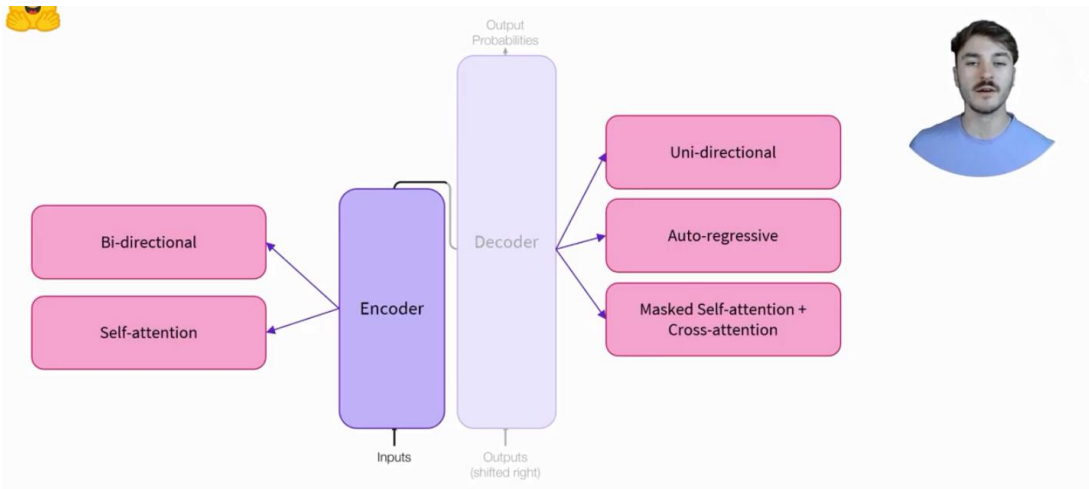


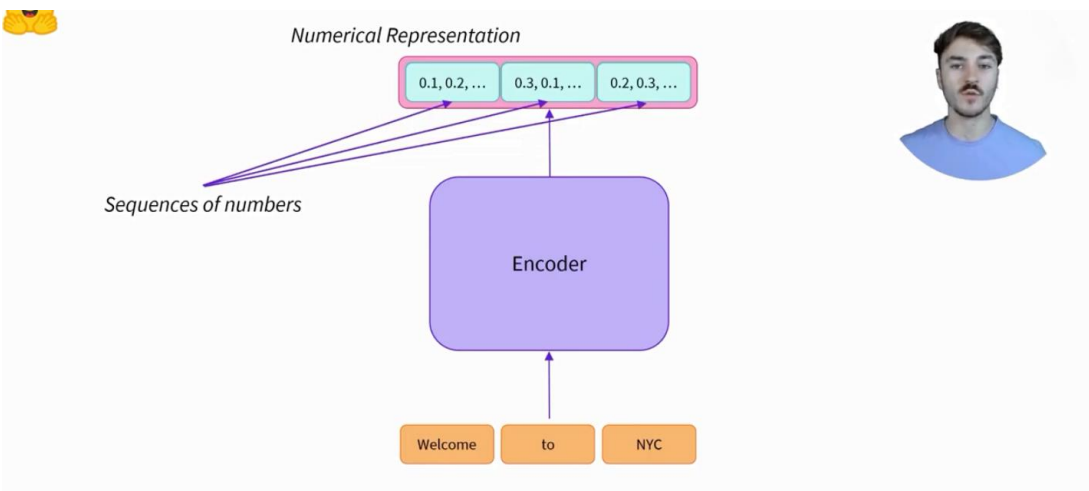
Encoders

How do they work?

A general high-level introduction to the Encoder part of the Transformer architecture. What is it, when should you use it? BERT is a popular encoder-only architecture.



Let's dive in the encoder



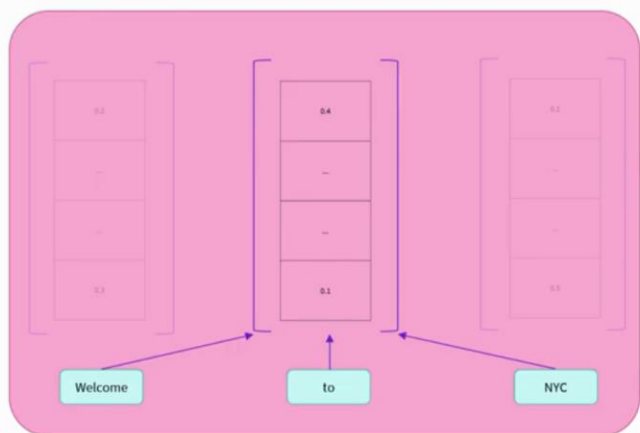
The encoder outputs a numerical representation for each word used as input

The encoder outputs one sequence of numbers per input word, this is called a feature vector or feature tensor.



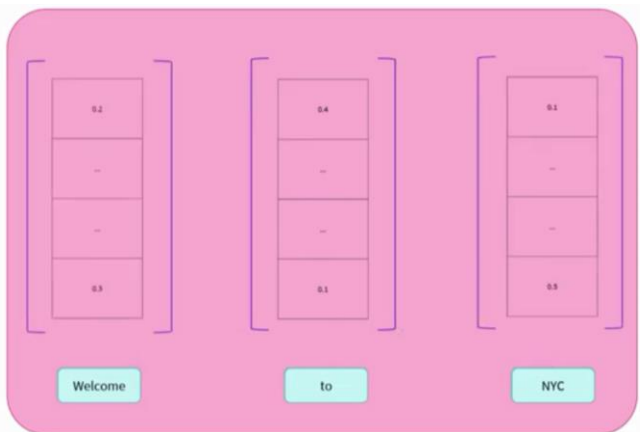
This representation is made up of a vector of values for each word of the initial sequence

The dimension of each word vector is defined by the architecture of the model, for the base BERT model it is 768.



Each word in the initial sequence affects every word's representation

These representations contain the value of the word but contextualized, the representation of the word 'to' above also takes the surrounding left and right words into its context.



This is thanks to the self-attention mechanism!

The self-attention mechanism relates to different positions of different words in a single sequence in order to compute a representation of that sequence. This means that the resulting representation of a word has been affected by other words in the sequence.

- Bi-directional: context from the left, and the right
- Good at extracting meaningful information
- Sequence classification, question answering, masked language modeling
- NLU: Natural Language Understanding
- Example of encoders: BERT, RoBERTa, ALBERT

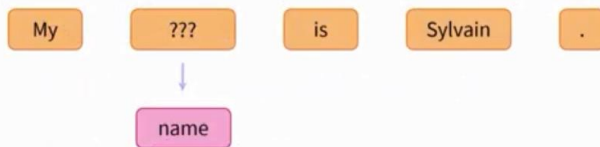


Why would one use an encoder?

Encoders are very powerful at extracting vectors that carry meaningful information about a sequence, this vector can then be handled down the road by additional neurons to make sense of them.

Masked Language Modeling

Guessing a randomly masked word

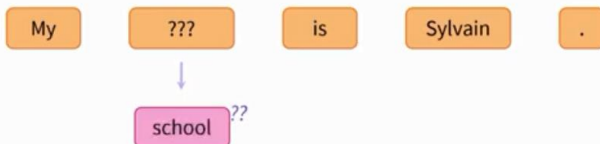


Encoders, with their bi-directional context, are good at guessing words in the middle of a sequence

MLM is the task of predicting a hidden word in a sequence of words. BERT was trained to predict hidden words in a sequence of words. Encoders work great here since bidirectional words around the missing word are crucial in this task.

Masked Language Modeling

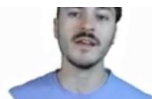
Guessing a randomly masked word



This requires a semantic understanding as well as a syntactic understanding

Sentiment analysis

Analyze the sentiment of a sequence



Even though I am sad to see them go, I couldn't be more grateful.

Positive

I am sad to see them go, I can't be grateful.

Negative

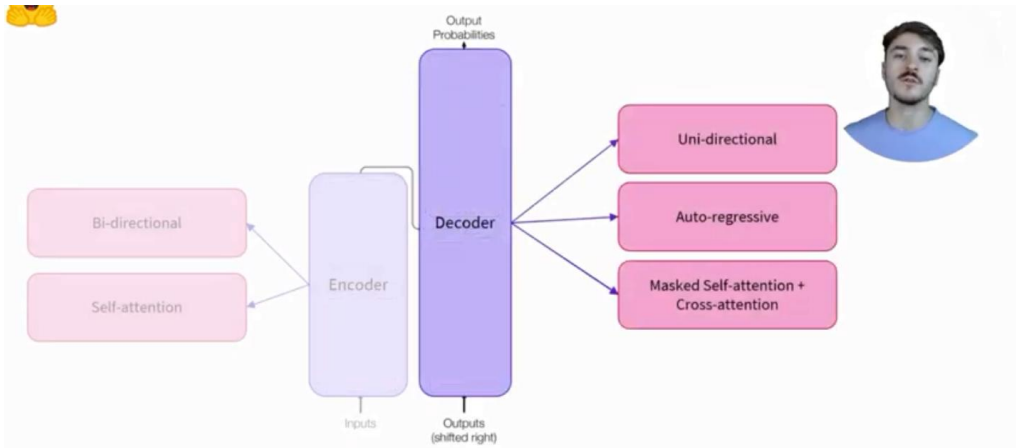
Encoders are good at obtaining an understanding of sequences; and the relationship/interdependence between words.

We will use the model to make a prediction that classifies the sequences among the 2 classes possible.

Decoders

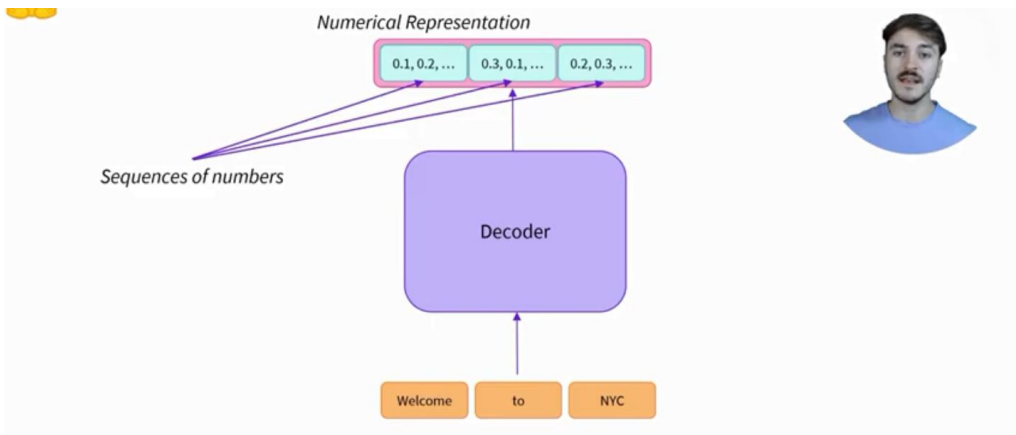
How do they work?

Let us now study the decoder architecture, a popular decoder-only architecture is GPT-2

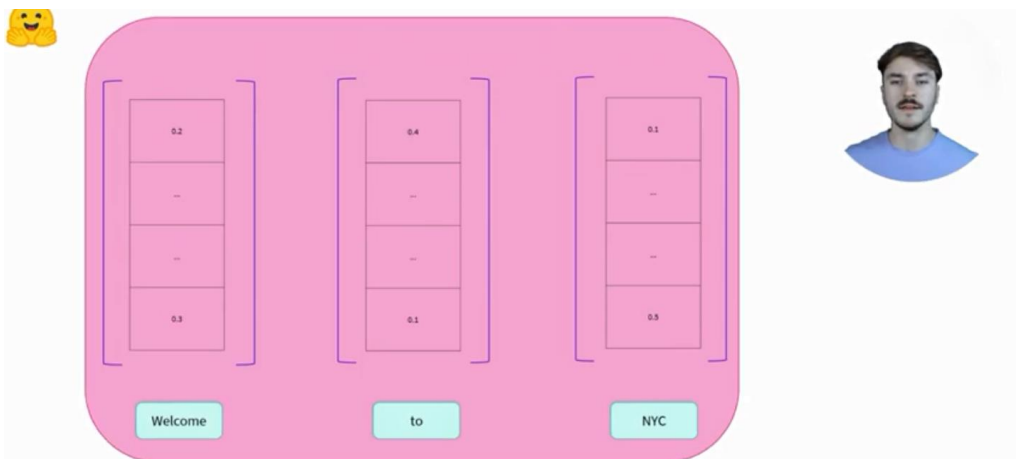


Let's dive in the decoder

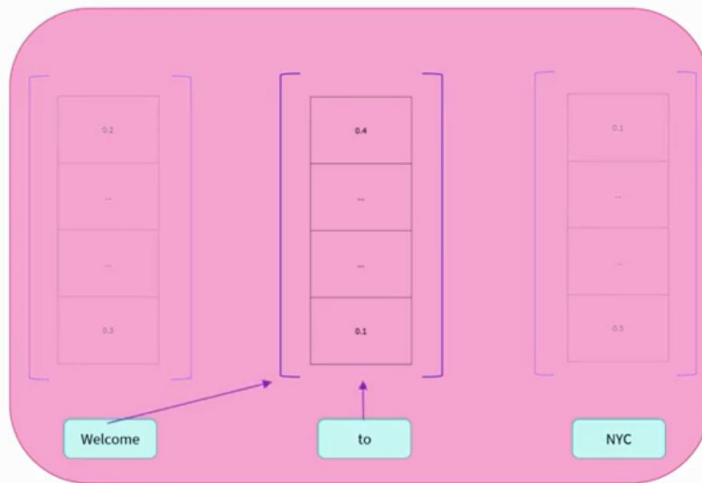
One can use a decoder for most of the same tasks as an encoder with a little loss of performance.



The decoder outputs numerical representation from an initial sequence



This feature tensor is a made up of a vector of values for each word of the initial sequence



Words can only see the words on their left side; the right side is hidden!

A decoder differs from an encoder in its self-attention mechanism, it uses a Masked Self-Attention because decoders only have access to a single context and are not bidirectional and can only look at the left of the word.



The masked self-attention layer hides the values of context on the right

- Unidirectional: access to their left (or right!) context
- Great at causal tasks; generating sequences
- NLG: Natural Language generation
- Example of decoders: GPT-2, GPT Neo

When should I use a decoder?

The strength of a decoder lies in the way the word can only have access to its left context, this explains the decoders good ability at text generation (NLG) for generating a word or a known sequence of words when given some words.

Causal Language Modeling

Guessing the next word in a sentence

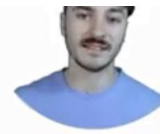


Decoders, with their uni-directional context, are good at generating words given a context

We start with an initial word which **My** as the input to the decoder model, the model outputs a vector of numbers that contains information about the sequence (which is a single word in the above example!). we apply a small transformation to that vector so that it maps to all the words known by the model which is a mapping that's called a language modelling head that identifies the word that the model thinks are probably the next sequence like **name**.

Causal Language Modeling

Guessing the next word in a sentence



Decoders, with their uni-directional context, are good at generating words given a context

We then take that new word and add it to the initial sequence of words to get **My name** as the next decoder input. This is where the **auto-regressive** nature comes in by using their past outputs as inputs in the following steps. Once again, we do the exact same operation by passing that sequence through the decoder and retrieve the most probable following words.

Causal Language Modeling

Guessing the next word in a sentence



Decoders, with their uni-directional context, are good at generating words given a context

We repeat the operation until we are satisfied.

Causal Language Modeling

Guessing the next word in a sentence



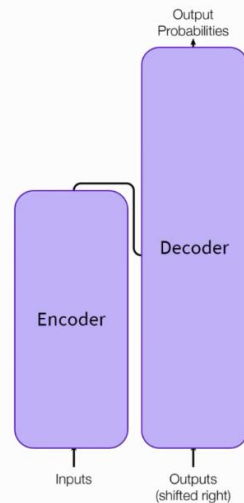
Decoders, with their uni-directional context, are good at generating words given a context

We have now generated a full sentence; we could continue for a while and can generate up to 1024 words and the decoder will still have some memory about the first word in the sequence. GPT-2 model has a maximum context size of 1024.

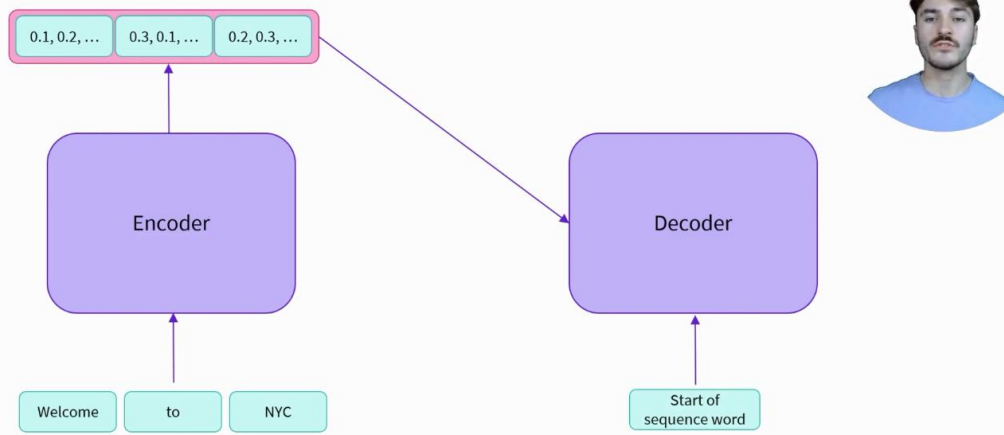
Encoder-decoders

How do they work?

A general high-level introduction to the Encoder-Decoder, or sequence-to-sequence models using the Transformer architecture. What is it, when should you use it? An example of a popular encoder-decoder model is the T5.

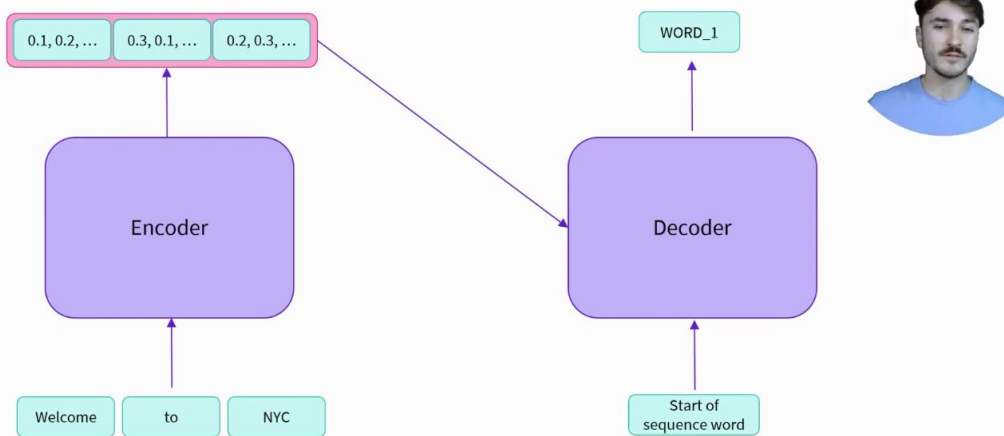


Let's dive in the encoder-decoder!

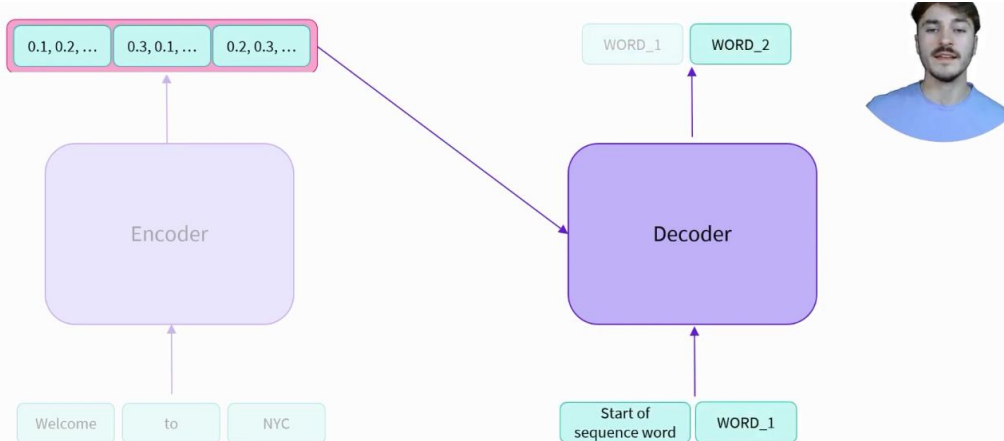


That representation is now used as an input for the decoder

We are passing the output of the encoder as inputs to the decoder, we also give the decoder an additional sequence to indicate the start of a sequence.

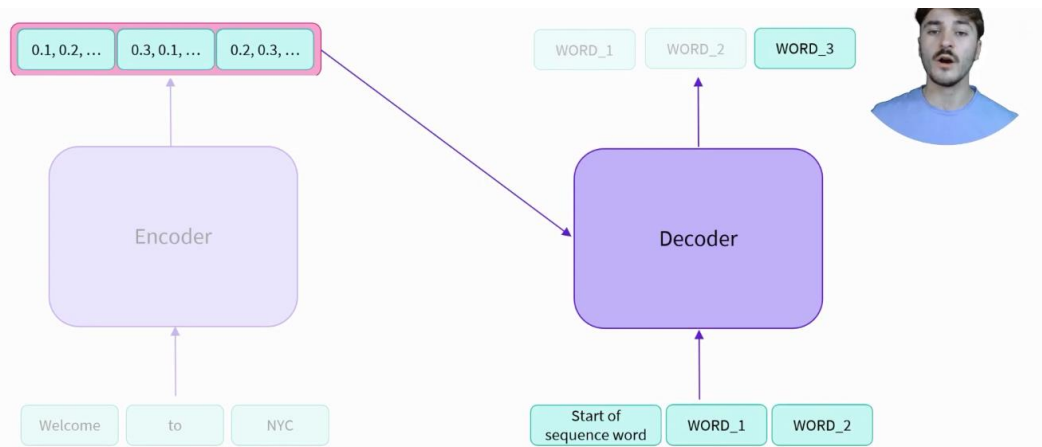


Using this representation and a prompt as input, the decoder generates a word



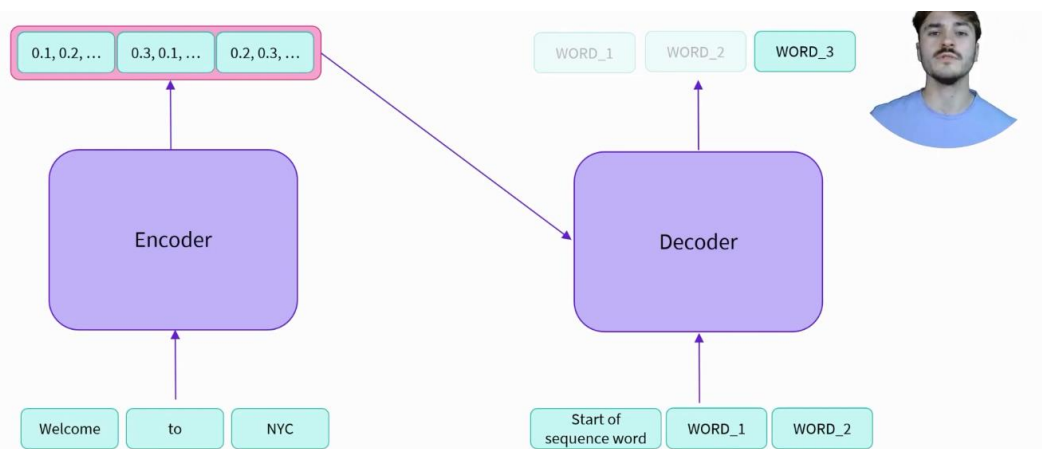
It then uses a combination of the representation and the word it just generated to generate a second word

We don't need the encoder anymore because the decoder can act in an auto-regressive manner to generate the next words in a sequence.

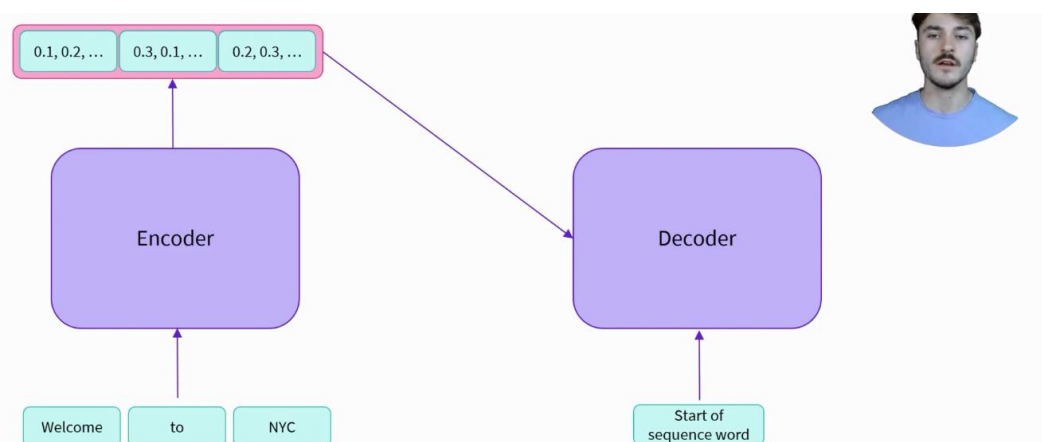


And a third!

We can continue until the decoder outputs a value that we consider as a stopping value like a **full-stop**.

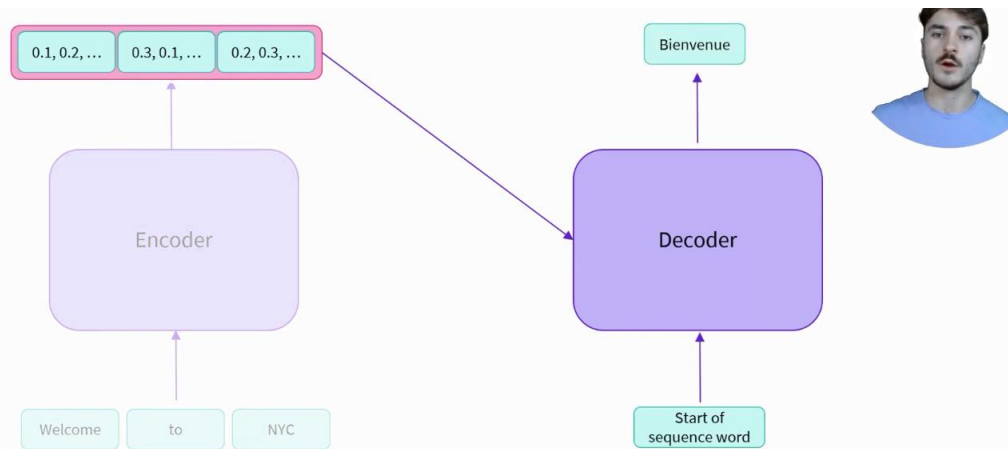


And that's the encoder-decoder!

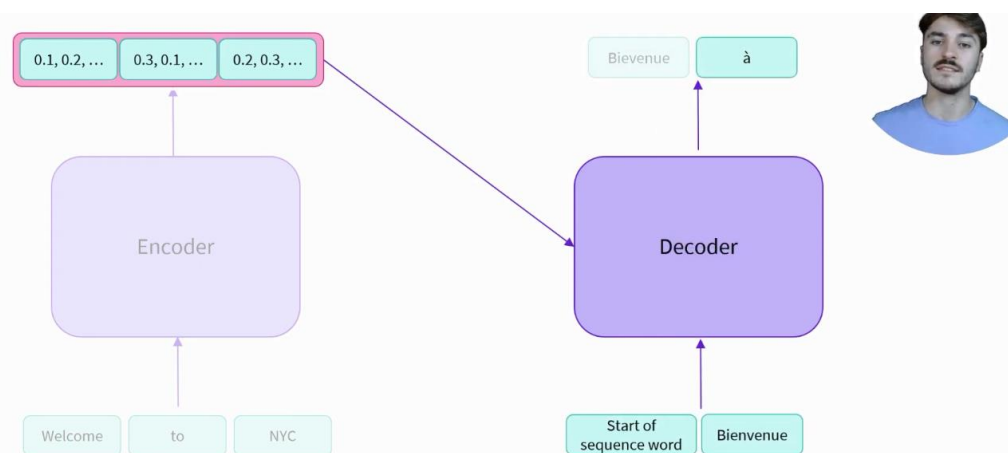


Let's go over it one more time using translation as an example

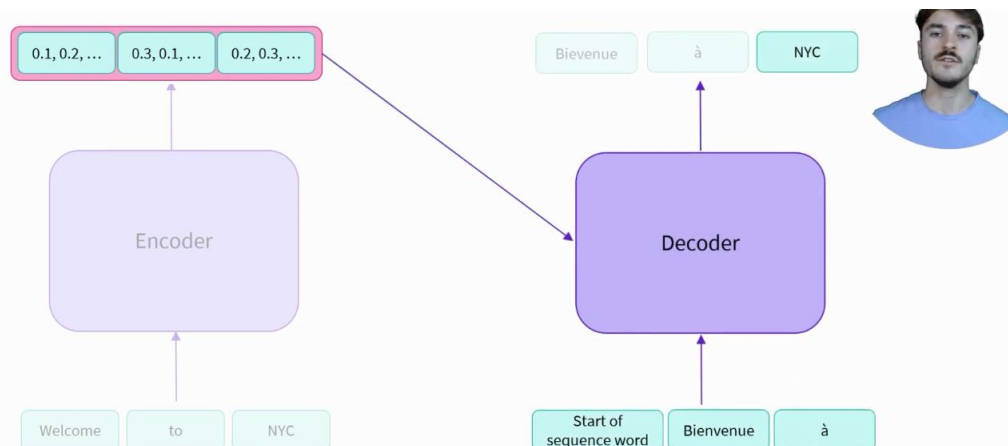
Let us take the example of translating a sequence from English to French. We will use a transformer model that is trained for that task explicitly. We first use the encoder to create the representation of the English sequence



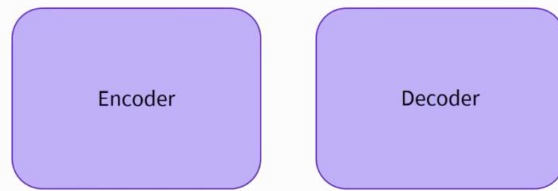
Given the representation of the English sentence "Welcome to NYC", the decoder generates the first word: the French word "Bienvenue"



Using the representation and the first word, "Bienvenue", it is able to generate the second word: "à"

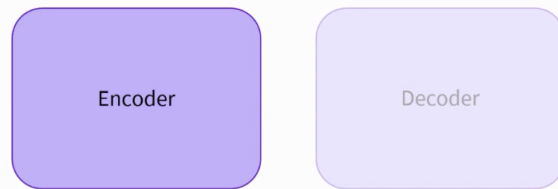


And, finally, leveraging both the encoder output and the two initial words, it can generate the third one: "NYC"



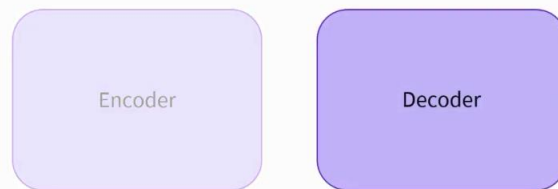
The encoder-decoder is interesting due to the separation between its two components

The encoder and decoder models often don't share weights



The encoder takes care of understanding the sequence

The encoder can be trained to understand the sequence and extract the relevant information by parsing into a vector



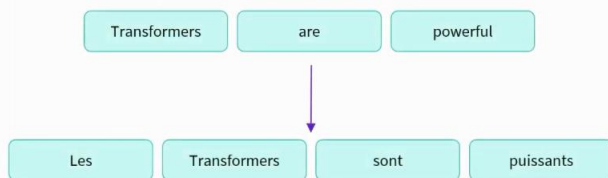
The decoder takes care of generating a sequence according to the understanding of the encoder

Trained to understand the encoder output.

- - Sequence to sequence tasks; many-to-many: translation, summarization
- - Weights are not necessarily shared across the encoder and decoder
- - Input distribution different from output distribution

When should I use a sequence-to-sequence model?

Sequence to Sequence



Translation

Output length is independent of input length in encoder-decoder models

Let us see another example of translation where we are translating a sequence of 3 words to an output of 4 words. We can use an encoder and decoder in an autoregressive manner for this task.

Sequence to Sequence



🤖 Transformers (formerly known as pytorch-transformers and pytorch-pretrained-bert) provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between TensorFlow 2.0 and PyTorch.

Transformers provides general-purpose architectures for Natural Language Understanding and Natural Language Generation.

Summarization

Sequence to sequence language models handle variable output lengths

We can also do summarization with an encoder-decoder structure where the encoder and decoder have different context lengths.

Sequence to Sequence



... and many others!

The Transformers library contains a lot of very different sequence-to-sequence models

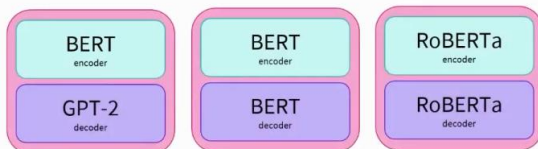
There are lots of encoder-decoder models



Sequence to Sequence



... and many others!



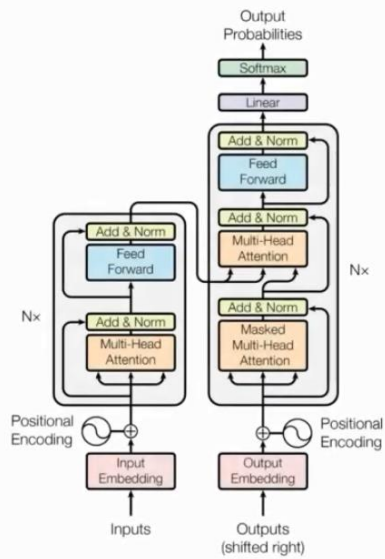
Sequence-to-sequence models can be built from separate encoders and decoders

You can also load an encoder-decoder network inside an encoder-decoder model depending on the task you have.

The Transformer architecture

Encoders, decoders, encoder-decoders

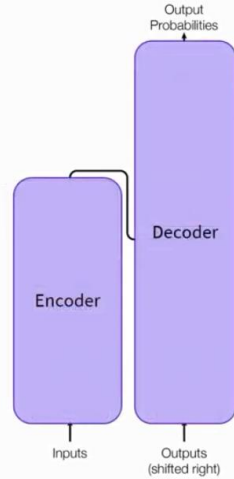
What makes a transformer network?



Attention Is All You Need

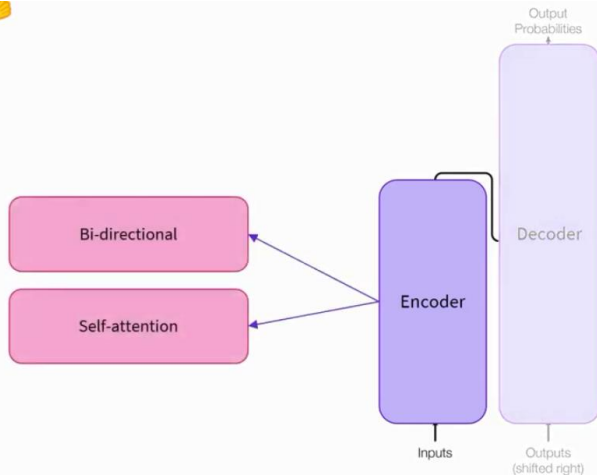
Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

The Transformer is based on the attention mechanism

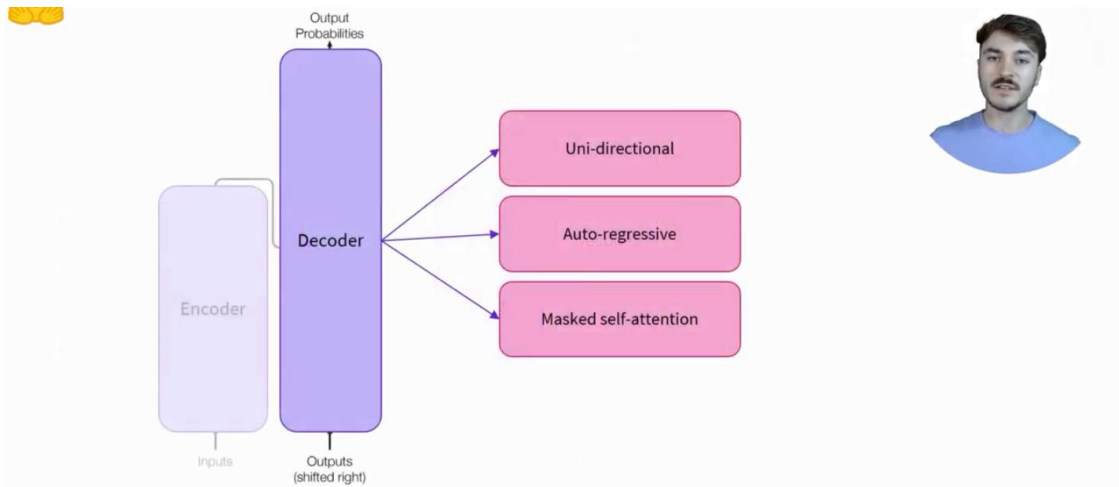


The Transformer architecture has two pieces: an encoder and a decoder

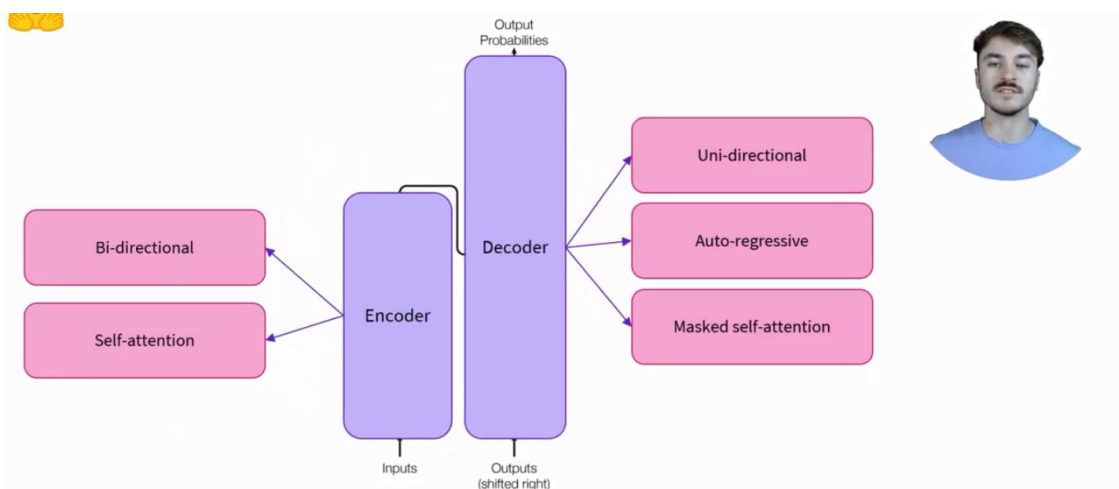
These 2 can be used together or used independently.



The encoder "encodes" text into numerical representations



The decoder "decodes" the representations from the encoder



The combination of the two parts is known as an encoder-decoder, or a sequence-to-sequence transformer

The encoder accepts text inputs and computes a high-level representation of those inputs as outputs that are then passed to the decoder. The decoder uses the encoder outputs and other inputs to generate a prediction as output to be re-used in future iterations, hence auto-regressive.