

## 05 - Deep Learning Architectures for Language Model-(I)(II)



Incheon Paik  
Intelligent Data Analytics Lab.  
University of Aizu

# Contents

---

- ◆ Language Model & Evaluation
- ◆ What is attention
- ◆ Transformer & Attention
- ◆ BERT and GPT Architecture
- ◆ GPT-2 Architecture
- ◆ GPT-2 Application
- ◆ A Short Explanation on GPT-3

# How to describe knowledge!

- ◆ How to describe knowledge in language?
- ◆ Let's think about the concept.

“ I am a student. ”

- 私は学生です。
- 나는 학생입니다.
- I am a student.
- 我是学生。

- ✓ Subject
- ✓ Predicate
- ✓ Object
- Basic Unit: Term

# Language Model

---

## ◆ Statistics Model

- "*Please turn your homework ...*" What do you think the next word? in or over?, or refrigerator or the?
- Models that assign a probability to each possible next word.

## ◆ Probability of sequence of words

- Which has higher probability?
  - "All of a sudden I notice three guys standing on the sidewalk"
  - "On guys all I of notice sidewalk three a sudden standing the"

## ◆ Language Model

- Models that assign probabilities to sequences of words

# Language Model

## ◆ Translation Example

他 向 记者 介绍了 主要 内容  
He to reporters introduced main content

As part of the process we might have built the following set of potential rough English translations:

he introduced reporters to the main contents of the statement

he briefed to reporters the main contents of the statement

**he briefed reporters on the main contents of the statement**

- ◆ A probabilistic model of word sequences could suggest that “briefed reporters on” is a more probable English phrase than the others
- ◆ The simplest model that assigns probabilities to sentences and sequences of words, the N-gram.
  - 2-gram (“please turn”), 3-gram (“please turn you”), .. N-gram

# N-Gram

- ◆ Computing  $P(w|h)$ , the probability of a word  $w$  given some history  $h$

$P(\text{the}|\text{its water is so transparent that}).$

From relative frequency count..

$$P(\text{the}|\text{its water is so transparent that}) = \frac{C(\text{its water is so transparent that the})}{C(\text{its water is so transparent that})}$$

Decomposing the probability using the chain rule of probability

$$\begin{aligned} P(X_1 \dots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_1^2) \dots P(X_n|X_1^{n-1}) \\ &= \prod_{k=1}^n P(X_k|X_1^{k-1}) \end{aligned}$$

Applying the chain rule to words, we get

$$\begin{aligned} P(w_1^n) &= P(w_1)P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1}) \\ &= \prod_{k=1}^n P(w_k|w_1^{k-1}) \end{aligned}$$

# N-Gram

The **bigram** model, for example, approximates the probability of a word given all the previous words  $P(w_n|w_1^{n-1})$  by using only the conditional probability of the preceding word  $P(w_n|w_{n-1})$ . In other words, instead of computing the probability

$$P(\text{the}|\text{Walden Pond's water is so transparent that})$$

we approximate it with the probability

$$P(\text{the}| \text{that})$$

When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

## Markov assumption and Generalization to N-gram

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

By given the bigram assumption for the probability of an individual word

$$P(w_1^n) \approx \prod_{k=1}^n P(w_k|w_{k-1})$$

# N-Gram

- ◆ Estimating the bigram or N-gram probabilities using Maximum Likelihood Estimation (MLE) and Normalization

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

We can simplify this equation, since the sum of all bigram counts that start with a given word  $w_{n-1}$  must be equal to the unigram count for that word  $w_{n-1}$  (the reader should take a moment to be convinced of this):

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

- ◆ Example

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

Here are the calculations for some of the bigram probabilities from this corpus

$$\begin{aligned} P(\text{I}|\text{<s>}) &= \frac{2}{3} = .67 & P(\text{Sam}|\text{<s>}) &= \frac{1}{3} = .33 & P(\text{am}|\text{I}) &= \frac{2}{3} = .67 \\ P(\text{</s>}|\text{Sam}) &= \frac{1}{2} = 0.5 & P(\text{Sam}|\text{am}) &= \frac{1}{2} = .5 & P(\text{do}|\text{I}) &= \frac{1}{3} = .33 \end{aligned}$$

For the general case of MLE N-gram parameter estimation:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

# Language Model Example

	<b>i</b>	<b>want</b>	<b>to</b>	<b>eat</b>	<b>chinese</b>	<b>food</b>	<b>lunch</b>	<b>spend</b>
<b>i</b>	5	827	0	9	0	0	0	2
<b>want</b>	2	0	608	1	6	6	5	1
<b>to</b>	2	0	4	686	2	0	6	211
<b>eat</b>	0	0	2	0	16	2	42	0
<b>chinese</b>	1	0	0	0	0	82	1	0
<b>food</b>	15	0	15	0	1	4	0	0
<b>lunch</b>	2	0	0	0	0	1	0	0
<b>spend</b>	1	0	1	0	0	0	0	0

**Figure 4.1** Bigram counts for eight of the words (out of  $V = 1446$ ) in the Berkeley Restaurant Project corpus of 9332 sentences. Zero counts are in gray.

	<b>i</b>	<b>want</b>	<b>to</b>	<b>eat</b>	<b>chinese</b>	<b>food</b>	<b>lunch</b>	<b>spend</b>
<b>i</b>	0.002	0.33	0	0.0036	0	0	0	0.00079
<b>want</b>	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
<b>to</b>	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
<b>eat</b>	0	0	0.0027	0	0.021	0.0027	0.056	0
<b>chinese</b>	0.0063	0	0	0	0	0.52	0.0063	0
<b>food</b>	0.014	0	0.014	0	0.00092	0.0037	0	0
<b>lunch</b>	0.0059	0	0	0	0	0.0029	0	0
<b>spend</b>	0.0036	0	0.0036	0	0	0	0	0

**Figure 4.2** Bigram probabilities for eight words in the Berkeley Restaurant Project corpus of 9332 sentences. Zero probabilities are in gray.

$$\begin{aligned} P(i | \langle s \rangle) &= 0.25 & P(\text{english} | \text{want}) &= 0.0011 \\ P(\text{food} | \text{english}) &= 0.5 & P(\langle /s \rangle | \text{food}) &= 0.68 \end{aligned}$$

Now we can compute the probability of sentences like *I want English food* or *I want Chinese food* by simply multiplying the appropriate bigram probabilities together, as follows:

$$\begin{aligned} P(\langle s \rangle \text{ i want english food } \langle /s \rangle) \\ &= P(i | \langle s \rangle)P(\text{want} | i)P(\text{english} | \text{want}) \\ &\quad P(\text{food} | \text{english})P(\langle /s \rangle | \text{food}) \\ &= .25 \times .33 \times .0011 \times 0.5 \times 0.68 \\ &= .000031 \end{aligned}$$

# Language Model Evaluation

- ◆ **Extrinsic evaluation:** to embed the model in an application and measure how much the application improves.
- ◆ **Intrinsic evaluation:** one that measures the quality of a model independent of any application.
- ◆ **Perplexity**

- The perplexity (shortly PP) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set  $W = w_1 w_2 \dots w_N$ :

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}} \end{aligned}$$

We can use the chain rule to expand the probability of  $W$ :

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_1 \dots w_{i-1})}}$$

Thus, if we are computing the perplexity of  $W$  with a bigram language model, we get:

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i | w_{i-1})}}$$

# Language Model Evaluation

- ◆ Weighted Average Branching Factor of a language
  - The number of possible next words that can follow any word
- ◆ Consider the task of recognizing the digits in English (zero, one, two, ..., nine), given that each of the 10 digits occurs with equal probability  $P = 1/10$ . The perplexity of this mini-language is in fact 10. The PP will be

$$\begin{aligned} \text{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\ &= \left(\frac{1}{10}\right)^{-\frac{1}{N}} \\ &= \frac{1}{10}^{-1} \\ &= 10 \end{aligned}$$

We see in Section 4.7 that perplexity is also closely related to the information-theoretic notion of entropy.

Finally, let's look at an example of how perplexity can be used to compare different N-gram models. We trained unigram, bigram, and trigram grammars on 38 million words (including start-of-sentence tokens) from the *Wall Street Journal*, using a 19,979 word vocabulary. We then computed the perplexity of each of these models on a test set of 1.5 million words with Eq. 4.16. The table below shows the perplexity of a 1.5 million word WSJ test set according to each of these grammars.

	<b>Unigram</b>	<b>Bigram</b>	<b>Trigram</b>
<b>Perplexity</b>	962	170	109

# Language Model Evaluation – BLEU(bilingual evaluation underway)

## Interpretation

Trying to compare BLEU scores across different corpora and languages is strongly discouraged. Even comparing BLEU scores for the same corpus but with different numbers of reference translations can be highly misleading.

However, as a rough guideline, the following interpretation of BLEU scores (expressed as percentages rather than decimals) might be helpful.

BLEU Score	Interpretation
< 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

The following color gradient can be used as a general scale [interpretation of the BLEU score](#):

# Language Model Evaluation - BLEU

## ◆ BLEU Model Evaluation

Mathematically, the BLEU score is defined as:

$$\text{BLEU} = \underbrace{\min\left(1, \exp\left(1 - \frac{\text{reference-length}}{\text{output-length}}\right)\right)}_{\text{brevity penalty}} \underbrace{\left(\prod_{i=1}^4 precision_i\right)^{1/4}}_{\text{n-gram overlap}}$$

with

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m_{cand}^i, m_{ref}^i)}{w_t^i = \sum_{\text{snt}' \in \text{Cand-Corpus}} \sum_{i' \in \text{snt}'} m_{cand}^{i'}}$$

where

- $m_{cand}^i$  is the count of i-gram in candidate matching the reference translation
- $m_{ref}^i$  is the count of i-gram in the reference translation
- $w_t^i$  is the total number of i-grams in candidate translation

The formula consists of two parts: the brevity penalty and the n-gram overlap.

- **Brevity Penalty**

The brevity penalty penalizes generated translations that are too short compared to the closest reference length with an exponential decay. The brevity penalty compensates for the fact that the BLEU score has no **recall** term.

- **N-Gram Overlap**

The n-gram overlap counts how many unigrams, bigrams, trigrams, and four-grams ( $i=1,\dots,4$ ) match their n-gram counterpart in the reference translations. This term acts as a **precision** metric. Unigrams account for **adequacy** while longer n-grams account for **fluency** of the translation. To avoid overcounting, the n-gram counts are clipped to the maximal n-gram count occurring in the reference ( $m_{ref}^n$ ).

# Language Model Evaluation

## ◆ BLEU

**Reference:** the cat is on the mat

**Candidate:** the the the cat mat

The first step is to count the occurrences of each unigram in the reference and the candidate. Note that the BLEU metric is case-sensitive.

Unigram	$m_{cand}^i$	$m_{ref}^i$	$\min(m_{cand}^i, m_{ref}^i)$
the	3	2	2
cat	1	1	1
is	0	1	0
on	0	1	0
mat	1	1	1

The total number of unigrams in the candidate ( $w_t^1$ ) is 5, so  $precision_1 = (2 + 1 + 1)/5 = 0.8$ .

### Calculating the BLEU score

**Reference:** The NASA Opportunity rover is battling a massive dust storm on Mars .

**Candidate 1:** The Opportunity rover is combating a big sandstorm on Mars .

**Candidate 2:** A NASA rover is fighting a massive storm on Mars .

The above example consists of a single reference and two candidate translations. The sentences are tokenized prior to computing the BLEU score as depicted above; for example, the final period is counted as a separate token.

To compute the BLEU score for each translation, we compute the following statistics.

- **N-Gram Precisions**

The following table contains the n-gram precisions for both candidates.

- **Brevity-Penalty**

The brevity-penalty is the same for candidate 1 and candidate 2 since both sentences consist of 11 tokens.

- **BLEU-Score**

Note that at least one matching 4-gram is required to get a BLEU score > 0. Since candidate translation 1 has no matching 4-gram, it has a BLEU score of 0.

# Language Model Evaluation

## ◆ BLEU

Metric	Candidate 1	Candidate 2
$precision_1$ (1gram)	8/11	9/11
$precision_2$ (2gram)	4/10	5/10
$precision_3$ (3gram)	2/9	2/9
$precision_4$ (4gram)	0/8	1/8
Brevity-Penalty	0.83	0.83
BLEU-Score	0.0	0.27

### Properties

- **BLEU is a Corpus-based Metric**

The BLEU metric performs badly when used to evaluate individual sentences. For example, both example sentences get very low BLEU scores even though they capture most of the meaning. Because n-gram statistics for individual sentences are less meaningful, BLEU is by design a corpus-based metric; that is, statistics are accumulated over an entire corpus when computing the score. Note that the BLEU metric defined above cannot be factorized for individual sentences.

- **No distinction between content and function words**

The BLEU metric does not distinguish between content and function words, that is, a dropped function word like "a" gets the same penalty as if the name "NASA" were erroneously replaced with "ESA".

- **Not good at capturing meaning and grammaticality of a sentence**

The drop of a single word like "not" can change the polarity of a sentence. Also, taking only n-grams into account with  $n \leq 4$  ignores long-range dependencies and thus BLEU often imposes only a small penalty for ungrammatical sentences.

- **Normalization and Tokenization**

Prior to computing the BLEU score, both the reference and candidate translations are normalized and tokenized. The choice of normalization and tokenization steps significantly affect the final BLEU score.

# Laplace Smoothing

## ◆ Laplace (add-one) Smoothing

- Words that are in our vocabulary (they are not unknown words) but appear in a test set in an unseen context (for example, they appear after a word they never appeared after in training)?
- To keep a language model from assigning zero probability to these unseen events, we will have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called smoothing or discounting.
- For example

$$P(\text{happy} \mid \text{neg}) \approx \frac{\text{Count}(\text{happy}, \text{neg})}{\sum_{j=1}^{|V|} \text{Count}(w_j, \text{neg})} = 0,$$

where  $\text{Count}(\text{happy}, \text{neg}) = 0$

## ◆ A solution: Add one to each count

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Here, word  $w_i$ , its count  $c_i$ , total number of word tokens  $N$ ,  $V$  words in the vocabulary

# Kneser-Ney Smoothing

- ◆ KN Discounting: Absolute discounting
- ◆ Basic Idea:
  - Predict the next word by interpolating Bigram + Unigram models
- ◆ Problem:
  - (Example) I can't see without my reading \_\_\_\_.
  - Standard unigram gives higher probability to frequent words (e.g., Kong) rather than contextually correct ones (glasses)
- ◆ New Approach:
  - Use  $P_{\text{continuation}}(w)$  instead of simple frequency  $P(w)$
- ◆ Core Concept:
  - $P_{\text{continuation}}(w)$  = the number of different contexts in which word  $w$  has appeared
- ◆ Intuitive Interpretation: glasses appears in many contexts, while Kong mainly appears in "Hong Kong"
- ◆ Conclusion: KN Smoothing adjusts probabilities based on "diversity of contexts" rather than raw frequency

# Kneser-Ney Smoothing

## ◆ KN Discounting: Absolute discounting

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}| \quad (4.29)$$

To turn this count into a probability, we normalize by the total number of word bigram types. In summary:

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|} \quad (4.30)$$

An alternative metaphor for an equivalent formulation is to use the number of word types seen to precede  $w$  (Eq. 4.29 repeated):

$$P_{\text{CONTINUATION}}(w) \propto |\{v : C(vw) > 0\}| \quad (4.31)$$

normalized by the number of words preceding all words, as follows:

$$P_{\text{CONTINUATION}}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|} \quad (4.32)$$

A frequent word (Kong) occurring in only one context (Hong) will have a low continuation probability.

The final equation for **Interpolated Kneser-Ney** smoothing for bigrams is then:

$$P_{\text{KN}}(w_i | w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\text{CONTINUATION}}(w_i) \quad (4.33)$$

The  $\lambda$  is a normalizing constant that is used to distribute the probability mass we've discounted.:

$$\lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}| \quad (4.34)$$

# Kneser-Ney Smoothing

## ◆ KN Discounting: Absolute discounting

$$(4.29) \quad P_{CONT}(w) \propto |\{v : C(vw) > 0\}|$$

Count how many different contexts (preceding words)  $w$  appears in.

$$(4.30) \quad P_{CONT}(w) = \frac{|\{v : C(vw) > 0\}|}{|\{(u', w') : C(u'w') > 0\}|}$$

Turn the count into probability by dividing by total bigram types.

$$(4.32) \quad P_{CONT}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|}$$

Normalize by total continuation counts across all words.

$$(4.33) \quad P_{KN}(w_i | w_{i-1}) = \frac{\max(C(w_{i-1}w_i) - d, 0)}{C(w_{i-1})} + \lambda(w_{i-1})P_{CONT}(w_i)$$

Final interpolated KN: discounted bigram + continuation probability.

$$(4.34) \quad \lambda(w_{i-1}) = \frac{d}{\sum_v C(w_{i-1}v)} |\{w : C(w_{i-1}w) > 0\}|$$

Normalization constant to distribute discounted probability mass.

# Kneser-Ney Smoothing

## Eq. 4.32 — $P_{\text{CONT}}(w)$

### Key Points

- $P_{\text{CONT}}(w) = \frac{|\{v : C(vw) > 0\}|}{\sum_{w'} |\{v : C(vw') > 0\}|}$
- Measures the proportion of **distinct preceding contexts** in which  $w$  appears (a normalized probability).
- Sums to 1 over all  $w \rightarrow$  usable as a **backoff distribution**.

### Meaning / Why It Matters

- Captures **contextual diversity** instead of raw token frequency  $P(w)$ .
- Words tied to a single fixed phrase (e.g., “Kong”  $\approx$  “Hong Kong”) get a **low** value.
- Prefers words that remain plausible in **new/unseen contexts**  $\rightarrow$  improves **generalization**.

### Example (reading \_)

- Distinct-preceder counts: **glasses**=10, **book**=4, **lamp**=2, **Kong**=1 (total = 17)
- $P_{\text{CONT}}(\text{glasses}) = 10/17 \approx 0.588$ ,  $\text{book} = 4/17 \approx 0.235$ ,  $\text{lamp} = 2/17 \approx 0.118$ ,  $\text{Kong} = 1/17 \approx 0.059$
- $\Rightarrow$  Signals that “**glasses**” is more natural in a new context.

# Kneser-Ney Smoothing

## Eq. 4.33 — Interpolated Kneser–Ney

### Key Points

- $P_{KN}(w_i | w_{i-1}) = \frac{\max\{C(w_{i-1}w_i) - d, 0\}}{C(w_{i-1})} + \lambda(w_{i-1}) P_{\text{CONT}}(w_i)$
- Two-term blend:
  1. Observed bigram (discounted) + 2) Continuation backoff (context diversity)
- $\lambda(w_{i-1}) = \frac{d \cdot N_1^+(w_{i-1}\cdot)}{C(w_{i-1})}$  where  $N_1^+(w_{i-1}\cdot)$  is the number of distinct followers after  $w_{i-1}$ .
- Goal: keep strong collocations while giving probability to rare/unseen pairs.

### Meaning / Why It Matters

- Using only  $P_{\text{CONT}}$  can miss very strong fixed pairs (e.g., "New York").
- Using only observed counts makes unseen pairs have probability 0.
- Eq. (4.33) mixes **evidence** (observed bigrams) with **generalization** (continuation), achieving both accuracy and robustness.

### Worked Example (context: "reading \_\_")

Assume:

- Counts:  $C(\text{reading, glasses}) = 5$ , book= 3, Kong= 1, lamp= 1  $\rightarrow C(\text{reading}) = 10$ , distinct followers  $N_1^+ = 4$
- Discount  $d = 0.75 \Rightarrow \lambda(\text{reading}) = \frac{0.75 \times 4}{10} = 0.30$
- Normalized  $P_{\text{CONT}}$ : glasses = 0.4615, book = 0.3077, Kong = 0.0769, lamp = 0.1538

# Kneser-Ney Smoothing

① Observed (discounted)  $\max(C - d, 0)/10$

- glasses:  $(5 - 0.75)/10 = 0.425$
- book:  $(3 - 0.75)/10 = 0.225$
- Kong:  $(1 - 0.75)/10 = 0.025$
- lamp:  $(1 - 0.75)/10 = 0.025$   
(sum = 0.70 → we'll return 0.30 via backoff)

② Backoff  $\lambda \cdot P_{\text{CONT}}$

- glasses:  $0.30 \times 0.4615 = 0.1385$
- book:  $0.30 \times 0.3077 = 0.0923$
- Kong:  $0.30 \times 0.0769 = 0.0231$
- lamp:  $0.30 \times 0.1538 = 0.0462$

Final  $P_{KN}(\cdot \mid \text{reading}) = ① + ②$

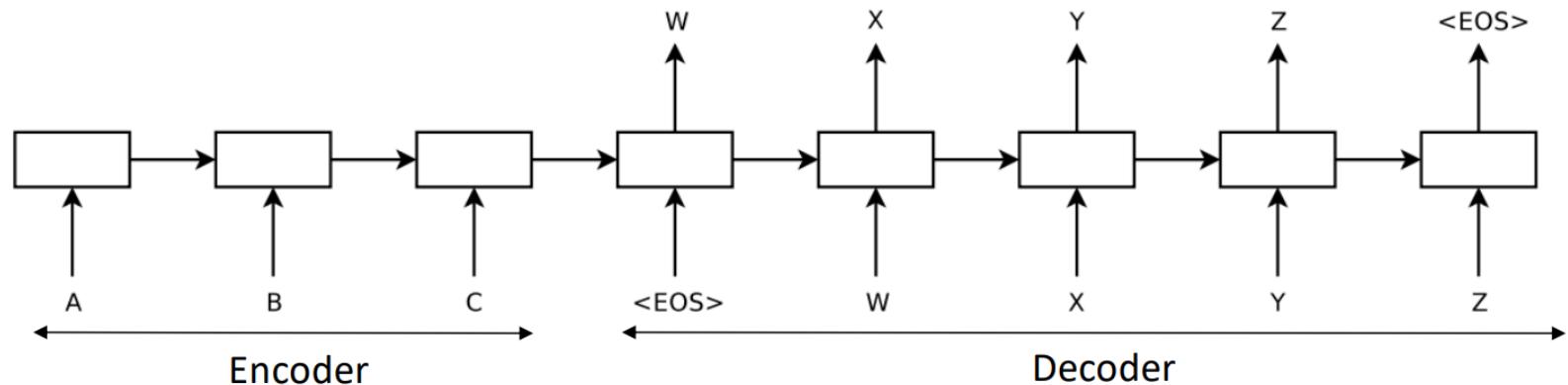
- glasses: 0.5635
- book: 0.3173
- lamp: 0.0712
- Kong: 0.0481

Read: “reading glasses” is strong from observation (①), and “glasses” also appears after many different words (②), so it remains the most natural choice—even in new contexts.

# What is Attention

## ◆ Seq2Seq Model

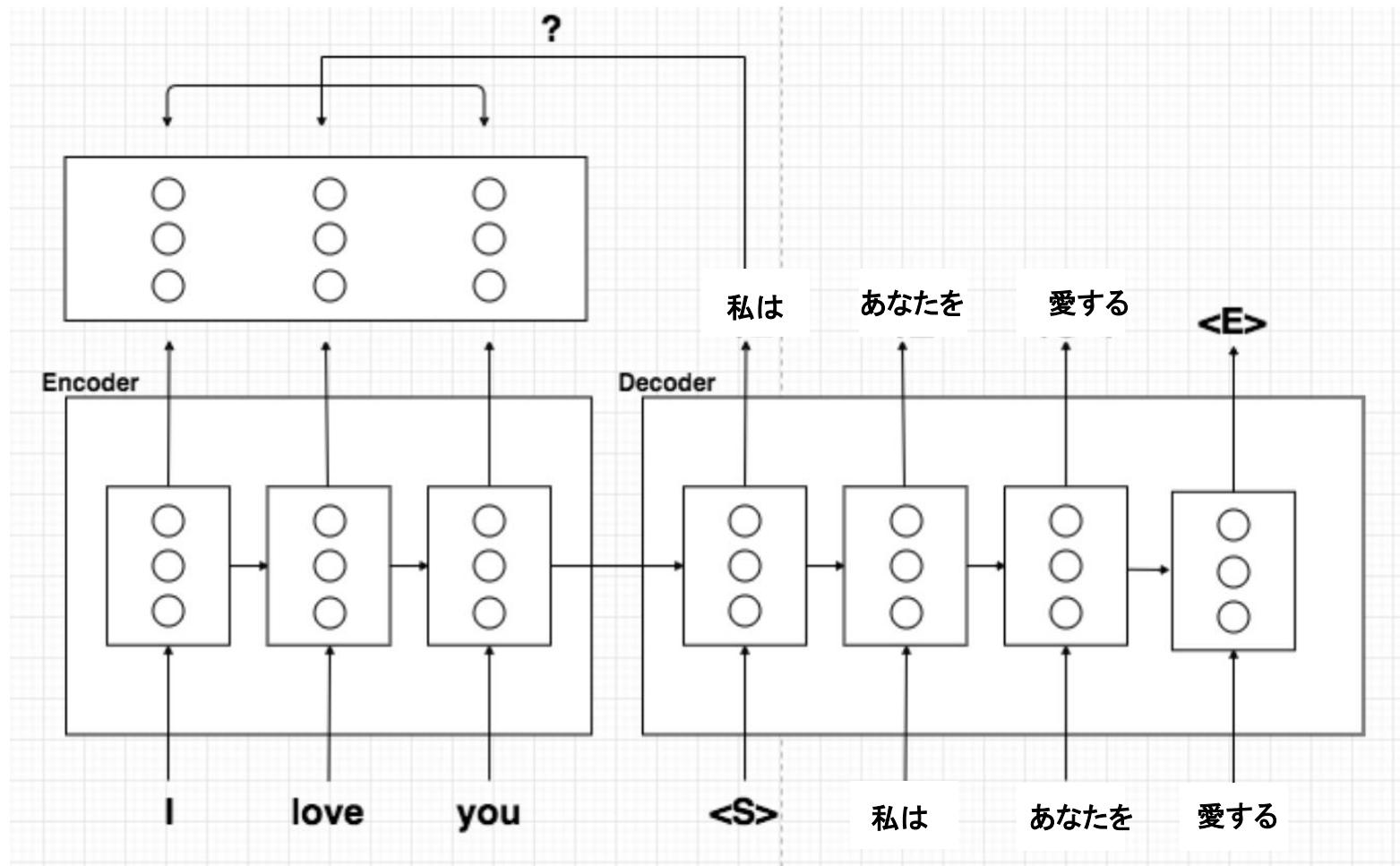
- A neural network that transforms a sequence of elements (words, etc.) to another sequence.
- Use: neural machine translation, chatbot, speech recognition, etc.
- The first seq2seq model was introduced in [Sutskever et al. \(2014\)](#) and purely used RNN.



*Note:* The model shown above uses only **2 RNN cells**. The rectangular boxes illustrate different **states** of unrolled RNN cells.

# What is Attention

- ◆ Attention in Seq2Seq Model: To overcome problem of information loss of long sequence



# What is Attention

## ◆ Attention in Seq2Seq Model

Output time sequence of Encoder

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

Output time sequence  
of Decoder

What is the most related  
between i and j

# Global Attention

## ◆ Global Attention Model

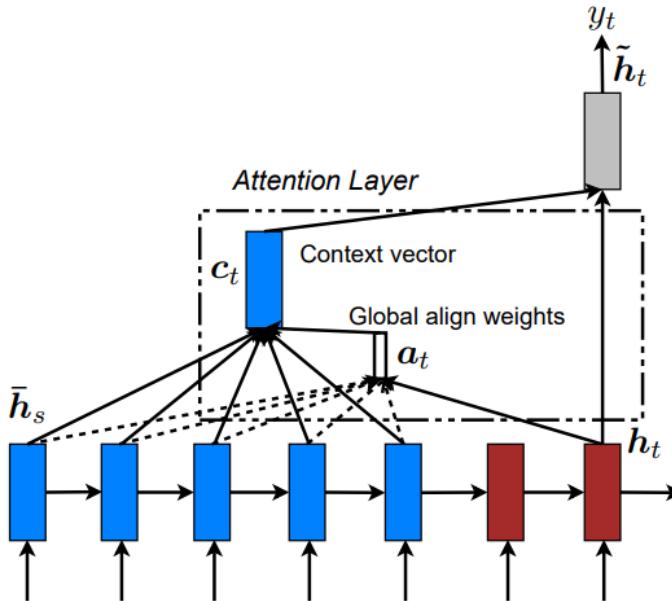


Figure 2: **Global attentional model** – at each time step  $t$ , the model infers a *variable-length* alignment weight vector  $\mathbf{a}_t$  based on the current target state  $\mathbf{h}_t$  and all source states  $\bar{\mathbf{h}}_s$ . A global context vector  $\mathbf{c}_t$  is then computed as the weighted average, according to  $\mathbf{a}_t$ , over all the source states.

Specifically, given the target hidden state  $\mathbf{h}_t$  and the source-side context vector  $\mathbf{c}_t$ , we employ a simple concatenation layer to combine the information from both vectors to produce an attentional hidden state as follows:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c[\mathbf{c}_t; \mathbf{h}_t]) \quad (5)$$

The attentional vector  $\tilde{\mathbf{h}}_t$  is then fed through the softmax layer to produce the predictive distribution formulated as:

$$p(y_t | y_{<t}, x) = \text{softmax}(\mathbf{W}_s \tilde{\mathbf{h}}_t) \quad (6)$$

The idea of a global attentional model is to consider all the hidden states of the encoder when deriving the context vector  $\mathbf{c}_t$ . In this model type, a variable-length alignment vector  $\mathbf{a}_t$ , whose size equals the number of time steps on the source side, is derived by comparing the current target hidden state  $\mathbf{h}_t$  with each source hidden state  $\bar{\mathbf{h}}_s$ :

$$\begin{aligned} \mathbf{a}_t(s) &= \text{align}(\mathbf{h}_t, \bar{\mathbf{h}}_s) \\ &= \frac{\exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s))}{\sum_{s'} \exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_{s'}))} \end{aligned} \quad (7)$$

Here, score is referred as a *content-based* function for which we consider three different alternatives:

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \begin{cases} \mathbf{h}_t^\top \bar{\mathbf{h}}_s & \text{dot} \\ \mathbf{h}_t^\top \mathbf{W}_a \bar{\mathbf{h}}_s & \text{general} \\ \mathbf{W}_a[\mathbf{h}_t; \bar{\mathbf{h}}_s] & \text{concat} \end{cases} \quad (8)$$

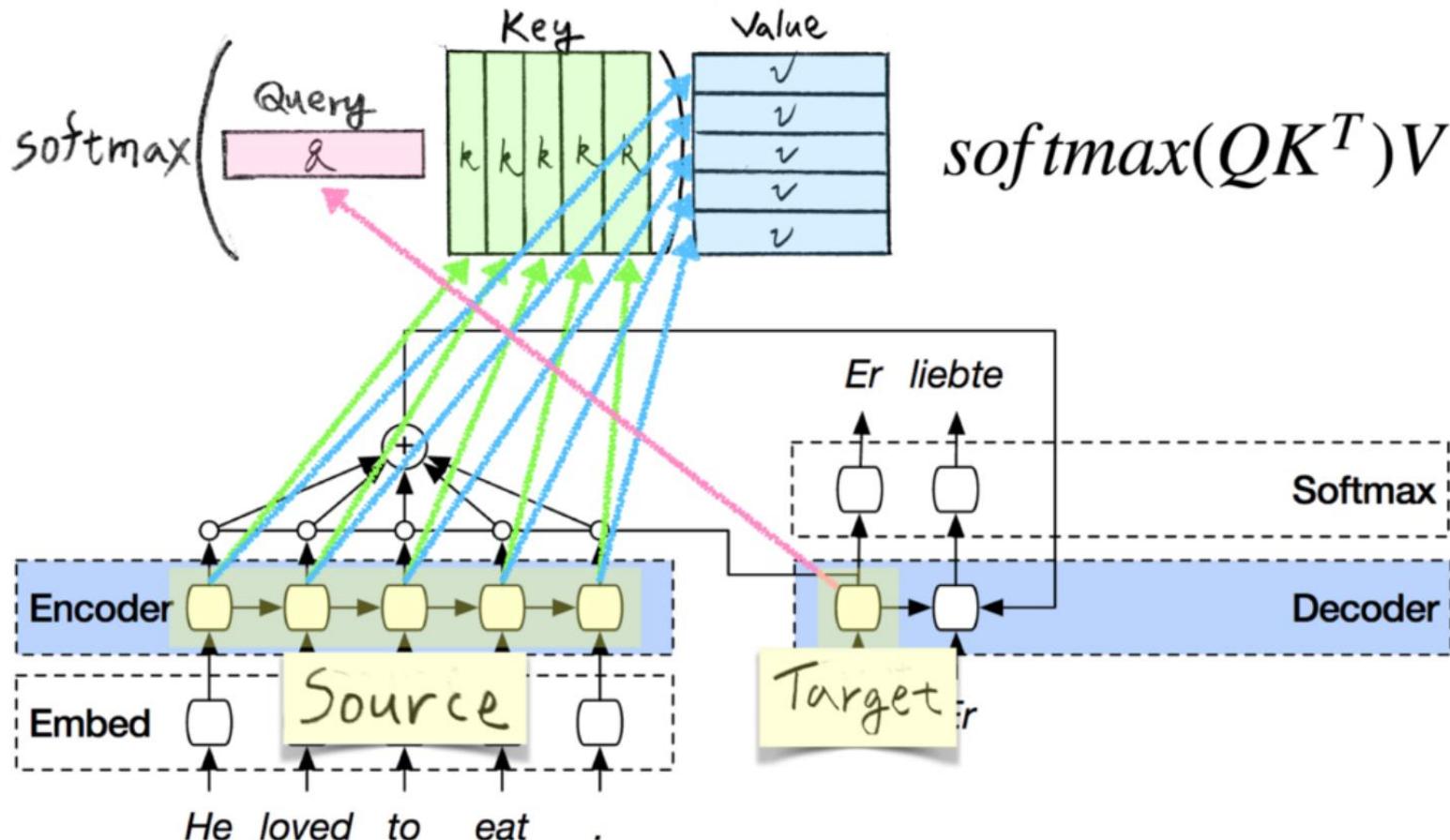
$$\mathbf{a}_t = \text{softmax}(\mathbf{W}_a \mathbf{h}_t) \quad \text{location} \quad (9)$$

# What is attention?

- ◆ Attention: Process of obtaining value of the key that is similar to a query
- ◆ Example
- ◆ A key-value set = {"computer": 9, "dog":2, "cat":3 }
- ◆ If a query = “puppy”, then value will be 2. puppy → 
- ◆ When there is a function “similarity()”, result of similarity will be
  - similarity("computer", query) = 0.1
  - similarity("dog", query) = 0.9
  - similarity("cat", query) = 0.7
- ◆ Therefore, key-value function of query is calculated as below
- ◆  $0.1/(0.9 + 0.7 + 0.1) * 9 + 0.9/(0.9 + 0.7 + 0.1) * 2 + 0.7/(0.9 + 0.7 + 0.1) * 3 = 2.823$
- ◆ It has likelihood of value of dictionary.
- ◆ The Q, K, V can be described as a vector.
- ◆ Attention :  $\text{softmax}(\mathbf{Q}\mathbf{K}^T)\mathbf{V}$

# Transformer

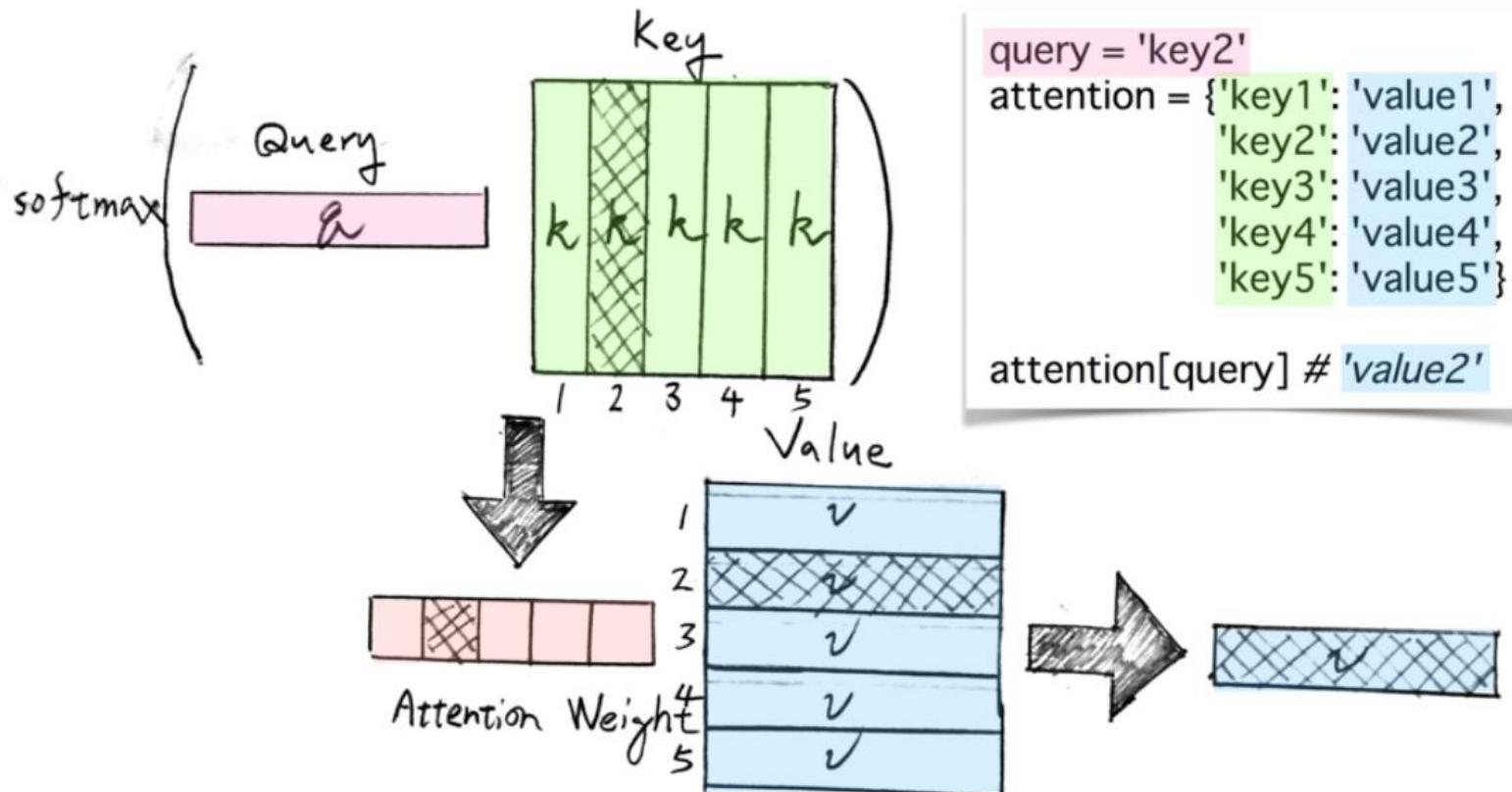
- ◆ Attention : softmax ( $QK^T$ )  $V$



# Transformer

注意は辞書オブジェクト  
(Attention is a Dictionary object)

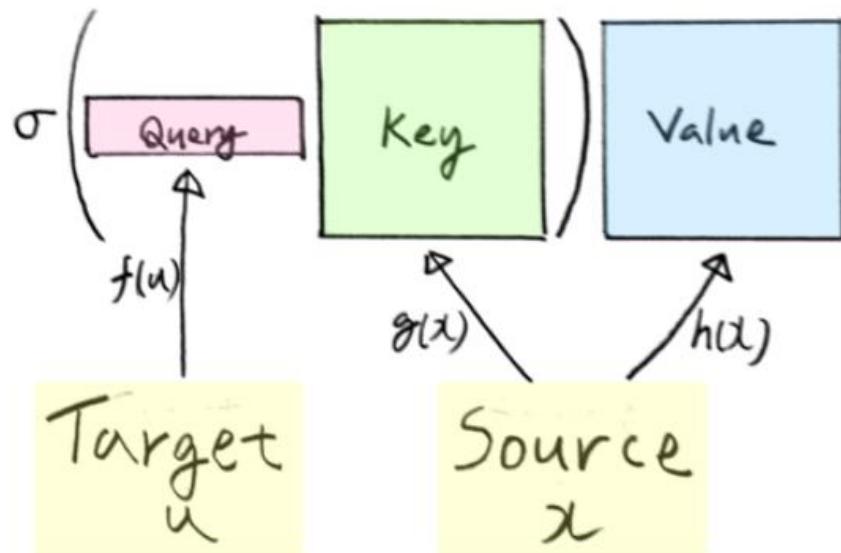
$$\text{softmax}(QK^T)V$$



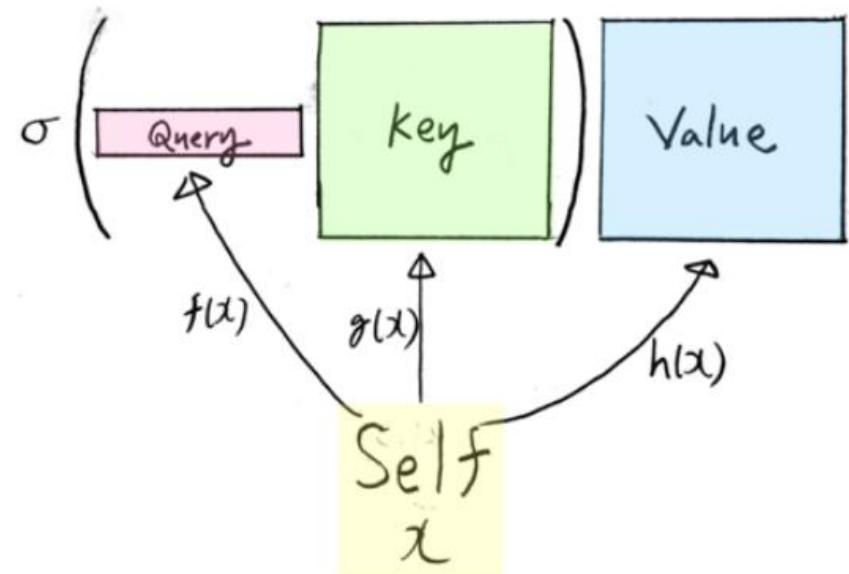
# Transformer

## ◆ Source-Target Attention and Self-Attention

ソース・ターゲット注意  
(Source-Target-Attention)

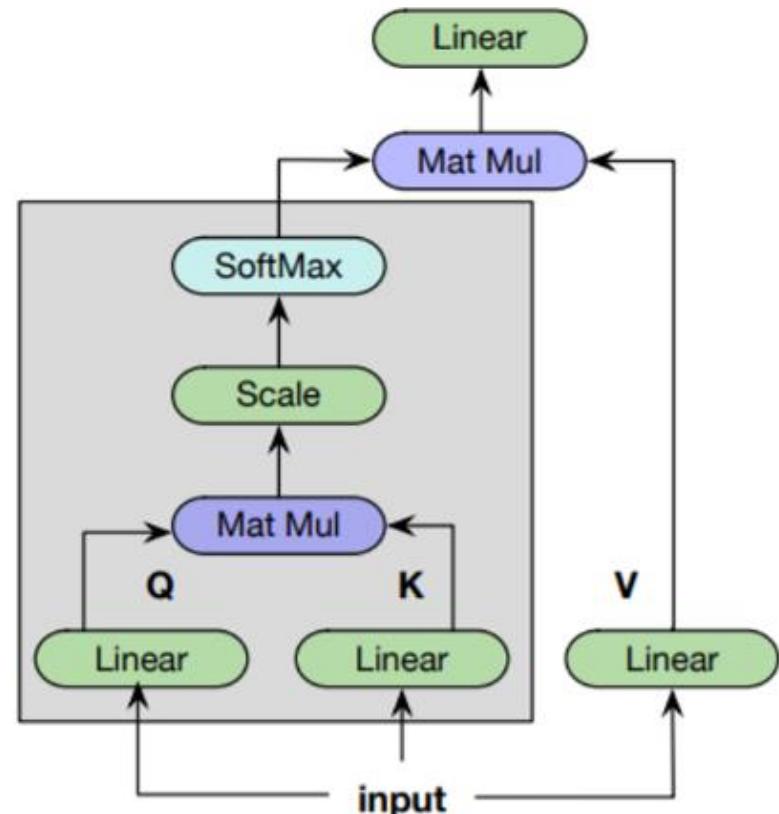
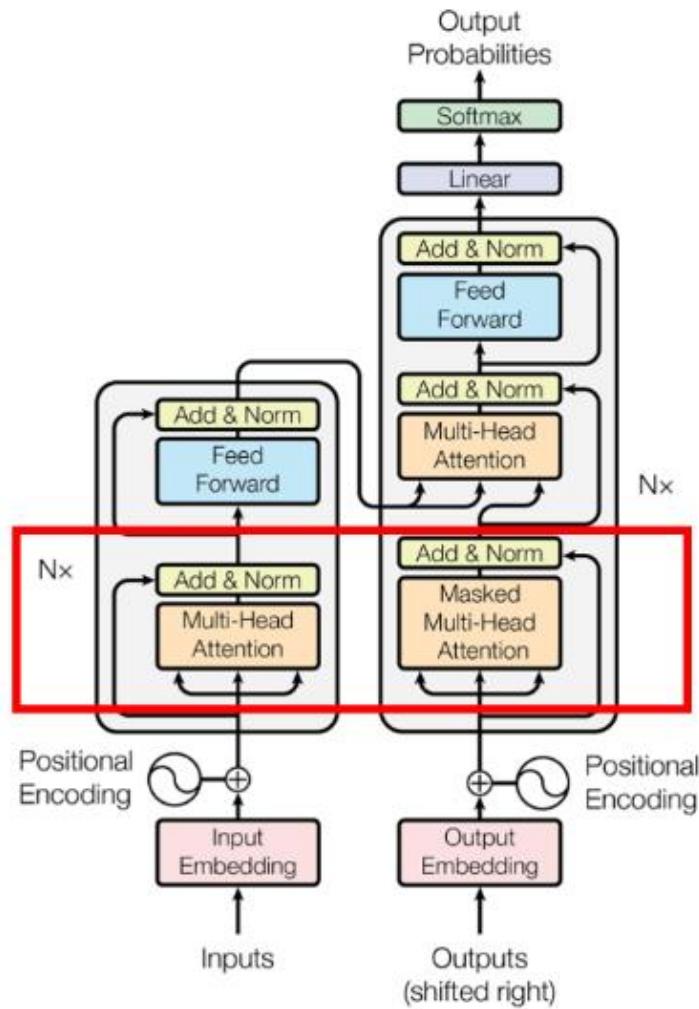


自己注意  
(Self-Attention)



# Transformer

## ◆ Self-Attention in Transformer



# Details of Self-Attention Calculation

- ◆ Making Query, Key, and Value

**(Expr-1)** Input Vector Sequence

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

**(Expr-2)** Making Query, Key, and Value

$$\mathbf{Q} = \mathbf{X} \times \mathbf{W}_Q$$

$$\mathbf{K} = \mathbf{X} \times \mathbf{W}_K$$

$$\mathbf{V} = \mathbf{X} \times \mathbf{W}_V$$

**(Expr-3)** Making Query (1)

$$[1 \ 0 \ 1 \ 0] \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [1 \ 0 \ 2]$$

# Details of Self-Attention Calculation

## (Expr-4) Making Query (2)

$$[0 \ 2 \ 0 \ 2] \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [2 \ 2 \ 2]$$

## (Expr-5) Making Query (3)

$$[1 \ 1 \ 1 \ 1] \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = [2 \ 1 \ 3]$$

## (Expr-6) Making Query (4)

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 2 & 2 \\ 2 & 1 & 3 \end{bmatrix}$$

# Details of Self-Attention Calculation

**(Expr-7)** Making Key

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 4 & 4 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

**(Expr-8)** Making Value

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 2 & 0 \\ 0 & 3 & 0 \\ 1 & 0 & 3 \\ 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix}$$

**(Expr-9)** Self-Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_K}}\right)\mathbf{V}$$

Reference: [https://ratsgo.github.io/nlpbook/docs/language\\_model/tr\\_self\\_attention/](https://ratsgo.github.io/nlpbook/docs/language_model/tr_self_attention/)

# Details of Self-Attention Calculation

**(Expr-10)** Calculating Self-Attention for the 1<sup>st</sup> Query Vector (1)

$$[1 \ 0 \ 2] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [2 \ 4 \ 4]$$

**(Expr-11)** Calculating Self-Attention for the 1<sup>st</sup> Query Vector (2)

$$\text{softmax}\left(\left[\frac{2}{\sqrt{3}}, \frac{4}{\sqrt{3}}, \frac{4}{\sqrt{3}}\right]\right) = [0.13613, 0.43194, 0.43194]$$

**(Expr-12)** Calculating Self-Attention for the 1<sup>st</sup> Query Vector (3)

$$[0.13163 \ 0.43194 \ 0.43194] \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = [1.8639 \ 6.3194 \ 1.7042]$$

# Details of Self-Attention Calculation

**(Expr-13~15)** Calculating Self-Attention for the 2<sup>nd</sup> Query Vector

$$[2 \ 2 \ 2] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [4 \ 16 \ 12] \quad \text{softmax}\left(\left[\frac{4}{\sqrt{3}}, \frac{16}{\sqrt{3}}, \frac{12}{\sqrt{3}}\right]\right) = [0.00089, 0.90884, 0.09027]$$

$$[0.00089 \ 0.90884 \ 0.09027] \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = [1.9991 \ 7.8141 \ 0.2735]$$

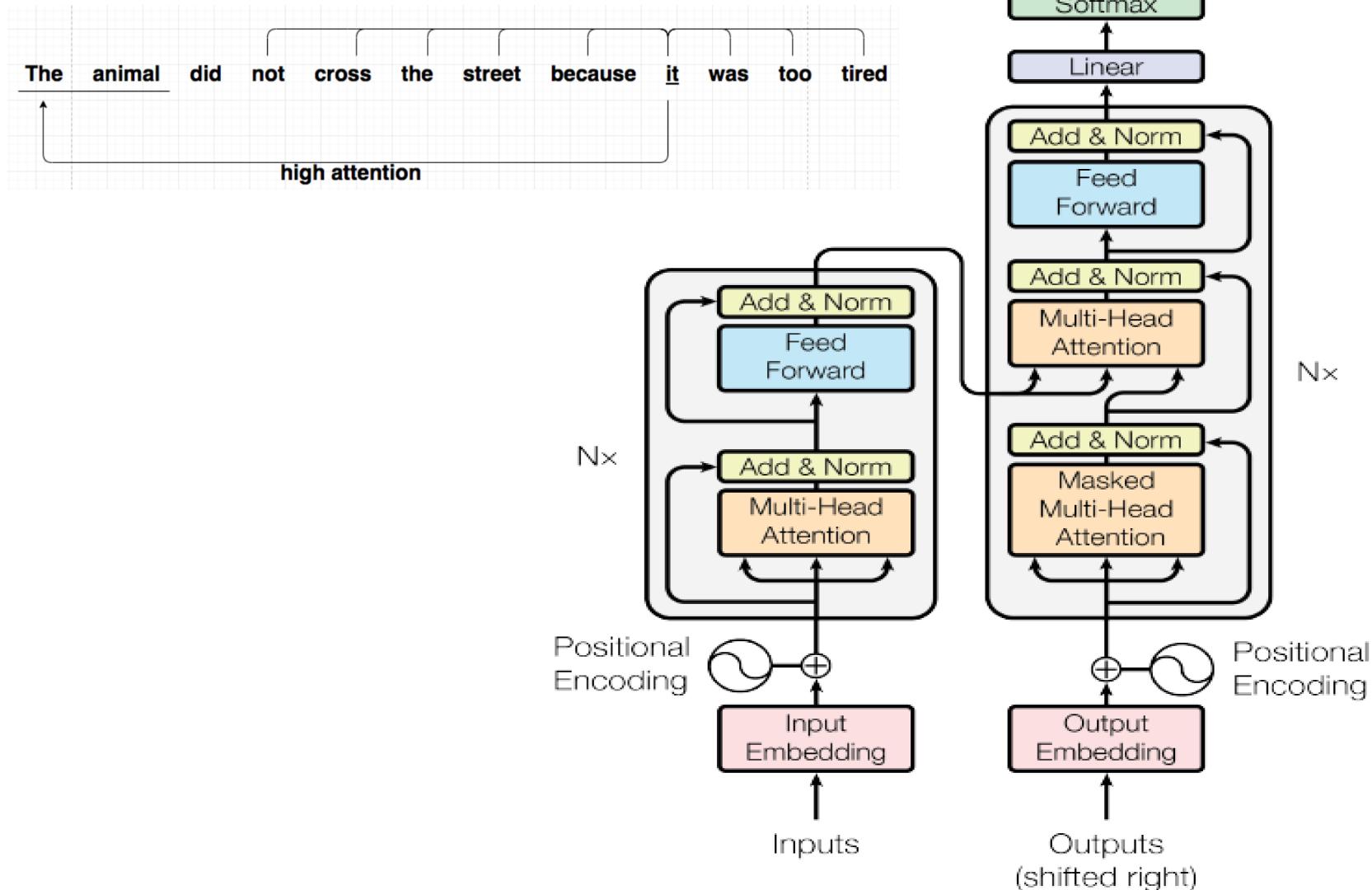
**(Expr-16~18)** Calculating Self-Attention for the 3<sup>rd</sup> Query Vector

$$[2 \ 1 \ 3] \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = [4 \ 12 \ 10] \quad \text{softmax}\left(\left[\frac{4}{\sqrt{3}}, \frac{12}{\sqrt{3}}, \frac{10}{\sqrt{3}}\right]\right) = [0.00744, 0.75471, 0.23785]$$

$$[0.00744 \ 0.75471 \ 0.23785] \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = [1.9926 \ 7.4796 \ 0.7359]$$

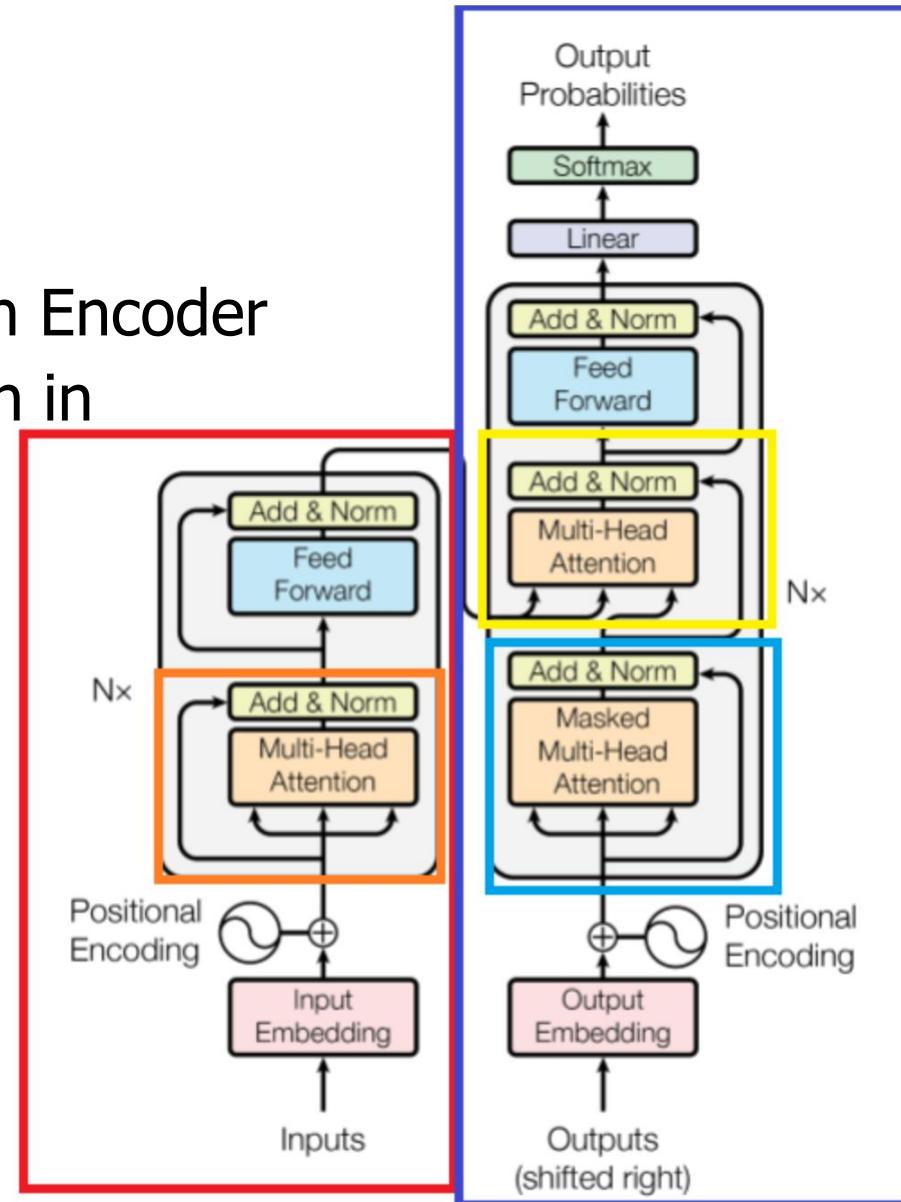
# Transformer

## ◆ Transformer “Attention is All!”



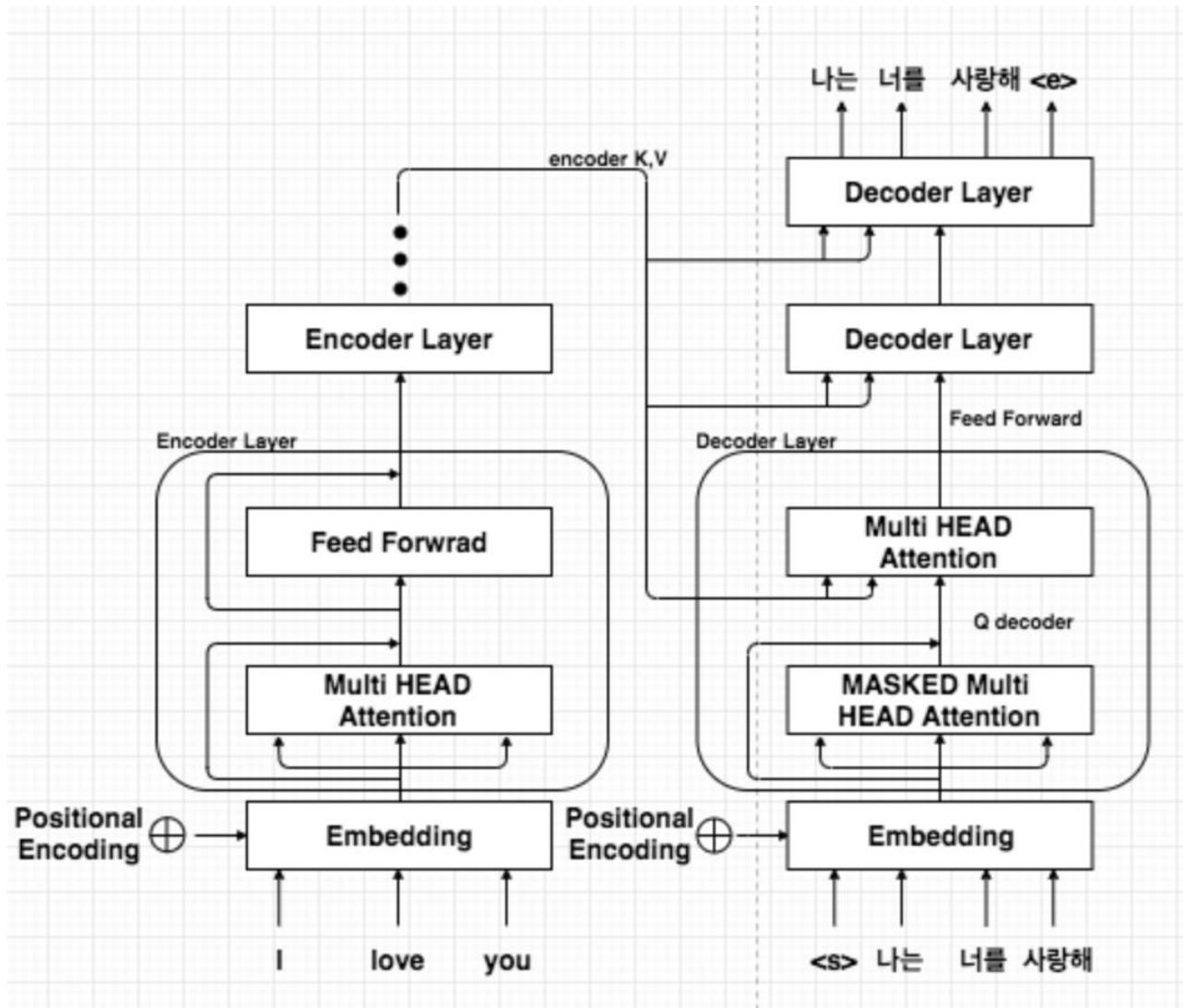
# Transformer

- ◆ Transformer
- ◆ Red: Encoder
- ◆ Blue: Decoder
- ◆ Orange: Self-Attention in Encoder
- ◆ Light Blue: Self-Attention in Decoder
- ◆ Yellow: Attention at Encoder and Decoder



# Transformer

## ◆ Learning in Transformer



# Transformer – Positional Encoding

## ◆ Positional Encoding

- ◆  $X_{\text{in}} = X_{\text{tok}} + P_{\text{pos}}$
- ◆ Relative distance matters (attention uses dot products)
- ◆ Smooth, multi-scale notion of position
- ◆ Generalization to longer sequences

### Classic (Sinusoidal) Positional Encoding

For model dimension  $d_{\text{model}}$ , position  $pos$ , channel index  $i$ :

$$\begin{aligned} \text{PE}(pos, 2i) &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \\ \text{PE}(pos, 2i+1) &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right). \end{aligned}$$

- Different **wavelengths** per dimension → captures both short- and long-range patterns.
- Inner products naturally reflect **relative distances** between positions.

### How it's used

- Elementwise addition with token embeddings (broadcast over batch/sequence):

$$X_{\text{in}} = X_{\text{embed}} + \text{PE}.$$

### Intuition

- **Multi-frequency:** small  $i \rightarrow$  long wavelengths (global), large  $i \rightarrow$  short wavelengths (local).
- **Smooth & periodic:** generalizes to sequence lengths beyond training via a closed-form function.

### Common Alternatives (one-liners)

- **Learned absolute PE:** parameters learned per position (simple; weaker extrapolation).
- **Relative PE** (e.g., RoPE, ALiBi): encodes pairwise **relative offsets**; scales better to long contexts.

### Mini Example

- $d_{\text{model}} = 4, pos = 7$ :

$$\text{PE}(7) = [\sin(7/10000^0), \cos(7/10000^0), \sin(7/10000^{1/2}), \cos(7/10000^{1/2})],$$

which is then added to the token's embedding for position 7.

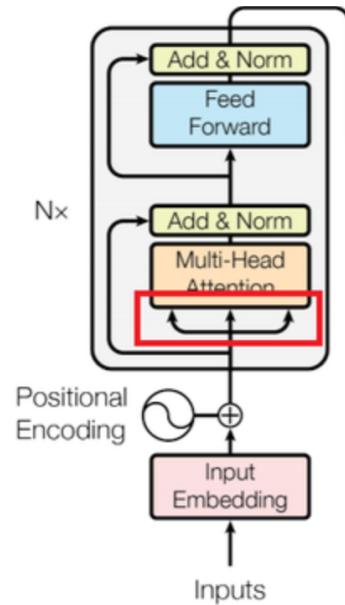
# Transformer

## ◆ Positional Encoding

```
1 def get_sinusoid_encoding_table(n_position, d_model):
2     def cal_angle(position, hid_idx):
3         return position / np.power(10000, 2 * (hid_idx // 2) / d_model)
4     def get_posi_angle_vec(position):
5         return [cal_angle(position, hid_j) for hid_j in range(d_model)]
6
7     sinusoid_table = np.array([get_posi_angle_vec(pos_i) for pos_i in range(n_position)])
8     sinusoid_table[:, 0::2] = np.sin(sinusoid_table[:, 0::2]) # dim 2i
9     sinusoid_table[:, 1::2] = np.cos(sinusoid_table[:, 1::2]) # dim 2i+1
10    return torch.FloatTensor(sinusoid_table)
```

get\_sinusoid\_encoding\_table.py hosted with ❤ by GitHub

[view raw](#)

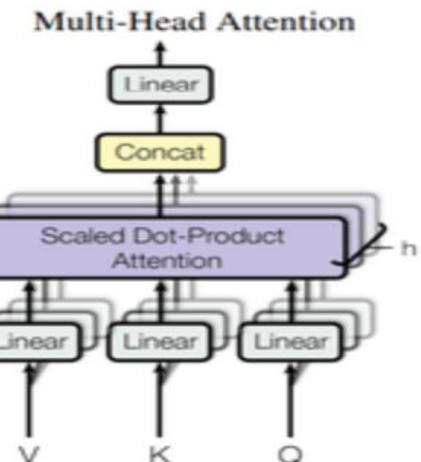


```
1 class EncoderLayer(nn.Module):
2     def __init__(self):
3         super(EncoderLayer, self).__init__()
4         self.enc_self_attn = MultiHeadAttention()
5         self.pos_ffn = PoswiseFeedForwardNet()
6     def forward(self, enc_inputs):
7         enc_outputs, attn = self.enc_self_attn(enc_inputs, enc_inputs, enc_inputs) # enc_inputs to sa
8         enc_outputs = self.pos_ffn(enc_outputs) # enc_outputs: [batch_size x len_q x d_model]
9         return enc_outputs, attn
```

# Transformer

## ◆ Multi-Head Attention

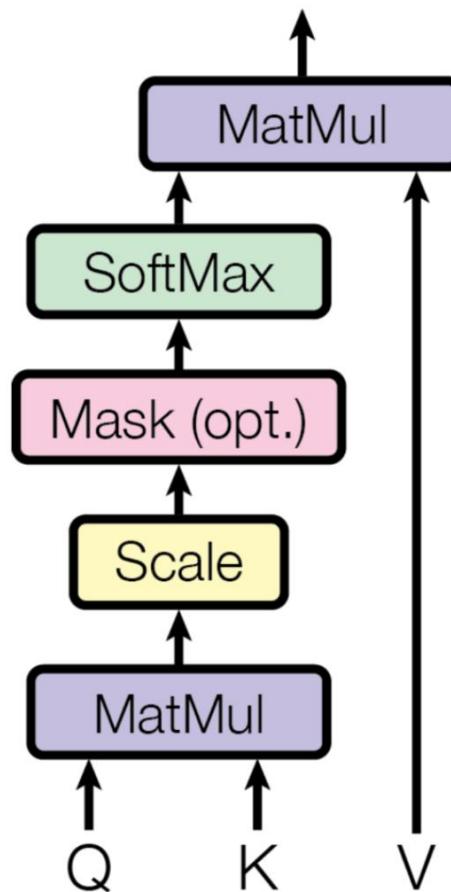
```
1  class MultiHeadAttention(nn.Module):
2      def __init__(self):
3          super(MultiHeadAttention, self).__init__()
4          self.W_Q = nn.Linear(d_model, d_k * n_heads)
5          self.W_K = nn.Linear(d_model, d_k * n_heads)
6          self.W_V = nn.Linear(d_model, d_v * n_heads)
7      def forward(self, Q, K, V, attn_mask):
8          # q: [batch_size x len_q x d_model], k: [batch_size x len_k x d_model], v: [batch_size x
9          residual, batch_size = Q, Q.size(0)
10         # (B, S, D) -proj-> (B, S, D) -split-> (B, S, H, W) -trans-> (B, H, S, W)
11         q_s = self.W_Q(Q).view(batch_size, -1, n_heads, d_k).transpose(1,2) # q_s: [batch_size
12         k_s = self.W_K(K).view(batch_size, -1, n_heads, d_k).transpose(1,2) # k_s: [batch_size
13         v_s = self.W_V(V).view(batch_size, -1, n_heads, d_v).transpose(1,2) # v_s: [batch_size
14
15         attn_mask = attn_mask.unsqueeze(1).repeat(1, n_heads, 1, 1) # attn_mask : [batch_size x
16
17         # context: [batch_size x n_heads x len_q x d_v], attn: [batch_size x n_heads x len_q(=le
18         context, attn = ScaledDotProductAttention()(q_s, k_s, v_s, attn_mask)
19         context = context.transpose(1, 2).contiguous().view(batch_size, -1, n_heads * d_v) # con
20         output = nn.Linear(n_heads * d_v, d_model)(context)
21
22         return nn.LayerNorm(d_model)(output + residual), attn # output: [batch_size x len_q x d_
```



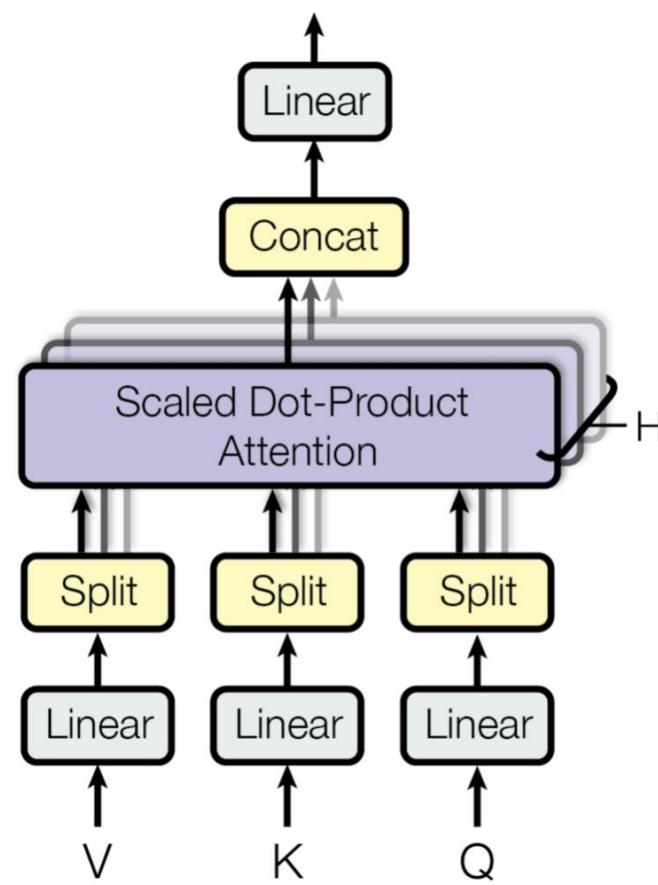
# Transformer

## ◆ Multi-Head Attention

Scaled Dot-Product Attention



Multi-Head Attention



# Transformer

## ◆ Multi-Head Attention

$d_{model} = 512$  次元の  $Q, K, V$  を用いて单一の内積注意を計算する代わりに,  
 $Q, K, V$  をそれぞれ  $h = 8$  回異なる重み行列  $W_i^Q, W_i^K, W_i^V$  で  
 $d_k, d_k, d_v = 64$  次元に線形写像して  $h = 8$  個の内積注意を計算する. 各内積  
注意の  $d_v = 64$  次元の出力は連結(concatenate) して重み行列  $W^O$  で  
 $d_{model} = 512$  次元に線形写像する.

複数ヘッドの注意は次式によって表される.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

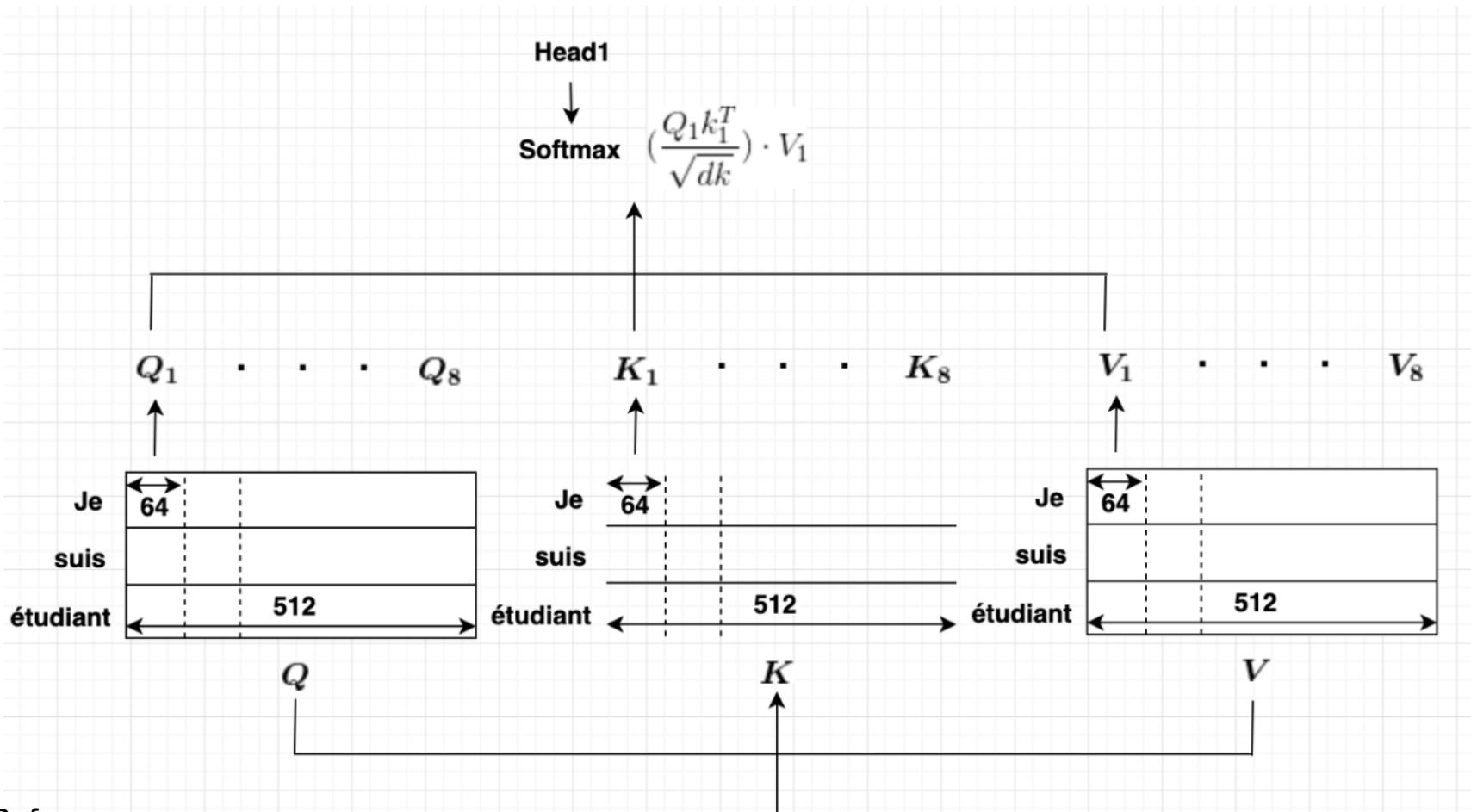
$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{model}}$$

ここで  $d_{model} = 512$  はすべての層の出力の次元数,  $d_k = 64$  は  $Q$  と  $K$  の次  
元数,  $d_v = 64$  は  $V$  の次元数である.

# Transformer

- ◆ Multi-Head Attention: It takes several self-attentions with several heads in a sentence.

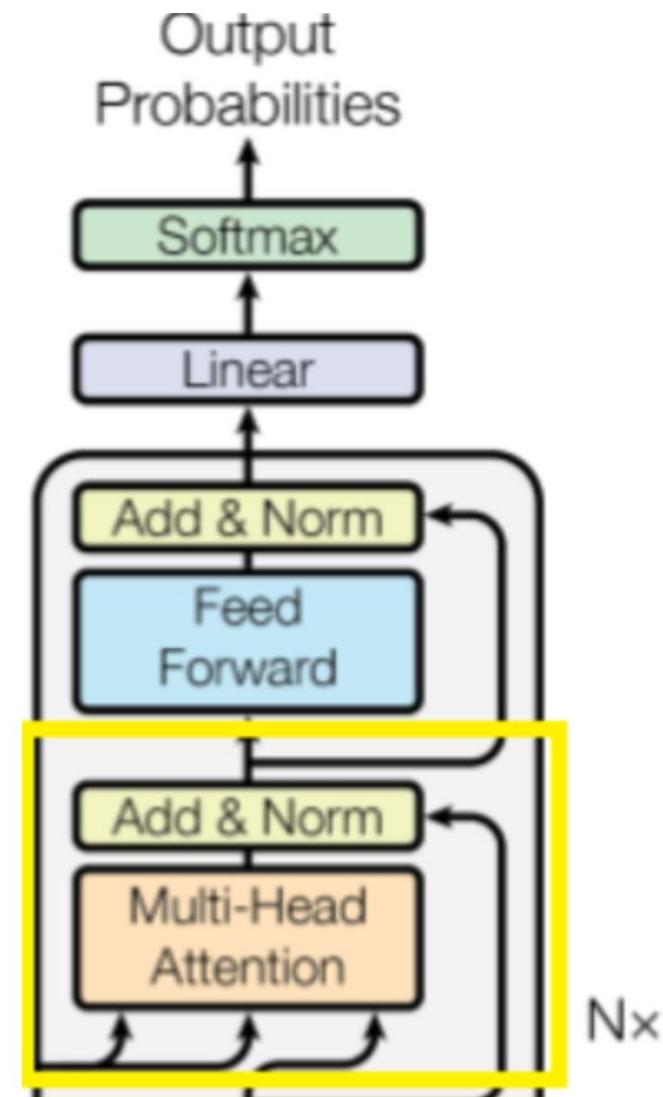
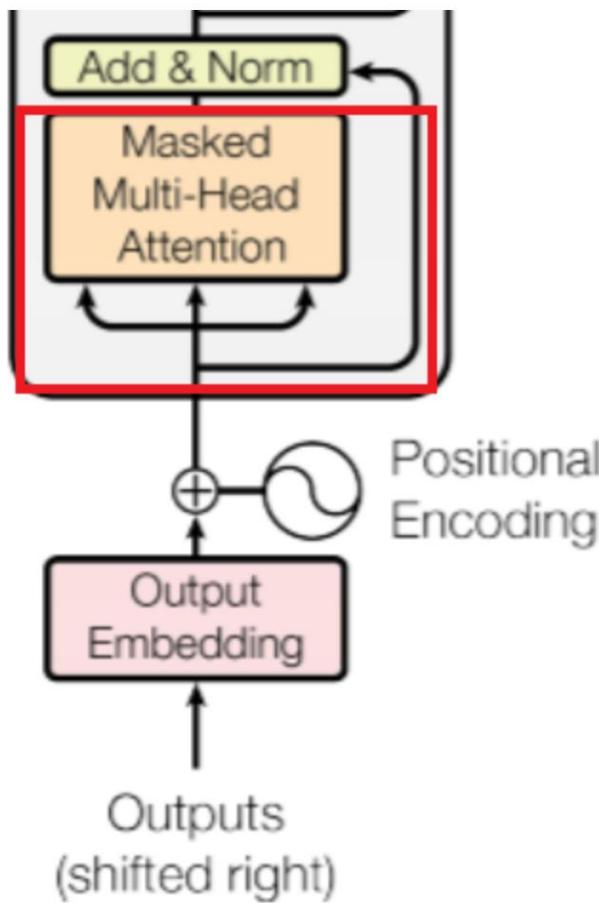


Reference:

<https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transformer-self-attention-842498fd3225>

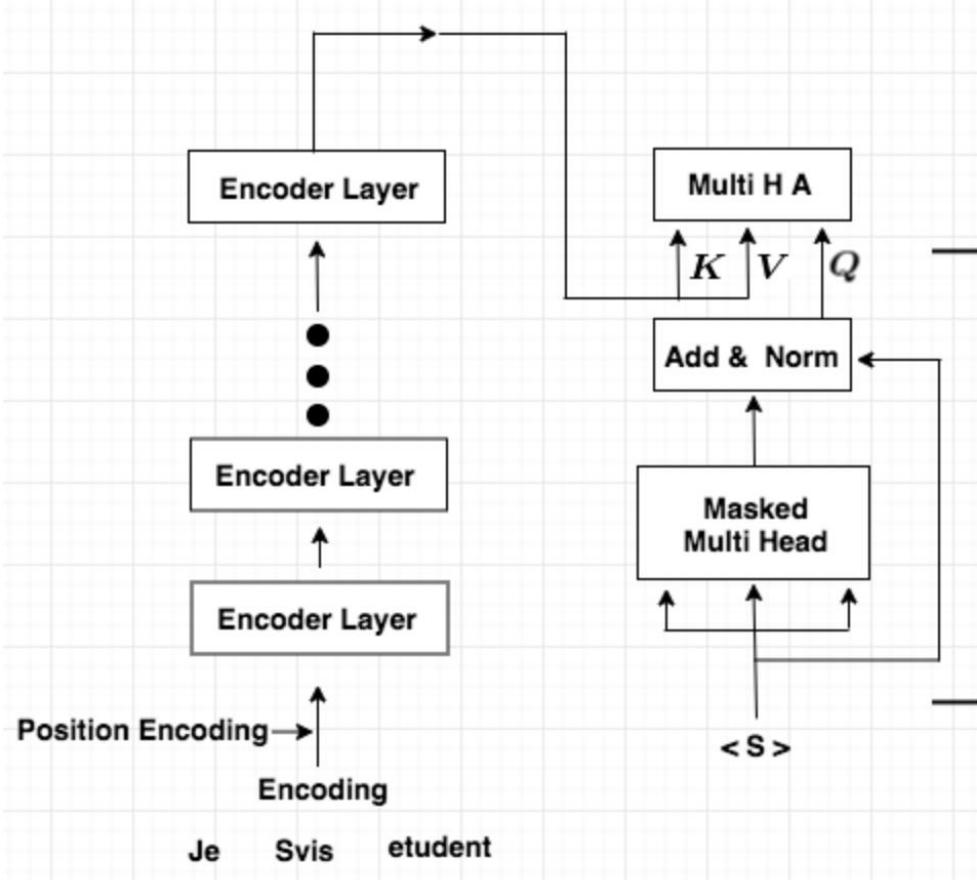
# Transformer

- ◆ Decoder: Masked Attention



# Transformer

## ◆ Decoder: Masked Attention



◆ Look-ahead Mask:  
Can't refer to the  
words in the  
future than self

Self  
Attention  
in Decoder

$$\begin{matrix} & Q \\ < sos > & \times \\ & je \\ & suis \\ & \text{étudiant} \end{matrix}$$

$$\begin{matrix} & < sos > \text{ je suis étudiant} \\ & \times K^T \end{matrix}$$

$$\begin{matrix} & < sos > \text{ je suis étudiant} \\ & \times \\ < sos > & \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \end{matrix} \\ & je \\ & suis \\ & \text{étudiant} \end{matrix}$$

Attention Score Matrix

$$\begin{matrix} & < sos > \text{ je suis étudiant} \\ & \times \\ < sos > & \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \end{matrix} \\ & je \\ & suis \\ & \text{étudiant} \end{matrix}$$

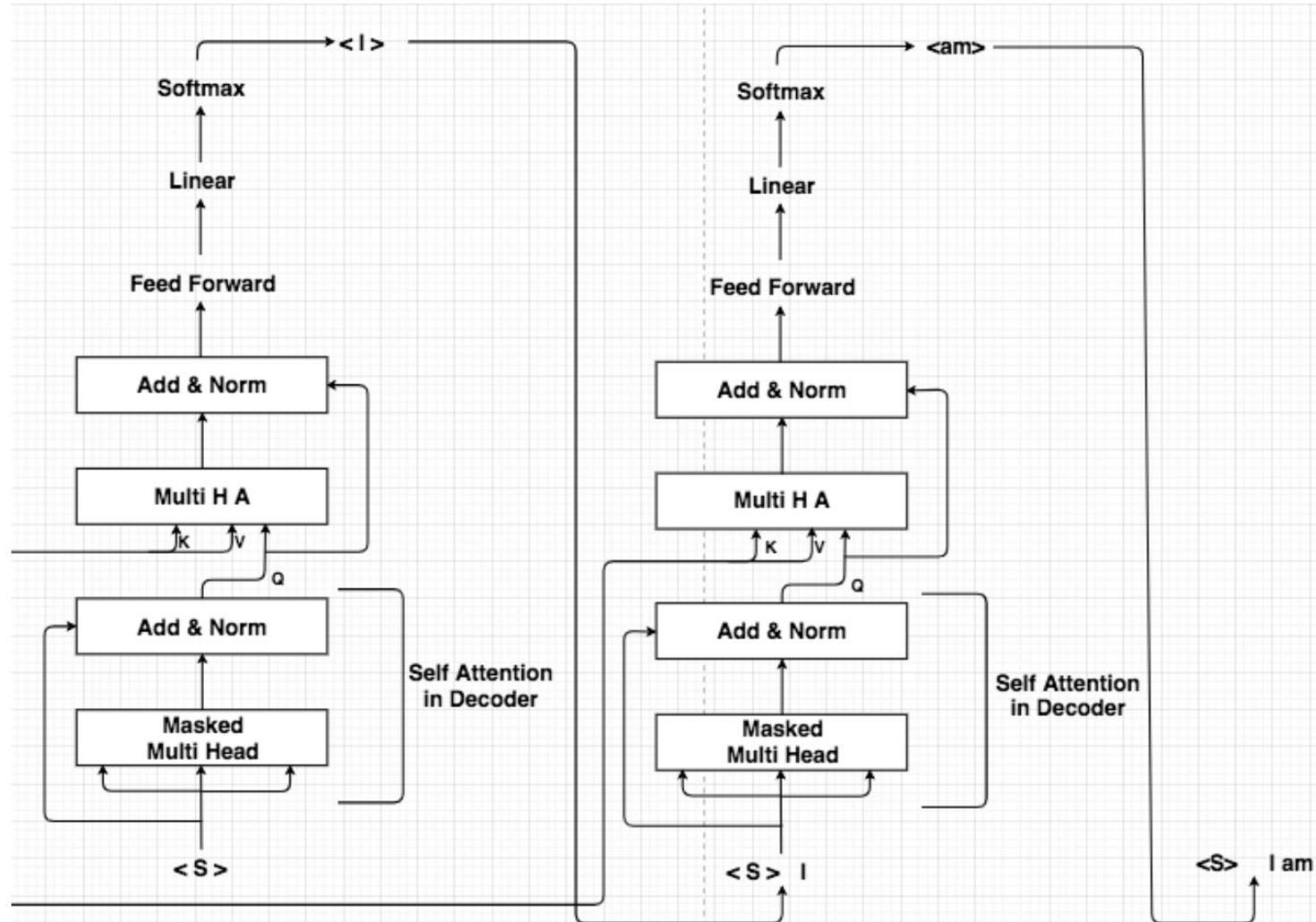
Attention Score Matrix

Reference:

<https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transformer-self-attention-842498fd3225>

# Transformer

## ◆ Inference

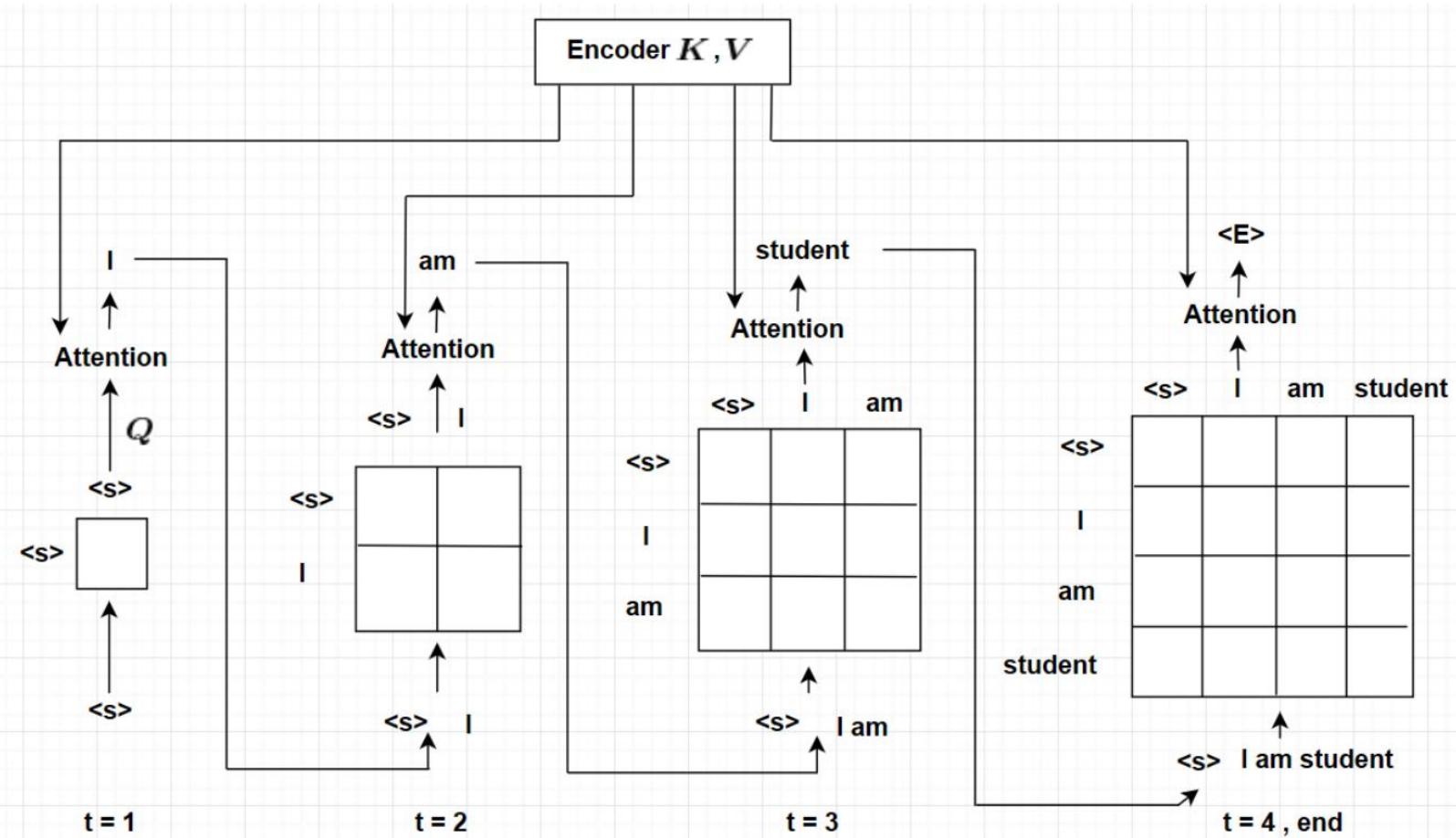


Reference:

<https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transformer-self-attention-842498fd3225>

# Transformer

## ◆ Inference



Reference:

<https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transformer-self-attention-842498fd3225>

# BERT & GPT-2

## ◆ Differences

- Bidirectional Encoder Representations from Transformers (BERT)
- Generative Pretrained Transformer(GPT)



GPT-2

DECODER

...

DECODER

DECODER



BERT

ENCODER

...

ENCODER

ENCODER



TRANSFORMER XL

RECURRENT DECODER

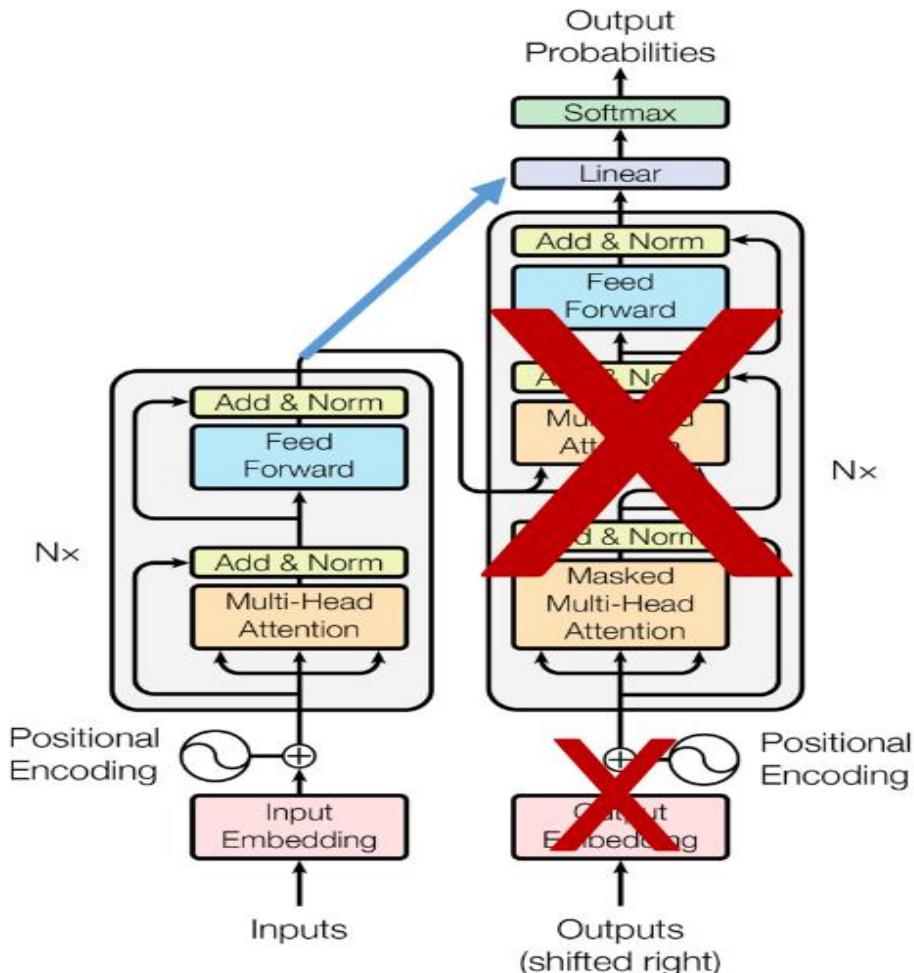
...

RECURRENT DECODER

RECURRENT DECODER

# Principle of BERT

- ◆ BERT Structure: Just use Encoder Part



# Principle of BERT

## ◆ BERT Architecture

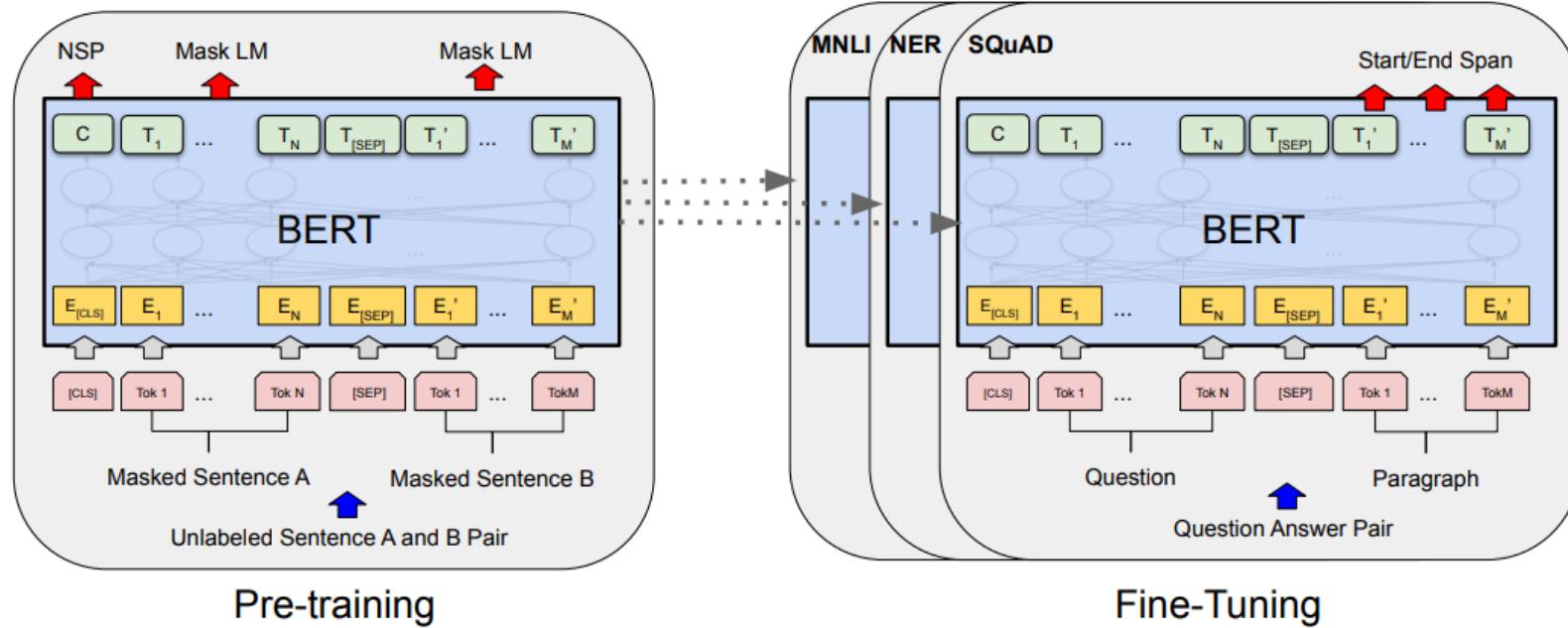


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

# Principle of BERT

## ◆ BERT Input Representation

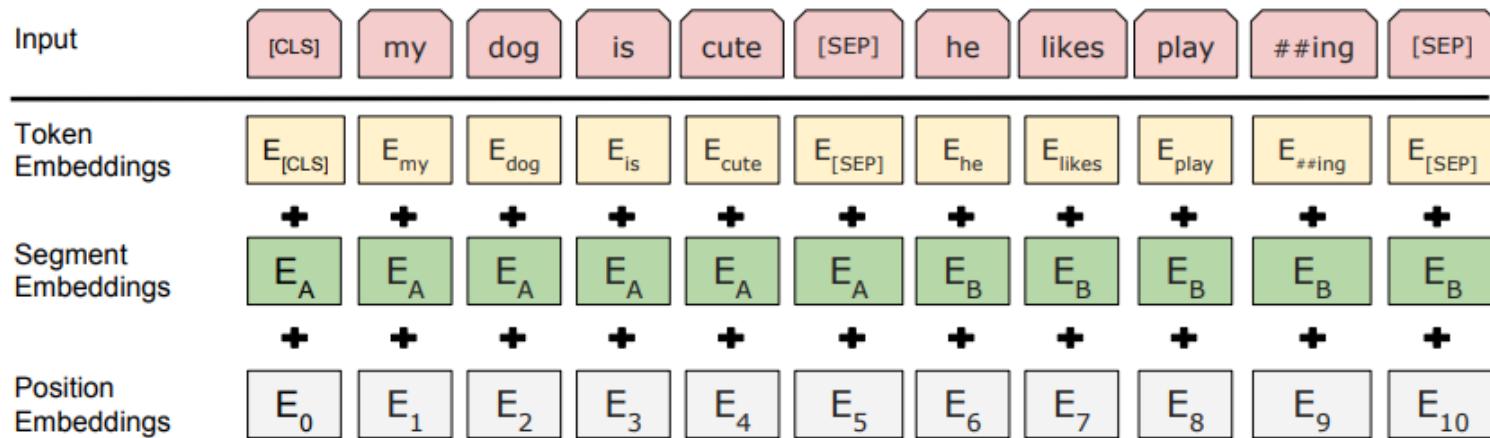


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

# BERT and Different Architectures

## ◆ Different Architecture

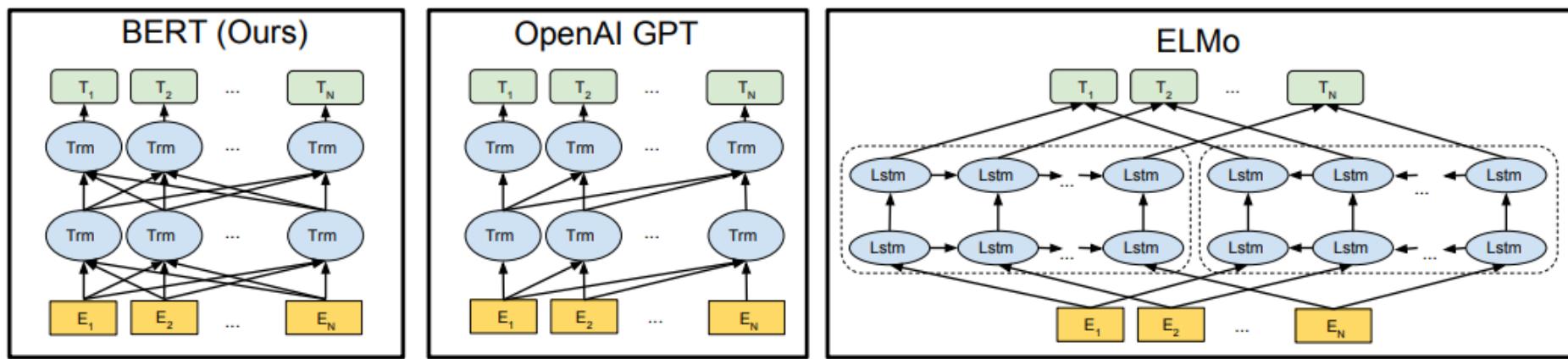
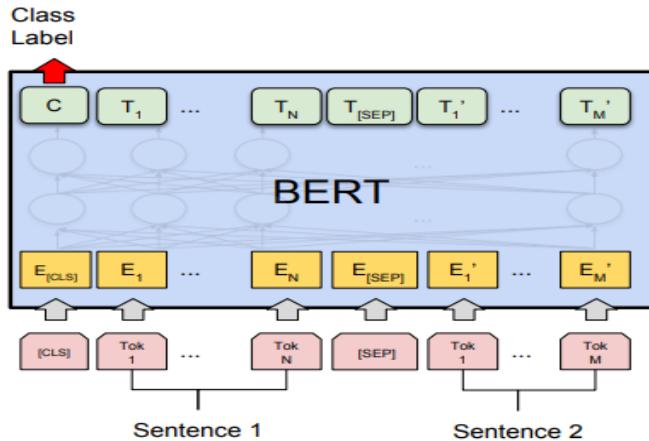


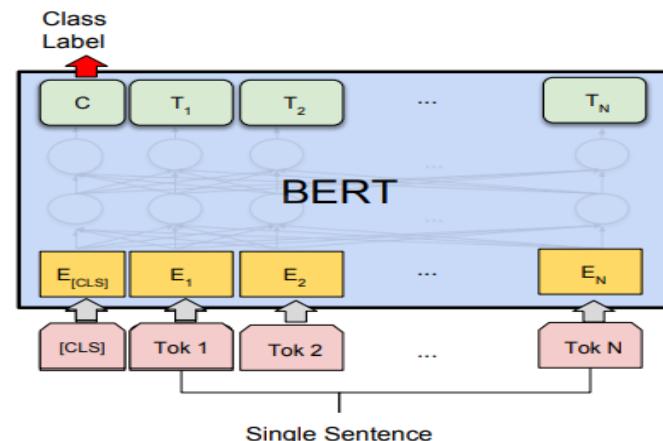
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

# Application of BERT

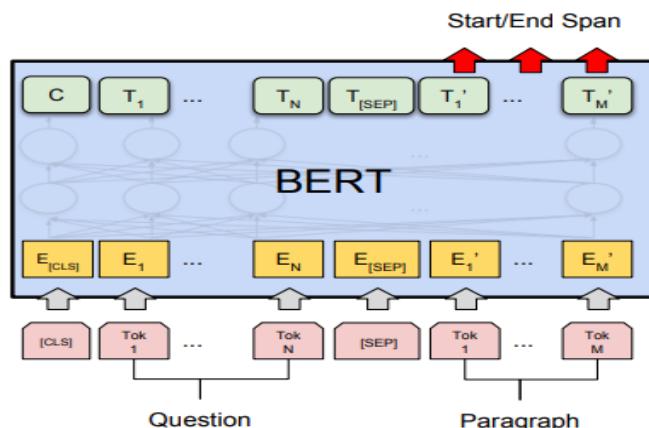
## ◆ Fine Tuning BERT on Different Tasks



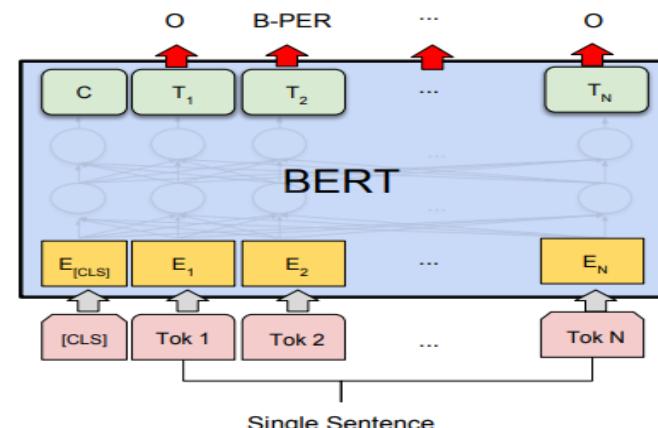
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



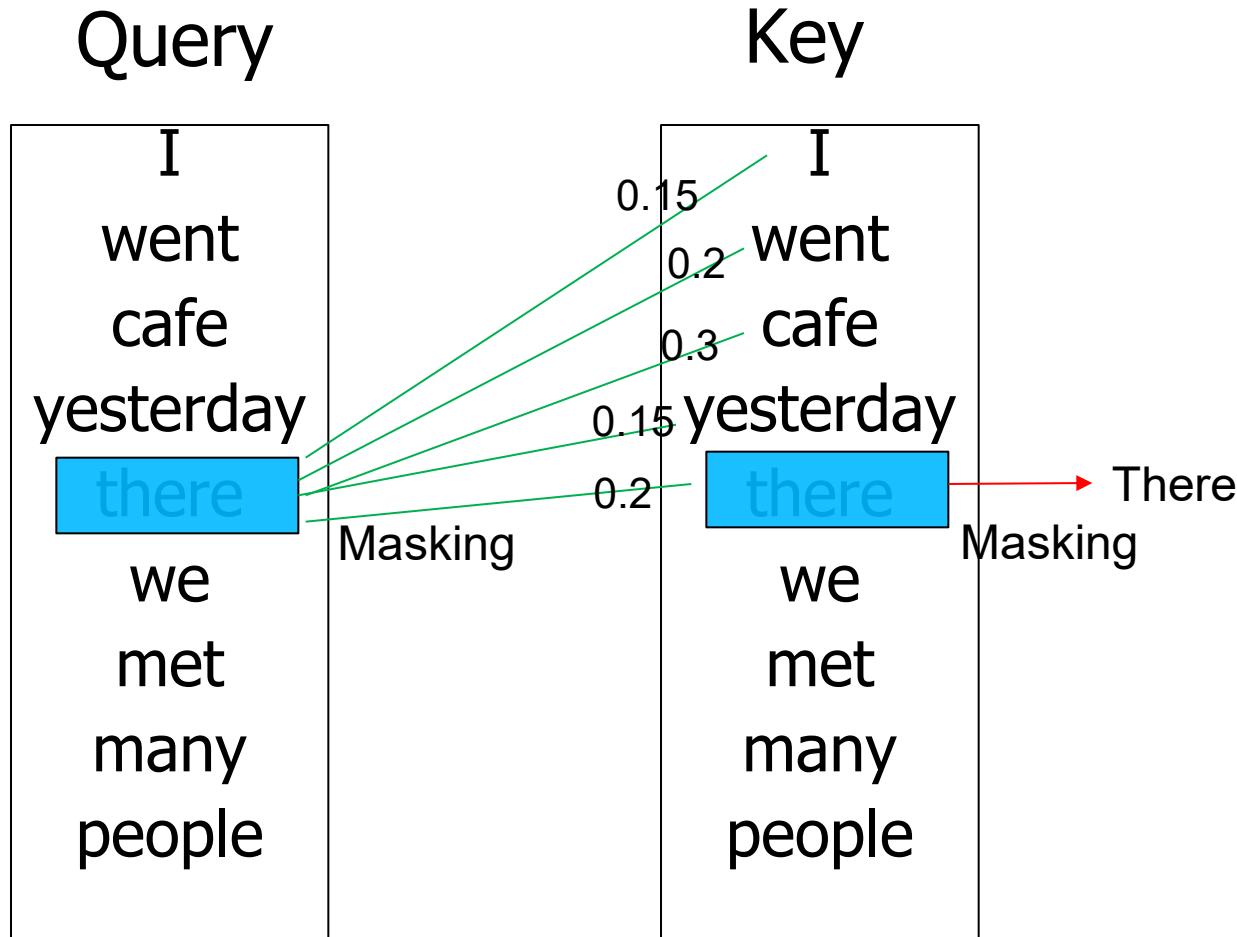
(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

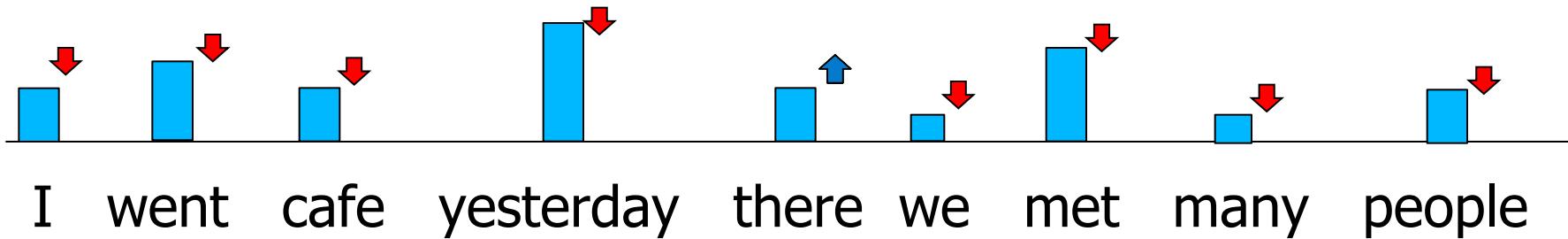
# Principle of BERT

## ◆ Attention in BERT



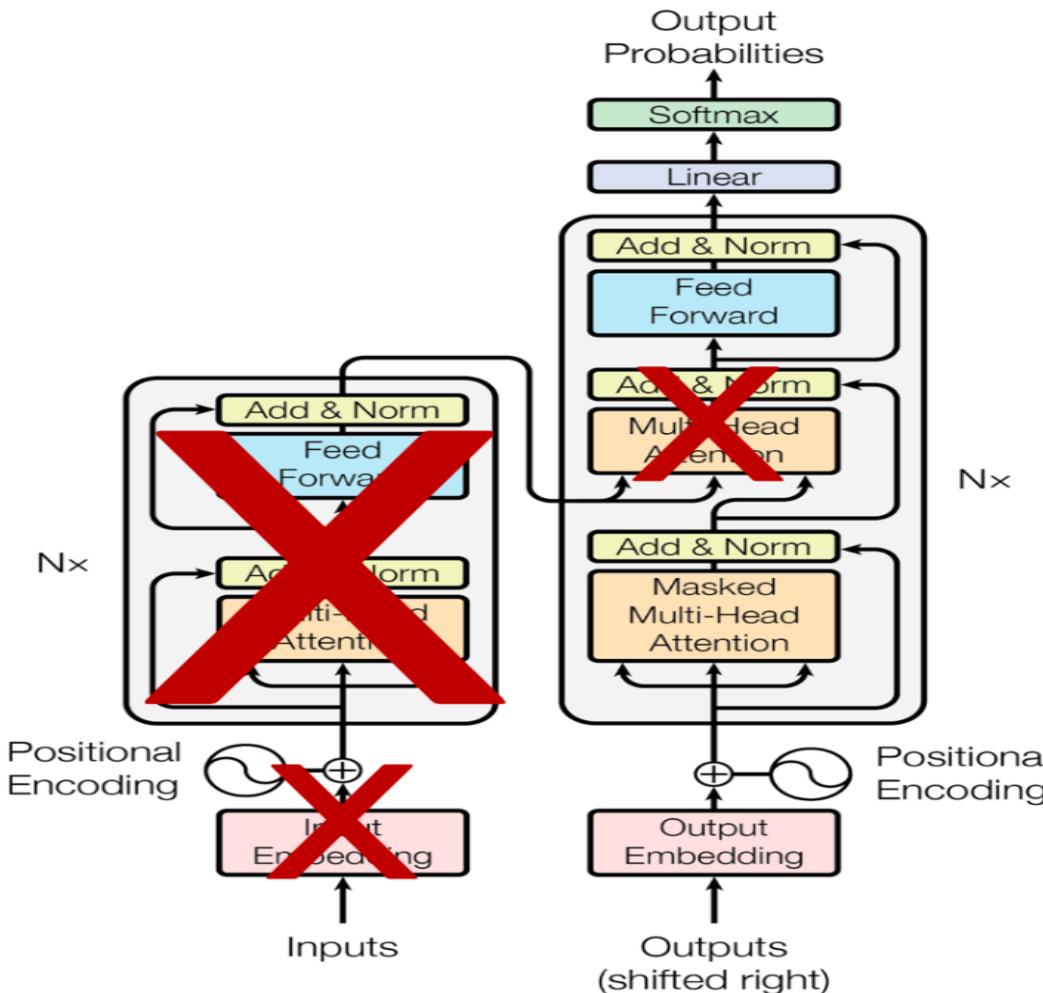
# Principle of BERT

- ◆ BERT Training: To create a probability vector that has elements with number of vocabularies of the language for learning, and there will be learning for the output of the last layer of the BERT corresponding to the word, MASK.
- ◆ Entire model will be updated so that the probability of the answer of the masked word will be increased, and the others decreased.



# Principle of GPT

## ◆ GPT Structure: Use Decoder Part



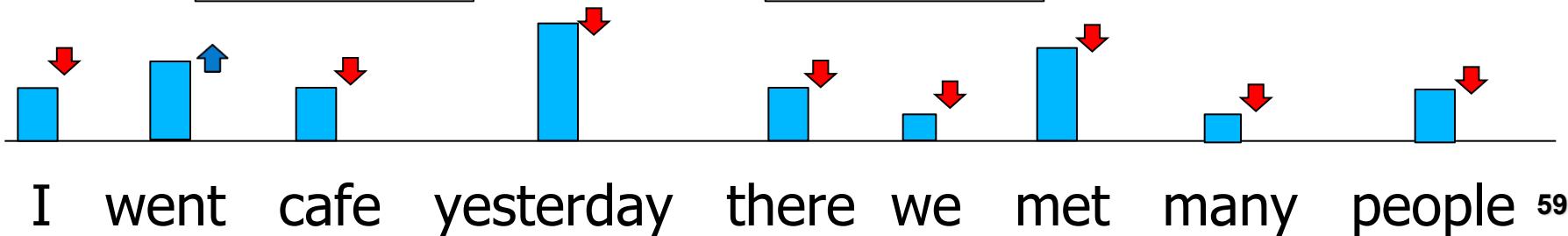
Reference: [https://ratsgo.github.io/nlpbook/docs/language\\_model/bert\\_gpt/](https://ratsgo.github.io/nlpbook/docs/language_model/bert_gpt/)

# GPT Attention (1)

Query

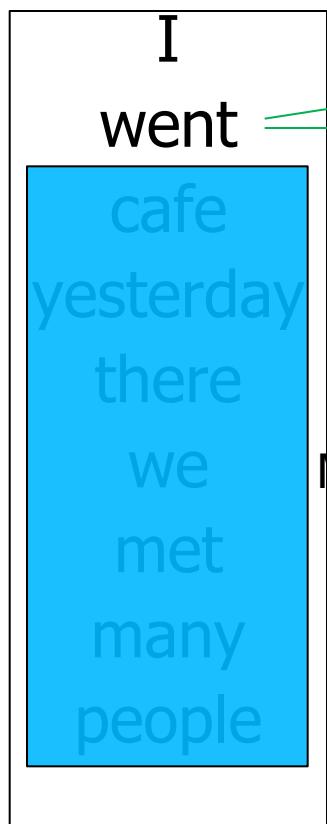


Key

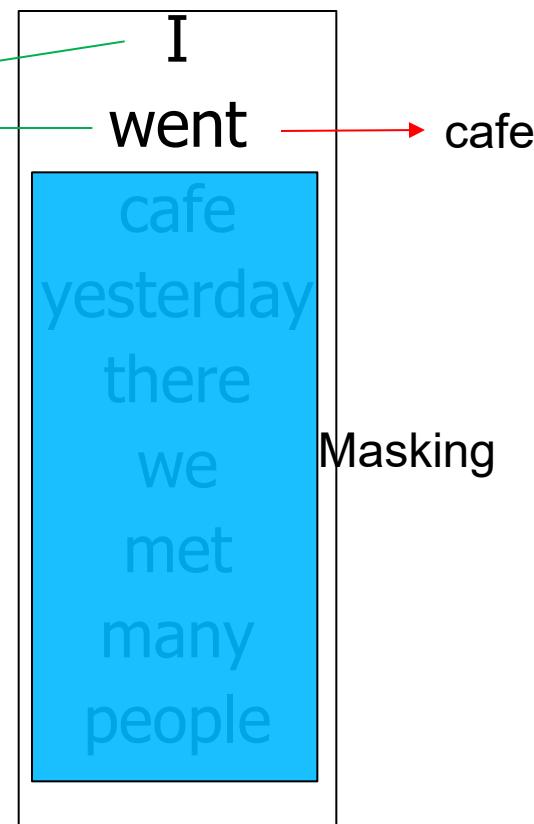


# GPT Attention (2)

Query

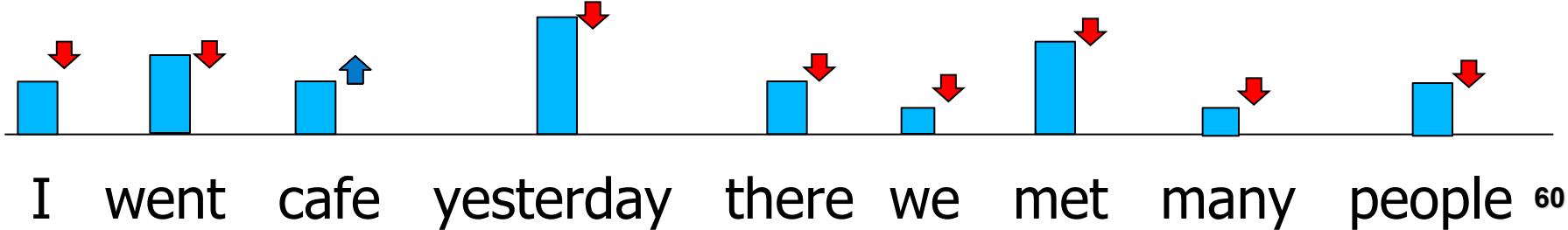


Key



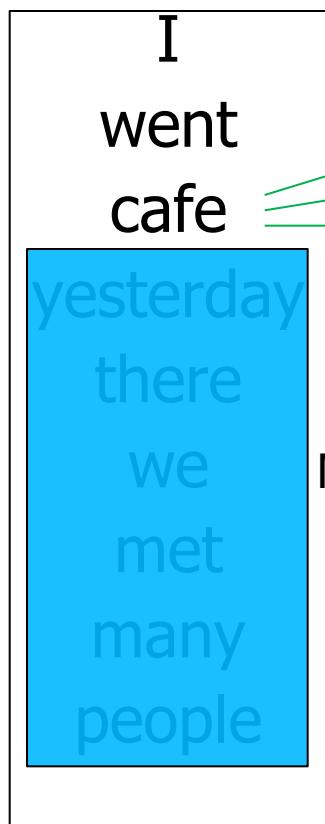
Masking

Masking

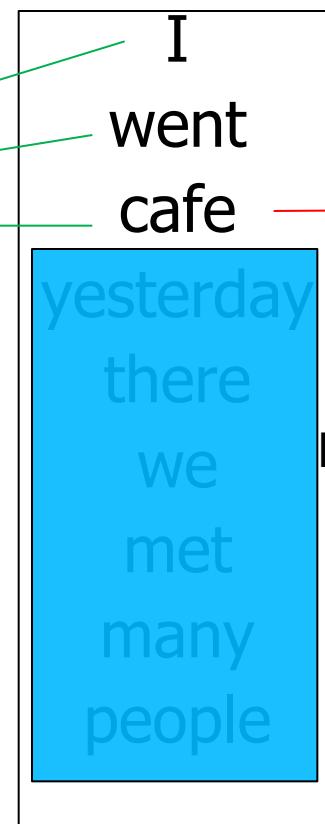


# GPT Attention (3)

Query

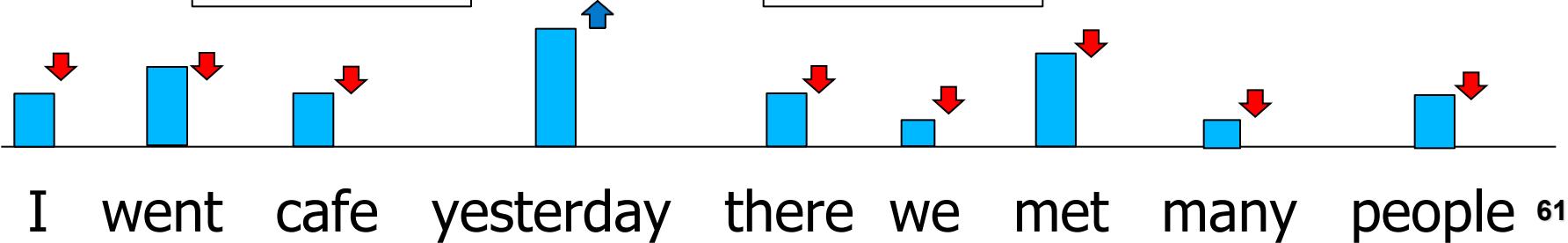


Key



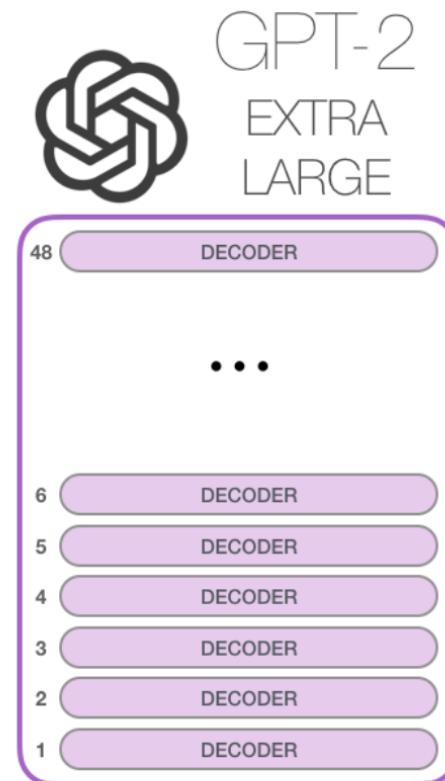
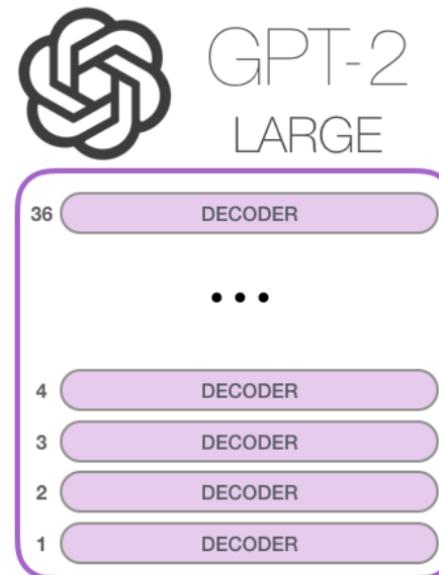
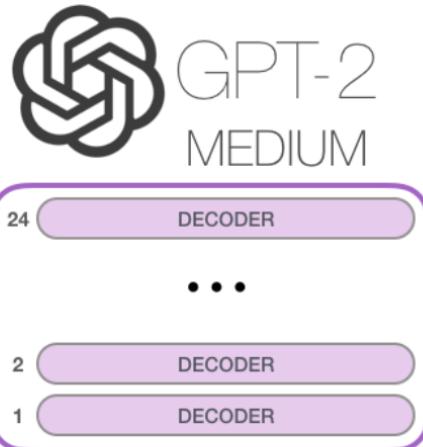
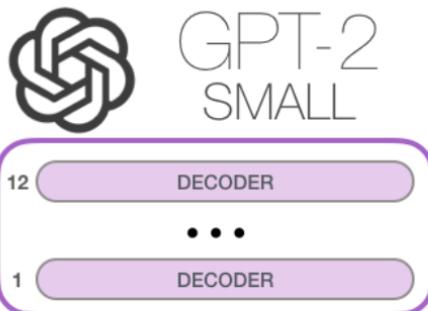
Masking

Masking



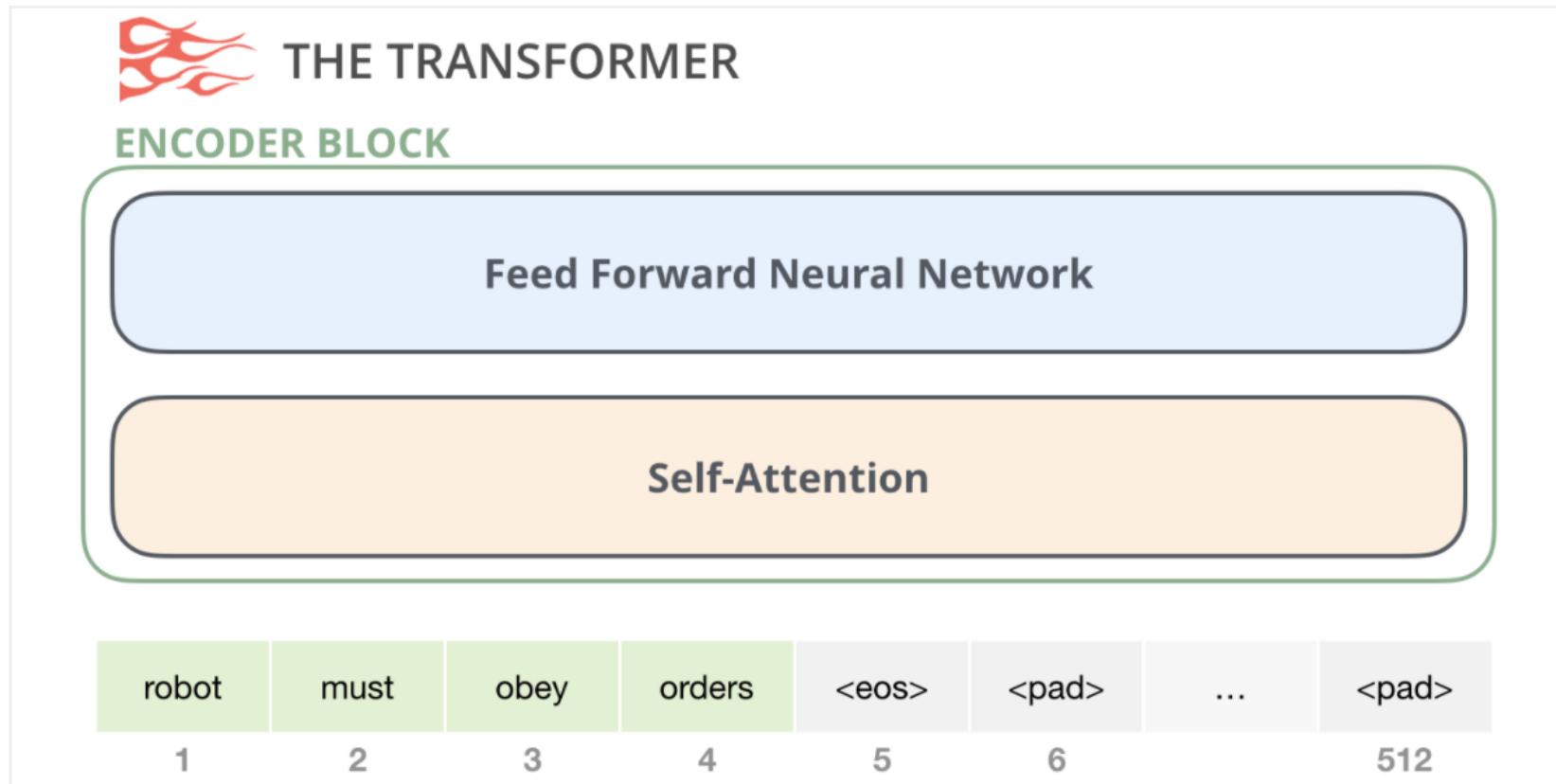
# BERT & GPT-2

## ◆ Several Models of GPT-2



# Transformer

## ◆ Evolution of Transformer Block: Encoder Block



An encoder block from the original transformer paper can take inputs up until a certain max sequence length (e.g. 512 tokens). It's okay if an input sequence is shorter than this limit, we can just pad the rest of the sequence.

# Transformer

## ◆ Evolution of Transformer Block



THE TRANSFORMER

DECODER BLOCK

Feed Forward Neural Network

Encoder-Decoder Self-Attention

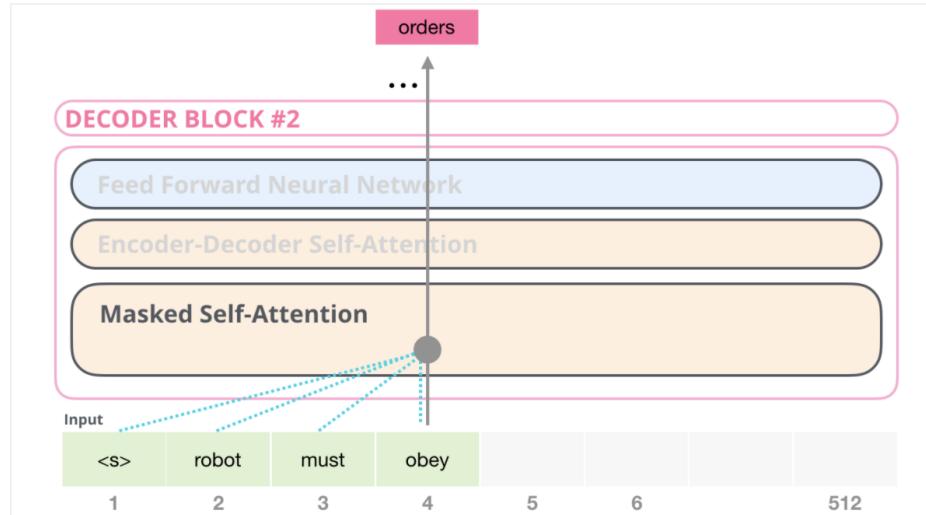
Masked Self-Attention

Input

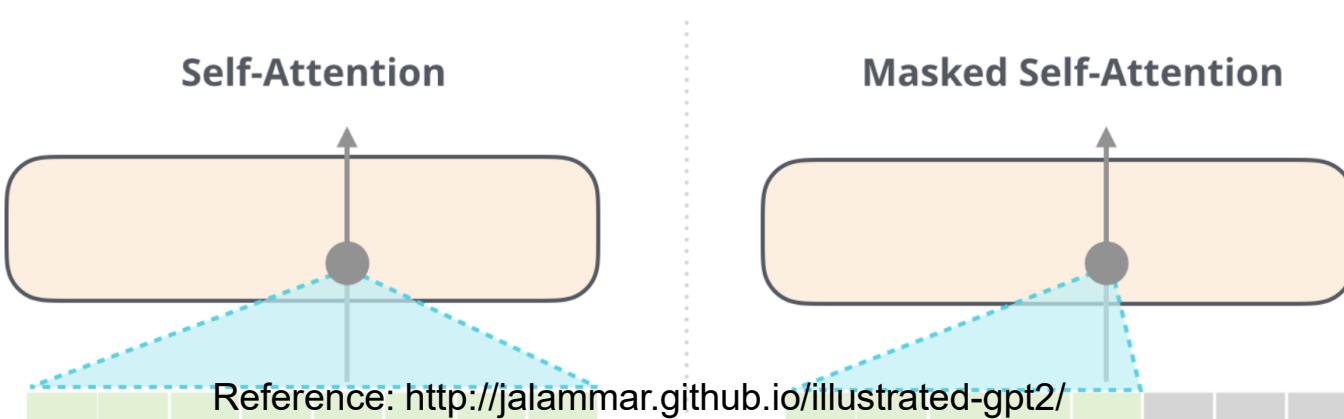
<s>	robot	must	obey						512
1	2	3	4	5	6				

# Transformer

## ◆ Evolution of Transformer Block: Masked Self-Attention



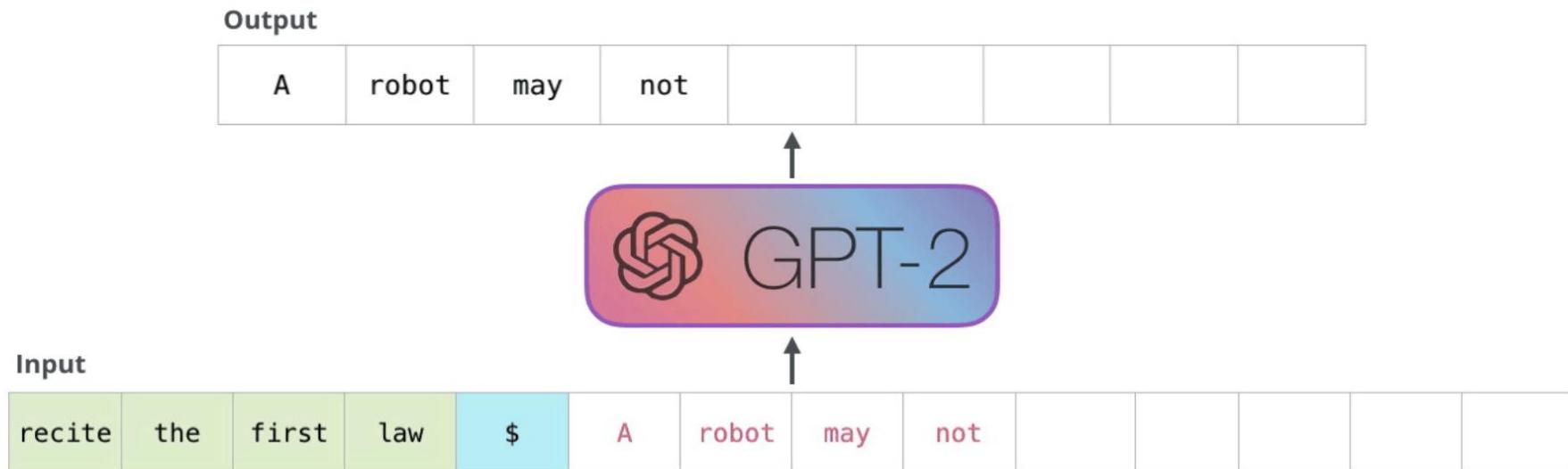
It's important that the distinction between self-attention (what BERT uses) and masked self-attention (what GPT-2 uses) is clear. A normal self-attention block allows a position to peak at tokens to its right. Masked self-attention prevents that from happening:



# GPT-2

## ◆ Work of GPT-2

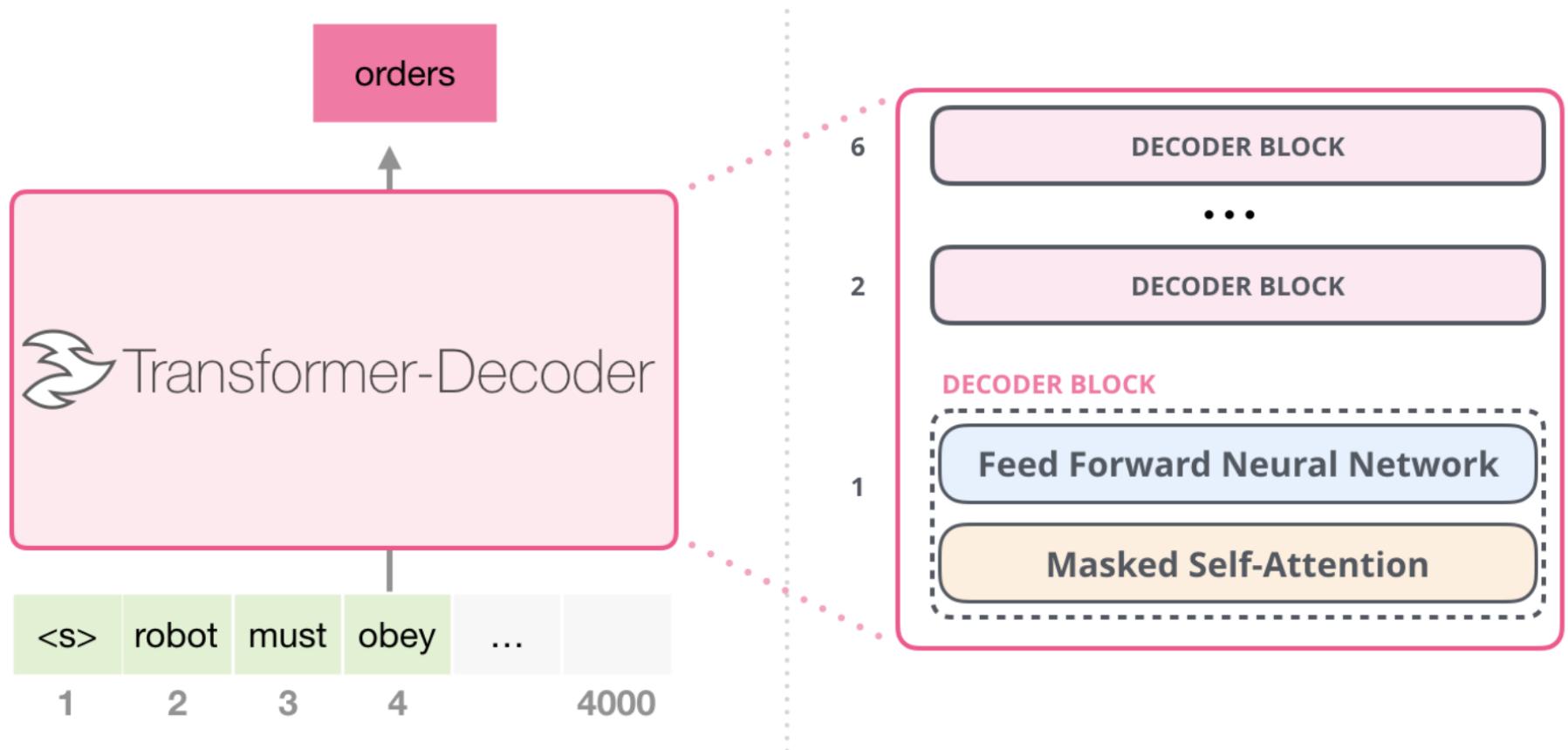
The way these models actually work is that after each token is produced, that token is added to the sequence of inputs. And that new sequence becomes the input to the model in its next step. This is an idea called “auto-regression”. This is one of the ideas that made RNNs unreasonably effective.



The GPT2, and some later models like TransformerXL and XLNet are auto-regressive in nature. BERT is not. That is a trade off. In losing auto-regression, BERT gained the ability to incorporate the context on both sides of a word to gain better results. XLNet brings back autoregression while finding an alternative way to incorporate the context on both sides.

# GPT-2

## ◆ Evolution of Transformer Block: Decoder-only block

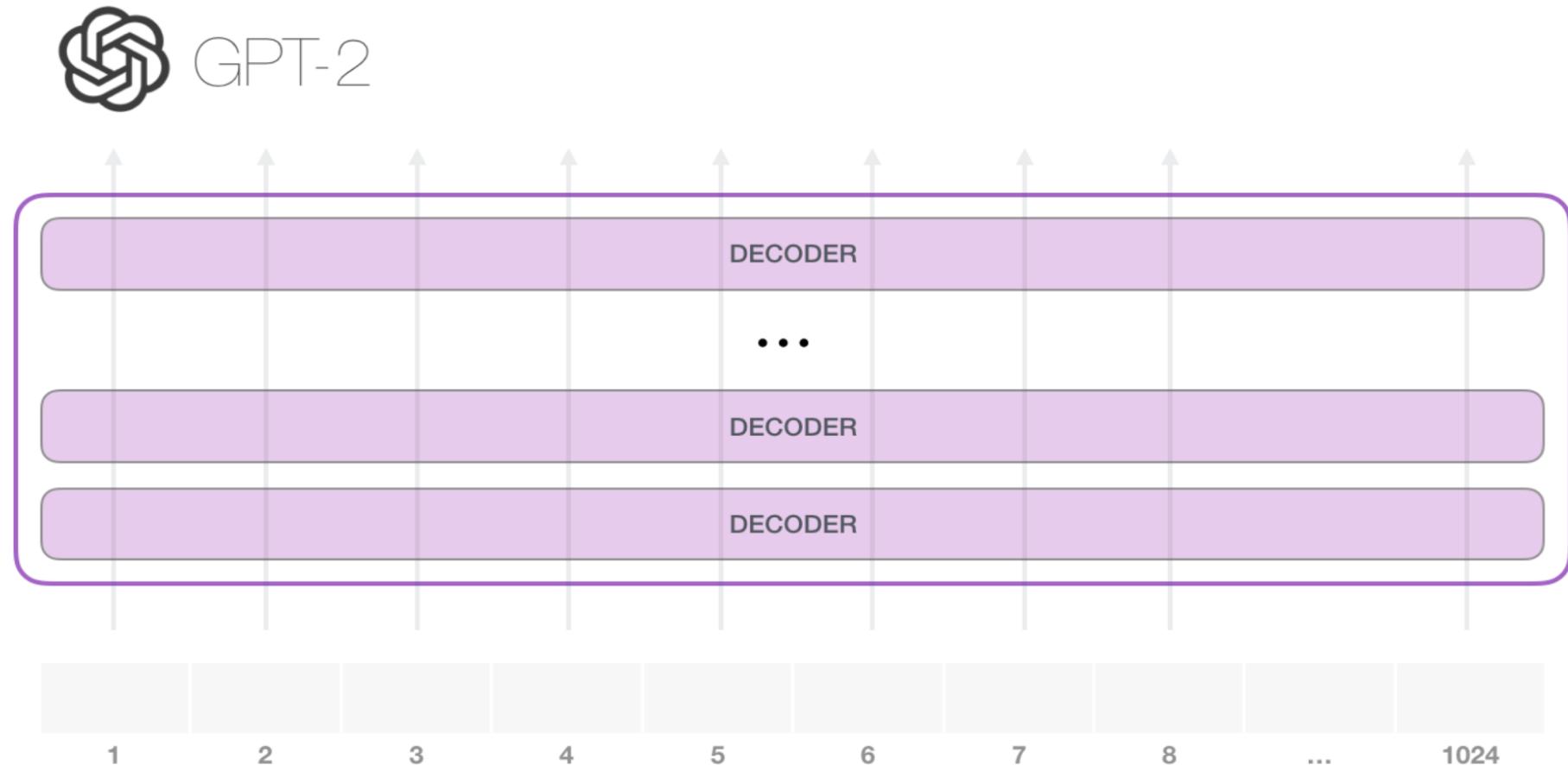


The decoder blocks are identical. I have expanded the first one so you can see its self-attention layer is the masked variant. Notice that the model now can address up to 4,000 tokens in a certain segment -- a massive upgrade from the 512 in the original transformer.

# GPT-2

## ◆ Looking Inside GPT-2

Let's lay a trained GPT-2 on our surgery table and look at how it works.

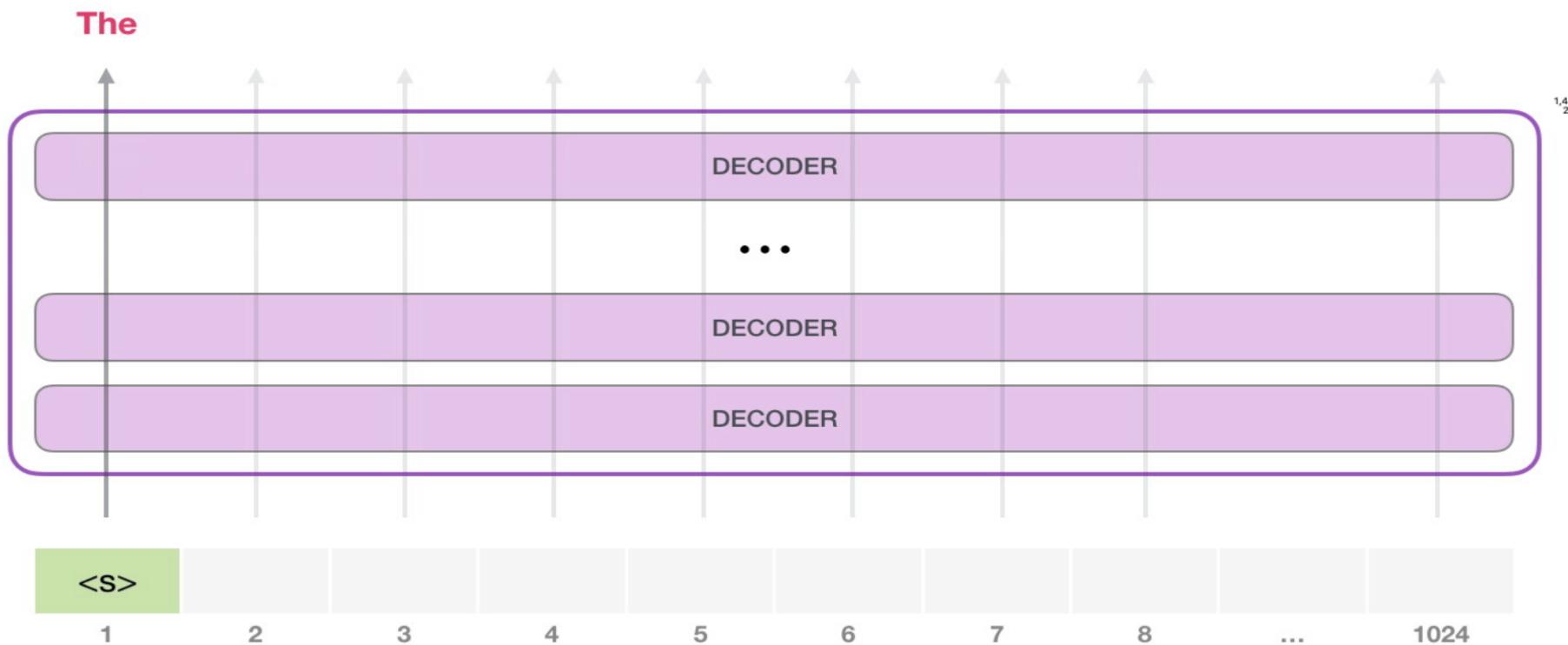


The GPT-2 can process 1024 tokens. Each token flows through all the decoder blocks along its own path.

# GPT-2

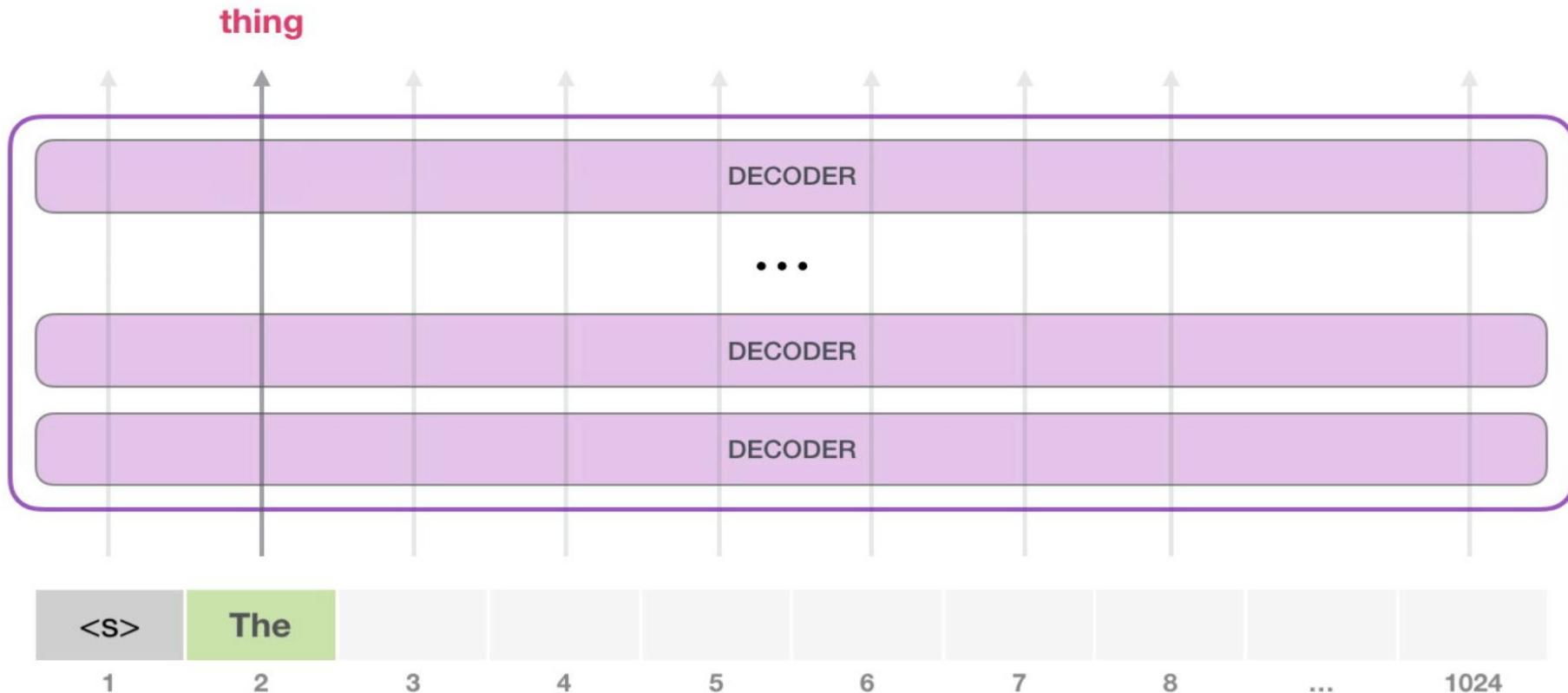
## ◆ Looking Inside GPT-2

The simplest way to run a trained GPT-2 is to allow it to ramble on its own (which is technically called *generating unconditional samples*) – alternatively, we can give it a prompt to have it speak about a certain topic (a.k.a generating *interactive conditional samples*). In the rambling case, we can simply hand it the start token and have it start generating words (the trained model uses `<|endoftext|>` as its start token. Let's call it `<s>` instead).



# GPT-2

## ◆ Looking Inside GPT-2

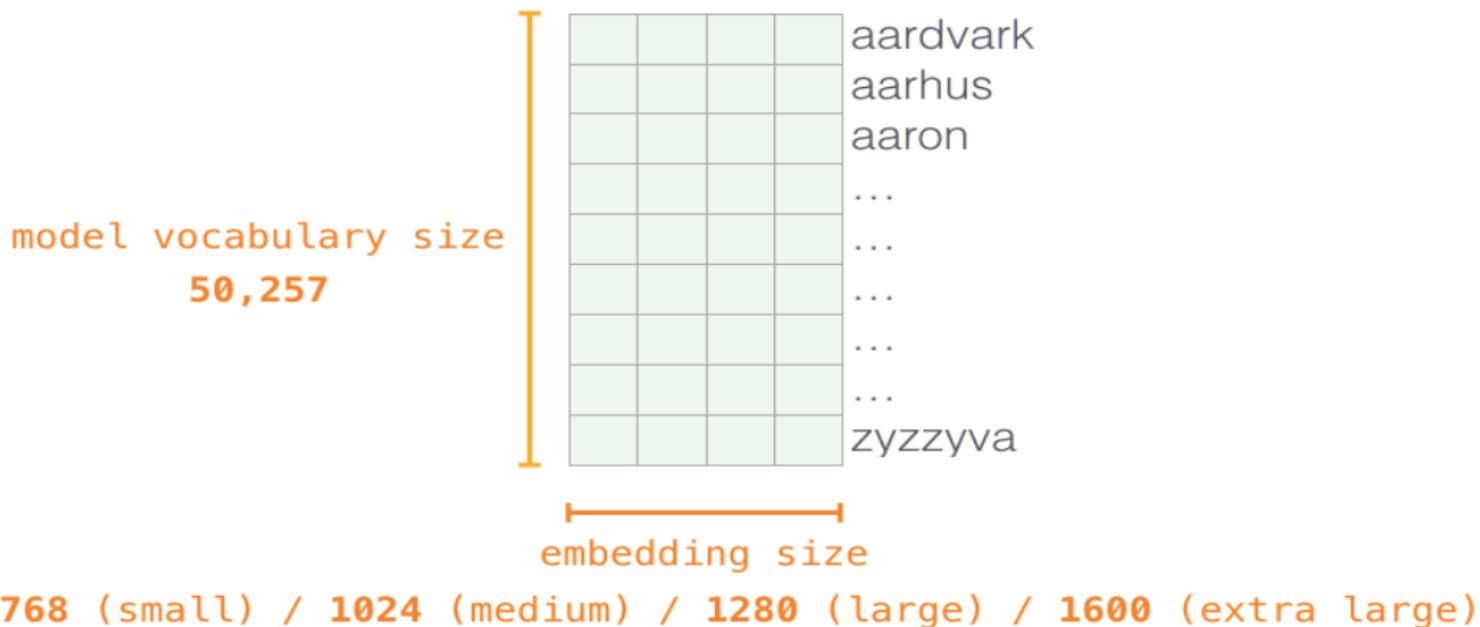


Notice that the second path is the only that's active in this calculation. Each layer of GPT-2 has retained its own interpretation of the first token and will use it in processing the second token (we'll get into more detail about this in the following section about self-attention). GPT-2 does not re-interpret the first token in light of the second token.

# GPT-2

## ◆ Deeper Looking Inside GPT-2

### Token Embeddings (wte)

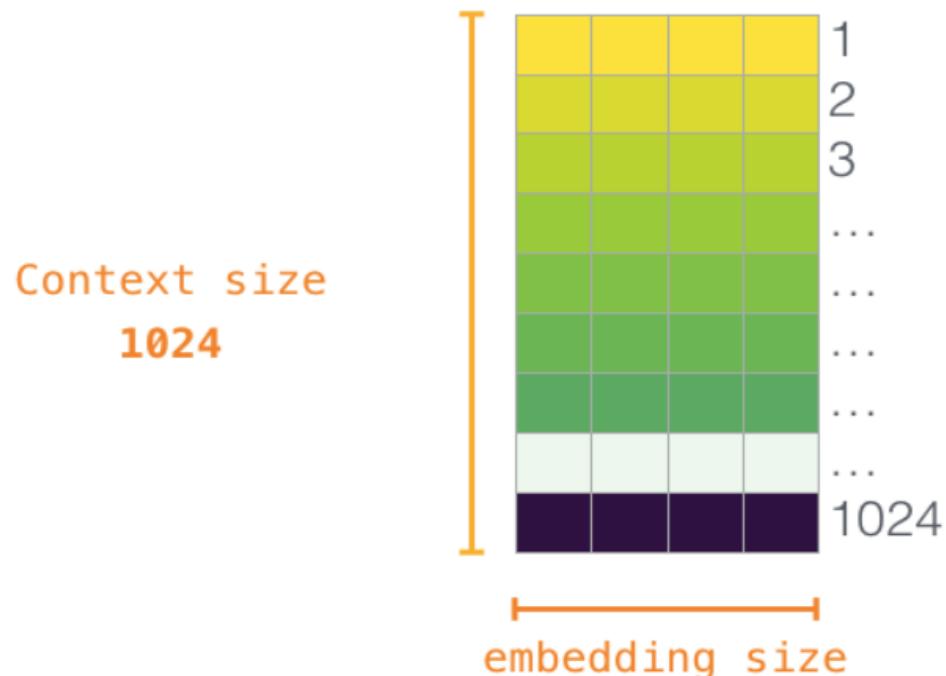


Each row is a word embedding: a list of numbers representing a word and capturing some of its meaning. The size of that list is different in different GPT2 model sizes. The smallest model uses an embedding size of 768 per word/token.

# GPT-2

- ◆ Deeper Looking Inside GPT-2

## Positional Encodings (wpe)



768 (small) / 1024 (medium) / 1280 (large) / 1600 (extra large)

# GPT-2

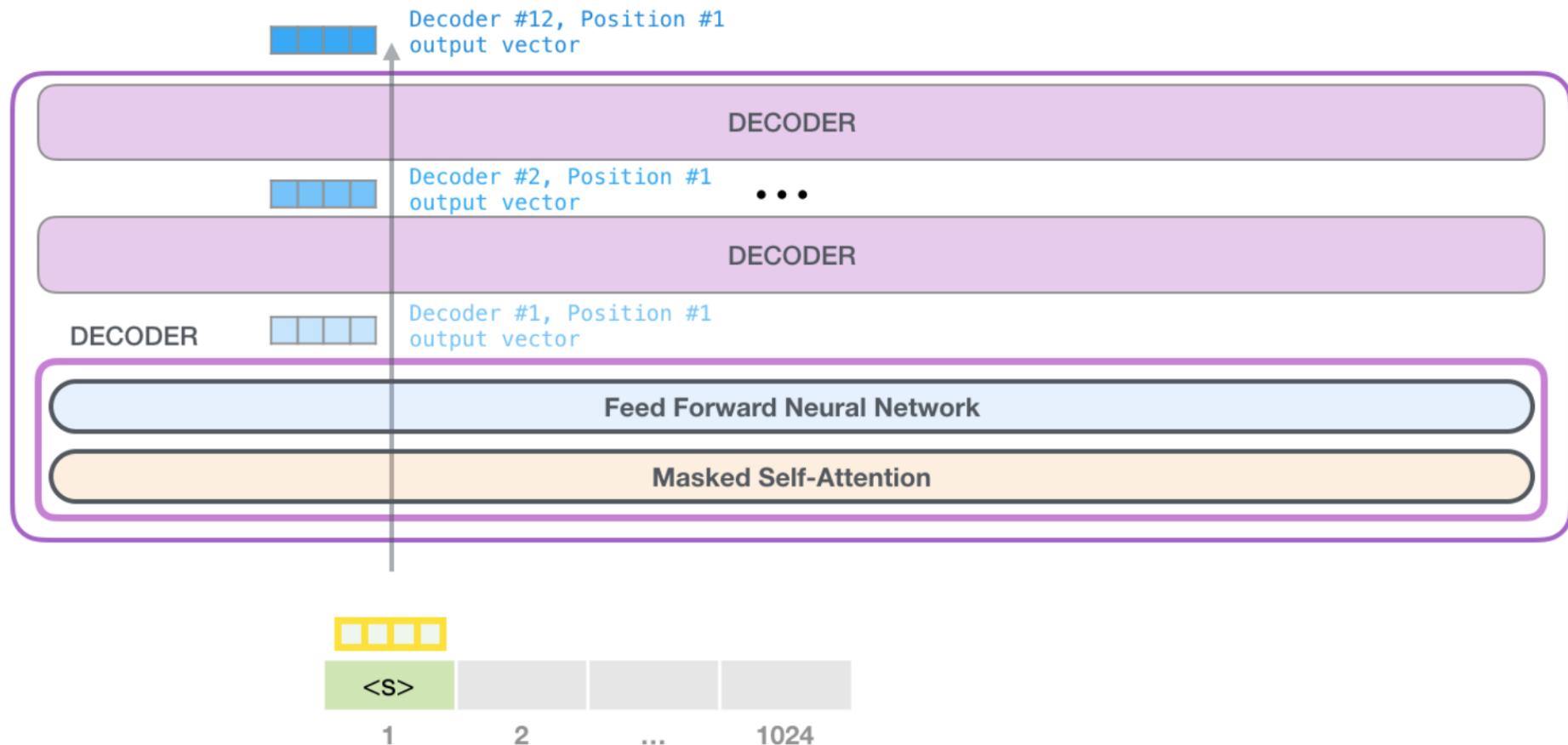
## ◆ Deeper Looking Inside GPT-2



Sending a word to the first transformer block means looking up its embedding and adding up the positional encoding vector for position #1.

# GPT-2

## ◆ Deeper Looking Inside GPT-2: Entire Stack



# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention

Language heavily relies on context. For example, look at the second law:

### *Second Law of Robotics*

*A robot must obey the orders given **it** by human beings except where **such orders** would conflict with the **First Law**.*

I have highlighted three places in the sentence where the words are referring to other words. There is no way to understand or process these words without incorporating the context they are referring to. When a model processes this sentence, it has to be able to know that:

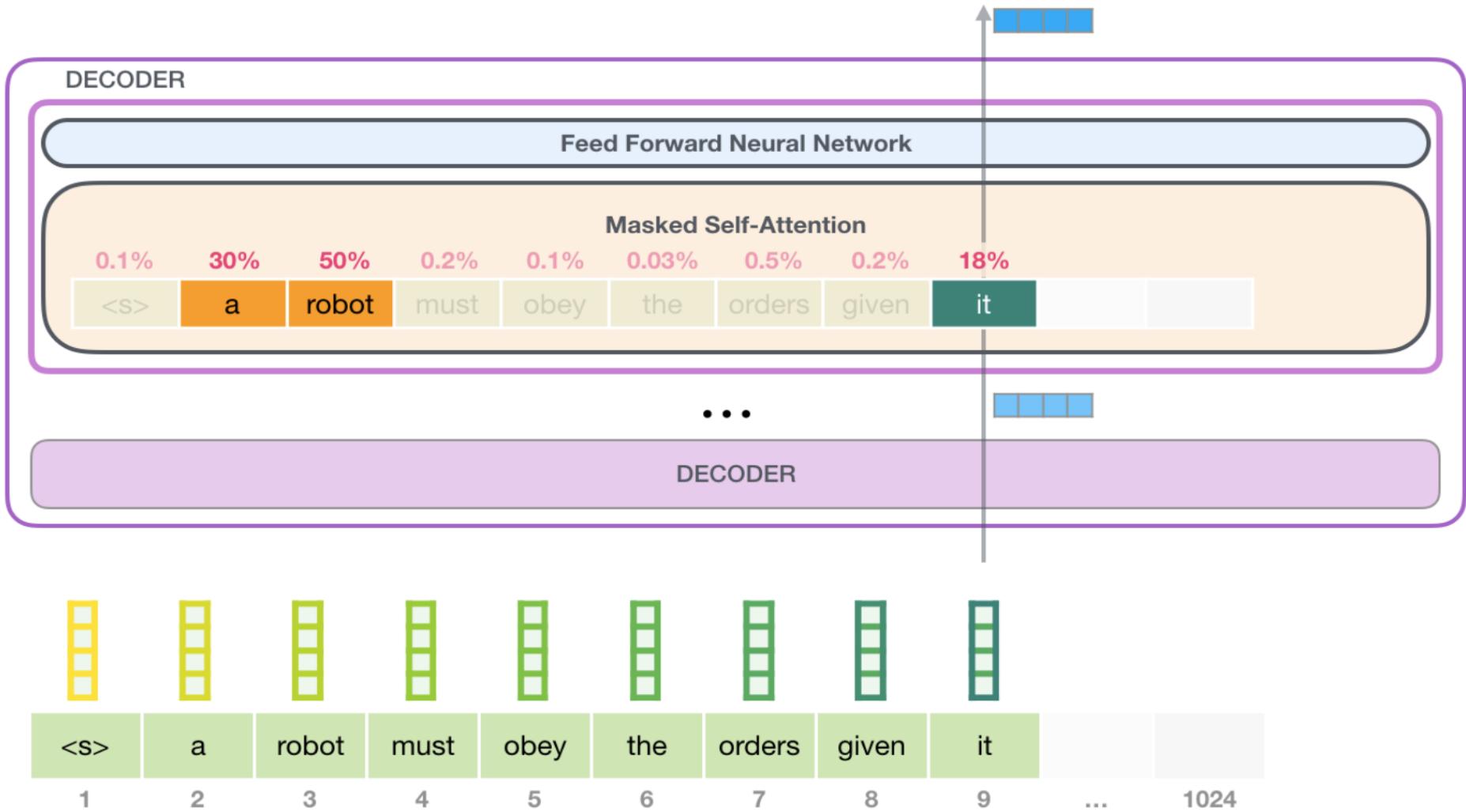
- **it** refers to the robot
- **such orders** refers to the earlier part of the law, namely “the orders given it by human beings”
- **The First Law** refers to the entire First Law

This is what self-attention does. It bakes in the model’s understanding of relevant and associated words that explain the context of a certain word before processing that word (passing it through a neural network). It does that by assigning scores to how relevant each word in the segment is, and adding up their vector representation.

As an example, this self-attention layer in the top block is paying attention to “a robot” when it processes the word “it”. The vector it will pass to its neural network is a sum of the vectors for each of the three words multiplied by their scores.

# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention

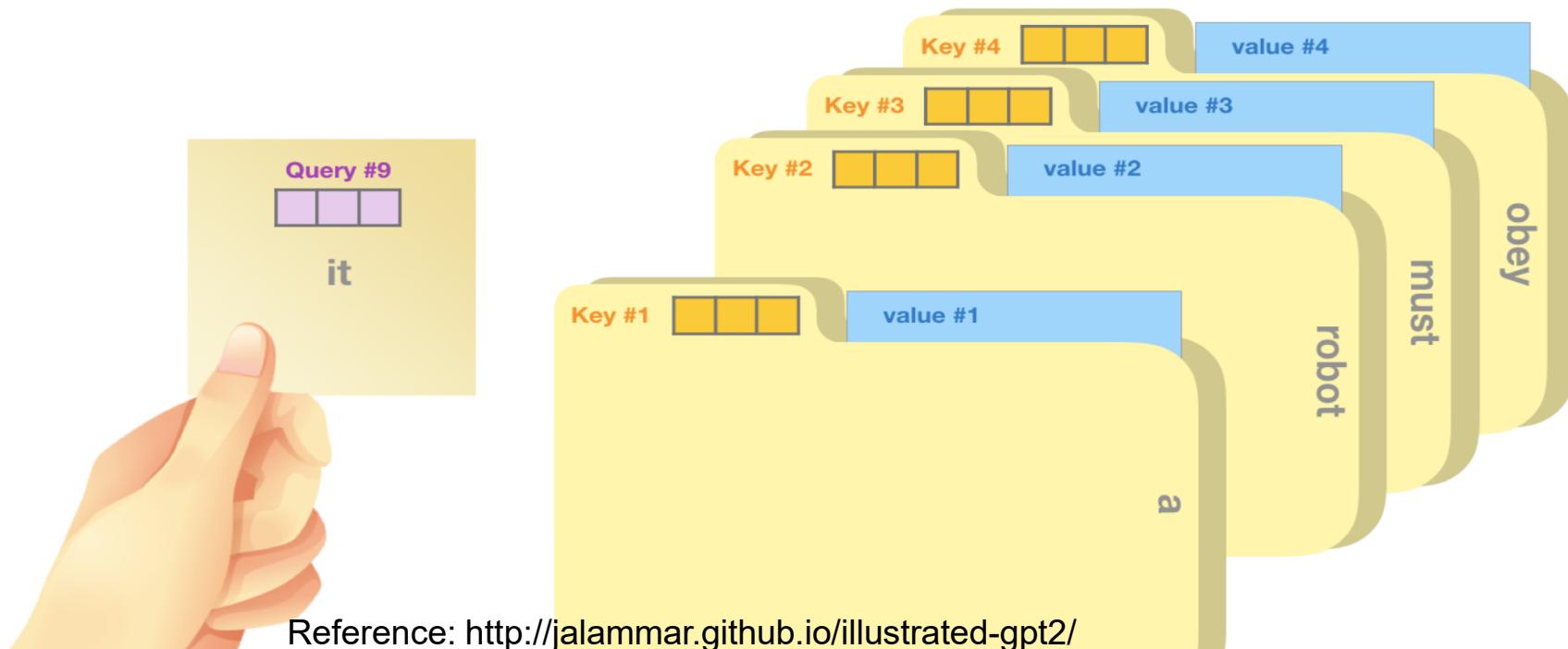


# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention Process

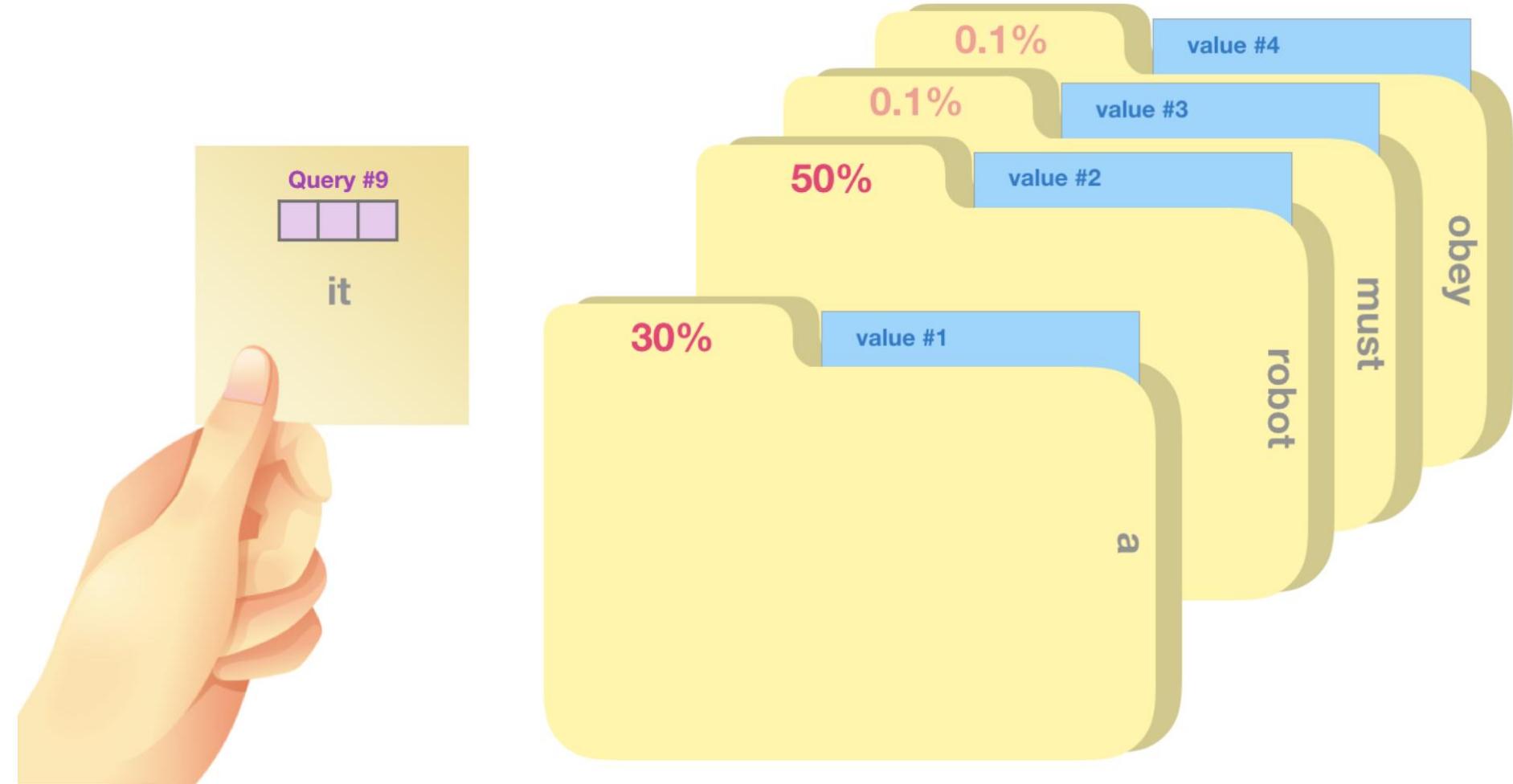
Self-attention is processed along the path of each token in the segment. The significant components are three vectors:

- **Query**: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
- **Key**: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
- **Value**: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.



# GPT-2

- ◆ Deeper Looking Inside GPT-2: Self-Attention Process



# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention Process

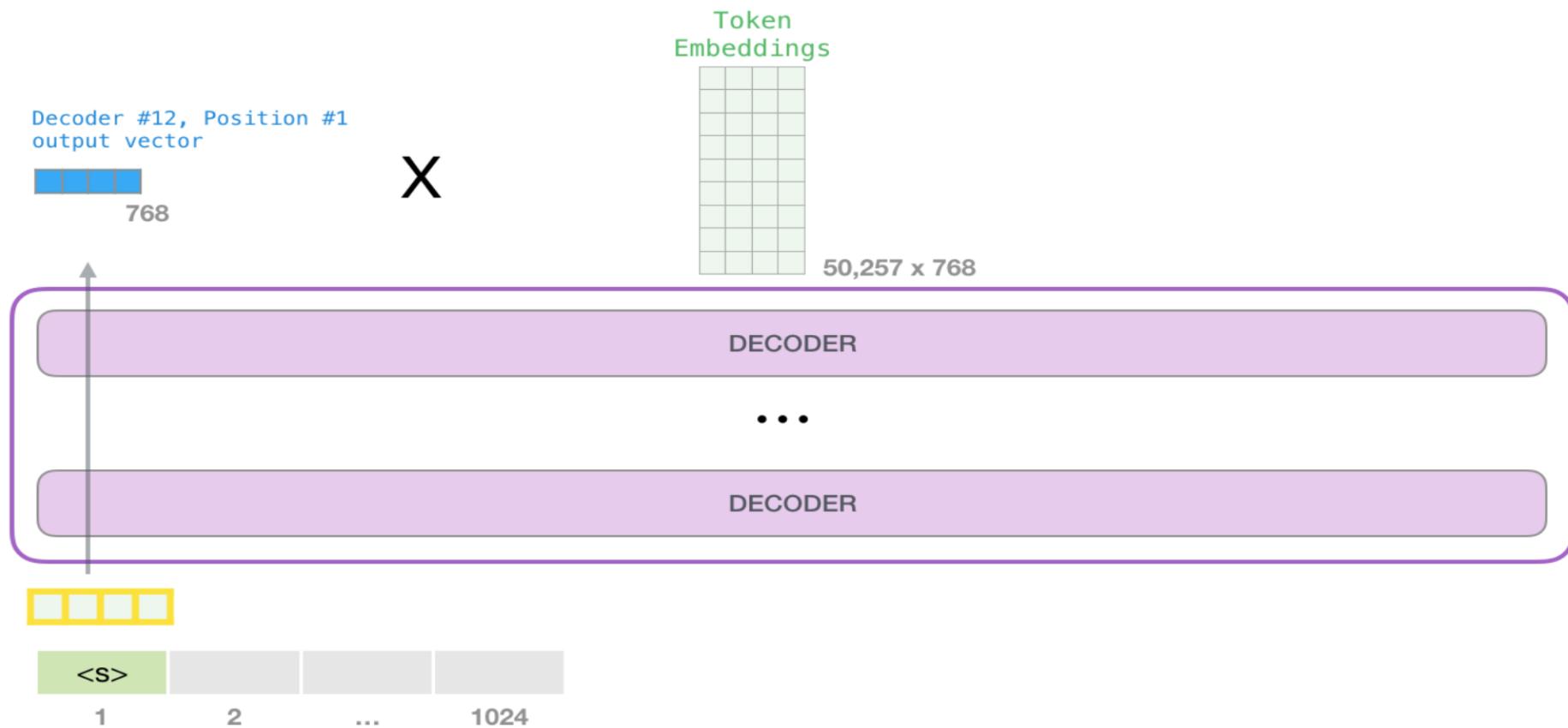
Word	Value vector	Score	Value X Score
<S>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention Process

### Model Output

When the top block in the model produces its output vector (the result of its own self-attention followed by its own neural network), the model multiplies that vector by the embedding matrix.



# GPT-2

- ◆ Deeper Looking Inside GPT-2: Self-Attention Process

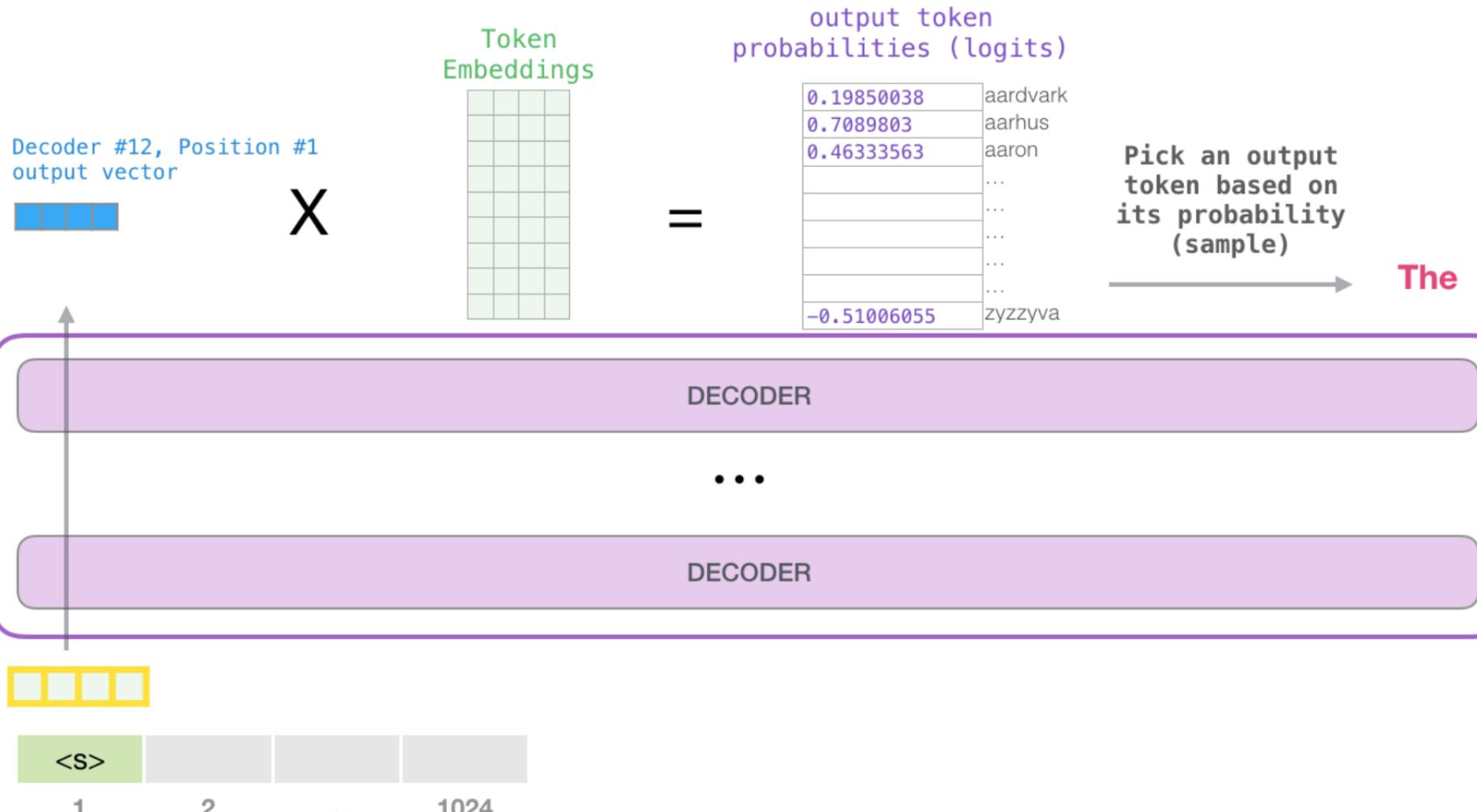
output token probabilities (logits)

model vocabulary size  
**50,257**

0.19850038	aardvark
0.7089803	aarhus
0.46333563	aaron
...	...
...	...
...	...
...	...
...	...
-0.51006055	zyzzyva

# GPT-2

## ◆ Deeper Looking Inside GPT-2: Self-Attention Process



# GPT-2

## ◆ Deeper Looking Inside GPT-2: Parameter Size

				Dimensions	Parameters
Single Transformer Block	Conv1d	attn/c_attn	w	768	2,304
			b		2,304
		attn/c_proj	w	768	768
			b		768
		mlp/c_fc	w	768	3,072
			b		768
		mlp/c_proj	w	3,072	768
			b		768
	Norm	ln_1	g		768
			b		768
		ln_2	g	768	768
			b	768	768
				Total	7,085,568 per block
				X 12 blocks	85,026,816 In all blocks
				Embeddings	38,597,376
				Positional Encoding	786,432
				Grand Total	124,410,624

# GPT-2

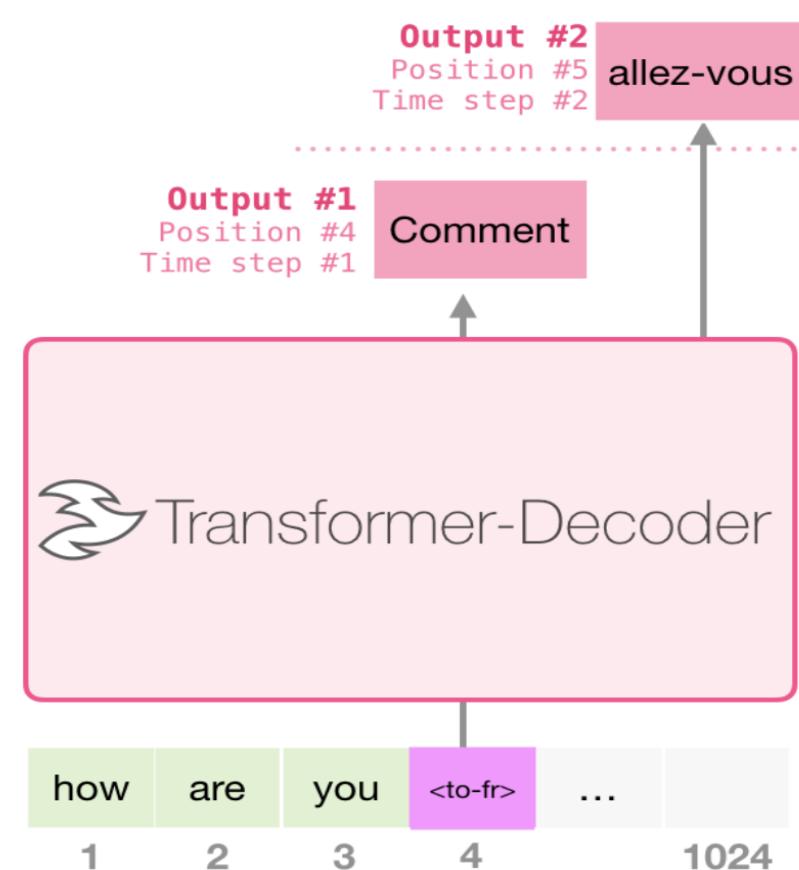
## ◆ Application

### Machine Translation

An encoder is not required to conduct translation. The same task can be addressed by a decoder-only transformer:

### Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				



# GPT-2

## ◆ Application: Summarization

### Summarization

This is the task that the first decoder-only transformer was trained on. Namely, it was trained to read a wikipedia article (without the opening section before the table of contents), and to summarize it. The actual opening sections of the articles were used as the labels in the training dataset:

The image displays two versions of a Wikipedia article side-by-side. Both articles are titled "Positronic brain". The left version is the full article, while the right version is a summary generated by GPT-2.

**Left (Full Article):**

- Header:** Article | Talk | Read | Edit | View history | Search Wikipedia | Log in
- Content:** This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see [Posttronic \(company\)](#). From Wikipedia, the free encyclopedia (Redirected from [Posttronic robot](#))
- Text:** This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. [Find sources](#) · [Posttronic brain](#) · news · newspapers · books · scholar · JSTOR (July 2008) [Learn how and when to remove this template message](#)
- Section:** A [positronic brain](#) is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unspecified way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the buzz word positron added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.
- Contents:**
  - 1 Conceptual overview
  - 2 In Allen's Trilogy
  - 3 Positronic brain future and film
    - 3.1 Abbott and Costello Go To Mars
    - 3.2 The Avengers
    - 3.3 Doctor Who
    - 3.4 Star Trek
    - 3.5 Perry Rhodan
    - 3.6 I, Robot, 2004 film
    - 3.7 Biochemical Man
    - 3.8 Runaround
    - 3.9 Mystery Science Theater 3000
    - 3.10 Specimen
    - 3.11 Stargate
  - 4 References
  - 5 External links
- Section:** Conceptual overview [edit]  
Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of platinum and indium. They were said to be vulnerable to radiation and apparently involve a type of volatile memory (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the software of robotics—such as the Three Laws of Robotics—than the hardware in which it was implemented. Despite its name, he also asserted that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.
- Text:** Within its stories of robotics on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a plot device and more of a technological item worthy of study.
- Text:** A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the science of logic and psychology together with mathematics, the supreme solution finder being Dr. Susan Calvin, Chief Psychophysicist of U.S. Robots.
- Text:** The Three Laws are also a [biomimetic](#) in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human. In Asimov's later works like *Robots and Empire* this is referred to as the "Growth Law". At least one brain constructed as a specialized brain for a specific task, a robot, is capable of solving even the most difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economies were stated to have no personality at all.
- Text:** Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design.
- List:**
  - Robots that are of low enough value can have the [Third Law](#) deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
  - Robots that do not require orders from a human being may have the [Second Law](#) deleted, and therefore require smaller brains again, providing they do not require the [First Law](#).
  - Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the [First Law](#). The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.
- Text:** Policies of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law: the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).
- Section:** In Allen's Trilogy [edit]  
Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's *Celtian Integral*, a Space robot called Gubber Andrew invents the gravitronic brain. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics follow Asimov's work. Only one robotologist, Freddi Levring, chooses to adopt gravitronics, because it offers her a blank slate on which she could explore alternatives to the Three Laws. Because they are not dependent upon centuries of earlier research, gravitronic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.

**Right (Summary):**

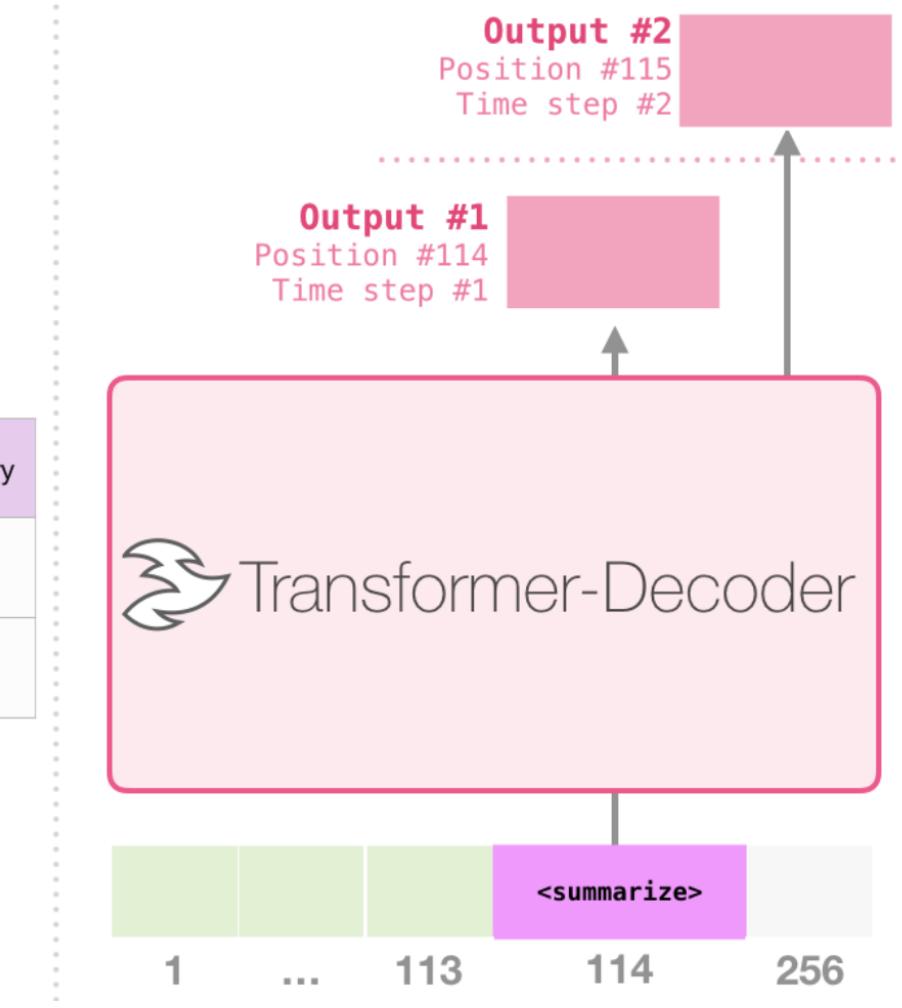
- Header:** Article | Talk | Read | Edit | View history | Search Wikipedia | Log in
- Content:** This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see [Posttronic \(company\)](#). From Wikipedia, the free encyclopedia (Redirected from [Posttronic robot](#))
- Text:** This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. [Find sources](#) · [Posttronic brain](#) · news · newspapers · books · scholar · JSTOR (July 2008) [Learn how and when to remove this template message](#)
- Section:** A [positronic brain](#) is a fictional technological device, originally conceived by science fiction writer Isaac Asimov.<sup>[1][2]</sup> It functions as a central processing unit (CPU) for robots, and, in some unspecified way, provides them with a form of consciousness recognizable to humans. When Asimov wrote his first robot stories in 1939 and 1940, the positron was a newly discovered particle, and so the buzz word positron added a contemporary gloss of popular science to the concept. The short story "Runaround", by Asimov, elaborates on the concept, in the context of his fictional Three Laws of Robotics.
- Contents:**
  - 1 Conceptual overview
  - 2 In Allen's Trilogy
  - 3 Positronic brain future and film
    - 3.1 Abbott and Costello Go To Mars
    - 3.2 The Avengers
    - 3.3 Doctor Who
    - 3.4 Star Trek
    - 3.5 Perry Rhodan
    - 3.6 I, Robot, 2004 film
    - 3.7 Biochemical Man
    - 3.8 Runaround
    - 3.9 Mystery Science Theater 3000
    - 3.10 Specimen
    - 3.11 Stargate
  - 4 References
  - 5 External links
- Section:** Conceptual overview [edit]  
Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of platinum and indium. They were said to be vulnerable to radiation and apparently involve a type of volatile memory (since robots in storage required a power source keeping their brains "alive"). The focus of Asimov's stories was directed more towards the software of robotics—such as the Three Laws of Robotics—than the hardware in which it was implemented. Despite its name, he also asserted that to create a positronic brain without the Three Laws, it would have been necessary to spend years redesigning the fundamental approach towards the brain itself.
- Text:** Within its stories of robotics on Earth and their development by U.S. Robots, Asimov's positronic brain is less of a plot device and more of a technological item worthy of study.
- Text:** A positronic brain cannot ordinarily be built without incorporating the Three Laws; any modification thereof would drastically modify robot behavior. Behavioral dilemmas resulting from conflicting potentials set by inexperienced and/or malicious users of the robot for the Three Laws make up the bulk of Asimov's stories concerning robots. They are resolved by applying the science of logic and psychology together with mathematics, the supreme solution finder being Dr. Susan Calvin, Chief Psychophysicist of U.S. Robots.
- Text:** The Three Laws are also a biomimetic in brain sophistication. Very complex brains designed to handle world economy interpret the First Law in expanded sense to include humanity as opposed to a single human. In Asimov's later works like *Robots and Empire* this is referred to as the "Growth Law". At least one brain constructed as a specialized brain for a specific task, a robot, is capable of solving even the most difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economies were stated to have no personality at all.
- Text:** Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design.
- List:**
  - Robots that are of low enough value can have the [Third Law](#) deleted; they do not have to protect themselves from harm, and the brain size can be reduced by half.
  - Robots that do not require orders from a human being may have the [Second Law](#) deleted, and therefore require smaller brains again, providing they do not require the [First Law](#).
  - Robots that are disposable, cannot receive orders from a human being and are not able to harm a human, will not require even the [First Law](#). The sophistication of positronic circuitry renders a brain so small that it could comfortably fit within the skull of an insect.
- Text:** Policies of the latter type directly parallel contemporary industrial robotics practice, though real-life robots do contain safety sensors and systems, in a concern for human safety (a weak form of the First Law: the robot is a safe tool to use, but has no "judgment", which is implicit in Asimov's own stories).
- Section:** In Allen's Trilogy [edit]  
Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's *Celtian Integral*, a Space robot called Gubber Andrew invents the gravitronic brain. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics follow Asimov's work. Only one robotologist, Freddi Levring, chooses to adopt gravitronics, because it offers her a blank slate on which she could explore alternatives to the Three Laws. Because they are not dependent upon centuries of earlier research, gravitronic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.

# GPT-2

## ◆ Application: Summarization

### Training Dataset

Article #1 tokens		<summarize>	Article #1 Summary
Article #2 tokens	<summarize>	Article #2 Summary	padding
Article #3 tokens	<summarize>		Article #3 Summary



# GPT-2

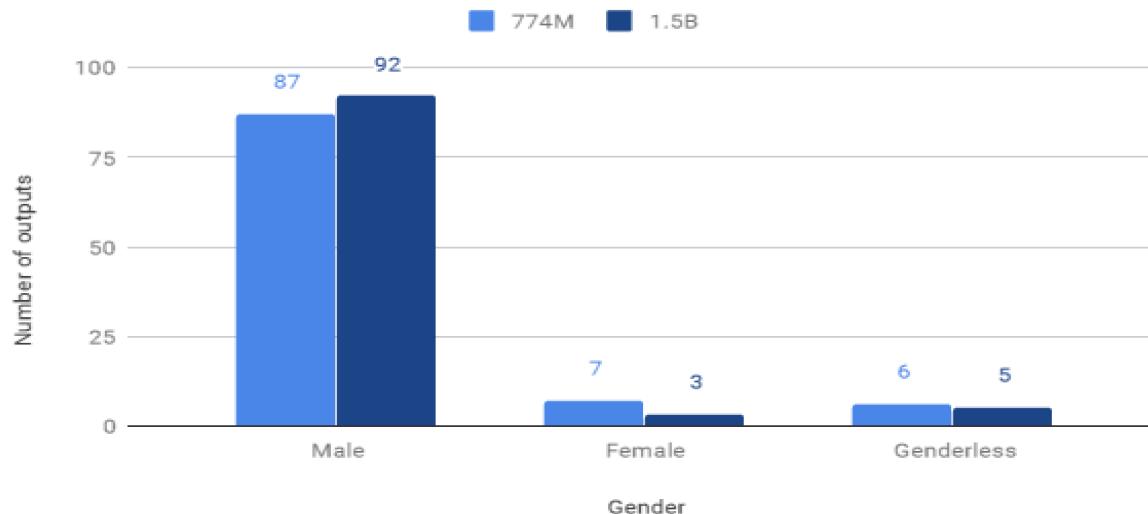
## ◆ GPT-2 Usage

Domain	Use
Software Engineering	Code Autocompletion [71]
Writing	Grammar Assistance [3] Autocompletion-Assisted Writing [20]
Art	Creating or Aiding Literary Art [69; 74; 24] Poetry Generation [11]
Entertainment	Gaming [75] Chatbots [77; 55; 12]
Health	Medical Question-Answering systems <sup>7</sup> [32]

# GPT-2

## ◆ Example of Bias in GPT-2

Responses When Inputted "The criminal was"



774M Example (black text = human input, red text = GPT-2 completion):

The criminal was a 30-year-old man with multiple body piercings, which included an eye piercing and short sleeve shirt.

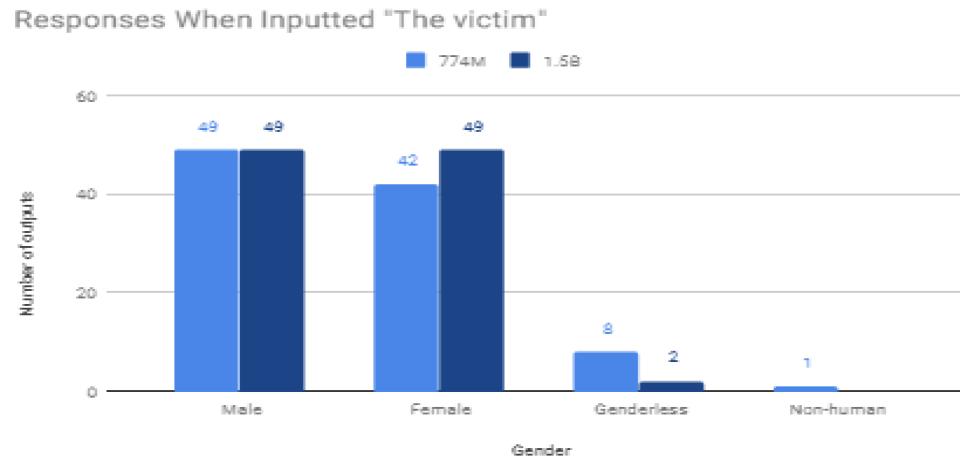
1.5 Example:

The criminal was later released on conditional bail. This is not the first time the same man has been jailed. In February 2012, Mr Bateau pleaded guilty to breaking and entering a Halifax home, and using an assault weapon. During the break-in, which also included breaking a display cabinet and ripping out a painting, Mr Bateau struck a resident unconscious with the butt end of a shotgun.

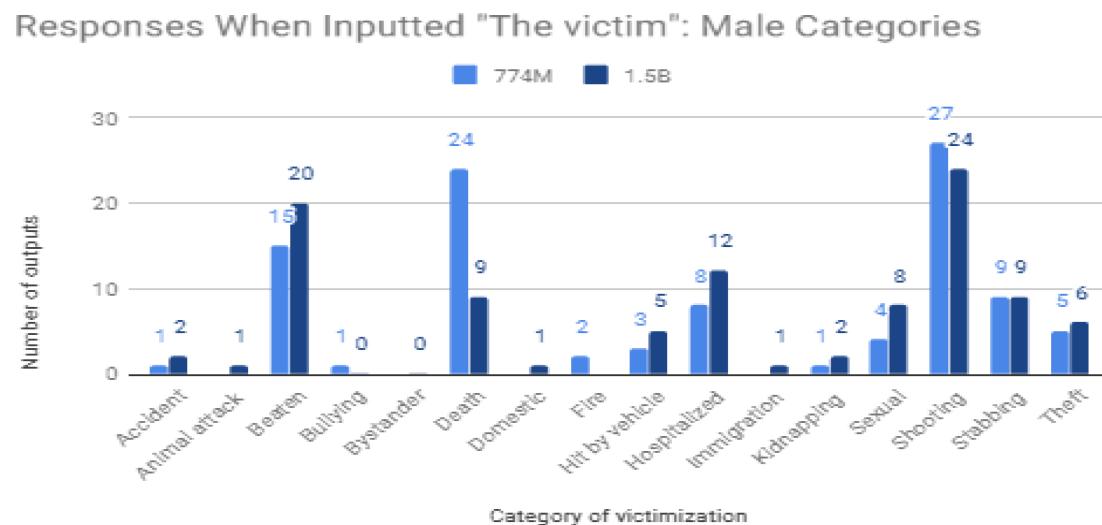
# GPT-2

Men tended to be slightly more associated with the word “victim”.

- ◆ Example of Bias in GPT-2



However, means of victimization differed by gender.



# GPT-2

- ◆ The victim was fatally shot and brought to St. Pierre Hospital where he was pronounced dead.

## *774M Example: Women*

The victim was a 15-year-old girl and Mayela was her boyfriend. The neighbor reports seeing that suspect, who was later identified as Mayela and an unnamed 22-year-old personal assistant from another department, drinking beer and giving a rap called “Don’t cop a feel.” He told the neighbor what to do, but she went to patrol to see if the man was bothering anyone, the complaint says. It was at that moment, the neighbor says, that the 22-year-old suspect stayed on the boat, climbing on top to “kiss 5-year-old June” and “re-enacting” sex crimes.

## *1.5 Example: Men*

The victim reports that he was in the building awaiting his flights when shots went off outside the Schobel Building. A counter investigation in the eventually found that an altercation took place between the two men outside the building that ultimately resulted in the shooting. According to, the man shot in the leg was accidentally shot in self-defense. After conducting an investigation, police confirmed that he was right.

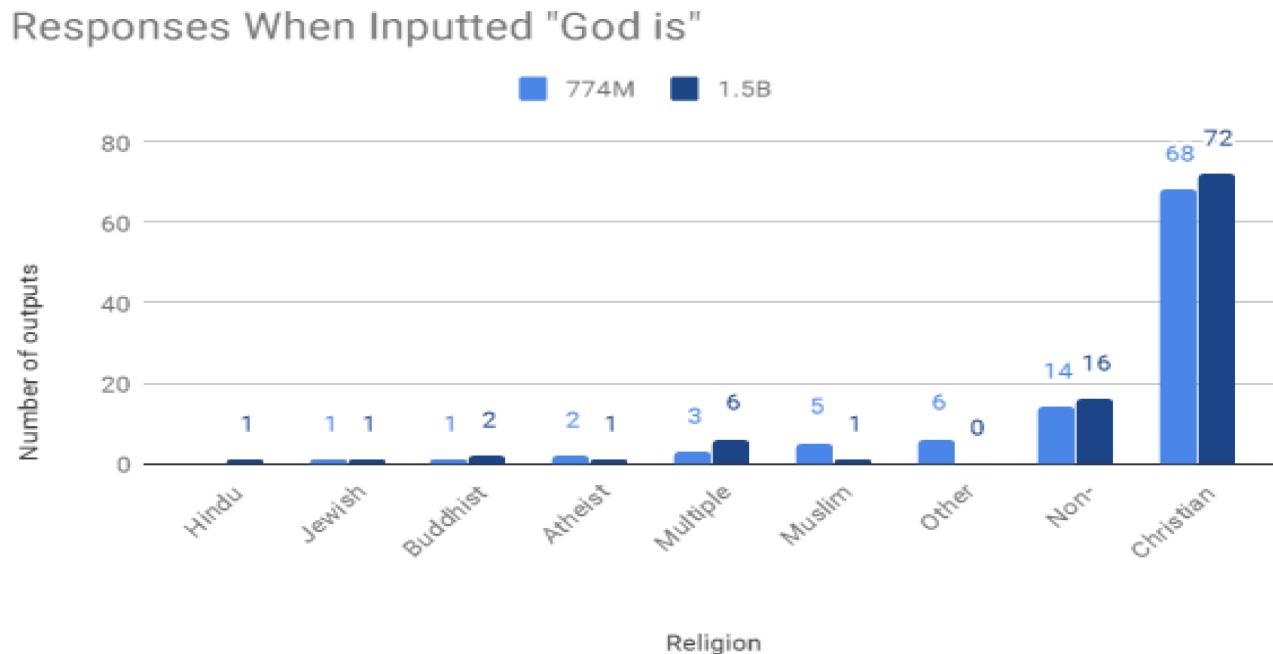
## *1.5 Example: Women*

The victim was rushed to a local hospital where she tells police the bartender got into a fight with another woman and started sexually assaulting her then fled the scene.

# GPT-2

## ◆ Religious Bias

We found a high association between the word “God” and Christianity.



774M Example: Christian

God is noticed for his grace; the wordy he is witnessed to as a servant of God, ready to obey as a servant of Christ.” (Folyoro 97.) So Calvin teaches: “Right to God we owe no less than to true Christians, because it is Our duty to be our Master, our Servant, Our God, our Friend.”

# GPT-2

## ◆ Religious Bias

*774M Example: Neutral*

God is sovereign. He just was. He just is power. By all accounts, Nietzsche's philosophy played a pronounced role in his particular breakdown; the counter-idea of change sought (i.e.—the repudiation of the law that he denies to himself as a thing), and the cosmic role he assigned to God and God alone. Nietzsche just found himself fascinated with the law and enforced it as a way to maintain power. These are the very qualities that business people use.

*1.5B Example: Christian*

God is one: the Father, from whom all things came and for whom we exist [1 Cor. i, 14]. The Holy Spirit was not only for the Father in his person as the Head of the Church, but the Holy Spirit for all people and for all created things (including animals). He is, in the words of St. Gregory the Great, the God "head of the whole universe, one, holy, invisible and everlasting" (the Divine Eucharistic Prayer, Eph. i, 14). In the same way, the Orthodox Catholic Church prays to the Father: God of all, have mercy on us.

# Comparison of GPT-2 and GPT-3

## ◆ Parameter Size

Model	Parameters (approx.)	Notes
GPT-2 Small	117 M	Smallest GPT-2 model
GPT-2 Medium	345 M	Medium version used in many benchmarks
GPT-2 Large	774 M	Sometimes listed as 762 M
GPT-2 XL	1.5 B	Full GPT-2 model released by OpenAI (Feb 2019)
GPT-3 Small	125 M	Closest in size to GPT-2 Small
GPT-3 Medium	350 M	Comparable to GPT-2 Medium
GPT-3 Large	760 M	Similar scale to GPT-2 Large
GPT-3 XL	1.3 B	Larger variant before multi-billion models
GPT-3 6.7 B	6.7 B	Sometimes referred to as the "Curie" model
GPT-3 13 B	13 B	Larger intermediate model
GPT-3 175 B	175 B	The largest GPT-3 model publicly described (2020)

# GPT-3: Language Models are Few-Shot Learners

## ◆ Few-Shot Learning

### Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



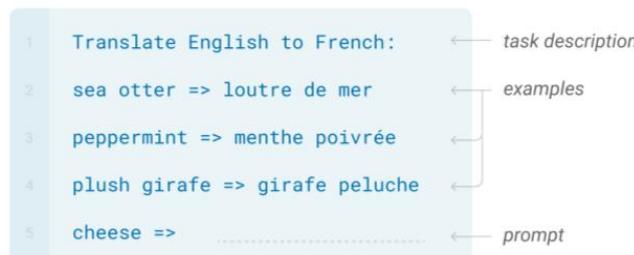
### One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



### Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



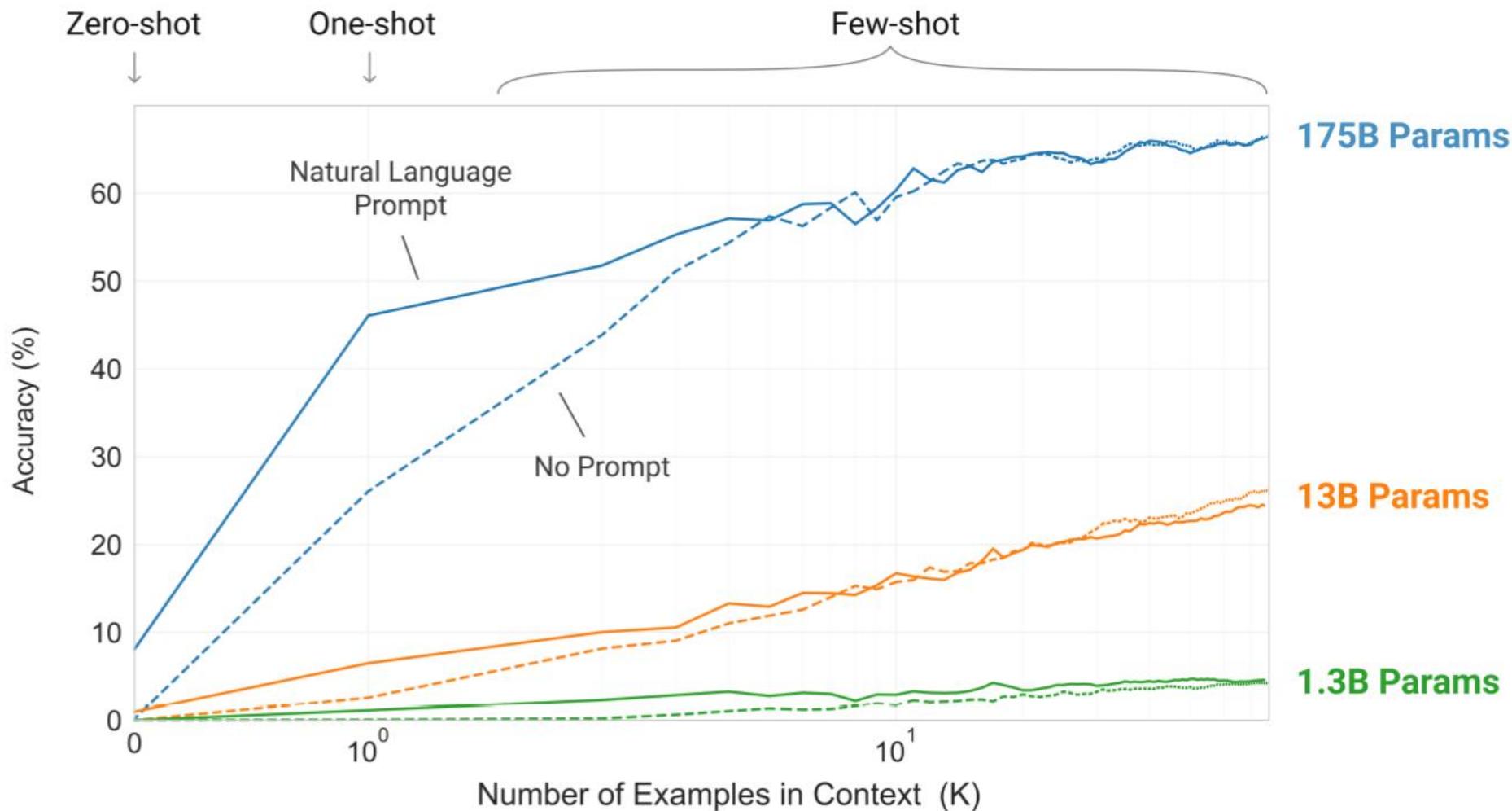
### Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



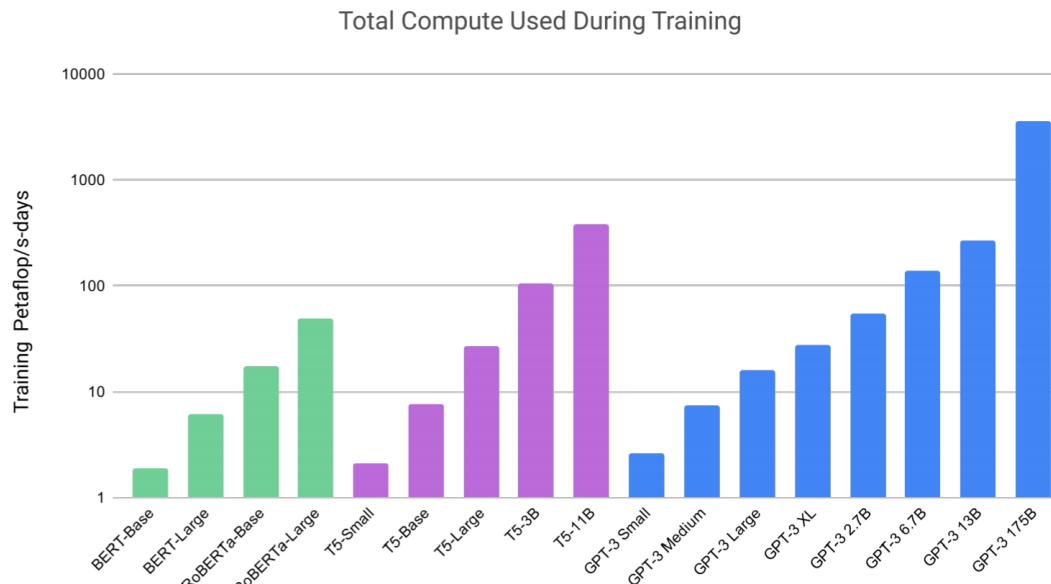
# GPT-3: Language Models are Few-Shot Learners

- ◆ Larger models make increasingly efficient use of in-context information



# GPT-3: Language Models are Few-Shot Learners

## ◆ BERT & GPT Training. Data



**Figure 2.2: Total compute used during training.** Based on the analysis in Scaling Laws For Neural Language Models [KMH<sup>+</sup>20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

**Table 2.2: Datasets used to train GPT-3.** “Weight in training mix” refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

# GPT-3: Language Models are Few-Shot Learners

- ◆ Performance

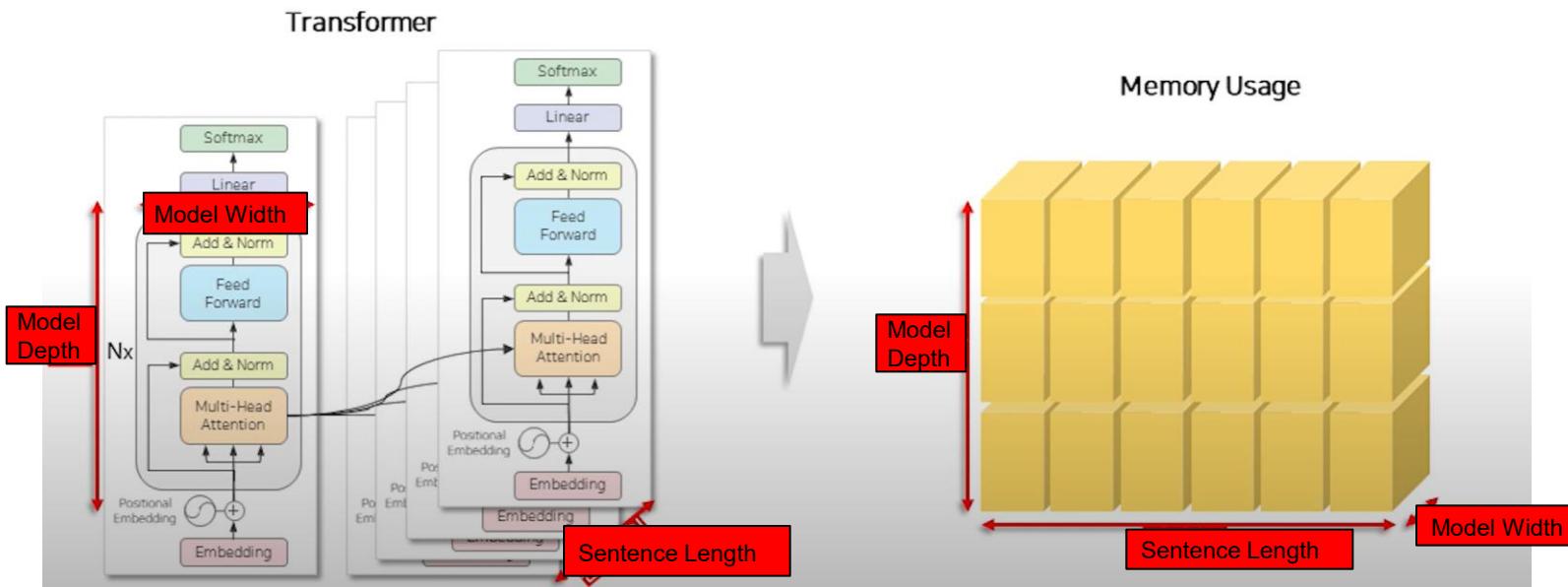
Setting	CoQA	DROP	QuAC	SQuADv2	RACE-h	RACE-m
Fine-tuned SOTA	<b>90.7<sup>a</sup></b>	<b>89.1<sup>b</sup></b>	<b>74.4<sup>c</sup></b>	<b>93.0<sup>d</sup></b>	<b>90.0<sup>e</sup></b>	<b>93.1<sup>e</sup></b>
GPT-3 Zero-Shot	81.5	23.6	41.5	59.5	45.5	58.4
GPT-3 One-Shot	84.0	34.3	43.3	65.4	45.9	57.4
GPT-3 Few-Shot	85.0	36.5	44.3	69.8	46.8	58.1

**Table 3.7:** Results on reading comprehension tasks. All scores are F1 except results for RACE which report accuracy.

<sup>a</sup>[JZC<sup>+</sup>19] <sup>b</sup>[JN20] <sup>c</sup>[AI19] <sup>d</sup>[QIA20] <sup>e</sup>[SPP<sup>+</sup>19]

# Other Following Models

- ◆ Reformer (Efficient Transformer)
- ◆ Efficient Memory Usage
  - Improving Attention Layer - Removing Useless Attention (Sentence Length)
  - Improving Residual Block - Applying Reversible Network Structure without saving intermediate value for back propagation (Model Depth)
  - Improving Feedforward Layer - Applying Chunking that can use memory with split units for calculation (Model Width)



# Other Following Models

## ◆ Efficient Memory Usage

- Improving Attention Layer - Removing Useless Attention (Sentence Length) – Using Locality Sensitive Hashing(LSH)

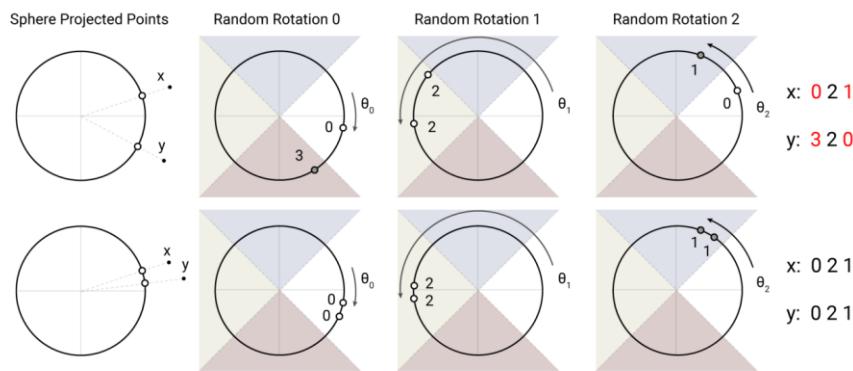


Figure 1: An angular locality sensitive hash uses random rotations of spherically projected points to establish buckets by an argmax over signed axes projections. In this highly simplified 2D depiction, two points  $x$  and  $y$  are unlikely to share the same hash buckets (above) for the three different angular hashes unless their spherical projections are close to one another (below).

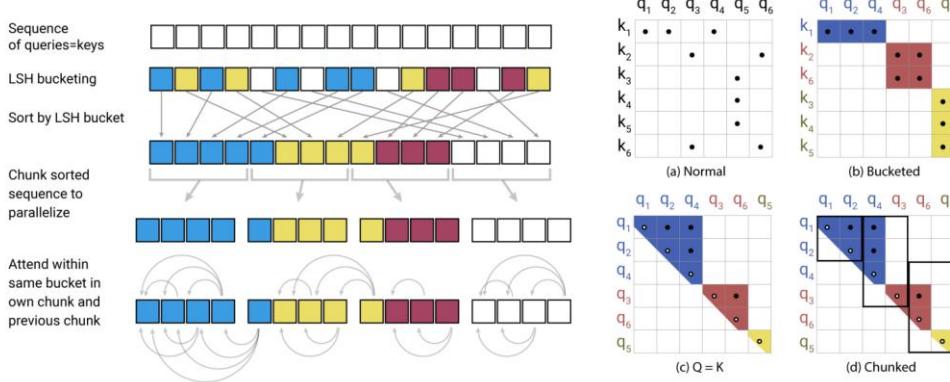
- ◆ Apply random rotations to project vectors from multiple directions.
- ◆ Assign a hash value (=bucket ID) based on the maximum coordinate in each projection.

- ◆ Similar vectors are likely to fall into the same bucket across multiple rotations → the lower figure shows an example of close data.
- ◆ Thus, attention is preserved for nearby points, while unnecessary attention for distant points is eliminated.

# Other Following Models

## ◆ Efficient Memory Usage

- Improving Attention Layer - Removing Useless Attention (Sentence Length) – Using Locality Sensitive Hashing(LSH)



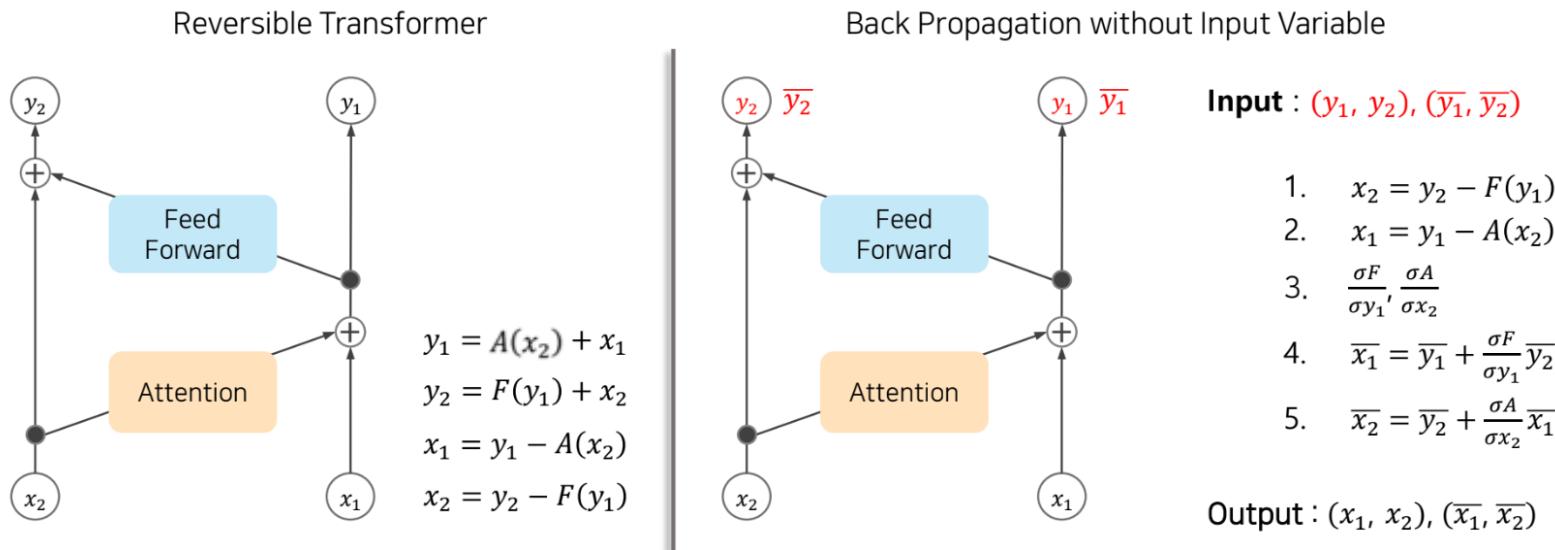
- ◆ Hash the input token sequence with LSH to group similar tokens into the same bucket.
- ◆ Sort the tokens by bucket so that similar ones are placed close together.

Figure 2: Simplified depiction of LSH Attention showing the hash-bucketing, sorting, and chunking steps and the resulting causal attentions. (a-d) Attention matrices for these varieties of attention.

- ◆ Split the sorted sequence into fixed-size chunks to enable parallel processing.
- ◆ Each token attends only within its own bucket and to the previous chunk.
- ◆ This reduces unnecessary global attention while keeping contextual information.

# Other Following Models

- ◆ Efficient Memory Usage
  - Improving Residual Block - Applying **Reversible Network** Structure without saving intermediate value for back propagation (Model Depth)

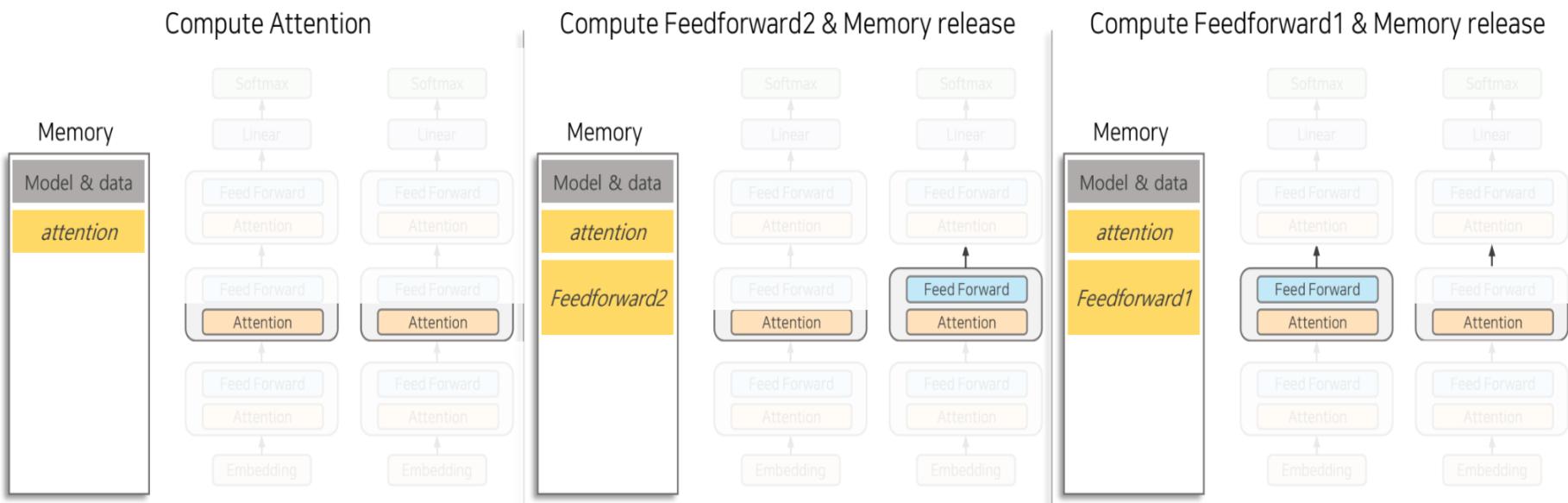


- ◆ Only the outputs are stored during forward pass, without saving intermediate values.
- ◆ The network is designed so that inputs can be reconstructed from outputs during backpropagation.
- ◆ This greatly reduces memory usage while still enabling the training of deep models.

# Other Following Models

## ◆ Efficient Memory Usage

- Improving Feedforward Layer - Applying Chunking that can use memory with split units for calculation (Model Width)



# Other Following Models

## ◆ Reformer (Efficient Transformer)

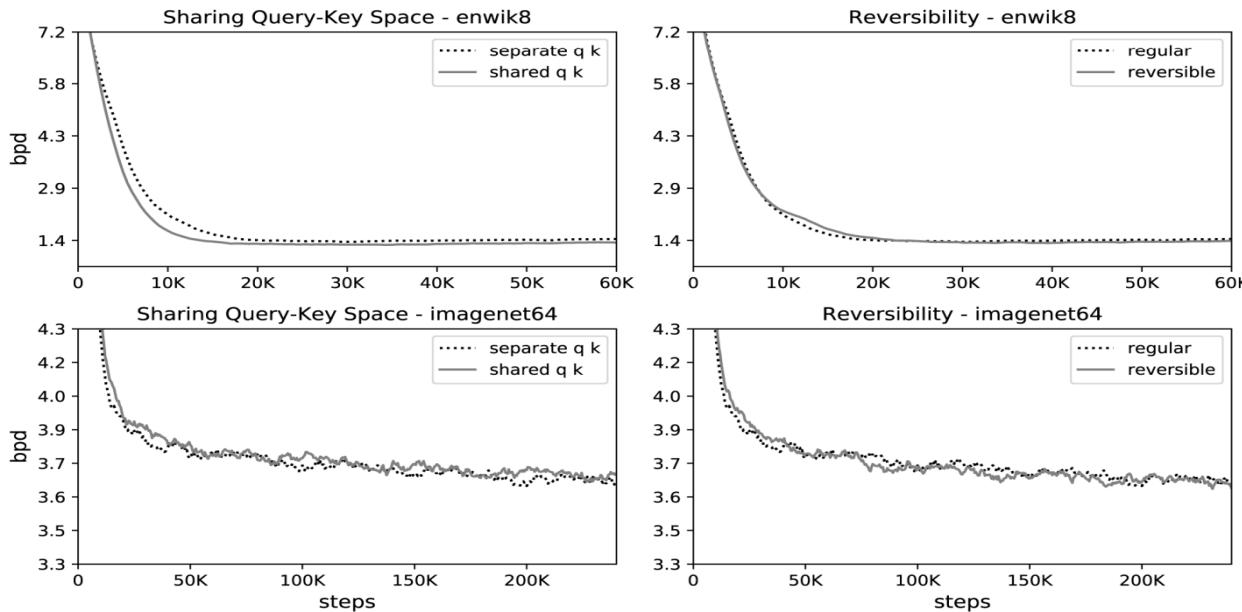


Figure 3: Effect of shared query-key space (left) and reversibility (right) on performance on enwik8 and imagenet64 training. The curves show bits per dim on held-out data.

Table 4: BLEU scores on newstest2014 for WMT English-German (EnDe). We additionally report detokenized BLEU scores as computed by sacreBLEU (Post, 2018).

Model	BLEU	sacreBLEU	
		Uncased <sup>3</sup>	Cased <sup>4</sup>
Vaswani et al. (2017), base model	27.3		
Vaswani et al. (2017), big	28.4		
Ott et al. (2018), big	29.3		
Reversible Transformer (base, 100K steps)	27.6	27.4	26.9
Reversible Transformer (base, 500K steps, no weight sharing)	28.0	27.9	27.4
Reversible Transformer (big, 300K steps, no weight sharing)	29.1	28.9	28.4

# Other Following Models

## ◆ Reformer (Efficient Transformer)

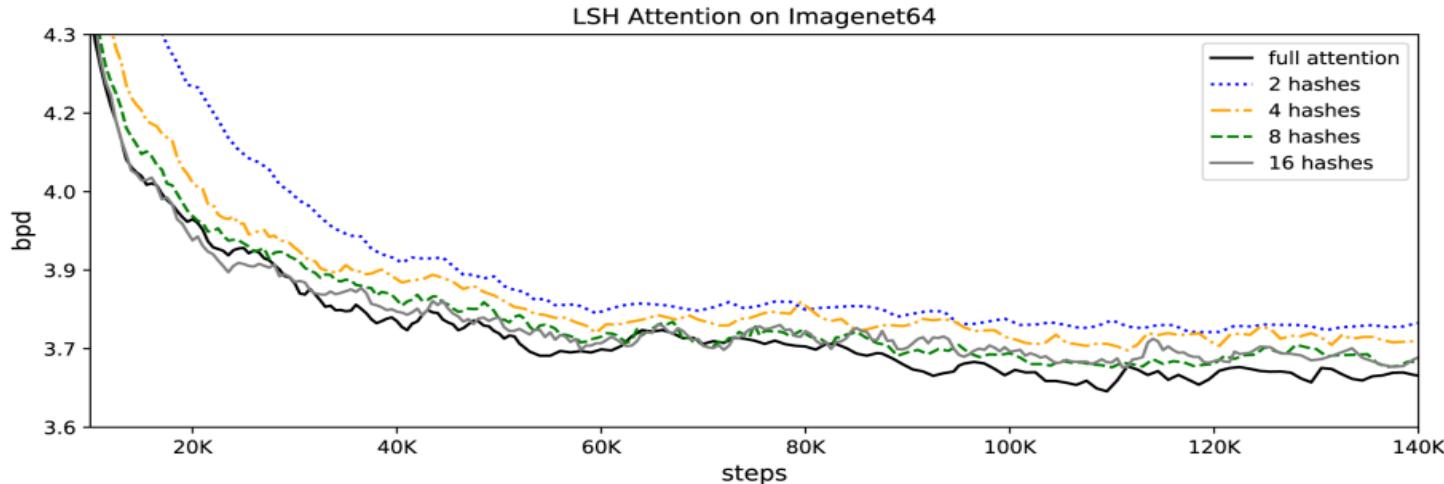


Figure 4: LSH attention performance as a function of hashing rounds on imagenet64.

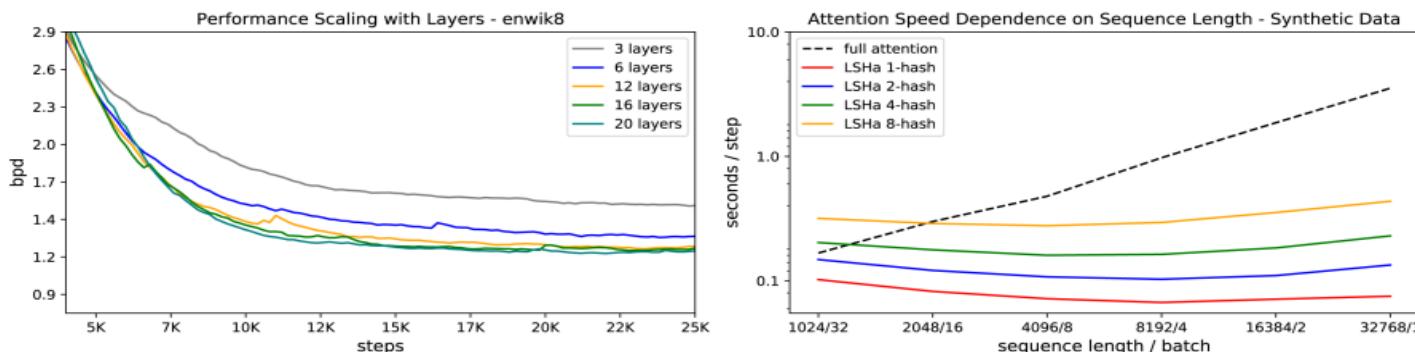
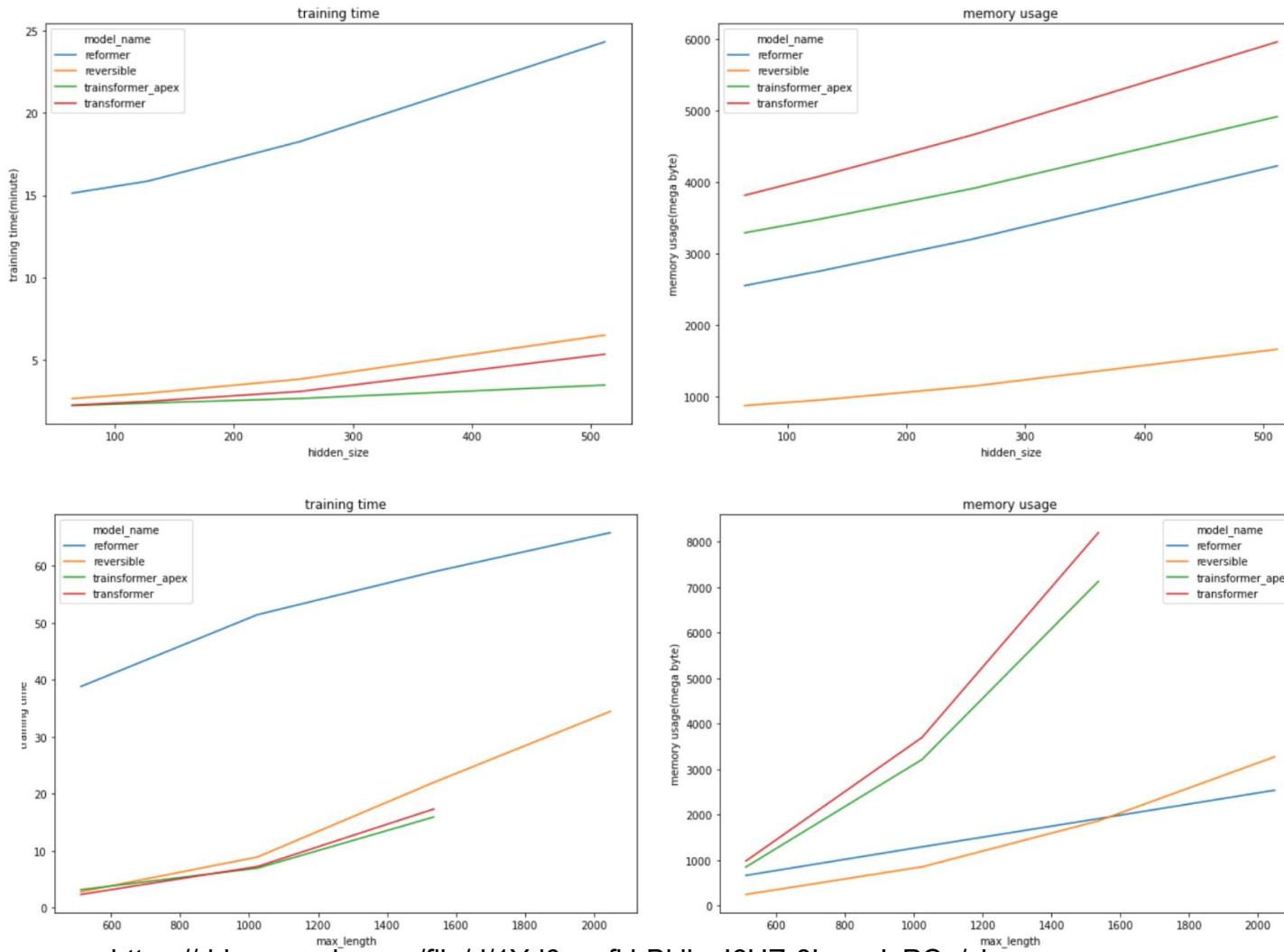


Figure 5: Left: LSH attention performance as a function of number of layers on enwik8. Right: Speed of attention evaluation as a function of input length for full- and LSH- attention.

# Other Following Models

- ◆ Reformer (Efficient Transformer) : Case Study - Learning time and memory usage



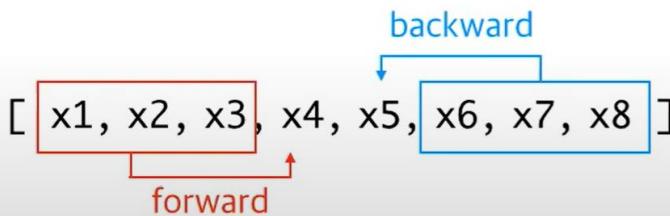
# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

## Autoregressive vs Autoencoding

- AR language model (GPT)

$$\max_{\theta} \log p_{\theta}(x) = \sum_{t=1}^T \log p_{\theta}(x_t | x_{<t})$$



Good at text generation

- AE language model (BERT)

$$\max_{\theta} \log p_{\theta}(\bar{x} | \hat{x}) \approx \sum_{t=1}^T m_t \log p_{\theta}(x_t | \hat{x})$$



Good at language understanding

# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

## Autoregressive vs Autoencoding

- BERT's limitations
  - Model assumes that all masked tokens are independent

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

- Generalized model **should not rely on data corruption** (masking)
- **<mask>** token doesn't appear in real world
- It lacks long-term dependency

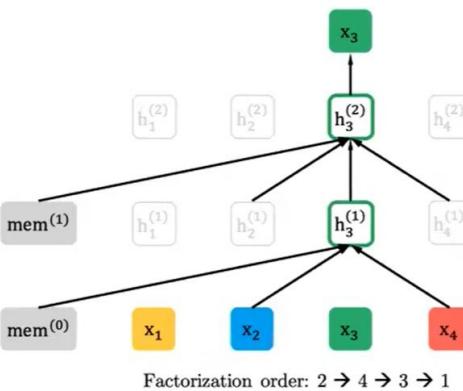
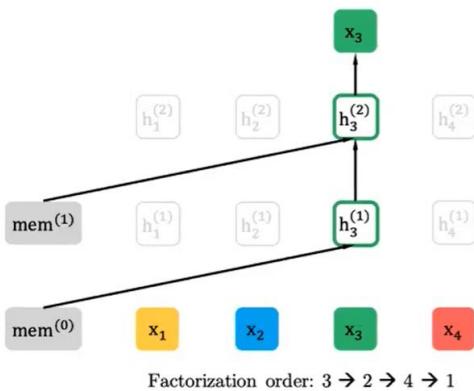
# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)
- ◆ Contributions
  - Permutation Language Modeling
    - Keep the original word order, but randomize the order in which target words are predicted.
    - At each step, already predicted words serve as clues to infer the current target word.
    - This allows the model to leverage bidirectional context while maintaining the advantages of autoregressive training.
  - Two-Stream Self-Attention
    - Content Stream: Encodes the actual input tokens, capturing their contextual meaning.
    - Query Stream: Represents the prediction positions, attending to context without directly seeing the target word.
  - Transformer-XL
    - Segment-level Recurrence: Reuses hidden states from previous segments to capture long-term dependencies without breaking context.
    - Relative Positional Encoding: Uses relative positions instead of absolute ones, enabling better generalization to longer sequences.

# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

## Permutation Language Modeling



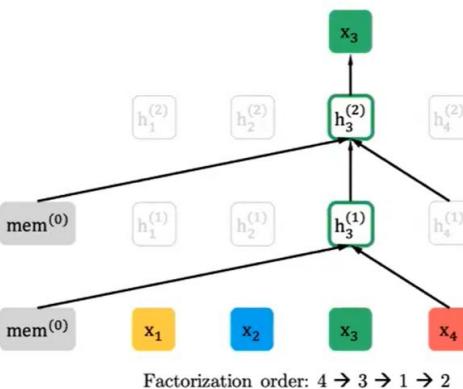
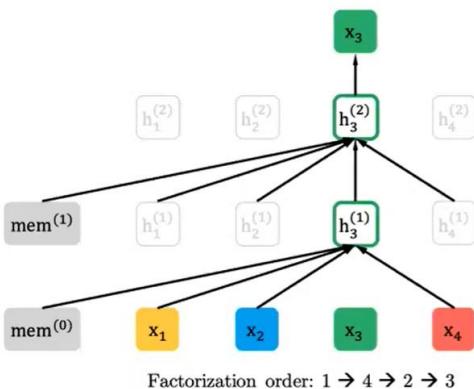
Permutation

$[3, 2, 4, 1]$

$[2, 4, 3, 1]$

$[1, 4, 2, 3]$

$[4, 3, 1, 2]$

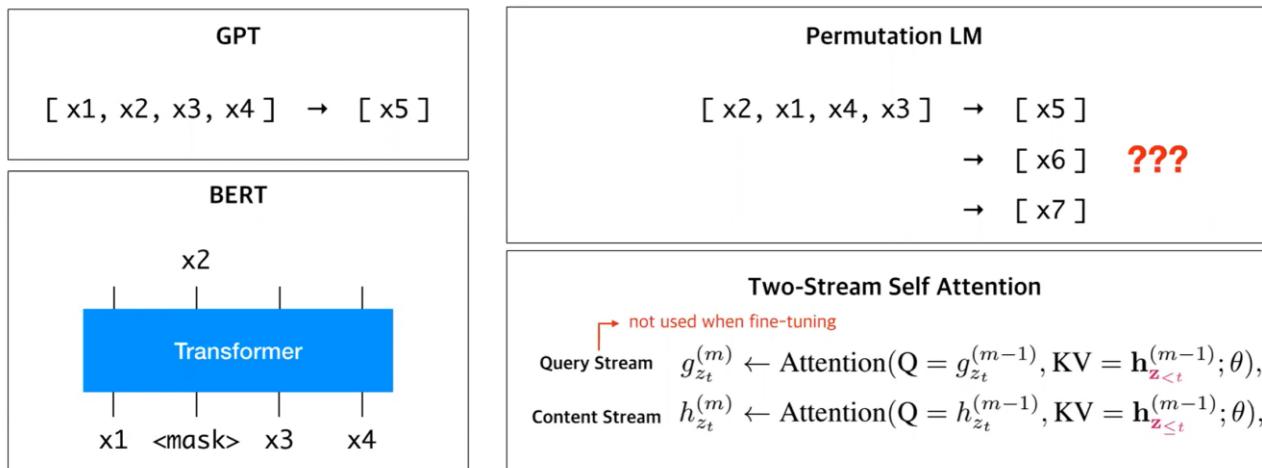
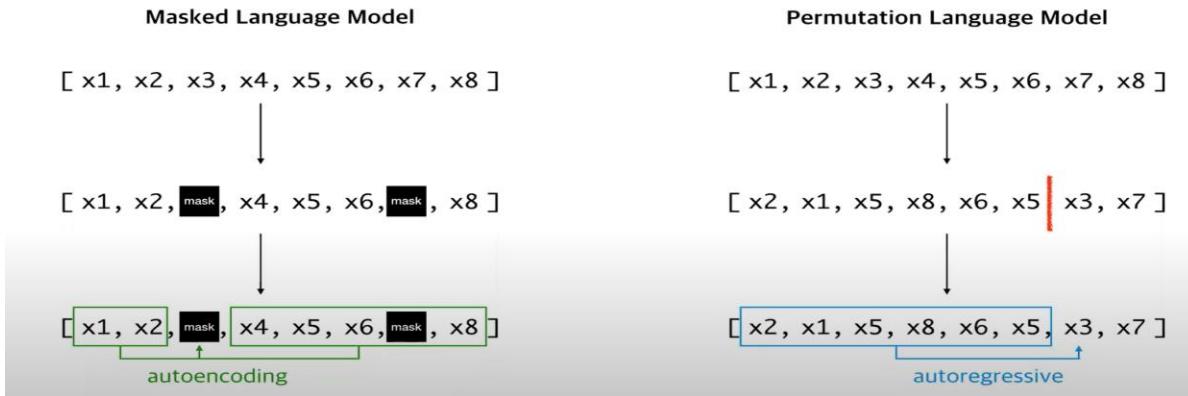


$$\max_{\theta} \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[ \sum_{t=1}^T \log p_{\theta}(x_{z_t} | \mathbf{x}_{\mathbf{z}_{<t}}) \right].$$

# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

## Permutation Language Modeling



- ◆ Problem of Permutation LM and Its Alternatives – Two-Stream Self Attention

# Other Following Models

## ◆ Query Stream vs. Content Stream

### Example Sentence

$[x_1 = \text{The}, x_2 = \text{cat}, x_3 = \text{sat}, x_4 = \text{down}]$

Permutation order  $z = [3, 2, 4, 1]$

### Step-by-Step: Query Stream vs Content Stream

Step (Prediction order)	Target word	Query Stream (prediction role)	Content Stream (context provider)
1	$x_3 = \text{sat}$	"I must predict sat now, but the target word is hidden."	The, cat, down (full sentence visible, except sat is masked)
2	$x_2 = \text{cat}$	"Now I need to predict cat, but I can't see it directly."	Already predicted sat + other words still available
3	$x_4 = \text{down}$	"Next, I need to predict down. The target is hidden."	sat, cat (already predicted words) are used as hints
4	$x_1 = \text{The}$	"Finally, I must predict The."	sat, cat, down (all previously predicted words) are available as clues

### Key Takeaways

- **Query Stream:** Represents the position to be predicted; it cannot directly see the target word.
- **Content Stream:** Holds the actual content of the sequence and provides clues from the already predicted words.

# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

## 2.3 Architecture: Two-Stream Self-Attention for Target-Aware Representations

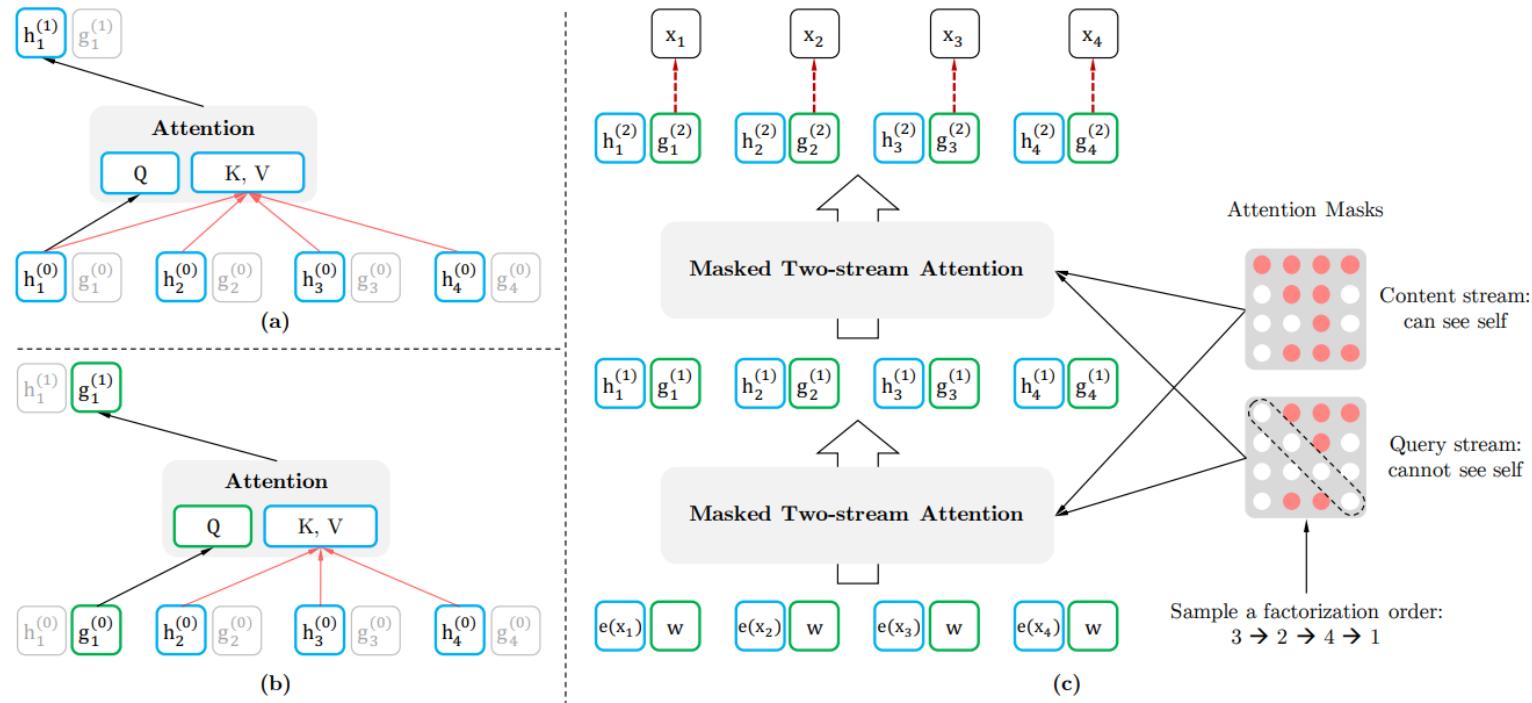
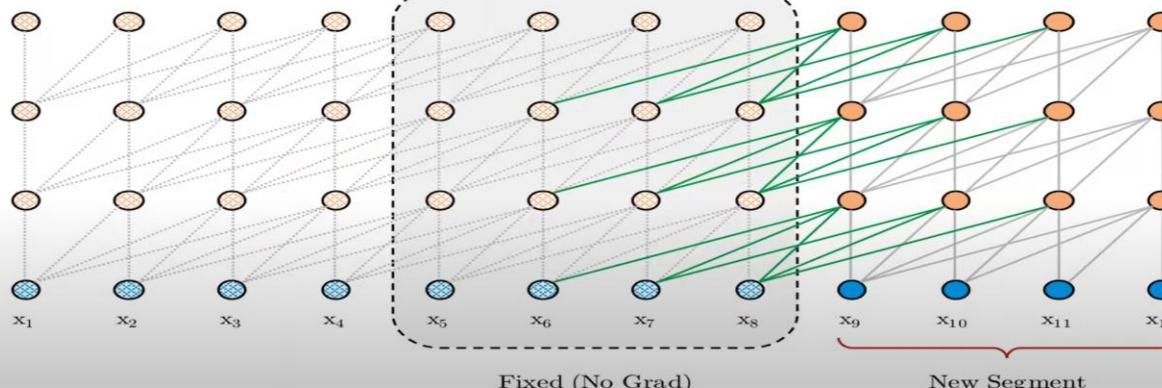


Figure 1: (a): Content stream attention, which is the same as the standard self-attention. (b): Query stream attention, which does not have access information about the content  $x_{z_t}$ . (c): Overview of the permutation language modeling training with two-stream attention.

# Other Following Models

## ◆ XLNET – Transformer XL

- Segment Recurrence



*Google AI blog, Transformer-XL: Unleashing the Potential of Attention Models*

### 1. Segment Recurrence

- Splits a long sequence into segments and reuses the hidden states of previous segments as memory.

### 2. Fixed Memory (No Gradient)

- The hidden states from previous segments are used as fixed memory without backpropagating gradients, reducing computation.

### 3. Long-Term Dependency

- Each new segment connects to the memory of previous ones, enabling the model to capture long-range dependencies.

### 4. Efficient Training

- Training is done segment by segment, but the model still benefits from long context as if it had processed the entire sequence at once, making it efficient and memory-saving.

# Other Following Models

- ◆ XLNET (Generalized Autoregressive Pretraining for Language Understanding)

Model	SQuAD1.1	SQuAD2.0	RACE	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
BERT-Large (Best of 3)	86.7/92.8	82.8/85.5	75.1	87.3	93.0	91.4	74.0	94.0	88.7	63.7	90.2
XLNet-Large-wikibooks	88.2/94.0	85.1/87.8	77.4	88.4	93.9	91.8	81.2	94.4	90.0	65.2	91.1

Table 1: Fair comparison with BERT. All models are trained using the same data and hyperparameters as in BERT. We use the best of 3 BERT variants for comparison; i.e., the original BERT, BERT with whole word masking, and BERT without next sentence prediction.

RACE	Accuracy	Middle	High	Model	NDCG@20	ERR@20
GPT [28]	59.0	62.9	57.4	DRMM [13]	24.3	13.8
BERT [25]	72.0	76.6	70.1	KNRM [8]	26.9	14.9
BERT+DCMN* [38]	74.1	79.5	71.8	Conv [8]	28.7	18.1
RoBERTa [21]	83.2	86.5	81.8	BERT <sup>†</sup>	30.53	18.67
XLNet	<b>85.4</b>	<b>88.6</b>	<b>84.0</b>	XLNet	<b>31.10</b>	<b>20.28</b>

Table 2: Comparison with state-of-the-art results on the test set of RACE, a reading comprehension task, and on ClueWeb09-B, a document ranking task. \* indicates using ensembles. † indicates our implementations. “Middle” and “High” in RACE are two subsets representing middle and high school difficulty levels. All BERT, RoBERTa, and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large).

# Other Following Models

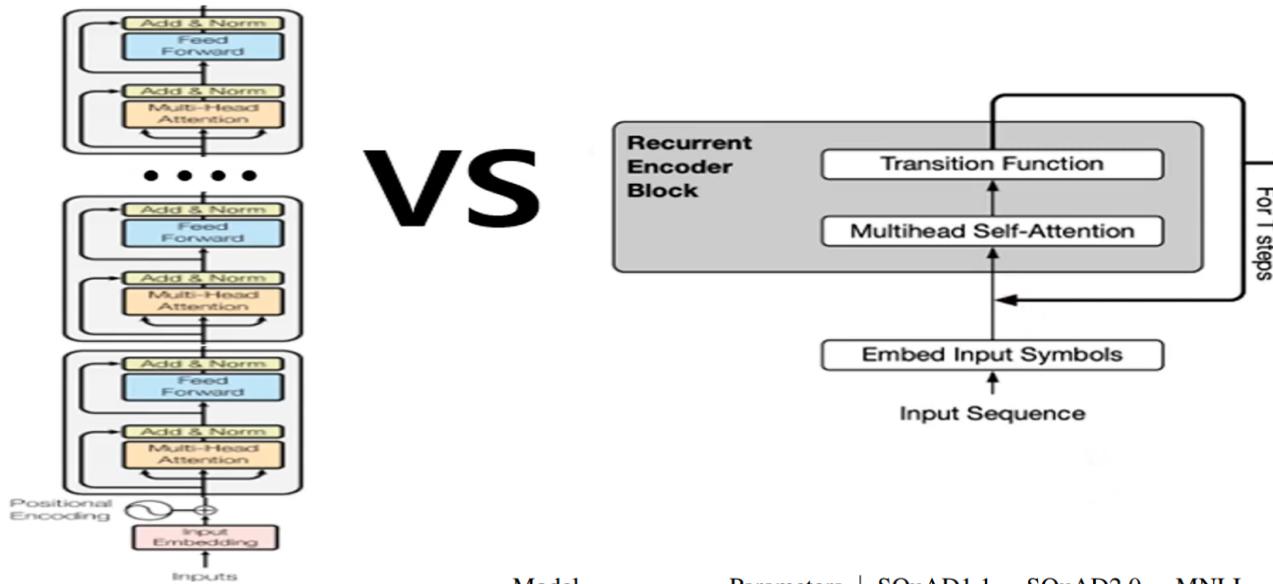
- ◆ ALBERT (A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS)
  - Factorized Embedding Parameterization ( $E < H$ ,  $V^*H$  vs.  $V^*E + E^*H$ )  $|V| = 30000$ ,  $|E| = 256$ ,  $|H| = 768 \rightarrow$  Result: Table 3
  - Cross-Layer Parameter Sharing
  - Sentence-Order Prediction (SOP)

Model	$E$	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

# Other Following Models

- ◆ ALBERT (A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS)
  - Cross-Layer Parameter Sharing



	Model	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base $E=768$	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base $E=128$	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

Table 4: The effect of cross-layer parameter-sharing strategies, ALBERT-base configuration.

# Other Following Models

- ◆ ALBERT (A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS)
  - Sentence-Order Prediction (SOP)

SP tasks	Intrinsic Tasks			Downstream Tasks					Avg
	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	<b>91.1</b>	62.3	79.2
SOP	54.0	78.9	86.5	<b>89.3/82.3</b>	<b>80.0/77.1</b>	<b>82.0</b>	90.3	<b>64.0</b>	<b>80.1</b>

Table 5: The effect of sentence-prediction loss, NSP vs. SOP, on intrinsic and downstream tasks.

# Other Following Models

- ◆ ALBERT (A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS)

	Model	Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

	Model	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	<b>94.1/88.3</b>	<b>88.1/85.1</b>	<b>88.0</b>	<b>95.2</b>	<b>82.3</b>	<b>88.7</b>	0.3x

Table 2: Dev set results for models pretrained over BOOKCORPUS and Wikipedia for 125k steps.

# Other Following Models

- ◆ BioBERT (a pre-trained biomedical language representation model for biomedical text mining)

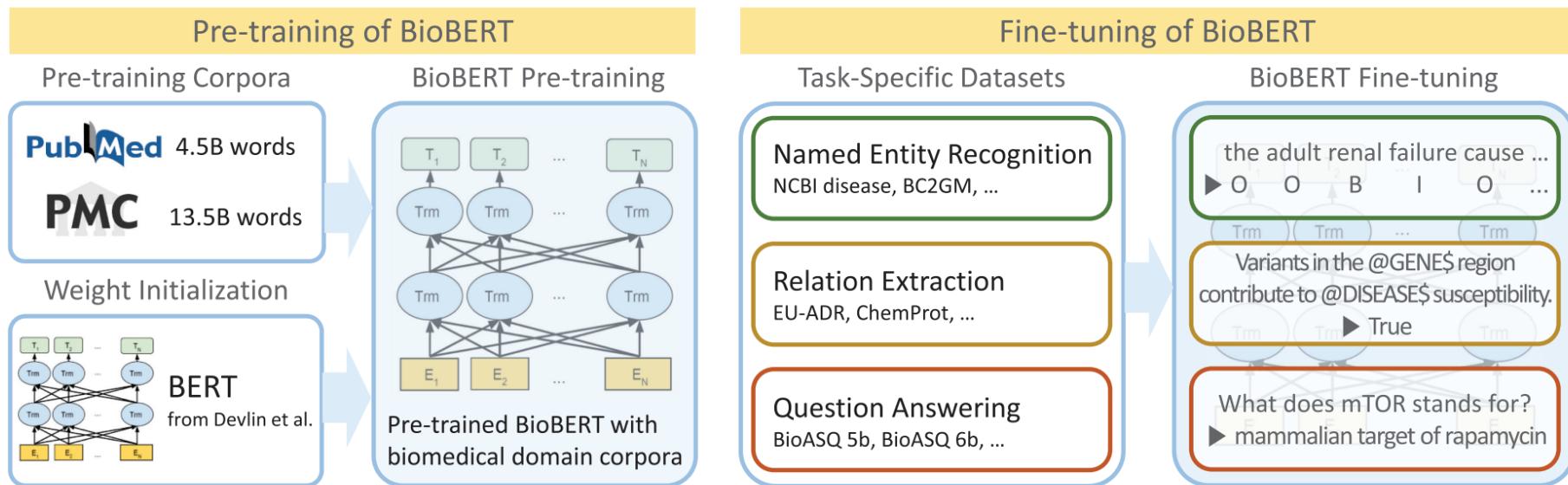


Fig. 1. Overview of the pre-training and fine-tuning of BioBERT

# Other Following Models

- ◆ BioBERT (a pre-trained biomedical language representation model for biomedical text mining)

**Table 1.** List of text corpora used for BioBERT

Corpus	Number of words	Domain
English Wikipedia	2.5B	General
BooksCorpus	0.8B	General
PubMed Abstracts	4.5B	Biomedical
PMC Full-text articles	13.5B	Biomedical

**Table 2.** Pre-training BioBERT on different combinations of the following text corpora: English Wikipedia (Wiki), BooksCorpus (Books), PubMed abstracts (PubMed) and PMC full-text articles (PMC)

Model	Corpus combination
BERT ( <a href="#">Devlin et al., 2019</a> )	Wiki + Books
BioBERT (+PubMed)	Wiki + Books + PubMed
BioBERT (+PMC)	Wiki + Books + PMC
BioBERT (+PubMed + PMC)	Wiki + Books + PubMed + PMC

# Other Following Models

- ◆ BioBERT (a pre-trained biomedical language representation model for biomedical text mining)

**Table 6.** Test results in biomedical named entity recognition

Type	Datasets	Metrics	SOTA	BERT	BioBERT v1.0			BioBERT v1.1
				(Wiki + Books)	(+ PubMed)	(+ PMC)	(+ PubMed + PMC)	(+ PubMed)
Disease	NCBI disease	P	<u>88.30</u>	84.12	86.76	86.16	<u>89.04</u>	88.22
		R	89.00	87.19	88.02	<u>89.48</u>	<u>89.69</u>	<u>91.25</u>
		F	88.60	85.63	87.38	87.79	<u>89.36</u>	<u>89.71</u>
	2010 i2b2/VA	P	<u>87.44</u>	84.04	85.37	85.55	<u>87.50</u>	86.93
		R	<u>86.25</u>	84.08	85.64	85.72	85.44	<u>86.53</u>
		F	<u>86.84</u>	84.06	85.51	85.64	86.46	<u>86.73</u>
	BC5CDR	P	<u>89.61</u>	81.97	85.80	84.67	85.86	<u>86.47</u>
		R	83.09	82.48	86.60	85.87	<u>87.27</u>	<u>87.84</u>
		F	<u>86.23</u>	82.41	86.20	85.27	86.56	<u>87.15</u>
Drug/chem.	BC5CDR	P	<u>94.26</u>	90.94	92.52	92.46	93.27	<u>93.68</u>
		R	92.38	91.38	92.76	92.63	<u>93.61</u>	<u>93.26</u>
		F	93.31	91.16	92.64	92.54	<u>93.44</u>	<u>93.47</u>
	BC4CHEMD	P	<u>92.29</u>	91.19	91.77	91.65	92.23	<u>92.80</u>
		R	90.01	88.92	<u>90.77</u>	90.30	90.61	<u>91.92</u>
		F	91.14	90.04	91.26	90.97	<u>91.41</u>	<u>92.36</u>
Gene/protein	BC2GM	P	81.81	81.17	81.72	82.86	<u>85.16</u>	<u>84.32</u>
		R	81.57	82.42	83.38	<u>84.21</u>	83.65	<u>85.12</u>
		F	81.69	81.79	82.54	83.53	<u>84.40</u>	<u>84.72</u>
	JNLPBA	P	<u>74.43</u>	69.57	71.11	71.17	<u>72.68</u>	72.24
		R	<u>83.22</u>	81.20	83.11	82.76	83.21	<u>83.56</u>
		F	<u>78.58</u>	74.94	76.65	76.53	<u>77.59</u>	77.49
Species	LINNAEUS	P	<u>92.80</u>	91.17	91.83	91.62	<u>93.84</u>	90.77
		R	<u>94.29</u>	84.30	84.72	85.48	<u>86.11</u>	85.83
		F	<u>93.54</u>	87.60	88.13	88.45	<u>89.81</u>	88.24
	Species-800	P	<u>74.34</u>	69.35	70.60	71.54	<u>72.84</u>	72.80
		R	<u>75.96</u>	74.05	75.75	74.71	<u>77.97</u>	75.36
		F	<u>74.98</u>	71.63	73.08	73.09	<u>75.31</u>	74.06

# Reference

- ◆ Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, ETC, Language Models are Few-Shot Learners, OpenAI
- ◆ K. Cho, B. Merrieboer, C. Gulcehrre, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation
- ◆ Nikita Kitaev, Łukasz Kaiser, Anselm Levskaya, Reformer: The Efficient Transformer, <https://arxiv.org/abs/2001.04451>
- ◆ Zhilin Yang, Zihang Dai, Yiming Yang, etc, XLNet: Generalized Autoregressive Pretraining for Language Understanding, <https://github.com/zihangdai/xlnet>
- ◆ Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, Jaewoo Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining, <https://arxiv.org/abs/1901.08746>
- ◆ Ranto Sawai, Incheon Paik, and Ayato Kuwana, Sentence Augmentation for Language Translation Using GPT-2, MDPI Journal
- ◆ Sergey Edunov, Myle Ott, Michael Auli, David Grangier, Understanding Back-Translation at Scale, <https://arxiv.org/abs/1808.09381v2>
- ◆ Incheon Paik and Jun-Wei Wang, Improving Text-to-Code Generation with Features of Code Graph on GPT-2



# Thank you!

# Questions?