Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

December 10, 2024

Introduction

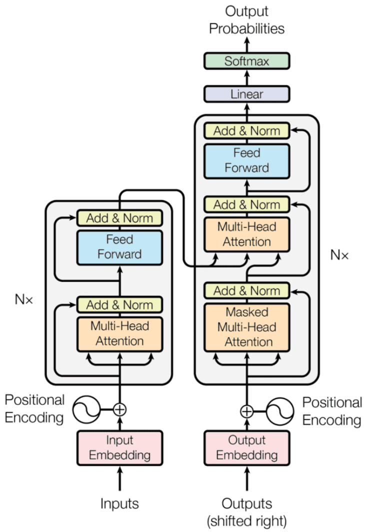
Why Transformers and BERT?

- Tasks like translation, sentiment analysis, and other NLP challenges require robust models.
- Two major architectures to address these tasks:
  - **Transformers** for tasks like translation.
  - **BERT** for tasks like sentiment analysis.

Transformers: A General Architecture

- Transformers are designed for sequence-to-sequence tasks.
- Consist of two main parts:
  - **Encoder:** Encodes the input sequence.
  - **Decoder:** Decodes the encoded sequence to produce the output.
- Widely used in tasks like machine translation.

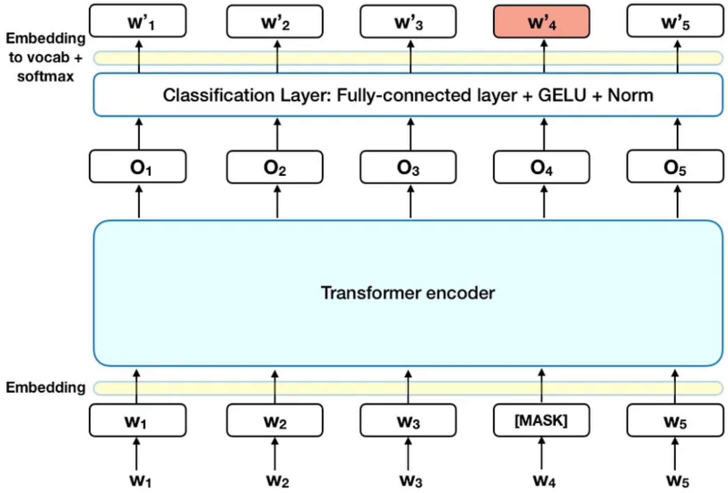
Transformers: A General Architecture



BERT: A Specialized Transformer Architecture

- Derived from the Transformer architecture.
- **Key differences:**
  - Only uses the **Encoder** part of the Transformer.
  - Pretrained on large corpora using bidirectional context.
- Designed for tasks like:
  - Sentiment analysis.
  - Named Entity Recognition (NER).
  - Question Answering.

BERT: A Specialized Transformer Architecture

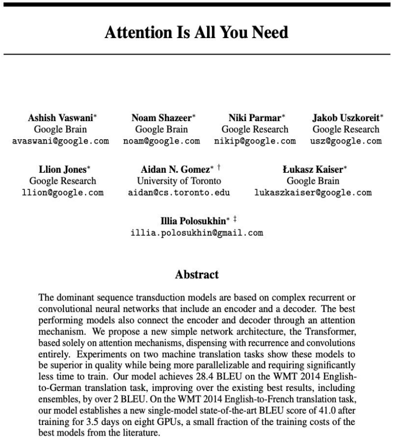

W'1
W'2
W'3
W'4
W'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Summary

Feature	**BERT**	**Transformers**
Purpose	Text understanding	General sequence-to-sequence tasks
Core Architecture	Transformer Encoder	Encoder and Decoder
Directionality	Bi-directional	Flexible (uni-/bi-directional)
Training Tasks	MLM, NSP	Task-dependent
Applications	Language understanding tasks	Understanding and generation tasks

Let's dive deeper into these architectures!

Attention is All You Need!



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

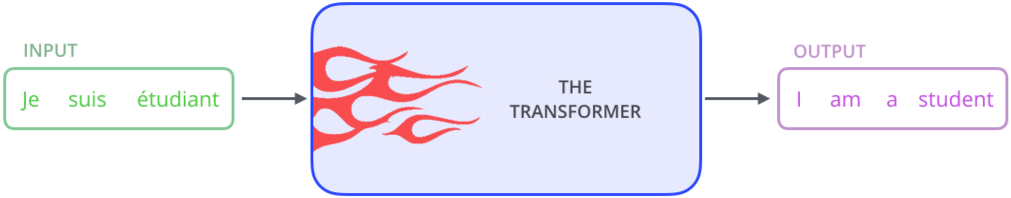
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

https://arxiv.org/abs/1706.03762

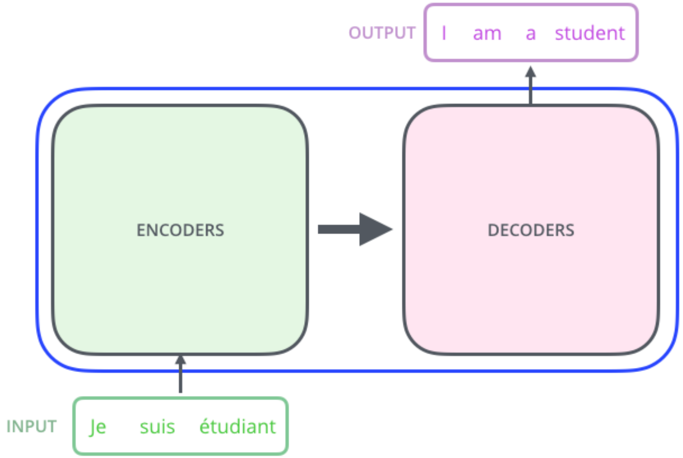
A High-Level Look

Let's begin by looking at the model as a single black box. In a machine translation application, it would take a sentence in one language, and output its translation in another.



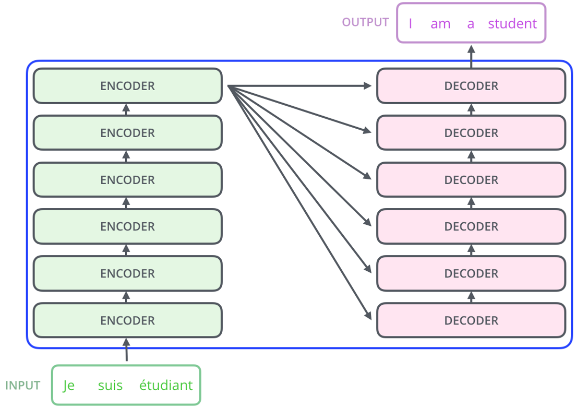
Now, we will pop open this black box, revealing its inner workings step by step.

Encoder, a Decoder, and connections between them.



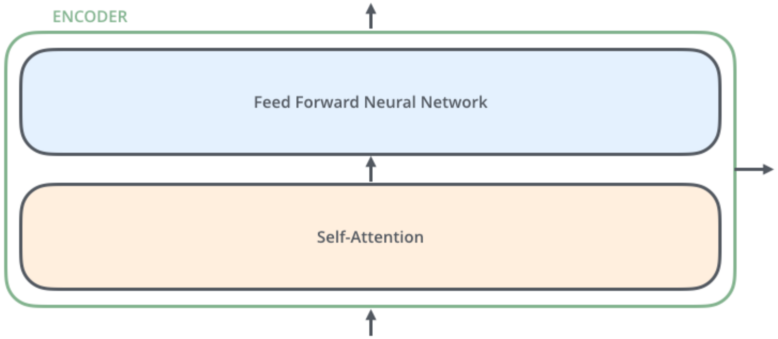
Encoder+Decoder

The encoding component is a stack of encoders (the paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements). The decoding component is a stack of decoders of the same number.



Encoder

The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:

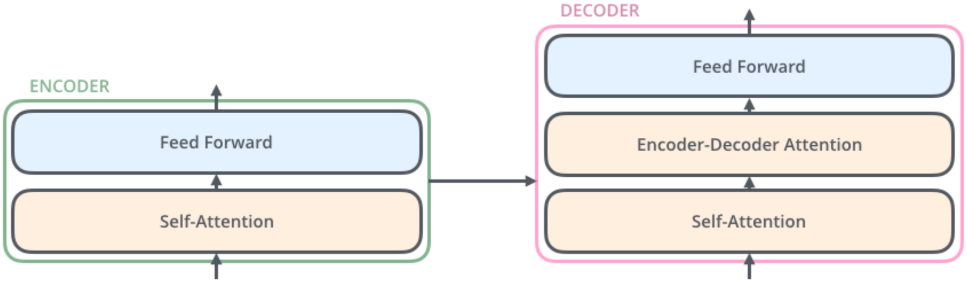


Encoder

1. The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word.
2. The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position.

Decoder

The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence.



Bringing The Tensors Into The Picture

Now that we've seen the major components of the model, let's start to look at the various vectors/tensors and how they flow between these components to turn the input of a trained model into an output.
As is the case in NLP applications in general, we begin by turning each input word into a vector using an embedding algorithm.

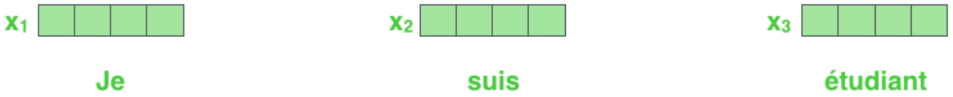
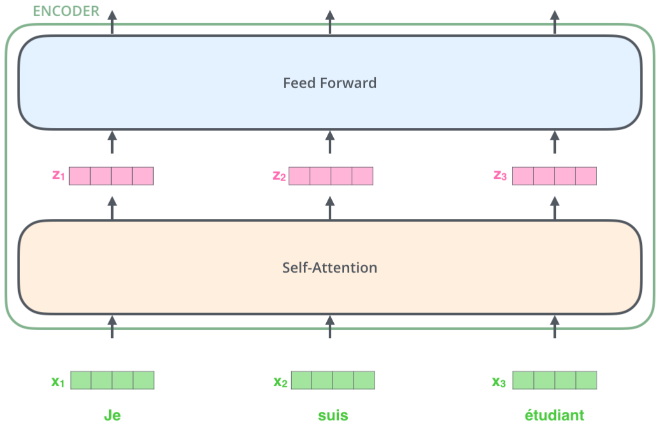


Figure 1: Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

Bringing The Tensors Into The Picture

After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

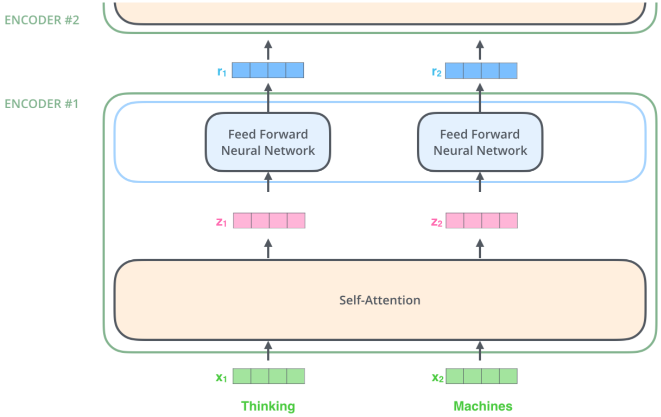


Key Property of the Transformer

- Each word in the input flows through its **own path** in the encoder.
- **Self-Attention Layer:** Introduces dependencies between these paths.
- **Feed-Forward Layer:** No dependencies between paths, allowing **parallel execution** for all positions.

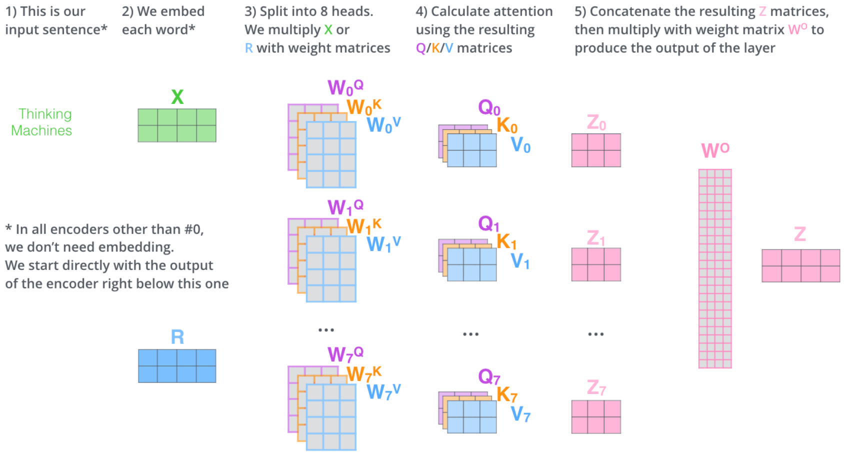
Now We're Encoding!

As we've mentioned already, an encoder receives a list of vectors as input. It processes this list by passing these vectors into a **self-attention** layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder.



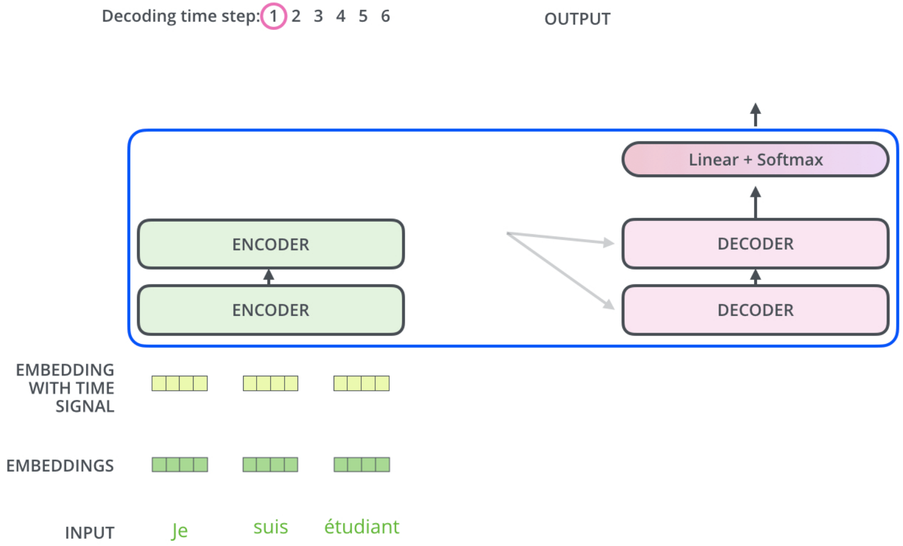
Multi-Headed Attention

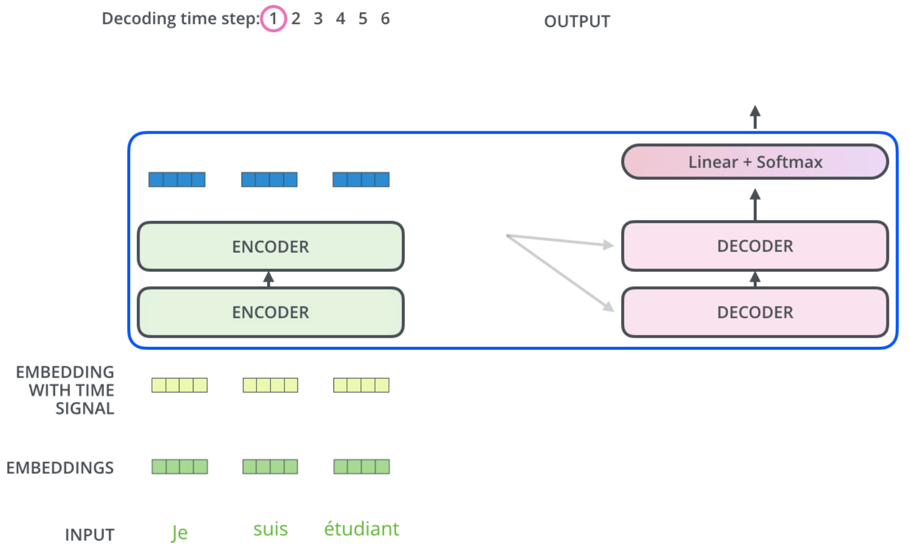
The paper further refined the self-attention layer by adding a mechanism called **multi-headed** attention.

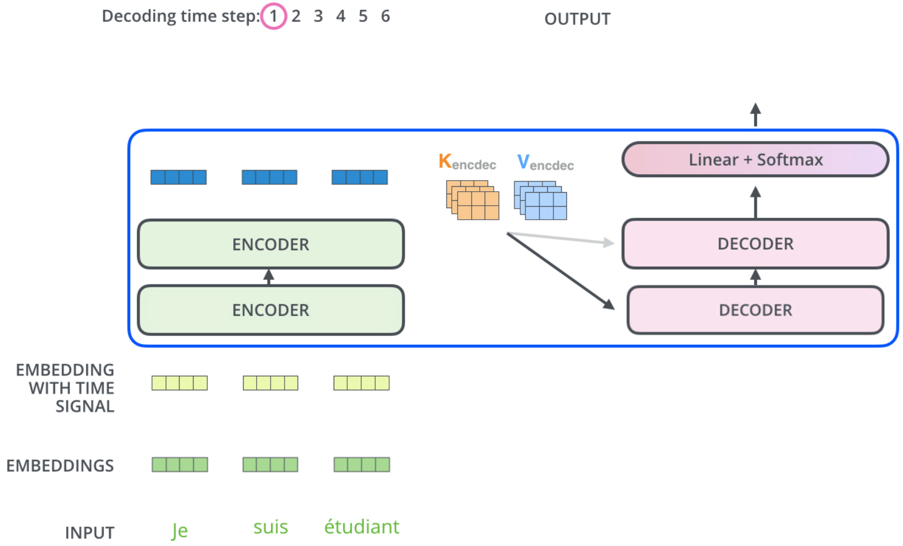

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W_0^Q W_0^K W_0^V
W_1^Q W_1^K W_1^V
...
W_7^Q W_7^K W_7^V
Q_0 K_0 V_0
Q_1 K_1 V_1
...
Q_7 K_7 V_7
Z_0
Z_1
...
Z_7
W^O
Z

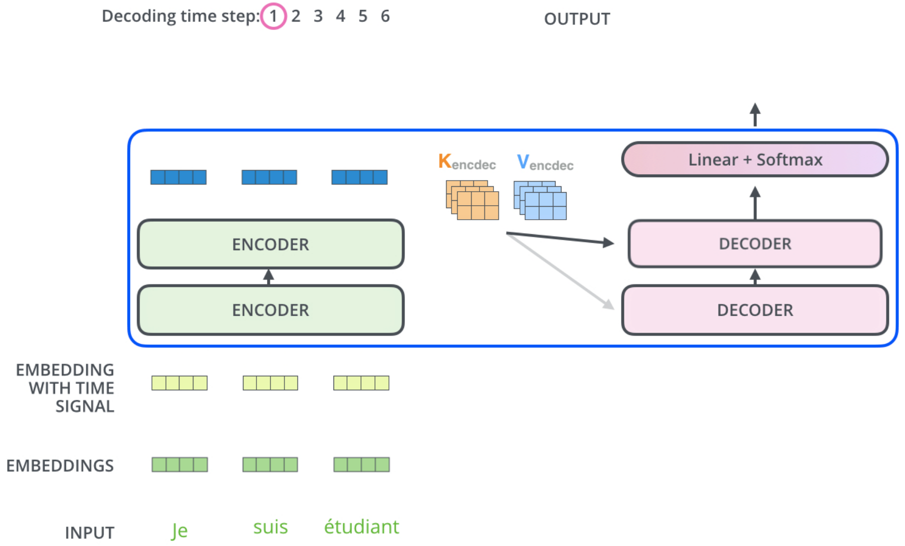
The Decoder Side

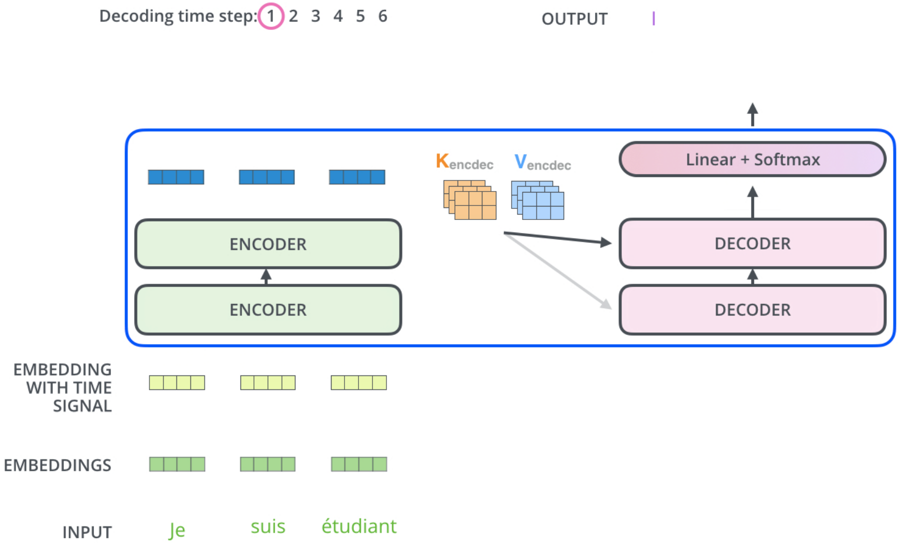
Now that we've covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. But let's take a look at how they work together.

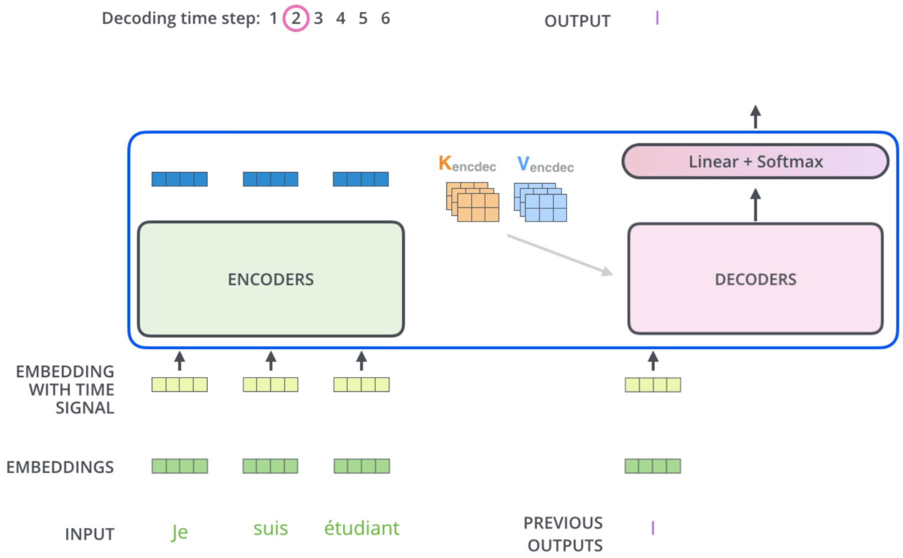

Decoding time step: 1 2 3 4 5 6
OUTPUT
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant

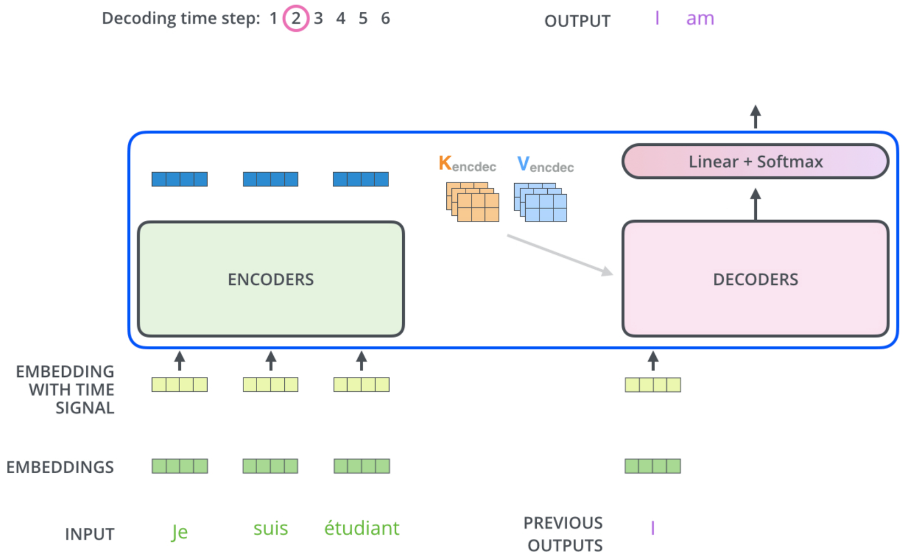

Decoding time step: 1 2 3 4 5 6
OUTPUT
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant

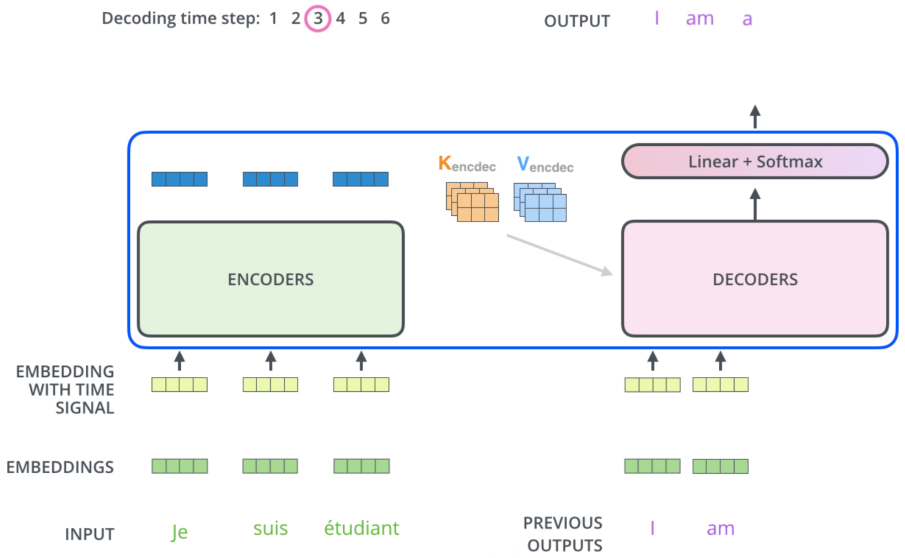

Decoding time step: 1 2 3 4 5 6
OUTPUT
Kencdec
Vencdec
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant

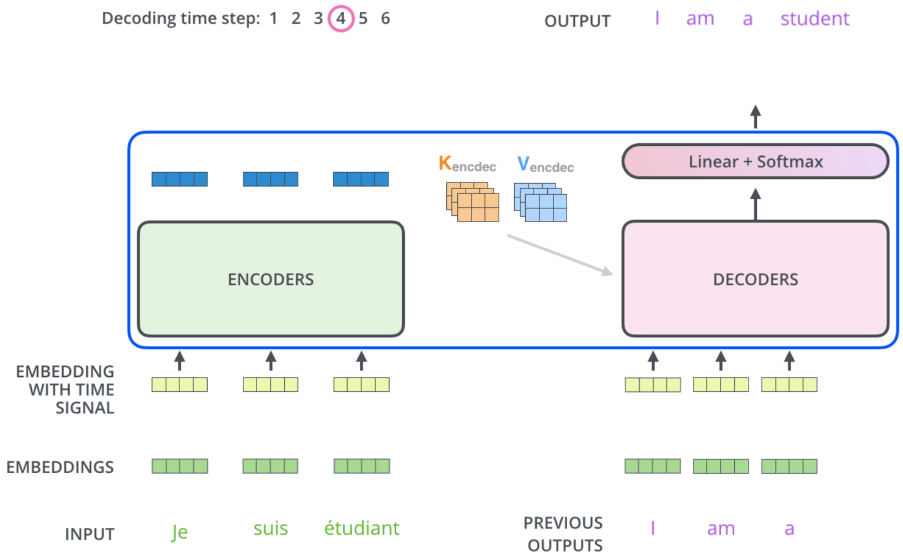

Decoding time step: 1 2 3 4 5 6
OUTPUT
K encdec
V encdec
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant

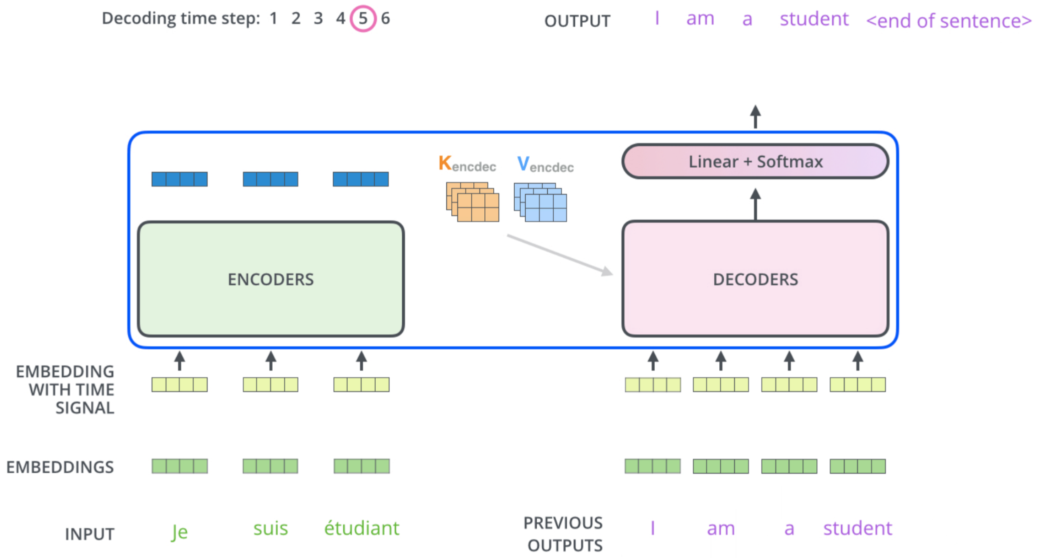

Decoding time step: 1 2 3 4 5 6
OUTPUT
Kencdec
Vencdec
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
K encdec
V encdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I


Decoding time step: 1 2 3 4 5 6
OUTPUT I am
K encdec
V encdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

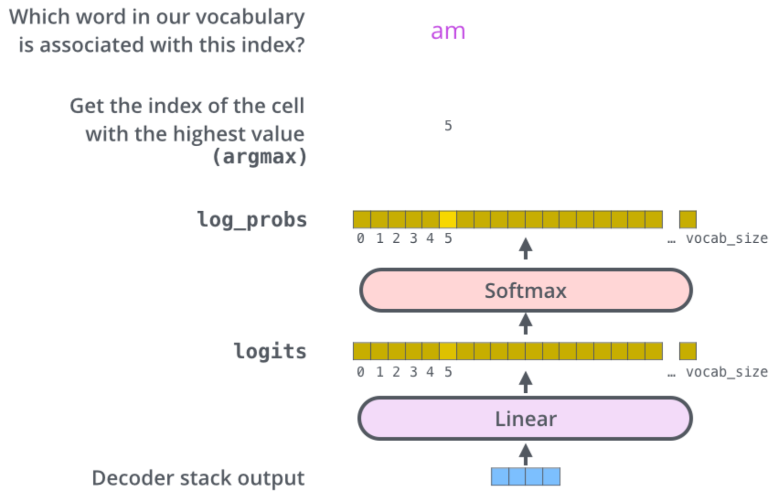

Decoding time step: 1 2 3 4 5 6
OUTPUT
I
am
a
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I
am


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
am
a
student
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I
am
a


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
am
a
student
<end of sentence>
Kencdec
Vencdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I
am
a
student

The Final Linear and Softmax Layer

- The decoder stack outputs a vector of floats, which is turned into a word by the final Linear layer, followed by a Softmax layer.
- The Linear layer:
  - Is a fully connected neural network.
  - Projects the vector produced by the decoder stack into a larger vector, known as the logits vector.
- Example:
  - If the model's output vocabulary contains 10,000 unique English words, the logits vector will have 10,000 cells, each representing the score of a unique word.
- The softmax layer:
  - Converts the scores into probabilities (all positive, summing to 1.0).
  - The word associated with the cell having the highest probability is selected as the output for this time step.


Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output