

Language Modeling

Mausam

(Based on slides of Michael Collins, Dan Jurafsky, Dan Klein, Chris Manning, Luke Zettlemoyer, Yejin Choi, Yoav Goldberg, Andrej Karpathy, Chris Manning, Graham Neubig, Jay Allamar and Keshav Kolluru)

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Evaluation
- Neural Language Modeling

The Language Modeling Problem

- **Setup:** Assume a (finite) vocabulary of words

$\mathcal{V} = \{\text{the, a, man, telescope, Beckham, two, Madrid, ...}\}$

- We can construct an (infinite) set of strings

$\mathcal{V}^\dagger = \{\text{the, a, the a, the fan, the man, the man with the telescope, ...}\}$

- **Data:** given a *training set* of example sentences $x \in \mathcal{V}^\dagger$

- **Problem:** estimate a probability distribution

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1$$

and $p(x) \geq 0$ for all $x \in \mathcal{V}^\dagger$

$$p(\text{the}) = 10^{-12}$$

$$p(\text{a}) = 10^{-13}$$

$$p(\text{the fan}) = 10^{-12}$$

$$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$$

$$p(\text{the fan saw saw}) = 10^{-15}$$

...

The Noisy-Channel Model

- We want to predict a sentence given acoustics:

$$w^* = \arg \max_w P(w|a)$$

- The noisy channel approach:

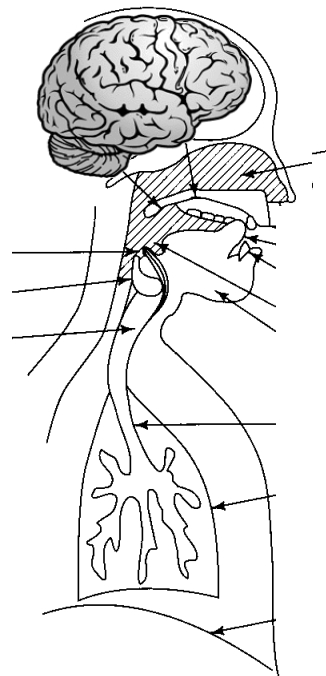
$$w^* = \arg \max_w P(w|a)$$

$$= \arg \max_w P(a|w)P(w)/P(a)$$

$$\propto \arg \max_w P(a|w)P(w)$$

Acoustic model: Distributions
over acoustic waves given a
sentence

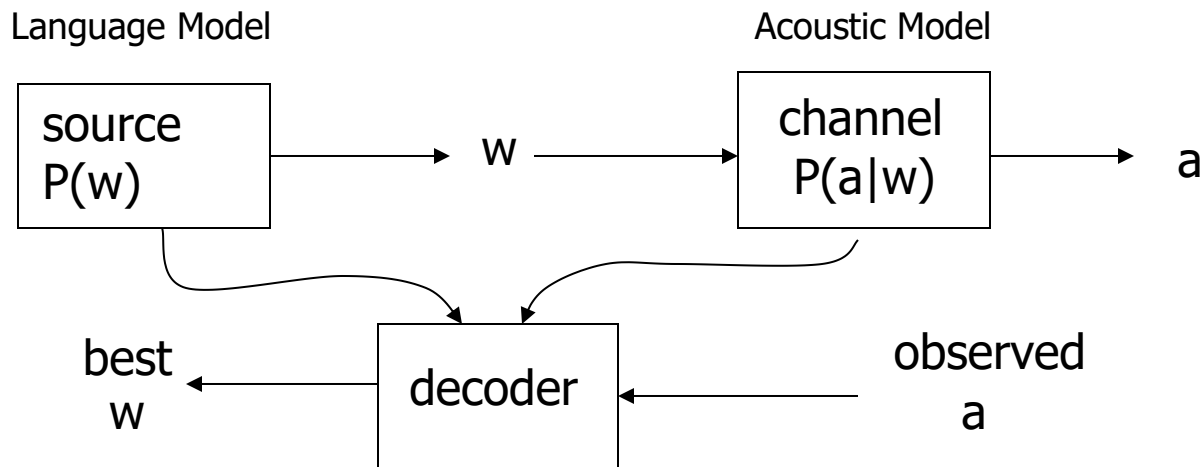
Language model:
Distributions over sequences
of words (sentences)



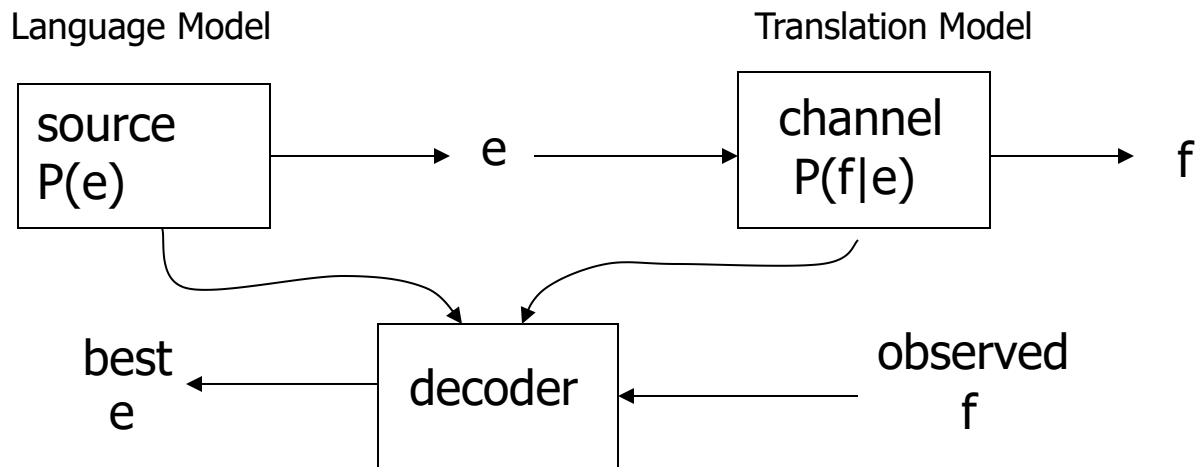
Acoustically Scored Hypotheses

the station signs are in deep in english	-14732
the stations signs are in deep in english	-14735
the station signs are in deep into english	-14739
the station 's signs are in deep in english	-14740
the station signs are in deep in the english	-14741
the station signs are indeed in english	-14757
the station 's signs are indeed in english	-14760
the station signs are indians in english	-14790
the station signs are indian in english	-14799
the stations signs are indians in english	-14807
the stations signs are indians and english	-14815

ASR System Components



MT System Components



$$\operatorname{argmax}_e P(e|f) = \operatorname{argmax}_e P(f|e)P(e)$$

Probabilistic Language Models: Other Applications

- Why assign a probability to a sentence?
 - Machine Translation:
 - $P(\text{high winds tonite}) > P(\text{large winds tonite})$
 - Speech Recognition
 - $P(\text{I saw a van}) \gg P(\text{eyes awe of an})$
 - Spell Correction
 - The office is about fifteen **minuets** from my house
 - $P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$
 - + Summarization, question-answering, etc., etc.!!

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Evaluation
- Neural Language Modeling

Probabilistic Language Modeling

- Goal: compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, w_4, w_5 \dots w_n)$$

- Related task: probability of an upcoming word:

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- A model that computes either of these:

$P(W)$ or $P(w_n | w_1, w_2 \dots w_{n-1})$ is called a **language model**.

How to compute $P(W)$

- How to compute this joint probability:
 - $P(\text{its, water, is, so, transparent, that})$

$P(\text{"its water is so transparent"}) =$

$$\begin{aligned} &P(\text{its}) \times P(\text{water}|\text{its}) \times P(\text{is}|\text{its water}) \\ &\quad \times P(\text{so}|\text{its water is}) \times P(\text{transparent}|\text{its water is so}) \end{aligned}$$

How to estimate these probabilities

- Could we just count and divide?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{\textit{Count}(\text{its water is so transparent that the})}{\textit{Count}(\text{its water is so transparent that})}$$

- No! Too many possible sentences!
- We'll never see enough data for estimating these

Markov Assumption



Andrei Markov

- Simplifying assumption:

$P(\text{the } | \text{its water is so transparent that}) \square P(\text{the } | \text{that})$

- Or maybe

$P(\text{the } | \text{its water is so transparent that}) \square P(\text{the } | \text{transparent that})$

Markov Assumption

$$P(w_1 w_2 \cdots w_n) \approx \prod_i P(w_i \mid w_{i-k} \cdots w_{i-1})$$

- In other words, we approximate each component in the product

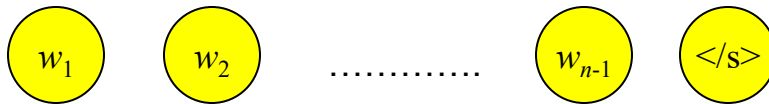
$$P(w_i \mid w_1 w_2 \cdots w_{i-1}) \approx P(w_i \mid w_{i-k} \cdots w_{i-1})$$

Simplest Case: Unigram Models

- Simplest case: unigrams

$$P(w_1 w_2 \dots w_n) \approx \prod_i P(w_i)$$

- Generative process: pick a word, pick a word, ... until you pick </s>
- Graphical model:



- Examples:

- fifth, an, of, futures, the, an, incorporated, a, a, the, inflation, most, dollars, quarter, in, is, mass
- thrift, did, eighty, said, hard, 'm, july, bullish
- that, or, limited, the

- Big problem with unigrams: $P(\text{the the the the}) \gg P(\text{I like ice cream})!$

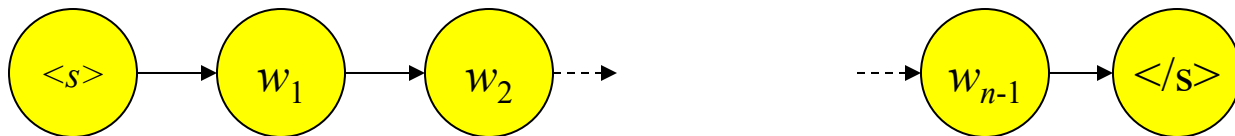
Bigram Models

- Conditioned on previous single word

$$P(w_i \mid w_1 w_2 \dots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

- **Generative process:** pick <s>, pick a word conditioned on previous one, repeat until to pick </s>

- **Graphical model:**



- **Examples:**

- texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen
- outside, new, car, parking, lot, of, the, agreement, reached
- this, would, be, a, record, november

N-Gram Models

- We can extend to trigrams, 4-grams, 5-grams
- N-gram models are (weighted) regular languages
 - Many linguistic arguments that language isn't regular.
 - Long-distance effects: “The computer which I had just put into the machine room on the fifth floor ____.”
 - Recursive structure
 - We often get away with n-gram models

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Evaluation
- Neural Language Modeling

Evaluation: How good is our model?

- Does our language model prefer good sentences to bad ones?
 - Assign higher probability to “real” or “frequently observed” sentences
 - Than “ungrammatical” or “rarely observed” sentences?
- We train parameters of our model on a **training set**.
- We test the model’s performance on data we haven’t seen.
 - A **test set** is an unseen dataset that is different from our training set, totally unused.
 - An **evaluation metric** tells us how well our model does on the test set.

Extrinsic evaluation of N-gram models

- Best evaluation for comparing models A and B
 - Put each model in a task
 - spelling corrector, speech recognizer, MT system
 - Run the task, get an accuracy for A and for B
 - How many misspelled words corrected properly
 - How many words translated correctly
 - Compare accuracy for A and B

Difficulty of extrinsic (in-vivo) evaluation of N-gram models

- Extrinsic evaluation
 - Time-consuming; requires building applications, new data
- So
 - Sometimes use **intrinsic** evaluation: **perplexity**
 - Bad approximation
 - unless the test data looks **just** like the training data
 - So **generally only useful in pilot experiments**
 - But is helpful to think about.

Intuition of Perplexity

- The Shannon Game:

- How well can we predict the next word?

I always order pizza with cheese and _____

The 33rd President of the US was _____

I saw a _____

- Unigrams are terrible at this game. (Why?)

- A better model of a text

- is one which assigns a higher probability to the word that actually occurs

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

....

fried rice 0.0001

....

and 1e-100

Perplexity

The best language model is one that best predicts an unseen test set

- Gives the highest $P(\text{sentence})$

Perplexity is the inverse probability of the test set, normalized by the number of words:

Chain rule:

For bigrams:

$$\begin{aligned} PP(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

$$PP(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

Minimizing perplexity is the same as maximizing probability

The Shannon Game intuition for perplexity

- From Josh Goodman
- How hard is the task of recognizing digits '0,1,2,3,4,5,6,7,8,9'
 - Perplexity 10
- How hard is recognizing (30,000) names at Microsoft.
 - Perplexity = 30,000
- If a system has to recognize
 - Operator (1 in 4)
 - Sales (1 in 4)
 - Technical Support (1 in 4)
 - 30,000 names (1 in 120,000 each)
 - Perplexity is 53
- Perplexity is weighted equivalent branching factor

Perplexity as branching factor

- Let's suppose a sentence consisting of random digits
- What is the perplexity of this sentence according to a model that assign $P=1/10$ to each digit?

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

Another form of Perplexity

$$2^{-l} \text{ where } l = \frac{1}{M} \sum_{i=1}^m \log p(s_i)$$

- Lower is better!
- **Example:** $|\mathcal{V}| = N$ and $q(w|\dots) = \frac{1}{N}$
 - uniform model \rightarrow perplexity is N
- **Interpretation:** effective vocabulary size (accounting for statistical regularities)
- **Typical values for newspaper text:**
 - Uniform: 20,000; Unigram: 1000s, Bigram: 700-1000, Trigram: 100-200
- **Important note:**
 - Its easy to get bogus perplexities by having bogus probabilities that sum to more than one over their event spaces. Be careful!

Lower perplexity = better model

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Outline

- Motivation
- Task Definition
- N-Gram Probability Estimation
- Evaluation
- Neural Language Modeling

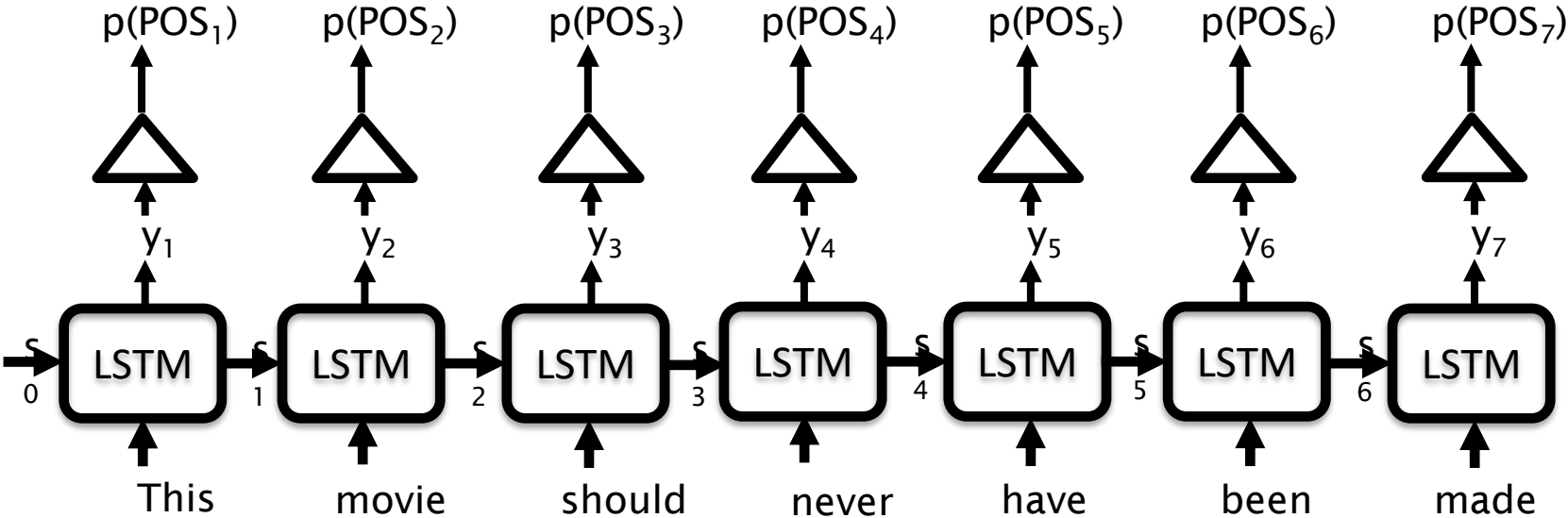
Neural Language Models

Outline

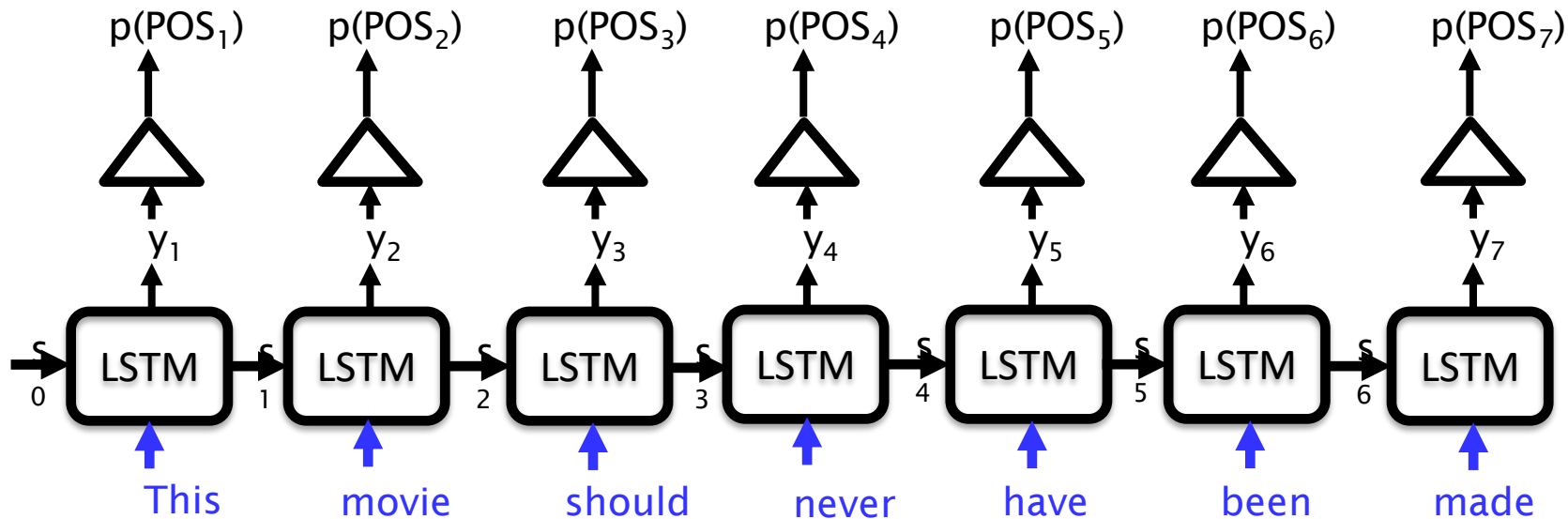
- Neural Language Models: LSTMs
- Neural Language Models: Transformers
- Seq2Seq Models with LSTMs
- Seq2Seq Models with Transformers

Sequence Decoder

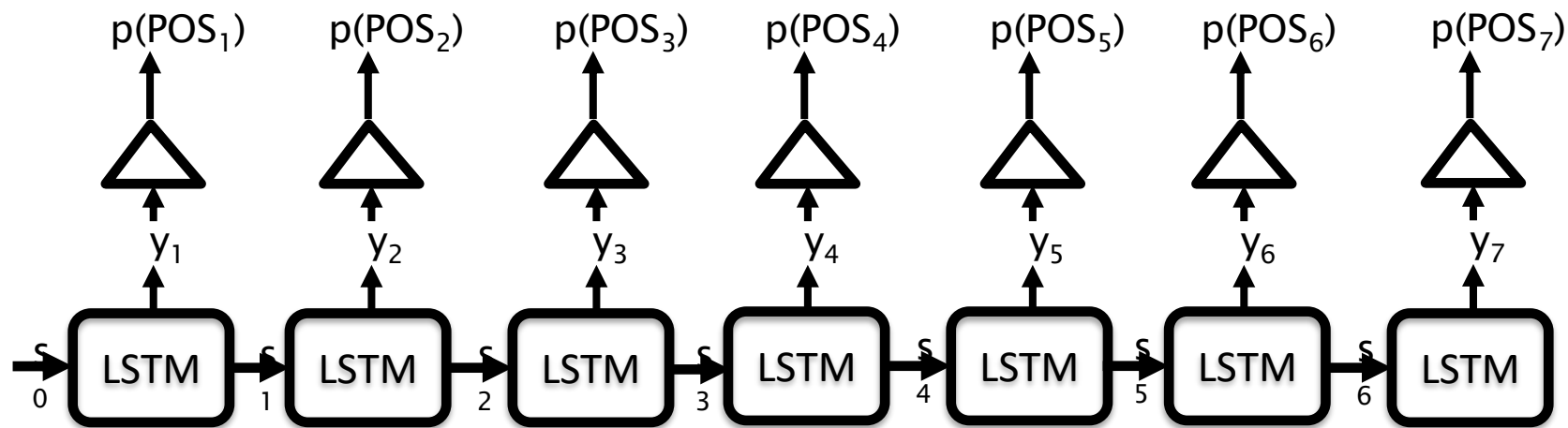
Neural Language Model



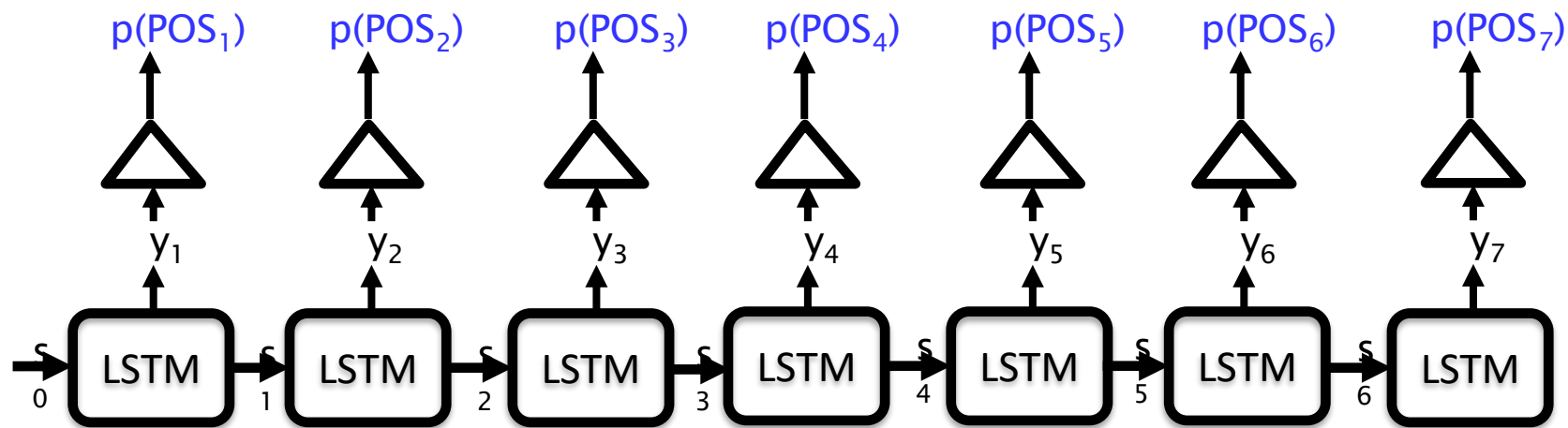
Neural Language Model



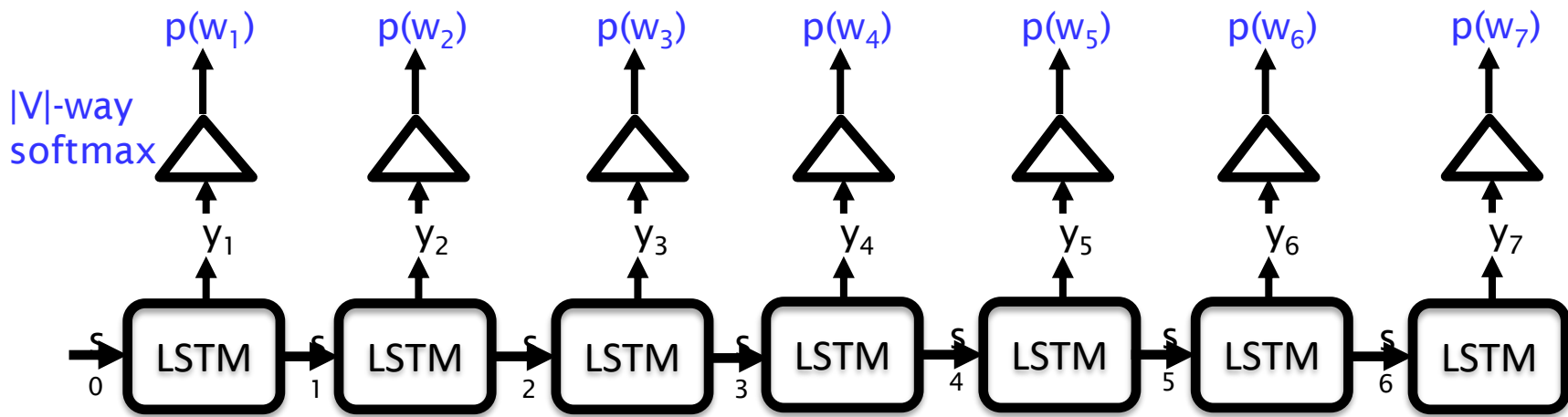
Neural Language Model



Neural Language Model

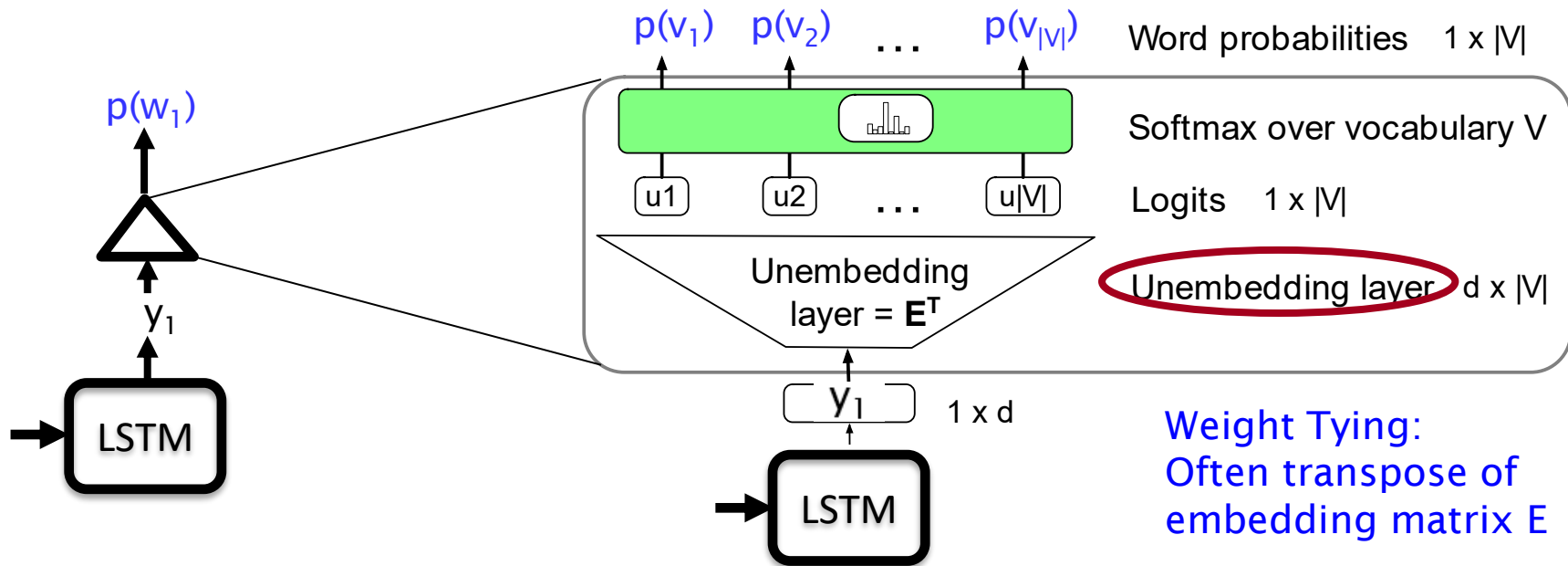


Neural Language Model

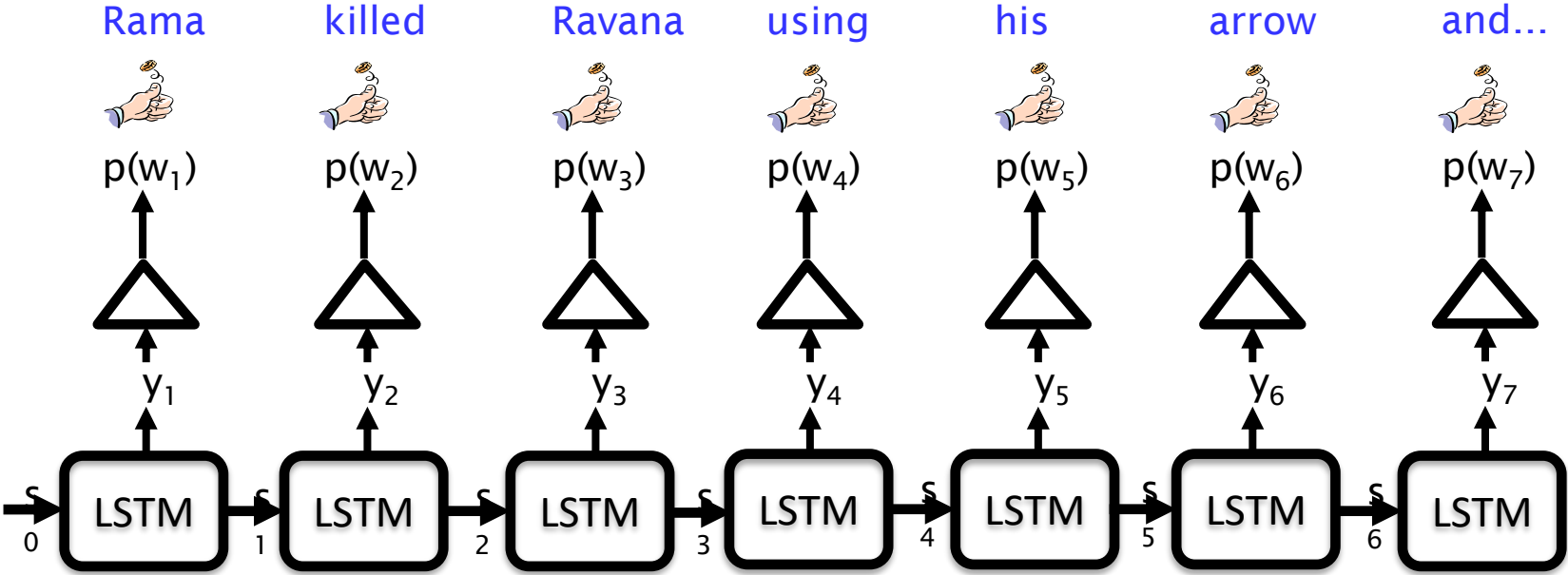


How do we get the actual sentence from a sequence of probability distributions?

Language Modeling Head

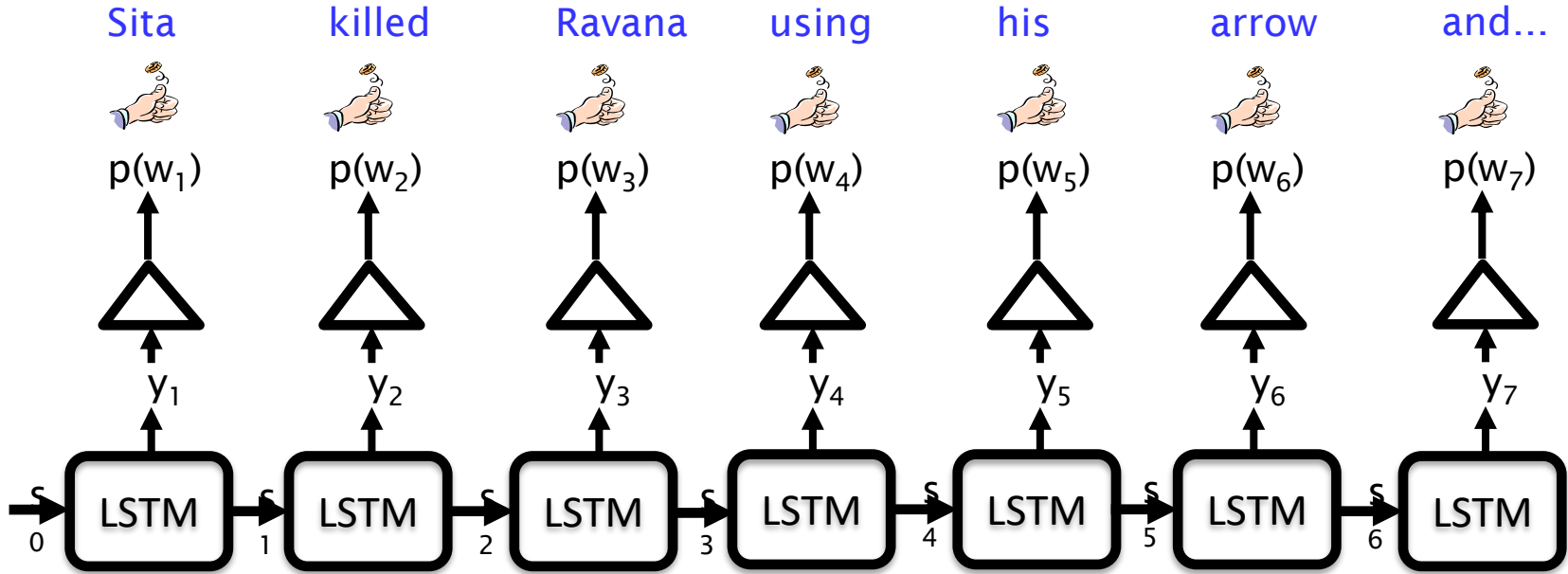


Neural Language Model



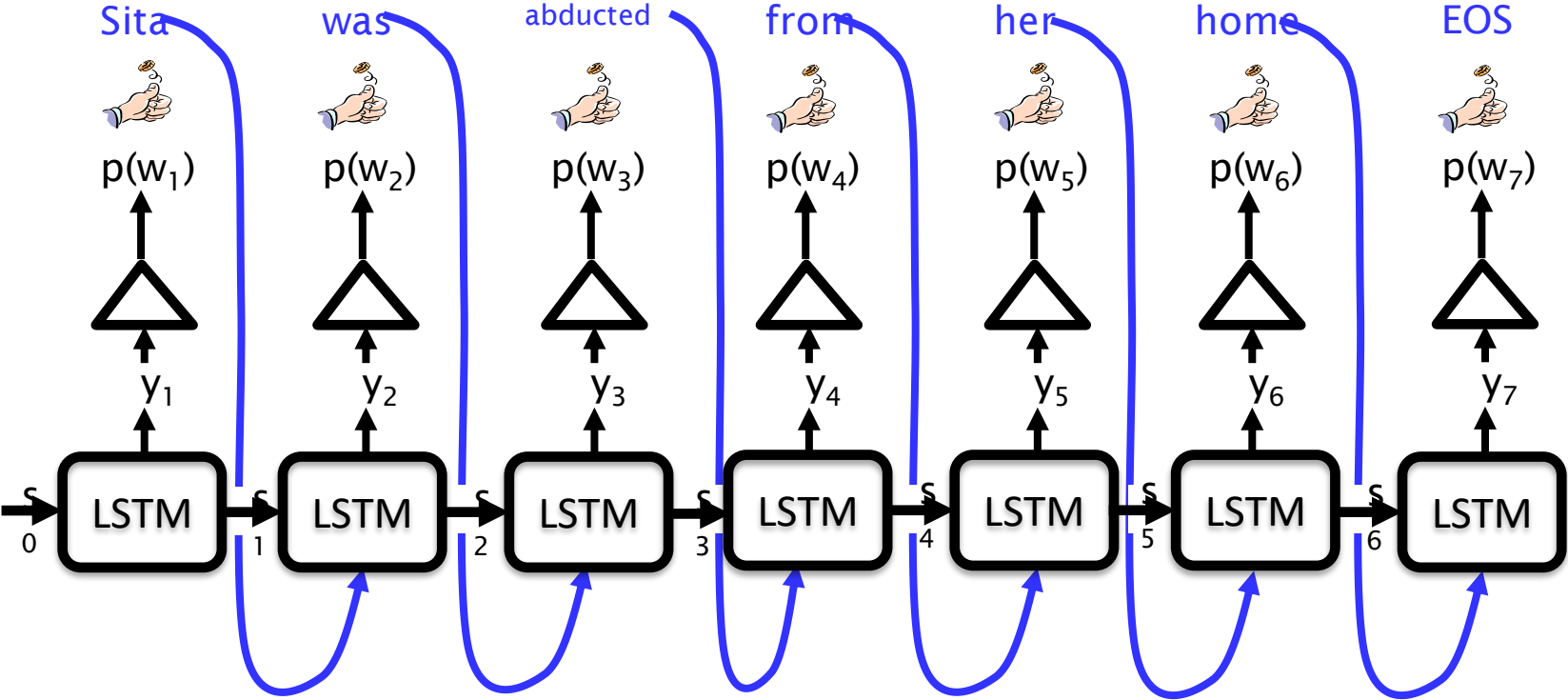
Can you think of a fundamental problem in this design?

Neural Language Model

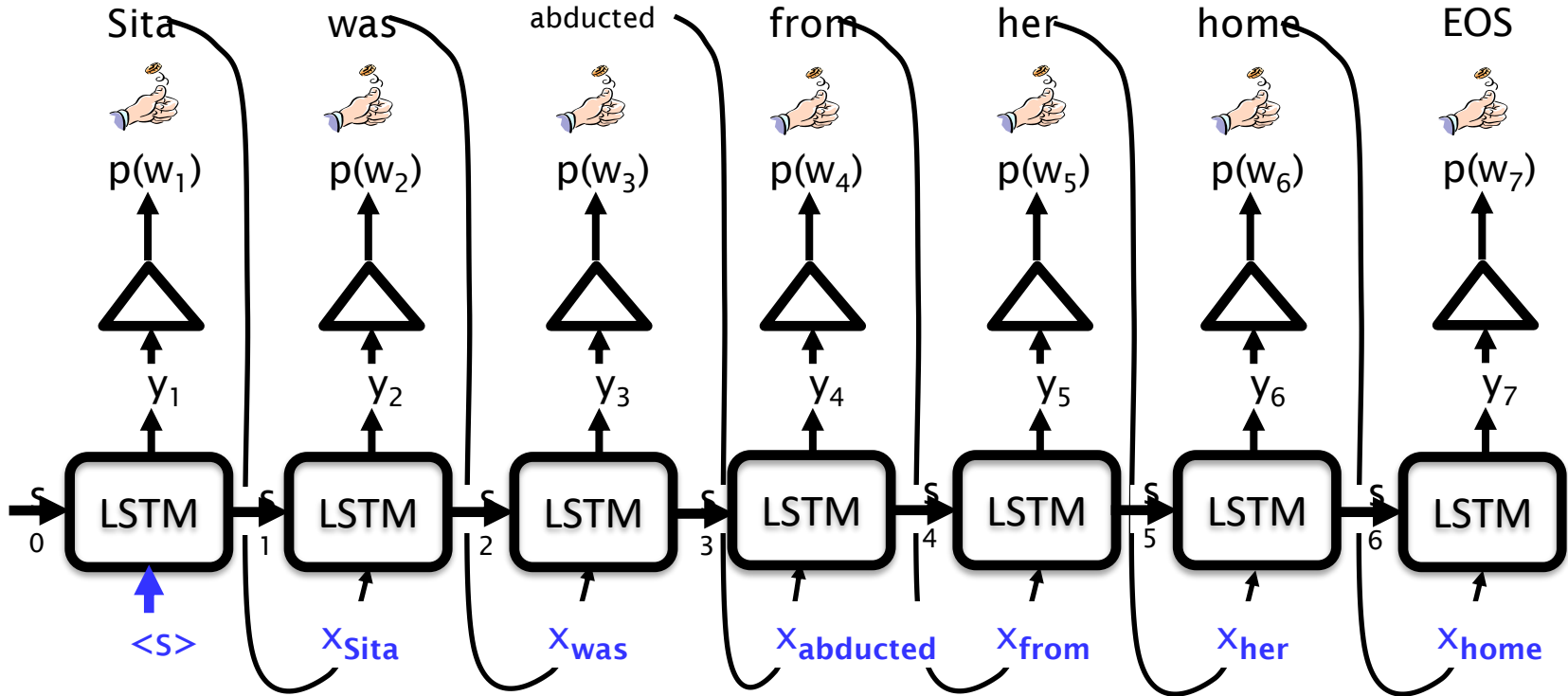


Can you think of a fundamental problem in this design?

Neural Language Model



Neural Language Model



Called "Auto-regressive models"

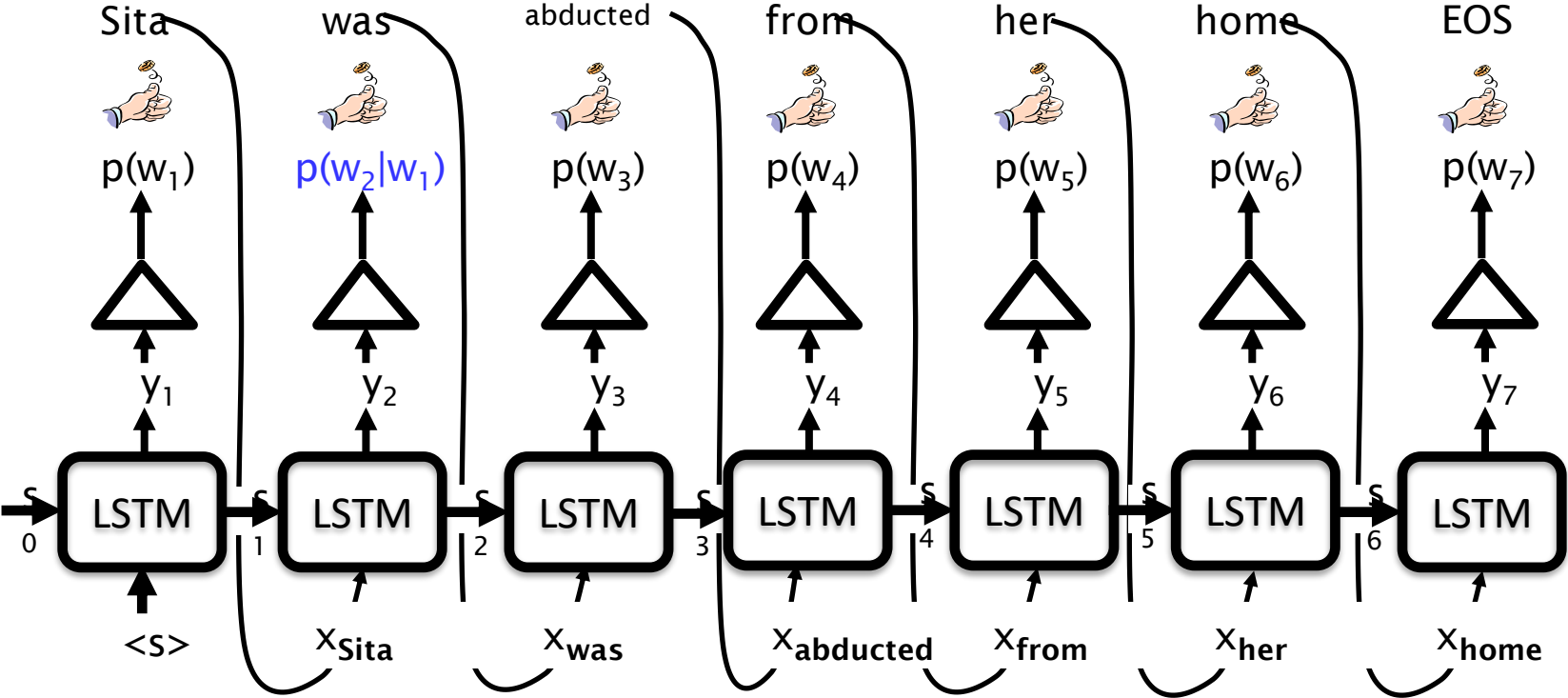
Neural Language Model

- 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).

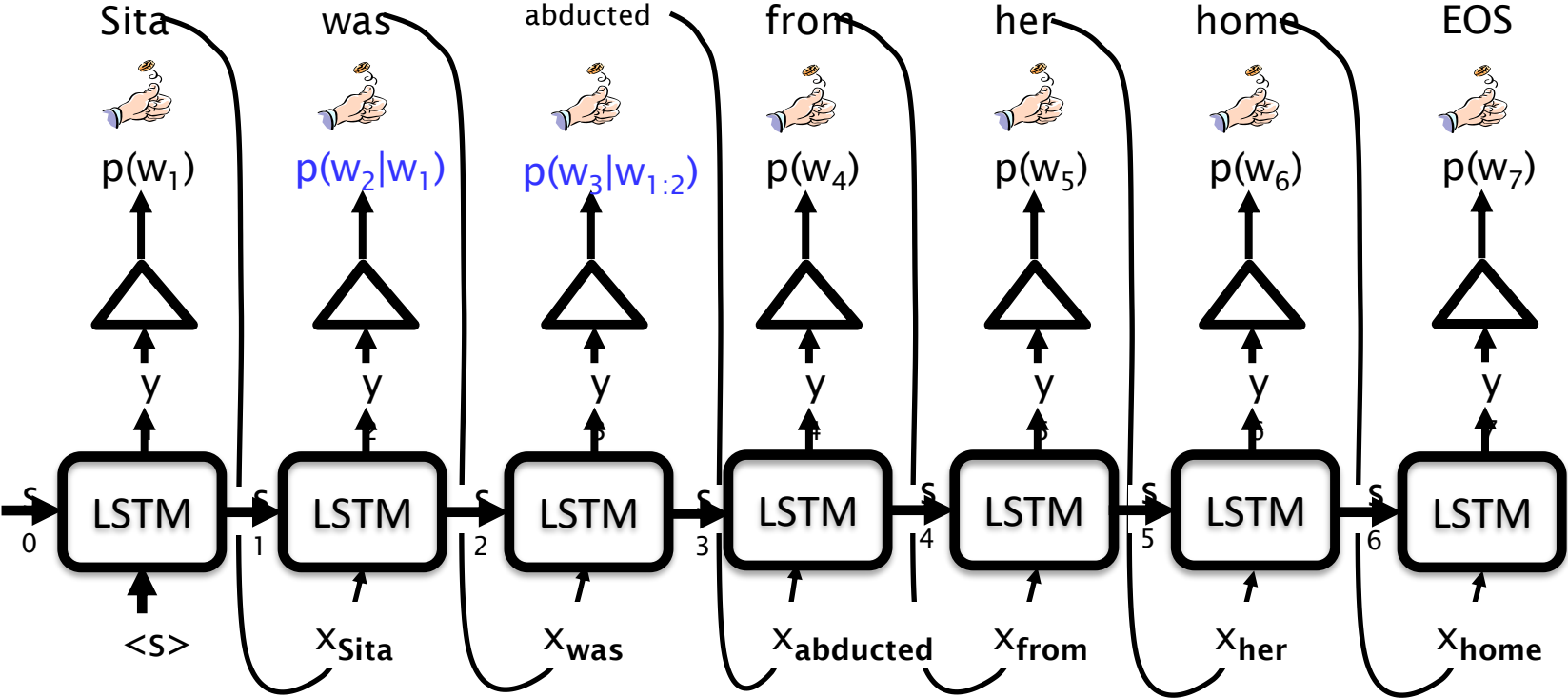
$$P(w_{1:n}) = P(w_1)P(w_2 | w_1)P(w_3 | w_{1:2})P(w_4 | w_{1:3}) \dots P(w_n | w_{1:n-1})$$

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

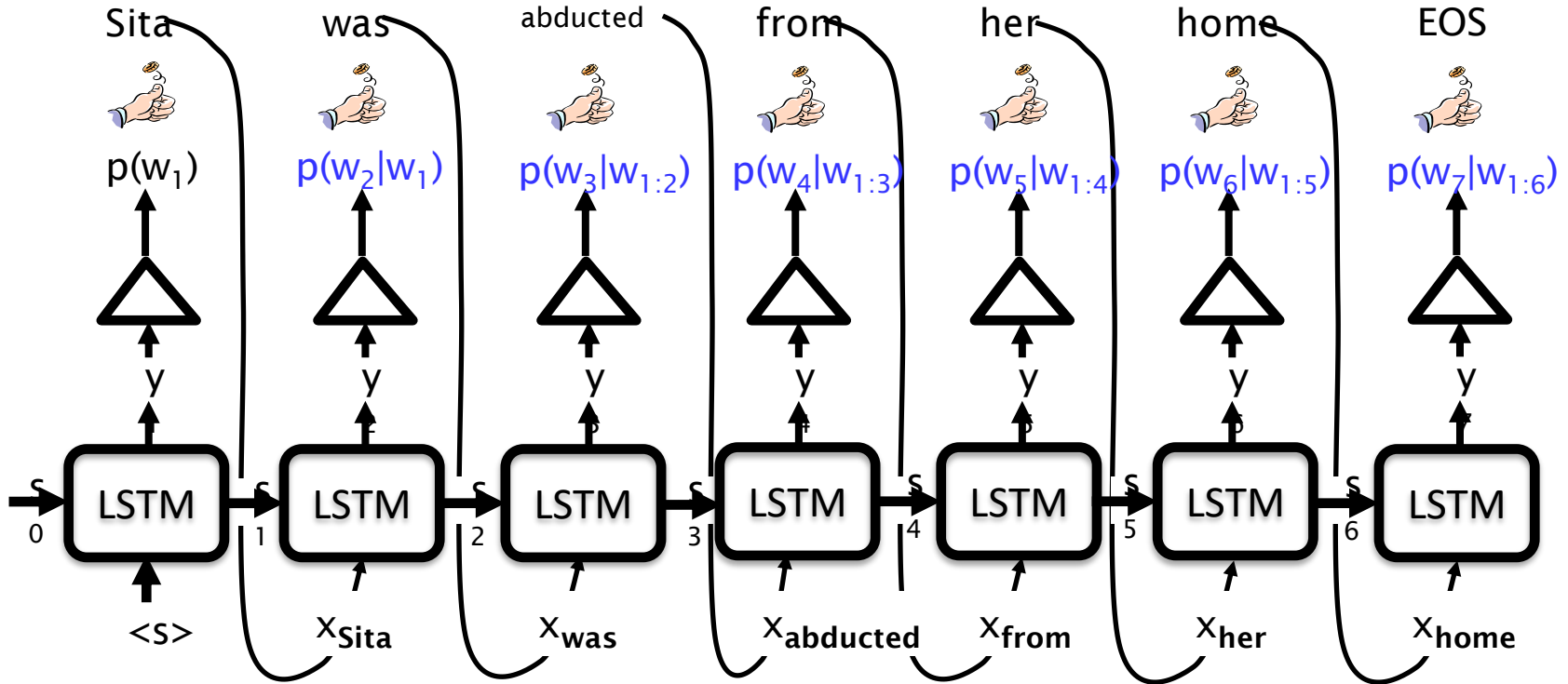
Neural Language Model



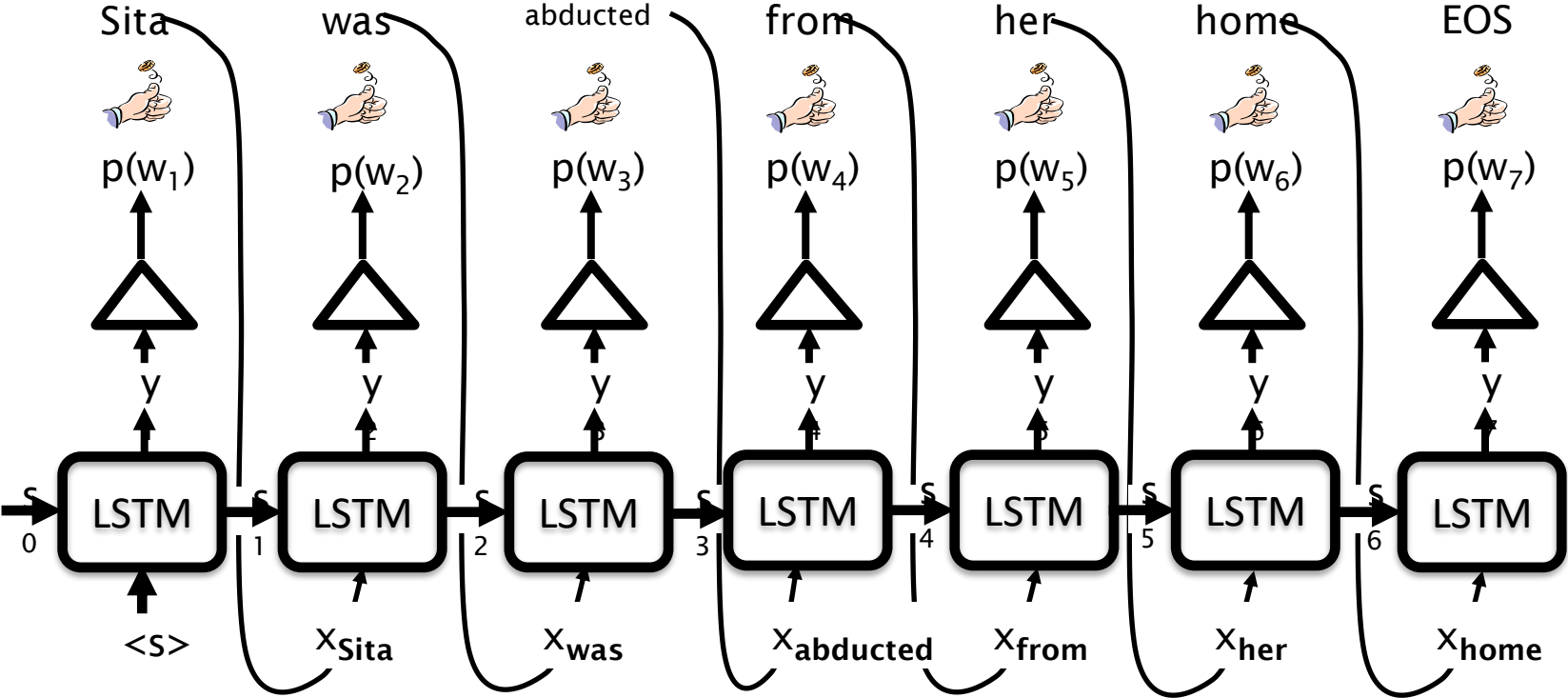
Neural Language Model



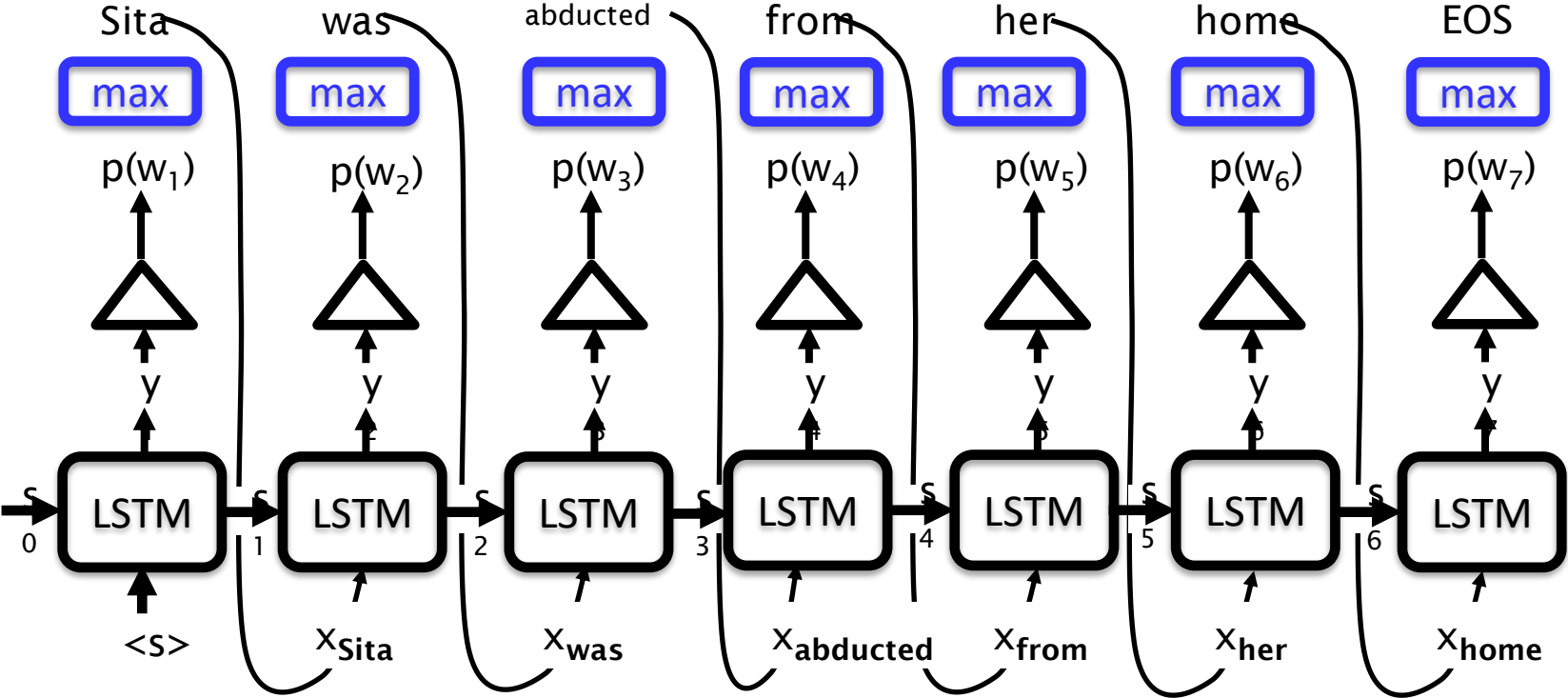
Neural Language Model



Neural Language Model: Inference Time



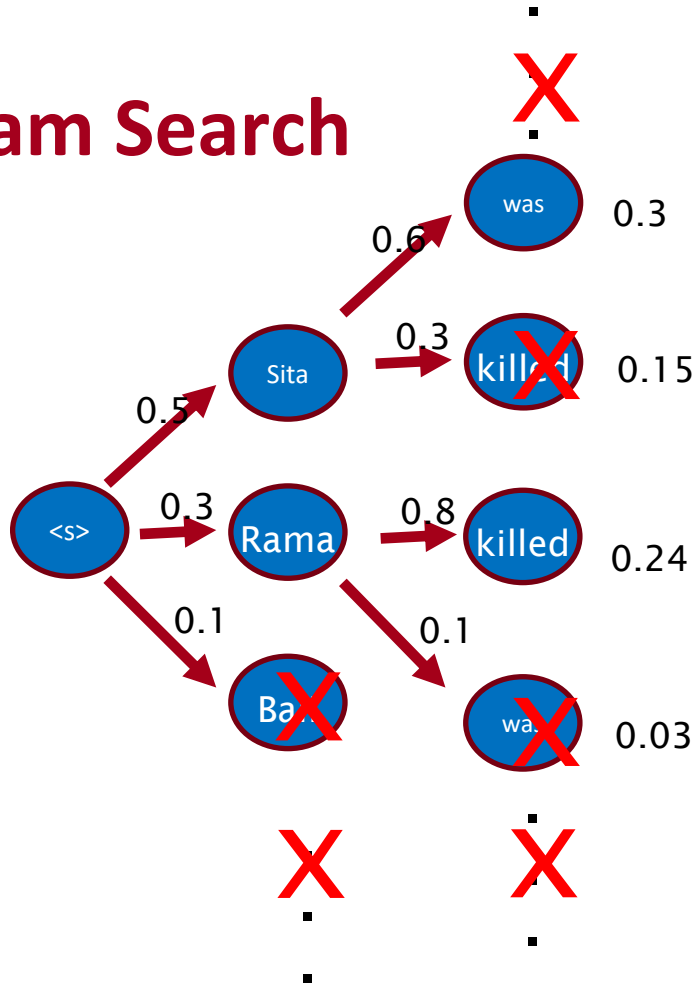
Neural Language Model: Inference Time



Neural Language Model: Greedy Decoding

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

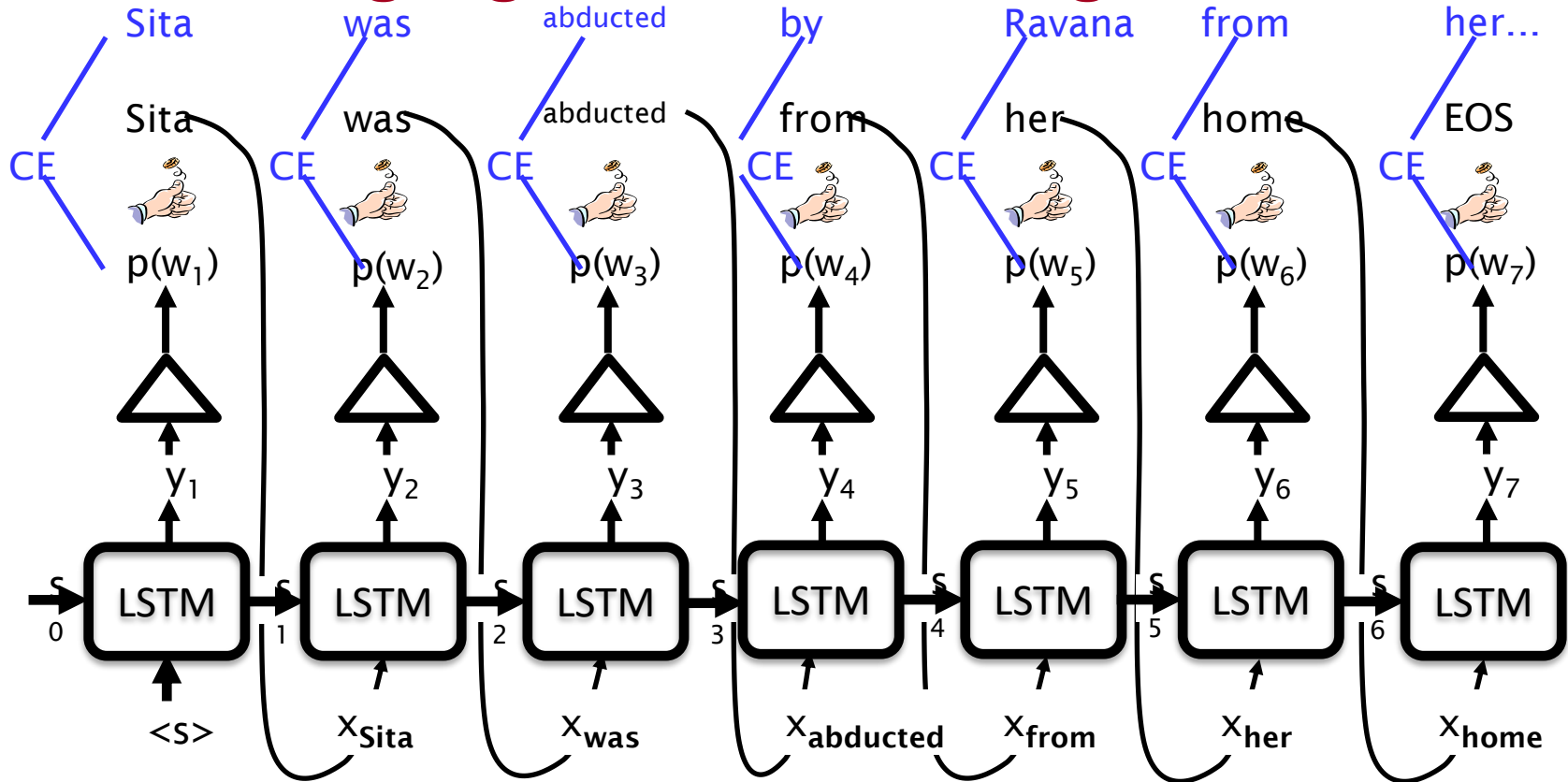


Instead of picking one greedy path, maintain multiple greedy paths

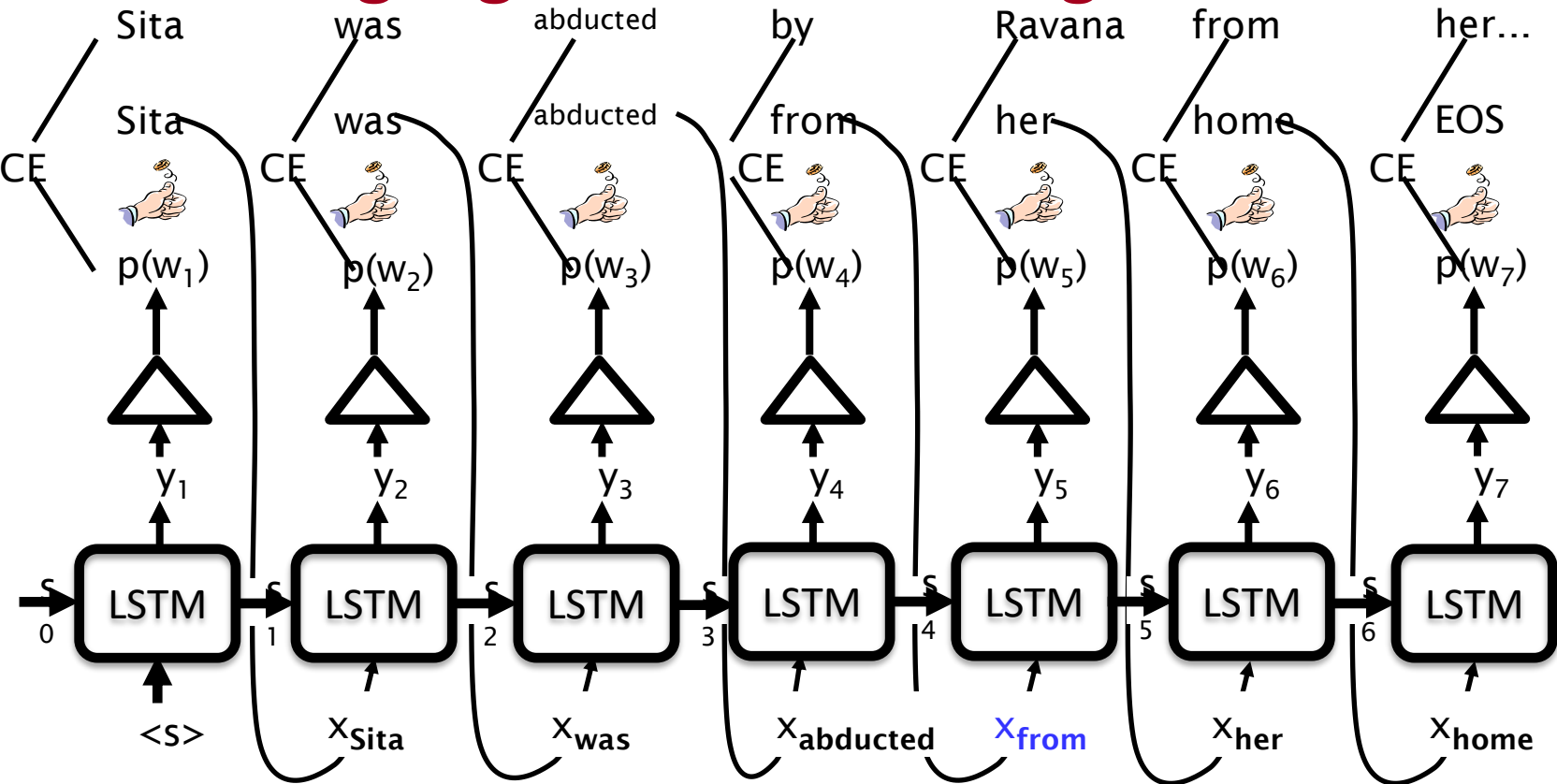
Upto a constant beam of b

Example for beam size = 2

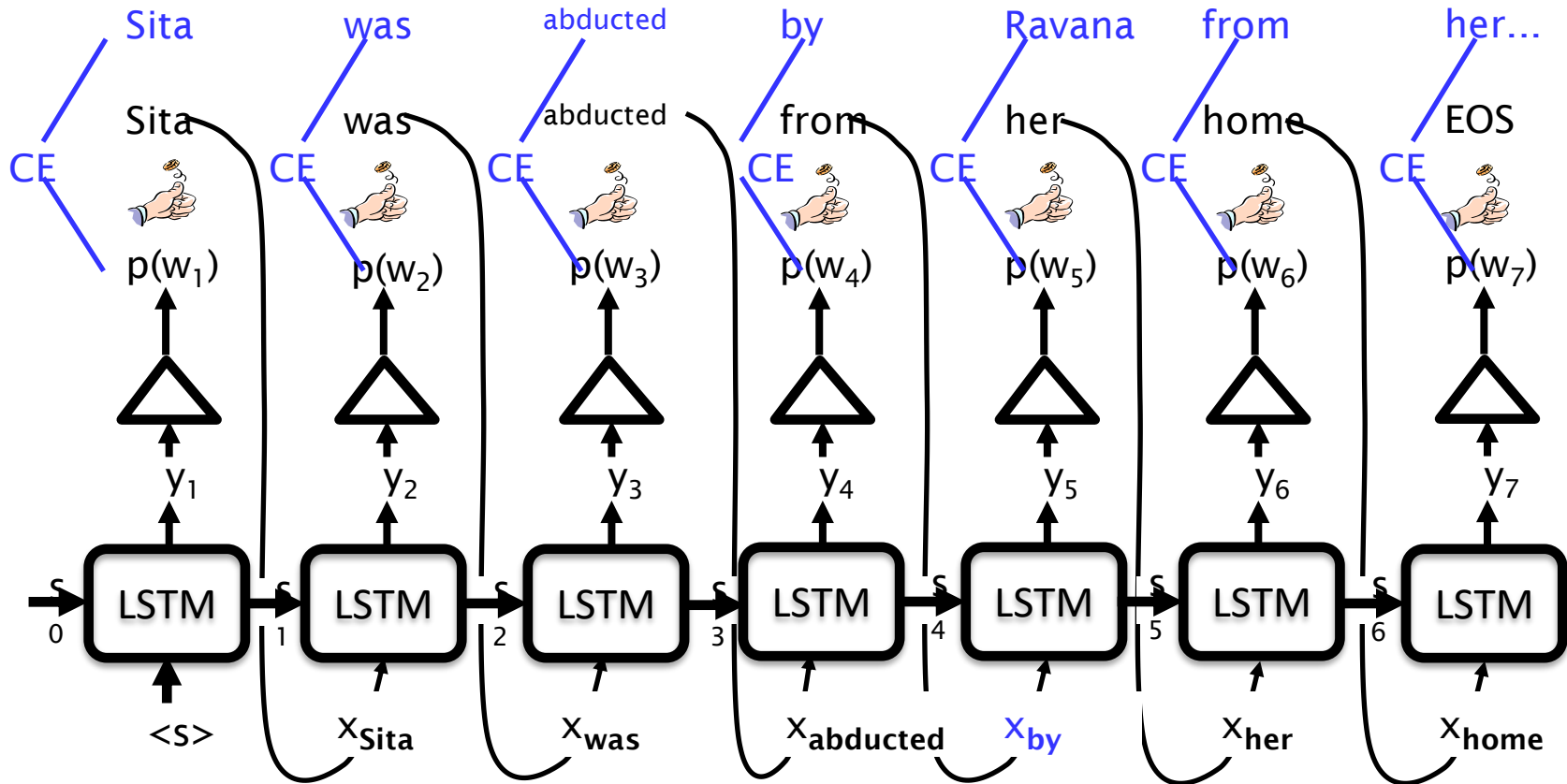
Neural Language Model: Training



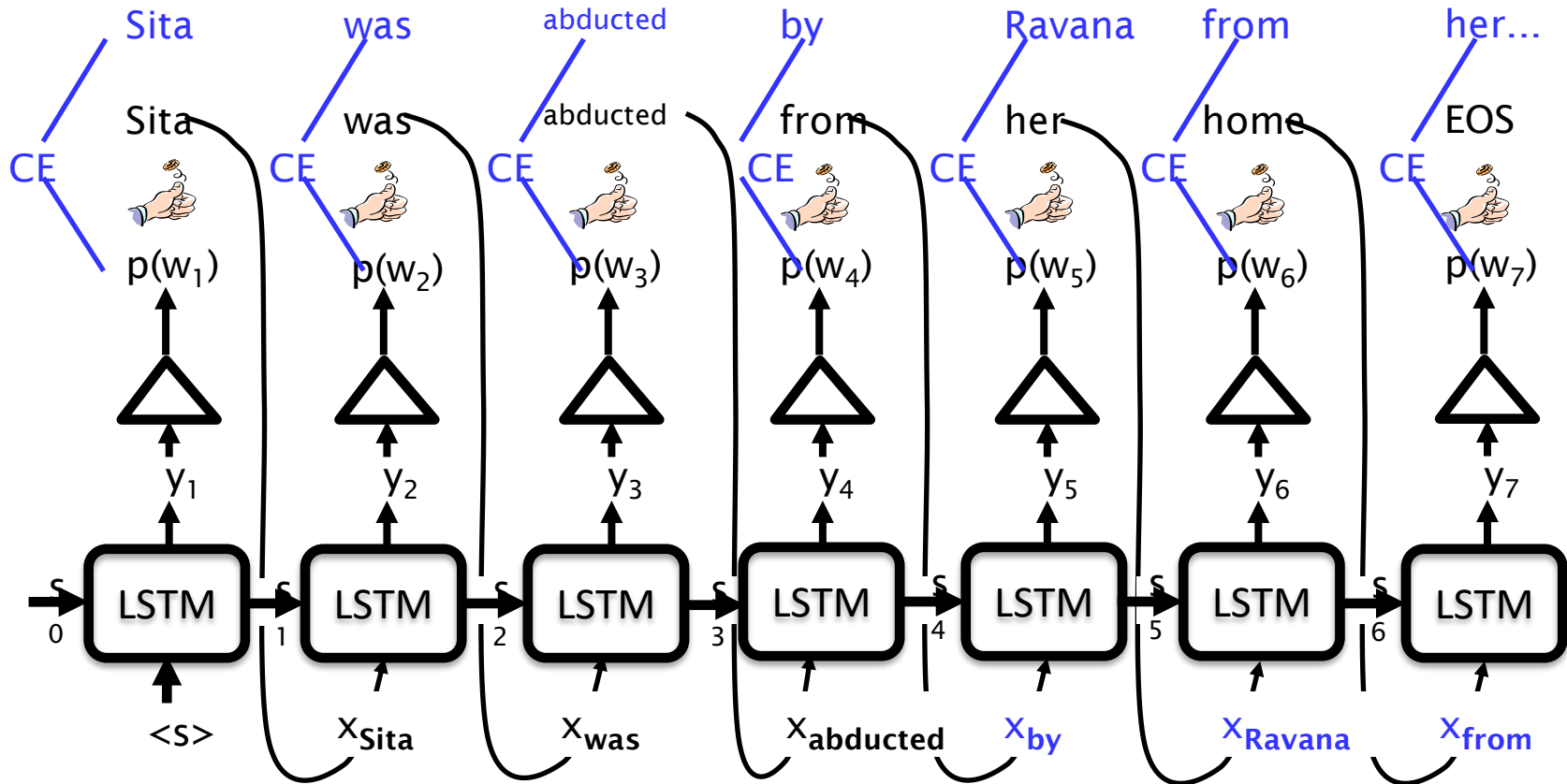
Neural Language Model: Training



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

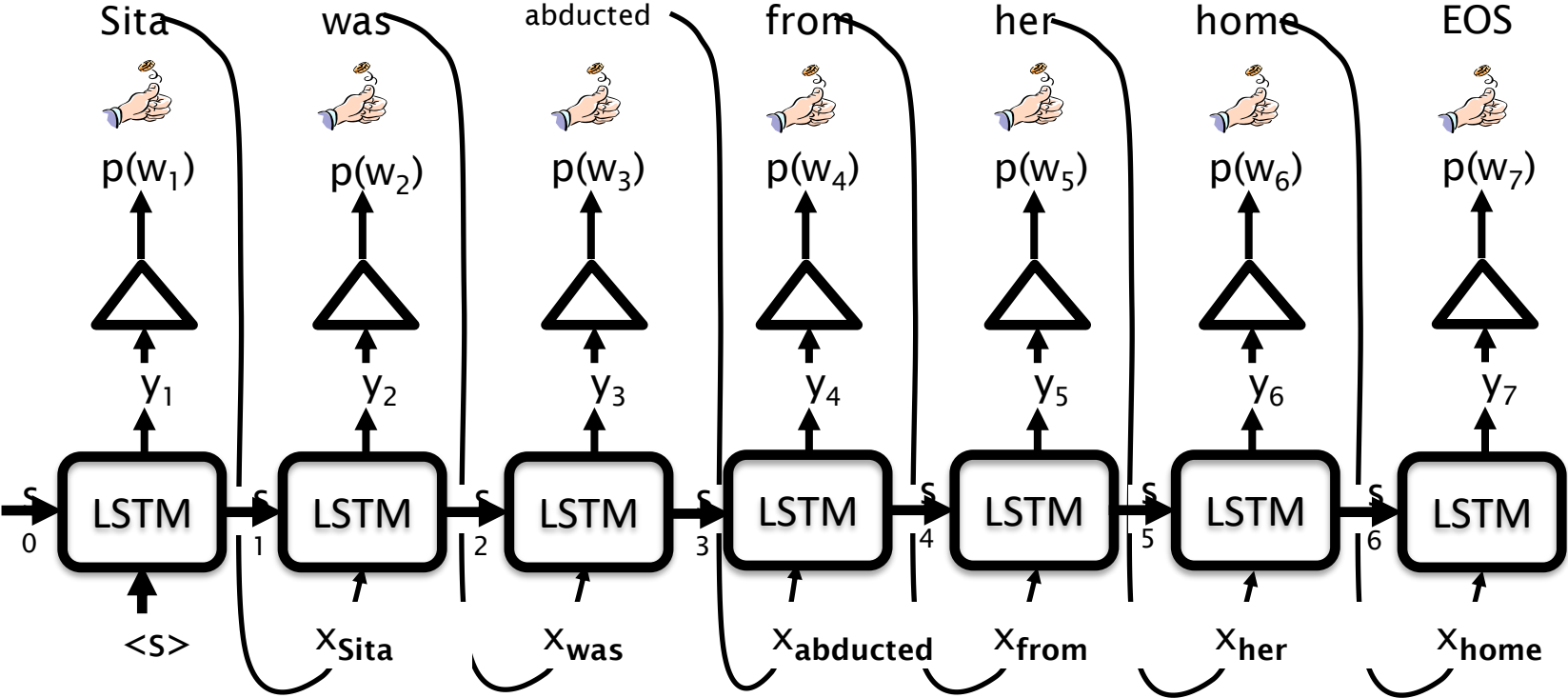
Solutions to Exposure Bias

- Scheduled Sampling (Bengio et al. 2015)
 - With some probability, decode a token and feed that as the next input, rather than the gold token.
 - Increase probability over the course of training

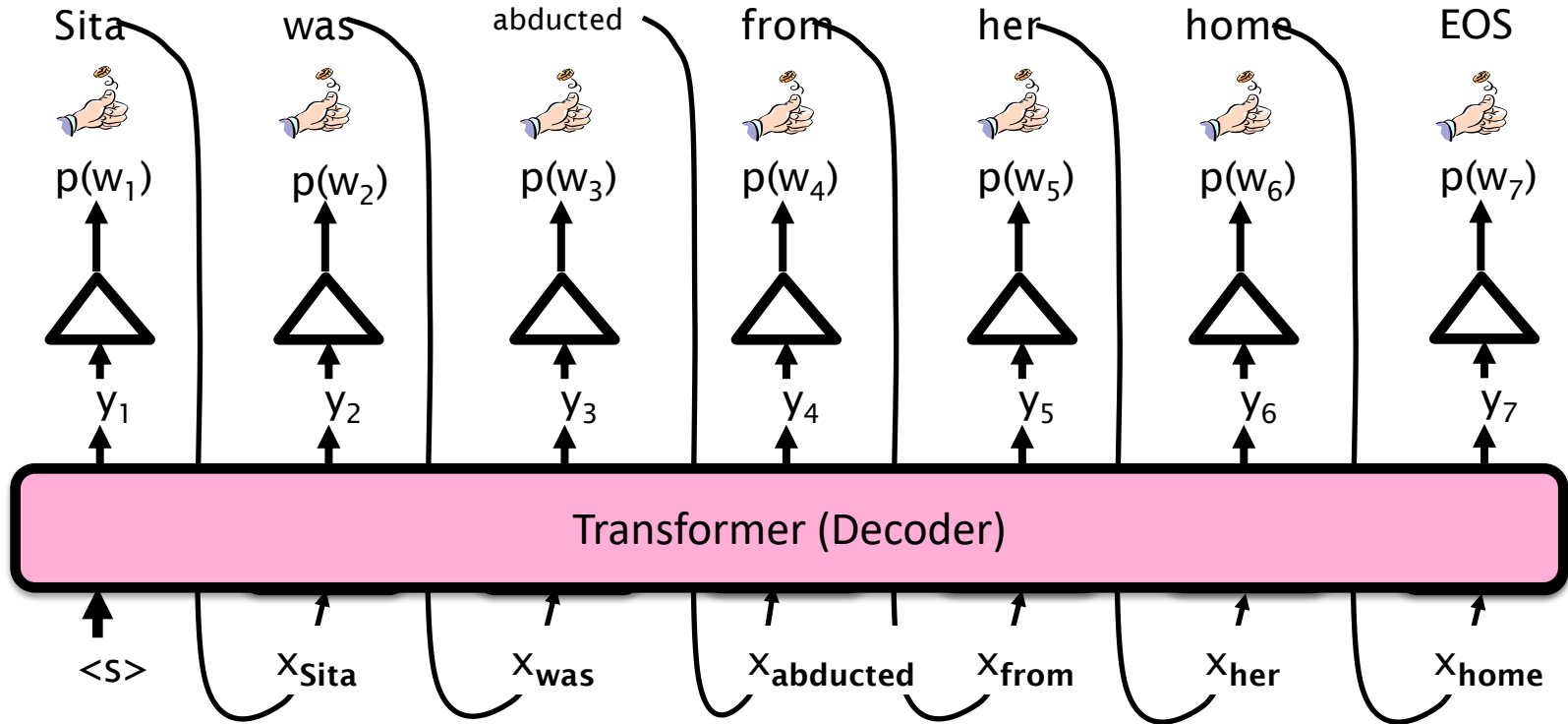
Outline

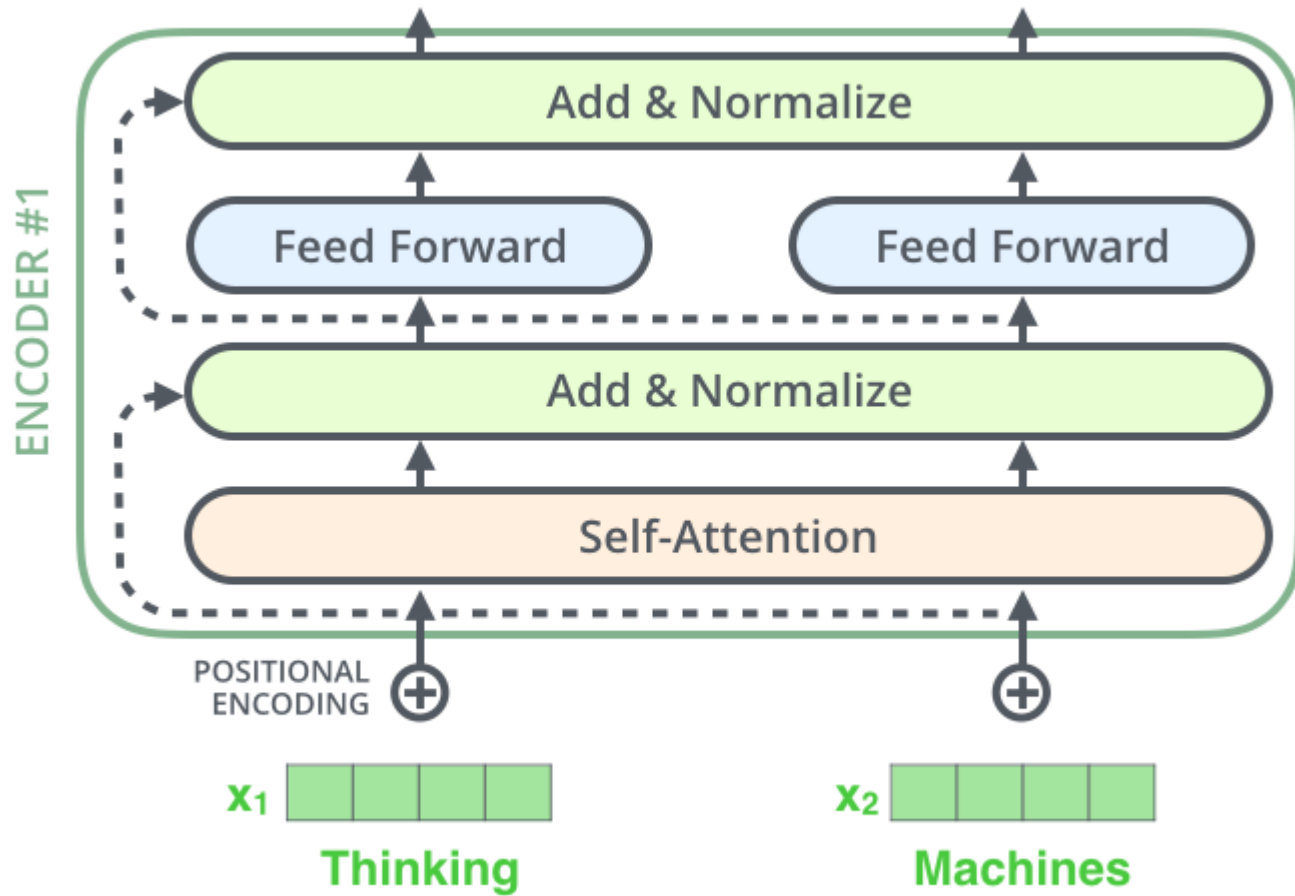
- Neural Language Models: LSTMs
- Neural Language Models: Transformers
- Seq2Seq Models with LSTMs
- Seq2Seq Models with Transformers

Neural Language Model



Transformer Language Model

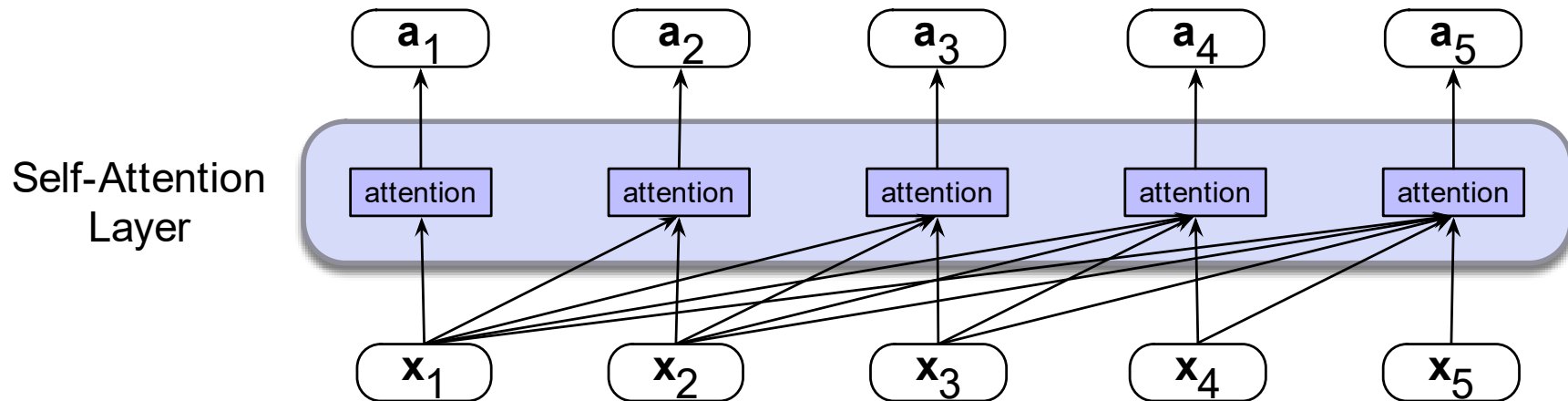




Challenge for Transformer Encoder → Decoder

- All pair attention is not meaningful. Why?
- Solution: all pair attention only for the past
 - Self-attention only on words generated upto now, not on whole sentence.

Attention is left-to-right

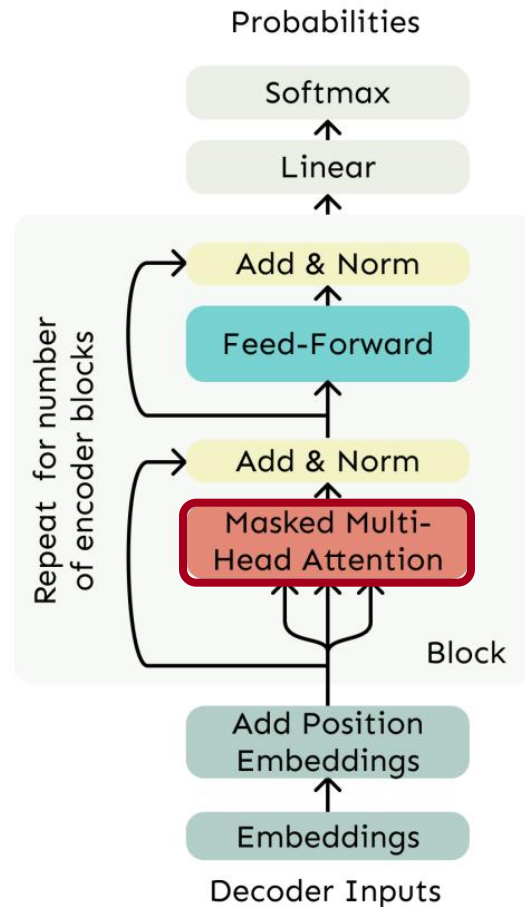


Transformer Decoder

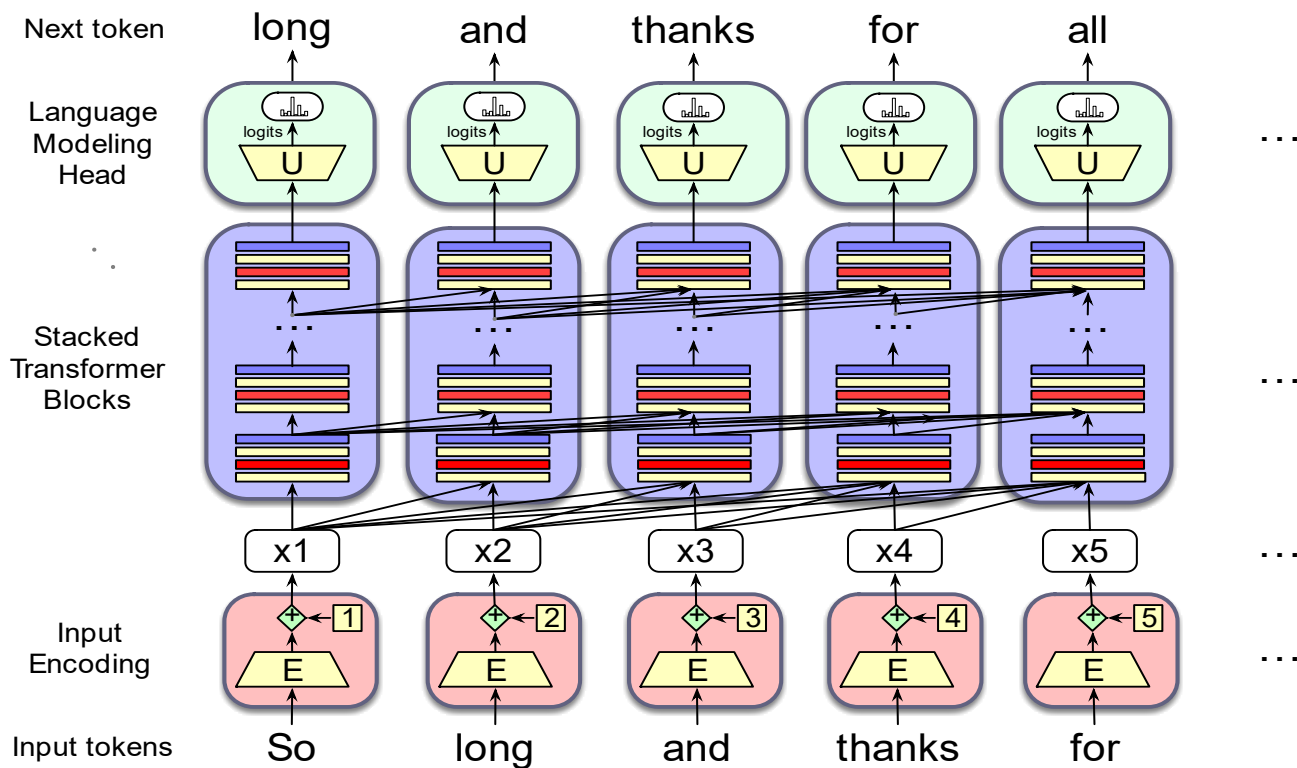
- Need to ensure we don't "look at the future" when predicting a sequence

$$\mathbf{A} = \text{softmax} \left(\text{mask} \left(\frac{\mathbf{QK}^T}{\sqrt{d_k}} \right) \right) \mathbf{V}$$

$q1 \cdot k1$	$-\infty$	$-\infty$	$-\infty$
$q2 \cdot k1$	$q2 \cdot k2$	$-\infty$	$-\infty$
$q3 \cdot k1$	$q3 \cdot k2$	$q3 \cdot k3$	$-\infty$
$q4 \cdot k1$	$q4 \cdot k2$	$q4 \cdot k3$	$q4 \cdot k4$



Transformer Language Model



Outline

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

Goal

- Generate text based on (varied) inputs
- Examples
 - Machine Translation: Language \rightarrow Language
 - Summarization: Language \rightarrow Language
 - Dialogue Systems: Language \rightarrow Language
 - Speech Recognition: Speech \rightarrow Language
 - Image Captioning: Image \rightarrow Language
 - Video Captioning: Video \rightarrow Language
 - Speech Recognition in Videos: Video+Speech \rightarrow Language

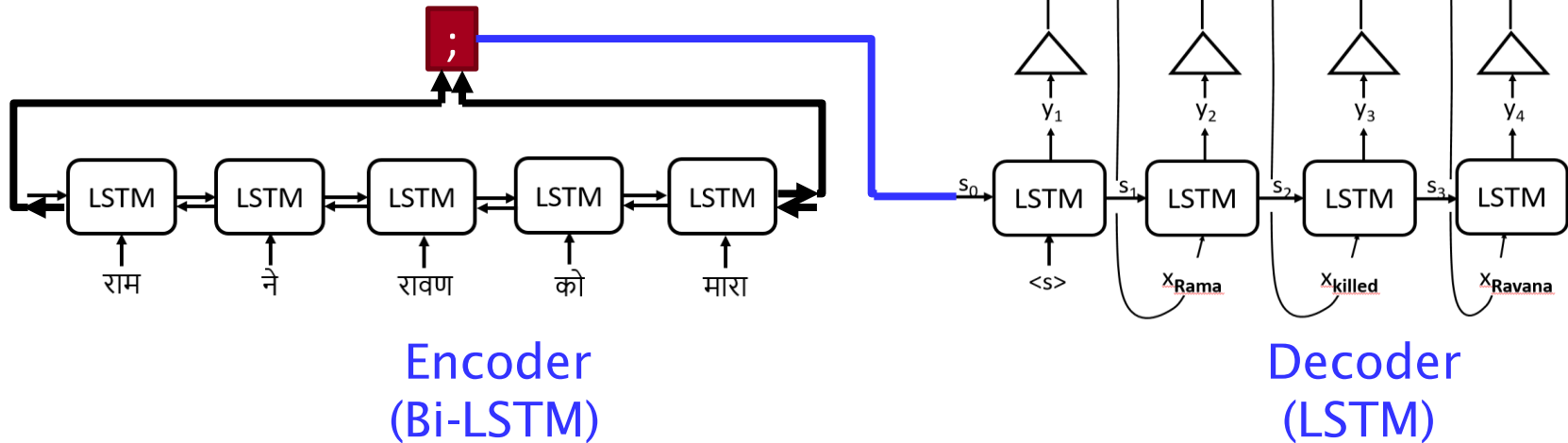
Goal

- 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

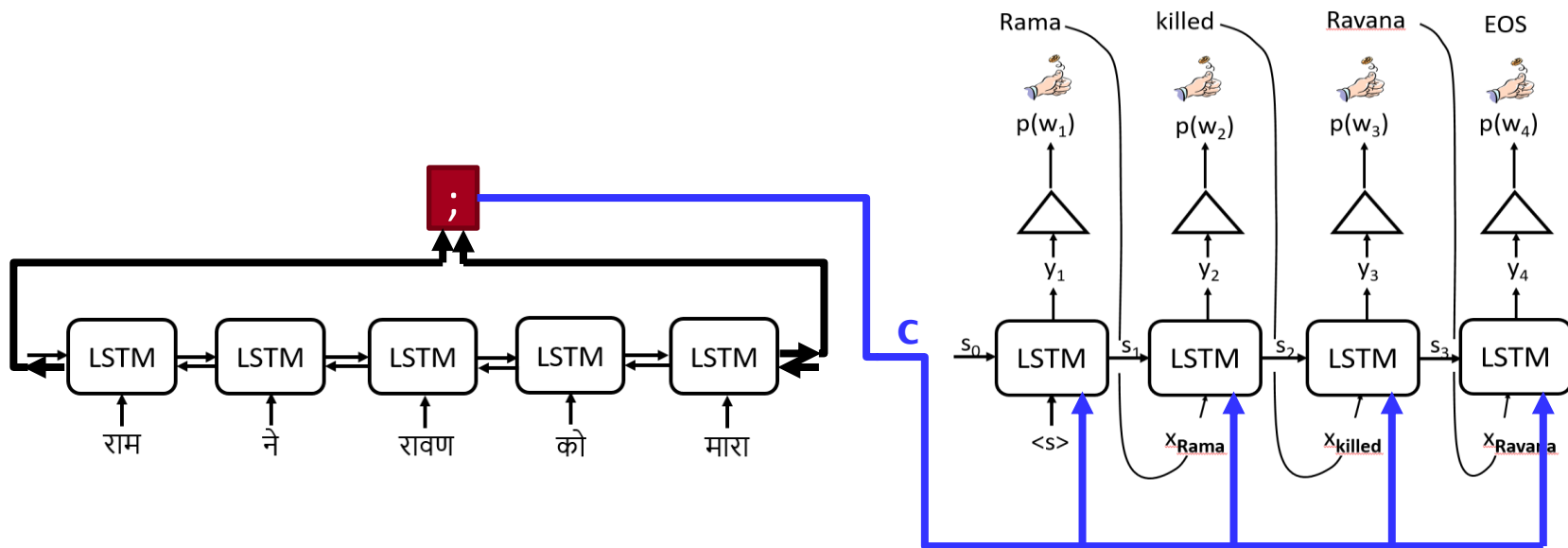
Idea 1: Encoder-Decoder

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

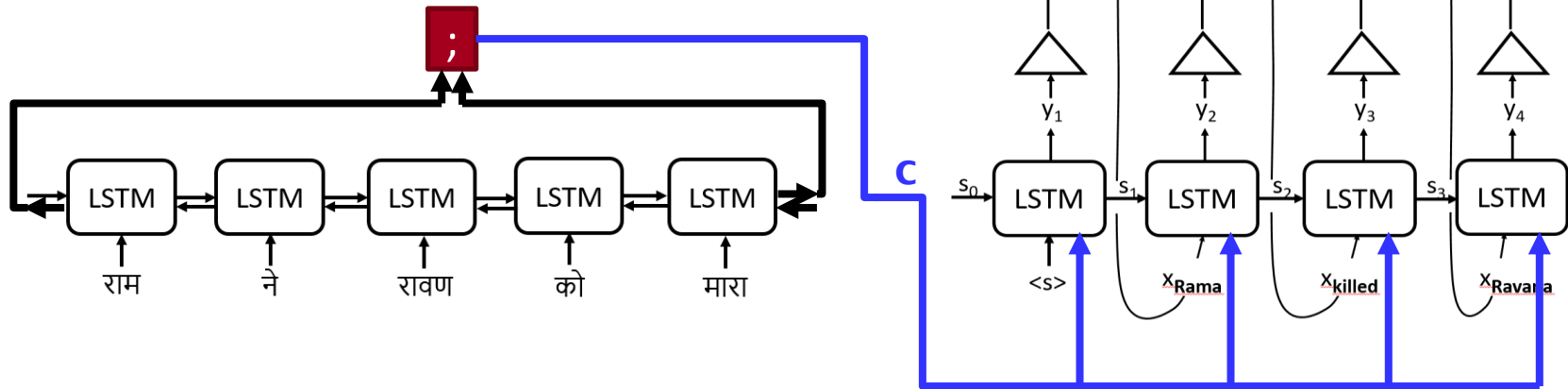


Idea 2: Encoder-Decoder

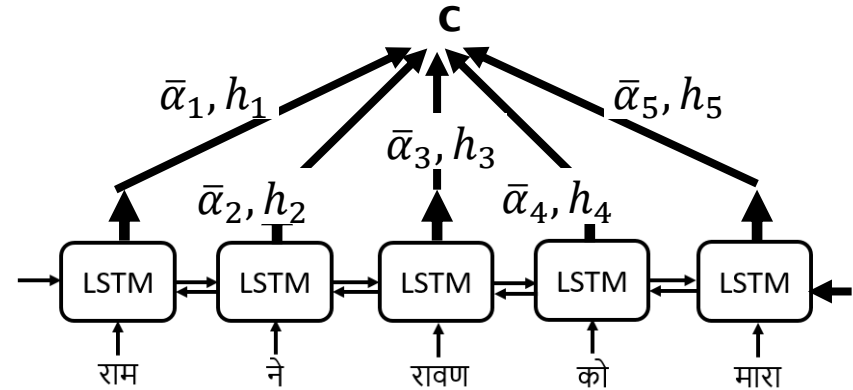
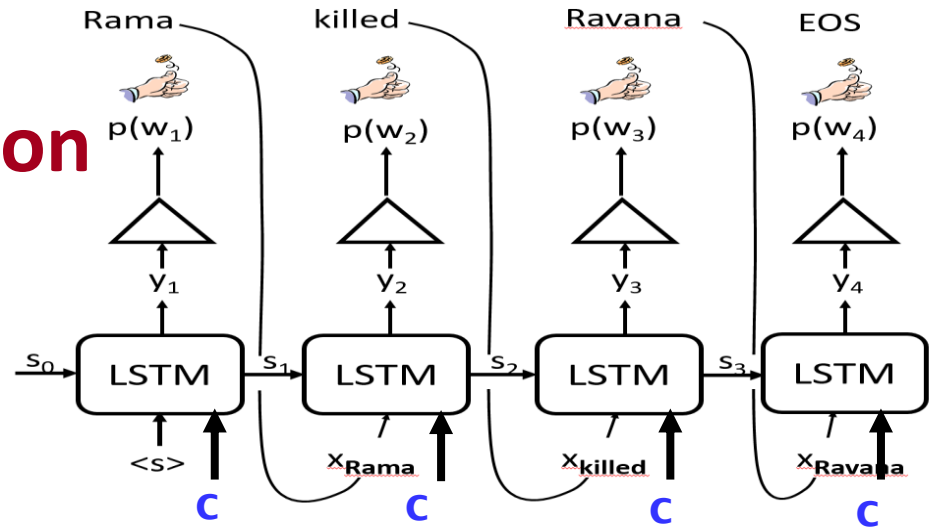
- Pass encoder output as input to *each* decoder unit
- Input at decoder = $\text{concat}(z, x_{\text{prev word}})$



Seq2Seq without Attention



Seq2Seq with Attention

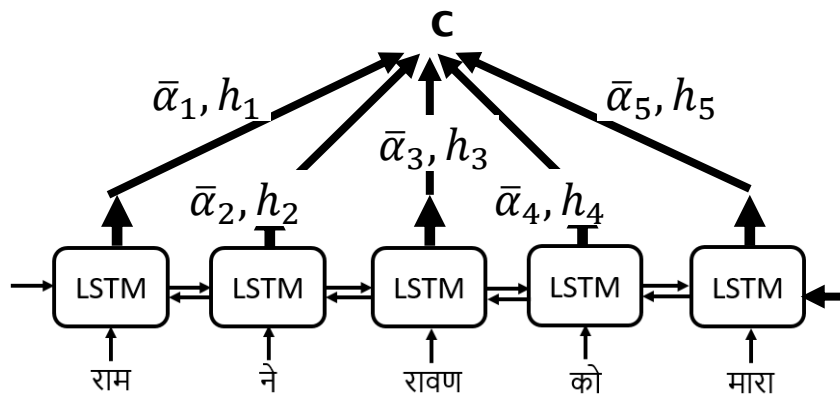


Multiple Encoded Vectors \rightarrow Single Summary

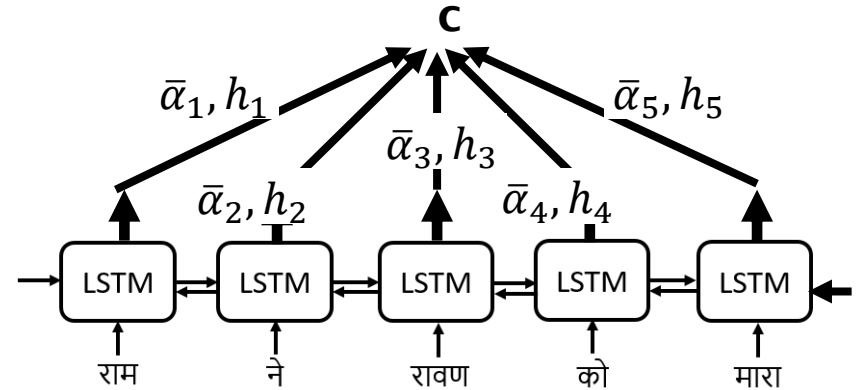
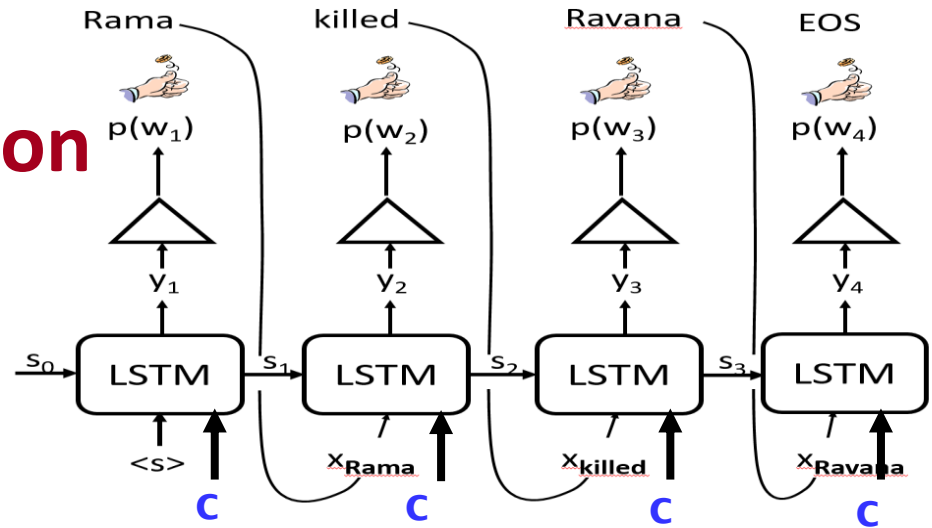
$$c = \sum_{i=1}^T \alpha_i \cdot h_i$$

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

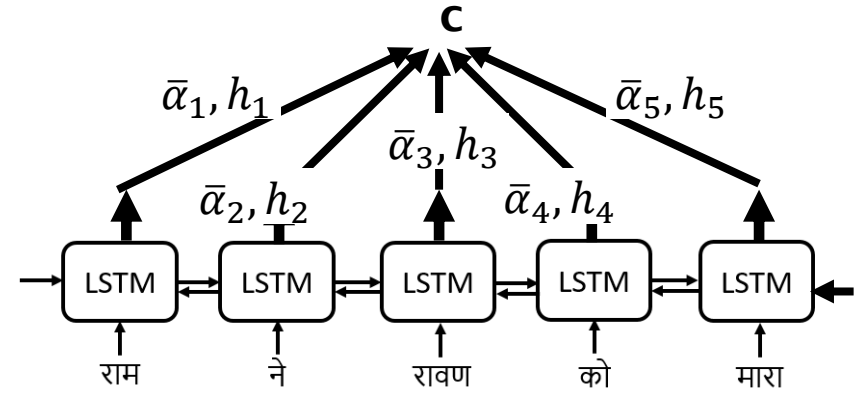
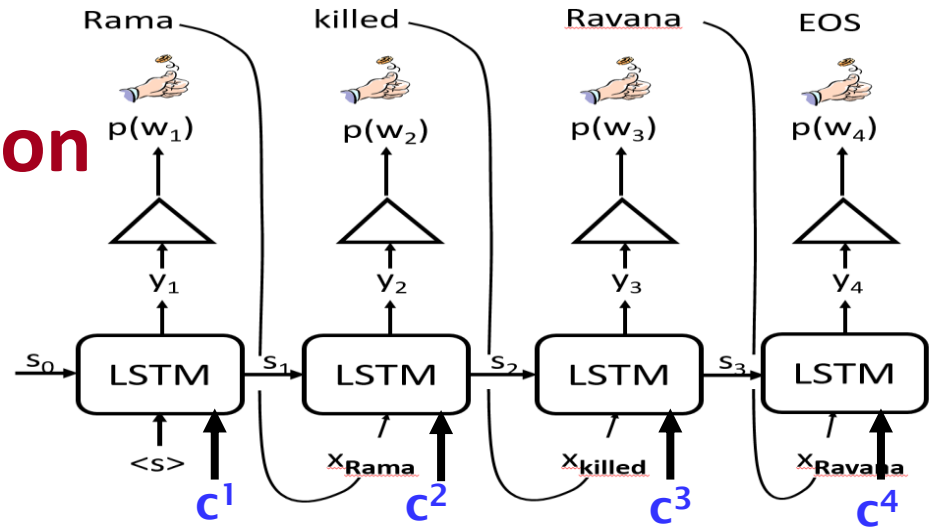
$$\bar{\alpha}_i = \phi^{\text{att}}(q, h_i)$$



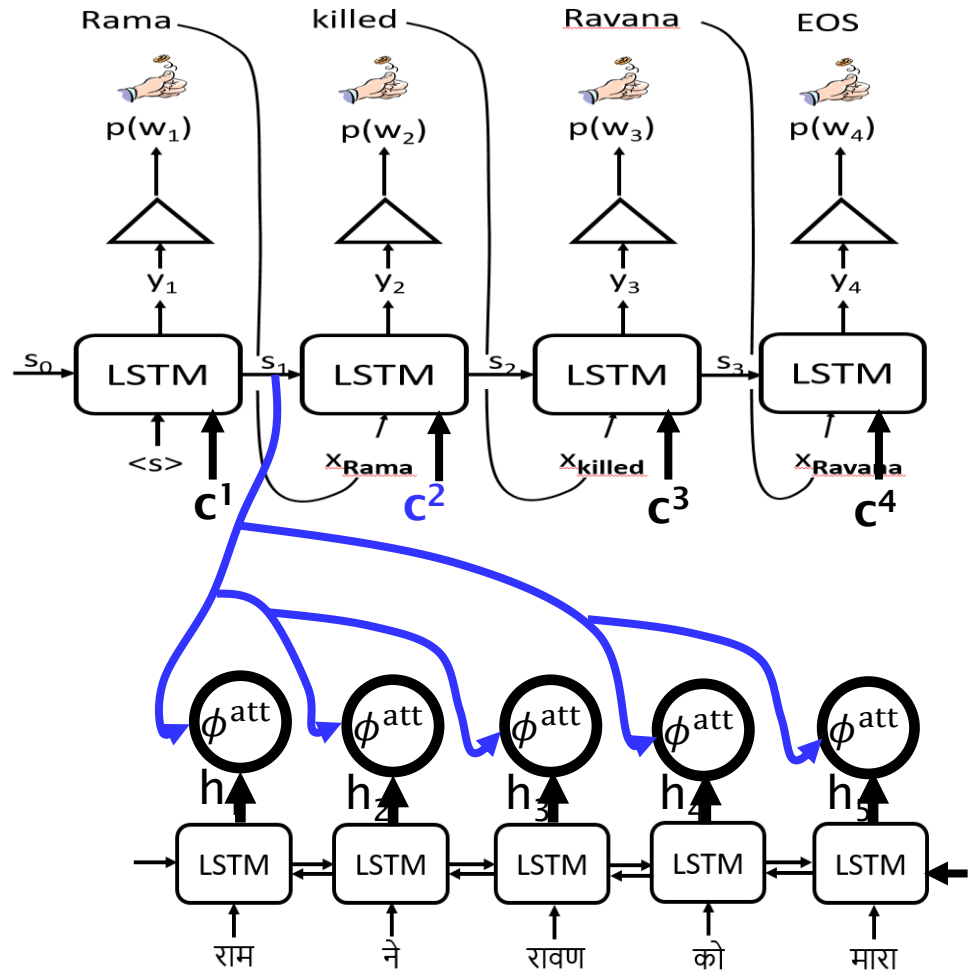
Seq2Seq with Attention



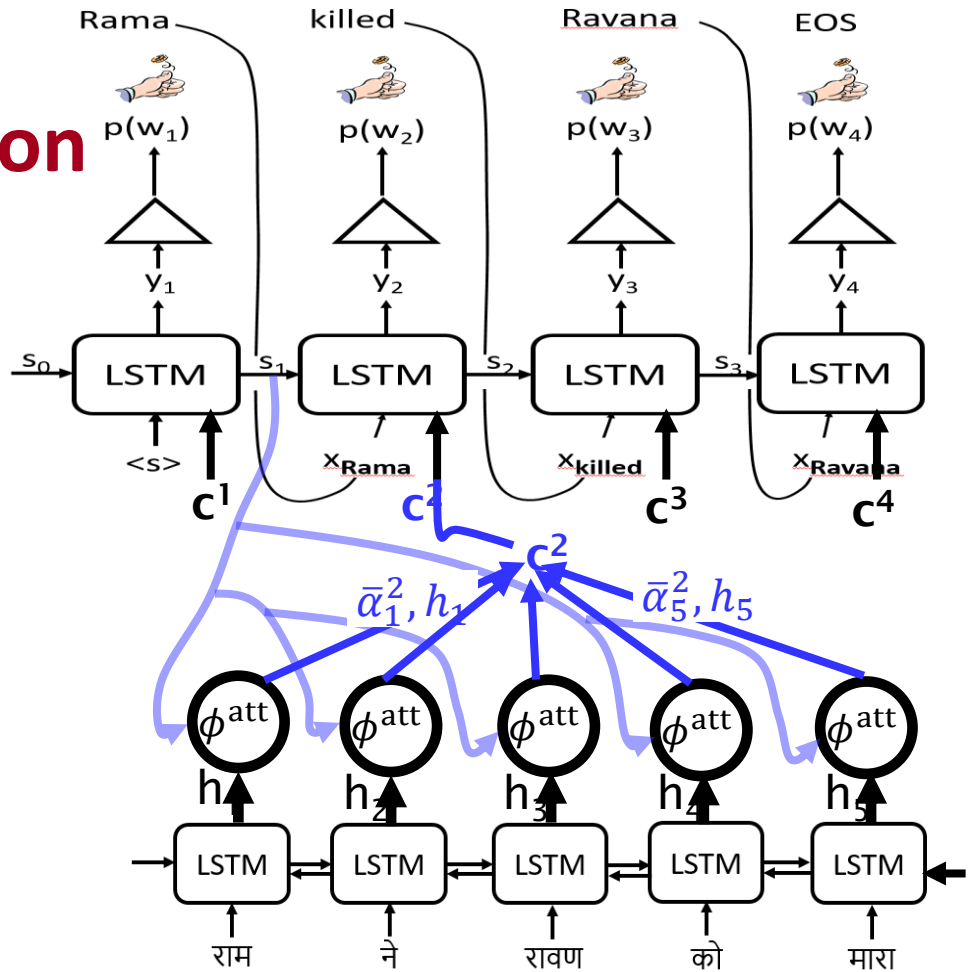
Seq2Seq with Attention



Seq2Seq with Attenti



Seq2Seq with Attention



Seq2Seq with Attention

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

$$\begin{aligned}\bar{\alpha}_i^j &= \phi^{\text{att}}(s_{j-1}, h_i) \\ \alpha^j &= \text{softmax}(\bar{\alpha}_1^j, \bar{\alpha}_2^j, \dots, \bar{\alpha}_T^j) \\ c^j &= \sum_{i=1}^T \alpha_i^j \cdot h_i\end{aligned}$$

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

$$\begin{aligned}p_j(w) &= \text{softmax}(\text{MLP}^{\text{out}}(s_j)) \\ z[j] &\sim p_j(w)\end{aligned}$$

Encoder-Decoder with Attention

- Encoder encodes a sequence of vectors, h_1, \dots, 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 .

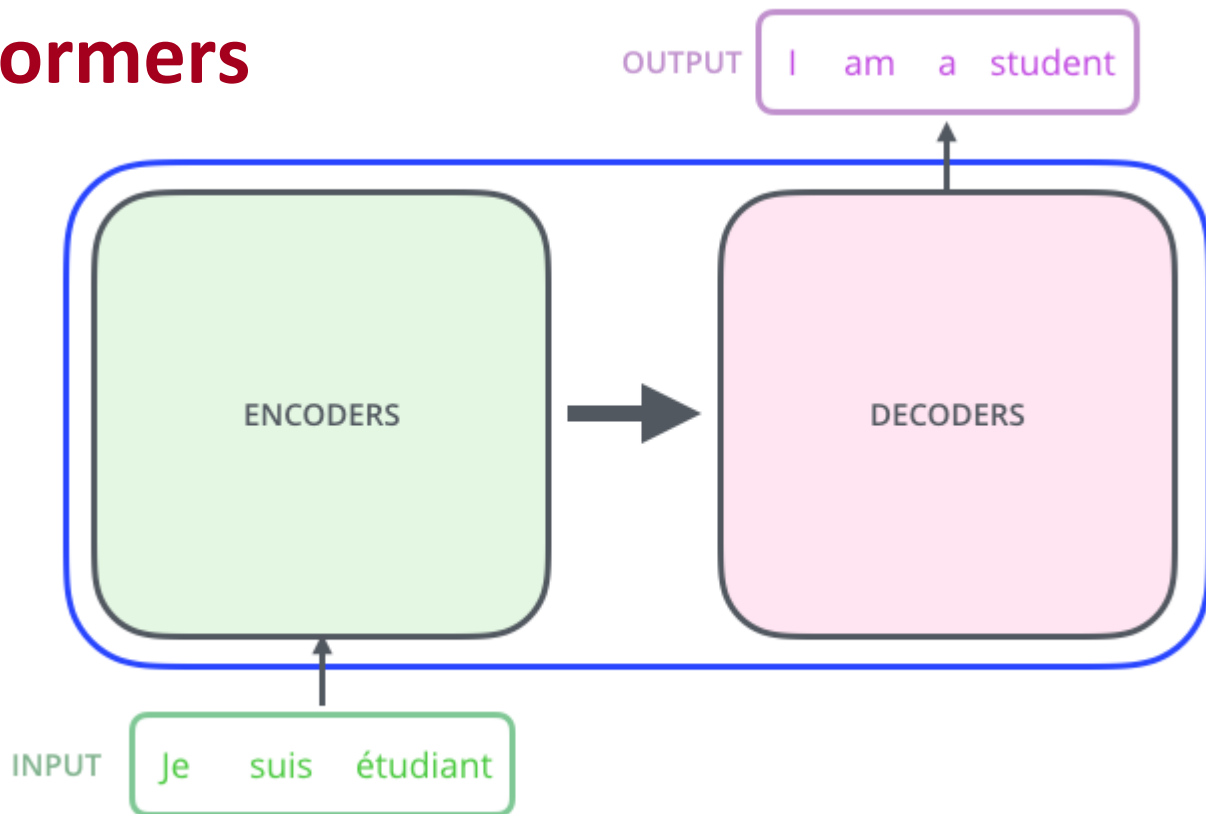
Encoder-Decoder with Attention

- 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: $O(nm)$

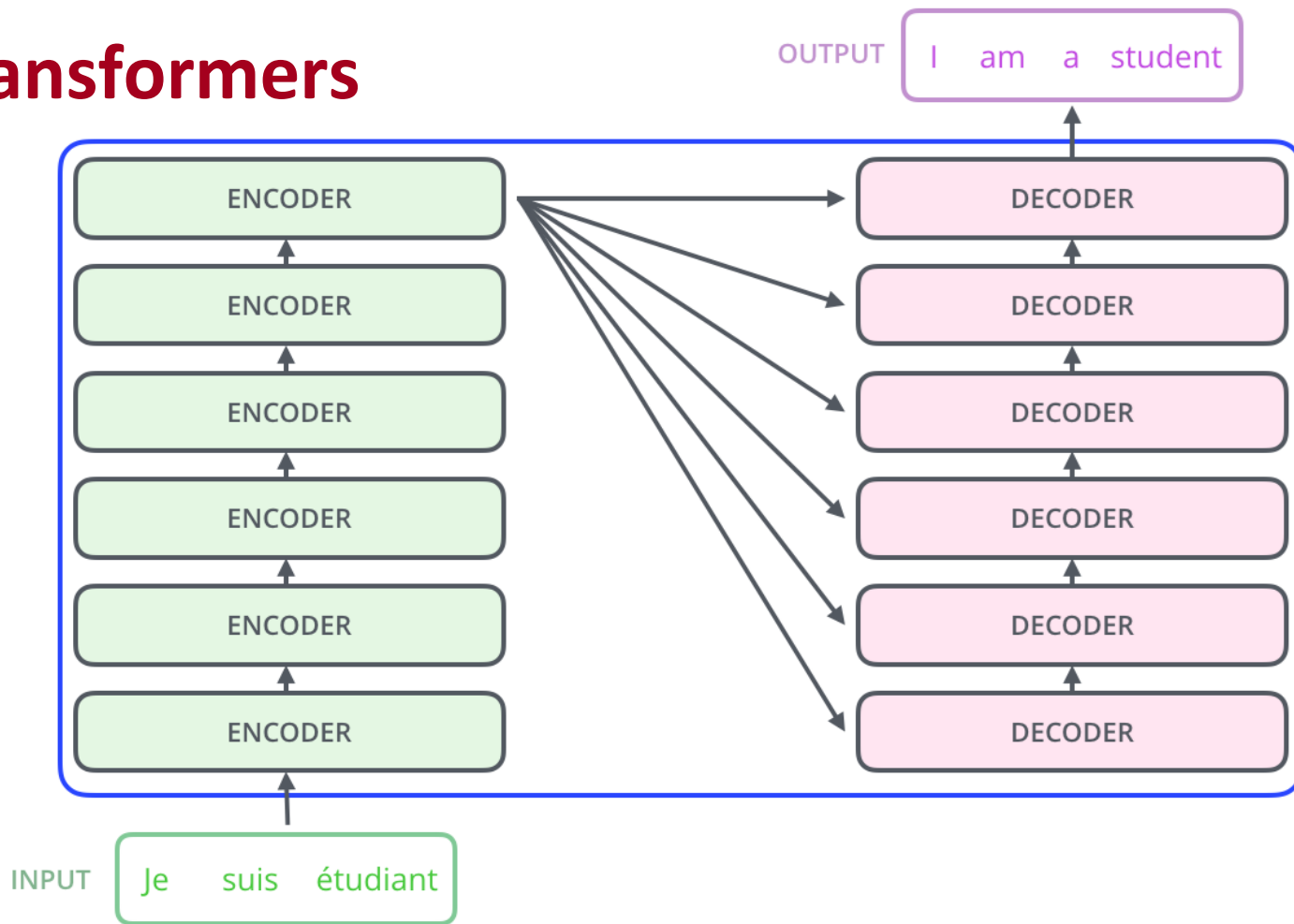
Outline

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

Transformers



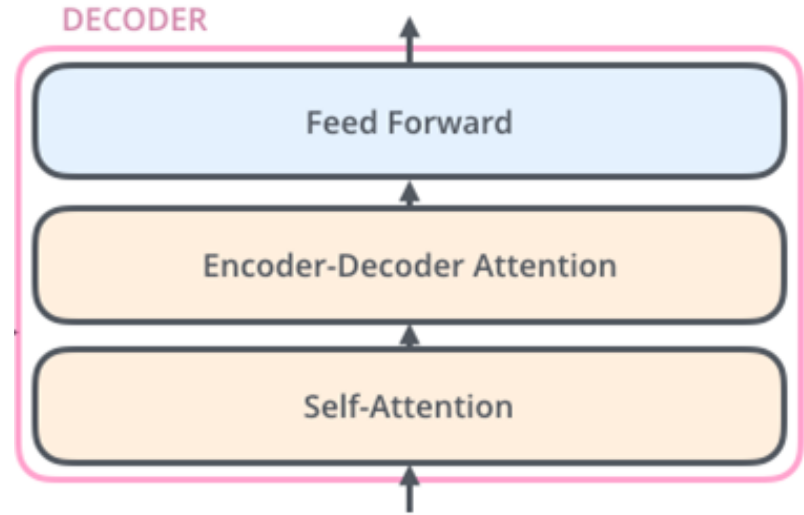
Transformers



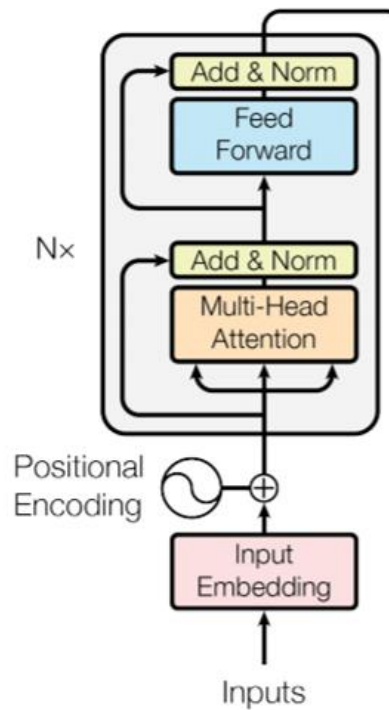
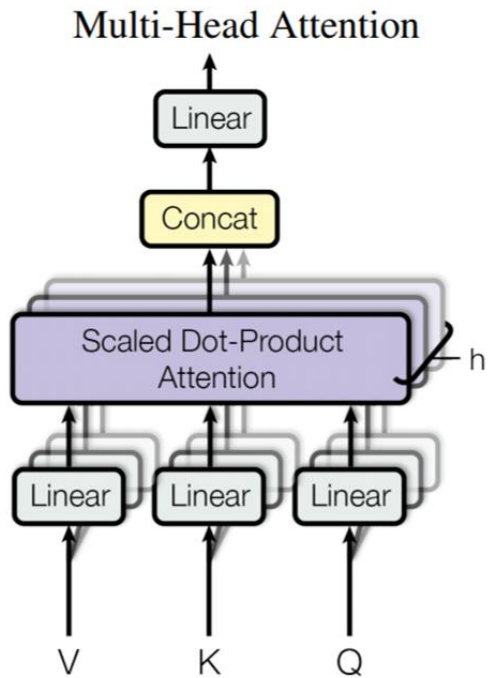
Encoder-Decoder

One key difference from decoder:

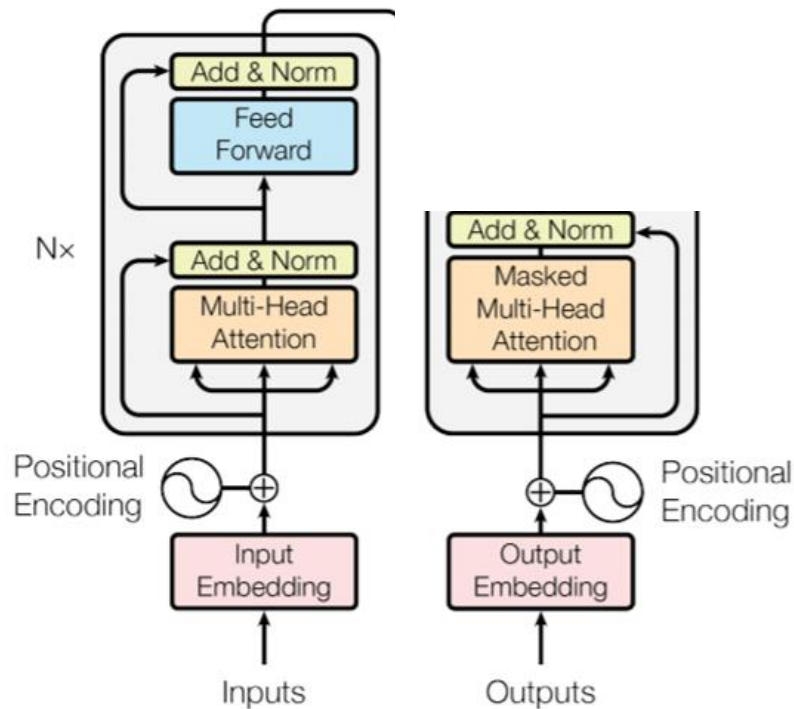
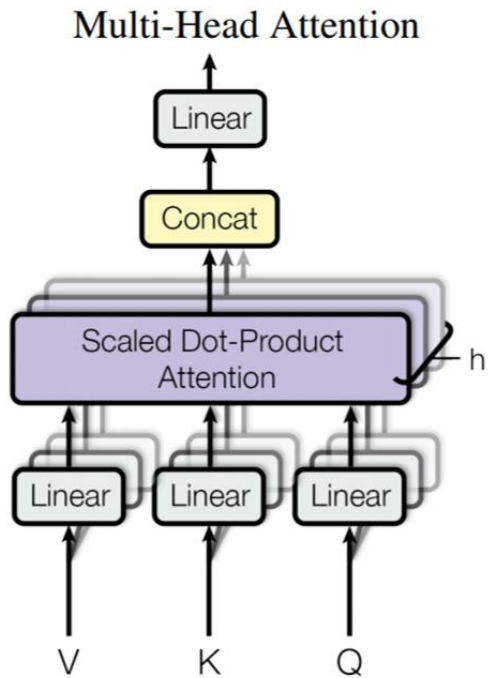
- Additional encoder-decoder attention layer where keys, values come from last encoder layer.



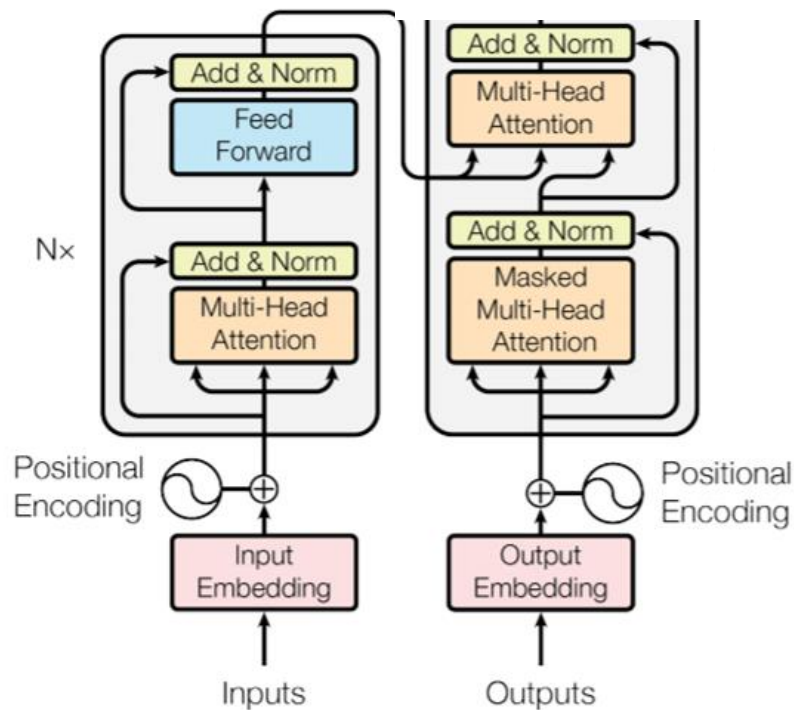
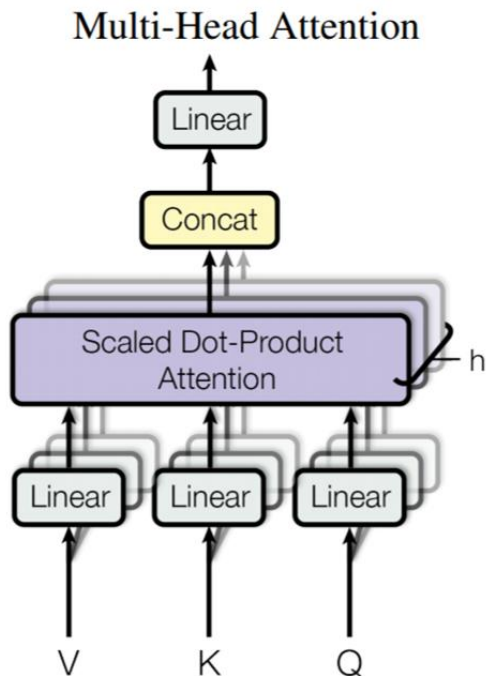
Full architecture with Attention reference



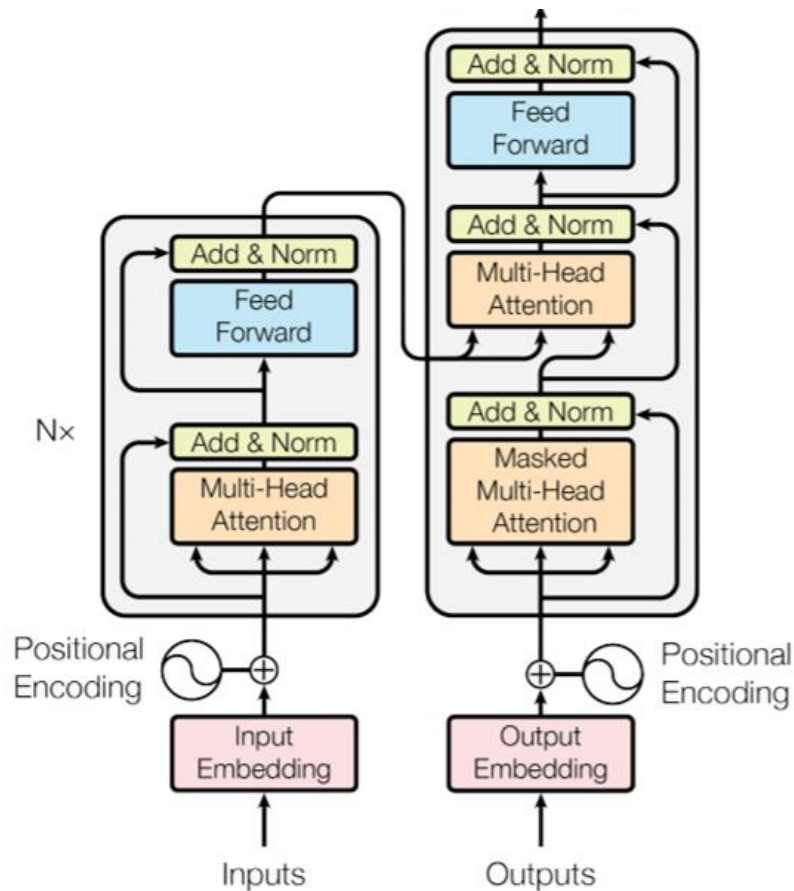
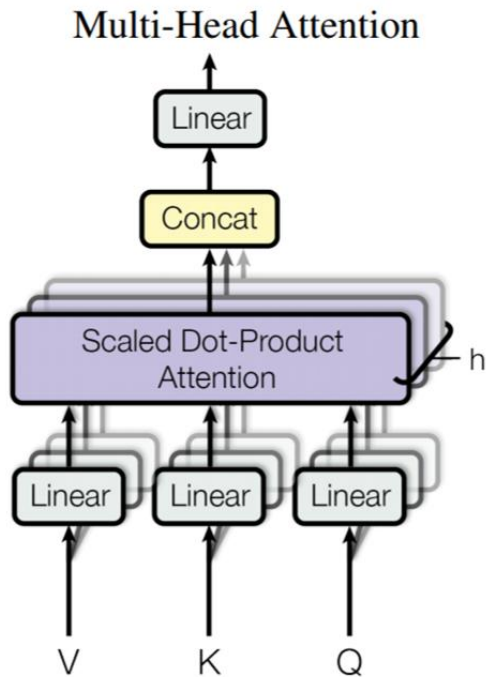
Full architecture with Attention reference



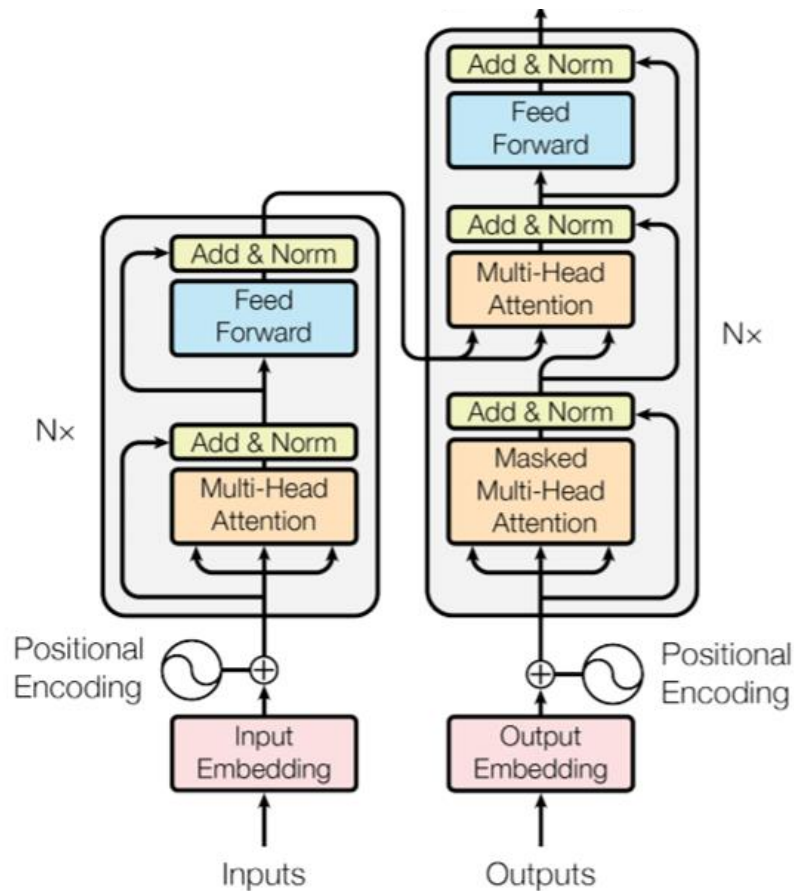
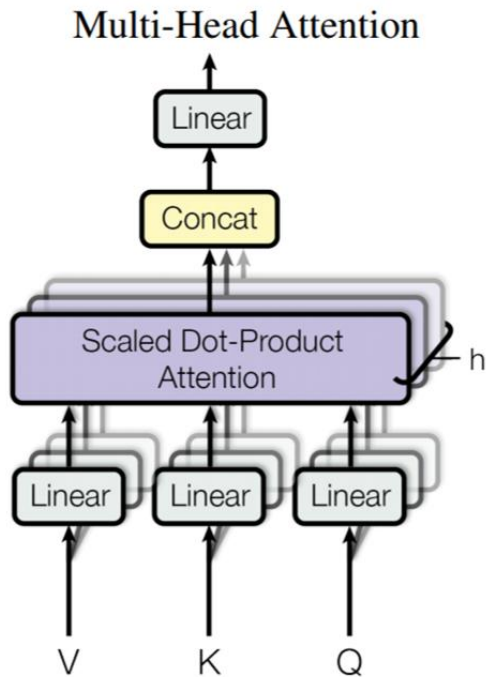
Full architecture with Attention reference



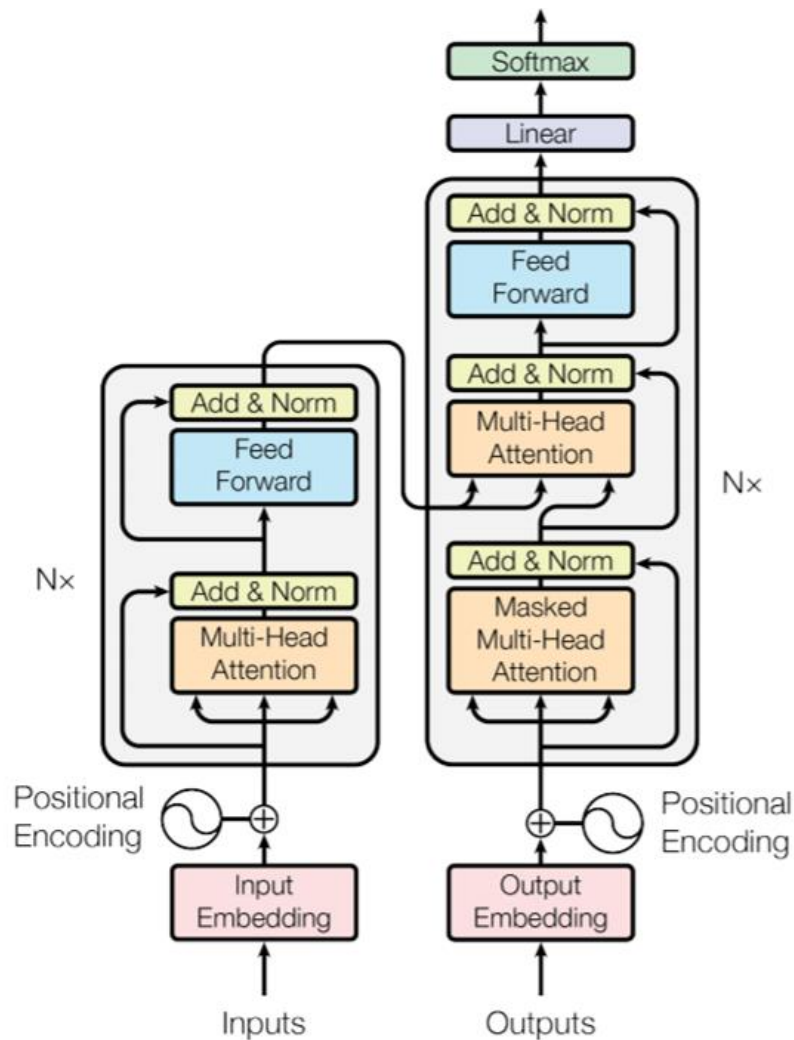
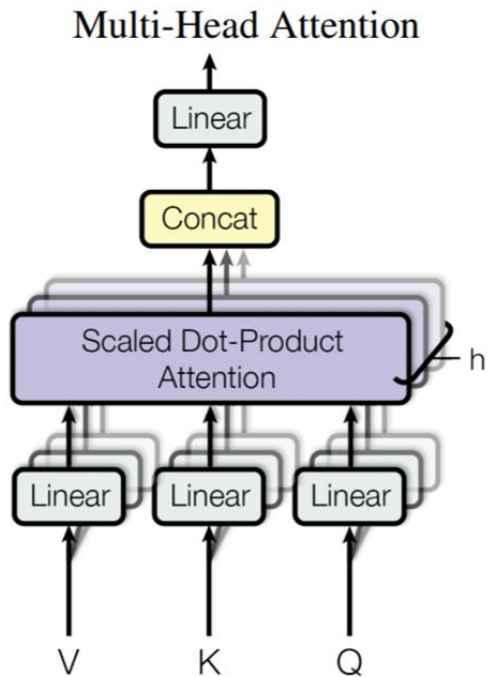
Full architecture with Attention reference



Full architecture with Attention reference

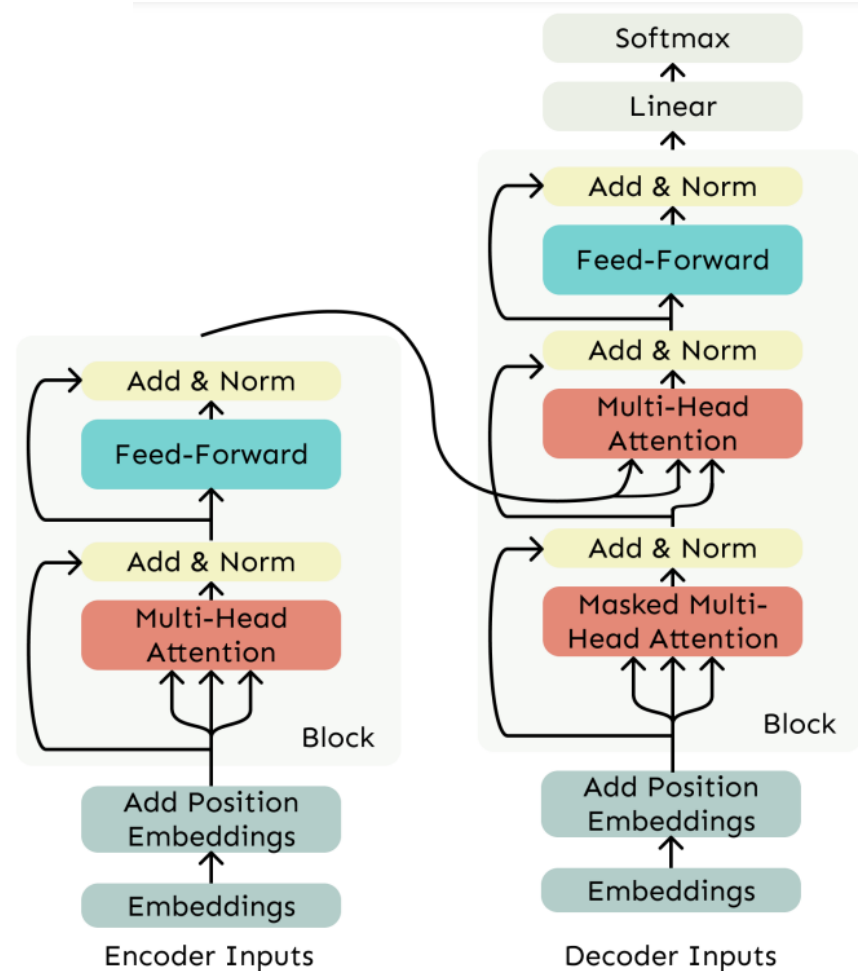


Full architecture with Attention reference



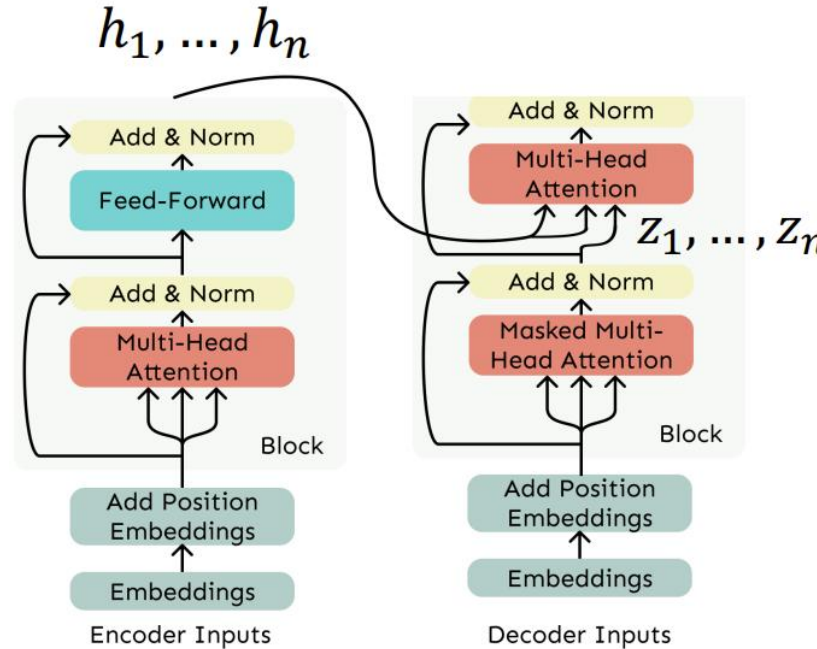
Encoder-Decoder

- we process the source sentence with a bidirectional model and generated the target with a unidirectional model.
- For this kind of seq2seq format, we often use a Transformer Encoder-Decoder.
- We use a normal Transformer Encoder.
- Our Transformer Decoder is modified to perform cross-attention to the output of the Encoder.



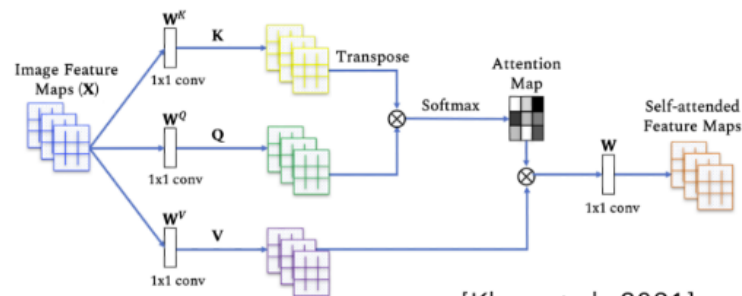
Cross-Attention Details

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like standard key-value attention
- Let h_1, \dots, h_n be **output** vectors from the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let z_1, \dots, z_n be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
 - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.



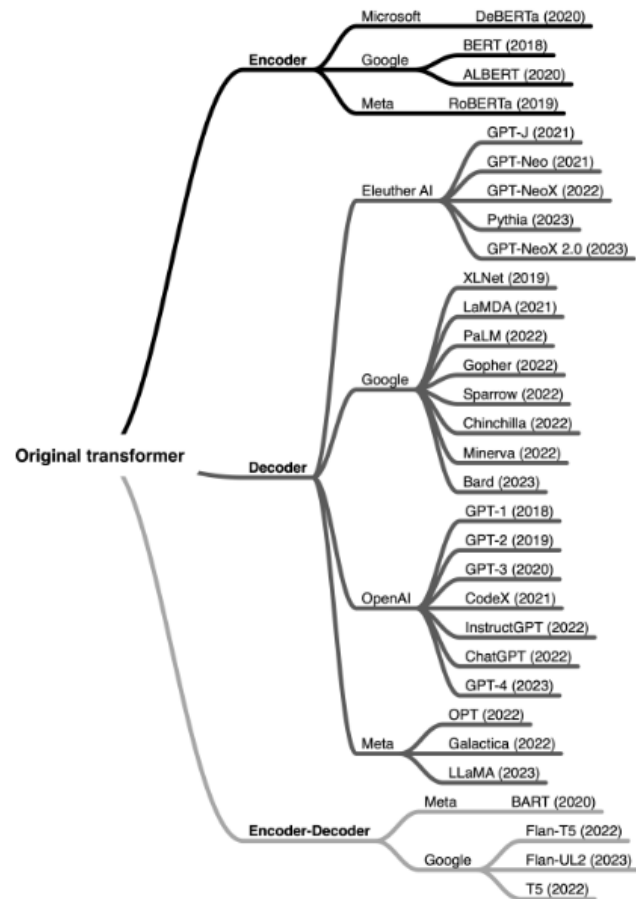
The Revolutionary Impact of Transformers

- **Almost all current-day leading language models** use Transformer building blocks.
 - E.g., GPT1/2/3/4, T5, Llama 1/2, BERT, ... almost anything we can name
 - Transformer-based models dominate nearly all NLP leaderboards.
- Since Transformer has been popularized in language applications, computer vision also adapted Transformers, e.g., **Vision Transformers**.



What's next after
Transformers?

Transformer Examples



Do Transformer Modifications Transfer Across Implementations and Applications?

- Generally, no!
 - (Narang et al 2021)

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	223M	11.1T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
GeLU	223M	11.1T	3.58	2.179 ± 0.003	1.838	75.79	17.86	25.13	26.47
Swish	223M	11.1T	3.62	2.186 ± 0.003	1.847	73.77	17.74	24.34	26.75
ELU	223M	11.1T	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	223M	11.1T	3.59	2.174 ± 0.003	1.814	74.20	17.42	24.34	27.12
GeGLU	223M	11.1T	3.55	2.130 ± 0.006	1.792	75.96	18.27	24.87	26.87
ReLU	223M	11.1T	3.57	2.145 ± 0.004	1.803	76.17	18.36	24.87	27.02
SeLU	223M	11.1T	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	223M	11.1T	3.53	2.127 ± 0.003	1.789	76.00	18.20	24.34	27.02
LiGLU	223M	11.1T	3.59	2.149 ± 0.005	1.798	75.34	17.97	24.34	26.53
Sigmoid	223M	11.1T	3.63	2.291 ± 0.019	1.867	74.31	17.51	23.02	26.30
Softplus	223M	11.1T	3.47	2.207 ± 0.011	1.850	72.45	17.05	24.34	26.89
RMS Norm	223M	11.1T	3.68	2.167 ± 0.008	1.821	75.45	17.54	24.07	27.14
Resero	223M	11.1T	3.51	2.282 ± 0.003	1.939	61.69	15.64	20.90	26.37
Resero + LayerNorm	223M	11.1T	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Resero + RMS Norm	223M	11.1T	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	223M	11.1T	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31
24 layers, $d_f = 1536, H = 6$	224M	11.1T	3.33	2.200 ± 0.007	1.843	74.89	17.75	25.13	26.89
18 layers, $d_f = 2048, H = 8$	223M	11.1T	3.38	2.185 ± 0.005	1.831	76.45	16.83	24.34	27.10
8 layers, $d_f = 4608, H = 18$	223M	11.1T	3.69	2.190 ± 0.005	1.847	74.58	17.69	23.28	26.85
6 layers, $d_f = 6144, H = 24$	223M	11.1T	3.70	2.201 ± 0.010	1.857	73.55	17.59	24.60	26.66
Block sharing	65M	11.1T	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	45M	9.4T	4.21	2.631 ± 0.305	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	20M	9.1T	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	170M	11.1T	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	144M	11.1T	3.70	2.352 ± 0.029	2.082	67.93	16.13	23.81	26.08
Factorized Embedding	227M	9.4T	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.25	26.50
Factorized & shared embeddings	202M	9.1T	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.72	26.44
Tied encoder/decoder input embeddings	248M	11.1T	3.55	2.192 ± 0.002	1.840	71.70	17.72	24.34	26.49
Tied decoder input and output embeddings	248M	11.1T	3.57	2.187 ± 0.007	1.827	74.86	17.74	24.87	26.67
Untied embeddings	273M	11.1T	3.53	2.195 ± 0.005	1.834	72.99	17.58	23.28	26.48
Adaptive input embeddings	204M	9.2T	3.55	2.250 ± 0.002	1.899	66.57	16.21	24.07	26.66
Adaptive softmax	204M	9.2T	3.60	2.364 ± 0.005	1.982	72.91	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.8T	3.43	2.229 ± 0.009	1.914	71.82	17.10	23.02	25.72
Mixture of softmaxes	232M	16.3T	2.24	2.227 ± 0.017	1.821	76.77	17.62	22.75	26.82
Transparent attention	223M	11.1T	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	26.80
Dynamic convolution	257M	11.8T	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.4T	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.9T	3.09	2.220 ± 0.003	1.863	73.67	10.76	24.07	26.58
Synthesizer (dense)	224M	11.4T	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	26.63
Synthesizer (dense plus)	243M	12.6T	3.22	2.191 ± 0.010	1.840	73.98	16.96	23.81	26.71
Synthesizer (dense plus alpha)	243M	12.6T	3.01	2.180 ± 0.007	1.828	74.25	17.02	23.28	26.61
Synthesizer (factorized)	207M	10.1T	3.94	2.341 ± 0.017	1.968	62.78	15.39	23.55	26.42
Synthesizer (random)	254M	10.1T	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.0T	3.63	2.189 ± 0.004	1.842	73.32	17.04	24.87	26.43
Synthesizer (random plus alpha)	292M	12.0T	3.42	2.186 ± 0.007	1.828	75.24	17.08	24.08	26.39
Universal Transformer	84M	40.0T	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.7T	3.20	2.148 ± 0.006	1.785	74.55	18.13	24.08	26.94
Switch Transformer	1100M	11.7T	3.18	2.135 ± 0.007	1.758	75.38	18.02	26.19	26.81
Funnel Transformer	223M	1.9T	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.0T	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	421M	386.6T	0.25	2.135 ± 0.003	1.798	75.16	17.04	23.55	26.73

Transformers (2017) Made Many Changes at Once!

- Delete all RNN components → Position encodings
- Residual connections
- Interspersing of Attention and MLP layers
- LayerNorms
- Multiple heads
- Carefully tuned hyperparameters

- Most original choices have stuck
 - sinusoidal embeddings!

What is missing in Transformers?

- Long sequence length
- External memory
- Better human controllability
- Align with language models of brain