IIT PALAKKAD

IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

Introduction to Deep Learning

Natural Language Processing

Introduction & RNNs

Mausam
IIT Delhi

National Supercomputing Mission

Centre for Development of Advanced Computing

What is Natural Language Processing?

- Natural Language: languages spoken by people (English, Hindi..)
  - As opposed to used by machines (Java, C++..)

- Natural Language Processing
  - Use of computers to
    - understand
    - process
    - generate
  - natural language

What is NLP?

Computer processing of human language

Sentiment Analysis, POS Tagging

NL Generation

Machine Translation, Summarization, Chatbots

Common NLP Tasks

- Word-level Tasks
  - Understanding word synonyms, word senses…
- Sentence/Document Classification
  - Sentiment Mining, Fake news detection, Racist tweet classification
- Sequence Labeling
  - POS Tagging, Noun Phrase Chunking, Named Entity Recognition
- Parsing: converting sentence to its syntactic structure
- Generation Tasks
  - Machine Translation, Summarization, Dialogue Systems

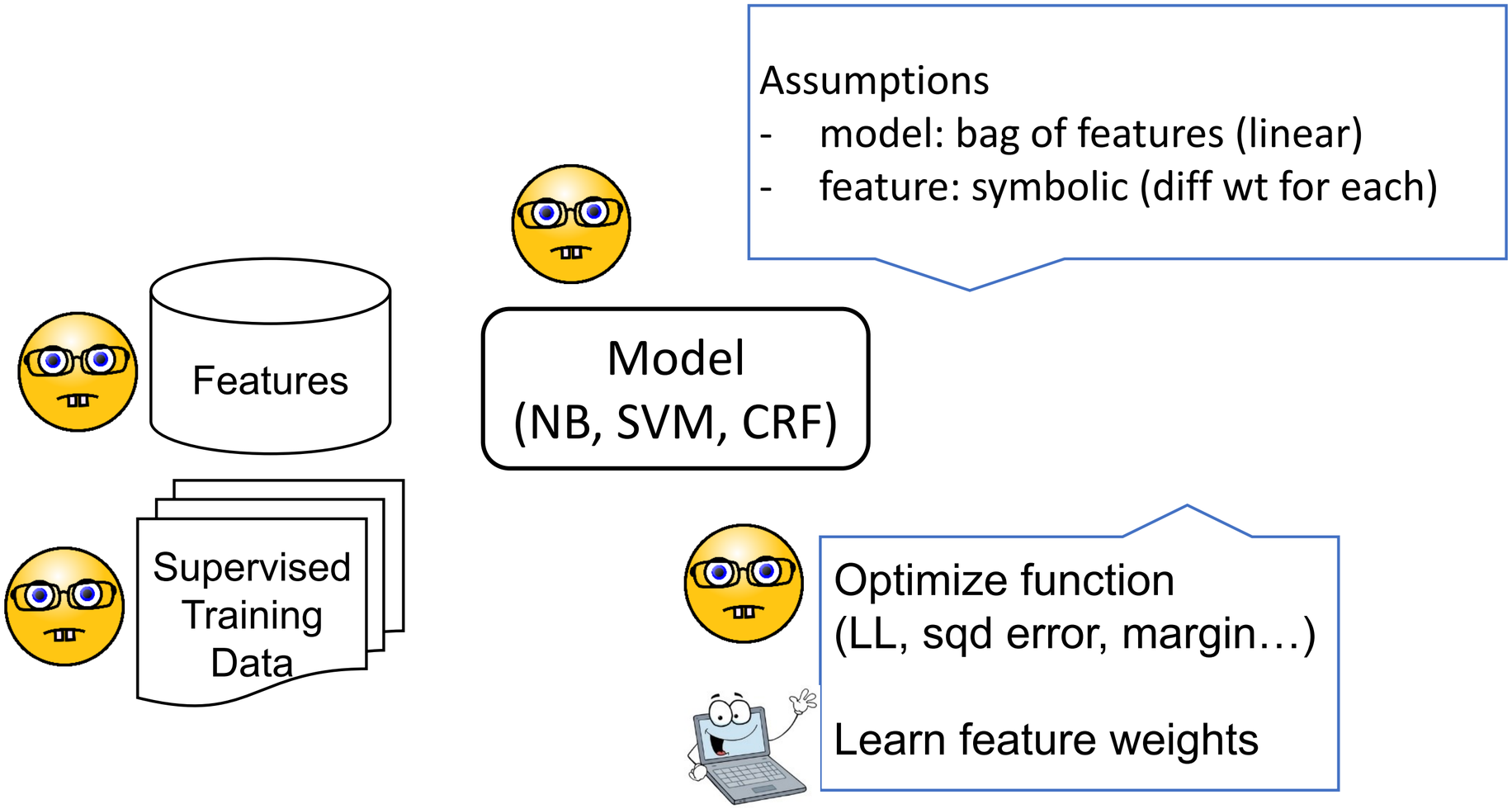
Three Generations of NLP

- Hand-crafted Systems – Knowledge Engineering [1950s–]
  - Rules written by hand; adjusted by error analysis
  - Require experts who understand both the systems and domain
  - Iterative guess-test-tweak-repeat cycle

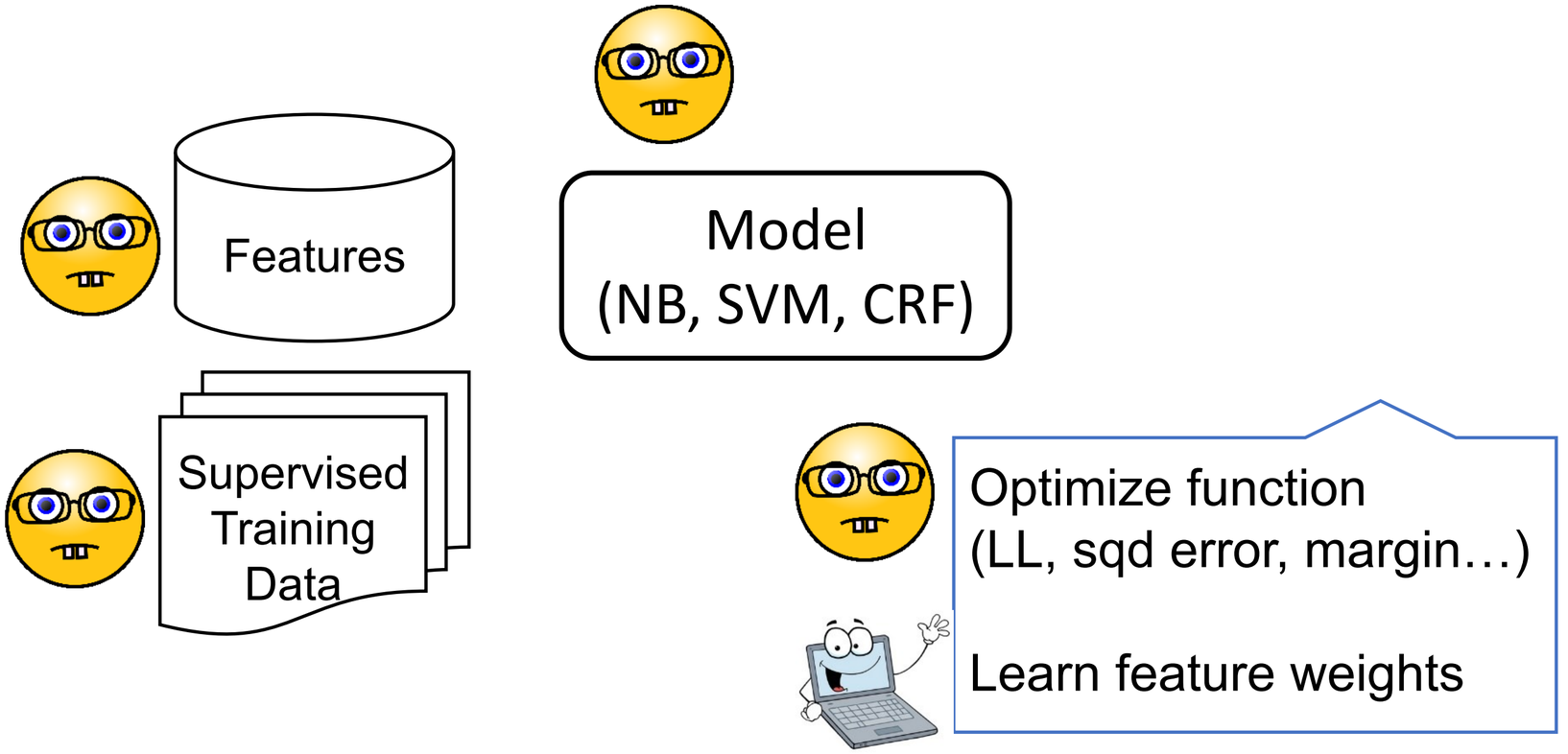
- Automatic, Trainable (Machine Learning) System [1985s–]
  - The tasks are modeled in a statistical way
  - More robust techniques based on rich annotations
  - Features provided by humans, weights learned by models

- Neural Models for NLP [2012 –]
  - The tasks are modeled in a neural architecture
  - Features are trained as part of the neural model

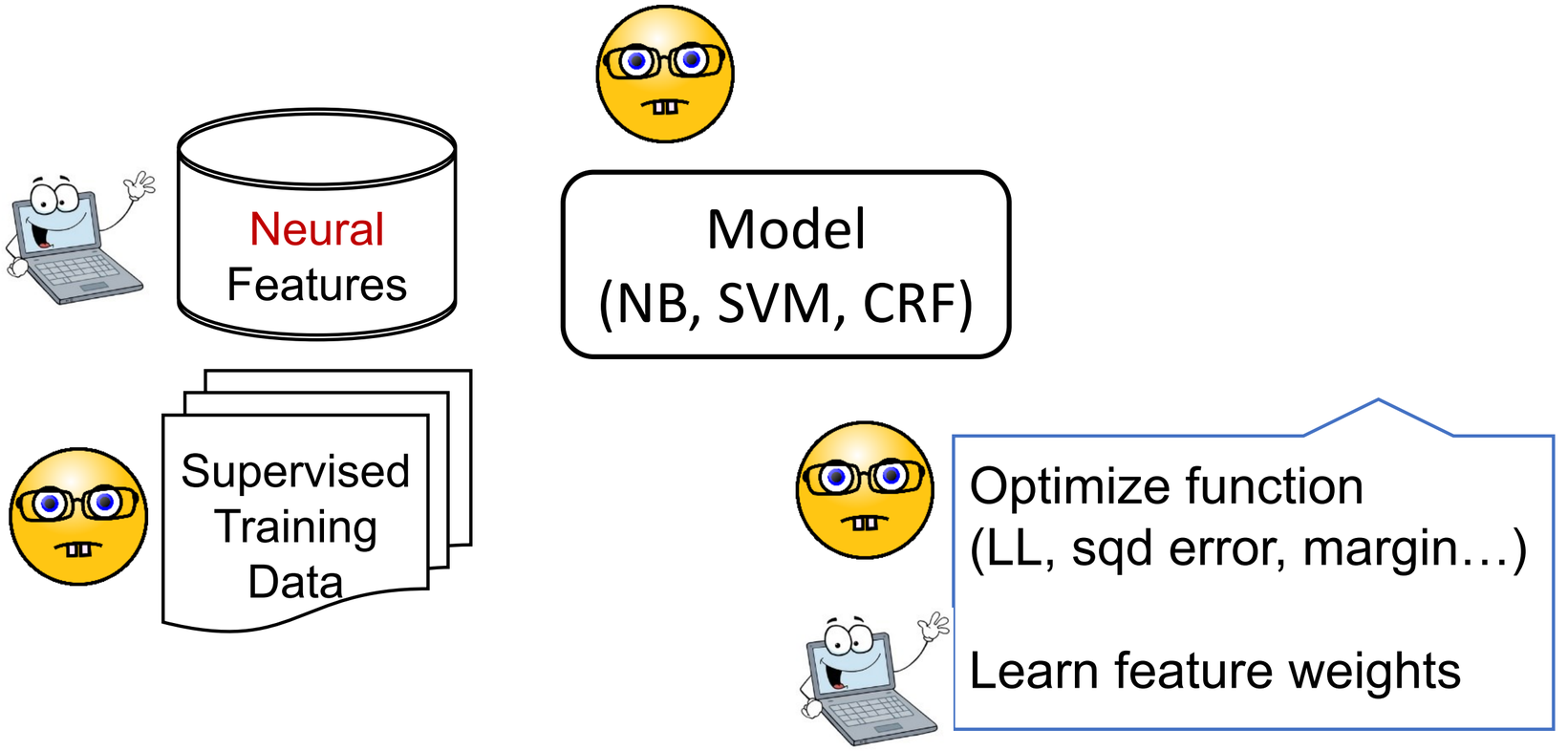
NLP before DL



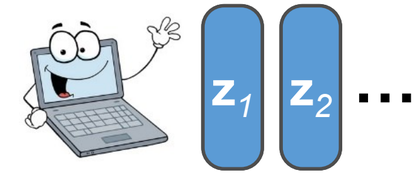
NLP with DL


Features
Supervised Training Data
Model (NB, SVM, CRF)
Optimize function (LL, sqd error, margin…)
Learn feature weights

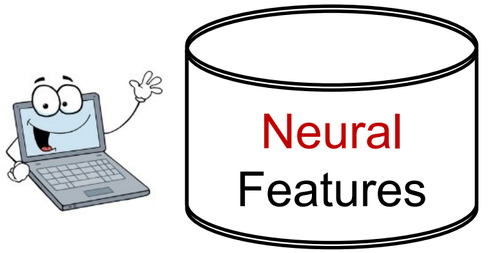
NLP with DL


Neural
Features
Model
(NB, SVM, CRF)
Supervised
Training
Data
Optimize function
(LL, sqd error, margin…)
Learn feature weights

NLP with DL



Word vectors

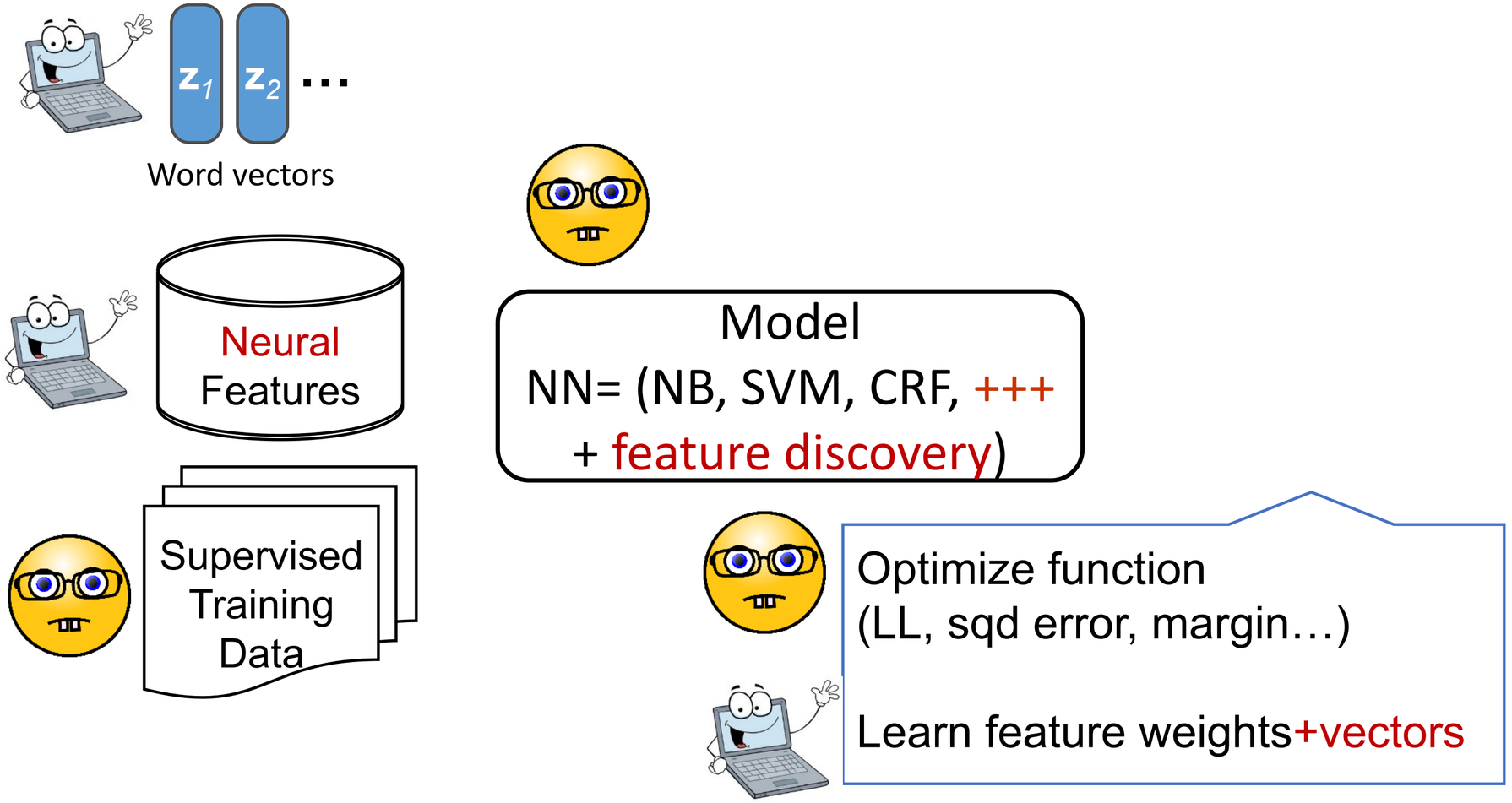


Model
(NB, SVM, CRF)

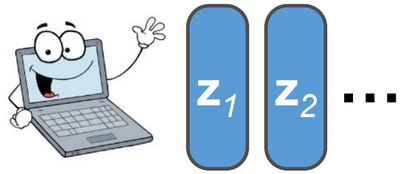
Optimize function
(LL, sqd error, margin…)

Learn feature weights+vectors

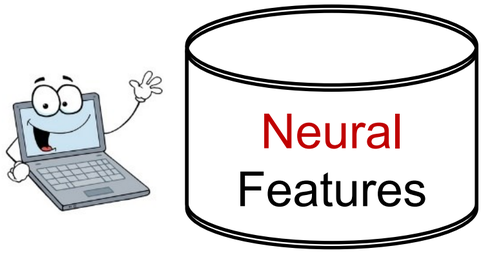
NLP with DL


z_1 z_2 ...
Word vectors
Neural Features
Supervised Training Data
Model
NN= (NB, SVM, CRF, +++
+ feature discovery)
Optimize function
(LL, sqd error, margin…)
Learn feature weights+vectors

NLP with DL



Word vectors



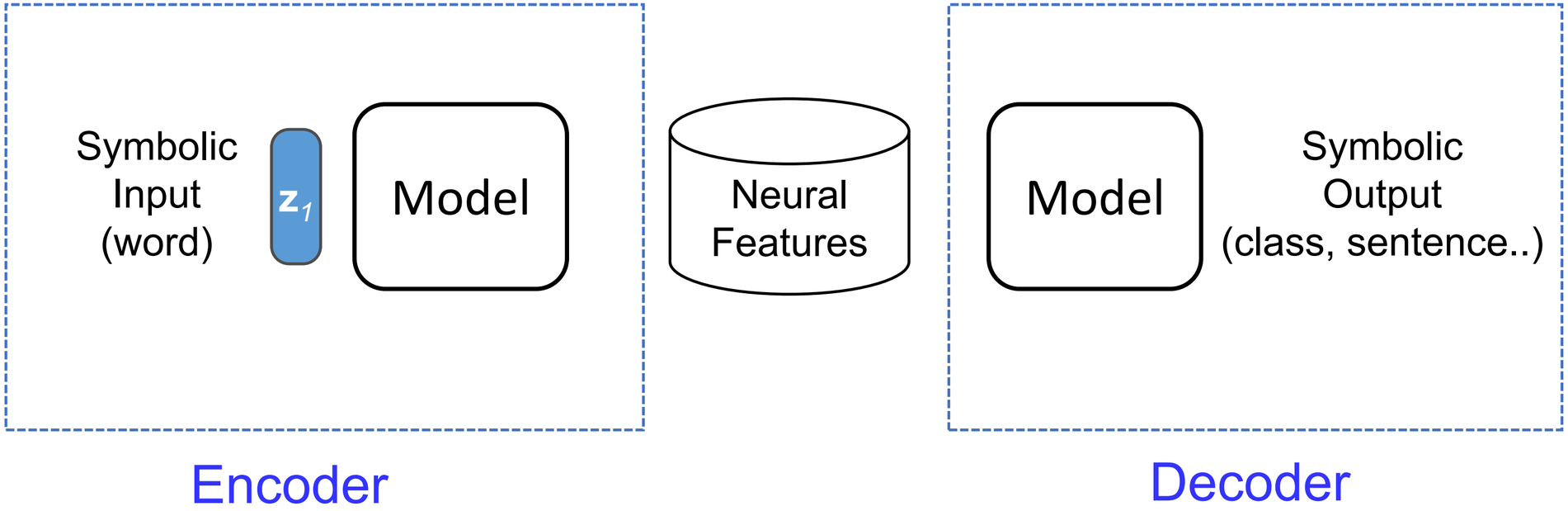
Assumptions
- doc/query/word is a vector of numbers
- feature: neural (weights are shared)
- model: bag/seq of features (non-linear)

Model
NN= (NB, SVM, CRF, +++
+ feature discovery)

Optimize function
(LL, sqd error, margin…)

Learn feature weights+vectors

Encoder-Decoder



Different assumptions on data create different architectures

A Primer on D.L. Building Blocks

- A single vector for an ordered pair of vectors?
- A single vector for a variable-sized bag of vectors?
- Project a vector to a new space?
- Are two vectors (from same space) similar?
- Are two vectors (from different space) similar?
- A new vector that depends on some vector input?

- x;y
- $\sum_i x_i$
- Wx
- x.y
- xWy
- g(Wx+b)

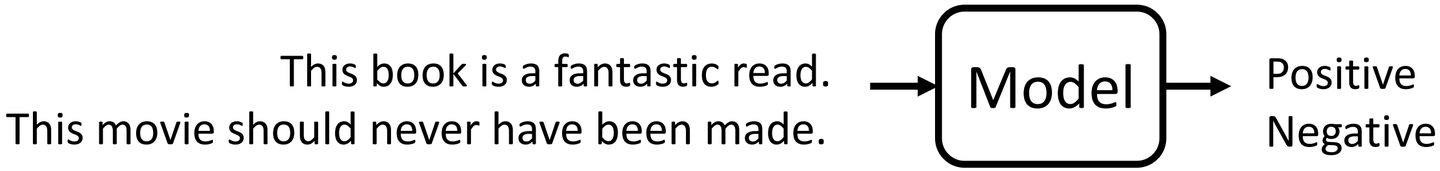
A Primer on D.L. Building Blocks

- Output a probability
- Output one of two classes
- Output one of many classes
- A feature w/ positive & negative influence
- A feature w/ positive influence for “deep” nets

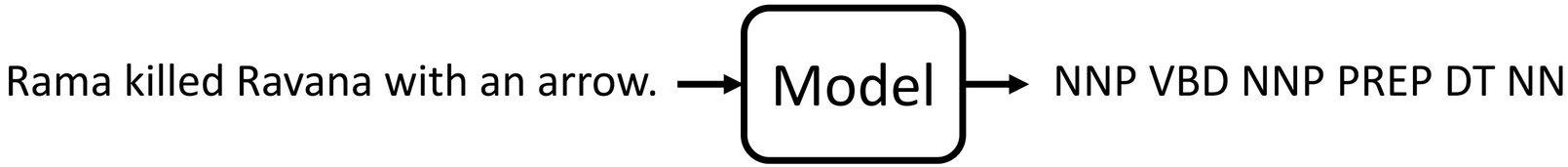
- σ
- σ
- softmax
- tanh
- ReLu

Main Challenge in Text Data

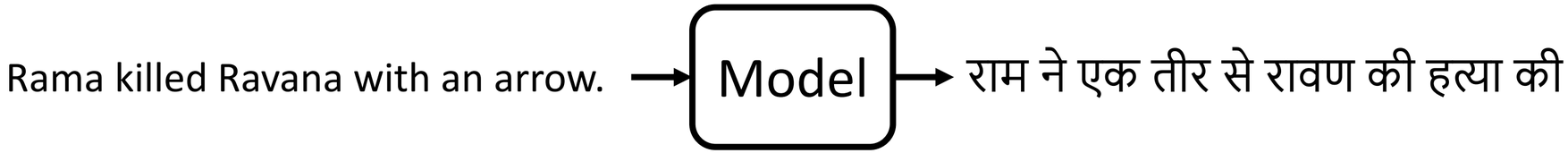
- Input (sentence) is *variable* length
- Output (classification) may be a single bit



- Output (sequence labeling) may be a sequence of same length as input

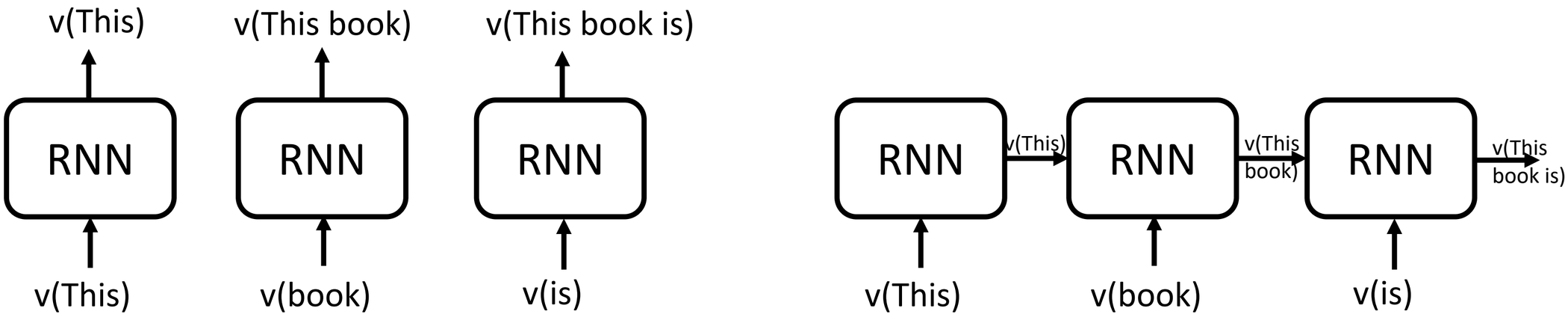


- Output (generation) may be sequence of length different from input

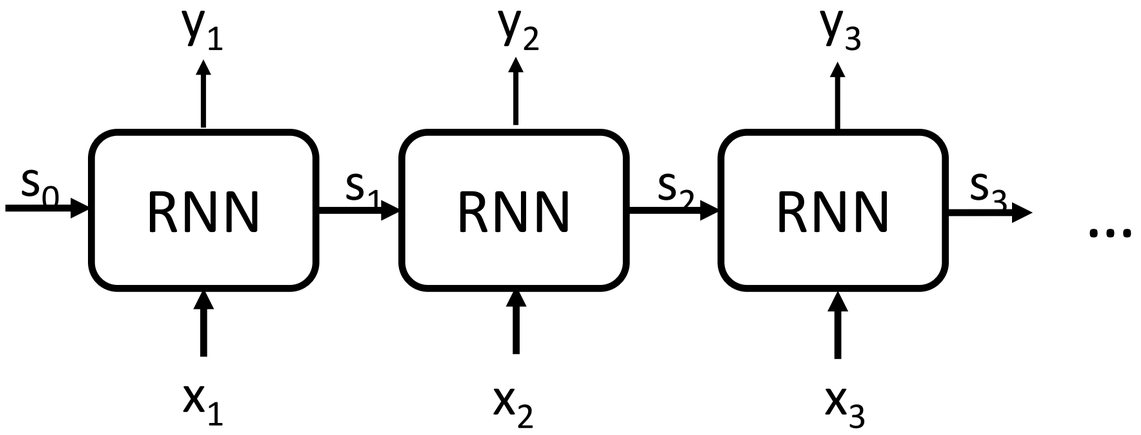


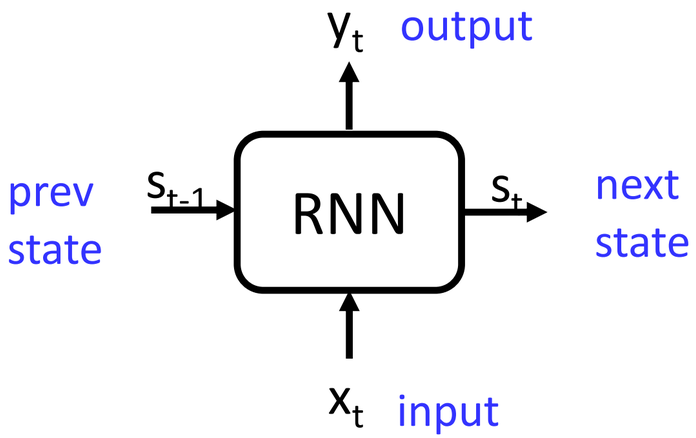
Recurrent Neural Networks (Encoder)

- Model to handle variable length input
  - Parameters/model cannot be position dependent
  - Same computation will be repeated at every position



Recurrent Neural Networks (Encoder)



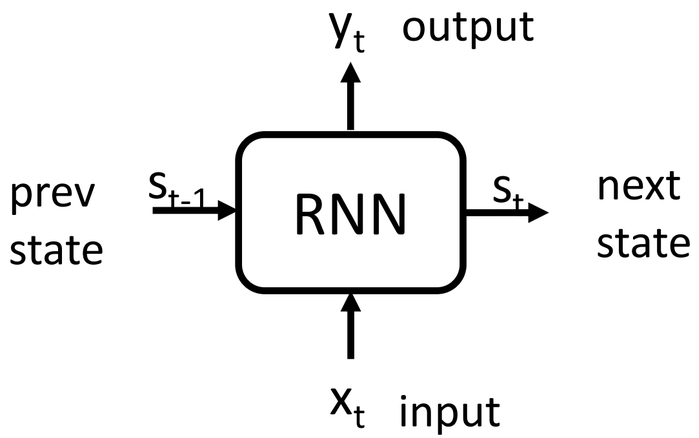


Recurrent Neural Networks (Encoder)

$$RNN(s_{t-1}, x_t) = s_t, y_t$$

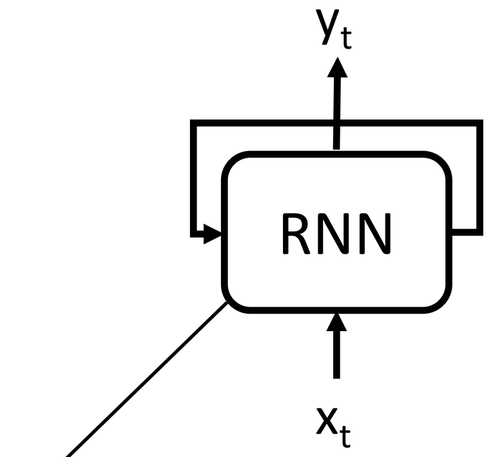
$$s_t = R(s_{t-1}, x_t)$$

$$y_t = O(s_t)$$



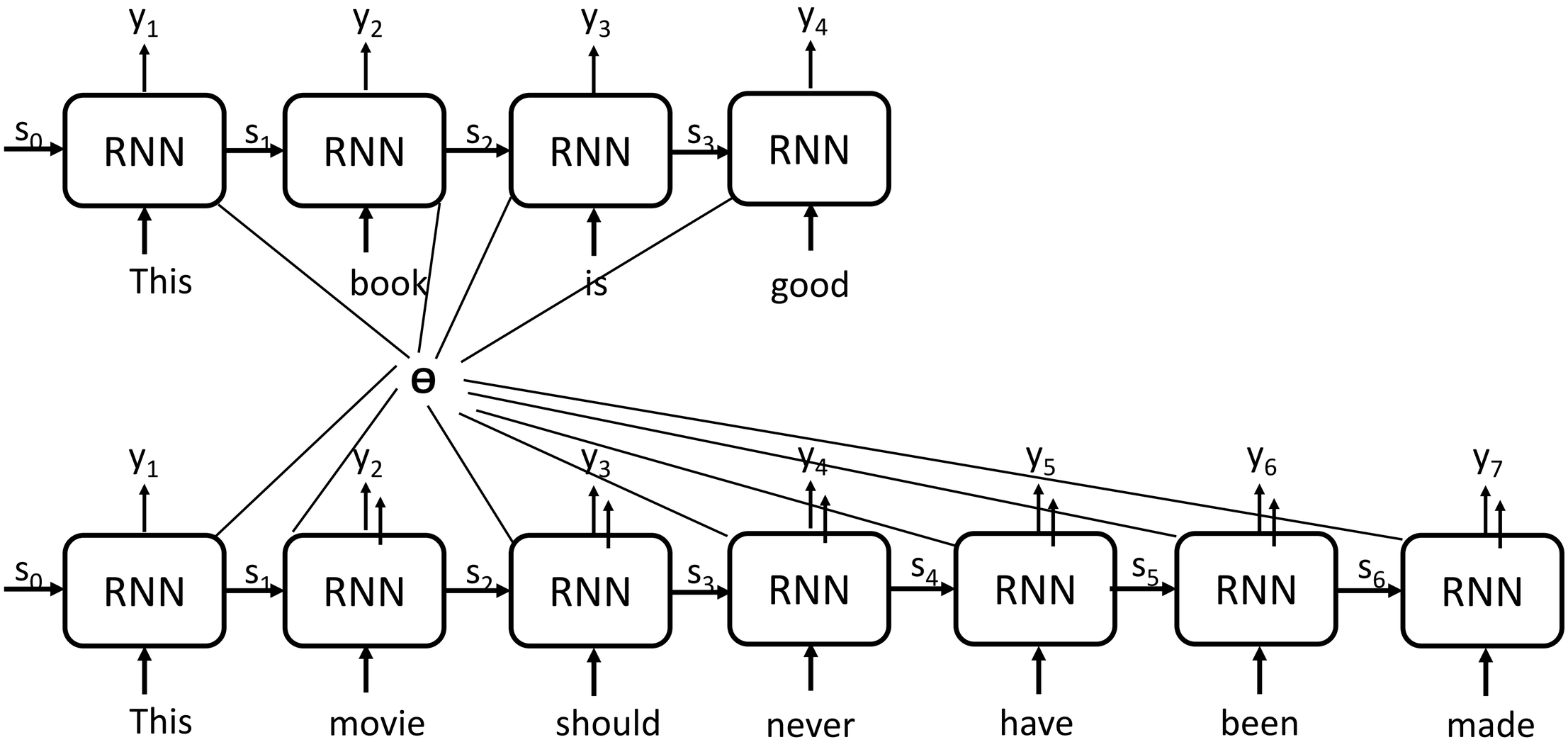
$$x_t \in \mathbb{R}^{din}$$
$$y_t \in \mathbb{R}^{dout}$$
$$s_t \in \mathbb{R}^{dstate}$$

- They are called recurrent nets
  - because the same computation recurs at each position
- There's a vector y_t for every prefix $x_{1:t}$



parameters don't depend on position

Unrolling an RNN



y_t depends on $x_{1:t}$

$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= R(s_{t-1}, x_t) \\
&= R(R(s_{t-2}, x_{t-1}), x_t) \\
&= R(R(R(s_{t-3}, x_{t-2}), x_{t-1}), x_t) \\
&\quad \ldots. \\
&= R(R(R \ldots R(s_0, x_1), x_2), \ \ldots), x_t)
\end{aligned}
$$

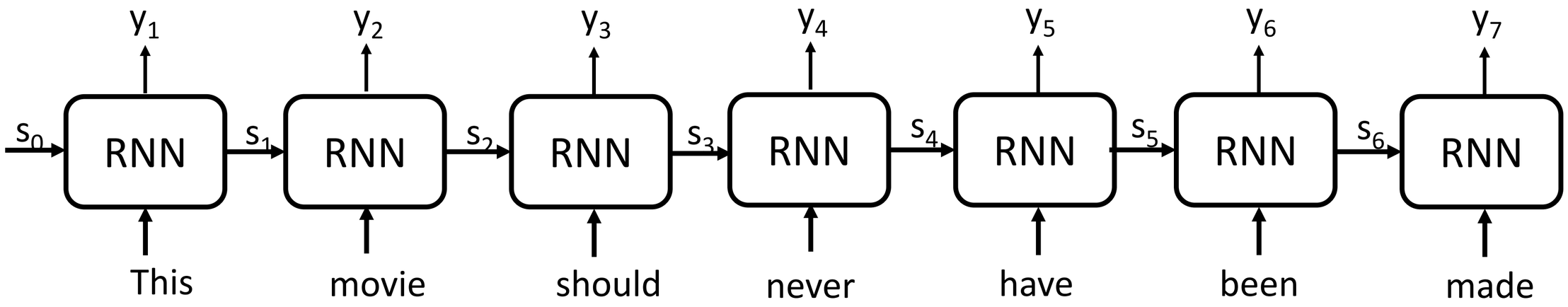
y_t depends on $x_{1:t}$

$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= R(s_{t-1}, x_t) \\
&= R(R(s_{t-2}, x_{t-1}), x_t) \\
&= R(R(R(s_{t-3}, x_{t-2}), x_{t-1}), x_t) \\
&\quad \ldots. \\
&= R(R(R \ldots R(s_0, x_1), x_2), \ \ldots), x_t)
\end{aligned}
$$

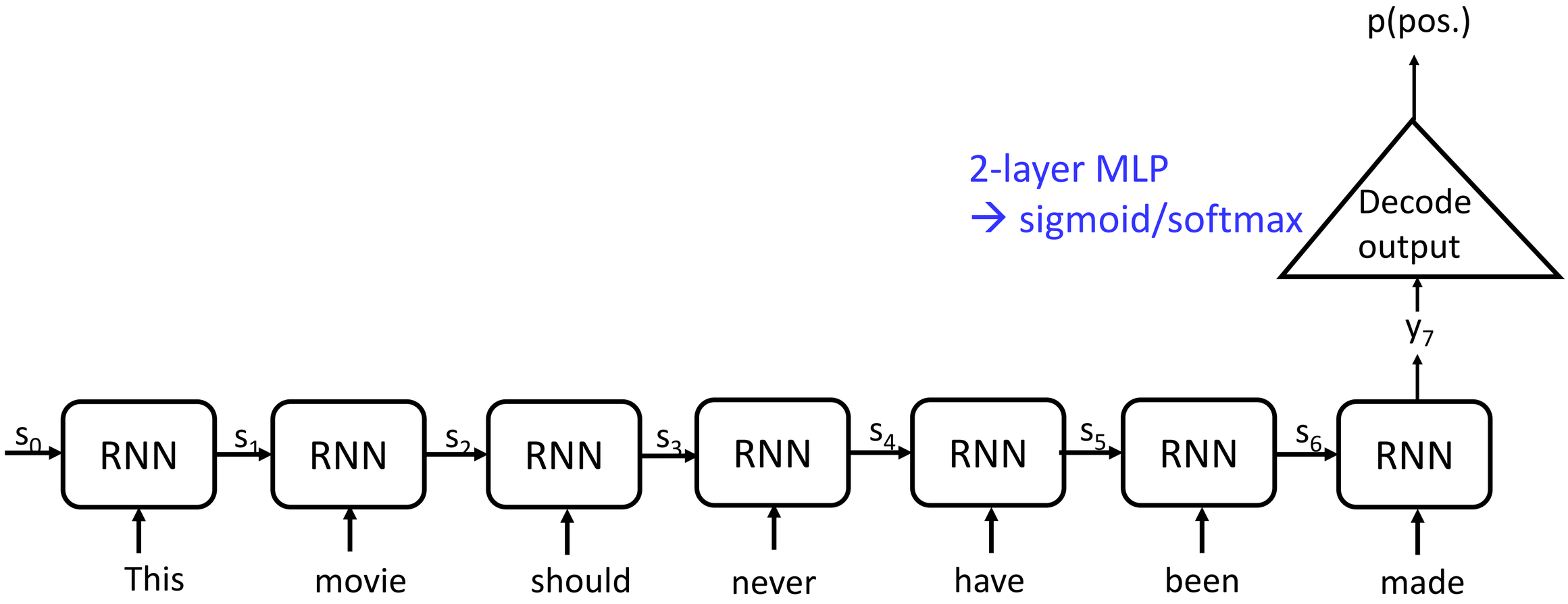
$$
\begin{aligned}
y_t &= O(s_t) \\
s_t &= RNN(s_0, x_{1:t})
\end{aligned}
$$

To make a single bit prediction for the full sentence decode y_t

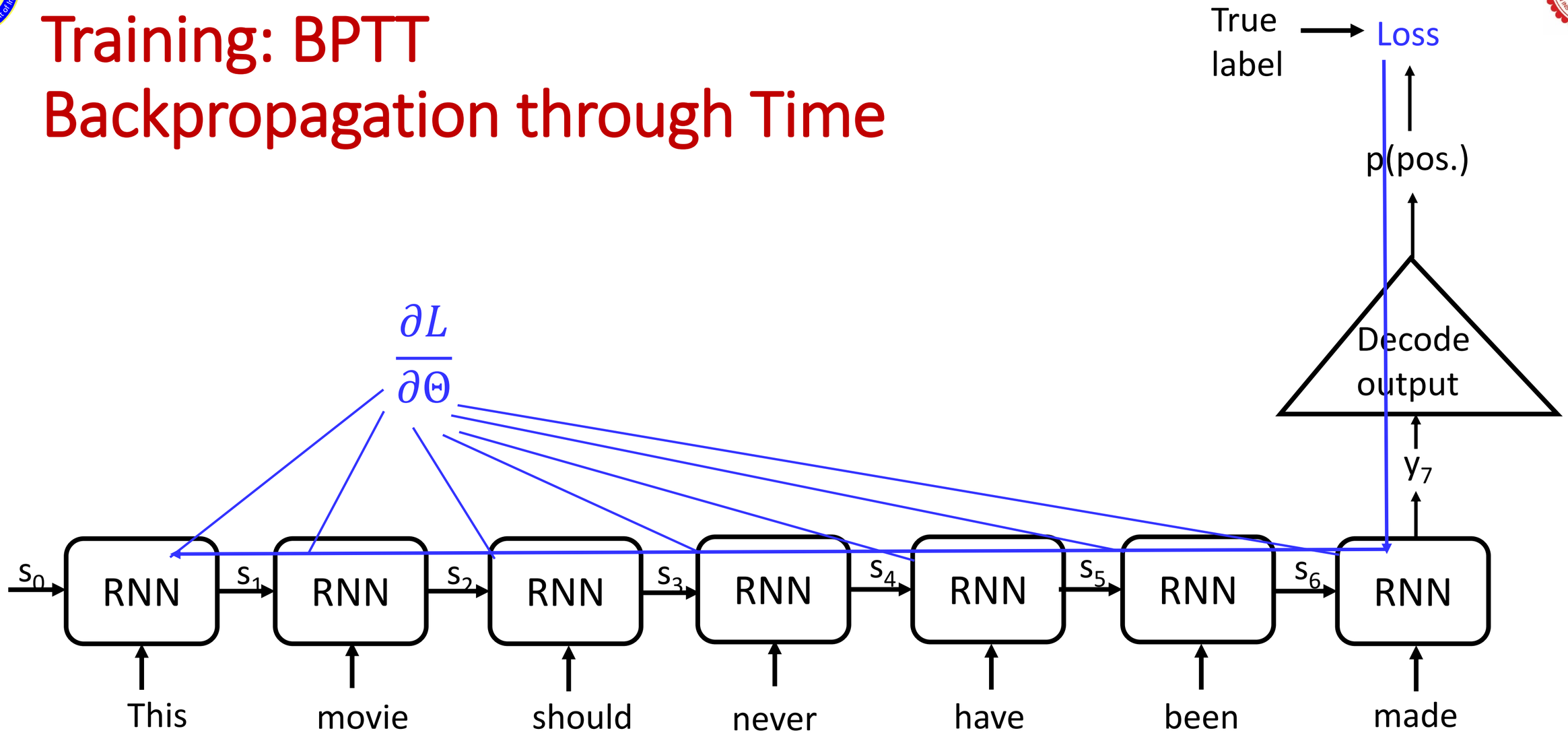
Sentiment Classification



Sentence Classification (Sentiment Mining)



Training: BPTT
Backpropagation through Time



Building a Simple RNN

- What are good functions for R and O ?

$$s_t = R(s_{t-1}, x_t)$$

$$y_t = O(s_t)$$

- Suggestion 1: $s_t = s_{t-1} + x_t$
- Problem?

- Suggestion2: $s_t = \tanh(s_{t-1} + x_t + bs)$
- Problem?

Building a Simple RNN

$$s_t = R(s_{t-1}, x_t)$$

$$y_t = O(s_t)$$

- What are good functions for R and O ?

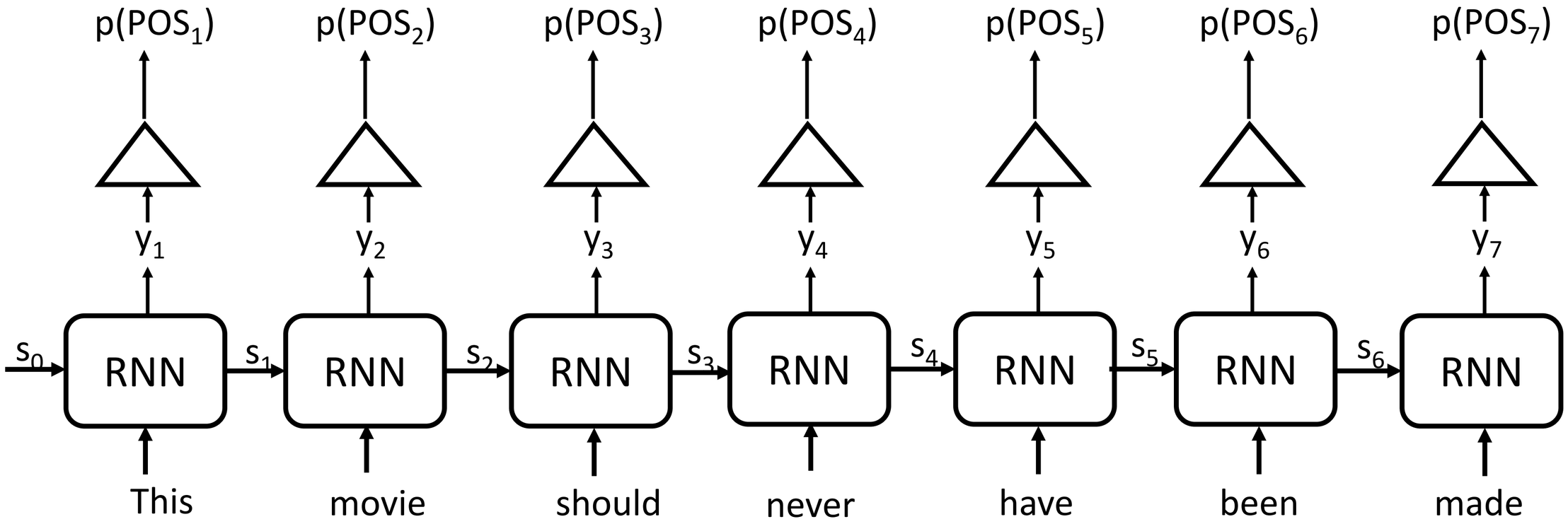
- Suggestion 1: $s_t = s_{t-1} + x_t$
- Problem?

- Suggestion2: $s_t = \tanh(s_{t-1} + x_t + b^s)$
- Problem?

- Elman's RNN: $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$ ⟵ $s_t = R(s_{t-1}, x_t)$

  $y_t = \tanh(W^y s_t + b^y)$ ⟵ $y_t = O(s_t)$

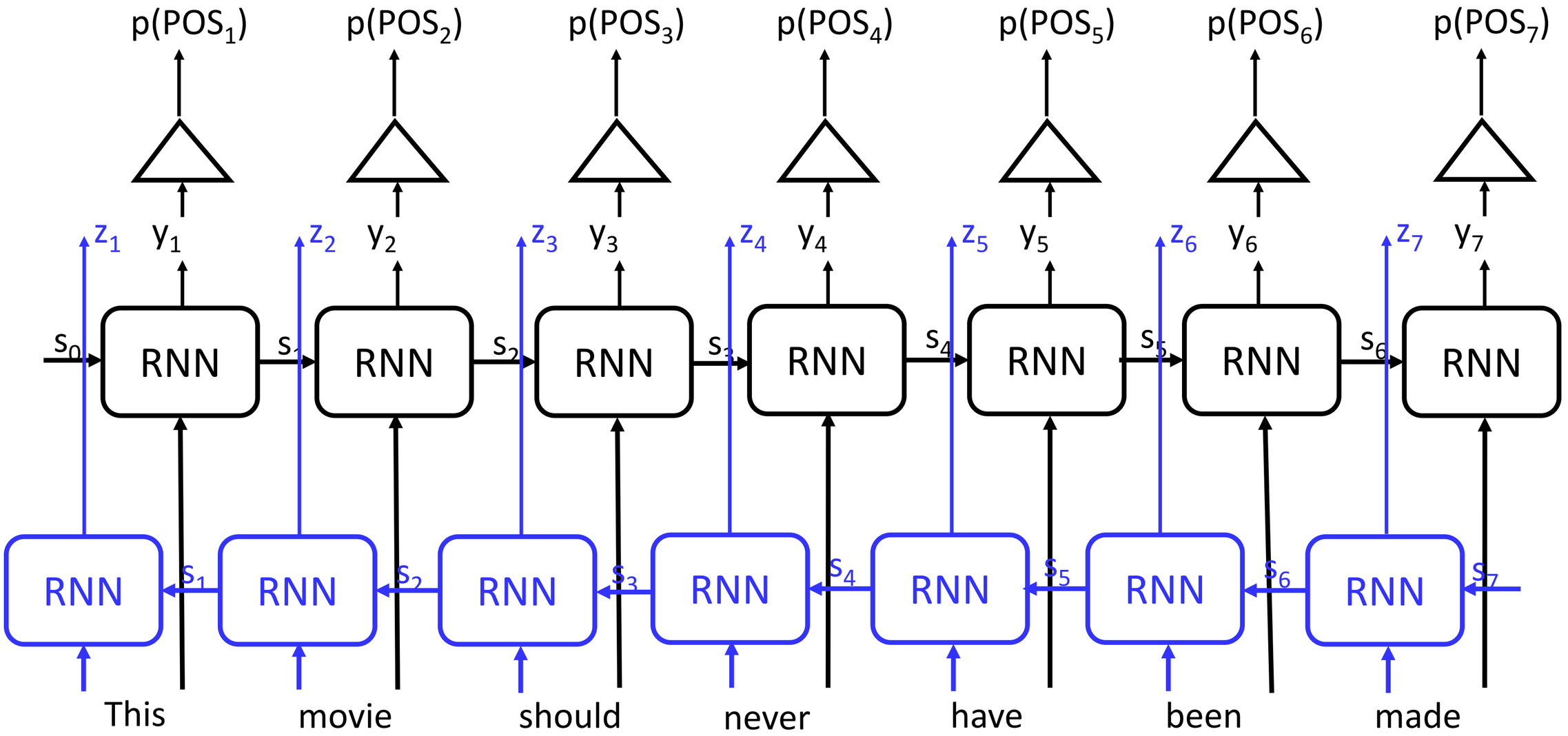
RNN Transducer for Sequence Labeling (POS Tagging)



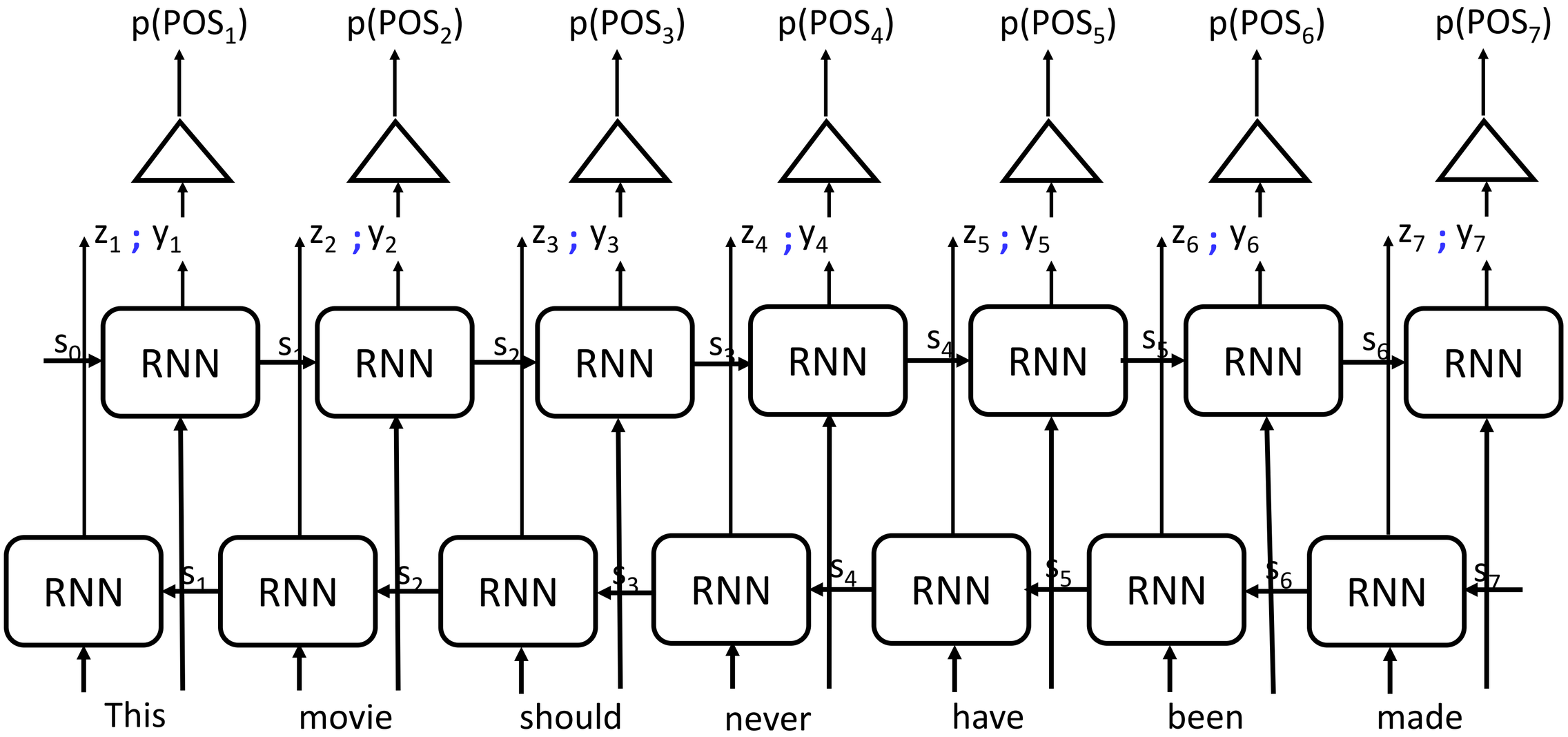
RNN → Bidirectional RNN

- An RNN s_t encodes all history $x_{1:t}$.
- But, future can also help in making a prediction
- Example: “the length is 6 hours” vs. “the length is 6 metres”
- A bidirectional RNN runs two unidirectional RNNs
- The final state encodes $x_{1:t}$ and $x_{t:T}$

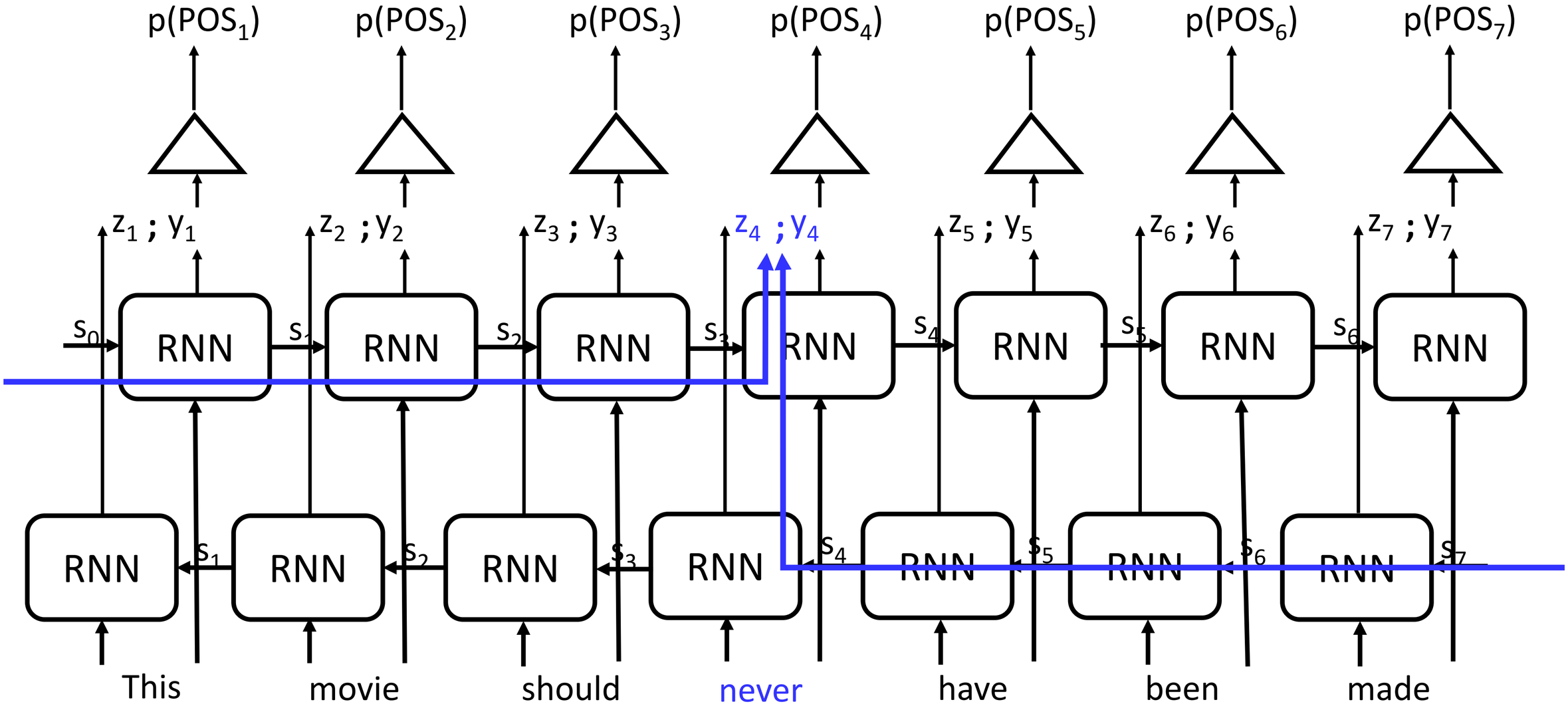
Bidirectional RNN



Bidirectional RNN

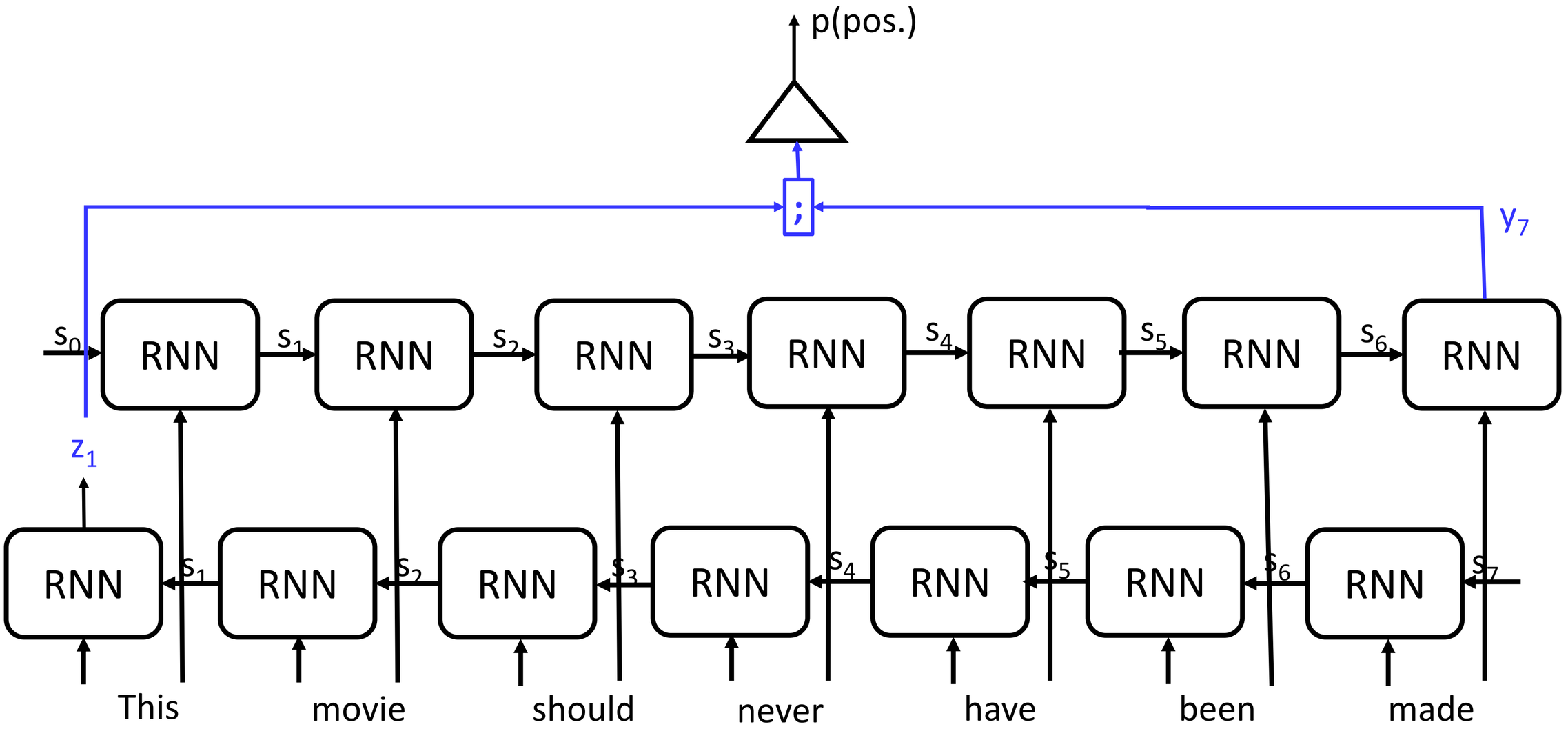


Bidirectional RNN



Infinite window around a word

Bidirectional RNN for Classification



Elman's RNN

- $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$
- $y_t = \tanh(W^y s_t + b^y)$

- Theorem: Any non-linear dynamical system can be approximated to any accuracy by an Elman's RNN, provided that the network has enough hidden units.

- Just because it can approximate it, doesn't mean it knows how to!
  - In practice: Elman's RNN is very hard to train
  - This is because of vanishing/exploding gradients!

$$\frac{\partial L}{\partial W^s} = \sum_{k=1}^{t} \left(\frac{\partial L}{\partial s_T} \frac{\partial s_k}{\partial W^s} \prod_{i=k+1}^{T} \frac{\partial \mathrm{R}(s_{i-1}, x_i)}{\partial d_i} W^s \right)$$

Next Class

- RNN → LSTM to fix the vanishing gradient problem!

Thank You

IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

Introduction to Deep Learning

Natural Language Processing

LSTM Encoder & Decoder

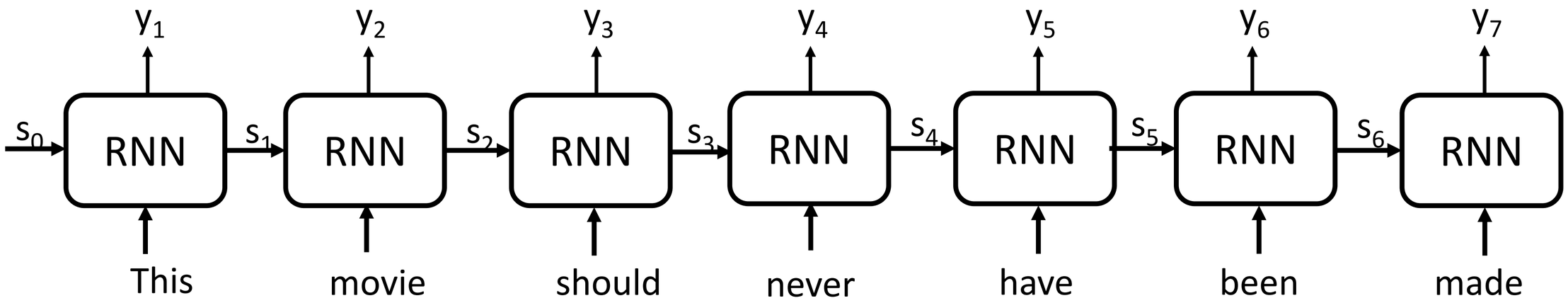
Mausam
IIT Delhi

(some slides adapted from Yoav Goldberg)

National Supercomputing Mission

Centre for Development of Advanced Computing

LSTMs


y_1
y_2
y_3
y_4
y_5
y_6
y_7
s_0
RNN
s_1
RNN
s_2
RNN
s_3
RNN
s_4
RNN
s_5
RNN
s_6
RNN
This
movie
should
never
have
been
made

Reminder: Elman's RNN

- $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$
- $y_t = \tanh(W^y s_t + b^y)$

- Theorem: Any non-linear dynamical system can be approximated to any accuracy by an Elman's RNN, provided that the network has enough hidden units.

- Just because it can approximate it, doesn't mean it knows how to!
  - In practice: Elman's RNN is very hard to train
  - This is because of vanishing/exploding gradients!

$$\frac{\partial L}{\partial W^s} = \sum_{k=1}^{t} \left(\frac{\partial L}{\partial s_T} \frac{\partial s_k}{\partial W^s} \prod_{i=k+1}^{T} \frac{\partial \mathrm{R}(s_{i-1}, x_i)}{\partial d_i} W^s \right)$$

A Memory View of Elman's RNN

- $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$
- $y_t = \tanh(W^y s_t + b^y)$

- Think of RNN as a computer. Input (x_t) arrives. Memory s gets updated

- In Elman RNN entire memory is rewritten at every time step!
  - There is no inertia!

- Memory predicts the output PLUS maintains the history
  - Ideally those two calculations should be separated.

Building Towards LSTM

- Main Idea: control the reading and writing of memory

$$s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$$

write (current) memory read (previous) memory write (current) input

We'd like to:

- Selectively read from some memory "cells".
- Selectively write to some memory "cells".
- Selectively write from the “input”.

Vector of Gates

- Read/write selectivity

(element-wise multiplication)
Hadamard product

$$\begin{bmatrix} 5 \\ -2 \\ 12 \\ 4.2 \\ -7 \\ 11 \end{bmatrix} \odot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \\ 4.2 \\ -7 \\ 0 \end{bmatrix}$$

vector of values

gate controls access

$$s_{t-1} \odot \mathbf{g}$$

$$s_{t-1} \in \mathbb{R}^{dstate}$$

$$g \in \{0,1\}^{dstate}$$

Gating to Control Access in an LSTM

- Main Idea: control the reading and writing of memory

$$s_t = s_{t-1} \odot f + x_t \odot i$$

$$f \in \{0,1\}^{dstate}$$
$$i \in \{0,1\}^{dstate}$$

forget gate
what to forget/remember?

input gate
what to write from the input?

Problem with 0-1 Gates

- They are fixed
- They don't depend on inputs or outputs
- We need to make them differentiable!

- Solution: make the gates "soft" and "input dependent"
- Instead of $f \in \{0,1\}^{dstate}$, use $f \in [0,1]^{dstate}$
- Moreover, compute $f = \sigma(W s_{t-1} + W' x_t + b)$

sigmoid → number between 0 and 1

dependent on state & input

Differentiable Gating to Control Access in an LSTM

- Main Idea: control the reading and writing of memory

$$s_t = s_{t-1} \odot f_t + x_t \odot i_t$$

$$f_t \in [0,1]^{dstate}$$
$$i_t \in [0,1]^{dstate}$$

time-dependent soft forget gate

time-dependent soft input gate

$$f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$$

Differentiable Gating to Control Access in an LSTM

- Not a good idea adding input to state

$$\cancel{s_t = s_{t-1} \odot f_t + x_t \odot i_t}$$

$$f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$$

$$s_t = s_{t-1} \odot f_t + \tilde{s}_t \odot i_t$$

$$\tilde{s}_t = \phi(s_{t-1}, x_t)$$

proposal for new state

From RNN to Prototype LSTM

- RNN: $s_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$

  $y_t = \tanh(W^y s_t + b^y)$

- Prototype LSTM:

  $s_t = s_{t-1} \odot f_t + \tilde{s}_t \odot i_t$

  $\tilde{s}_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$

  $f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$

  $i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$

Problem: same s_t will be used for output and maintaining state

Prototype LSTM → LSTM by Splitting the State

- Prototype LSTM:

$$\tilde{s}_t = \tanh(W^s s_{t-1} + W^x x_t + b^s)$$
$$s_t = s_{t-1} \odot f_t + \tilde{s}_t \odot i_t$$

$$f_t = \sigma\left(W^{sf} s_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} s_{t-1} + W^{xi} x_t + b^i\right)$$

s_t → c_t: internal/cell state

s_t → h_t: output state/state

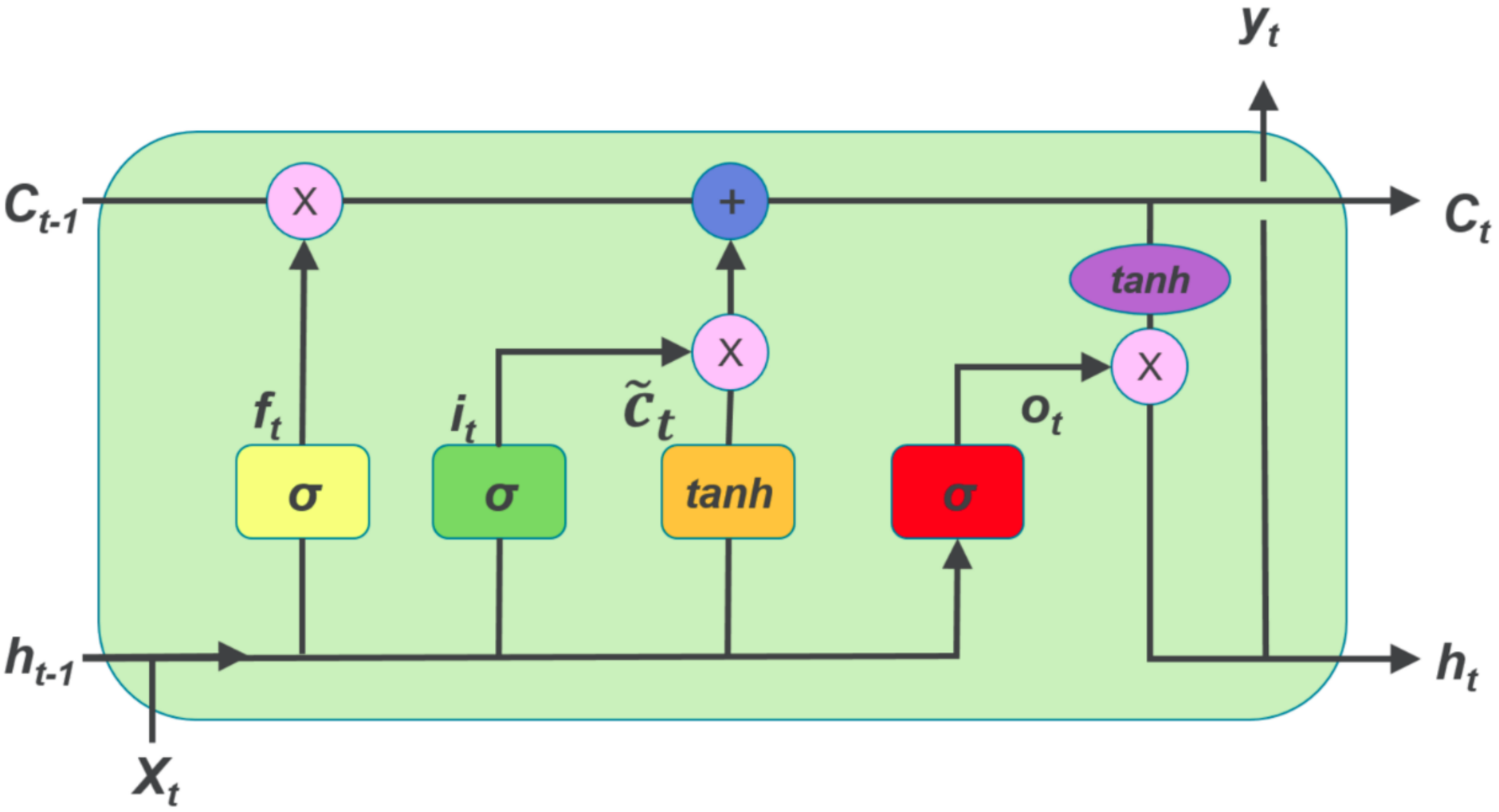
- LSTM:

$$\tilde{c}_t = \tanh(W^s h_{t-1} + W^x x_t + b^s)$$
$$c_t = c_{t-1} \odot f_t + \tilde{c}_t \odot i_t$$
$$h_t = \tanh(c_t) \odot o_t$$

$$f_t = \sigma\left(W^{sf} h_{t-1} + W^{xf} x_t + b^f\right)$$
$$i_t = \sigma\left(W^{si} h_{t-1} + W^{xi} x_t + b^i\right)$$
$$o_t = \sigma(W^{so} h_{t-1} + W^{xo} x_t + b^o)$$

Asssumption: information irrelevant for previous output is irrelevant for gate computation

LSTM



Less Problem of Vanishing Gradient

$$c_t = c_{t-1} \odot f_t + \tilde{c}_t \odot i_t$$

$$f_t = \sigma\left(W^{sf} h_{t-1} + W^{xf} x_t + b^f\right)$$

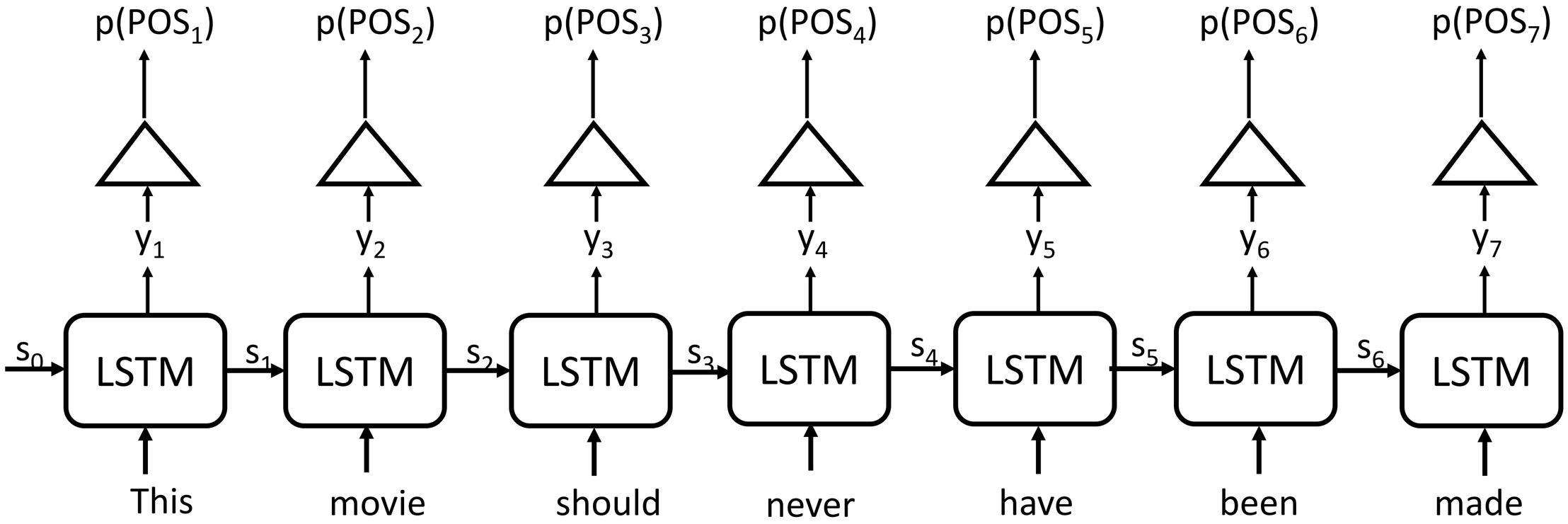
$$\frac{\partial c_t}{\partial c_{t-1}} = \frac{\partial f_t}{\partial c_{t-1}} c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} f_t + \frac{\partial i_t}{\partial c_{t-1}} \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} i_t$$

Initialize such that $f_t \rightarrow 1$

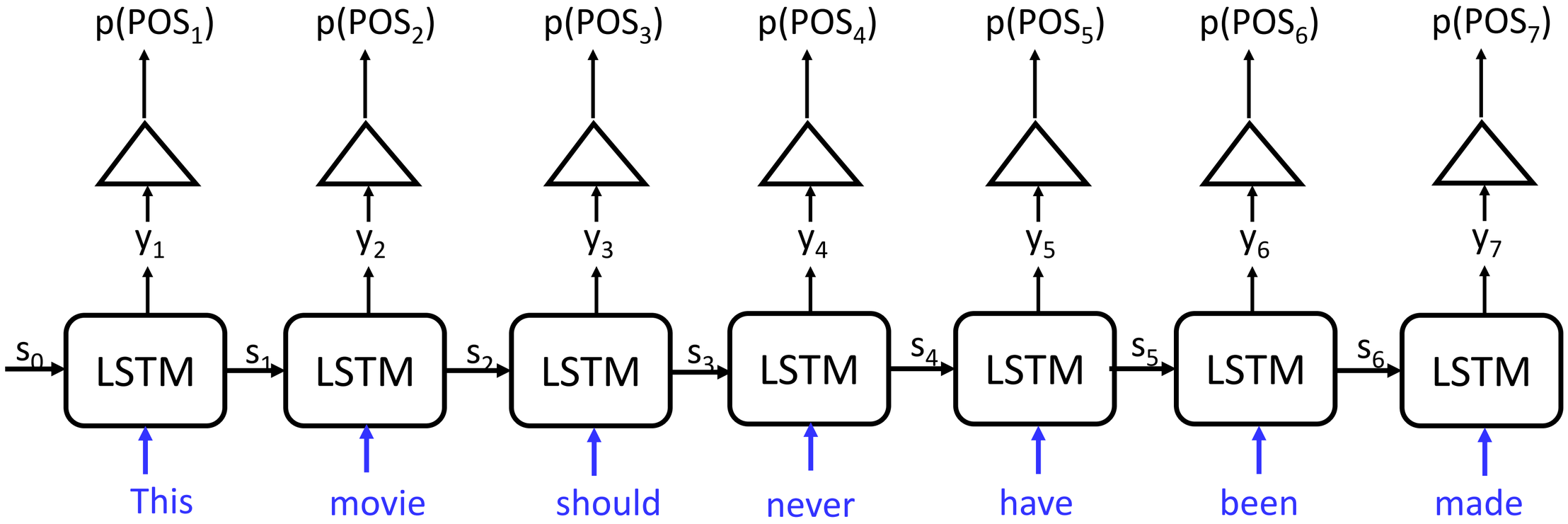
=> b^f = 1 or more

Sequence Decoder

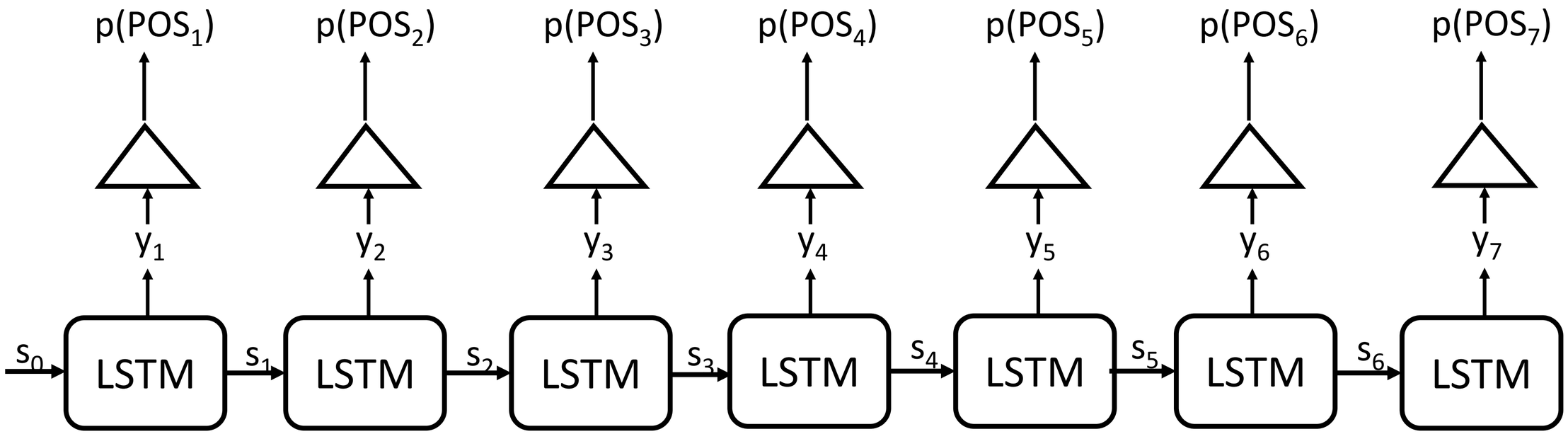
Neural Language Model



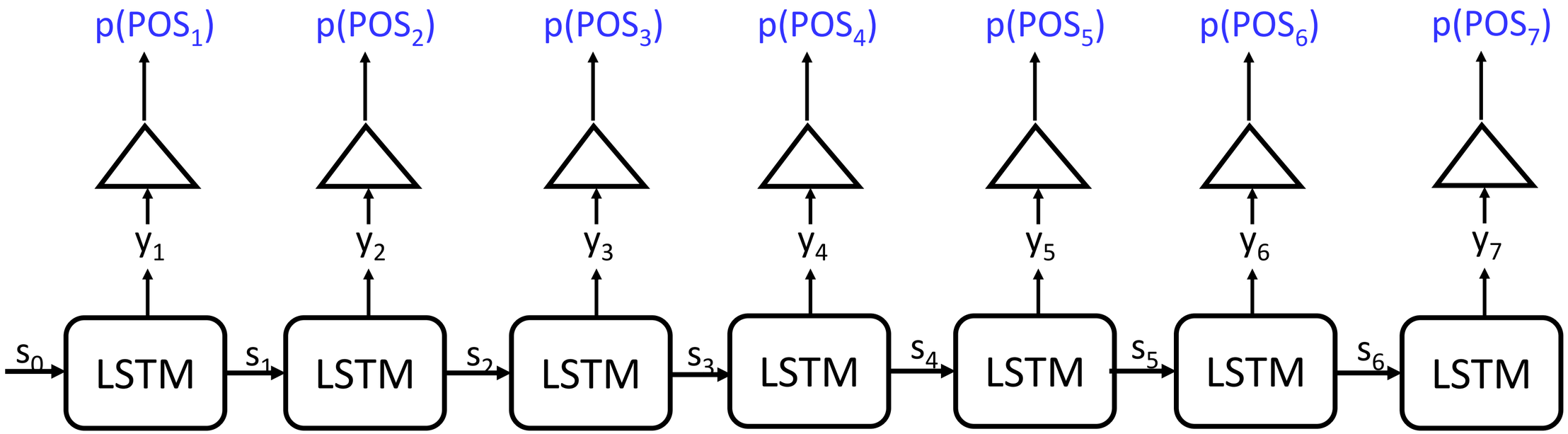
Neural Language Model



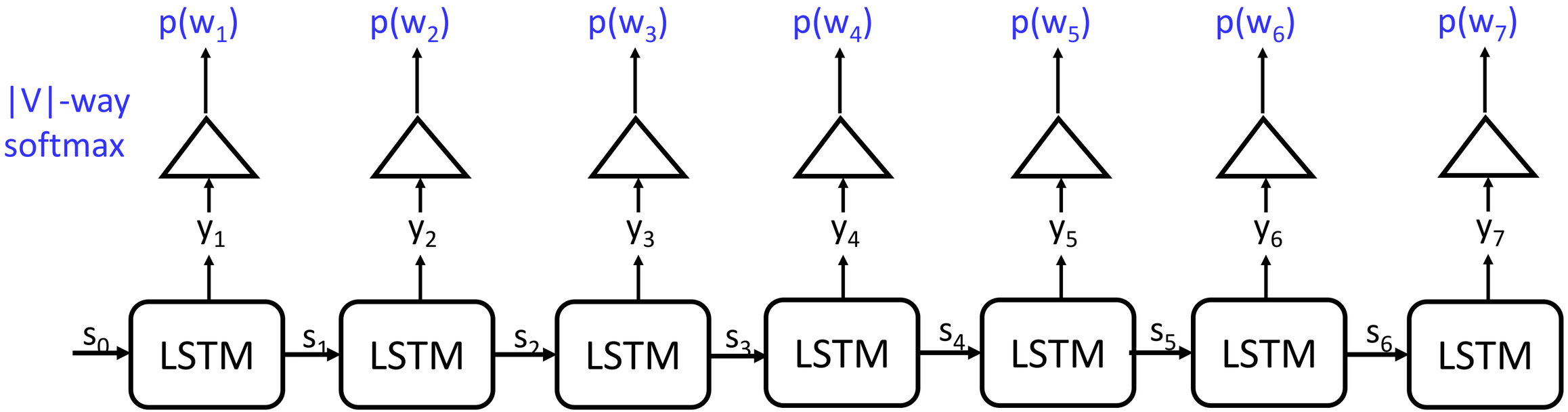
Neural Language Model



Neural Language Model

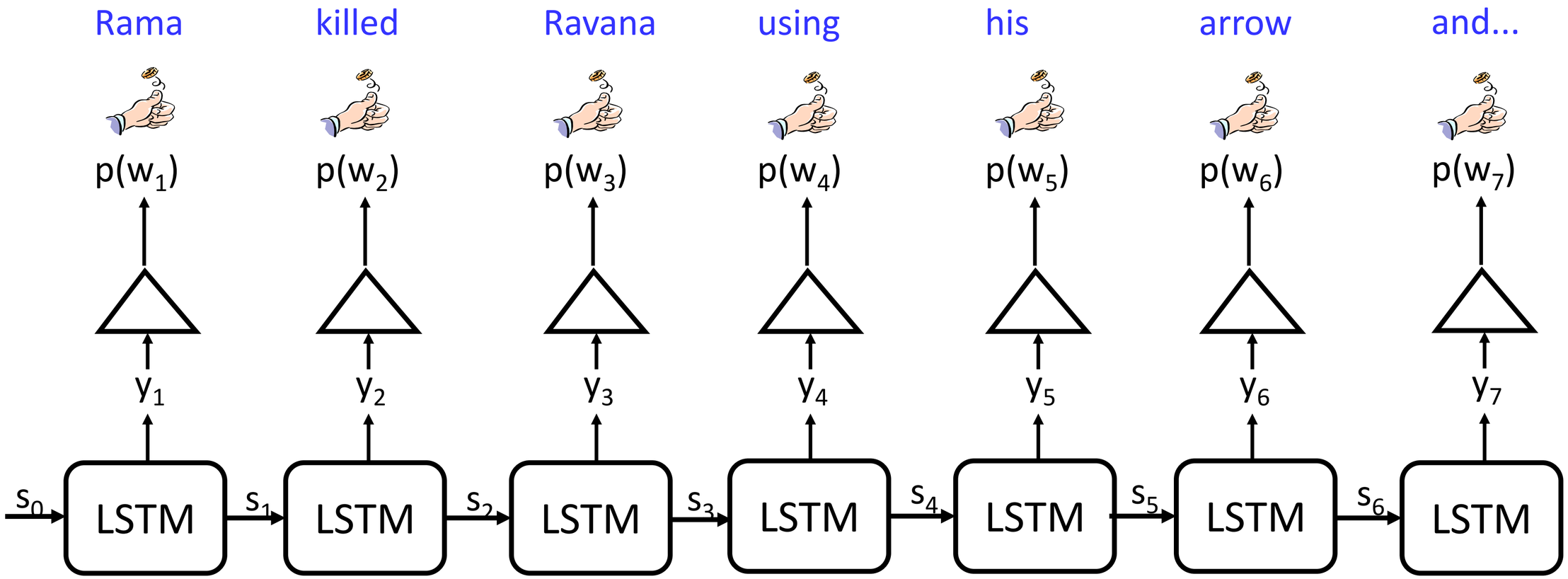


Neural Language Model



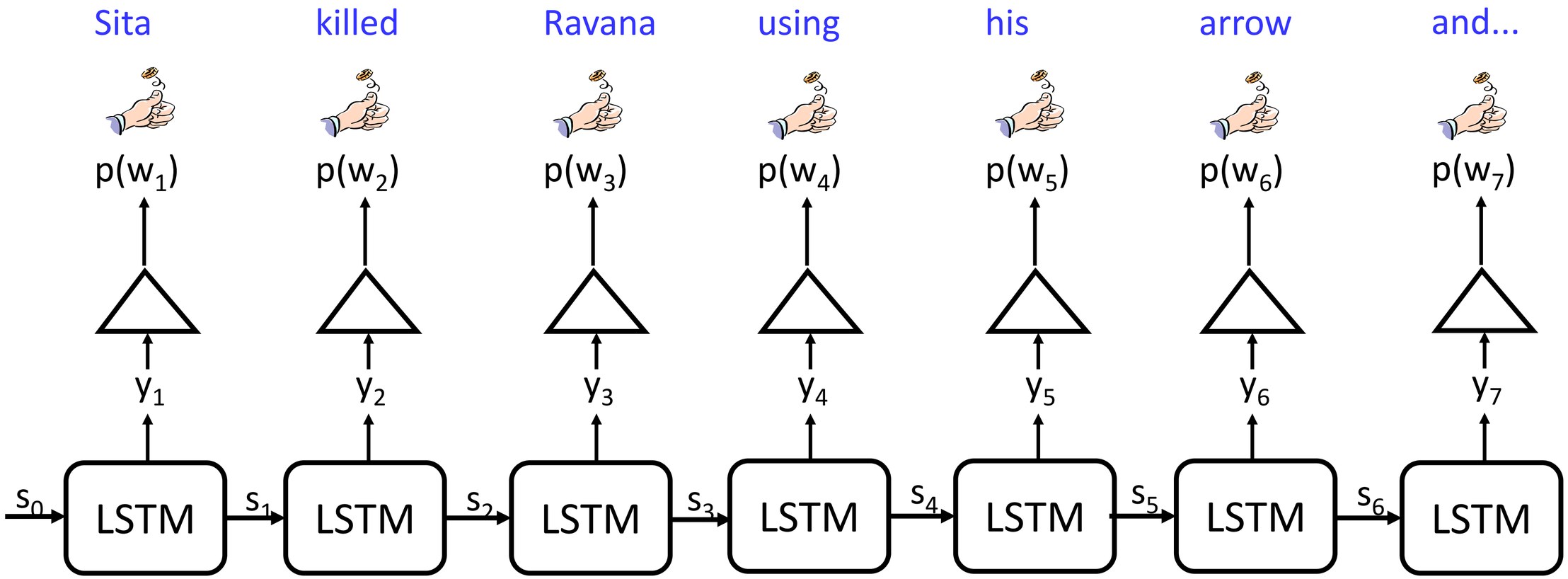
How do we get the actual sentence from a sequence of probability distributions?

Neural Language Model



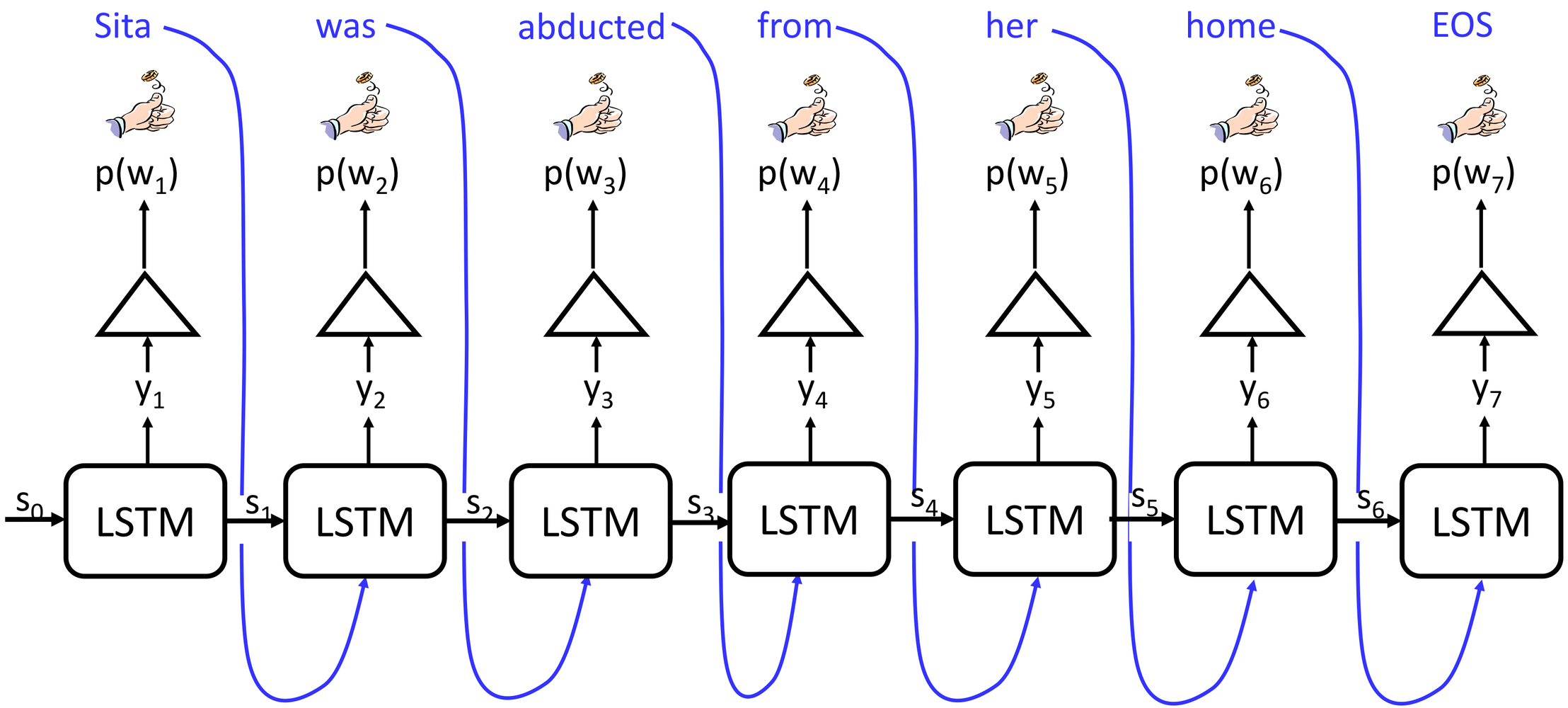
Can you think of a fundamental problem in this design?

Neural Language Model

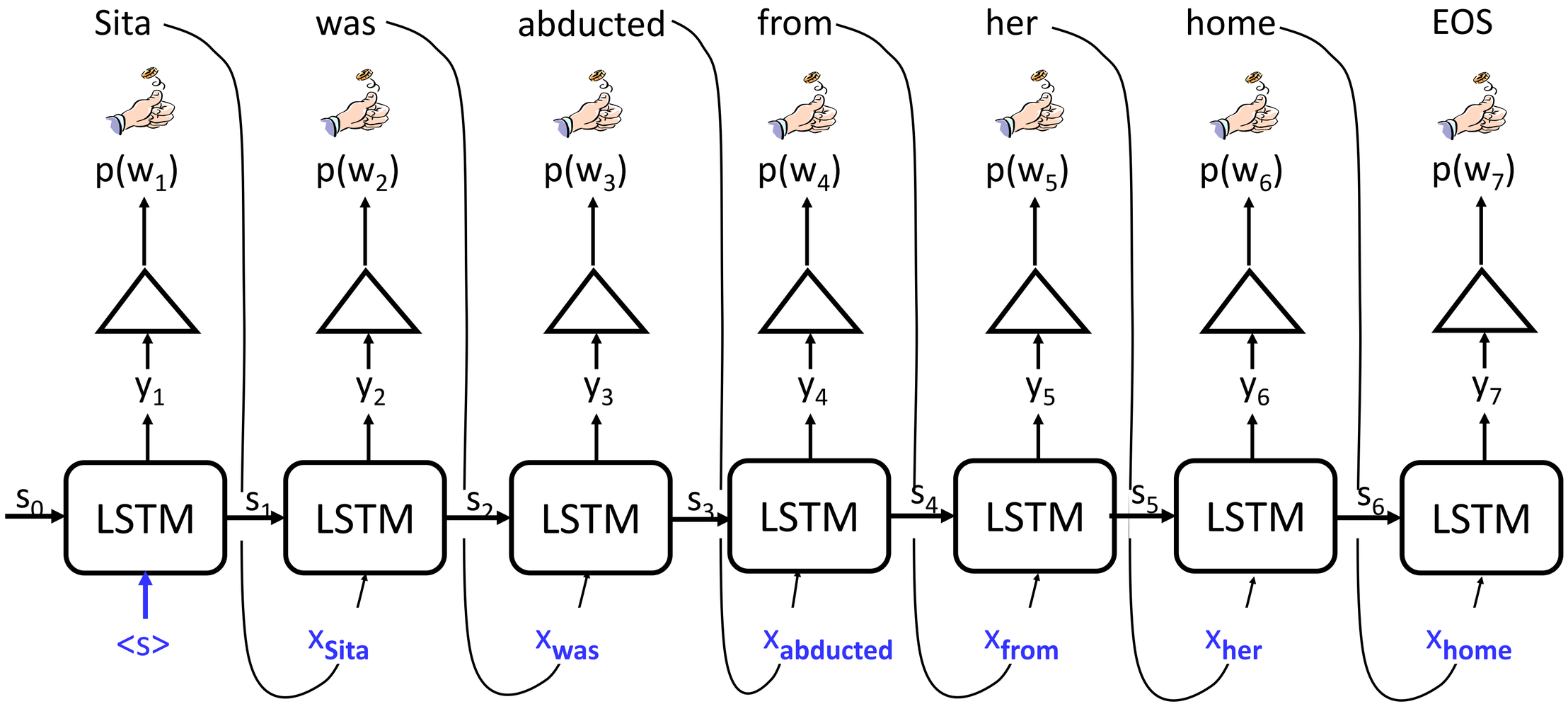


Can you think of a fundamental problem in this design?

Neural Language Model



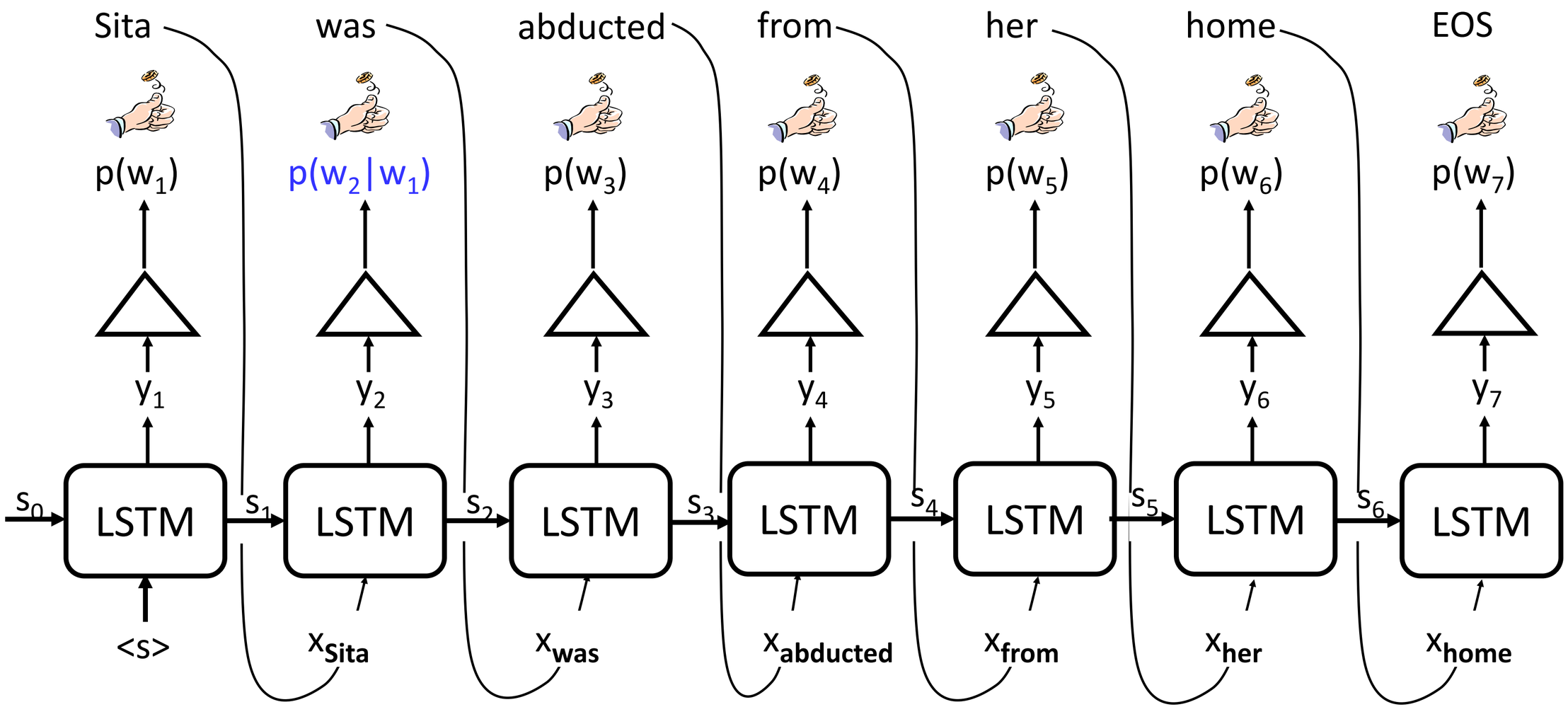
Neural Language Model



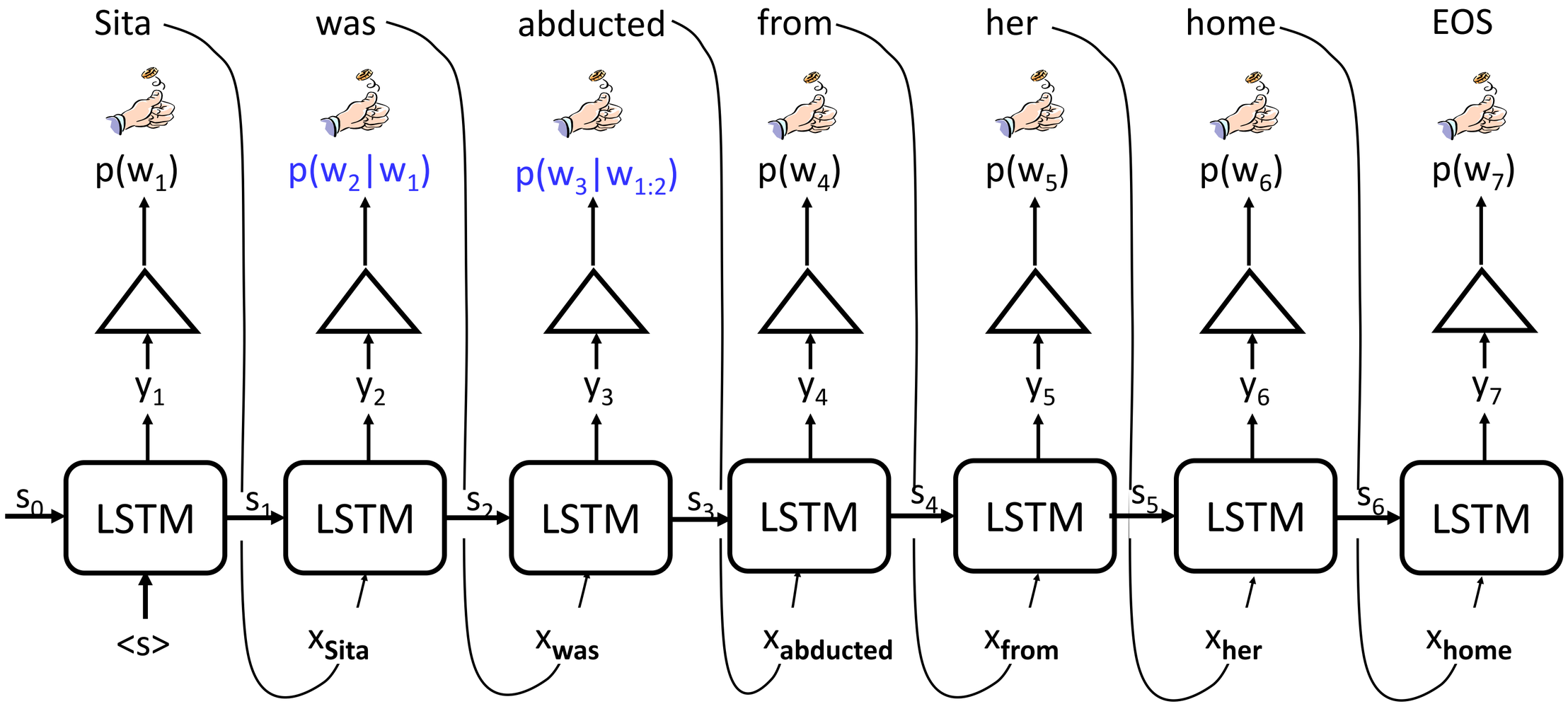
Language Models

- A Language Model is a way to assign probability to each sentence.
- $P(w_1 w_2 w_3 \ldots w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2) \ldots P(w_n|w_1 w_2 \ldots w_{n-1})$
- LSTM implements a language model…

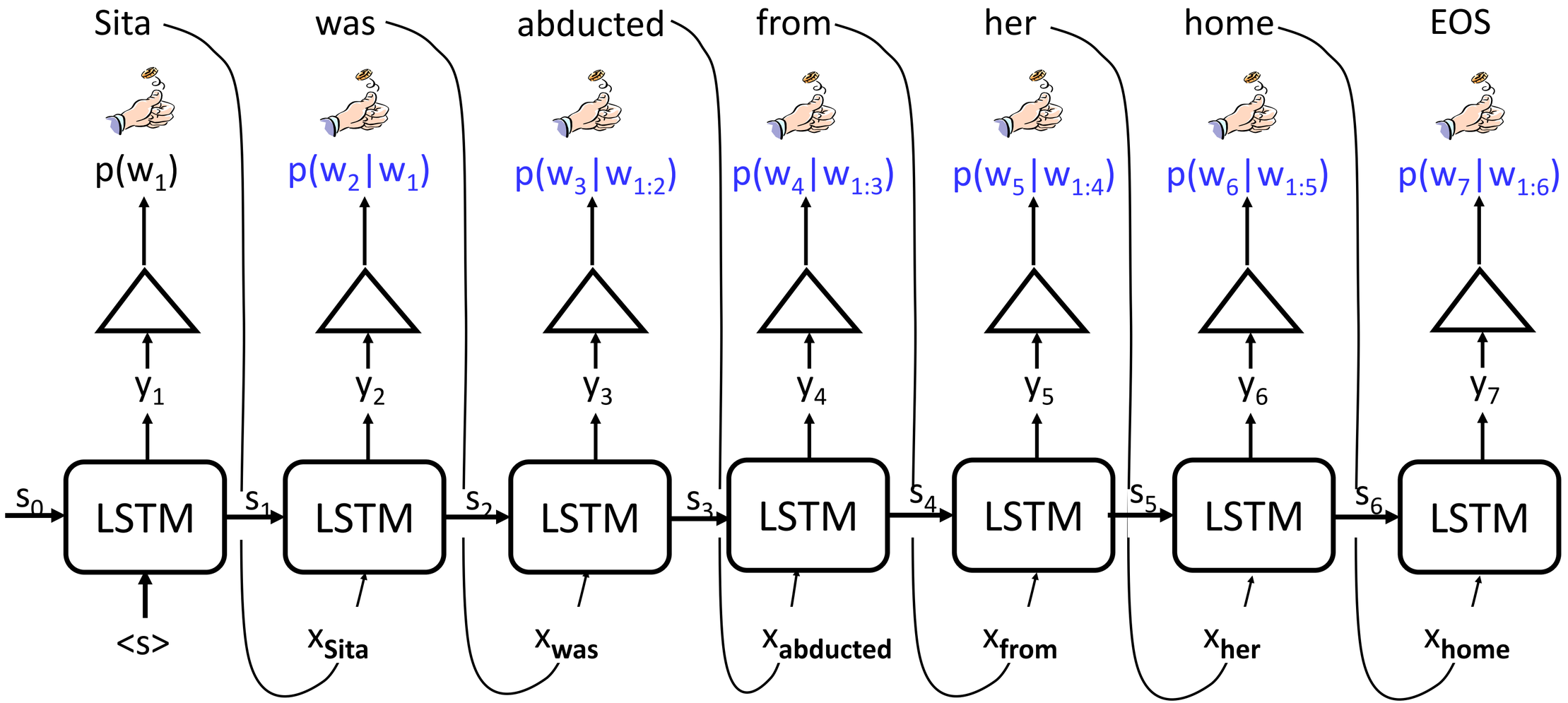
Neural Language Model



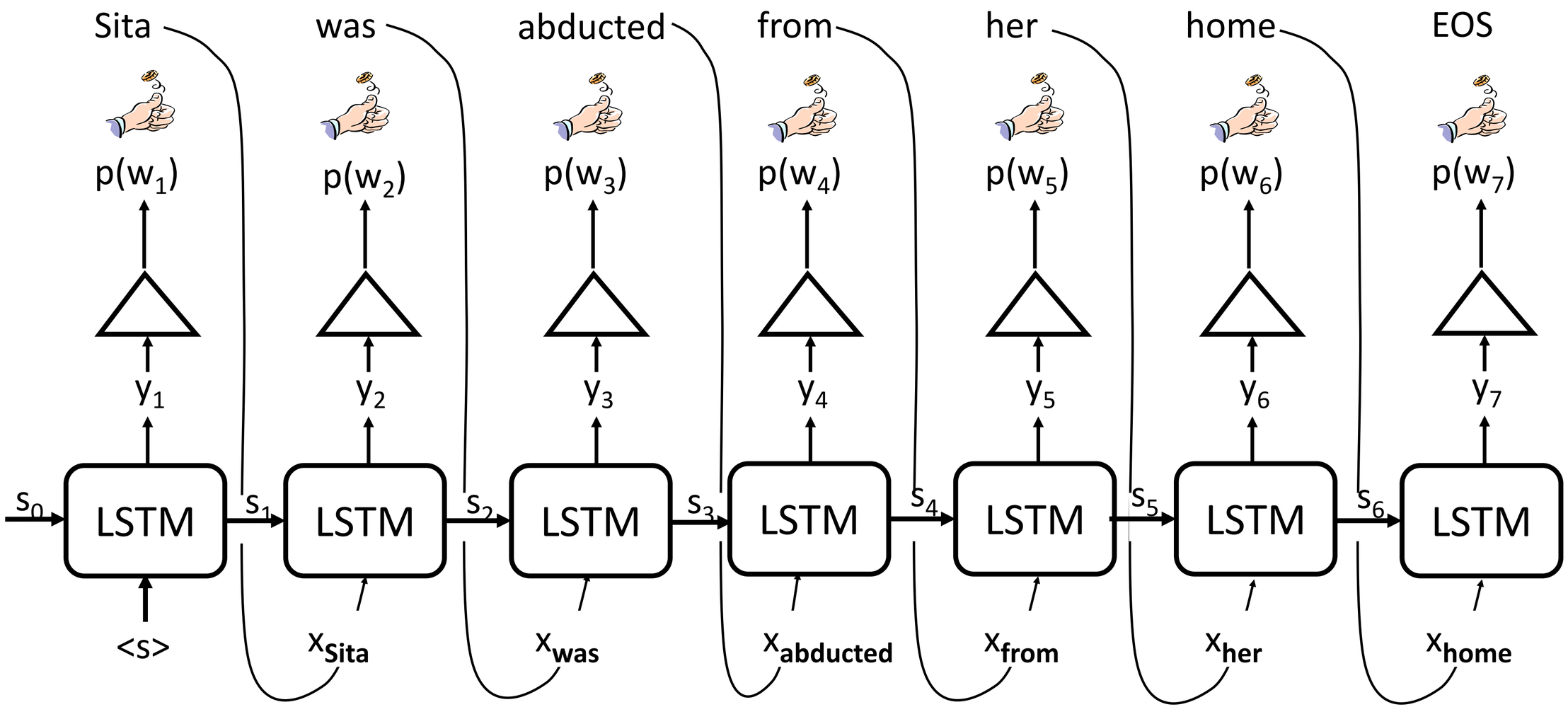
Neural Language Model



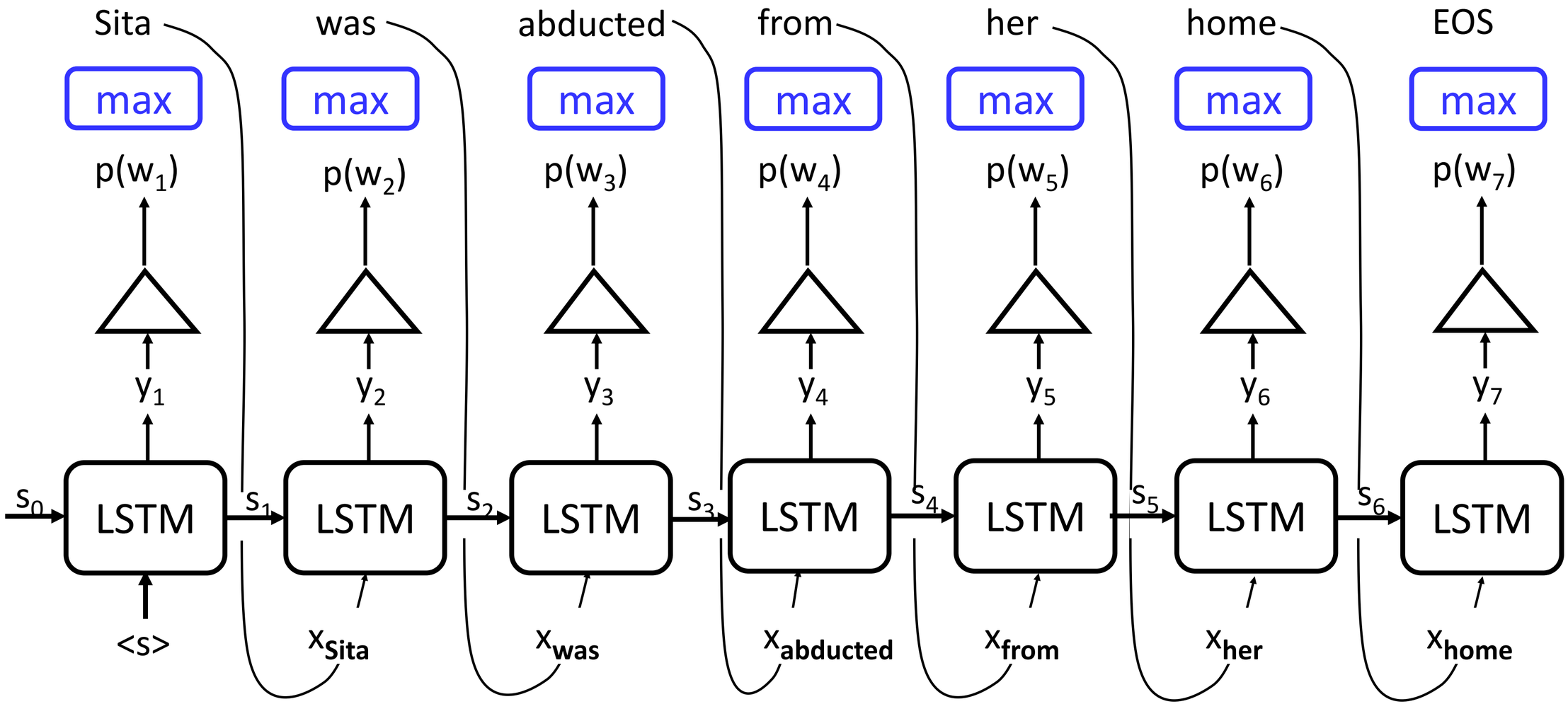
Neural Language Model



Neural Language Model: Inference Time



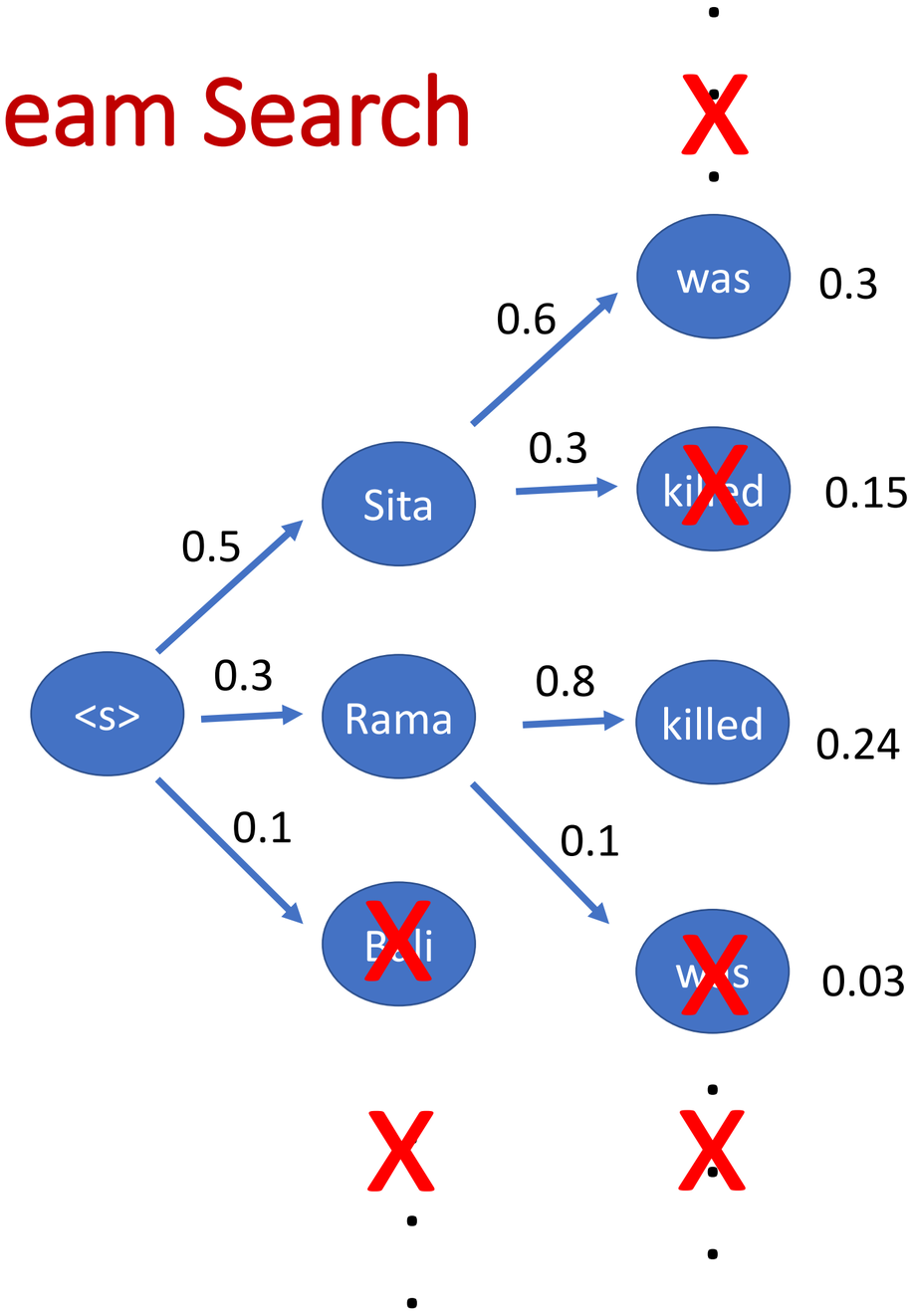
Neural Language Model: Inference Time



Neural Language Model: Greedy Decoding

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

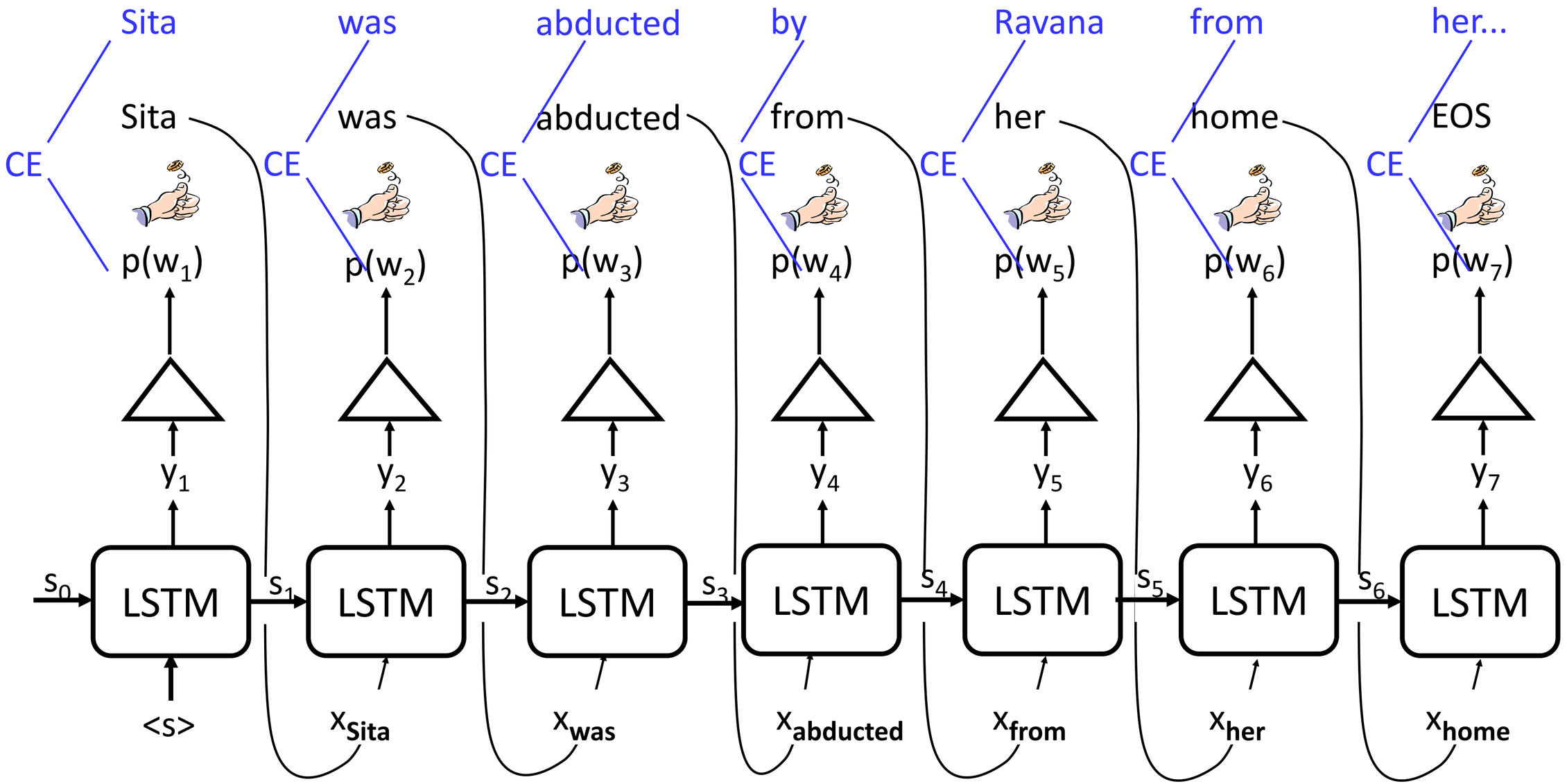


Instead of picking one greedy path, maintain multiple greedy paths

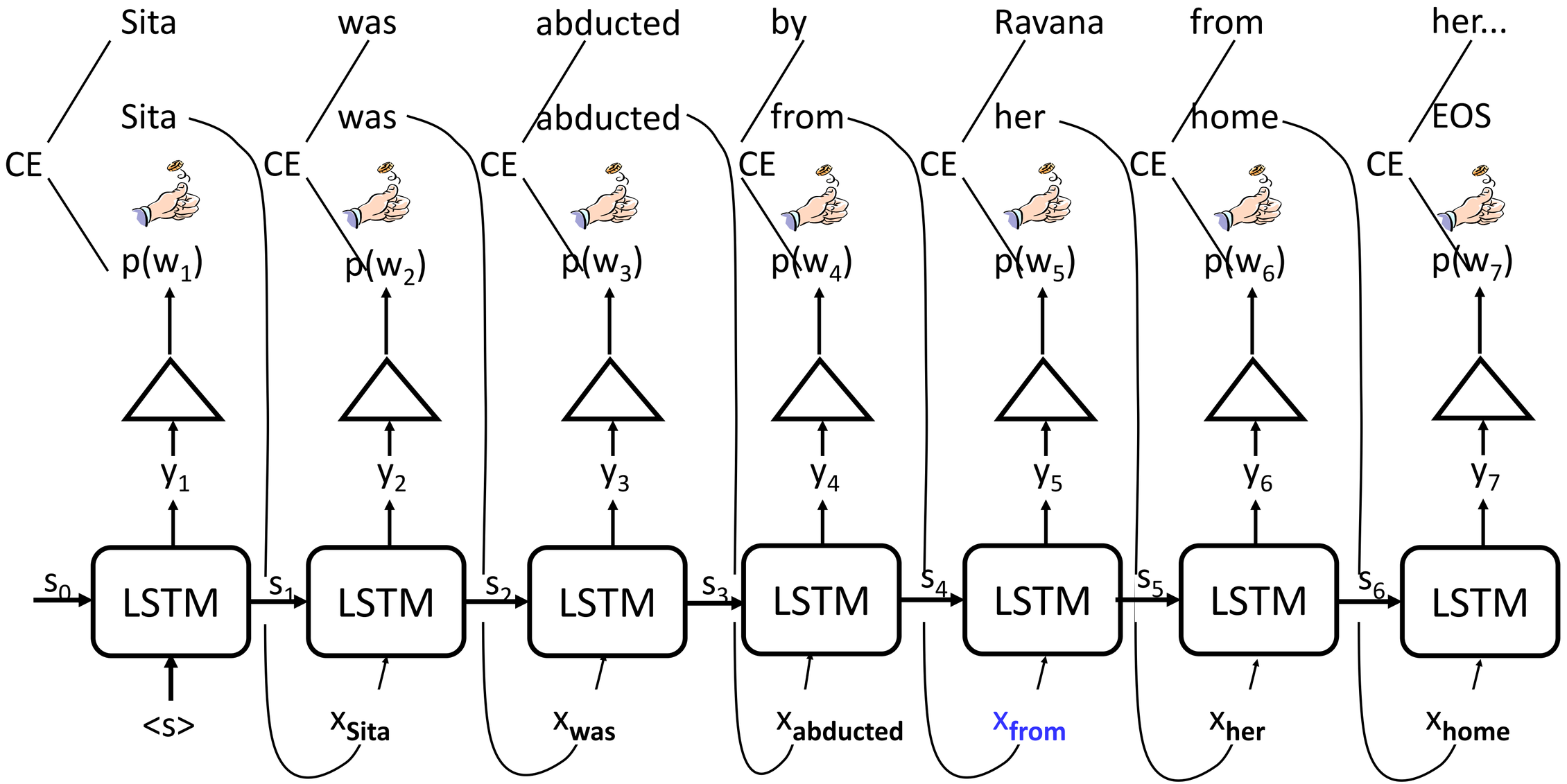
Upto a constant beam of b

Example for beam size =2

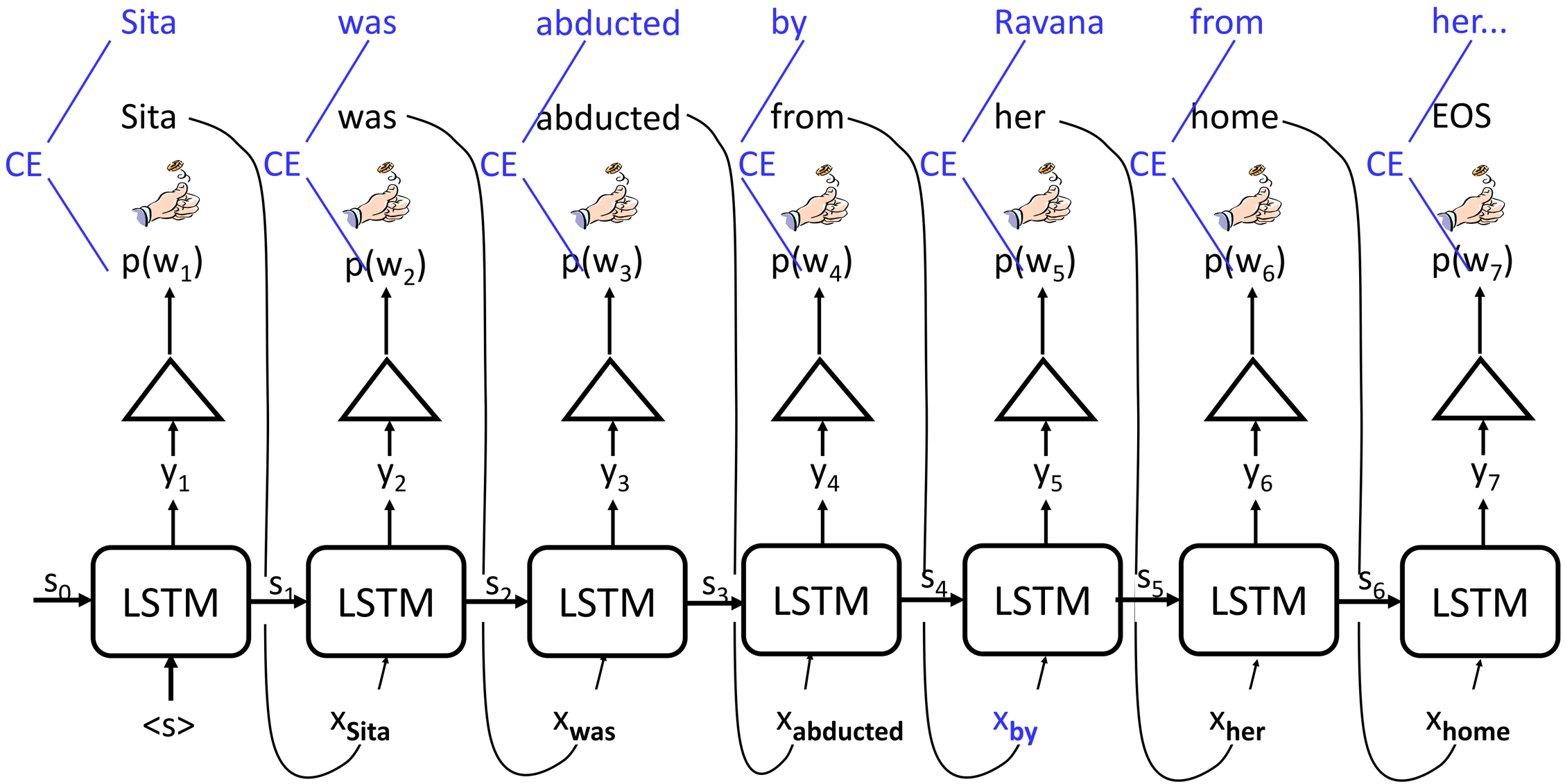
Neural Language Model: Training



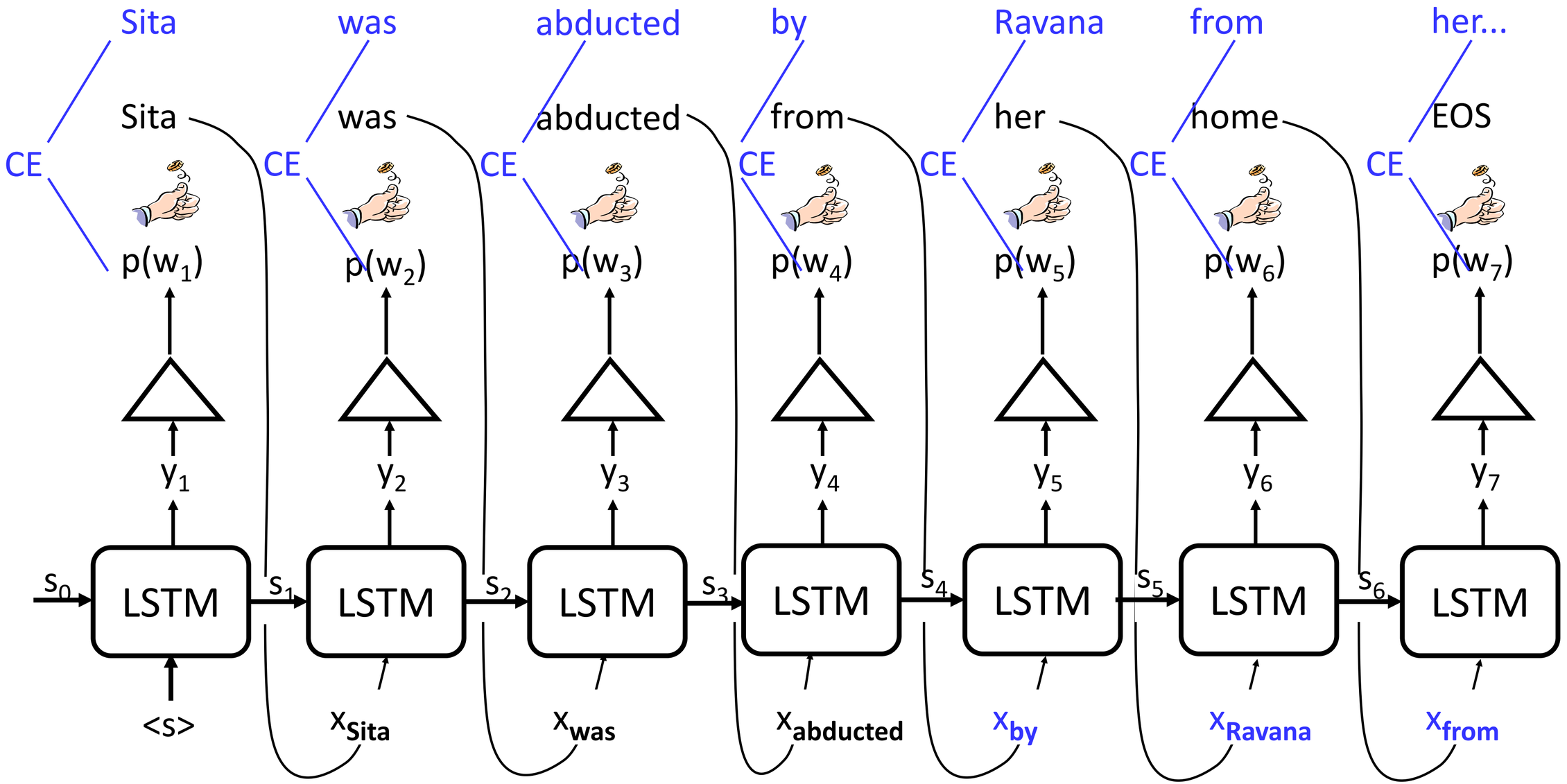
Neural Language Model: Training



Neural Language Model: Training (Teacher Forcing)



Neural Language Model: Training (Teacher Forcing)



Next Class

- Encoder Decoder Architectures
- Attention Models

Thank You

IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

Introduction to Deep Learning

Natural Language Processing

Seq2Seq & Transformers

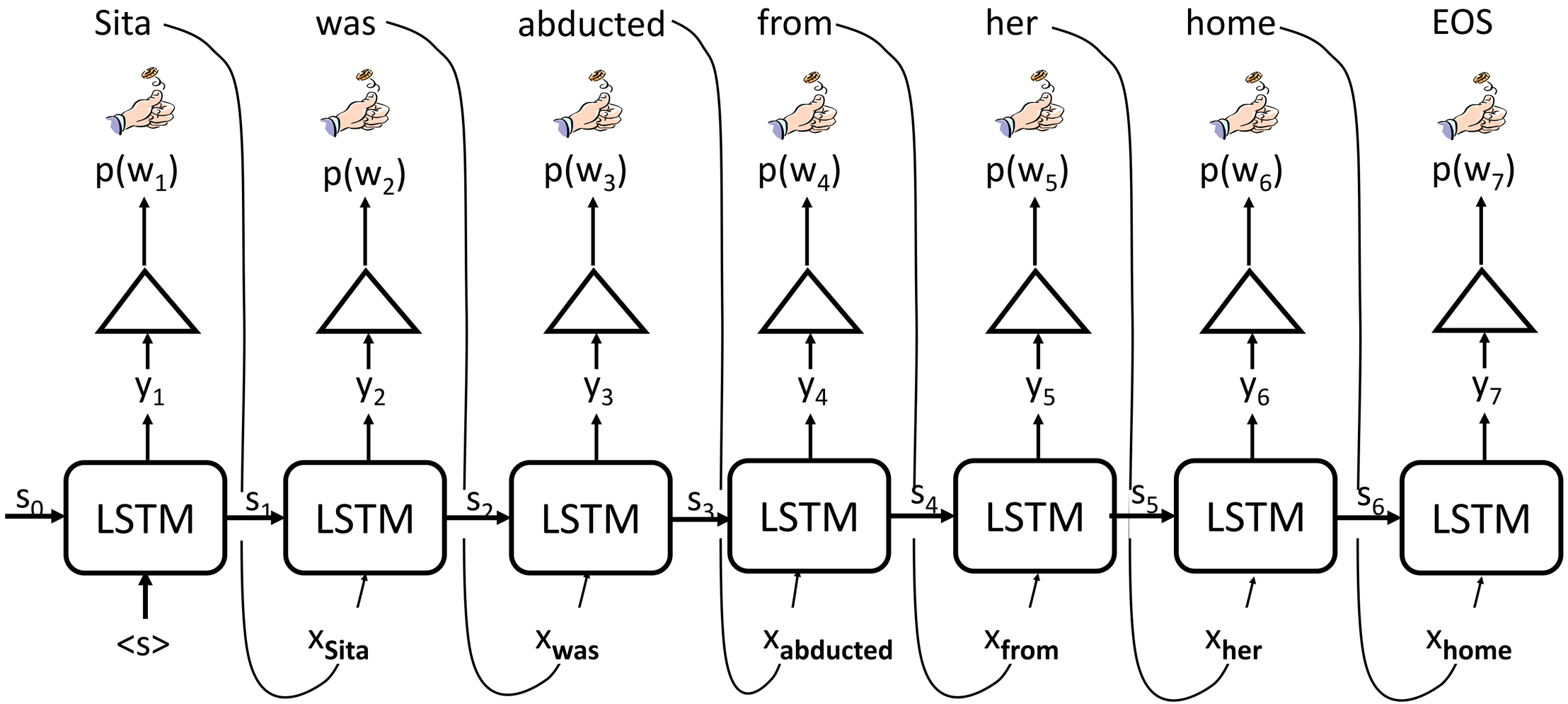
Mausam
IIT Delhi

(some slides adapted from Yoav Goldberg
some figures taken from Jay Alammar's blog)

National Supercomputing Mission

Centre for Development of Advanced Computing

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

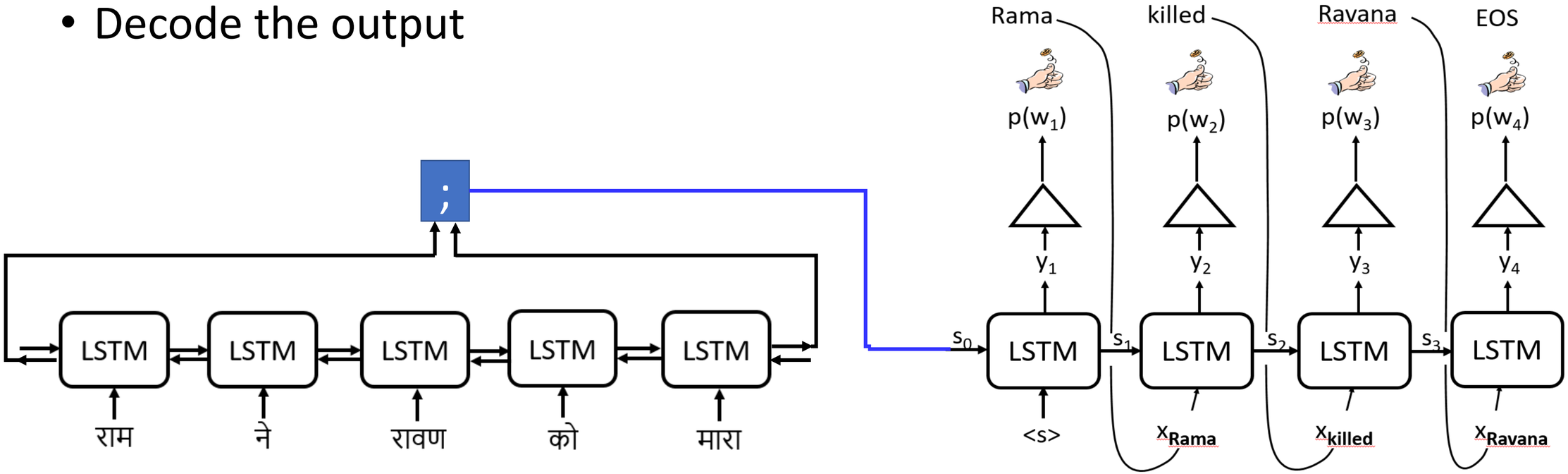
- Generate text based on (varied) inputs

- 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

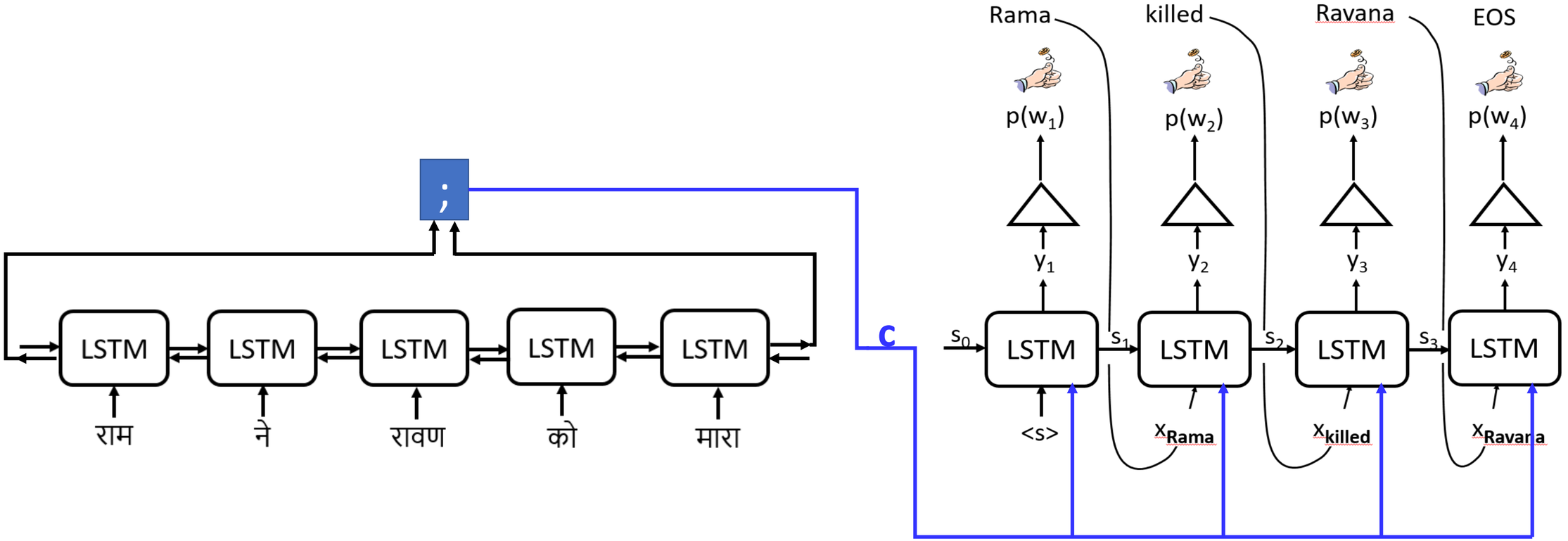


Encoder
(Bi-LSTM)

Decoder
(LSTM)

Idea 2: Encoder-Decoder

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



Attention

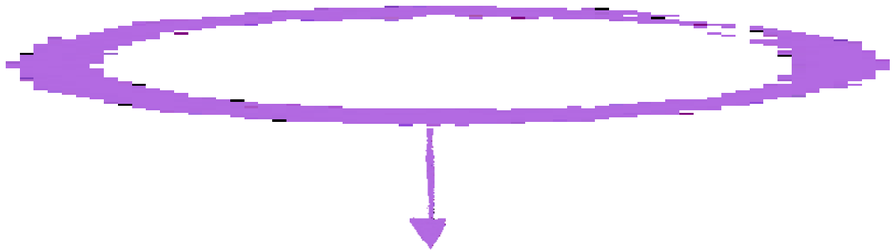
Sentence Representation

*You can't cram the meaning of the whole *%#@ing sentence in a single *%#@ing vector.*

- Encoding a single vector is too restrictive.
  Instead of producing a single vector for the sentence, produce one vector for each word.
- But, eventually need 1 vector.
  Multiple vectors → Single vector
  Sum/Avg operators give equal importance to each input
- We dynamically decide which input is more/less important for a task.
- Create a weighted sum to reflect this variation: Attention
- query (q): decides importance of each input
  attention weights (α_i): normalized importance of input
  unnormalized attention weights ($\bar{\alpha}_i$): intermediate step to compute α_i
  attended summary: weighted avg of input with α weights

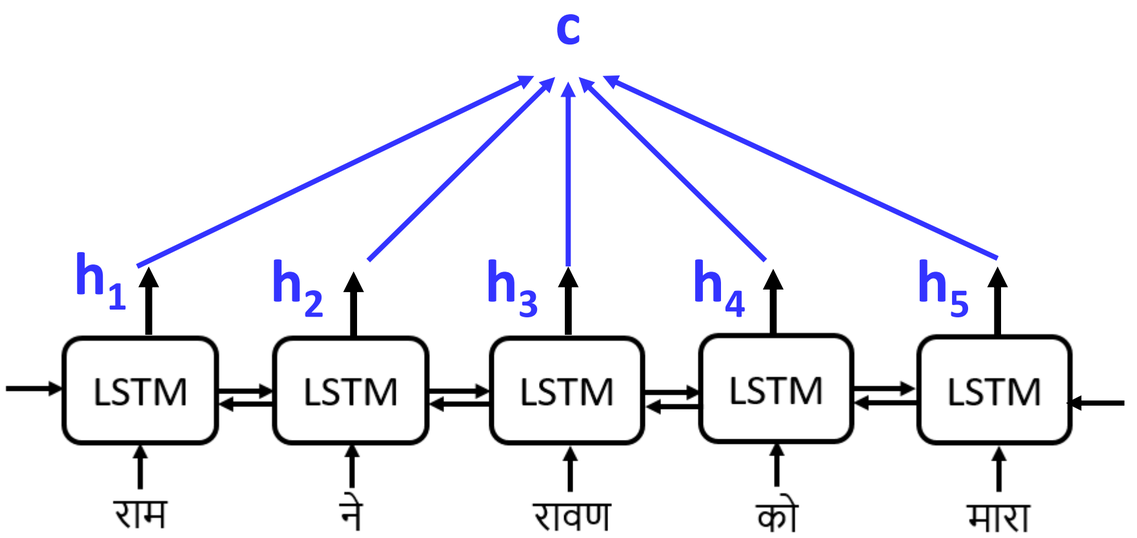
Multiple Encoded Vectors → Single Summary

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



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

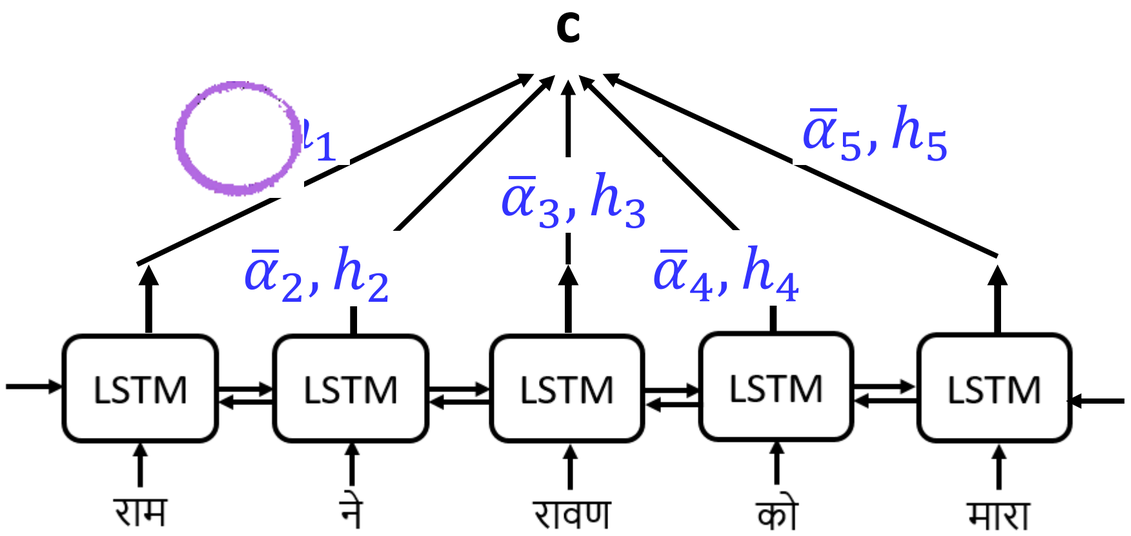
Need to convert h_is to c



Multiple Encoded Vectors → Single Summary

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

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

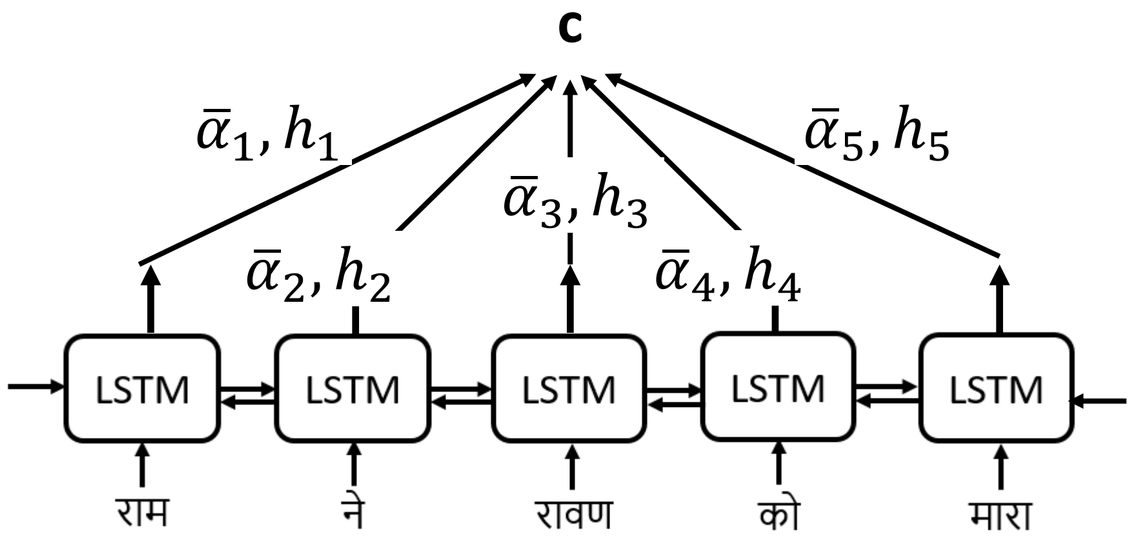


Multiple Encoded Vectors → Single Summary

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

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

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



Attention: Encoding

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

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

$$\alpha = \text{softmax}(\bar{\alpha}_1, \ldots, \bar{\alpha}_T)$$

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

what is ϕ^{att}?

what is q?

Attention Functions ϕ^{att}

- Bahadanau Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{u}.\,\mathrm{g}(\mathrm{Wq} + \mathrm{W}'\mathrm{h} + \mathrm{b})$
- Luong Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{q}.\,\mathrm{h}$
- Scaled Dot Product Attention: $\phi^{\mathrm{att}}(q,h) = \frac{\mathrm{q}.\mathrm{h}}{\sqrt{d}}$
- Bilinear Attention: $\phi^{\mathrm{att}}(q,h) = \mathrm{hWq}$

Attention: Encoding

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

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

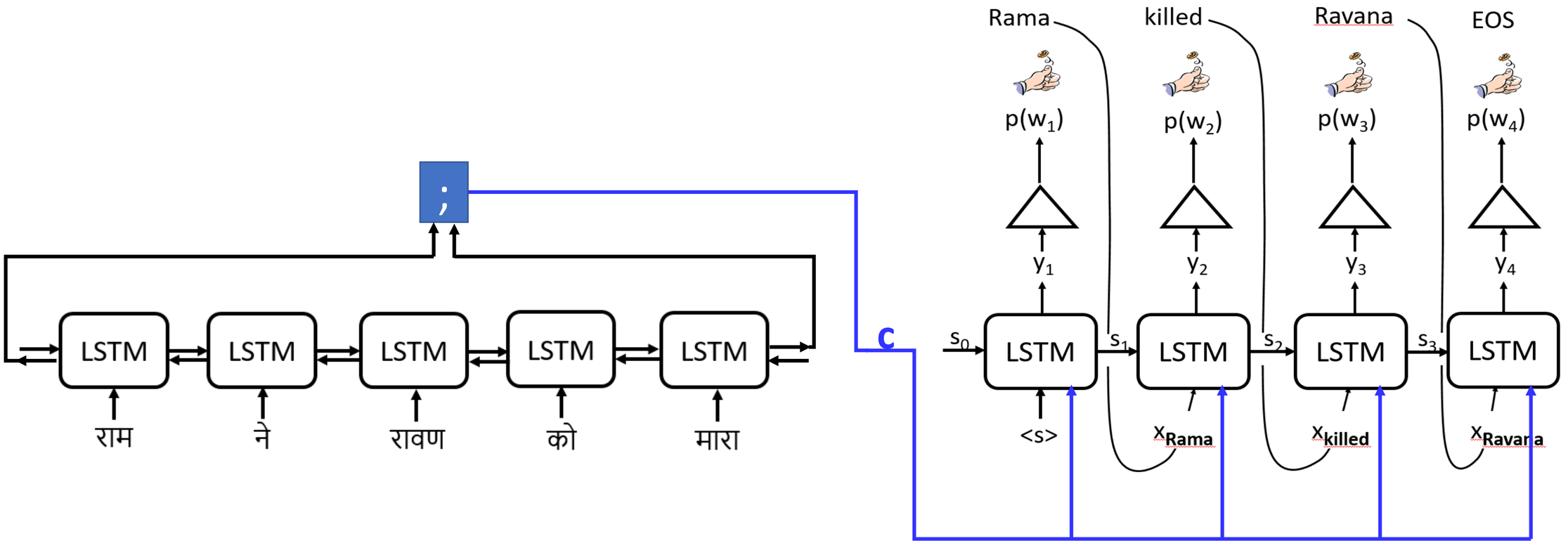
$$\alpha = \text{softmax}(\bar{\alpha}_1, \ldots, \bar{\alpha}_T)$$

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

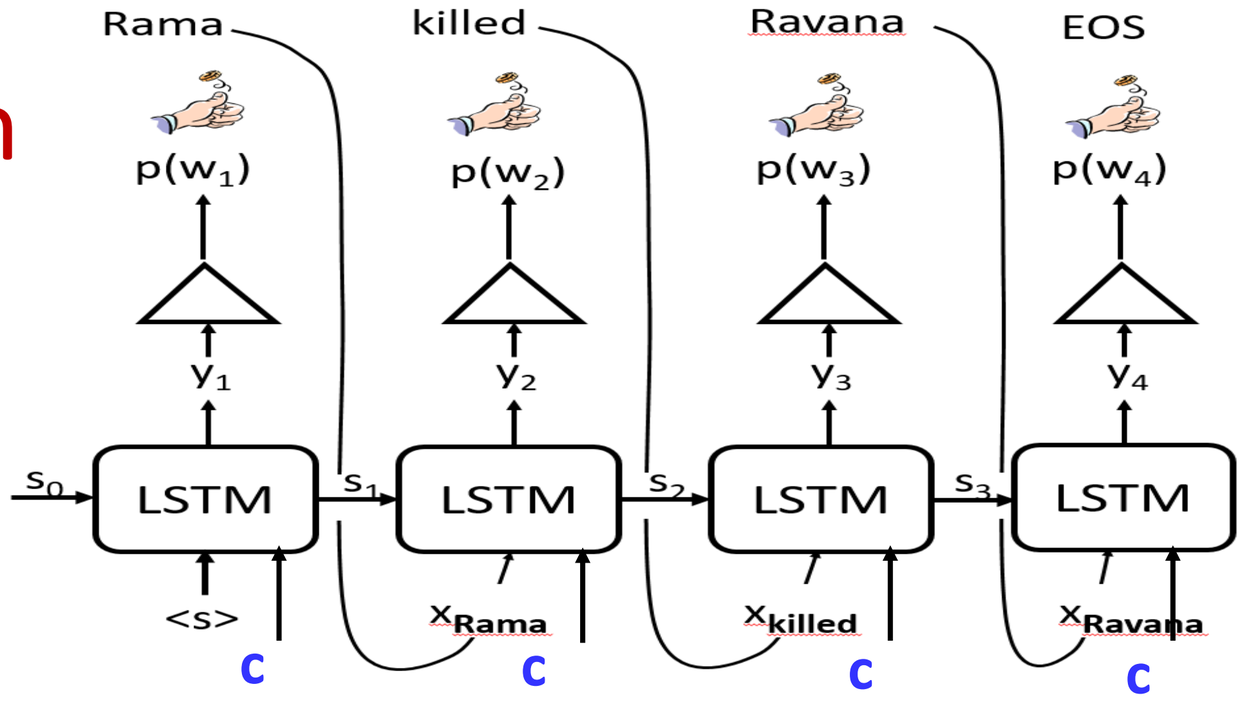
what is q?

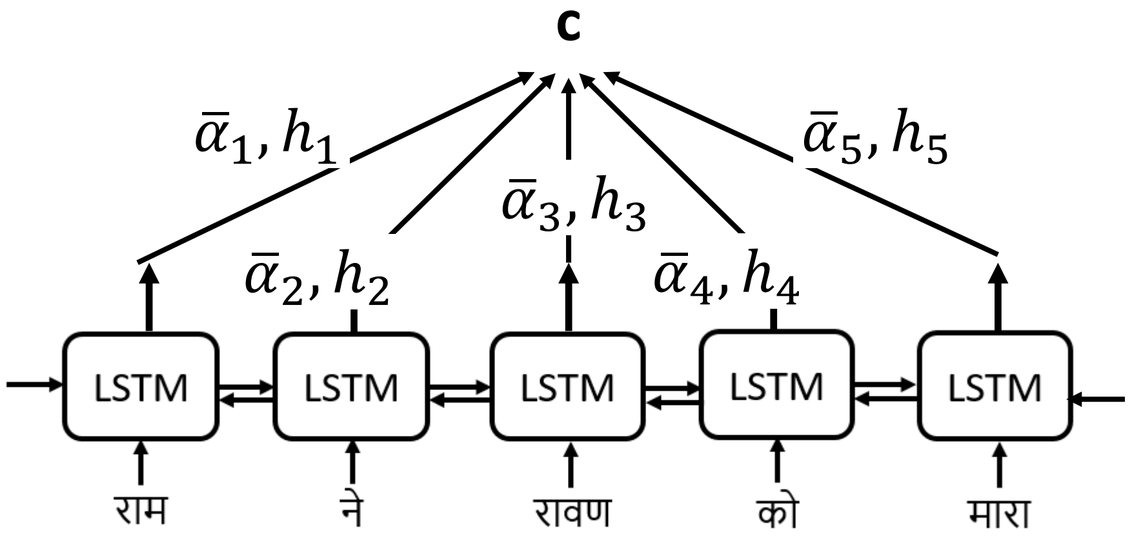
Seq2Seq with Attention

Seq2Seq without Attention

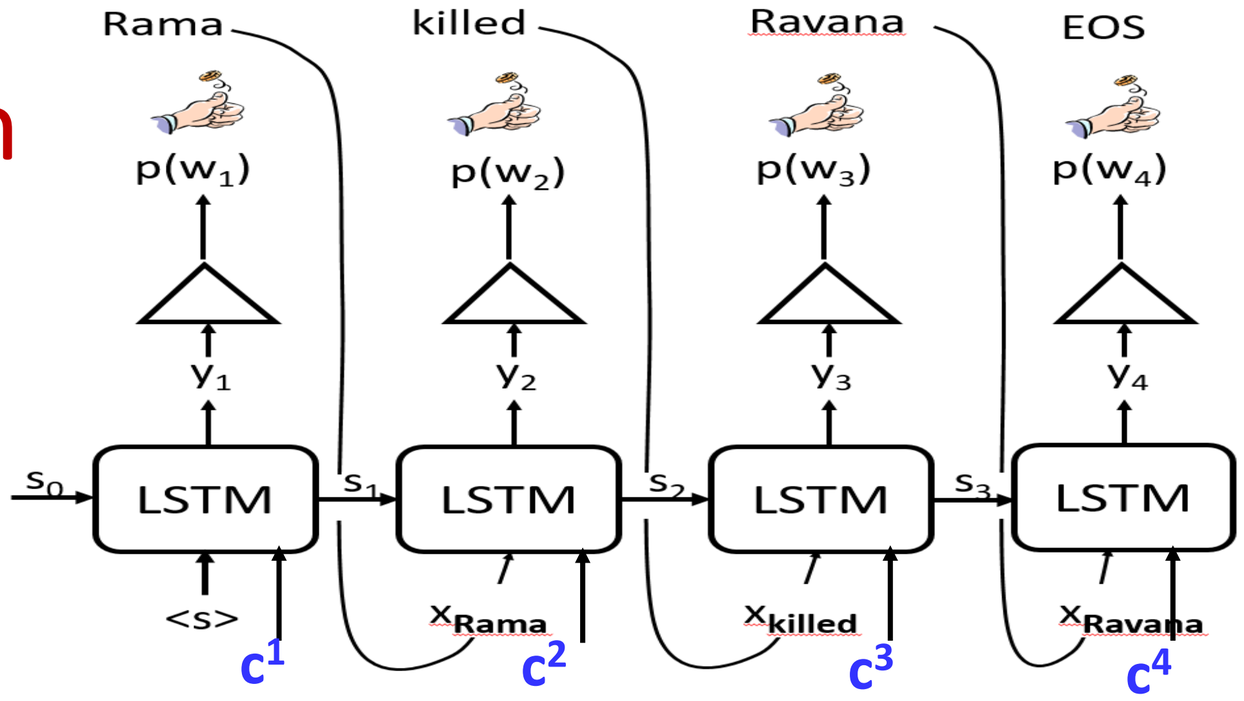


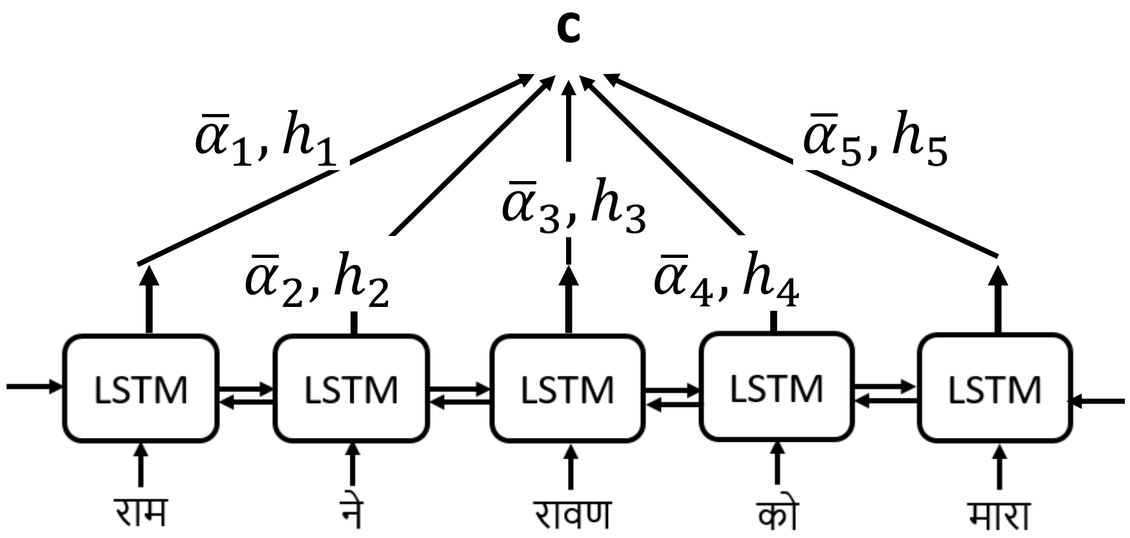
Seq2Seq with Attention



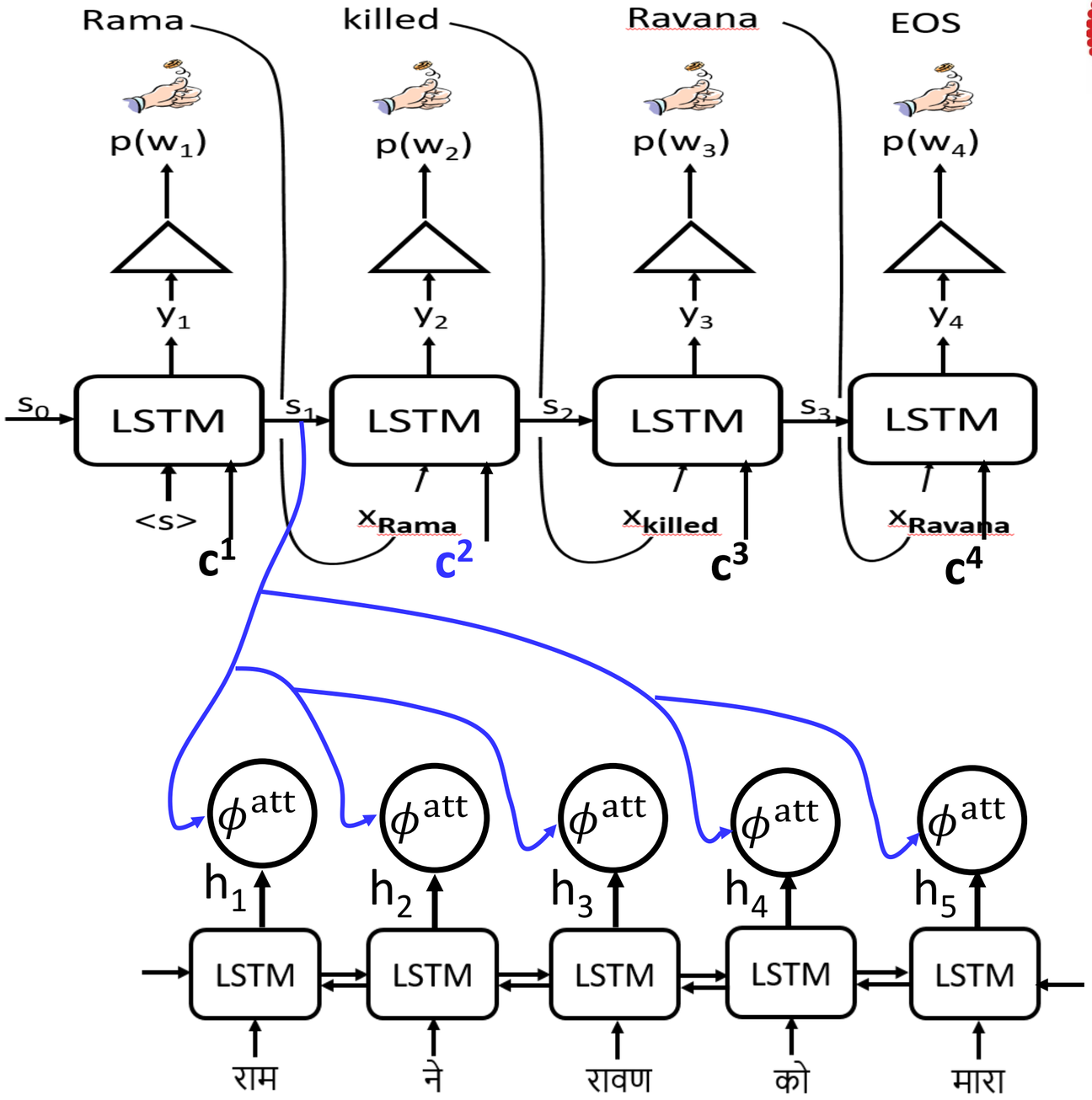


Seq2Seq with Attention

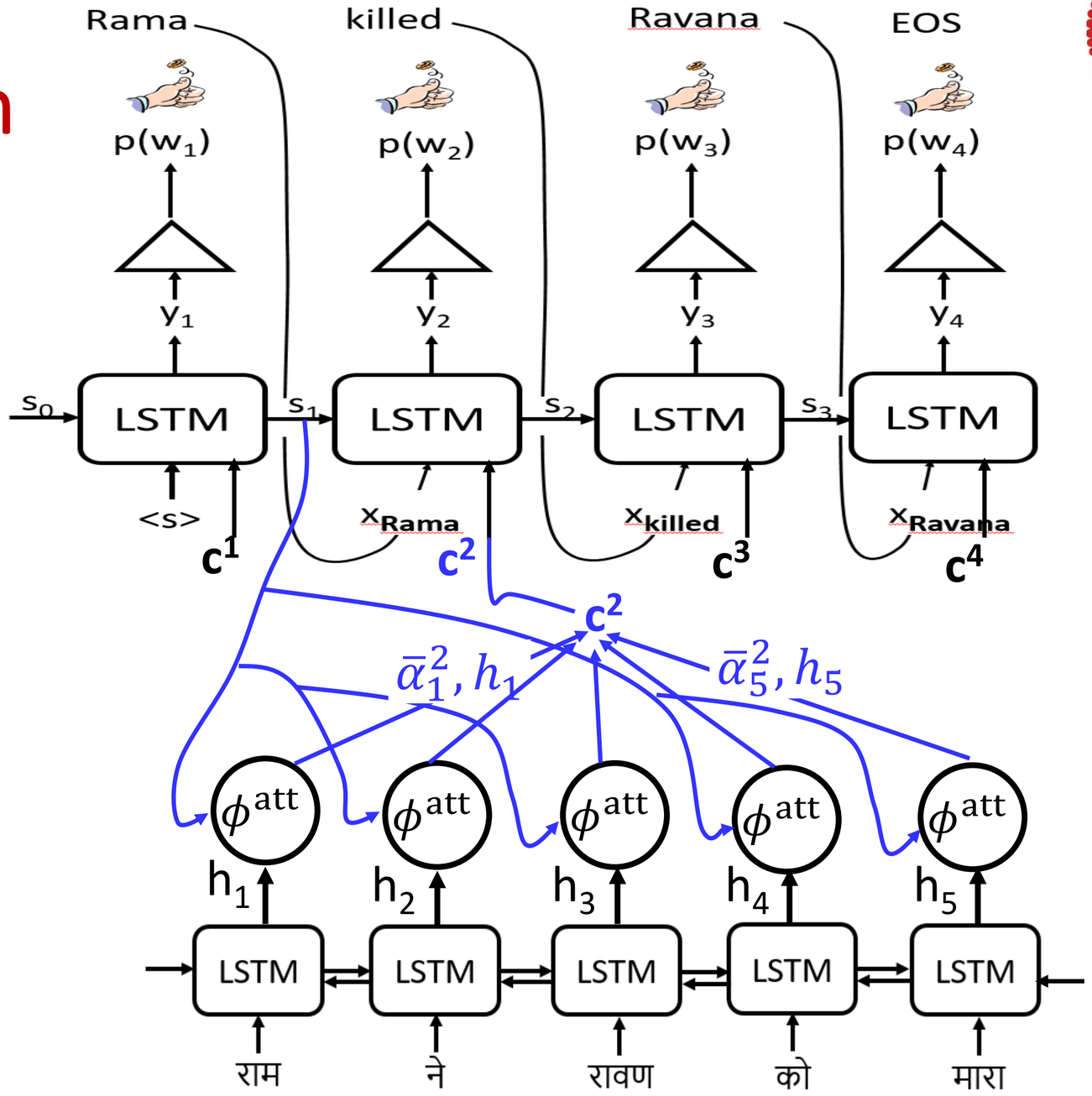




Seq2Seq with Attentior



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.

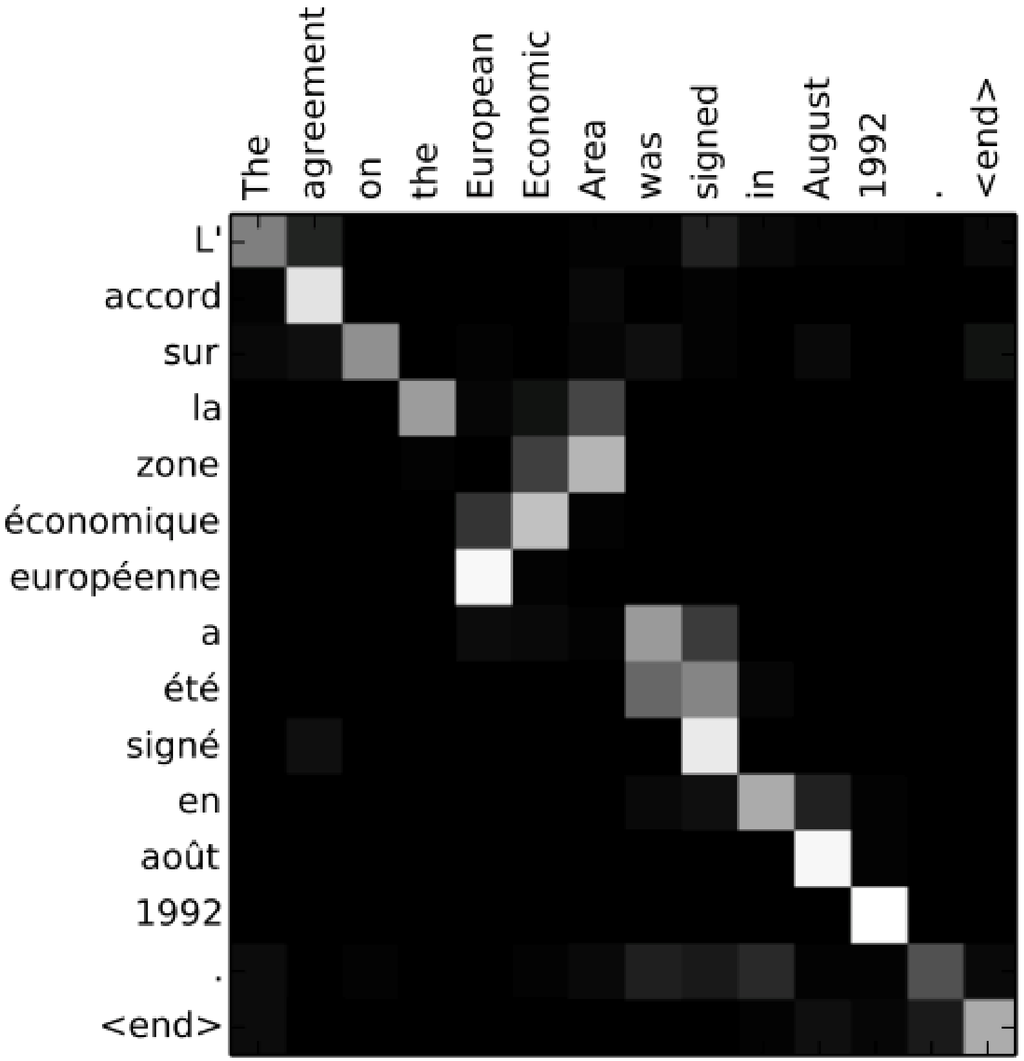
Encoder-Decoder with Attention

- 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>

Multi-head Key-Value Self Attention

Attention: Encoding (h → x)

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

$$\alpha = \text{softmax}(\bar{\alpha}_1, \ldots, \bar{\alpha}_T)$$

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

Attention: Encoding

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

$$\alpha = \text{softmax}(\bar{\alpha}_1, \ldots, \bar{\alpha}_T)$$

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

Each vector (x) playing two roles

(1) computing importance

(2) weighted sum

Key-Value Attention

- Project an input vector x_i into two vectors
  k: key vector $k_i = W^K x_i$
  v: value vector $v_i = W^V x_i$

- Use key vector for computing attention

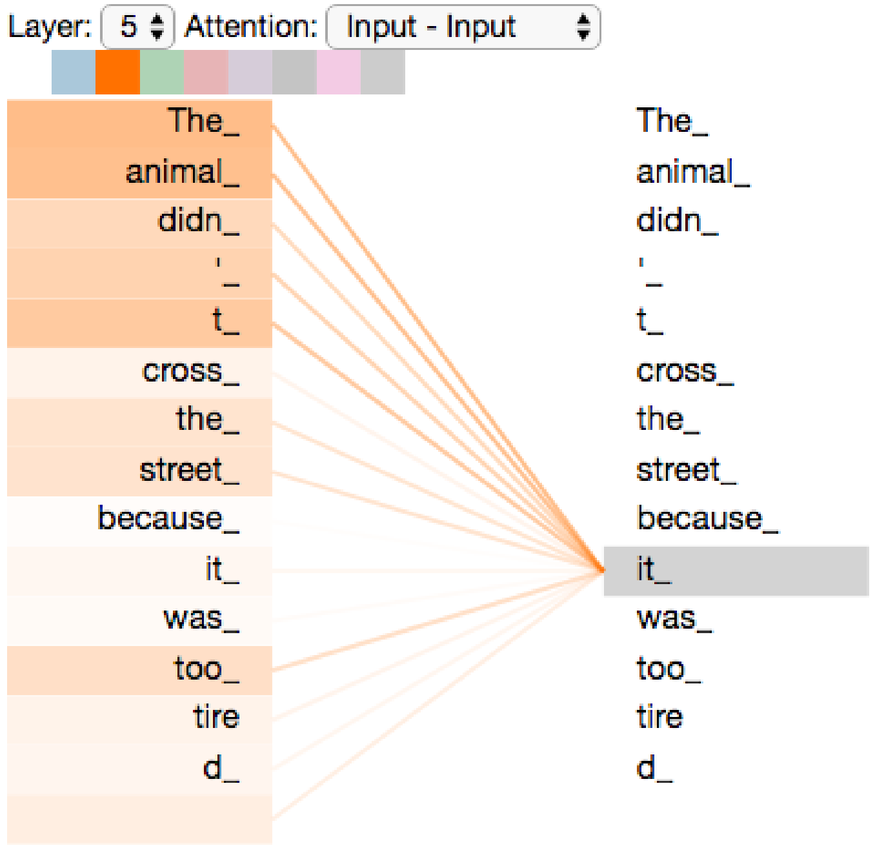
$$\phi^{att}(q, x_i) = \phi^{att}(q, k_i) = \frac{k_i . q}{\sqrt{d}}$$ //scaled multiplicative

- Use value vector for computing attended summary

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

Self-attention (single-head, high-level)

”The animal didn't cross the street because it was too tired”



There is no external query q.
The input is also the query.
Many approaches:
https://ruder.io/deep-learning-nlp-best-practices/

Transformers: query q is another $x_{j:}$ $\phi^{att}(x_j,x_i)$

Key-Value Single-Head Self Attention

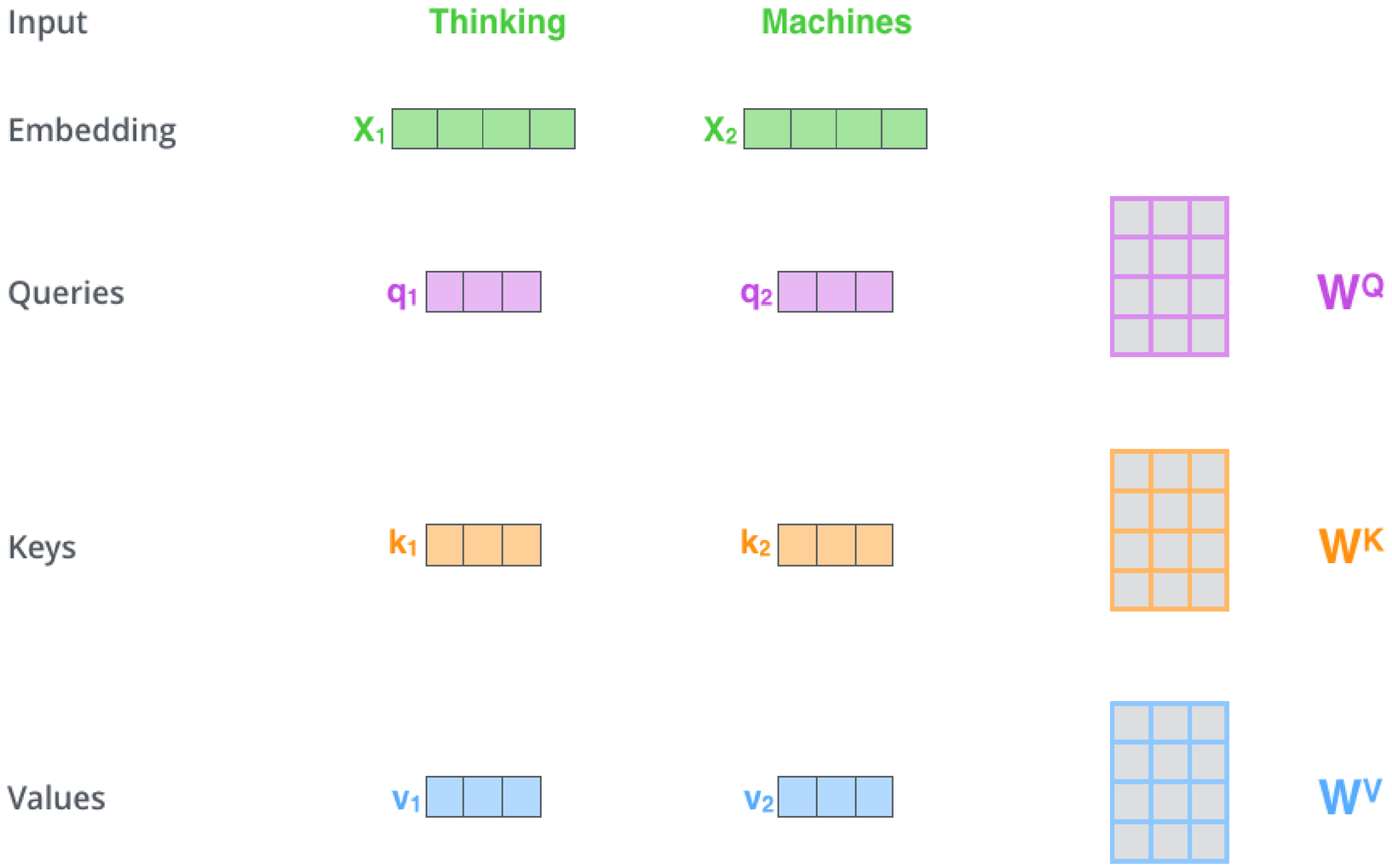
- Project an input vector x_i intc vectors
  k: key vector: $k_i = W^K x_i$
  v: value vector: $v_i = W^v x_i$
  q: query vector:

- Use key and query vectors for computing attention of i^{th} word at word j

  $\frac{k_i . q_j}{\sqrt{d}}$ //scaled multiplicative

- Use value vector for computing attended summary = $\sum_{i=1}^{T} \alpha_i . v_i$

Key-Value Single-Head Self Attention



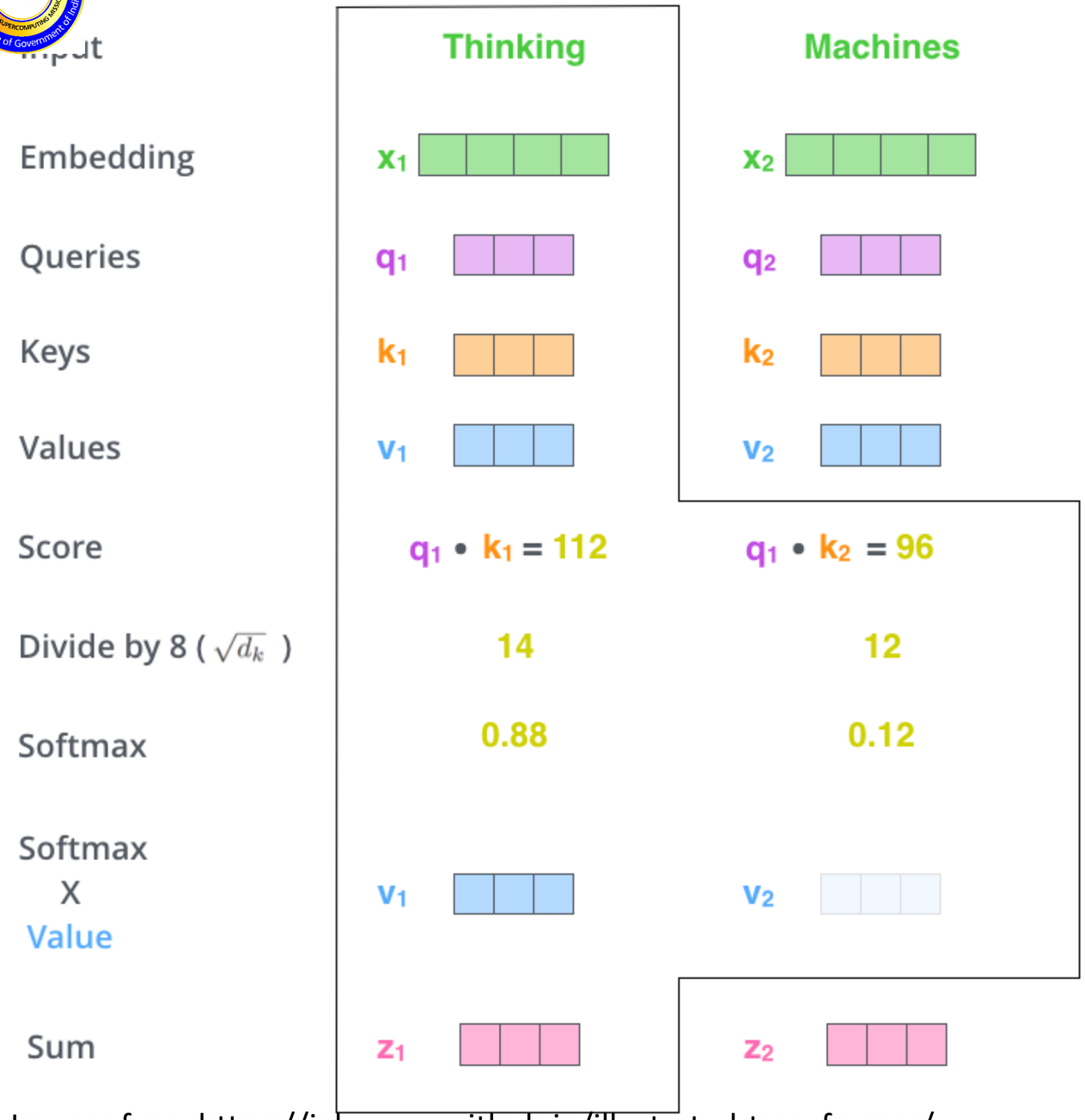
Creation of query, key and value vectors by multiplying by trained weight matrices

Separation of Value and Key and Query

Matrix multiplications are quite efficient and can be done in aggregated manner

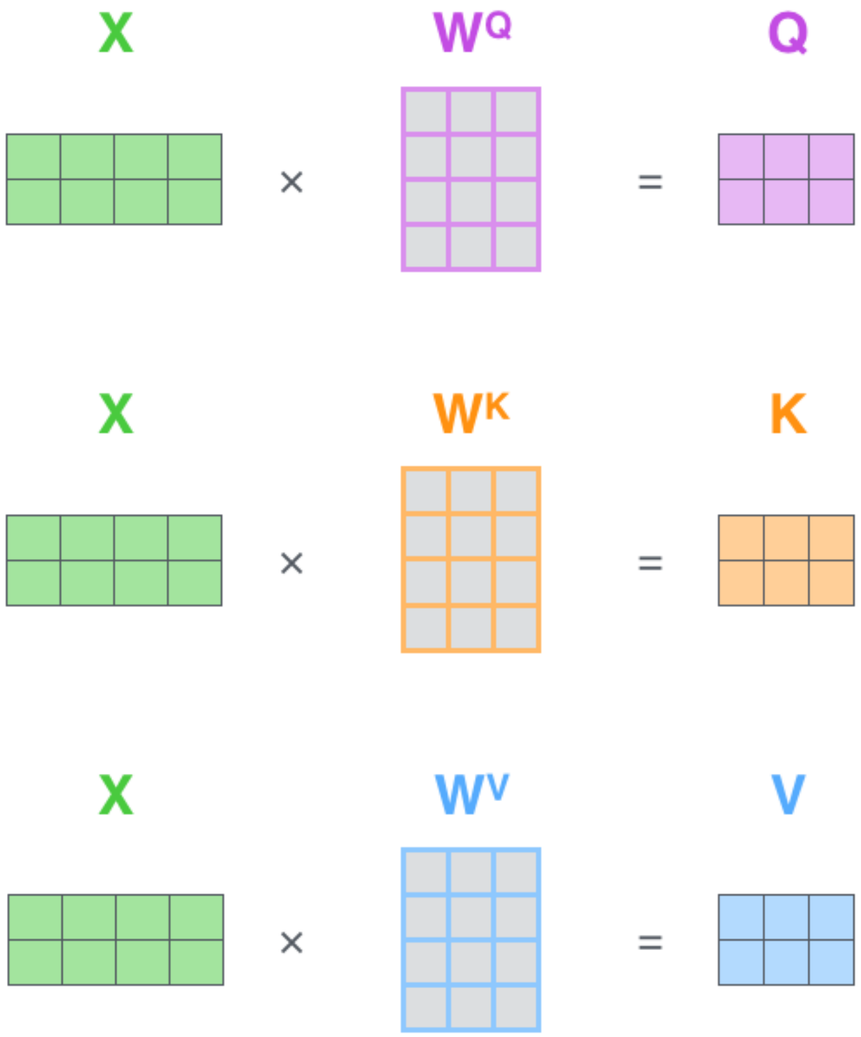
Images from https://jalammar.github.io/illustrated-transformer/

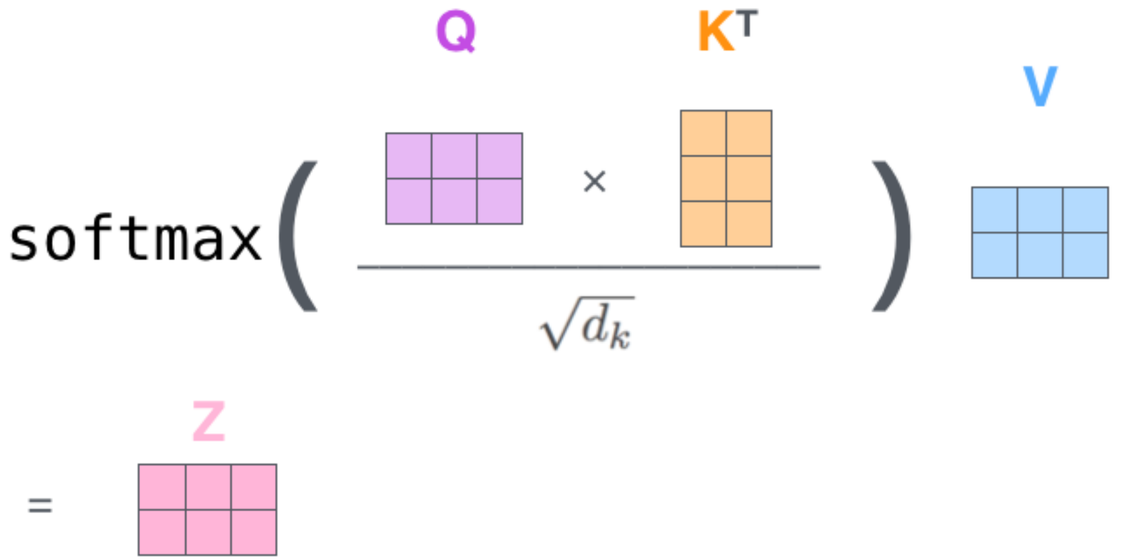
Key-Value Single-Head Self Attention



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

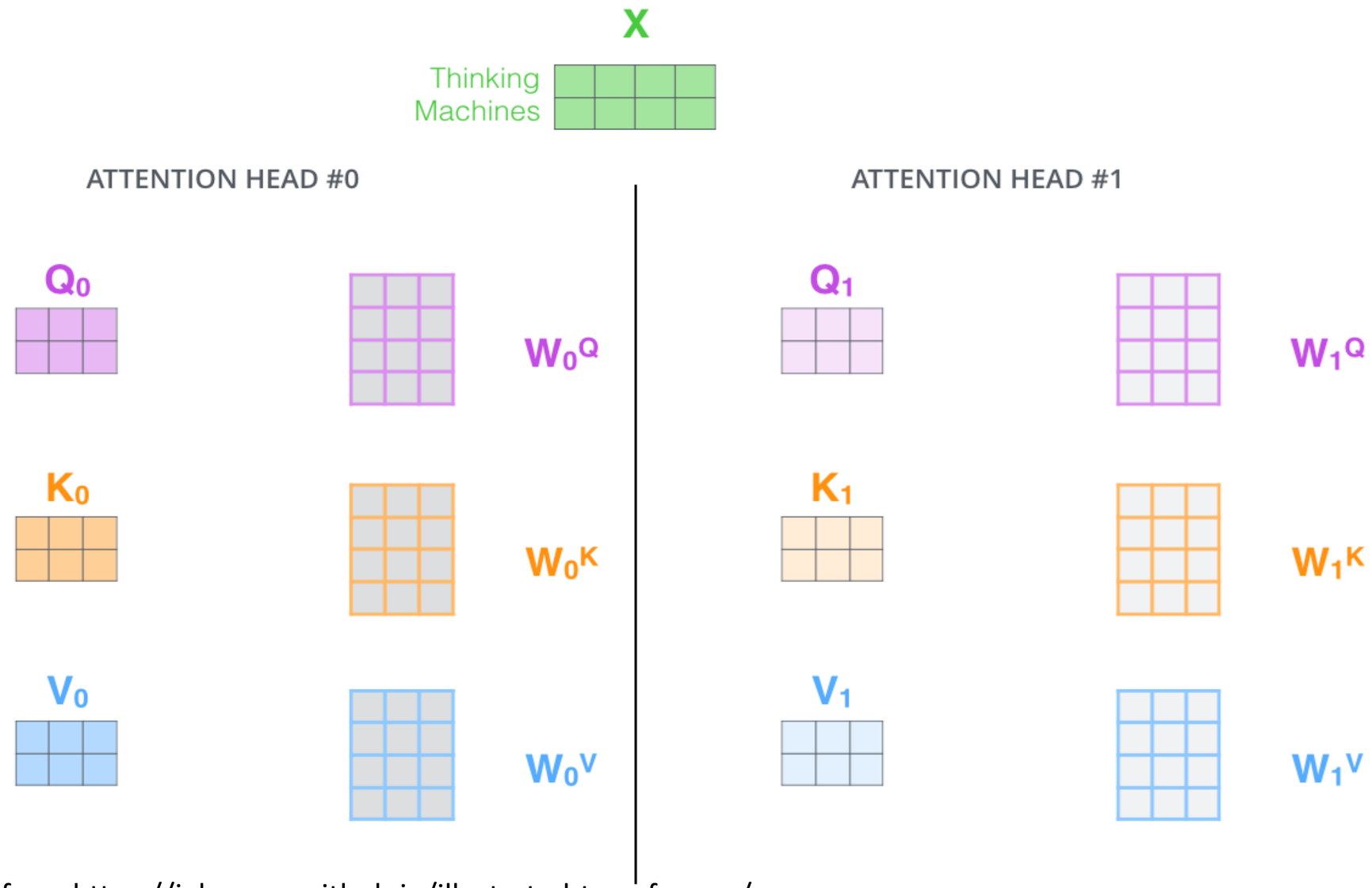
Key-Value Single-Head Self Attention





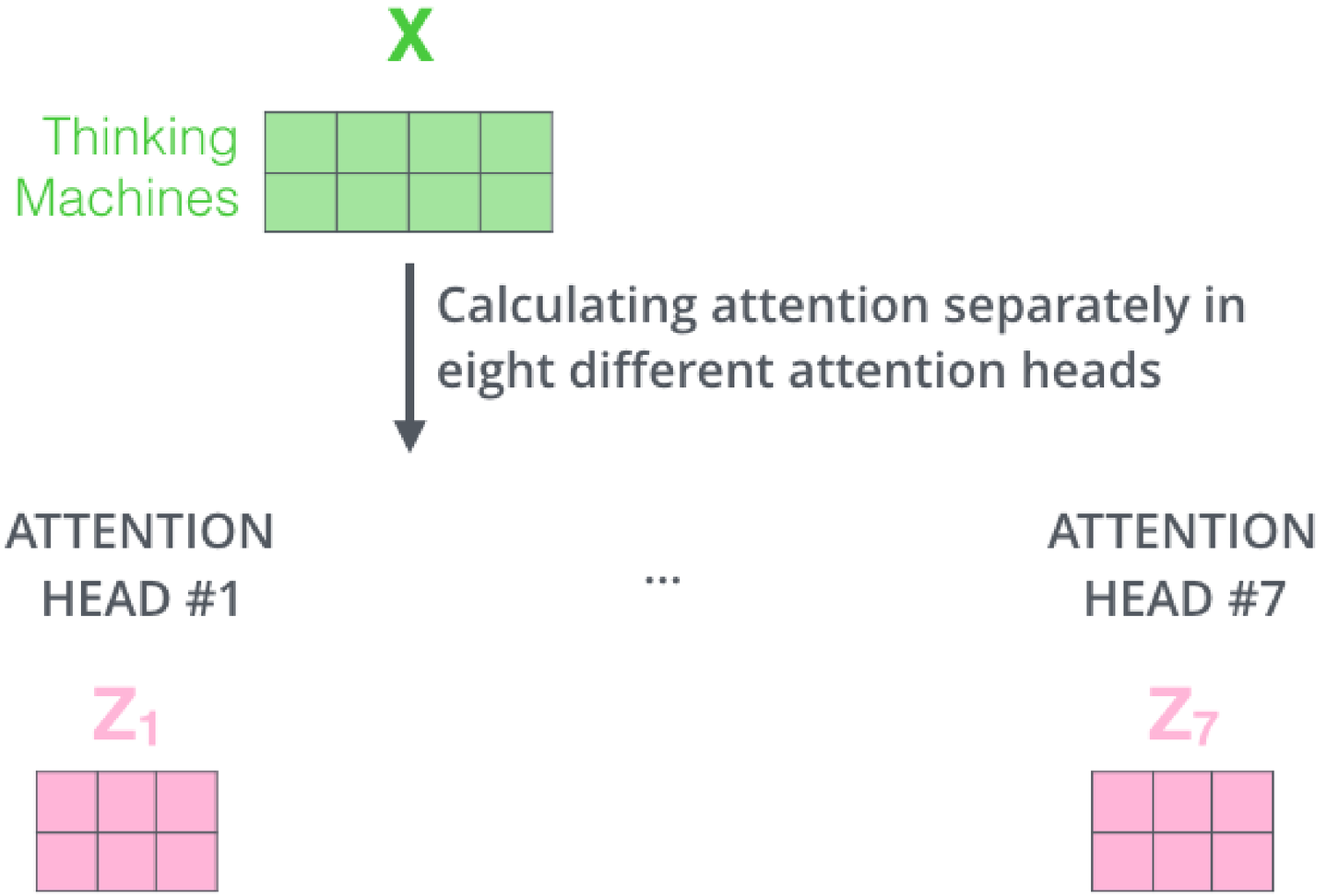
Images from https://jalammar.github.io/illustrated-transformer/

Key-Value Multi-Head Self Attention



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

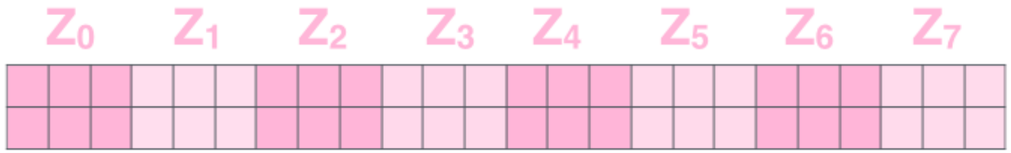
Multi-Head Attention



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

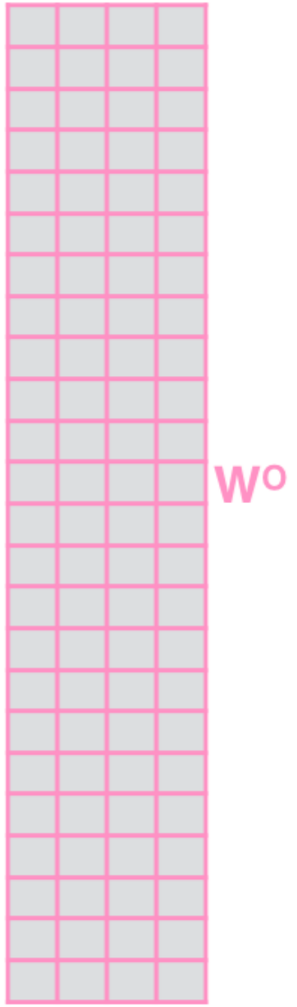
Multi-Head Attended Vector → Output

1) Concatenate all the attention heads



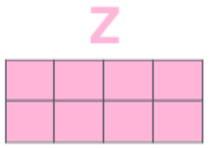
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



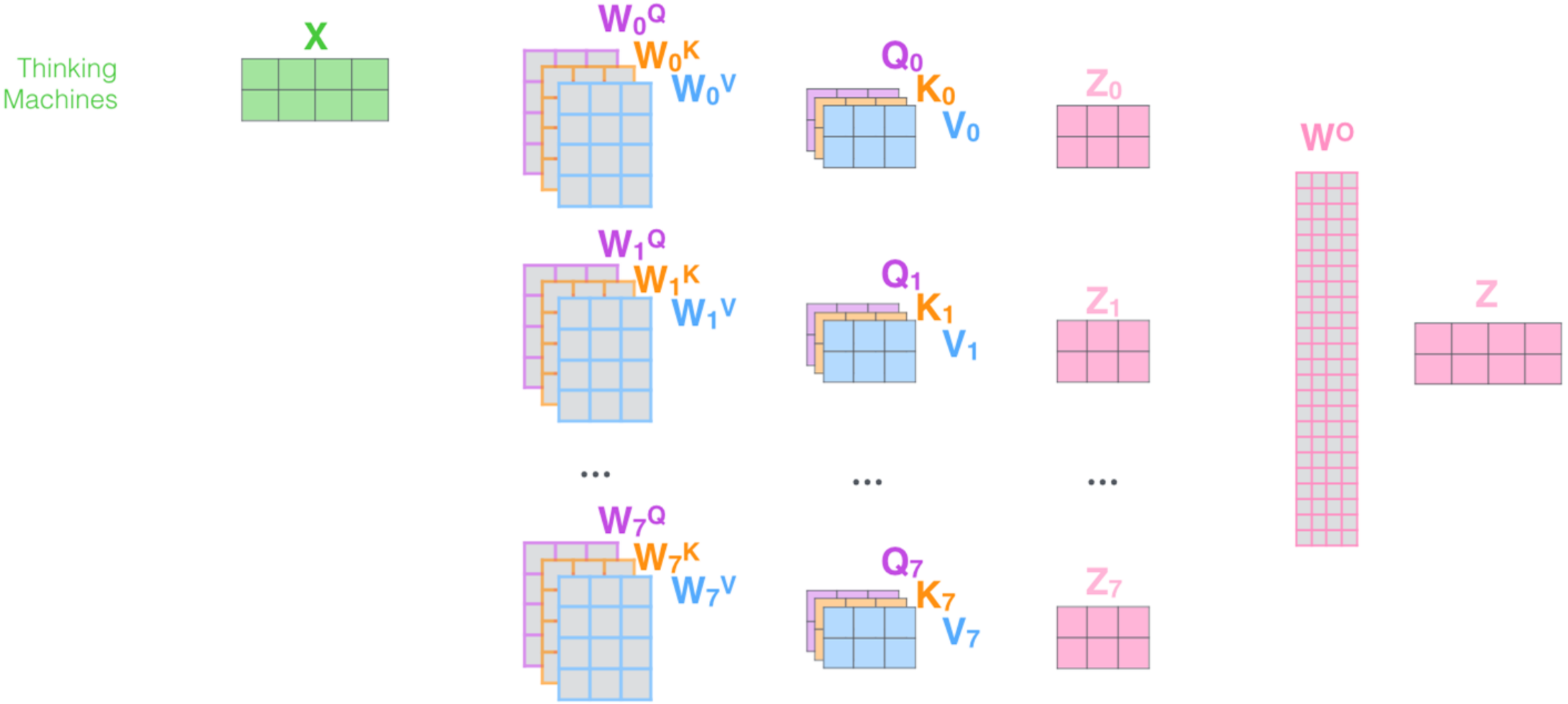
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



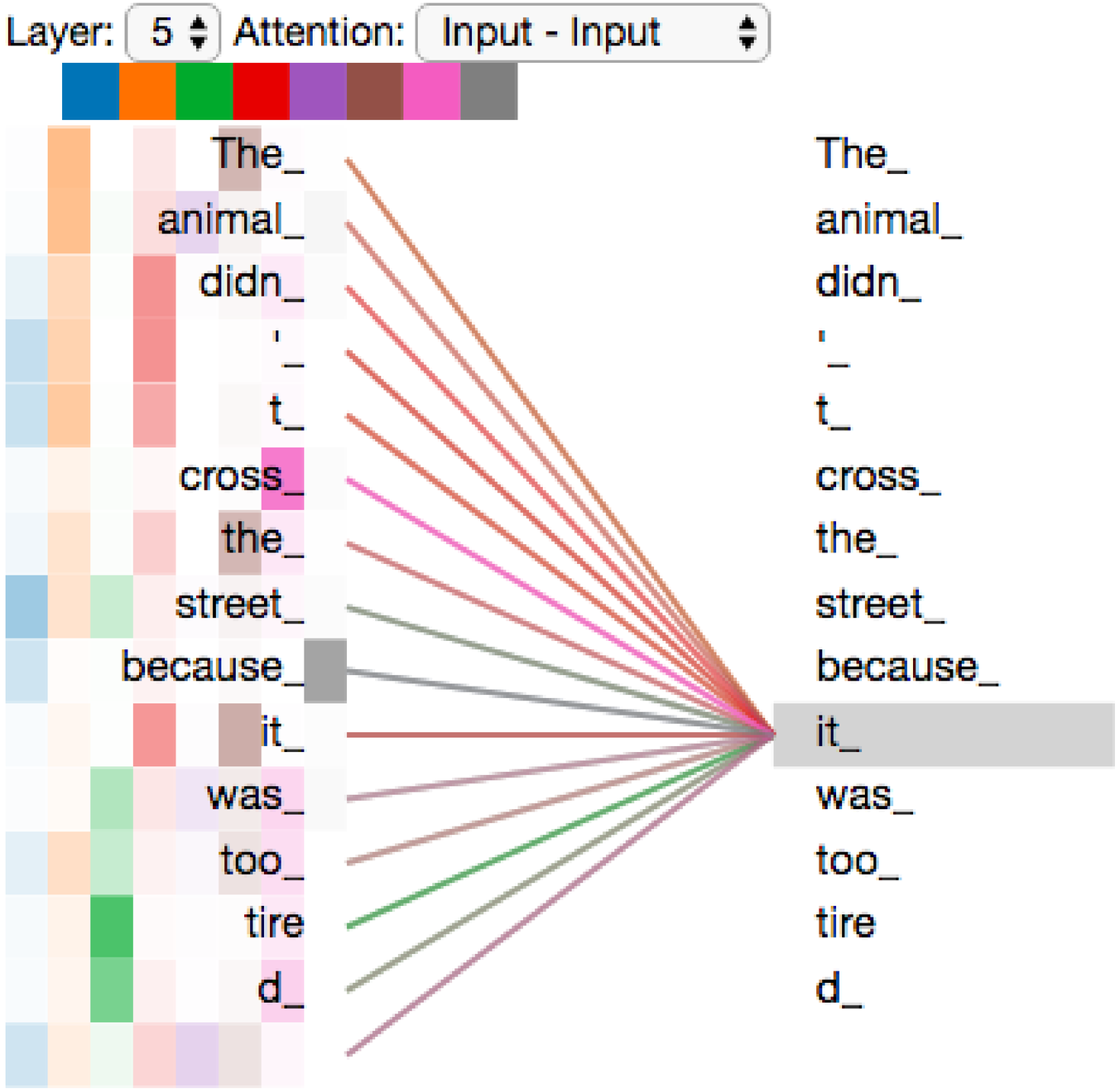
Images from https://jalammar.github.io/illustrated-transformer/

Key-Value Multi-Head Self Attention (summary)



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

Multi-head Self attention visualisation (Interpretable?!)



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

IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

Introduction to Deep Learning

Natural Language Processing

Transformers & BERT

Mausam
IIT Delhi

(some figures taken from Jay Alammar's blog)

National Supercomputing Mission

Centre for Development of Advanced Computing

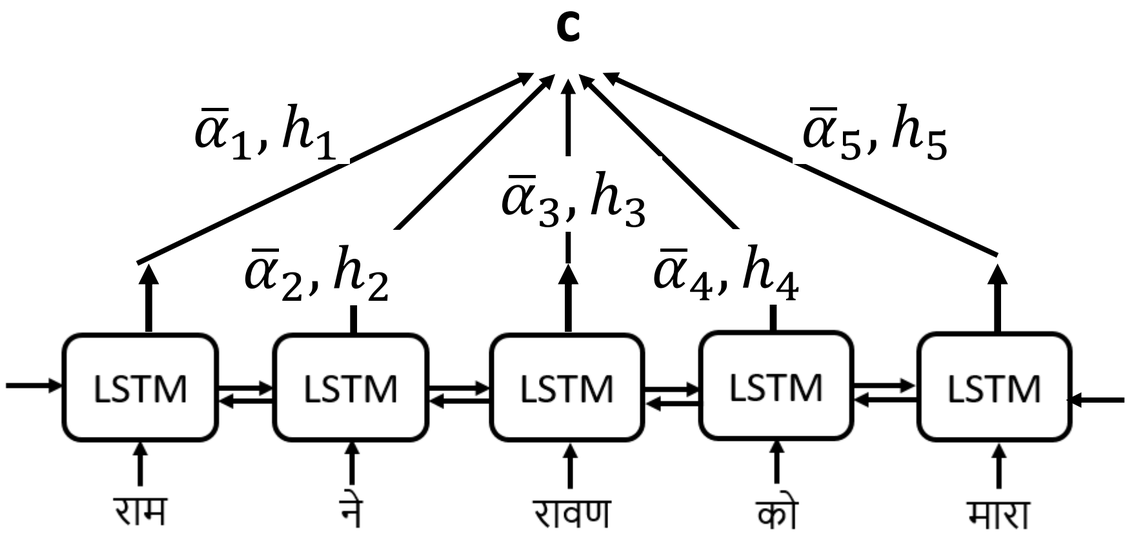
Revision of Key Concepts

Attention

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

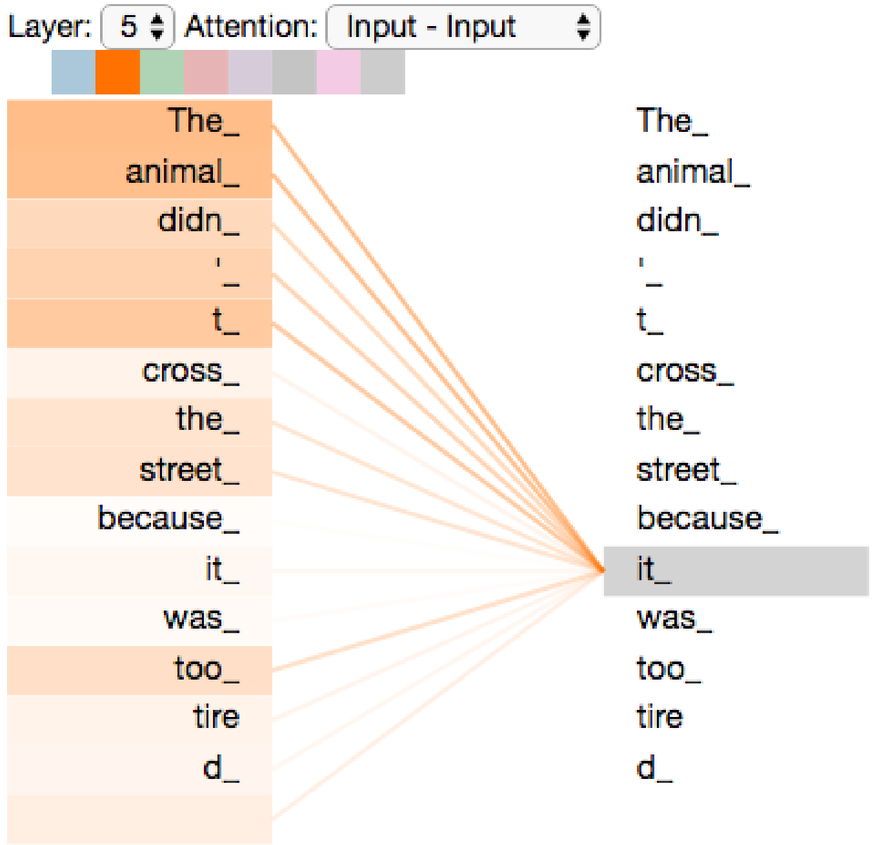
$$\alpha_{1:T} = \text{softmax}(\bar{\alpha}_1, \bar{\alpha}_2, \ldots, \bar{\alpha}_T)$$

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



Self-attention (single-head, high-level)

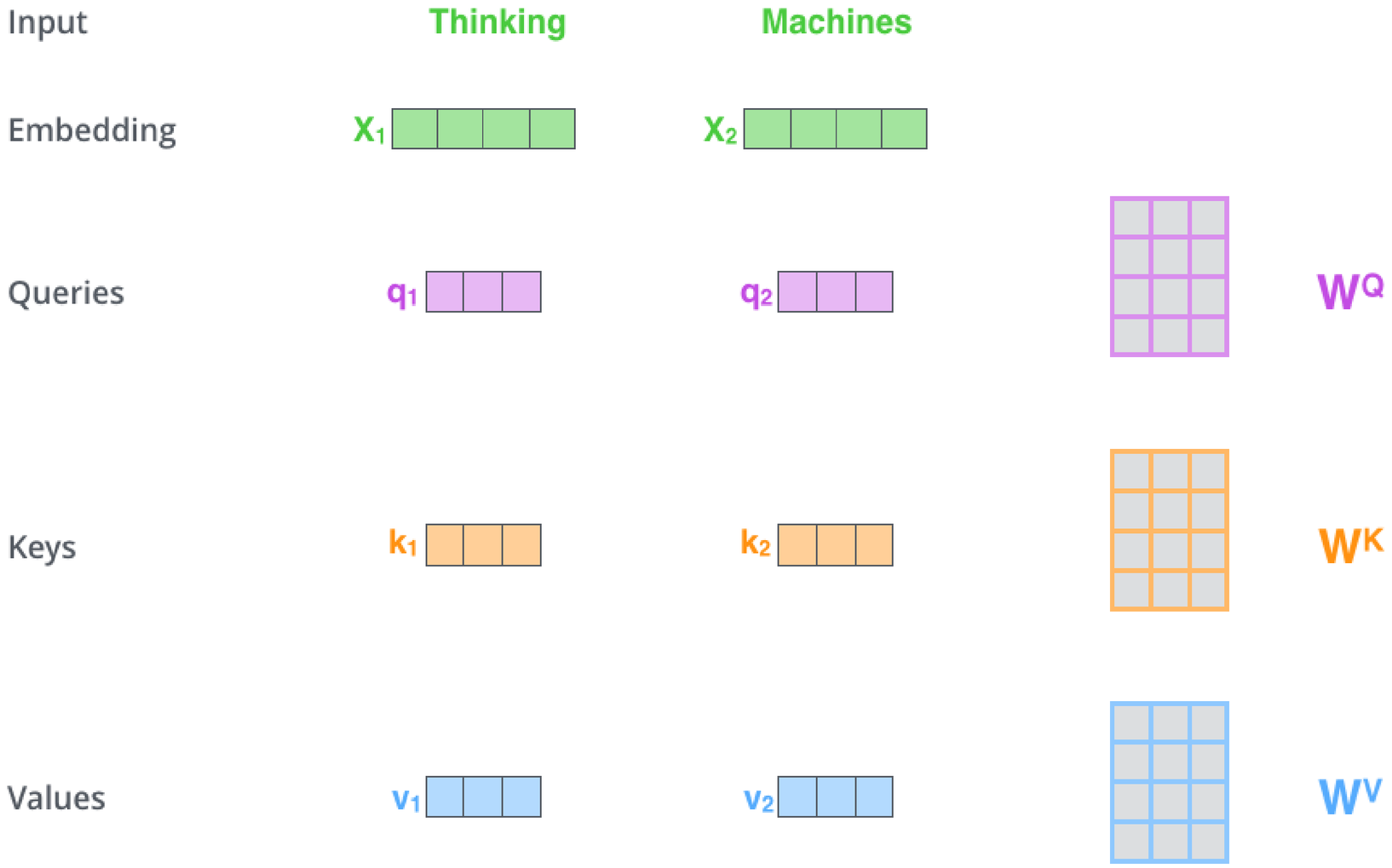
”The animal didn't cross the street because it was too tired”



There is no external query q.
The input is also the query.
Many approaches:
https://ruder.io/deep-learning-nlp-best-practices/

Transformers: query q is another $x_{j:}$ $\phi^{att}(x_j,x_i)$

Key-Value Single-Head Self Attention



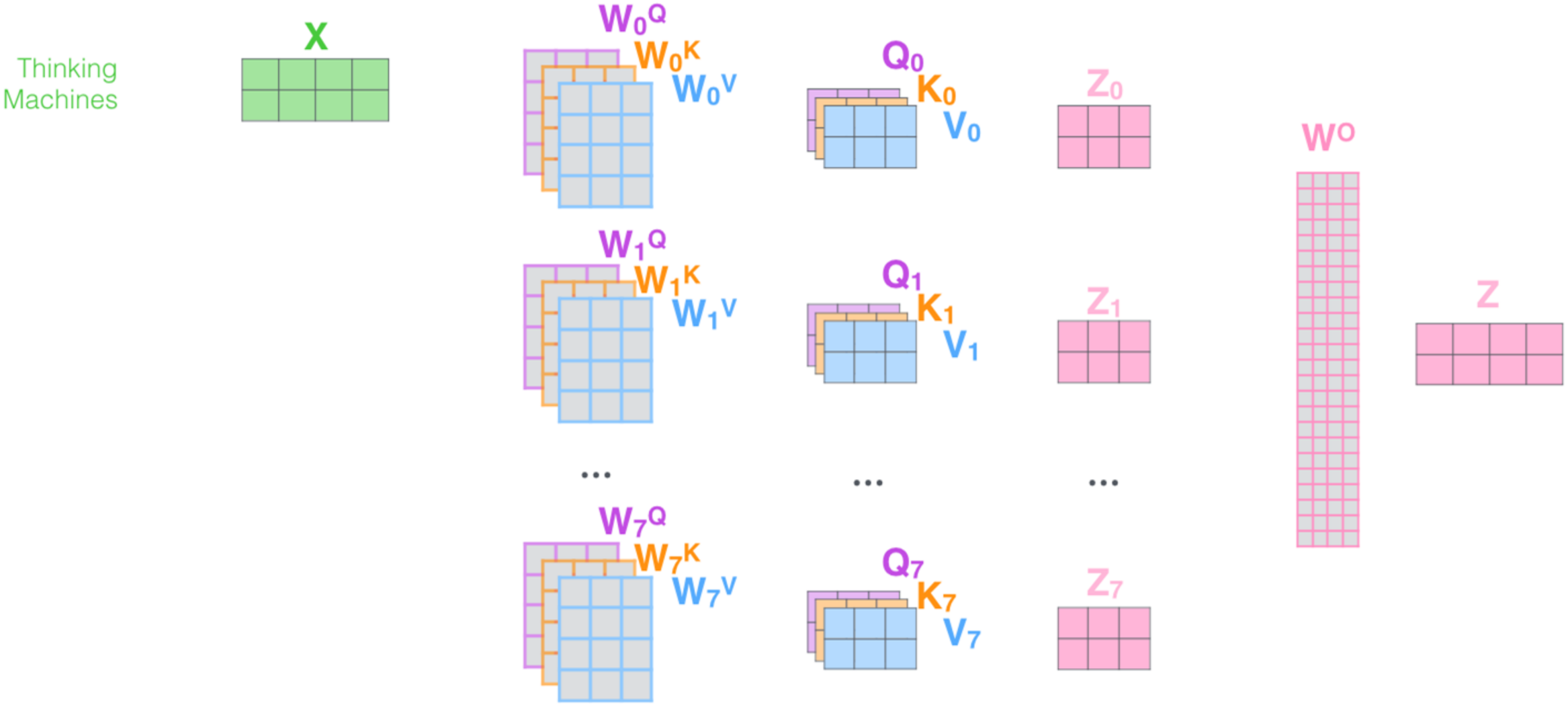
Creation of query, key and value vectors by multiplying by trained weight matrices

Separation of Value and Key and Query

Matrix multiplications are quite efficient and can be done in aggregated manner

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

Key-Value Multi-Head Self Attention



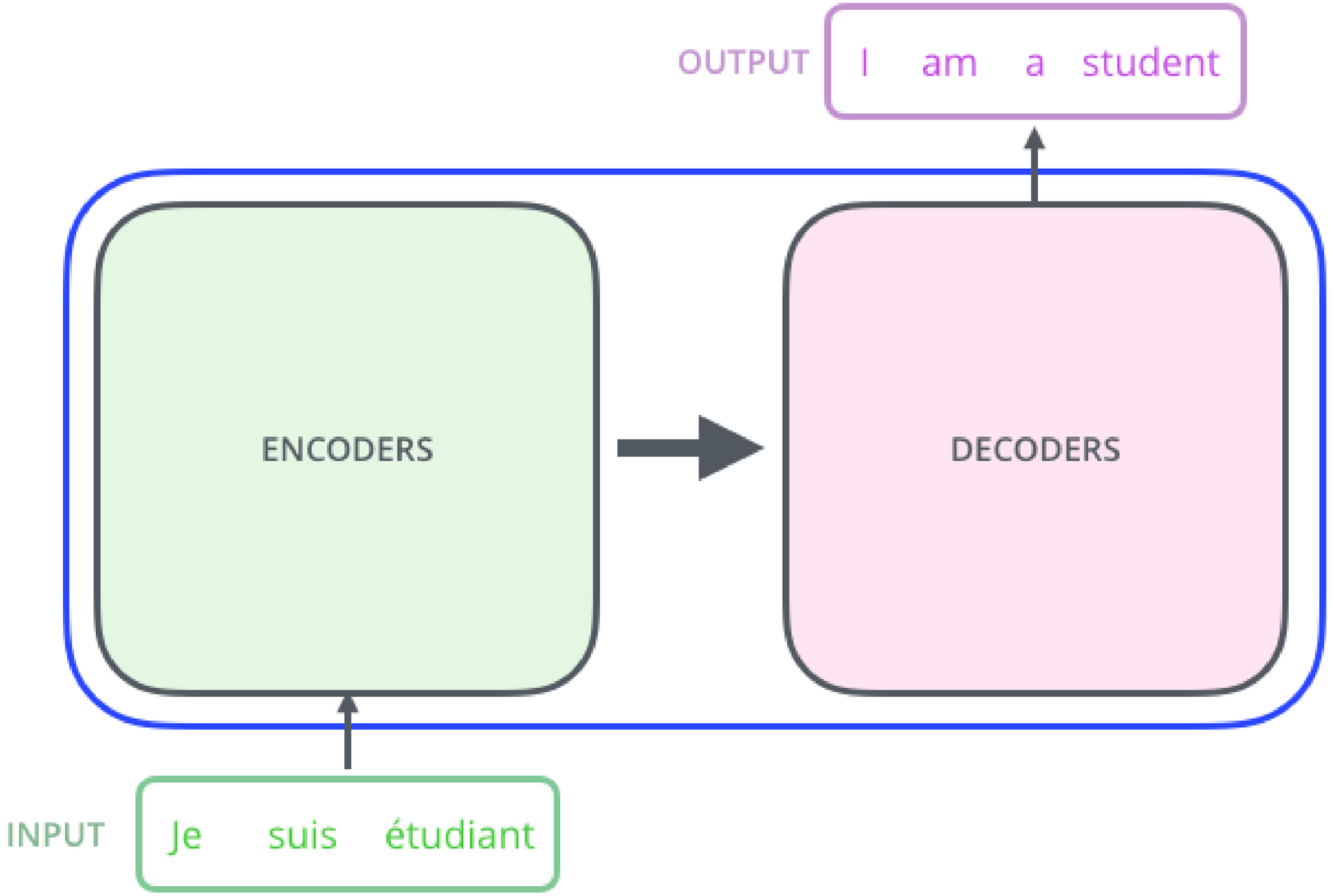
Images from https://jalammar.github.io/illustrated-transformer/

Transformers

Motivation

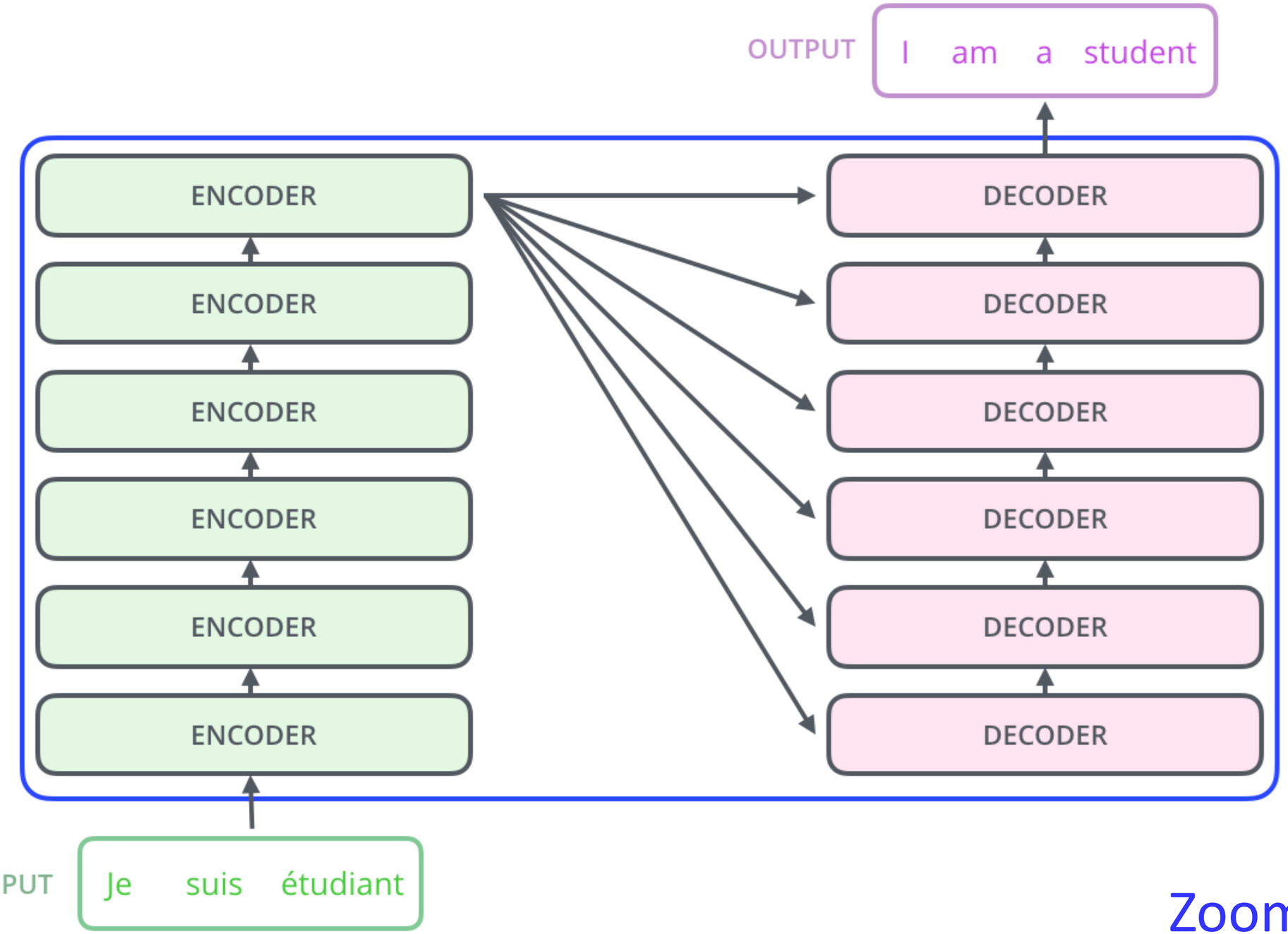
- Recurrence is powerful but
  - Issues with learnability: vanishing gradients
  - Issues with remembering long sentences
  - Issues with scalability:
    - backpropagation time high due to sequentiality in sentence length
  - Issues with scalability:
    - can’t be parallelized even at test time – O(sentence length)

- Remove recurrence: only use attention
  “Attention is All You Need”



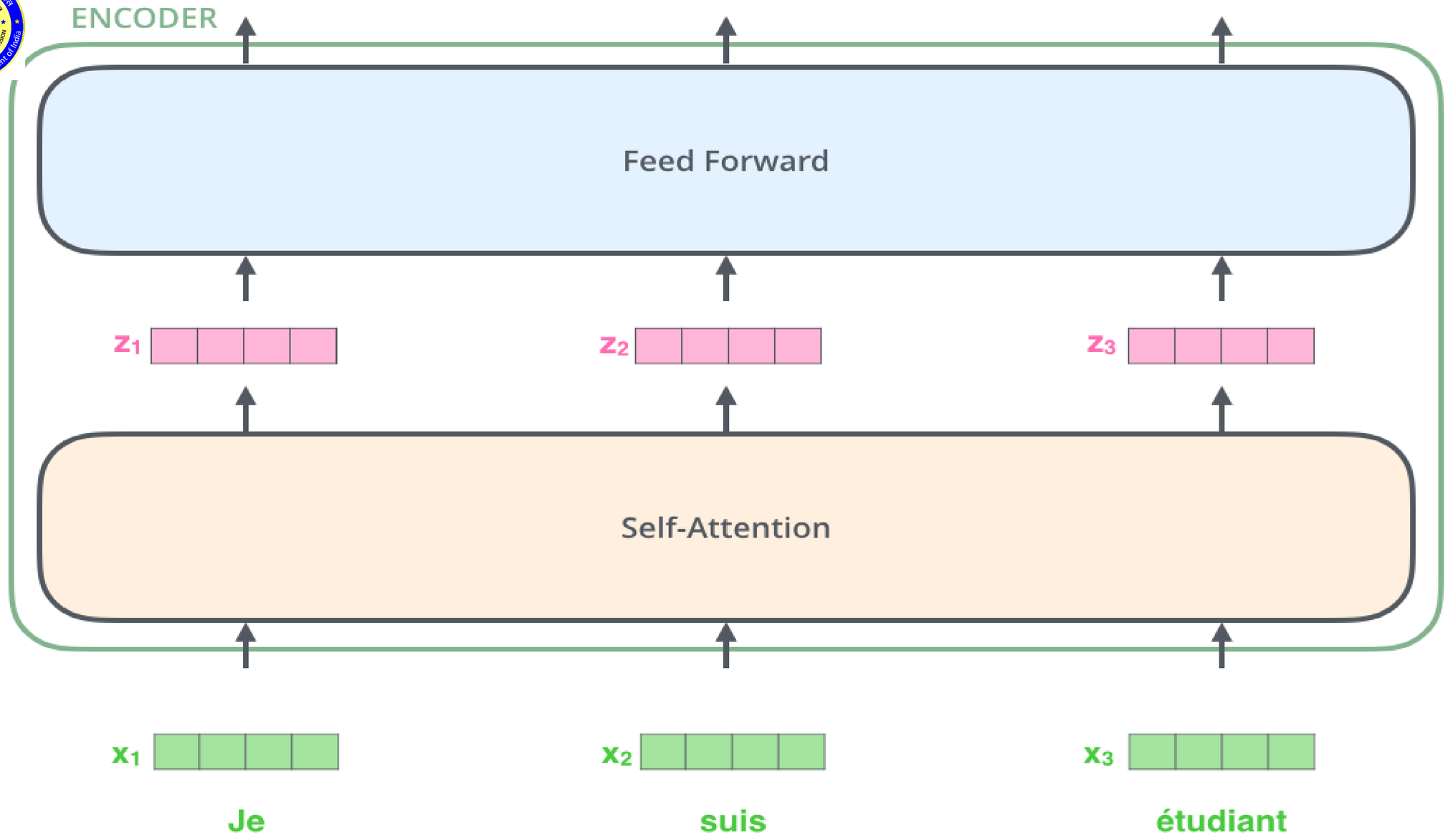
We focus only on encoder… (decoder is an extension of sequence decoders)

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



Zooming in...

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



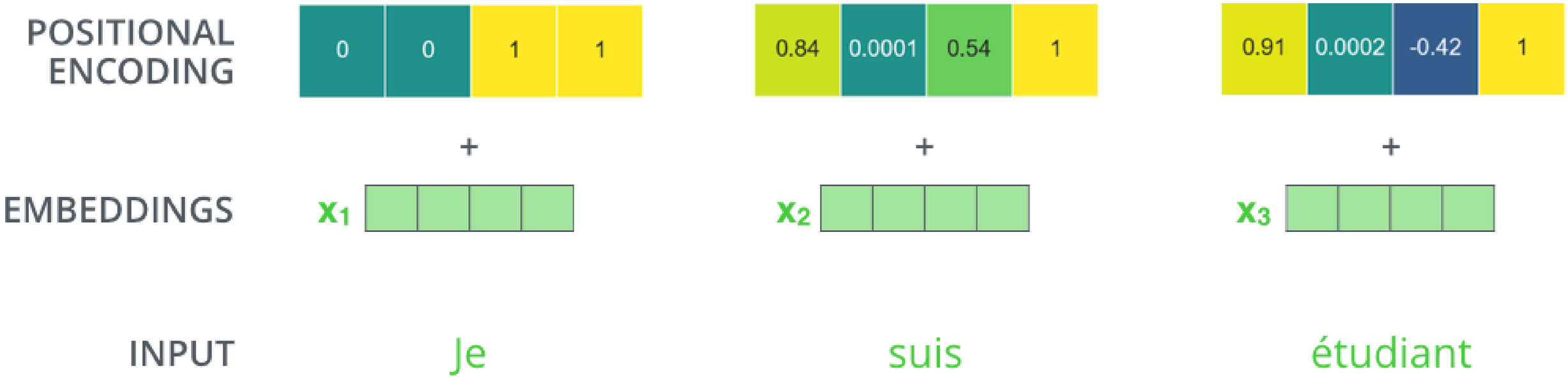
Can you see a fundamental limitation?

Encoders have same architecture but different weights…

Zooming in further...

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

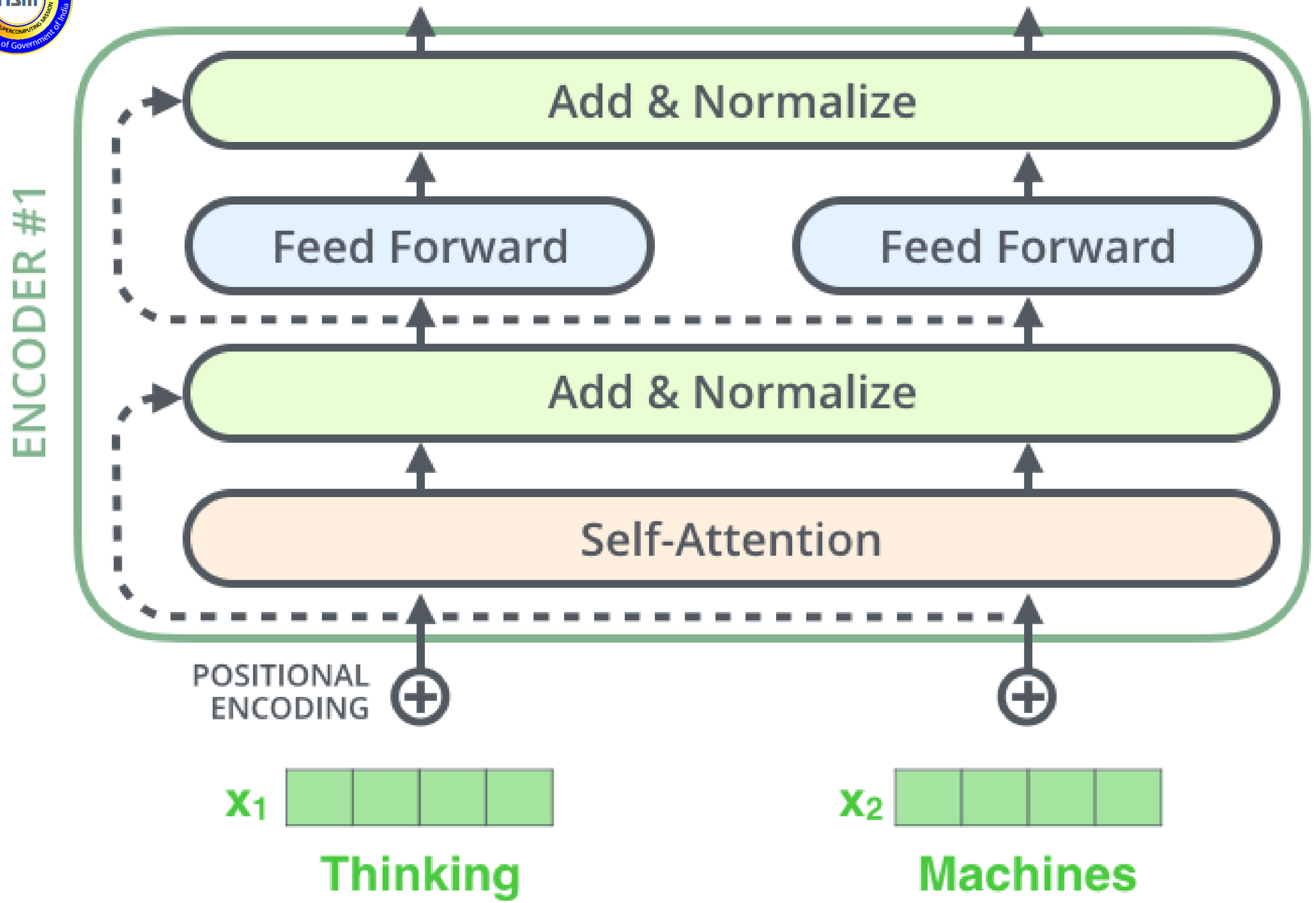
A note on Positional embeddings



Positional embeddings can be extended to any sentence length but if any test input is longer than all training inputs then we will face issues.

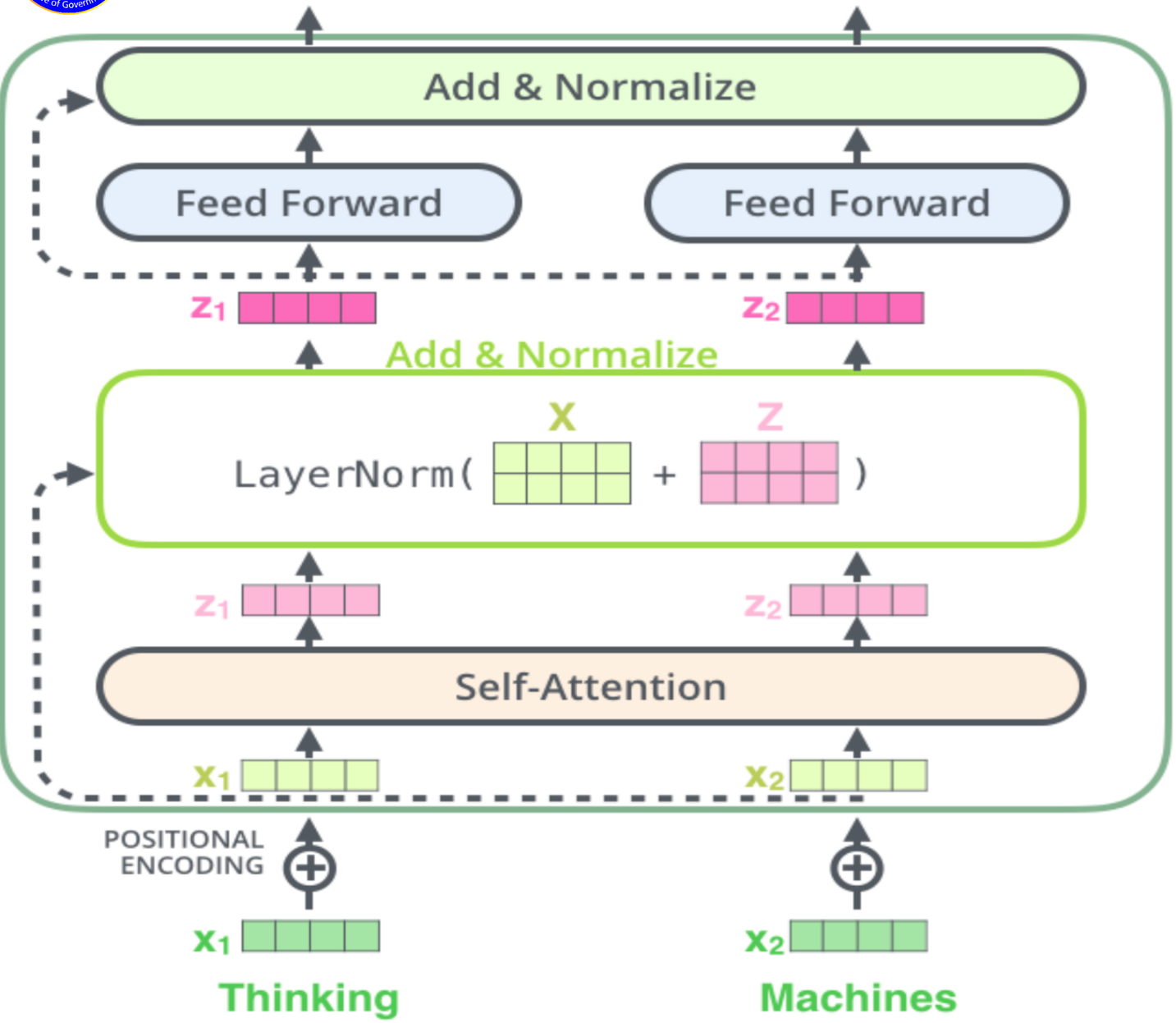
Solution: use a functional form (as in Transformer paper – sinuisoidal encoding)

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



Adding residual connections...

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



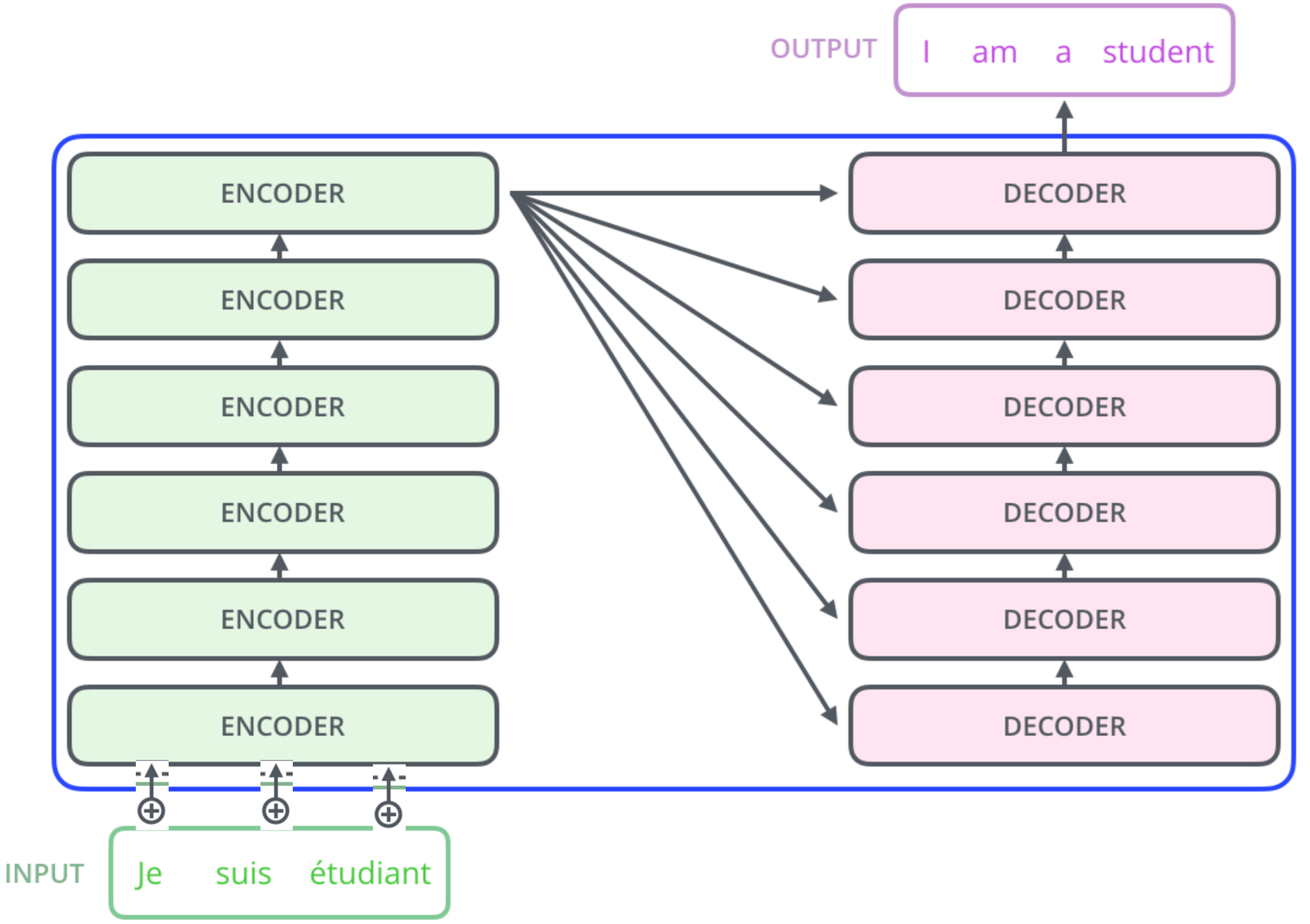
Images from https://jalammar.github.io/illustrated-transformer/

The residual connections help the network train, by allowing gradients to flow through the networks directly.

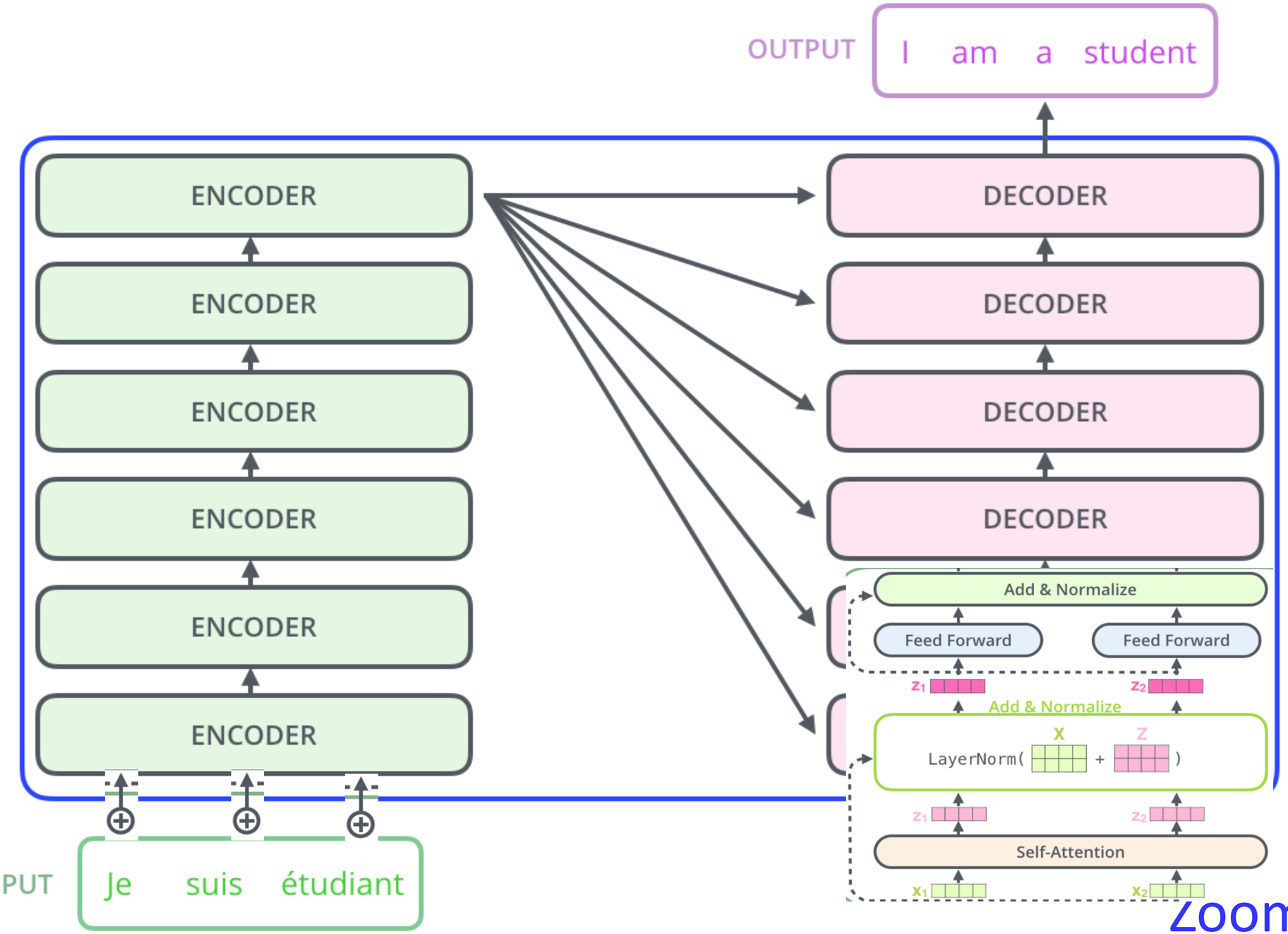
The layer normalizations stabilize the network -- substantially reducing the training time necessary.

$$z=\text{LayerNorm}(x+z)=\gamma\frac{x+z-\mu}{\sigma}+\beta$$

The pointwise feedforward layer is used to project the attention outputs potentially giving it a richer representation.

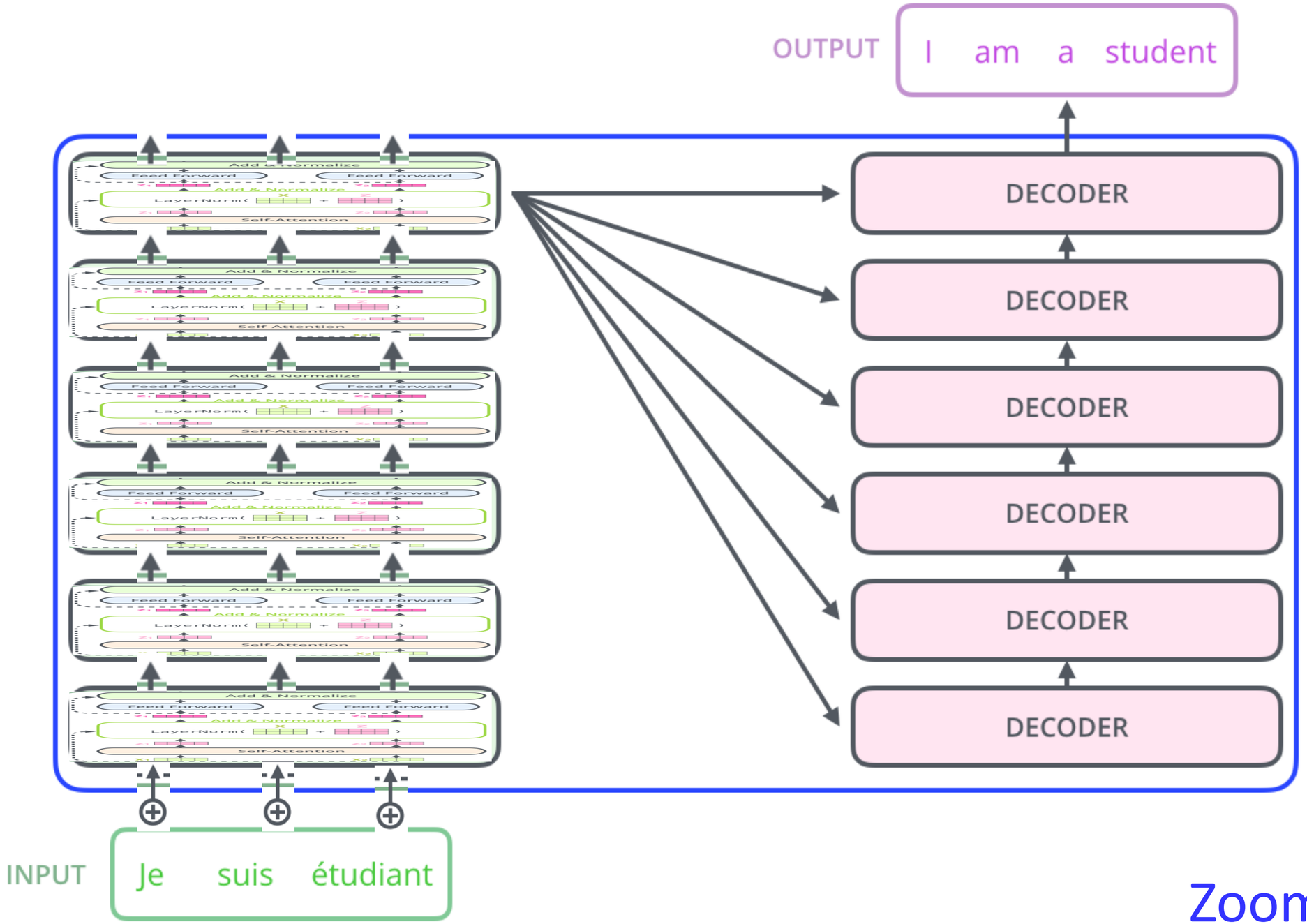


Images from https://jalammar.github.io/illustrated-transformer/



Zooming in...

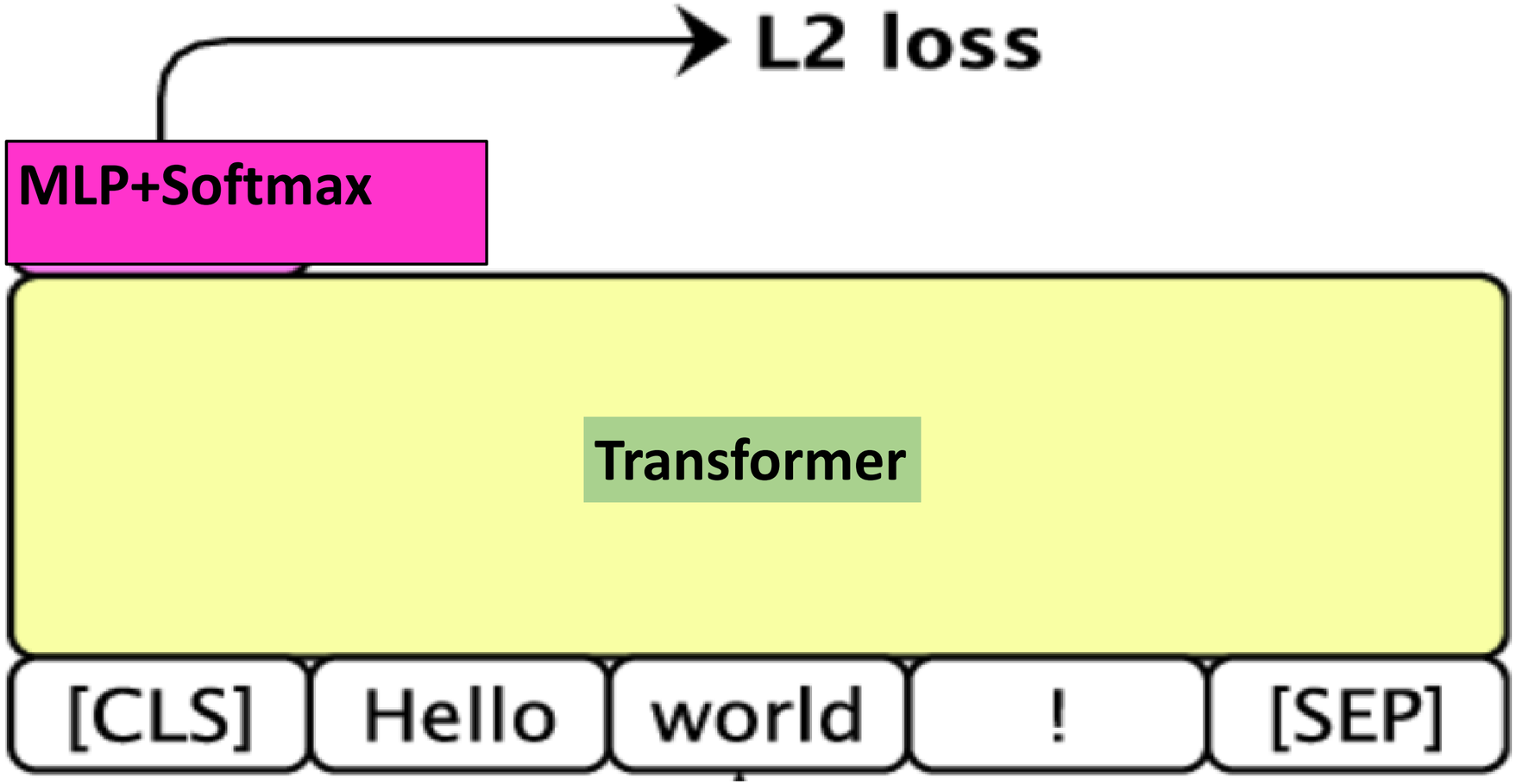
Images from https://jalammar.github.io/illustrated-transformer/



Zooming in...

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

Use of [CLS] for Text Classification



Pros

- Current state-of-the-art.
- Enables deep architectures
- Easier learning of long-range dependencies
- Can be efficiently parallelized
- Gradients don’t suffer from vanishing gradients

Cons

Huge number of parameters so

- Very data hungry
- Takes a long time to train
- No study of memory utilisation

Other issues

- Keeping sentence length limited
- How to ensure multi-head attention has diverse perspectives.

Pre-trained Language Models

Pretraining

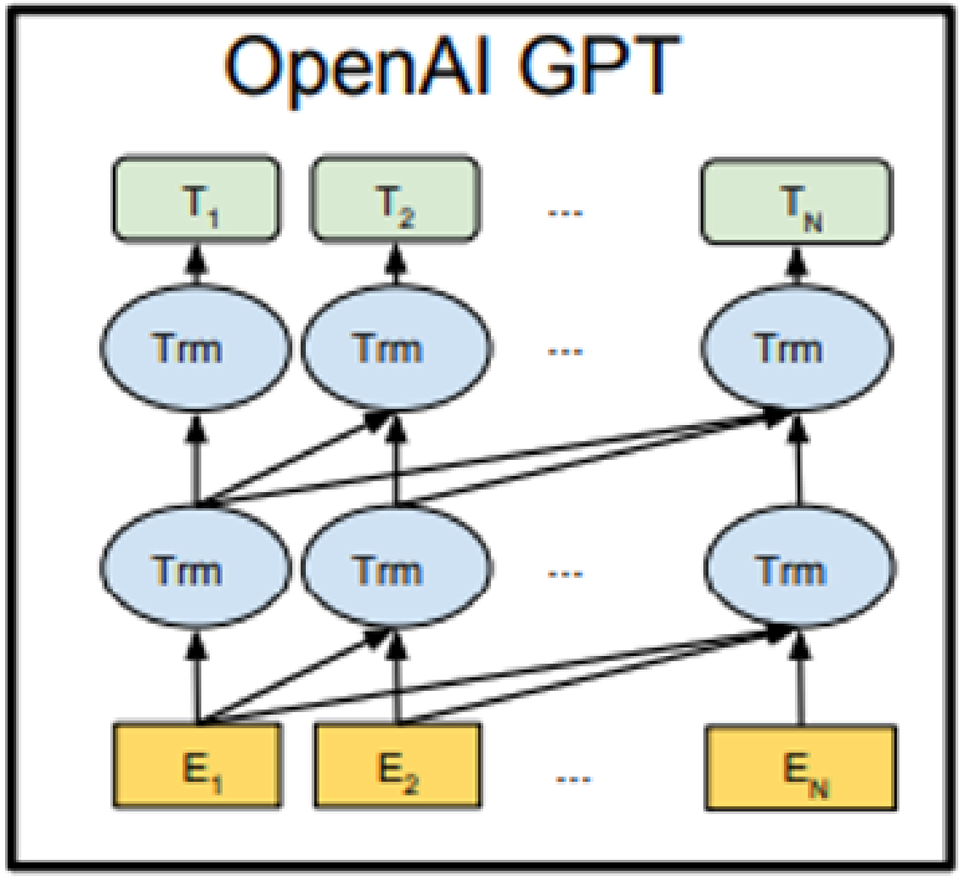
- In NLP, we are interested in solving a variety of end tasks
- One approach - train neural models from scratch
- Issue - This involves two things
  - Modelling of Syntax and Semantics of the language
  - Modelling of the end-task
- Pretraining: Learns the modelling of syntax and semantics through another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

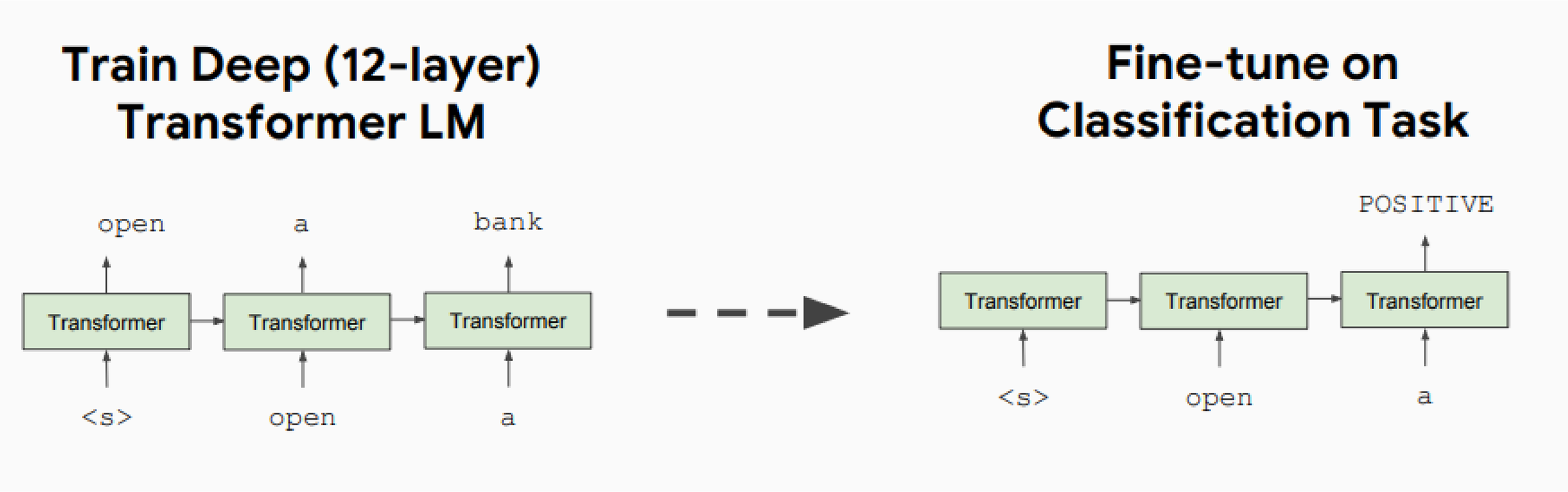
- Solution: Language Modelling
  - Does not require human annotated labels - abundance of sentences
  - Requires understanding of both syntax and semantics to predict the next word in sentence

Model: 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.

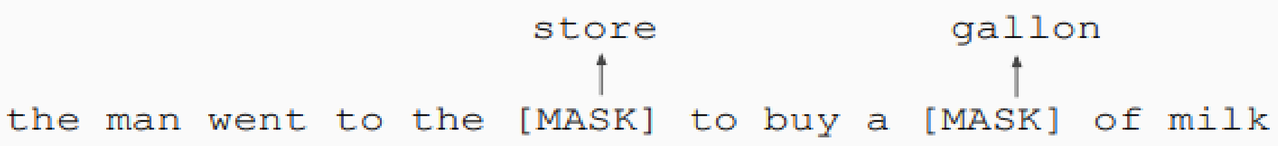
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock 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 2: 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%

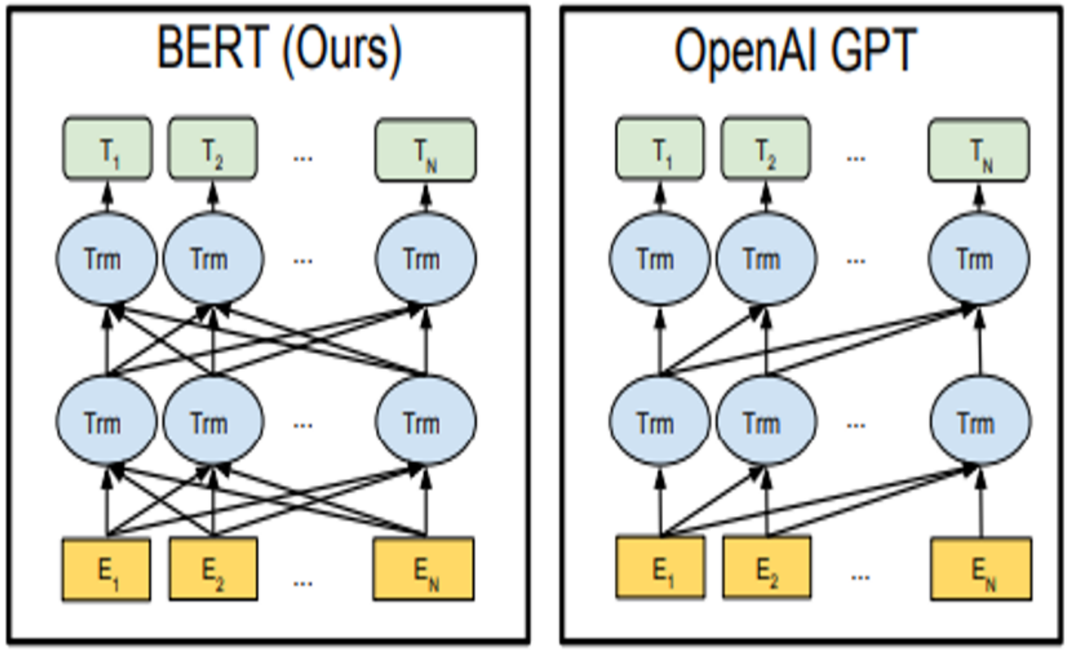


Solution 2: Masked Language Modelling

- Issue with Language modelling - Unidirectional
- Cannot train model on bidirectional context - required for many end tasks
- Solution 2: Masked Language Modelling
  - Randomly mask a word in the sentence
  - Train the model to predict it

*Image Credits: [1]

BERT vs. OpenAI-GPT



Bidirectional Unidirectional

Practical Tips

- New Paradigm: Pre-train → Fine-tune
- Proper modelling of input for BERT is extremely important
  - Question Answering: [CLS] Query [SEP] Passage [SEP]
  - Natural Language Inference: [CLS] Sent1 [SEP] Sent2 [SEP]
- 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
- Highly optimized for TPUs, not so much for GPUs

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

System	Metric		
	Opt. F1	AUC	Last F1
Stanford-IE	23	13.4	22.9
OllIE	41.1	22.5	40.9
PropS	31.9	12.6	31.8
MinIE	41.9	-*	41.9
OpenIE-4	51.6	29.5	51.5
OpenIE-5	48.5	25.7	48.5
ClausIE	45.1	22.4	45.1
CopyAttention	35.4	20.4	32.8
RNN-OIE	49.2	26.5	49.2
Sense-OIE	17.2	-*	17.2
Span-OIE	47.9	-*	47.9
CopyAttention + BERT	51.6	32.8	49.6
IMoJIE	**53.5**	**33.3**	**53.3**

Table 3: Comparison of various OpenIE systems - non-neural, neural and proposed models. (*) Cannot compute AUC as Sense-OIE, MinIE do not emit confidence values for extractions and released code for Span-OIE does not provision calculation of confidence values. In these cases, we report the Last F1 as the Opt. F1

Images from https://arxiv.org/pdf/2005.08178.pdf

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.

Thank You