



CSCE 670 - Information Storage and Retrieval

Lecture 17: Contextualized Language Models, BERT

Yu Zhang

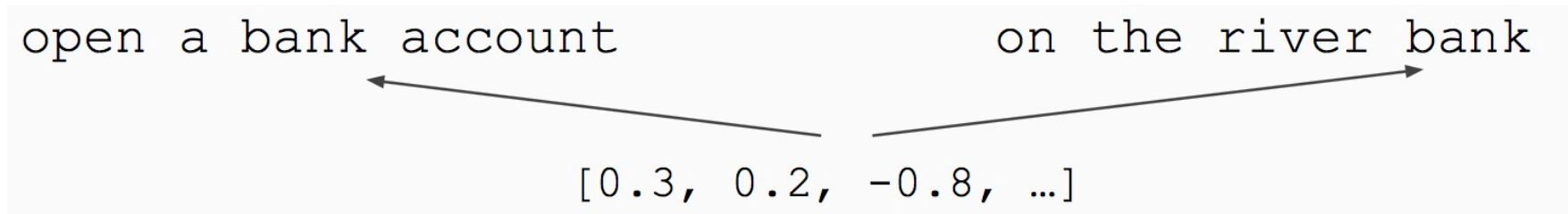
yuzhang@tamu.edu

October 23, 2025

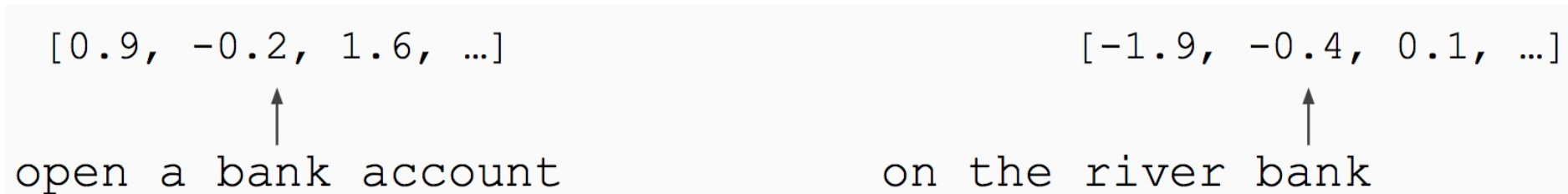
Course Website: <https://yuzhang-teaching.github.io/CSCE670-F25.html>

Recap: Limitations of word2vec-based Ranking

- Word embeddings are applied in a context-free manner!



- Is it possible to have different vectors of the same **word** given different **contexts**?



Transformer [Vaswani et al., NIPS 2017]

Attention Is All You Need

Attention is all you need

[A Vaswani, N Shazeer, N Parmar...](#) - Advances in neural ...,
2017 - proceedings.neurips.cc

... to attend to **all** positions in the decoder up to and including

We need to prevent ... **We** implement this inside of scaled dot
attention by masking out (setting to $-\infty$) ...

☆ Cited by 199331 Related articles ↗

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez*[†]

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaizer@google.com

Illia Polosukhin*[‡]

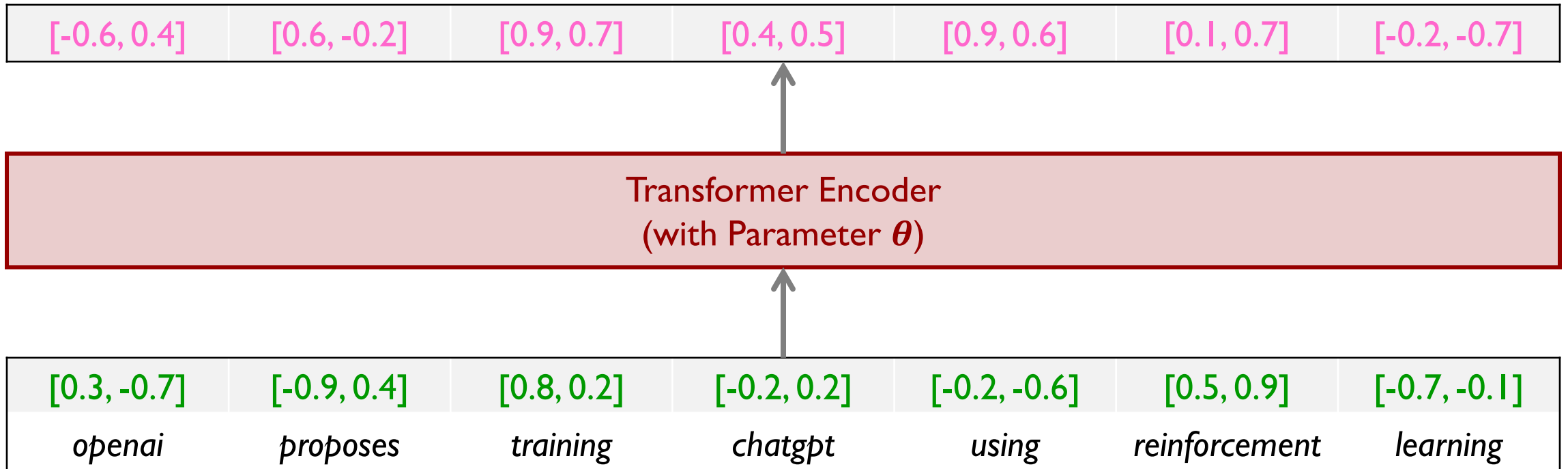
illia.polosukhin@gmail.com

Transformer as a Black Box

- Transformer is a **neural network**
- It has two types of architecture: **encoder** and **decoder**
- In this lecture, we focus on the **Transformer encoder**
- Input to a **Transformer encoder** can be a piece of text:
 - A sequence of words w_1, w_2, \dots, w_L
 - Represented by their corresponding embeddings $e_{w_1}, e_{w_2}, \dots, e_{w_L}$
- Then, the output is a sequence of contextualized word vectors $h_{w_1}, h_{w_2}, \dots, h_{w_L}$
 - The output vector h_{w_i} captures the meaning of w_i by considering the entire input sequence as w_i 's context
 - $h_{w_i} = \text{Transformer}(e_{w_i} | e_{w_1}, e_{w_2}, \dots, e_{w_L})$

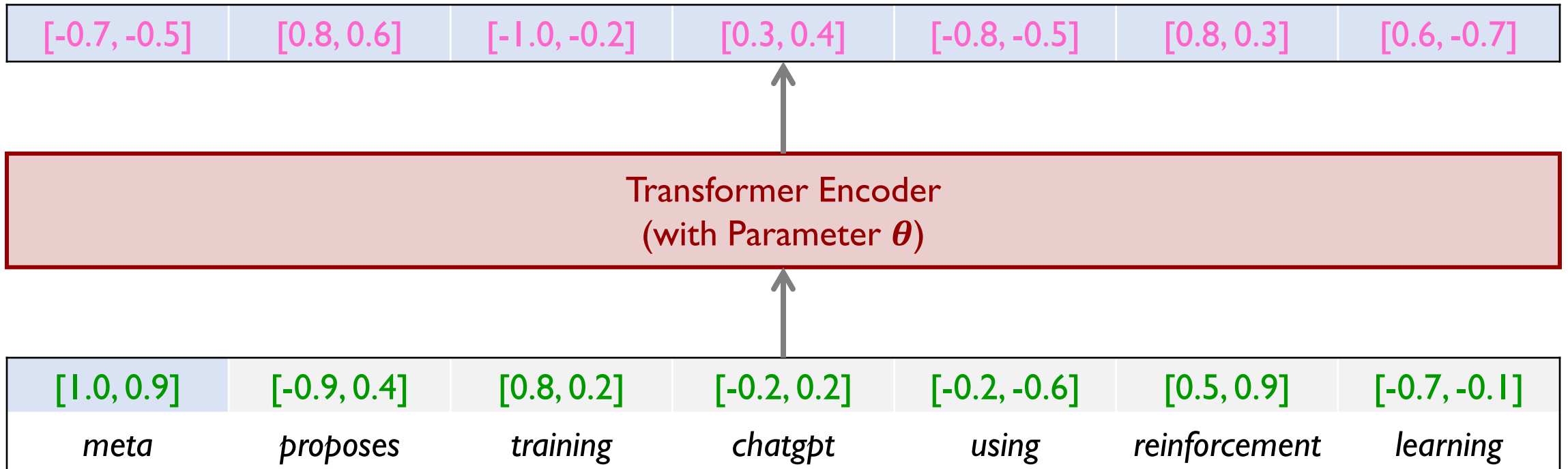
Transformer as a Black Box

- $h_{w_i} = \text{Transformer}(\mathbf{e}_{w_i} | \mathbf{e}_{w_1}, \mathbf{e}_{w_2}, \dots, \mathbf{e}_{w_L})$



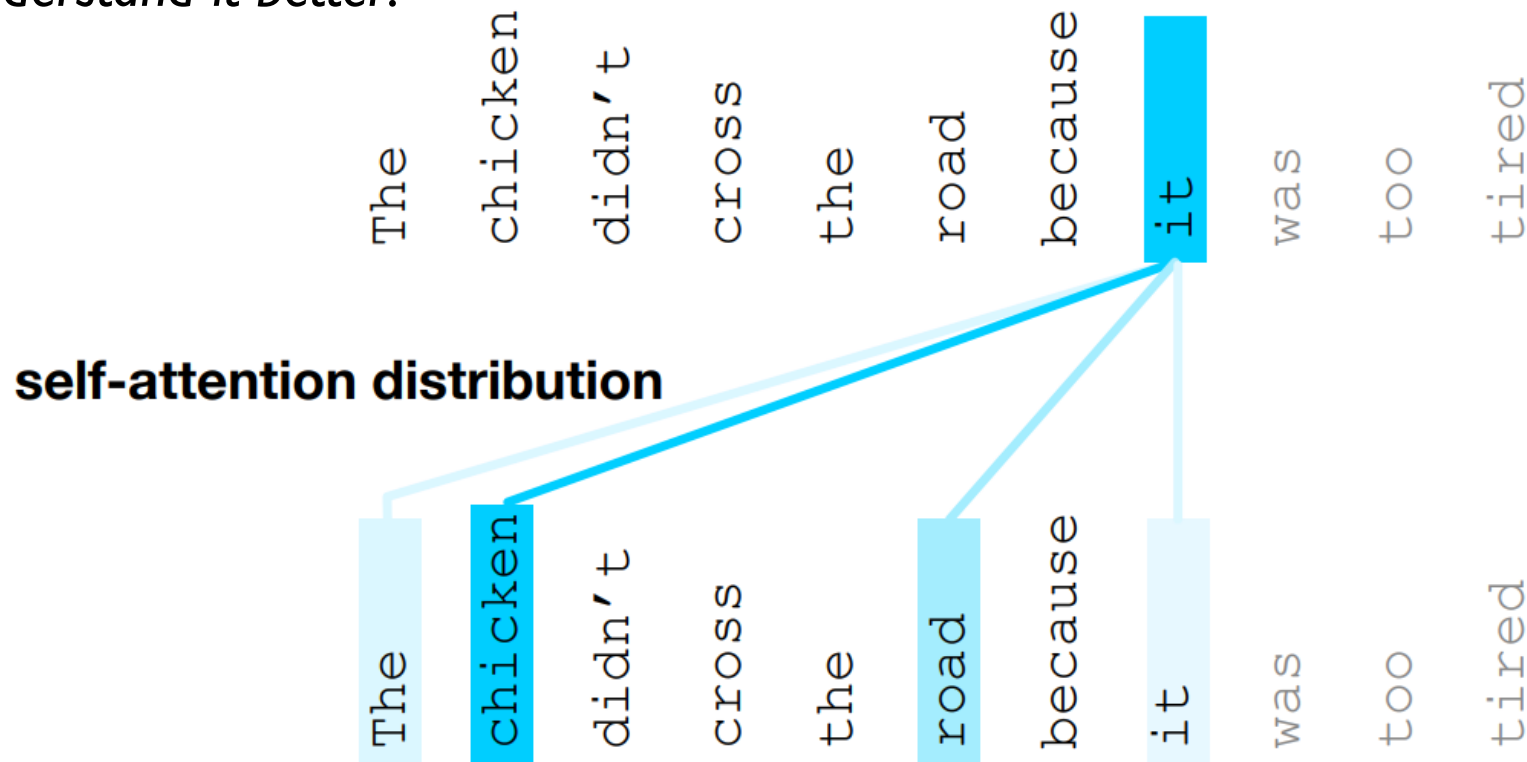
Transformer as a Black Box

- $h_{w_i} = \text{Transformer}(\mathbf{e}_{w_i} | \mathbf{e}_{w_1}, \mathbf{e}_{w_2}, \dots, \mathbf{e}_{w_L})$



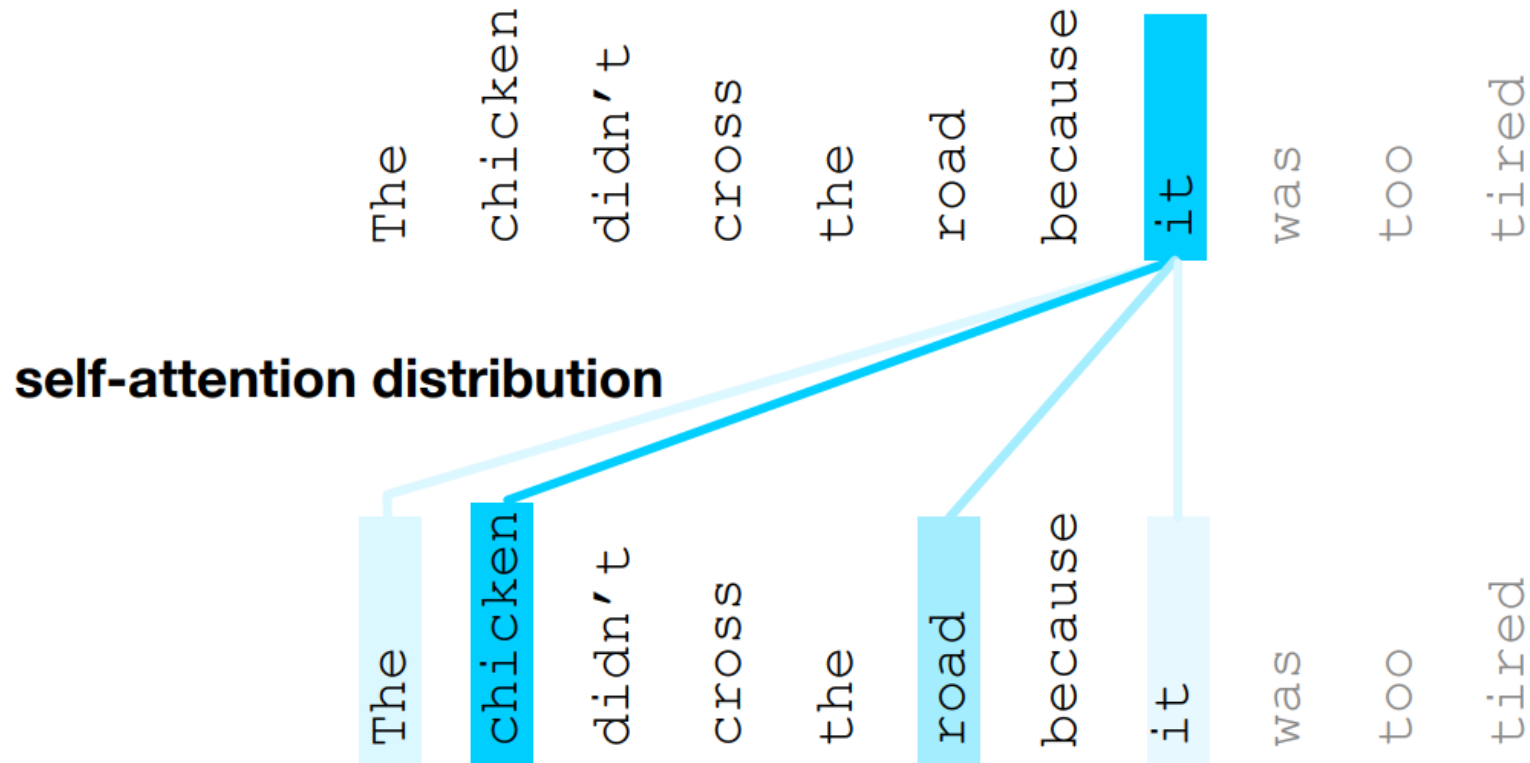
How does the model transform the input into the output?

- “(Self-)Attention”: weigh the importance of different words in a sequence when processing a specific word
 - *When I am looking at this word, which other words should I pay attention to in order to understand it better?*



Self-Attention: A Simplified Example

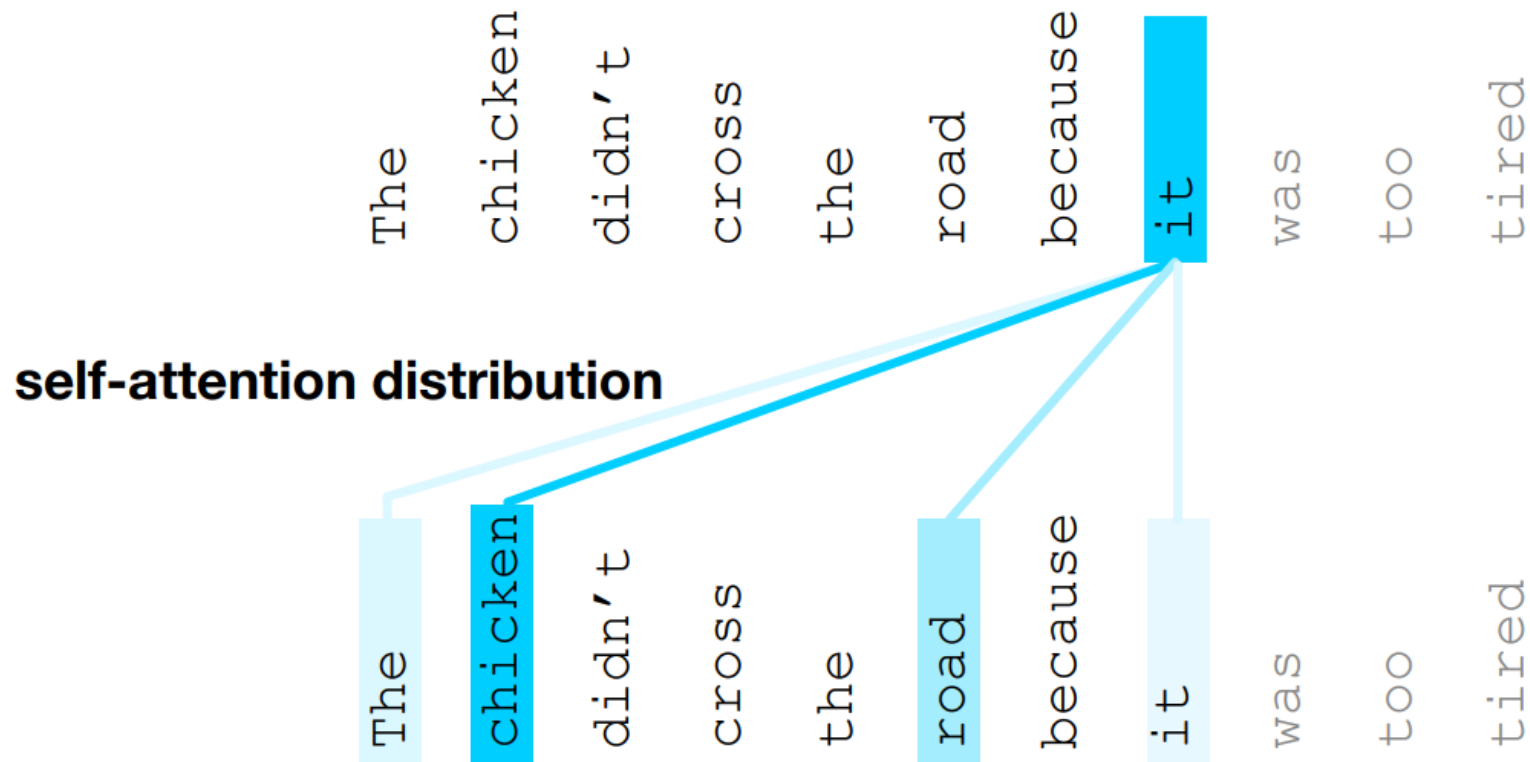
- $\mathbf{h}_{w_i} = \sum_{j=1}^L a_{ij} \mathbf{e}_{w_j}$
- a_{ij} : attention score, where $\sum_{j=1}^L a_{ij} = 1$



Self-Attention: A Simplified Example

- $h_{w_i} = \sum_{j=1}^L \text{Softmax}(\mathbf{e}_{w_i}^T \mathbf{e}_{w_j}) \cdot \mathbf{e}_{w_j}$
- $\text{Softmax}(\mathbf{e}_{w_i}^T \mathbf{e}_{w_j}) = \frac{\exp(\mathbf{e}_{w_i}^T \mathbf{e}_{w_j})}{\sum_{k=1}^L \exp(\mathbf{e}_{w_i}^T \mathbf{e}_{w_k})}$

The weight is determined by the similarity between the context word and the center word.



Self-Attention: Query, Key, Value

- Each word in self-attention is represented by three different vectors
- **Query (Q)**: Represent the current word for which information is being sought
- **Key (K)**: Represent the reference (context) used for comparison with the query
- **Value (V)**: Represent the actual content of each token, which will be aggregated into the final output

$$h_{w_i} = \sum_{j=1}^L \text{Softmax}(e_{w_i}^T e_{w_j}) \cdot e_{w_j}$$

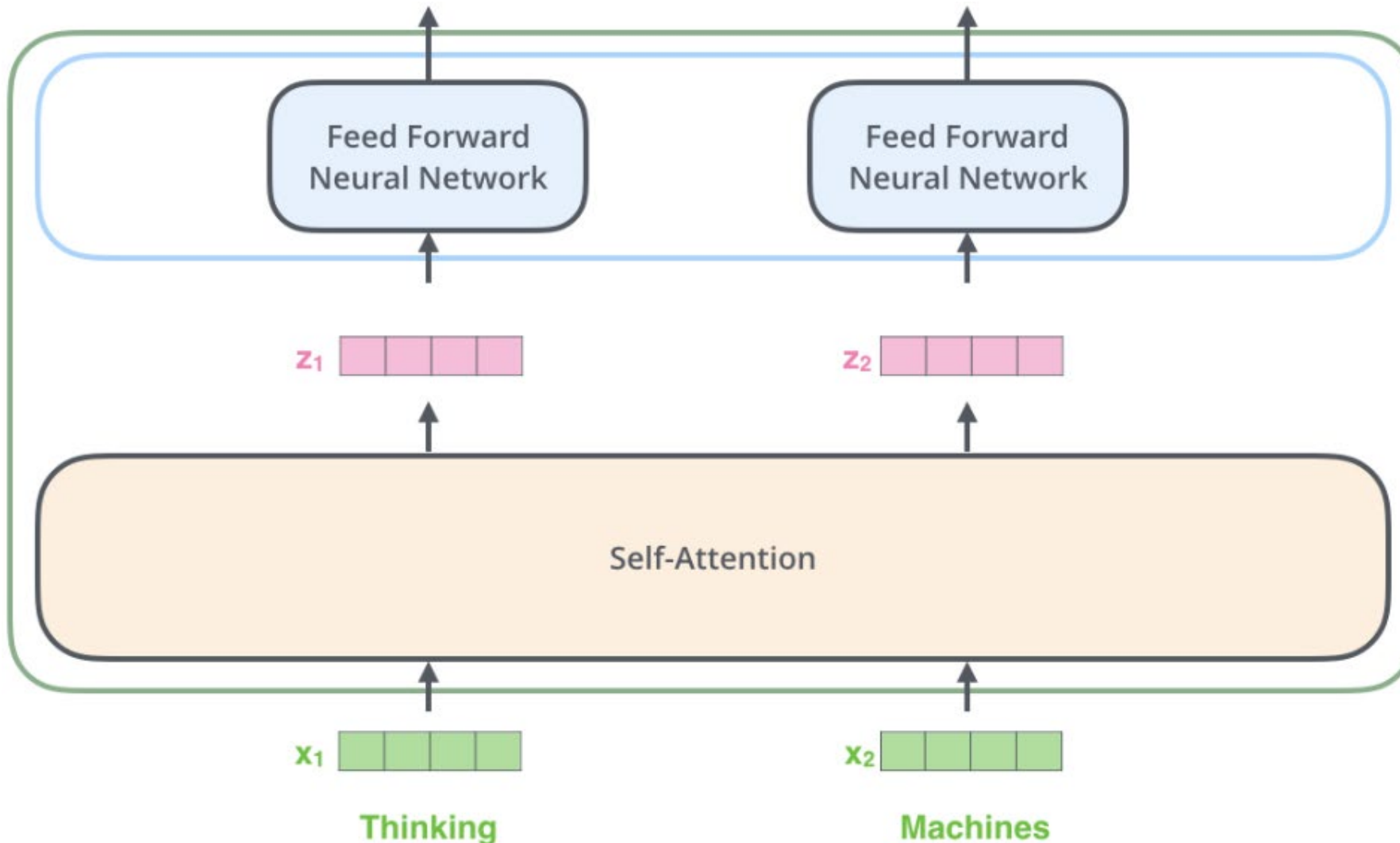
The diagram illustrates the components of the self-attention formula. Three arrows point from the terms in the equation to labels in yellow boxes below:
1. An arrow from $e_{w_i}^T$ (purple) points to a box labeled "Query" (purple).
2. An arrow from e_{w_j} (orange) in the dot product points to a box labeled "Key" (orange).
3. An arrow from e_{w_j} (blue) in the multiplication points to a box labeled "Value" (blue).

Extended Content
(will not appear in quizzes or the exam)

Self-Attention in Transformer: Learning Three Matrices

- Input word vector: \mathbf{e}_{w_i}
- Learn a query matrix \mathbf{W}^Q : $\mathbf{q}_i = \mathbf{e}_{w_i} \mathbf{W}^Q$
- Learn a key matrix \mathbf{W}^K : $\mathbf{k}_i = \mathbf{e}_{w_i} \mathbf{W}^K$
- Learn a value matrix \mathbf{W}^V : $\mathbf{v}_i = \mathbf{e}_{w_i} \mathbf{W}^V$
- Compute attention scores with query and key: $a_{ij} = \text{Softmax}\left(\frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d}}\right)$
 - The dot product of two vectors usually has an expected magnitude proportional to \sqrt{d} , where d is the dimensionality of \mathbf{q}_i and \mathbf{k}_j
 - Divide the attention score by \sqrt{d} to avoid extremely large values in $\text{Softmax}(\cdot)$
- Sum the value vectors weighted by attention scores: $\mathbf{z}_i = \sum_{j=1}^L a_{ij} \mathbf{v}_j$

Self-Attention in Transformer: Visualization

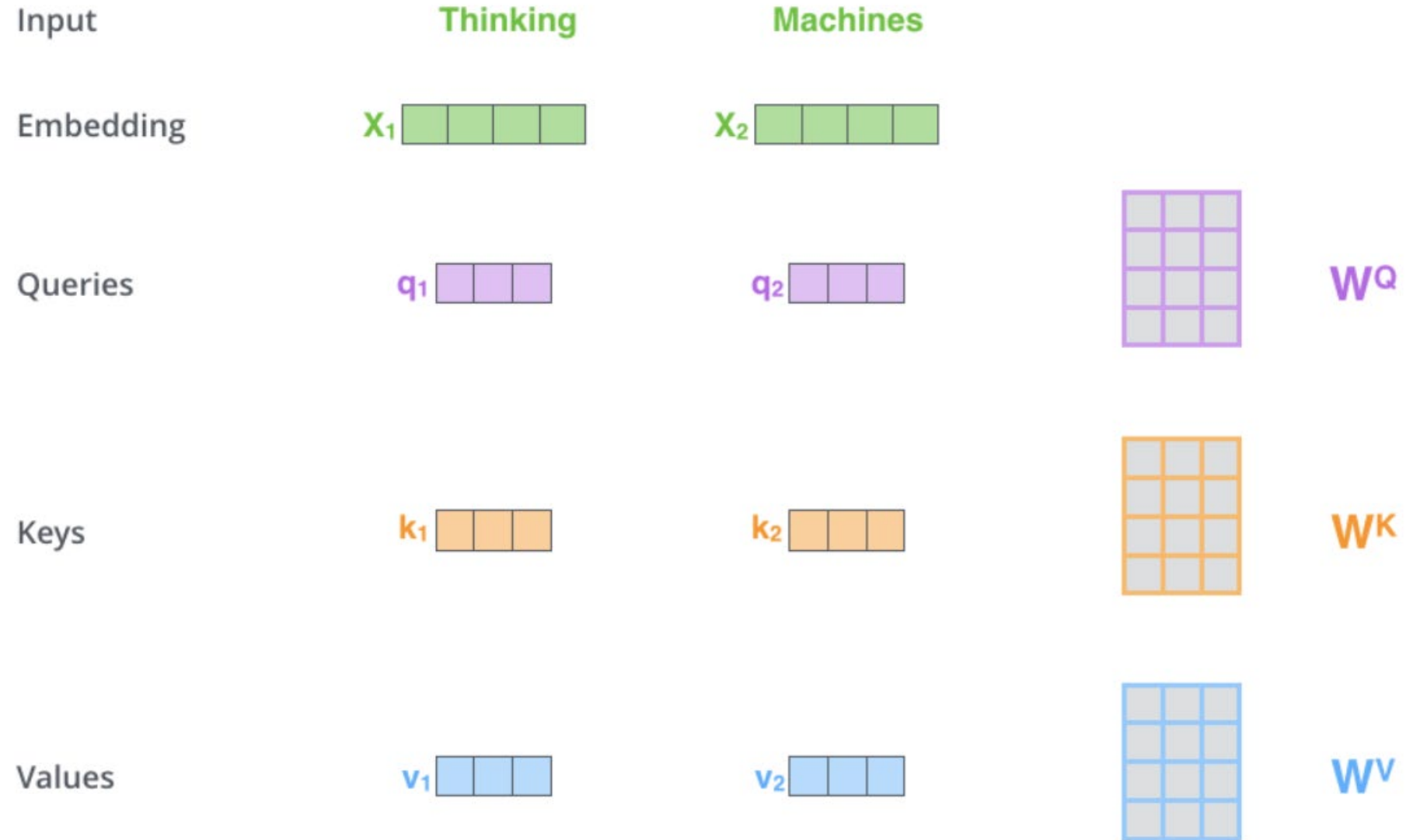


$h_{w_i} = \text{FFN}(z_i)$:
output of one
Transformer layer

z_i : output of
self-attention

x_i : input vector
(i.e., e_{w_i})

Self-Attention in Transformer: Visualization

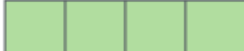


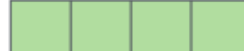
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 

q_2 

Keys

k_1 

k_2 

Values

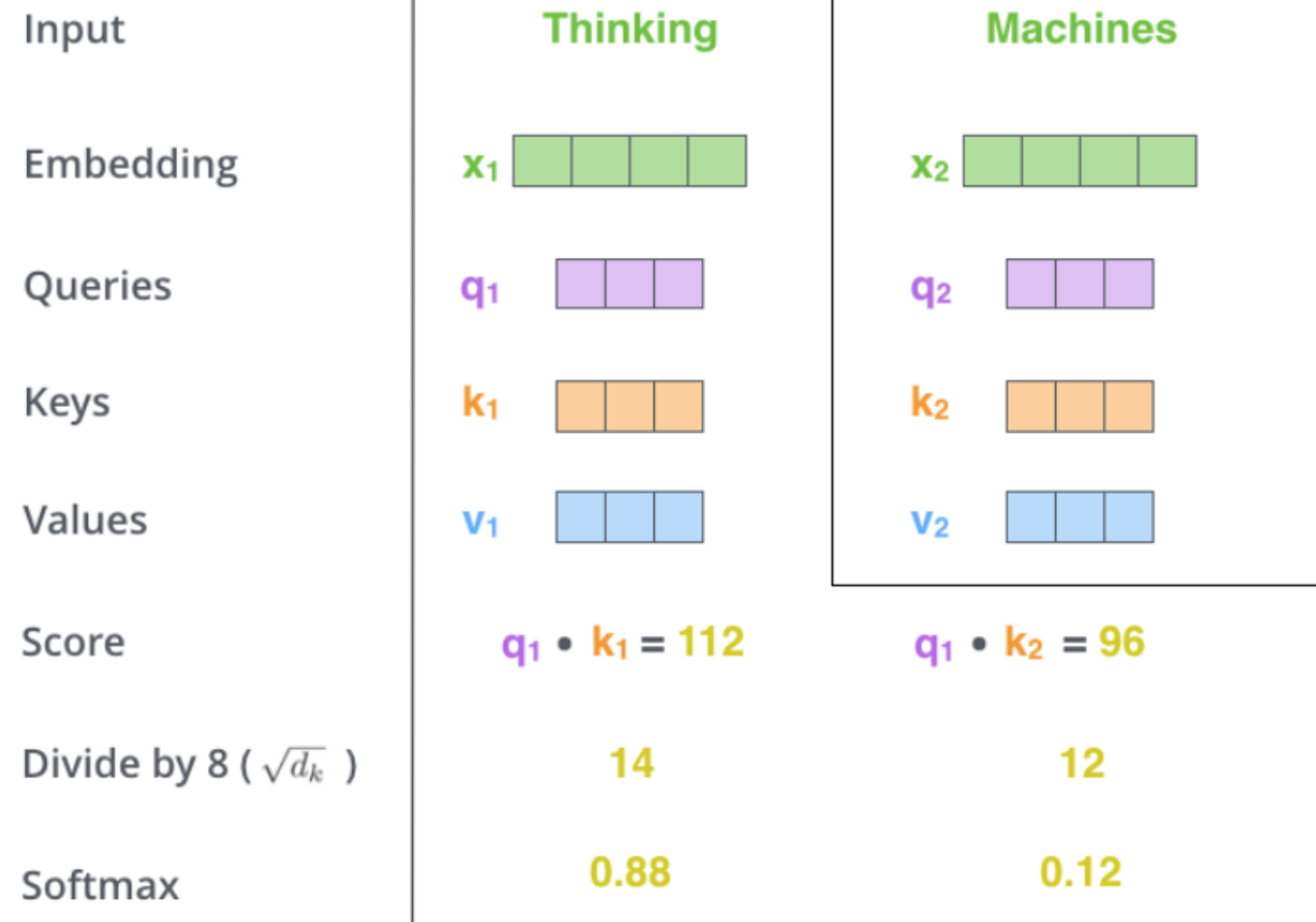
v_1 

v_2 

Score

$$q_1 \cdot k_1 = 112$$

$$q_1 \cdot k_2 = 96$$



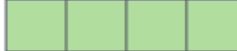
Let's assume these q_i and k_j vectors are 64-dimensional

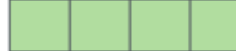
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 

q_2 

Keys

k_1 

k_2 

Values

v_1 

v_2 

Score

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

Divide by 8 ($\sqrt{d_k}$)

14

12

Softmax

0.88

0.12

Softmax

X

Value

v_1 

v_2 

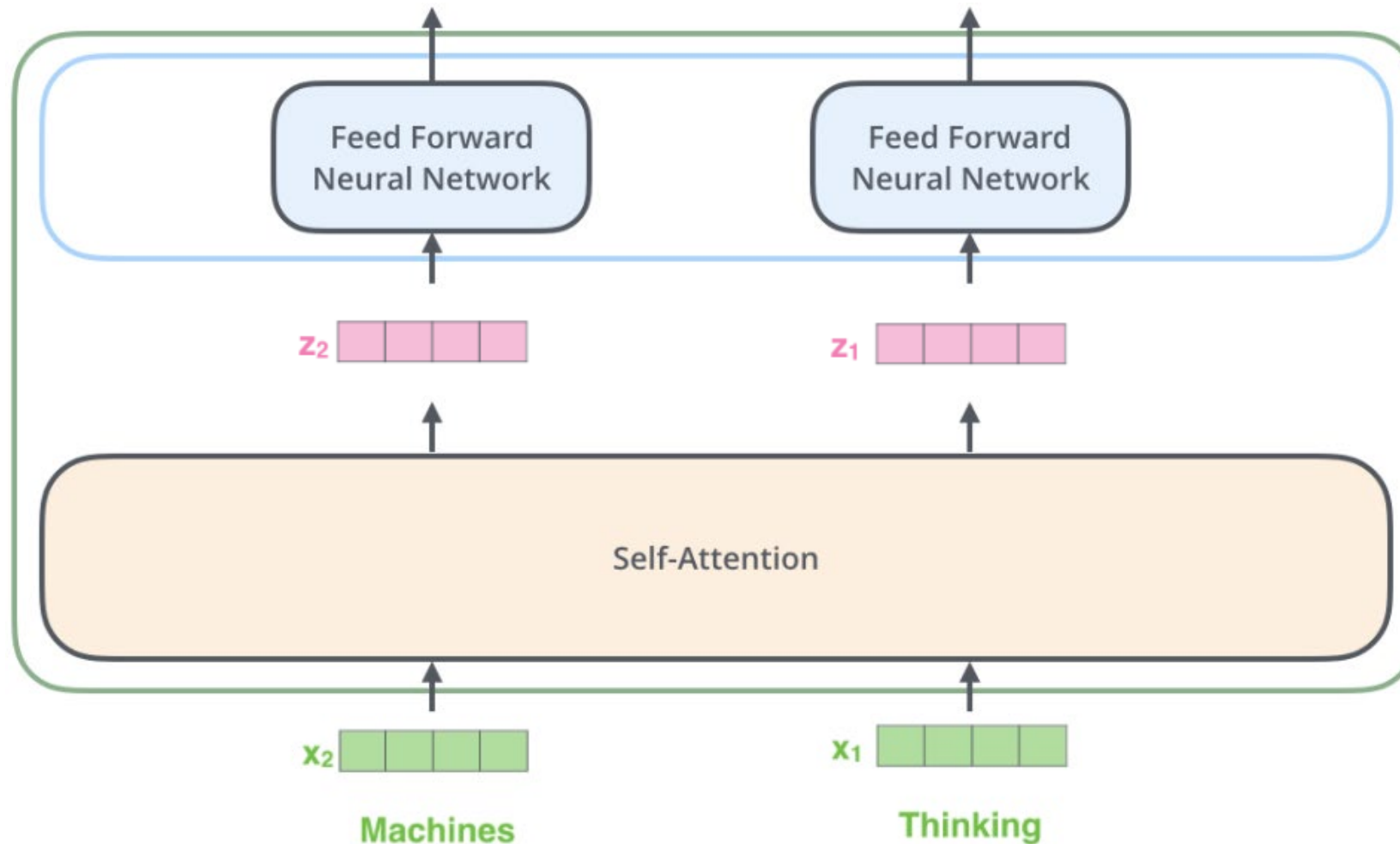
Sum

z_1 

z_2 

Position Encoding

- What will happen to the output if I swap “*Thinking*” and “*Machines*”?



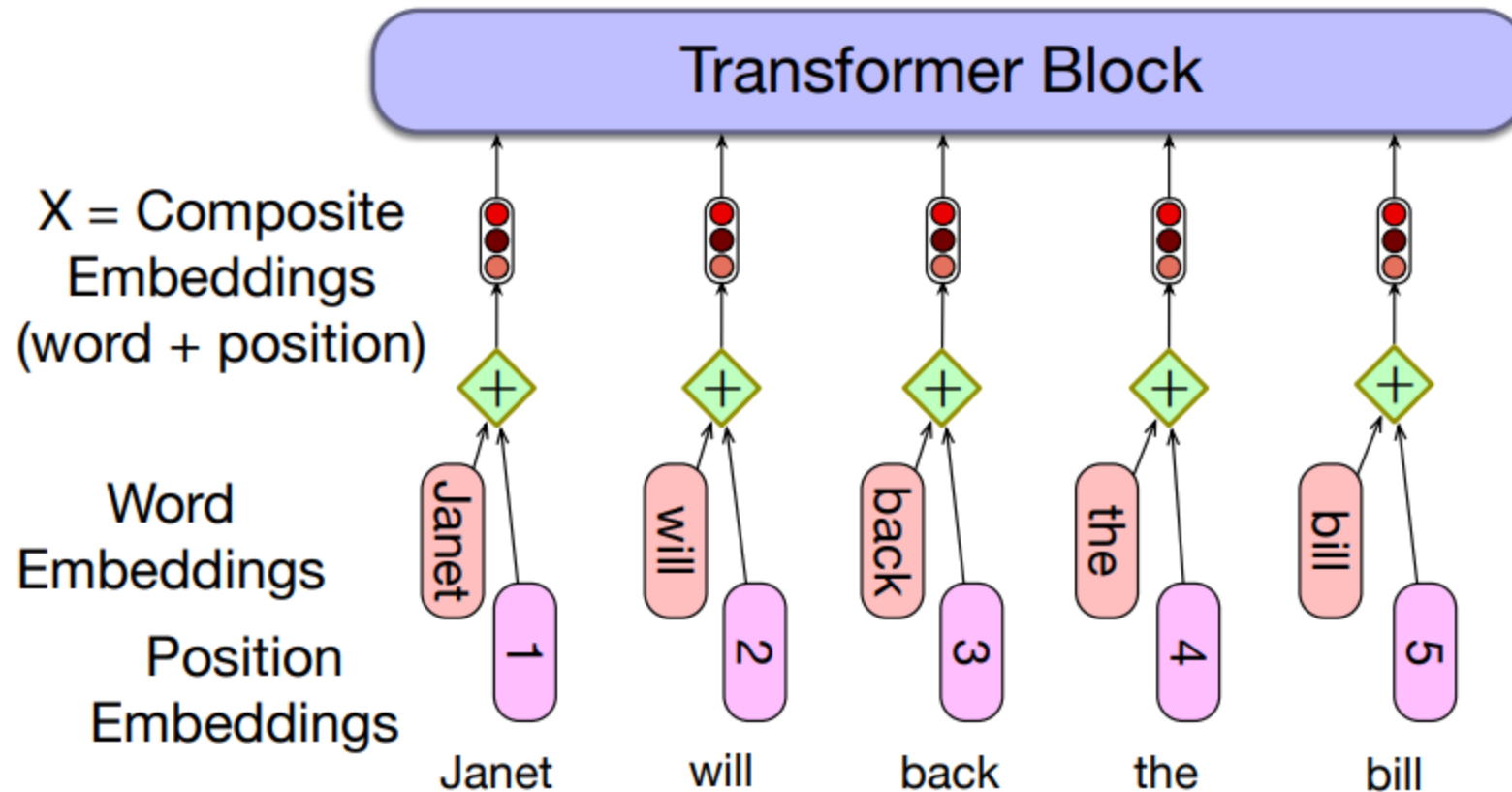
Position Encoding

- What if there are two sentences?
 - “*a dog bites a man*”
 - “*a man bites a dog*”
- The order does not affect the output vector of each word!
- But language is **order-sensitive**!
- **Solution:** Add a learnable position encoding vector p_i

input	$e_a + p_1$	$e_{\text{dog}} + p_2$	$e_{\text{bites}} + p_3$	$e_a + p_4$	$e_{\text{man}} + p_5$
word	<i>a</i>	<i>dog</i>	<i>bites</i>	<i>a</i>	<i>man</i>

input	$e_a + p_1$	$e_{\text{man}} + p_2$	$e_{\text{bites}} + p_3$	$e_a + p_4$	$e_{\text{dog}} + p_5$
word	<i>a</i>	<i>man</i>	<i>bites</i>	<i>a</i>	<i>dog</i>

Position Encoding: Visualization



Tokenization: Handling Out-of-Vocabulary Words

- Byte-Pair Encoding (BPE)
- **Intuition**: start with a character-level vocabulary and iteratively merge the most frequent pairs of tokens
- **Step 1 (Initialization)**: let vocabulary be the set of all individual characters: $\{A, B, C, D, \dots, a, b, c, d, \dots\}$
- **Step 2 (Frequency counting)**: count all adjacent symbol pairs (could be a single character or a previously merged pair) in the training corpus
- **Step 3 (Pair merging)**: merge the most frequent pair of symbols (e.g., “*t*”, “*h*” \rightarrow “*th*”)
- **Step 4 (Update corpus)**: replace all instances of the merged pair in the corpus with the new token & update the frequency of pairs
- **Step 5 (Repeat)**: repeat the process of counting, merging, and updating until a predefined number of merges (or vocabulary size) is reached

Example

- Suppose we have the following corpus:
 - “set new new renew reset renew”

Special “begin-of-word” character
(distinguish between subword
units vs. whole word)

corpus

2	␣	n	e	w		
2	␣	r	e	n	e	w
1	s	e	t			
1	␣	r	e	s	e	t

vocabulary

␣, e, n, r, s, t, w

Example

- Suppose we have the following corpus:
 - “set new new renew reset renew”

corpus

```
2    _ n e w
2    _ r e n e w
1    s e t
1    _ r e s e t
```

merging
“n” and “e”



corpus

```
2    _ ne w
2    _ r e ne w
1    s e t
1    _ r e s e t
```

vocabulary

_ , e , n , r , s , t , w

vocabulary

_ , e , n , r , s , t , w , ne

Example

- Suppose we have the following corpus:
 - “set new new renew reset renew”

corpus

```
2  _ ne w
2  _ r e ne w
1  s e t
1  _ r e s e t
```

merging “ne”
and “w”



corpus

```
2  _ new
2  _ r e new
1  s e t
1  _ r e s e t
```

vocabulary

_ , e , n , r , s , t , w , ne

vocabulary

_ , e , n , r , s , t , w , ne ,
new

Example

- Suppose we have the following corpus:
 - “set new new renew reset renew”

corpus

```
2  _ new
2  _ r e new
1  s e t
1  _ r e s e t
```



corpus

```
2  _ new
2  _re new
1  s e t
1  _re s e t
```

vocabulary

_, e, n, r, s, t, w, ne,
new

vocabulary

_, e, n, r, s, t, w, ne,
new, _r, _re

Example

- Suppose we have the following corpus:
 - “*set new new renew reset renew*”
- If we continue, the next merges are:

merge	current vocabulary
(<code>_</code> , new)	<code>_</code> , e, n, r, s, t, w, ne, new, <code>_r</code> , <code>_re</code> , <code>_new</code>
(<code>_re</code> , new)	<code>_</code> , e, n, r, s, t, w, ne, new, <code>_r</code> , <code>_re</code> , <code>_new</code> , <code>_renew</code>
(s, e)	<code>_</code> , e, n, r, s, t, w, ne, new, <code>_r</code> , <code>_re</code> , <code>_new</code> , <code>_renew</code> , se
(se, t)	<code>_</code> , e, n, r, s, t, w, ne, new, <code>_r</code> , <code>_re</code> , <code>_new</code> , <code>_renew</code> , se, set

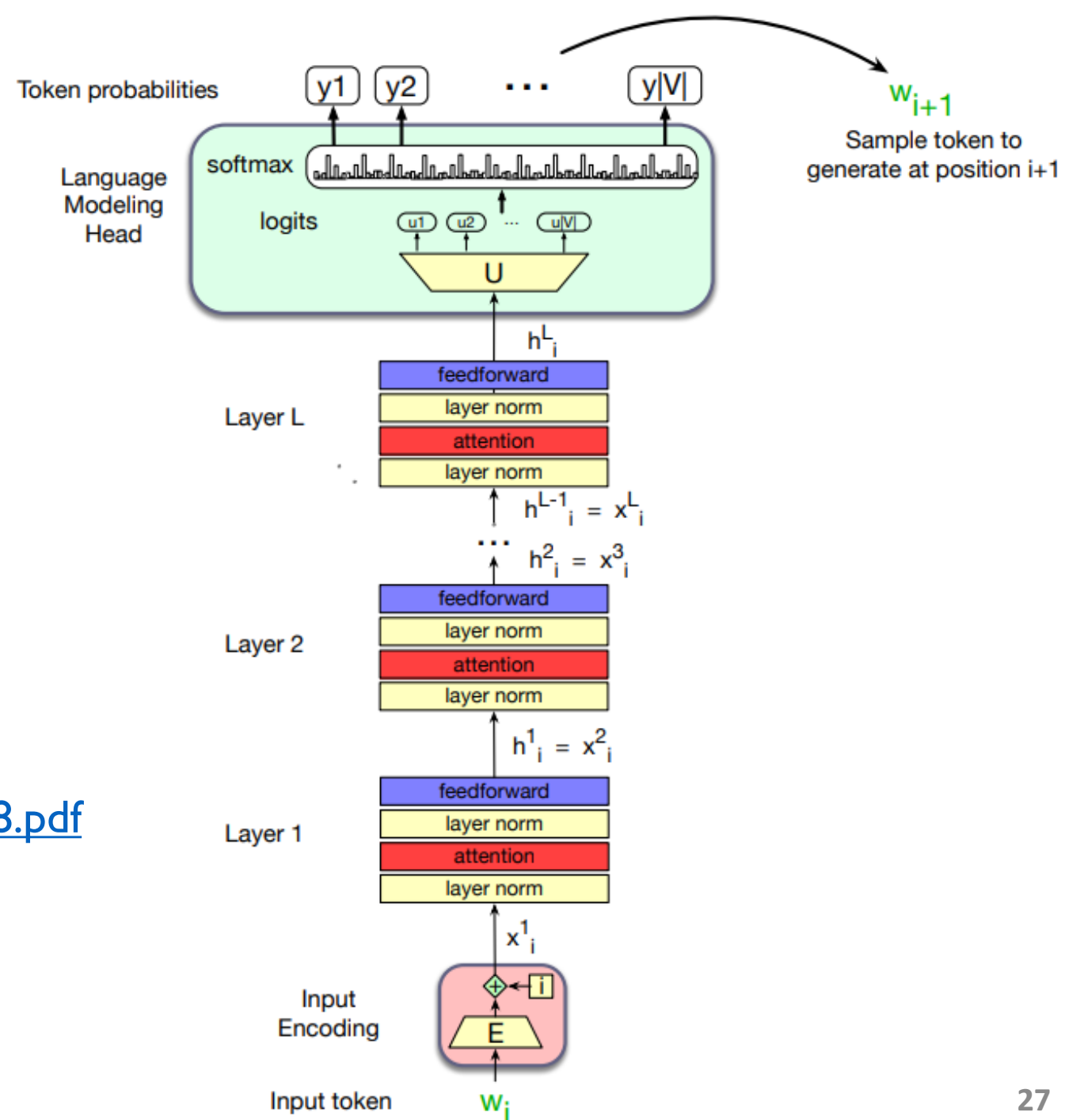
- Given a new sentence, how to tokenize it?
 - Just run (greedily based on training data frequency) on the merge rules we have learned from the training data on the new sentence

Still many missing details in the Transformer!

- Multi-head attention
- Feedforward network
- Layer normalization
- Residual connection

- Refer to:

<https://web.stanford.edu/~jurafsky/slp3/8.pdf>



Training Transformers with Self-Supervision

(Required content begins from this slide,
which may appear in quizzes and the exam!)

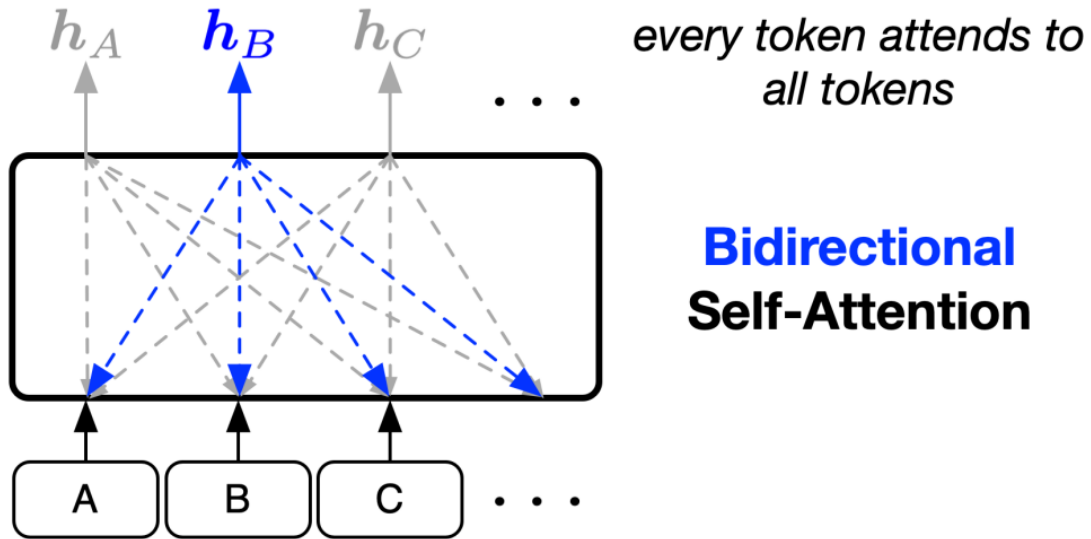
Information in Raw Texts

- *Verb*: the task is to all the structured chemical reactions from papers
- *Preposition*: a liquid handler equipped two microplates
- *Time*: the Transformer paper was published in
- *Location*: data from the University of MD Anderson Cancer Center
- *Math*: the sequence goes 1, 1, 2, 3, 5, 8, 13, 21,
- *Chemistry*: sugar is composed of carbon, hydrogen, and
- ...
- How to harvest underlying patterns, structures, and semantic knowledge from raw texts?
 - Train the model to predict masked tokens given their contexts

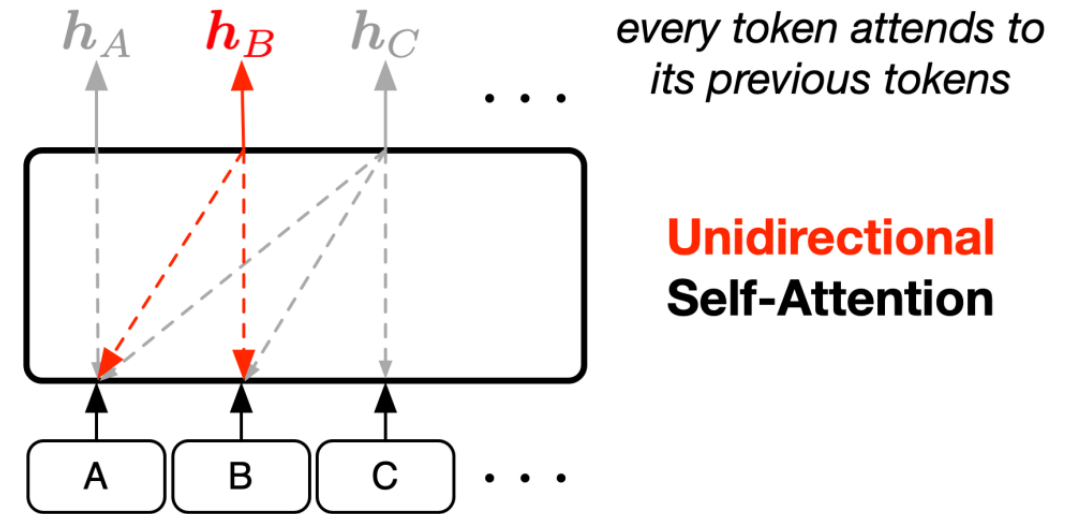
Information in Raw Texts

- *Verb*: the task is to extract all the structured chemical reactions from papers
- *Preposition*: a liquid handler equipped with two microplates
- *Time*: the Transformer paper was published in 2017
- *Location*: data from the University of Texas MD Anderson Cancer Center
- *Math*: the sequence goes 1, 1, 2, 3, 5, 8, 13, 21, 34
- *Chemistry*: sugar is composed of carbon, hydrogen, and oxygen
- ...
- Encoder-based language models (e.g., **BERT**) – predict a token from **all other tokens** in the input sequence
chatgpt learns from vast amounts of text data and generates responses ...
- Decoder-based language models (e.g., **ChatGPT**) – predict a token from **all previous tokens**
chatgpt learns from vast amounts of text data and generates responses ...

Two Types of Transformer Architecture



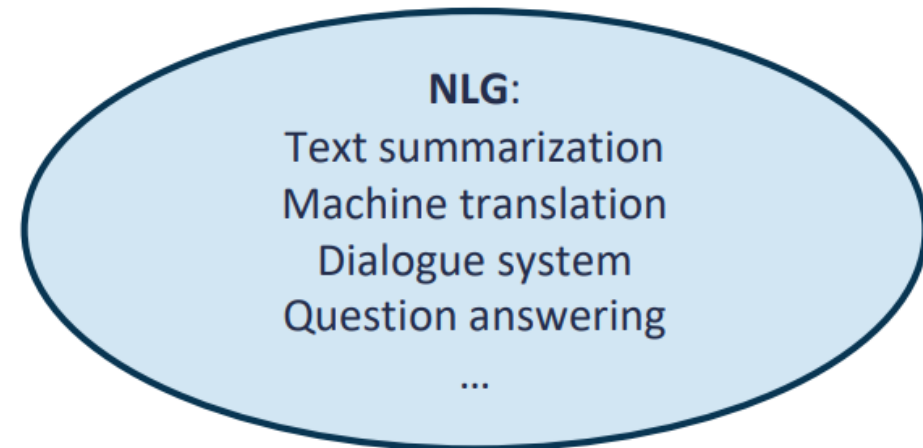
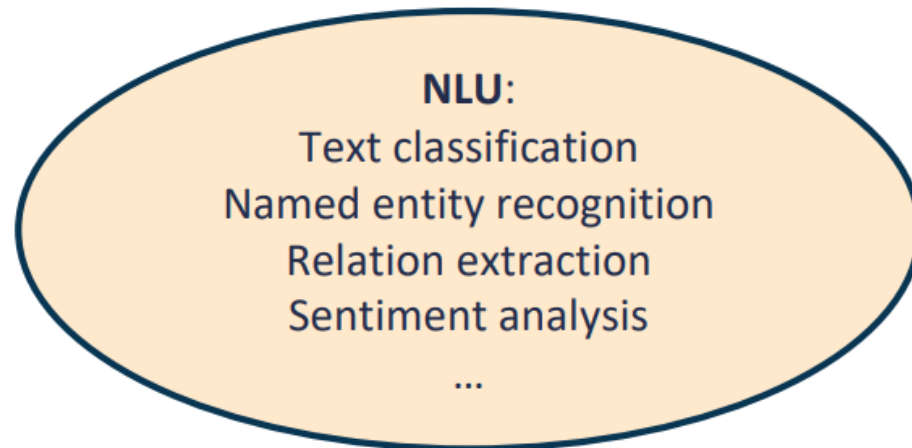
$q1 \cdot k1$	$q1 \cdot k2$	$q1 \cdot k3$	$q1 \cdot k4$
$q2 \cdot k1$	$q2 \cdot k2$	$q2 \cdot k3$	$q2 \cdot k4$
$q3 \cdot k1$	$q3 \cdot k2$	$q3 \cdot k3$	$q3 \cdot k4$
$q4 \cdot k1$	$q4 \cdot k2$	$q4 \cdot k3$	$q4 \cdot k4$



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

Encoder vs. Decoder

- Encoder:
 - Each token can attend to all other tokens **to better learn its representation vector**
 - Suitable for natural language understanding (NLU) tasks
- Decoder:
 - Each token can only attend to previous tokens **to predict the next token**
 - Suitable for natural language generation (NLG) tasks



BERT [Devlin et al., NAACL 2019]

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **Bidirectional Encoder Representations from Transformers**. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a re-

There are two existing pre-trained language representation models: *feature-based* and *task-specific*. The *feature-based* approach, (Devlin et al., 2018a), uses task-specific pre-trained representations. The *task-specific* approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal

Bert: Pre-training of deep bidirectional transformers for language understanding

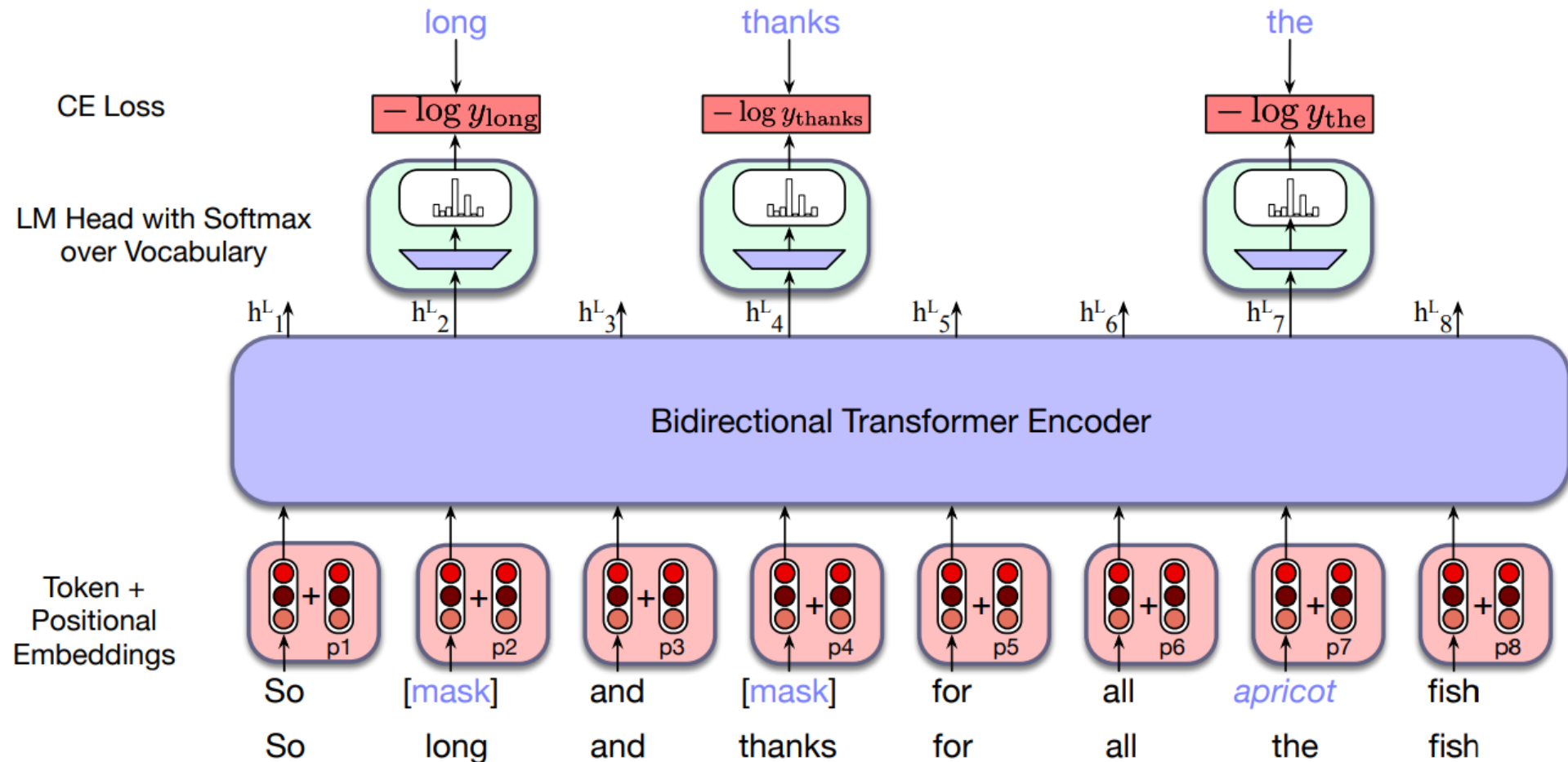
J Devlin, MW Chang, K Lee... - Proceedings of the 2019 ..., 2019...
aclanthology.org

... **deep** bidirectionality of **BERT** by evaluating two **pretraining** objectives: **Masked Language Modeling** (MLM) and **No** NSP: A **bidirectional** model is trained using the “masked LM” (MLM) ...

☆ Cited by 146893 Related articles

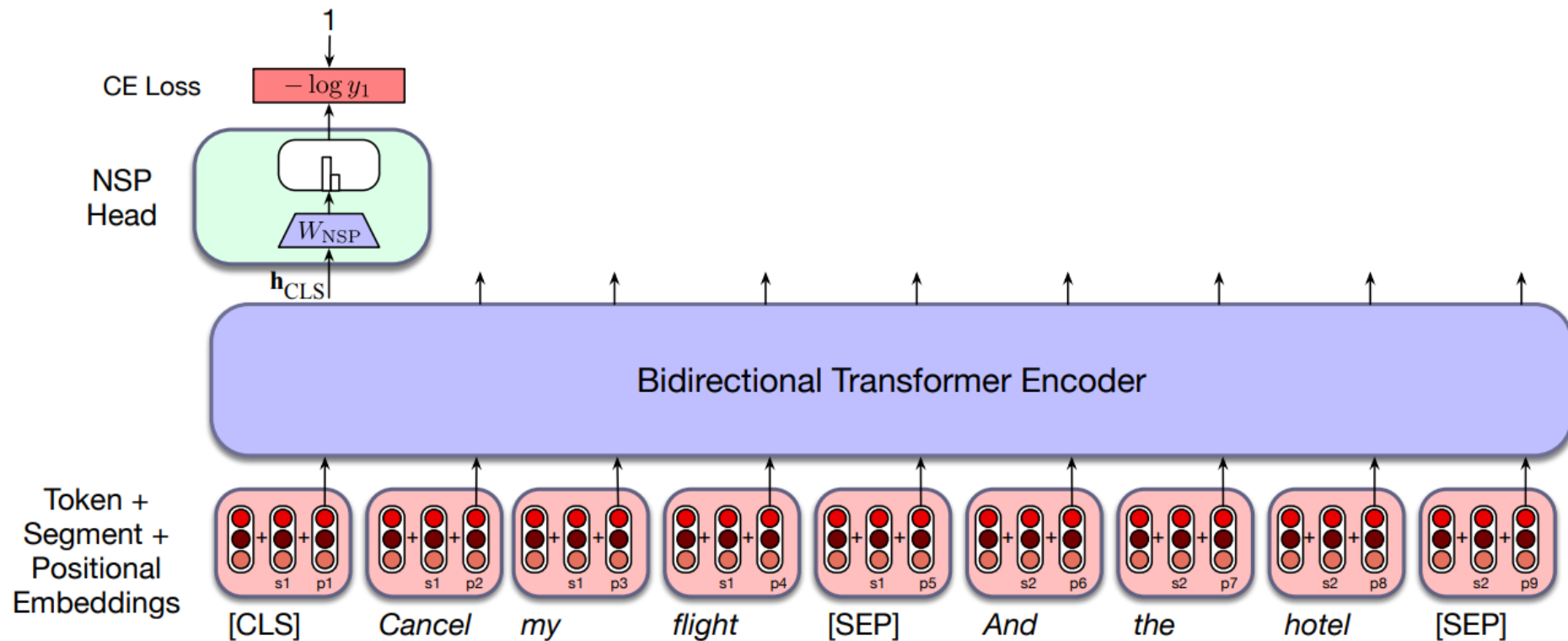
BERT Pre-training

- **Task 1 – Masked Language Modeling (MLM):** With 15% words randomly masked, the model learns bidirectional contextual information to predict the masked words.



BERT Pre-training

- **Task 2 – Next Sentence Prediction (NSP):** The model is presented with pairs of sentences. It is trained to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentence.



Immediate Impact of BERT

- In 2018, BERT came out and largely outperformed most previous methods on common NLP tasks (e.g., sentiment classification, natural language inference, question answering).
- BERT got the best paper award at the NAACL 2019 conference.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

- “Open AI GPT”: GPT-2
- “BERT-Base”: 12 Transformer encoder layers; ~110M parameters
- “BERT-Large”: 24 Transformer encoder layers; ~340M parameters

Improving BERT: RoBERTa [Liu et al., arXiv 2019]

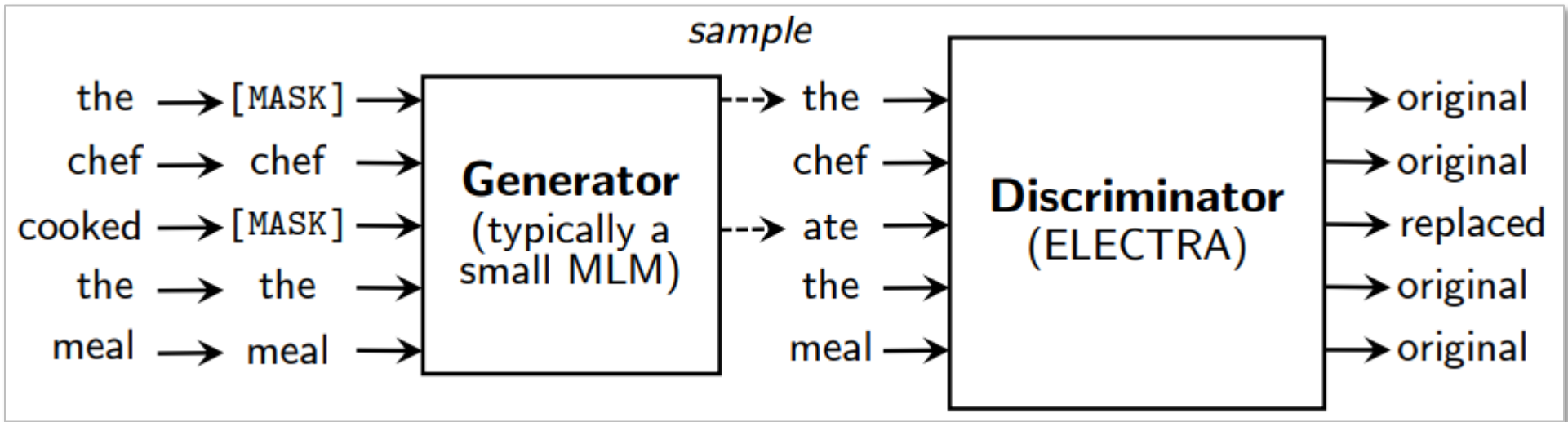
- Next Sentence Prediction (NSP) is not helpful!
- Only use Masked Language Modeling (MLM)
- Pretrain on longer sequences
- Pretrain the model for longer, with bigger batches
- Pretrain over more data
- Dynamically change the masking patterns applied to the training data in each epoch

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
<i>Our reimplementation (with NSP loss):</i>				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
<i>Our reimplementation (without NSP loss):</i>				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4

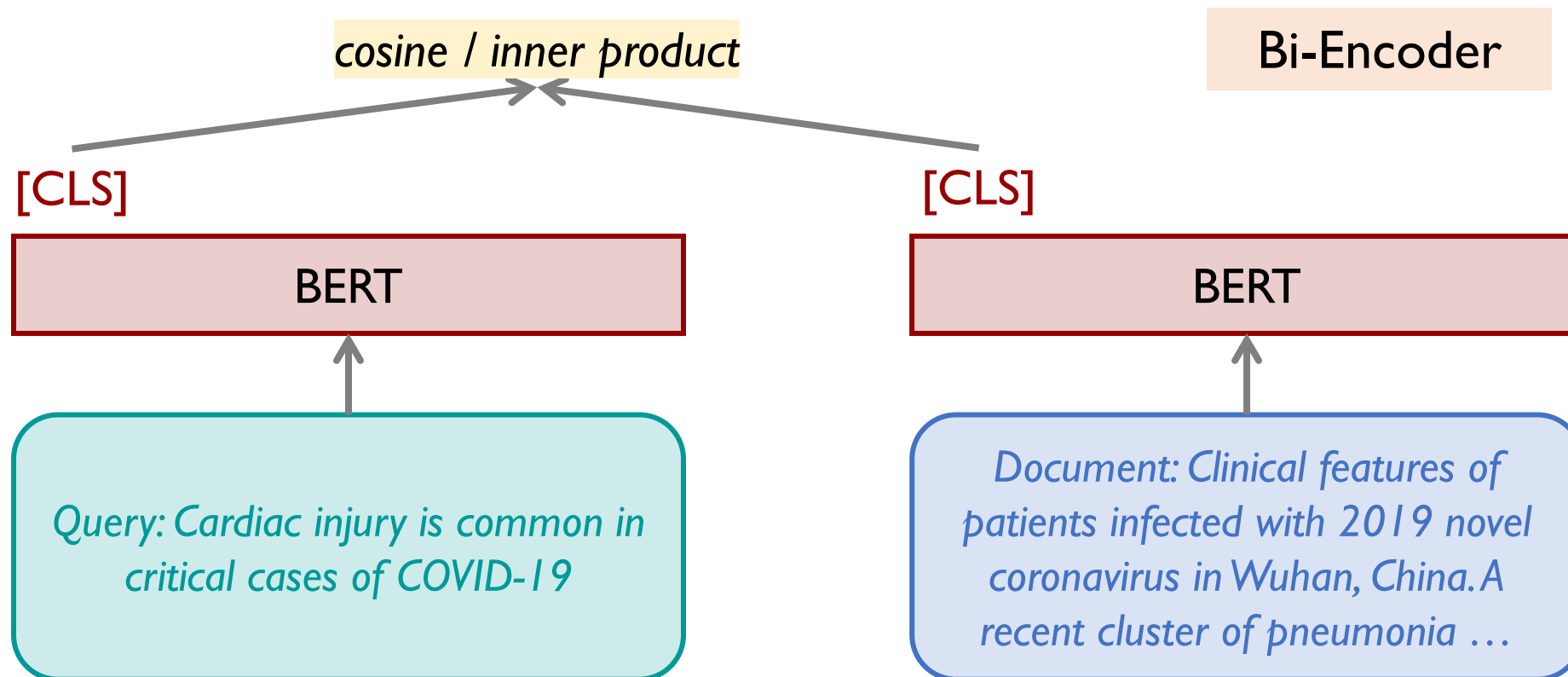
Improving BERT: ELECTRA [Clark et al., ICLR 2020]

- Use a small MLM as an auxiliary generator (discarded after pretraining)
- Pretrain the main model as a discriminator
- The small auxiliary MLM and the main discriminator are jointly trained.
- The main model's pretraining task becomes **more and more challenging** in pretraining.




How to use BERT for retrieval?

- Encode **query** and **document** separately
- The output vector of the [CLS] token serves as **query** / **document** embedding



Python Implementation to Encode a Query / Document

python

 Copy code

```
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained BERT-base model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Input text
text = "Cardiac injury is common in critical cases of COVID-19"

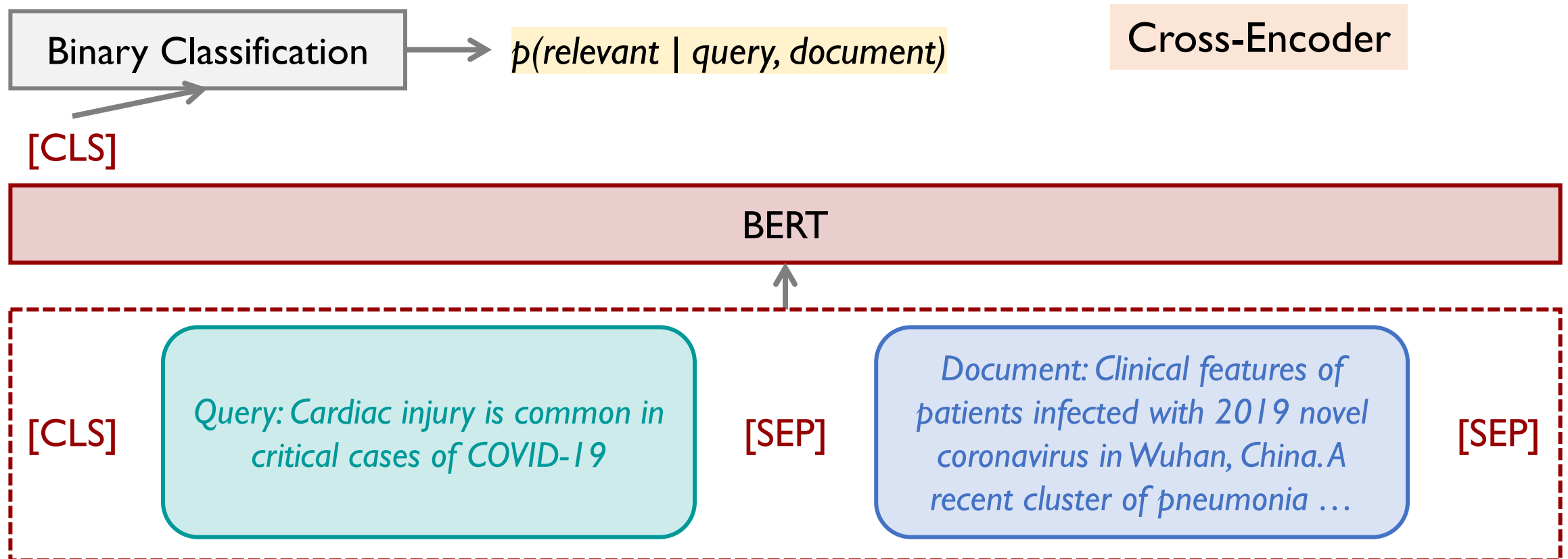
# Tokenize and encode input
inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True)

# Forward pass (no gradient needed)
with torch.no_grad():
    outputs = model(**inputs)

# Extract [CLS] token embedding
cls_embedding = outputs.last_hidden_state[:, 0, :] # shape: [1, 768]
```


How to use BERT for retrieval?

- Concatenate the **query** and **document** into a single input sequence
- Get the representation of the entire sequence and perform binary classification





Thank You!

Course Website: <https://yuzhang-teaching.github.io/CSCE670-F25.html>