11-785 Recitation 10
Attention, MT, LAS

Rucha Kulkarni

Muhammed Danso

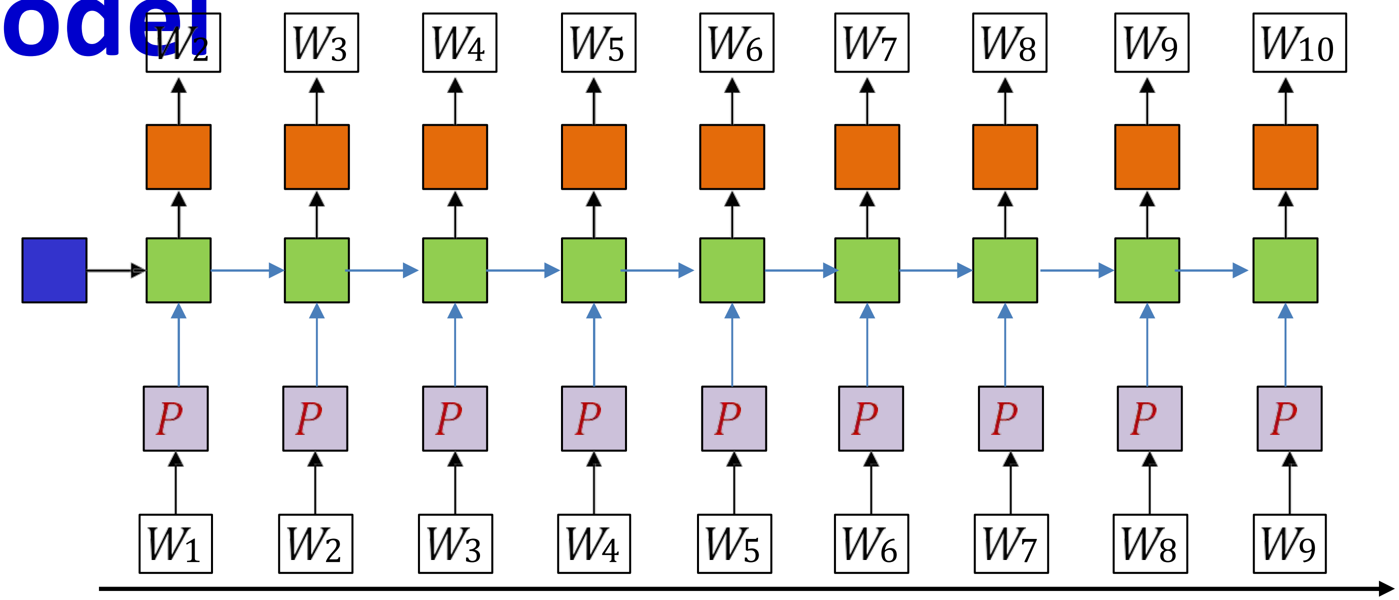
Sequence to sequence

 → Seq2seq → I ate an apple

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.”

Generating Language: The model



- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the "vocabulary"
  - Projected down to lower-dimensional "embeddings"
- The hidden units are (one or more layers of) LSTM units
- Output at each time: A probability distribution for the next word in the sequence
- All parameters are trained via backpropagation from a lot of text

A note on beginnings and ends

- A sequence of words by itself does not indicate if it is a complete sentence or not

  … four score and eight …

  – Unclear if this is the *start* of a sentence, the *end* of a sentence, or *both* (i.e. a complete sentence)

- To make it explicit, we will add two additional symbols (in addition to the words) to the base vocabulary
  – **<sos>** : Indicates start of a sentence
  – **<eos>** : Indicates end of a sentence

A note on beginnings and ends

- Some examples:

  four score and eight

  - This is clearly the middle of sentence

  <sos> four score and eight

  - This is a fragment from the *start* of a sentence

  four score and eight <eos>

  - This is the *end* of a sentence

  <sos> four score and eight <eos>

  - This is a full sentence

- In situations where the start of sequence is obvious, the <sos> may not be needed, but <eos> is required to terminate sequences

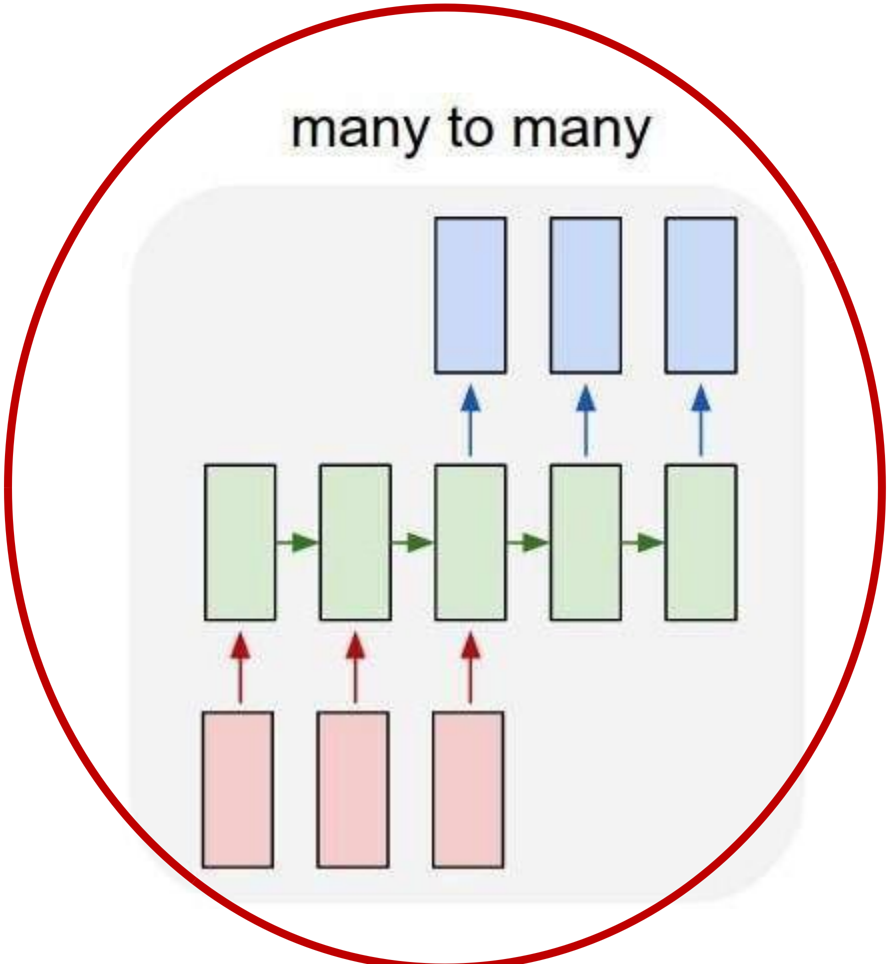
- Sometimes we will use a single symbol to represent both start and end of sentence, e.g just <eos> , or even a separate symbol, e.g. <s>

Returning to our problem

I ate an apple → Seq2seq → Ich habe einen apfel gegessen

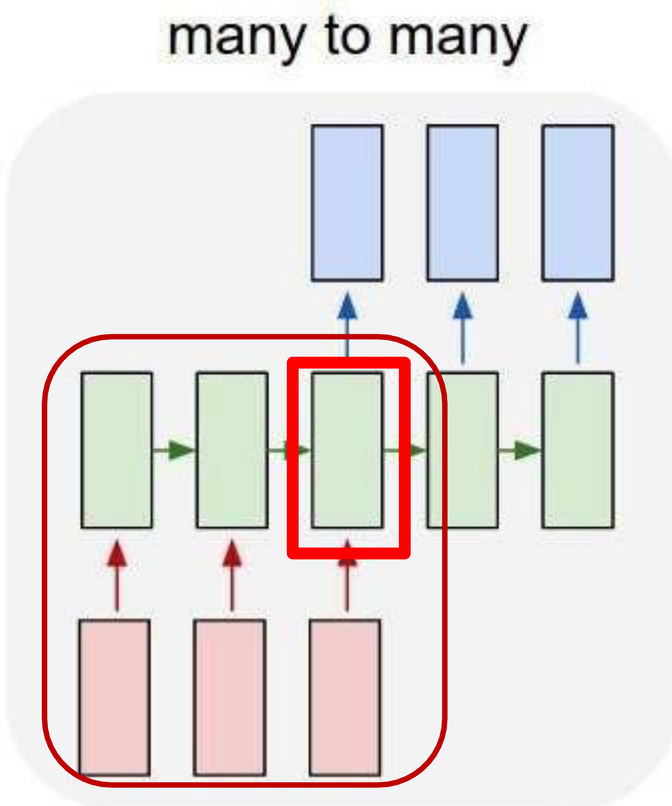
- Problem:
  - A sequence $X_1 \dots X_N$ goes in
  - A different sequence $Y_1 \dots Y_M$ comes out
- No expected synchrony between input and output

Modelling the problem



- *Delayed* sequence to sequence

Modelling the problem

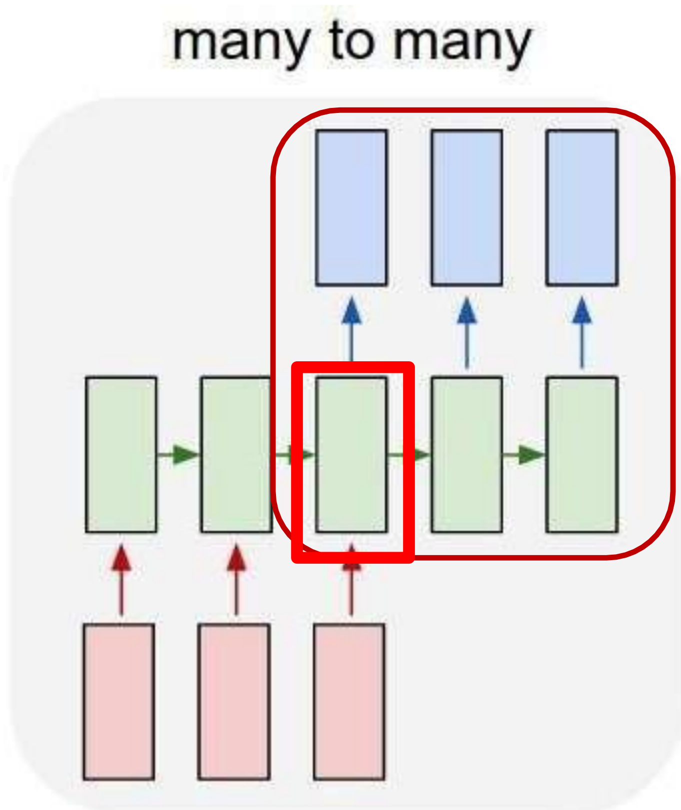


First process the input and generate a hidden representation for it

- *Delayed* sequence to sequence

Modelling the problem

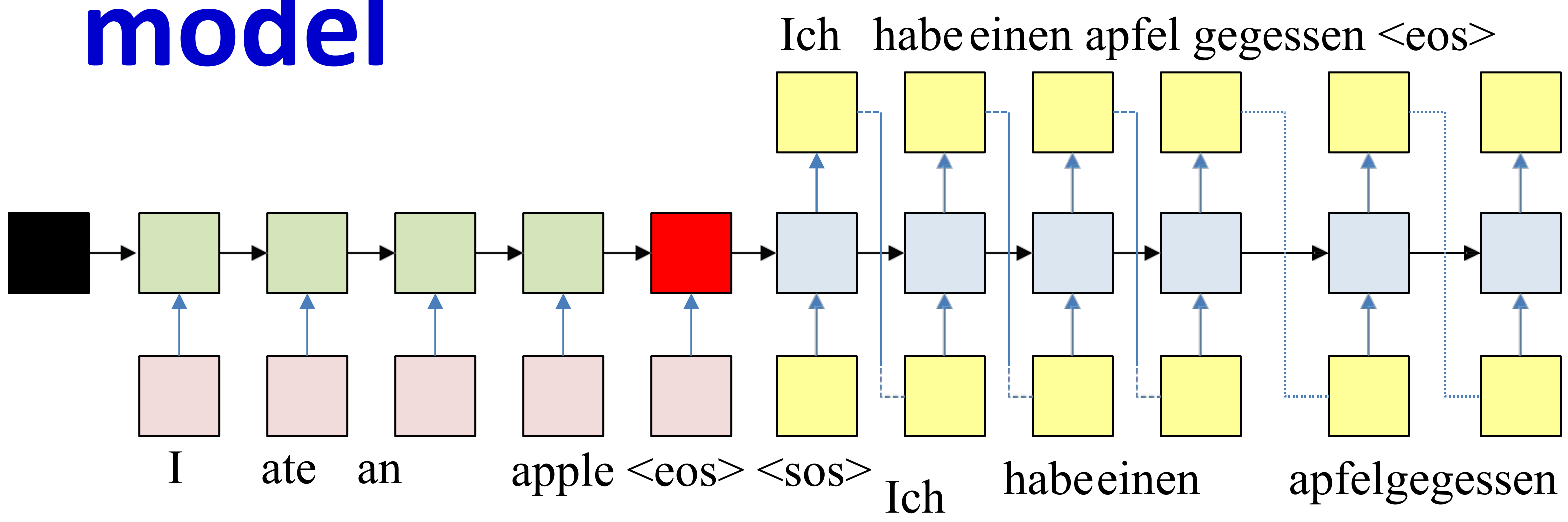
First process the input and generate a hidden representation for it



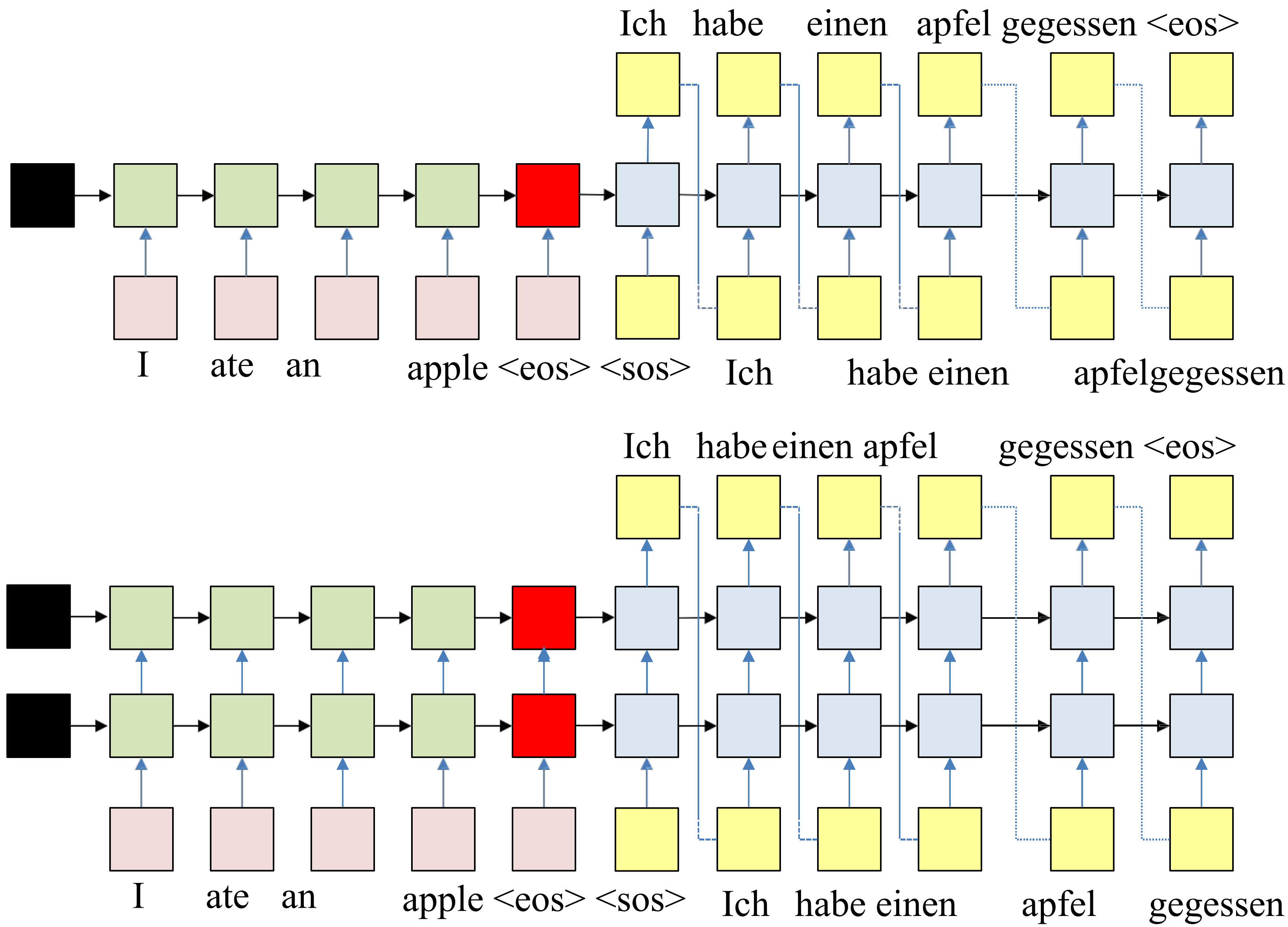
Then use it to generate an output

- *Delayed* sequence to sequence

The “simple” translation model

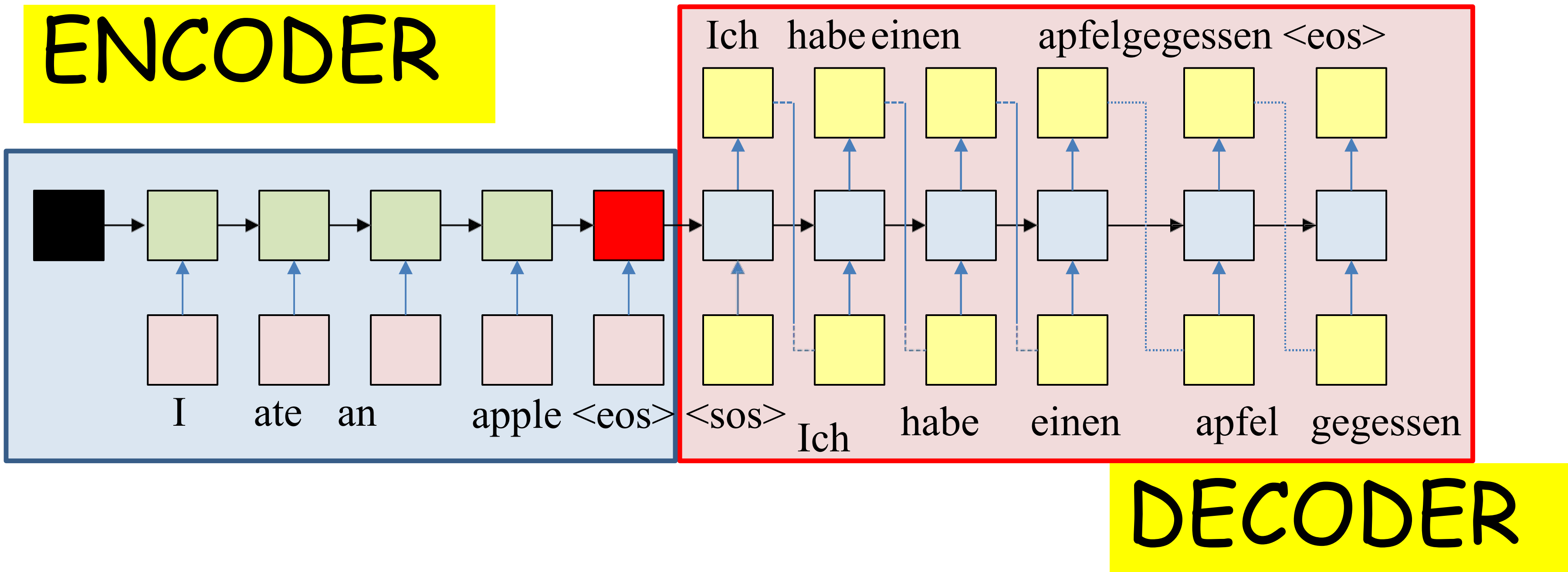


- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  – The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  – The output at each time becomes the input at the next time
  – Output production continues until an <eos> is produced



- We will illustrate with a single hidden layer, but the discussion generalizes to more layers

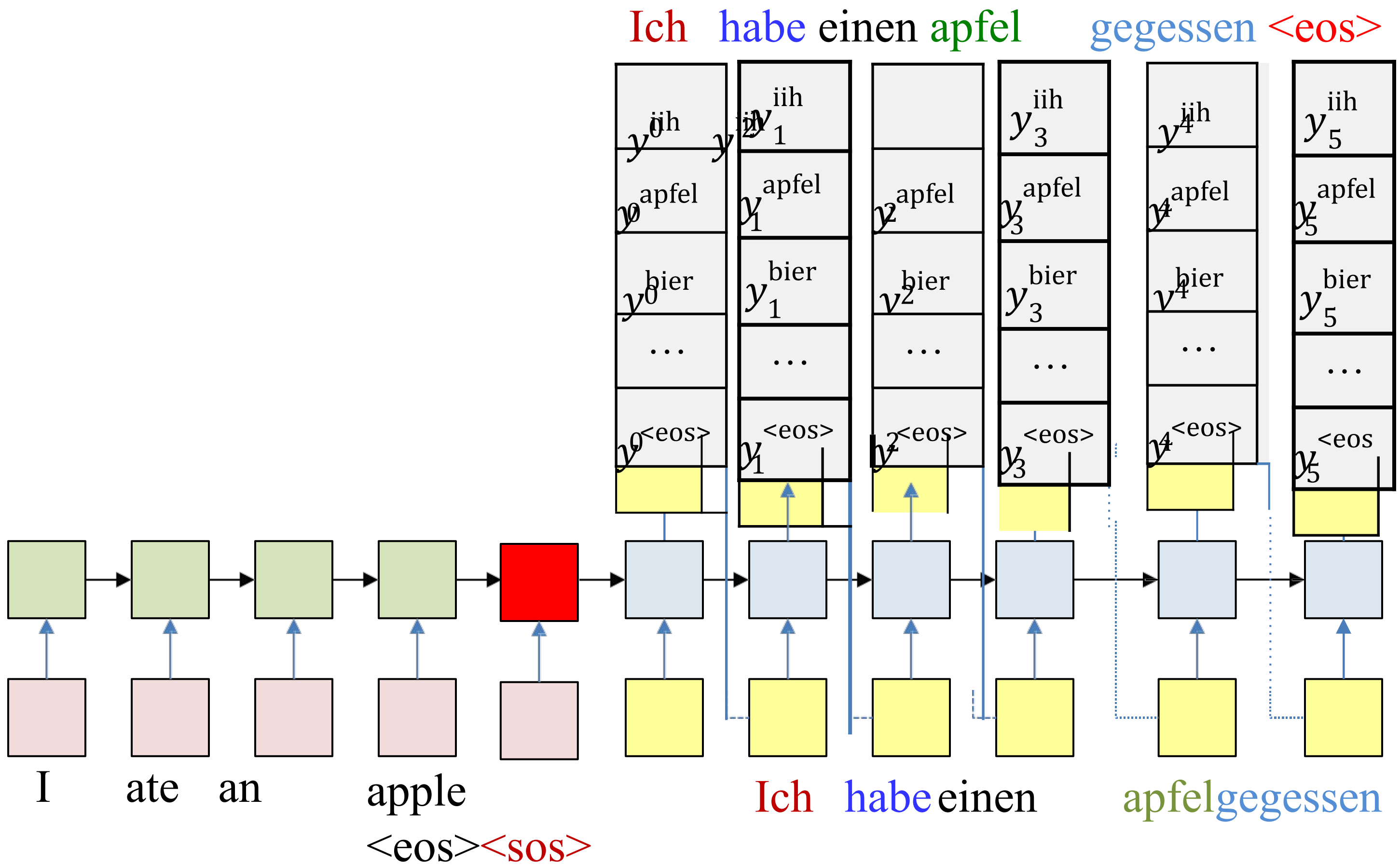
The “simple” translation model



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

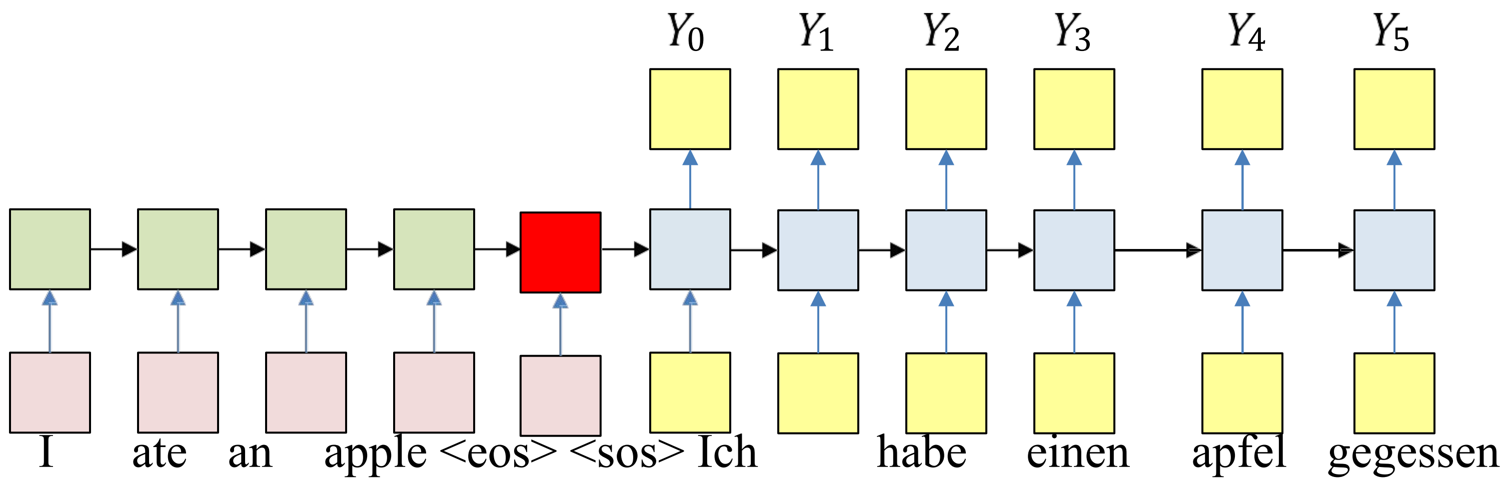
- The recurrent structure that utilizes this representation to produce the output sequence is the *decoder*

Generating an output from the net



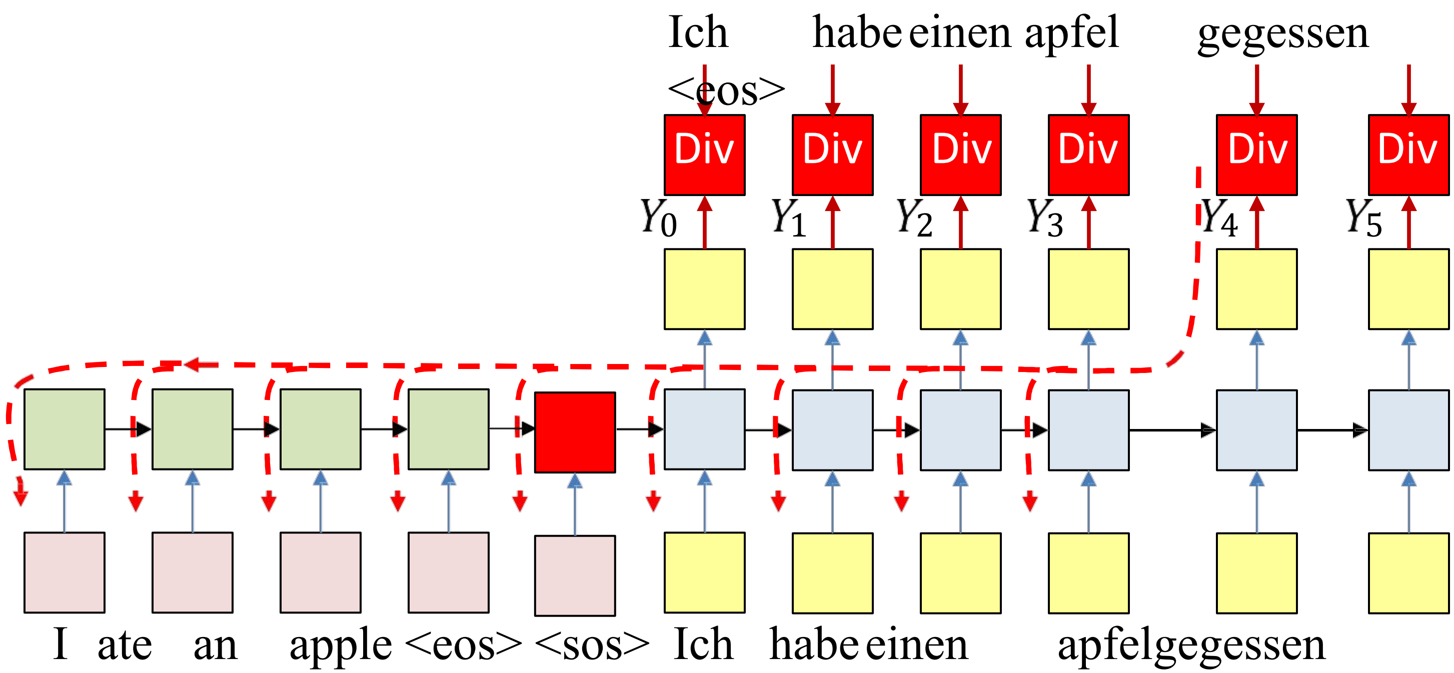
- At each time the network produces a probability distribution over words, given the entire input and entire output sequence so far
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time
- The process continues until an <eos> is generated

Training : Forward pass



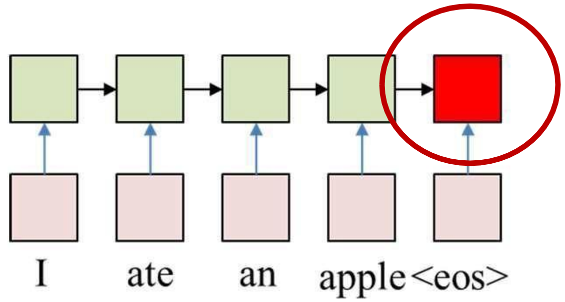
- Forward pass: Input the source and target sequences, sequentially
  - Output will be a probability distribution over target symbol set (vocabulary)

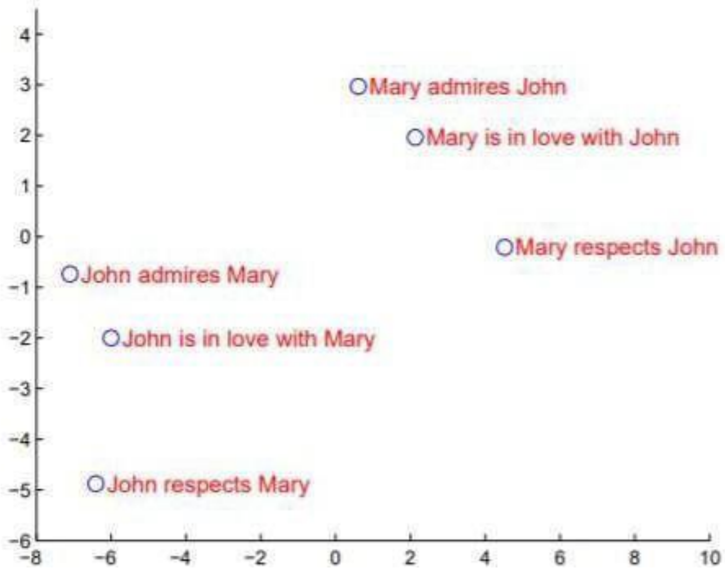
Training : Backward pass

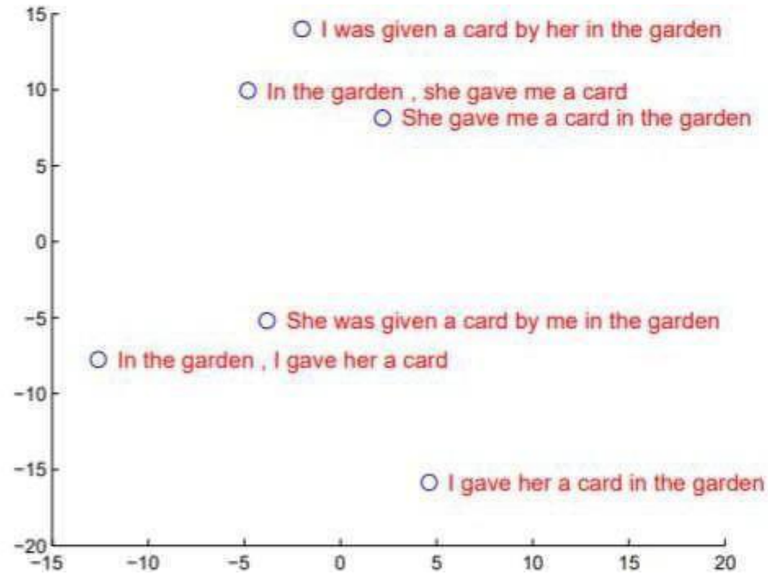


- In practice, if we apply SGD, we may randomly sample words from the output to actually use for the backprop and update
  - Typical usage: Randomly select one word from each input training instance (comprising an input-output pair)
    - For each iteration
      - Randomly select training instance: (input, output)
      - Forward pass
      - Randomly select a single output y(t) and corresponding desired output d(t) for backprop

Machine Translation Example

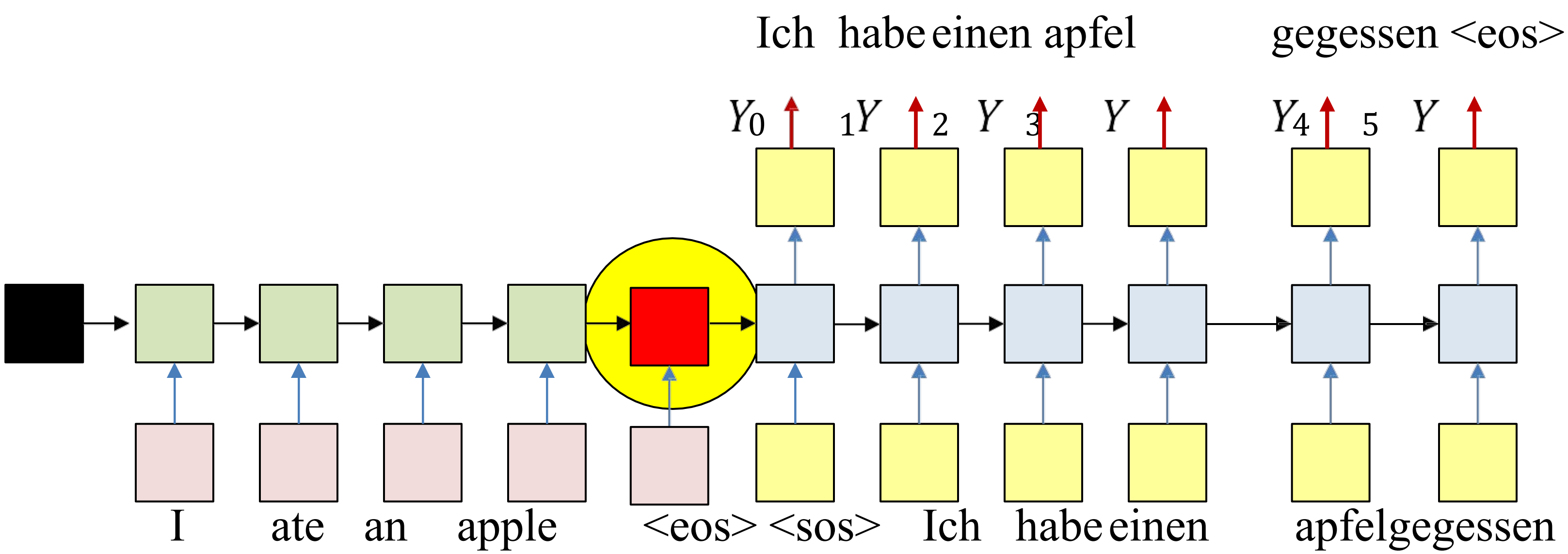






- Hidden state clusters by meaning!
  – From "Sequence-to-sequence learning with neural networks", Sutskever, Vinyals and Le

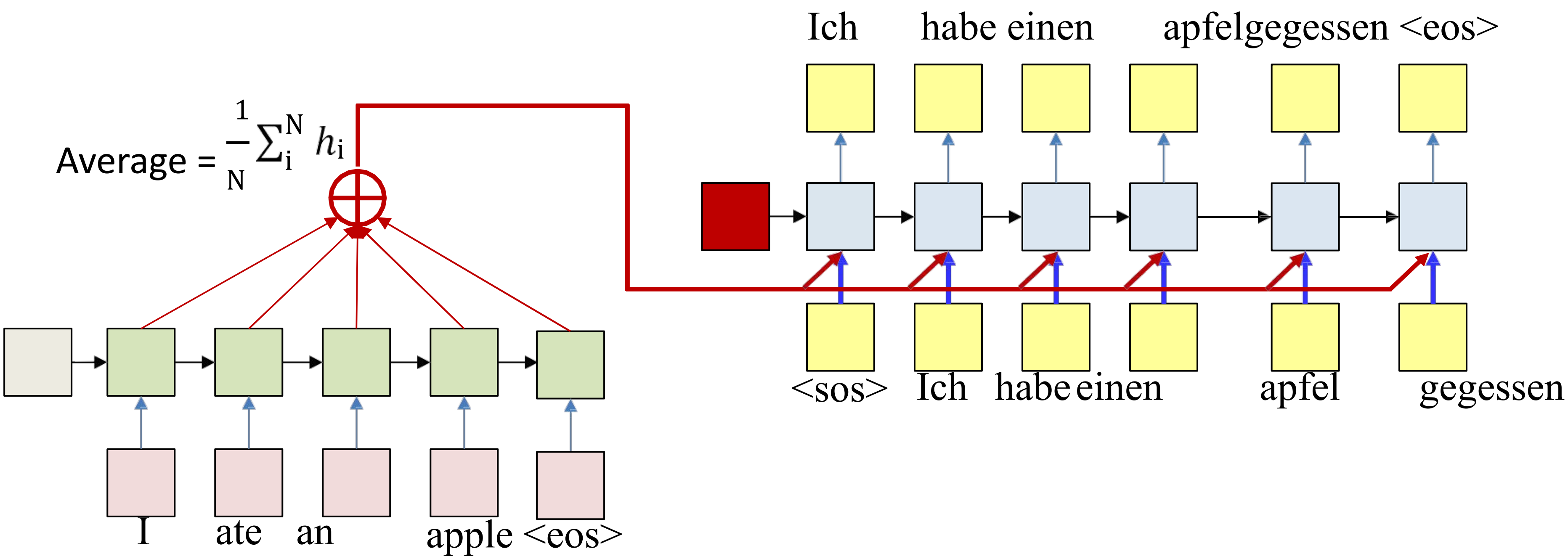
A problem with this framework



- *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

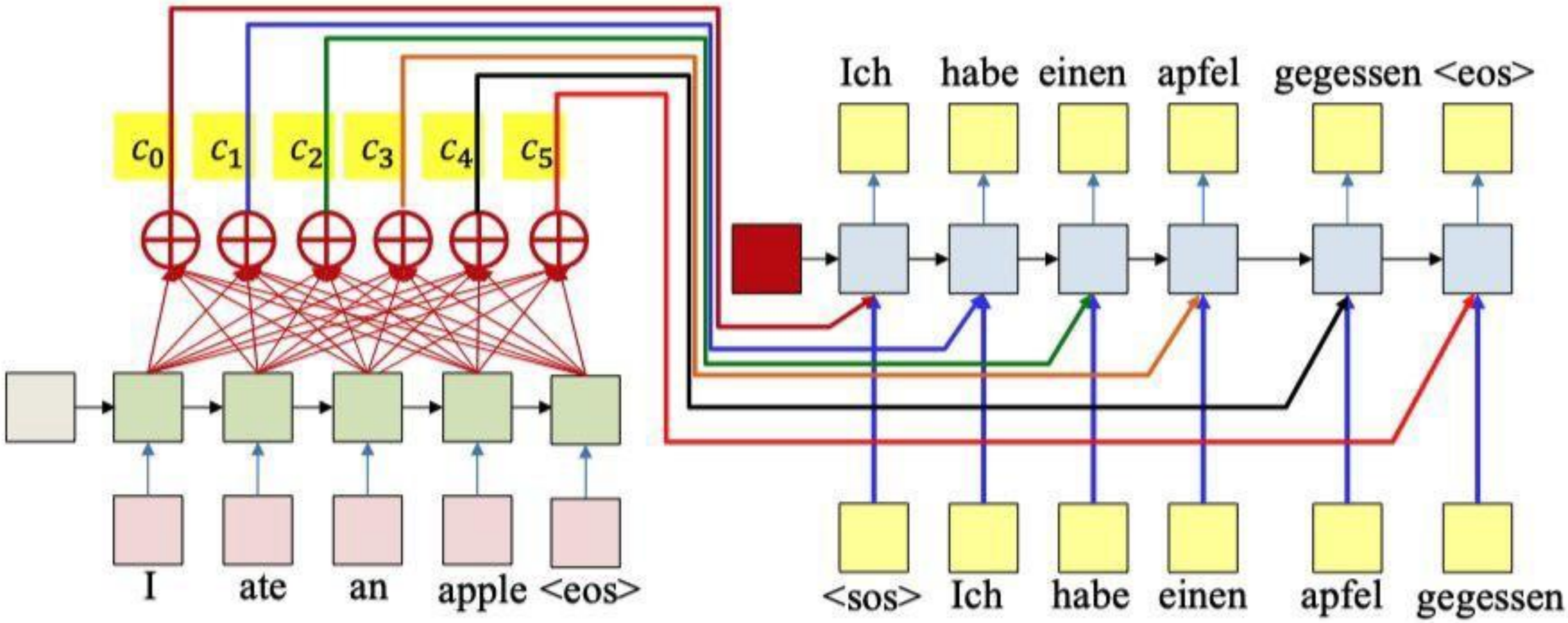
\+ Source and target words can be far apart

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”

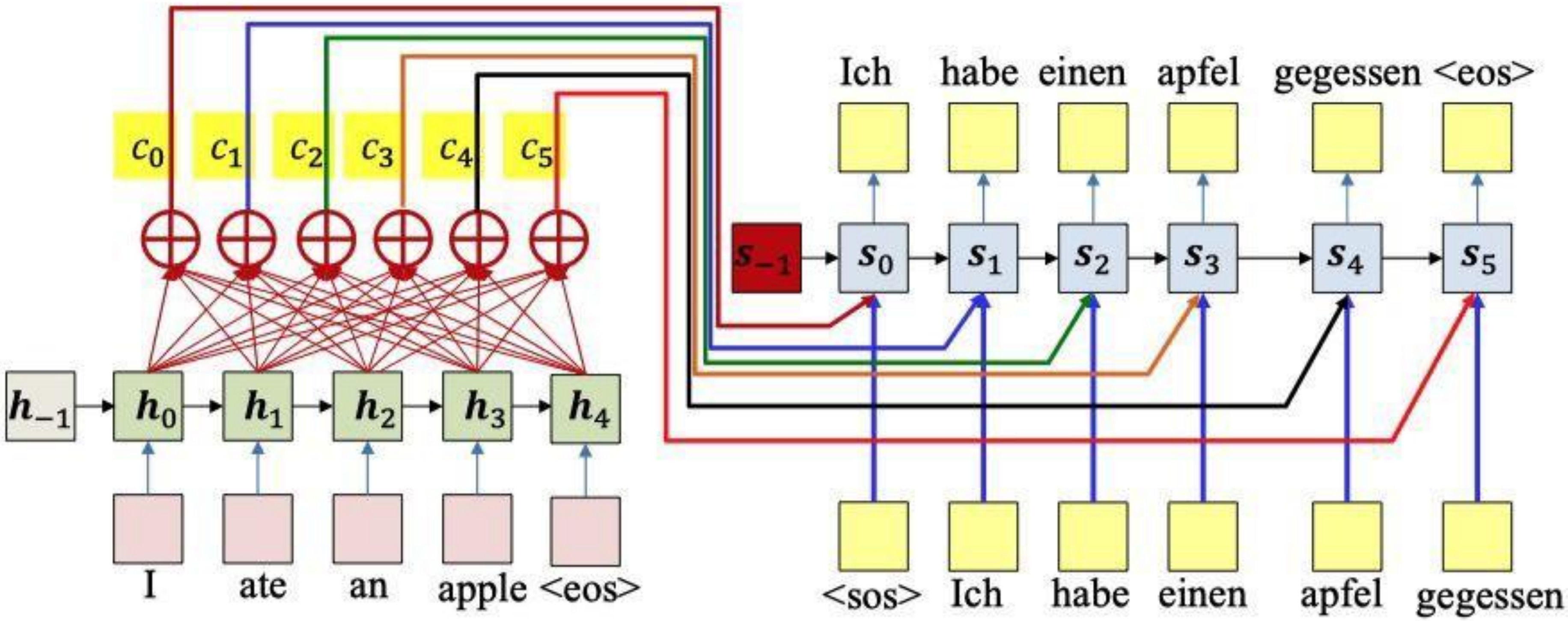
Using all input hidden states



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

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

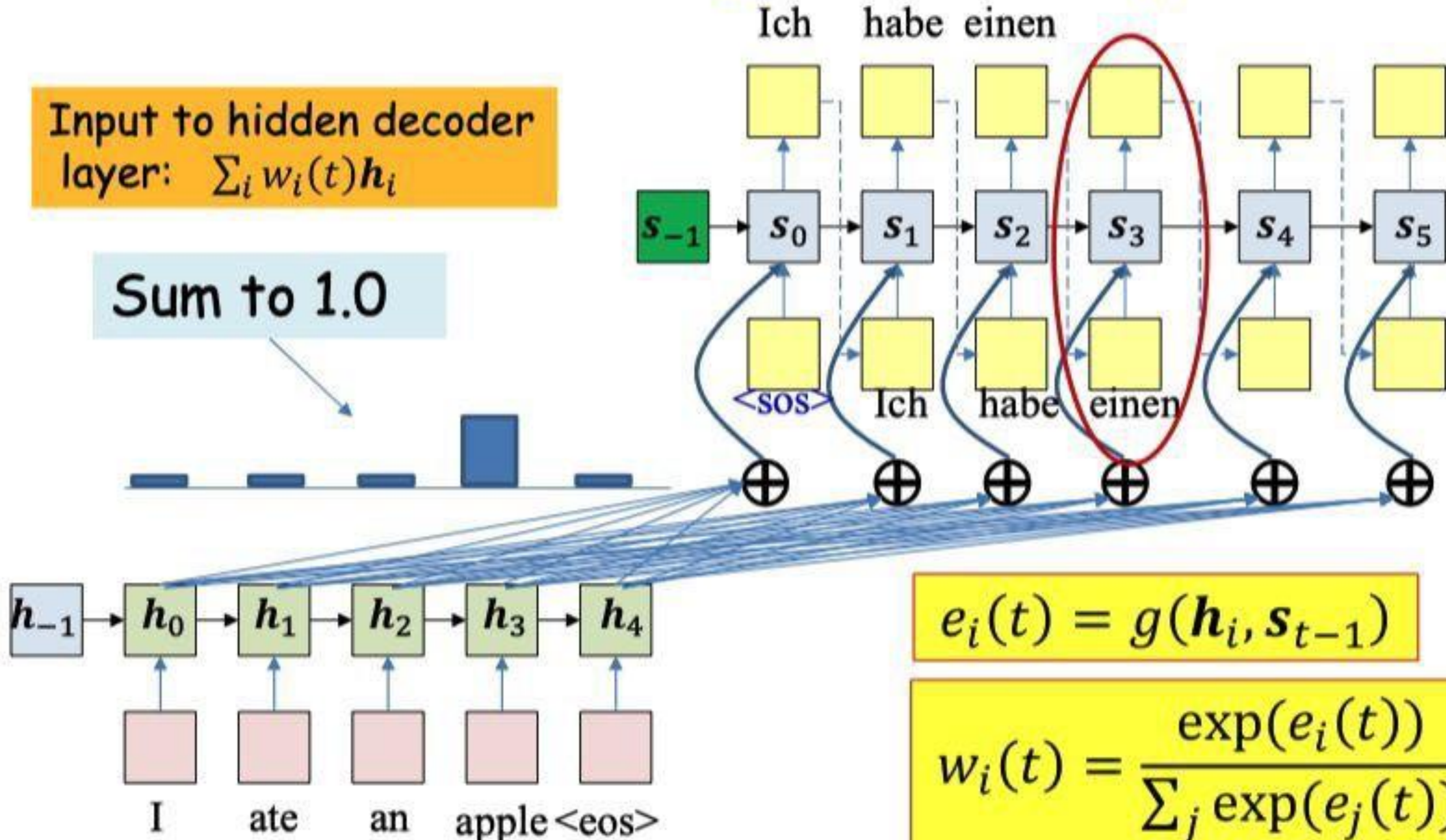
Attention Models



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

- **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?

Summarizing the computation

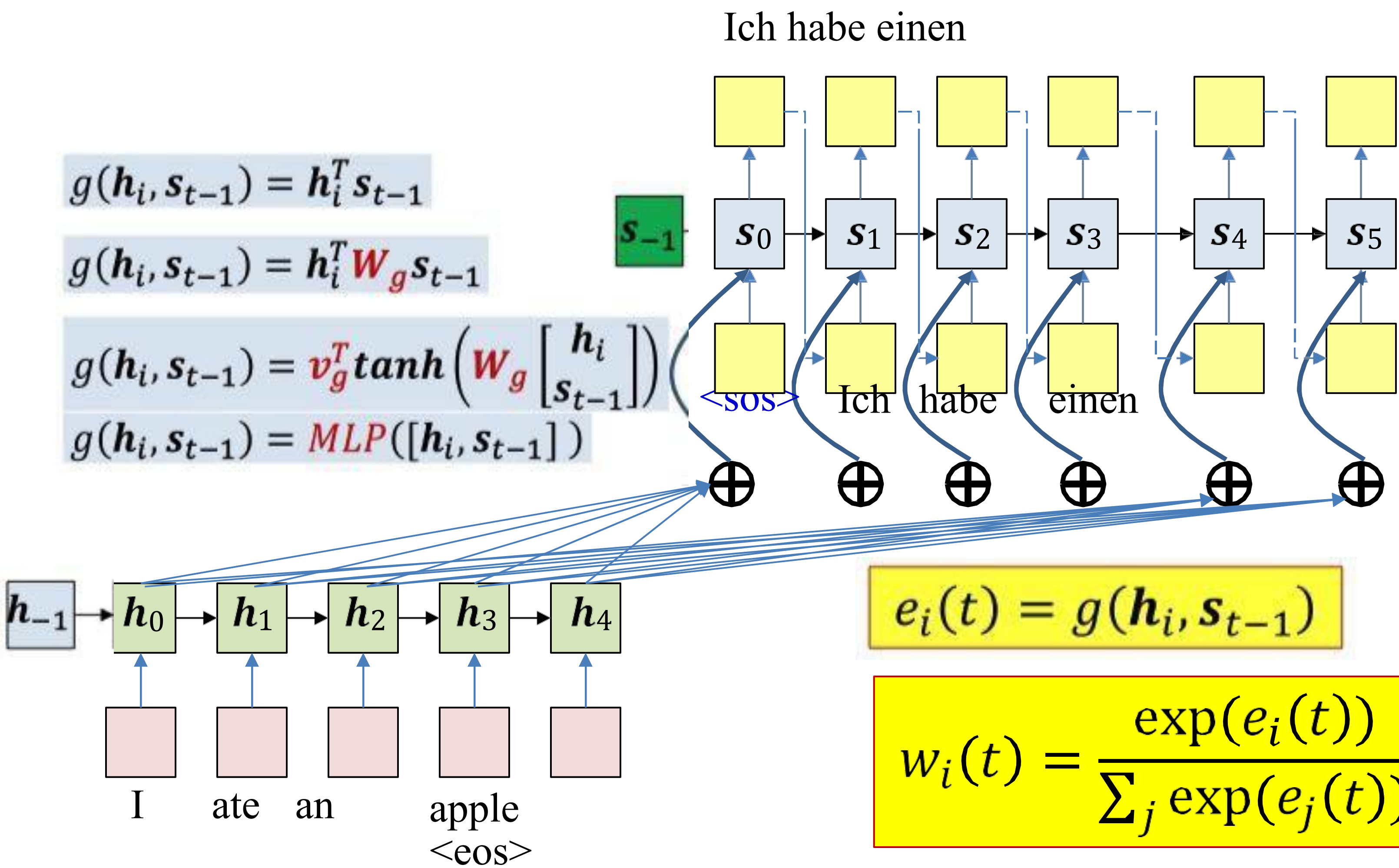


$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

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

- “Raw” weight at any time: A function $g()$ that works on the two hidden states
- Actual weight: softmax over raw weights

Attention models



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

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

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

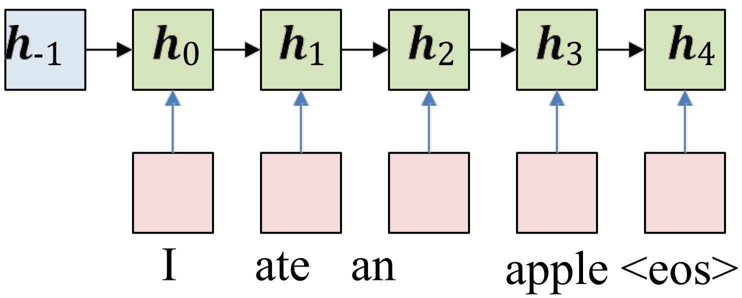
$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$$

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

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

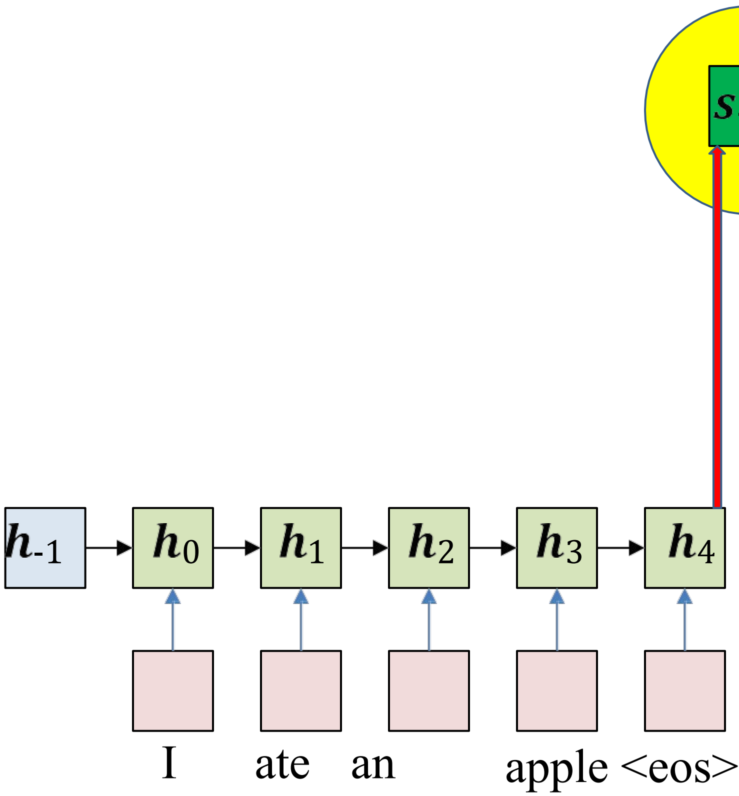
- Typical options for $g()$
  - Variables in red are to be learned

Converting an input (forward pass)



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

Converting an input (forward pass)



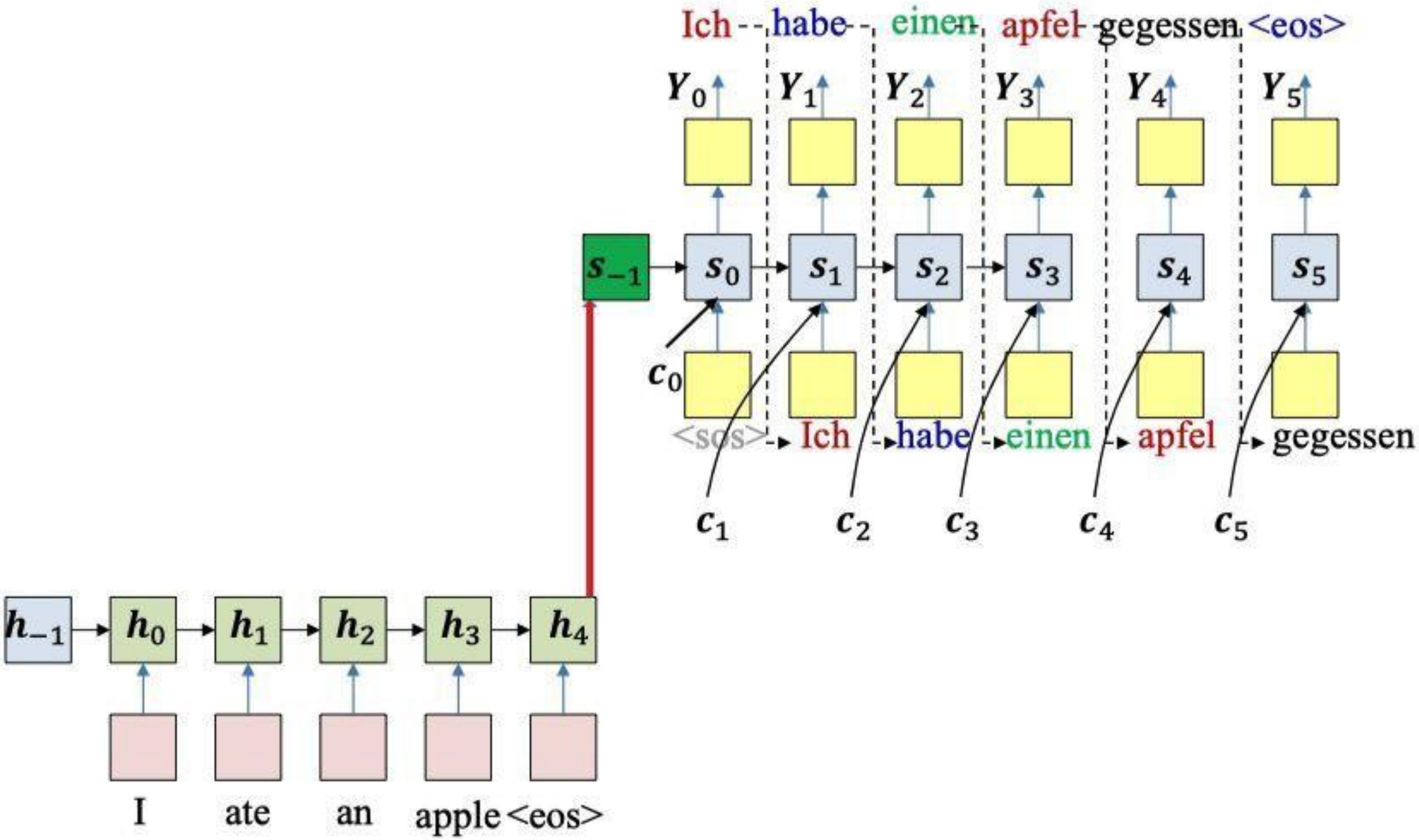
What is this?
Multiple options

Simplest: $s_{-1} = 0$
Alternative: learn s_{-1}
Alternative 2: $s_{-1} = h_N$
If s and h are different sizes:

$$s_{-1} = W_s h_N$$

W_s is learnable parameter

- Initialize decoder hidden state



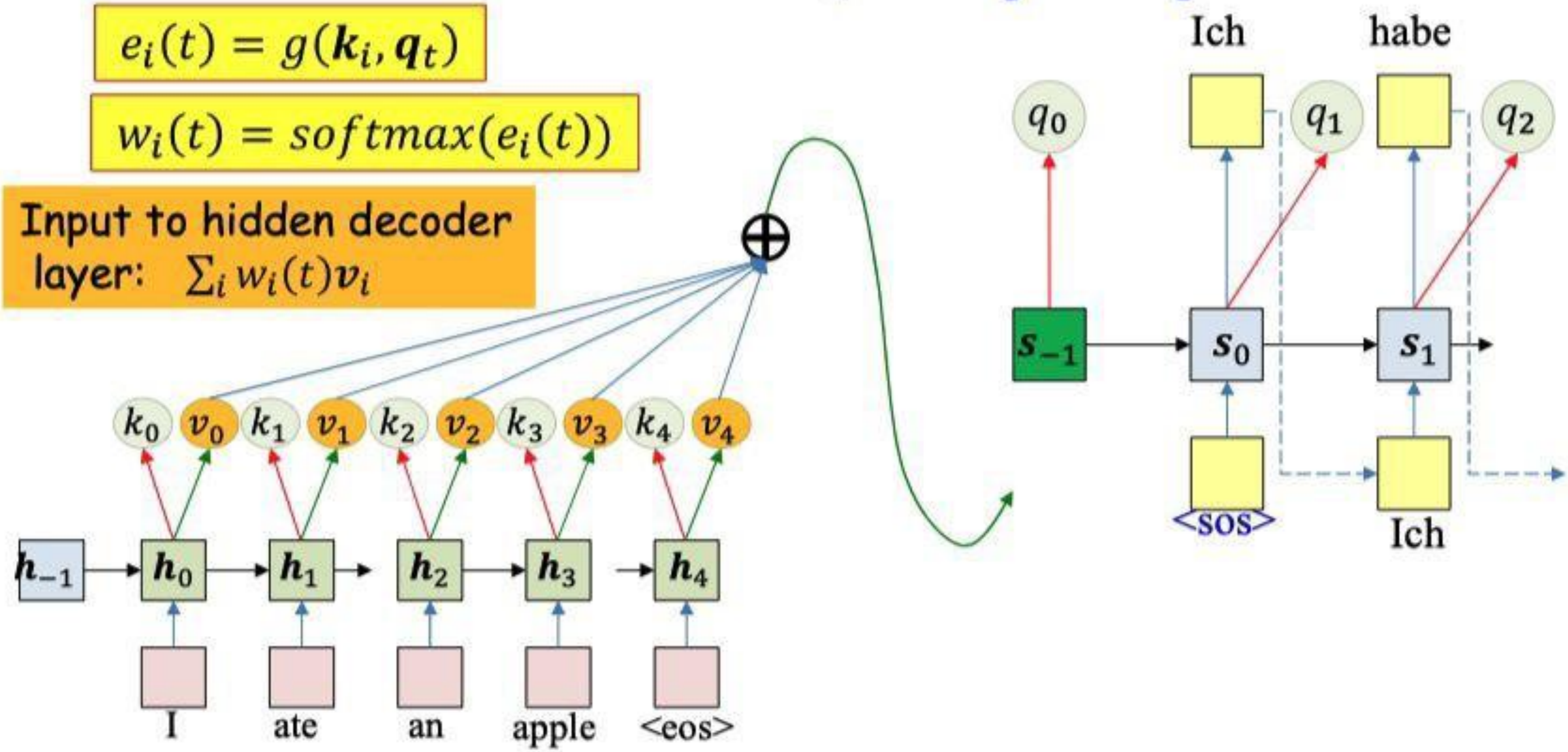
- Continue the process until an end-of-sequence symbol is produced

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: $\sum_i w_i(t)\boldsymbol{v}_i$



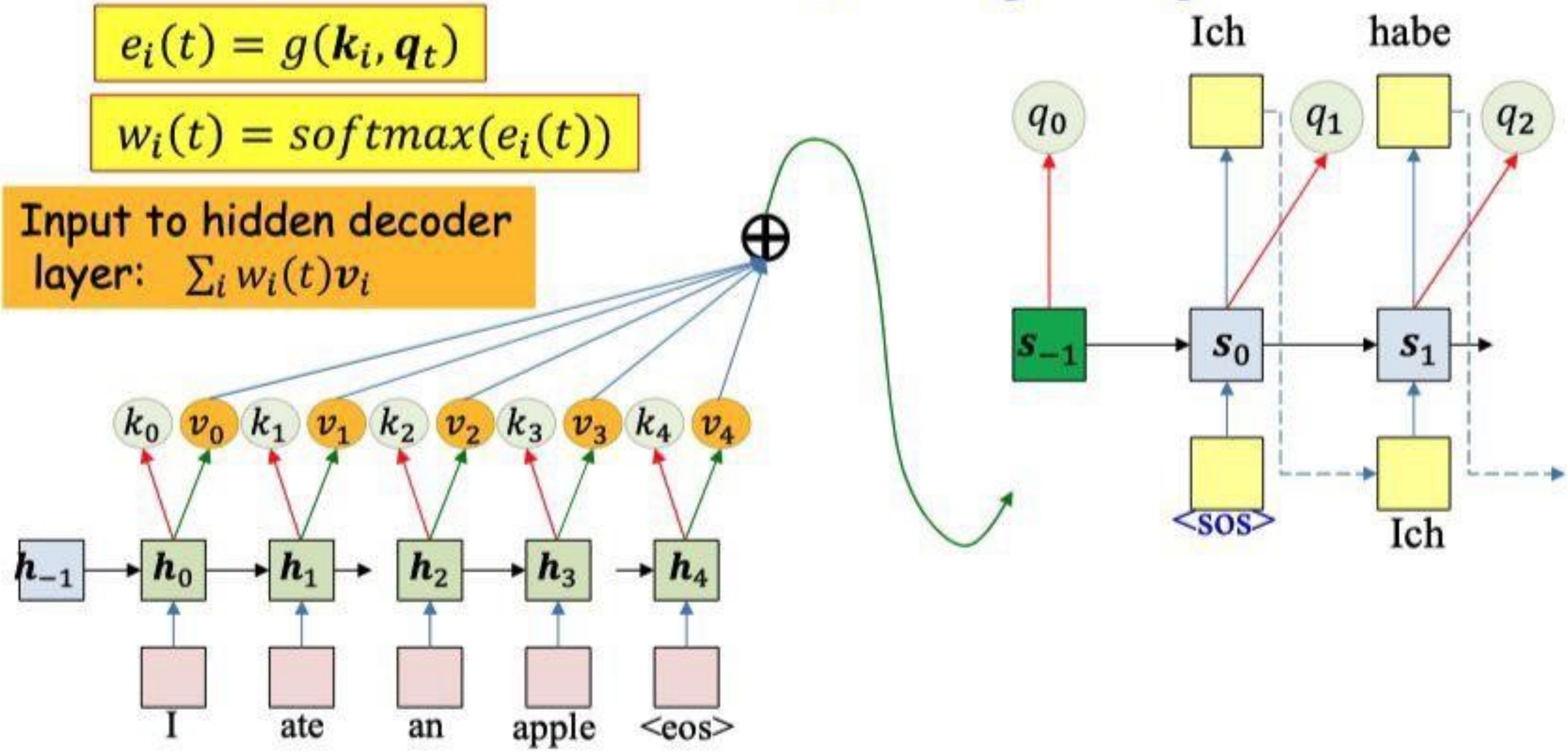
- Encoder outputs an explicit "key" and "value" at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit "query" at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function 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: $\sum_i w_i(t)\boldsymbol{v}_i$

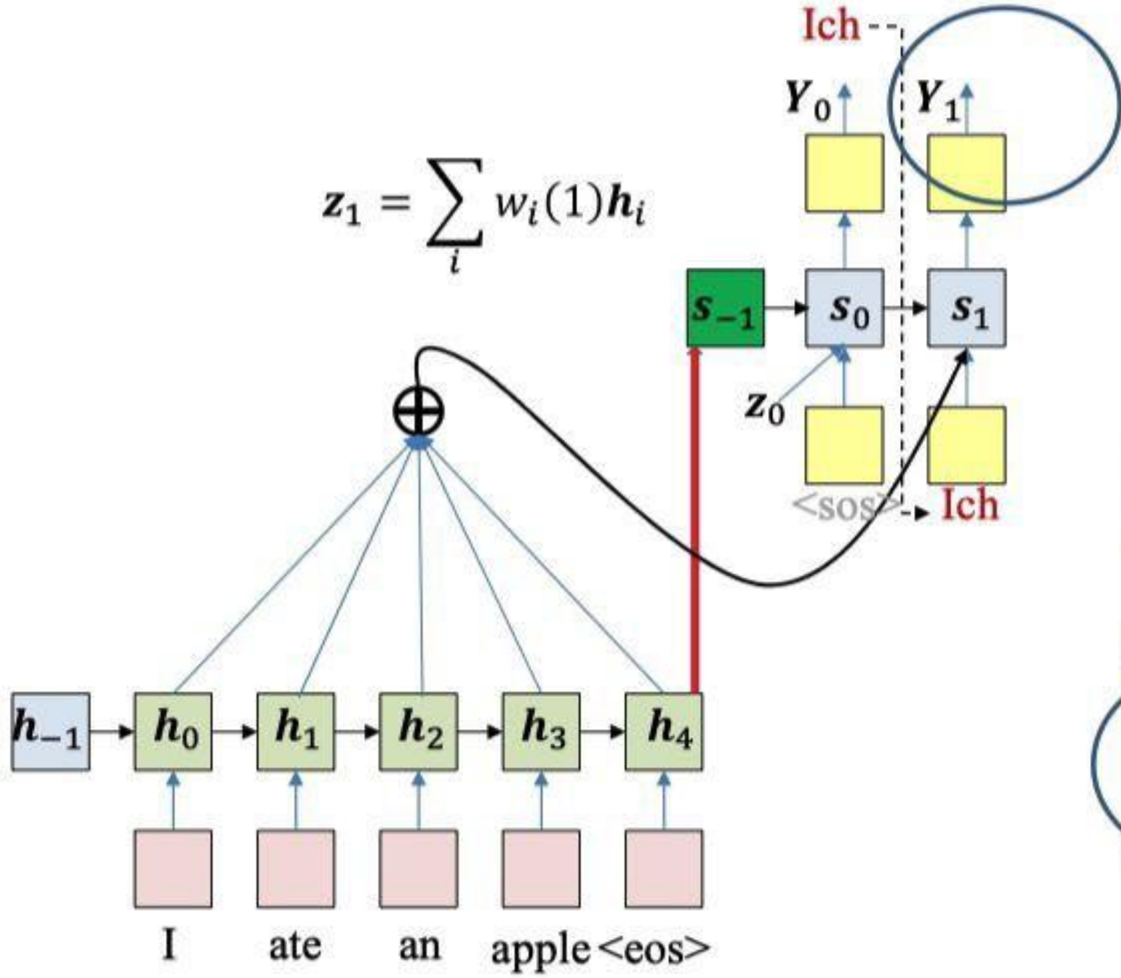


Special case: $k_i = v_i = h_i$

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

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

What does the attention learn?



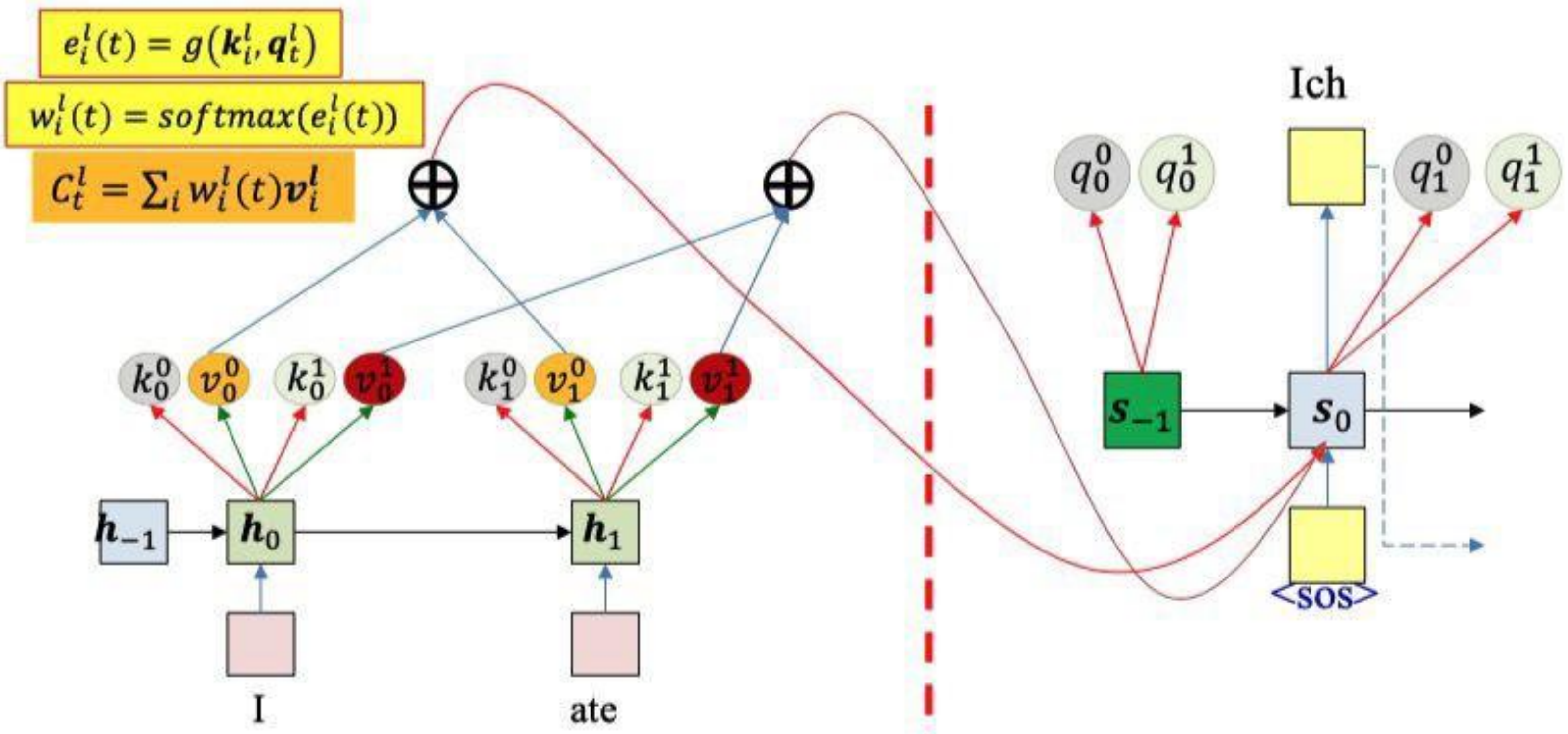
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

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

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

- 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

Extensions: Multihead attention

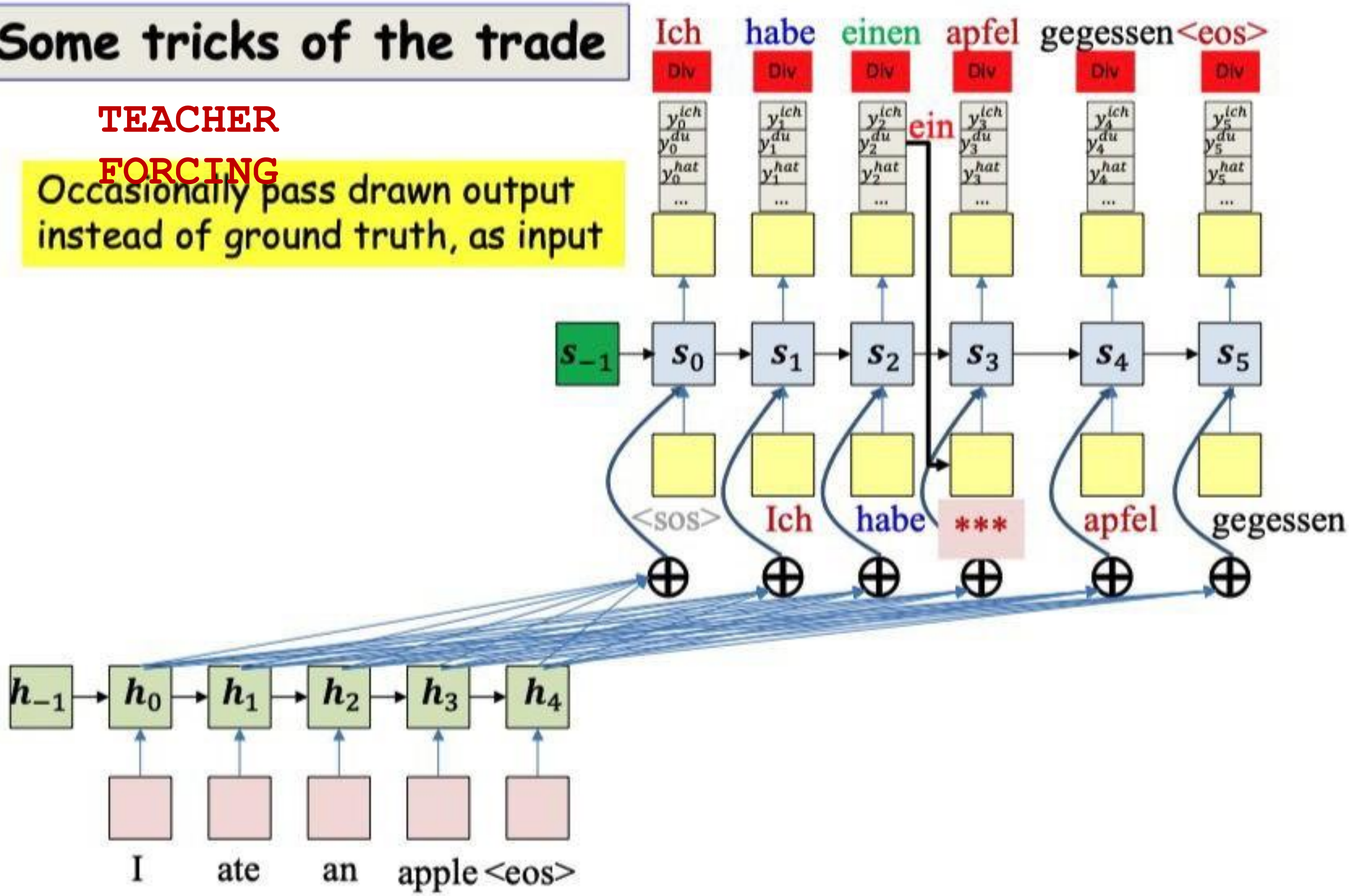


- Have multiple query/key/value sets.
  - Each attention "head" uses one of these sets
  - The combined contexts from all heads are passed to the decoder
- Each "attender" focuses on a different aspect of the input that's important for the decode

Some tricks of the trade

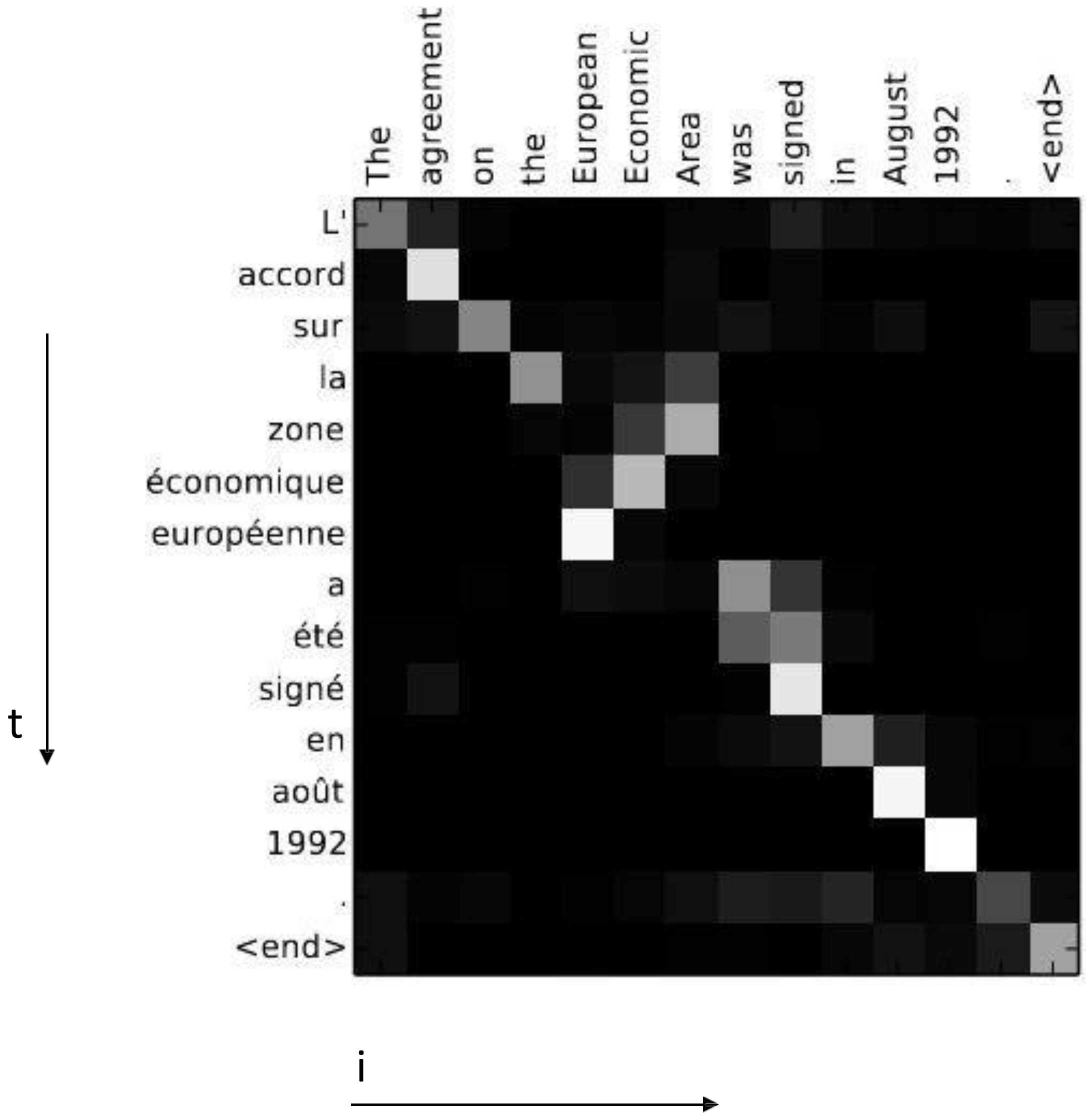
TEACHER FORCING

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

"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

Task	Examples
Sentiment Analysis (Positive)	. . . characters are portrayed with such saddening realism that you can't help but love them , as pathetic as they really are . although levy stands out , guest , willard , o'hara , and posey are all wonderful and definitely should be commended for their performances ! if there was an oscar for an ensemble performance , this is the group that should sweep it . . .
Sentiment Analysis (Negative)	. . . then , as it's been threatening all along , the film explodes into violence . and just when you think it's finally over , schumacher tags on a ridiculous self-righteous finale that drags the whole unpleasant experience down even further . trust me . there are better ways to waste two hours of your life . . .
NLI (Entailment)	**P:** a white dog drinks water on a mountainside. **H:** there is a dog drinking water right now.
NLI (Contradiction)	**P:** a dog leaping off a boat **H:** dogs drinking water from pond

Table 6: Examples of documents (and true label) with feature feedback (highlighted in yellow).

Attention models in image captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddv bear.

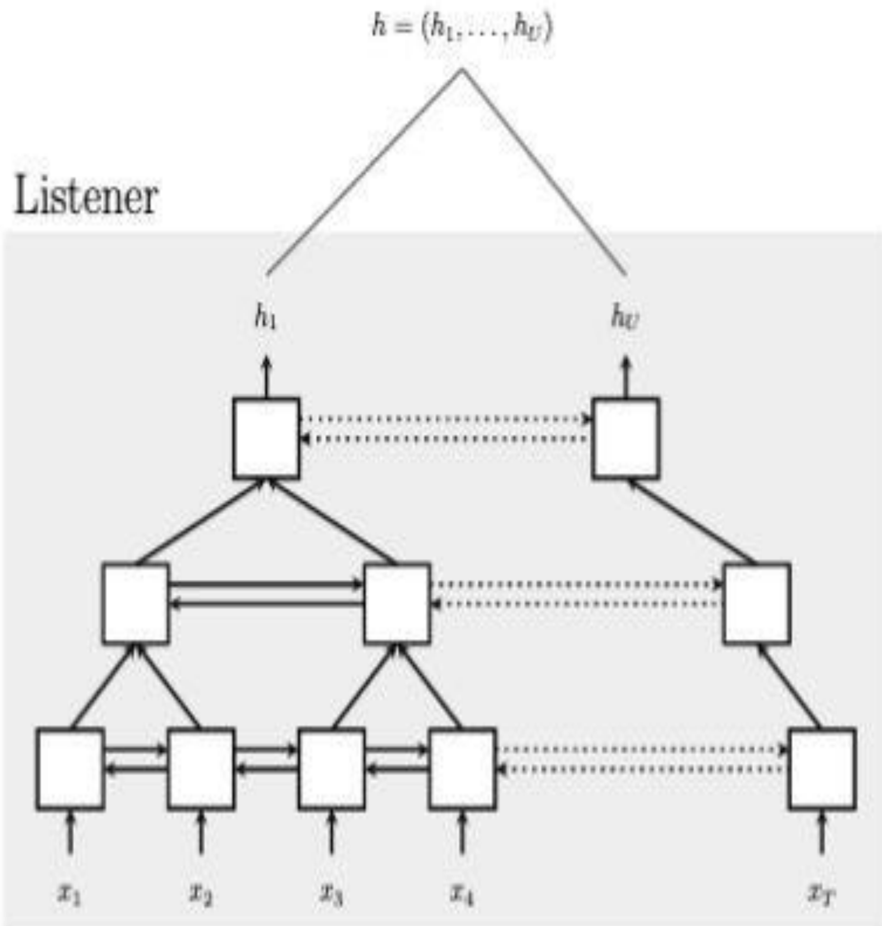
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees 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

LAS: Listener - Pyramidal LSTM

$$\mathbf{h} = (h_1, \ldots, h_U) \text{ with } U \leq T$$



Concerns:

1. Reducing Length
2. Odd/Even Length Input

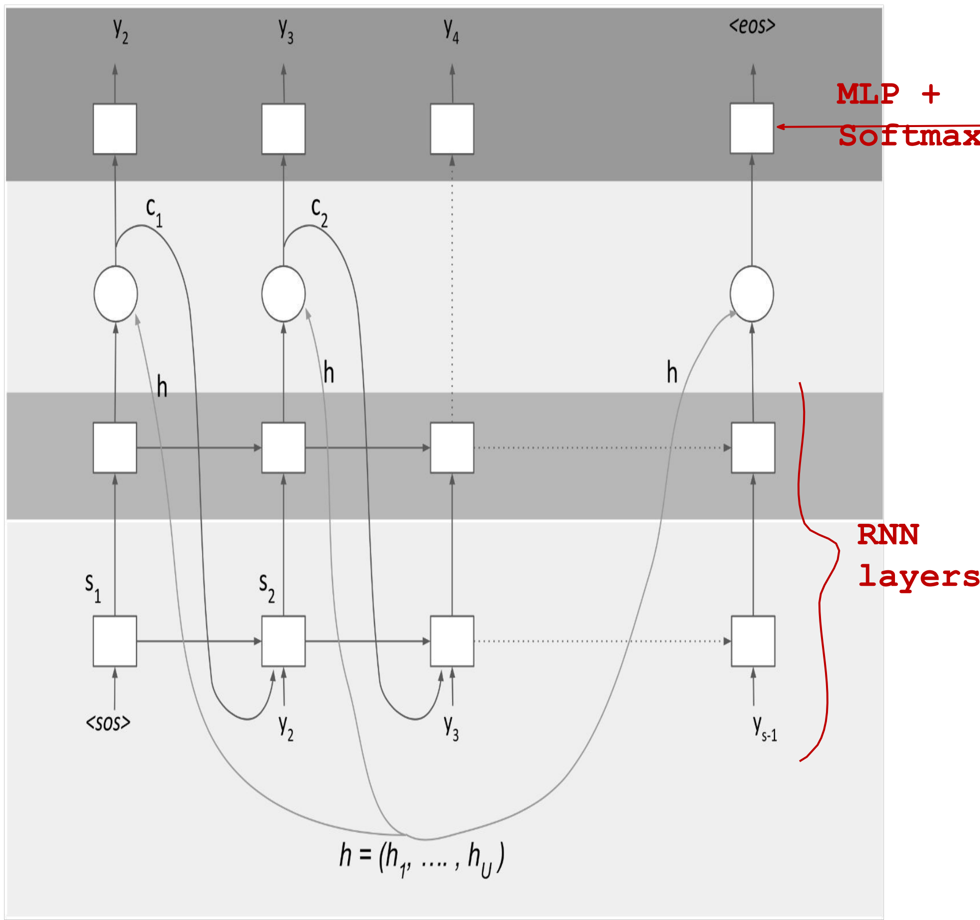
Design:

- 1 bottom BLSTM
- 3 pBLSTMs on top
- Reducing input length by factor of 8

$$\mathbf{h} = \text{Listen}(\mathbf{x})$$
$$P(\mathbf{y}|\mathbf{x}) = \text{AttendAndSpell}(\mathbf{h}, \mathbf{y})$$

$$h_i^j = \text{pBLSTM}(h_{i-1}^j, \left[h_{2i}^{j-1}, h_{2i+1}^{j-1}\right])$$

LAS: Attend and Spell - Decoder



$$c_i = \text{AttentionContext}(s_i, \mathbf{h}) \tag{6}$$

$$s_i = \text{RNN}(s_{i-1}, y_{i-1}, c_{i-1}) \tag{7}$$

$$P(y_i|\mathbf{x}, y_{<i}) = \text{CharacterDistribution}(s_i, c_i) \tag{8}$$

Design:

- Dot-product attention
- 2 RNN layers
- MLP + Softmax for CharacterDistributio n
- Beam Search for decoding