Sequence-to-Sequence Models

CSE 849 Deep Learning
Spring 2025

Zijun Cui

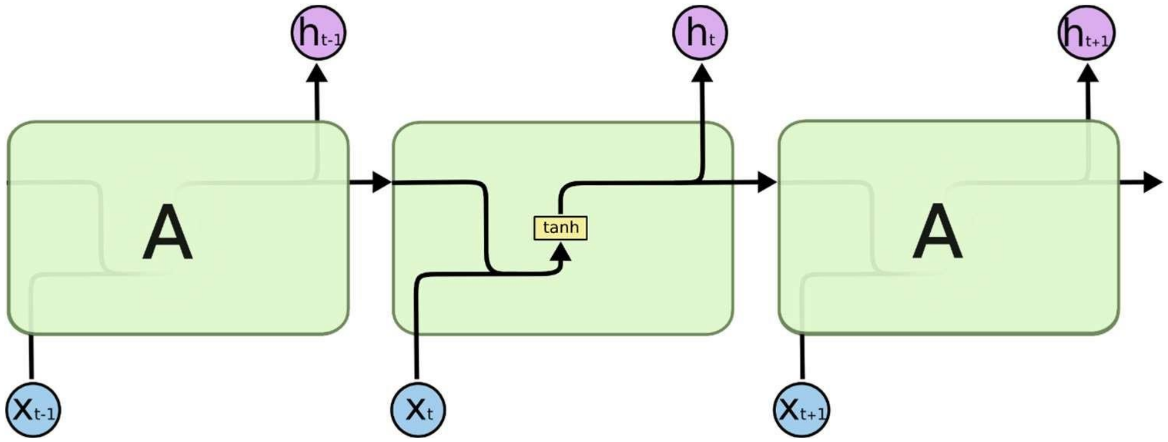
Partial slides are adapted from CMU 11-785 Introduction to Deep Learning by Bhiksha Raj

Continuing from the last lecture: Enter *LSTM*

- *Long Short-Term Memory*
- Explicitly latch information to prevent decay / blowup

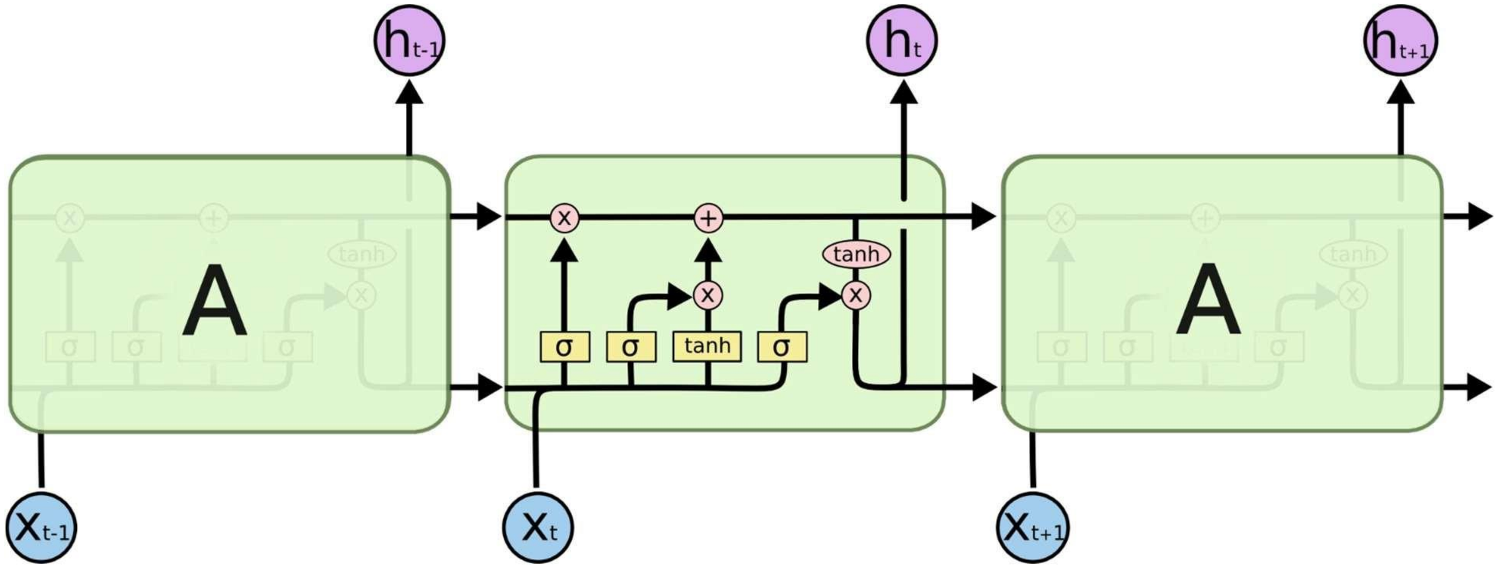
Following slides are based on http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Standard RNN



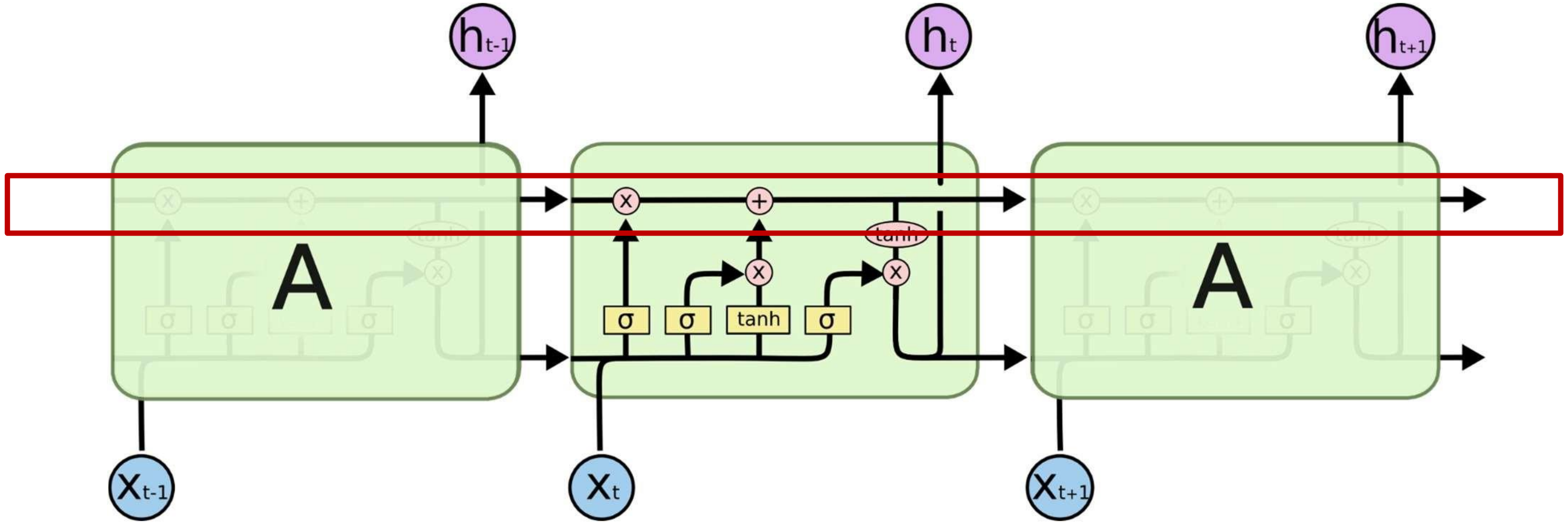
- Recurrent neurons receive past recurrent outputs and current input as inputs
- Processed through a tanh() activation function
  - As mentioned earlier, tanh() is the generally used activation for the hidden layer
- Current recurrent output passed to next higher layer and next time instant

Long Short-Term Memory



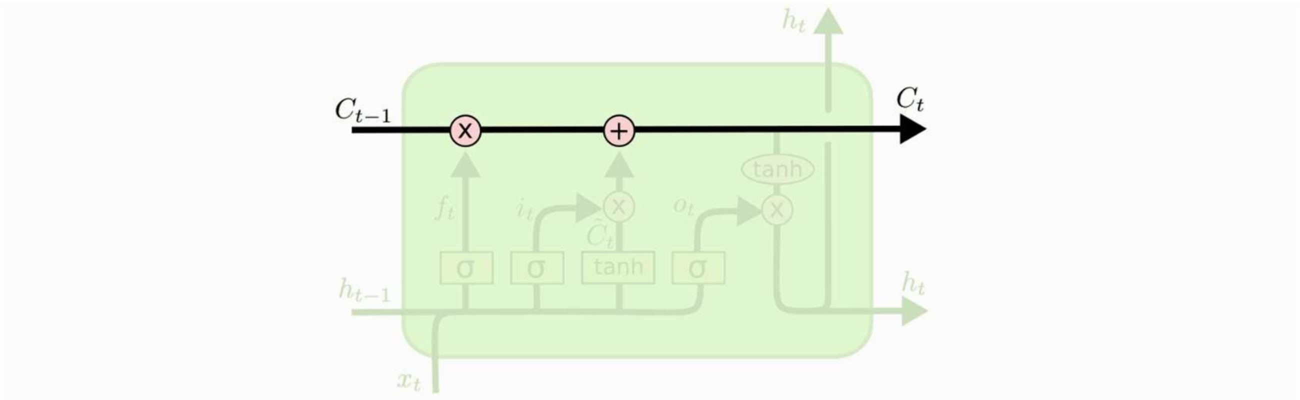
- The $\sigma()$ are *multiplicative gates* that decide if something is important or not
- Remember, every line actually represents a *vector*

LSTM: Constant Error Carousel



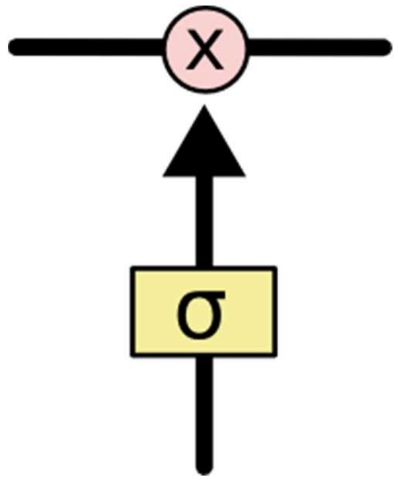
- Key component: a *remembered cell state*

LSTM: CEC



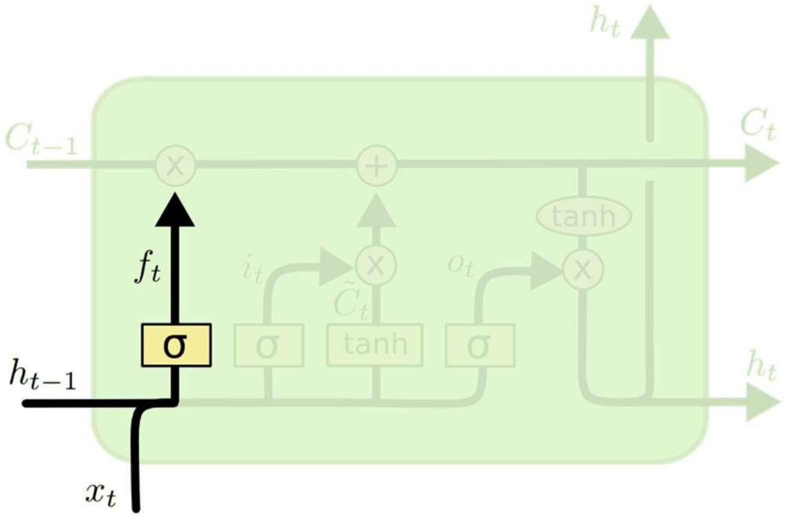
- C_t is the linear history carried by the *constant-error carousel*
- Carries information through, only affected by a gate
  - And *addition of history,* which too is gated..

LSTM: Gates



- Gates are simple sigmoidal units with outputs in the range (0,1)
- Controls how much of the information is to be let through

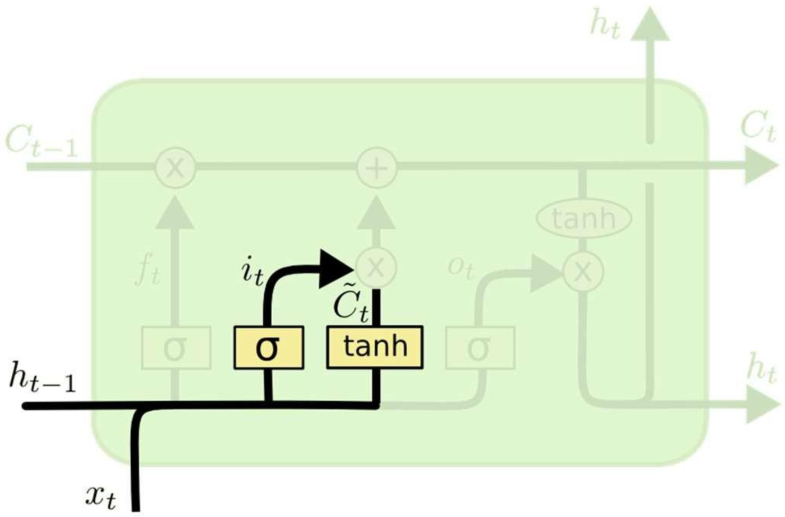
LSTM: Forget gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

- The first gate determines whether to carry over the history or to forget it
  - More precisely, how much of the history to carry over
  - Also called the "forget" gate
  - Note, we're actually distinguishing between the cell memory C and the state h that is coming over time! They're related though

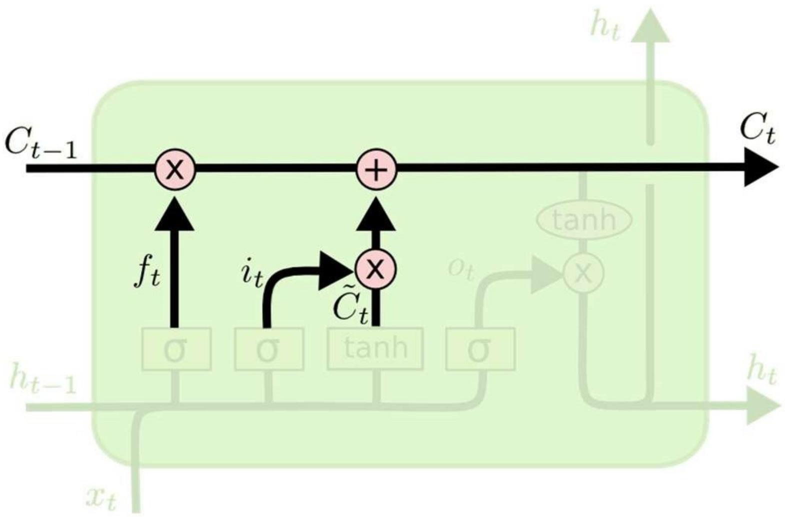
LSTM: Input gate



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- The second input has two parts
  - A perceptron layer that determines if there’s something new and interesting in the input
  - A gate that decides if its worth remembering

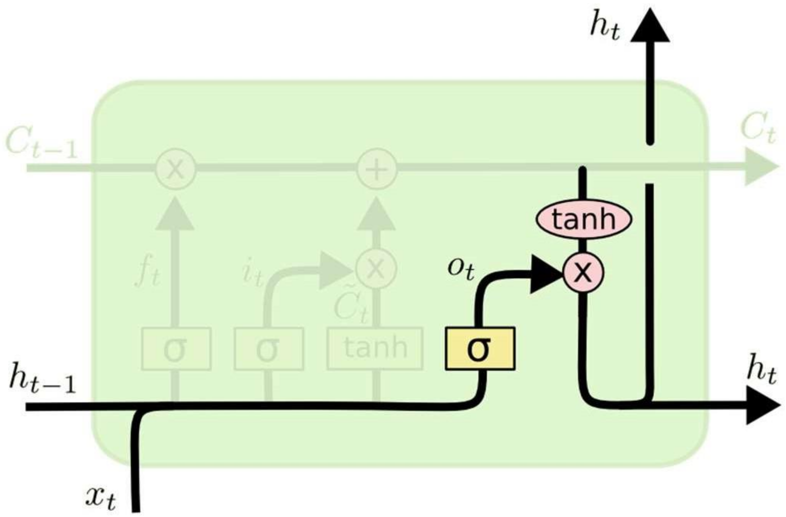
LSTM: Memory cell update



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- The second input has two parts
  - A perceptron layer that determines if there’s something interesting in the input
  - A gate that decides if its worth remembering
  - **If so its added to the current memory cell**

LSTM: Output and Output gate

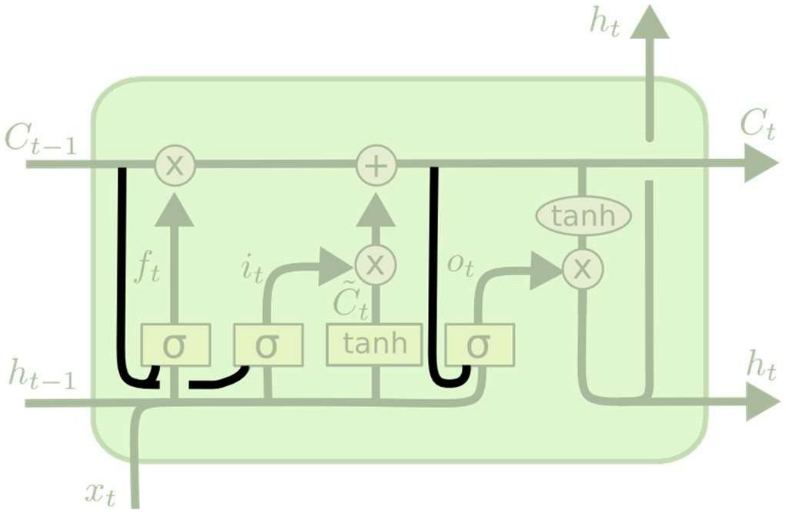


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- The *output* of the cell
  - Simply compress it with tanh to make it lie between 1 and -1
    - Note that this compression no longer affects our ability to *carry* memory forward
  - Controlled by an *output* gate
    - To decide if the memory contents are worth reporting at *this* time

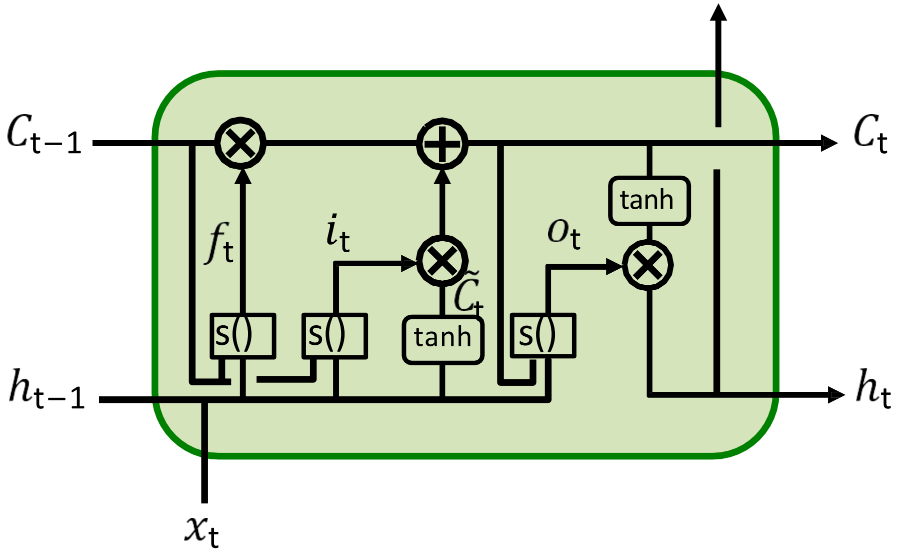
LSTM: The "Peephole" Connection



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

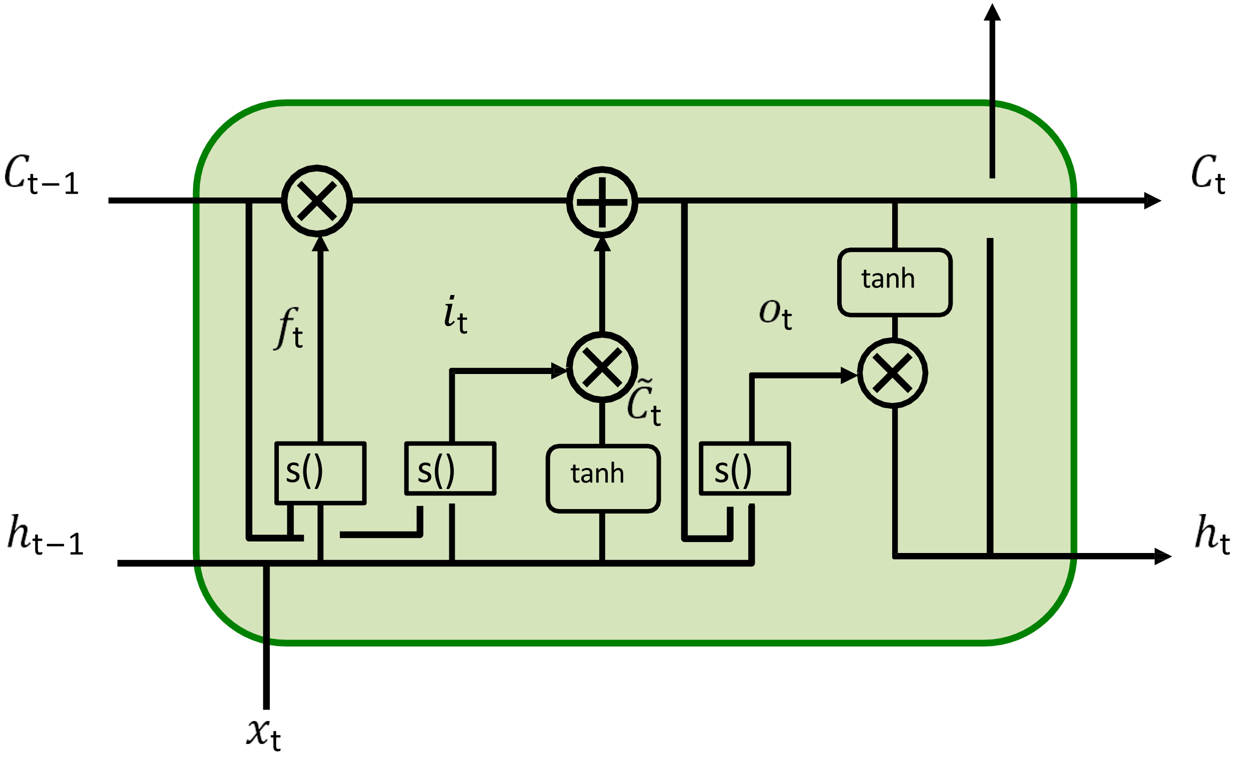
- The raw memory is informative by itself and can also be input
  - Note, we're using both C and h

The complete LSTM unit



- With input, output, and forget gates and the peephole connection..

LSTM computation: Forward



- Forward rules:

Gates

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$

$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

Variables

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM cell (single unit) Definitions

```
# Input:
#    C : previous value of CEC
#    h : previous hidden state value ("output" of cell)
#     x: Current input
# [W,b]: The set of all model parameters for the cell
#        These include all weights and biases
# Output
#    C : Next value of CEC
#    h : Next value of h
# In the function:  sigmoid(x) = 1/(1+exp(-x))
#                   performed component-wise

# Static local variables to the cell
static local zf, zi, zc, zo, f, i, o, Ci
function [C,h] = LSTM_cell.forward(C,h,x,[W,b])
    code on next slide
```

LSTM cell forward

```
# Continuing from previous slide
# Note: [W,h] is a set of parameters, whose individual elements are
#       shown in red within the code.  These are passed in

# Static local variables which aren't required outside this cell
static local zf, zi, zc, zo, f, i, o, Ci
function [Co, ho] = LSTM_cell.forward(C,h,x, [W,b])
    zf = WfcC + Wfhh + Wfxx + bf
    f = sigmoid(zf) # forget gate

    zi = WicC + Wihh + Wixx + bi
    i = sigmoid(zi) # input gate

    zc = WccC + Wchh + Wcxx + bc
    Ci = tanh(zc)  # Detecting input pattern

    Co = f∘C + i∘Ci # "∘" is component-wise multiply

    zo = WocCo + Wohh + Woxx + bo
    o = sigmoid(zo) # output gate

    ho = o∘tanh(Co) # "∘" is component-wise multiply

    return Co,ho
```

Assuming a peephole connection into the tanh

LSTM network forward

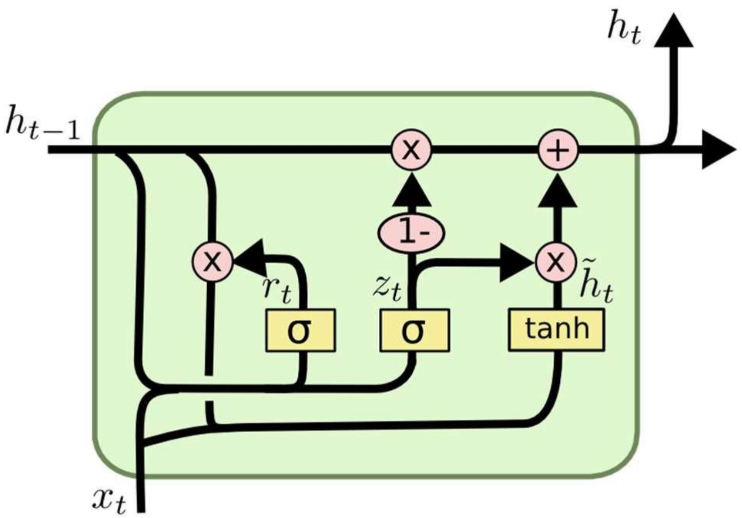
```
# Assuming h(-1,*) is known and C(-1,*)=0
# Assuming L hidden-state layers and an output layer
# Note: LSTM_cell is an indexed class with functions
# [W{l},b{l}] are the entire set of weights and biases
#                for the lth hidden layer
# Wo and bo are output layer weights and biases

for t = 0:T-1  # Including both ends of the index
    h(t,0) = x(t) # Vectors. Initialize h(0) to input
    for l = 1:L  # hidden layers operate at time t
        [C(t,l),h(t,l)] = LSTM_cell(t,l).forward(…
                …C(t-1,l),h(t-1,l),h(t,l-1)[W{l},b{l}])
    zo(t) = Woh(t,L) + bo
    Y(t) = softmax( zo(t) )
```

Training the LSTM

- Identical to training regular RNNs with one difference
  - Commonality: Define a sequence divergence and backpropagate its derivative through time

- Difference: Instead of backpropagating gradients through an RNN unit, we will backpropagate through an LSTM cell

Gated Recurrent Units: Let's simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

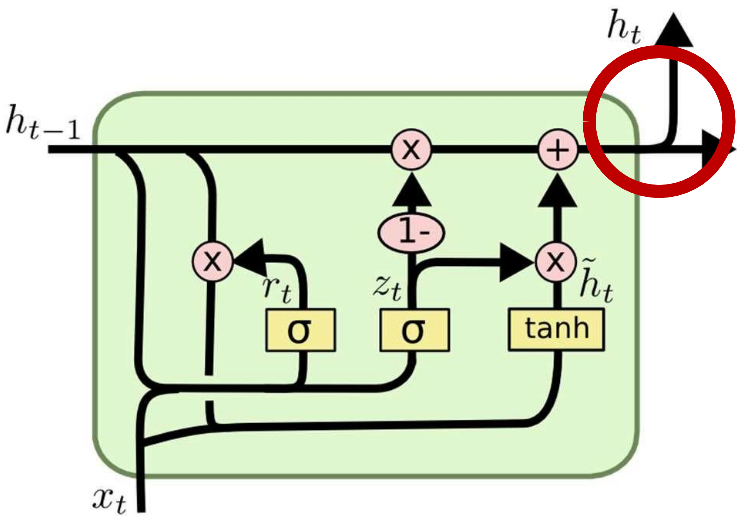
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Simplified LSTM which addresses some of your concerns of *why*

Gated Recurrent Units: Lets simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

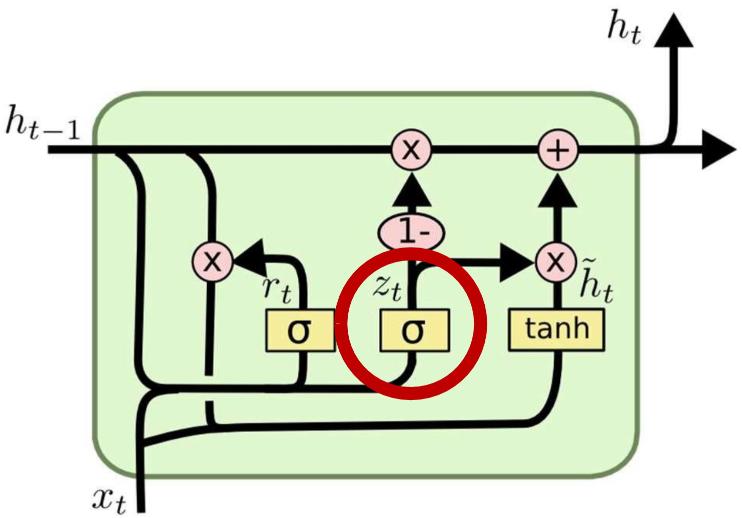
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Don’t bother to separately maintain compressed and regular memories
  - Redundant representation

Gated Recurrent Units: Lets simplify the LSTM



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

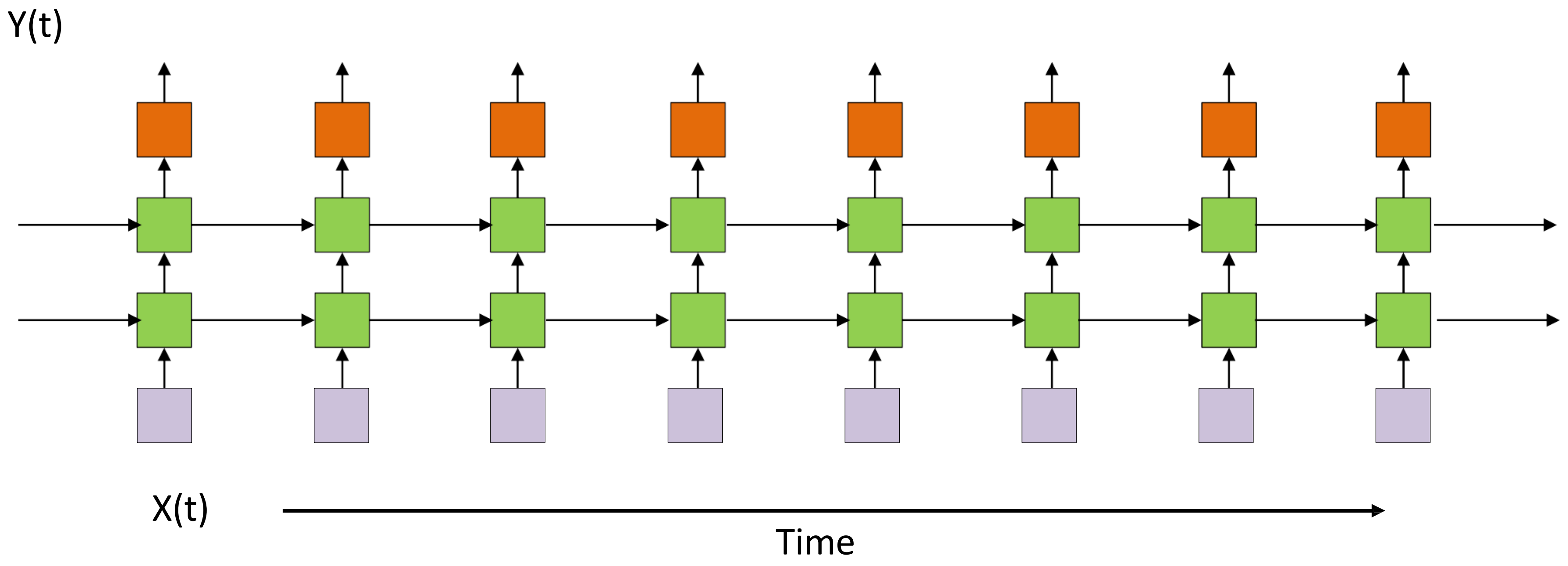
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

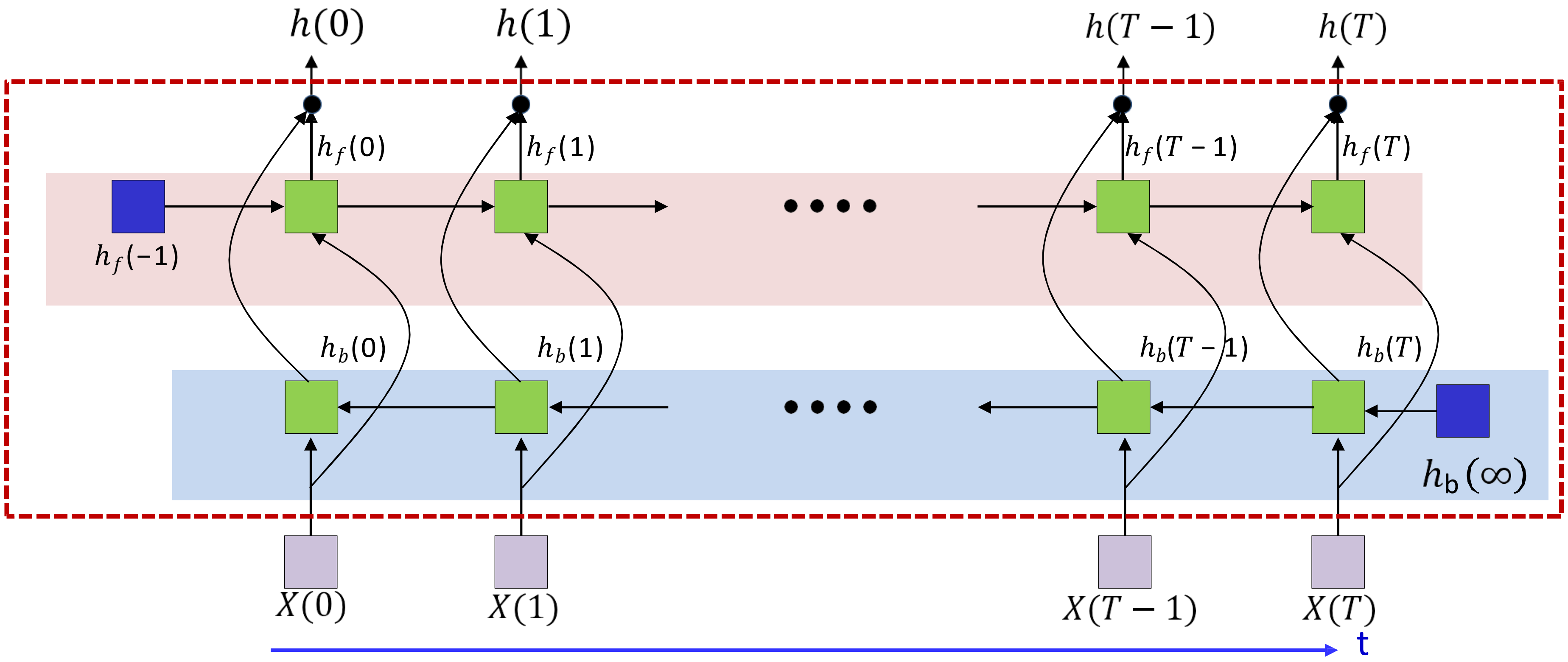
- Combine forget and input gates
  - If new input is to be remembered, then this means old memory is to be forgotten

LSTM architectures example



- Each green box is now a (layer of) LSTM or GRU cell(s)
  - Keep in mind each box is an *array* of units
  - For LSTMs the horizontal arrows carry both $C(t)$ and $h(t)$

Bidirectional LSTM



- Like the BRNN, but now the hidden nodes are LSTM units.
  - Or layers of LSTM units

Sequence-to-sequence modelling

- Problem:
  - A sequence $X_1 \dots X_N$ goes in
  - A different sequence $Y_1 \dots Y_M$ 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 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.”

Brief detour: Language models

- Modelling language using recurrent nets
- More generally language models and embeddings..

Language Models

- LMs model the probability distribution of token sequences in the language
  - Word sequences, if words are the tokens

- Can be used to
  - Compute the probability of a given token sequence
  - Generate sequences from the distribution of the language

Language Models

$$P(w_1\ w_2\ w_3\ w_4\ \ldots.) = P(w_1)\ P(w_2|w_1)$$
$$P(w_3|w_1\ w_2)\ P(w_4|w_1\ w_2\ w_3)\ldots$$

- The actual target is to model the probabilities of entire word sequences
- However, we typically use chain rule to compute this incrementally
  - Language models generally perform *next symbol prediction*
  - Always predicting the next symbol, given all previous symbols
- However, never forget, they *actually* model the probability of entire sequences
  - Sentences, paragraphs, books
  - They model *language*

Language modelling through next-word prediction 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

Language modelling: Representing words

- Represent words as one-hot vectors
  - Pre-specify a vocabulary of N words in fixed (e.g. lexical) order
    - *E.g. [A AARDVARK AARON ABACK ABACUS… ZZYP]*
  - Represent each word by an N-dimensional vector with N-1 zeros and a single 1 (in the position of the word in the ordered list of words)
    - E.g. "AARDVARK" - > [0 1 0 0 0 …]
    - E.g. "AARON" - > [0 0 1 0 0 0 …]

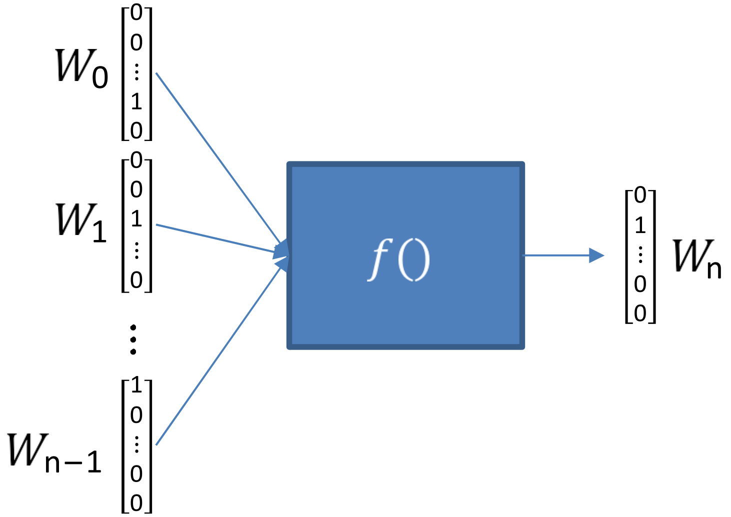
- Characters can be similarly represented
  - English will require about 100 characters, to include both cases, special characters such as commas, hyphens, apostrophes, etc., and the space character

Predicting words

Four score and seven years ???

$$W_n = f(W_0, \ldots, W_{n-1})$$

Nx1 one-hot vectors

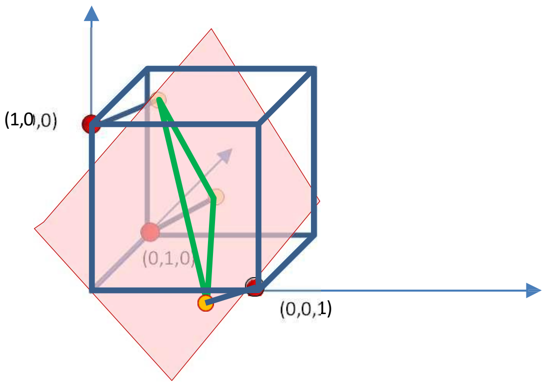


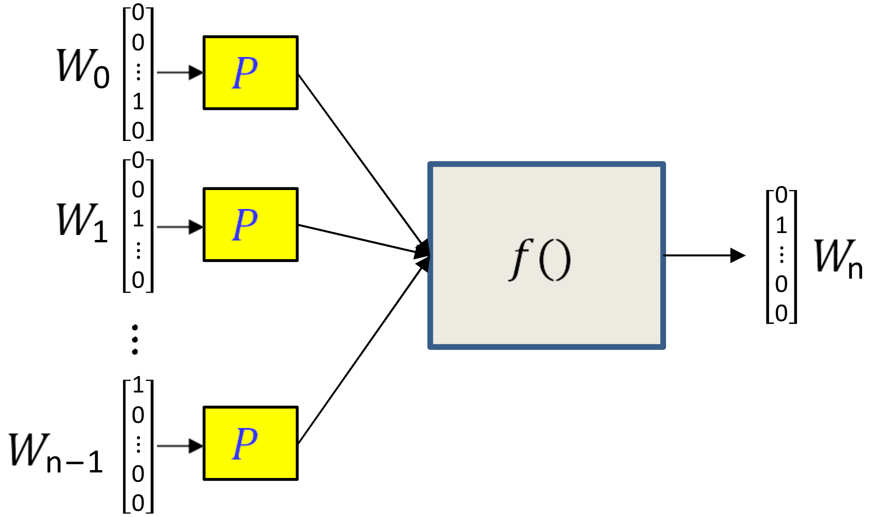
- Given one-hot representations of $W_0 \ldots W_{n-1}$, predict W_n
- **Dimensionality problem:** All inputs $W_0 \ldots W_{n-1}$ are both very high-dimensional and very sparse

The *Projected* word vectors

Four score and seven years ???

$$W_n = f(PW_0, PW_2, \ldots, PW_{n-1})$$

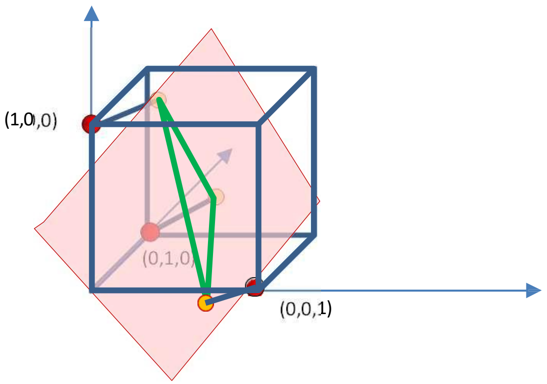


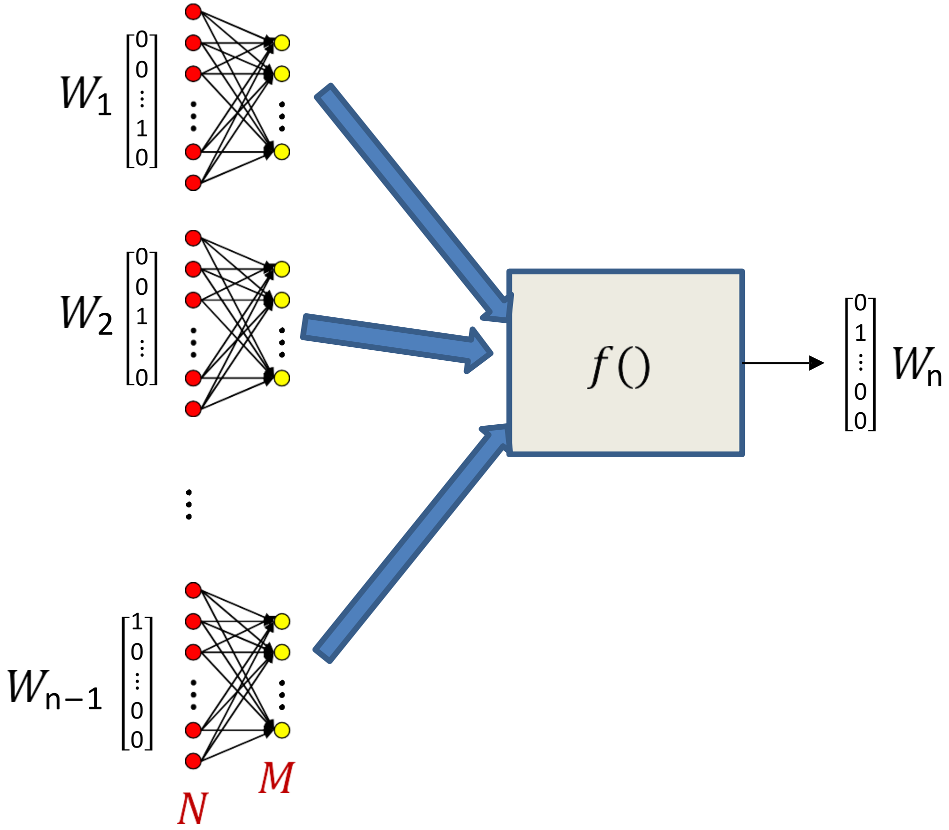


- *Project* the N-dimensional one-hot word vectors into a lower-dimensional space
  - Replace every one-hot vector W_i by PW_i
  - P is an $M \times N$ matrix
  - PW_i is now an M-dimensional vector
  - *Learn* P using an appropriate objective
    - Distances in the projected space will reflect relationships imposed by the objective

“Projection”

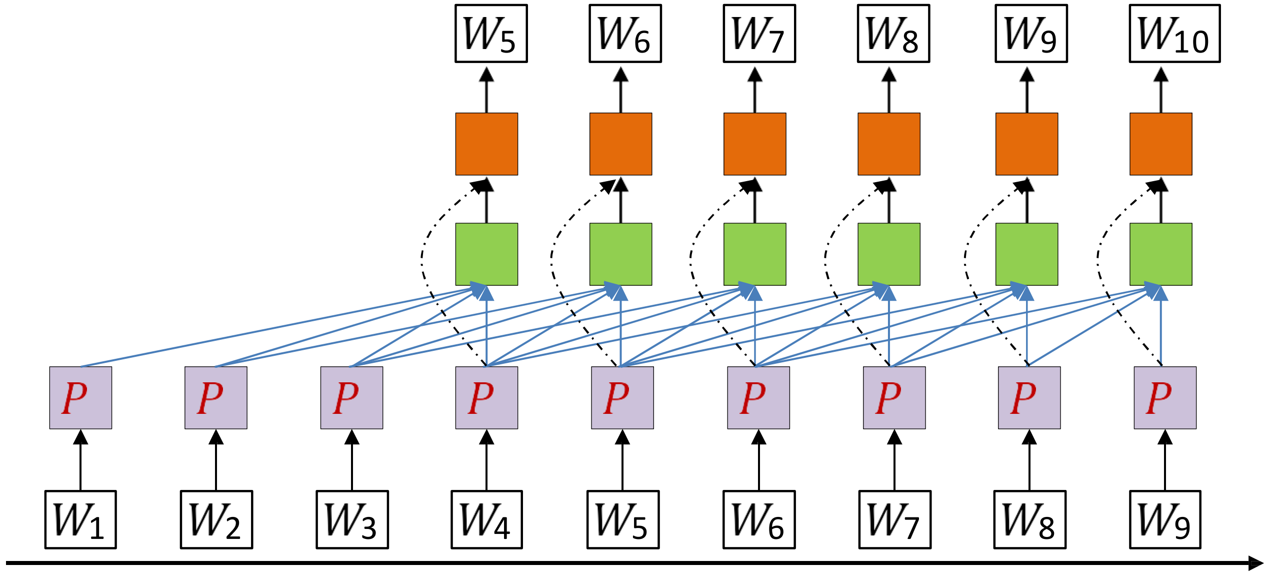
$$W_n = f(PW_1, PW_2, \ldots, PW_{n-1})$$





- P is a simple linear transform
- A single transform can be implemented as a linear layer with M outputs
  - The same linear layer applies to all inputs
  - Also viewable as a network with identical subnets (shared parameter network)

Predicting words: The TDNN model



- Predict each word based on the past N words
  - "A neural probabilistic language model", Bengio et al. 2003
  - Hidden layer has Tanh() activation, output is softmax

- One of the outcomes of learning this model is that we also learn low-dimensional representations PW of words

Returning to our problem: Sequence to sequence modelling

 → Seq2seq → I ate an apple

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

- Sequence $X_1 \ldots X_N$ goes in, sequence $Y_1 \ldots Y_M$ comes out
- Cases
  - 1 : order correspondence between input and output
    - The nth output corresponds to the nth segment of the input
  - 2 : No correspondence between input and output
    - May even not even maintain order of symbols
      - E.g. “I ate an apple” -> “Ich habe einen apfel gegessen”
    - Or may even even seem unrelated to the input
    - E.g. “My screen is blank” - > “Please check if your computer is plugged in.”

Returning to our problem: Sequence to sequence modelling

→ Seq2seq → I ate an apple

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

- Sequence $X_1 \dots X_N$ goes in, sequence $Y_1 \dots Y_M$ comes out
- Cases
  - 1 : order correspondence between input and output
    - The nth output corresponds to the nth segment of the input
  - 2 : No correspondence between input and output
    - May even not even maintain order of symbols
      - E.g. "I ate an apple" -> "Ich habe einen apfel gegessen"
    - Or may even even seem unrelated to the input
    - E.g. "My screen is blank" -> "Please check if your computer is plugged in."

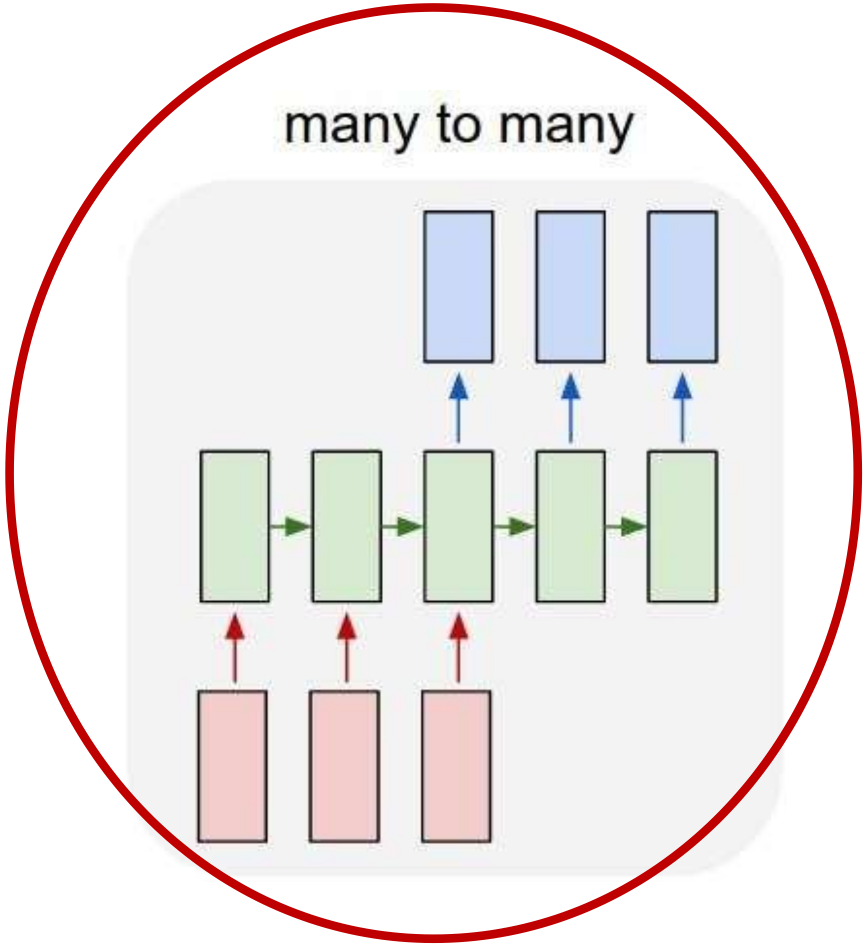
Returning to our problem: Sequence to sequence modelling

→ Seq2seq → I ate an apple

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

- Sequence $X_1 \ldots X_N$ goes in, sequence $Y_1 \ldots Y_M$ comes out
- Cases
  - 1 : order correspondence between input and output
    - The nth output corresponds to the nth segment of the input
  - 2 : No correspondence between input and output
    - May even not even maintain order of symbols
      - E.g. "I ate an apple" ->"Ich habe einen apfel gegessen"
    - Or may even even seem unrelated to the input
    - E.g. "My screen is blank" - > "Please check if your computer is plugged in."

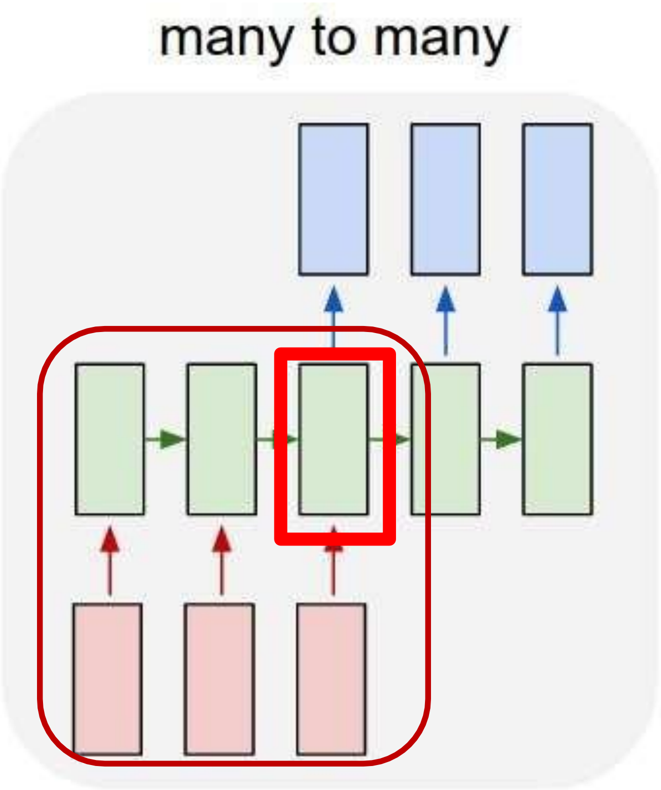
Modelling the problem



- *Delayed* sequence to sequence

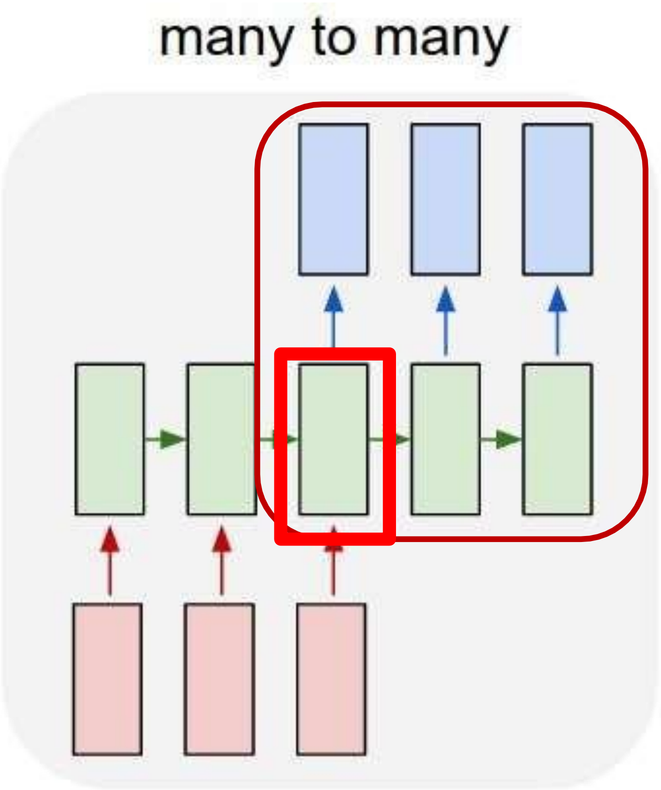
Modelling the problem

First process the input and generate a hidden representation for it



- *Delayed* sequence to sequence

Modelling the problem

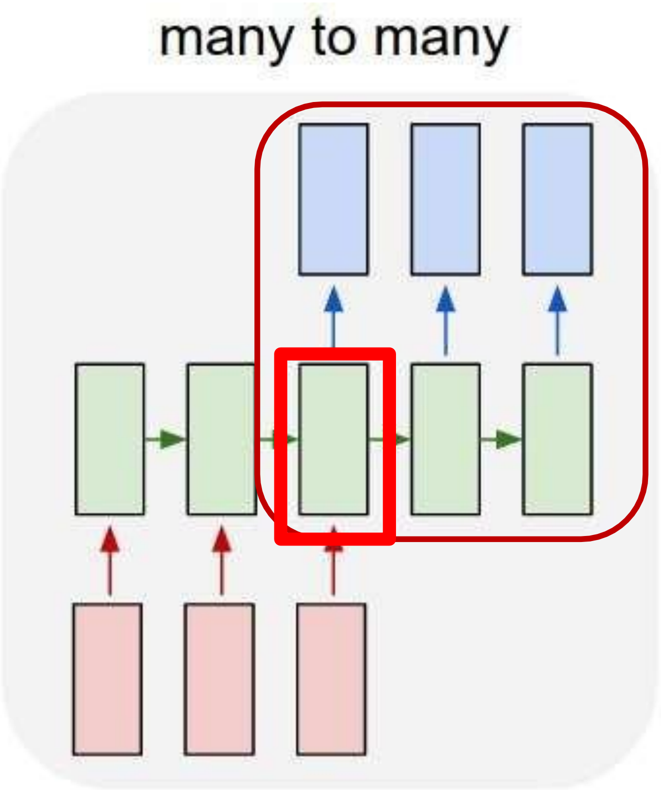


First process the input and generate a hidden representation for it

Then use it to generate an output

- *Delayed* sequence to sequence

Modelling the problem

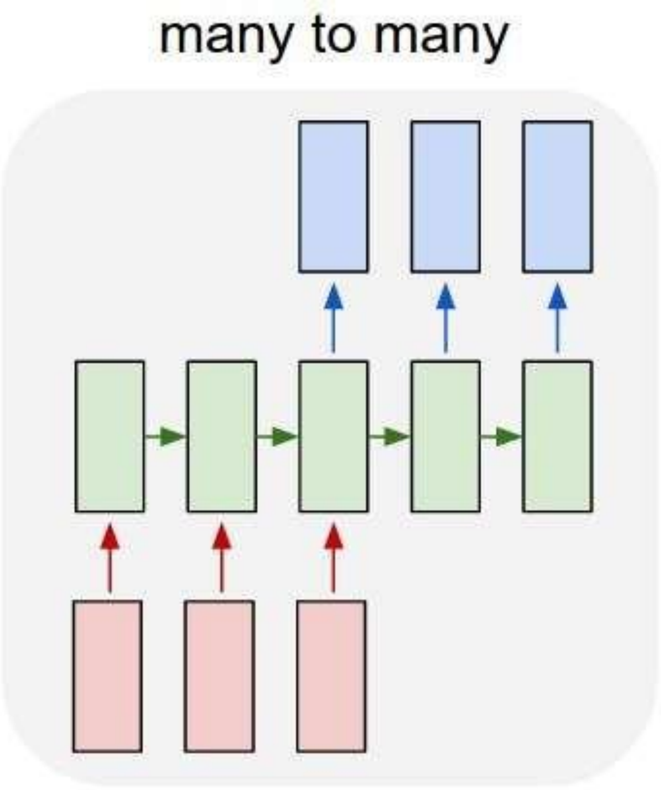


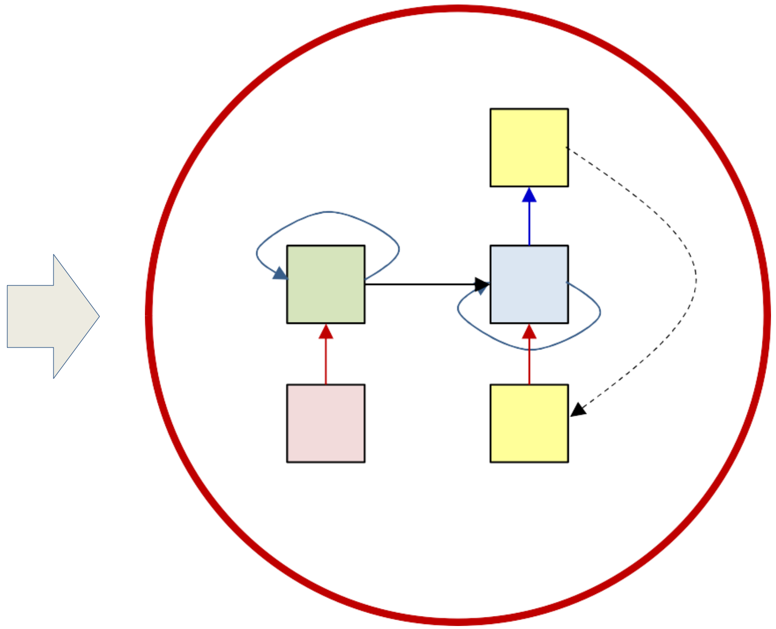
First process the input and generate a hidden representation for it

Then use it to generate an output

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

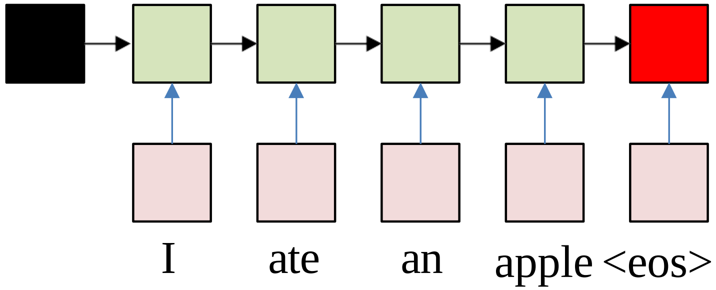
Modelling the problem

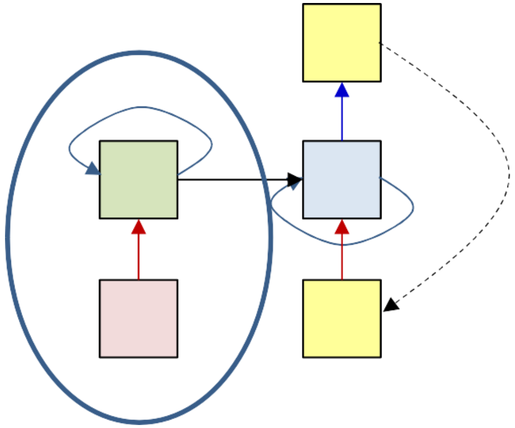




- *Delayed* sequence to sequence
  - Delayed *self-referencing* sequence-to-sequence

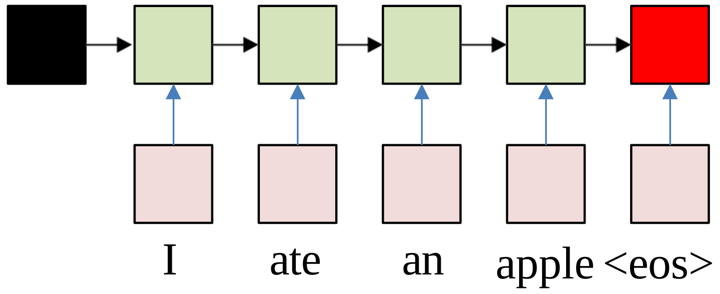
The “simple” translation model

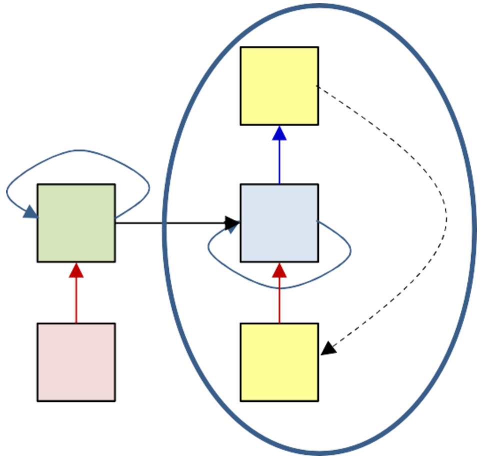




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

The "simple" translation model

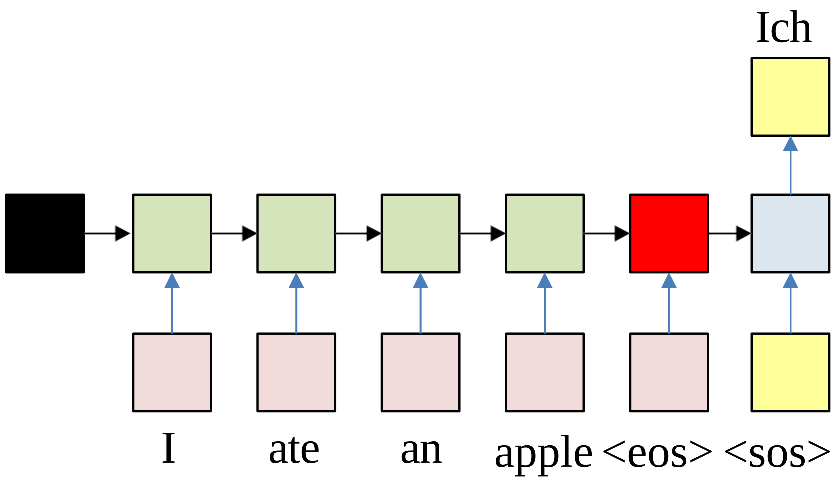




- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence

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

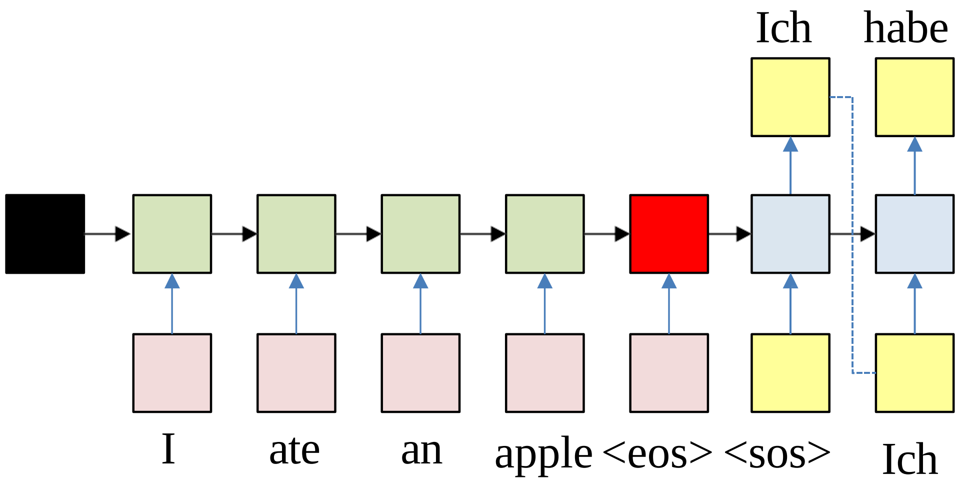
The “simple” translation model



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

- Subsequently a *second* RNN uses the hidden activation as initial state, 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

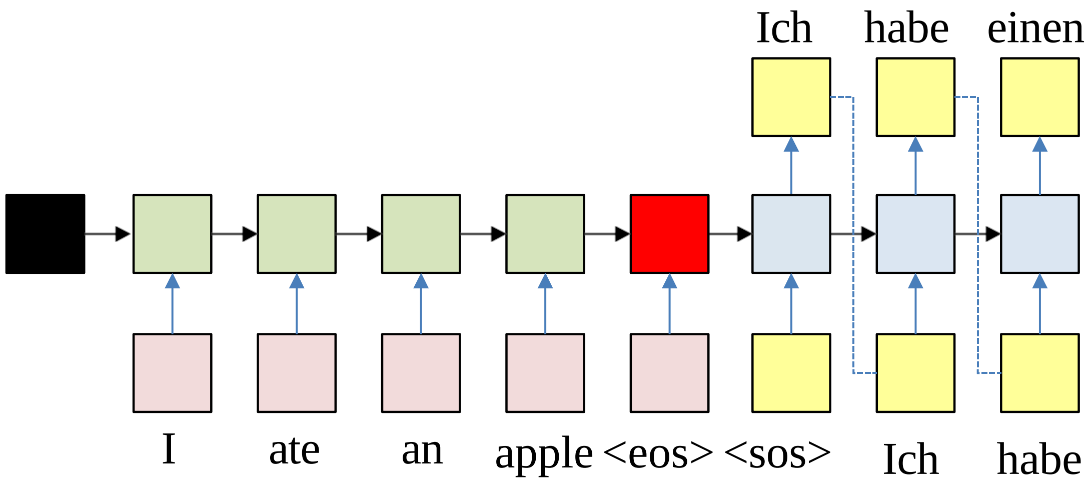
The “simple” translation model



- 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, 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

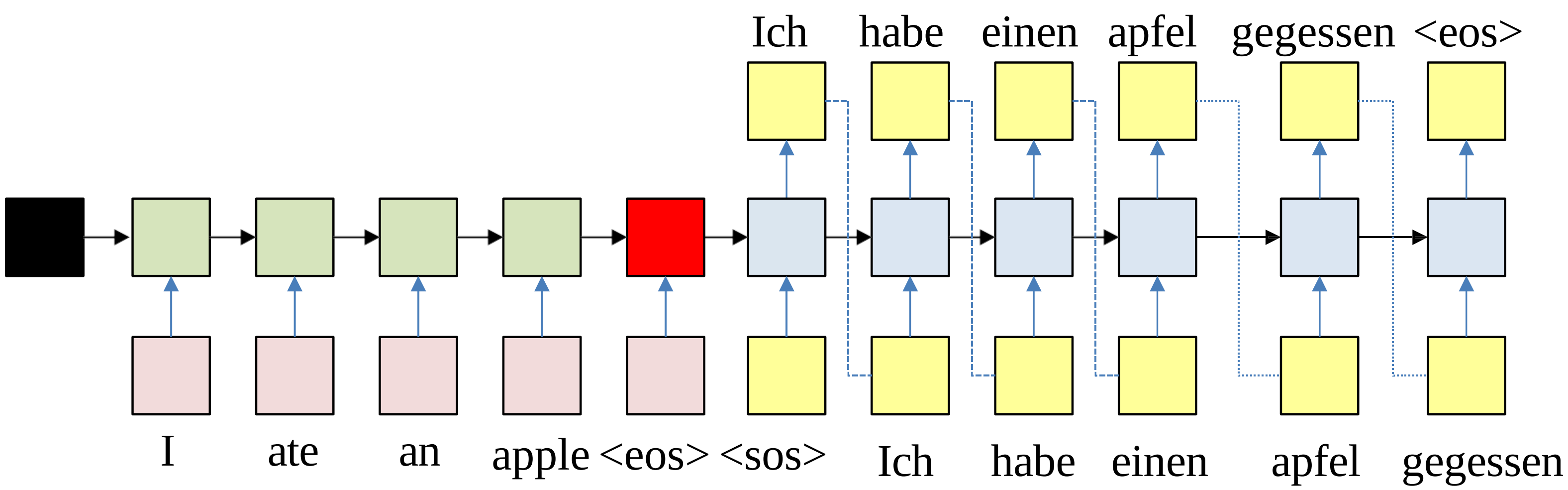
The “simple” translation model



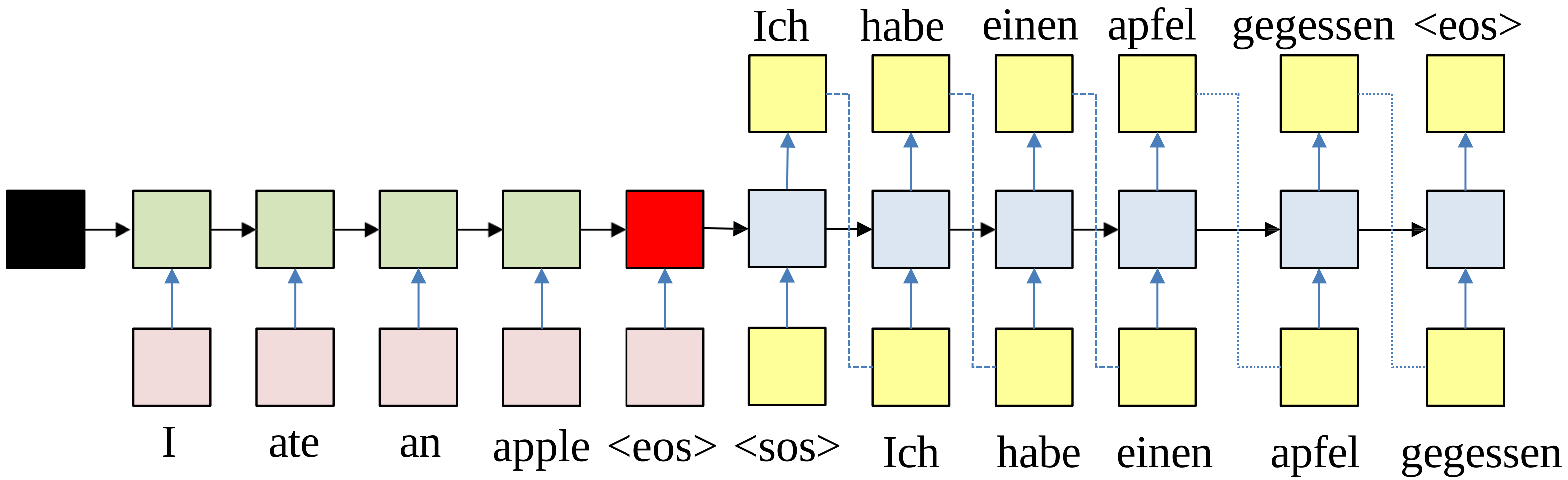
- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

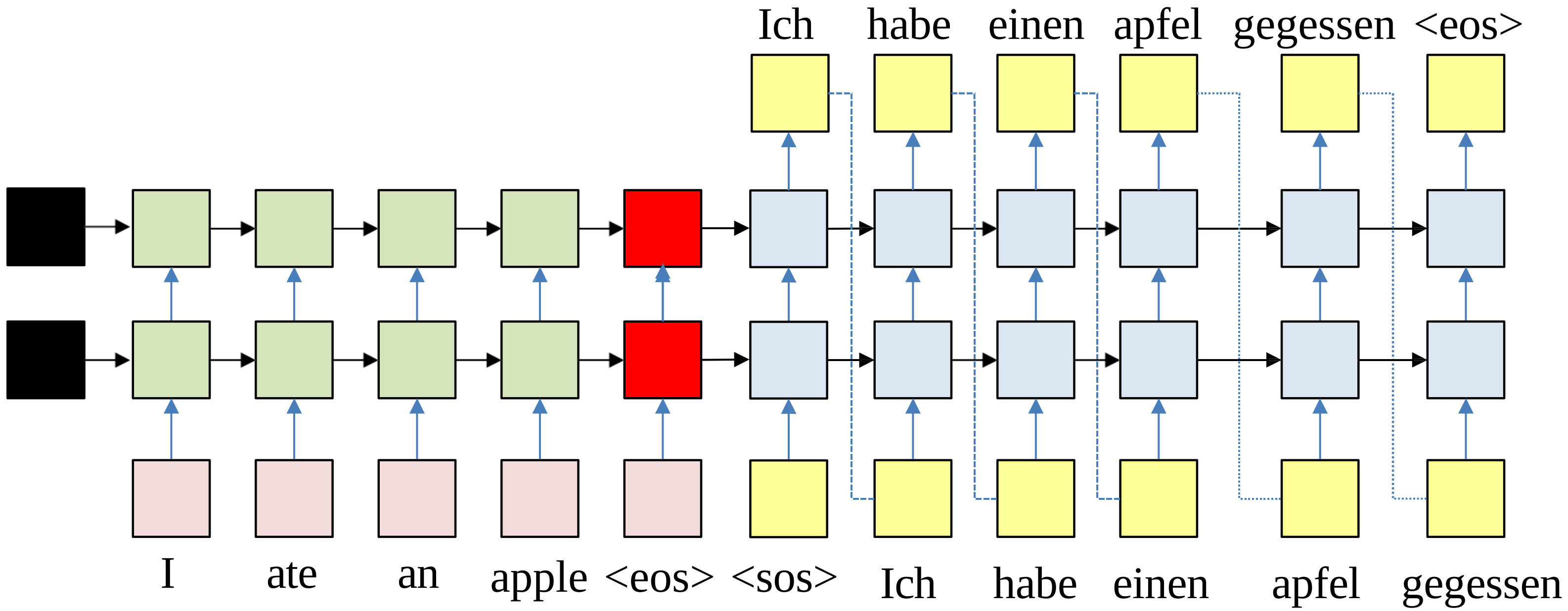
- Subsequently a *second* RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

The “simple” translation model



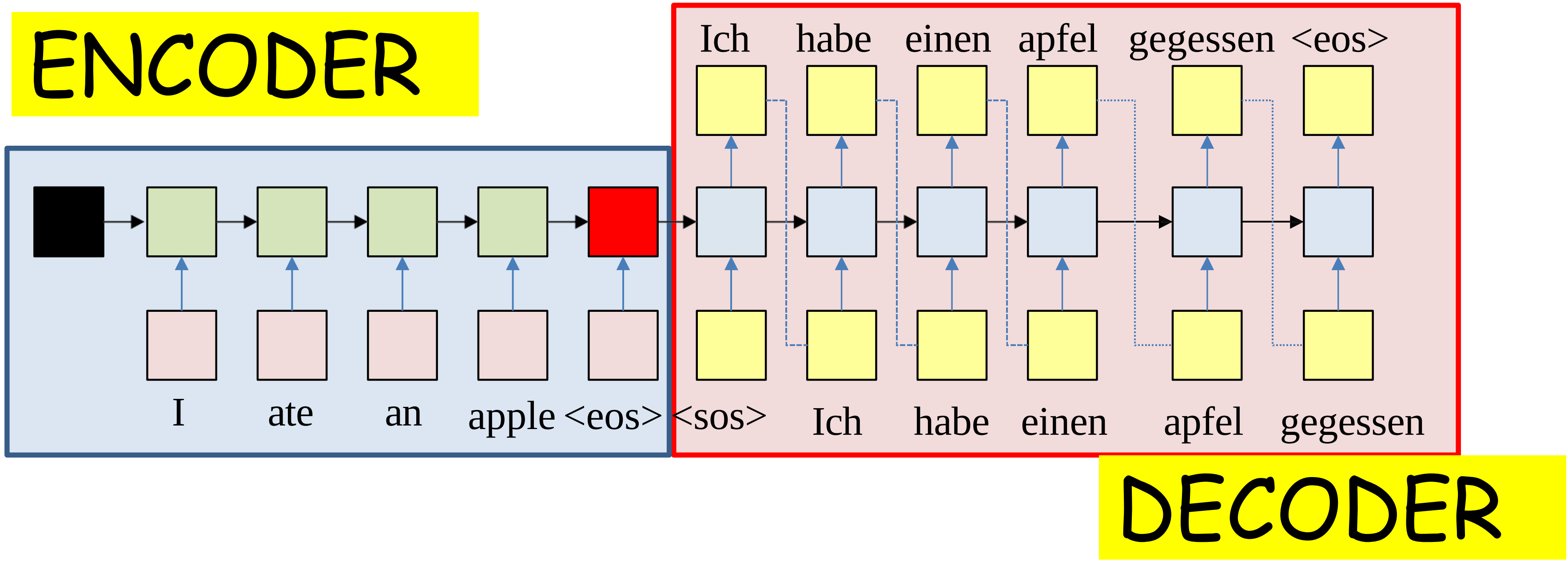
- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state, and <sos> as initial symbol, 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





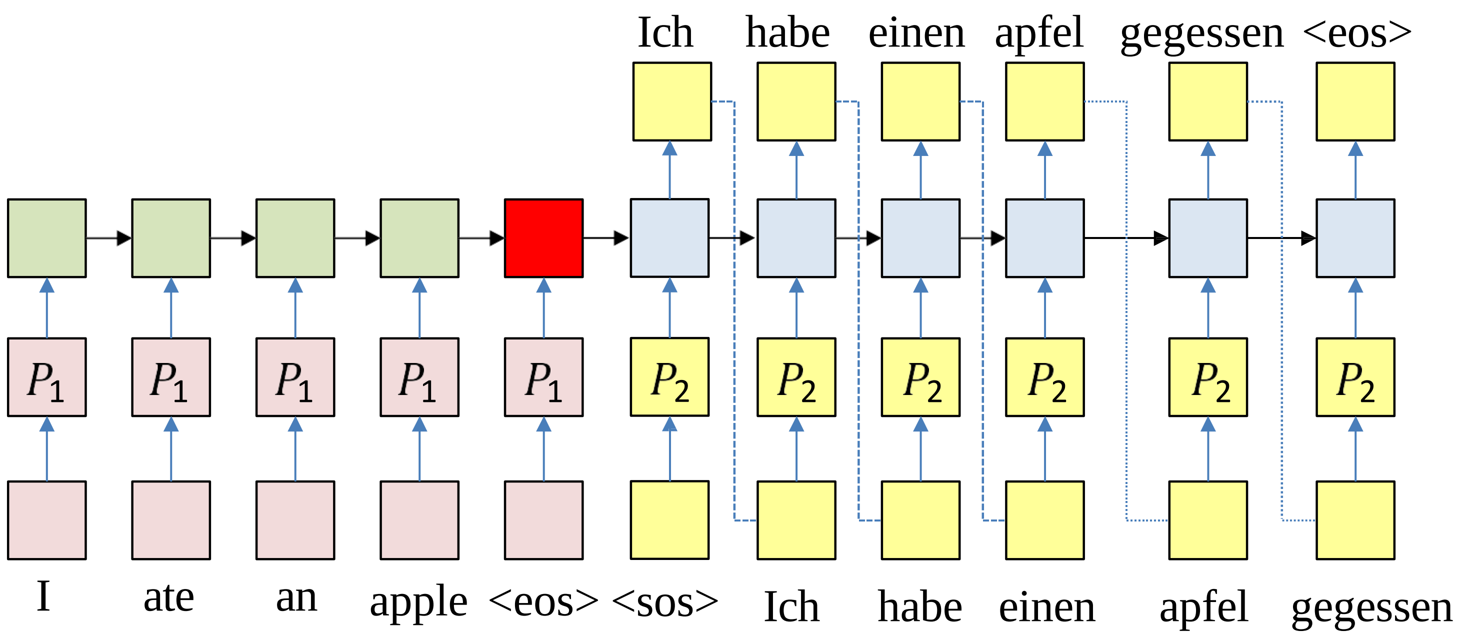
- We illustrate with a single hidden layer
- It generalizes to more layers

The “simple” translation model



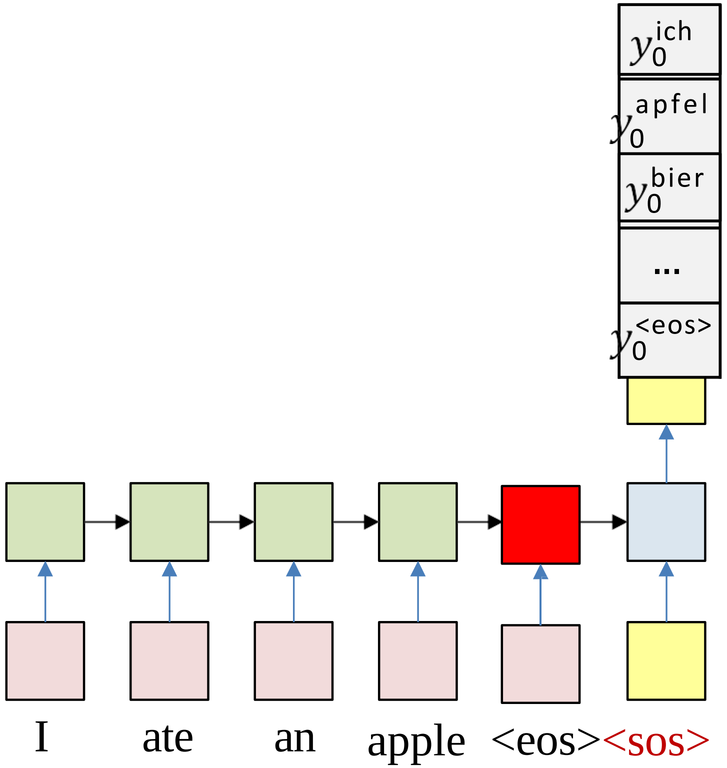
- The recurrent structure that extracts the hidden representation from the input sequence is the *encoder*
- The recurrent structure that utilizes this representation to produce the output sequence is the *decoder*

The “simple” translation model



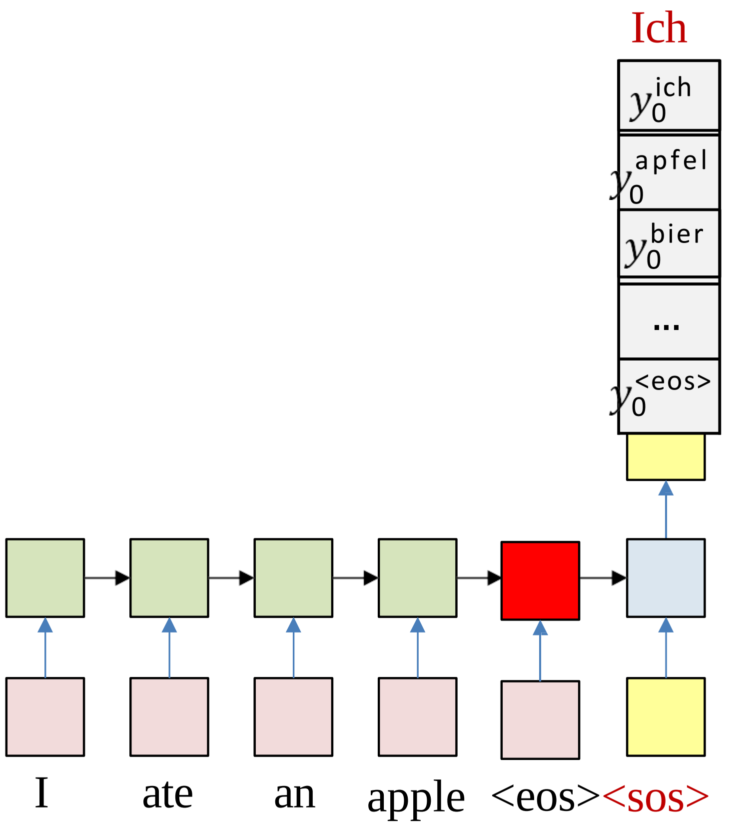
- A more detailed look: The one-hot word representations may be compressed via embeddings
  - Embeddings will be learned along with the rest of the net
  - In the following slides we will not represent the projection matrices

What the network actually produces



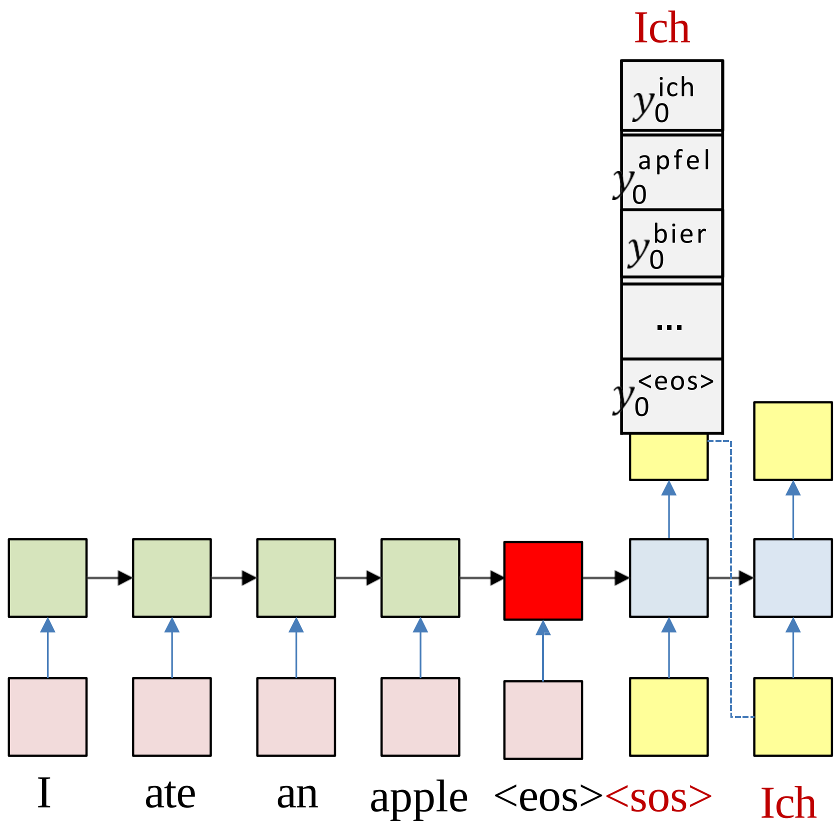
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w \mid O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until 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



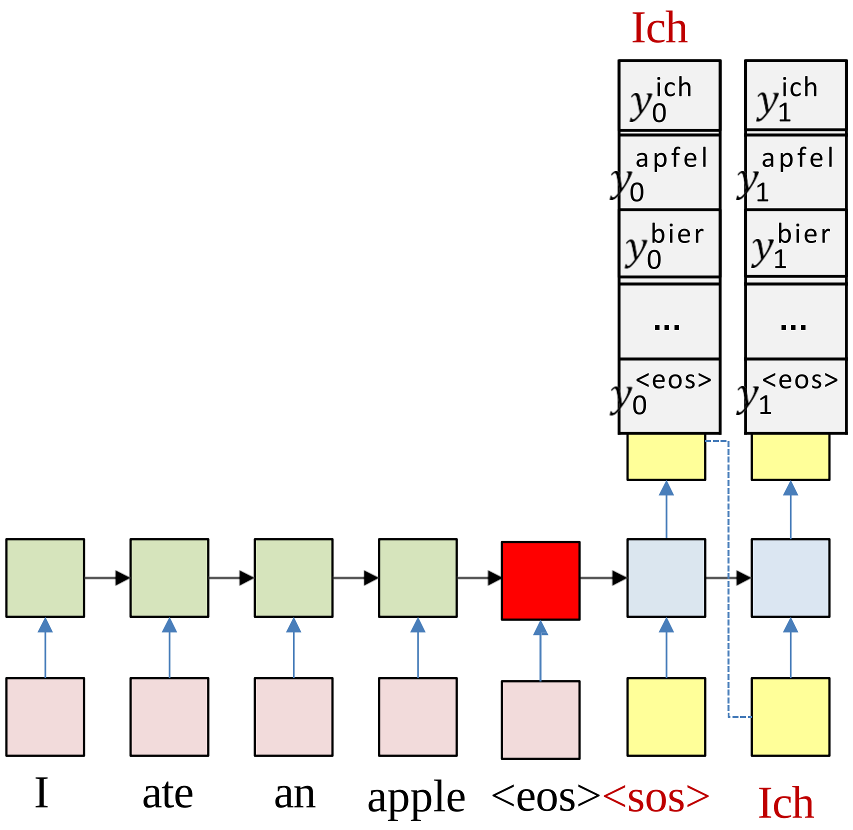
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w \mid O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until 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



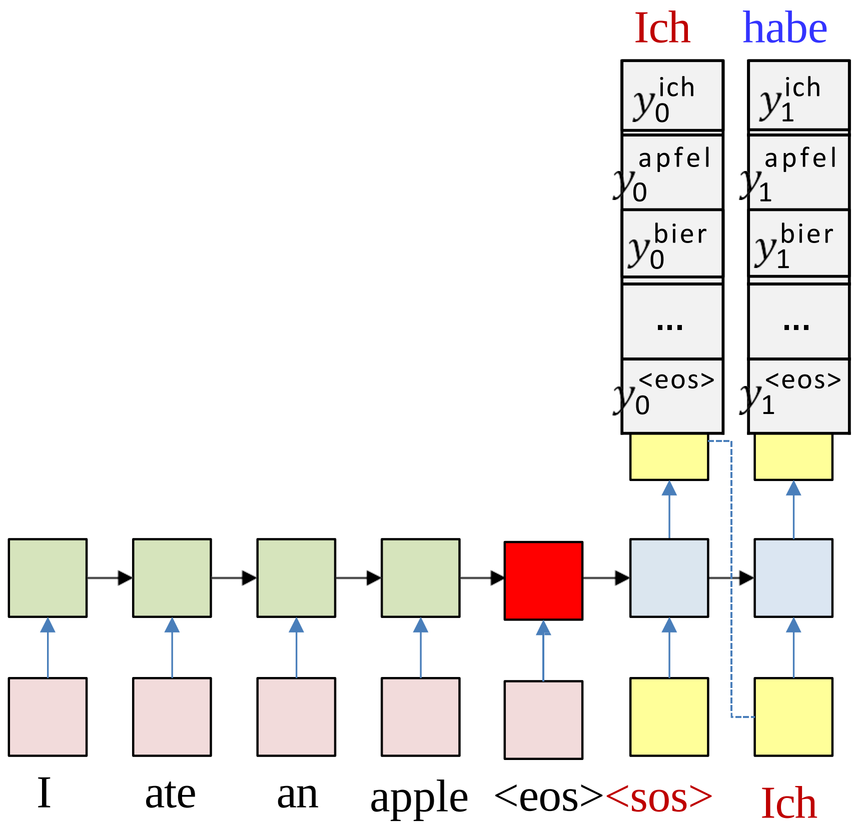
- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w \mid O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until 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

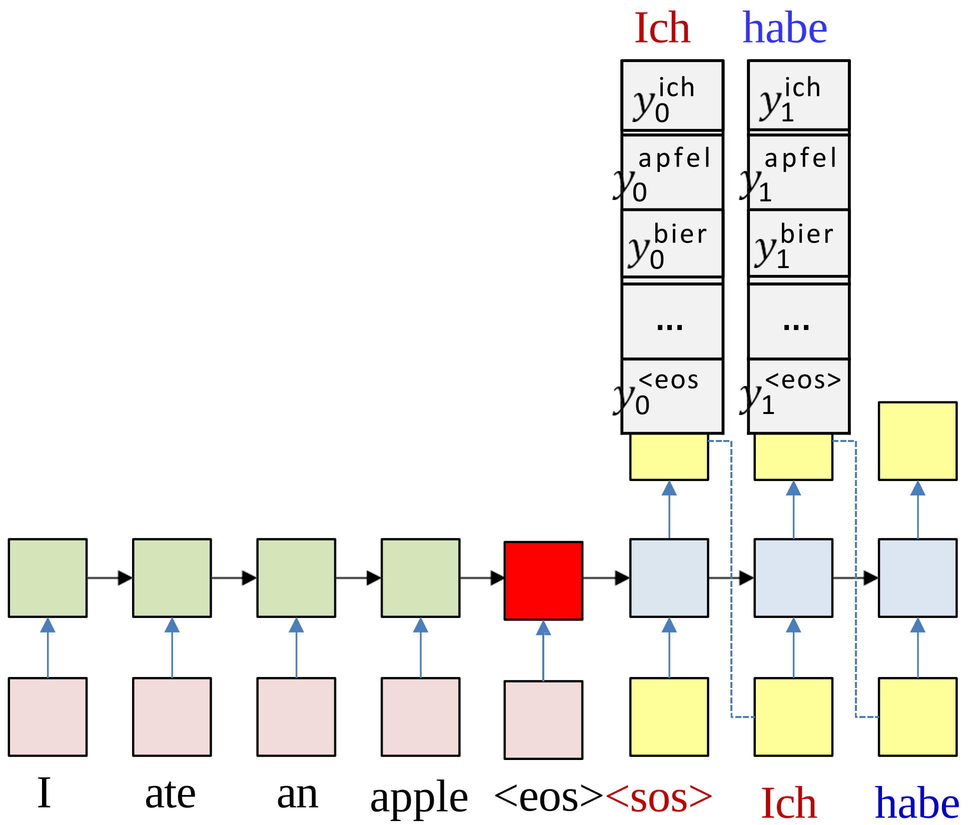


- At each time k the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w \mid O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until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

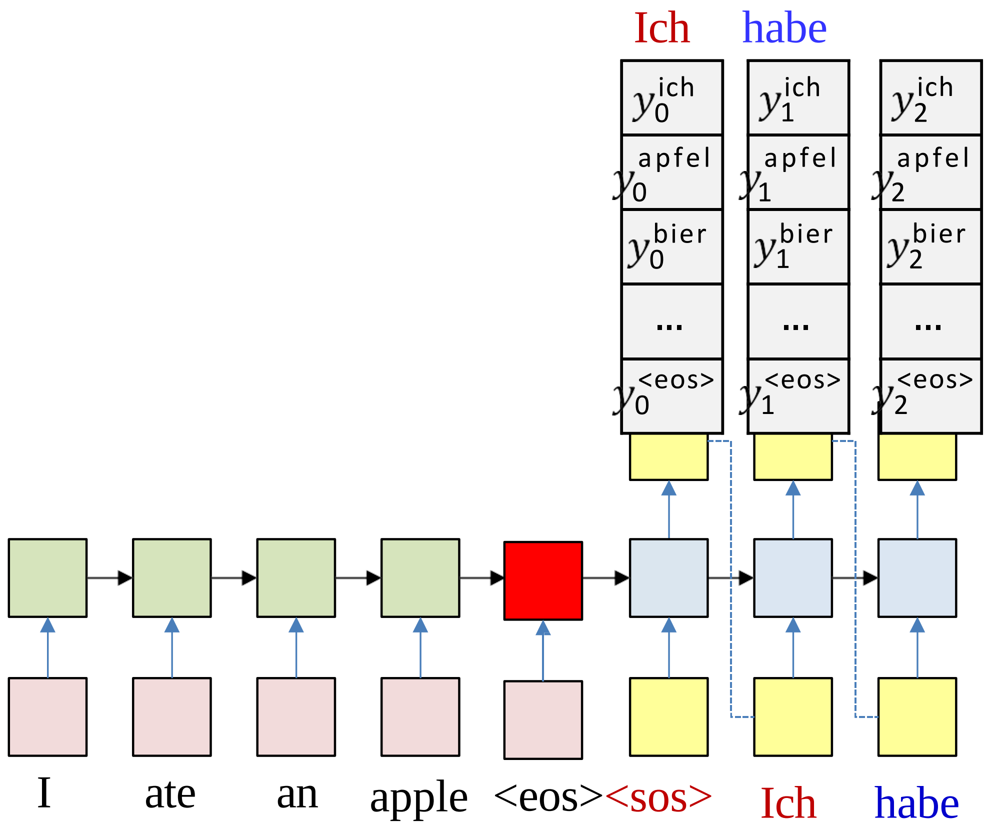
What the network actually produces



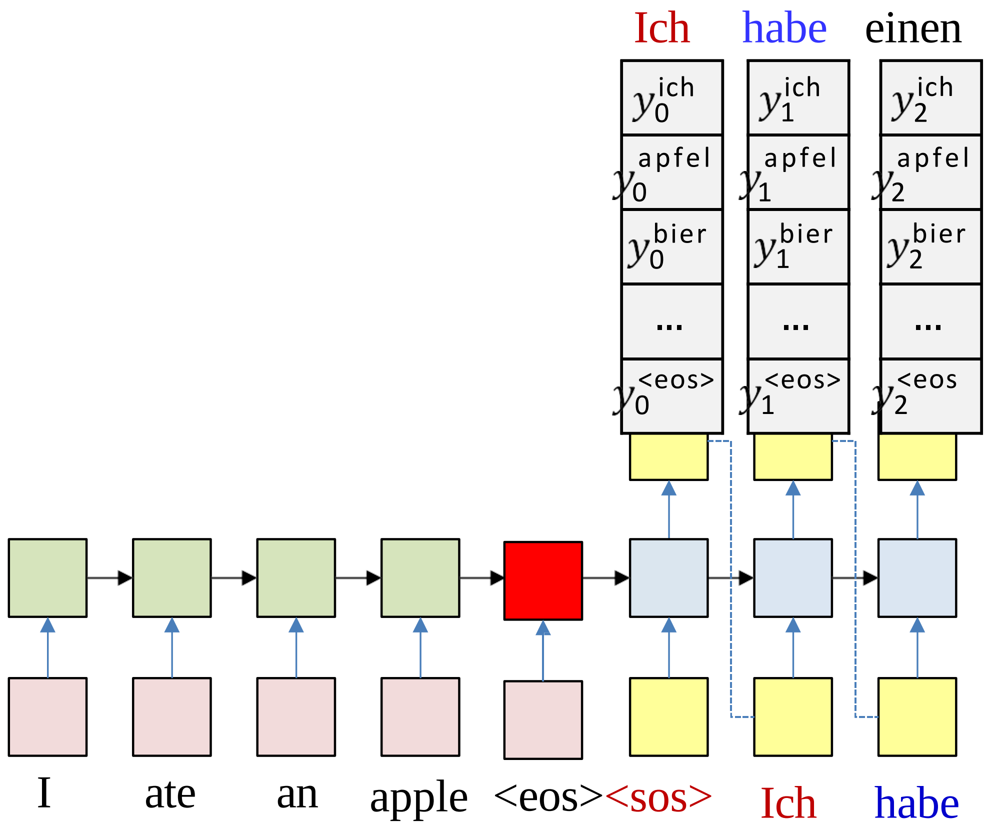
What the network actually produces



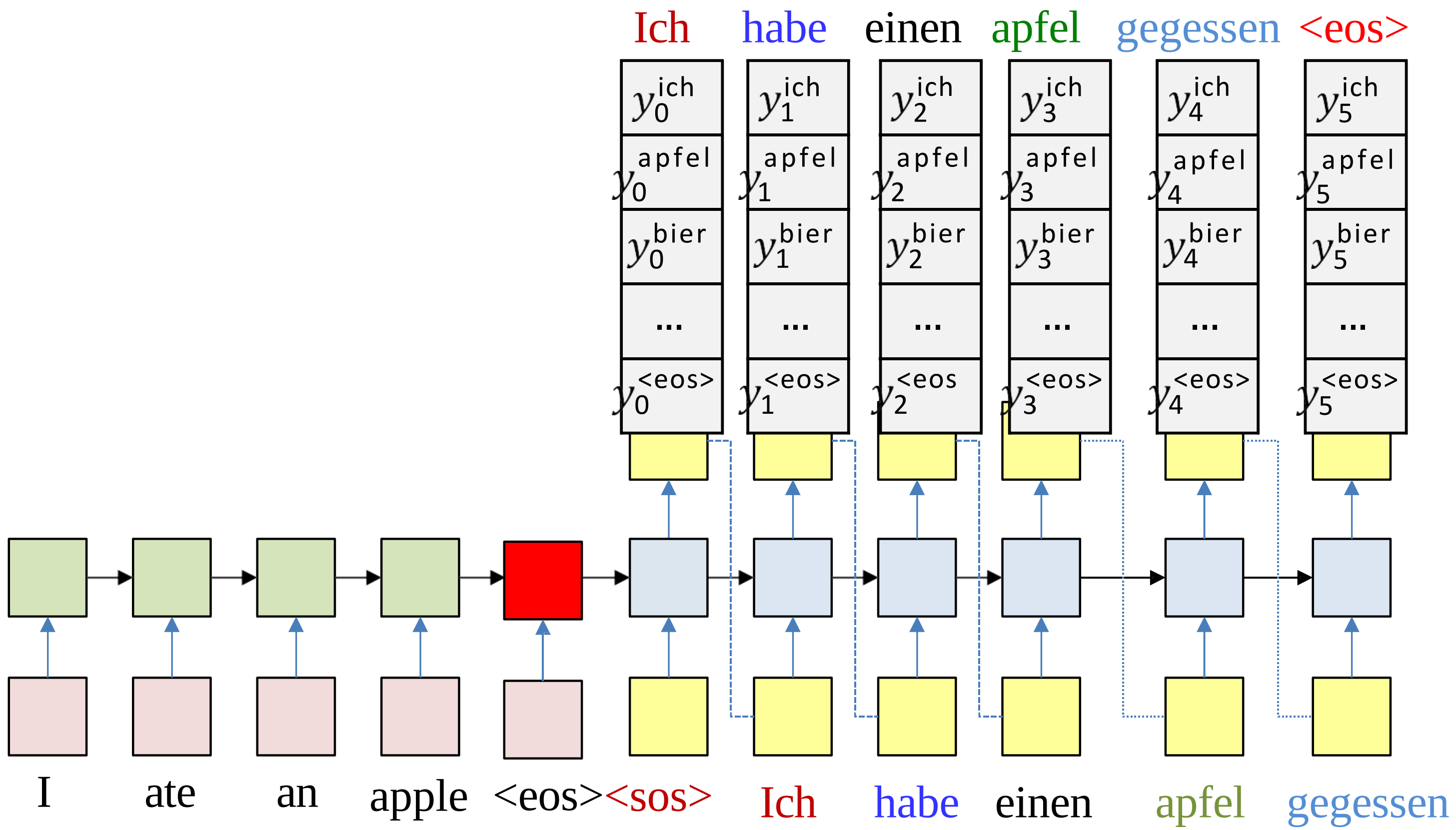
What the network actually produces



What the network actually produces

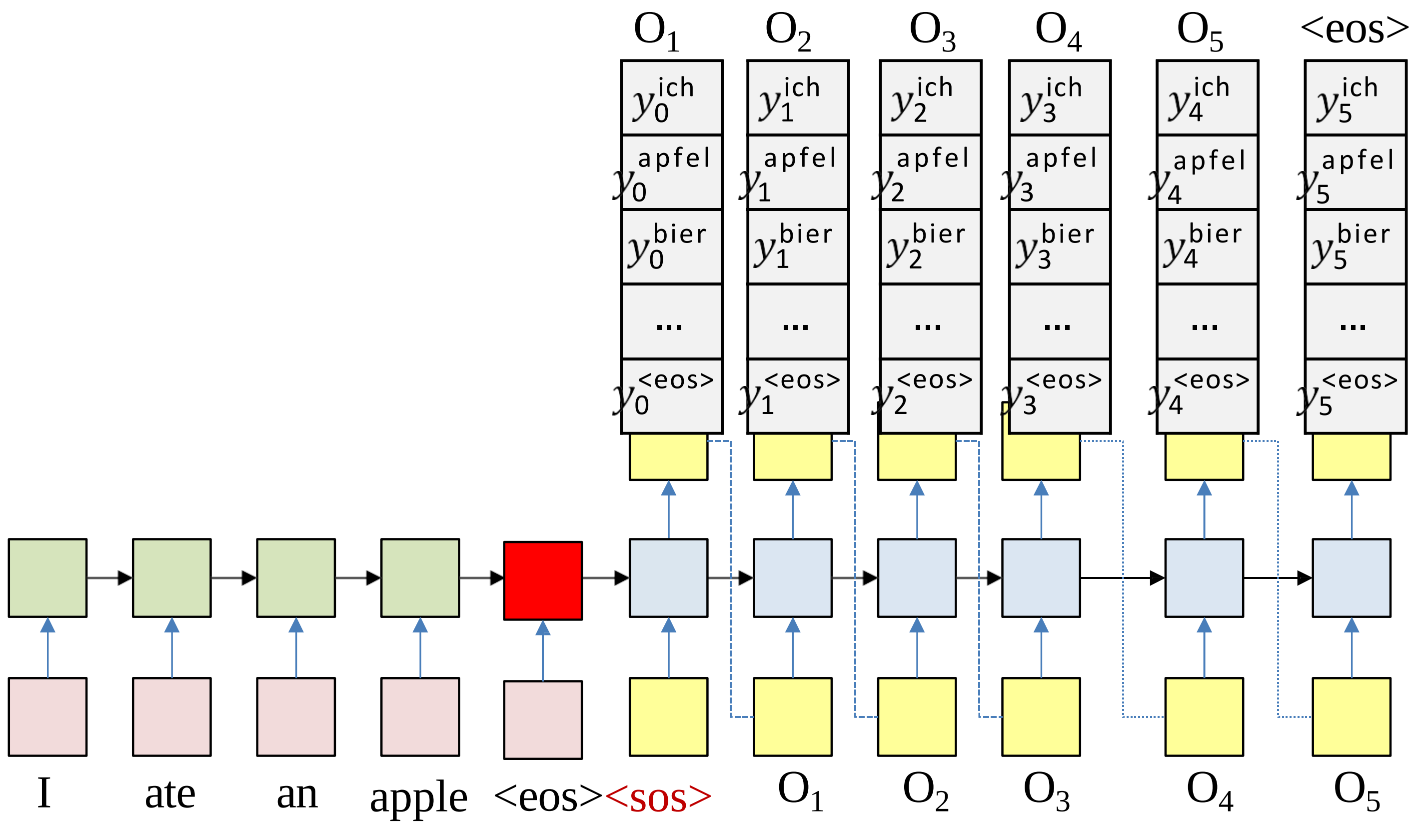


Generating an output from the net



- **The process continues until an <eos> is drawn**

The probability of the output

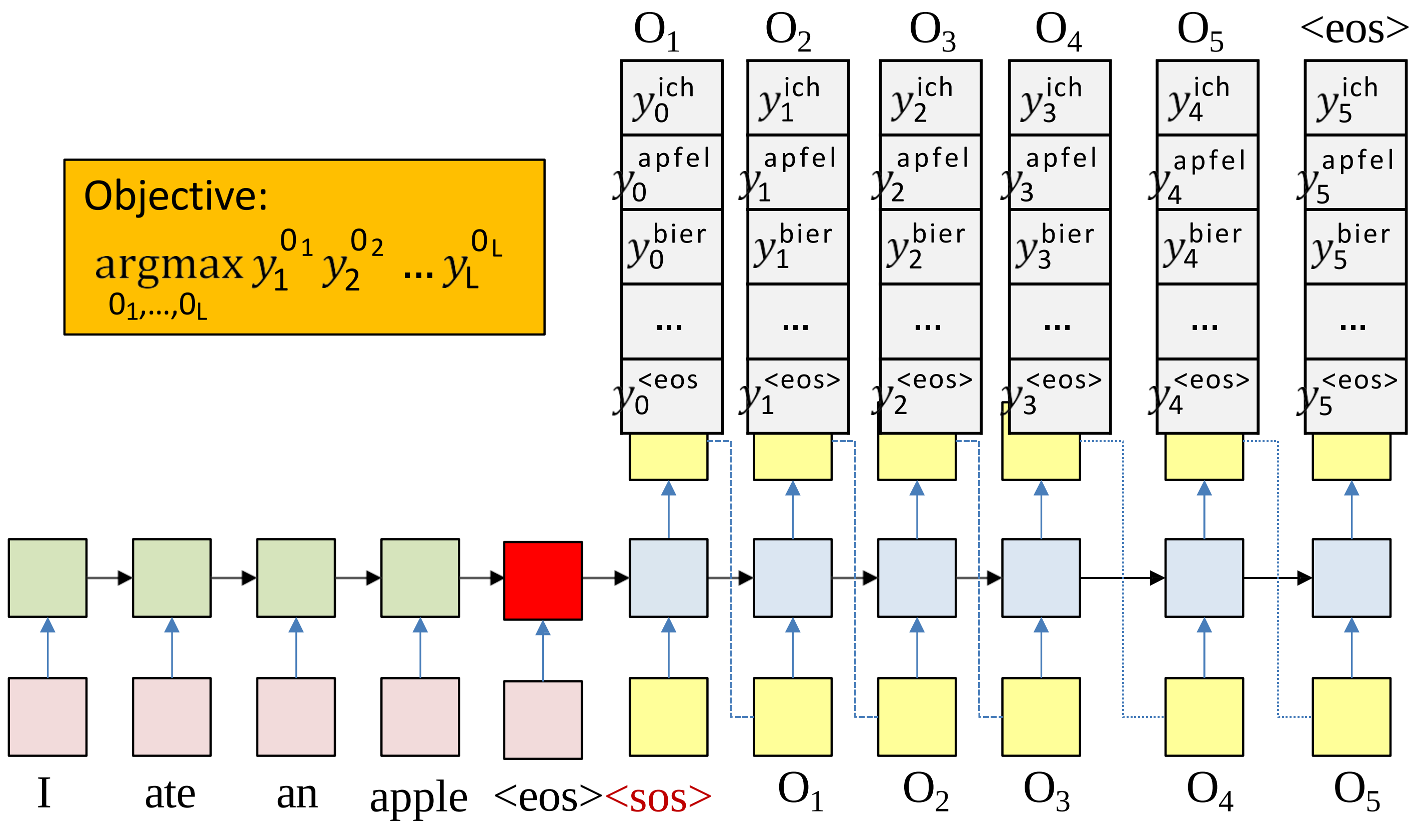


- ***The objective of drawing: Produce the most likely output (that ends in an <eos>)***

$$\underset{O_1,\ldots,O_L}{\operatorname{argmax}} P(O_1, \ldots, O_L | W_1^{\text{in}}, \ldots, W_N^{\text{in}})$$

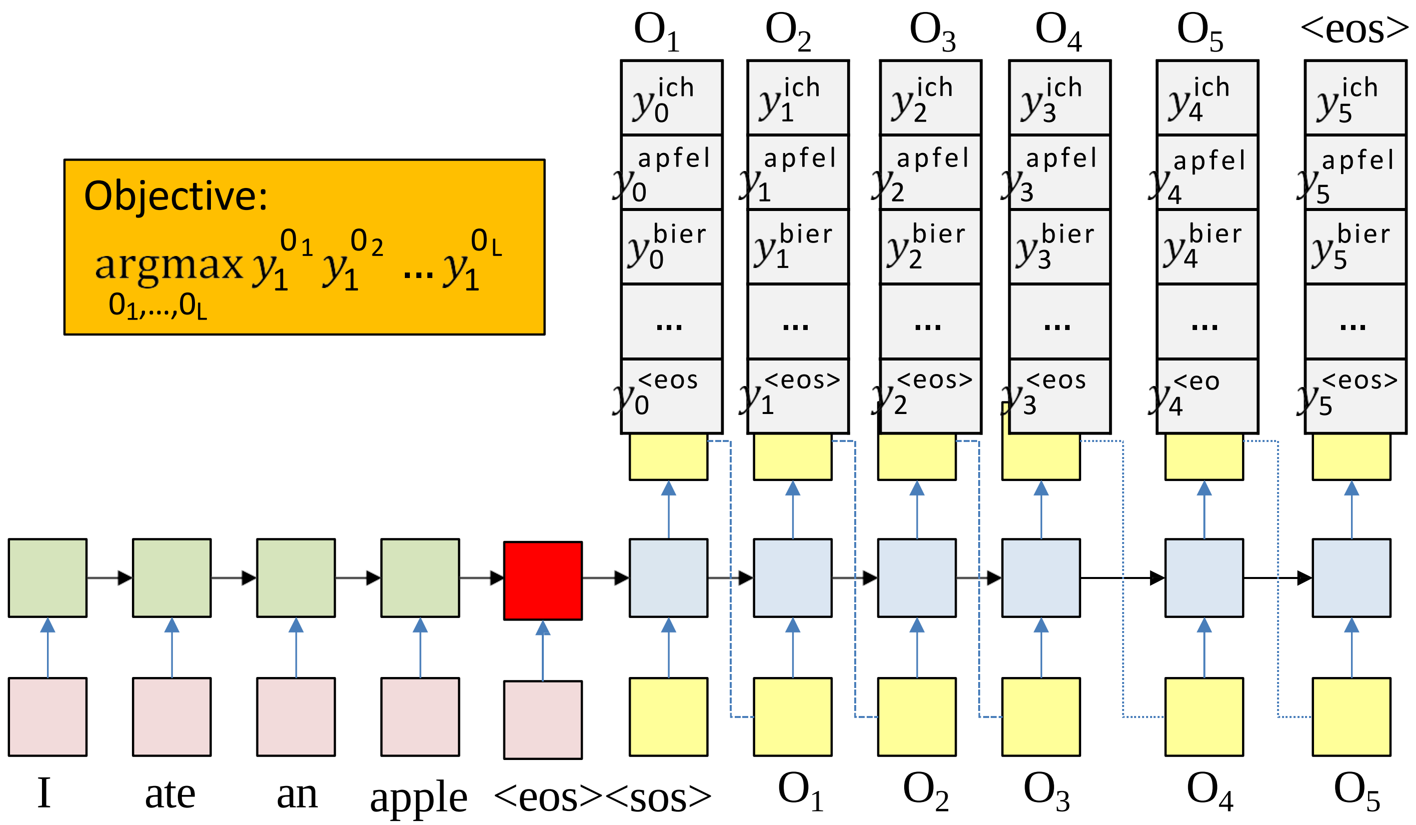
$$= \underset{O_1,\ldots,O_L}{\operatorname{argmax}} y_1^{O_1} y_2^{O_2} \ldots y_L^{O_L}$$

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

Drawing by random sampling

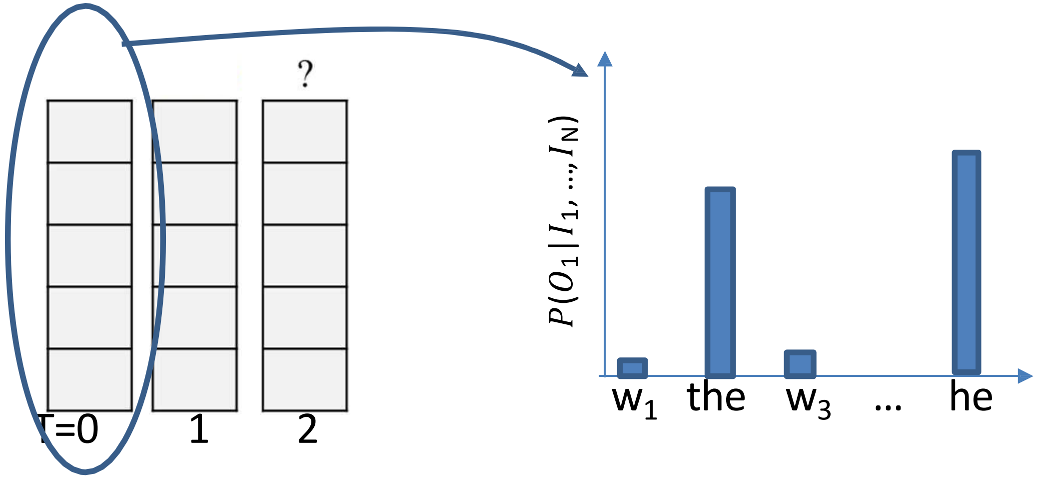


- Alternate option: Randomly draw a word at each time according to the output probability distribution

Your choices can get you stuck

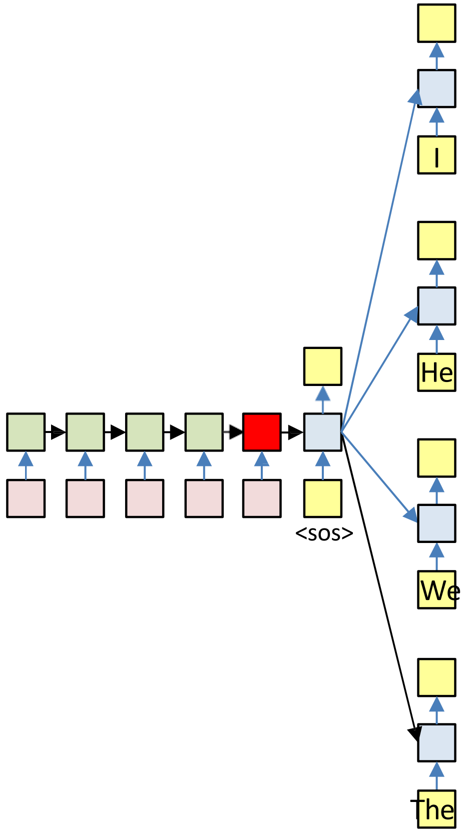
What should we
have chosen at t=1??

Choose “the” or “he”?



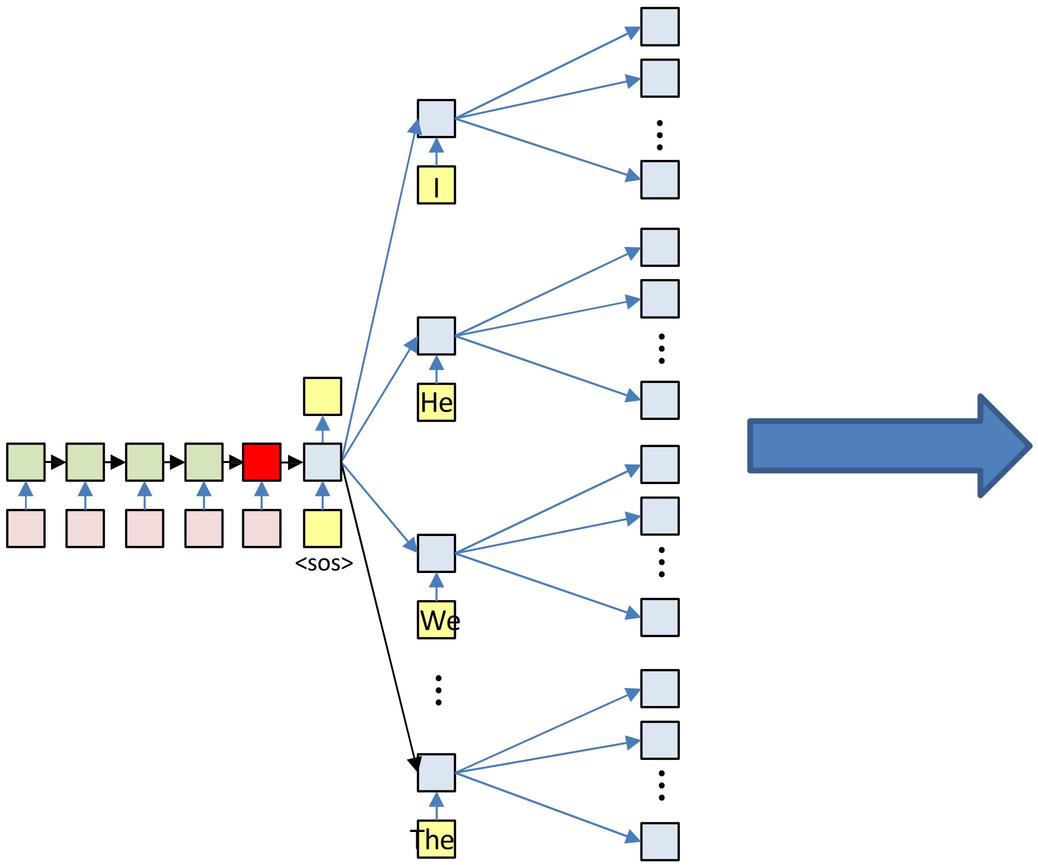
- Problem: making a poor choice at any time commits us to a poor future
  – But we cannot know at that time the choice was poor
- Solution: Don’t choose..

Optimal Solution: Multiple choices



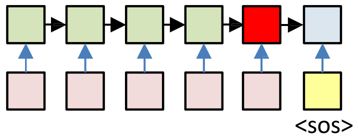
- Retain all choices and *fork* the network
  - With every possible word as input

Problem: Multiple choices



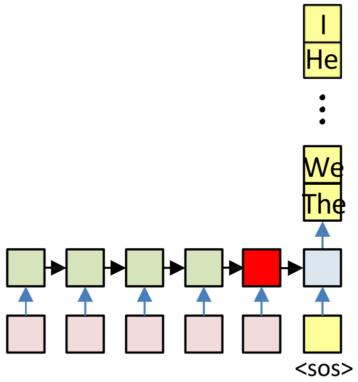
- **Problem**: This will blow up very quickly
  - For an output vocabulary of size V, after T output steps we'd have forked out V^T branches

Optimal Solution: Multiple choices



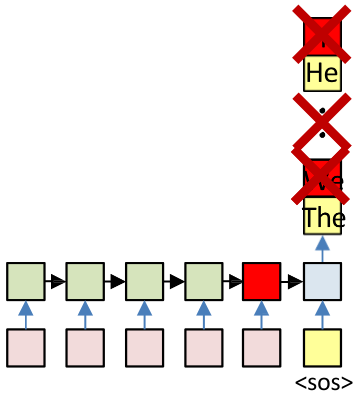
- Retain all choices and *fork* the network
  - With every possible word as input

Solution: Prune



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

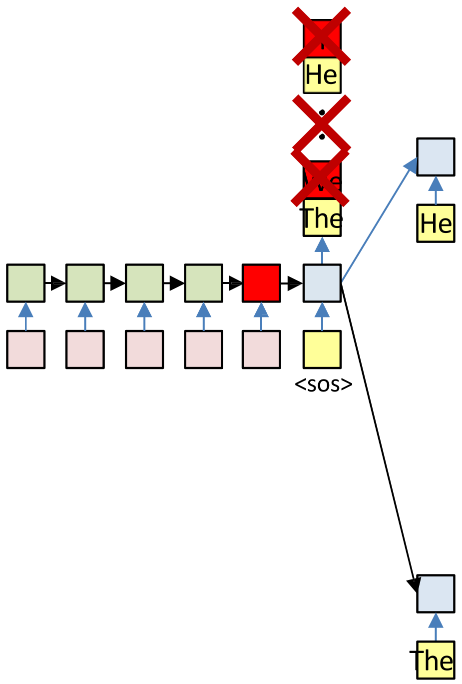
Solution: Prune



$$Top_{\mathrm{K}}\ P(O_1|I_1,\dots,I_{\mathrm{N}})$$

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

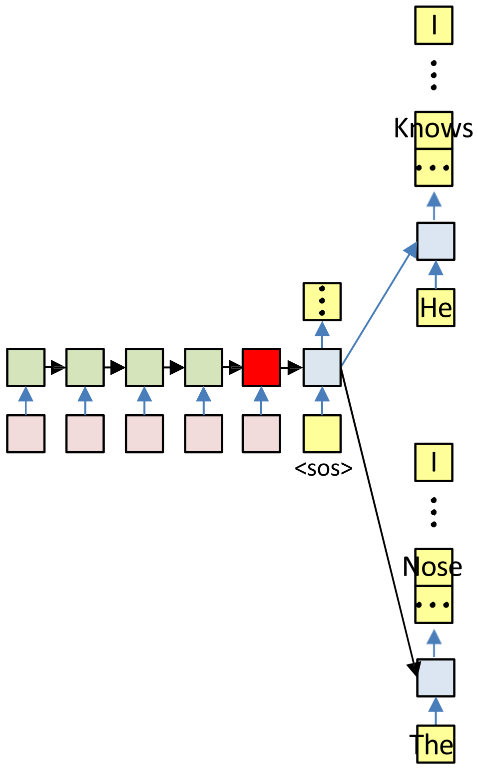
Solution: Prune



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

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

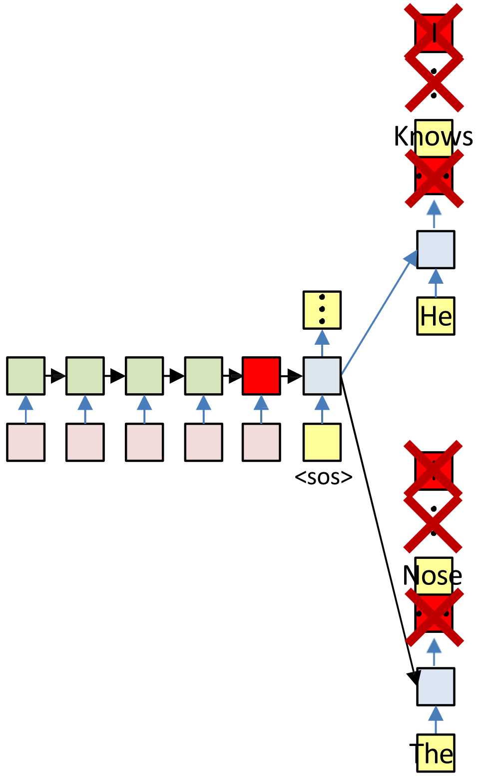
Solution: Prune



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

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

Solution: Prune



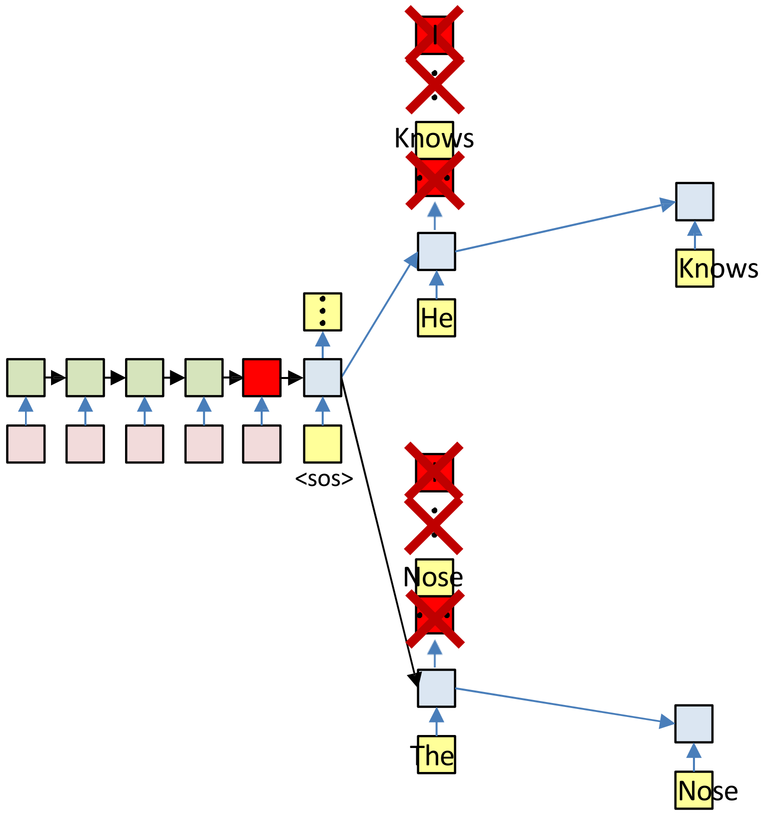
Note: based on product

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

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

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

Solution: Prune



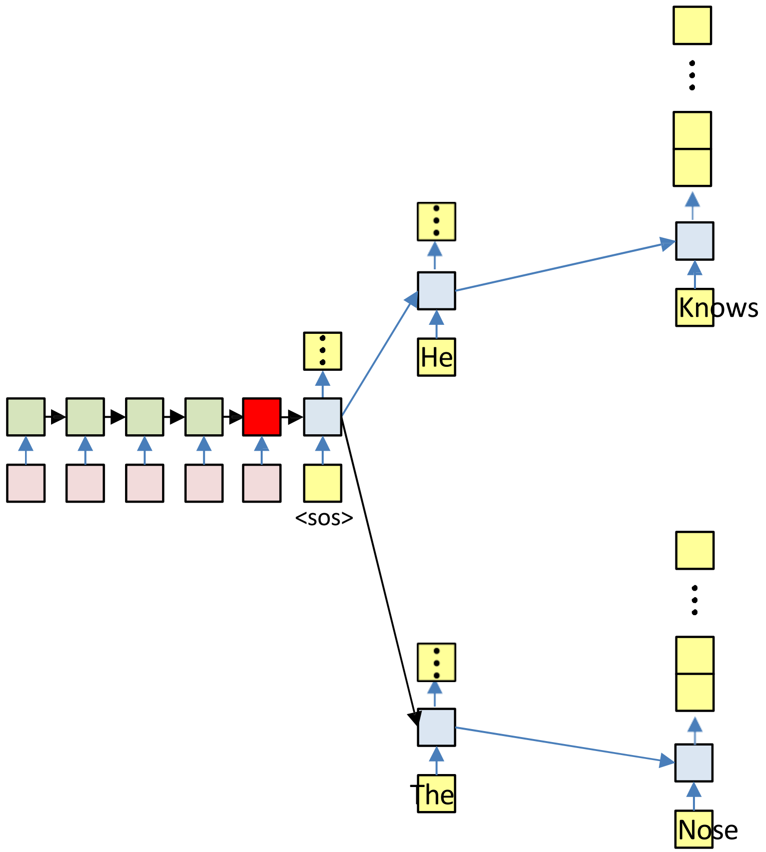
Note: based on product

$$Top_K\ P(O_2O_1|I_1,\dots,I_N)$$

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

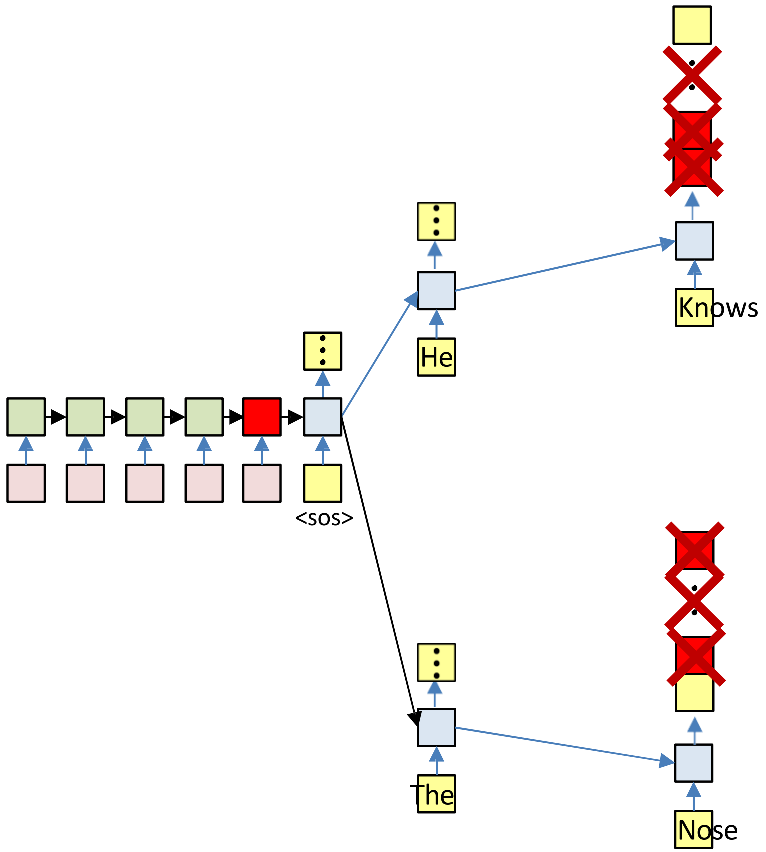
- **Solution: Prune**
  - At each time, retain only the top K scoring forks

Solution: Prune



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

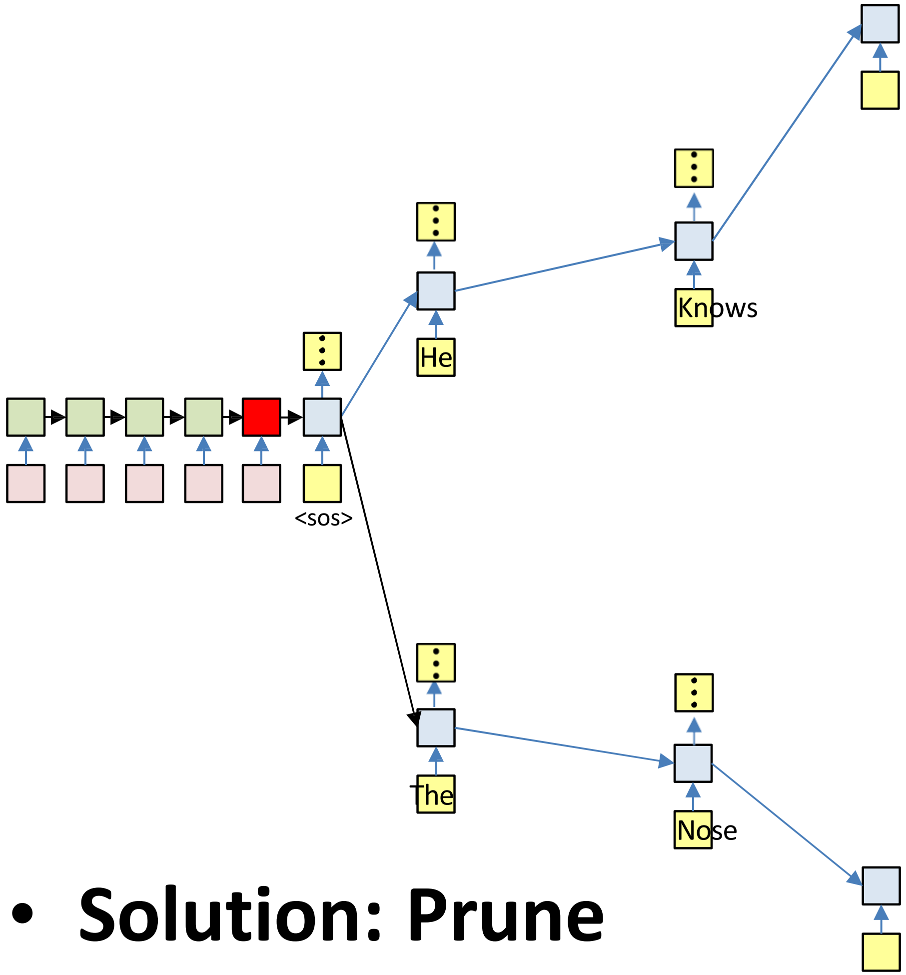
Solution: Prune



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

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

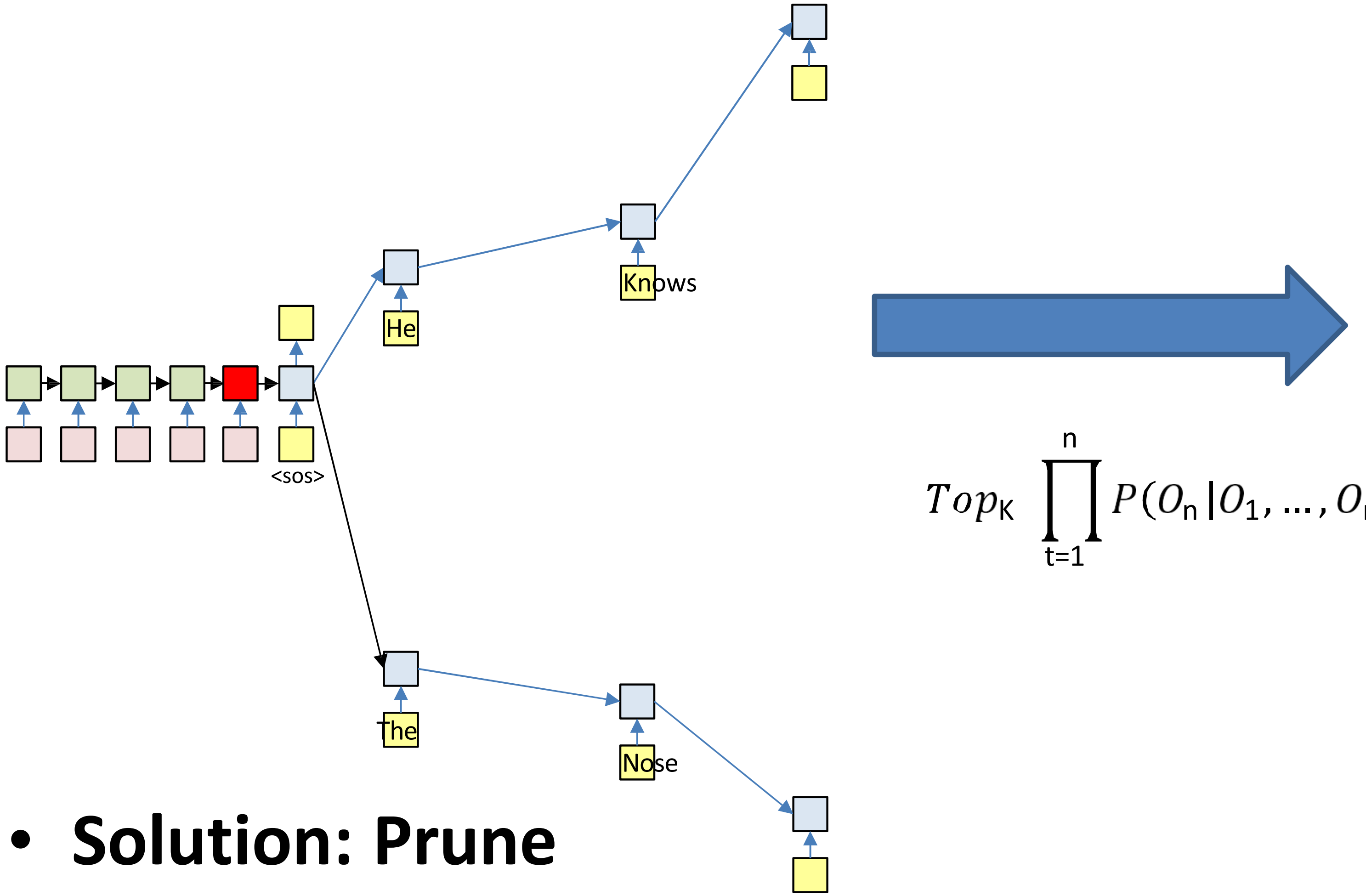
Solution: Prune



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

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

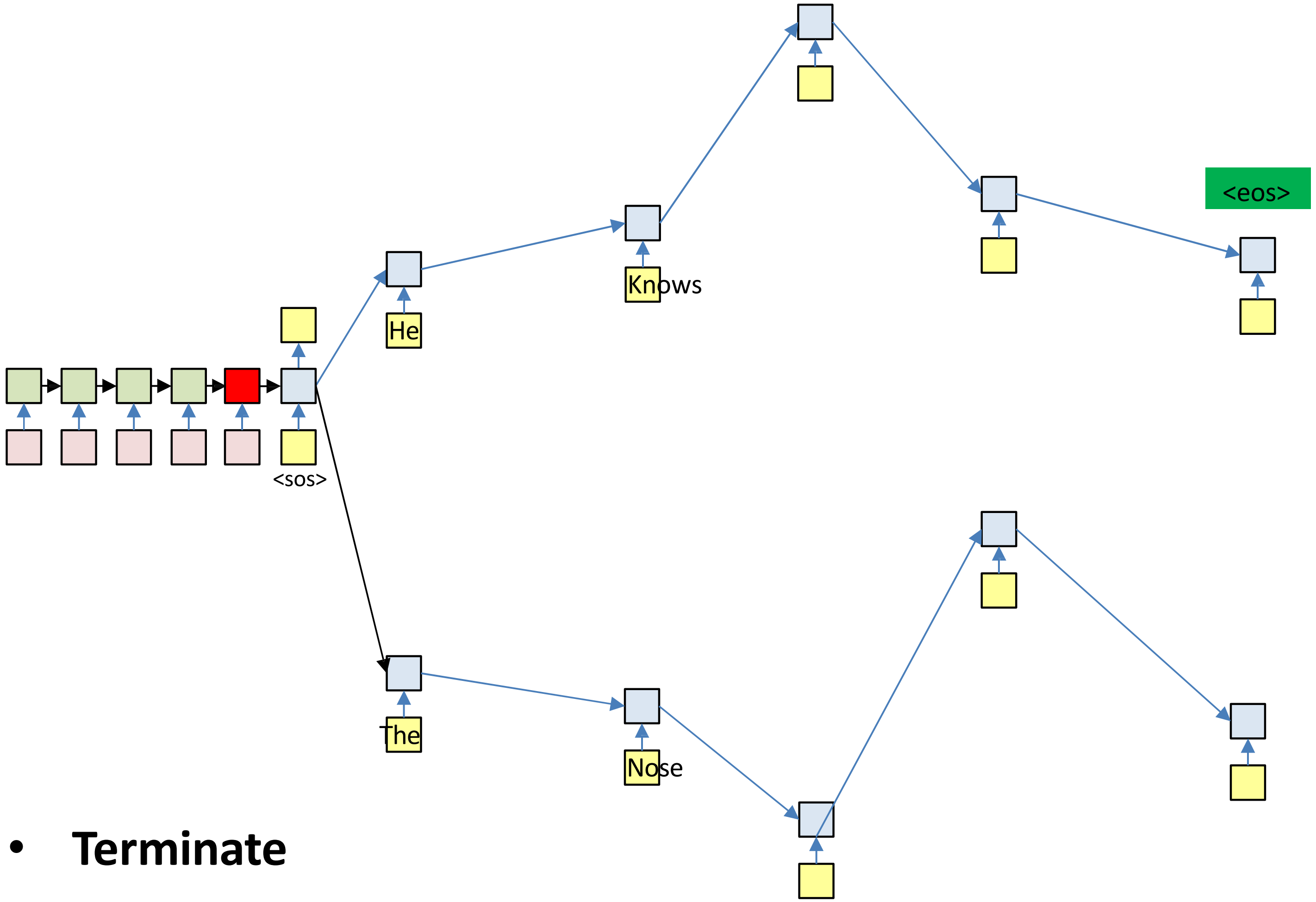
Solution: Prune



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

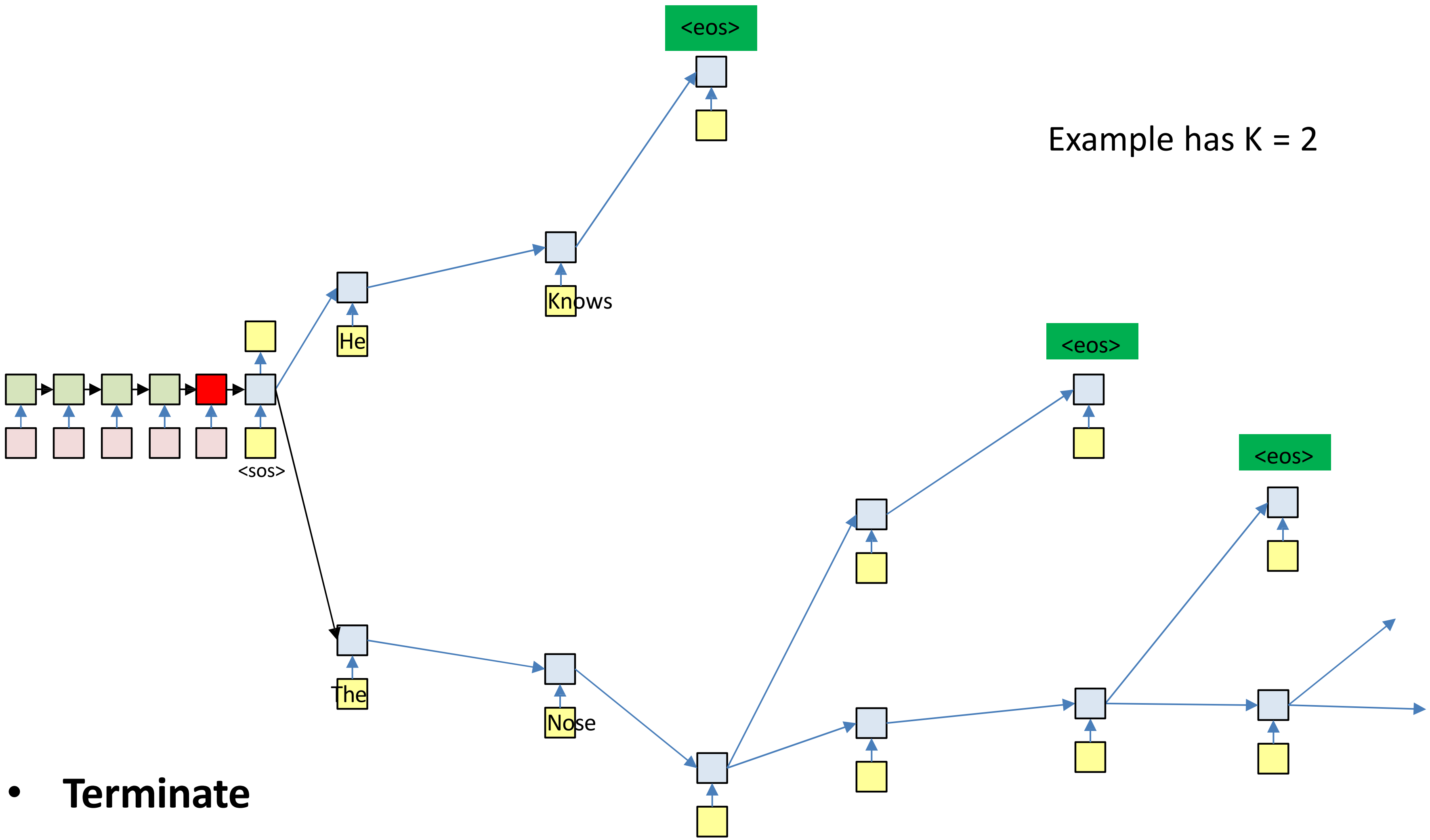
- **Solution: Prune**
  - At each time, retain only the top K scoring forks

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

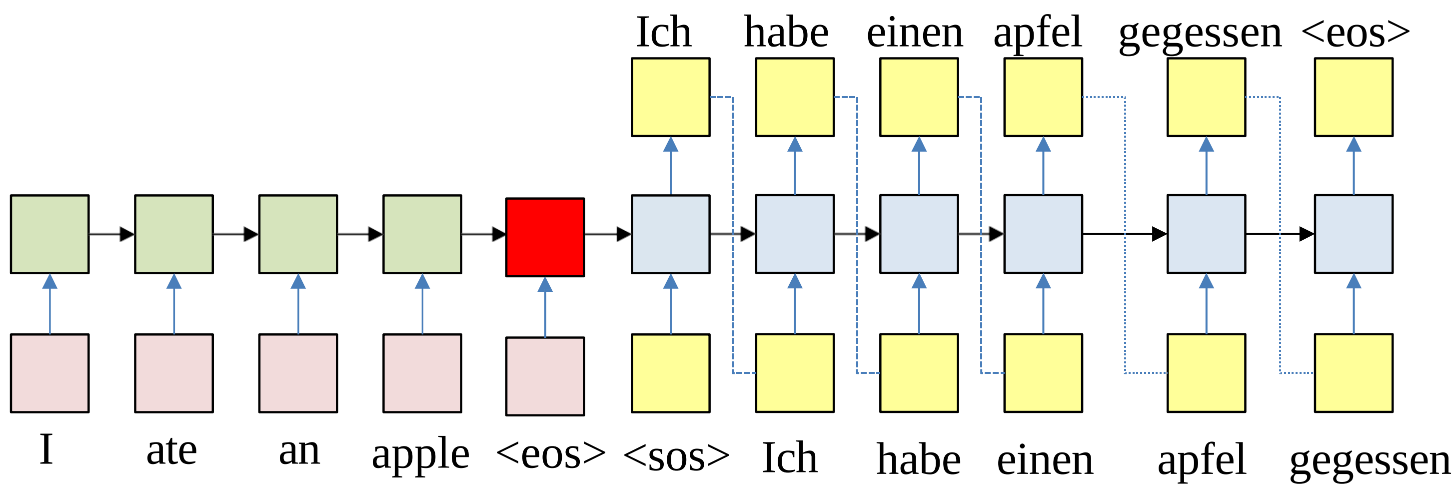
Termination: <eos>



Example has K = 2

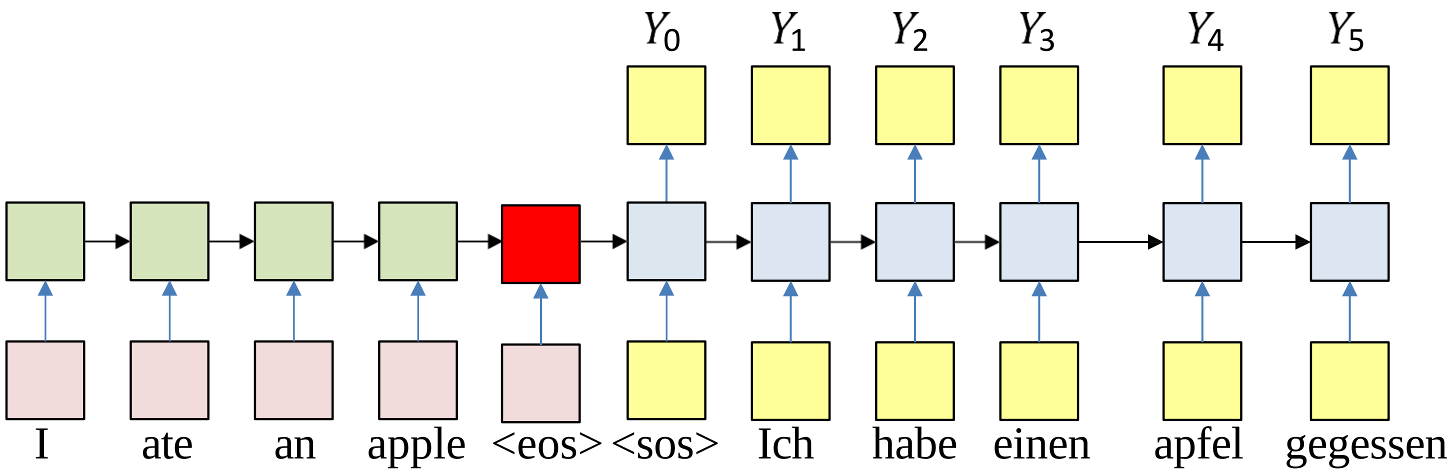
- **Terminate**
  - Paths cannot continue once the output an <eos>
    - So paths may be different lengths
      - Select the most likely sequence ending in <eos> across *all* terminating sequences

Training the system



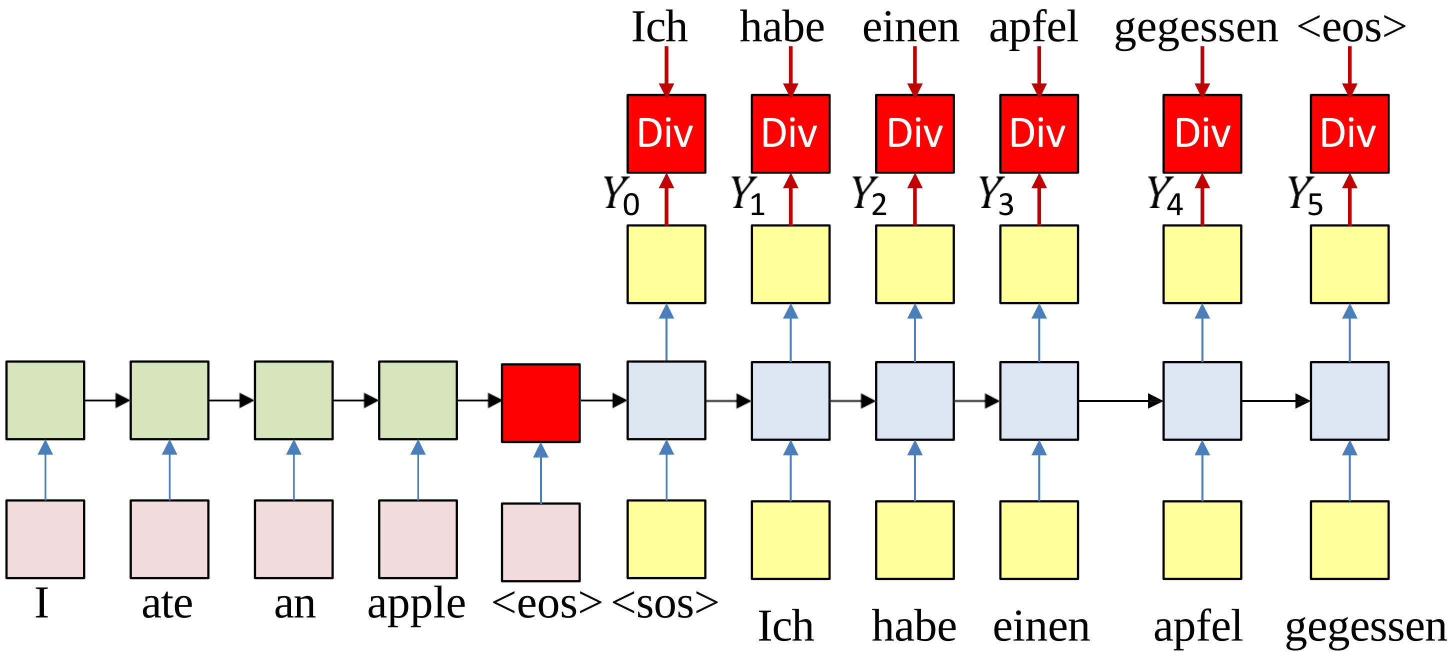
- Must learn to make predictions appropriately
  - Given “I ate an apple <eos>”, produce “Ich habe einen apfel gegessen <eos>”.

Training : Forward pass



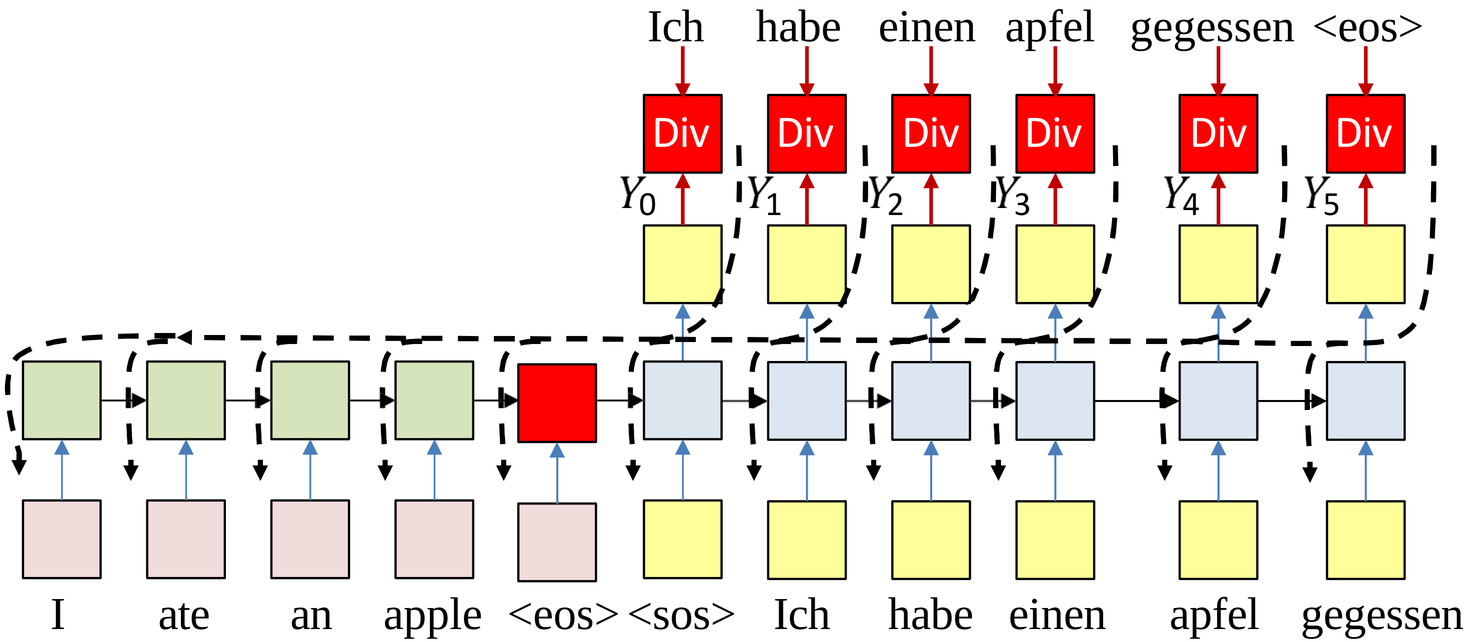
- Forward pass: Input the source and target sequences, sequentially
  - Output will be a probability distribution over target symbol set (vocabulary)

Training : Backward pass



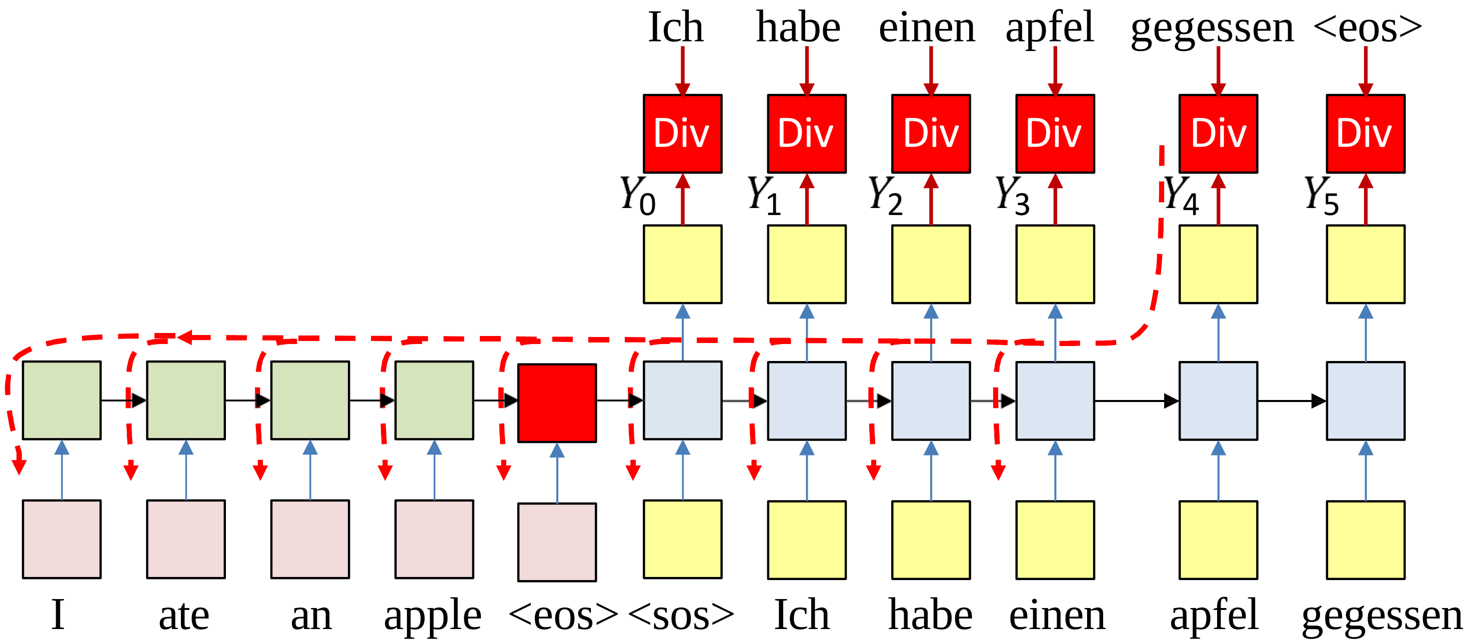
- Backward pass: Compute the divergence between the output distribution and target word sequence

Training : Backward pass



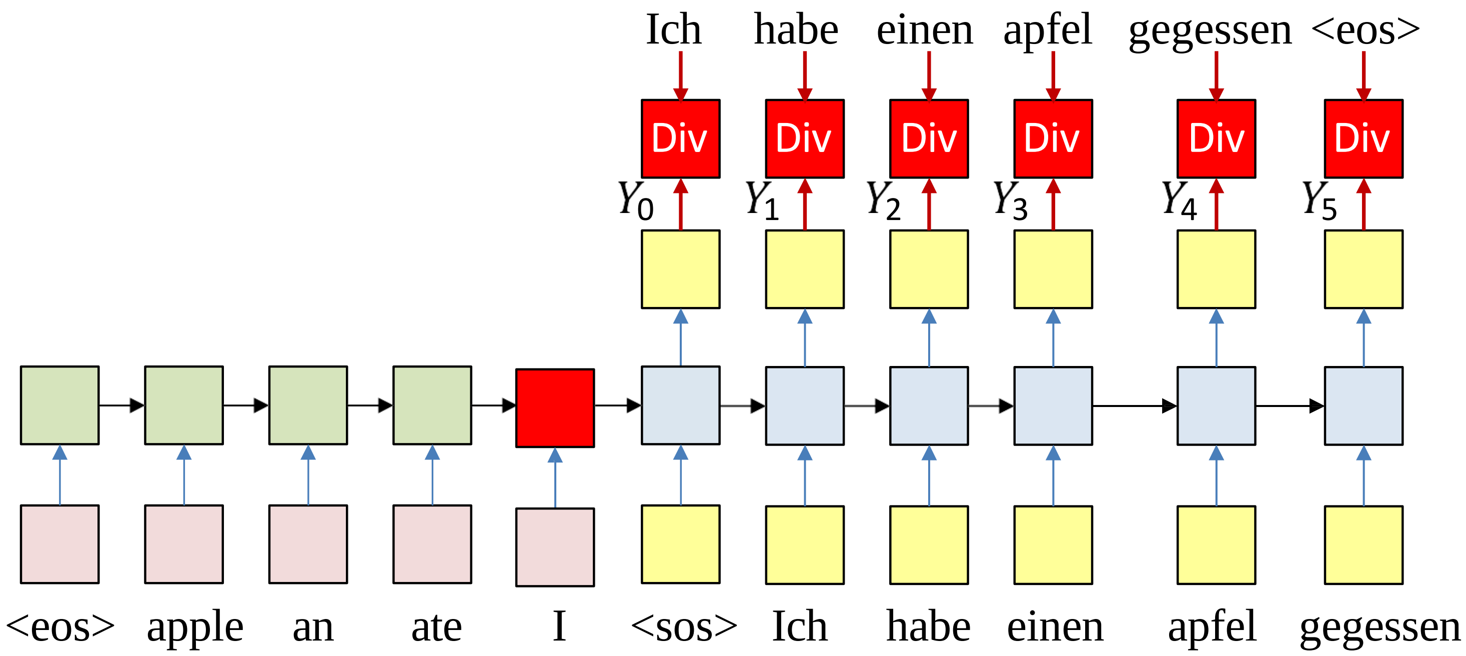
- Backward pass: Compute the divergence between the output distribution and target word sequence
- Backpropagate the derivatives of the divergence through the network to learn the net

Training : Backward pass



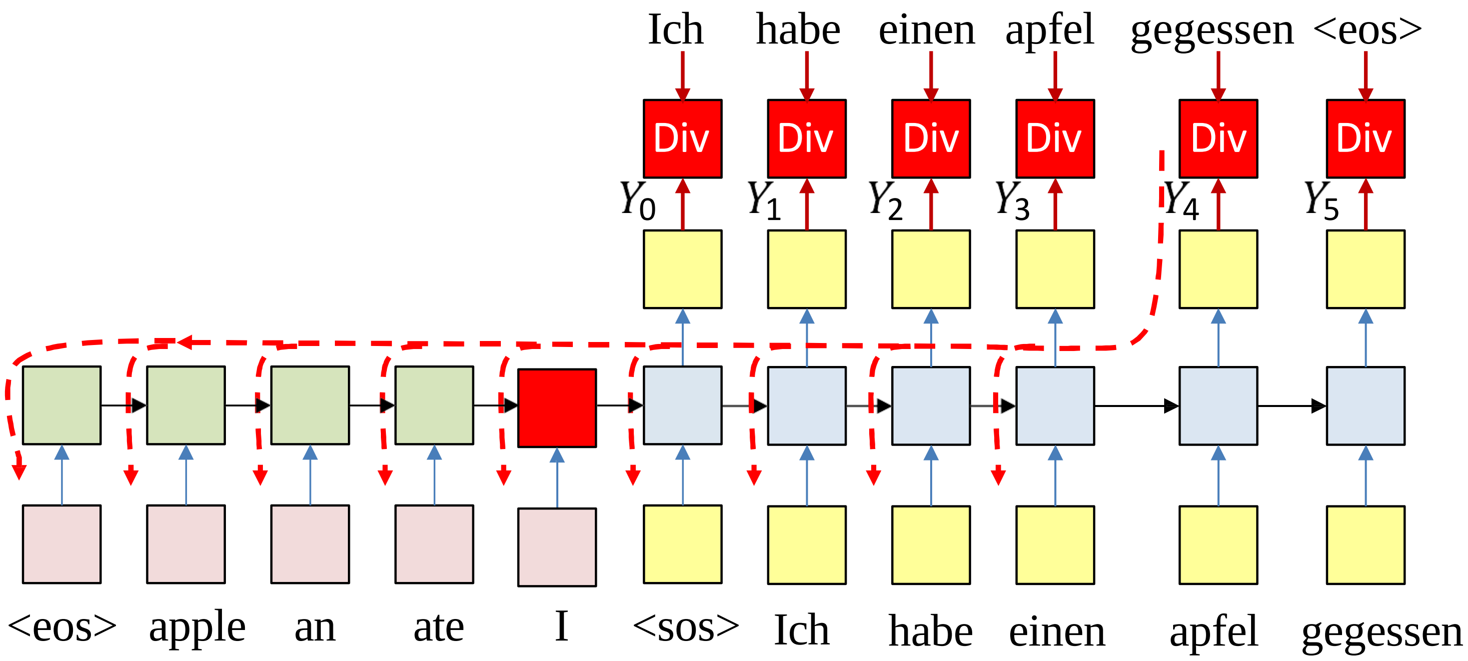
- In practice, if we apply SGD, we may randomly sample words from the output to actually use for the backprop and update
  - Typical usage: Randomly select one word from each input training instance (comprising an input-output pair)
    - For each iteration
      - Randomly select training instance: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

Trick of the trade: Reversing the input



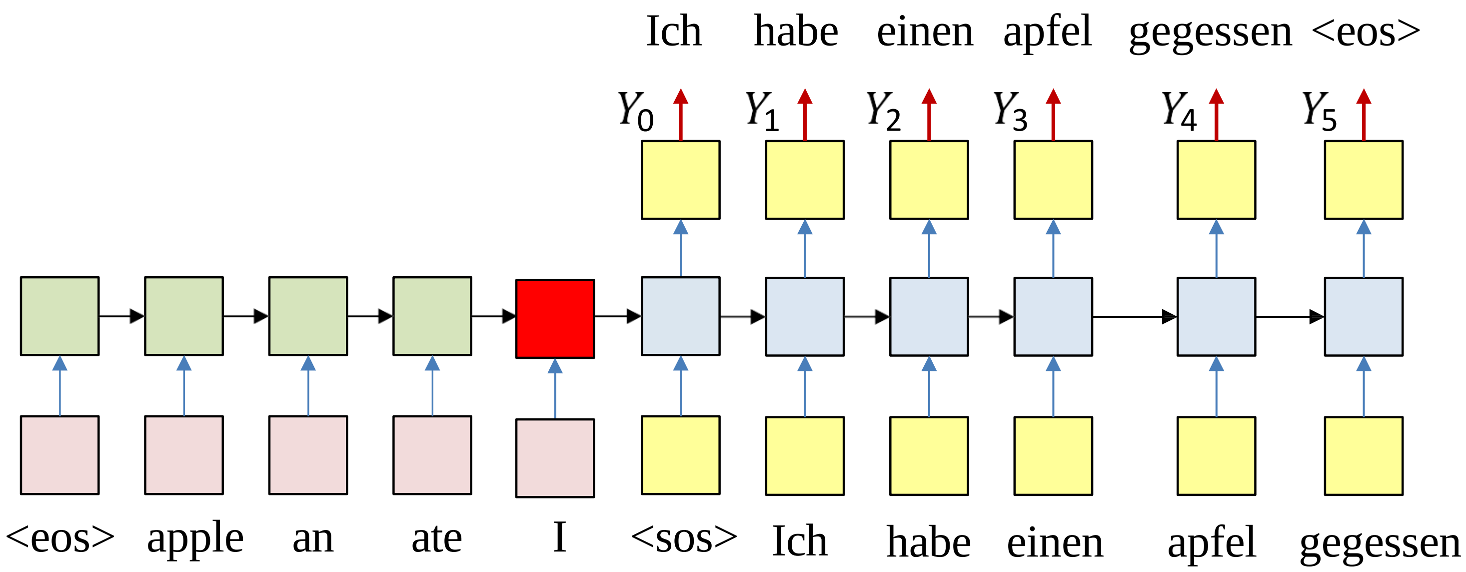
- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way

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

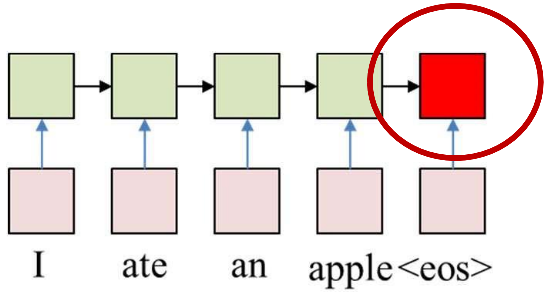


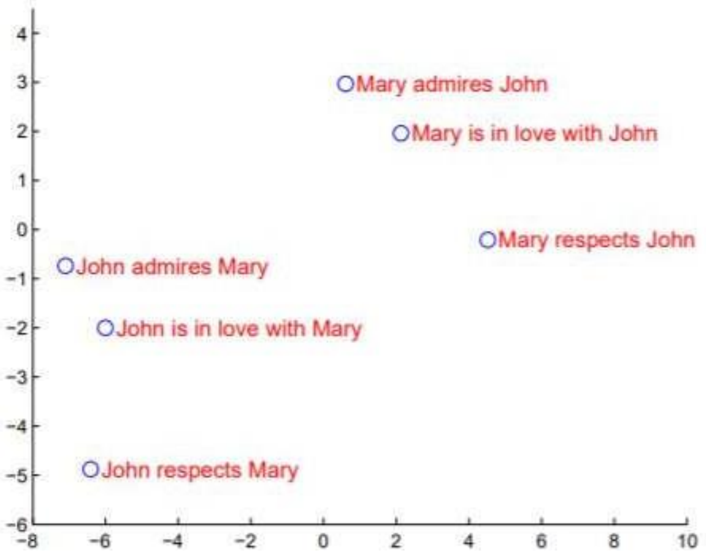
- Standard trick of the trade: The input sequence is fed *in reverse order*
  - Things work better this way
- *This happens both for training and during inference on test data*

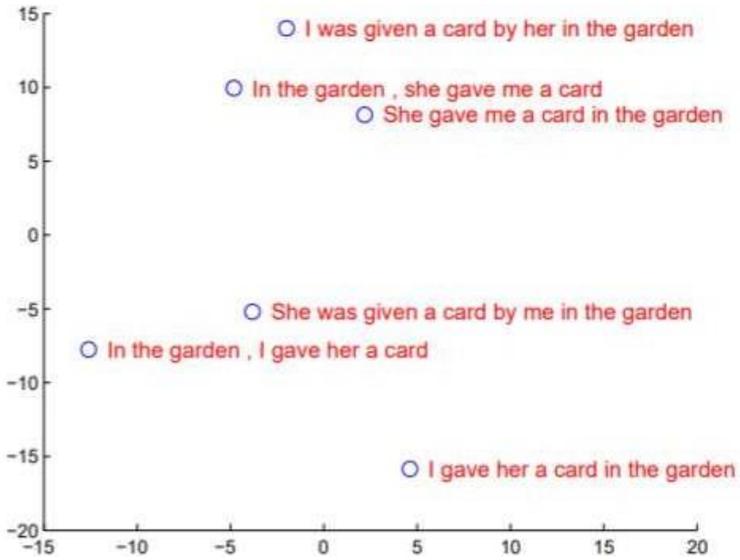
Applications

- 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







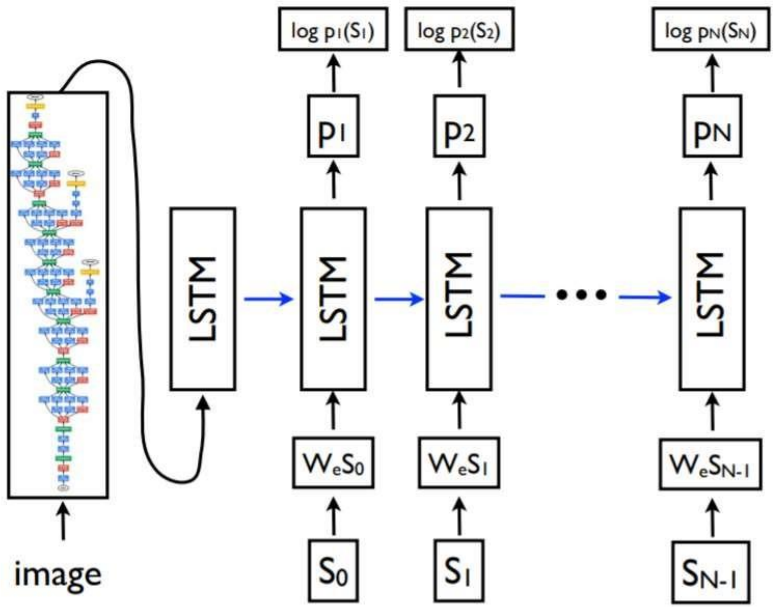
- Hidden state clusters by meaning!
  - From “Sequence-to-sequence learning with neural networks”, Sutskever, Vinyals and Le

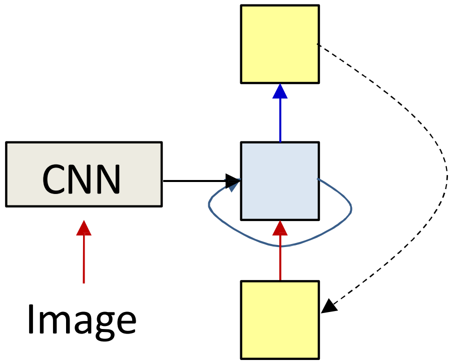
Human Machine Conversation: Example

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

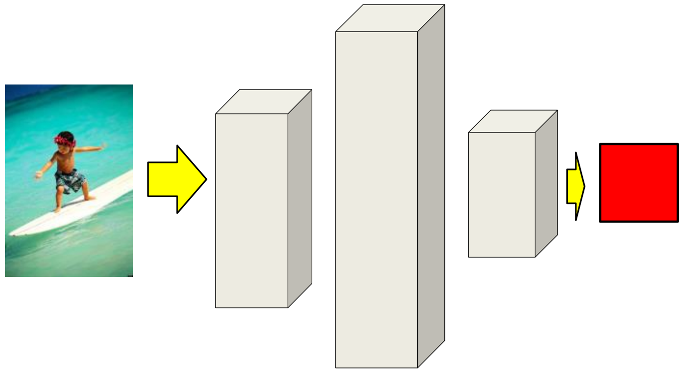
Generating Image Captions





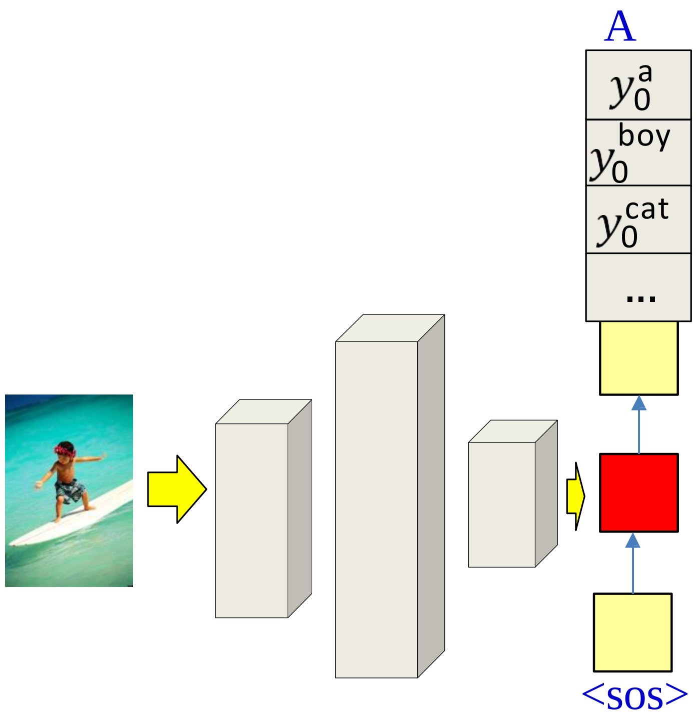
- Not really a seq-to-seq problem, more an image-to-sequence problem
- Initial state is produced by a state-of-art CNN-based image classification system
  - Subsequent model is just the decoder end of a seq-to-seq model
    - "Show and Tell: A Neural Image Caption Generator", O. Vinyals, A. Toshev, S.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

Generating Image Captions



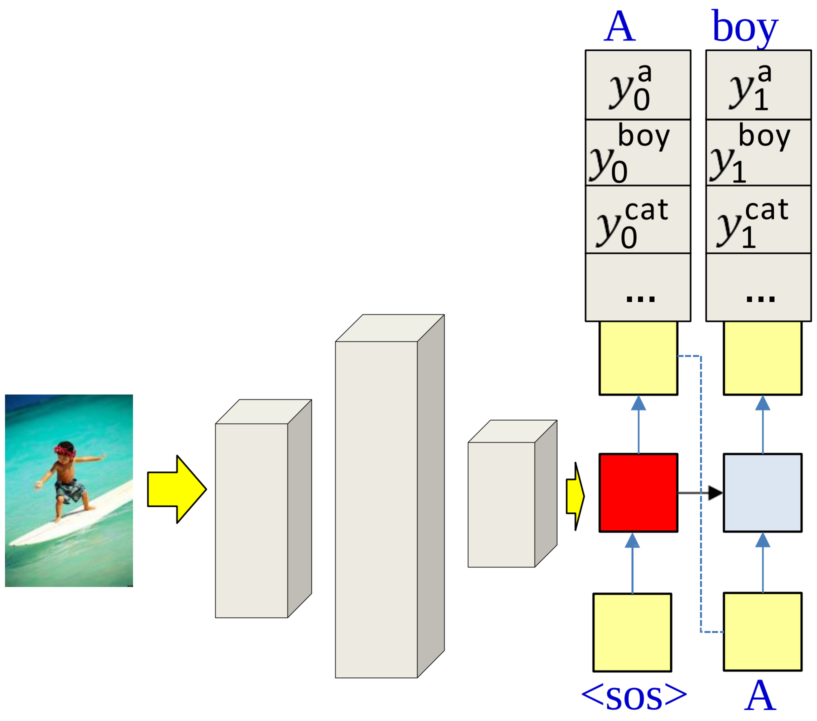
- Decoding: Given image
  - Process it with CNN to get output of classification layer

Generating Image Captions



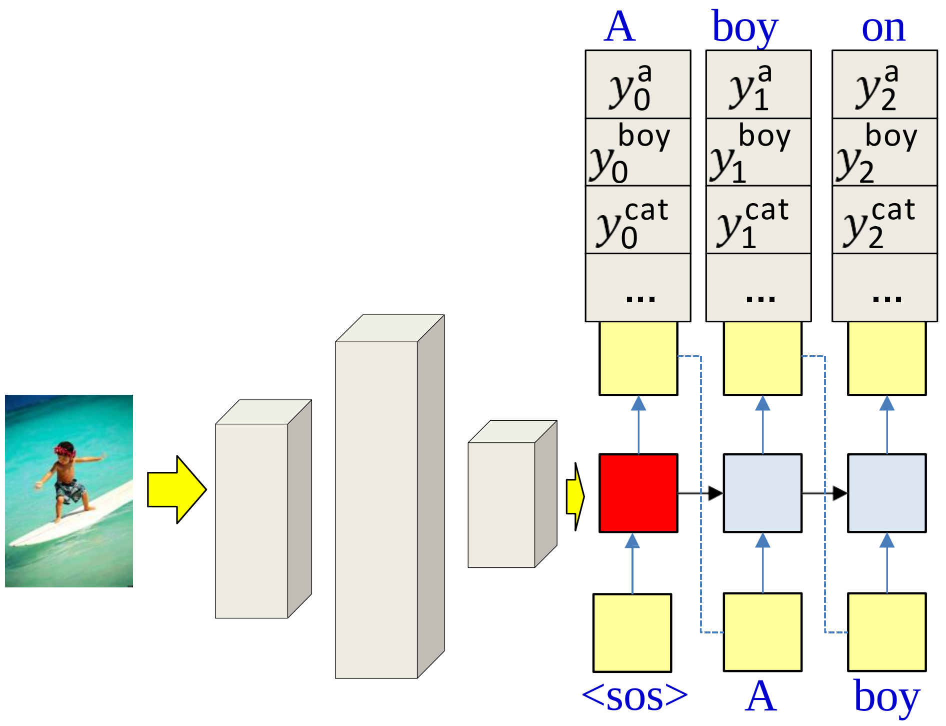
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \dots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



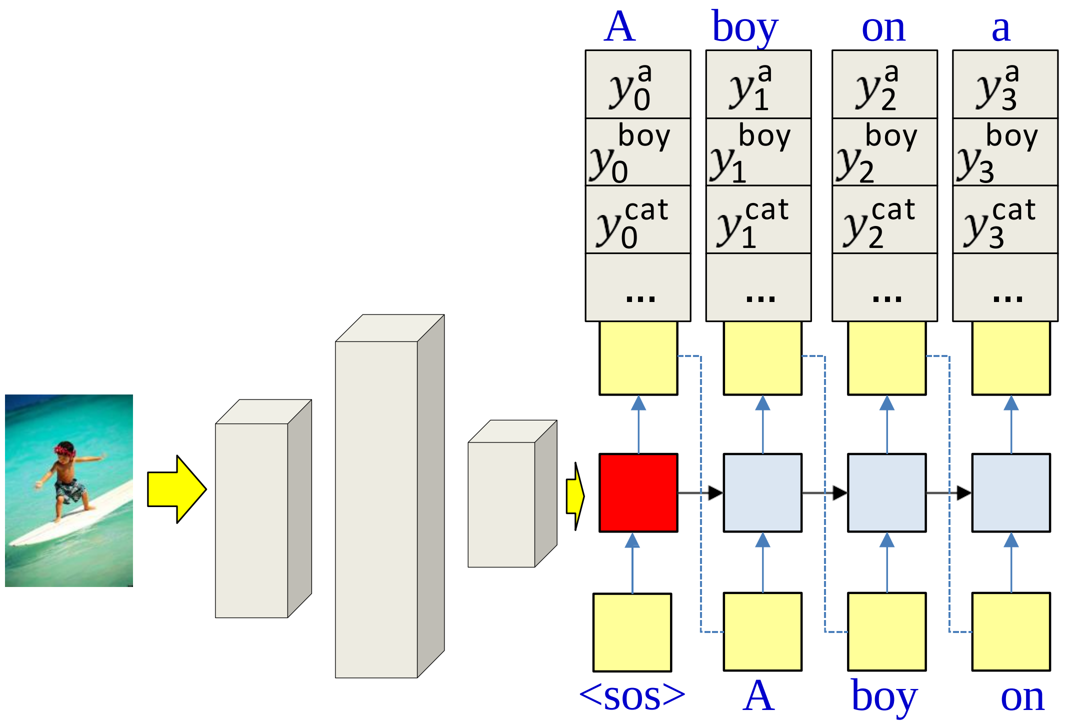
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \dots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



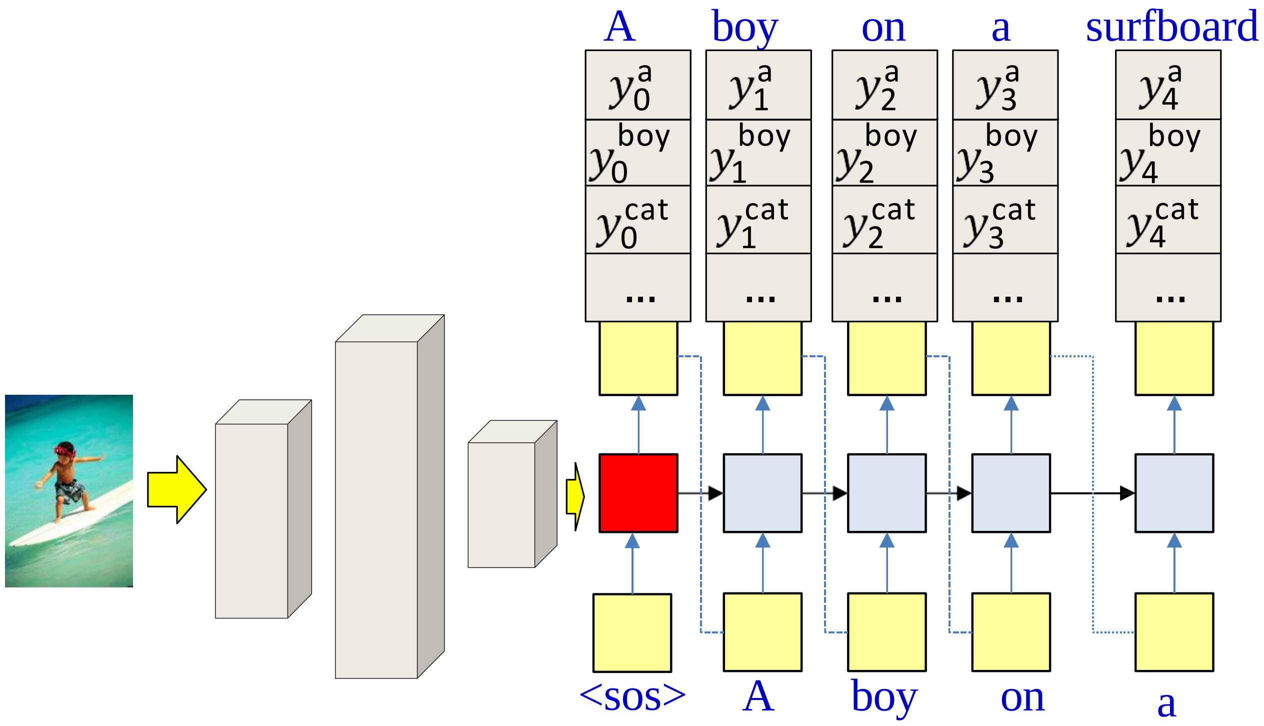
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \dots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



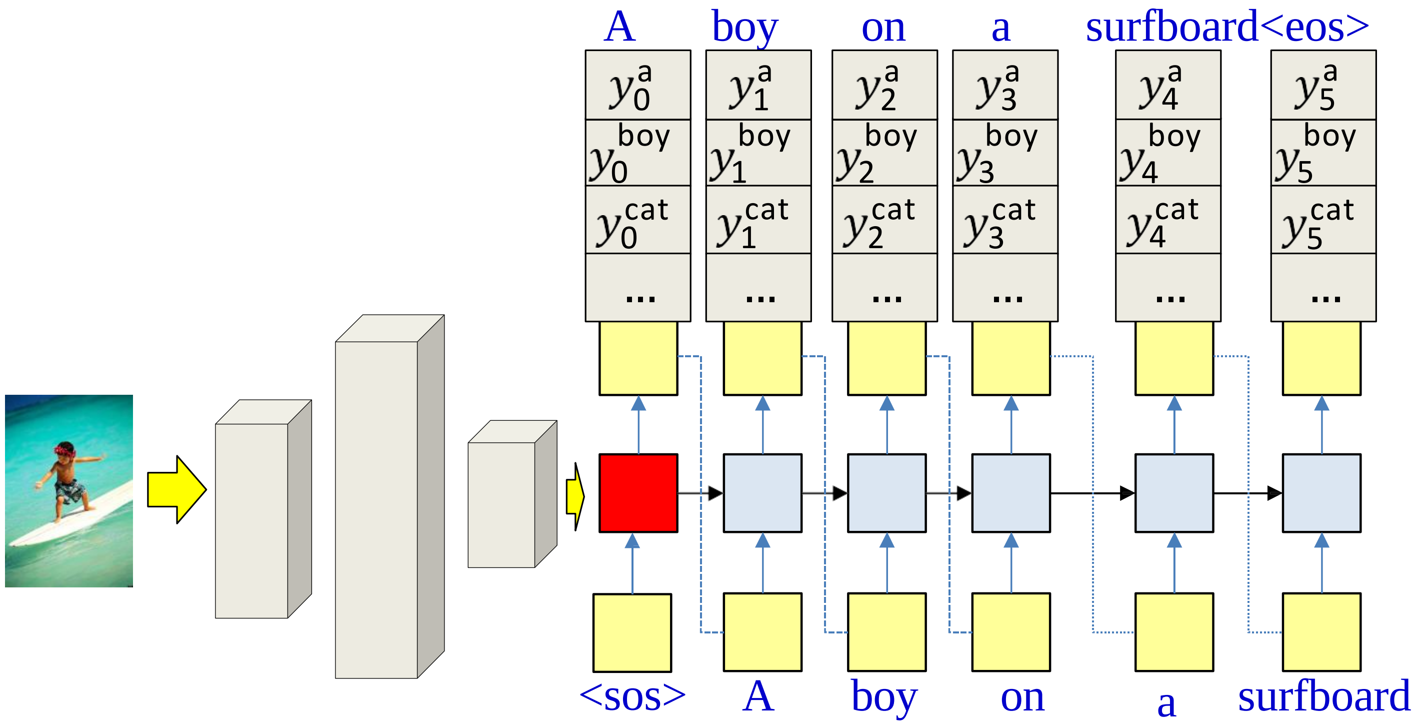
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Generating Image Captions



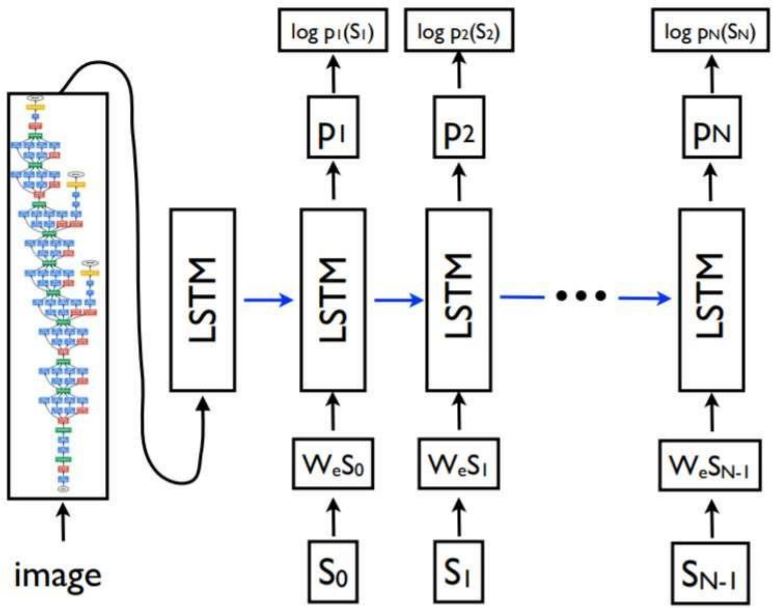
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

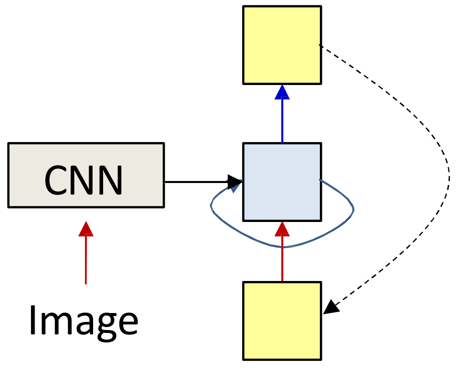
Generating Image Captions



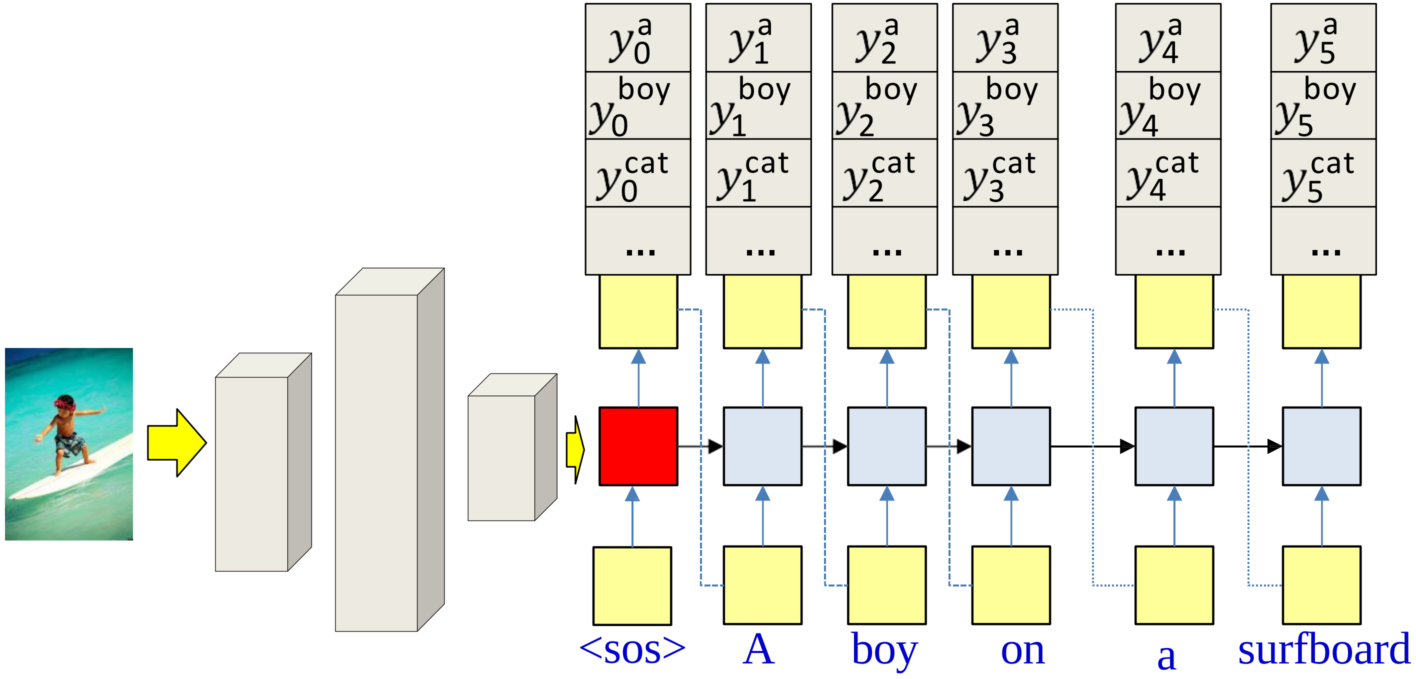
- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

Training

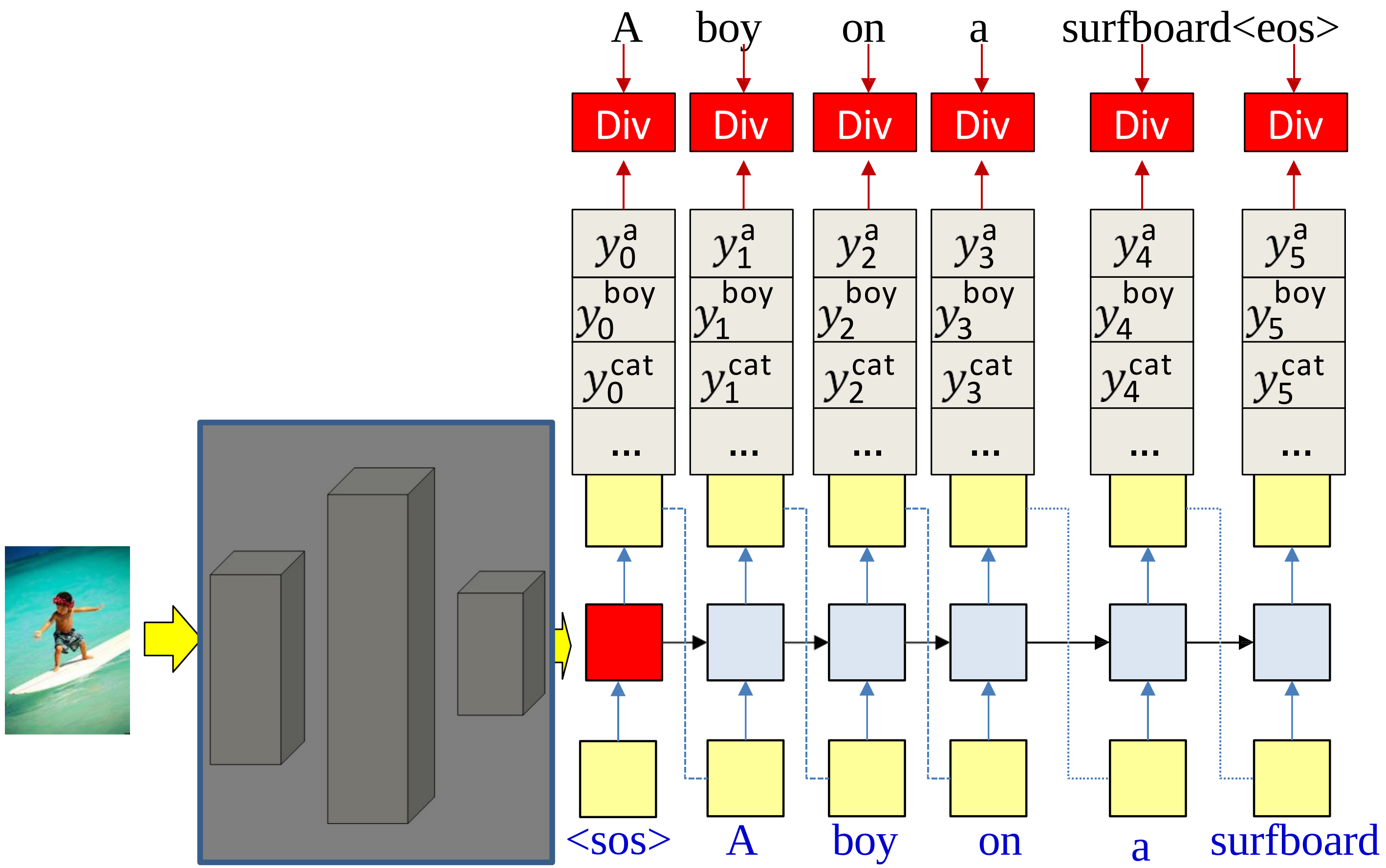




- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net



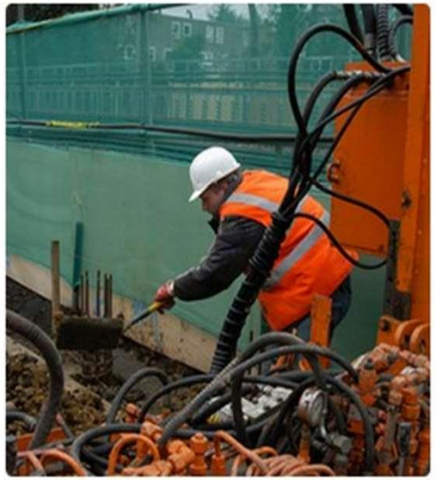
- **Training**: Given several (Image, Caption) pairs
  - The image network is pretrained on a large corpus, e.g. image net
- **Forward pass:** Produce output distributions given the image and caption



- **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."

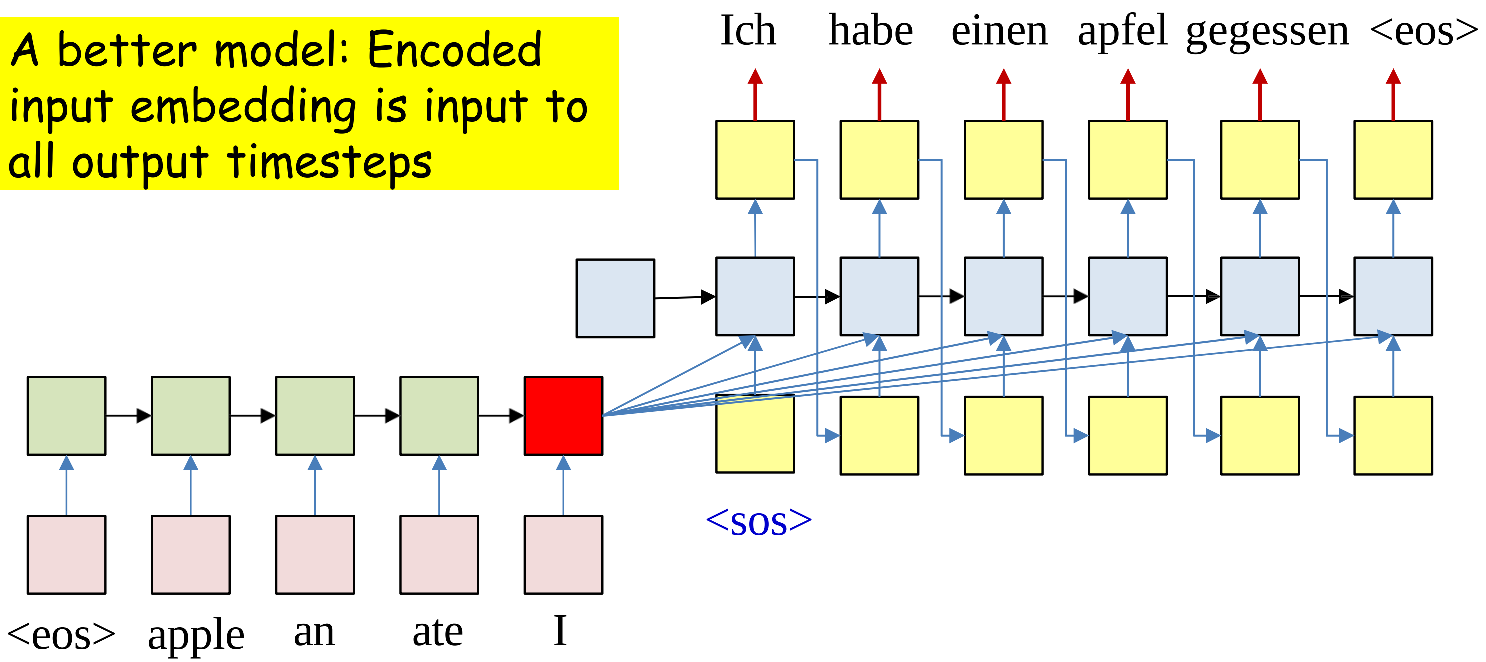
"a cat is sitting on a couch with a remote control."

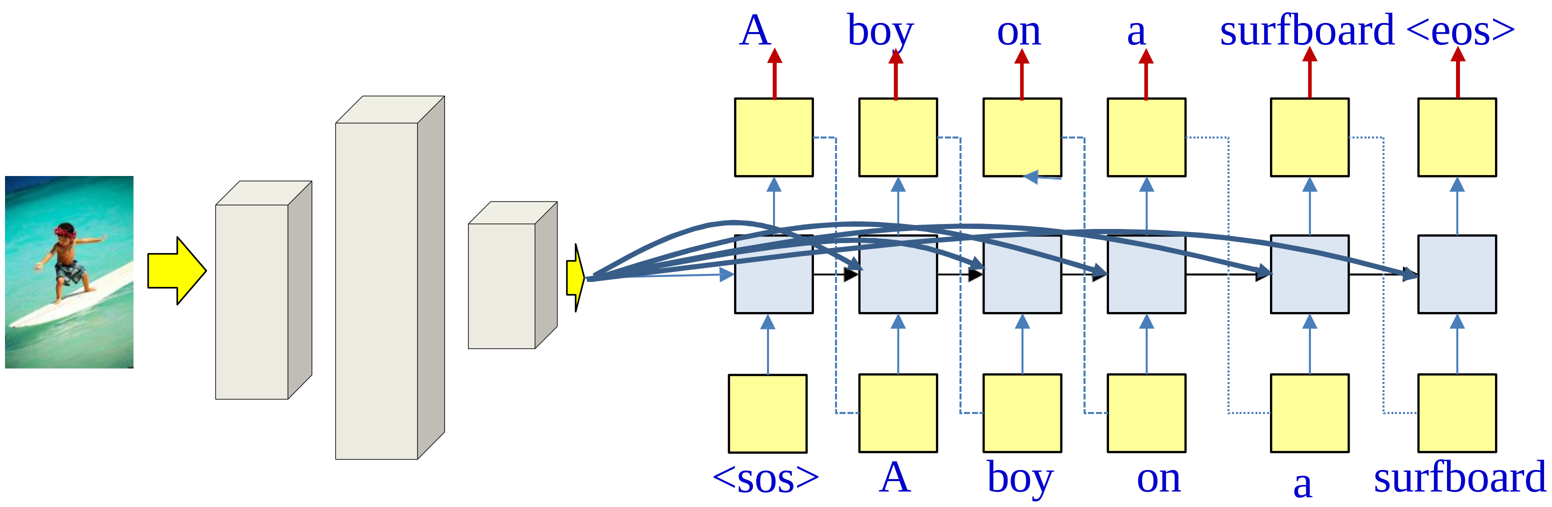
"a woman holding a teddy bear in front of a mirror."

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

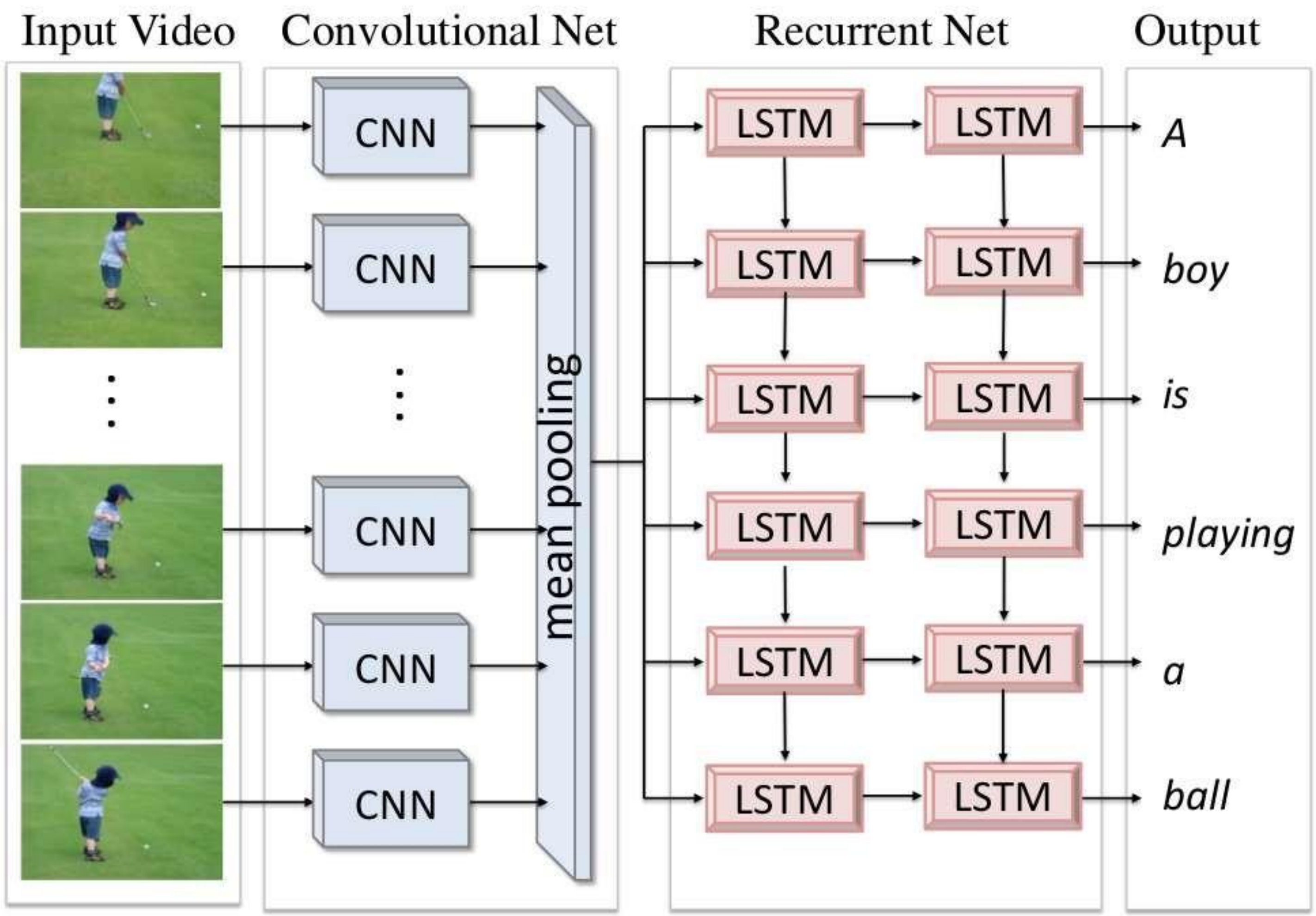
Variants

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.