CSCE 670 - Information Storage and Retrieval

Lecture 17: Contextualized Language Models, BERT

Yu Zhang

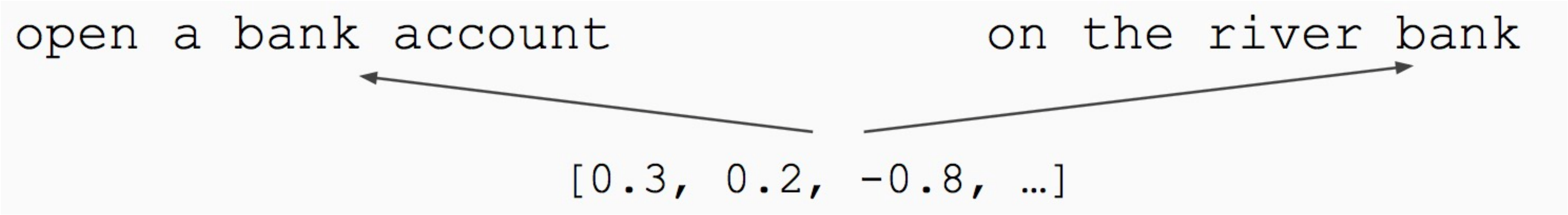
yuzhang@tamu.edu

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

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 d **attention** by masking out (setting to $-\infty$) …

☆ Cited by 199331 Related articles

Attention Is All You Need

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 lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer as a Black Box

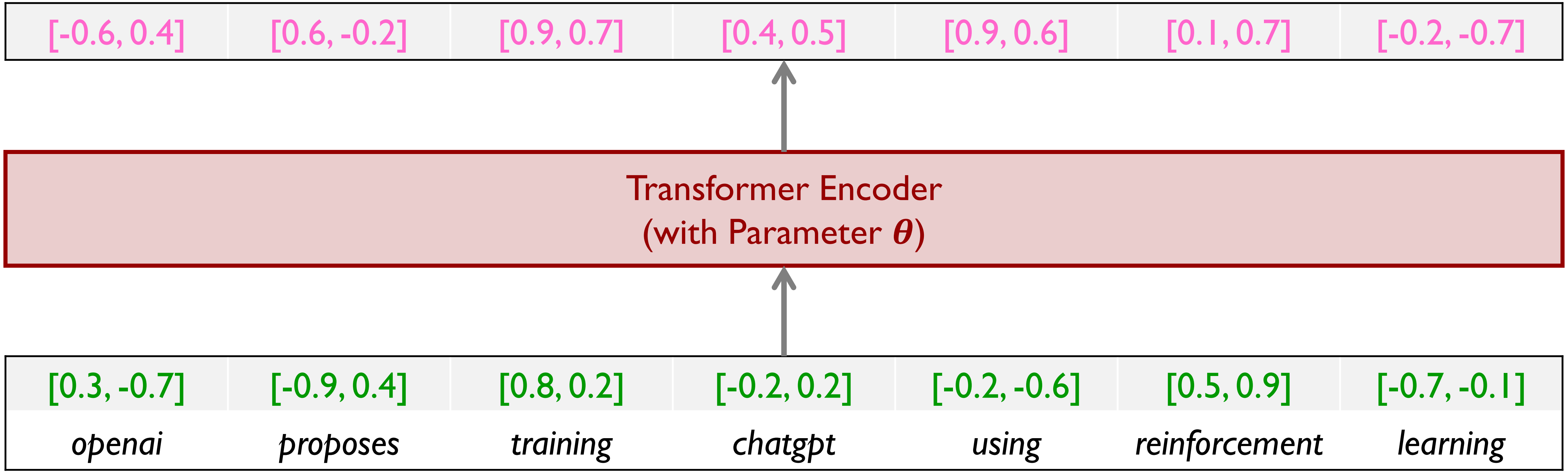
- Transformer is a neural network
- It has two types of architecture: encoder and decoder
- In this lecture, we focus on the Transformer encoder

- 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 $\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \dots, \boldsymbol{e}_{w_L}$

- Then, the output is a sequence of contextualized word vectors $\boldsymbol{h}_{w_1}, \boldsymbol{h}_{w_2}, \dots, \boldsymbol{h}_{w_L}$
  - The output vector $\boldsymbol{h}_{w_i}$ captures the meaning of w_i by considering the entire input sequence as w_i's context
  - $\boldsymbol{h}_{w_i} = \mathrm{Transformer}(\boldsymbol{e}_{w_i} | \boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \dots, \boldsymbol{e}_{w_L})$

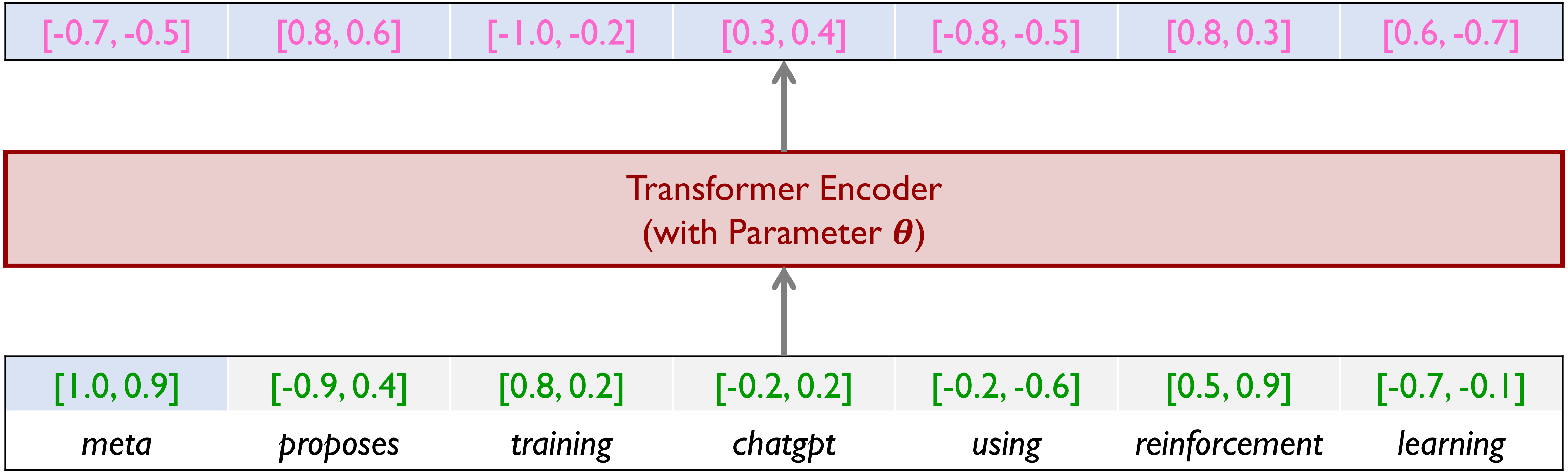
Transformer as a Black Box

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



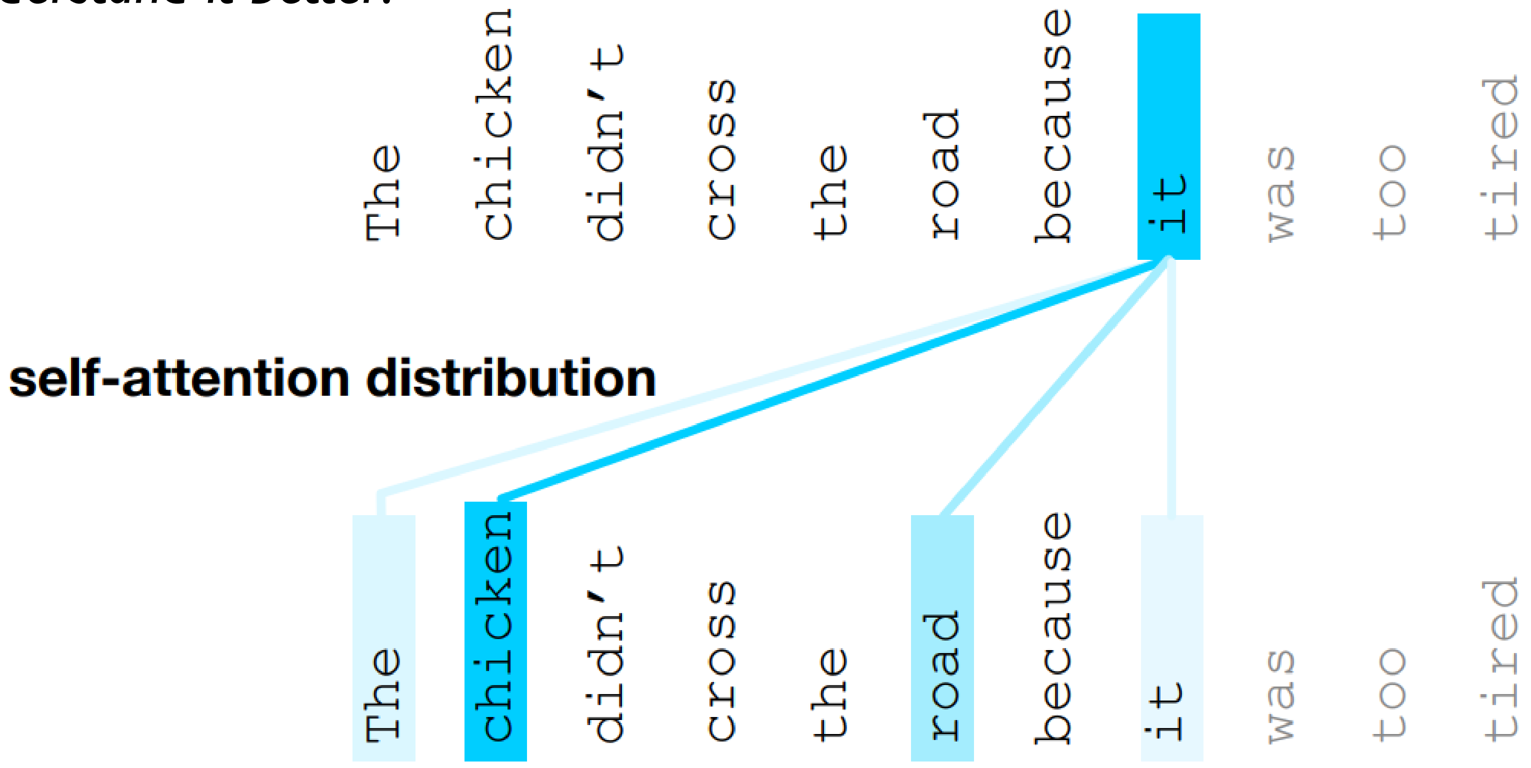
Transformer as a Black Box

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



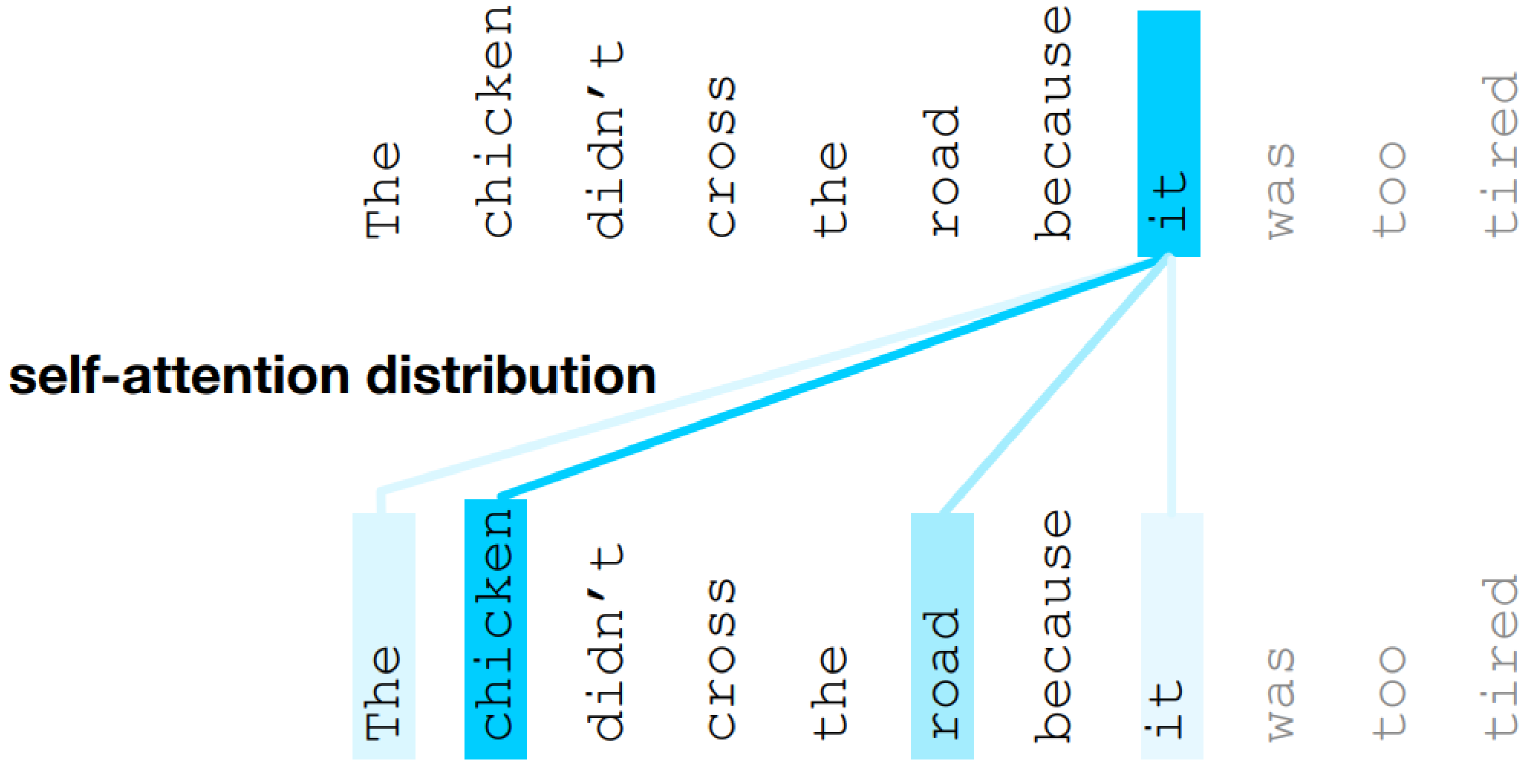
How does the model transform the input into the output?

- "(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

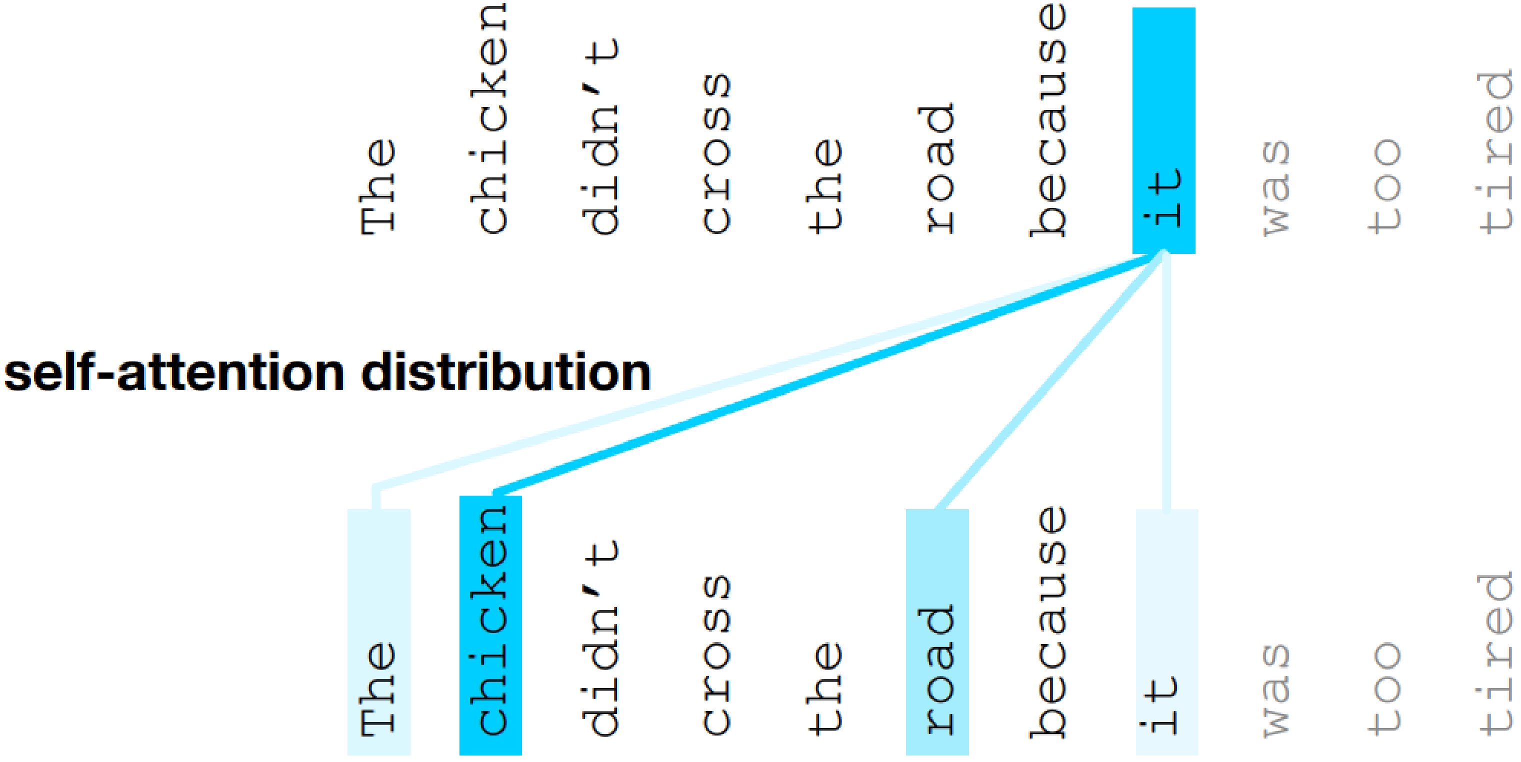
- $\boldsymbol{h}_{w_i} = \sum_{j=1}^{L} a_{ij} \boldsymbol{e}_{w_j}$
- a_{ij}: attention score, where $\sum_{j=1}^{L} a_{ij} = 1$



Self-Attention: A Simplified Example

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

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

$$\boldsymbol{h}_{w_i} = \sum_{j=1}^{L} \text{Softmax}(\boldsymbol{e}_{w_i}^{T} \boldsymbol{e}_{w_j}) \cdot \boldsymbol{e}_{w_j}$$

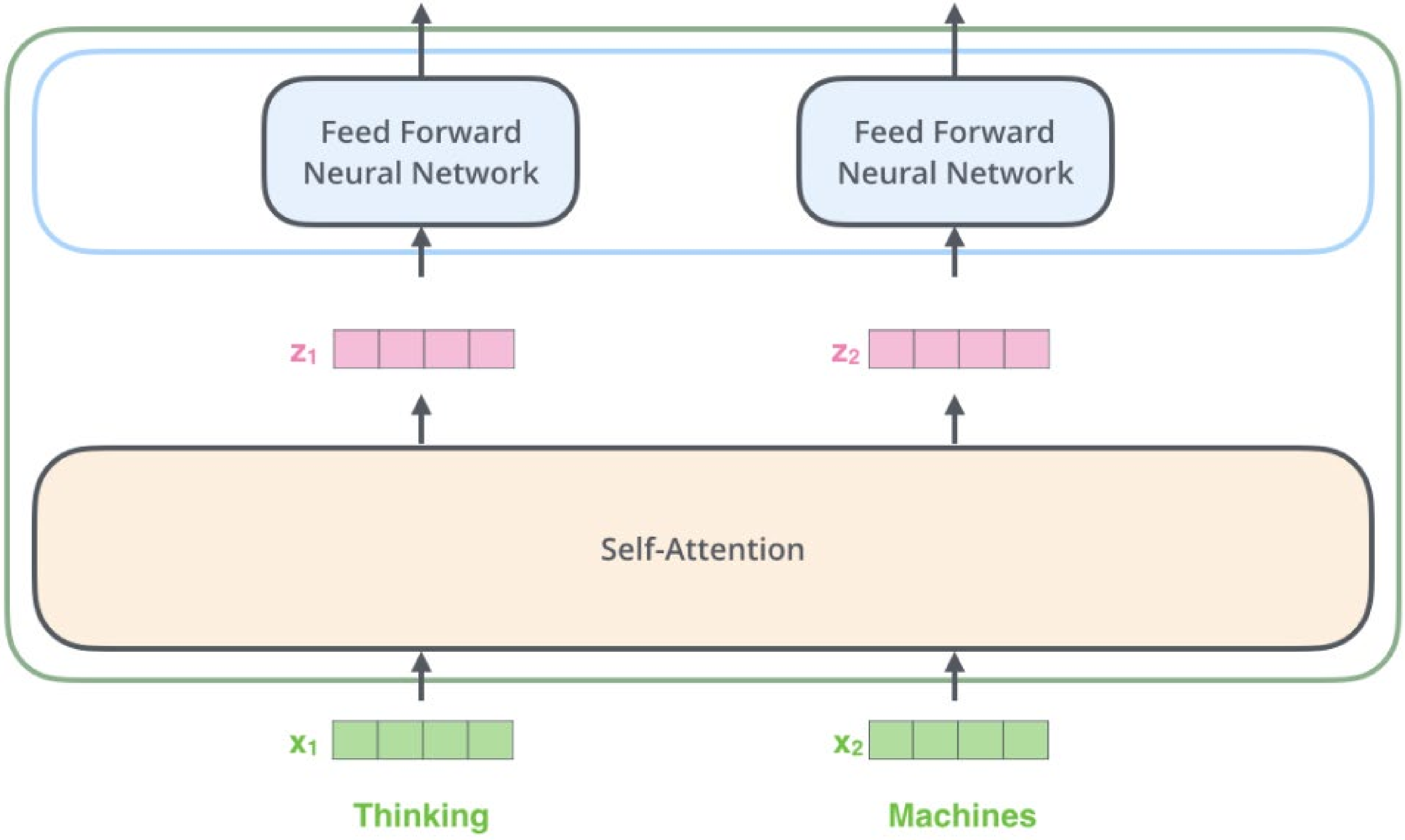
Query Key Value

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

Self-Attention in Transformer: Learning Three Matrices

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

Self-Attention in Transformer: Visualization



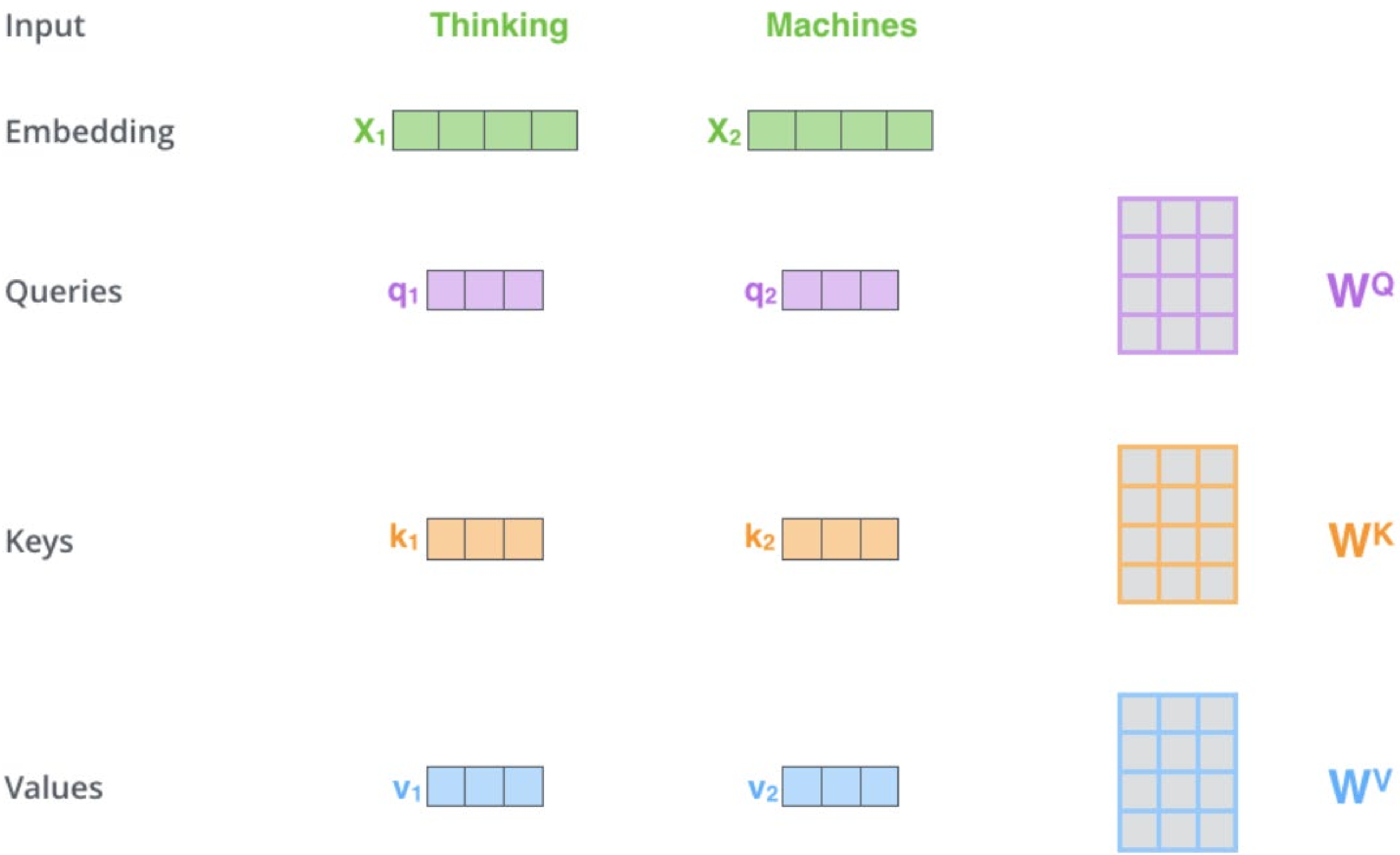
$\boldsymbol{h}_{w_i} = \mathrm{FFN}(\boldsymbol{z}_i)$: output of one Transformer layer

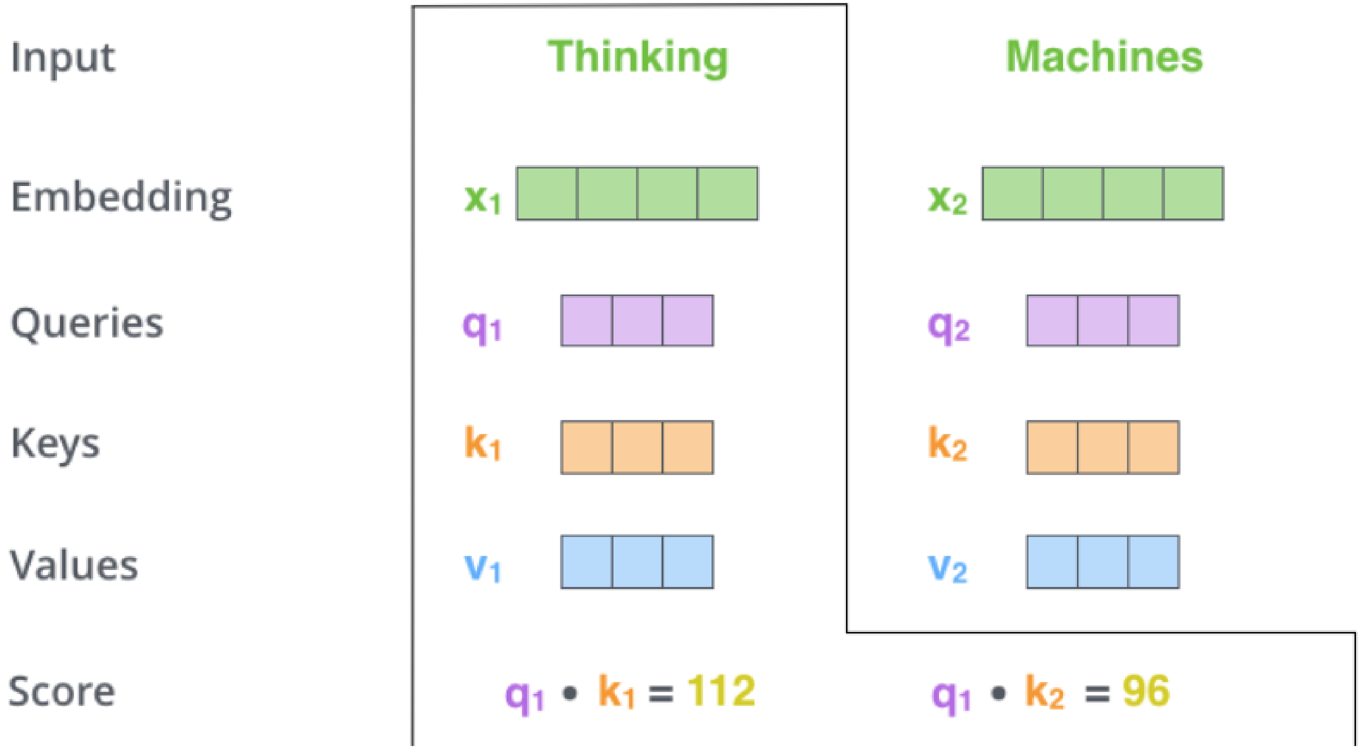
$\boldsymbol{z}_i$: output of self-attention

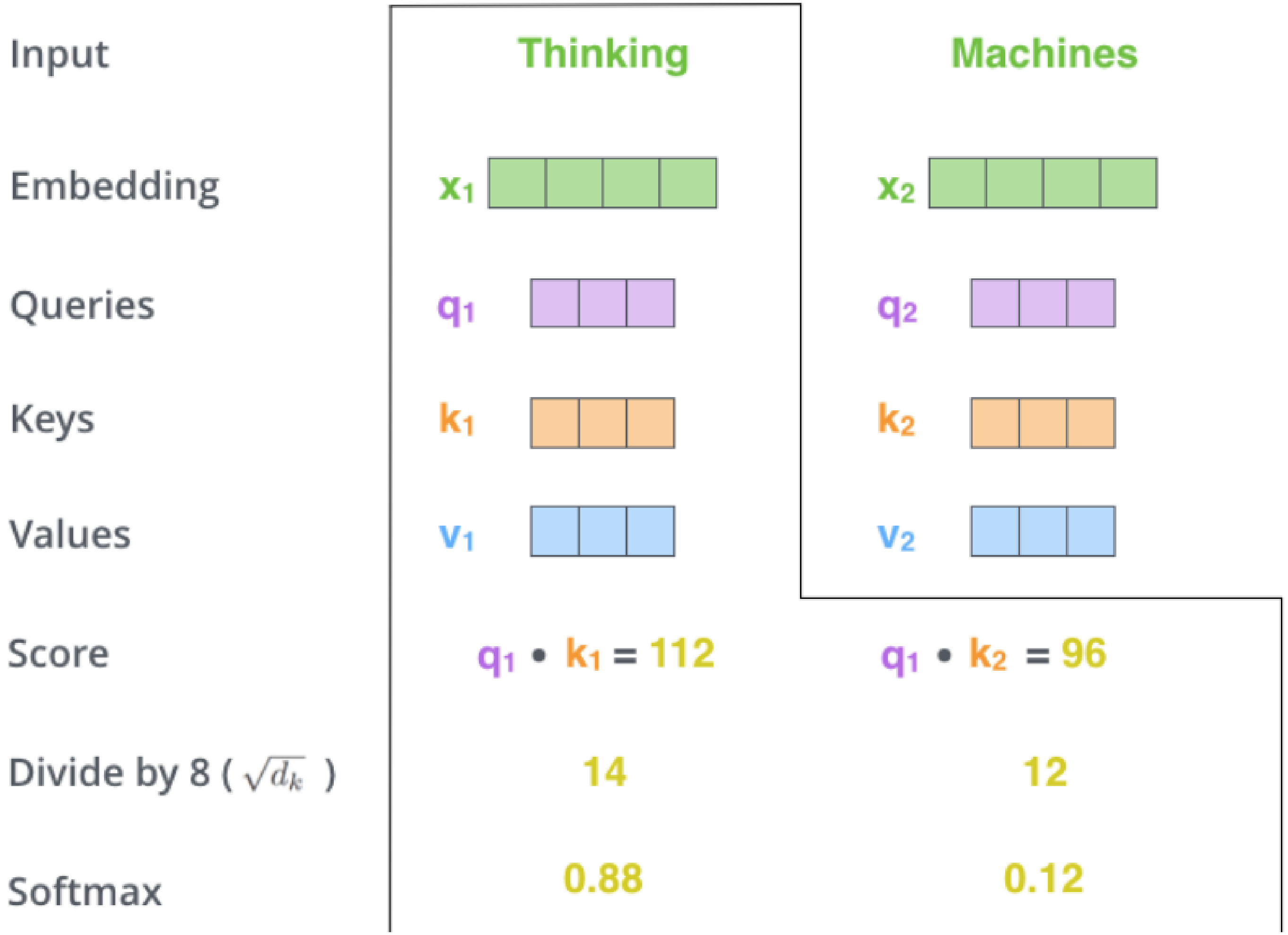
$\boldsymbol{x}_i$: input vector (i.e., $\boldsymbol{e}_{w_i}$)

Source: https://jalammar.github.io/illustrated-transformer/

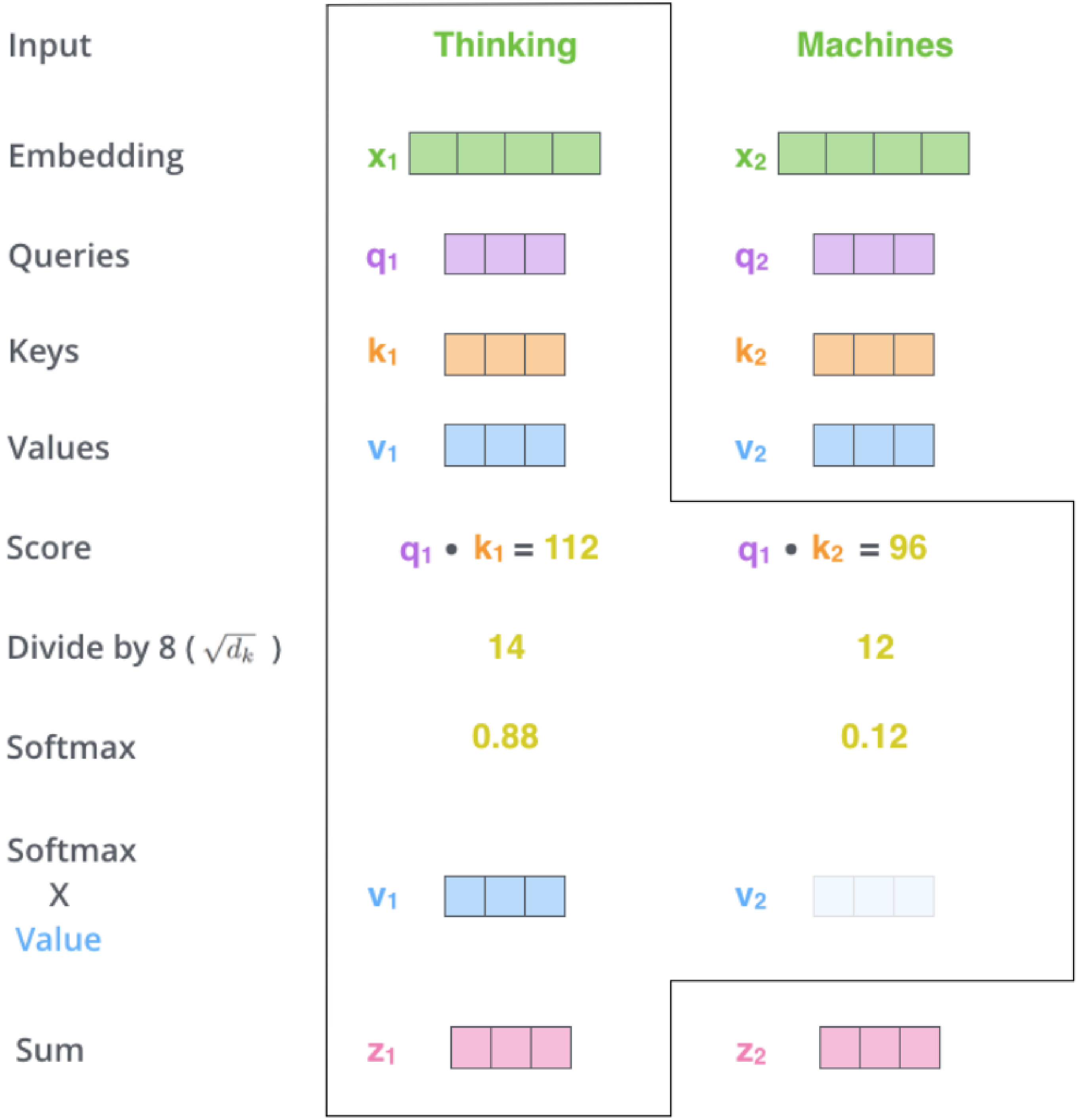
Self-Attention in Transformer: Visualization




Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96

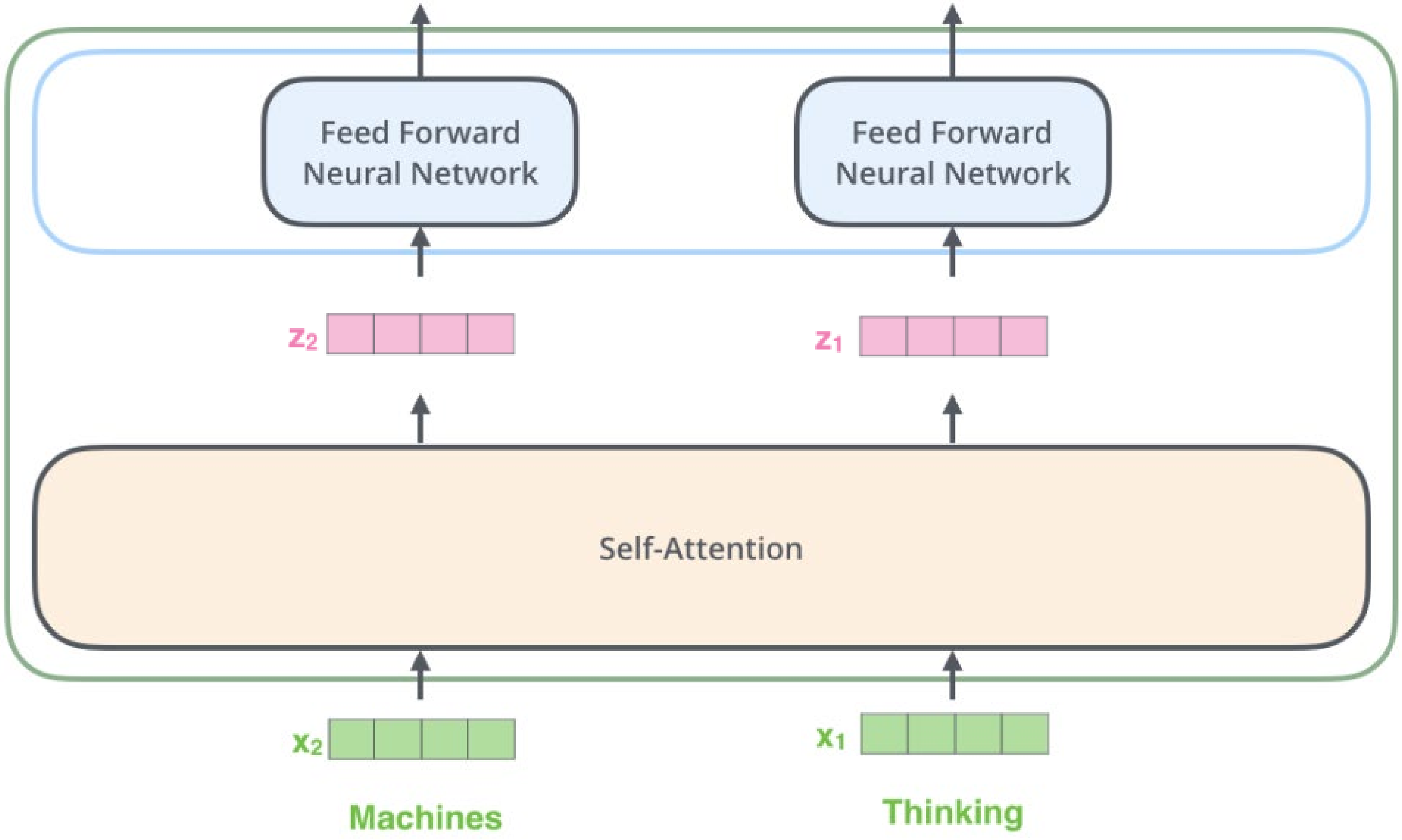


Let's assume these q_i and k_j vectors are 64-dimensional


Input
Thinking
Machines
Embedding
x1
x2
Queries
q1
q2
Keys
k1
k2
Values
v1
v2
Score
q1 • k1 = 112
q1 • k2 = 96
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12
Softmax
X
Value
v1
v2
Sum
z1
z2

Position Encoding

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



Position Encoding

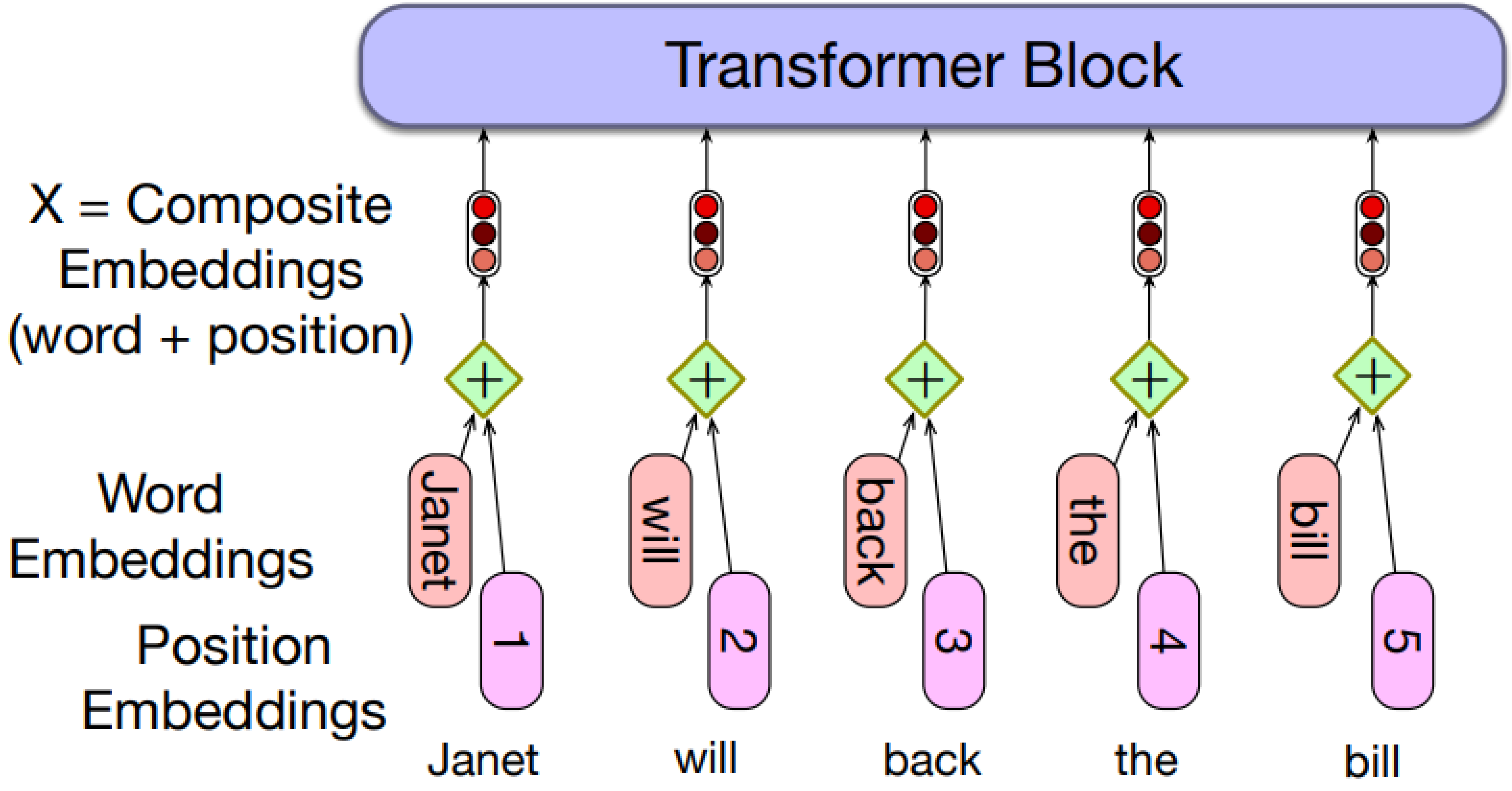
- What if there are two sentences?
  - "*a dog bites a man*"
  - "*a man bites a dog*"
- The order does not affect the output vector of each word!
- But language is order-sensitive!

- Solution: Add a learnable position encoding vector $\boldsymbol{p}_i$

input	$\boldsymbol{e}_{\text{a}} + \boldsymbol{p}_1$	$\boldsymbol{e}_{\text{dog}} + \boldsymbol{p}_2$	$\boldsymbol{e}_{\text{bites}} + \boldsymbol{p}_3$	$\boldsymbol{e}_{\text{a}} + \boldsymbol{p}_4$	$\boldsymbol{e}_{\text{man}} + \boldsymbol{p}_5$
word	*a*	*dog*	*bites*	*a*	*man*

input	$\boldsymbol{e}_{\text{a}} + \boldsymbol{p}_1$	$\boldsymbol{e}_{\text{man}} + \boldsymbol{p}_2$	$\boldsymbol{e}_{\text{bites}} + \boldsymbol{p}_3$	$\boldsymbol{e}_{\text{a}} + \boldsymbol{p}_4$	$\boldsymbol{e}_{\text{dog}} + \boldsymbol{p}_5$
word	*a*	*man*	*bites*	*a*	*dog*

Position Encoding: Visualization



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

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*, ..., *a*, *b*, *c*, *d*, ...}
- 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*" → "*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

Source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

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

vocabulary

␣, e, n, r, s, t, w

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, 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

vocabulary

␣, e, n, r, s, t, w, ne

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, 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

vocabulary

␣, e, n, r, s, t, w, ne, new

corpus

2	␣ new
2	␣re new
1	s e t
1	␣re s e t

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:

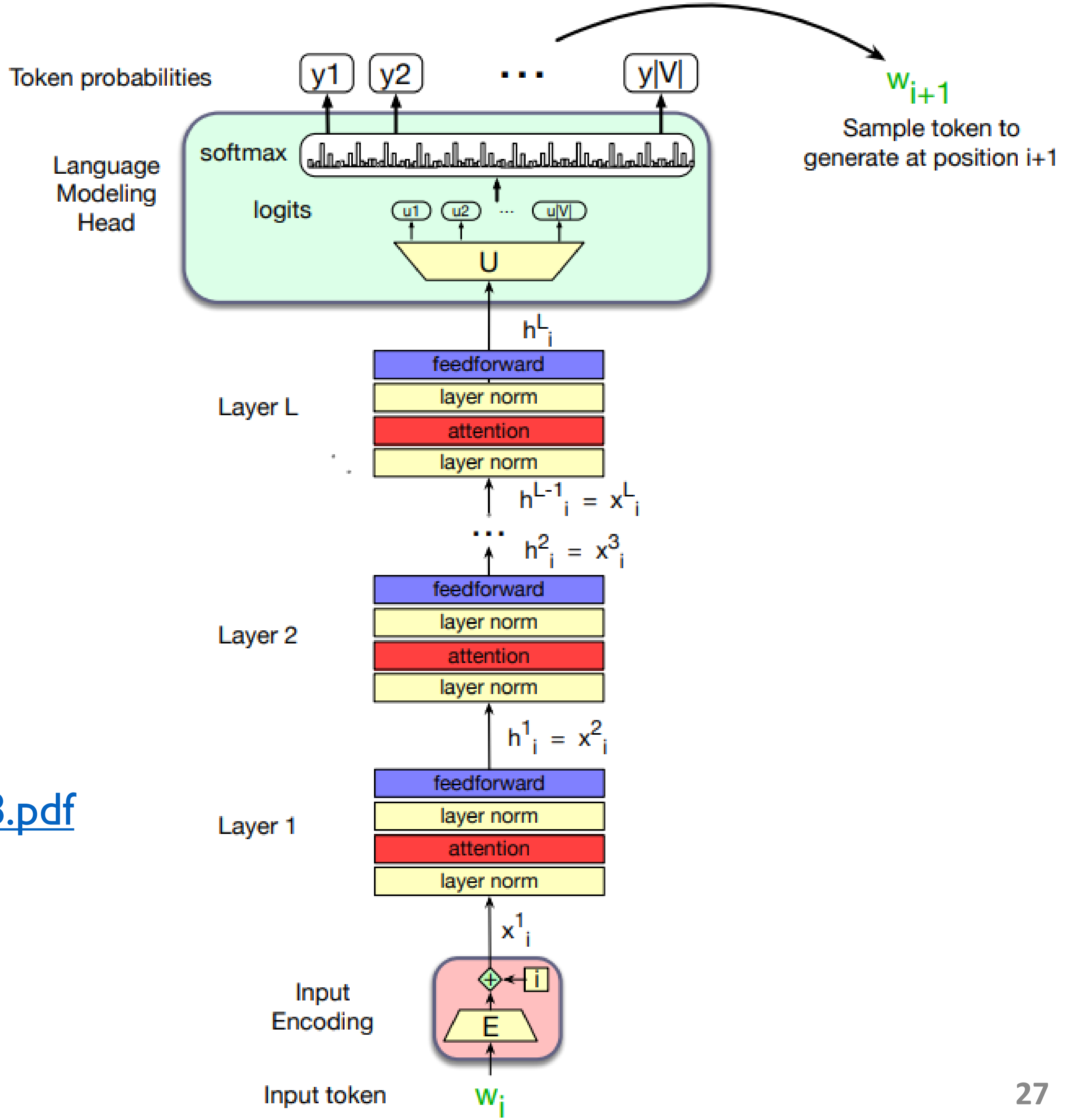
merge	**current vocabulary**
(␣, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new
(␣re, new)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew
(s, e)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se
(se, t)	␣, e, n, r, s, t, w, ne, new, ␣r, ␣re, ␣new, ␣renew, se, set

- 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
- …

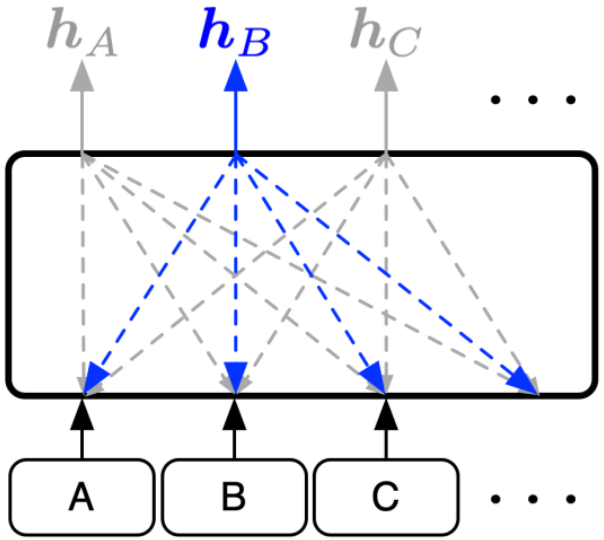
- Encoder-based language models (e.g., BERT) – predict a token from all other tokens in the input sequence

  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 …

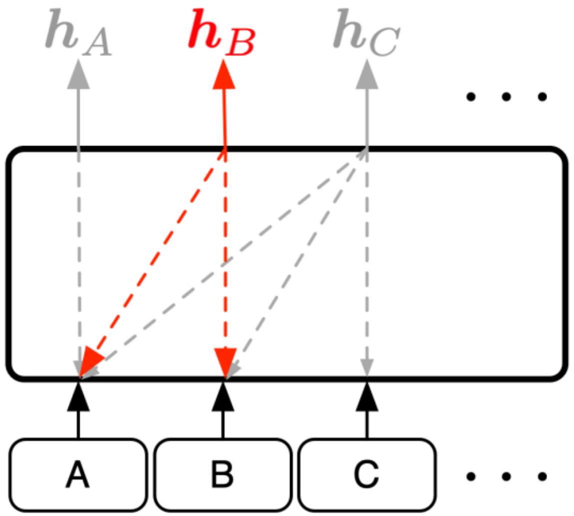
Two Types of Transformer Architecture



every token attends to all tokens

Bidirectional Self-Attention

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4



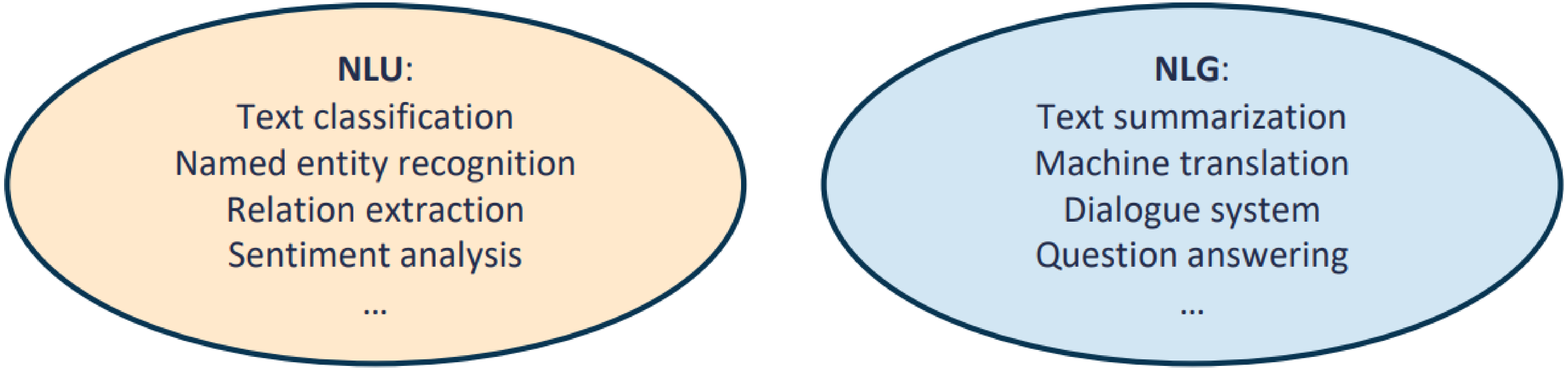
every token attends to its previous tokens

Unidirectional Self-Attention

q1·k1	$-\infty$	$-\infty$	$-\infty$
q2·k1	q2·k2	$-\infty$	$-\infty$
q3·k1	q3·k2	q3·k3	$-\infty$
q4·k1	q4·k2	q4·k3	q4·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

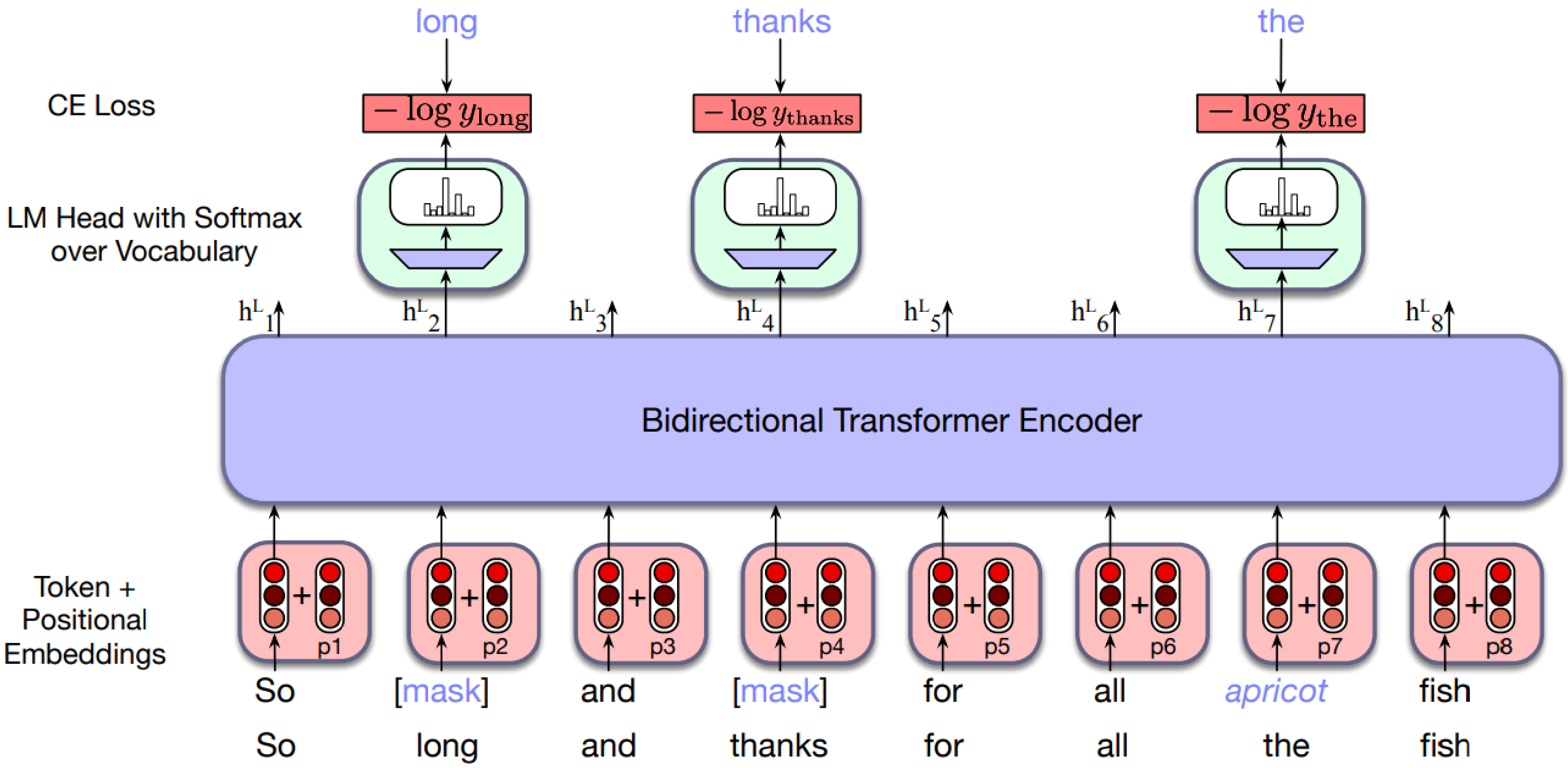
We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both

There are two existin
ing pre-trained language r
stream tasks: *feature-base*
feature-based approach,
et al., 2018a), uses task-sp
include the pre-trained r
tional features. The fine-tuning 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** obj
using exactly the same **pretraining** … No NSP: A **bidirectional** m
is trained using the "masked LM" (MLM) …

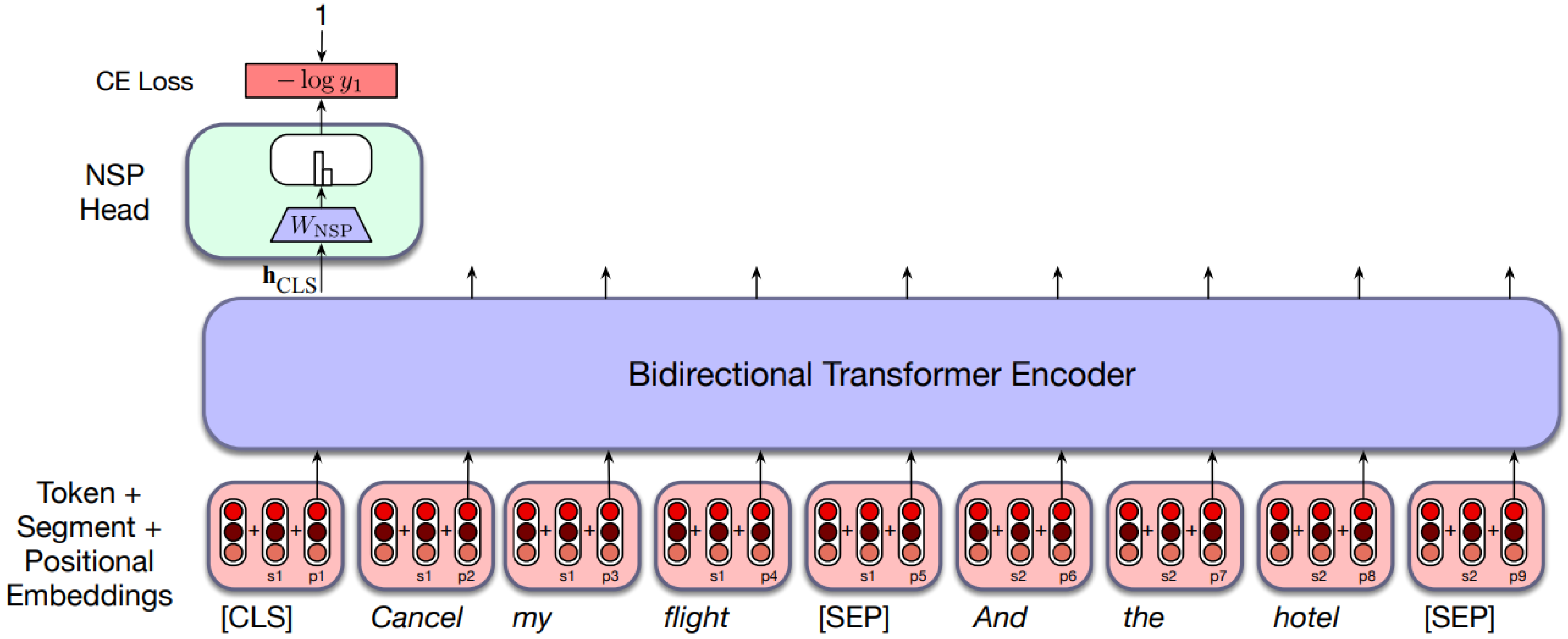
BERT Pre-training

- 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.



Source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**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)

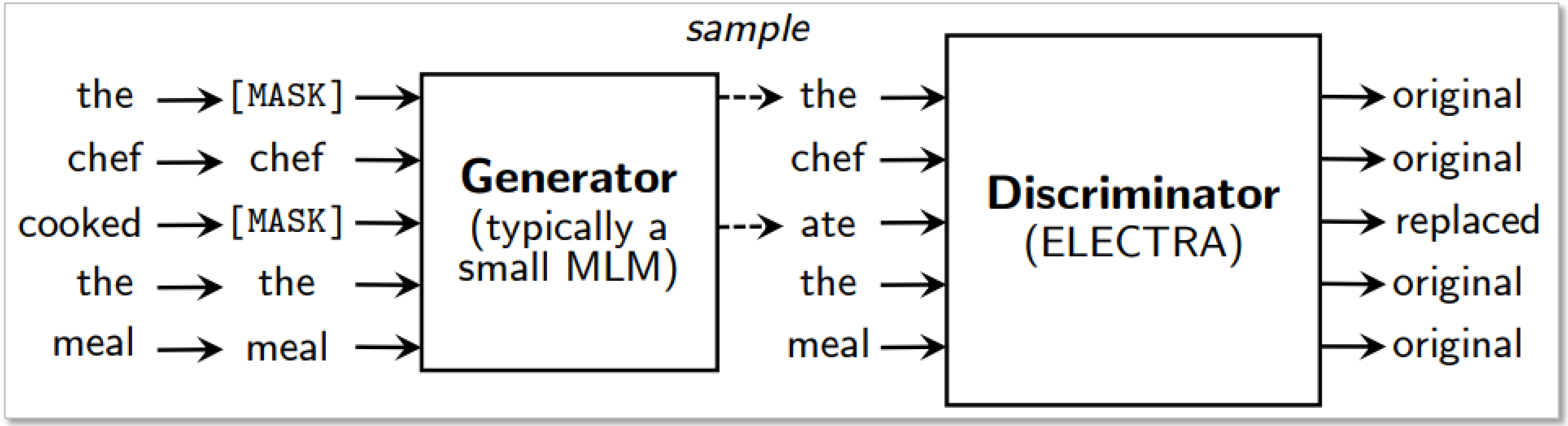
Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6

- 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	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**

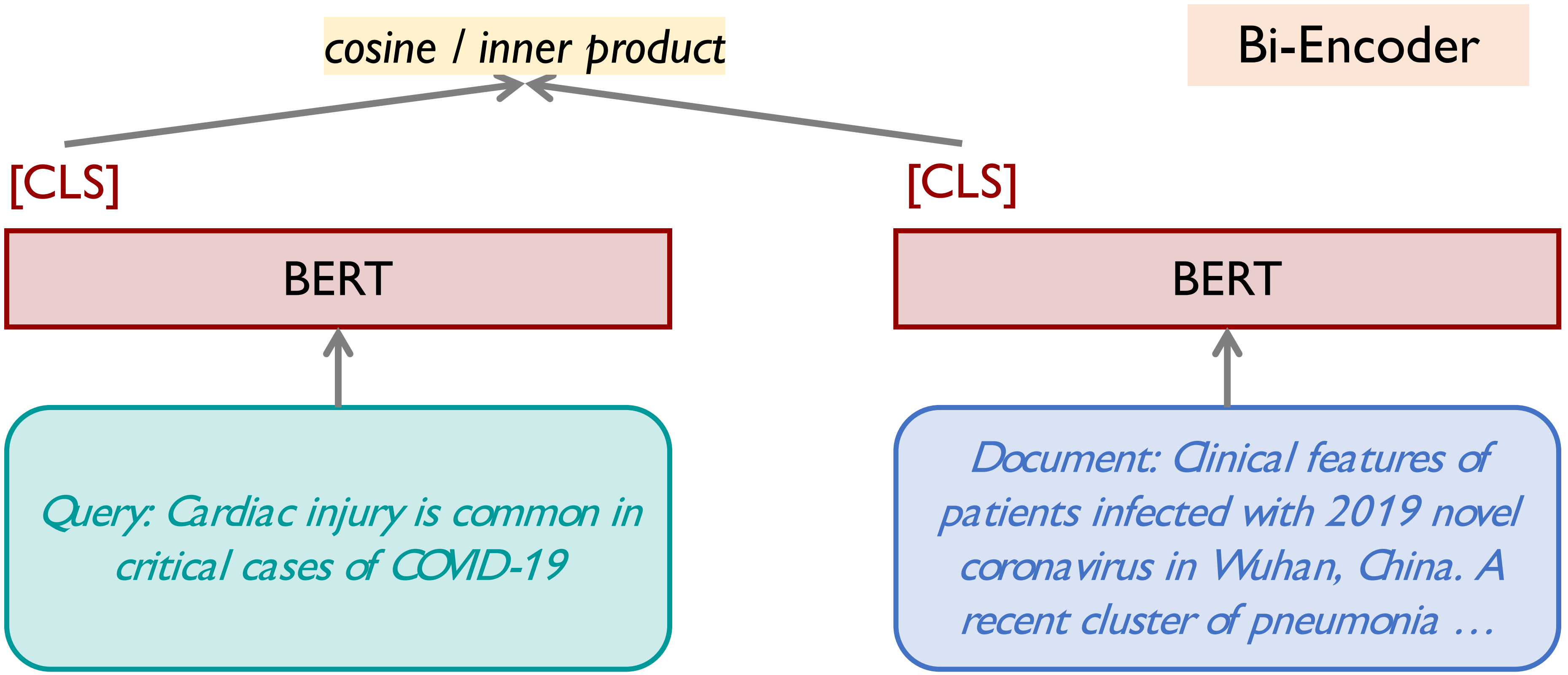
Improving BERT: ELECTRA [Clark et al., ICLR 2020]

- 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

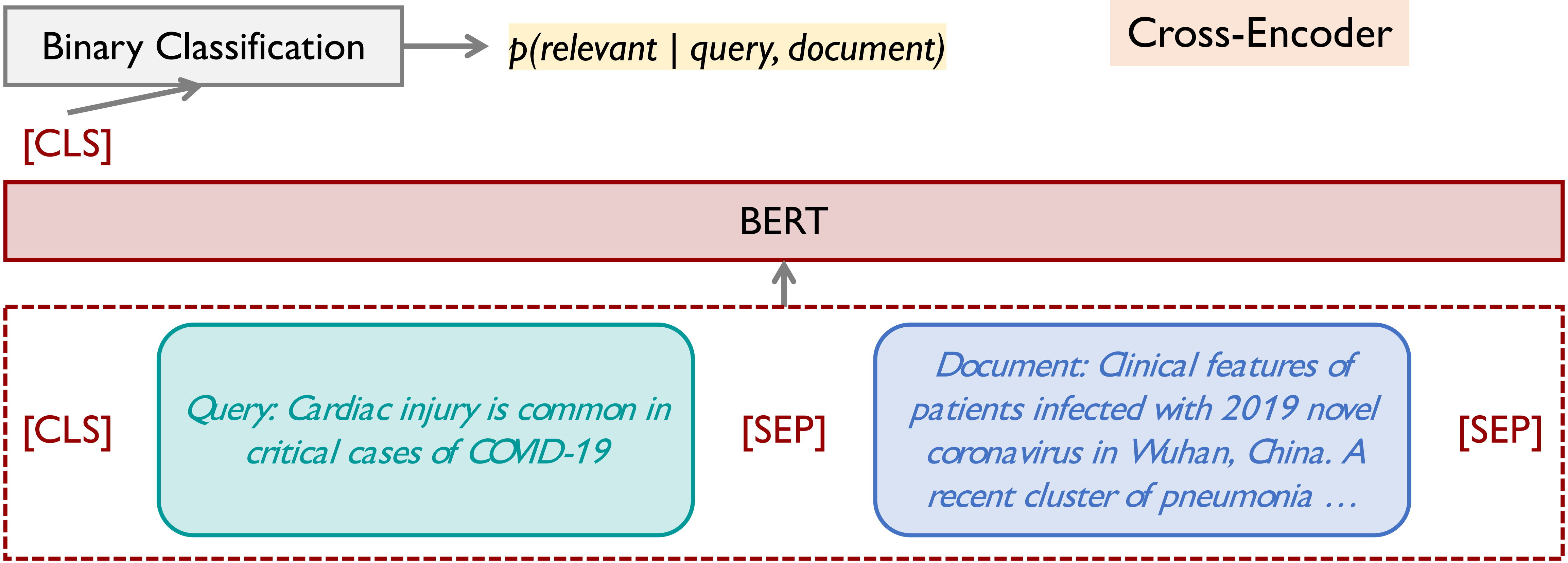
python

Copy code

```python
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