Deep Learning
Sequence to Sequence models: Attention Models

Sequence-to-sequence modelling

- Problem:
  - A sequence $X_1 \ldots X_N$ goes in
  - A different sequence $Y_1 \ldots Y_M$ comes out
- E.g.
  - Speech recognition: Speech goes in, a word sequence comes out
    - Alternately output may be phoneme or character sequence
  - Machine translation: Word sequence goes in, word sequence comes out
  - Dialog : User statement goes in, system response comes out
  - Question answering : Question comes in, answer goes out
- In general $N \neq M$
  - No synchrony between X and Y.

Sequence to sequence

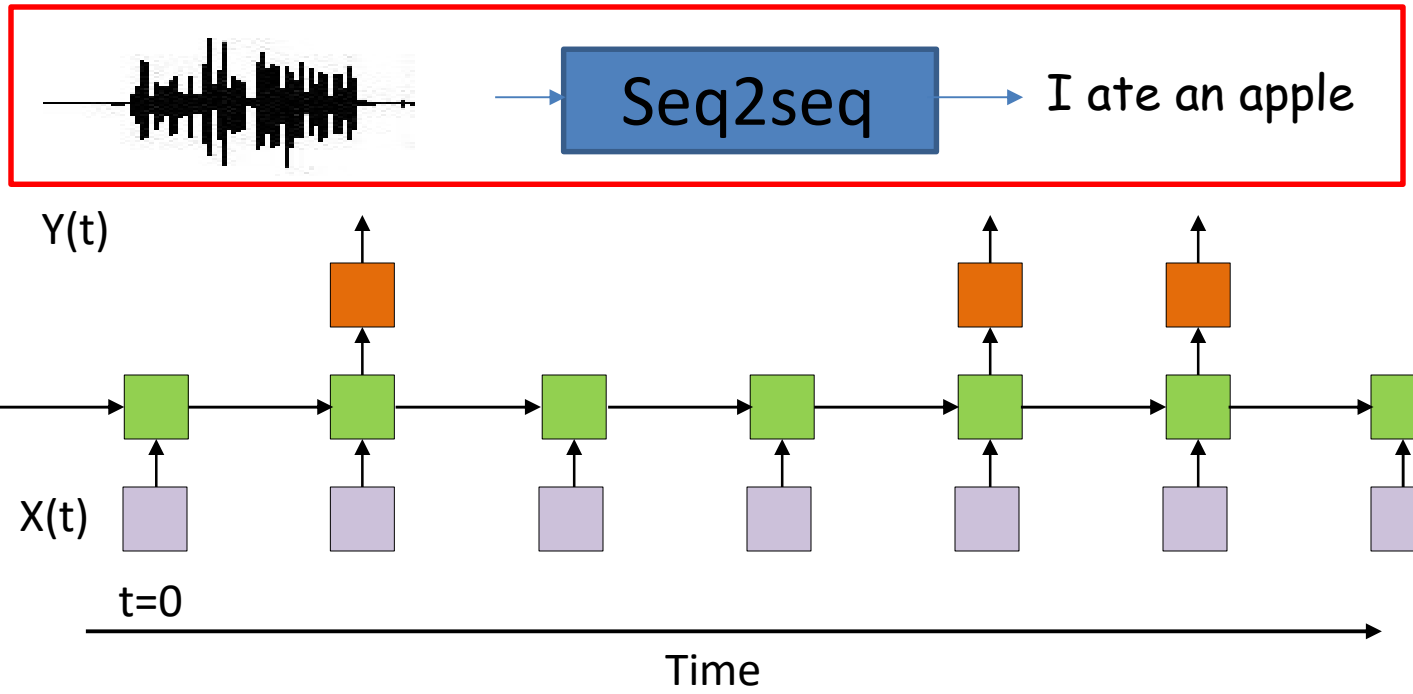
Seq2seq → I ate an apple

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."

Recap: Have dealt with the "aligned" case: CTC



- The input and output sequences happen in the same order
  - Although they may be asynchronous
    - Order-correspondence, but no time synchrony
  - E.g. Speech recognition
    - The input speech corresponds to the phoneme sequence output

Today

[redacted]

- 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."

Recap: Predicting text

- Simple problem: Given a series of symbols (characters or words) w_1 w_2… w_n, predict the next symbol (character or word) w_{n+1}

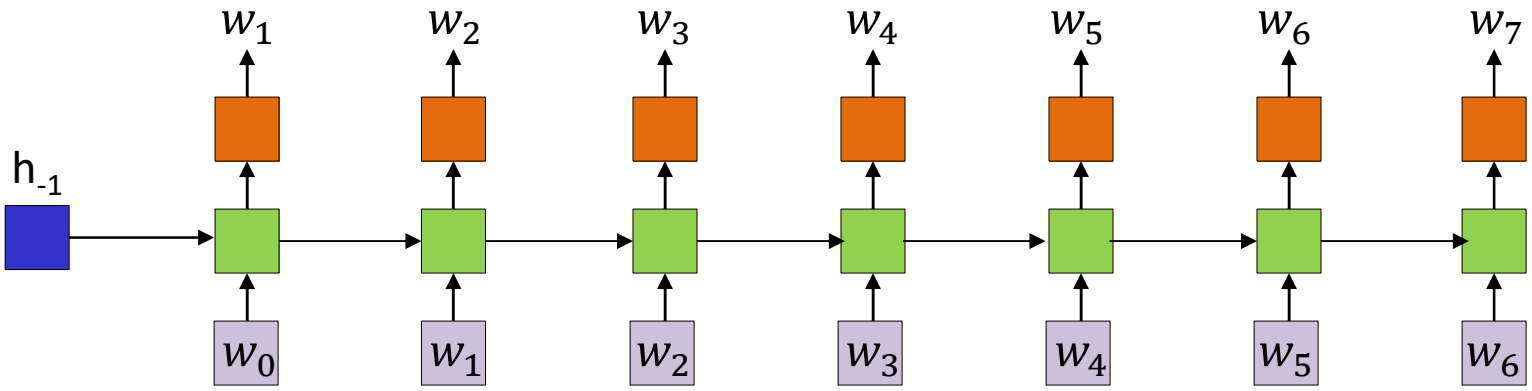
Language modelling using RNNs

Four score and seven years ???

A B R A H A M L I N C O L ??

- Problem: Given a sequence of words (or characters) predict the next one
  - The problem of learning the sequential structure of language

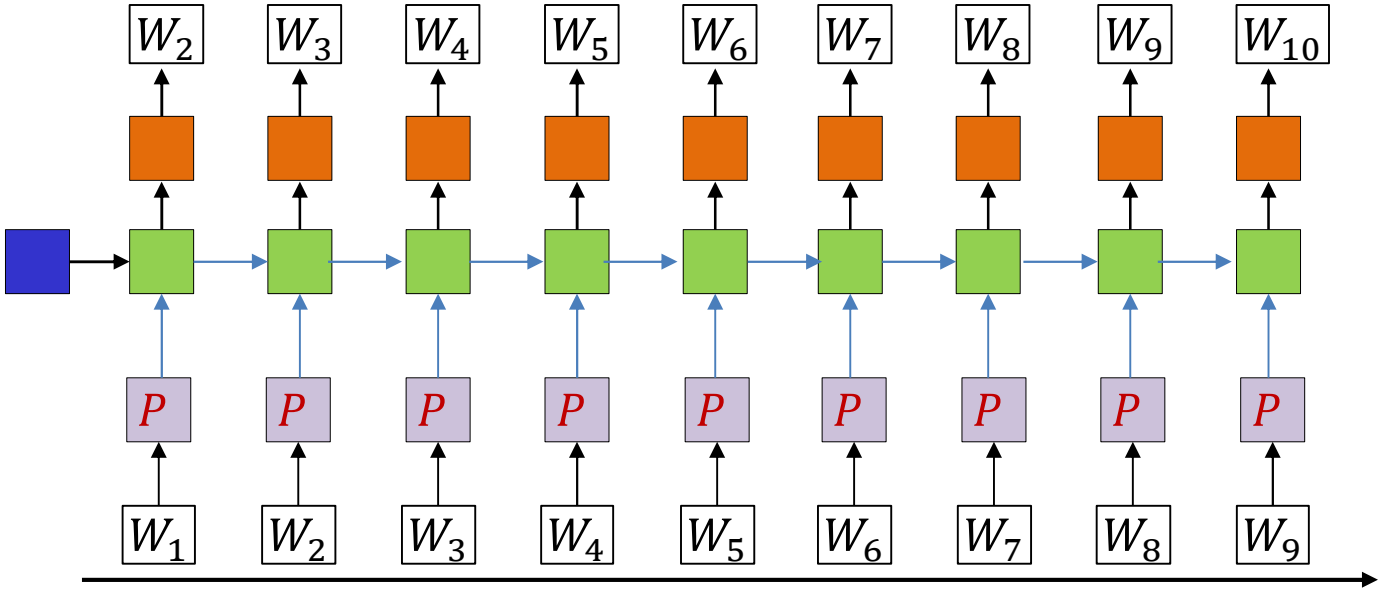
Simple recurrence : Text Modelling



- Learn a model that can predict the next symbol given a sequence of symbols
  - Characters or words

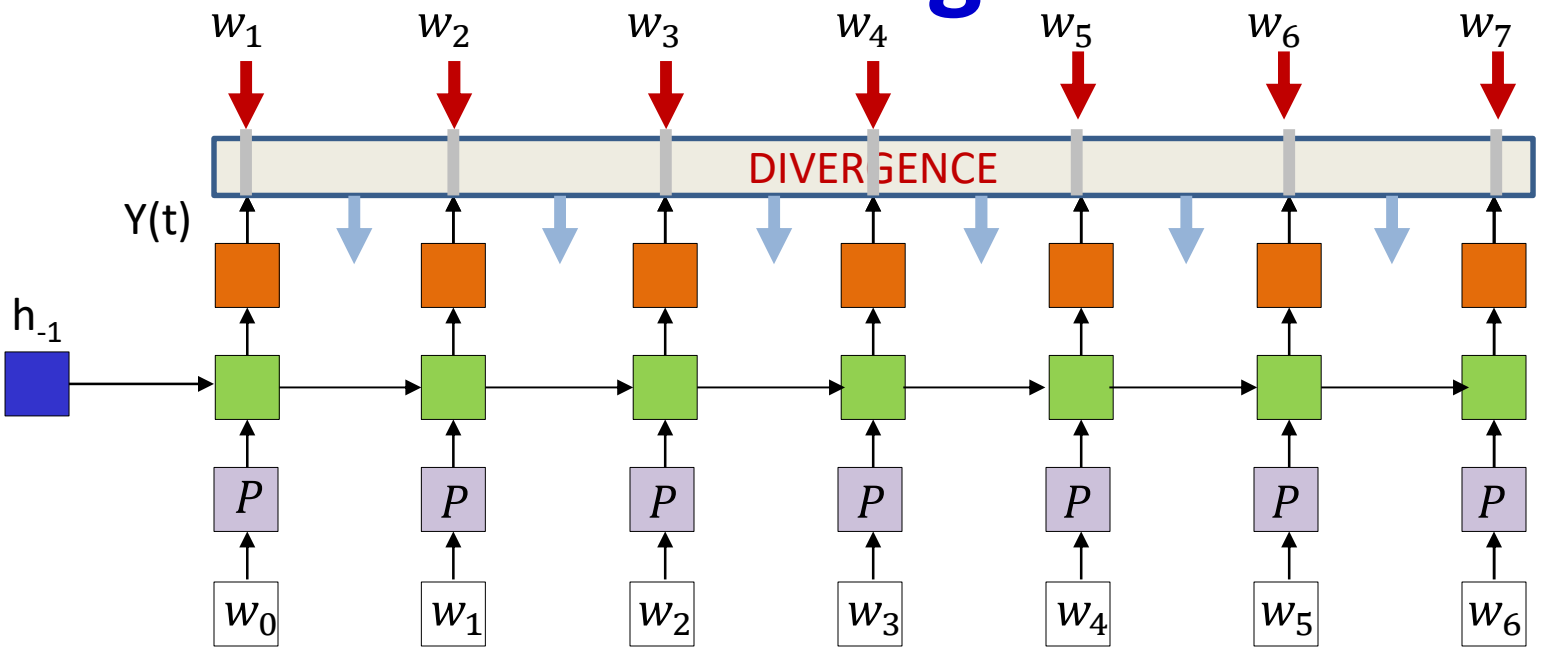
- After observing inputs w_0 ... w_k it predicts w_{k+1}
  - In reality, outputs a probability distribution for w_{k+1}

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

Training



- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the "vocabulary"
  - Projected down to lower-dimensional "embeddings"
- Output: Probability distribution over symbols

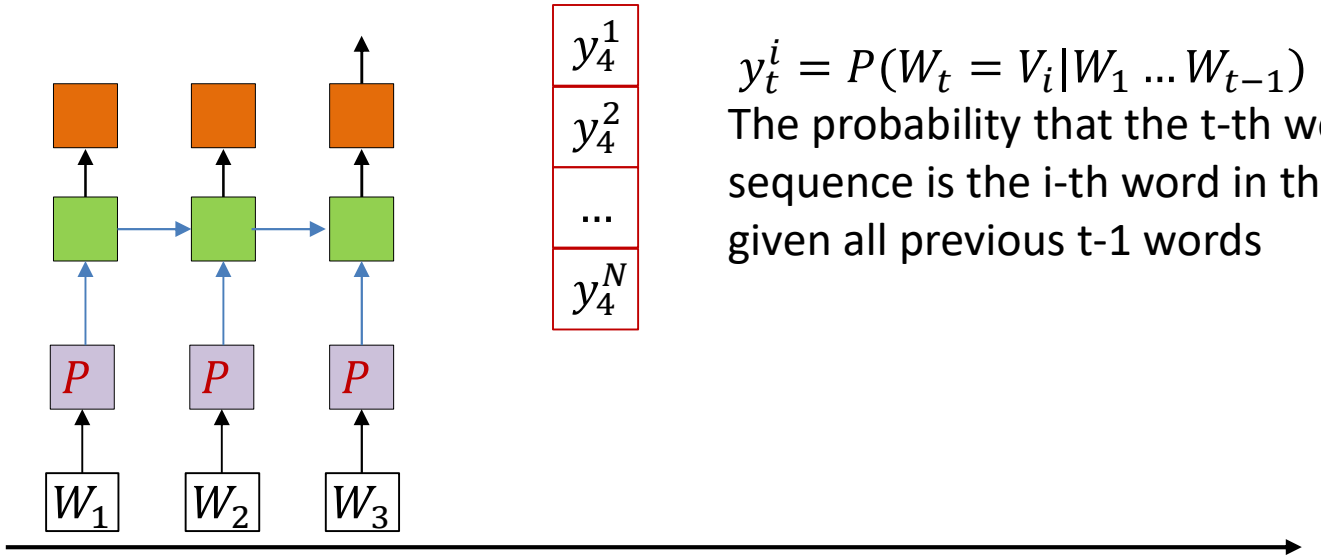
$$Y(t,i) = P(V_i | w_0 \dots w_{t-1})$$

  - V_i is the i-th symbol in the vocabulary
- Divergence

$$Div(w(1 \dots T), \mathbf{Y}(0 \dots T-1)) = \sum_t KL(w(t+1), \mathbf{Y}(t)) = -\sum_t \log Y(t, w_{t+1})$$

The probability assigned to the correct next word

Generating Language: Synthesis

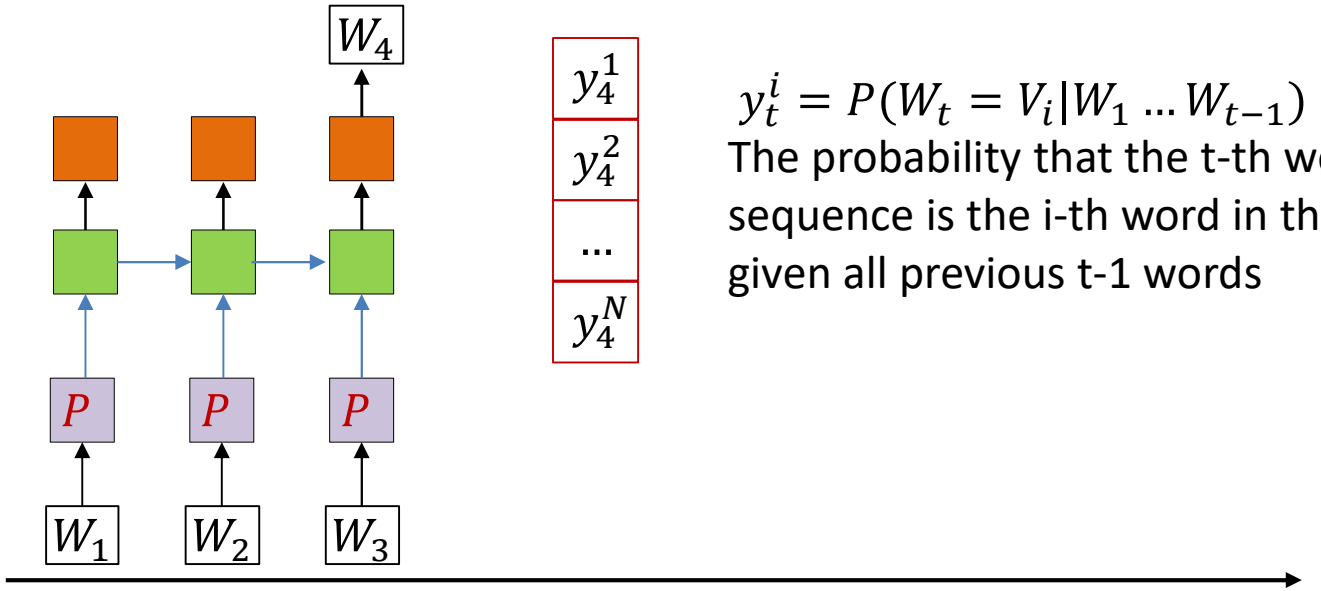


$$y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1})$$

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words

- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector

Generating Language: Synthesis

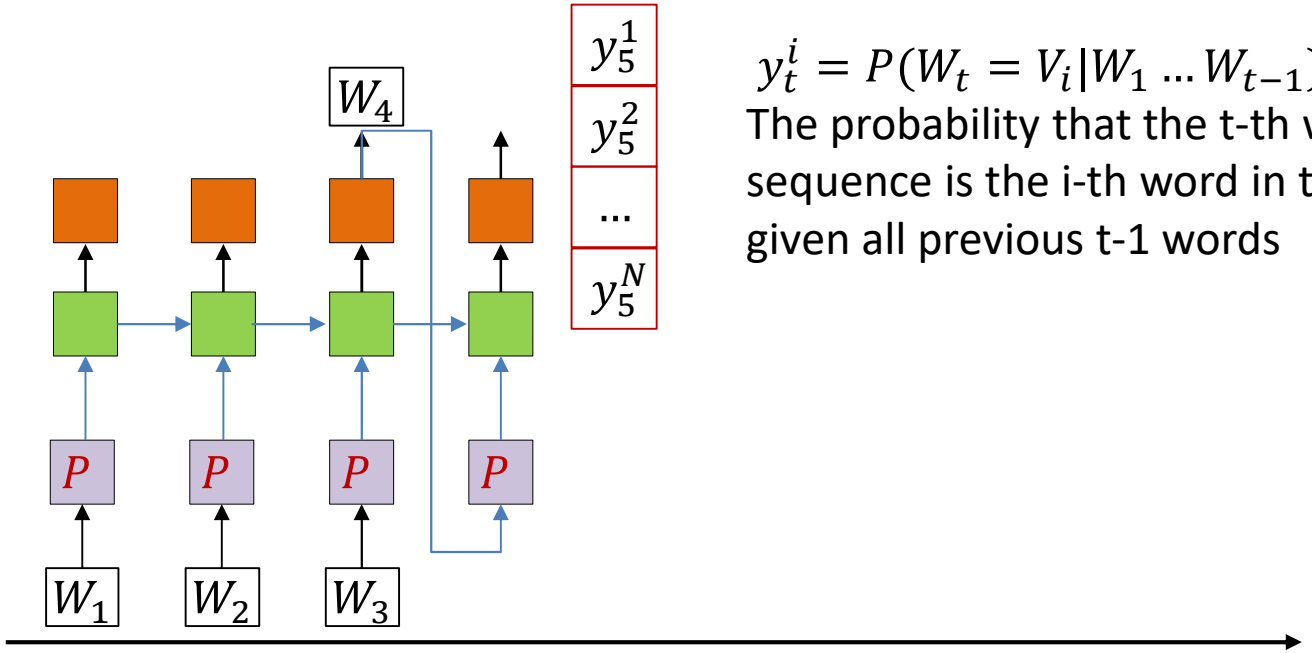


$$y_t^i = P(W_t = V_i | W_1 \dots W_{t-1})$$

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words

- On trained model : Provide the first few words
  - One-hot vectors
- After the last input word, the network generates a probability distribution over words
  - Outputs an N-valued probability distribution rather than a one-hot vector
- Draw a word by sampling from the distribution
  - And set it as the next word in the series

Generating Language: Synthesis

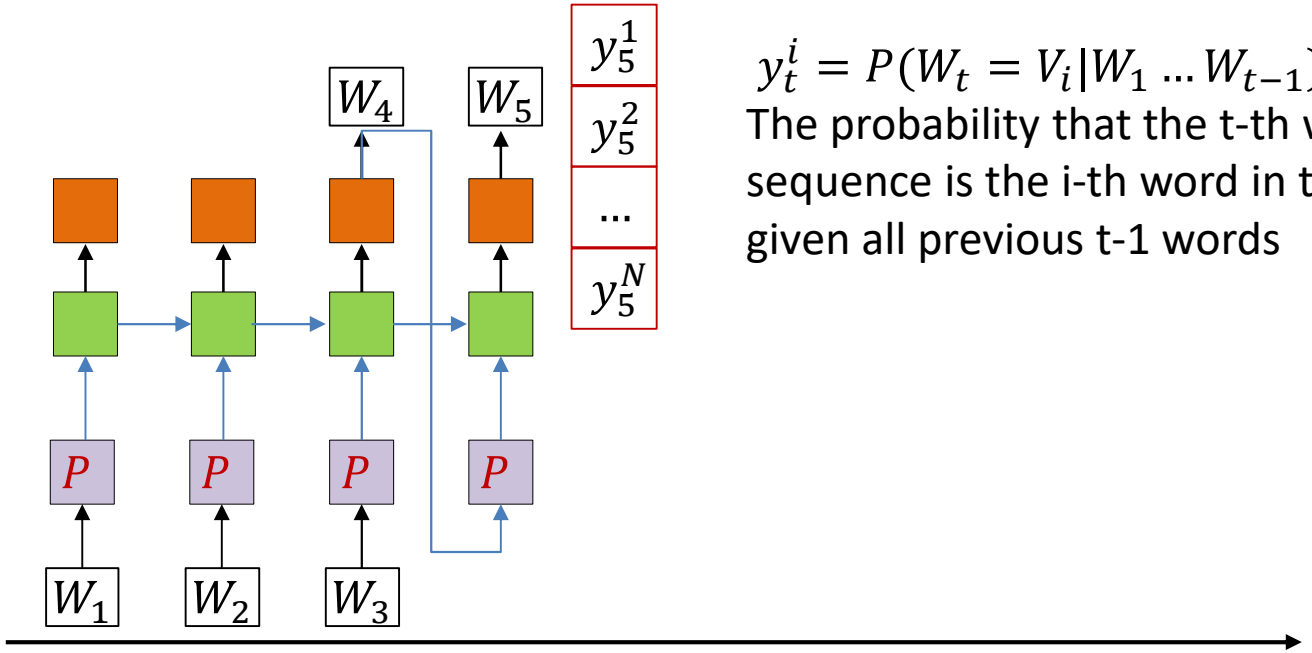


$y_t^i = P(W_t = V_i | W_1 \dots W_{t-1})$

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words

- Feed the drawn word as the next word in the series
  - And draw the next word by sampling from the output probability distribution

Generating Language: Synthesis

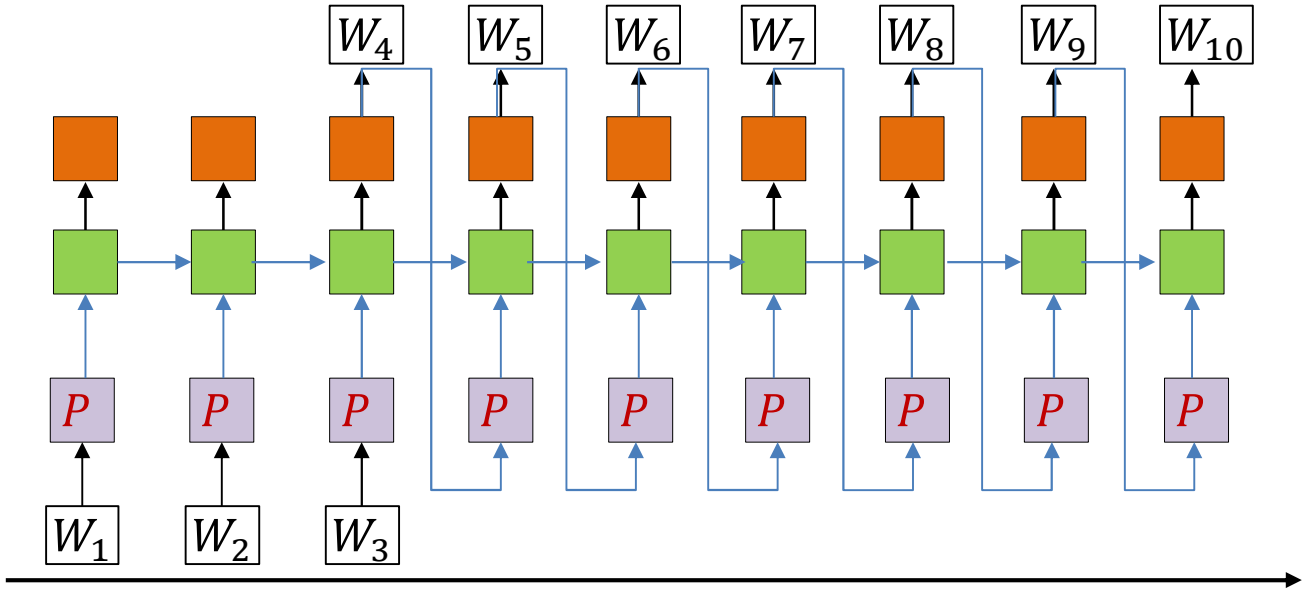


$y_t^i = P(W_t = V_i | W_1 \ldots W_{t-1})$

The probability that the t-th word in the sequence is the i-th word in the vocabulary given all previous t-1 words

- Feed the drawn word as the next word in the series
  - And draw the next word by sampling from the output probability distribution

Generating Language: Synthesis



- Feed the drawn word as the next word in the series
  - And draw the next word by sampling from the output probability distribution
- **When do we stop?**

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

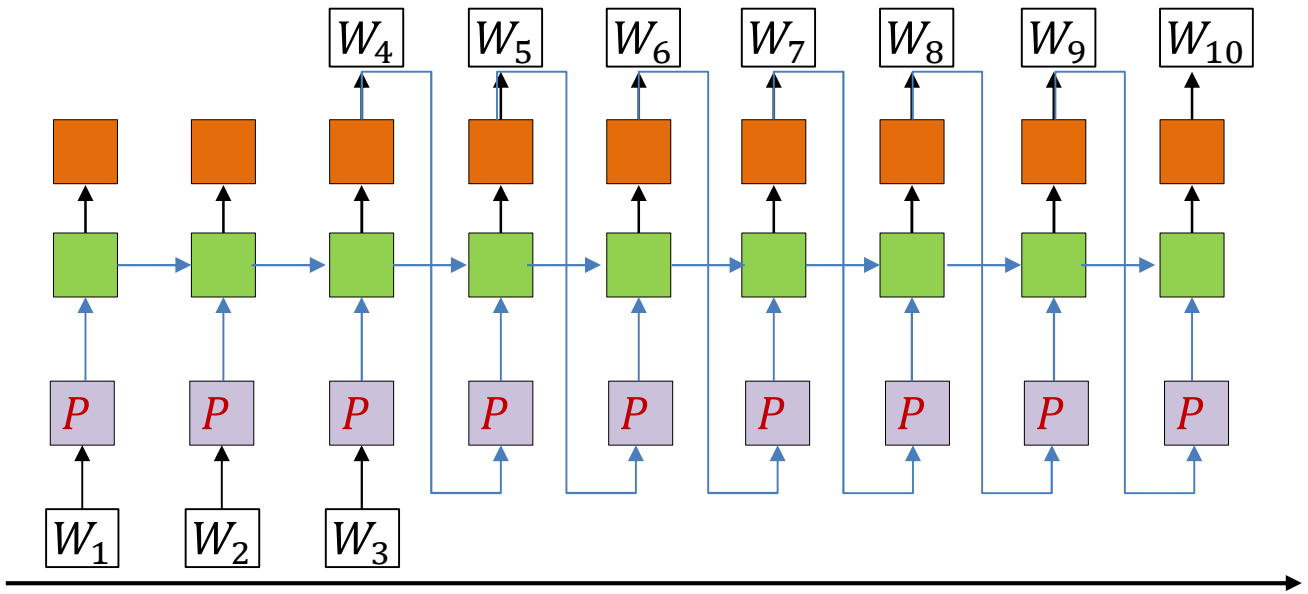
  <sos> four score and eight <eos>

  – 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

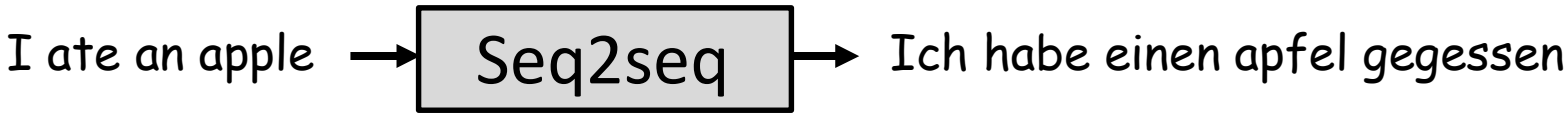
- Sometimes we will use a single symbol to represent both start and end of sentence, e.g just <eos> , or even a separate symbol, e.g. <s>

Generating Language: Synthesis



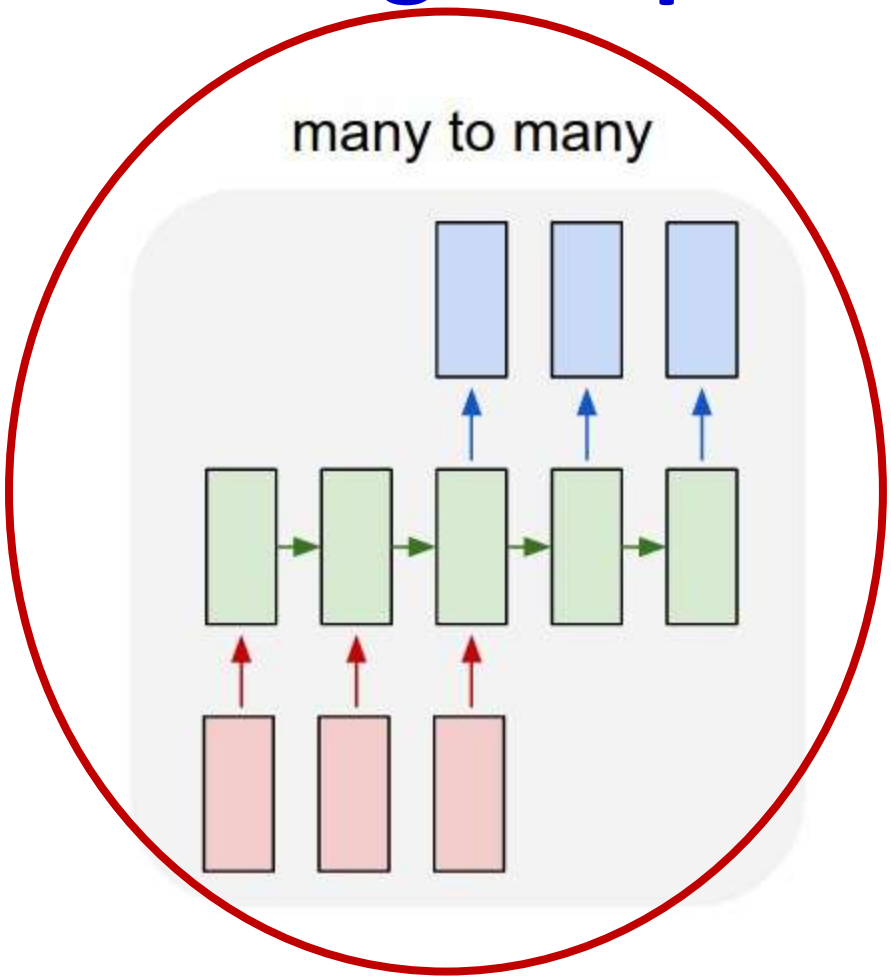
- Feed the drawn word as the next word in the series
  - And draw the next word by sampling from the output probability distribution
- Continue this process until we draw an <eos>
  - Or we decide to terminate generation based on some other criterion

Returning our problem



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

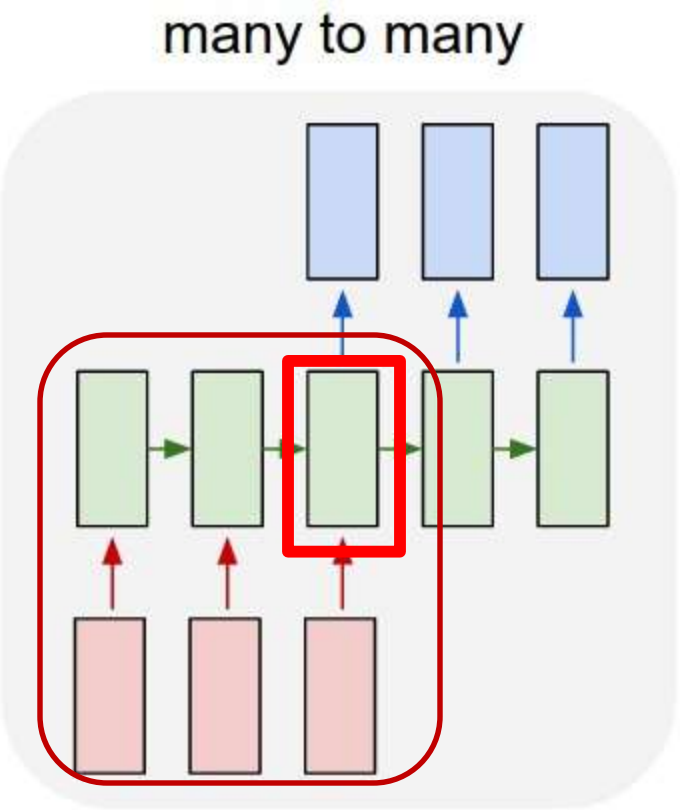
Modelling the problem



- *Delayed* sequence to sequence

Modelling the problem

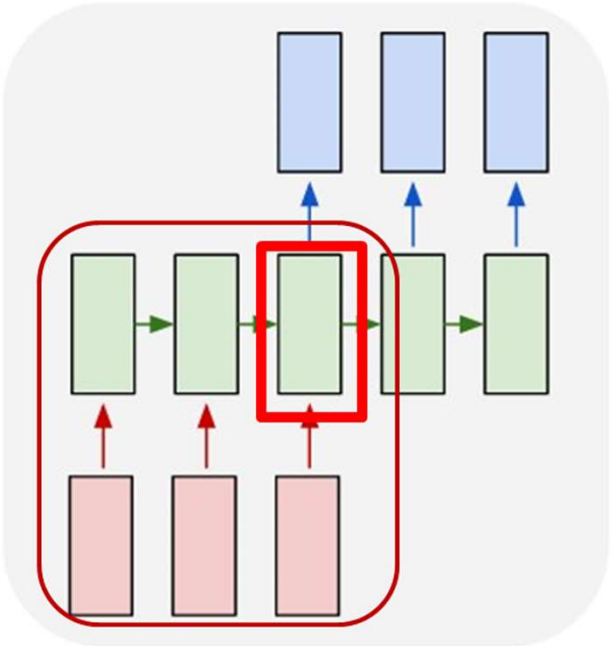
First process the input and generate a hidden representation for it



- *Delayed* sequence to sequence

Pseudocode

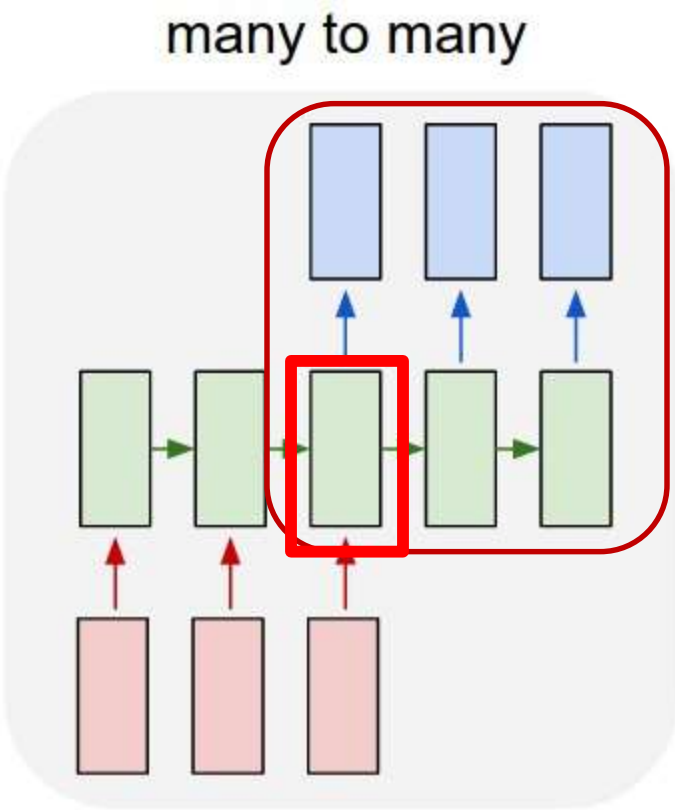
```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
H = h(T-1)
```



"**RNN_input**" may be a multi-layer RNN of any kind

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

Pseudocode

```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
H = h(T-1)


# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1))
    y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

Pseudocode

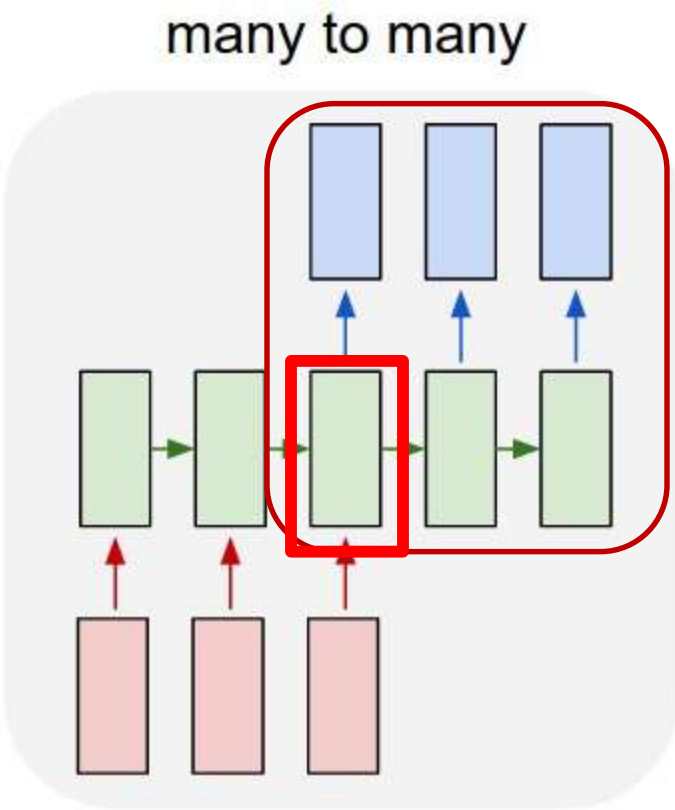
```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
for t = 0:T-1  # Including both ends of the index
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
H = h(T-1)
```

The output at each time is a probability distribution over symbols.
We draw a word from this distribution

```
# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1))
    y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

Modelling the problem

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

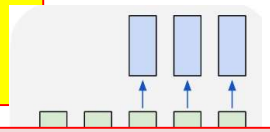
Pseudocode

Changing this output at time t does not affect the output at t+1

E.g. If we have drawn "It was a" vs "It was an", the probability that the next word is "dark" remains the same (dark must ideally not follow "an")

This is because the output at time t does not influence the computation at t+1

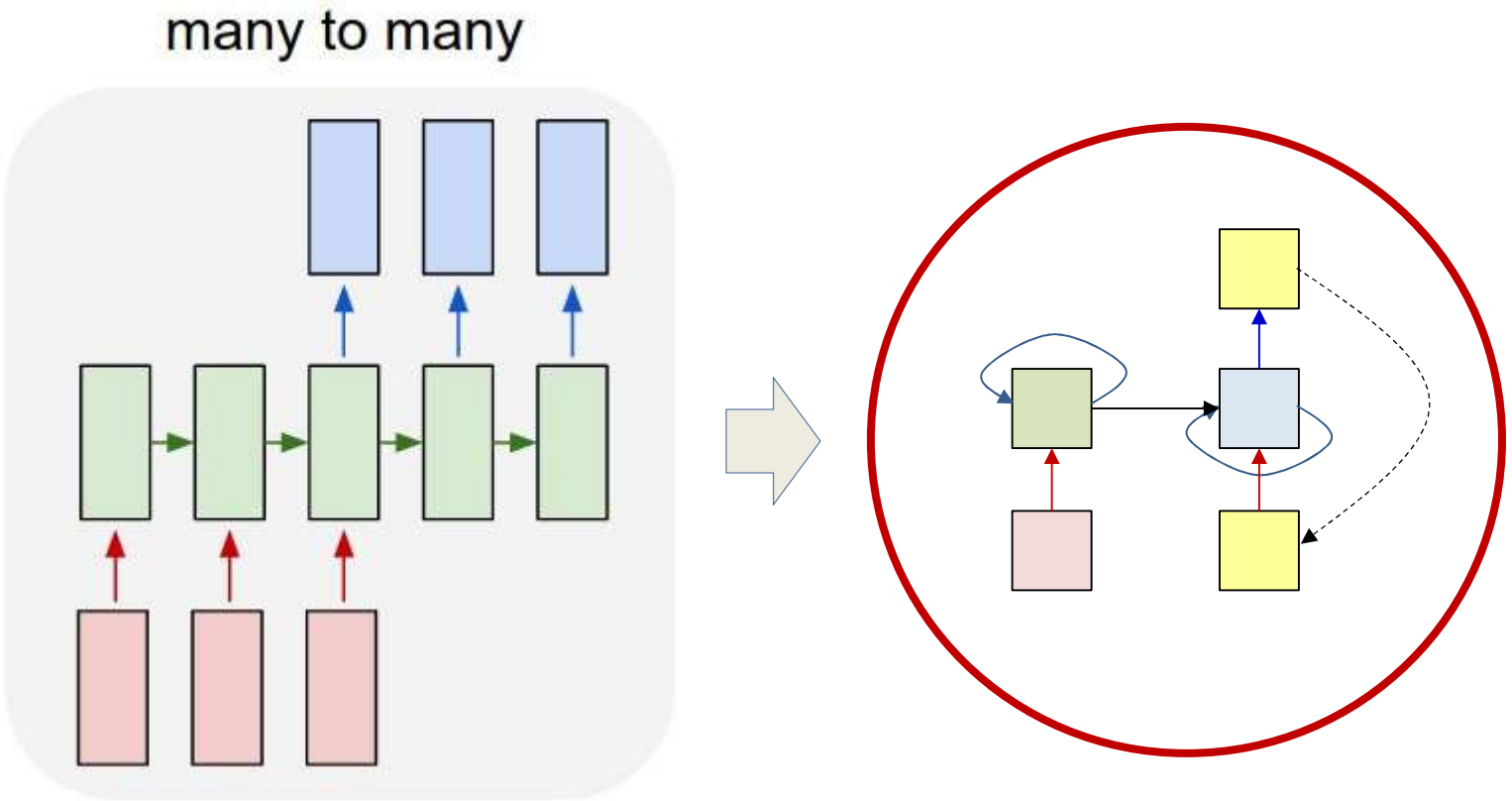
dex



The RNN recursion only considers the hidden state h(t-1) from the previous time and not the actual output word $y_{out}(t-1)$

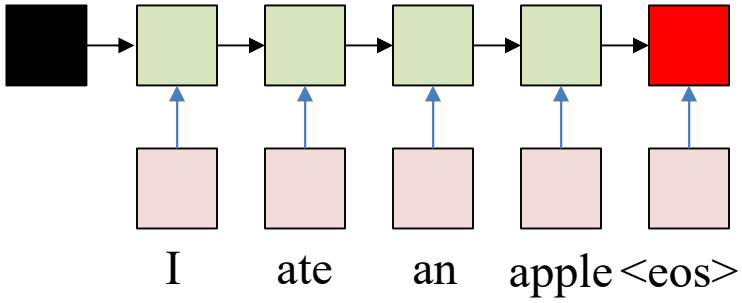
```
# Now generate the output y_out(1) y_out(2)
t = 0
h_out(0) = H
do
     t = t+1
     [y(t),h_out(t)] = RNN_output_step(h_out(t-1))
      y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

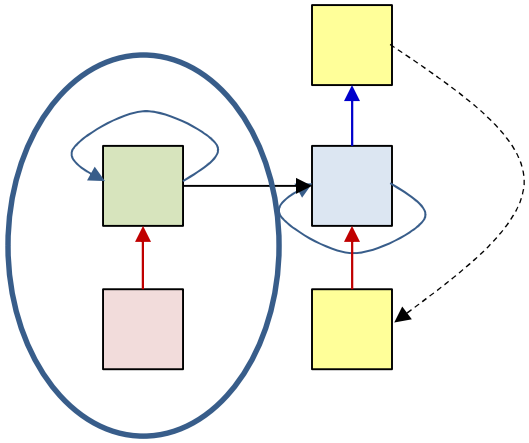
Modelling the problem



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

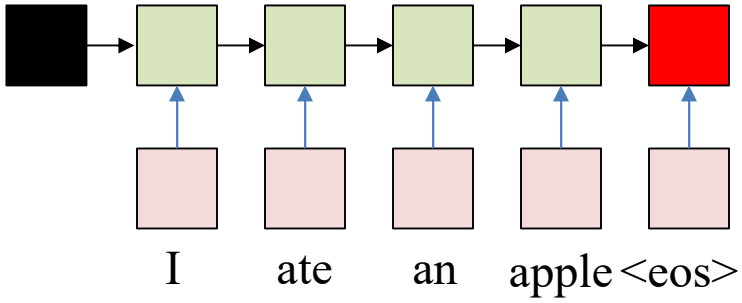
The “simple” translation model

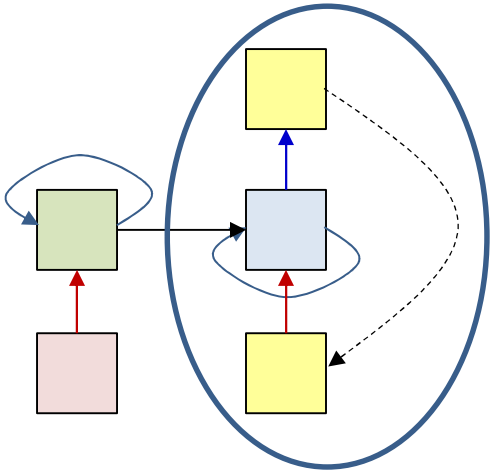




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

The “simple” translation model

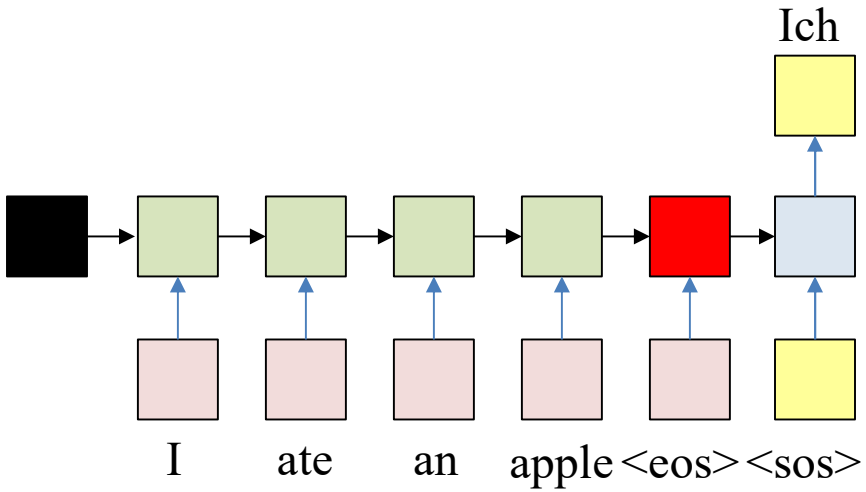




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

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

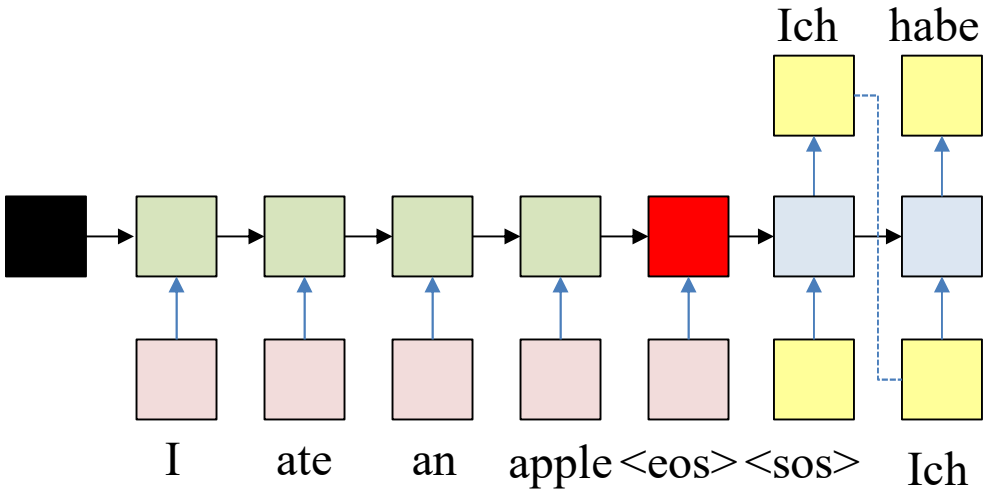
The “simple” translation model



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

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

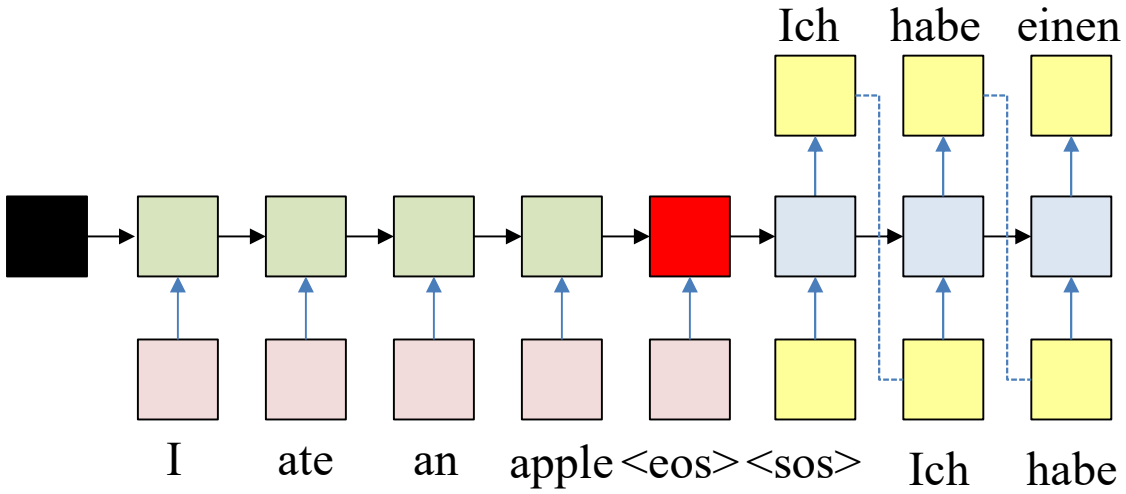
The “simple” translation model



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

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

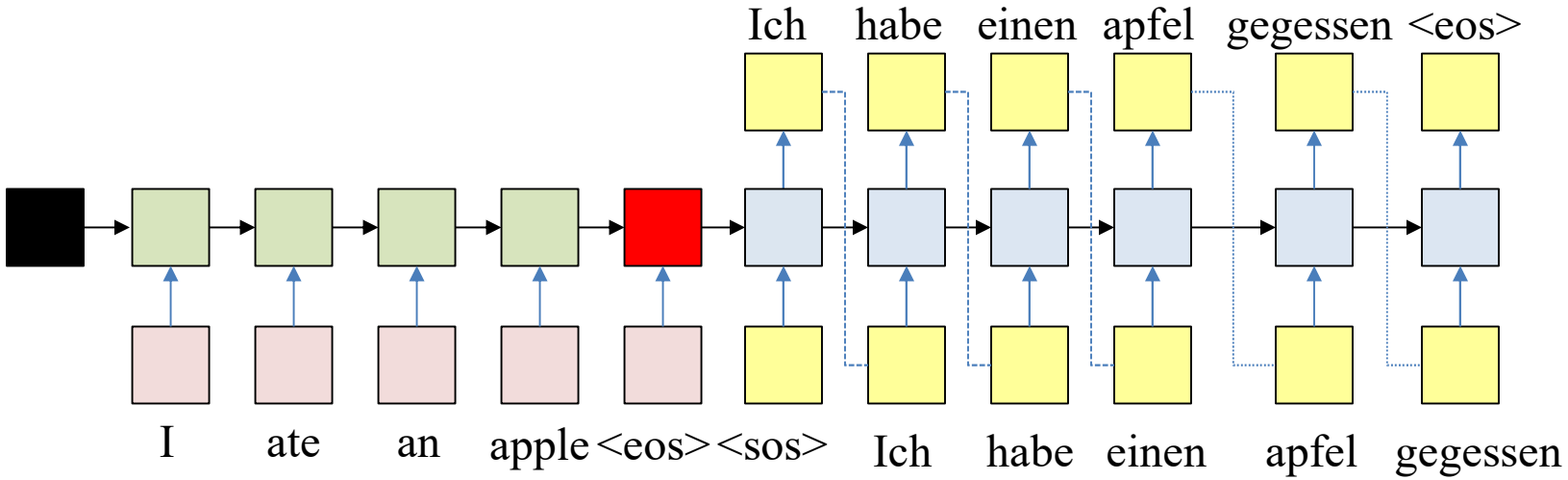
The “simple” translation model



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

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

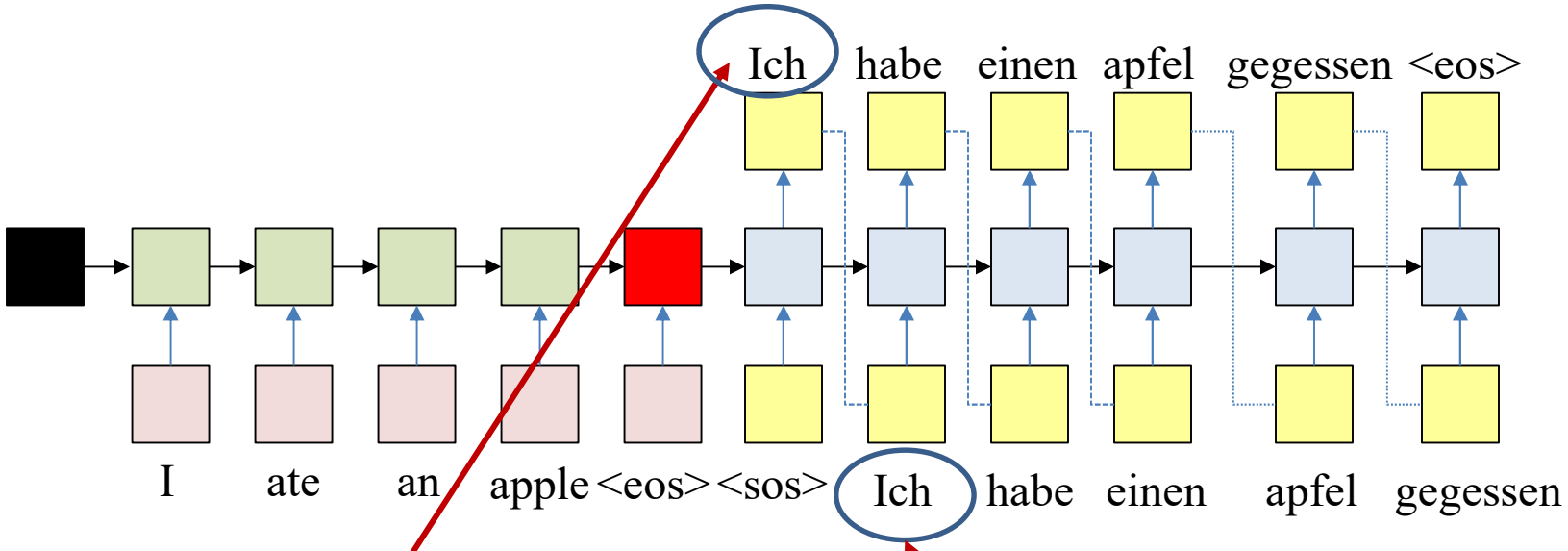
The “simple” translation model



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

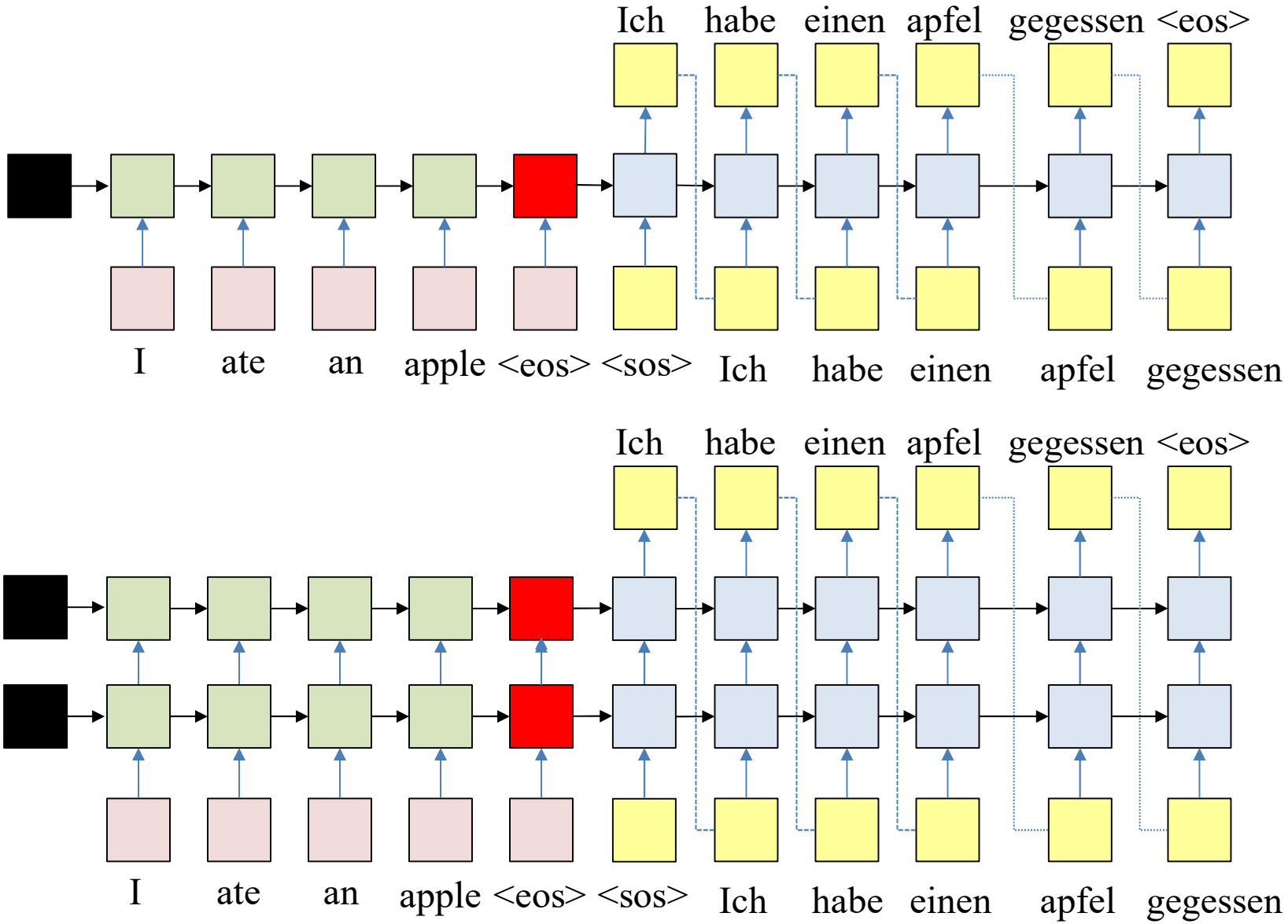
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

The “simple” translation model



Note that drawing a different word here

Would result in a different word being input here, and as a result the output here and subsequent outputs would all change



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

Pseudocode

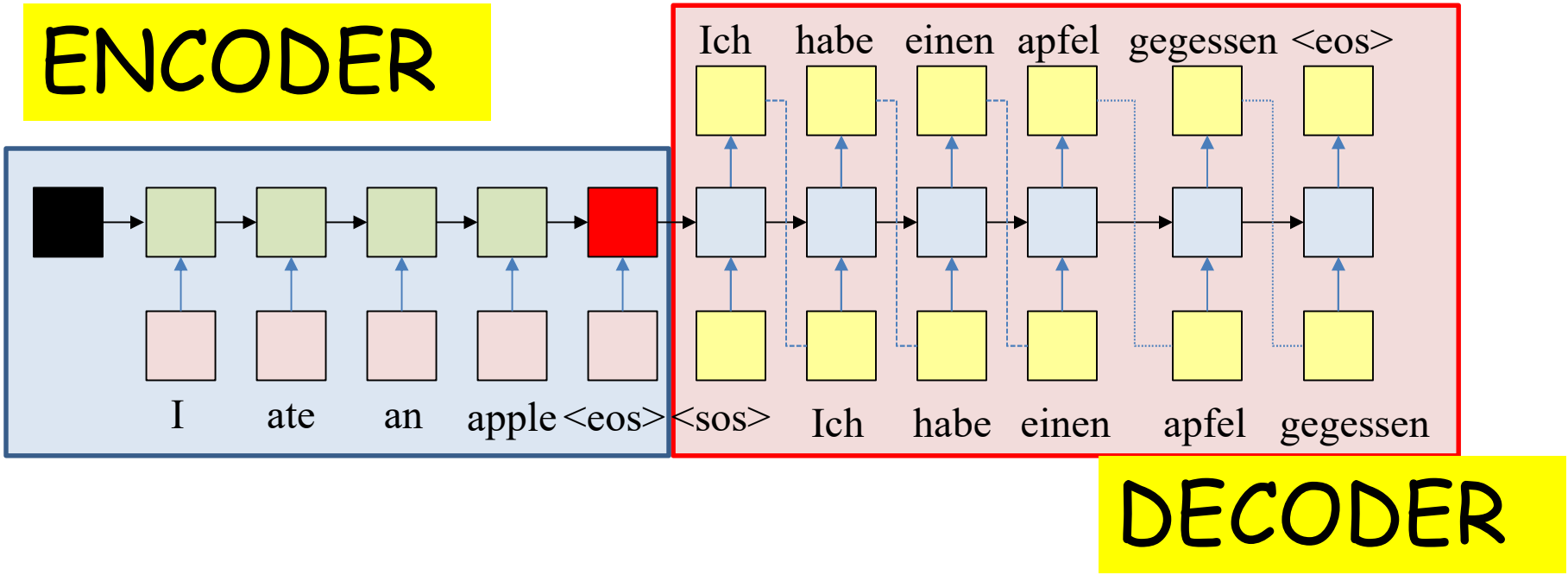
```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
t = 0
do
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
until x(t) == "<eos>"
H = h(T-1)

# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1))
    y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

Pseudocode

```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
t = 0
do
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
until x(t) == "<eos>"
H = h(T-1)

# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1))
    y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

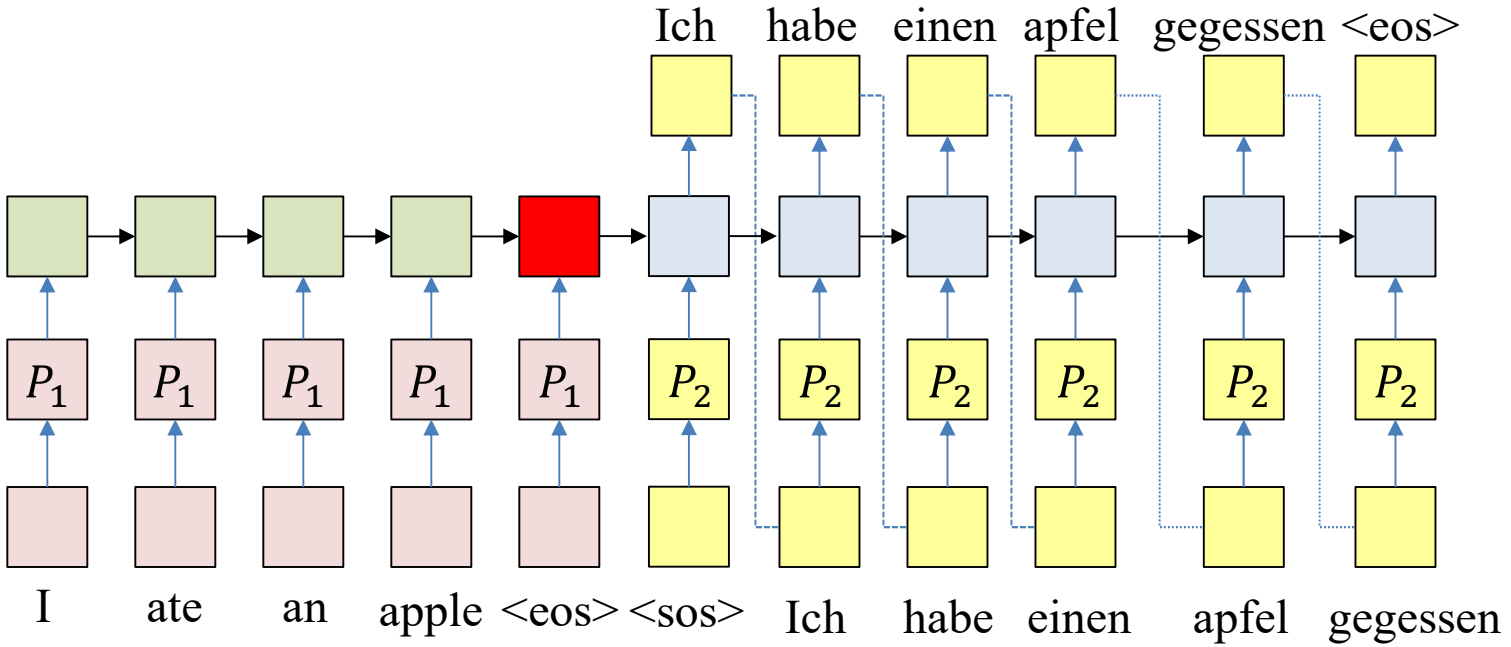
Drawing a different word at t will change the next output since $y_{out}(t)$ is fed back as input

The “simple” translation model



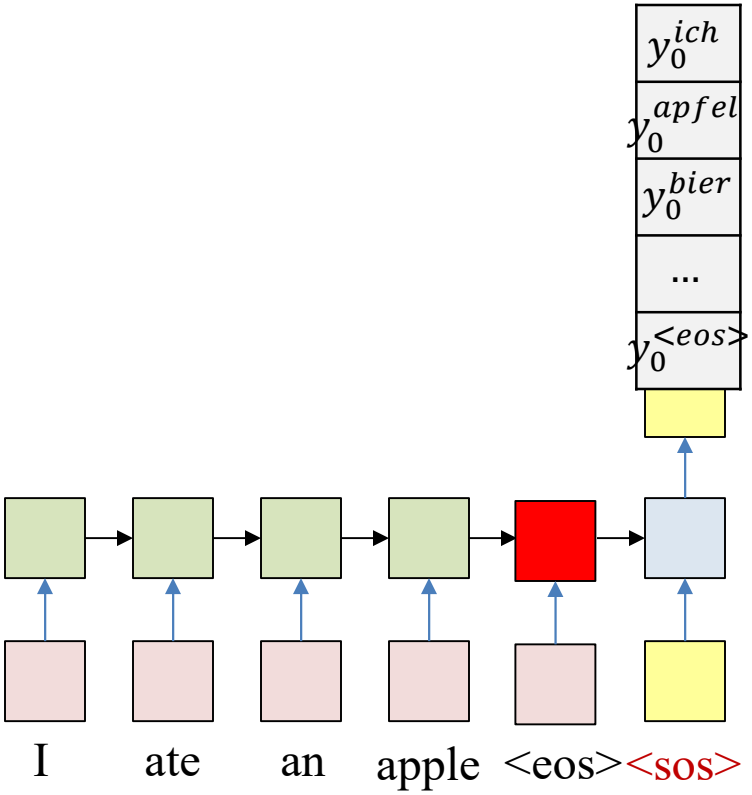
- The recurrent structure that extracts the hidden representation from the input sequence is the *encoder*
- The recurrent structure that utilizes this representation to produce the output sequence is the *decoder*

The "simple" translation model



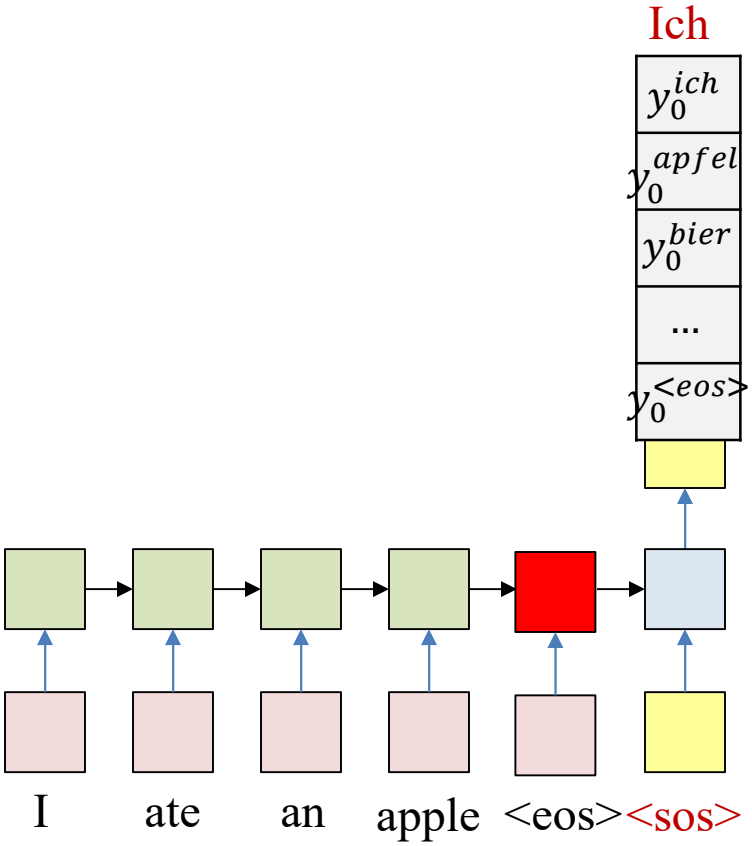
- A more detailed look: The one-hot word representations may be compressed via embeddings
  - Embeddings will be learned along with the rest of the net
  - In the following slides we will not represent the projection matrices

What the network actually produces



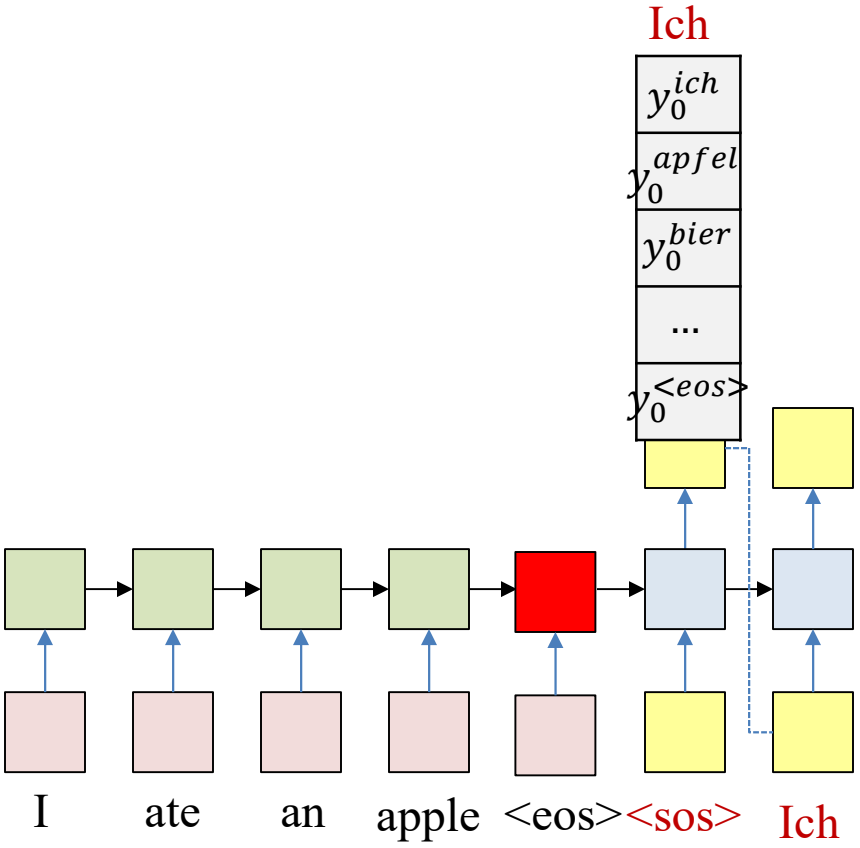
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



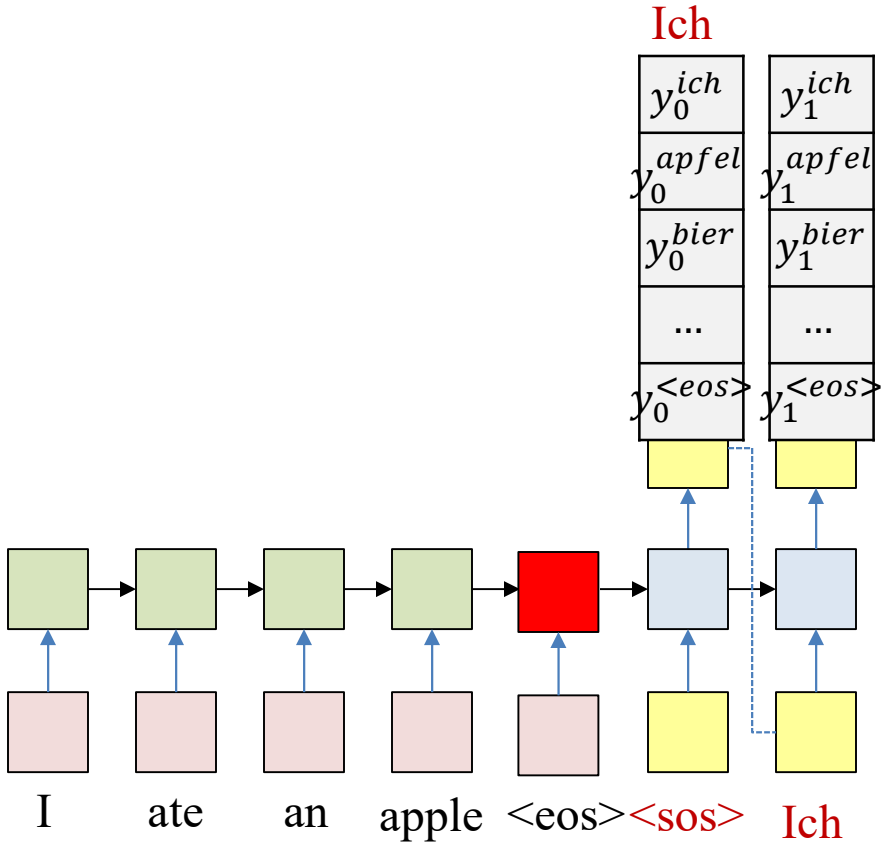
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



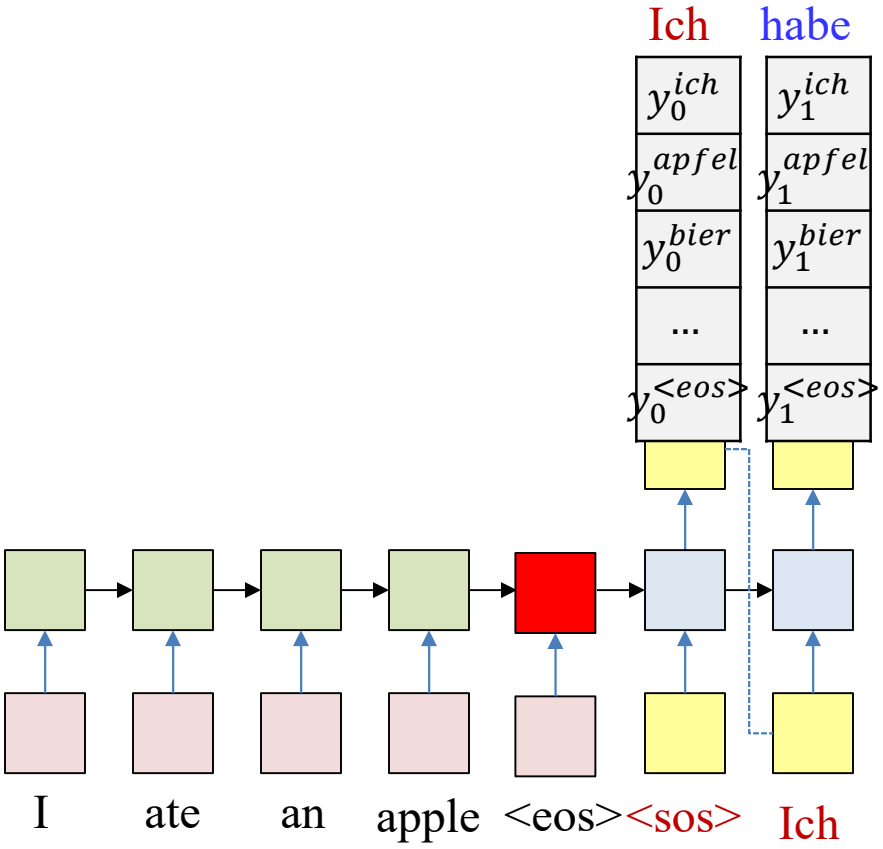
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



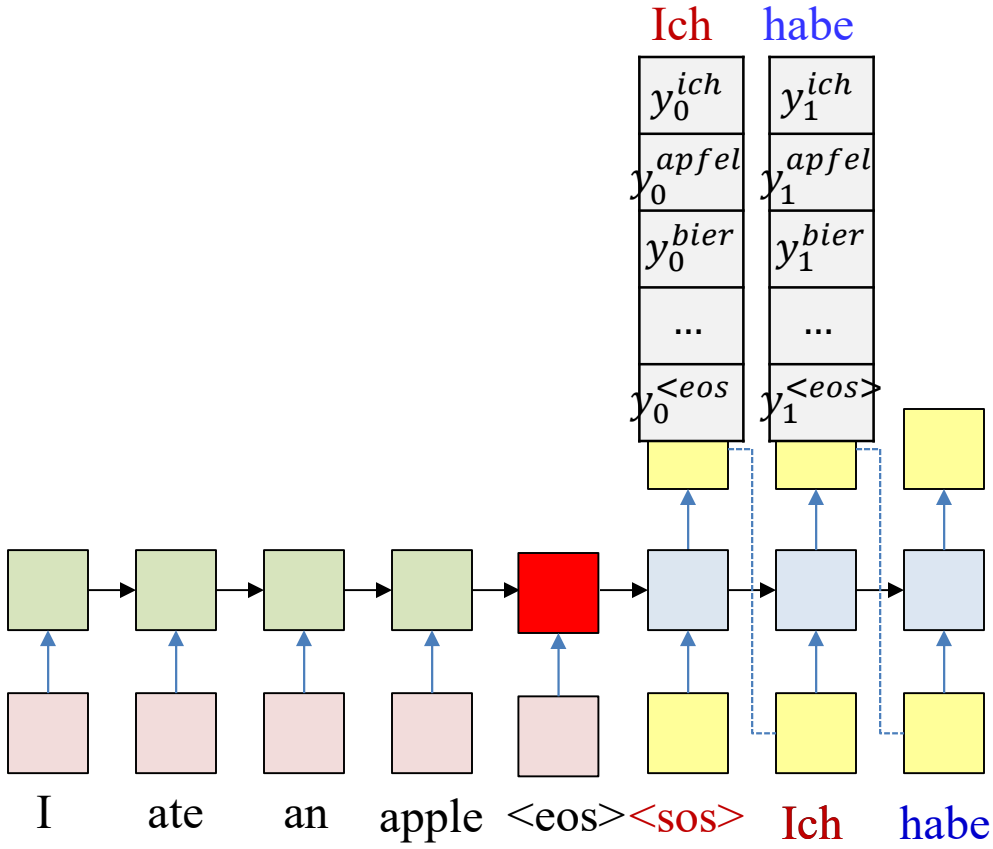
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



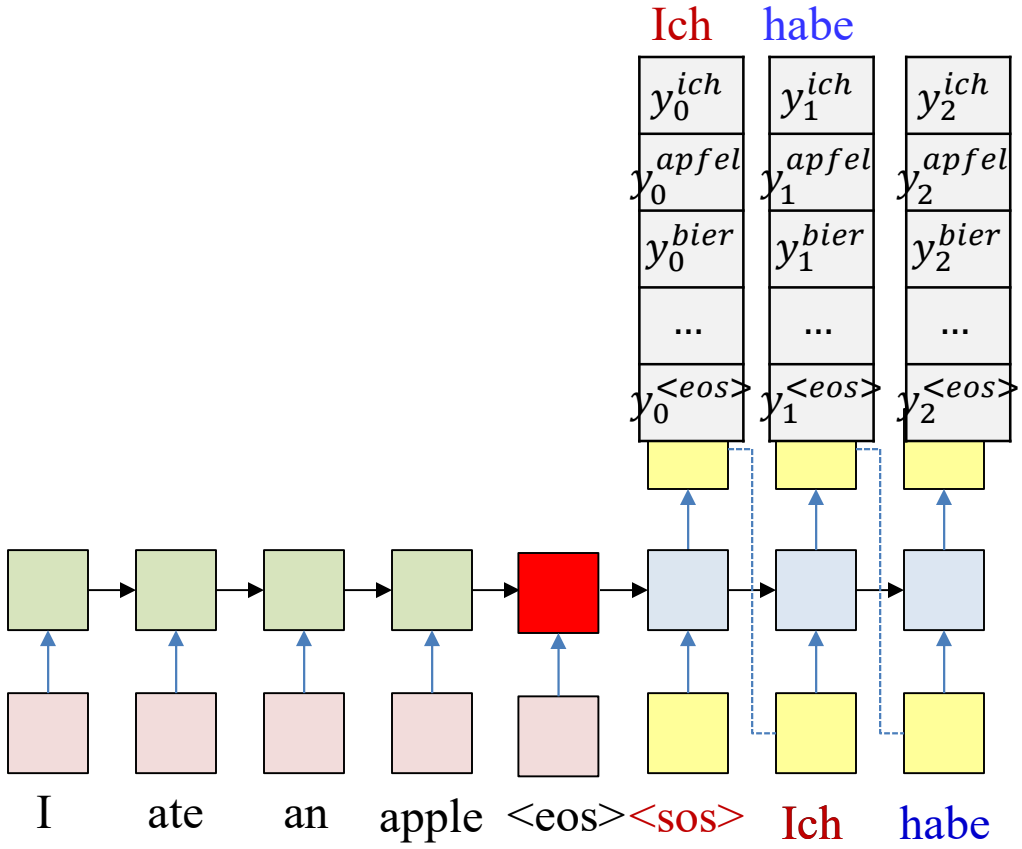
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



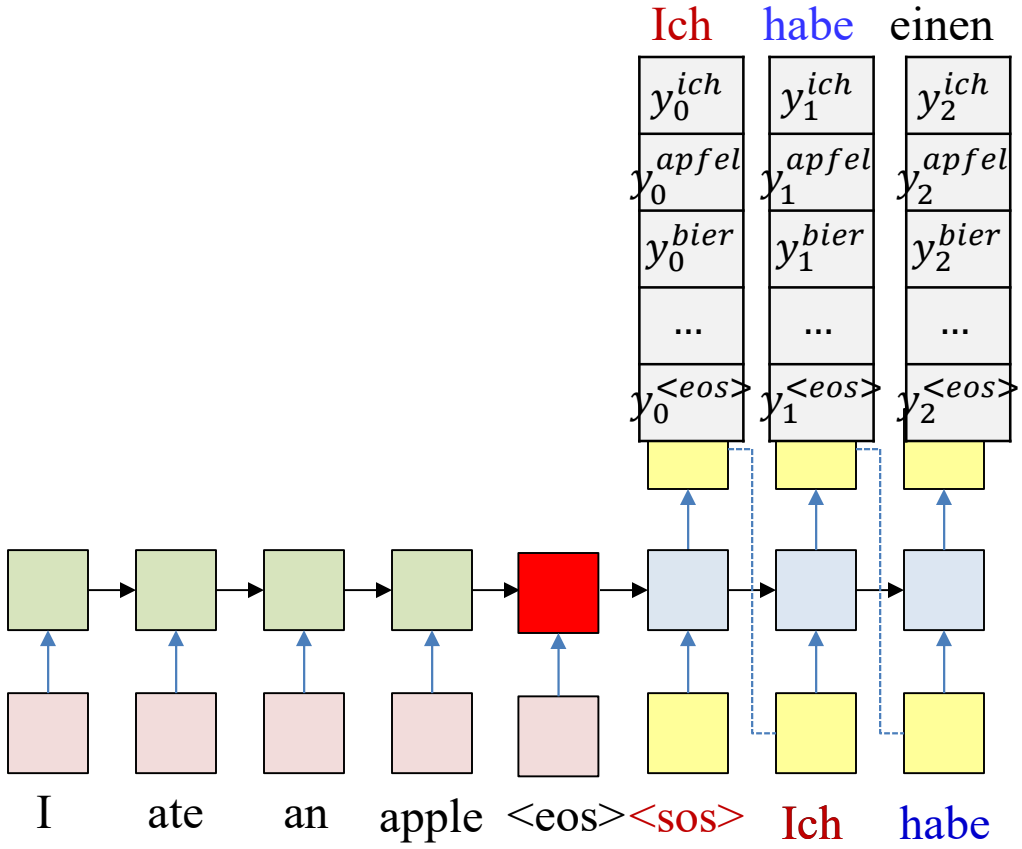
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



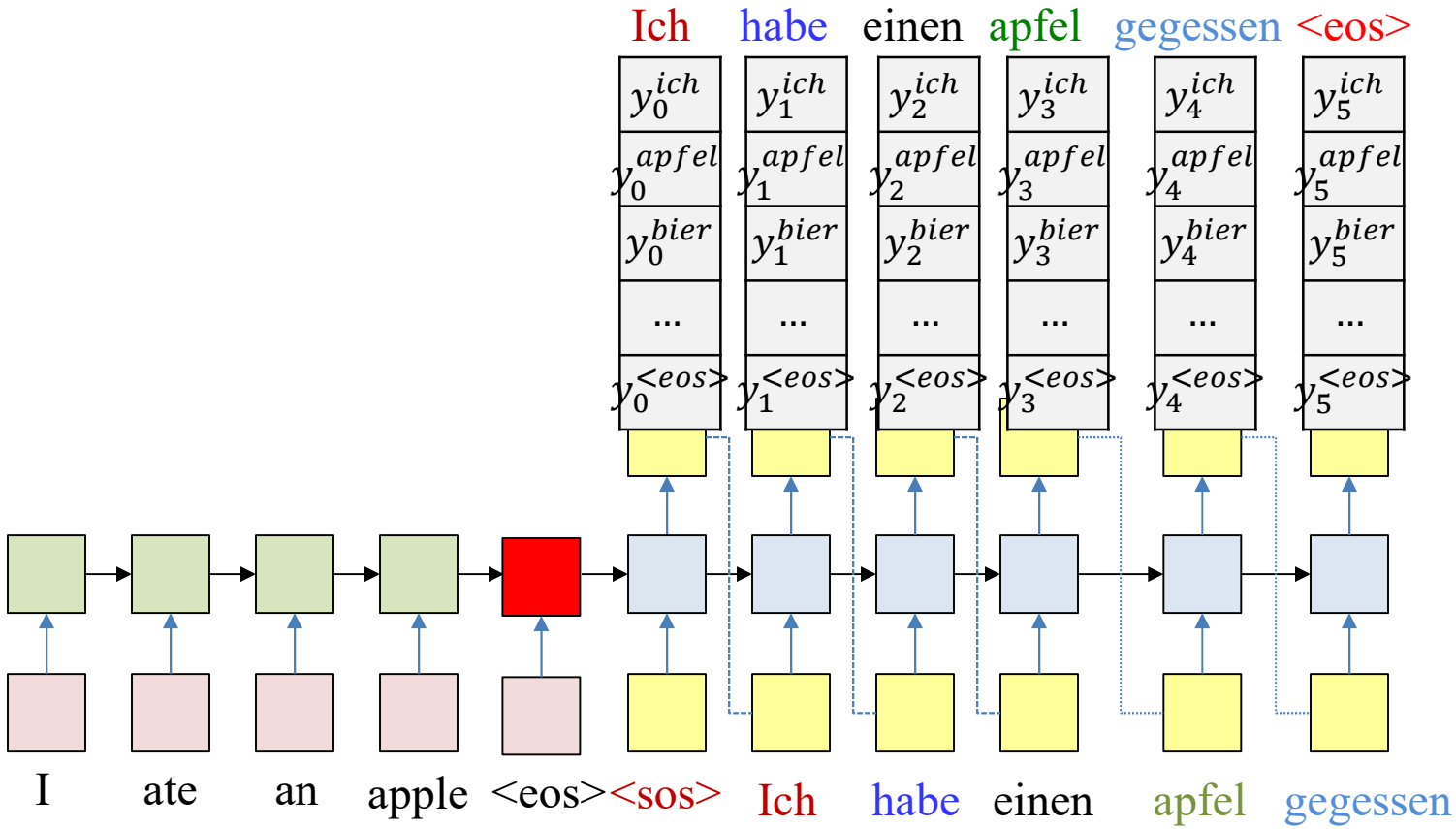
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



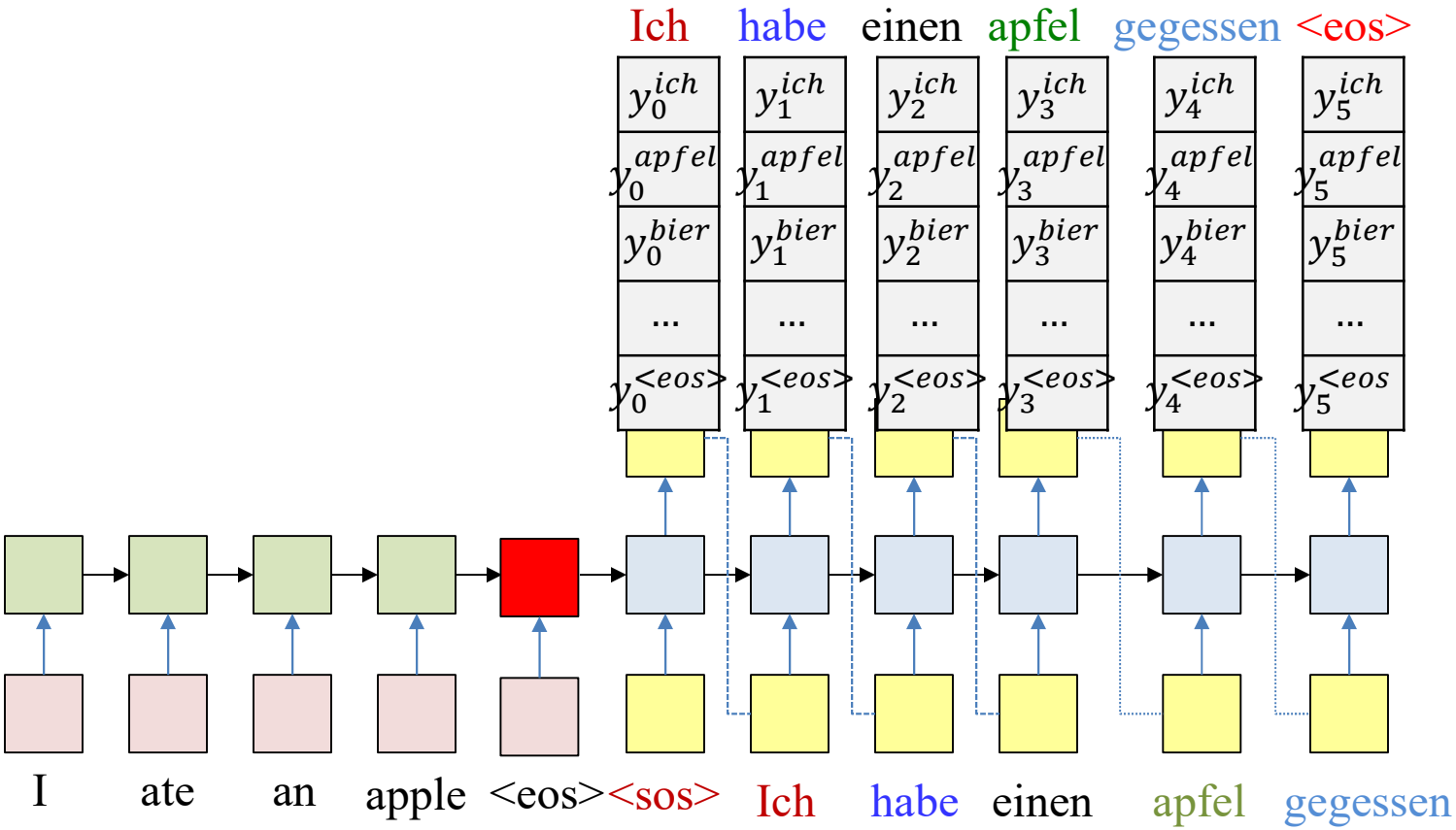
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

What the network actually produces



- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until k
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

Generating an output from the net



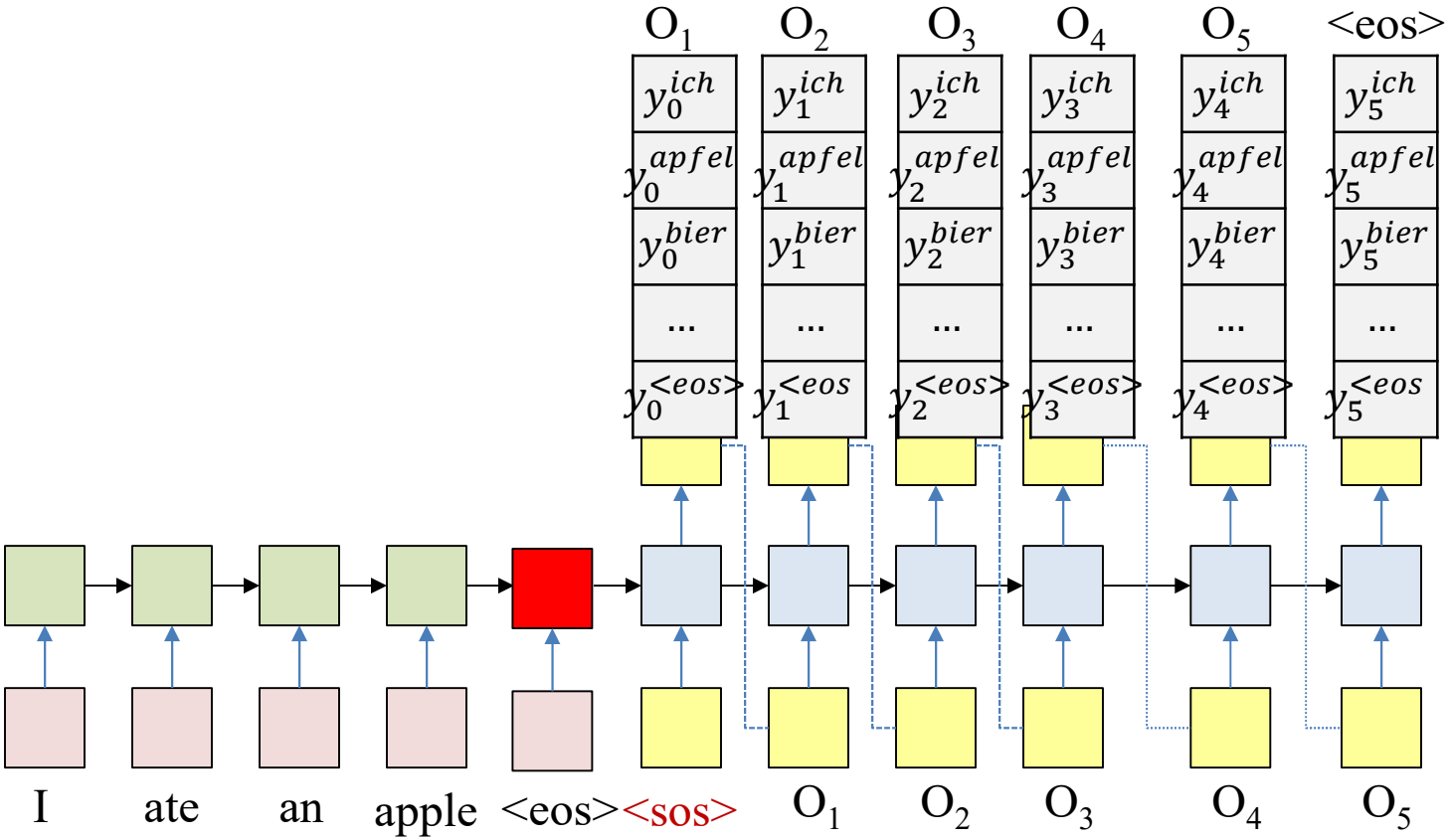
- At each time the network produces a probability distribution over words, given the entire input and entire output sequence so far
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time
- The process continues until an <eos> is generated

Pseudocode

```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
t = 0
do
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
until x(t) == "<eos>"
H = h(T-1)

# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1))
    y_out(t) = draw_word_from(y(t))
until y_out(t) == <eos>
```

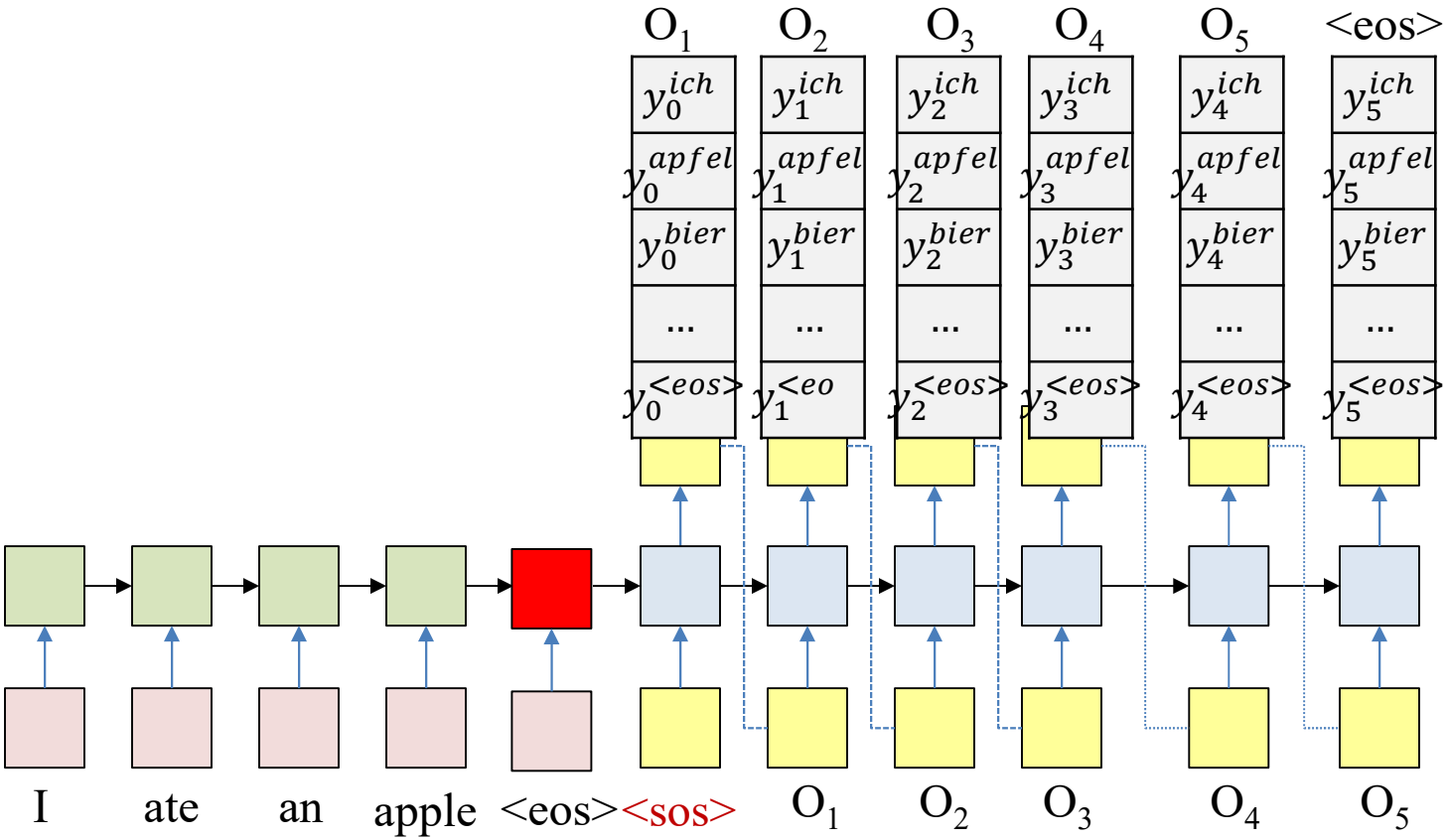
What is this magic operation?

The probability of the output



$$P(O_1, \dots, O_L | I_1, \dots, I_N)$$
$$= P(O_1 | I_{1\dots N}, \dots, I_N) P(O_2 | O_1, I_1, \dots, I_N) P(O_3 | O_1, O_2, I_1, \dots, I_N) \dots P(O_L | O_1, \dots, O_{L-1}, I_1, \dots, I_N)$$
$$= y_1^{O_1} y_2^{O_2} \dots y_L^{O_L}$$

The probability of the output

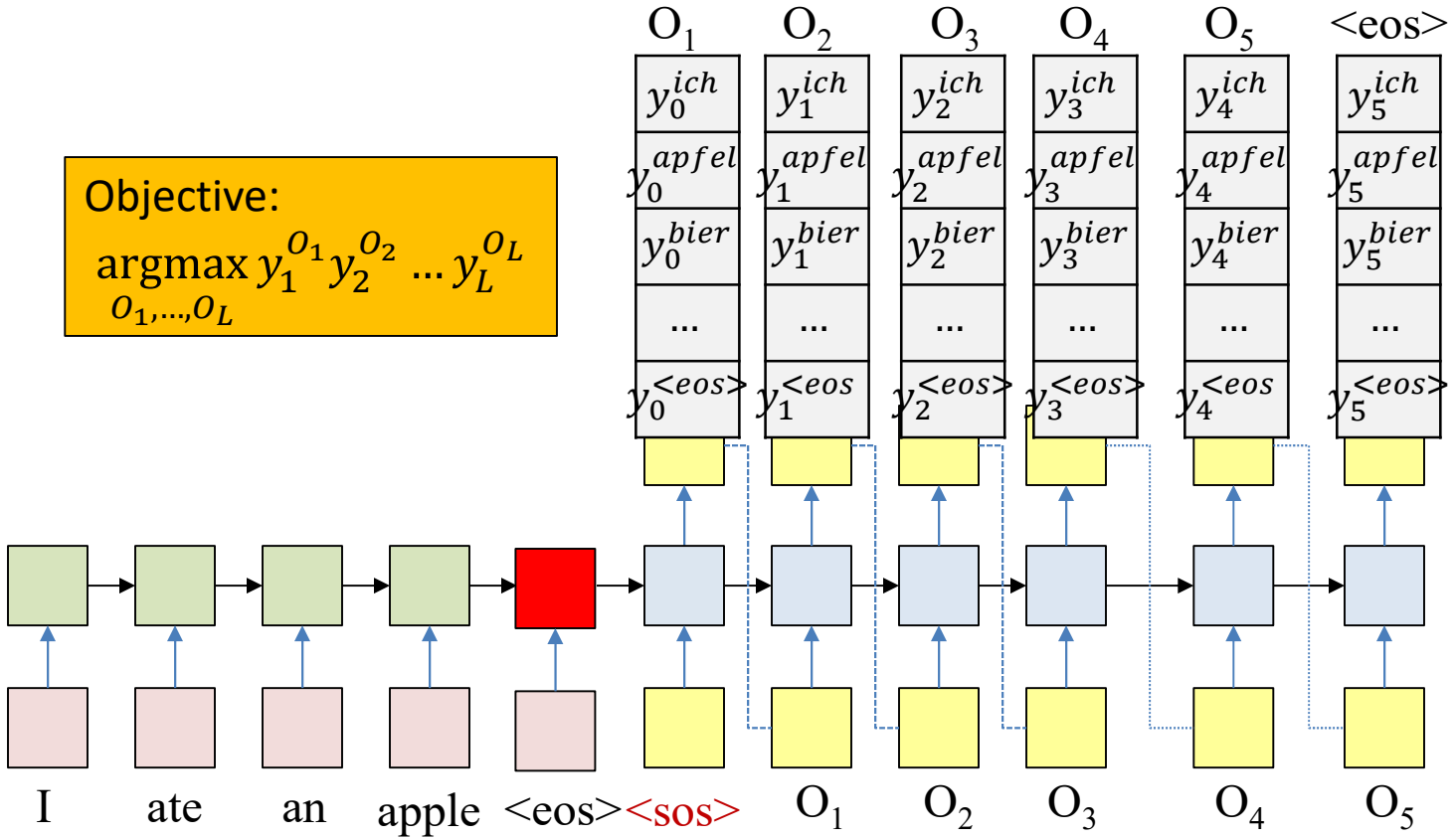


- ***The objective of drawing: Produce the most likely output (that ends in an <eos>)***

$$\underset{O_1,\dots,O_L}{\operatorname{argmax}} P\left(O_1,\dots,O_L|W_1^{in},\dots,W_N^{in}\right)$$

$$= \underset{O_1,\dots,O_L}{\operatorname{argmax}} y_1^{O_1} y_2^{O_2} \dots y_L^{O_L}$$

Greedy drawing



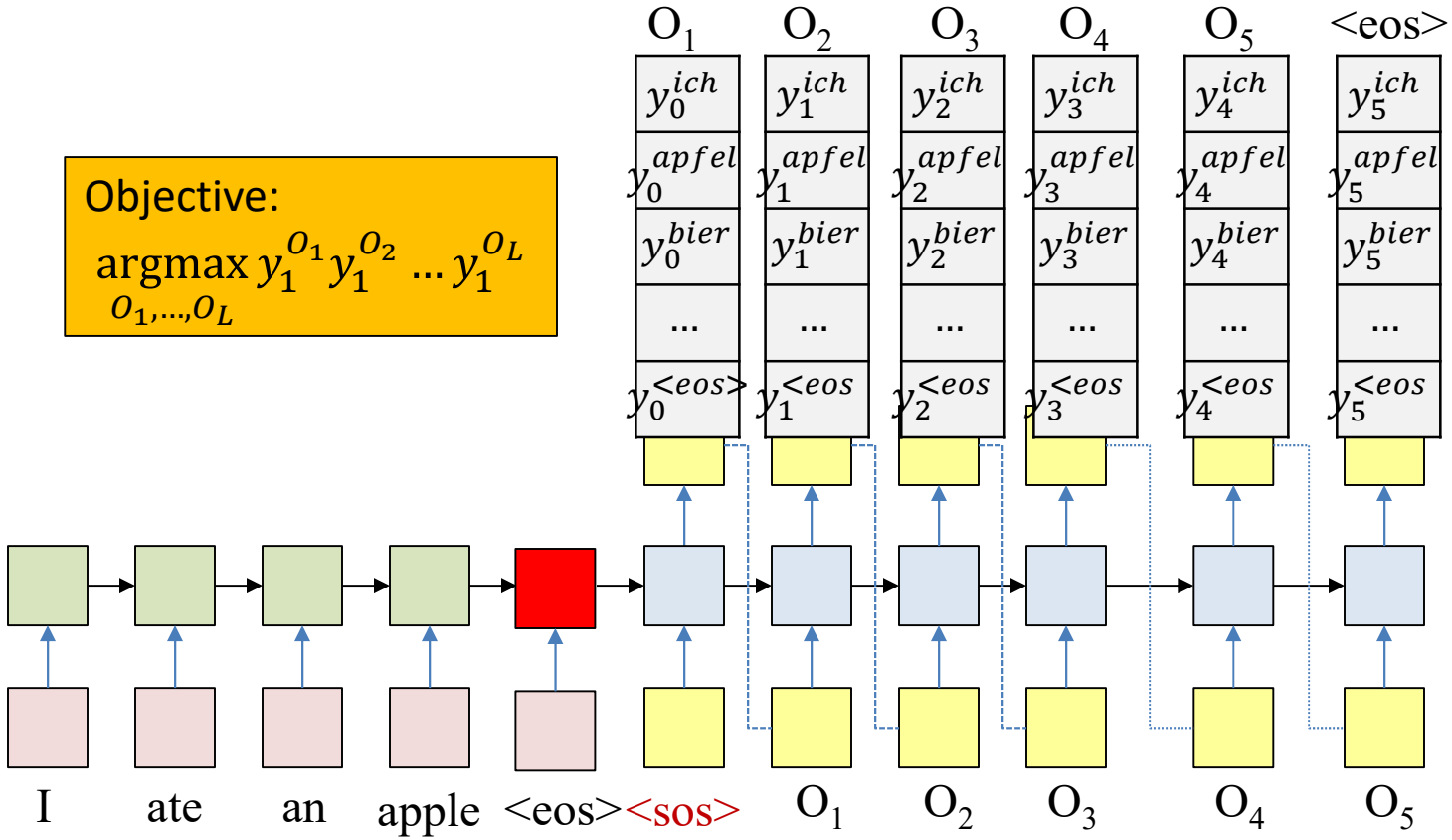
- So how do we draw words at each time to get the most likely word sequence?
- *Greedy* answer – select the most probable word at each time

Pseudocode

```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
t = 0
do
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
until x(t) == "<eos>"
H = h(T-1)

# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1))
    y_out(t) = argmax_i(y(t,i))
until y_out(t) == <eos>
```

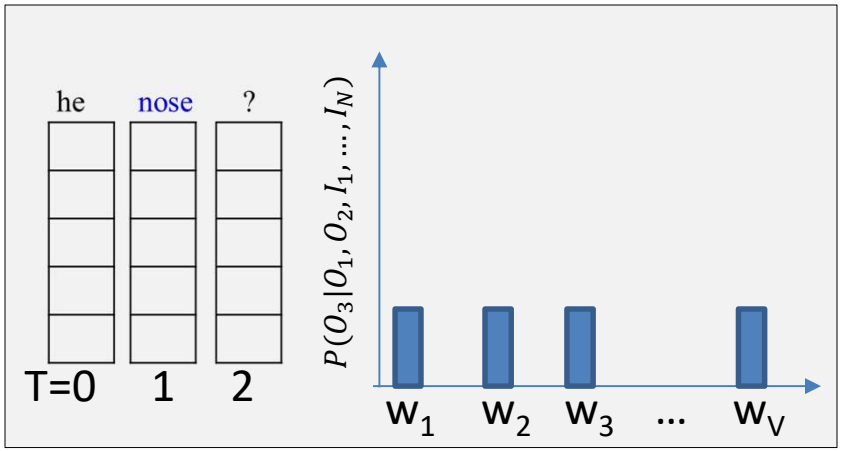
Select the most likely output at each time

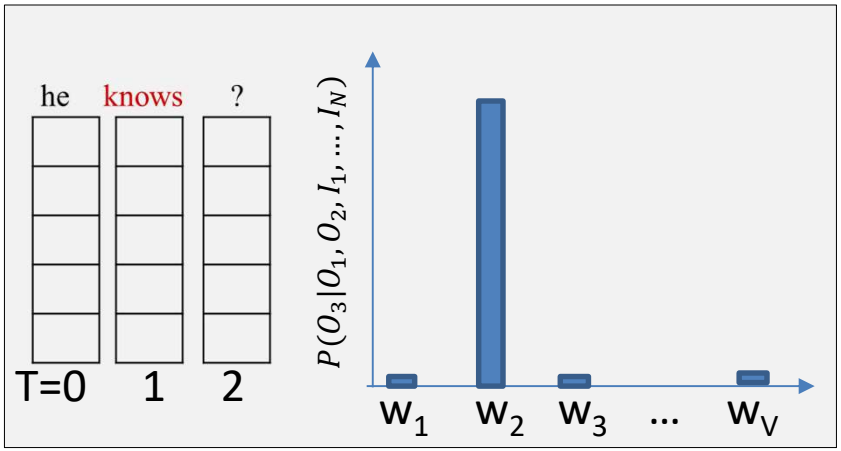
Greedy drawing



- Cannot just pick the most likely symbol at each time
  - That may cause the distribution to be more "confused" at the next time
  - Choosing a different, less likely word could cause the distribution at the next time to be more peaky, resulting in a more likely output overall

Greedy is not good



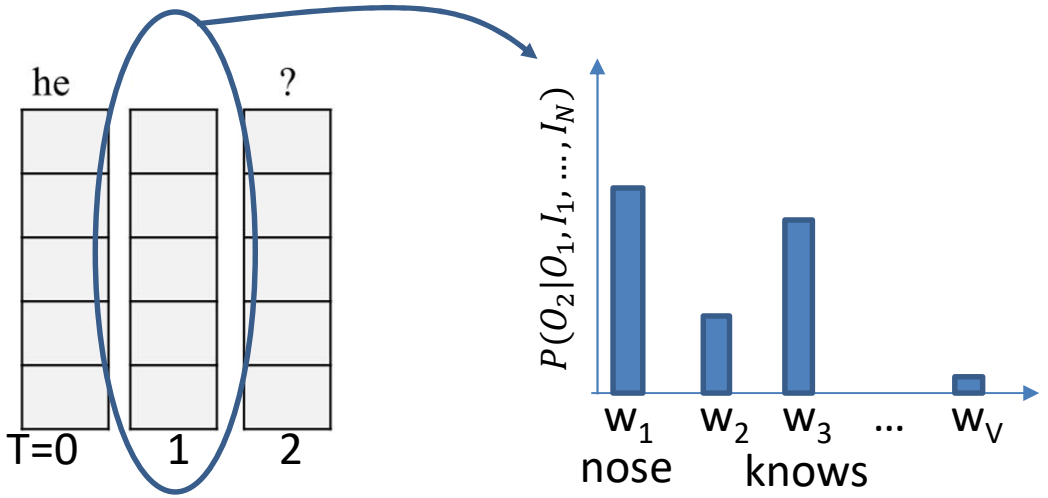


- Hypothetical example (from English speech recognition : Input is speech, output must be text)
- "Nose" has highest probability at t=2 and is selected
  - The model is very confused at t=3 and assigns low probabilities to many words at the next time
  - Selecting any of these will result in low probability for the entire 3-word sequence
- "Knows" has slightly lower probability than "nose", but is still high and is selected
  - "he knows" is a reasonable beginning and the model assigns high probabilities to words such as "something"
  - Selecting one of these results in higher overall probability for the 3-word sequence

Greedy is not good

What should we have chosen at t=2??

Will selecting "nose" continue to have a bad effect into the distant future?

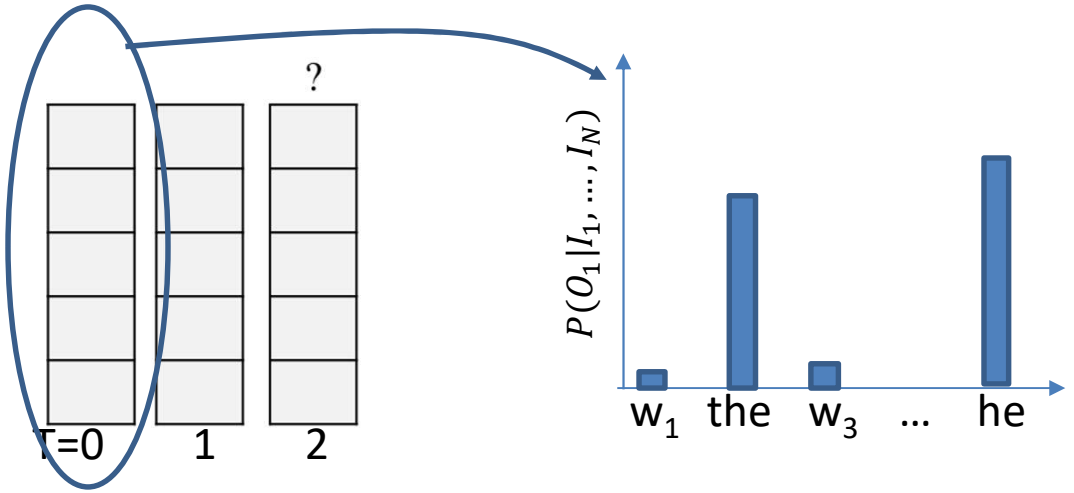


- Problem: Impossible to know a priori which word leads to the more promising future
  - Should we draw "nose" or "knows"?
  - Effect may not be obvious until several words down the line
  - Or the choice of the wrong word early may cumulatively lead to a poorer overall score over time

Greedy is not good

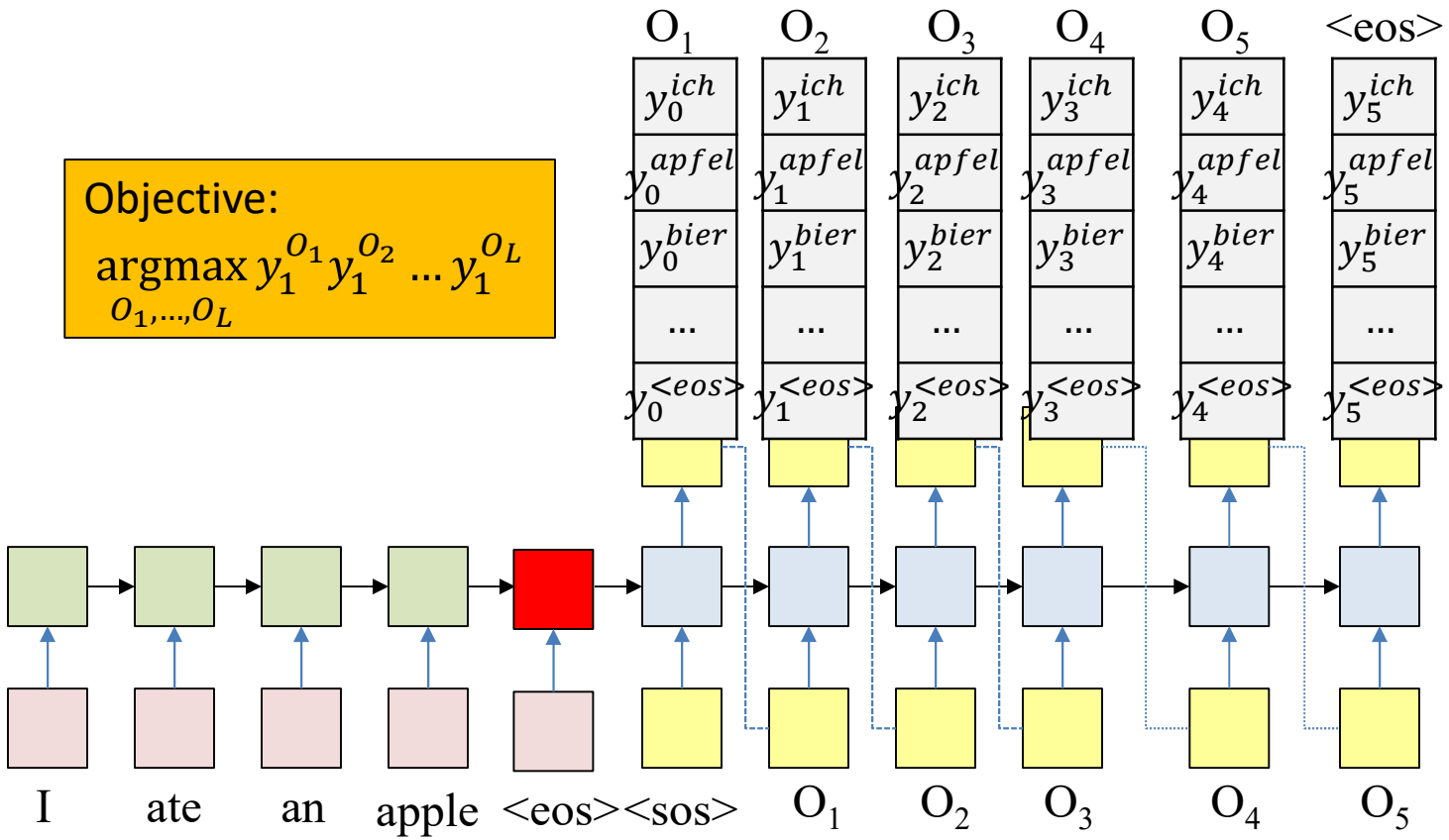
What should we have chosen at t=1??

Choose “the” or “he”?



- Problem: Impossible to know a priori which word leads to the more promising future
  - Even earlier: Choosing the lower probability “the” instead of “he” at T=0 may have made a choice of “nose” more reasonable at T=1..
- In general, making a poor choice at any time commits us to a poor future
  - But we cannot know at that time the choice was poor

Drawing by random sampling



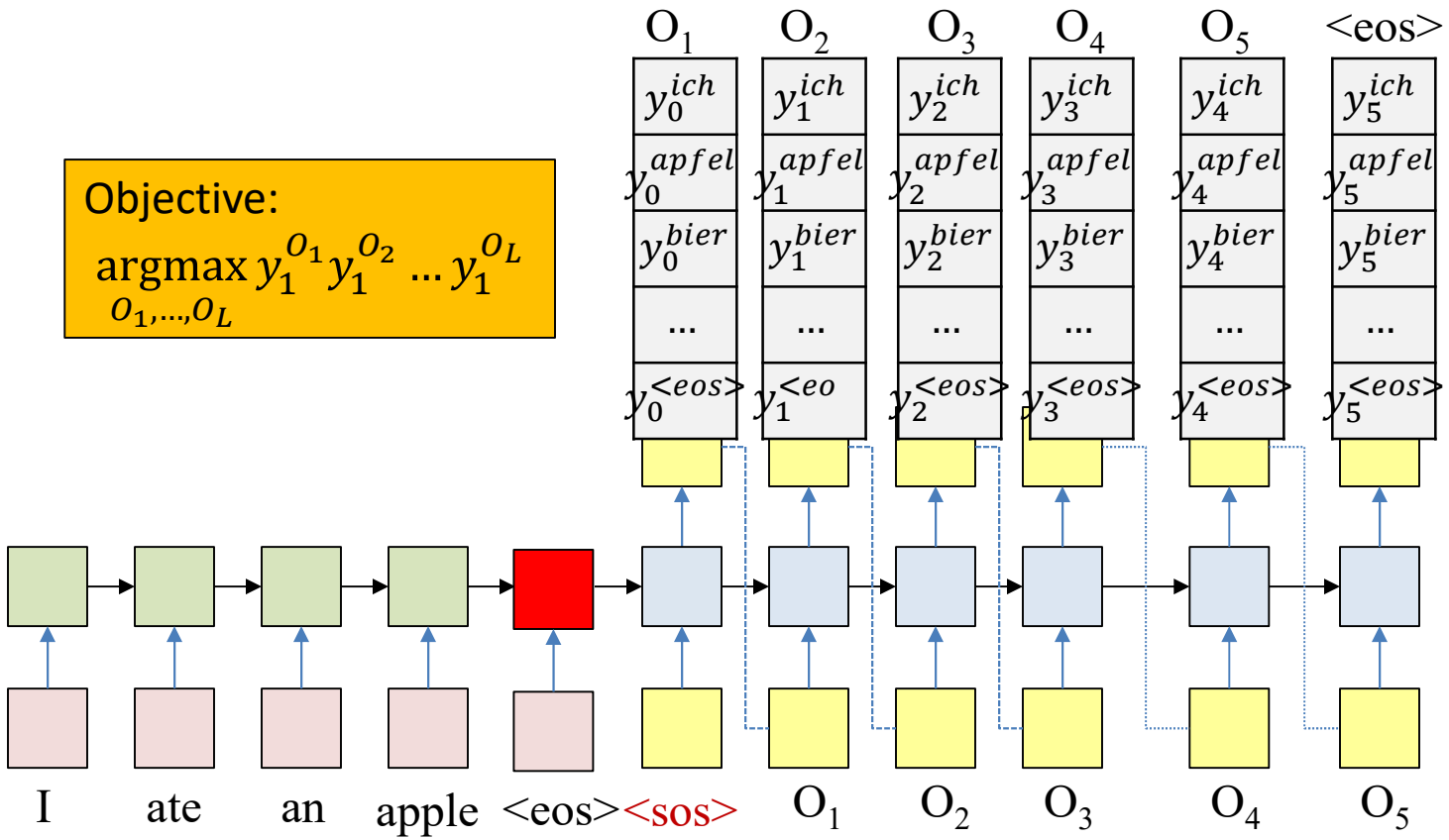
- Alternate option: Randomly draw a word at each time according to the output probability distribution

Pseudocode

```
# First run the inputs through the network
# Assuming h(-1,l) is available for all layers
t = 0
do
    [h(t),..] = RNN_input_step(x(t),h(t-1),...)
until x(t) == "<eos>"
H = h(T-1)

# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
    t = t+1
    [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1))
    y_out(t) = sample(y(t))
until y_out(t) == <eos>
```

Randomly sample from the output distribution.

Drawing by random sampling

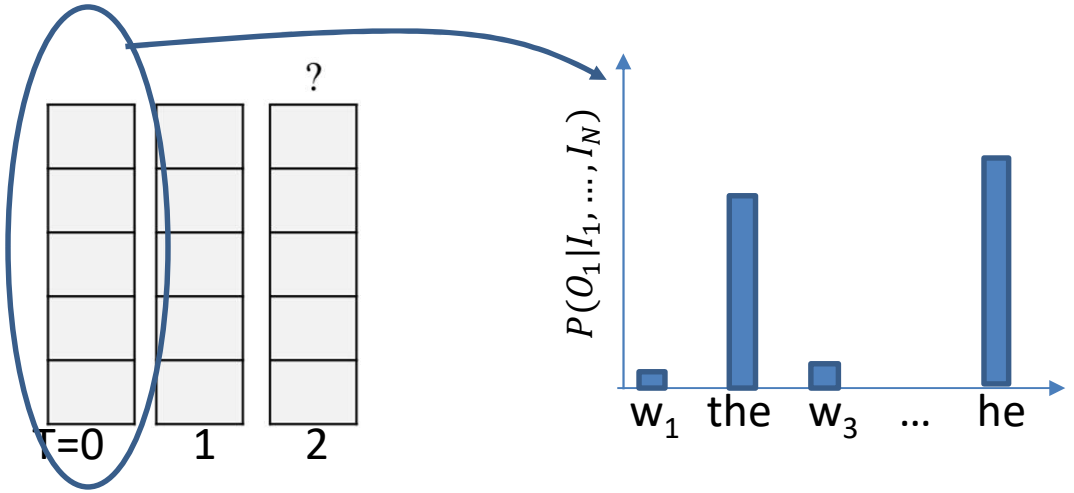


- Alternate option: Randomly draw a word at each time according to the output probability distribution
  - Unfortunately, not guaranteed to give you the most likely output
  - May sometimes give you more likely outputs than greedy drawing though

Your choices can get you stuck

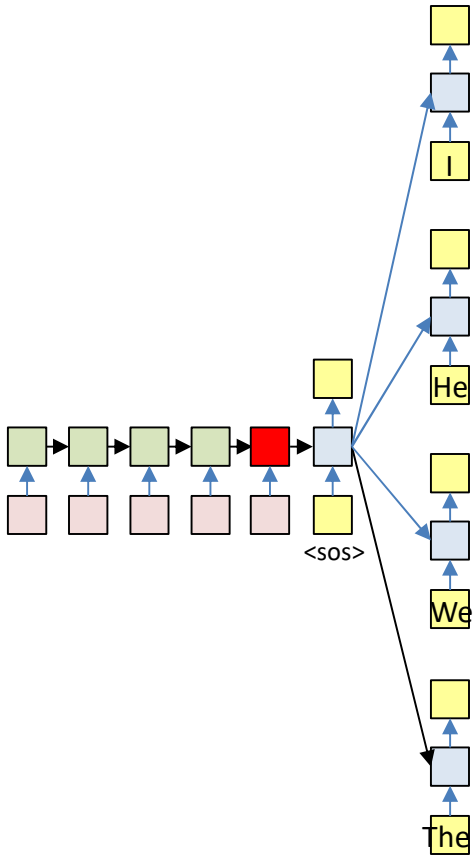
What should we have chosen at t=1??

Choose “the” or “he”?



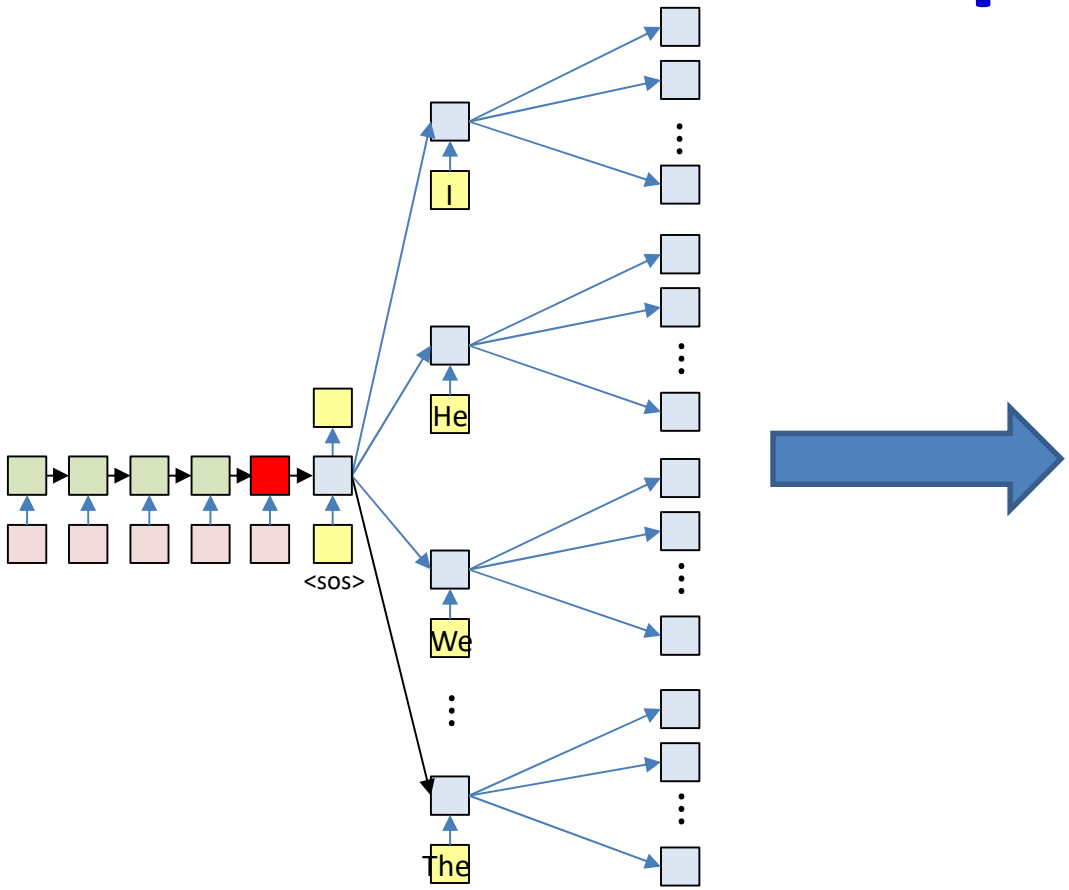
- Problem: making a poor choice at any time commits us to a poor future
  - But we cannot know at that time the choice was poor
- Solution: Don’t choose..

Optimal Solution: Multiple choices



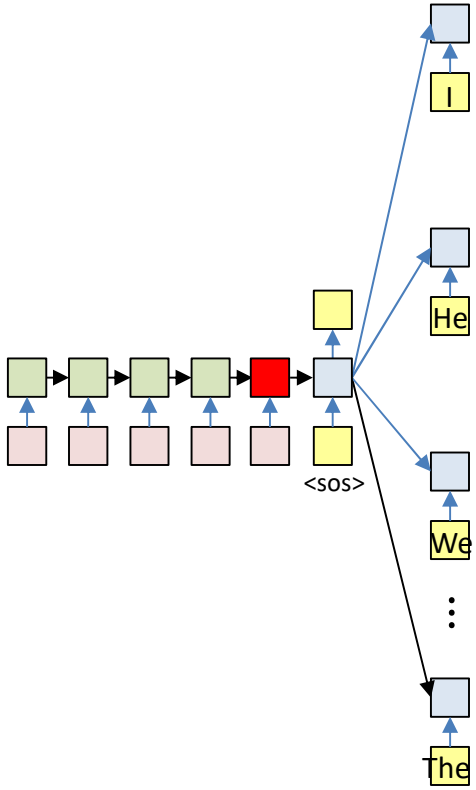
- Retain all choices and *fork* the network
  - With every possible word as input

Problem: Multiple choices



- **Problem**: This will blow up very quickly
  - For an output vocabulary of size V, after T output steps we'd have forked out V^T branches

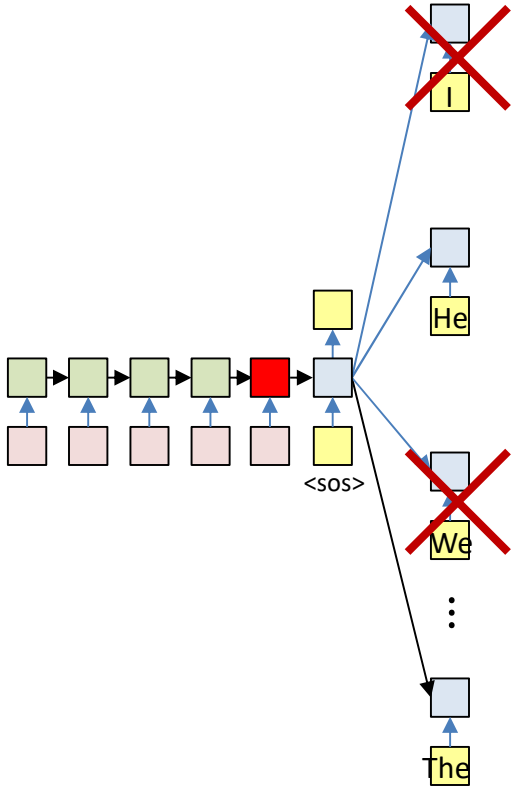
Solution: Prune



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

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

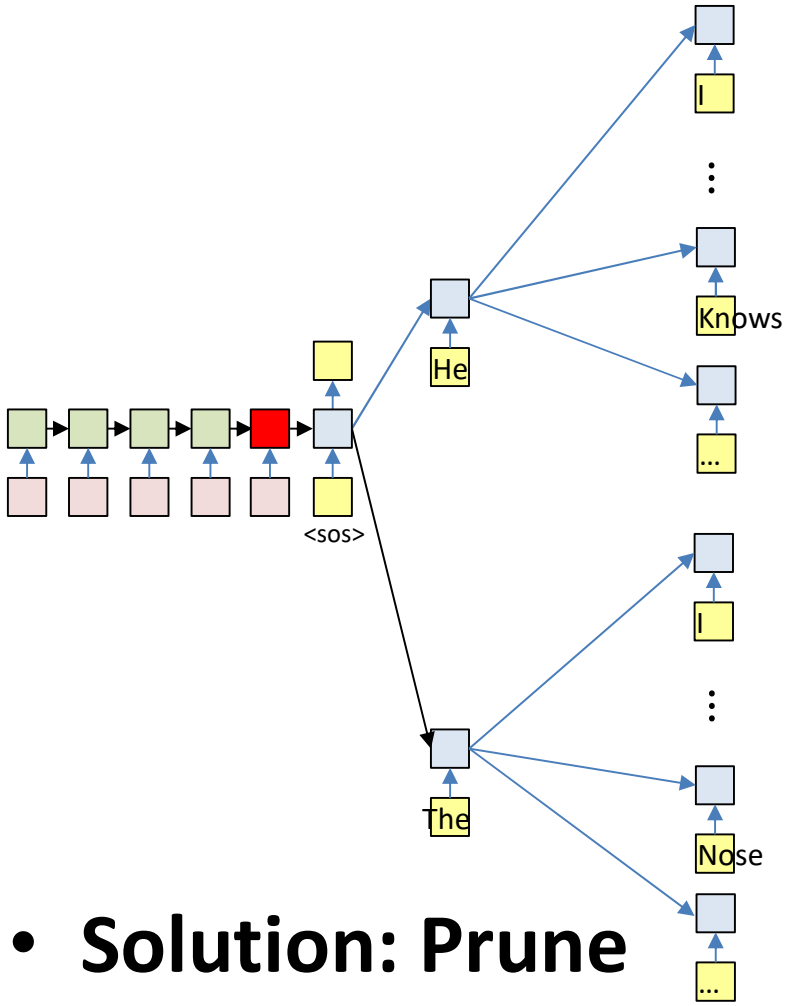
Solution: Prune



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

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

Solution: Prune



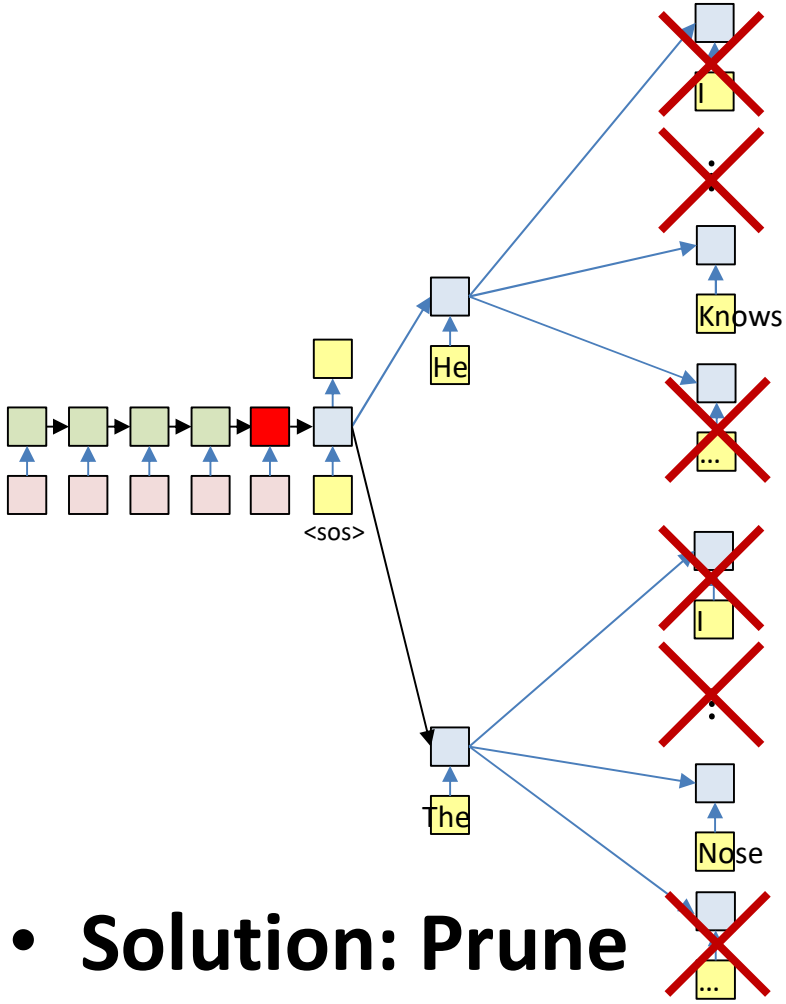
Note: based on product

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

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

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

Solution: Prune



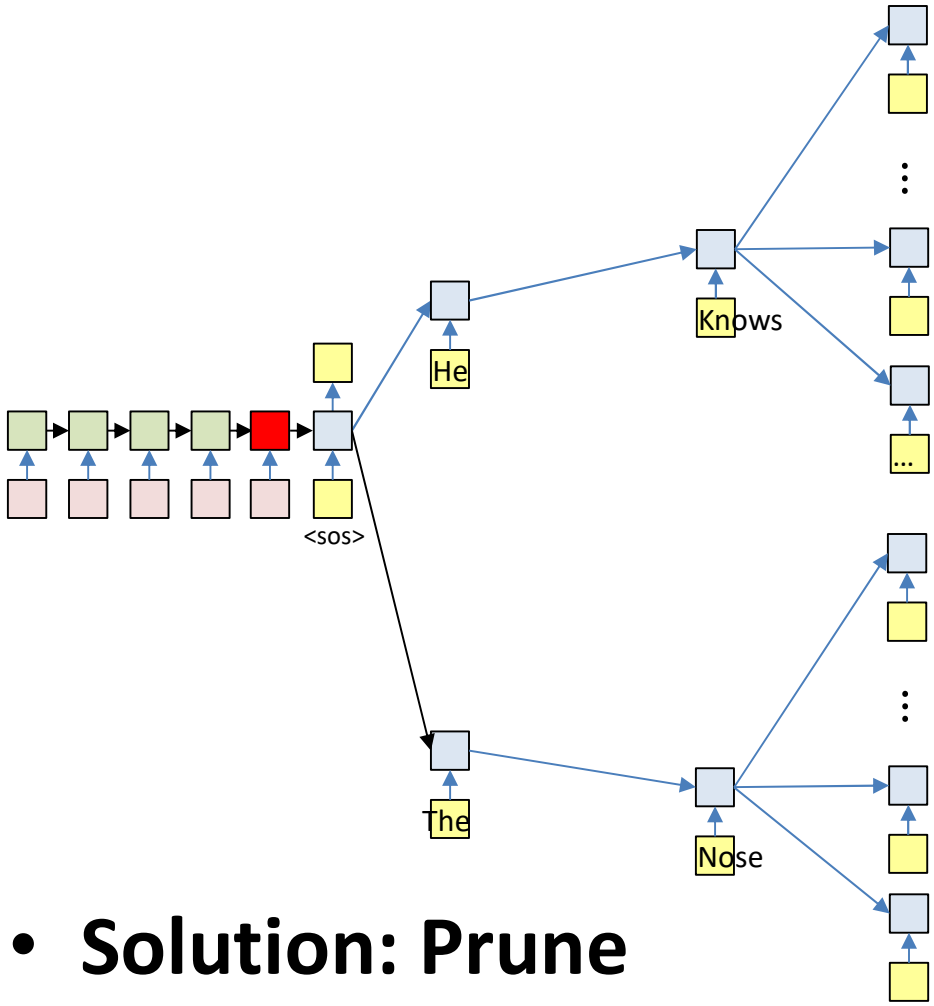
Note: based on product

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

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

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

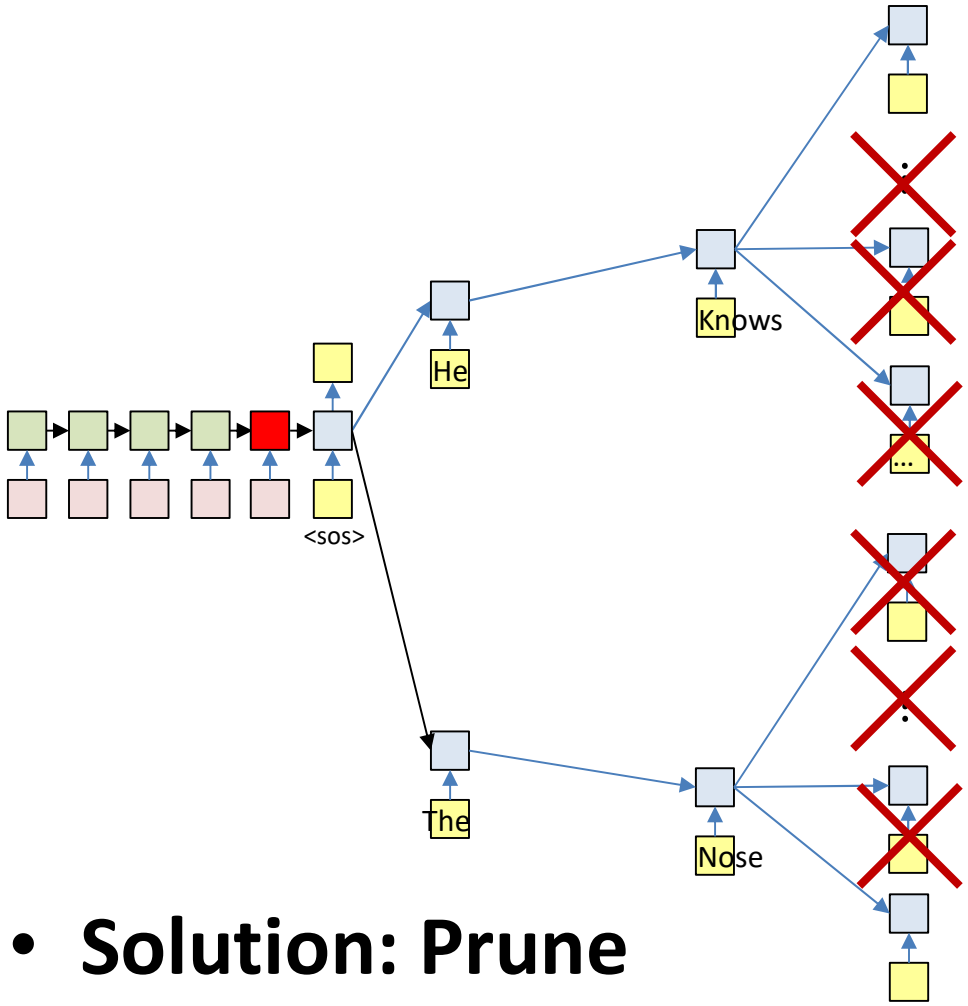
Solution: Prune



$$= Top_K P(O_3|O_1, O_2, I_1, \ldots, I_N) \times P(O_2|O_1, I_1, \ldots, I_N) \times P(O_1|I_1, \ldots, I_N)$$

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

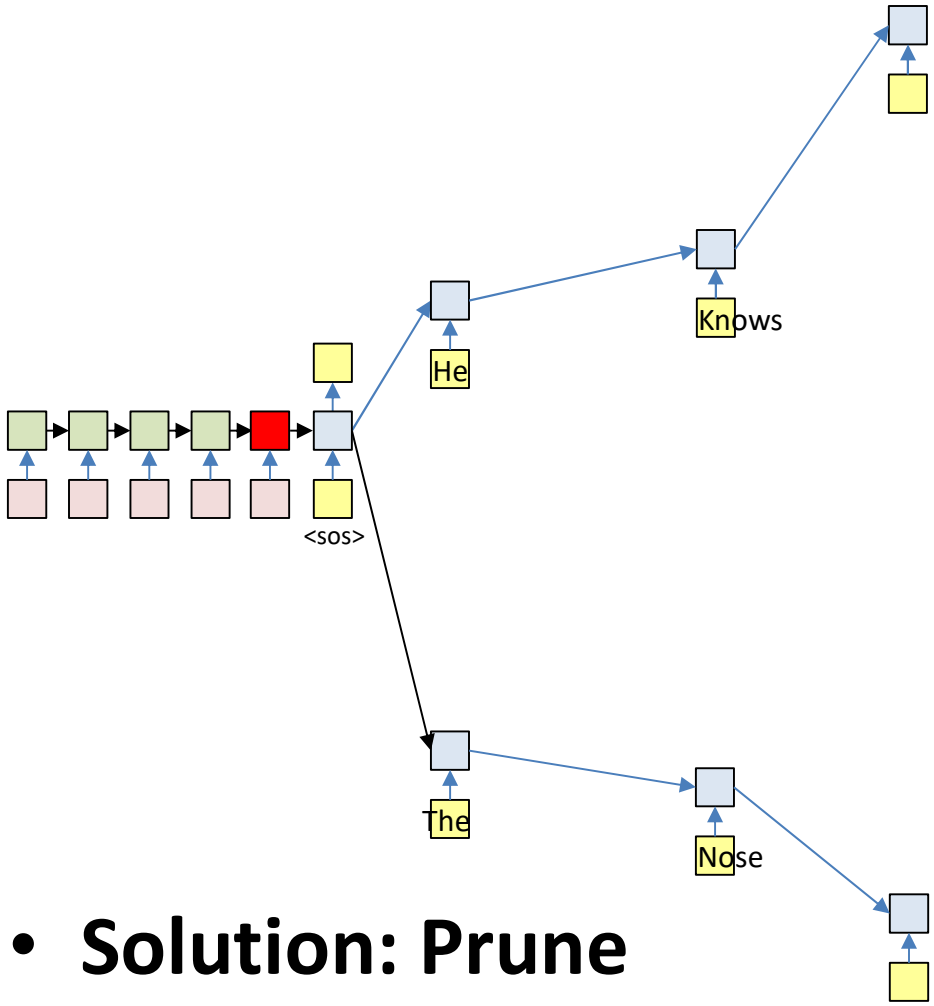
Solution: Prune



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

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

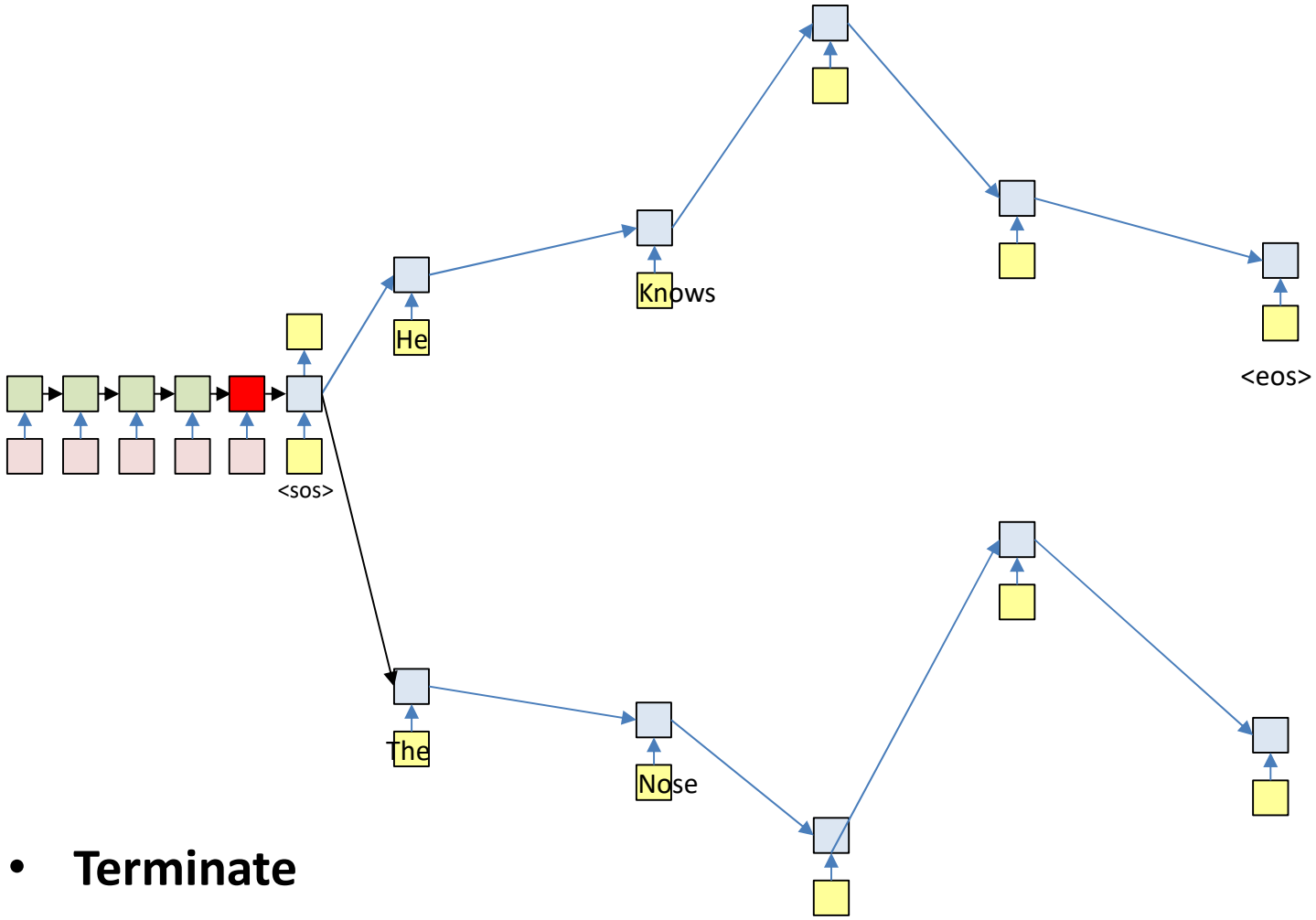
Solution: Prune



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

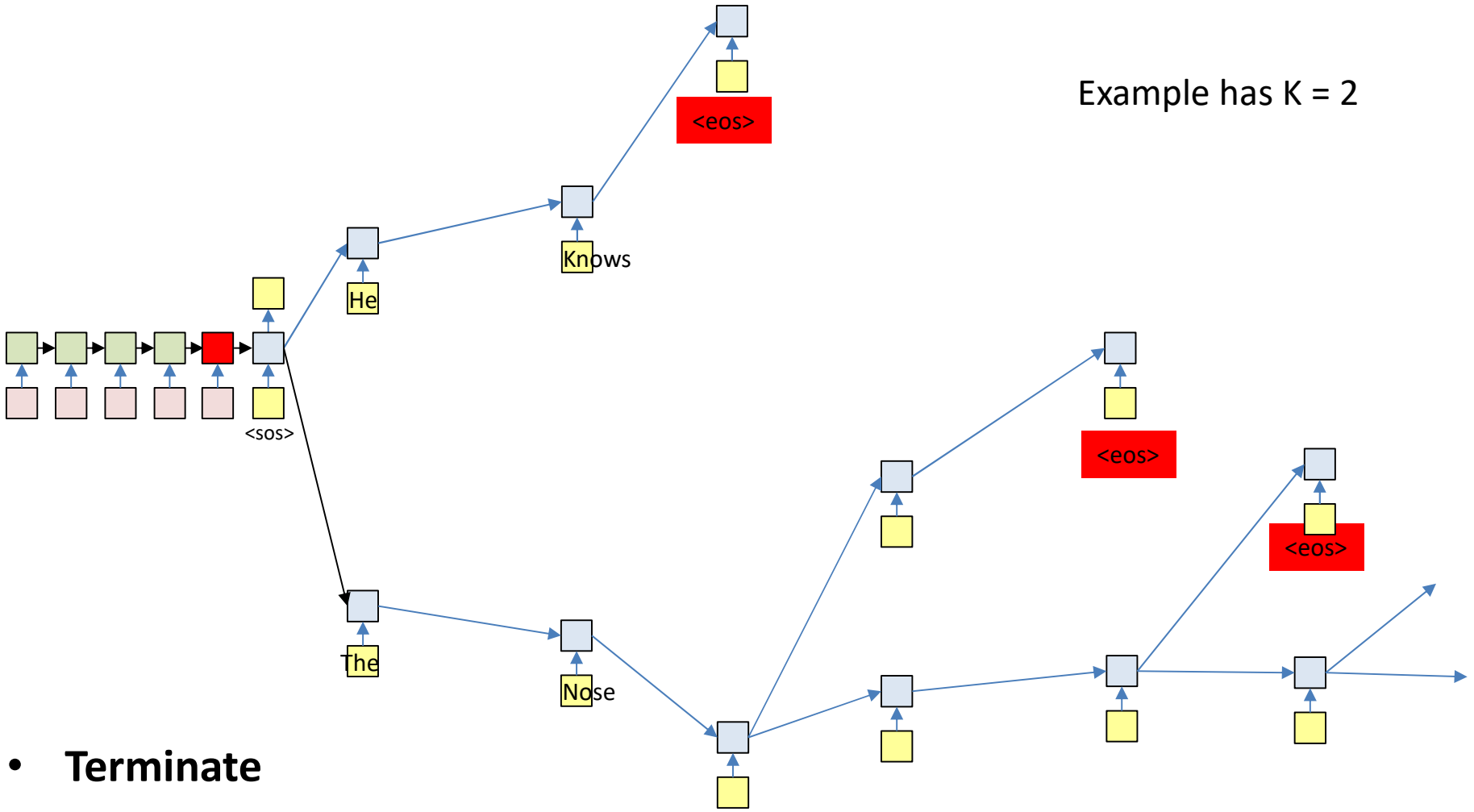
- **Solution: Prune**
  - At each time, retain only the top K scoring forks

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

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

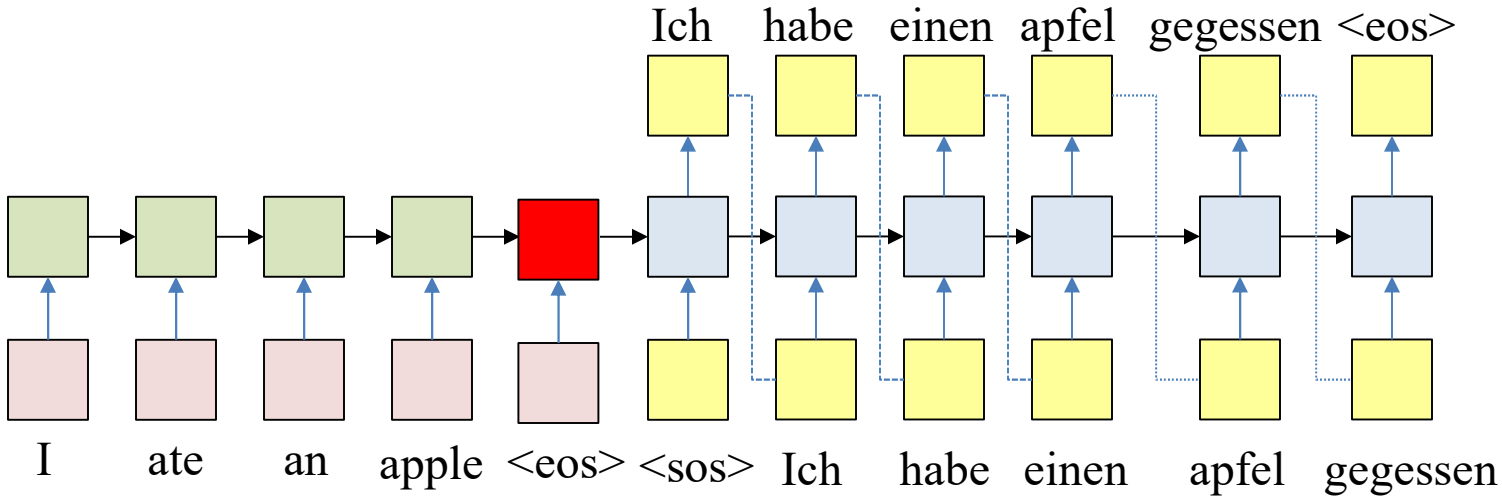
Pseudocode: Beam search

```
# Assuming encoder output H is available
path = <sos>
beam = {path}
pathscore = [path] = 1
state[path] = h[0]  # Output of encoder
do  # Step forward
    nextbeam = {}
    nextpathscore = []
    nextstate = {}
    for path in beam:
        cfin = path[end]
        hpath = state[path]
        [y,h] = RNN_output_step(hpath,cfin)
        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,bw)
until bestpath[end] = <eos>
```

Pseudocode: Prune

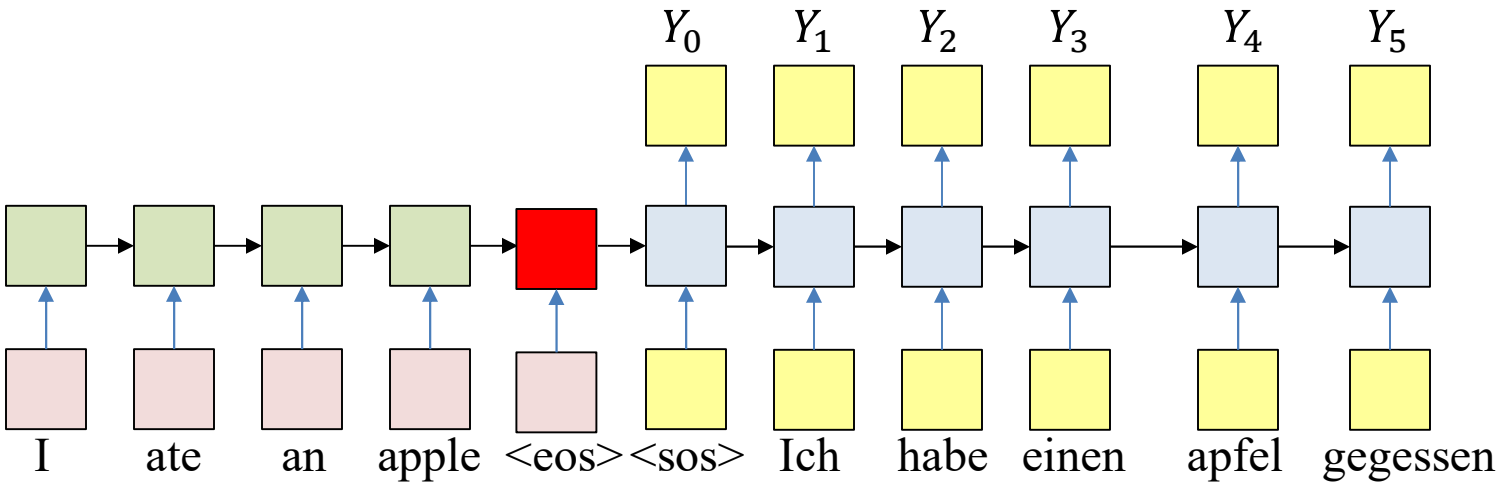
```
# Note, there are smarter ways to implement this

function prune (state, score, beam, beamwidth)
    sortedscore = sort(score)
    threshold = sortedscore[beamwidth]

    prunedstate = {}
    prunedscore = []
    prunedbeam = {}

    bestscore = -inf
    bestpath = none
    for path in beam:
        if score[path] > threshold:
            prunedbeam += path # set addition
            prunedstate[path] = state[path]
            prunedscore[path] = score[path]
            if score[path] > bestscore
                bestscore = score[path]
                bestpath = path
            end
        end
    end
    return prunedbeam, prunedscore, prunedstate, bestpath
```

Training the system



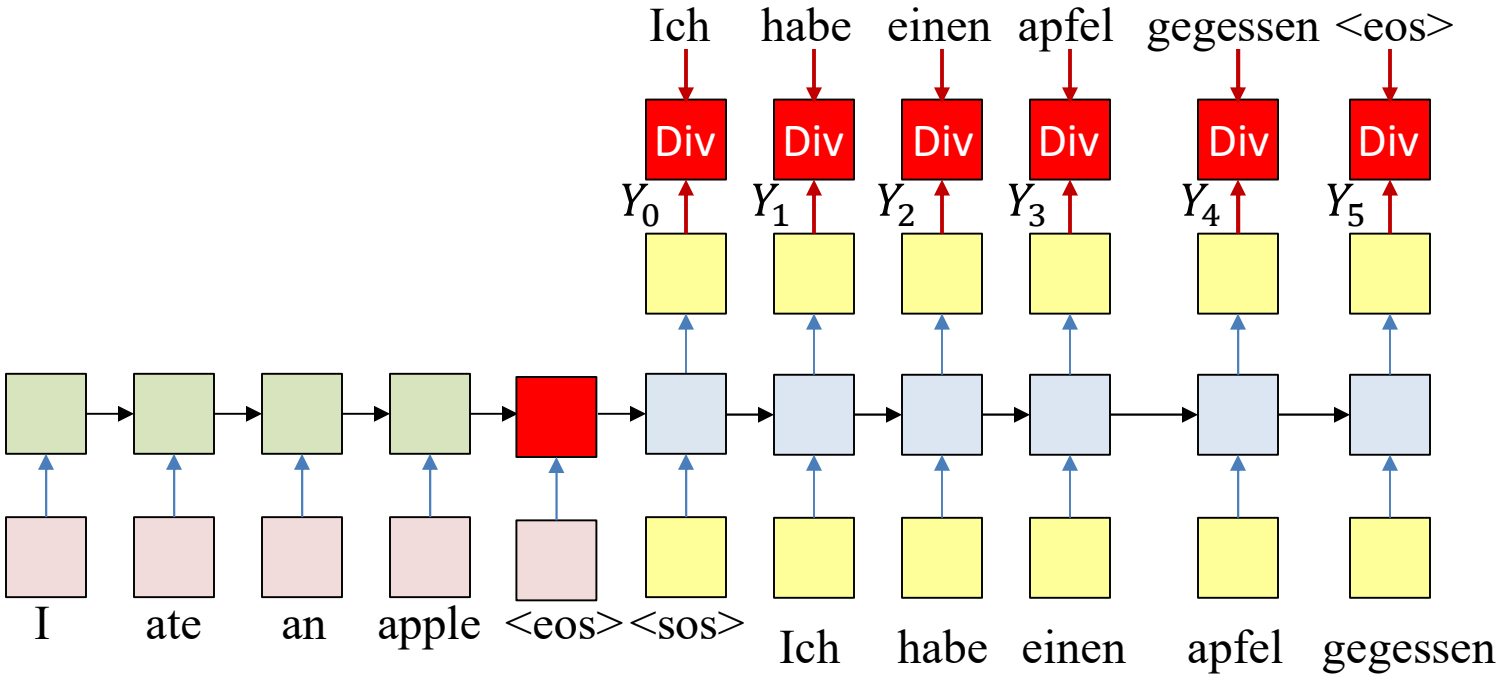
- Must learn to make predictions appropriately
  - Given “I ate an apple <eos>”, produce “Ich habe einen apfel gegessen <eos>”.

Training : Forward pass



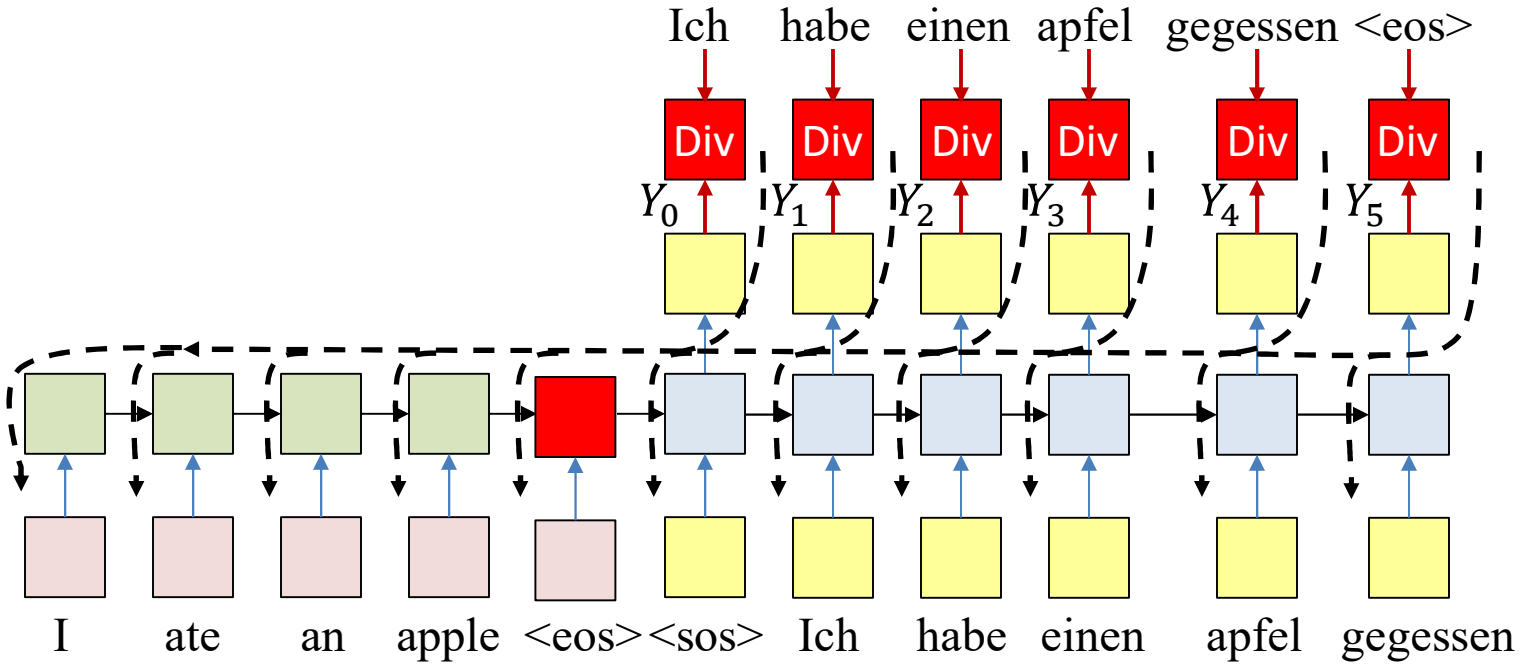
- Forward pass: Input the source and target sequences, sequentially
  - Output will be a probability distribution over target symbol set (vocabulary)

Training : Backward pass



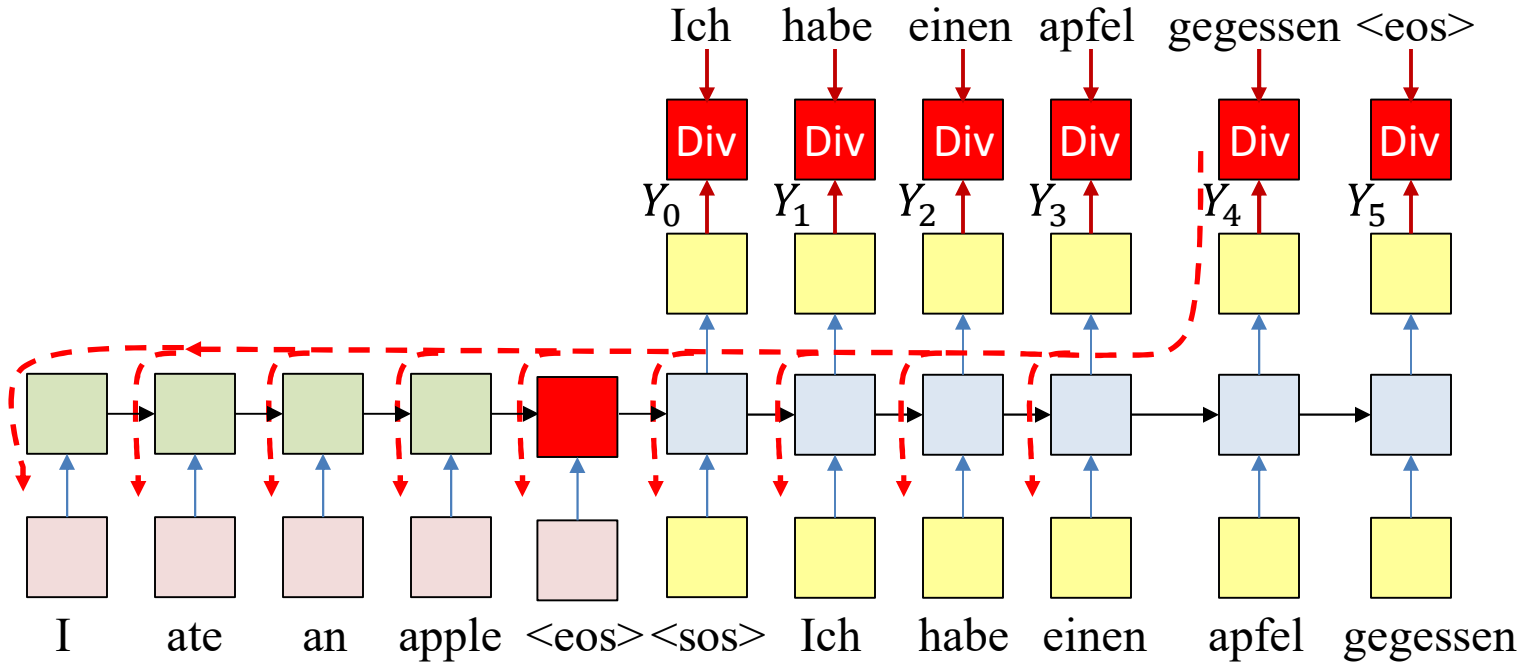
- Backward pass: Compute the divergence between the output distribution and target word sequence

Training : Backward pass



- Backward pass: Compute the divergence between the output distribution and target word sequence
- Backpropagate the derivatives of the divergence through the network to learn the net

Training : Backward pass

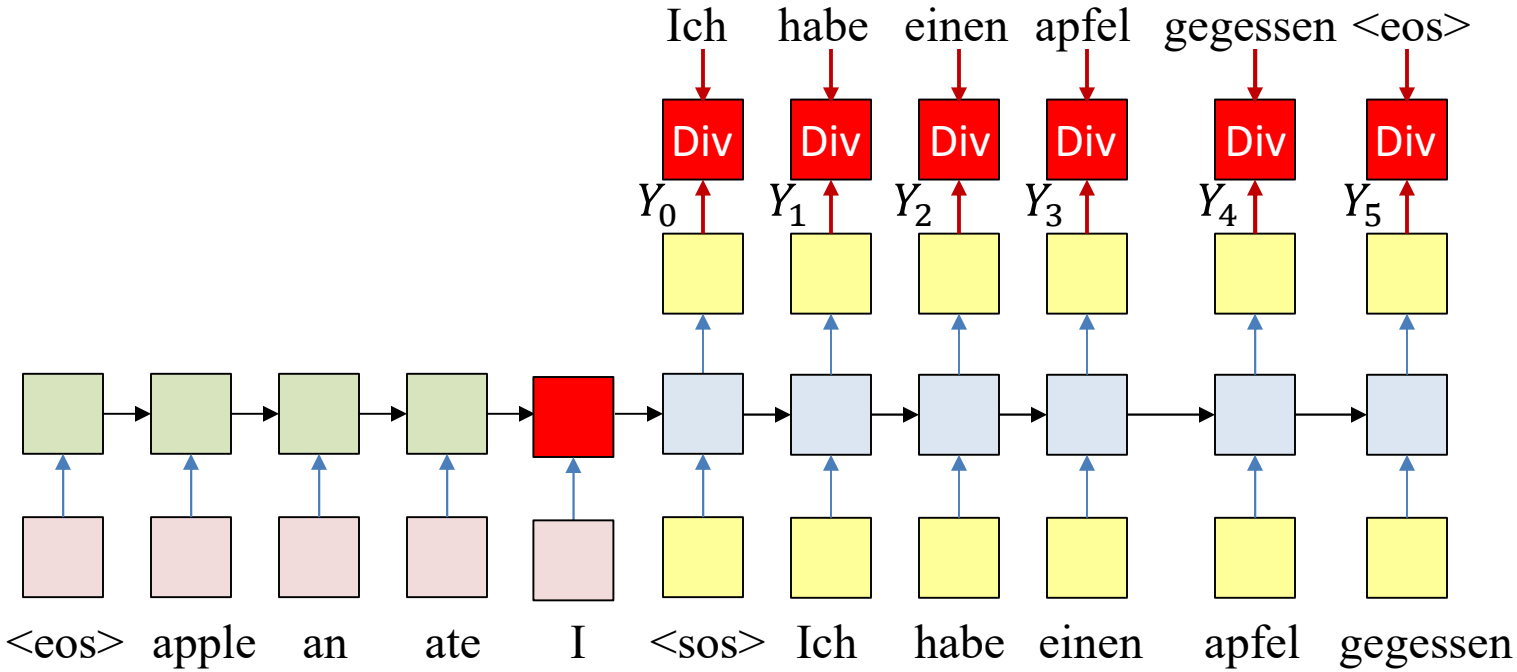


- In practice, if we apply SGD, we may randomly sample words from the output to actually use for the backprop and update
  - Typical usage: Randomly select one word from each input training instance (comprising an input-output pair)
    - For each iteration
      - Randomly select training instance: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

Overall training

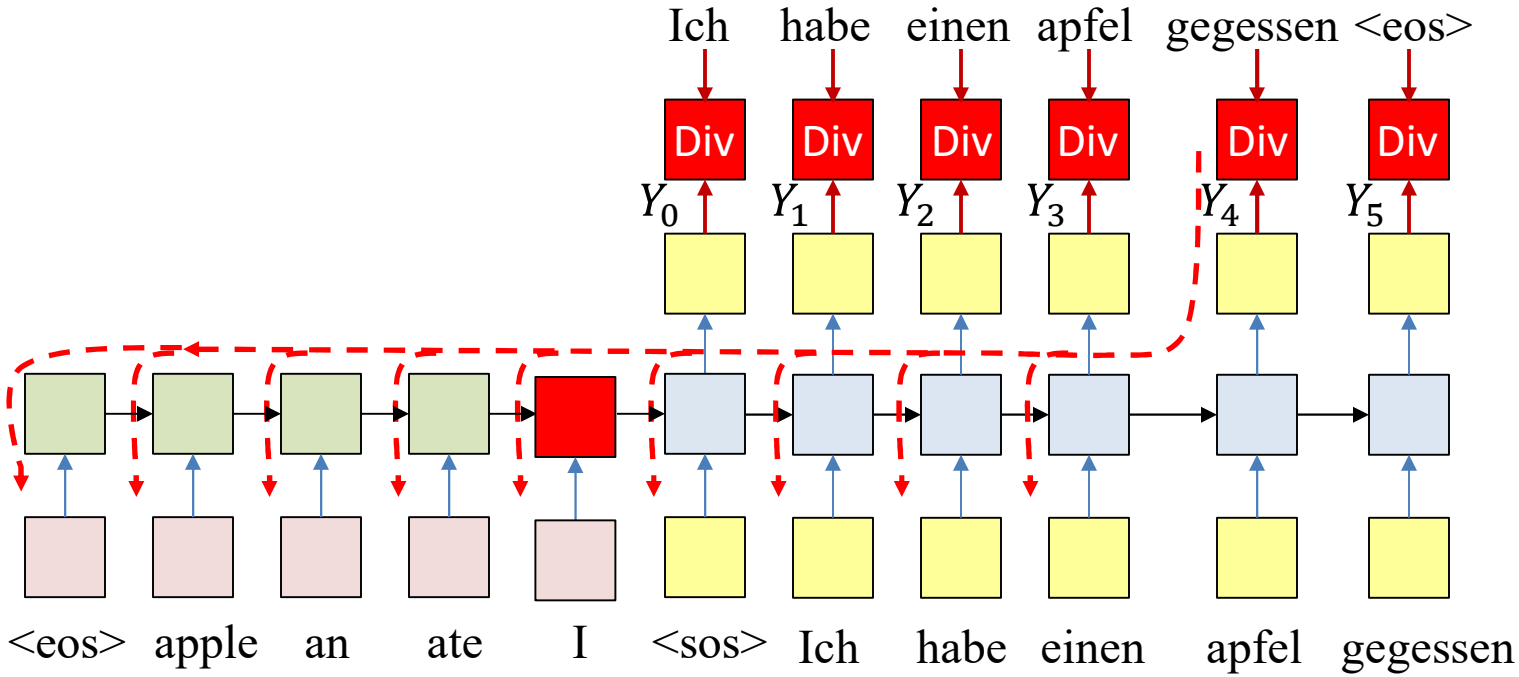
- Given several training instance $(\mathbf{X}, \mathbf{D})$
- For each training instance
  - Forward pass: Compute the output of the network for $(\mathbf{X}, \mathbf{D})$
    - Note, both $\mathbf{X}$ *and* $\mathbf{D}$ are used in the forward pass
  - Backward pass: Compute the divergence between selected words of the desired target $\mathbf{D}$ and the actual output $\mathbf{Y}$
    - Propagate derivatives of divergence for updates
- Update parameters

Trick of the trade: Reversing the input



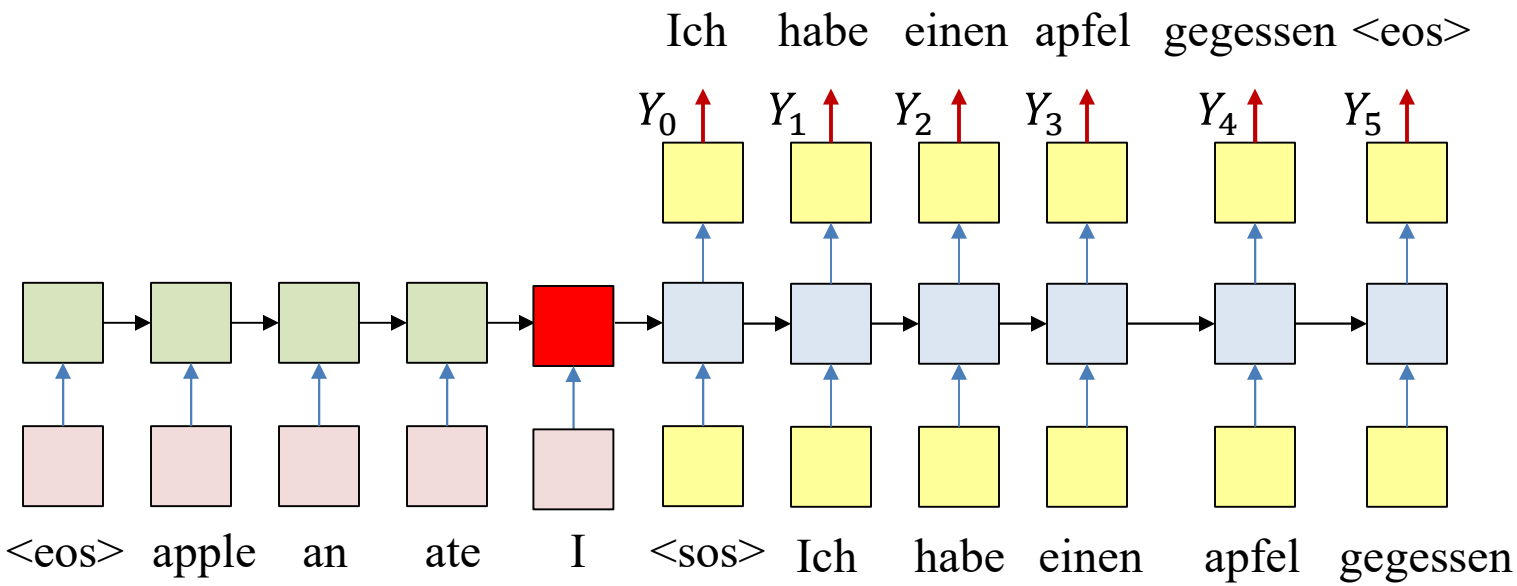
- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way

Trick of the trade: Reversing the input



- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way

Trick of the trade: Reversing the input



- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way
- *This happens both for training and during inference on test data*

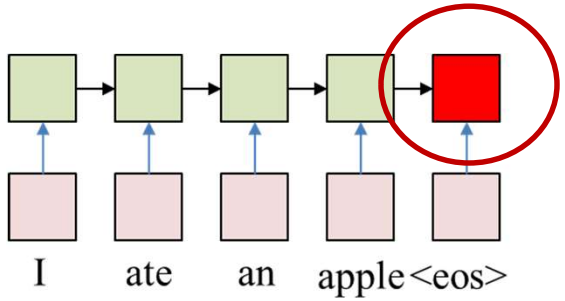
Overall training

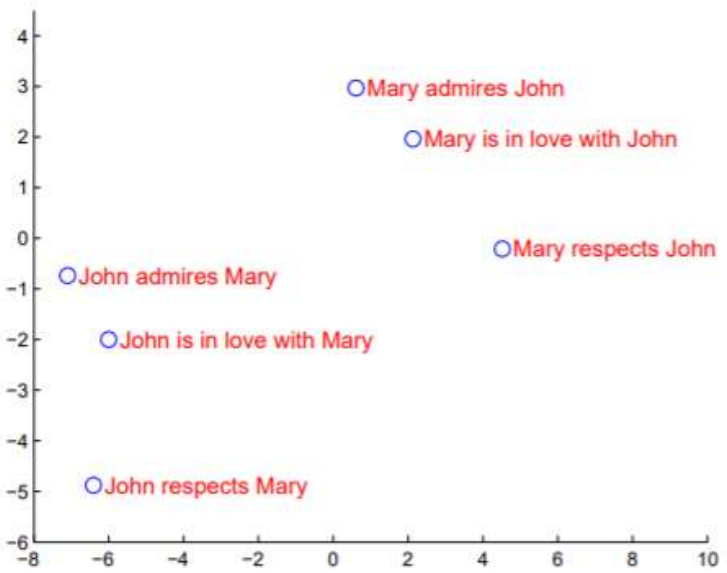
- Given several training instance $(\mathbf{X}, \mathbf{D})$
- Forward pass: Compute the output of the network for $(\mathbf{X}, \mathbf{D})$ *with input in reverse order*
  - Note, both $\mathbf{X}$ *and* $\mathbf{D}$ are used in the forward pass
- Backward pass: Compute the divergence between the desired target $\mathbf{D}$ and the actual output $\mathbf{Y}$
  - Propagate derivatives of divergence for updates

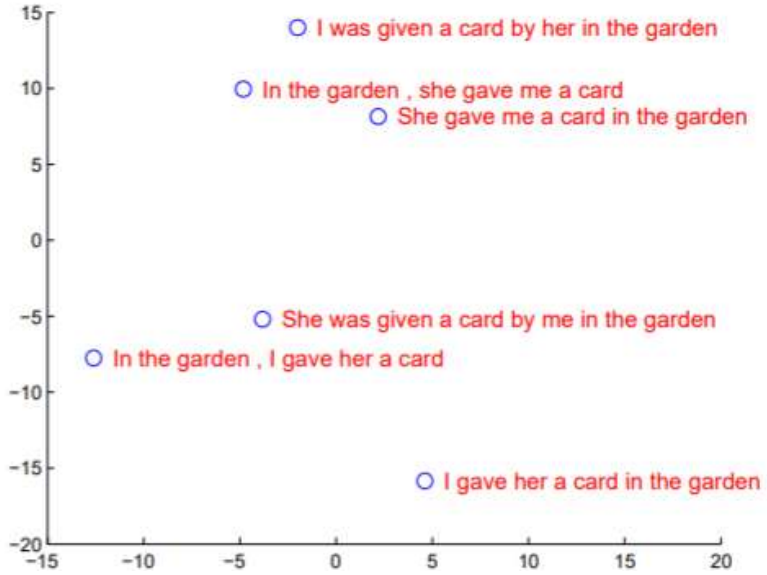
Applications

- Machine Translation
  - My name is Tom → Ich heisse Tom/Mein name ist Tom
- Automatic speech recognition
  - Speech recording → “My name is Tom”
- Dialog
  - “I have a problem” → “How may I help you”
- Image to text
  - Picture → Caption for picture

Machine Translation Example







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

Machine Translation Example

Type	Sentence
Our model	Ulrich UNK , membre du conseil d' administration du constructeur automobile Audi , affirme qu' il s' agit d' une pratique courante depuis des années pour que les téléphones portables puissent être collectés avant les réunions du conseil d' administration afin qu' ils ne soient pas utilisés comme appareils d' écoute à distance .
Truth	Ulrich Hackenberg , membre du conseil d' administration du constructeur automobile Audi , déclare que la collecte des téléphones portables avant les réunions du conseil , afin qu' ils ne puissent pas être utilisés comme appareils d' écoute à distance , est une pratique courante depuis des années .
Our model	" Les téléphones cellulaires , qui sont vraiment une question , non seulement parce qu' ils pourraient potentiellement causer des interférences avec les appareils de navigation , mais nous savons , selon la FCC , qu' ils pourraient interférer avec les tours de téléphone cellulaire lorsqu' ils sont dans l' air " , dit UNK .
Truth	" Les téléphones portables sont véritablement un problème , non seulement parce qu' ils pourraient éventuellement créer des interférences avec les instruments de navigation , mais parce que nous savons , d' après la FCC , qu' ils pourraient perturber les antennes-relais de téléphonie mobile s' ils sont utilisés à bord " , a déclaré Rosenker .
Our model	Avec la crémation , il y a un " sentiment de violence contre le corps d' un être cher " , qui sera " réduit à une pile de cendres " en très peu de temps au lieu d' un processus de décomposition " qui accompagnera les étapes du deuil " .
Truth	Il y a , avec la crémation , " une violence faite au corps aimé " , qui va être " réduit à un tas de cendres " en très peu de temps , et non après un processus de décomposition , qui " accompagnerait les phases du deuil " .

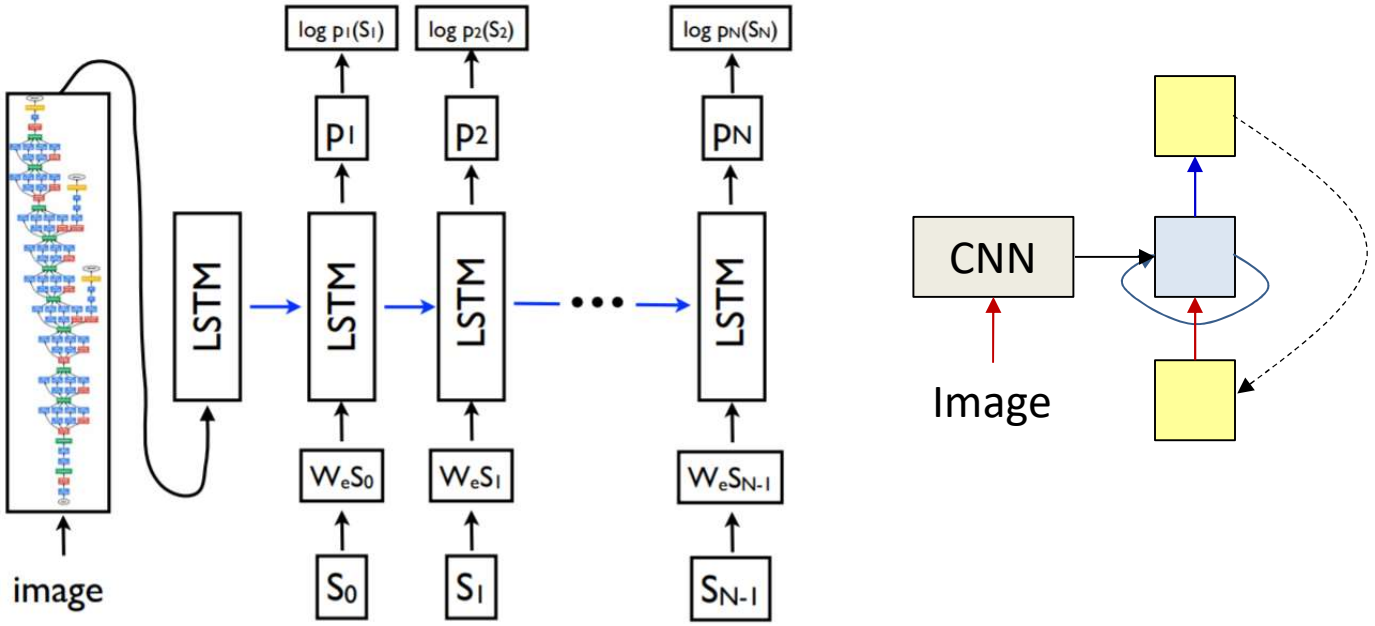
- Examples of translation
  - From "Sequence-to-sequence learning with neural networks", Sutskever, Vinyals and Le

Human Machine Conversation: Example

Machine: *what is the error that you are running please*
Human: *i am seeing an error related to vpn*
Machine: *what is the error message that you are getting when connecting to vpn using network connect ?*
Human: *connection refused or something like that*
Machine: *may i know the version of network connect you connect ?*
Human: *i am not sure i know that*
Machine: *is the network connect prompting for any user certificate*

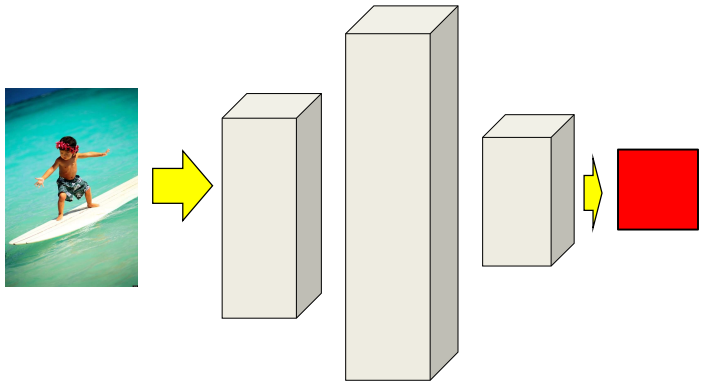
- From "A neural conversational model", Orin Vinyals and Quoc Le
- Trained on human-human converstations
- Task: Human text in, machine response out

Generating Image Captions



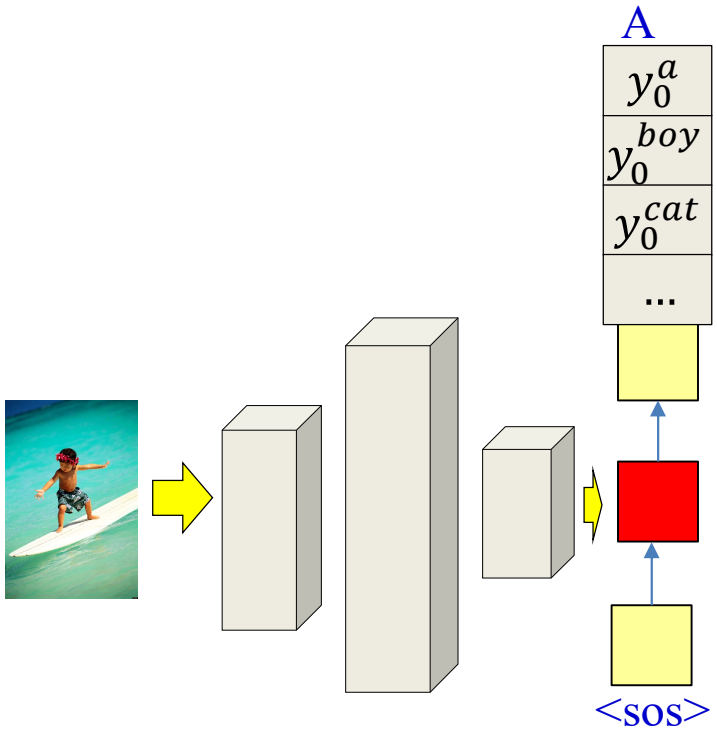
- Not really a seq-to-seq problem, more an image-to-sequence problem
- Initial state is produced by a state-of-art CNN-based image classification system
  - Subsequent model is just the decoder end of a seq-to-seq model
    - "Show and Tell: A Neural Image Caption Generator", O. Vinyals, A. Toshev, S. Bengio, D. Erhan

Generating Image Captions



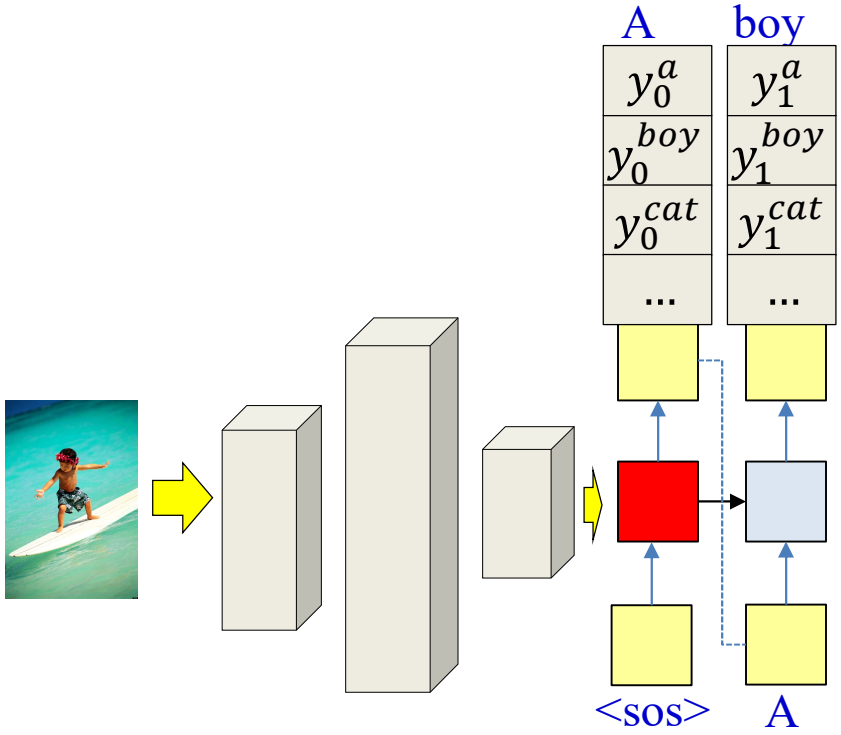
- Decoding: Given image
  - Process it with CNN to get output of classification layer

Generating Image Captions



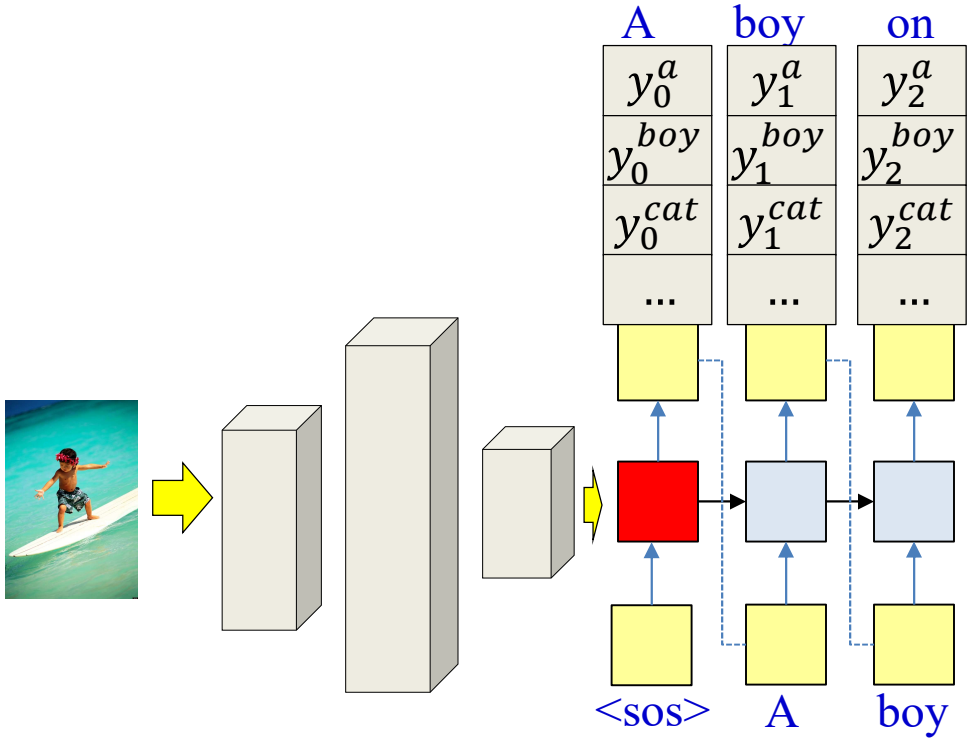
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



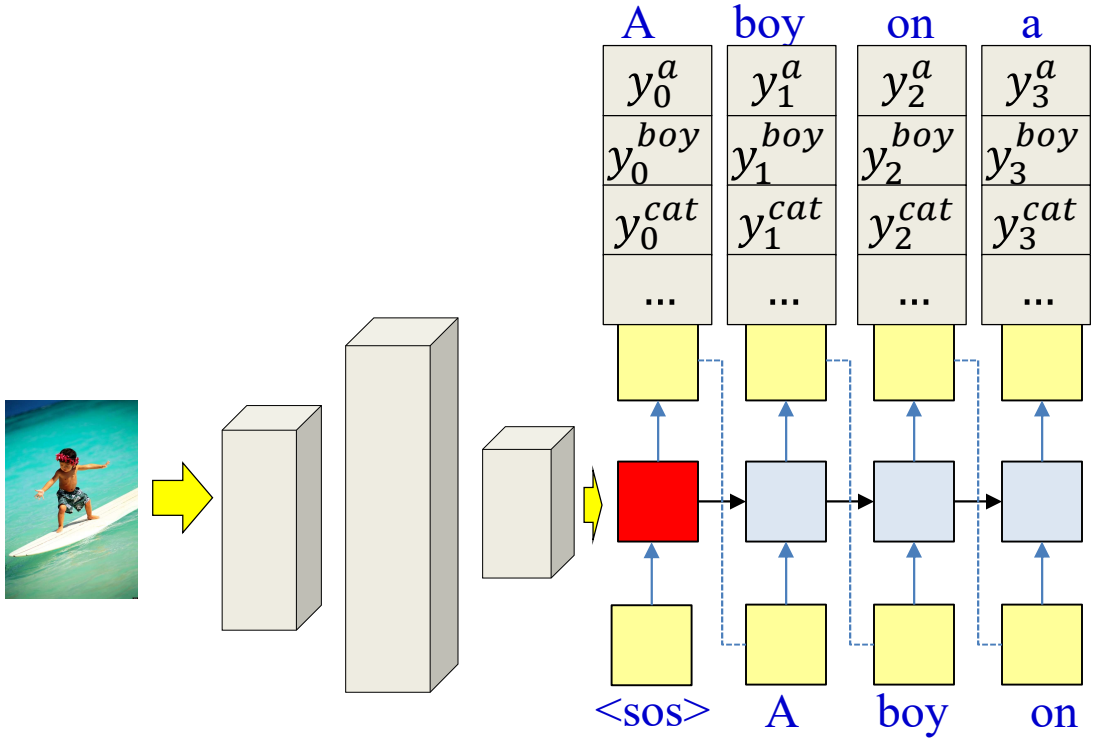
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



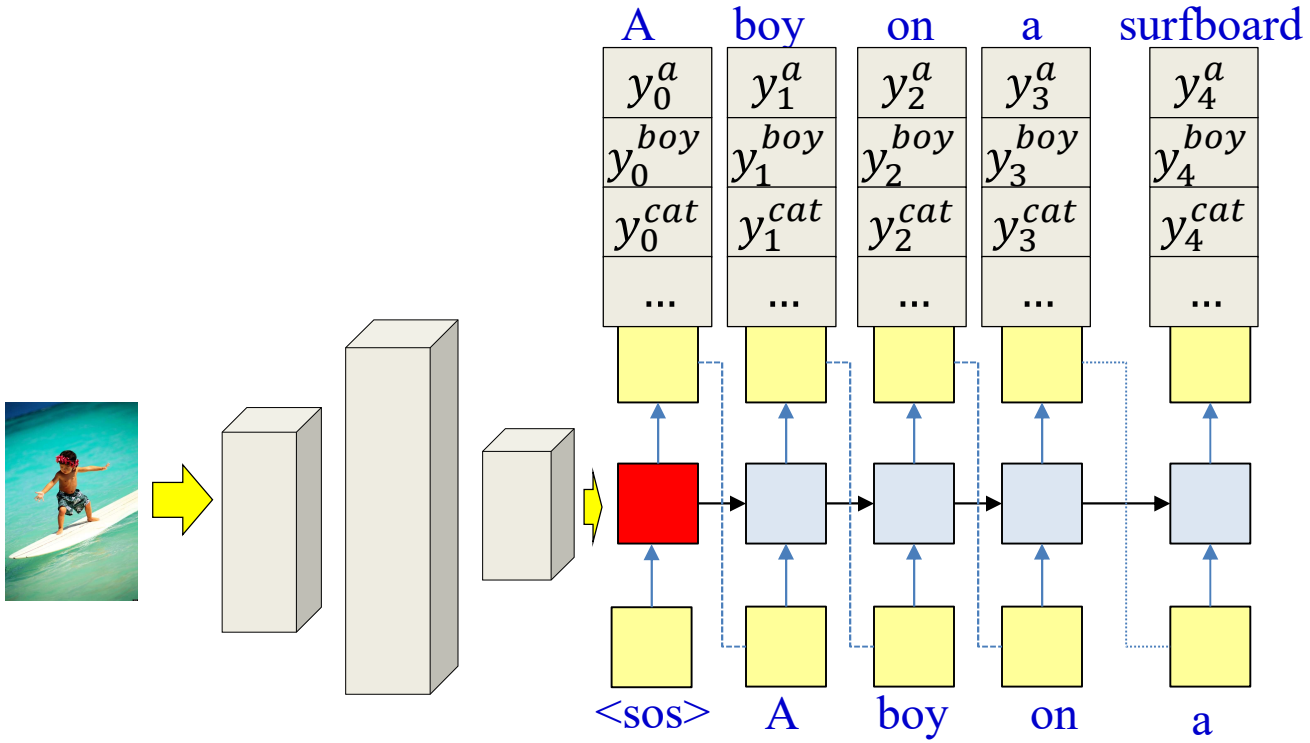
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



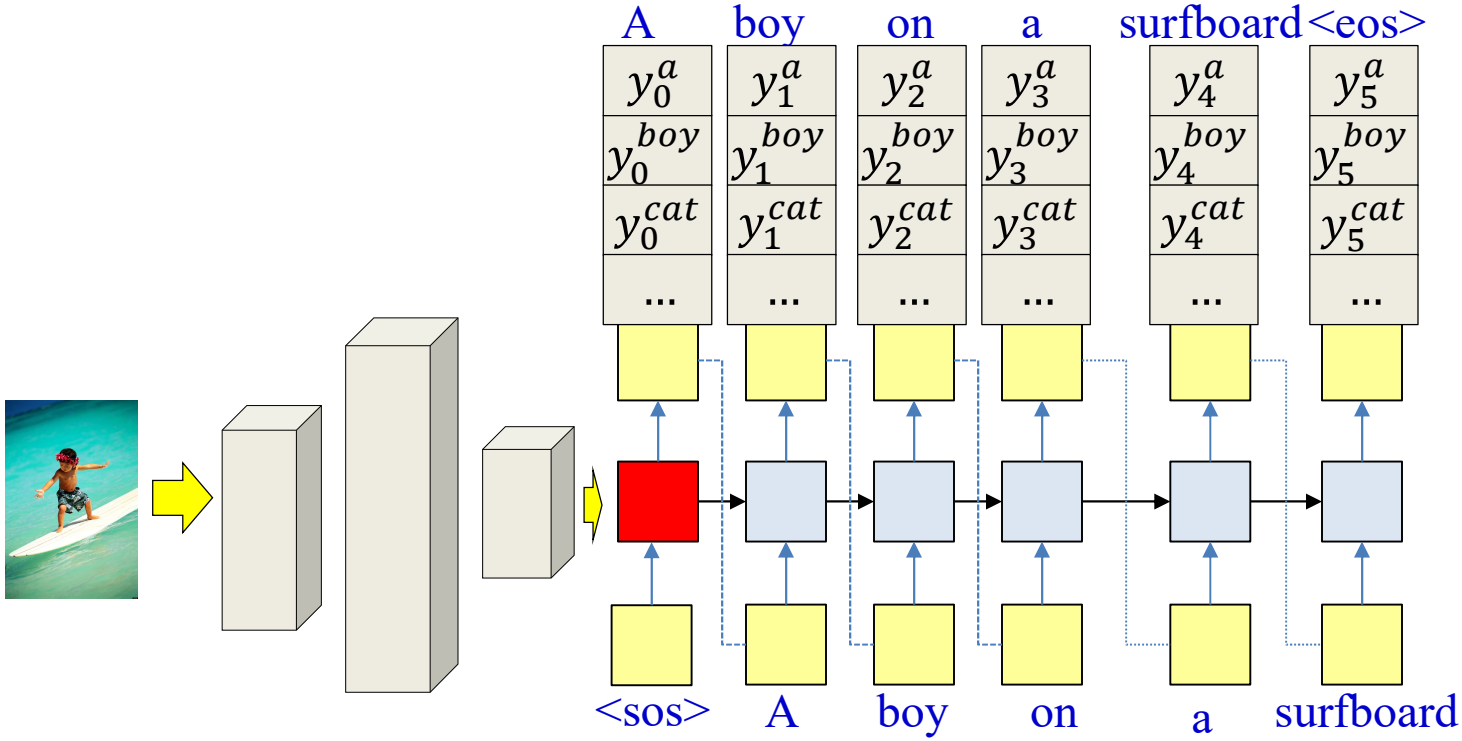
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



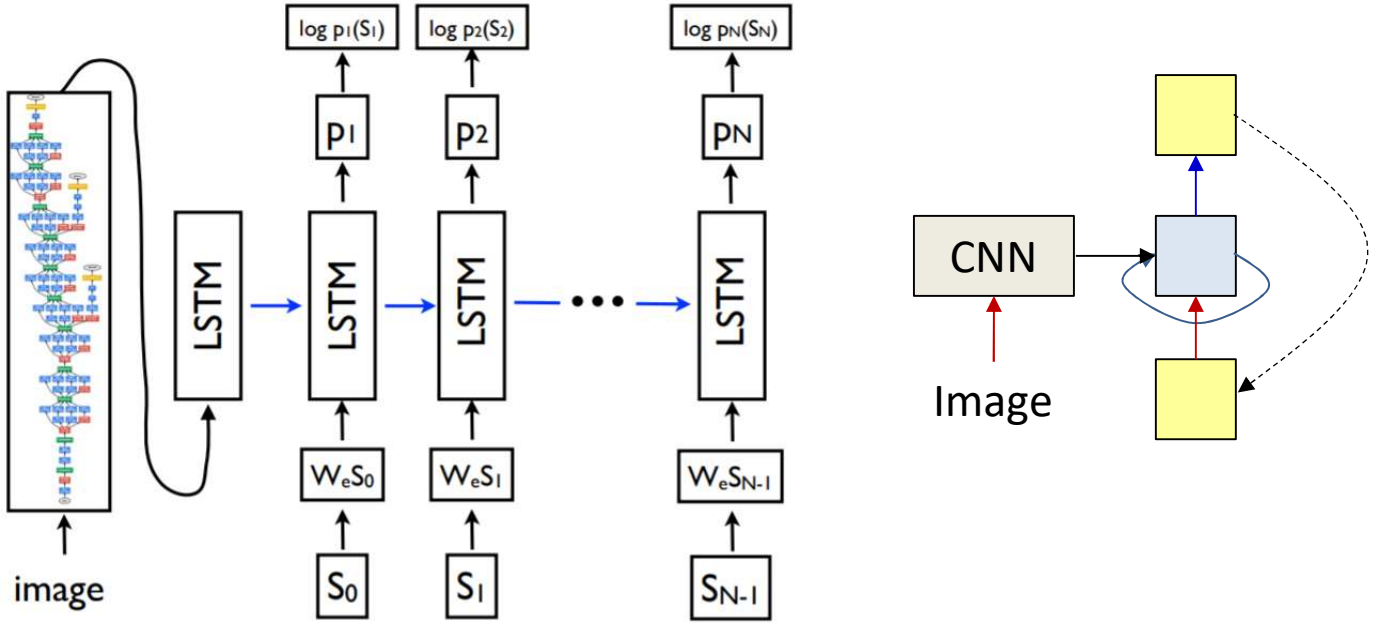
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions

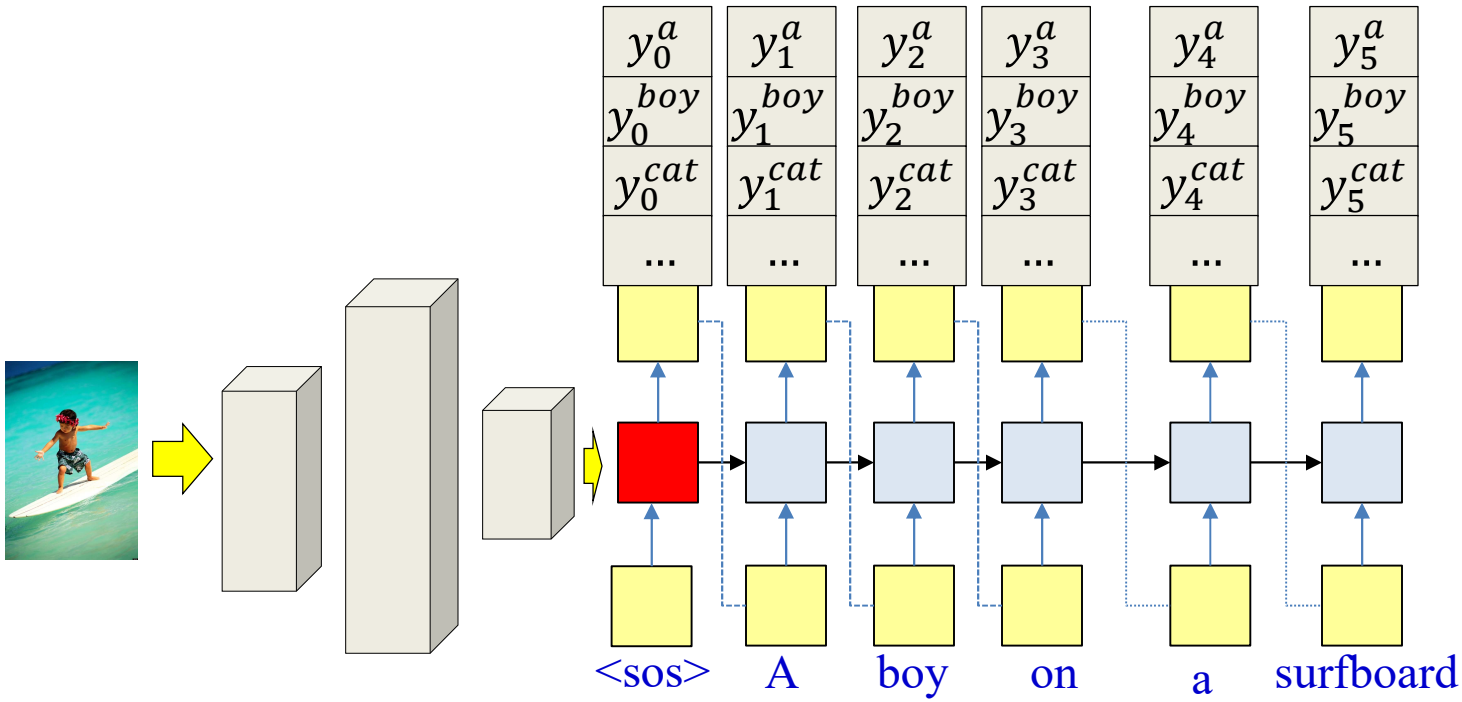


- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

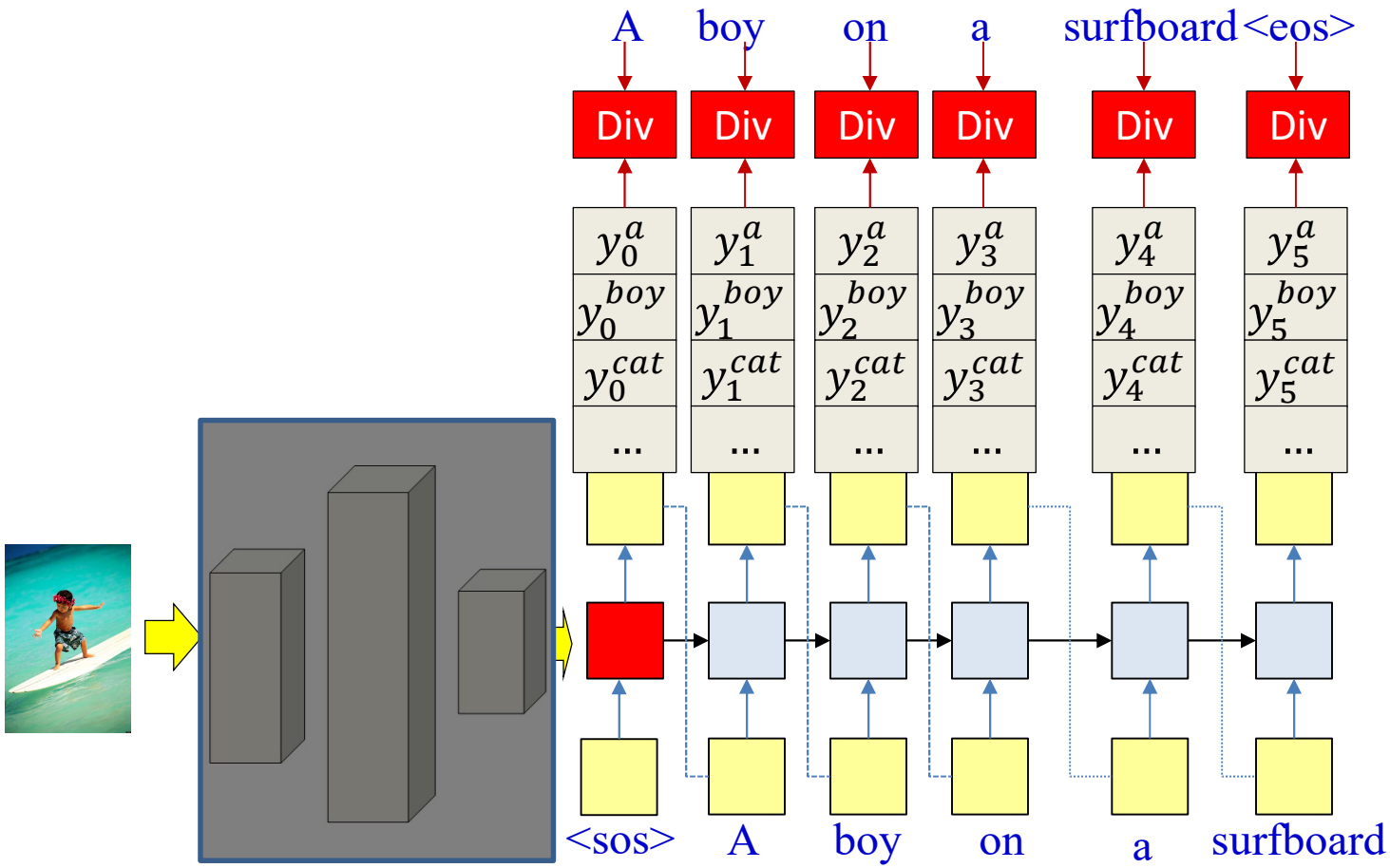
Training



- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net



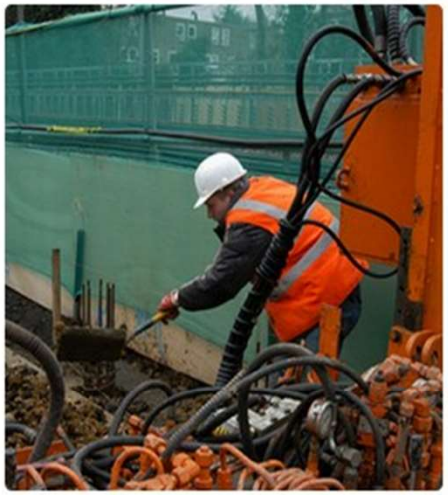
- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net
- **Forward pass:** Produce output distributions given the image and caption



- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net
- **Forward pass:** Produce output distributions given the image and caption
- **Backward pass:** Compute the divergence w.r.t. training caption, and backpropagate derivatives
  - All components of the network, including final classification layer of the image classification net are updated
  - The CNN portions of the image classifier are not modified (transfer learning)

Examples from Vinyals et al.

"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

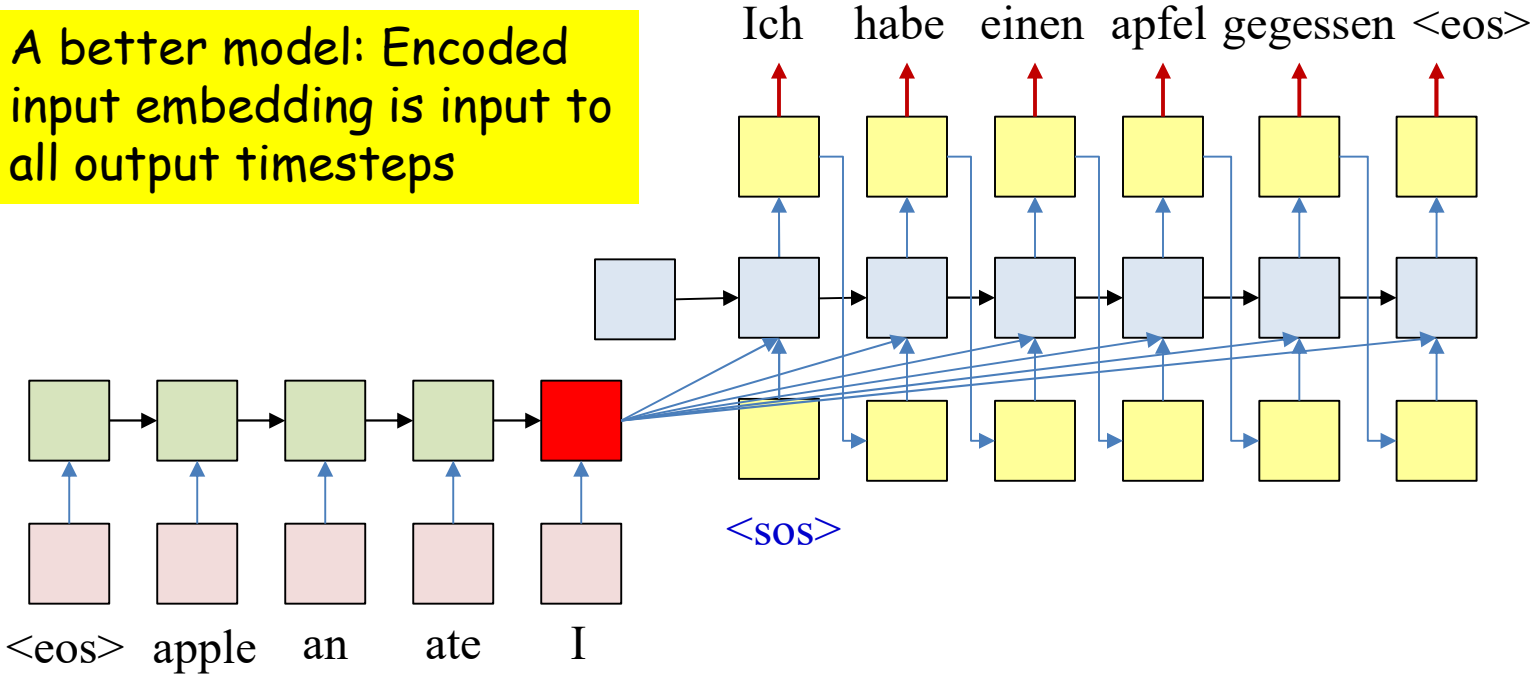
"a cat is sitting on a couch with a remote control."

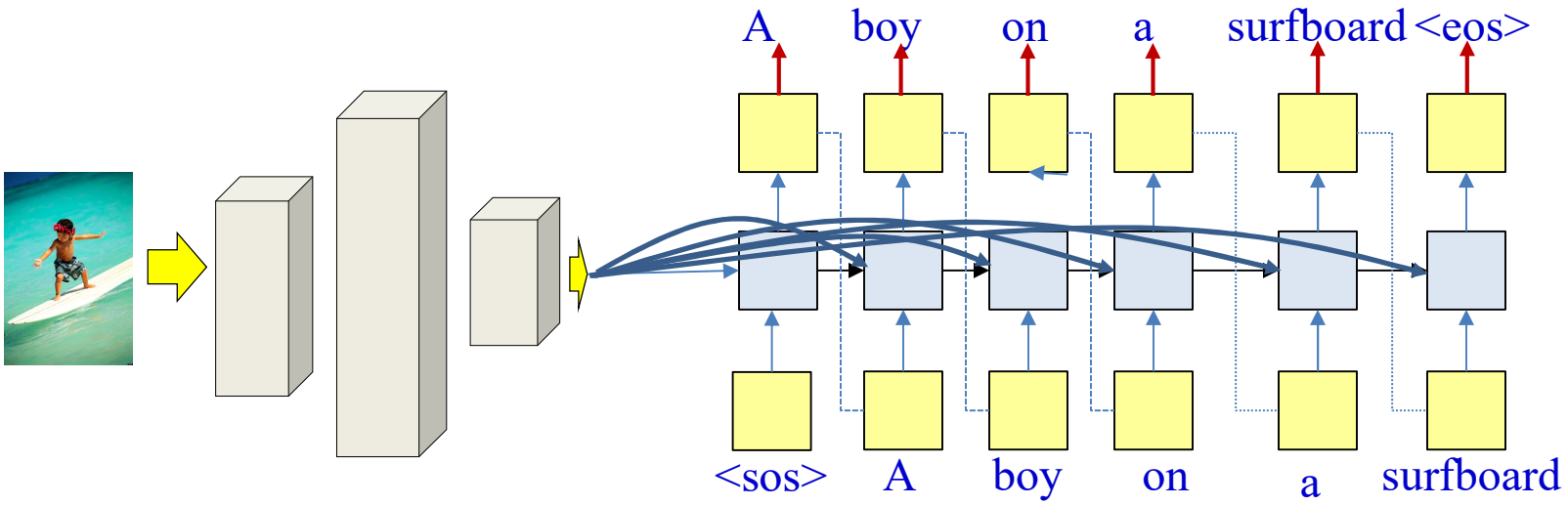
"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

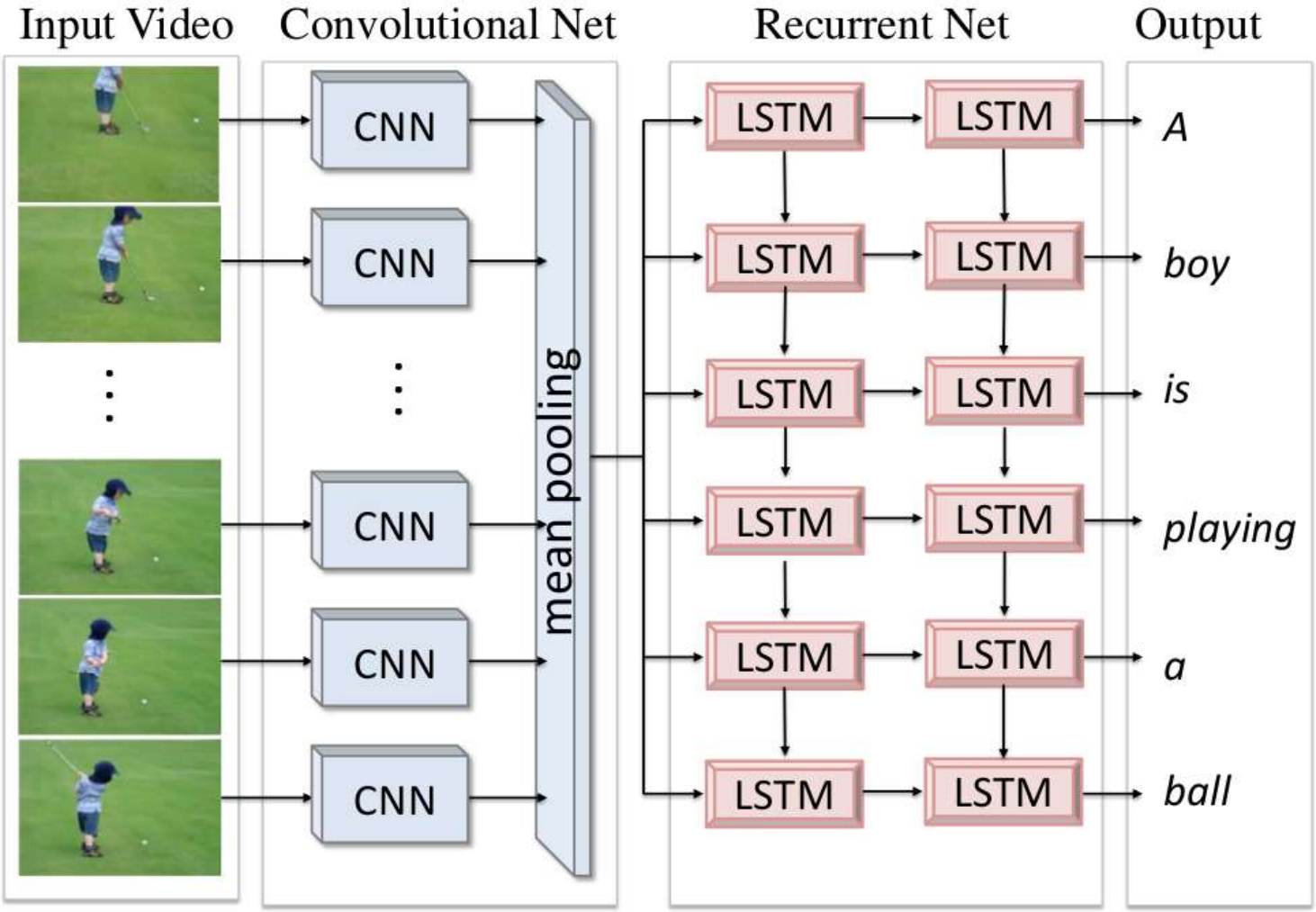
Variants

A better model: Encoded input embedding is input to all output timesteps





Translating Videos to Natural Language Using Deep Recurrent Neural Networks



Translating Videos to Natural Language Using Deep Recurrent Neural Networks
Subhashini Venugopalan, Huijun Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, Kate Saenko
North American Chapter of the Association for Computational Linguistics, Denver, Colorado, June 2015.

Pseudocode

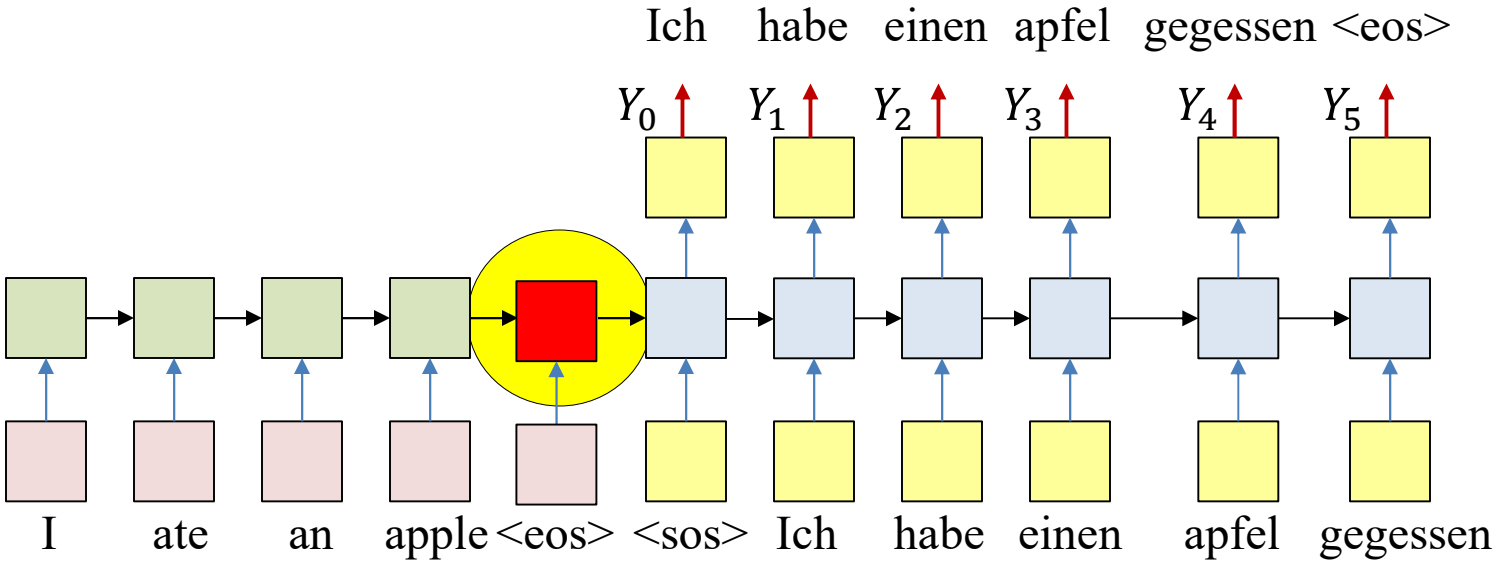
```
# Assuming encoded input H (from text, image, video)
# is available
# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H    # Encoder embedding

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
     t = t+1
     [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1), H)
     y_out(t) = generate(y(t)) # Beam search, random, or greedy
until y_out(t) == <eos>
```

Pseudocode

```
# Assuming encoded input H (from text, image, video)
# is available
# Now generate the output y_out(1),y_out(2),…
t = 0
h_out(0) = H    # Encoder embedding

# Note: begins with a "start of sentence" symbol
#       <sos> and <eos> may be identical
y_out(0) = <sos>
do
     t = t+1
     [y(t),h_out(t)] = RNN_output_step(h_out(t-1), y_out(t-1), H)
     y_out(t) = generate(y(t)) # Beam search, random, or greedy
until y_out(t) == <eos>
```

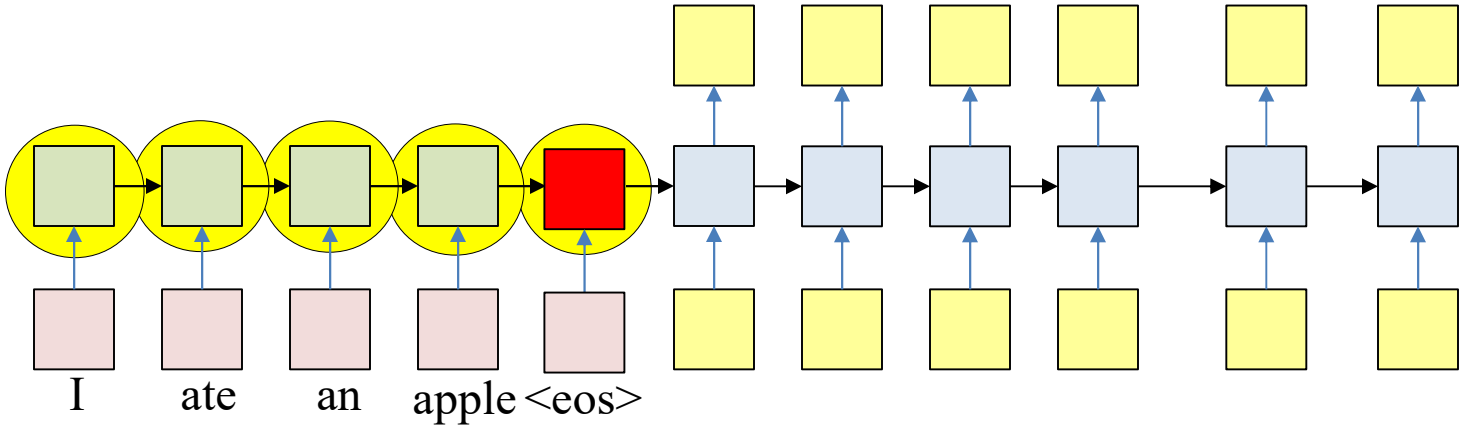
Also consider encoder embedding

A problem with this framework



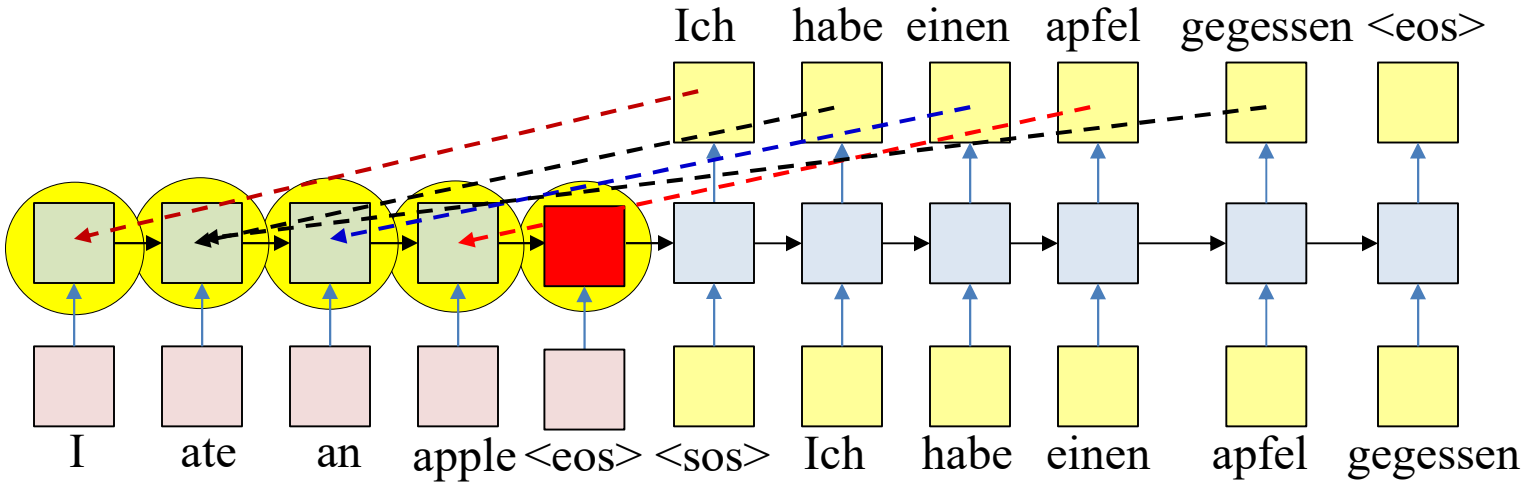
- *All* the information about the input sequence is embedded into a *single* vector
  - The "hidden" node layer at the end of the input sequence
  - This one node is "overloaded" with information
    - Particularly if the input is long

A problem with this framework



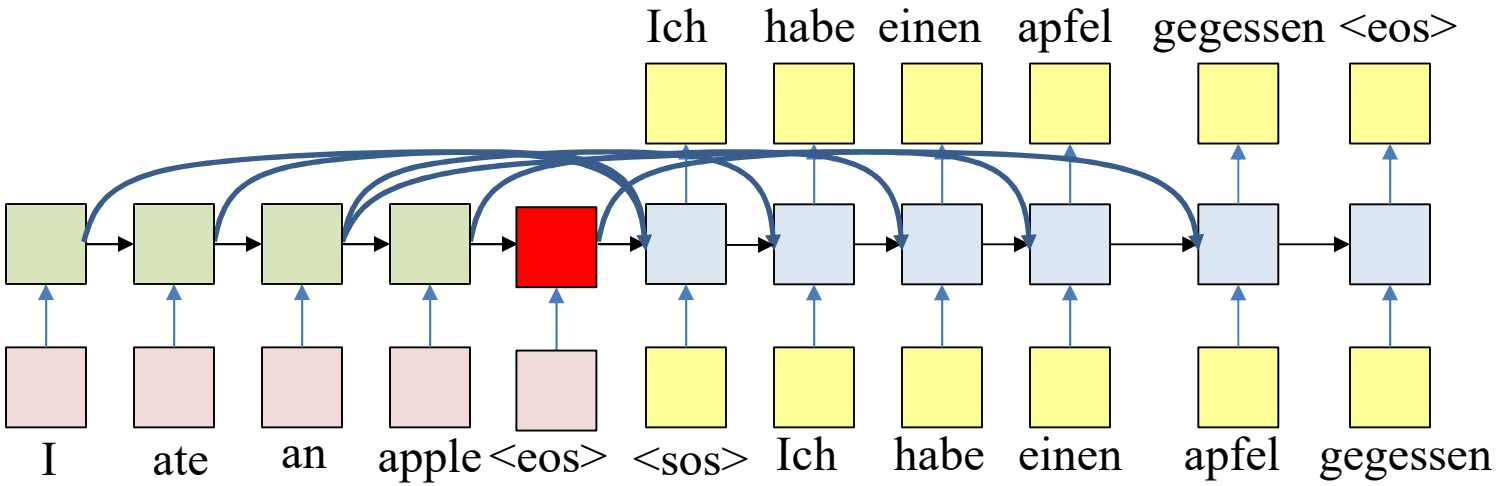
- In reality: *All* hidden values carry information
  - Some of which may be diluted downstream

A problem with this framework



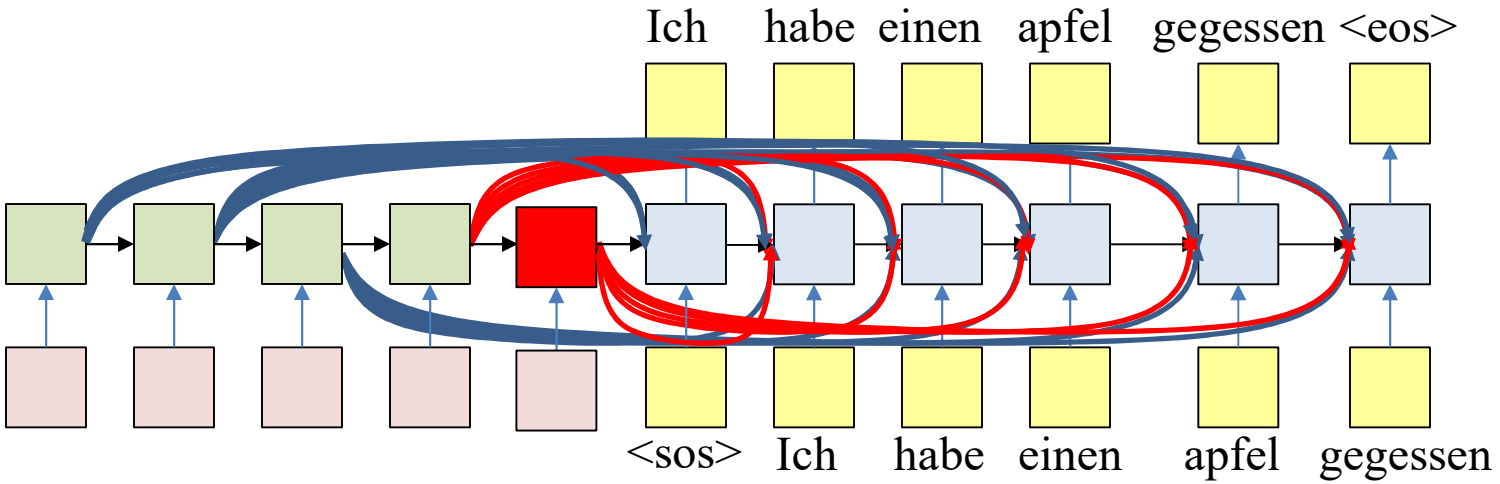
- In reality: *All* hidden values carry information
  - Some of which may be diluted downstream
- Different outputs are related to different inputs
  - Recall input and output may not be in sequence

A problem with this framework



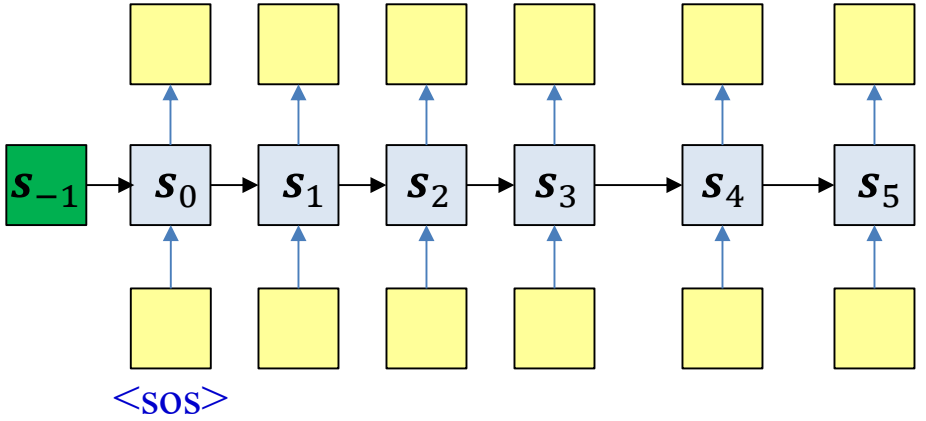
- In reality: *All* hidden values carry information
  - Some of which may be diluted downstream
- Different outputs are related to different inputs
  - Recall input and output may not be in sequence
  - Have no way of knowing a priori which input must connect to what output

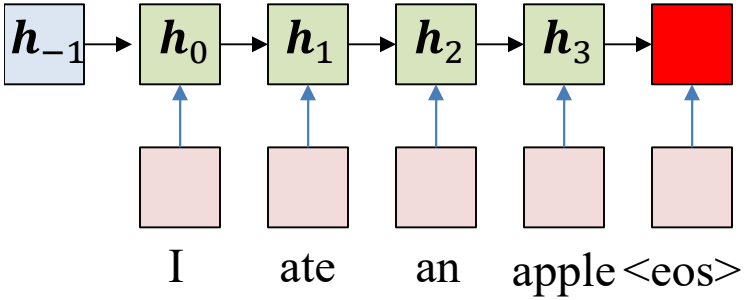
A problem with this framework



- In reality: *All* hidden values carry information
  - Some of which may be diluted downstream
- Different outputs are related to different inputs
  - Recall input and output may not be in sequence
  - Have no way of knowing a priori which input must connect to what output
- Connecting everything to everything is infeasible
  - Variable sized inputs and outputs
  - Overparametrized
  - Connection pattern ignores the actual asynchronous dependence of output on input

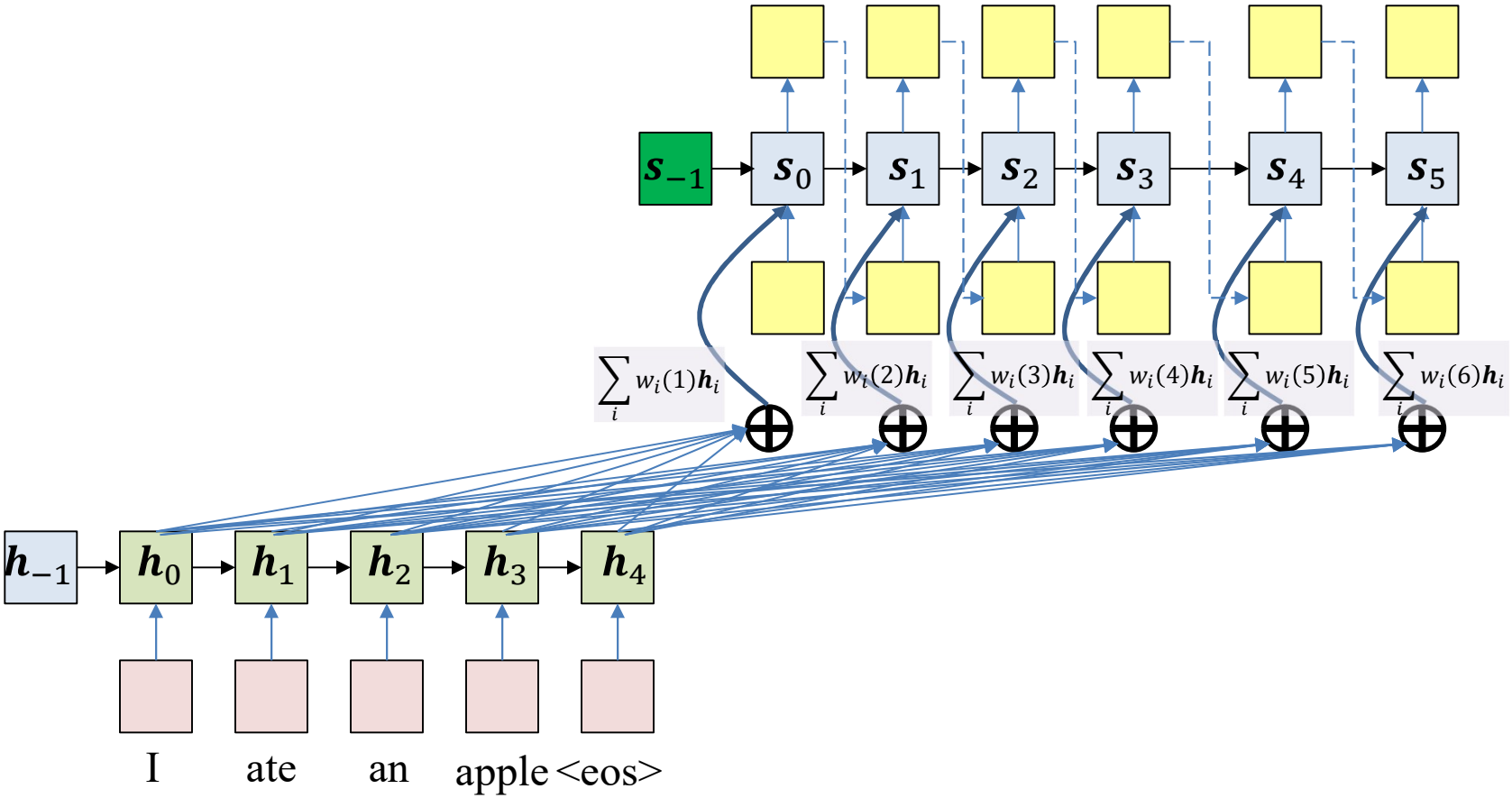
Solution: Attention models





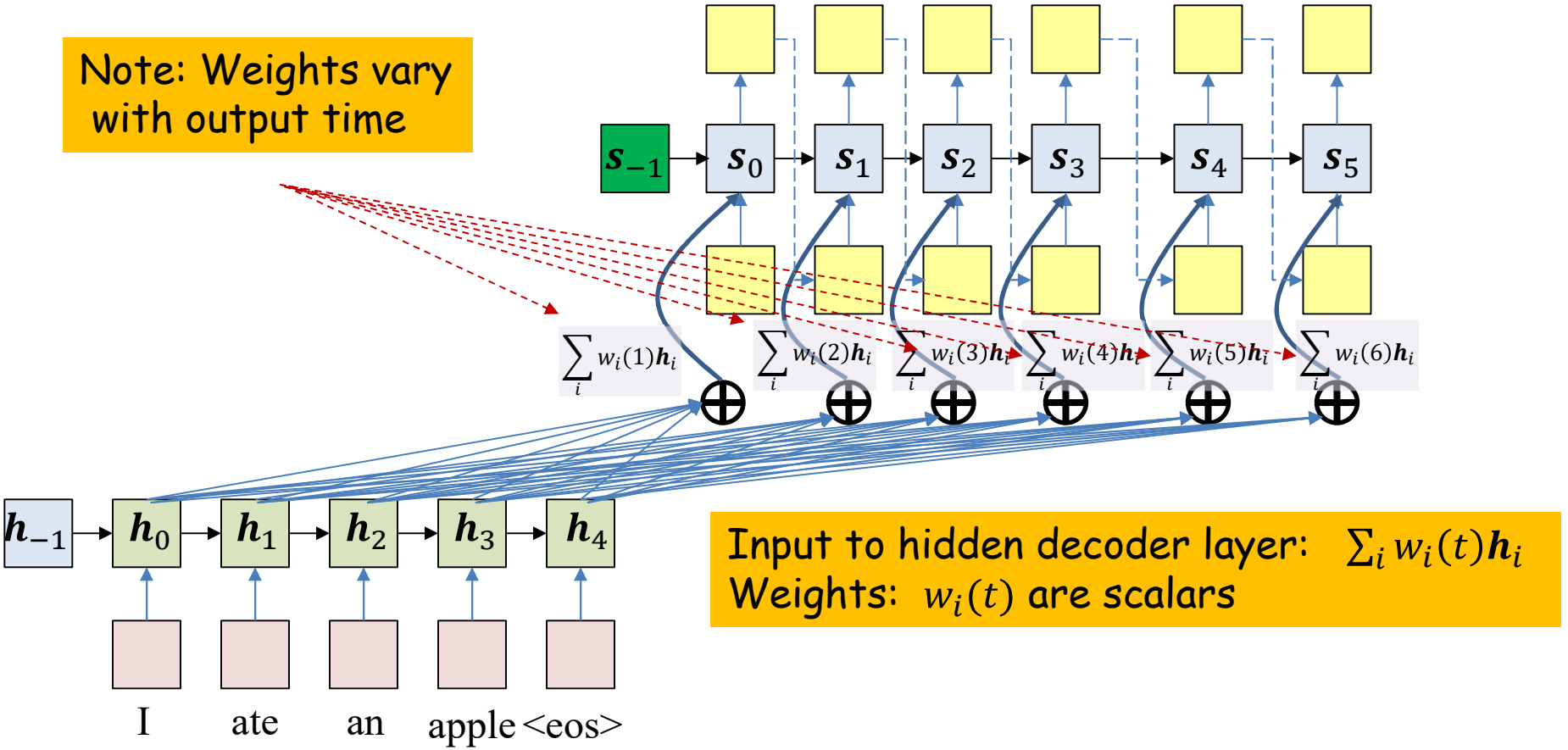
- Separating the encoder and decoder in illustration

Solution: Attention models



- Compute a weighted combination of all the hidden outputs into a single vector
  - Weights vary by output time

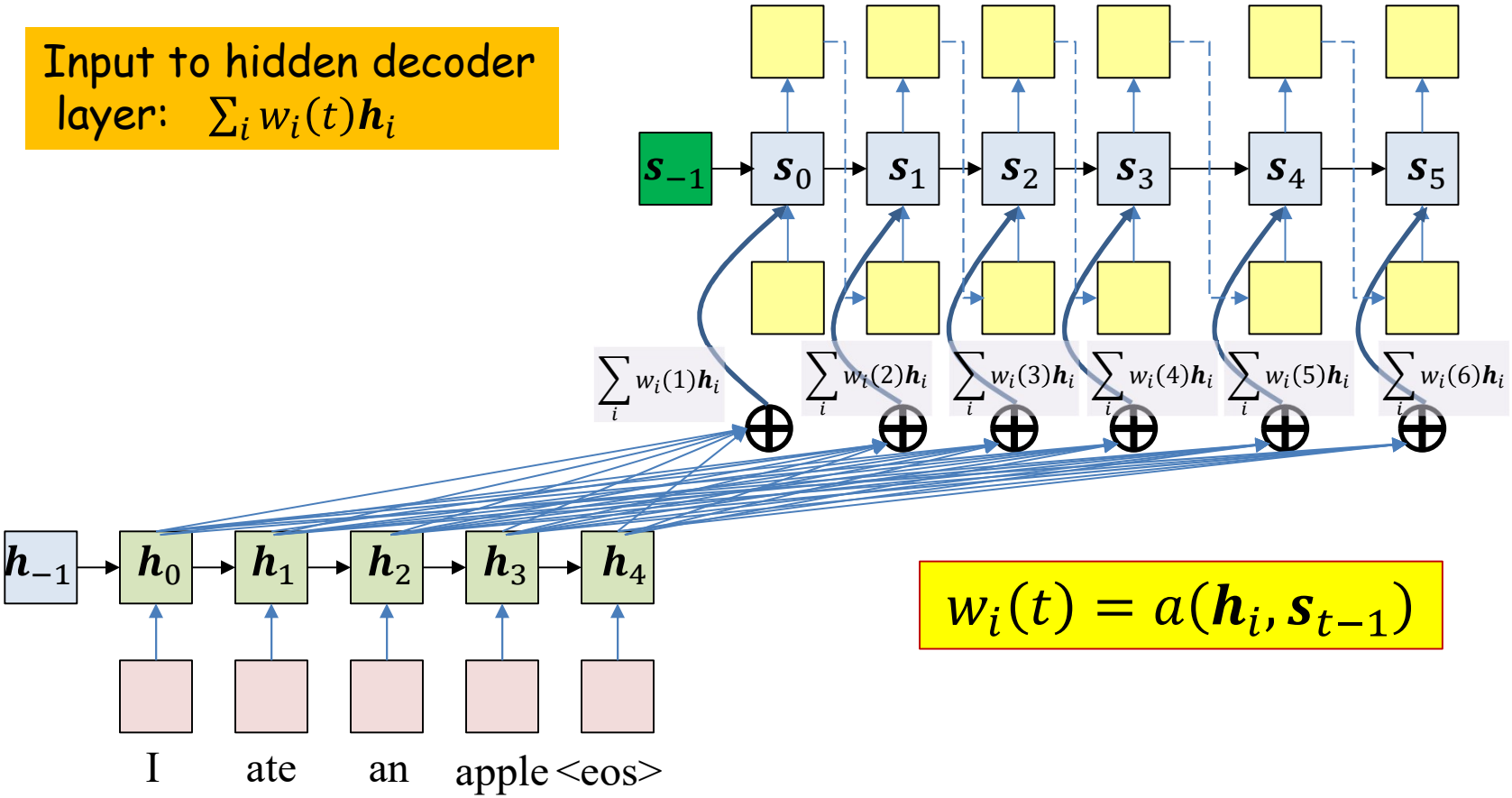
Solution: Attention models



- Compute a weighted combination of all the hidden outputs into a single vector
  - Weights vary by output time

Solution: Attention models

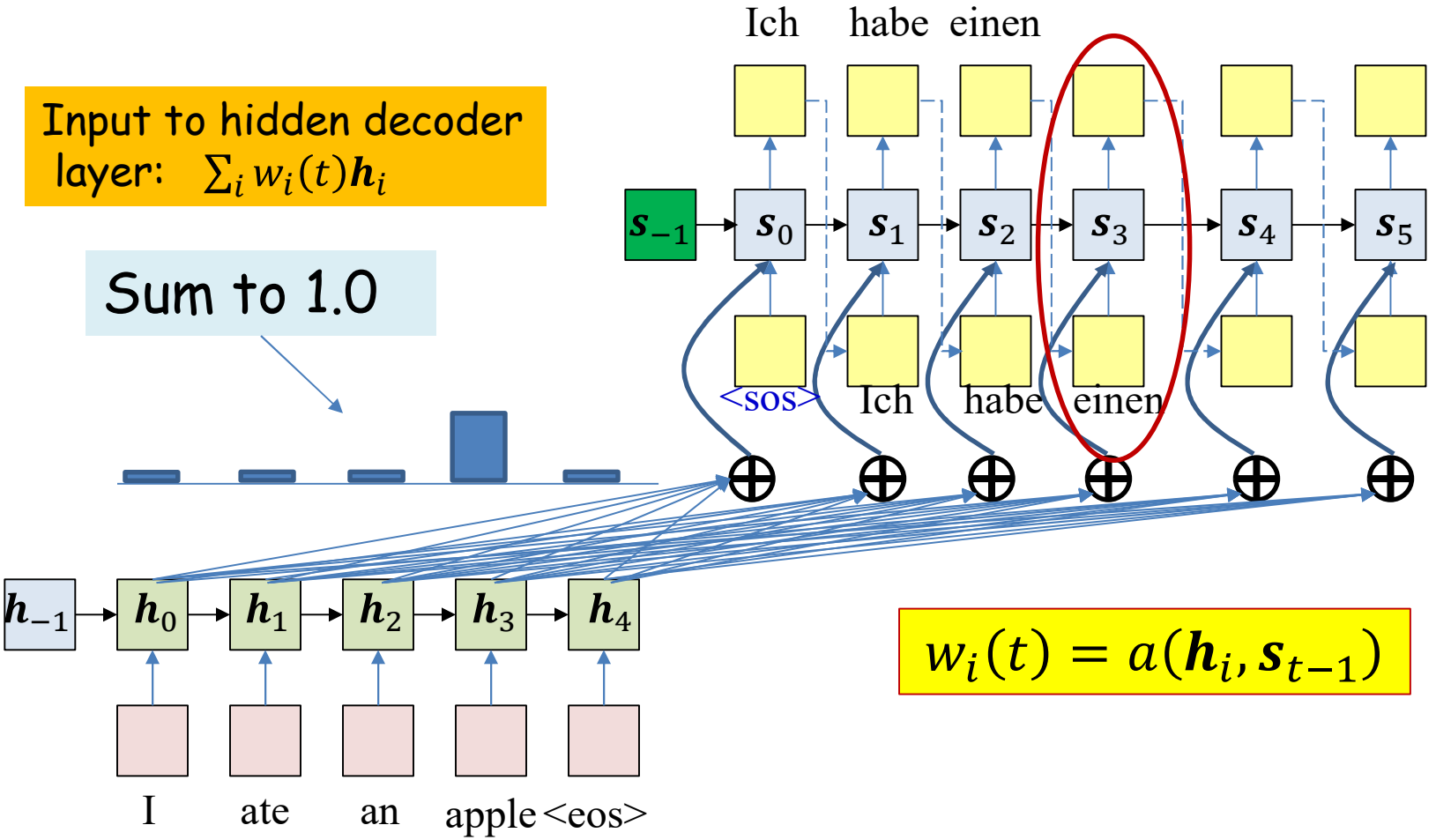
Input to hidden decoder layer: $\sum_i w_i(t)\boldsymbol{h}_i$



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

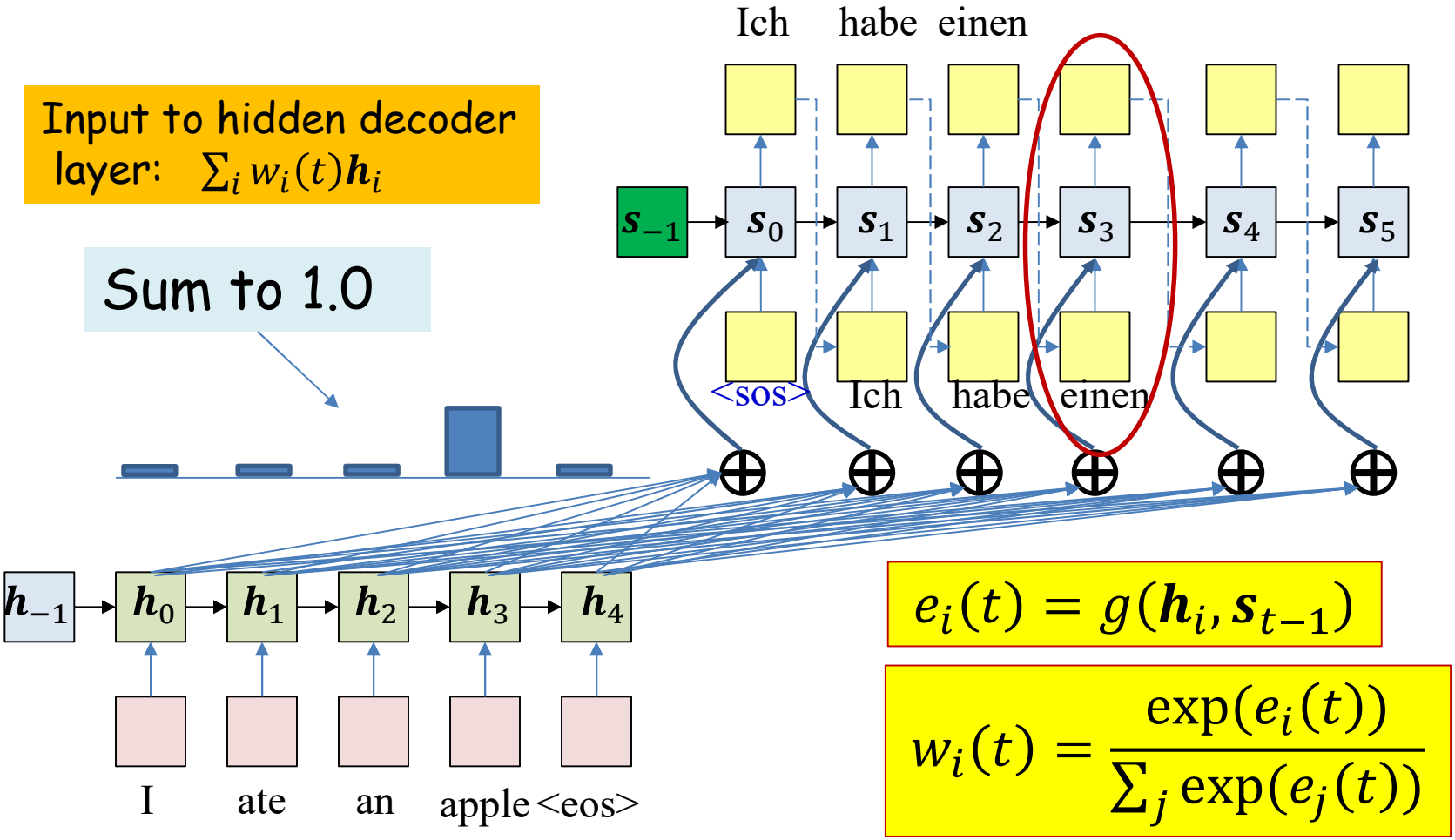
- Require a time-varying weight that specifies relationship of output time to input time
  - Weights are *functions* of current output state

Attention models



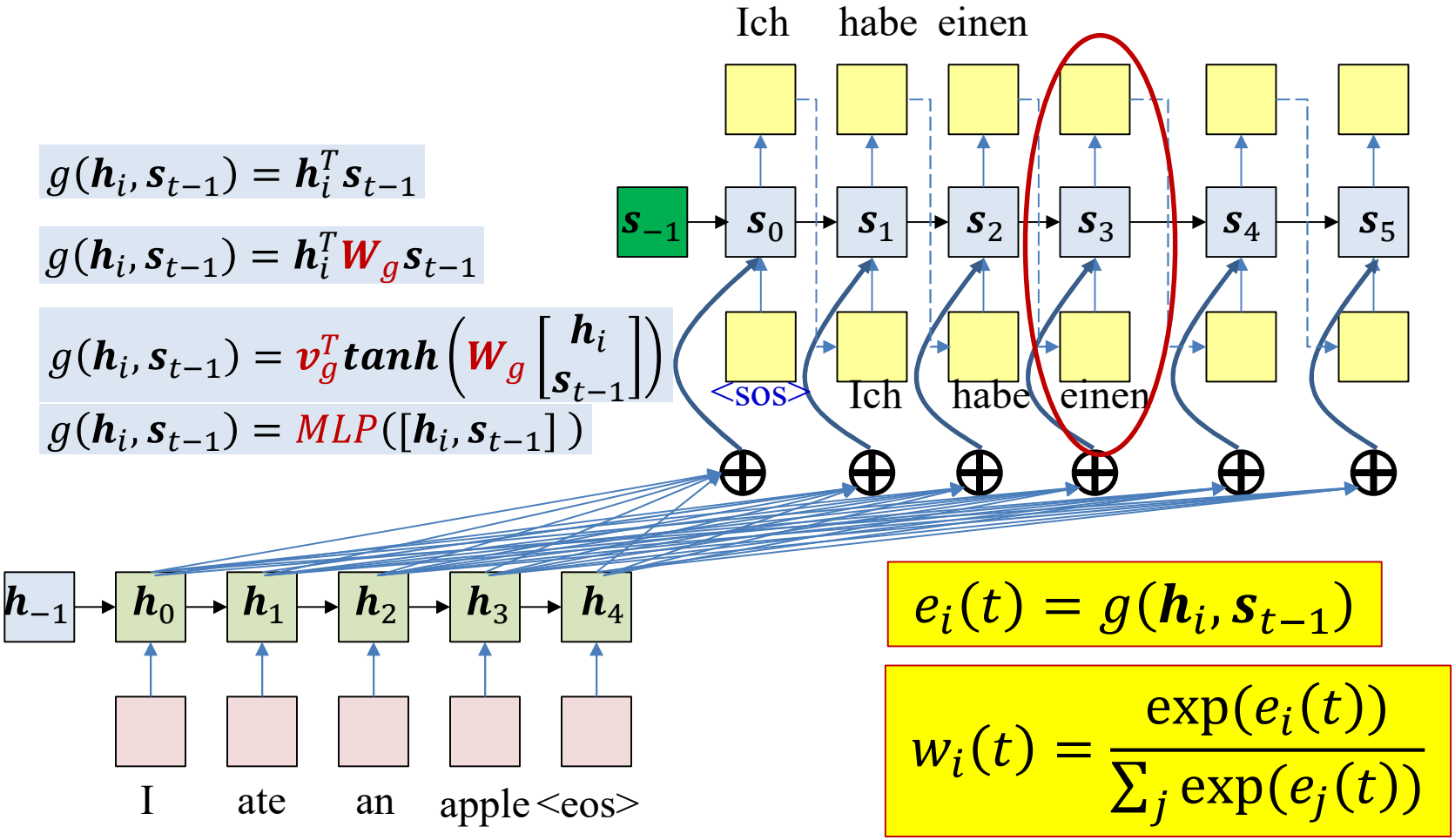
- The weights are a distribution over the input
  - Must automatically highlight the most important input components for any output

Attention models



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

Attention models



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

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

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

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

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

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

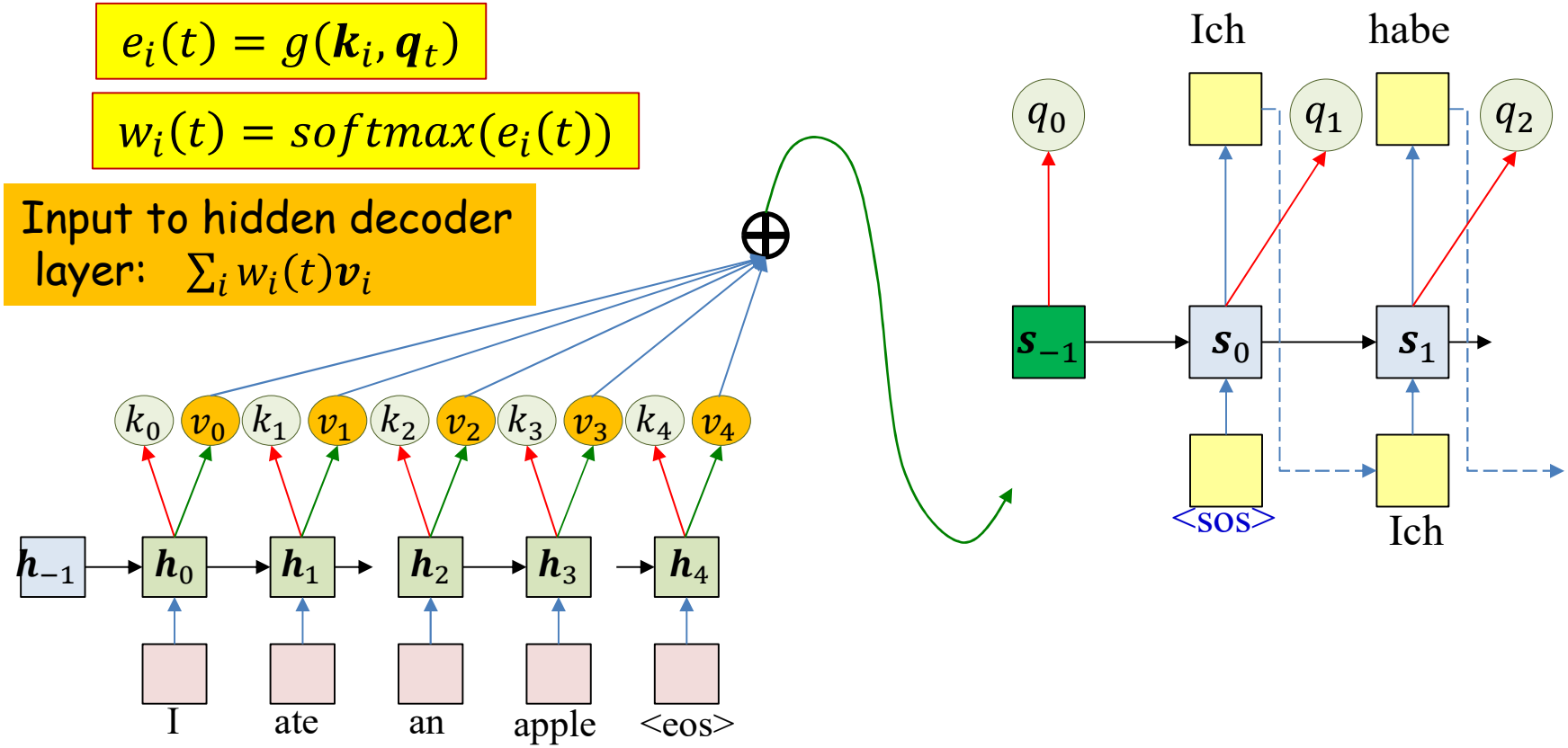
- Typical options for $g()$…
  - Variables in red are to be learned

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



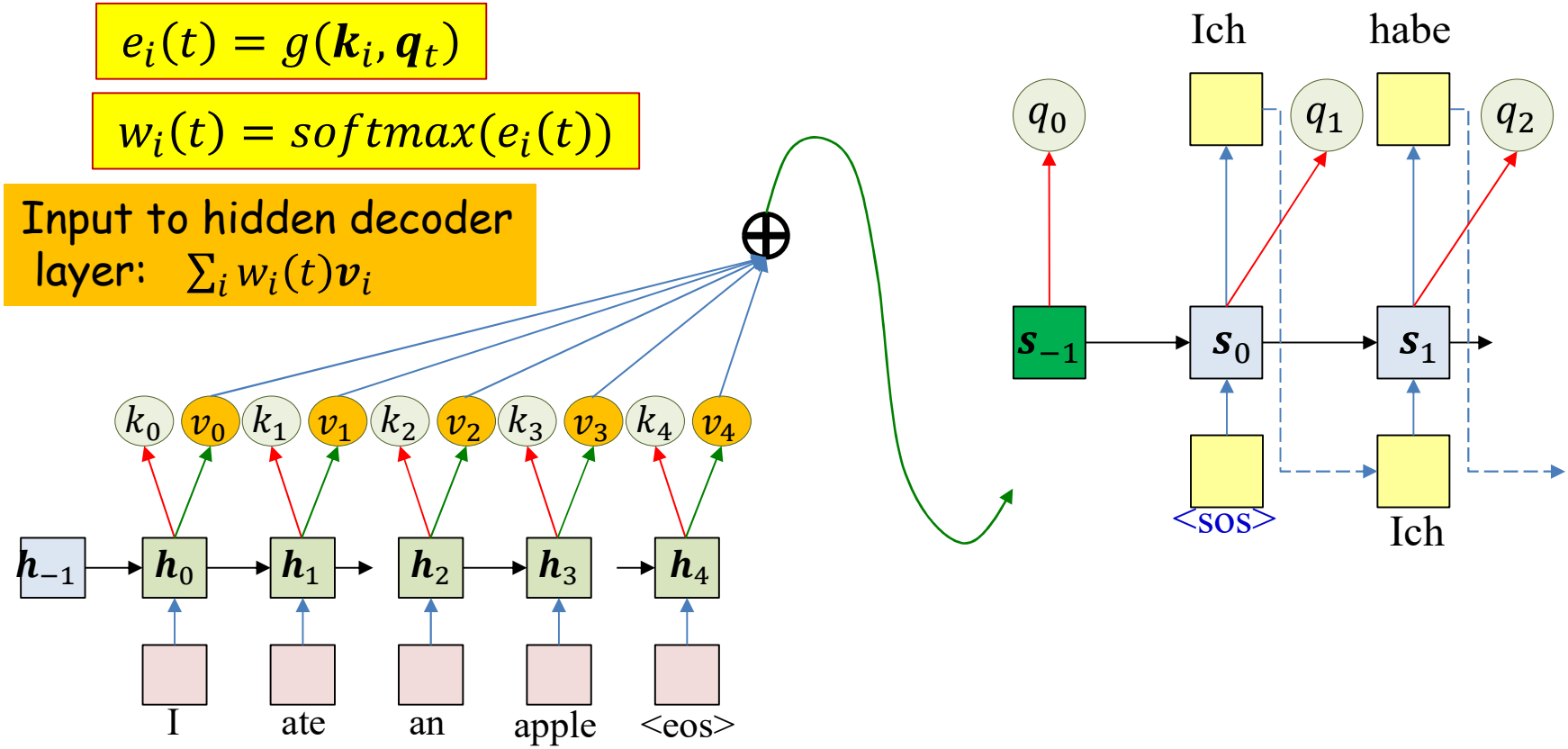
- Encoder outputs an explicit "key" and "value" at each input time
  - Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit "query" at each output time
  - Query is used to evaluate which inputs to pay attention to
- The weight is a function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

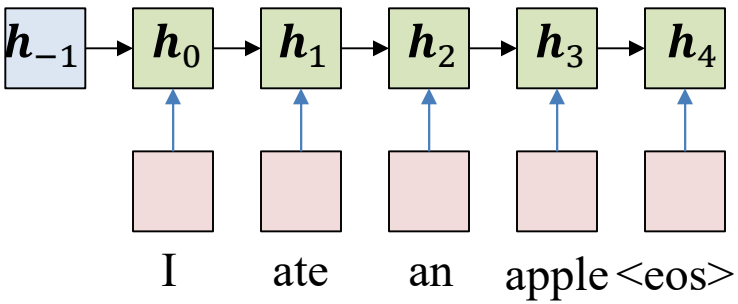


Special case: $k_i = v_i = h_i$

$q_t = s_{t-1}$

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

Converting an input (forward pass)



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

Converting an input (forward pass)

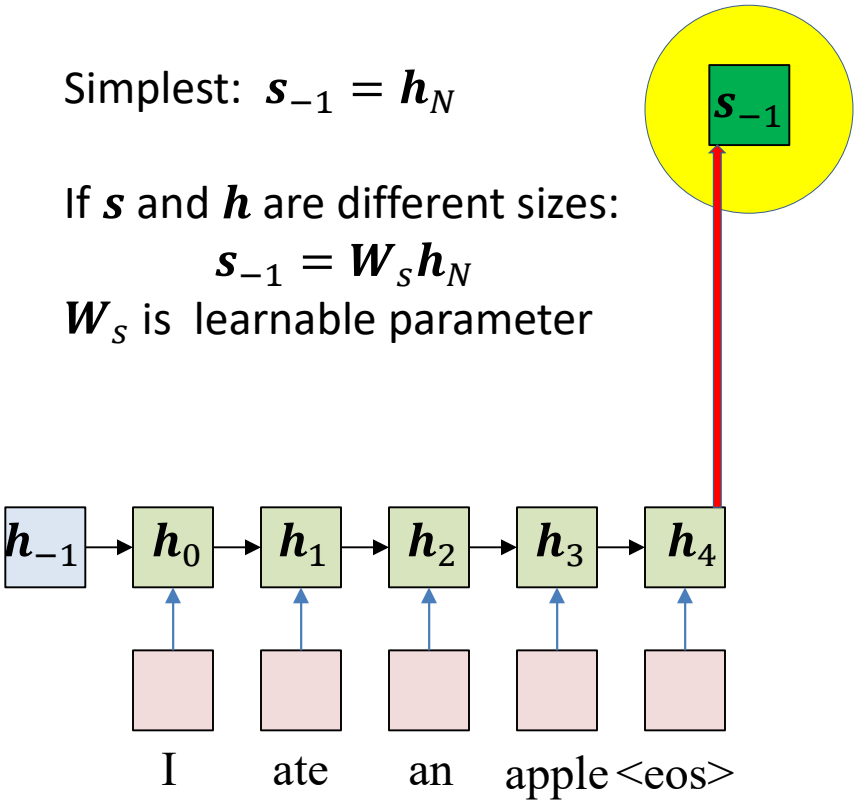
What is this?
Multiple options

Simplest: $\boldsymbol{s_{-1} = h_N}$

If $\boldsymbol{s}$ and $\boldsymbol{h}$ are different sizes:

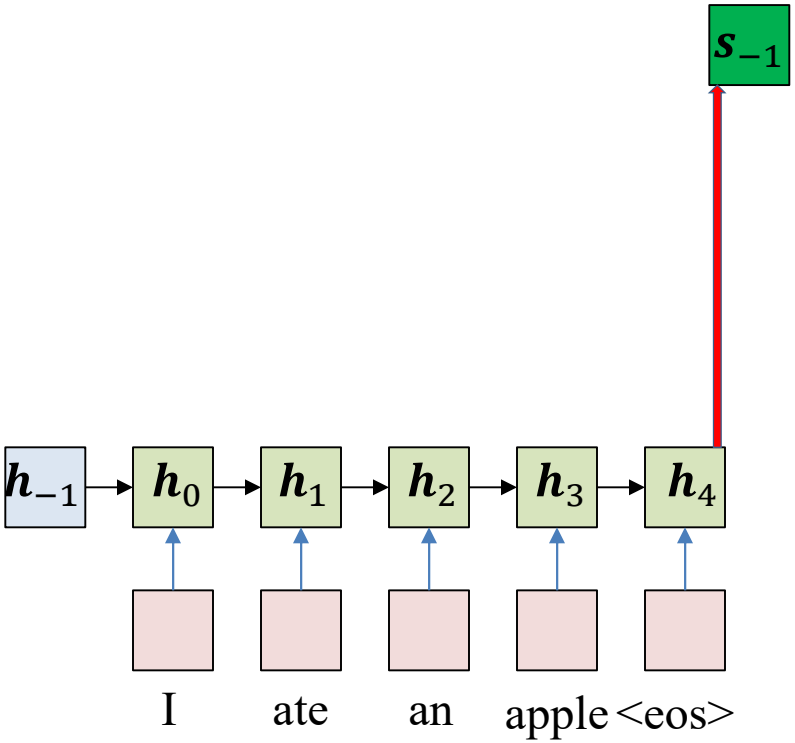
$$\boldsymbol{s_{-1} = W_s h_N}$$

$\boldsymbol{W_s}$ is learnable parameter



- Initialize decoder hidden state

Converting an input (forward pass)



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

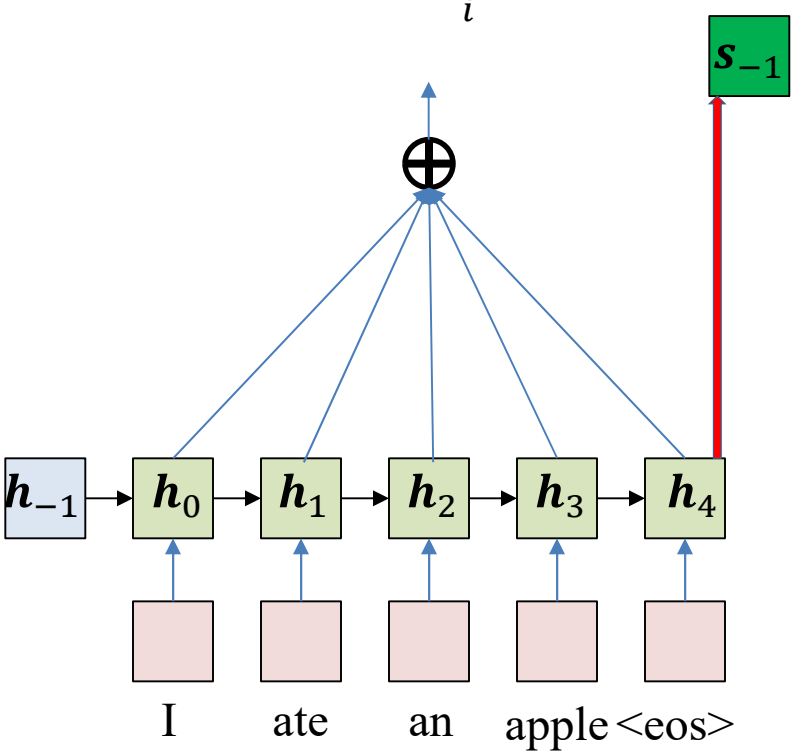
$$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$$

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

- Compute weights (for every $\boldsymbol{h}_i$) for first output

Converting an input (forward pass)

$$z_0 = \sum_i w_i(0) h_i$$



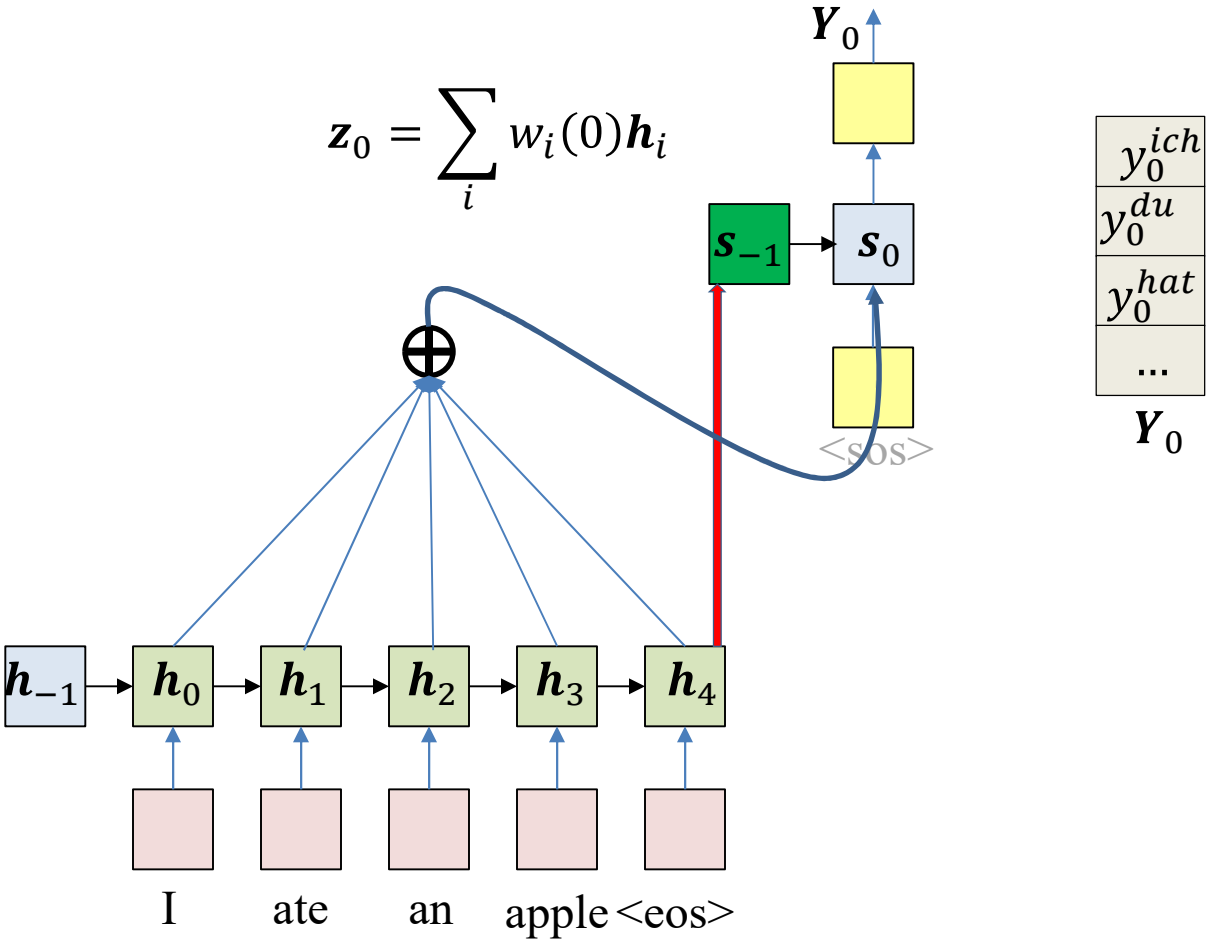
$$g(h_i, s_{-1}) = h_i^T W_g s_{-1}$$

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

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

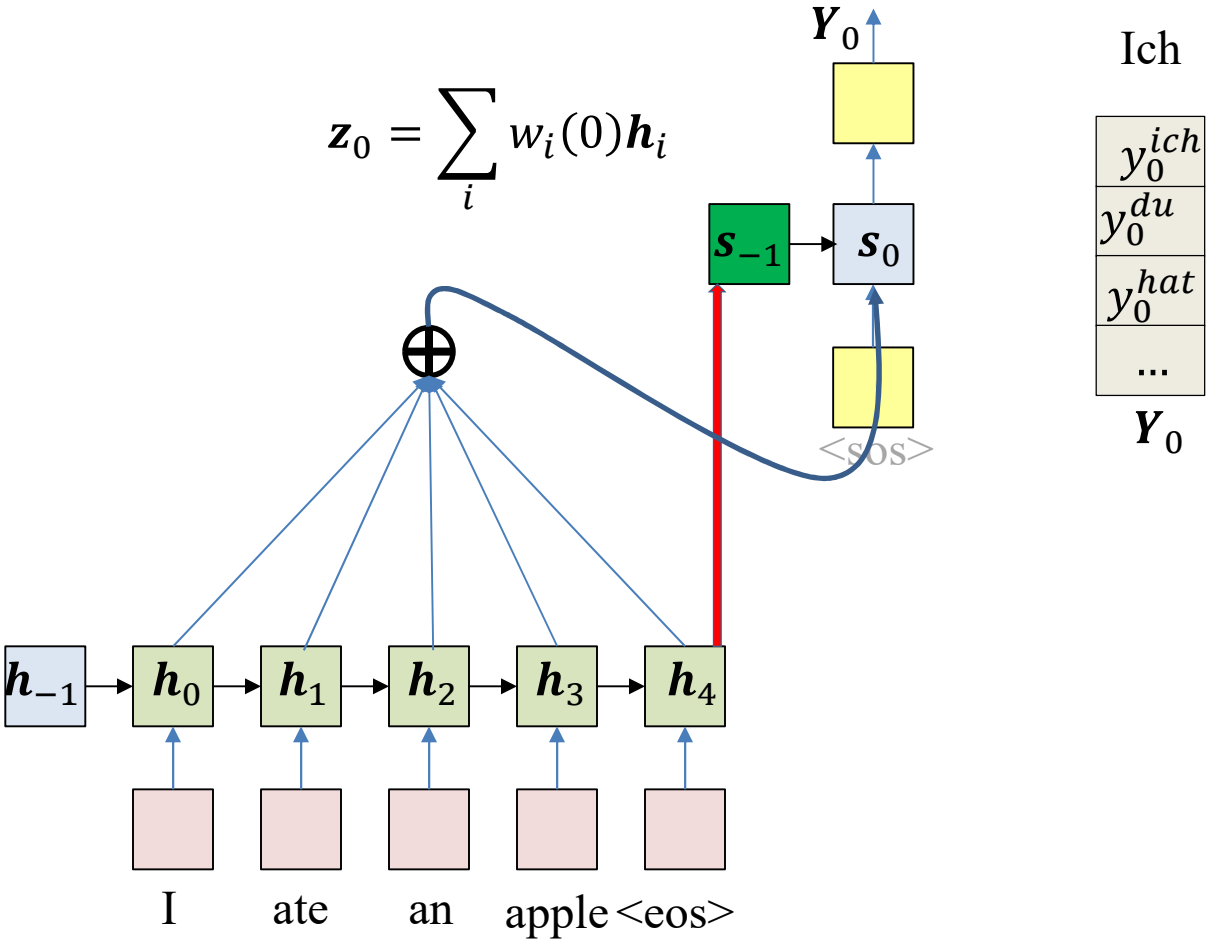
- Compute weights (for every h_i) for first output
- Compute weighted combination of hidden values

Converting an input (forward pass)

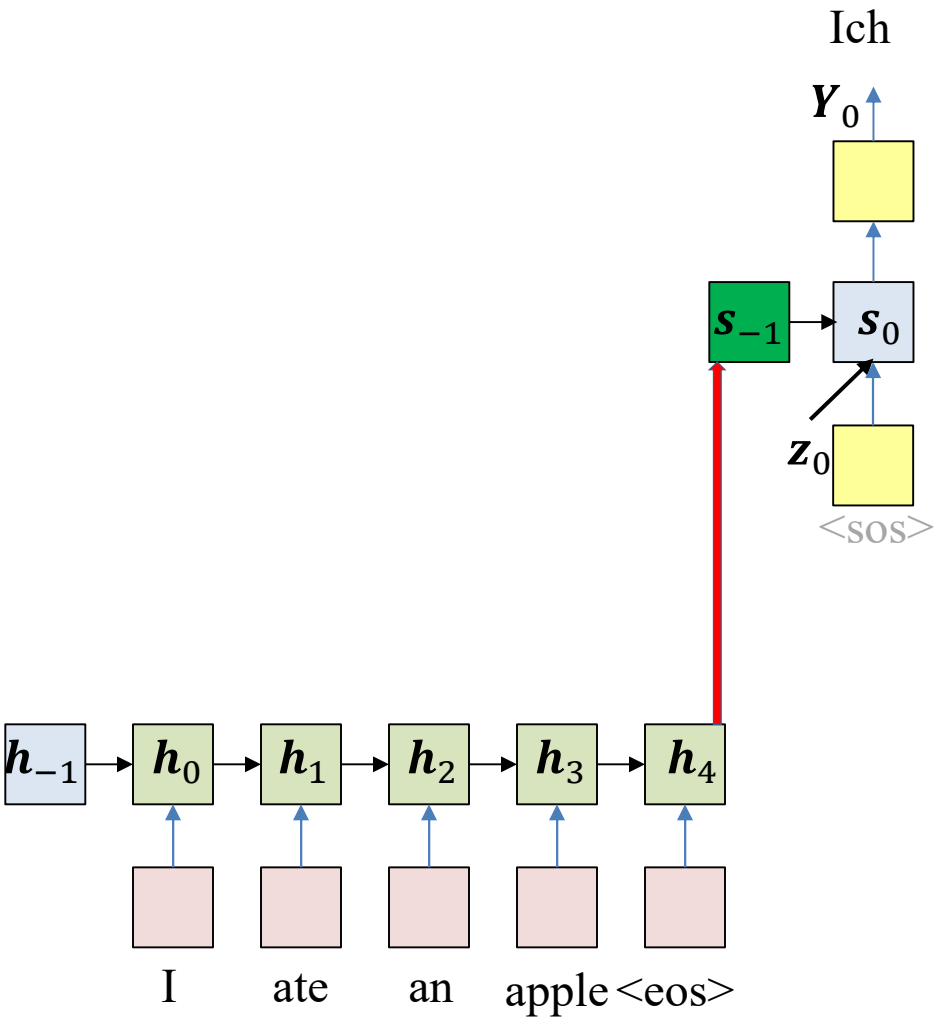


- Produce the first output
  - Will be distribution over words

Converting an input (forward pass)



- Produce the first output
  - Will be distribution over words
  - Draw a word from the distribution

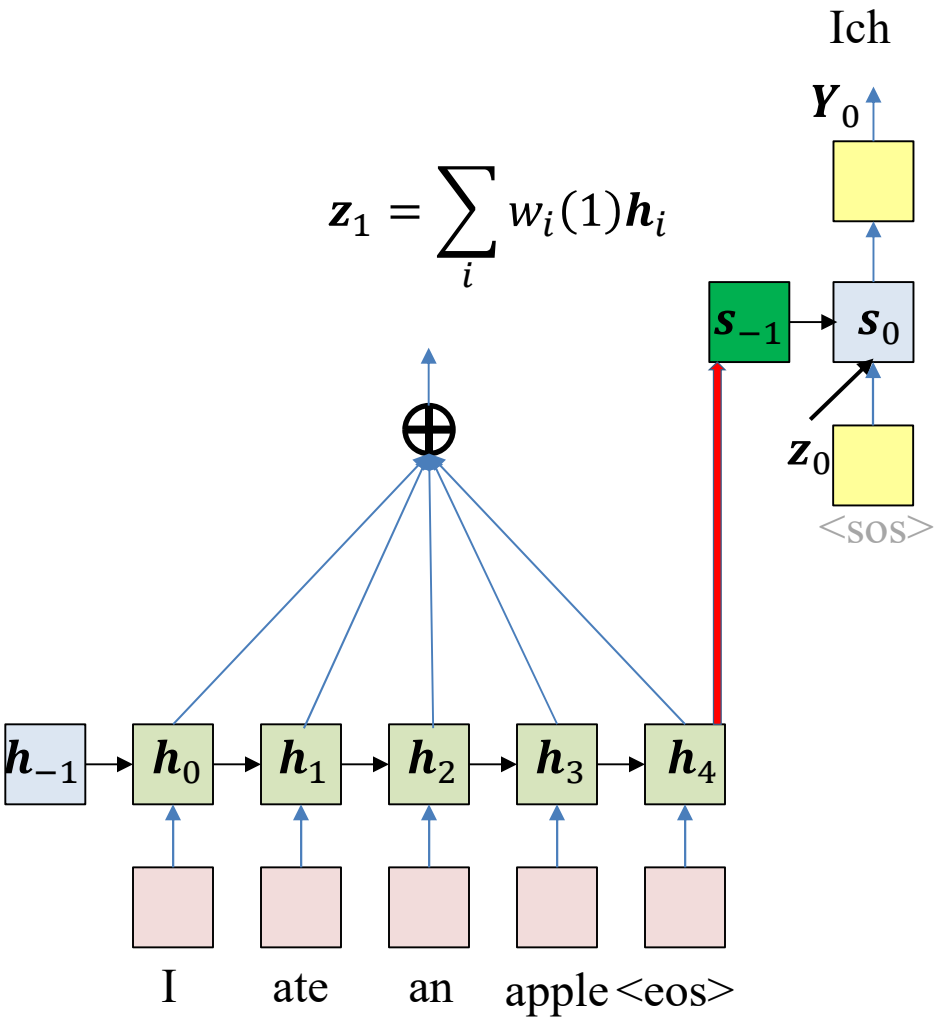


$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

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

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

- Compute the weights for all instances for time = 1

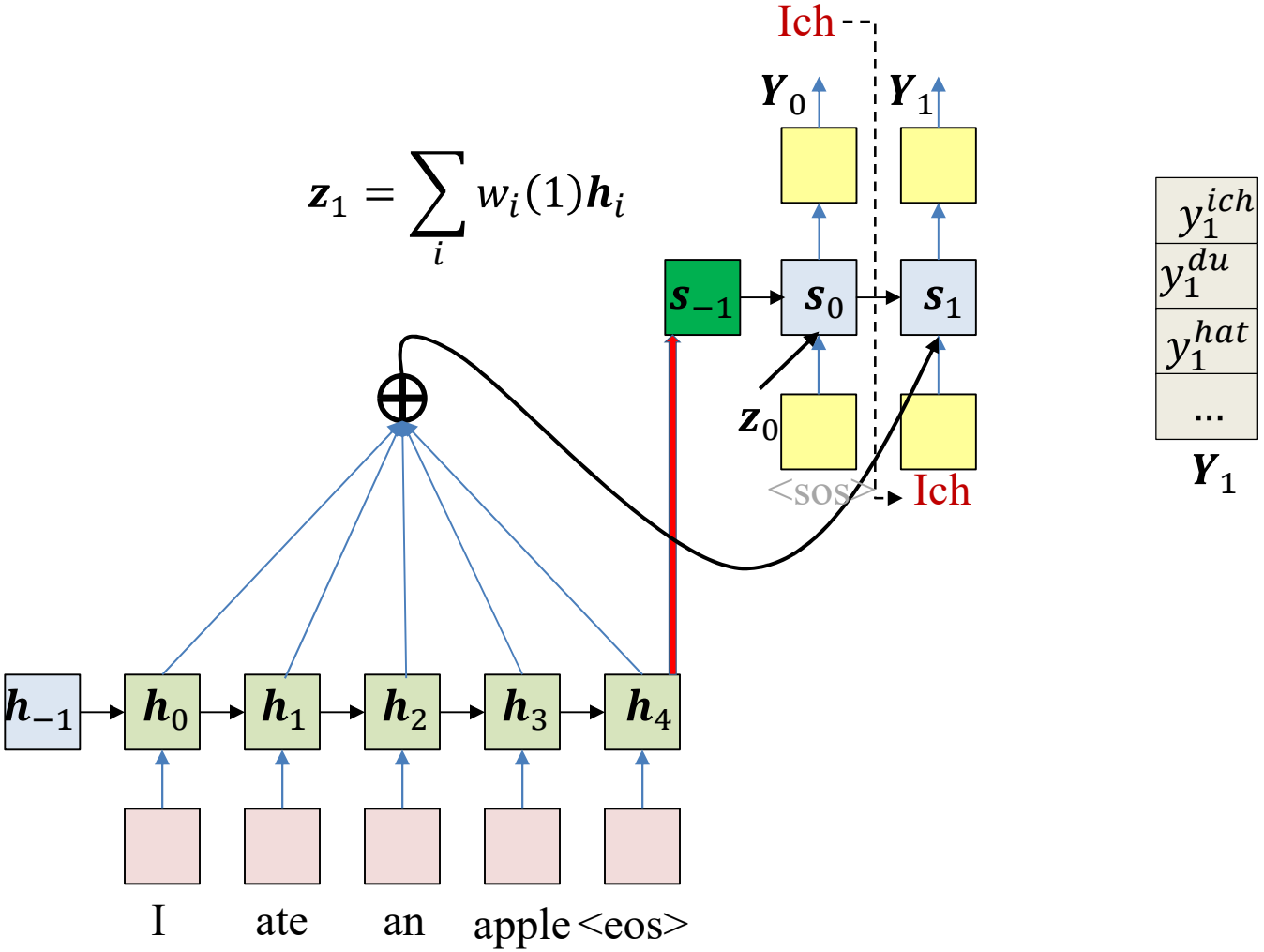


$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

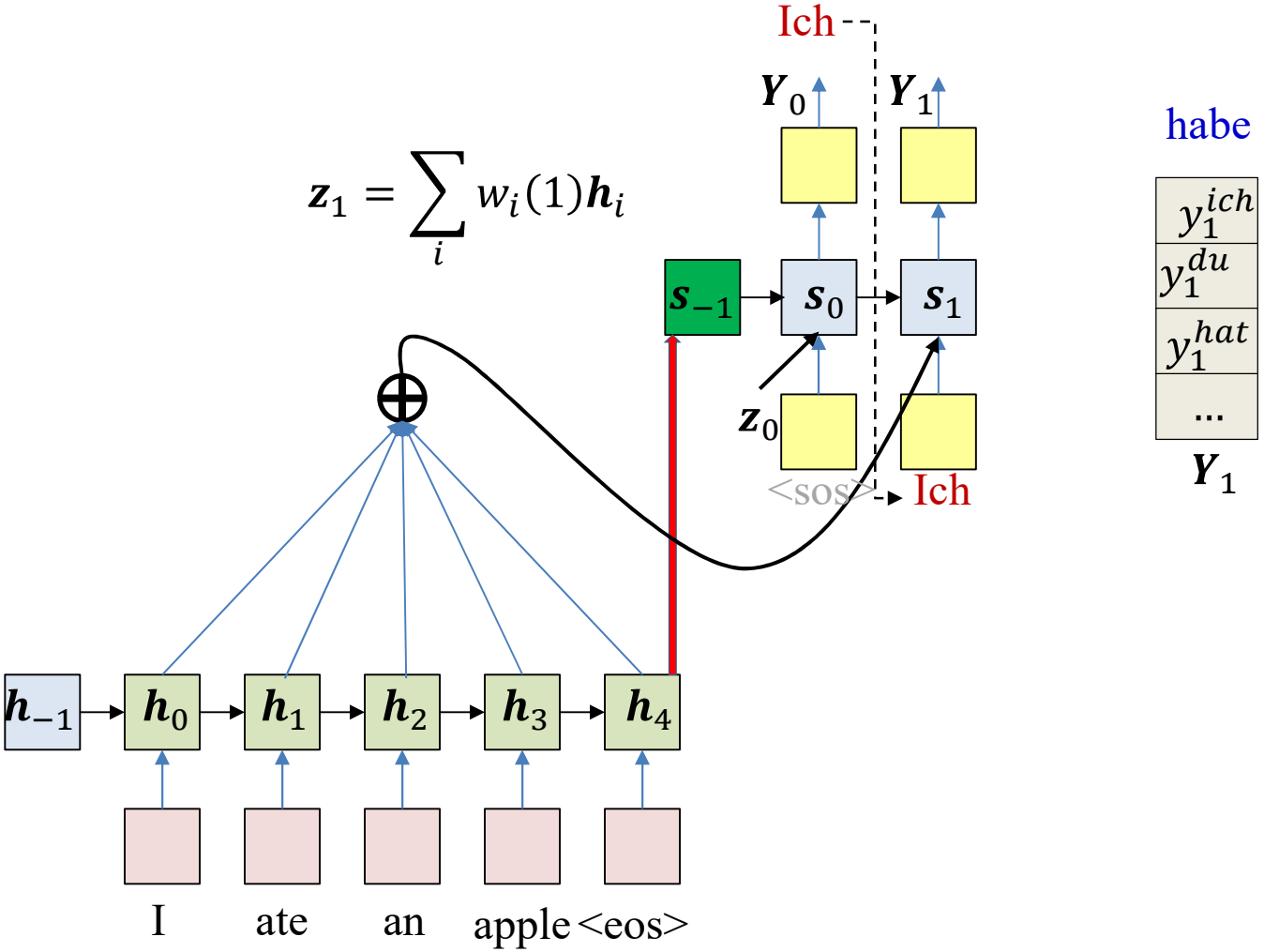
$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

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

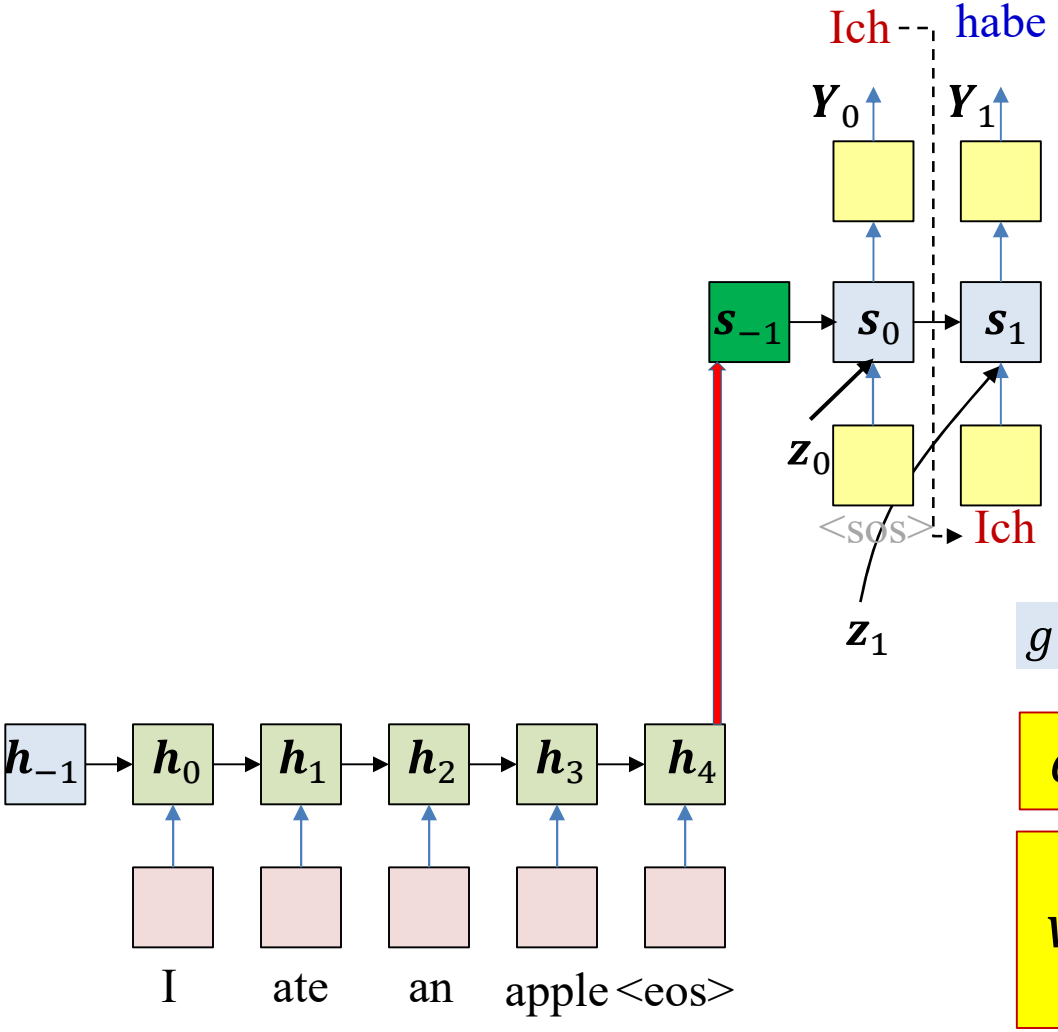
- Compute the weighted sum of hidden input values at t=1



- Compute the output at t=1
  - Will be a probability distribution over words



- Draw a word from the output distribution at t=1

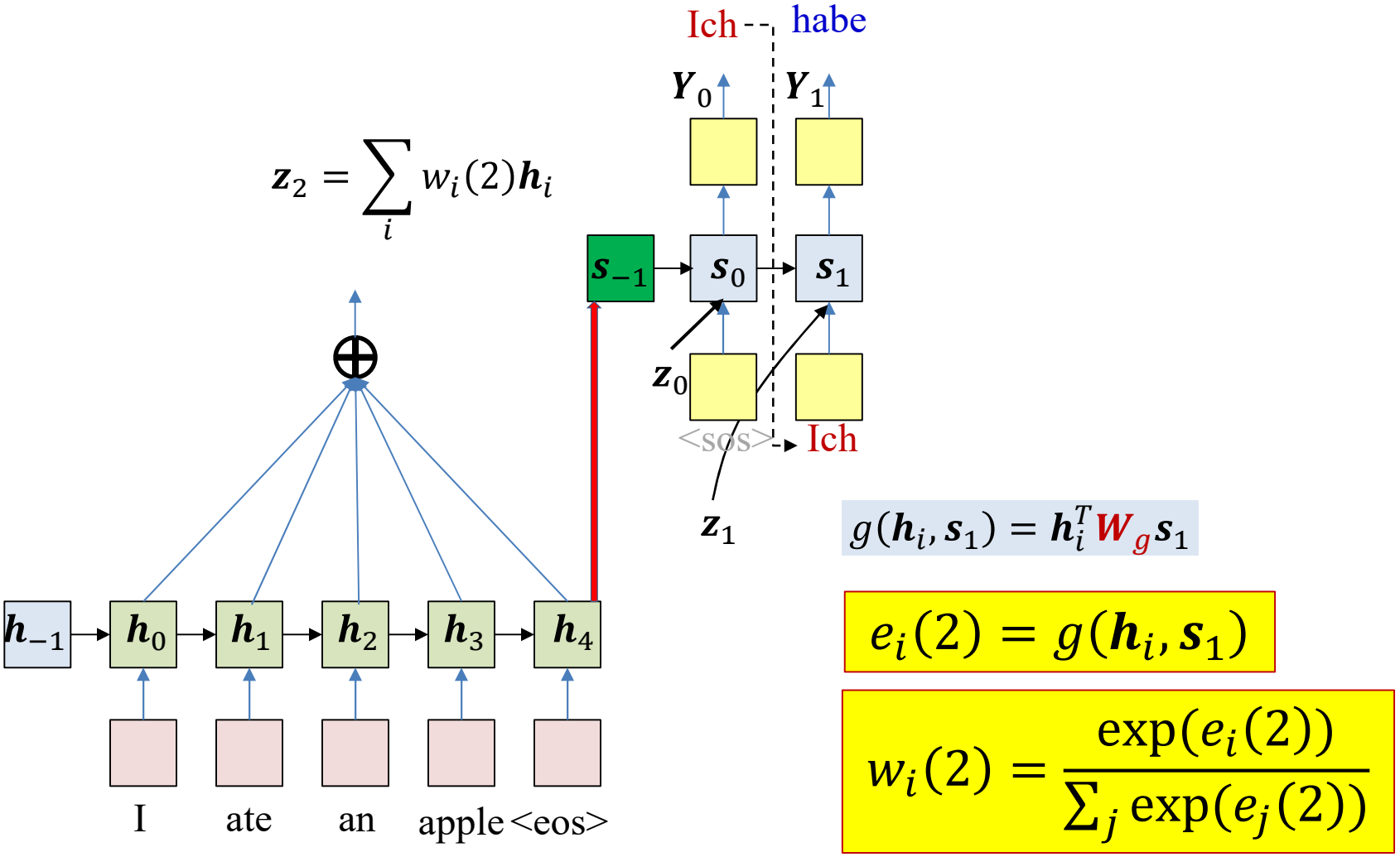


$$g(\boldsymbol{h}_i, \boldsymbol{s}_1) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_1$$

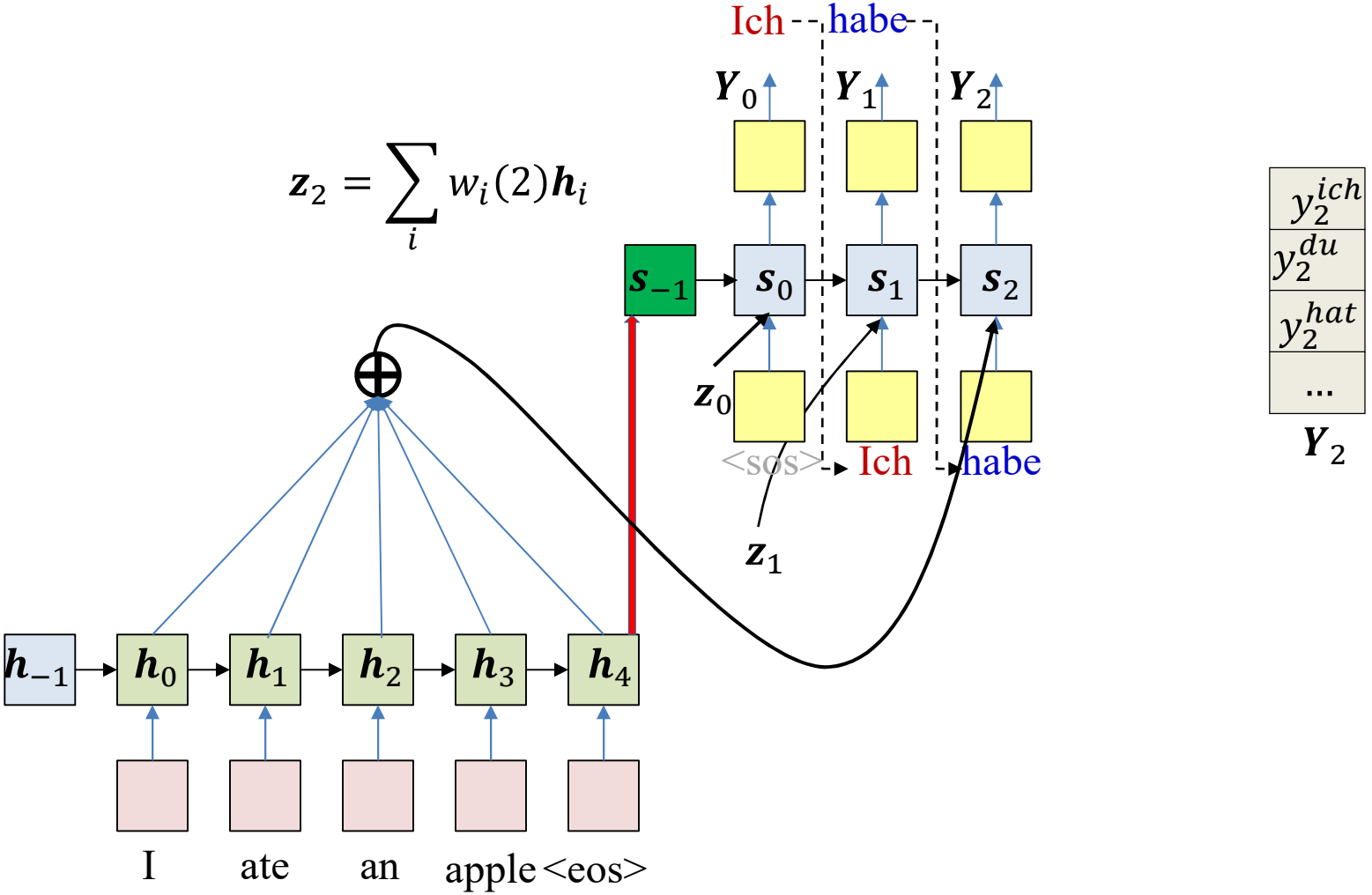
$$e_i(2) = g(\boldsymbol{h}_i, \boldsymbol{s}_1)$$

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

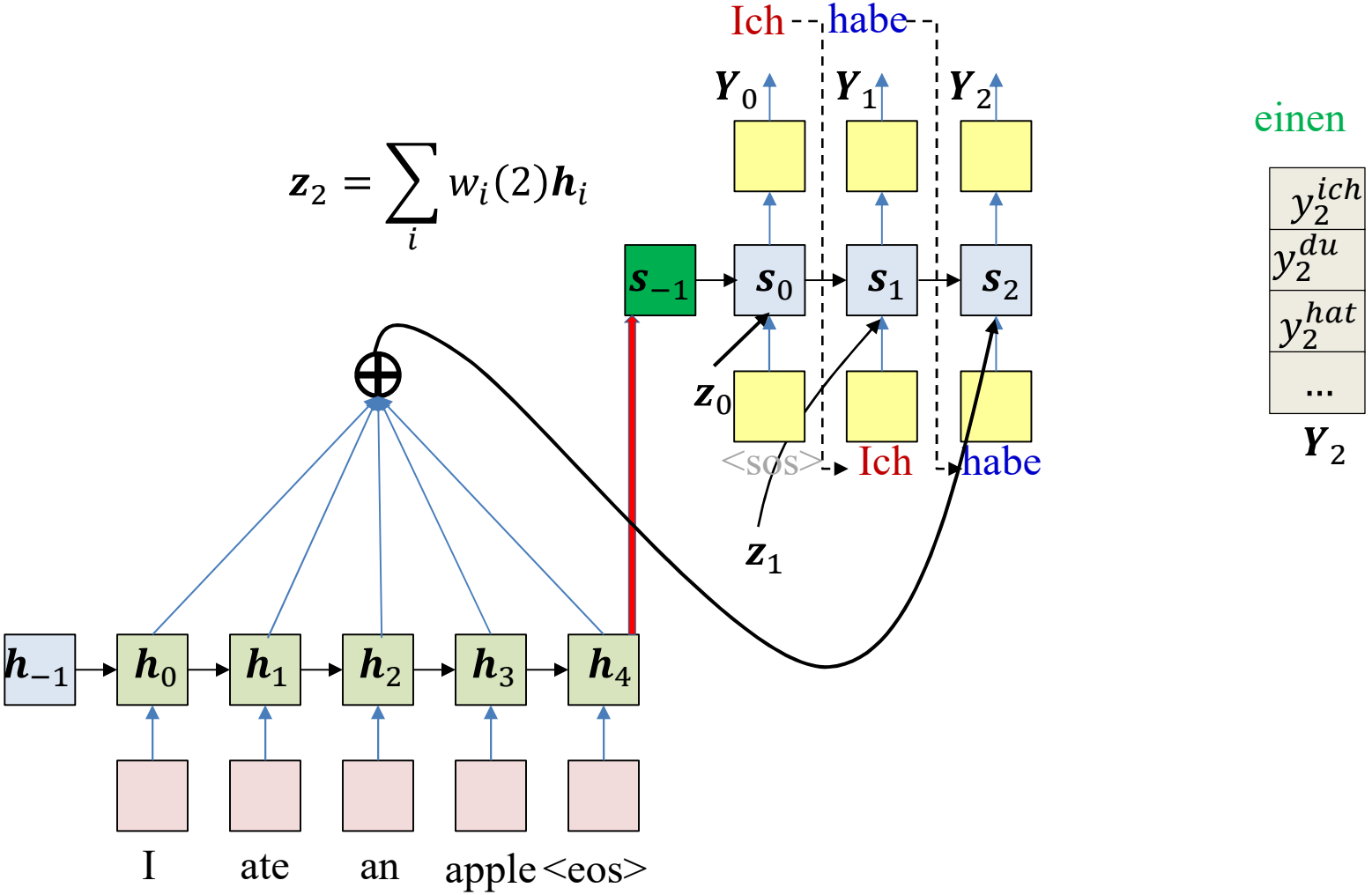
- Compute the weights for all instances for time = 2



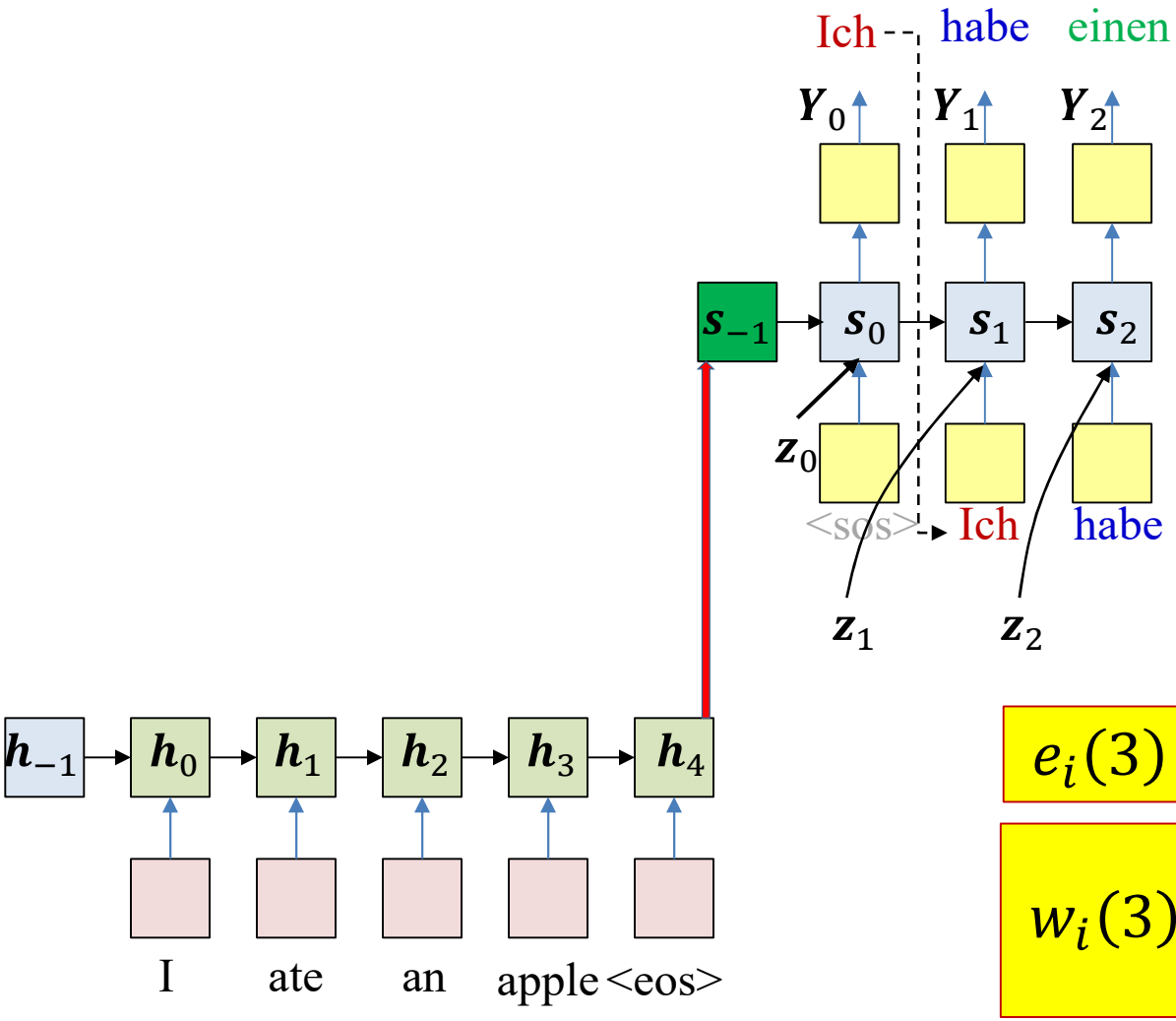
- Compute the weighted sum of hidden input values at t=2



- Compute the output at t=2
  - Will be a probability distribution over words



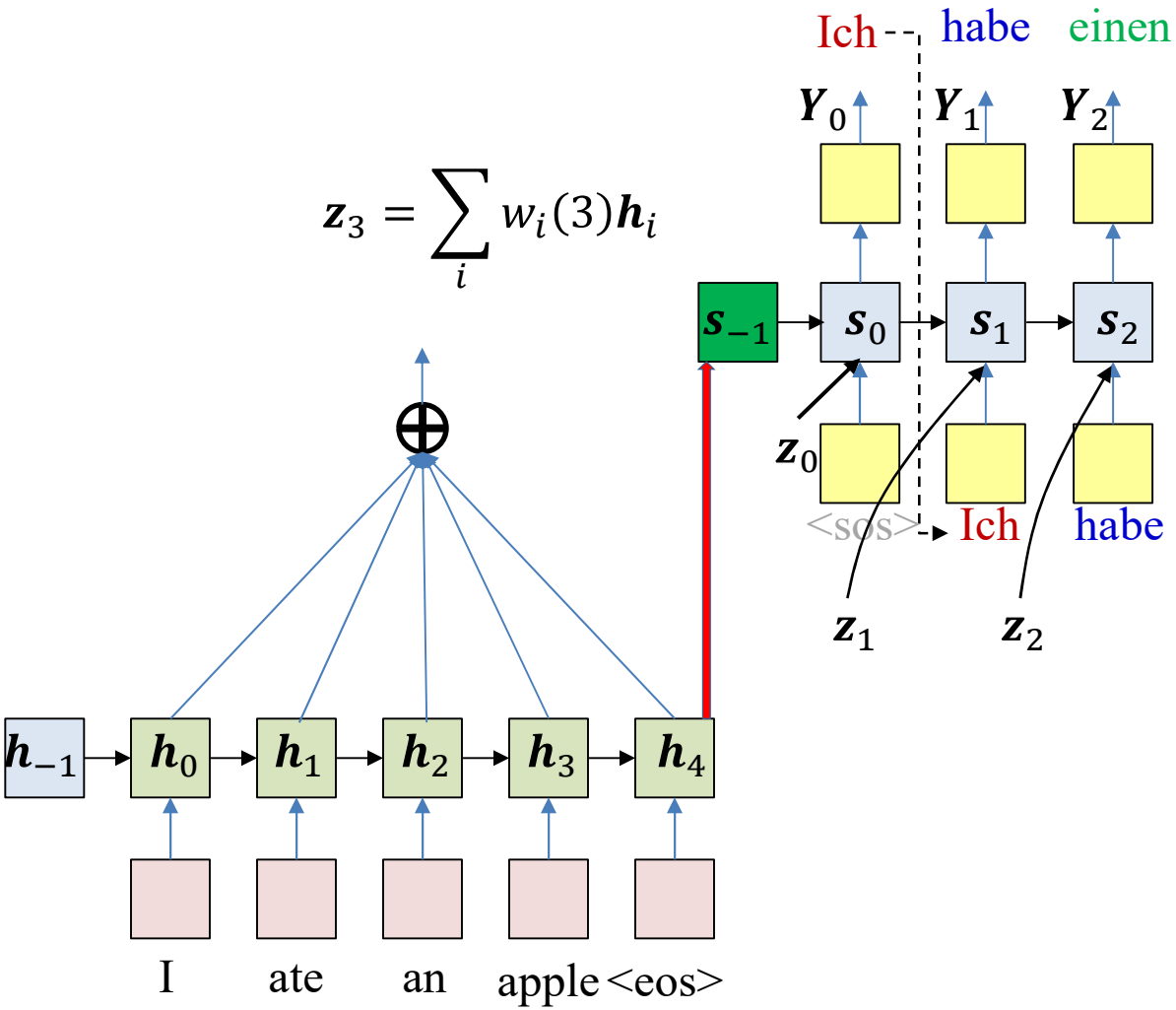
- Draw a word from the distribution



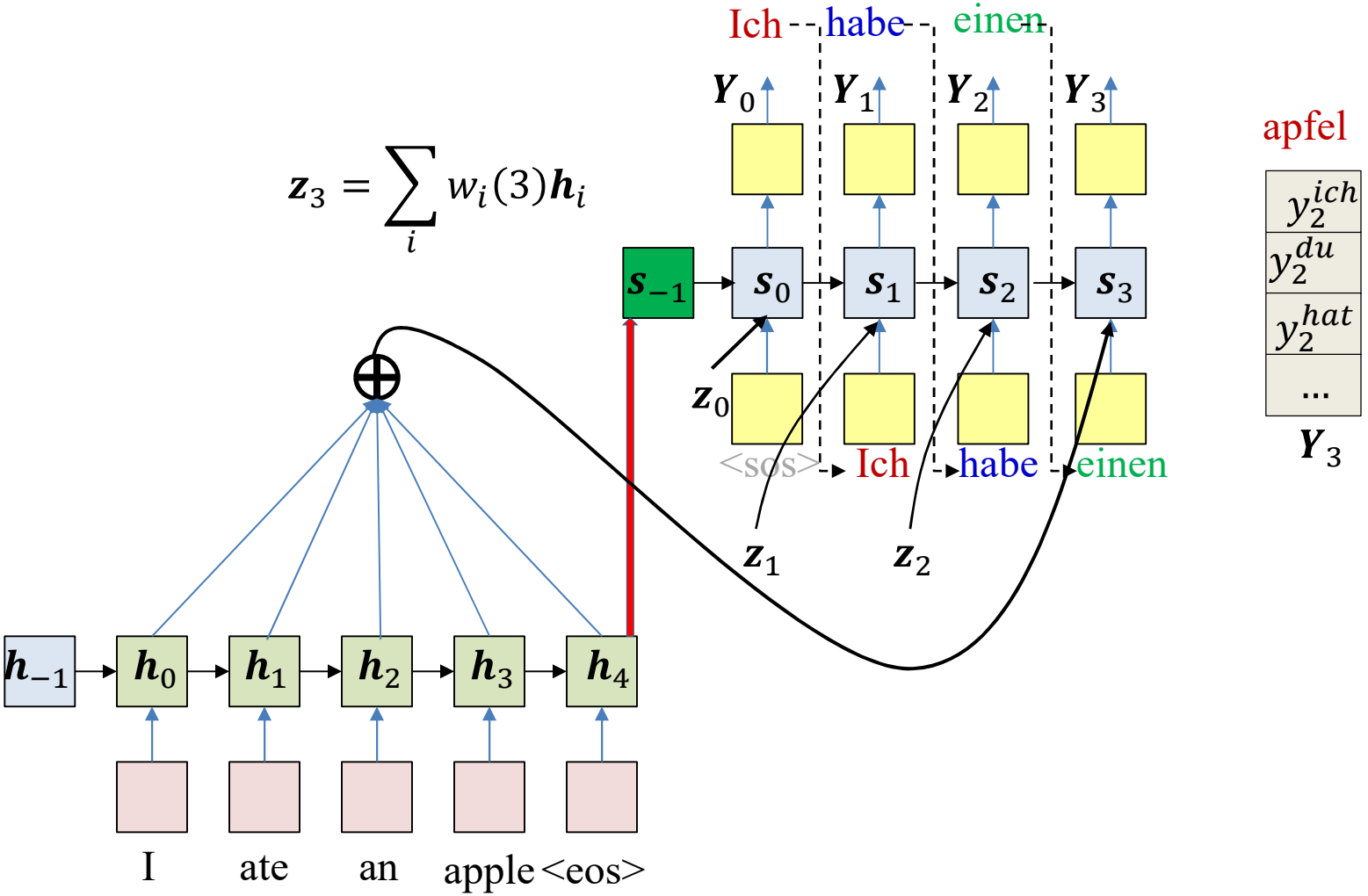
$$e_i(3) = g(\boldsymbol{h}_i, \boldsymbol{s}_2)$$

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

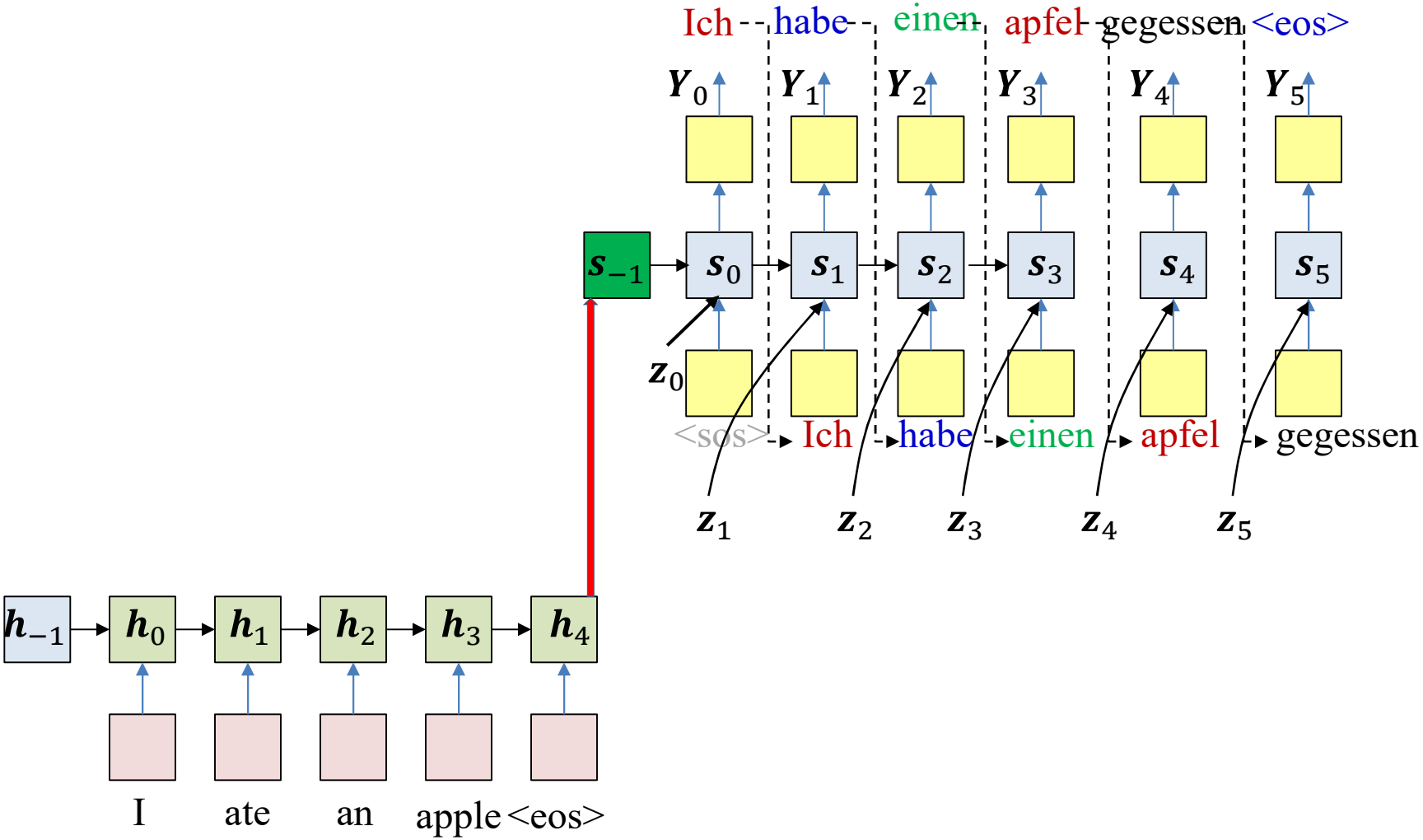
- Compute the weights for all instances for time = 3



- Compute the weighted sum of hidden input values at t=3



- Compute the output at t=3
  - Will be a probability distribution over words
  - Draw a word from the distribution



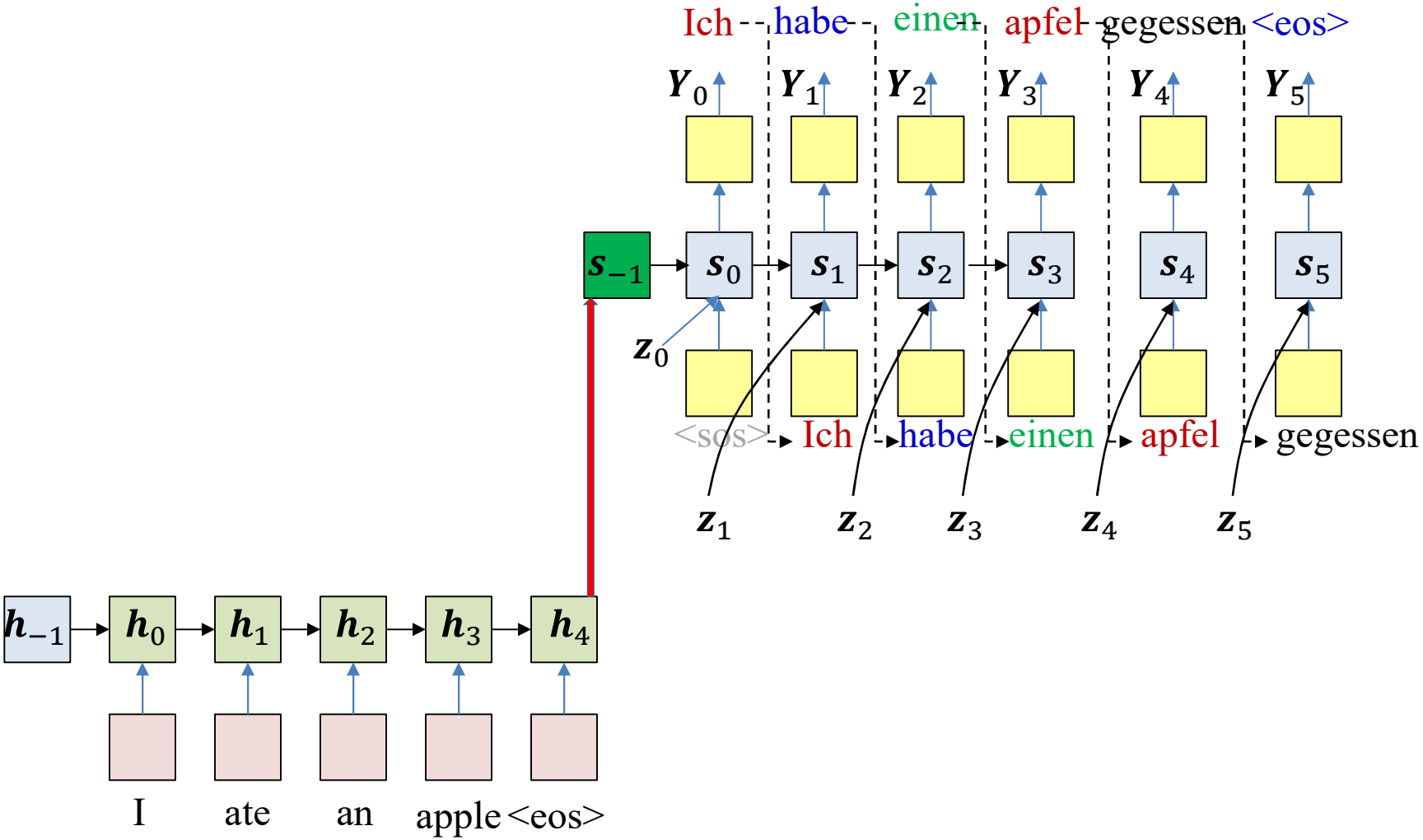
- Continue the process until an end-of-sequence symbol is produced

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
```

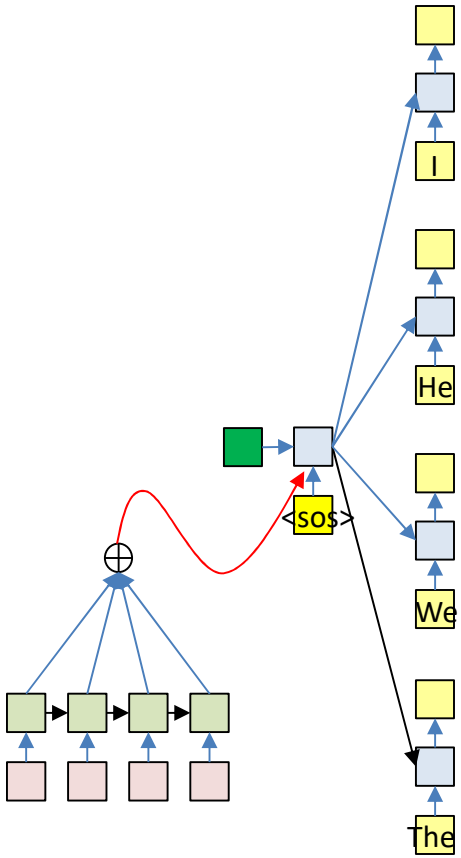


- As before, the objective of drawing: Produce the most likely output (that ends in an <eos>)

$$\underset{O_1,\ldots,O_L}{\operatorname{argmax}} y_1^{O_1} y_1^{O_2} \ldots y_1^{O_L}$$

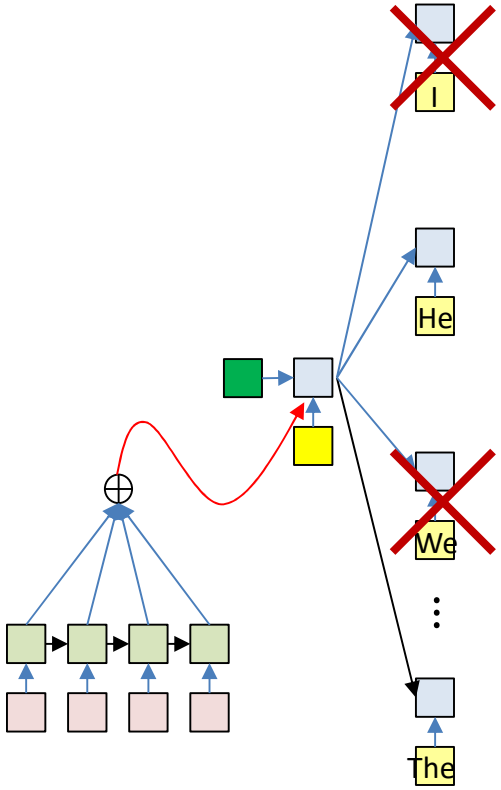
- Simply selecting the most likely symbol at each time may result in suboptimal output

Solution: Multiple choices



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

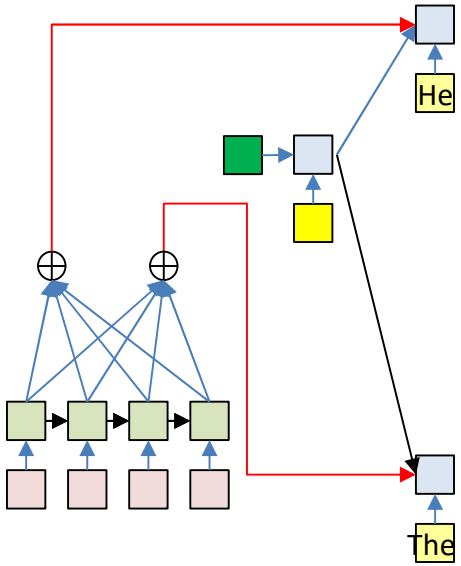
To prevent blowup: Prune



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

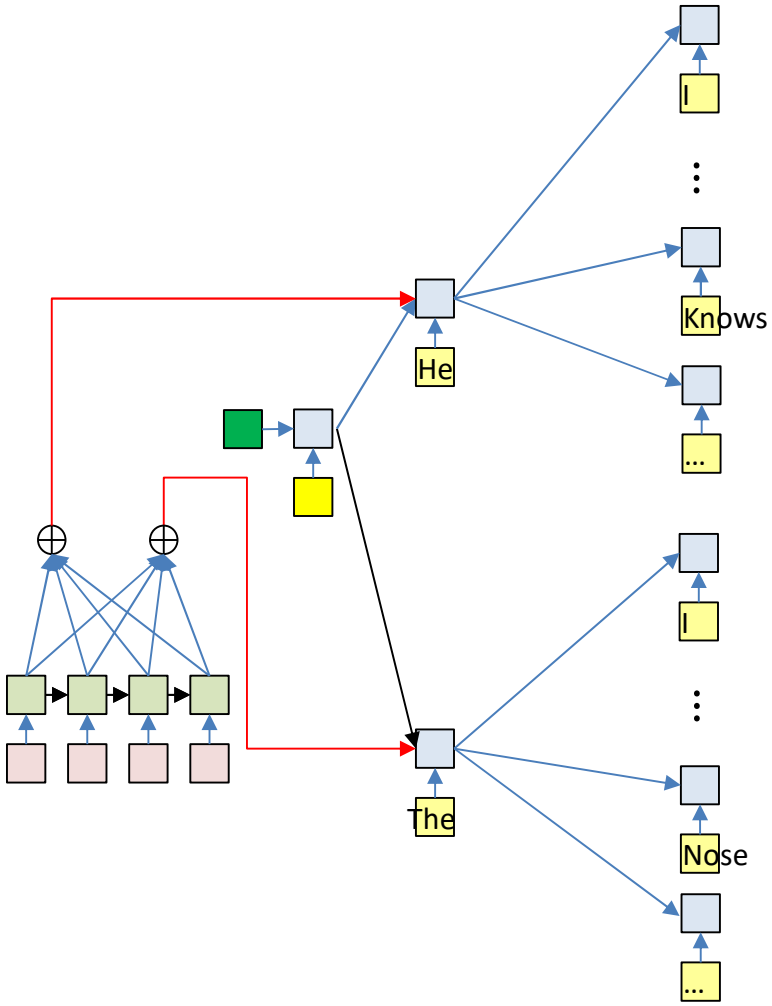
- **Prune**
  - At each time, retain only the top K scoring forks

Decoding



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

Decoding



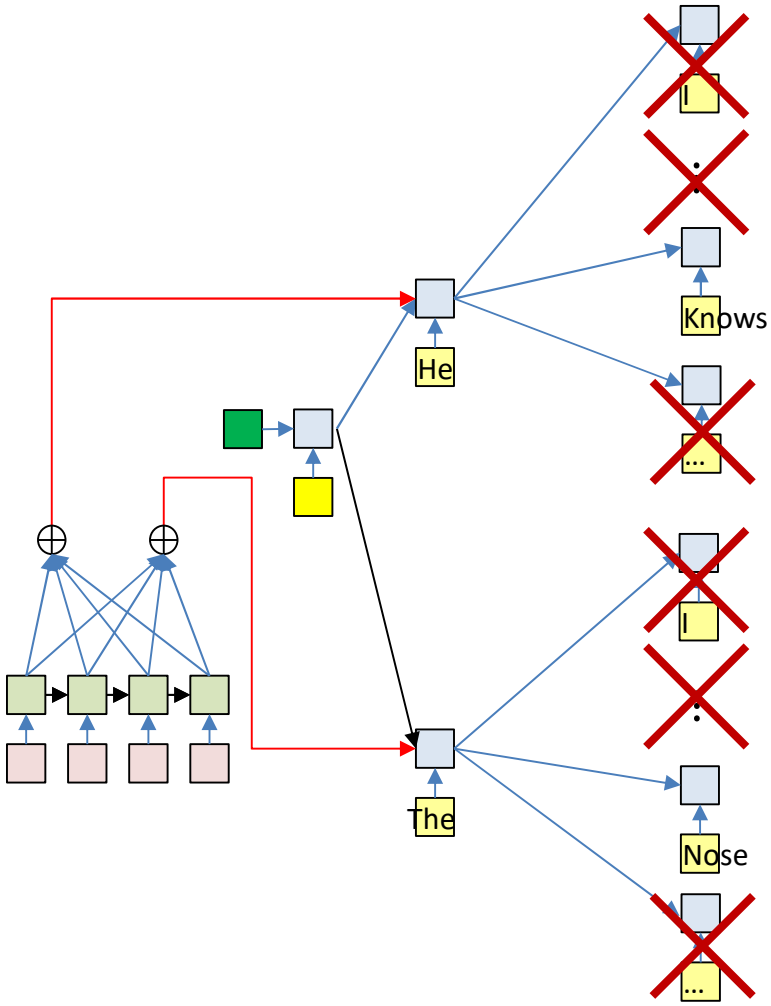
Note: based on product

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

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

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

Decoding



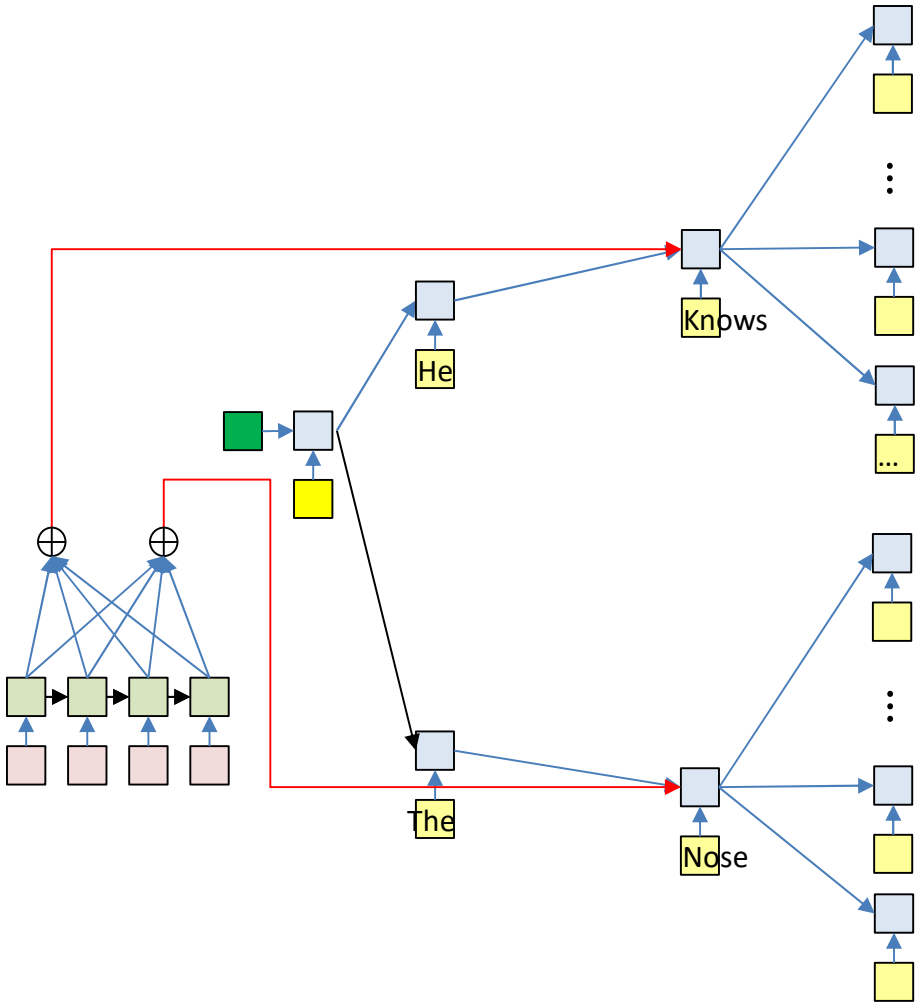
Note: based on product

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

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

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

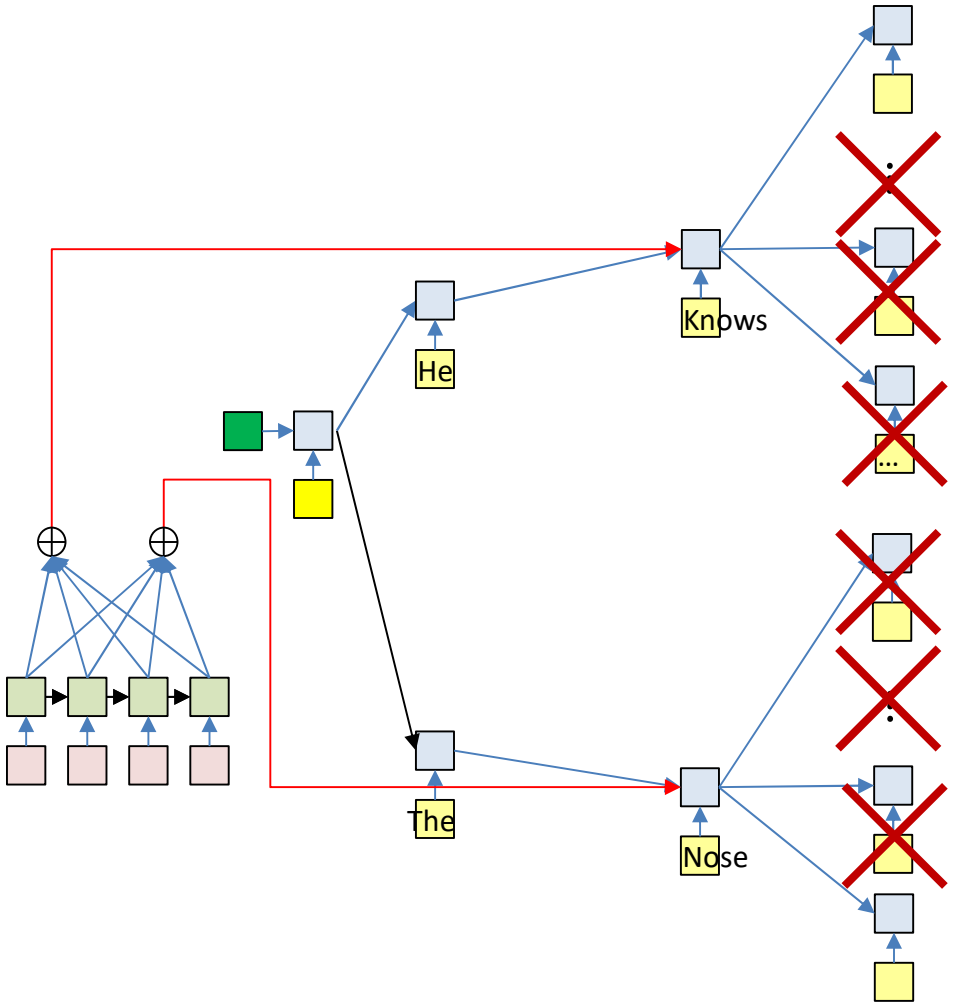
Decoding



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

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

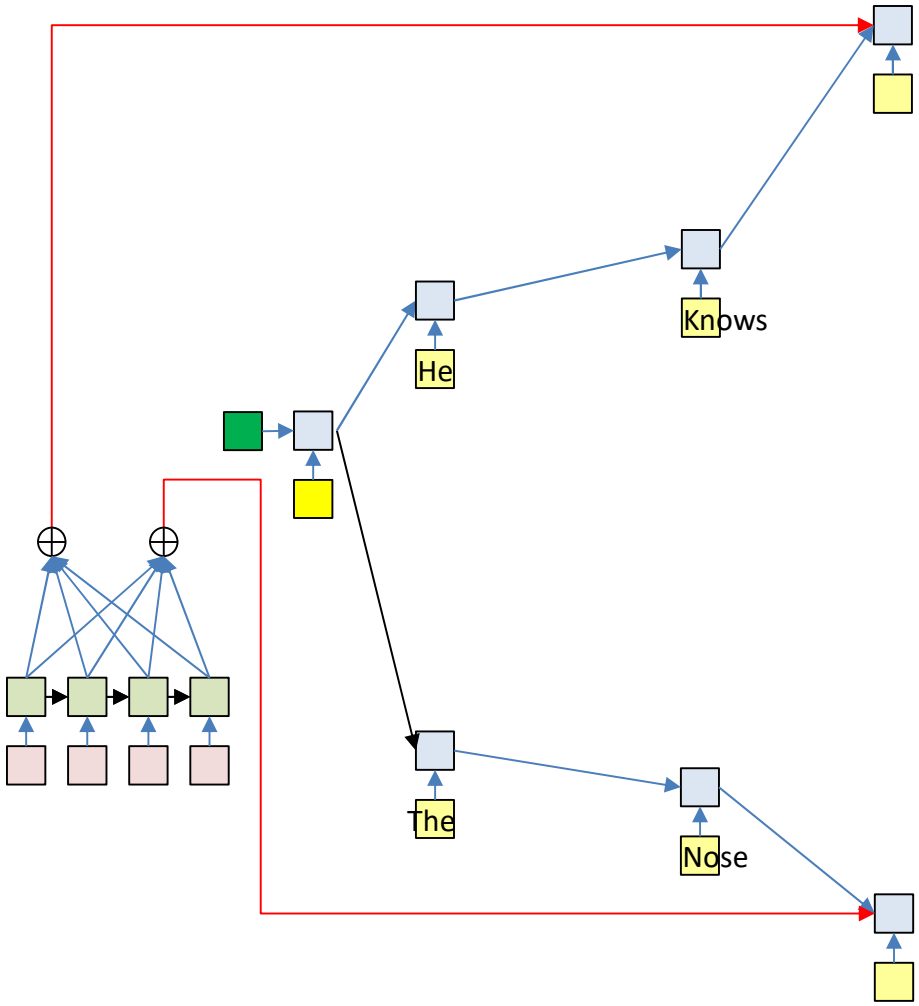
Decoding



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

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

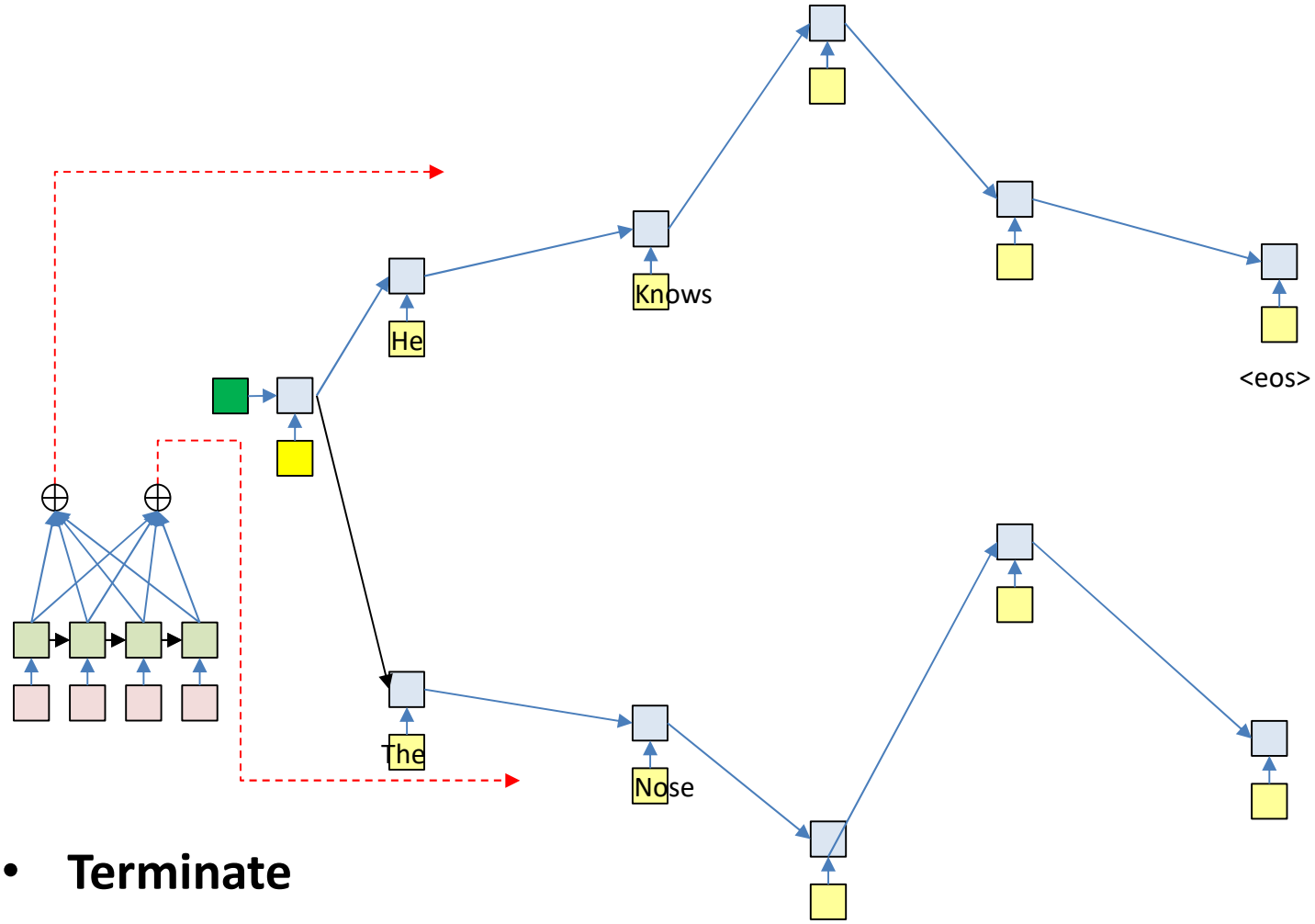
Decoding



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

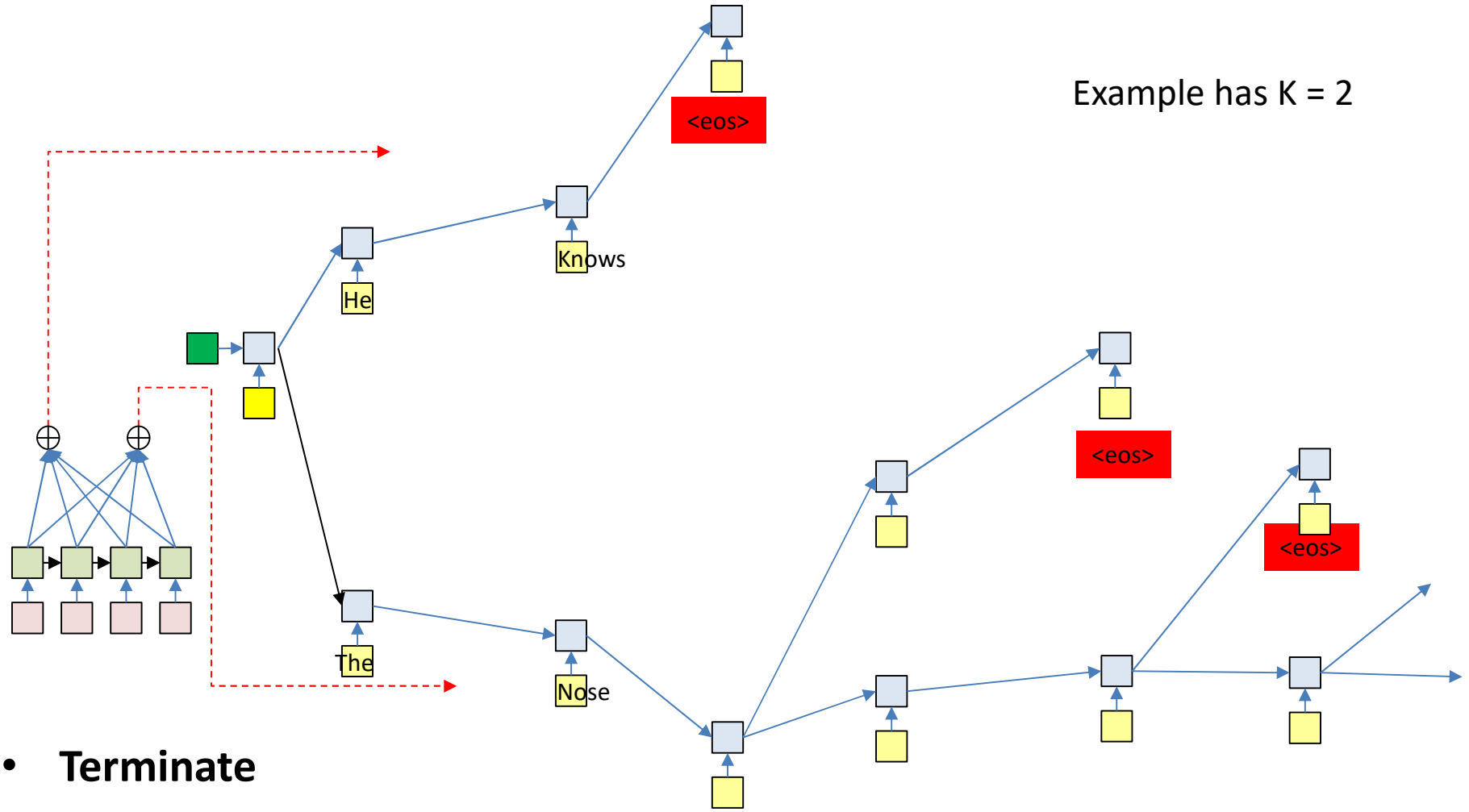
- At each time, retain only the top K scoring forks

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

Termination: <eos>



- **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

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>
```

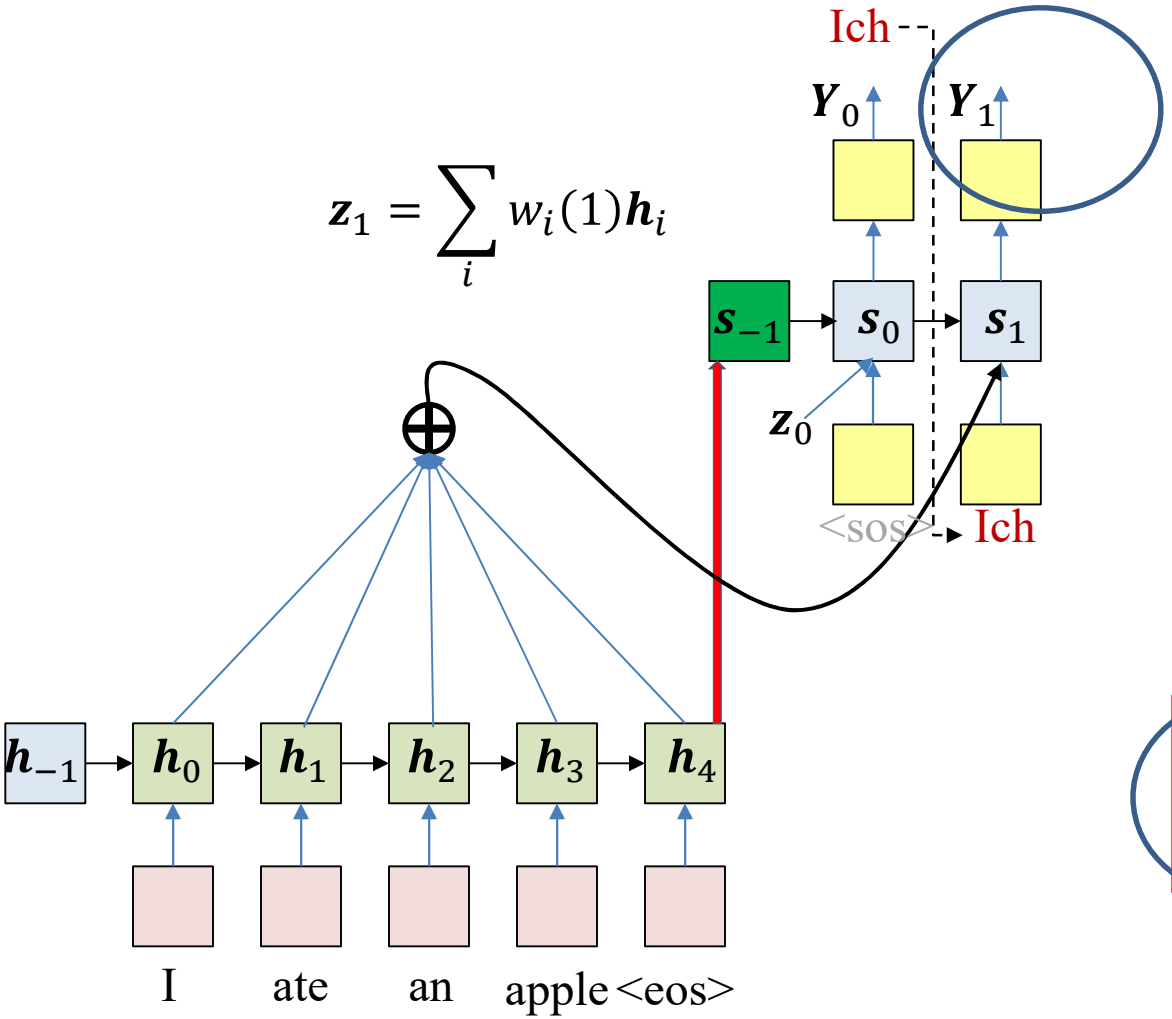
Pseudocode: Beam search

```
# 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

What does the attention learn?



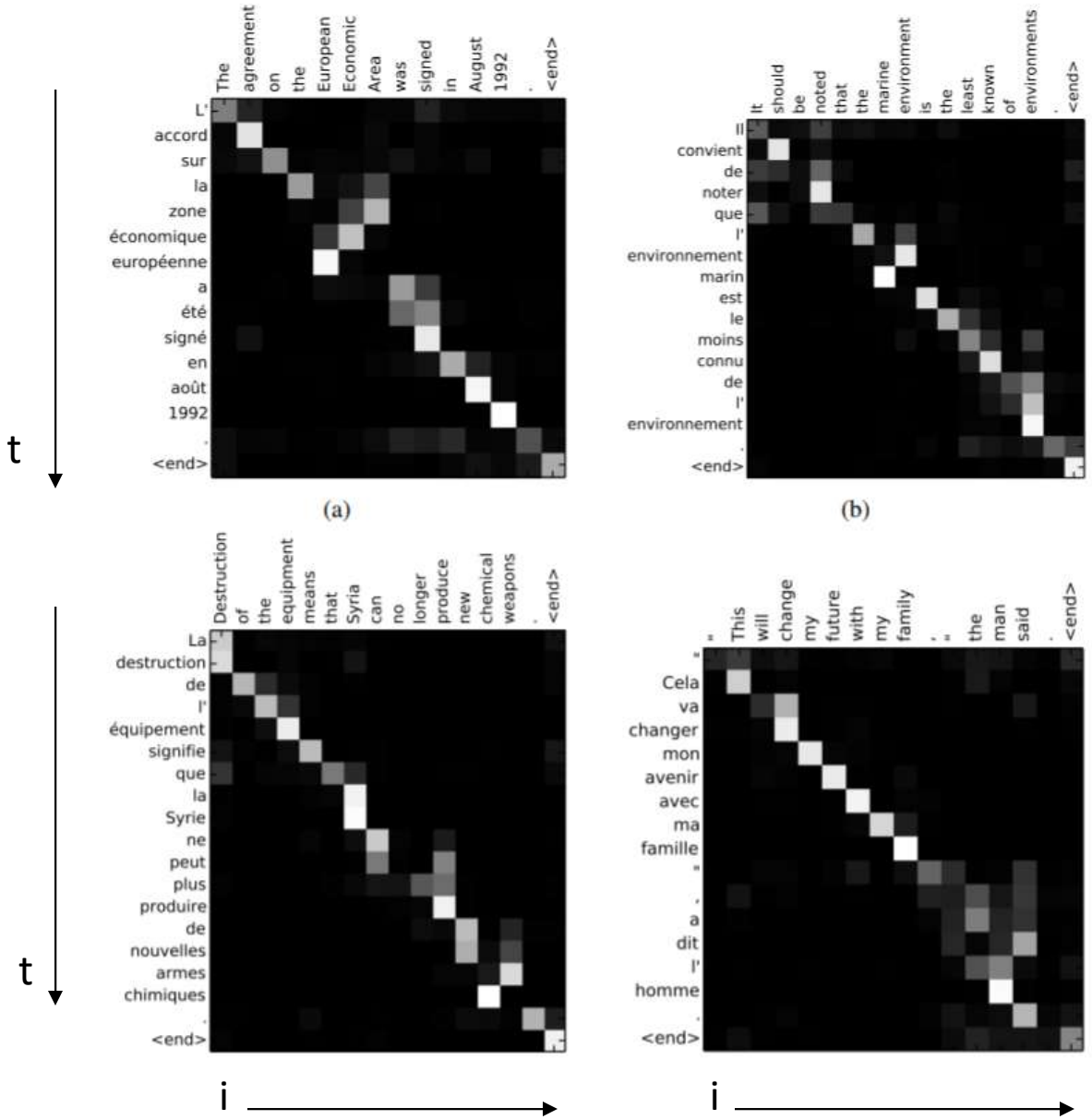
$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

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

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

- 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

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.

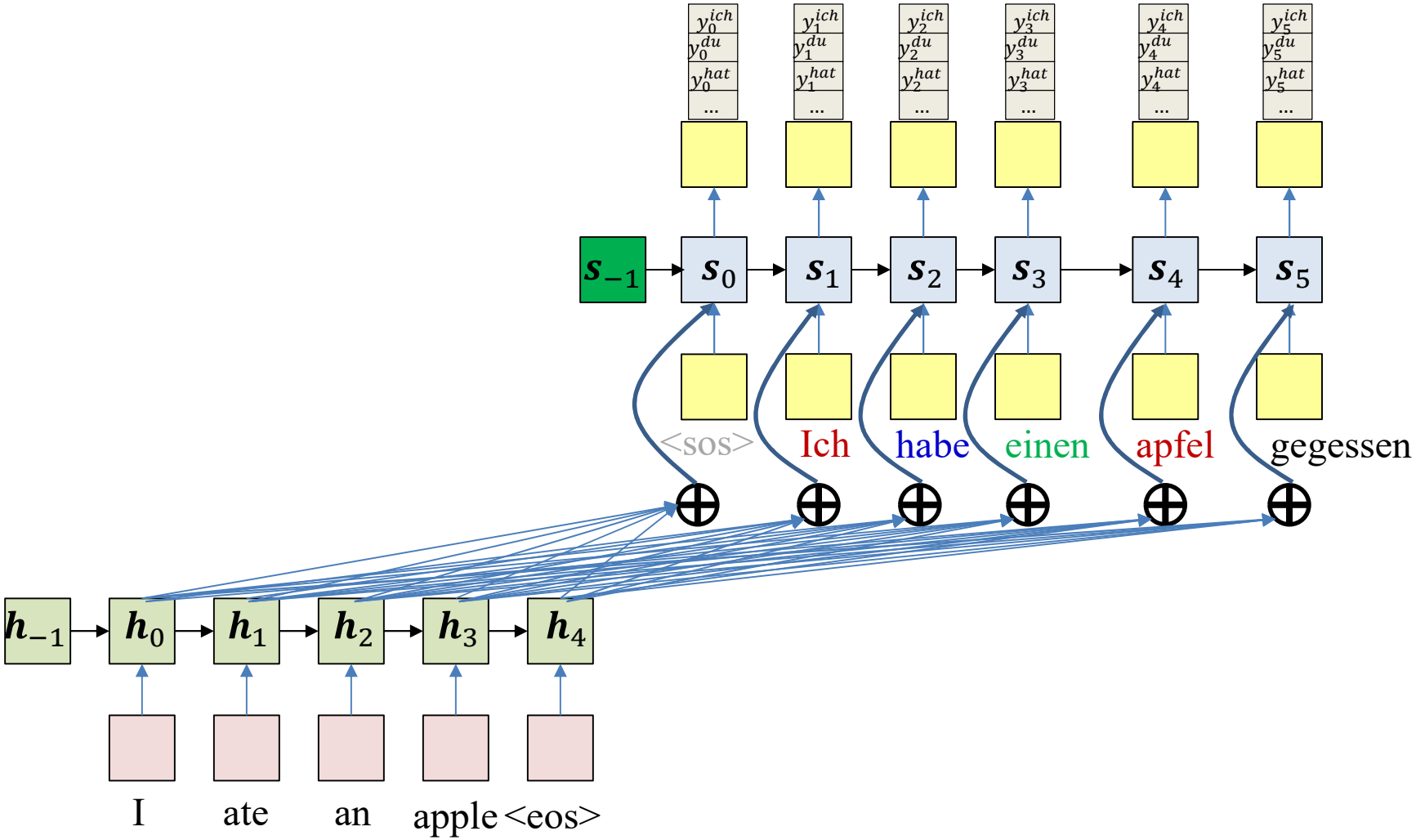
Source	This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.
Reference	Ce type d'expérience entre dans le cadre des efforts de Disney pour "étendre la 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

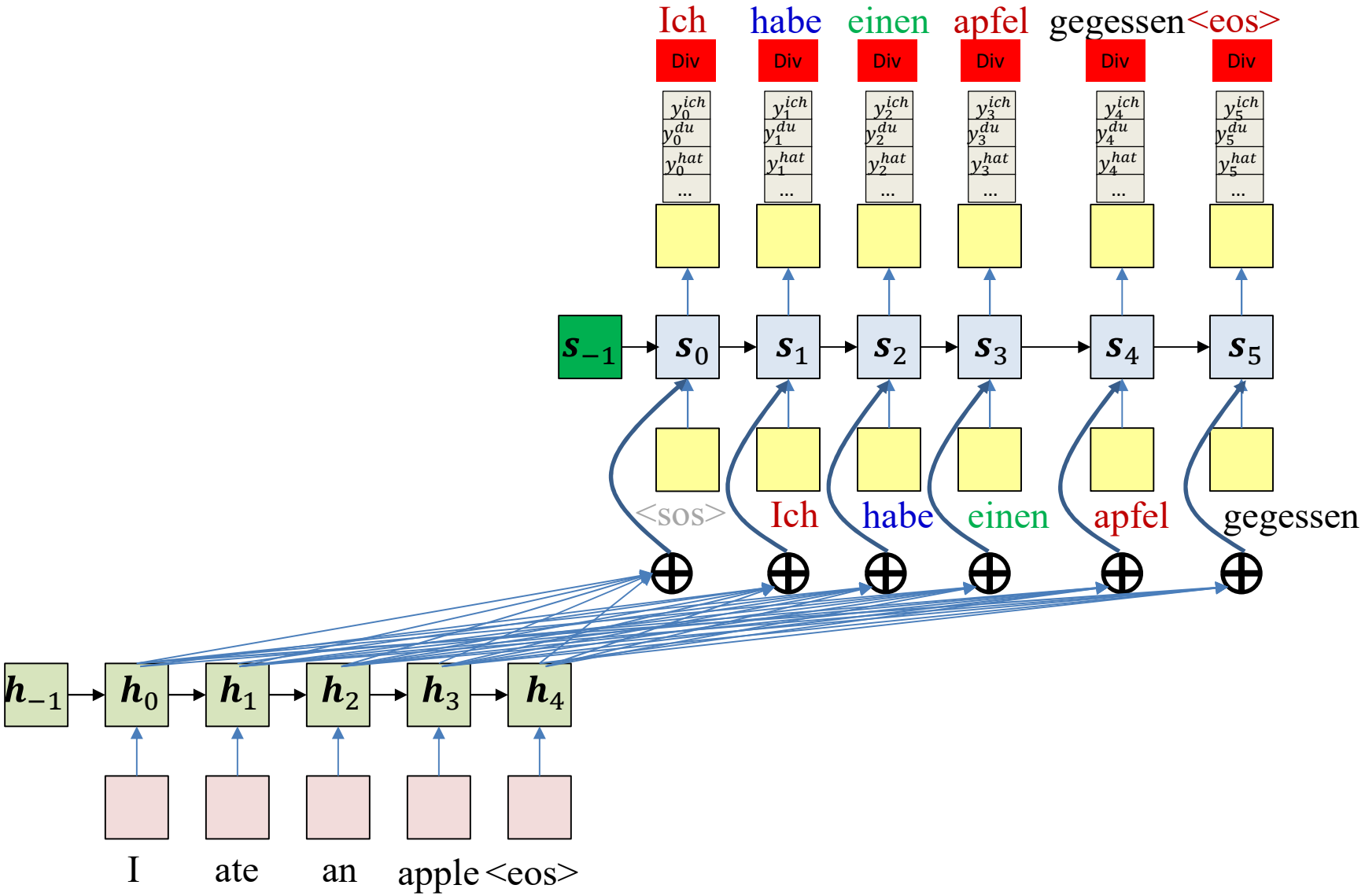
Training the network

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

- Lets consider training..

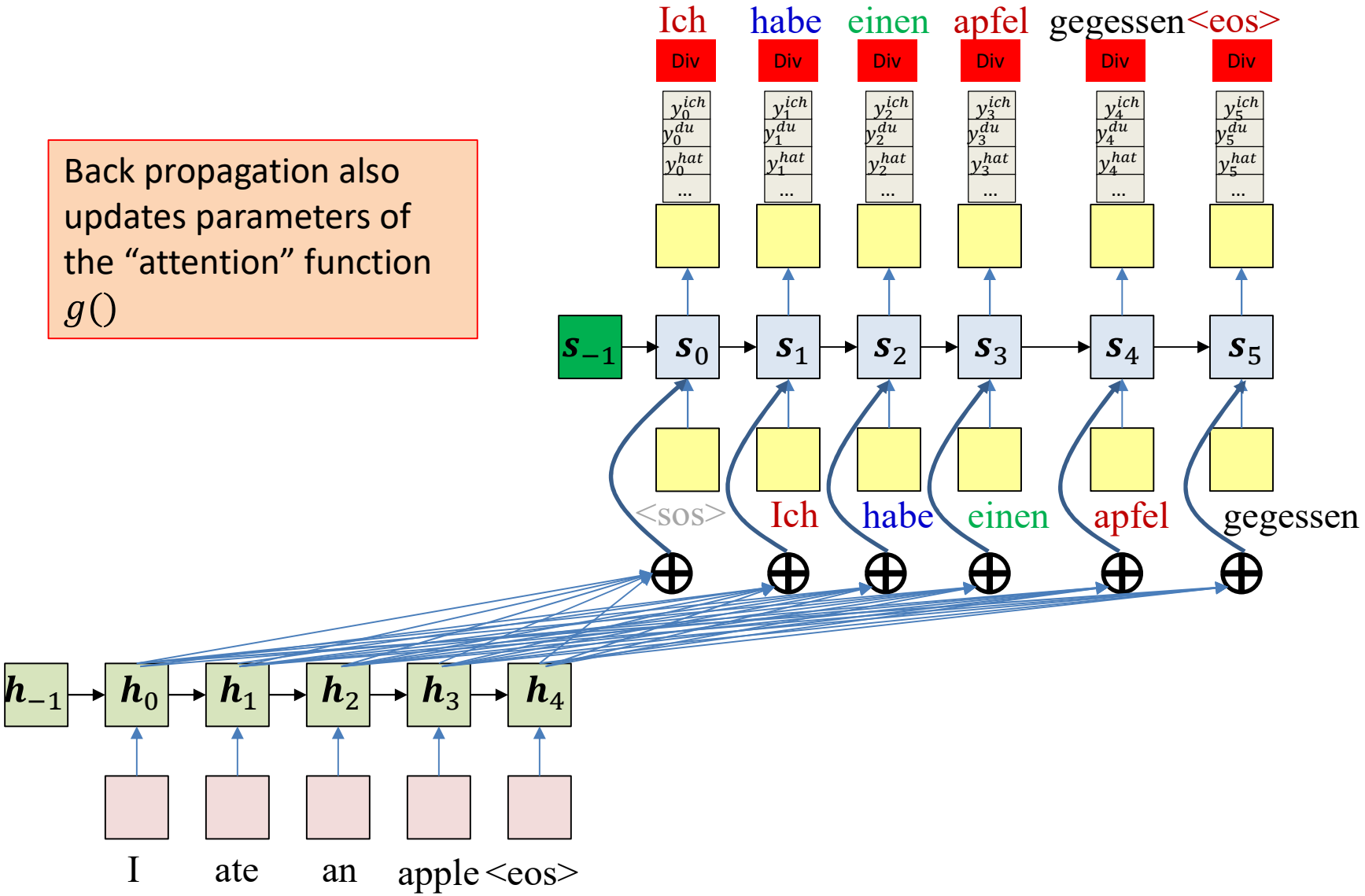


- Given training input (source sequence, target sequence) pairs
- **Forward pass**: Pass the actual Pass the input sequence through the encoder
  - At each time the output is a probability distribution over words



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

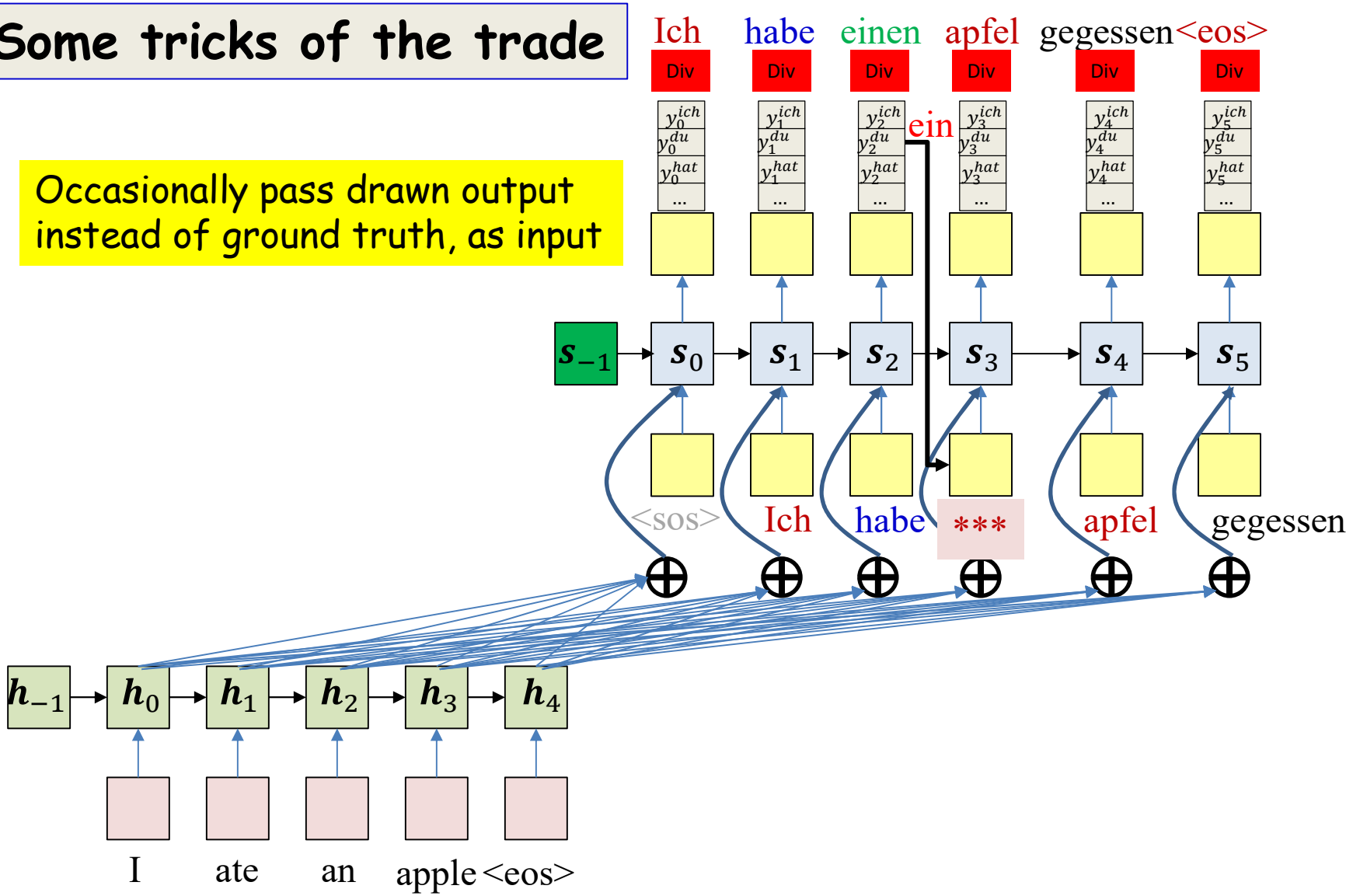
Back propagation also updates parameters of the "attention" function $g()$



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

Some tricks of the trade

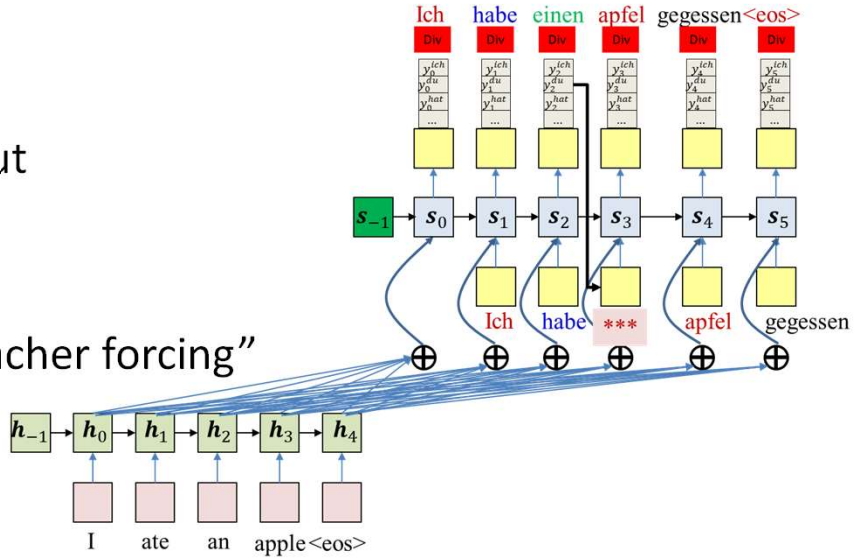
Occasionally pass drawn output instead of ground truth, as input



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

Tricks of the trade…

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



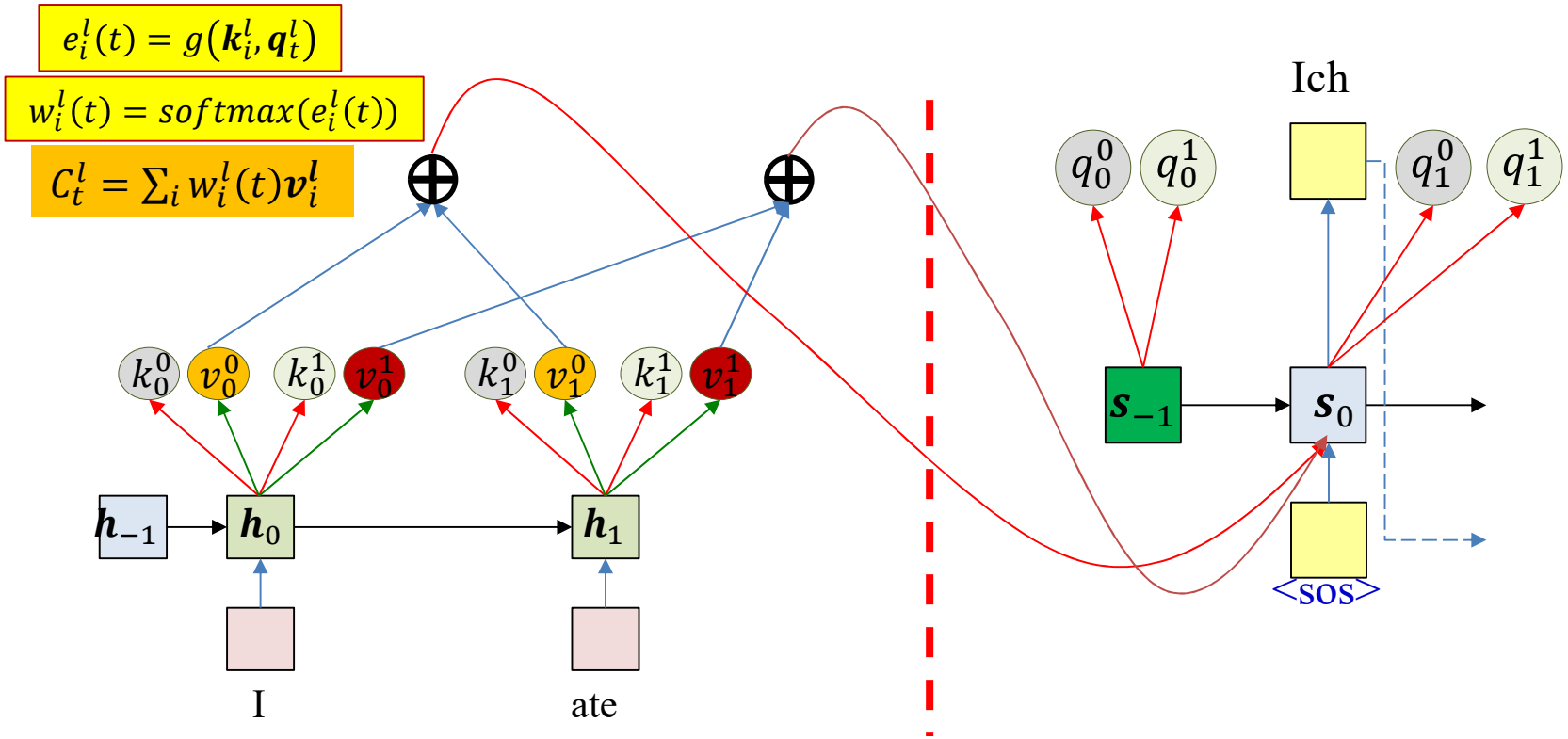
- Sampling the output:
  - Sample the system output and
  - as input during training for only some of the time

- 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

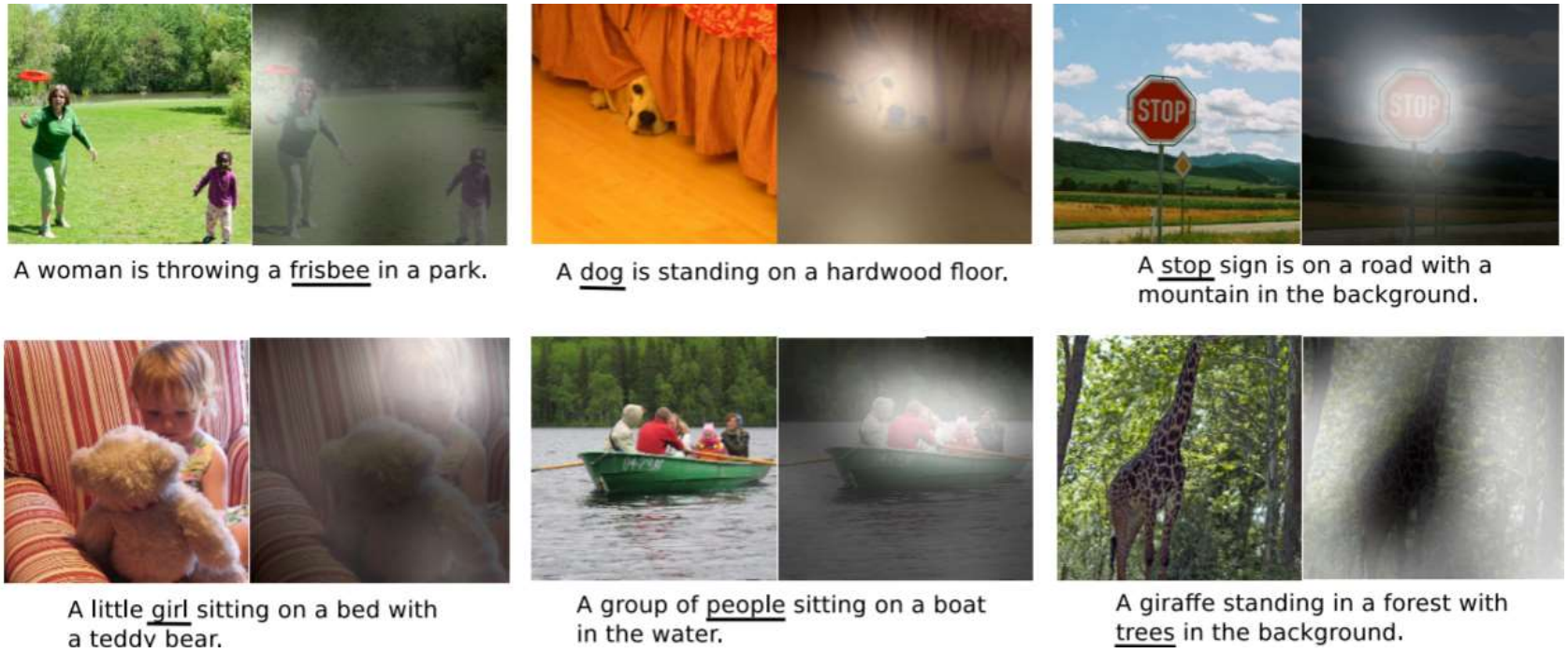


- Have multiple query/key/value sets.
  - Each attention "head" uses one of these sets
  - The combined contexts from all heads are passed to the decoder
- Each "attender" focuses on a different aspect of the input that's important for the decode

Some 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



- "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

In closing

- 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**