**The Transformer Architecture From a Top View**

Exploring the encoder-decoder magic in NLP behind LLMs

[[](https://medium.com/@dimitris.effrosynidis?source=post_page-----e8079c96b473--------------------------------)](https://medium.com/@dimitris.effrosynidis?source=post_page-----e8079c96b473--------------------------------)

[[Towards AI](https://pub.towardsai.net/?source=post_page-----e8079c96b473--------------------------------)](https://pub.towardsai.net/?source=post_page-----e8079c96b473--------------------------------)

[Dimitris Effrosynidis](https://medium.com/@dimitris.effrosynidis?source=post_page-----e8079c96b473--------------------------------)

·

Follow

Published in

[Towards AI](https://pub.towardsai.net/?source=post_page-----e8079c96b473--------------------------------)

·

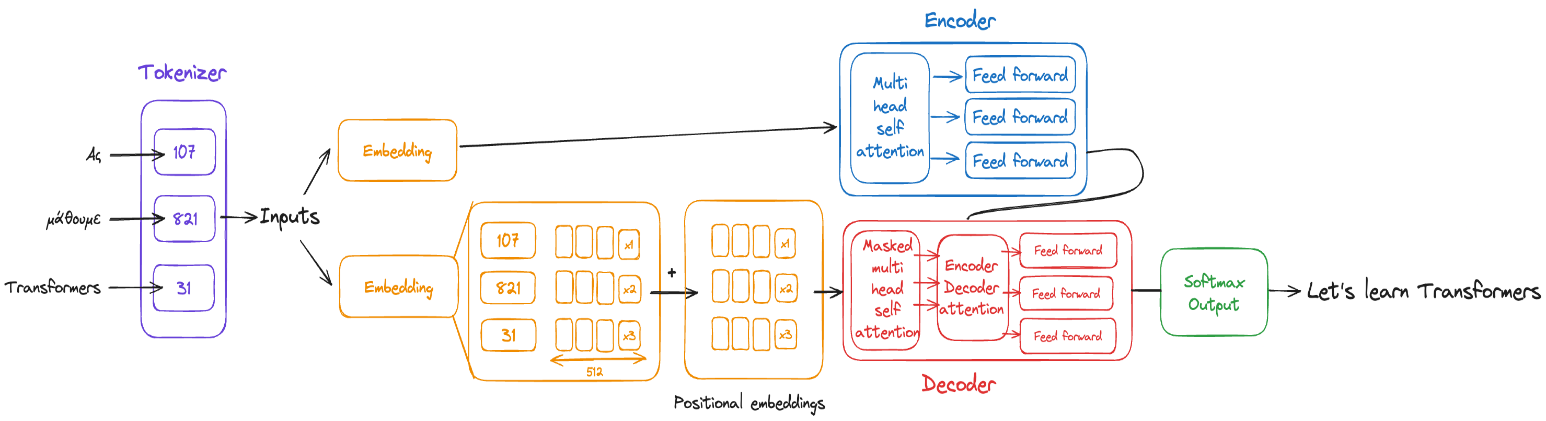
7 min read

·

Feb 21, 2024

430

3

Image created by the author.

The state-of-the-art Natural Language Processing (NLP) models used to be Recurrent Neural Networks (RNN) among others.

And then came Transformers.

Transformer architecture significantly improved natural language task performance compared to earlier RNNs.

Developed by Vaswani et al. in their 2017 paper “Attention is All You Need,” Transformers revolutionized NLP by leveraging self-attention mechanisms, allowing the model to learn the relevance and context of all words in a sentence.

Unlike RNNs that process data sequentially, **Transformers analyze all parts of the sentence simultaneously**. This parallel processing capability allows Transformers to learn the context and relevance of each word about every other word in a sentence or document, overcoming limitations related to long-term dependency and computational efficiency found in RNNs.

But let’s explore the architecture step by step.

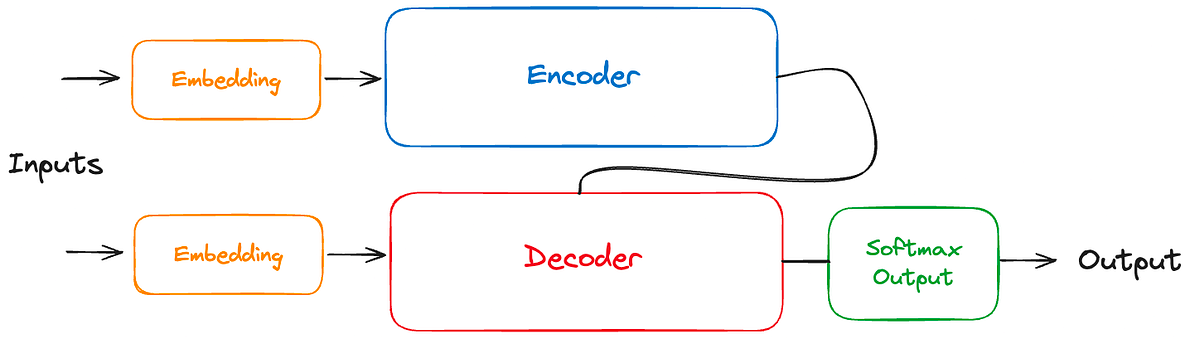


Image created by the author.

* There are two components in a Transformer Architecture: the Encoder and the Decoder.
* These components work in conjunction with each other and they share several similarities.
* **Encoder**: Converts an input sequence of tokens into a rich, continuous representation that captures the context of each token within the sequence. Its output is a sequence of embedding vectors, often called the hidden state or context.
* **Decoder**: Uses the encoder’s hidden state to iteratively generate an output sequence of tokens, one token at a time.

Although both the Encoder and the Decoder exist in the Transformer Architecture, there are 3 types of transformers depending on whether they use only the encoder, only the decoder, or both.

**Encoder-only Transformers**

* Think of these models as **expert analysts** who can deeply understand and interpret a block of text.
* These models convert an input sequence of text into a rich numerical representation that is well-suited for tasks like **text classification** or **named entity recognition (NER)**.
* **BERT** and its variants, like **RoBERTa** and **DistilBERT**, belong to this class of architectures.
* These models use **bidirectional** attention. They’re designed to pay attention to the entire context around a word — both what comes before and after it.

**Decoder-only Transformers**

* Imagine these models as **creative storytellers** who pick up where you leave off.
* Given a prompt of text like “Learning transformers is…” these models will autocomplete the sequence by iteratively **predicting the most probable next word**(hopefully “fun”).
* The family of **GPT** models belongs to this class.
* The representation computed for a given token in this architecture depends only on the left context (the story so far) to predict the future (called **autoregressive** attention).

**Encoder-decoder Transformers**

* These are the **versatile multitaskers** of the Transformer family, capable of transforming text from one form to another.
* They first digest the input text, capturing its essence and nuances (thanks to the encoder), and then, drawing on this deep understanding, the decoder part crafts a new piece of text in response.
* Their suitability fits in **machine translation** and **summarization** tasks.
* Transformer models belonging to this category are **T5** and **BART**.

*In this article, we will explore the****encoder-decoder transformer****as it includes both modules. Our example will be a****machine translation****task from****Greek to English****.*

**1. Tokenizer**

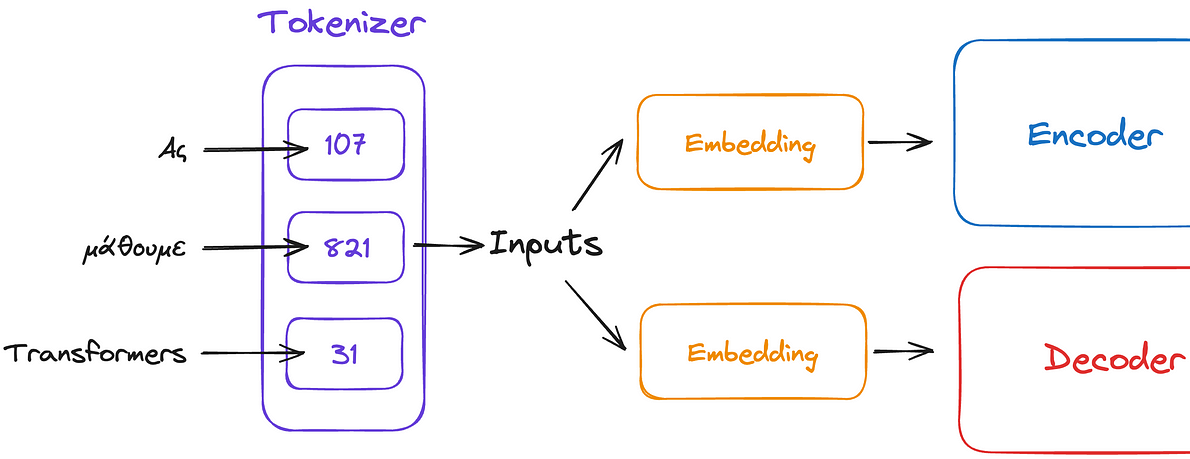


Image created by the author.

* Before processing texts with a model, the first step is tokenization.
* This step helps the computer to interpret the words by converting them into numbers. Each unique token will have its unique number.
* Once you’ve selected a tokenizer to train the model, you must use the same tokenizer when you generate text.
* Now, you can pass the input to the embedding layer.

**2. Embedding Layer**

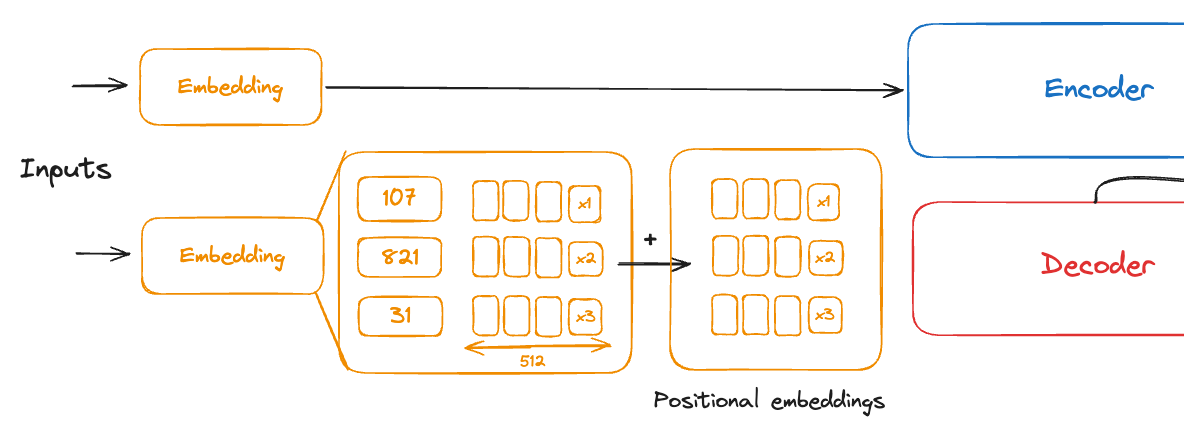


Image created by the author.

Note that in the above image, we peek inside one embedder, but both are the same.

* The embedding layer **transforms the tokenized numerical representations into dense vector embeddings**.
* A trainable vector embedding space is a high-dimensional space where each token is represented as a vector and occupies a unique location within that space.
* Each token in the vocabulary is matched to a multi-dimensional vector (with a size of 512 for example), and the intuition is that these vectors learn to encode the semantic meaning and context of individual tokens in the input sequence.

**Positional embeddings**

* In combination with raw embeddings, we also add positional.
* The model processes each of the input tokens in parallel.
* The positional embeddings give the model knowledge of the word order, which is very important when trying to understand text.

**3. Encoder**

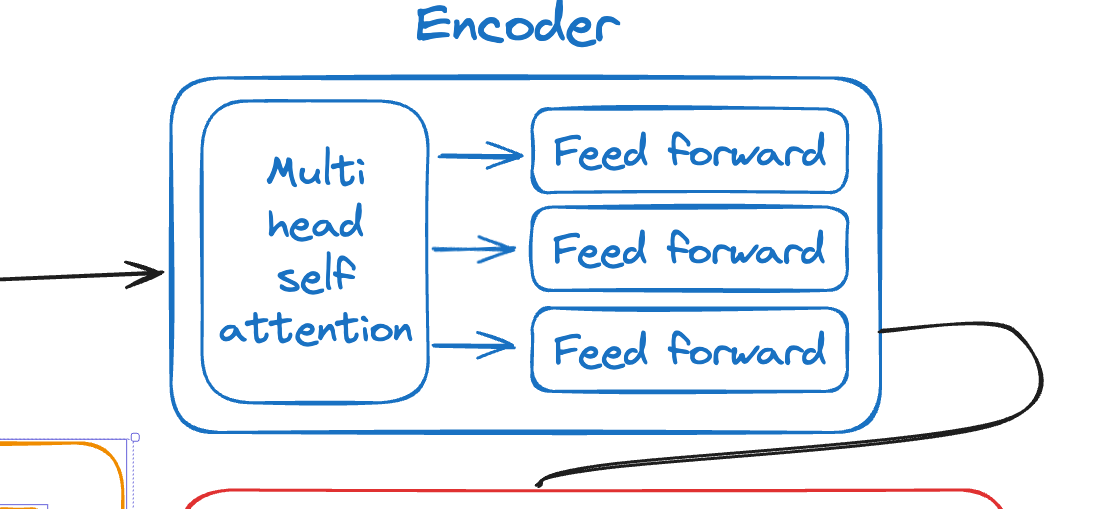


Image created by the author.

* First of all, it is important to note that a transformer does not have a single encoder but **a stack of many encoders next to each other**. All the encoders are identical. For example, BERT has a stack of 24 encoders.
* The sequence of embeddings from the embedded layer is the input of the encoder and they are first fed to a layer called multi-head self-attention and after that to a fully connected feed-forward layer.
* The output is a vector of logits that are proportional to the probability score for each possible token of the tokenizer dictionary.

**Multi-head self-attention**

* This layer inside the Encoder has a specific task: **to understand every word in a sentence not just on its own, but also with every other word**.

Now, why “multi-head”?

* Because our model doesn’t just look at these relationships in one way. Instead, it has multiple “heads”, each looking at the sentence from a different perspective.
* One head might focus on the grammatical structure, another on the meaning of specific terms, and another on the tone of the sentence.
* By **examining**these**different aspects simultaneously**, the model gets a richer understanding of the text.

**Feed-forward**

* It is structured as a **two-layer fully connected (dense) neural network**.
* It does not process the whole sequence of embeddings as a single vector.
* Each embedding is processed on its own.
* It outputs transformed embeddings.
* These are then passed through the final output layer of the Transformer (not within this feed-forward layer itself) to produce logits, which are proportional to the token probabilities in the context of the entire model’s architecture.
* As in other neural networks, an activation function must be used. In this case, the GELU is used.
* **GELU** allows the model to introduce non-linearity in a way that’s particularly effective for the types of data distributions encountered in natural language, balancing between linear and non-linear transformations.

**4. Decoder**

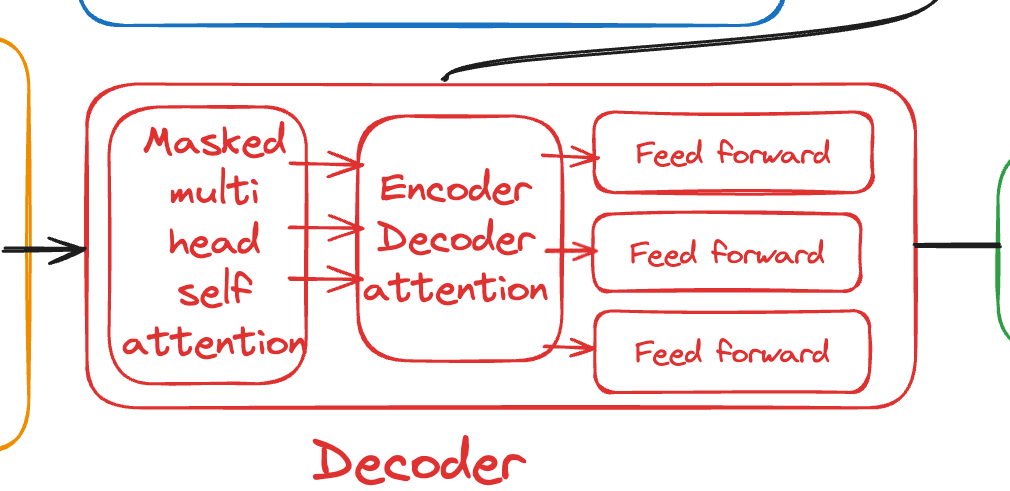


Image created by the author.

Similar to the encoder, the decoder is also **a stack of many decoders** (the same number as the encoders in an encoder-decoder model) next to each other, which are identical. For example, the (decoder-only) GPT-2 Extra Large model has a stack of 48 decoder layers.

The main difference between the decoder and encoder is that the decoder has two attention sublayers:

**Masked multi-head self-attention**

* Ensures that the tokens we generate at each timestep are only **based on the past outputs** and the current token being predicted. This is the notion behind “mask”.
* Without this, the decoder could cheat during training by simply copying the target translations.
* The term multi-head is the same as in the encoder. Each head learns different aspects of the data by focusing on different parts of the sequence and considering various relationships between the tokens.

**Encoder-decoder attention**

* This layer enables the decoder to **focus on different parts of the input sequence** (such as two different languages) while generating each token of the output sequence.
* As the decoder works on producing the next token in the output sequence, it takes into consideration the current context and what it has generated so far.
* This allows the decoder to “see” the most relevant parts of the input sequence that should influence the generation of the next output token.

The Decoder’s output will be the probability score for each token existing in the tokenizer dictionary (all adding to 1) and the token with the higher probability will be returned.

**Keep in touch**

➥Follow me on [Medium](https://medium.com/@dimitris.effrosynidis/subscribe) for more content like this.  
➥Let’s connect on [LinkedIn](https://www.linkedin.com/in/dimitrios-effrosynidis/) or [**𝕏**](https://twitter.com/dimeffro).  
➥Check my [GitHub](https://github.com/Deffro).