GenAI Learning Journey

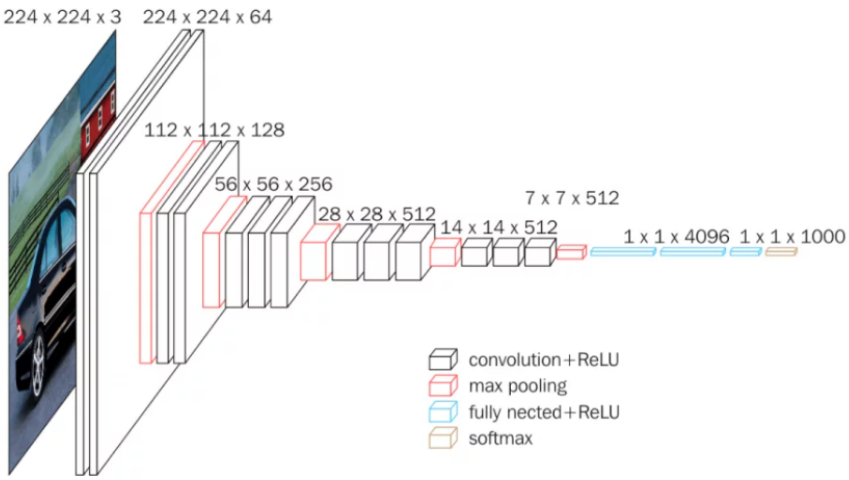
by Michael Monschke, Aug 8, 2024

Introduction

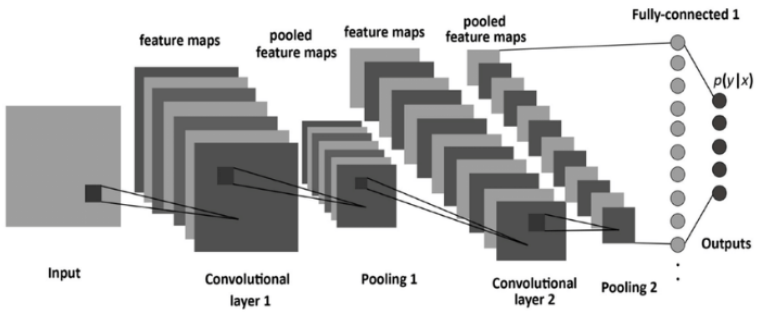
The following document explains the journey I have taken thus far in learning about GenAI, Large Language Models (LLMs), and Generative Pre-Trained Transformers (GPTs). The goal of the document is to give the reader some big picture understanding and intuition to LLMs/GPTs. My knowledge is limited in this space so I might be parroting some information I found that could be inaccurate; you have been warned. I will also provide links for the reader to research on their own to go deeper and give some suggested tracks for further learning.

Neural Networks

LLMs/GPTs are based on neural networks. I have researched neural networks in the past where you learn about topics such as forward/backward propagation, gradient descent, and activation functions. You can see the Appendix for links if interested in neural networks. But after that, there are even more advanced techniques to make deep learning work. I have learned that even if you understand the techniques, it is a far cry from understanding how the neural networks "truly" work unless you deep dive into the specific fields themselves. For example, I once was looking into how image classification worked that is based on convolutional neural networks (CNNs); the below diagram explains CNNs.



If you keep researching though, you will find out these image classification neural networks are constantly being engineered in different ways using this CNN technique, but it is way more complicated than just a CNN technique.



General configuration of a convolutional neural network (CNN).

LLMs/GPTs are just the same. They have a set of multiple techniques to make it work. And if you learn the techniques, it can give some intuition on how LLMs/GPTs work, but you cannot understand how it "truly" works unless you can get your hands on the code, the training data, documentation explaining designs such as the tokenization techniques, because it all works together cohesively. The following sections describe the techniques to create LLMs/GPTs to give you some intuition on how this all works.

Under the Hood with LLMs (get some intuition)

LLMs/GPTs use a set of techniques call tokenization, LLM vector embeddings and relative relational spacing in those vectors, sequences and next word (or token) prediction, and transformers. There are tons of good articles out there, but getting your head around it all is difficult because everyone comes from a different level of understanding, lots of techniques must align, and I found that sometimes the abstractions can create more confusion than help.

I will start by giving my own two cents on how to think of GPT/LLMs, but I will keep it as short and simple as possible because of reasons given before regarding difficulty in understanding. I will then provide a set of links for you to investigate further if interested in the Appendix.

The Core Mechanism

LLMs/GPTs work by predicting the next word (or token) in a sequence. *Note, a token could be a word, a word part, or a punctation, token examples could include as **small**, **ish**, ** (I will use token from now on, but you can just think of a token as a word).* Then, the LLMs/GPTs will append that token into the existing sequence, then use the updated sequence to generate a new token, and so on, until you get an end of response token.

It's crazy, these LLMs/GPTs are not creating meaningful responses by "thinking" meaningfully, it is a probability matrix to guess the next best token to finish a sequence of other tokens – that is it! For example, when you see 1,3,5,7,11 or 2,4,6,8, you know 13 and 10 are the next values to come; LLMs/GPTs are doing the same thing with our natural languages, with a little randomness sprinkled in.

So, how do LLMs/GPTs learn to do this. Let's explain with an example. Assume we are training with two sentences.

The capital of Texas is Austin The capital of Illinois is Springfield

The sentences will be organized to show the target token in a sequence one token at a time. The neural networks will train the LLMs/GPTs based on having a sequence of tokens such as "The capital" resulting in a target token of "of".

The
The capital
The capital of
The capital of Texas
The capital of Texas is
The capital of Texas is Austin
The
The capital
The capital of
The capital of Illinois
The capital of Illinois is
The capital of Illinois is Springfield

So, with enough of these examples, the GPT can learn that “The capital” is usually followed by the word “of”, the next best token. Also, while we haven’t talked about LLM vector embeddings or relational spacing, just note the LLM can also find associations between words based on the sentence structure. “The capital of” will usually precede some type of region name (hence, Texas and Illinois), so they related in some fashion. And Texas relates closely to Austin and Illinois relates to Springfield based on the data in the sentences themselves. The LLM doesn’t know what these things are, but it can figure out meaningful relationships based on how the words are sequenced in a sentence.

Tokenization & LLM Vector Embeddings

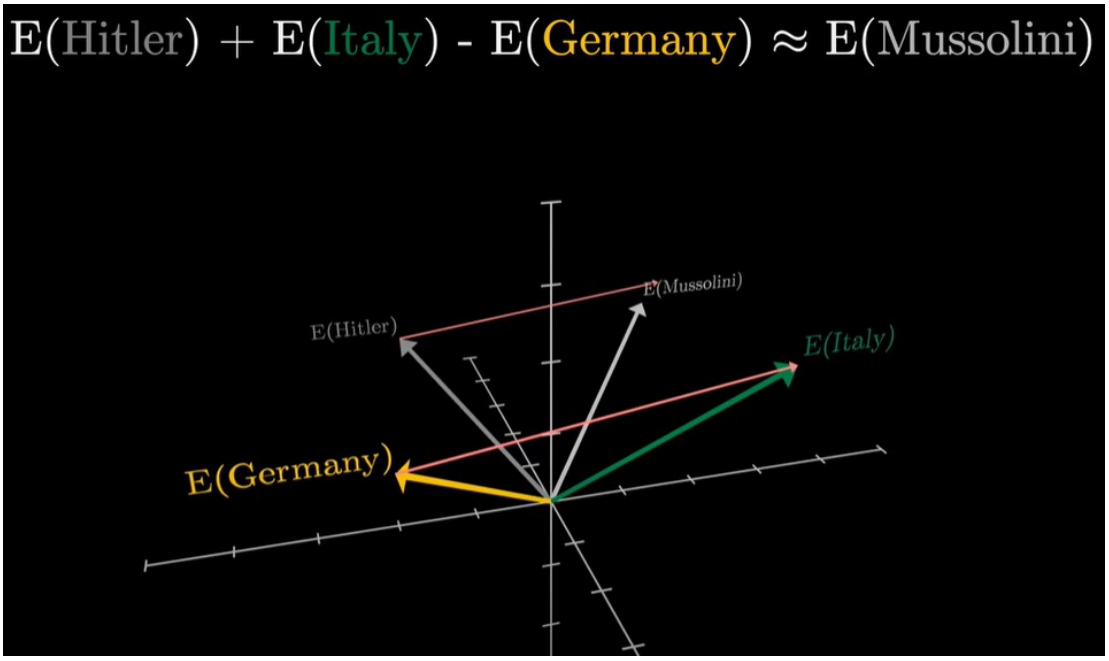
The LLMs are trained with tokens versus words. You can think of the tokens as the vocabulary of the LLM. Google uses SentencePiece and OpenAI uses tiktoken. A trade off exists in relation to having a large or small vocabulary in relation to contextual meaning overlaps and computational resources. A lot of engineering design and work has been done in this area – I didn’t investigate it much; I will provide links in Appendix if interested. The main thing to note is that the first step in LLMs/GPTs is to take chunks of text and break them into tokens appropriately, that is, break down text into a vocabulary the LLMs/GPTs understand. Finally, just keep in mind, a token of “the” just becomes a classifier defined by a number such as 38,482 (the LLMs/GPTs don’t think in “words”).

The next step is to take each token and relate that token to a vector embedding (the vector embeddings were created within the LLMs/GPTs learning cycle as well). A vector embedding is a highly dimensional vector space that provides contextual meanings to words. *OK, let me pause.* I could try to explain this, but I know this idea is hard to get across because I read really good articles and still couldn’t get it. I would suggest you watch these two videos to get a feel for it (the Andrej one will be a bit more difficult to ingest) and just see the picture below. Even with the pictures and getting the idea from the videos, it still feels a bit like magic that the magnitudes align so well through this process.

3Blue1Brown: https://www.youtube.com/watch?v=wjZofJX0v4M&t=654s

Karpathy: https://www.youtube.com/watch?v=TCH_1BHY58I&list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ&index=4

If you could follow along, 1:05:30 is the key part that makes it jump out at you (sort of cool).



Piecing the parts Together Thus Far

So, when training the LLMs/GPTs, we will need a few items to create our matrices to train the LLMs/GPTs. We have the context window and the number of tokens in a sequence that can be used to determine the next token. Let's assume we have a context window of 6 (much, much larger in reality). The data used for next token training could look like this.

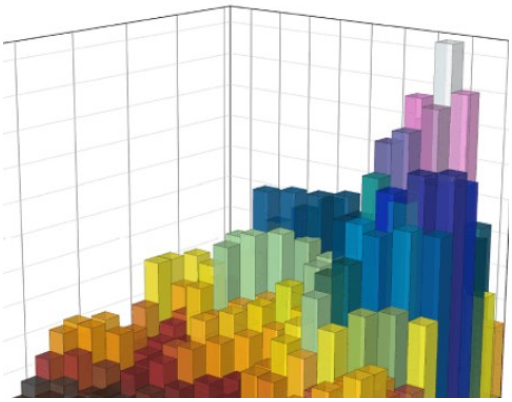
					The
				The	capital
			The	capital	of
		The	capital	of	Austin
	The	capital	of	Austin	is
The	capital	of	Austin	is	Texas

But, then, these words are actually numbers.

					1
				1	43
			1	43	987
		1	43	987	435
	1	987	987	435	22
1	987	of	435	22	4726

But then, these token numbers are translated first into a vector embedding space to represent word meaning.

We pretty much have described just getting the data into the generative engine for the LLM. Transformers come next.



Transformers

In 2017, a paper was created to help with language translations. It was something new in neural networks to beat other methods like RNN and LSTM. It took only 7 years from there to now to get where we are at with LLMs/GPTs capabilities. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

The videos in the Appendix can try to explain all this to you. The neural network is a bunch of transformers listed one after another (lots of engineering to make this work). But I "think" this is figuring out how to figure out the best next token in probabilistic manner by seeing patterns in the token sequences and knowing how to shift things around in the vector space embeddings. This is where all the magic is really happening.

But Wait, There's More

What we have discussed thus far has created a jibber/jabber machine. You can say start talking and it will generate sentences that sound right but will ultimately go all over the place. We have discussed the **Generative** and **Transformer** in GPT to get the next token, but we have not discussed the importance of **Pre-Trained** data sets. There is still a lot more engineering left to make these things work by pre-training the model well. We have discussed thus far a tool that has an opportunity to be turned into a very sophisticated template engine. The GPT engineers will now add prompts with answers to tell the LLMs/GPTs how to respond in a way we expect. For example, when the prompt is "What is the capital of Texas?", what should the next token be? Well, the LLMs/GPTs engineers can train it with examples such as:

"What is the capital of Texas? That is a great question. The capital of Texas is Austin. Austin is well-known for its "

Then, all the other word links about Austin can continue from there from jibber jabber model. But the "What is the capital of ___?" prompt will now respond in a way we expect to be conversational. Note, I totally made this example up, I have not seen anything regarding examples on how the training data makes these models return well, but I think it must work something like this. It probably is a combination of this type of pre-training data combined in conjunction with well-engineered neural networks to make these state-of-the-art systems run as well as they do.

Prompt Engineering & Tuning (learning track one)

The LLMs/GPTs models have been created by advanced techniques in neural networking along with pre-trained data to give it an idea on how to responds to prompts. The prompt is the first set of sequence of tokens from which it can carry forward from in generating the next tokens one by one. All LLMs are created differently, so you must learn how to be an expert at each one. While I am sure some ideas can be generalized, getting the detailed right responses will probably take knowledge of the specific models. I am not going into details regarding prompt engineering, but I think this is probably the most fun aspect to learn of GenAI. Prompt engineering consists of a lot of learning from others, finding any type of documentation for prompting a specific model, and trial and error experience to give you the feel (or art) on how to get the models to respond the way you want. See Appendix to get your started on this learning track if interested.

Also, you can fine tune an existing model. You can try to train model parameters with new data, but I think that will generally cause the model to lose some of its knowledge and be biased to your new data set. But there is other fine tuning methods where you freeze the existing model and build on top of it. See this link for a description regarding these strategies. https://medium.com/@sasirekharameshkumar/everything-you-need-to-know-about-fine-tuning-llms-part-3-d849f6fb0d39

RAGs (learning track two)

Retrieval Augmented Generation (RAG) sounds super fancy, but it is a straightforward concept if you understand LLMs/GPTs and the prompt engineering paradigm. You must also add to your knowledge searchable vector databases, which is something on my list to learn further, just don't confuse it with the LLM token vector embedding.

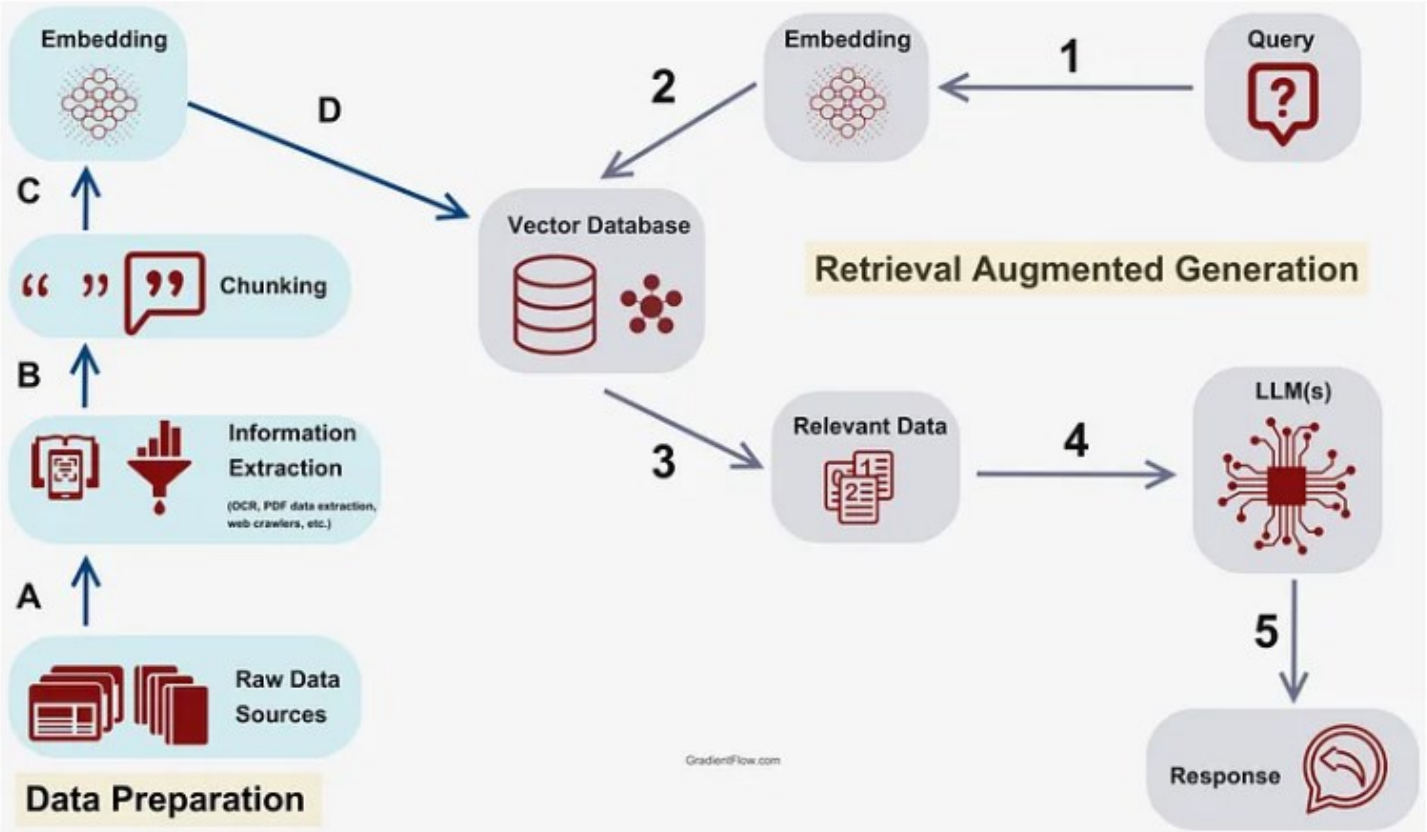
In essence, if you take documents/data that the LLM model was not trained on (say proprietary organizational data), you can turn those documents into text chunks that get identified by numbers (a vector in a vector database). When a person creates a query, you have a program find applicable documents or text chunks (via vector database search) and format those into a prompt to help give a specific query the LLMs/GPTs is not aware of. The prompt might be something like this:

DOCUMENT:
(document text gathered in early step in RAG process, post vector database search)

QUESTION:
(users original query)

INSTRUCTIONS:
Answer the users QUESTION using the DOCUMENT text above.
Keep your answer ground in the facts of the DOCUMENT.
If the DOCUMENT doesn't contain the facts to answer the QUESTION return {NONE}

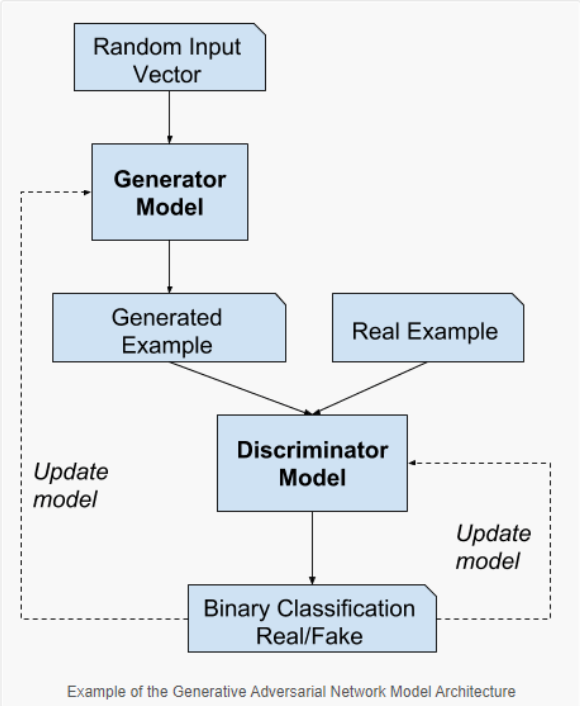
I have provided a link below to a medium article and a picture to help you understand further.



https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b

GANs (learning track three)

Generative Adversarial Networks are also straightforward to explain but making it a solution is another thing. In essence, you want to train your model to get better at generating content that seems "real". You can do this by having the generator model compete with "real" content. A separate model is trained how to separate real from generated content via supervised learning (the discriminator). The discriminator can then communicate back to the generator model that it knows it is creating fake material until it can no longer differentiate (that is, the generator model gets better and better without a person having to tell it when it is good or bad). This probably has more to do with images, video, and voice than text, but it is good to include in the conversation.



Example of the Generative Adversarial Network Model Architecture

https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

Architecture & Practice (learning track four)

I have not done a deep dive into this topic. But, privacy, security, costs, capabilities, how to pick LLMs/GPTs? Something to follow-up on if you want to be a genius in this space. Learn differences between Llama, OpenAI, Gemini, etc.

https://medium.com/@genengineerAI/choosing-the-right-large-language-model-llm-for-your-business-a-comprehensive-guide-18860ad48b20#:~:text=Selecting%20the%20Appropriate%20Model%20Size.%20When%20it,indication%20of%20its%20complexity%20and%20learning%20capacity.

https://www.solitontech.com/key-factors-to-consider-in-llm/#:~:text=Licensing.%20Licensing%20is%20pivotal%20when%20picking%20a,commercialization%2C%20guarantees%20ethical%20usage%2C%20and%20supports%20customization.

Appendix

GPT/LLM Getting Started

https://www.snowflake.com/resource/generative-ai-and-llms-for-dummies/

Under the Hood

PlayList: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Neural Video: https://www.youtube.com/watch?v=QDX-1M5Nj7s&list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI

Neural Video: https://www.youtube.com/watch?v=ySEx_Bqxvvo&t=1417s

Article: https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f

PlayList (coding): https://www.youtube.com/watch?v=VMj-3S1tku0&list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ

Tokenization Article: https://emaggiori.com/chatgpt-vocabulary/#:~:text=ChatGPT%20is%20quite%20similar%E2%80%94it,is%20known%20as%20a%20token.

Tokenization Video: https://www.youtube.com/watch?v=zduSFxRajkE

Embeddings Article: https://www.lesswrong.com/posts/pHPmMGEMYefk9jLeh/llm-basics-embedding-spaces-transformer-token-vectors-are

Additional Article: https://medium.com/towards-data-science/decoding-llms-creating-transformer-encoders-and-multi-head-attention-layers-in-python-from-scratch-631429553ce8

Prompt Engineering

https://aws.amazon.com/what-is/prompt-engineering/

https://www.reddit.com/r/PromptEngineering/comments/1bz2kc7/different_models_require_very_different_prompt/

https://platform.openai.com/docs/guides/prompt-engineering

https://community.openai.com/t/prompt-engineering-for-rag/621495/2 (context window length issues)

https://medium.com/@russkohn/mastering-ai-token-limits-and-memory-ce920630349a