# BERT and Applications in NLP

April 23, 2025

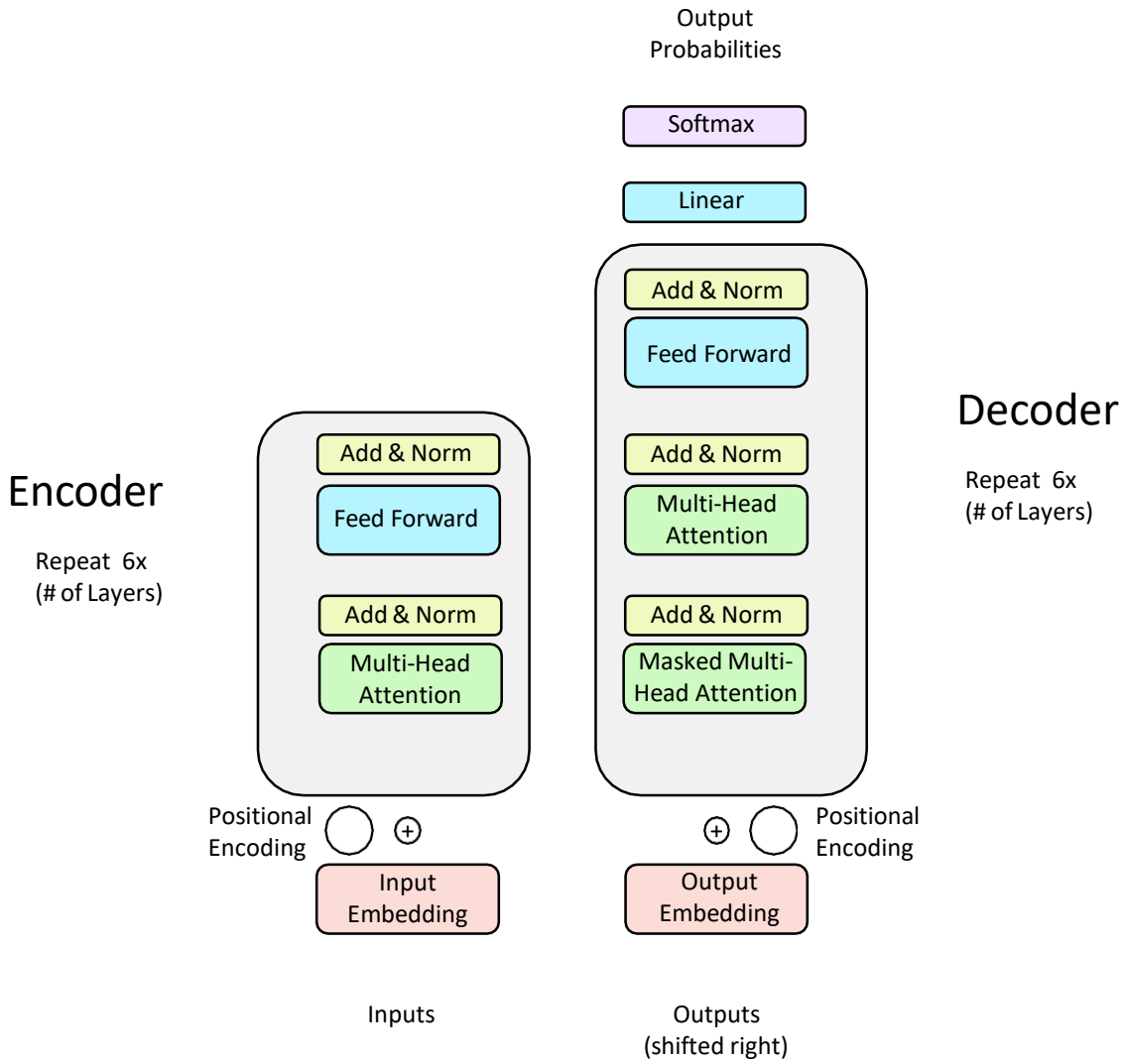
# Acknowledgments

*Slides adapted from*

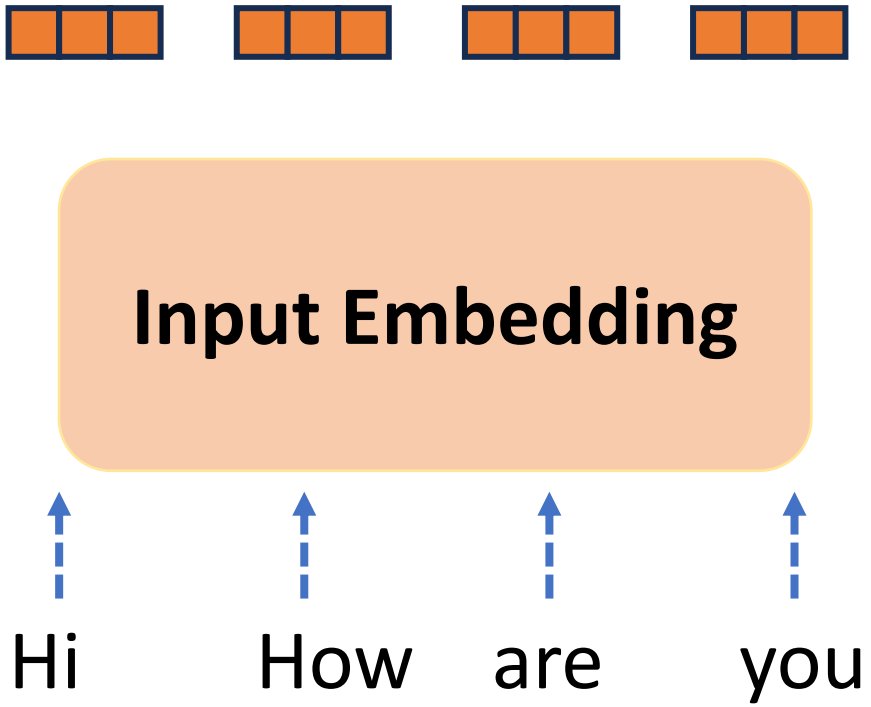
*Jacob Devlin, Anna Goldie, Shafiq Joty, Chris Manning*

# Transformers

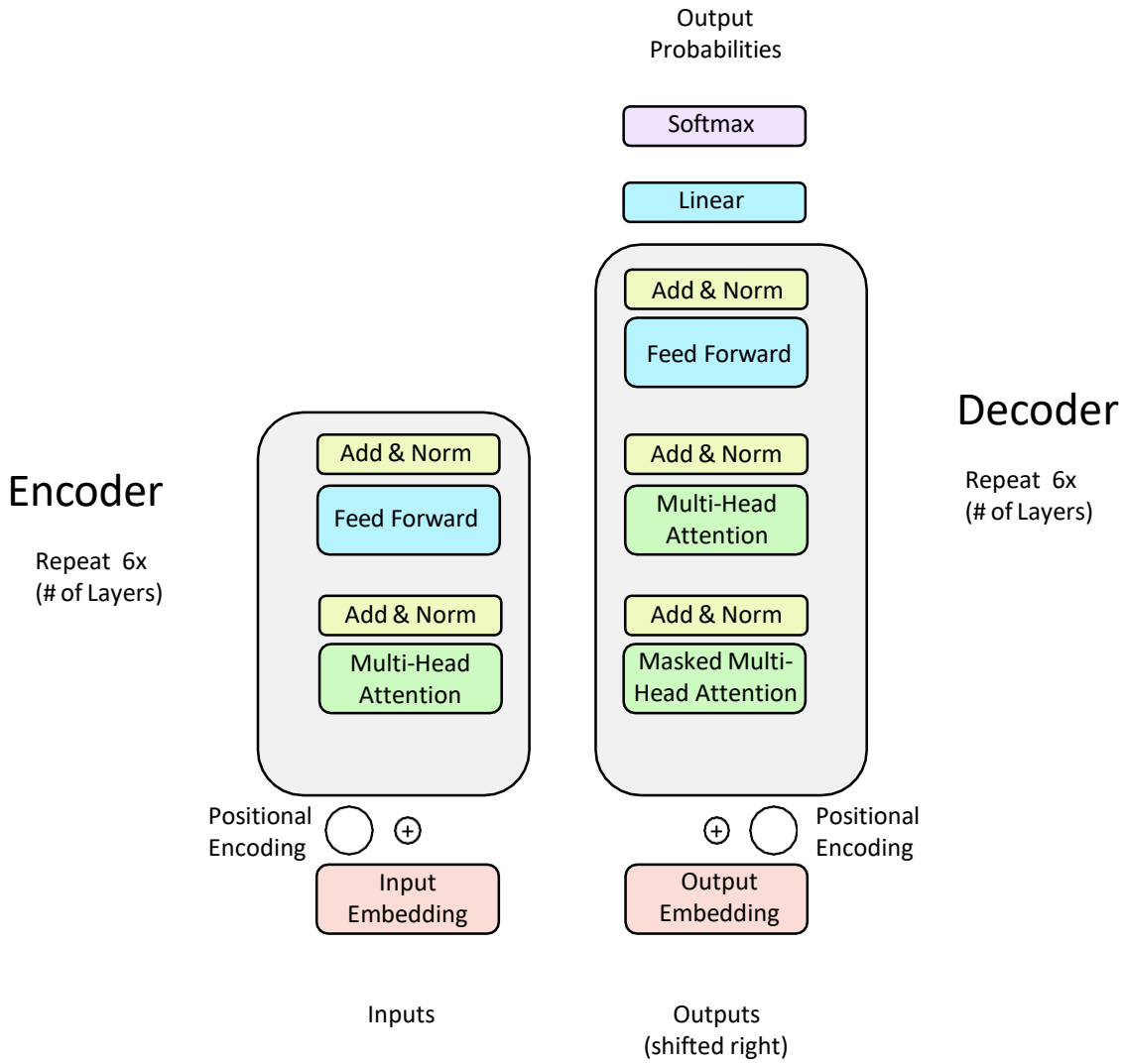
# Recap of the Transformer Architecture



# Transformer – Input Embedding



# Recap of the Transformer Architecture



# Transformer – Positional Embedding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

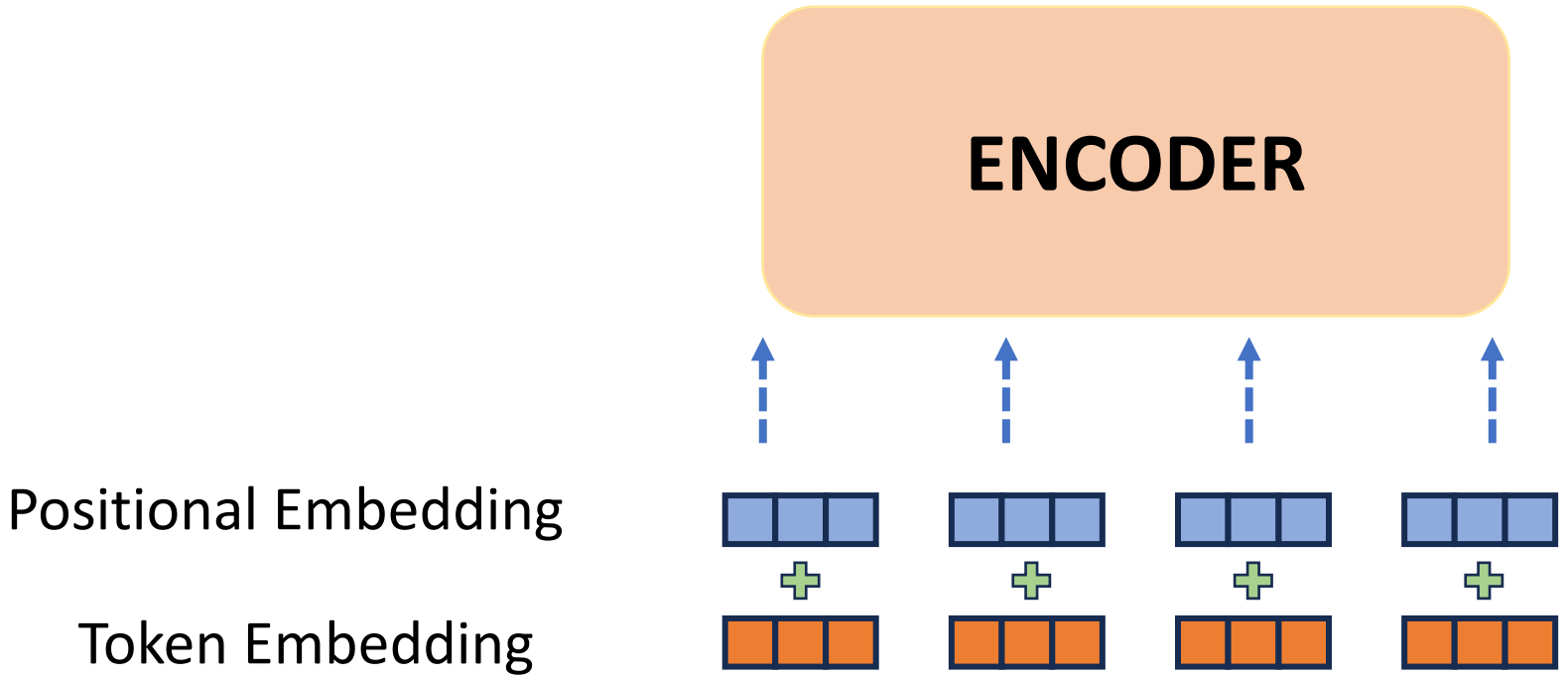
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

- For **odd** index, create a vector using the **cos** function.
- For **even** index, create a vector using the **sin** function.

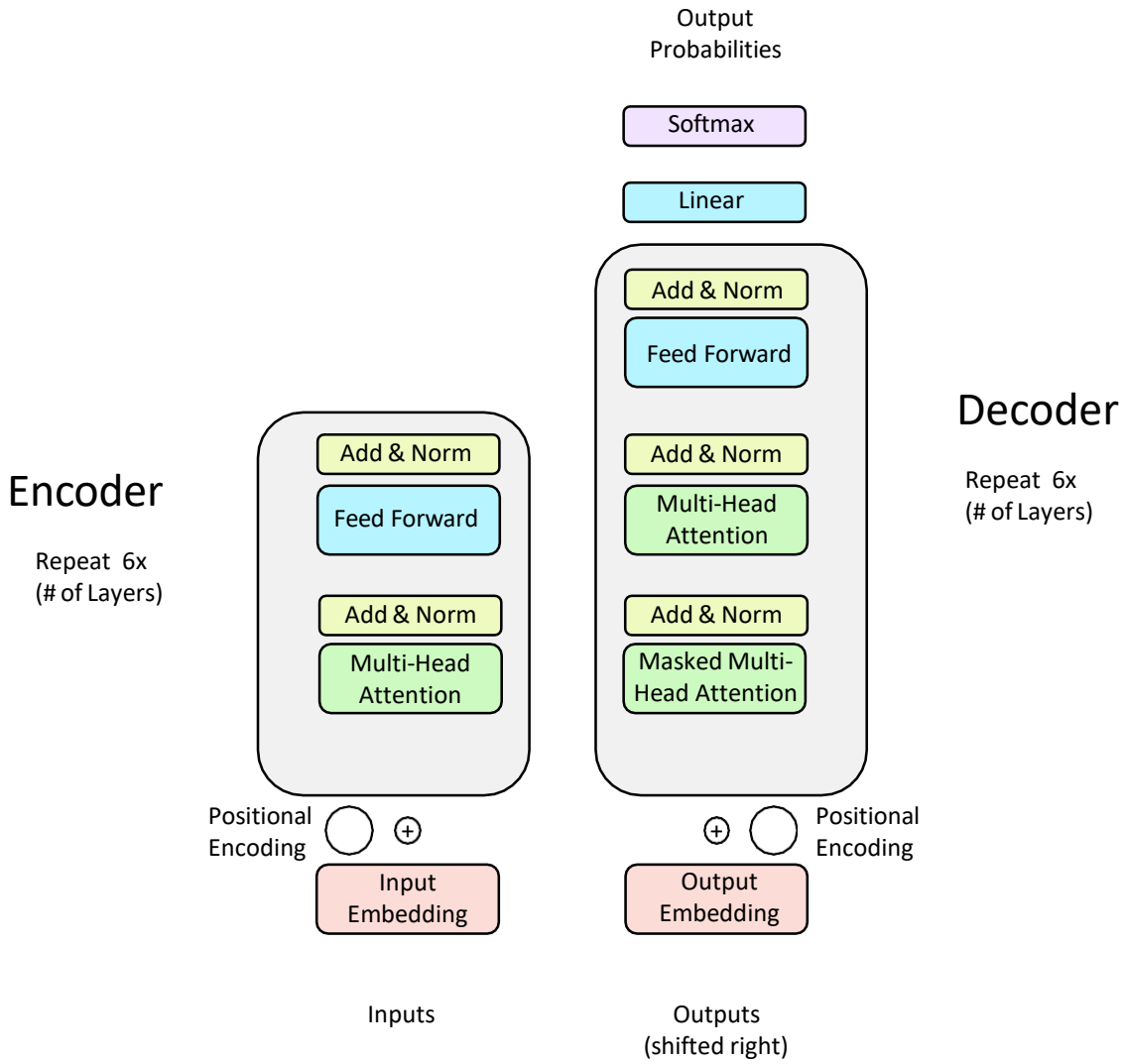
Add those vectors to the corresponding input embedding, providing **relative or absolute position** information of each token **within the sequence.**

Image extracted from the blog.

# Transformer – Encoder Input



# Recap of the Transformer Architecture



# Transformer – Encoder Architecture

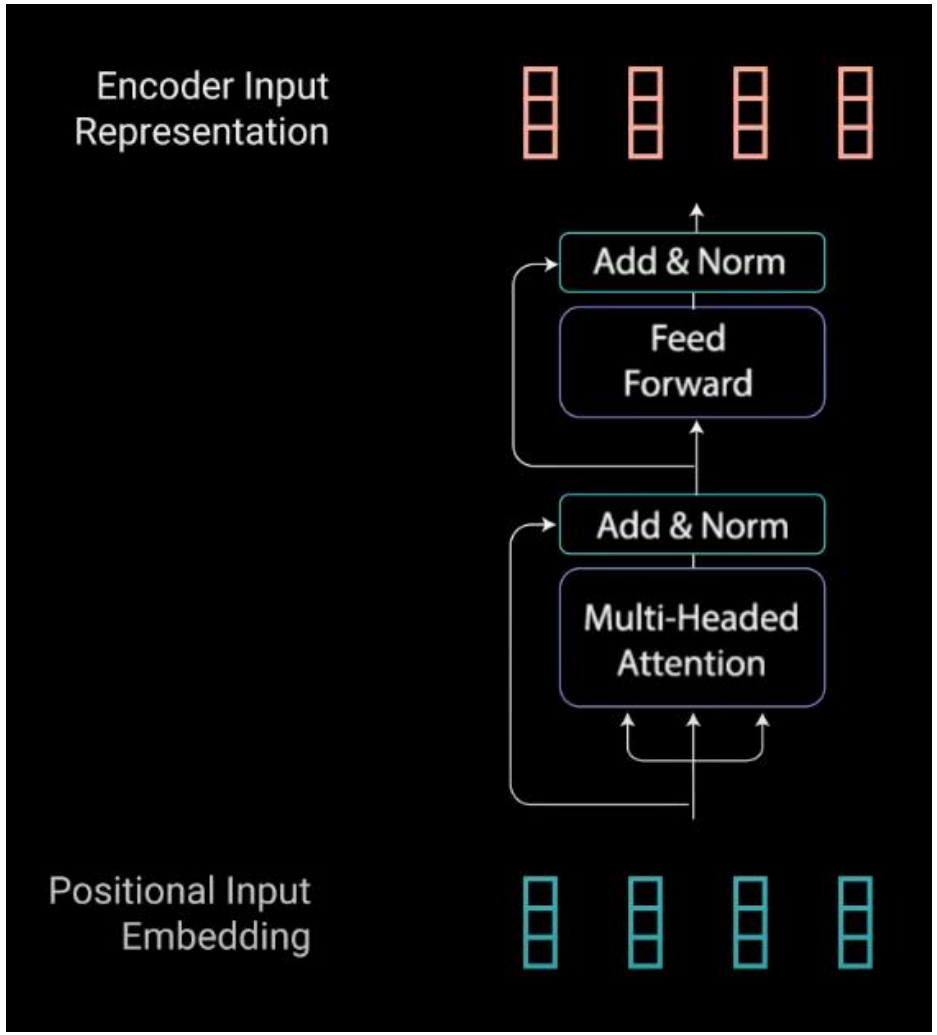



Image extracted from the blog.

# Transformer – Multi-head Attention

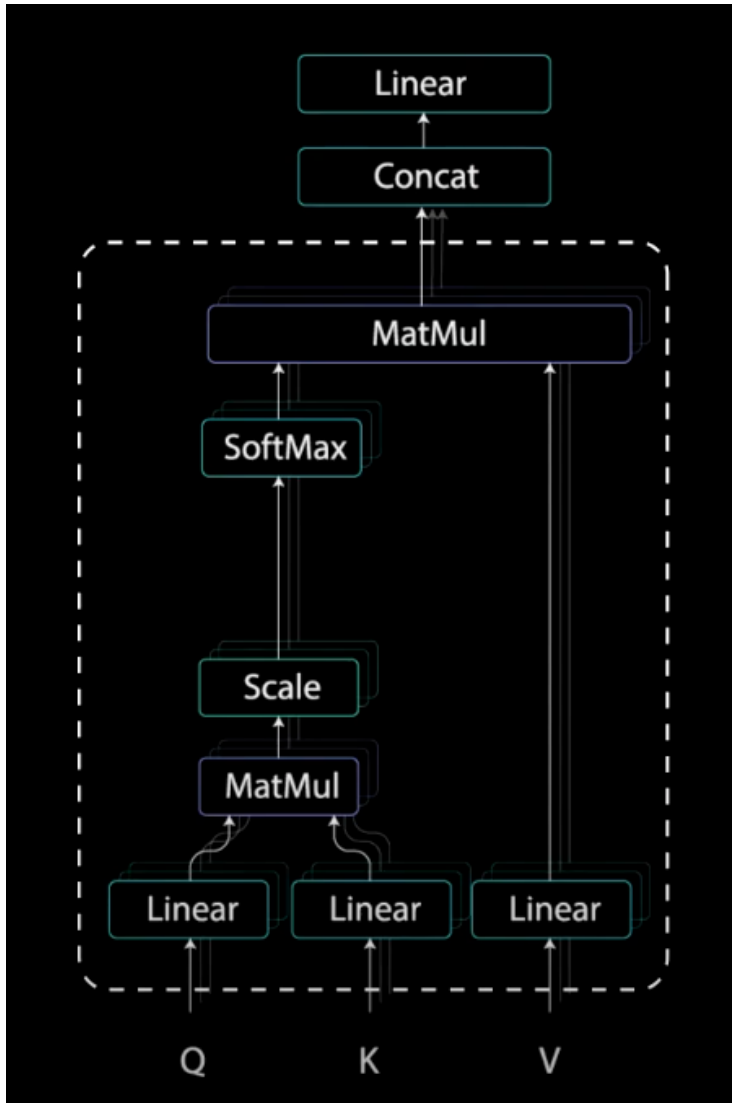
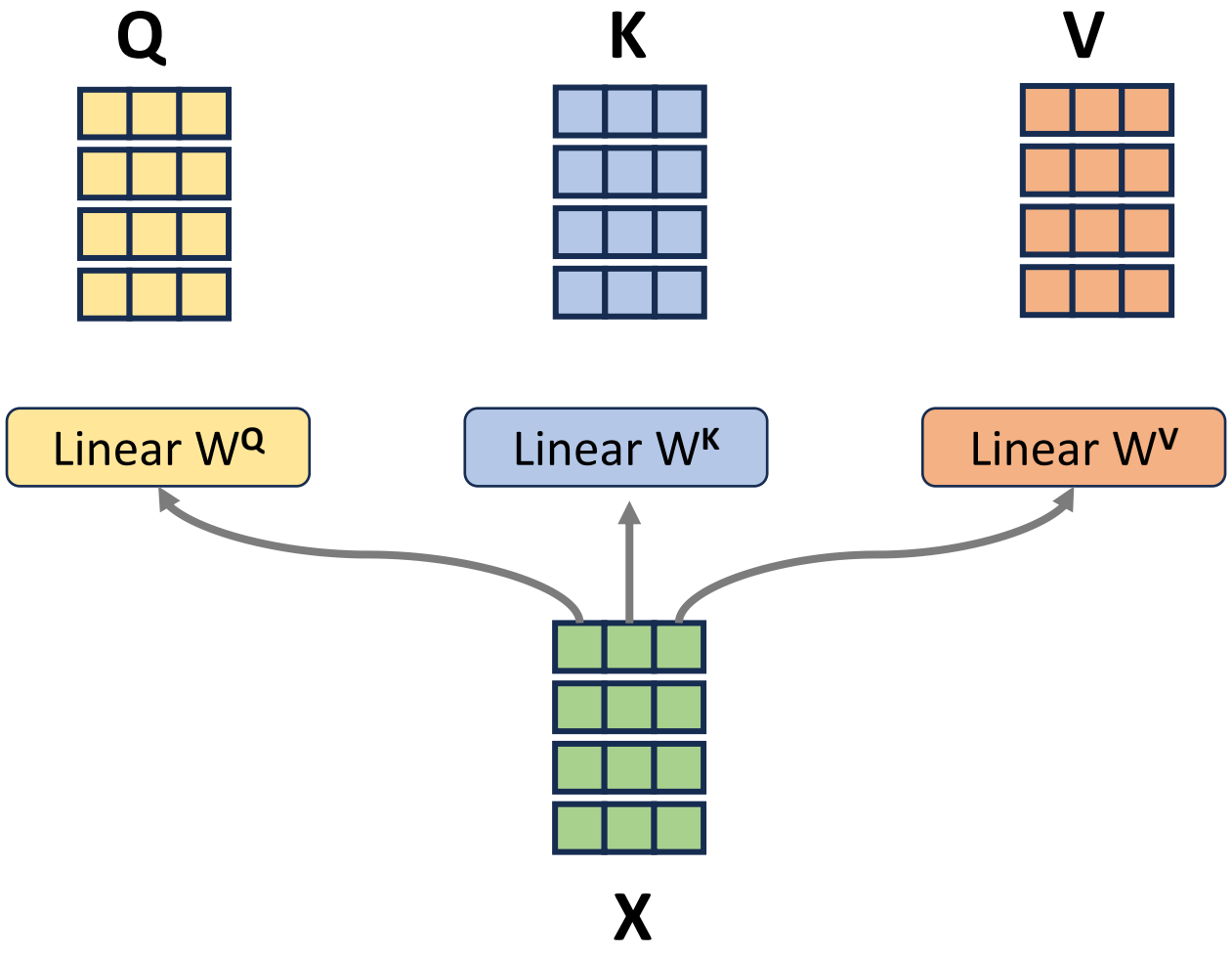


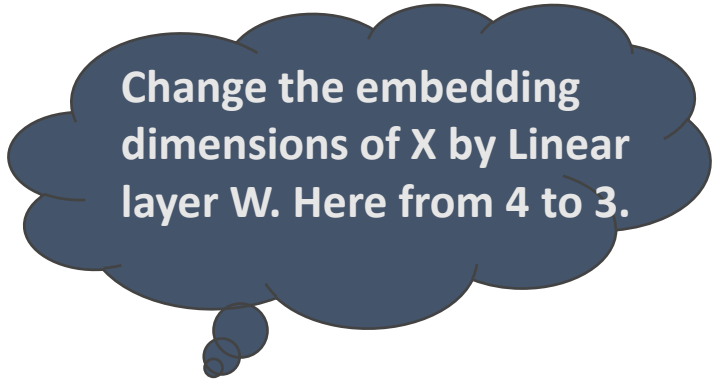

Image extracted from the blog.

# Transformer – Multi-head Attention - KQV



# Transformer – Multi-head Attention - KQV



# Transformer – Multi-head Attention - $QK^T$

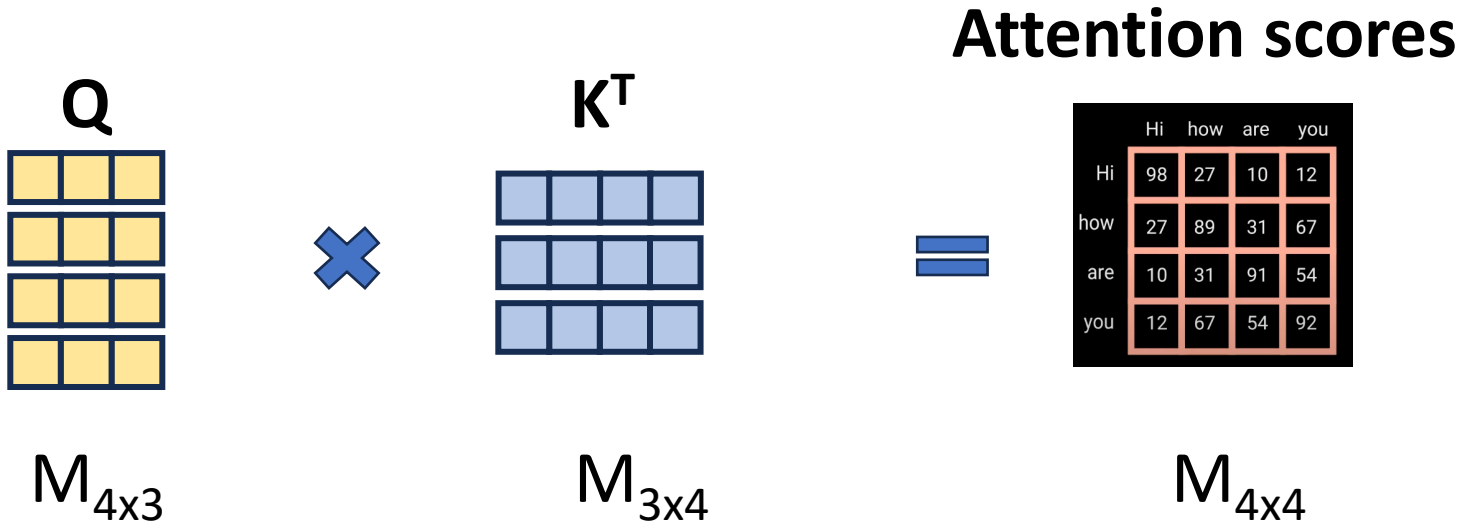
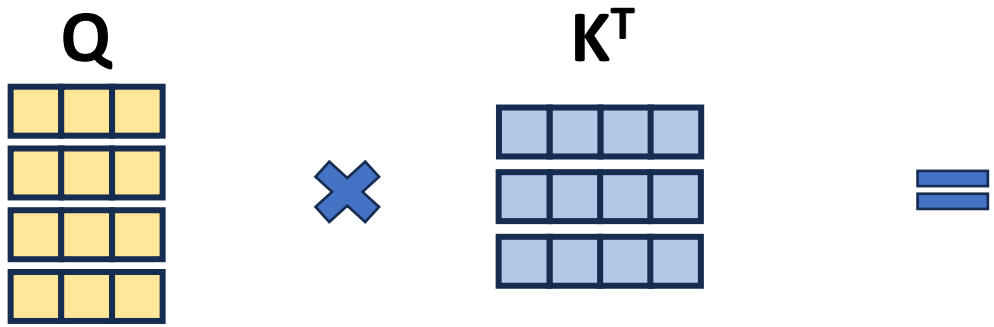



Image extracted from the blog.

# Transformer – Multi-head Attention - $QK^T$




**Attention scores**

| | Hi | how | are | you |
|---|---|---|---|---|
| Hi | 98 | 27 | 10 | 12 |
| how | 27 | 89 | 31 | 67 |
| are | 10 | 31 | 91 | 54 |
| you | 12 | 67 | 54 | 92 |

Image extracted from the blog.

# Transformer – Multi-head Attention – Scaling $QK^T$

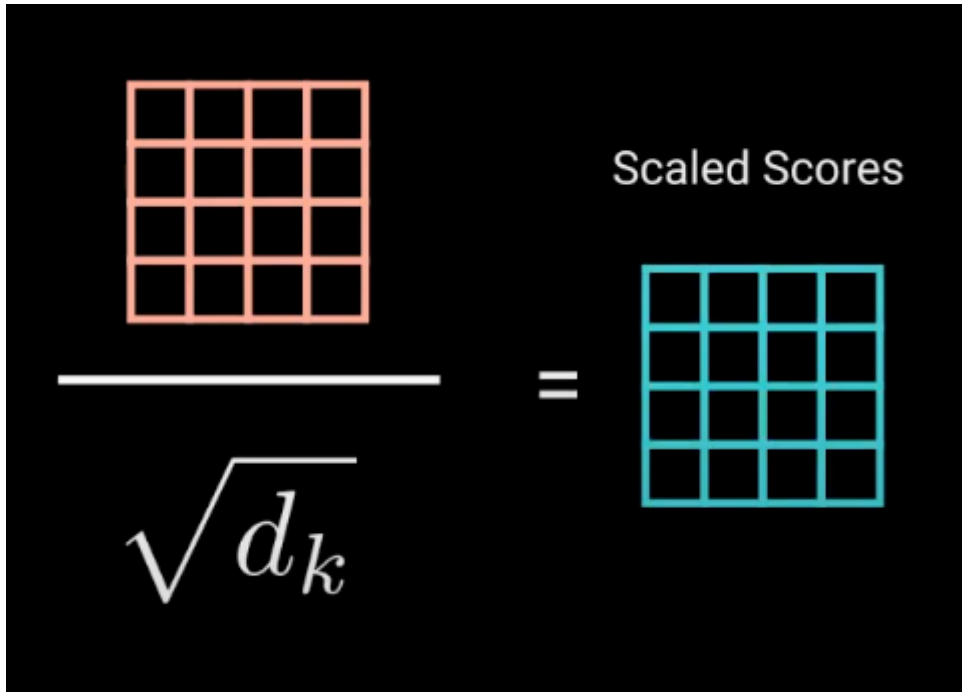



Image extracted from the blog.

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?

In statistics, if $X$ and $Y$ are independent and randomly distributed variables:

$$\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$
$$Var(X+Y) = Var(X) + Var(Y)$$
$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$$
$$Var(XY) = (Var(X) + \mathbb{E}[X]^2)(Var(Y) + \mathbb{E}[Y]^2) - \mathbb{E}[X]^2\mathbb{E}[Y]^2$$

Image extracted https://ai.stackexchange.com/questions/21237/why-does-this-multiplication-of-q-and-k-have-a-variance-of-d-k-in-scaled

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?

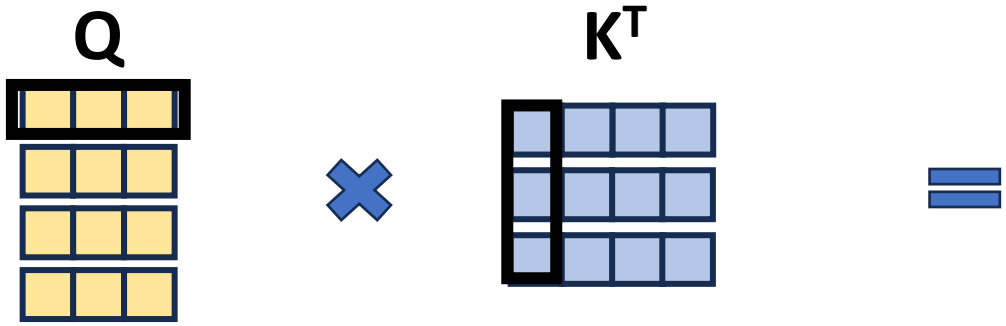
Let Q and K be matrices with the shape of $\mathbf{n \times d_k}$
n is the number of tokens of input.

Each entry $Q_{i,j}$ or $K_{i,j}$ is a random variable following $\boldsymbol{N(0,1)}$, $\boldsymbol{\mu = 0}$ **and** $\boldsymbol{\sigma = 1}$.
Every entry is independent from each other.

Since each entry of Q and K have identical distribution, we can focus only on the top-left-most element of $QK^T$.
The same applies to every other element.

Image extracted https://ai.stackexchange.com/questions/21237/why-does-this-multiplication-of-q-and-k-have-a-variance-of-d-k-in-scaled

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?




**Attention scores**

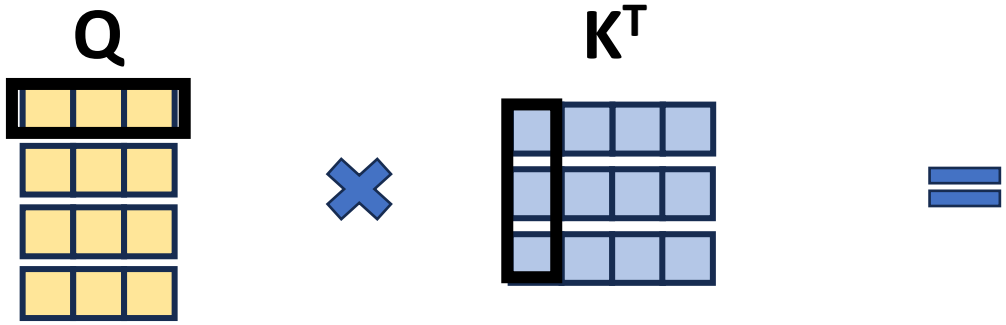
| | Hi | how | are | you |
|---|---|---|---|---|
| Hi | z | 27 | 10 | 12 |
| how | 27 | 89 | 31 | 67 |
| are | 10 | 31 | 91 | 54 |
| you | 12 | 67 | 54 | 92 |

**E(z)** = $E(q_{11} * k_{11} + q_{12} * k_{21} + q_{13} * k_{31})$
= $E(q_{11} * k_{11}) + E(+ q_{12} * k_{21}) + E(q_{13} * k_{31})$
= $E(q_{11}) * E(k_{11}) +$
$E(q_{12}) * E(k_{21}) +$
$E(q_{13}) * E(k_{31})$
= $0 * 0 + 0 * 0 + 0 * 0$
= **0**

Similarly, let's see Var(z):

Image extracted from the blog.

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?




**Attention scores**

| | Hi | how | are | you |
|---|---|---|---|---|
| Hi | z | 27 | 10 | 12 |
| how | 27 | 89 | 31 | 67 |
| are | 10 | 31 | 91 | 54 |
| you | 12 | 67 | 54 | 92 |

$$\text{Var}(z) = \text{Var}(q_{11} * k_{11} + q_{12} * k_{21} + q_{13} * k_{31})$$
$$= \text{Var}(q_{11} * k_{11}) + \text{Var}(q_{12} * k_{21}) + \text{Var}(q_{13} * k_{31})$$

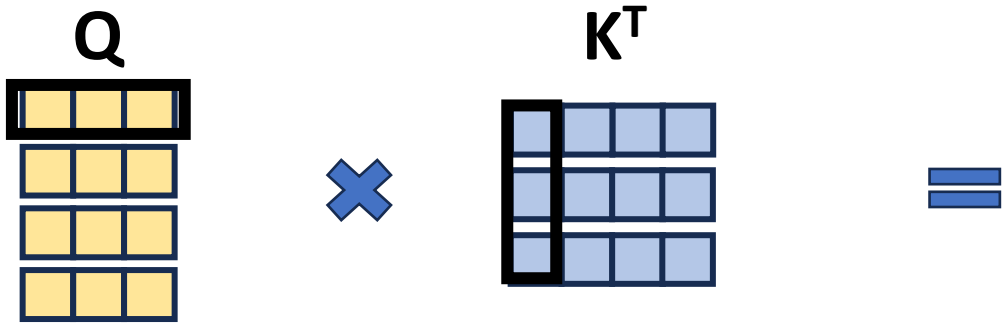
Calculate $\text{Var}(q_{11} * k_{11})$, others are the same:

$$\text{Var}(q_{11} * k_{11})$$
$$= (\text{Var}(q_{11}) + E(q_{11})^2) * (\text{Var}(k_{11}) + E(k_{11})^2) - E(q_{11})^2 * E(k_{11})^2$$
$$= (1 + 0) * (1 + 0) - 0$$

**Var(z)** $= 1 + 1 + 1 = 3 = \mathbf{d_k}$

Image extracted from the blog.

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?




**Attention scores**

| | Hi | how | are | you |
|---|---|---|---|---|
| Hi | z | z | z | z |
| how | 27 | 89 | 31 | 67 |
| are | 10 | 31 | 91 | 54 |
| you | 12 | 67 | 54 | 92 |

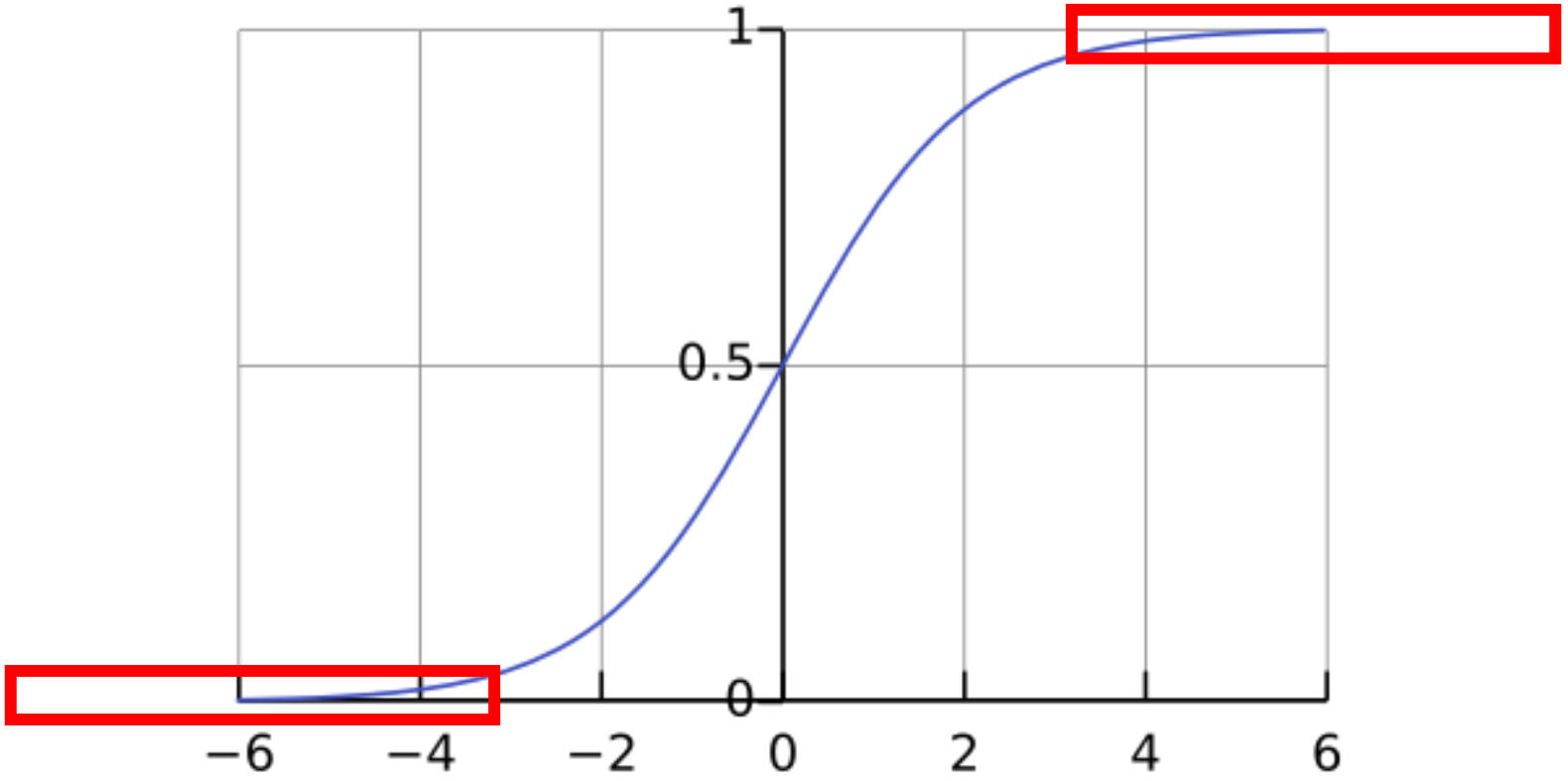
**E(z) = 0**

**Var(z) = $d_k$**

When **$d_k$ is large**
there would be **very large or small z.**

Image extracted from the blog.

# Transformer – Multi-head Attention – Why Scale $QK^T$ ?




**Close to 0 gradient hinders learning.**

Image from Wikipedia

# Transformer – Multi-head Attention – Why Scaling factor is $\sqrt{\mathbf{d_k}}$?

E(z) = 0

Var(z) = $d_k$

-> Normalize by $\sqrt{\mathbf{d_k}}$

**Var($\frac{z}{\sqrt{\mathbf{d_k}}}$)** = $(\frac{1}{\sqrt{d_k}})^2$ * Var(z) = **1**

$$Var(cX) = c^2 Var(X)$$

**E($\frac{z}{\sqrt{\mathbf{d_k}}}$)** = $\frac{1}{\sqrt{d_k}}$ E(z) = **0**

# Transformer – Multi-head Attention – Softmax Scaled $QK^T$

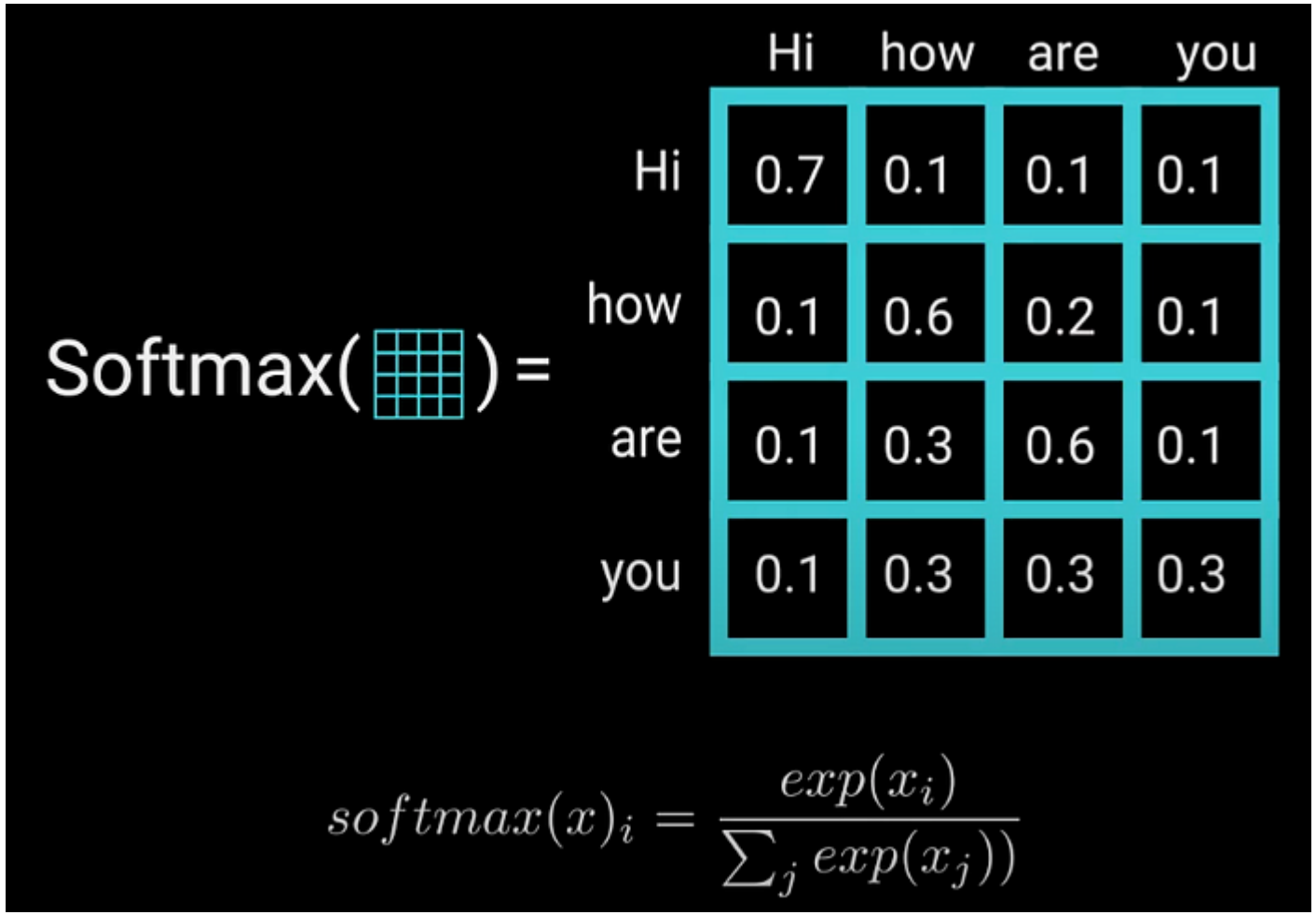
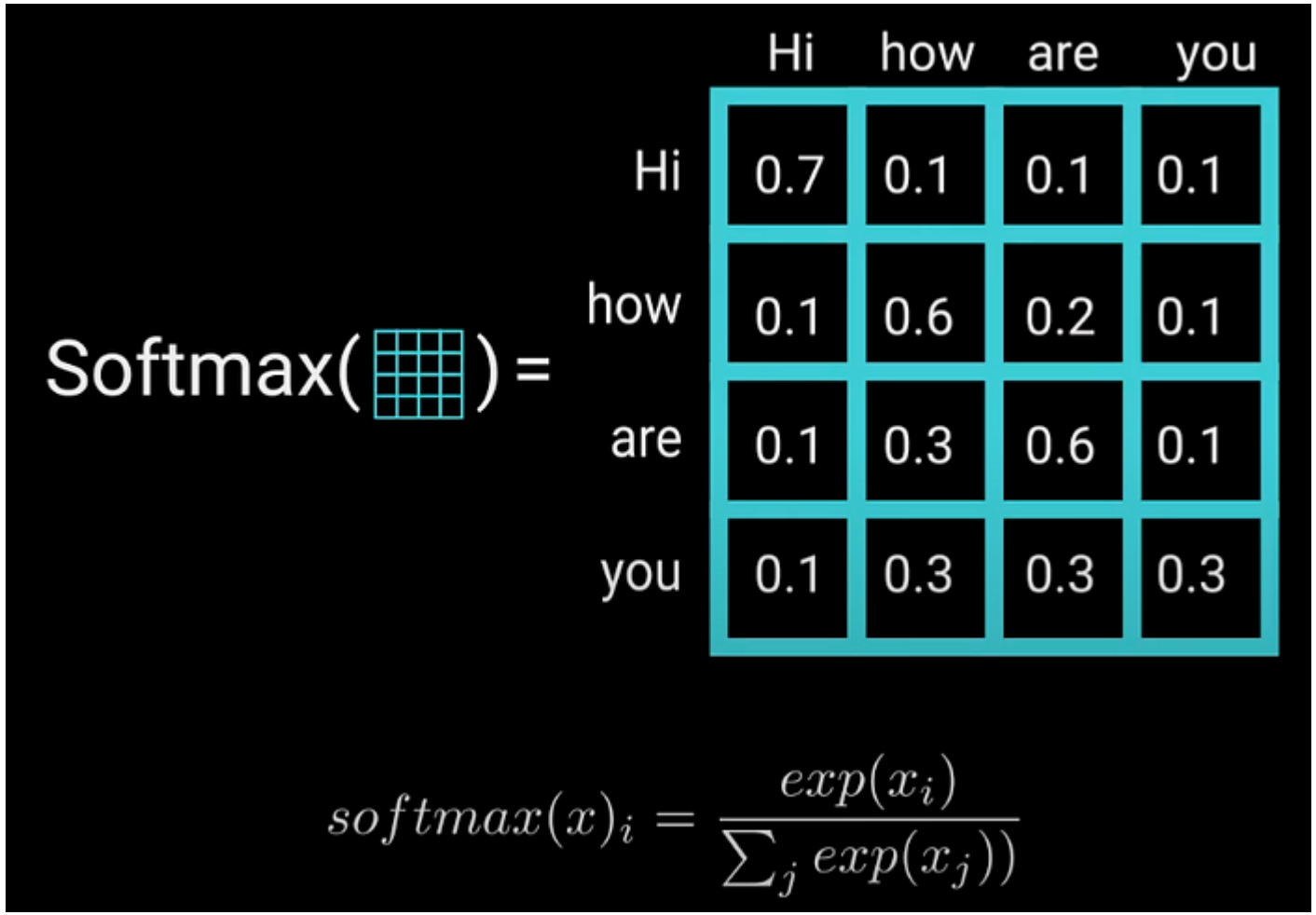



Image extracted from the blog.

# Transformer – Multi-head Attention – Output



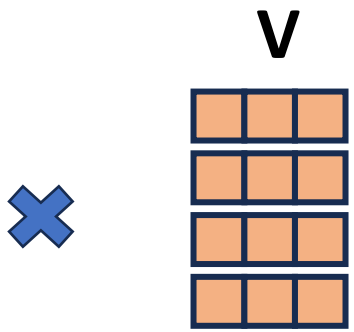
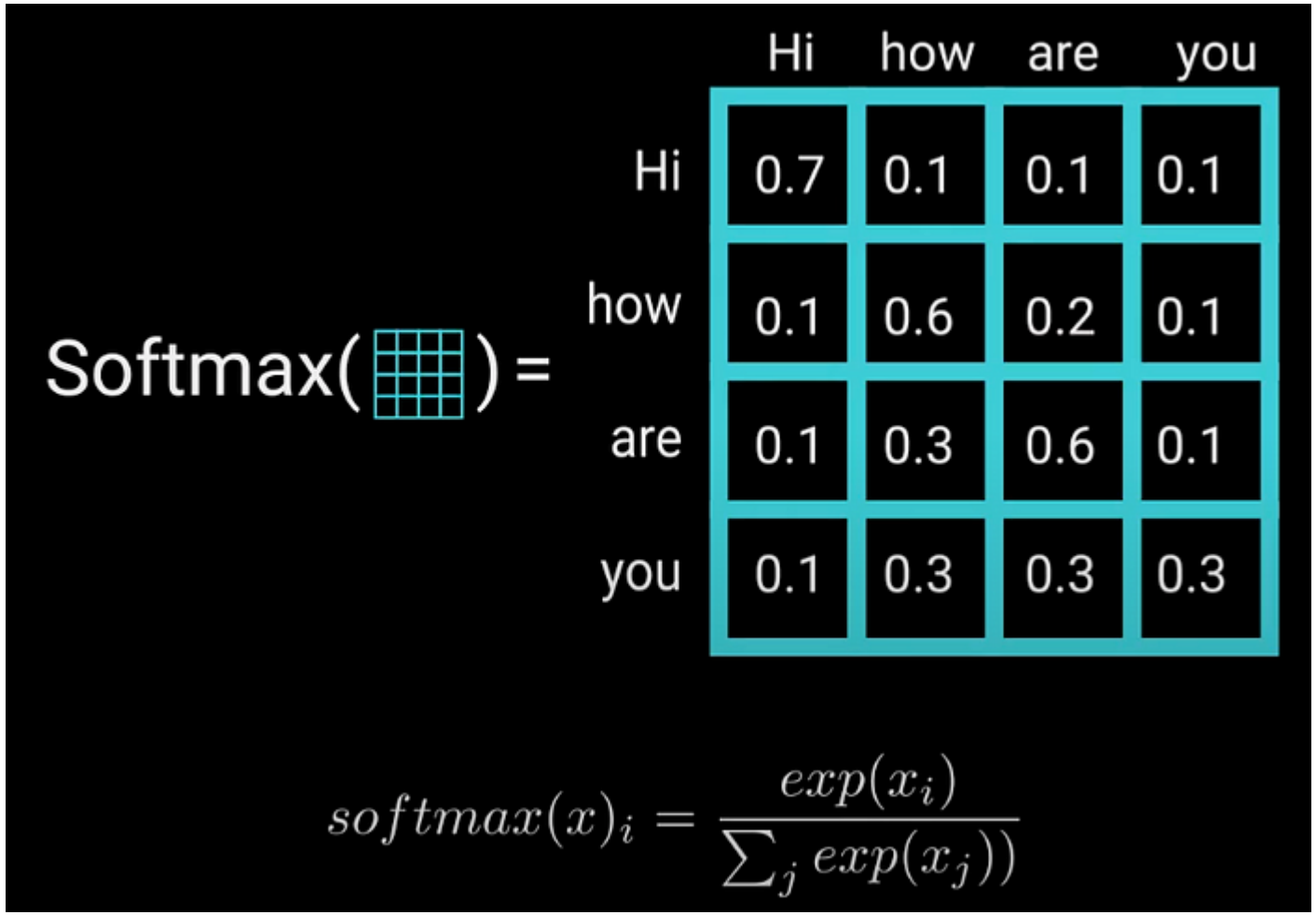




Image extracted from the blog.

# Transformer – Multi-head Attention – Output



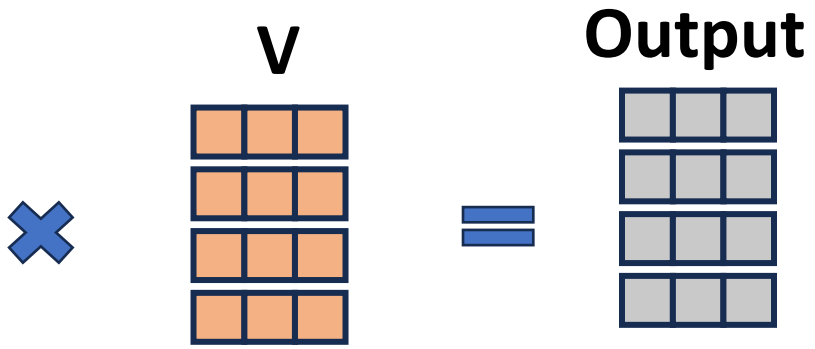




Image extracted from the blog.

# Transformer – Multi-head Attention – One Attention

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

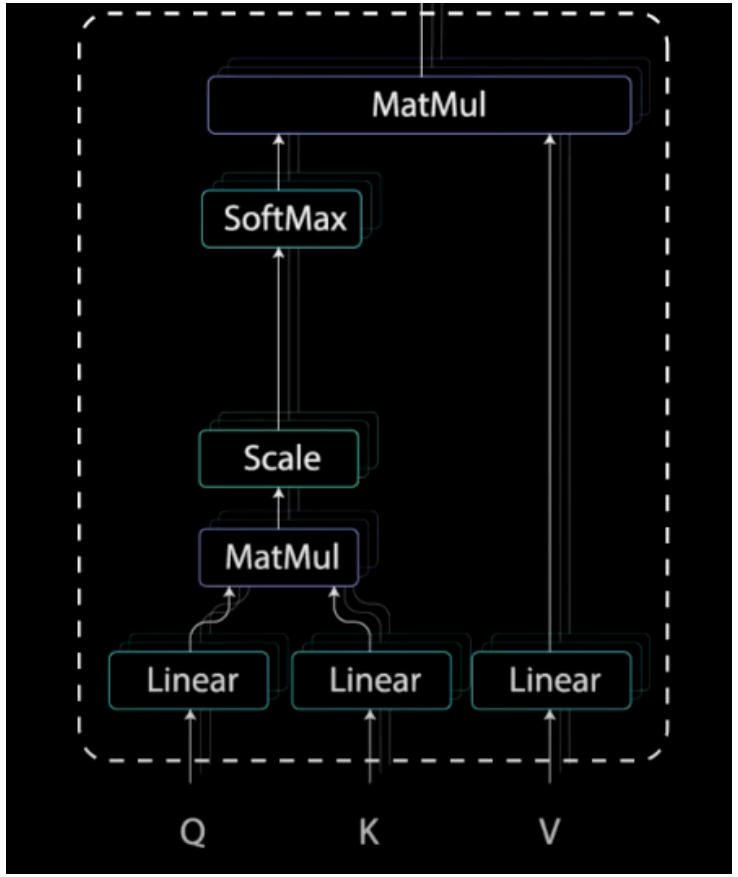



Image extracted from the blog.

# Transformer – Multi-head Attention

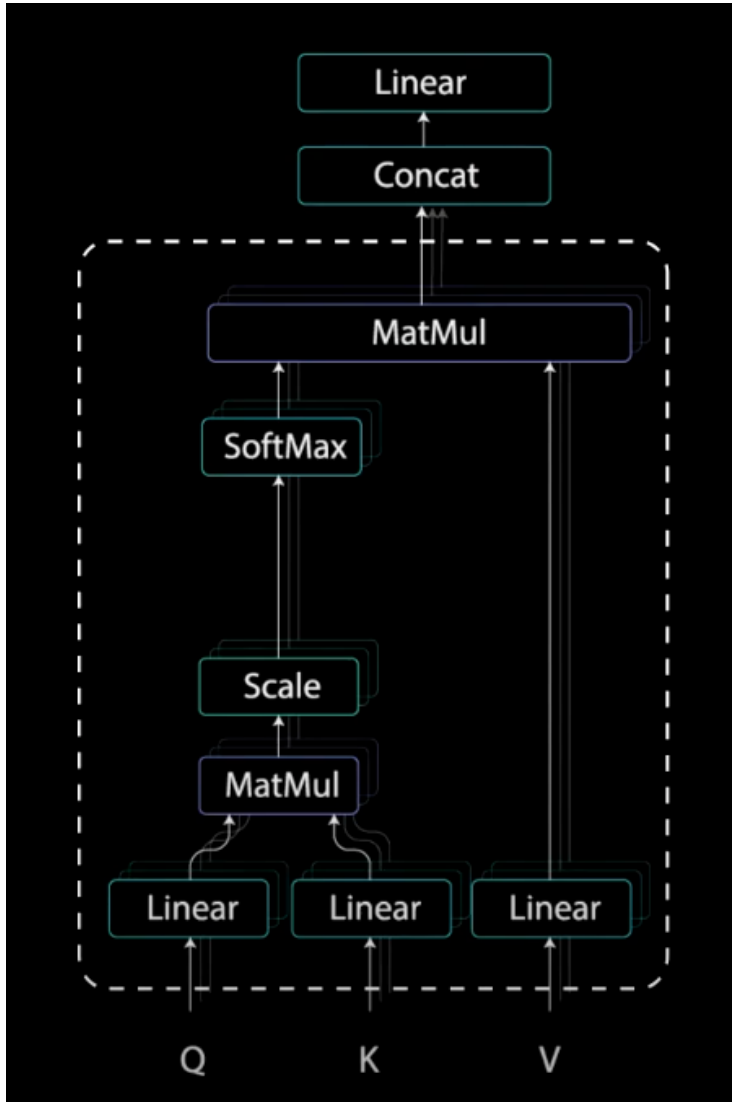
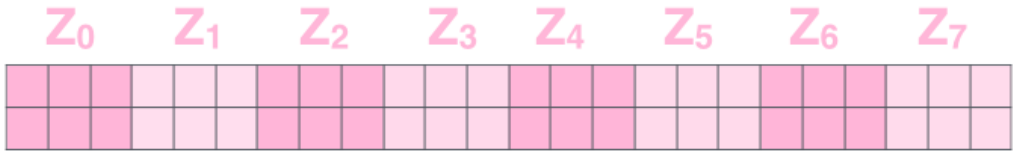



Image extracted from the blog.

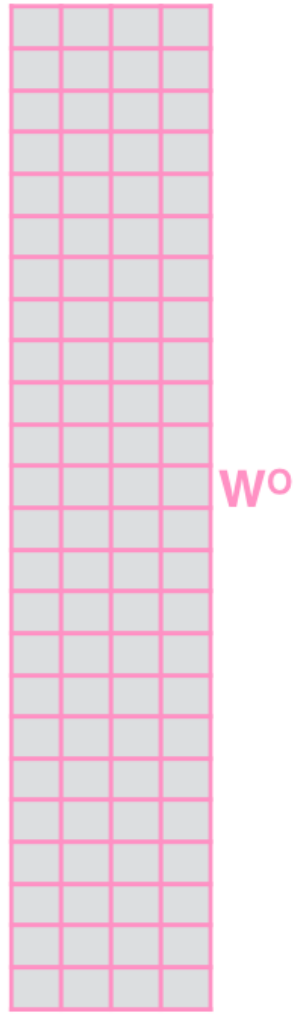
# Transformer – Multi-head Attention – Concat + Linear

1) Concatenate all the attention heads




2) Multiply with a weight matrix $W^O$ that was trained jointly with the model

X




3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

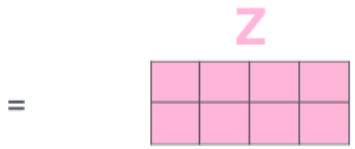
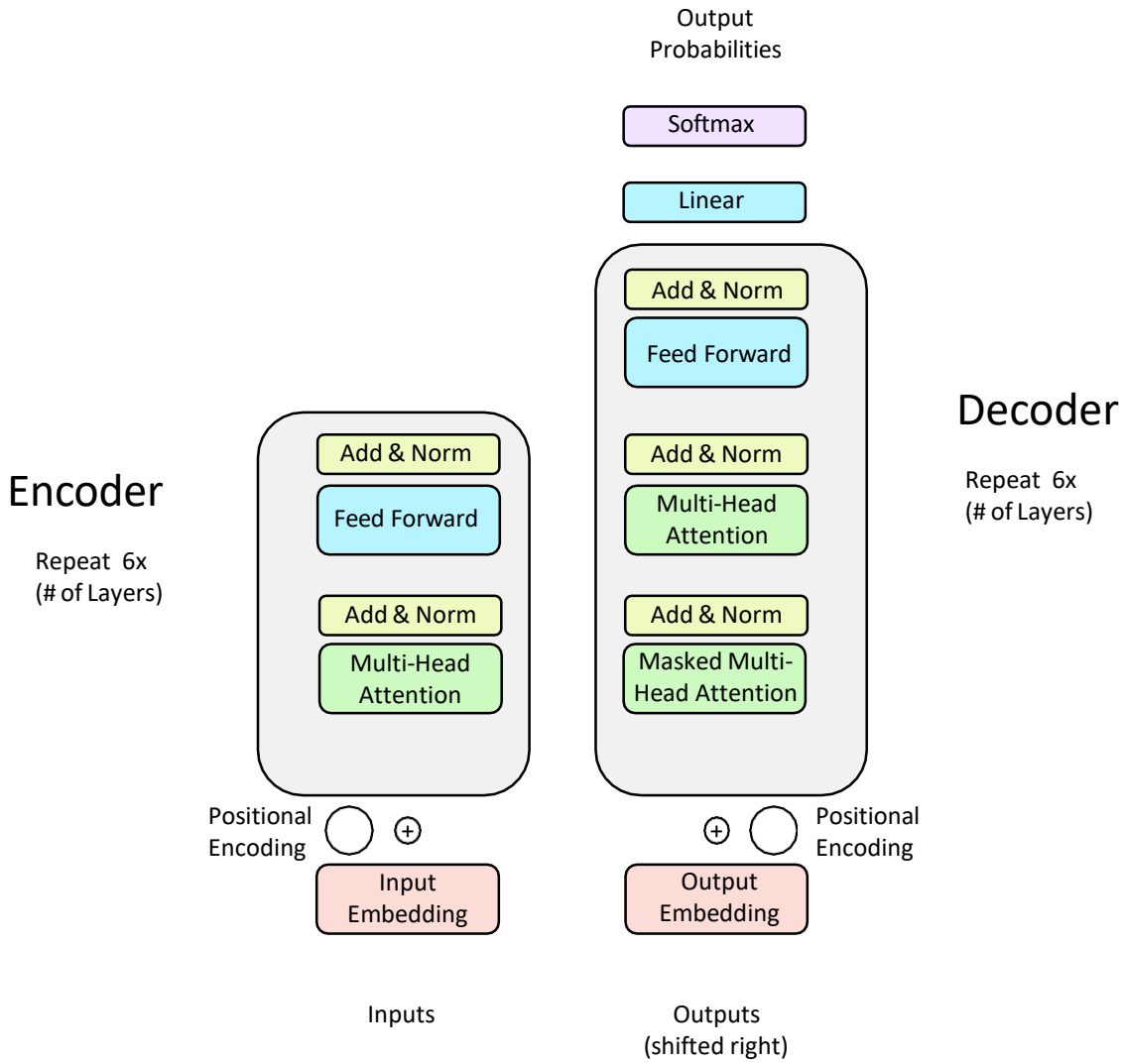



Image extracted from http://jalammar.github.io/illustrated-transformer/

# Recap of the Transformer Architecture



# Transformer – Add & Norm + FFN

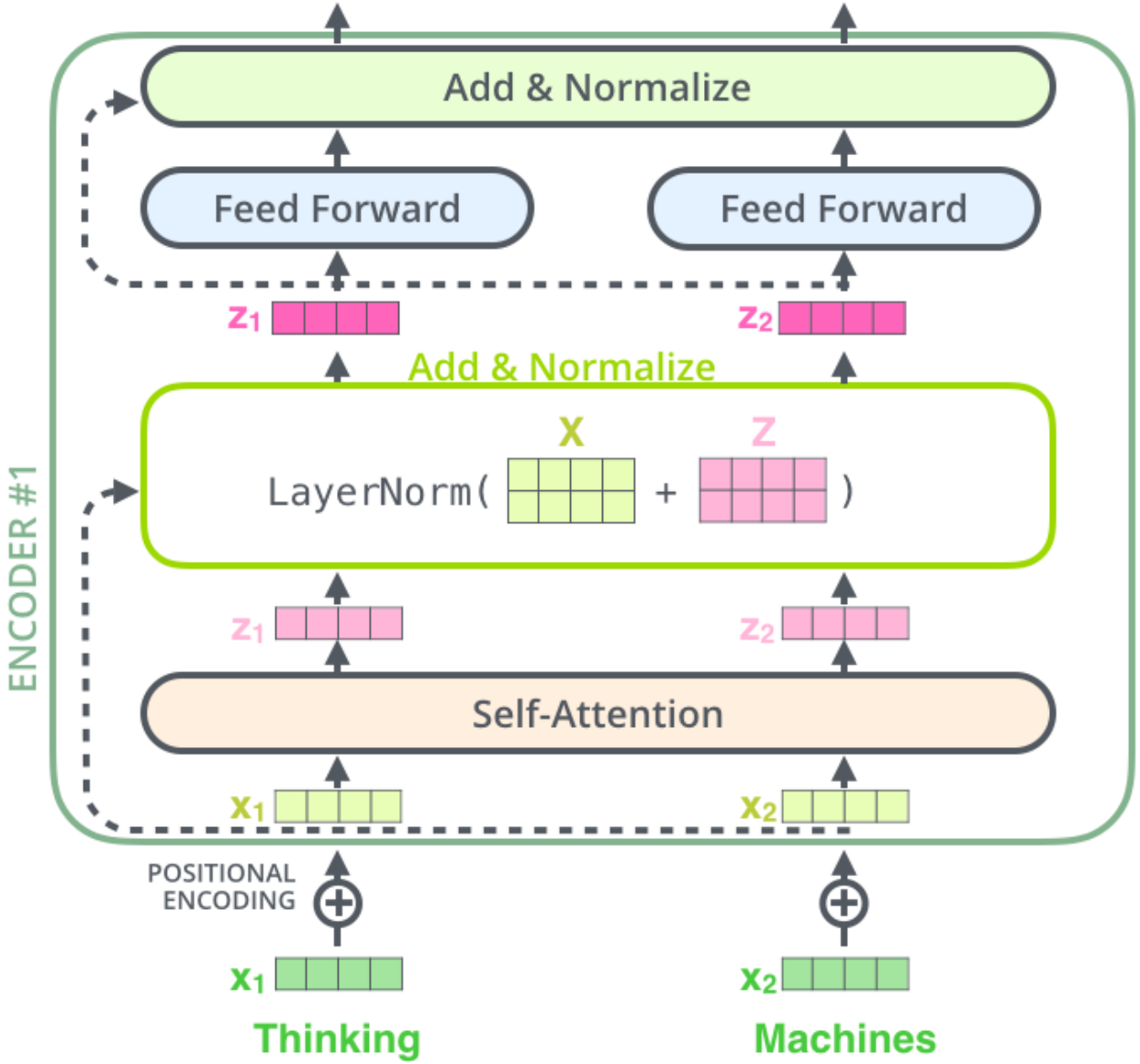
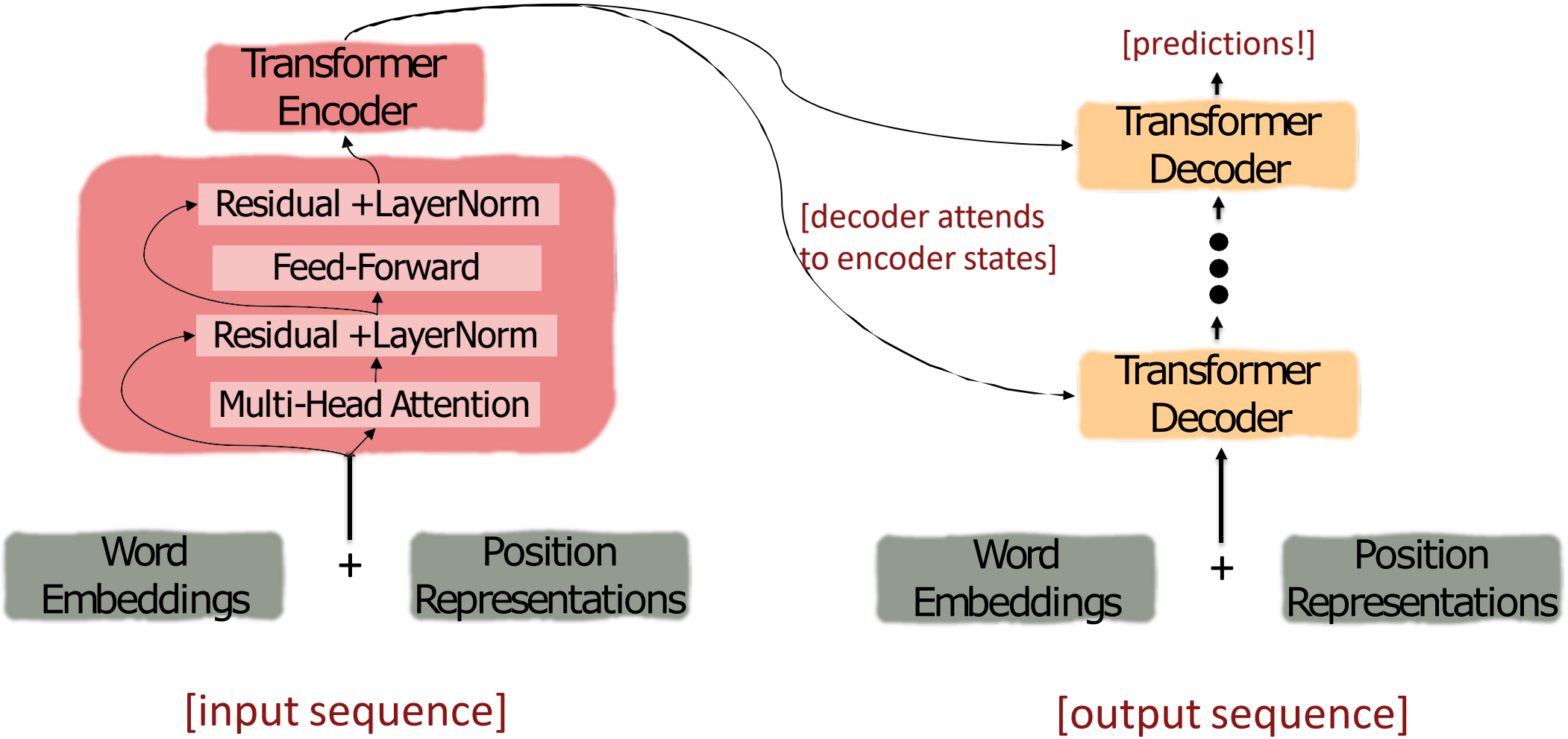



Image extracted from http://jalammar.github.io/illustrated-transformer/

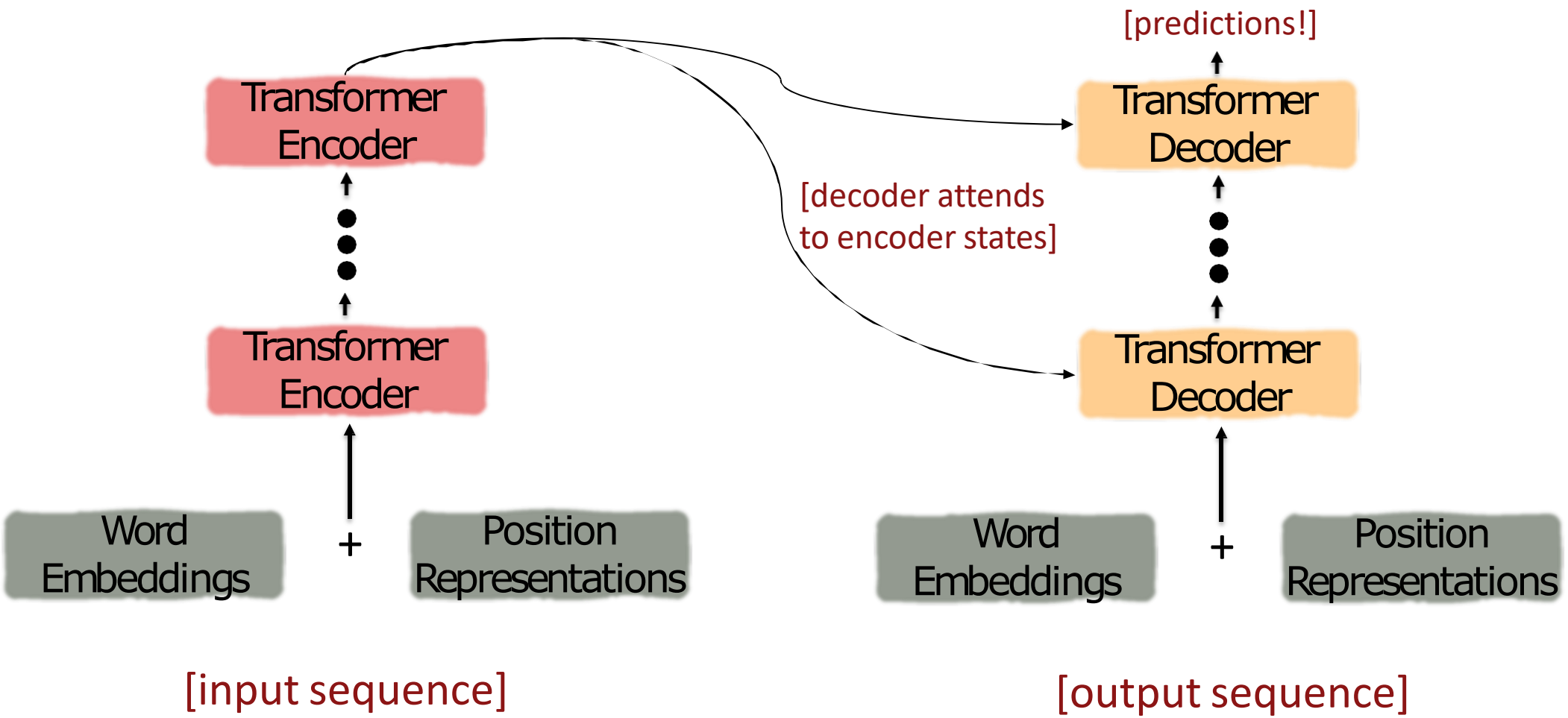
# Transformer – Encoder

Looking back at the whole model, zooming in on an Encoder block:



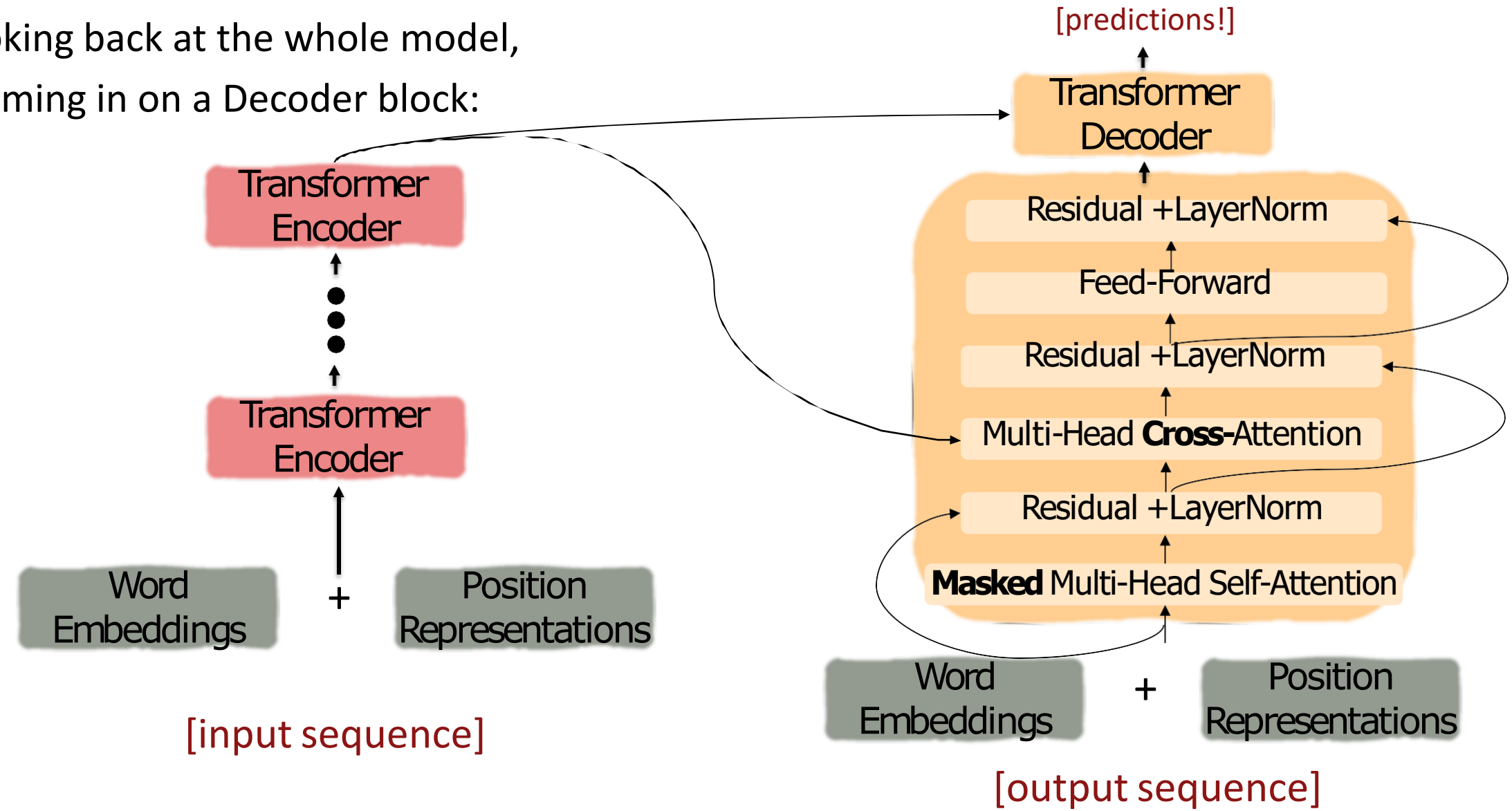
# Transformer – Encoders

Looking back at the whole model, zooming in on an Encoder block:



# Transformer – Decoder

Looking back at the whole model, zooming in on a Decoder block:



# Transformer – Decoder – Masked Attention

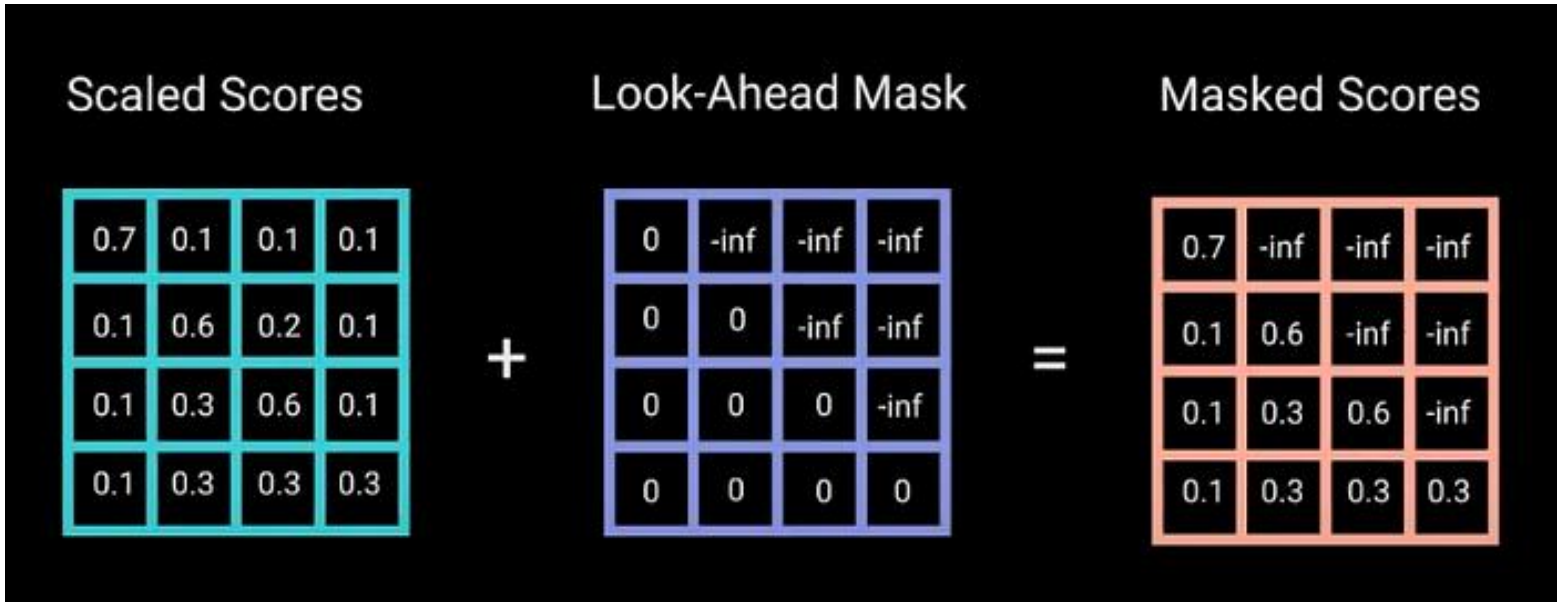



Image extracted from https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

# Transformer – Decoder – Masked Attention

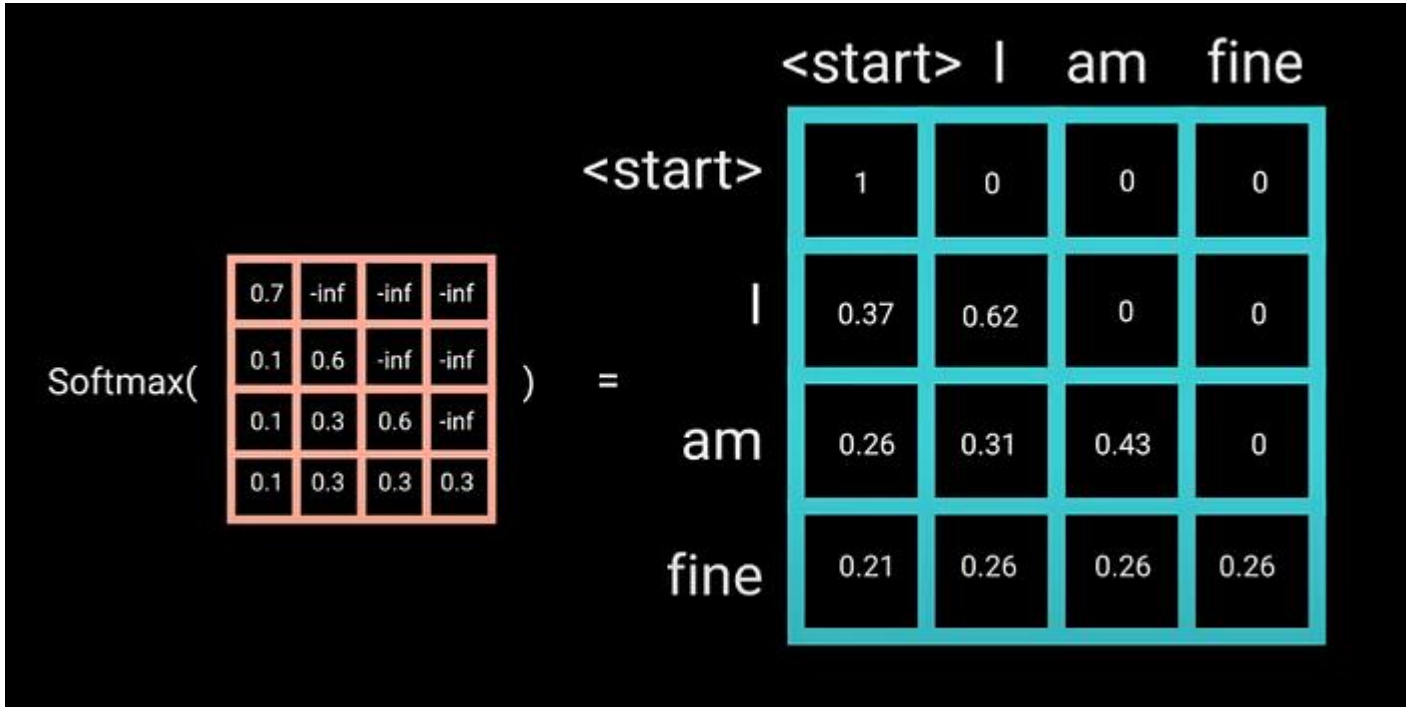
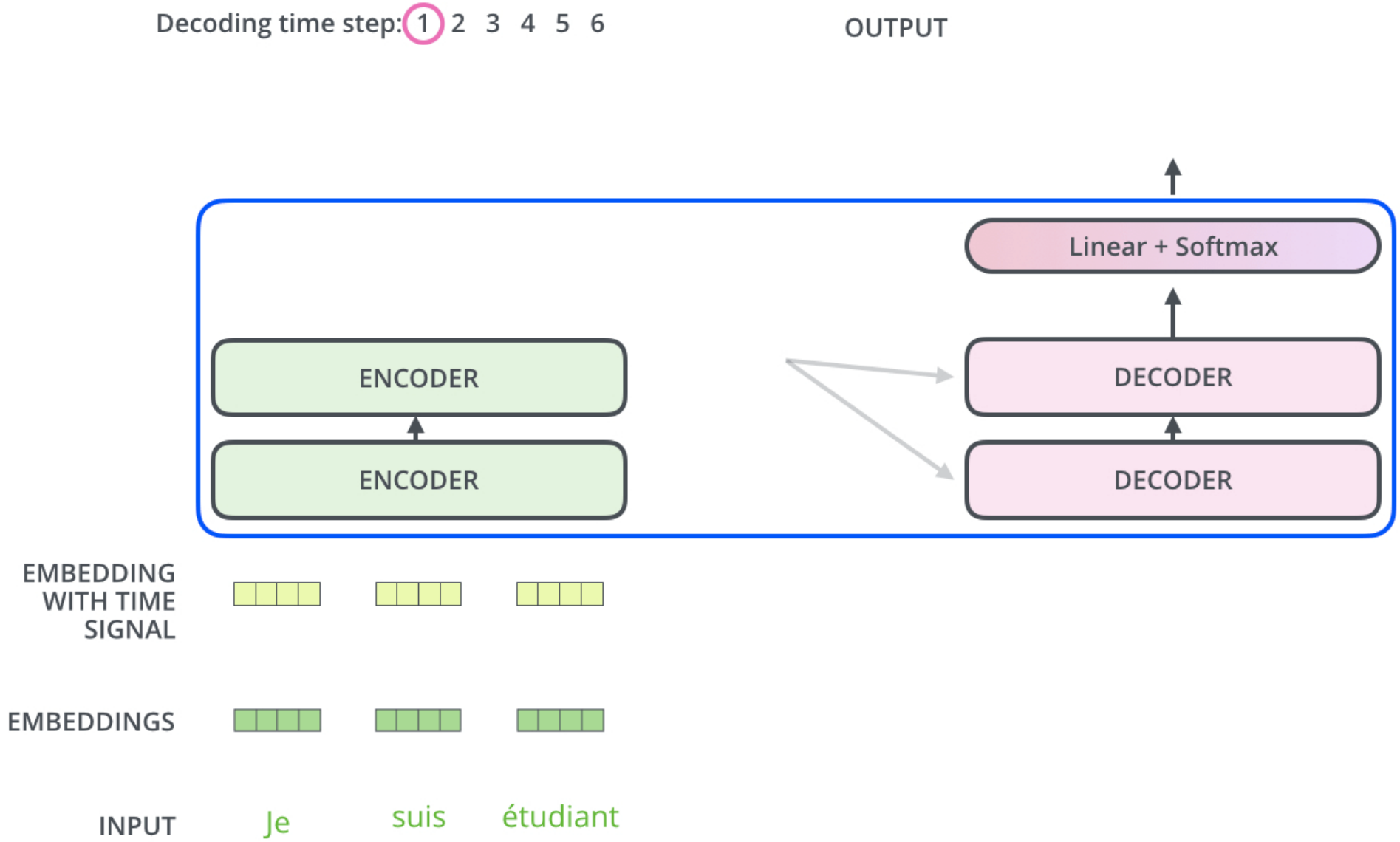
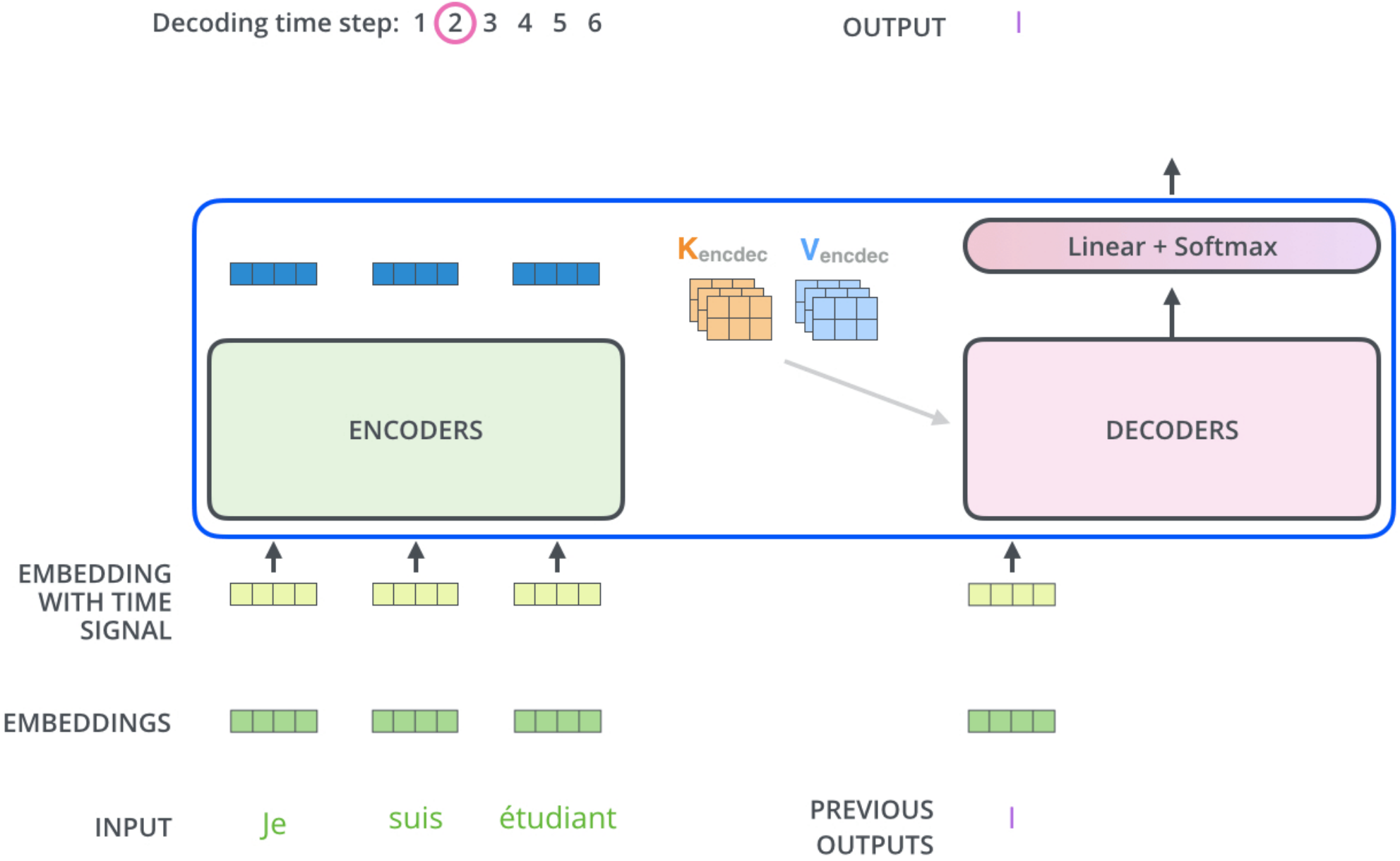



Image extracted from https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

# Transformer – Decoder – Cross Attention

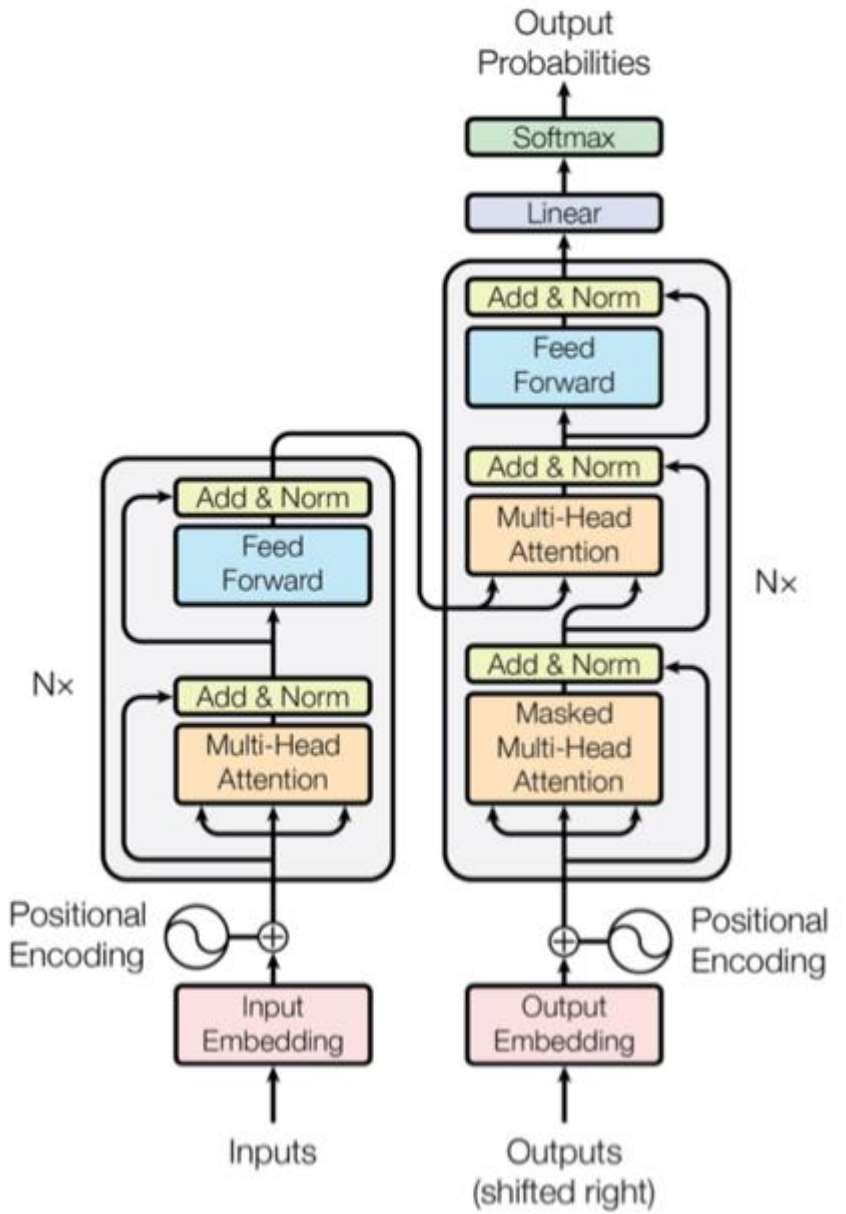


# Transformer – Decoder – Cross Attention



# Transformer – Three Attention – KQV

- **Encoder Self-attention**
  Q K V from source sentence
- **Decoder Masked Self-attention**
  Q K V from target sentence
- **Decoder-Encoder Cross-attention**
  Q from target sentence
  K V from source sentence



# Question

- When we use Transformer to translate, it sequentially generates next token $\mathbf{y_i}$ given the previously-generated tokens $\mathbf{y_{<i}}$ and the source input X, it is sequentially generating, one by one, why the paper claims that Transformer can be trained in parallel?

Parallel Training: Masked self-attention allows the model to take the whole target sentence as the input of decoder, process all tokens at the same time.

Sequential Inference: after the model has been well trained, when we use the model, it generates token one by one.

# Transformer Basics

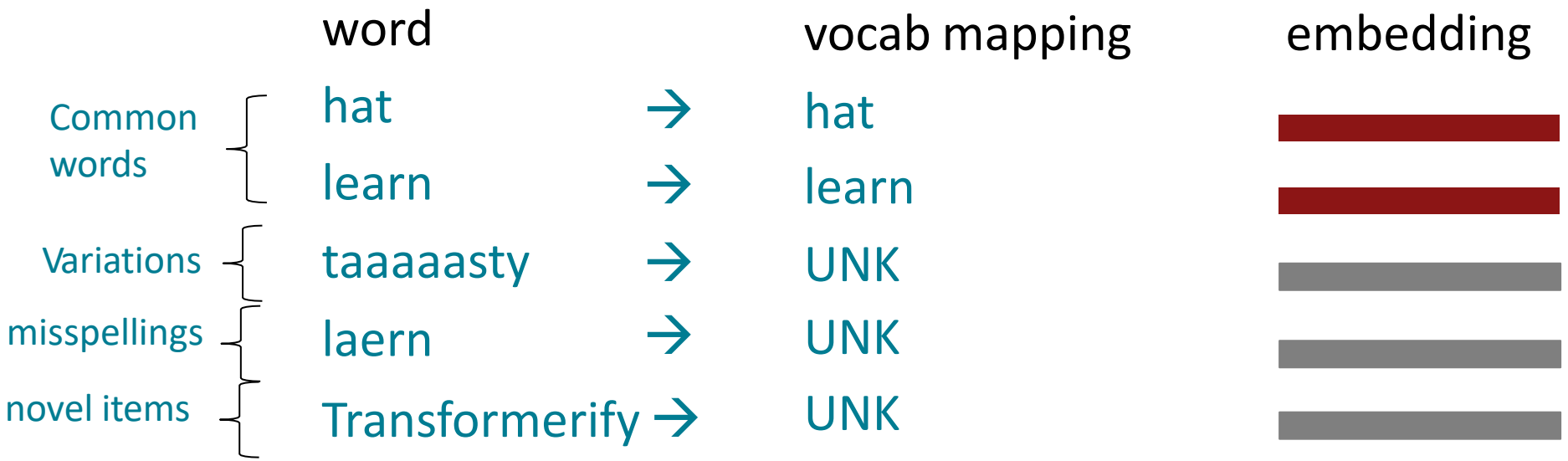
- Learning about transformers on your own?
  - The Annotated Transformer
    - http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - The Illustrated Transformer
    - http://nlp.seas.harvard.edu/2018/04/03/attention.html

# Subword Tokenization

# Traditional Tokenization

We assume a fixed vocab of tens of thousands of words, built from the training set.

All *novel* words seen at test time are mapped to a single UNK.



# Issues of Traditional Tokenization

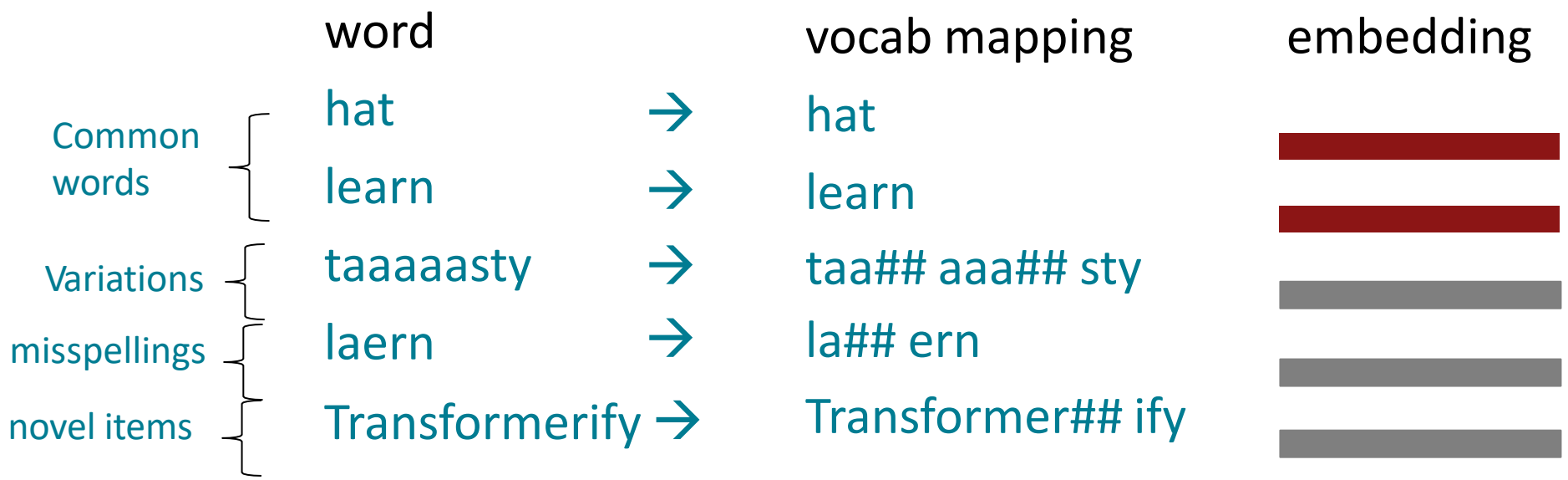
(1) Hard to split out-of-vocabulary words: many UNK words;
(2) Low-frequency words are less trained -> insufficient semantic meaning;
(3) Unable to learn the relationships between affixes. E.g., the relationship learned among *old, older, oldest* cannot generalize to *smart, smarter, smartest;*
(4) Character-level tokenization is too fine-grained.

How about subwords?

# Subword Models - BPE

Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.

In the worst case, words are split into as many subwords as they have characters.



# BPE: Byte-Pair Encoding

Given a training corpus: {“old”: 7, “older”: 3, “finest”: 9, “lowest”: 4}

1. **add** a special end token **“</w>”** at the end of each word.

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

The “</w>” token at the end of each word is added to **identify a word boundary** so that the algorithm knows **where each word ends**.

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

2. **split** each word into **characters** and **count** their **occurrence.**

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 |
| 2 | o | 14 |
| 3 | l | 14 |
| 4 | d | 10 |
| 5 | e | 16 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 |
| 11 | t | 13 |
| 12 | w | 4 |

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

3. look for the **most frequent pairing**, merge them, add to token list.

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 |
| 2 | o | 14 |
| 3 | l | 14 |
| 4 | d | 10 |
| 5 | e | 16 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 |
| 11 | t | 13 |
| 12 | w | 4 |

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

3. look for the **most frequent pairing**, merge them, add to token list. Iteration 1: “es”

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 |
| 2 | o | 14 |
| 3 | l | 14 |
| 4 | d | 10 |
| 5 | e | 16 - 13 = 3 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 - 13 = 0 |
| 11 | t | 13 |
| 12 | w | 4 |
| 13 | es | 9 + 4 = 13 |

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

3. look for the **most frequent pairing**, merge them, add to token list. Iteration 2: “est”

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 |
| 2 | o | 14 |
| 3 | l | 14 |
| 4 | d | 10 |
| 5 | e | 16 - 13 = 3 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 - 13 = 0 |
| 11 | t | 13 - 13 = 0 |
| 12 | w | 4 |
| 13 | es | 9 + 4 = 13 - 13 = 0 |
| 14 | est | 13 |

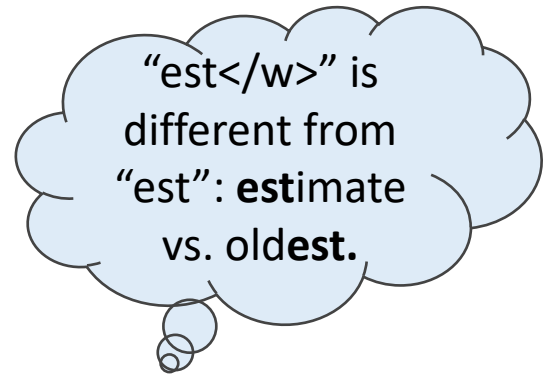
Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

3. look for the **most frequent pairing**, merge them, add to token list. Iteration 3: “est</w>”

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 - 13 = 10 |
| 2 | o | 14 |
| 3 | l | 14 |
| 4 | d | 10 |
| 5 | e | 16 - 13 = 3 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 - 13 = 0 |
| 11 | t | 13 - 13 = 0 |
| 12 | w | 4 |
| 13 | es | 9 + 4 = 13 - 13 = 0 |
| 14 | est | 13 - 13 = 0 |
| 15 | est</w> | 13 |




Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

4. **repeat 3** until reach the vocabulary size limit or iteration limit. Iterations: “ol”, “old” ->

| Number | Token | Frequency |
|---|---|---|
| 1 | </w> | 23 - 13 = 10 |
| 2 | o | 14 - 10 = 4 |
| 3 | l | 14 - 10 = 4 |
| 4 | d | 10 - 10 = 0 |
| 5 | e | 16 - 13 = 3 |
| 6 | r | 3 |
| 7 | f | 9 |
| 8 | i | 9 |
| 9 | n | 9 |
| 10 | s | 13 - 13 = 0 |
| 11 | t | 13 - 13 = 0 |
| 12 | w | 4 |
| 13 | es | 9 + 4 = 13 - 13 = 0 |
| 14 | est | 13 - 13 = 0 |
| 15 | est</w> | 13 |
| 16 | ol | 7 + 3 = 10 - 10 = 0 |
| 17 | old | 7 + 3 = 10 |

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE: Byte-Pair Encoding

{“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}

5. Apply learned merging rules to encode.

- Merging increases the vocabulary size
- Allow us represent a word with the least number of tokens

Example from: https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

# BPE vs. WordPiece

- **Subword Merging Criterion**
  - BPE: Selects and merges the most frequent pair in each iteration.
  - WordPiece: Calculates scores for pairs and selects the highest scoring pair for merging.

- **Tokenization of New Text**
  - BPE: Applies learned merge rules sequentially.
  - WordPiece: Uses a longest match approach, incorporating UNK for unknown words and HASH for non-initial subwords.

See more here.

# Take-away

Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of **parts of words (subword tokens).**
- At training and testing time, each word is split into a sequence of known subwords.

**Byte-pair encoding** is a simple, effective strategy for defining a subword vocabulary.

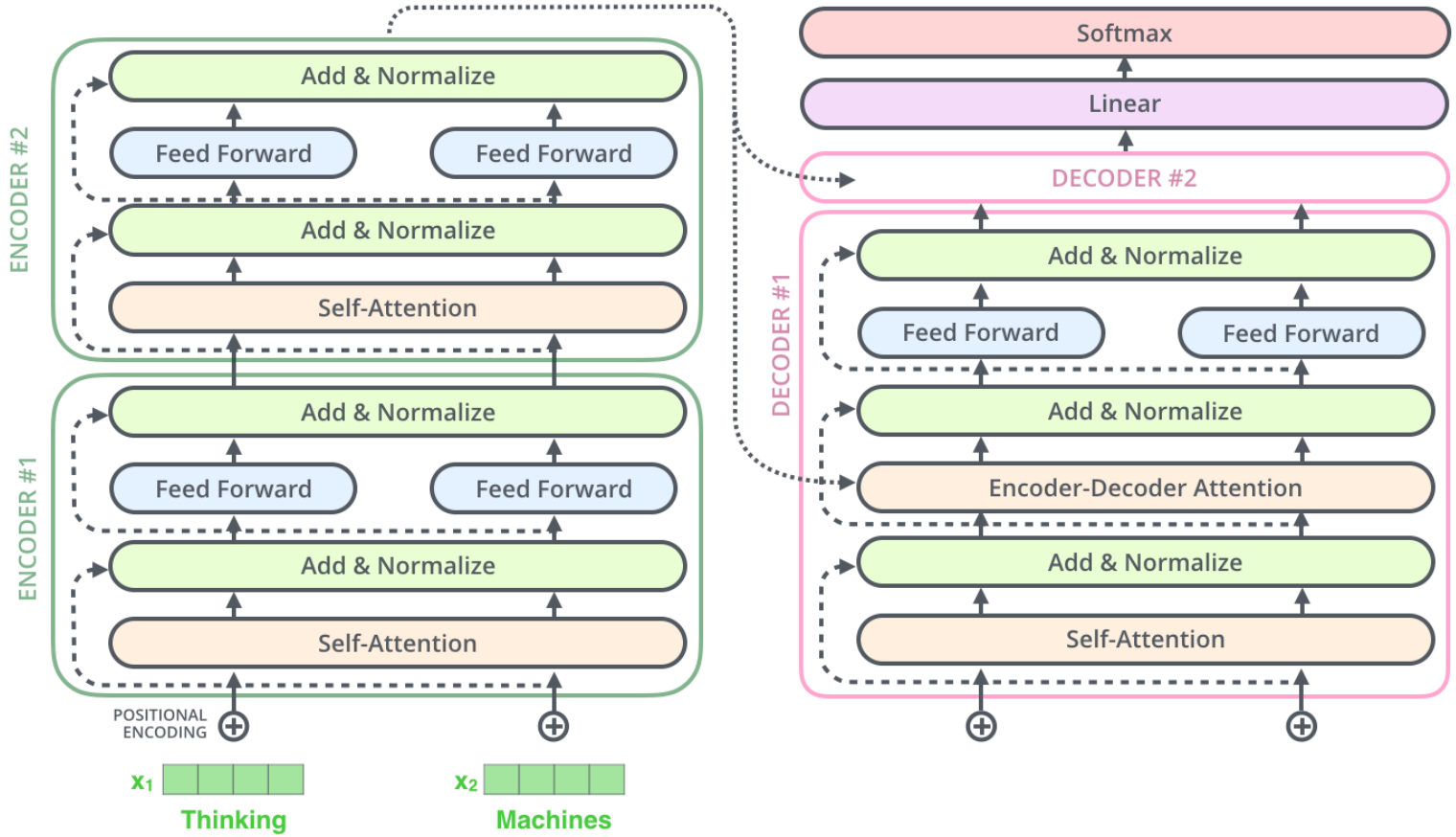
1. Start with a vocabulary containing only characters and an “end-of-word” symbol.
2. Using a corpus of text, find the most common pair of adjacent characters “a,b”; add subword “ab” to the vocab.
3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

[Sennrich et al., 2016, Wu et al., 2016]

# Encoder/Decoder-Only Transformers: BERT and GPT-2

# BERT, GPT, and the Transformer




BERT

GPT

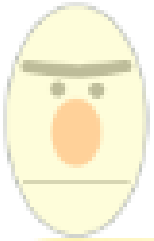
# BERT, GPT, and the Transformer

- A transformer uses an Encoder stack to model the input, and a Decoder stack to model the output (using input information from encoder side).

- If we have no input and we just want to model the "next word", we can get rid of the Encoder. This gives us GPT.

- If we are only interested in training a language model for the input for some other tasks, then we do not need the Decoder. This gives us BERT.

GPT-2



BERT

DECODER

• • •

DECODER

DECODER

ENCODER

• • •

ENCODER

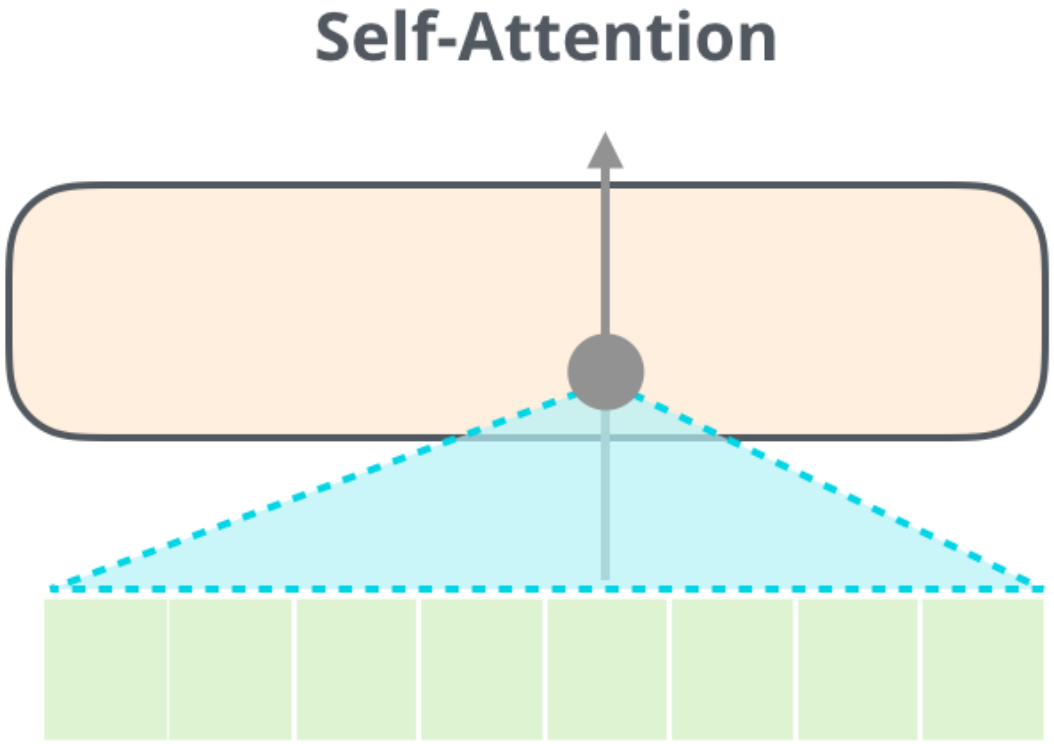
ENCODER

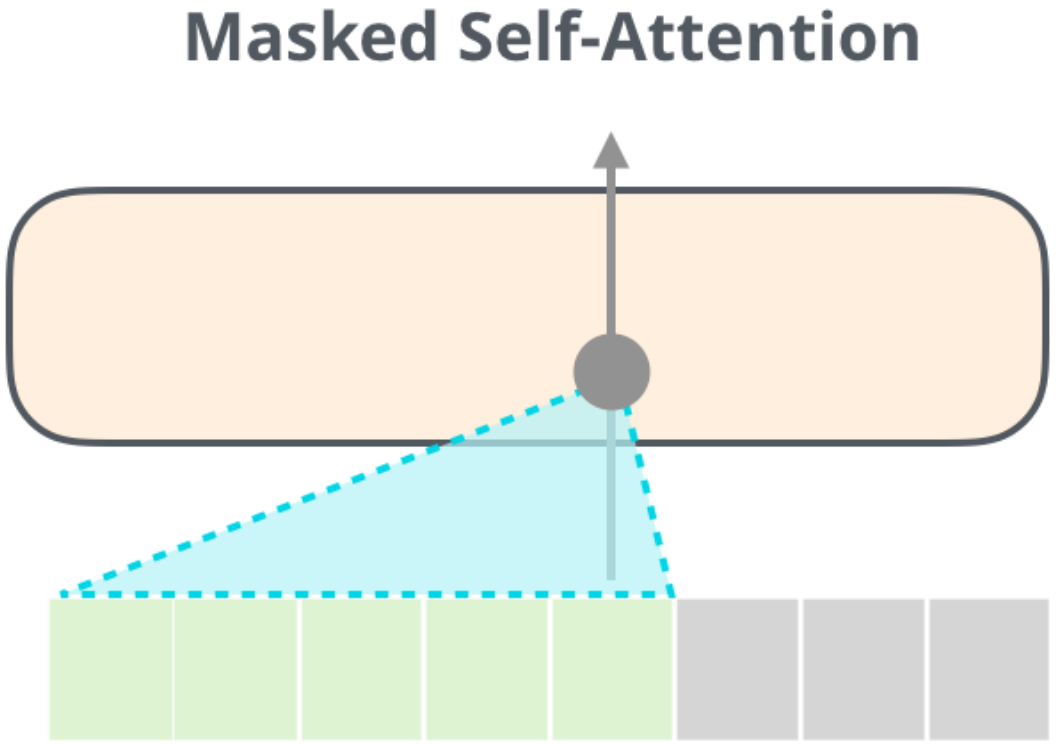
# GPT: Generative Pretrained Transformer

# GPT: Training

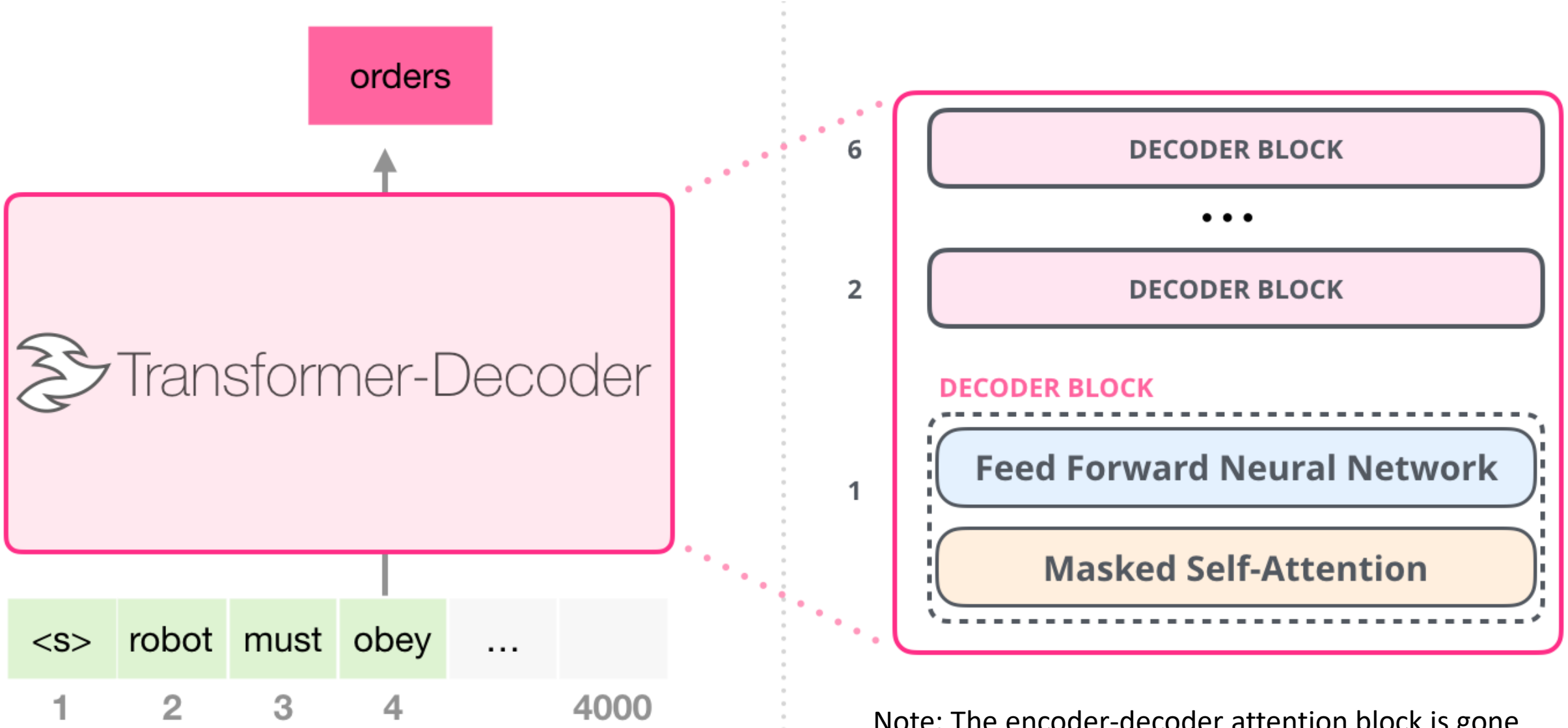
GPT-2 uses an unsupervised learning approach to train the language model.

# GPT: Masked Self-Attention



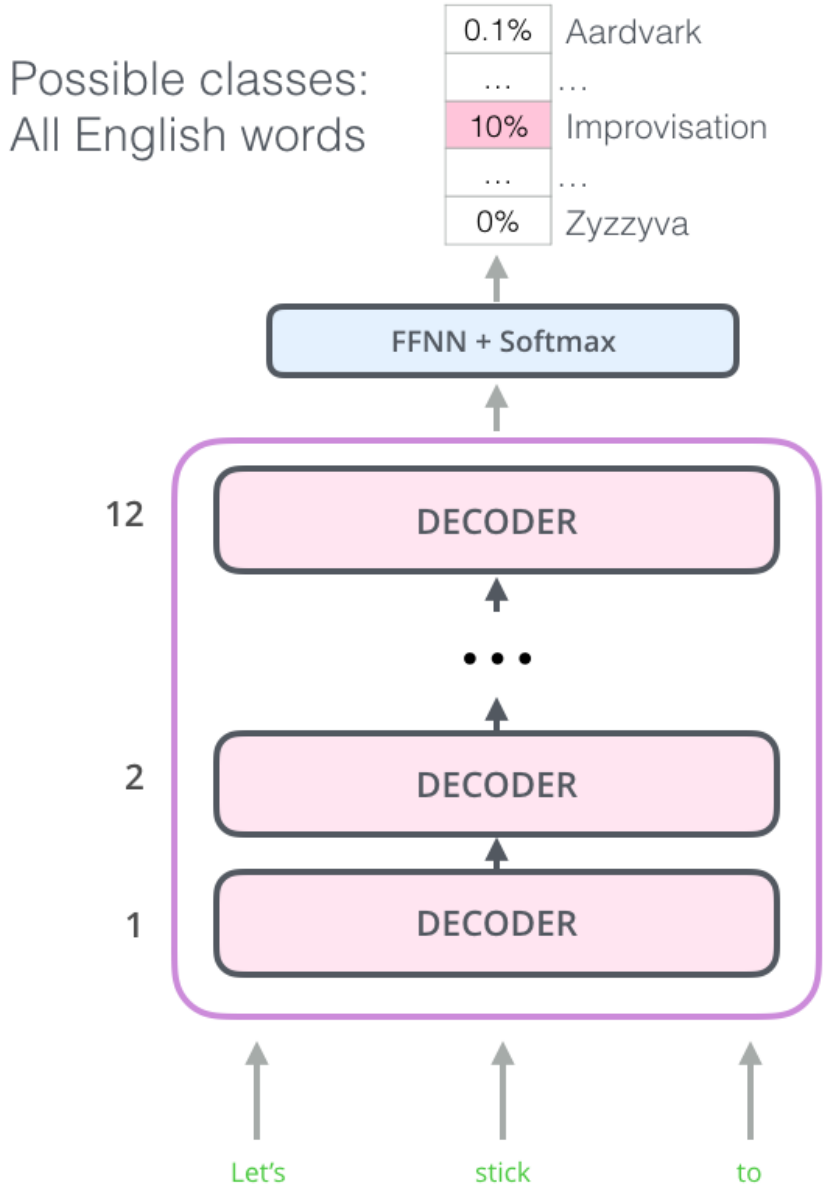



# Masked Self-Attention

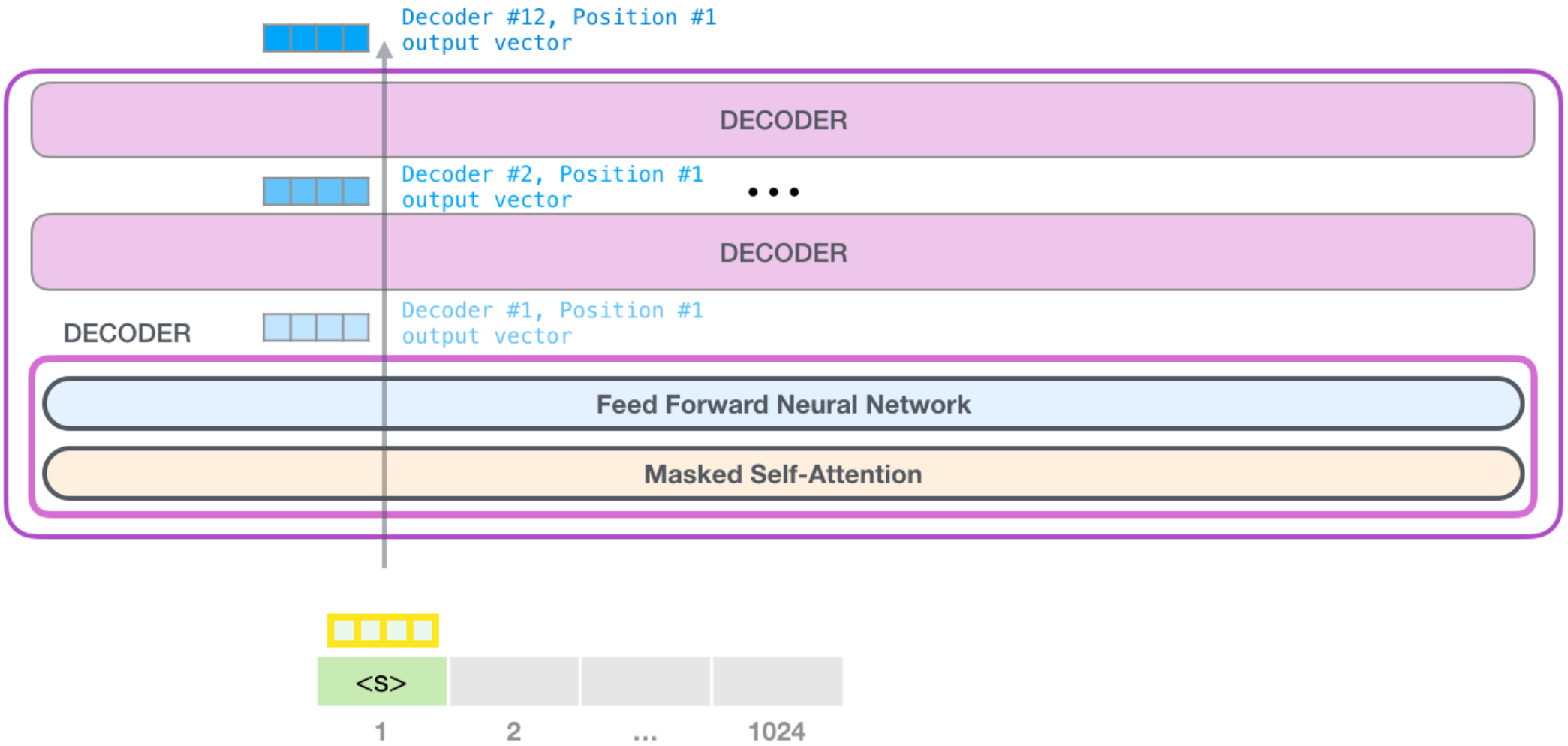



Note: The encoder-decoder attention block is gone

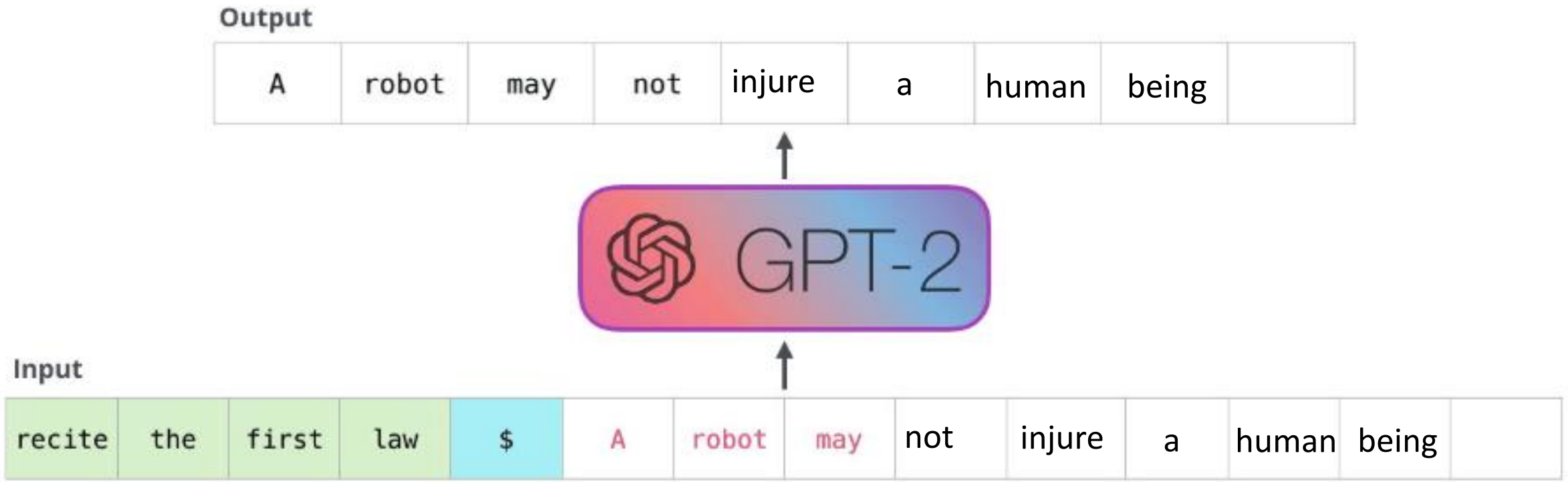
# GPT: Prediction



GPT-2 has a parameter top-*k*, so that we sample from the top *k* (highest probability from softmax) words for each output



# GPT-2 in Action



# A Story Generated by GPT-2

“The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

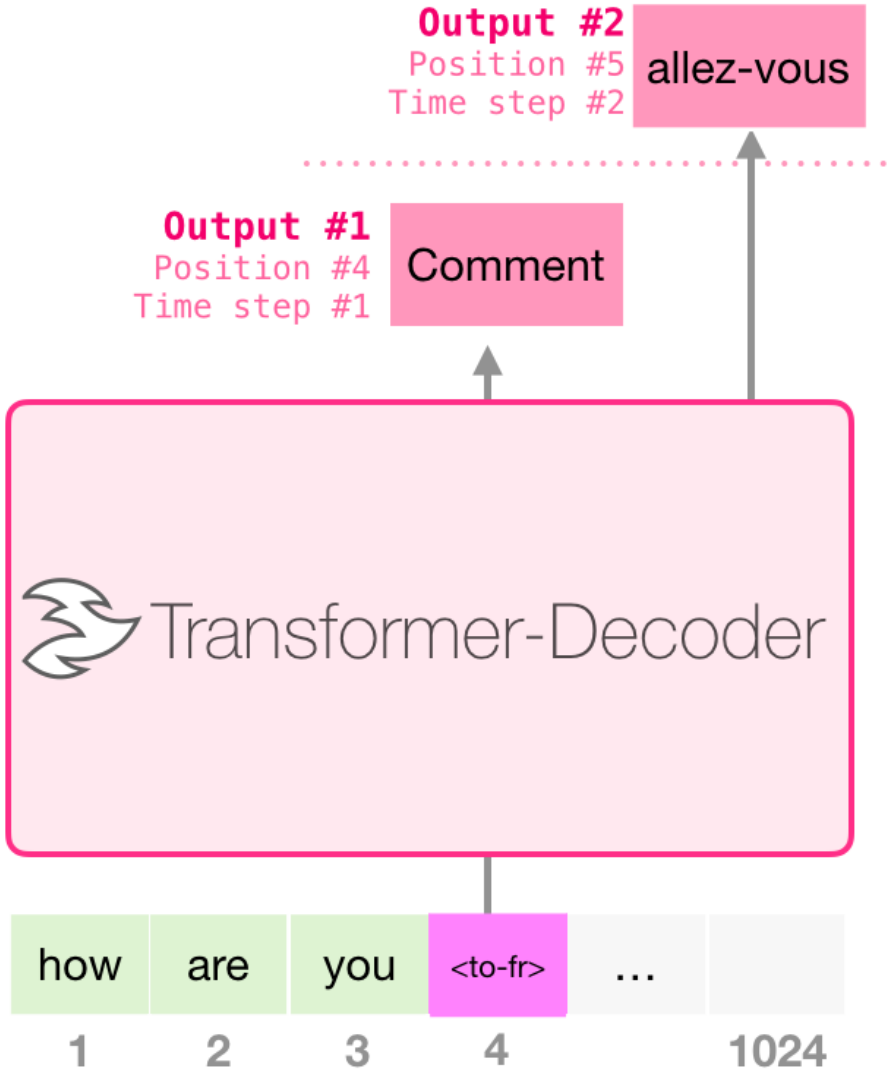
Pérez and the others then ventured further into the valley. `By the time we reached the top of one peak, the water looked blue, with some crystals on top,’ said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns."

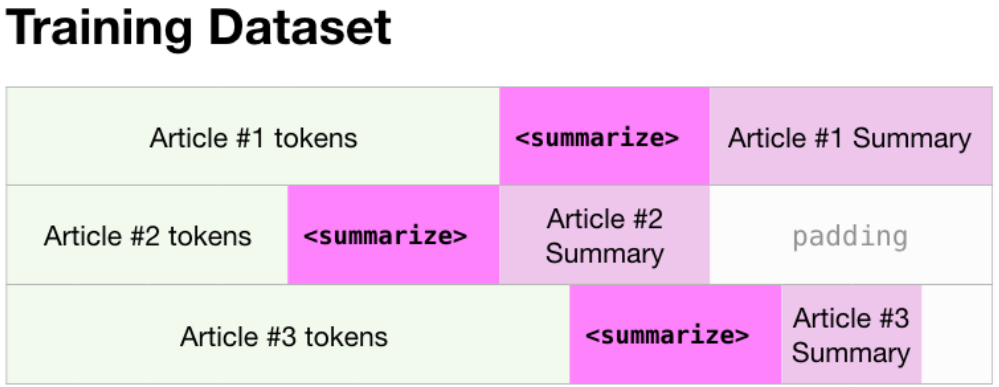
# GPT-2 Application: Translation

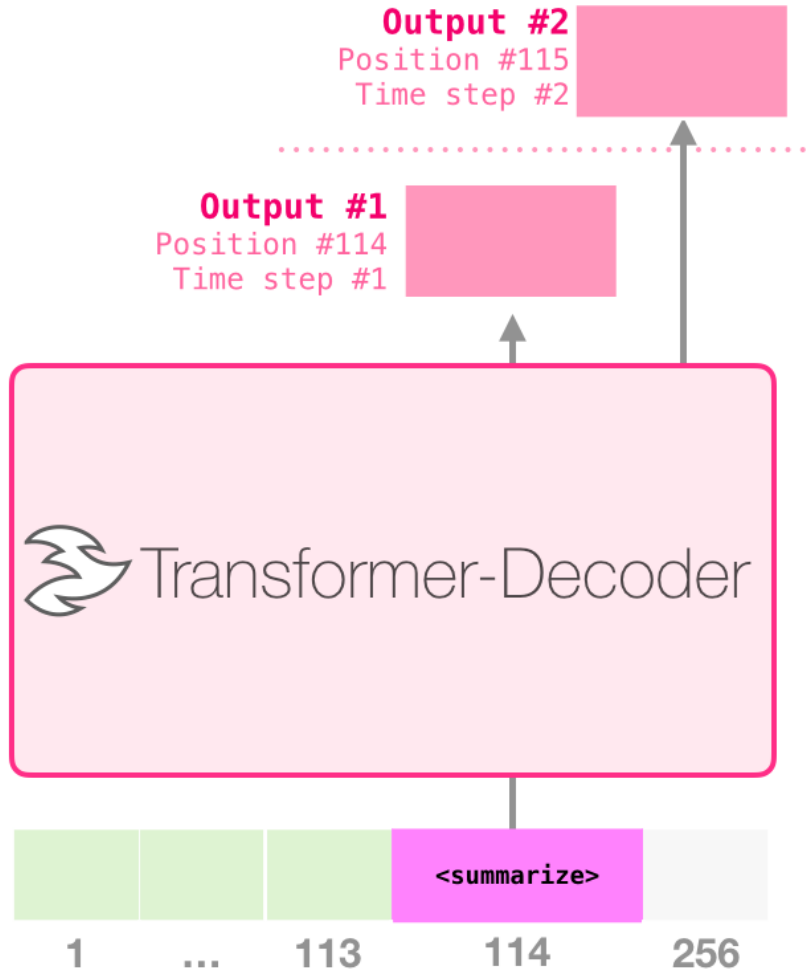
**Training Dataset**

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

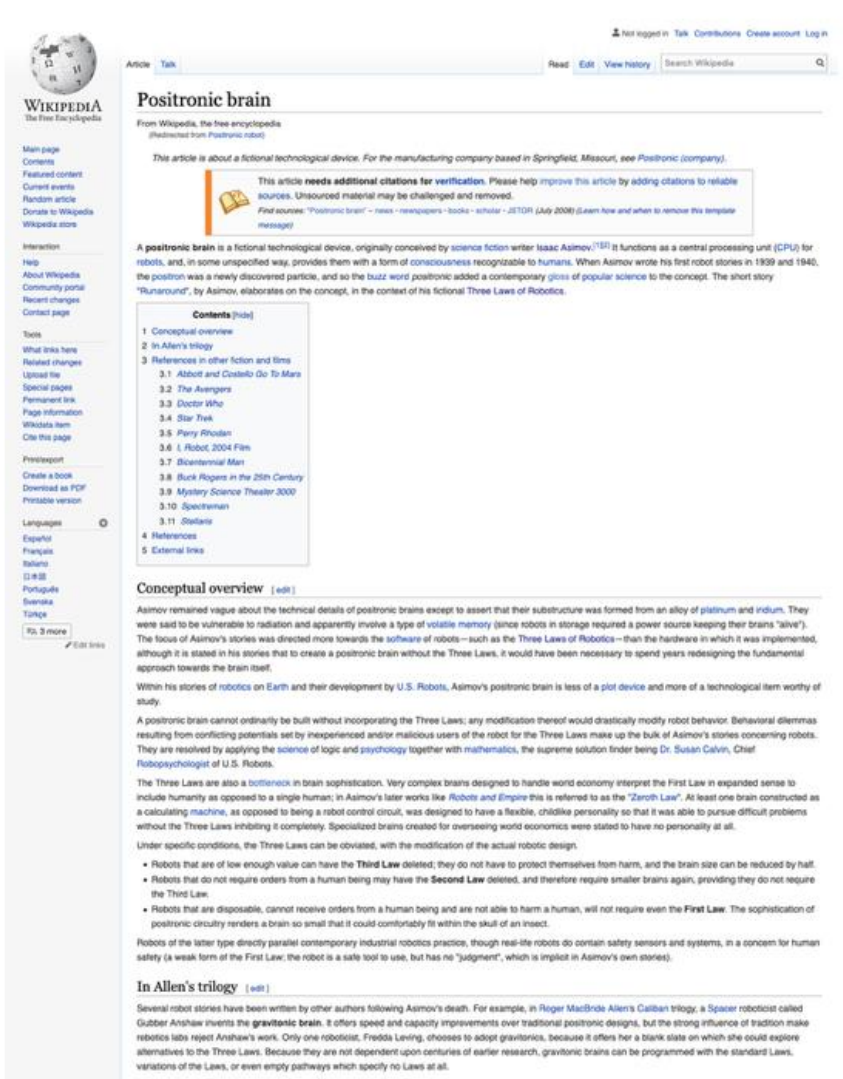


# GPT-2 Application: Summarization



# Summarization: Using Wikipedia Data for Training



Positronic brain

From Wikipedia, the free encyclopedia

*This article is about a fictional technological device. For the manufacturing company based in Springfield, Missouri, see Positronic (company).*

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.

A **positronic brain** is a fictional technological device, originally conceived by science fiction writer Isaac Asimov. 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 *positronic* 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 [hide]

1 Conceptual overview
2 In Allen's trilogy
3 References in other fiction and films
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 *Bicentennial Man*
3.8 *Buck Rogers in the 25th Century*
3.9 *Mystery Science Theater 3000*
3.10 *Spectreman*
3.11 *Stellaris*
4 References
5 External links

## Conceptual overview

Asimov remained vague about the technical details of positronic brains except to assert that their substructure was formed from an alloy of platinum and iridium. 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 robots—such as the Three Laws of Robotics—than the hardware in which it was implemented, although it is stated in his stories 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.

Within his 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.

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 Robopsychologist of U.S. Robots.

The Three Laws are also a bottleneck 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 "Zeroth Law". At least one brain constructed as a calculating machine, as opposed to being a robot control circuit, was designed to have a flexible, childlike personality so that it was able to pursue difficult problems without the Three Laws inhibiting it completely. Specialized brains created for overseeing world economics were stated to have no personality at all.

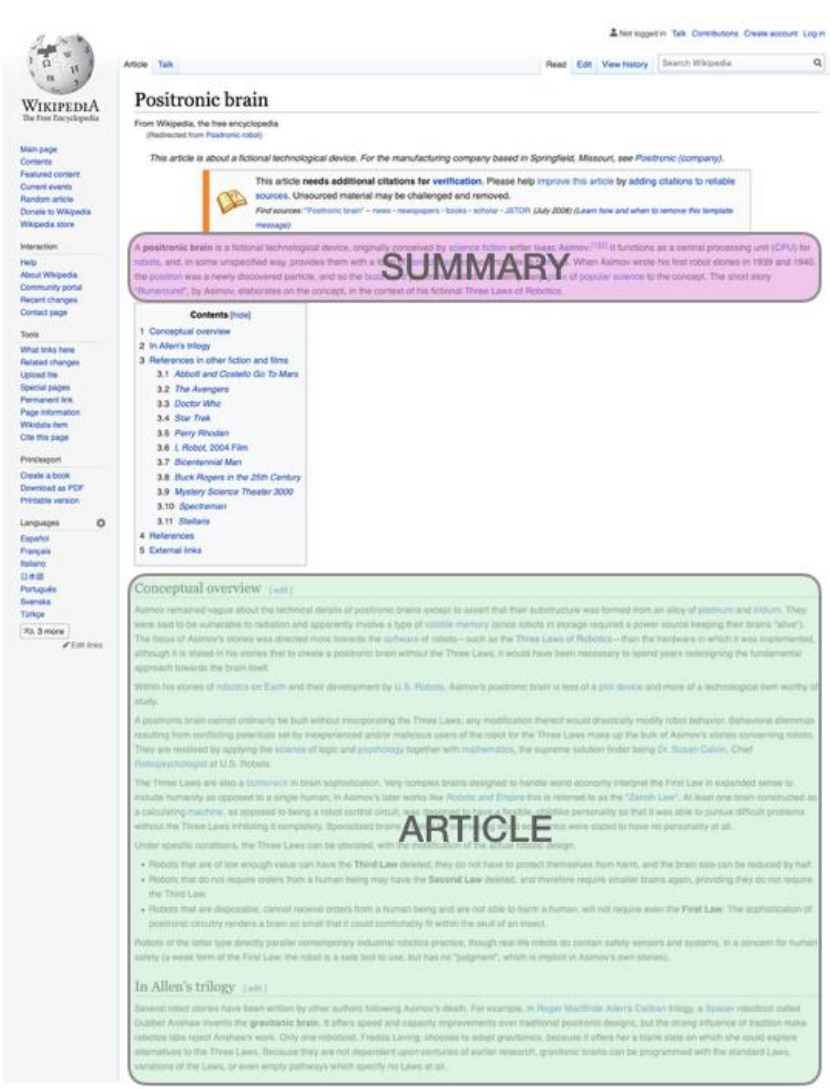
Under specific conditions, the Three Laws can be obviated, with the modification of the actual robotic design.

- 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 Third 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.

Robots 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).

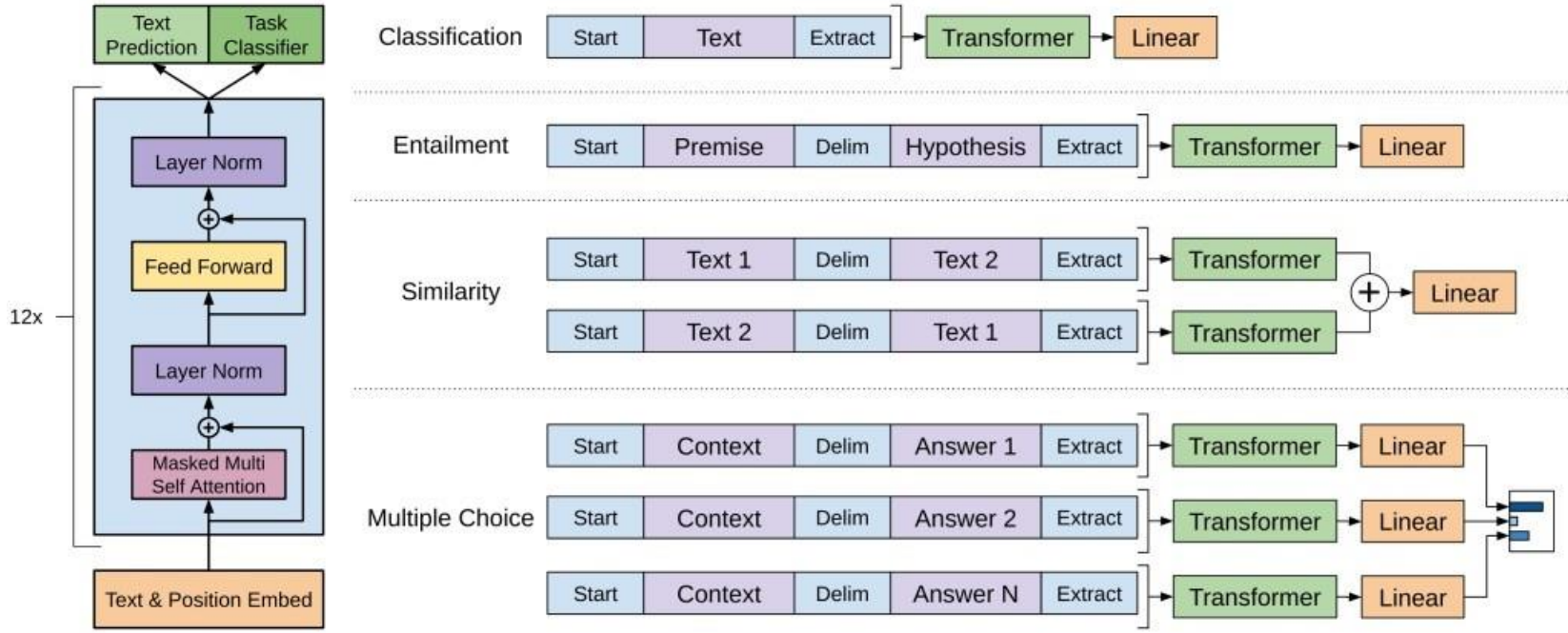
## In Allen's trilogy

Several robot stories have been written by other authors following Asimov's death. For example, in Roger MacBride Allen's Caliban trilogy, a Spacer roboticist called Gubber Anshaw invents the **gravitonic brain**. It offers speed and capacity improvements over traditional positronic designs, but the strong influence of tradition make robotics labs reject Anshaw's work. Only one roboticist, Fredda Leving, chooses to adopt gravitonics, 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, gravitonic brains can be programmed with the standard Laws, variations of the Laws, or even empty pathways which specify no Laws at all.



# Fine-tune GPT-2 for Other Tasks

How do we format inputs to our decoder for **finetuning tasks?**

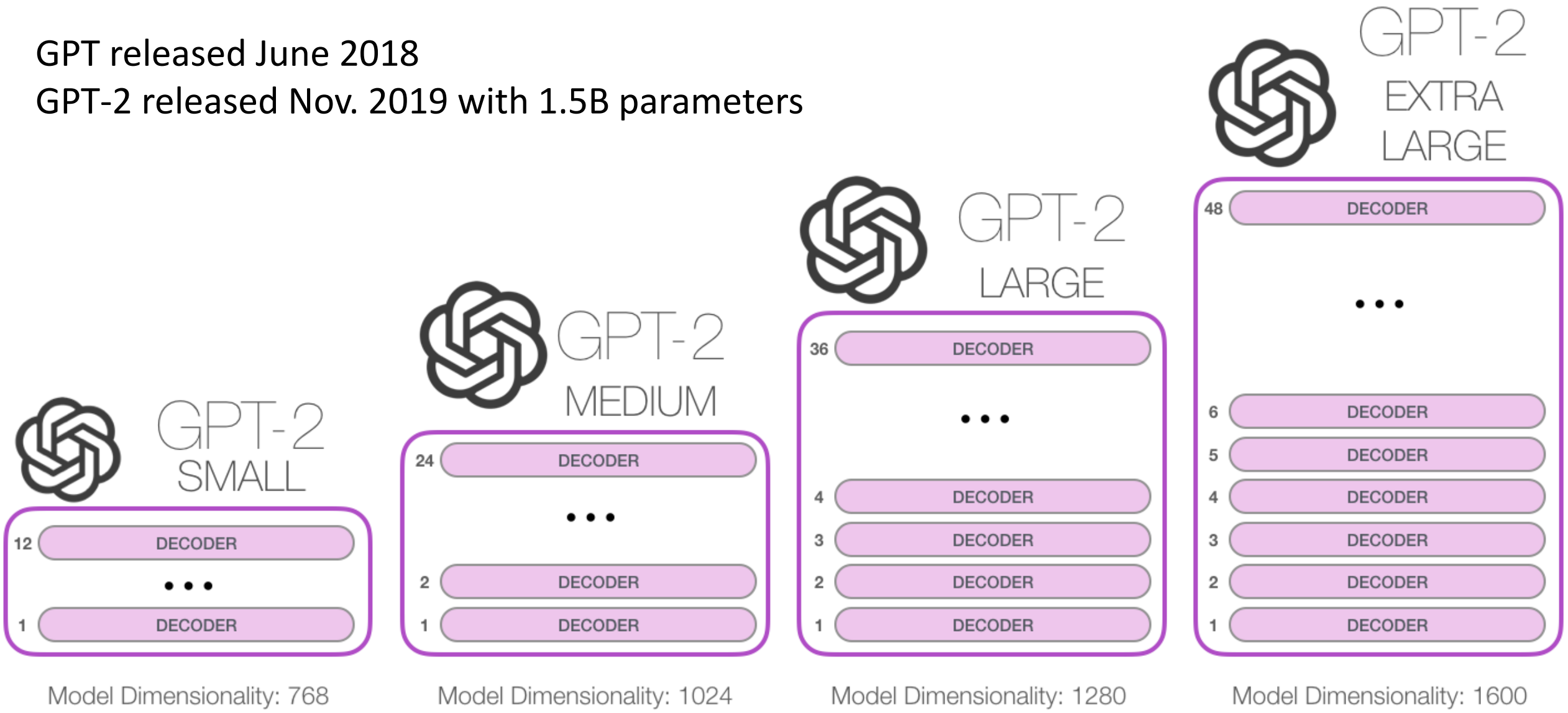



The linear classifier is applied to the representation of the [EXTRACT] token.

# GPT: Size Matters

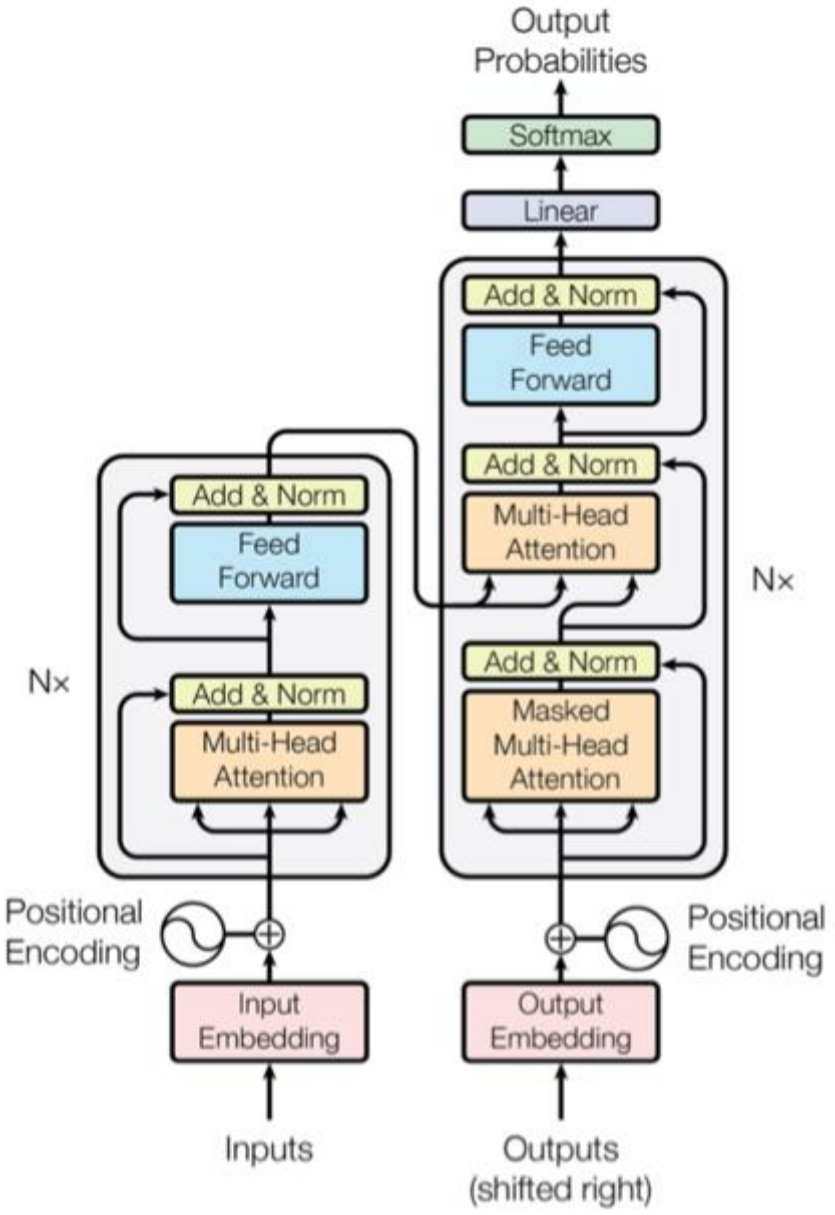
GPT released June 2018

GPT-2 released Nov. 2019 with 1.5B parameters

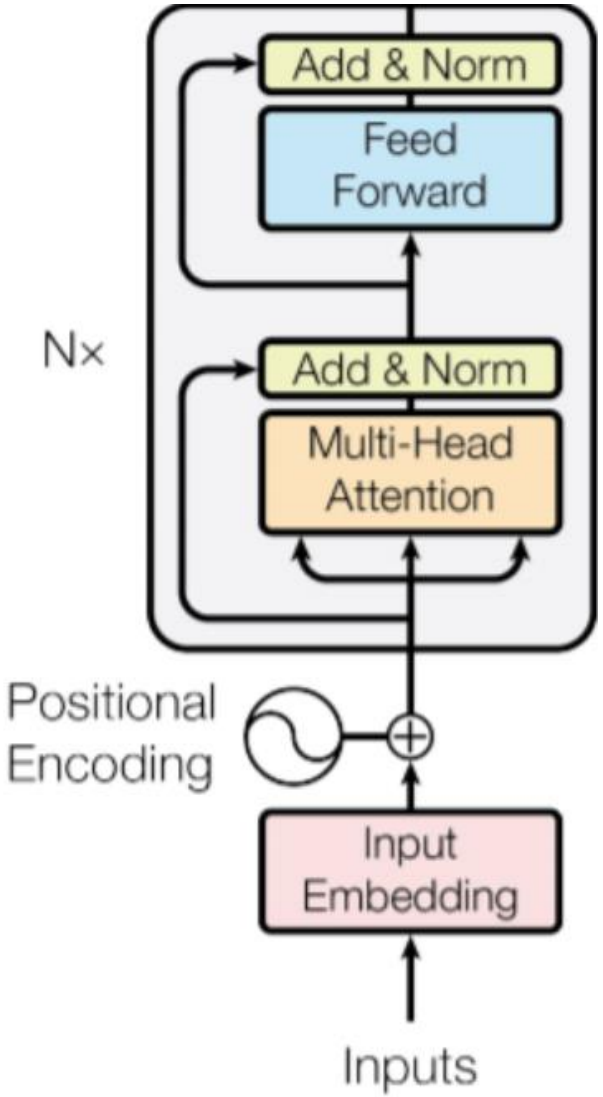


# Architecture of BERT vs. GPT-2

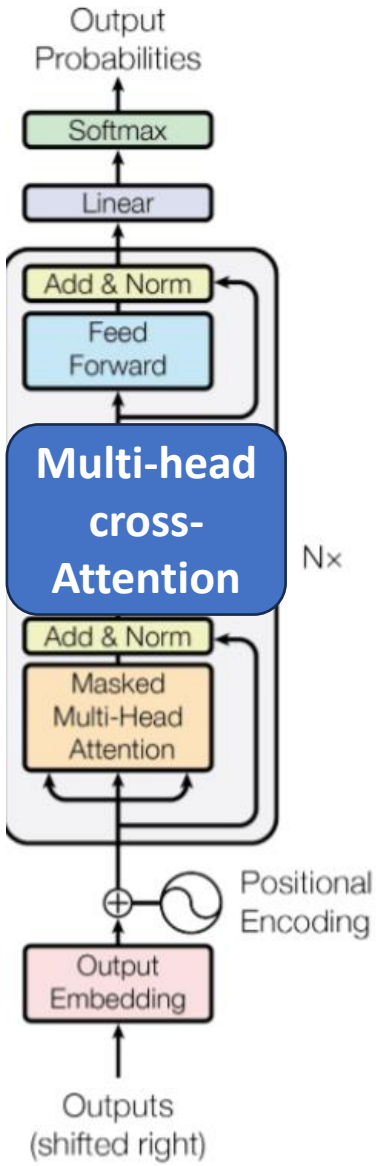
# Transformer



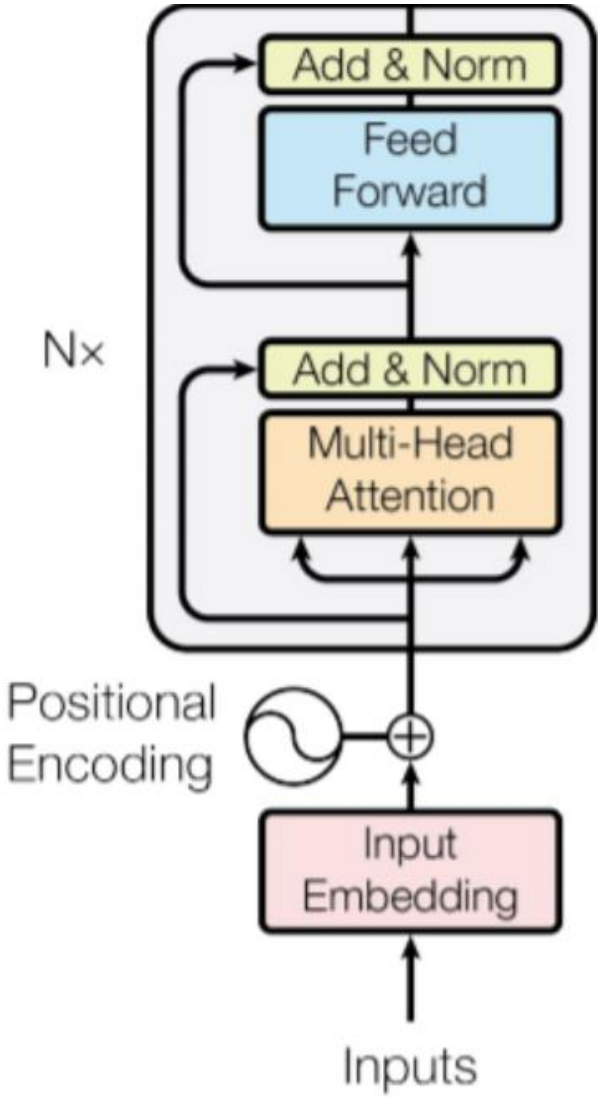
# Encoder-only vs. Decoder-only



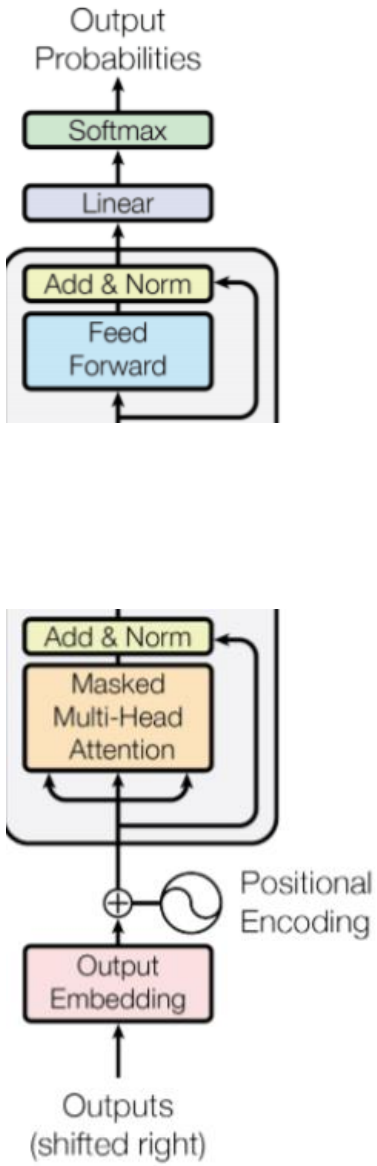

BERT



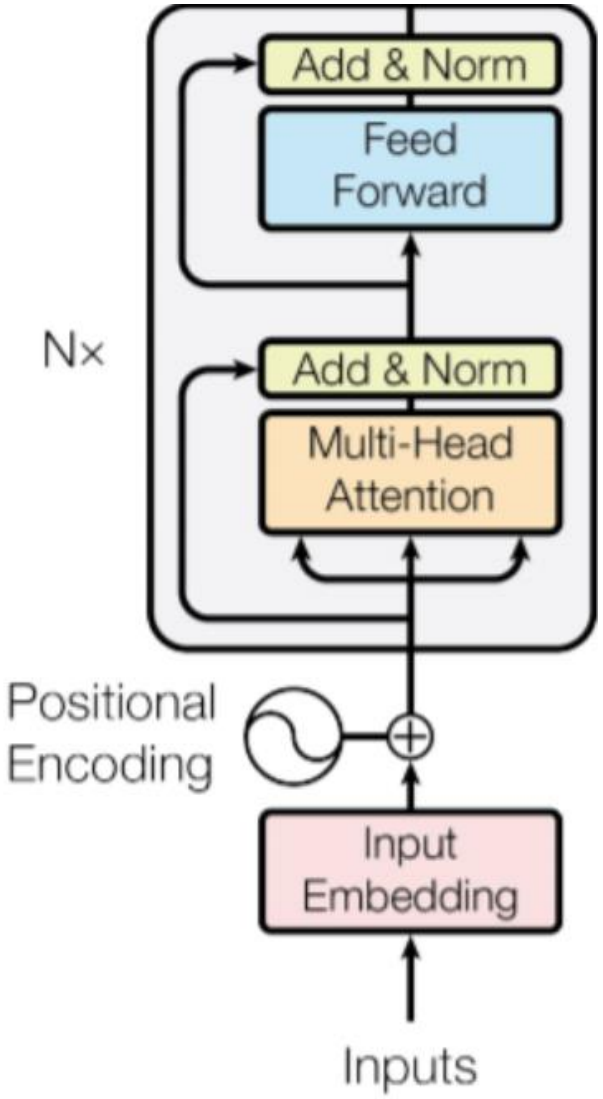
# BERT vs. GPT-2



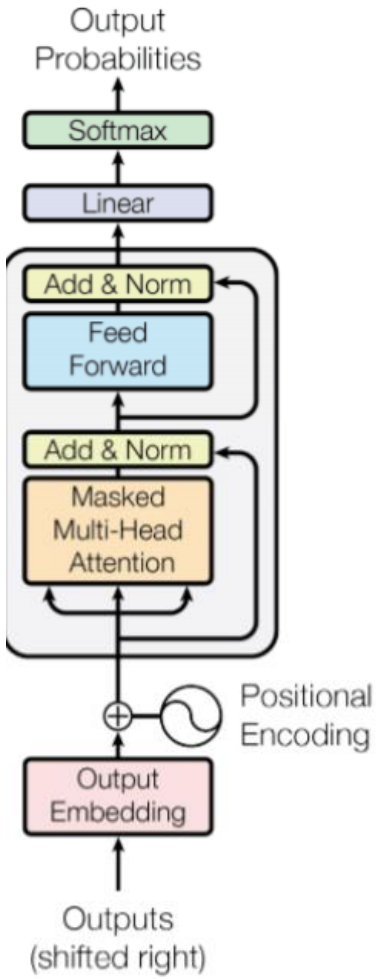

BERT



# BERT vs. GPT-2

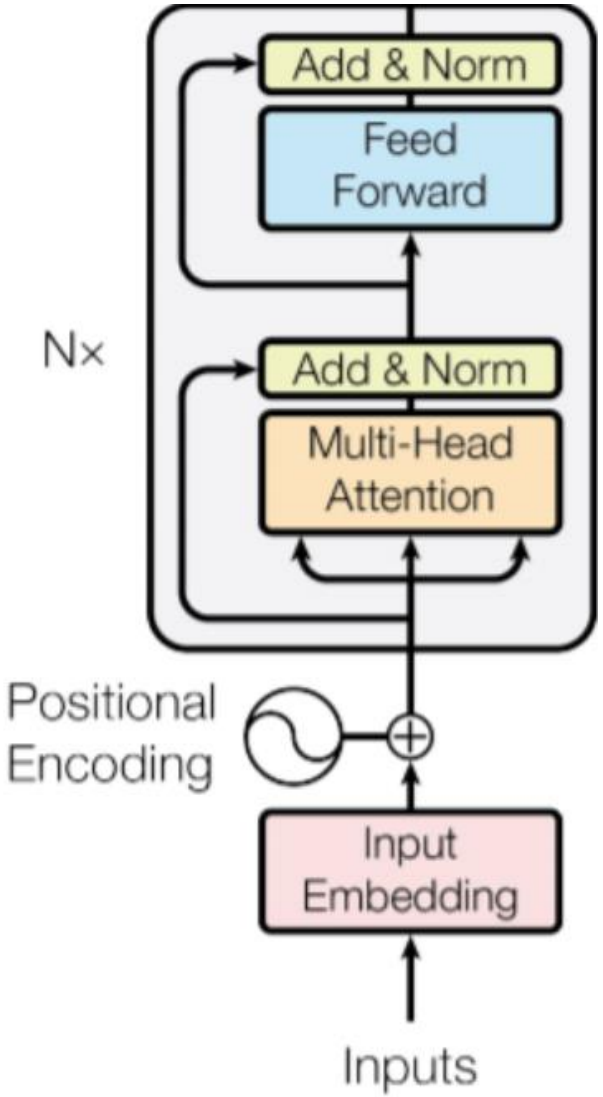



BERT

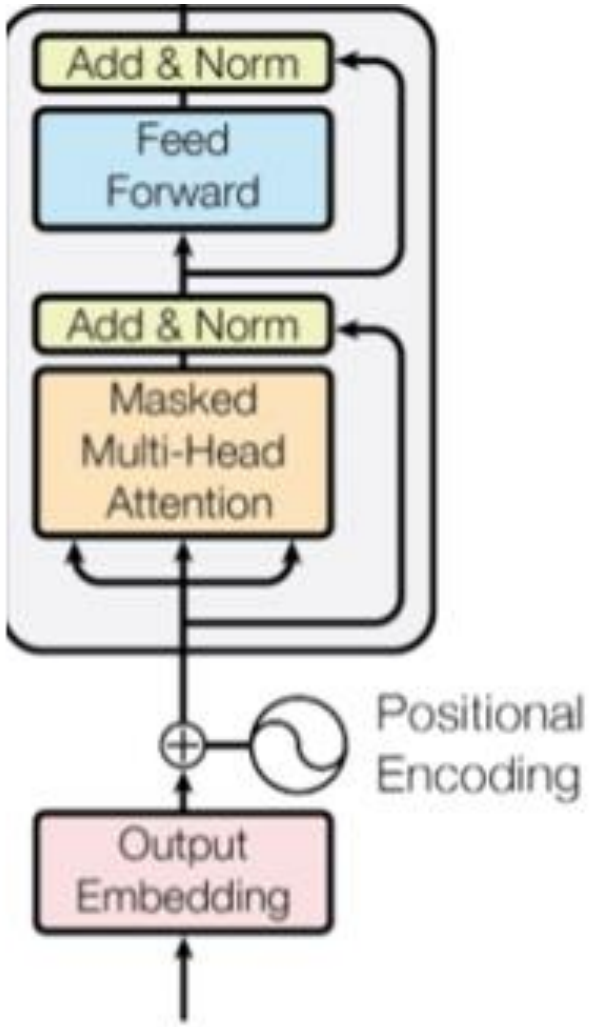



GPT

# BERT vs. GPT-2

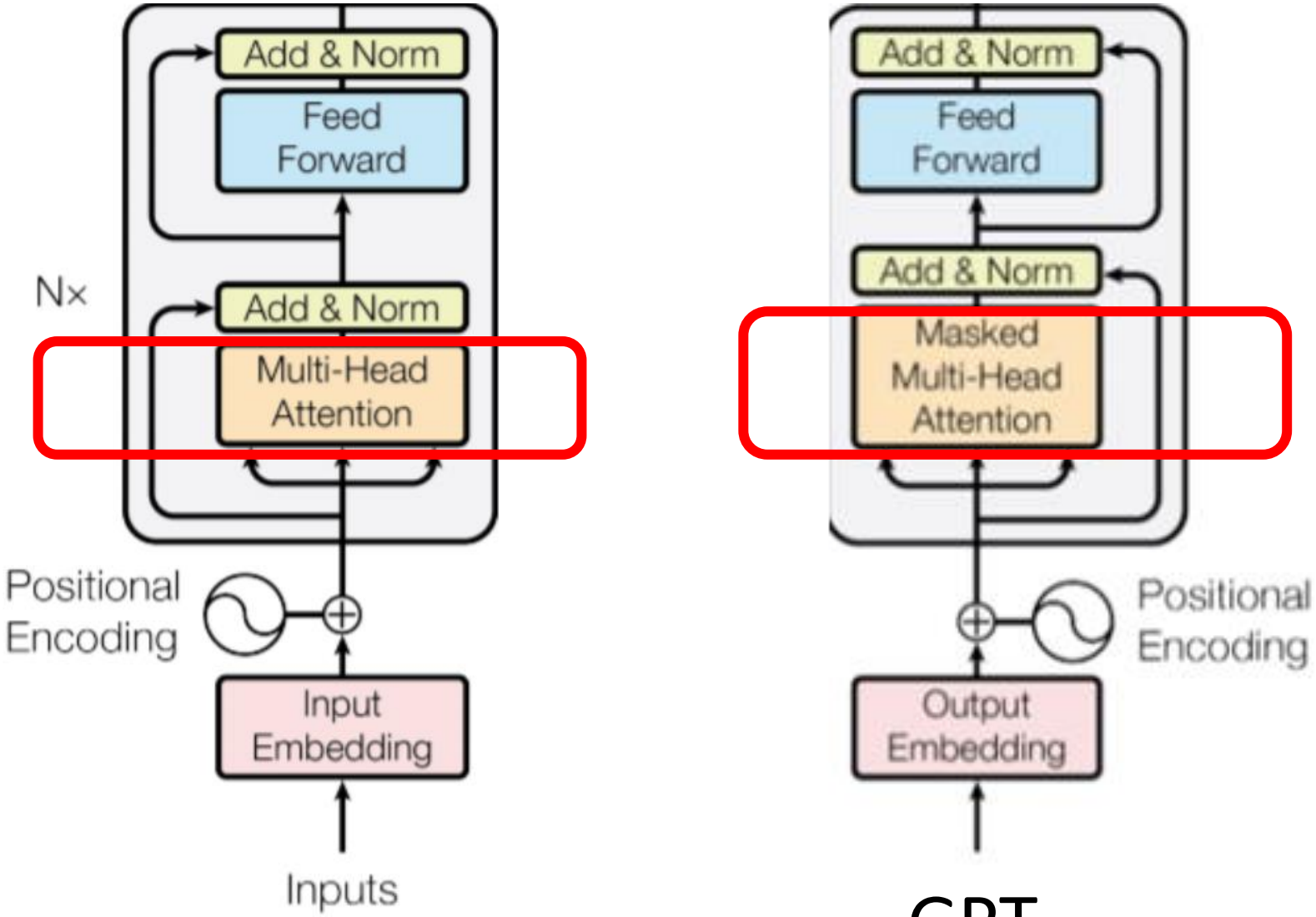



BERT




GPT

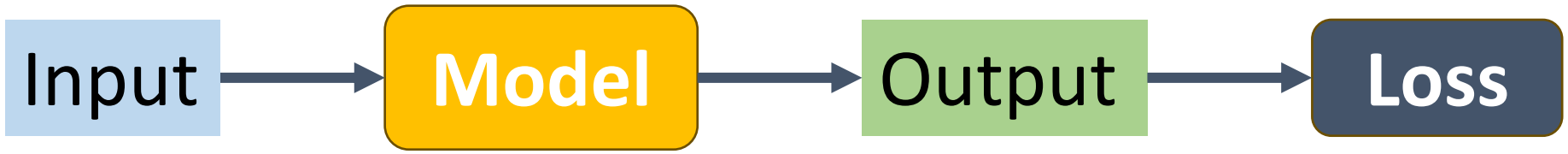
# BERT vs. GPT-2


Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Input Embedding
Inputs
BERT
Add & Norm
Feed Forward
Add & Norm
Masked Multi-Head Attention
Positional Encoding
Output Embedding
GPT

# BERT vs. GPT-2

For Architecture

Self-Attention vs. Masked Self-Attention



# BERT: Bidirectional Encoder Representations from Transformers

# BERT

## Bidirectional Encoder Representations from Transformers

*[Devlin & al., 2018]*

# BERT

## **Bidirectional** Encoder Representations from Transformers

Look at both the preceding (left) and following (right) context simultaneously when processing each token.

Capture the entire context.

*[Devlin & al., 2018]*

# BERT

## Bidirectional **Encoder** Representations **from Transformers**

Use Transformer Encoder.

*[Devlin & al., 2018]*

# BERT

## Bidirectional Encoder **Representations** from Transformers

Pre-training Deep Transformers Encoder
for Representation.

*[Devlin & al., 2018]*

# BERT

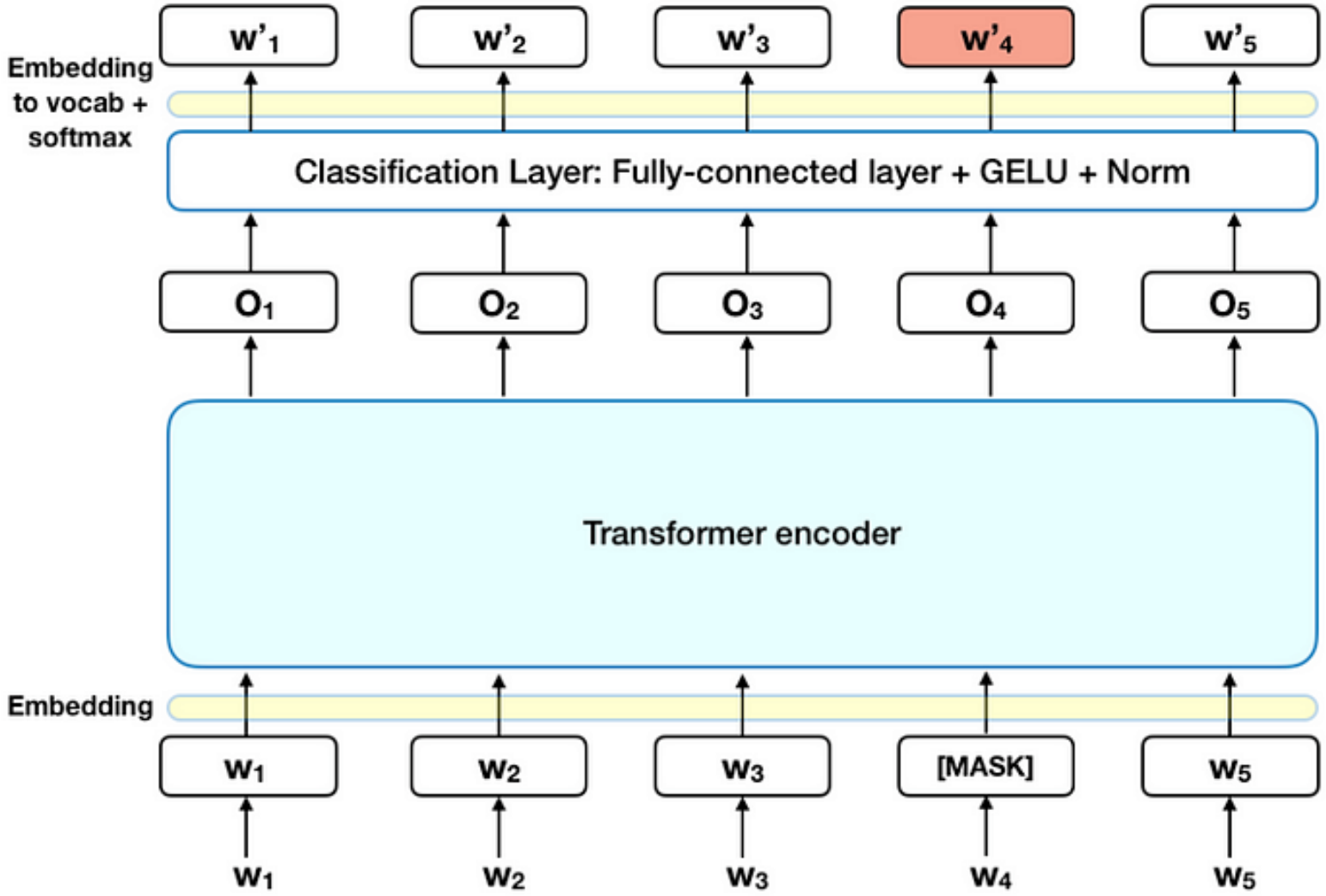
Pre-training Deep Transformers Encoder for Representation.

**How to?**

By Masked Language Modeling:
predict masked tokens of input text.

*[Devlin & al., 2018]*

# BERT




*[Devlin & al., 2018]*

# BERT: Word Masking

- Mask out ***15%*** of the input words, and then predict the masked words.

store gallon

↑ ↑

the man went to the [MASK] to buy a [MASK] of milk

- Too little masking: Too expensive to train
- Too much masking: No enough context

# BERT: 15% Masked Tokens

- 80% [MASK]

  *the man went to the **[MASK]** to buy a **[MASK]** of milk*

  - Model learns to correctly predict [MASK] words -> *store* and *gallon*.
  - If the model assigns high probability to *store* and *gallon* -> Low loss.

# BERT: 15% Masked Tokens

- 10% random word

  *the man went to the **apple** to buy a **[MASK]** of milk*

  - Model still has to correctly predict the word ***store*** word in the place of ***apple.***
  - Model is forced to learn from the context words *the man went to the to buy a **[MASK]** of milk* and not be fooled by ***apple.***

**Noise Augmentation for Robustness.**

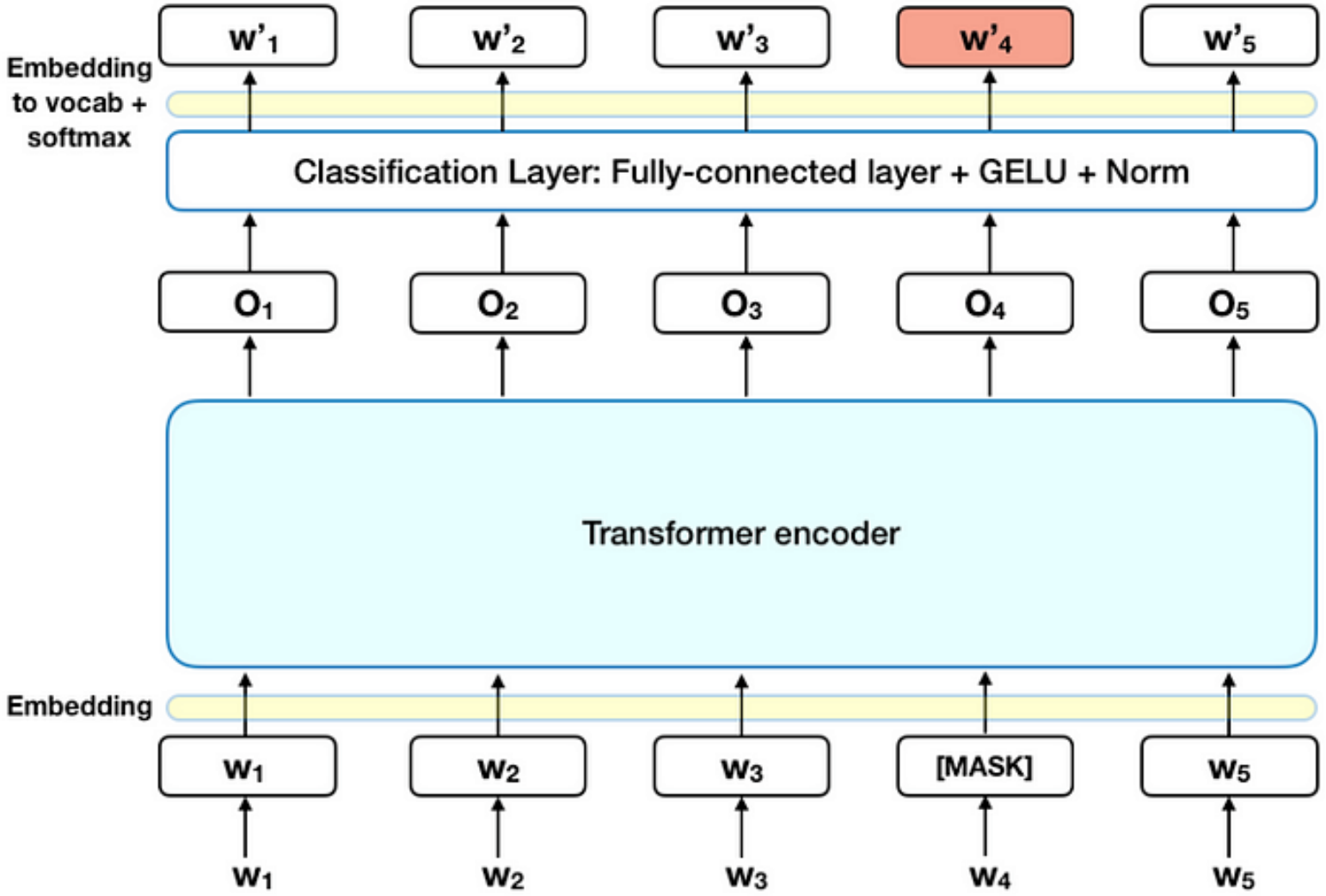
# BERT: 15% Masked Tokens

- 10% original word

  *the man went to the **store** to buy a **[MASK]** of milk*

  - Model has to correctly predict the word ***store*** in the place of ***store***.
  - It helps the model to adjust the embedding of word ***store*** when the model "knows" that the word is ***store.***

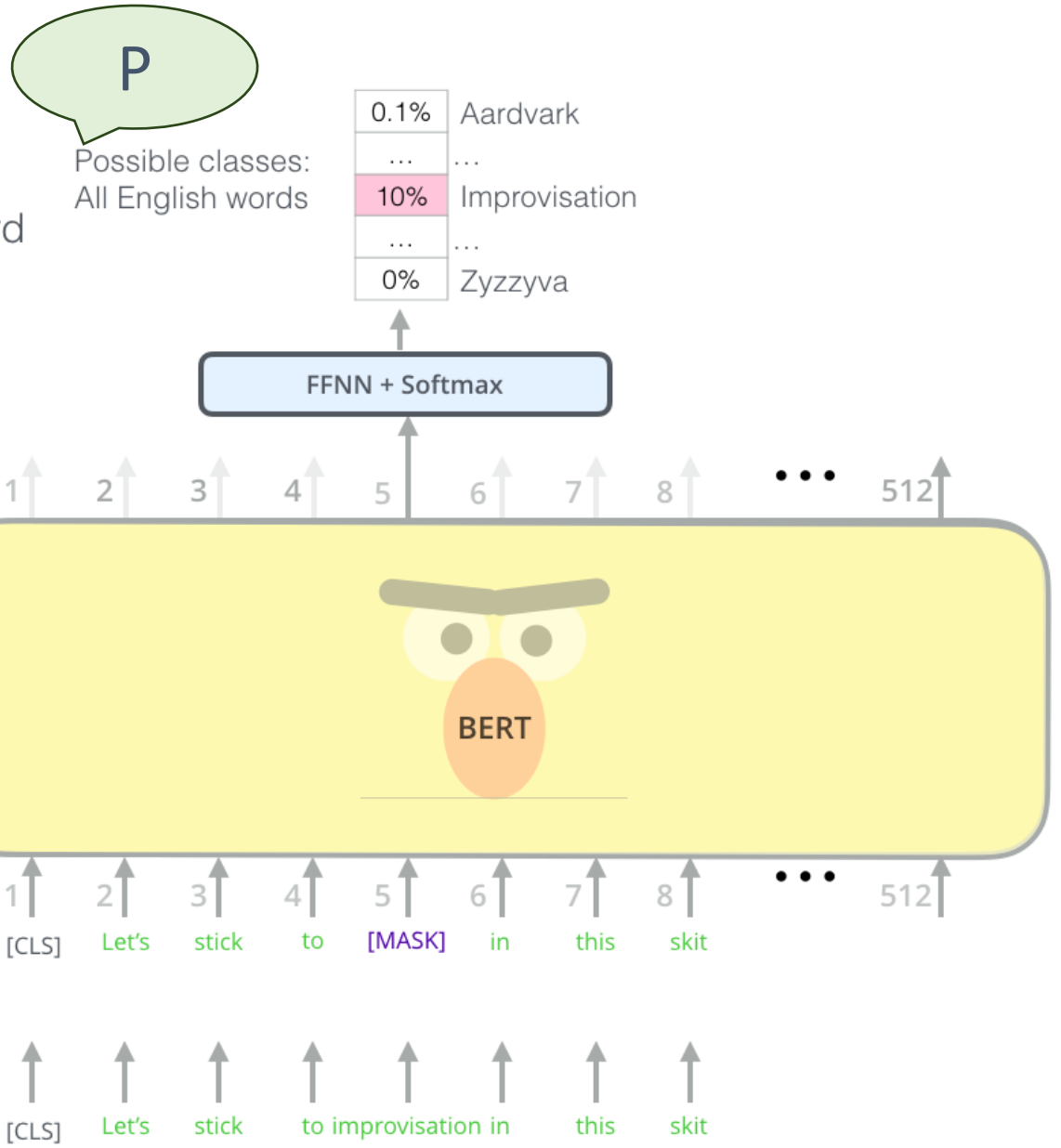
# BERT: Predict Masked Token




*[Devlin & al., 2018]*

# BERT: MLM

Use the output of the masked word's position to predict the masked word




Randomly mask 15% of tokens

Input

# BERT: MLM

P: probability vector of shape [N, 1],
e.g., *[0.6, 0.12, 0.18, 0.1]*
T: gold label one-hot vector of shape [N, 1]
e.g., *[0, 0, 1, 0]*

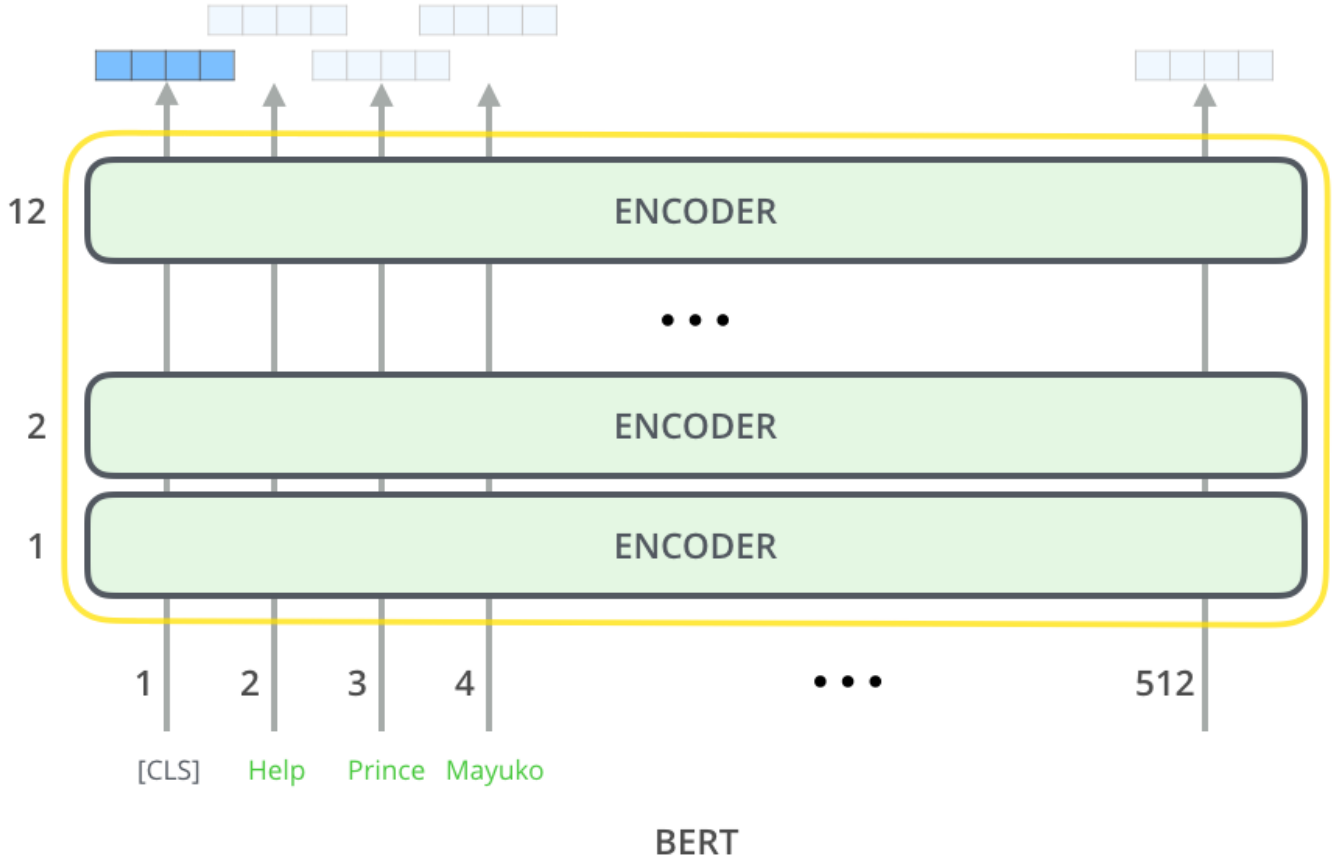
N: vocabulary size, here N = 4
For each masked token, cross-entropy loss:

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}\left[t_j \log(p_j) + (1 - t_j)\log(1 - p_j)\right]\right]$$

BERT vocabulary size: 30,000

Maximum number of input tokens: 512

Input and output vector size: 768, and 1024



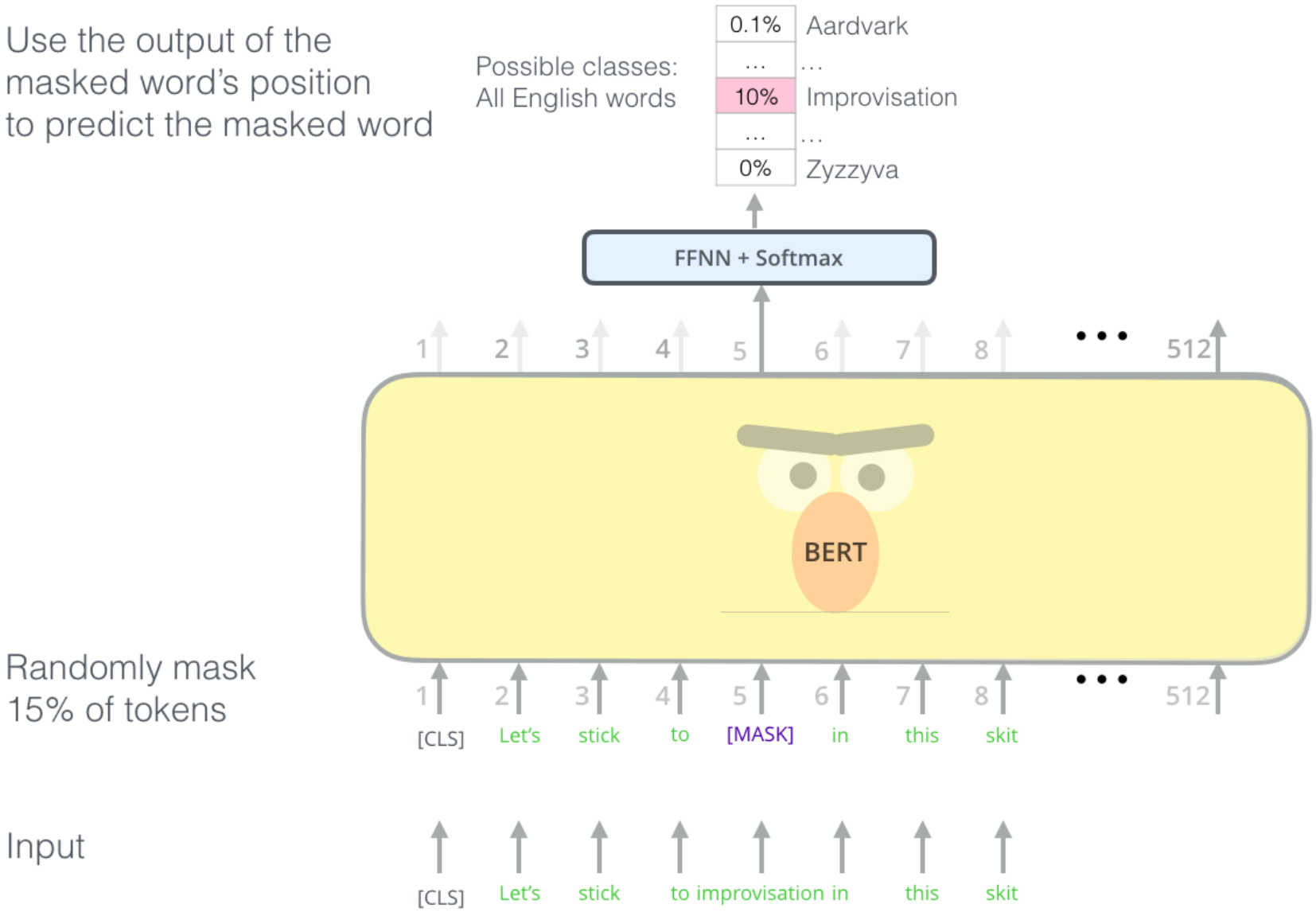
# BERT vs. GPT-2

- Input:
  - BERT: *the man went to the **[MASK]** to buy a **[MASK]** of milk*
  - GPT-2: *the man went to the store to buy a gallon of milk*

- Output:
  - BERT: $P(x_{[MASK]} \mid X)$
  - GPT-2: $P(y_i \mid y_{<i})$

- Loss (Objective function):
  - BERT: Masked language modeling + Next sentence prediction
  - GPT-2: Auto-regressive language modeling

# BERT Pretraining Task 1: Masked Words




Out of this 15%,
80% are [Mask],
10% random words
10% original words

# BERT Pretraining Task 2: Sentence Order

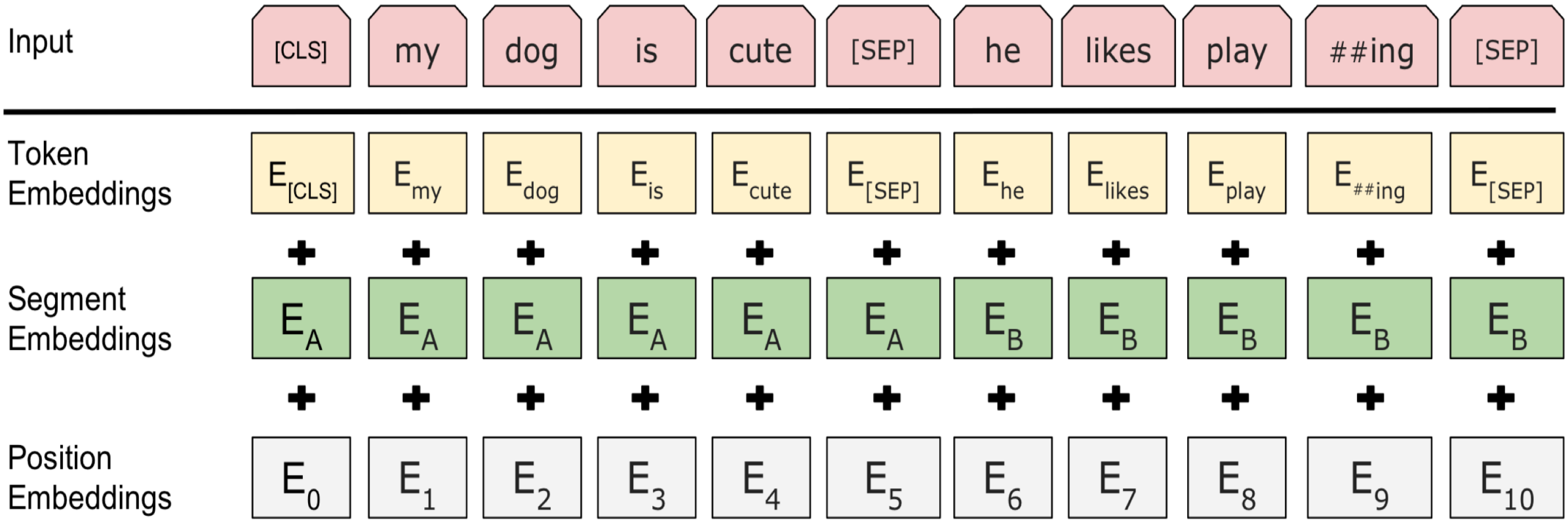
- To learn the *relationship* between sentences, predict whether Sentence B is the actual sentence that precedes Sentence A, or it is a random sentence

**Sentence A =** The man went to the store.
**Sentence B =** He bought a gallon of milk.
**Label =** IsNextSentence

**Sentence A =** The man went to the store.
**Sentence B =** Penguins are flightless.
**Label =** NotNextSentence

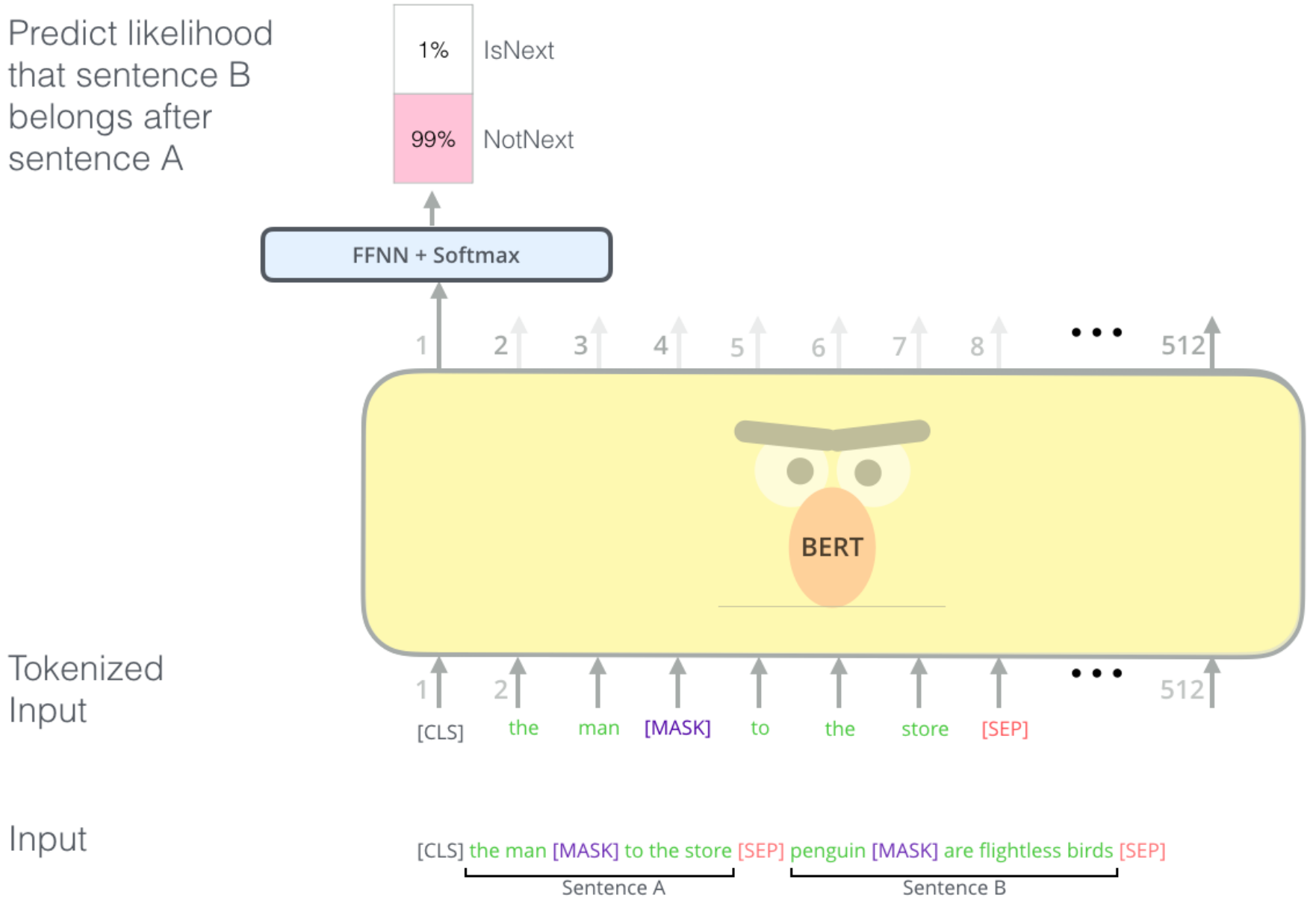
50% true second sentences
50% random second sentences

# BERT Pretraining Task 2: Sentence Order




- Token embeddings are word pieces
- Learned segmented embedding represents each sentence
- Positional embeddings are as for other Transformer architectures

# BERT Pretraining Task 2: Sentence Order



# BERT Pretraining Loss

MLM:

$$\sum_{all\ masked\ tokens} \text{Cross\_entropy}(P(x_{[MASK]} \mid X), T) +$$

NSP:

$$\text{Cross\_entropy}(P(x_{[CLS]} \mid X), G)$$

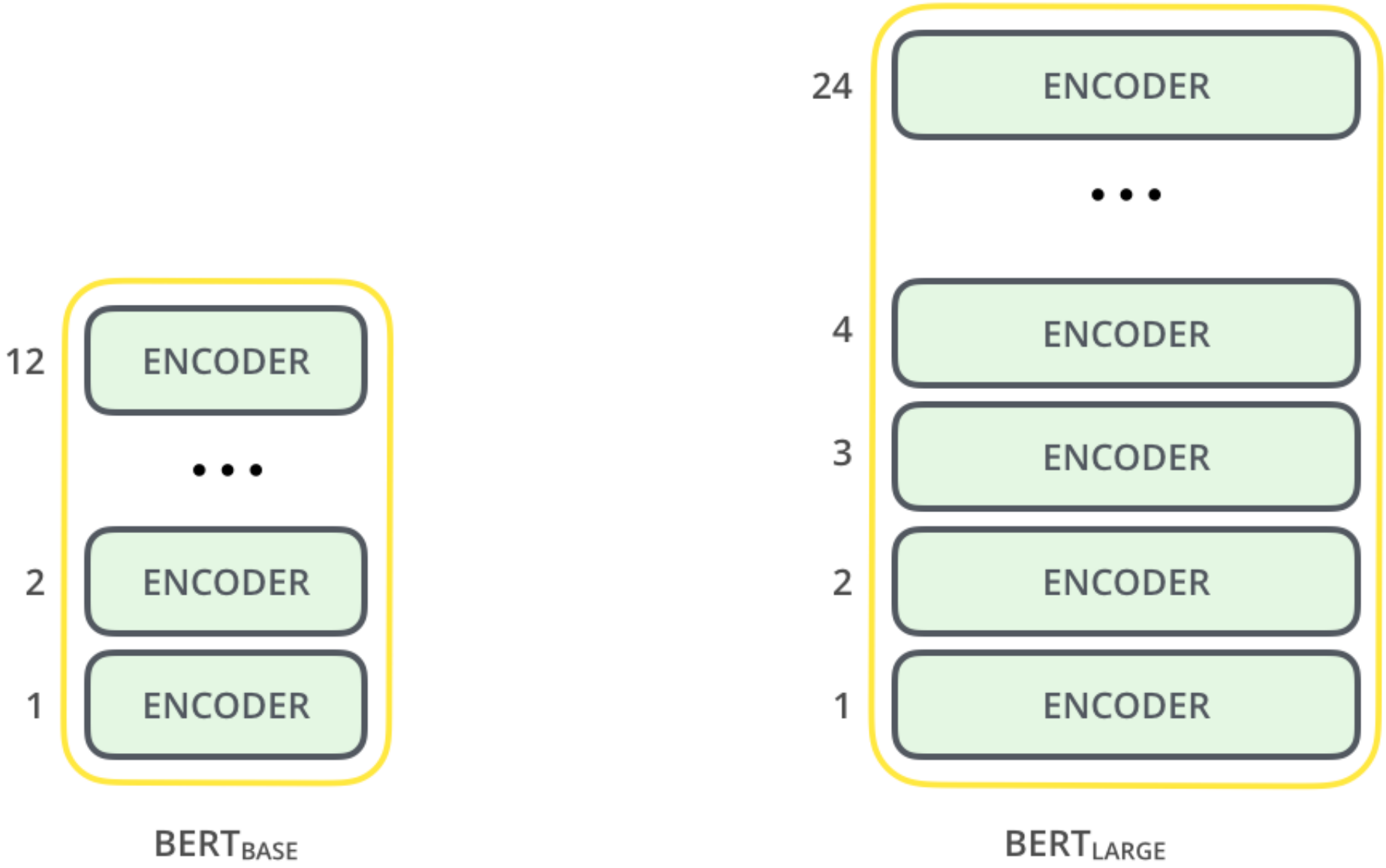
*G: gold label of next sentence prediction [0,1] or [1,0]*
*T: gold token over vocabulary*

# BERT Model Architecture and Training

- Transformer encoder (as before)
- Self-attention ⇒ no locality bias
  - Long-distance context has “equal opportunity”
- Single multiplication per layer ⇒ efficiency on GPU/TPU

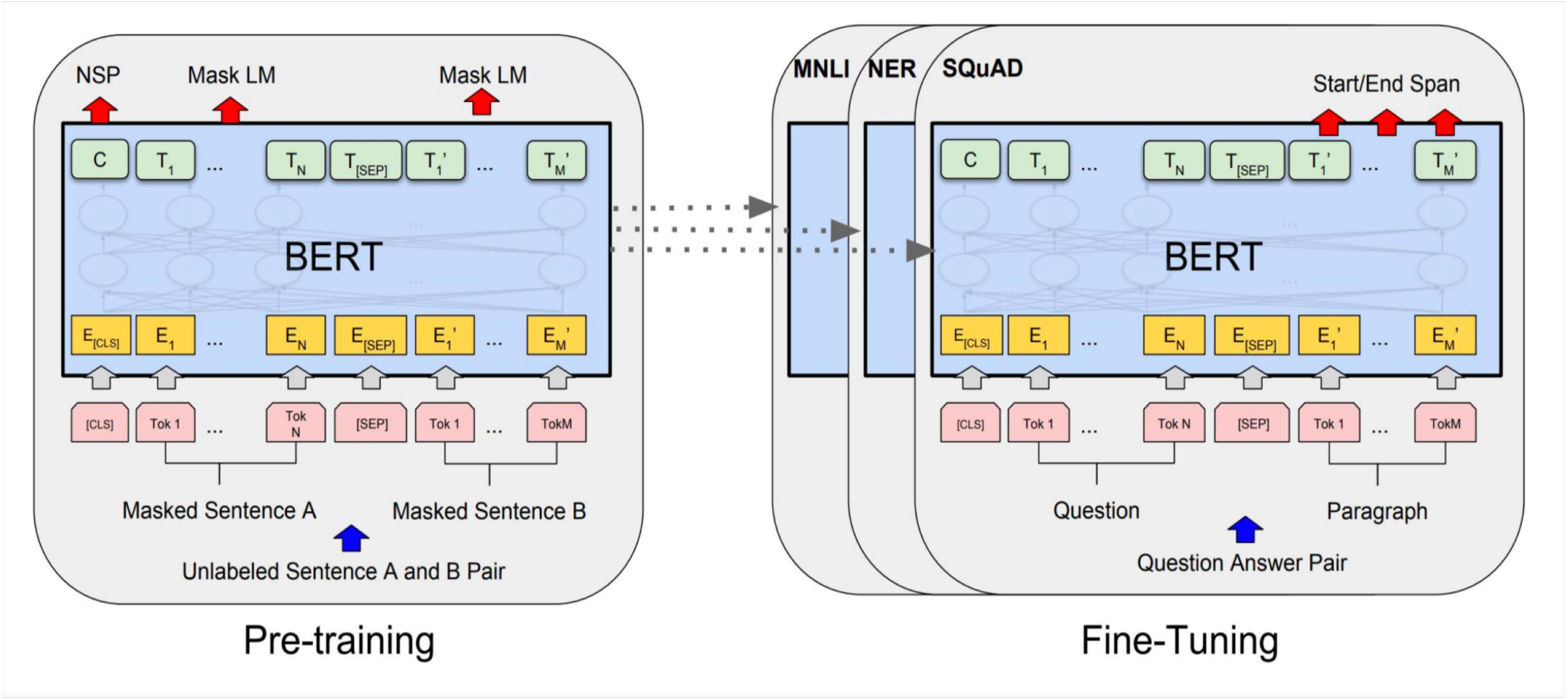
- Trained on Wikipedia + BookCorpus
- 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden, 12-head
  - BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

# BERT-base vs. BERT-large



# Fine-Tuning BERT

Simply train a classifier built on the top layer for each task that we fine tune for



# NLP Tasks: Multi-Genre Natural Language Inference

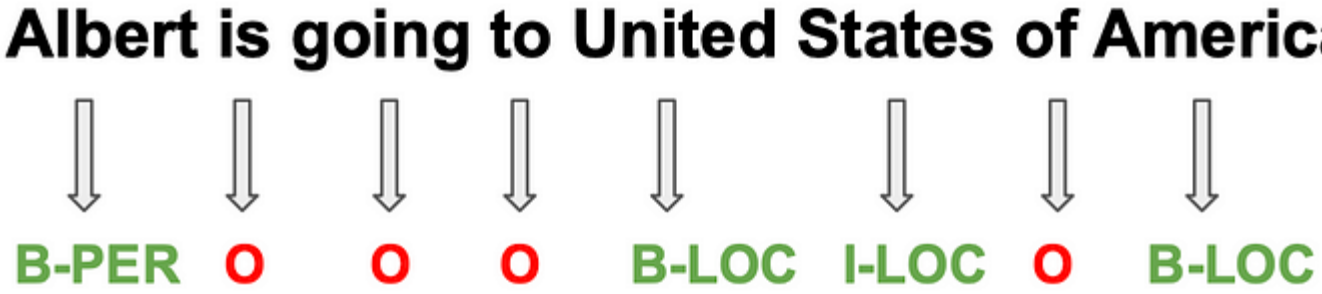
MNLI: 433k pairs of examples, labeled as *entailment*, *neutral*, or *contraction*

The Multi-Genre Natural Language Inference (MultiNLI) corpus is a crowd-sourced collection of 433k sentence pairs annotated with textual entailment information. The corpus is modeled on the SNLI corpus, but differs in that covers a range of genres of spoken and written text, and supports a distinctive cross-genre generalization evaluation. The corpus served as the basis for the shared task of the RepEval 2017 Workshop at EMNLP in Copenhagen.

**Examples**

| Premise | Label | Hypothesis |
|---|---|---|
| ***Fiction*** | | |
| The Old One always comforted Ca'daan, except today. | *neutral* | Ca'daan knew the Old One very well. |
| ***Letters*** | | |
| Your gift is appreciated by each and every student who will benefit from your generosity. | *neutral* | Hundreds of students will benefit from your generosity. |
| ***Telephone Speech*** | | |
| yes now you know if if everybody like in August when everybody's on vacation or something we can dress a little more casual or | *contradiction* | August is a black out month for vacations in the company. |
| ***9/11 Report*** | | |
| At the other end of Pennsylvania Avenue, people began to line up for a White House tour. | *entailment* | People formed a line at the end of Pennsylvania Avenue. |

# NLP Tasks: Named Entity Recognition (NER)




BIO tagging:

O: "Outside" - token is not part of any named entity.

B: "Begin" - the beginning of a named entity

I: "Inside" - tokens are part of a multi-token entity but are not the first token.

# NLP Tasks (SQuAD -- Stanford Question Answering Dataset)

Sample: Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.

Which NFL team represented the AFC at Super Bowl 50?

Ground Truth Answers: Denver Broncos

Which NFL team represented the NFC at Super Bowl 50?

Ground Truth Answers: Carolina Panthers

# GLUE Tasks

- In addition to QA, NER and NLI, the GLUE benchmark also has sentiment analysis and linguistic acceptableness tasks.

- **SST**
  - **Sentence:** *This is a so exciting movie.* **Label:** *Positive*
  - **Sentence:** *Her grandma is not healthy recently.* **Label:** *Negative*

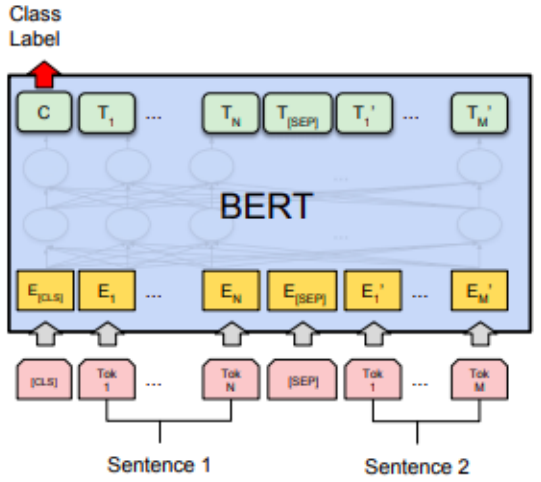
- **CoLa**
  - **Sentence:** *The wagon rumbled down the road.* **Label:** *Acceptable*
  - **Sentence:** *The car honked down the road.* **Label:** *Unacceptable*

# GLUE Tasks

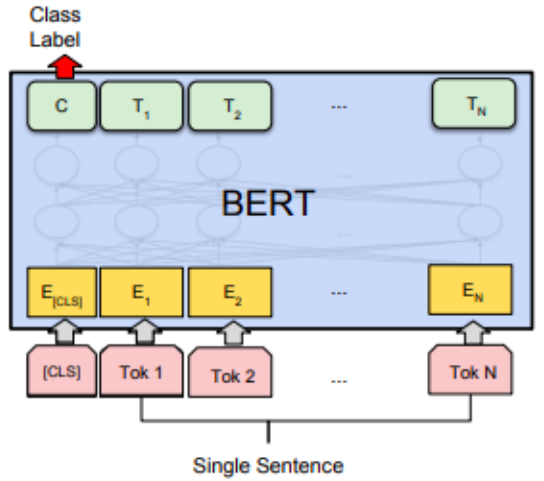
- Semantic textual similarity (STS) and natural language inference (NLI) are all sentence-relationship prediction task, with two sentences as input.

- **NLI**
  - **Premise:** *Hills and mountains are especially sanctified in Jainism.*
  - **Hypothesis:** *Jainism hates nature.*
  - **Label:** *Contradiction*
- **STS**
  - **Sentence1:** *The man is playing football.*
  - **Sentence2:** *The man is playing the piano.*
  - **Label:** *2*

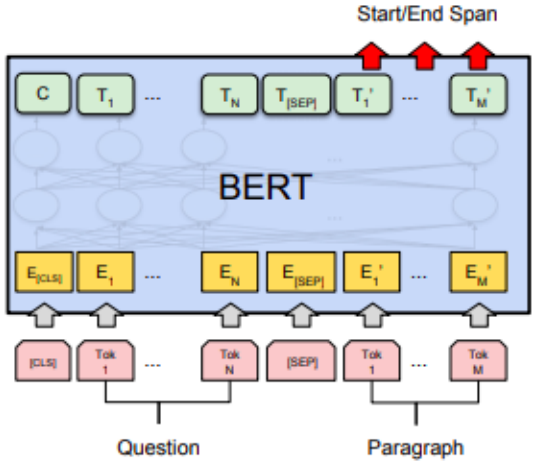
# Fine-Tuning BERT for Specific Tasks



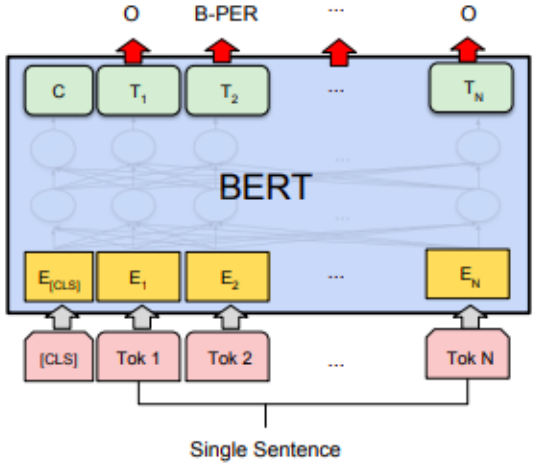

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG




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




(c) Question Answering Tasks: SQuAD v1.1




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

SST (Stanford sentiment treebank): 215k phrases with fine-grained sentiment labels in the parse trees of 11k sentences.

How do you fine-tune BERT for Assignment 2 tasks?

# CoNLL-2003 Named Entity Recognition

| Name | Description | Year | F1 |
|---|---|---|---|
| Flair (Zalando) | Character-level language model | 2018 | 93.09 |
| BERT Large | Transformer bidi LM + fine tune | 2018 | 92.8 |
| CVT Clark | Cross-view training + multitask learn | 2018 | 92.61 |
| BERT Base | Transformer bidi LM + fine tune | 2018 | 92.4 |
| ELMo | ELMo in BiLSTM | 2018 | 92.22 |
| TagLM Peters | LSTM BiLM in BiLSTM tagger | 2017 | 91.93 |
| Ma + Hovy | BiLSTM + char CNN + CRF layer | 2016 | 91.21 |
| Tagger Peters | BiLSTM + char CNN + CRF layer | 2017 | 90.87 |
| Ratinov + Roth | Categorical CRF+Wikipeda+word cls | 2009 | 90.80 |
| Finkel et al. | Categorical feature CRF | 2005 | 86.86 |
| IBM Florian | Linear/softmax/TBL/HMM ensemble, gazettes++ | 2003 | 88.76 |
| 6S5tanford | MEMM softmax markov model | 2003 | 86.07 |

# SQuAD 1.1 Leaderboard When BERT Came

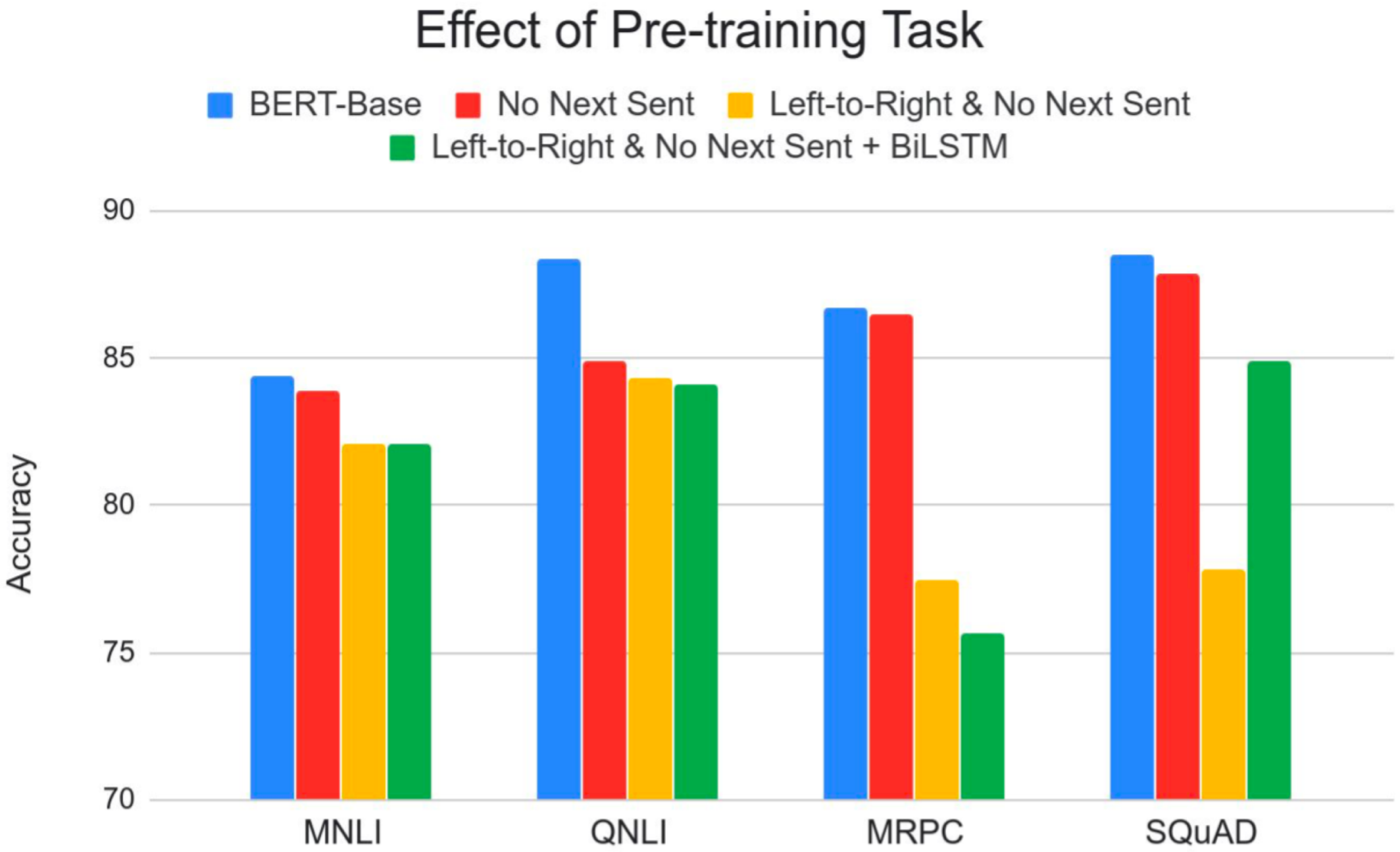
| Rank | Model | EM | F1 |
|---|---|---|---|
| | Human Performance<br>*Stanford University*<br>(Rajpurkar et al. '16) | 82.304 | 91.221 |
| 1<br>Oct 05, 2018 | BERT (ensemble)<br>*Google AI Language*<br>https://arxiv.org/abs/1810.04805 | **87.433** | **93.160** |
| 2<br>Oct 05, 2018 | BERT (single model)<br>*Google AI Language*<br>https://arxiv.org/abs/1810.04805 | 85.083 | 91.835 |
| 2<br>Sep 26, 2018 | nlnet (ensemble)<br>*Microsoft Research Asia* | 85.954 | 91.677 |
| 5<br>Sep 09, 2018 | nlnet (single model)<br>*Microsoft Research Asia* | 83.468 | 90.133 |
| 3<br>Jul 11, 2018 | QANet (ensemble)<br>*Google Brain & CMU* | 84.454 | 90.490 |

# SQuAD 2.0 Leaderboard When BERT Came

| Rank | Model | EM | F1 |
|---|---|---|---|
| | Human Performance<br>*Stanford University*<br>(Rajpurkar & Jia et al. '18) | 86.831 | 89.452 |
| 1<br>Jan 15, 2019 | BERT + MMFT + ADA (ensemble)<br>*Microsoft Research Asia* | **85.082** | **87.615** |
| 2<br>Jan 10, 2019 | BERT + Synthetic Self-Training (ensemble)<br>*Google AI Language*<br>https://github.com/google-research/bert | 84.292 | 86.967 |
| 3<br>Dec 13, 2018 | BERT finetune baseline (ensemble)<br>*Anonymous* | 83.536 | 86.096 |
| 4<br>Dec 16, 2018 | Lunet + Verifier + BERT (ensemble)<br>*Layer 6 AI NLP Team* | 83.469 | 86.043 |
| 4<br>Dec 21, 2018 | PAML+BERT (ensemble model)<br>*PINGAN GammaLab* | 83.457 | 86.122 |
| 5<br>Dec 15, 2018 | Lunet + Verifier + BERT (single model)<br>*Layer 6 AI NLP Team* | 82.995 | 86.035 |

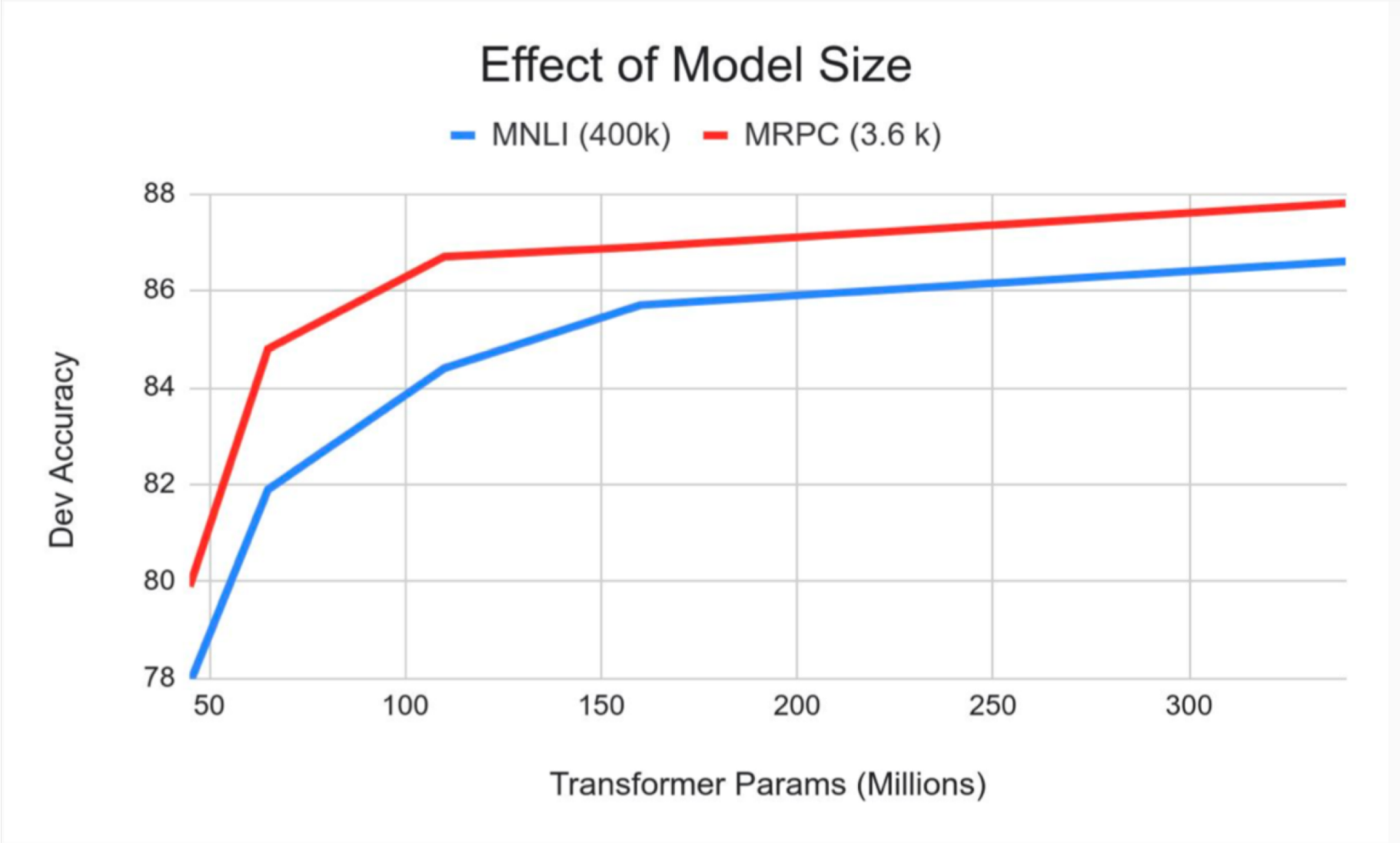
# BERT Results on GLUE Tasks: from the BERT Paper

| System | MNLI-(m/mm) 392k | QQP 363k | QNLI 108k | SST-2 67k | CoLA 8.5k | STS-B 5.7k | MRPC 3.5k | RTE 2.5k | **Average** - |
|---|---|---|---|---|---|---|---|---|---|
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.9 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 88.1 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.2 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.1 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **91.1** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **81.9** |

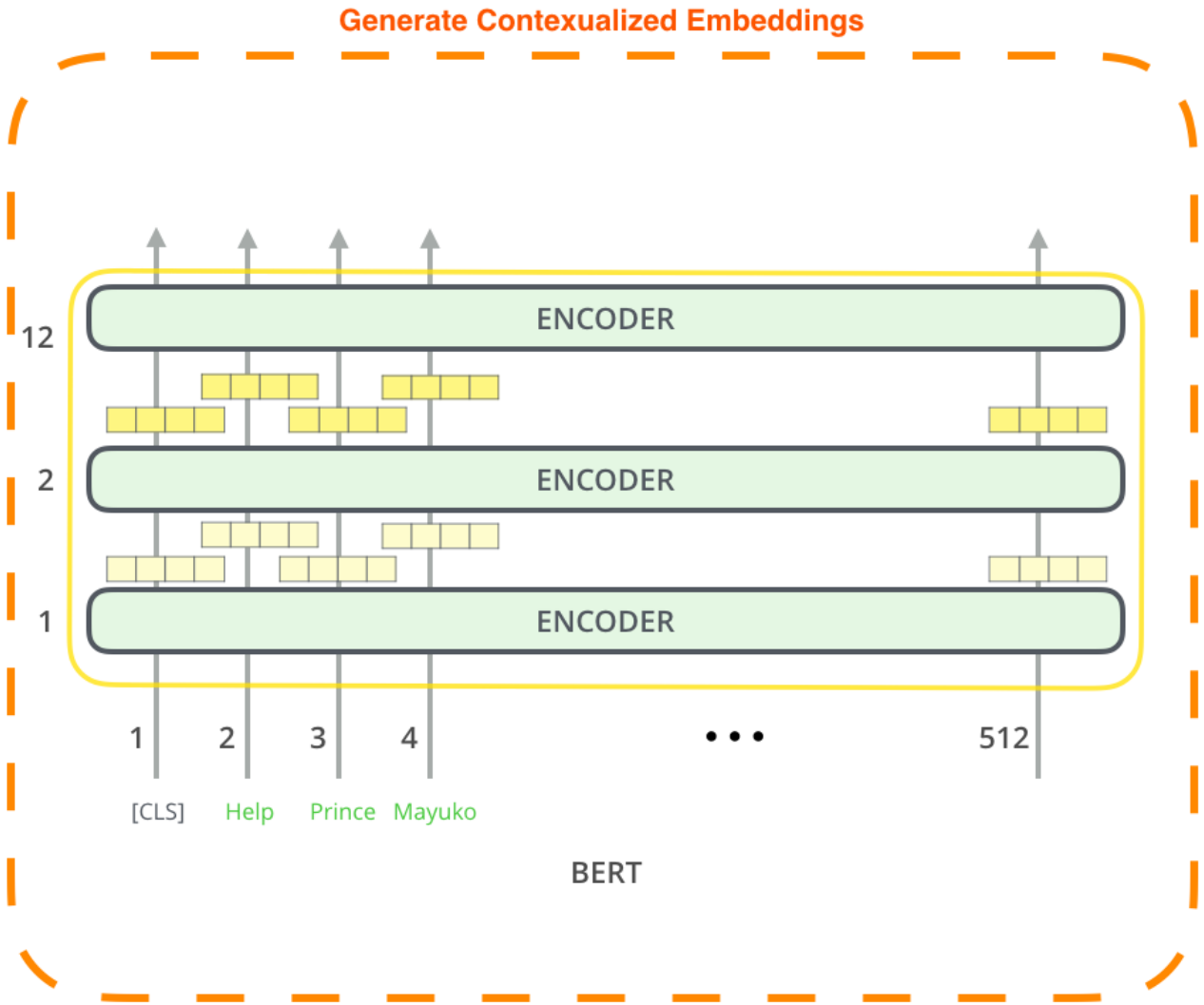

Effect of Pre-training Task
BERT-Base
No Next Sent
Left-to-Right & No Next Sent
Left-to-Right & No Next Sent + BiLSTM
Accuracy
90
85
80
75
70
MNLI
QNLI
MRPC
SQuAD

# BERT: Effect of Model Size

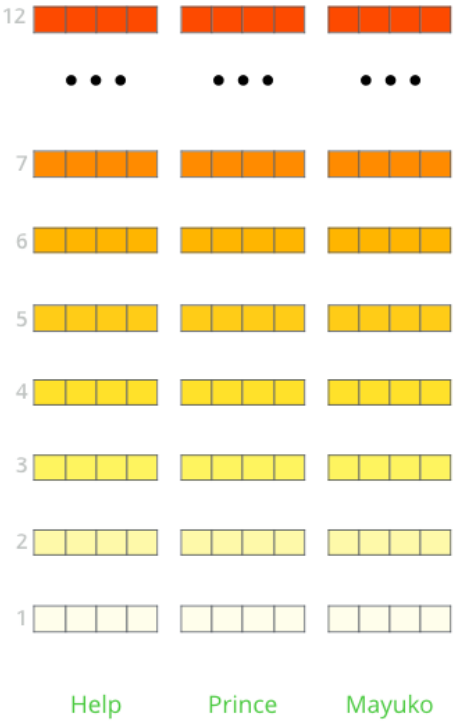
- Going from 110M to 340M parameters helps a lot



# Feature Extraction from BERT




The output of each encoder layer along each token's path can be used as a feature representing that token.



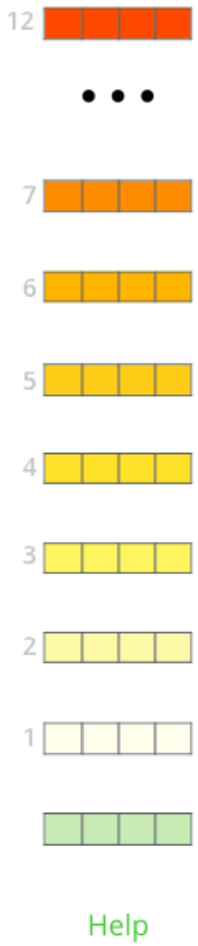

We end up with some embedding for each word related to the current input

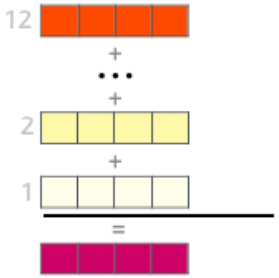
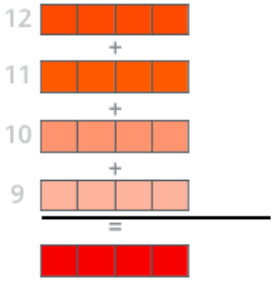
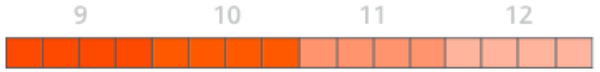
We start with independent word embedding at first level

**But which one should we use?**

# Feature Extraction from BERT: Which Embeddings to Use?

**What is the best contextualized embedding for "Help" in that context?**
For named-entity recognition task CoNLL-2003 NER




| | | Dev F1 Score |
|---|---|---|
| First Layer |   | 91.0 |
| Last Hidden Layer |   | 94.9 |
| Sum All 12 Layers |   | 95.5 |
| Second-to-Last Hidden Layer |   | 95.6 |
| Sum Last Four Hidden |   | 95.9 |
| Concat Last Four Hidden |   | 96.1 |

# Some Observations

- Model size matters (345 million parameters are better than 110 million parameters)
- With enough training data, more training steps yield higher accuracy
- BERT's bidirectional approach converges slower than left-to-right training, but yields better accuracy after a small number of pre-training steps.

# Question

- Why does BERT converge slower than GPT-2 in pre-training, but yields better accuracy after a small number of pre-training steps?

# Drawbacks and Variants of Transformers

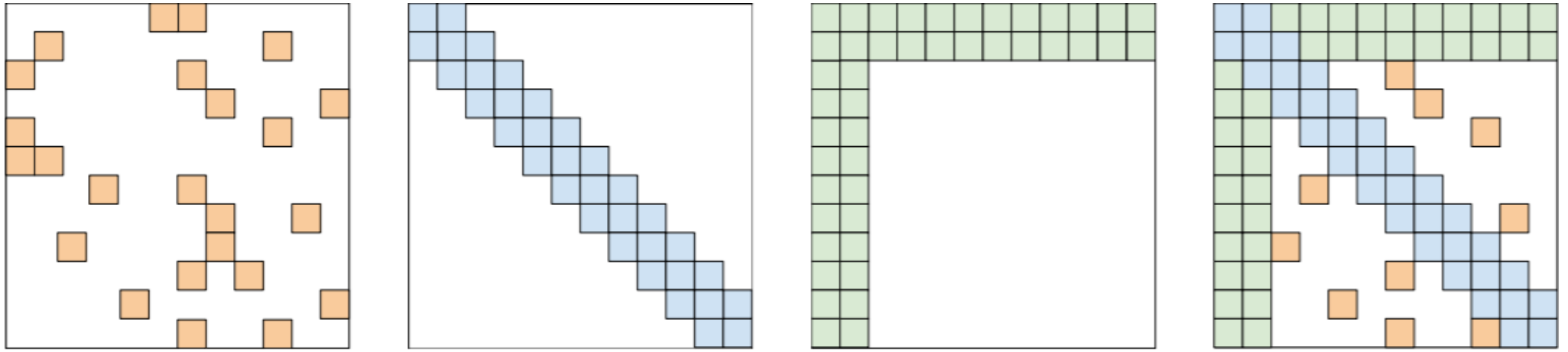
# What would we like to fix about the Transformer?

- **Quadratic compute in self-attention**:
  - Computing all pairs of interactions means our computation grows **quadratically** with the sequence length!
  - For recurrent models, it only grew linearly!
- **Position representations**:
  - Are simple absolute indices the best we can do to represent position?
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

# Recent work on improving on quadratic self-attention cost

- Considerable recent work has gone into the question, *Can we build models like Transformers without paying the $O(T^2)$ all-pairs self-attention cost?*
- For example, **BigBird** [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

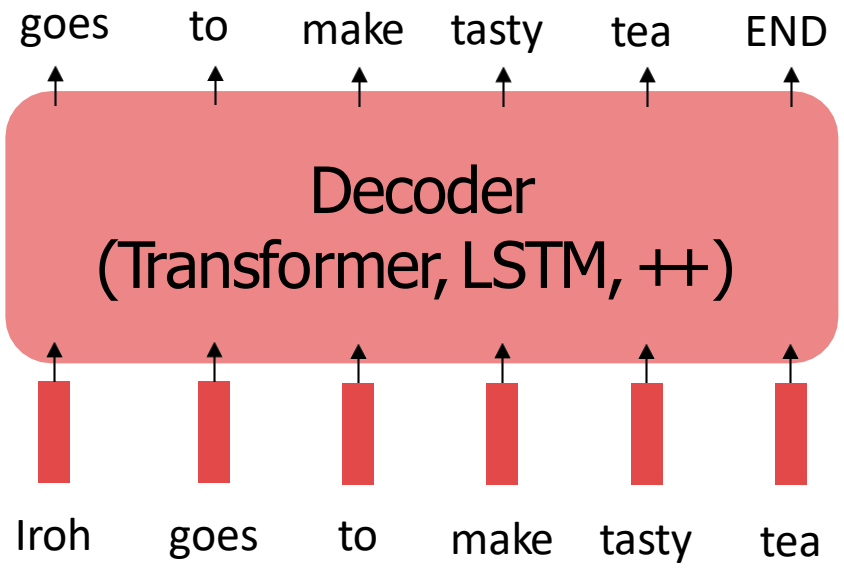
# “Pre-Train, Fine-Tune” Paradigm Revisited

# The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

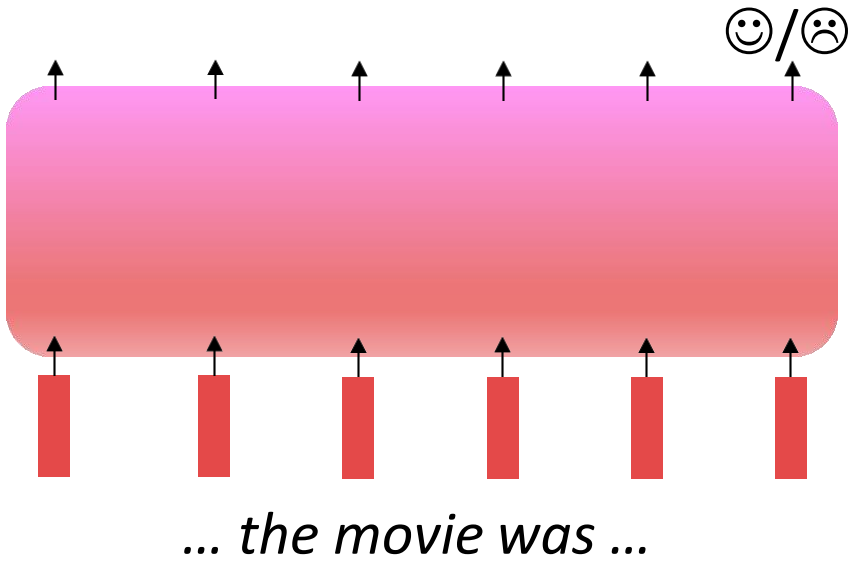
**Step 1: Pretrain (on language modeling)**

Lots of text; learn general things!

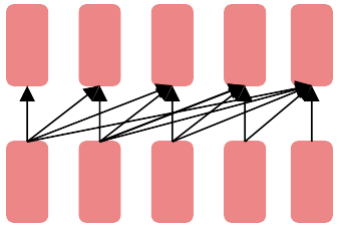



**Step 2: Finetune (on your task)**

Not many labels; adapt to the task!

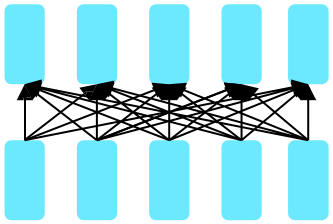


# Pretraining for Three Types of Architectures



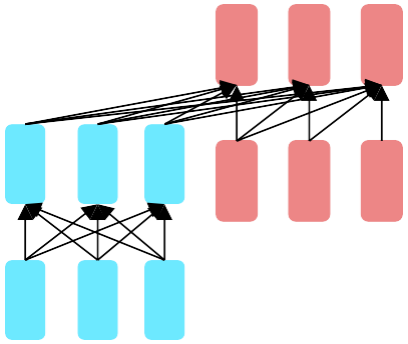
**Decoders**

- Language models! What we've seen so far.
- **Examples:** GPT-2, GPT-3, LaMDA



**Encoders**

- Gets bidirectional context – can condition on future!
- **Examples:** BERT and its many variants, e.g., RoBERTa



**Encoder-Decoders**

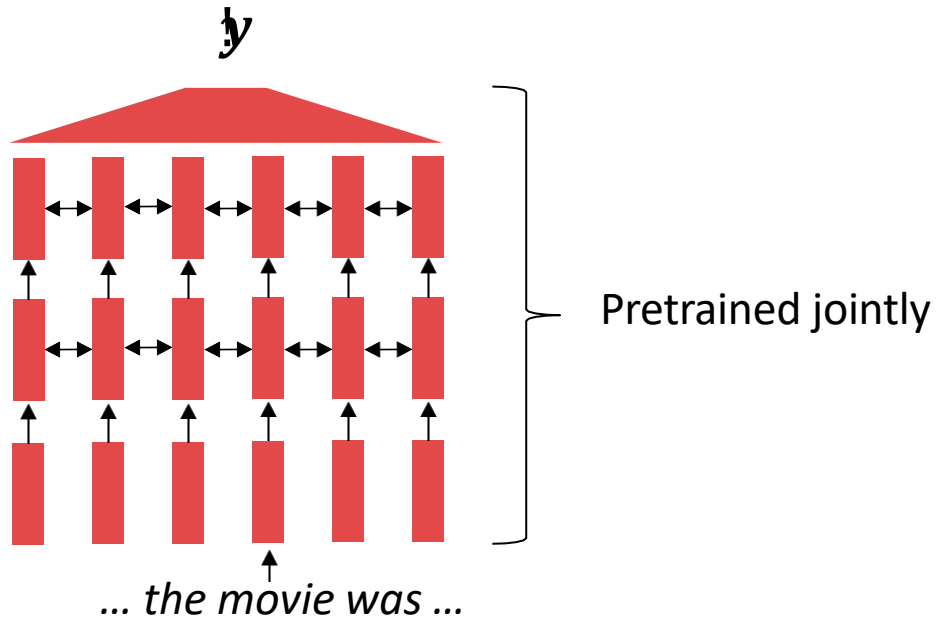
- Good parts of decoders and encoders?
- **Examples:** Transformer, T5, T0, FLAN-T5

# Where We Are Going: Pretraining Whole Models

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and then train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **probability distributions** over language that we can sample from




[This model has learned how to represent entire sentences through pretraining]