Deep Learning
Sequence to Sequence models: Attention Models

Spring 2024
Attendance: @1484

Sequence to sequence models

I ate an apple → **Seq2seq** → Ich habe einen apfel gegessen
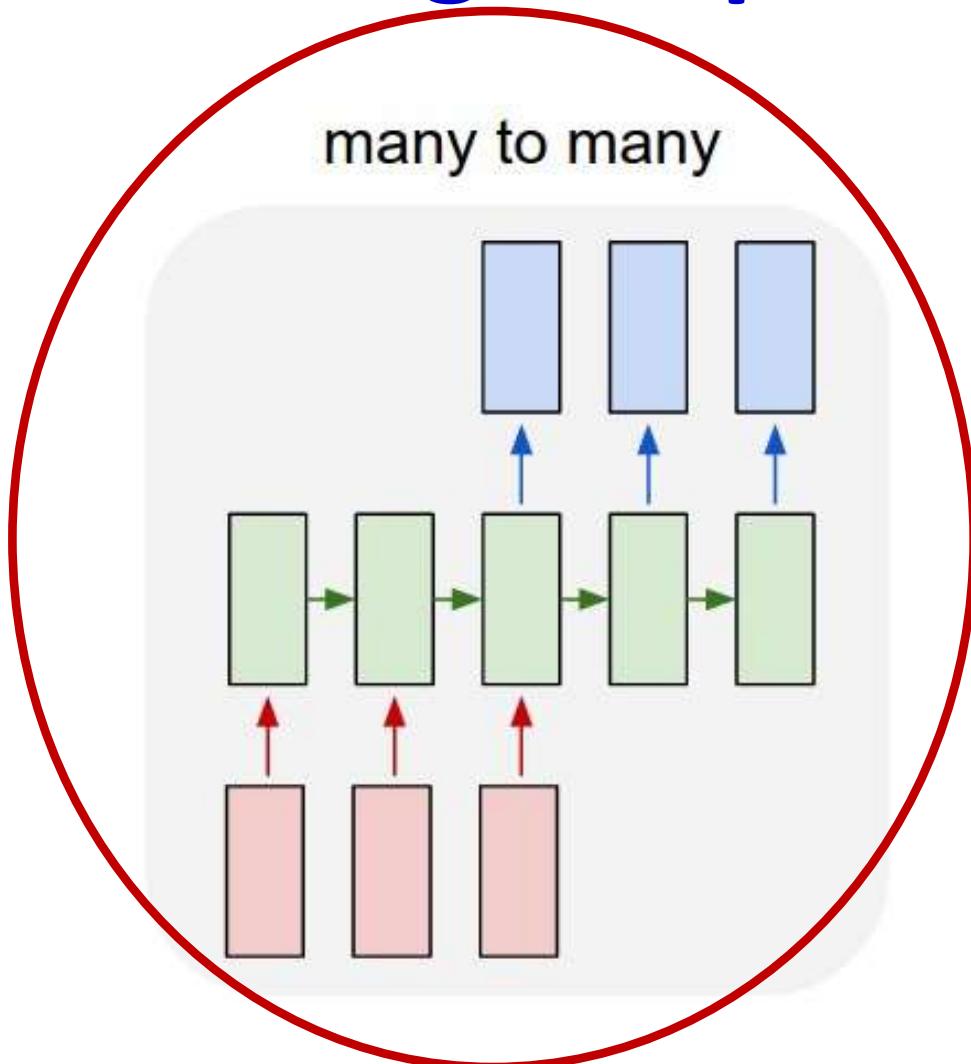
- Sequence goes in, sequence comes out
- No notion of "time synchrony" between input and output
  - May even not even maintain order of symbols
    - E.g. "I ate an apple" → "Ich habe einen apfel gegessen"
  - Or even seem related to the input
    - E.g. "My screen is blank" → "Please check if your computer is plugged in."

Modelling the problem
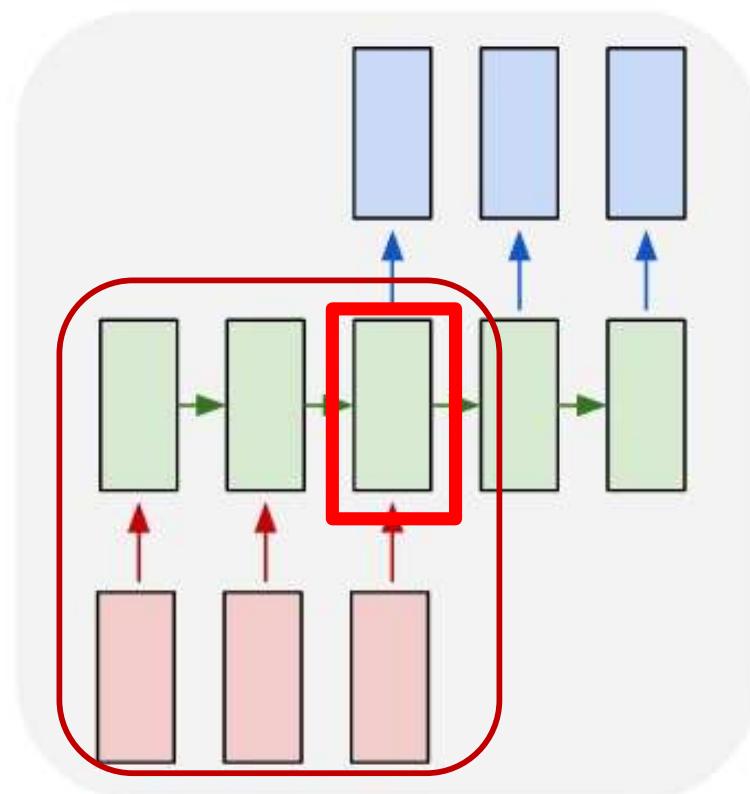
- *Delayed* sequence to sequence

# Modelling the problem

many to many

First process the input and generate a hidden representation for it
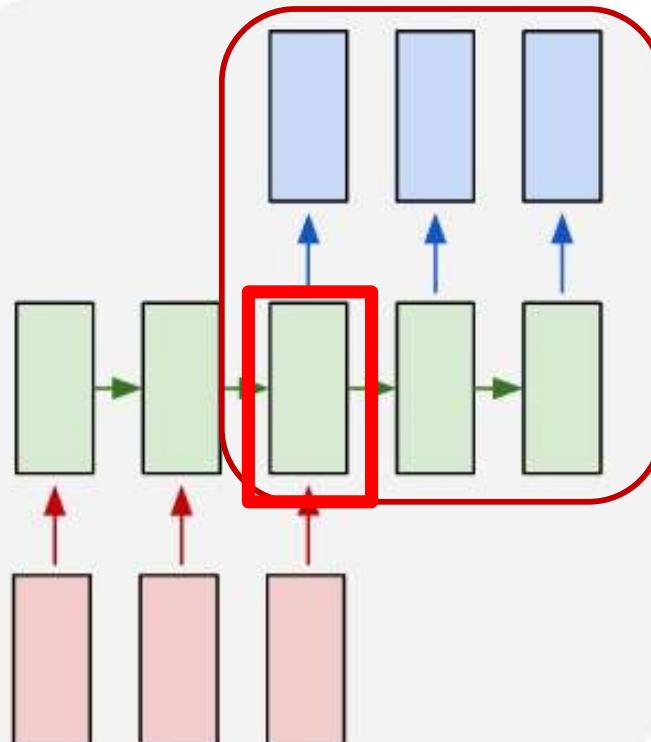
- *Delayed* sequence to sequence

Modelling the problem

many to many

First process the input and generate a hidden representation for it

Then use it to generate an output
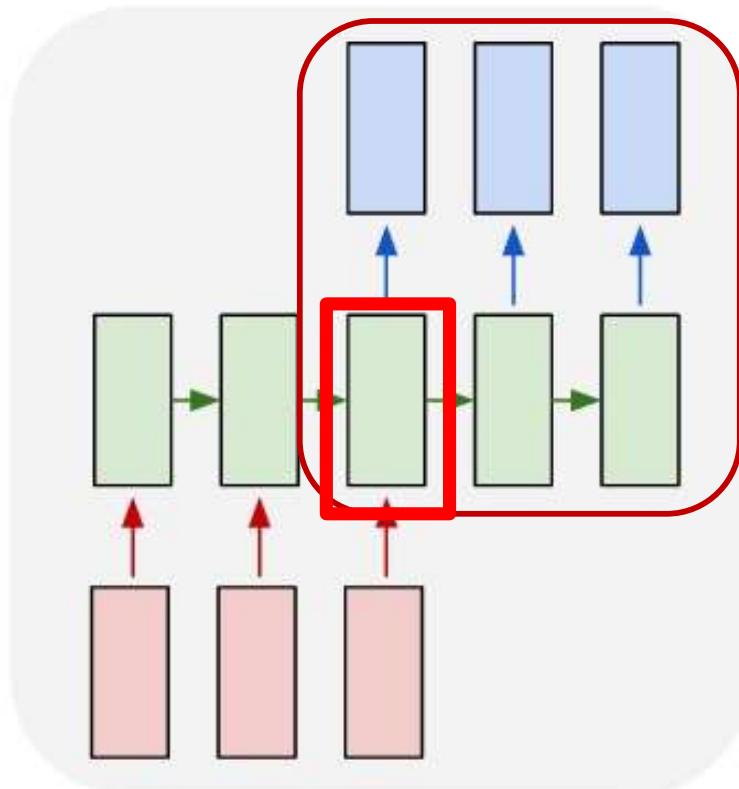
- *Delayed* sequence to sequence

Modelling the problem

many to many

First process the input and generate a hidden representation for it
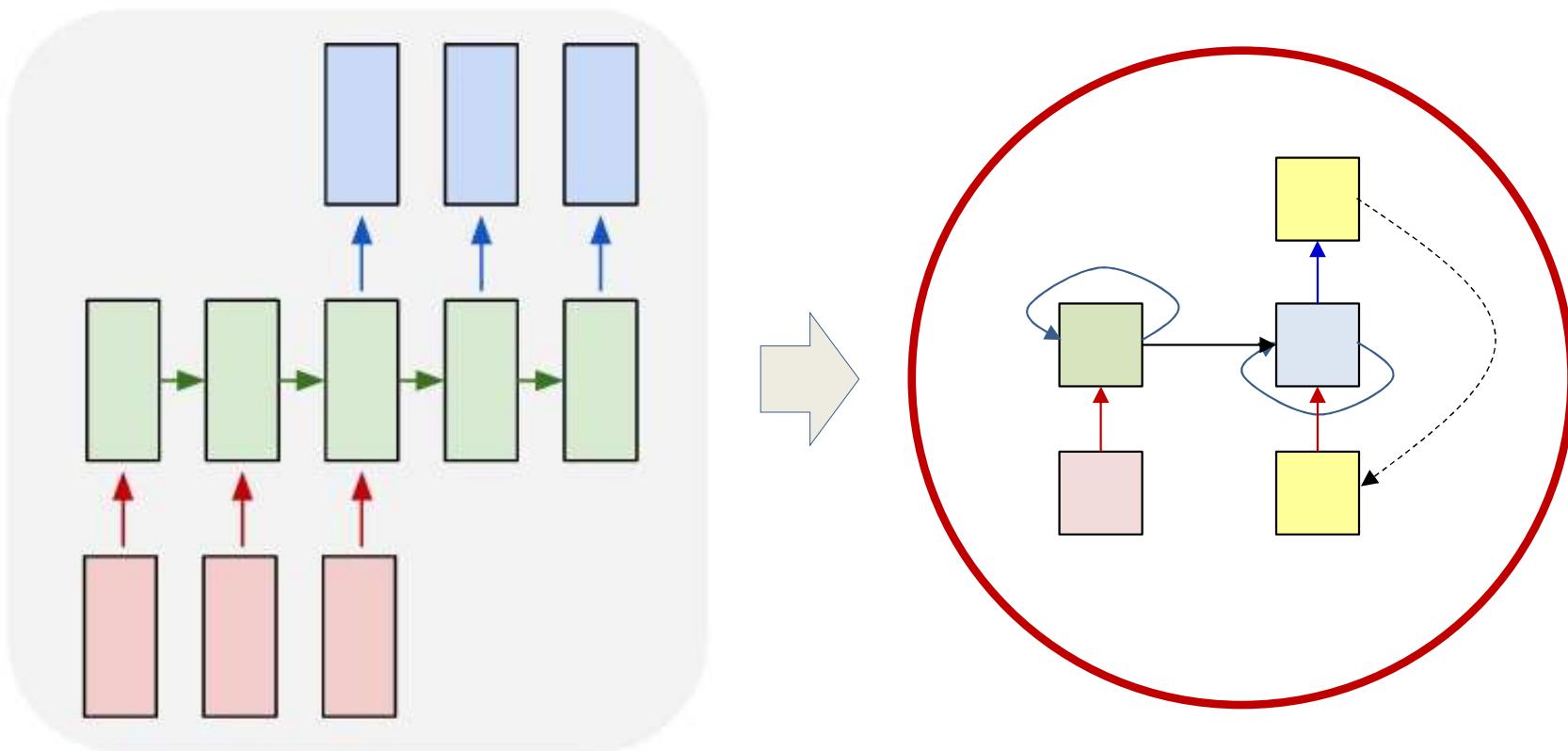
Then use it to generate an output

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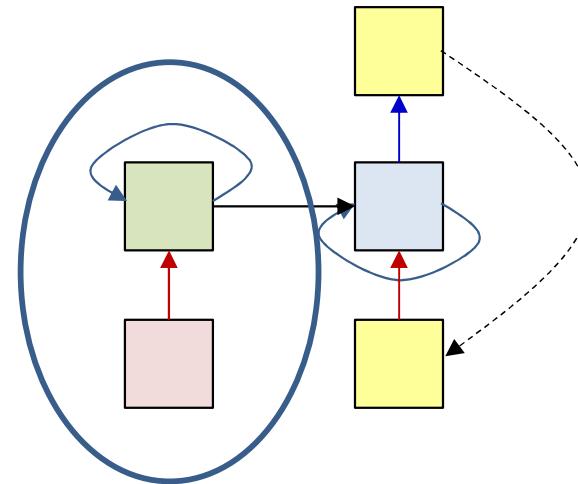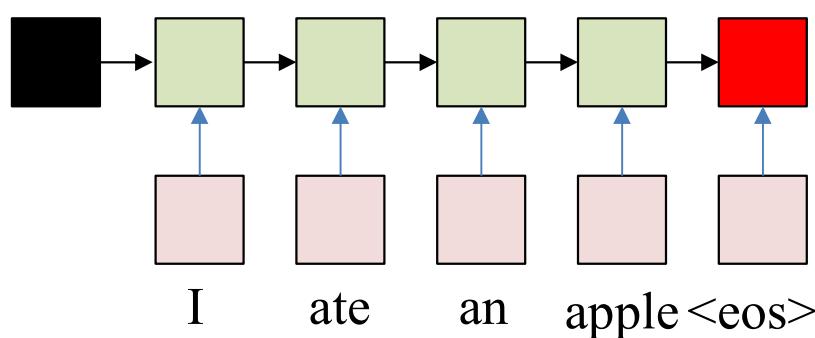
Modelling the problem

many to many
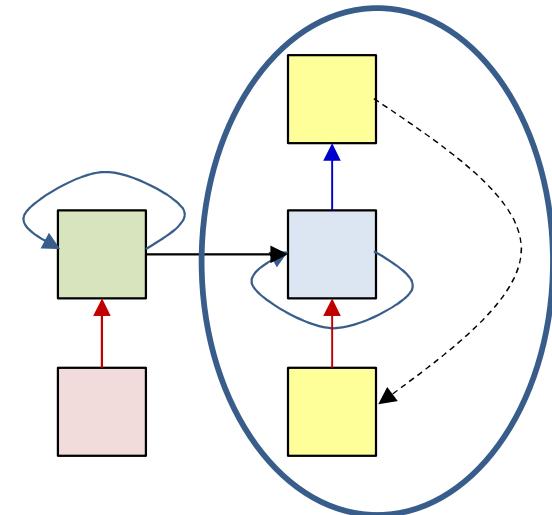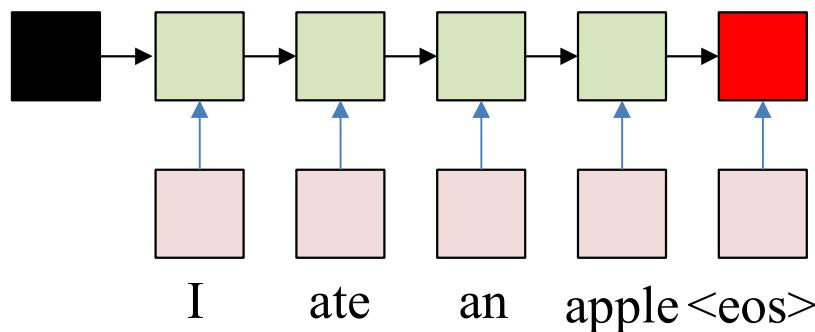
- *Delayed* sequence to sequence
  - Delayed *self-referencing* sequence-to-sequence
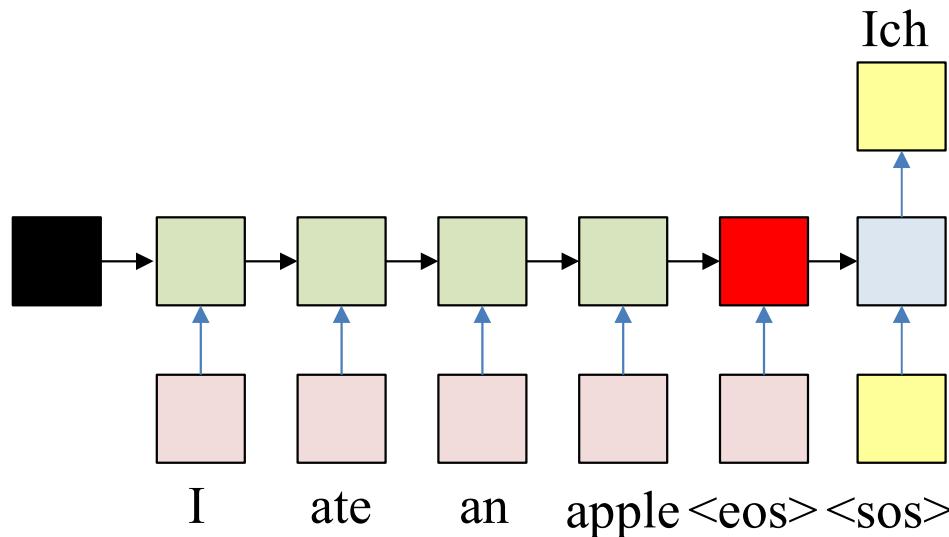
7

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
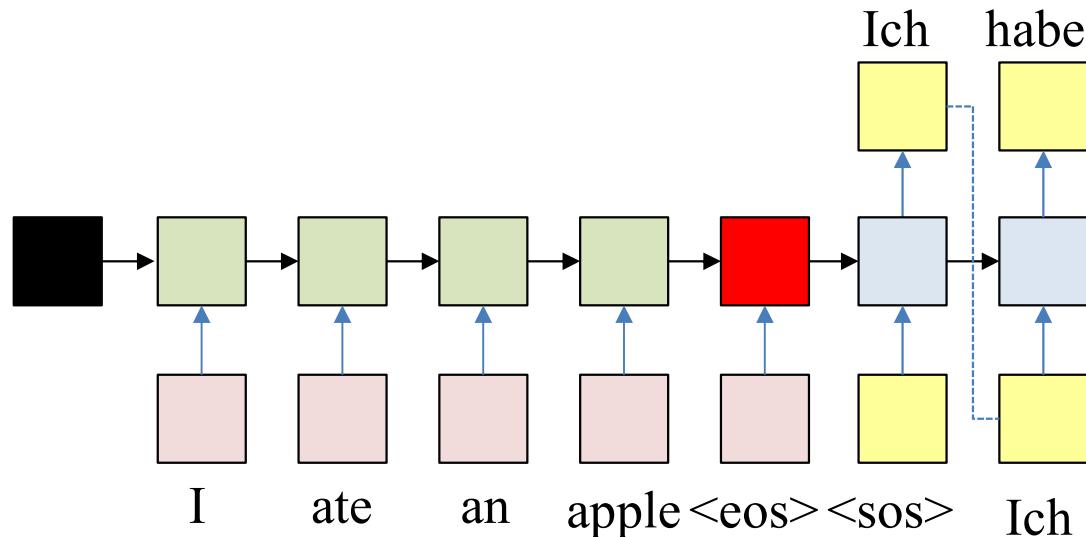
I ate an apple <eos>

- 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

- 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*

A problem with this framework

Ich   habe   einen   apfel   gegessen   <eos>

Y_0 Y_1 Y_2 Y_3 Y_4 Y_5

I   ate   an   apple   <eos>   <sos>   Ich   habe   einen   apfel   gegessen
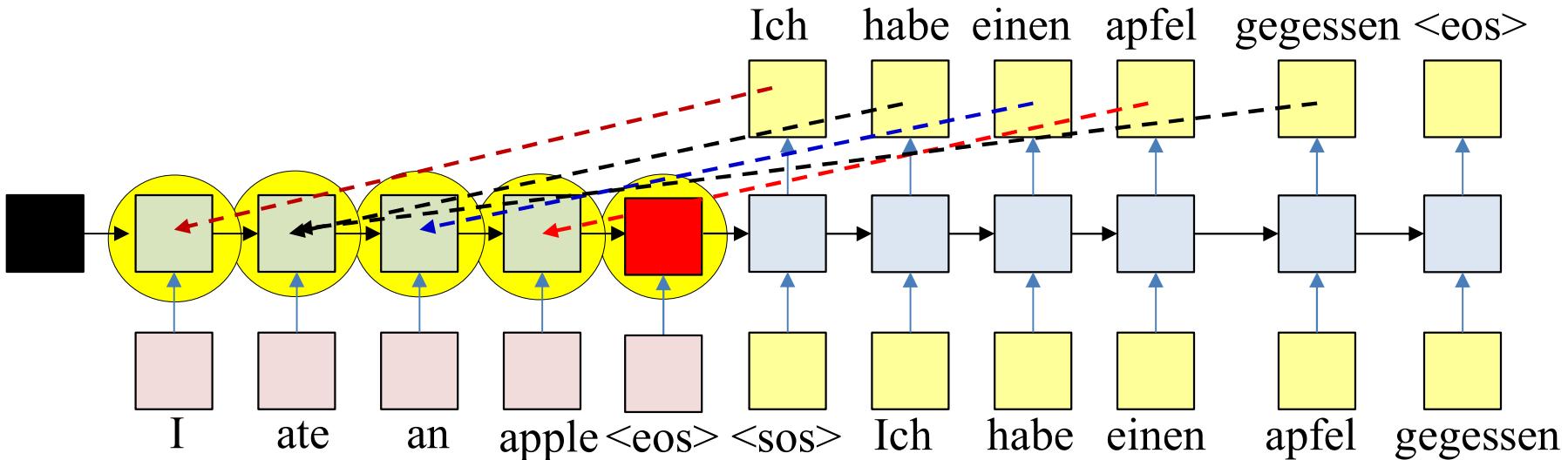
- *All* the information about the input sequence is embedded into a *single* vector
  - The "hidden" node layer at the end of the input sequence
  - This one node is "overloaded" with information
    - Particularly if the input is long

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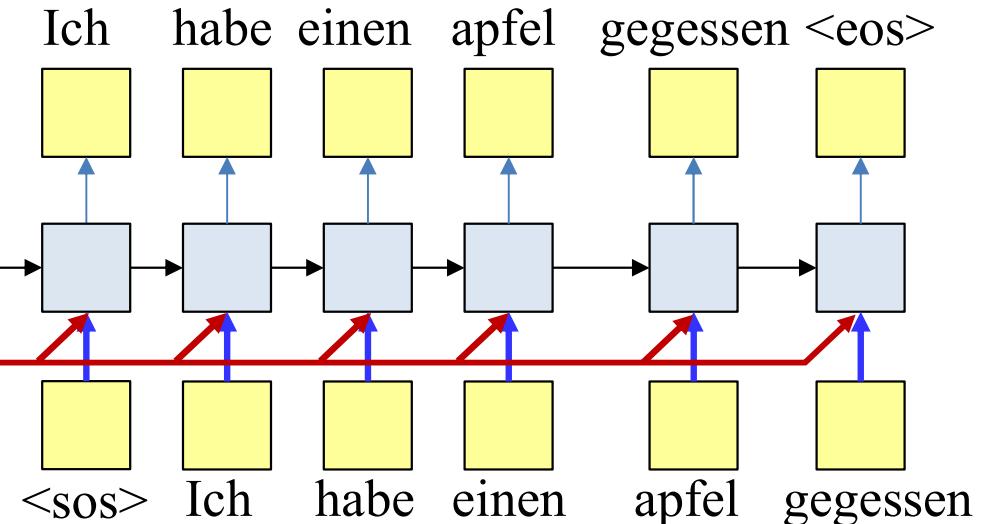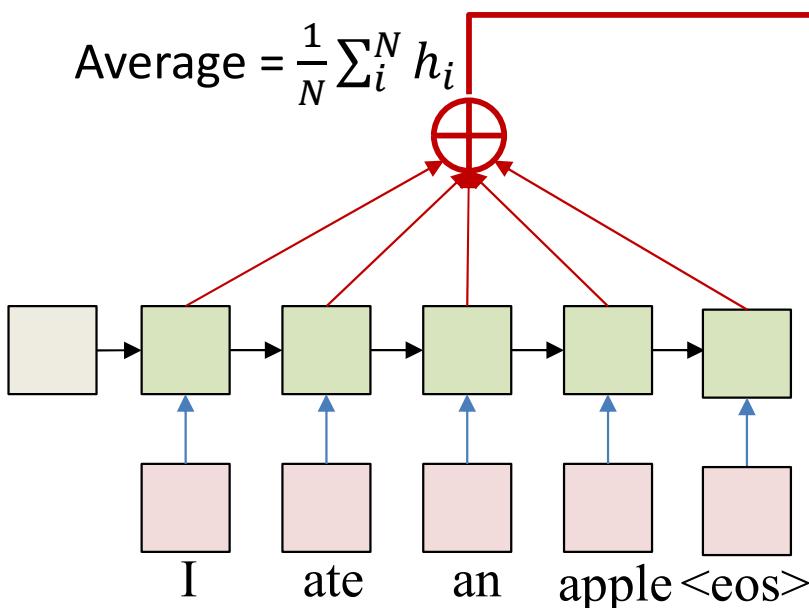
- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder

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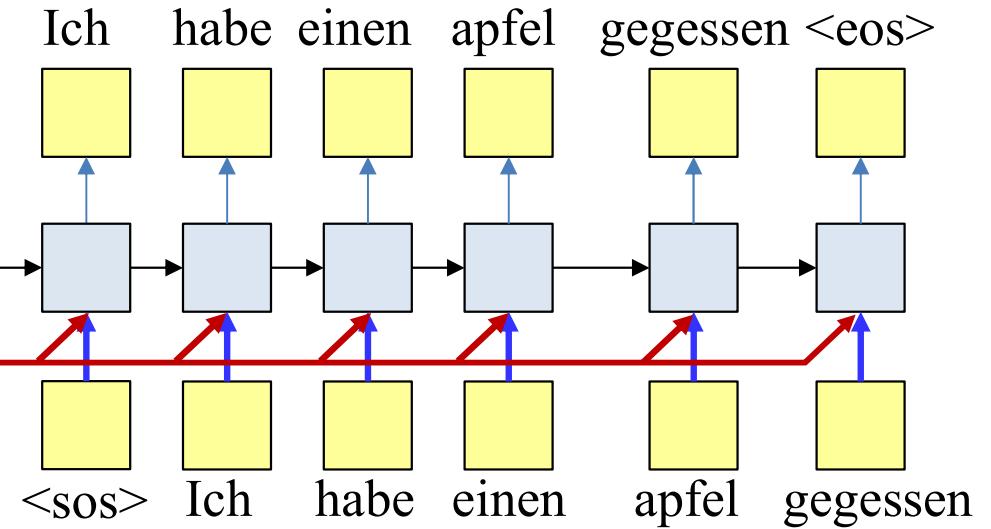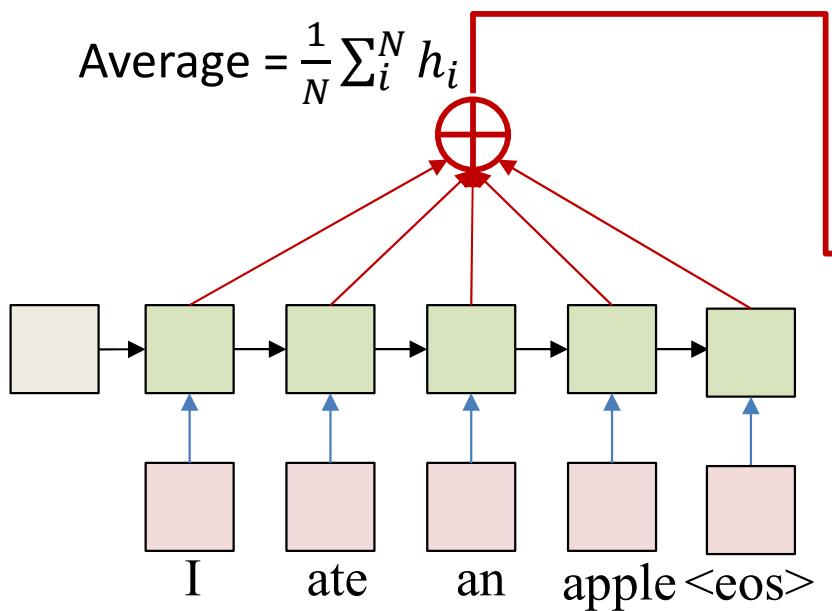
- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder

- *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

Using all input hidden states

Average = $\frac{1}{N}\sum_i^N h_i$
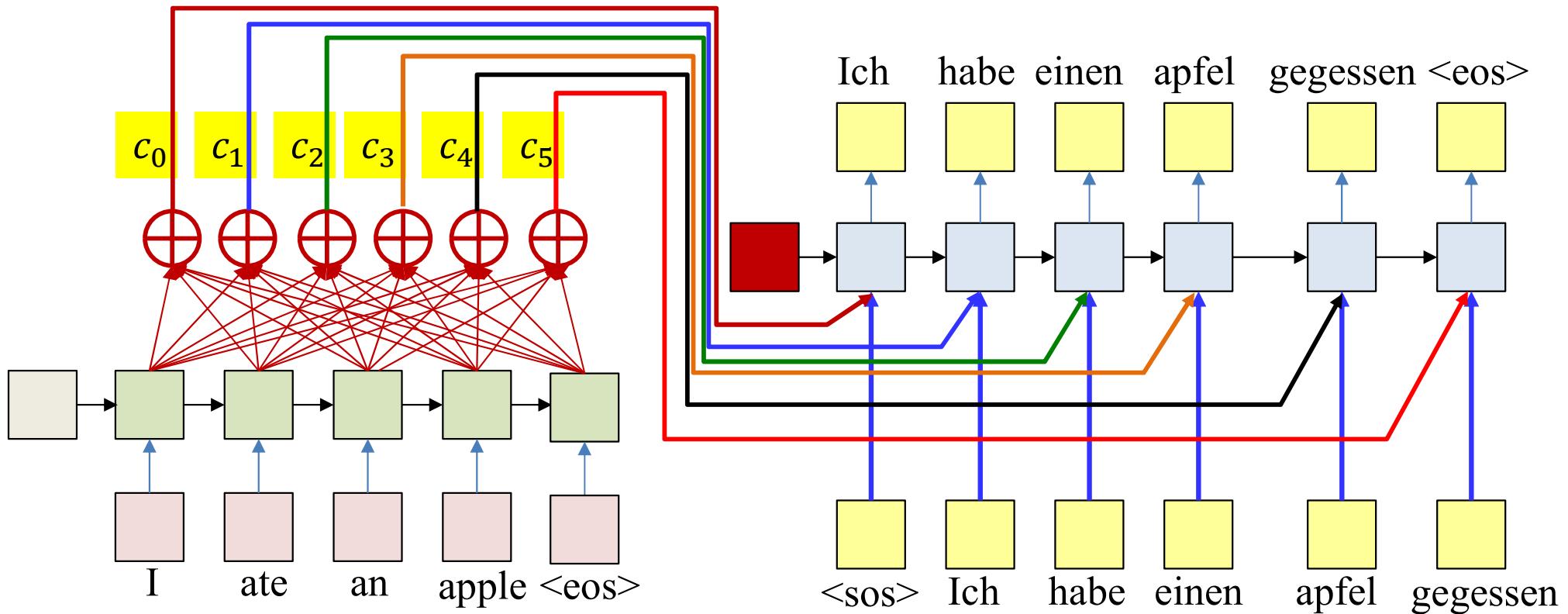
- Simple solution: Compute the average of all encoder hidden states
- Input this average to every stage of the decoder
- The initial decoder hidden state is now separate from the encoder
  - And may be a learnable parameter

Using all input hidden states

Average = $\frac{1}{N}\sum_i^N h_i$

- **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"
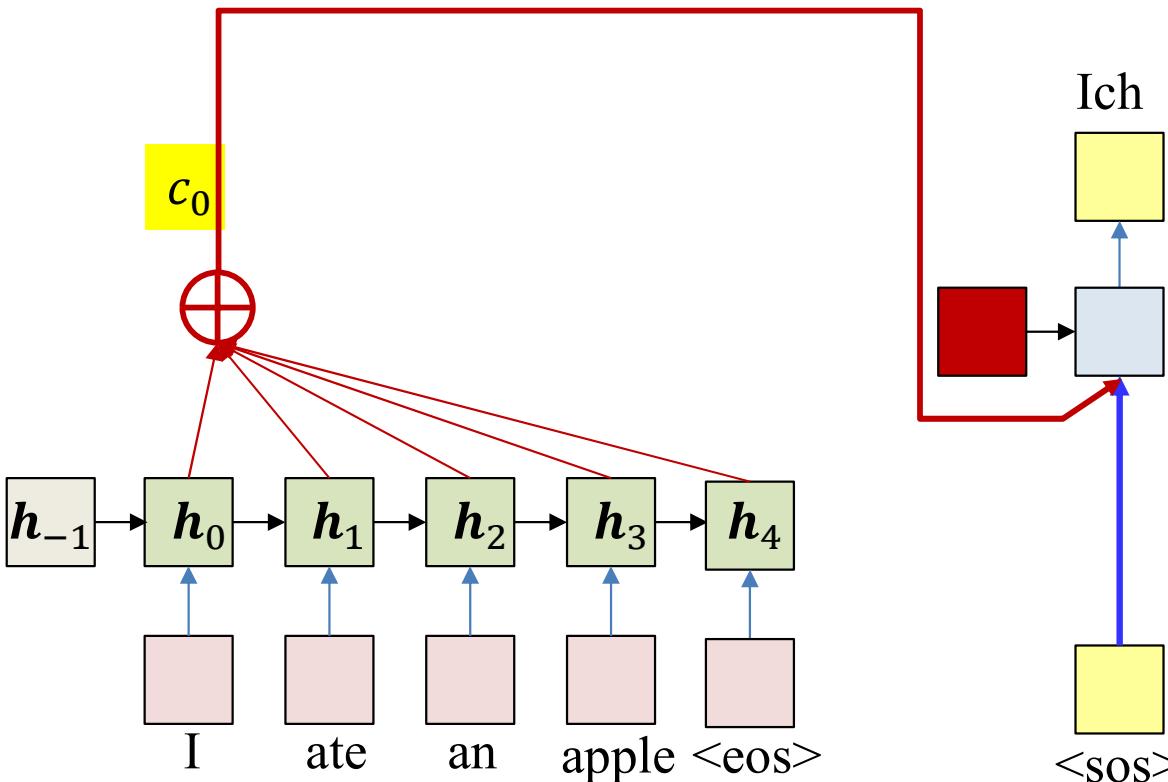
Using all input hidden states

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

$$c_t = \frac{1}{N} \sum_{i}^{N} w_i(t) h_i$$
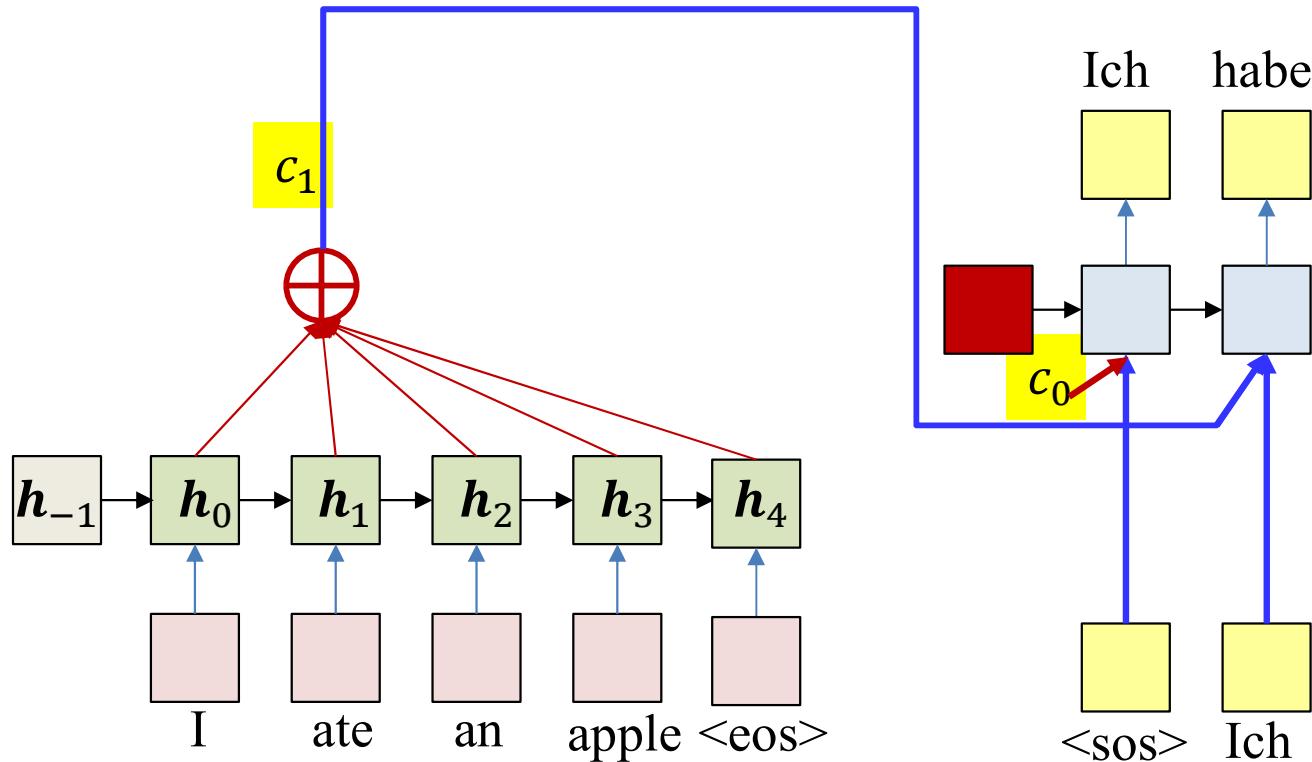
Using all input hidden states

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

$$c_0 = \frac{1}{N} \sum_i^N w_i(0) h_i$$
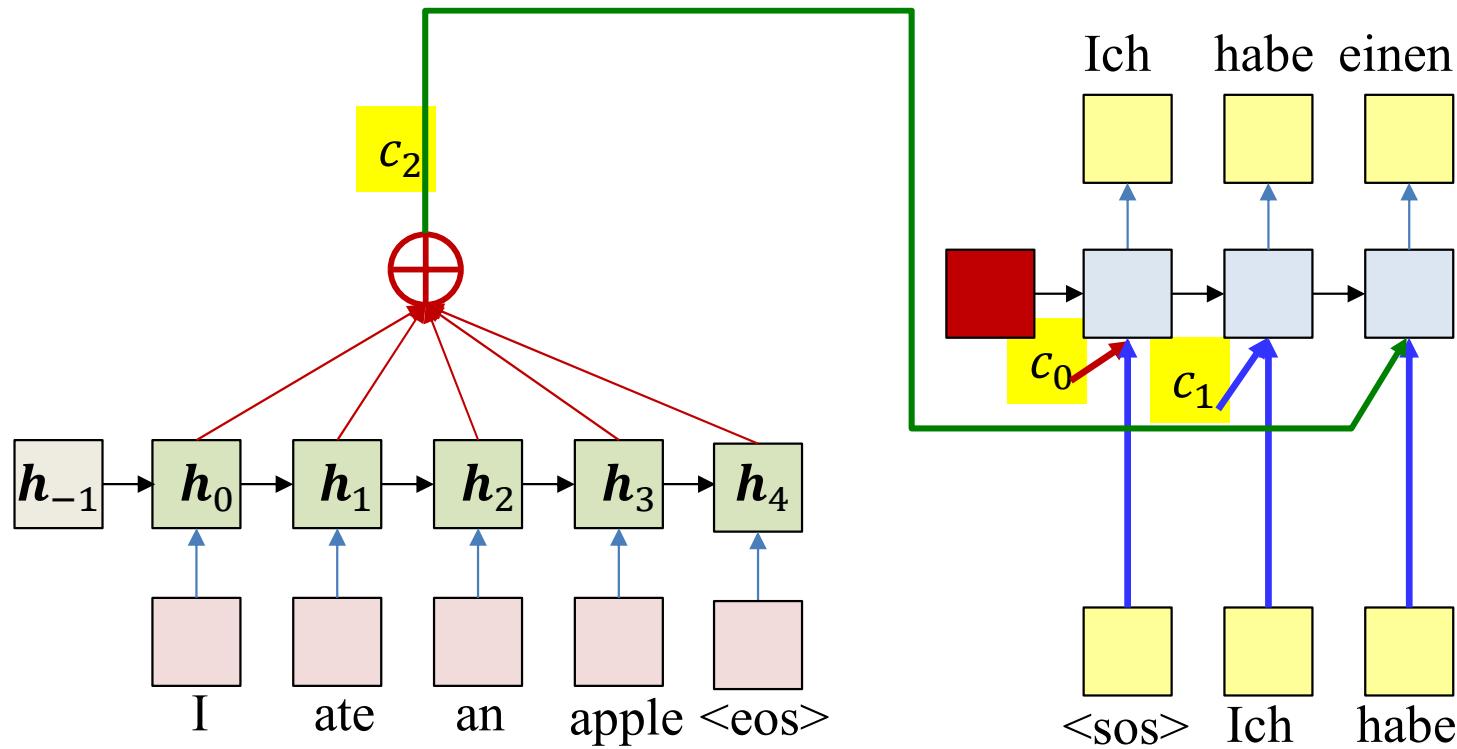
Using all input hidden states

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

$$c_1 = \frac{1}{N}\sum_{i}^{N} w_i(1) h_i$$
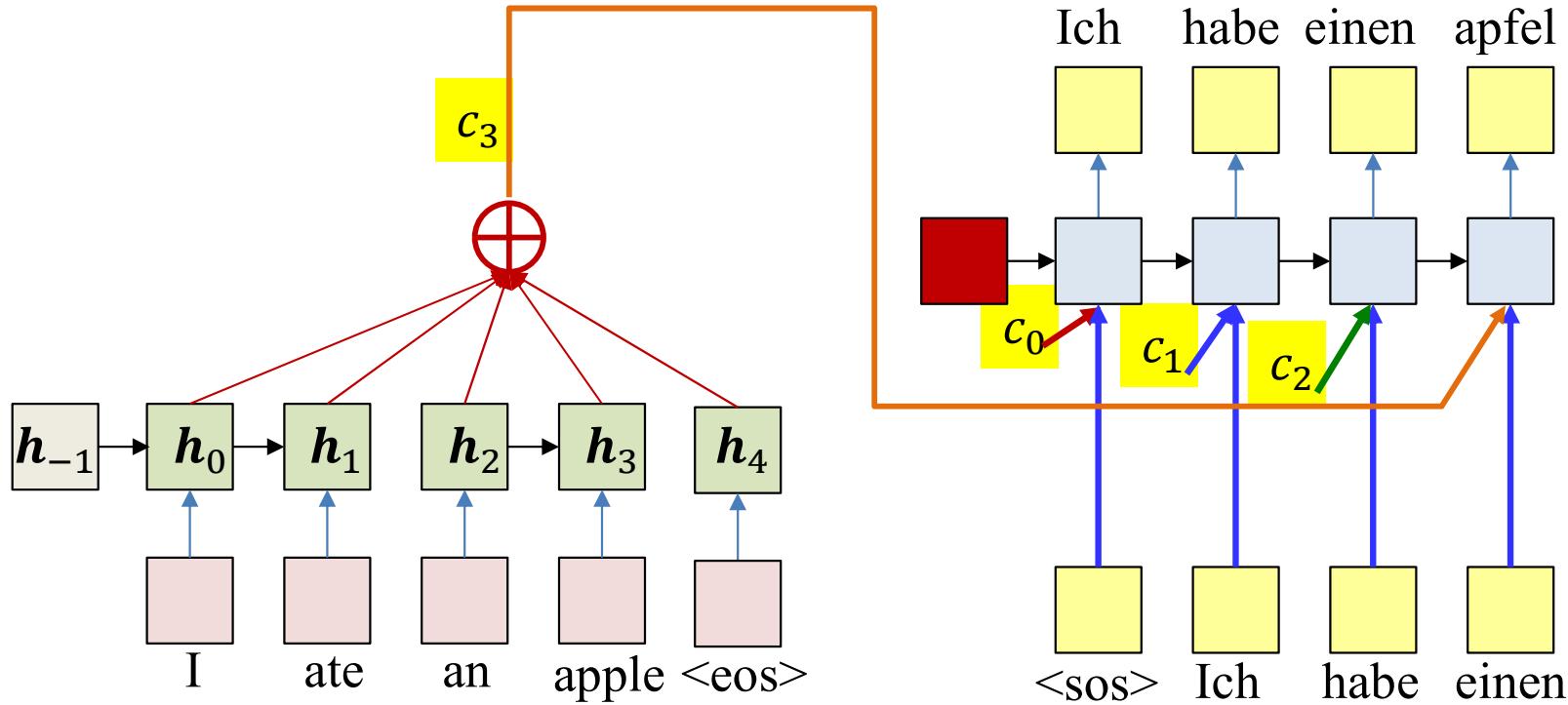
Using all input hidden states

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

$$c_2 = \frac{1}{N} \sum_i^N w_i(2) h_i$$
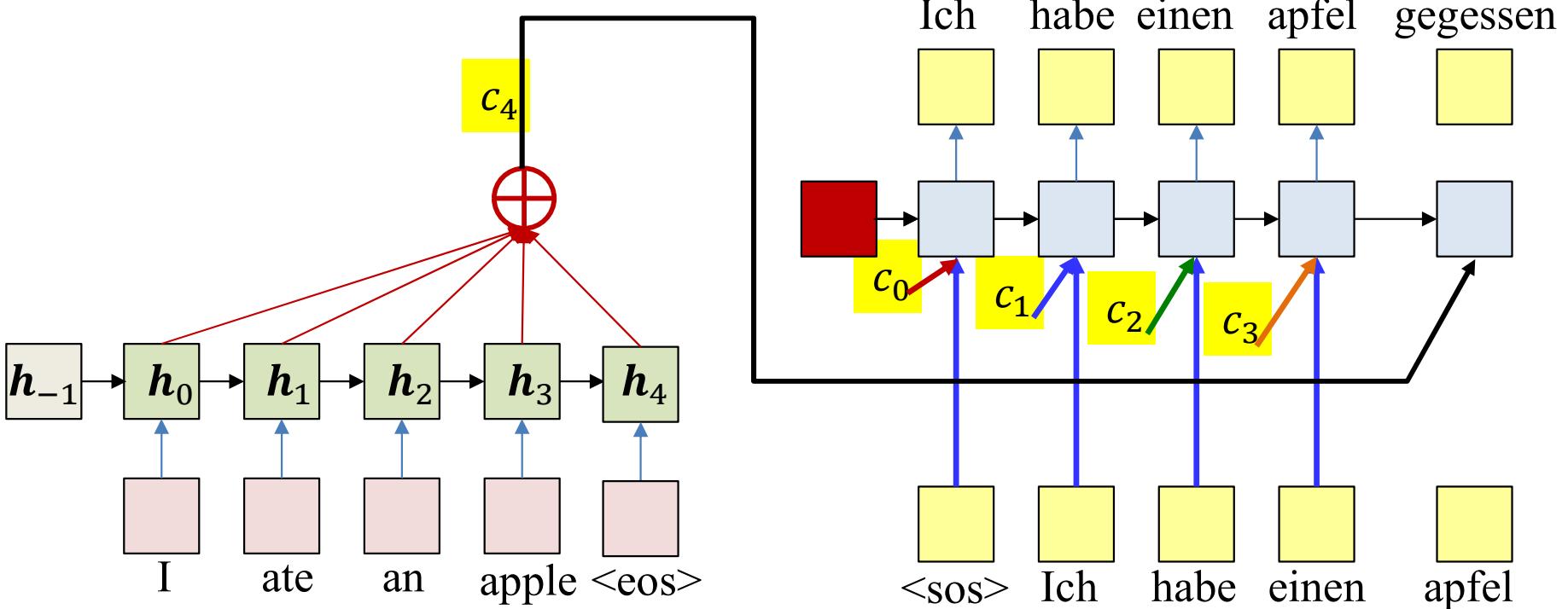
Using all input hidden states

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

$$c_3 = \frac{1}{N} \sum_{i}^{N} w_i(3) h_i$$
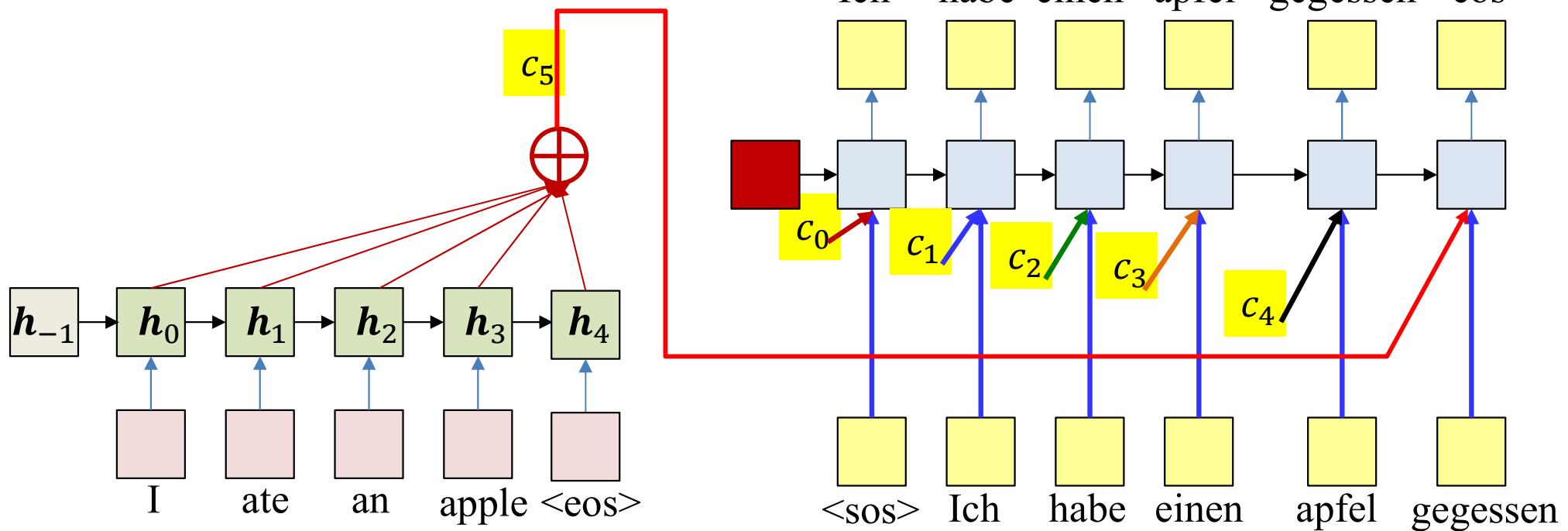
Using all input hidden states

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

$$c_4 = \frac{1}{N} \sum_{i}^{N} w_i(4) h_i$$
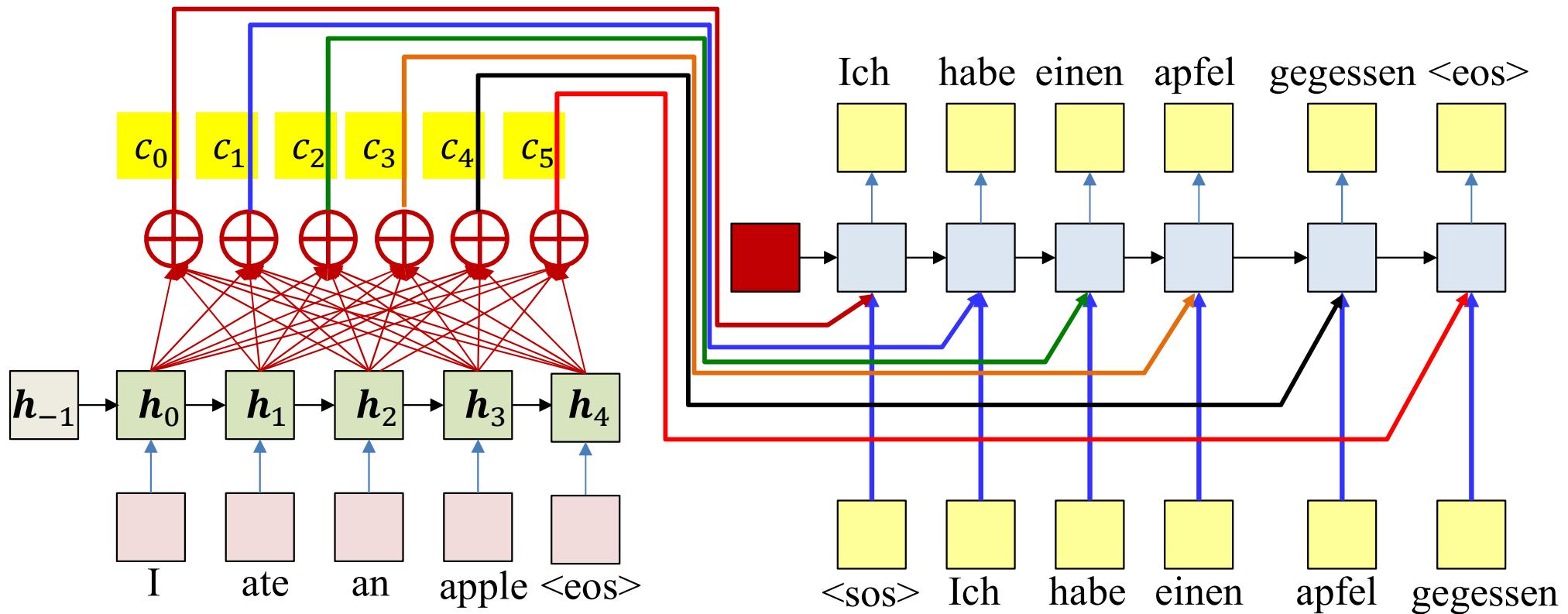
Using all input hidden states

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

$$c_5 = \frac{1}{N} \sum_{i}^{N} w_i(5) h_i$$
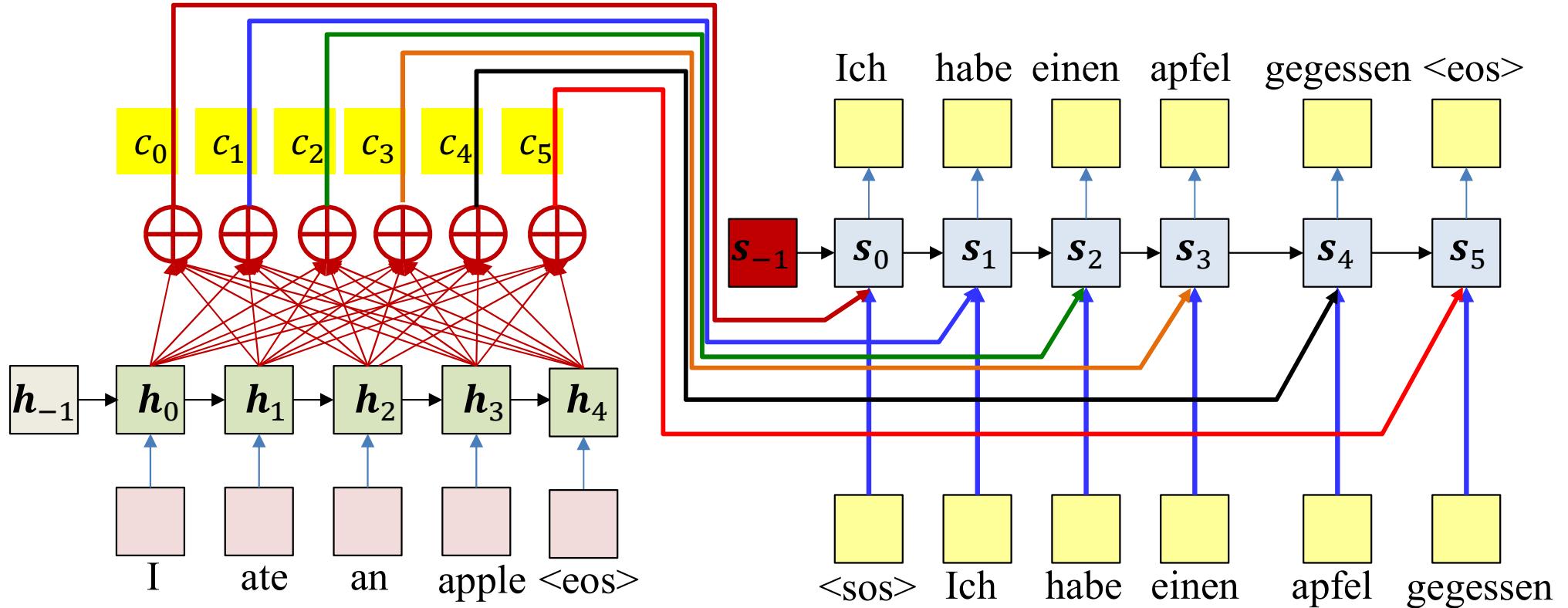
Using all input hidden states

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

- 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??
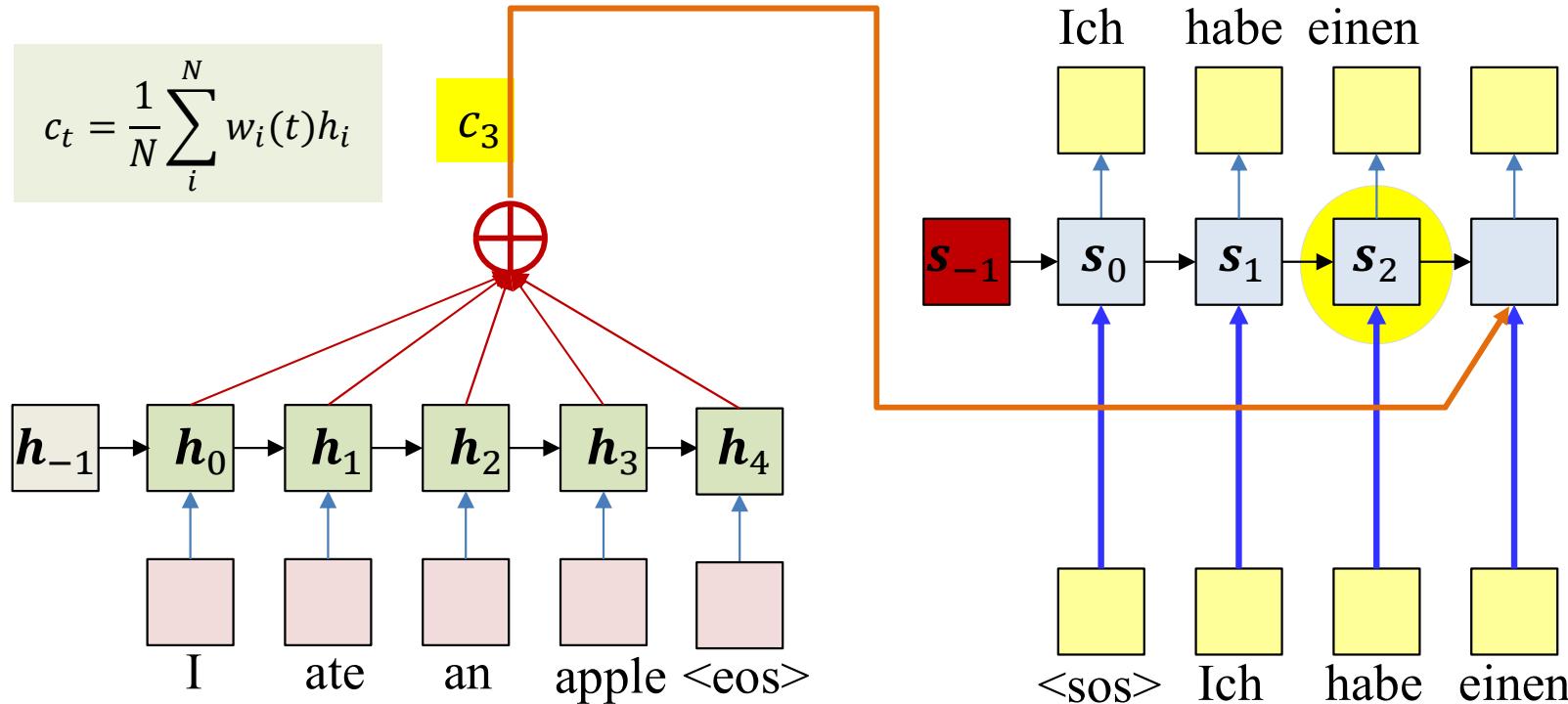
Attention Models

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

- **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?
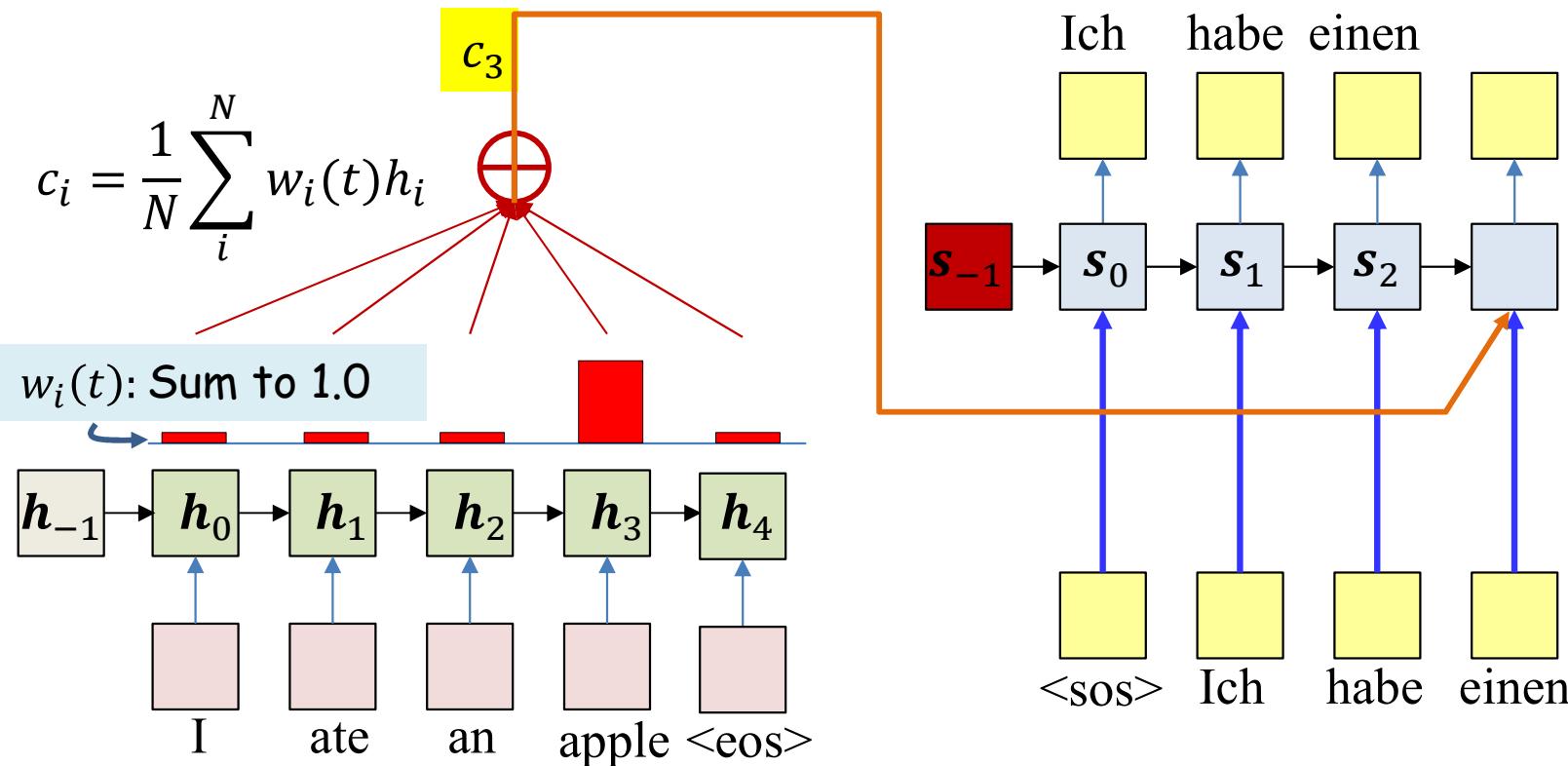
Attention weights at time t

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

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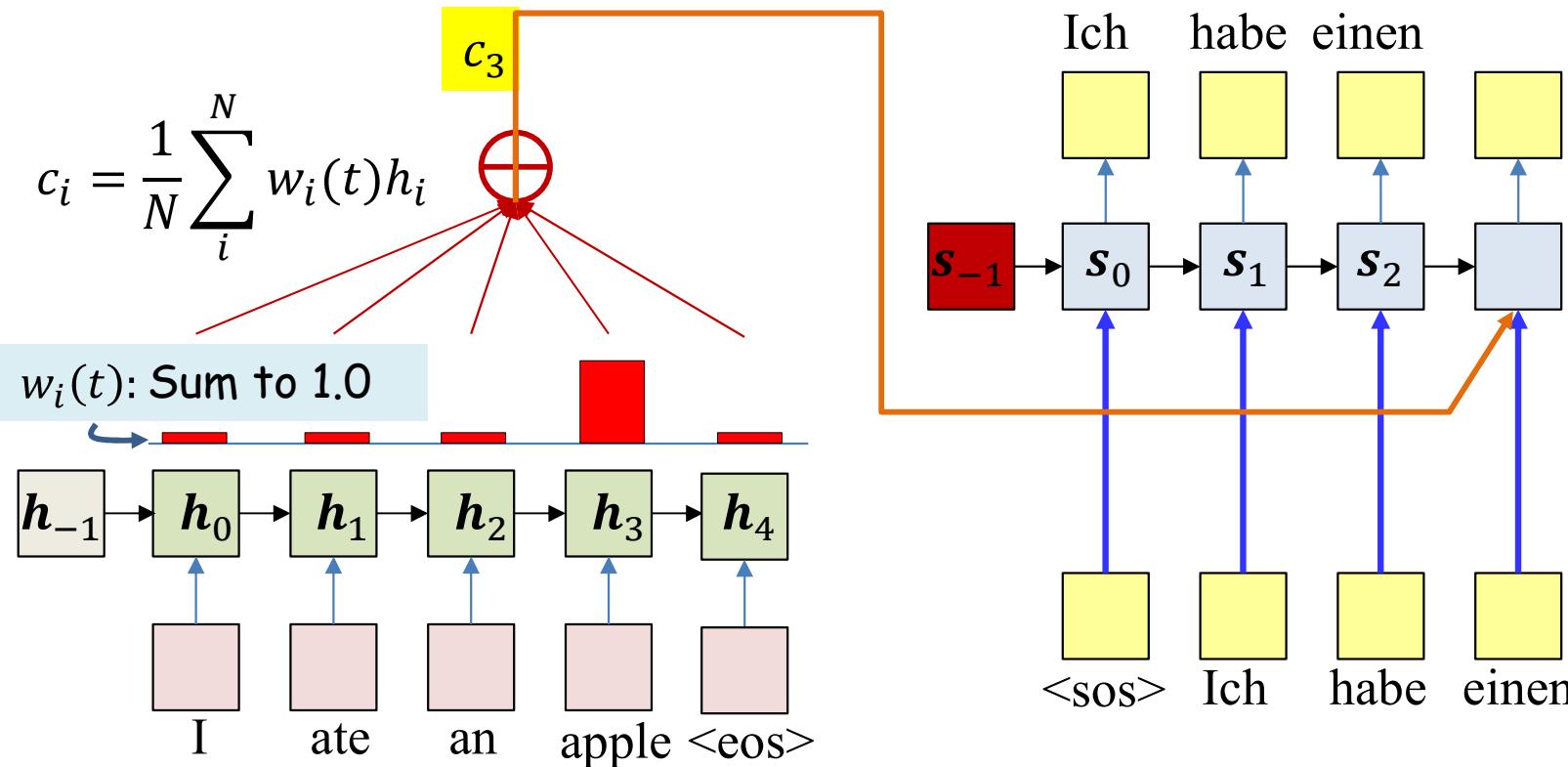
$$w_i(t) = a(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

Requirement on attention weights

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

$w_i(t)$: Sum to 1.0

- 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

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

$w_i(t)$: Sum to 1.0

- 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

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

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

Poll 1 (@1485,@1486,@1487)

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

Poll 1

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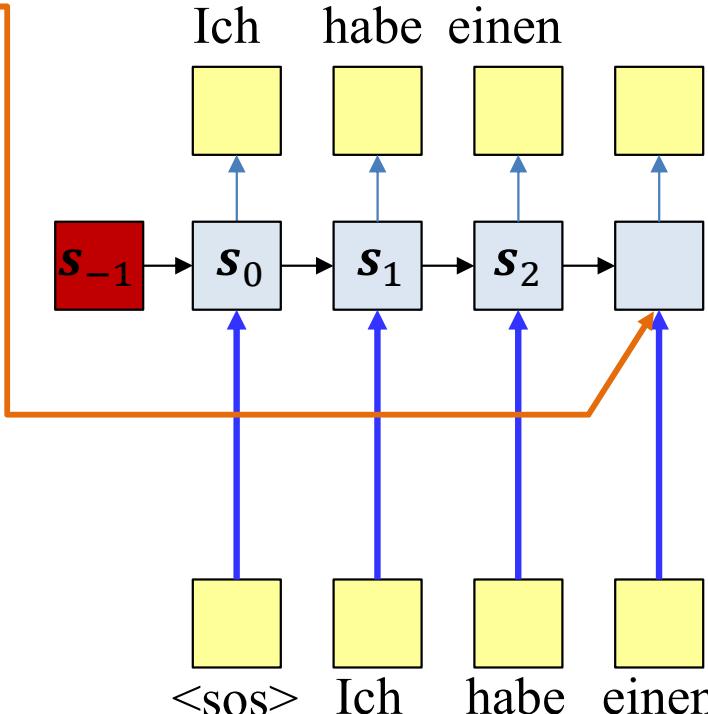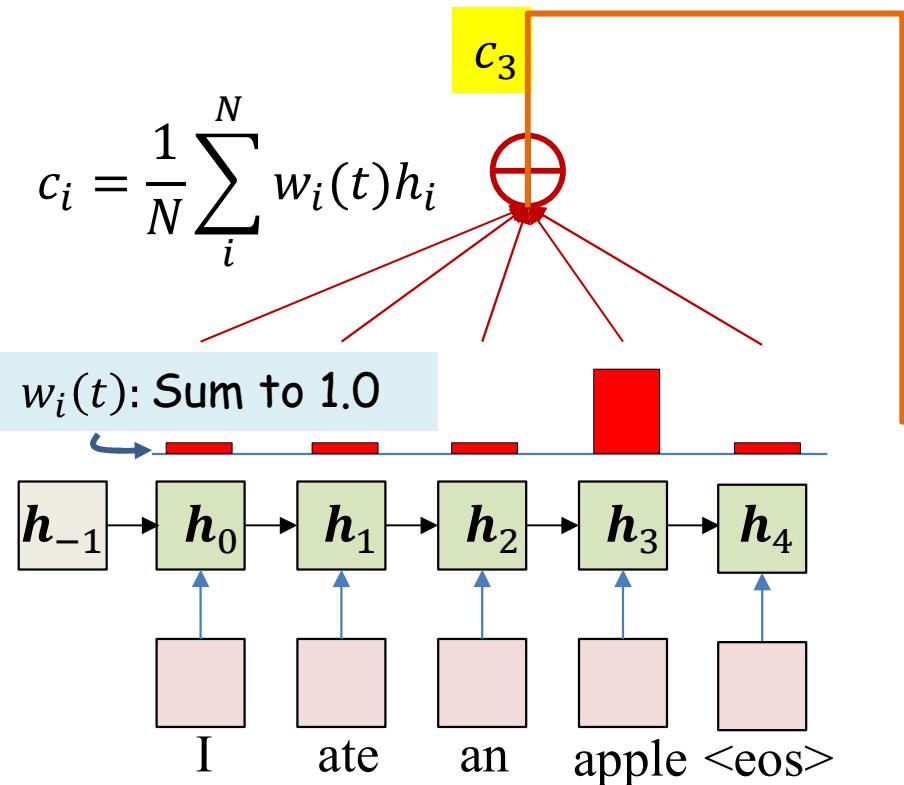
- **True**
- False

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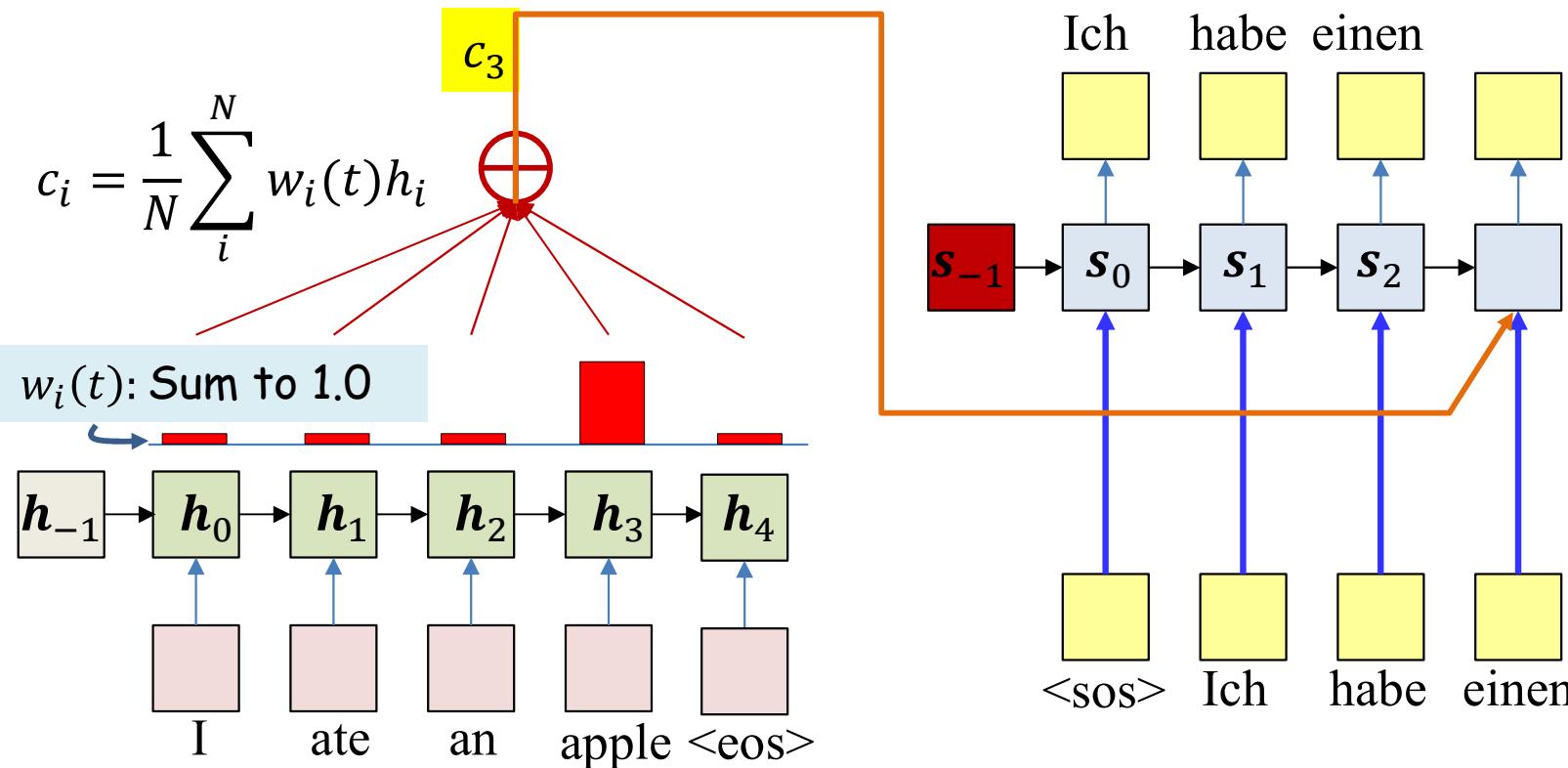
$$c_i = \frac{1}{N}\sum_i^N w_i(t)h_i$$

$w_i(t)$: Sum to 1.0

Ich habe einen

c_3

s_{-1} → s_0 → s_1 → s_2 →

h_{-1} → h_0 → h_1 → h_2 → h_3 → h_4

I ate an apple \<eos\>

\<sos\> Ich habe einen

- The weights $w_i(t)$ must be positive and sum to
  - 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

What is this function?

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

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

Attention weights

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

$w_i(t)$: Sum to 1.0

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

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

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

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

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

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

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

35

Attention weights

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

c_3

$w_i(t)$: Sum to 1.0

$h_{-1} \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4$

Ich   habe einen

$s_{-1} \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow$

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

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

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

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

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

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

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

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

36

Converting an input: Inference

$$\boxed{h_{-1}} \rightarrow \boxed{h_0} \rightarrow \boxed{h_1} \rightarrow \boxed{h_2} \rightarrow \boxed{h_3} \rightarrow \boxed{h_4}$$

| | I | ate | an | apple | \<eos\> |
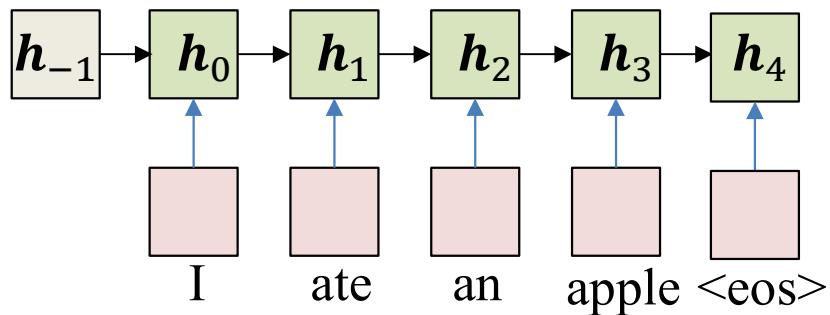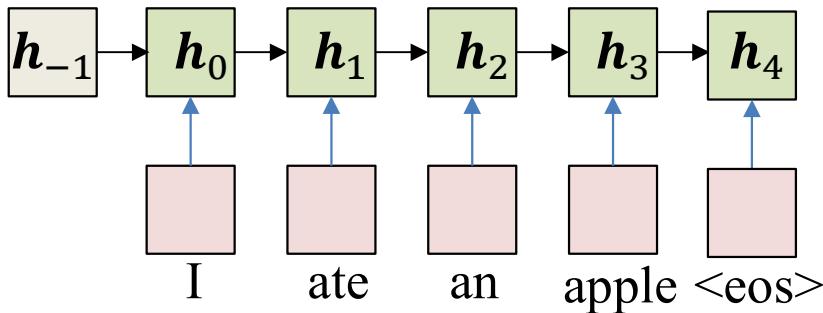
- Pass the input through the encoder to produce hidden representations h_i

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 $W_{init}\ mean(h_0, \dots, h_4)$ where W_{init} is a learned parameter
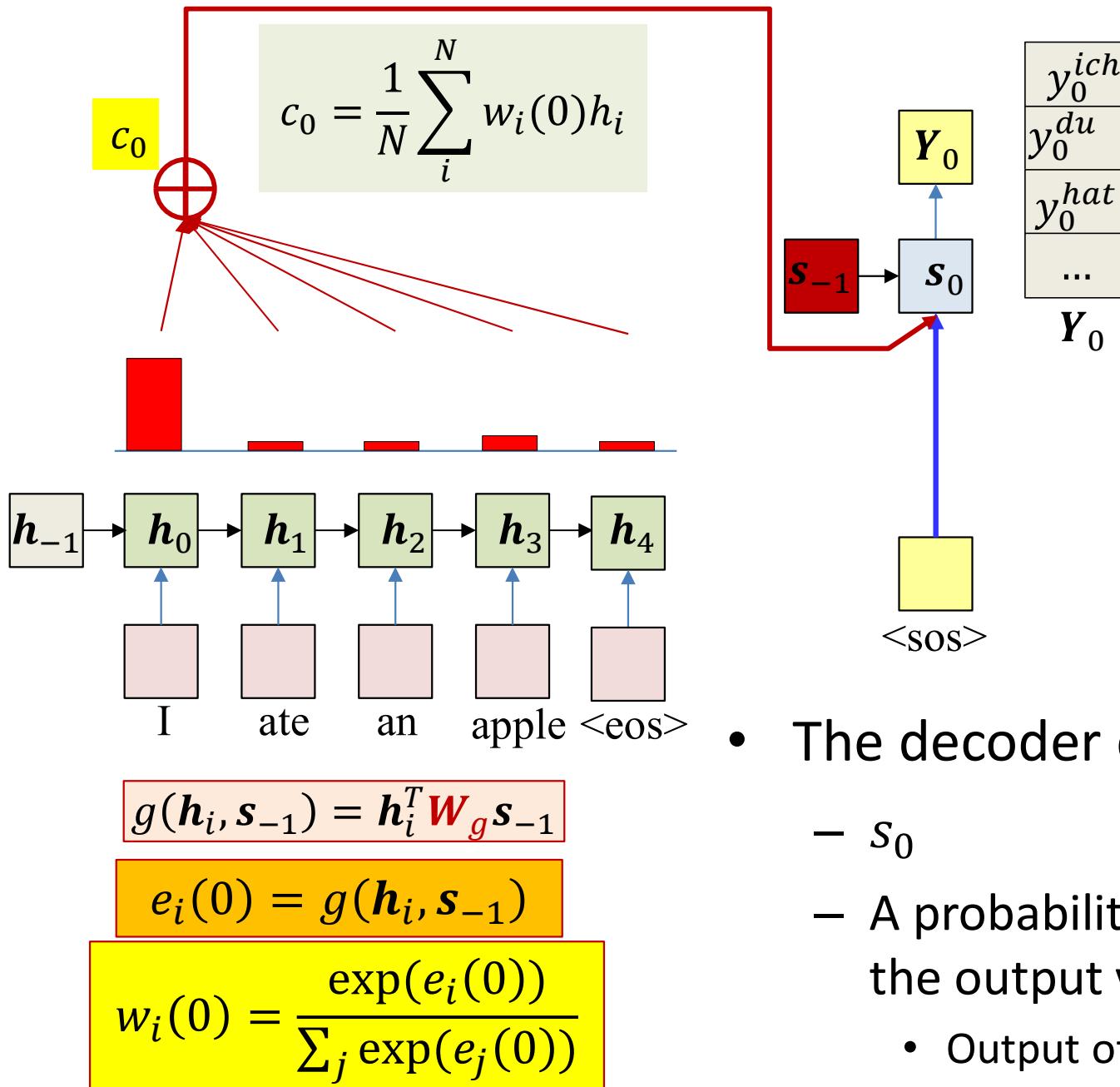
s_{-1}

$h_{-1} \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4$

I   ate   an   apple   \<eos\>

- Pass the input through the encoder to produce hidden representations h_i

Converting an input: Inference

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

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

$$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$$

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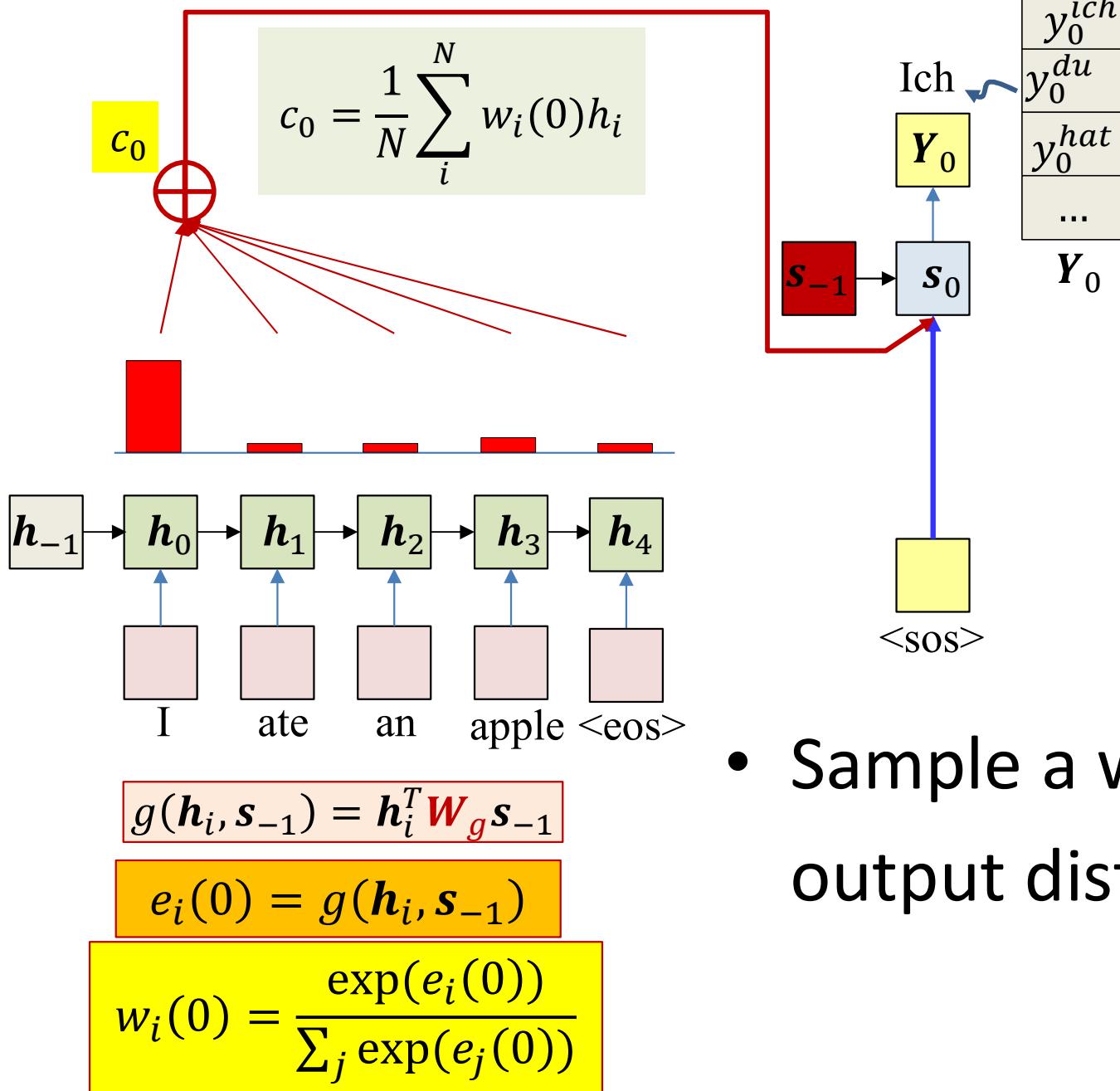
- Compute the attention weights $w_i(0)$ for the first output using s_{-1}
  - Will be a distribution over the input words
- Compute "context" c_0
  - Weighted sum of input word hidden states
- Input c_0 and <sos> to the decoder at time 0
  - <sos> because we are starting a new sequence
  - In practice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

39

Converting an input: Inference

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

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

$$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$$

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$
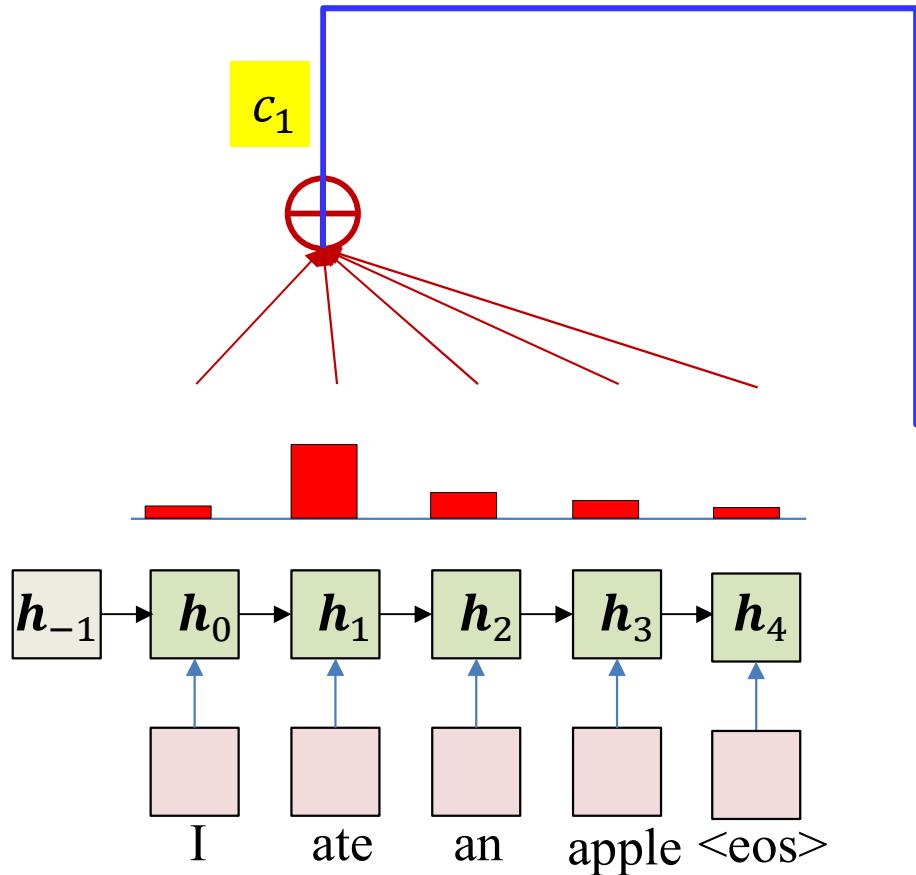
- The decoder computes
  - s_0
  - A probability distribution over the output vocabulary
    - Output of softmax output layer

Converting an input: Inference

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

c_0

Ich

Y_0

$\begin{array}{|c|}\hline y_0^{ich} \\ y_0^{du} \\ y_0^{hat} \\ \dots \\ \hline \end{array}$

Y_0

s_{-1} → s_0

h_{-1} → h_0 → h_1 → h_2 → h_3 → h_4

I   ate   an   apple <eos>

<sos>

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

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

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$
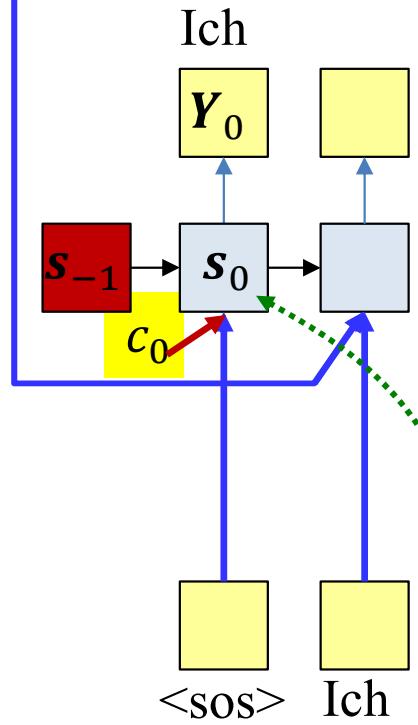
- Sample a word from the output distribution

Converting an input: Inference

c_1

Ich

Y_0

$s_{-1} \rightarrow s_0 \rightarrow$

c_0

$h_{-1} \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4$

I   ate   an   apple <eos>

<sos>   Ich

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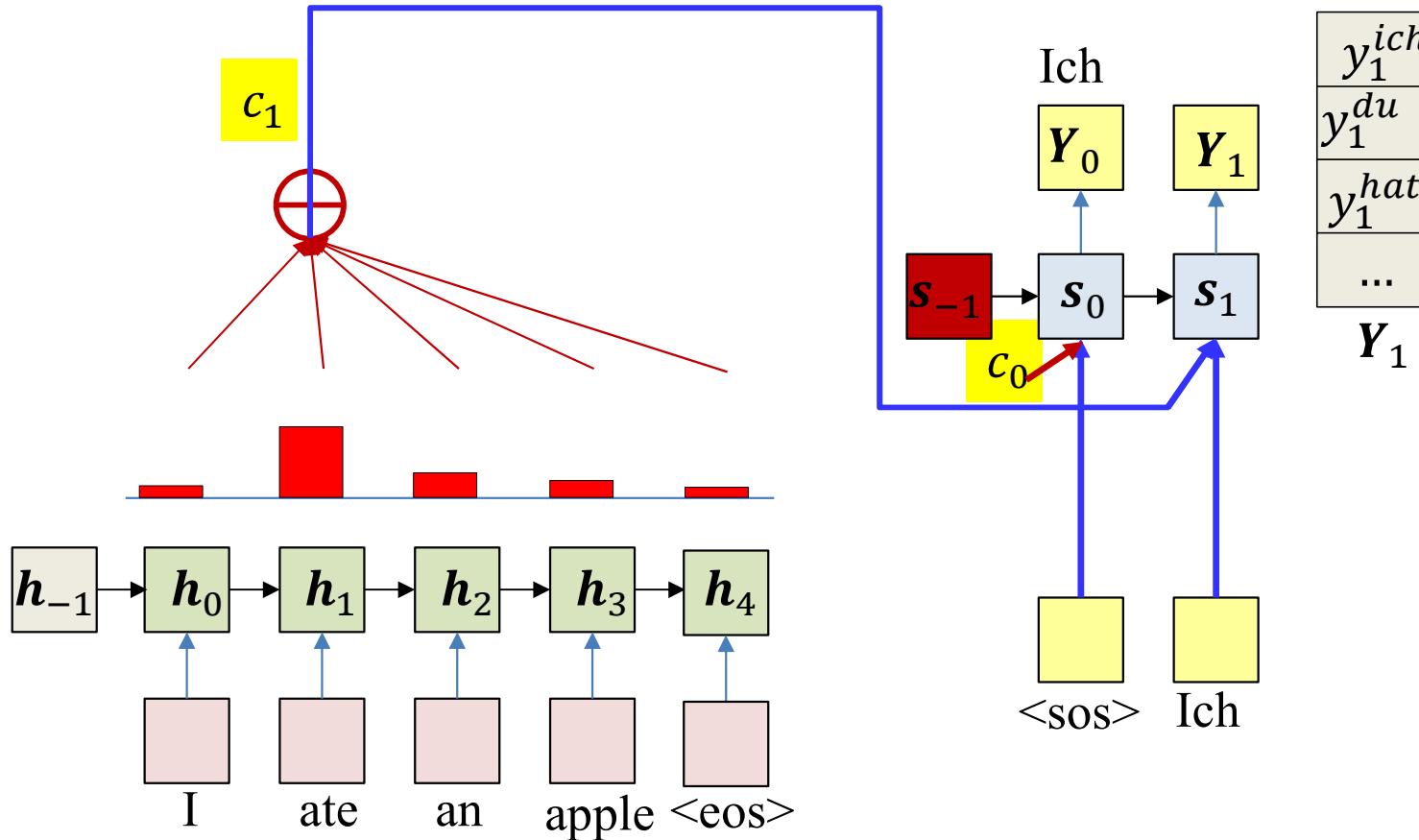
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

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

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

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

Converting an input: Inference

- The decoder computes
  - s_1
  - A probability distribution over the output vocabulary
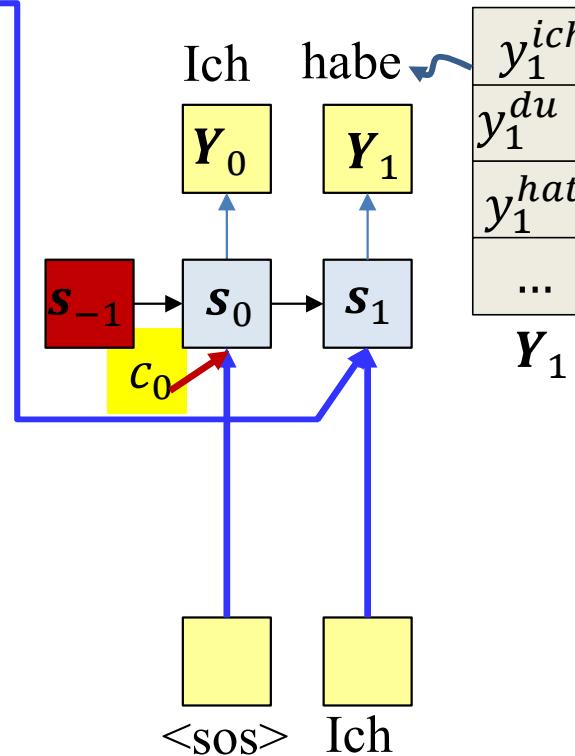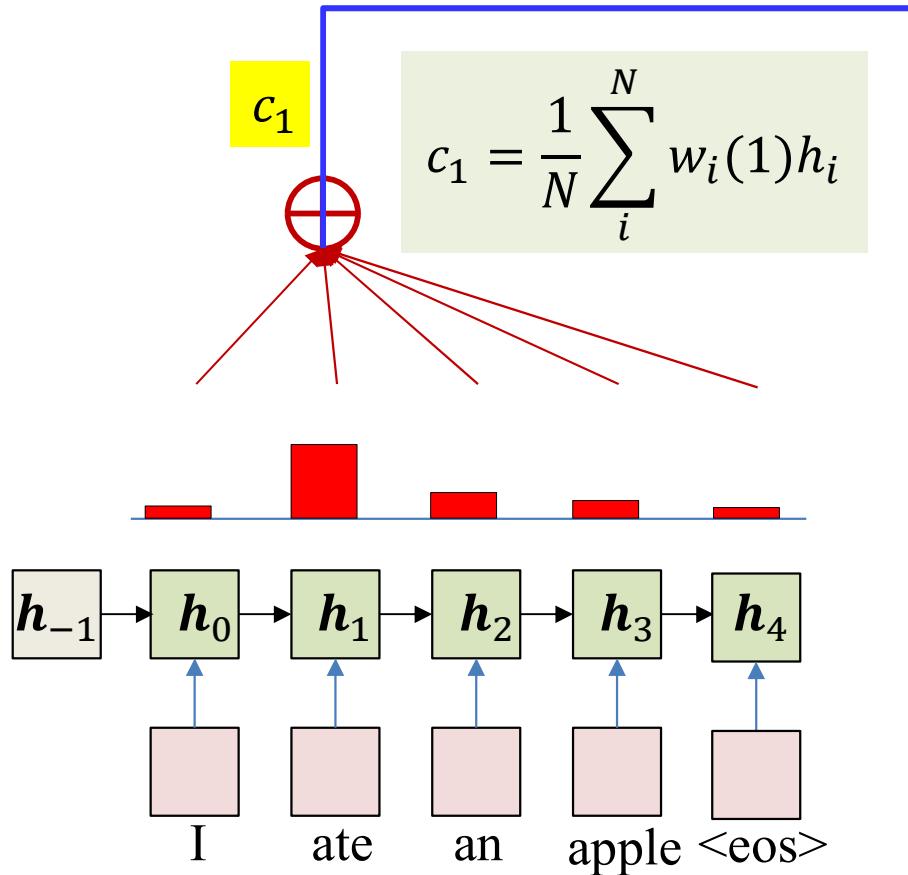
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

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

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

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

Converting an input: Inference

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

c_1

Ich   habe

Y_0 Y_1

$s_{-1} \rightarrow s_0 \rightarrow s_1$

c_0

y_1^{ich}
y_1^{du}
y_1^{hat}
...
Y_1

$h_{-1} \rightarrow h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4$

I   ate   an   apple   <eos>

<sos>   Ich

- Sample the second word from the output distribution

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

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

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

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

44

Converting an input: Inference

c_2

Ich   habe

Y_0 Y_1

s_{-1} → s_0 → s_1 →

c_0 c_1

h_{-1} → h_0 → h_1 → h_2 → h_3 → h_4

I   ate   an   apple   <eos>

<sos>   Ich   habe

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

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

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

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

45

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

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

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

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

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

Converting an input: Inference

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

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

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

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

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

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

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

Converting an input: Inference

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

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

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

Converting an input: Inference

$$e_i(4) = g(h_i, s_3)$$

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

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

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

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

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

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

54

Converting an input: Inference

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

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

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

Converting an input: Inference
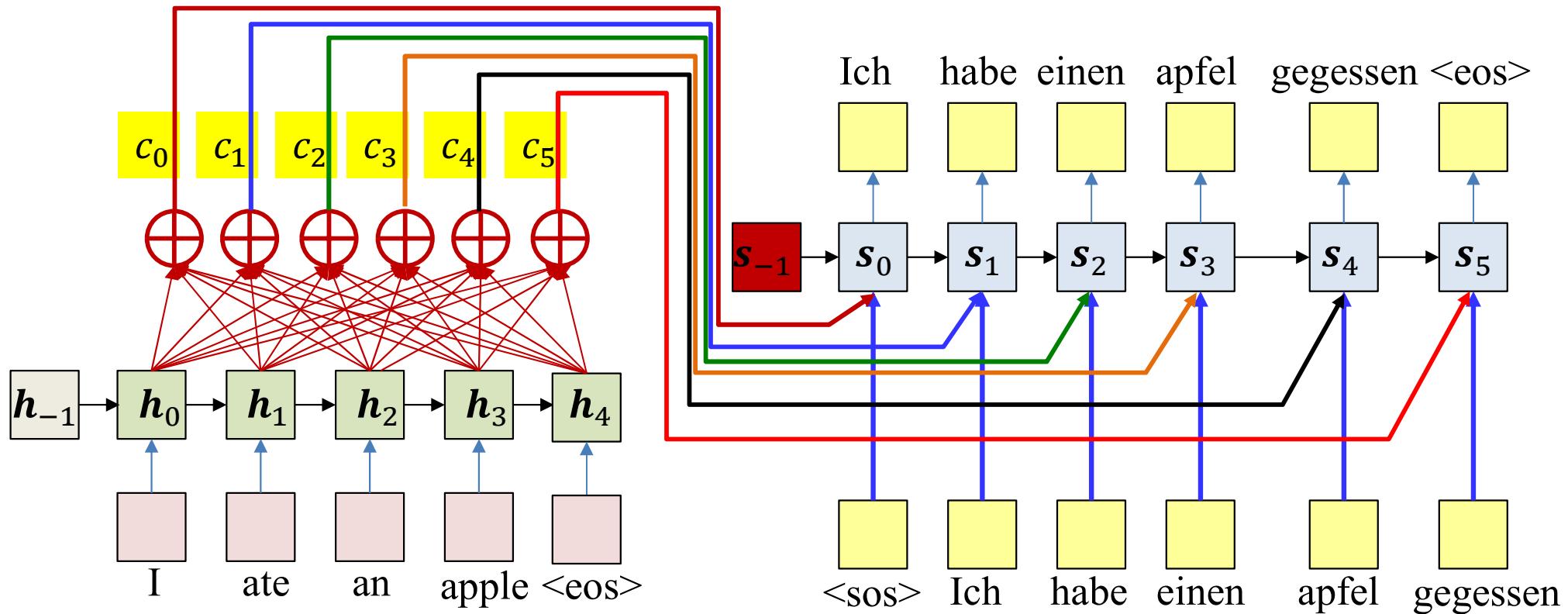
Continue this process until `<eos>` is drawn

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

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

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

56

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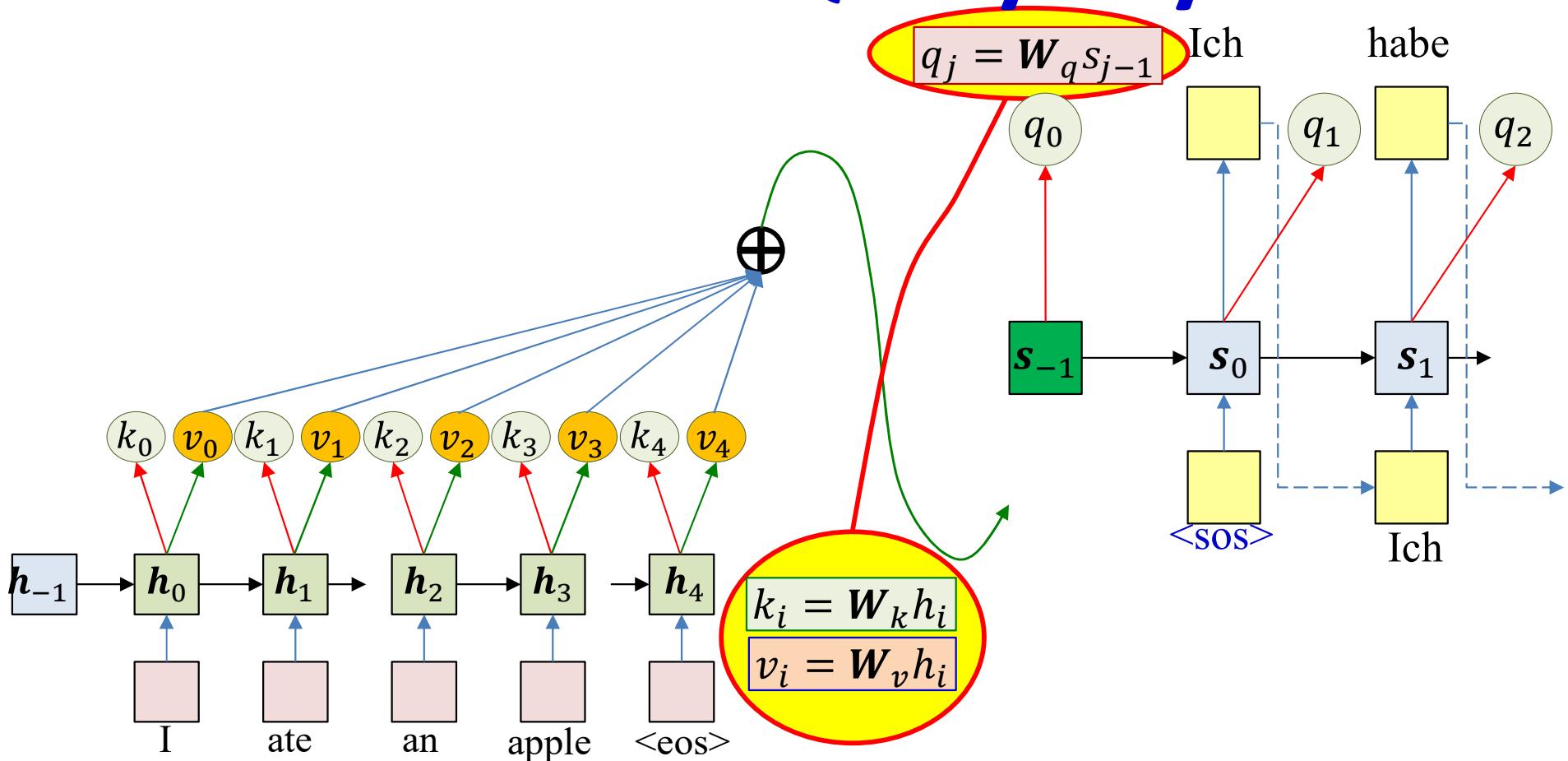
$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

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

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

Modification: Query key value
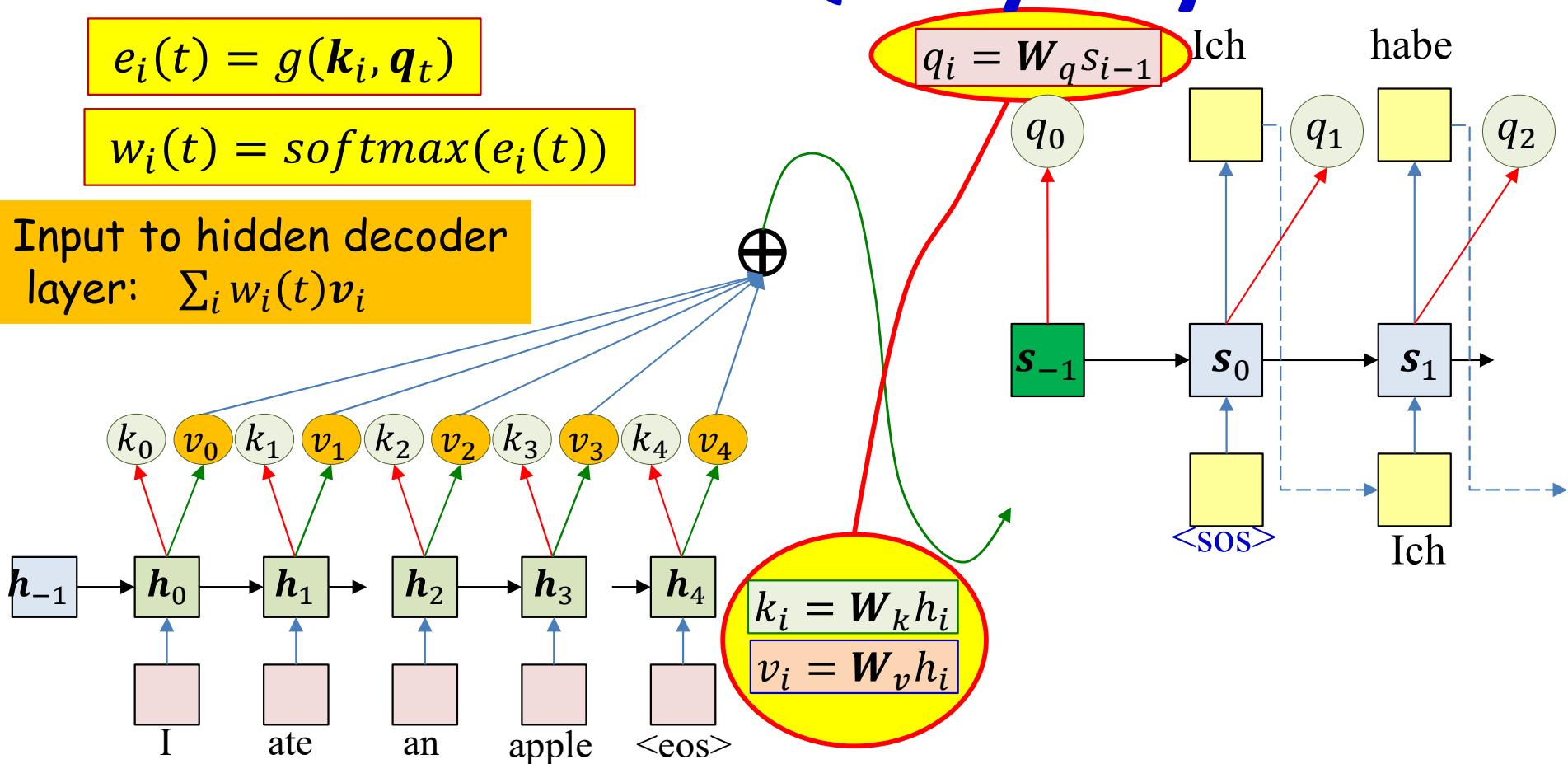
$$q_j = \boldsymbol{W}_q s_{j-1}$$

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

- 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

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

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

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

$$q_i = \mathbf{W}_q s_{i-1}$$

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

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

Ich   habe

q_0 q_1 q_2

s_{-1} s_0 s_1

$\langle$SOS$\rangle$ Ich

k_0 v_0 k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4

h_{-1} h_0 h_1 h_2 h_3 h_4

I   ate   an   apple $\langle$eos$\rangle$

- 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

59

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: $k_i = v_i = h_i$

$$q_t = s_{t-1}$$

We will continue using this assumption in the following slides but in practice the query-key-value format is used

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>
```
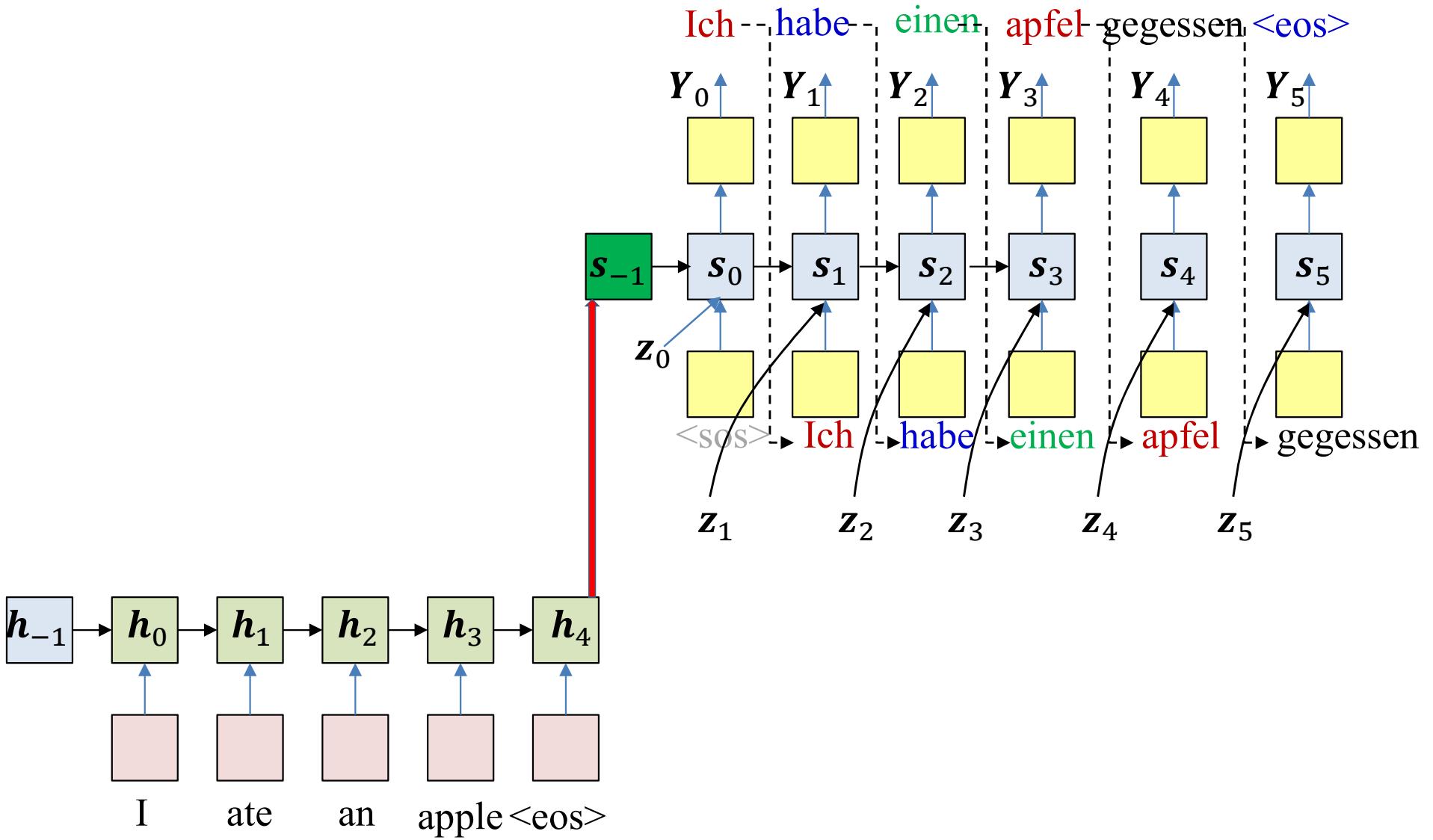
Pseudocode : Computing context with attention

```
# 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
```

Ich habe einen apfel gegessen <eos>

Y_0 Y_1 Y_2 Y_3 Y_4 Y_5

s_{-1} s_0 s_1 s_2 s_3 s_4 s_5

z_0

<sos> ➤ Ich ➤ habe ➤ einen ➤ apfel ➤ gegessen

z_1 z_2 z_3 z_4 z_5

h_{-1} ➔ h_0 ➔ h_1 ➔ h_2 ➔ h_3 ➔ h_4

I   ate   an   apple <eos>
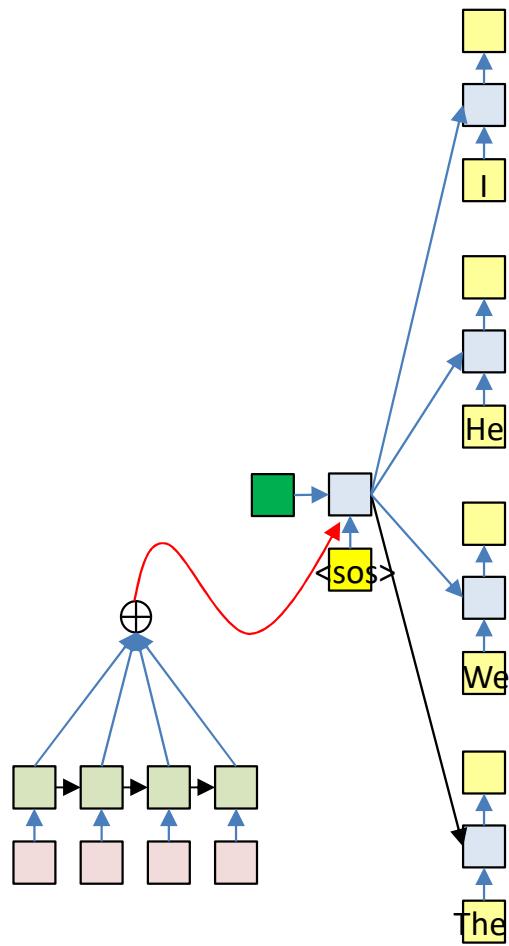
- As before, the objective of drawing: Produce the most likely output (that ends in an <eos>)

$$\underset{O_1,\ldots,O_L}{\mathrm{argmax}}\; y_1^{O_1} y_1^{O_2} \ldots y_1^{O_L}$$
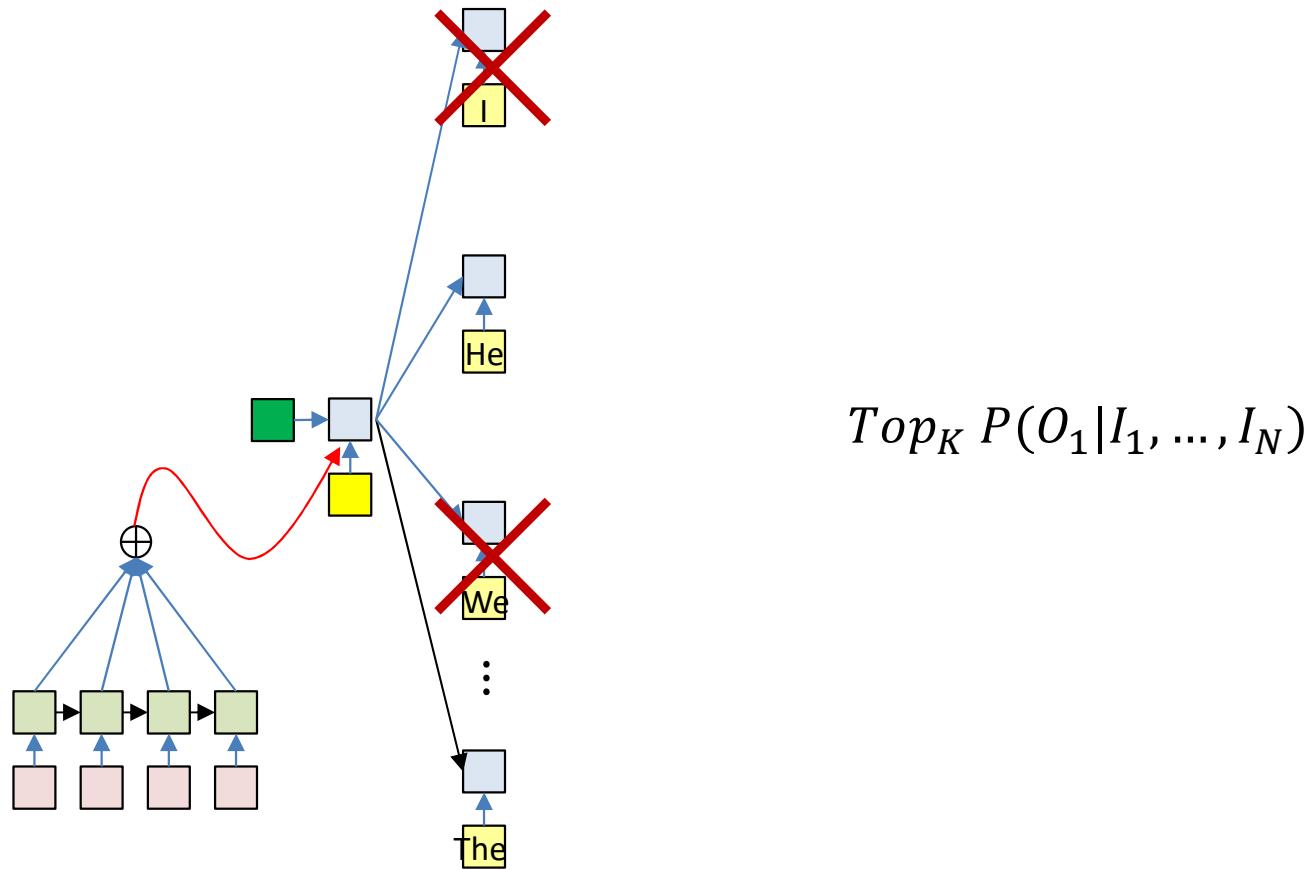
- Simply selecting the most likely symbol at each time may result in suboptimal output

Solution: Multiple choices
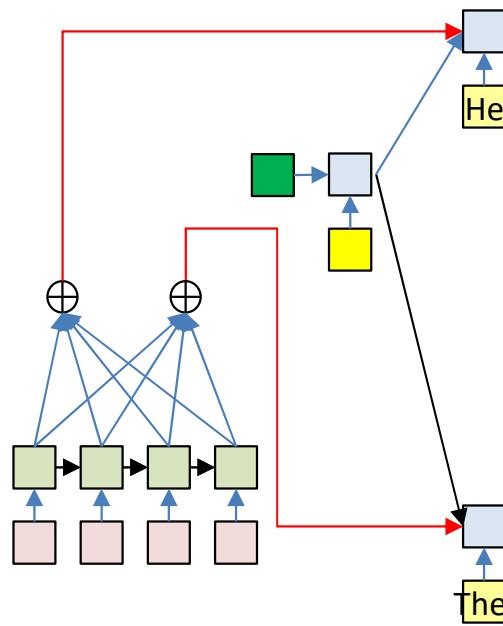
- Retain all choices and *fork* the network
  - With every possible word as input

To prevent blowup: Prune
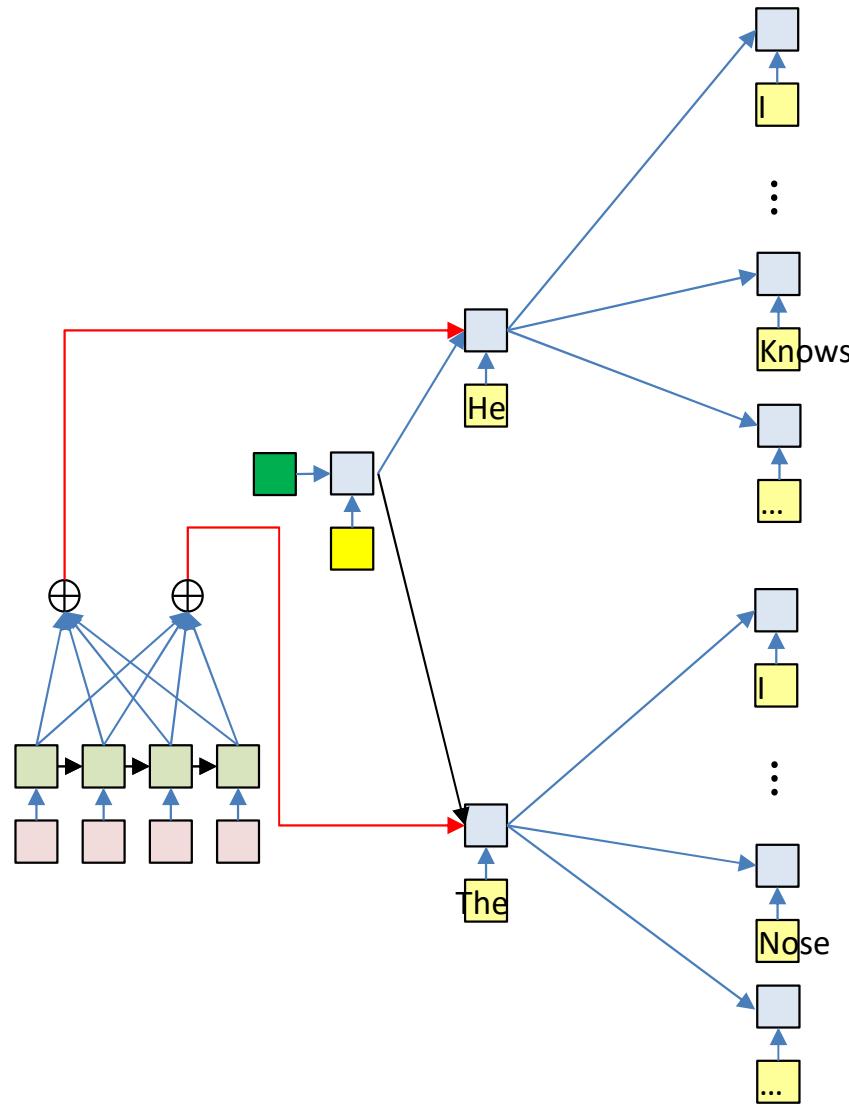
$$Top_K \; P(O_1|I_1, \ldots, I_N)$$

- **Prune**
  - At each time, retain only the top K scoring forks

Decoding

- At each time, retain only the top K scoring forks

Decoding

Note: based on product

$$Top_K \ P(O_2 O_1 | I_1, \dots, I_N)$$

$$= \ Top_K \ P(O_2 | O_1, I_1, \dots, I_N) P(O_1 | I_1, \dots, I_N)$$

- At each time, retain only the top K scoring forks

Decoding

Note: based on product

$$Top_K \, P(O_2 O_1 | I_1, \dots, I_N)$$

$$= Top_K \, P(O_2 | O_1, I_1, \dots, I_N) P(O_1 | I_1, \dots, I_N)$$

- At each time, retain only the top K scoring forks

Decoding

$$= Top_K P(O_2|O_1, I_1, \ldots, I_N) \times$$
$$P(O_2|O_1, I_1, \ldots, I_N) \times$$
$$P(O_1|I_1, \ldots, I_N)$$

- At each time, retain only the top K scoring forks

Decoding

$$= Top_K P(O_2|O_1, I_1, \ldots, I_N) \times$$
$$P(O_2|O_1, I_1, \ldots, I_N) \times$$
$$P(O_1|I_1, \ldots, I_N)$$

- At each time, retain only the top K scoring forks

Decoding

$$Top_K \prod_{t=1}^{n} P(O_n|O_1, \dots, O_{n-1}, I_1, \dots, I_N)$$

- At each time, retain only the top K scoring forks

Terminate

- **Terminate**
  - When the current most likely path overall ends in <eos>
    - Or continue producing more outputs (each of which terminates in <eos>) to get N-best outputs

72

Termination: <eos>

Example has K = 2

- **Terminate**
  - Paths cannot continue once the output an <eos>
    - So paths may be different lengths
      - Select the most likely sequence ending in <eos> across *all* terminating sequences

73

Pseudocode: Beam search

```
# Assuming encoder output H = h_in[1]… h_in[T] is available
path = <sos>
beam = {path}
pathscore = [path] = 1
state[path] = h[0]   # initial state (computed using your favorite method)
do   # Step forward
    nextbeam = {}
    nextpathscore = []
    nextstate = {}
    for path in beam:
        cfin = path[end]
        hpath = state[path]
        C   = compute_context_with_attention(hpath, H)
        y,h = RNN_decode_step(hpath, cfin, C)
        for c in Symbolset
            newpath = path + c
            nextstate[newpath] = h
            nextpathscore[newpath] = pathscore[path]*y[c]
            nextbeam += newpath # Set addition
        end
    end
    beam, pathscore, state, bestpath = prune(nextstate,nextpathscore,nextbeam)
until bestpath[end] = <eos>
```
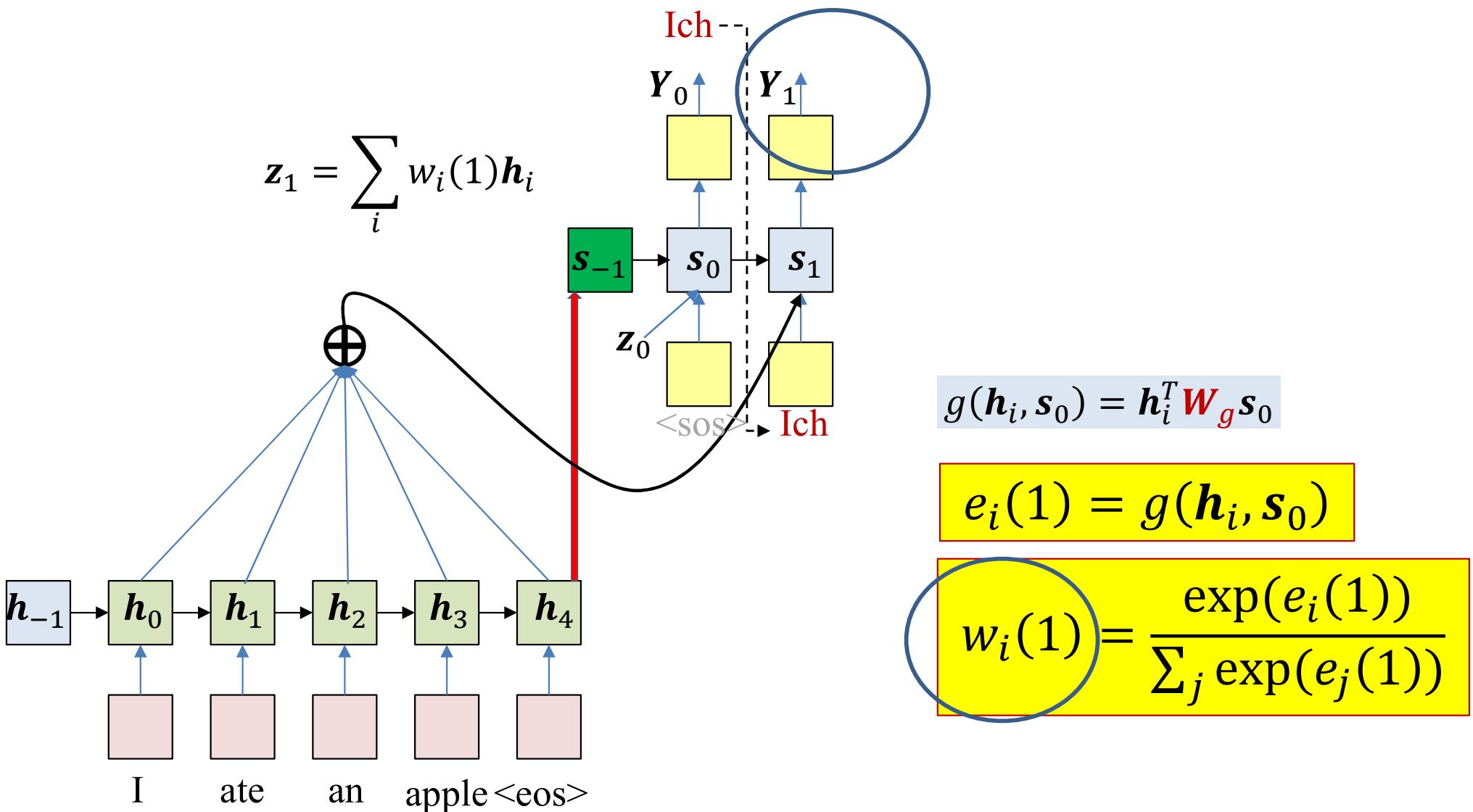
Pseudocode: Beam search

```
# Assuming encoder output H = h_in[1]… h_in[T] is available
path = <sos>
beam = {path}
pathscore = [path] = 1
state[path] = h[0]   # computed using your favorite method
context[path] = compute_context_with_attention(h[0], H)
do   # Step forward
    nextbeam = {}
    nextpathscore = []
    nextstate = {}
    nextcontext = {}
    for path in beam:
        cfin = path[end]
        hpath = state[path]
        C = context[path]
        y,h = RNN_decode_step(hpath, cfin, C)
        nextC = compute_context_with_attention(h, H)
        for c in Symbolset
            newpath = path + c
            nextstate[newpath] = h
            nextcontext[newpath] = nextC
            nextpathscore[newpath] = pathscore[path]*y[c]
            nextbeam += newpath # Set addition
        end
    end
    beam, pathscore, state, context, bestpath =
                    prune (nextstate, nextpathscore, nextbeam, nextcontext)
until bestpath[end] = <eos>
```

Slightly more efficient.
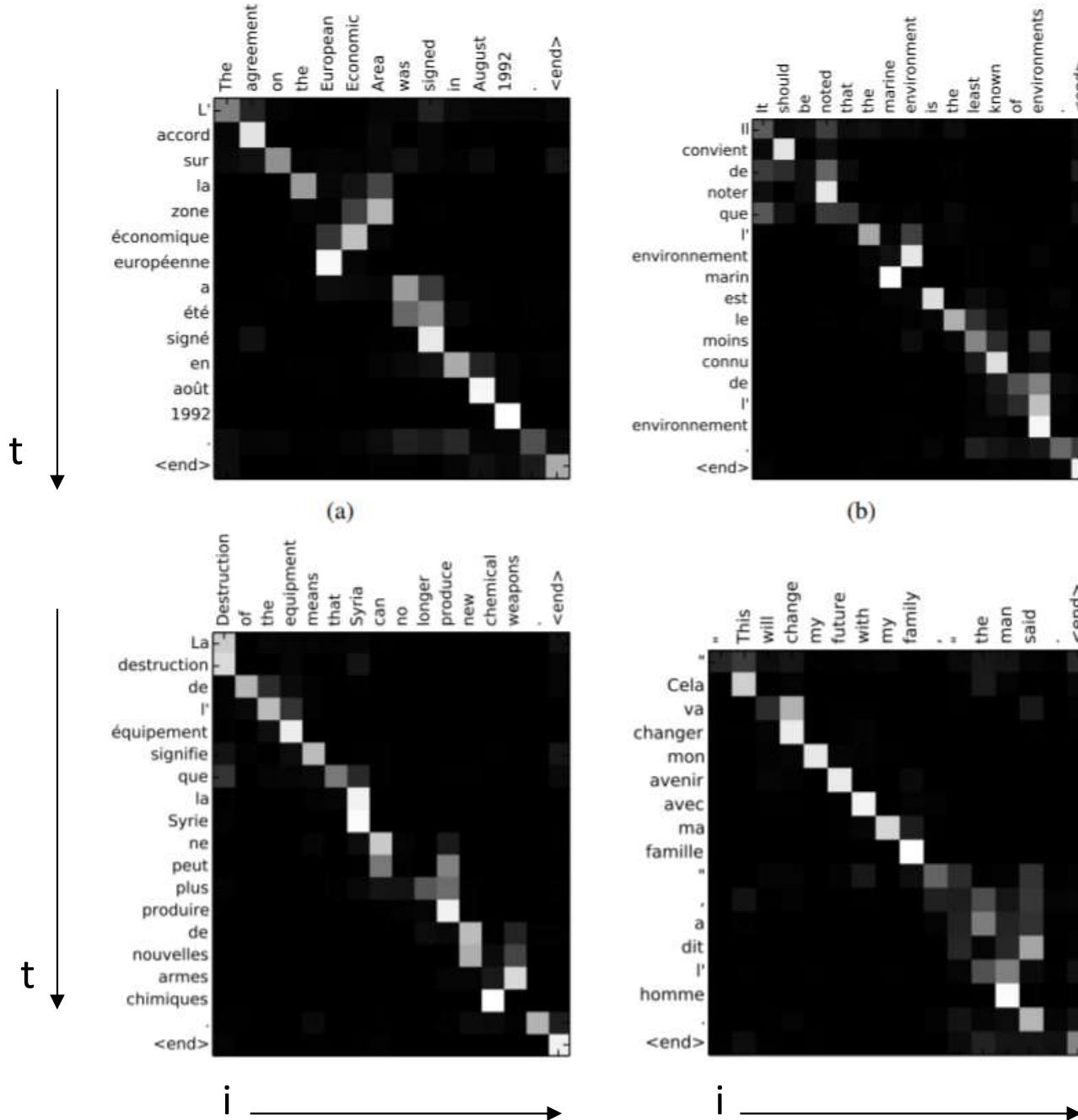
Does not perform redundant context computation

75

What does the attention learn?

$$z_1 = \sum_i w_i(1) h_i$$

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

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

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

- The key component of this model is the attention weight
  - It captures the relative importance of each position in the input to the current output

"Alignments" example: Bahdanau et al.

Plot of $w_i(t)$

Color shows value (white is larger)

Note how most important input words for any output word get automatically highlighted

The general trend is somewhat linear because word order is roughly similar in both languages

77
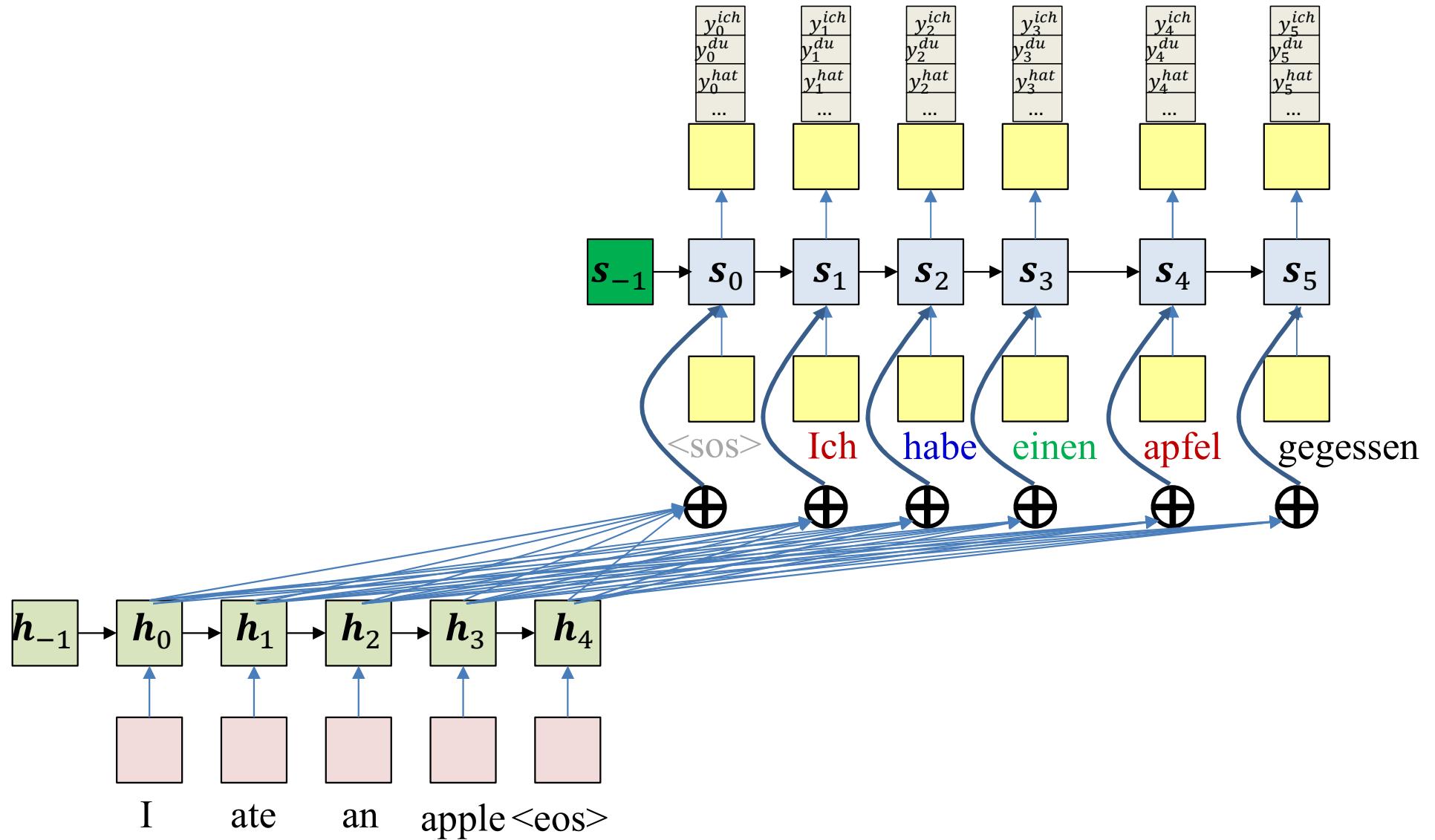
Translation Examples

Source	An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.
Reference	Le privilège d'admission est le droit d'un médecin, en vertu de son statut de membre soignant d'un hôpital, d'admettre un patient dans un hôpital ou un centre médical afin d'y délivrer un diagnostic ou un traitement.
RNNenc-50	Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.
RNNsearch-50	Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.
Google Translate	Un privilège admettre est le droit d'un médecin d'admettre un patient dans un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, fondée sur sa situation en tant que travailleur de soins de santé dans un hôpital.

| Source	This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.
Reference	Ce type d'expérience entre dans le cadre des efforts de Disney pour "étendre la durée de vie de ses séries et construire de nouvelles relations avec son public grâce à des plateformes numériques qui sont de plus en plus importantes", a-t-il ajouté.
RNNenc-50	Ce type d'expérience fait partie des initiatives du Disney pour "prolonger la durée de vie de ses nouvelles et de développer des liens avec les lecteurs numériques qui deviennent plus complexes.
RNNsearch-50	Ce genre d'expérience fait partie des efforts de Disney pour "prolonger la durée de vie de ses séries et créer de nouvelles relations avec des publics via des plateformes numériques de plus en plus importantes", a-t-il ajouté.
Google Translate	Ce genre d'expérience fait partie des efforts de Disney à "étendre la durée de vie de sa série et construire de nouvelles relations avec le public par le biais des plates-formes numériques qui deviennent de plus en plus important", at-il ajouté.
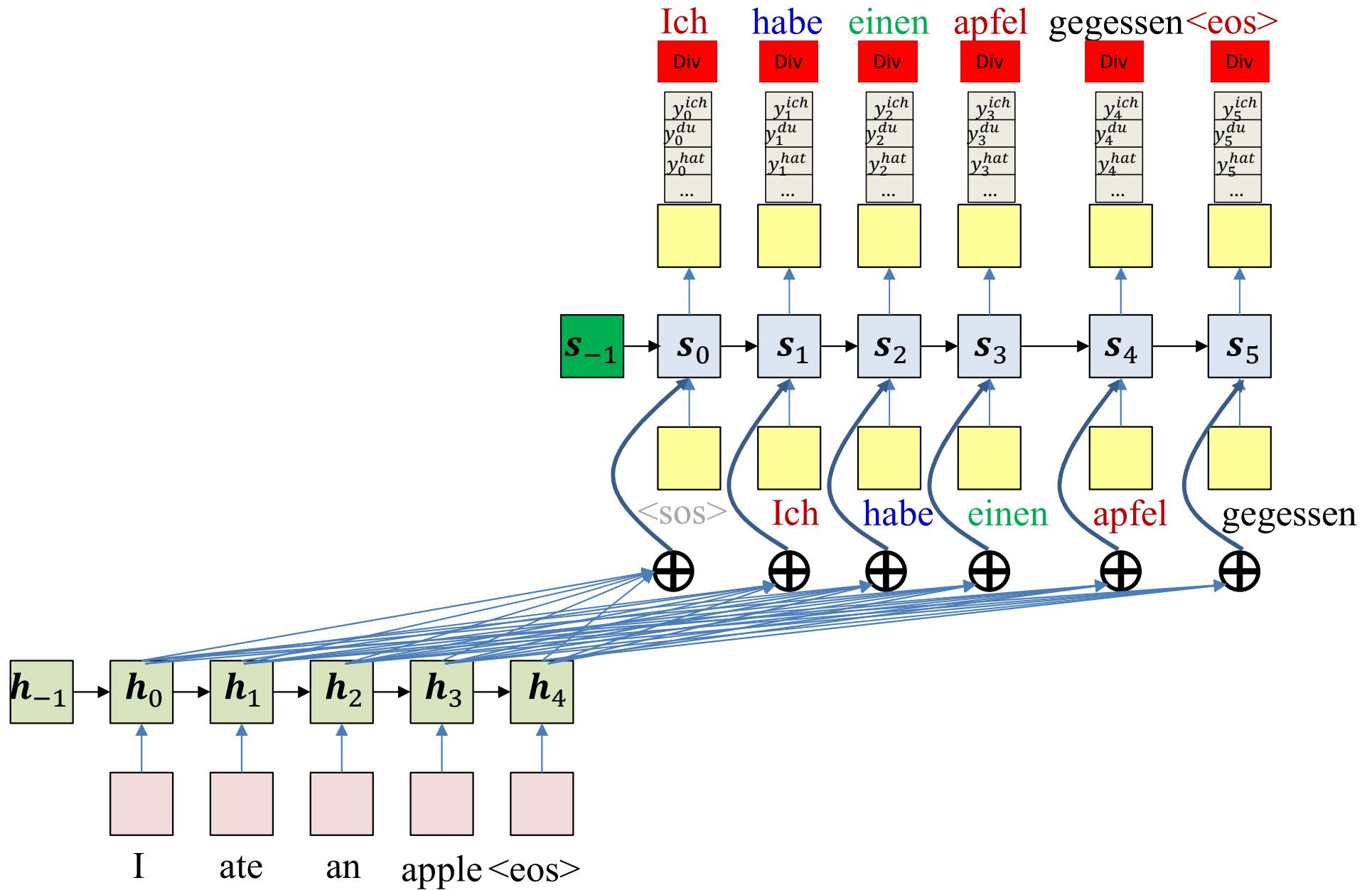
- Bahdanau et al. 2016

Training the network

- We have seen how a trained network can be used to compute outputs
  - Convert one sequence to another

- Let's consider training..
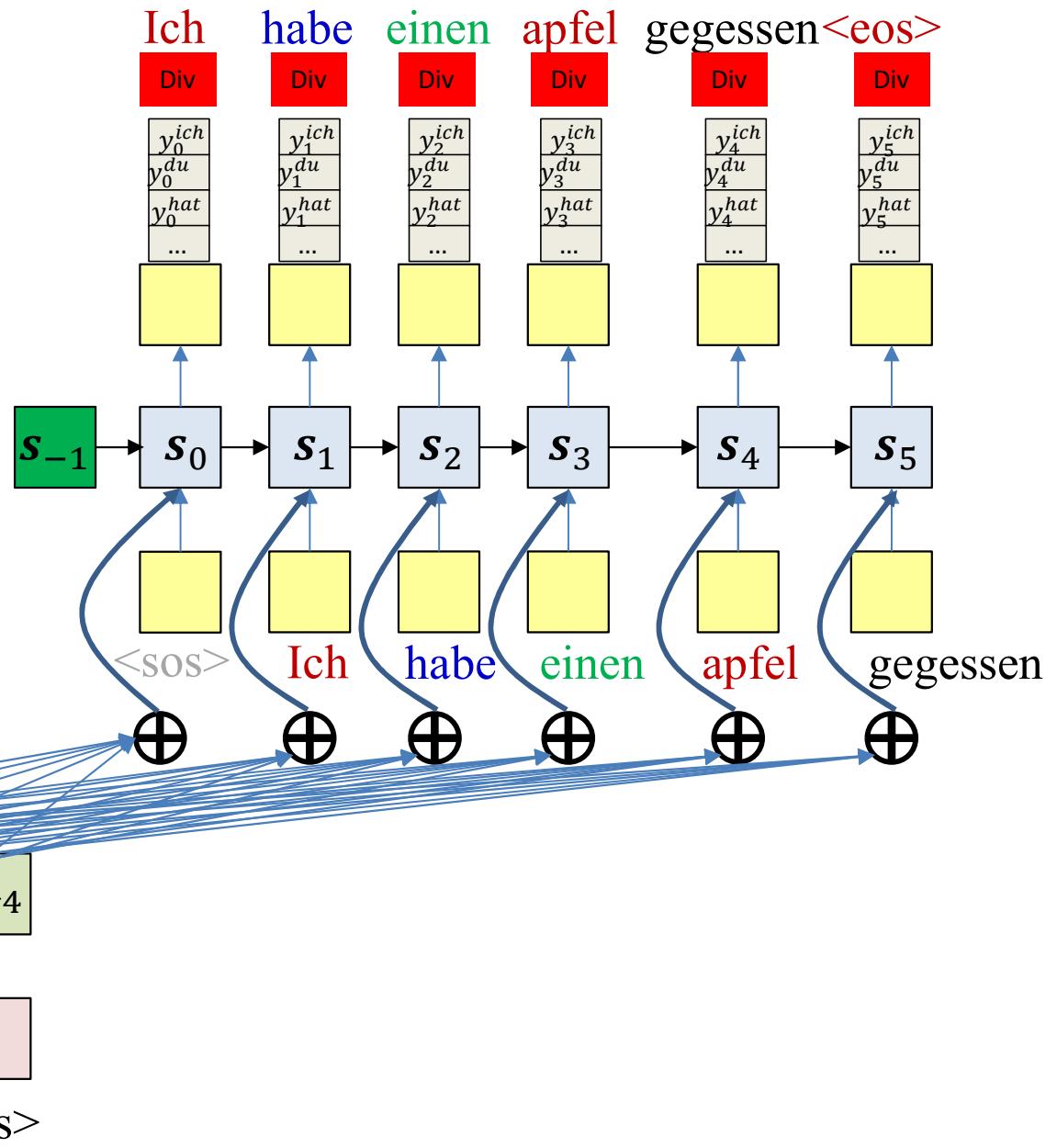
- Given training input (source sequence, target sequence) pairs
- **Forward pass**: Pass the actual input sequence through the encoder
  - At each time the output is a probability distribution over words

Ich   habe   einen   apfel   gegessen   <eos>

- **Backward pass**: Compute a divergence between target output and output distributions
  - Backpropagate derivatives through the network
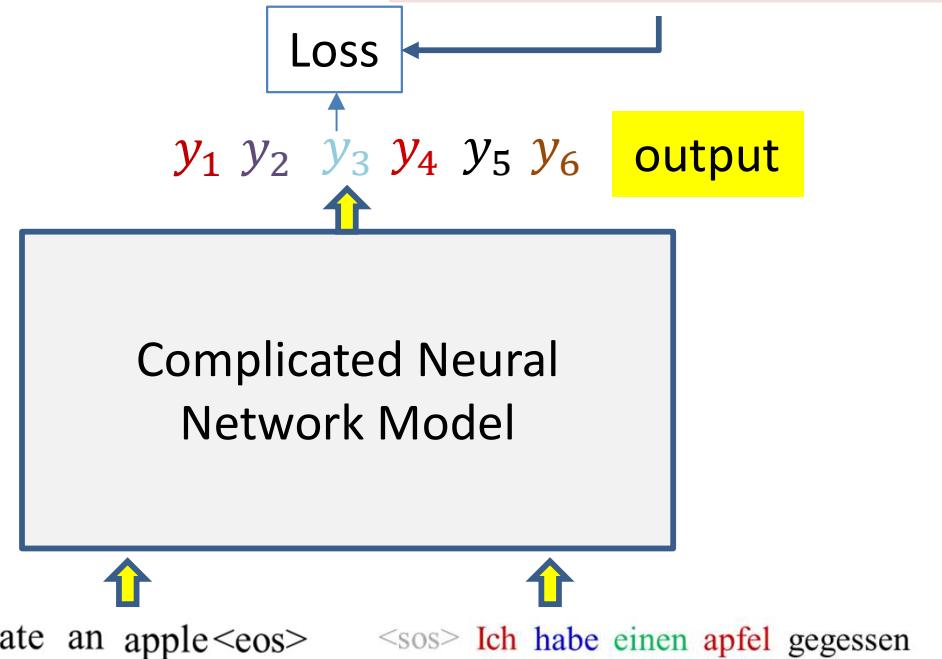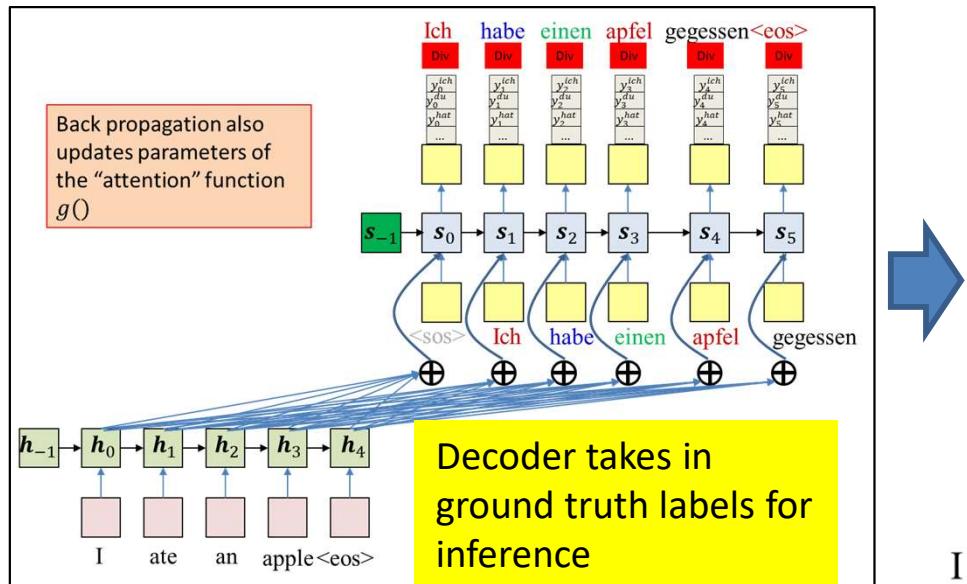
Back propagation also updates parameters of the "attention" function $g()$

- **Backward pass**: Compute a divergence between target output and output distributions
  - Backpropagate derivatives through the network

Training Paradigms
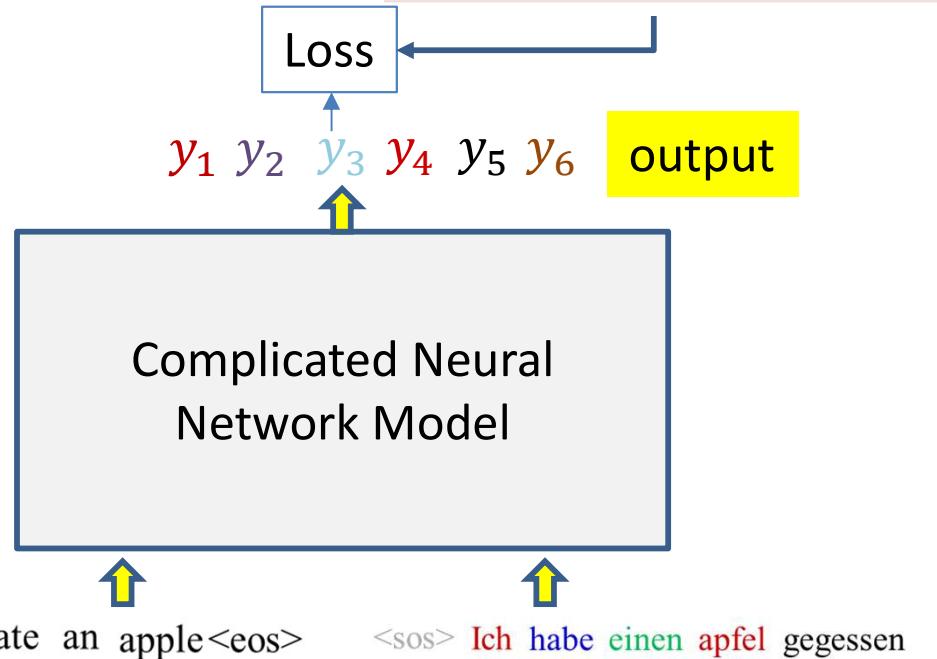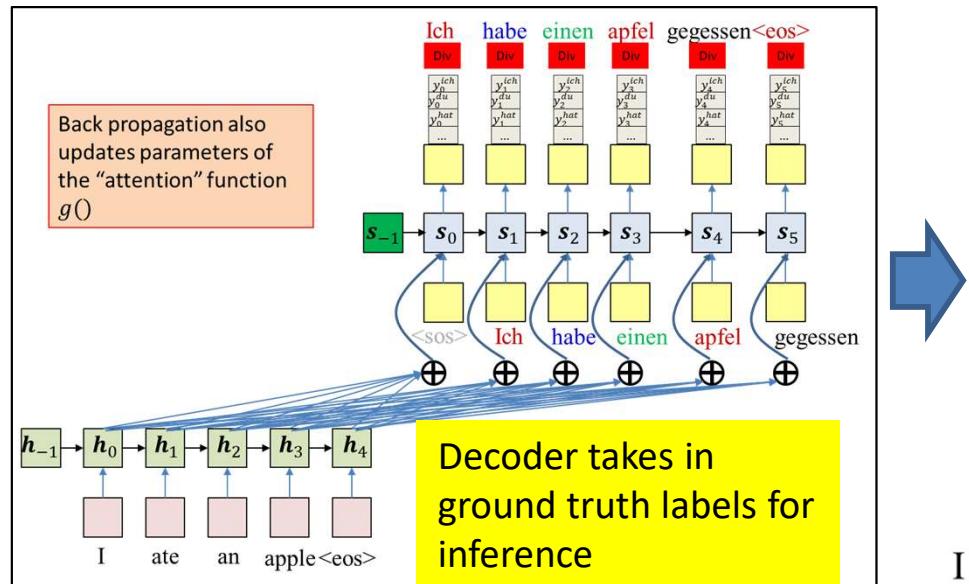
Ich habe einen apfel gegessen<eos>

Loss

$y_1 \; y_2 \; y_3 \; y_4 \; y_5 \; y_6$ output

Complicated Neural Network Model

Back propagation also updates parameters of the "attention" function $g()$

Decoder takes in ground truth labels for inference

I ate an apple<eos>

<sos> Ich habe einen apfel gegessen

- This approach, in fact, performs maximum-likelihood estimation of the model
  - The forward pass of training takes in both inputs *and ground-truth* output labels for generation
  - This forward pass is *not* like regular inference

Training Paradigms
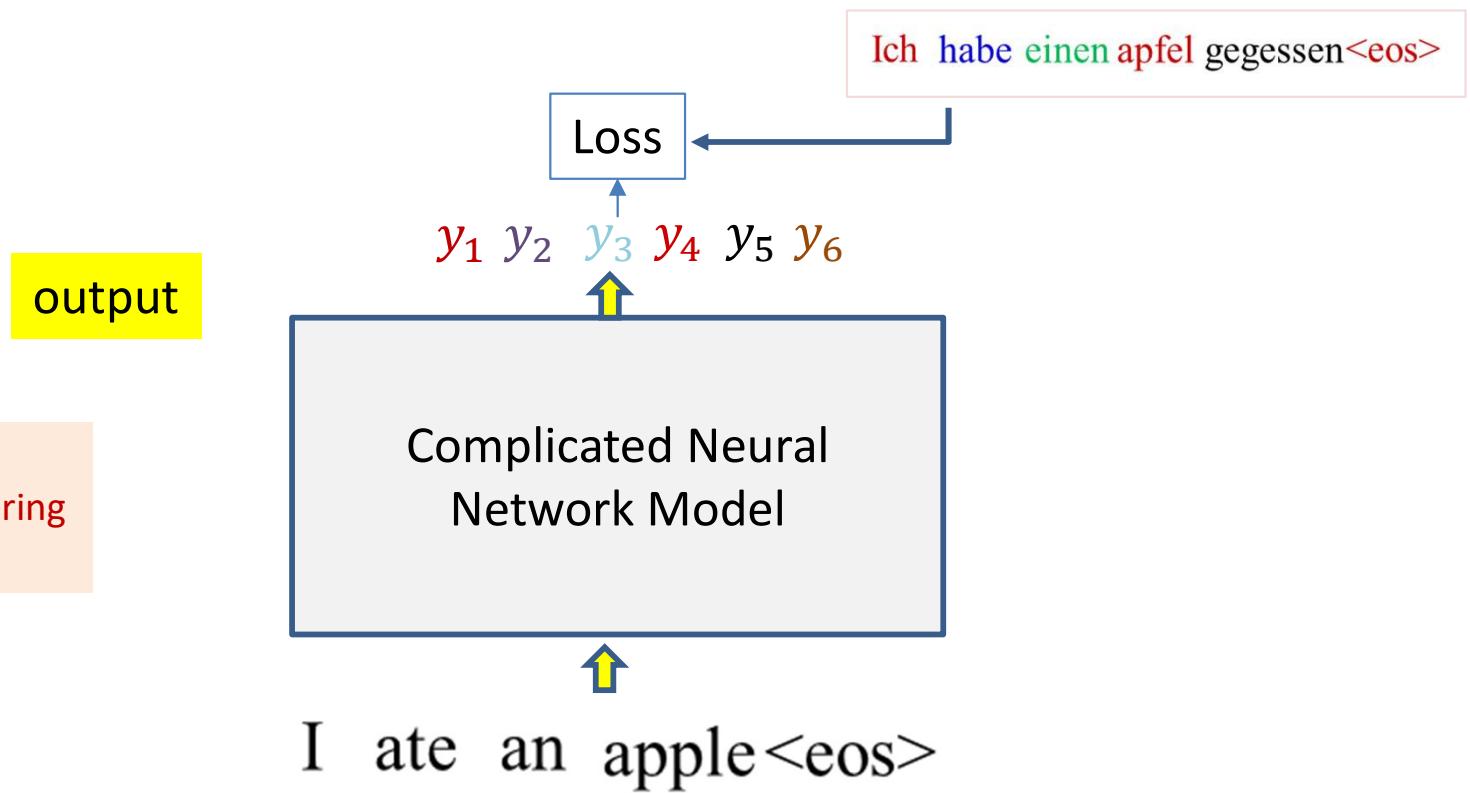
This approach, in fact, performs maximum-likelihood estimation of the model
- The forward pass of training takes in both inputs *and ground-truth* output labels for generation
- This forward pass is *not* like regular inference

But there's another more traditional mechanism for training
- Where the model *only* gets the input during the forward pass,
  - The forward pass is identical to inference
- and the actual ground truth label is *only* used to compute loss…
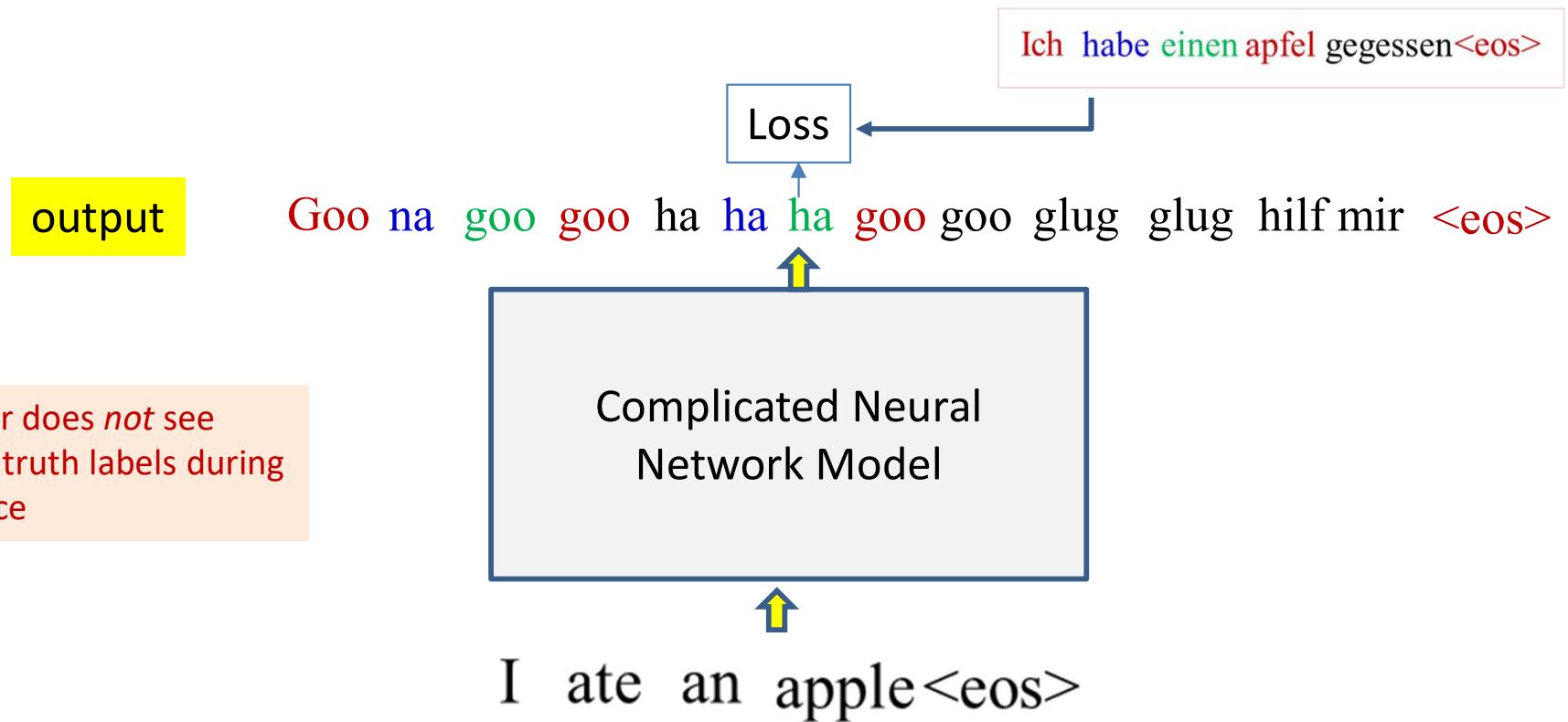
84

Training Paradigms

Ich habe einen apfel gegessen<eos>

Loss

y_1 y_2 y_3 y_4 y_5 y_6

output

Decoder does *not* see ground truth labels during inference

Complicated Neural Network Model

I ate an apple<eos>

- The traditional mechanism for training
  - Why would this not work?

85

Training Paradigms

Ich habe einen apfel gegessen<eos>

Loss

output

Goo na goo goo ha ha ha goo goo glug glug hilf mir <eos>

Decoder does *not* see ground truth labels during inference

```
Complicated Neural
Network Model
```

I ate an apple<eos>
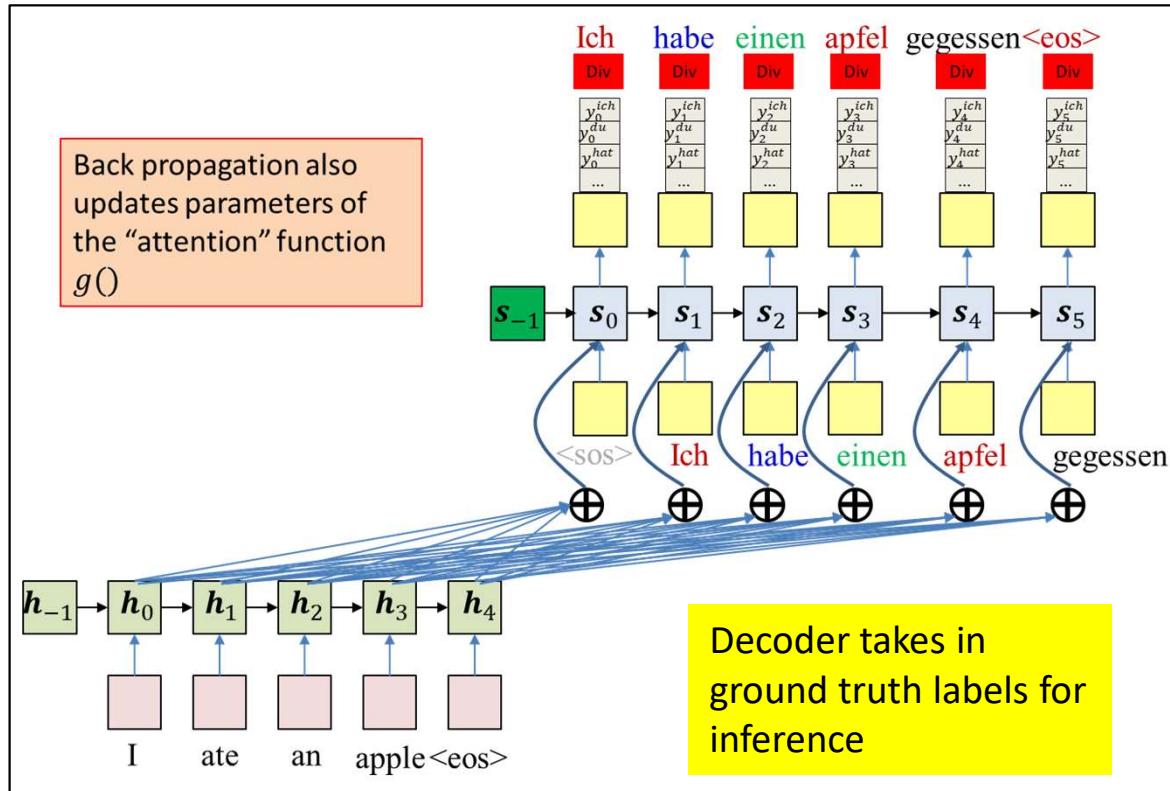
- The traditional mechanism for training
  - Why would this not work?

- The output could be *anything*
  - And in the early stages of training, *will* be random
  - Cannot really compute a loss
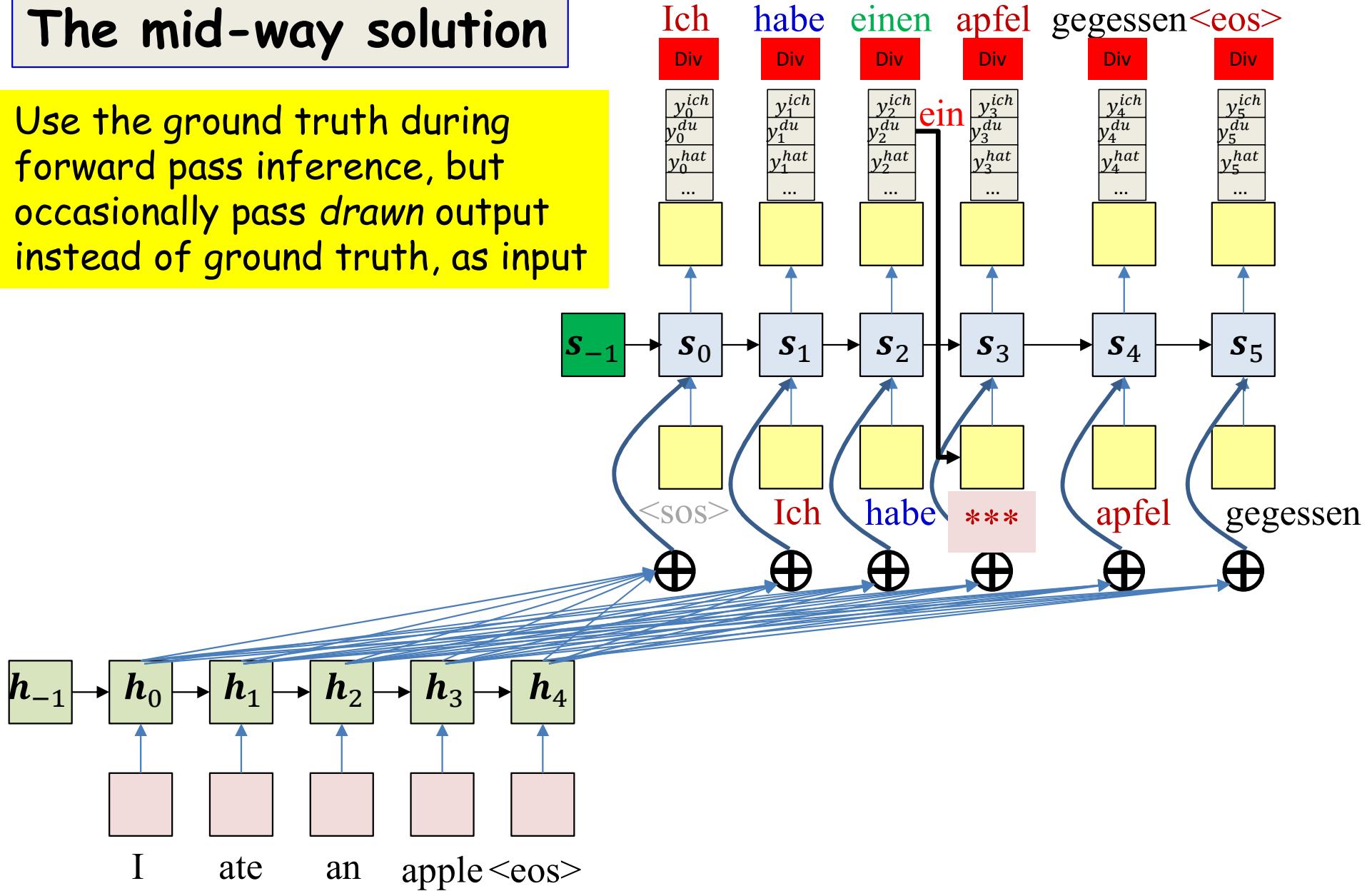
Training Paradigms

- The maximum-likelihood approach ensures a one-to-one correspondence between outputs and target outputs

- But from the perspective of *traditional* training, it is "cheating" during the inference of the forward pass, where the ground truth is being passed in instead of drawn samples
  - "Teacher forcing"
  - Is there a midway solution?
    - Use drawn words during the forward pass inference while maintaining one-to-one correspondence
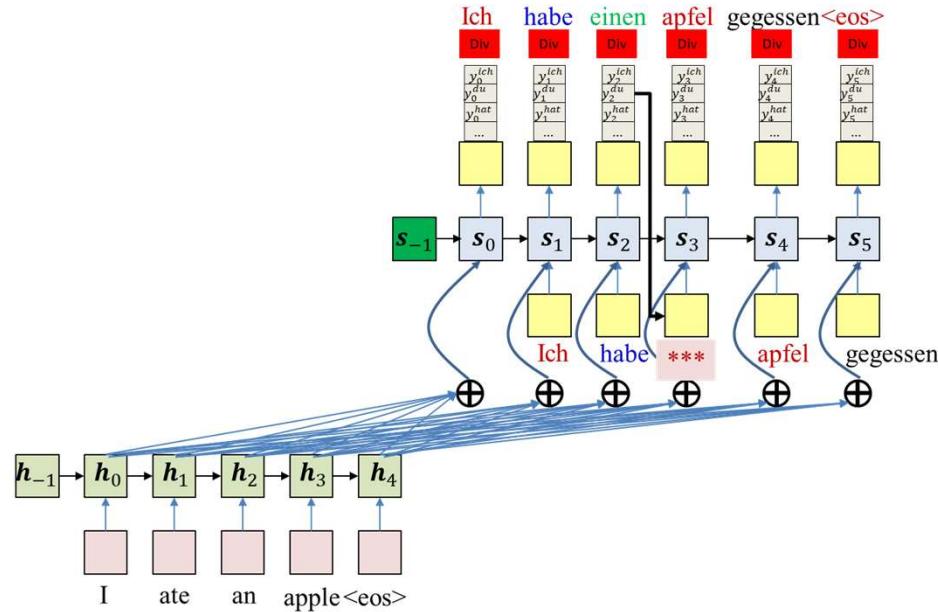
The mid-way solution

Use the ground truth during forward pass inference, but occasionally pass *drawn* output instead of ground truth, as input

- **Backward pass**: Compute a divergence between target output and output distributions
 - Backpropagate derivatives through the network

Gumbel Noise trick

- Sampling is not differentiable

- The "Gumbel noise" trick:
  - "Reparametrization" :
  $$RandomSample(Y) = \underset{i}{\mathrm{argmax}}(G_i + \log(Y))$$
  - G_i is drawn from the standard Gumbel distribution $Gumbel(0,1)$
- The "argmax" can be replaced by a "softmax", making the process differentiable w.r.t. network outputs
  - $decoderoutput(t) = softmax(G_i + \log(Y(t)))$
- $\nabla_{Y(t)}decoderouput(t)$ is employed in the chain rule to pass derivatives from t+1 back to $Y(t)$
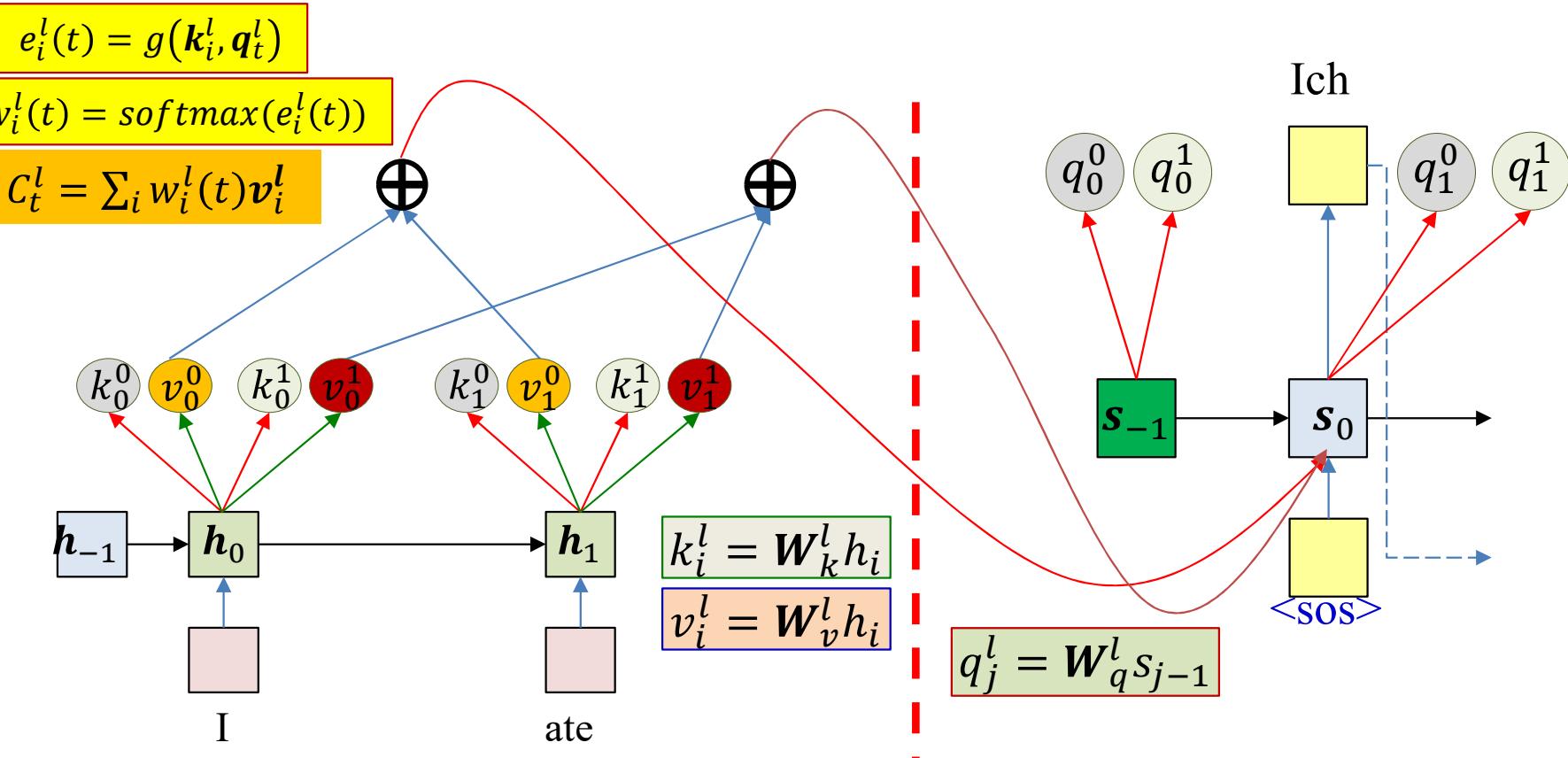
Tricks of the trade…

- Teacher forcing:
  - Ideally we would only use the decoder output during inference
  - This will not be stable
  - Passing in ground truth instead is "teacher forcing"

- Sampling the output:
  - Sample the system output and
  - as input during training for only some of the time

- The "Gumbel noise" trick:
  - Sampling is not differentiable, and gradients cannot be passed through it
  - The "Gumbel noise" approach recasts sampling as computing the argmax of a Gumbel distribution, with the network output as parameters
  - The "argmax" can be replaced by a "softmax", making the process differentiable w.r.t. network outputs

Various extensions

- Bidirectional processing of input sequence
  - Bidirectional networks in encoder
  - E.g. "Neural Machine Translation by Jointly Learning to Align and Translate", Bahdanau et al. 2016

- Attention: Local attention vs global attention
  - E.g. "Effective Approaches to Attention-based Neural Machine Translation", Luong et al., 2015
  - Other variants

Extensions: Multihead attention

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

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

$$C_t^l = \sum_i w_i^l(t)\boldsymbol{v}_i^l$$

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

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

$$q_j^l = \boldsymbol{W}_q^l s_{j-1}$$

Ich

I ate

<SOS>

- Have multiple query/key/value sets.
  - Each attention "head" uses one of these sets
  - The combined contexts from all heads are passed to the decoder
- Each "attender" focuses on a different aspect of the input that's important for the decode

Poll 2 (@1488, @1489)

Which of the following will give you the optimal decode with an attention-based decoder?

- Full tree search
- Beam search

Mark all that are true

- In a query-key-value based attention mechanism, the key and value are used to compute attention weights
- Multi-head attention computes a separate set of keys and values for each head, at each input
- Multi-head attention computes a separate query for each head, at each output
- Training with teacher forcing computes the theoretically correct loss and minimizes it

Poll 2

Which of the following will give you the optimal decode with an attention-based decoder?

- **Full tree search**
- Beam search

Mark all that are true

- In a query-key-value based attention mechanism, the key and value are used to compute attention weights
- **Multi-head attention computes a separate set of keys and values for each head, at each input**
- **Multi-head attention computes a separate query for each head, at each output**
- Training with teacher forcing computes the theoretically correct loss and minimizes it

Some impressive results..

- Attention-based models are currently responsible for the state of the art in many sequence-conversion systems
  - Machine translation
    - Input: Text in source language
    - Output: Text in target language
  - Speech recognition
    - Input: Speech audio feature vector sequence
    - Output: Transcribed word or character sequence

Attention models in image captioning

A woman is throwing a <u>frisbee</u> in a park.

A <u>dog</u> is standing on a hardwood floor.

A <u>stop</u> sign is on a road with a mountain in the background.

A little <u>girl</u> sitting on a bed with a teddy bear.

A group of <u>people</u> sitting on a boat in the water.
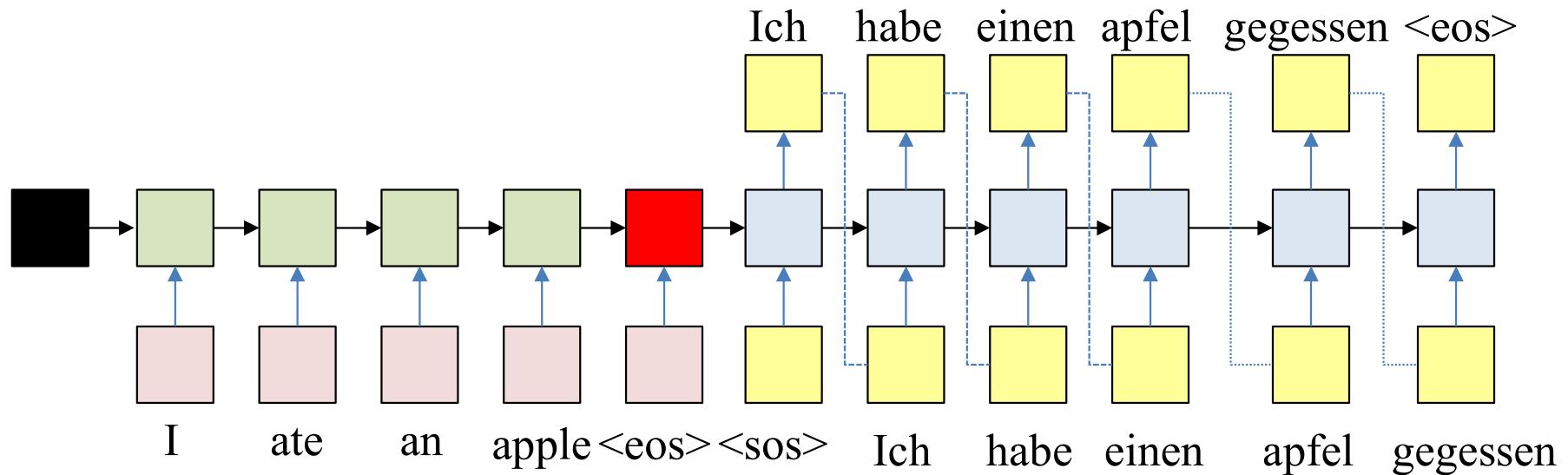
A giraffe standing in a forest with <u>trees</u> in the background.

- "Show attend and tell: Neural image caption generation with visual attention", Xu et al., 2016

- Encoder network is a convolutional neural network
  - Filter outputs at each location are the equivalent of $\boldsymbol{h}_i$ in the regular sequence-to-sequence model

Recap
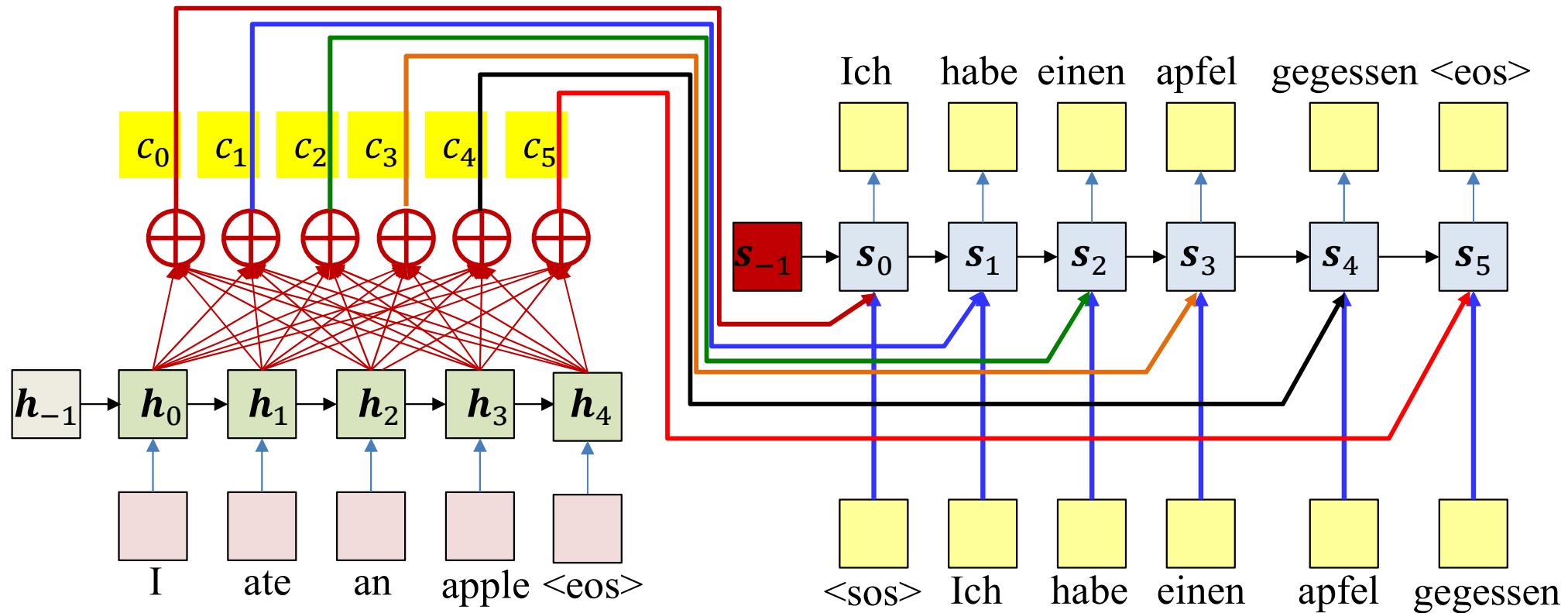
- Have looked at various forms of sequence-to-sequence models

- Generalizations of recurrent neural network formalisms

- For more details, please refer to papers
  - Post on piazza if you have questions

- Will appear in HW4: **Speech recognition with attention models**

Recap: Seq2Seq models
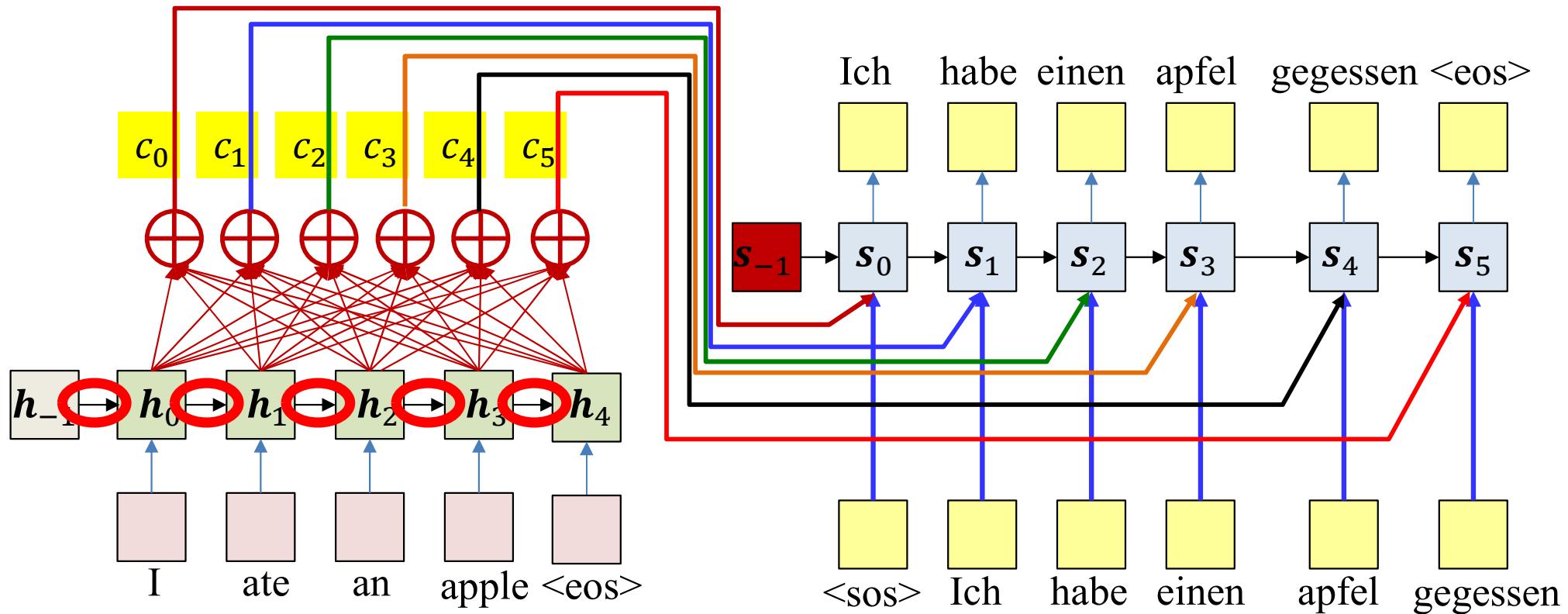
- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs

Recap: Attention Models
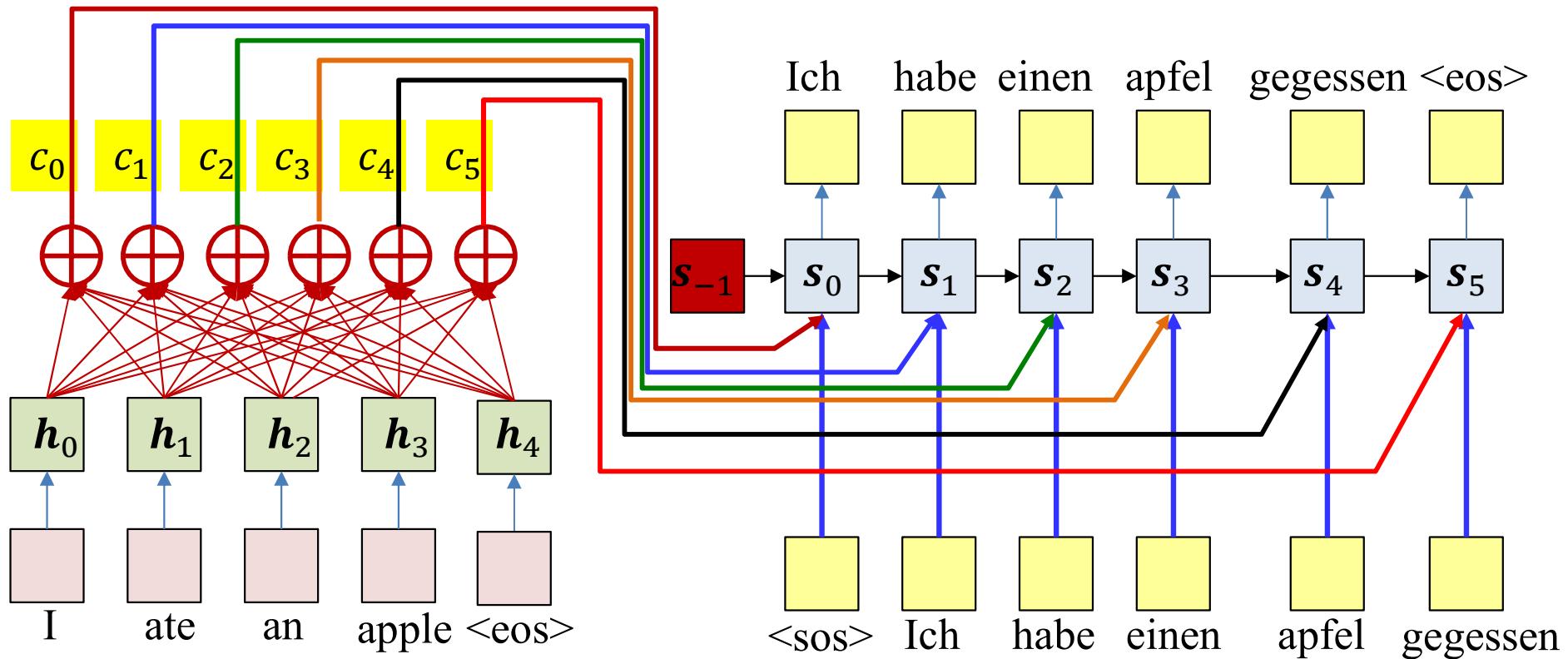
- Encoder recurrently produces hidden representations of input word sequence

- 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

99

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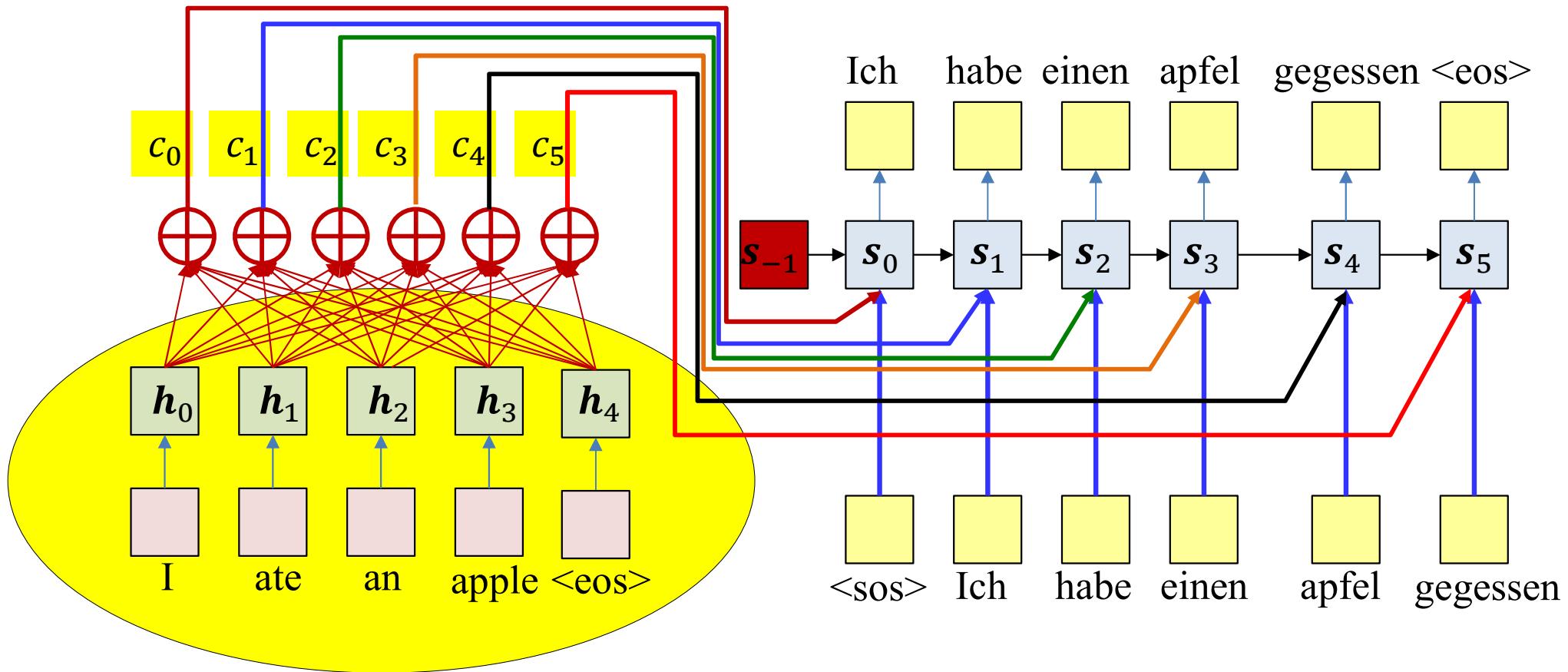
- Problem: Because of the recurrence, the hidden representation for any word is also influenced by *all* preceding words
  - The decoder is actually paying attention to the sequence, and not just the word

- 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?

Non-recurrent encoder
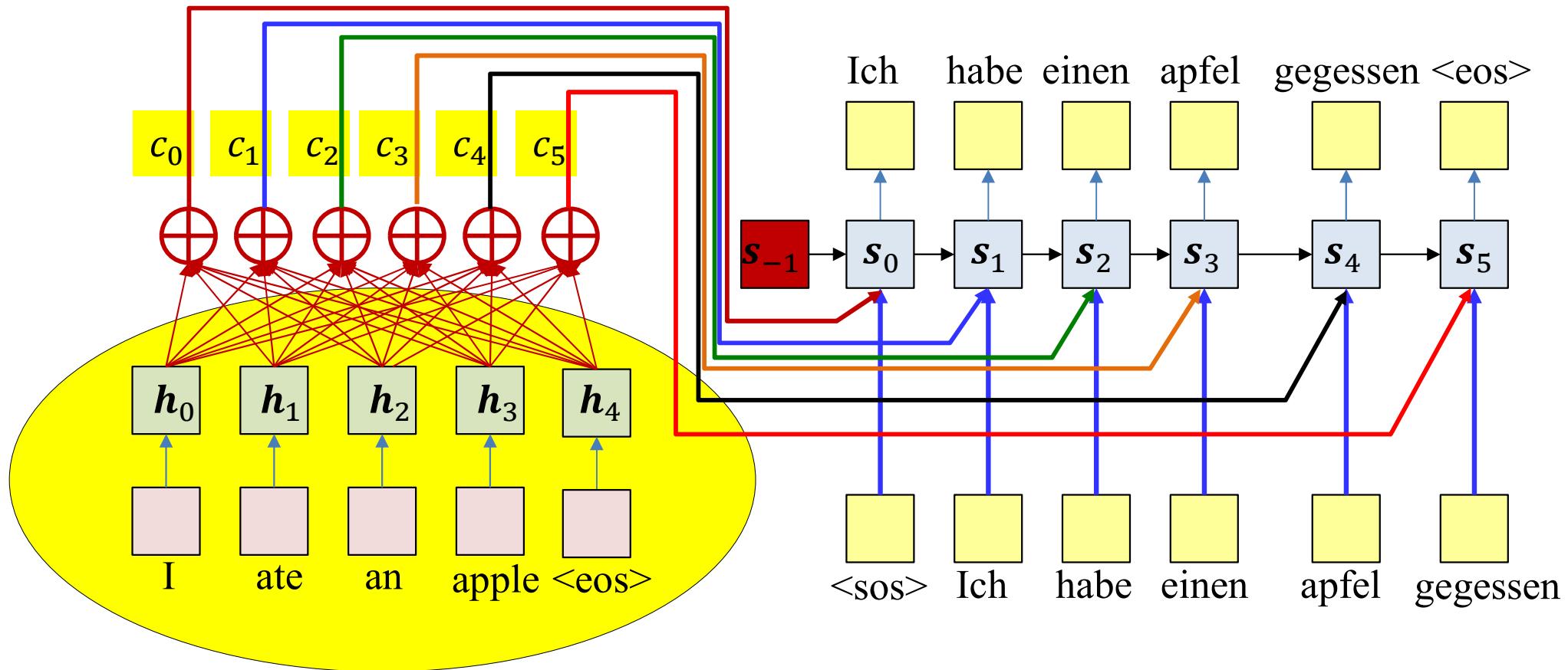
- Modification: Let us eliminate the recurrence in the encoder

Non-recurrent encoder
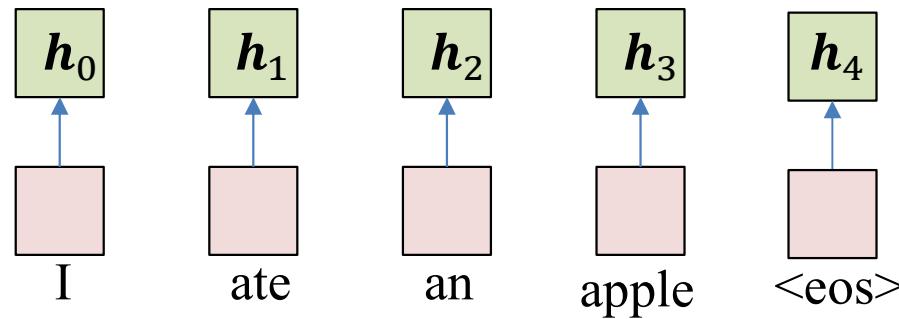
- But this will eliminate *context-specificity* in the encoder embeddings
  - The embedding for "an" must really depend on the remaining words
    - It could be translated to "ein", "einer", or "eines" depending on the context.
- Solution: Use the attention framework itself to introduce context-specificity in embeddings

Recap: Non-recurrent encoder

- The encoder in a sequence-to-sequence model can be composed without recurrence.
- Use the attention framework itself to introduce context-specificity in embeddings
  - "Self" attention

103
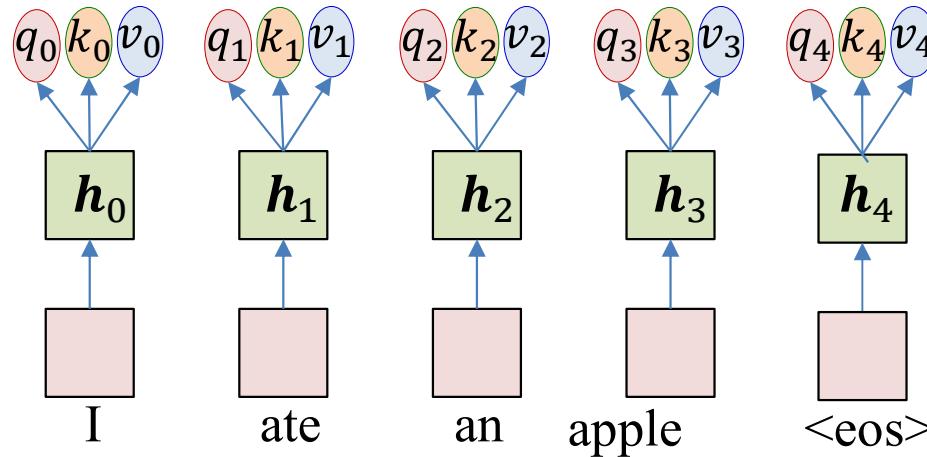
Self attention

- First, for every word in the input sequence we compute an initial representation
  - E.g. using a single MLP layer

Self attention

$$q_i = \boldsymbol{W}_q h_i$$

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

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

- Then, from each of the hidden representations, we compute a query, a key, and a value.
    - Using separate linear transforms
    - The weight matrices W_q, W_k and W_v are learnable parameters

Self Attention

$$e_{ij} = q_i^T k_j$$

$$w_{i0}, \ldots, w_{iN} = softmax(e_{i0}, \ldots, e_{iN})$$

$$q_i = \boldsymbol{W}_q h_i$$

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

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

$$w_{ij} = attn(q_i, k_{0:N})$$

Softmax

q_0 k_0 v_0 q_1 k_1 v_1 q_2 k_2 v_2 q_3 k_3 v_3 q_4 k_4 v_4

$\boldsymbol{h}_0$ $\boldsymbol{h}_1$ $\boldsymbol{h}_2$ $\boldsymbol{h}_3$ $\boldsymbol{h}_4$

I   ate   an   apple   \<eos\>

- For each word, we compute an attention weight between that word and all other words
  - The raw attention of the ith word to the jth word is a function of query q_i and key k_j
  - The raw attention values are put through a softmax to get the final attention weights
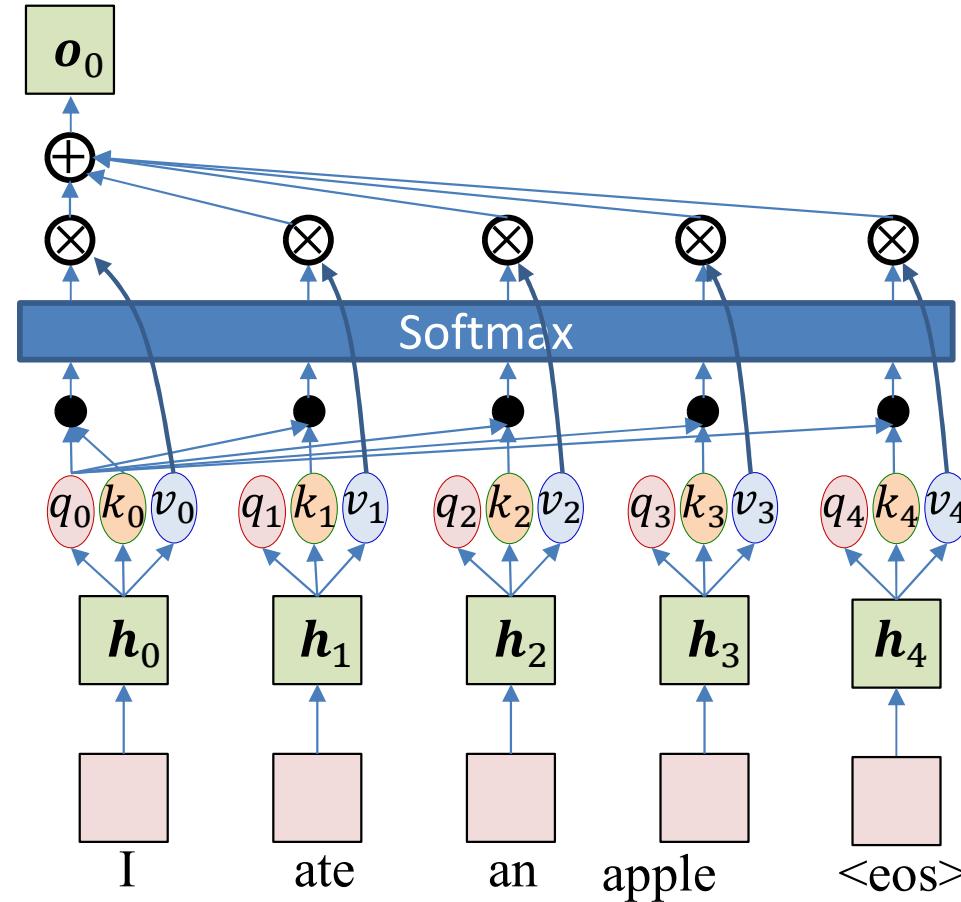
$q_i = W_q h_i$

$k_i = W_k h_i$
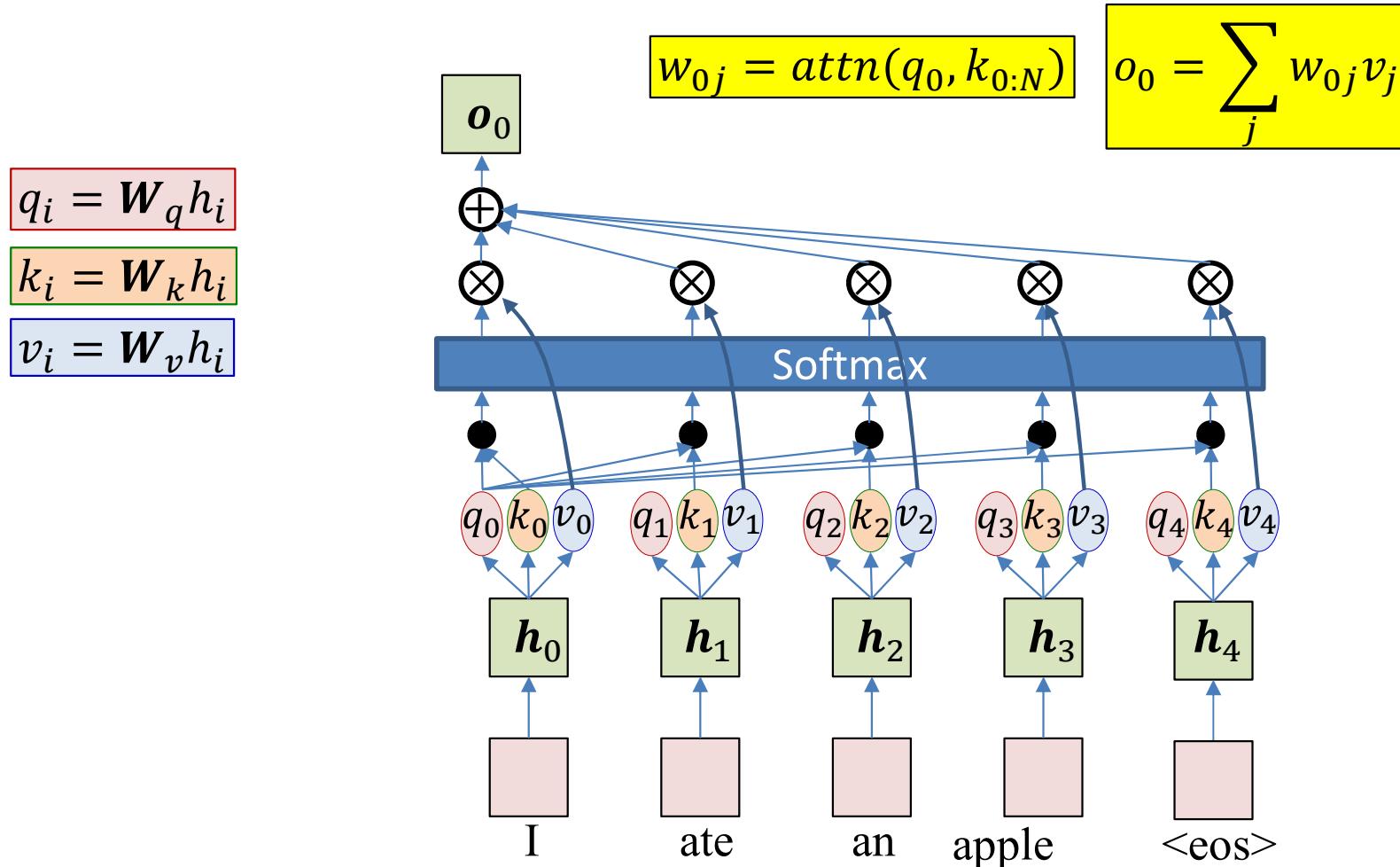
$v_i = W_v h_i$

$w_{ij} = attn(q_i, k_{0:N})$
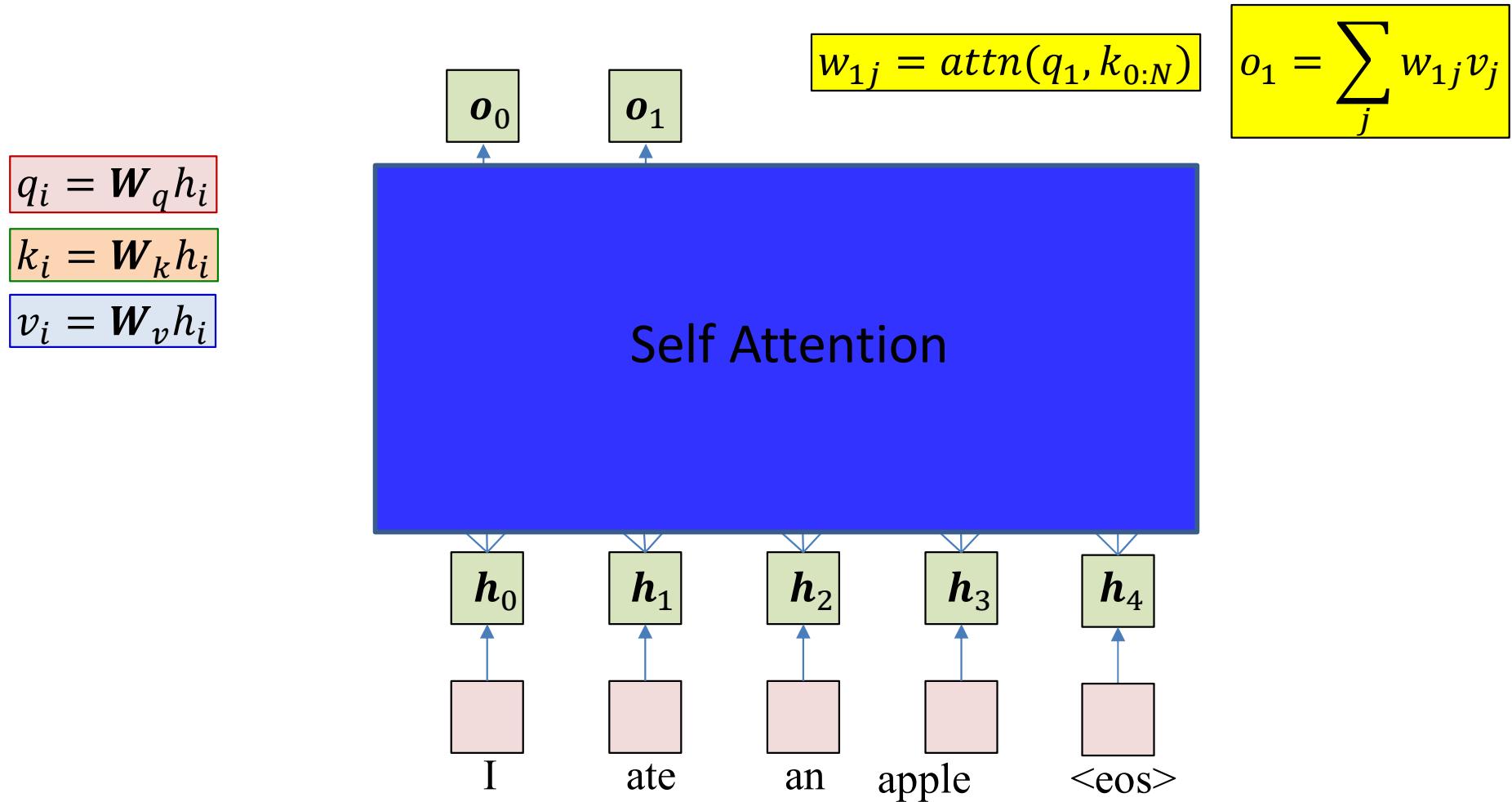
$$o_i = \sum_j w_{ij} v_j$$
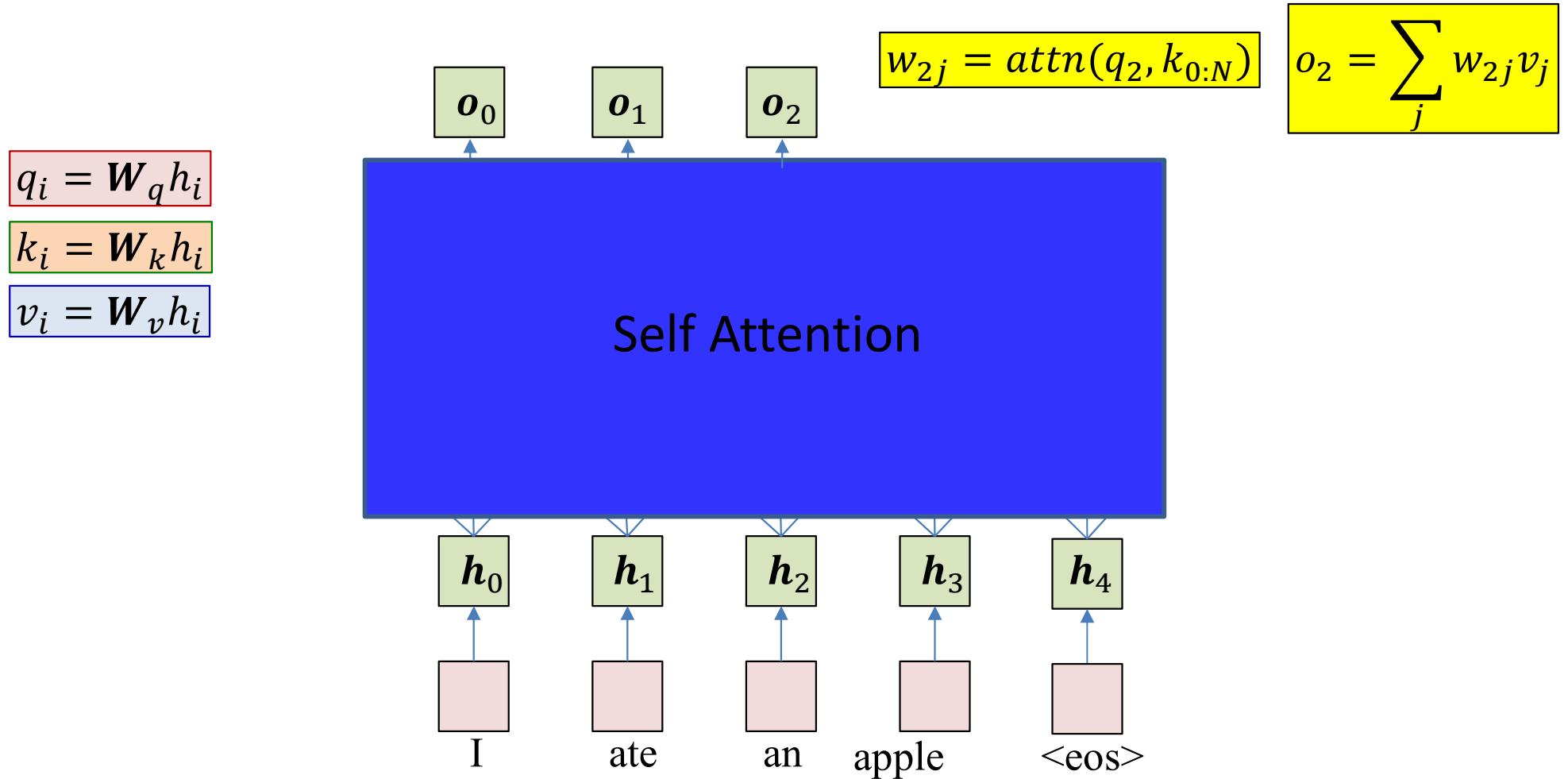
- The updated representation for the word is the attention-weighted sum of the values for all words
  - Including itself

$$w_{0j} = attn(q_0, k_{0:N})$$

$$o_0 = \sum_j w_{0j} v_j$$

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$v_i = W_v h_i$$

- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

$$w_{1j} = attn(q_1, k_{0:N})$$

$$o_1 = \sum_j w_{1j} v_j$$

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$v_i = W_v h_i$$

o_0 o_1

Self Attention

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   \<eos\>

- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

109

$$w_{2j} = attn(q_2, k_{0:N})$$

$$o_2 = \sum_j w_{2j} v_j$$

o_0 o_1 o_2

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$v_i = W_v h_i$$

Self Attention

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   <eos>

- Compute query-key-value sets for every word

- For each word

  – Using the query for that word, compute attention weights for all words using their keys

  – Compute updated representation for the word as attention-weighted sum of values of all words

$q_i = W_q h_i$

$k_i = W_k h_i$

$v_i = W_v h_i$

- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

$q_i = W_q h_i$

$k_i = W_k h_i$

$v_i = W_v h_i$

- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

112

$$w_{ij} = attn(q_i, k_{0:N})$$

$$q_i = W_q h_i$$

$$k_i = W_k h_i$$

$$v_i = W_v h_i$$

$$o_i = \sum_j w_{ij} v_j$$

- Compute query-key-value sets for every word
- For each word
  - Using the query for that word, compute attention weights for all words using their keys
  - Compute updated representation for the word as attention-weighted sum of values of all words

$$w_{ij} = attn(q_i, k_{0:N})$$

$$q_i = \boldsymbol{W}_q h_i$$

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

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

$$o_i = \sum_j w_{ij} v_j$$

| $\boldsymbol{o}_0$ | $\boldsymbol{o}_1$ | $\boldsymbol{o}_2$ | $\boldsymbol{o}_3$ | $\boldsymbol{o}_4$ |

Self Attention

| $\boldsymbol{h}_0$ | $\boldsymbol{h}_1$ | $\boldsymbol{h}_2$ | $\boldsymbol{h}_3$ | $\boldsymbol{h}_4$ |

I   ate   an   apple   \<eos\>
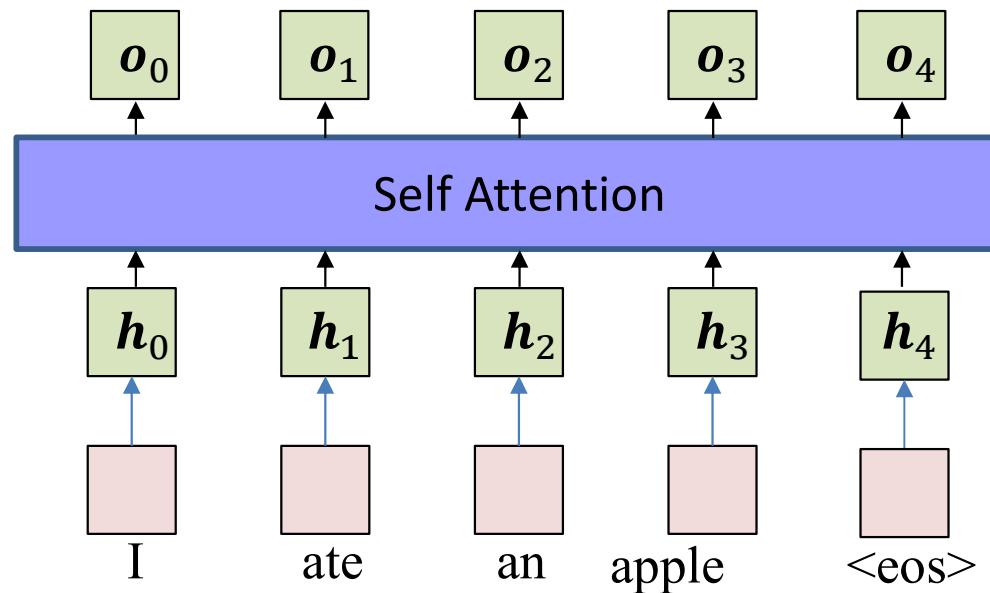
This is a "single-head" self-attention block

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$o_i = [o_i^1; o_i^2; o_i^3; \dots; o_i^H]$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

o_0 o_1 o_2 o_3 o_4

Attention head 0: $(q_i^0, k_i^0, v_i^0; W_q^0, W_k^0, W_v^0)$

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   <eos>
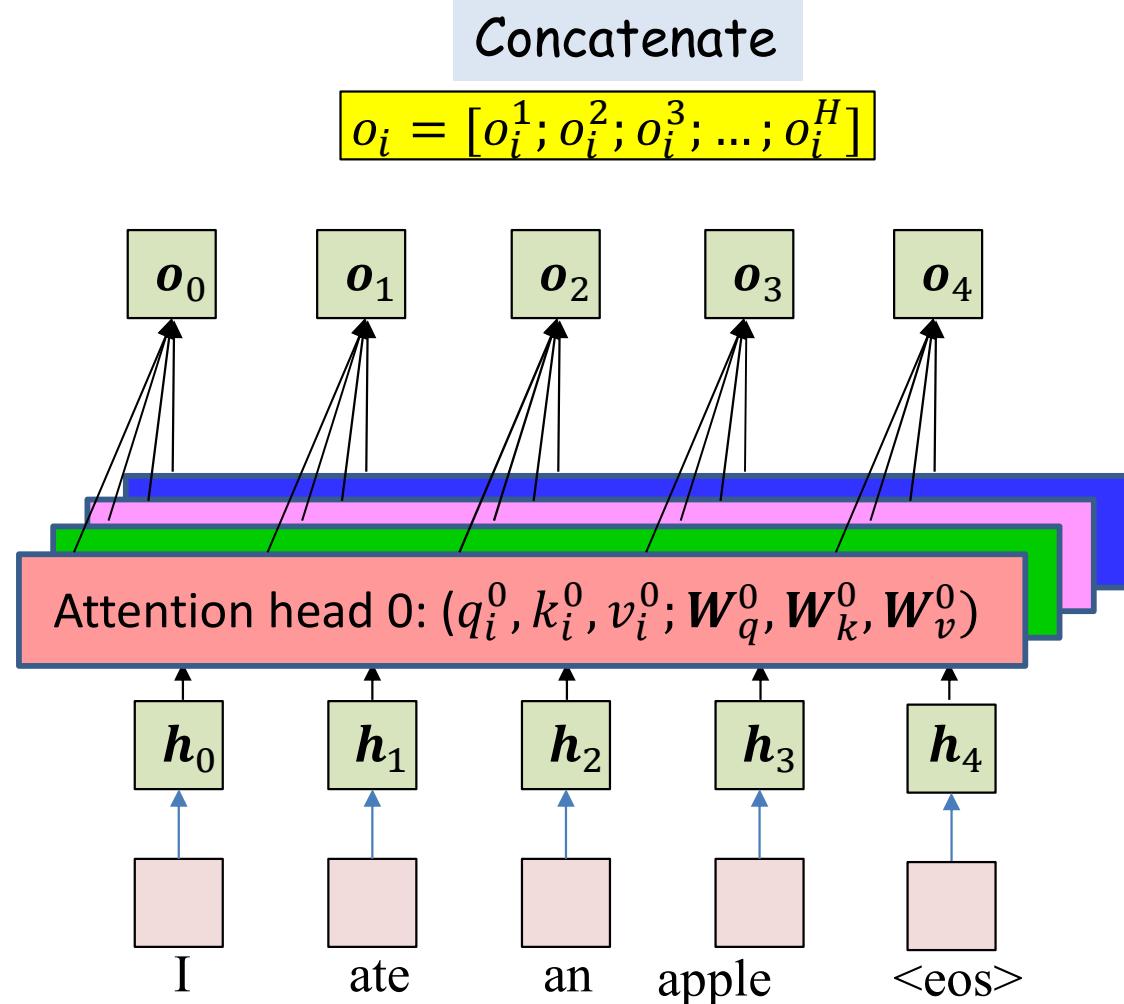
- We can have *multiple* such attention "heads"
  - Each will have an independent set of queries, keys and values
  - Each will obtain an independent set of attention weights
    - Potentially focusing on a different aspect of the input than other heads
  - Each computes an independent output
- The final output is the concatenation of the outputs of these attention heads
- "MULTI-HEAD ATTENTION" (actually Multi-head *self* attention)
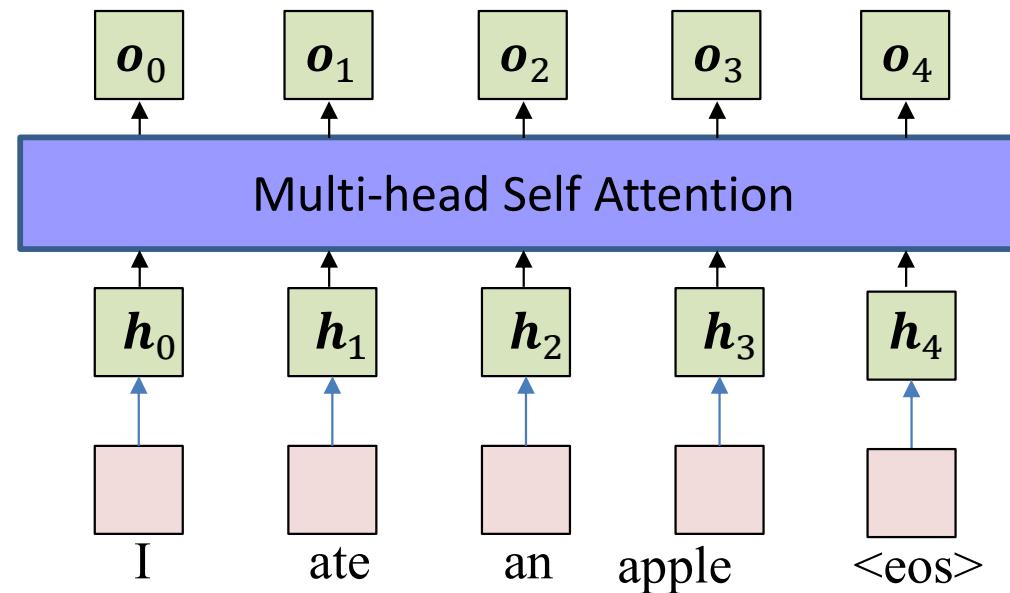
$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; ...; o_i^H]$$

| o_0 | o_1 | o_2 | o_3 | o_4 |

Multi-head Self Attention

| h_0 | h_1 | h_2 | h_3 | h_4 |

I   ate   an   apple   <eos>

- # Multi-head self attention
  - Multiple self-attention modules in parallel

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \dots; o_i^H]$$

$$y_i = MLP(o_i)$$

y_0 y_1 y_2 y_3 y_4

MLP

o_0 o_1 o_2 o_3 o_4

Multi-head Self Attention

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   <eos>
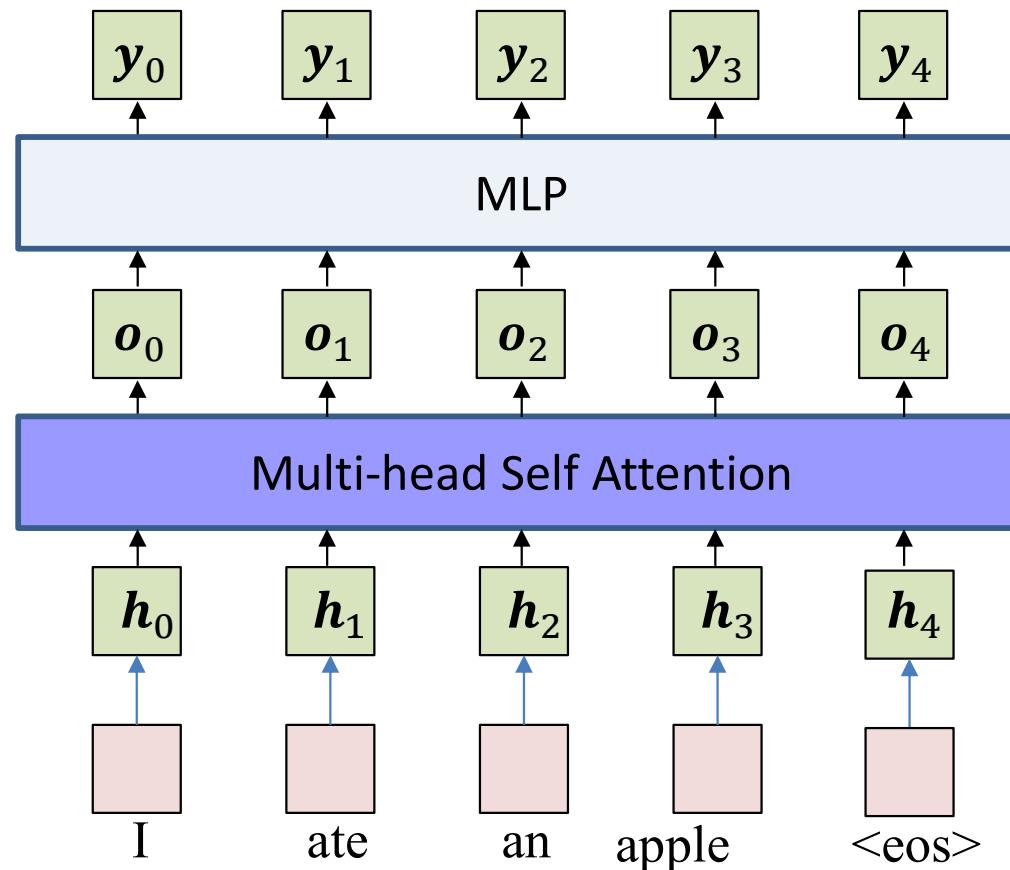
- Typically, the output of the multi-head self attention is passed through one or more regular feedforward layers
 - Affine layer followed by a non-linear activation such as ReLU

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$
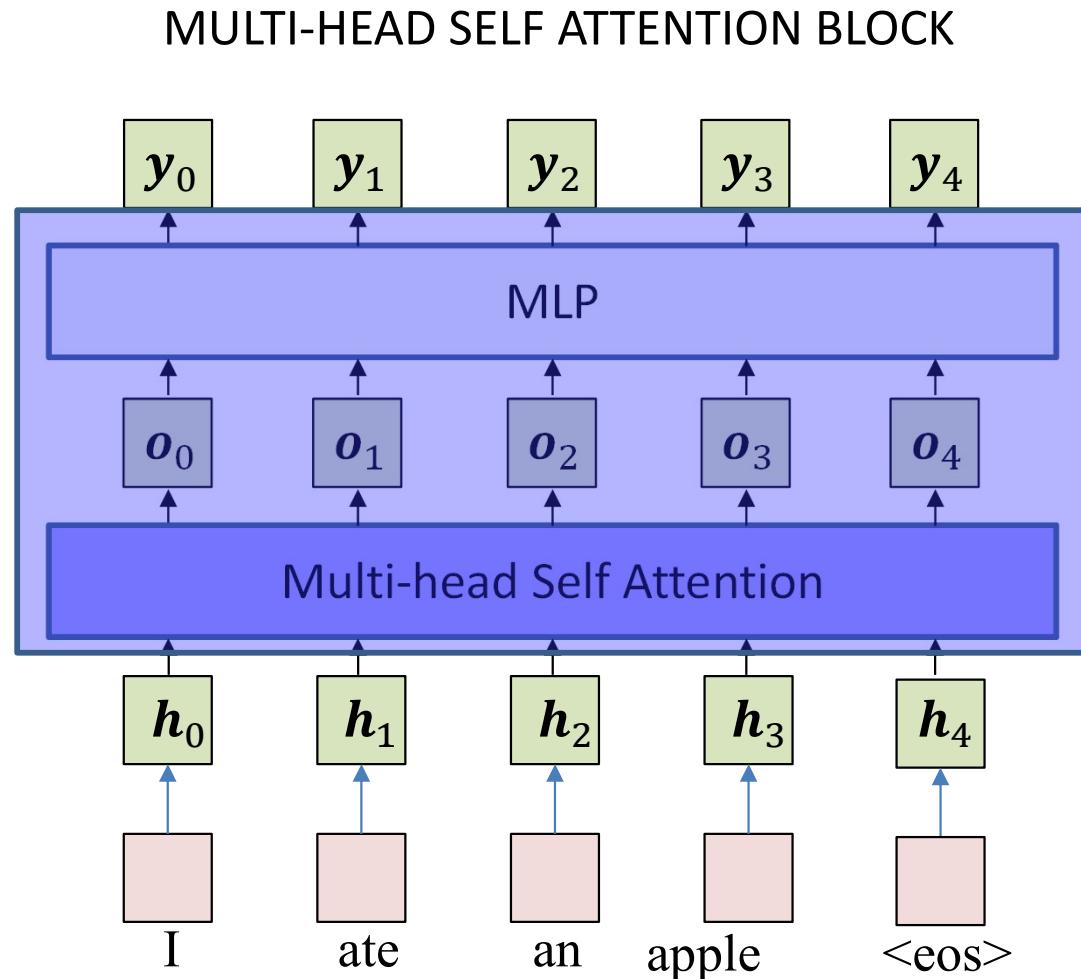
$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \dots; o_i^H]$$

$$y_i = MLP(o_i)$$

MULTI-HEAD SELF ATTENTION BLOCK

y_0 y_1 y_2 y_3 y_4

MLP

o_0 o_1 o_2 o_3 o_4

Multi-head Self Attention

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   <eos>

- The entire unit, including multi-head self-attention module followed by MLP is a ***multi-head self-attention block***

118

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:N}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

$$o_i = [o_i^1; o_i^2; o_i^3; \dots; o_i^H]$$

$$y_i = MLP(o_i)$$

Multi-head Self Attention Block

h_0 h_1 h_2 h_3 h_4

I   ate   an   apple   <eos>
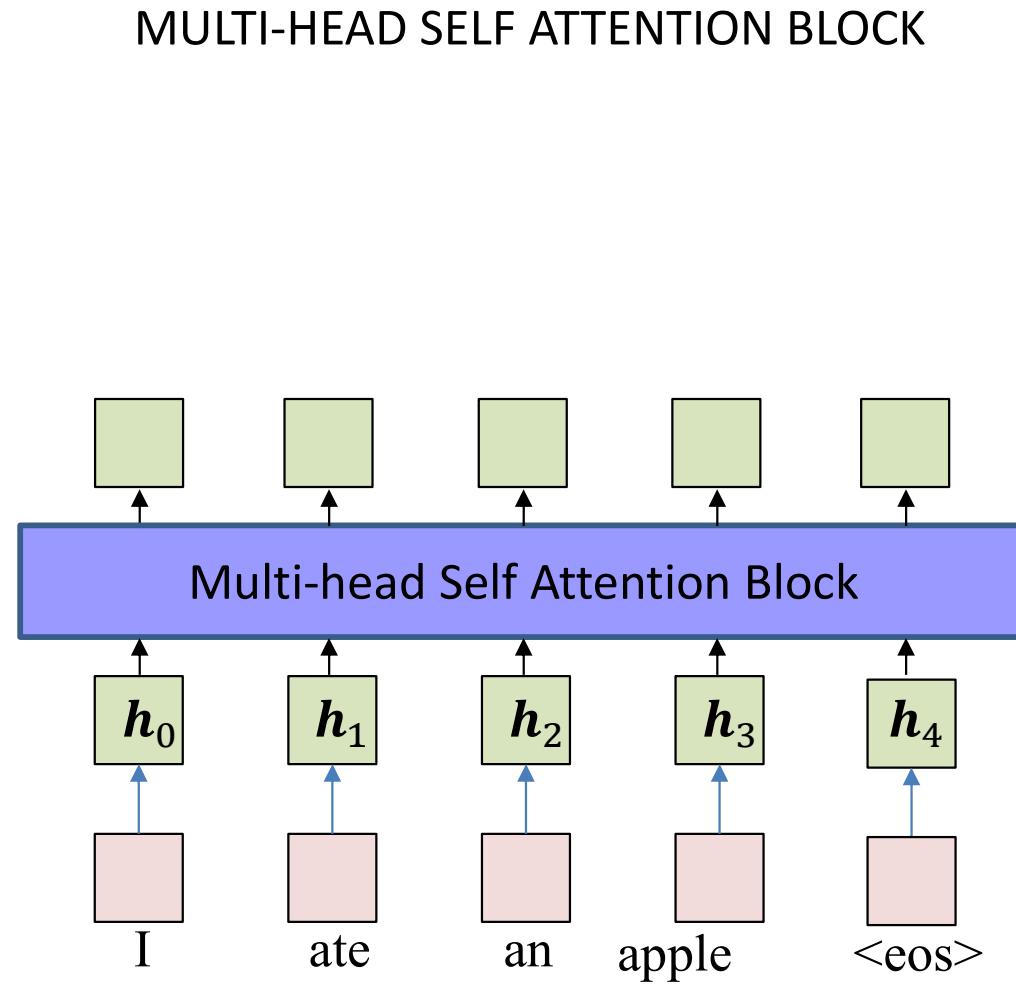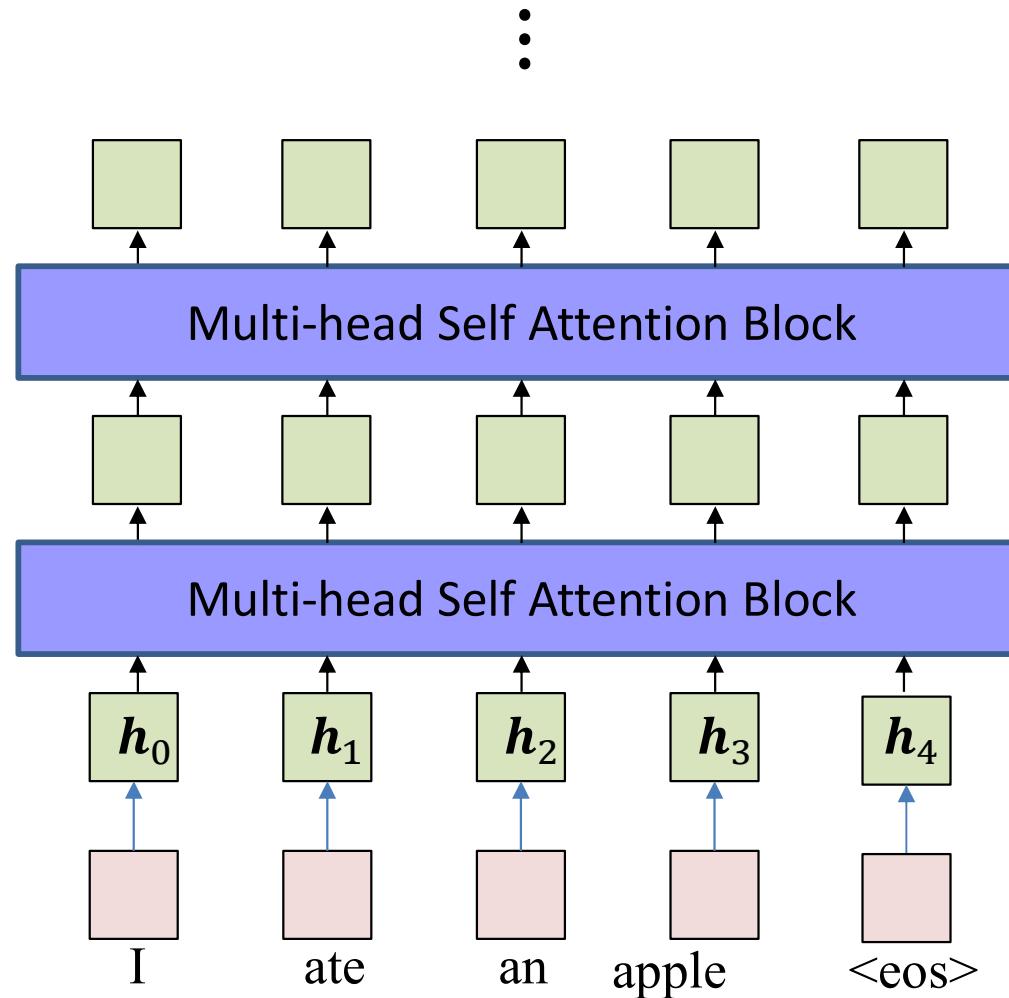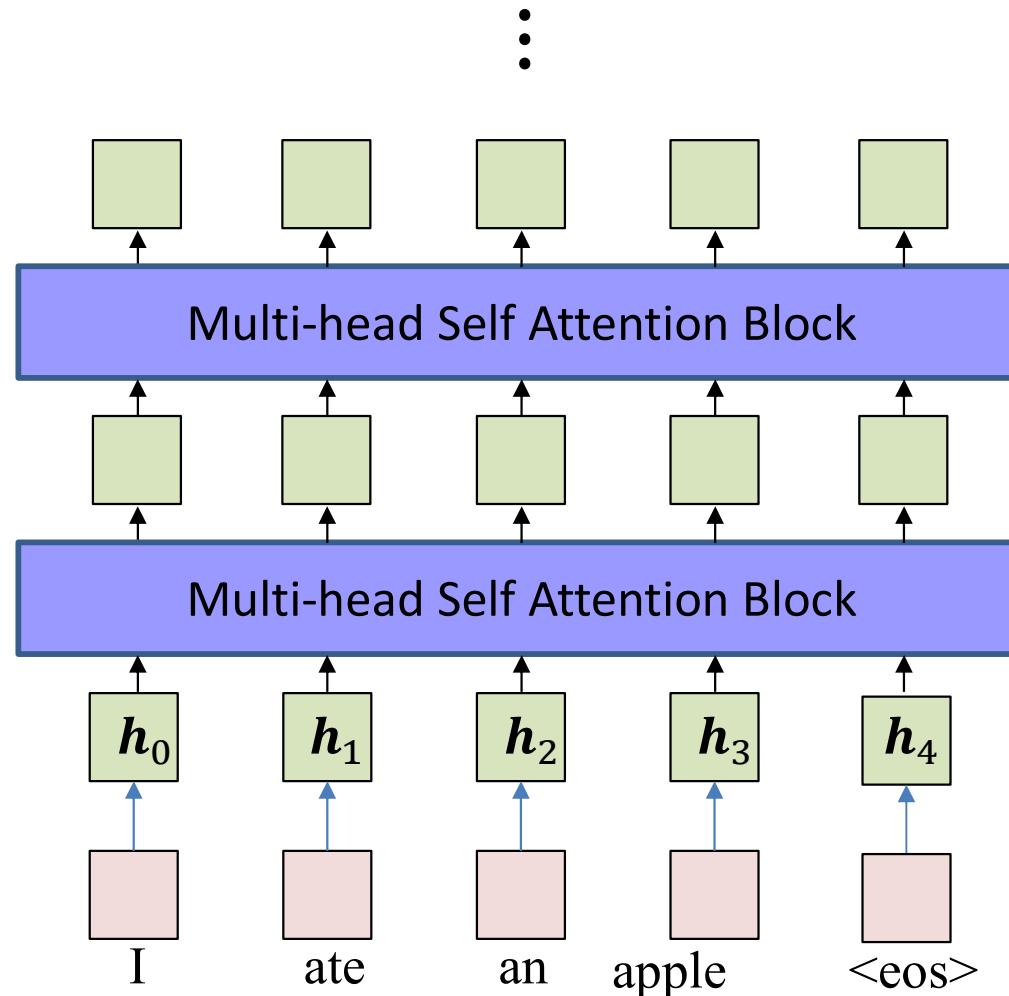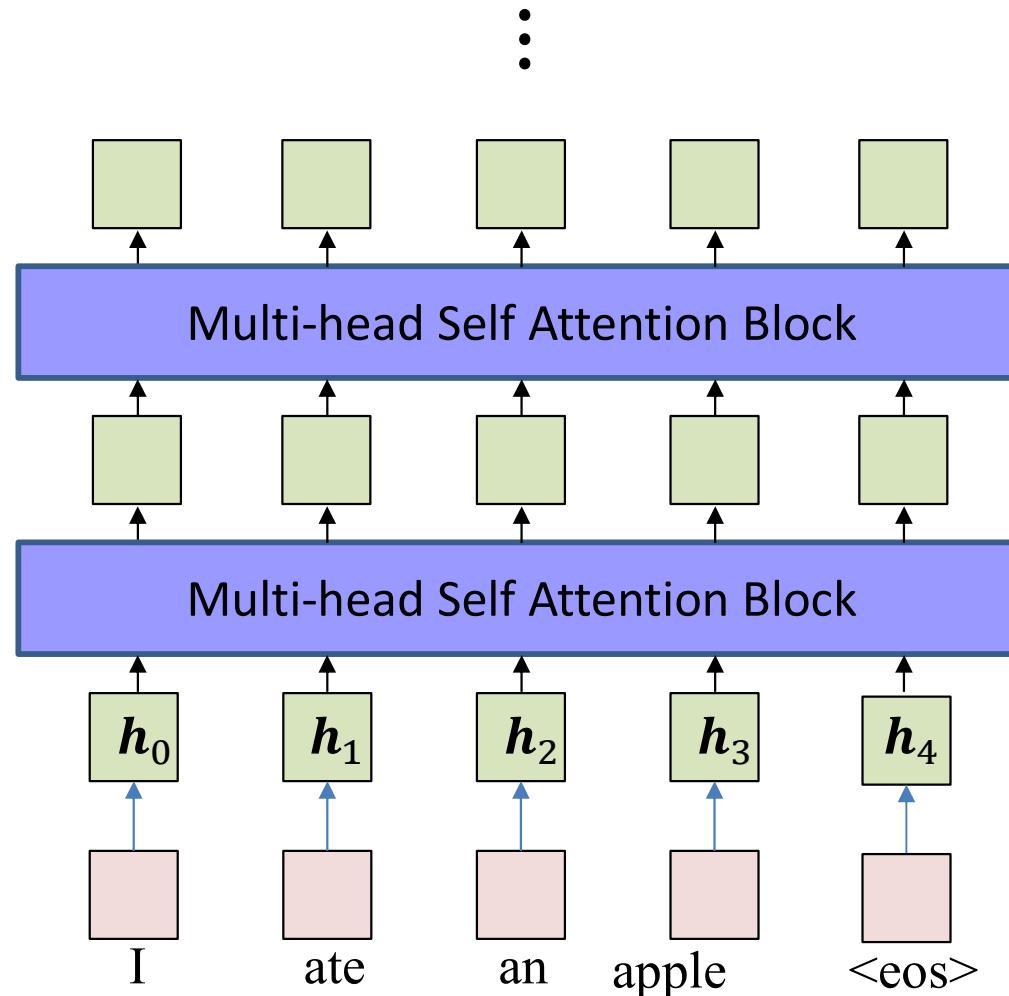
- The entire unit, including multi-head self-attention module followed by MLP is a ***multi-head self-attention block***

- The encoder can include many layers of such blocks
- No need for recurrence…
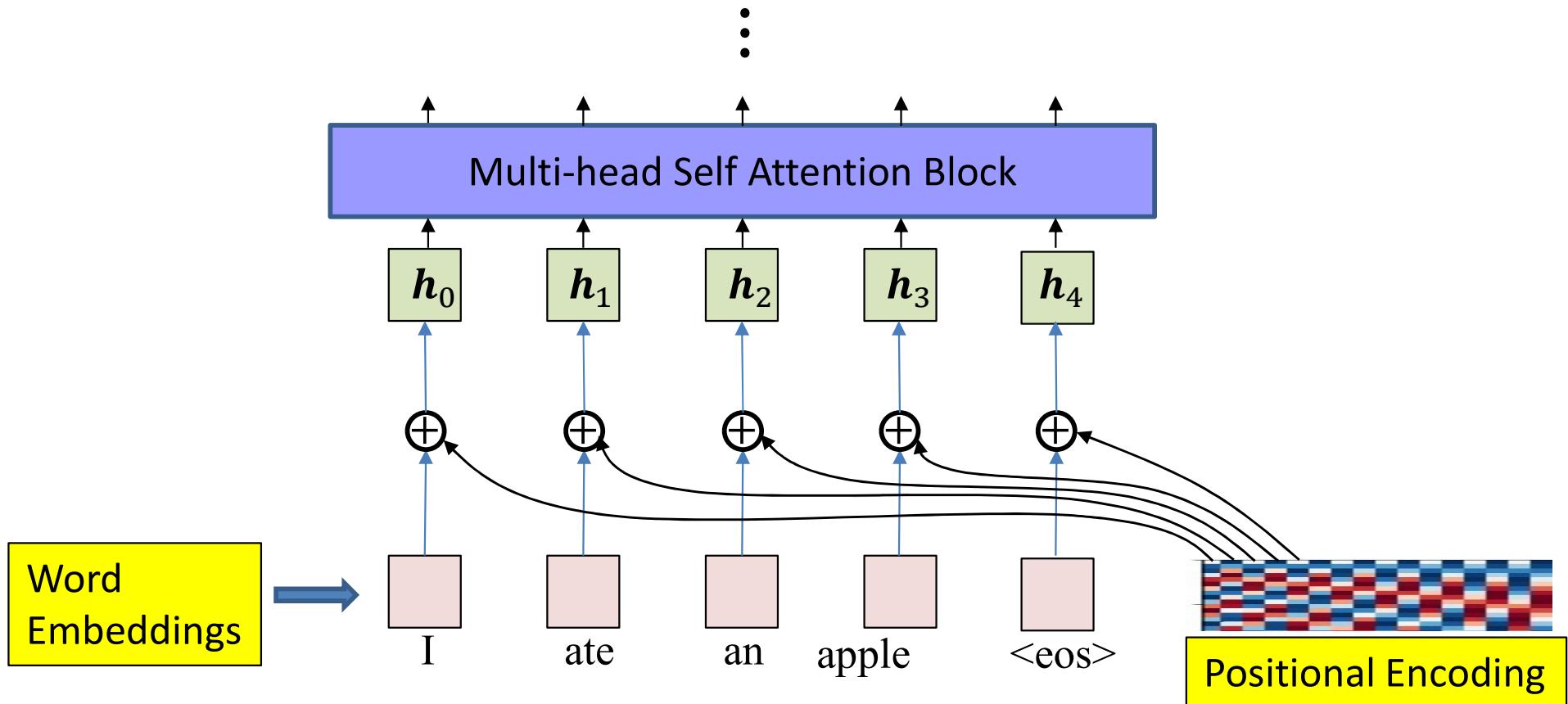
- Recap: The encoder in a sequence-to-sequence model can replace recurrence through a series of "multi-head self attention" blocks
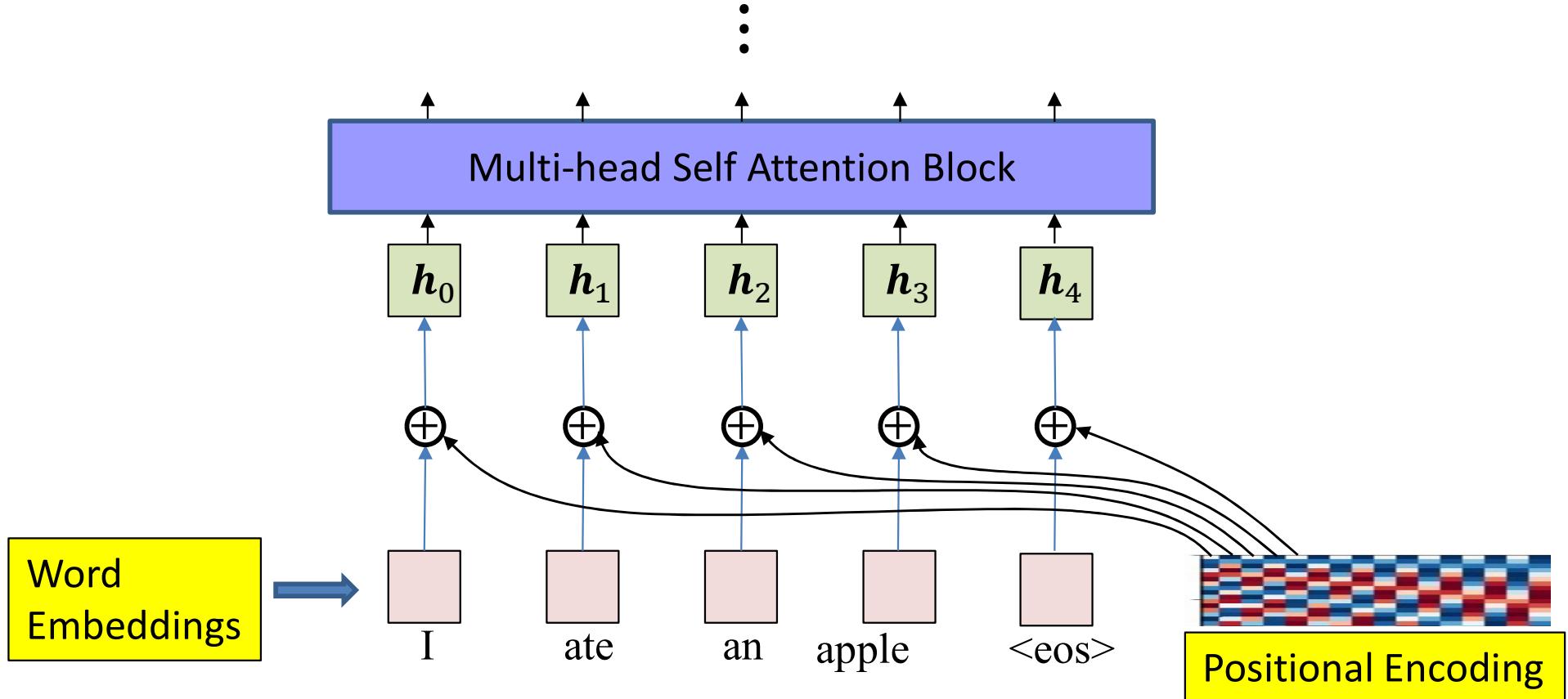
- Recap: The encoder in a sequence-to-sequence model can replace recurrence through a series of "multi-head self attention" blocks

- But this still ignores *relative position*
  - A context word one word away is different from one 10 words away
  - The attention framework does not take distance into consideration

⋮

Multi-head Self Attention Block

h_0 h_1 h_2 h_3 h_4

Word Embeddings →

I   ate   an   apple   <eos>

- Note that the inputs are actually word *embeddings*

- Note that the inputs are actually word *embeddings*

- We add a "positional" encoding to them to capture the relative distance from one another

- **Positional Encoding:** A sequence of vectors $P_0, \ldots, P_N$, to encode position
  - Every vector is unique (and uniquely represents time)
  - Relationship between P_t and $P_{t+\tau}$ only depends on the distance between them
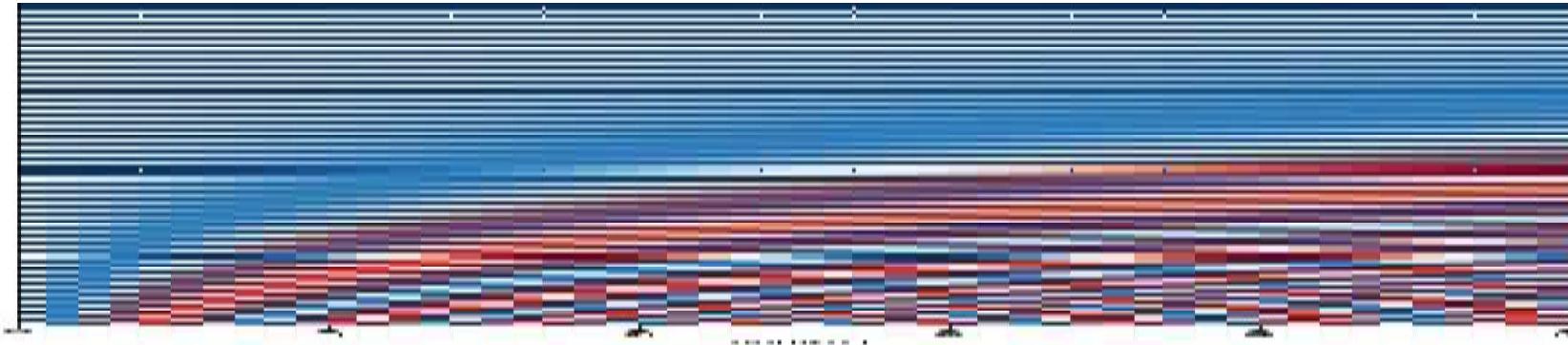
$$P_{t+\tau} = M_\tau P_t$$

- The linear relationship between P_t and $P_{t+\tau}$ enables the net to learn shift-invariant "gap" dependent relationships

Positional Encoding
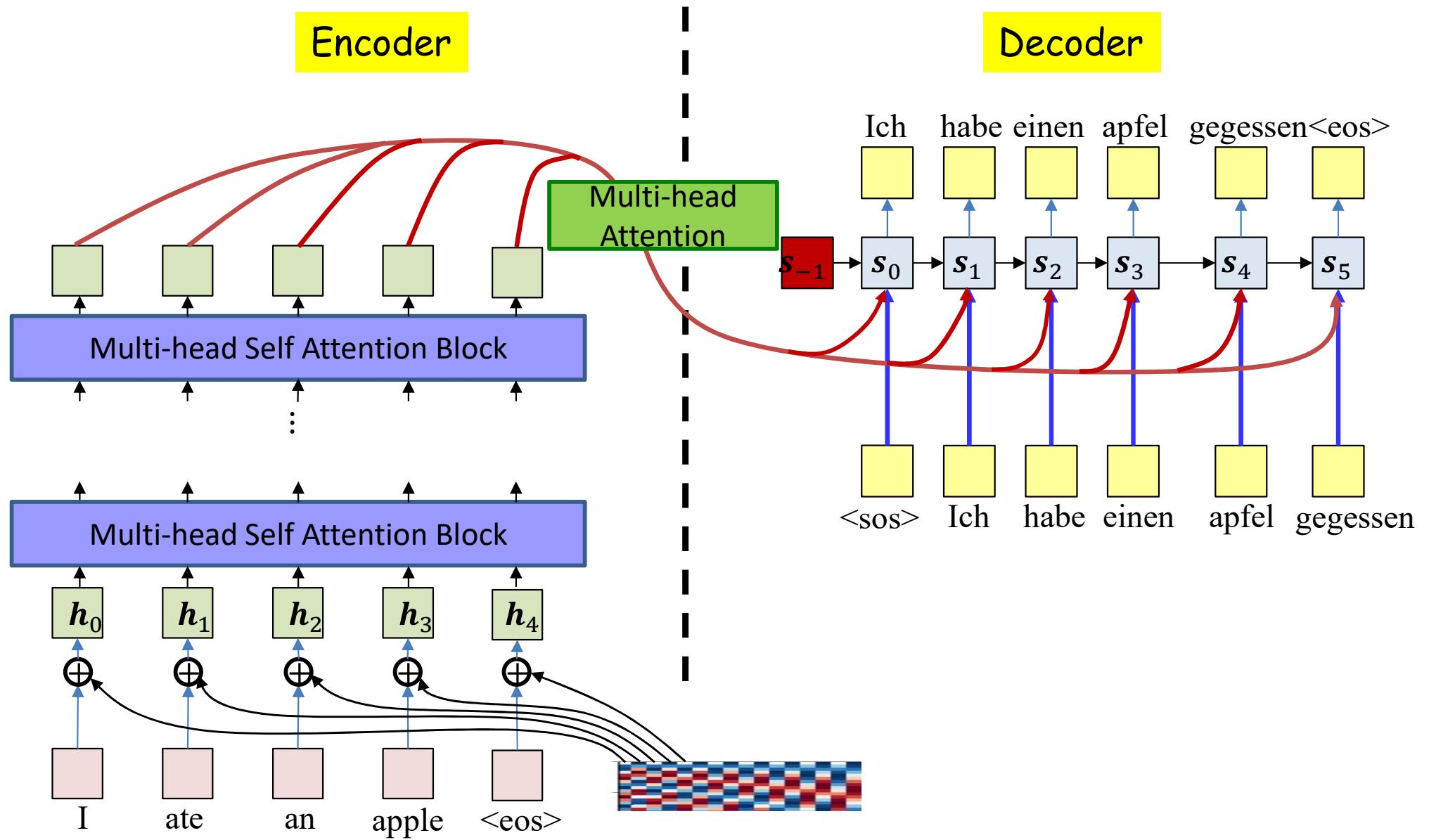
$$P_t = \begin{bmatrix} \sin \omega_1 t \\ \cos \omega_1 t \\ \sin \omega_2 t \\ \cos \omega_2 t \\ \vdots \\ \sin \omega_{d/2} t \\ \sin \omega_{d/2} t \end{bmatrix}$$

$$\omega_l = \frac{1}{10000^{2l/d}}$$

$$P_{t+\tau} = M_\tau P_t$$

$$M_\tau = diag \left(\begin{bmatrix} \cos \omega_l \tau & \sin \omega_l \tau \\ -\sin \omega_l \tau & \cos \omega_l \tau \end{bmatrix}, l = 1 \dots d/2 \right)$$
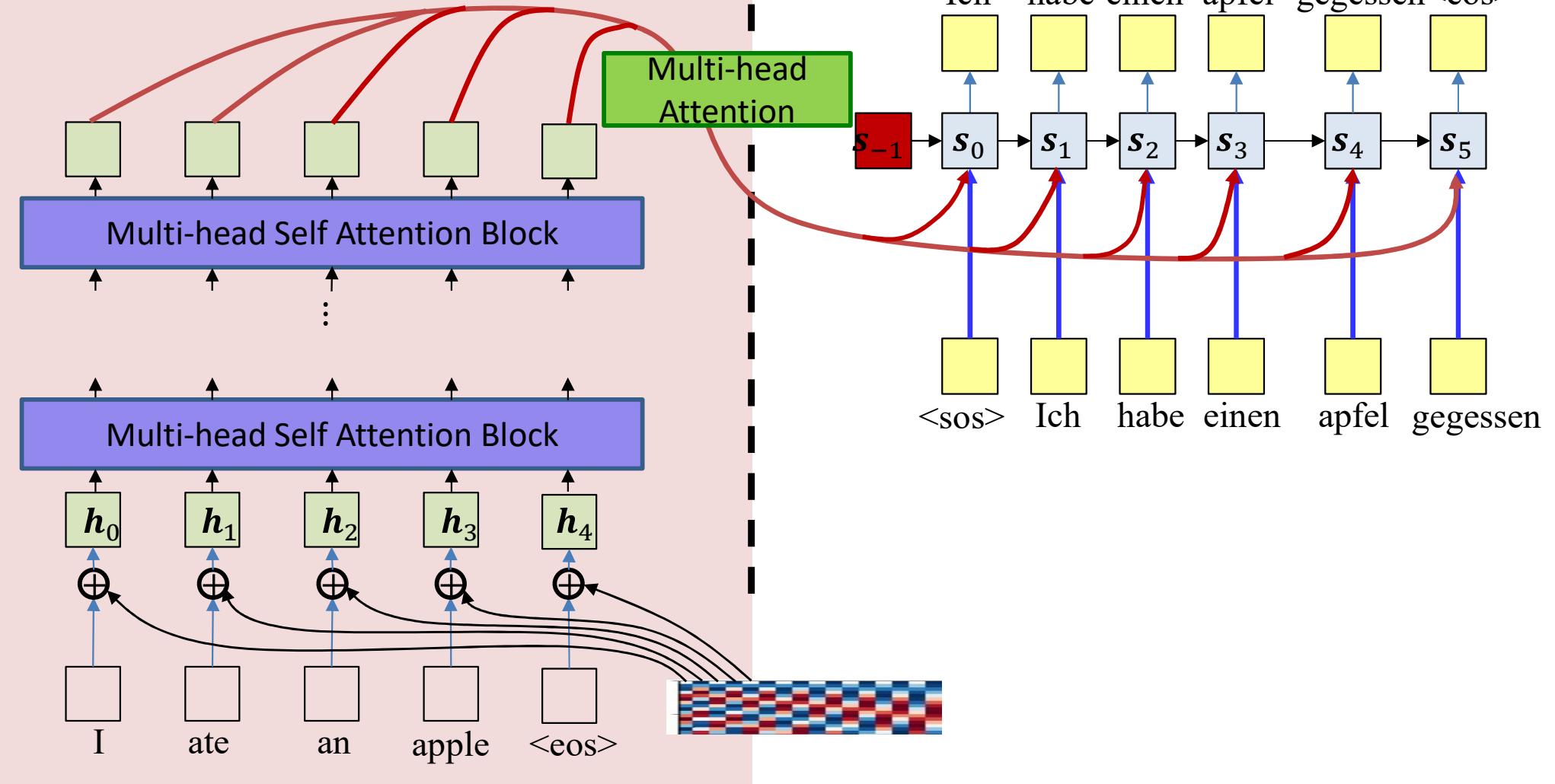
- A vector of sines and cosines of a harmonic series of frequencies
  - Every $2l$-th component of P_t is $\sin \omega_l t$
  - Every $2l+1$-th component of P_t is $\cos \omega_l t$
- Never repeats
- Has the linearity property required

126

- The linear relationship between P_t and $P_{t+\tau}$ enables the net to learn shift-invariant "gap" dependent relationships

- The self-attending encoder!!

Encoder

Decoder

Ich habe einen apfel gegessen <eos>

s_{-1} s_0 s_1 s_2 s_3 s_4 s_5

Multi-head Attention

<sos> Ich habe einen apfel gegessen

Multi-head Self Attention Block

Multi-head Self Attention Block

h_0 h_1 h_2 h_3 h_4

I ate an apple <eos>
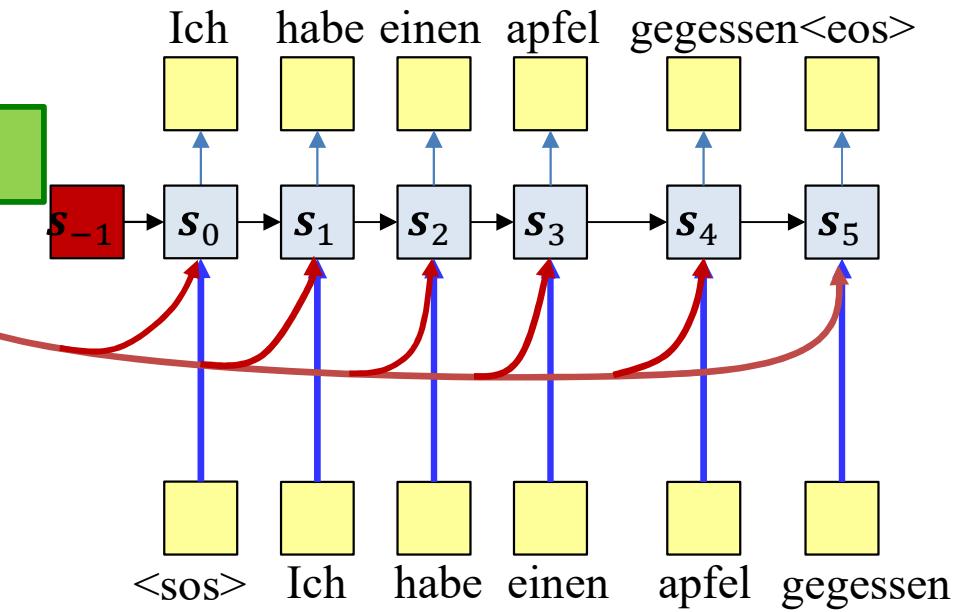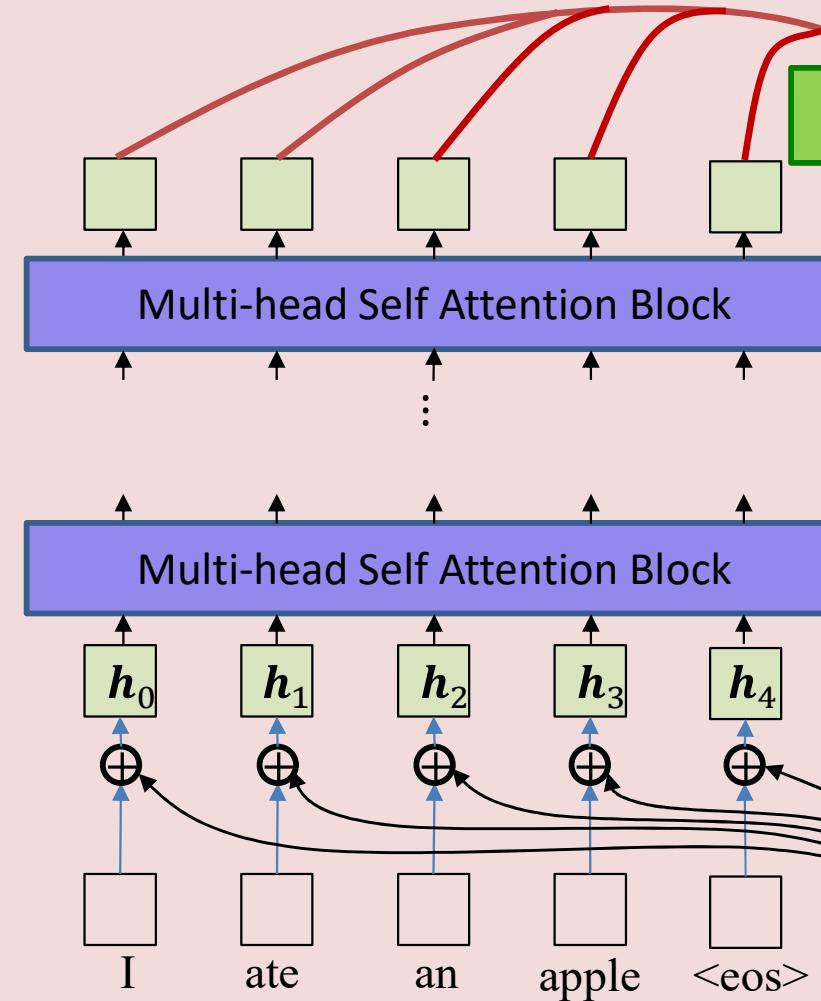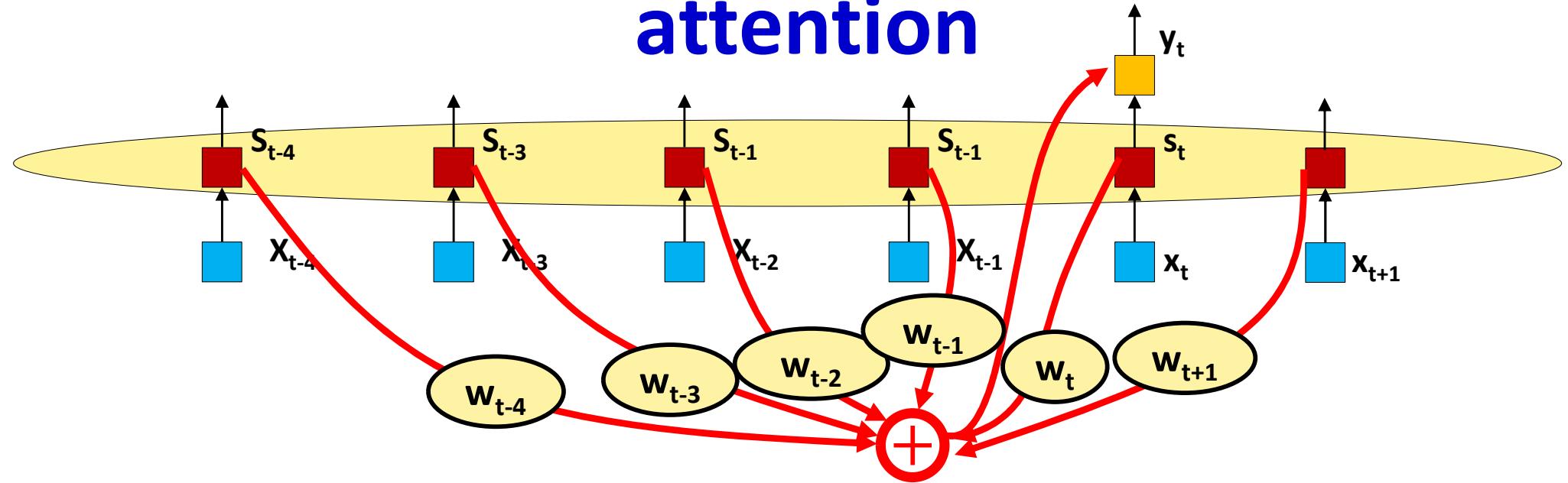
Can we use self attention to replace recurrence in the decoder?

- The self-attending encoder!!
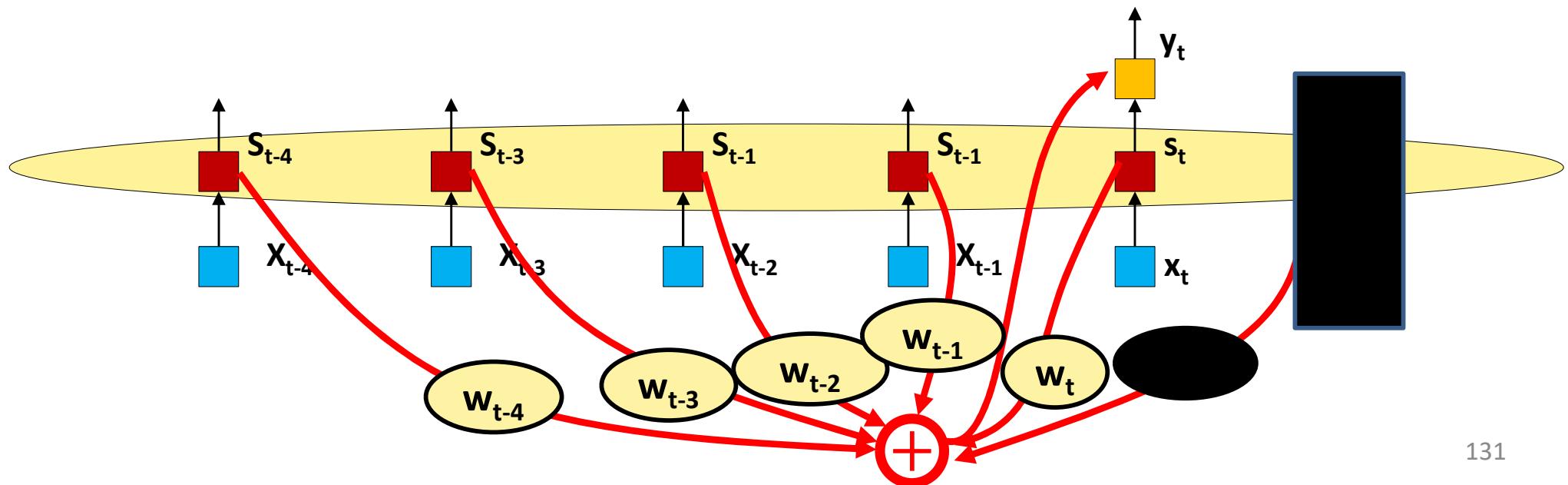
Self attention and masked self attention

- **Self attention in encoder:** Can use input embedding at time t+1 and further to compute output at time t, because all inputs are available

Self attention and masked self attention

- **Self attention in decoder:** Decoder is sequential
  - Each word is produced using the previous word as input
  - Only embeddings until time t are available to compute the output at time t

- The attention will have to be "masked", forcing attention weights for t+1 and later to 0

Masked self-attention block
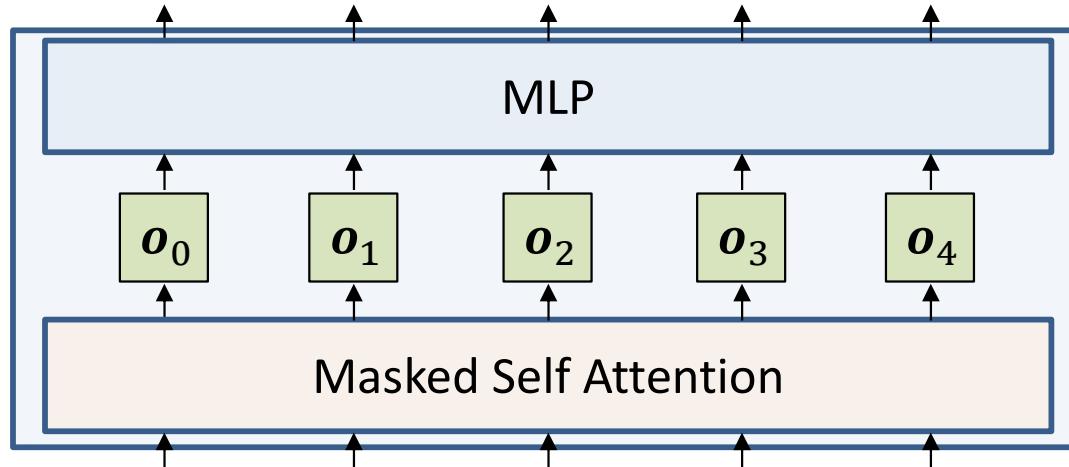
$$q_i = \boldsymbol{W}_q h_i$$

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

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

$$e_{ij} = q_i^T k_j$$

$$w_{i0}, \ldots, w_{ii} = softmax(e_{i0}, \ldots, e_{ii})$$

$$o_i = \sum_{j=0}^{i-1} w_{ij} v_j$$

```
                    ↑       ↑       ↑       ↑       ↑
┌─────────────────────────────────────────────────────────┐
│ ┌───────────────────────────────────────────────────┐   │
│ │                      MLP                            │   │
│ └───────────────────────────────────────────────────┘   │
│       ↑       ↑       ↑       ↑       ↑                   │
│    ┌─────┐ ┌─────┐ ┌─────┐ ┌─────┐ ┌─────┐               │
│    │ o₀  │ │ o₁  │ │ o₂  │ │ o₃  │ │ o₄  │               │
│    └─────┘ └─────┘ └─────┘ └─────┘ └─────┘               │
│       ↑       ↑       ↑       ↑       ↑                   │
│ ┌───────────────────────────────────────────────────┐   │
│ │              Masked Self Attention                  │   │
│ └───────────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────┘
        ↑       ↑       ↑       ↑       ↑
```

- The "masked self attention ***block***" includes an MLP after the masked self attention
  - Like in the encoder

Masked self-attention block

$$q_i = \boldsymbol{W}_q h_i$$

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

$$e_{ij} = q_i^T k_j$$

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

$$w_{i0}, \ldots, w_{ii} = softmax(e_{i0}, \ldots, e_{ii})$$

$$o_i = \sum_{j=0}^{i-1} w_{ij} v_j$$

Masked Self Attention block

- The "masked self attention **block**" sequentially computes outputs begin to end
  - Sequential nature of decoding prevents outputs from being computed in parallel
  - Unlike in an encoder

133

Masked multi-head self-attention

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

o_0 o_1 o_2 o_3 o_4

Masked attention head 0: $(q_i^0, k_i^0, v_i^0; W_q^0, W_k^0, W_v^0)$

- The "masked *multi-head* self attention *block*" includes multiple masked attention heads
  - Like in the encoder

Masked multi-head self-attention block

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

MLP

o_0 o_1 o_2 o_3 o_4

Masked attention head 0: $(q_i^0, k_i^0, v_i^0; W_q^0, W_k^0, W_v^0)$
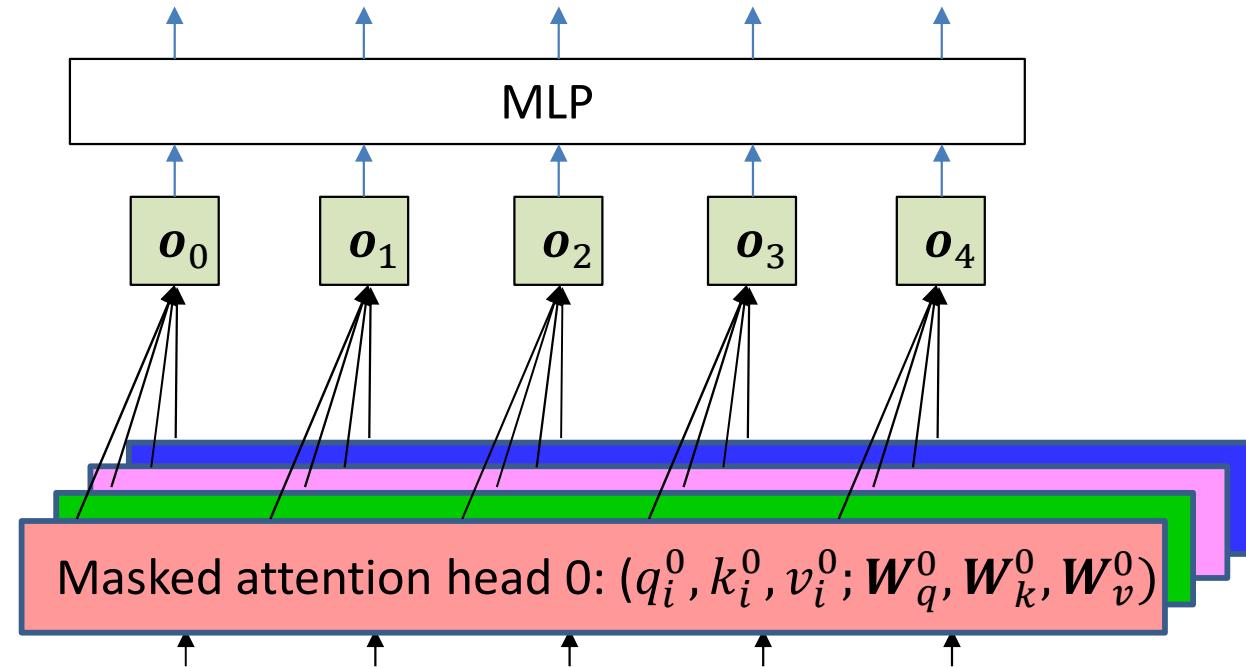
- The "masked *multi-head* self attention *block*" includes multiple masked attention heads
  - Like in the encoder

Masked multi-head self-attention block

$$q_i^a = W_q^a h_i$$

$$k_i^a = W_k^a h_i$$

$$v_i^a = W_v^a h_i$$

$$w_{ij}^a = attn(q_i^a, k_{0:i-1}^a)$$

$$o_i^a = \sum_j w_{ij}^a v_j^a$$

Masked Multi-head Self Attention block

- The "masked **multi-head** self attention **block**" includes multiple masked attention heads, followed by an MLP
  - Like in the encoder

Mark all that are true

- Self attention computed for an N-length input requires the computation of an N x N attention weight matrix for each head
- Masked self attention is only required in the first layer of the decoder. Subsequent layers see the entire output of the previous layers and can use full self attention
- We cannot combine recurrent layers with self attention layers
- Positional encodings are different in the encoder and decoder because the self attention in the decoder is masked.
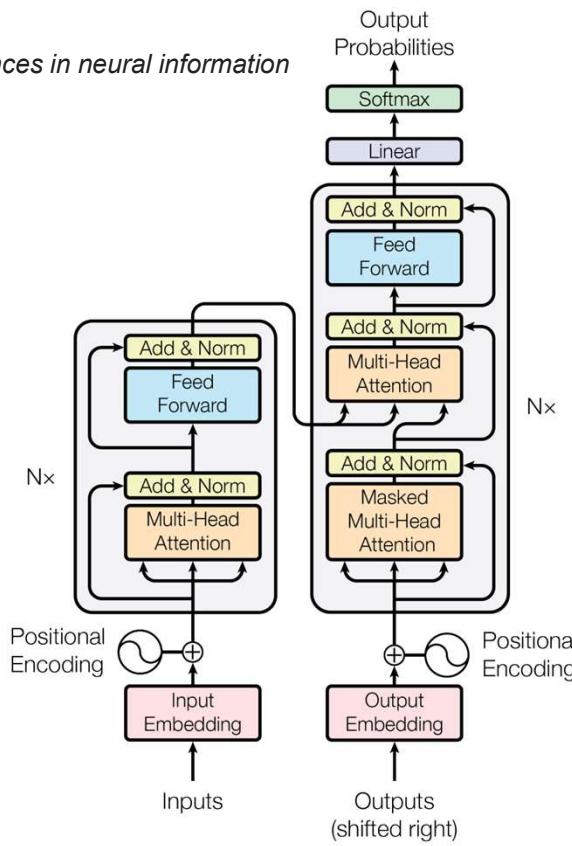
Poll 3

Mark all that are true

- **Self attention computed for an N-length input requires the computation of an N x N attention weight matrix for each head**
- Masked self attention is only required in the first layer of the decoder. Subsequent layers see the entire output of the previous layers and can use full self attention
- We cannot combine recurrent layers with self attention layers
- Positional encodings are different in the encoder and decoder because the self attention in the decoder is masked.
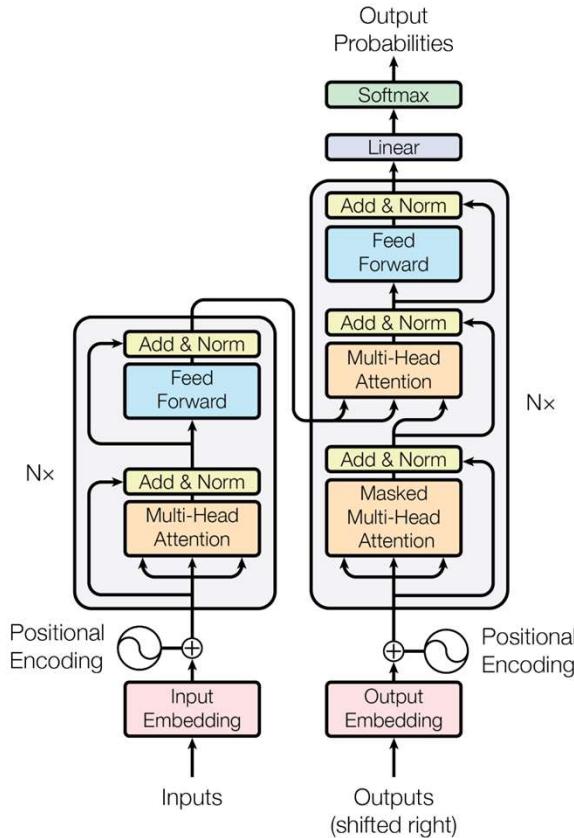
Transformer: Attention is all you need

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems*. 2017.

- Transformer: A sequence-to-sequence model that replaces recurrence with positional encoding and multi-head self attention
  - "Attention is all you need"

Transformer

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

Transformer

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**		$2.3 \cdot 10^{19}$

Why so good?   Why so fast?

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

142
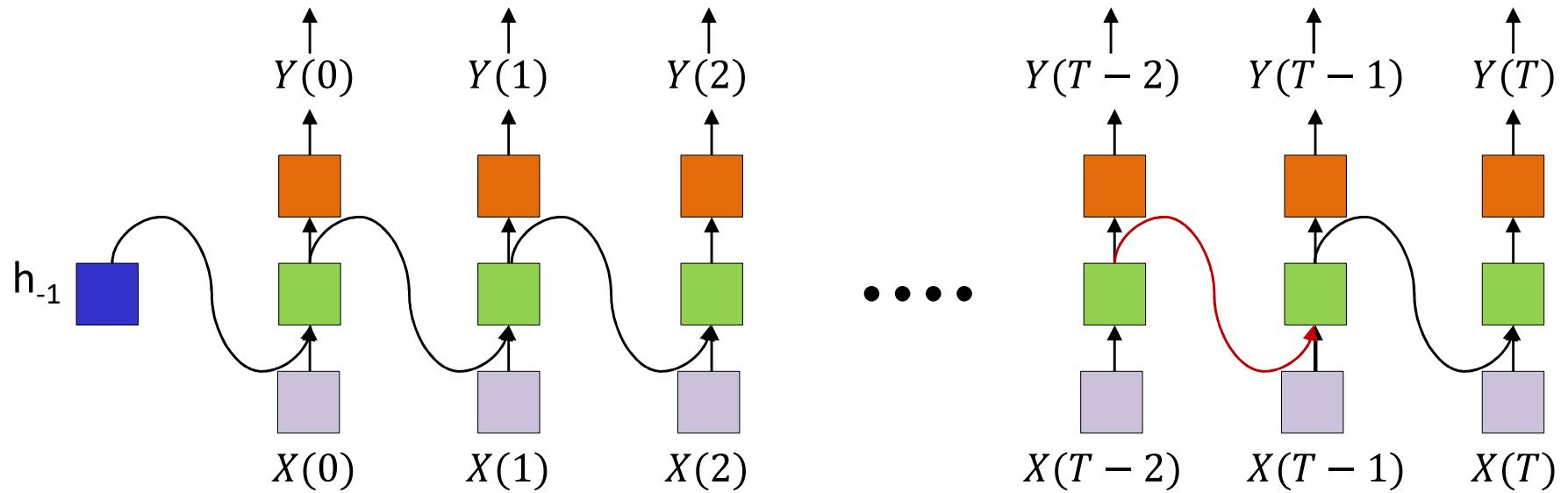
Recap: Vanishing/exploding gradients

output layer

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_N . \nabla f_{N-1} . W_{N-1} \dots \nabla f_{k+1} W_{k+1}$$
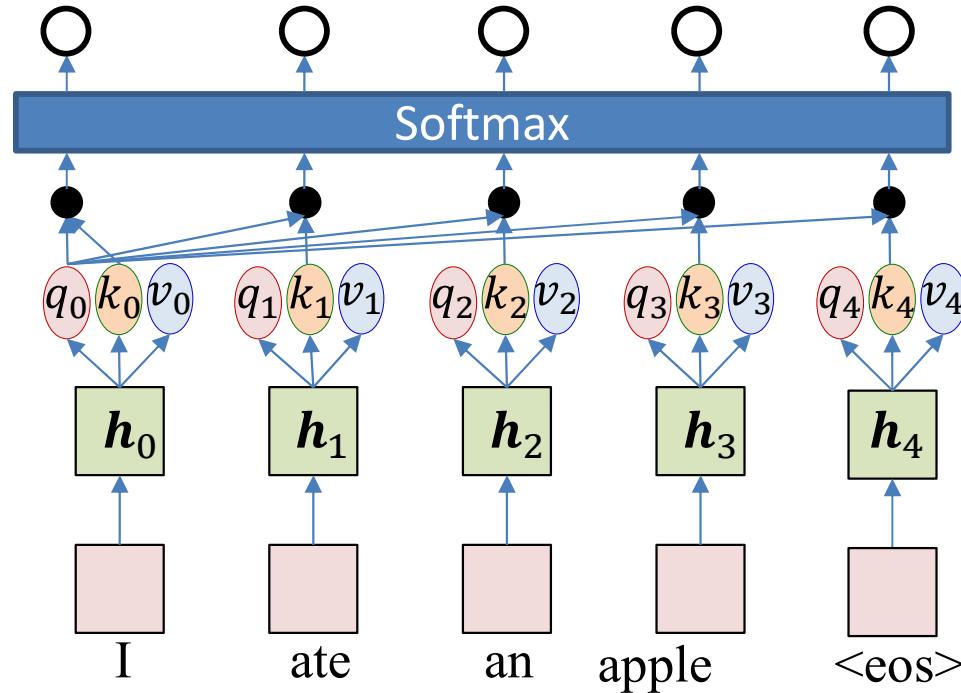
- RNNs are just very deep networks
- LSTMs mitigate the problem at the cost of 3x more matrix multiplications
- Transformers get rid of it! To encode a full sentence, they have way fewer layers than an unrolled RNN.
- The same goes with the vanishing memory issue to an extent.

Processing order
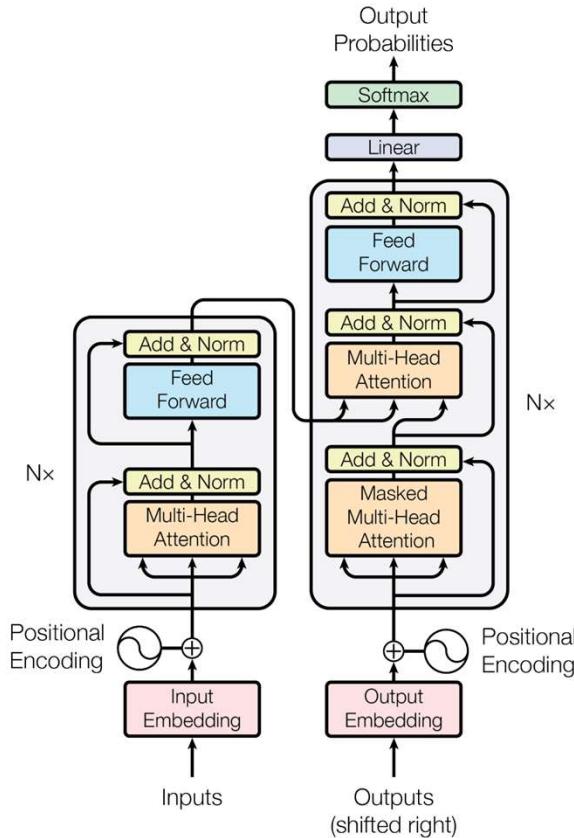
- Computing $Y(T)$ requires $Y(T-1)$...
- Which requires $Y(T-2)$, etc...
- RNN inputs must be processed *in order* → slow implementation

Processing order

- q_n, k_n, v_n can be computed separately.
- $n^2 \ <q_n, k_n> $ dot products to compute.
- Self attention is easy to compute in parallel $\rightarrow$ Faster implementations

Transformer
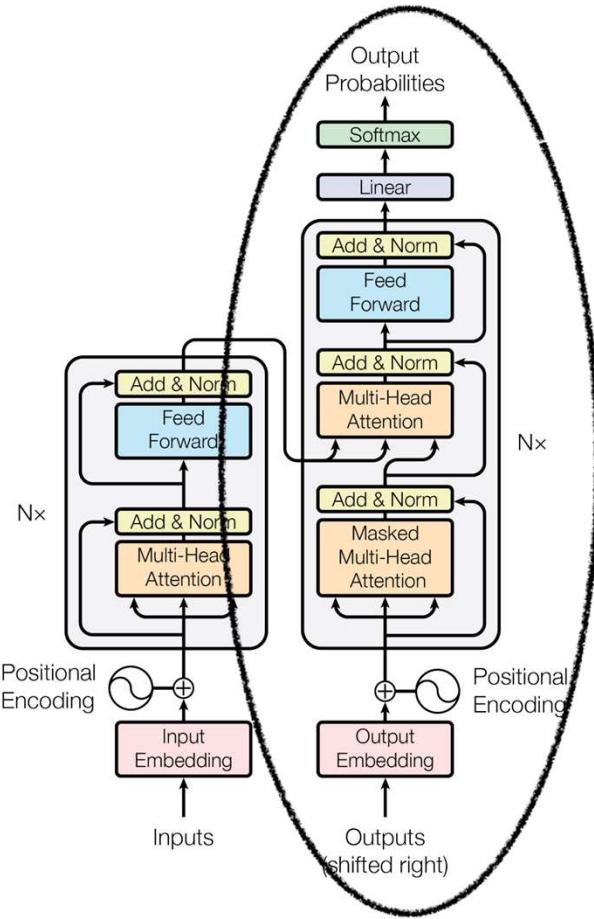
From "Attention is all you need"

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- Transformer: tremendous decrease in model computation for similar performance as state-of-art translation models
- The last row in the table shows transformer performance
- The final two columns show computational cost.

GPT

Alec Radford et. al., Improving Language Understanding by Generative Pre-Training

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

| Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

- ## GPT uses only the decoder of the transformer as an LM
  - "Transformer w/o aux LM"
- ## Large performance improvement in many tasks

GPT
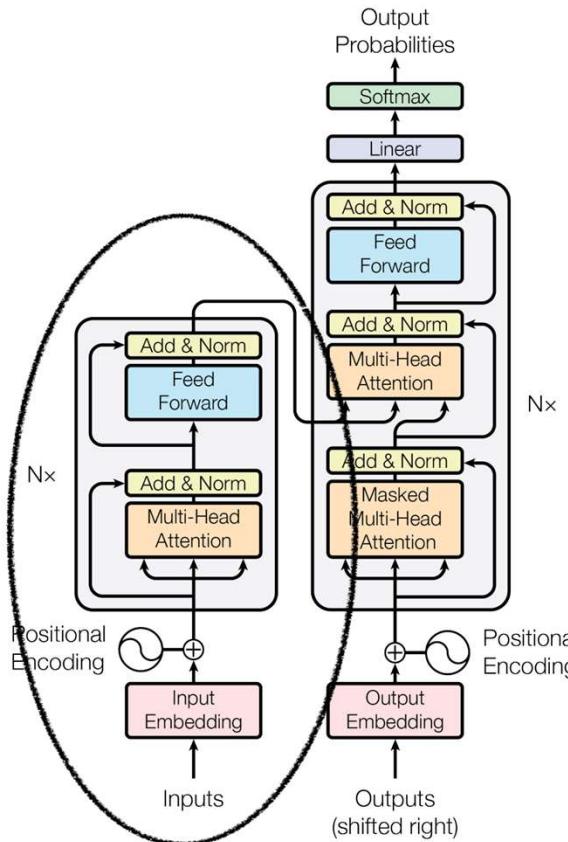
Alec Radford et. al., Improving Language Understanding by Generative Pre-Training

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

- Add *Task conditioning*: put the nature of your task in the input (not just LM)
- Parameters x1000
- → **GPT-3.5** : Generalizes to **more tasks**, not just more inputs!

148

BERT

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

- Bert: Only uses encoder of transformer to derive word and sentence embeddings
- Trained to "fill in the blanks"
- This is *representation learning* (more next lecture)

149

Attention is all you need

- Self-attention can effectively replace recurrence in sequence-to-sequence models
  - "Transformers"
  - Requires "positional encoding" to capture positional information

- Can also be used in regular sequence analysis settings as a substitute for recurrence

- Currently *the* state of the art in most sequence analysis/prediction…

Attention is all you need

- Self-attention can effectively replace recurrence in sequence-to-sequence models
  - "Transformers"
  - Requires "positional encoding" to capture positional information

- Can also be used in regular sequence analysis settings as a substitute for recurrence

- Currently *the* state of the art in most sequence analysis/prediction... and even computer vison problems!

Poll 4 (@1491)

Mark all that are true

- BERT is essentially the encoder of a transformer model
- GPT is essentially the encoder of a transformer model
- BERT is essentially the decoder of a transformer model
- GPT is essentially the decoder of a transformer model

Poll 4
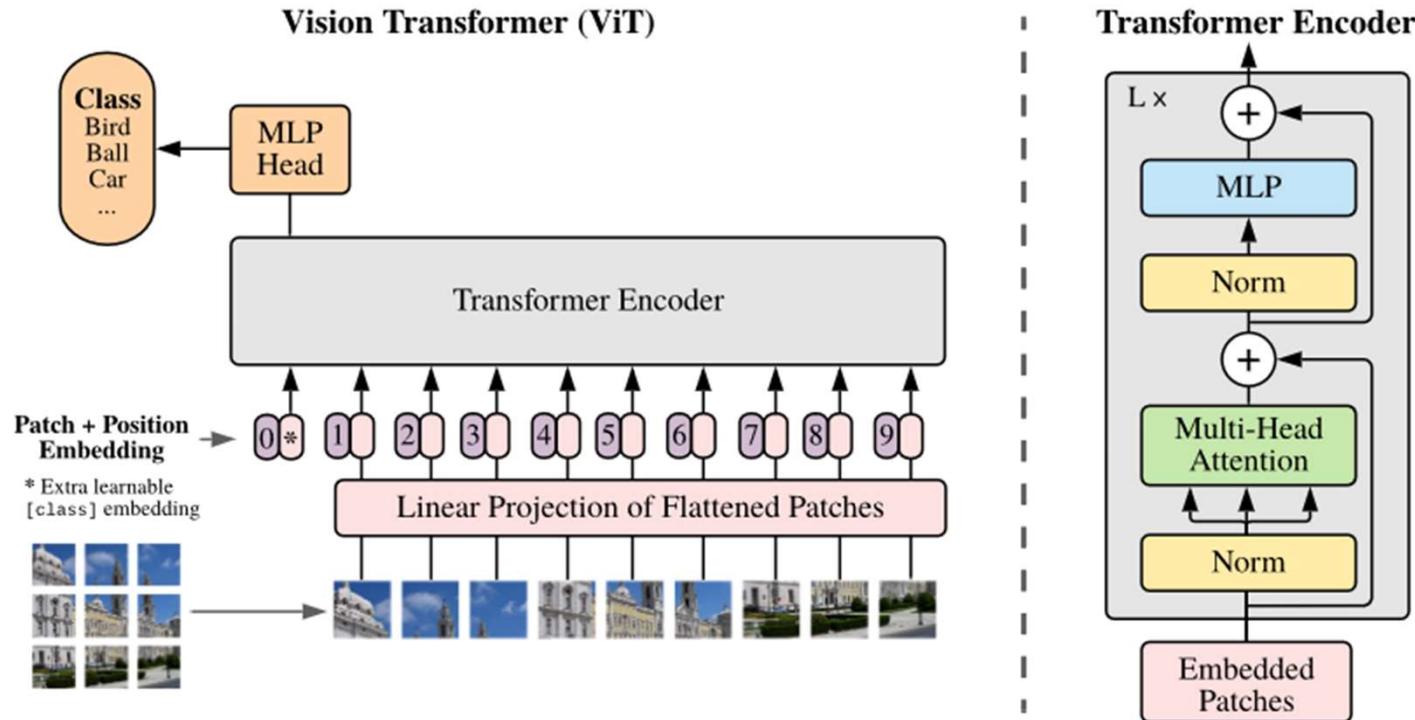
Mark all that are true

- **BERT is essentially the encoder of a transformer model**
- GPT is essentially the encoder of a transformer model
- BERT is essentially the decoder of a transformer model
- **GPT is essentially the decoder of a transformer model**

Next week

- More on transformers
  - Josh and Kate
- LLMs
  - Roshan Sharma from Google…

Vision Transformers
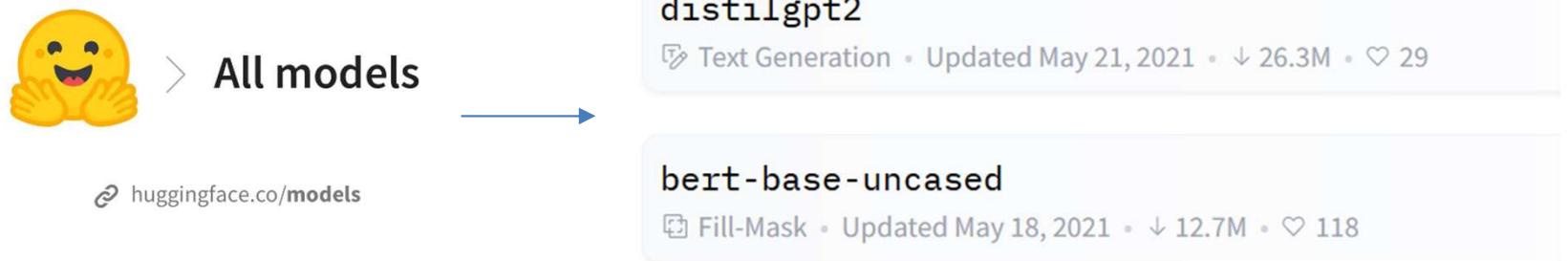
Dosovitskiy et al, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, 2020

- Divide your image in patches with pos. encodings
- Apply Self-Attention!

→ Sequential and image problems are similar when using transformers

Impact of Transformers

- Transformers have played a major role in the "uniformization" of DL-based tasks:
  - Find a pretrained "BERT-like" transformer (Text, Image, Speech)
  - Fine-tune on your task – or not! (Prompting…)
- This has helped democratize Deep Learning considerably

All models

huggingface.co/**models**

distilgpt2
Text Generation · Updated May 21, 2021 · ↓ 26.3M · ♡ 29

bert-base-uncased
Fill-Mask · Updated May 18, 2021 · ↓ 12.7M · ♡ 118

- **But…**

Caveat 1

- Not all transformers are the same: Big/small, fast/slow, mono-/multilingual, contrastive/generative, regressive/autoencoding…

- Pick the right one!

Caveat 2

- Transformers are not always the right choice.

- They often require more parameters than LSTMs at equal performance

→ Tricky on small hardware (phones, IoT, etc)