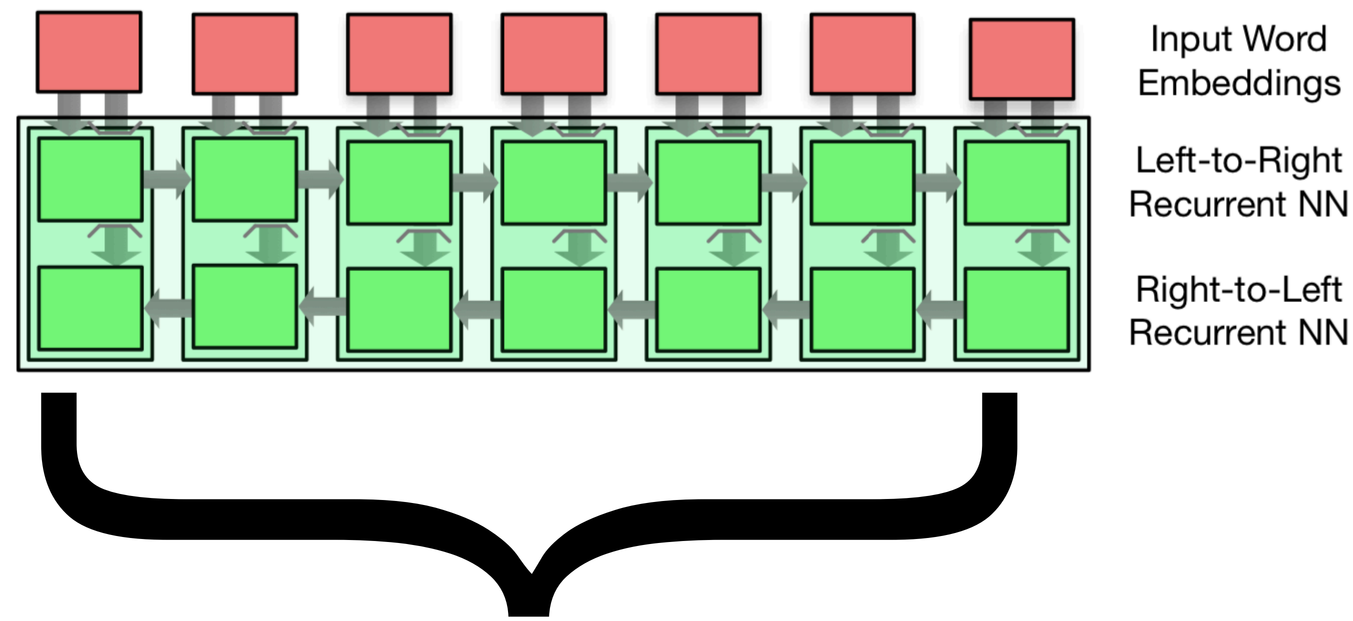
$$c_i = \sum_j \alpha_{ij} h_j$$

**Note this changes at each step i**
**What's paid attention has more influence on next prediction**

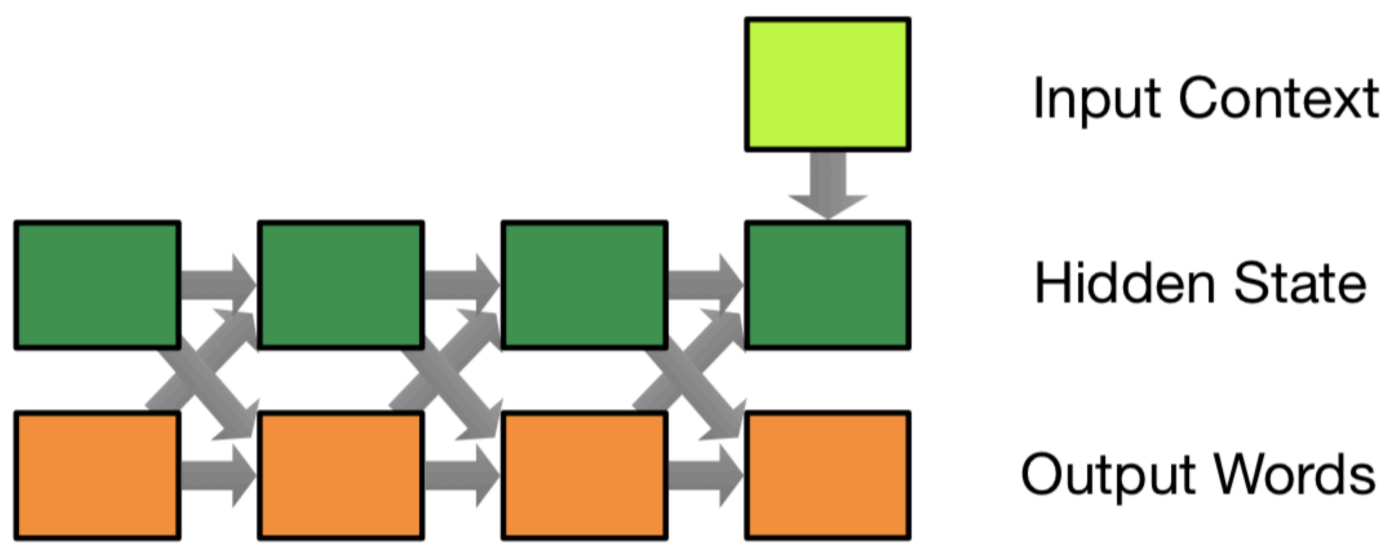


# To wrap up: Recurrent models with attention

**1. Encoder takes in arbitrary length input**




**2. Decoder generates output one word at a time, using current hidden state, input context (from attention), and previous output**




**Note: we can add layers to make this model "deeper"**

# Outline

1. Problem Definition
2. Recurrent Model with Attention
3. Transformer Model

# Motivation of Transformer Model

# Motivation of Transformer Model

- RNNs are great, but have two demerits:

# Motivation of Transformer Model

- RNNs are great, but have two demerits:
  - Sequential structure is hard to parallelize, may slow down GPU computation

# Motivation of Transformer Model

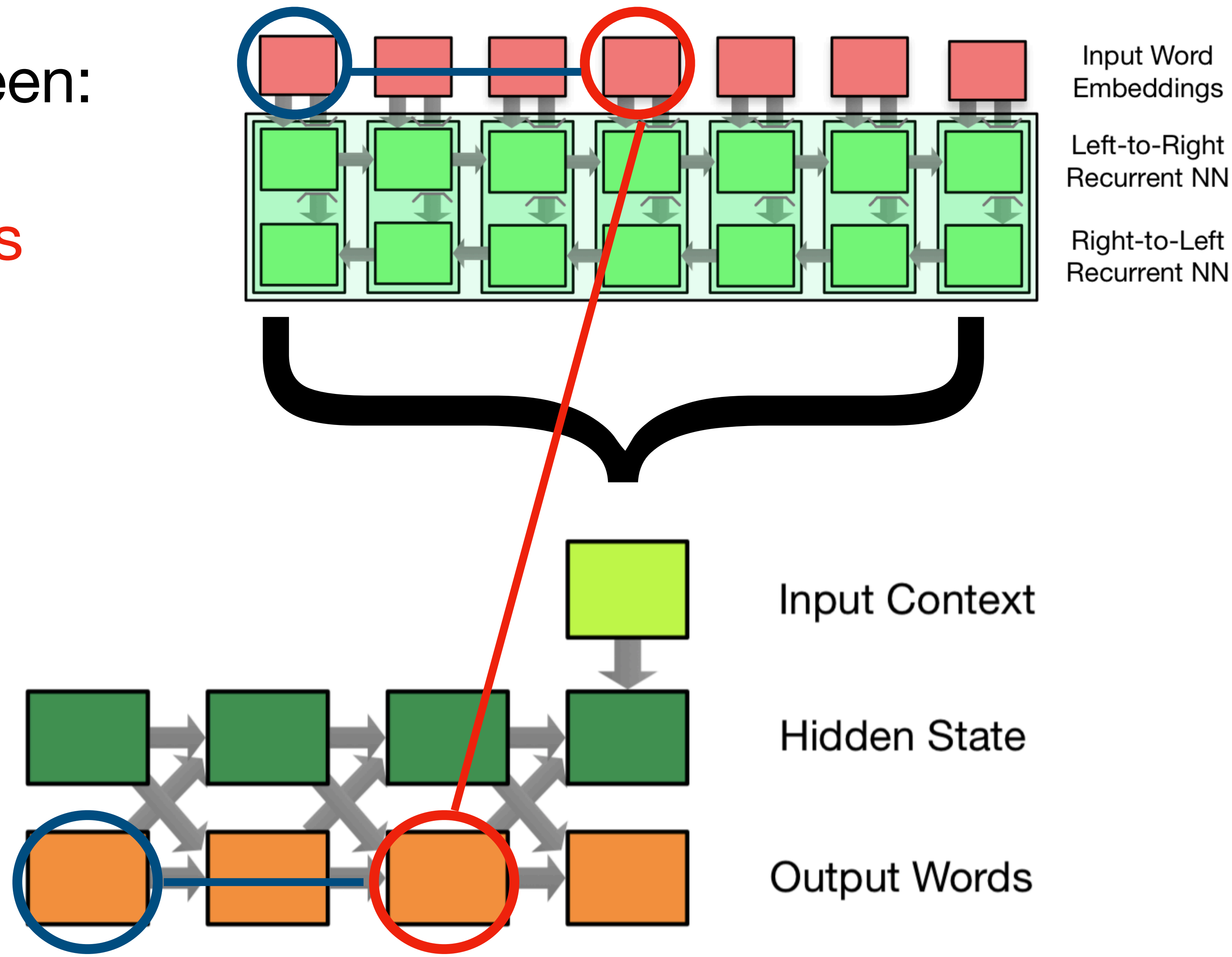
- RNNs are great, but have two demerits:
  - Sequential structure is hard to parallelize, may slow down GPU computation
  - Still has to model some kinds of long-term dependency (though addressed by LSTM/GRU)

# Motivation of Transformer Model

- RNNs are great, but have two demerits:
  - Sequential structure is hard to parallelize, may slow down GPU computation
  - Still has to model some kinds of long-term dependency (though addressed by LSTM/GRU)
- Transformers solve the sequence-to-sequence problem using only attention mechanisms, no RNN

# Long-term dependency

- Dependencies between:
  - Input-output words
  - Two input words
  - Two output words



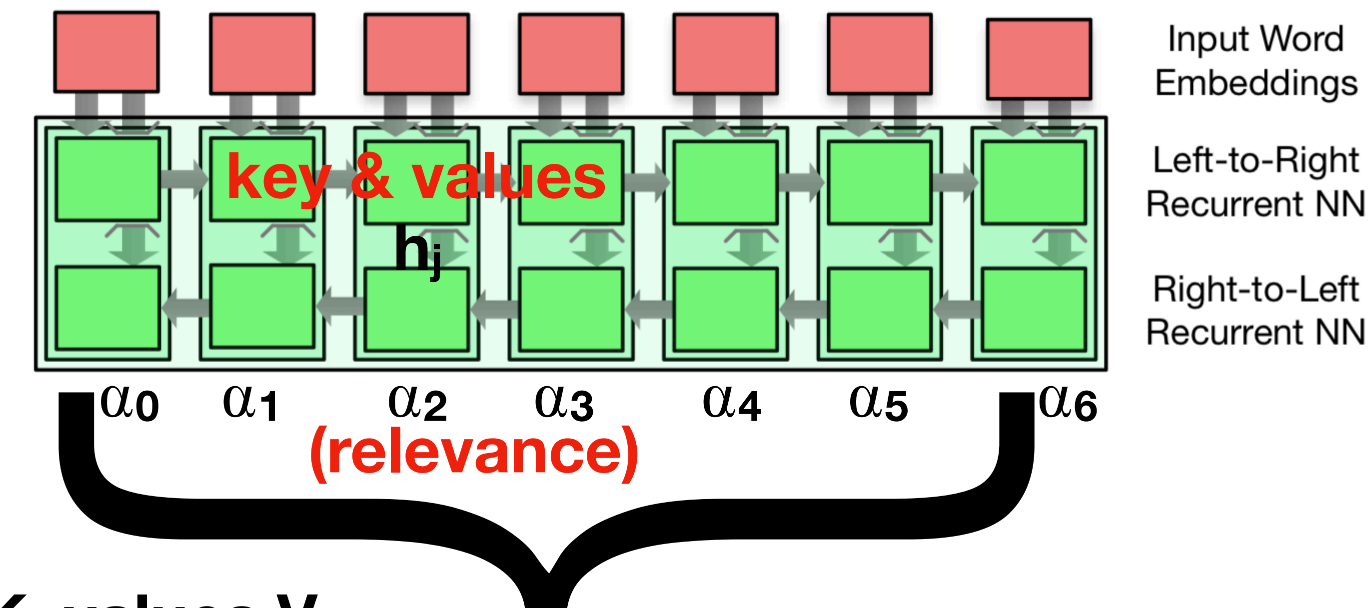

**Attention mechanism "shortens" path between input and output words.**
**What about others?**

# Attention, more abstractly

**Previous attention formulation:**

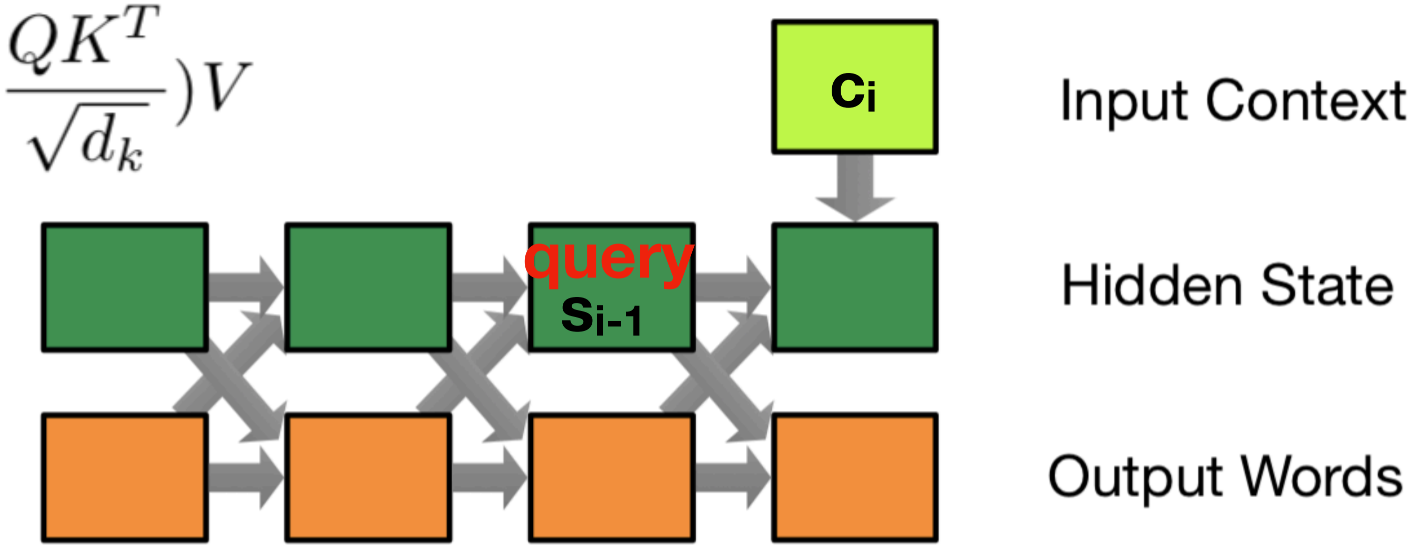
$$\alpha_{ij} = \frac{\exp(a(s_{i-1}, h_j))}{\sum_k \exp(a(s_{i-1}, h_k))}$$

$$c_i = \sum_j \alpha_{ij} h_j$$




**Abstract formulation:**
**Scaled dot-product for queries Q, keys K, values V**

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

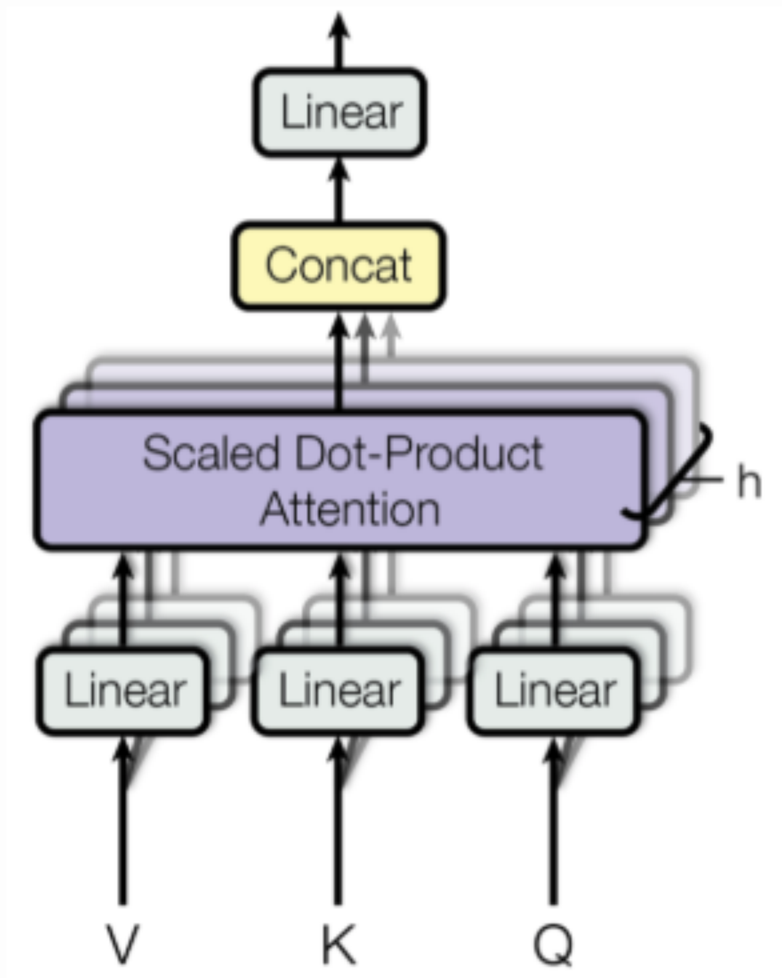


# Multi-head Attention

- For expressiveness, do at scaled dot-product attention multiple times
- Add different linear transform for each key, query, value

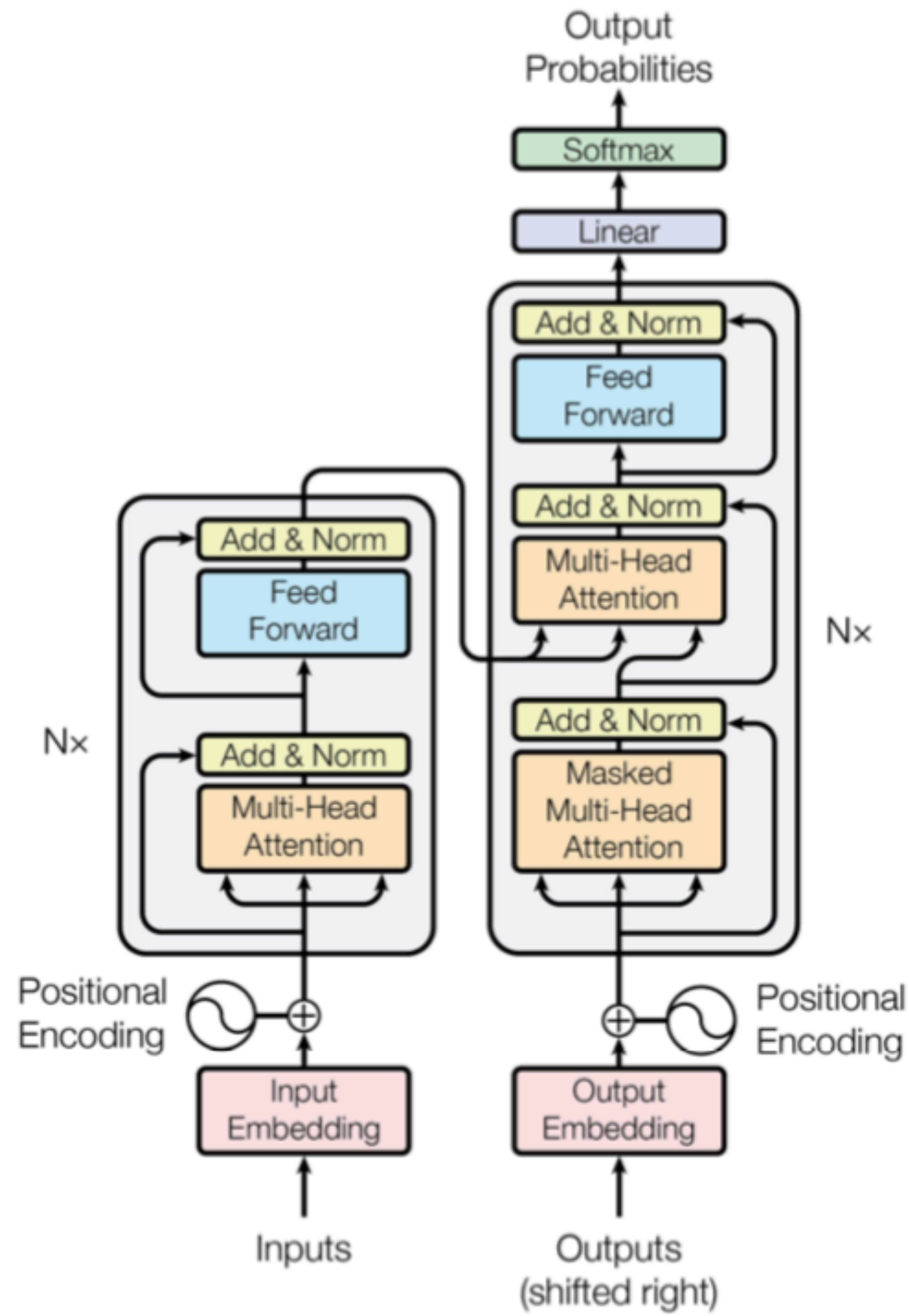
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v} \quad W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$



# Putting it together

- Multiple (N) layers
- For encoder-decoder attention, Q: previous decoder layer, K and V: output of encoder
- For encoder self-attention, Q/K/V all come from previous encoder layer
- For decoder self-attention, allow each position to attend to all positions up to that position
- Positional encoding for word order



**From: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html**

From: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

# Summary

1. Problem Definition:

   - Sequence-to-sequence problems are more complex, but can be solved by (a) encoding input to fixed representations and (b) decoding output one at a time

2. Recurrent Model with Attention

   - Bidirectional RNN encoder, RNN decoder, attention-based context vector tying it together

3. Transformer Model

   - Another way to solve sequence problems, without using sequential models

# Research directions

- Lots!!