Neural (Pre-Trained) Language Models

Mausam

(Based on slides of Yoav Goldberg, Graham Neubig, Jay Allamar and Keshav Kolluru)

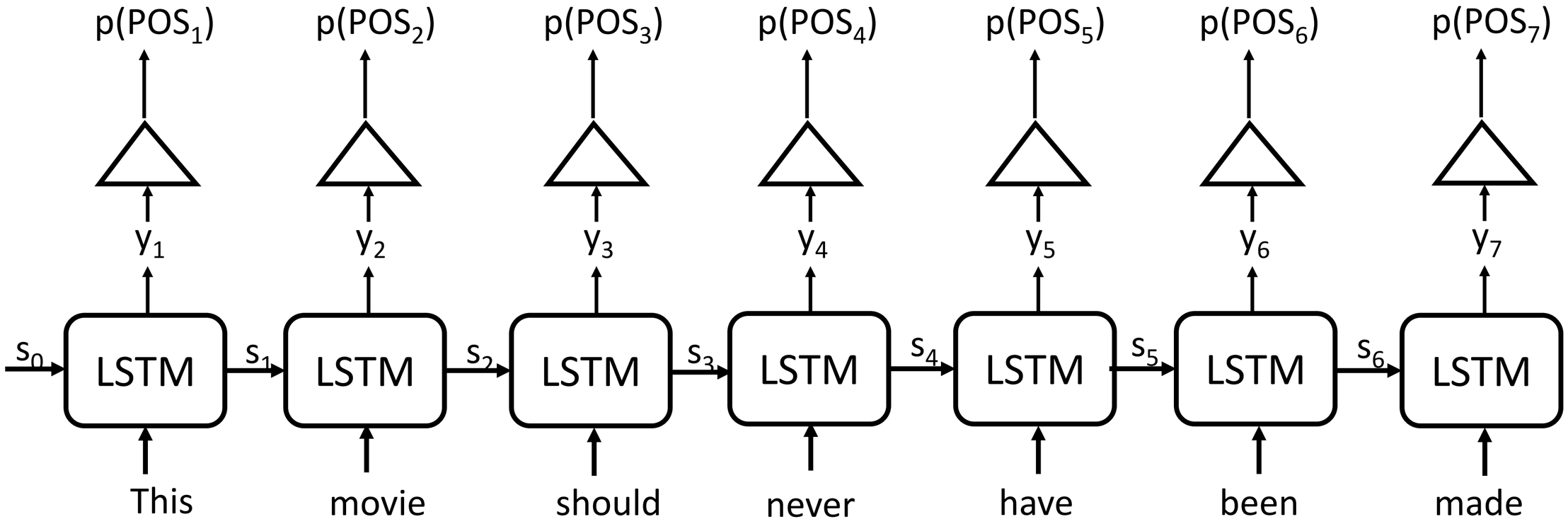
Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

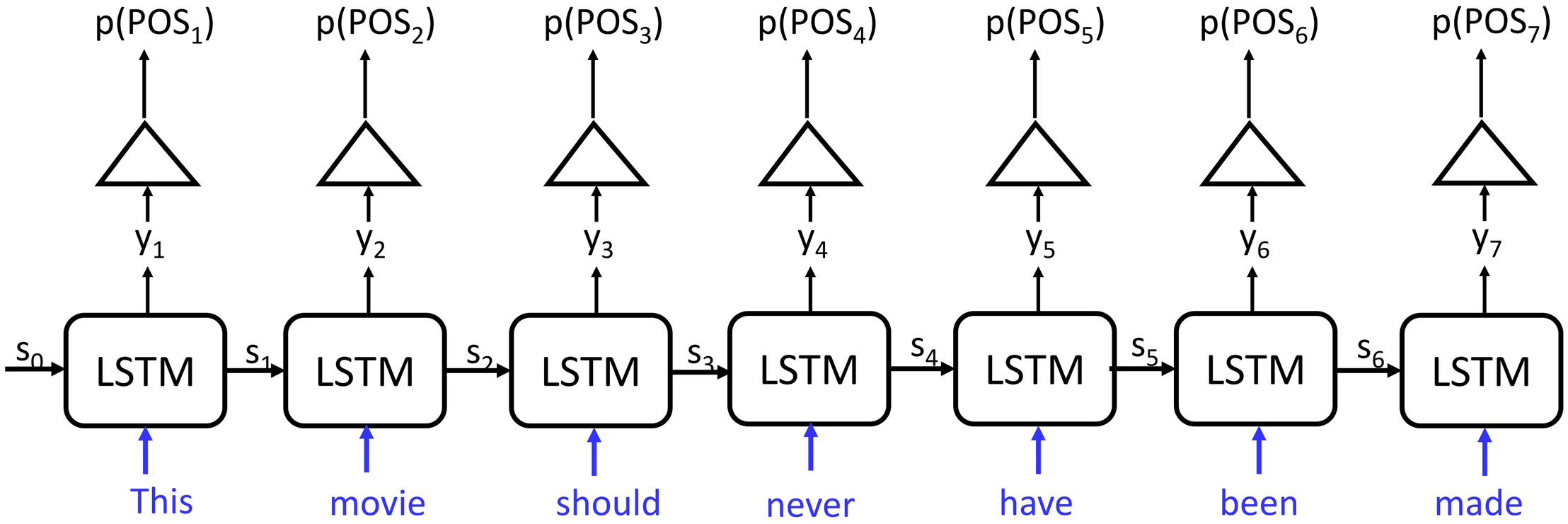
- Pre-trained Language Models: LSTMs (ELMo)
- Pre-trained Language Models: Transformers (GPT, BERT)

Sequence Decoder

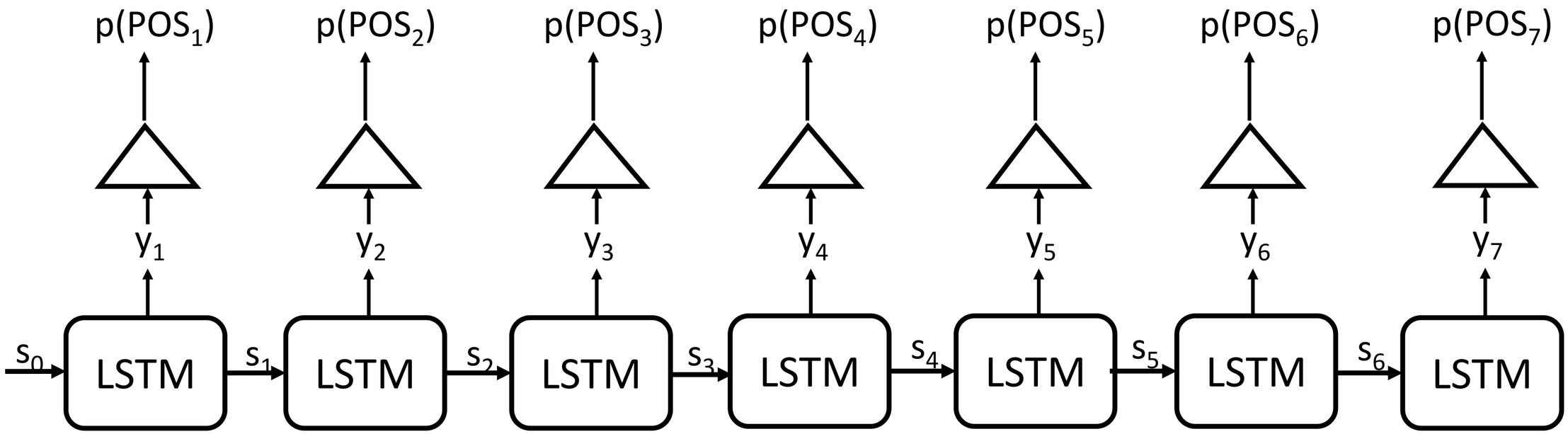
Neural Language Model



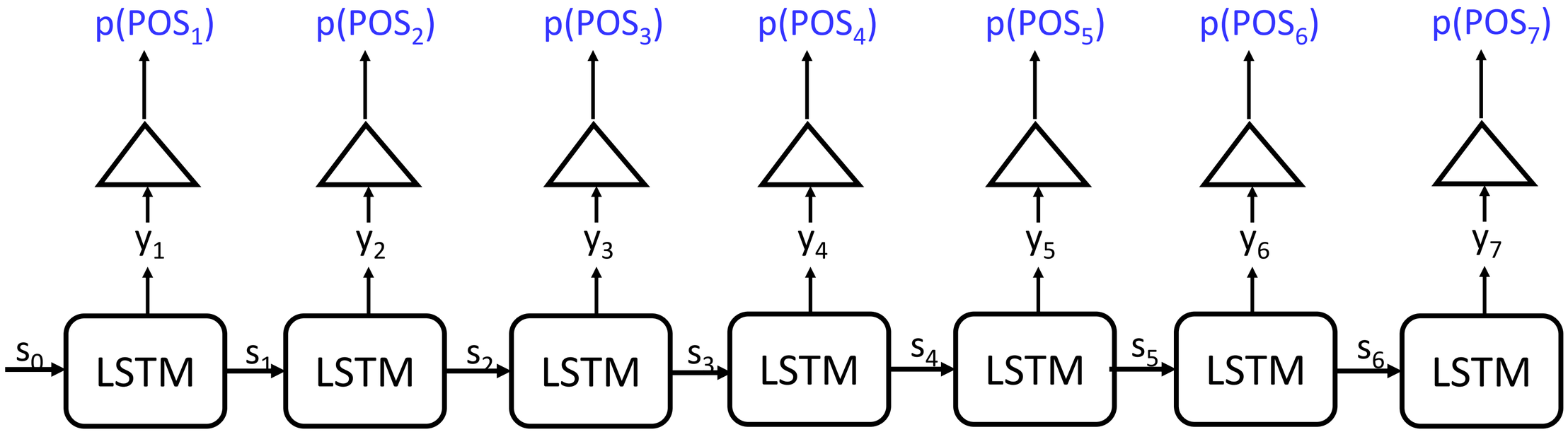
Neural Language Model



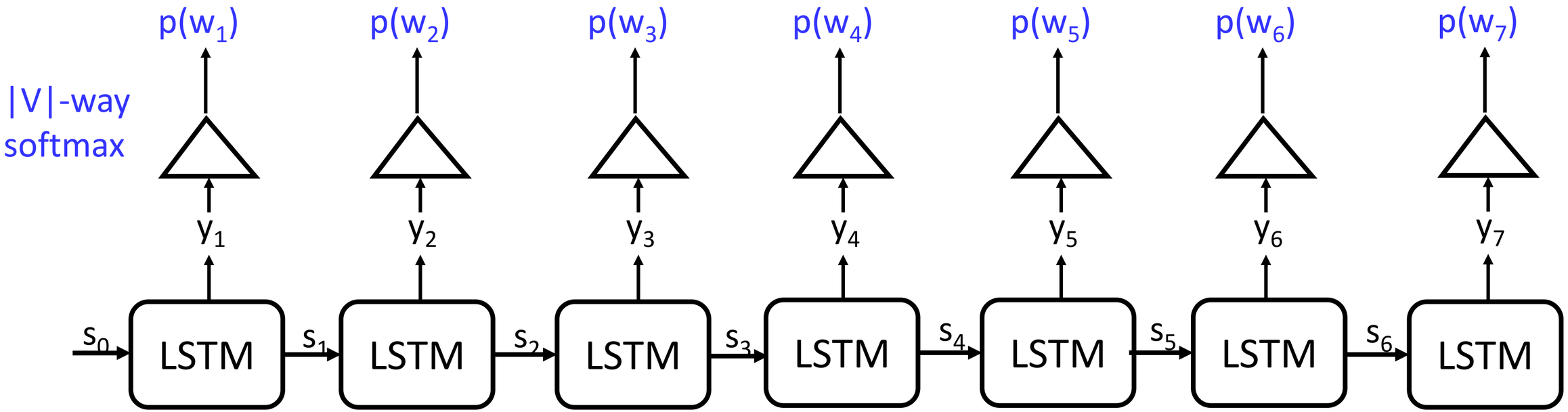
Neural Language Model



Neural Language Model

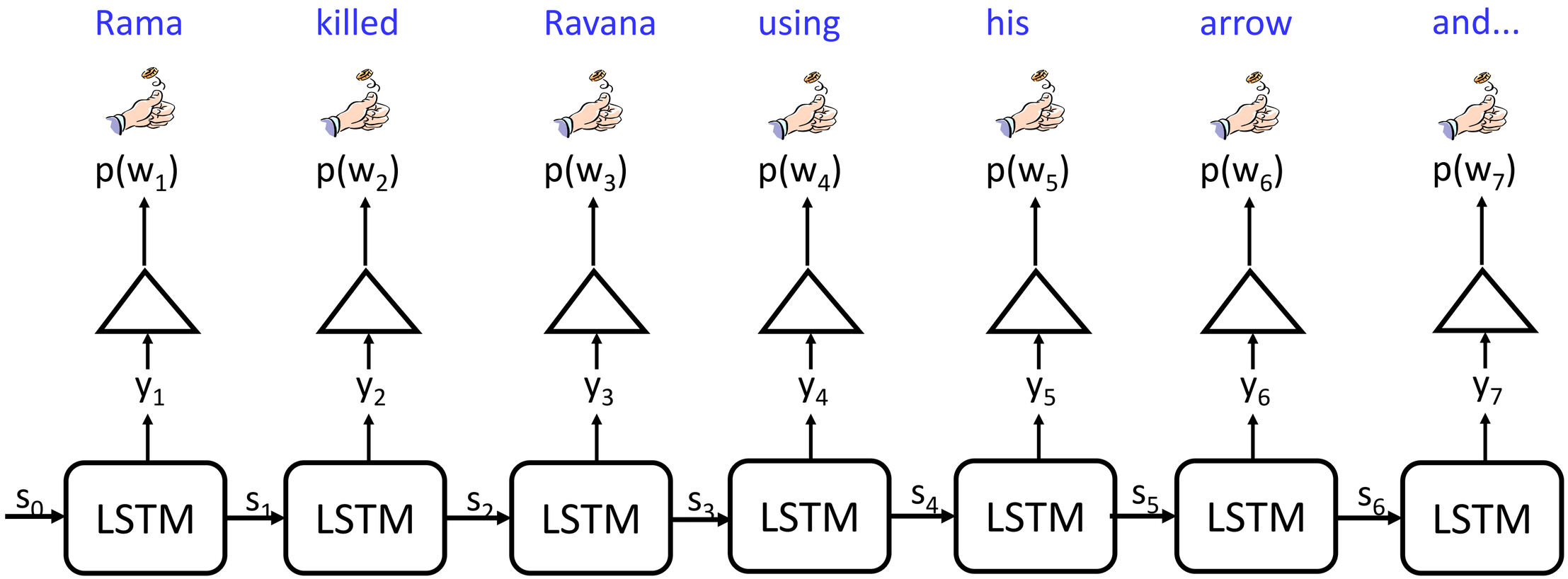


Neural Language Model



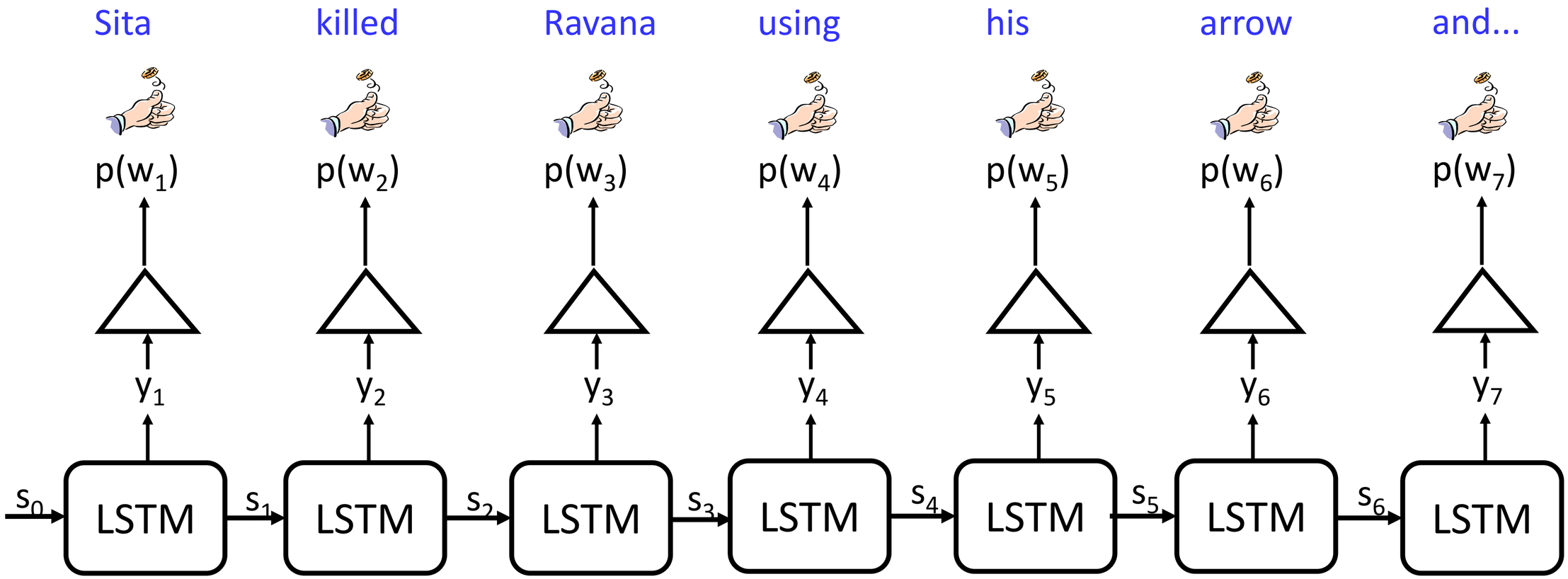
How do we get the actual sentence from a sequence of probability distributions?

Neural Language Model



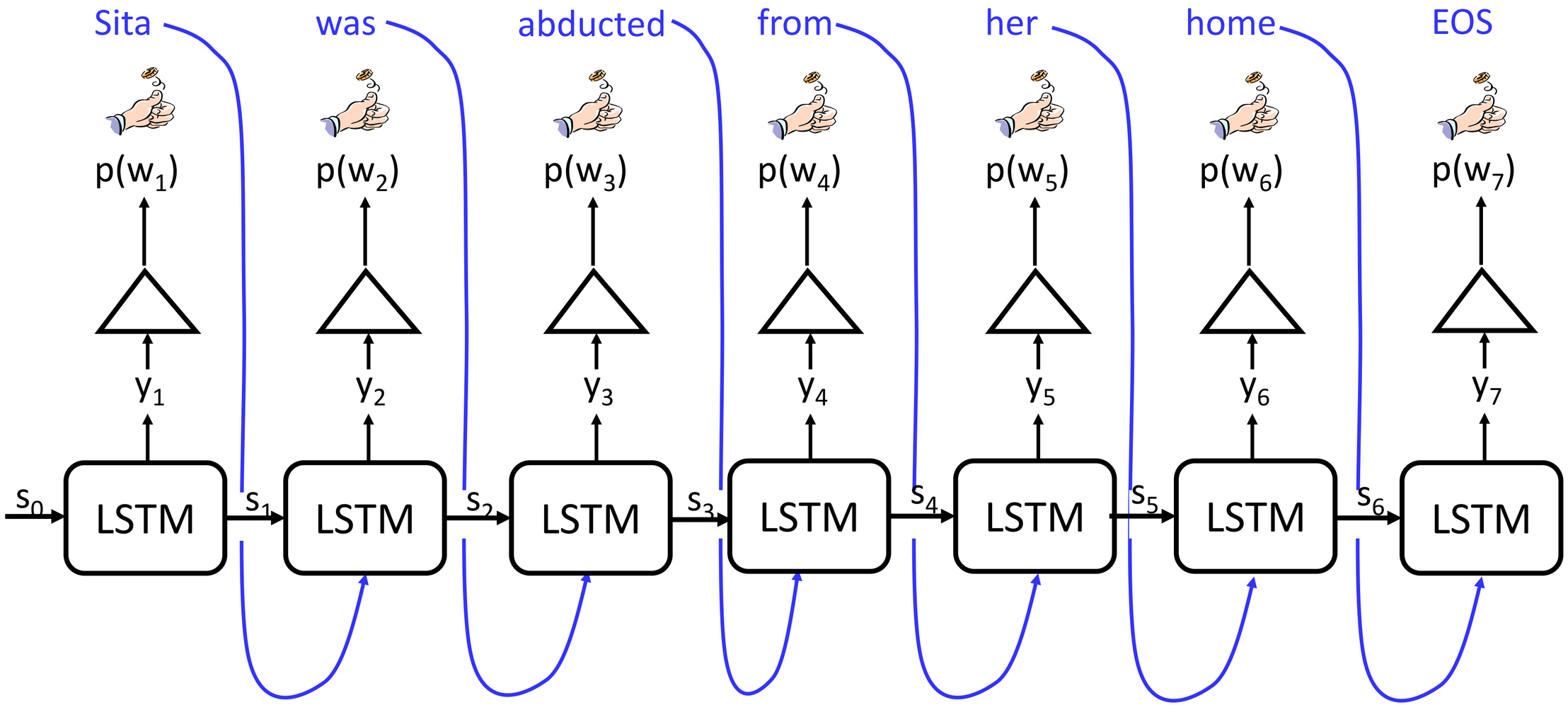
Can you think of a fundamental problem in this design?

Neural Language Model

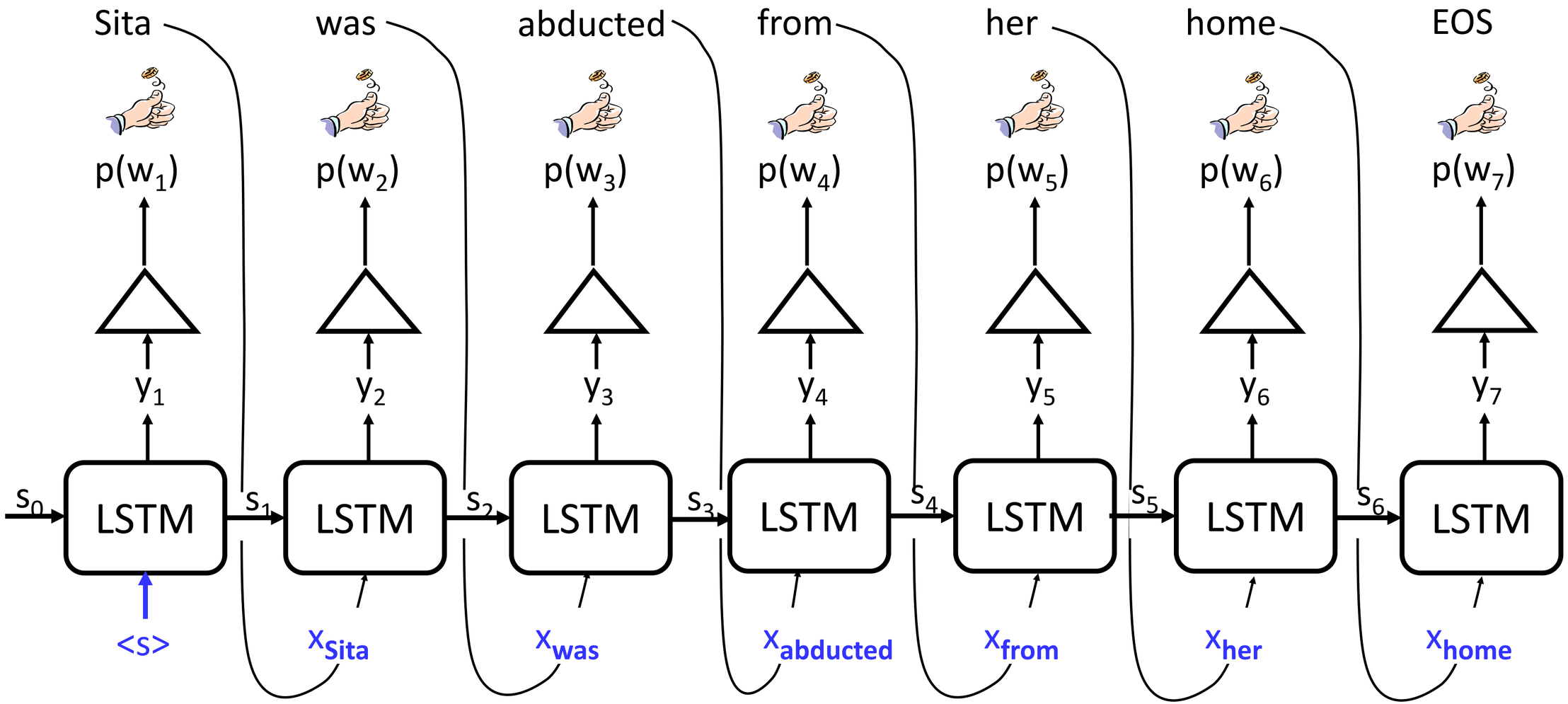


Can you think of a fundamental problem in this design?

Neural Language Model



Neural Language Model



Called “Auto-regressive models”

Neural Language Model

- Use LSTMs not BiLSTMs
  - Why?

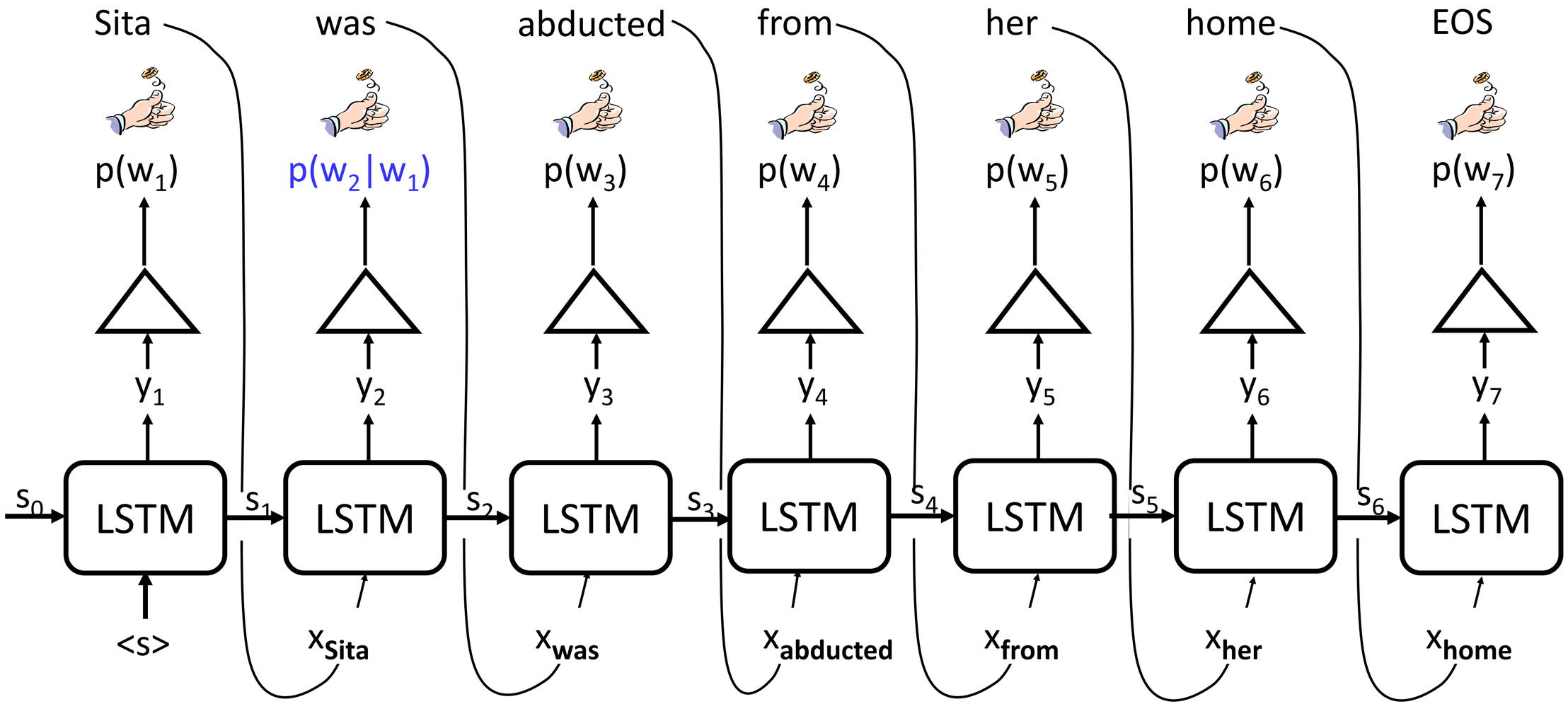
- When does it stop?

- Define the probability distribution over the next item in a sequence (and hence the probability of a sequence).

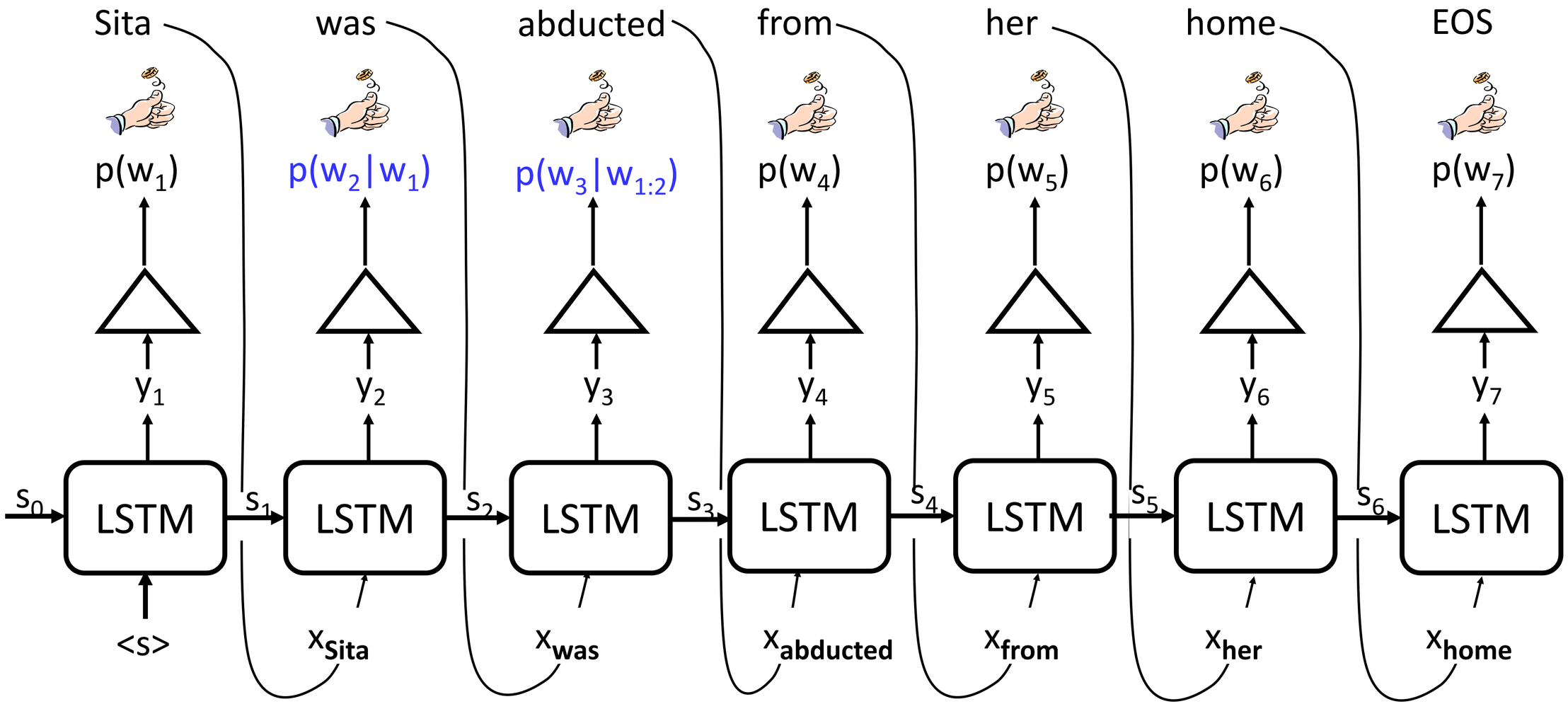
$$P(w_{1:n}) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_{1:2})P(w_4 \mid w_{1:3}) \ldots P(w_n \mid w_{1:n-1})$$

$$P(w_1, ..., w_n) = \prod_{i=1} P(t_i = w_i | w_1, ..., w_{i-1})$$

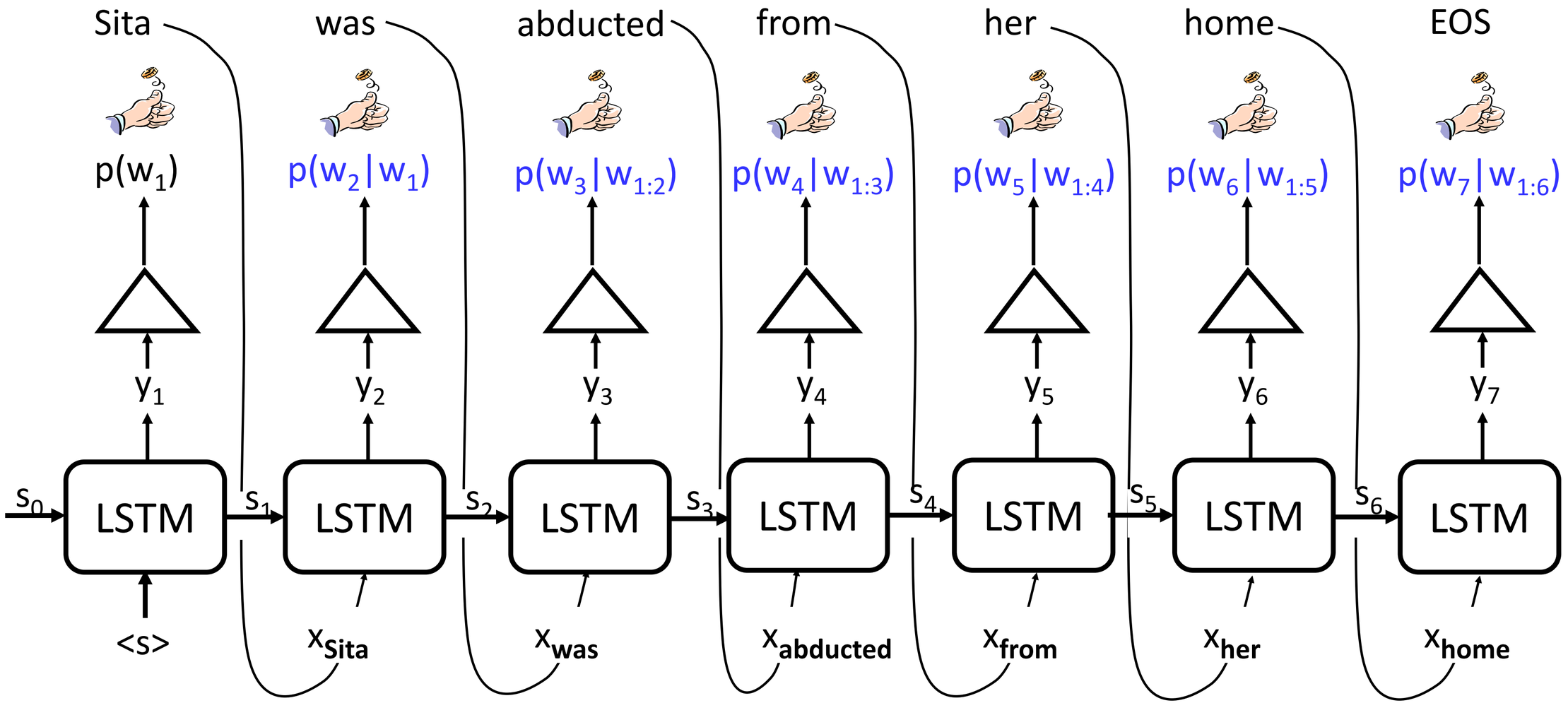
Neural Language Model



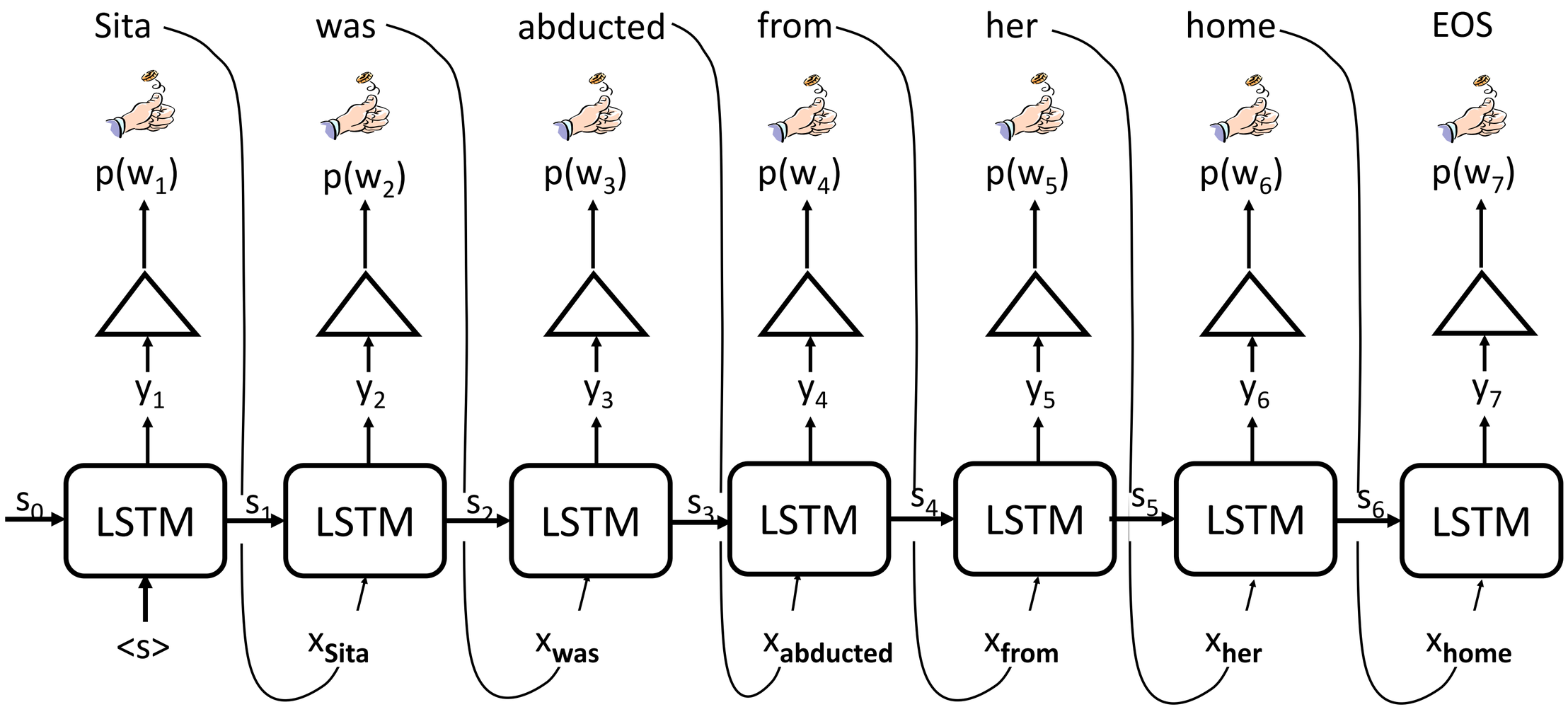
Neural Language Model



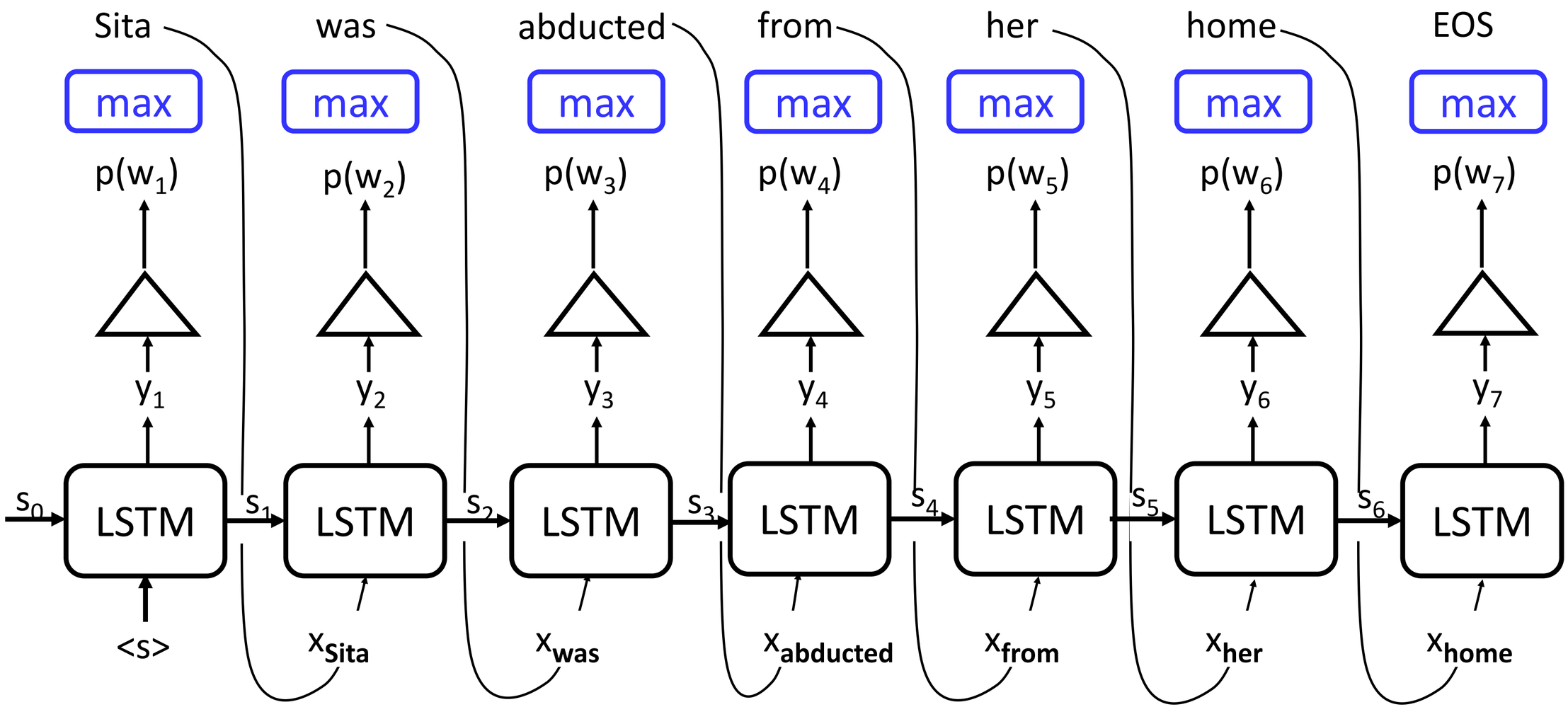
Neural Language Model



Neural Language Model: Inference Time



Neural Language Model: Inference Time

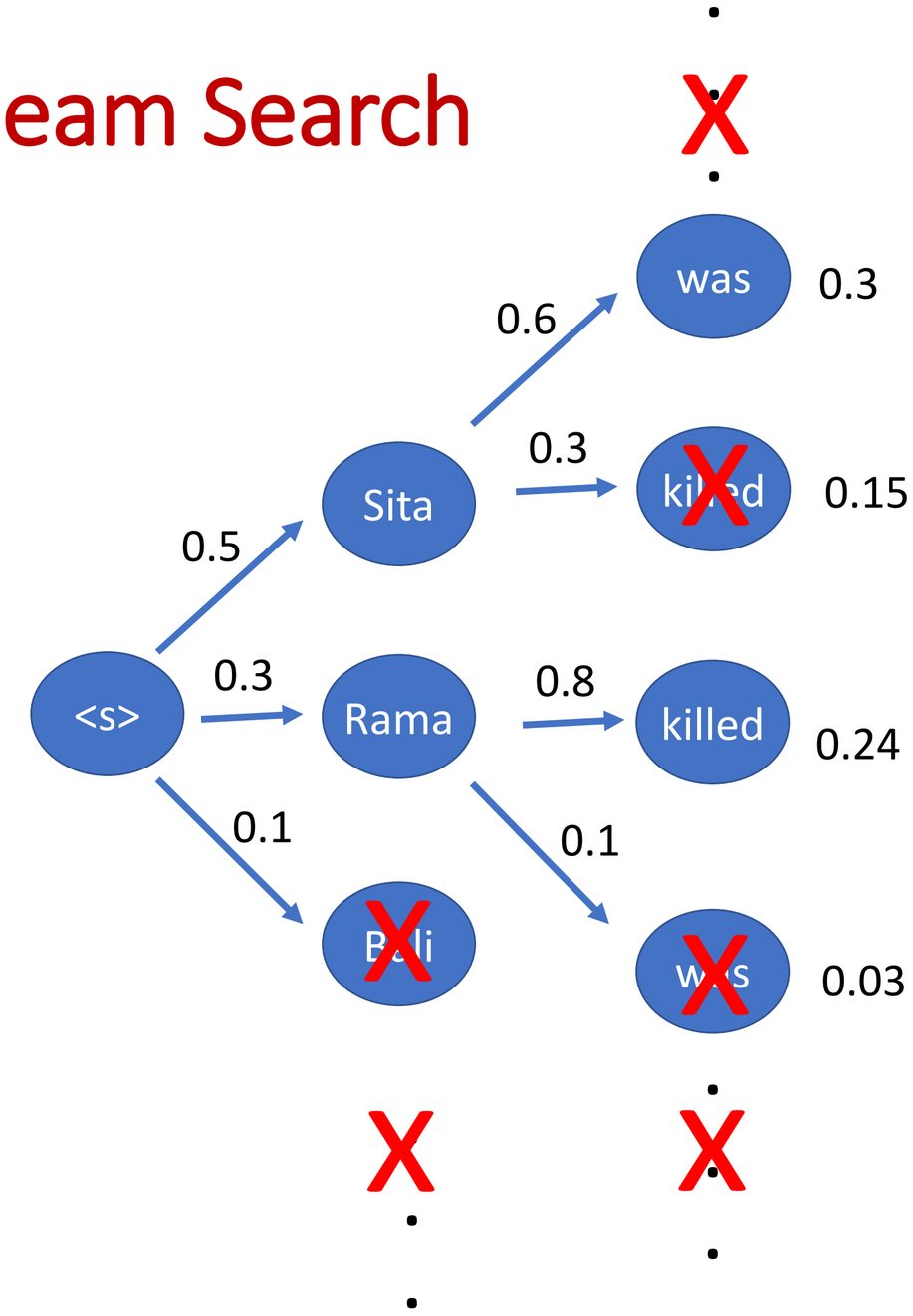


Neural Language Model: Greedy Decoding

- What is the function we actually wish to compute?
- $\underset{w_{1:n}}{\text{argmax}}\, P(w_{1:n})$

- Computing this expression is prohibitive.
- Greedy Approach: approximation can be bad because
  - model will never begin a sentence with a low probability word
  - model will prefer many common words to one rare word
- Solution: Beam Search

Beam Search

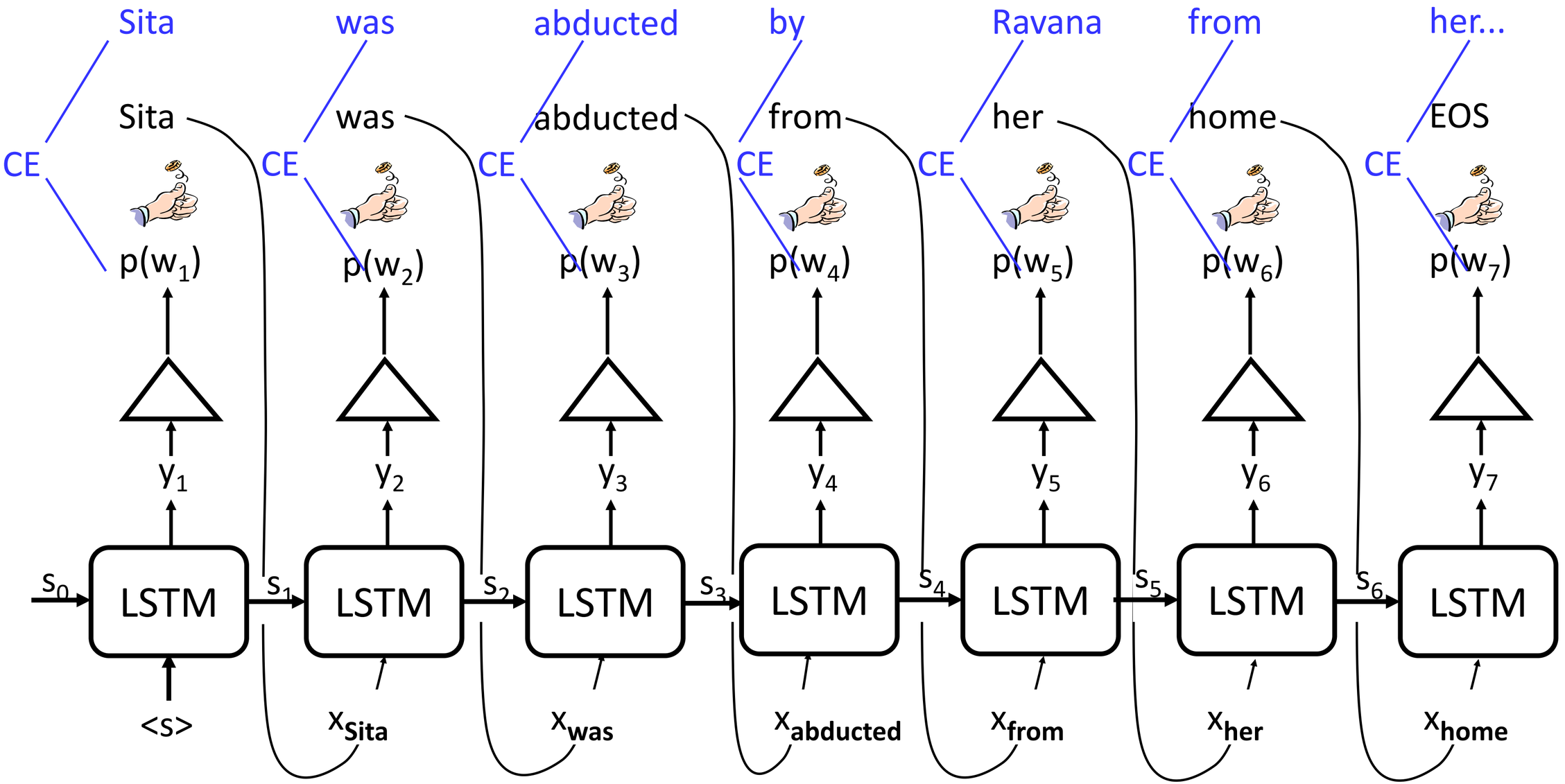


Instead of picking one greedy path, maintain multiple greedy paths

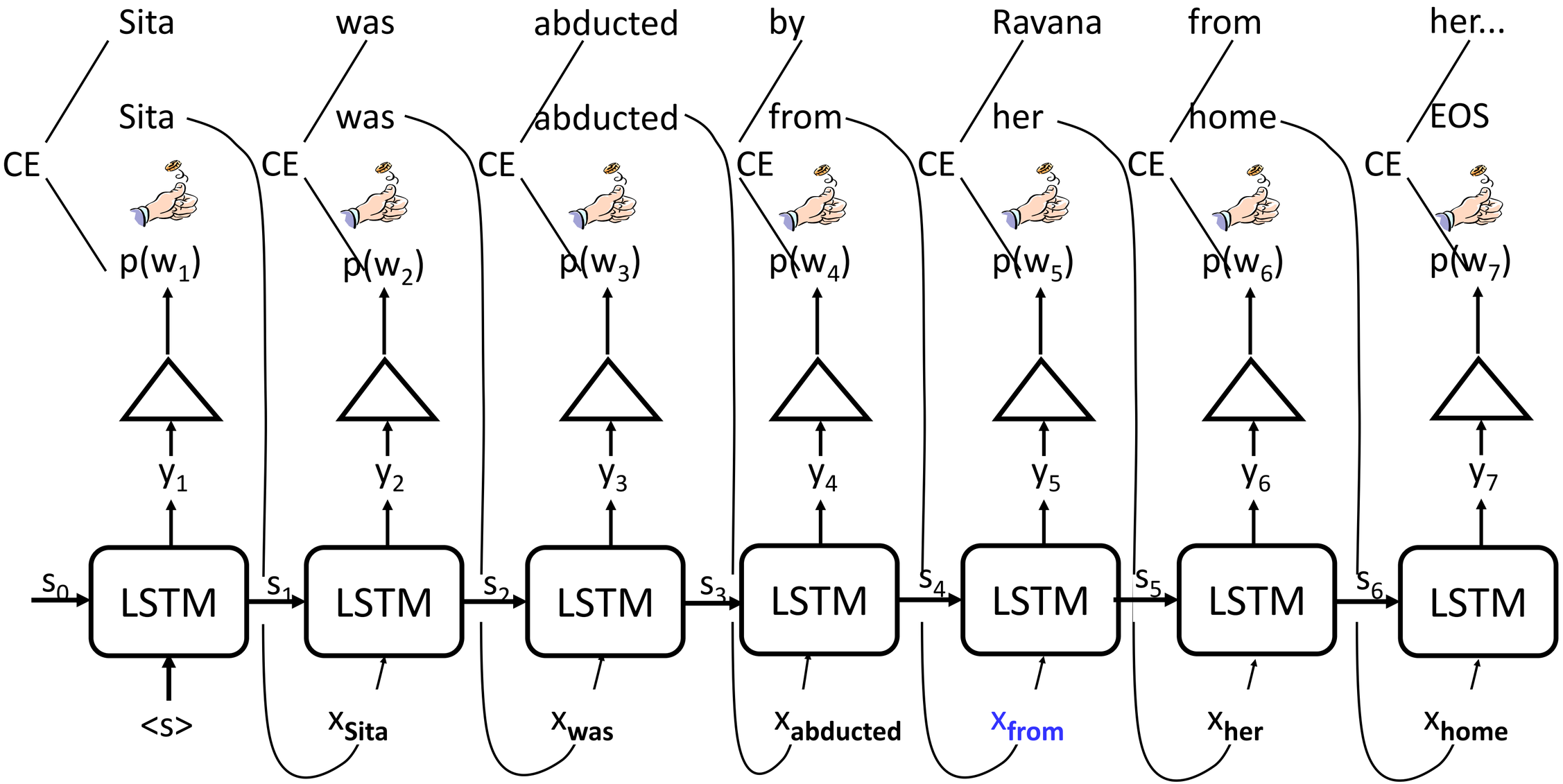
Upto a constant beam of b

Example for beam size =2

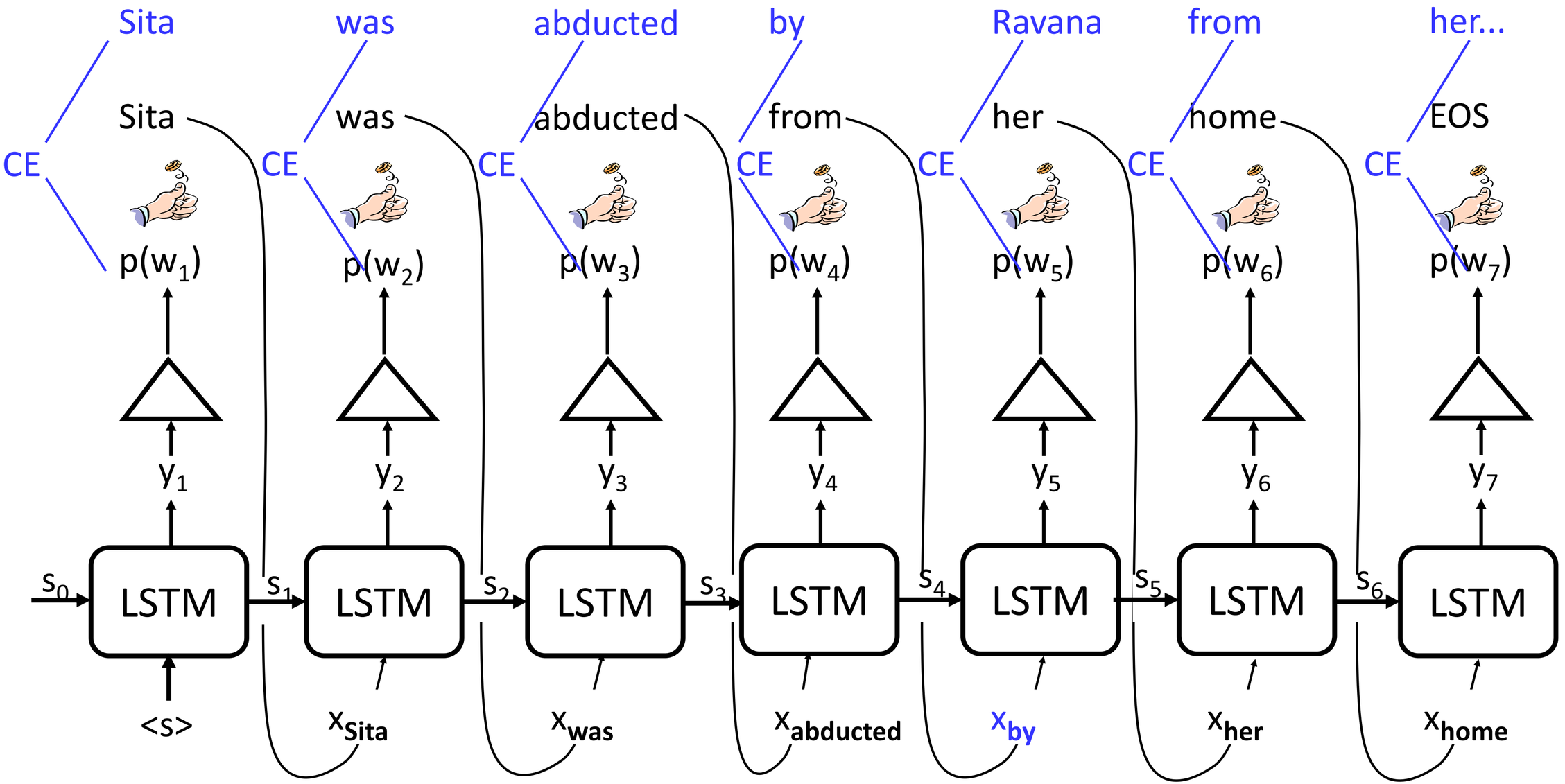
Neural Language Model: Training



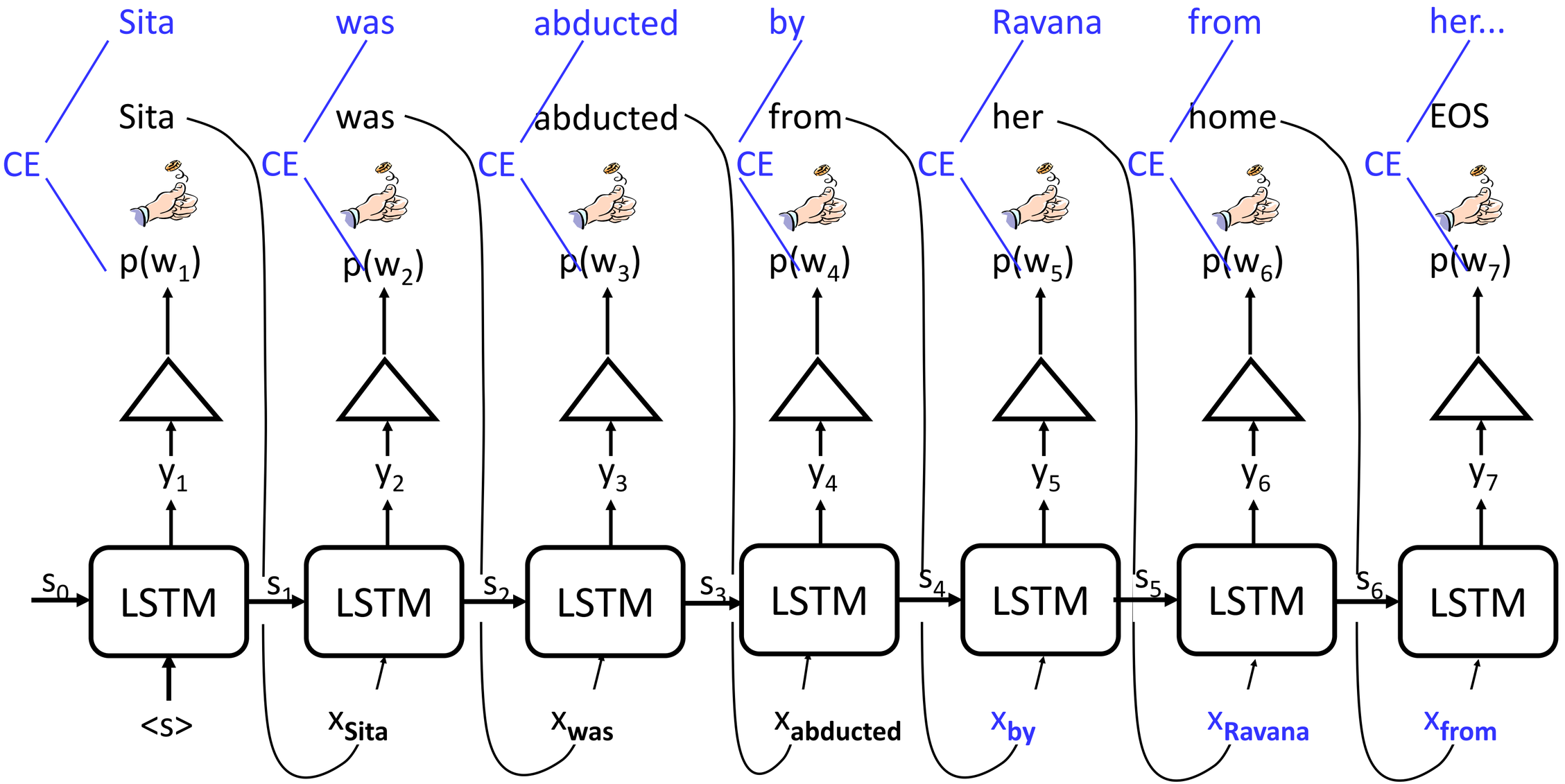
Neural Language Model: Training



Neural Language Model: Training (Teacher Forcing)



Neural Language Model: Training (Teacher Forcing)



How to Train this Model?

- Loss function: sum(cross entropy at each prediction)
- Issues with vanilla training
  - Slow convergence. Model instability. Poor skill.
- Simple idea: **Teacher Forcing**
  - Just feed in the *correct* previous tag during training
- Drawback: **Exposure bias**
  - Not exposed to mistakes during training

A Solution to Exposure Bias

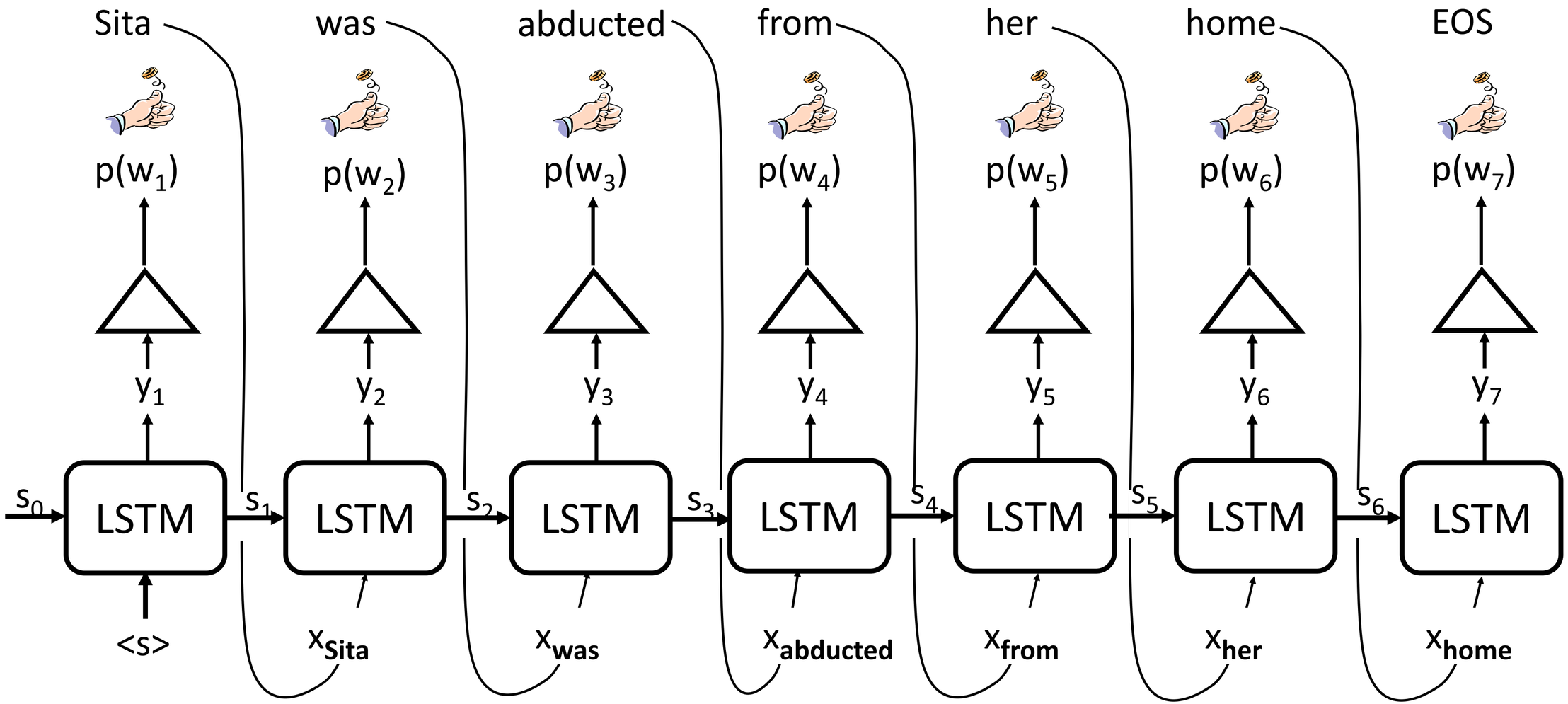
- DAgger (Ross et al. 2010) ~ “scheduled sampling”
- Start with no mistakes, and then
  - gradually introduce them using annealing

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo)
- Pre-trained Language Models: Transformers (GPT, BERT)

Neural Language Model



Goal

- Generate text based on (varied) inputs

- Examples
  - Machine Translation: Language → Language
  - Summarization: Language → Language
  - Dialogue Systems: Language → Language
  - Speech Recognition: Speech → Language
  - Image Captioning: Image → Language
  - Video Captioning: Video → Language
  - Speech Recognition in Videos: Video+Speech → Language

Goal

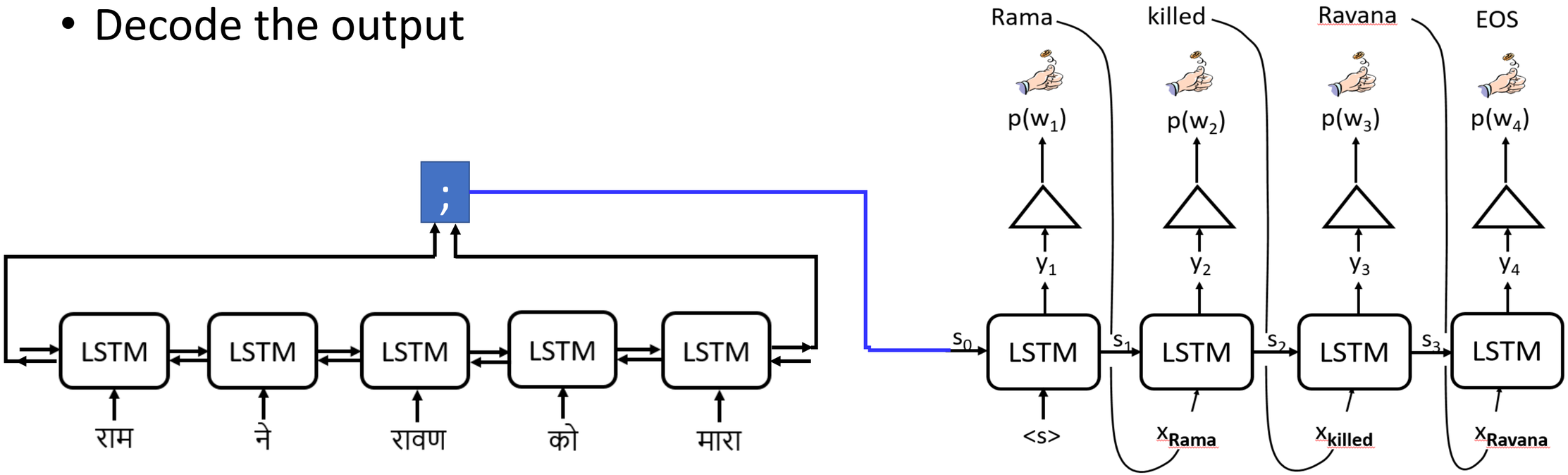
- Generate text based on (varied) inputs

- Examples
  - Machine Translation: Language → Language
  - Summarization: Language → Language
  - Dialogue Systems: Language → Language
  - Speech Recognition: Speech → Language
  - Image Captioning: Image → Language
  - Video Captioning: Video → Language
  - Speech Recognition in Videos: Video+Speech → Language

Seq2Seq

Idea 1: Encoder-Decoder

- Encode the input
- Pass the representation as starting state (s_0) to neural language model
- Decode the output

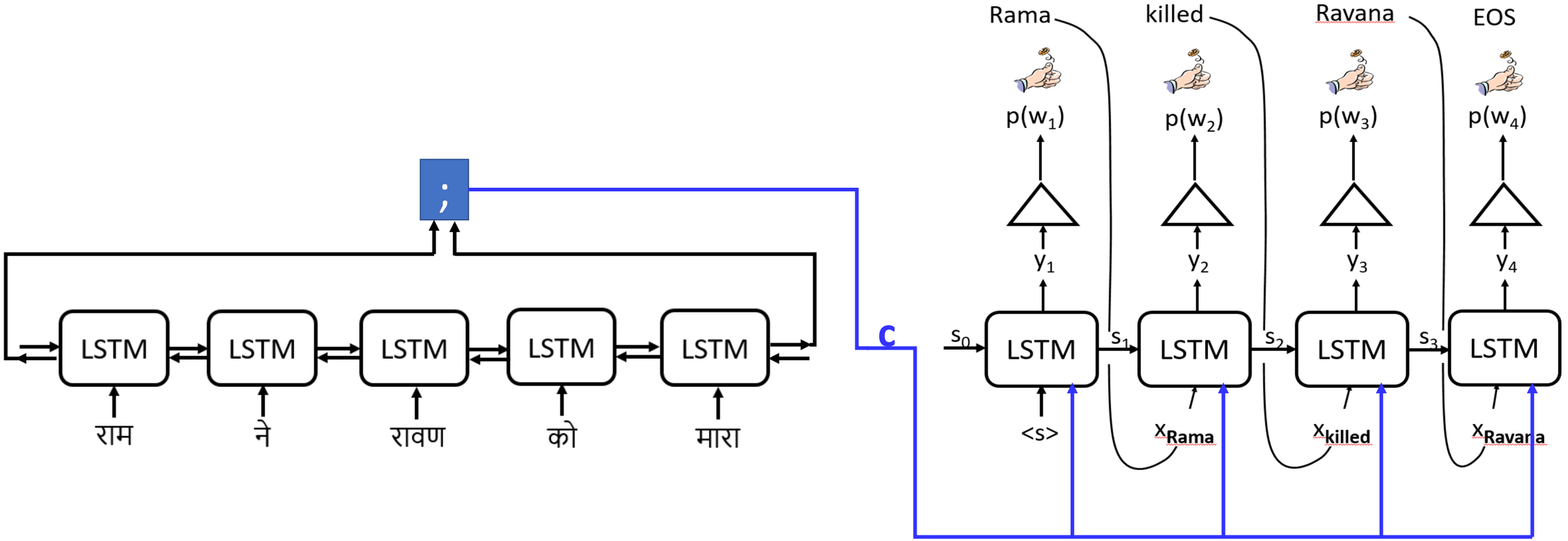


Encoder
(Bi-LSTM)

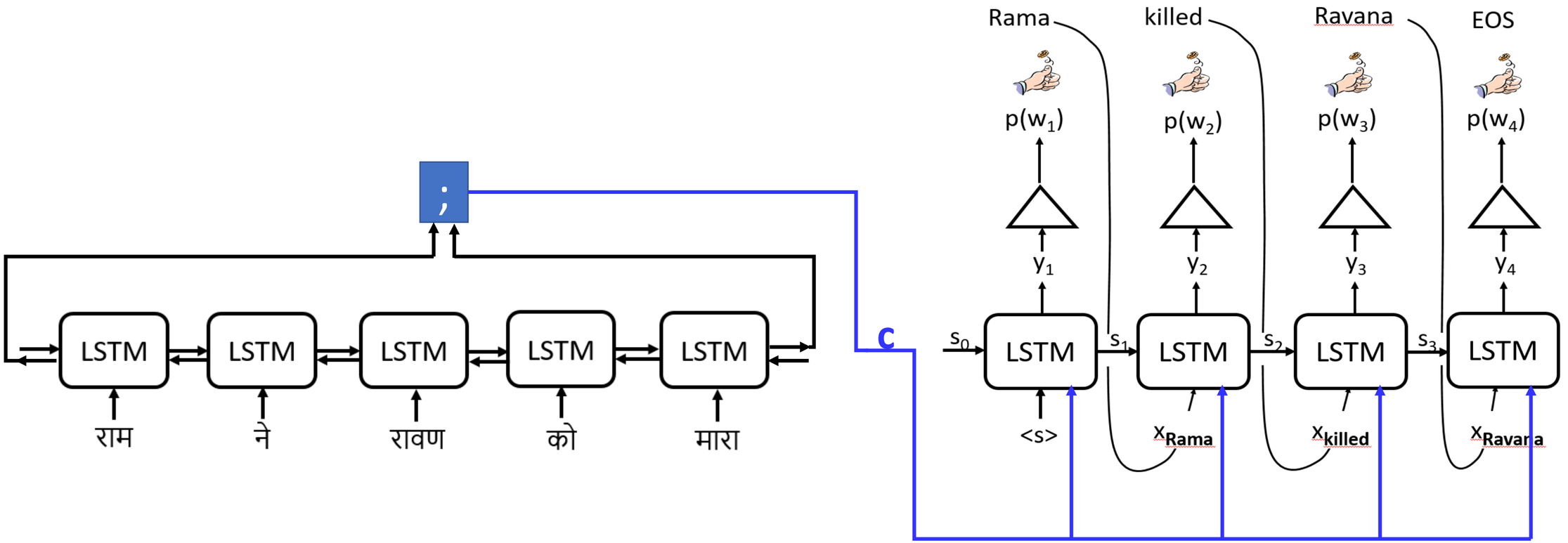
Decoder
(LSTM)

Idea 2: Encoder-Decoder

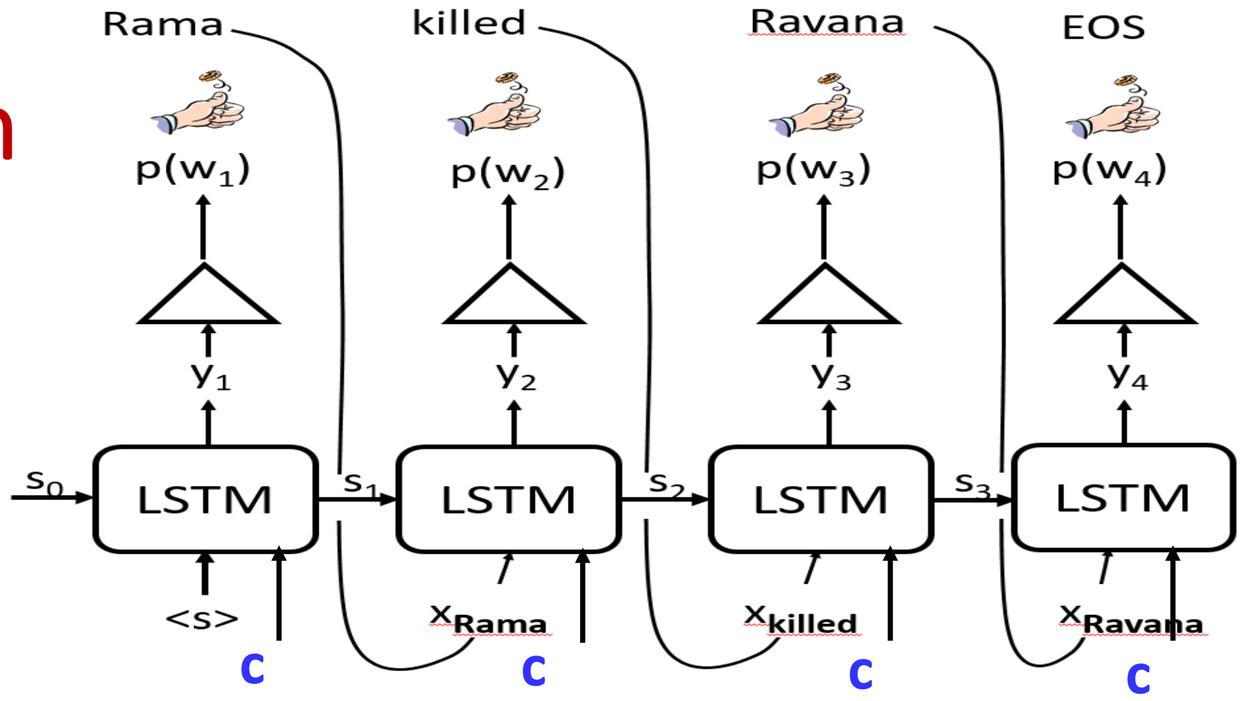
- Pass encoder output as input to *each* decoder unit
- Input at decoder = concat(c, $x_{\text{prev word}}$)

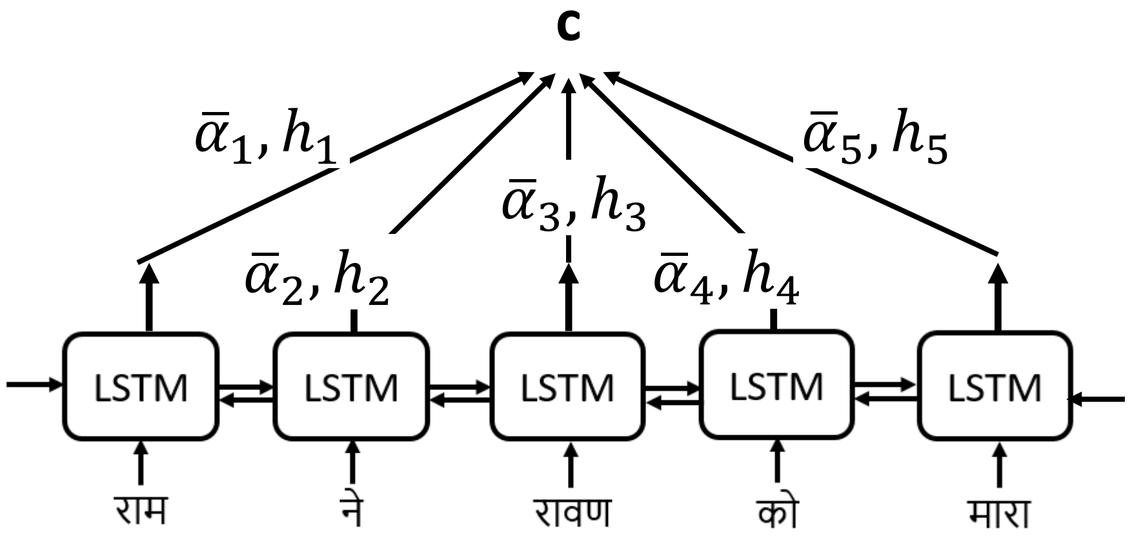


Seq2Seq without Attention



Seq2Seq with Attention



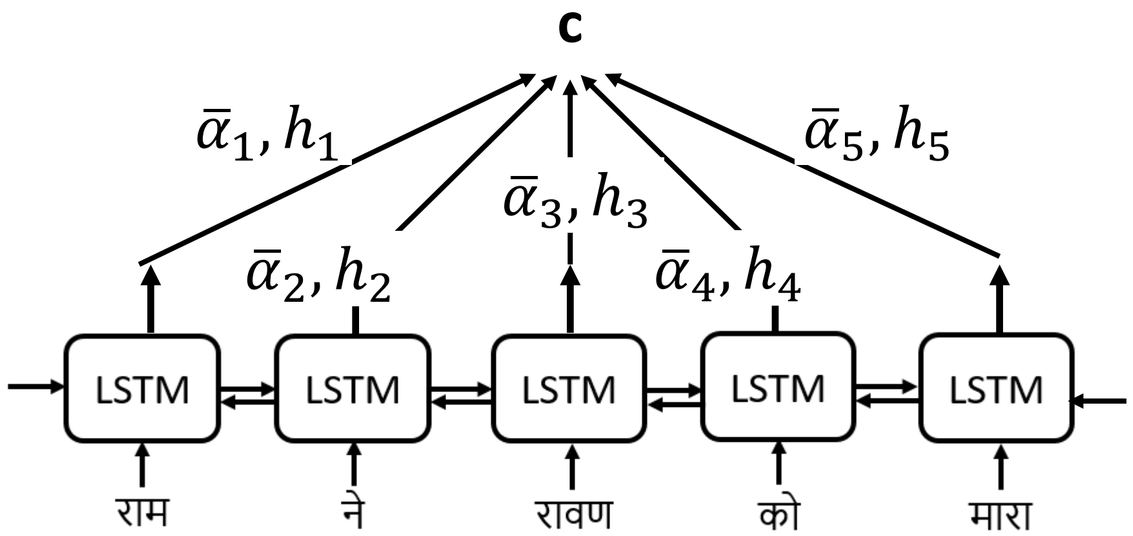


Multiple Encoded Vectors → Single Summary

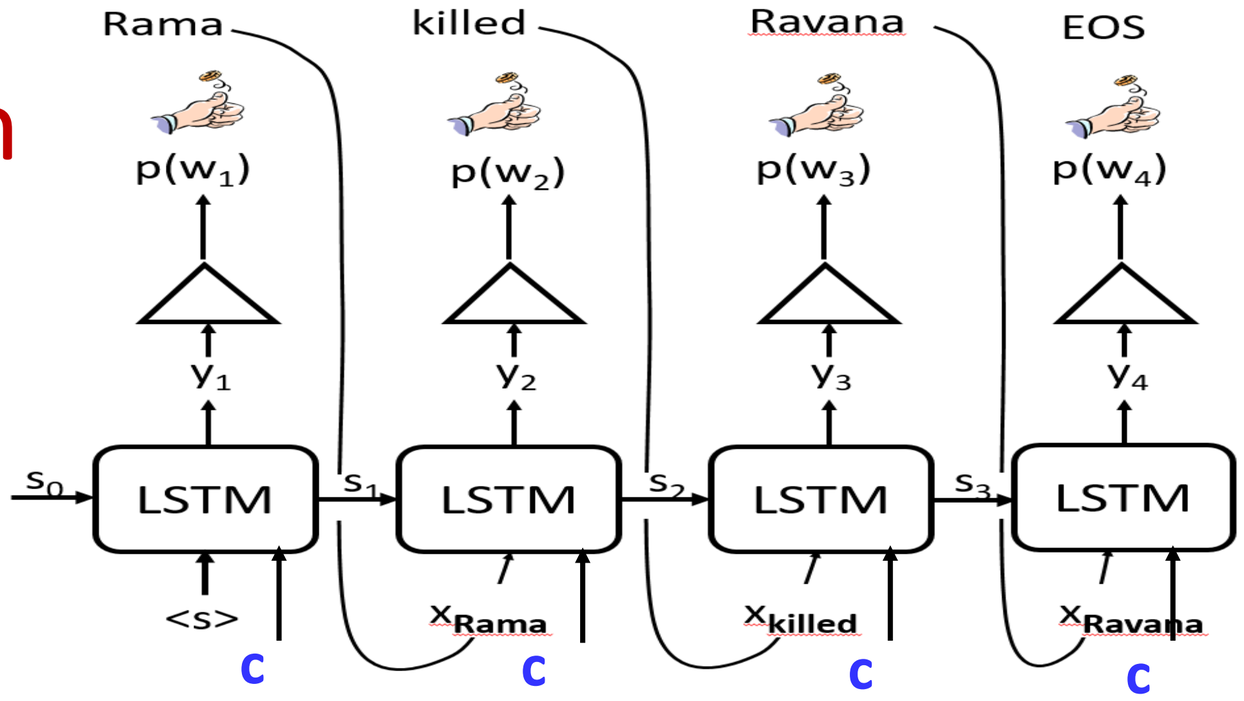
$$c = \sum_{i=1}^{T} \alpha_i . h_i$$

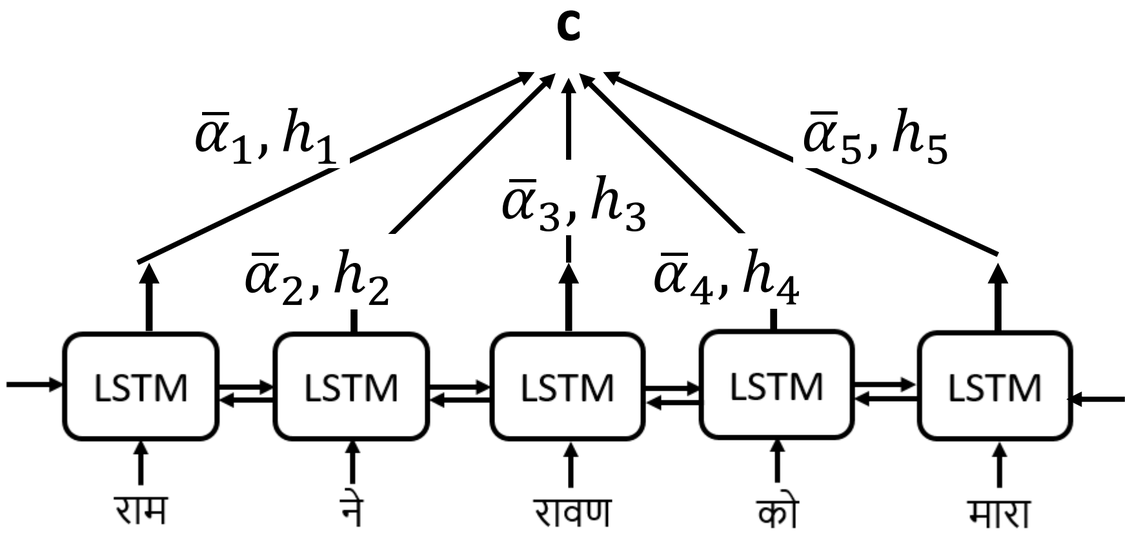
$$\alpha_{1:T} = \text{softmax}(\bar{\alpha}_1, \bar{\alpha}_2, \ldots, \bar{\alpha}_T)$$

$$\bar{\alpha}_i = \phi^{\text{att}}(q, h_i)$$

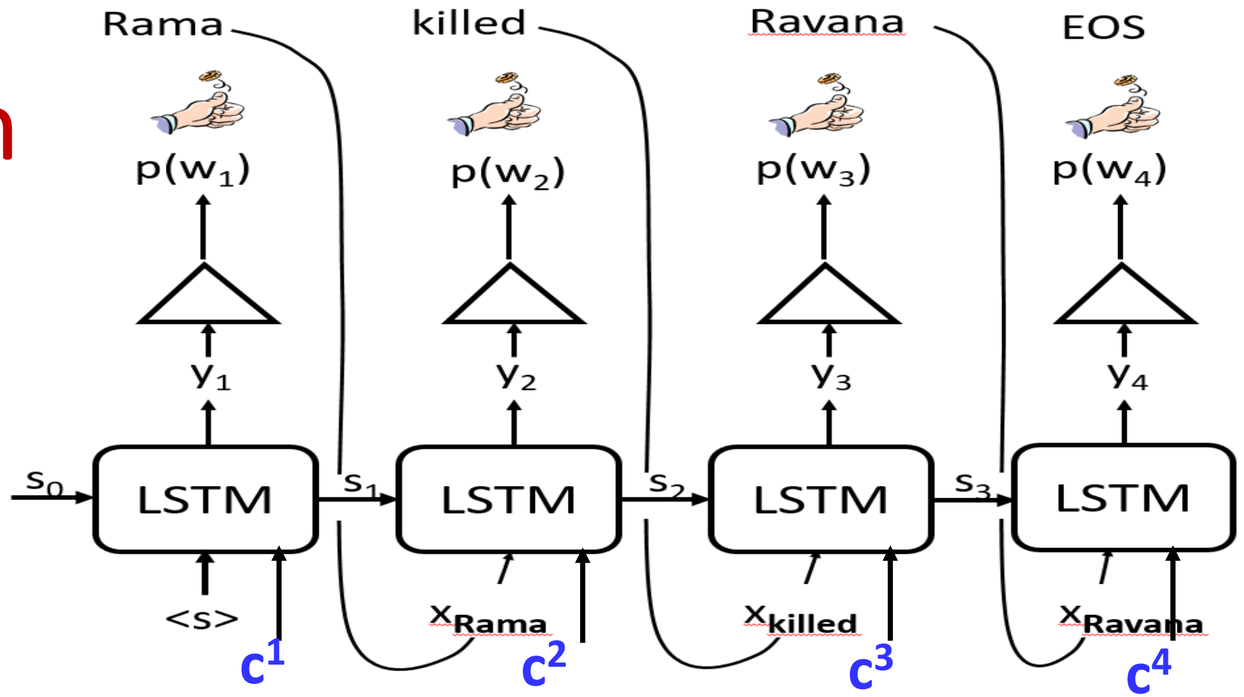


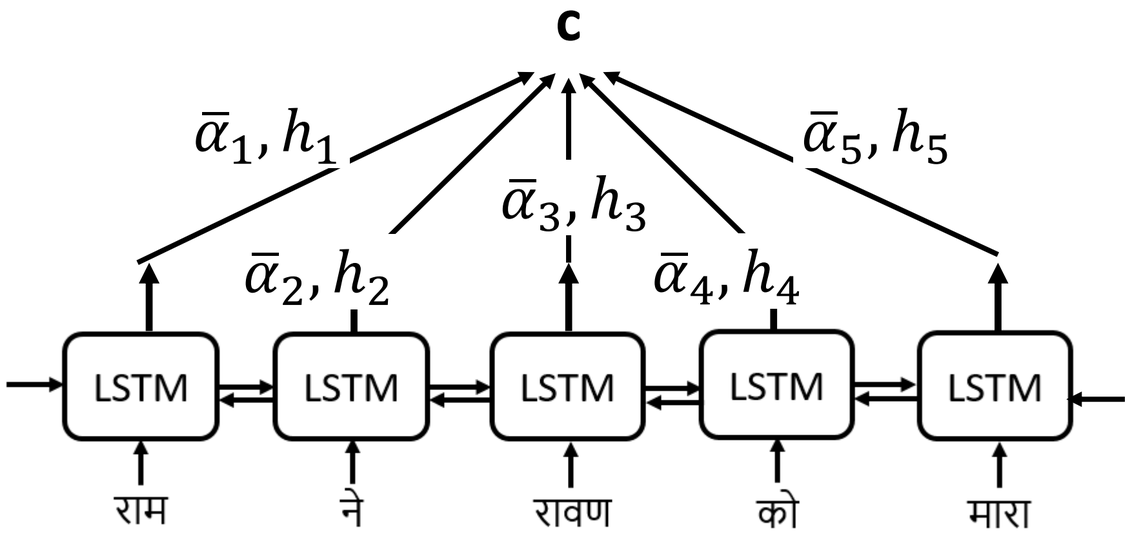
Seq2Seq with Attention



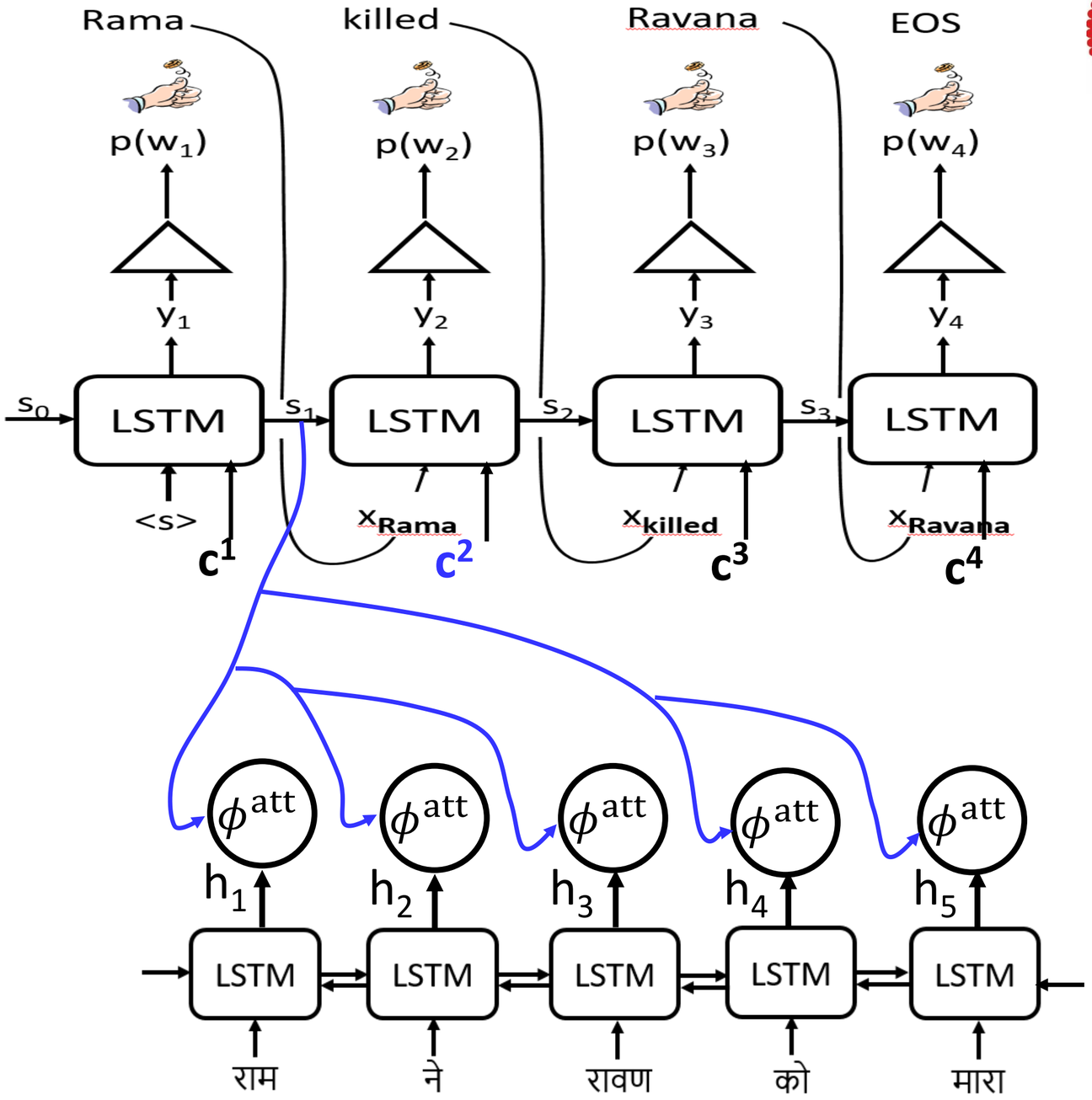


Seq2Seq with Attention

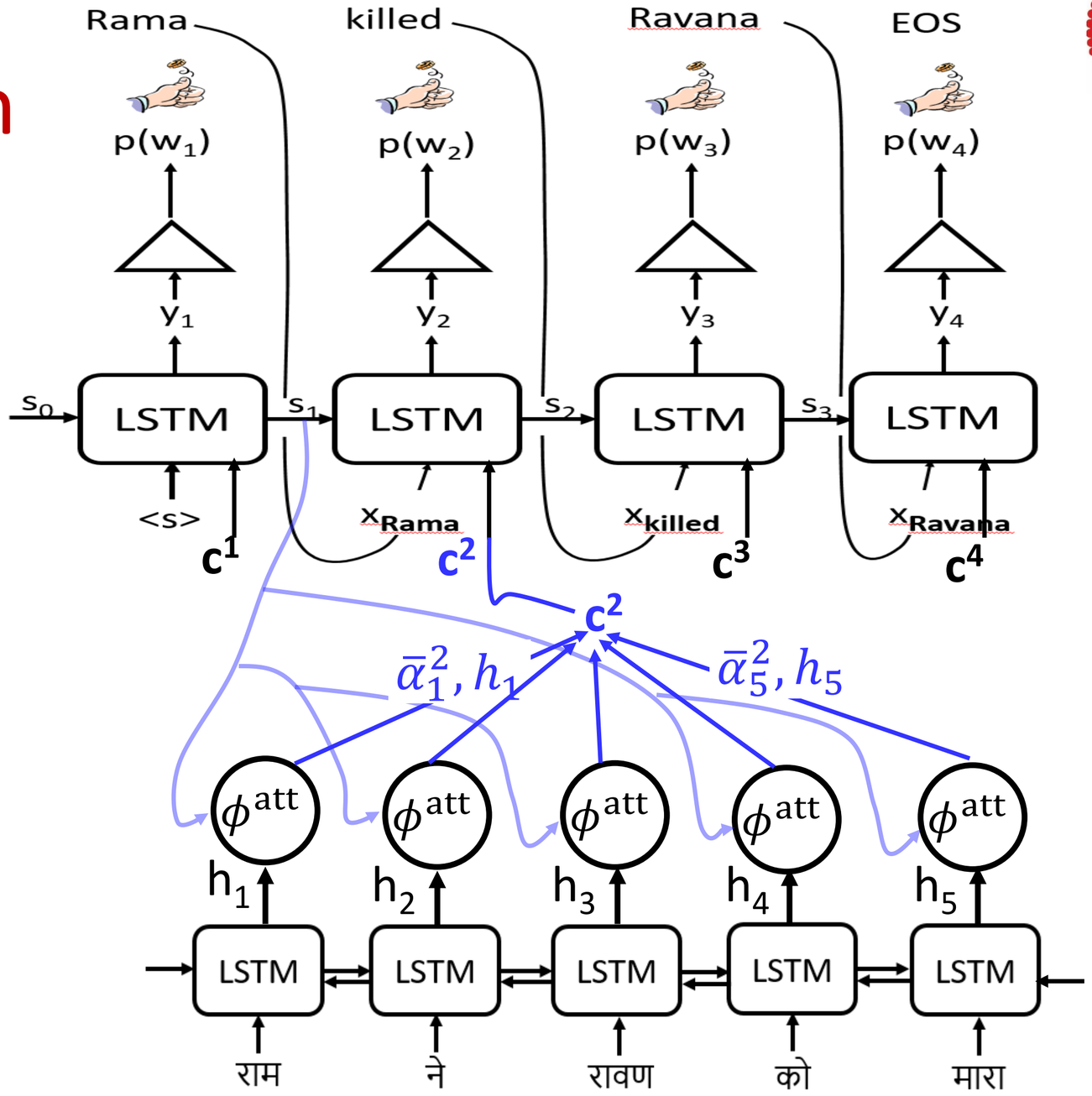




Seq2Seq with Attentior



Seq2Seq with Attention



Seq2Seq with Attention

$$h_{1:T} = \text{biLSTM}_{enc}(x_{1:T})$$

$$\bar{\alpha}_i^j = \phi^{\text{att}}(s_{j-1}, h_i)$$

$$\alpha^j = \text{softmax}\left(\bar{\alpha}_1^j, \bar{\alpha}_2^j, \ldots, \bar{\alpha}_T^j\right)$$

$$c^j = \sum_{i=1}^{T} \alpha_i^j . h_i$$

$$s_j = \text{LSTM}_{dec}\left(s_{j-1}, x_{z[j-1]}, c^j\right)$$

$$p_j(w) = \text{softmax}\left(\text{MLP}^{\text{out}}(s_j)\right)$$

$$z[j] \sim p_j(w)$$

Encoder-Decoder with Attention

- Encoder encodes a sequence of vectors, $h_1,...,h_T$
- At each decoding stage, MLP ϕ assigns a relevance score to each Encoder vector.
- The relevance score is based on h_i and the state s_{j-1}
- Weighted-sum (based on relevance) is used to produce the conditioning context for decoder step j.

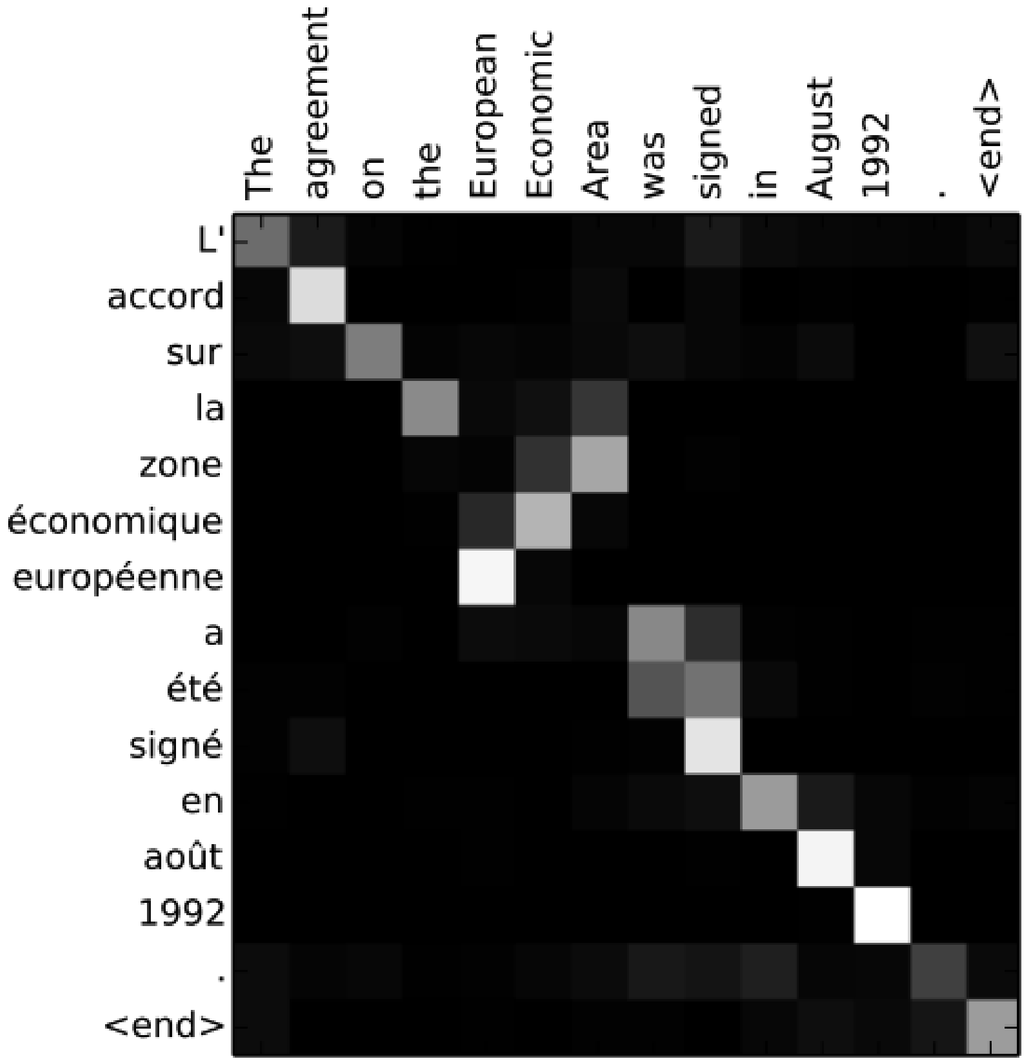
Encoder-Decoder with Attention

- Decoder "pays attention" to different parts of encoded sequence at each stage.
- The attention mechanism is "soft" -- it is a mixture of encoder states.

- The encoder acts as a read-only memory for the decoder
- The decoder chooses what to read at each stage

- Complexity
  - Encoder Decoder: O(n+m)
  - Encoder Decoder w/ Attention: O(nm)

Attention

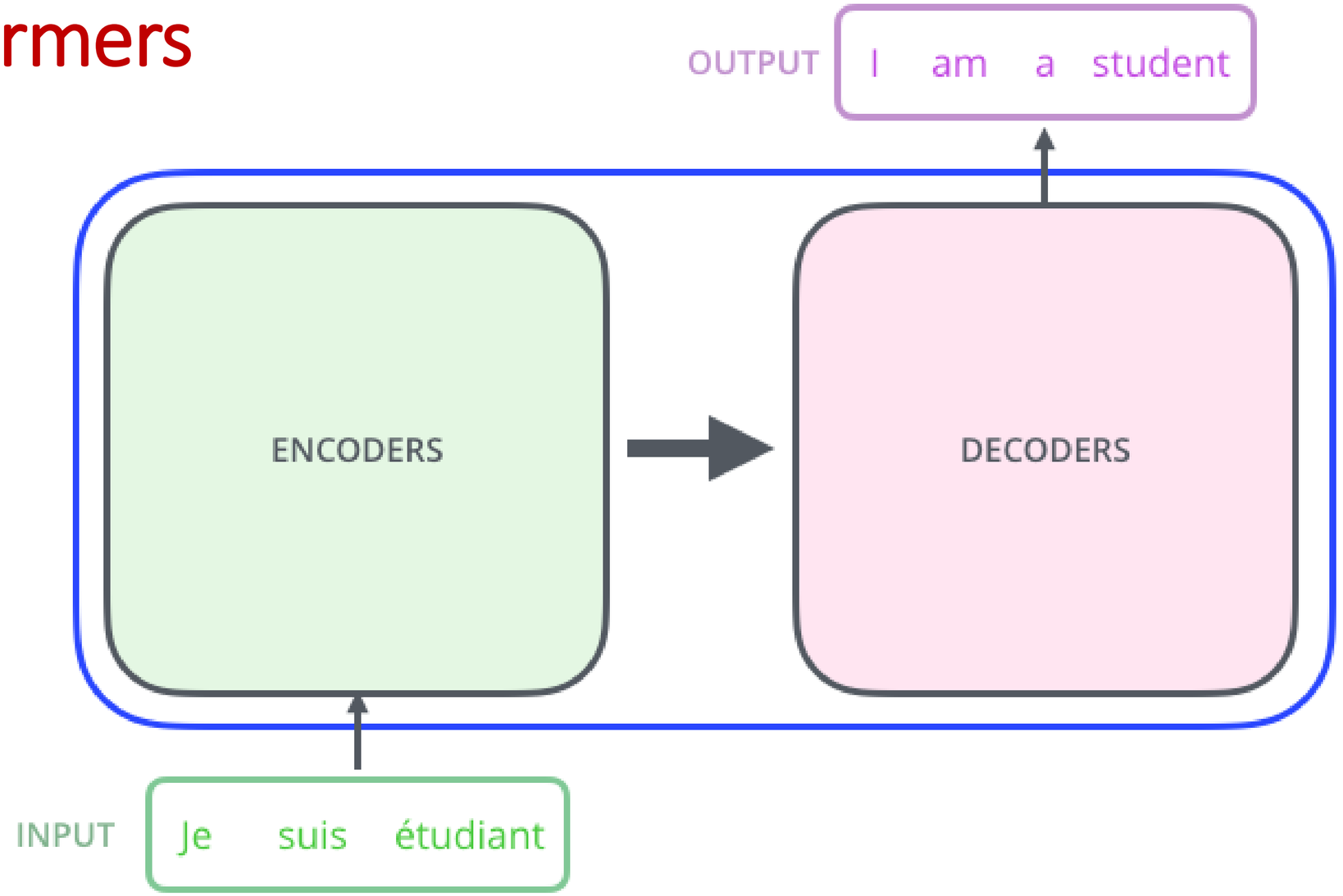

The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

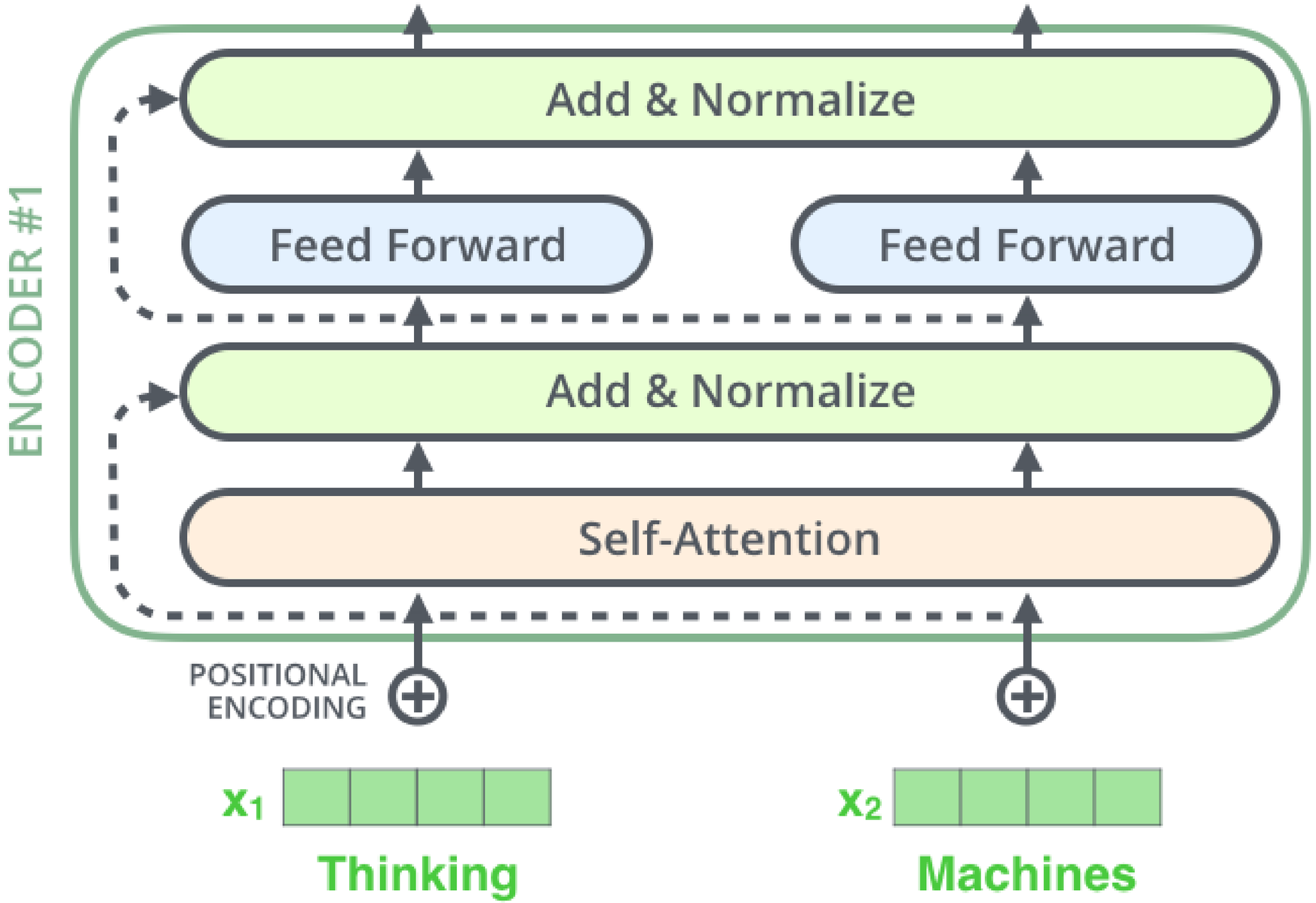
Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo)
- Pre-trained Language Models: Transformers (GPT, BERT)

Transformers





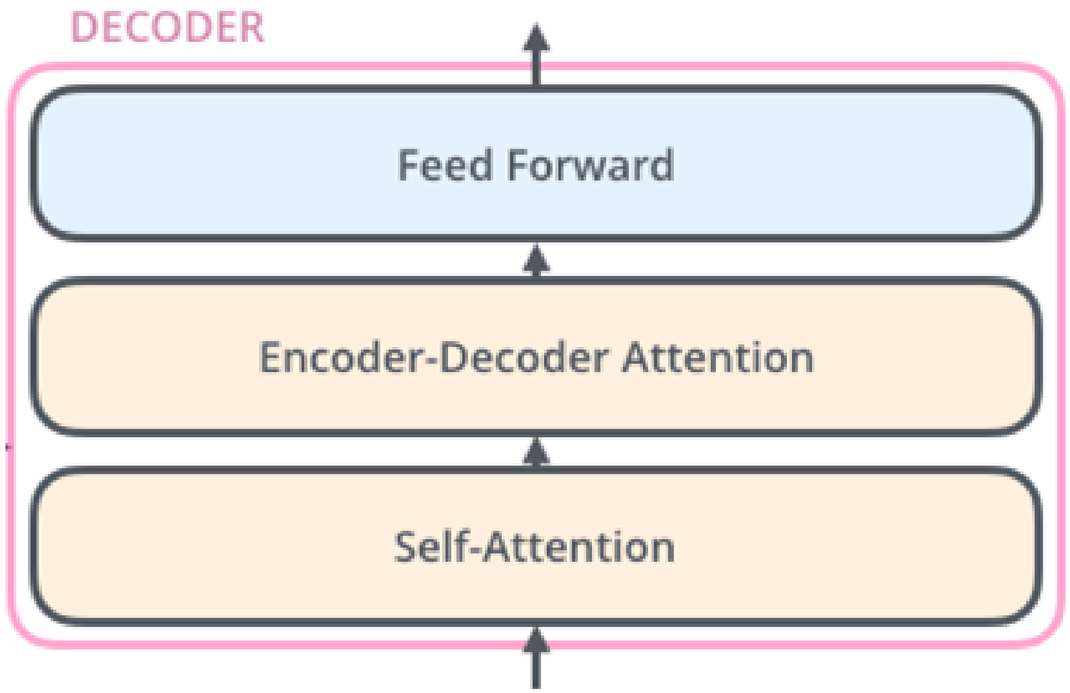
Adding residual connections...

Images from https://jalammar.github.io/illustrated-transformer/

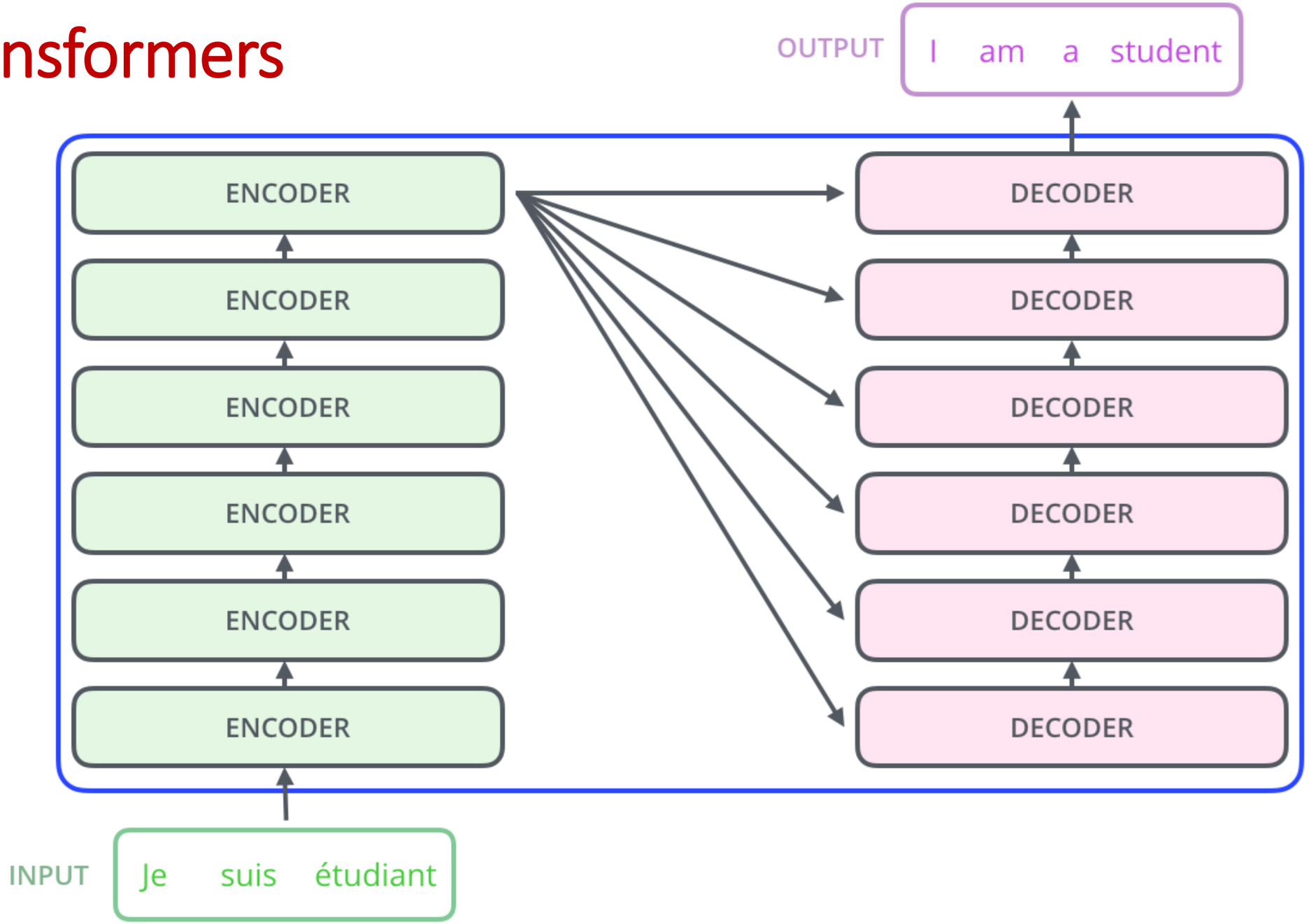
Decoders

Two key differences from encoder:

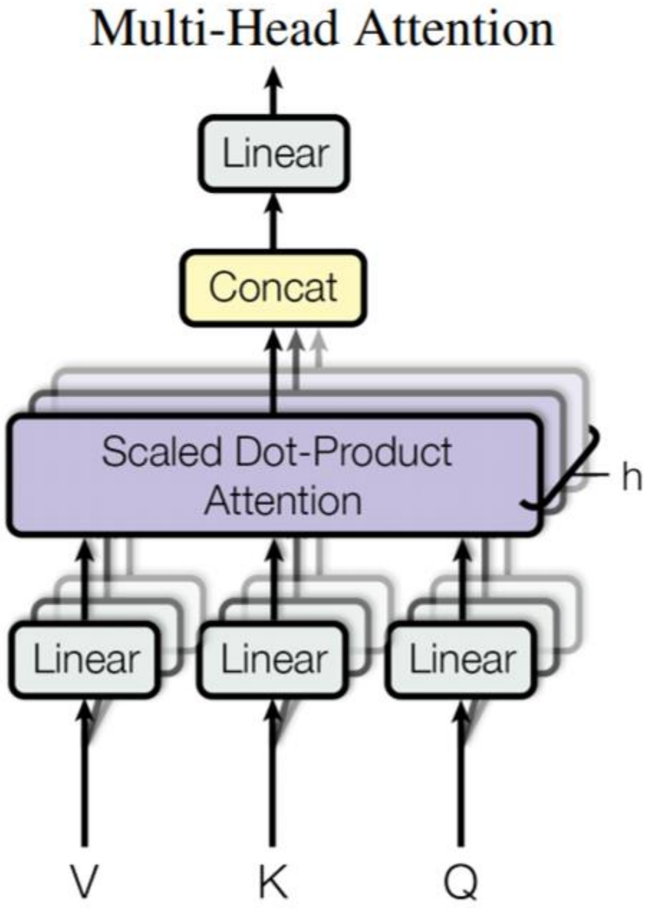
- Self-attention only on words generated uptil now, not on whole sentence.
- Additional encoder-decoder attention layer where keys, values come from last encoder layer.

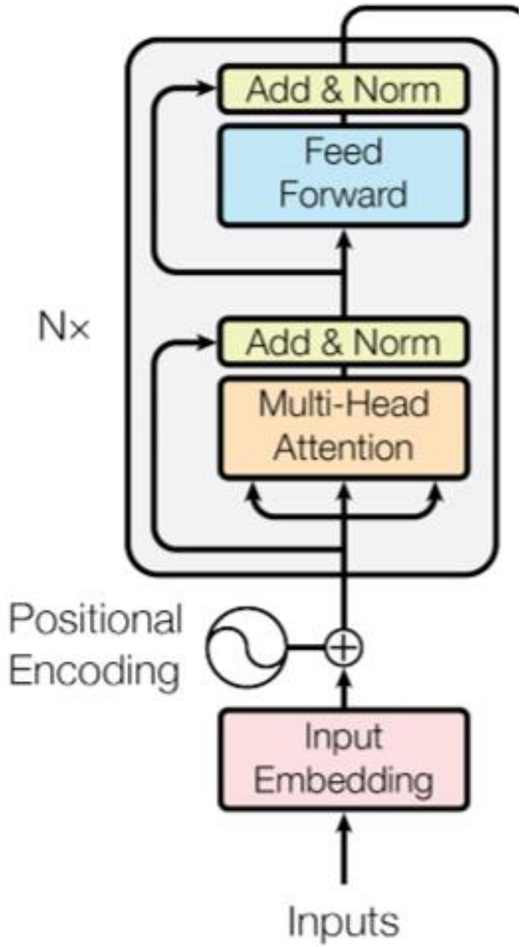


Transformers

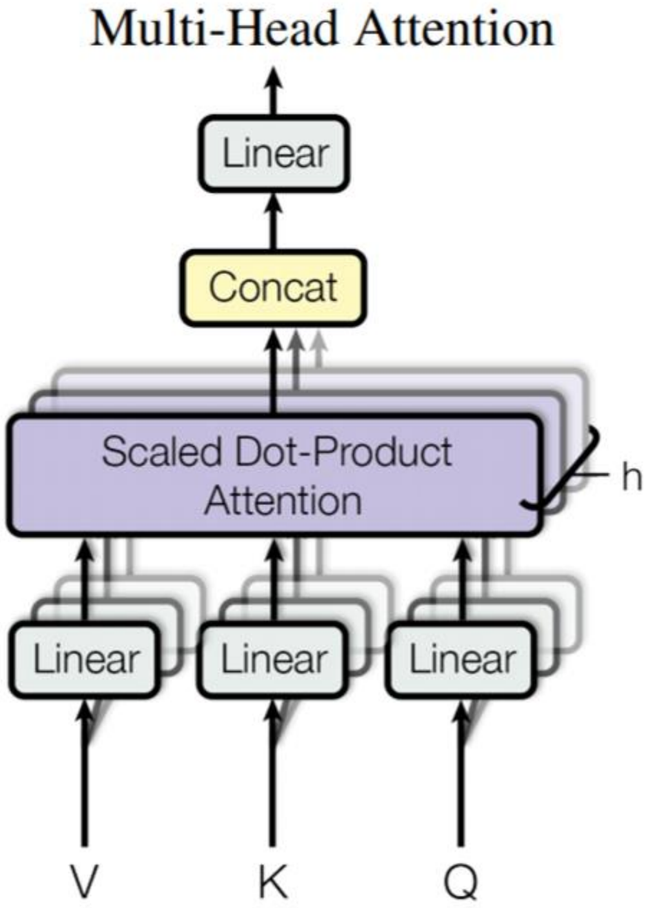


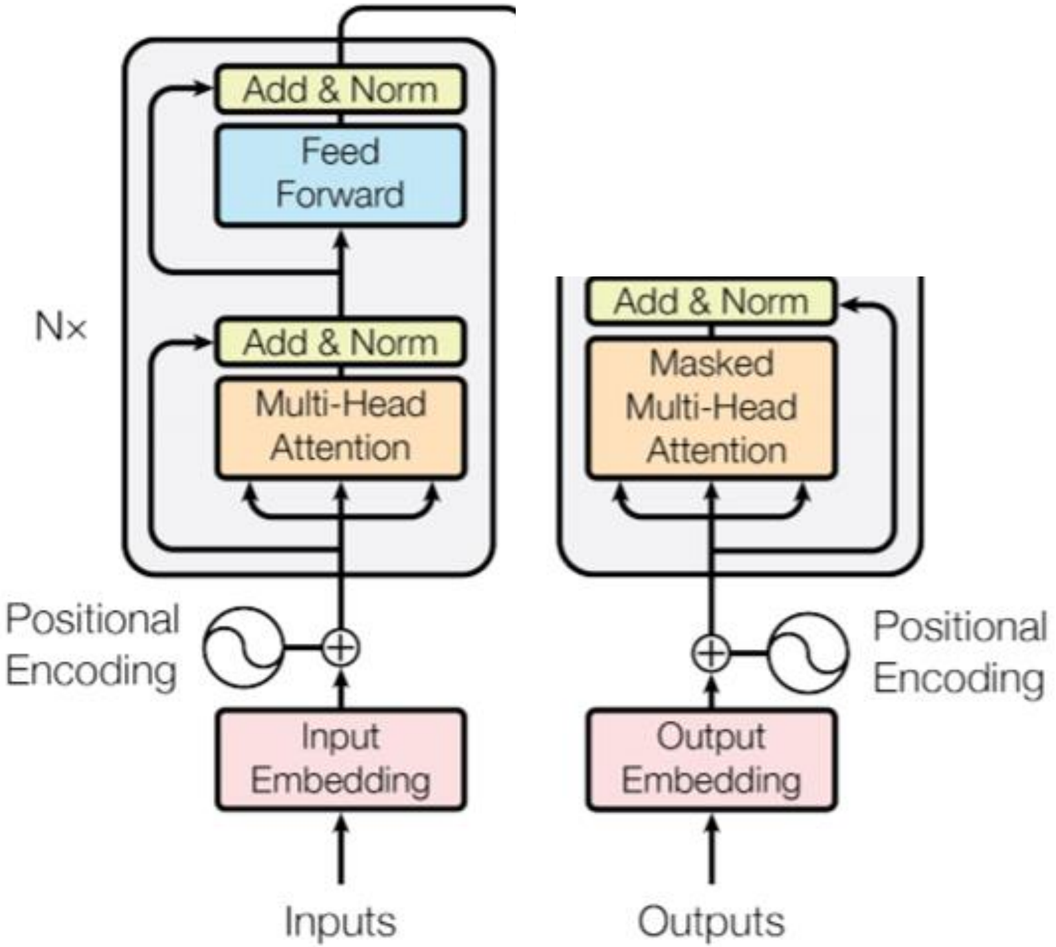
Full architecture with Attention reference



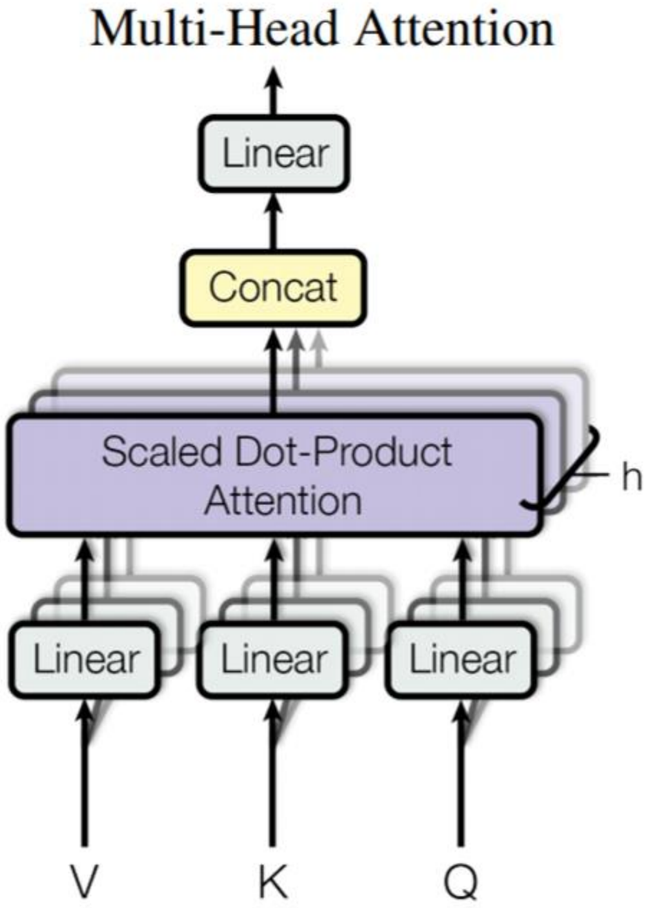


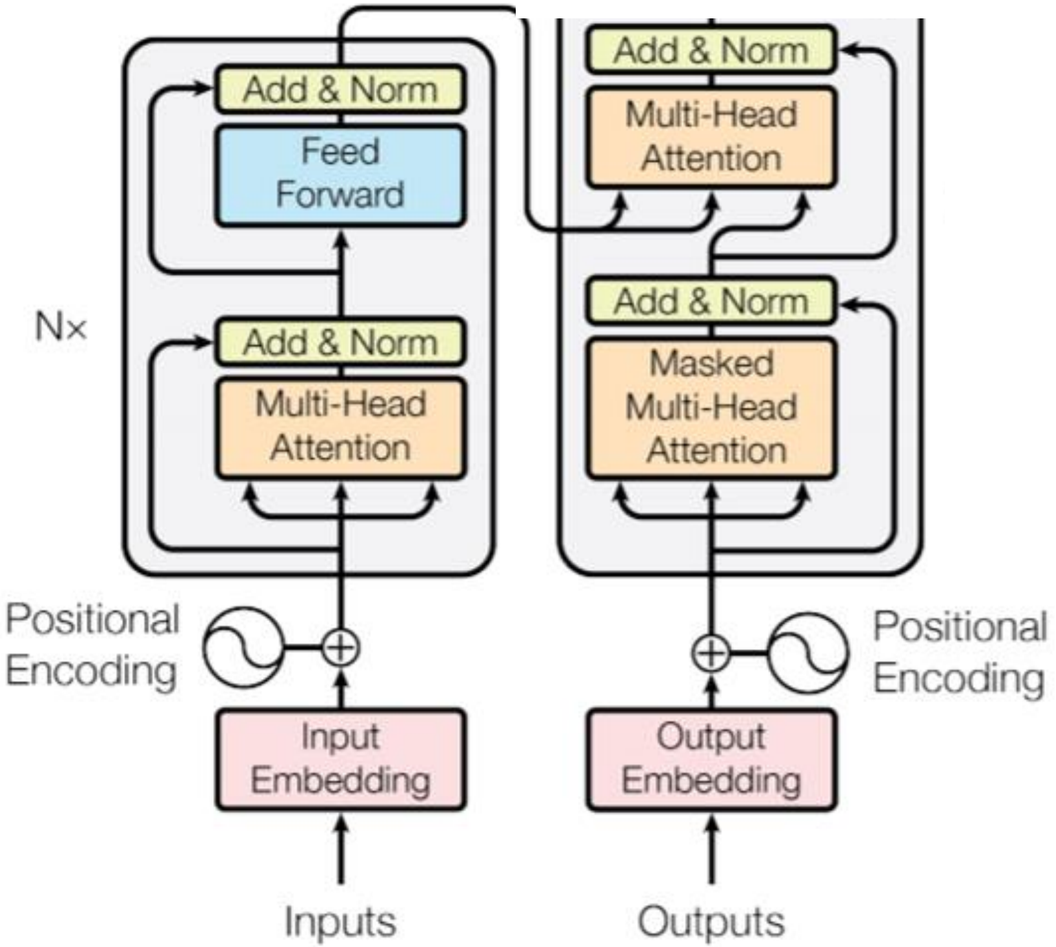
Full architecture with Attention reference



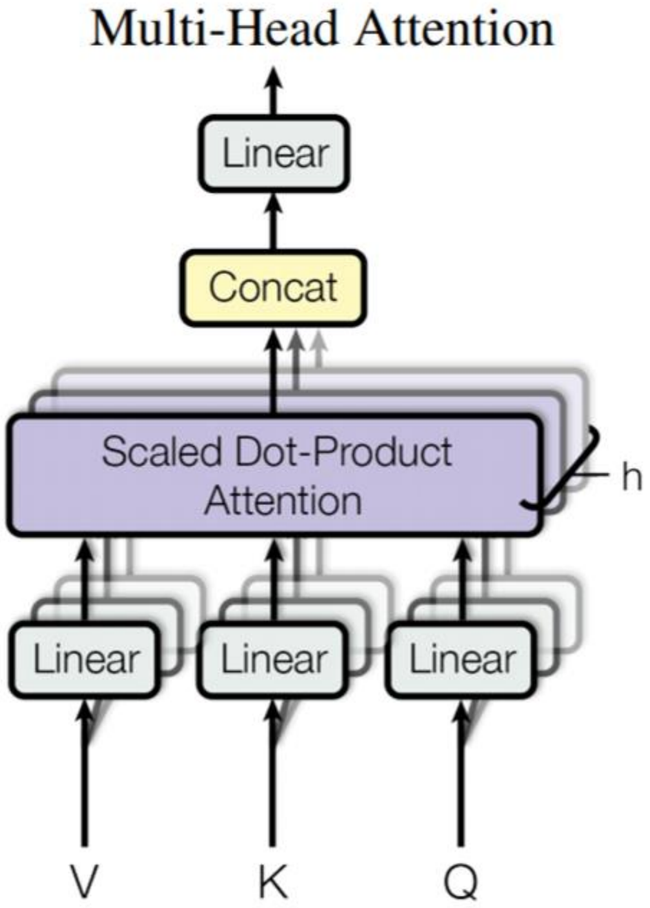


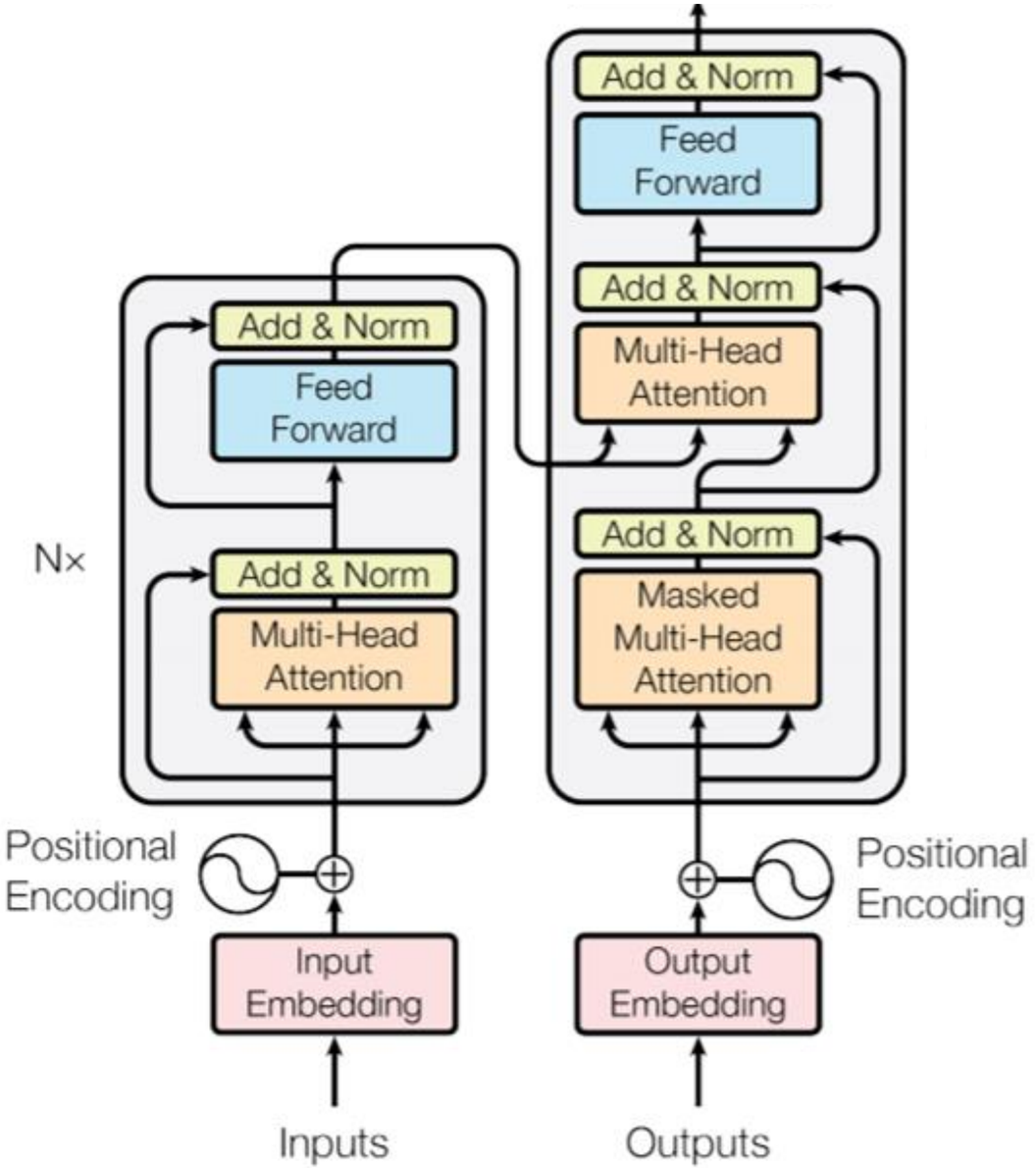
Full architecture with Attention reference



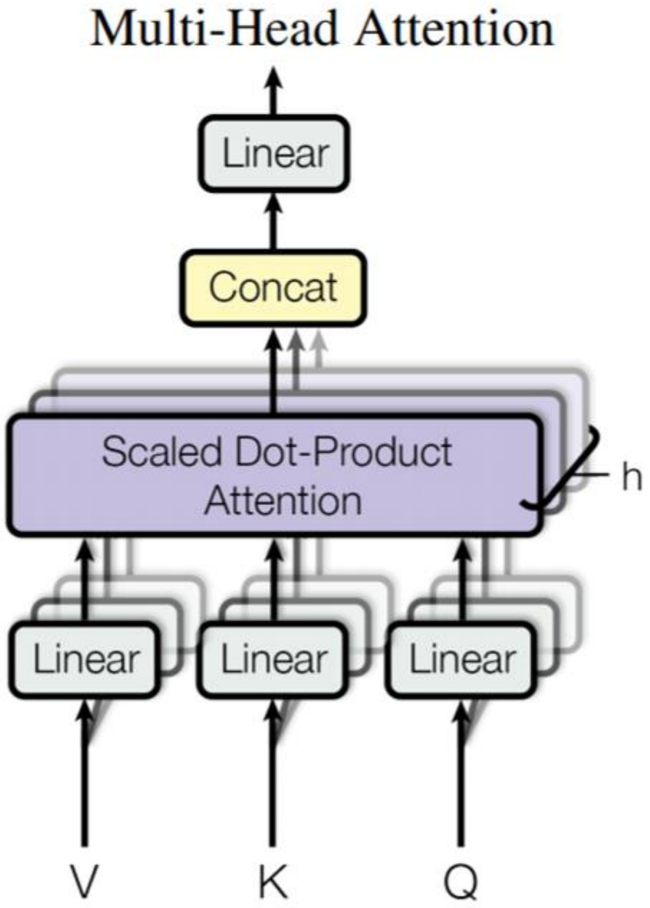


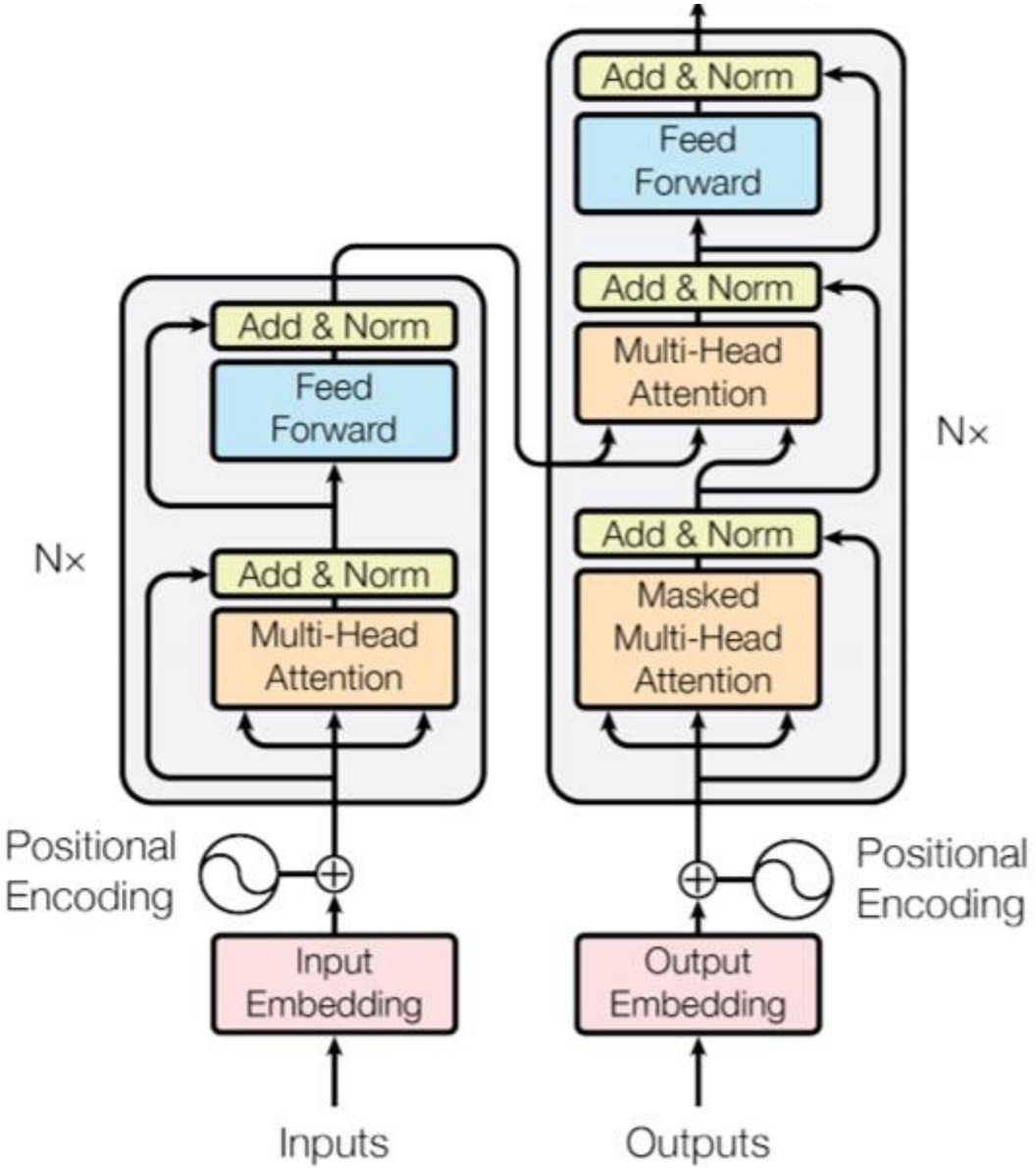
Full architecture with Attention reference



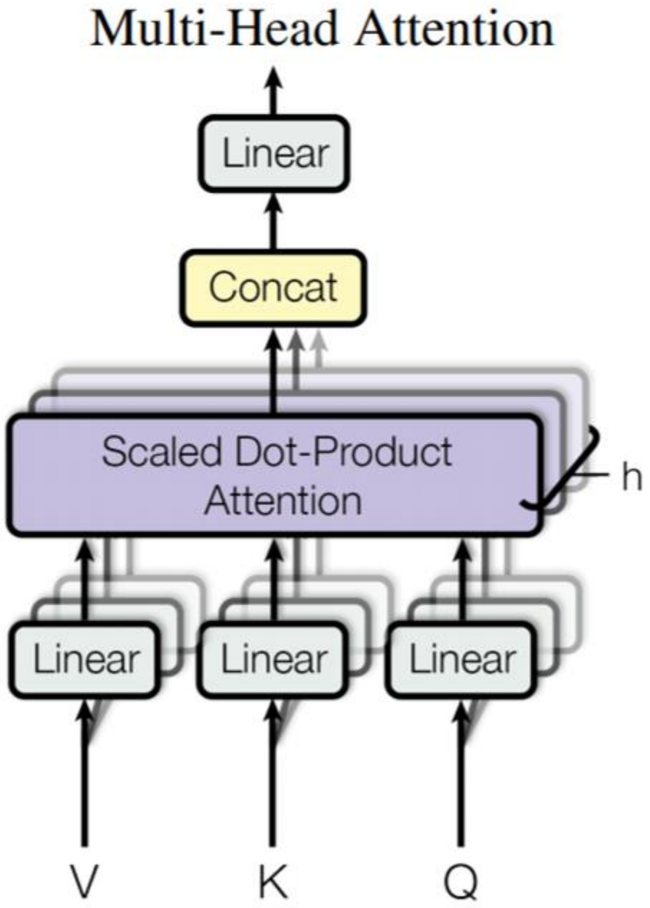


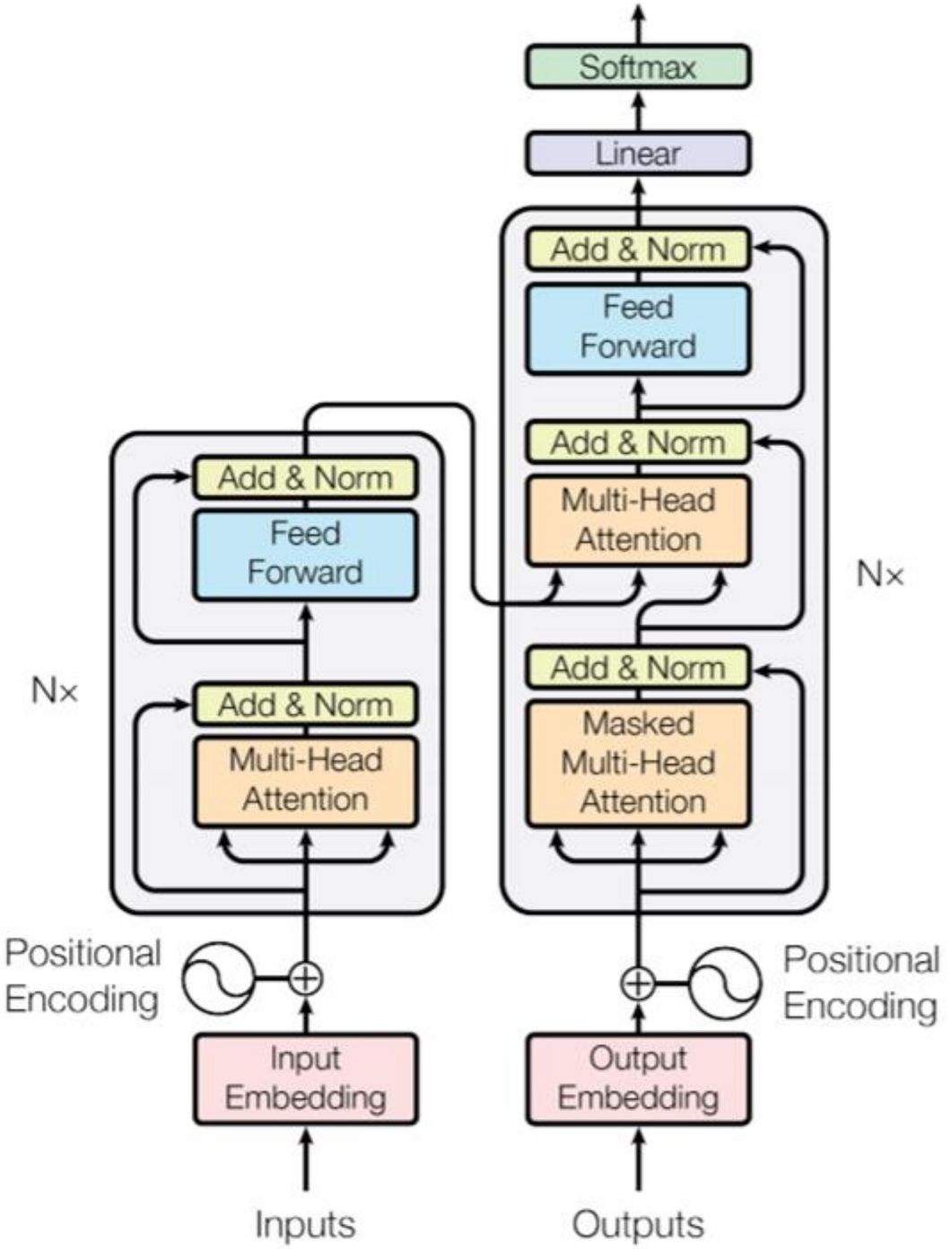
Full architecture with Attention reference





Full architecture with Attention reference





Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

- Pre-trained Language Models: LSTMs (ELMo)
- Pre-trained Language Models: Transformers (GPT, BERT)

Pretraining

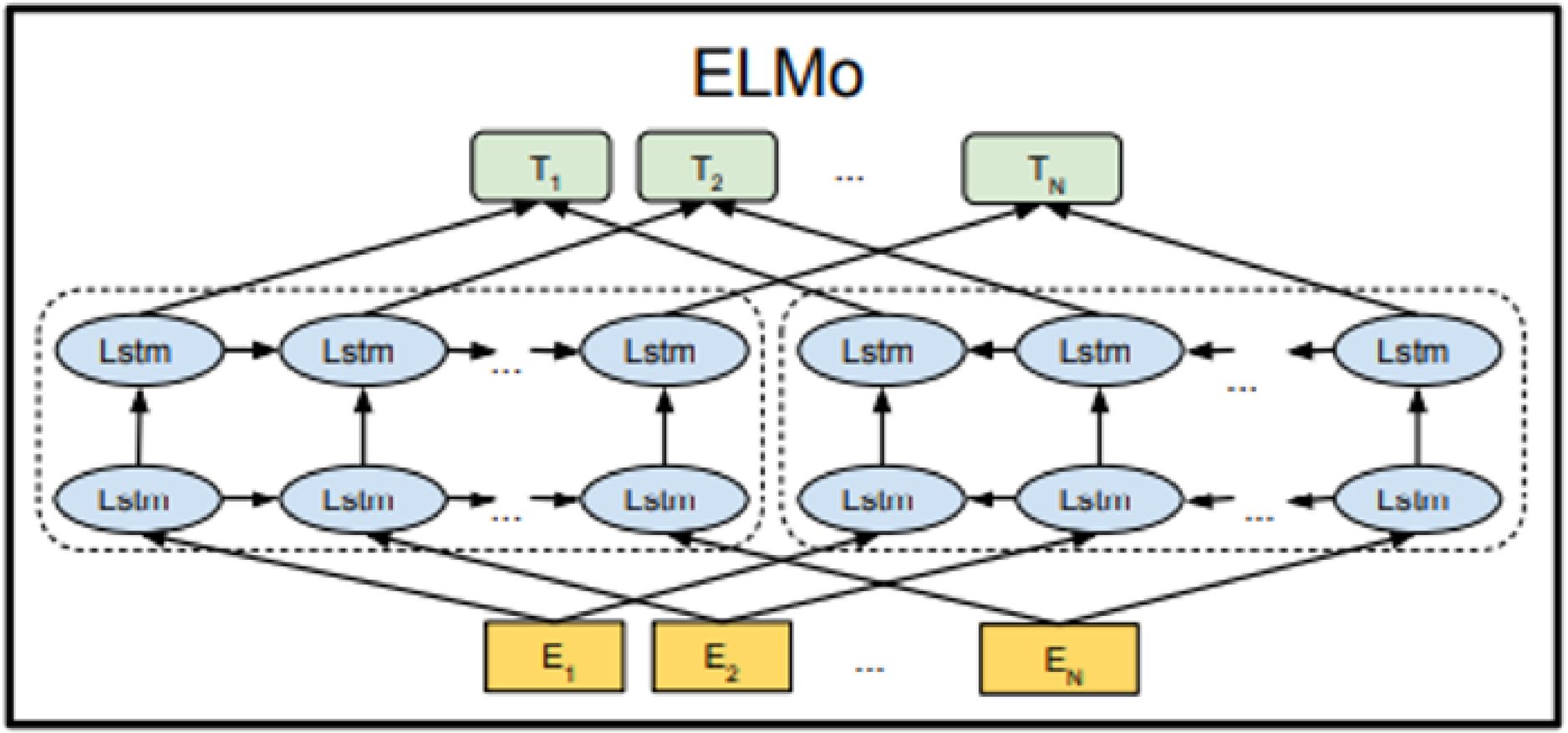
- In NLP, we are interested in solving a variety of end tasks - QA, Search, etc.
- One approach - train neural models from scratch
- Issue - This involves two things
  - Modelling of Syntax and Semantics of the language
  - Modelling of the end-task
- Pretraining: Learns the modelling of syntax and semantics - through another task
- So the model can focus exclusively on modelling of end-task

Pretraining

- Which base task to choose:
  - Must have abundant data available
  - Must require learning of syntax and semantics

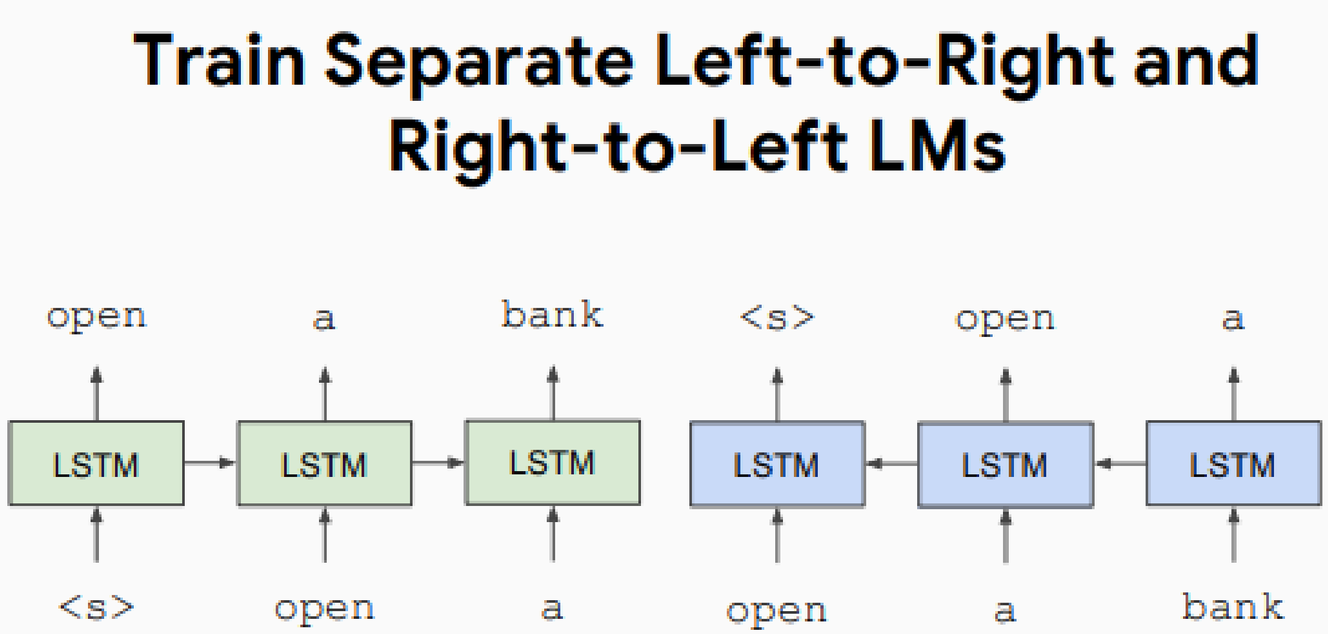
- Solution: Language Modelling
  - Does not require human annotated labels - abundance of sentences
  - Requires understanding of both syntax and semantics to predict the next word in sentence

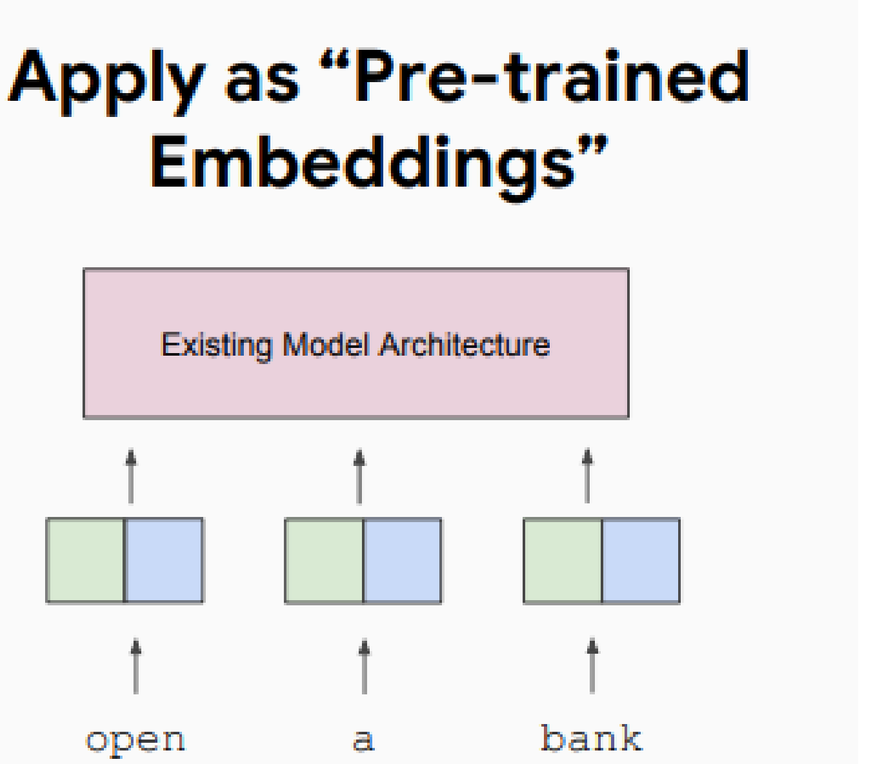
Model 1: ELMo (two LSTMs)



ELMo (Contextualized Embeddings)

- Bidirectional language modelling: separate forward and backward LSTMs
- Issue: Both LSTMs are not coupled with one another





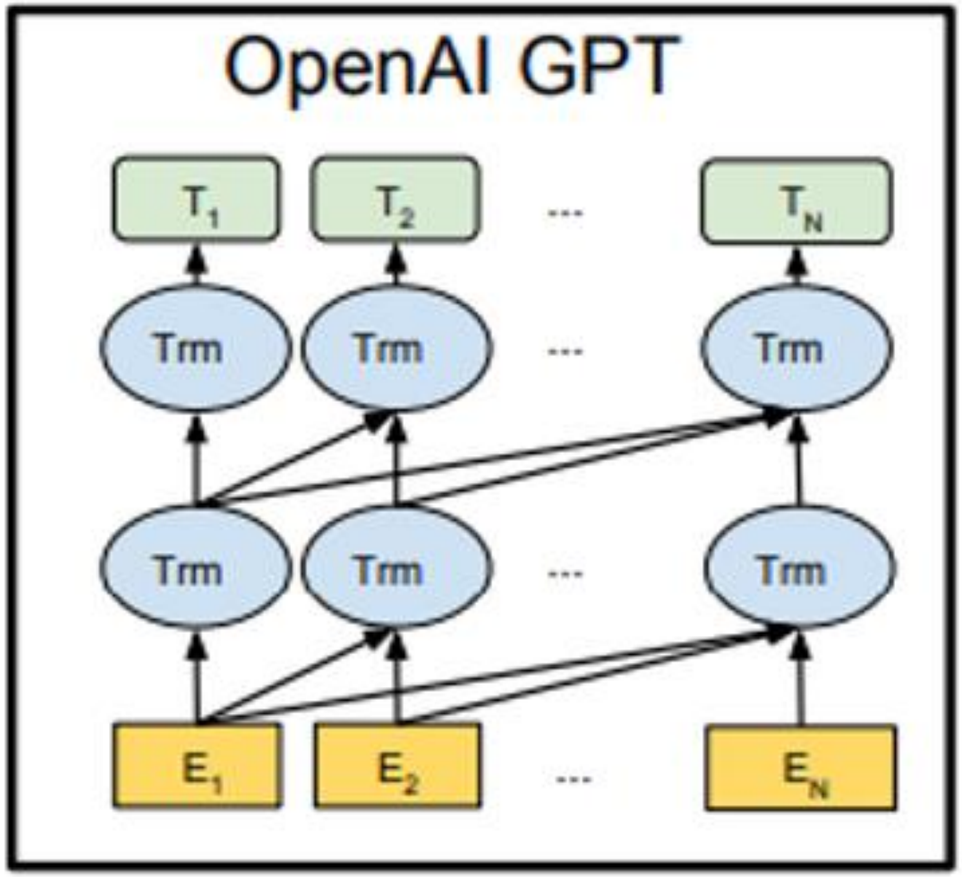
Reference: https://nlp.stanford.edu/seminar/details/jdevlin.pdf

Outline

- Neural Language Models: LSTMs
- Seq2Seq Models with LSTMs
- Neural Language Models: Transformers

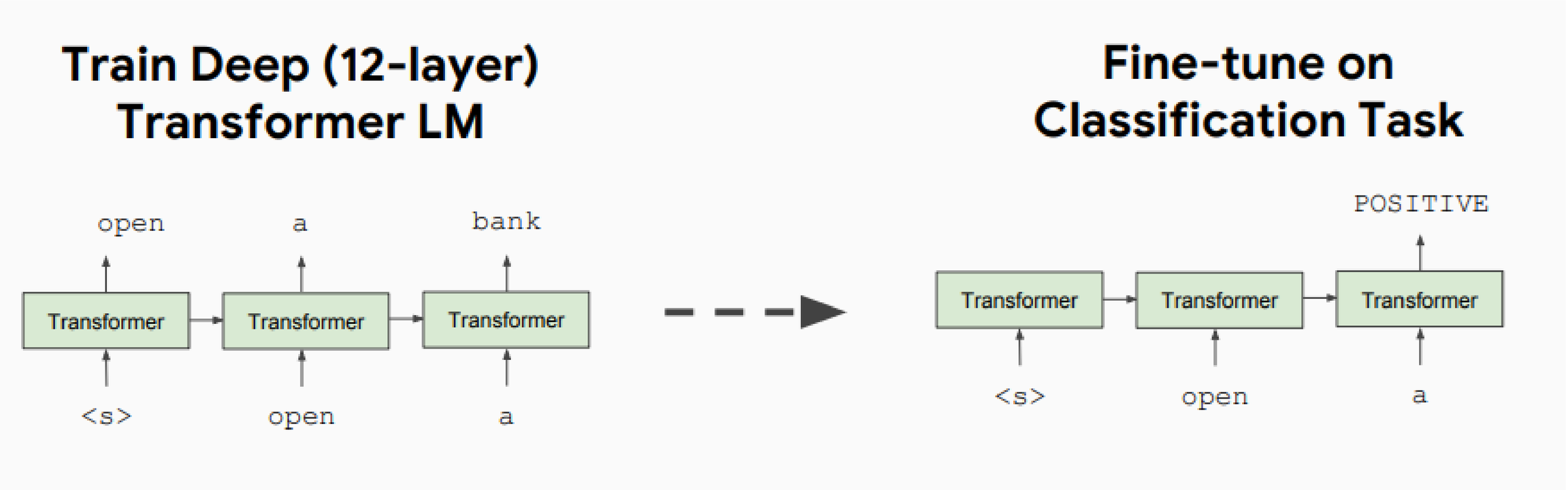
- Pre-trained Language Models: LSTMs (ELMo)
- Pre-trained Language Models: Transformers (GPT, BERT)

Model 2: Generative Pre-Training (Transformers)



Generative Pre-Training

- GPT - Uses Transformer decoder instead of LSTM for Language Modeling
- GPT-2 - Trained on larger corpus of text (40 GB) Model size:1.5 B parameters
- Can generate text given initial prompt - “unicorn” story, economist interview



Unicorn Story

SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

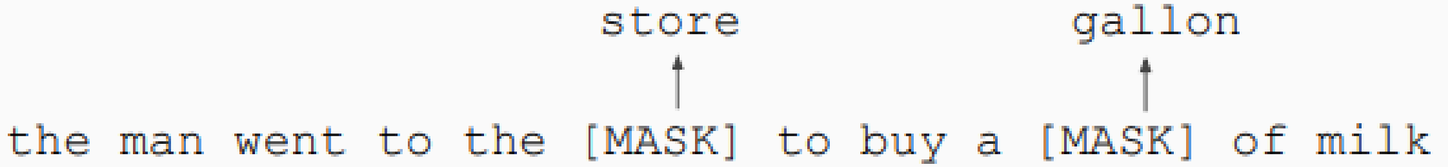
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Model 3: Masked language modeling (BERT)

- GPT/language model task is unidirectional.
- Tasks like classification - we already know all the words –
- Bidirectional context required for end tasks:
- using unidirectional model is sub-optimal

- **Solution**: Mask out *k*% of the input words, and then predict the masked words
  - We always use *k* = 15%

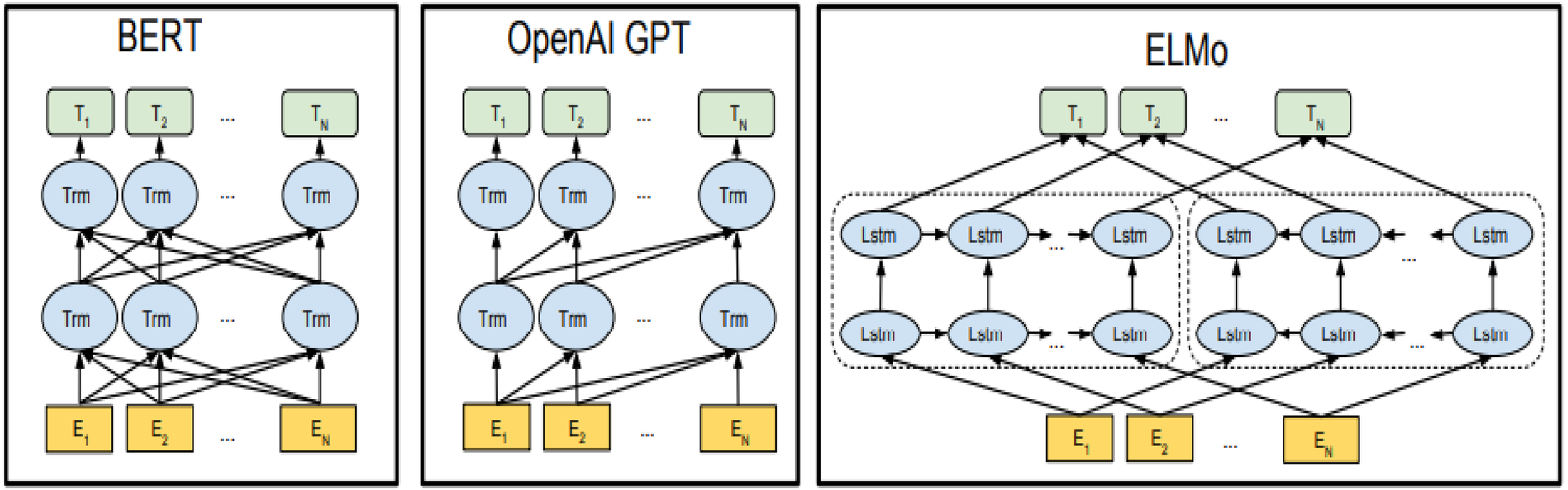


Solution: Masked Language Modelling

- Issue with Language modelling - Unidirectional
- Cannot train model on bidirectional context - required for many end tasks
- Solution: Masked Language Modelling
  - Randomly mask a word in the sentence
  - Train the model to predict it

*Image Credits: [1]

BERT vs. OpenAI-GPT vs. ELMo



Bidirectional

Unidirectional

De-coupled Bidirectionality

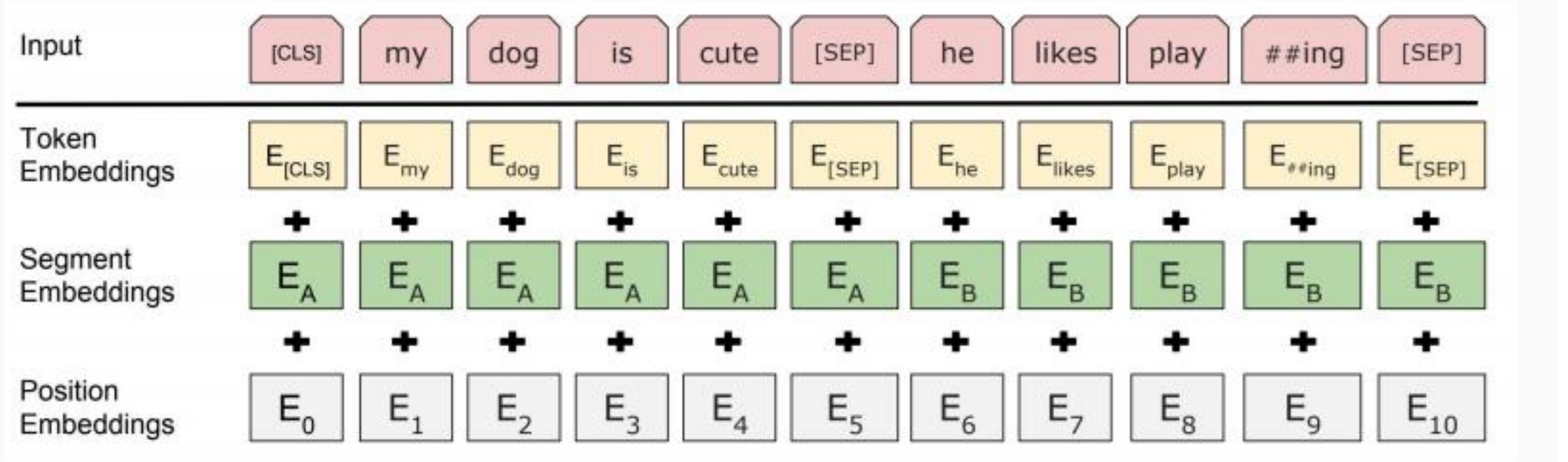
Word-Piece tokenizer

- Middle ground between character level and word level representations
- tweeting → tweet + ##ing
- xanax → xa + ##nax
- Technique originally taken from paper for Japanese and Korean languages from a speech conference

- Given a training corpus and a number of desired tokens D, the optimization problem is to select D wordpieces such that the resulting corpus is minimal in the number of wordpieces when segmented according to the chosen wordpiece model.

Schuster, Mike, and Kaisuke Nakajima. "Japanese and korean voice search." *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012.

Input Representation



- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings

Practical Tips

- Proper modelling of input for BERT is extremely important
  - Question Answering: [CLS] Query [SEP] Passage [SEP]
  - Natural Language Inference: [CLS] Sent1 [SEP] Sent2 [SEP]
  - BERT CLS cannot be used as a general purpose sentence encoder for retrieval
- Maximum input length is limited to 512. Truncation strategies have to be adopted
- BERT-Large model requires random restarts to work
- Always PRE-TRAIN, on related task - will improve accuracy

Small Hyperparameter search

- Because of using a pre-trained model - we can’t really change the model architecture any more
- Number of hyper-parameters are actually few:
  - Batch Size: 16, 32
  - Learning Rate: 3e-6, 1e-5, 3e-5, 5e-5
  - Number of epochs to run
- Compare to LSTMs where we need to decide number of layers, the optimizer, the hidden size, the embedding size, etc…
- This greatly simplifies using the model

Implementation for fine-tuning

- Using BERT requires 3 modules
  - Tokenization, Model and Optimizer
- Originally developed in Tensorflow
- HuggingFace ported it to Pytorch and to-date remains the most popular way of using BERT (18K stars)
- Tensorflow 2.0 also has a very compact way of using it - from TensorflowHub
  - But fewer people use it, so support is low

- Keshav's choice - use HuggingFace BERT API
  - Lightning provides a Keras-like API for Pytorch

Self-Supervised Learning

Yann LeCun shared a photo.

30 April 2019 ·

I now call it "self-supervised learning", because "unsupervised" is both a loaded and confusing term.

In self-supervised learning, the system learns to predict part of its input from other parts of it input. In other words a portion of the input is used as a supervisory signal to a predictor fed with the remaining portion of the input.

Self-supervised learning uses way more supervisory signals than supervised learning, and enormously more than reinforcement learning. That's why calling it "unsupervised" is totally misleading. That's also why more knowledge about the structure of the world can be learned through self-supervised learning than from the other two paradigms: the data is unlimited, and amount of feedback provided by each example is huge.

Roberta: A Robustly Optimized BERT Pretraining Approach

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.