Sequence-to-Sequence (Seq2Seq) Models

Naeemullah Khan

naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

Table of Contents

Motivation

Why Do We Need Seq2Seq and Attention?

- Many real-world problems require transforming one sequence to another:
  - Translation: "Bonjour" $\rightarrow$ "Hello"
  - Dialogue systems: Question $\rightarrow$ Response
  - Speech: Audio $\rightarrow$ Text

Standard RNNs struggle with input/output sequences of different lengths and long-term dependencies.

Seq2Seq models + attention solve this with a powerful encoder-decoder framework.

Learning Outcomes

By the end of this session, you should be able to:

- Explain the Seq2Seq architecture and encoder-decoder framework
- Understand the bottleneck problem in fixed-size representations
- Describe the motivation for and core idea behind attention mechanisms
- Appreciate how attention improves performance in NLP tasks
- Recognize future directions in attention-based modeling

Seq2Seq Architecture

Key Idea: Map input sequence $\rightarrow$ intermediate vector $\rightarrow$ output sequence.

- **Encoder RNN:** Processes input sequence and compresses it into a **fixed-length vector** (context).
- **Decoder RNN:** Generates output sequence from the context vector.

Applications:

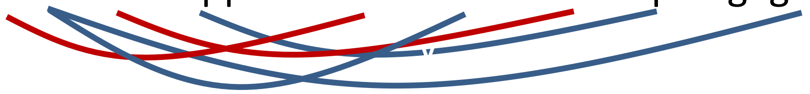
- Machine translation
- Summarization
- Dialogue systems
- Speech recognition

Sequence to sequence models

Seq2seq   I ate an apple

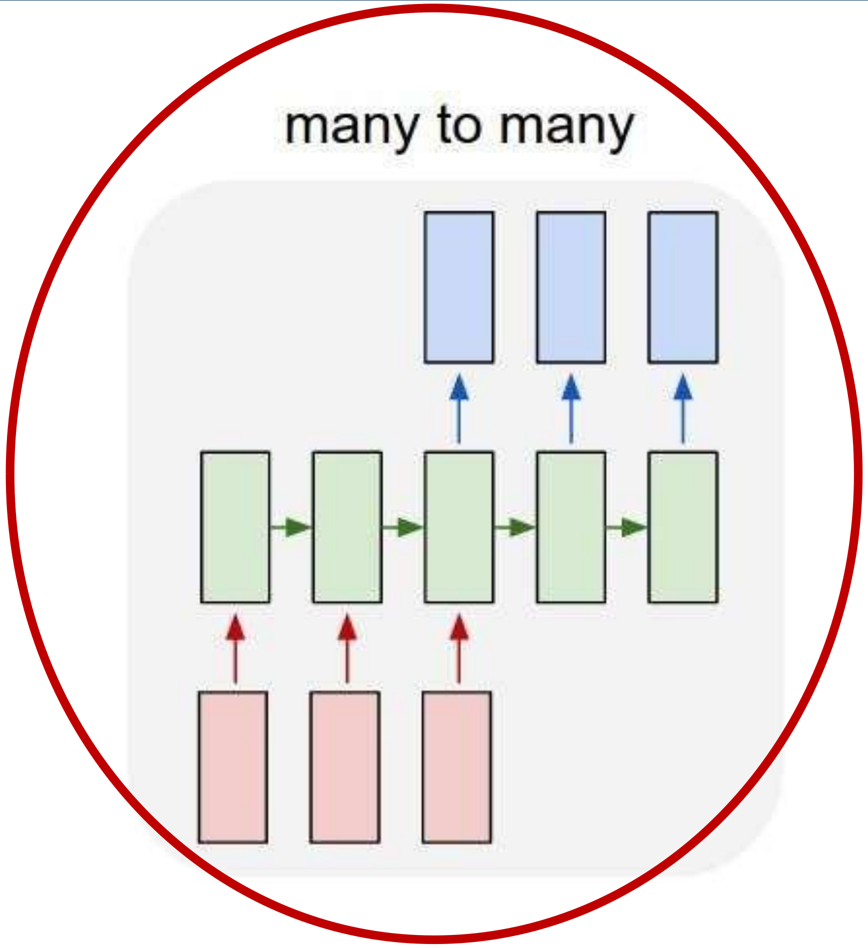
I ate an apple $\longrightarrow$ Seq2seq $\longrightarrow$ Ich habe einen apfel gegessen

- Sequence goes in,   sequence comes out
- No notion of "time synchrony" between input and output
  - May even not even maintain order of symbols
    - E.g.   "I ate an apple" 🡪 "Ich habe einen apfel gegessen"



  - Or even seem related to the input
    - E.g. "My screen is blank" 🡪 "Please check if your computer is plugged in."

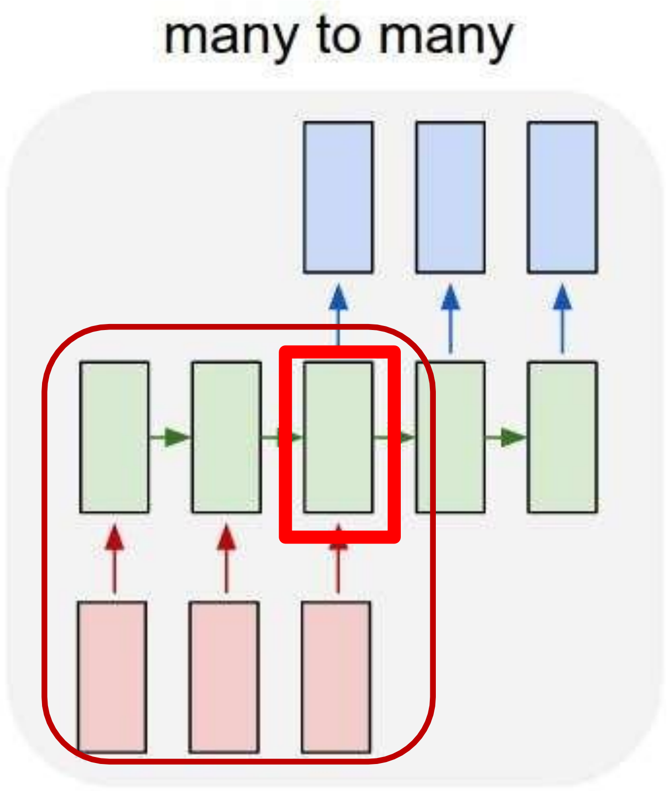
Modelling the problem



- *Delayed* sequence to sequence

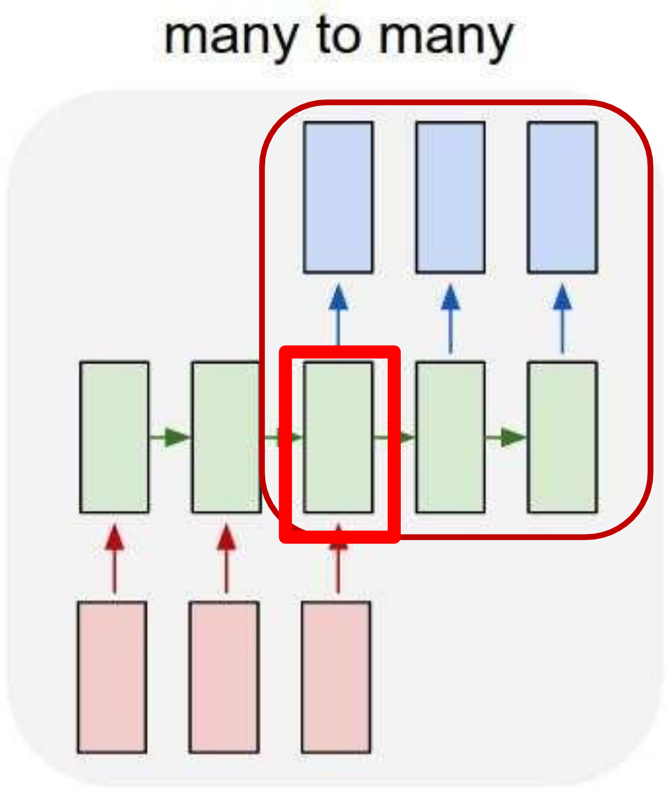
Modelling the problem

First process the input and generate a hidden representation for it



- *Delayed* sequence to sequence

Modelling the problem



First process the input and generate a hidden representation for it

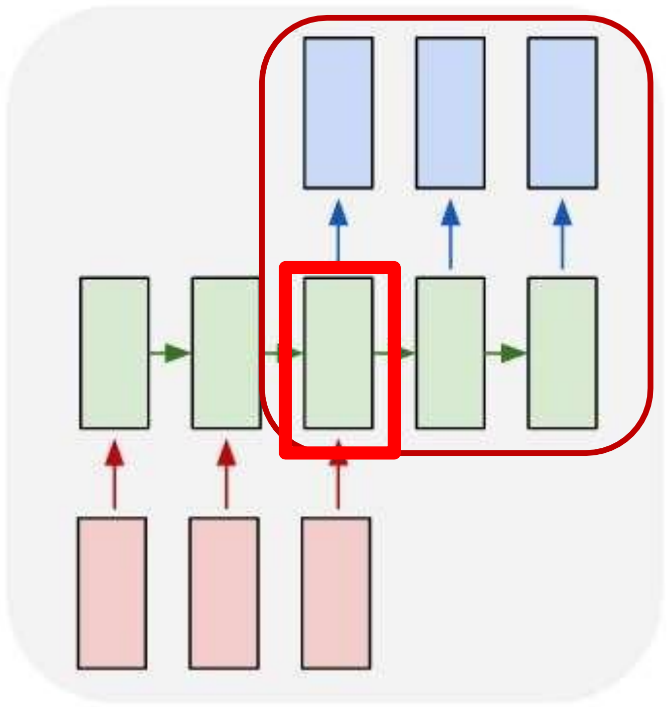
Then use it to generate an output

- *Delayed* sequence to sequence

Modelling the problem

many to many

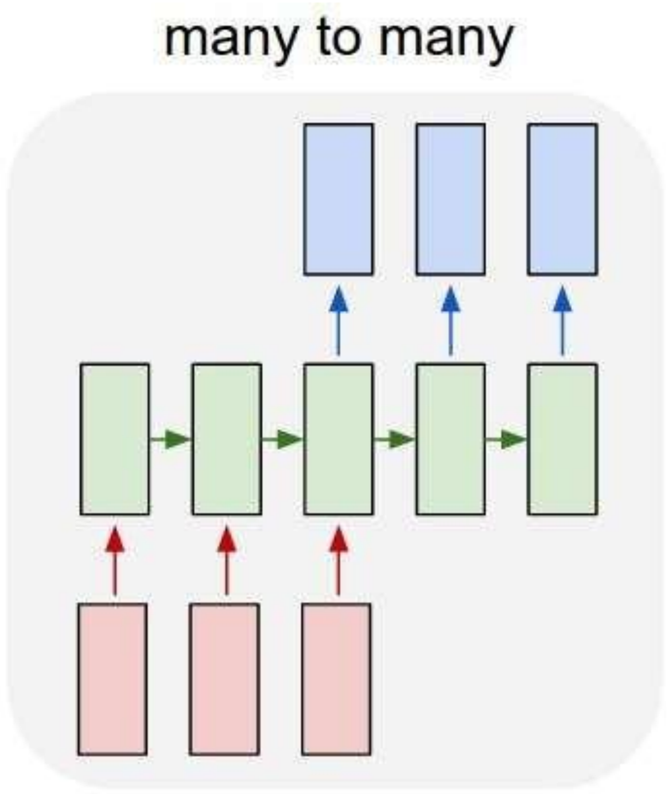
First process the input and generate a hidden representation for it

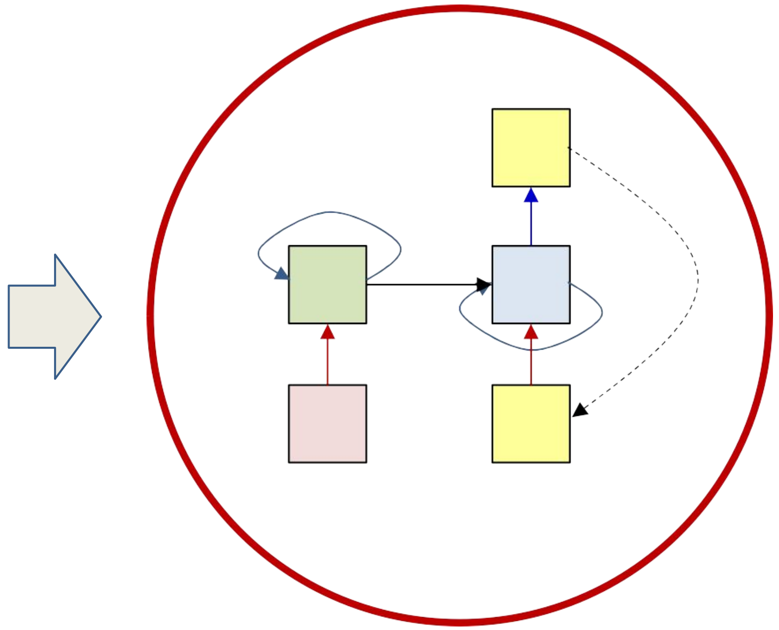


Then use it to generate an output

- *Problem:* Each word that is output depends only on current hidden state, and not on previous outputs

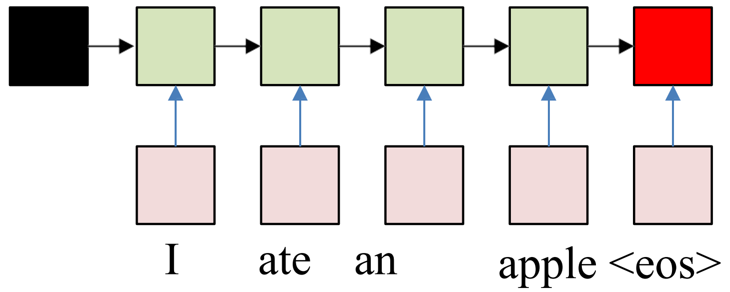
Modelling the problem

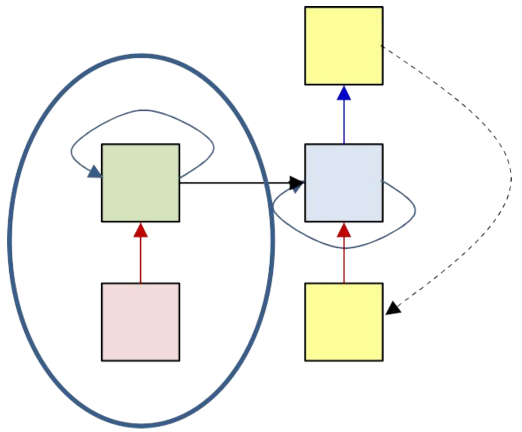




- *Delayed* sequence to sequence
  - Delayed *self-referencing* sequence-to-sequence

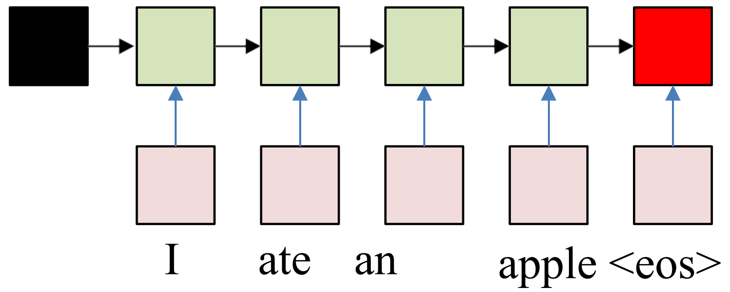
The “simple” translation model

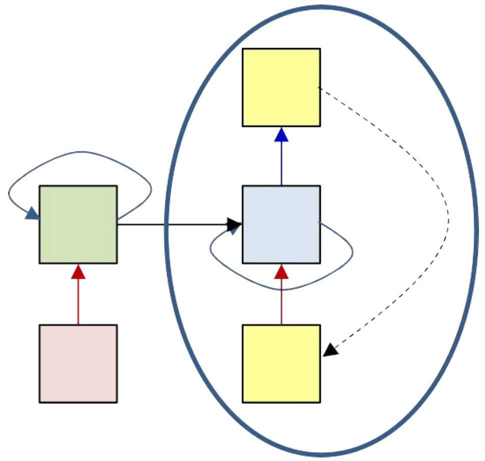




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

The “simple” translation model

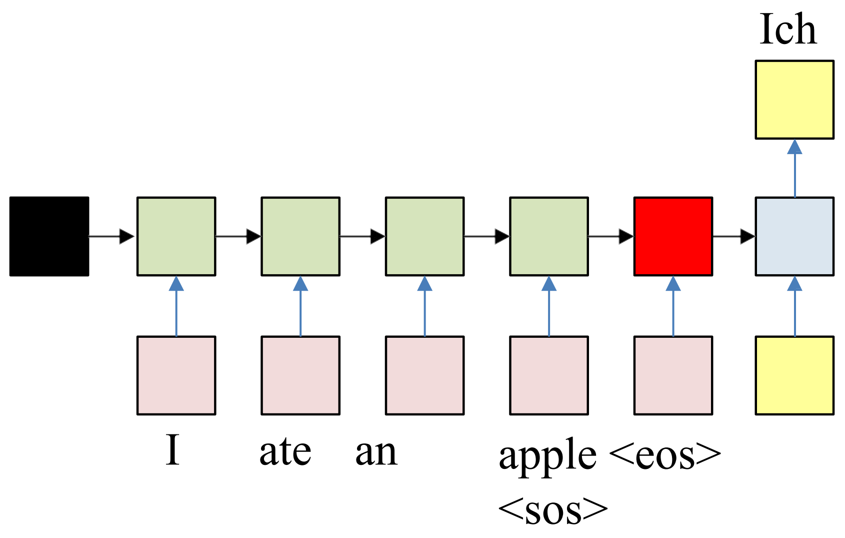




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a *second* RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

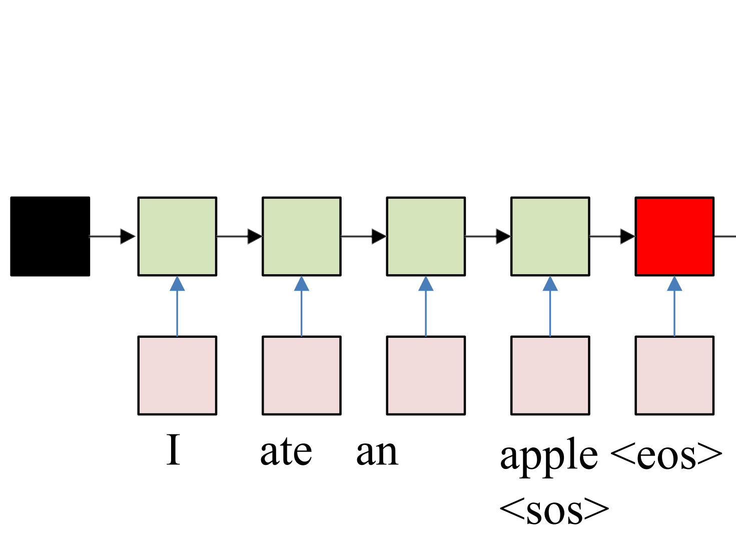
The “simple” translation model



- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

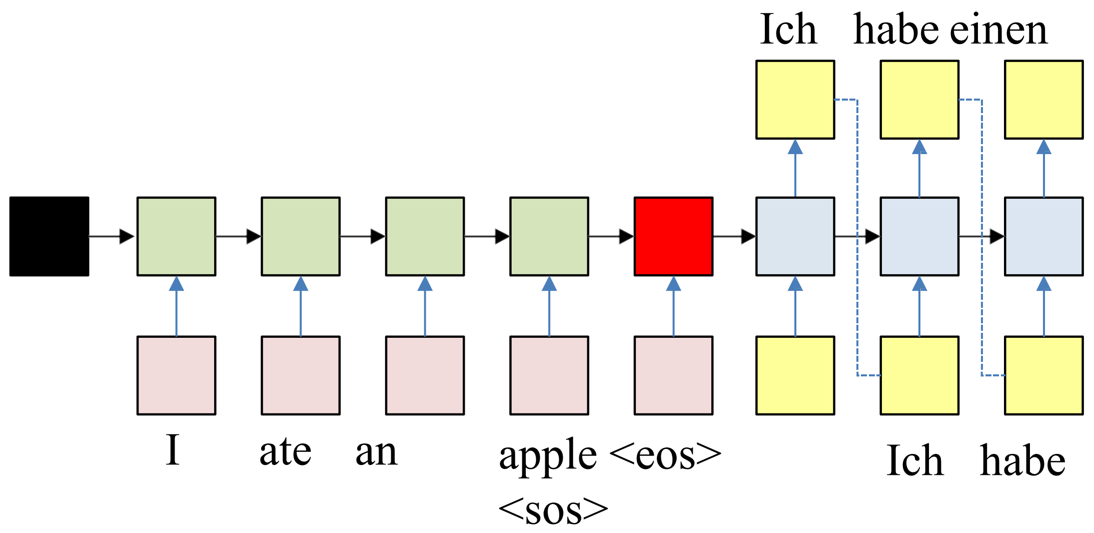
The “simple” translation model



- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

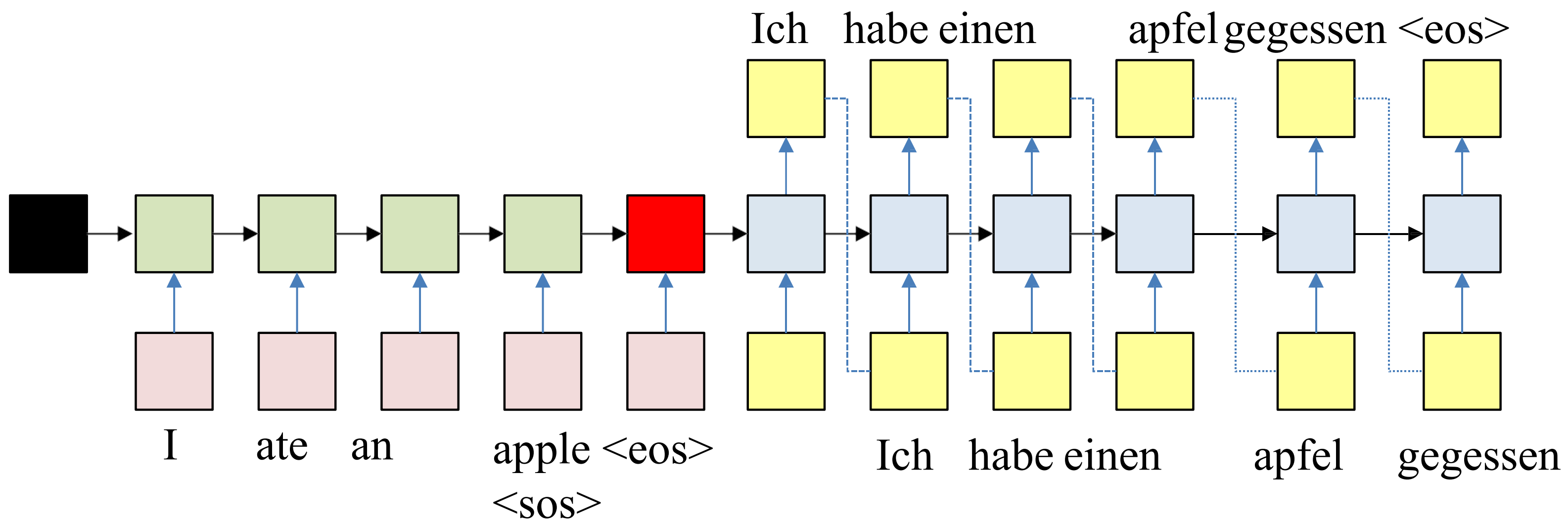
The “simple” translation model



- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

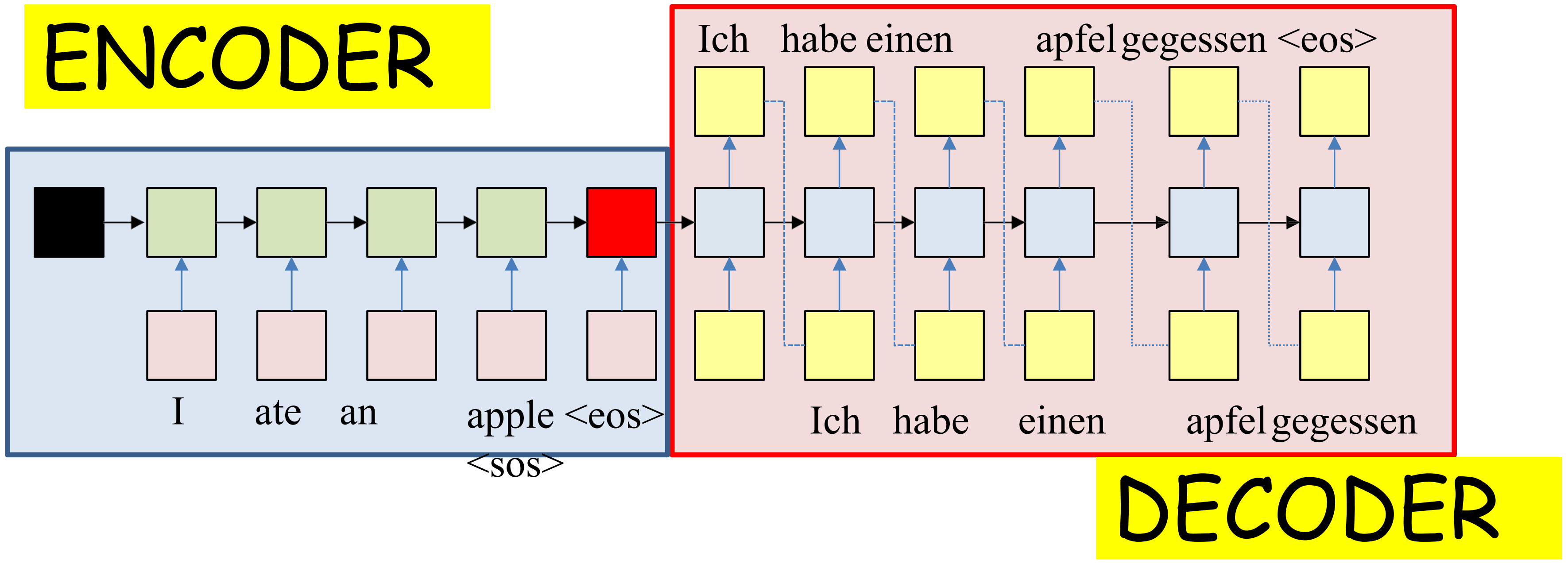
The “simple” translation model



- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

The "simple" translation model


ENCODER
Ich habe einen apfel gegessen <eos>
I ate an apple <eos>
<sos>
Ich habe einen apfel gegessen
DECODER

- The recurrent structure that extracts the hidden representation from the input sequence is the *encoder*

- The recurrent structure that utilizes this representation to produce the output sequence is the *decoder*

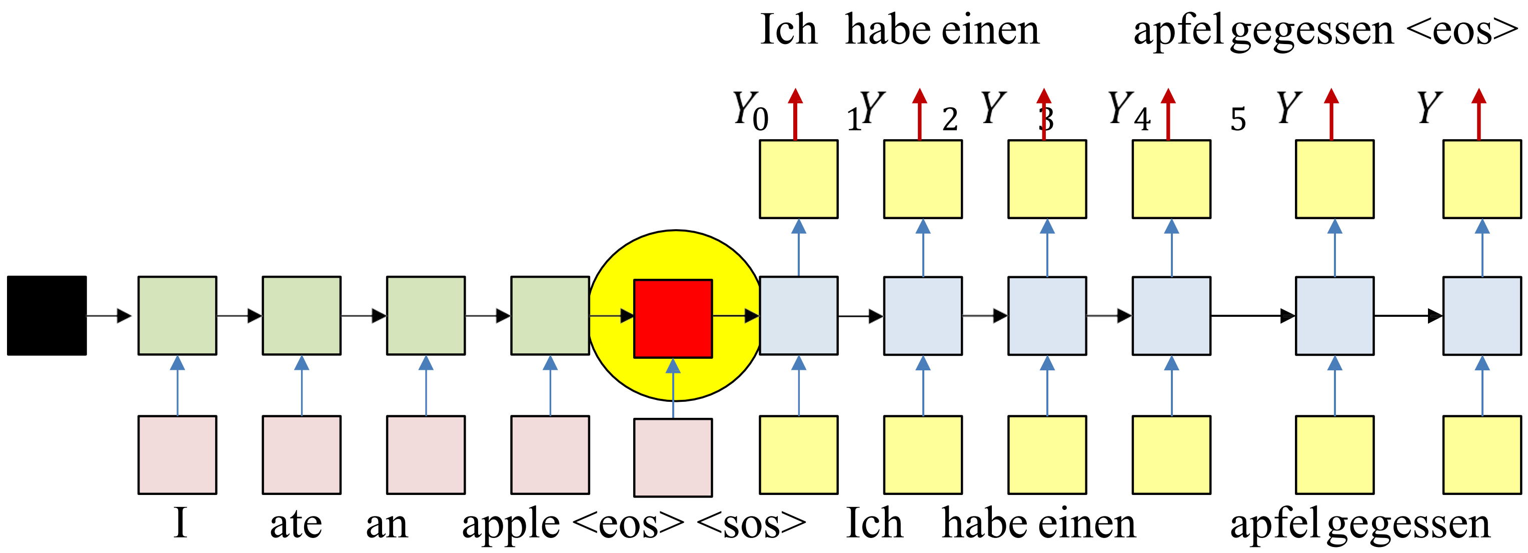
The Bottleneck Problem

Fixed-length context vector = information bottleneck

- Encoder must **compress entire input sequence** into a single vector
- Longer or more complex inputs $\rightarrow$ information loss
- Decoder relies solely on that vector to produce outputs

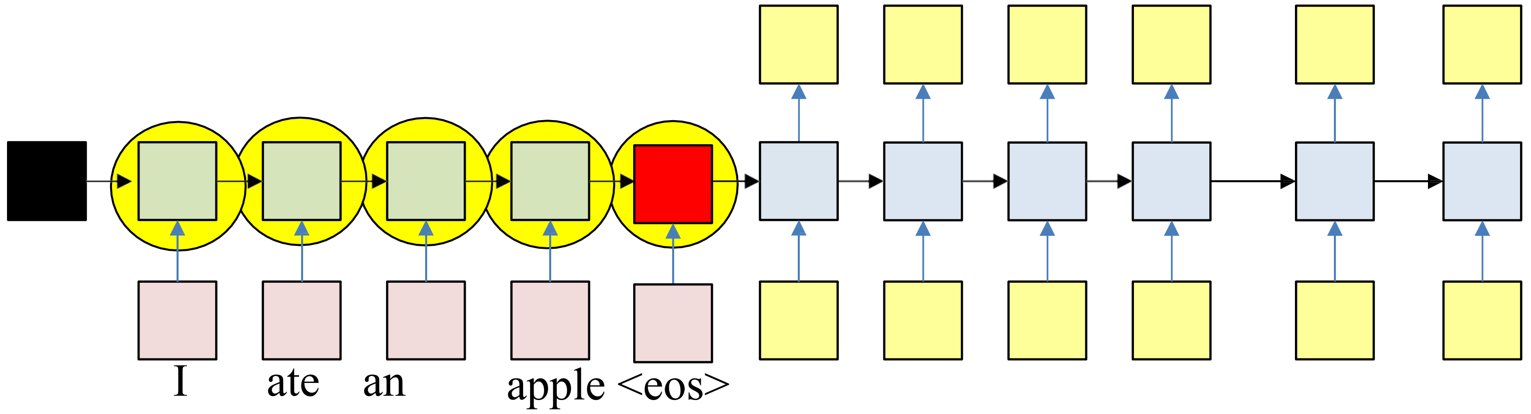
Leads to poor performance on long sentences or tasks requiring high context awareness

A problem with this framework



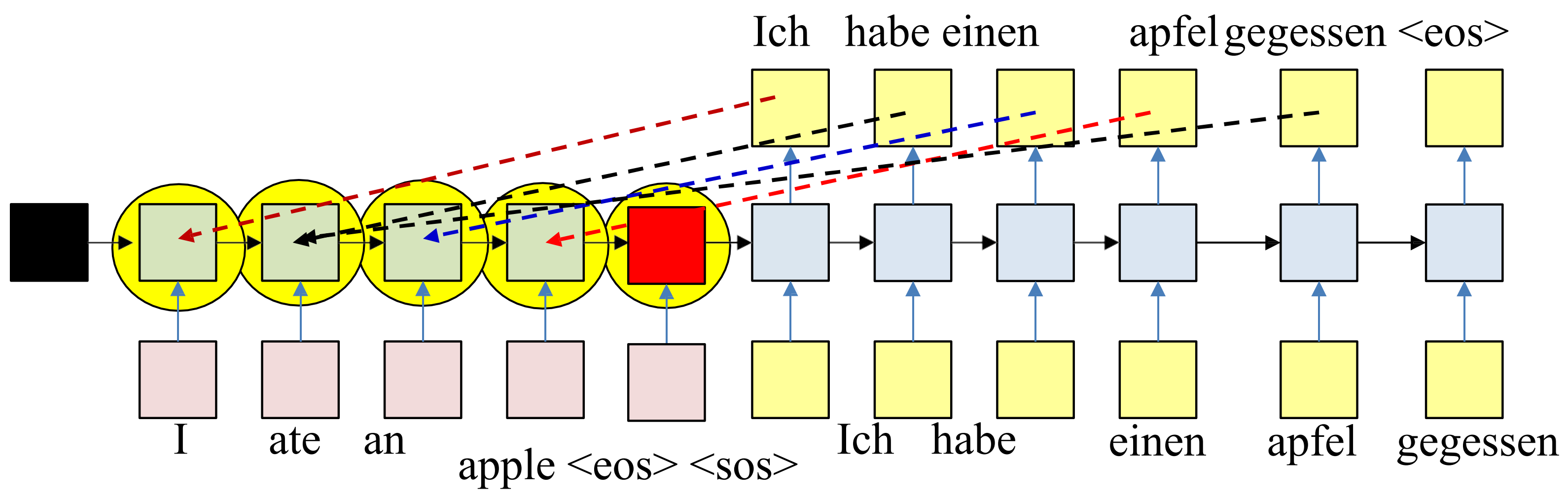
- *All* the information about the input sequence is embedded into a *single* vector
  - The "hidden" node layer at the end of the input sequence
  - This one node is "overloaded" with information
    - Particularly if the input is long

A problem with this framework



- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder

A problem with this framework



- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder
- *Every* output is related to the input directly
  - Not sufficient to have the encoder hidden state to *only* the initial state of the decoder
  - Misses the direct relation of the outputs to the inputs

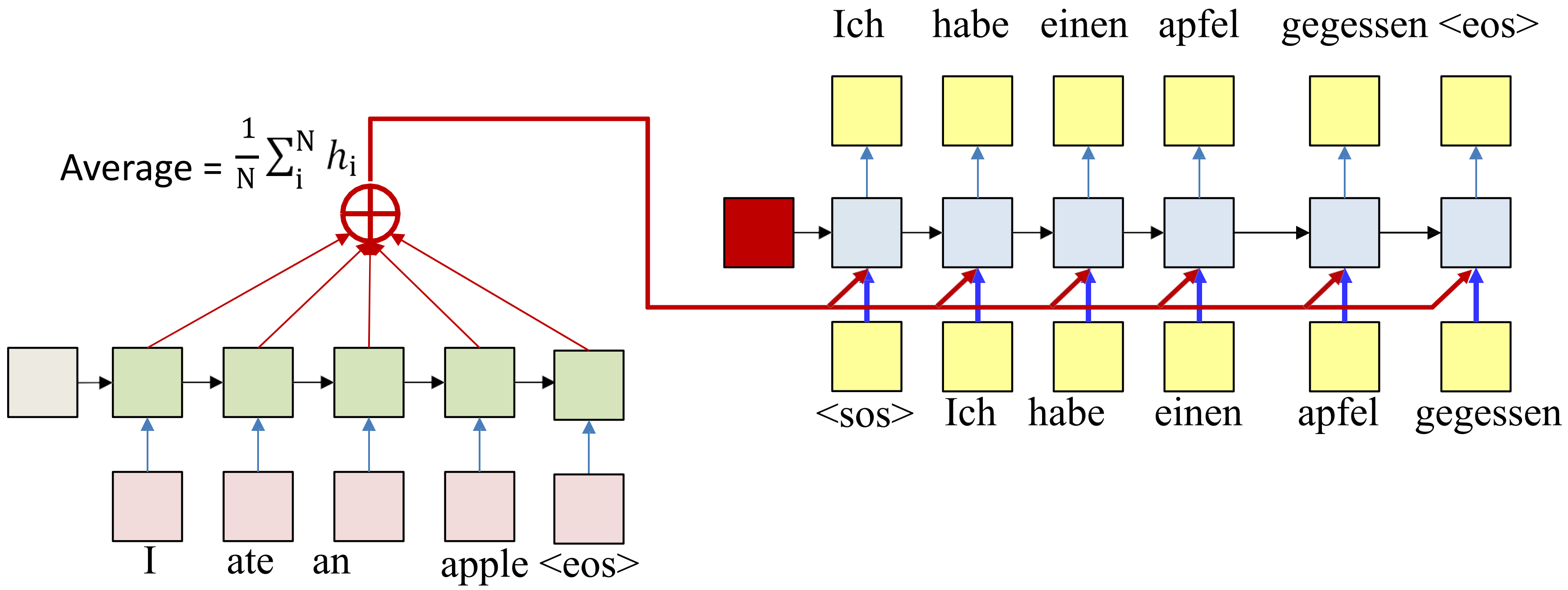
Realization: We Need More Context!

Decoder should have access to all encoder states, not just the final one.

This inspired the development of the **attention mechanism**.

Instead of passing only the final state, allow the decoder to *"look back"* at **all input positions**.

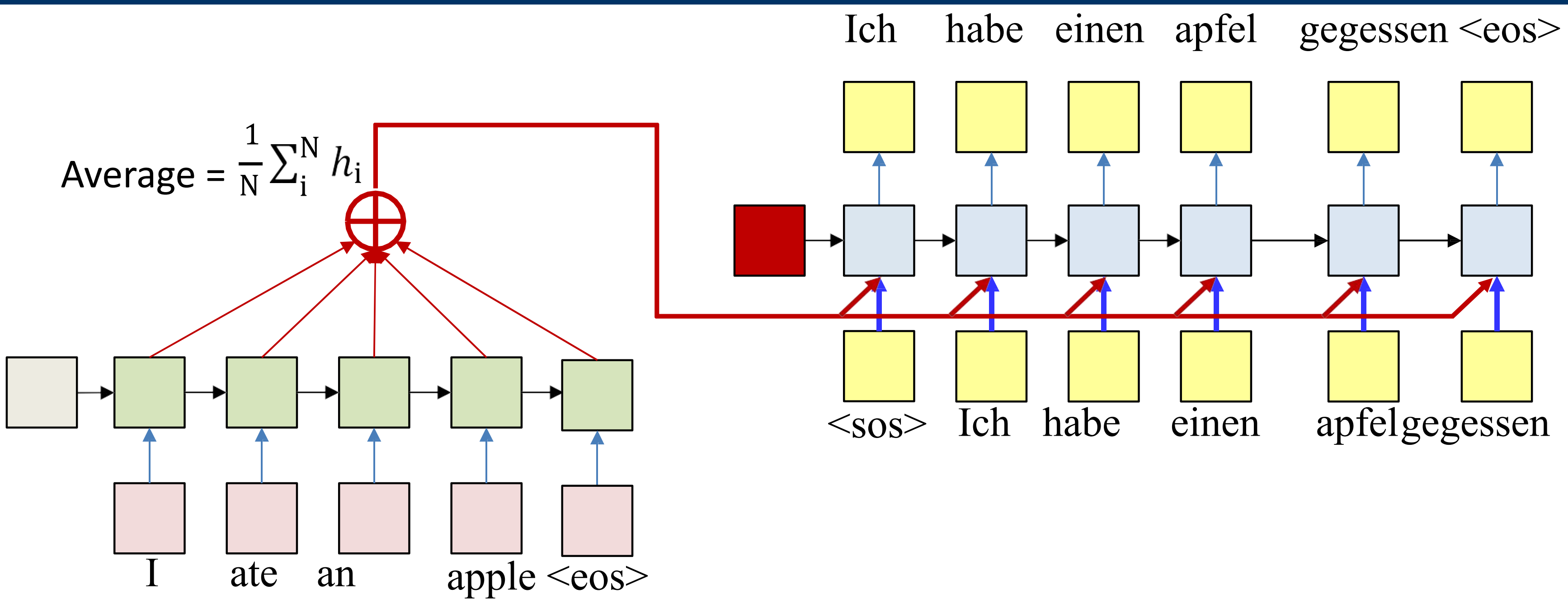
Using all input hidden states



Average = $\frac{1}{N}\sum_i^N h_i$
Ich habe einen apfel gegessen <eos>
<sos> Ich habe einen apfel gegessen
I ate an apple <eos>

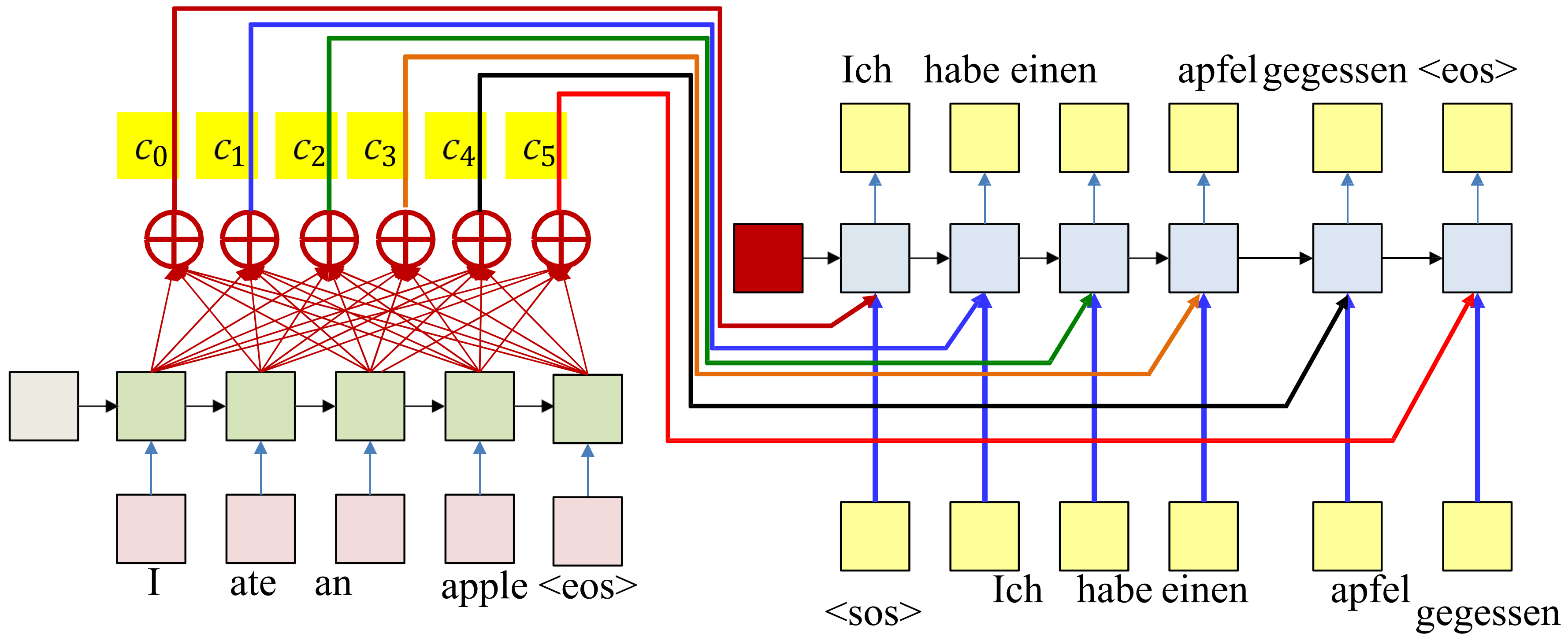
- Simple solution: Compute the average of all encoder hidden states
- Input this average to every stage of the decoder
- The initial decoder hidden state is now separate from the encoder
  – And may be a learnable parameter

Using all input hidden states



- **Problem:** The average applies the same weight to every input
- It supplies the same average to every output word
- In practice, different outputs may be related to different inputs
  - E.g. "Ich" is most related to "I", and "habe" and "gegessen" are both most related to "ate"

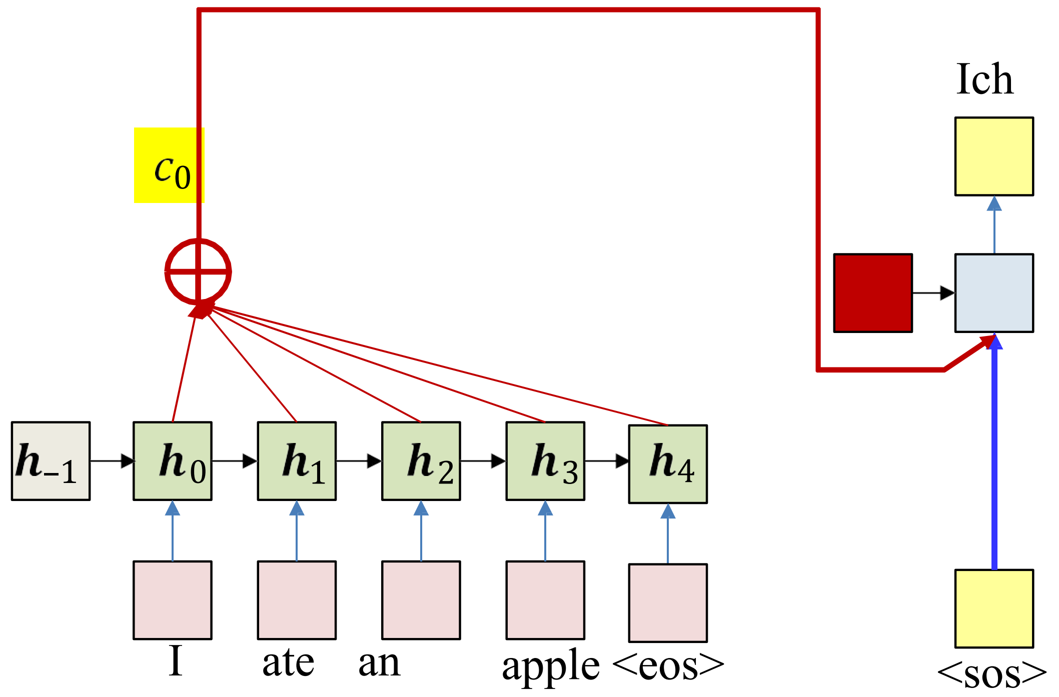
Using all input hidden states


c_0
c_1
c_2
c_3
c_4
c_5
Ich habe einen apfel gegessen <eos>
I ate an apple <eos>
<sos> Ich habe einen apfel gegessen

- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

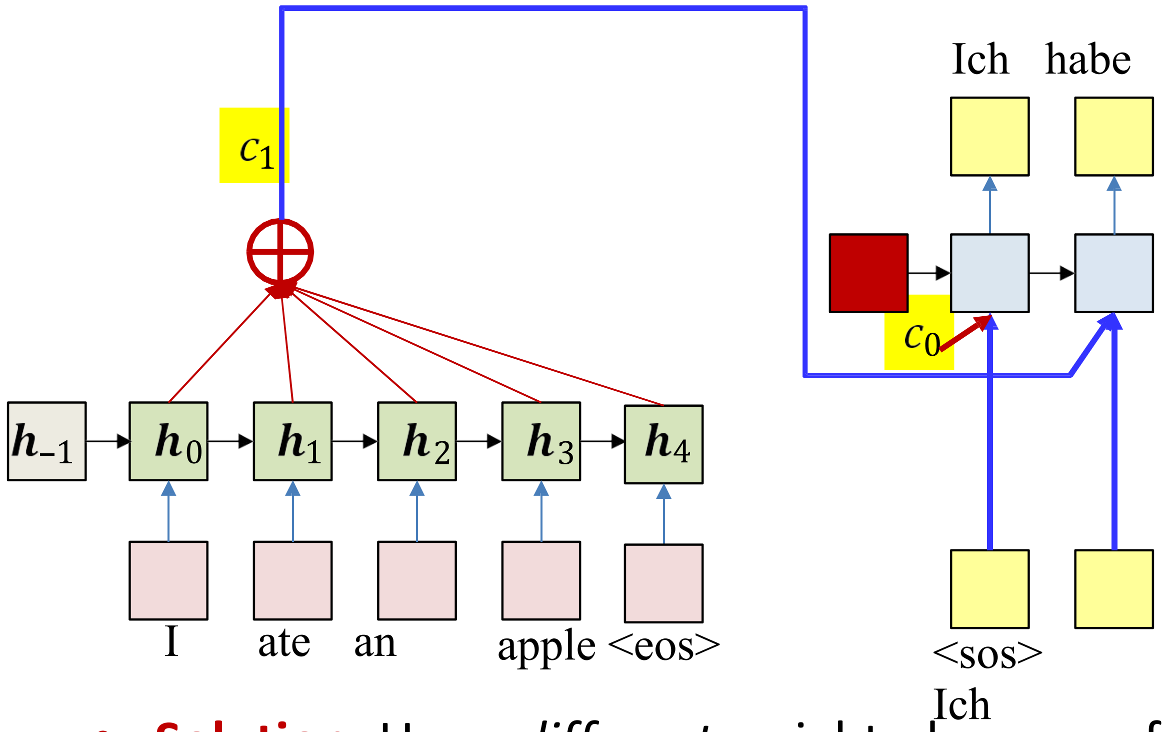
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_0 = \frac{1}{N}\sum_{i}^{N} w_i(0)h_i$$

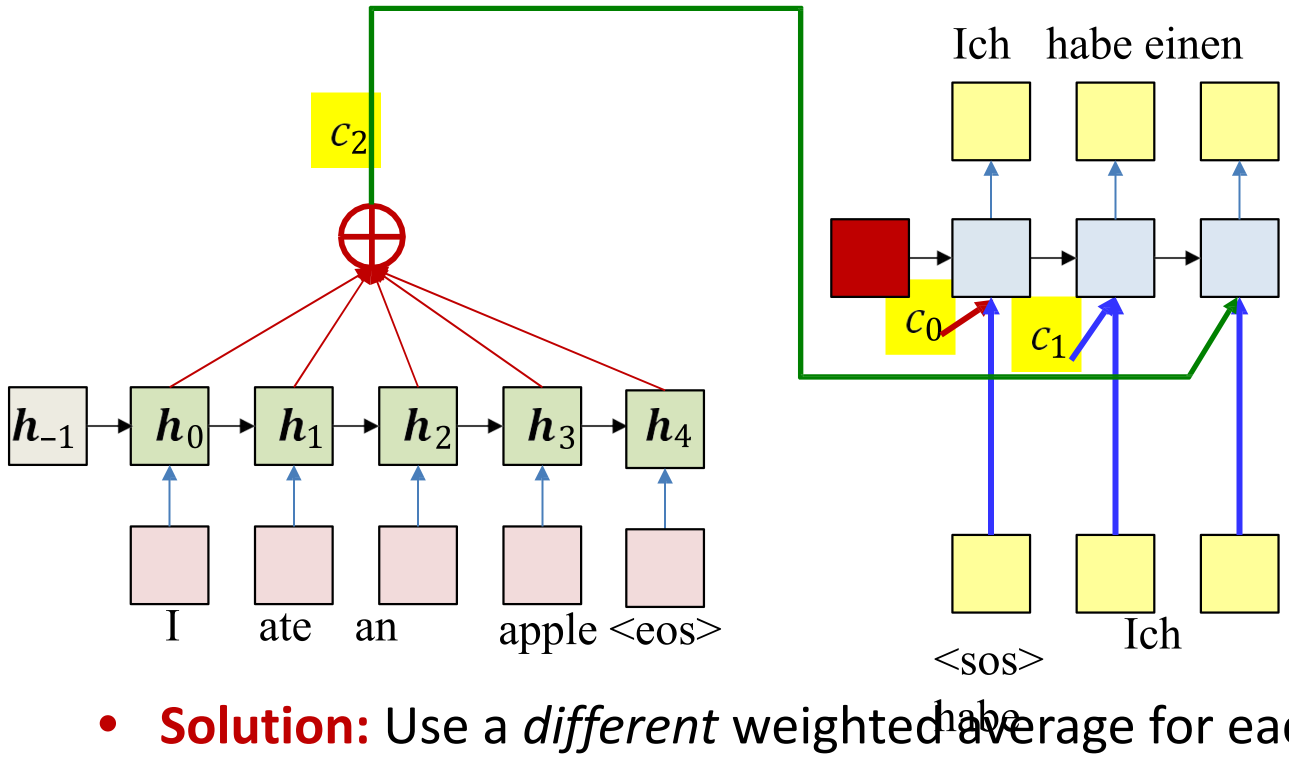
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_1 = \frac{1}{N}\sum_{i}^{N} w_i(1)h_i$$

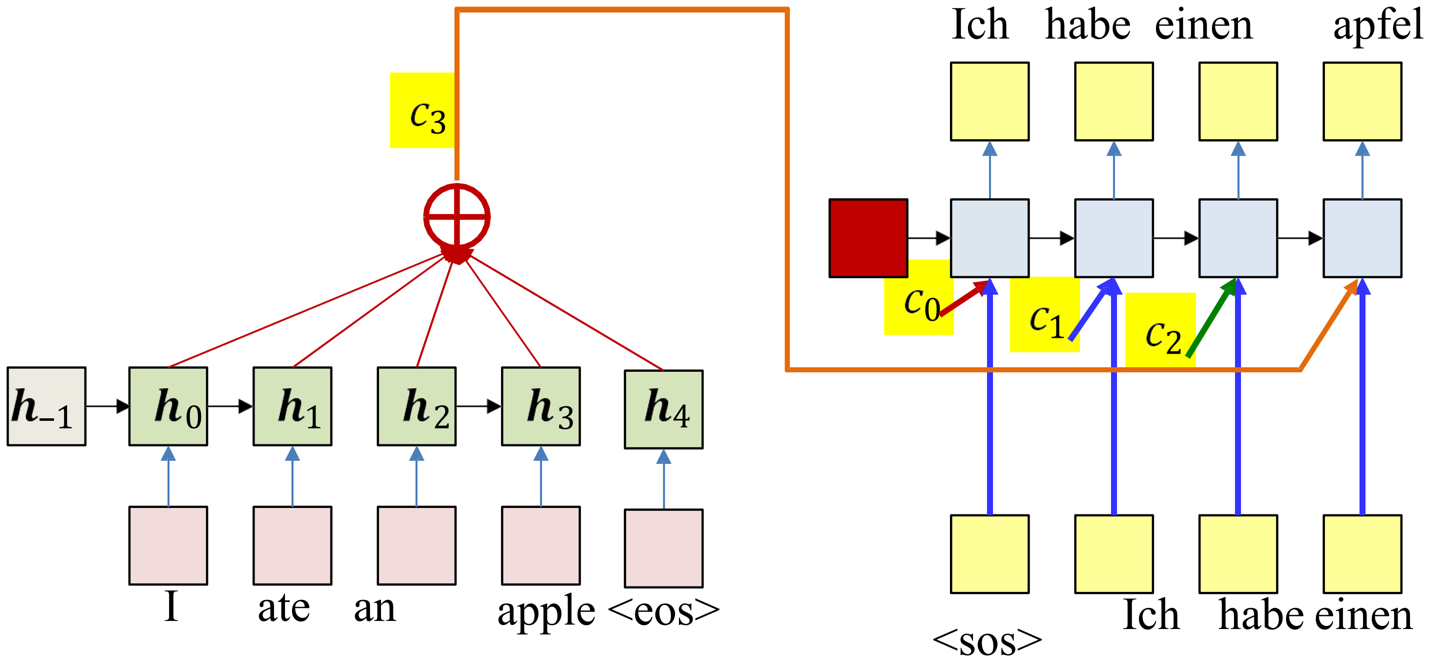
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_2 = \frac{1}{N}\sum_{i}^{N} w_i(2)h_i$$

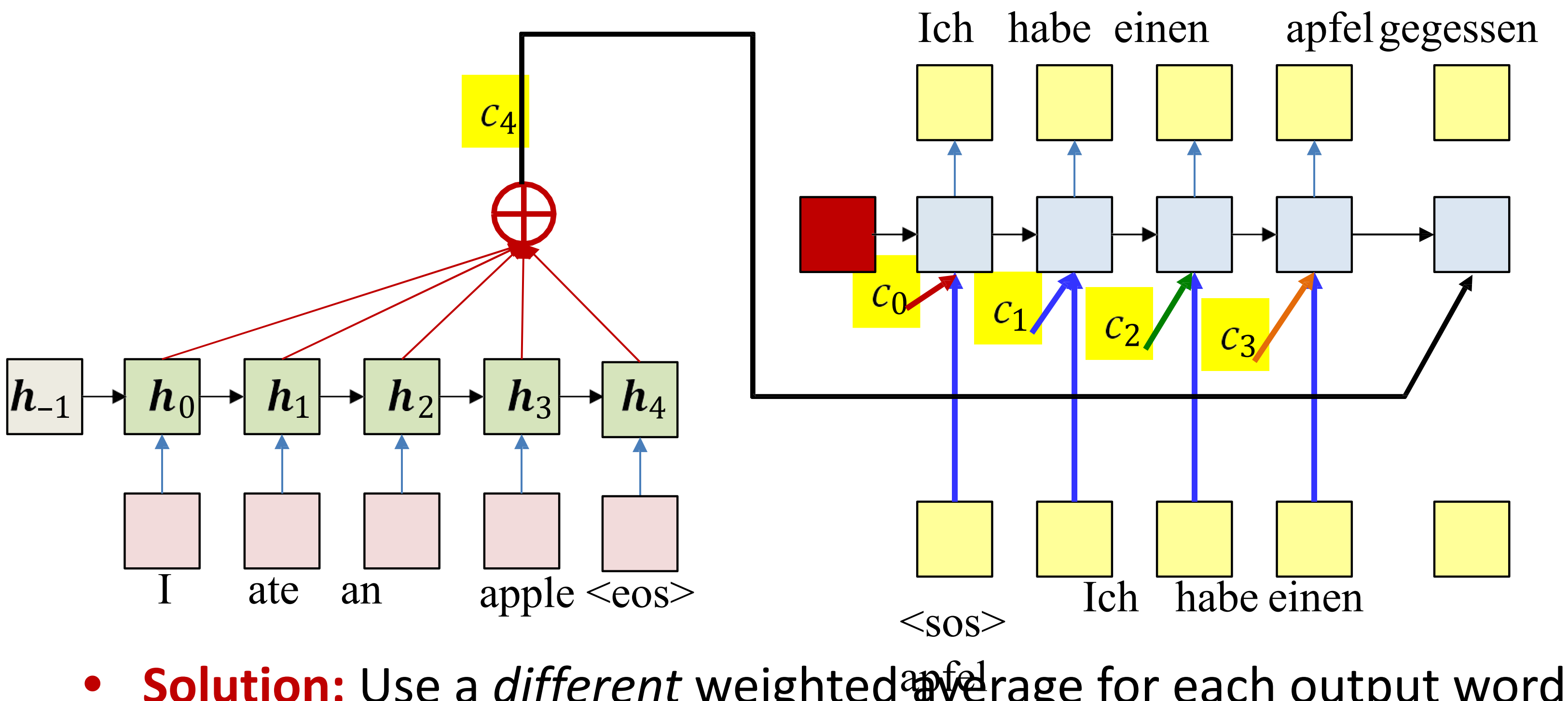
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_3 = \frac{1}{N} \sum_{i}^{N} w_i(3) h_i$$

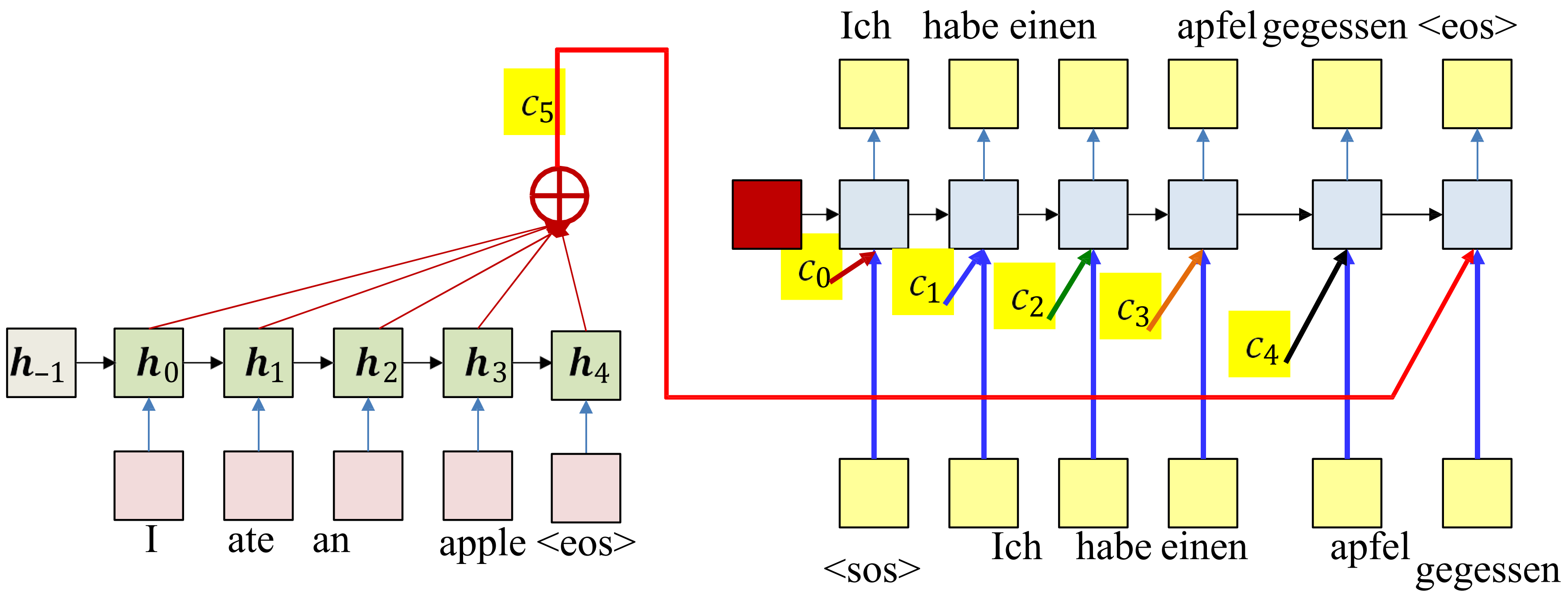
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_4 = \frac{1}{N}\sum_{i}^{N} w_i(4)h_i$$

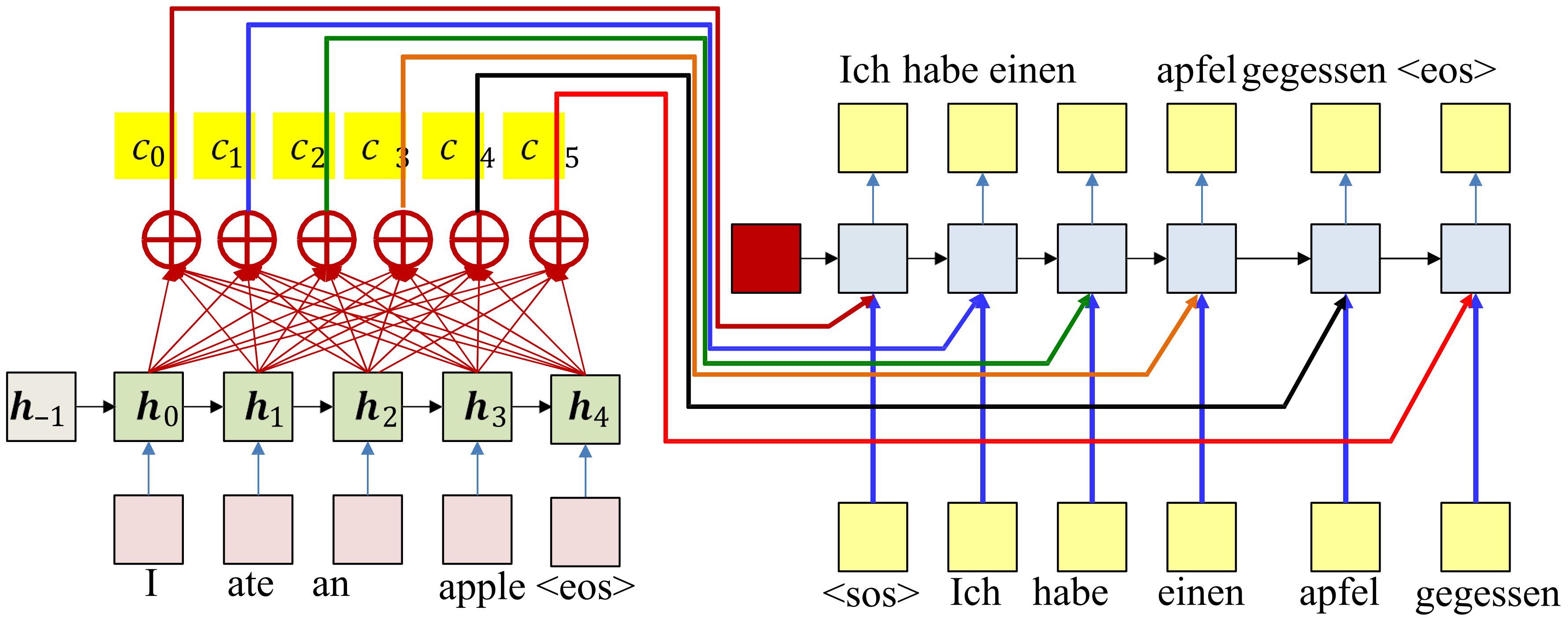
Using all input hidden states



- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_5 = \frac{1}{N}\sum_{i}^{N} w_i(5)h_i$$

Using all input hidden states


Ich habe einen apfel gegessen <eos>
c_0 c_1 c_2 c_3 c_4 c_5
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>
<sos> Ich habe einen apfel gegessen

$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

- This solution will work if the weights w_{ki} can somehow be made to “focus” on the right input word
  - E.g., when predicting the word “apfel”, $w_3(4)$, the weight for “apple” must be high while the rest must be low
- How do we generate such weights??

Introduction to Attention Mechanism

Core Idea: Let the decoder **focus on different parts of the input** sequence at each step of decoding.

- At each decoding step, compute a **weighted sum** over all encoder hidden states.
- Weights reflect **relevance** of each input word to the current output word.

“Soft search” over inputs $\rightarrow$ more context-awareness.

Mathematical Formulation of Attention

1. **Alignment Score:**

$$e_{t,s} = \text{score}(h_t^{\text{dec}}, h_s^{\text{enc}})$$

2. **Attention Weights (Softmax):**

$$\alpha_{t,s} = \frac{\exp(e_{t,s})}{\sum_{s'} \exp(e_{t,s'})}$$

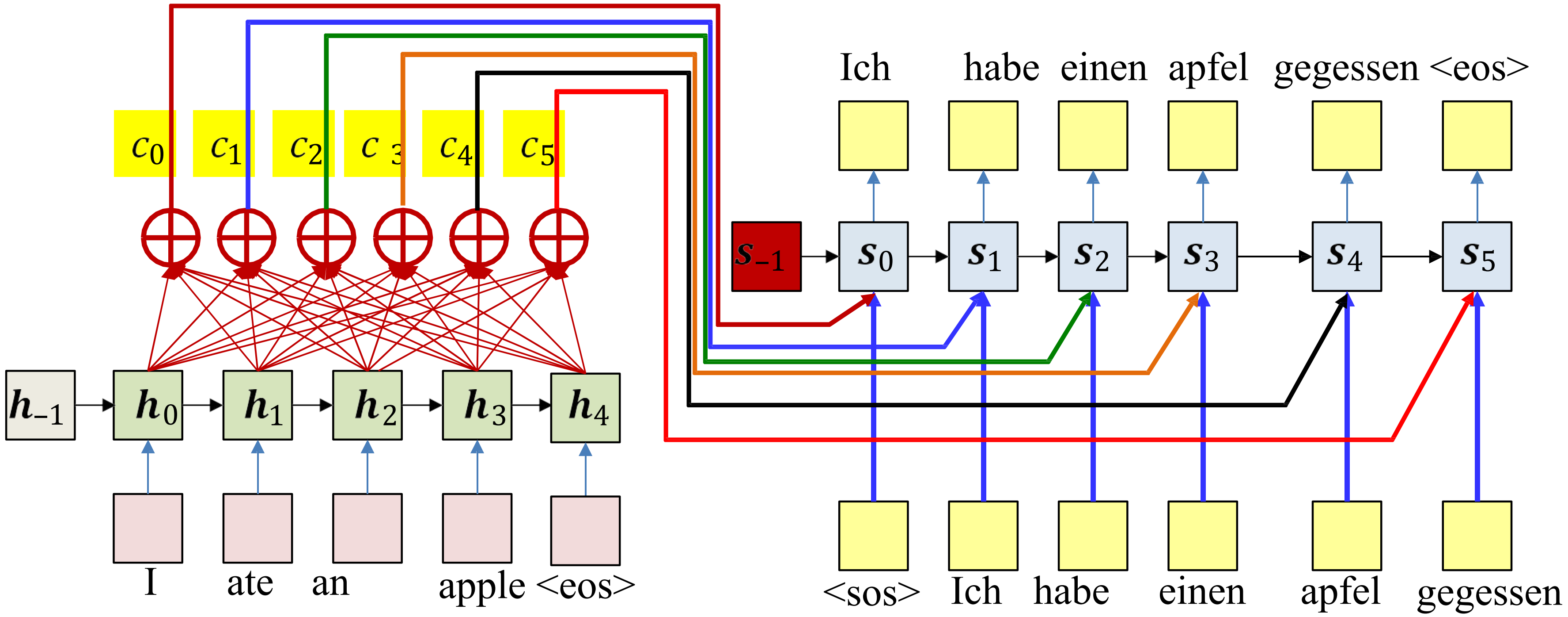
3. **Context Vector:**

$$c_t = \sum_s \alpha_{t,s} h_s^{\text{enc}}$$

4. **Decoder Input:**

$$y_t = \text{Decoder}(y_{t-1}, h_{t-1}^{\text{dec}}, c_t)$$

Attention Models

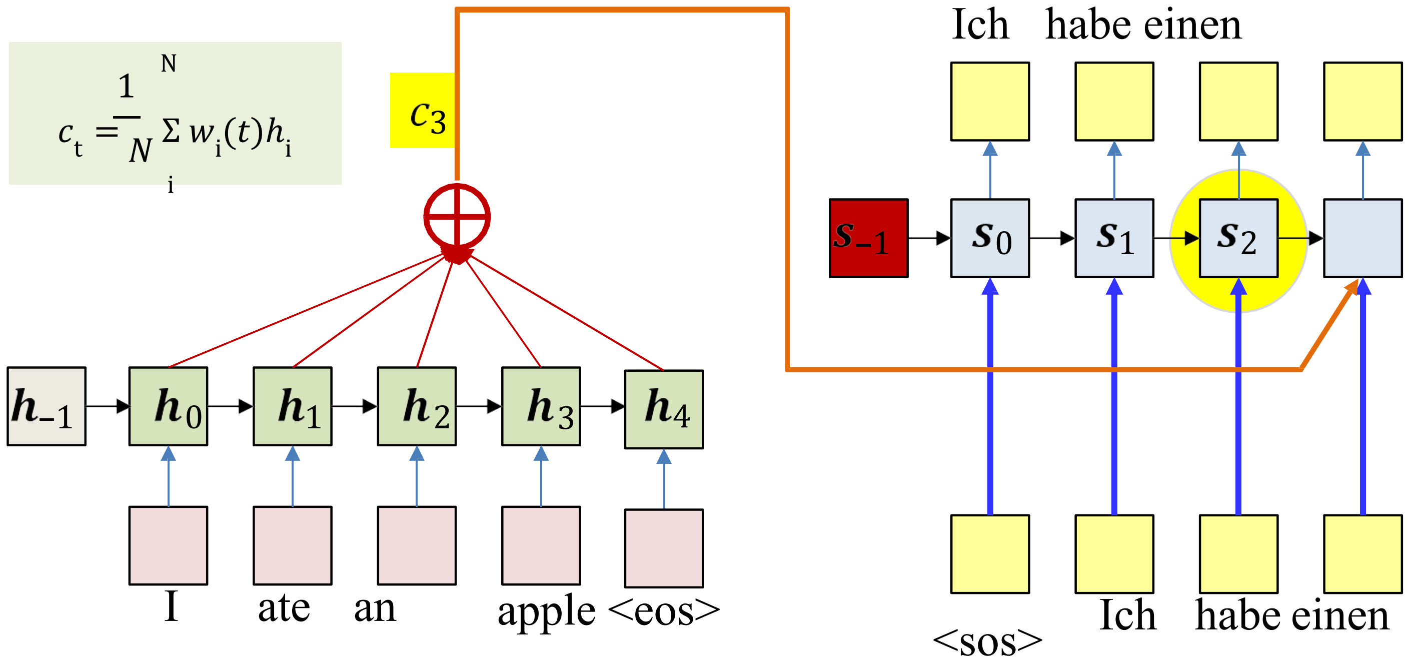


$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

- **Attention weights:** The weights $w_i(t)$ are dynamically computed as functions of decoder state
  - Expectation: if the model is well-trained, this will automatically "highlight" the correct input
- But how are these computed?

Attention weights at time

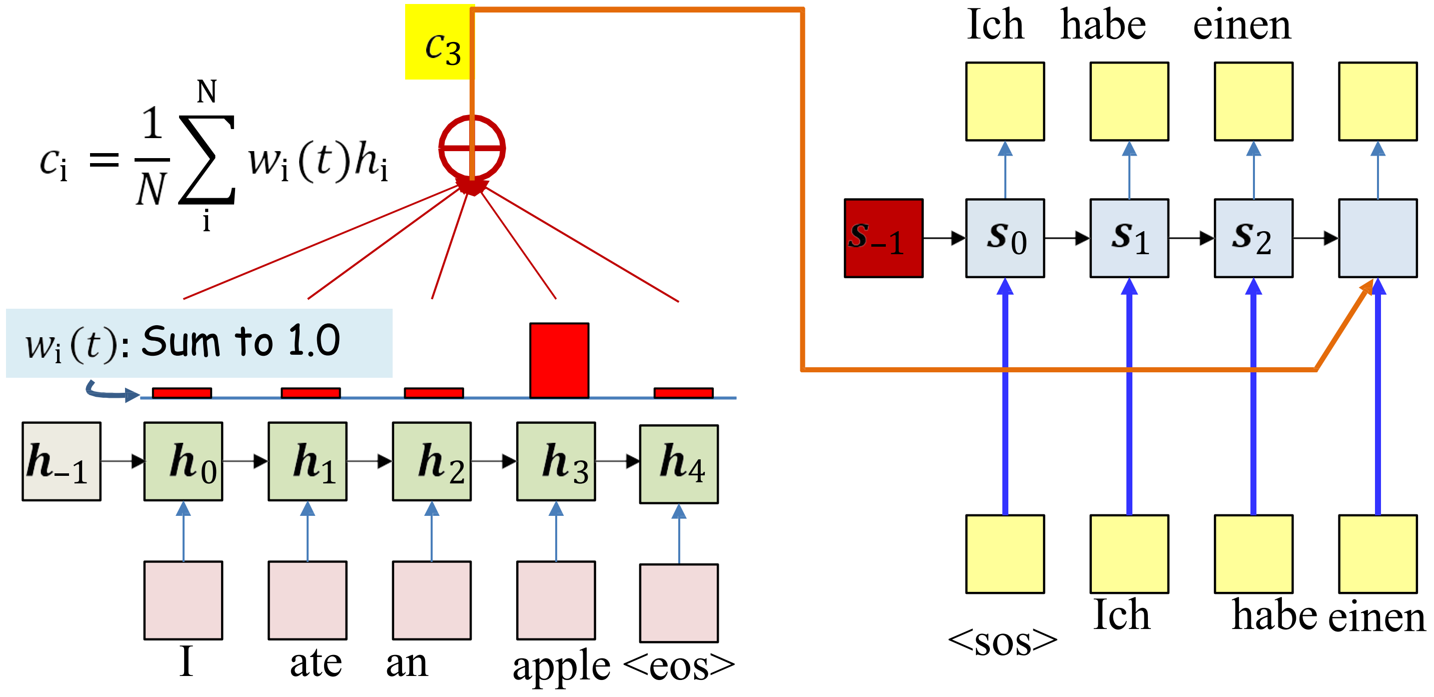
$$c_t = \frac{1}{N}\sum_i^N w_i(t) h_i$$



- The "attention" weights $w_i(t)$ at time t must be computed from available information at time t
- The primary information is s_{t-1} (the state at time time $t-1$)
  - Also, the input word at time t, but generally not used for simplicity

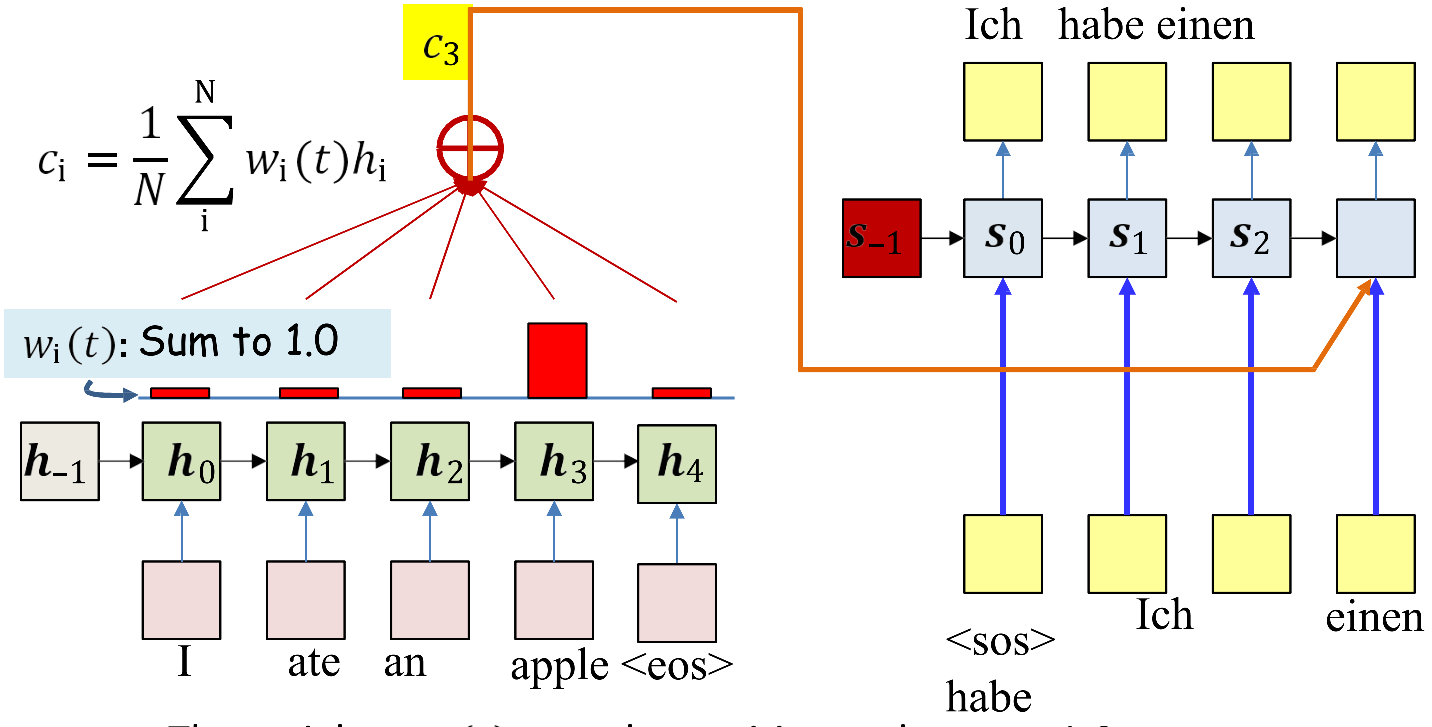
$$w_i(t) = a(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

Requirement on attention weights



- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

Requirement on attention weights



- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

- Solution: A two step weight computation
  - First compute *raw* weights (which could be +ve or –ve)
  - Then softmax them to convert them to a distribution

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

Quiz

أكاديمية كاوست
KAUST ACADEMY

The attention framework computes a different “context” vector at each output step (T/F)

- True
- False

The context vector is chosen as the hidden (encoder) representation of the input word that is assigned the highest attention weight (T/F)

- True
- False

The attention weight to any input word is a function of the hidden encoder representation of the word and the most recent decoder state (T/F)

- True
- False

Quiz

The attention framework computes a different "context" vector at each output step (T/F)

- **True**
- False

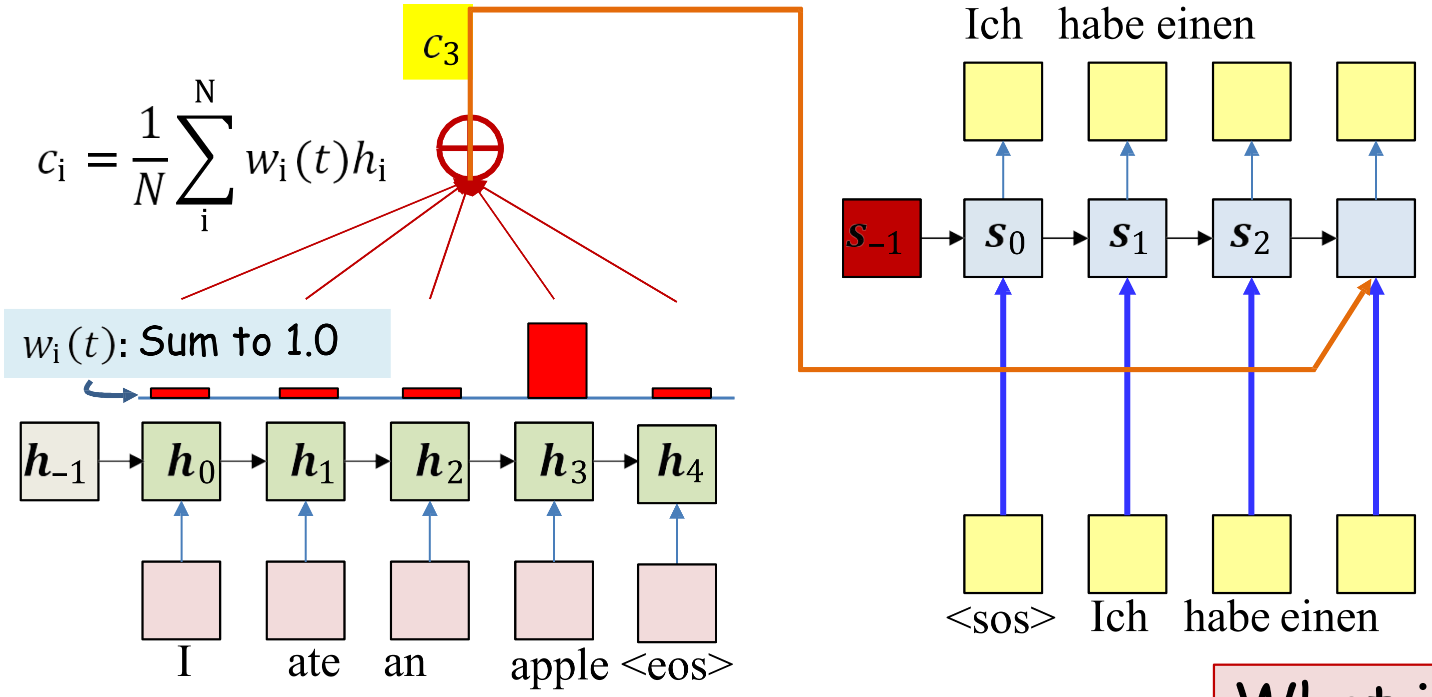
The context vector is chosen as the hidden (encoder) representation of the input word that is assigned the highest attention weight (T/F)

- True
- **False**

The attention weight to any input word is a function of the hidden encoder representation of the word and the most recent decoder state (T/F)

- **True**
- False

Requirement on attention weights



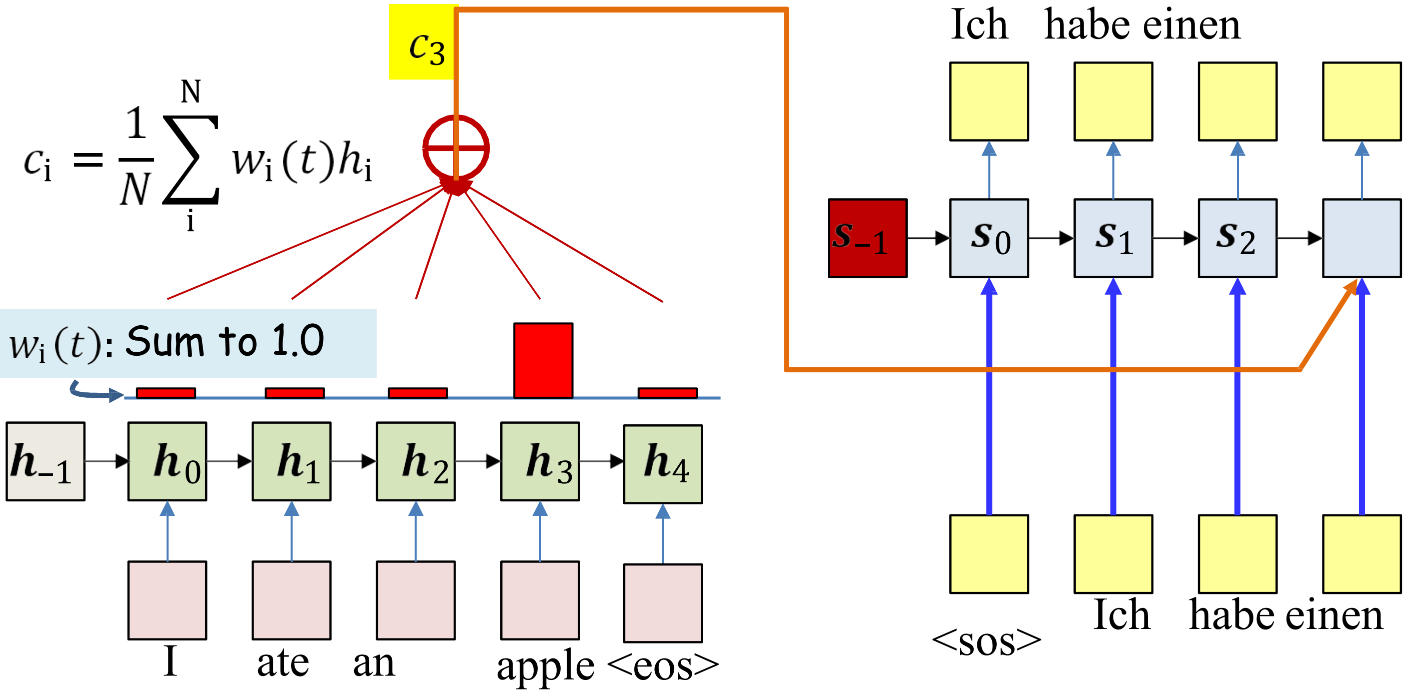
What is this function?

- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere
- Solution: A two step weight computation
  - First compute *raw* weights (which could be +ve or –ve)
  - Then softmax them to convert them to a distribution

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

Attention weights



c_3
$c_i = \frac{1}{N}\sum_i^N w_i(t) h_i$
$w_i(t)$: Sum to 1.0
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>
Ich habe einen
s_{-1} s_0 s_1 s_2
<sos> Ich habe einen

- Typical options for $g()$ (**variables in red must learned**)

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{s}_{t-1}$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{t-1}$$

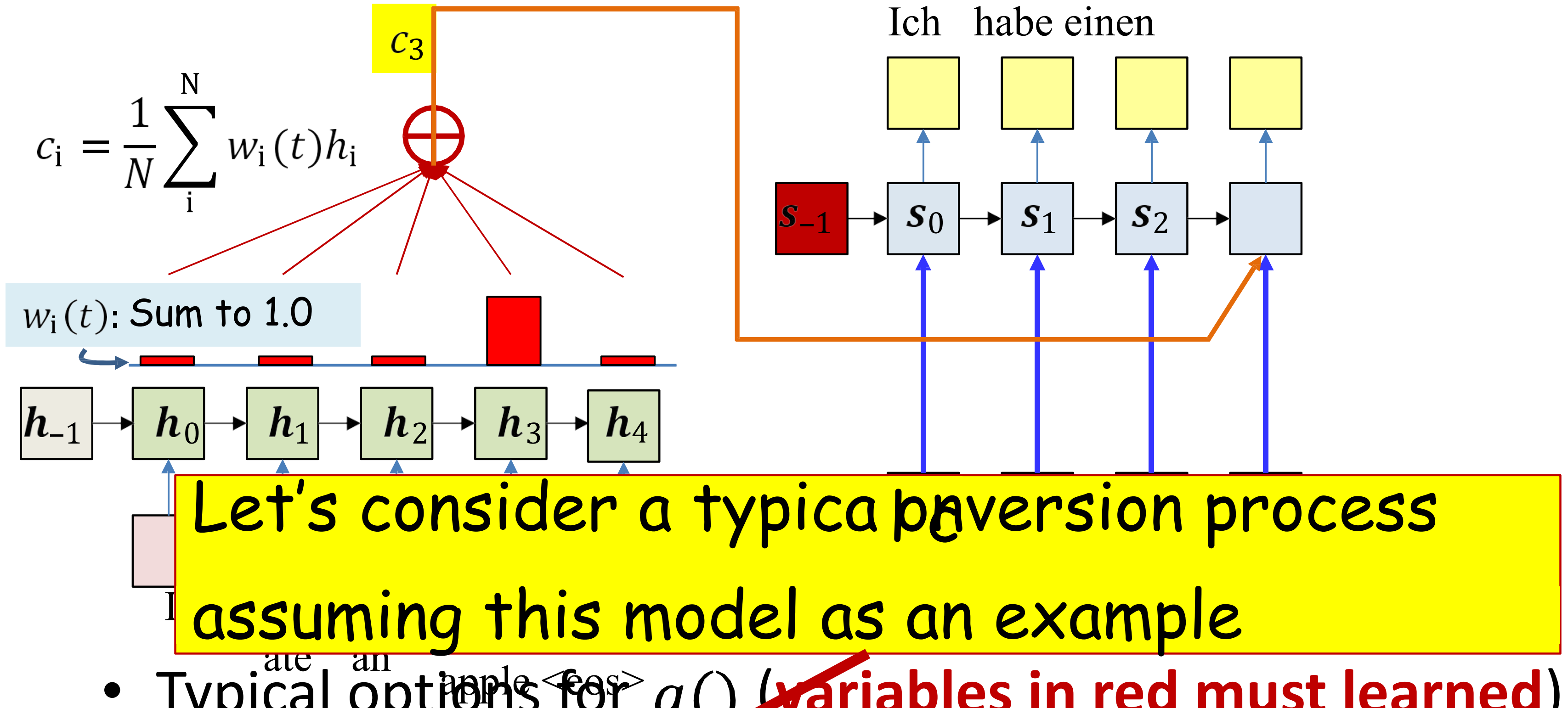
$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{v}_g^T \boldsymbol{tanh}\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{s}_{t-1} \end{bmatrix}\right)$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$$

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

Attention weights



c_3
$c_\mathrm{i} = \frac{1}{N}\sum_\mathrm{i}^{N} w_\mathrm{i}(t)h_\mathrm{i}$
$w_\mathrm{i}(t)$: Sum to 1.0
h_{-1}
h_0
h_1
h_2
h_3
h_4
Ich habe einen
s_{-1}
s_0
s_1
s_2
ate
an
apple
<eos>

Let's consider a typical conversion process assuming this model as an example

- Typical options for $g()$ (**variables in red must learned**)

$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{s}_{t-1}$

$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{t-1}$

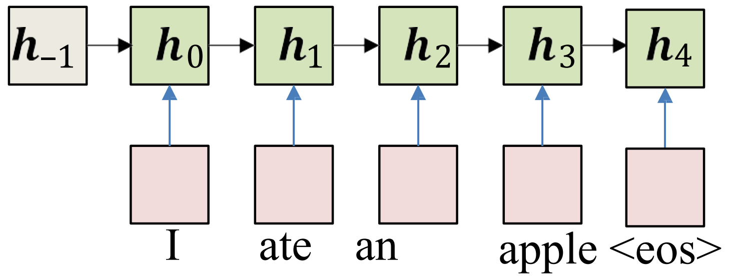
$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{v}_g^T \boldsymbol{tanh}\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{s}_{t-1} \end{bmatrix}\right)$

$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$

$e_\mathrm{i}(t) = g(\boldsymbol{h}_\mathrm{i}, \boldsymbol{s}_{\mathrm{t}-1})$

$w_\mathrm{i}(t) = \frac{\exp(e_\mathrm{i}(t))}{\sum_\mathrm{j} \exp(e_\mathrm{j}(t))}$

Converting an input: Inference

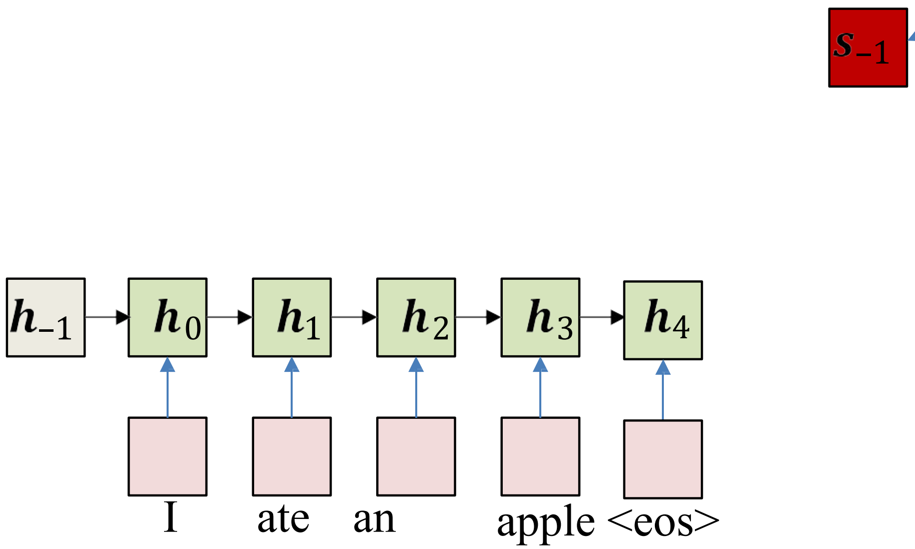


- Pass the input through the encoder to produce hidden representations $\boldsymbol{h}_i$

Converting an input: Inference

This may be

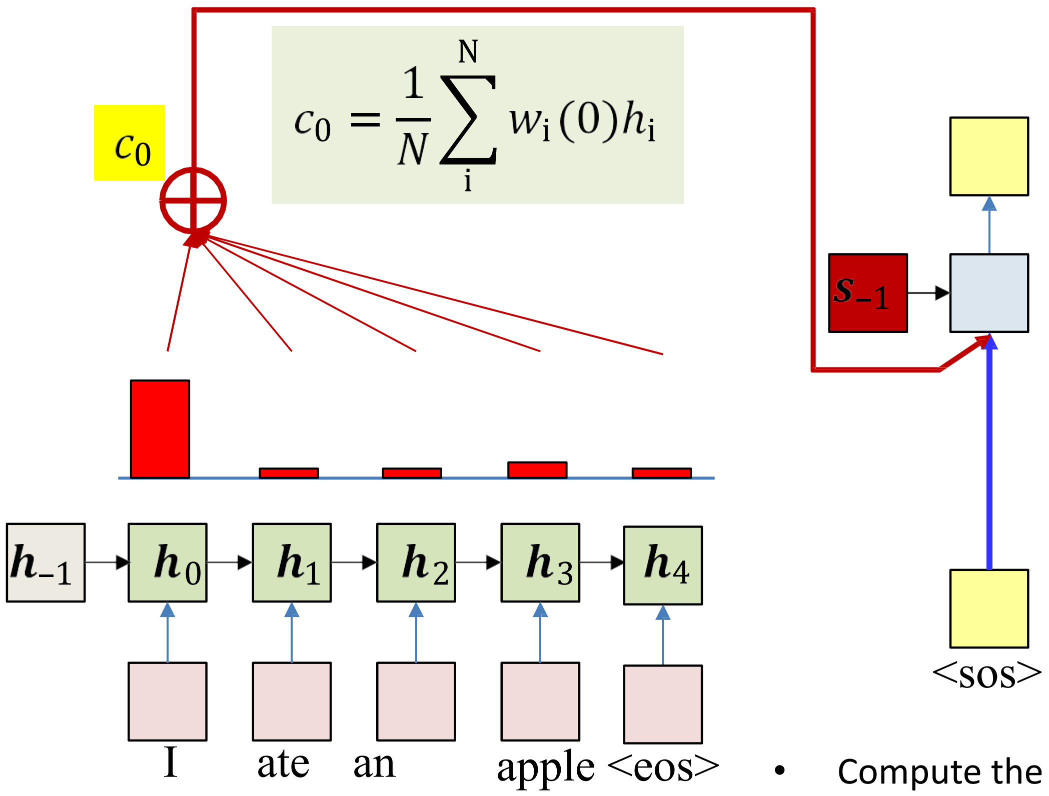
- a learned parameter, or
- Or just set to some fixed value, e.g. a vector of 1s or 0s, or
- Or the average of all the encoder embeddings: $mean(h_0, \dots, h_4)$
- Or $W_{\text{init}}\ mean(h_0, \dots, h_4)$ where W_{init} is a learned parameter



- Pass the input through the encoder to produce hidden representations $\boldsymbol{h}_i$

Converting an input: Inference

$$c_0 = \frac{1}{N}\sum_{i}^{N} w_i(0) h_i$$



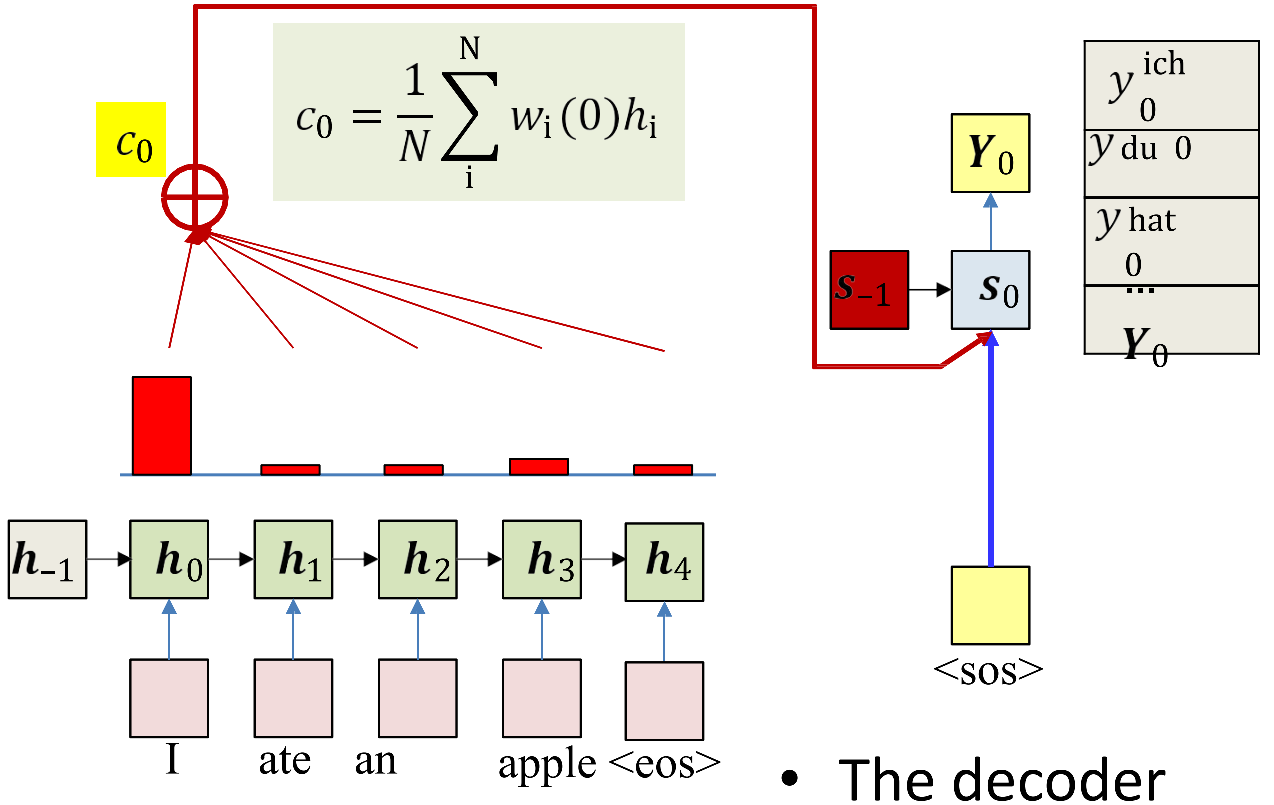
$$g(h_i, s_{-1}) = h_i^T W_g s_{-1}$$

$$e_i(0) = g(h_i, s_{-1})$$

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$

- Compute the attention weights $w_i(0)$ for the first output using s_{-1}
  - Will be a distribution over the input words
- Compute "context" c_0
  - Weighted sum of input word hidden states
- Input c_0 and <sos> to the decoder at time 0
  - <sos> because we are starting a new sequence
  - In practice we will enter the *embedding* of <sos>

Converting an input: Inference



$$c_0 = \frac{1}{N}\sum_{i}^{N} w_i(0) h_i$$

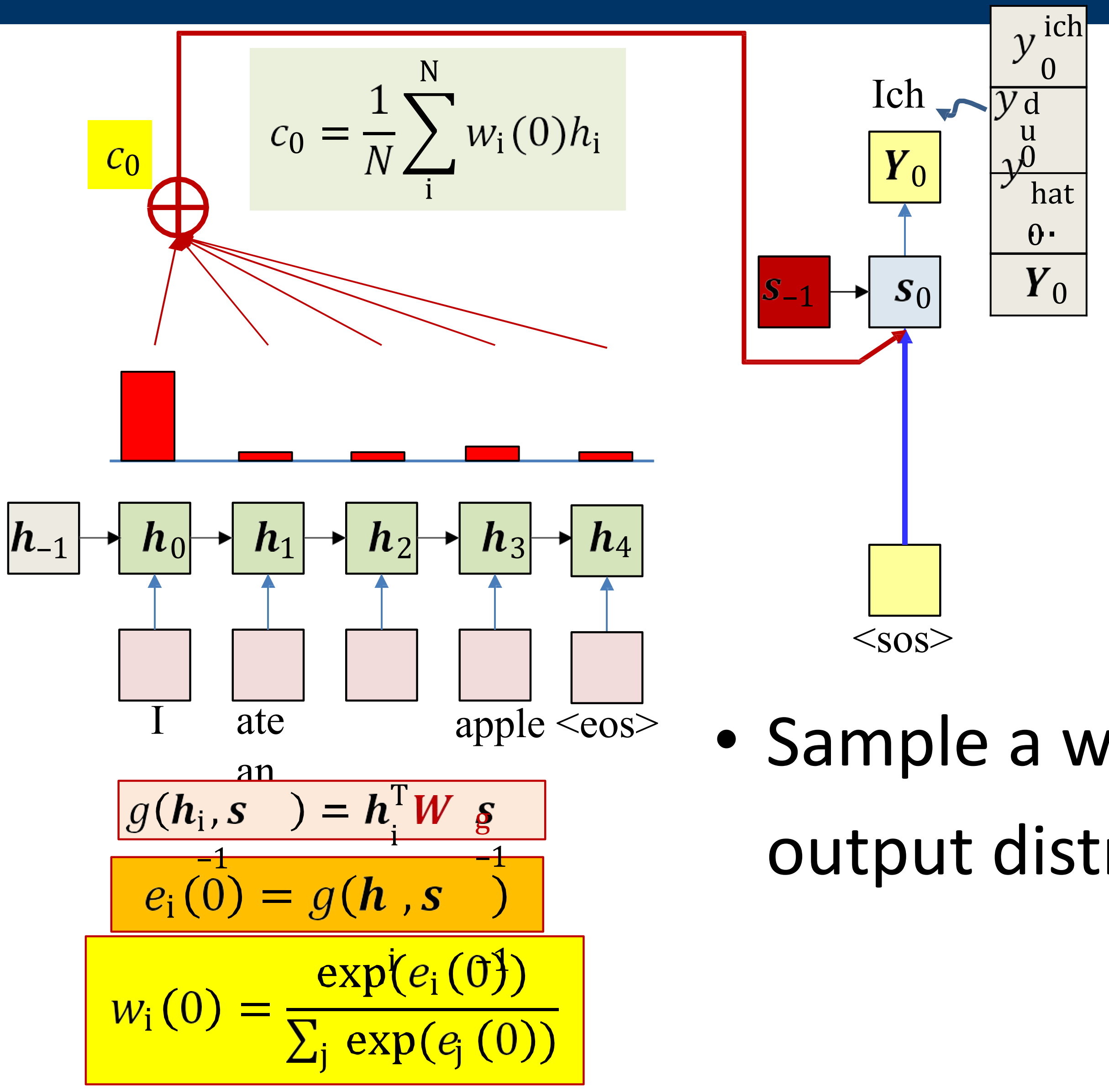
$$g(h_i, s_{-1}) = h_i^T W_g s_{-1}$$

$$e_i(0) = g(h_i, s_{-1})$$

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$

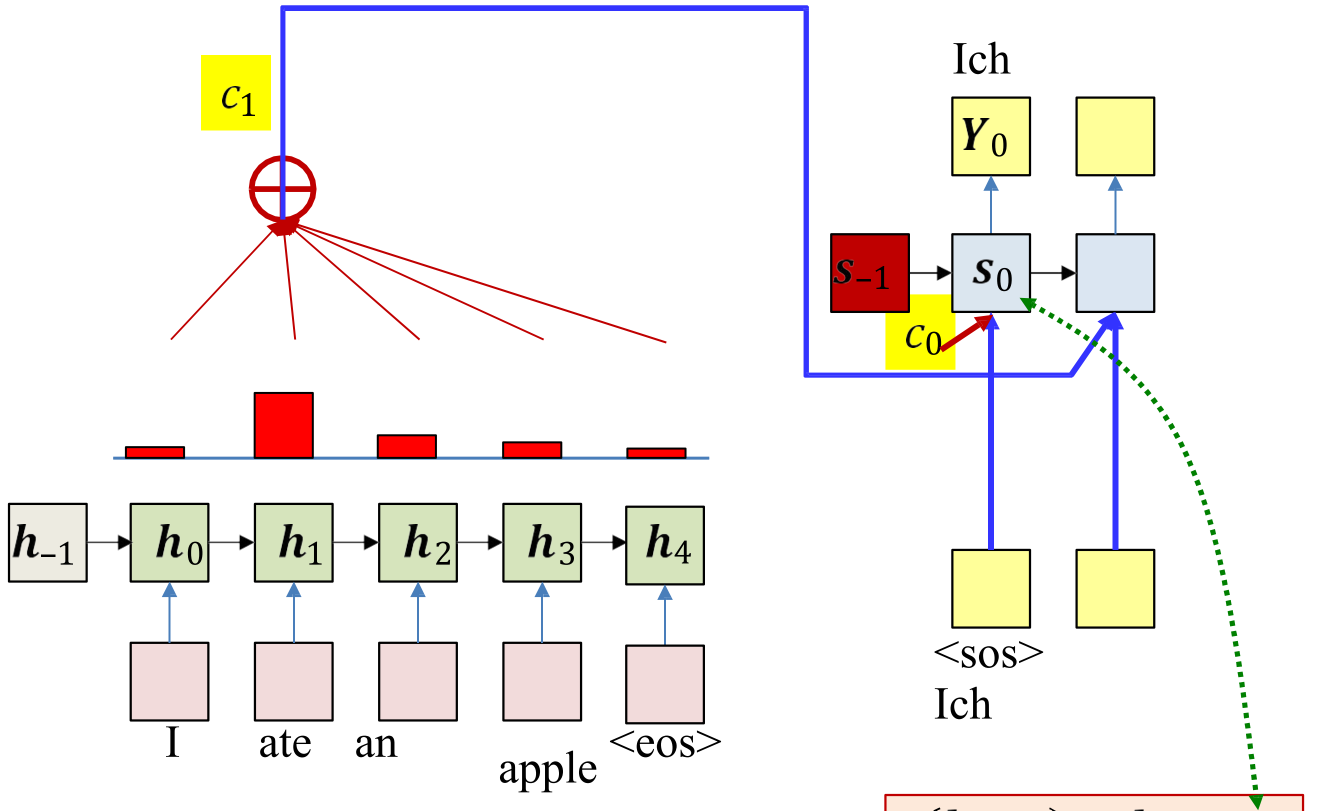
- The decoder computes
  – s_0
  – A probability distribution over the output vocabulary
    - Output of softmax output layer

Converting an input: Inference


$c_0 = \frac{1}{N}\sum_i^N w_i(0)h_i$
c_0
Ich
Y_0
s_{-1}
s_0
<sos>
$\boldsymbol{h}_{-1}$
$\boldsymbol{h}_0$
$\boldsymbol{h}_1$
$\boldsymbol{h}_2$
$\boldsymbol{h}_3$
$\boldsymbol{h}_4$
I
ate
an
apple
<eos>
y_0^{ich}
y_0^{du}
y_0^{hat}
Y_0
$g(\boldsymbol{h}_i, \boldsymbol{s}_{-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{-1}$
$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$
$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$

- Sample a word from the output distribution

Converting an input: Inference



- Compute the attention weights $w_i(1)$ over all inputs for the *second* output using s_0
  - Compute raw weights, followed by softmax
- Compute “context” c_1
  - Weighted sum of input hidden representations
- Input c_1 and first output word to the decoder
  - In practice we enter the *embedding* of the word

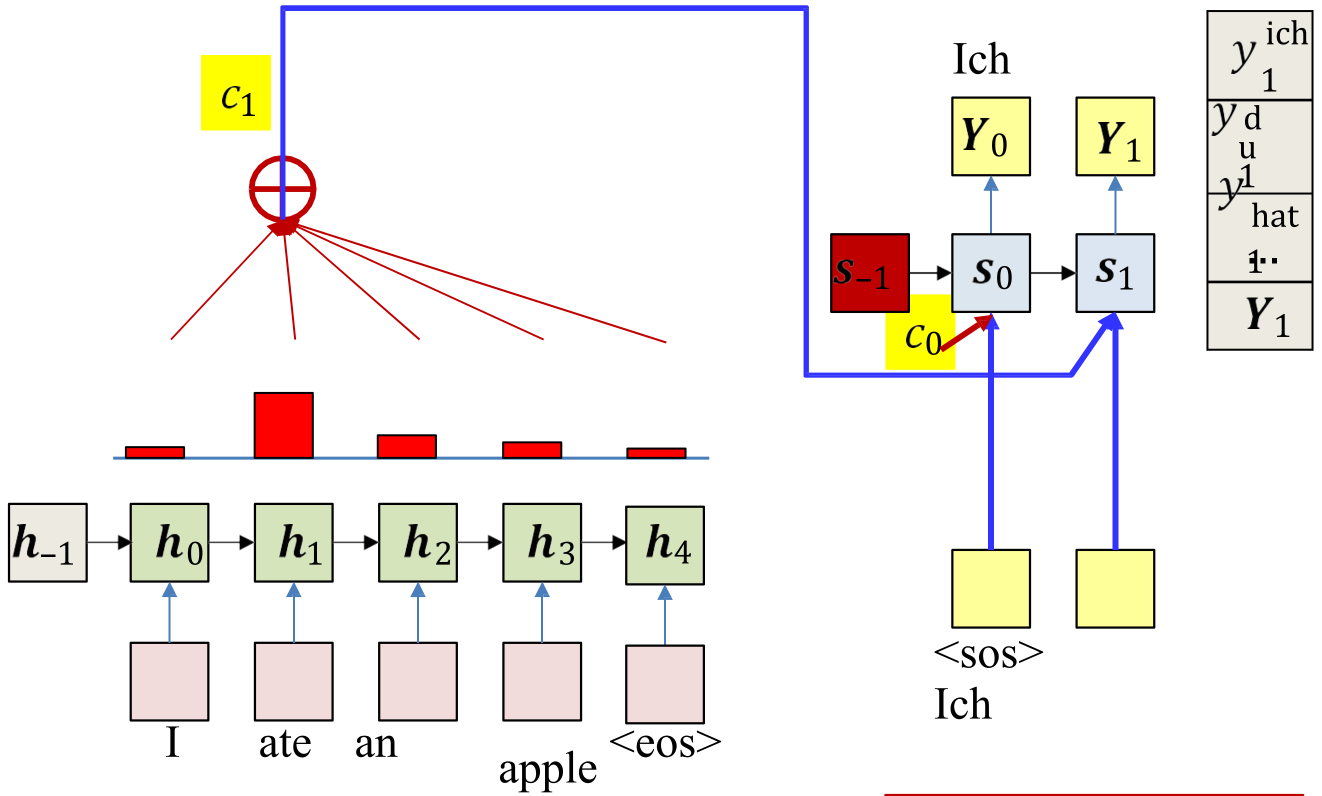
$$g(h_i, s_0) = h_i^T W_g s_0$$

$$e_i(1) = g(h_i, s_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

$$c_1 = \frac{1}{N}\sum_i^N w_i(1) h_i$$

Converting an input: Inference



- The decoder computes
  - s_1
  - A probability distribution over the output vocabulary

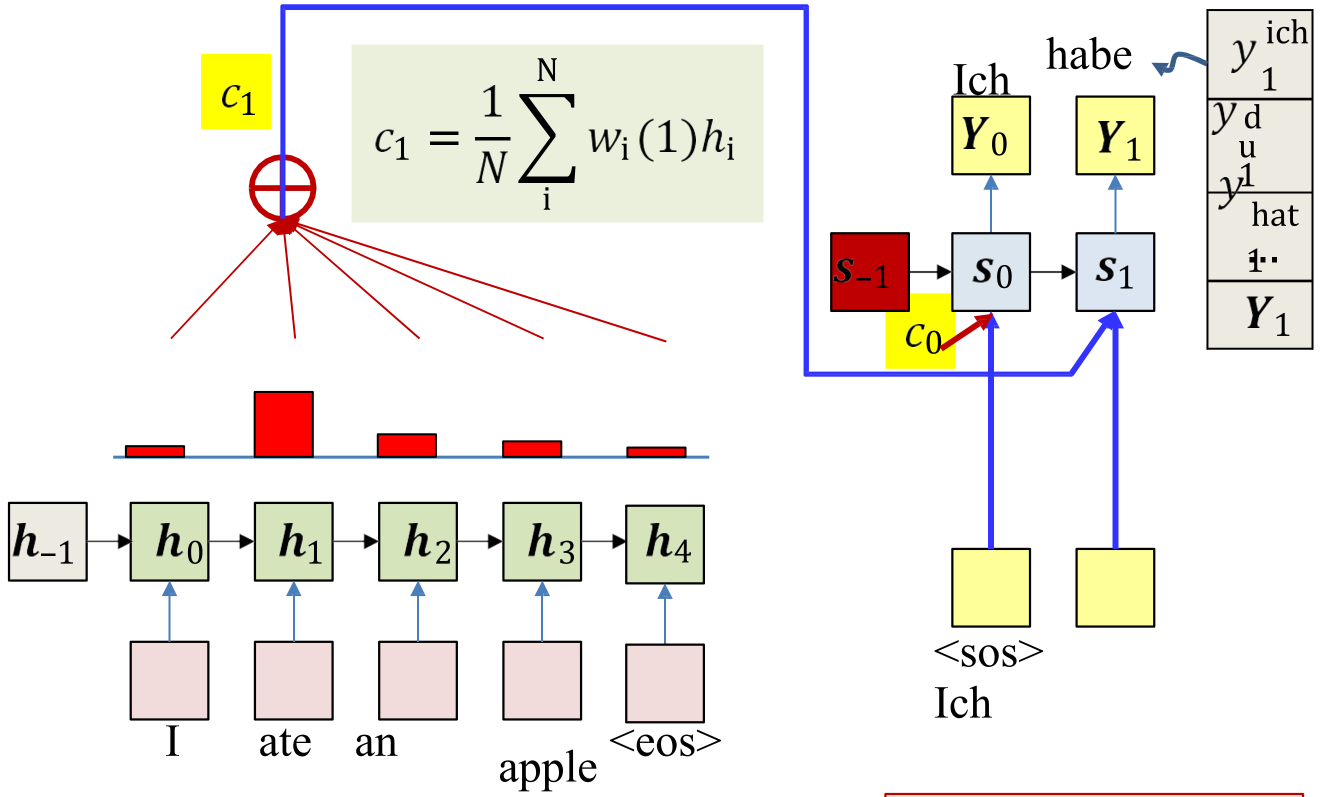
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

$$c_1 = \frac{1}{N}\sum_i^N w_i(1) h_i$$

Converting an input: Inference



- Sample the second word from the output distribution

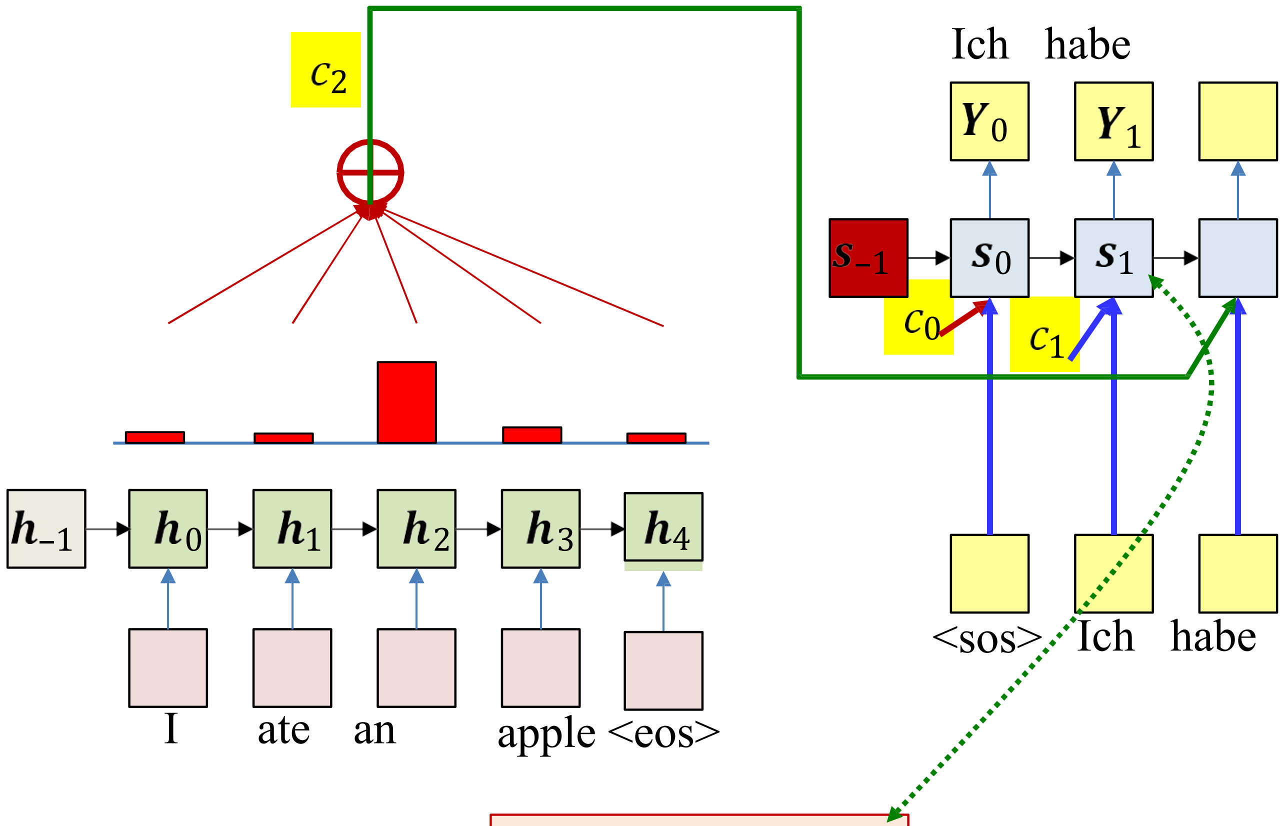
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

$$c_1 = \frac{1}{N}\sum_{i}^{N} w_i(1)\boldsymbol{h}_i$$

Converting an input: Inference



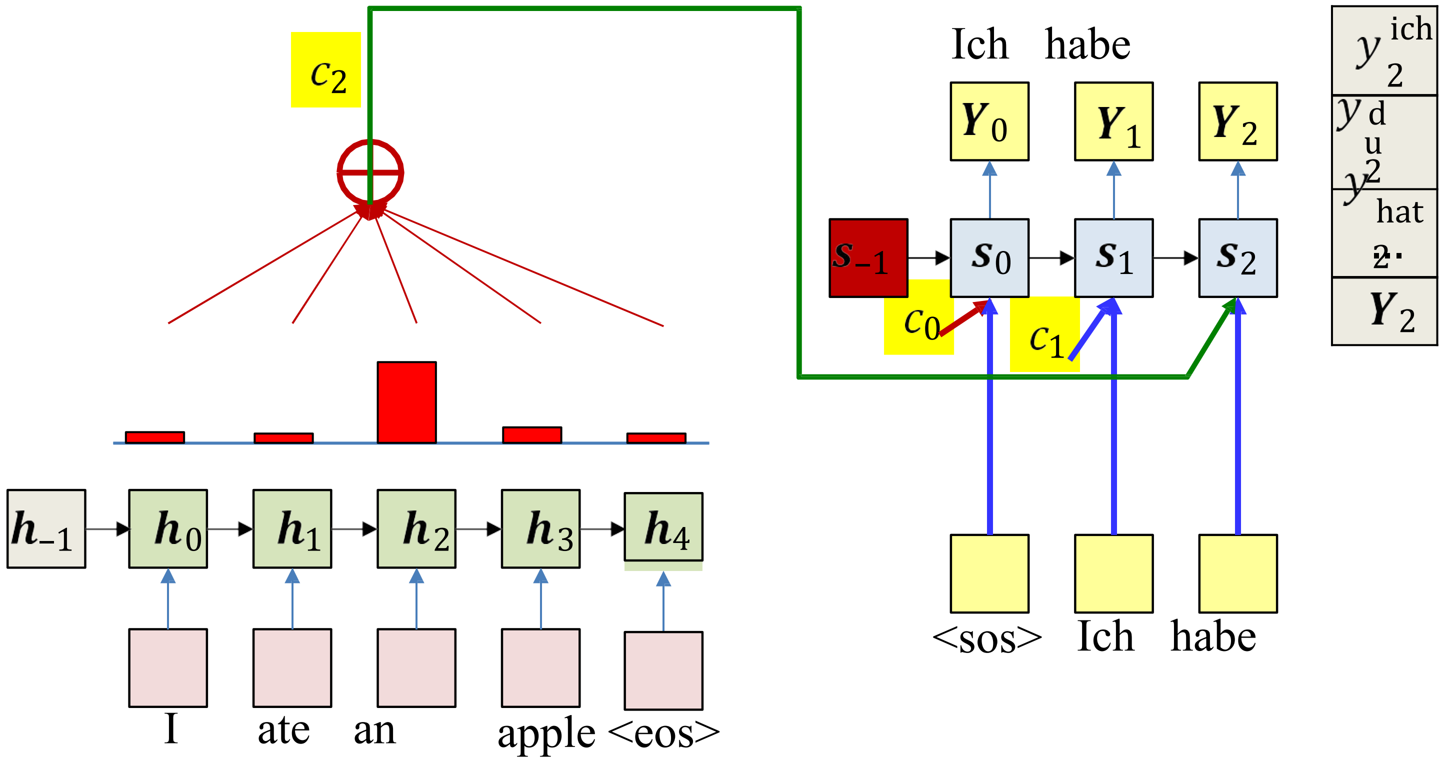
$$g(\boldsymbol{h}_i, \boldsymbol{s}_1) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_1$$

$$e_i(2) = g(\boldsymbol{h}_i, \boldsymbol{s}_1)$$

$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

$$c_2 = \frac{1}{N} \sum_i^N w_i(2) h_i$$

Converting an input: Inference



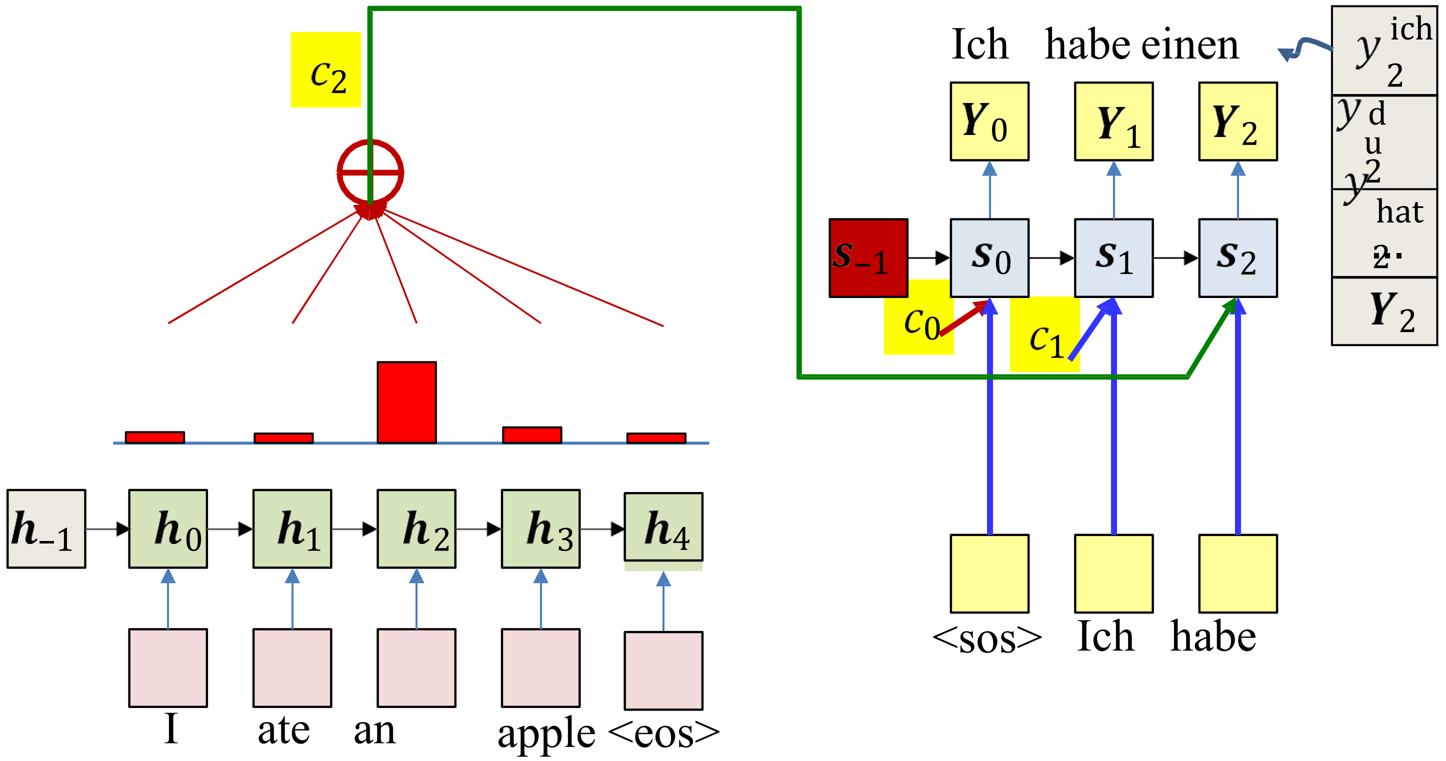
$$g(h_i, s_1) = h_i^T W_g s_1$$

$$e_i(2) = g(h_i, s_1)$$

$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

$$c_2 = \frac{1}{N}\sum_i^N w_i(2) h_i$$

Converting an input: Inference



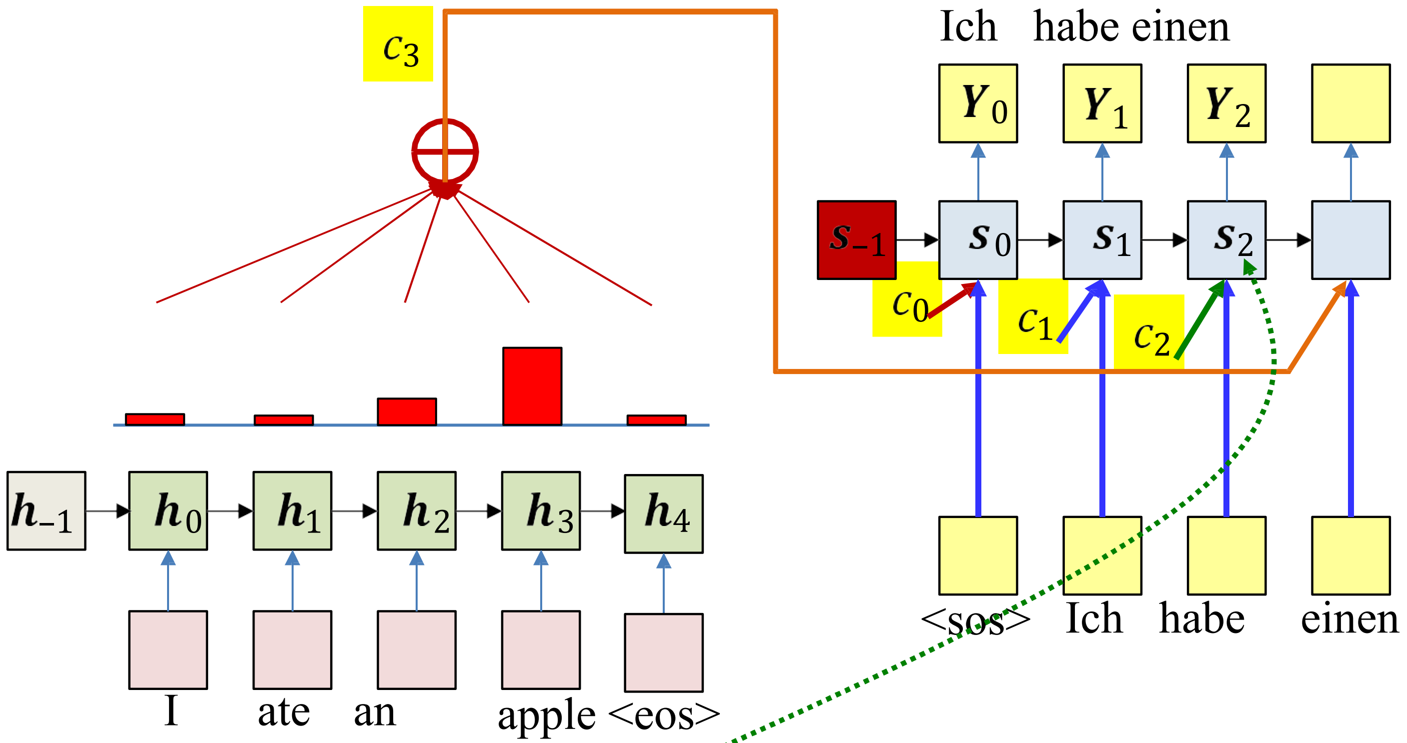
$$g(\boldsymbol{h}_i, \boldsymbol{s}_1) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_1$$

$$e_i(2) = g(\boldsymbol{h}_i, \boldsymbol{s}_1)$$

$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

$$c_2 = \frac{1}{N} \sum_{i}^{N} w_i(2) h_i$$

Converting an input: Inference

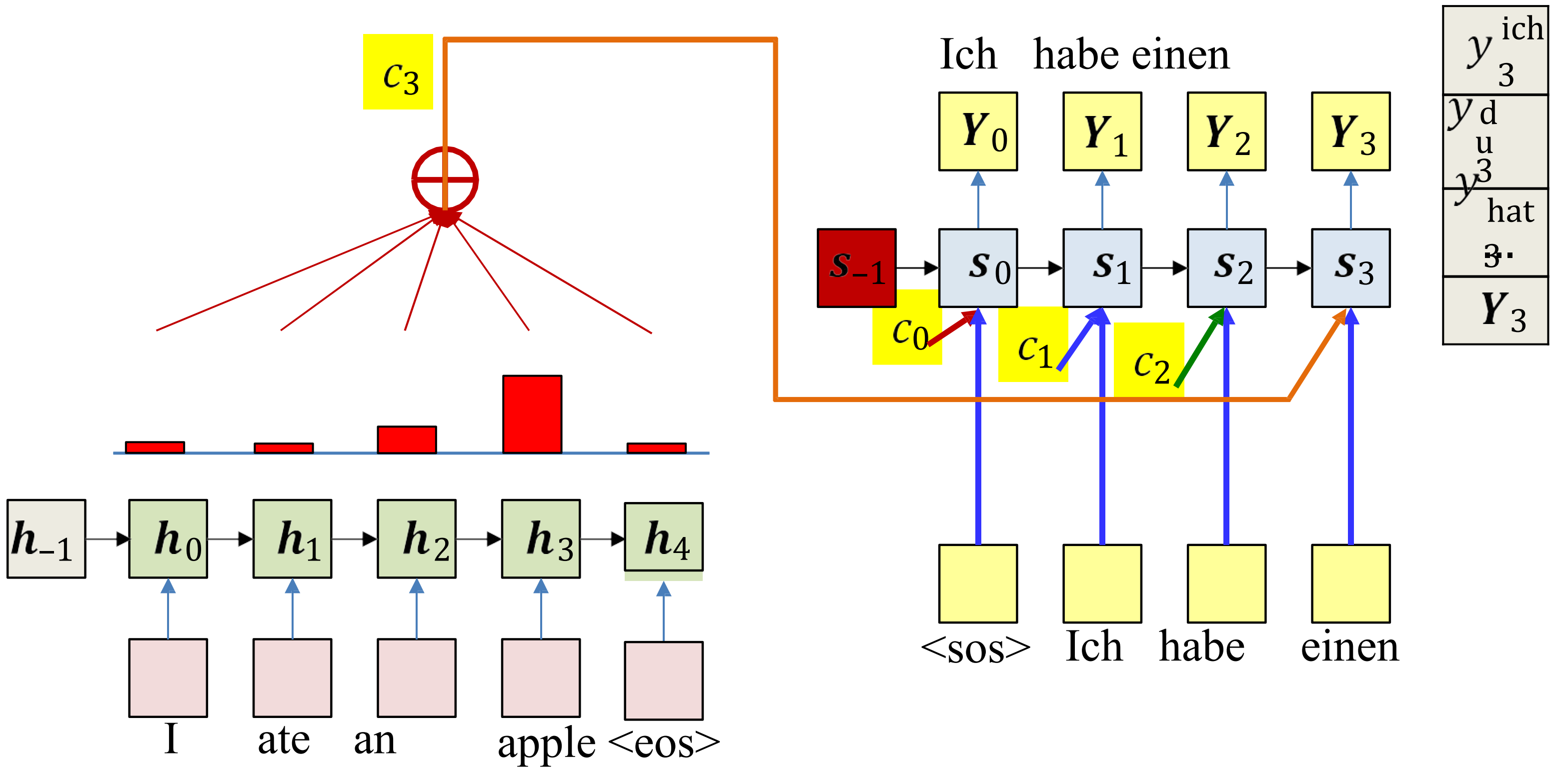


$$e_i(3) = g(h_i, s_2)$$

$$w_i(3) = \frac{\exp(e_i(3))}{\sum_j \exp(e_j(3))}$$

$$c_3 = \frac{1}{N}\sum_i^N w_i(3) h_i$$

Converting an input: Inference

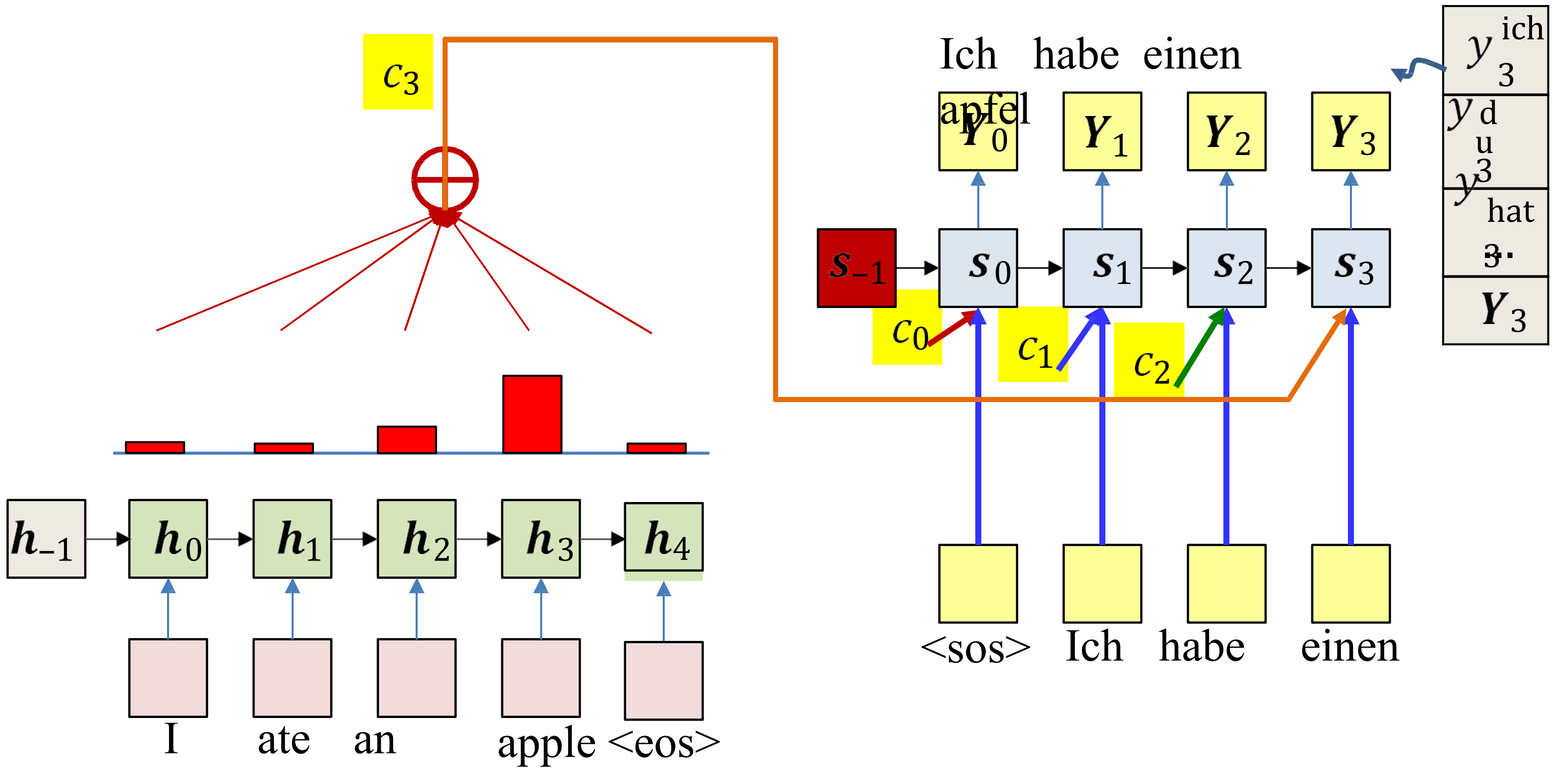


$$e_i(3) = g(\boldsymbol{h}_i, \boldsymbol{s}_2)$$

$$w_i(3) = \frac{\exp(e_i(3))}{\sum_j \exp(e_j(3))}$$

$$c_3 = \frac{1}{N}\sum_i^N w_i(3) h_i$$

Converting an input: Inference

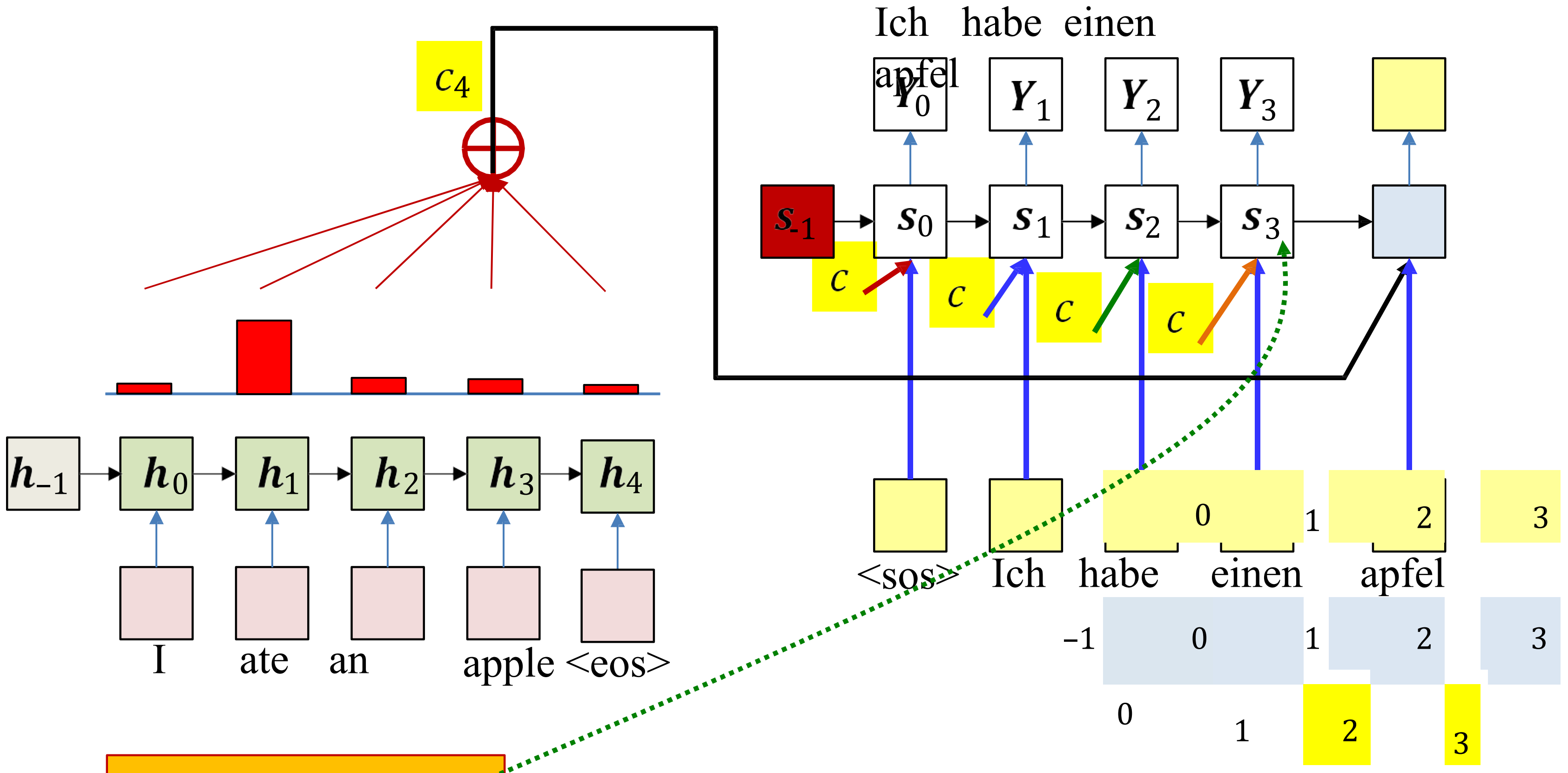


$$e_i(3) = g(\boldsymbol{h}_i, \boldsymbol{s}_2)$$

$$w_i(3) = \frac{\exp(e_i(3))}{\sum_j \exp(e_j(3))}$$

$$c_3 = \frac{1}{N}\sum_i^N w_i(3) h_i$$

Converting an input: Inference

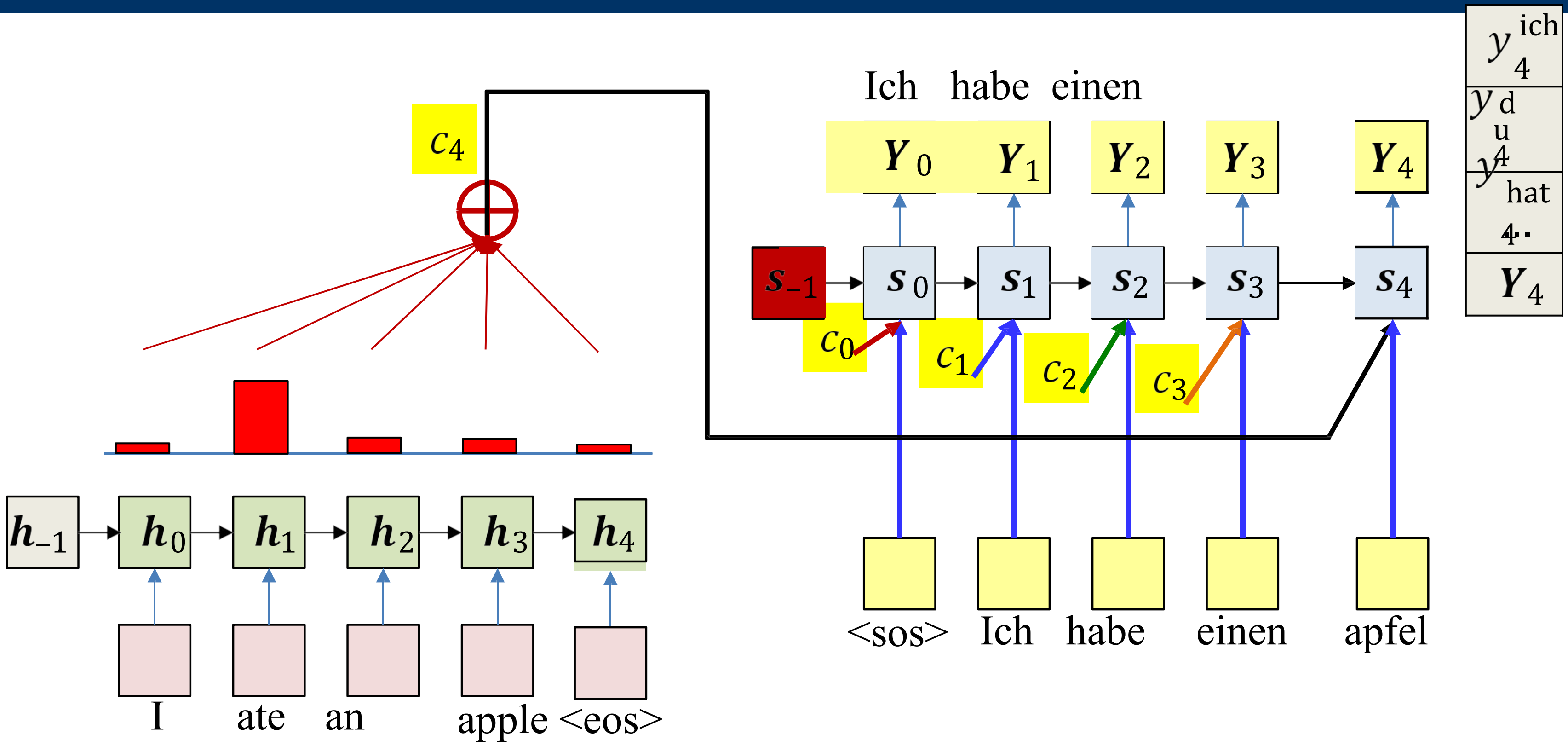


$$e_i(4) = g(h_i, s\ )$$

$$w_i(4) = \frac{\exp(e_i(4))}{\sum_j \exp(e_j(4))}$$

$$c_4 = \frac{1}{N}\sum_i^N w_i(4) h_i$$

Converting an input: Inference



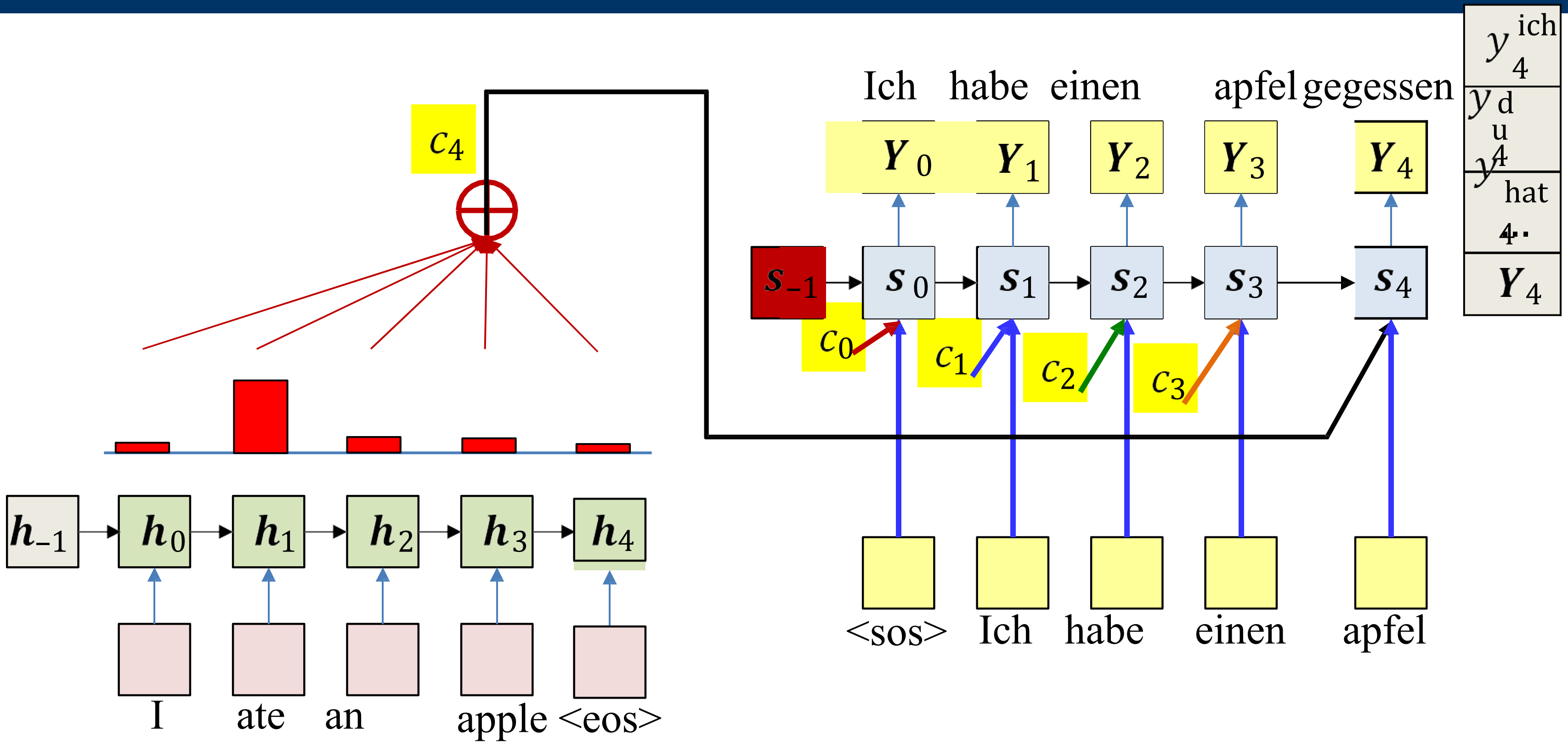
c_4
Ich habe einen
Y_0 Y_1 Y_2 Y_3 Y_4
y_4^{ich}
y_4^{du}
y_4^{hat}
$4\cdot\cdot$
Y_4
s_{-1} s_0 s_1 s_2 s_3 s_4
c_0 c_1 c_2 c_3
<sos> Ich habe einen apfel
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>

$$e_i(4) = g(\boldsymbol{h}_i, \boldsymbol{s}_3)$$

$$w_i(4) = \frac{\exp(e_i(4))}{\sum_j \exp(e_j(4))}$$

$$c_4 = \frac{1}{N}\sum_{i}^{N} w_i(4) h_i$$

Converting an input: Inference

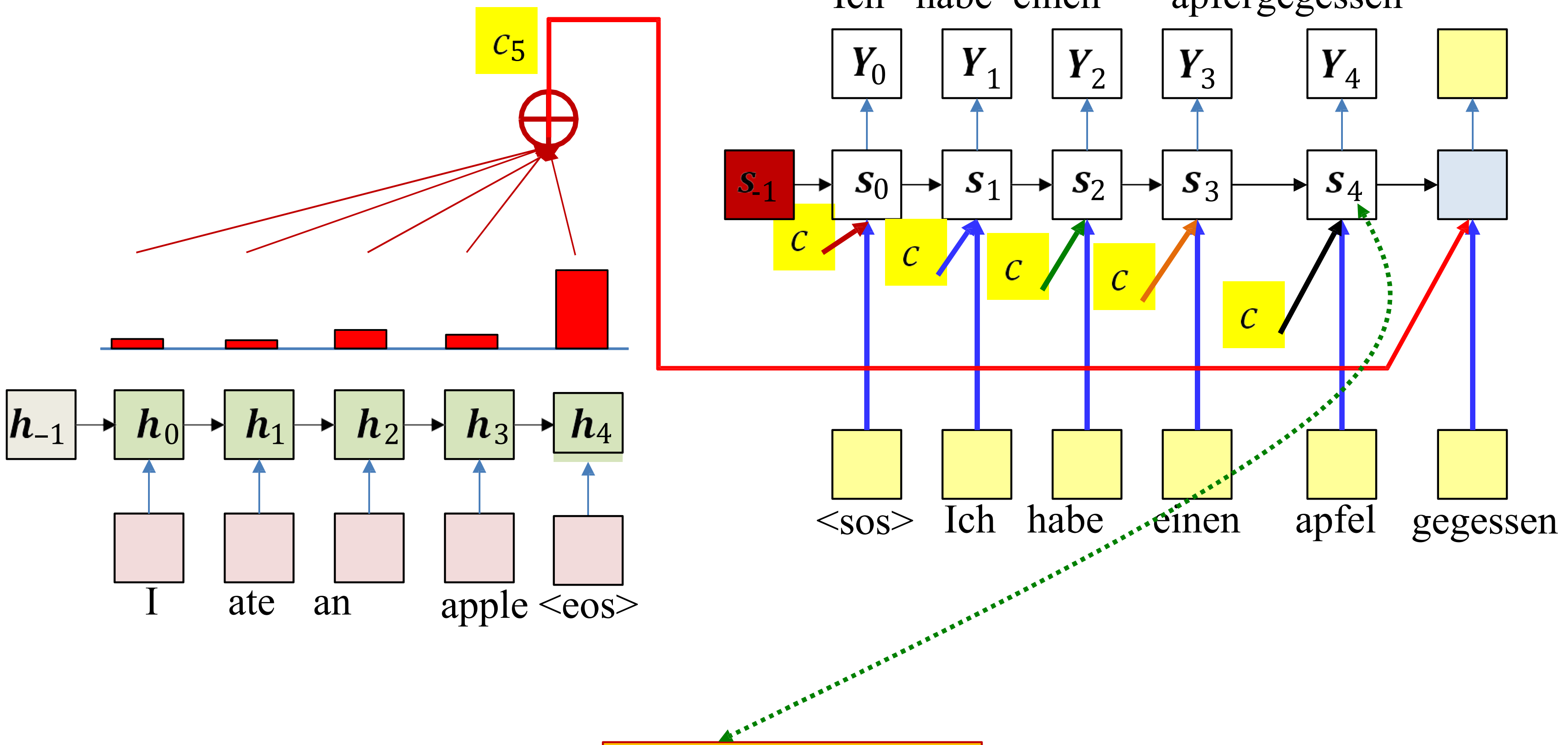


$$e_i(4) = g(\boldsymbol{h}_i, \boldsymbol{s}_3)$$

$$w_i(4) = \frac{\exp(e_i(4))}{\sum_j \exp(e_j(4))}$$

$$c_4 = \frac{1}{N}\sum_i^N w_i(4) h_i$$

Converting an input: Inference



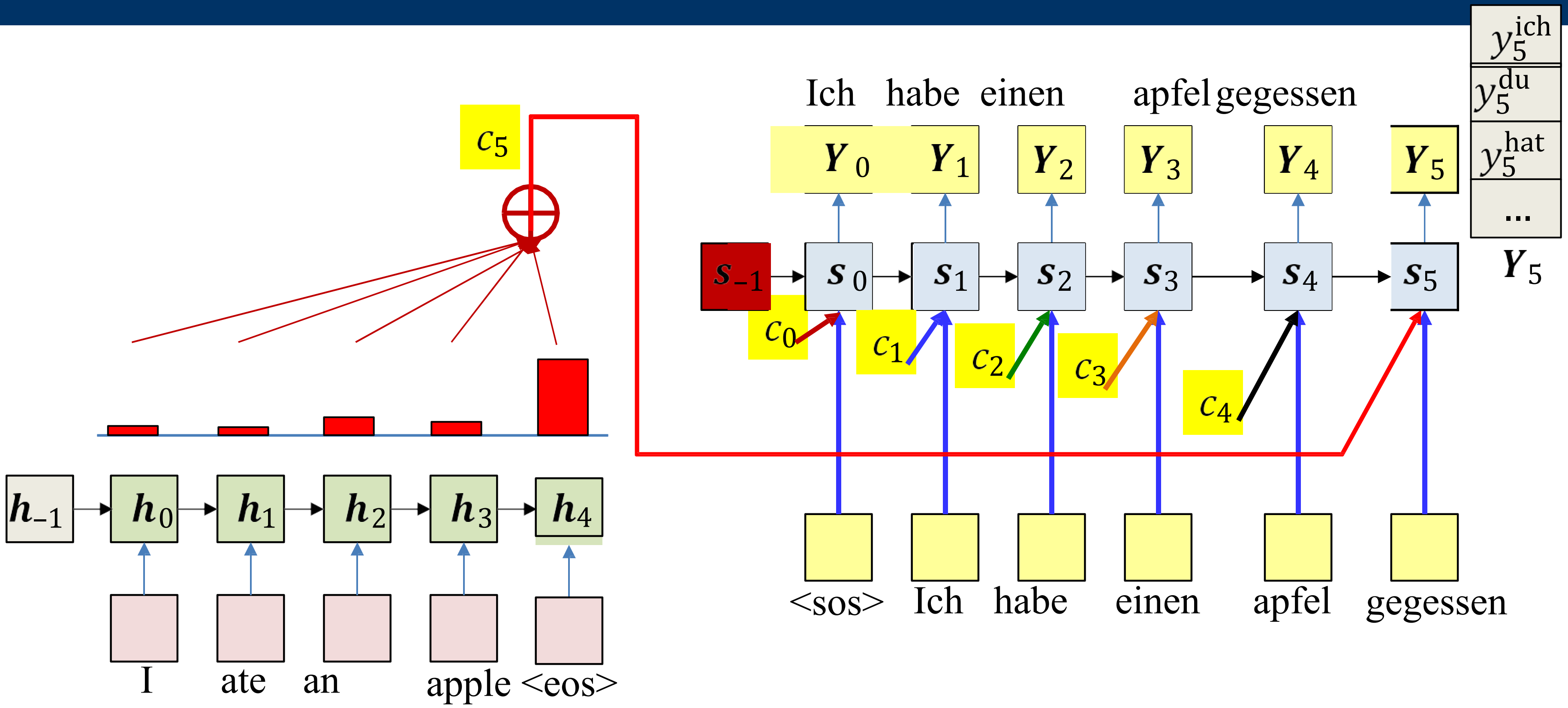
c_5
Ich habe einen apfel gegessen
Y_0 Y_1 Y_2 Y_3 Y_4
s_{-1} s_0 s_1 s_2 s_3 s_4
c
<sos> Ich habe einen apfel gegessen
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>

$$e_i(5) = g(\boldsymbol{h}_i, \boldsymbol{s}\ \ )$$

$$w_i(5) = \frac{\exp_4(e_i(5))}{\sum_j \exp(e_j(5))}$$

$$c_5 = \frac{1}{N}\sum_{i}^{N} w_i(5) h_i$$

Converting an input: Inference



c_5
Ich habe einen apfel gegessen
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5
y_5^{ich}
y_5^{du}
y_5^{hat}
...
Y_5
s_{-1} s_0 s_1 s_2 s_3 s_4 s_5
c_0 c_1 c_2 c_3 c_4
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>
<sos> Ich habe einen apfel gegessen

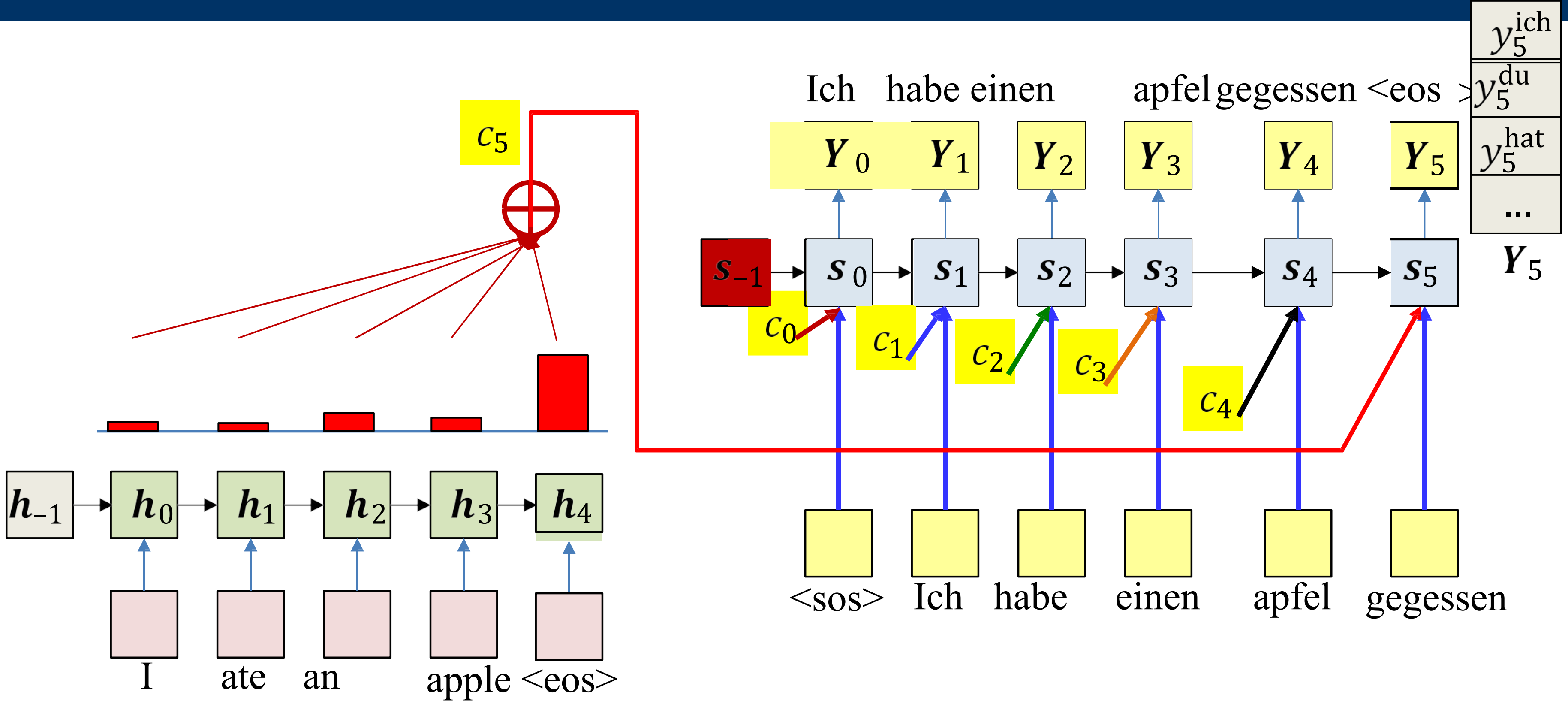
$$e_i(5) = g(h_i, s\ )$$

$$w_i(5) = \frac{\exp(e_i(5))}{\sum_j \exp(e_j(5))}$$

$$c_5 = \frac{1}{N}\sum_i^N w_i(5) h_i$$

55

Converting an input: Inference



c_5
Ich habe einen apfel gegessen <eos>
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5
y_5^{ich}
y_5^{du}
y_5^{hat}
...
Y_5
s_{-1} s_0 s_1 s_2 s_3 s_4 s_5
c_0 c_1 c_2 c_3 c_4
<sos> Ich habe einen apfel gegessen
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>

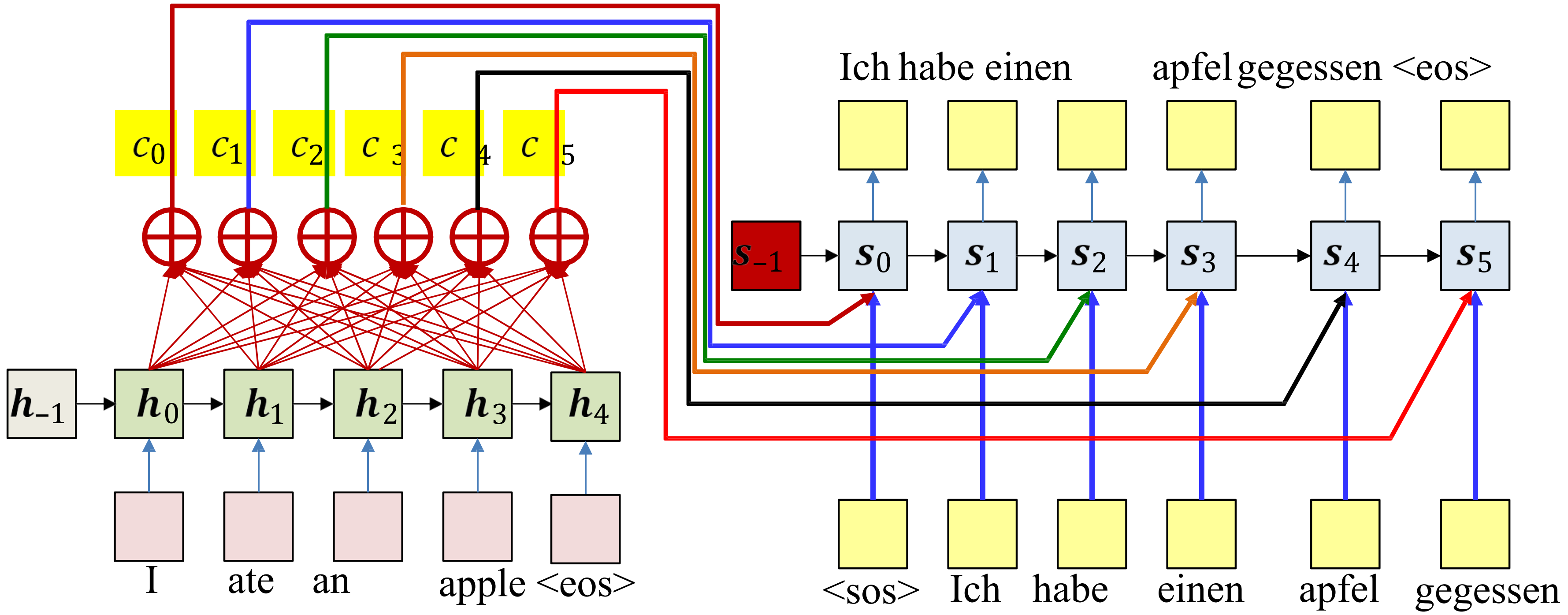
Continue this process until <eos> is drawn

$$e_i(5) = g(\boldsymbol{h}_i, \boldsymbol{s}_4)$$

$$w_i(5) = \frac{\exp(e_i(5))}{\sum_j \exp(e_j(5))}$$

$$c_5 = \frac{1}{N}\sum_i^N w_i(5) h_i$$

Attention-based decoding



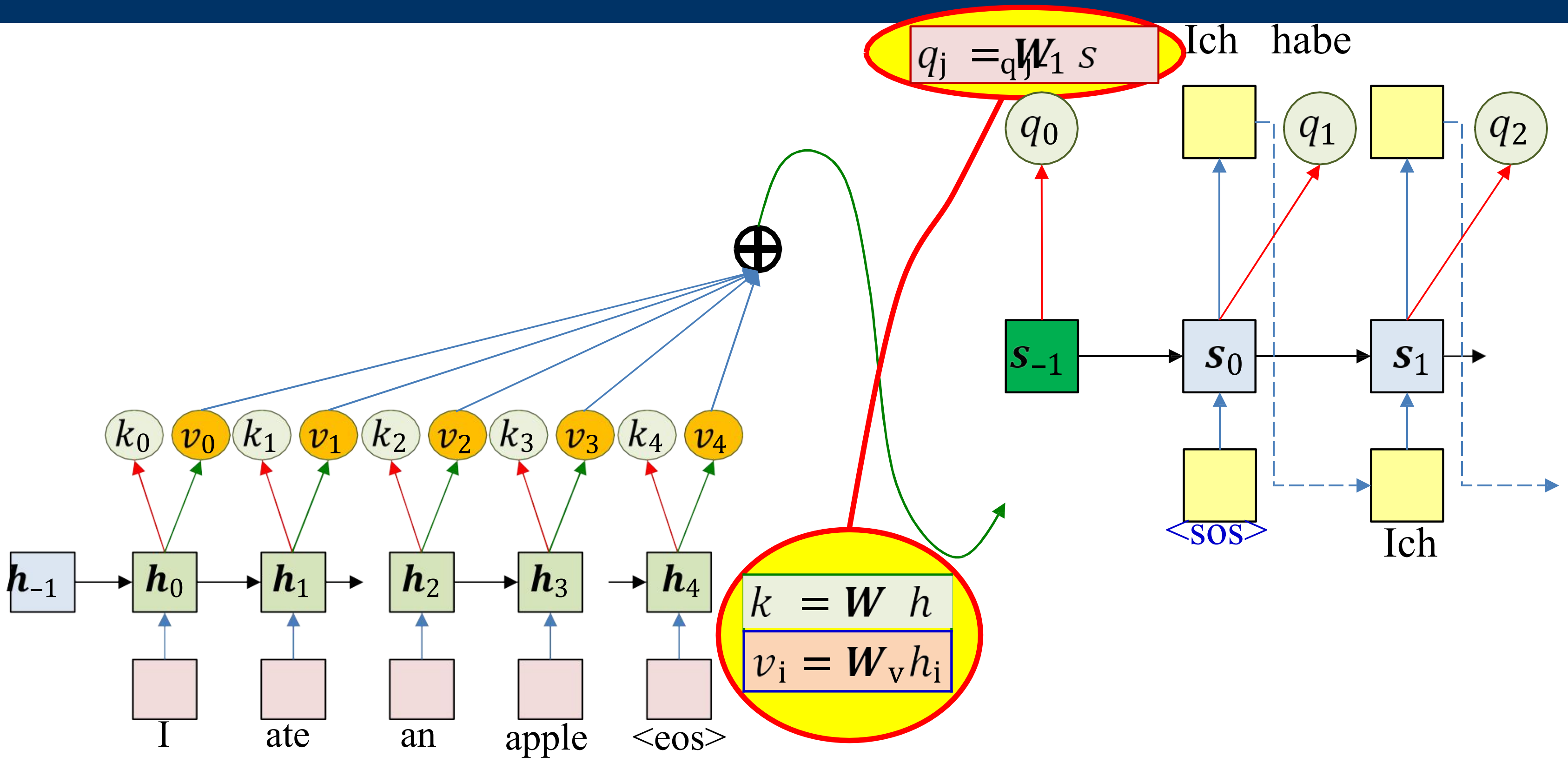
Ich habe einen apfel gegessen <eos>
c_0 c_1 c_2 c_3 c_4 c_5
s_{-1} s_0 s_1 s_2 s_3 s_4 s_5
h_{-1} h_0 h_1 h_2 h_3 h_4
I ate an apple <eos>
<sos> Ich habe einen apfel gegessen

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

$$c_t = \frac{1}{N}\sum_i^N w_i(t) h_i$$

Modification: Query key value



- Encoder outputs an explicit "key" and "value" at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit "query" at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

Modification: Query key value

$$e_i(t) = g(\boldsymbol{k}_i, \boldsymbol{q}_t)$$

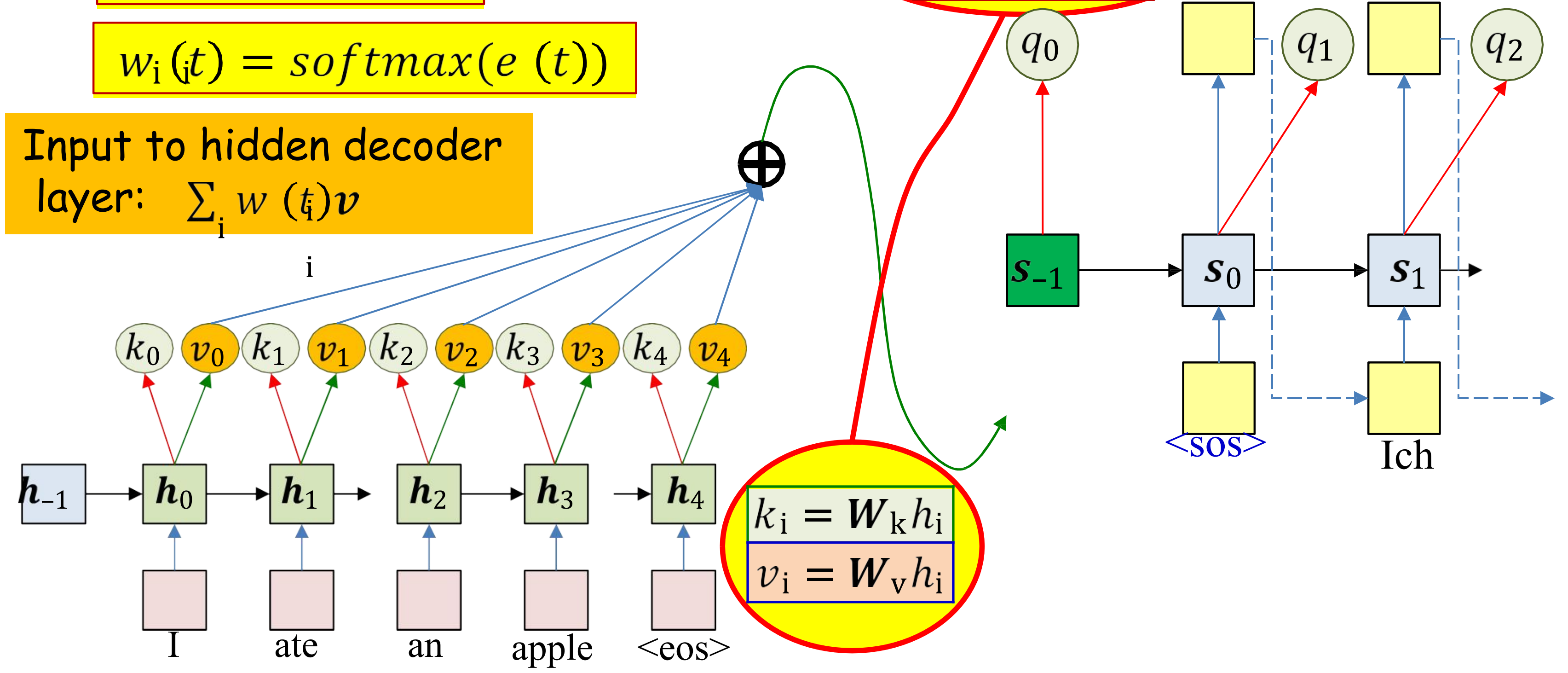
$$w_i(t) = softmax(e_i(t))$$

Input to hidden decoder layer: $\sum_i w_i(t)\boldsymbol{v}_i$

$$q_t = \boldsymbol{W}_q s_{t-1}$$

$$k_i = \boldsymbol{W}_k h_i$$

$$v_i = \boldsymbol{W}_v h_i$$



Ich
habe
q_0
q_1
q_2
$\boldsymbol{s}_{-1}$
$\boldsymbol{s}_0$
$\boldsymbol{s}_1$
<sos>
Ich
k_0 v_0 k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4
$\boldsymbol{h}_{-1}$ $\boldsymbol{h}_0$ $\boldsymbol{h}_1$ $\boldsymbol{h}_2$ $\boldsymbol{h}_3$ $\boldsymbol{h}_4$
I
ate
an
apple
<eos>

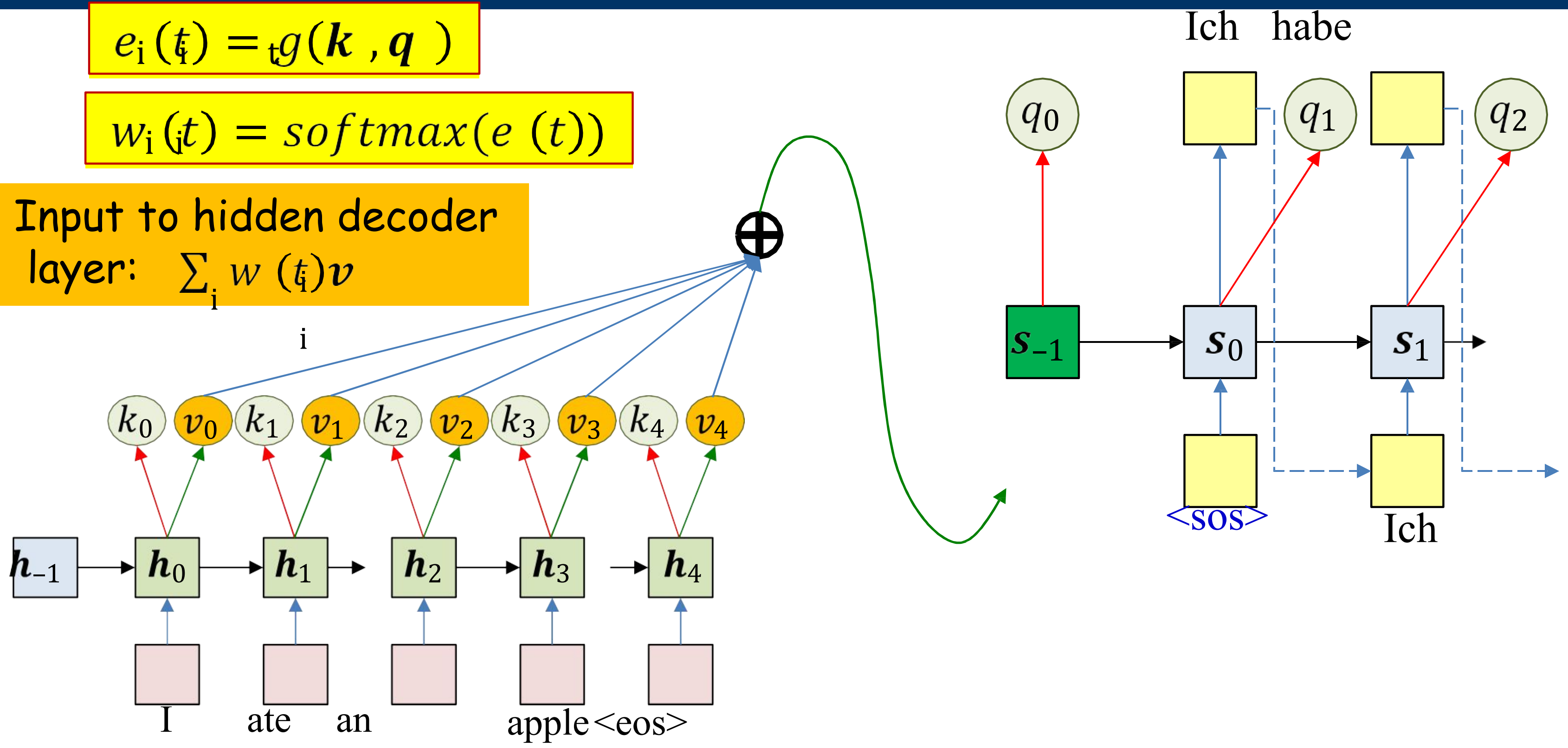
- Encoder outputs an explicit “key” and “value” at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit “query” at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

Modification: Query key value

$e_i(t) = g(\boldsymbol{k}_i, \boldsymbol{q}_t)$

$w_i(t) = softmax(e_i(t))$

Input to hidden decoder layer: $\sum_i w_i(t)\boldsymbol{v}_i$



Special case: $k_i = v_i = h_i$

$q_t = s_{t-1}$

The actual context is a weighted sum of value

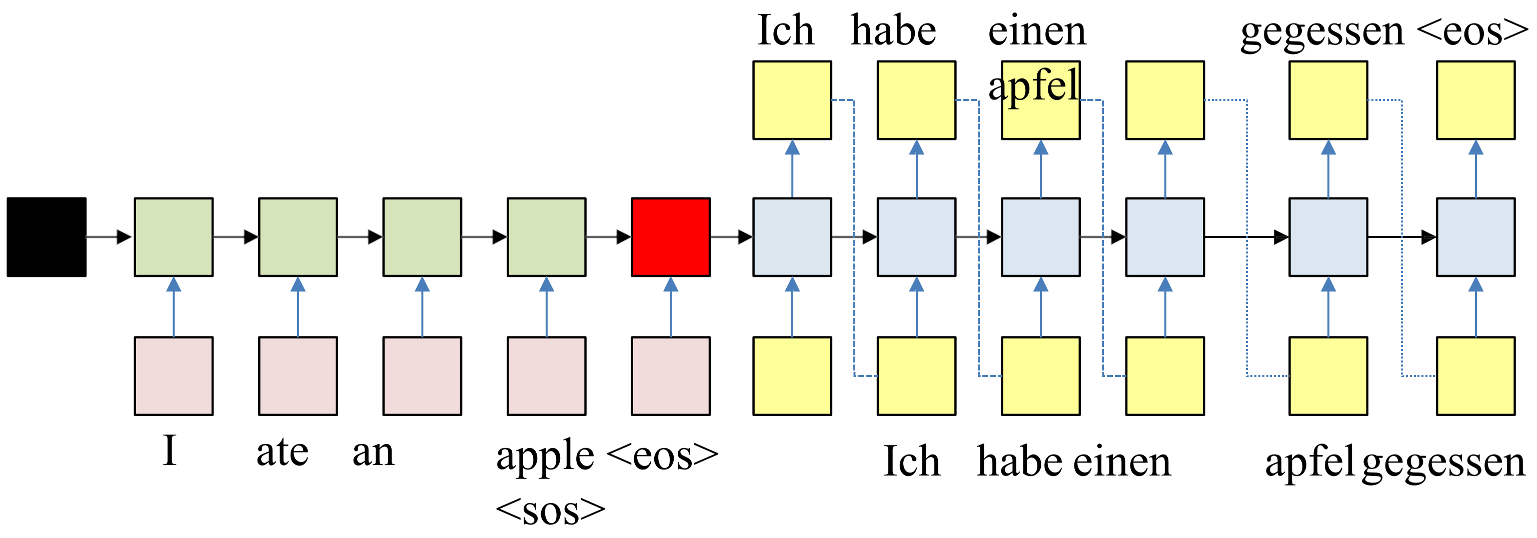
Pseudocode

```
# Assuming encoded input
#   (K,V) = [$k_{enc}$[0]… $k_{enc}$[T]], [$v_{enc}$[0]… $v_{enc}$[T]]
# is available

t = -1
$h_{out}$[-1] = 0    # Initial Decoder hidden state
q[0] = 0        # Initial query

# Note: begins with a “start of sentence” symbol
#  <sos> and <eos> may be identical
$Y_{out}$[0] = <sos>
do
    t = t+1
    C  = compute_context_with_attention(q[t], K, V)
    y[t],$h_{out}$[t],q[t+1] = RNN_decode_step($h_{out}$[t-1], $y_{out}$[t-1], C)
    $y_{out}$[t] = generate(y[t]) # Random, or greedy
until $y_{out}$[t] == <eos>
```

Pseudocode : Computing context with attention

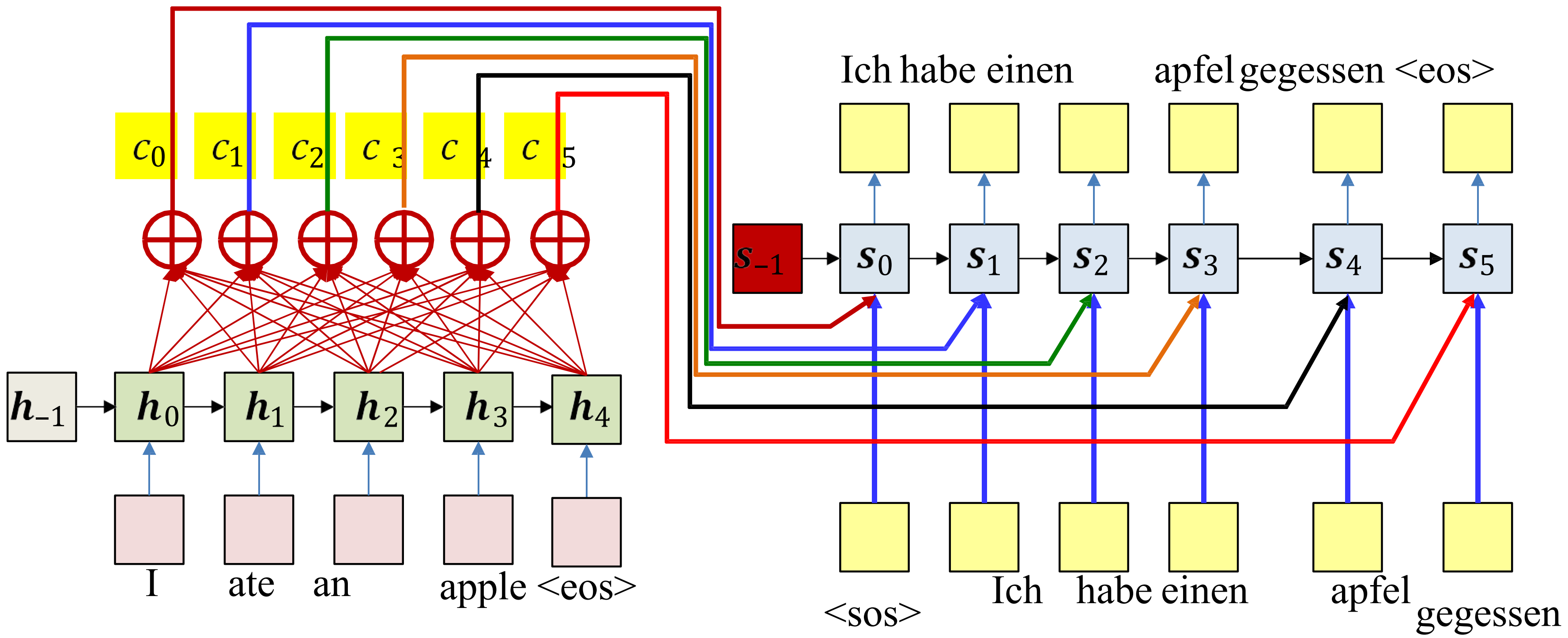
```
# Takes in previous state, encoder states, outputs attention-weighted context
function compute_context_with_attention(q, K, V)   #
    First compute attention
    e = []
    for t = 1:T    # Length of input
        e[t] = raw_attention(q, K[t])
    end
    maxe = max(e)  # subtract max(e) from everything to prevent underflow
    a[1..T] = exp(e[1..T] - maxe)    # Component-wise exponentiation
    suma = sum(a) # Add all elements of a
    a[1..T] = a[1..T]/suma

    C = 0
    for t = 1..T
        C += a[t] * V[t]
    end

    return C
```

Recap: Seq2Seq models



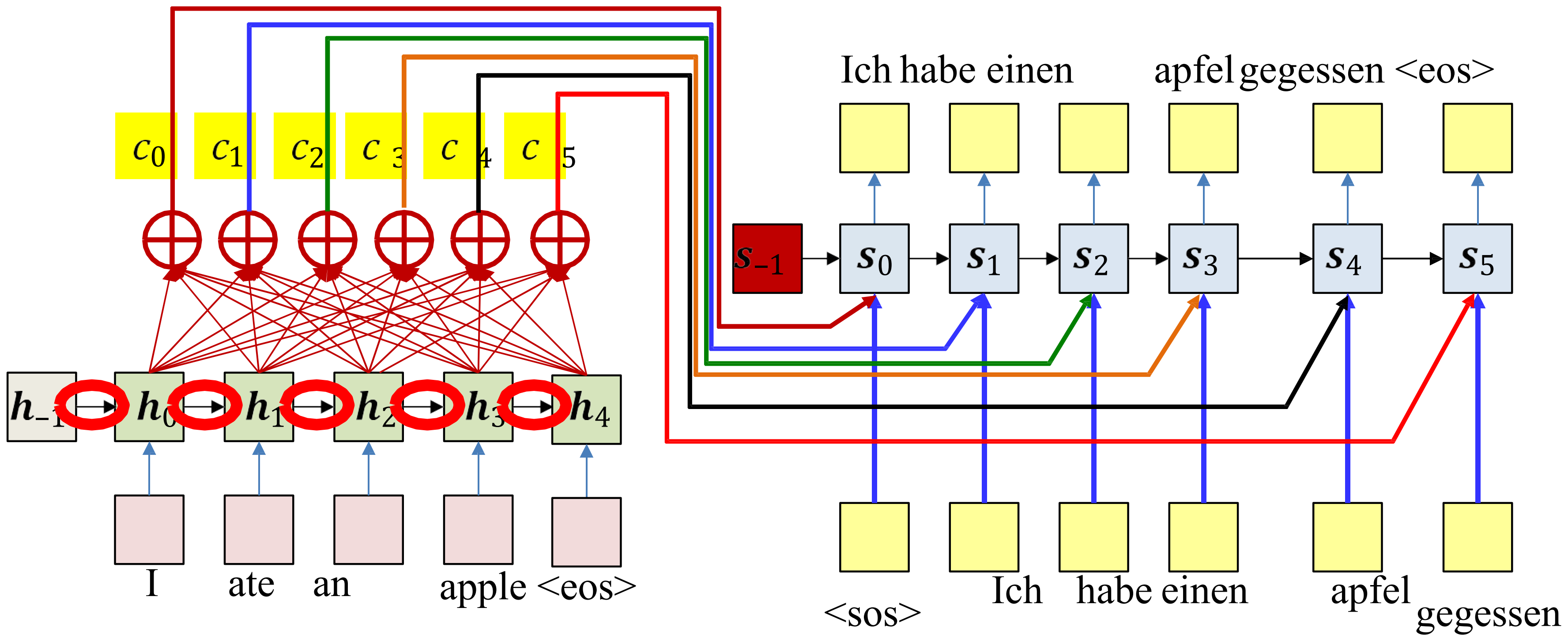
- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs

Recap: Attention Models



- Encoder recurrently produces hidden representations of input word sequence
- Decoder recurrently generates output word sequence
  - For each output word the decoder uses a weighted average of the hidden input representations as input “context”, along with the recurrent hidden state and the previous output word

Recap: Attention Models



- Problem: Because of the recurrence, the hidden representation for any word is also influenced by *all* preceding words
  - The decoder is actually paying attention to the sequence, and not just the word

- If the decoder is automatically figuring out which words of the input to attend to at each time, is recurrence in the input even necessary?

Impact of Attention

Improves translation accuracy

Helps handle **longer sequences**

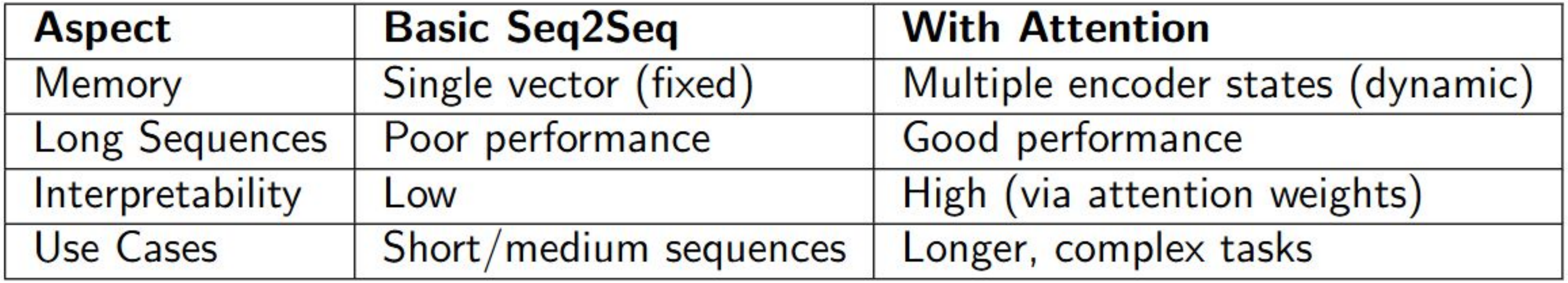
Interpretable: shows what the model is focusing on

Forms the foundation of **Transformer models**

Led to development of:

- ► Bahdanau Attention (Additive, 2014)
- ► Luong Attention (Multiplicative, 2015)
- ► Self-Attention and Transformers (2017+)

Seq2Seq vs Seq2Seq + Attention



Aspect	Basic Seq2Seq	With Attention
Memory	Single vector (fixed)	Multiple encoder states (dynamic)
Long Sequences	Poor performance	Good performance
Interpretability	Low	High (via attention weights)
Use Cases	Short/medium sequences	Longer, complex tasks

Applications of Seq2Seq + Attention

- Machine Translation
- Text Summarization
- Chatbots and Dialog Systems
- Speech Recognition
- Question Answering
- Video Captioning
- DNA Sequence Modeling

Limitations

- Still sequential — **not fully parallelizable**
- Attention adds computational cost
- Hard to interpret in large-scale models
- Might struggle with very long-range dependencies in huge contexts

Future Directions

- **Transformers**: Fully attention-based, no recurrence
- **Efficient Attention Models**: Longformer, Reformer, Linformer
- **Multimodal Attention**: Vision + Text
- **Memory-Augmented Models**
- **Structured Attention**: Parses, syntax, and alignment bias

Attention paved the way for GPT, BERT, and LLMs

Summary

- Seq2Seq enables mapping input to output sequences of variable lengths
- **Bottleneck:** Fixed-size context vector limits learning capacity
- **Attention:** Improves performance by giving **adaptive access** to input states
- Attention $\rightarrow$ Transformer $\rightarrow$ LLMs
- Still evolving: From additive attention to self-attention and beyond

References

These slides have been adapted from

- Younes Mourri & Lukasz Kaiser, Natural Language Processcing Specialization, DeepLearning.Ai
- Bhiksha Raj & Rita Singh, 11-785 Introduction to Deep Learning, CMU

References

Foundational Papers:

- Sutskever et al. (2014). *Sequence to Sequence Learning with Neural Networks*. NeurIPS.
- Bahdanau et al. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate*.
- Luong et al. (2015). *Effective Approaches to Attention-based Neural Machine Translation*.
- Vaswani et al. (2017). *Attention Is All You Need* (Transformer).
- Chan et al. (2016). *Listen, Attend and Spell* (Speech recognition).

References

Courses & Tutorials:

- Stanford CS224n: Lecture 9 (Attention)
- DeepLearning.ai NLP Specialization (Coursera)
- Harvard NLP Annotated Transformer: `http://nlp.seas.harvard.edu/2018/04/03/attention.html`
- Jay Alammar's blog: "The Illustrated Transformer"