**Contents**

1. Objectives
2. Core Ideas
3. What is a transformer?
4. The Encoder Flow
5. The Decoder Flow
6. Tasks
7. Additional Resources

# Objectives

The objective of this lab is to understand and implement the core components of the Transformer architecture, a state-of-the-art model in natural language processing.

# Core Ideas

* The Transformer model is a type of **neural network** that learns context from sequential data and can generate new sequences from it.
* Unlike traditional encoder-decoder architectures that use **RNNs**, Transformers **do not rely on recurrence**, instead, they use **self-attention mechanisms** to model relationships between tokens.
* Transformers are **state-of-the-art in NLP** and are structured using **encoder and decoder components**, each consisting of multiple identical layers.
* The **encoder** processes the input and generates a contextualized representation (e.g., from “How are you?” to encoded vectors).
* The **decoder** takes the encoder output and iteratively generates the target sequence (e.g., translating to a specific language).
* Both encoder and decoder are stacks of layers (typically 6 each), and **each layer includes**:
  + **Multi-head self-attention**
  + **Feed-forward neural network**
  + **Residual connections and layer normalization**
* The encoder captures the meaning of the input as a **context-rich matrix**, while the decoder uses that matrix and **autoregressive decoding** to generate the output one token at a time.
* This layered and attention-driven design enables Transformers to **comprehend context and meaning** without needing sequential recurrence.

1. **What is a transformer?**

Transformers were first developed to solve the problem of sequence transduction, or neural machine translation, which means they are meant to solve any task that transforms an input sequence to an output sequence. This is why they are called “Transformers”.

**3.1) Transformer Model**

A transformer model is a neural network that learns the context of sequential data and generates new data out of it. Transformers are a current state-of-the-art NLP model and are considered the evolution of the encoder-decoder architecture. However, while the encoder-decoder architecture relies mainly on Recurrent Neural Networks (RNNs) to extract sequential information, Transformers completely lack this recurrency. Rather they are specifically designed to comprehend context and meaning by analyzing the relationship between different elements, and they rely almost entirely on a mathematical technique called attention to do so.

**3.2) Transformer Architecture**

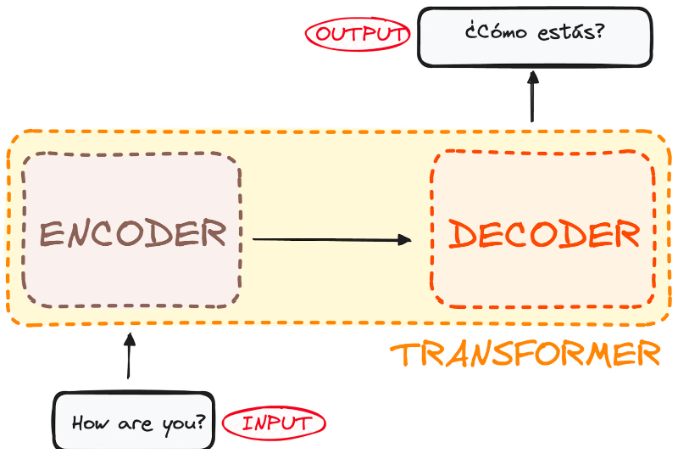
Originally devised for sequence transduction or neural machine translation, transformers excel in converting input sequences into output sequences. It is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. The main core characteristic of the Transformers architecture is that they maintain the encoder-decoder model.

If we start considering a Transformer for language translation as a simple black box, it would take a sentence in one language, English for instance, as an input and output its translation in English.



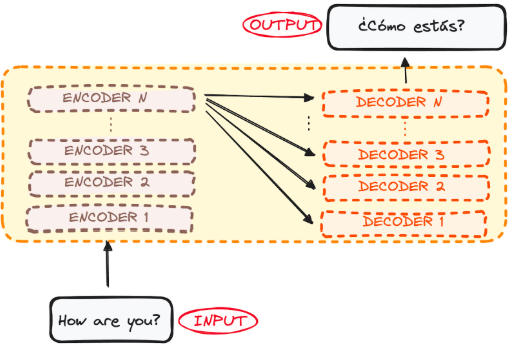
If we dive a little bit, we observe that this black box is composed of two main parts:

* The encoder takes in our input and outputs a matrix representation of that input. For instance, the English sentence “How are you?”
* The decoder takes in that encoded representation and iteratively generates an output. In our example, the translated sentence “¿Cómo estás?”



However, both the encoder and the decoder are actually a stack with multiple layers (same number for each). All encoders present the same structure, and the input gets into each of them and is passed to the next one. All decoders present the same structure as well and get the input from the last encoder and the previous decoder.

The original architecture consisted of 6 encoders and 6 decoders, but we can replicate as many layers as we want. So let’s assume N layers of each.

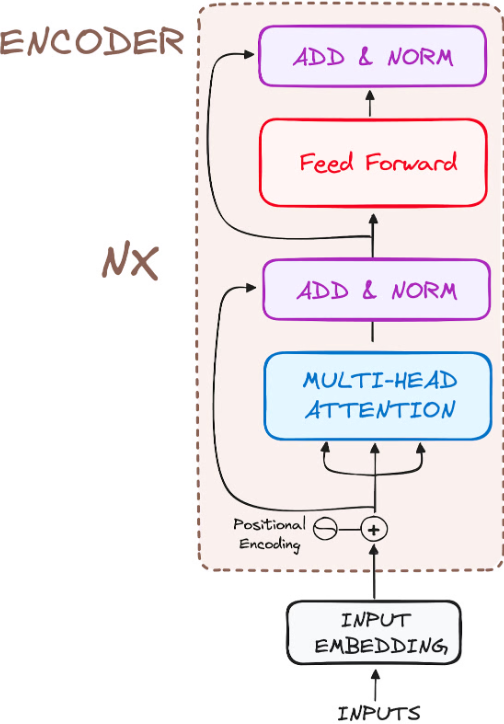


So now that we have a generic idea of the overall Transformer architecture, let’s focus on both Encoders and Decoders to understand better their working flow.

**4) The Encoder WorkFlow**

The encoder is a fundamental component of the Transformer architecture. The primary function of the encoder is to transform the input tokens into contextualized representations. Unlike earlier models that processed tokens independently, the Transformer encoder captures the context of each token with respect to the entire sequence.

Its structure composition consists as follows:

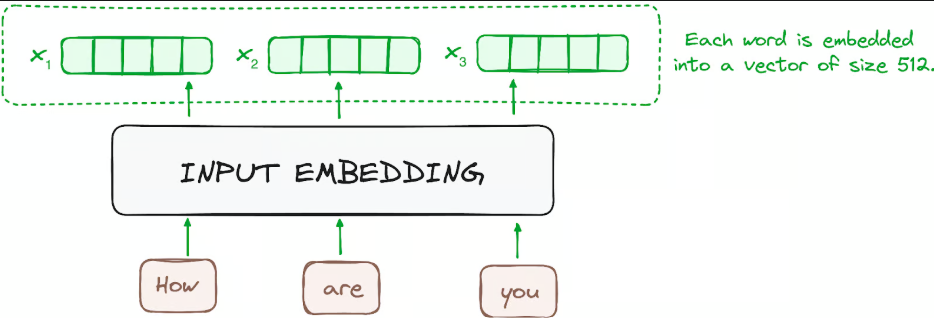


So let’s break its workflow into its most basic steps:

**4.1) STEP 1 - Input Embeddings**

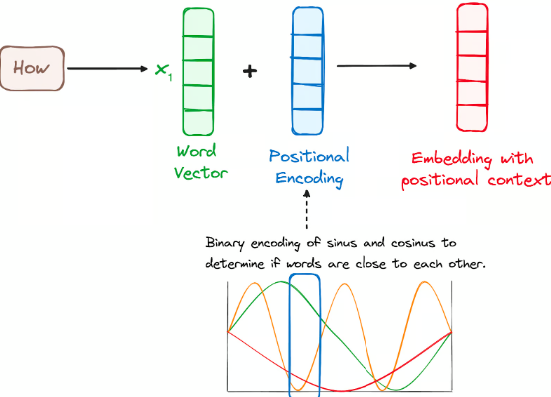
The embedding only happens in the bottom-most encoder. The encoder begins by converting input tokens - words or subwords - into vectors using embedding layers. These embeddings capture the semantic meaning of the tokens and convert them into numerical vectors.

All the encoders receive a list of vectors, each of size 512 (fixed-sized). In the bottom encoder, that would be the word embeddings, but in other encoders, it would be the output of the encoder that’s directly below them.

****

**4.2) STEP 2 - Positional Encoding**

Since Transformers do not have a recurrence mechanism like RNNs, they use positional encodings added to the input embeddings to provide information about the position of each token in the sequence. This allows them to understand the position of each word within the sentence. To do so, the researchers suggested employing a combination of various sine and cosine functions to create positional vectors, enabling the use of this positional encoder for sentences of any length. In this approach, each dimension is represented by unique frequencies and offsets of the wave, with the values ranging from -1 to 1, effectively representing each position.

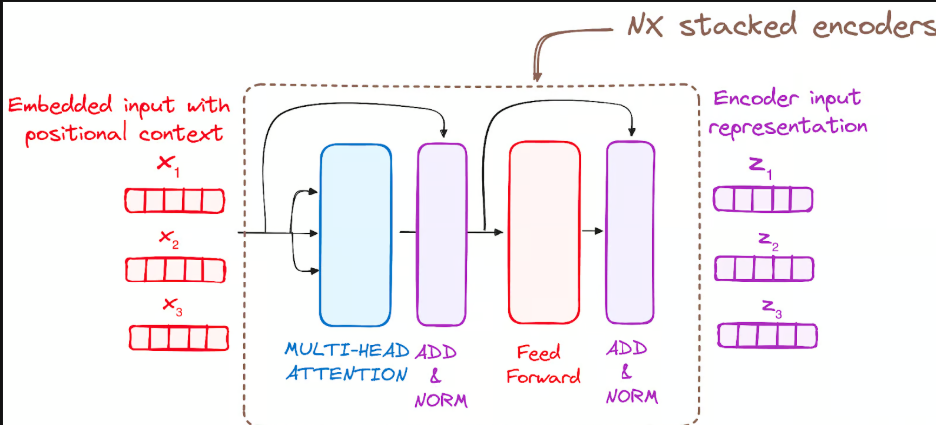
****

**4.3) STEP 3 - Stack of Encoder Layers**

The Transformer encoder consists of a stack of identical layers (6 in the original Transformer model). The encoder layer serves to transform all input sequences into a continuous, abstract representation that encapsulates the learned information from the entire sequence. This layer comprises two sub-modules:

* A multi-headed attention mechanism.
* A fully connected network.

Additionally, it incorporates residual connections around each sublayer, which are then followed by layer normalization.

****

**4.3.1) Multi-Headed Self-Attention Mechanism**

In the encoder, the multi-headed attention utilizes a specialized attention mechanism known as self-attention. This approach enables the models to relate each word in the input with other words. For instance, in a given example, the model might learn to connect the word “are” with “you”.

This mechanism allows the encoder to focus on different parts of the input sequence as it processes each token. It computes attention scores based on:

* A query is a vector that represents a specific word or token from the input sequence in the attention mechanism.
* A key is also a vector in the attention mechanism, corresponding to each word or token in the input sequence.
* Each value is associated with a key and is used to construct the output of the attention layer. When a query and a key match well, which basically means that they have a high attention score, the corresponding value is emphasized in the output.

This first Self-Attention module enables the model to capture contextual information from the entire sequence. Instead of performing a single attention function, queries, keys and values are linearly projected h times. On each of these projected versions of queries, keys and values the attention mechanism is performed in parallel, yielding h-dimensional output values.

The detailed architecture is given as follows:-

A diagram of a graph

AI-generated content may be incorrect.

Once the query, key, and value vectors are passed through a linear layer, a dot product matrix multiplication is performed between the queries and keys, resulting in the creation of a score matrix. The score matrix establishes the degree of emphasis each word should place on other words. Therefore, each word is assigned a score in relation to other words within the same time step. A higher score indicates greater focus.

This process effectively maps the queries to their corresponding keys.

A diagram of a diagram

AI-generated content may be incorrect.

**4.3.2) Normalization and Residual Connections**

Each sub-layer in an encoder layer is followed by a normalization step. Also, each sub-layer output is added to its input (residual connection) to help mitigate the vanishing gradient problem, allowing deeper models. This process will be repeated after the Feed-Forward Neural Network too.

A diagram of a multi-head attention

AI-generated content may be incorrect.

**4.3.3) Feed-Forward Neural Network**

The journey of the normalized residual output continues as it navigates through a pointwise feed-forward network, a crucial phase for additional refinement. Picture this network as a duo of linear layers, with a ReLU activation nestled in between them, acting as a bridge. Once processed, the output embarks on a familiar path: it loops back and merges with the input of the pointwise feed-forward network. This reunion is followed by another round of normalization, ensuring everything is well-adjusted and in sync for the next steps.

A diagram of a graph

AI-generated content may be incorrect.

**4.4) STEP 4 - Output of the Encoder**

The output of the final encoder layer is a set of vectors, each representing the input sequence with a rich contextual understanding. This output is then used as the input for the decoder in a Transformer model. This careful encoding paves the way for the decoder, guiding it to pay attention to the right words in the input when it's time to decode.

Think of it like building a tower, where you can stack up *N* encoder layers. Each layer in this stack gets a chance to explore and learn different facets of attention, much like layers of knowledge. This not only diversifies the understanding but could significantly amplify the predictive capabilities of the transformer network.

**5) The Decoder WorkFlow**

The decoder's role centers on crafting text sequences. Mirroring the encoder, the decoder is equipped with a similar set of sub-layers. It boasts two multi-headed attention layers, a pointwise feed-forward layer, and incorporates both residual connections and layer normalization after each sub-layer.

A diagram of a flowchart

AI-generated content may be incorrect.

The final of the decoder's process involves a linear layer, serving as a classifier, topped off with a softmax function to calculate the probabilities of different words.

The Transformer decoder has a structure specifically designed to generate this output by decoding the encoded information step by step. It is important to notice that the decoder operates in an autoregressive manner, kickstarting its process with a start token. It cleverly uses a list of previously generated outputs as its inputs, in tandem with the outputs from the encoder that are rich with attention information from the initial input. This sequential dance of decoding continues until the decoder reaches a pivotal moment: the generation of a token that signals the end of its output creation.

**5.1) STEP 1 - Output Embeddings**

At the decoder's starting line, the process mirrors that of the encoder. Here, the input first passes through an embedding layer.

**5.2) STEP 2 - Positional Encoding**

Following the embedding, again just like the decoder, the input passes by the positional encoding layer. This sequence is designed to produce positional embeddings. These positional embeddings are then channeled into the first multi-head attention layer of the decoder, where the attention scores specific to the decoder’s input are meticulously computed.

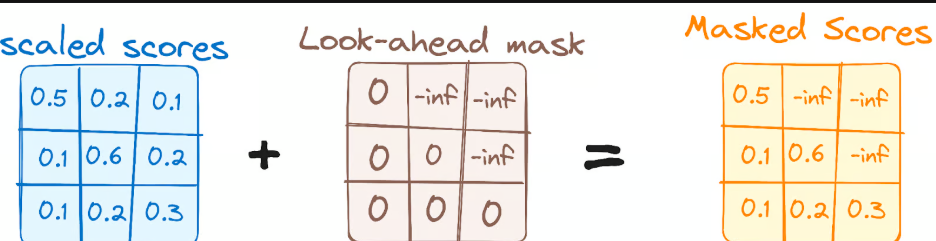
**5.3) STEP 3 - Stack of Decoder Layers**

The decoder consists of a stack of identical layers (6 in the original Transformer model). Each layer has three main sub-components:-

**5.3.1) Masked Self-Attention Mechanism**

This is similar to the self-attention mechanism in the encoder but with a crucial difference: it prevents positions from attending to subsequent positions, which means that each word in the sequence isn't influenced by future tokens.

For instance, when the attention scores for the word "are" are being computed, it's important that "are" doesn't get a peek at "you", which is a subsequent word in the sequence.

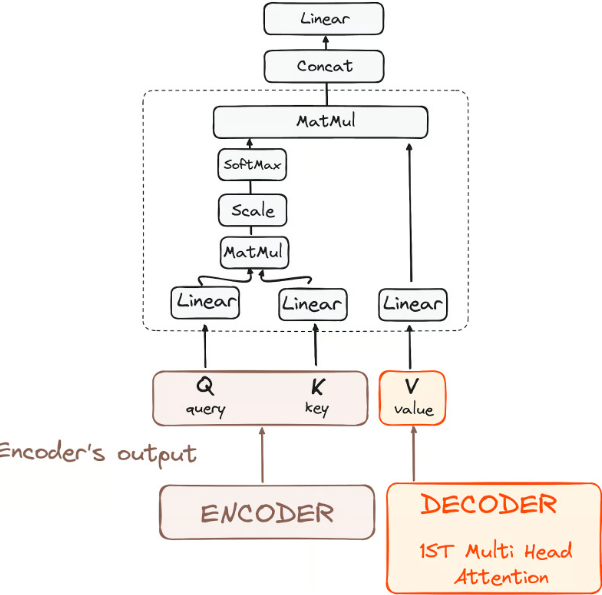


This masking ensures that the predictions for a particular position can only depend on known outputs at positions before it.

**5.3.2) Encoder-Decoder Multi-Head Attention or Cross Attention**

In the second multi-headed attention layer of the decoder, we see a unique interplay between the encoder and decoder's components. Here, the outputs from the encoder take on the roles of both queries and keys, while the outputs from the first multi-headed attention layer of the decoder serve as values.

This setup effectively aligns the encoder's input with the decoder's, empowering the decoder to identify and emphasize the most relevant parts of the encoder's input. Following this, the output from this second layer of multi-headed attention is then refined through a pointwise feedforward layer, enhancing the processing further.

****

In this sub-layer, the queries come from the previous decoder layer, and the keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence, effectively integrating information from the encoder with the information in the decoder.

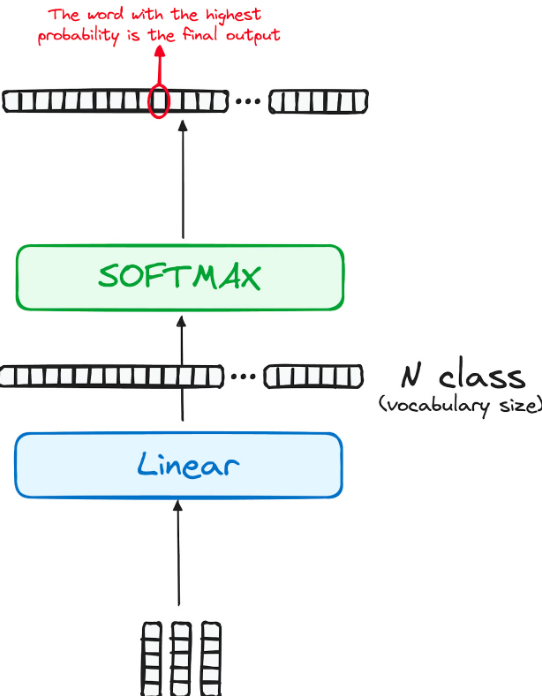
**5.3.3) Feed-Forward Neural Network**

Similar to the encoder, each decoder layer includes a fully connected feed-forward network, applied to each position separately and identically.

**5.4) STEP 4 - Linear Classifier and Softmax for Generating Output Probabilities**

The journey of data through the transformer model culminates in its passage through a final linear layer, which functions as a classifier.

The size of this classifier corresponds to the total number of classes involved (number of words contained in the vocabulary). For instance, in a scenario with 1000 distinct classes representing 1000 different words, the classifier's output will be an array with 1000 elements. This output is then introduced to a softmax layer, which transforms it into a range of probability scores, each lying between 0 and 1. The highest of these probability scores is key,its corresponding index directly points to the word that the model predicts as the next in the sequence.

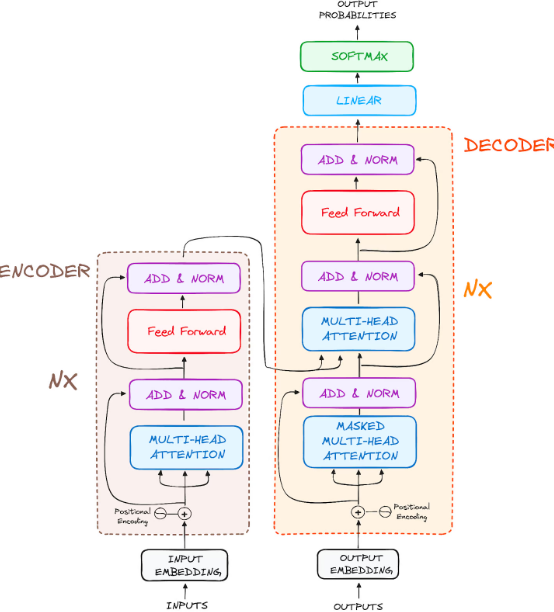


**5.5) STEP 4 - Output of the Decoder**

The final layer's output is transformed into a predicted sequence, typically through a linear layer followed by a softmax to generate probabilities over the vocabulary. The decoder, in its operational flow, incorporates the freshly generated output into its growing list of inputs, and then proceeds with the decoding process. This cycle repeats until the model predicts a specific token, signaling completion. The token predicted with the highest probability is assigned as the concluding class, often represented by the end token.

Again remember that the decoder isn't limited to a single layer. It can be structured with N layers, each one building upon the input received from the encoder and its preceding layers. This layered architecture allows the model to diversify its focus and extract varying attention patterns across its attention heads. Such a multi-layered approach can significantly enhance the model’s ability to predict, as it develops a more nuanced understanding of different attention combinations.

And the final architecture is something similar like this (form the original paper):-



**5.6) Some examples of the Transformers based architectures**

* BERT
* LaMDA
* GPT and ChatGPT

### 6) Tasks

**Task1:**  
Understand and implement the self-attention mechanism.

**Task 2:**

Understand and implement the feed forward neural network.

**Task 3:**

**Explore and implement the positional encoding.**

**Task 4:**

Analyze the composition and structure of a full Transformer block, and implement it completely by calling all the previous implemented functions in the correct order.

**Task 5:**

Understand the flow of the code, and draw the block digram/flowchart to explain the flow.

**Task 6:**

**M**odify the test data a little bit and study the impact of transformer's architecture on that changed data and explain the change in your words that you witness.

**Note:-** For the details of the tasks and basic structure of the code, please refer to the attached notebook.

|  |  |
| --- | --- |
| **7) Additional Resources**   * <https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452/> * <https://medium.com/data-science/transformers-89034557de14> |  |