

# CSCI 5541: Natural Language Processing

## Lecture 9: Tokenization, Embeddings, Attention, Transformers



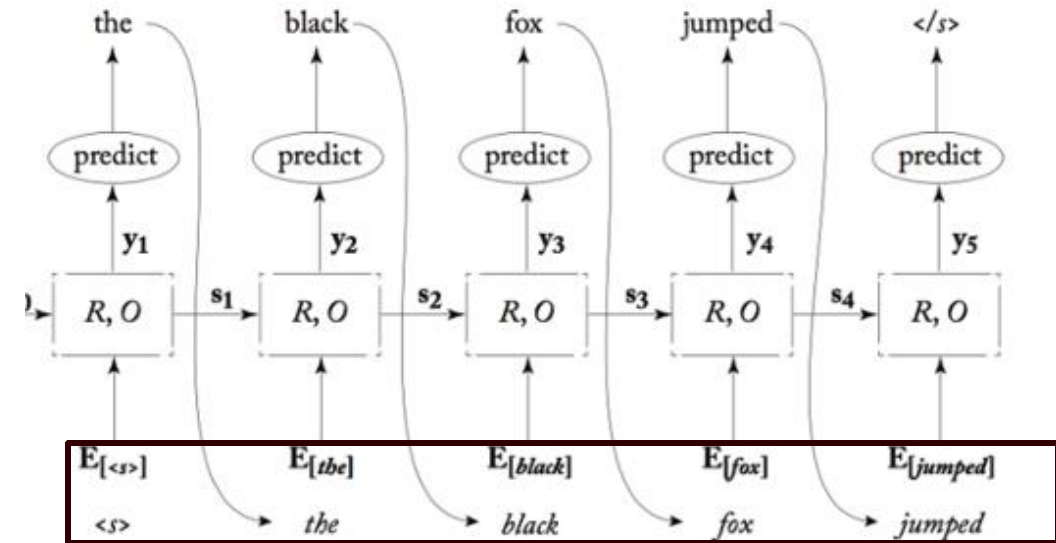
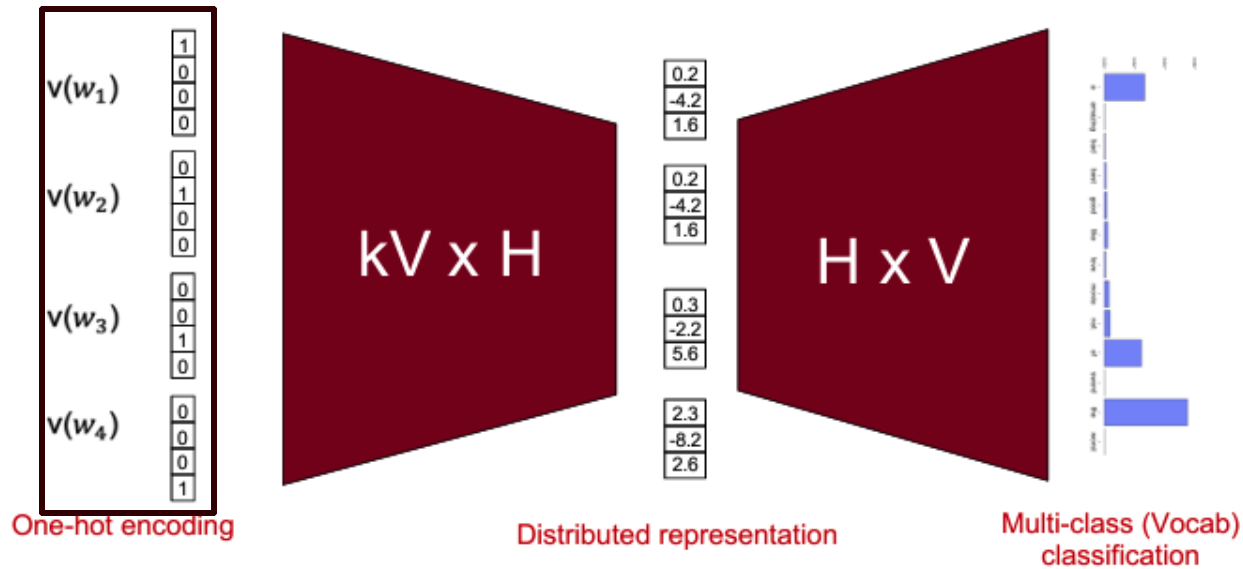
UNIVERSITY OF MINNESOTA  
Driven to Discover®

# Tokenization

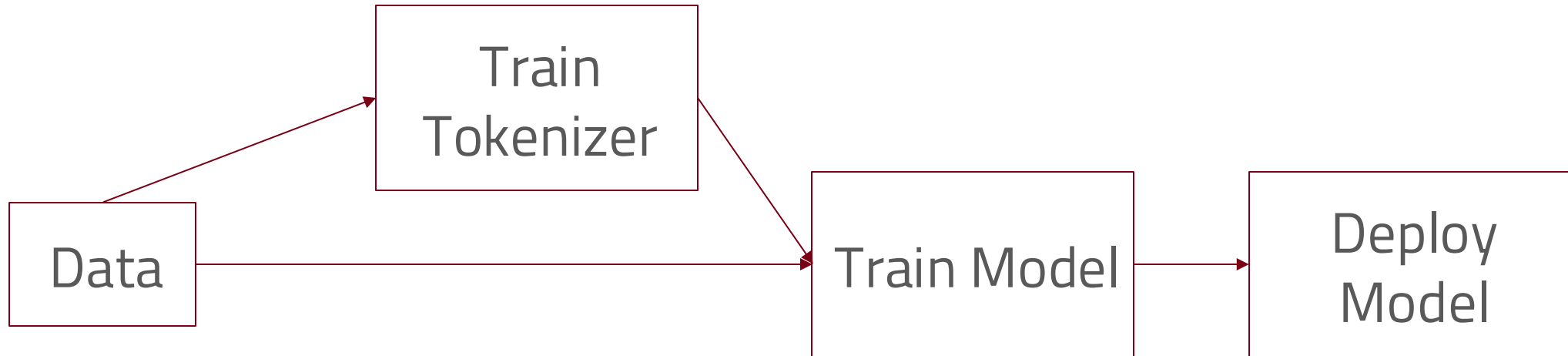


# How do we go from words to vectors?

How does one-hot encoding work?



# Typical Pipeline for LLMs

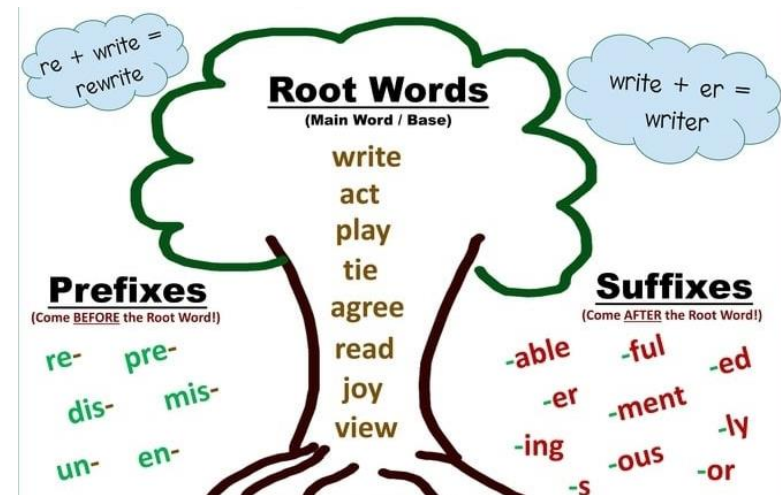


# Tokenization

- Up to this point, we have assumed the inputs to our MLP/RNNs were one-hot encoded words
- But, how do we handle
  - punctuation?
  - Mis-spellings?
  - Overlapping suffixes/prefixes/roots?
- Too much abstraction, let's dig in

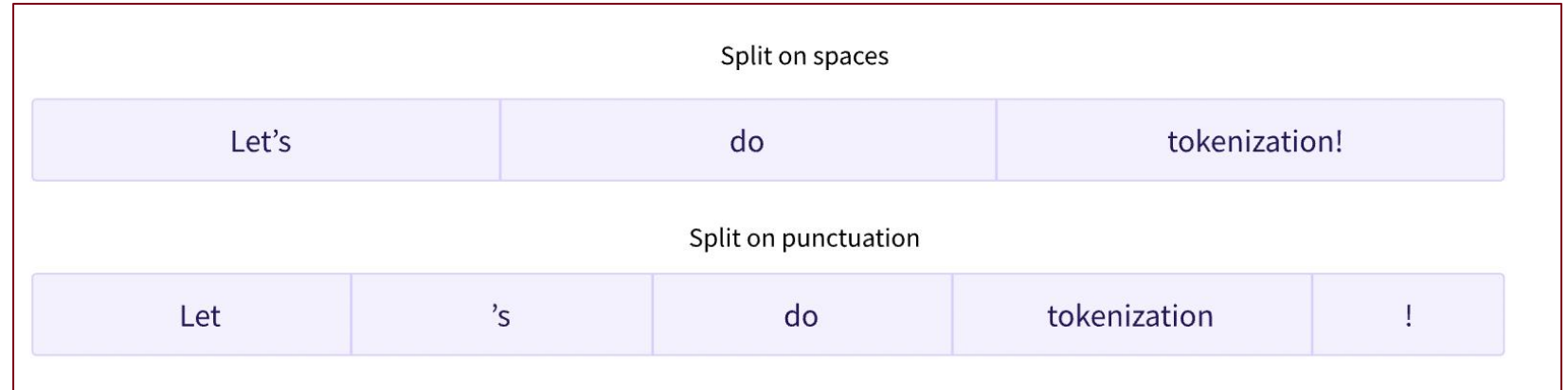
• Period / Full Stop	， Comma	？ Question Mark	！ Exclamation Point
： Colon	； Semicolon	’ Apostrophe	“ ” Quotation Marks
( ) Parentheses	— Dash	- Hyphen	… Ellipsis
[ ] Square Brackets	/ Slash	<b>Punctuation Marks</b> www.Games4esl.com	

## Proofreading



# Tokenization Approaches

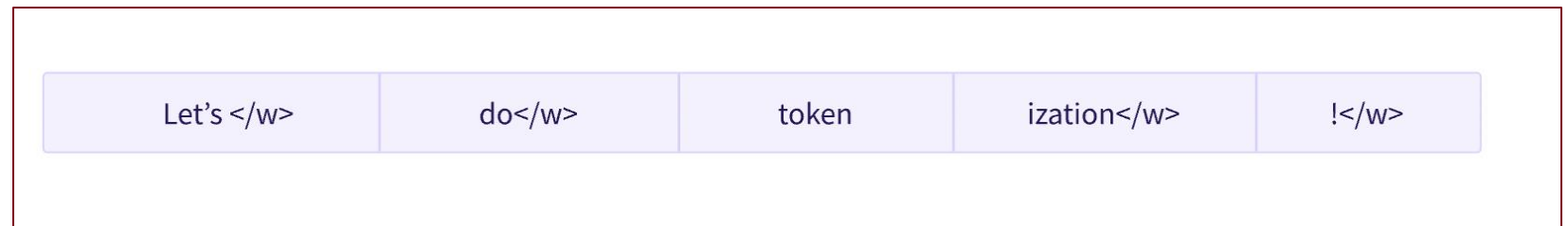
Word Based



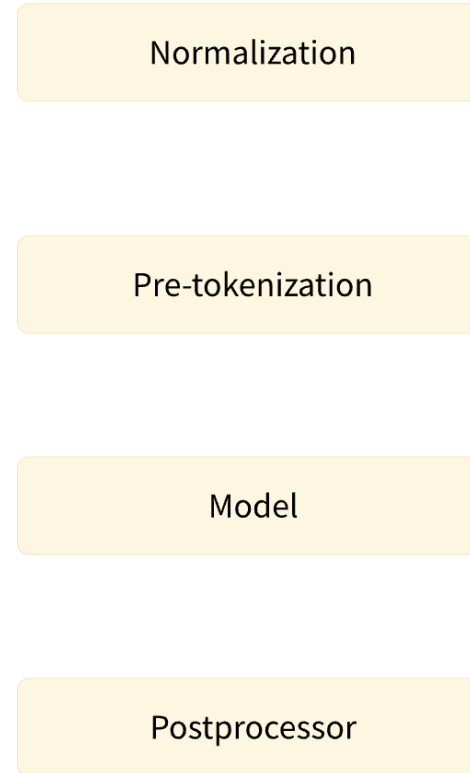
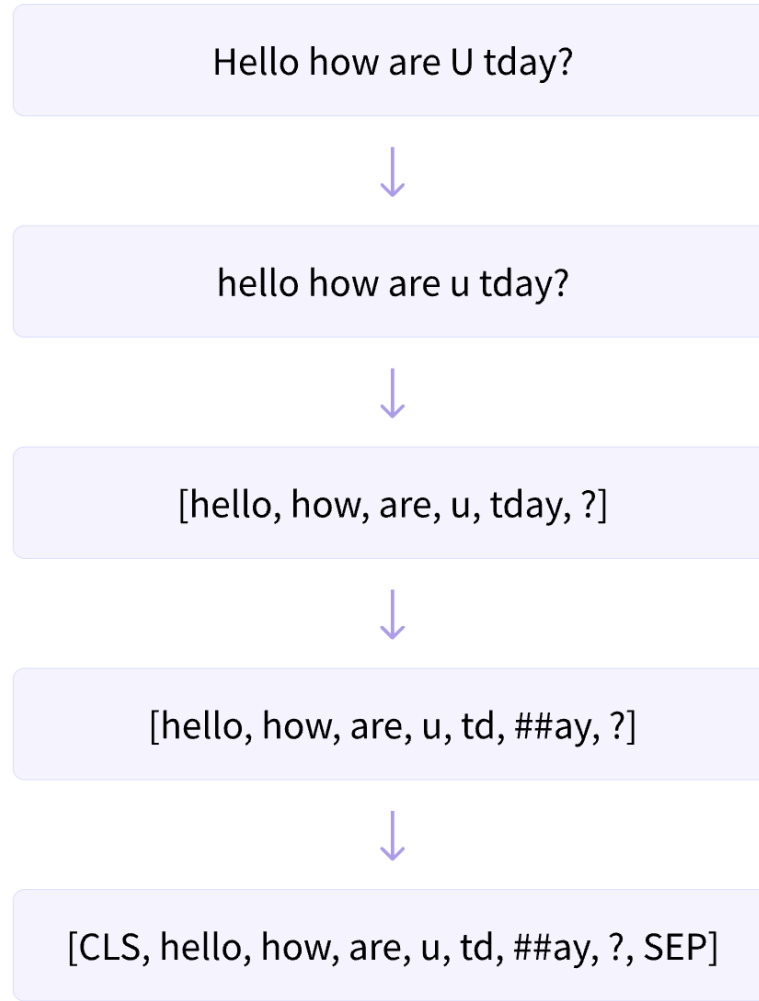
Character Based



Sub-word Based



# Tokenization Pipeline

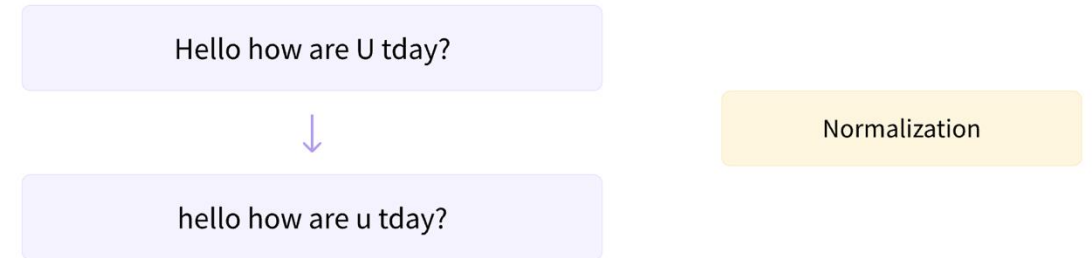


<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# Normalization

- ❑ First stage of cleaning up text (optionally applying some of the below)
- ❑ Remove excess whitespace
- ❑ Remove accents
- ❑ Lowercase words



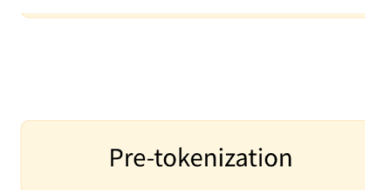
<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>





# Pre-Tokenization

- ❑ Convert a single string into a list of strings
- ❑ Split is typically performed by whitespace
- ❑ With Character tokenization, we split on each character and skip to the postprocessor step

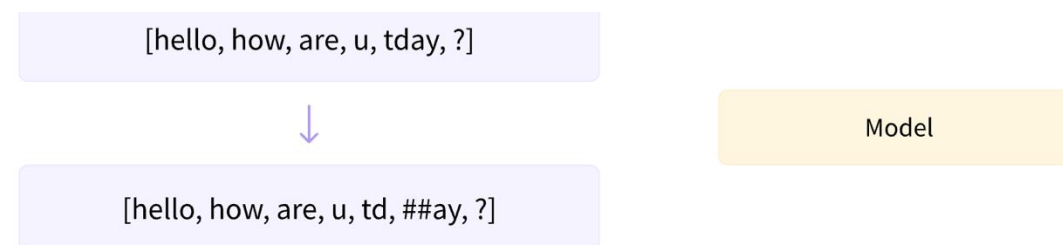


<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# Model

- ❑ Alter the array of strings from the previous step according to the vocabulary the given tokenizer has learned
- ❑ This step is specific to each tokenizer (WordPiece/BPE/etc.)
- ❑ From right, 'td' and '##ay' are in the vocabulary
- ❑ Special strings ('##') are appended in this case to denote a token which does not begin a word

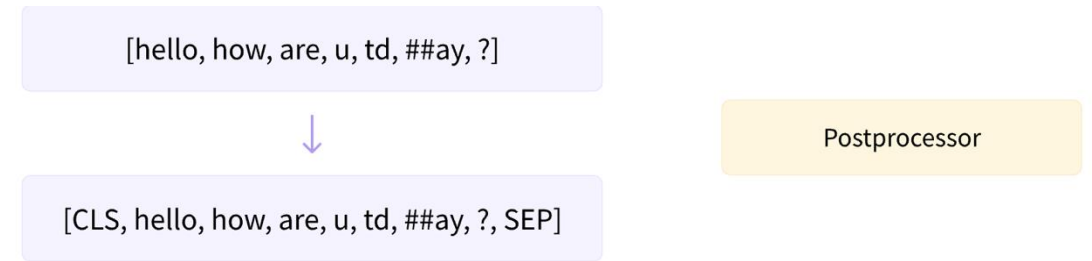


<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# Postprocessor

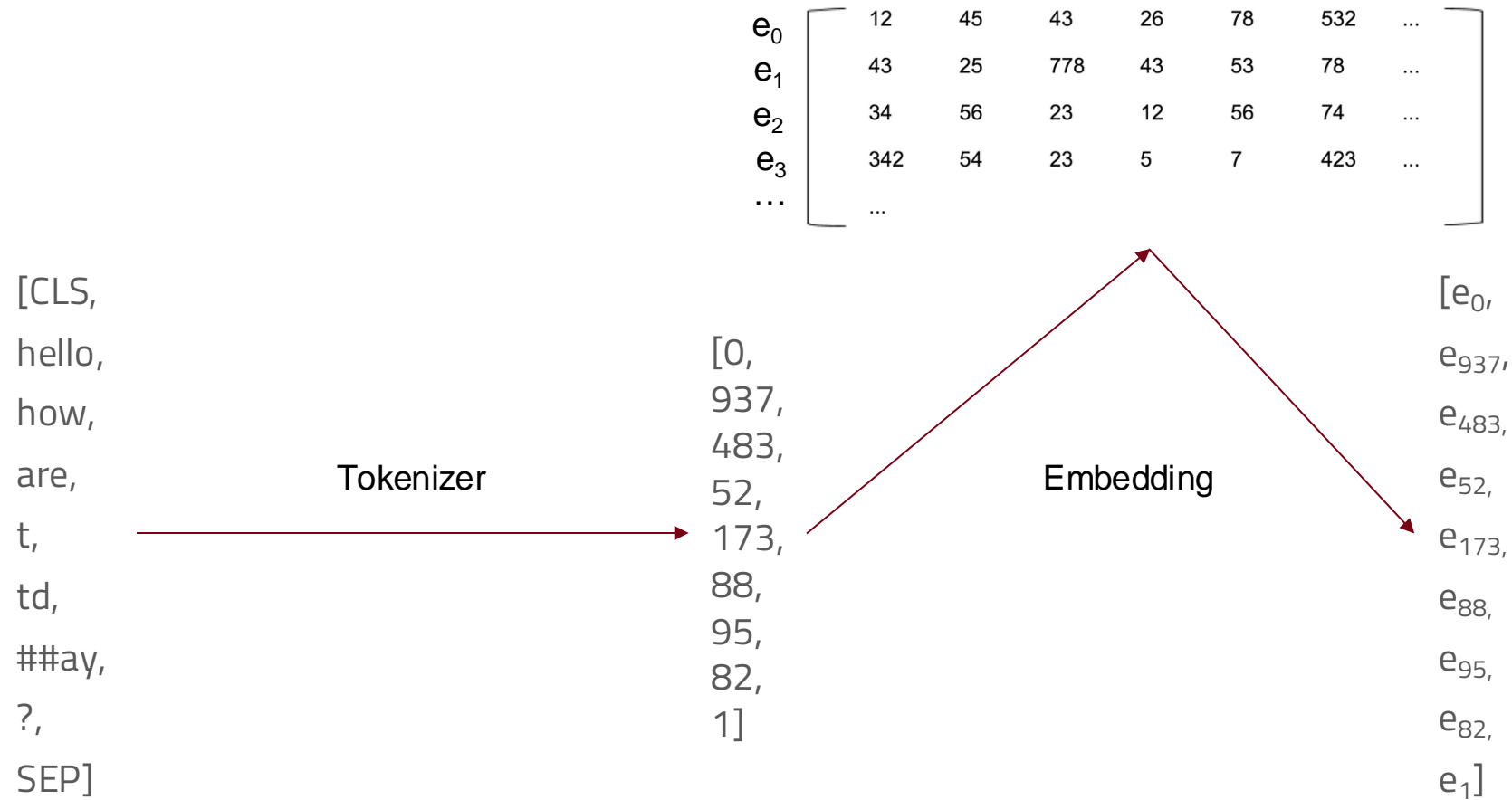
- ❑ Add beginning and end of sentence tokenizer
- ❑ Other tokenizer specific steps are taken here



<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



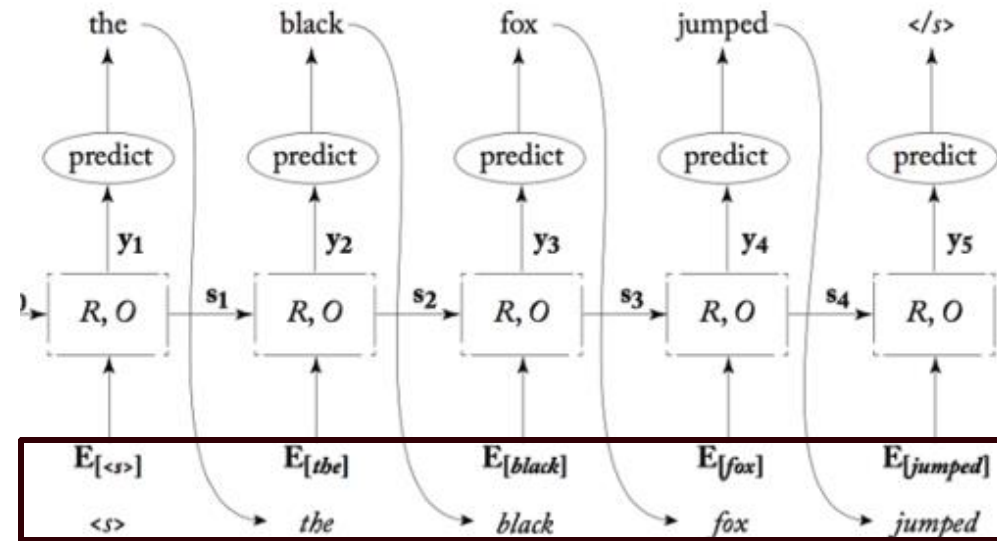
# Vector of strings to sequence of vectors



<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>

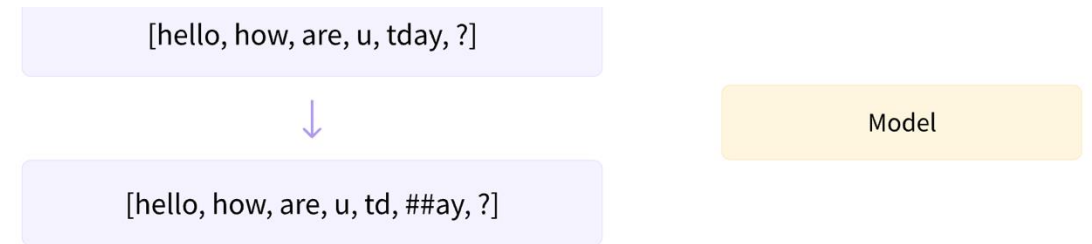


# Sentences become a sequence of vectors



# But how are tokenizers trained?

- ❑ How do we choose to go from 'tday' to 'td' and '##ay'?
- ❑ How does our model 'know' to do this?

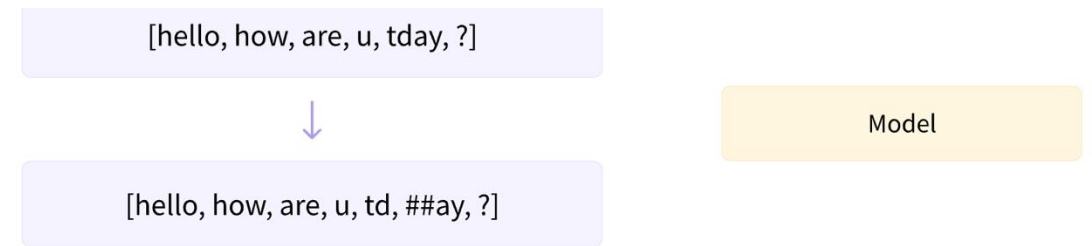


<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# ... by building a vocabulary

- ❑ We want to build out a complete list of all possible tokens that we should use as our list after the 'model' step
- ❑ The size of all possible tokens should be big enough to contain a sufficient number of tokens, but not too big to contain extremely rare tokens (most modern LLMs around ~100k)



<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# BPE Tokenizer

1. Apply normalization and pre-tokenization to a large dataset of text to get it into a standard format
2. This leaves us with a large set of strings
3. Start with a vocabulary of the characters present in this set of strings
4. Join the most common pair of successive elements in our vocabulary
5. Repeat (4) until the desired size of the vocabulary is reached





# BPE Tokenizer (Example)

1. Normalization and pretokenization applied
2. Our current set of strings is denoted at right (with corresponding frequencies)

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# BPE Tokenizer (Example)

1. Normalization and pretokenization applied
2. Our current set of strings is denoted at right (with corresponding frequencies)
3. Vocabulary representing all characters in this set

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

$V = \{ 'h', 'u', 'g', 'p', 'n', 'b', 's' \}$

<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# BPE Tokenizer (Example)

1. Normalization and pretokenization applied
2. Our current set of strings is denoted at right (with corresponding frequencies)
3. Vocabulary representing all characters in this set
4. Add most common occurring pair of elements in current vocabulary

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

$V = \{ 'h', 'u', 'g', 'p', 'n', 'b', 's' \}$

$|'hu'| = 15, |'ug'| = \mathbf{20}, |'pu'| = 17, |'un'| = 16, |'bu'| = 4, |'gs'| = 5$

$V = \{ 'h', 'u', 'g', 'p', 'n', 'b', 's', 'ug' \}$

<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# BPE Tokenizer (Example)

1. Normalization and pretokenization applied
2. Our current set of strings is denoted at right (with corresponding frequencies)
3. Vocabulary representing all characters in this set
4. Add most common occurring pair of elements in current vocabulary
5. Repeat until desired vocab size

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

$V = \{ 'h', 'u', 'g', 'p', 'n', 'b', 's' \}$

$|'hu'| = 15, |'ug'| = \mathbf{20}, |'pu'| = 17, |'un'| = 16, |'bu'| = 4, |'gs'| = 5$

$V = \{ 'h', 'u', 'g', 'p', 'n', 'b', 's', 'ug' \}$

...

<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# Other Tokenizers

## WordPiece

$$\text{score} = (\text{freq\_of\_pair}) / (\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$$

## Unigram

$$P(["p", "u", "g"]) = P("p") \times P("u") \times P("g") = \frac{5}{210} \times \frac{36}{210} \times \frac{20}{210} = 0.000389$$

<https://huggingface.co/learn/nlp-course/en/chapter6/1?fw=pt>



# Embeddings



# Types and tokens

**Type:** gopher

5.2	1.5	...	0.2	0.6
-----	-----	-----	-----	-----

**Token:**

- The gopher is a resident of the dry plains.
- One day, while I was out chasing a gopher, I wandered off too far.
- It's not often a team loses with stats like this.  
Gophers played very well tonight.

"gopher"

5.2	1.5	...	0.2	0.6
-----	-----	-----	-----	-----

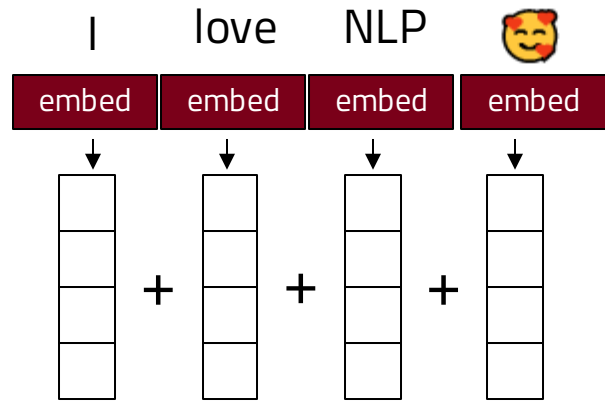
3.2	8.5	...	0.6	8.1
-----	-----	-----	-----	-----

-2.2	2.4	...	5.2	3.4
------	-----	-----	-----	-----

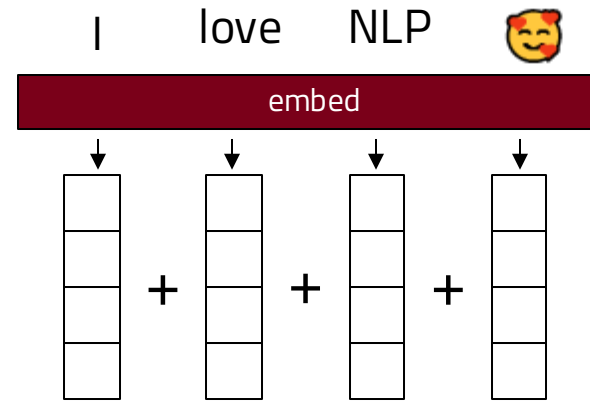


# Contextualization of word representations

Tokenization

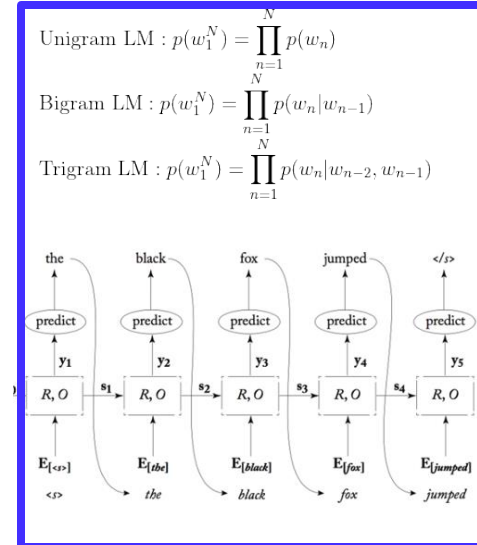


Static or non-contextualized representations



Contextualized representations

Language models





# Contextualized word representations

Transform the representation of a token in a sentence (e.g., from a static word embedding) to be sensitive to its **local context** in a sentence



# ELMo

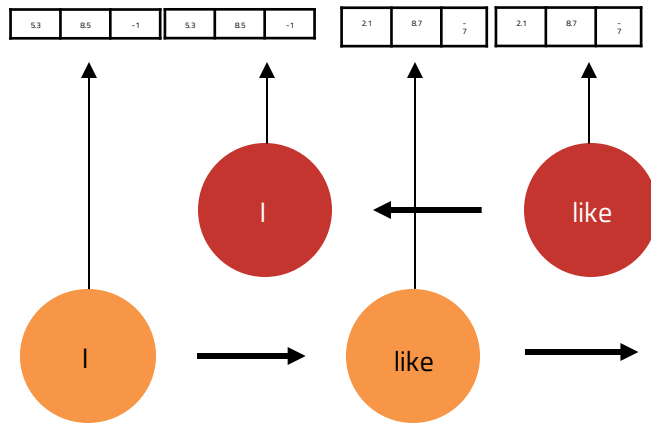
(Peters et al., 2018)



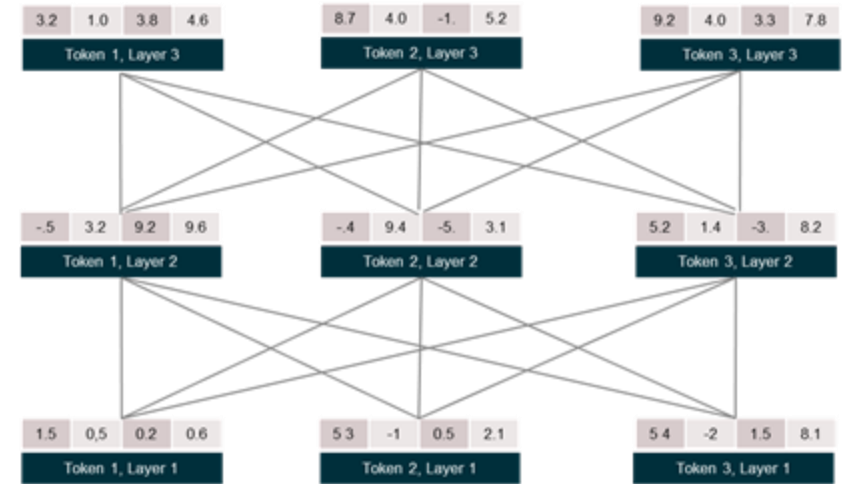
# BERT

(Devlin et al., 2019)

Stacked Bidirectional RNN trained to predict next word in language modeling task



Transformer-based model to predict masked word using bidirectional context and next sentence prediction



# ELMo

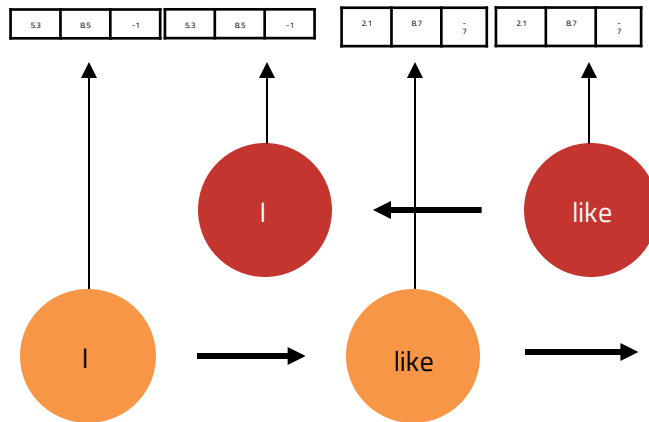
(Peters et al., 2018)



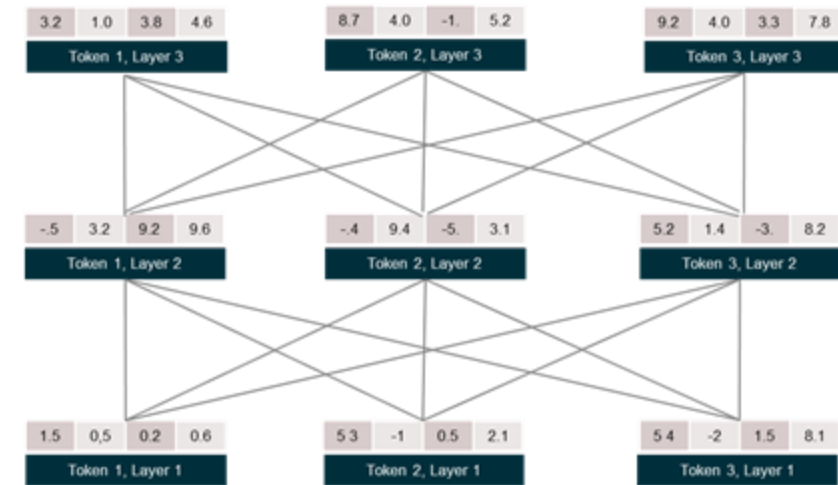
# BERT

(Devlin et al., 2019)

Stacked Bidirectional RNN trained to predict next word in language modeling task



Transformer-based model to predict masked word using bidirectional context and next sentence prediction

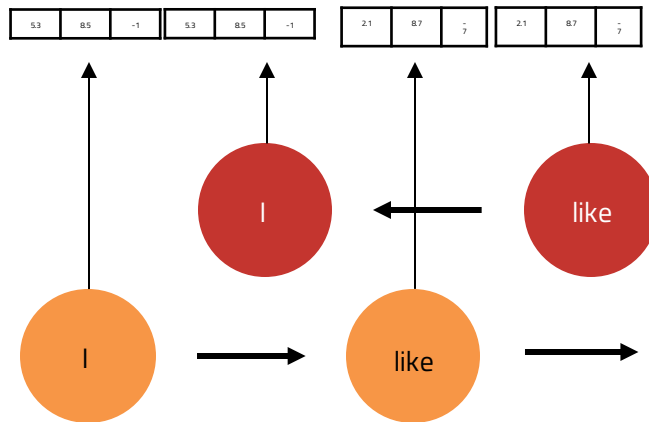


# ELMo

(Peters et al., 2018)



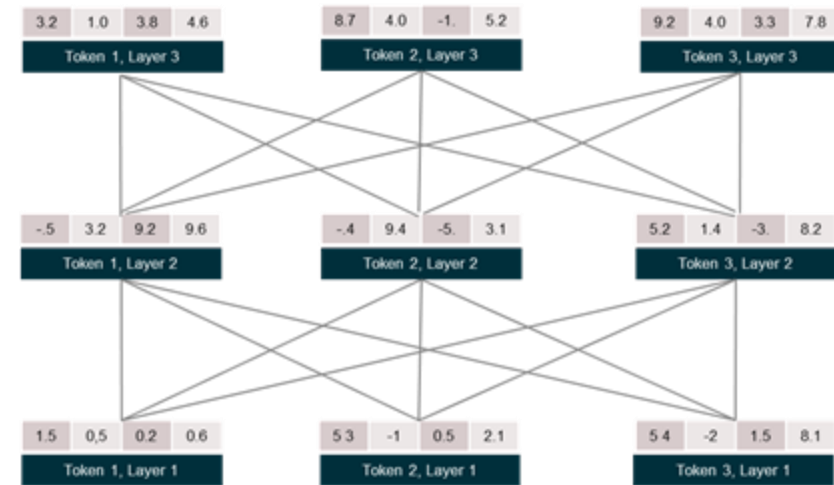
Stacked Bidirectional RNN trained to predict next word in language modeling task



# BERT

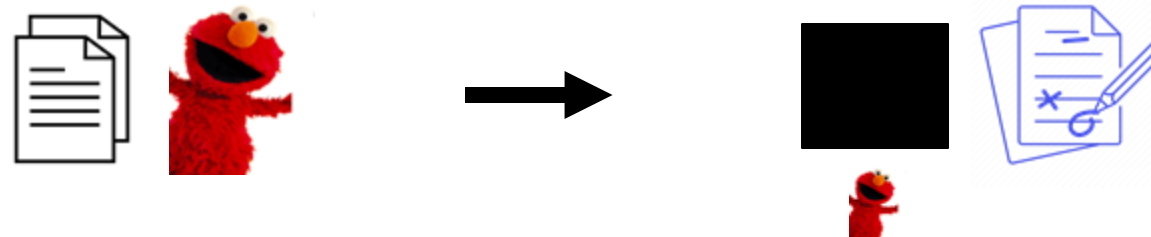
(Devlin et al., 2019)

Transformer-based model to predict masked word using bidirectional context and next sentence prediction



# ELMo (Embeddings from Language Models)

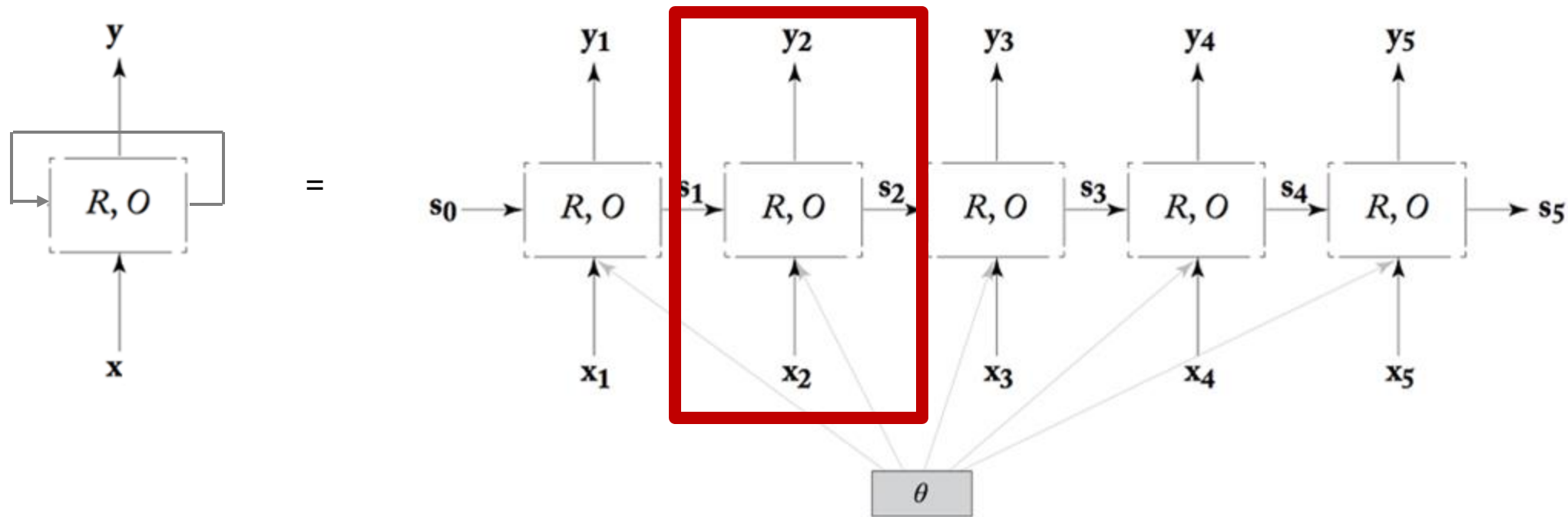
- ❑ Big idea: (i) transform the representation of a word (e.g., from a static word embedding) to be sensitive to its **local context** in a sentence and (ii) optimized for a specific NLP task.
- ❑ Output = word representations that can be **plugged into** just about any architecture a word embedding can be used.



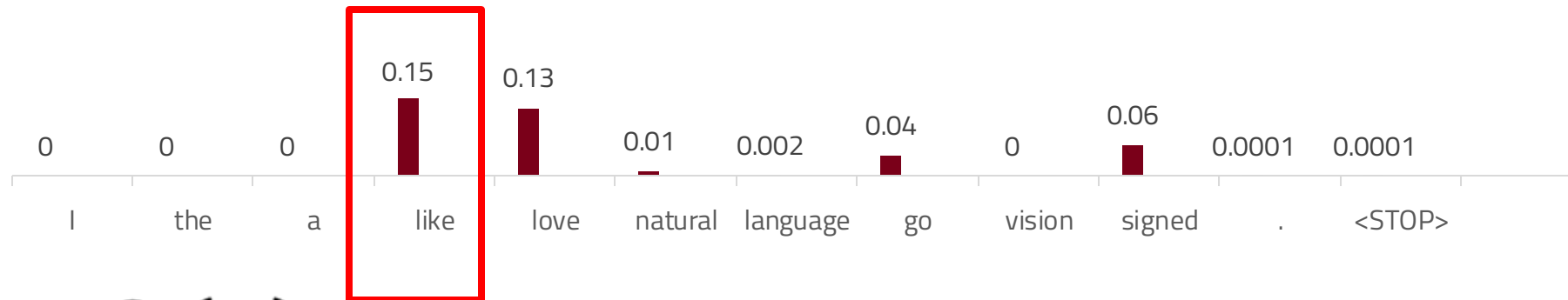
# Recurrent Neural Network



RNN allow arbitrarily-sized conditioning contexts;  
condition on the **entire sequence history**.

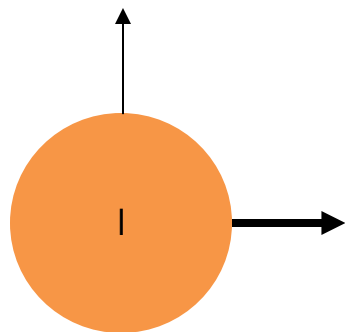


# Recurrent neural network language model



$$y_i = O(s_i)$$

5.3	8.5	-1	5.1
-----	-----	----	-----

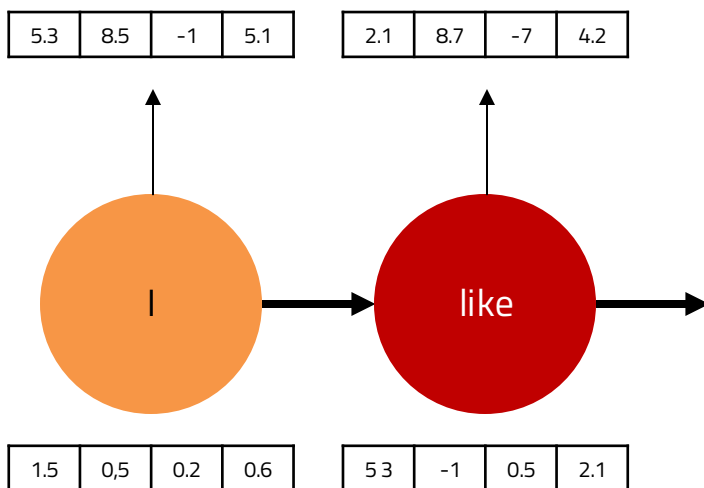


5.3
8.5
-1
5.1

$$s_i = R(x_i, s_{i-1})$$

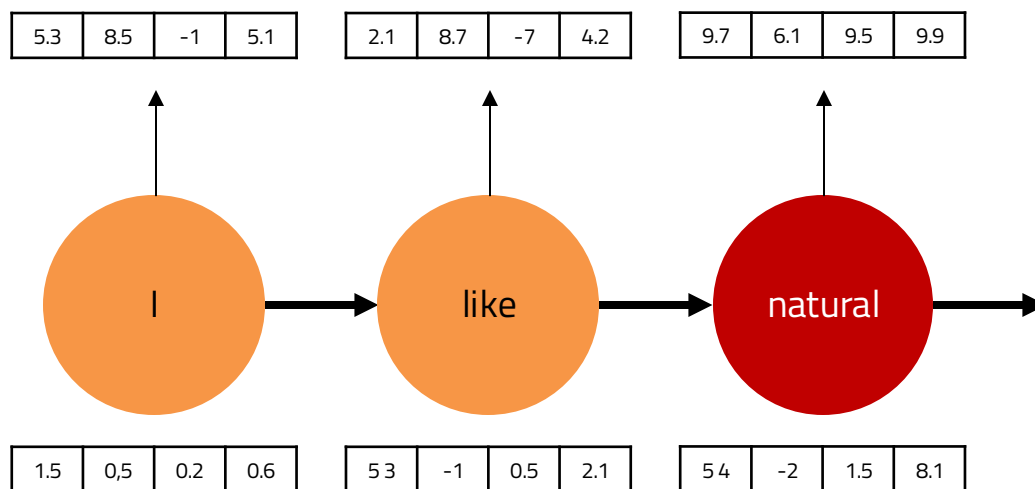
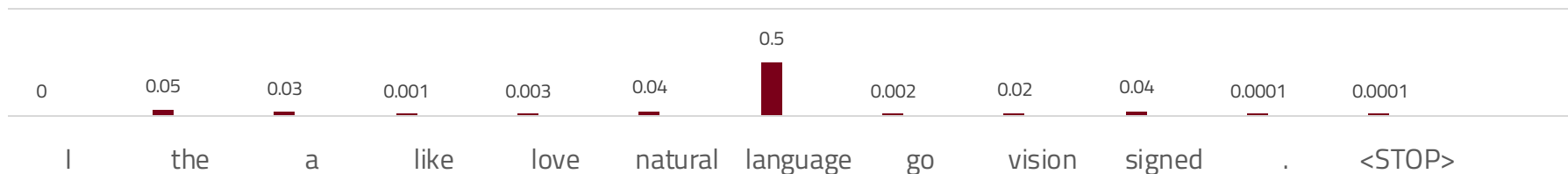


# Recurrent neural network language model

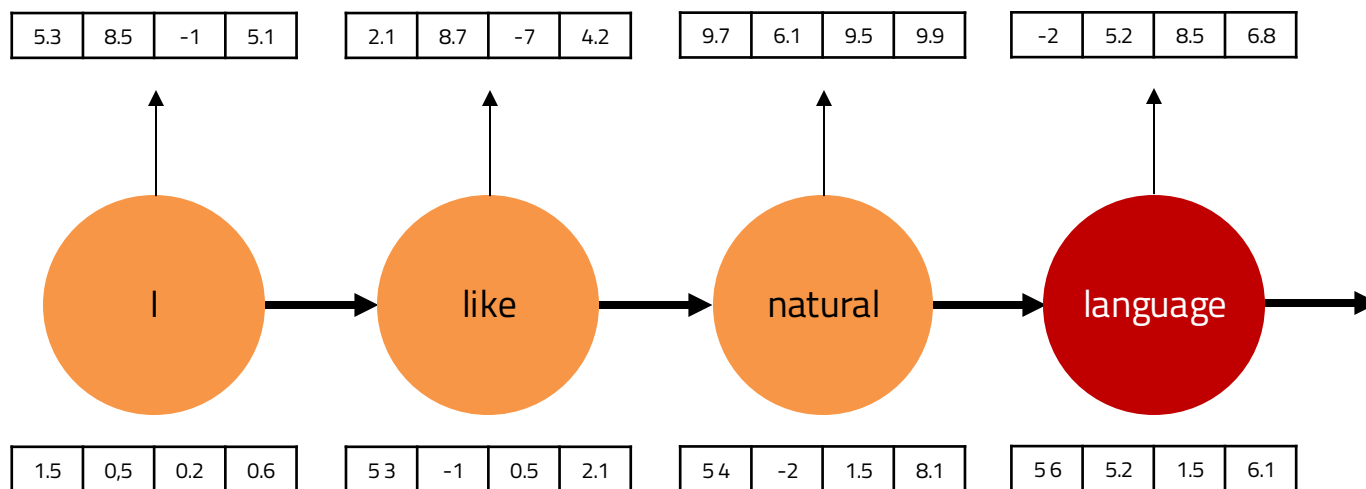
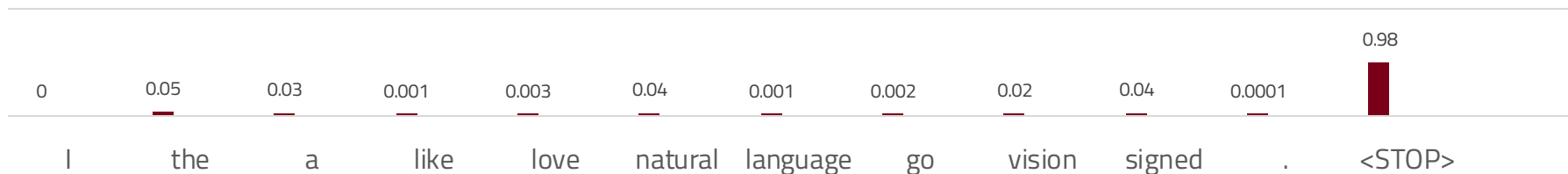




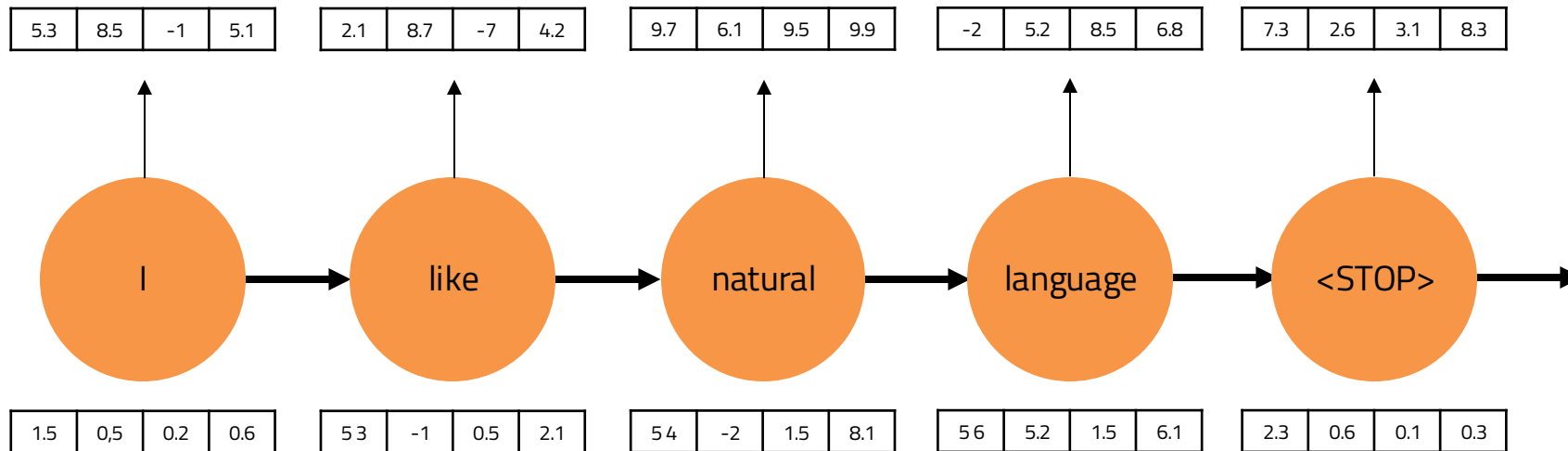
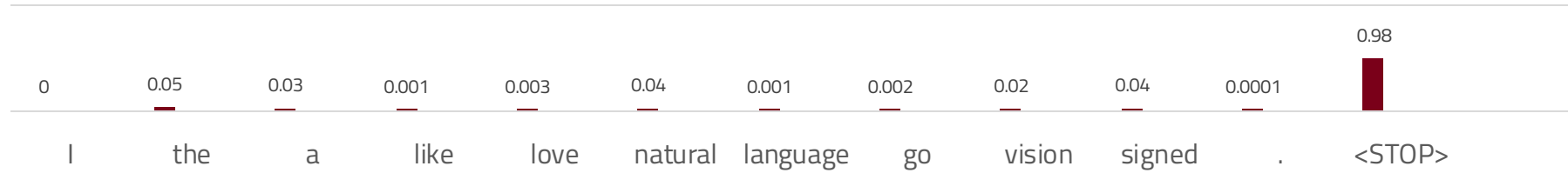
# Recurrent neural network language model



# Recurrent neural network language model

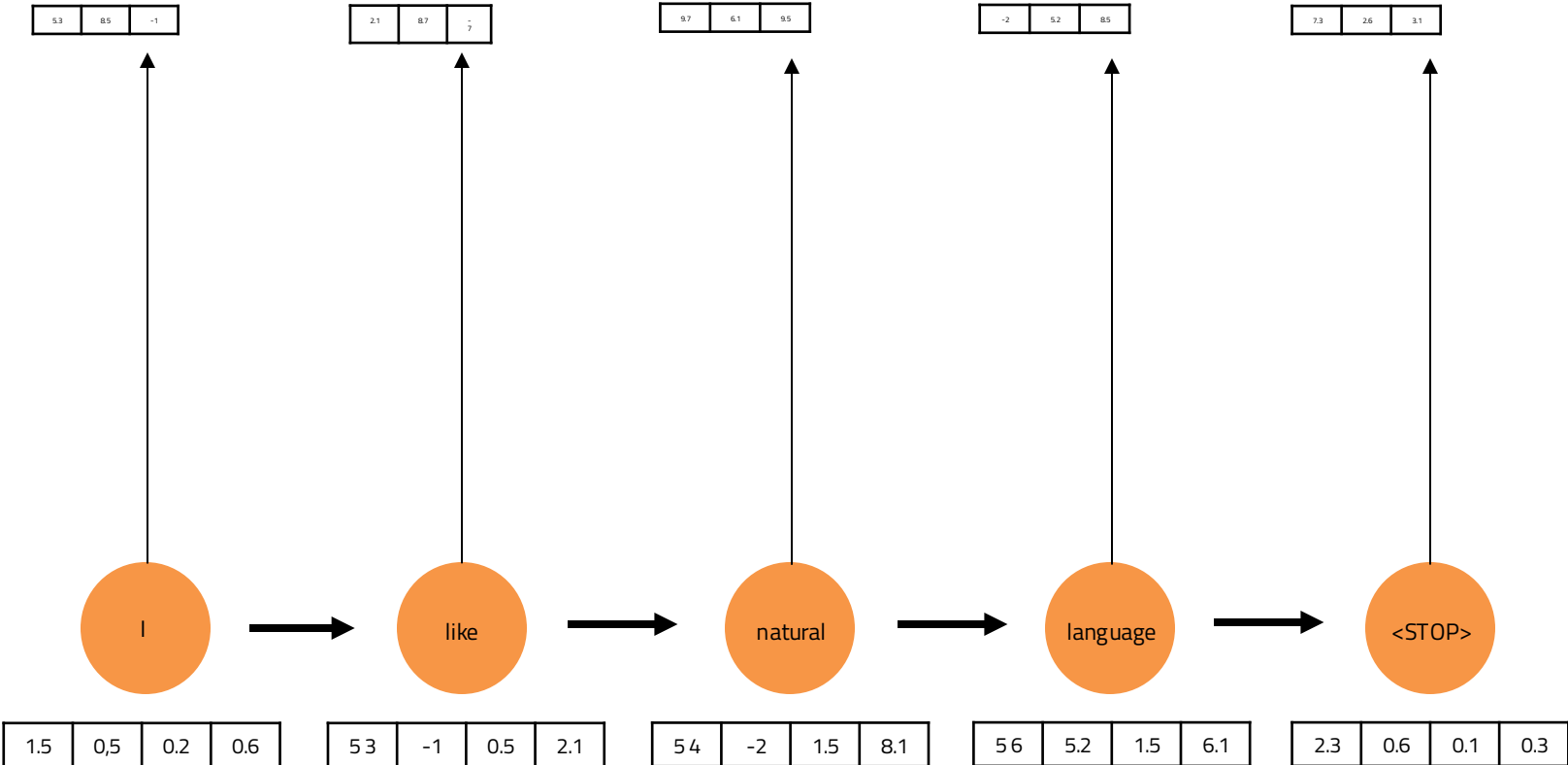


# Recurrent neural network language model



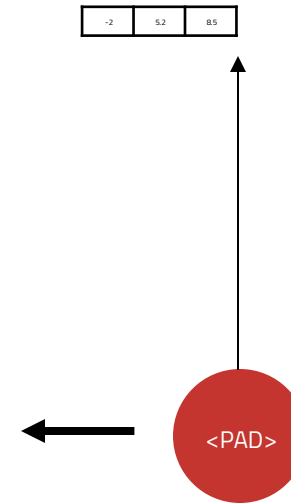
# Bidirectional RNN

Forward RNN



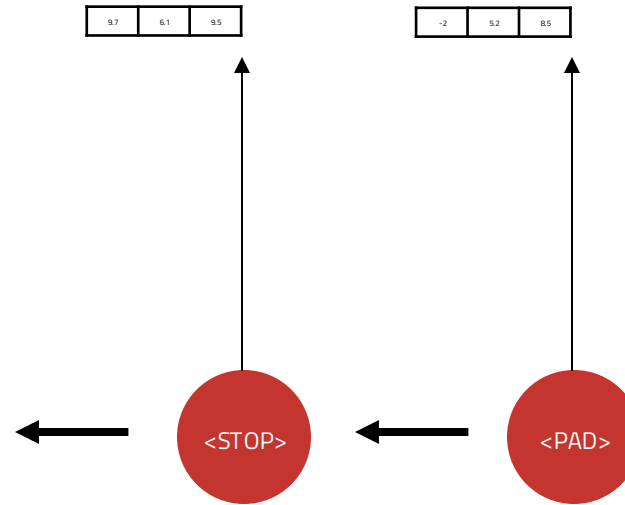
# Bidirectional RNN

*Backward RNN*



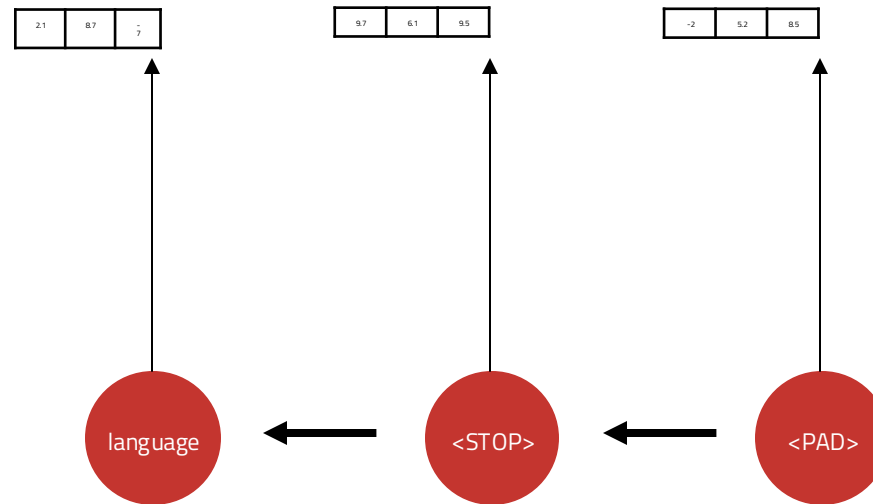
# Bidirectional RNN

*Backward RNN*



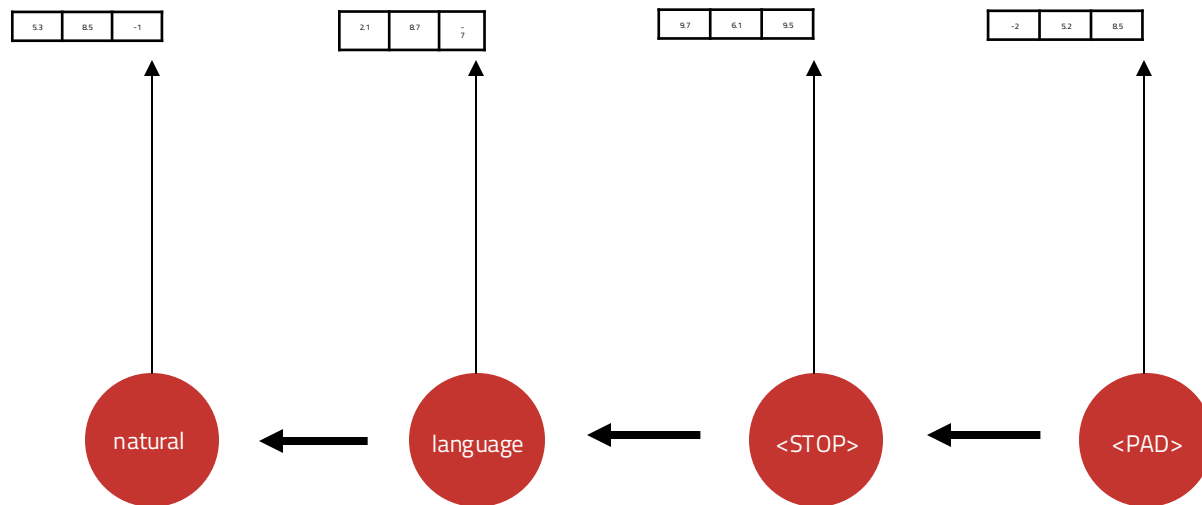
# Bidirectional RNN

*Backward RNN*



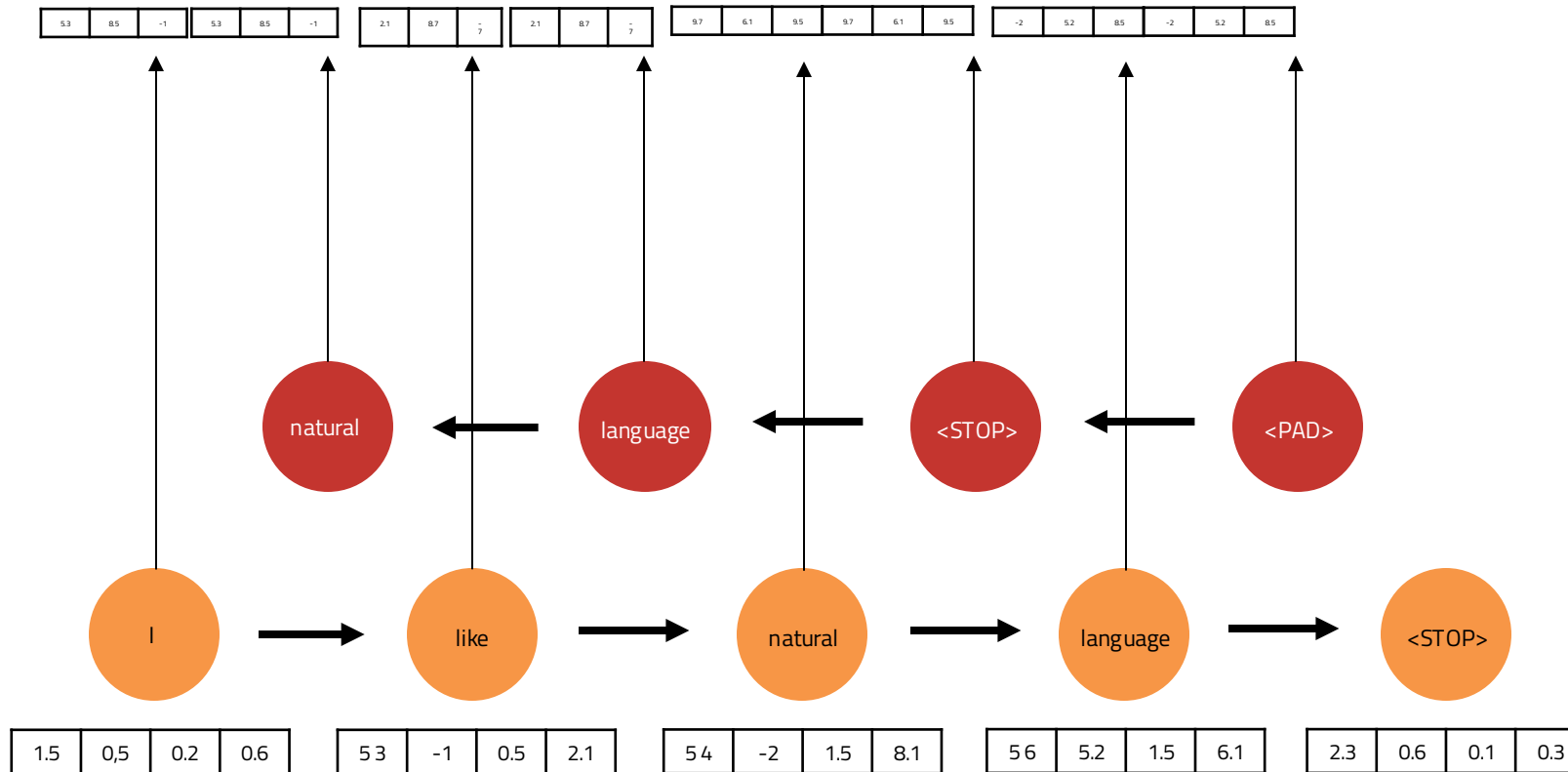
# Bidirectional RNN

*Backward RNN*



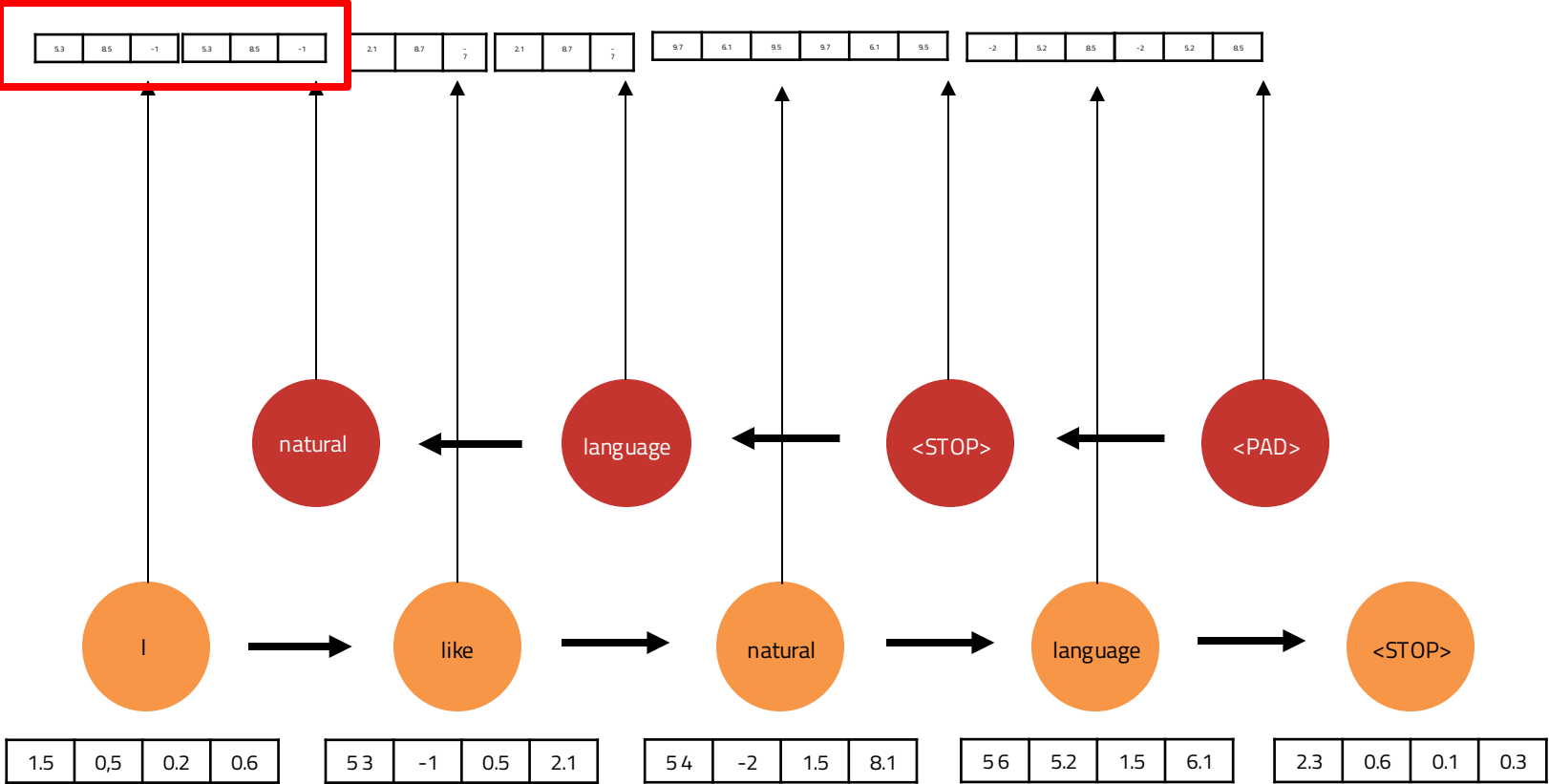


# Bidirectional RNN



# Bidirectional RNN

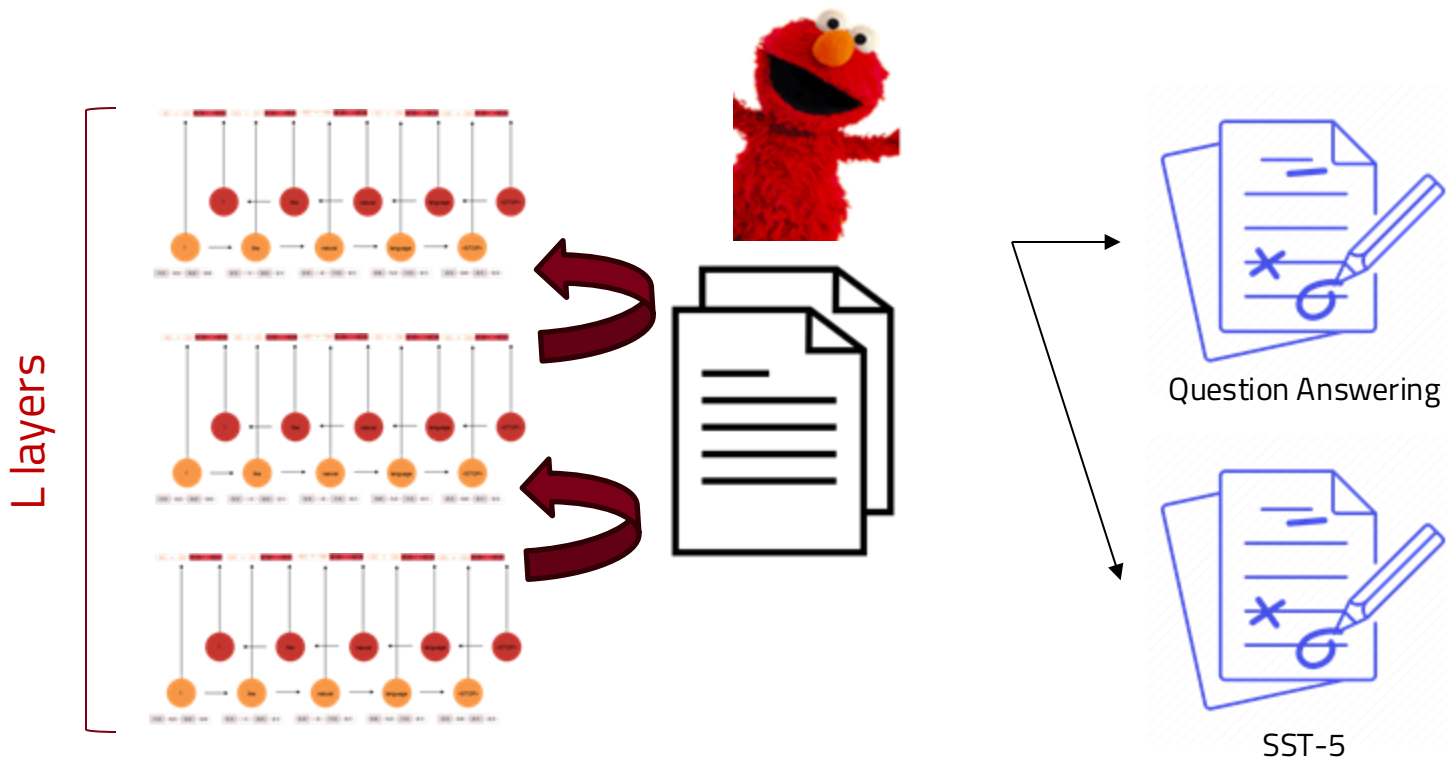
$$y_i = \mathbf{0} (s_i^f; s_i^b)$$



# ELMo (Embeddings from Language Models)

**Pre-training stage:**  
Train a Bi-RNN LM with L layers  
on unlabeled text corpora

**Fine-tuning stage:**  
Fine-tune it for a specific task by combining  
RNN output across all layers



TASK	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	85.8	4.7 / 24.9%
SNLI	88.7 ± 0.17	0.7 / 5.8%
SRL	84.6	3.2 / 17.2%
Coref	70.4	3.2 / 9.8%
NER	92.22 ± 0.10	2.06 / 21%
SST-5	54.7 ± 0.5	3.3 / 6.8%

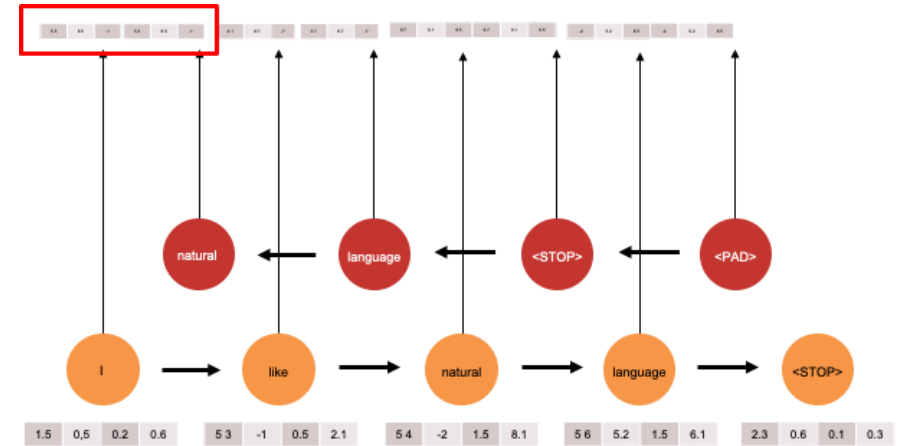
# Types and tokens

Type: **gopher**

5.2	1.5	...	0.2	0.6
-----	-----	-----	-----	-----

Token:

- The **gopher** is a resident of the dry plains.
- One day, while I was out chasing a **gopher**, I wandered off too far.
- It's not often a team loses with stats like this. **Gophers** played very well tonight.



"gopher"

5.2	1.5	...	0.2	0.6
-----	-----	-----	-----	-----

3.2	8.5	...	0.6	8.1
-----	-----	-----	-----	-----

-2.2	2.4	...	5.2	3.4
------	-----	-----	-----	-----



The **gopher** football team began playing at TCF Bank Stadium.

1.5
0.5
0.7
-3.6



2.5
1.4
2.6
-4.4



Ski-U-Mah, **gophers!**

The **gopher** is a resident of the dry plains.

4.2
0.7
-5.2
0.1
...



5.2
0.5
-6.2
0.5
...

One day, while I was out chasing a **gopher**, I wandered off too far.

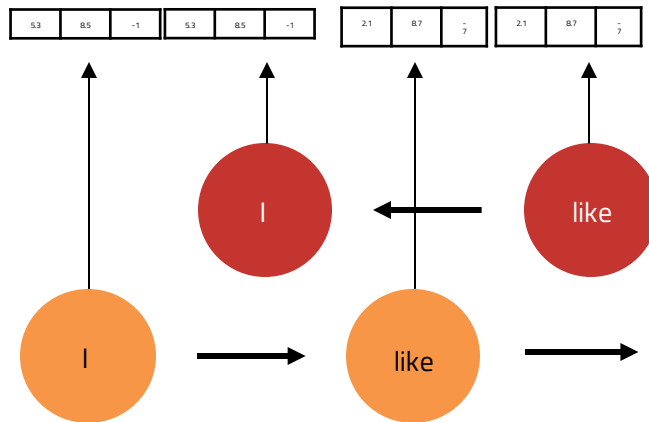


# ELMo

(Peters et al., 2018)



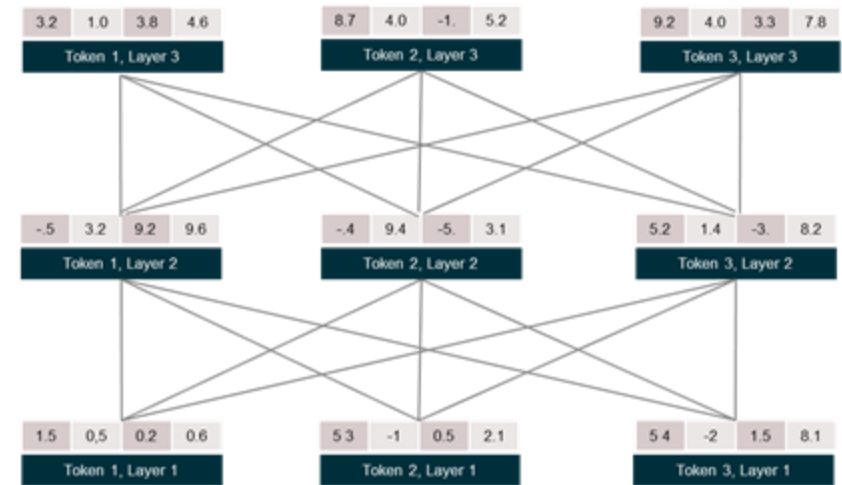
Stacked Bidirectional RNN trained to predict next word in language modeling task



# BERT

(Devlin et al., 2019)

Transformer-based model to predict masked word using bidirectional context and next sentence prediction



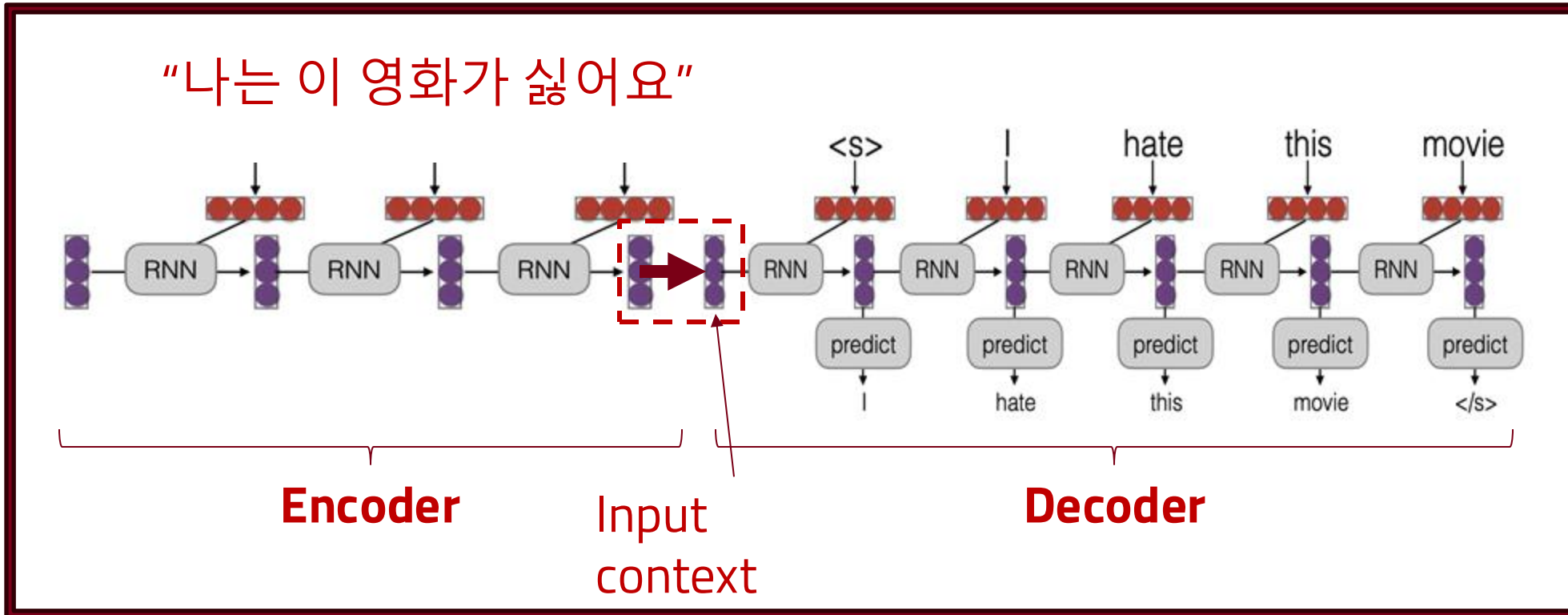
# Attention



# Old Seq2Seq Implementation



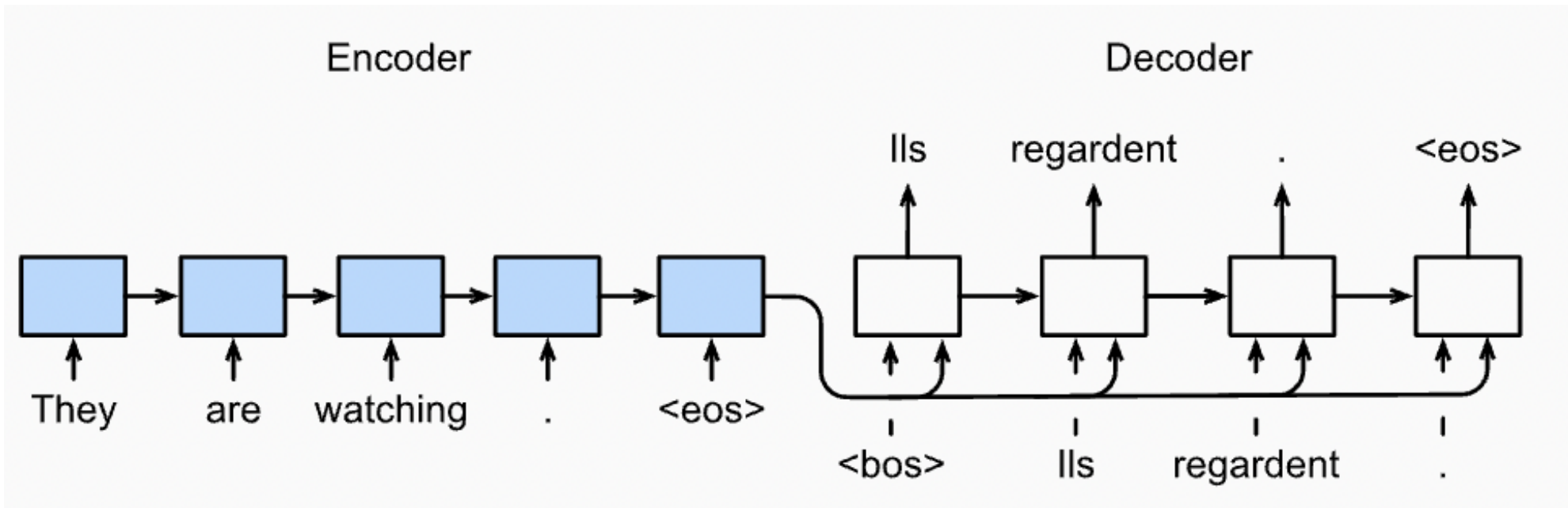
The input context serves as a significant bottleneck. Most modern language models (transformers) implement some improvements upon this → We'll revisit this ~~in the coming weeks~~ now





# Alleviating the bottleneck

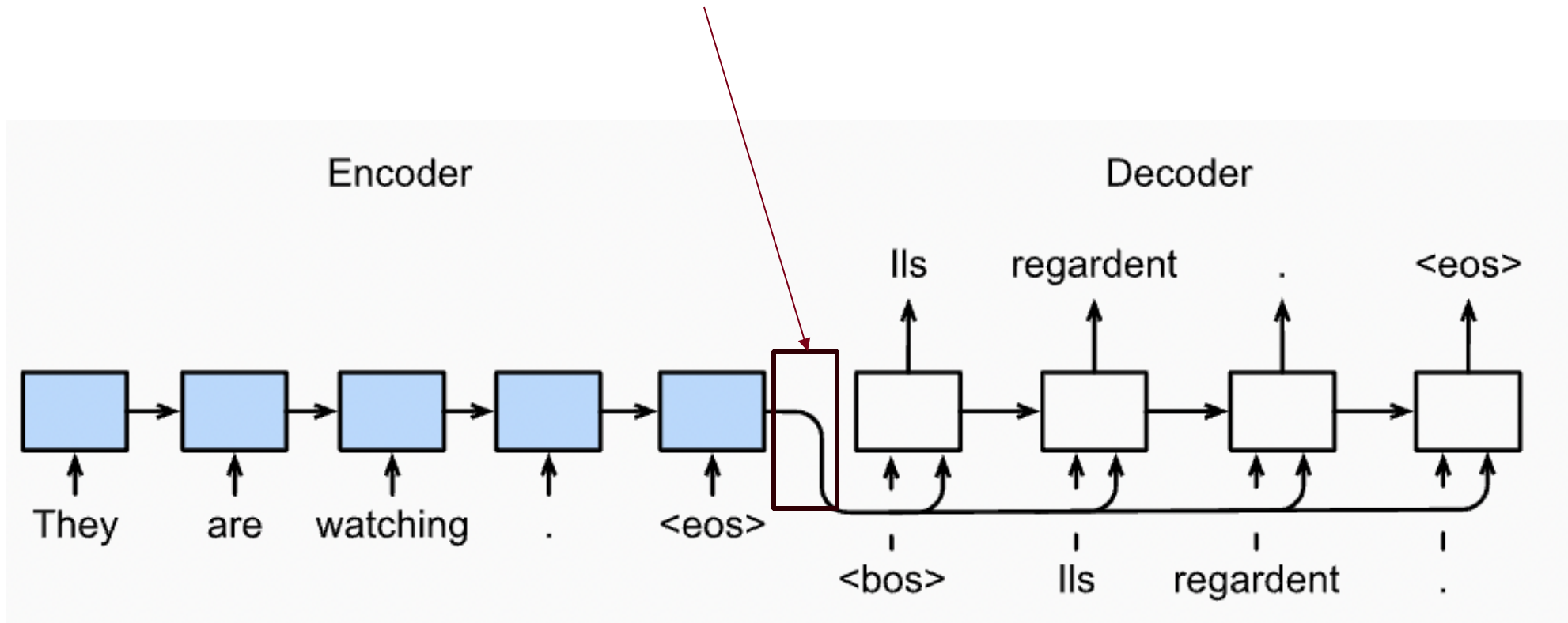
Encoder has more channels to communicate over...



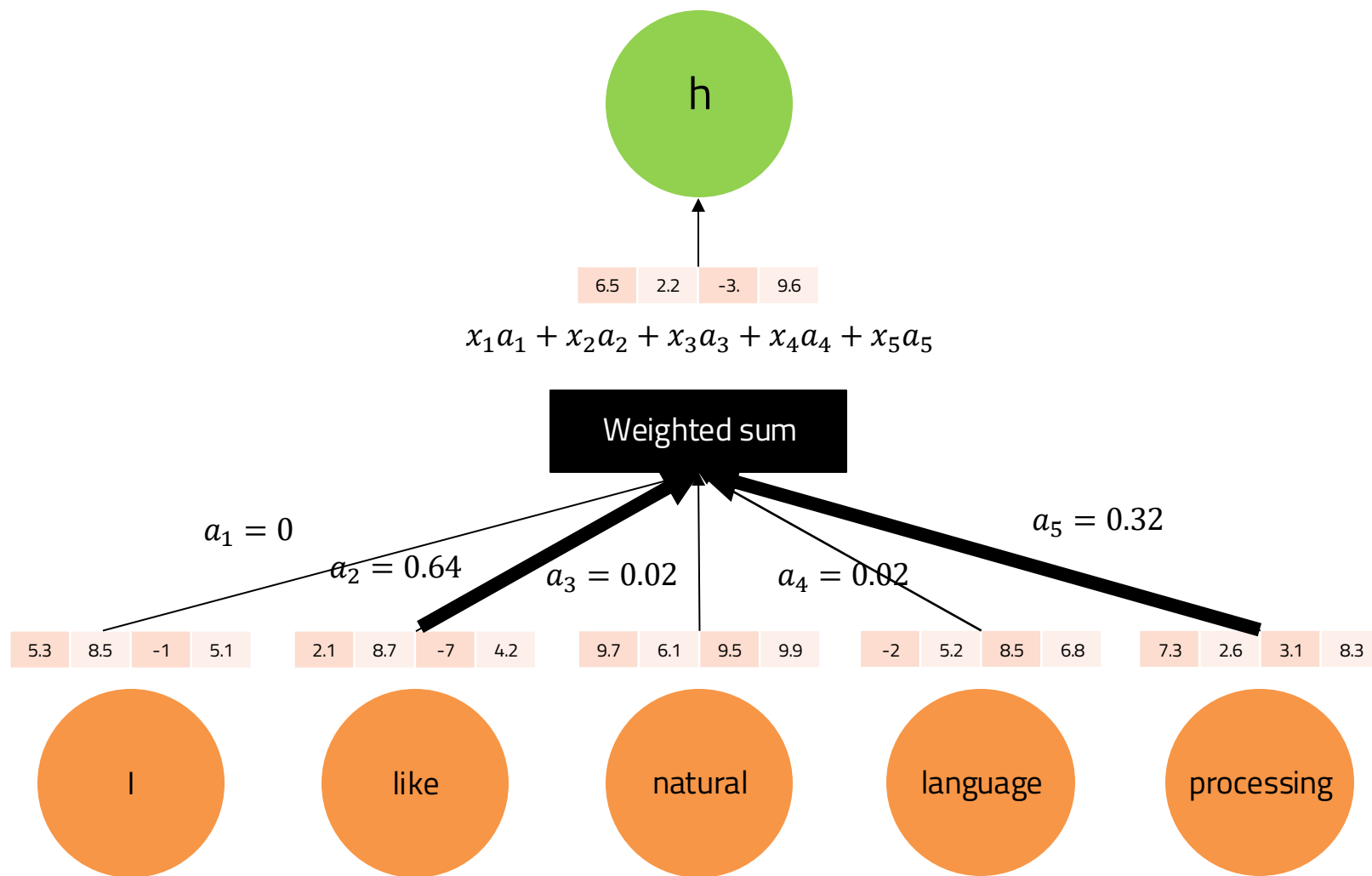
# Alleviating the bottleneck

Encoder now has more channels to communicate over...

but all the context must follow the final hidden state of the encoder



# Enter Attention



# Enter Attention

- Attention allows each output element to focus on only the relevant parts of the input sequence
- There is no longer a hidden state bottleneck – the model can focus on *any* hidden state in the input sequence

$$e_{ij} = a(s_{i-1}, h_j)$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

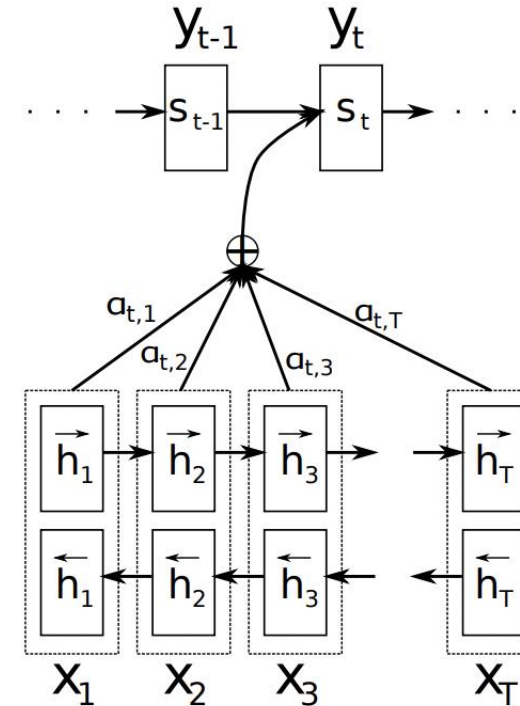
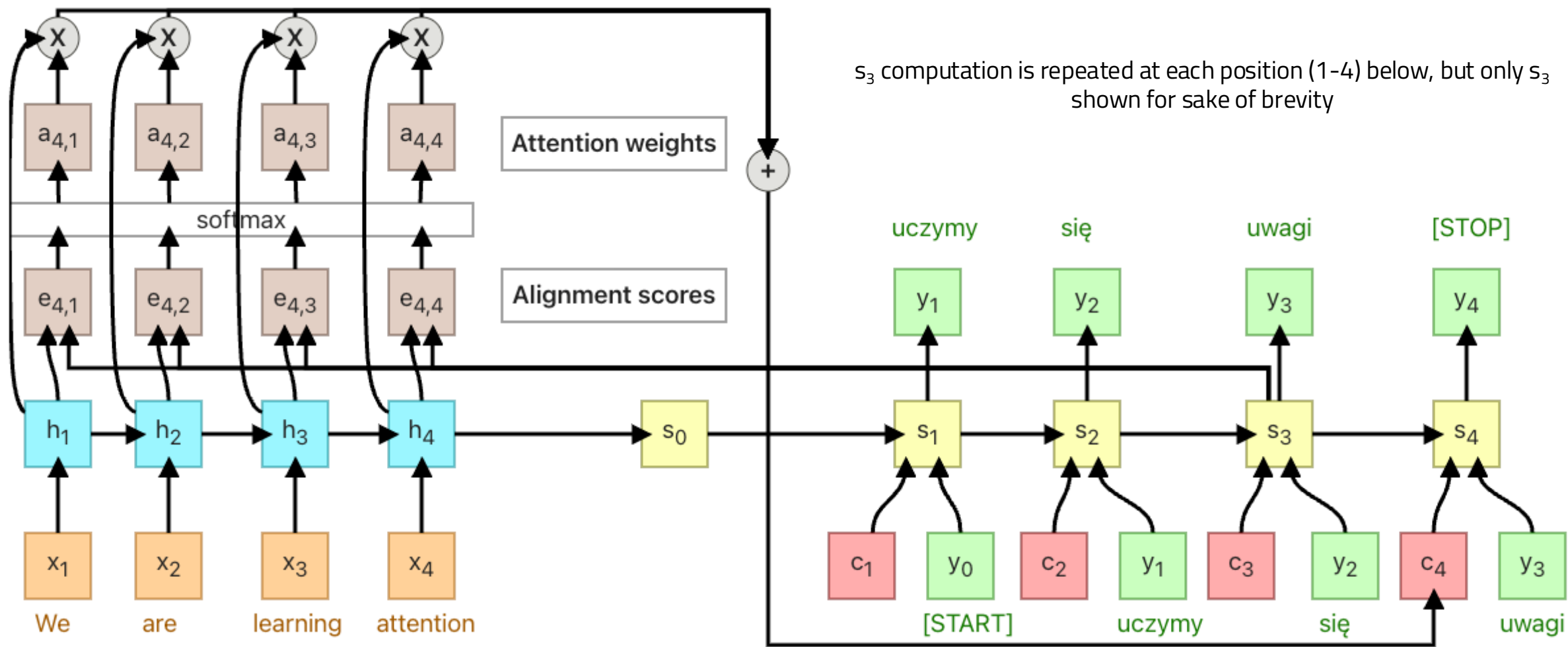


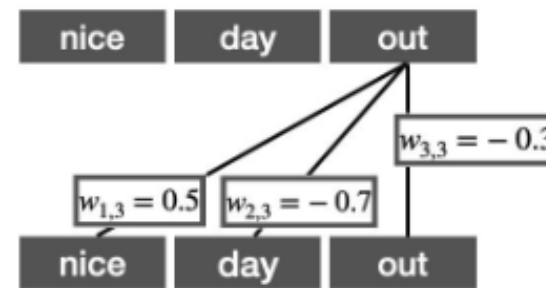
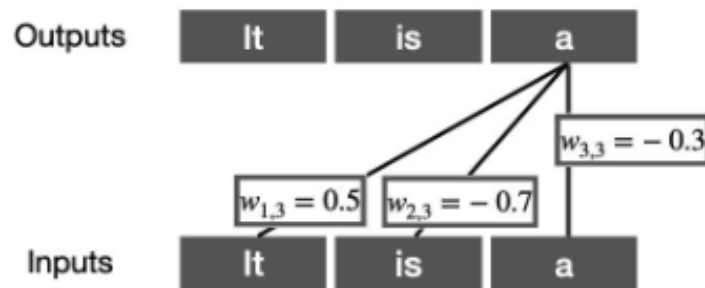
Figure 1: The graphical illustration of the proposed model trying to generate the  $t$ -th target word  $y_t$  given a source sentence  $(x_1, x_2, \dots, x_T)$ .

# Attention Visualized



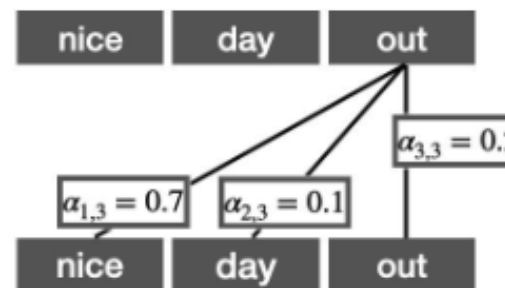
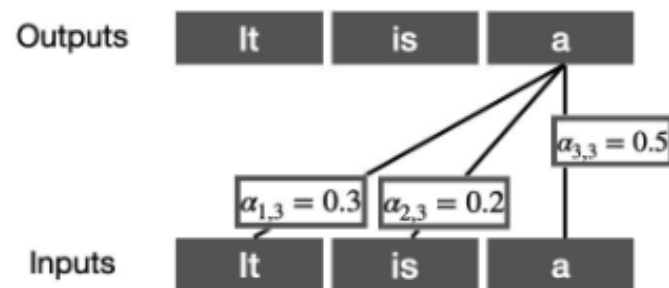
# Attention vs Weights from fully-connected layer?

- Fully-connected layer weights  $W$  are static w.r.t the input



$W$

- Attention scores  $a$  are dynamic w.r.t the input context

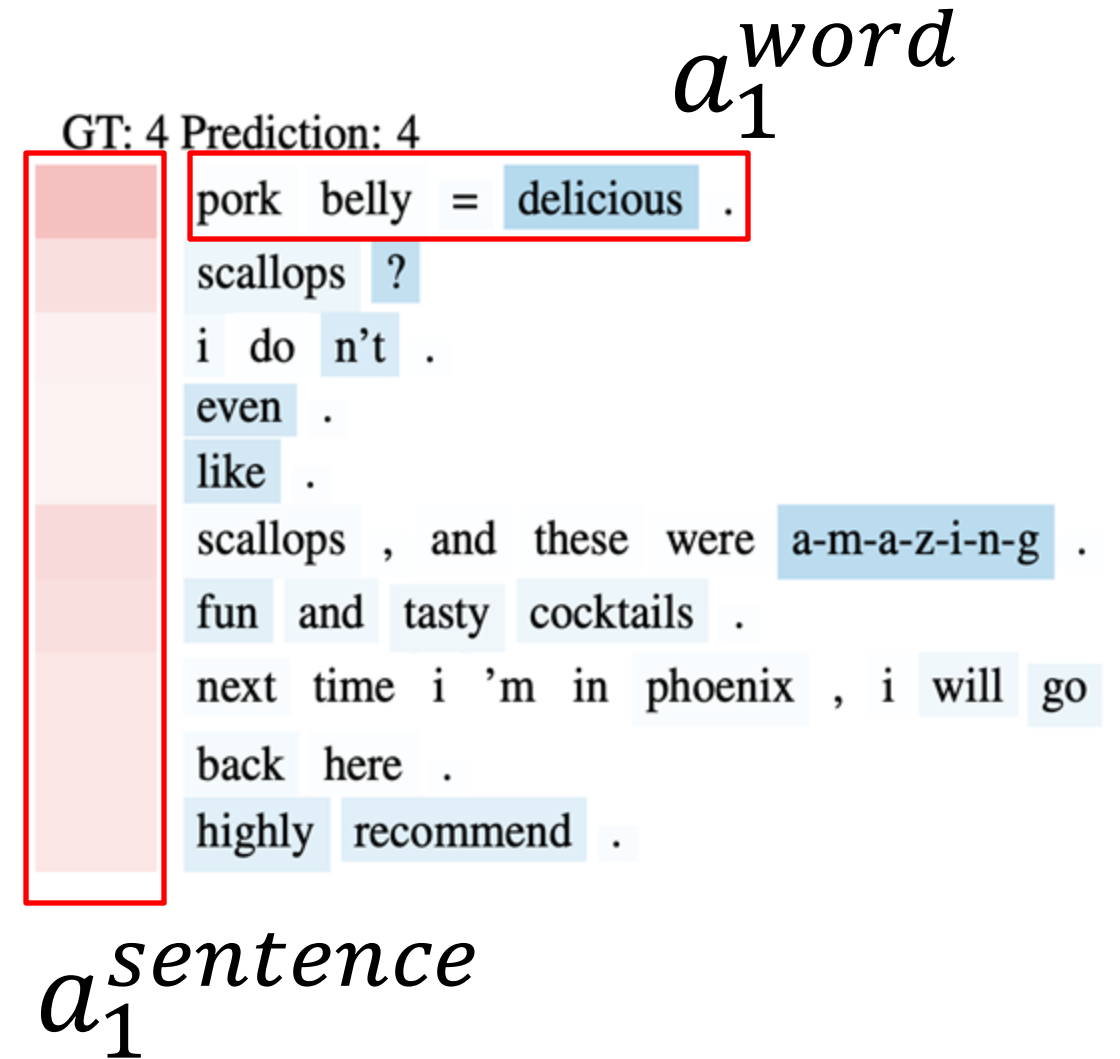
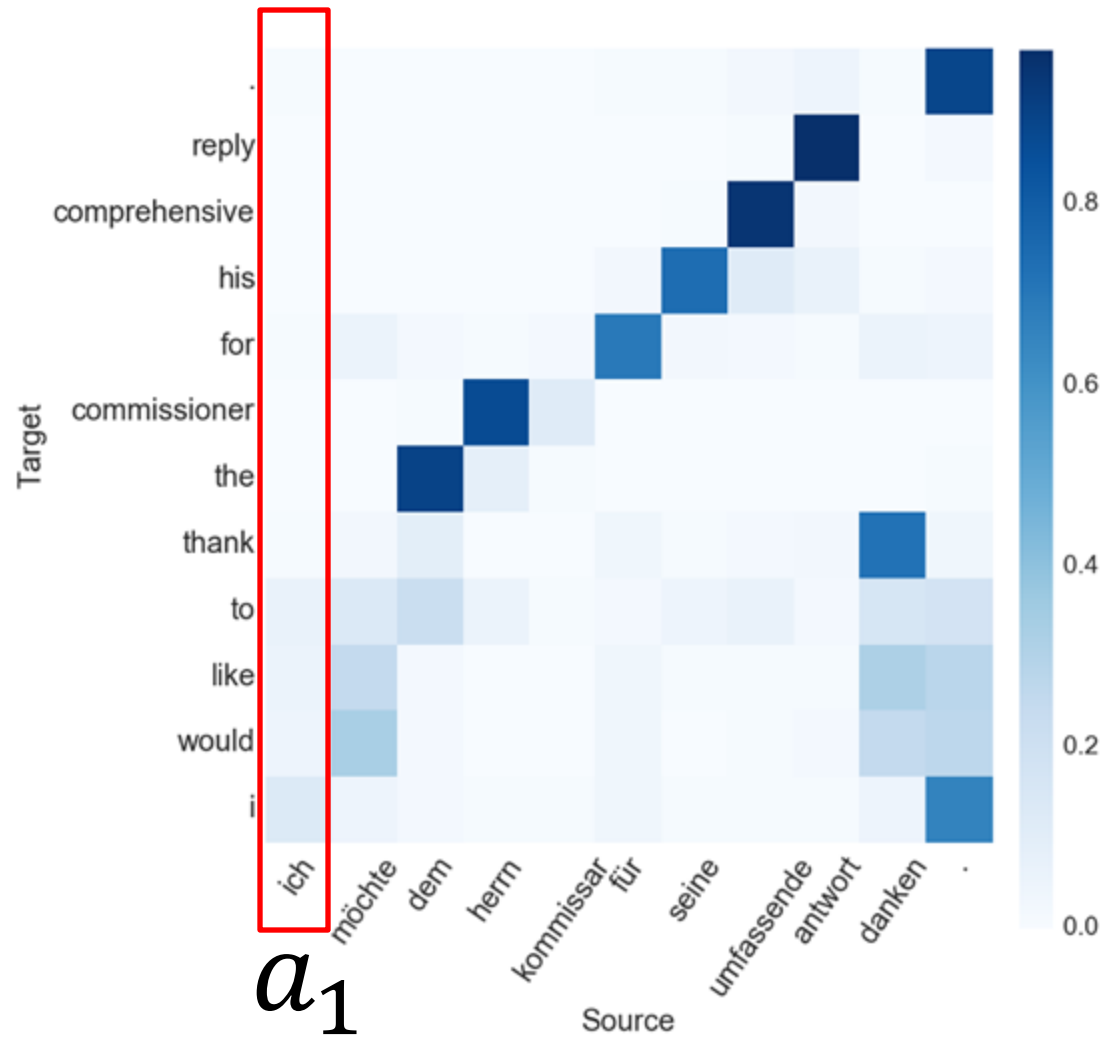


$a$

Examples from Sebastian Raschka



# Attention



# Attention



a man riding a bike down a road next to a body of water.



# Self-Attention (Transformers)



# BERT

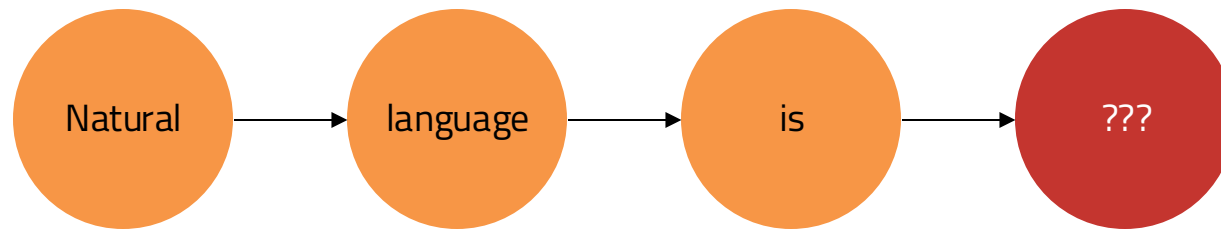


- ❑ **Transformer** or **self-attention** based (Vaswani et al., 2017) masked language model using bidirectional context and next sentence prediction
- ❑ Generates **multiple layers of representations** for each token sensitive to its context use.



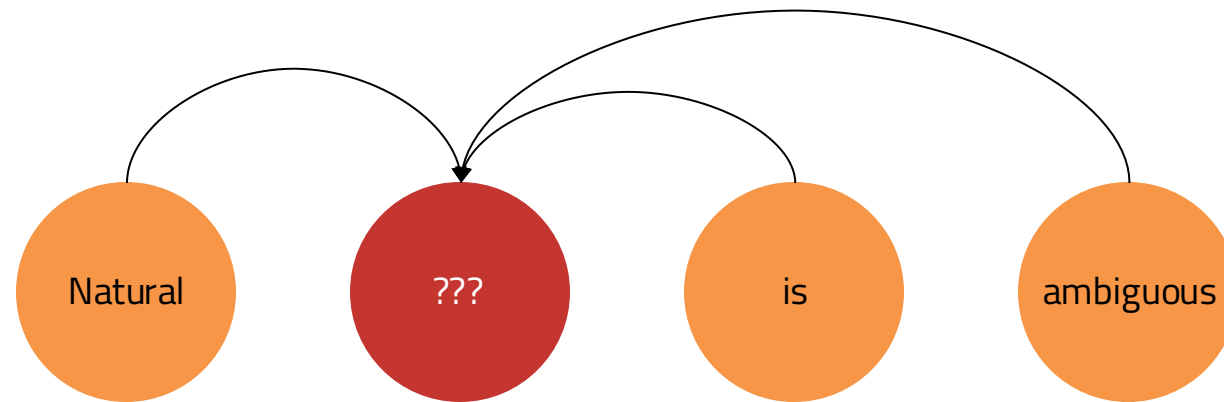
# Classical (causal) language model

Consider only the **left context** to predict **the next word**  
(i.e., the final word in a sequence is masked)



# Masked language model

Use **any context (left or right)** to predict a **masked** word



Each token in input starts represented by **token** and **position** embeddings

1.5	0,5	0.2	0.6
Token 1, Layer 1			

I

53	-1	0.5	2.1
Token 2, Layer 1			

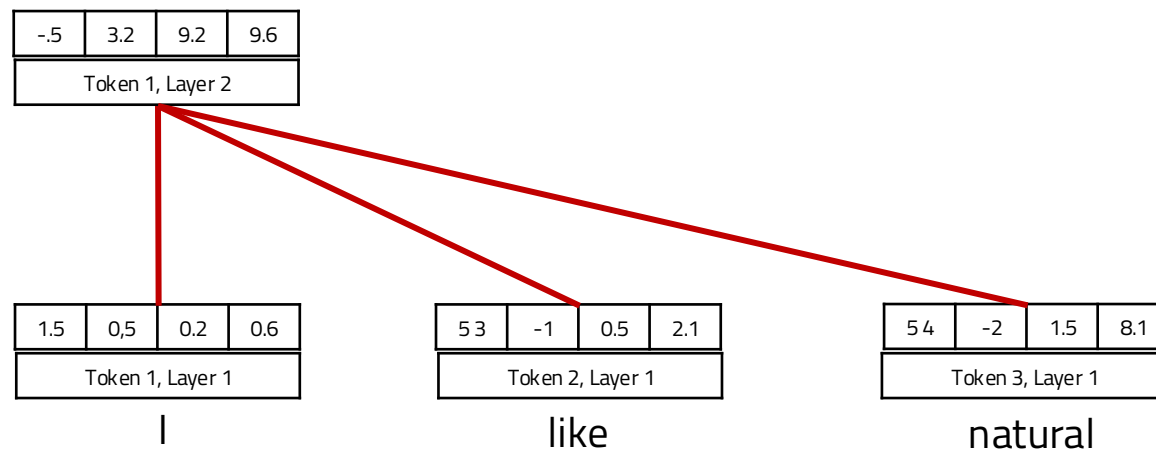
like

54	-2	1.5	8.1
Token 3, Layer 1			

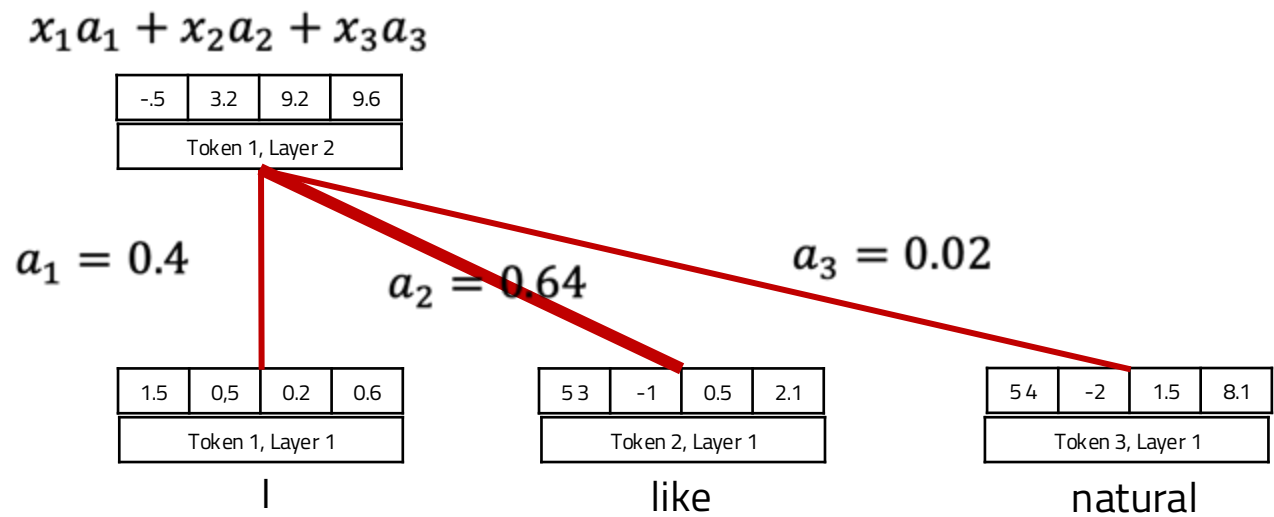
natural

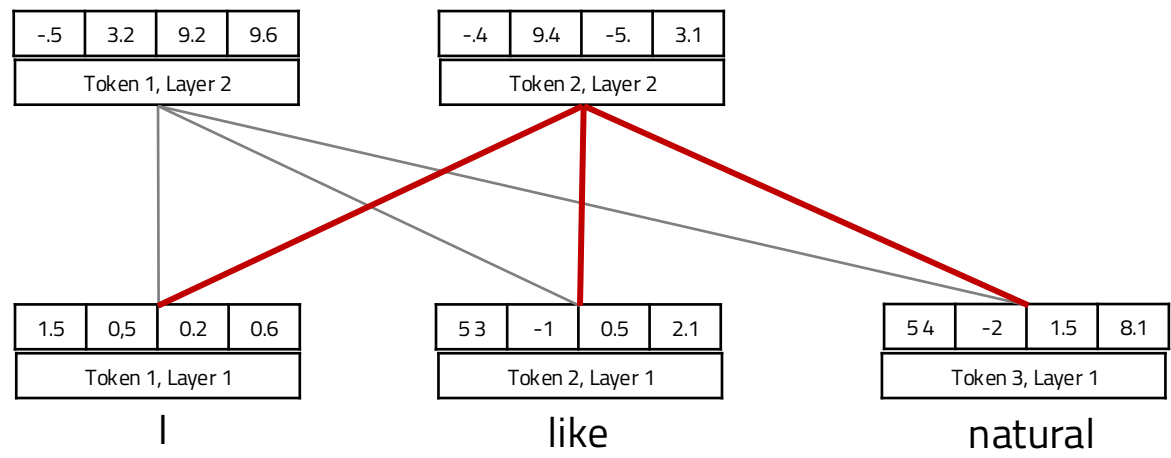


The value for time step  $j$  at layer  $i$  is the result of **attention** over all time steps in the previous layer  $i-1$

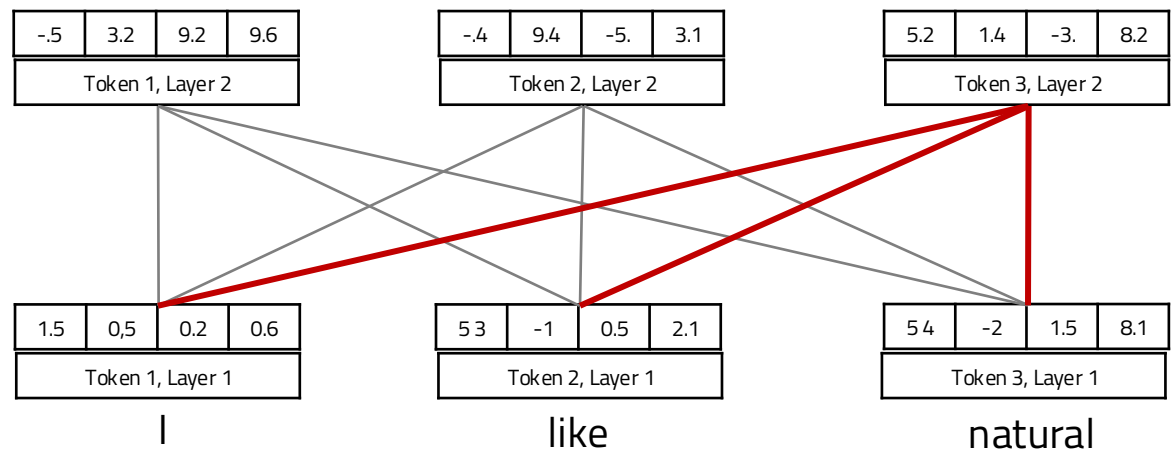


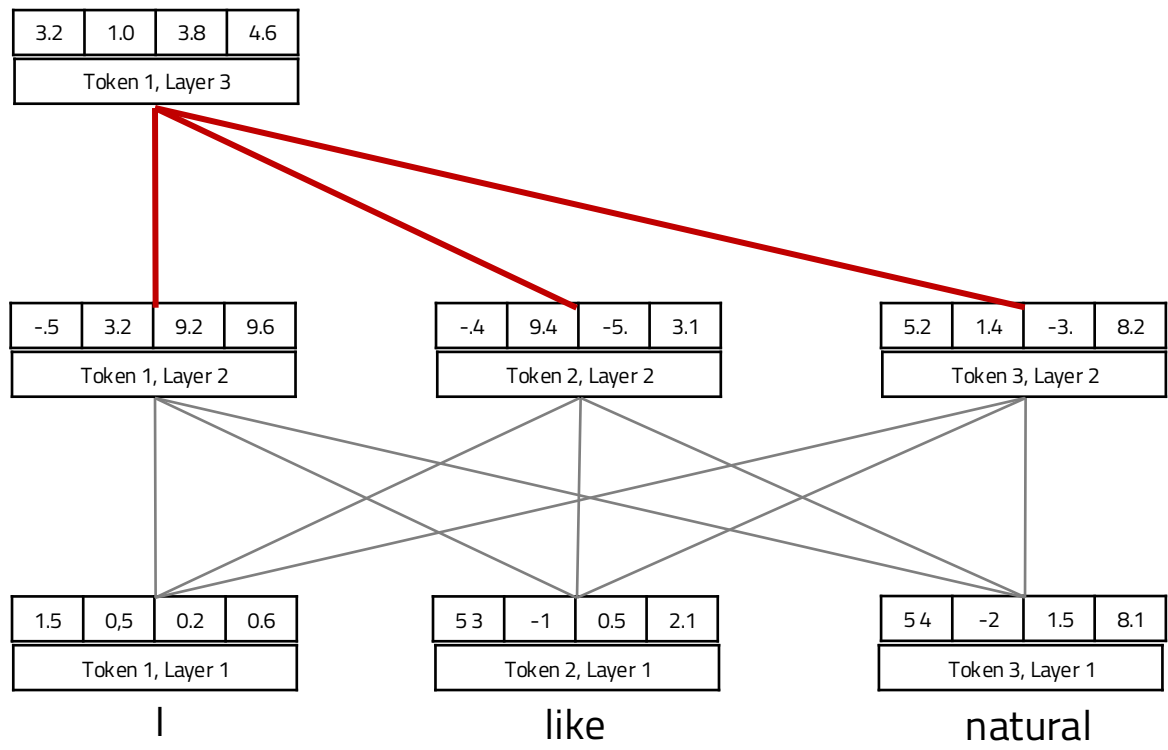
The value for time step  $j$  at layer  $i$  is the result of **attention** over all time steps in the previous layer  $i-1$

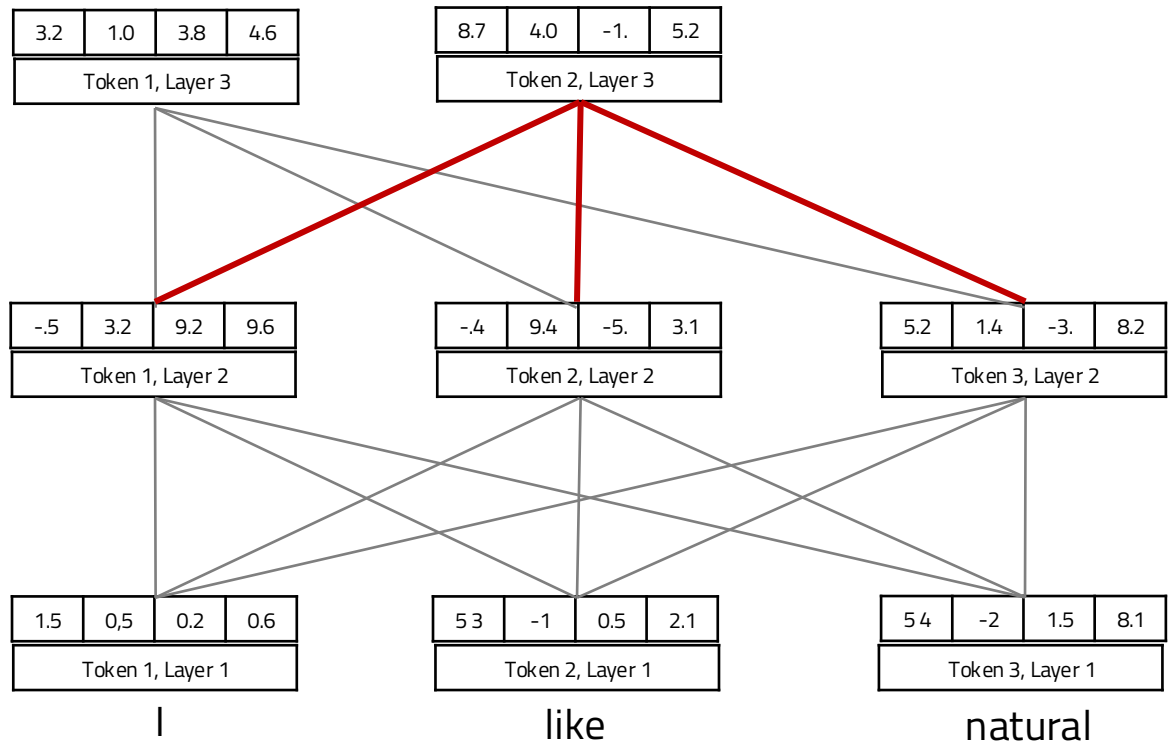


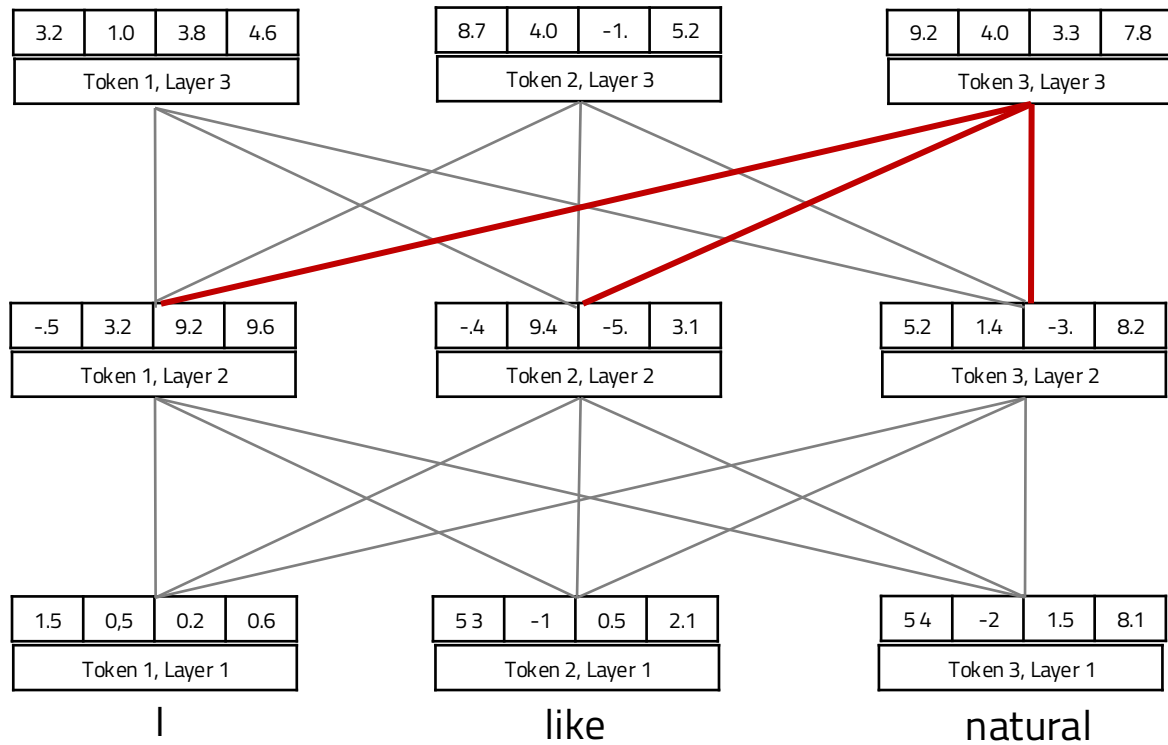






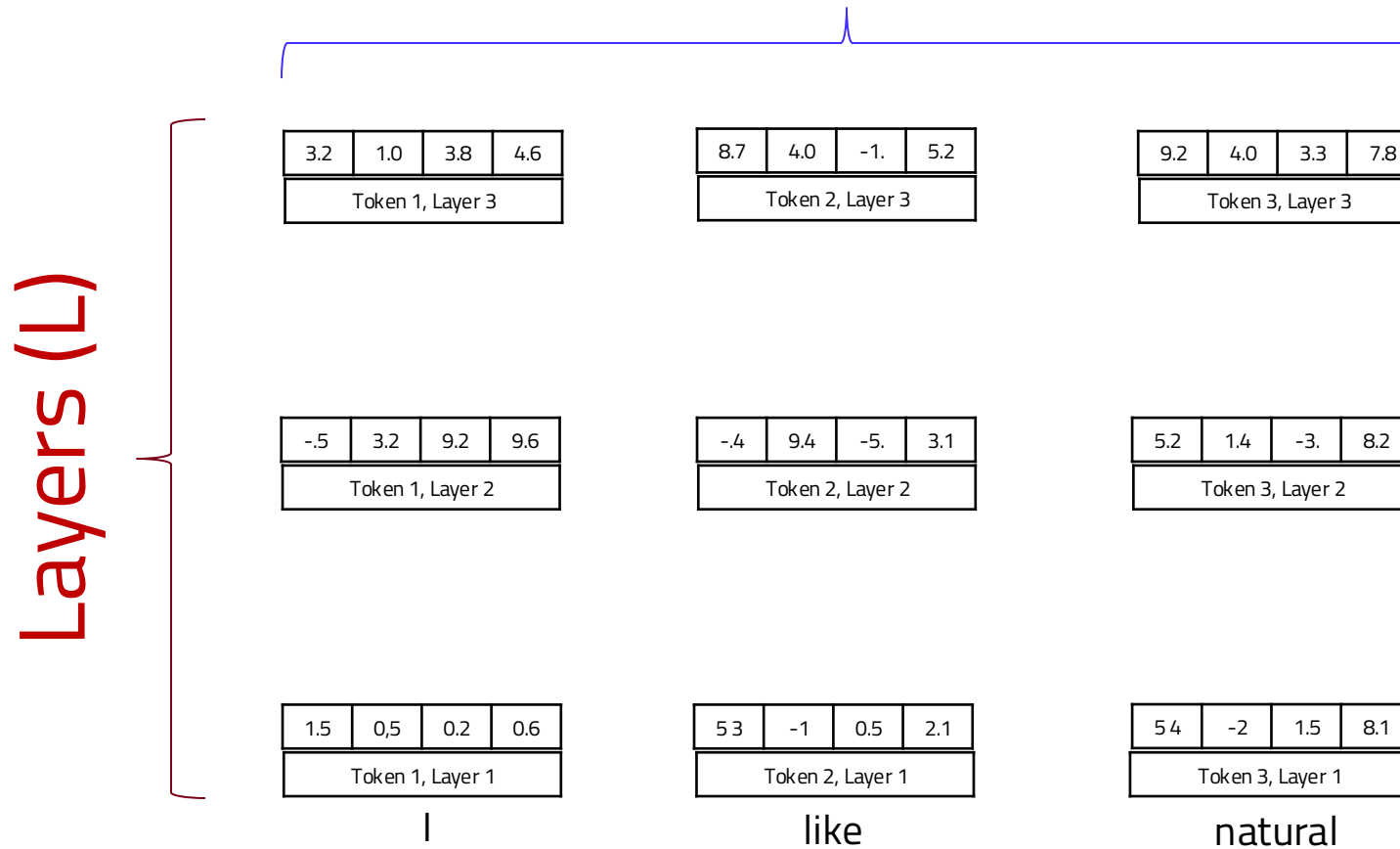






At the end, we have one representation  
for each layer for each token

## Input Length (T)



# Tokenization in BERT

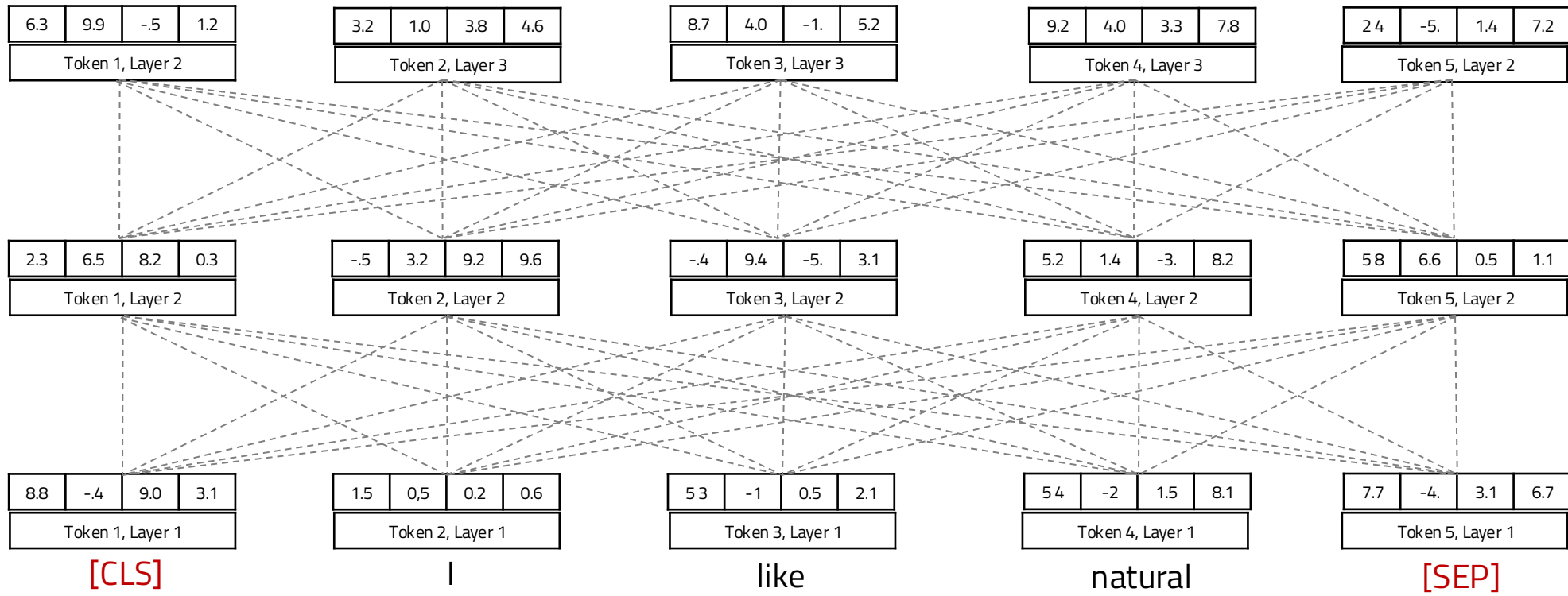
□ BERT uses **WordPiece** tokenization

□ Vocabulary size: 30,000

□ BERT encodes each sentence by appending a special token to the beginning (**[CLS]**) and end (**[SEP]**) of each sequence

The	The
unwilling	un #will #ing
barked	bark #ed



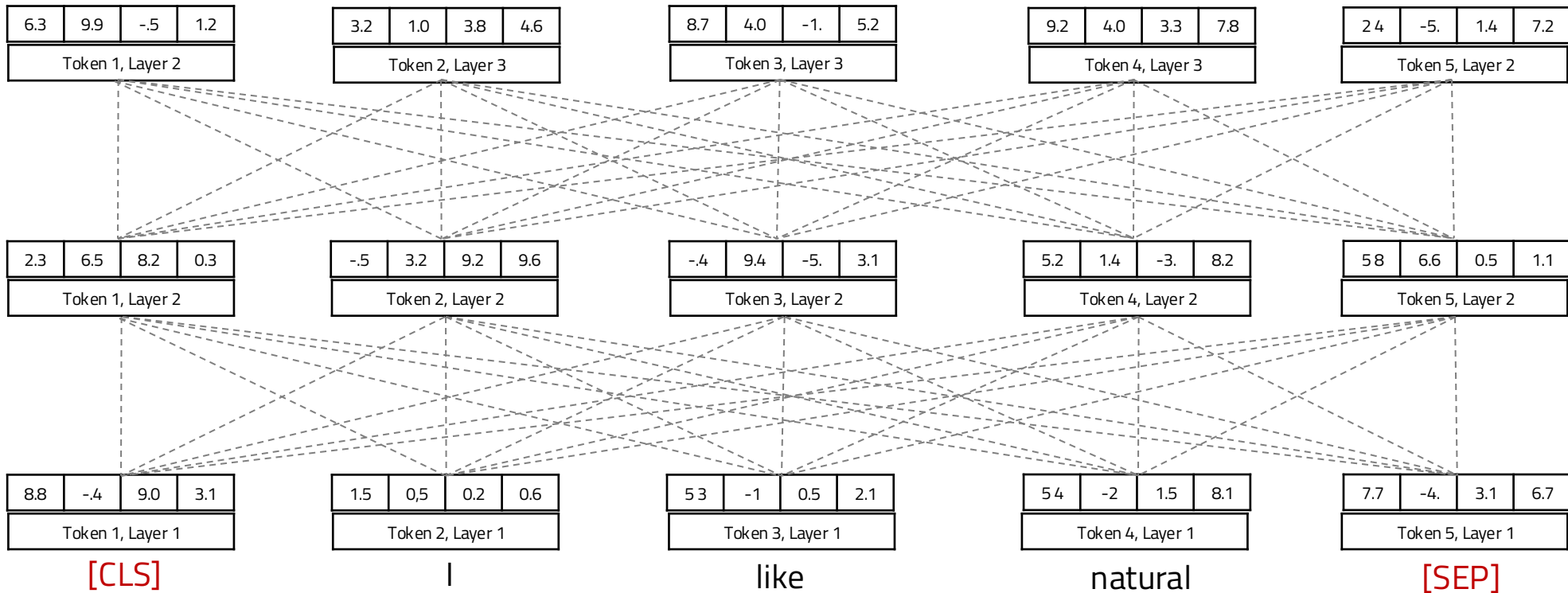


Positive sentiment

Sentiment classifier



Special tokens are helpful for providing a single token that can be optimized to represent the entire sequence (e.g., document classification)



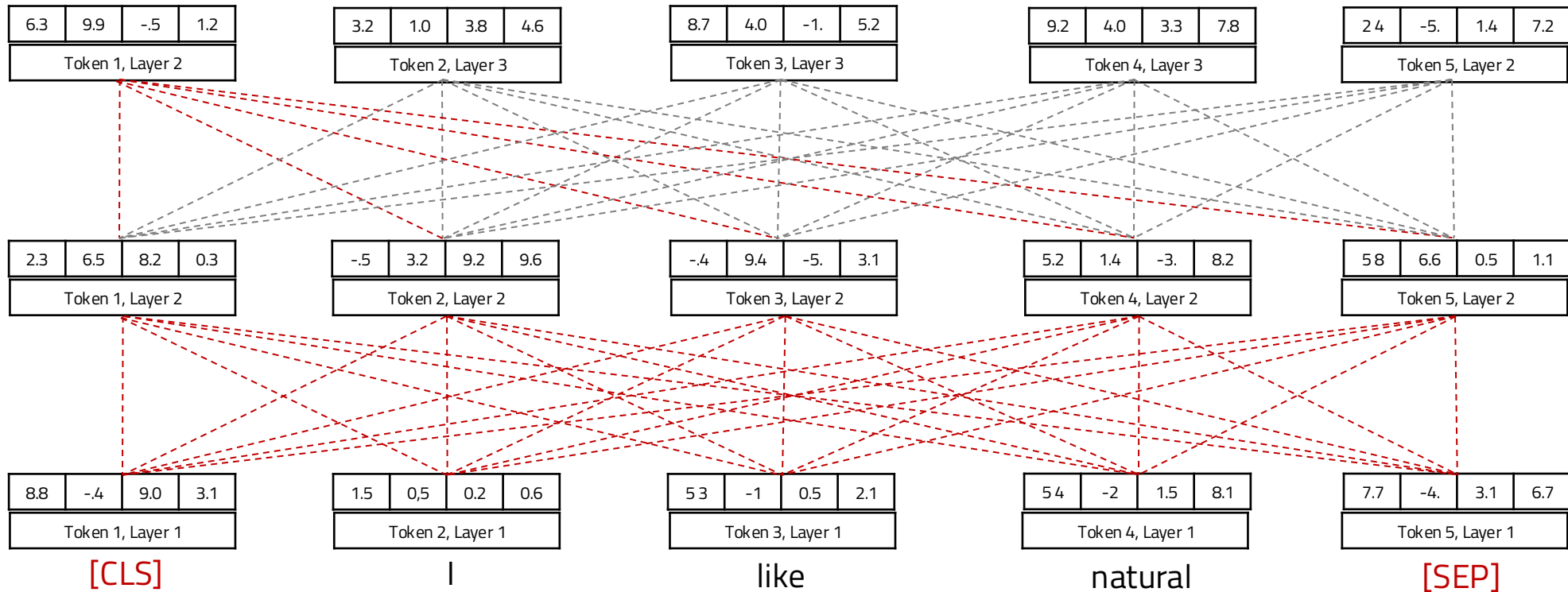


Positive sentiment

Sentiment classifier

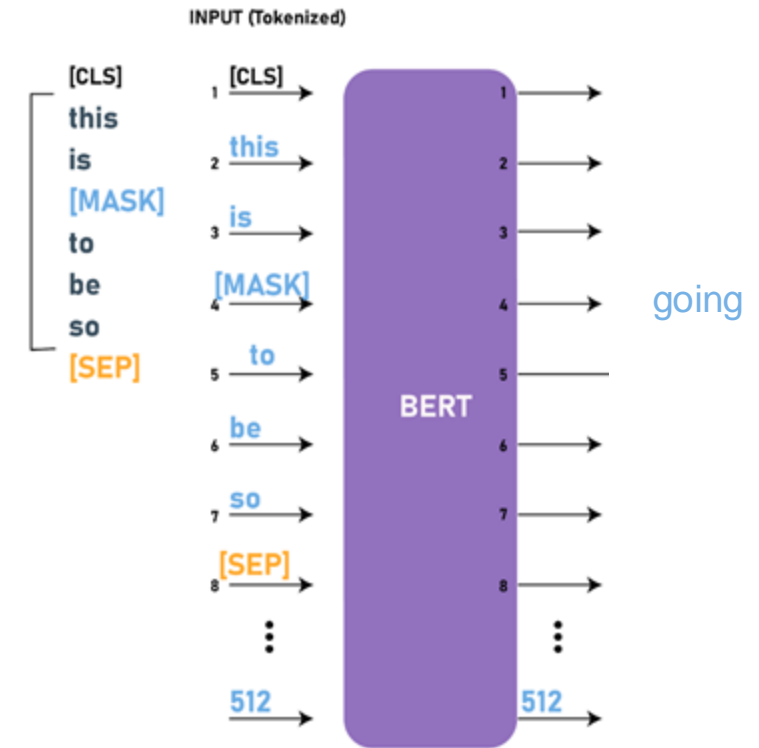


**How can we represent the entire document with this one [CLS] vector?**  
Classification decision relies entirely on that one vector where all the relevant information is compressed into that one vector



# Training BERT

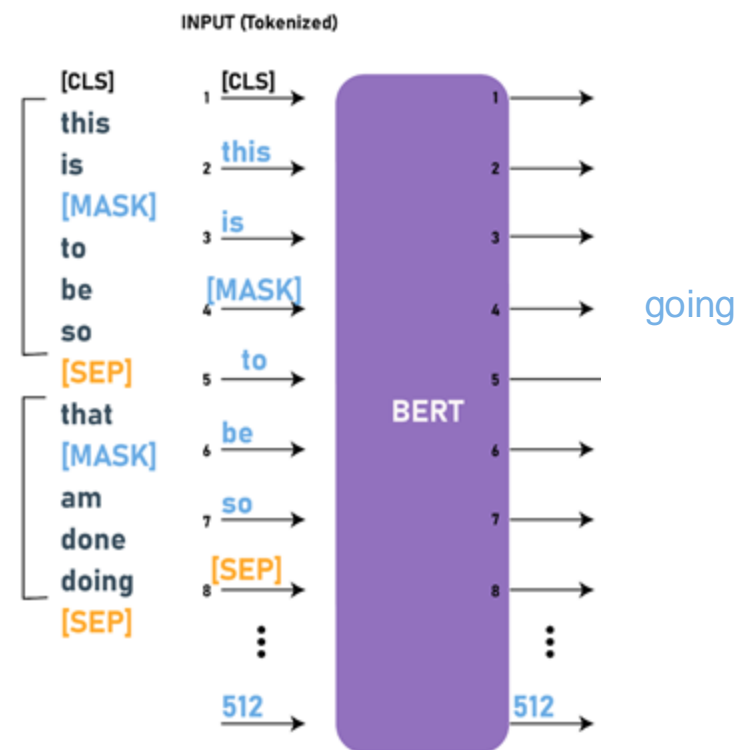
- #1: Masked language modeling
  - [Mask] one word from input and try to predict that word as output
  - Maximum length = 512



Input Length (T)= 512

# Training BERT

- #1: Masked language modeling
  - [Mask] one word from input and try to predict that word as output
  - Maximum length = 512
  - Concatenate two sentences with [SEP] token
  - More powerful than Bidirectional-RNN LM



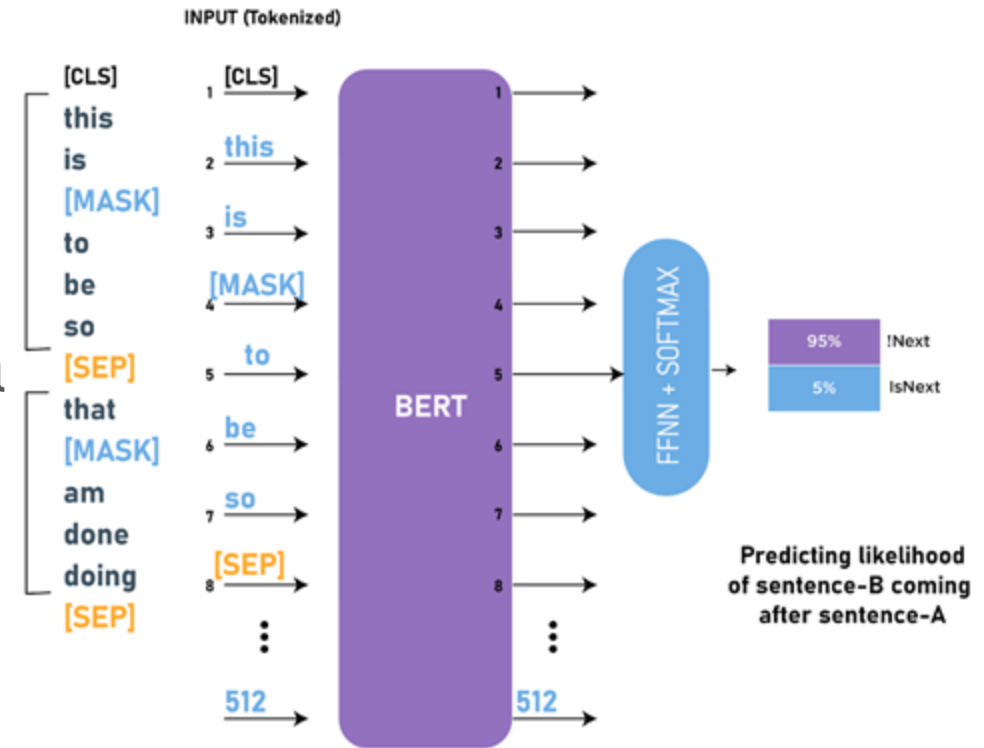
# Training BERT

## □ #2: Next sentence prediction

- For a pair of sentences, predict from [CLS] representation whether they appeared **sequentially** in the training data

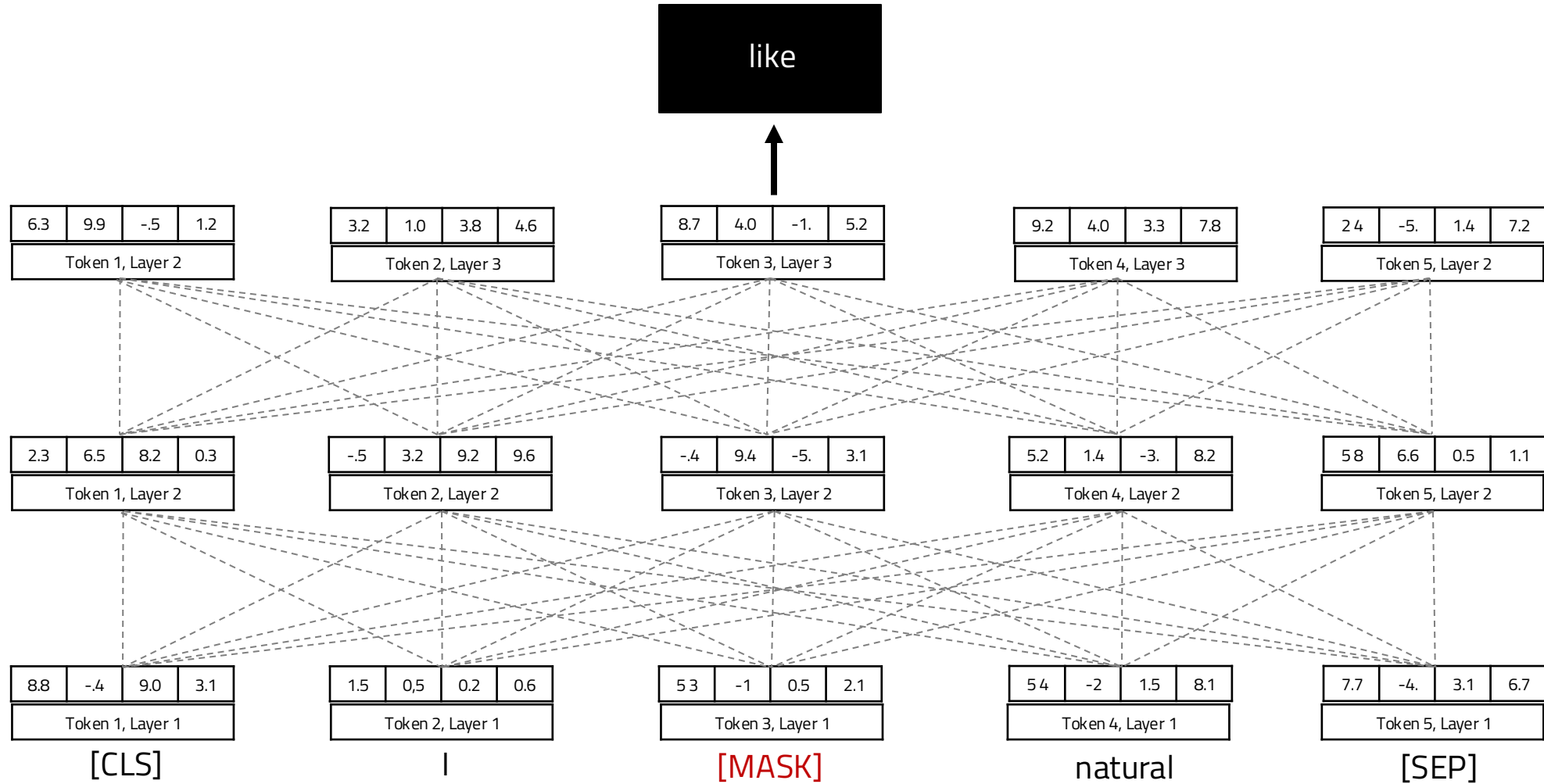
Next=True [CLS] I like natural language processing [SEP] because NLP is fun  
Next=False [CLS] I like natural language processing [SEP] Minnesota is cold.

- This objective turns out to be not that effective, found in RoBERTa paper (Liu et al., 2019)



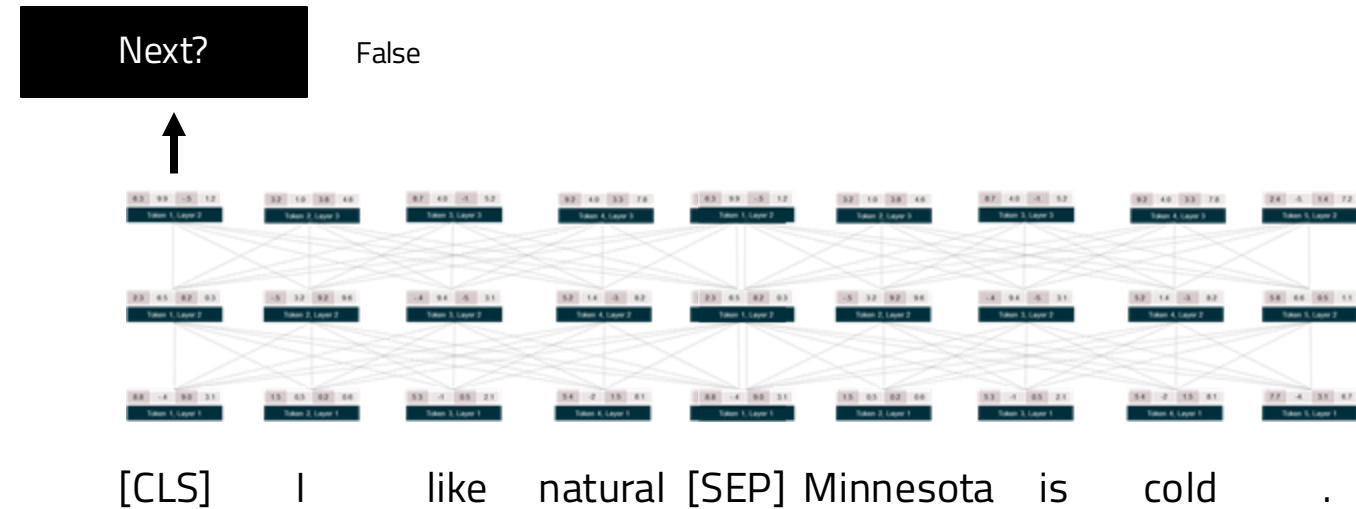
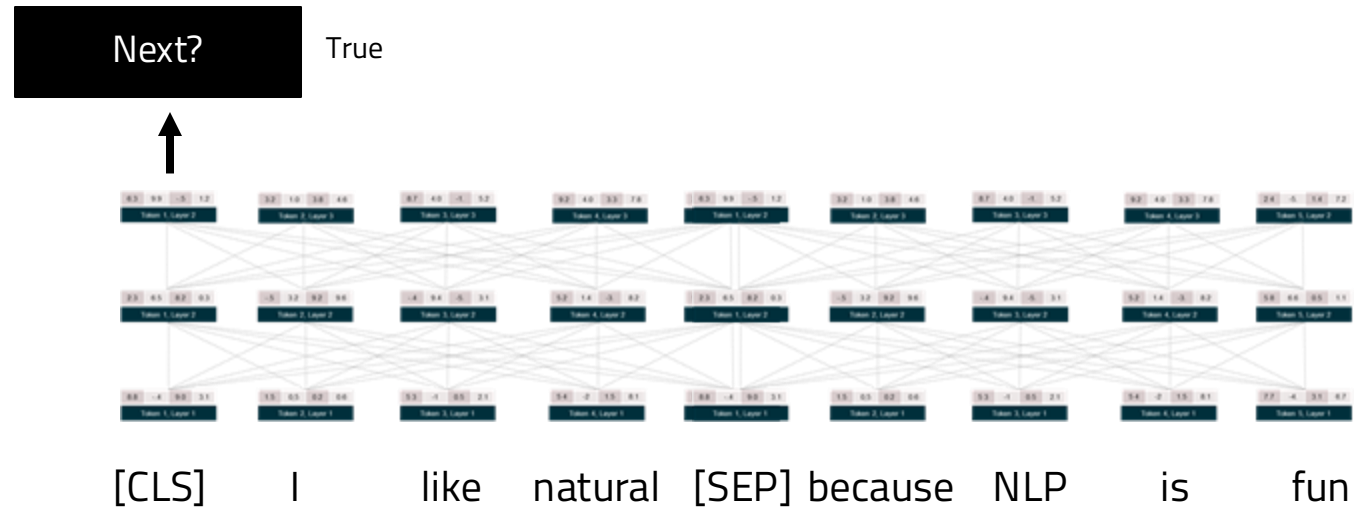
Objective #1: Masked language modeling

# $L_{MLM}$



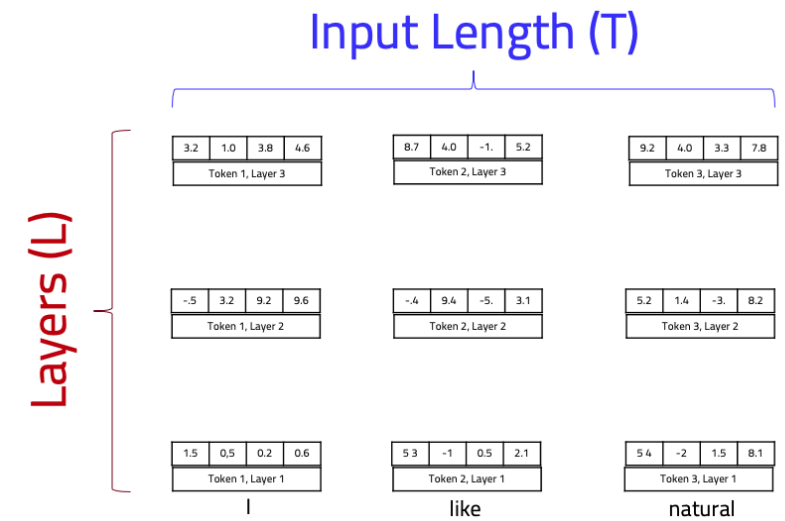
Objective #2: Next sentence prediction

$$L_{MLM} + L_{NSP}$$



# Details of BERT training

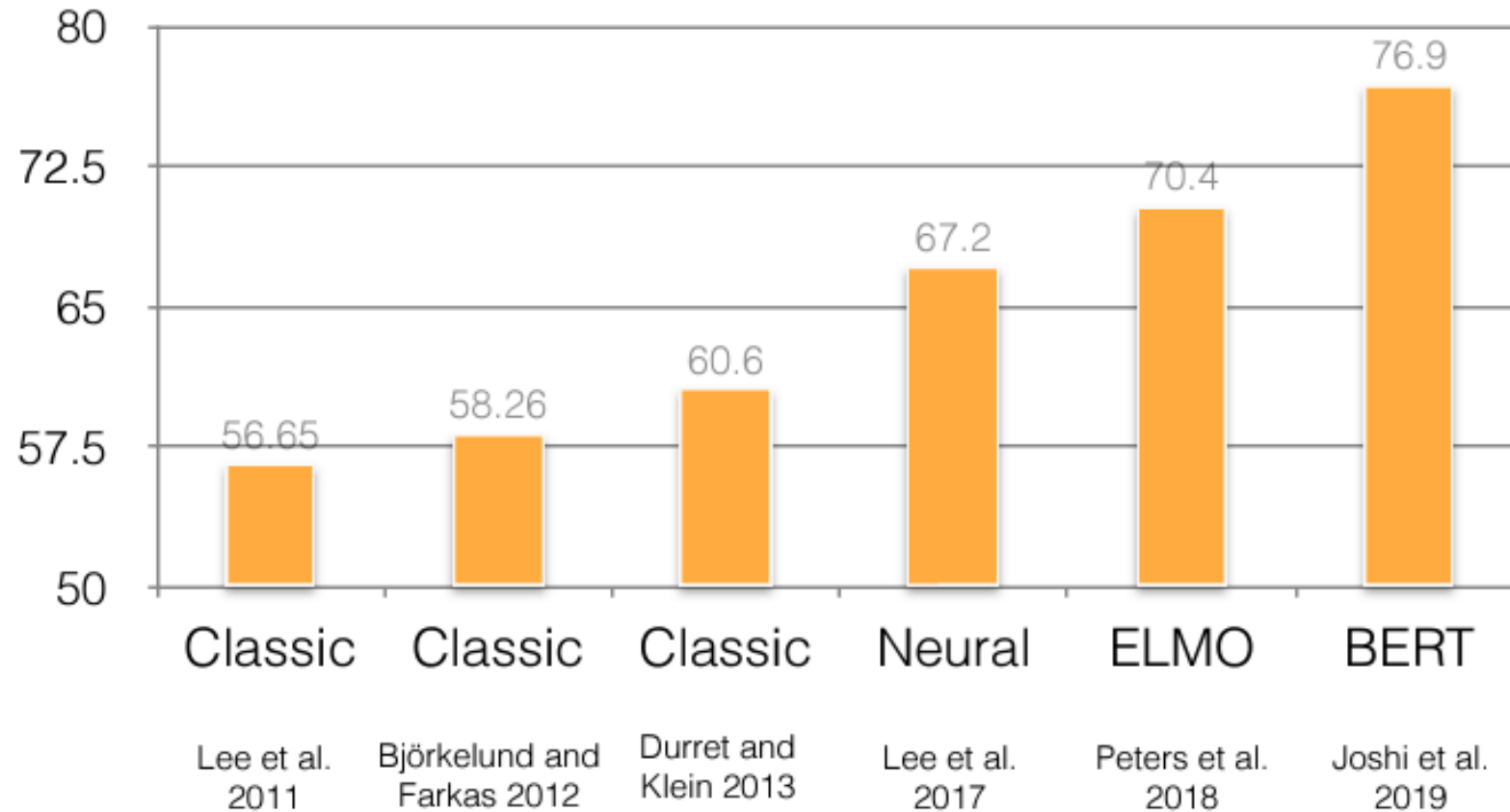
- ❑ Deep layers
  - 12 layers for BERT-base
  - 24 layers for BERT-large
- ❑ Large representation size (768 per layer)
- ❑ Pretrained on English Wikipedia (2.5B words) and BookCorpus (800M words)



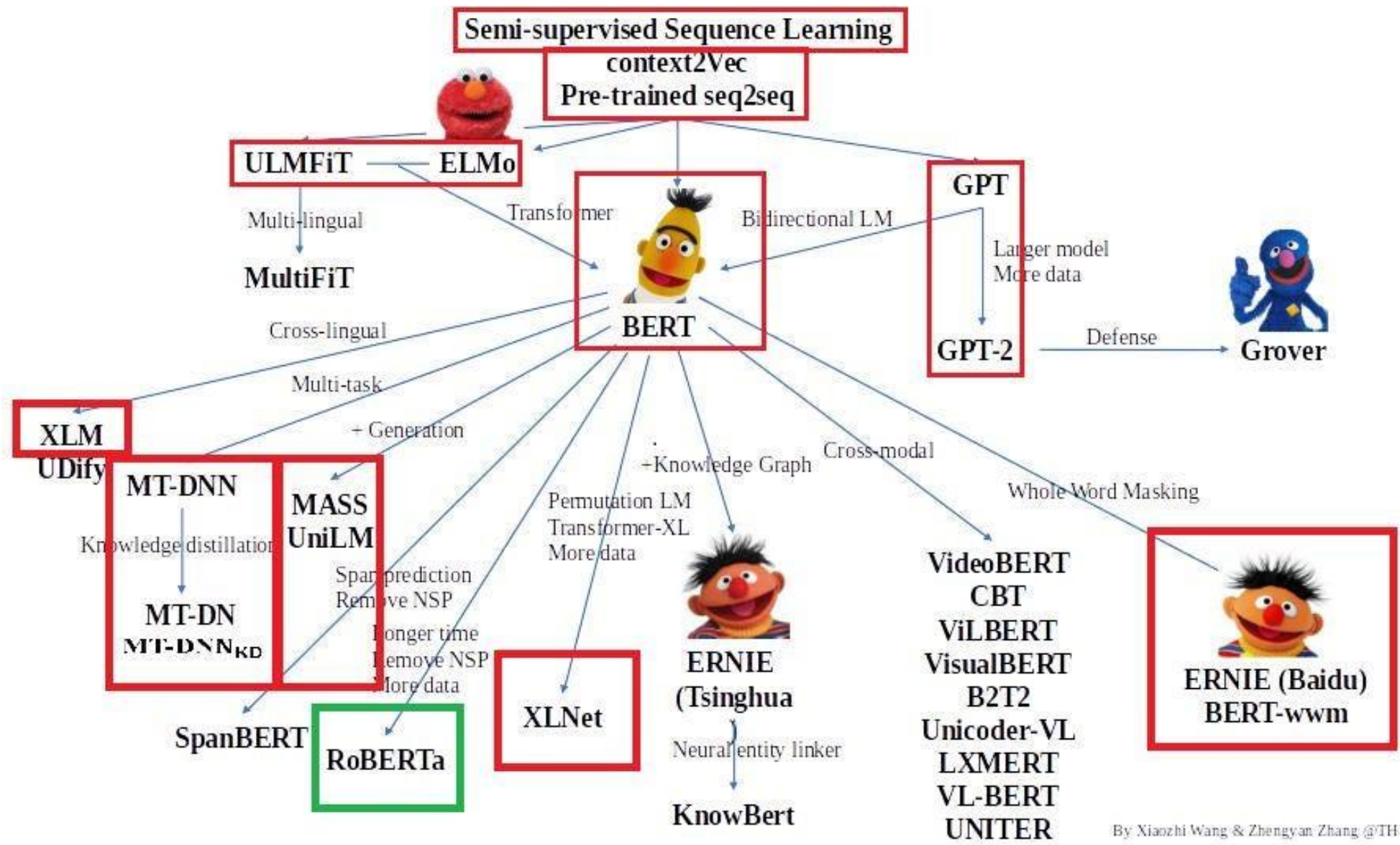
$$L_{MLM} + L_{NSP}$$



# Coreference resolution with BERT





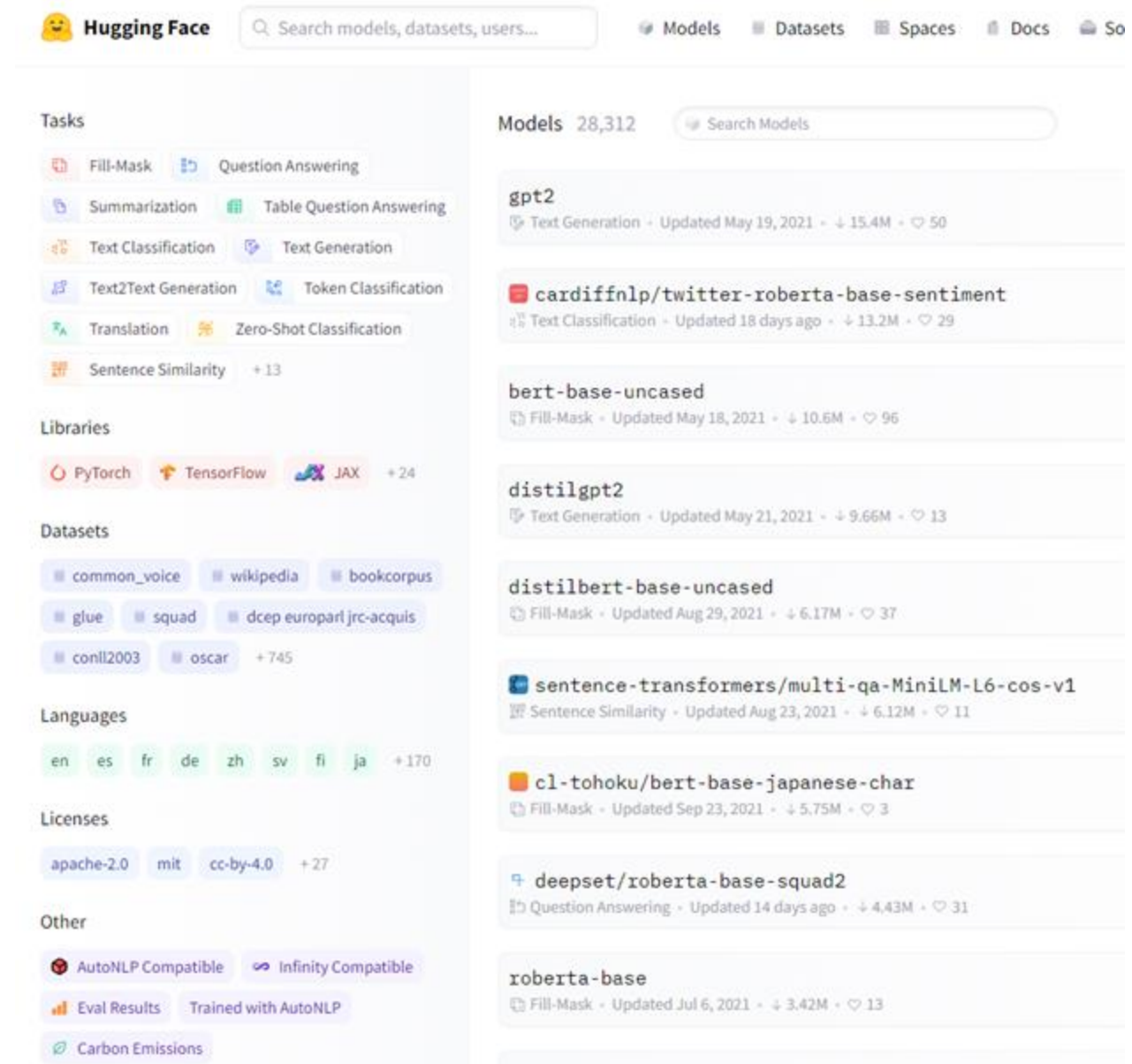


By Xiaozhi Wang & Zhengyan Zhang @THUNLP



# Other pretrained LMs

- BERT
- XLNet
- ALBERT
- RoBERTa
- DistilBERT
- GPT-2/3
- Multilingual-BERT



The screenshot shows the Hugging Face website interface. At the top, there is a search bar and navigation links for Models, Datasets, Spaces, Docs, and So. The main content is divided into several sections:

- Tasks:** A grid of task categories including Fill-Mask, Question Answering, Summarization, Table Question Answering, Text Classification, Text Generation, Text2Text Generation, Token Classification, Translation, Zero-Shot Classification, and Sentence Similarity.
- Libraries:** A row of library icons for PyTorch, TensorFlow, and JAX.
- Datasets:** A grid of dataset categories such as common\_voice, wikipedia, bookcorpus, glue, squad, dcep europarl jrc-acquis, conll2003, and oscar.
- Languages:** A row of language codes including en, es, fr, de, zh, sv, fi, ja.
- Licenses:** A row of license categories like apache-2.0, mit, and cc-by-4.0.
- Other:** A row of other categories including AutoNLP Compatible, Infinity Compatible, Eval Results, Trained with AutoNLP, and Carbon Emissions.

On the right side, there is a 'Models' section with a search bar and a list of model cards. The visible model cards include:

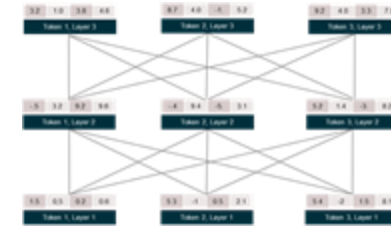
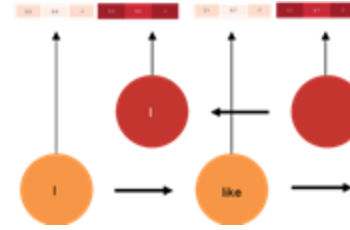
- gpt2:** Text Generation - Updated May 19, 2021 - 15.4M likes
- cardiffnlp/twitter-roberta-base-sentiment:** Text Classification - Updated 18 days ago - 13.2M likes
- bert-base-uncased:** Fill-Mask - Updated May 18, 2021 - 10.6M likes
- distilgpt2:** Text Generation - Updated May 21, 2021 - 9.66M likes
- distilbert-base-uncased:** Fill-Mask - Updated Aug 29, 2021 - 6.17M likes
- sentence-transformers/multi-qa-MiniLM-L6-cos-v1:** Sentence Similarity - Updated Aug 23, 2021 - 6.12M likes
- cl-tohoku/bert-base-japanese-char:** Fill-Mask - Updated Sep 23, 2021 - 5.75M likes
- deepset/roberta-base-squad2:** Question Answering - Updated 14 days ago - 4.43M likes
- roberta-base:** Fill-Mask - Updated Jul 6, 2021 - 3.42M likes

<https://huggingface.co/models>



# Summary

woman	king
5.2	1.5
0.5	0.4
-6.2	0.6



- ❑ Word embeddings can be **substituted for one-hot** encodings in many models (MLP, CNN, RNN, logistic regression).
- ❑ Bidirectional modeling in ELMo/BERT helps learn more **context sensitive information**.
- ❑ Attention gives us a mechanism to learn which parts of a sequence to **pay attention** to more in forming a representation of it.
- ❑ Static word embeddings (word2vec, Glove) provide representations of word **types**; contextualized word representations (ELMo, BERT) provide representations of **tokens** in context.

