Language Modeling III: Transformers

CSE 5525: Foundations of Speech and Language Processing

https://shocheen.github.io/cse-5525-spring-2025/

THE OHIO STATE UNIVERSITY

Sachin Kumar (kumar.1145@osu.edu)

Slide Credits: Chris Tanner

Logistics

- Homework 2 is due date in exactly one week.
  - Any thoughts, questions, concerns?

- Final project: have you formed teams already?
  - A project proposal will be due second/third week of February.
  - We will post sample project ideas on the website/teams later this week

Recap

- Feedforward Neural Language Model
    - Need to make unreasonable assumptions and lose information from the long context
- Recurrent Neural Network (RNN)
    - Infinitely long context in theory --- hard to train (exploding/vanishing gradients), difficult to parallelize, and could be infeasible (memorize a variable length sequence in a fixed length vector).
    - Encoder-decoder architecture
- RNN + Attention
    - Solves the last issue, still hard to train efficiently (on GPUs).
- Attention is all you need [will continue today]
    - Transformer Architecture

Transformers

- Replace the linear part of RNNs with **self-attention**

- Introduce **residual connections + layernorm** to improve gradient flow (avoid gradient vanishing issues)

- Introduce **positional embeddings** to encode sequential order

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- Language Modeling With Transformers

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Defining Self-Attention

- **Terminology:**
  - **Query**: to match others
  - **Key**: to be matched
  - **Value**: information to be extracted

- **Definition:** Given a set of vector **values**, and a vector **query**, *attention* is a technique to compute a weighted sum of the **value**, dependent on the **query**.

[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

Self-Attention

Step 1: Our Self-Attention Head I has just 3 weight matrices W_q, W_k, W_v in total. These same 3 weight matrices are multiplied by each x_i to create all vectors:

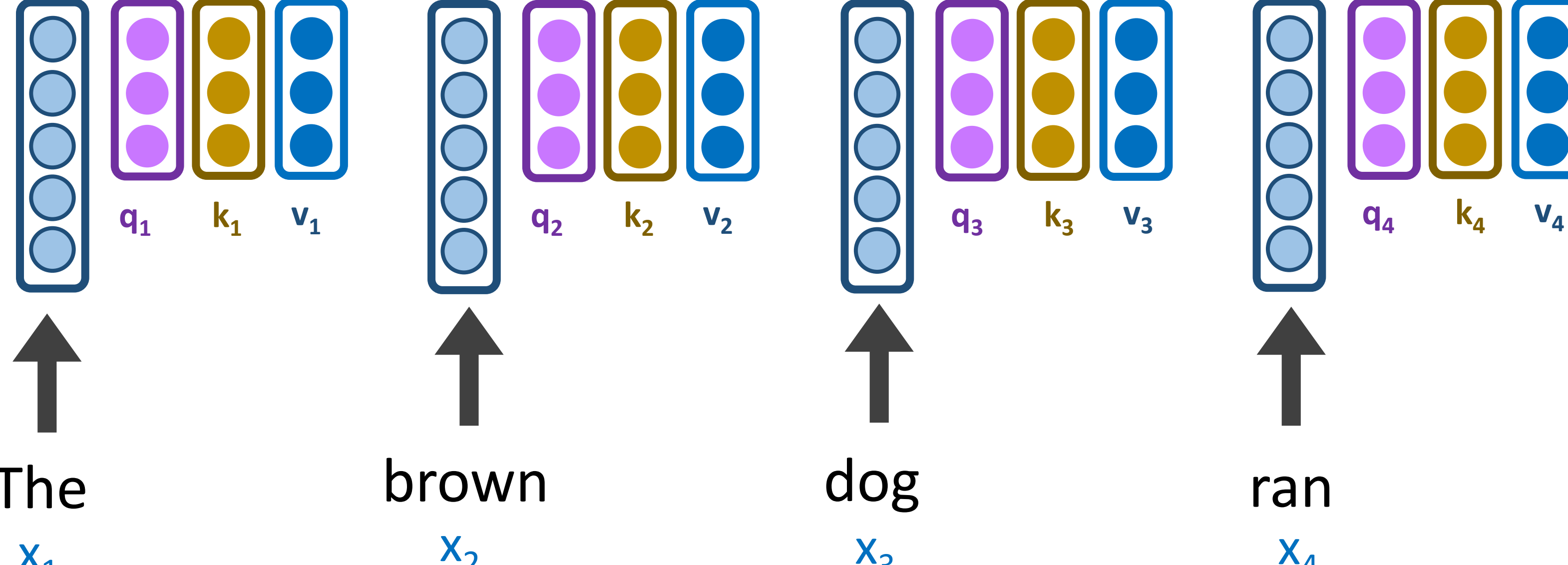
$$q_i = w_q x_i$$

$$k_i = w_k x_i$$

$$v_i = w_v x_i$$

Under the hood, each x_i has 3 small, associated vectors. For example, x_1 has:

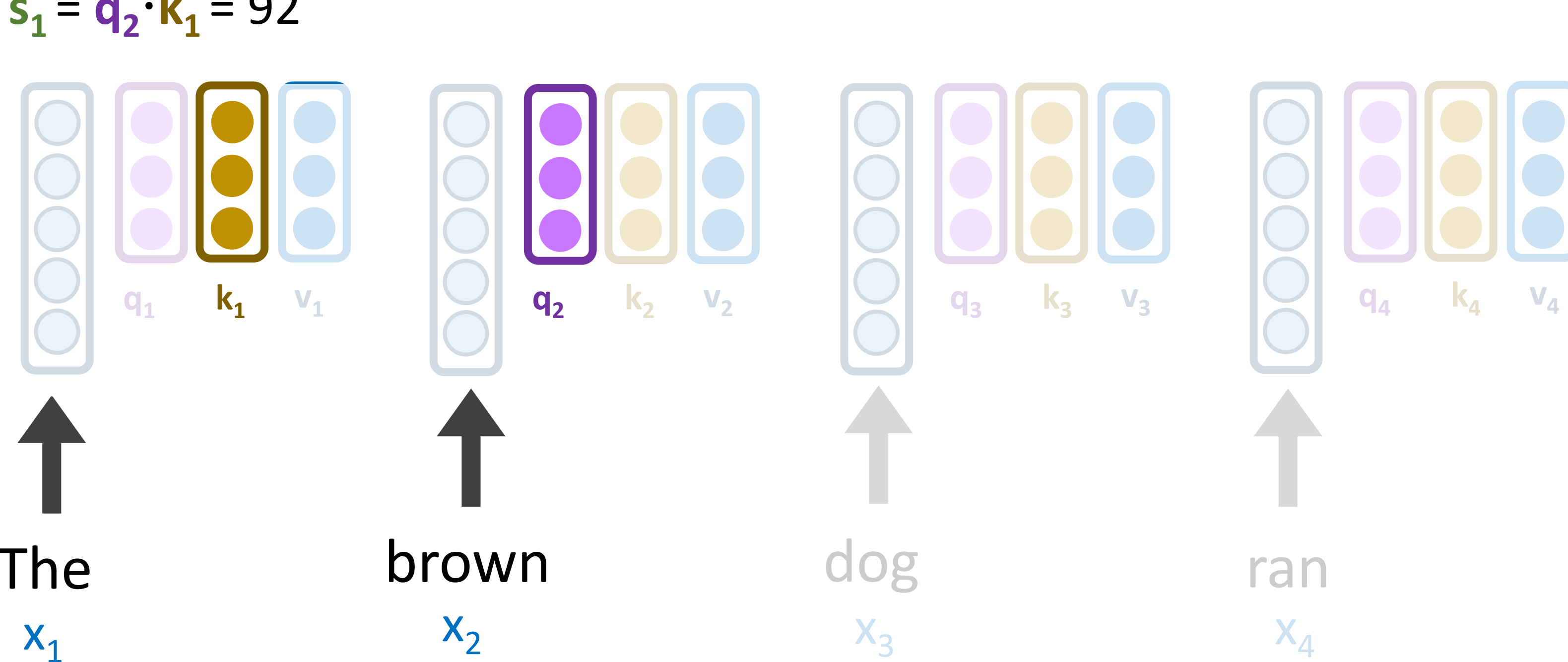
- Query $\mathbf{q_1}$
- Key $\mathbf{k_1}$
- Value $\mathbf{v_1}$



Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v}_i$

$s_1 = q_2 \cdot k_1 = 92$



Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v_i}$

$s_2 = \mathbf{q_2} \cdot \mathbf{k_2} = 124$

$s_1 = \mathbf{q_2} \cdot \mathbf{k_1} = 92$

Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" $\mathbf{v}_i$

$s_3 = \mathbf{q}_2 \cdot \mathbf{k}_3 = 22$

$s_2 = \mathbf{q}_2 \cdot \mathbf{k}_2 = 124$

$s_1 = \mathbf{q}_2 \cdot \mathbf{k}_1 = 92$

Self-Attention

Step 2: For word x_2, let's calculate the scores s_1, s_2, s_3, s_4, which represent how much attention to pay to each respective "word" v_i

$s_4 = q_2 \cdot k_4 = 8$

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$

q_1 k_1 v_1 q_2 k_2 v_2 q_3 k_3 v_3 q_4 k_4 v_4

The x_1 brown x_2 dog x_3 ran x_4

Self-Attention

Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_2 \cdot k_4 = 8$ $\quad a_4 = \sigma(s_4/8) = 0$

$s_3 = q_2 \cdot k_3 = 22$ $\quad a_3 = \sigma(s_3/8) = .01$

$s_2 = q_2 \cdot k_2 = 124$ $\quad a_2 = \sigma(s_2/8) = .91$

$s_1 = q_2 \cdot k_1 = 92$ $\quad a_1 = \sigma(s_1/8) = .08$

Self-Attention

Step 3: Our scores s_1, s_2, s_3, s_4 don't sum to 1. Let's divide by $\sqrt{len(k_i)}$ and softmax it

$s_4 = q_2 \cdot k_4 = 8$

$s_3 = q_2 \cdot k_3 = 22$

$s_2 = q_2 \cdot k_2 = 124$

$s_1 = q_2 \cdot k_1 = 92$

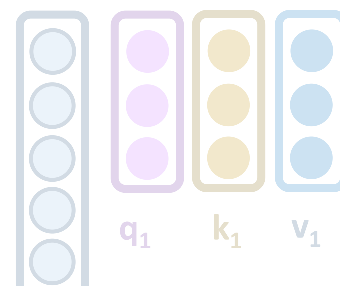
$a_4 = \sigma(s_4/8) = 0$

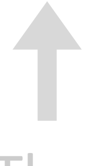
$a_3 = \sigma(s_3/8) = .01$

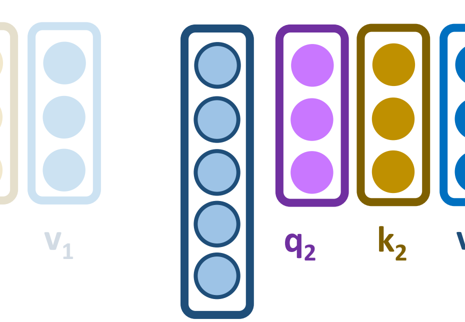
$a_2 = \sigma(s_2/8) = .91$

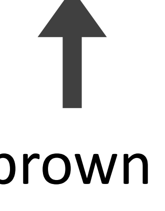
$a_1 = \sigma(s_1/8) = .08$

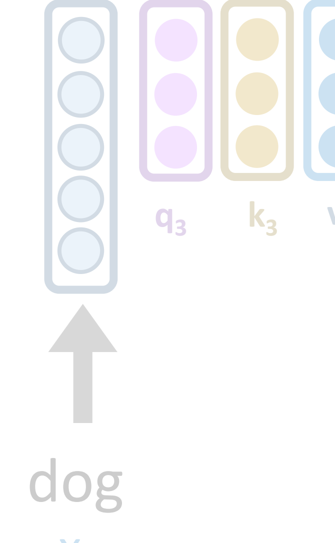
Instead of these a_i values directly weighting our original x_i word vectors, they directly weight our v_i vectors.

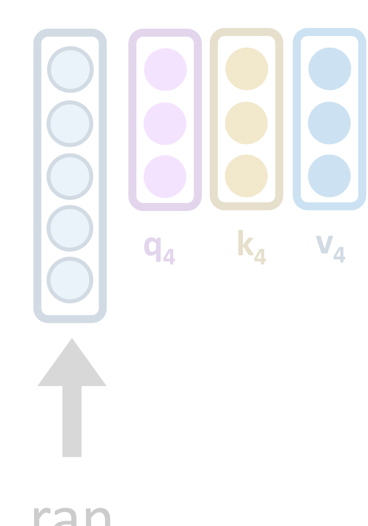






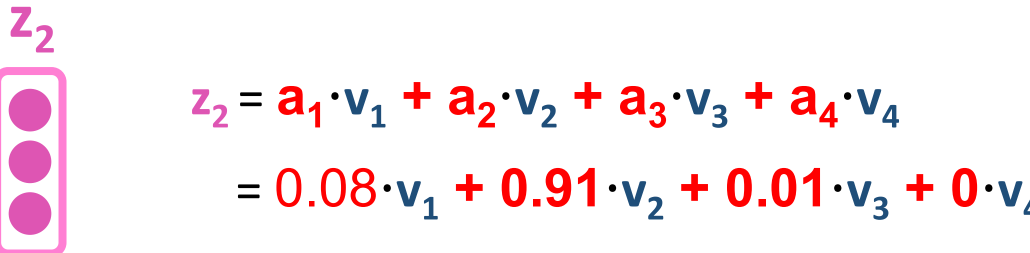






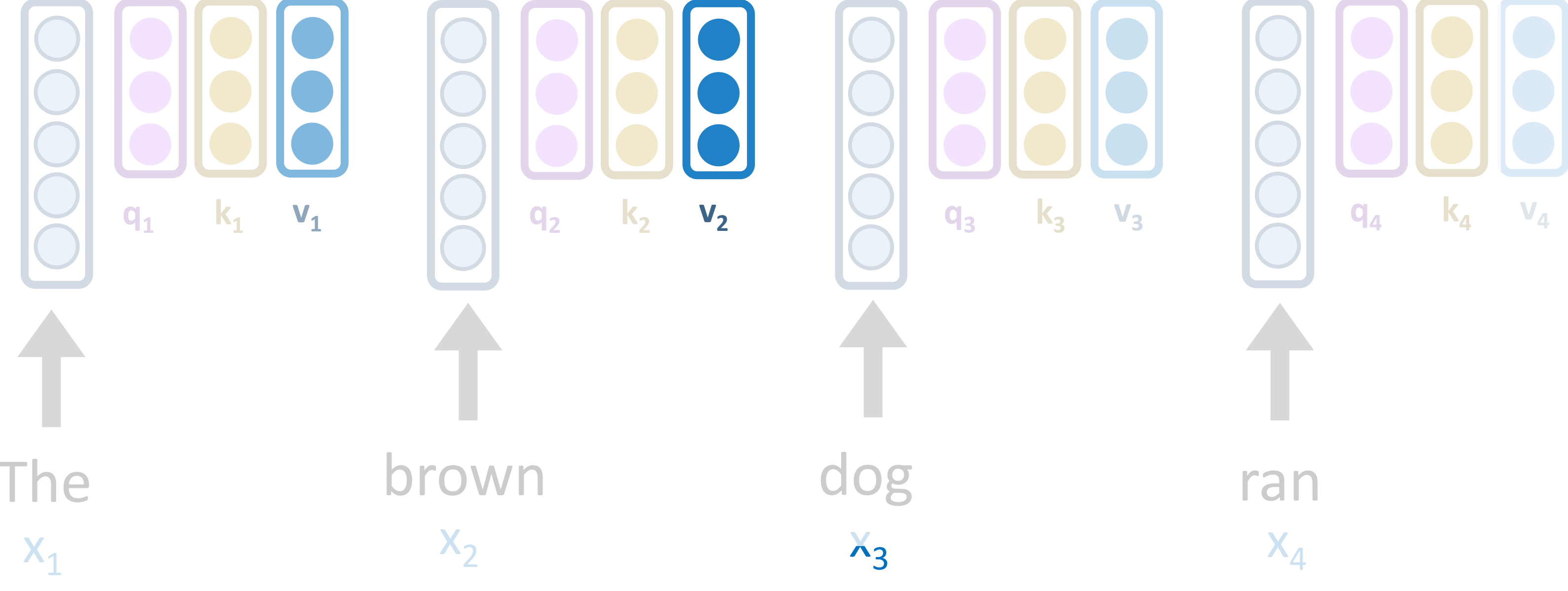
Self-Attention

Step 4: Let's weight our $\mathbf{v}_i$ vectors and simply sum them up!



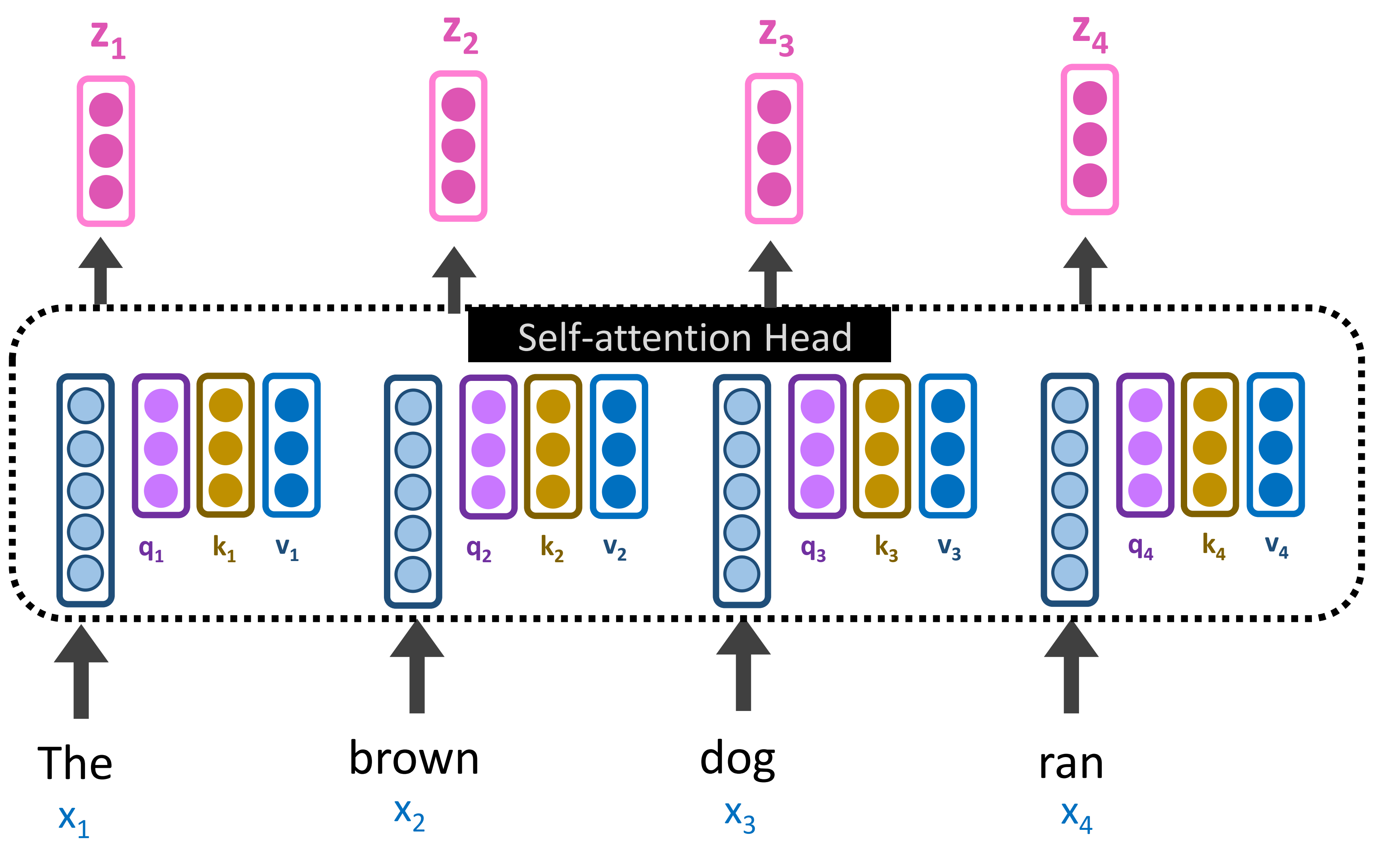
$$z_2 = a_1 \cdot v_1 + a_2 \cdot v_2 + a_3 \cdot v_3 + a_4 \cdot v_4$$

$$= 0.08 \cdot v_1 + 0.91 \cdot v_2 + 0.01 \cdot v_3 + 0 \cdot v_4$$



Self-Attention

Tada! Now we have great, new representations z_i via a self-attention head



Takeaway:
Self-Attention is powerful; allows us to create great, context-aware representations
The
brown
dog
ran

Self-Attention

$$\boldsymbol{b} = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\alpha}\right)V$$

hardmaru
@hardmaru

The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d, $X \in \mathbf{R}^{n \times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d \times d_q}$, $W_K \in \mathbf{R}^{d \times d_k}$, and $W_V \in \mathbf{R}^{d \times d_v}$ to extract feature representations Q, K, and V, referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q, K, V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \tag{1}$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \tag{2}$$

where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

9:08 PM · Feb 9, 2021 · Twitter Web App

553 Retweets 42 Quote Tweets 3,338 Likes

Outline

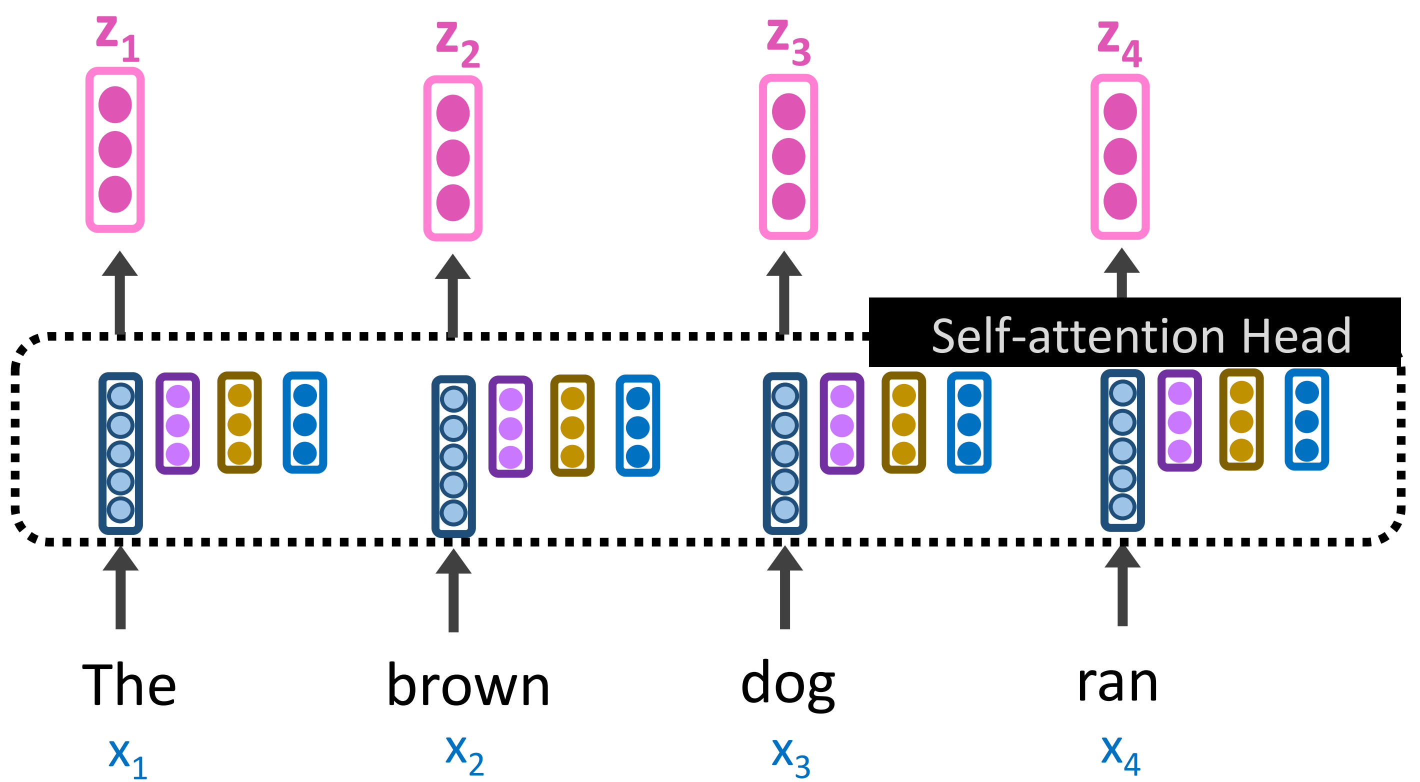
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Outline

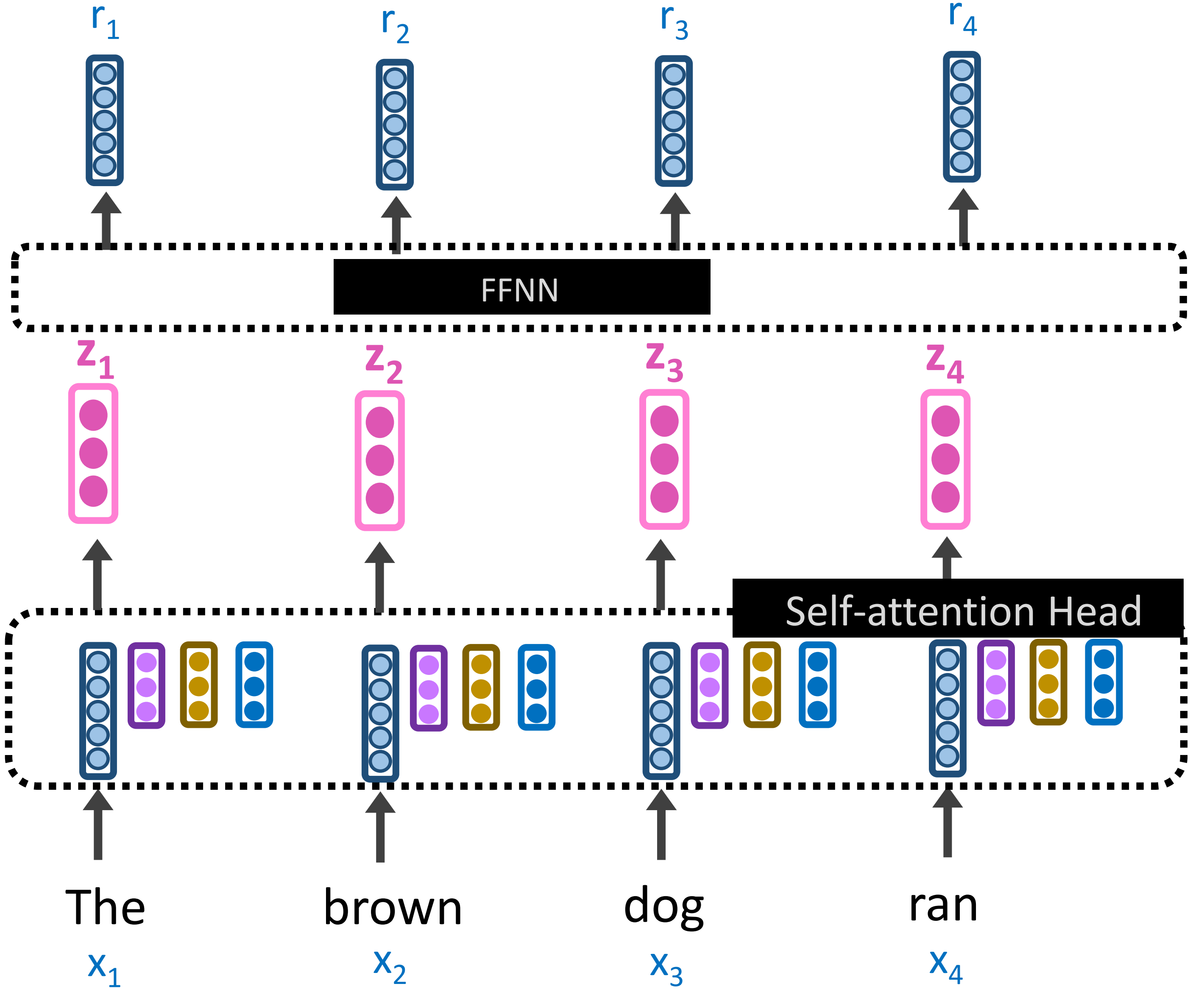
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Self-Attention

Let's further pass each z_i through a FeedForward NN

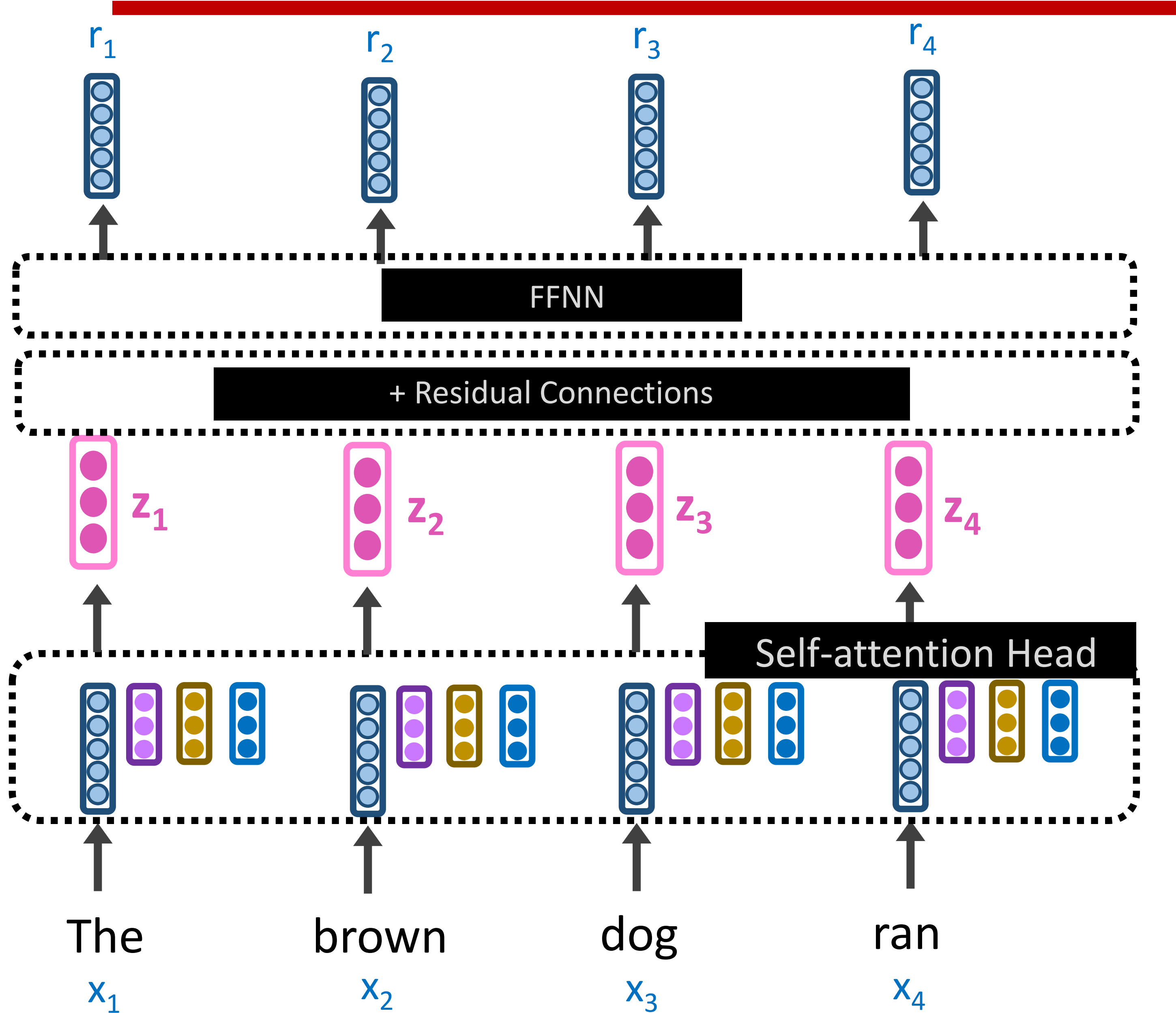


Self-Attention + FFNN



Let's further pass each z_i through a Feed Forward NN

Self-Attention + FFNN + Residual Connections

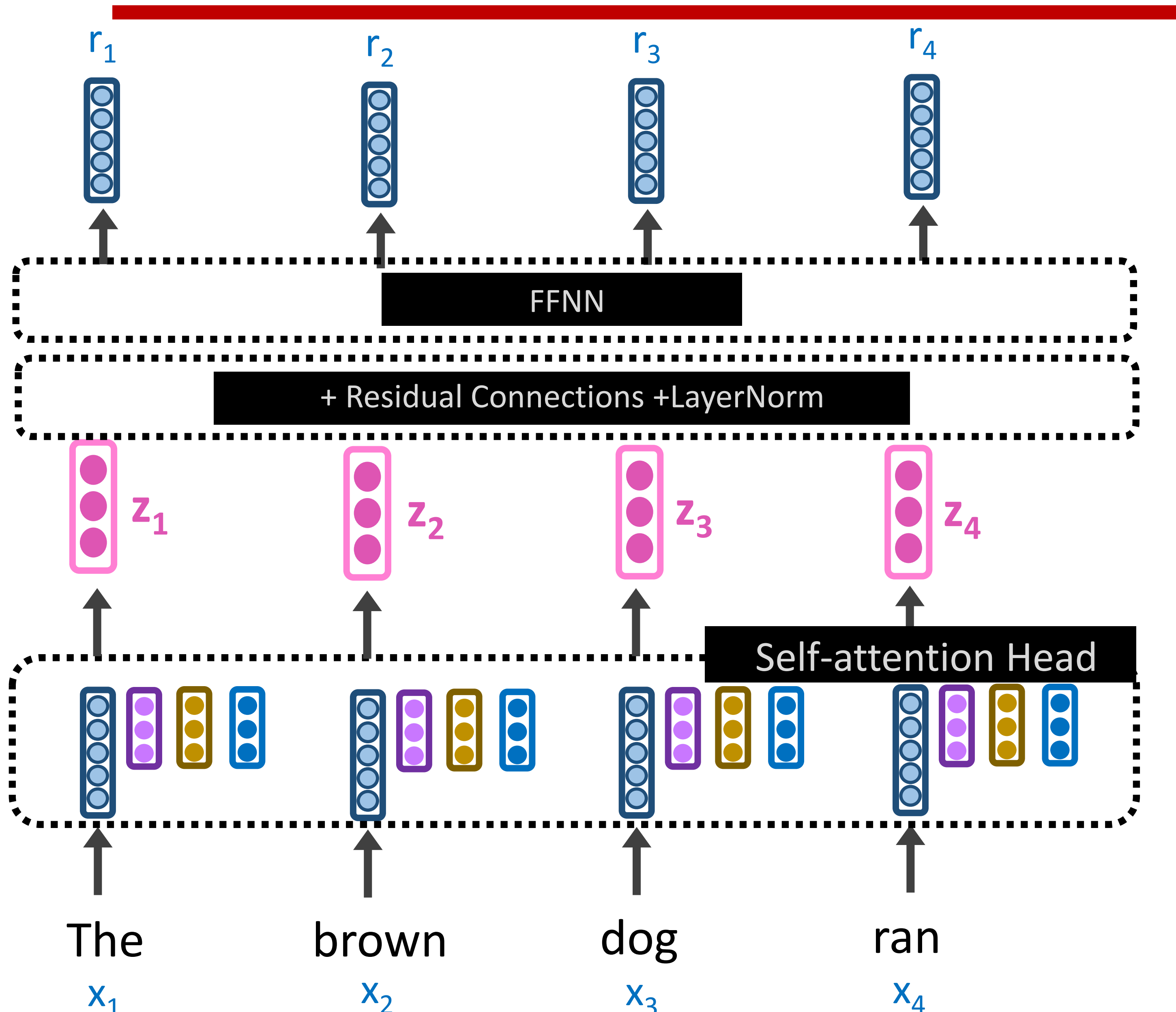


Let's further pass each z_i through a FFNN

We add a residual connection to help ensure relevant info is getting forward passed.

$$v = z + x$$

Self-Attention + FFNN + Residual Connections



Let's further pass each z_i through a FFNN

We add residual connection to help ensure relevant info is getting forward passed.

$$v = z + x$$

We perform LayerNorm to stabilize the network and allow for proper gradient flow. You should do this after the FFNN, too.

Stabilizing Gradient Flow: Residual Connection and Layernorm

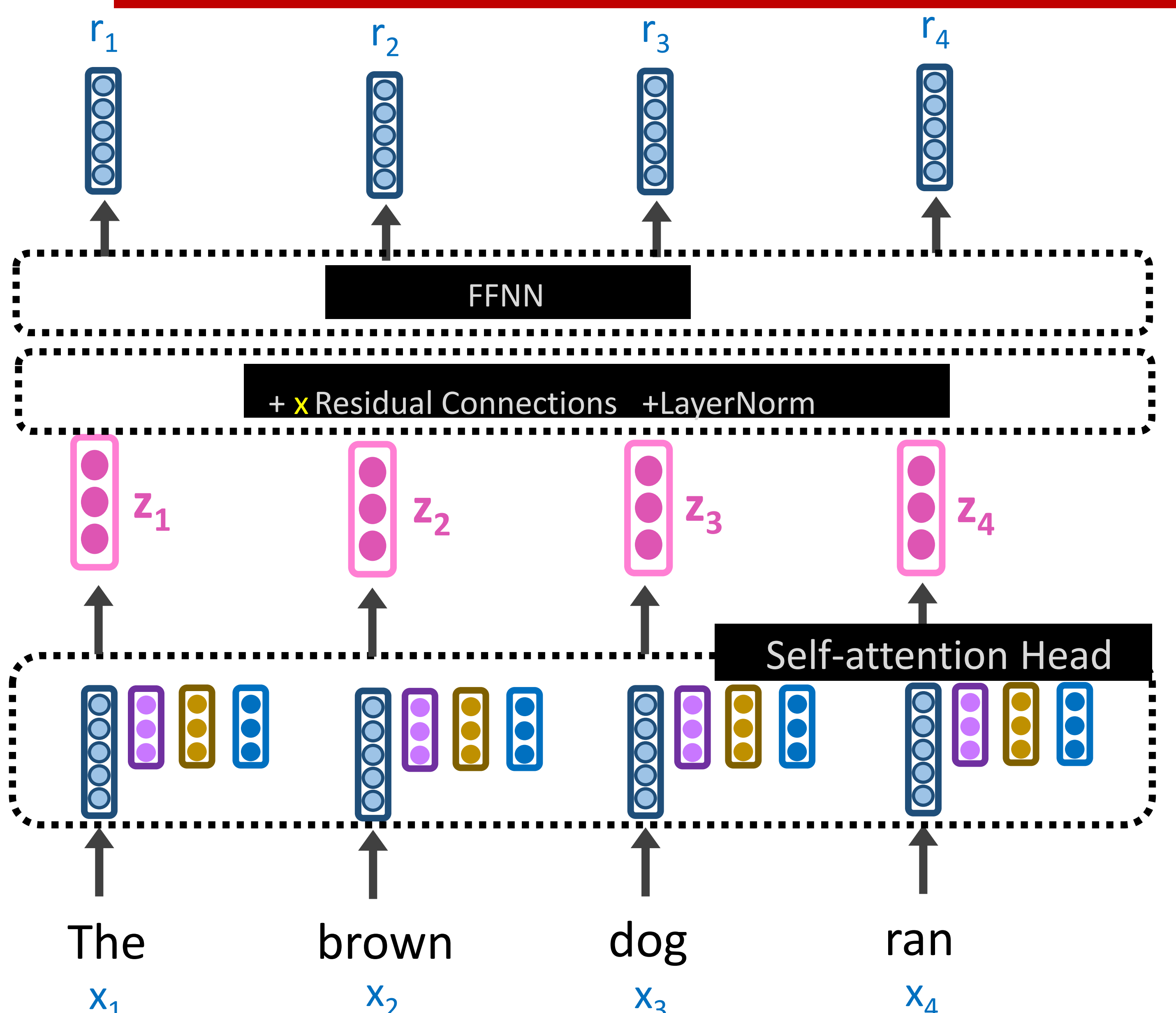
- Residual connection: y = f(x) + x
  - f might be a complex function and gives small gradients wrt x, adding x back to f(x) gives higher values of the gradient

- Layer Normalization (LayerNorm):
  - Another way to prevent vanishing gradients

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

arxiv.org/pdf/2409.12951

Self-Attention + FFNN



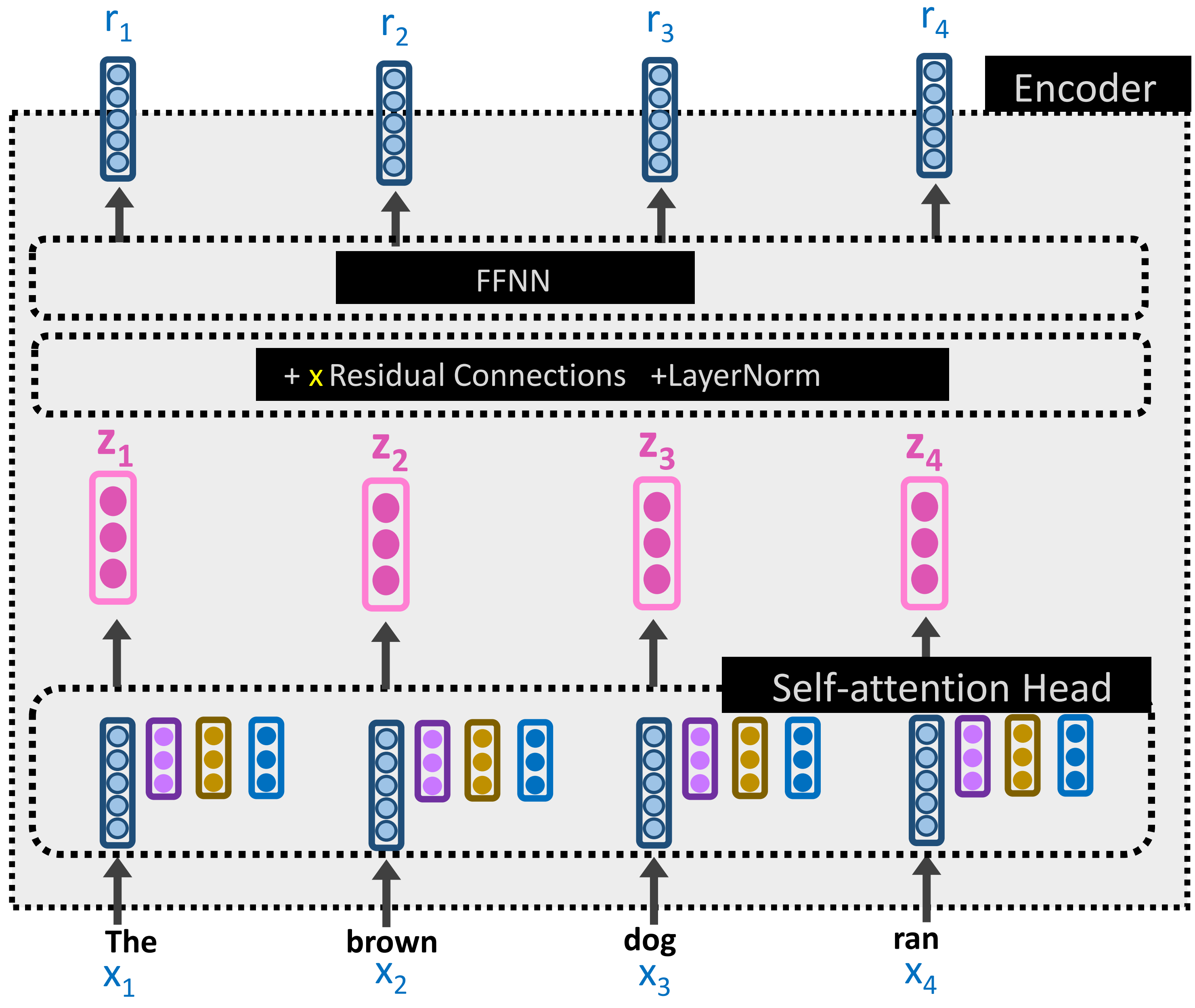
Let's further pass each z_i through a FFNN

We concat w/ a residual connection to help ensure relevant info is getting forward passed.

We perform LayerNorm to stabilize the network and allow for proper gradient flow. You should do this after the FFNN, too.

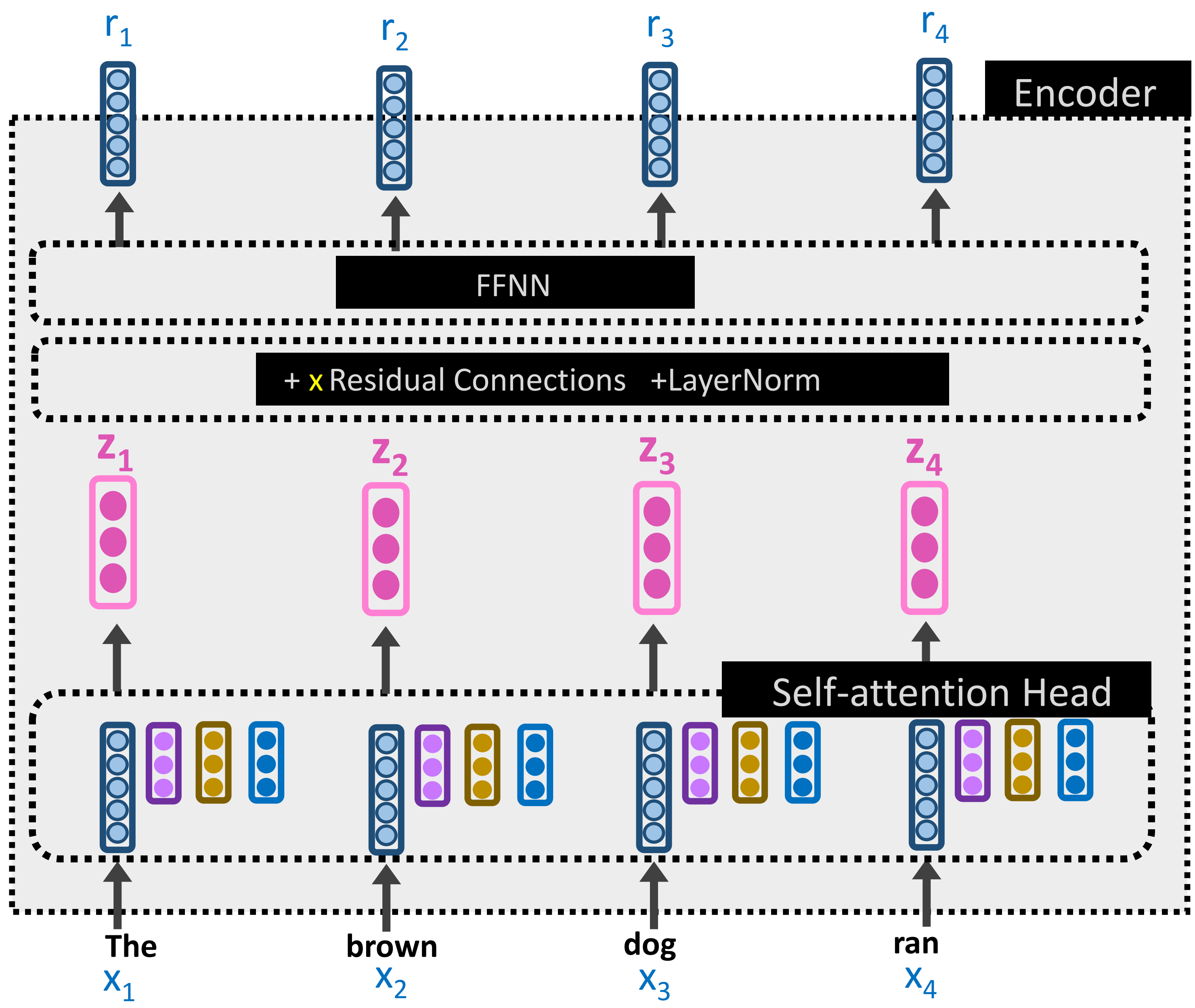
Each z_i can be computed in parallel, unlike RNNs!

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

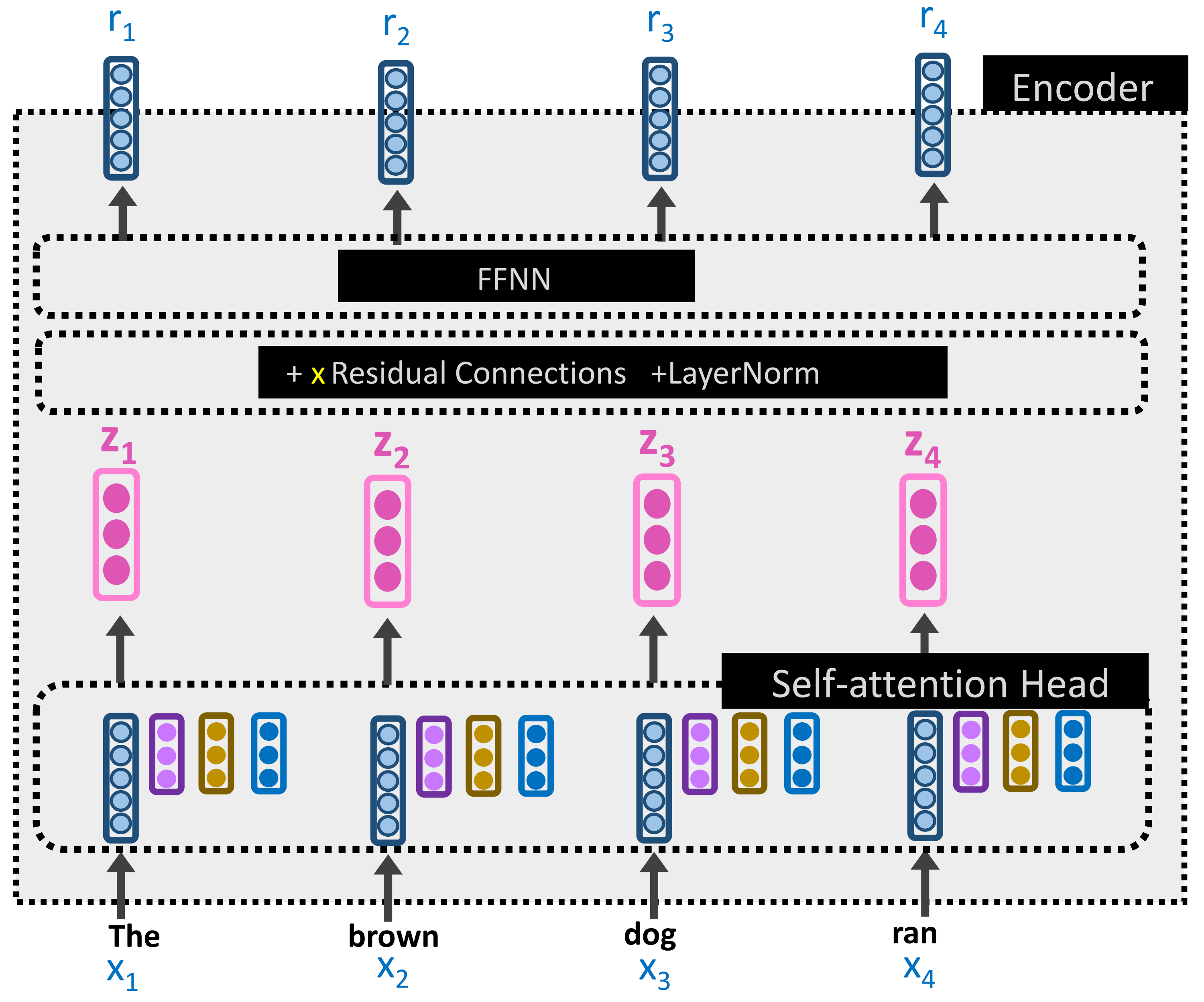
Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Solution: add to each input word x_i a positional encoding

Input to the model is now
$x_i + pos_i$

How to encode position information?

- Self attention doesn't have a way to know whether an input token comes before or after another
  - Position is important in sequence modeling in NLP

- A way to introduce position information is add individual position encodings to the input for each position in the sequence

$$x_i = x_i + pos_i$$

Where pos_{t} is a position vector

Properties of a good positional embedding

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
  - The cat sat on the mat
  - The happy cat sat on the mat
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

Absolute position embeddings

- Define a maximum context length you model can encode: say 1000 tokens.
    - Create a separate embedding table for each position.
    - Each index 1, 2, 3, ... gets an embedding.
    - Learn the embeddings with the model.

- Issues with Learned positions embeddings:
  - Maximum length that can be presented is limited (what if I get a 2000 token input)
  - Difficult to encode relative positions
    - The cat sat on the mat
    - The happy cat sat on the mat

Functional (and fixed) position embeddings

Sinusoidal embeddings

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}$$

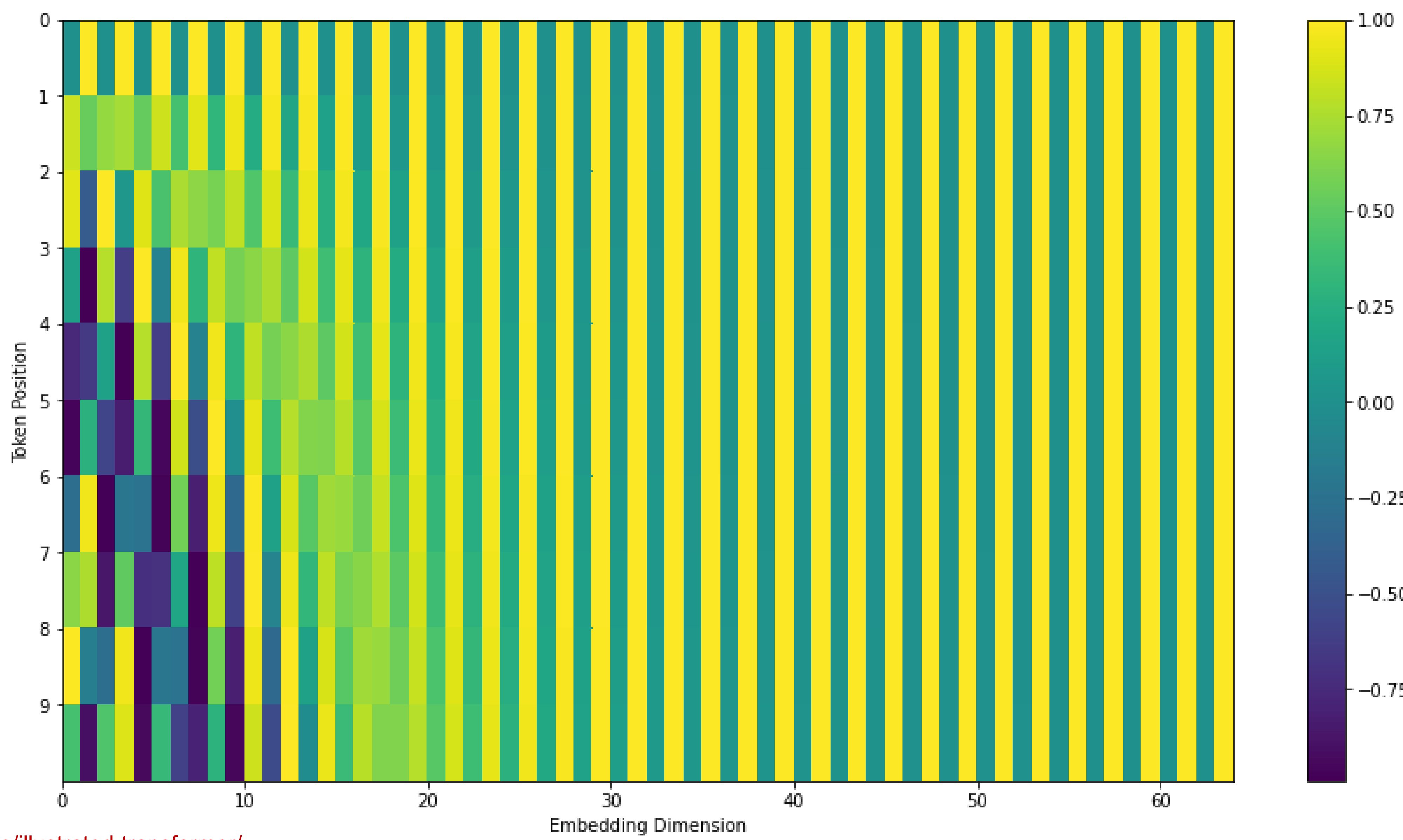
The frequencies are decreasing along the vector dimension. It forms a geometric progression on the wavelengths.

Sinusoidal Embeddings: Intuition

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

[Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog]

Position Encodings



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

Variants of Positional Embeddings

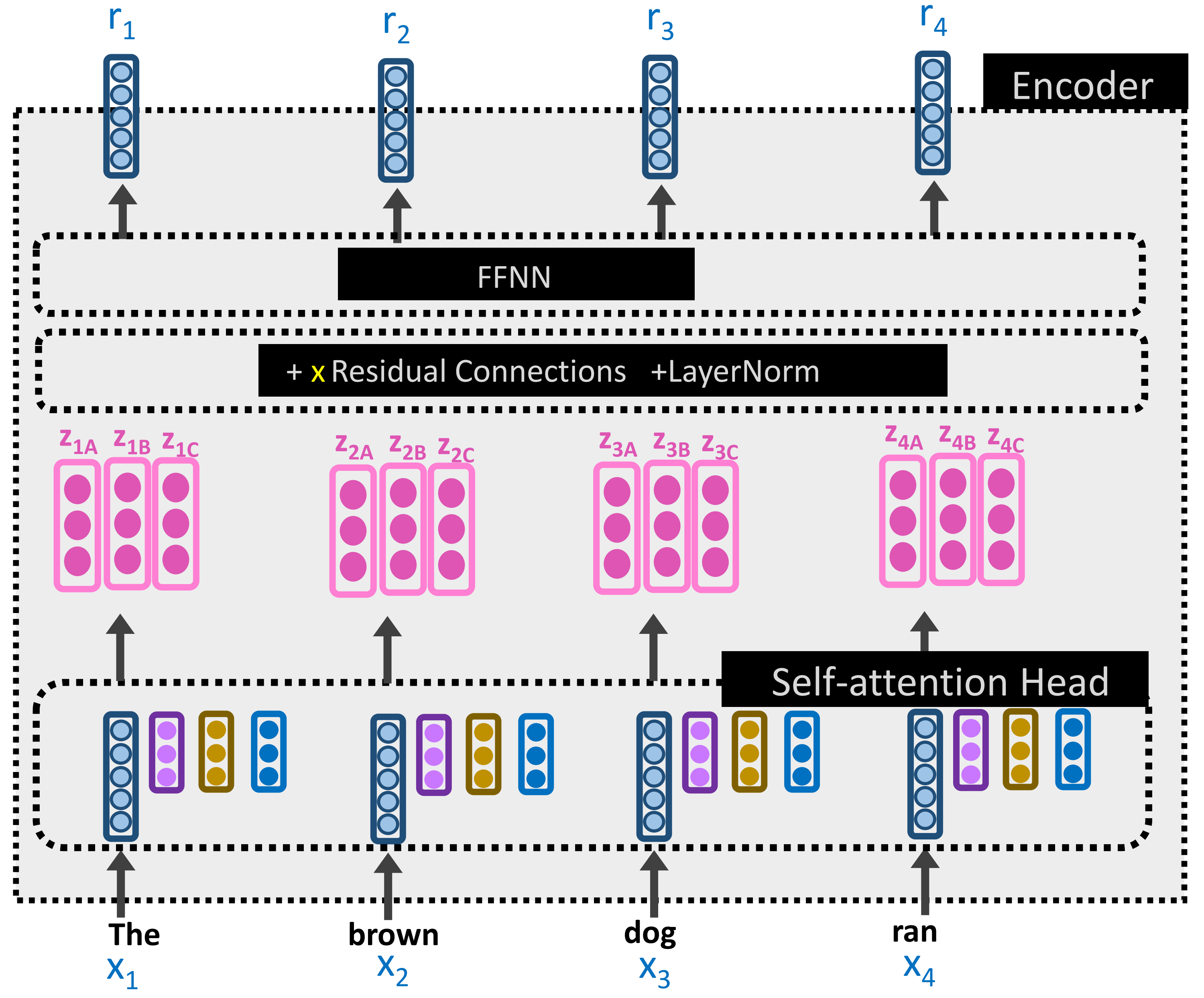
- Rotary Positional Embeddings (RoPE): [2104.09864] RoFormer: Enhanced Transformer with Rotary Position Embedding (arxiv.org)
- AliBi: [2108.12409] Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (arxiv.org)
- No embeddings(!?): [2203.16634] Transformer Language Models without Positional Encodings Still Learn Positional Information (arxiv.org)

A Self-Attention Head has just one set of query/key/value weight matrices w_q, w_k, w_v

Words can relate in many ways, so it's restrictive to rely on just one Self-Attention Head in the system.

Let's create Multi-headed Self-Attention

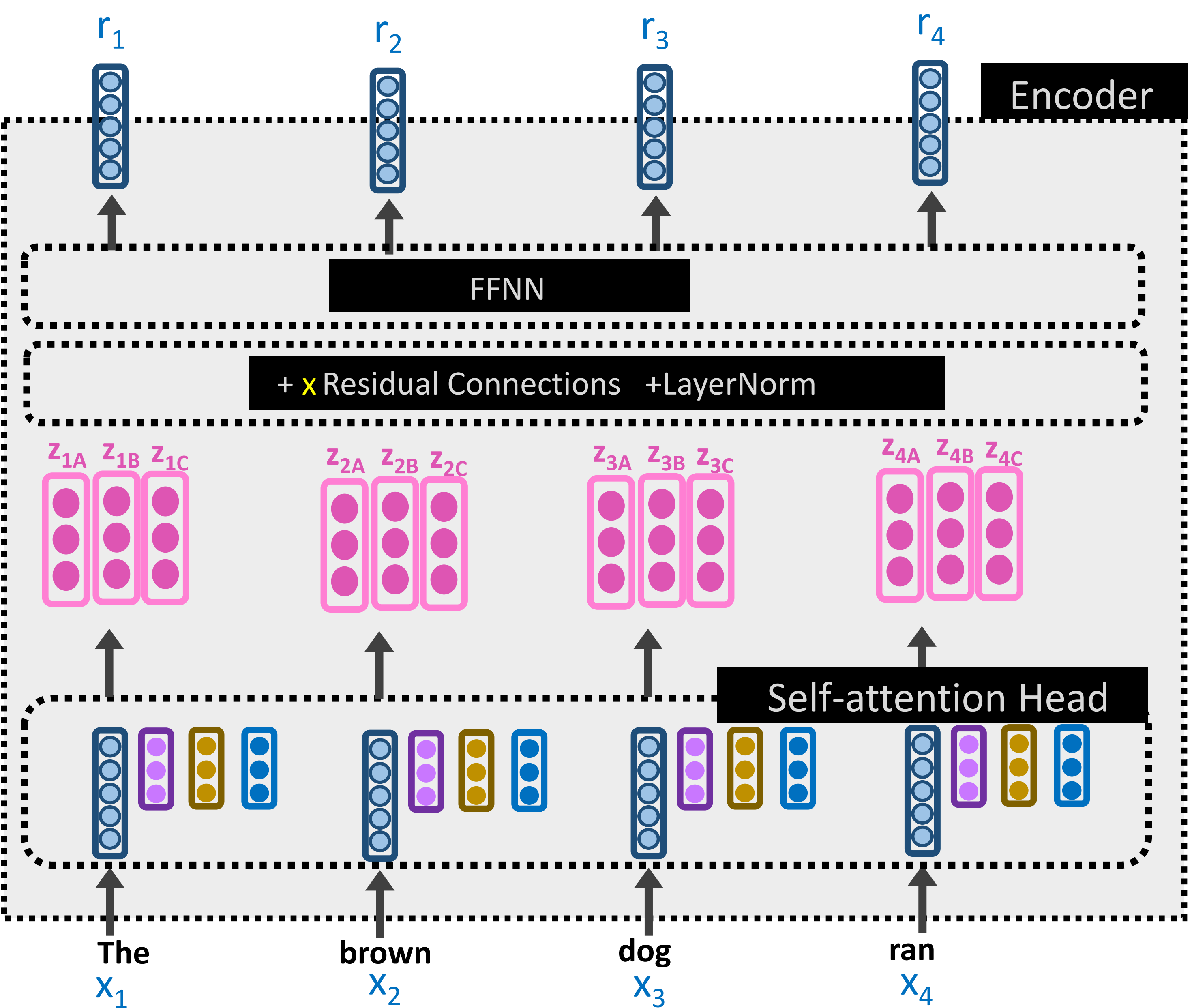
Multi-head Attention



Each Self-Attention Head produces a z_i vector using query, key, and value vectors

We can, in parallel, use multiple heads and concat the z_i's. For each input create multiple query, key, and value vectors

Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Transformer Encoder

r_1 r_2 r_3 r_4

Encoder

FFNN

+ x Residual Connections +LayerNorm

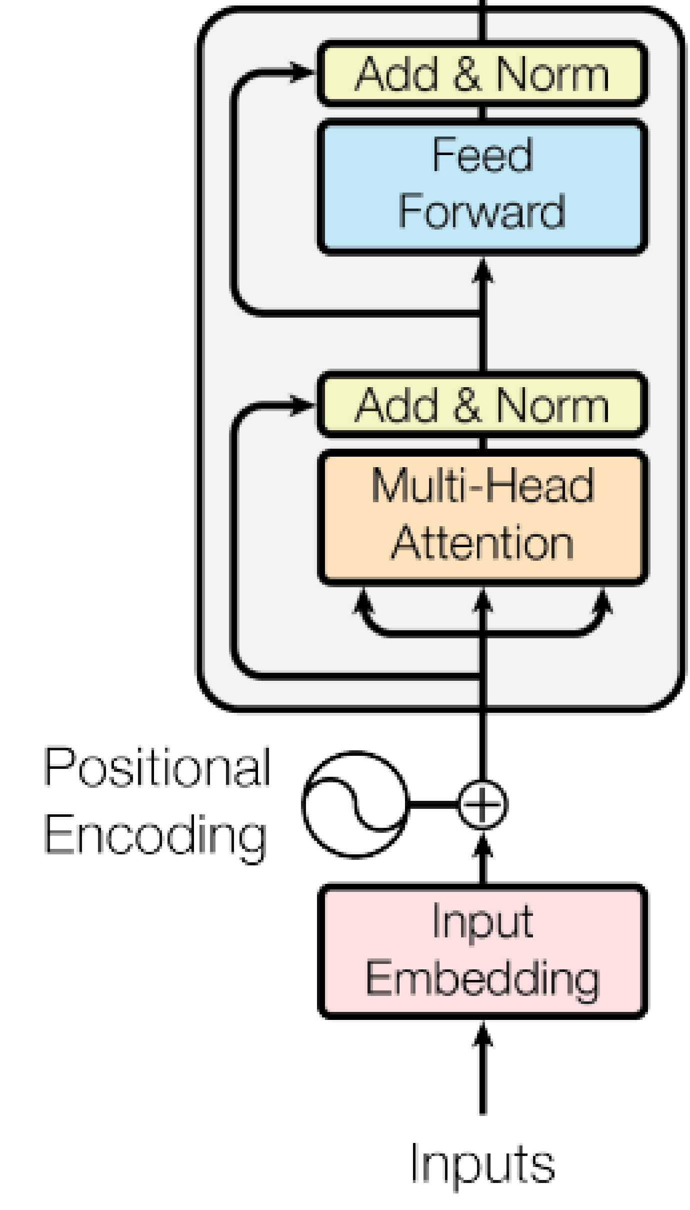
z_{1A} z_{1B} z_{1C} z_{2A} z_{2B} z_{2C} z_{3A} z_{3B} z_{3C} z_{4A} z_{4B} z_{4C}

Self-attention Head

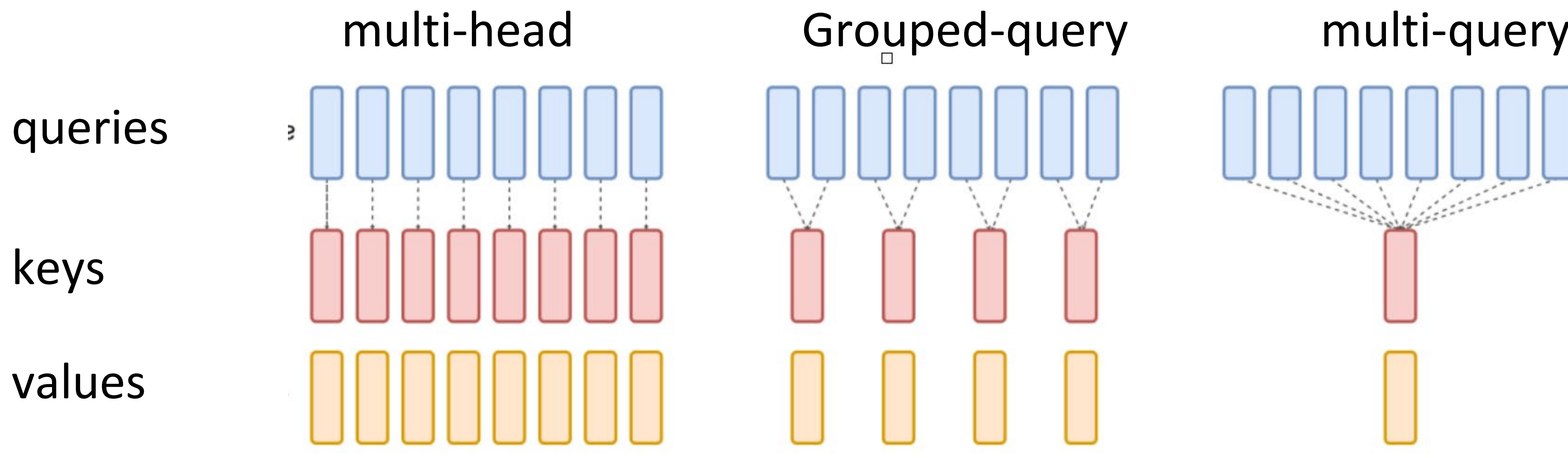
The brown dog ran

x_1 x_2 x_3 x_4

=



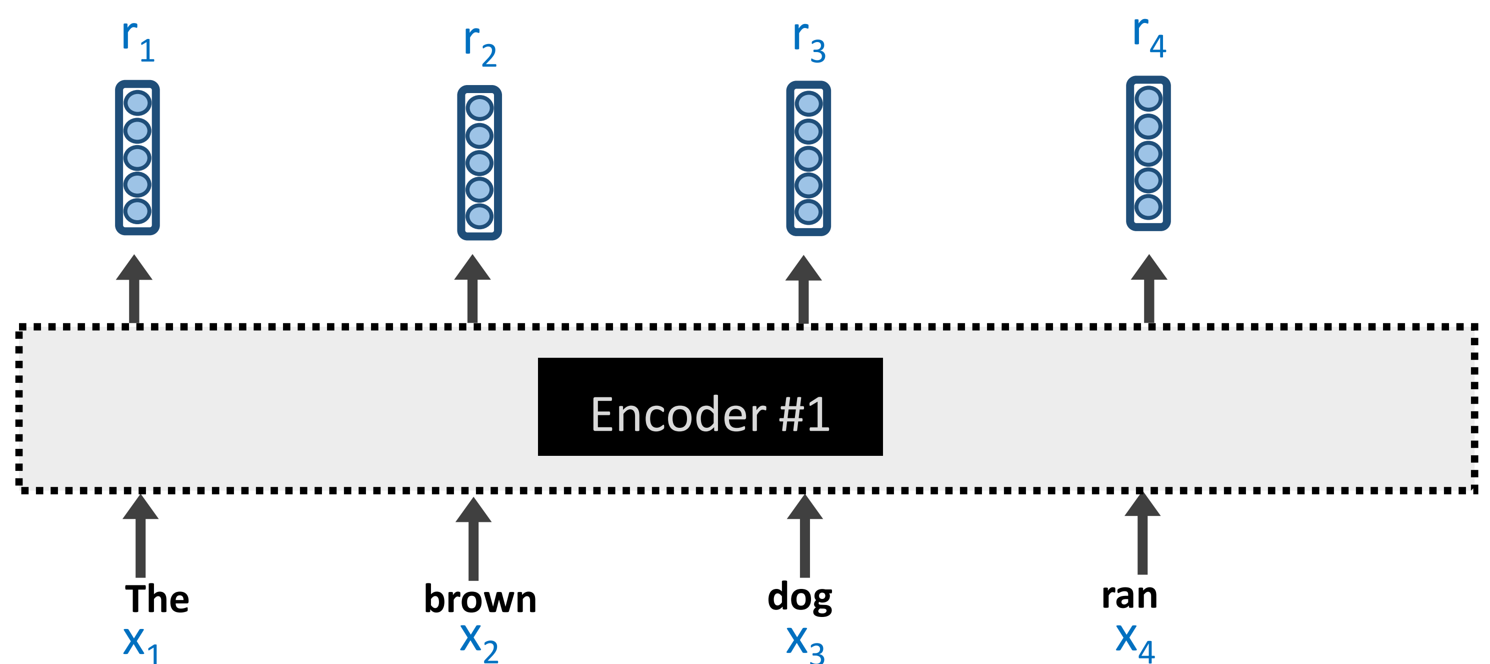
Variants of multi-head attention attention



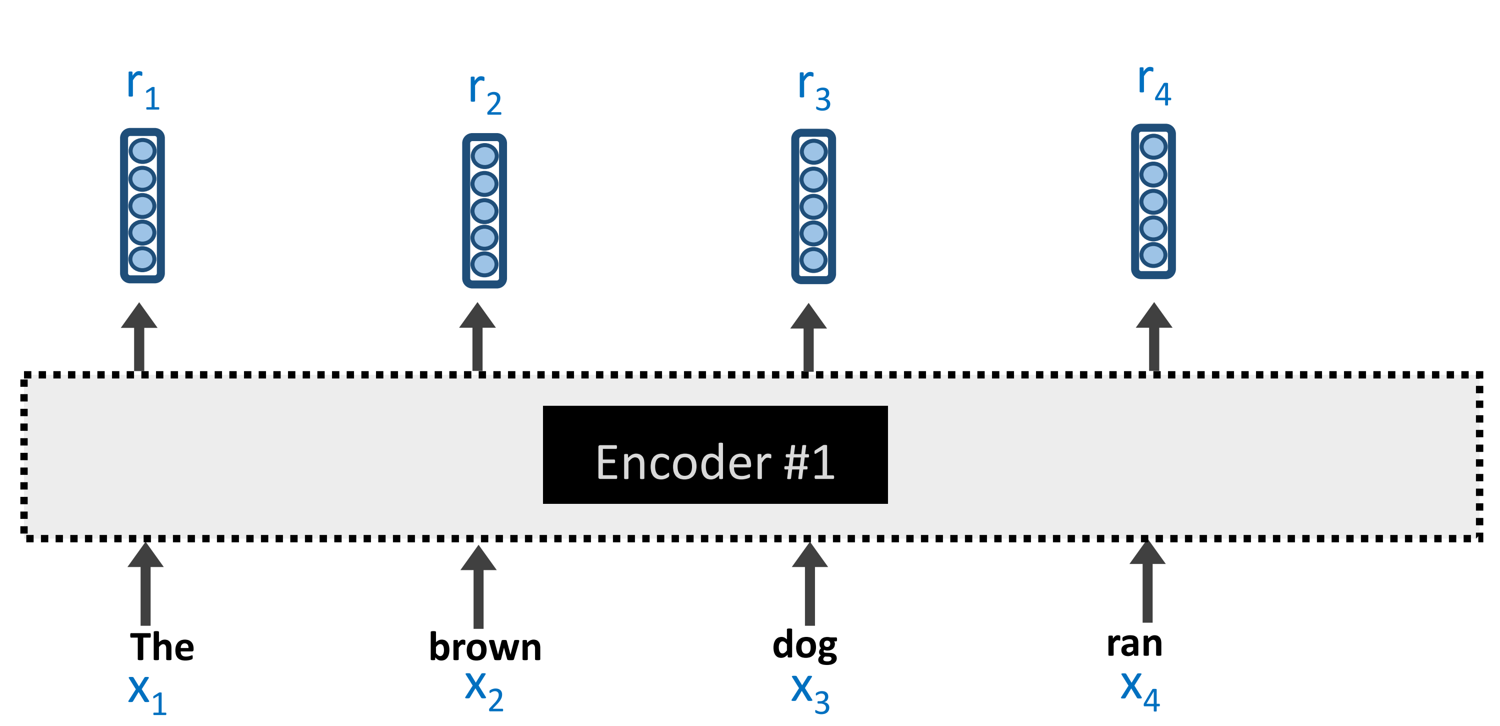
GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints (Ainslie et al., 2023)

Transformer Encoder

To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i



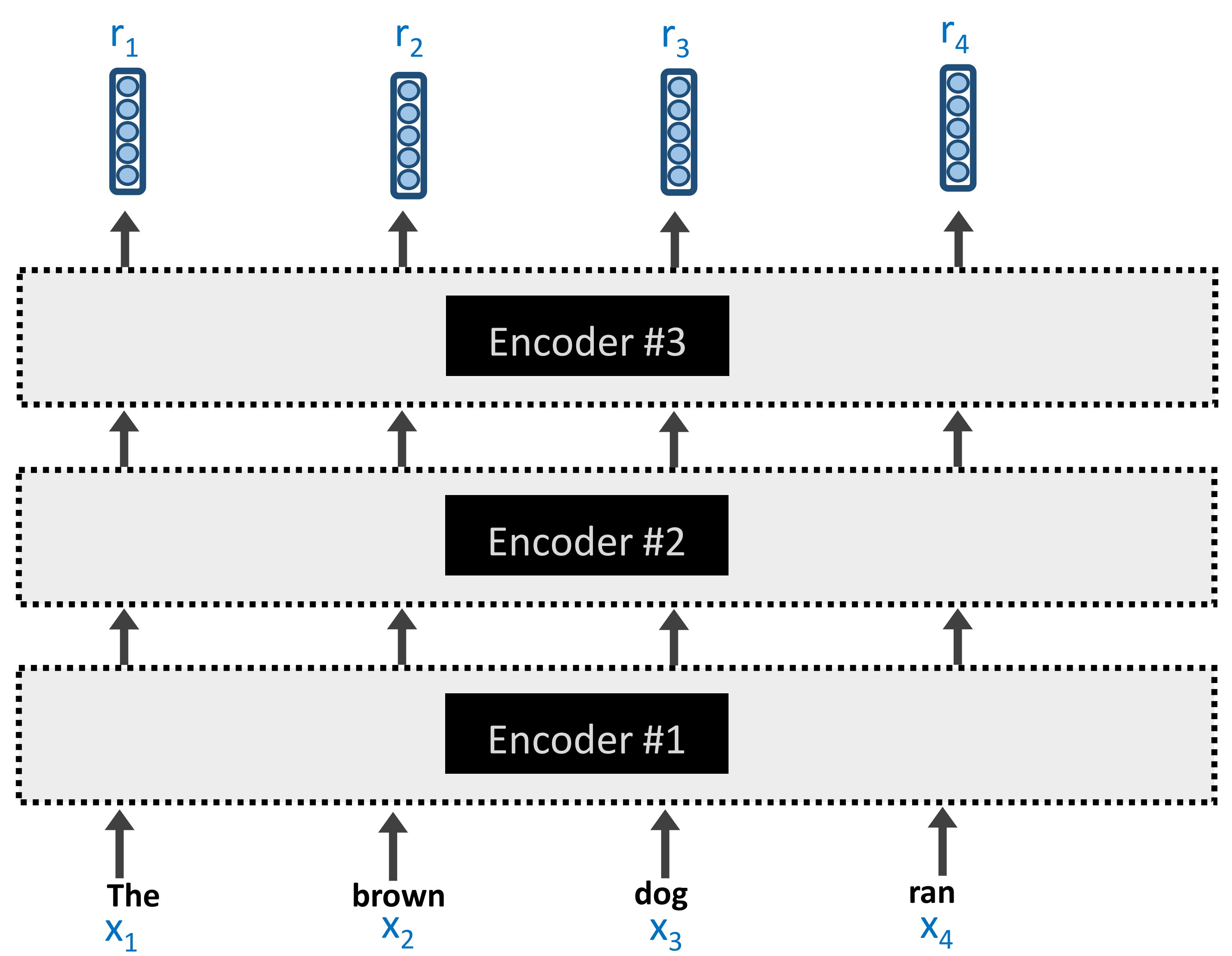
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

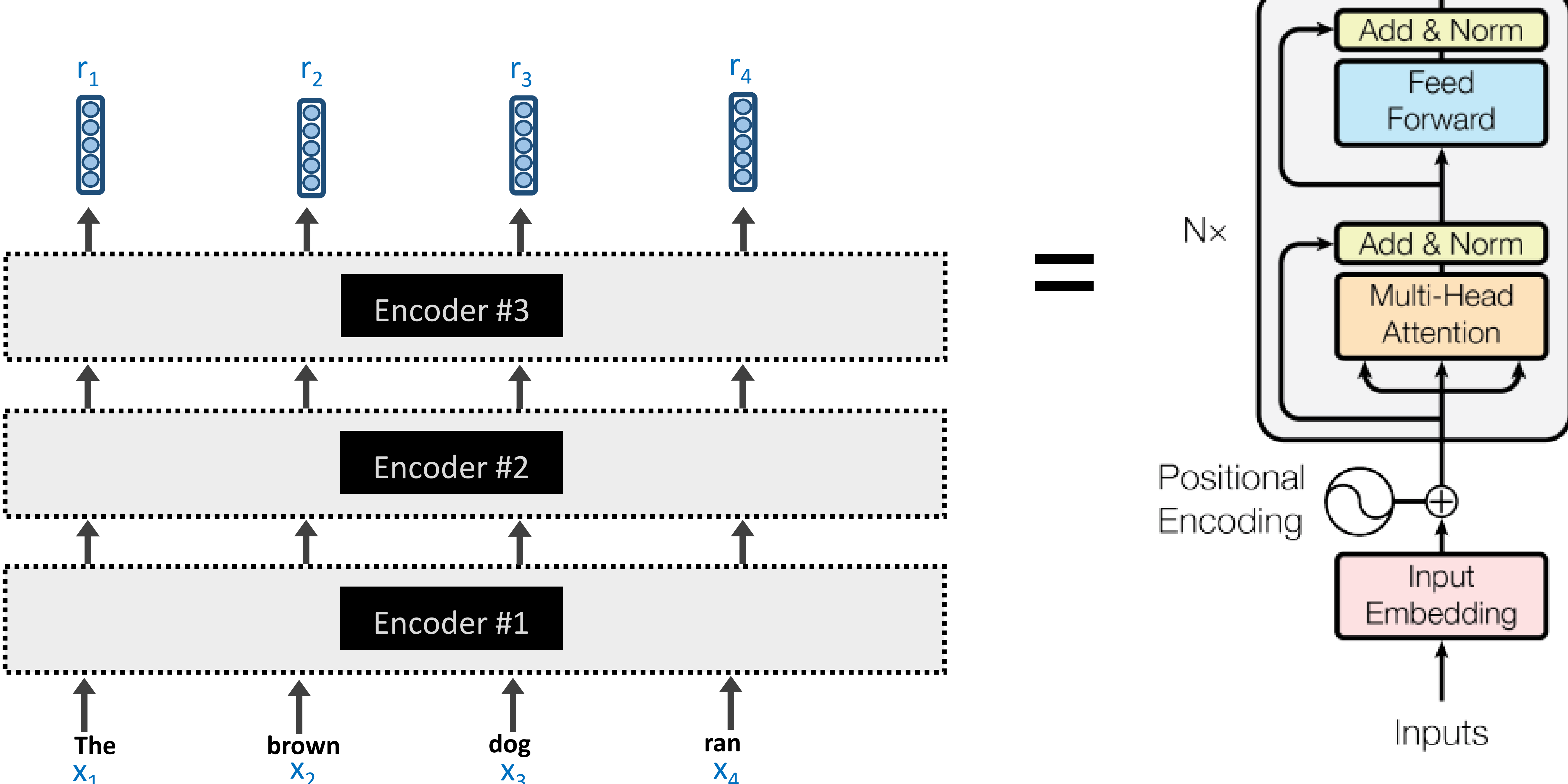
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

Transformer Encoder



The original Transformer model was intended for Machine Translation, so it had Decoders, too

Outline

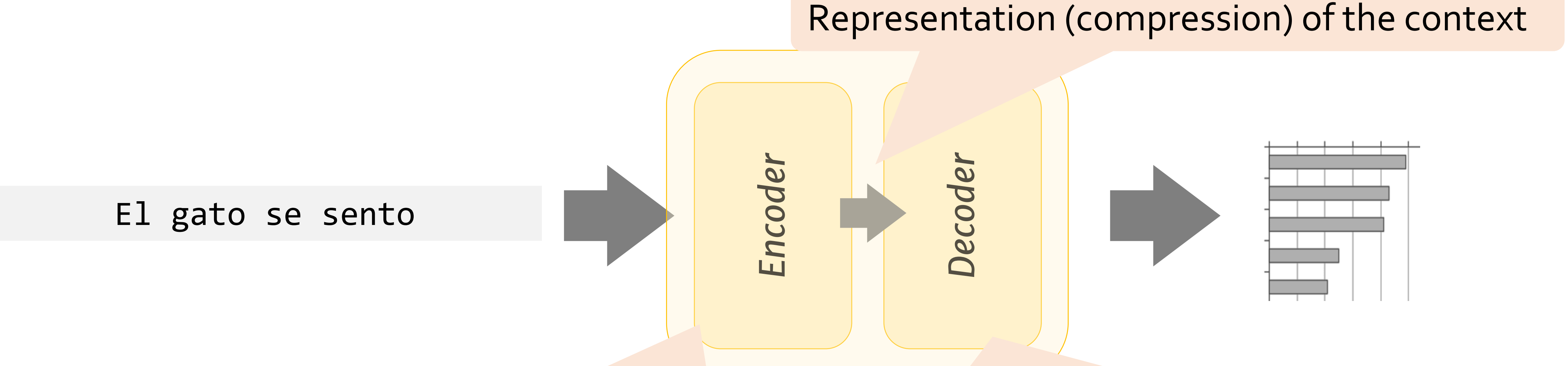
- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Encoder-Decoder Architectures

- Original transformer had two sub-models.



Encoder-Decoder Architectures

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Transformer Decoder

Decoder

r_1 r_2 r_3 r_4

FFNN

+ x Residual Connections +LayerNorm

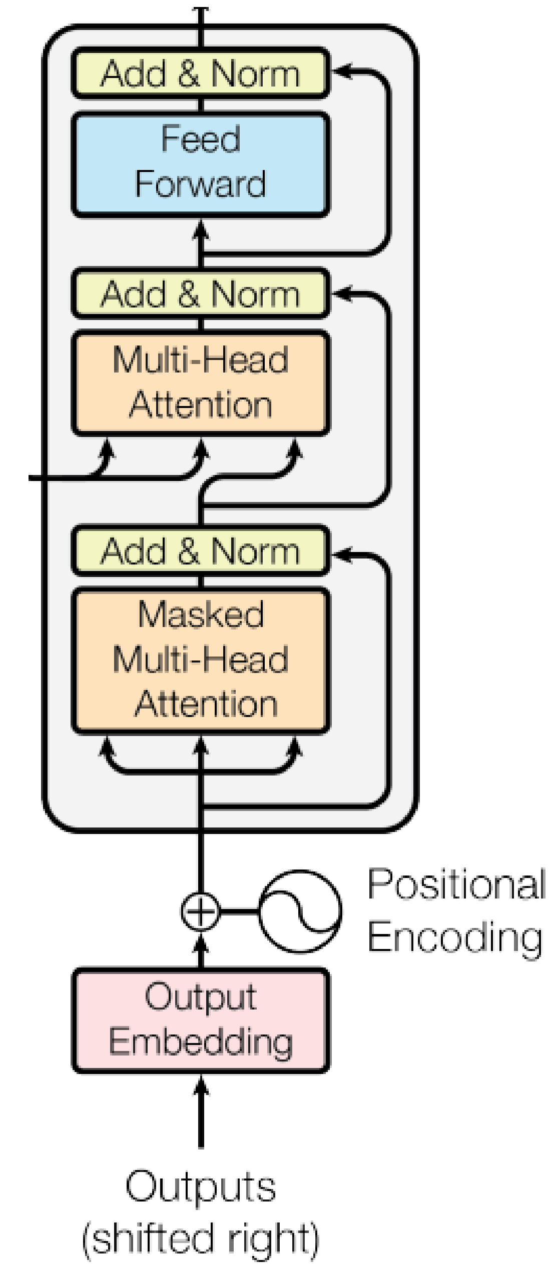
z_{1A} z_{1B} z_{1C} z_{2A} z_{2B} z_{2C} z_{3A} z_{3B} z_{3C} z_{4A} z_{4B} z_{4C}

Masked Self-attention Head

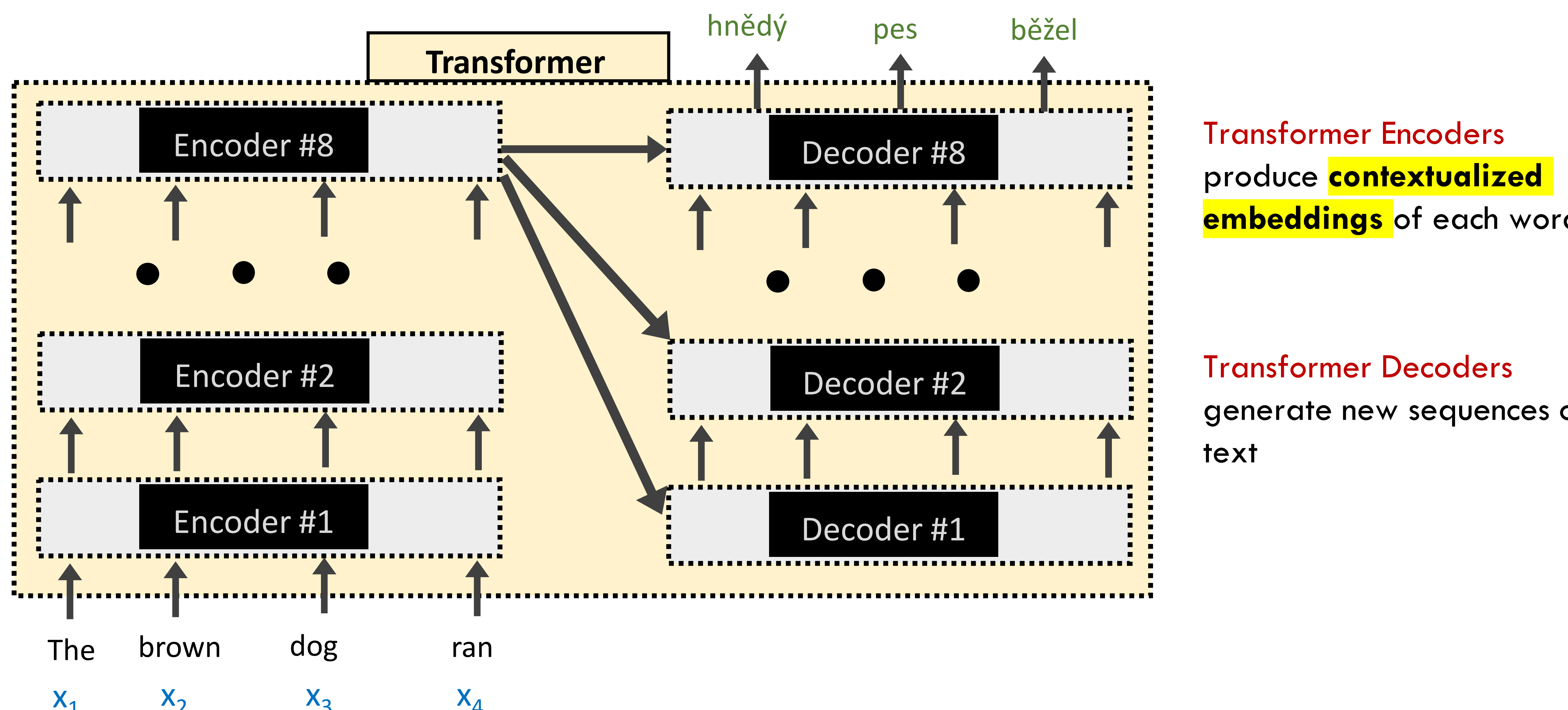
<s> El perro marrón

x_1 x_2 x_3 x_4

=



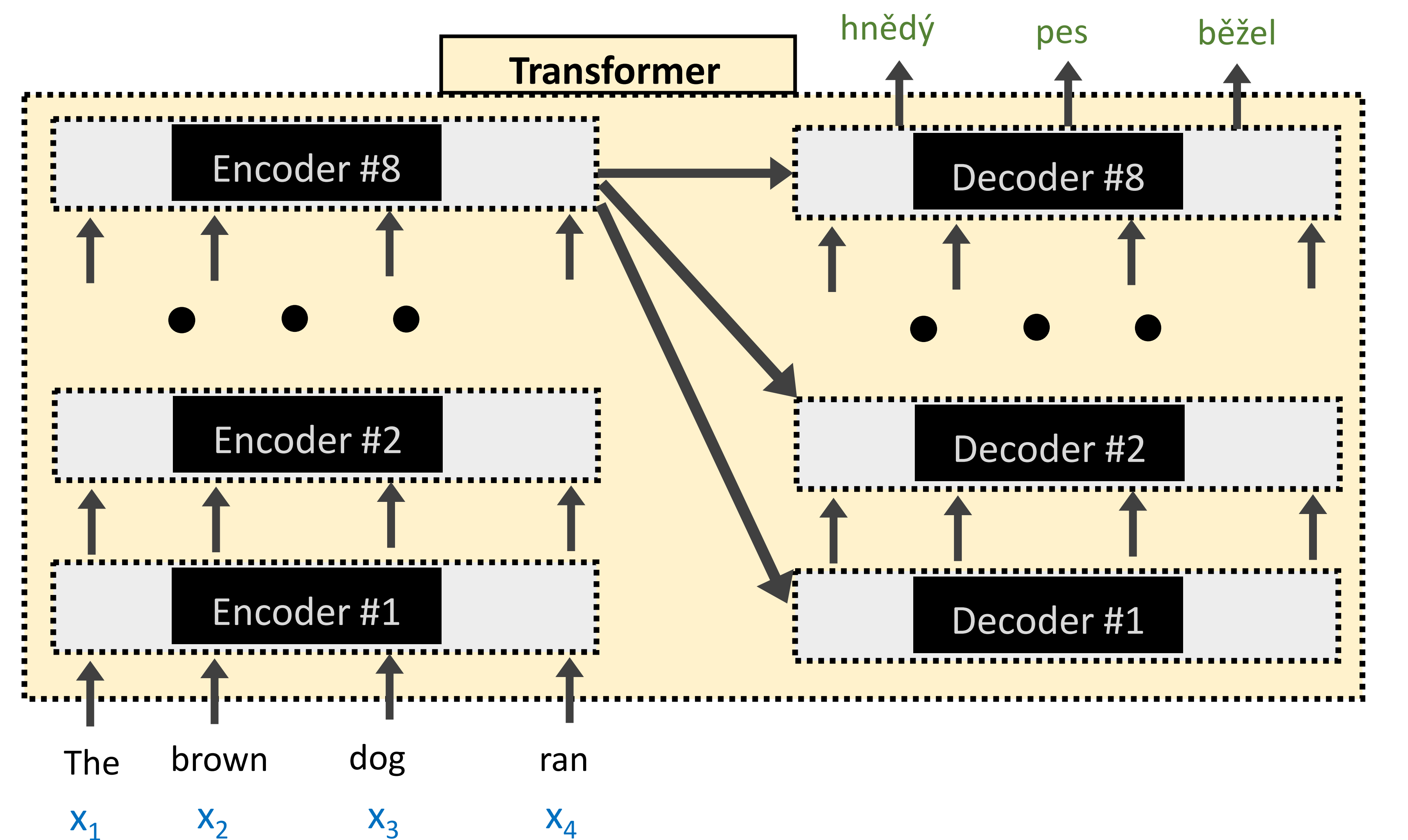
Transformer Encoders and Decoders



Transformer Encoders produce **contextualized embeddings** of each word

Transformer Decoders generate new sequences of text

Transformer Encoders and Decoders

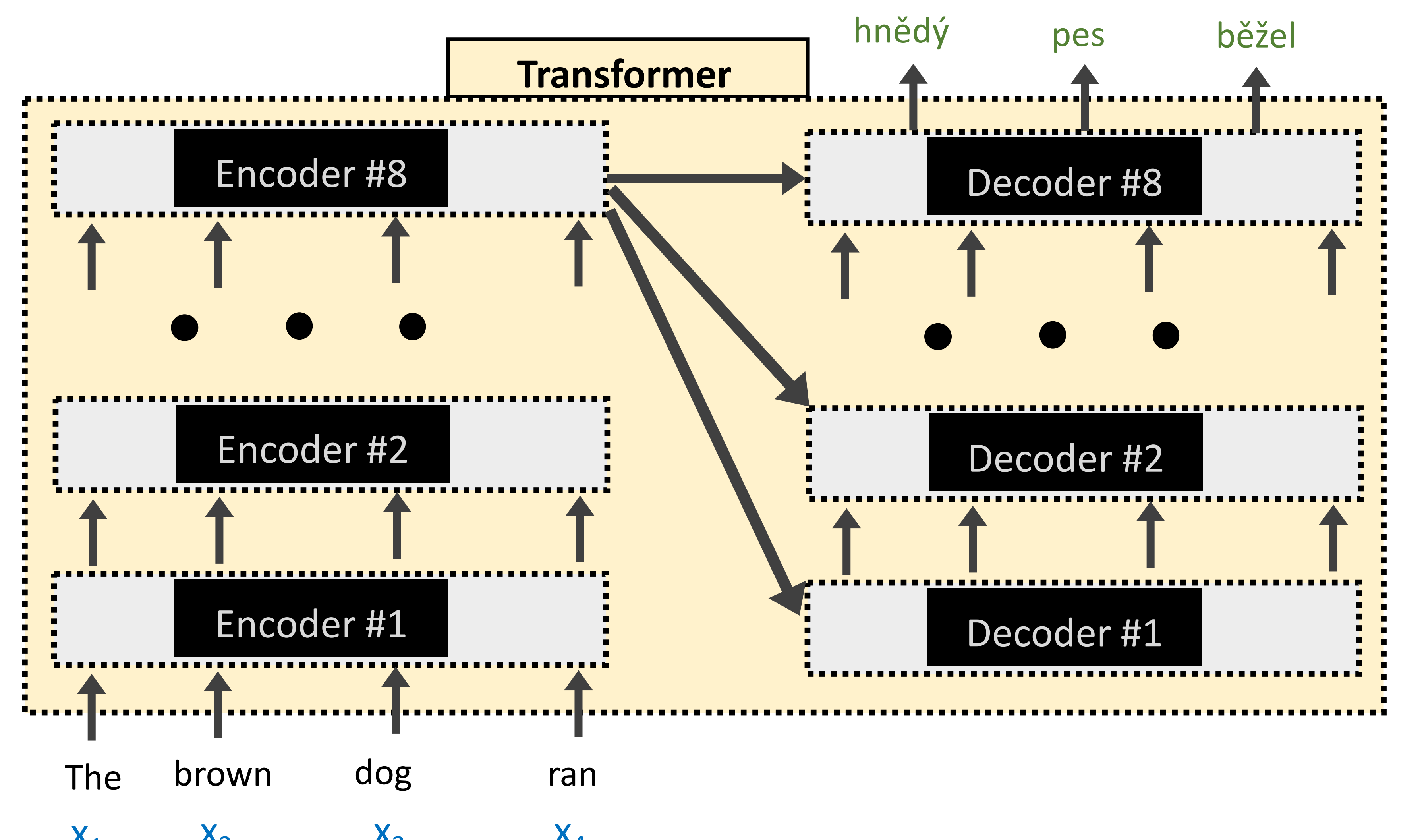


NOTE

Transformer Decoders are identical to the Encoders, except they have an additional Attention Head in between the Self-Attention and FFNN layers.

This additional Attention Head **focuses on parts of the encoder's representations.**

Transformer Encoders and Decoders

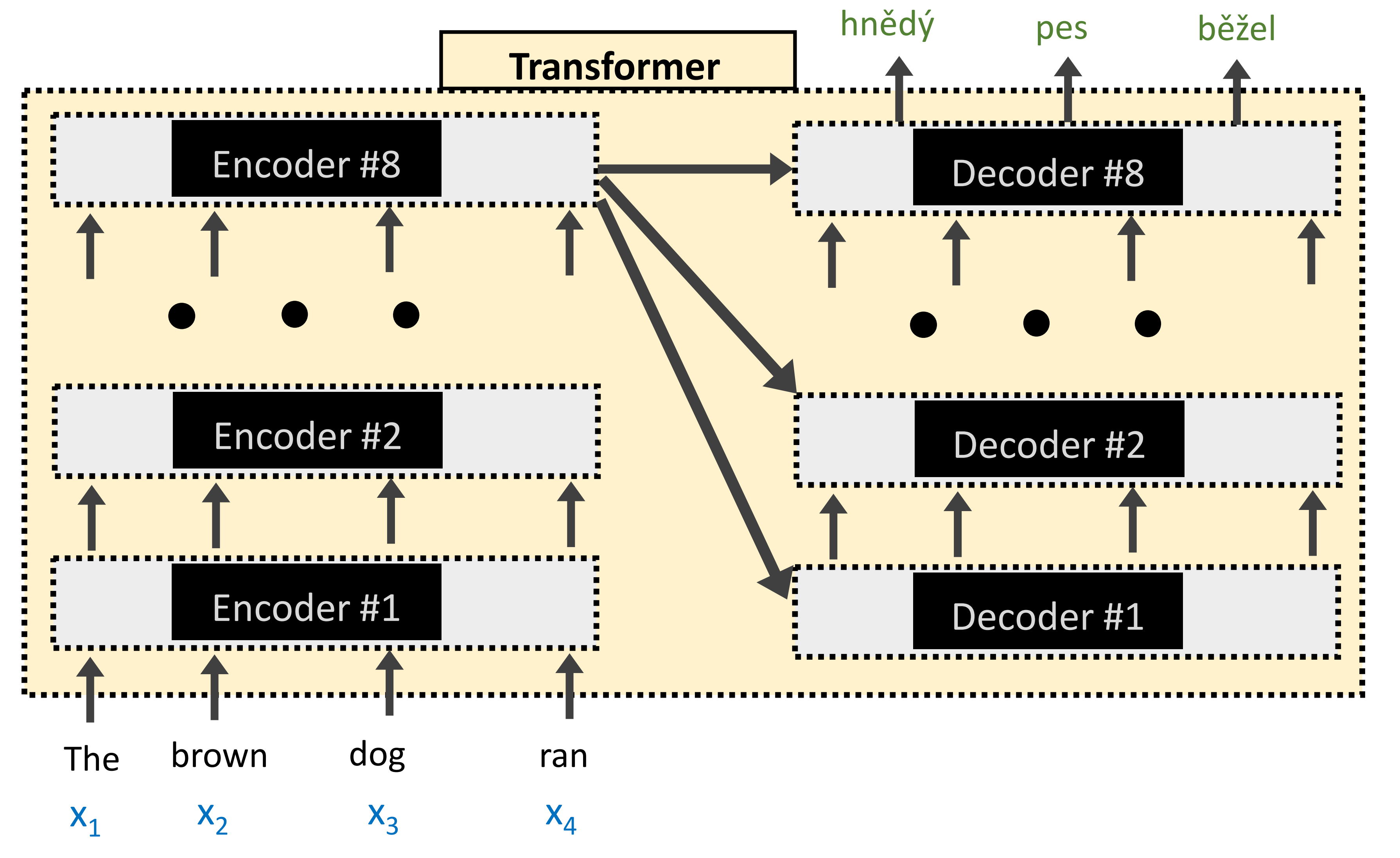


NOTE

The **query** vector for a Transformer Decoder's **Attention Head** (not Self-Attention Head) is from the output of the previous decoder layer.

However, the **key** and **value** vectors are from the Transformer Encoders' outputs.

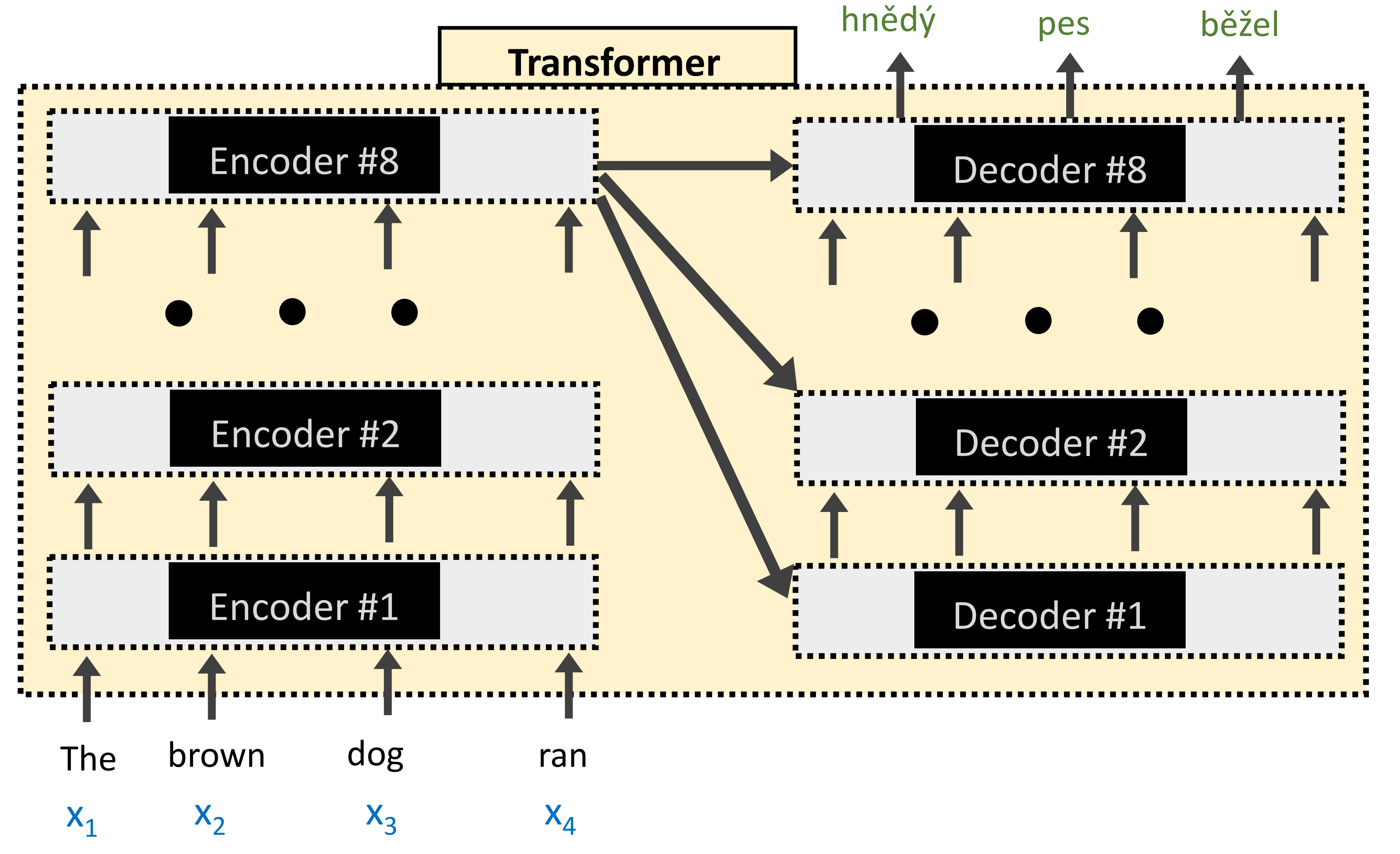
Transformer Encoders and Decoders



NOTE

The **query**, **key**, and **value** vectors for a Transformer Decoder's **Self-Attention Head** (not Attention Head) are all from the output of the previous decoder layer.

Transformer Encoders and Decoders



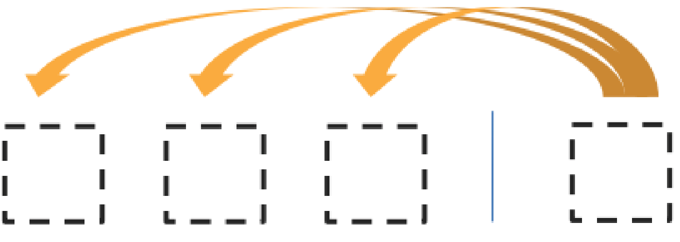
IMPORTANT

The Transformer Decoders have positional embeddings, too, just like the Encoders.

Critically, each position is only allowed to attend to the previous indices. This *masked* Attention preserves it as being an auto-regressive LM.

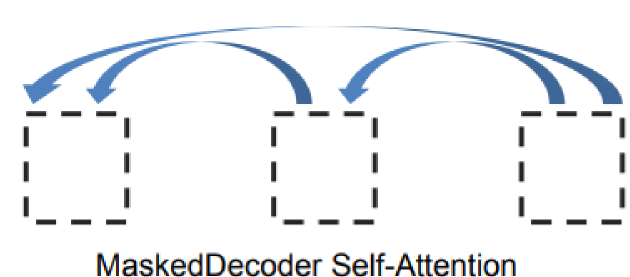
Transformer [Vaswani et al. 2017]

- An encoder-decoder architecture built with attention modules.
- 3 forms of attention

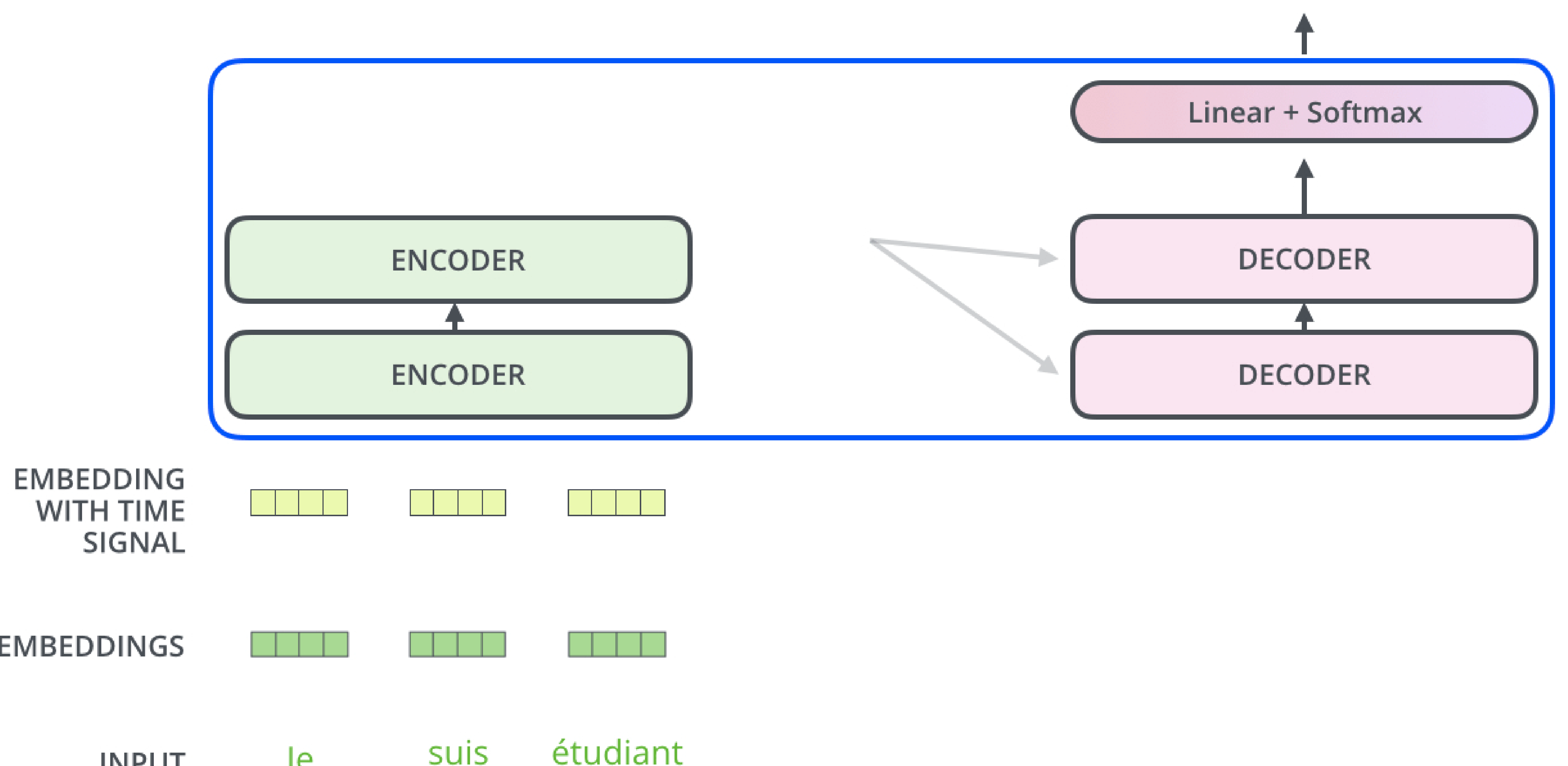


Encoder-Decoder Attention

Encoder Self-Attention

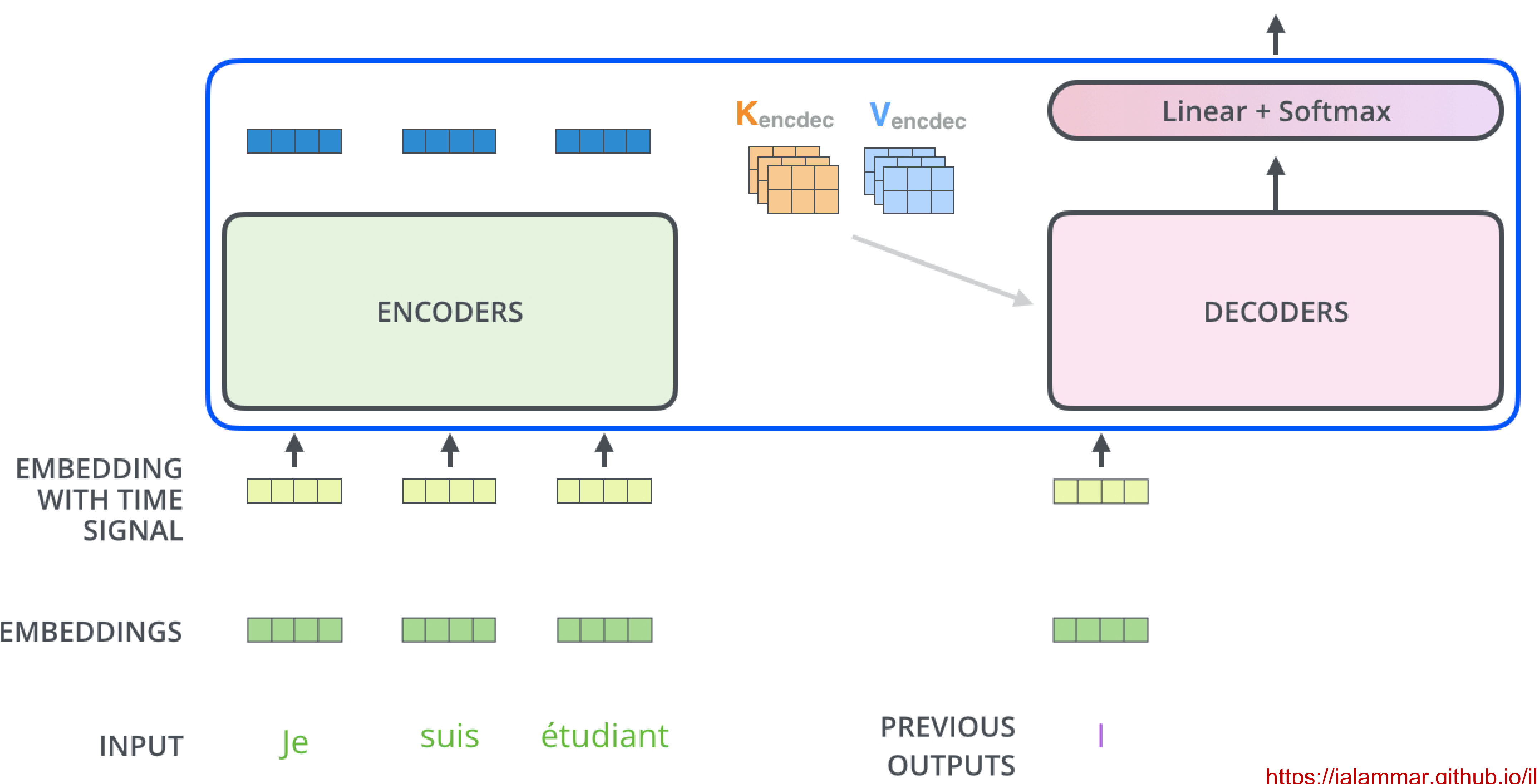


MaskedDecoder Self-Attention



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

Decoding time step: 1 2 3 4 5 6   OUTPUT I



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

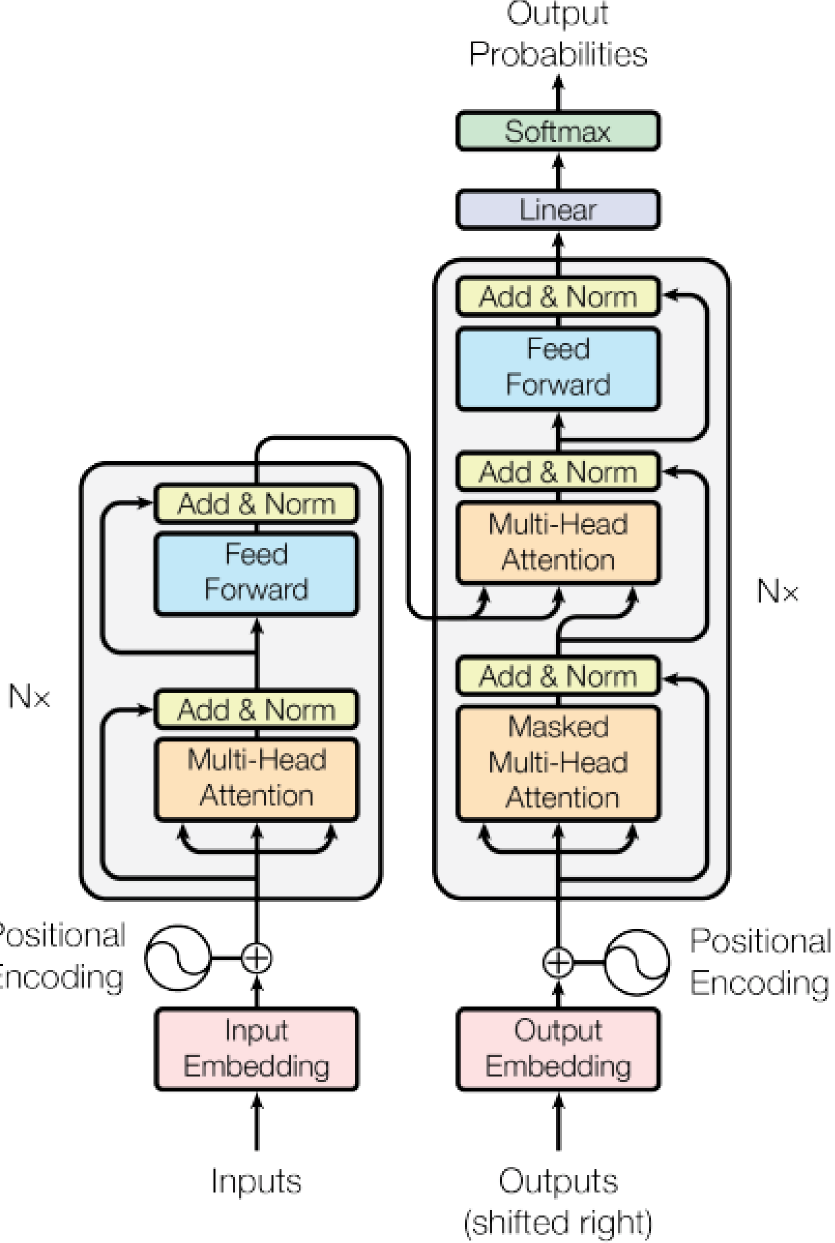


Figure 1: The Transformer - model architecture.

Attention is All you Need (2017) https://arxiv.org/pdf/1706.03762.pdf

Loss Function: cross-entropy (predicting translated word)

Training Time: ~4 days on (8) GPUs

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n = sequence length

d = length of representation (vector)

Q: Is the complexity of self-attention good?

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Important: when learning dependencies b/w words, you don't want long paths. Shorter is better.

Self-attention connects all positions with a constant # of sequentially executed operations, whereas RNNs require $O(n)$.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

https://arxiv.org/pdf/1706.03762.pdf

Machine Translation results: state-of-the-art (at the time)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Impact of Transformers

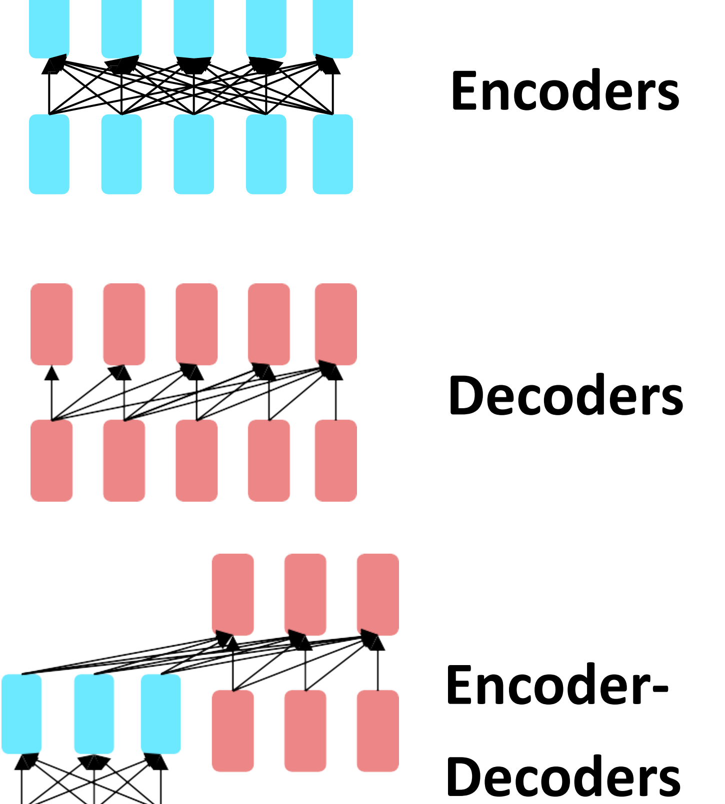
- Let to better predictive models of language ala GPTs!

Model	Layers	Heads	Perplexity
LSTMs (Grave et al., 2016)	-	-	40.8
QRNNs (Merity et al., 2018)	-	-	33.0
Transformer	16	16	19.8

["Efficient Content-Based Sparse Attention with Routing Transformers" Roy et al. 2020]

Impact of Transformers

- A building block for a variety of LMs



❖ **Examples:** BERT, RoBERTa, SciBERT.

❖ Captures bidirectional context. How do we pretrain them?

❖ **Examples:** GPT-2, GPT-3, Llama models, and many many more

❖ Other name: **causal or auto-regressive language model**

❖ Nice to generate from; can't condition on future words

❖ **Examples:** Transformer, T5, BART

❖ What's the best way to pretrain them?

Transformer LMs + Scale = LLMs

- 2 main dimensions:
- Model size, pretraining data size

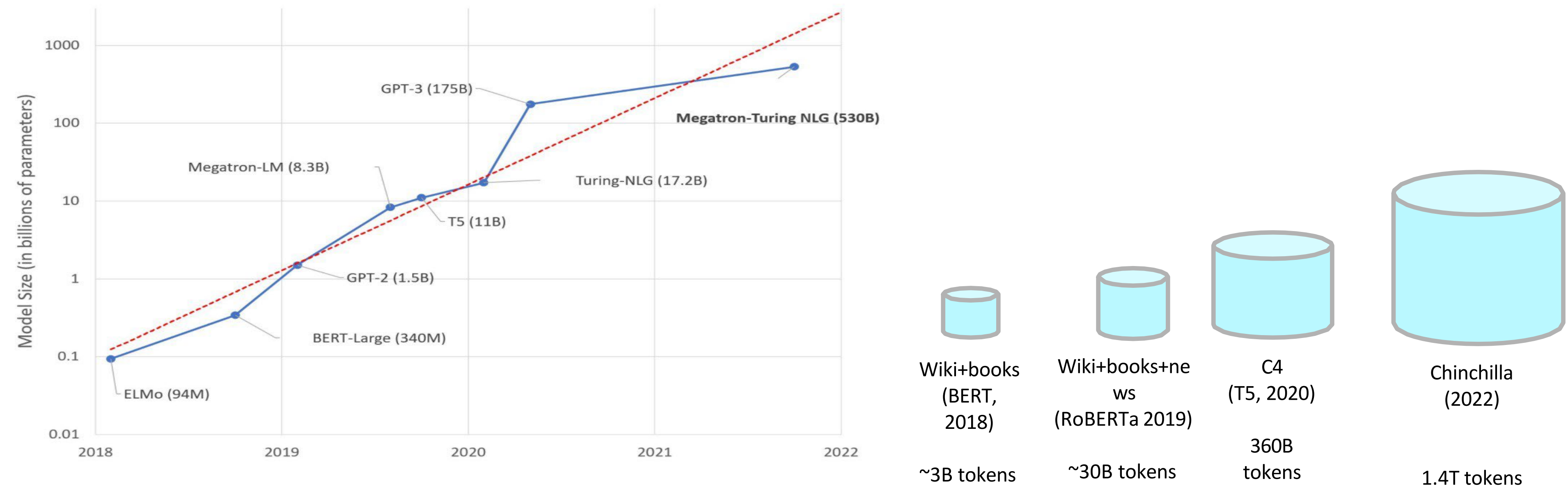
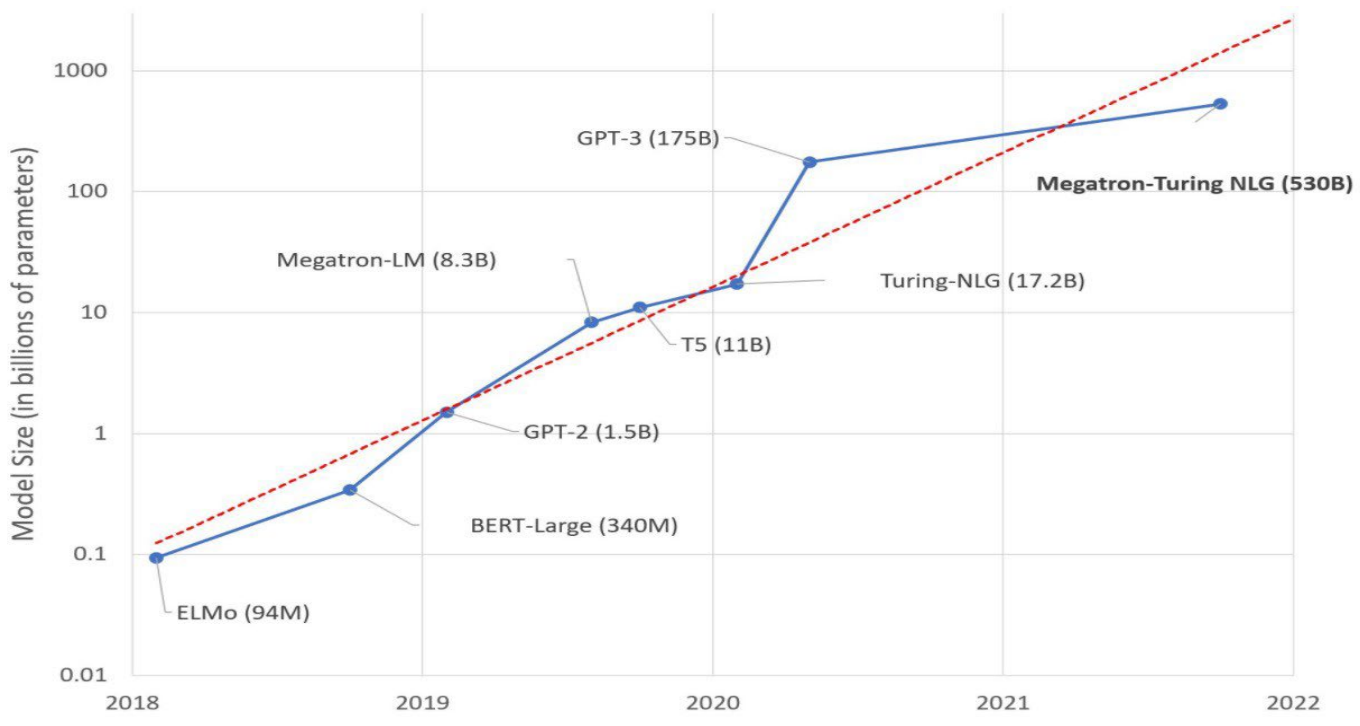


Photo credit: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

Large Language Models

- Not only they improved performance on many NLP tasks, but exhibited new capabilities



Transformers - Summary

- Self-attention + positional embedding + others = NLP go brr
- Much faster to train than any previous architectures, much easier to scale
- Perform on par or better than previous RNN based models
    - Ease of scaling allows to extract much better performance

- What if we don’t want to decode/translate?
- Just want to perform a particular task (e.g., classification)
- Want even more robust, flexible, rich representation!
- Want positionality to play a more explicit role, while not being restricted to a particular form (e.g., CNNs)

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

Outline

- Self-Attention
- Transformer Encoder
- Transformer Decoder
- BERT

BERT

Bidirectional Encoder Representations from Transformers

BERT

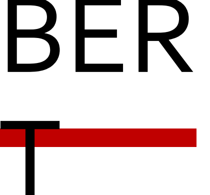
Bidirectional Encoder Representations from Transformers

Like *Bidirectional LSTMs*, let’s look in **both** directions

BERT

Bidirectional Encoder Representations from Transformers

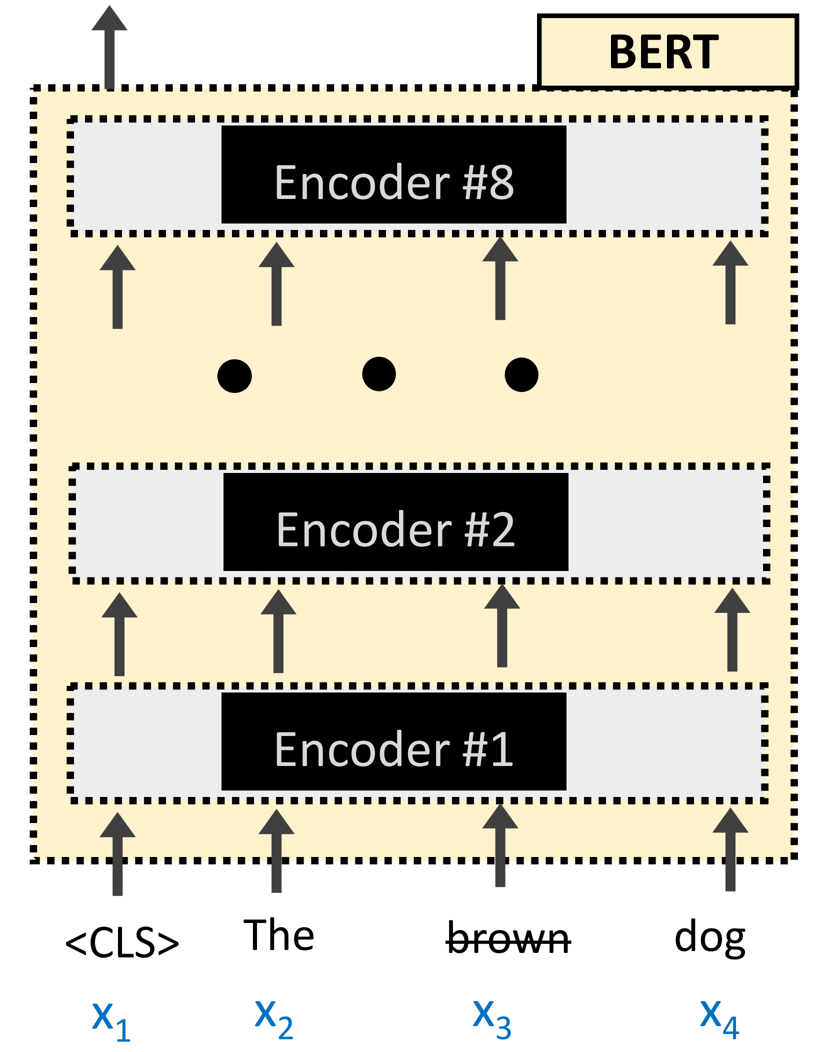
Let's only use Transformer *Encoders*, no Decoders



Bidirectional Encoder Representations from Transformers

It’s a language model that builds rich representations

BERT



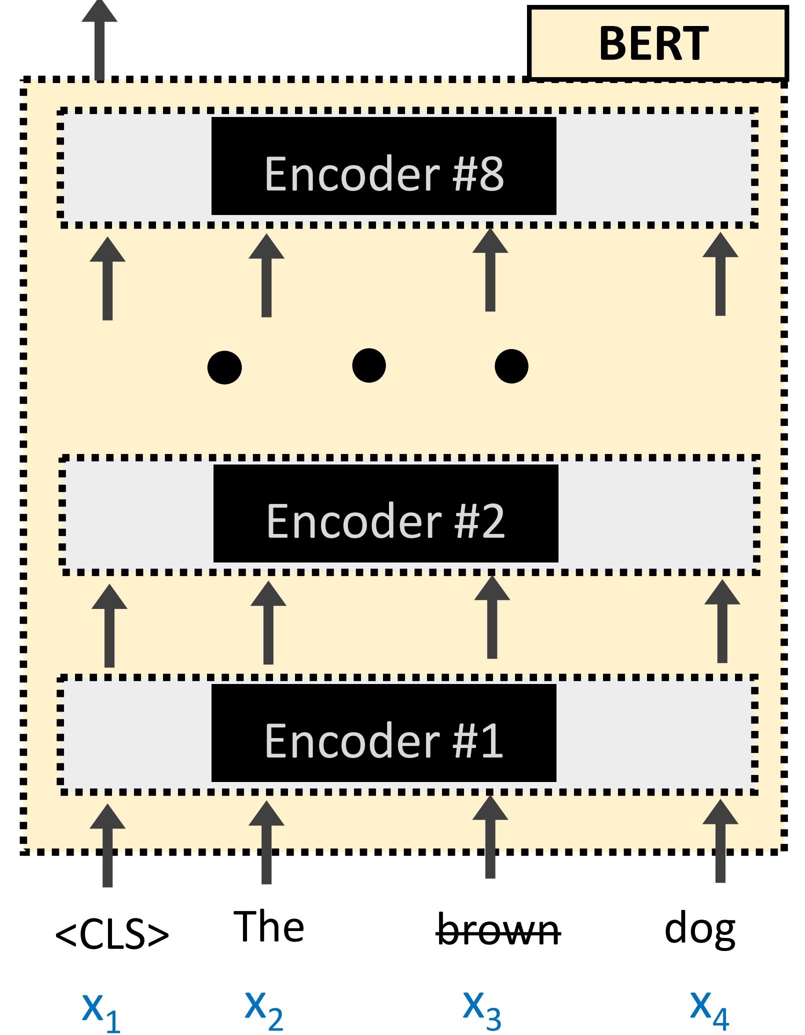
BERT has 2 training objectives:

1. Predict the **Masked word** (a la CBOW)

15% of all input words are randomly masked.

- 80% become [MASK]
- 10% become revert back
- 10% become are deliberately corrupted as wrong words

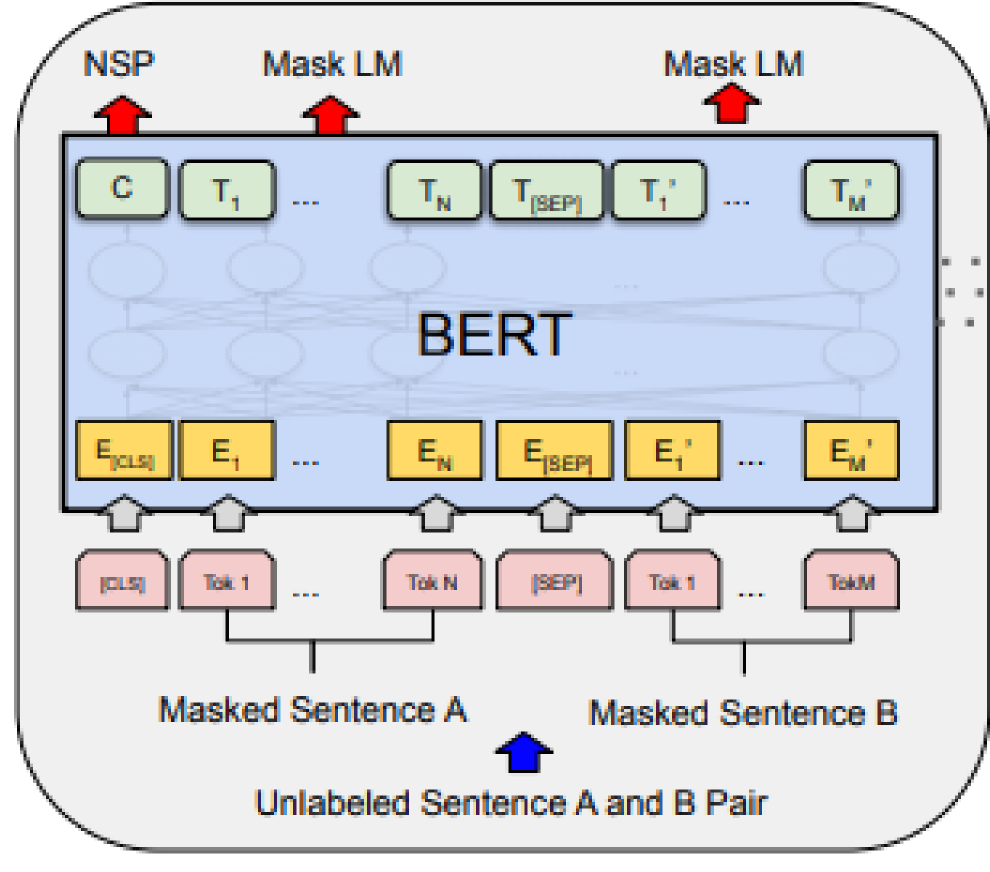
BERT



BERT has 2 training objectives:

2. Two sentences are fed in at a time. Predict the if the second sentence of input truly follows the first one or not.

BERT



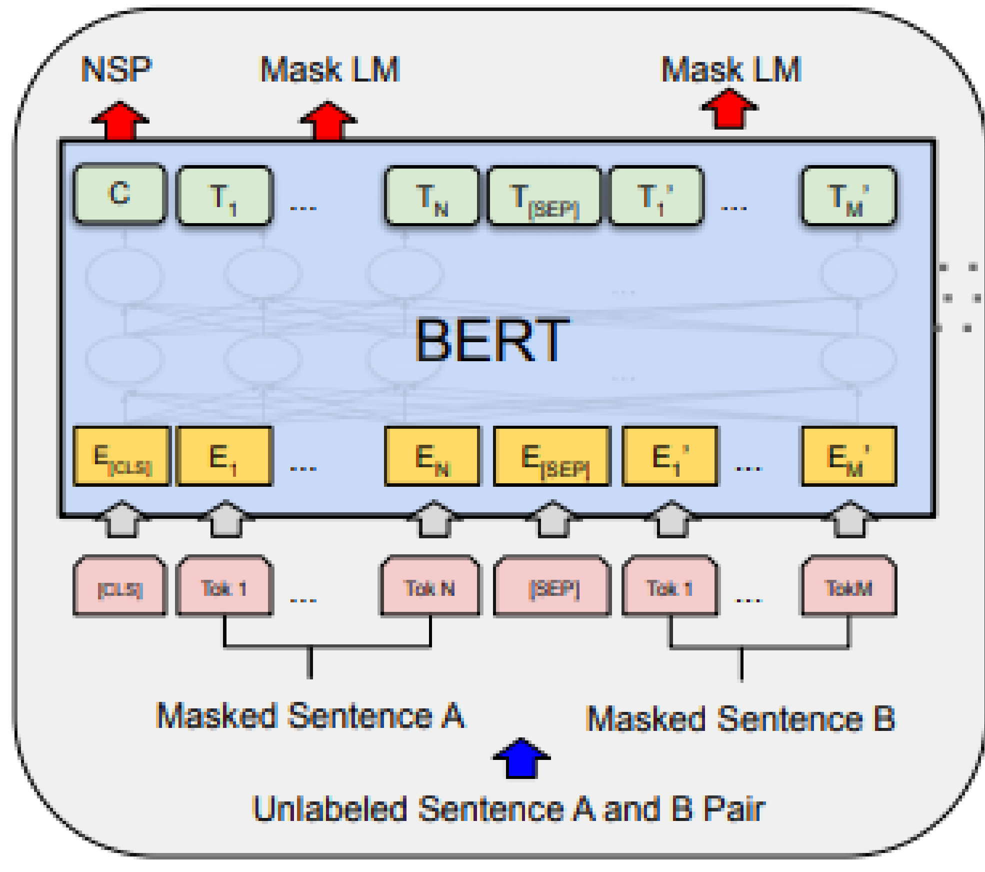
Every two sentences are separated by a **<SEP>** token.

50% of the time, the 2nd sentence is a randomly selected sentence from the corpus.

50% of the time, it truly follows the first sentence in the corpus.

Source: original BERT paper: https://arxiv.org/pdf/1810.04805.pdf

BERT



Pre-training

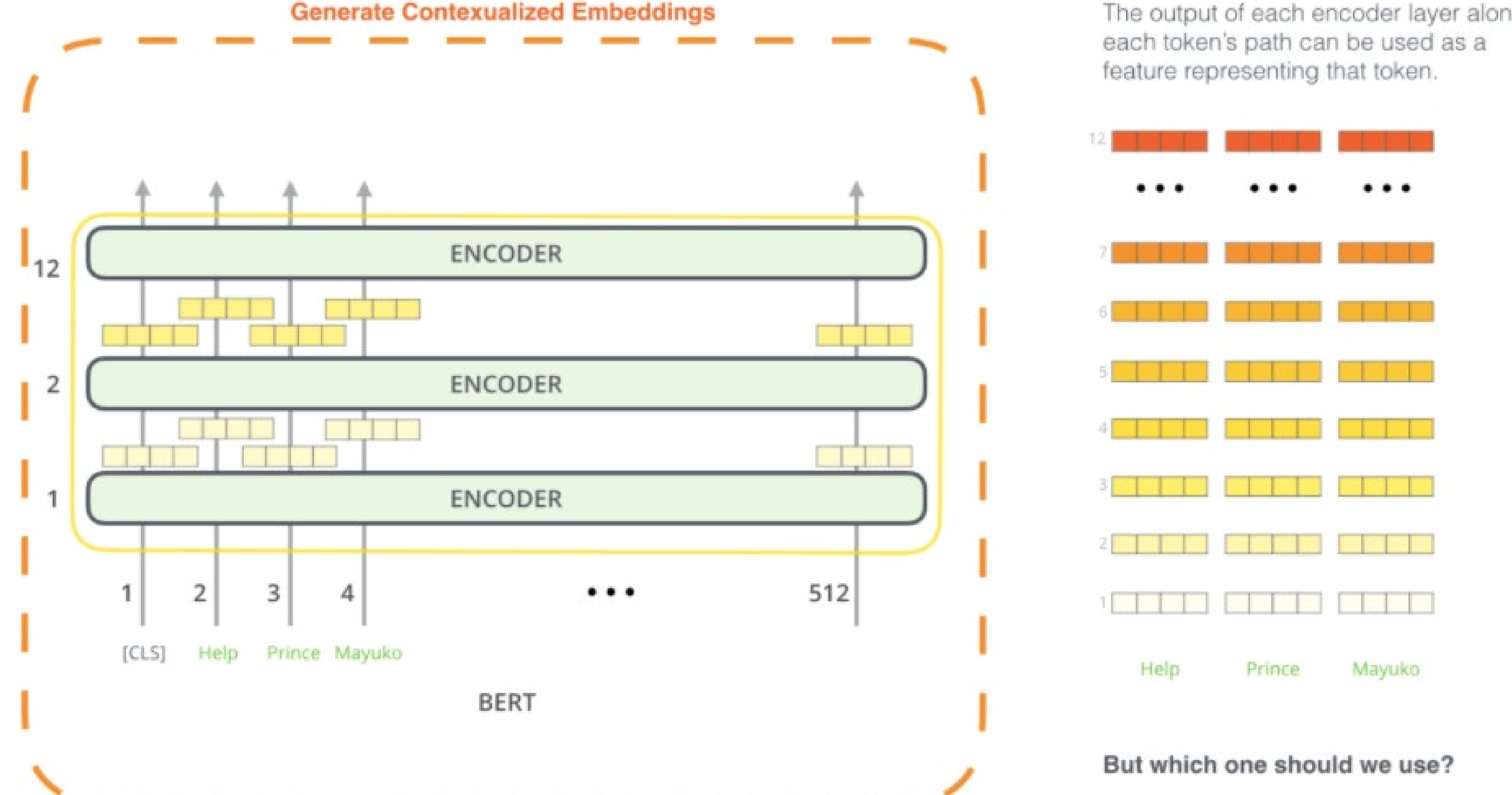
NOTE: BERT also embeds the inputs by their **WordPiece** embeddings.

WordPiece is a **sub-word tokenization** learns to merge and use characters based on which pairs maximize the likelihood of the training data if added to the vocab.

Source: original BERT paper: https://arxiv.org/pdf/1810.04805.pdf

BERT

One could extract the **contextualized embeddings**



Picture: https://jalammar.github.io/illustrated-bert/

Later layers have the best contextualized embeddings

12 ... 7 6 5 4 3 2 1

Help

		Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + ... + 2 + 1 =	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9 =	95.9
Concat Last Four Hidden	9 10 11 12	96.1

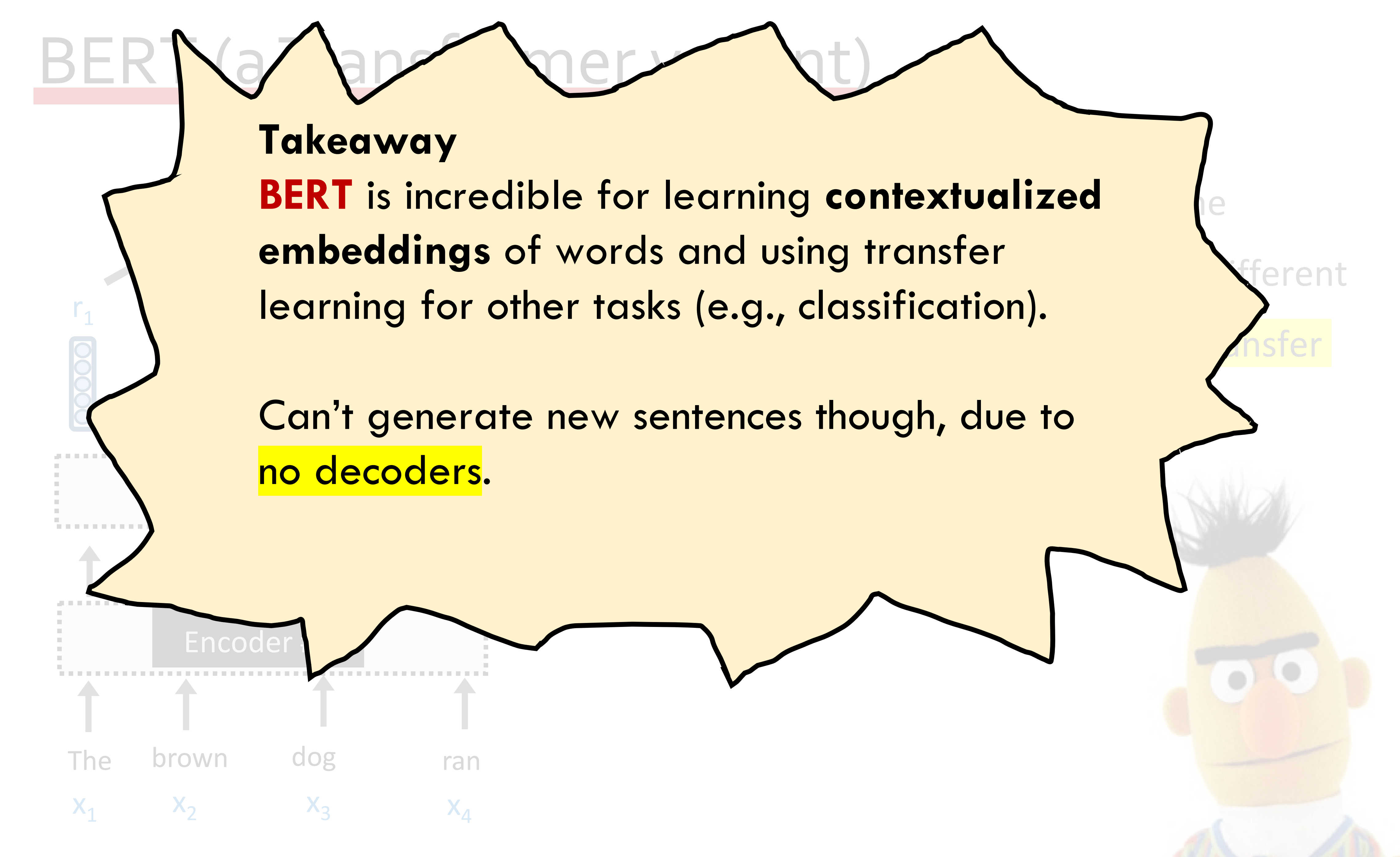
Picture: https://jalammar.github.io/illustrated-bert/

BERT yields state-of-the-art (SOTA) results on many tasks

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

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard).

Source: original BERT paper: https://arxiv.org/pdf/1810.04805.pdf


BERT (a Transformer variant)
Takeaway
BERT is incredible for learning contextualized embeddings of words and using transfer learning for other tasks (e.g., classification).
Can't generate new sentences though, due to no decoders.
Encoder
The
brown
dog
ran