TeIAS | Tehran Institute for Advanced Studies

10 December 2023

Attention Mechanism and Transformers

Machine Learning Course Fall 2022

Mahdi Zakizadeh (mahdizakizadeh.me@gmail.com)

Lecture Plan

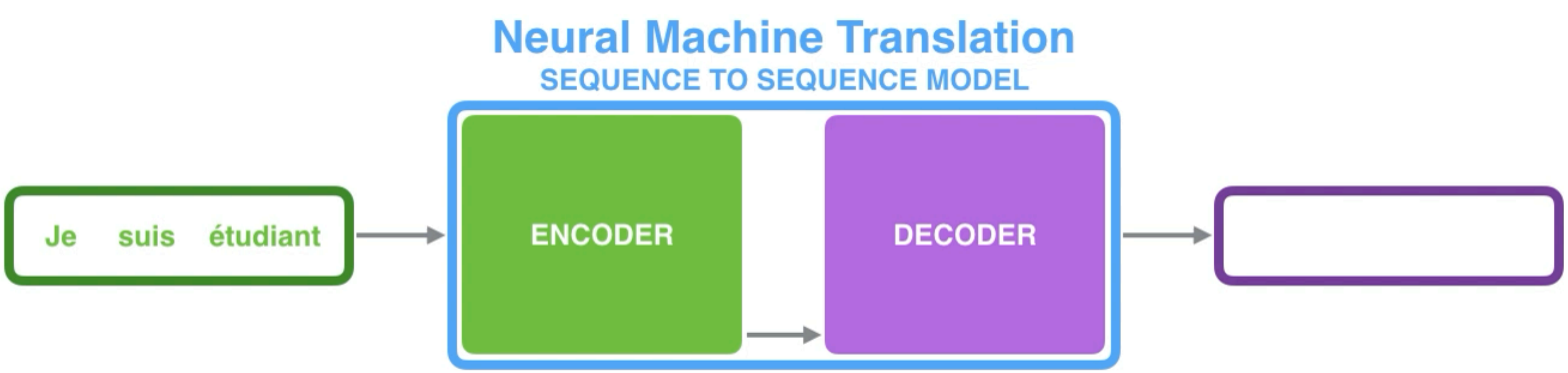
- Understand sequence-to-sequence (seq2seq) neural architecture
- How attention mechanism improved seq2seq models
- Understand the Transformer architecture
- Shortcomings of Transformer

Seq2Seq and Attention

Seq2Seq Architecture

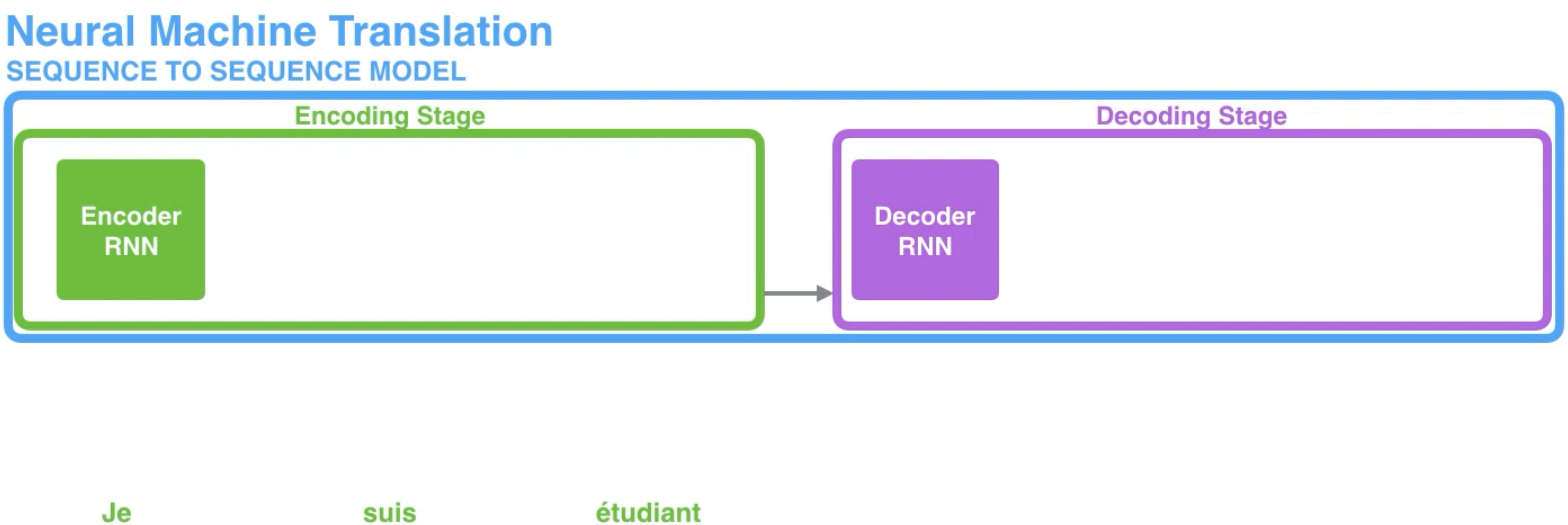
Motivation

- Both the input and the output is a sequence of data
  - Examples: Machine Translation
- Solution: Encoder-Decoder models



Seq2Seq Architecture

Closer look



Seq2Seq Architecture

The Problem

- Fixed source representation is suboptimal:

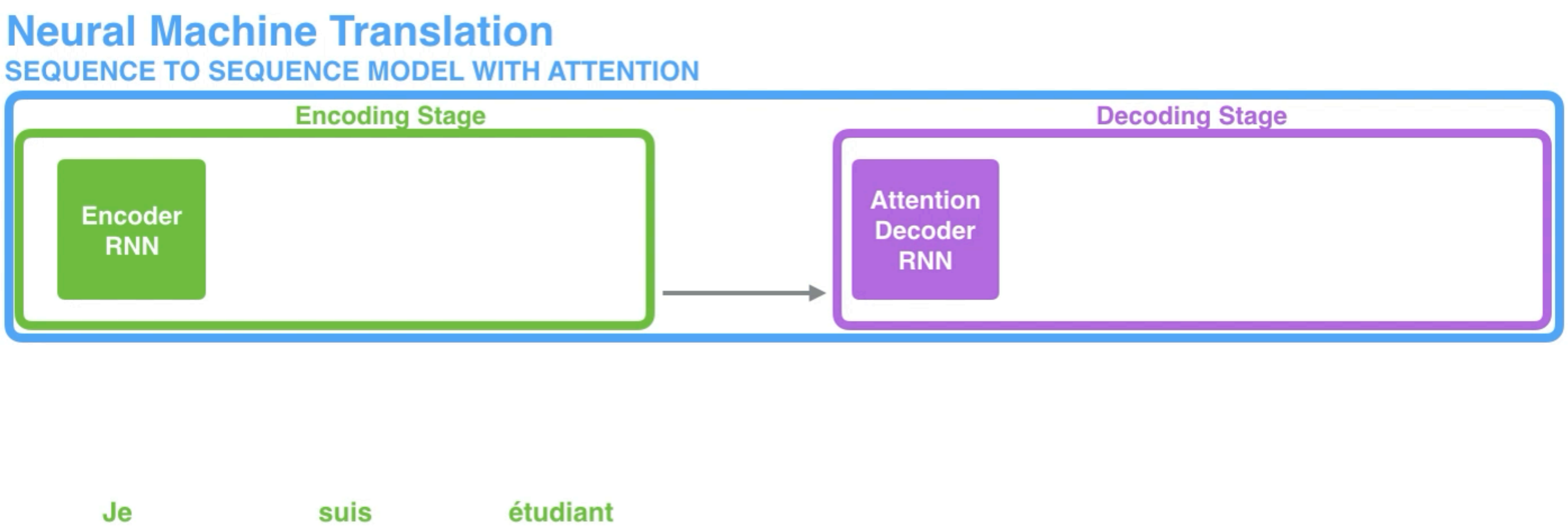
  1. It is hard to encode a sentence into a single vector

     - *Vanishing gradient problem*

  2. Different information may be relevant at different steps

Seq2Seq Architecture

Let’s Pay Attention Now



Seq2Seq Architecture

Let’s Pay Attention Now

Attention at time step 4

Seq2Seq Architecture

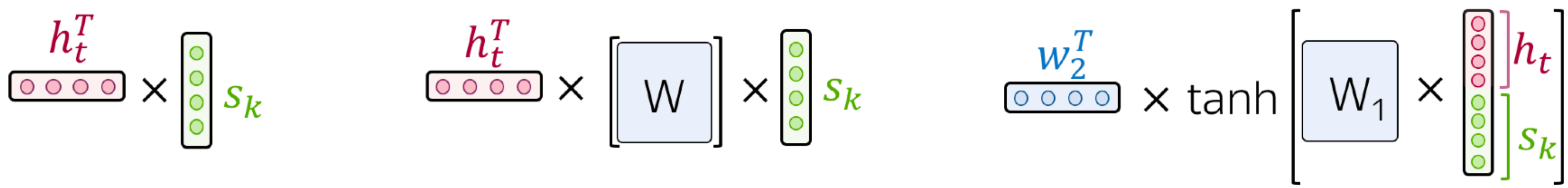
Computing Attention Output

- At each decoder state:
  - Inputs: $h_1, h_2, \ldots, h_m$ (Encoder states), d_t (Decoder state)
  - Compute attention scores: $score(d_t, h_k)$
  - Attention weights *(softmax)*: $$a_k^{(t)} = \frac{\exp\left(\text{score}\ (d_t, h_k)\right)}{\sum_{i=1}^{m} \exp\left(\text{score}\ (d_t, h_i)\right)}, \mathrm{k} = 1, \ldots, \mathrm{m}$$
  - Attention outputs (weighted sum): $$c^{(t)} = a_1^{(t)} h_1 + a_2^{(t)} h_2 + \cdots + a_m^{(t)} h_m = \sum_{k=1}^{m} a_k^{(t)} h_k$$

Seq2Seq Architecture

Attention Score Function

Dot-product | Bilinear | Multi-Layer Perceptron



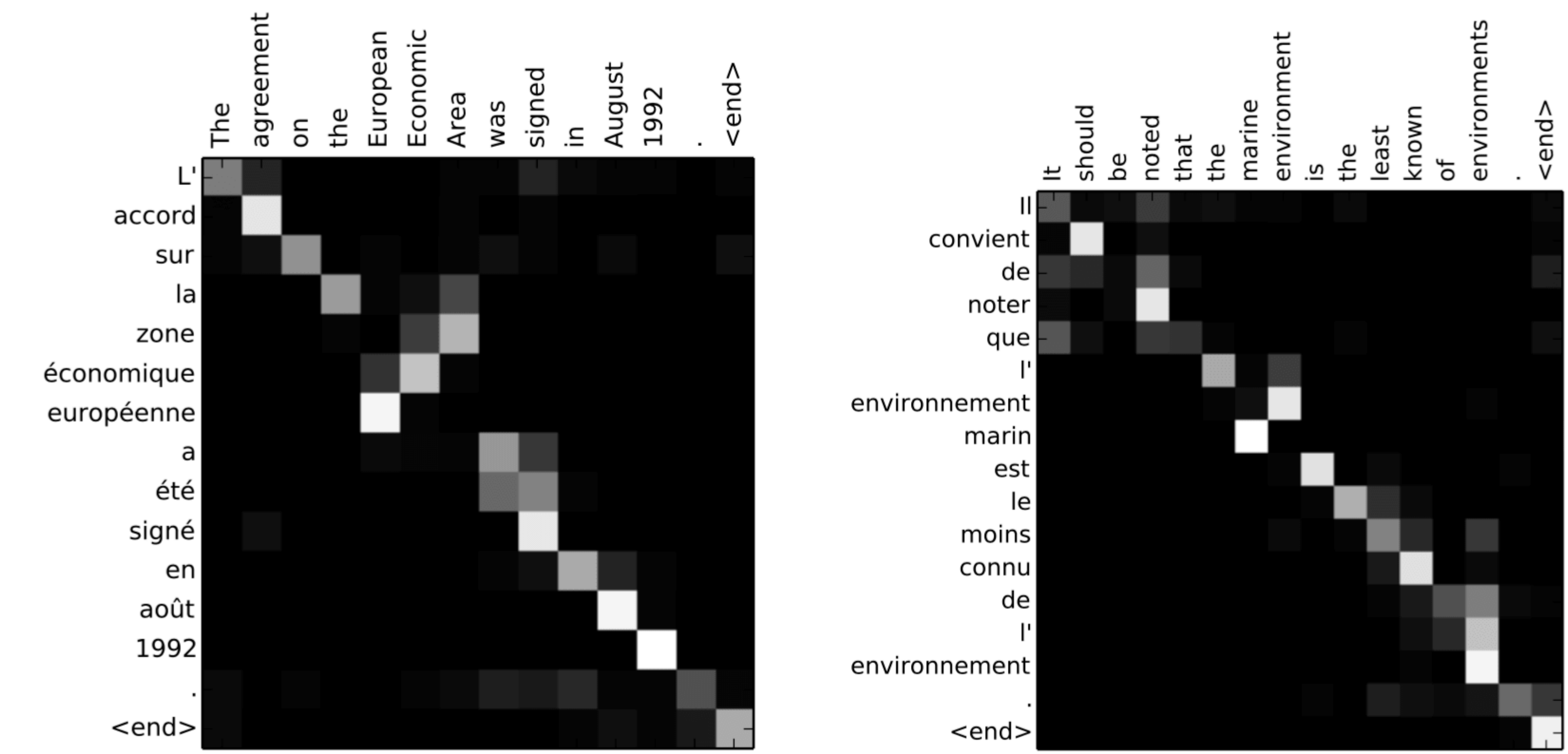
$\text{score}(h_t, s_k) = h_t^T\, s_k$

$\text{score}(h_t, s_k) = h_t^T\, W s_k$

$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1[h_t, s_k])$

Seq2Seq Architecture

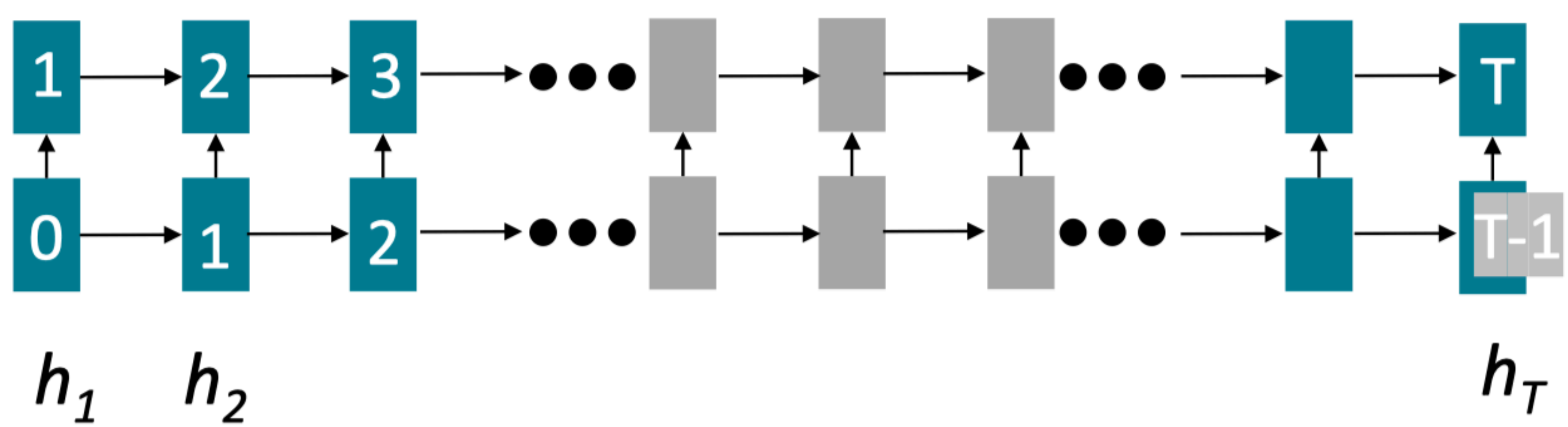
Attention Learns Alignment



Seq2Seq Architecture

The Problem

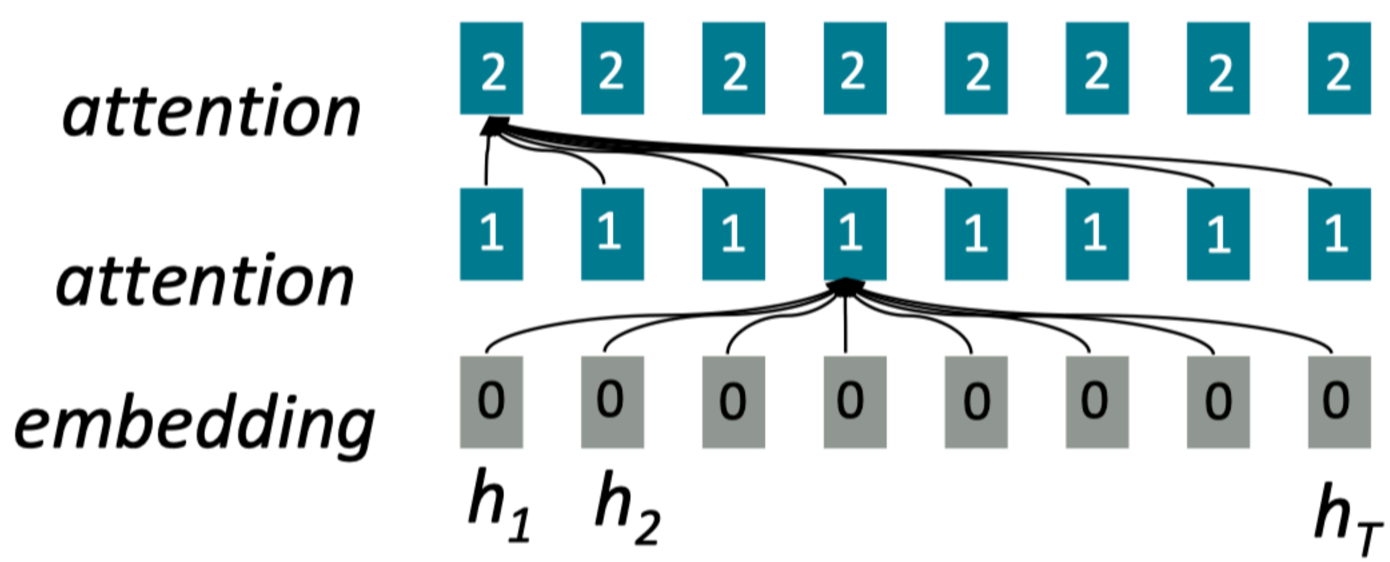
- Lack of parallelizability!
  - Forward and backward passes have **O(sequence length)** unparallelizable operations



Let’s make a building block out of Attention!

Attention as a Building Block

- Word representation → **Query** to access and incorporate information from a set of **Values**
- Attention in Seq2Seq models: From the **decoder** to **encoder**
- Self-attention: Compute attention within a sentence
- Number of unparallelizable operations does not increase sequence length
- Maximum interaction distance: **O(1)**, since all words interact at every layer



Attention as a Building Block

Self-Attention

- Attention operates on **queries**, **keys**, and **values**:
  - **Queries** $q_1, q_2, \ldots, q_T$ and $q_i \in \mathbb{R}^d$
  - **Keys** $k_1, k_2, \ldots, k_T$ and $k_i \in \mathbb{R}^d$
  - **Values** $v_1, v_2, \ldots, v_T$ and $v_i \in \mathbb{R}^d$
- In **self-attention,** the queries, keys, and values are drawn from the same source.

Attention as a Building Block

Self-Attention

- The self-attention operations are same as Seq2Seq architecture with attention:

Compute Key-Query affinities

$$e_{ij} = q_i^T k_j$$

Compute attention weights from affinities (Softmax)

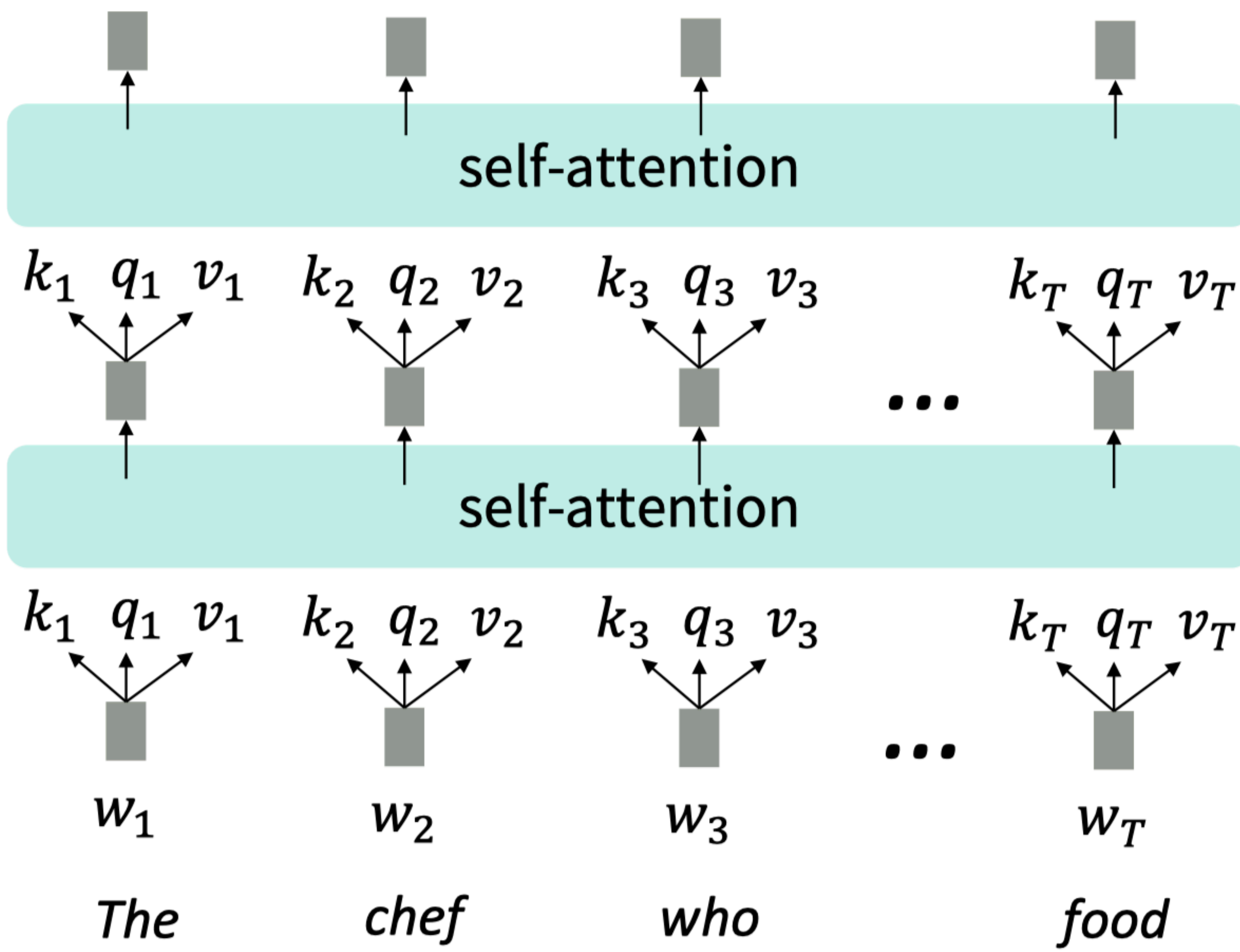
$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{j'} \exp\left(e_{ij'}\right)}$$

Compute outputs as weighted sum of values

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

Attention as a Building Block

The Problem?



Attention as a Building Block

Problems and Solutions

Problem	Solution
Doesn’t have an inherent notion of order!	

Attention as a Building Block

Problem: Sequence Order

- Self-attention doesn't know the order of its inputs
- We need to encode the order of the sentence in our keys, queries, and values
- Consider representing each **sequence index** as a **vector**

$p_i \in \mathbb{R}^d$, for $i \in \{1,2,\ldots,T\}$ are position vectors

- Let's add the p_i to our inputs. Let $\tilde{v}_i$, $\tilde{k}_i$ and $\tilde{q}_i$ be our old values:

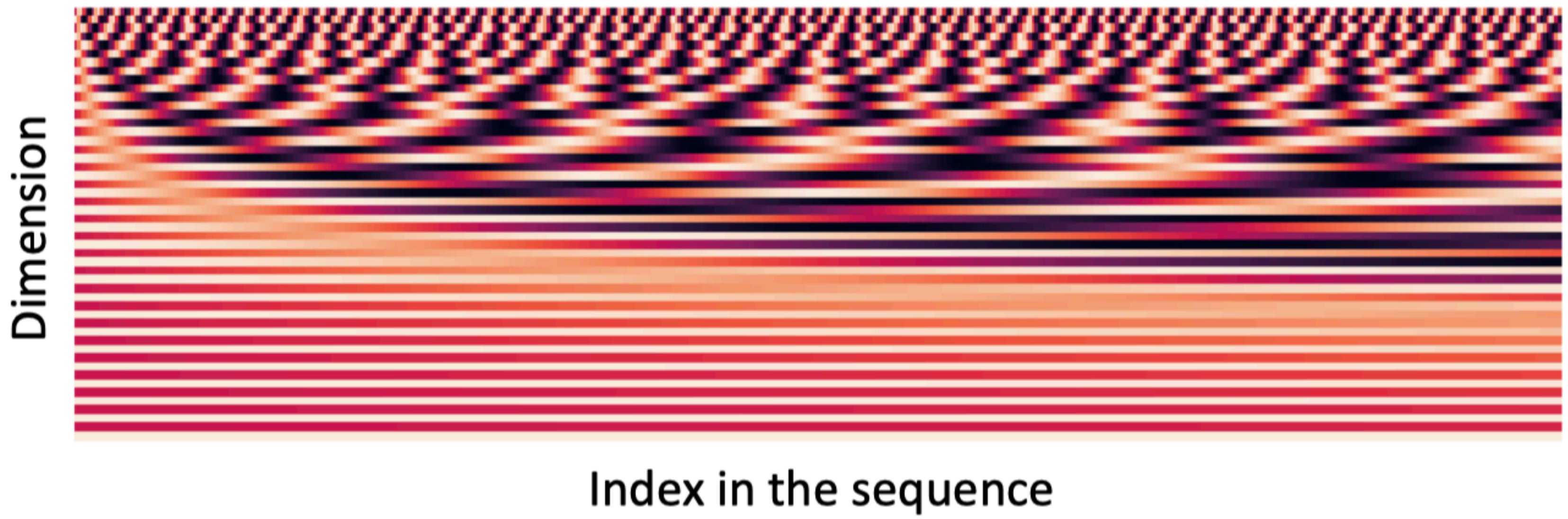
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

Attention as a Building Block

Problem: Sequence Order

- **Sinusoidal position representation:**

$$p_i = \begin{pmatrix} \sin\left(i/10000^{2*1/d}\right) \\ \cos\left(i/10000^{2*1/d}\right) \\ \vdots \\ \sin\left(i/10000^{2*\frac{d}{2}/d}\right) \\ \cos\left(i/10000^{2*\frac{d}{2}/d}\right) \end{pmatrix}$$



- **Pros:**
  - Periodicity indicates that maybe “absolute position” isn’t as important
  - Maybe can extrapolate to longer sequences as periods restart!
- **Cons:**
  - Not learnable; also the extrapolation doesn’t really work!

Attention as a Building Block

Problem: Sequence Order

- **Learnable position representation:**
  - Let all p_i be learnable parameters
  - Learn a matrix $p \in \mathbb{R}^{d \times T}$ and let p_i be a column of that matrix
  - **Pros:**
    - Flexibility: each position gets to be learned to fit the data
  - **Cons:**
    - Definitely can't extrapolate to indices outside $1, \ldots, T$.
  - Most systems use this!

  - Sometimes people try more flexible representations of position:
    - Relative linear position attention [Shaw et al., 2018]
    - Dependency syntax-based position [Wang et al., 2019]

Attention as a Building Block

Problems and Solutions

Problem	Solution
Doesn't have an inherent notion of order!	Add position representations to the input
No nonlinearities for deep learning! It's all just weighted averages	

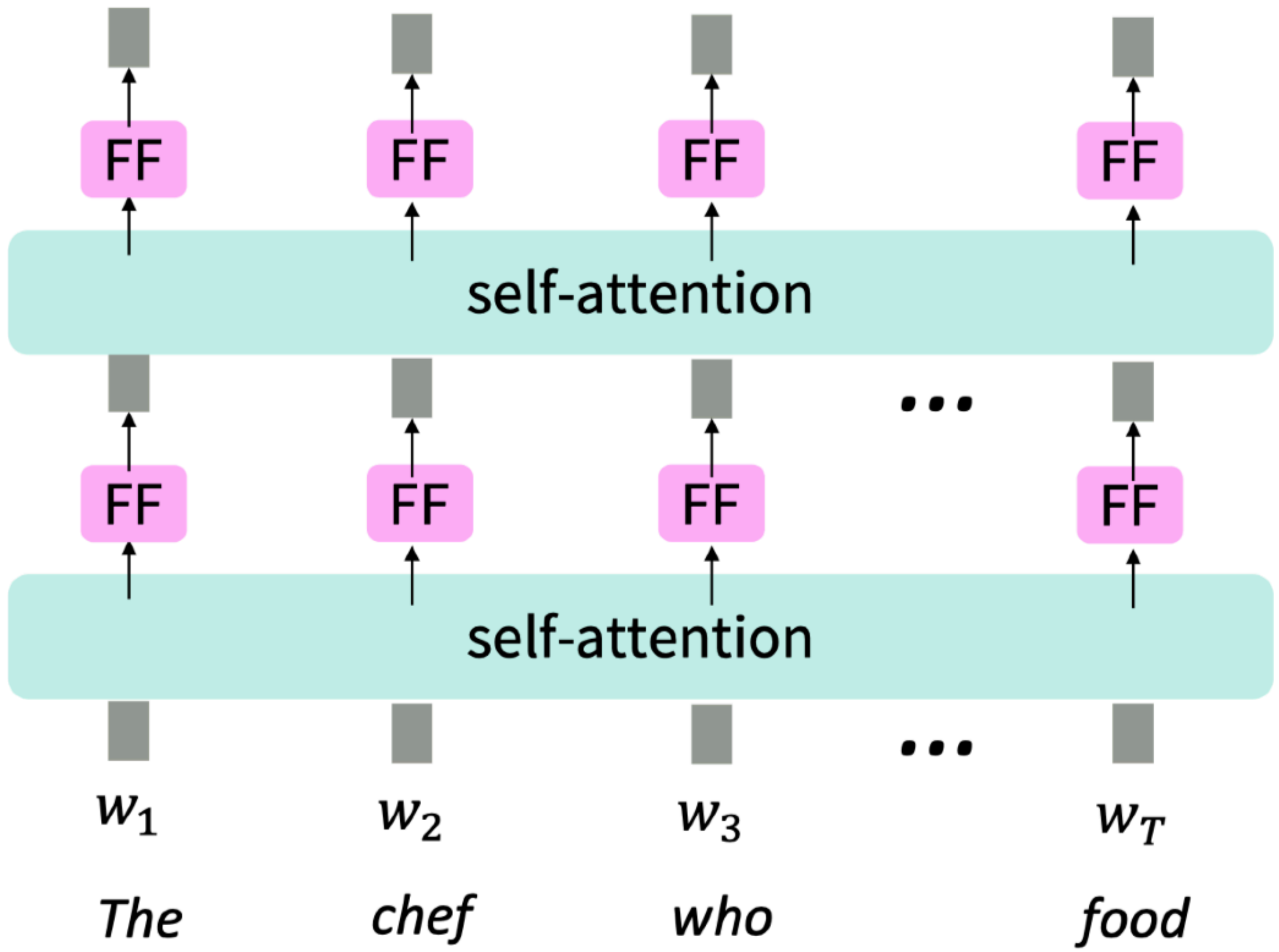
Attention as a Building Block

Problem: No Nonlinearities

- Stacking more self-attention layers just re-averages value vectors!

- Fix: Add feed-forward networks to post-process each output vector

- $m_i = MLP\left(\text{output}_i\right)$
  $= W_2 * \mathrm{ReLU}\left(W_1 \times \text{output}_i + b_1\right) + b_2$



Attention as a Building Block

Problems and Solutions

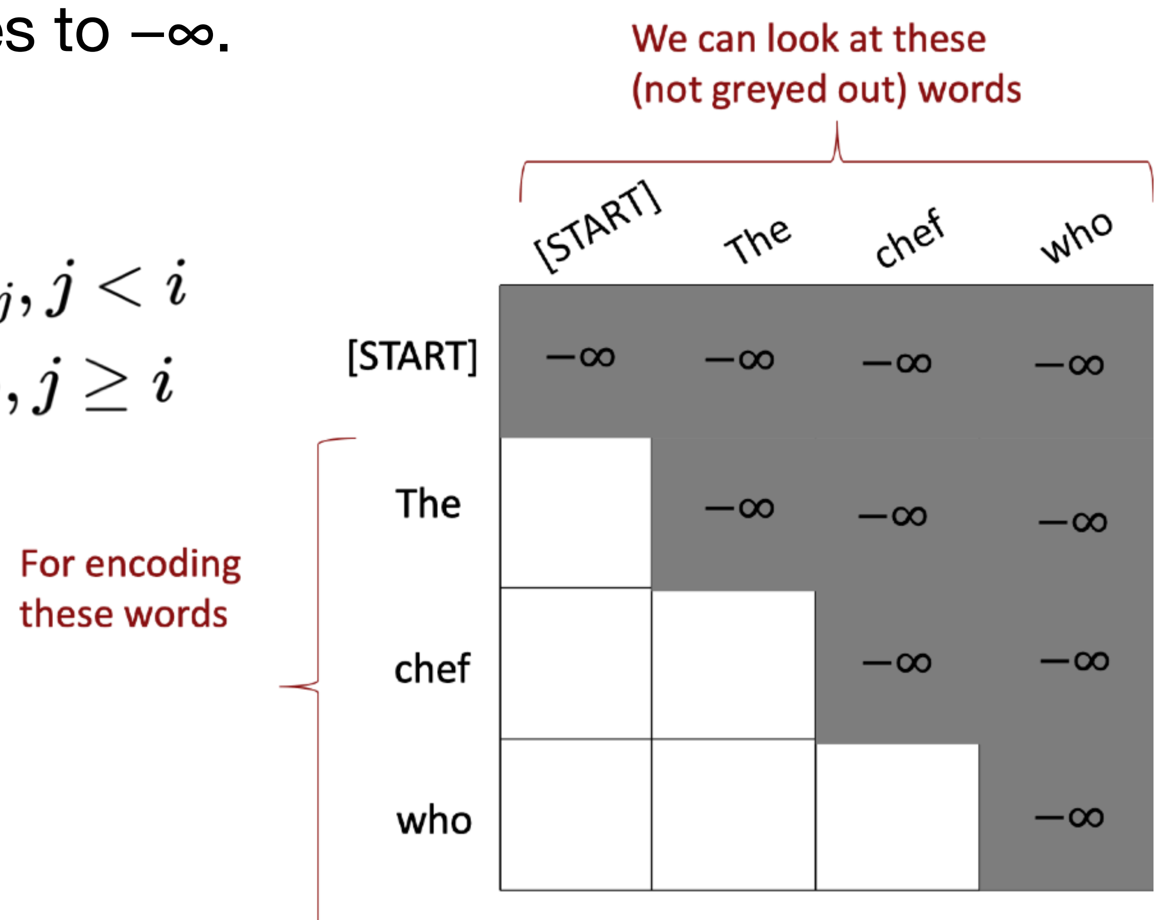
Problem	Solution
Doesn’t have an inherent notion of order!	Add position representations to the input
No nonlinearities for deep learning! It’s all just weighted averages	Apply the same feedforward network to each self-attention output.
We need to ensure we can’t peek at the future when predicting the next token.	

Attention as a Building Block

Problem: “Don’t Look Ahead” in Decoders

- We need to ensure we can’t peek at the future when predicting the next token
- At every timestep, we could change the set of keys and queries to include only past words. (Inefficient!)
- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Attention as a Building Block

Problems and Solutions

Problem	Solution
Doesn’t have an inherent notion of order!	Add position representations to the input
No nonlinearities for deep learning! It’s all just weighted averages	Apply the same feedforward network to each self-attention output.
We need to ensure we can’t peek at the future when predicting the next token.	Mask out the future by artificially setting attention weights to 0!

Attention as a Building Block

Review

- **Self-Attention:**
  - The basis of the method
- **Position Representation:**
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities:**
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking:**
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.

That’s it! But this is not the **Transformer** model we’ve been hearing about.

The Transformer

The Transformer Encoder-Decoder

[Vaswani et al., 2017]

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

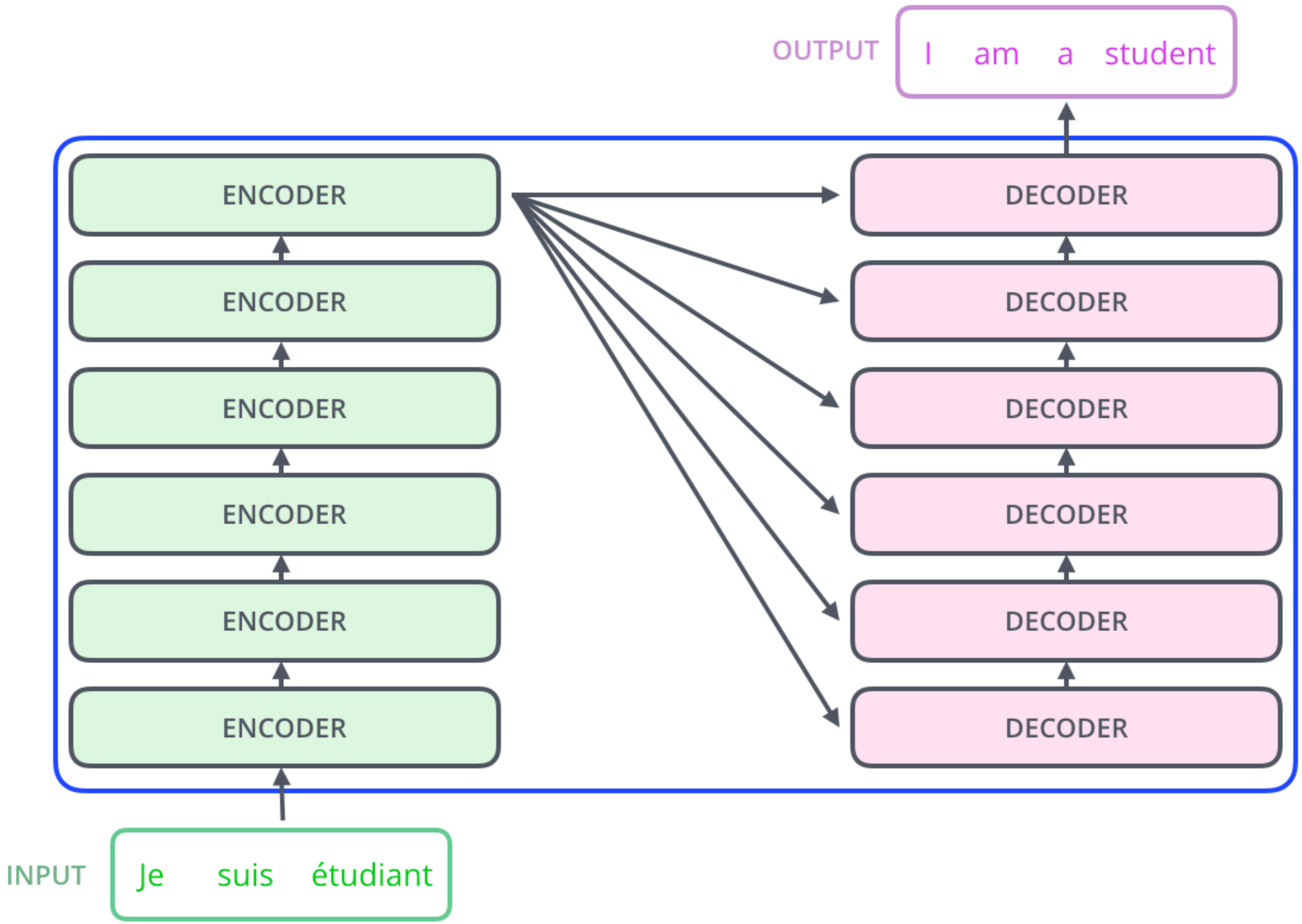
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

The Transformer Encoder-Decoder



The Transformer Encoder-Decoder

More Details!

1. **Key-Query-Value attention:** How to get k, q and v vectors from word embeddings
2. **Multi-headed attention:** Attend to multiple places in a single layer
3. **Tricks to help training:**
   - Residual connections
   - Layer normalization
   - Scaling the dot product
   - These tricks don't improve what the model is able to do; they help improve the training process. Both of these types of modeling improvements are very important!

The Transformer Encoder-Decoder

Key-Query-Value Attention

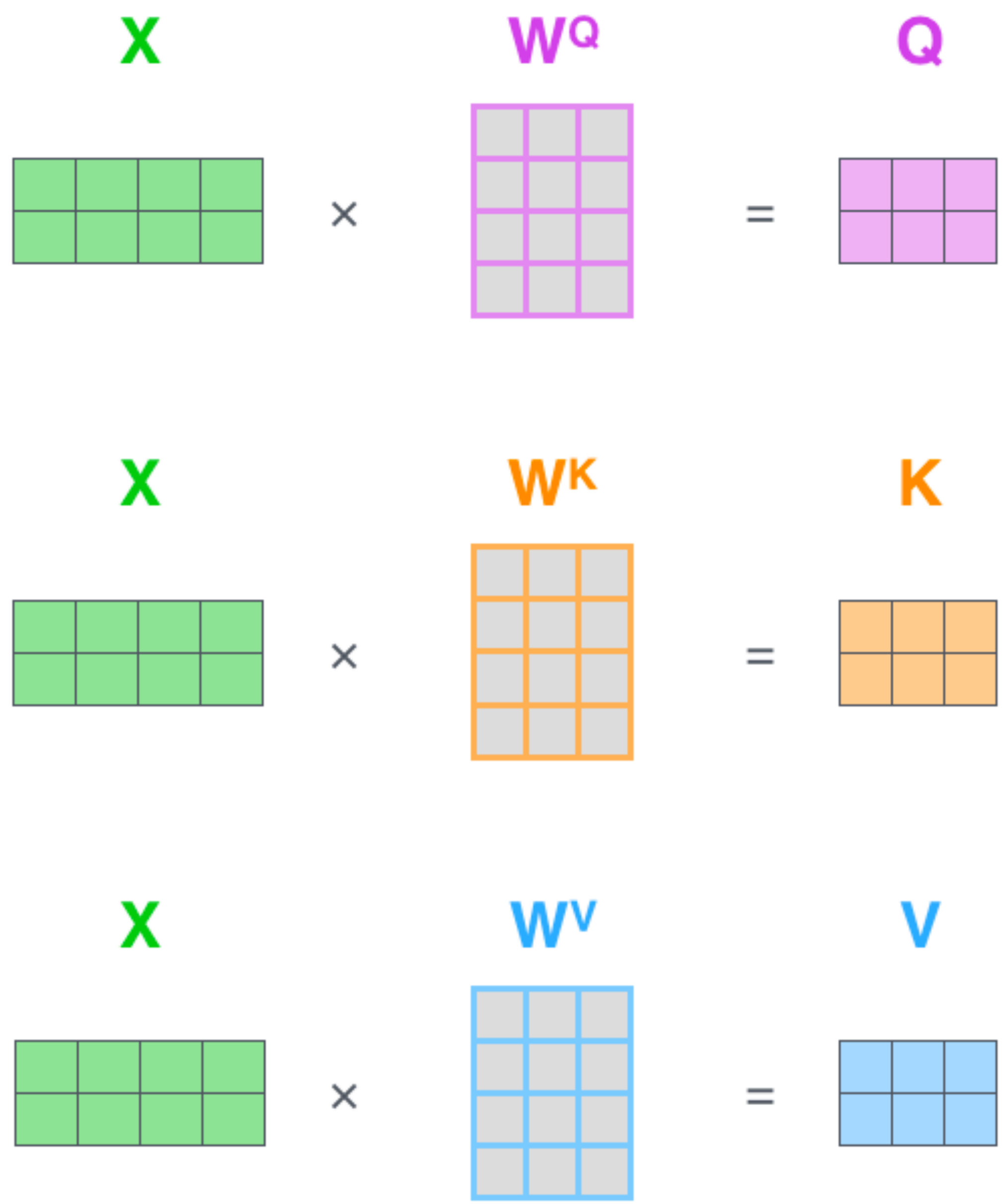
- We saw that the k, q and v vectors come from the same source
  - The Transformer does this in a particular way
  - Let $x_1, \ldots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^d$

- The keys, queries, and values are:
  - $k_i = W^K x_i$, where $W^K \in \mathbb{R}^{d \times d}$ is the key matrix
  - $q_i = W^Q x_i$, where $W^Q \in \mathbb{R}^{d \times d}$ is the query matrix
  - $v_i = W^V x_i$, where $W^V \in \mathbb{R}^{d \times d}$ is the key matrix

- These matrices allow *different* aspects of the x vectors to be used/emphasized in each of the three roles

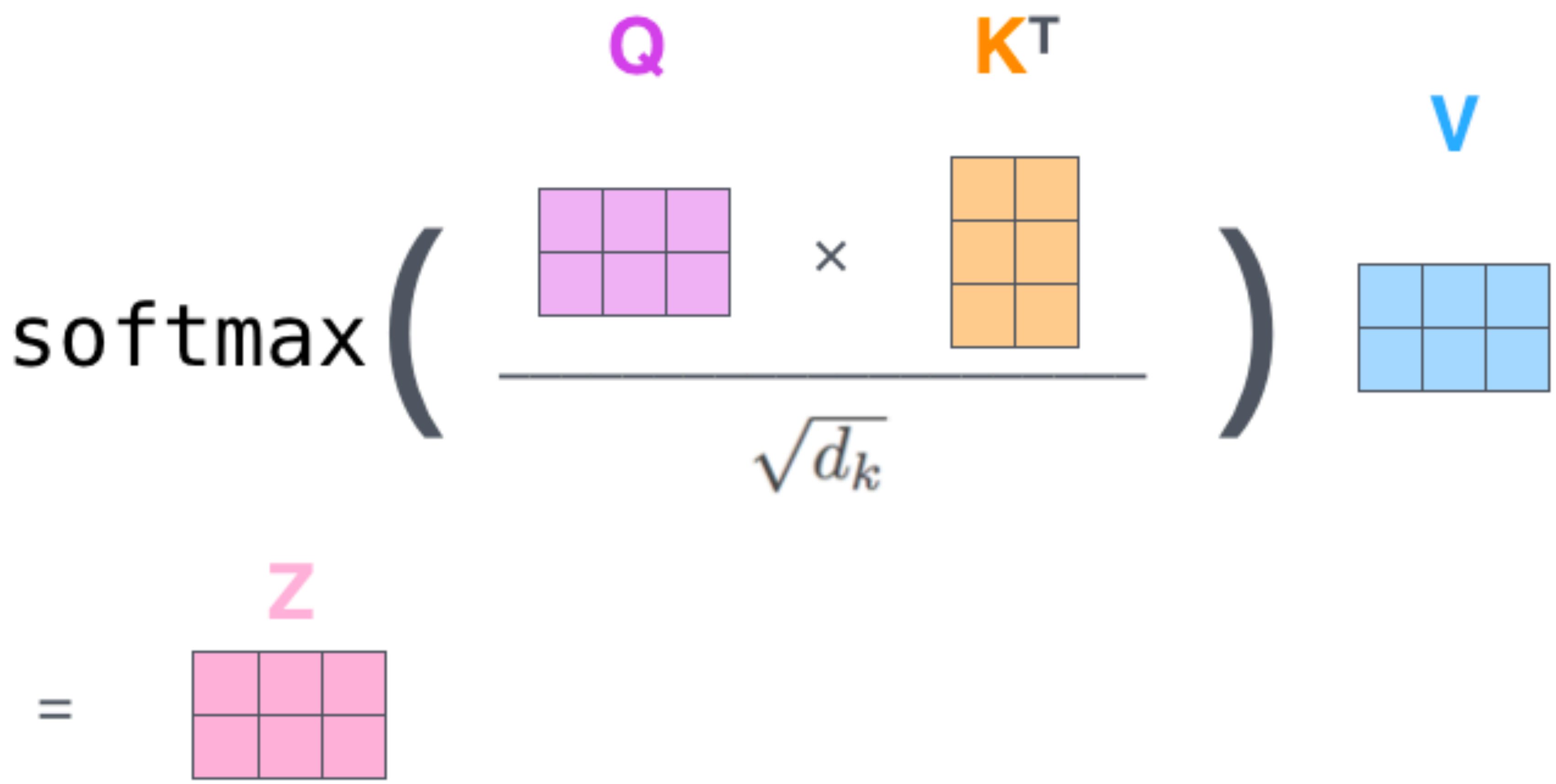
The Transformer Encoder-Decoder

Key-Query-Value Attention



The Transformer Encoder-Decoder

Key-Query-Value Attention



The Transformer Encoder-Decoder

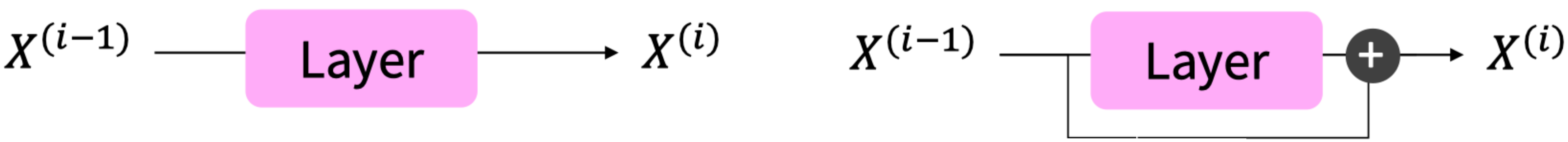
Multi-headed Attention

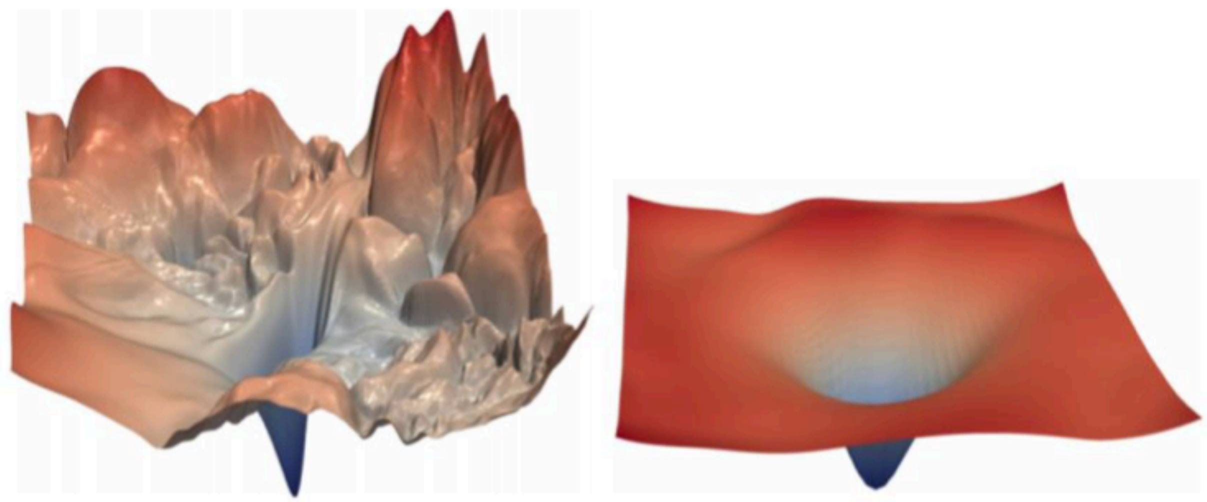
- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_j^T Q^T K x_j$ j is high, but maybe we want to focus on different j for different reasons?
- We'll define multiple attention "heads" through multiple W^Q, W^K, and W^V matrices
- Let, $W_l^K, W_l^Q, W_l^V \in \mathbb{R}^{d \times \frac{d}{h}}$ where h is the number of attention heads

The Transformer Encoder-Decoder

Residual Connections

- Residual connections are a trick to help models train better.





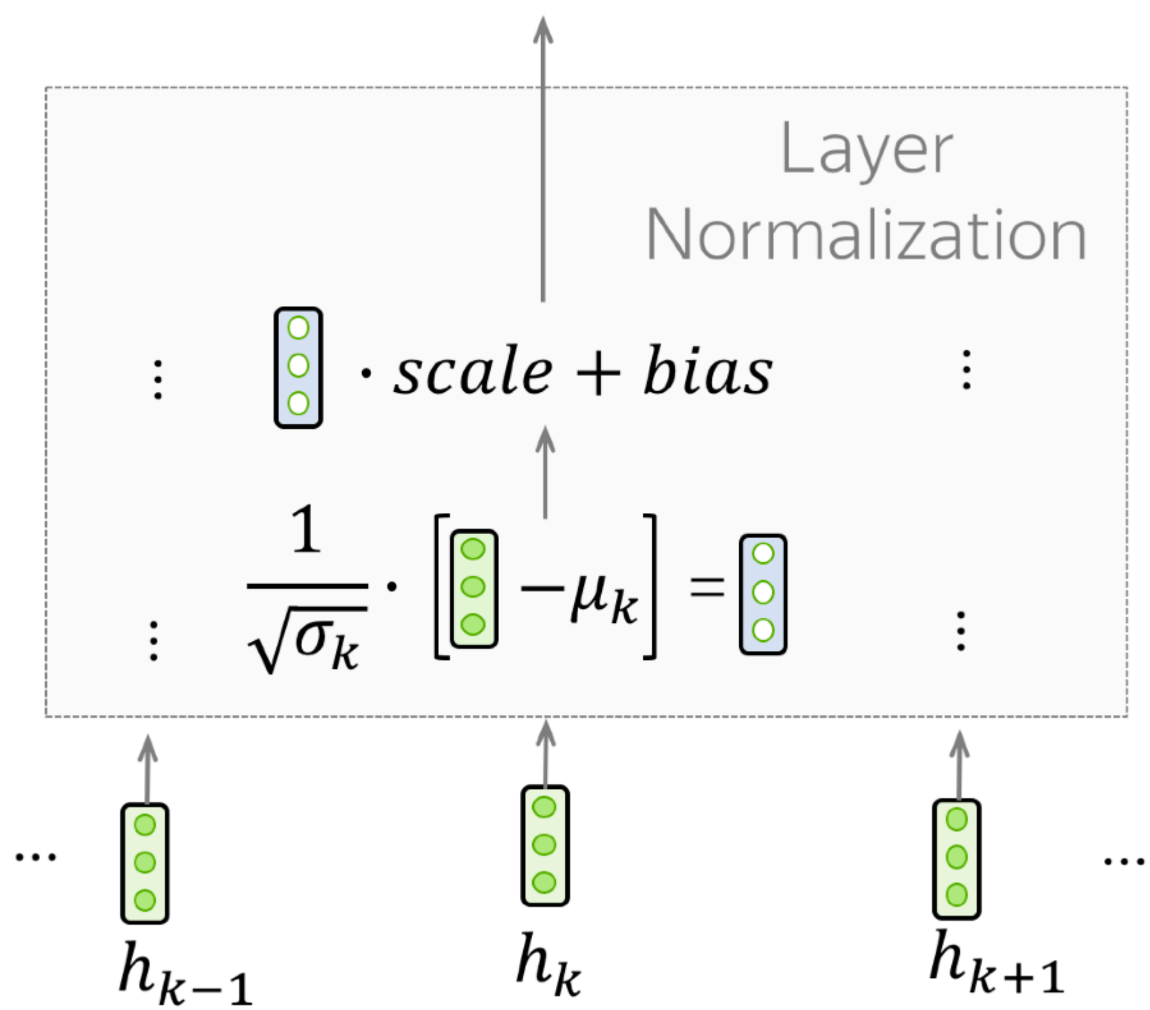
[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

The Transformer Encoder-Decoder

Layer Normalization

- Independently normalizes vector representation of each example in batch
- Improves convergence stability and sometimes even quality



The Transformer Encoder-Decoder

Scaled Dot Product

- When dimensionality d becomes large, dot products between vectors tend to become large.
- Because of this, inputs to the softmax function can be large, making the gradients small.
- Consequently instead of:

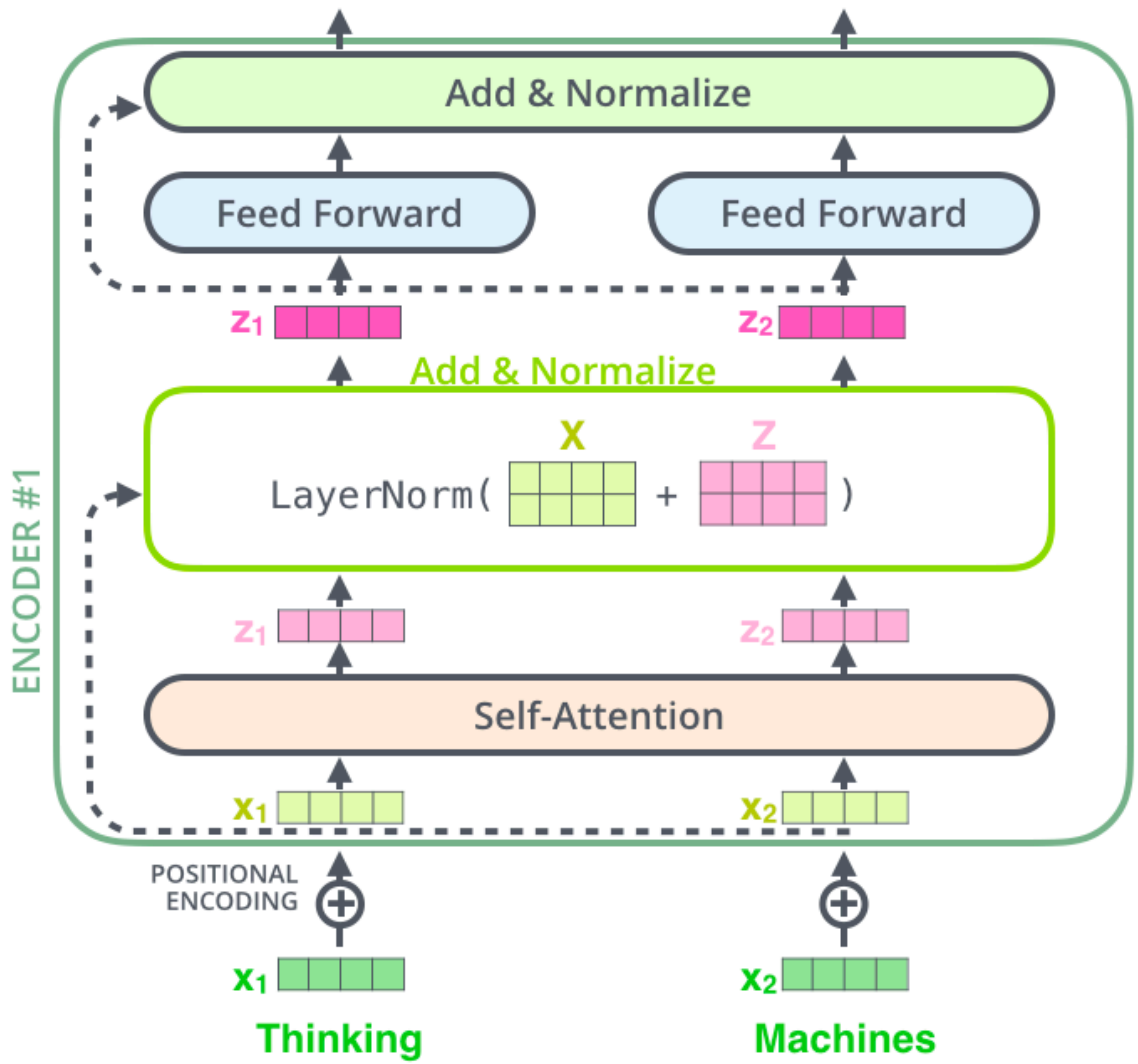
$$\text{output}_{\ell} = \text{softmax}\left(Q_\ell K_\ell^\top\right) * V_\ell$$

- We divide the attention scores by $\sqrt{\frac{d}{h}}$, to stop the scores from becoming large just as a function of $\frac{d}{h}$ (The dimensionality divided by the number of heads.)

$$\text{output}_{\ell} = \text{softmax}\left(\frac{Q_\ell K_\ell^\top}{\sqrt{\frac{d}{h}}}\right) * V_\ell$$

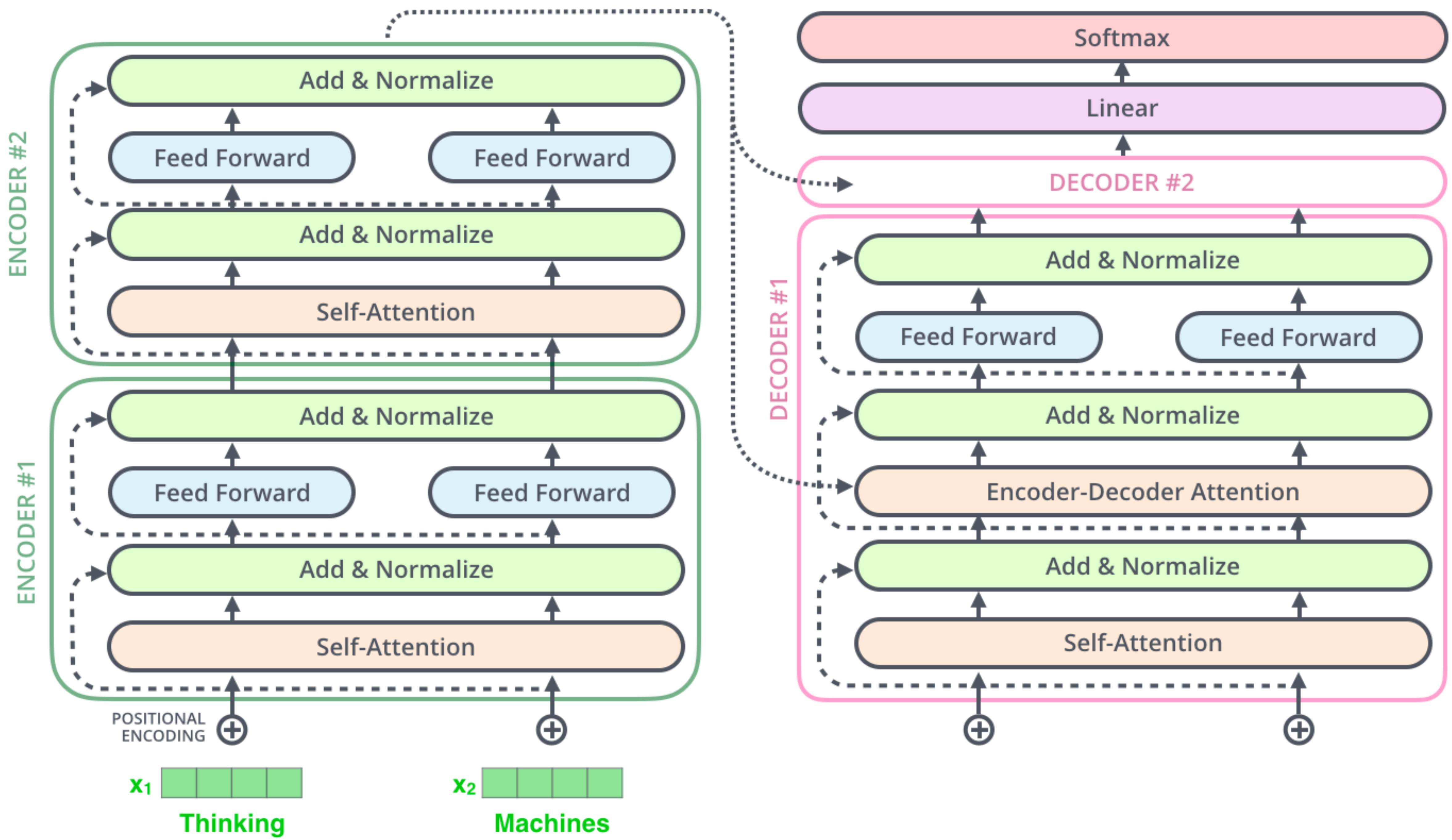
The Transformer Encoder-Decoder

Final Result - Encoder Block



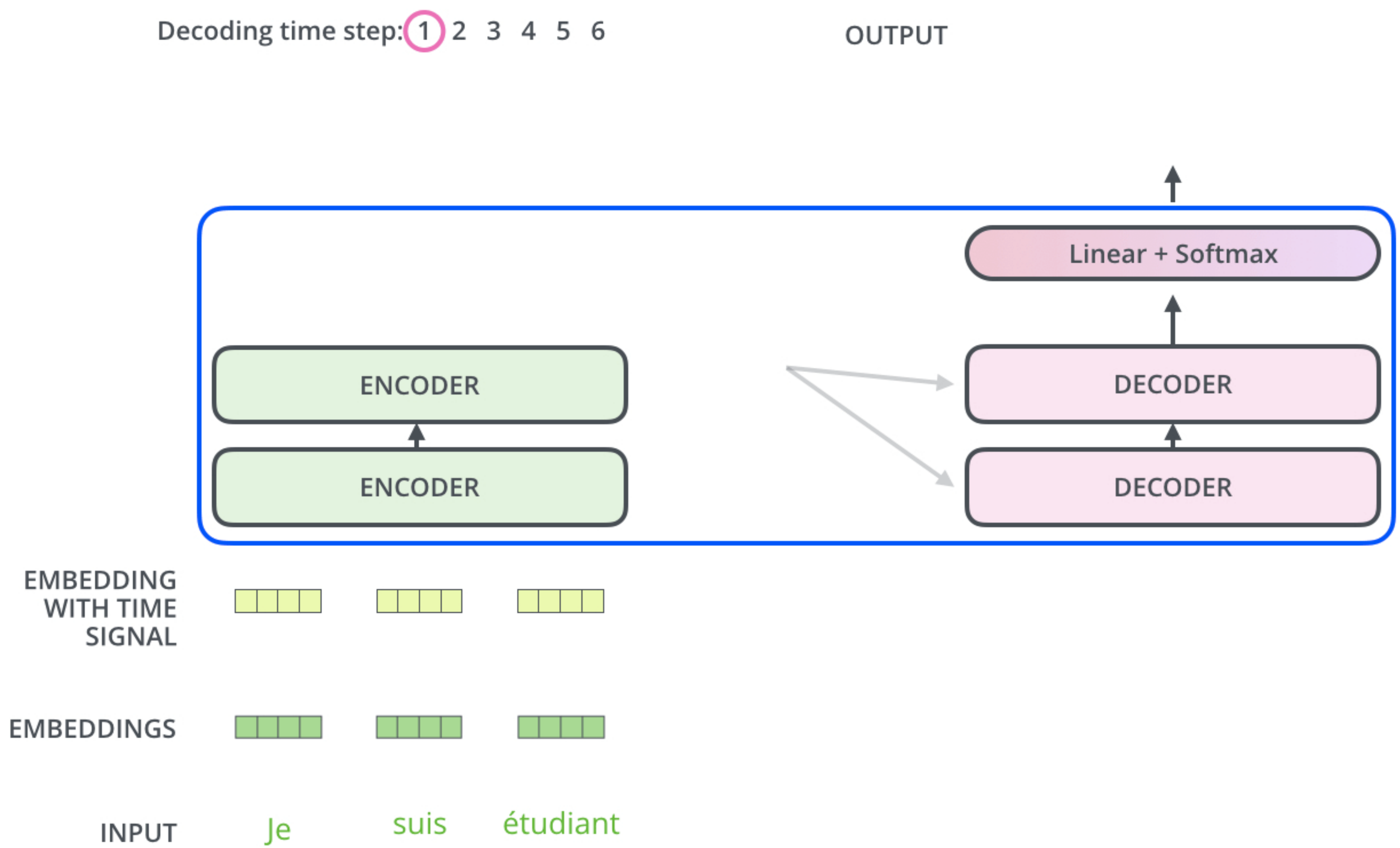
The Transformer Encoder-Decoder

Final Result - Transformer Architecture



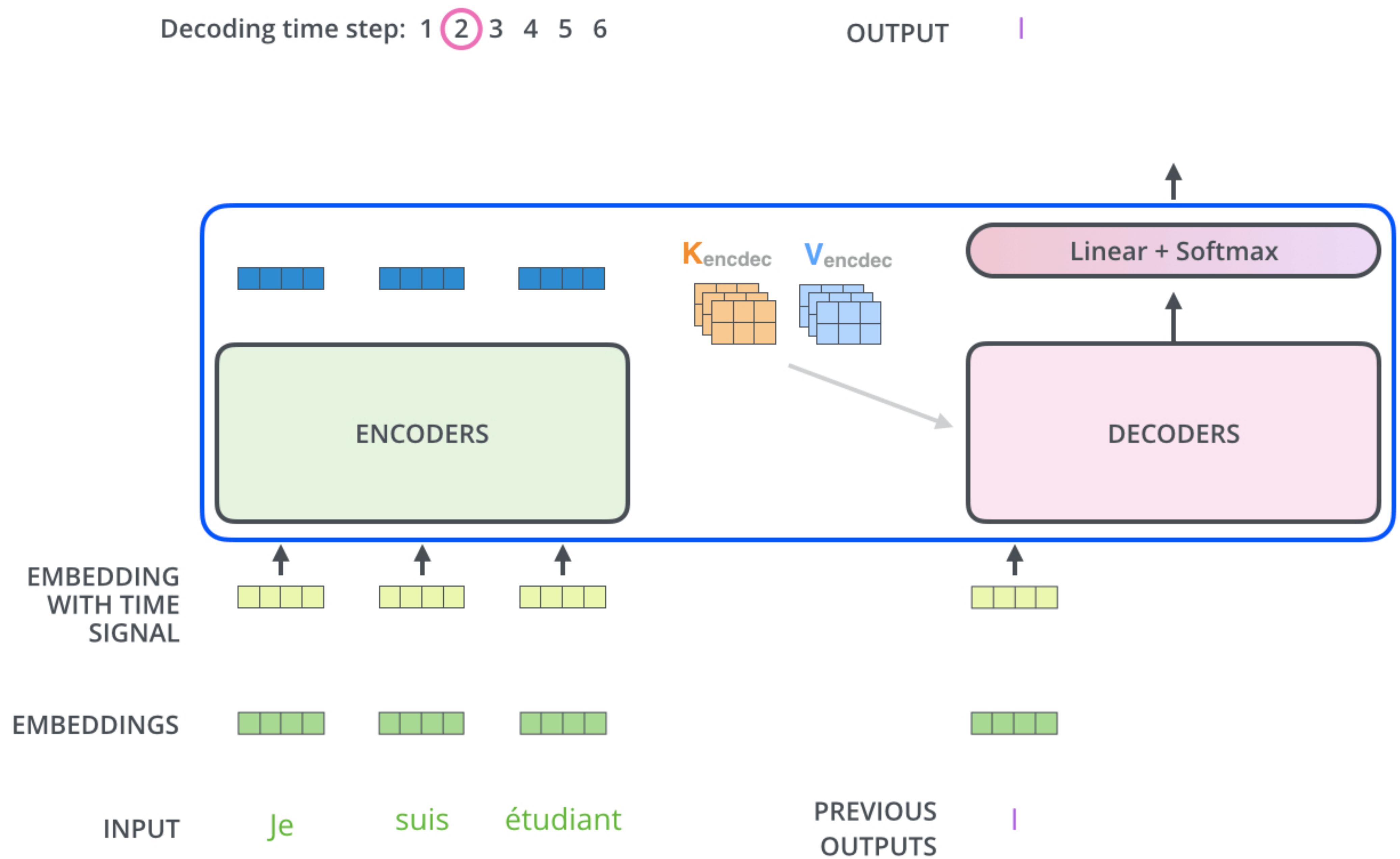
The Transformer Encoder-Decoder

Final Result - Transformer Architecture



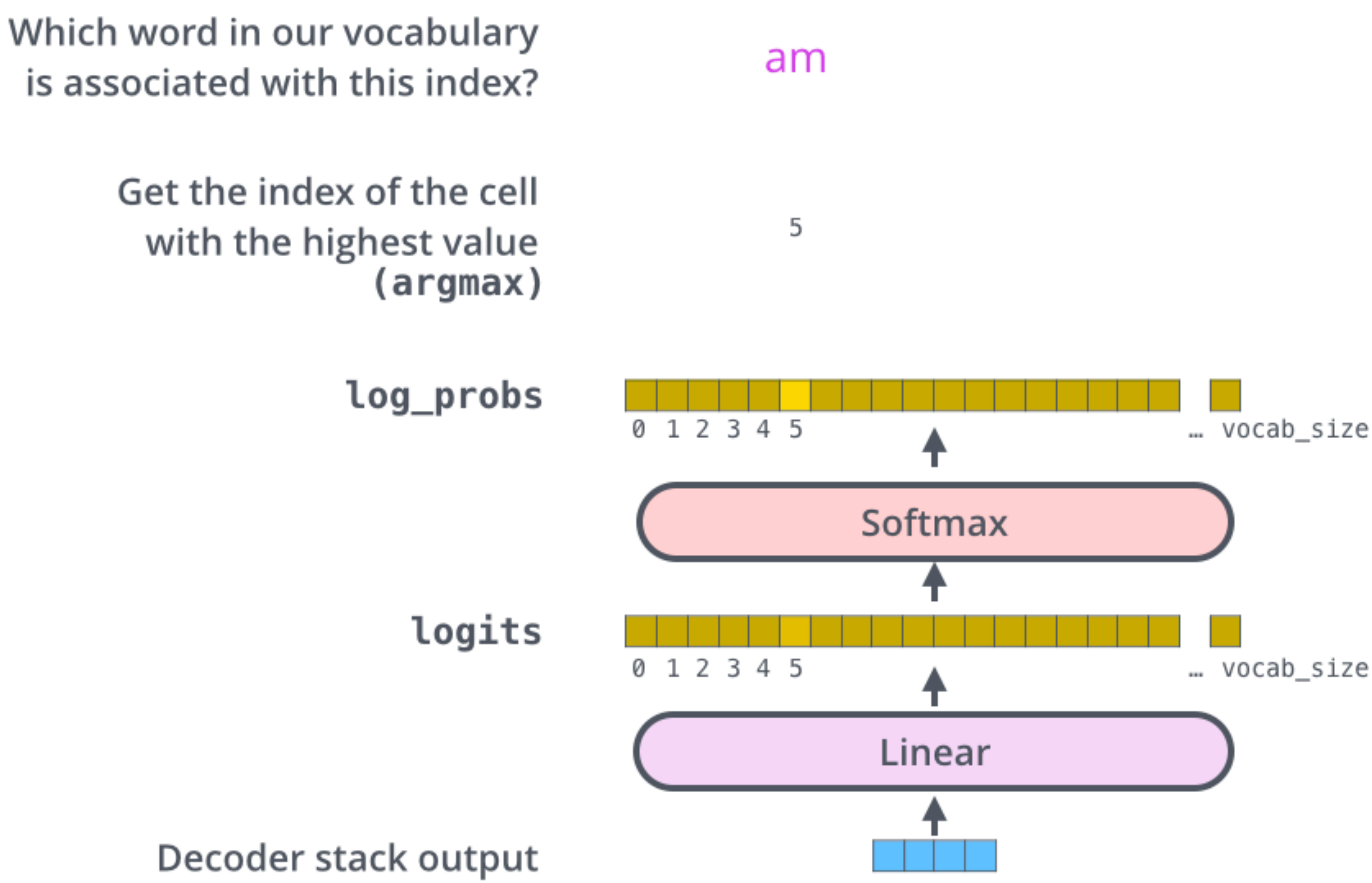
The Transformer Encoder-Decoder

Final Result - Transformer Architecture



The Transformer Encoder-Decoder

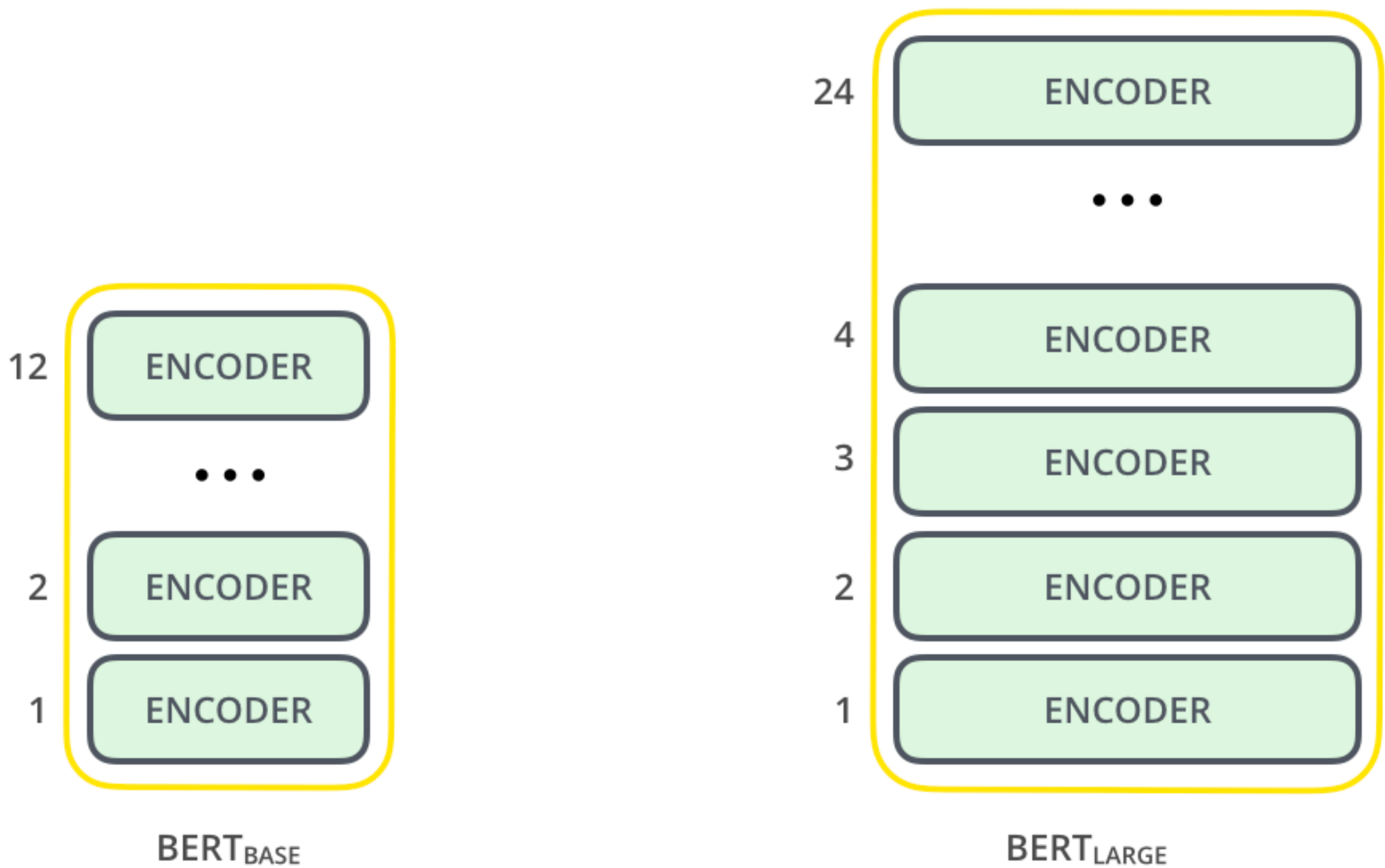
Final Result - Transformer Architecture



What’s next?

Contextualized Embeddings

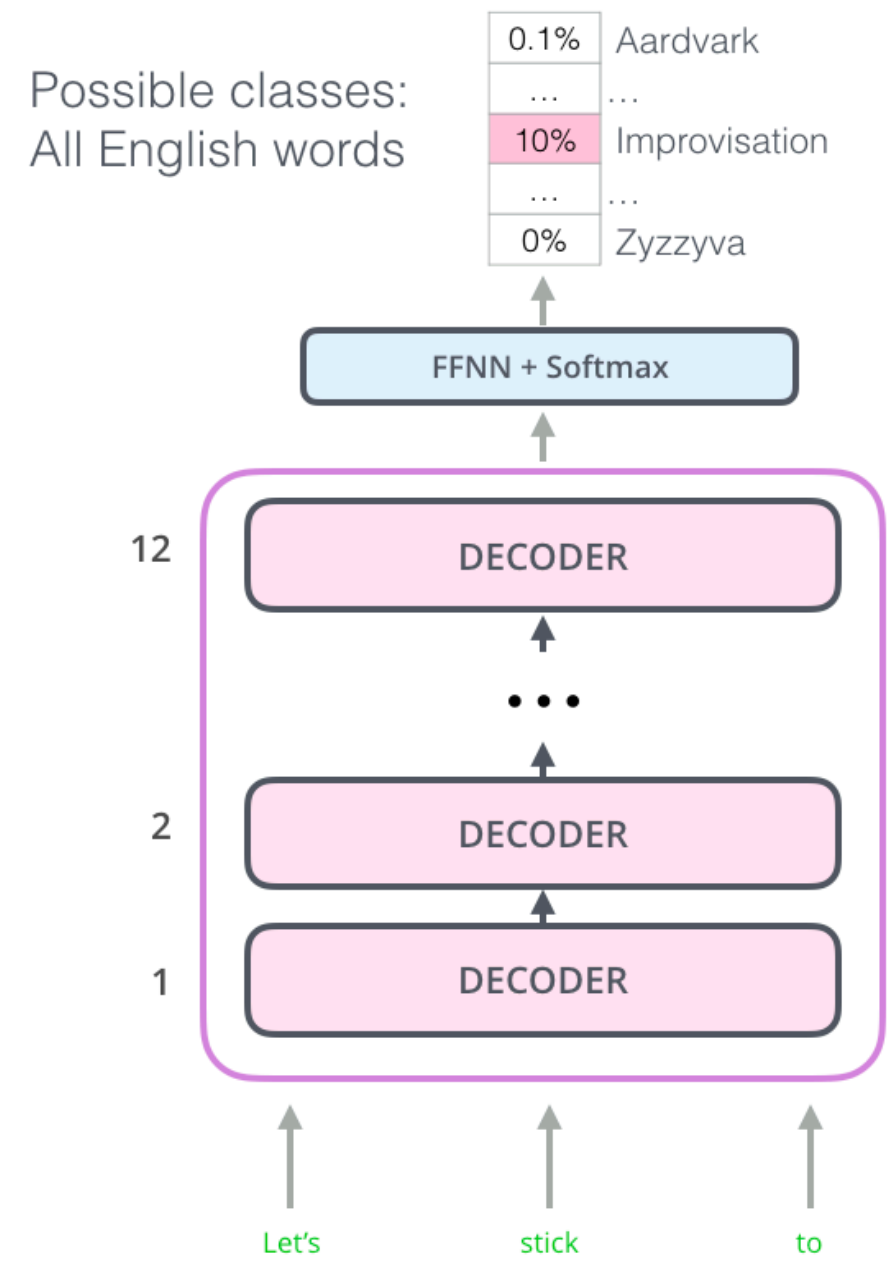
Using Transformer Architecture



Contextualized Embeddings

Using Transformer Architecture

- 175 billion parameters
  - GPT-2 had 1.5B
  - The largest so far (by Microsoft) had 17B
- Training cost: $12M



Transformer Problems

- **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]

Transformer Problems

Quadratic Computation

- Transformer benefit over recurrent networks → Highly parallelizable
- Drawback → total number of operations grows as $O(T^2 d)$
  - T is the sequence length, and d is the dimensionality.

$$XQ \quad K^\top X^\top = XQK^\top X^\top \in \mathbb{R}^{T \times T}$$

Need to compute all pairs of interactions! $O(T^2 d)$

- Example: d as around 1000:
  - If $T \leq 30$; $T^2 \leq 900$
  - In practice we set a bound like $T = 512$
  - What if we are working on a long sentence where $T \geq 10{,}000$

Transformer Problems

Quadratic Computation

- Linformer [Wang et al., 2020]
- Key Idea:
    - map the sequence length dimension to a lower dimensional space for values, keys

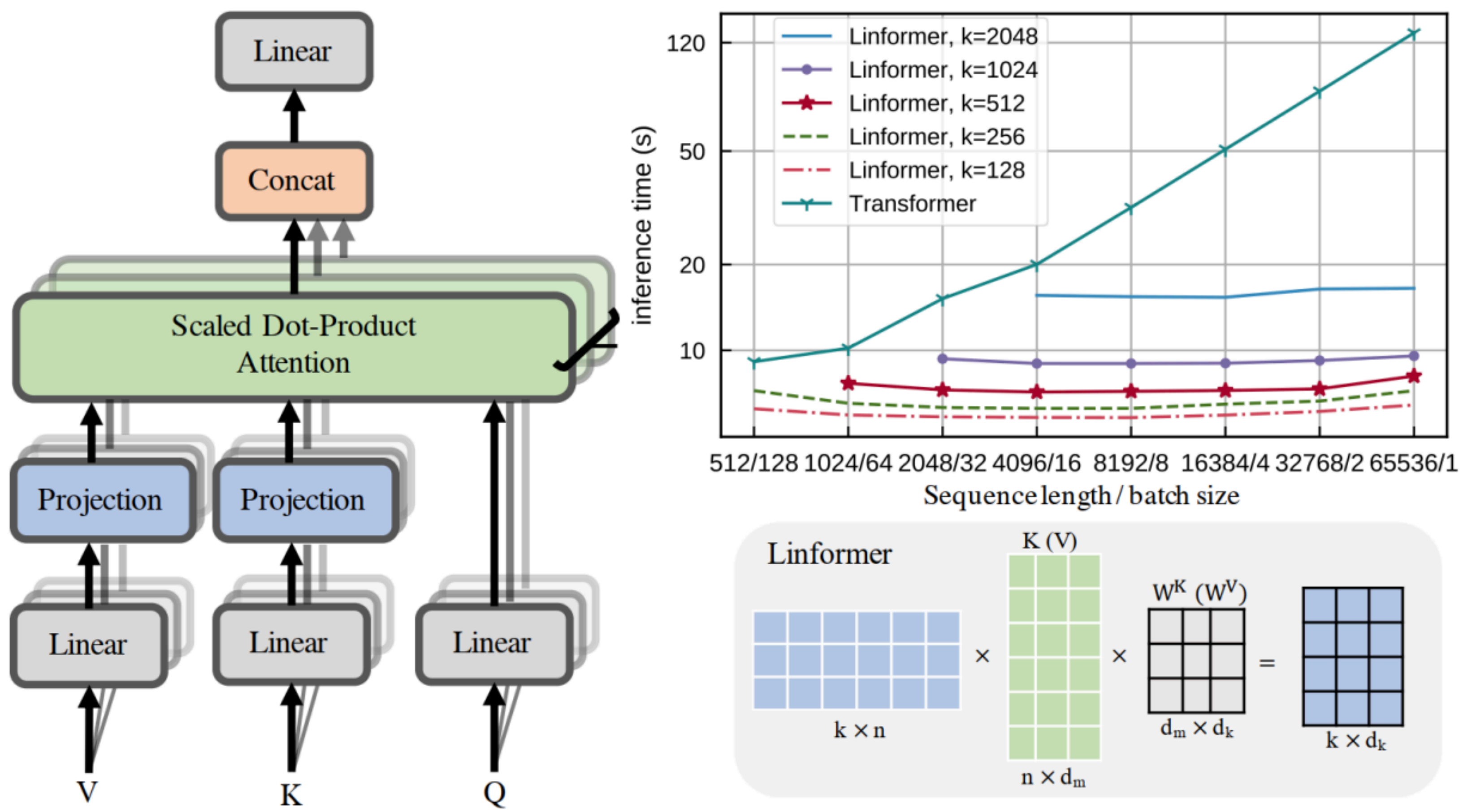
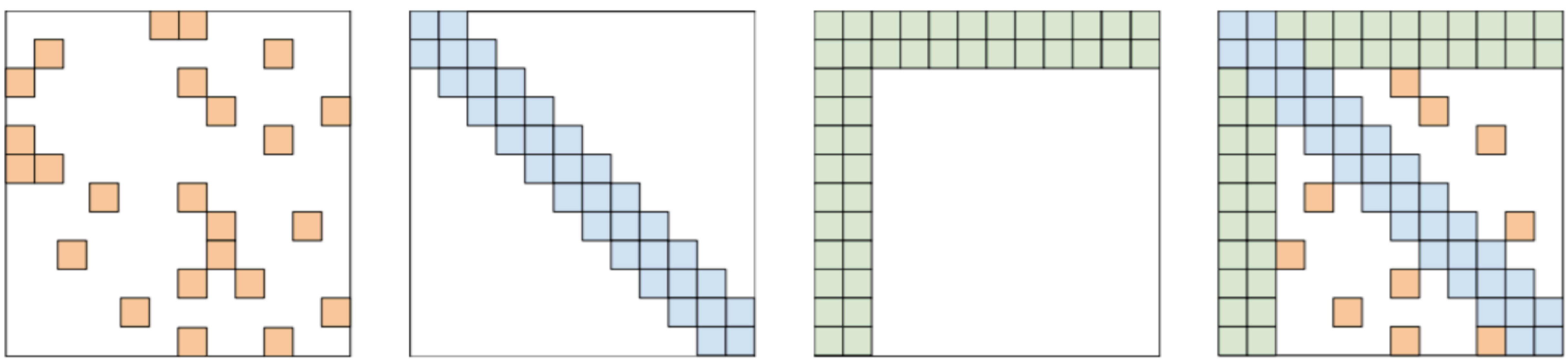


Figure 2: Left and bottom-right show architecture and example of our proposed multihead linear self-attention. Top right shows inference time vs. sequence length for various Linformer models.

Transformer Problems

Quadratic Computation

- 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

Transformer Problems

Position Embeddings

- **Q: Do we actually need absolute position embeddings?**
    - A: It turns out, not really!

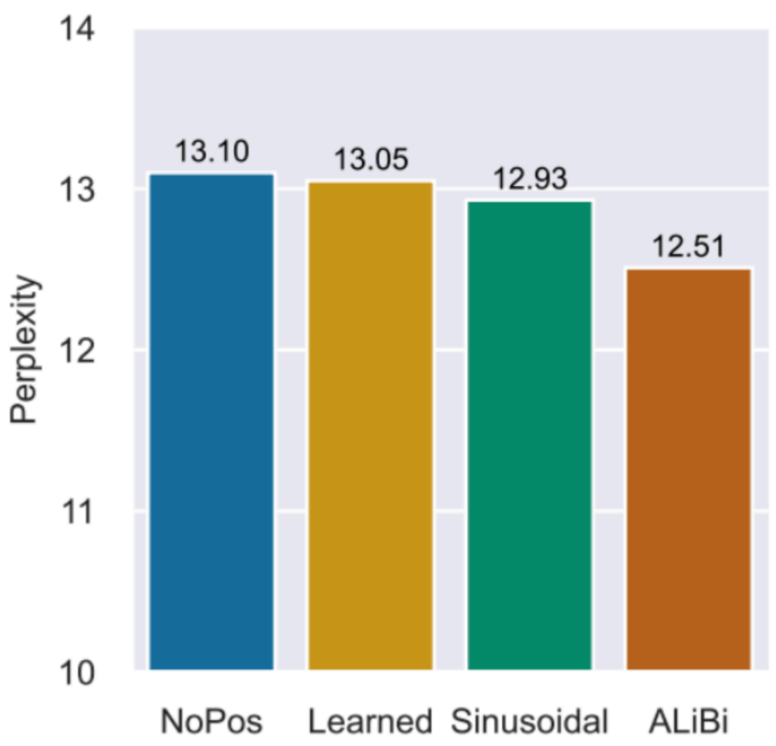


Figure 1: Transformer language models trained without explicitly encoding positional information (*NoPos*) approach the performance of models trained with various positional encoding methods. All models have 1.3B parameters, and are trained on an excerpt of the Pile.

Question?

References and Further Reading

- Jay Alammar, The Illustrated Transformer
- Lena Voita, Sequence to Sequence (seq2seq) and Attention
- John Hewitt, Stanford CS224N Lecture 9 - Self- Attention and Transformers