NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 16 : RNN Language Models

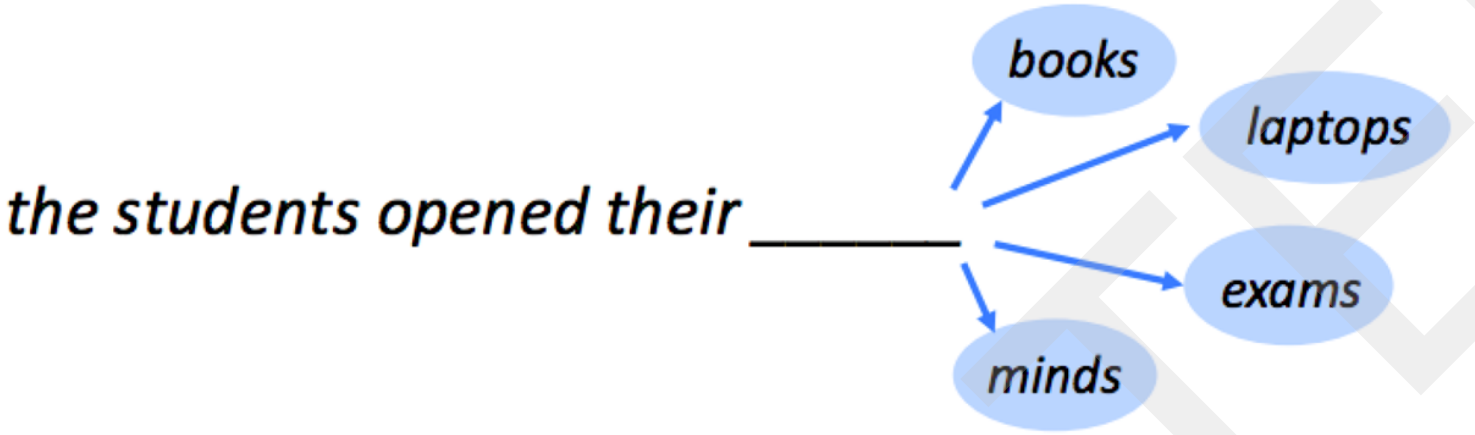
PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- A Neural Language Model
- Recurrent Neural Networks: Intuition
- RNN Language Model

Recap: Language Modeling

Language Modeling is the task of predicting what word comes next.



- **Goal:** Compute the probability of a sentence or sequence of words:

$$P(W) = P(w_1, w_2, w_3, \ldots, w_n)$$

- **Related Task:** probability of an upcoming word:

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

A model that computes either of these is called a language model

What is an issue?

- We treat all words / prefixes independently of each other!

students opened their ___

pupils opened their ___

scholars opened their ___

undergraduates opened their ___

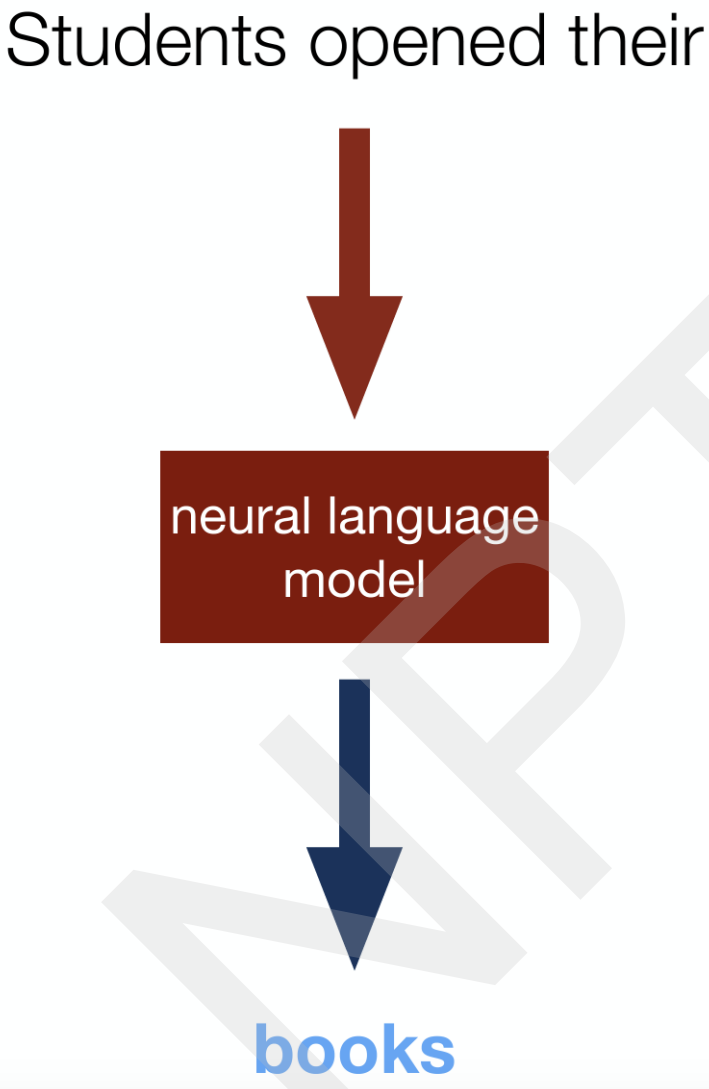
students turned the pages of their ___

students attentively perused their ___

Shouldn't we *share information* across these semantically-similar prefixes?

Source: https://people.cs.umass.edu/~miyyer/cs685/

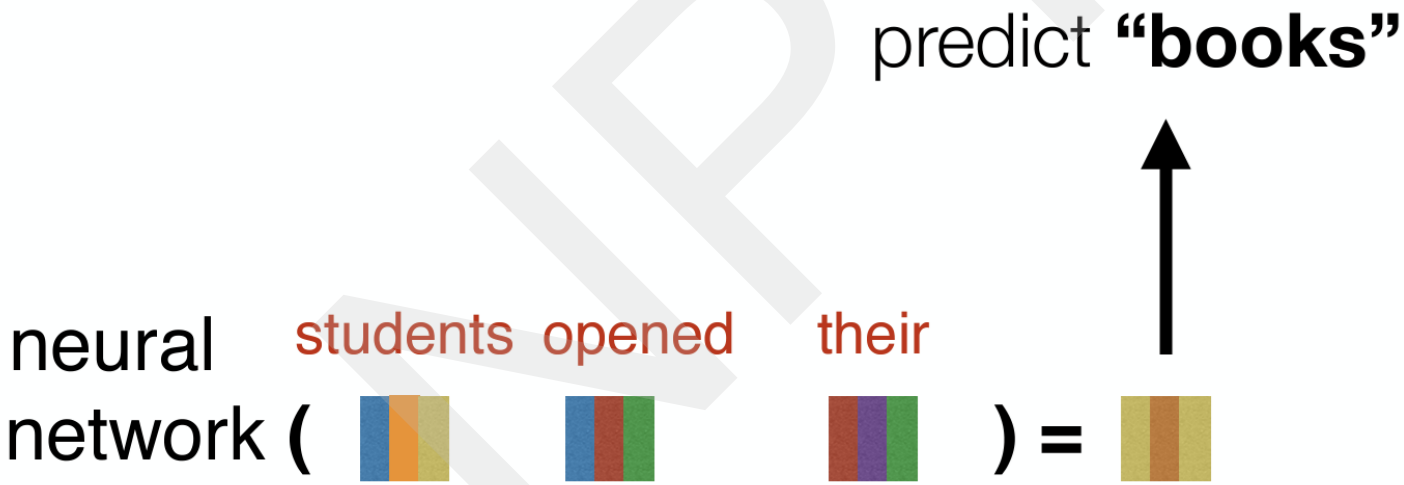
Enter Neural Networks



Source: https://people.cs.umass.edu/~miyyer/cs685/

Composing word embeddings

- Neural networks compose word embeddings into vectors for phrases, sentences, and documents
- Predict the next word from composed prefix representation

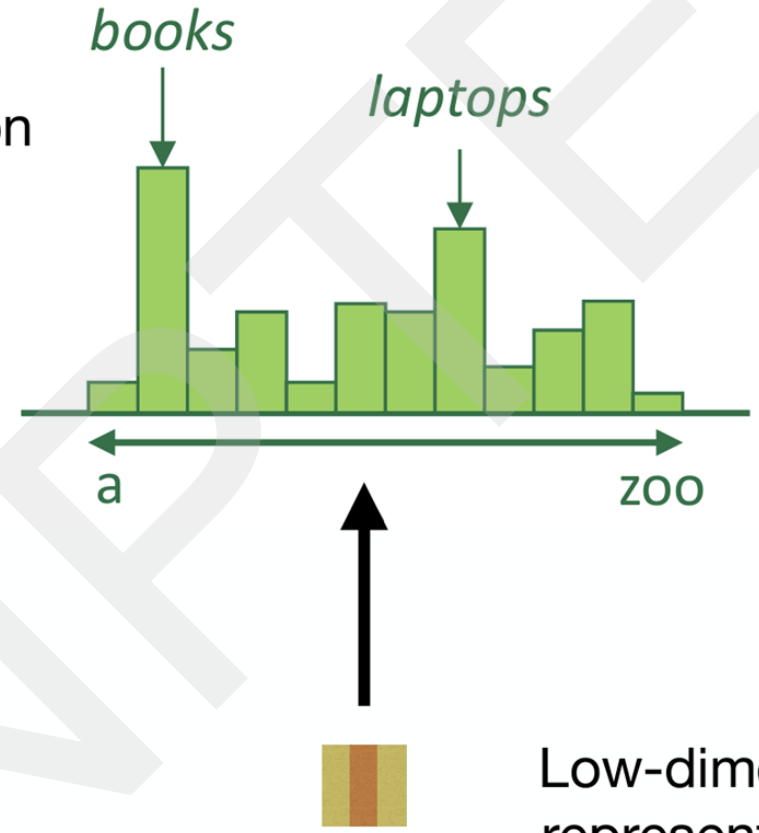


Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

$P(w_i \mid \text{vector for "students opened their"})$

Probability distribution over the entire vocabulary



Low-dimensional representation of "students opened their"

Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

Let's say our output vocabulary consists of just four words: "books", "houses", "lamps", and "stamps".

books houses lamps stamps

<0.6, 0.2, 0.1, 0.1>

We want to get a probability distribution over these four words

$\boldsymbol{x}$ = <-2.3, 0.9, 5.4>

Here's an example 3-d prefix vector

Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

W is a *weight matrix*. It contains *parameters* that we can *update* to control the final probability distribution of the next word

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

first, we'll project our 3-d prefix representation to 4-d with a matrix-vector product

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

Here's an example 3-d prefix vector

Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

intuition: each row of **W** contains *feature weights* for a corresponding word in the vocabulary

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix} \begin{matrix} \text{books} \\ \text{houses} \\ \text{lamps} \\ \text{stamps} \end{matrix}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

intuition: each dimension of $\boldsymbol{x}$ corresponds to a *feature* of the prefix

Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

$\mathbf{W}\boldsymbol{x} = \langle 1.8, -11.9, 12.9, -8.9 \rangle$

How did we compute this? Just the dot product of each row of **W** with ***x***!

$$\mathbf{W} = \begin{Bmatrix} 1.2, & -0.3, & 0.9 \\ 0.2, & 0.4, & -2.2 \\ 8.9, & -1.9, & 6.5 \\ 4.5, & 2.2, & -0.1 \end{Bmatrix}$$

$$\boldsymbol{x} = \langle -2.3, 0.9, 5.4 \rangle$$

1.2 * -2.3
+ -0.3 * 0.9
+ 0.9 * 5.4

Source: https://people.cs.umass.edu/~miyyer/cs685/

How does this happen?

Okay, so how do we go from this 4-d vector to a probability distribution?

We'll use the **softmax** function!

$$\text{softmax}(x) = \frac{e^x}{\sum_j e^{x_j}}$$

Wx = <1.8, -11.9, 12.9, -8.9>

Wx = <1.8, -1.9, 2.9, -0.9>

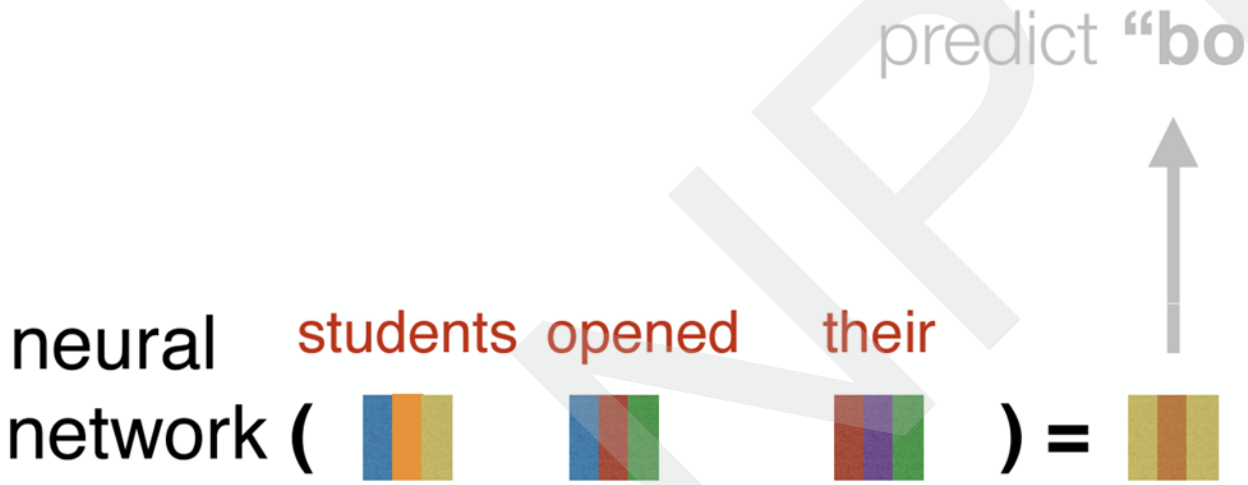
softmax(Wx) = <0.24, 0.006, 0.73, 0.02>

books houses lamps stamps

Source: https://people.cs.umass.edu/~miyyer/cs685/

Computing prefix representation

Now that we know how to predict **“books”**, let’s focus on how to compute the prefix representation ***x*** in the first place!



Source: https://people.cs.umass.edu/~miyyer/cs685/

Composition functions

input: sequence of word embeddings corresponding to the tokens of a given prefix

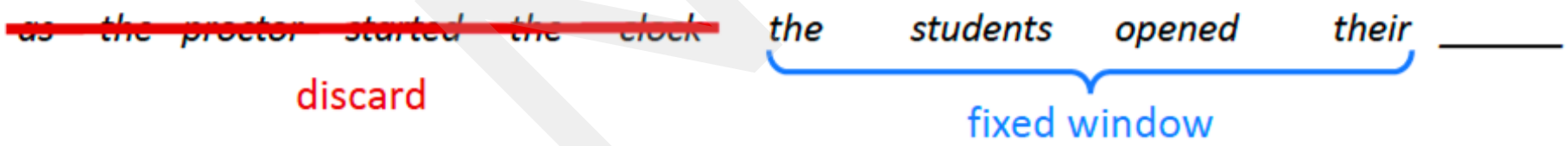
output: single vector

- Element-wise functions
  - e.g., *just sum up all of the word embeddings*!
- Concatenation
- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Transformers

Source: https://people.cs.umass.edu/~miyyer/cs685/

A fixed-window neural language model

Let's look first at *concatenation*



Source: https://web.stanford.edu/class/cs224n/

A fixed-window neural language model

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

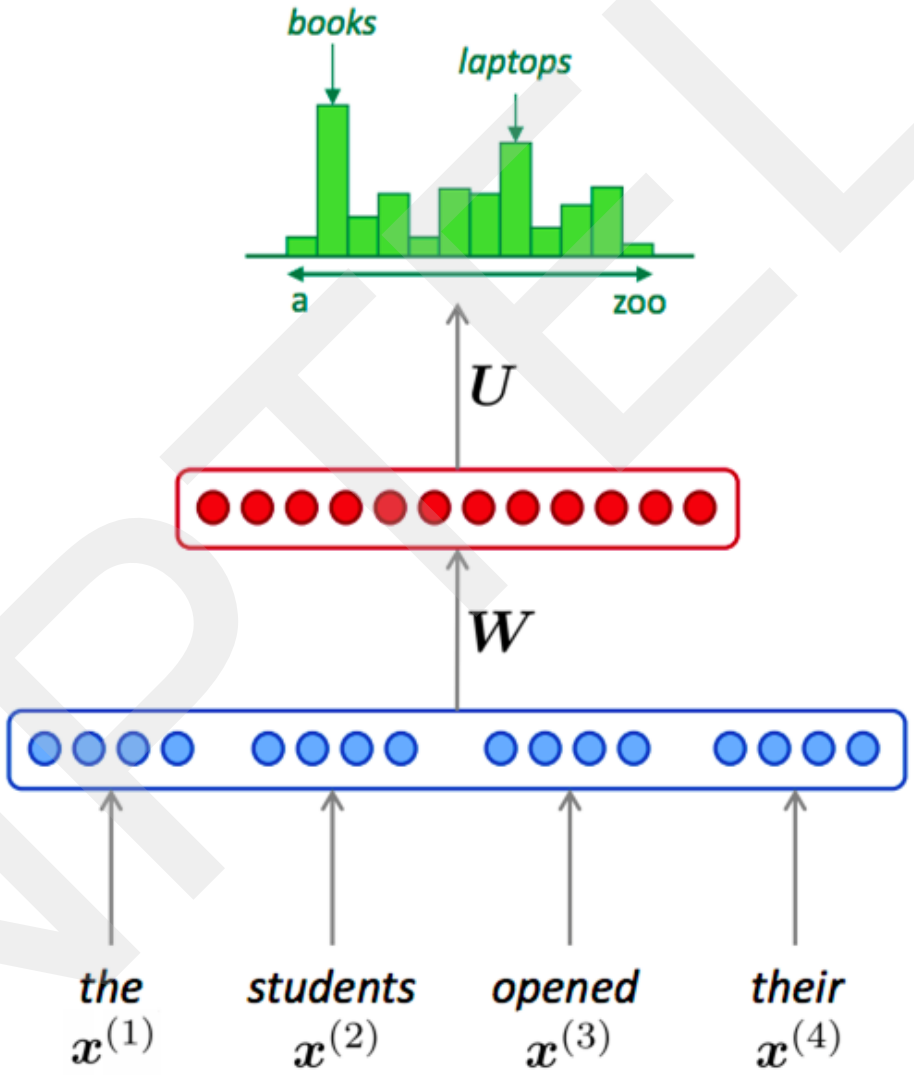
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Source: https://web.stanford.edu/class/cs224n/

A fixed-window neural language model

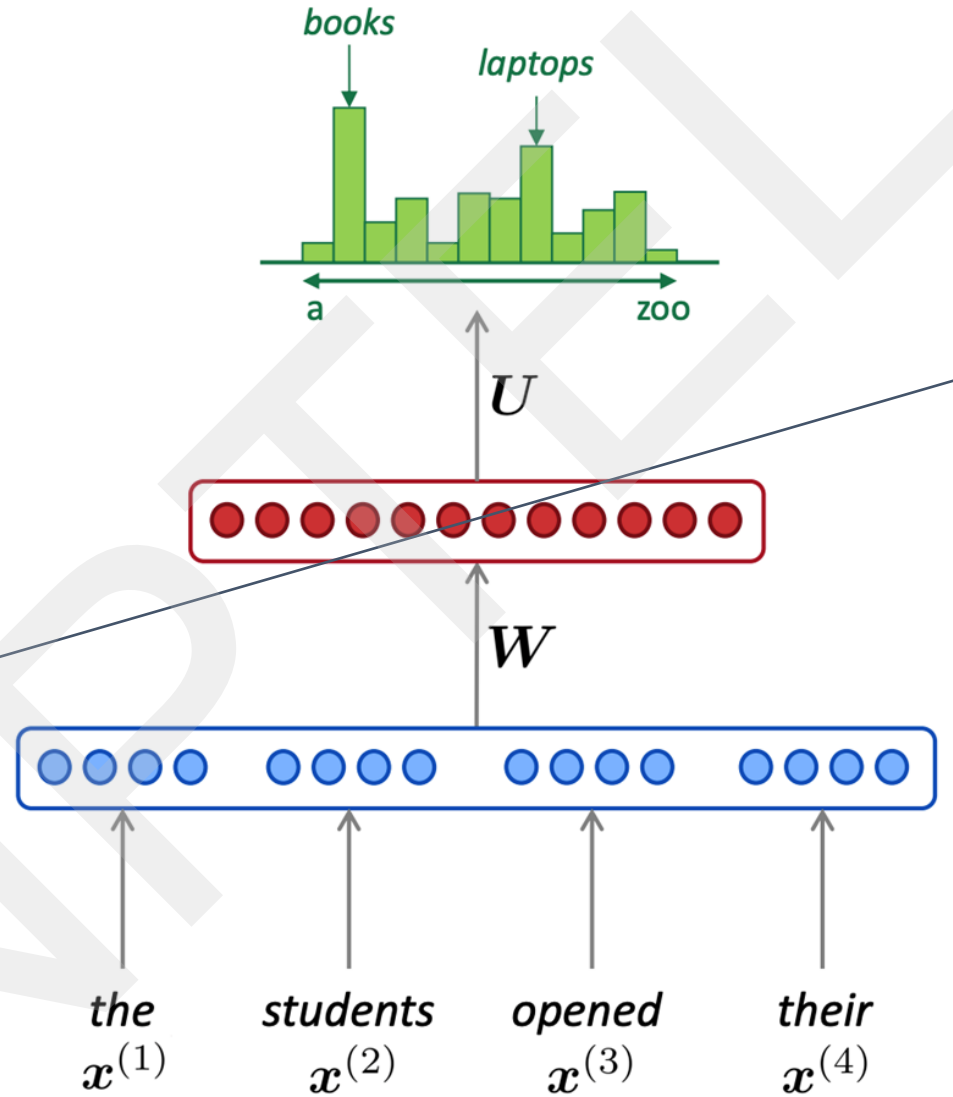
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

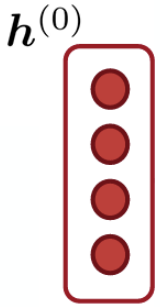
We need a neural architecture that can process *any length input*



e.g., *students have opened their*

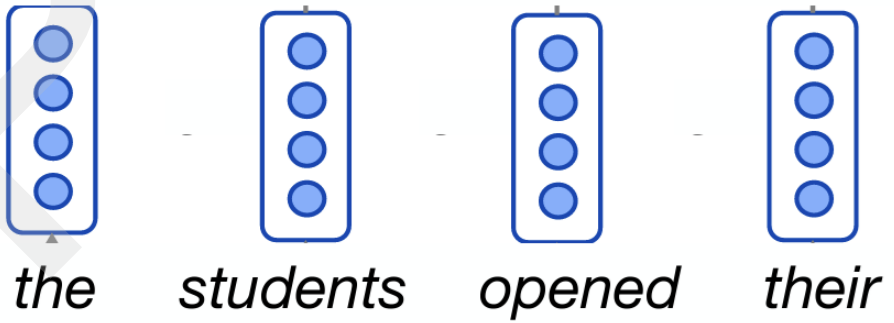
Source: https://web.stanford.edu/class/cs224n/

RNN Language Model



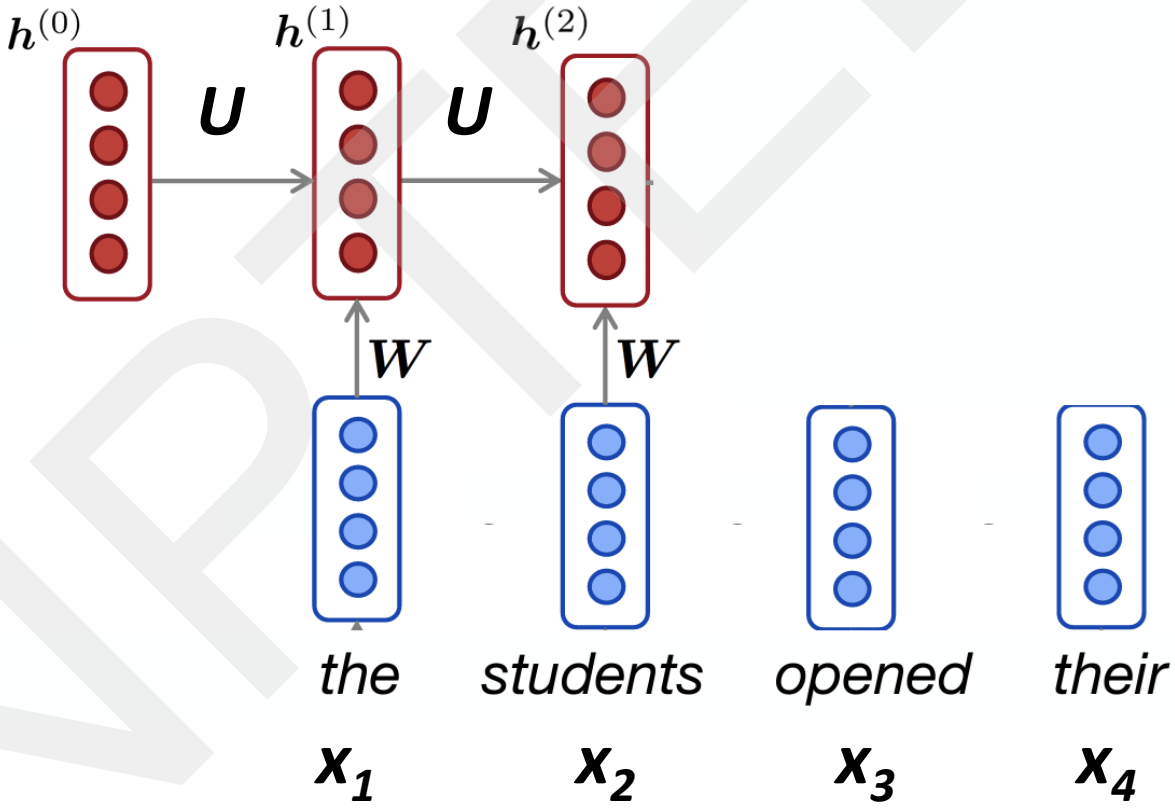
$h^{(0)}$ is initial hidden state!

$$h_t = g(Uh_{t-1} + Wx_t)$$



Source: https://web.stanford.edu/~jurafsky/slp3 .

RNN Language Model



$h^{(0)}$ is initial hidden state!

$$h_t = g(Uh_{t-1} + Wx_t)$$

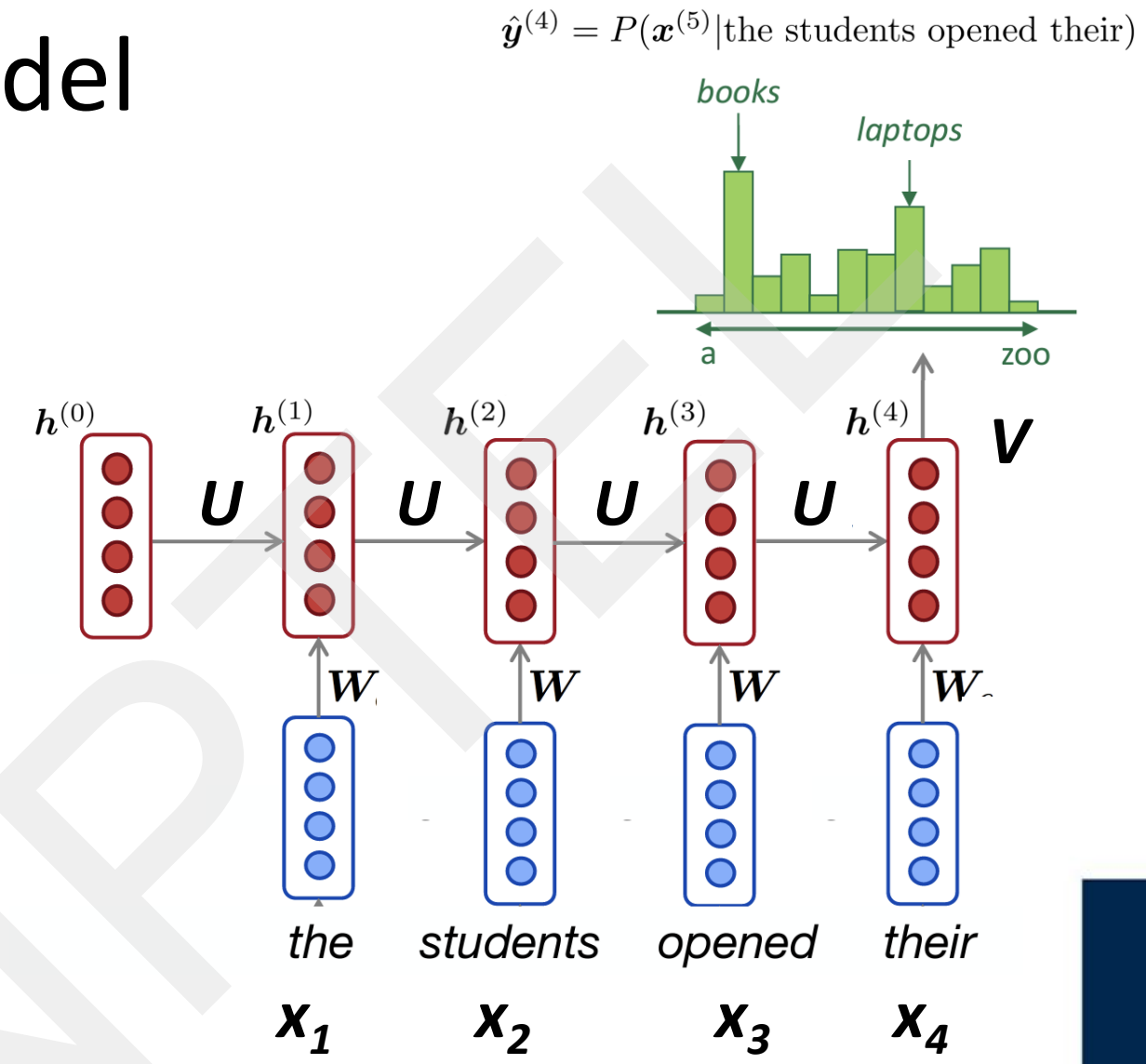
Source: https://web.stanford.edu/~jurafsky/slp3 .

RNN Language Model

why is this good?

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step *t* can (in theory) use information from many steps back
- Weights are shared across timesteps → representations are shared



Source: https://web.stanford.edu/~jurafsky/slp3 .

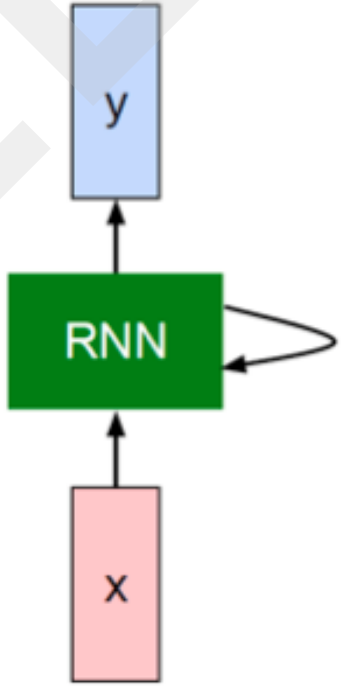
Recurrent Neural Networks (RNNs)

$$h_t = f_W(h_{t-1}, x_t)$$

new state — old state — input vector at some time step

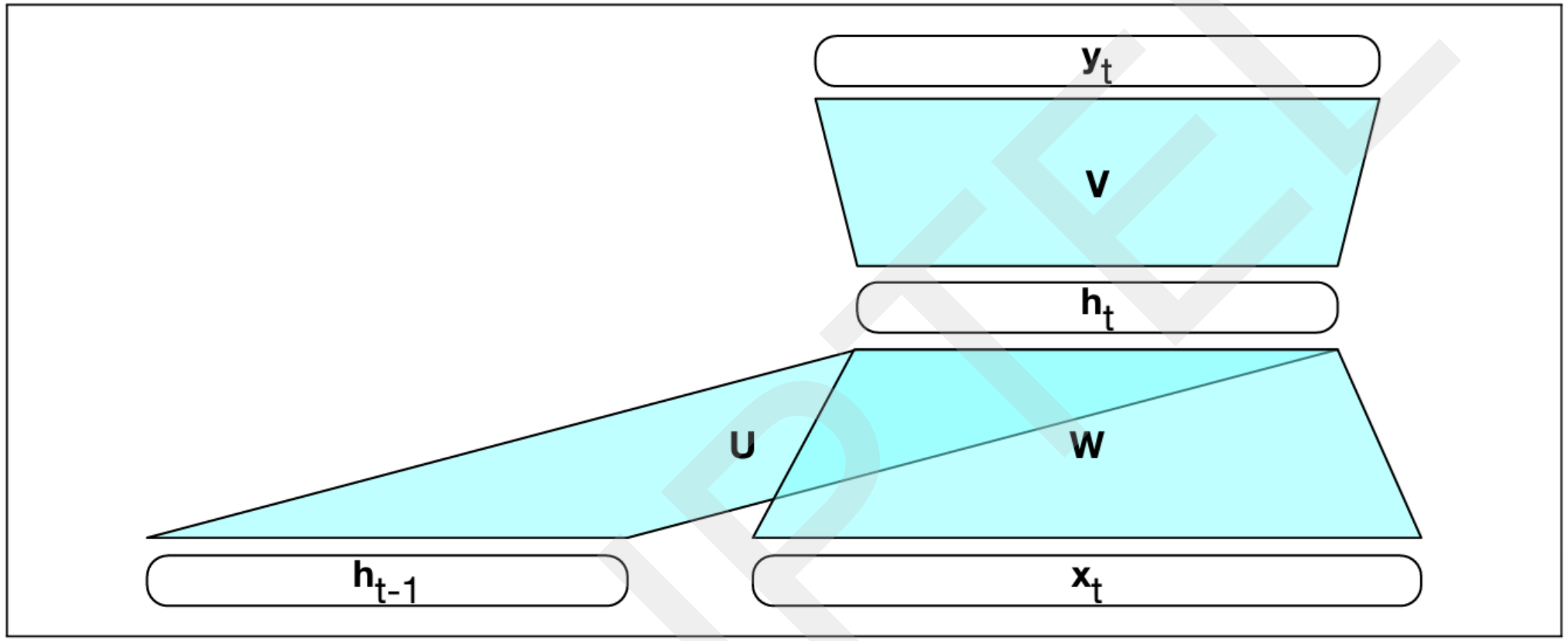
some function with parameters W

Notice: the same function and the same set of parameters are used at every time step.



We can process a sequence of vectors x by applying a recurrence formula at each step:

RNN as a feed-forward network



Source: https://web.stanford.edu/~jurafsky/slp3 .

Forward Propagation in RNN

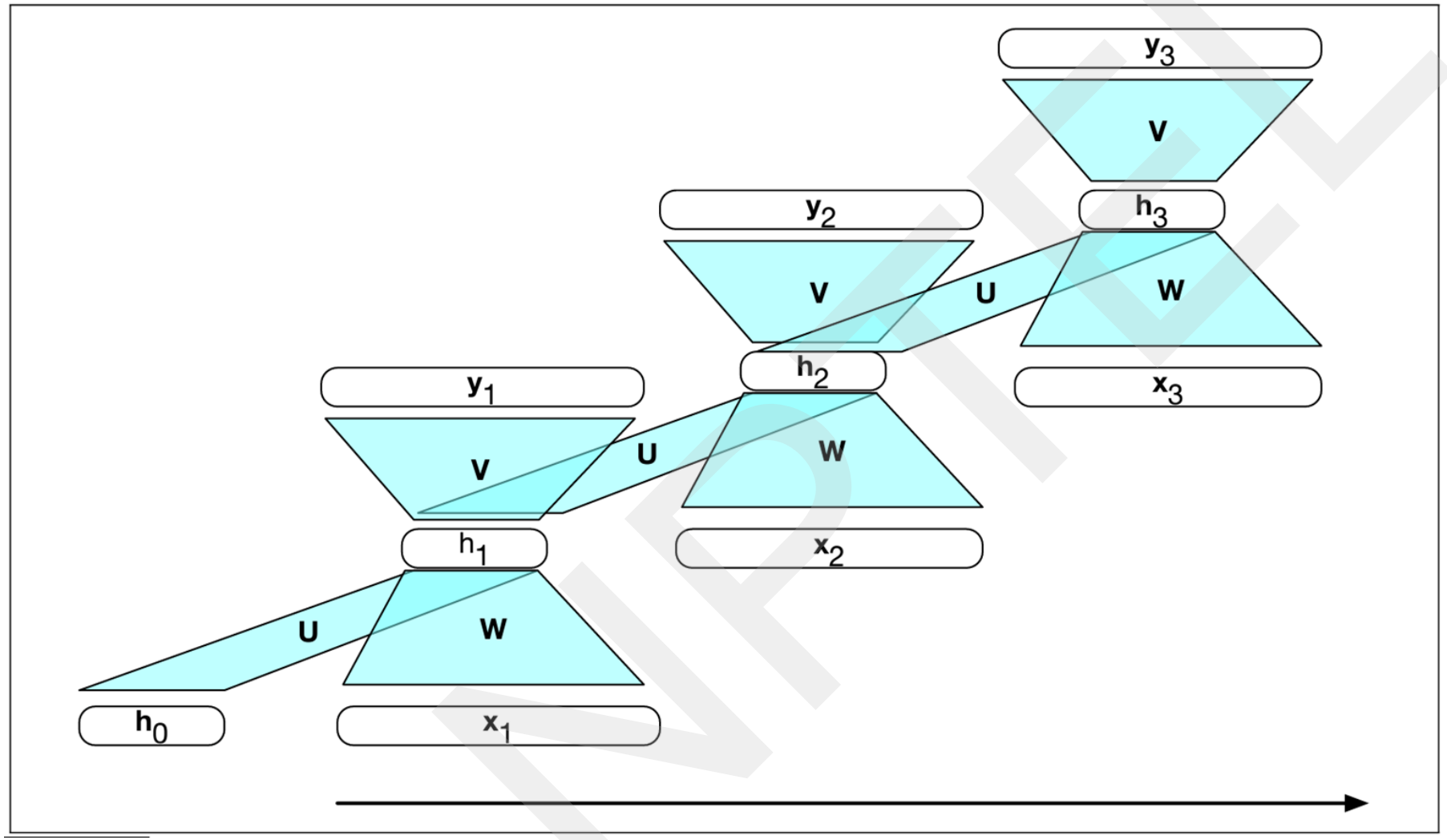
$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = softmax(Vh_t)$$

- Let the dimensions of the input, hidden and output be d_{in}, d_h and d_{out}, respectively
- The three parameter matrices: $W : d_h \times d_{in}$, $U : d_h \times d_h$, $V : d_{out} \times d_h$

Note: We are ignoring the bias terms in the above equations

RNN unrolled in time



Source: https://web.stanford.edu/~jurafsky/slp3 .

Try this problem

Consider the problem of predicting the next word from a sequence of words using an RNN. Suppose, there are overall 5 words in your vocabulary, {‘I’, ‘he’, ‘go’, ‘walk’, ‘eat’}. And each word is represented using a one-hot vector as per the order above.

Suppose the hidden layer has ‘6’ dimensions, what are the total number of parameters that you will have to learn? Please ignore the bias terms for this problem.

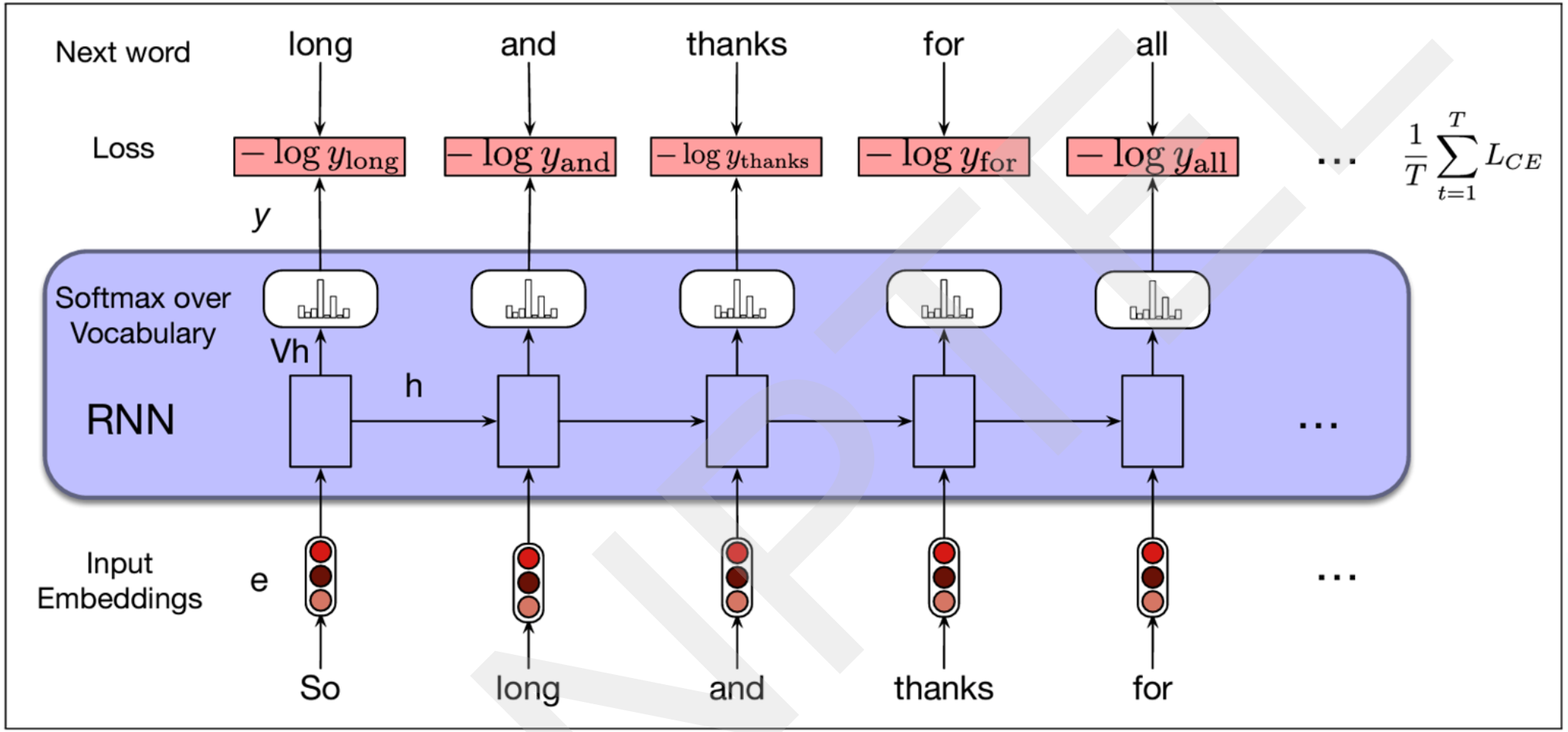
Training an RNN LM

- To train RNN LM, we use self-supervision (or self-training)
- We take a corpus of text as training material
- At each time step t, we ask the model to predict the next word

Why is it called self-supervision?

- We do not add any gold data, the natural sequence of words is its own supervision!
- We simply train the model to minimize the error in predicting the true next word in the training sequence

Training an RNN LM



Source: https://web.stanford.edu/~jurafsky/slp3 .

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 8]


THANK YOU
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 17 : RNN Applications: Text Generation, Sequence Labeling, Text Classification

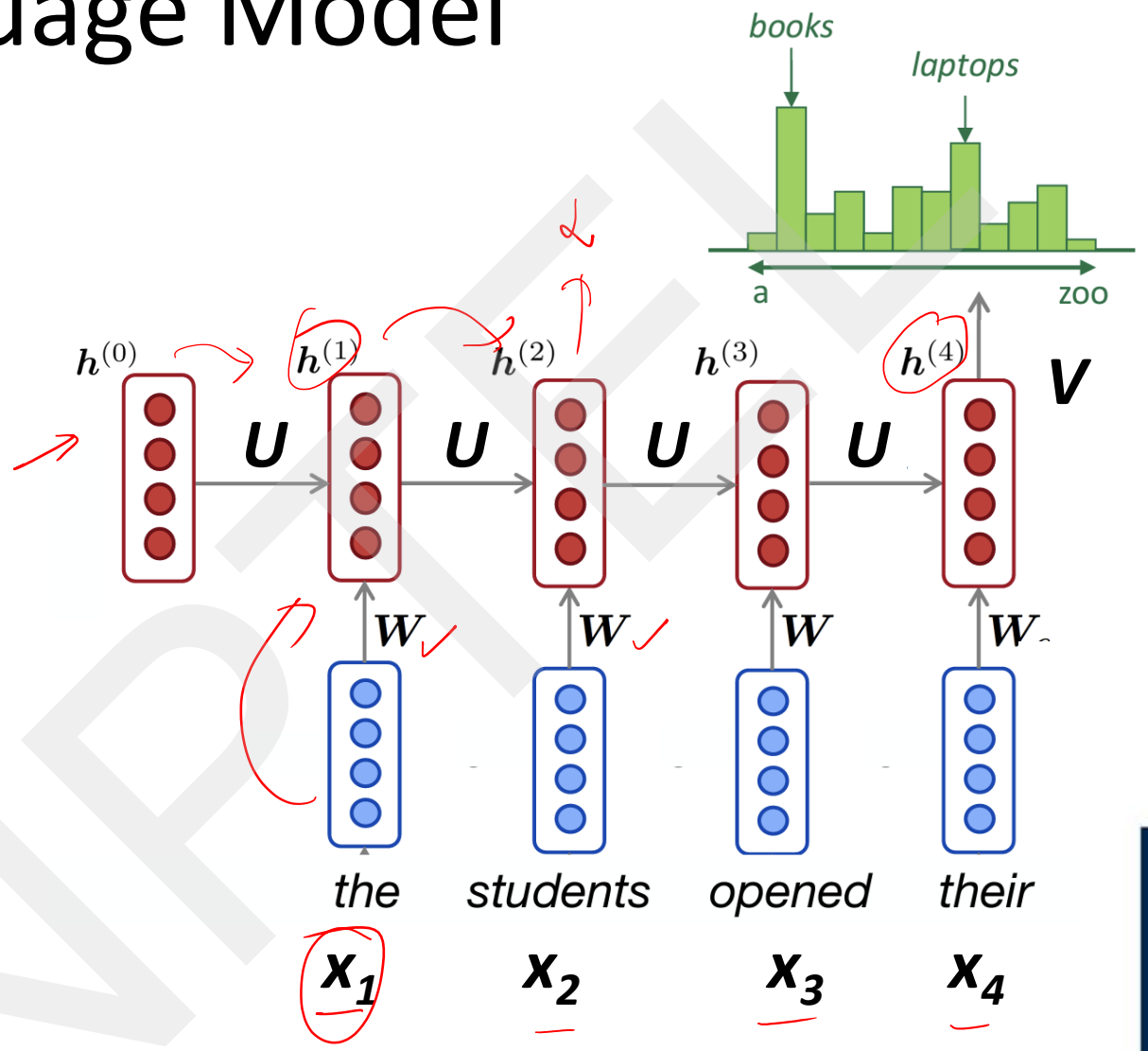
PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- RNNs for Text Generation
- RNNs for Sequence Labeling
- RNNs for Text Classification

Recap: RNN Language Model

$\hat{y}^{(4)} = P(x^{(5)}|\text{the students opened their})$



why is this good?

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step *t* can (in theory) use information from many steps back
- Weights are shared across timesteps → representations are shared

BPTT

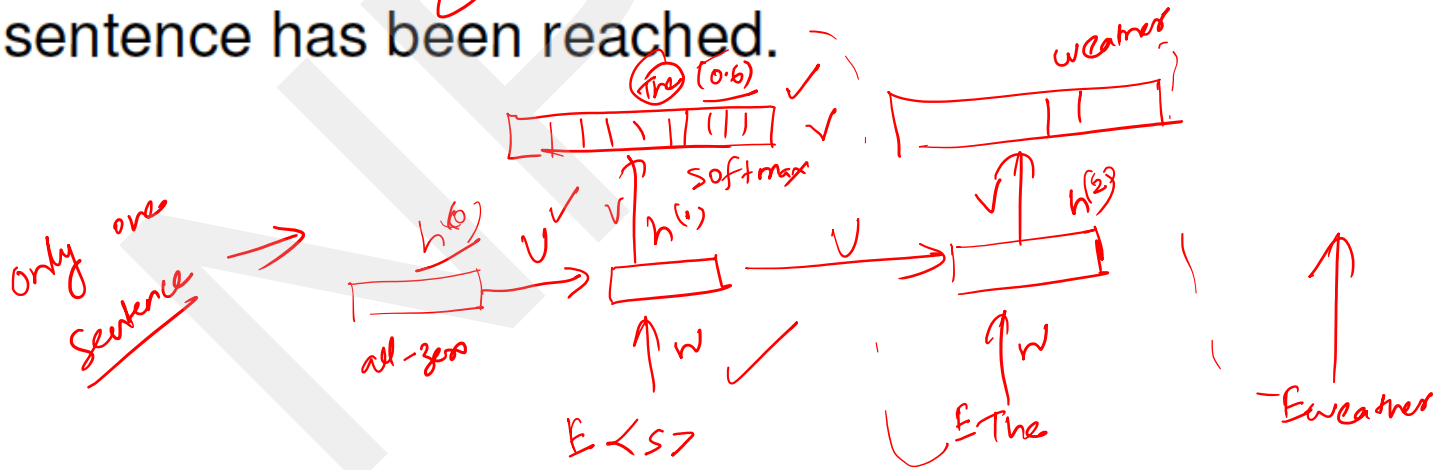
Source: https://web.stanford.edu/~jurafsky/slp3 .

Generating text with an RNN Language Model

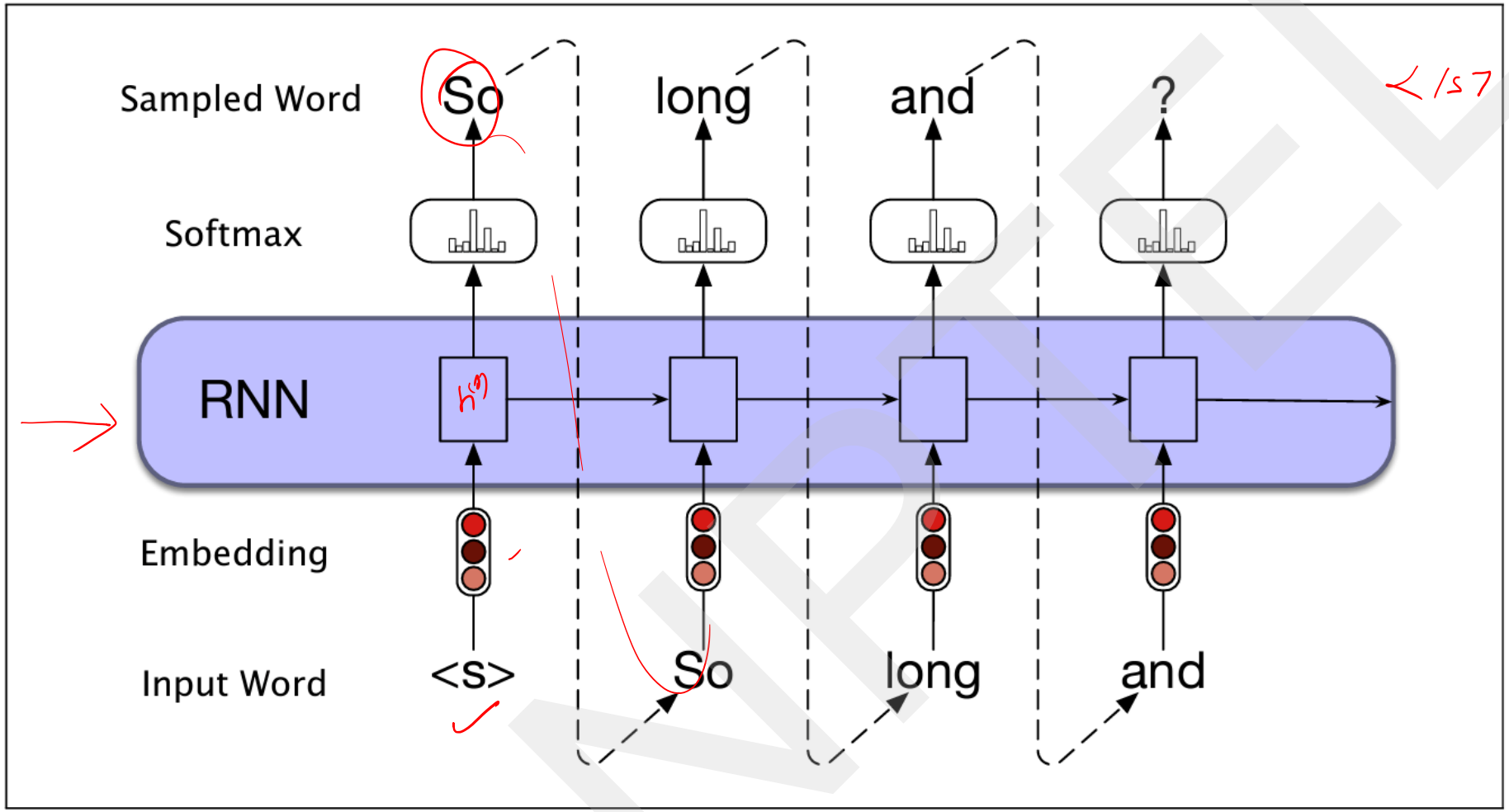
- RNN-based language models can be used for language generation (and hence, for machine translation, dialog, etc.)
- A language model can incrementally generate words by repeatedly sampling the words conditioned on the previous choices – also known as *autoregressive generation*.

Autoregressive Generation with RNNs

- All your parameters have already been trained.
- Start with a special begin of sentence token <s> as input
- Through forward propagation, obtain the probability distribution at the output, and sample a word
- Feed the word as input at the next time-step (its word vector)
- Continue generating until the end of sentence token is sampled, or a fixed length of the sentence has been reached.

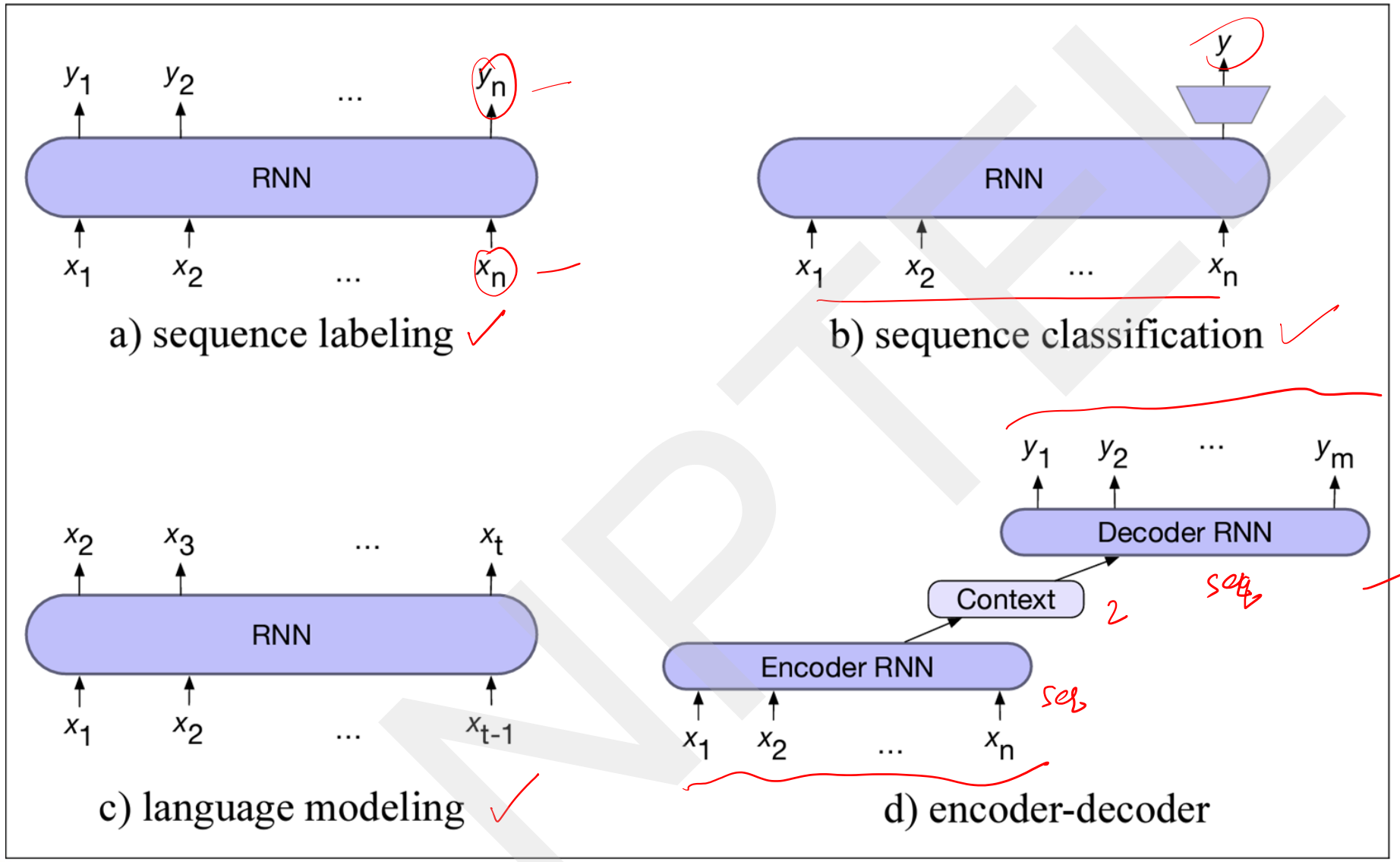


Autoregressive Generation with RNNs



Source: https://web.stanford.edu/~jurafsky/slp3 .

RNNs can be used for various applications



Source: https://web.stanford.edu/~jurafsky/slp3 .

RNNs for Sequence Labeling

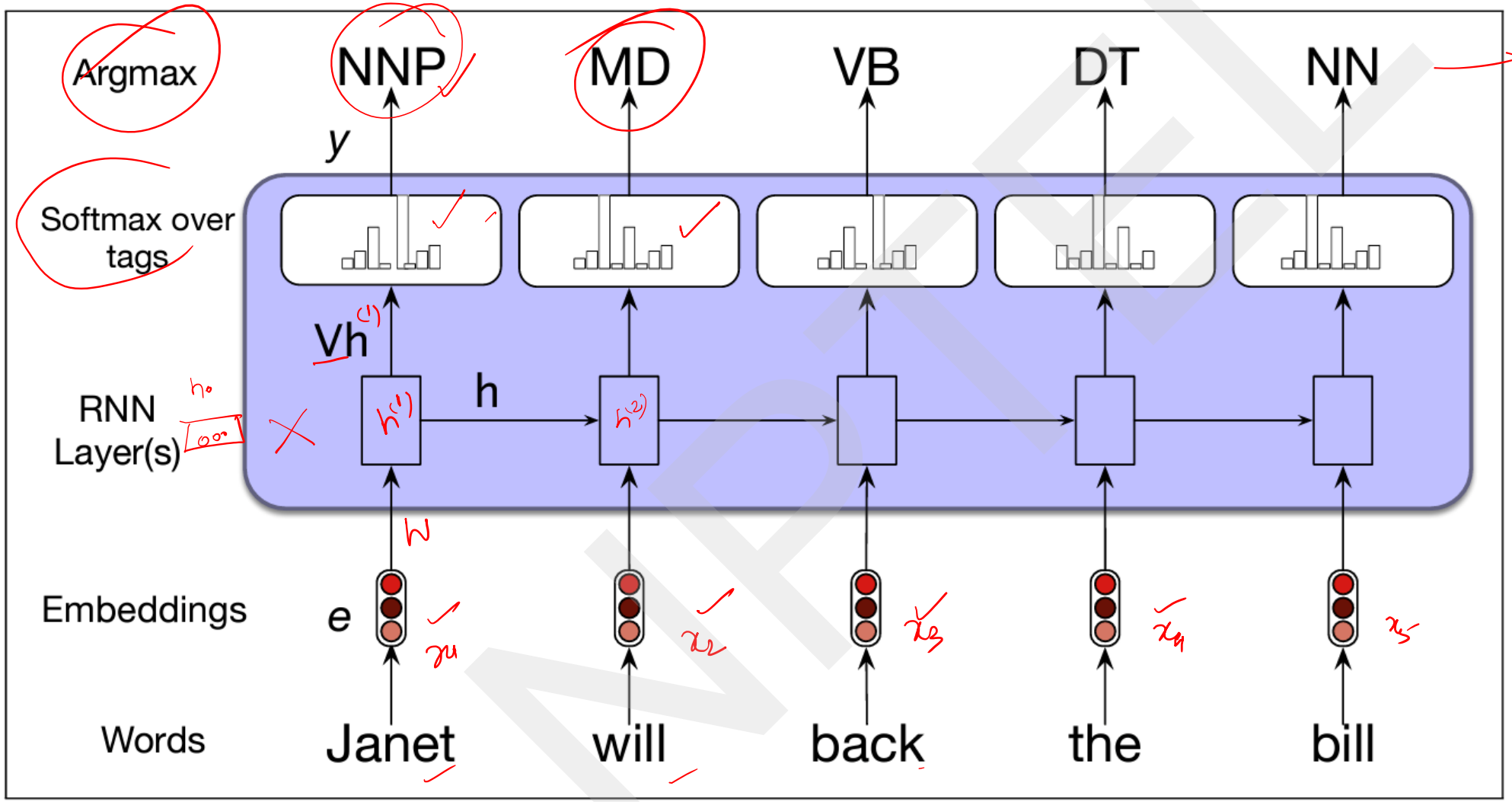
Task

Assign a label chosen from a small fixed set of labels to each element of the sequence

- Inputs: Word embeddings
- Outputs: Tag probabilities generated by the softmax layer

Source: https://web.stanford.edu/~jurafsky/slp3 .

RNNs for Sequence Labeling: POS Tagging



Tag Sequence

Training:
Model params are yet to be trained
Cross Entropy

Inference:
Params are already frozen

Source: https://web.stanford.edu/~jurafsky/slp3 .

Named Entities

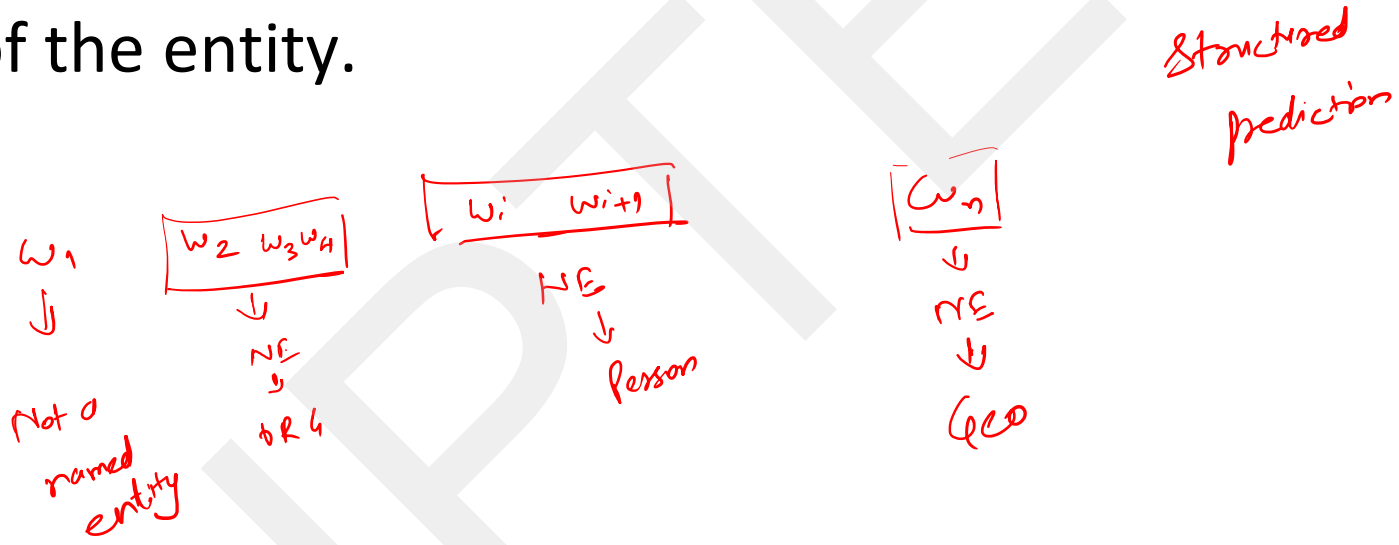
- **Named entity**, in its core usage, means anything that can be referred to with a proper name. Most common 4 tags:
  - PER (Person): “Marie Curie”
  - LOC (Location): “New York City”
  - ORG (Organization): “Stanford University”
  - GPE (Geo-Political Entity): "Boulder, Colorado"
- Often multi-word phrases
- But the term is also extended to things that aren't entities:
  - dates, times, prices

Source: https://web.stanford.edu/~jurafsky/slp3 .

Named Entity tagging

The task of named entity recognition (NER):

- find spans of text that constitute proper names
- tag the type of the entity.



Source: https://web.stanford.edu/~jurafsky/slp3 .

NER output

Citing high fuel prices, [ORG **United Airlines**] said [TIME **Friday**] it has increased fares by [MONEY **$6**] per round trip on flights to some cities also served by lower-cost carriers. [ORG **American Airlines**], a unit of [ORG **AMR Corp.**], immediately matched the move, spokesman [PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAL Corp.**], said the increase took effect [TIME **Thursday**] and applies to most routes where it competes against discount carriers, such as [LOC **Chicago**] to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

Source: https://web.stanford.edu/~jurafsky/slp3 .

Why NER?

Sentiment analysis: consumer's sentiment toward a particular company or person?

Aspect-based sentiment analysis

Question Answering: answer questions about an entity?

Information Extraction: Extracting facts about entities from text.

Why NER is hard

1) Segmentation
   - In POS tagging, no segmentation problem since each word gets one tag.
   - In NER we have to find and segment the entities!

2) Type ambiguity

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

Source: https://web.stanford.edu/~jurafsky/slp3 .

BIO Tagging

How can we turn this structured problem into a sequence problem like POS tagging, with one label per word?

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Source: https://web.stanford.edu/~jurafsky/slp3 .

BIO Tagging

[PER Jane Villanueva] of [ORG United] , a unit of [ORG United Airlines Holding] , said the fare applies to the [LOC Chicago] route.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

B - Tag

I - Tag

O

Now we have one tag per token!!!

B-LOC

B-LOC I-LOC

B-LOC I-LOC I-LOC

Source: https://web.stanford.edu/~jurafsky/slp3 .

BIO Tagging

B: token that *begins* a span

I: tokens *inside* a span

O: tokens outside of any span

# of tags (where n is #entity types):

1 O tag,

n B tags,

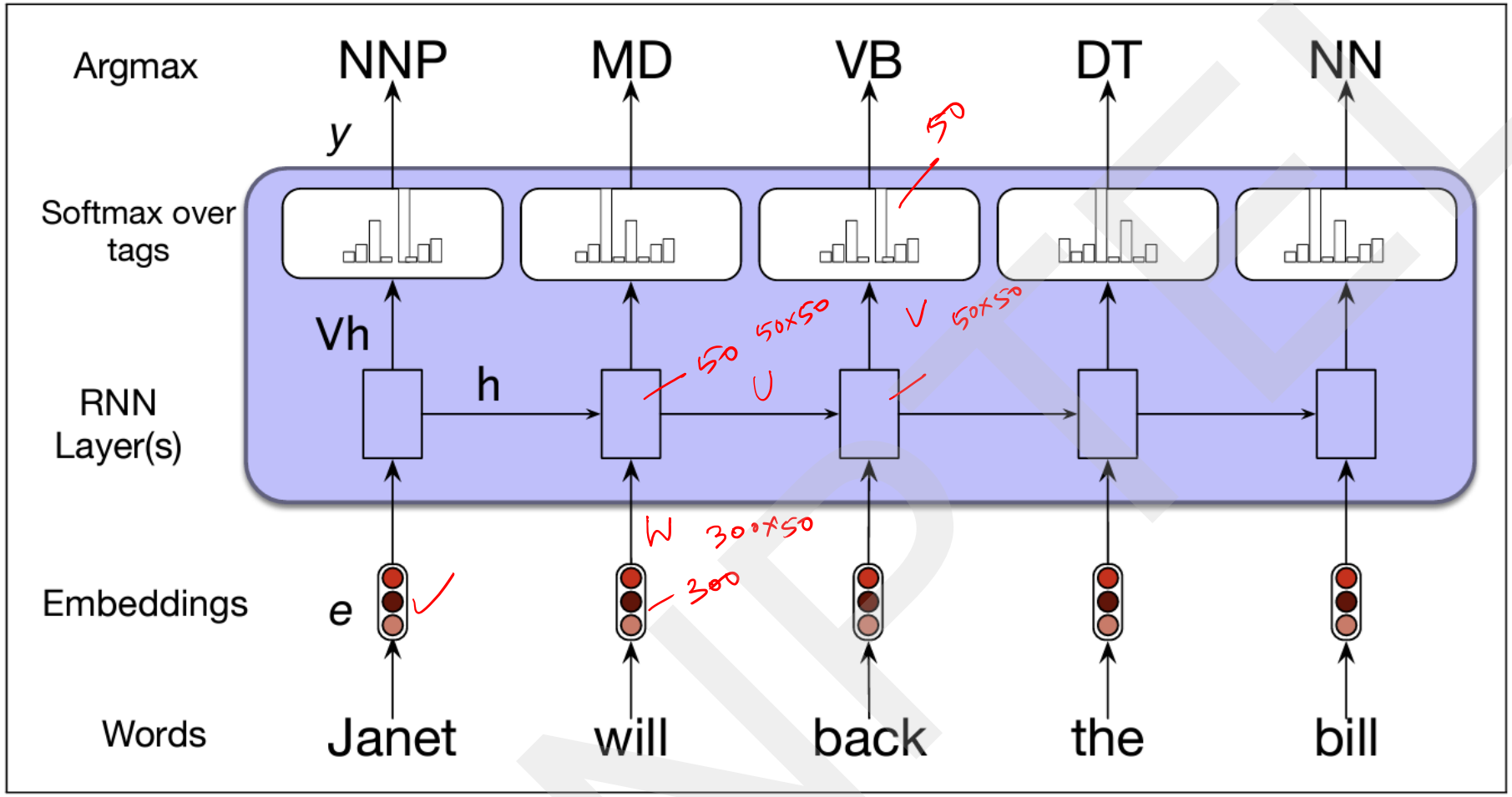
n I tags

total of *2n+1*

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

Source: https://web.stanford.edu/~jurafsky/slp3 .

RNNs for Sequence Labeling



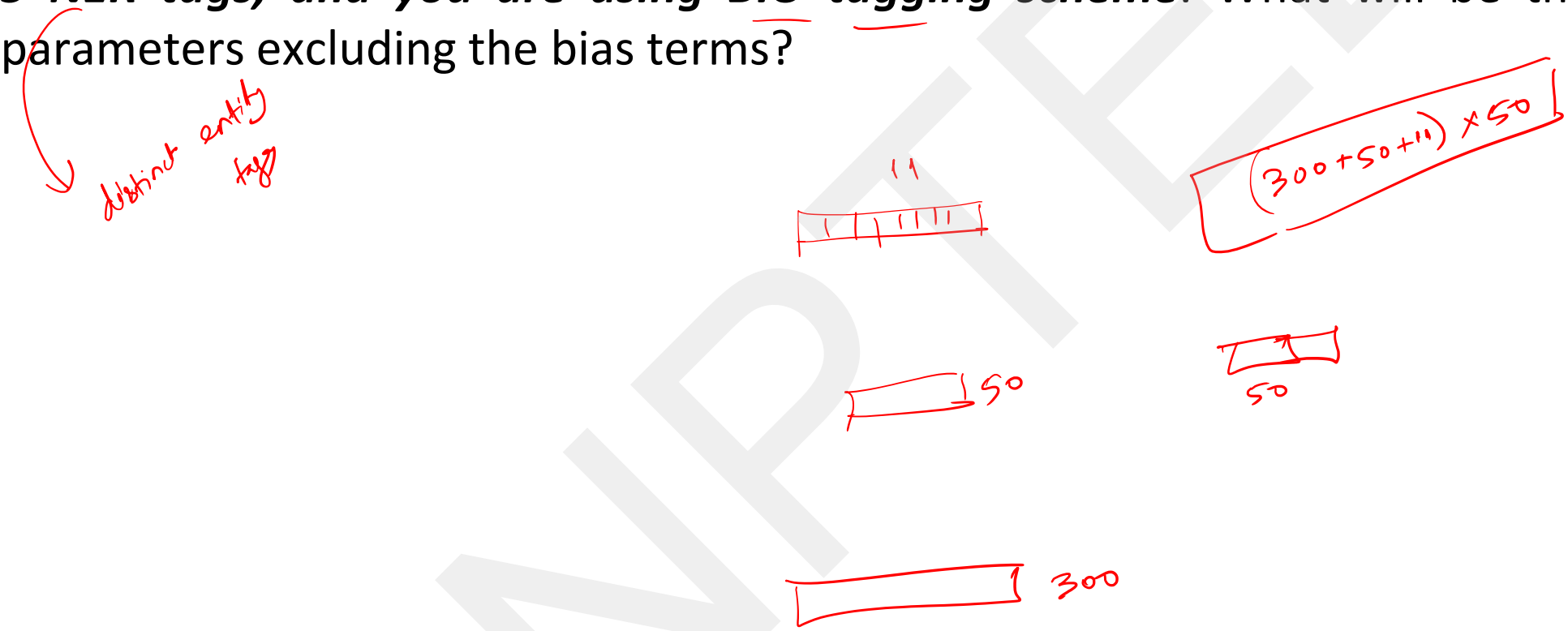
Suppose you are using 300-dim word embeddings, 50-dim hidden vector, and you have 50 POS tags. What will be the number of parameters excluding the bias terms?

(300 + 50 + 50) × 50

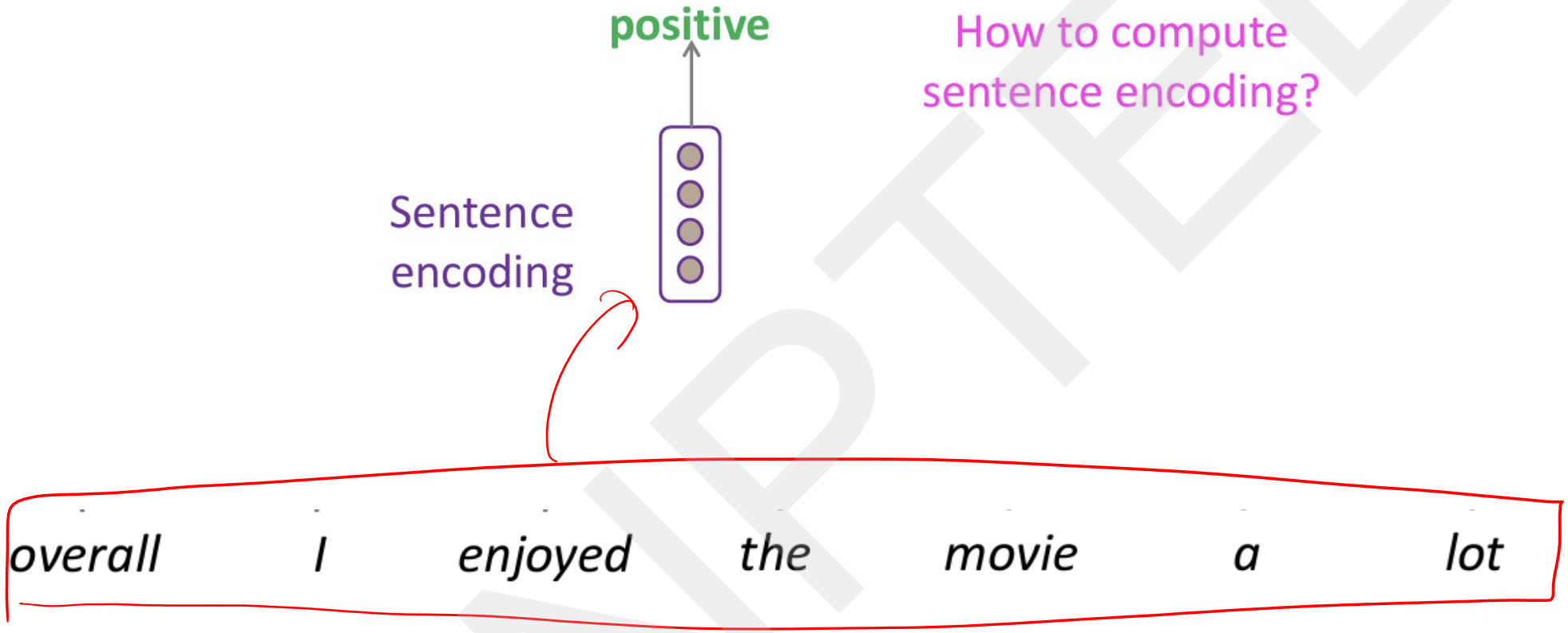
400 × 50 = 20000

RNNs for Sequence Labeling

Suppose you are using 300-dim word embeddings, 50-dim hidden vector, and you have ***5 NER tags, and you are using BIO tagging scheme***. What will be the number of parameters excluding the bias terms?

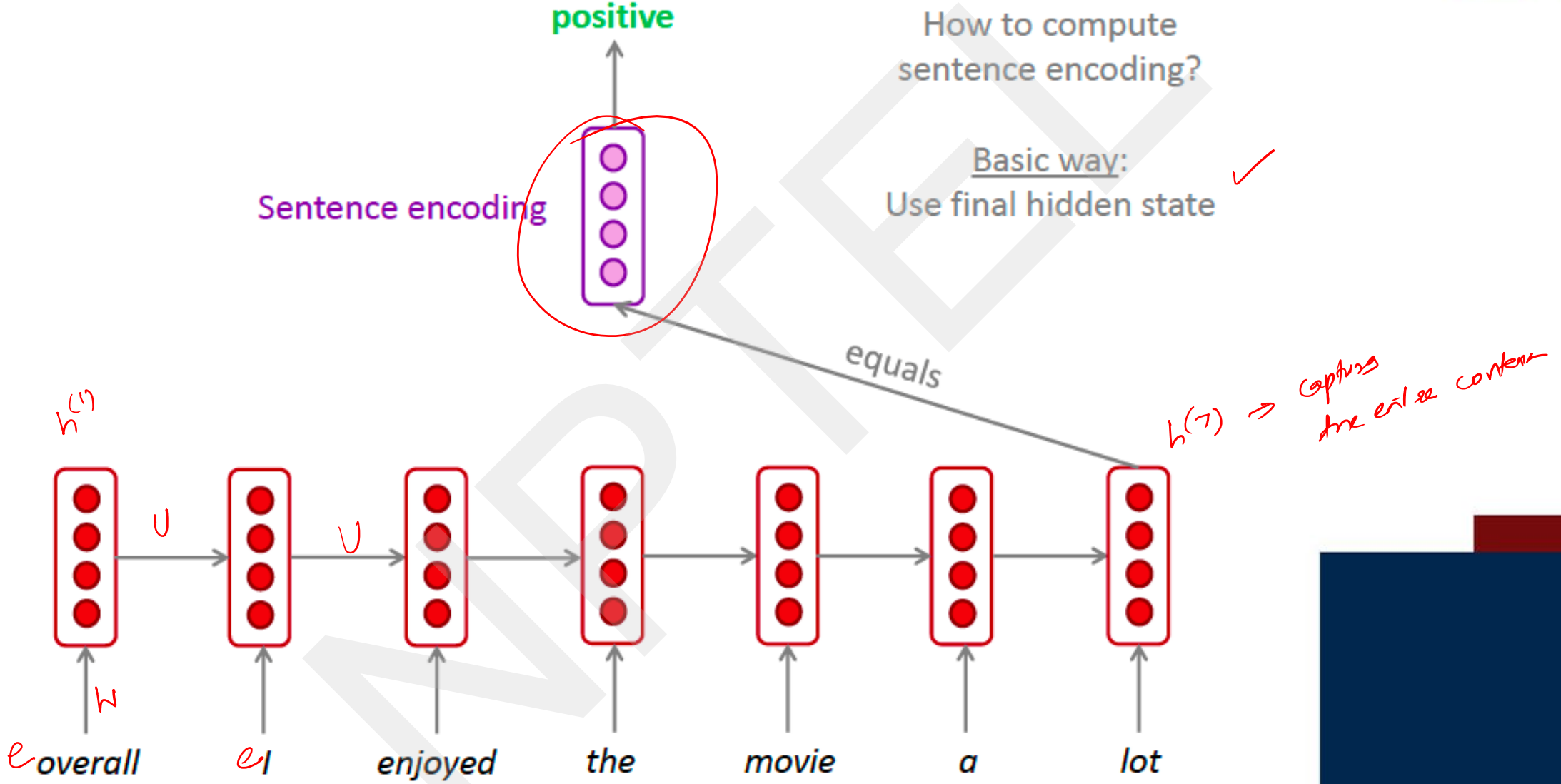


How to use RNNs for Text classification?



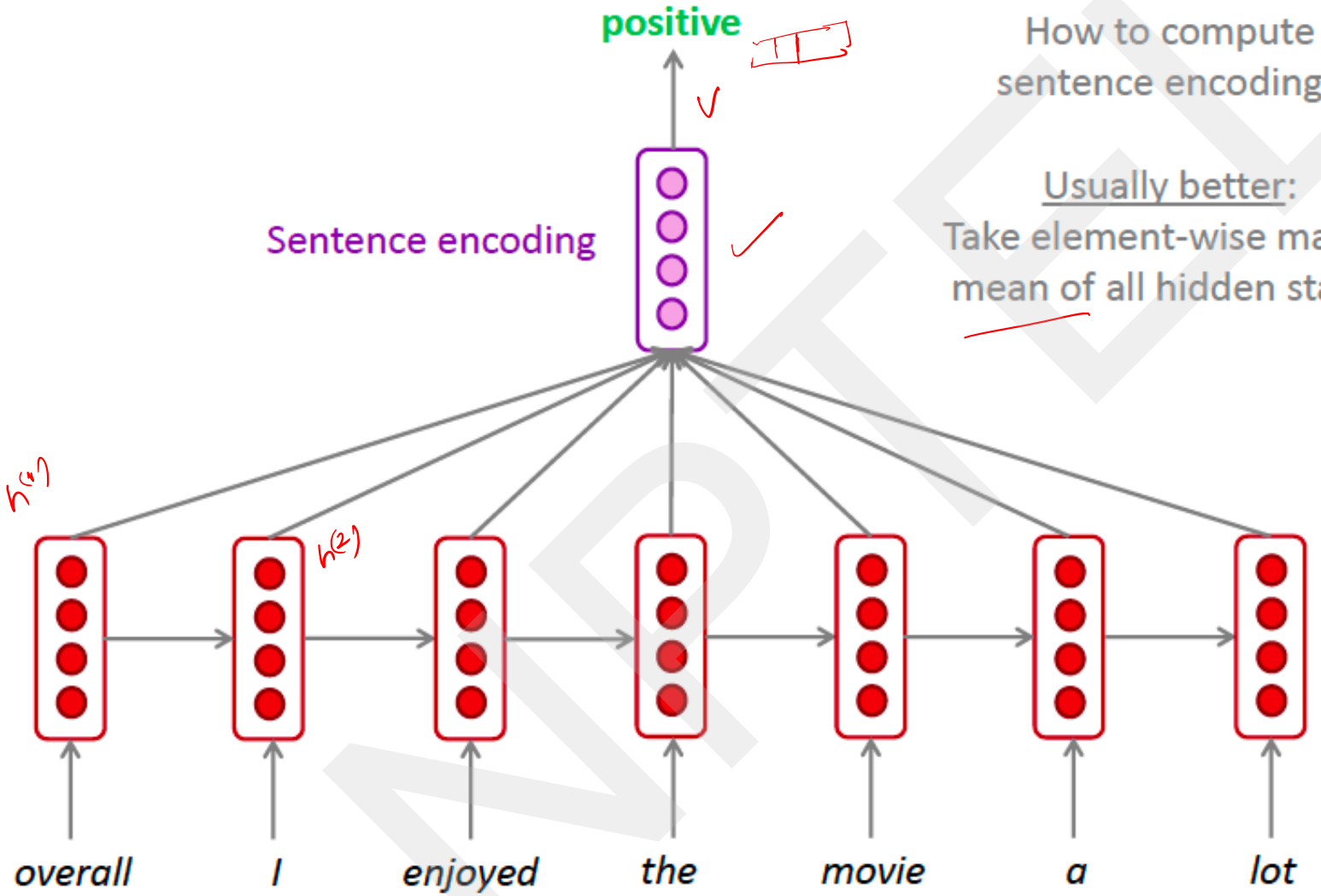
Source: https://web.stanford.edu/class/cs224n/

RNNs for Sentence Classification



Source: https://web.stanford.edu/class/cs224n/

RNNs for Sentence Classification



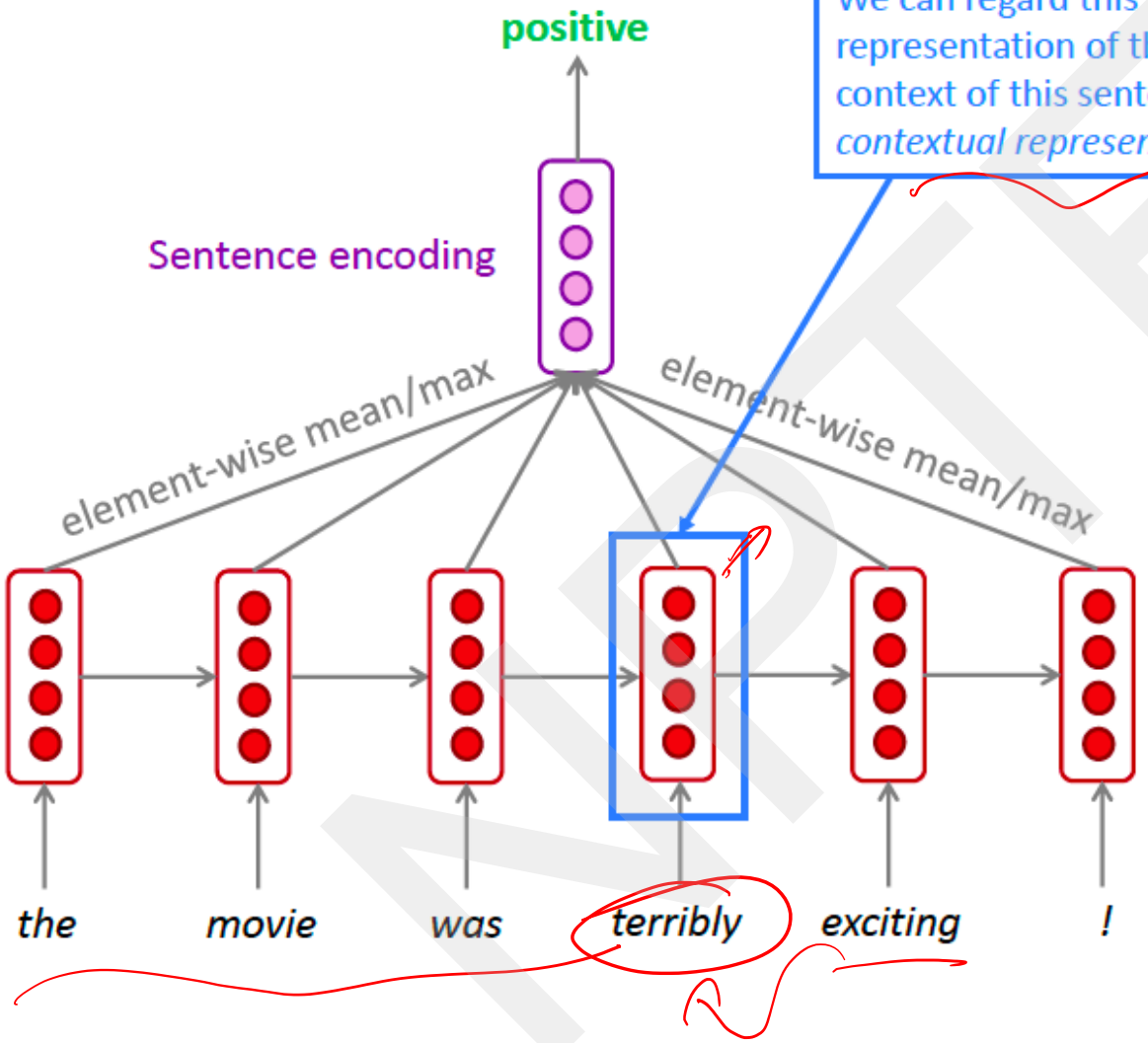
How to compute sentence encoding?

Usually better:
Take element-wise max or mean of all hidden states

Source: https://web.stanford.edu/class/cs224n/

Bidirectional RNNs: motivation for sentiment classification

Task: Sentiment Classification



We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Source: https://web.stanford.edu/class/cs224n/

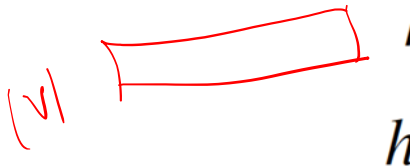
Bidirectional RNNs: Formally

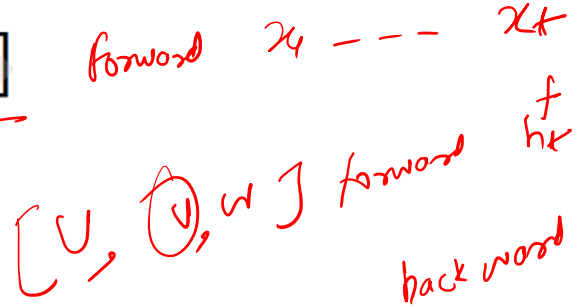
- RNN makes use of information from left (prior) context to predict at time t
- In many applications, the entire sequence is available; so it makes sense to also make use of the right context to predict at time t
- Bidirectional RNNs combine two independent RNNs, one where the input is processed from left to right (forward RNN), and another from end to the start (backward RNN).

$$h_t^f = RNN_{forward}(x_1, \ldots, x_t)$$

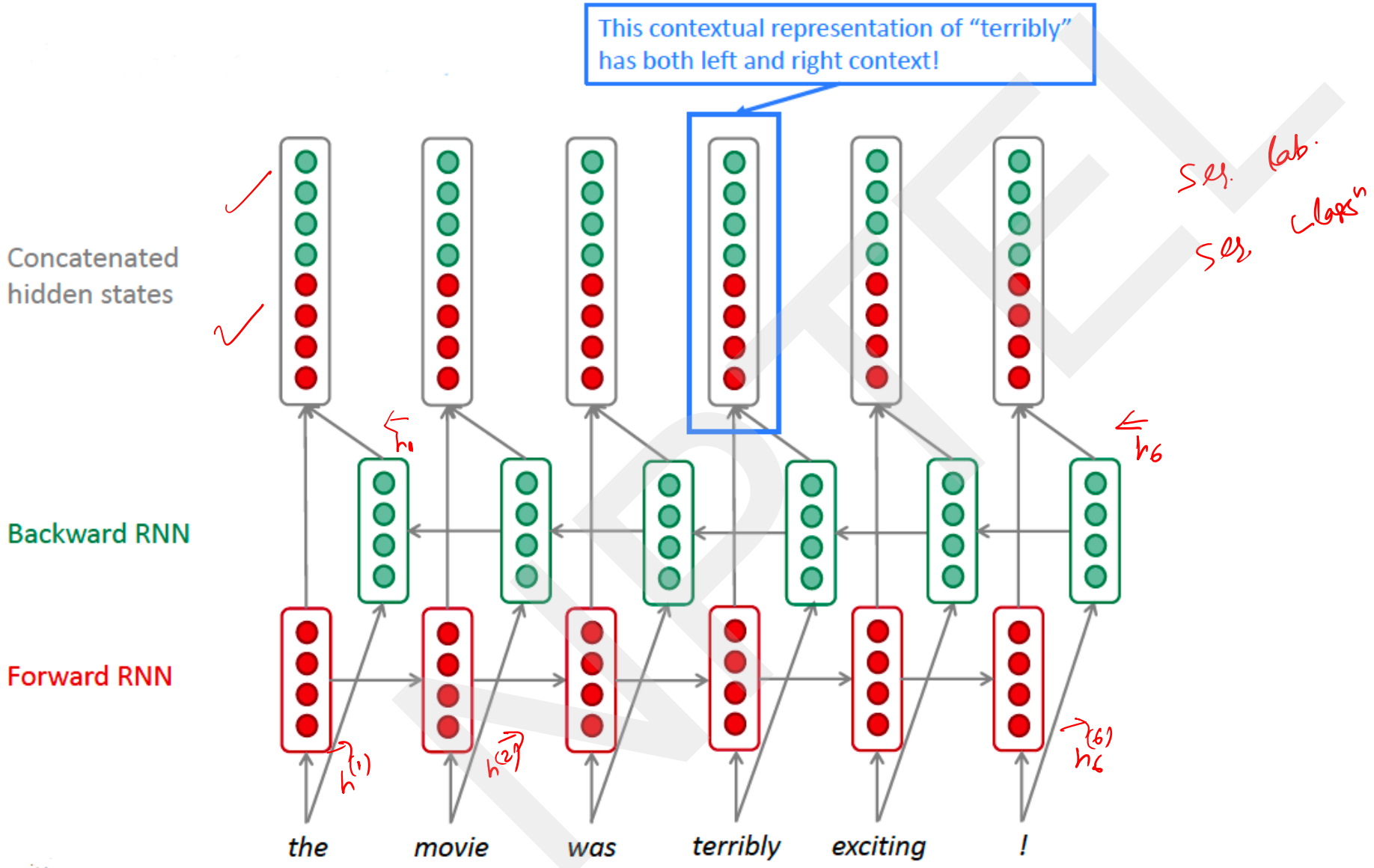
$$h_t^b = RNN_{backward}(x_n, \ldots, x_t)$$

$$h_t = [h_t^f; h_t^b]$$





Bidirectional RNNs



Source: https://web.stanford.edu/class/cs224n/

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 8]


THANK YOU
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 18 : RNN for Sequence to Sequence

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Sequence to Sequence: Machine Translation
- Sequence to Sequence (Seq2Seq using RNN Encoder-Decoder)
- Attention Mechanism

Try this problem

Suppose you are using Bi-LSTM for NER problem, and there are 3 NER tags: {PER, LOC, ORG}, and you are using BIO tagging scheme. Assume that each word has a 300-dimension embedding, and the hidden state for the forward and backward LSTMs are 200 and 100 dimensions, respectively. How many total parameters will need to be trained? Ignore the bias terms.

Try this problem contd…

Now, suppose you also want to train the word embeddings for a vocabulary of 40,000. How many additional parameters will need to be trained?

An example of Seq2Seq: Machine Translation

Machine Translation (MT) is the task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language)

x: *L'homme est né libre, et partout il est dans les fers*

y: *Man is born free, but everywhere he is in chains*

- Rousseau

Sequence to sequence is versatile

Many NLP tasks can be phrased as sequence-to-sequence

- Summarization (long text $\rightarrow$ short text)
- Dialogue (previous utterances $\rightarrow$ next utterance)

Sequence-to-sequence architecture

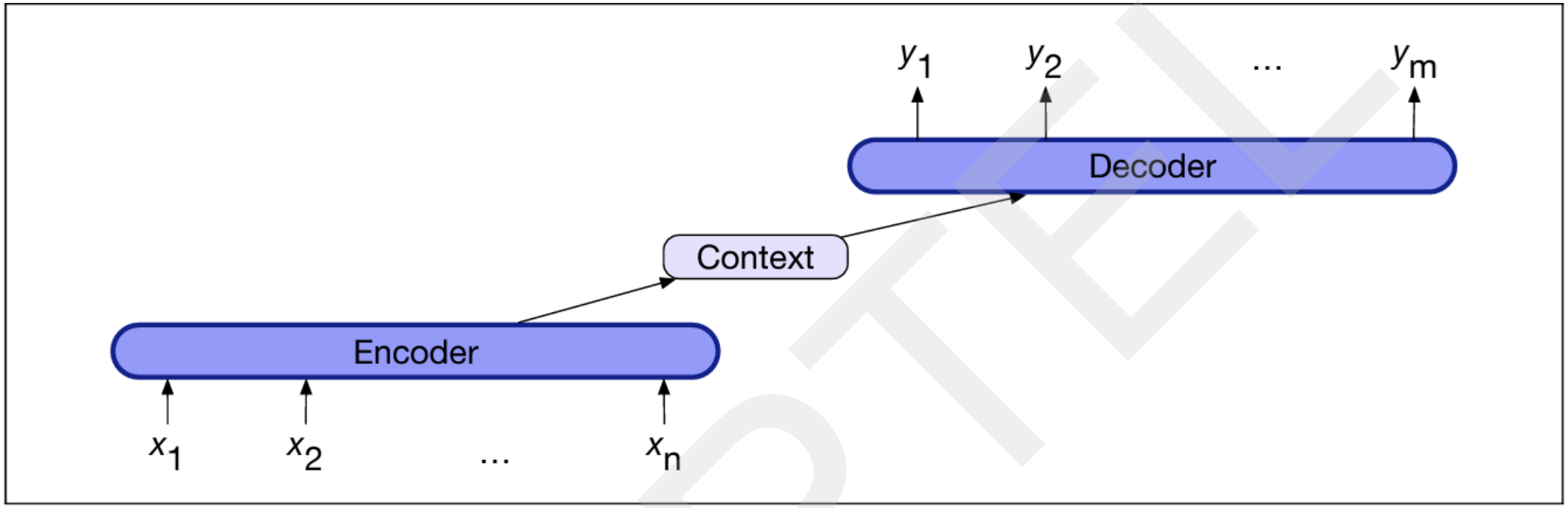
Also known as encoder-decoder architecture

- Input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$
- Output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$
- Encoder (reader/input) RNN: Emits the context C, a vector summarizing the input sequence, usually as a simple function of its final hidden state
- Decoder (writer/output) RNN: Is conditioned on the context C to generate the output sequence.

What is the innovation?

The lengths n_x and n_y can vary from each other

Encoder-Decoder Architecture

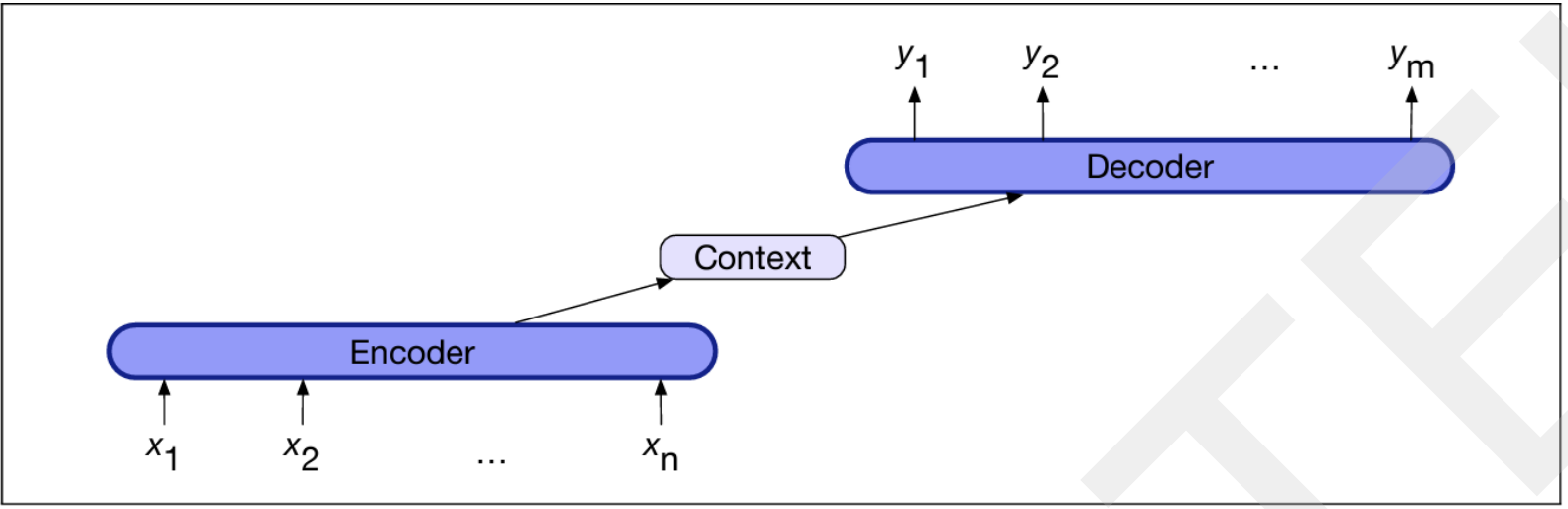


Encoder-decoder networks consist of three components:

1. An **encoder** that accepts an input sequence, x_1^n, and generates a corresponding sequence of contextualized representations, h_1^n. LSTMs, convolutional networks, and Transformers can all be employed as encoders.

Source: https://web.stanford.edu/~jurafsky/slp3.

Encoder-Decoder Architecture



2. A **context vector**, c, which is a function of h_1^n, and conveys the essence of the input to the decoder.
3. A **decoder**, which accepts c as input and generates an arbitrary length sequence of hidden states h_1^m, from which a corresponding sequence of output states y_1^m, can be obtained. Just as with encoders, decoders can be realized by any kind of sequence architecture.

Source: https://web.stanford.edu/~jurafsky/slp3.

Machine Translation Objective

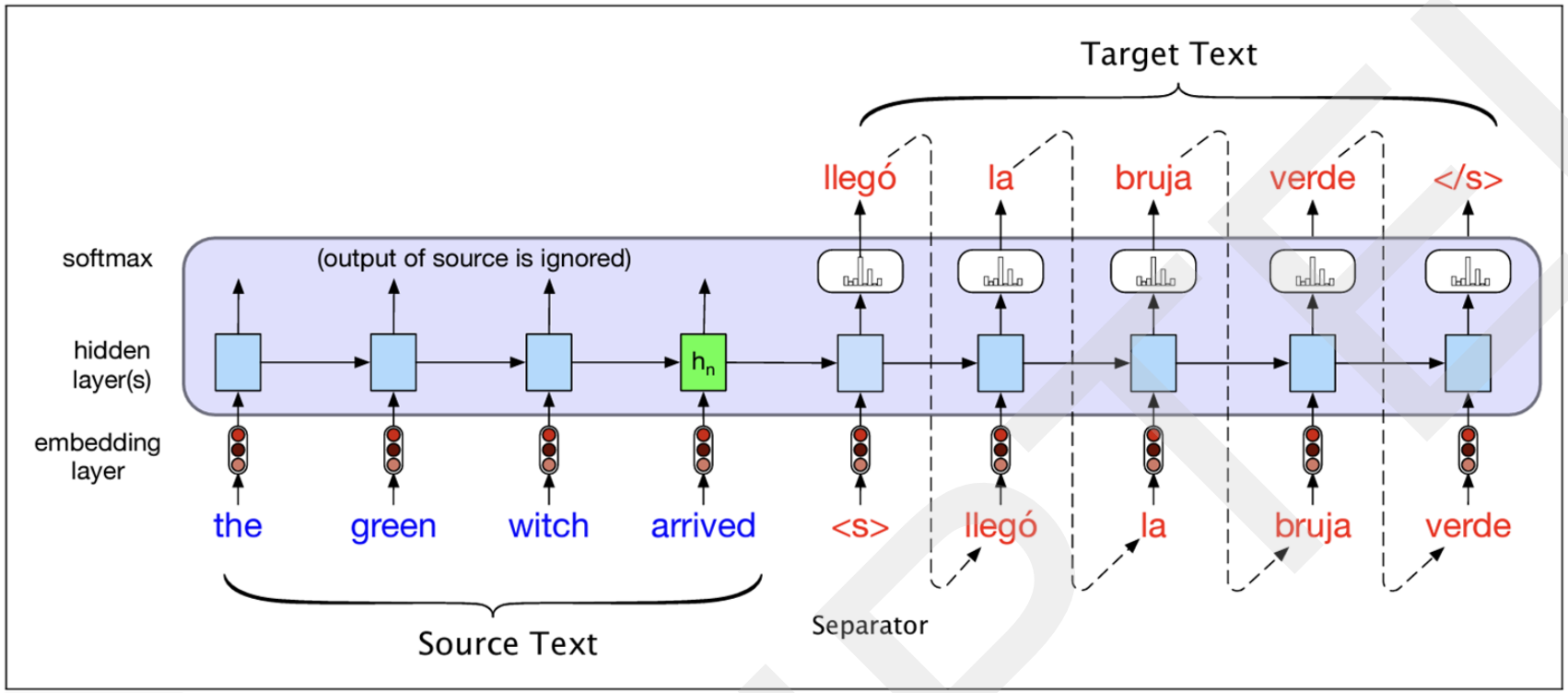
let's think about translating an English source text ("the green witch arrived"), to a Spanish sentence ("llego la bruja verde") (which can be glossed word-by-word as "arrived the witch green").

Let x denote the source text, and y the target text
An encoder-decoder model computes the probability p(y|x) as

$$p(y|x) = p(y_1|x)p(y_2|y_1,x)p(y_3|y_1,y_2,x)...P(y_m|y_1,...,y_{m-1},x)$$

It is also known as *Conditional Language Model*

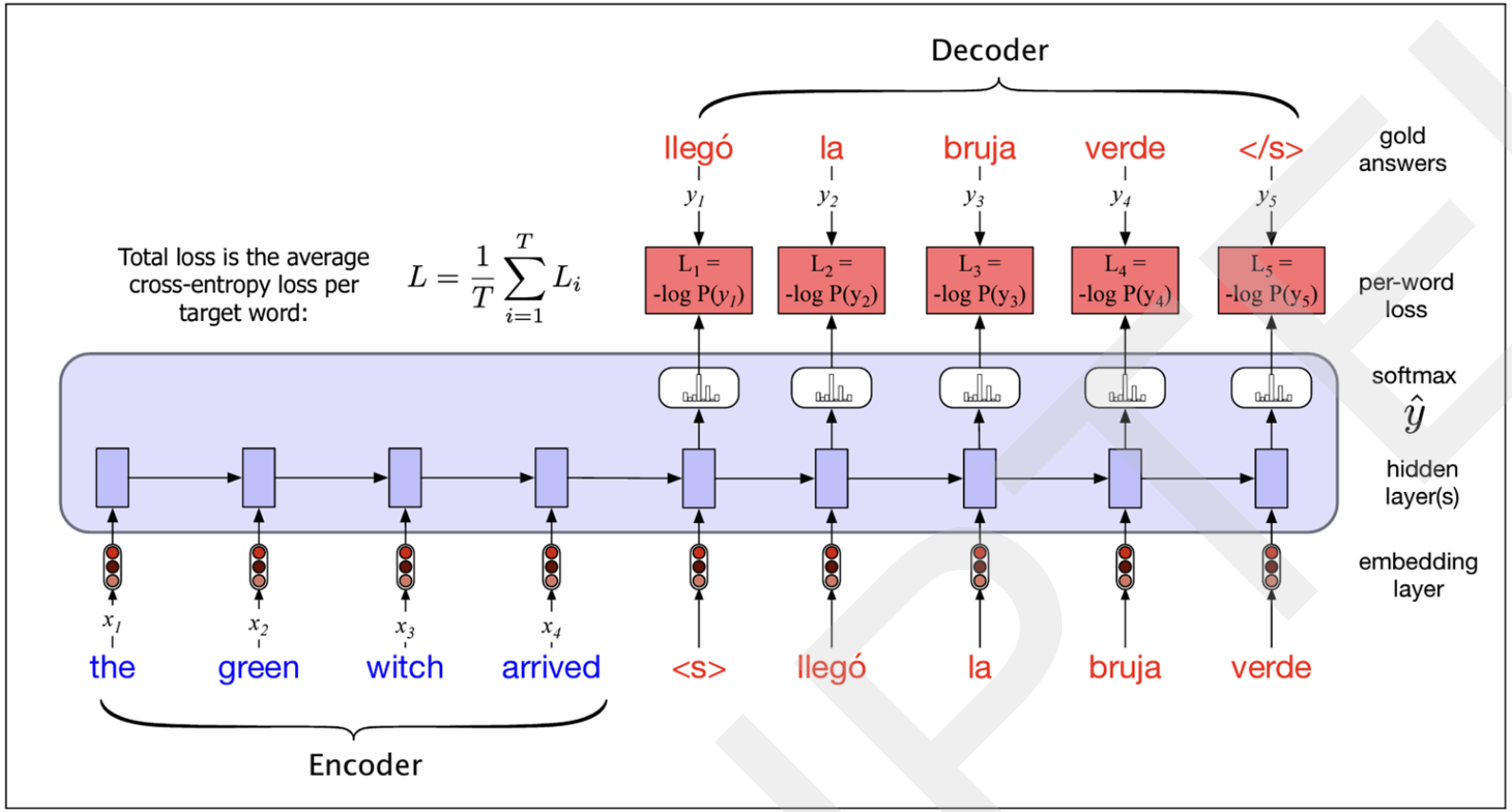
Encoder-decoder model



Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

Source: https://web.stanford.edu/~jurafsky/slp3.

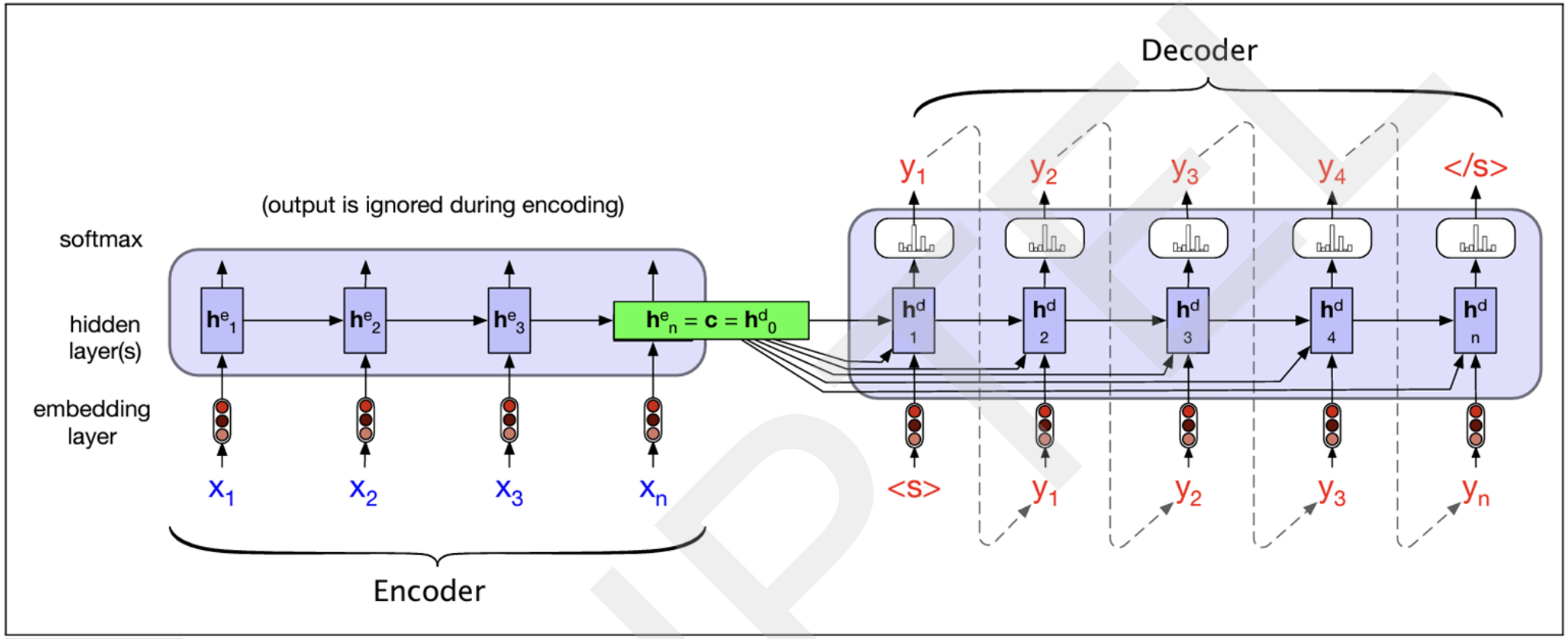
Training the Encoder-decoder model: Teacher forcing



In the decoder, we use **teacher forcing** to force each input to the correct gold value for training. We compute the softmax output distribution in the decoder in order to compute the loss at each token, which can then be averaged to compute a loss for the sentence. This loss is then propagated through the decoder parameters and the encoder parameters.

Source: https://web.stanford.edu/~jurafsky/slp3.

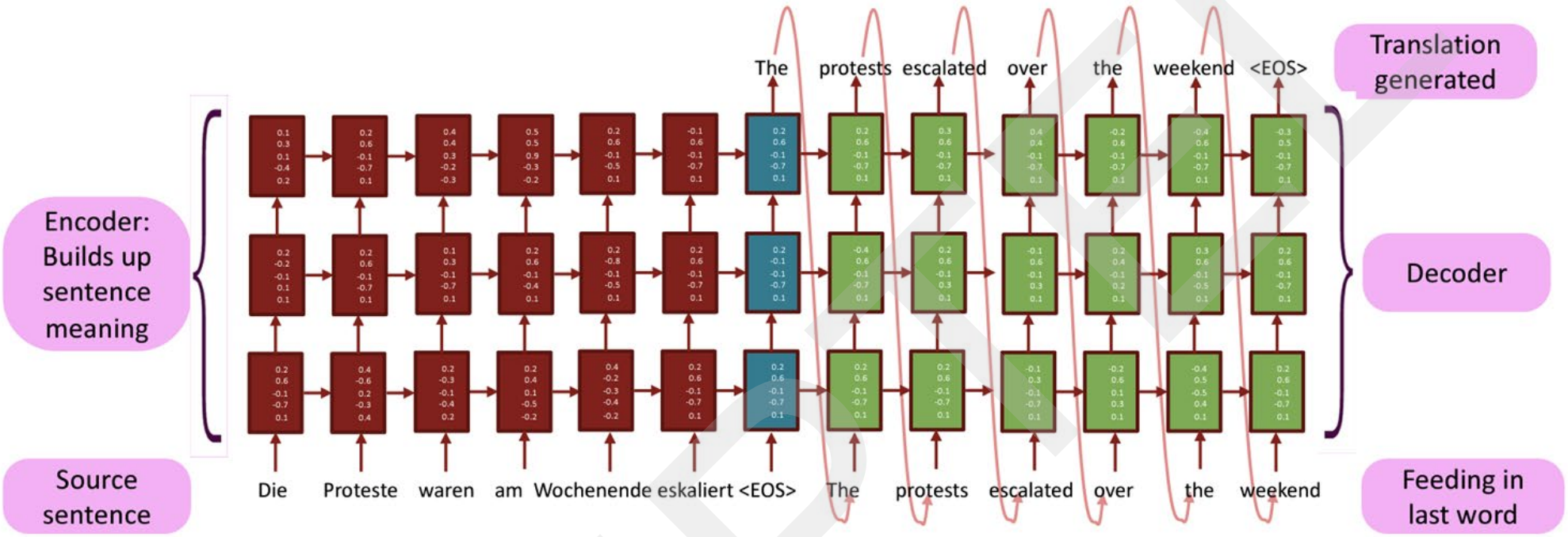
Making the context available in each step



A variation of RNN-based encoder-decoder architecture, where the final hidden state of the encoder RNN is also made available to each decoder hidden state.

Source: https://web.stanford.edu/~jurafsky/slp3.

Multi-layer deep encoder-decoder machine translation net



The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1

Source: https://web.stanford.edu/class/cs224n/

NMT: the first big success story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published [Sutskever et al. 2014]
- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone had
  - https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html

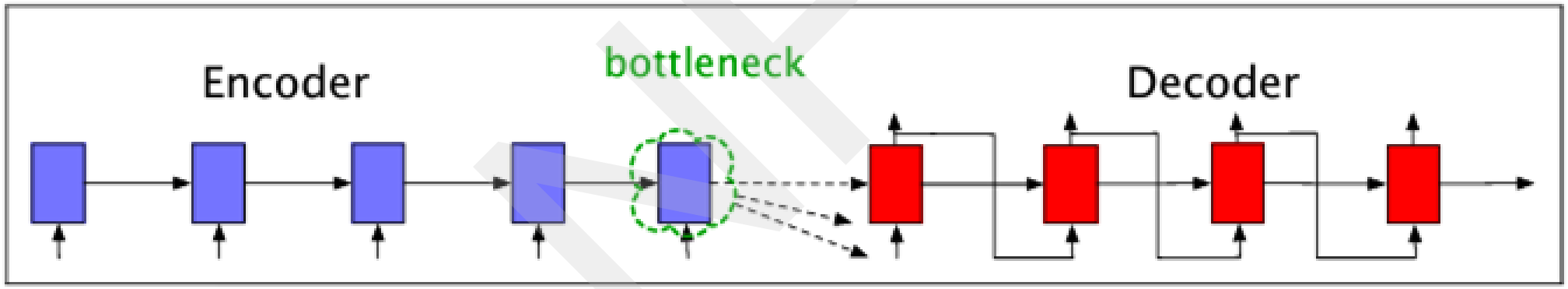
Microsoft SYSTRAN beyond language Google facebook

Baidu 百度 網易 NetEase www.163.com Tencent 腾讯 搜狗搜索

Source: https://web.stanford.edu/class/cs224n/

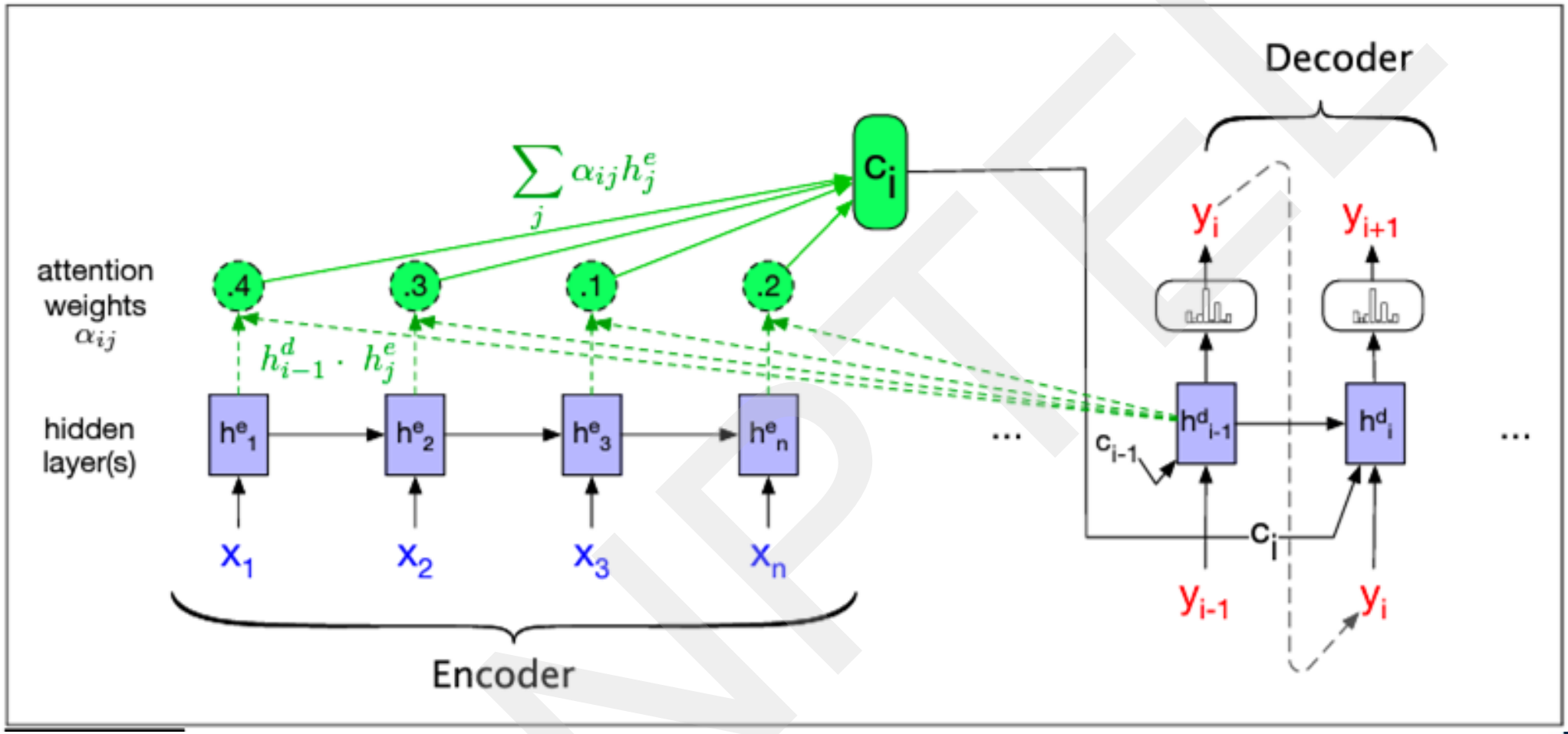
Encoder-decoder: Bottleneck

- The context vector, h_n is the hidden state of the last time step of the source text
- It acts as a bottleneck, as it has to represent absolutely everything about the meaning of the source text, as this is the only thing decoder knows about the source text



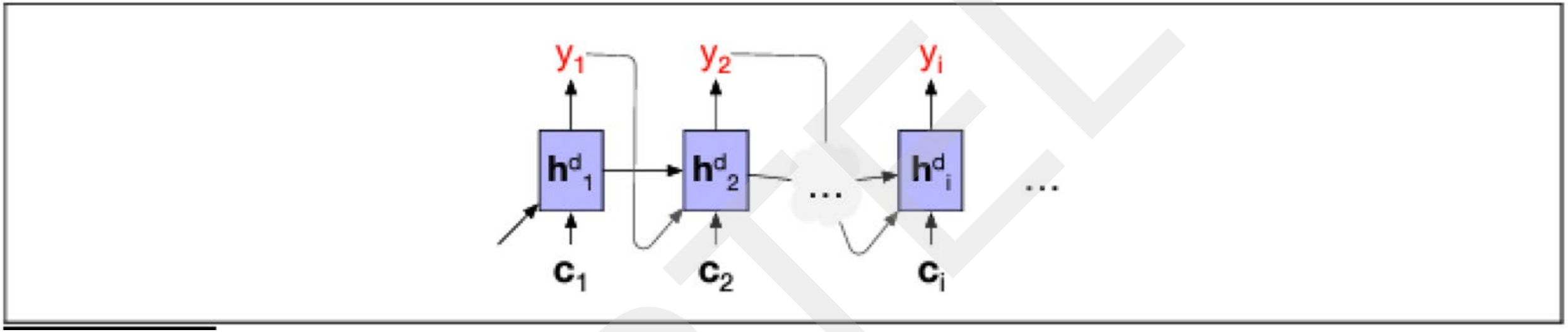
Source: https://web.stanford.edu/~jurafsky/slp3.

Encoder-decoder with attention



Source: https://web.stanford.edu/~jurafsky/slp3.

Encoder-decoder with attention



The attention mechanism allows each hidden state of the decoder to see a different, dynamic context, which is a function of all the encoder states

The context vector c_i is generated anew with each decoding step i

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

Source: https://web.stanford.edu/~jurafsky/slp3.

Attention: In Equations

Computing c_i

- Compute how much to focus on each encoder state, by seeing how relevant it is to the decoder state captured in h^d_{i-1} – give it a score
- Simplest scoring mechanism is dot-product attention

$$score(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$

- Normalize these scores using softmax to create a vector of weights $\alpha_{ij} = softmax(score(h^d_{i-1}, h^e_j))$
- A fixed-length context vector is created for the current decoder state

$$c_i = \sum_j \alpha_{ij} h^e_j$$

Try this problem

Suppose you are using sequence to sequence RNN model with attention. Suppose the encoder has 4 hidden states (2-dimensional) [1,-1], [-1,1], [0,-1], [-1,0] and the decoder hidden state at a given time point is [-1,-1]. Using attention (dot product) over the encoder states, what will be the context vector to the next hidden state of the decoder?

Attention is quite helpful

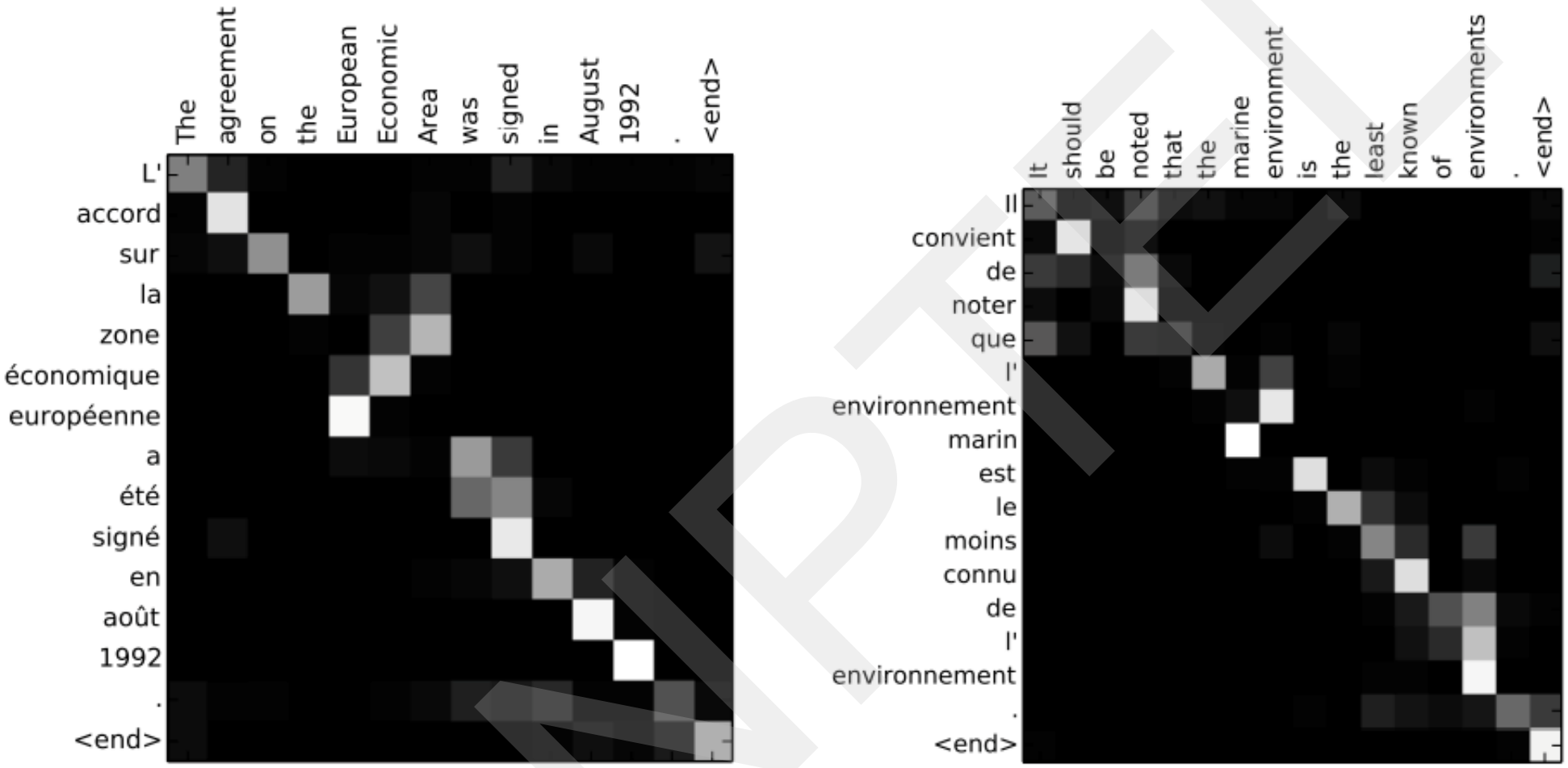
Attention improves NMT performance

It is useful to allow decoder to focus on certain parts of the source

Attention provides some interpretability

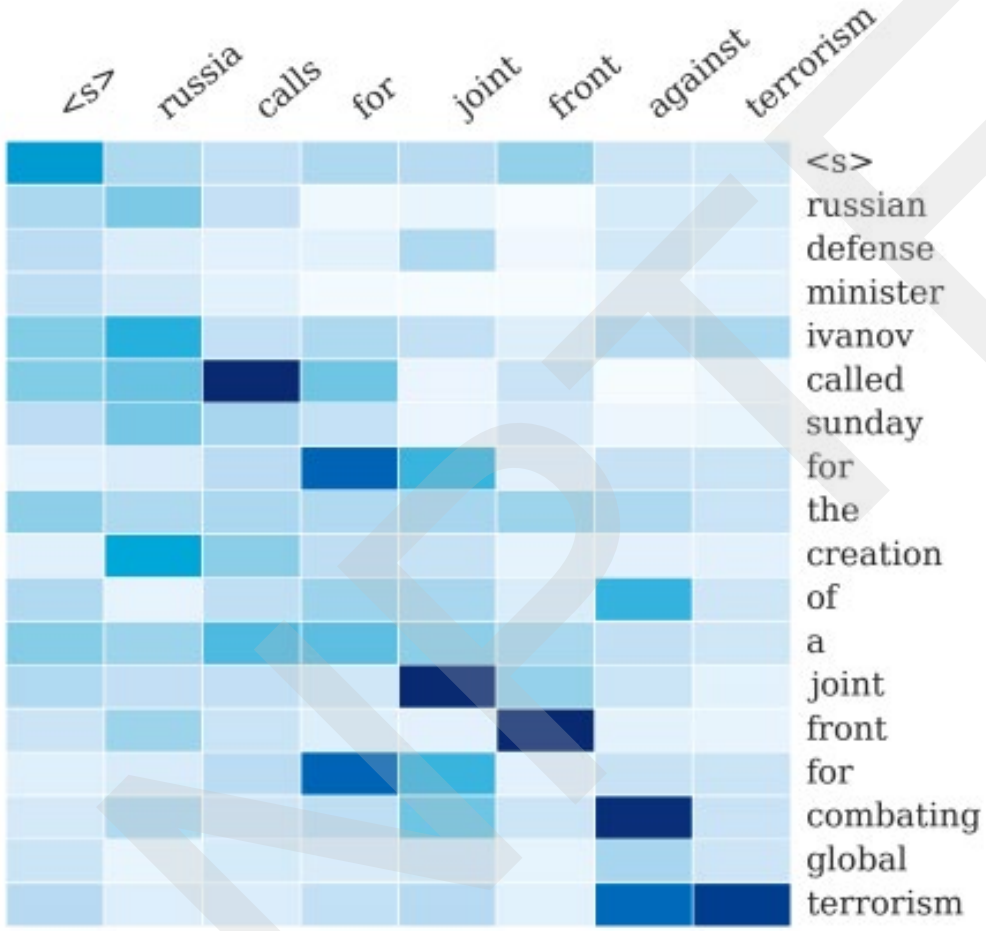
- By inspecting attention distribution, we can see what the decoder was focusing on
- We get alignment for free even if we never explicitly trained an alignment system

Example: Machine Translation



Neural Machine Translation by jointly learning to align and Translate, ICLR 2015

Example: Machine Translation



A Neural Attention Model for Sentence Summarization, EMNLP 2015

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 8]


THANK YOU
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

Lecture 19 : Decoding Strategies

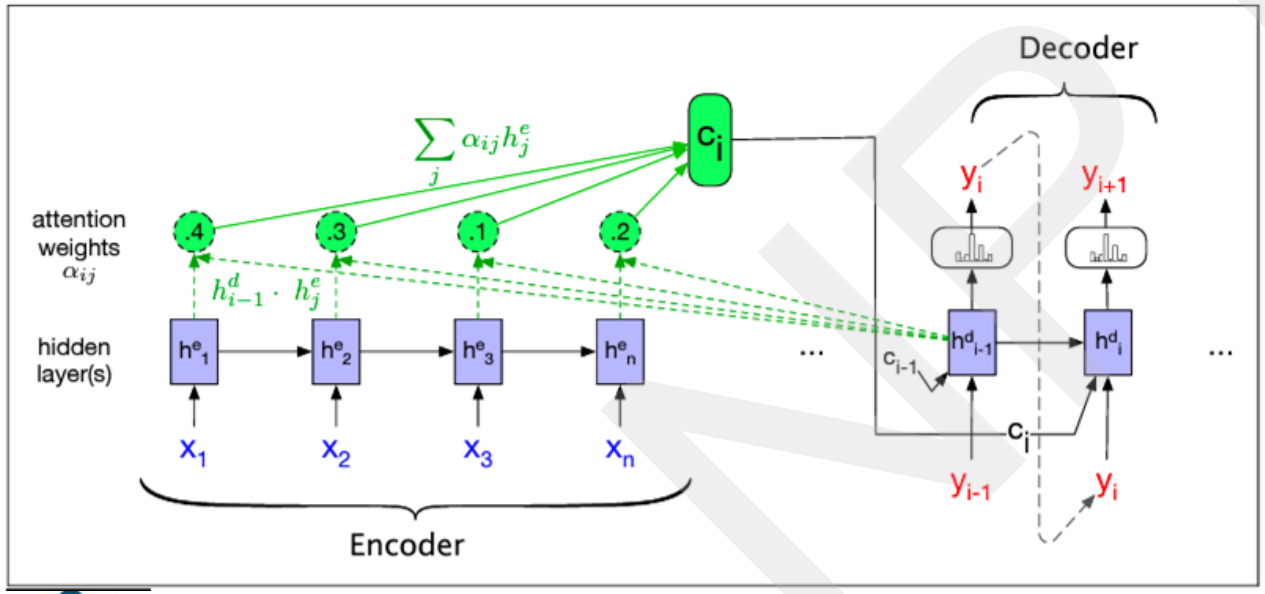
PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

- Greedy Decoding, Beam Search Decoding
- Random Sampling with temperature
- Top-k Sampling, Top-p Sampling

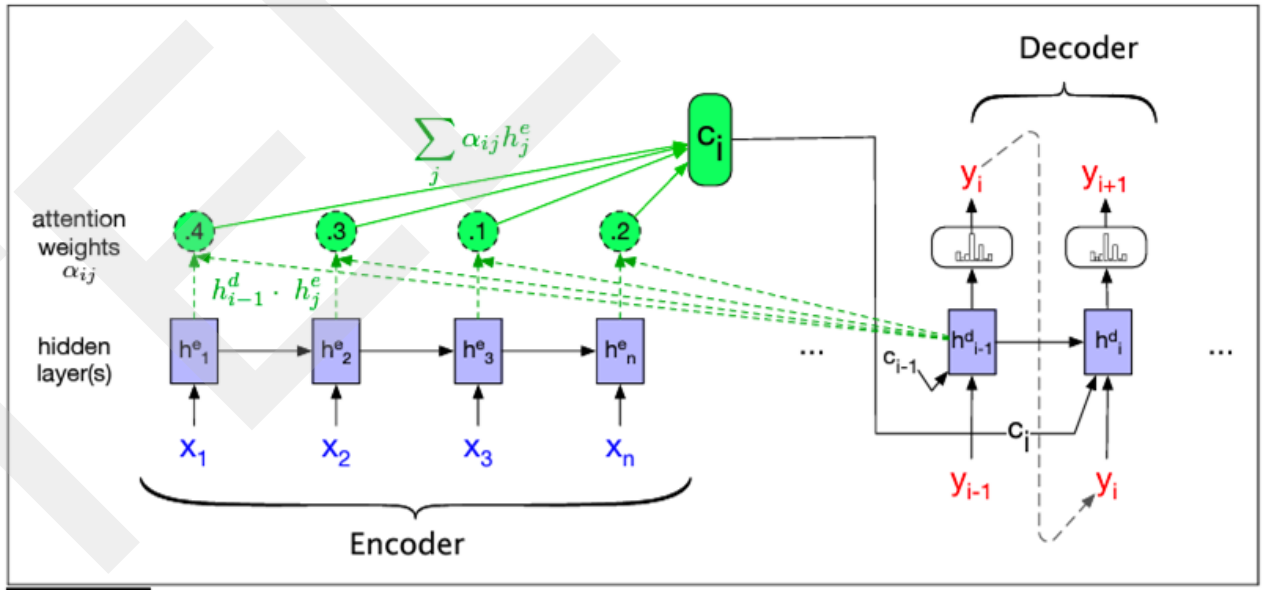
Try this Problem

Suppose you are using sequence to sequence RNN model with attention for Machine Translation from English to Hindi. Assume that the vocabulary for English is 40k and that for Hindi is 50k, and both use 300-dimensional embeddings. Also, assume that the encoder hidden state is 100-dimensional, while decoder hidden state is 200-dimensional. What will be the number of parameters to be trained? You may ignore the bias terms.

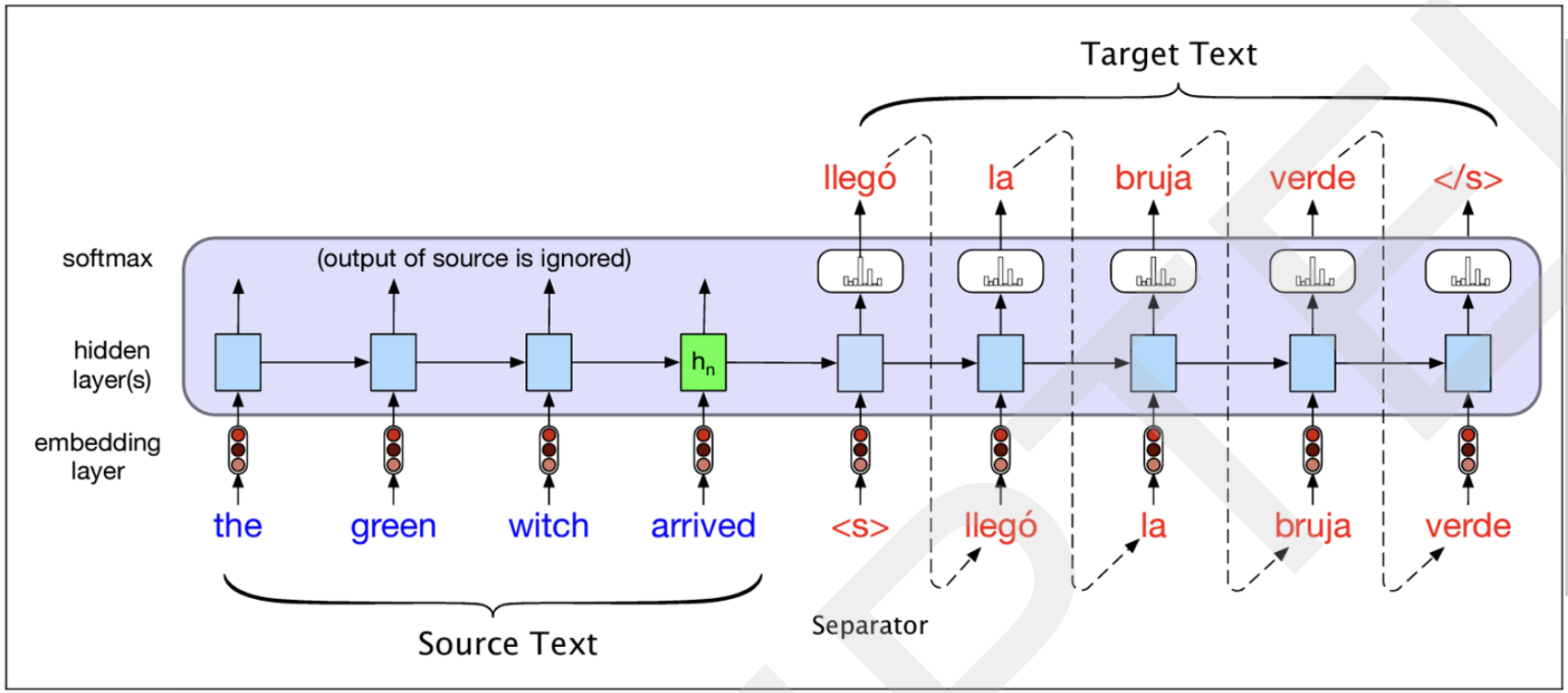


Try this Problem

Vocabulary for English is 40k and that for Hindi is 50k, 300-d embeddings. Encoder hidden:100, decoder hidden state: 200.



Recap: Encoder-decoder model at inference



Translating a single sentence (inference time) in the basic RNN version of encoder-decoder approach to machine translation. Source and target sentences are concatenated with a separator token in between, and the decoder uses context information from the encoder's last hidden state.

How do we generate the tokens in an autoregressive manner?

Source: https://web.stanford.edu/~jurafsky/slp3 .

Which words do we generate at each step?

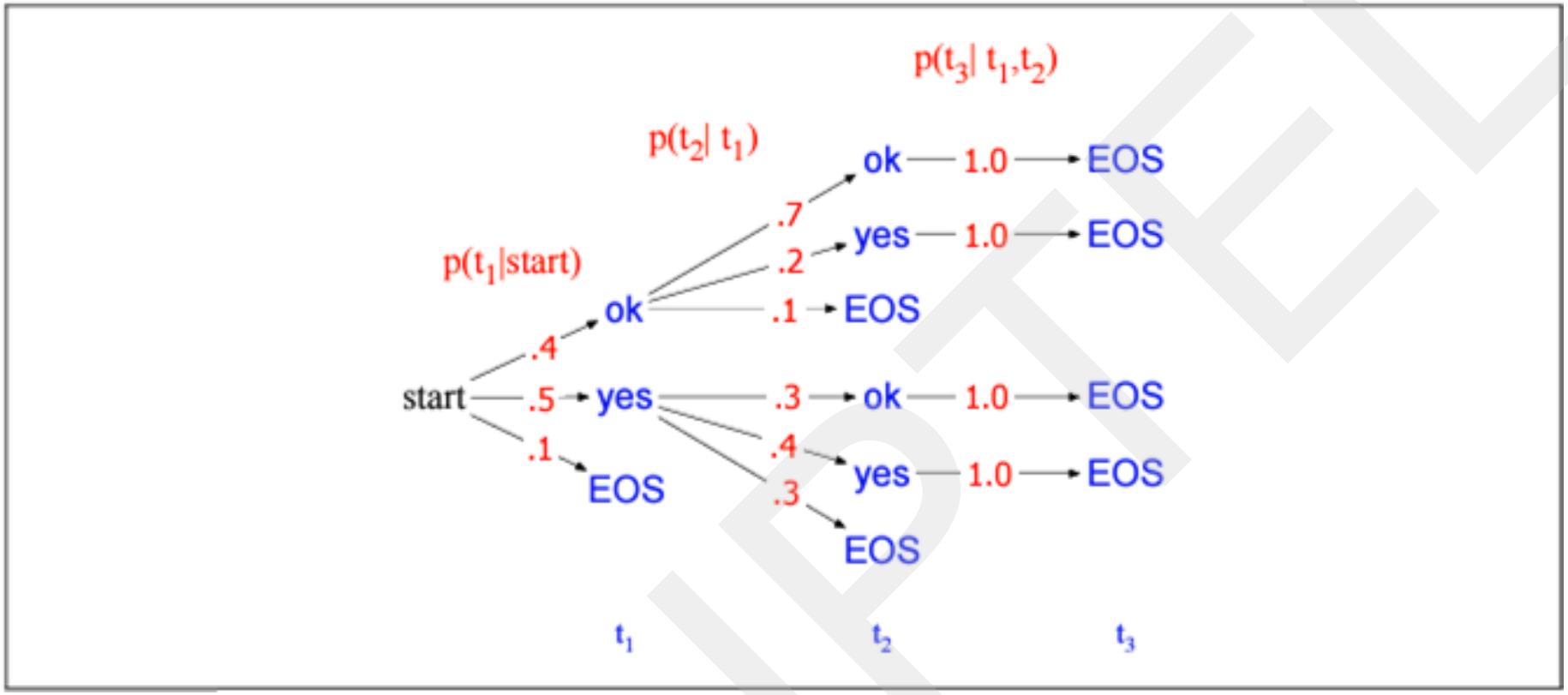
Greedy Decoding

One simple way is to always generate the most likely word given the context

Generating the most likely word given the context is called *greedy decoding*.

$$\hat{w}_t = \arg\max_{w \in V} P(w|w_{<t})$$

What is the issue with greedy decoding?



- *What will greedy decoding choose?*
- *What is globally optimal?*

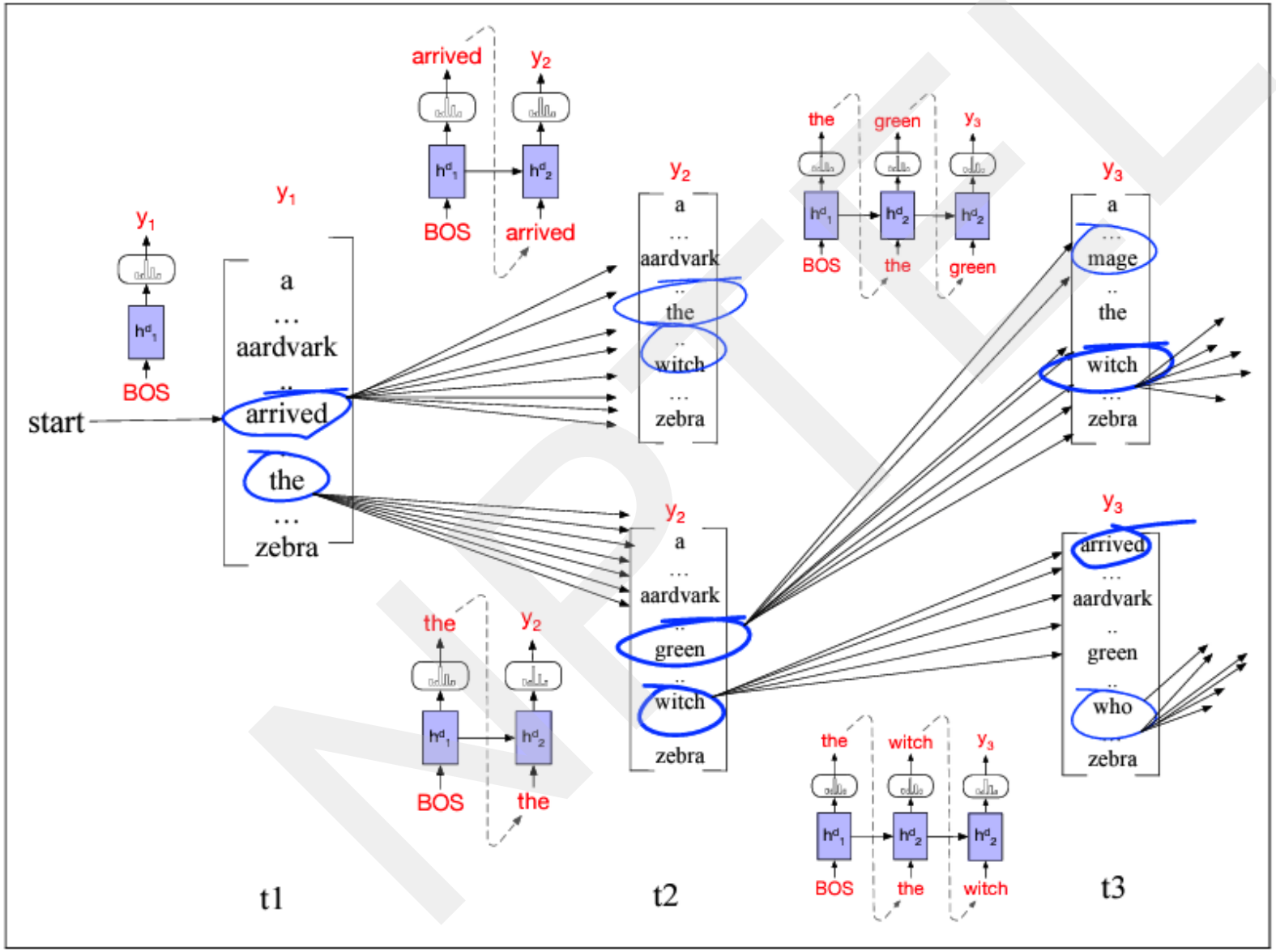
Source: https://web.stanford.edu/~jurafsky/slp3 .

Beam Search

Core Idea

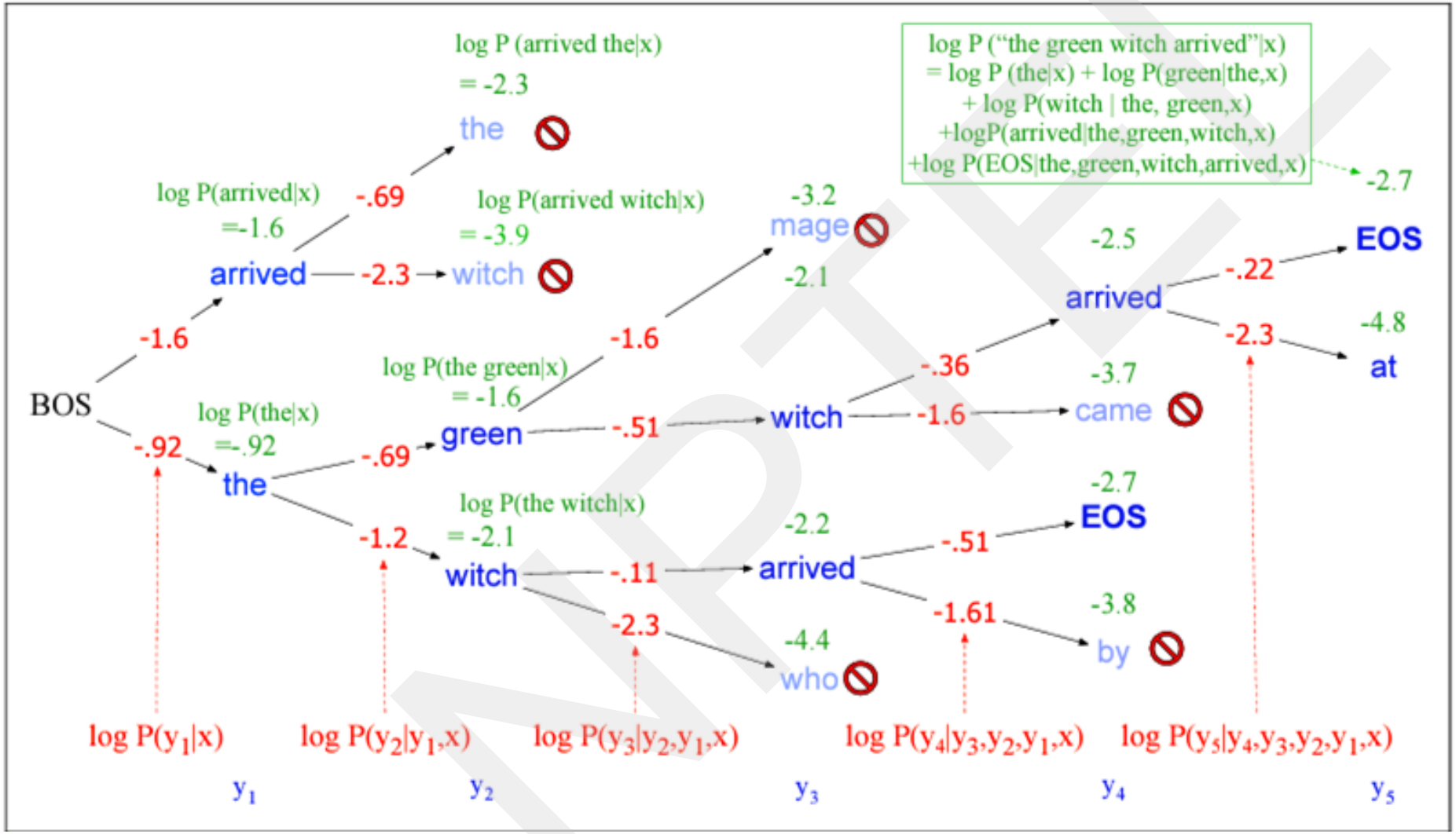
- On each step of the decoder, keep track of the k most probable partial translations (hypotheses)
- k is the beam size / beam width (in practice around 5 to 10)

Beam Search: How does it work?



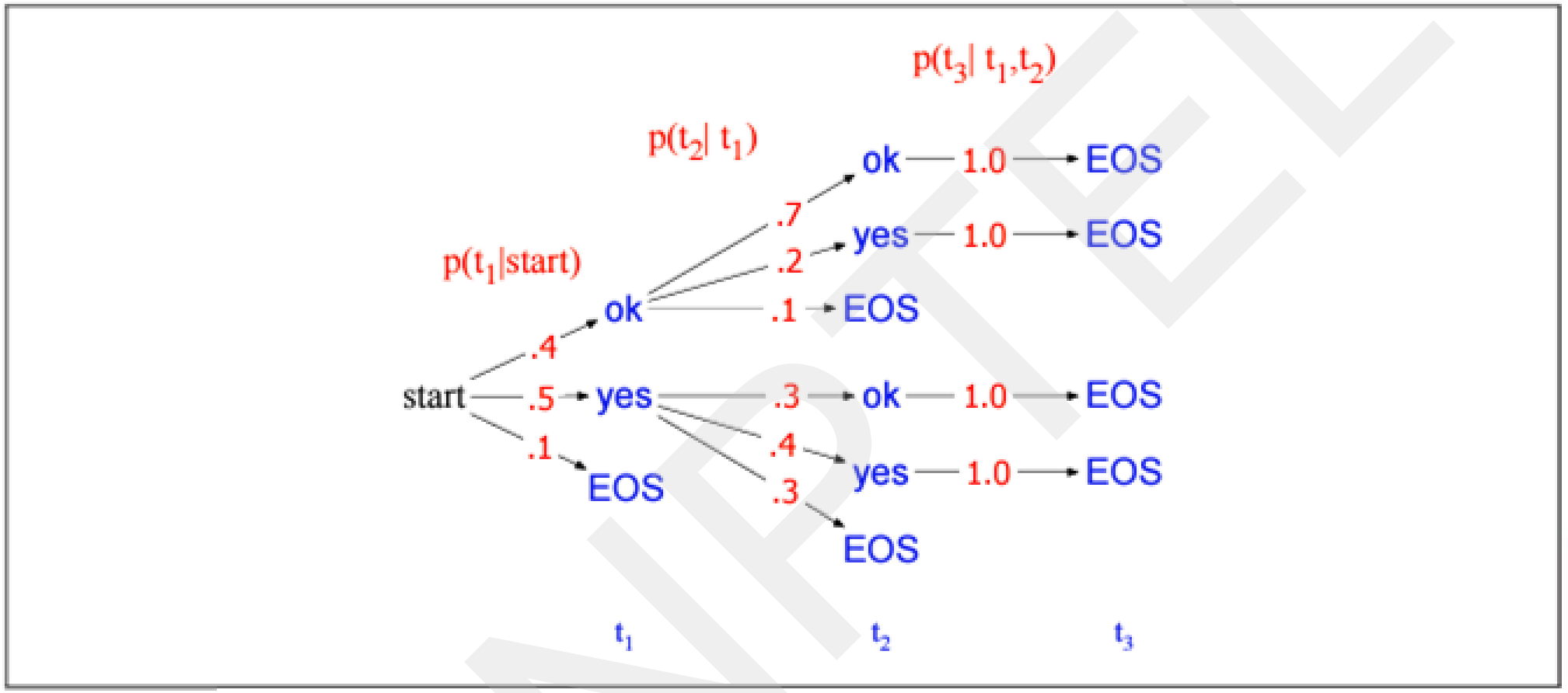
https://web.stanford.edu/~jurafsky/slp3

Beam Search: How does it work?



https://web.stanford.edu/~jurafsky/slp3

Beam Search on the prev. example



Take Beam Size = 2. Is the Beam search solution same as the optimal solution?

https://web.stanford.edu/~jurafsky/slp3

Beam Search: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding, different hypotheses may produce $\langle END \rangle$ tokens on different timesteps
  - When a hypothesis produces $\langle END \rangle$, that hypothesis is complete
  - Place it aside and continue exploring other hypotheses via beam search
- Usually we continue beam search until:
  - We reach timestep T (some pre-defined cutoff), or
  - We have at least n completed hypotheses (some pre-defined cutoff)

Beam Search: Finishing up

- We have our list of completed hypotheses
- How to select the top one with the highest score?

Each hypothesis $y_1, \ldots, y_t$ has a score

$$score(y_1, \ldots, y_t) = logP_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

Problem with this

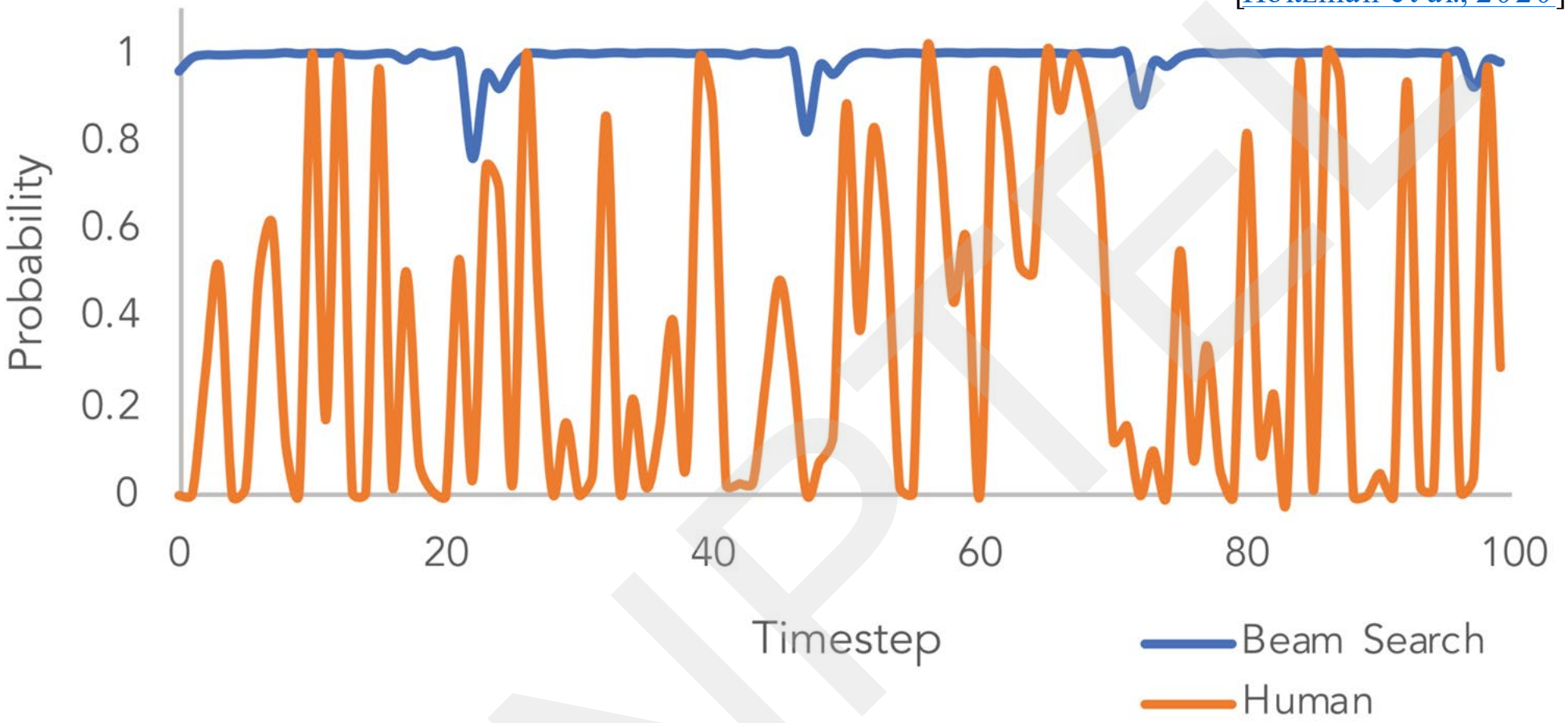
Longer hypotheses have lower scores.

Fix: Normalize the score by length, and use the normalized score to select the top one instead.

$$\frac{1}{t}\sum_{i=1}^{t} logP_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$$

Beam Search Text is Less Surprising

[Holtzman et al., 2020]



https://utah-cs6340-nlp.notion.site/Natural-Language-Processing-bd1a2ca290fc44f69556908ad8d25c70

Greedy and Beam Search are deterministic!

- Greedy decoding as well as Beam Search decoding will give a “deterministic” output
- Other common decoding algorithms involve “sampling”, and bring in some degree of “randomness”

$x \sim p(x) \rightarrow$ choose x by sampling from the distribution $p(x)$

Random Sampling

i $\leftarrow$ 1
$w_i \sim$ p(w)
while w_i != EOS
  i $\leftarrow$ i + 1
  $w_i \sim$ p($w_i \mid w_{<i}$)

Quality vs Diversity trade-off

Various sampling methods enable trading off two important factors in generation: *quality* and *diversity*.

Quality vs Diversity trade-off

- Methods that emphasize the most probable words tend to produce more coherent and accurate generations but also tend to be repetitive and boring
- Methods that give bit more weight to the middle probability words tend to be more creative and diverse, but likely to be incoherent and less factual

Random Sampling with Temperature

Intuition from thermodynamics

A system at a *high temperature* is flexible and can explore various states, while a system at a *low temperature* is likely to explore a subset of lower energy (better) states

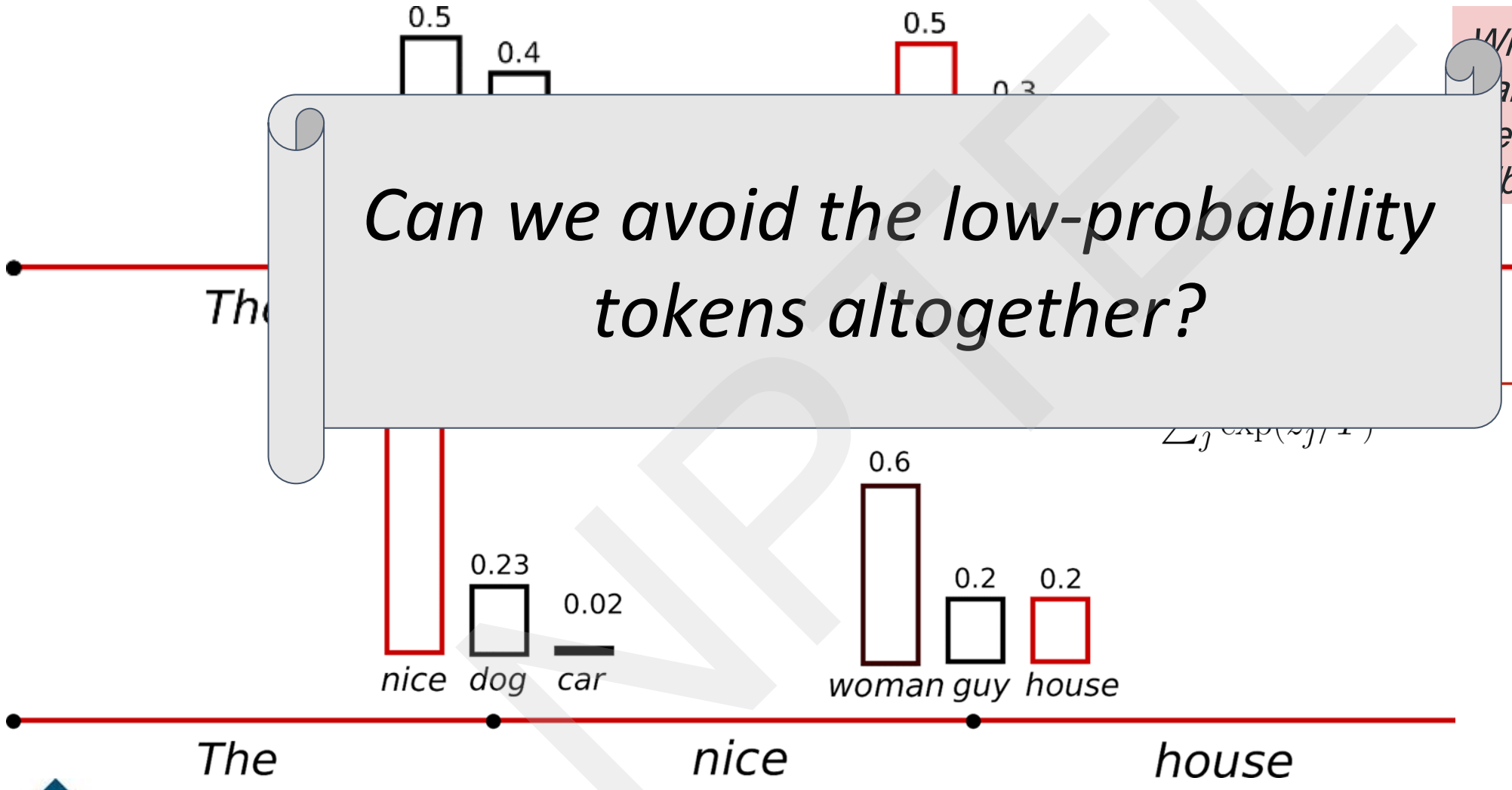
How is this implemented

Divide the logits by a temperature parameter $\tau \in (0, 1]$ before passing it through softmax

Random sampling: $y = softmax(u)$

Random sampling with temperature: $y = softmax(u/\tau)$

Why does this work?



Can we avoid the low-probability tokens altogether?

Without scaling, random sampling often generates incoherent gibberish

Higher T: softens probabilities.
Lower T: sharpens probabilities.

https://utah-cs6340-nlp.notion.site/Natural-Language-Processing-bd1a2ca290fc44f69556908ad8d25c70

Top-k Sampling

In Top-k Sampling, only Top k tokens (as per prob.) are considered for generation, so the less probable words would not have any chance

1. Choose in advance a number of words k
2. For each word in the vocabulary V, use the language model to compute the likelihood of this word given the context $p(w_t|\mathbf{w}_{<t})$
3. Sort the words by their likelihood, and throw away any word that is not one of the top k most probable words.
4. Renormalize the scores of the k words to be a legitimate probability distribution.
5. Randomly sample a word from within these remaining k most-probable words according to its probability.

Nucleus Sampling or Top-p sampling

Issues with Top-k Sampling

Shape of the probability distribution differs in different contexts. Top-k may include most of the probability mass in some cases, and very small mass in other cases.

Nucleus Sampling or top-p sampling

Keep not the top k words but top p percent of the probability mass

Given a distribution $P(w_t|w_{<t})$, top-p vocabulary $V^{(p)}$ is the smallest set of words such that

$$\sum_{w \in V^{(p)}} P(w|w_{<t}) \geq p$$

Try this problem

Suppose you have a vocabulary of size 5 and during decoding, the output vector is [3, -1, 2, 1, -2]. Write down the effective probability distribution when you use the following sampling strategies.

- Random sampling with temperature 0.5
- Top-2 sampling
- Nucleus sampling with p = 0.5

Try this problem

Suppose you have a vocabulary of size 5 and during decoding, the output vector is [3, -1, 2, 1, -2]. Write down the effective probability distribution when you use the following sampling strategies.

- Random sampling with temperature 0.5
- Top-2 sampling
- Nucleus sampling with p = 0.5

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 10]


THANK YOU
NPTEL

NPTEL ONLINE CERTIFICATION COURSES

DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING

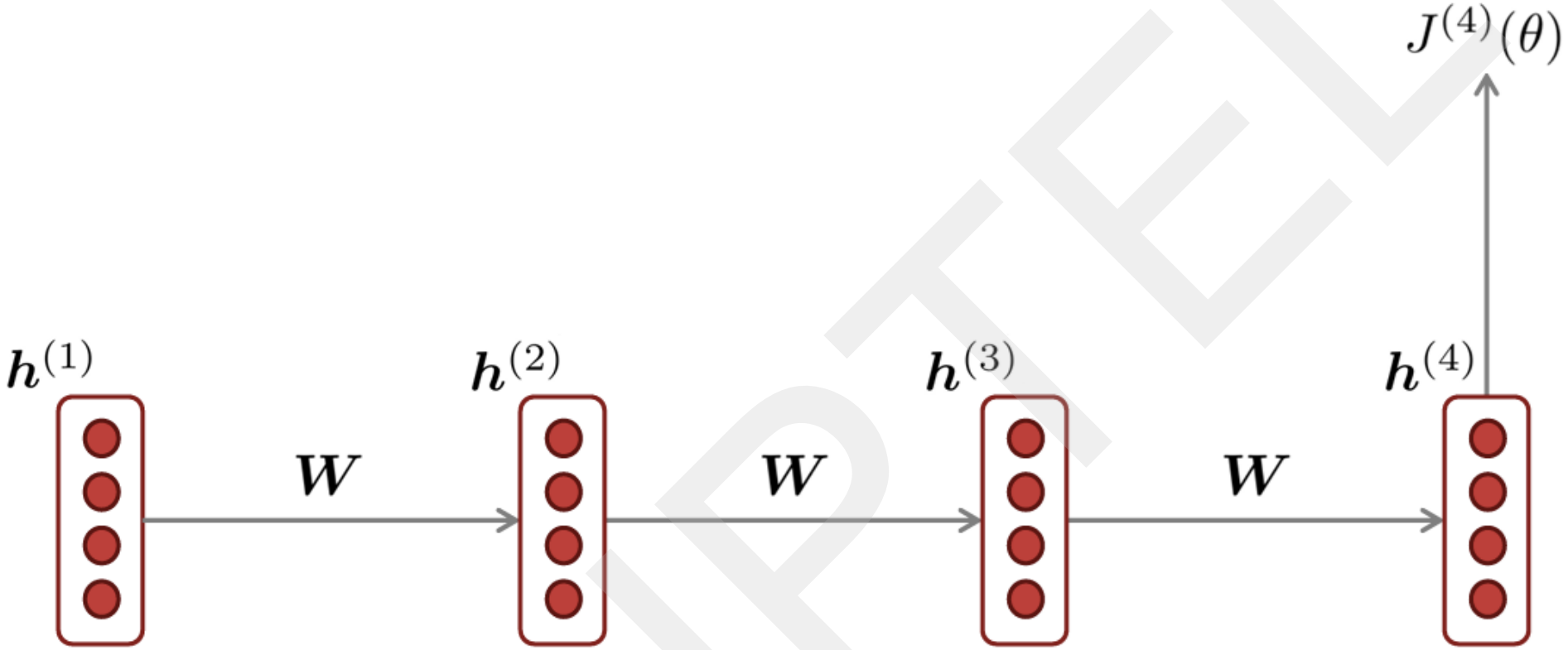
Lecture 20 : Better RNN Units: GRU, LSTM

PROF. PAWAN GOYAL
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CONCEPTS COVERED

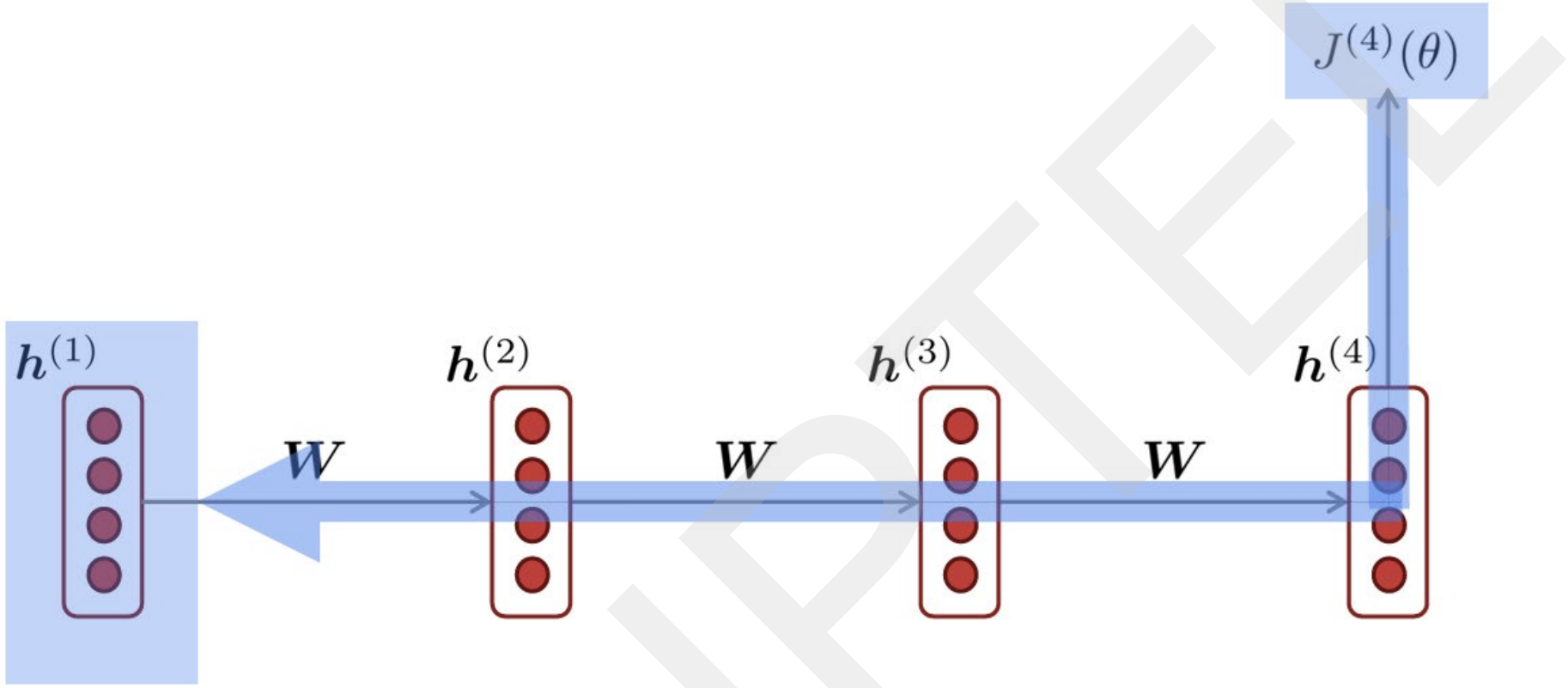
- Issues with Vanilla RNN: Vanishing Gradient
- Gates for better Units: LSTMs

Problem with RNN: Vanishing Gradient



Source: https://web.stanford.edu/class/cs224n/

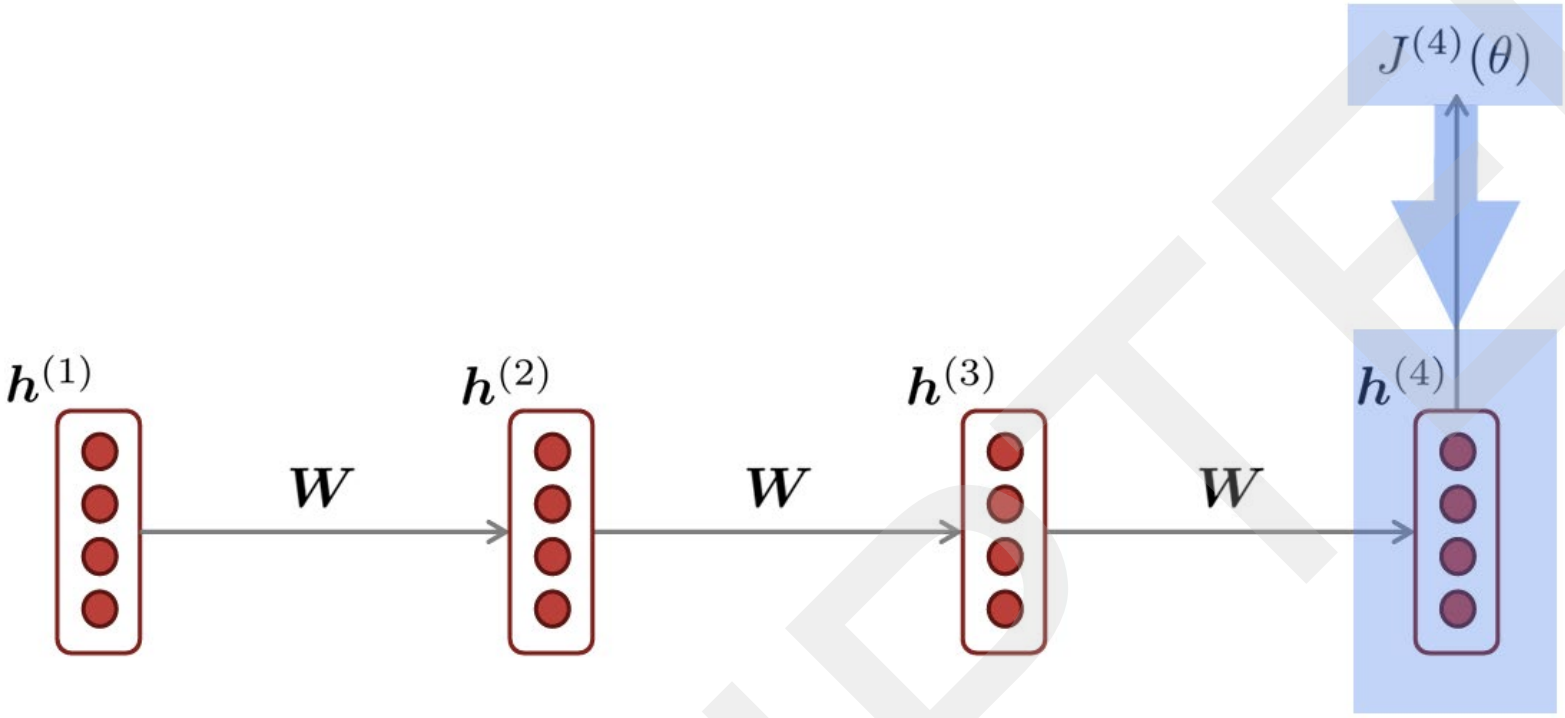
Vanishing Gradient: Intuition



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?$$

Source: https://web.stanford.edu/class/cs224n/

Vanishing Gradient: Intuition

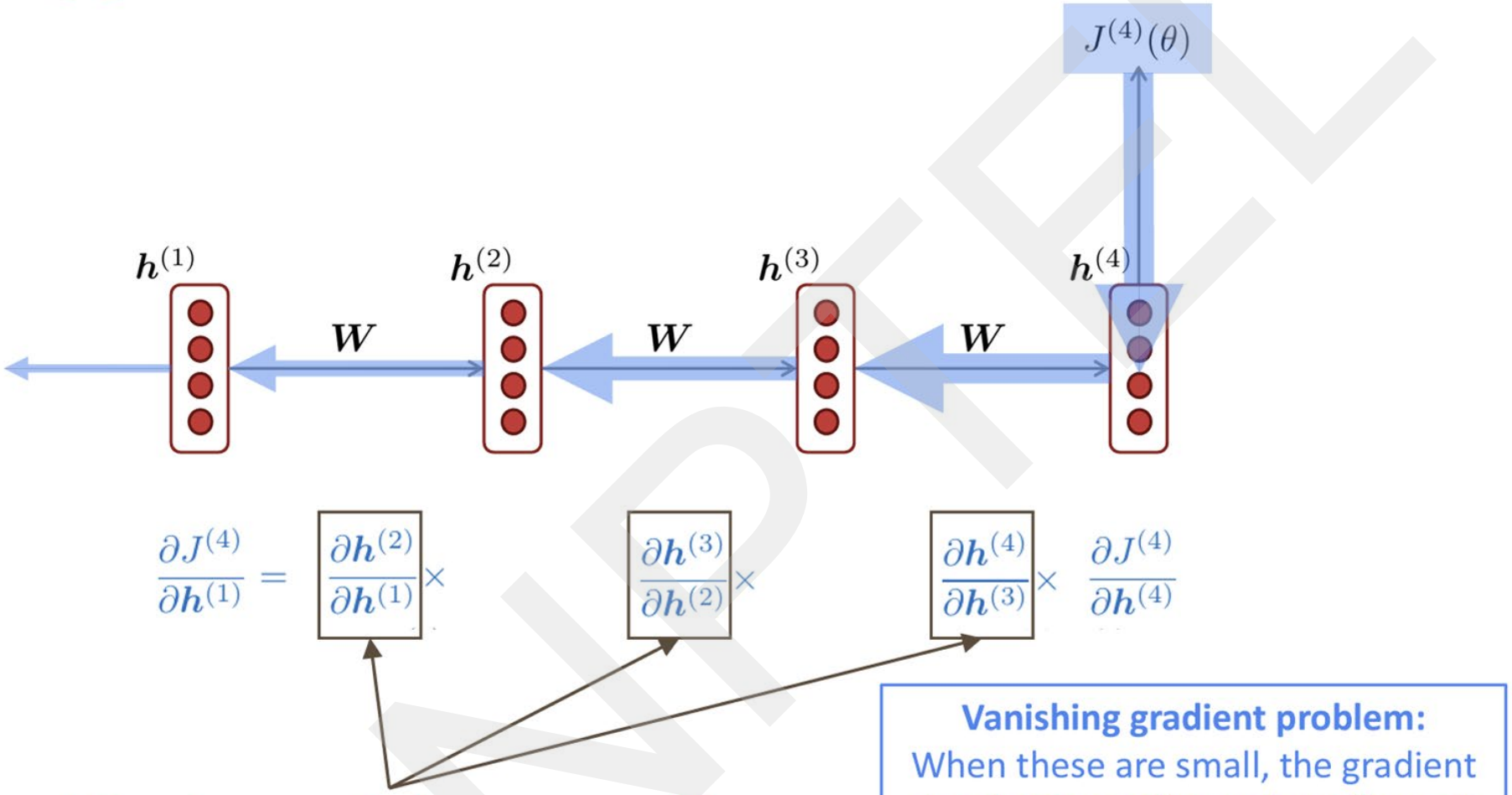


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

chain rule!

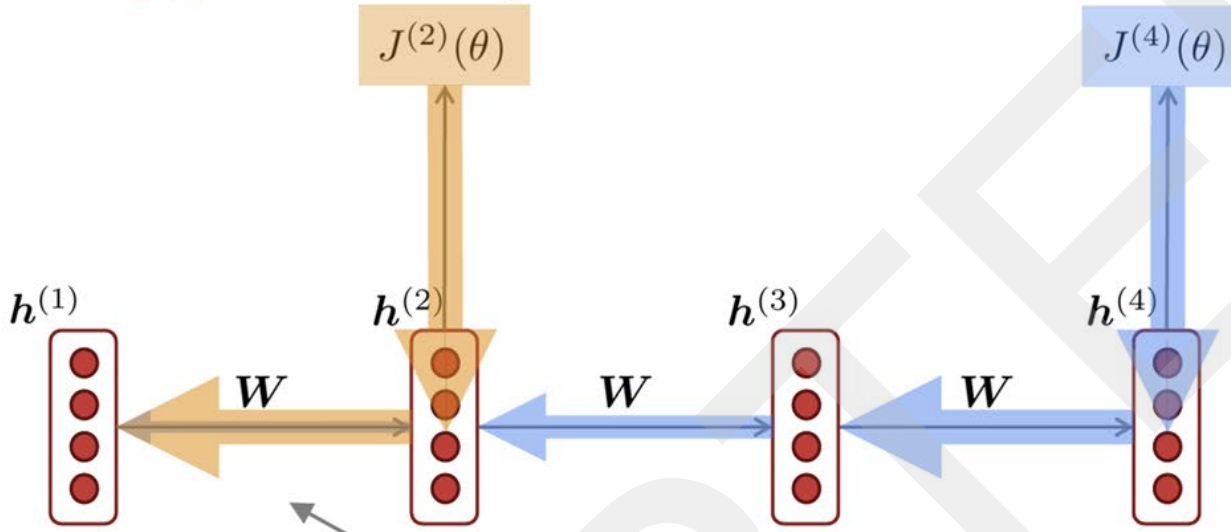
Source: https://web.stanford.edu/class/cs224n/

Vanishing Gradient: Intuition



Source: https://web.stanford.edu/class/cs224n/

Why is this a problem?



Gradient signal from far away is lost because it's much smaller than gradient signal from close-by.

So, model weights are basically updated only with respect to near effects, not long-term effects.

Source: https://web.stanford.edu/class/cs224n/

Effect of vanishing gradient on RNN-LM

- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if the gradient is small, the model can't learn this dependency
  - So, the model is unable to predict similar long-distance dependencies at test time

- **Syntactic recency:** *The writer of the books is* (correct)

- **Sequential recency:** *The writer of the books are* (incorrect)

Source: https://web.stanford.edu/class/cs224n/

How to fix vanishing gradient problem?

- The main problem is that it is too difficult for the RNN to learn to preserve information over many timesteps.
- In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = tanh(Uh^{(t-1)} + Wx^{(t)})$$

- How about better RNN units?

Using Gates for better RNN Units

- The gates are also vectors
- On each timestep, each element of the gates can be open (1), close (0) or somewhere in-between.
- The gates are dynamic: their value is computed based on the current context.

Two famous architectures

GRUs, LSTMs

Long Short Term Memory (LSTM)

On step t, there is a hidden state $\boldsymbol{h}^{(t)}$ and a cell state $\boldsymbol{c}^{(t)}$

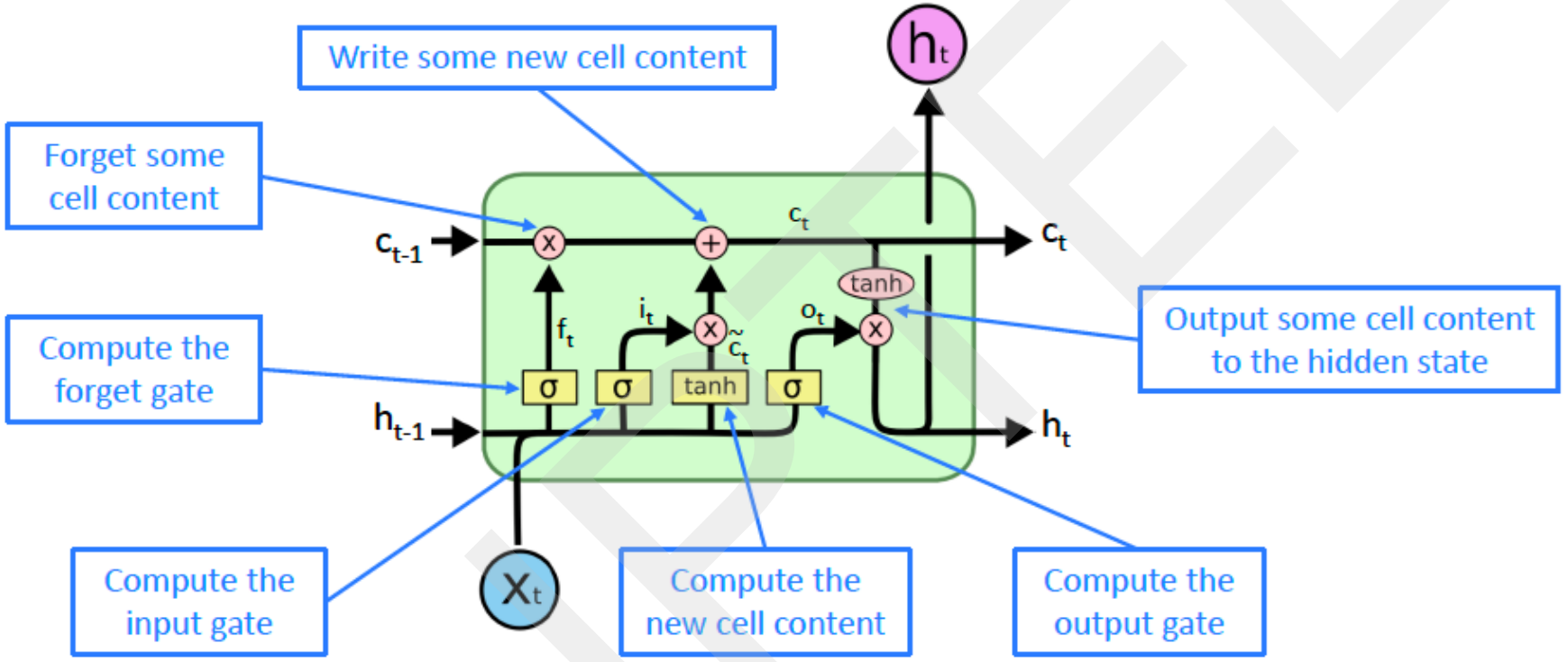
- Both are vectors length n
- The cell stores long-term information
- The LSTM can read, erase, and write information from the cell

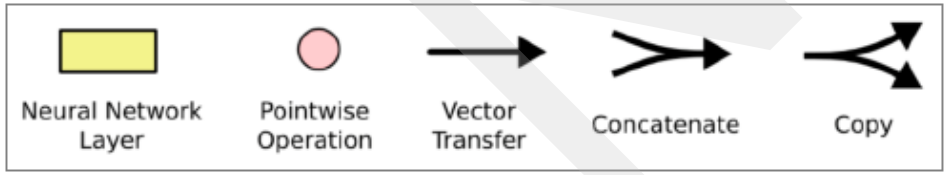
The selection of which information is erased/written/read is controlled by three corresponding gates (gates are calculated things whose values are probabilities)

Source: https://web.stanford.edu/class/cs224n/

Long Short Term Memory (LSTM)

You can think of the LSTM equations visually like this:





Source: https://web.stanford.edu/class/cs224n/

Long Short Term Memory (LSTM): In Equations

Forget Gate

Controls what is kept vs forgotten from the context

$$f_t = \sigma(U_f h_{t-1} + W_f x_t)$$

Input Gate

Controls what parts of new cell content are written to the context

$$i_t = \sigma(U_i h_{t-1} + W_i x_t)$$

New Cell content: $g_t = tanh(U_g h_{t-1} + W_g x_t)$

New Context Vector: $c_t = i_t \odot g_t + f_t \odot c_{t-1}$

Long Short Term Memory (LSTM): In Equations

Output Gate

Controls what part of context are output to hidden state

$$o_t = \sigma(U_o h_{t-1} + W_o x_t)$$

New Hidden State: $h_t = o_t \odot tanh(c_t)$

How does LSTM solve Vanishing Gradient?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g., if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
- By contrast, it is harder for vanilla RNN to learn a recurrent weight matrix U that preserves info in hidden state

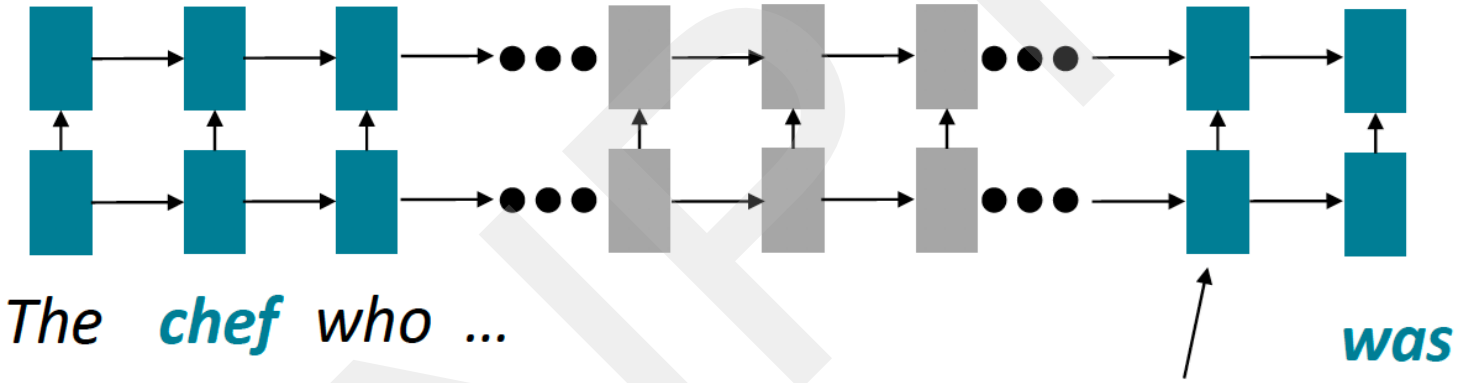
Try this problem

Suppose you are using Bi-LSTM for language labeling in code-switched data, and there are 5 possible language tags per word. Assume that each word has a 50-dimension embedding, and the hidden state for both the forward and backward LSTMs are 40-dimension each. How many total parameters will need to be trained? Ignore the bias terms.

Issues with recurrent models

O(sequence length) steps for distant word pairs

- Hard to learn long-distance dependencies (gradient problems!)
- Linear order of words is "baked in"; not the right way to think about sentences



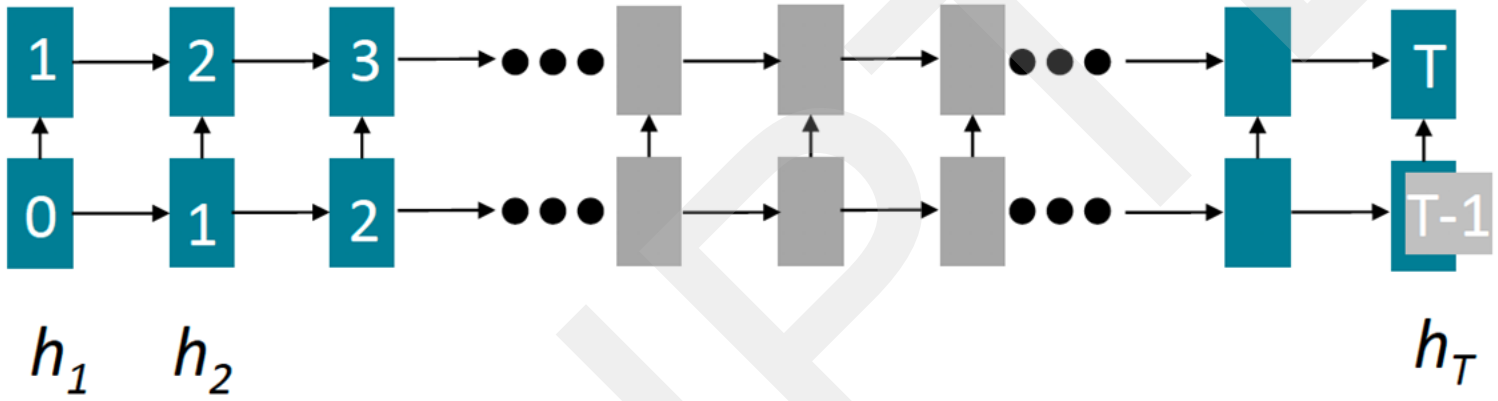
Info of ***chef*** has gone through O(sequence length) many layers!

https://web.stanford.edu/class/cs224n/

Issues with recurrent models

Lack of parellilizability

- Future RNN hidden states can't be computed in full before past RNN hidden states have been computed



Inhibits training on very large datasets!

Numbers indicate min # of steps before a state can be computed

https://web.stanford.edu/class/cs224n/

LSTMs Revamped: xLSTMs

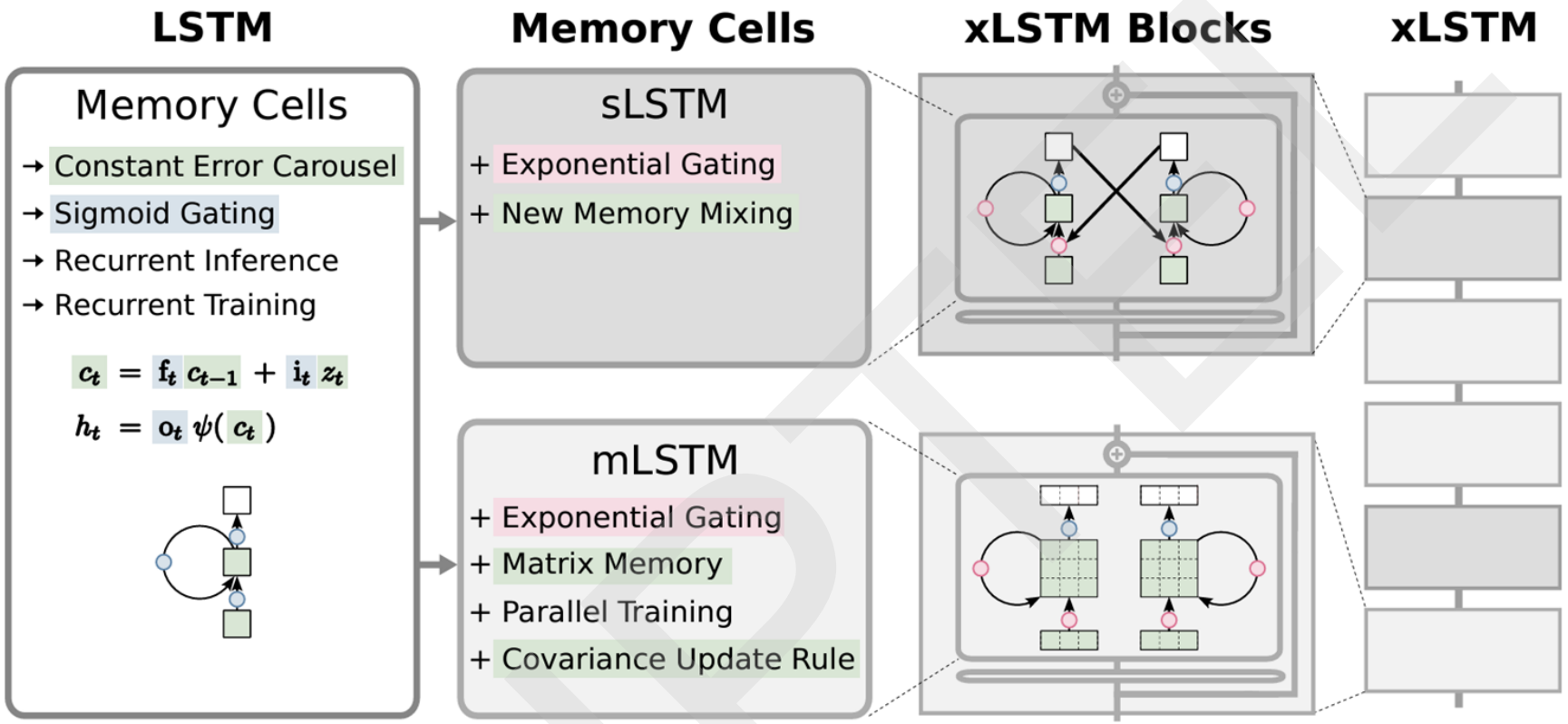


Figure 1: The extended LSTM (xLSTM) family. From left to right: 1. The original LSTM memory cell with constant error carousel and gating. 2. New sLSTM and mLSTM memory cells that introduce exponential gating. sLSTM offers a new memory mixing technique. mLSTM is fully parallelizable with a novel matrix memory cell state and new covariance update rule. 3. mLSTM and sLSTM in residual blocks yield xLSTM blocks. 4. Stacked xLSTM blocks give an xLSTM architecture.

xLSTM: Extended Long Short-Term Memory. https://arxiv.org/pdf/2405.04517

REFERENCES

Daniel Jurafsky and James H. Martin. 2024. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd edition. Online manuscript released August 20, 2024. [Chapter 8]


THANK YOU
NPTEL