# Attention in NLP

CS 6956: Deep Learning for NLP

THE UNIVERSITY OF UTAH

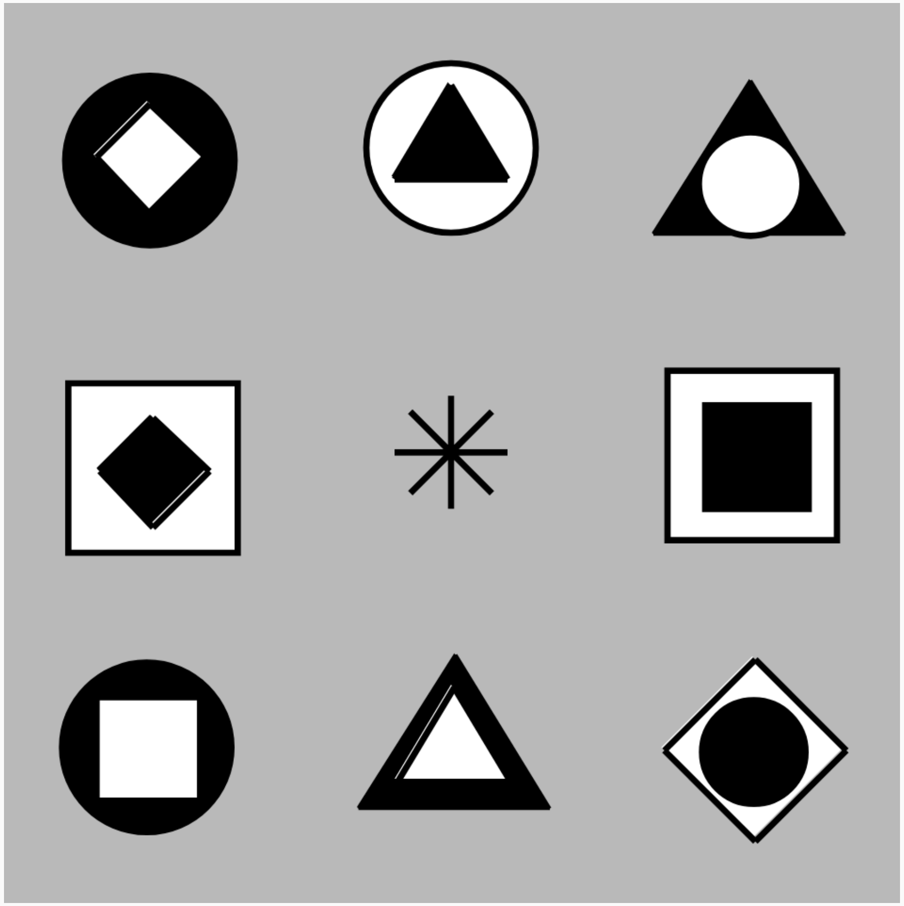
# Overview

- What is attention
- Attention in encoder-decoder networks
- Various kinds of attention

# Overview

- What is attention?
- Attention in encoder-decoder networks

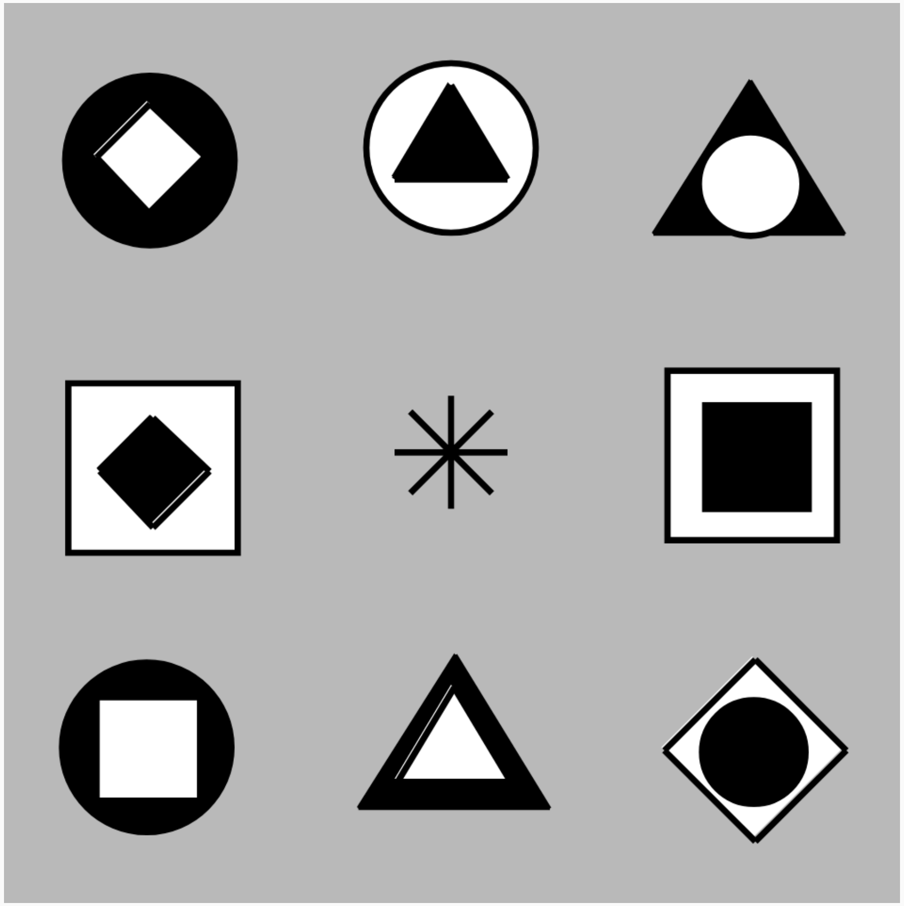
# Visual attention



Keep your eyes fixed on the star at the center of the image

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

# Visual attention

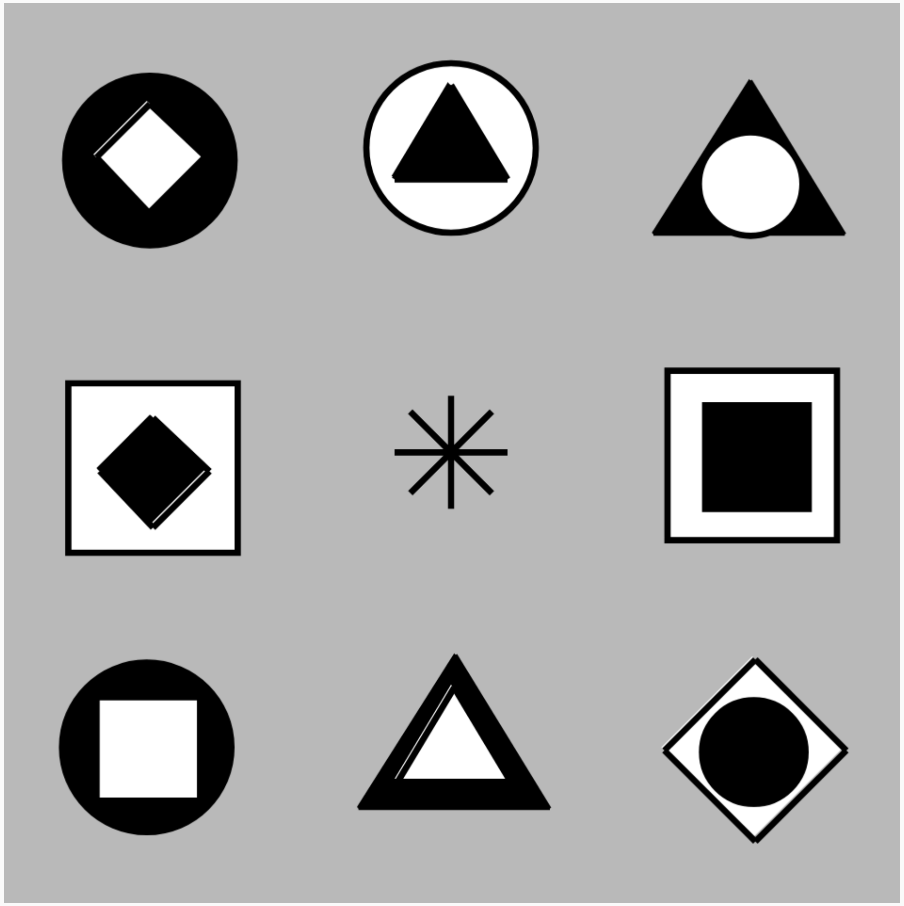


Keep your eyes fixed on the star at the center of the image

Now (without changing focus) where is the black circle surrounding a white square?

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

# Visual attention

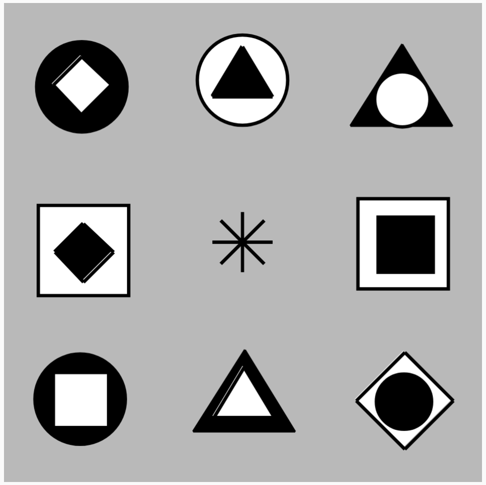


Keep your eyes fixed on the star at the center of the image

Next (without changing focus) where is the black triangle surrounding a white square?

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

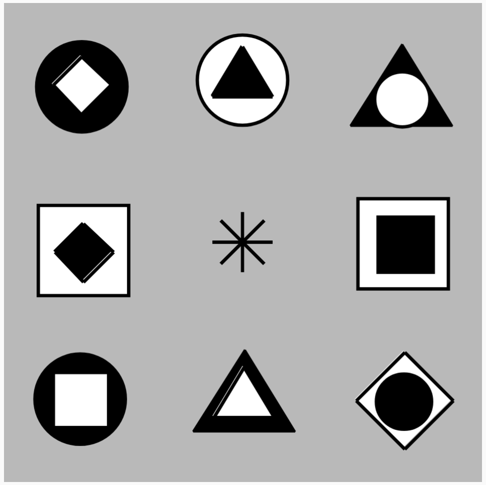
# Visual attention



To answer the questions, you needed to check one object at a time.

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

# Visual attention

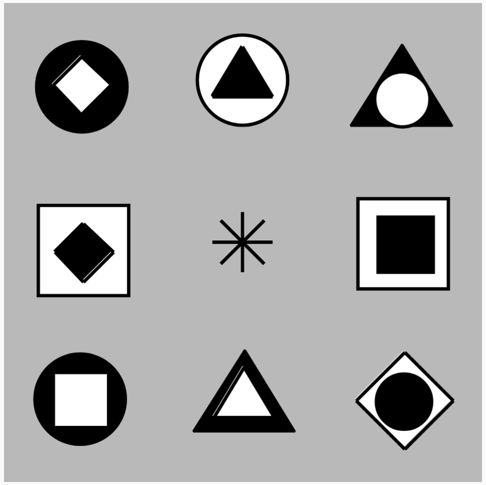


To answer the questions, you needed to check one object at a time.

If you were looking at the center of the image to answer the questions, then you internally changed how to process the input without the input changing

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

# Visual attention



To answer the questions, you needed to check one object at a time.

If you were looking at the center of the image to answer the questions, then you internally changed how to process the input without the input changing

In other words, you exercised your ***visual attention***

Wolfe J. Visual attention. In: De Valois KK, editor. Seeing. 2nd ed. San Diego, CA: Academic Press; 2000. p. 335-386.

# What is attention?

- All inputs may not need careful processing at all points of time

- Attention: A mechanism for selecting a subset of information for further analysis/processing/computation
  - *Focus* on the most relevant information, and ignore the rest

- Widely studied in cognitive psychology, neuroscience and related fields
  - Often seen in the context of visual information

# Overview

- What is attention?
- Attention in encoder-decoder networks

# Attention in NLP

- Attention is widely used in various NLP applications
- First introduced in the context of encoder-decoder networks for machine translation

# Attention in NLP

- Attention is widely used in various NLP applications
- First introduced in the context of encoder-decoder networks for machine translation
- Generally it takes the following form:
  – We have a large input, but need to focus on only a small part
  – An auxiliary network predicts a distribution over the input that decides the attention over its parts
  – The output is the weighted sum of the attention and the input

# Example application: Machine Translation

Suppose we have to convert a Dutch sentence into its English translation

*Piet de kinderen helpt zwemmen*

↓

*Piet helped the children swim*

# Example application: Machine Translation

Suppose we have to convert a Dutch sentence into its English translation

*Piet de kinderen helpt zwemmen*

↓

*Piet helped the children swim*

This requires us to consume a sequence and generate a new one that means the same

# Consuming and generating sequences

Recurrent neural networks as general sequence processors

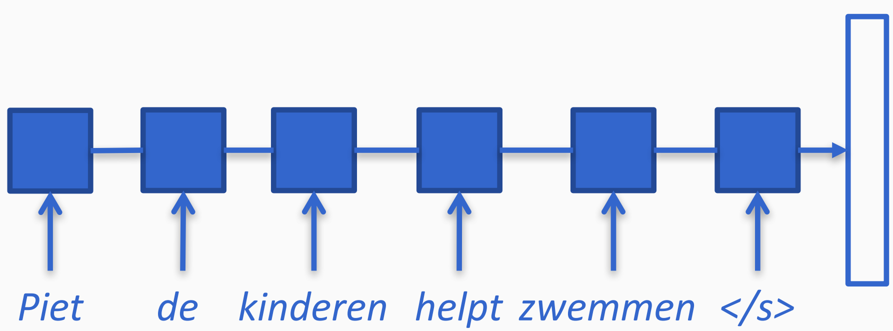
- RNNs can encode a sequence into sequence of state vectors
- RNNs can generate sequences starting with an initial input
  - And can even take inputs at each step to guide the generation

# The encoder-decoder approach

[Sutskever, et al 2014, Cho et al 2014]

Encode the input using an RNN till a special end-of-input token is reached

(Could be a bi-directional RNN)



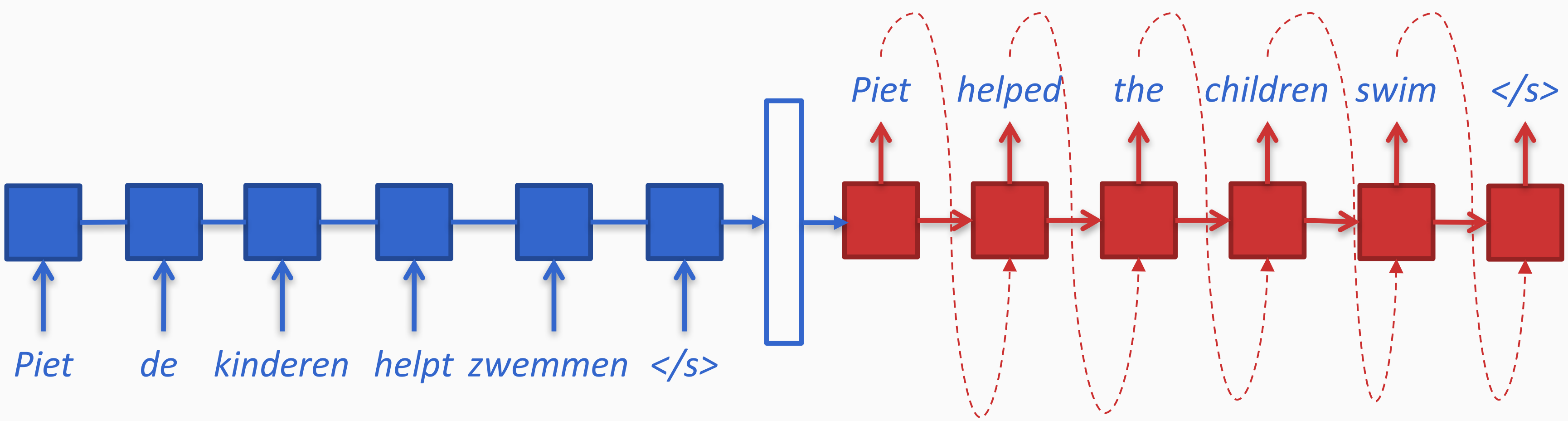
# The encoder-decoder approach

[Sutskever, et al 2014, Cho et al 2014]

Encode the input using an RNN till a special end-of-input token is reached

(Could be a bi-directional RNN)

Then generate the output using a different RNN – the decoder



# The encoder-decoder approach

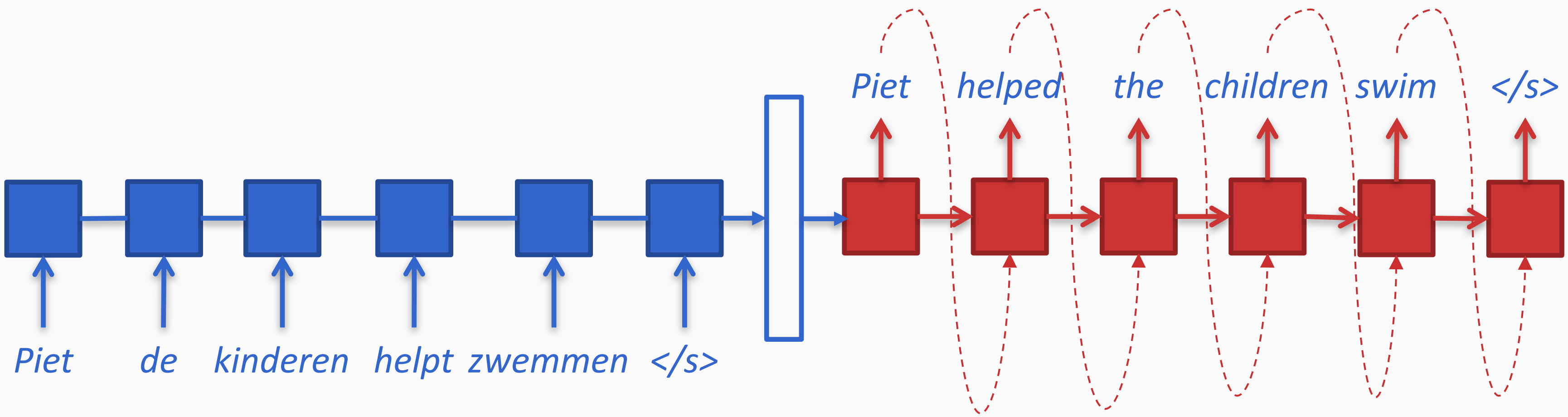
[Sutskever, et al 2014, Cho et al 2014]

Encode the input using an RNN till a special end-of-input token is reached

(Could be a bi-directional RNN)

Then generate the output using a different RNN – the decoder

The decoder produces probabilities over the output sequence words



# The encoder-decoder model: Design choices

# The encoder-decoder model: Design choices

- What RNN cell to use? Multiple layers of encoders?

# The encoder-decoder model: Design choices

- What RNN cell to use? Multiple layers of encoders?
- In what order should the inputs be consumed? In what order should the outputs be generated?
  - Eg: The decoder could produce the output in reverse order

# The encoder-decoder model: Design choices

- What RNN cell to use? Multiple layers of encoders?
- In what order should the inputs be consumed? In what order should the outputs be generated?
  - Eg: The decoder could produce the output in reverse order
- How to summarize the input sequence using the RNN?
  - Should the summary be static? Or should it be dynamically be changed as outputs are being produced?

# The encoder-decoder model: Design choices

- What RNN cell to use? Multiple layers of encoders?
- In what order should the inputs be consumed? In what order should the outputs be generated?
  - Eg: The decoder could produce the output in reverse order
- How to summarize the input sequence using the RNN?
  - Should the summary be static? Or should it be dynamically be changed as outputs are being produced?
- Should the output words be chosen greedily one at a time? Or should we use a more sophisticated search algorithm that entertains multiple sequences to find the overall best sequence?

# The encoder-decoder model: Design choices

- What RNN cell to use? Multiple layers of encoders?
- In what order should the inputs be consumed? In what order should the outputs be generated?
  – Eg: The decoder could produce the output in reverse order
- *How to summarize the input sequence using the RNN?*
  – *Should the summary be static? Or should it be dynamically be changed as outputs are being produced?*
- Should the output words be chosen greedily one at a time? Or should we use a more sophisticated search algorithm that entertains multiple sequences to find the overall best sequence?

# The encoded input

Suppose we have a fixed encoding vector (e.g. the hidden final states of the bi-LSTM in both directions)

# The encoded input

Suppose we have a fixed encoding vector (e.g. the hidden final states of the bi-LSTM in both directions)

What information should it contain?

- Information about the entire input sentence
- After each word is generated, it should somehow help keep track of what information from the input is yet to be covered

# The encoded input

Suppose we have a fixed encoding vector (e.g. the hidden final states of the bi-LSTM in both directions)

What information should it contain?

- Information about the entire input sentence
- After each word is generated, it should somehow help keep track of what information from the input is yet to be covered

In practice: such a simple encoder-decoder network works for short sentences (10-15 words)

Needs other modeling refinements to improve beyond this

# Adding attention to the decoder

[Bahdanau, 2014]

- Deciding on each output word does not depend on *all* input words
- Instead, if we can dynamically attend over the inputs for each output, then the decision of which output word to generate could be more targeted
- Let's build such a model from scratch

# Step 1: The encoder

- Input sequence of words: $x_1, x_2, \cdots$
  - Assume that the we have special start and end tokens

- Bidirectional RNN (usually LSTM) encodes the sequence to produce a sequence of hidden states

$$\mathbf{h}_i = [\overleftarrow{\mathbf{h}_{\mathrm{i}}}, \overrightarrow{\mathbf{h}_i}] = BiRNN(x_1, x_2, \cdots)$$

# Step 1: The encoder

- Input sequence of words: $x_1, x_2, \cdots$
  - Assume that the we have special start and end tokens

- Bidirectional RNN (usually LSTM) encodes the sequence to produce a sequence of hidden states

$$\mathbf{h}_i = \boxed{[\overleftarrow{\mathbf{h}}_{\mathrm{i}}, \overrightarrow{\mathbf{h}}_i]} = BiRNN(x_1, x_2, \cdots)$$

Concatenated states from the left and right RNNs

# Step 2: The decoder

- Suppose the output words are $y_1, y_2, \cdots$
- For the $i^{th}$ output word, suppose we summarize the input into a vector $\mathbf{c}_i$
  - We will look at what this vector is very soon
- The probability of $i^{th}$ output word depends on
  - The previous word generated $y_{i-1}$
  - The hidden state of the decoder, say $\mathbf{s}_{i-1}$
  - And the input summary $\mathbf{c}_i$

$$\text{softmax}(W_o y_{i-1} + W_h \mathbf{s}_{i-1} + W_c \mathbf{c}_i + b)$$

# Step 2: The decoder

- Suppose the output words are $y_1, y_2, \cdots$
- For the $i^{th}$ output word, suppose we summarize the input into a vector $\mathbf{c}_i$
  - We will look at what this vector is very soon
- The probability of $i^{th}$ output word depends on
  - The previous word generated $y_{i-1}$
  - The hidden state of the decoder, say $\mathbf{s}_{i-1}$
  - And the input summary $\mathbf{c}_i$

$$\text{softmax}(W_o y_{i-1} + W_h \mathbf{s}_{i-1} + W_c \mathbf{c}_i + b)$$

The previous word is represented by its embedding

# Step 2: The decoder

- Suppose the output words are $y_1, y_2, \cdots$
- For the $i^{th}$ output word, suppose we summarize the input into a vector $\mathbf{c}_i$
  - We will look at what this vector is very soon
- The probability of $i^{th}$ output word depends on
  - The previous word generated $y_{i-1}$
  - The hidden state of the decoder, say $\mathbf{s}_{i-1}$
  - And the input summary $\mathbf{c}_i$

$$\text{softmax}(W_o y_{i-1} + W_h \mathbf{s}_{i-1} + W_c \mathbf{c}_i + b)$$

# Step 2: The decoder

- Suppose the output words are $y_1, y_2, \cdots$
- For the $i^{th}$ output word, suppose we summarize the input into a vector $\mathbf{c}_i$
  - We will look at what this vector is very soon
- The probability of $i^{th}$ output word depends on
  - The previous word generated $y_{i-1}$
  - The hidden state of the decoder, say $\mathbf{s}_{i-1}$
  - And the input summary $\mathbf{c}_i$

$$\text{softmax}(W_o y_{i-1} + W_h \mathbf{s}_{i-1} + W_c \mathbf{c}_i + b)$$

# Step 2: The decoder

- Suppose the output words are $y_1, y_2, \cdots$

- For the $i^{th}$ output word, suppose we summarize the input into a vector $\mathbf{c}_i$
  – We will look at what this vector is very soon

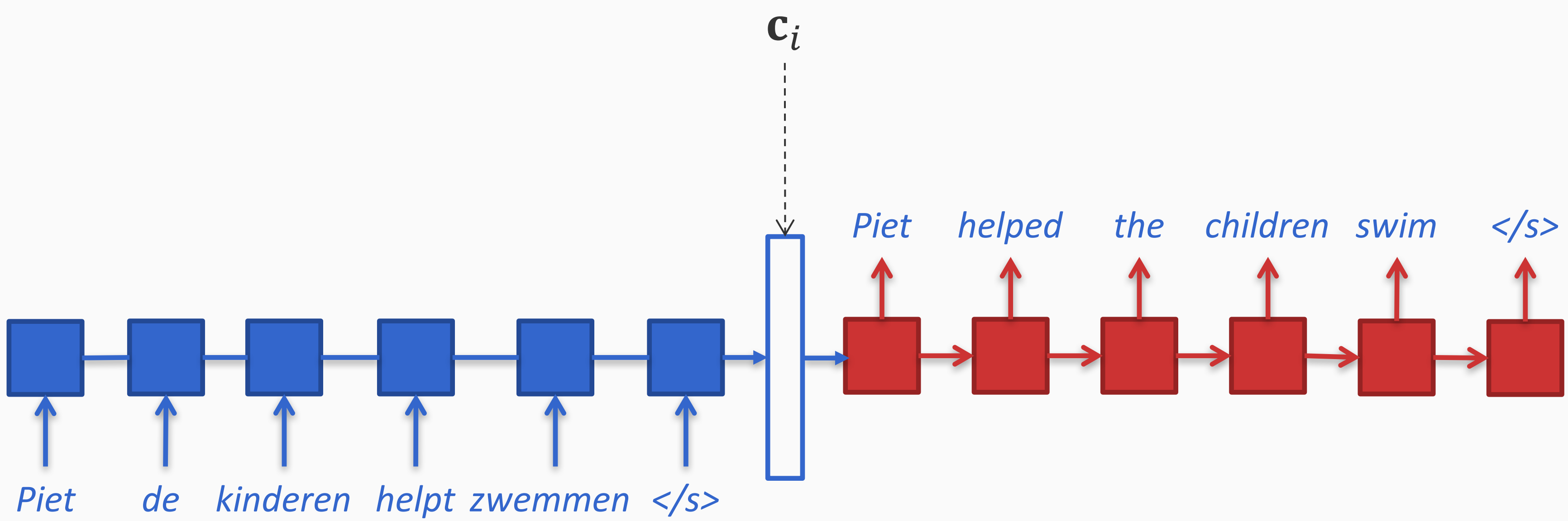
- The probability of $i^{th}$ output word depends on
  – The previous word generated $y_{i-1}$
  – The hidden state of the decoder, say $\mathbf{s}_{i-1}$
  – And the input summary $\mathbf{c}_i$

$$\text{softmax}(W_o y_{i-1} + W_h \mathbf{s}_{i-1} + W_c \mathbf{c}_i + b)$$

Probability over all the target words

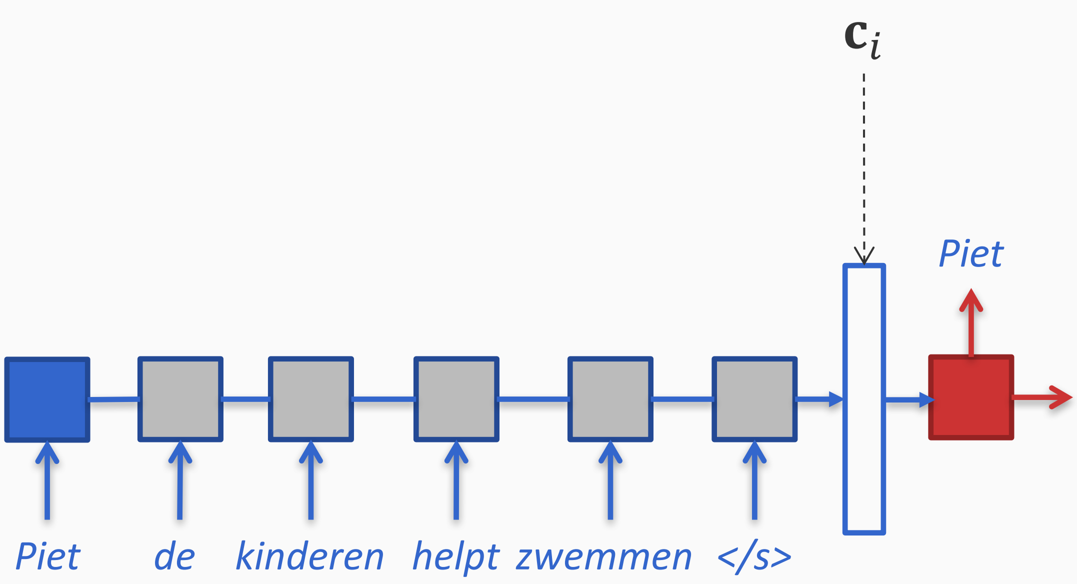
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



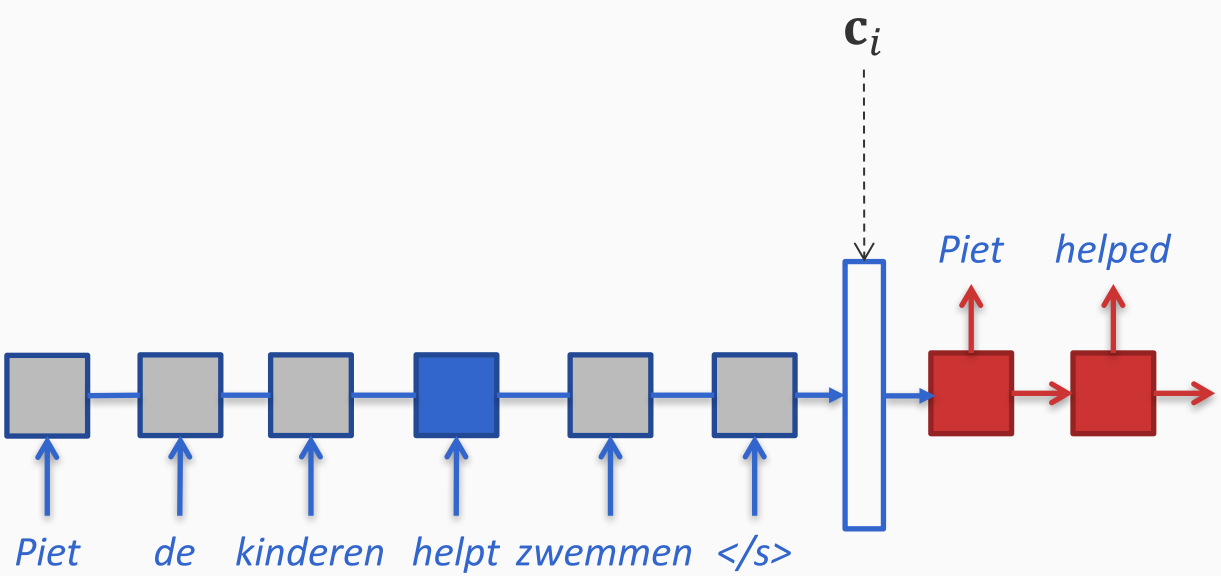
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



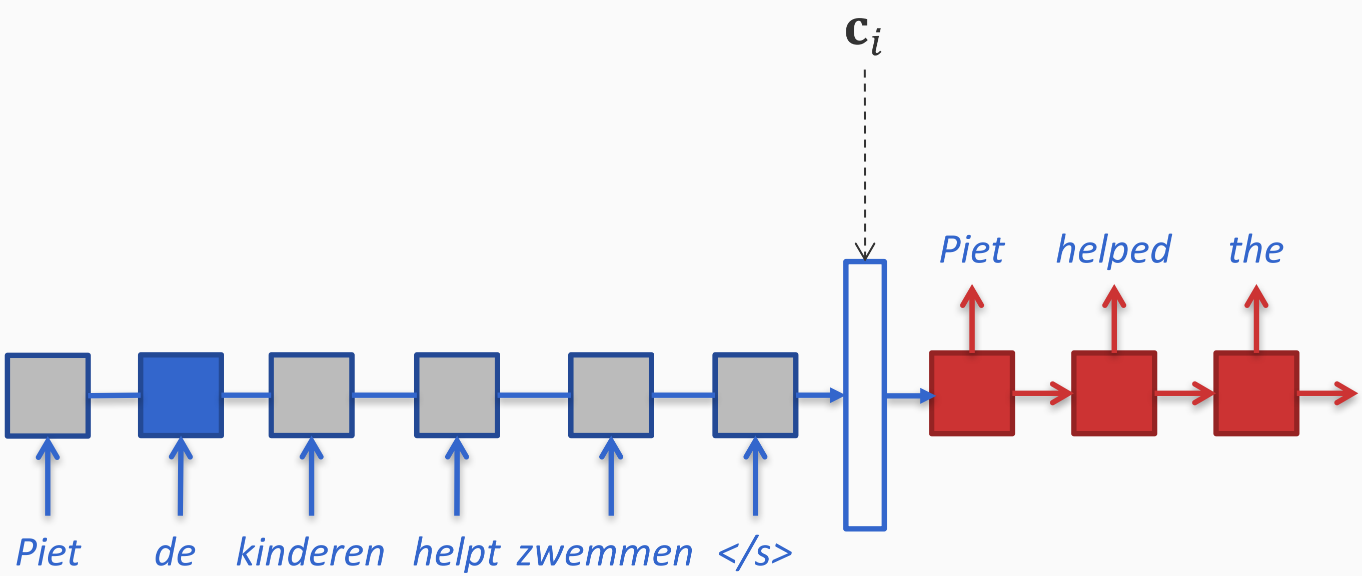
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



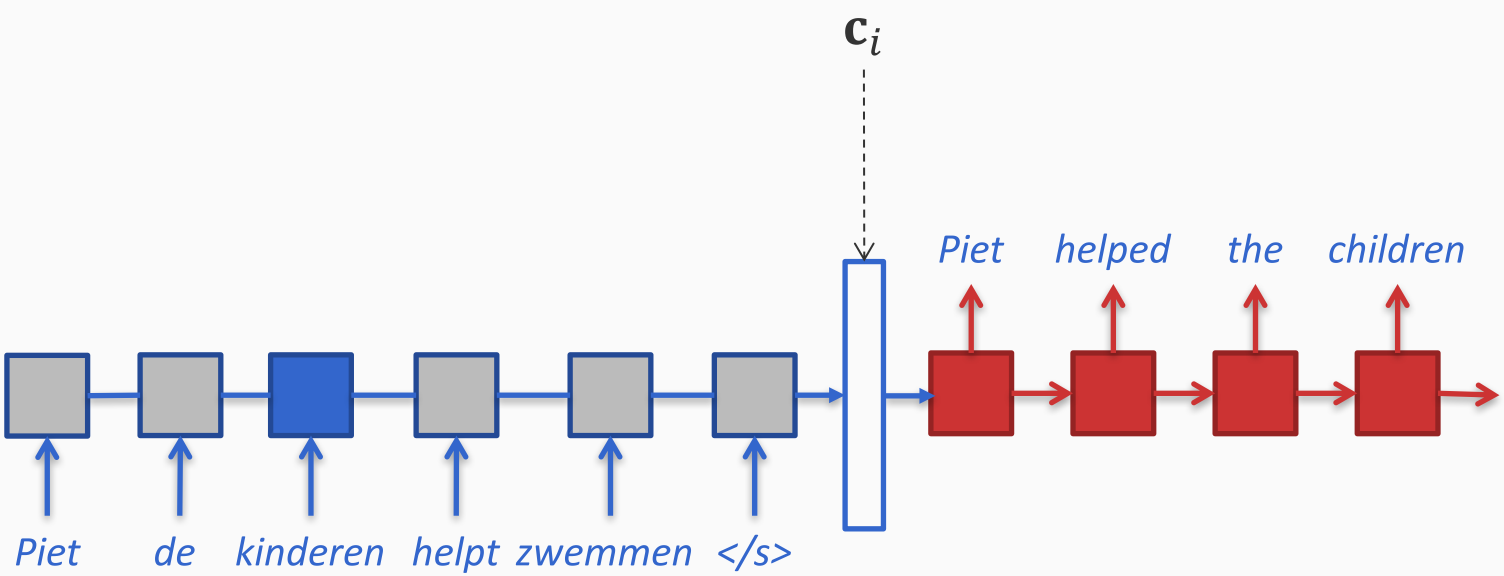
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



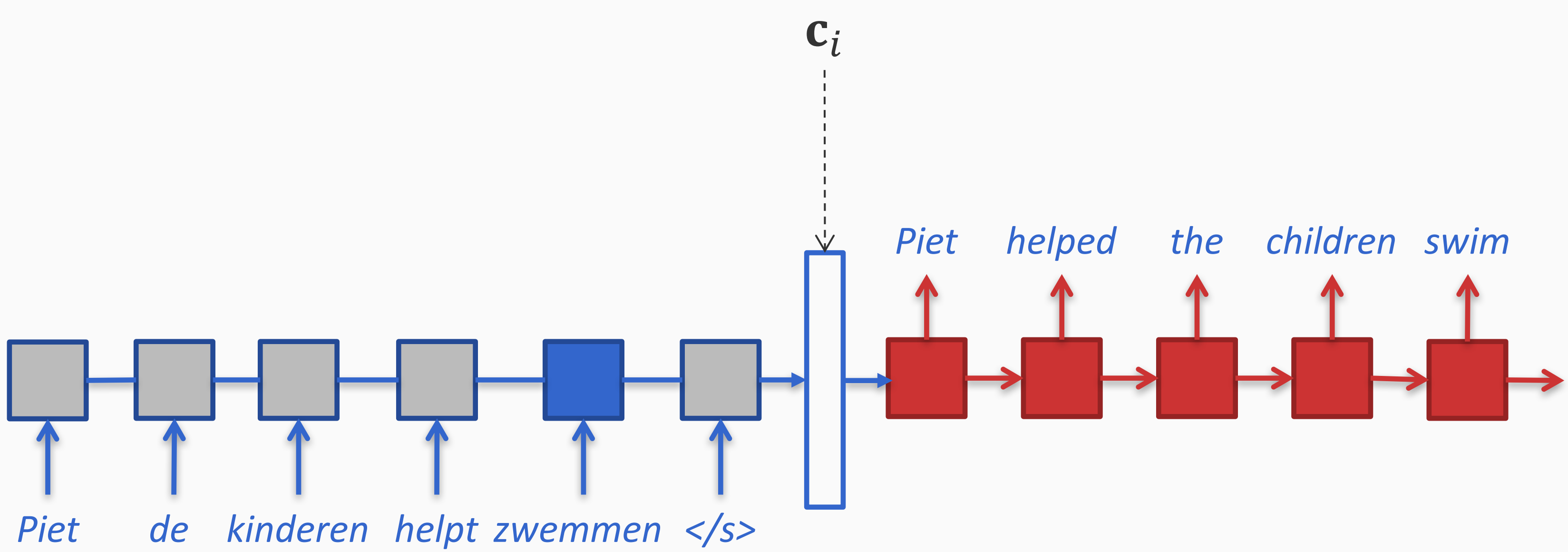
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



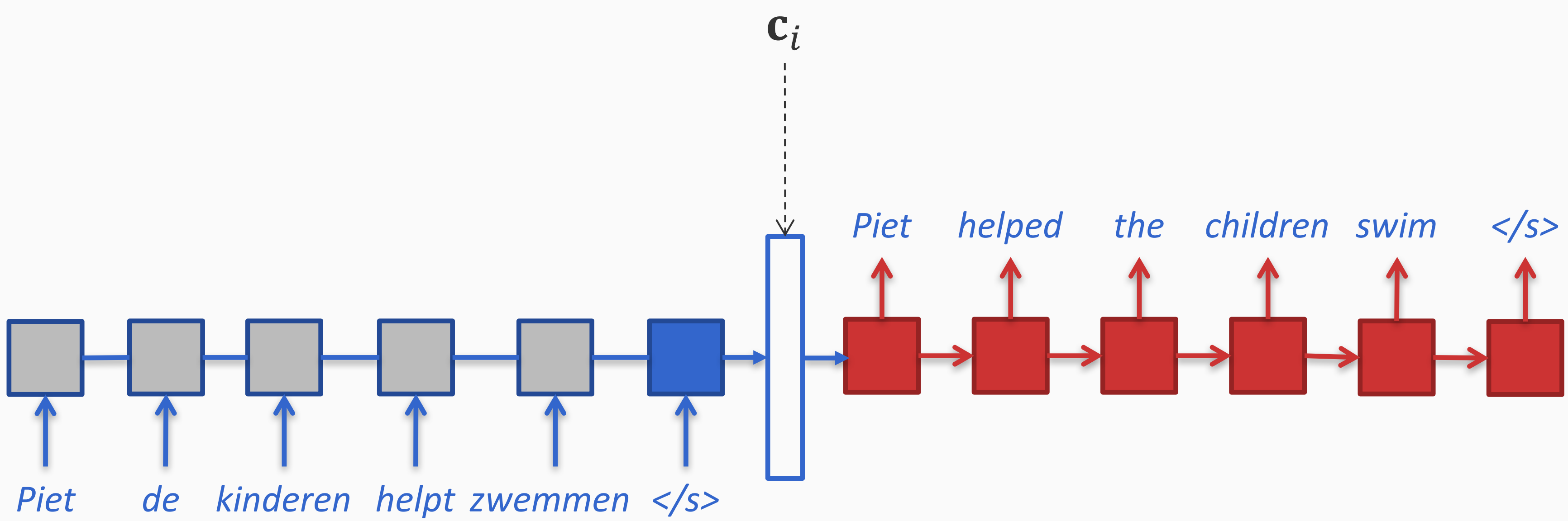
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



# Summarizing inputs for generating outputs

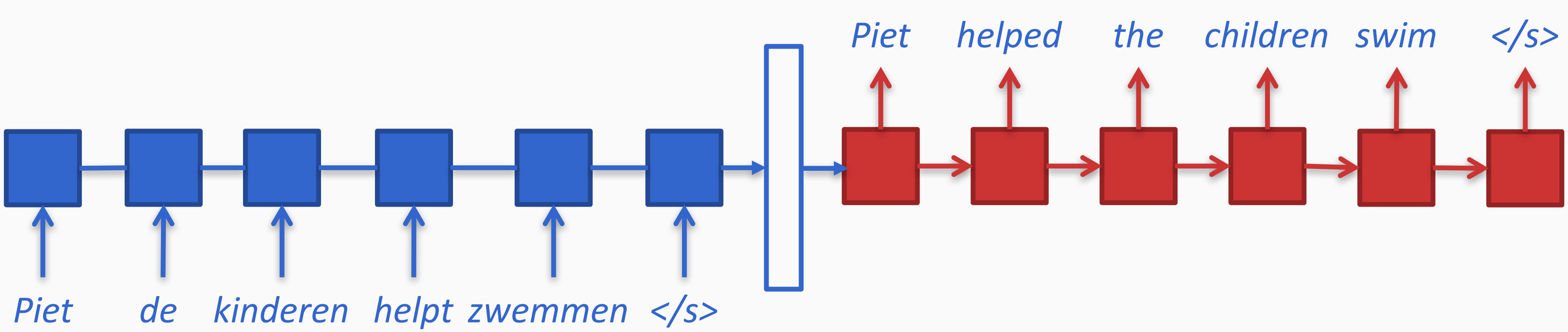
At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated



# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

- At each step, this can be seen as a decision: *Which word is currently relevant?*

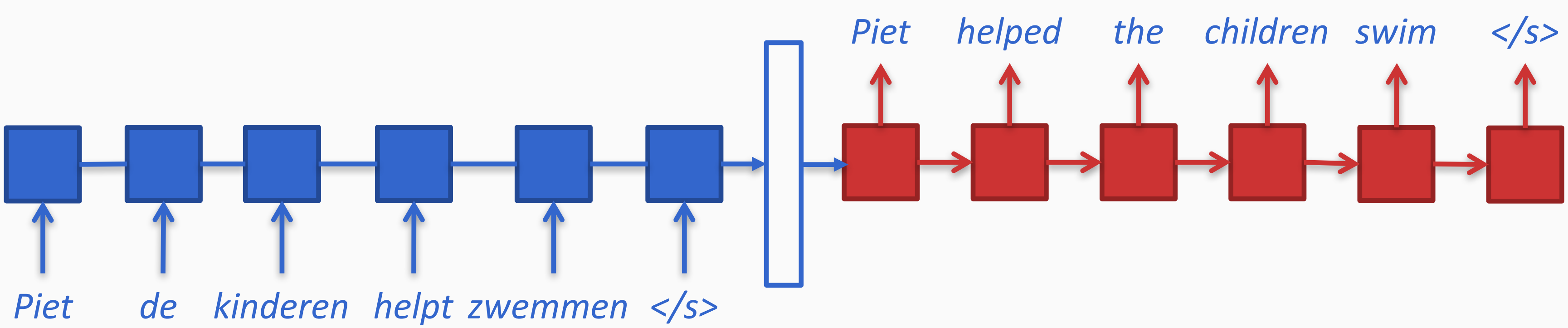


# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

At each step, this can be seen as a decision: *Which word is currently relevant?*

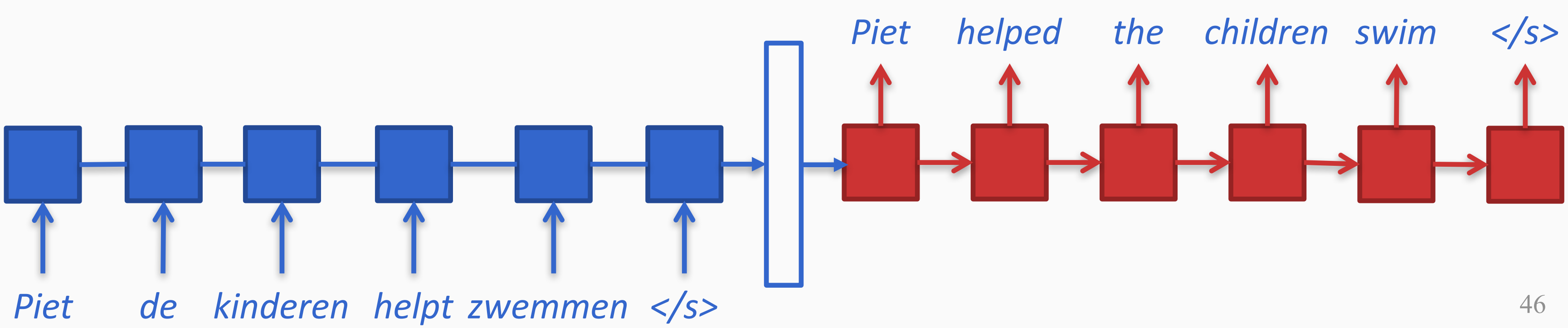
Instead of a hard decision, we can ask for a soft decision: a probability



# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

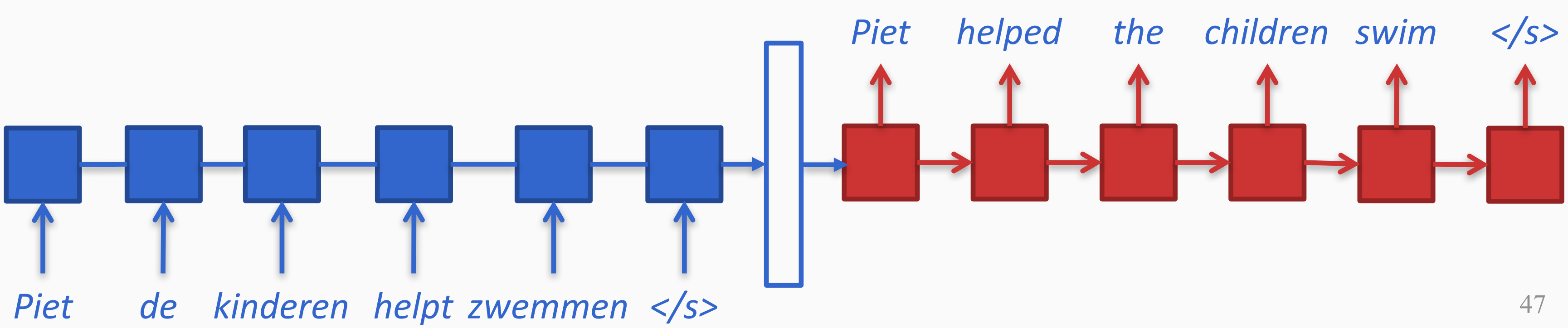
Let's see how we can construct the encoding using such a mechanism



# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

1. Attention over input words: A number for the $j^{th}$ input word

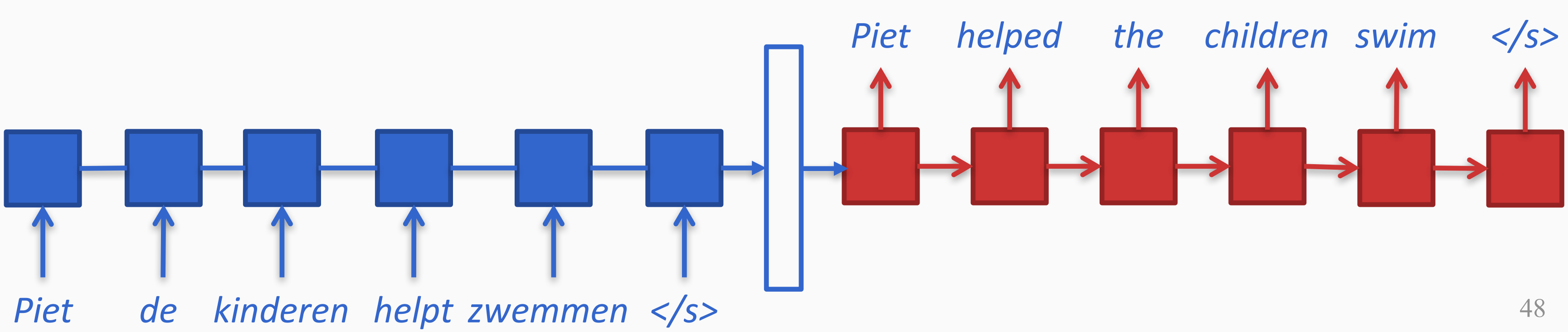


# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

1. Attention over input words: A number for the $j^{th}$ input word

$$a(s_{i-1}, h_j) = W_a s_{i-1} + W_I h_j + b$$



# Summarizing inputs for generating outputs

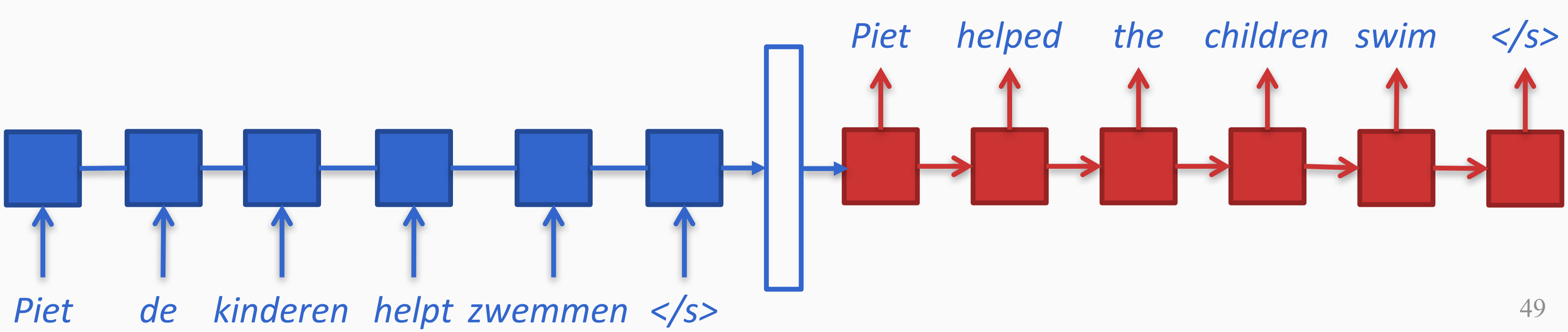
At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

1. Attention over input words: A number for the $j^{th}$ input word

A score that depends on the current state of the decoder and the word encodings

$$a(s_{i-1}, h_j) = W_a s_{i-1} + W_I h_j + b$$

Characterizes how important the $j^{th}$ input word is at this point



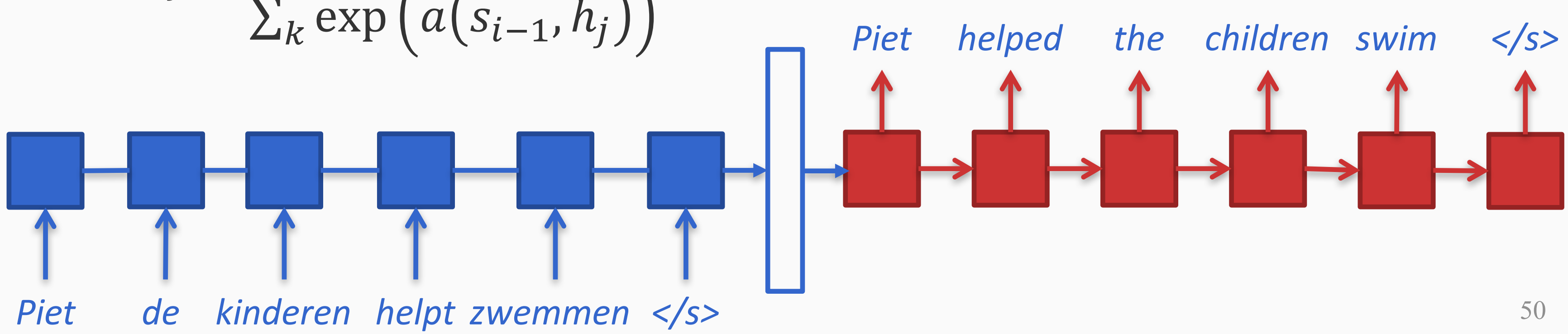
# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

1. Attention over input words: A number for the $j^{th}$ input word

$$a(s_{i-1}, h_j) = W_a s_{i-1} + W_I h_j + b$$

$$a_{ij} = \frac{\exp\left(a(s_{i-1}, h_j)\right)}{\sum_k \exp\left(a(s_{i-1}, h_j)\right)}$$



# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

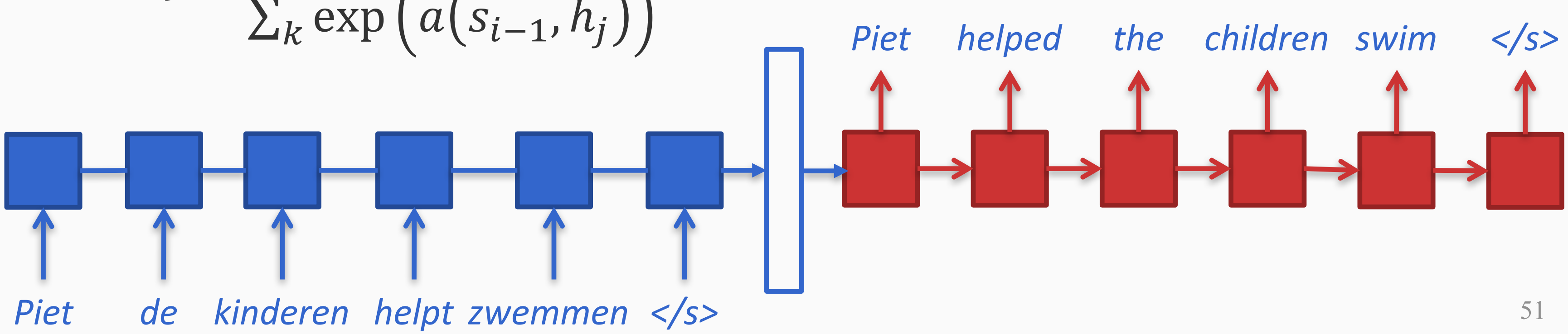
1. Attention over input words: A number for the $j^{th}$ input word

$$a(s_{i-1}, h_j) = W_a s_{i-1} + W_I h_j + b$$

$$a_{ij} = \frac{\exp\left(a(s_{i-1}, h_j)\right)}{\sum_k \exp\left(a(s_{i-1}, h_j)\right)}$$

Convert this into a probability by taking softmax over the inputs

**What we have**: A distribution over inputs at each step of the decoder



# Summarizing inputs for generating outputs

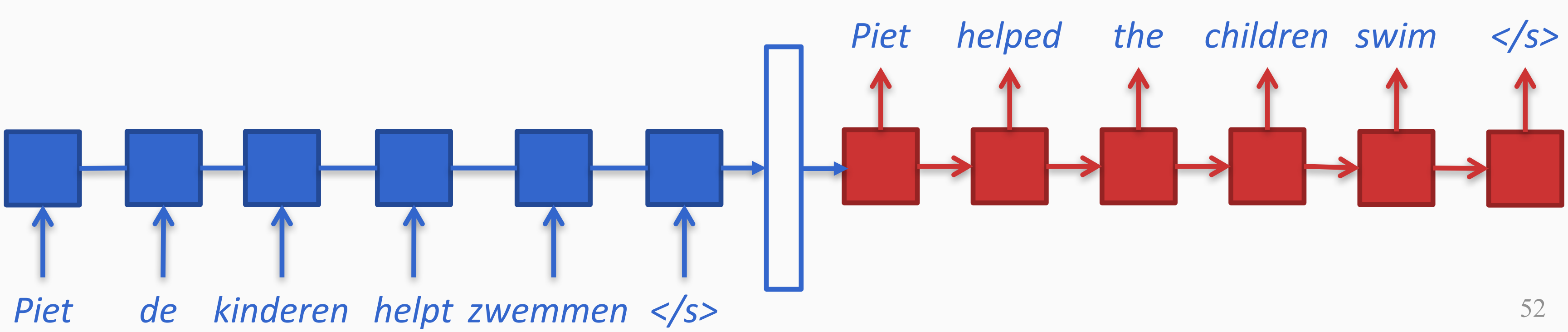
At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

1. Attention over input words: A number for the $j^{th}$ input word

$$a_{ij} = \frac{\exp\left(a\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a\left(s_{i-1}, h_j\right)\right)}$$

2. Attended encoding: At each step

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{h}_j$$



# Summarizing inputs for generating outputs

At the $i^{th}$ step, the vector $\mathbf{c}_i$ should highlight information about the input words that is being translated

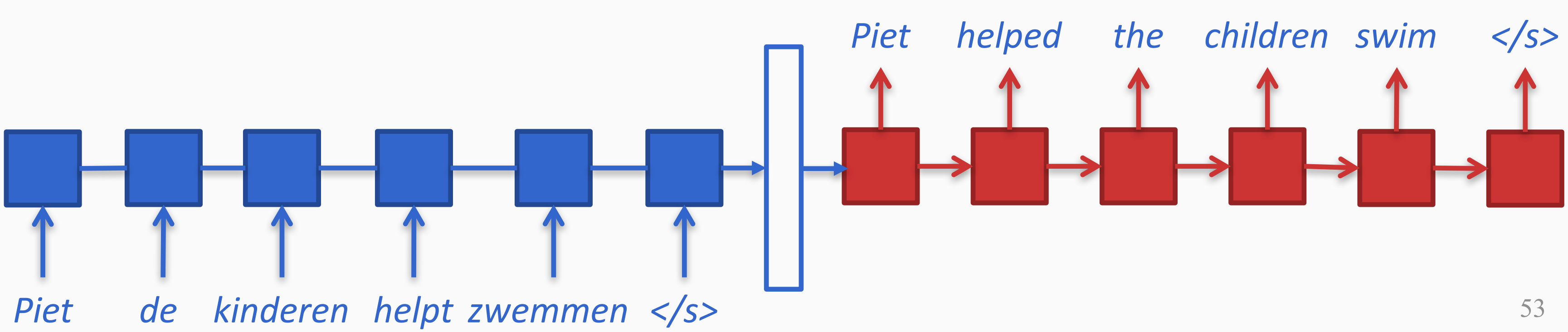
1. Attention over input words: A number for the $j^{th}$ input word

$$a_{ij} = \frac{\exp\left(a\left(s_{i-1}, h_j\right)\right)}{\sum_k \exp\left(a\left(s_{i-1}, h_j\right)\right)}$$

2. Attended encoding: At each step

$$\mathbf{c}_i = \sum_j a_{ij} \mathbf{h}_j$$

A weighted average of the word encodings



# Overview

- What is attention
- Attention in encoder-decoder networks
- Various kinds of attention

# General idea of attention

- Given a prediction problem whose inputs consist of many sub-components
  - The sub-components may be encoded (e.g. with word embeddings, hidden states of RNNs)
  - Or they may be the intermediate nodes in a larger network
  - We will refer to these as $\mathbf{h}_1, \mathbf{h}_2, \cdots$

# General idea of attention

- Given a prediction problem whose inputs consist of many sub-components
  - The sub-components may be encoded (e.g. with word embeddings, hidden states of RNNs)
  - Or they may be the intermediate nodes in a larger network
  - We will refer to these as $\mathbf{h}_1, \mathbf{h}_2, \cdots$

- We have a summary of a current state of the system
  - Represents the context under which we need to find attention
  - Refer to this as $\boldsymbol{s}$

# General idea of attention

- Given a prediction problem whose inputs consist of many sub-components
  - The sub-components may be encoded (e.g. with word embeddings, hidden states of RNNs)
  - Or they may be the intermediate nodes in a larger network
  - We will refer to these as $\mathbf{h}_1, \mathbf{h}_2, \cdots$

- We have a summary of a current state of the system
  - Represents the context under which we need to find attention
  - Refer to this as $\mathbf{s}$

- **The goal:** Find a distribution of the $\mathbf{h}_1, \mathbf{h}_2, \cdots$ that captures how relevant each of them are in the current state
  - Attention = softmax(some function of $\mathbf{h}_1, \mathbf{h}_2, \cdots$ and $\mathbf{s}$)

# General idea of attention

- Given a prediction problem whose inputs consist of many sub-components
  - The sub-components may be encoded (e.g. with word embeddings, hidden states of RNNs)
  - Or they may be the intermediate nodes in a larger network
  - We will refer to these as $\mathbf{h}_1, \mathbf{h}_2, \cdots$

Sometimes this is called the ***source*** sequence

- We have a summary of a current state of the system
  - Represents the context under which we need to find attention
  - Refer to this as $\mathbf{s}$

- **The goal:** Find a distribution of the $\mathbf{h}_1, \mathbf{h}_2, \cdots$ that captures how relevant each of them are in the current state
  - Attention = softmax(some function of $\mathbf{h}_1, \mathbf{h}_2, \cdots$ and $\mathbf{s}$)

# What we saw so far: Additive attention

1. Compute a score for each sub-component of the input

$$a\left(\mathbf{s}, \mathbf{h}_j\right) = \mathrm{W}_a\mathbf{s} + W_I\mathbf{h}_j + b$$

# What we saw so far: Additive attention

1. Compute a score for each sub-component of the input

$$a(\mathbf{s}, \mathbf{h}_j) = \mathrm{W}_a \mathbf{s} + W_I \mathbf{h}_j + b$$

2. Normalize with softmax to get attention

$$\text{Attention } a_j = \frac{\exp\left(a(\mathbf{s}, \mathbf{h}_j)\right)}{\sum_k \exp\left(a(\mathbf{s}, \mathbf{h}_k)\right)}$$

# What we saw so far: Additive attention

1. Compute a score for each sub-component of the input

$$a(\mathbf{s}, \mathbf{h}_j) = \mathrm{W}_a\mathbf{s} + W_I\mathbf{h}_j + b$$

2. Normalize with softmax to get attention

$$\text{Attention } a_j = \frac{\exp\left(a(\mathbf{s}, \mathbf{h}_j)\right)}{\sum_k \exp(a(\mathbf{s}, \mathbf{h}_k))}$$

Why should the score be additive?
Maybe other functions are possible

# Different scoring functions for attention

| Name | Scoring function $a(\mathbf{s}, \mathbf{h}_j)$ | Reference |
|---|---|---|
| Additive attention | $W_{\mathbf{a}}\mathbf{s} + W_I\mathbf{h}_j + b$ | Bahdanau et al 2015 |

We have already seen this

# Different scoring functions for attention

| Name | Scoring function $a(\mathbf{s}, \mathbf{h}_j)$ | Reference |
|---|---|---|
| Additive attention | $W_{\mathbf{a}}\mathbf{s} + W_I\mathbf{h}_j + b$ | Bahdanau et al 2015 |
| Dot product | $\mathbf{s}^T\mathbf{h_j}$ | Luong et a l2015 |
| Generalized dot product | $\mathbf{s}^T \mathrm{W}\mathbf{h_j}$ | Luong et al 2015 |

# Different scoring functions for attention

| **Name** | **Scoring function** $a(\mathbf{s}, \mathbf{h}_j)$ | **Reference** |
|---|---|---|
| Additive attention | $W_{\mathbf{a}}\mathbf{s} + W_I\mathbf{h}_j + b$ | Bahdanau et al 2015 |
| Dot product | $\mathbf{s}^T\mathbf{h_j}$ | Luong et a l2015 |
| Generalized dot product | $\mathbf{s}^T\mathrm{W}\mathbf{h_j}$ | Luong et al 2015 |
| Scaled dot product | $\frac{\mathbf{s}^T\mathbf{h_j}}{\sqrt{\mathrm{n}}}$ | Vaswani et al 2017 |

We will see this in more detail when we visit Transformers

# Different scoring functions for attention

| **Name** | **Scoring function** $a(\mathbf{s}, \mathbf{h}_j)$ | **Reference** |
|---|---|---|
| Additive attention | $W_{\mathbf{a}}\mathbf{s} + W_I\mathbf{h}_j + b$ | Bahdanau et al 2015 |
| Dot product | $\mathbf{s}^T\mathbf{h_j}$ | Luong et a l2015 |
| Generalized dot product | $\mathbf{s}^T\mathrm{W}\mathbf{h_j}$ | Luong et al 2015 |
| Scaled dot product | $\frac{\mathbf{s}^T\mathbf{h_j}}{\sqrt{\mathrm{n}}}$ | Vaswani et al 2017 |

In all cases, after the scoring function is applied, we have a softmax to produce the attention probability

# Hard vs soft attention

- Attention is a probability over the input sub-components
  - How relevant is each component in the context of a state **s**?
  - Also called soft attention

# Hard vs soft attention

- Attention is a probability over the input sub-components
  - How relevant is each component in the context of a state **s**?
  - Also called soft attention

- What if there are *many* sub-components?
  - *Needs an expensive softmax*
  - Can we avoid this?

# Hard vs soft attention

- Attention is a probability over the input sub-components
  - How relevant is each component in the context of a state **s**?
  - Also called soft attention
- What if there are *many* sub-components?
  - *Needs an expensive softmax*
  - Can we avoid this?
- Hard attention: Select one of the components – the argmax
  - Less computation
  - But not differentiable. Involves reinforcement learning for training

# Self-attention Also called *intra-attention*

- So far: We have a sequence of inputs and a separate description of the current state
  - We want to compute attention over the inputs

# Self-attention

Also called *intra-attention*

- So far: We have a sequence of inputs and a separate description of the current state
  - We want to compute attention over the inputs
- Suppose the "current" state is an element of the sequence
  - And we repeat this for each element
  - In our notation from before, $\mathbf{s}$ is one of the $\mathbf{h}_j$'s

# Self-attention Also called *intra-attention*

- So far: We have a sequence of inputs and a separate description of the current state
  - We want to compute attention over the inputs
- Suppose the “current” state is an element of the sequence
  - And we repeat this for each element
  - In our notation from before, $\mathbf{s}$ is one of the $\mathbf{h}_j$’s
- Intuition: Compute attention over a sentence with respect to each word in the sentence
  - Captures interactions between the words of a sentence

# Self-attention example

Cheng et al 2016

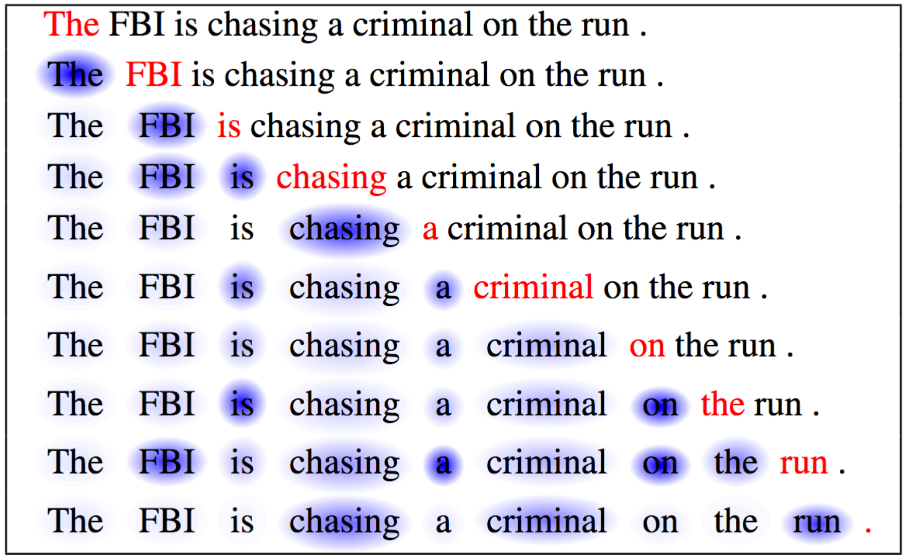



Figure 1: Illustration of our model while reading the sentence *The FBI is chasing a criminal on the run.* Color *red* represents the current word being fixated, *blue* represents memories. Shading indicates the degree of memory activation.

# Why is self-attention interesting?

- Allows for contextual encoding of words
  - Weighted average of the attended word encodings
- Unlike a recurrent neural network, there is no sequential dependencies
  - Better parallelism for contextual encodings
- Forms the basis of more sophisticated models such as the Transformer architecture