Seq2Seq + Attention

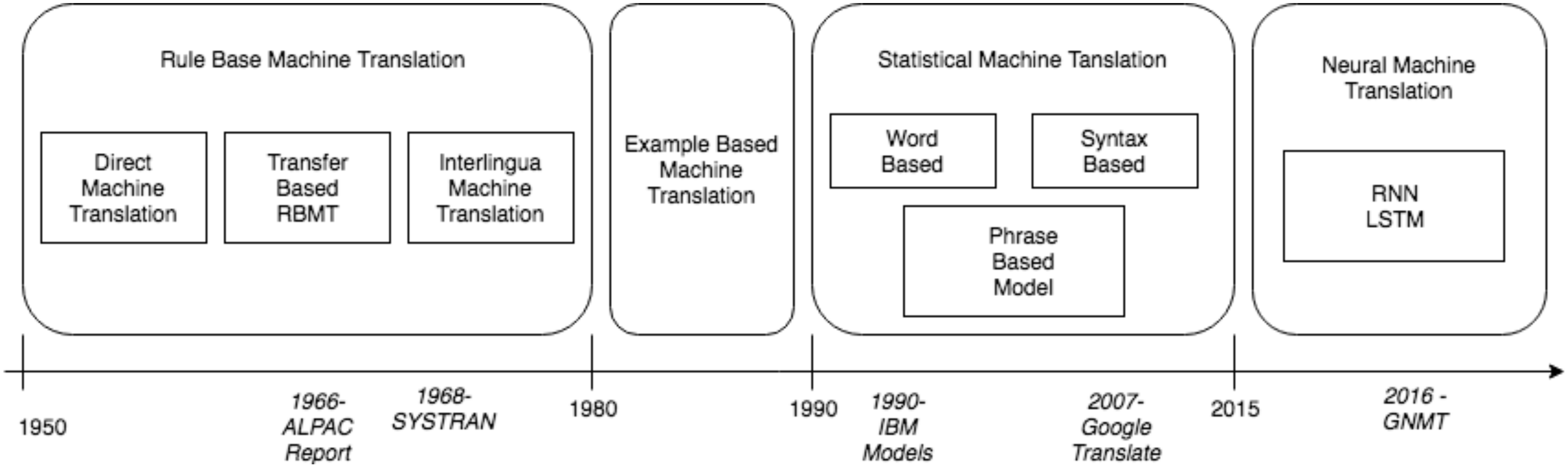
Wei Xu

(many slides from Greg Durrett)

This Lecture

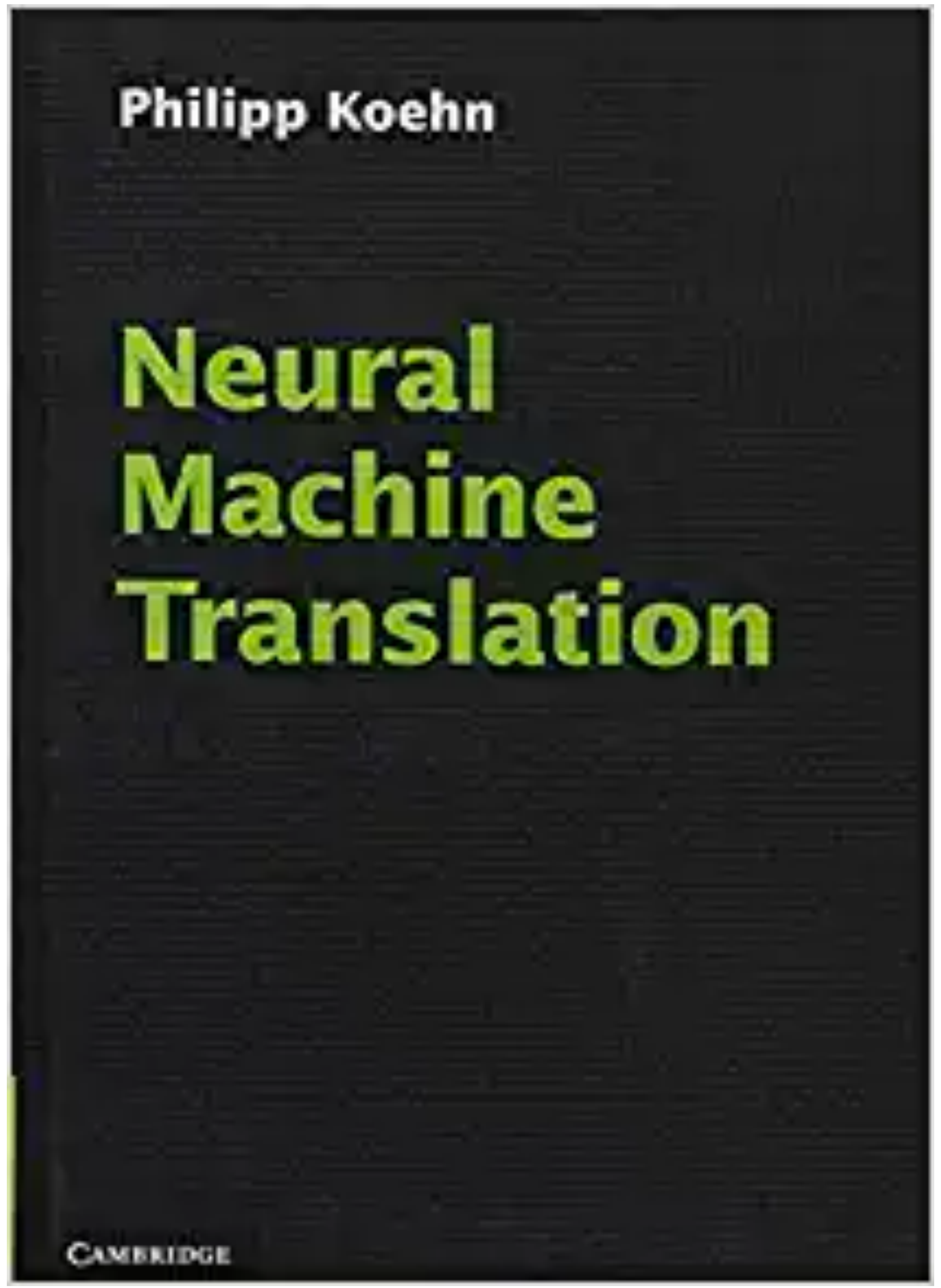
- Sequence-to-Sequence Model
- Attention Mechanism
- Neural MT & Other Applications (if time)
- Midterm Review

Recap: History of MT

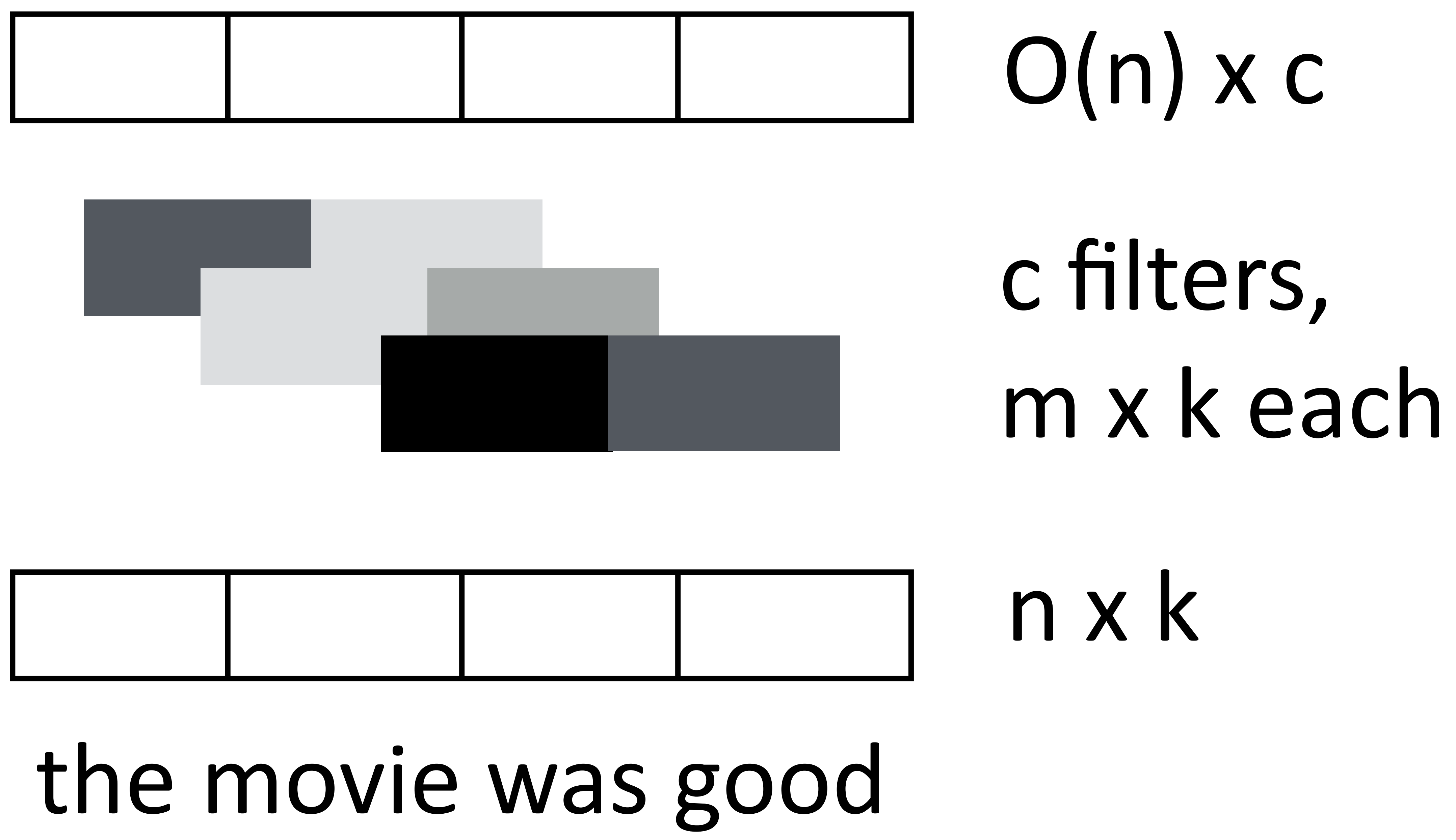

Rule Base Machine Translation
Direct Machine Translation
Transfer Based RBMT
Interlingua Machine Translation
Example Based Machine Translation
Statistical Machine Tanslation
Word Based
Syntax Based
Phrase Based Model
Neural Machine Translation
RNN LSTM
1950
1966- ALPAC Report
1968- SYSTRAN
1980
1990
1990- IBM Models
2007- Google Translate
2015
2016 - GNMT

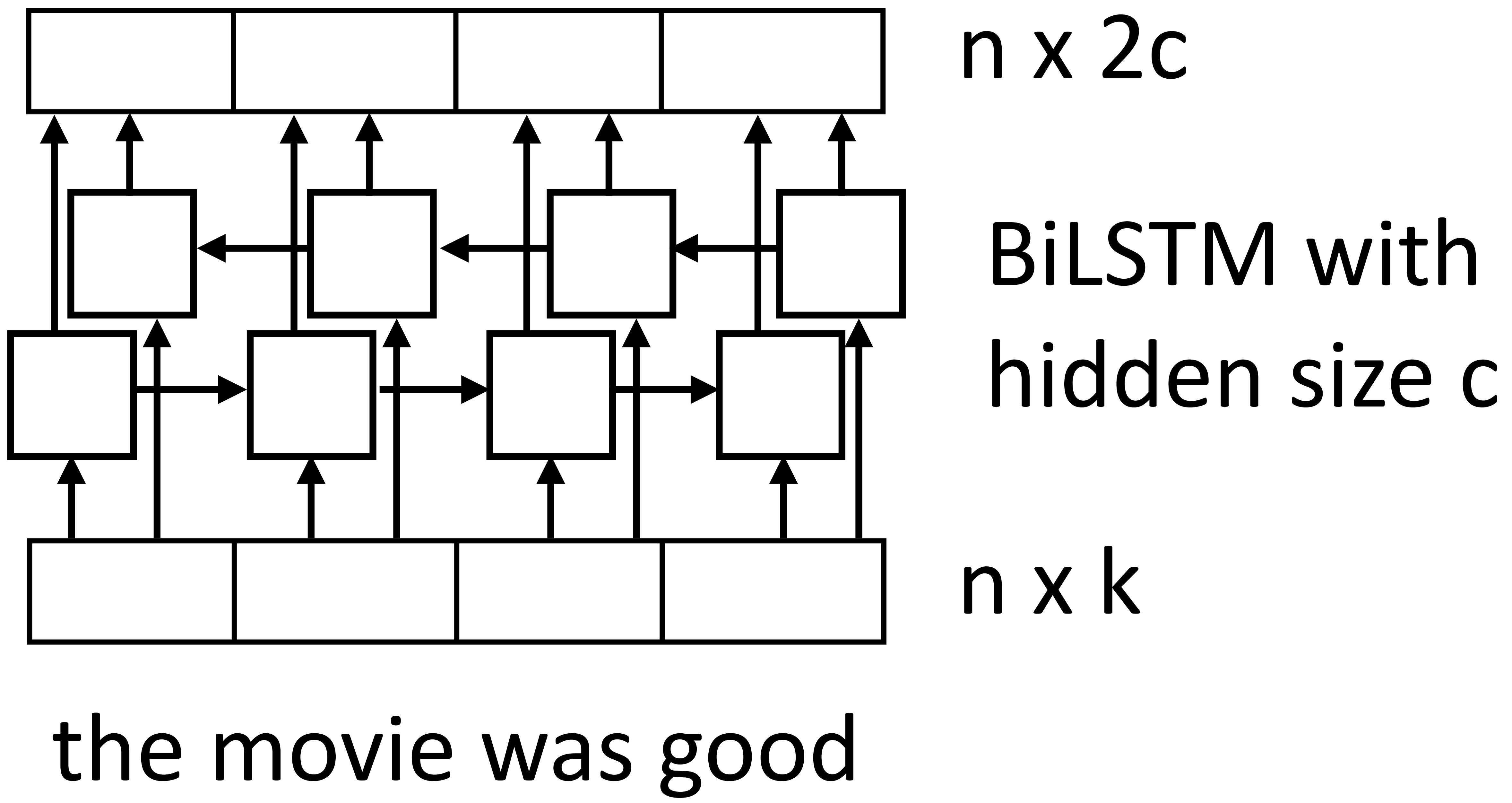
Administrivia

- Reading — Eisenstein 18.3-18.5
- Additional Reading — http://mt-class.org/jhu/
- Course Project Proposal is due 10/12
- No classes on 10/12 and 10/19



Recall: CNNs vs. LSTMs

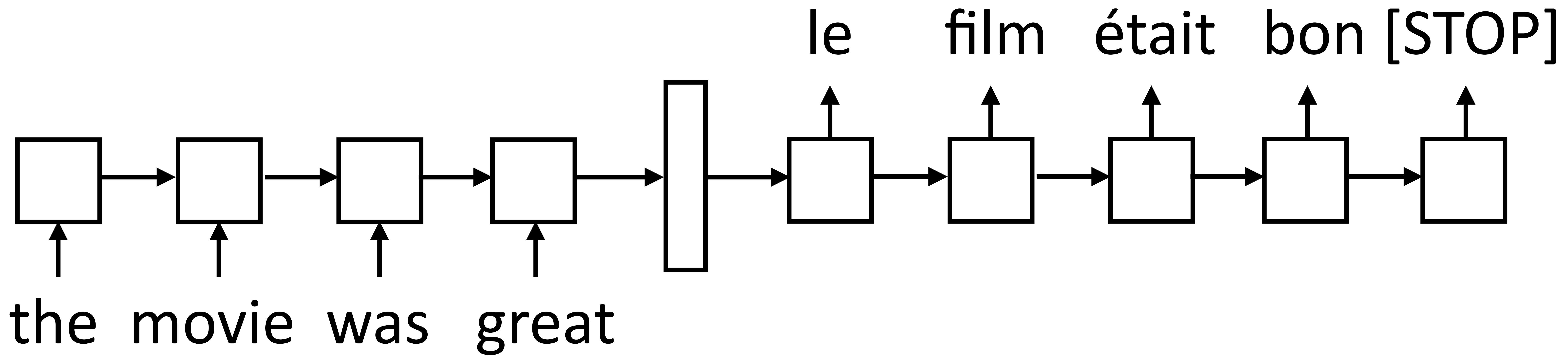




- Both LSTMs and convolutional layers transform the input using context
- LSTM: "globally" looks at the entire sentence (but local for many problems)
- CNN: local depending on filter width + number of layers

Encoder-Decoder

- Encode a sequence into a fixed-sized vector



- Now use that vector to produce a series of tokens as output from a separate LSTM *decoder*
- Machine translation, NLG, summarization, dialog, and many other tasks (e.g., semantic parsing, syntactic parsing) can be done using this framework.

Sutskever et al. (2014)

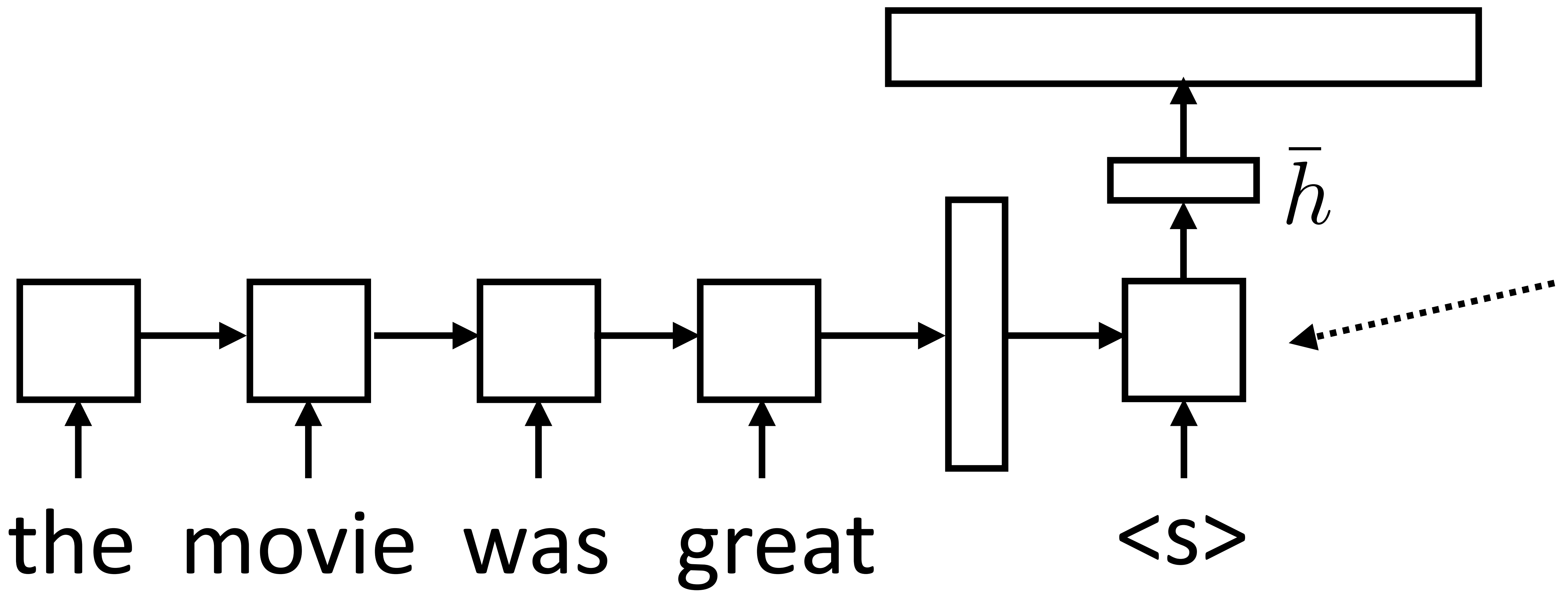
Model

- Generate next word conditioned on previous word as well as hidden state

- W size is |vocab| x |hidden state|, softmax over entire vocabulary

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

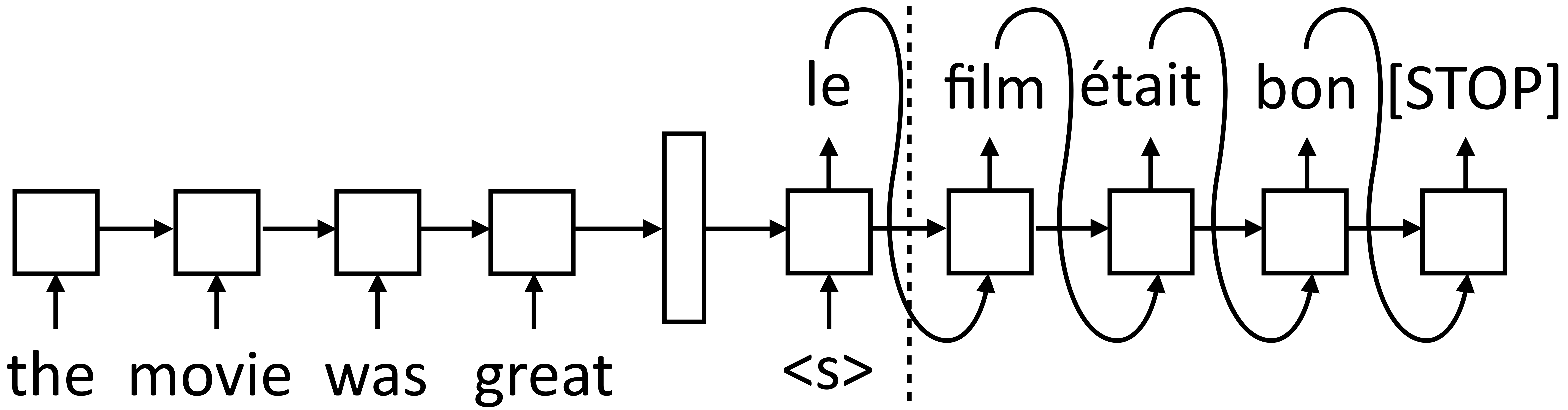
$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n} P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1})$$



Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)

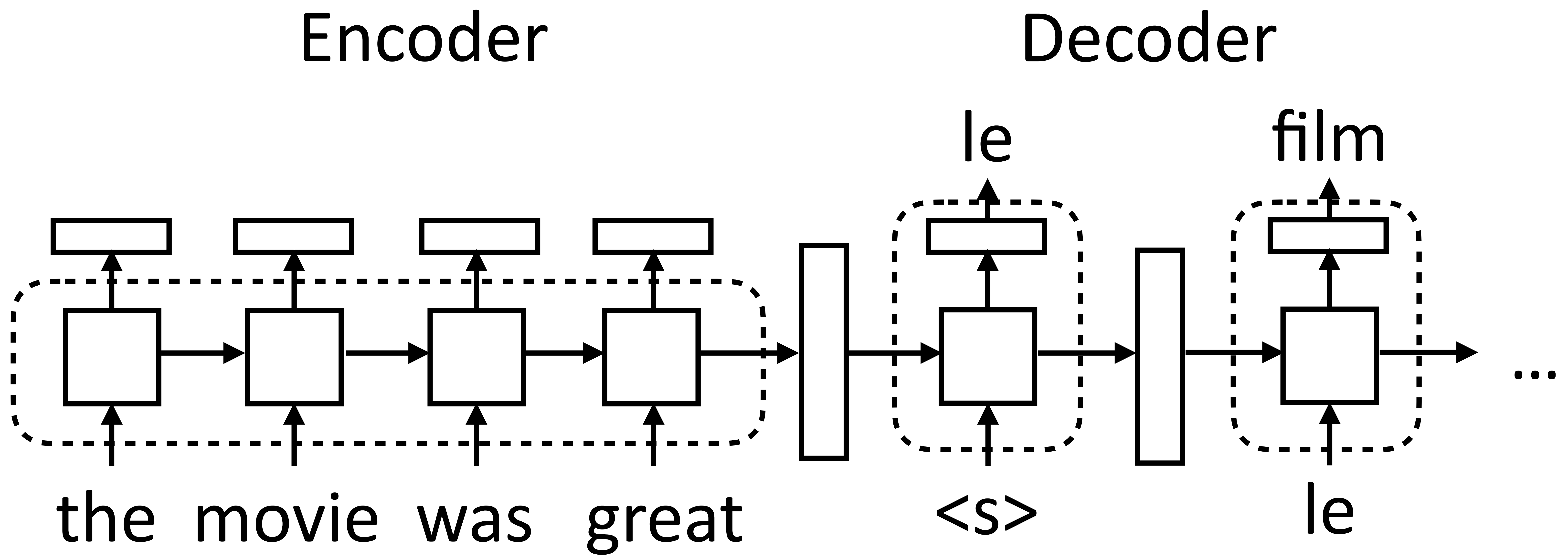
Inference

- Generate next word conditioned on previous word as well as hidden state



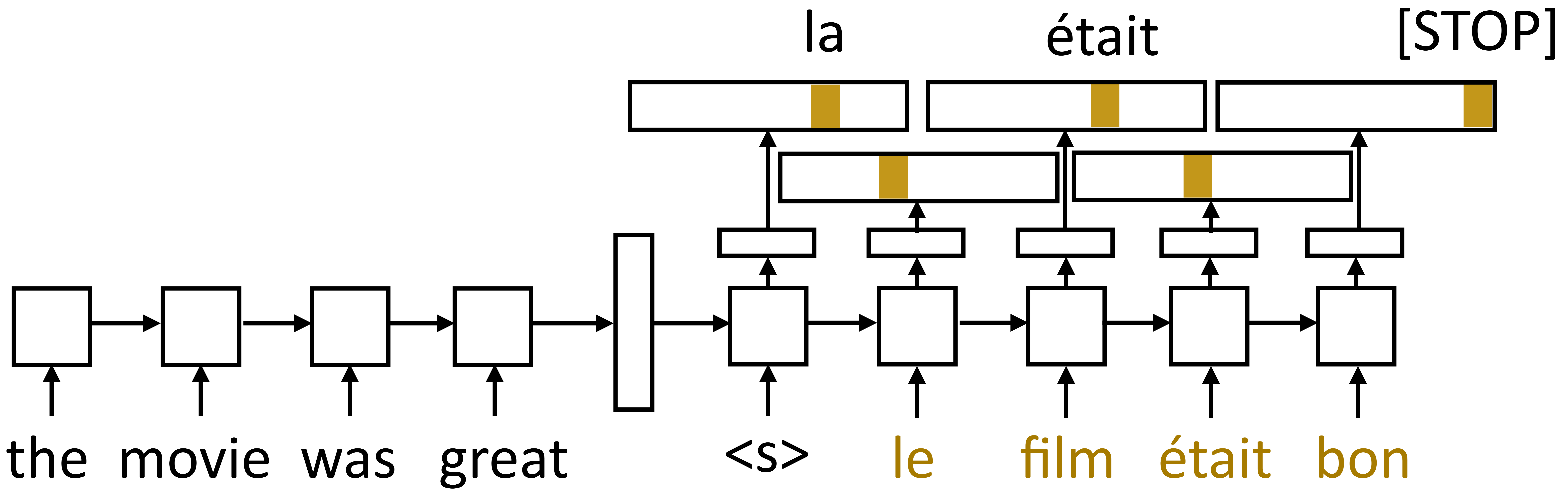
- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state
- Need to actually evaluate computation graph up to this point to form input for the next state
- Decoder is advanced one state at a time until [STOP] is reached

Implementing seq2seq Models



- Encoder: consumes sequence of tokens, produces a vector. Analogous to encoders for classification/tagging tasks
- Decoder: separate module, single cell. Takes two inputs: hidden state (vector *h* or tuple (*h*, *c*)) and previous token. Outputs token + new state

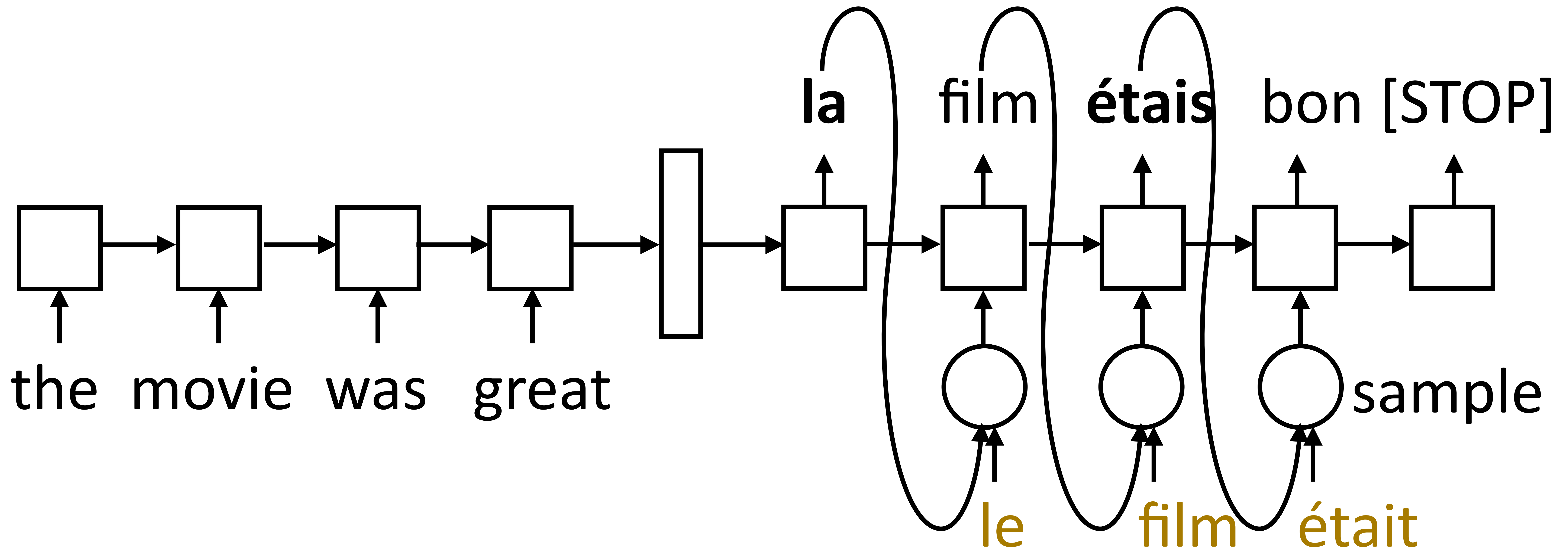
Training



- Objective: maximize $\sum_{(\mathbf{x},\mathbf{y})} \sum_{i=1}^{n} \log P(y_i^* | \mathbf{x}, y_1^*, \ldots, y_{i-1}^*)$
- One loss term for each target-sentence word, feed the correct word regardless of model's prediction (this is what called "teacher forcing")

Training: Scheduled Sampling

- Model needs to do the right thing even with its own predictions



- Scheduled sampling: with probability *p*, take the gold (human) translation as input, else take the model's prediction
- Starting with *p* = 1 and decaying it works best

Bengio et al. (2015)

Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length

- Encoder: Can be a CNN/LSTM/Transformer…

- Batching is a bit tricky:
  - encoder should use pack_padded_sequence to handle different lengths.
  - The decoder should pad everything to the same length and use a mask to only accumulate “valid” loss terms

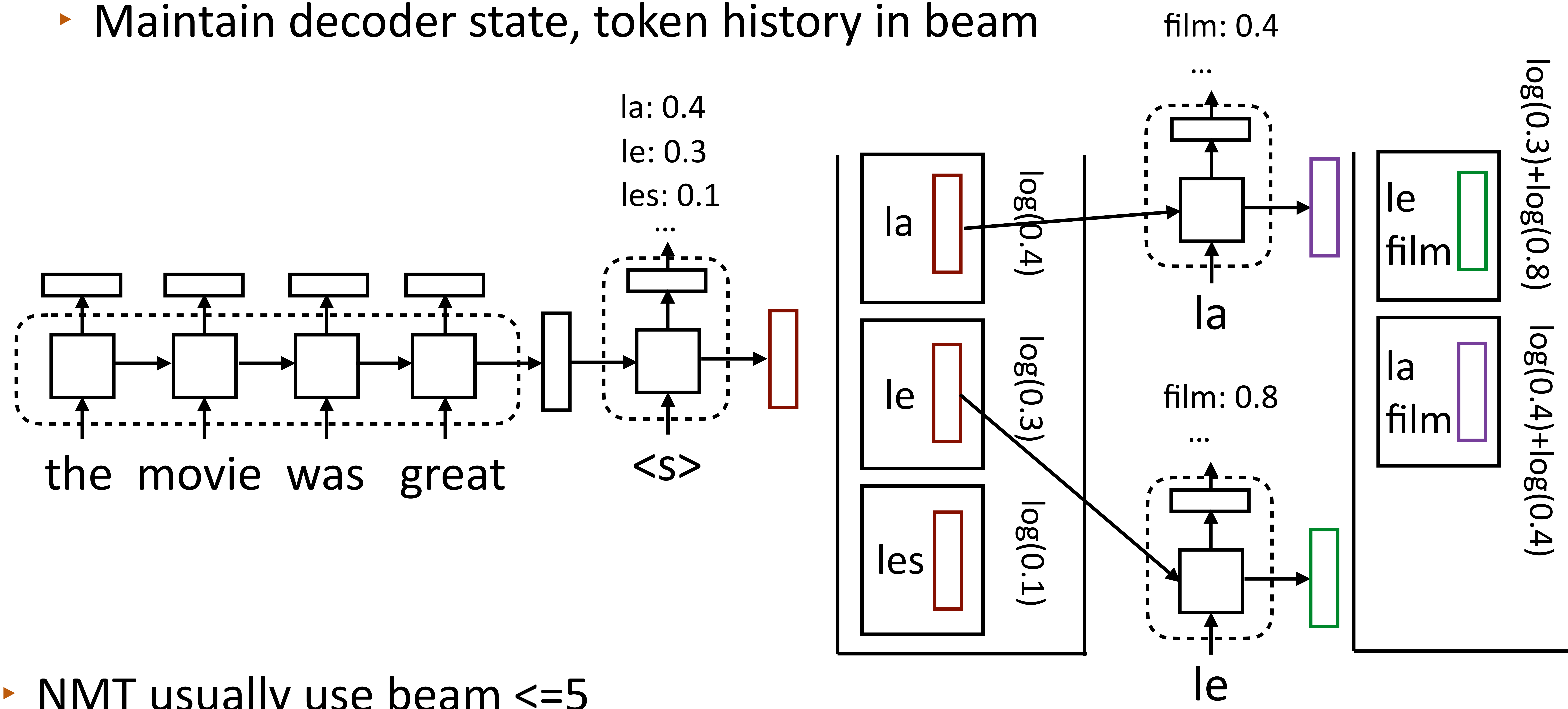
Implementation Details (cont')

- Decoder:
  - Test time: execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state. Until reach <STOP>.
  - Training time: you can execute all timesteps as part of one computation graph
- Beam search: can help with lookahead. Finds the (approximate) highest scoring sequence:

$$\text{argmax}_{\mathbf{y}} \prod_{i=1}^{n} P(y_i | \mathbf{x}, y_1, \ldots, y_{i-1})$$

Beam Search

- Maintain decoder state, token history in beam



- NMT usually use beam <=5
- Keep **both** *film* states! Hidden state vectors are different

Meister et al. (2020)

Attention

Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

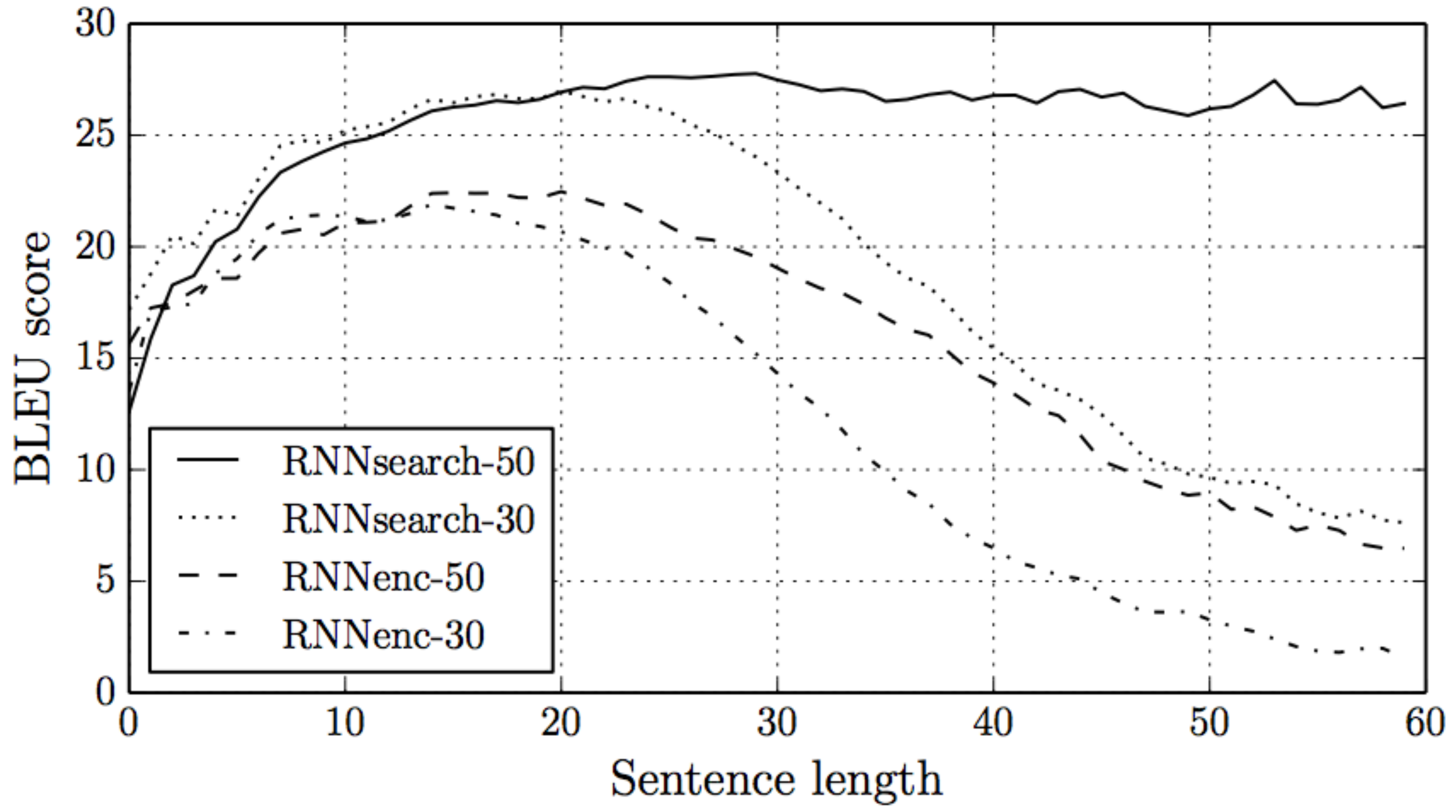
Un garçon joue dans la neige → *A boy plays in the snow* ***boy plays boy plays***

- Often a byproduct of training these models poorly. Input is forgotten by the LSTM so it gets stuck in a “loop” of generation the same output tokens again and again.

- Need some notion of input coverage or what input words we’ve translated

Problems with Seq2seq Models

- Bad at long sentences: 1) a fixed-size hidden representation doesn't scale; 2) LSTMs still have a hard time remembering for really long sentences



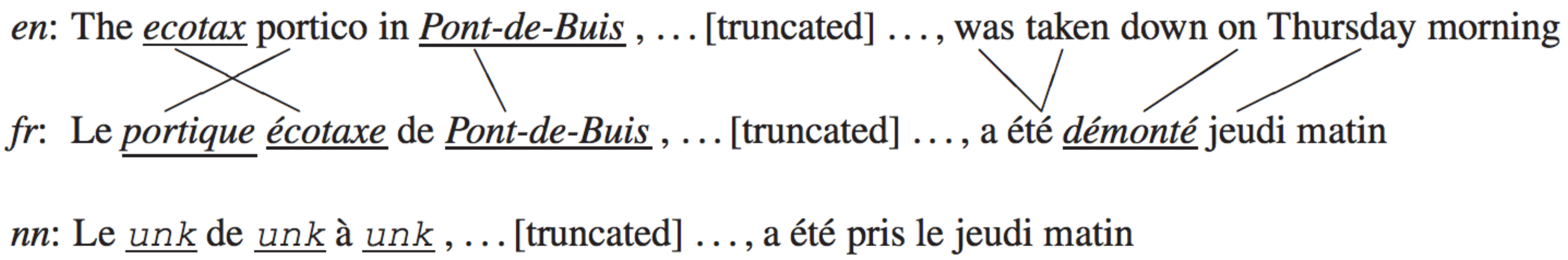
RNNenc: the model we've discussed so far

RNNsearch: uses attention

Bahdanau et al. (2014)

Problems with Seq2seq Models

- Unknown words:



en: The *ecotax* portico in *Pont-de-Buis* , . . . [truncated] . . . , was taken down on Thursday morning

fr: Le *portique* *écotaxe* de *Pont-de-Buis* , . . . [truncated] . . . , a été *démonté* jeudi matin

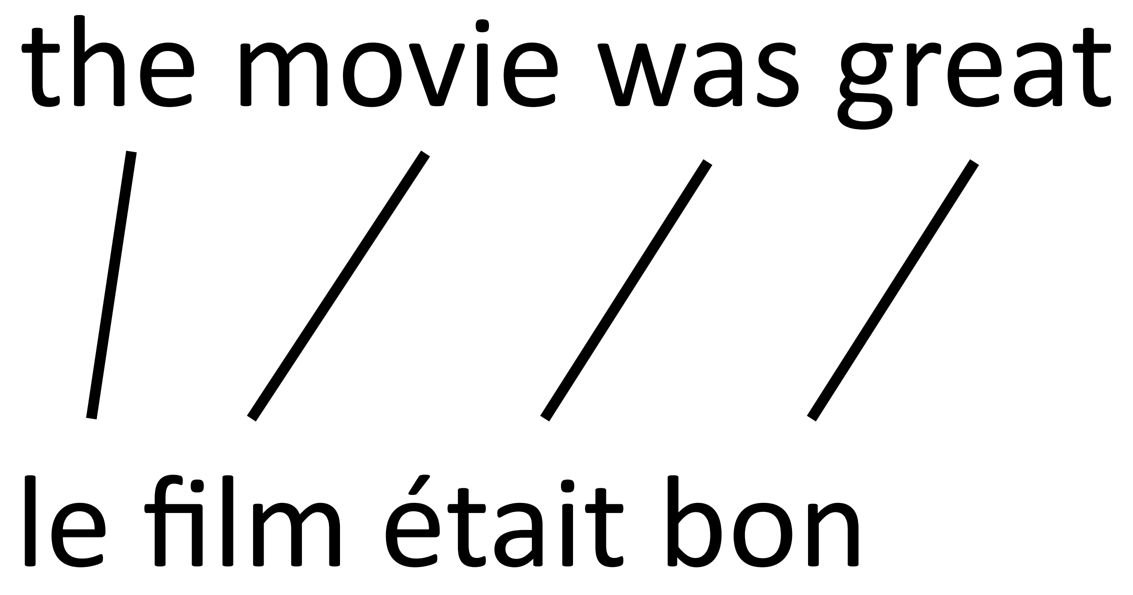
nn: Le unk de unk à unk , . . . [truncated] . . . , a été pris le jeudi matin

- Encoding these rare words into a vector space is really hard
- In fact, we don't want to encode them, we want a way of directly looking back at the input and copying them (Pont-de-Buis)

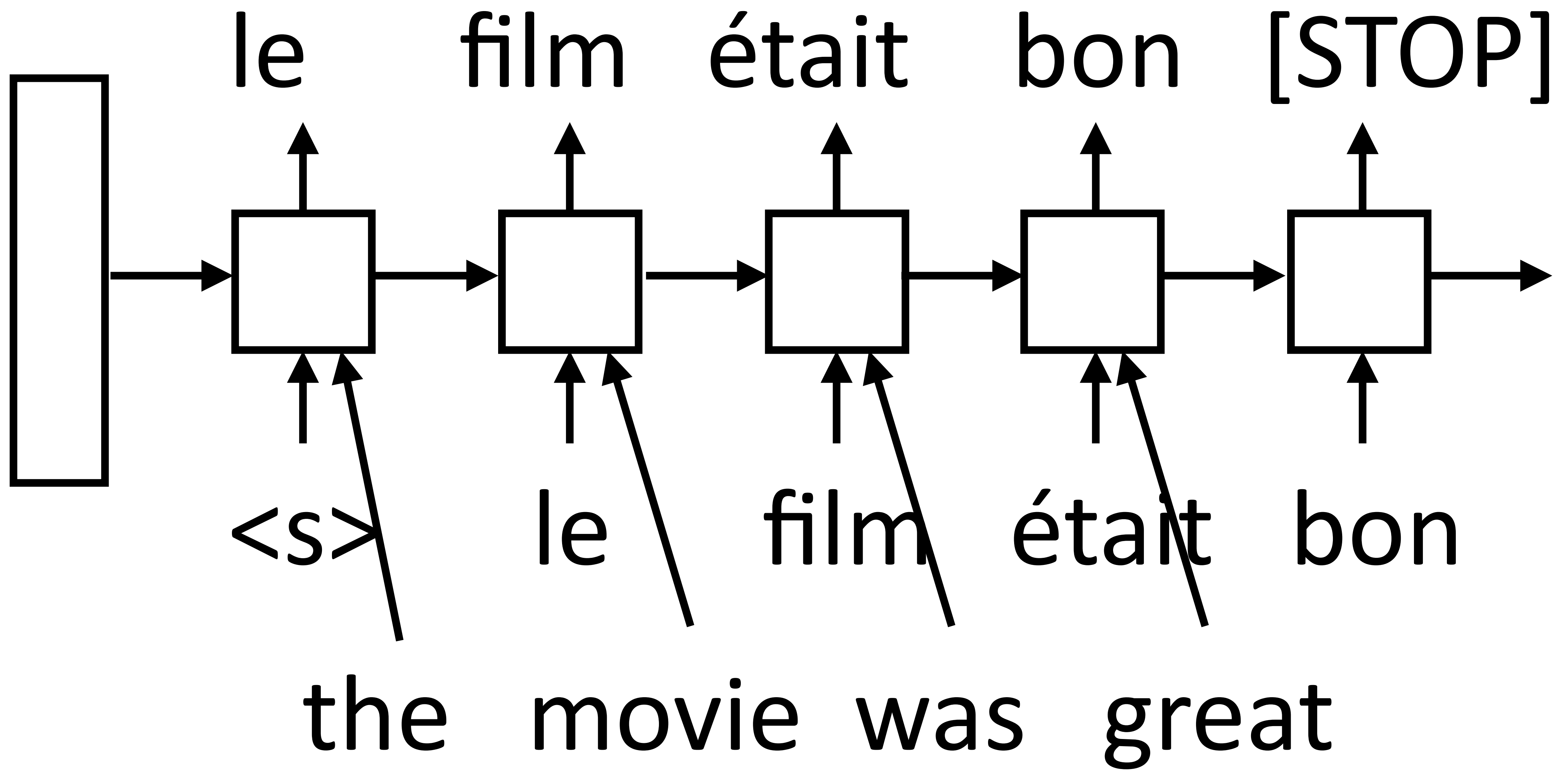
Jean et al. (2015), Luong et al. (2015)

Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated (recall the word alignment we talked about in phrase-based MT)

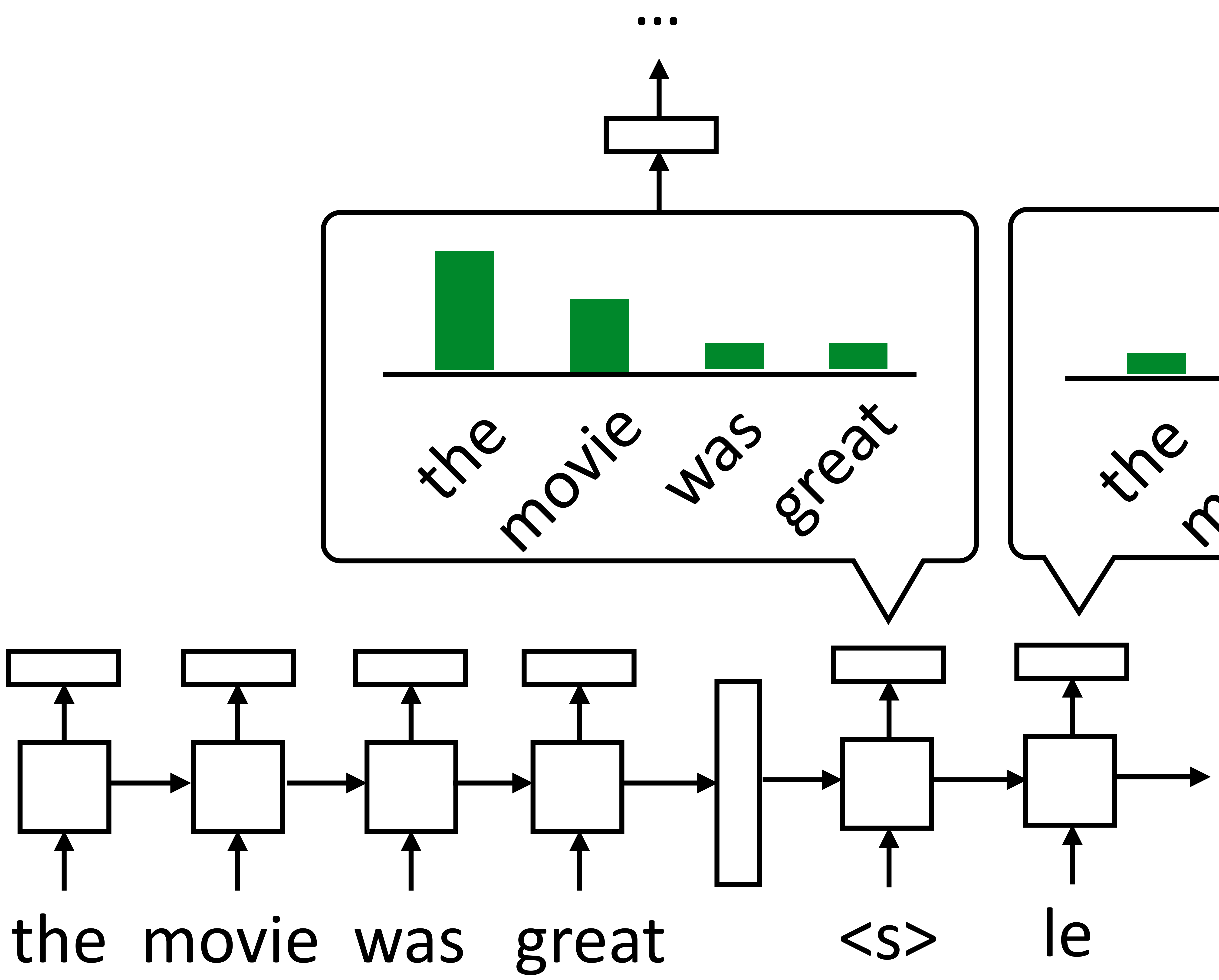


- Can look at the corresponding input word when translating — this could scale!



- How can we achieve this without hardcoding it?

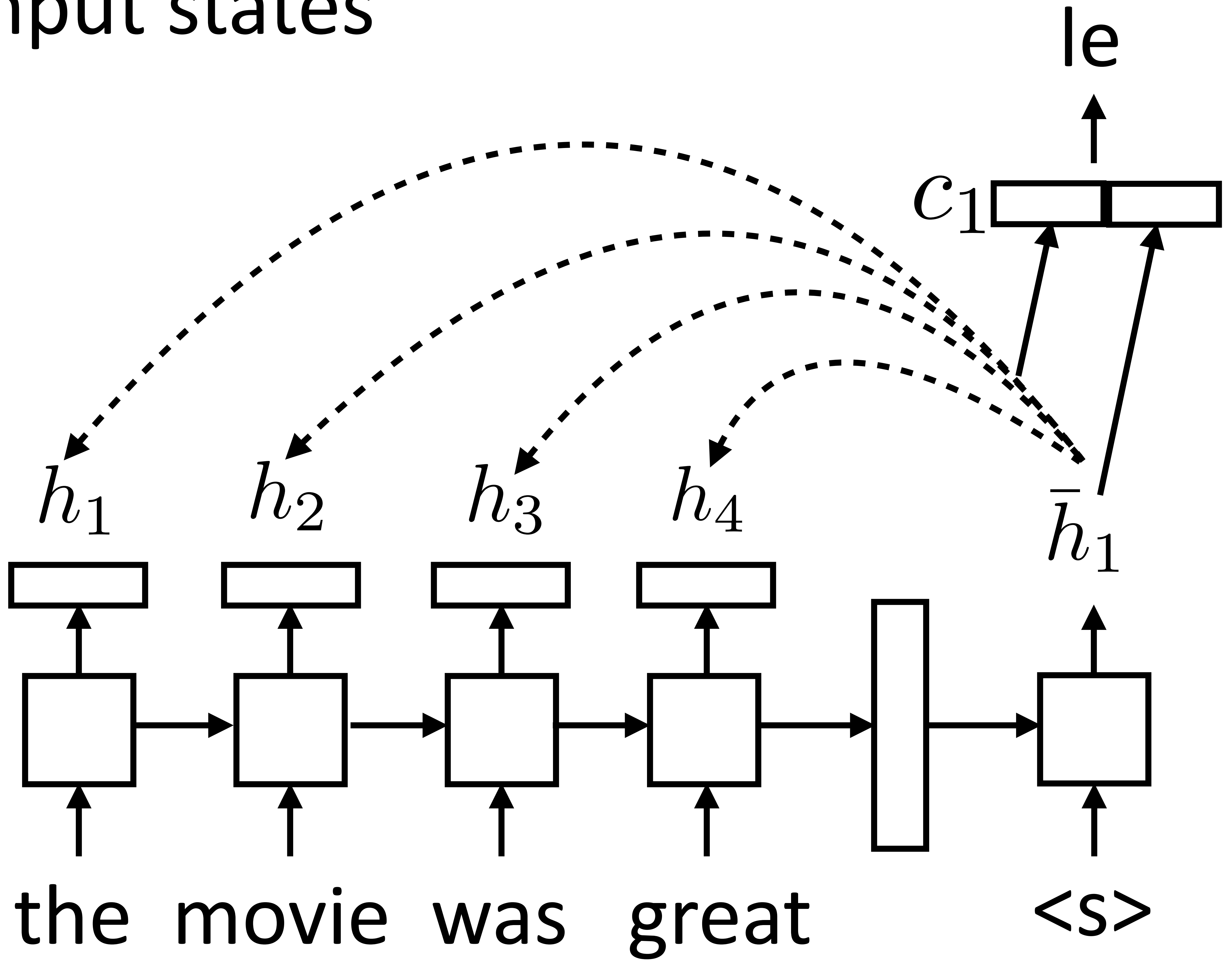
Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state
- Use the weighted sum of input tokens to predict output

Attention

- For each decoder state, compute weighted sum of input states



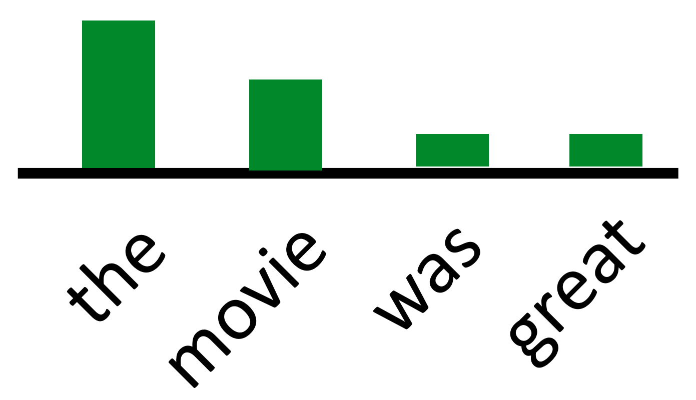
- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

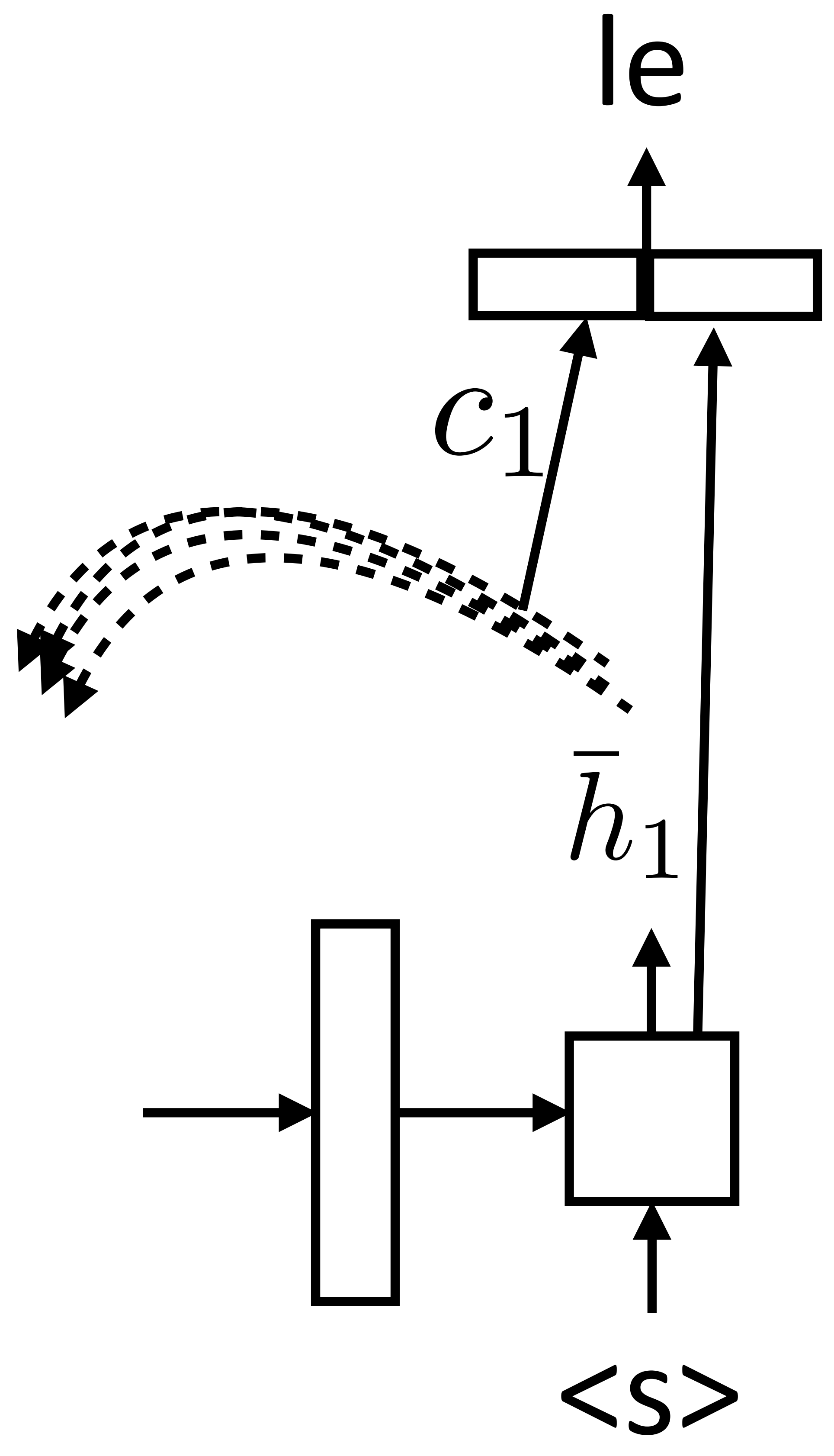
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



$$e_{ij} = f(\bar{h}_i, h_j)$$

- Some function f (next slide)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

- Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

- Luong+ (2015): dot product

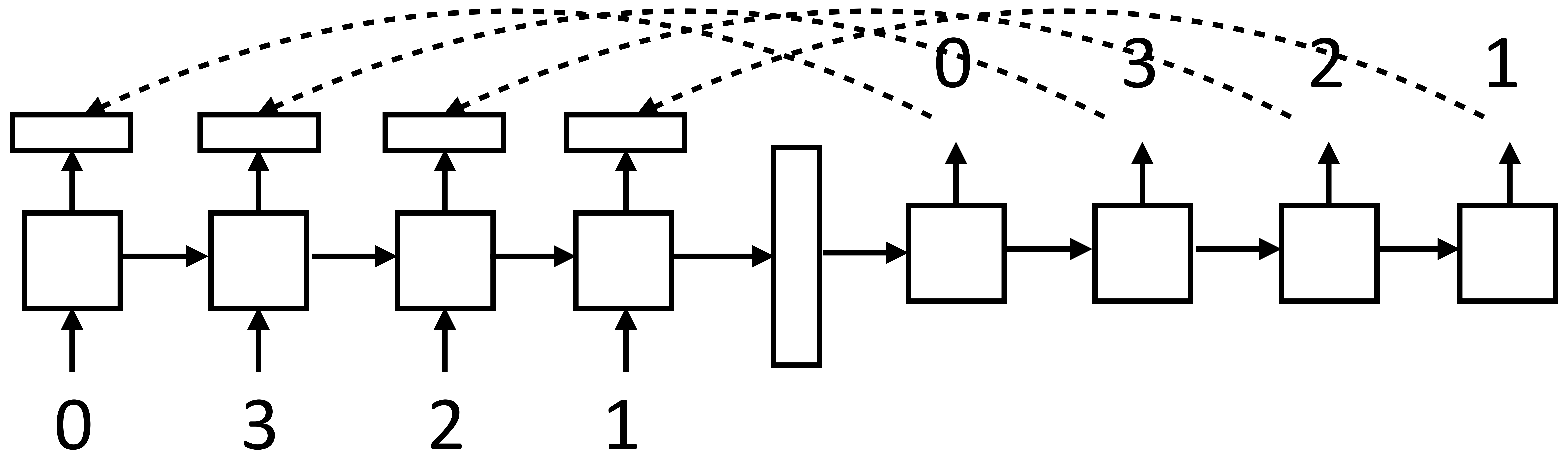
$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

- Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

What can attention do?

- Learning to copy — how might this work?



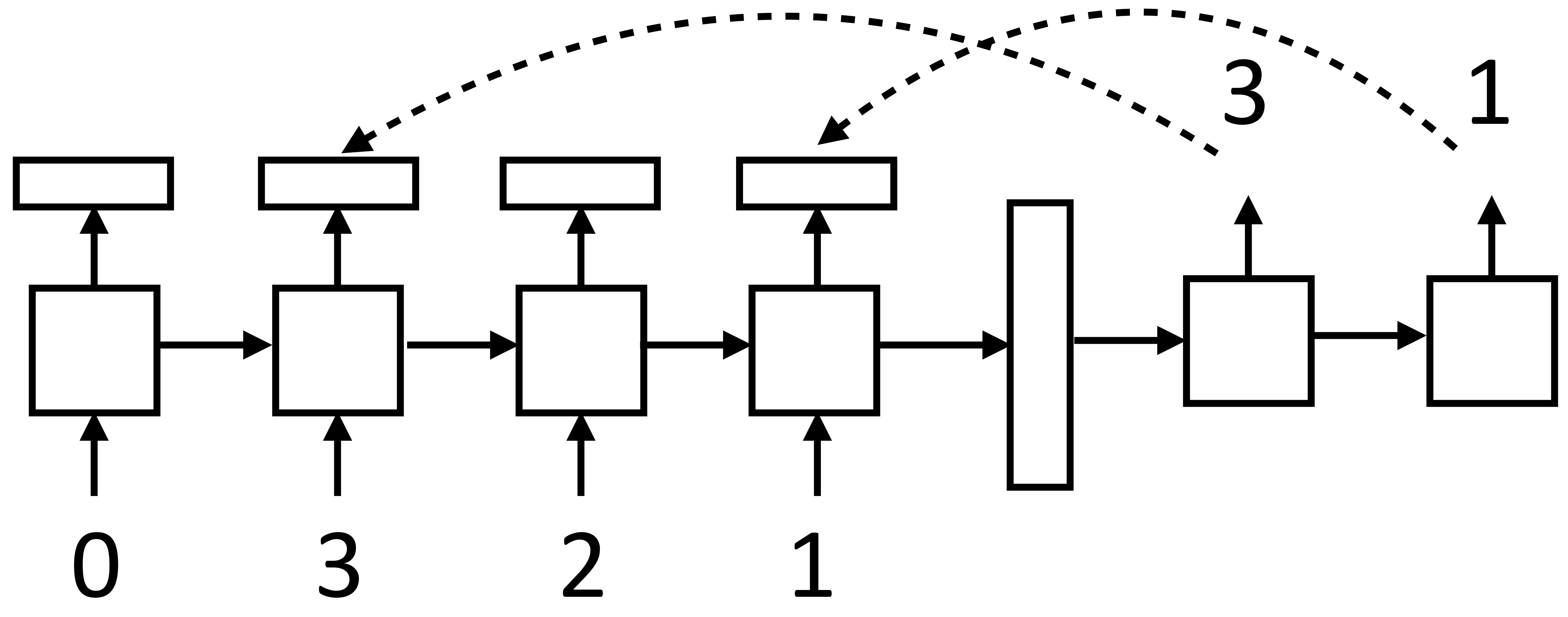
- LSTM can learn to count with the right weight matrix

- This is a kind of position-based addressing

Luong et al. (2015)

What can attention do?

- Learning to subsample tokens



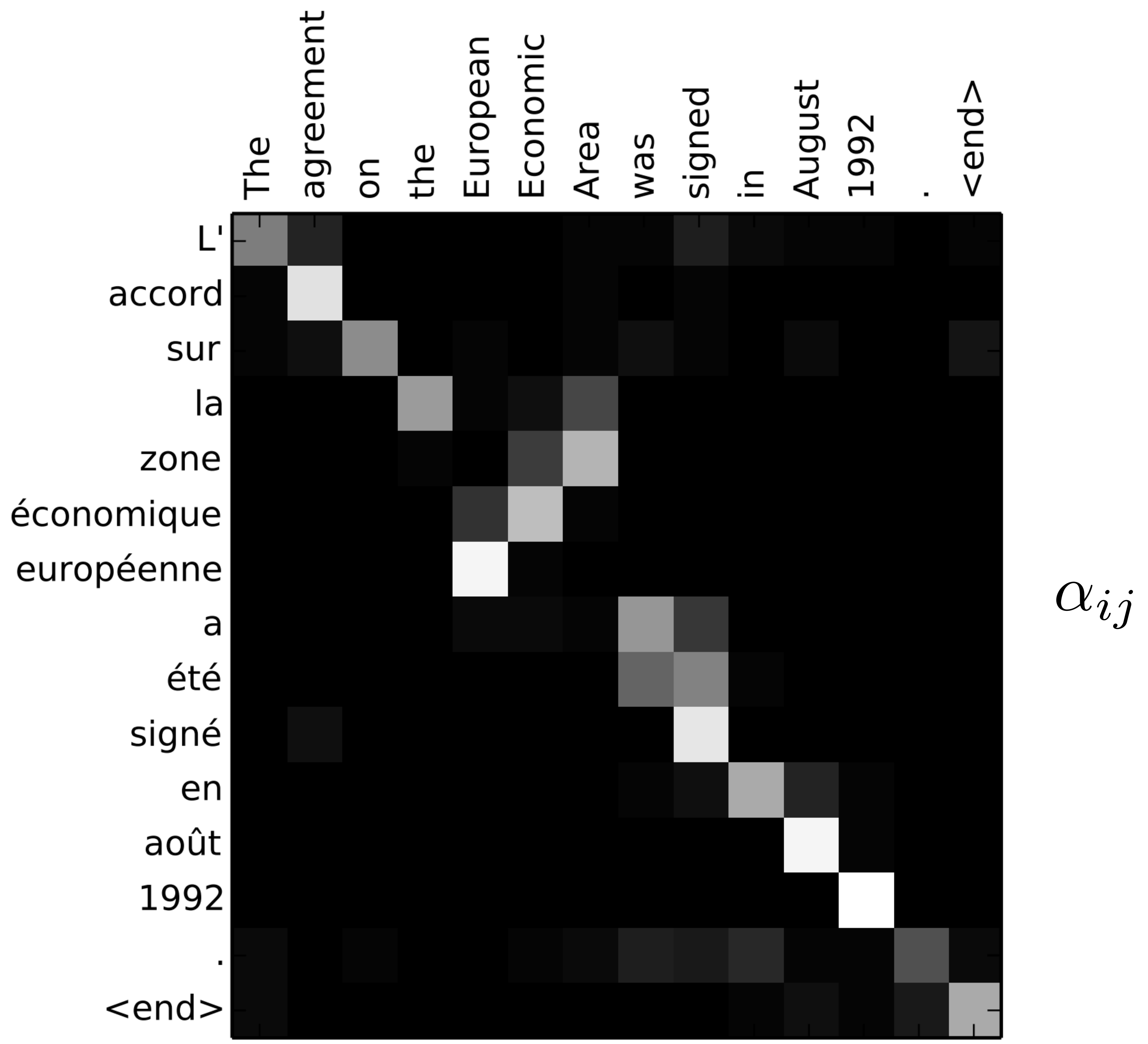
- Need to count (for ordering) and also determine which tokens are in/out

- Content-based addressing

Luong et al. (2015)

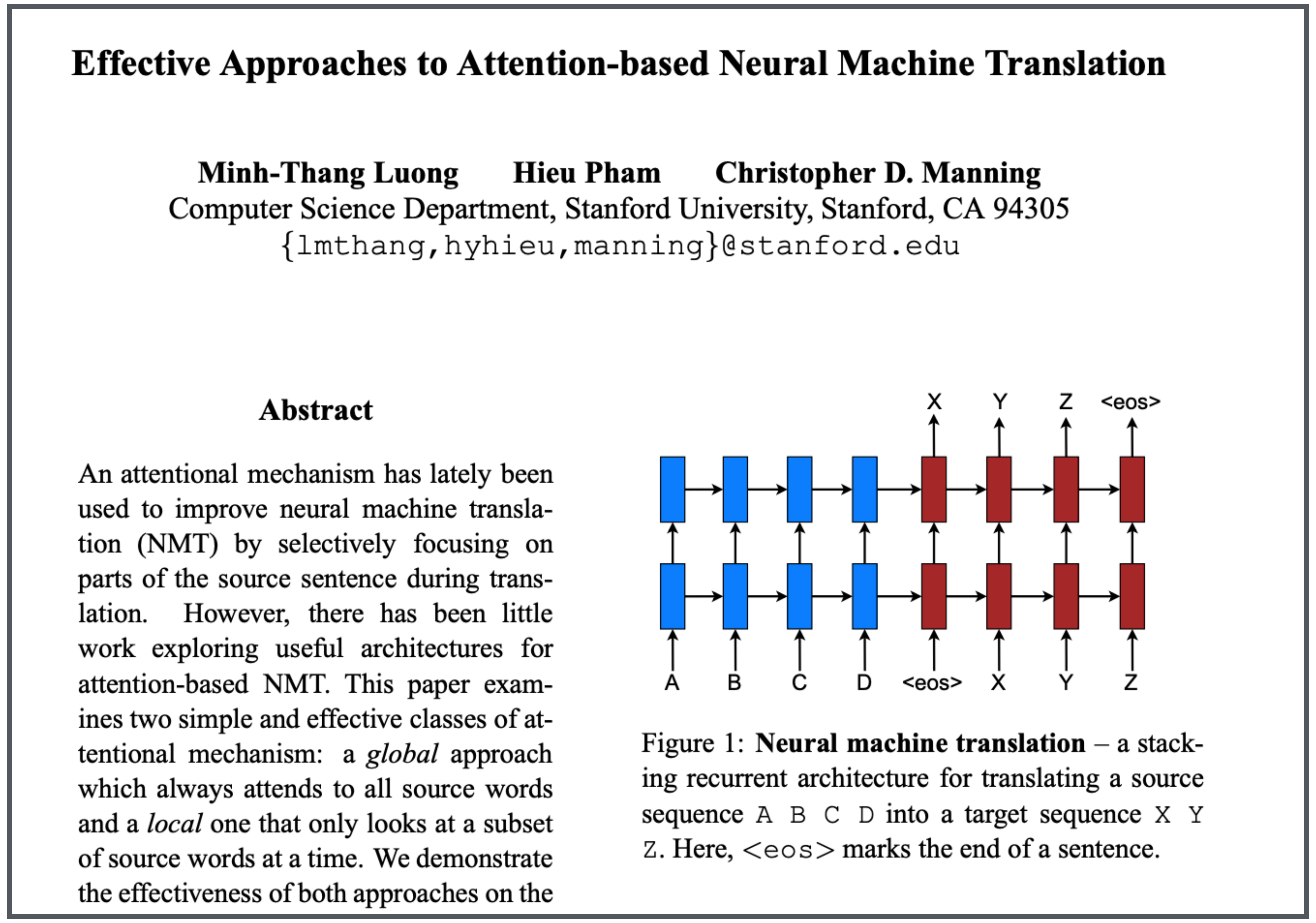
Attention

- Encoder hidden states capture contextual source word identity ("soft" word alignment)
- Decoder hidden states are now mostly responsible for selecting what to attend to
- Doesn't take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations



Luong et al. (2015)

"Early" Neural MT



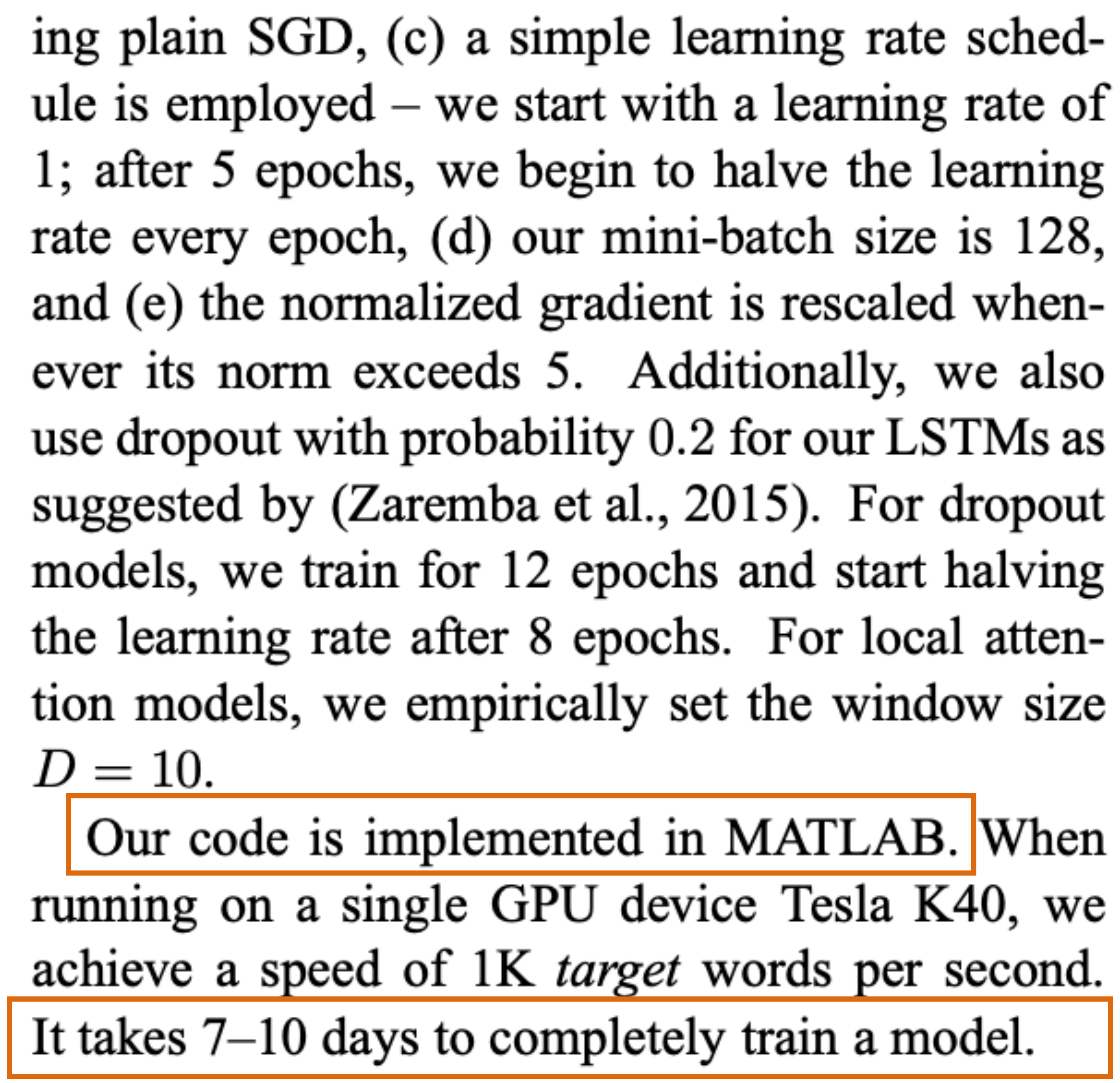
Effective Approaches to Attention-based Neural Machine Translation

Minh-Thang Luong **Hieu Pham** **Christopher D. Manning**
Computer Science Department, Stanford University, Stanford, CA 94305
{lmthang,hyhieu,manning}@stanford.edu

Abstract

An attentional mechanism has lately been used to improve neural machine translation (NMT) by selectively focusing on parts of the source sentence during translation. However, there has been little work exploring useful architectures for attention-based NMT. This paper examines two simple and effective classes of attentional mechanism: a *global* approach which always attends to all source words and a *local* one that only looks at a subset of source words at a time. We demonstrate the effectiveness of both approaches on the

Figure 1: **Neural machine translation** – a stacking recurrent architecture for translating a source sequence `A B C D` into a target sequence `X Y Z`. Here, `<eos>` marks the end of a sentence.



ing plain SGD, (c) a simple learning rate schedule is employed – we start with a learning rate of 1; after 5 epochs, we begin to halve the learning rate every epoch, (d) our mini-batch size is 128, and (e) the normalized gradient is rescaled whenever its norm exceeds 5. Additionally, we also use dropout with probability 0.2 for our LSTMs as suggested by (Zaremba et al., 2015). For dropout models, we train for 12 epochs and start halving the learning rate after 8 epochs. For local attention models, we empirically set the window size $D = 10$.

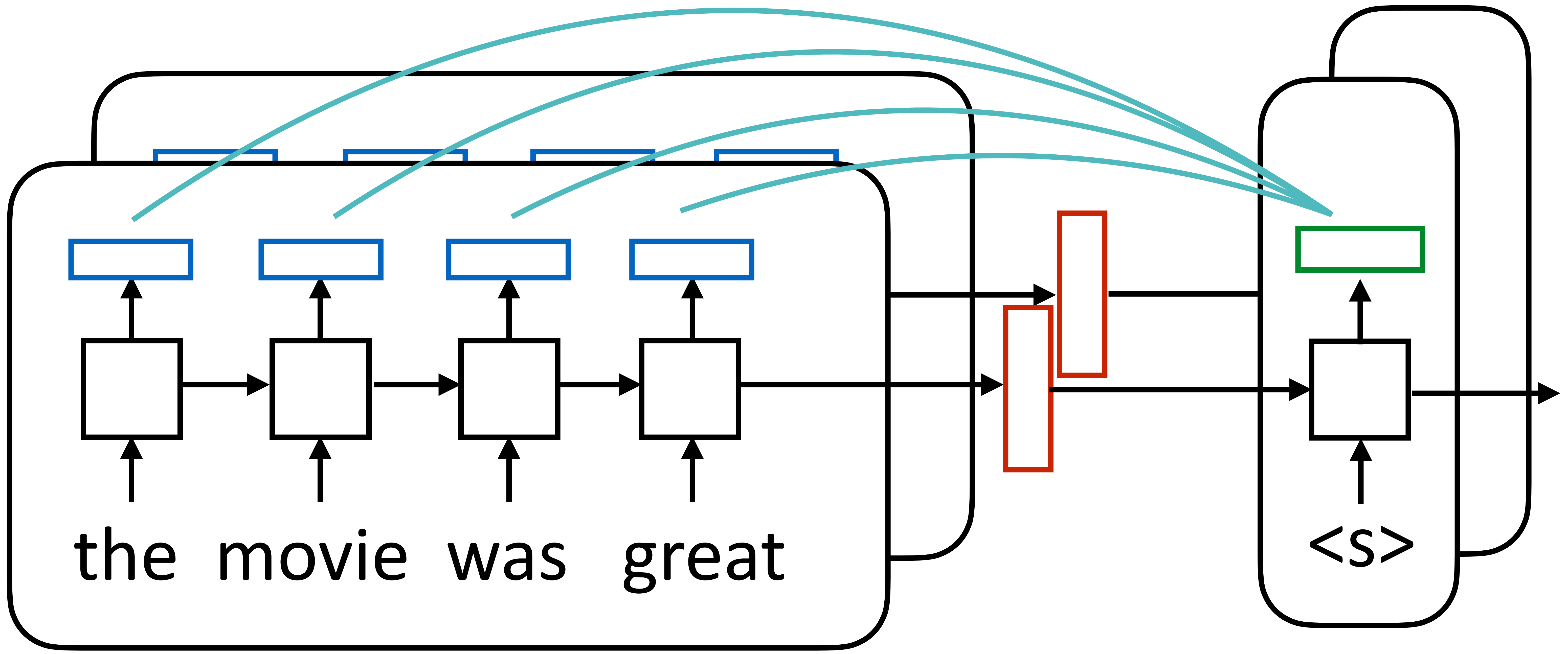
Our code is implemented in MATLAB. When running on a single GPU device Tesla K40, we achieve a speed of 1K *target* words per second. It takes 7–10 days to completely train a model.

- TensorFlow first released in Nov 2015.
- PyTorch first released in 2016.

Luong et al. (2015)

Batching Attention

token outputs: batch size x sentence length x dimension



hidden state: batch size x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

sentence outputs: batch size x hidden size

attention scores = batch size x sentence length

c = batch size x hidden size

$$c_i = \sum_j \alpha_{ij} h_j$$

- Make sure tensors are the right size!

Luong et al. (2015)

Results

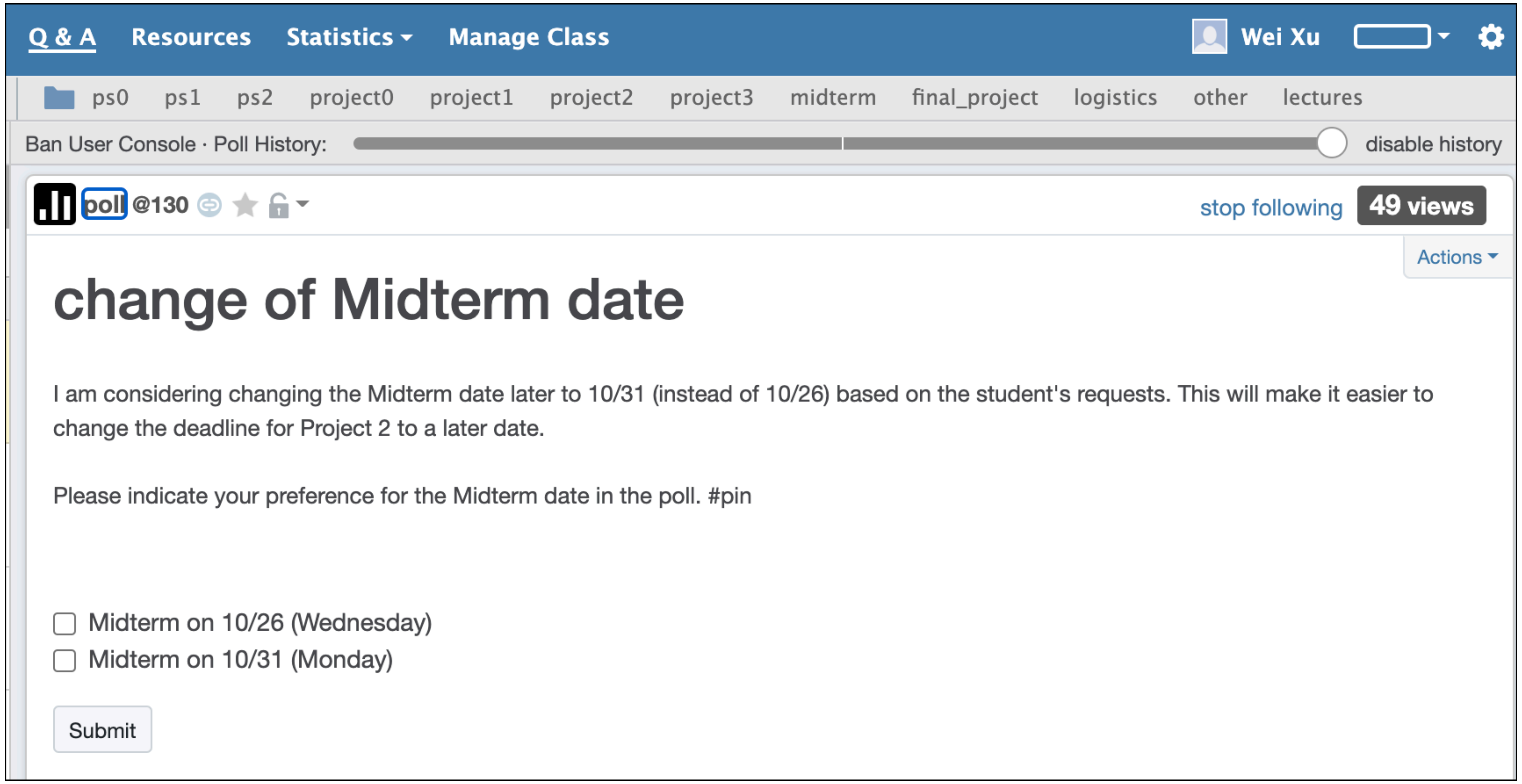
- Machine translation: BLEU score of 14.0 on English-German -> 16.8 with attention, 19.0 with smarter attention (constrained to a small windows)
- Summarization/headline generation: bigram recall from 11% -> 15%
- Semantic parsing: ~30-50% accuracy -> 70+% accuracy on Geoquery

Luong et al. (2015)
Chopra et al. (2016)
Jia and Liang (2016)

Midterm

Midterm

- Poll on Midterm date (10/26 or 10/31?). If you haven't voted, please do.



change of Midterm date

I am considering changing the Midterm date later to 10/31 (instead of 10/26) based on the student's requests. This will make it easier to change the deadline for Project 2 to a later date.

Please indicate your preference for the Midterm date in the poll. #pin

- ☐ Midterm on 10/26 (Wednesday)
- ☐ Midterm on 10/31 (Monday)

Submit

Midterm

- The midterm will be closed notes, books, laptops, smartphones, and people.
- 75 minutes in class.
- **Preparation:**
  - Lecture slides + readings
  - Written homework (PS1 and PS2)
  - Programming Project 1 and 2

Midterm

- Broad types of questions:
  - Long answers (similar to PS1/2), e.g. simple toy example for HMM/Viterbi
  - Short answers
  - True/False
  - Multiple choice

- Make sure you **understand** the fundamentals in addition to being able to procedurally execute a few key algorithms (e.g., Viterbi, Beam search).
- But, in general, we will have a greater emphasis on open-ended and conceptual questions in midterm (e.g., what are the main drawbacks/advantages of XXX model? Or, we will describe a new technique and ask you to reason about it).

Topics

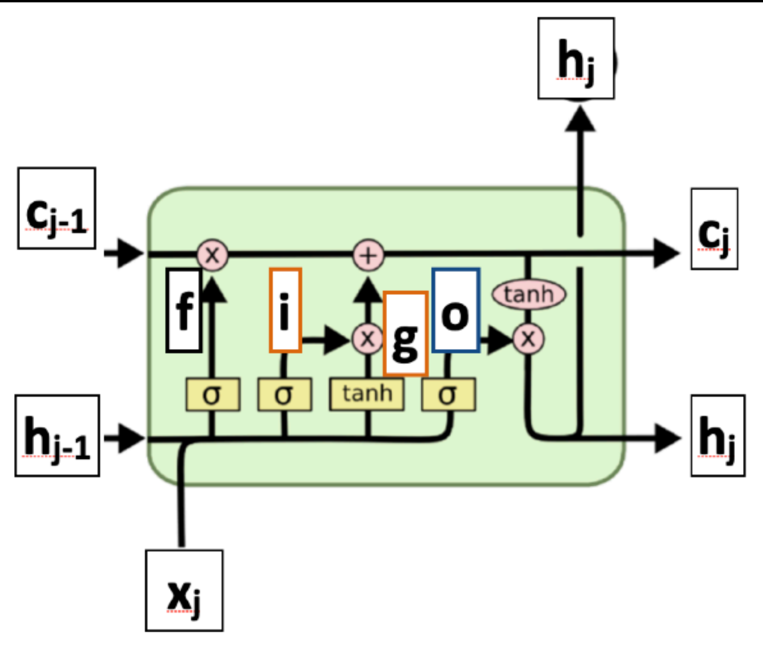
- **Important topics that we may test on (not limited to):**
  - Perceptron
  - Logistic regression: model, training objective, gradient update, etc.
  - Optimization: stochastic gradient descent, learning rate, initialization, etc.
  - Training neural networks
  - Feedforward, CNN, RNN, LSTM: model architecture, dimensionality, pros/cons, etc.
  - Word2vec/skip-gram
  - Sequence-to-sequence model, attention mechanism

Topics

- **Important topics that we may test on (not limited to):**
  - Sequential task: NER BIO tagging scheme
  - Evaluation: Precision/Recall/F1, BLEU score
  - HMM: definition, parameter estimation, Viterbi Algorithm
  - CRF: advantages, forward-backward algorithm
  - MT: Beam search, language models, word alignment
  - Runtime complexity of different algorithms

Topics

- **Topics that will not be tested:**
  - Naive Bayes
  - SVM
- **Topics that will not have very involved questions, but possibly conceptual or short-answer questions:**
  - No execution of forward-backward algorithm on toy example
  - No very involved question on word alignment
  - LSTM/GRU, if tested, we will provide corresponding equations
  - No need to memorize BLEU score's equation
  - Back-propagation



$$
\begin{aligned}
\mathbf{c_j} &= \mathbf{c_{j-1}} \odot \mathbf{f} + \mathbf{g} \odot \mathbf{i} \\
\mathbf{f} &= \sigma(\mathbf{x_j W^{xf}} + \mathbf{h_{j-1} W^{hf}}) \\
\mathbf{g} &= \tanh(\mathbf{x_j W^{xg}} + \mathbf{h_{j-1} W^{hg}}) \\
\mathbf{i} &= \sigma(\mathbf{x_j W^{xi}} + \mathbf{h_{j-1} W^{hi}}) \\
\mathbf{h_j} &= \tanh(\mathbf{c_j}) \odot \mathbf{o} \\
\mathbf{o} &= \sigma(\mathbf{x_j W^{xo}} + \mathbf{h_{j-1} W^{ho}})
\end{aligned}
$$

KEEP
CALM
AND
HAVE A
GOOD FALL BREAK
KeepCalmAndPosters.com