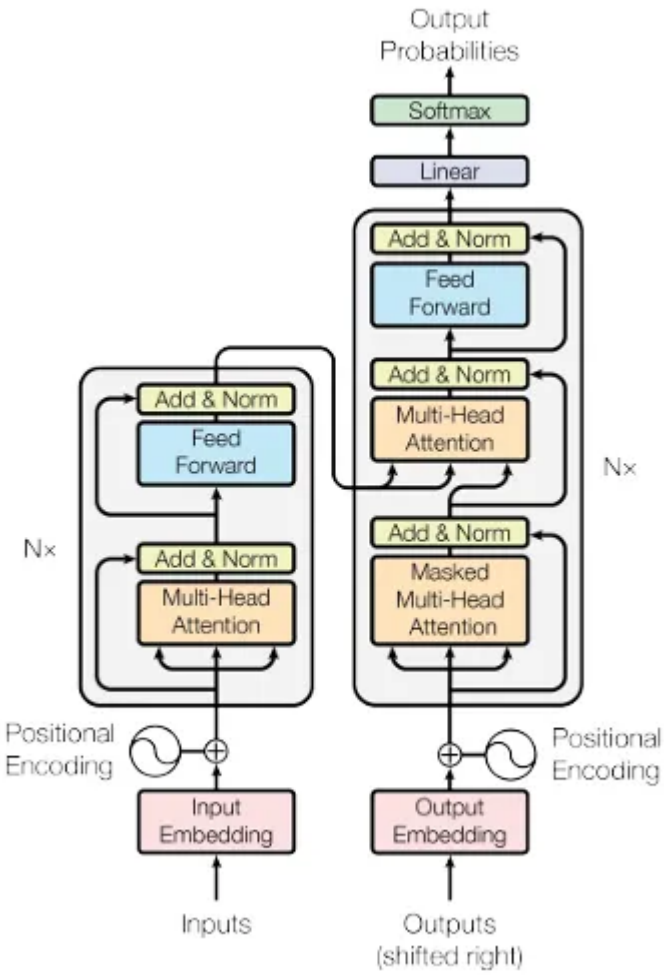
Data At The Core ! · Follow publication

How LLMs Work ? Explained in 9 Steps — Transformer Architecture

Kamna Sinha · Follow

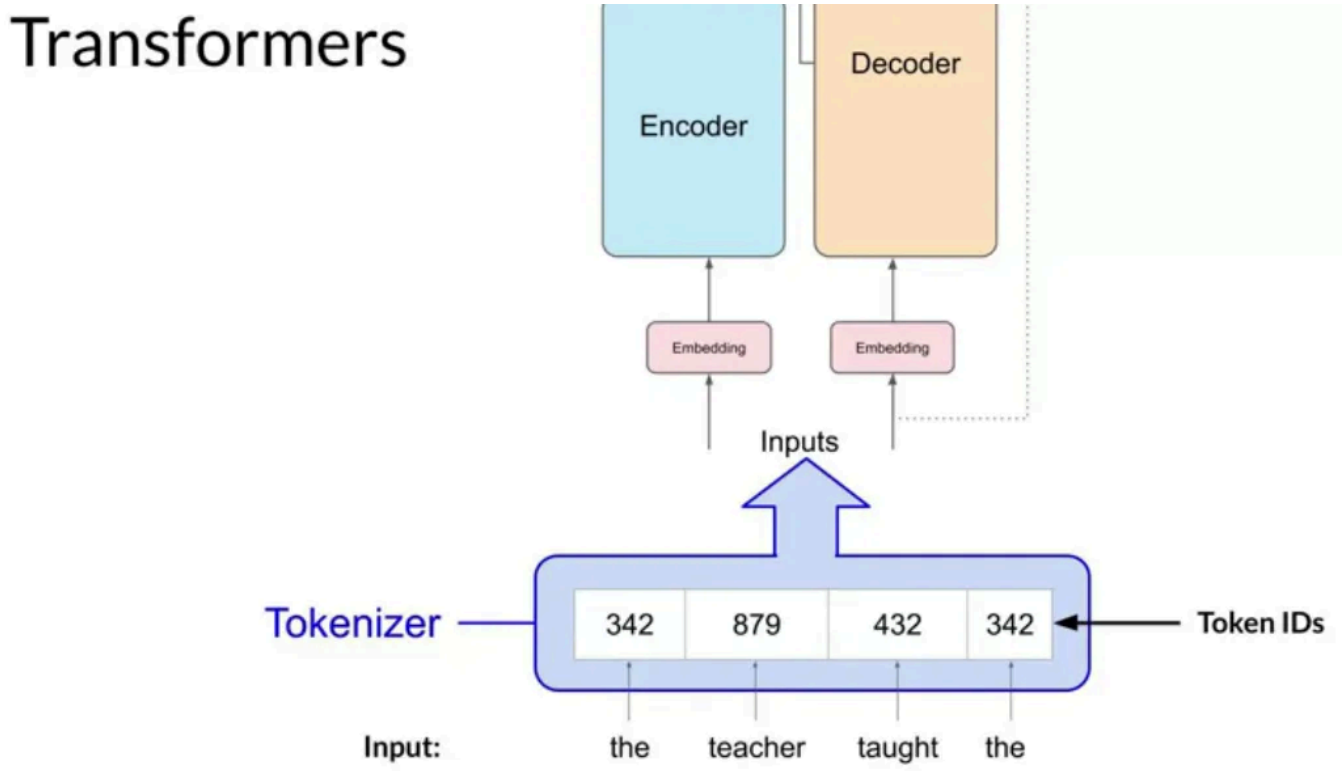
Published in Data At The Core !
4 min read · Dec 24, 2023



chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Transformer Architecture :

The power of the transformer architecture lies in its ability to learn the relevance and context of all of the words in a sentence.

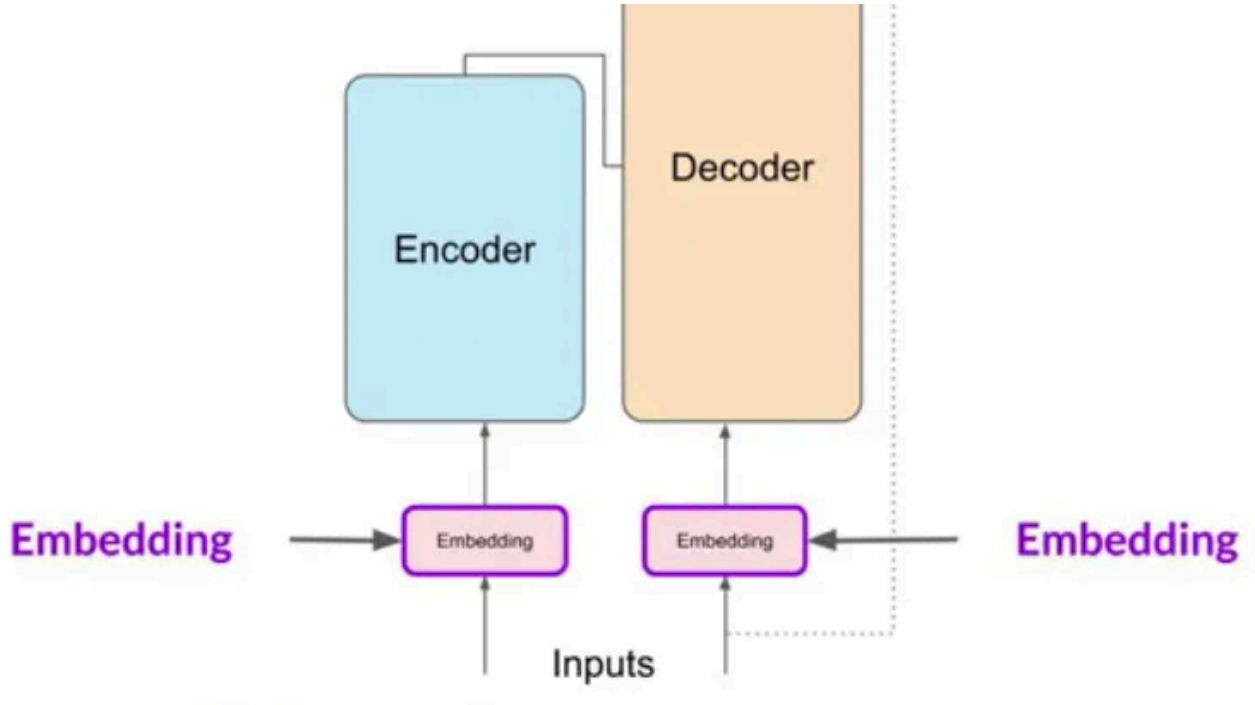


Tokenizers :

The transformer architecture is split into two distinct parts, the encoder and the decoder. These components work in conjunction with each other and they share a number of similarities.

Machine-learning models are just big statistical calculators and they work with numbers, not words. So before passing texts into the model to process, you must first tokenize the words using Tokenizers.

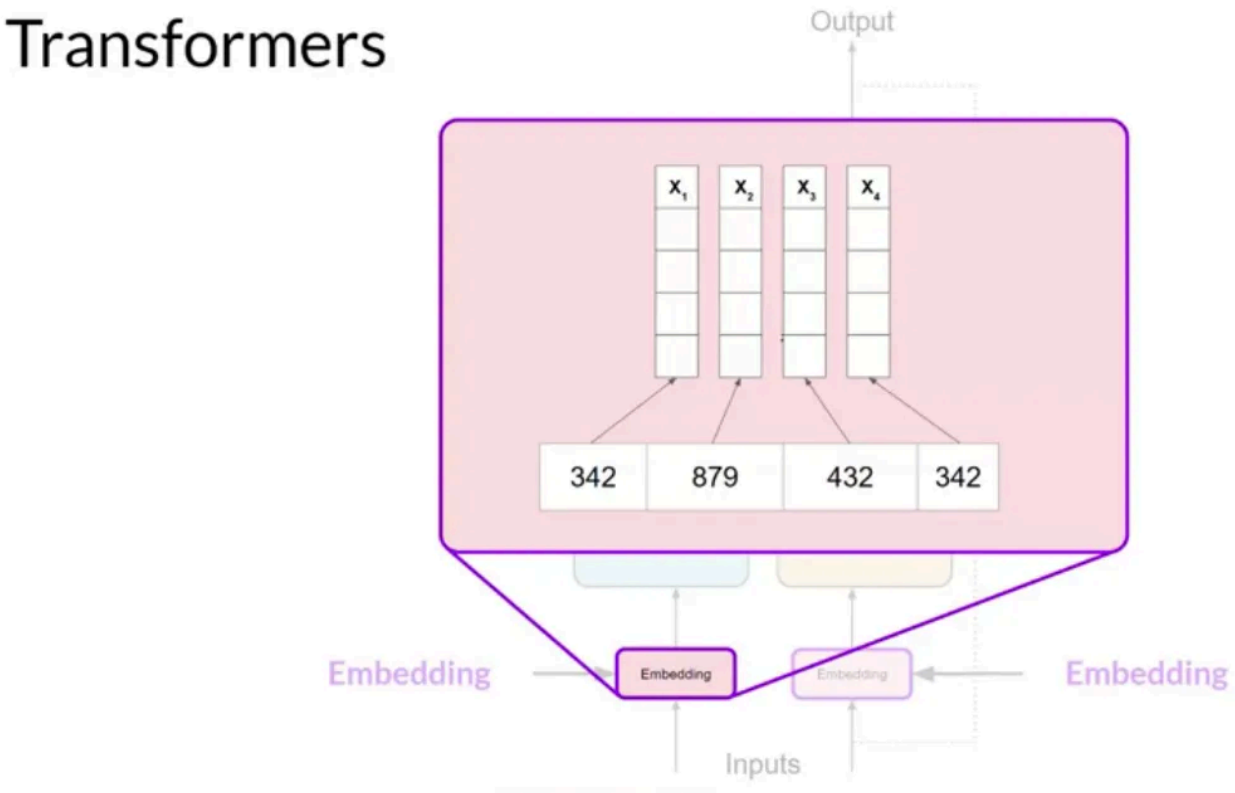
This converts the words into numbers, with each number representing a position in a dictionary of all the possible words that the model can work with.



Embeddings :

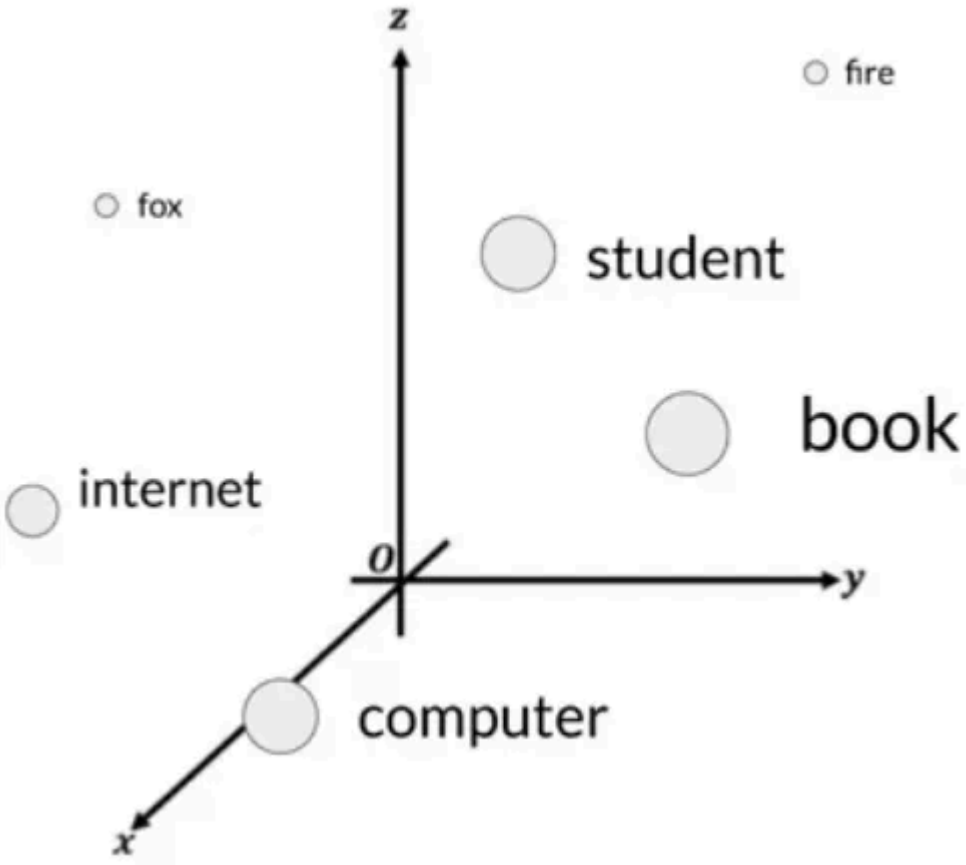
Now that your input is represented as numbers, you can pass it to the embedding layer.

Embedding vector spaces have been used in natural language processing for some time, previous generation language algorithms like Word2vec use this concept.

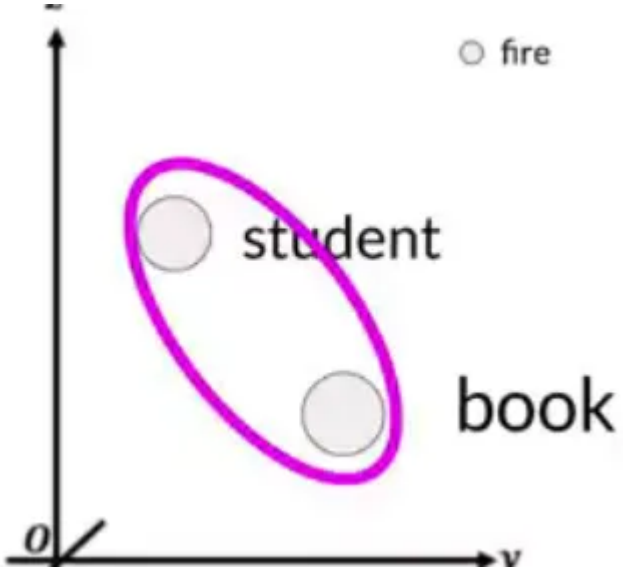


In this simple case, each word has been matched to a token ID, and each token is mapped into a vector.

If you imagine a vector size of just three, you could plot the words into a three-dimensional space and see the relationships between those words :

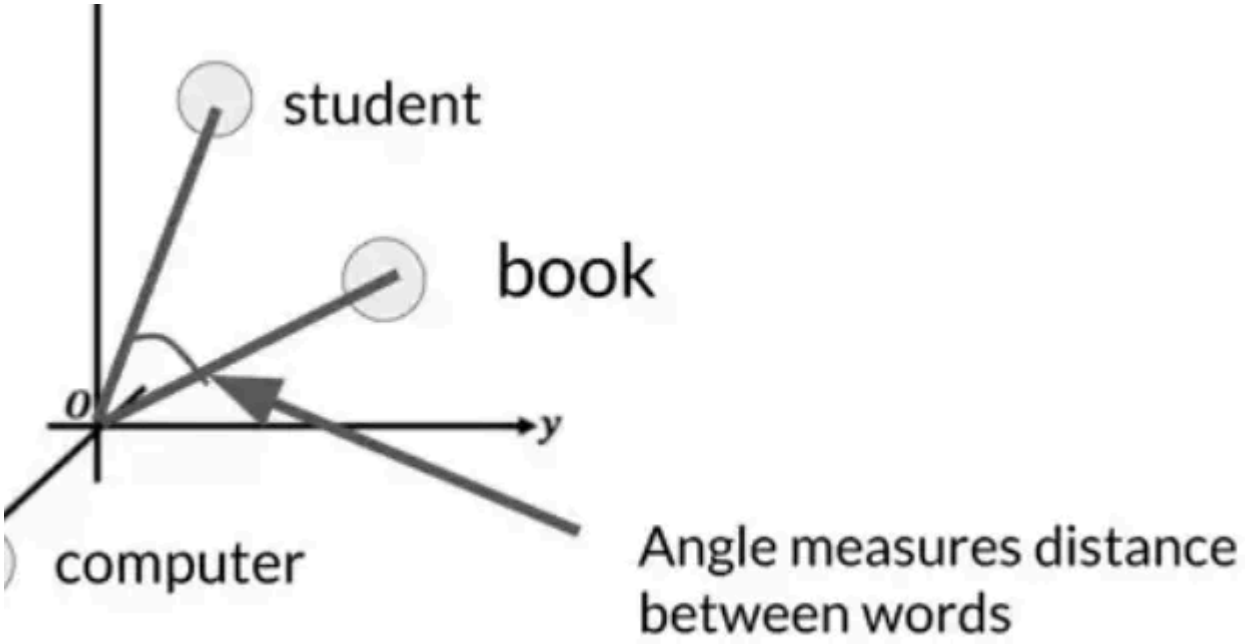


3-Dimensional vector space — a simple example for vector embeddings



related words in the embedding space

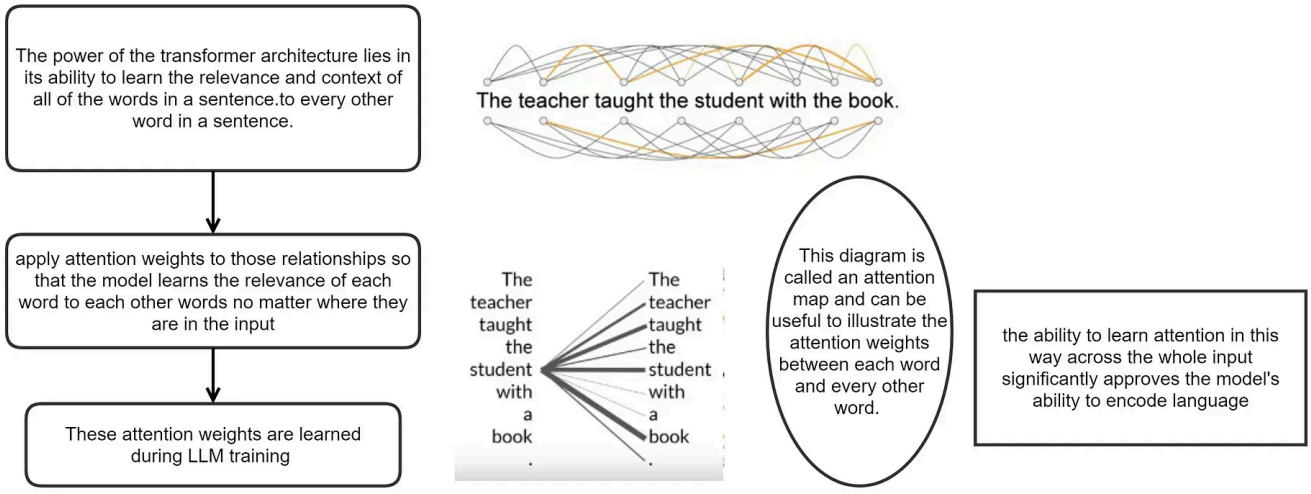
You can see now how you can relate words that are located close to each other in the embedding space, and how you can calculate the distance between the words as an angle, which gives the model the ability to mathematically understand language.



distance between the words as an angle

LLM Training

Below is a basic diagram showing how weights are assigned during LLM training so that the model understands context of language based on entire sentence and can better predict next word later.



LLM Training and Attention weights

STEPS TO GO TROUGH THIS MODEL

We will simplify the working of transformer architecture for LLMs and see in 9 steps how it makes the working of LLMs the way it is.

Step 1. before passing texts into the model to process, you must first tokenize the words.

Step 2. Once the input is represented as numbers, you can pass it to the embedding layer. This layer is a trainable vector embedding space, a high-dimensional space where each token is represented as a vector and occupies a unique location within that space.

Step 3. As you add the token vectors into the base of the encoder or the decoder, you also add positional encoding. By adding the positional encoding, you preserve the information about the word order and don't lose the relevance of the position of the word in the sentence.

Step 4. Once you've summed the input tokens and the positional encodings, you pass the resulting vectors to the self-attention layer.

Step 5. The self-attention weights that are learned during training and stored in these layers reflect the importance of each word in that input sequence to all other words in the sequence.

Step 6. Now that all of the attention weights have been applied to your input data, the output is processed through a fully-connected feed-forward network.

Step 7. The output of this layer is a vector of logits proportional to the probability score for each and every token in the tokenizer dictionary.

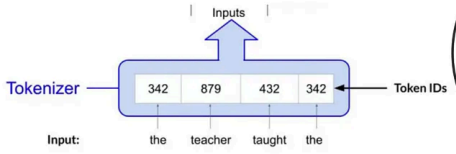
Step 8. You can then pass these logits to a final softmax layer, where they are normalized into a probability score for each word. This output includes a probability for every single word in the vocabulary, so there's likely to be thousands of scores here.

Step 9. One single token will have a score higher than the rest. This is the most likely predicted token.

The following diagram shows the above steps in order with diagramatic representation of steps for better understanding .

Step 1.

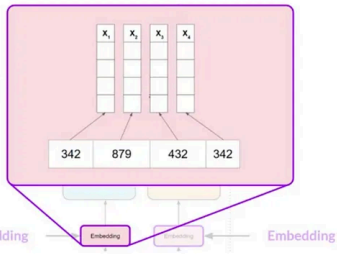
tokenize the words



once you've selected a tokenizer to train the model, you must use the same tokenizer when you generate text.

EXAMPLE

you can see that in this simple case, each word has been matched to a token ID, and each token is mapped into a vector.



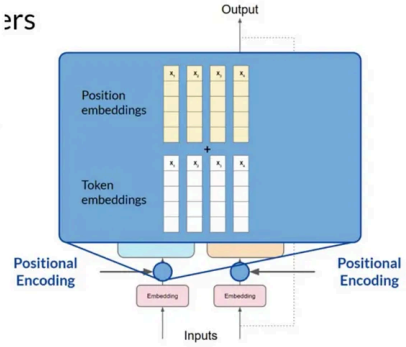
Step 2.

Tokens to vector embeddings

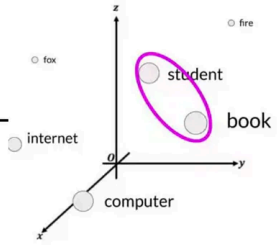
Each token ID in the vocabulary is matched to a multi-dimensional vector, and the intuition is that these vectors learn to encode the meaning and context of individual tokens in the input sequence.

Step 3.

Add Positional Encoding



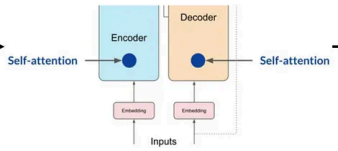
if you imagine a vector size of just three, you could plot the words into a three-dimensional space and see the relationships between those words.



you can relate words that are located close to each other in the embedding space, and how you can calculate the distance between the words as an angle, which gives the model the ability to mathematically understand language.

Step 4.

pass the resulting vectors to the self-attention layer.

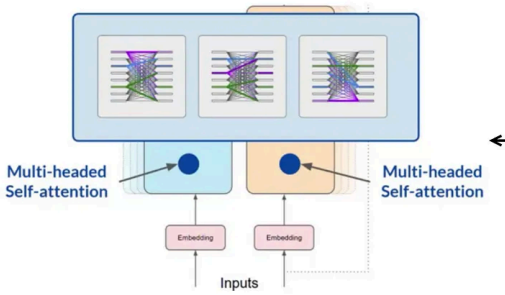


this allows the model to attend to different parts of the input sequence to better capture the contextual dependencies between the words.

Step 5.

Assign Self attention weights

But this does not happen just once, the transformer architecture actually has multi-headed self-attention.

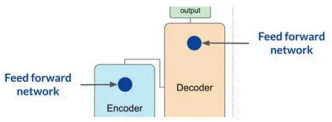


This means that multiple sets of self-attention weights or heads are learned in parallel independently of each other. The number of attention heads included in the attention layer varies from model to model, but numbers in the range of 12-100 are common.
The intuition here is that each self-attention head will learn a different aspect of language. For example, one head may see the relationship between the people entities in our sentence. Whilst another head may focus on the activity of the sentence. Whilst yet another head may focus on some other properties such as if the words rhyme.

It's important to note that you don't dictate ahead of time what aspects of language the attention heads will learn. The weights of each head are randomly initialized and given sufficient training data and time, each will learn different aspects of language.

Step 6.

process the output through a fully-connected feed-forward network.



Step 7.

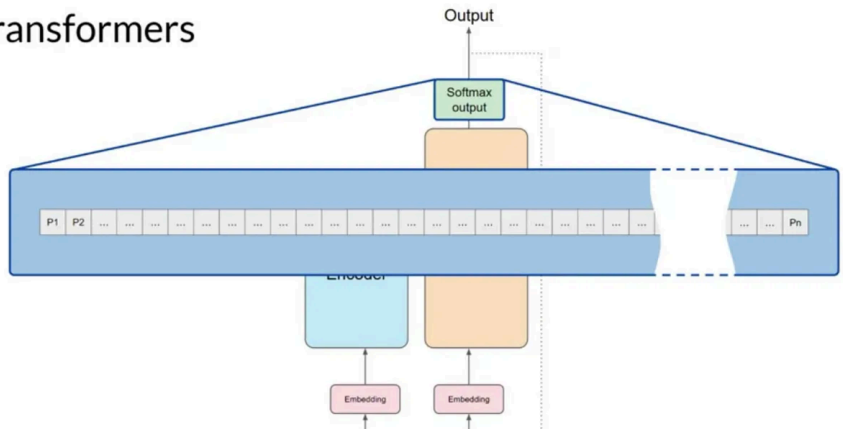
Vector output proportional to the probability score for each and every token in the tokenizer dictionary

Step 8.

Softmax layer normalizes each word into a probability score

Step 9.

Predict the word with highest token score



Inputs

Watch this space for more on LLMs.

Llm | Transformer Architecture | Generative Ai Solution | Attention Is All You Need | AI

Follow

Published in Data At The Core !

51 Followers · Last published Dec 24, 2023

Sharing my knowledge and experience from over a decade of working with Data and AI

Follow

Written by Kamna Sinha

528 Followers · 47 Following

AI | ML | Products | API | Digital Transformation | B2B SaaS https://www.linkedin.com/in/kamnasinha/

Responses (6)

F Zhangjunfelix

What are your thoughts?

Gerry Altmann
Apr 3, 2024

This was very clear. A problem my students have is in understanding how the LLM LEARNS the appropriate self-attention weights (for your Step 5) - I would love you to write something about that as clearly as you've written this! Thank you!

3 1 reply Reply

Let's Decode
Apr 15, 2024

is this for predicting next word or generating continuous words like gpt.

2 Reply

John Sundean
Apr 11, 2024

Do these models use backpropagation?

2 1 reply Reply

See all responses

More from Kamna Sinha and Data At The Core !

In Data At The Core ! by Kamna Sinha

Cleaning Data For Data Analysis—in Python with 21 examples and code.

Data cleaning is the process of identifying and correcting errors and inconsistencies in data sets so that they can be used for analysis...

Oct 6, 2023 609 4

In Data At The Core ! by Kamna Sinha

Marketing Analytics using Python—Series

As a data professional working in the field for 14 years, it has been a nonstop learning experience and a joyful and enriching one ever...

Sep 24, 2023 87 2

```
as



{'cust_id': pd.Categorical(range(n_cust))})
```

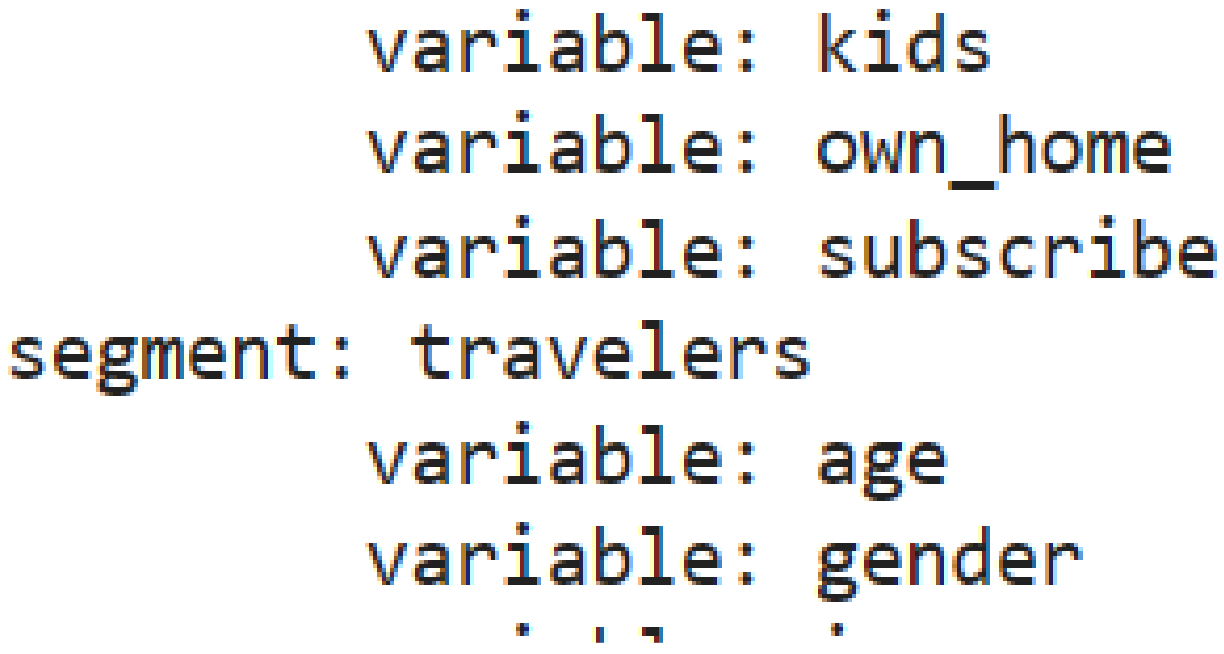
```
s/columns to dataframe - Columns in dataframes can be easily cr
r assigned to the new column has the appropriate length
dom.normal(loc=35, scale=5, size=n_cust)
 = np.random.normal(loc=3 * cust_df.age + 620,scale=50,size=n_c
ategorical(np.random.choice(a=['yes', 'no'],p=[0.8, 0.2],size=r
ore'] = np.exp(np.random.normal(loc=2,scale=1.2,size=n_cust))
```

In Data At The Core ! by Kamna Sinha

Understanding Marketing Analytics in Python.

We shall now go ahead in analyzing our data at hand which we have created and done initial analysis in previous parts of this series :

Sep 20, 2023 82



In Data At The Core ! by Kamna Sinha

Customer Segmentation with Marketing Data using Python—With 25 examples and code.

Comparing Groups: Tables and Visualizations

Sep 26, 2023 211

See all from Kamna Sinha

See all from Data At The Core !

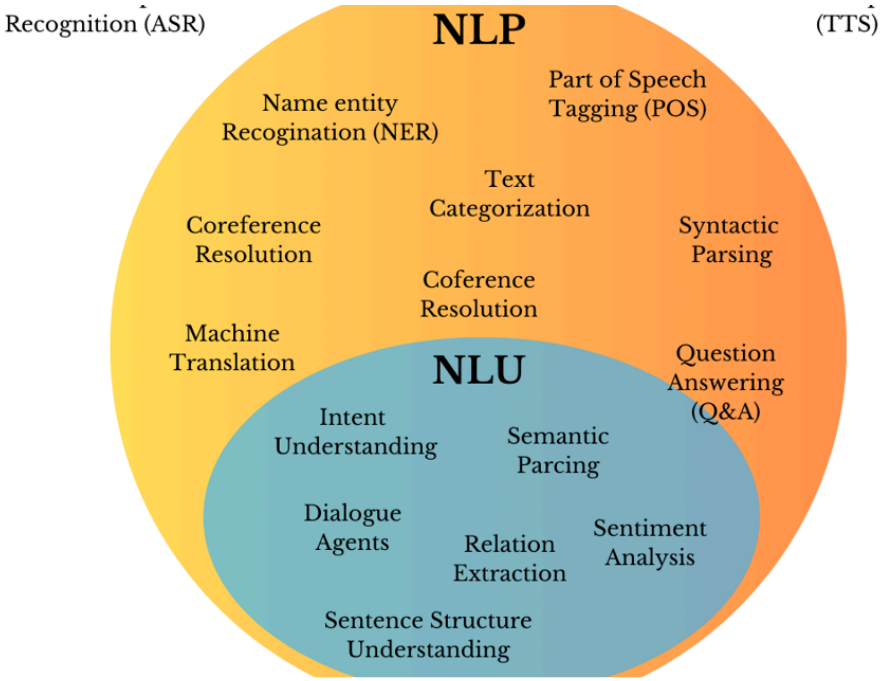
Recommended from Medium

Mark Riedl

A Very Gentle Introduction to Large Language Models without the Hype

1. Introduction

Apr 13, 2023 8.1K 129



Vipra Singh

LLM Architectures Explained: NLP Fundamentals (Part 1)

Deep Dive into the architecture & building of real-world applications leveraging NLP Models starting from RNN to the Transformers.

Aug 15, 2024 2.6K 25

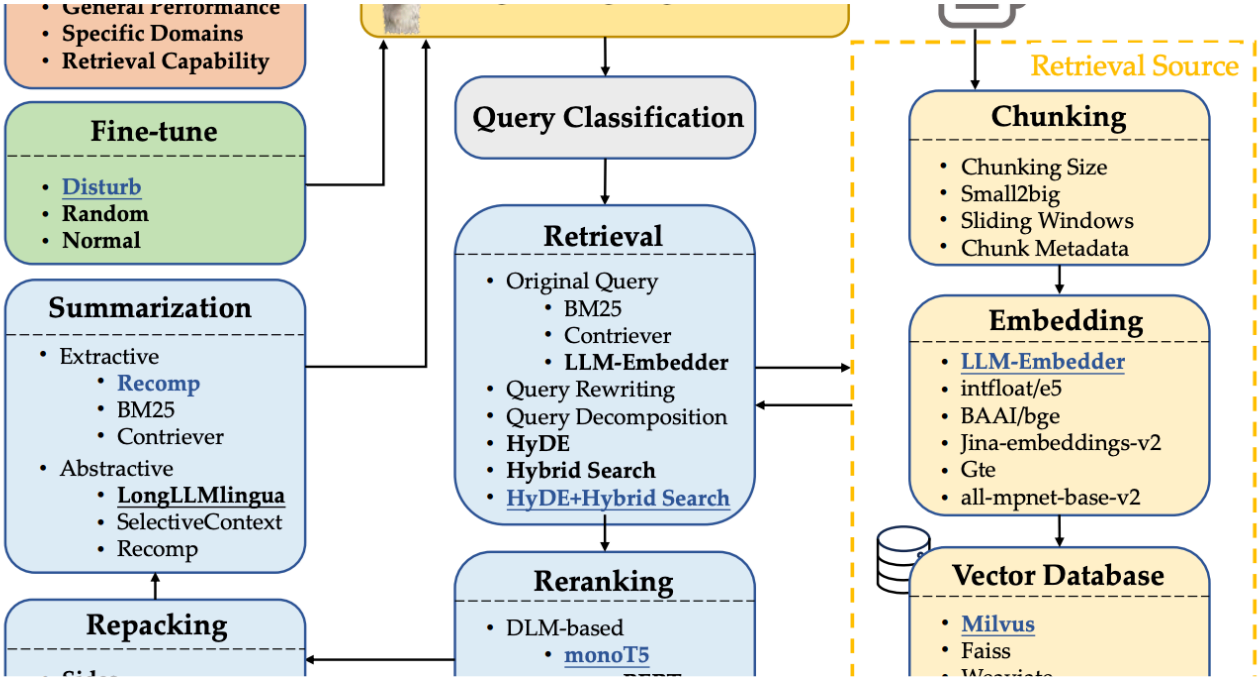
Lists

Generative AI Recommended Reading
52 stories · 1691 saves

What is ChatGPT?
9 stories · 521 saves

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 563 saves

Natural Language Processing
1977 stories · 1619 saves

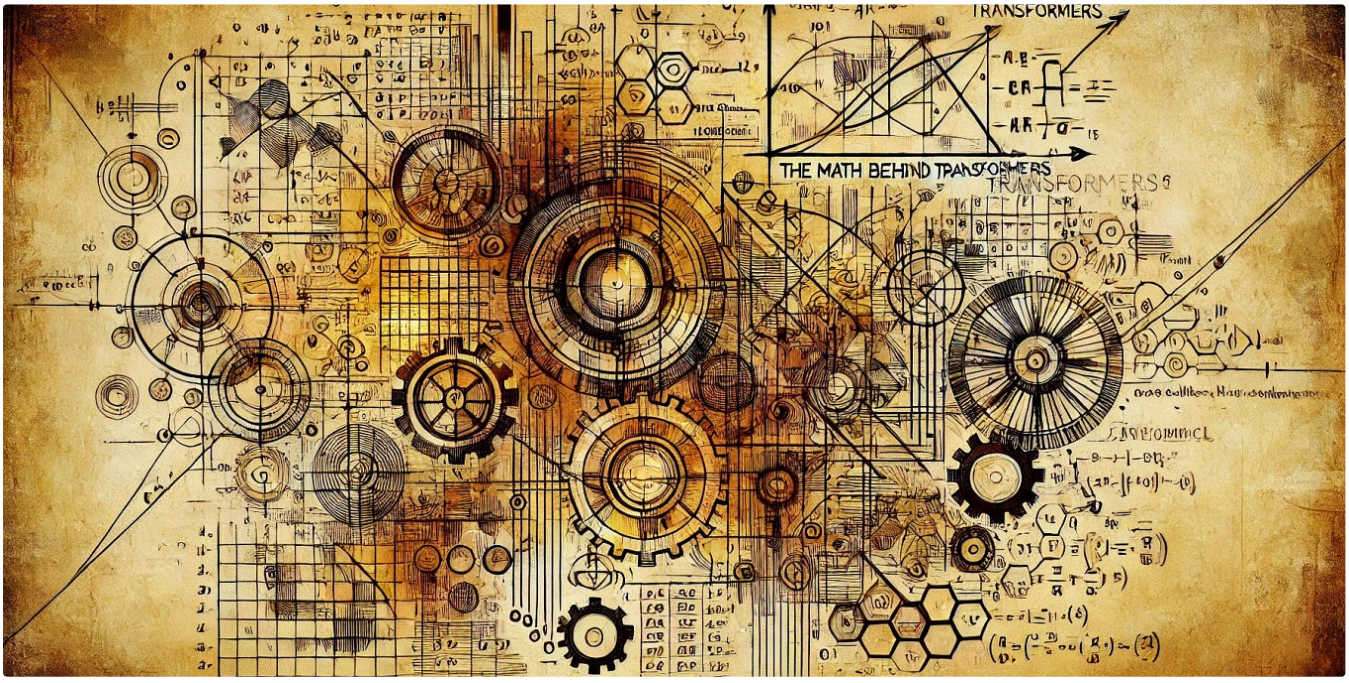


In Towards AI by Florian June

The Best Practices of RAG

Typical RAG Process, Best Practices for Each Module, and Comprehensive Evaluation

Aug 8, 2024 1.1K 10



Cristian Leo

The Math Behind Transformers

Deep Dive into the Transformer Architecture, the key element of LLMs. Let's explore its math, and build it from scratch in Python.

Jul 25, 2024 1K 9

Jessica Stillman

Jeff Bezos Says the 1-Hour Rule Makes Him Smarter. New Neuroscience Says He's Right

Jeff Bezos's morning routine has long included the one-hour rule. New neuroscience says yours probably should too.

Oct 30, 2024 25K 730

Alexander Nguyen