NATURAL LANGUAGE PROCESSING | MACHINE LEARNING | CHAT GPT

GPT — Intuitively and Exhaustively Explained

Exploring the architecture of OpenAI's Generative Pre-trained Transformers.

Daniel Warfield

·

Follow

Published in

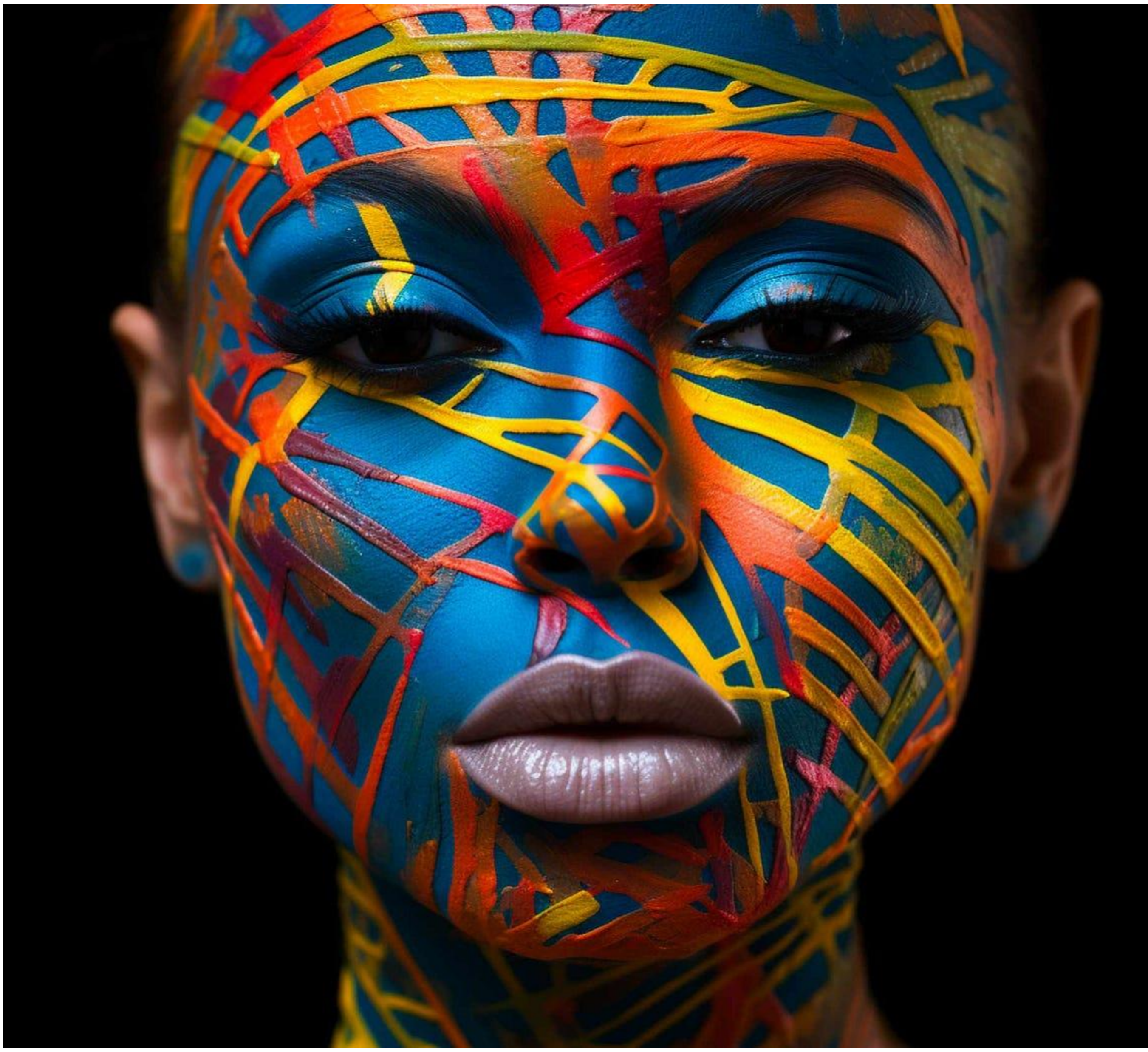
Towards Data Science

·

16 min read

·

Dec 1, 2023



“Mixture Expert” by the author using MidJourney. All images by the author unless otherwise specified.

In this article we’ll be exploring the evolution of OpenAI’s GPT models. We’ll briefly cover the transformer, describe variations of the transformer which lead to the first GPT model, then we’ll go through GPT1, GPT2, GPT3, and GPT4 to build a complete conceptual understanding of the state of the art.

Who is this useful for? Anyone interested in natural language processing (NLP), or cutting edge AI advancements.

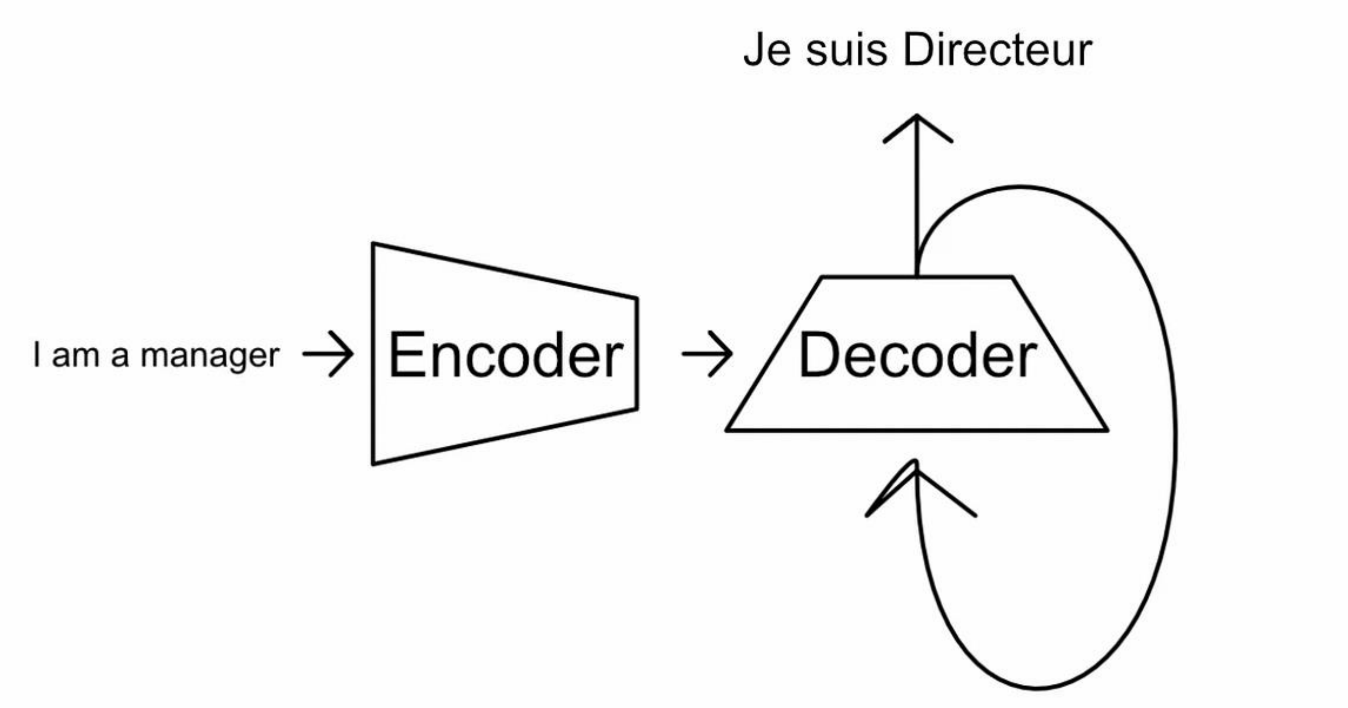
How advanced is this post? This post should be accessible to all experience levels.

Pre-requisites: I'll briefly cover transformers in this article, but you can refer to my dedicated article on the subject for more information.

Transformers — Intuitively and Exhaustively Explained
Exploring the modern wave of machine learning: taking apart the transformer step by step
towardsdatascience.com

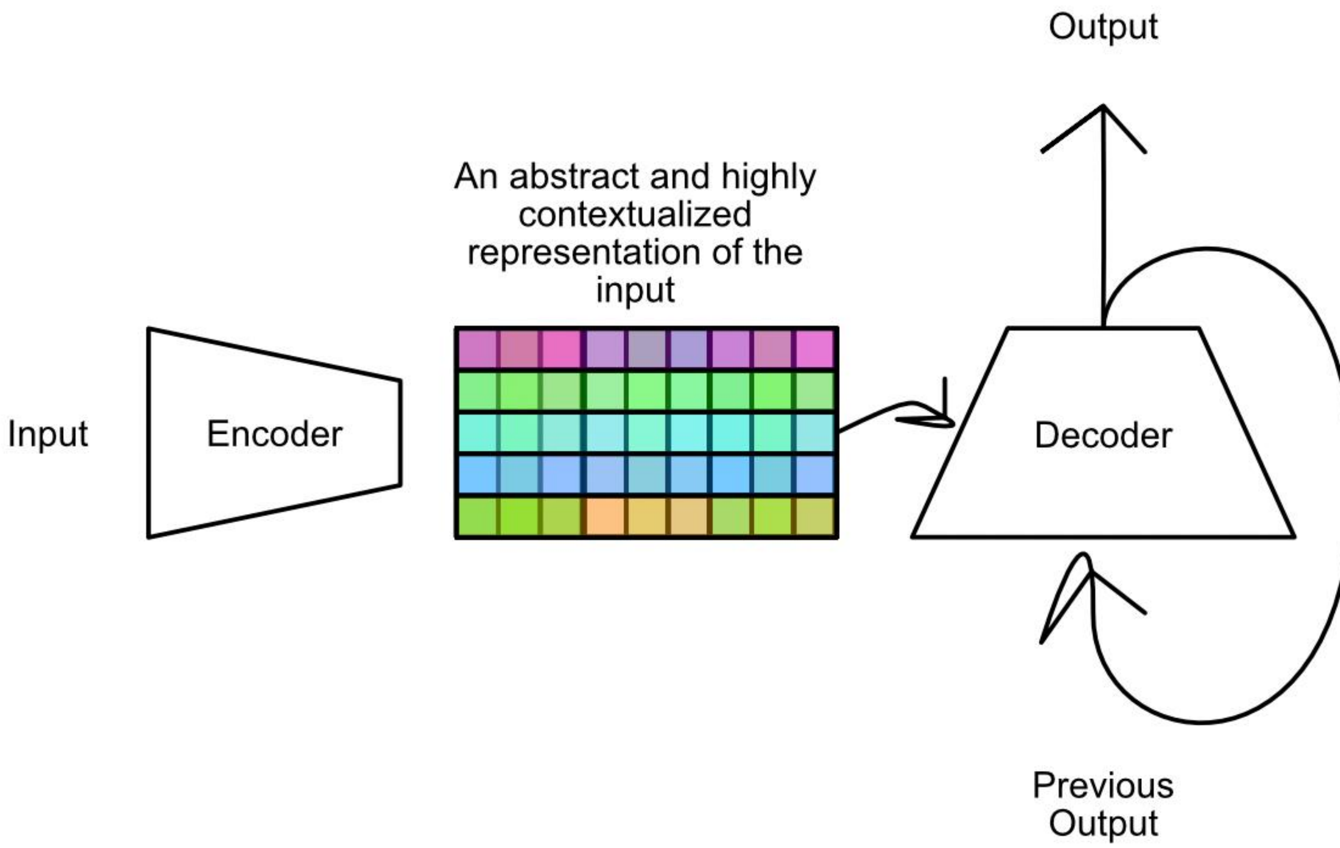
A Brief Introduction to Transformers

Before we get into GPT I want to briefly go over the transformer. In its most basic sense, the transformer is an encoder-decoder style model.



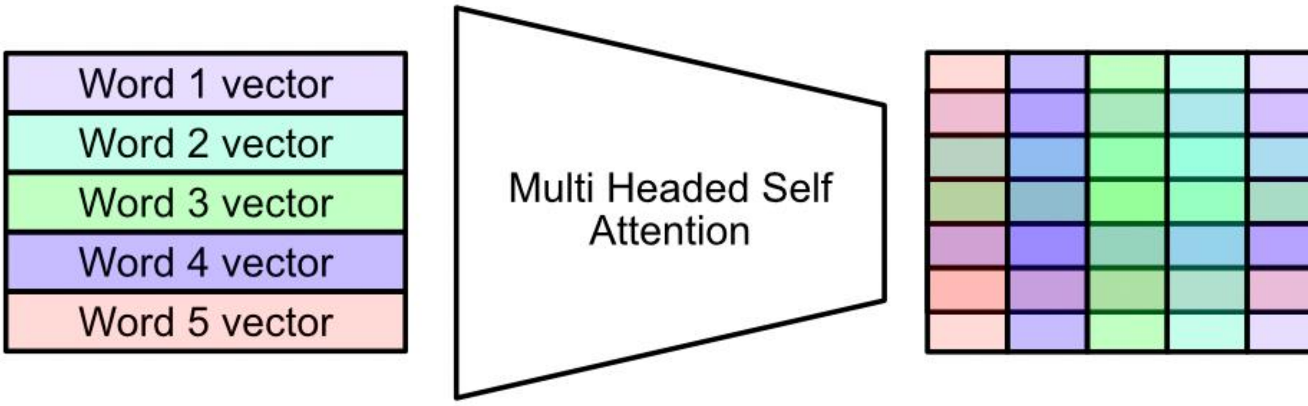
A transformer working in a translation task. The input (I am a manager) is compressed to some abstract representation that encodes the meaning of the entire input. The decoder works recurrently, by feeding into itself, to construct the output. Frommy article on transformers

The encoder converts an input into an abstract representation which the decoder uses to iteratively generate output.



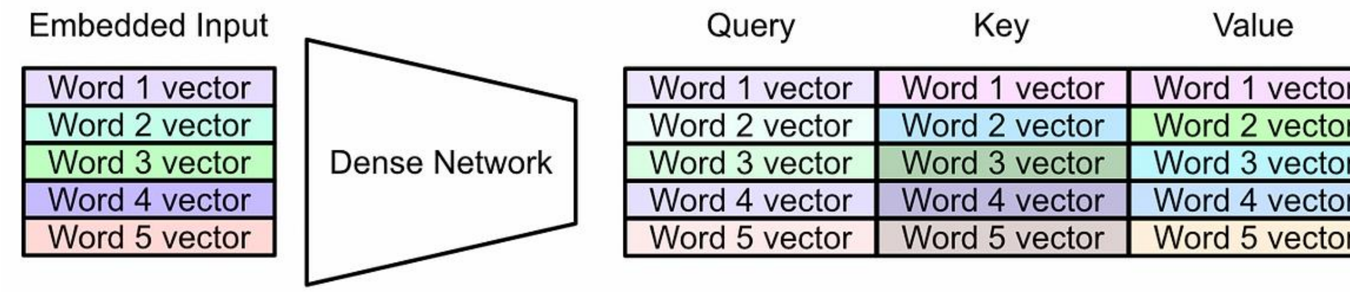
high level representation of how the output of the encoder relates to the decoder. the decoder references the encoded input for every recursive loop of the output. From[my article on transformers]

both the encoder and decoder employ an abstract representations of text which is created using multi headed self attention.



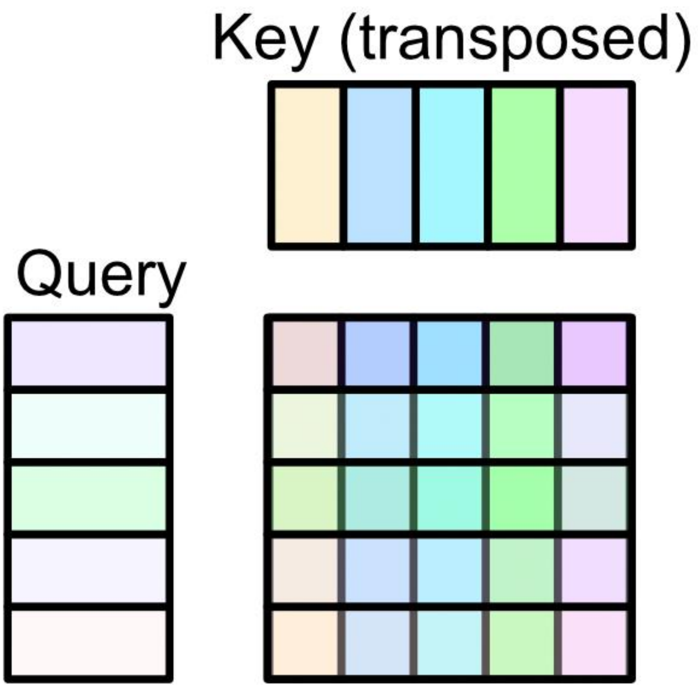
Multi Headed self attention, in a nutshell. The mechanism mathematically combines the vectors for different words, creating a matrix which encodes a deeper meaning of the entire input. From[my article on transformers]

There's a few steps which multiheaded self attention employs to construct this abstract representation. In a nutshell, a dense neural network constructs three representations, usually referred to as the query, key, and value, based on the input.



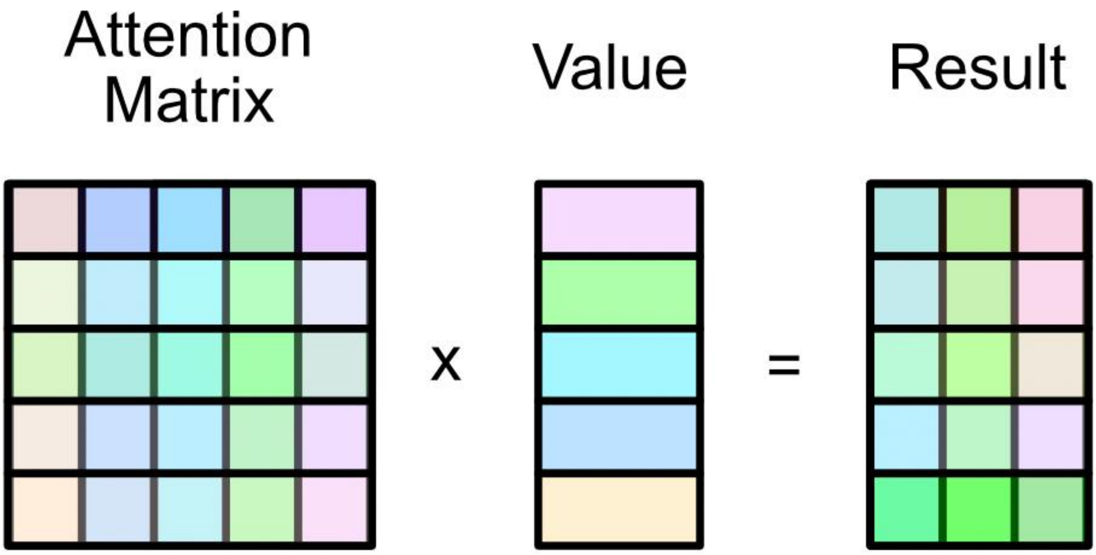
Turning the embedded input into the query, key, and value. The query, key, and value all have the same dimensions as the input, and can be thought of as the input which has been passed through a filter. From[my article on transformers]

The query and key are multiplied together. Thus, some representation of every word is combined with a representation of every other word.



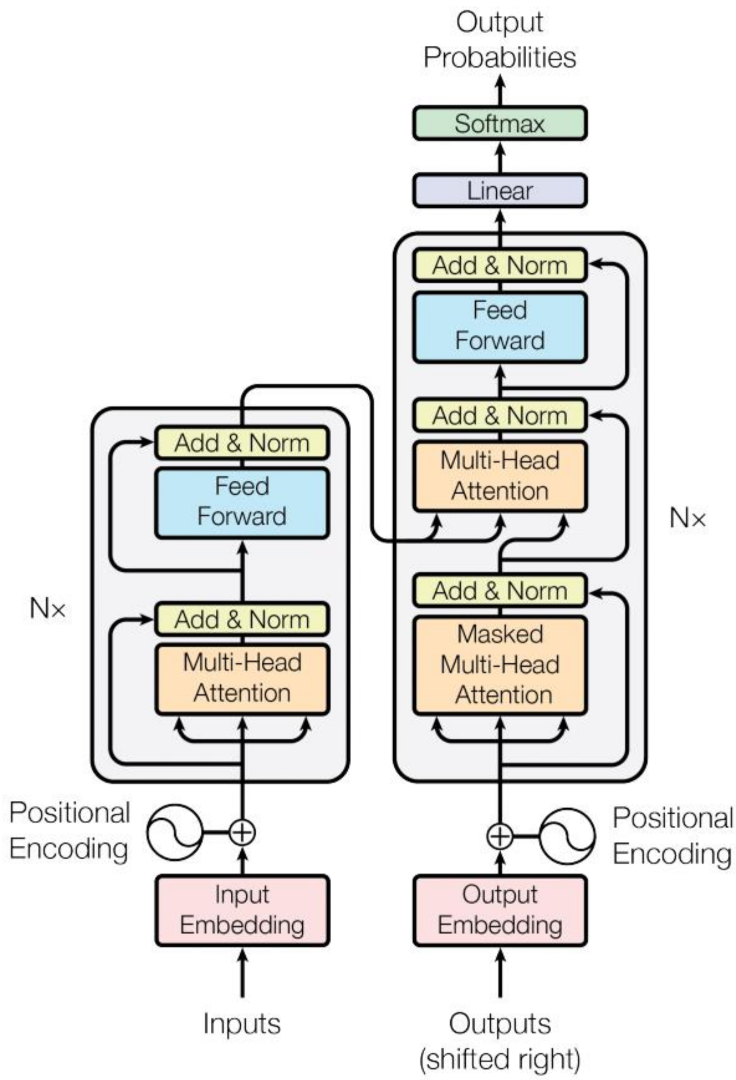
Calculating the attention matrix with the query and key. The attention matrix is then used, in combination with the value, to generate the final output of the attention mechanism. Frommy article on transformers

The value is then multiplied by this abstract combination of the query and key, constructing the final output of multi headed self attention.

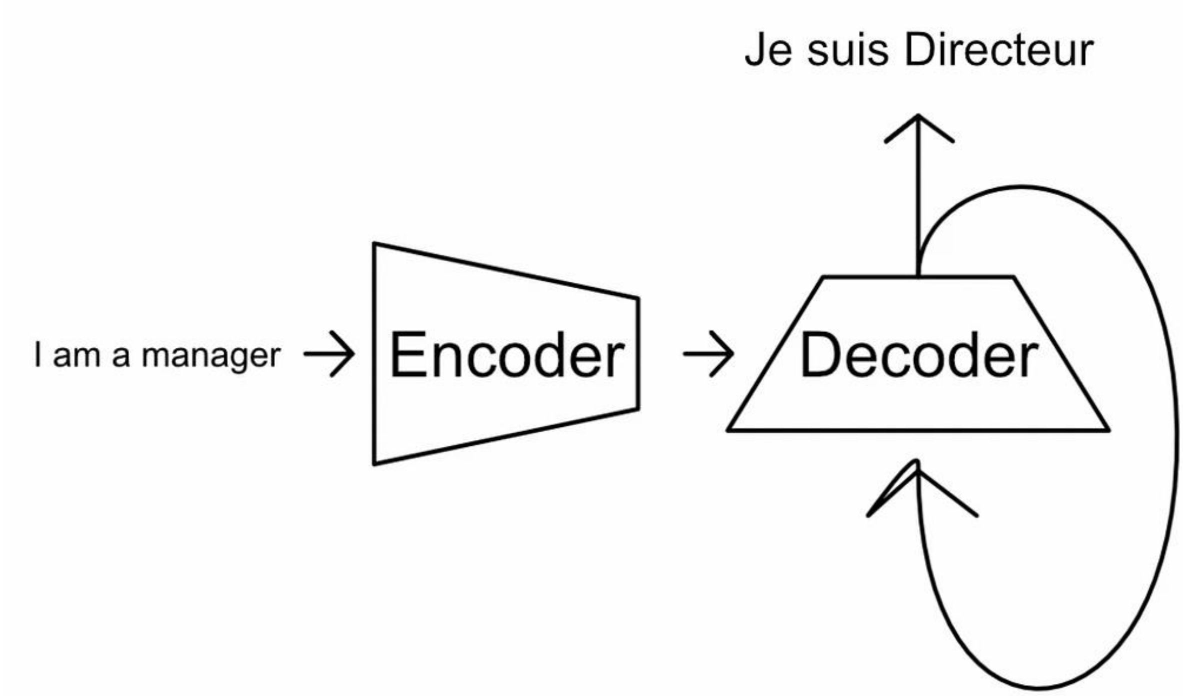


The attention matrix (which is the matrix multiplication of the query and key) multiplied by the value matrix to yield the final result of the attention mechanism. Because of the shape of the attention matrix, the result is the same shape as the value matrix. Frommy article on transformers

The encoder uses multi-headed self attention to create abstract representations of the input, and the decoder uses multi-headed self attention to create abstract representations of the output.



The Transformer Architecture, with the encoder on the left and the decoder on the right. There’ s a lot of details I skimmed over for brevity, but if you look at it from a high level it’ s not too complicated. The encoder uses multi-headed self attention to encode the input, and the decoder uses multi-headed self attention twice; once to encode the previously constructed outputs, and once to combine the input representation with the previous outputs. imagesource



Recall that this is what the transformer is doing, in essence.

That was a super quick rundown on transformers. I tried to cover the high points without getting too in the weeds, feel free to refer to my article on transformers for more information. Now that we vaguely understand the essentials we can start talking about GPT.

GPT-1 (Released June 2018)

The paper Improving Language Understanding by Generative Pre-Training introduced the GPT style model. This is a fantastic paper, with a lot of cool details. We'll summarize this paper into the following key concepts:

1. A decoder-only style architecture
2. Unsupervised pre-training

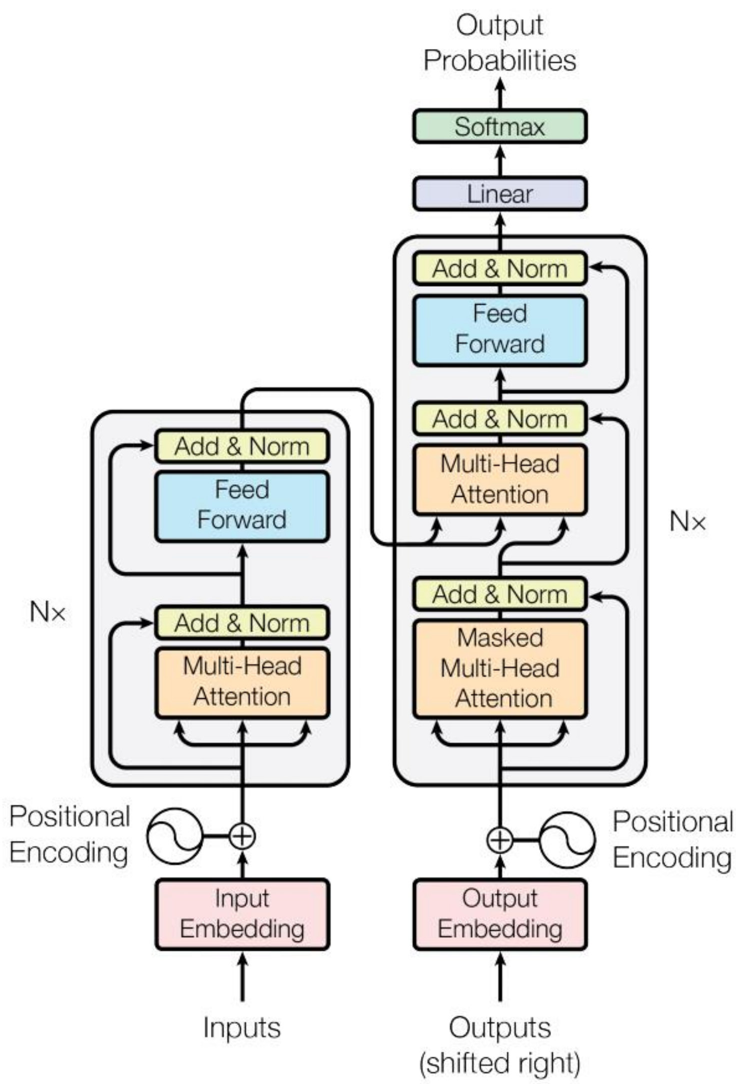
3. Supervised fine tuning

4. Task-specific input transformation

Let's unpack each of these ideas one by one.

Decoder Only Transformers

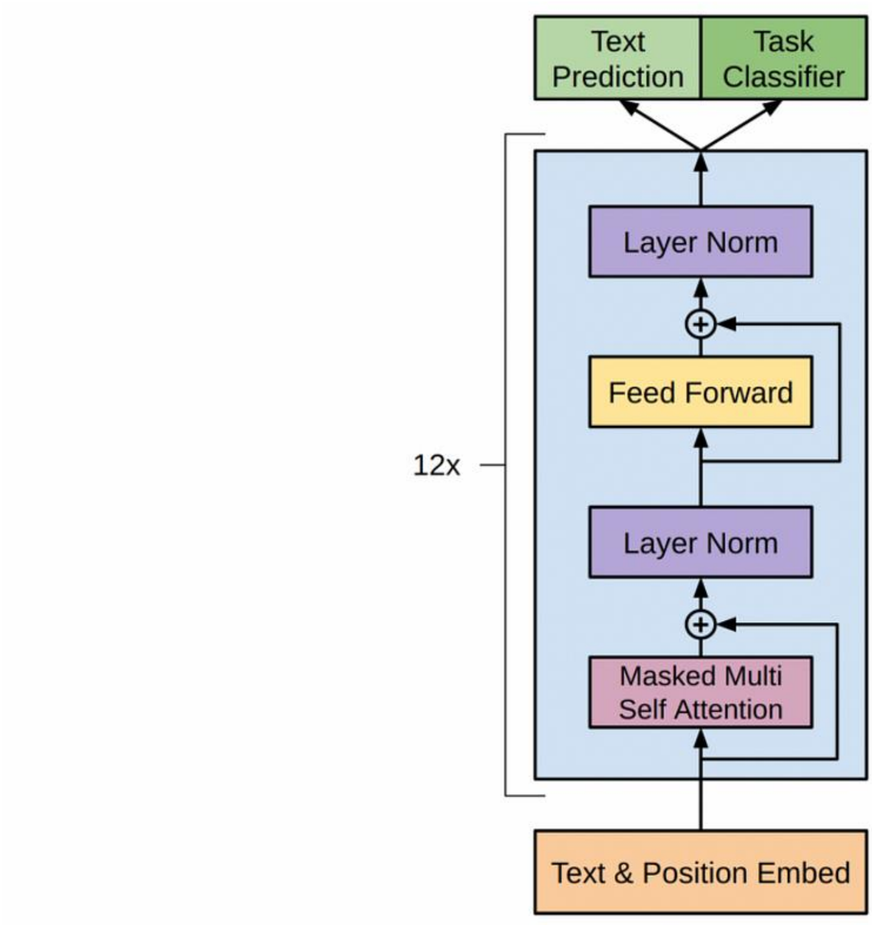
As we previously discussed, the transformer is an encoder-decoder style architecture. The encoder converts some input into an abstract representation, and the decoder iteratively generates the output.



The Transformer Architecture, with the encoder on the left and the decoder on the right. Imagesource

You might notice that both the encoder and the decoder are remarkably similar. Since the transformer was published back in 2017, researchers have played around with each of these subcomponents, and have found that both of them are phenomenally good at language representation. Models which use only an encoder, or only a decoder, have been popular ever since.

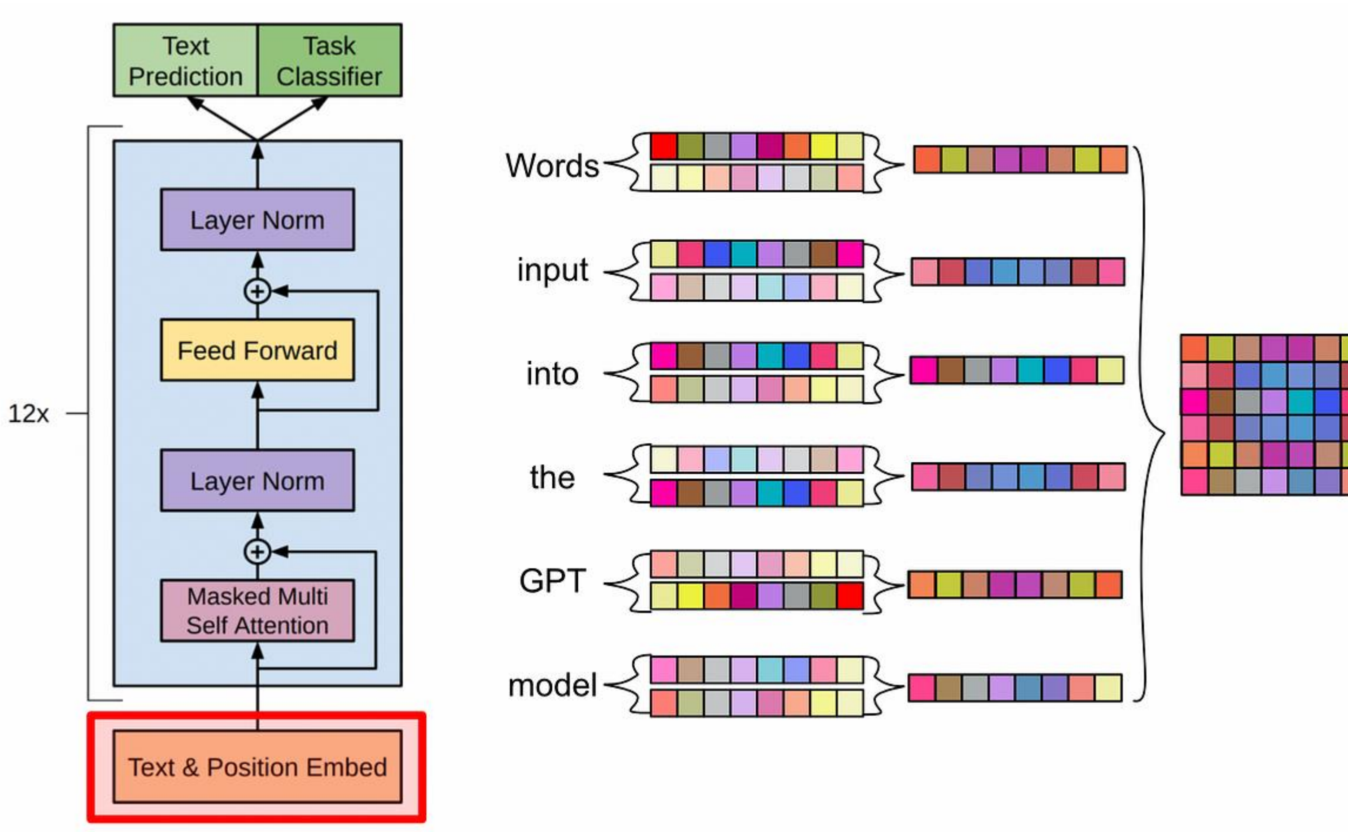
Generally speaking, encoder-only style models are good at extracting information from text for tasks like classification and regression, while decoder-only style models focus on generating text. GPT, being a model focused on text generation, is a decoder only style model.



The model architecture of GPT-1, a decoder-only style model.source

The decoder-only style of model used in GPT has very similar components to the traditional transformer, but also some important and subtle distinctions. Let's run through the key ideas of the architecture.

GPT-1 uses a text and position embedding, which converts a given input word into a vector which encodes both the words general meaning and the position of the word within the sequence.

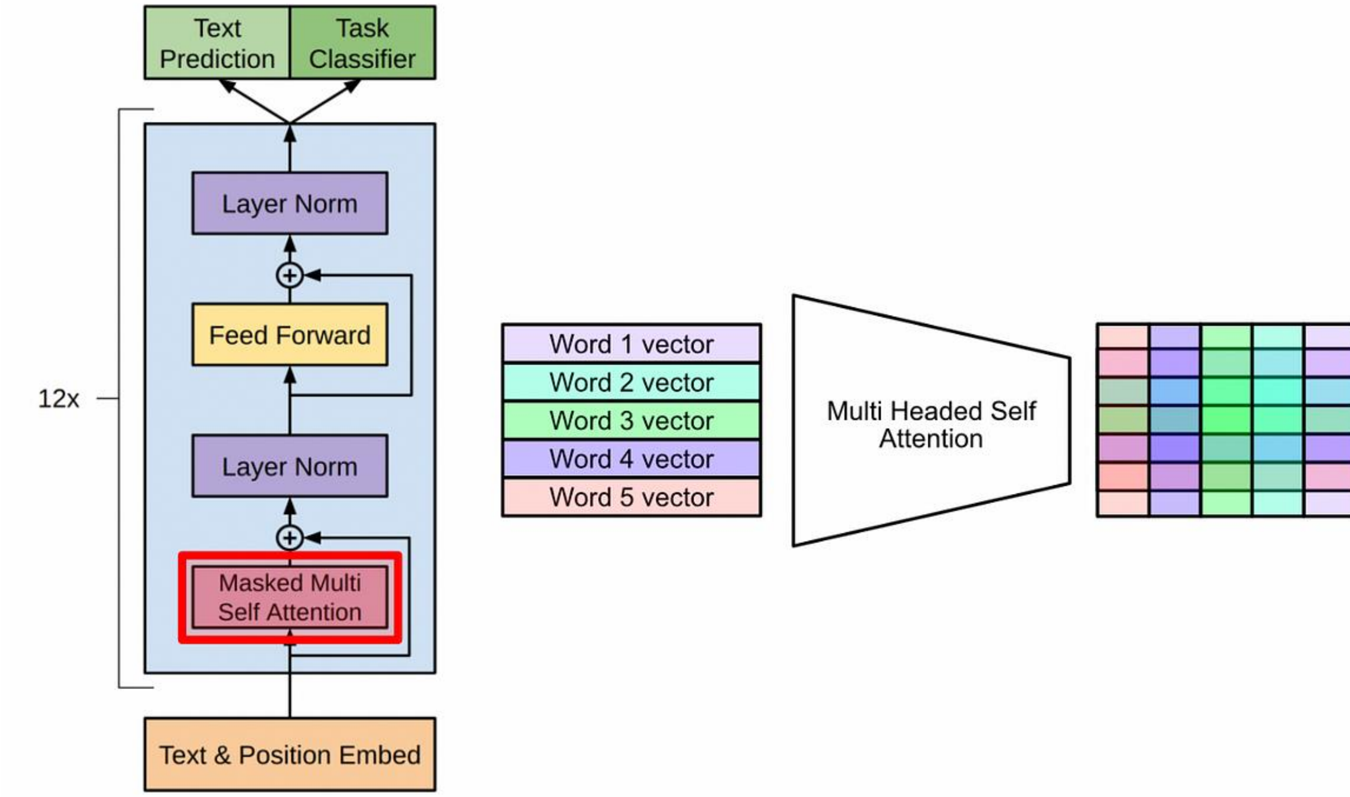


A conceptual diagram of GPT’ s input embedding. Each word has two vectors, one for the word and one for the location, which are added together to represent each word. These are used to construct the input to the model.source

Like the original transformer, GPT uses a “learned word embedding.” Essentially, a vector for each word in the vocabulary of the model is randomly initialized, then updated throughout model training.

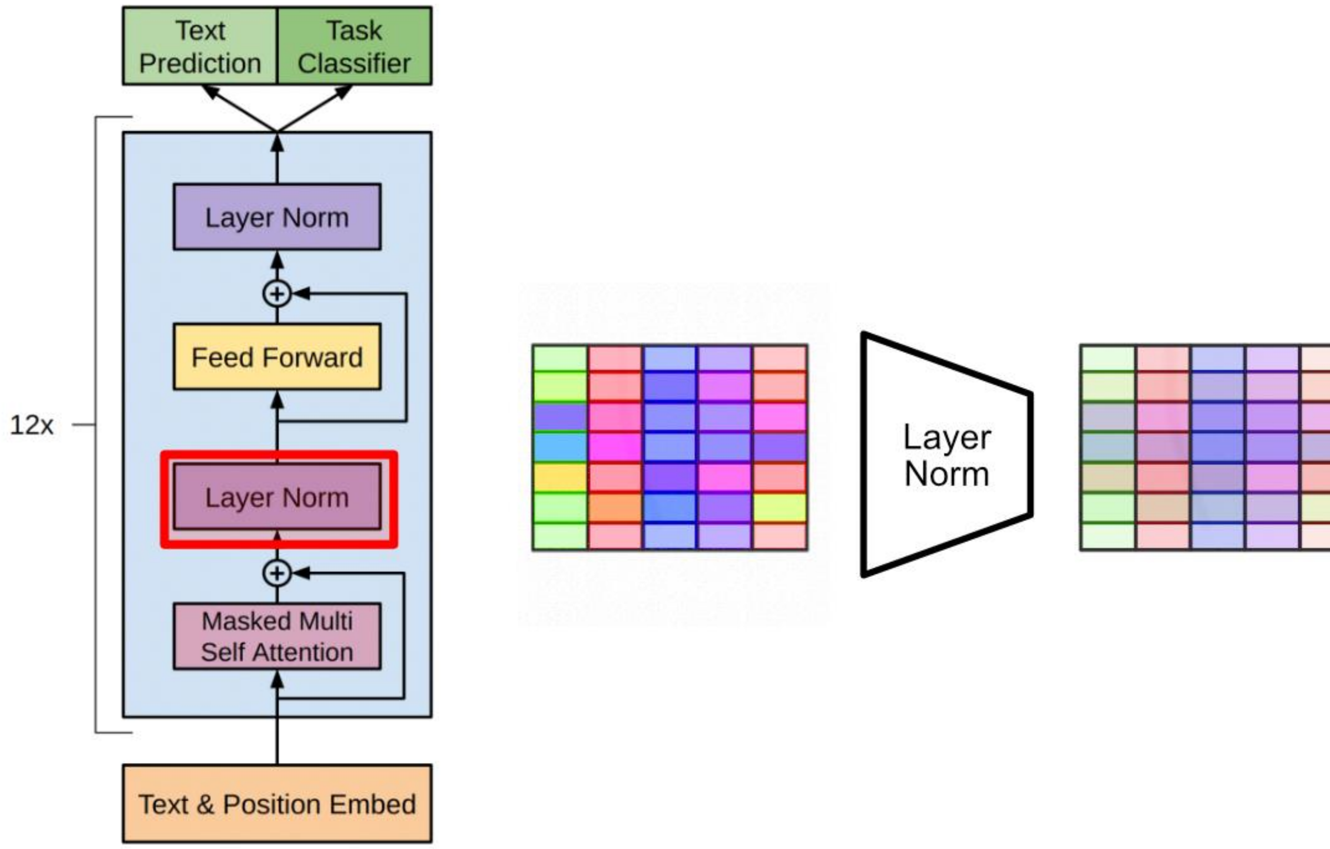
Unlike the original transformer, GPT uses a “learned positional encoding.” GPT learns a vector for each input location, which it adds to the learned vector embedding. This results in an embedding which contains information about the word, and where the word is in the sequence.

These embedded words, with positional information, get passed through masked multi-headed self attention. For the purposes of this article we'll stick with the simplification that this mechanism combines every word vector with every other word vector to create an abstract matrix encoding the whole input in some meaningful way.



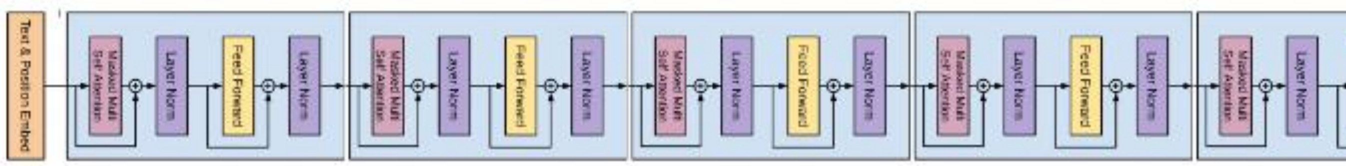
Multi Headed self attention, in a nutshell. The mechanism mathematically combines the vectors for different words, creating a matrix which encodes a deeper meaning of the entire input. From[my article on transformers](#)and[source](#)

A lot of math happens in self attention, which could result in super big or super small values. This has a tendency to make models perform poorly, so all the values are squashed down into a reasonable range with layer normalization.



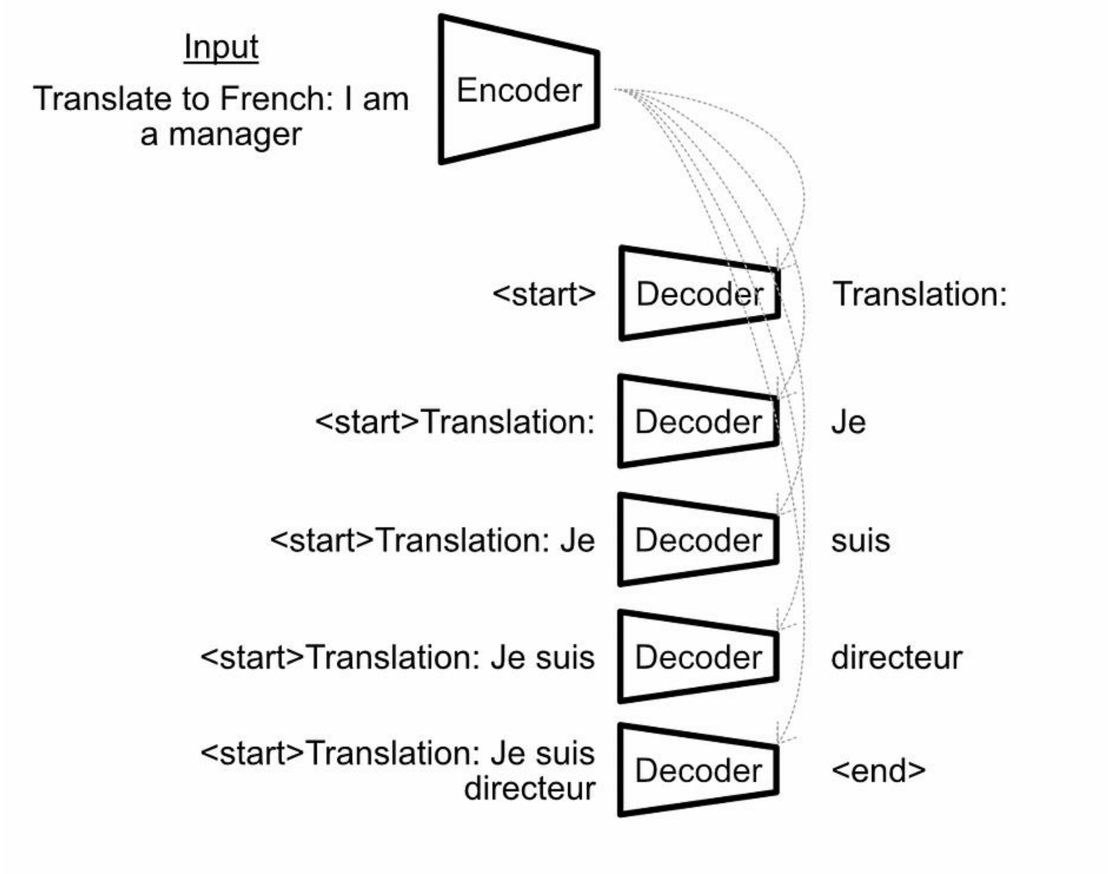
Layer Normalization squashes numbers into a reasonable output, which is important in machine learning for models to learn effectively.source

The data is then passed through a dense network (your classic neural network) then passed through another layer normalization. This all happens in several decoder blocks which are stacked on top of eachother, allowing the GPT model to do some pretty complex manipulations to the input text.



The entire GPT-1 model. 12 decoder layers stacked on top of each other.source

I wanted to take a moment to describe how decoder style models actually go about making inferences; a topic which, for whatever reason, a lot of people don’t seem to take the time to explain. The typical encoder-decoder style transformer encodes the entire input, then recurrently constructs the output.



The traditional encoder-decoder style transformer in action. The encoder compresses the input into an abstract representation, then the decoder uses previously predicted outputs to guess the next word in the output sequence. Thus, the transformer iteratively constructing a response.

Decoder only transformers, like GPT, don’t have an encoder to work with. Instead, they simply concatenate the previous output to the input sequence and pass the entire input and all previous outputs for every inference, a style referred to in the literature as “autoregressive generation”.

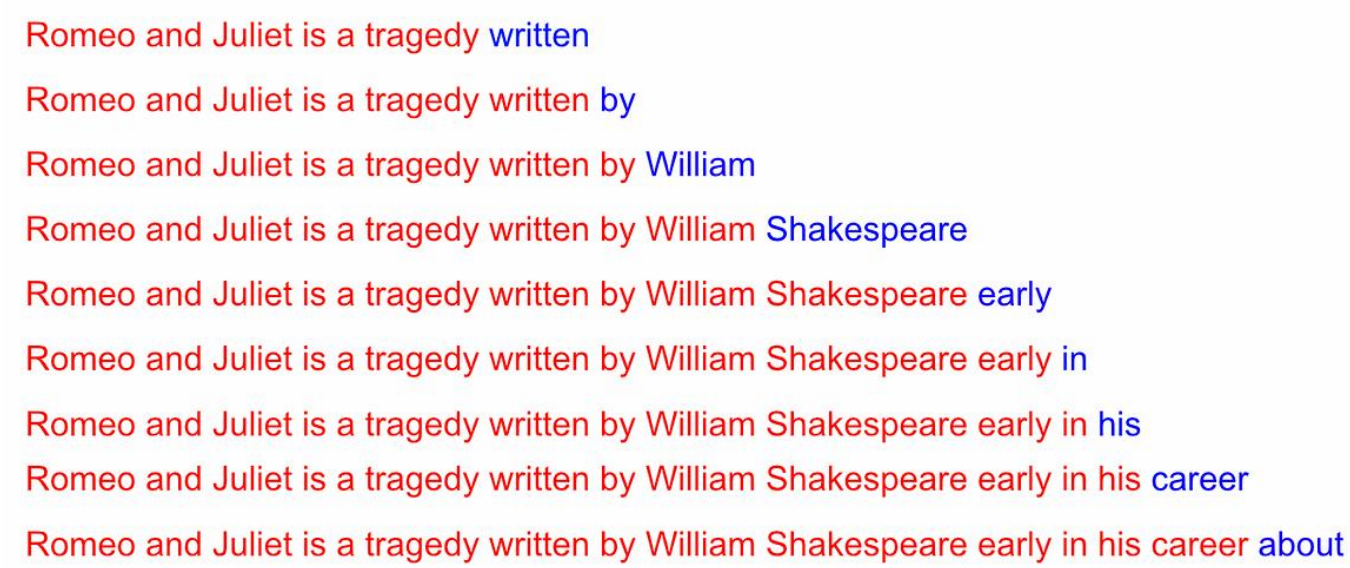
A decoder only style model, like GPT, iteratively constructing an output with autoregressive generation.

This idea of treating outputs similarly to inputs is critical in one of the main deviations GPT took from transformers to reach cutting edge performance.

Semi-Supervised Pre-Training, Then Supervised Fine Tuning

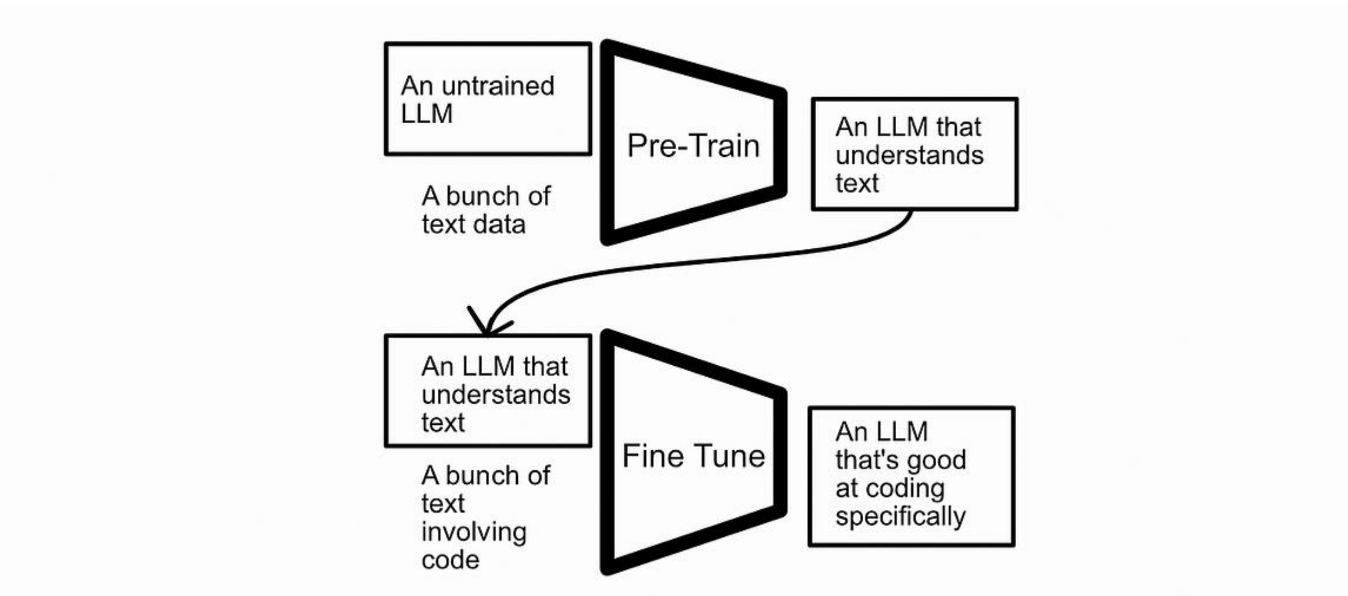
If you research GPT, or language modeling as a whole, you might find the term "language modeling objective." This term refers to the act of predicting the next word given an input sequence, which, essentially models textual language. The idea is, if you can get really really good at predicting the next word in a sequence of text, in theory you can keep predicting the next word over and over until you've output a whole book.

Because the traditional transformer requires the concept of an explicit "input" and "output" (an input for the encoder and an output from the decoder), the idea of next word prediction doesn't really make a tone of sense. Decoder models, like GPT, *only* do next word prediction, so the idea of training them on language modeling makes a tone of sense.



An example of the data a model might see when being trained on language modeling. The model is provided some intput (in red) and asked to predict the next word (in blue). Of course, for a model like GPT, this data would be extracted from large corpus of documents.

This opens up a tone of options in terms of training strategies. I talk a bit about pre-training and fine tuning in my [article on LoRA](). In a nutshell, pre-training is the idea of training on a bulk dataset to get a general understanding of some domain, then that model can be fine tuned on a specific task.



A diagram depicting what pre-training and fine tuning might look like. A language model might be trained on bulk data to understand language, then be fine tuned on a specific task. From my[article on LoRA](#)

GPT is an abbreviation of "Generative Pre-Trained Transformer" for a reason. GPT is pre-trained on a vast amount of text using language modeling (next word prediction). It essentially learns "given an input sequence X, the next word should be Y."

This form of training falls under the broader umbrella of "semi-supervision". Here's a quote from another article which highlights how semi-supervised learning deviates from other training strategies:

- **Supervised Learning** is the process of training a model based on **labeled** information. When training a model to predict if images contain cats or dogs, for instance, one curates a set of images which are labeled as having a cat or a dog, then trains the model (using gradient descent) to understand the difference between images with cats and dogs.

- **Unsupervised Learning** is the process of giving some sort of model **unlabeled** information, and extracting useful inferences through some sort of transformation of the data. A classic example of unsupervised learning is clustering; where groups of information are extracted from un-grouped data based on local position.

Self-supervised learning is somewhere in between. **Self-supervision uses labels that are generated programmatically, not by humans.** In some ways it's supervised because the model learns from labeled data, but in other ways it's unsupervised because no labels are provided to the training algorithm. Hence self-supervised.

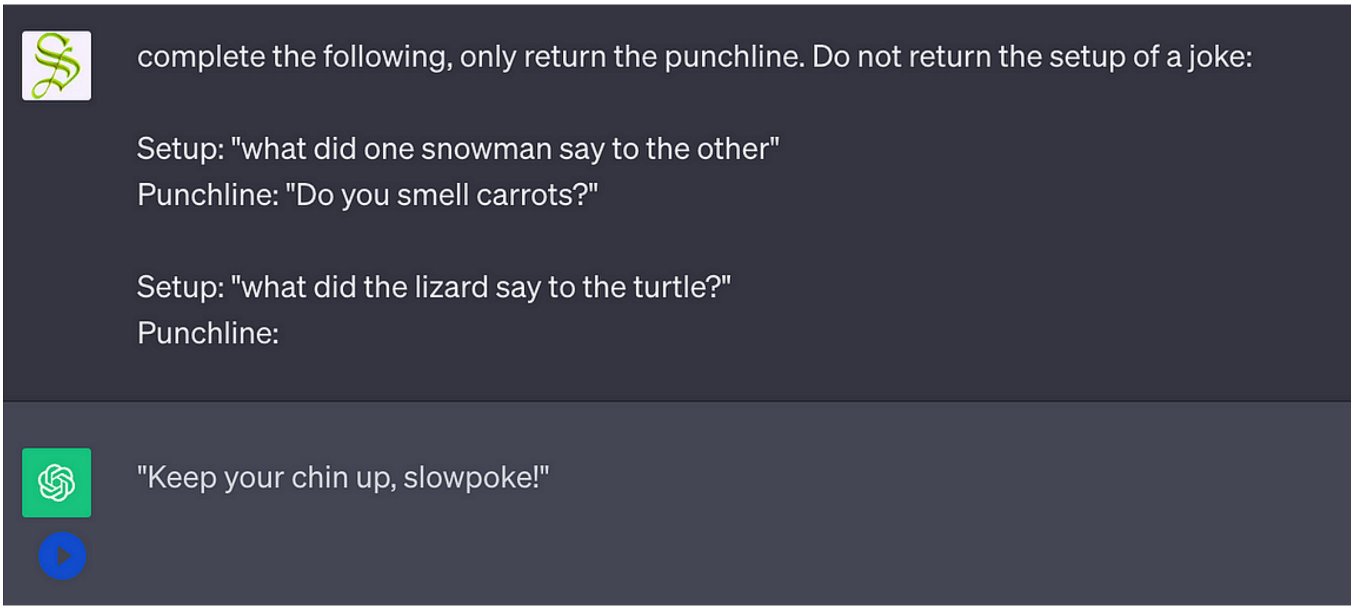
Using a semi-supervised approach, in the form of language modeling, allows GPT to be trained on a previously unprecedented volume of training data, allowing the model to create robust linguistic representations. This model, with a firm understanding of language, can then be fine-tuned on more specific datasets for more specific tasks.

We use the BooksCorpus dataset [71] for training the language model. It contains over 7,000 unique unpublished books from a variety of genres including Adventure, Fantasy, and Romance. — The GPT paper on pre-training

We perform experiments on a variety of supervised tasks including natural language inference, question answering, semantic similarity, and text classification. — The GPT paper on fine tuning

Task-Specific Input Transformation

Language models, like GPT, are currently known to be incredibly powerful “in context learners”; you can give them information in their input as context, then they can use that context to construct better responses. I used this concept in my article on retreival augmented generation, my article on visual question answering, and my article on parsing the output from language models. It’s a pretty powerful concept.



An example of ChatGPT succeeding at a task with the help of in context learning. From myRAG article

At the time of the first GPT paper in context learning was not nearly so well understood. The paper dedicates an entire section to “Task-specific input transformations”. Basically, instead of adding special components to the model to make it work for a specific task, the authors of GPT opted to format those tasks textually.

As an example, for textual entailment (the process of predicting if a piece of text directly relates with another piece of text) the GPT paper simply concatenated both pieces of text together, with a dollar sign in between. This allowed them to fine tune GPT on textual entailment without adding any new parameters to the model. Other objectives, like text similarity, question answering, and commonsense reasoning, were all fine tuned in a similar way.

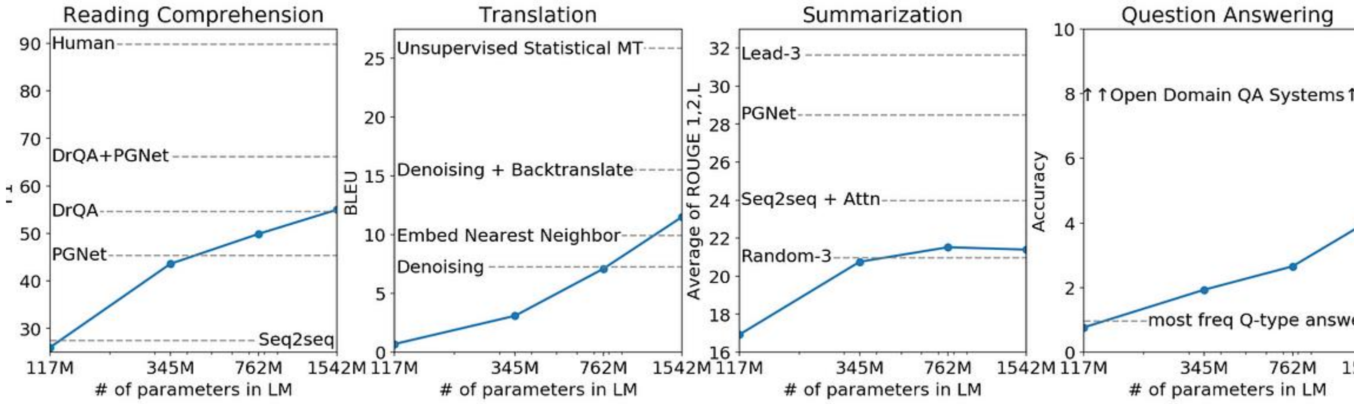
GPT-2 (Released February 2019)

The paper [Language Models are Unsupervised Multitask Learners]() introduced the world to GPT-2, which is essentially identical to GPT-1 save two key differences:

1. GPT-2 is way bigger than GPT-1
2. GPT-2 doesn't use any fine tuning, only pre-training

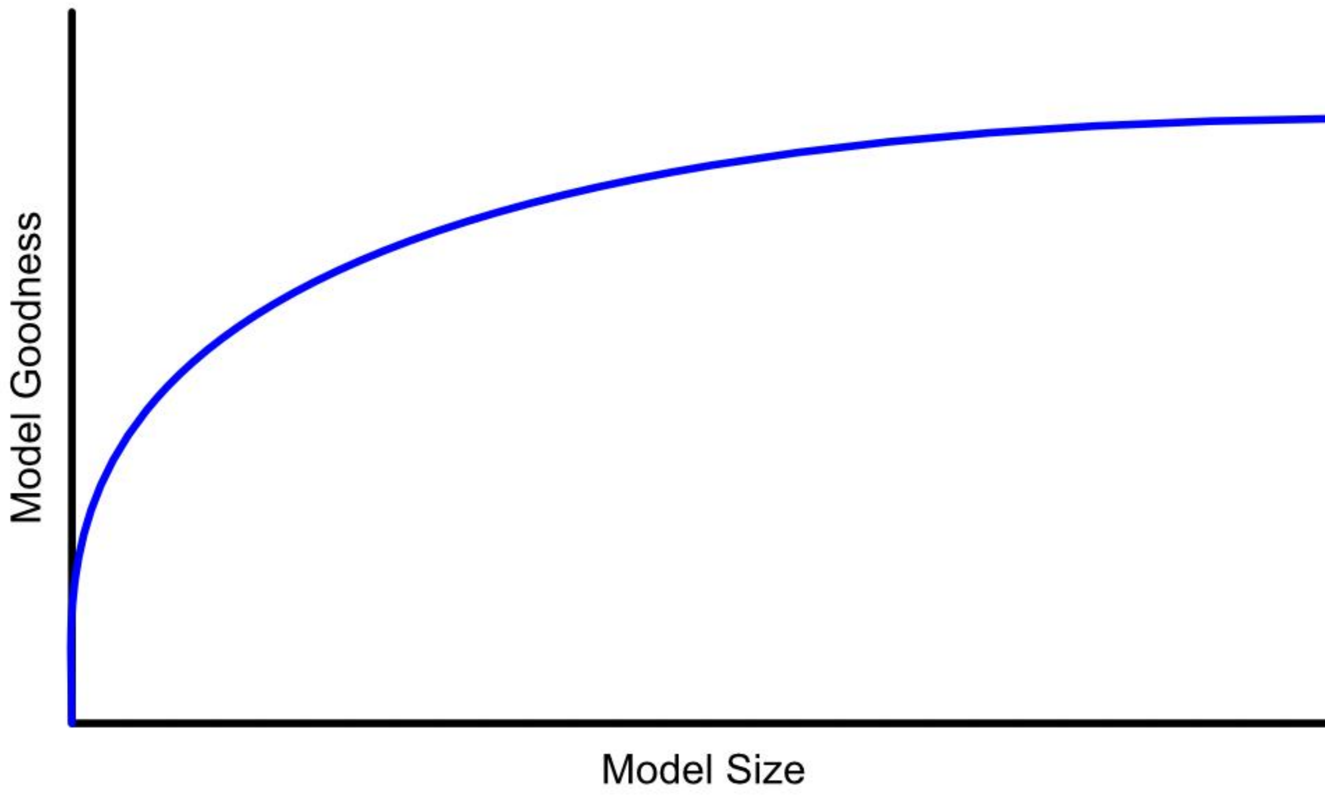
Also, as a brief note, the GPT-2 architecture is ever so slightly different from the GPT-1 architecture. In GPT-1 each block consists of `[Attention, Norm, Feed Forward, Norm]`, where GPT-2 consists of `[Norm, Attention, Norm, Feed Forward]`. However, this difference is so minor it's hardly worth mentioning.

Bigger is Better



The performance of GPT-2 on several objectives, as a function of parameter count. As can be seen, the more parameters a language model is, the better it generally performs. From theGPT-2 paper

One of the findings of the GPT-2 paper is that larger models are better. Specifically, they theorize that language model performance scales "log-linearly", so something like this:



The relationship between how good a model is, and how big it is, seems to be log-linear, based on the findings of GPT-2

As a result GPT-2 contains 1.5 billion parameters, which is roughly ten times larger than GPT-1. GPT-2 was also trained on roughly ten times the data.

Language Understanding is General

In GPT-1 they focused on using the language modeling objective to create a solid baseline of textual understanding, and then used that baseline to fine tune models for specific tasks. In GPT-2 they got rid of fine tuning completely, operating under the assumption that a sufficiently pre-trained model can perform well on specific problems without being explicitly trained on them.

Our suspicion is that the prevalence of single task training on single domain datasets is a major contributor to the lack of generalization observed in current systems. Progress towards robust systems with current architectures is likely to require training and measuring performance on a wide range of domains and tasks…

When conditioned on a document plus questions, the answers generated by the language model reach 55 F1 on the CoQA dataset — matching or exceeding the performance of 3 out of 4 baseline systems without using the 127,000+ training examples…

Our largest model, GPT-2, is a 1.5B parameter Transformer that achieves state of the art results on 7 out of 8 tested *language modeling datasets in a zero-shot setting — The GPT-2 paper*

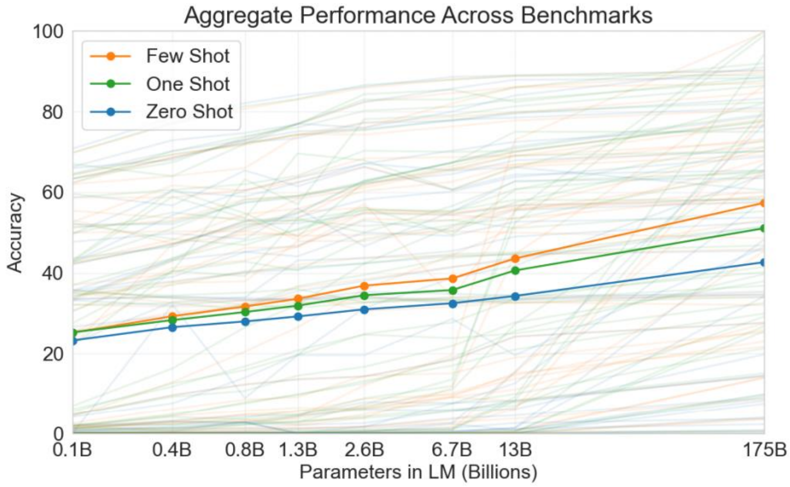
They achieved this generalized performance by using a super big model and feeding it a whole bunch of high quality data.

Our approach motivates building as large and diverse a dataset as possible in order to collect natural language demonstrations of tasks in as varied of domains and contexts as possible…

we created a new web scrape which emphasizes document quality. To do this we only scraped web pages which have been curated/filtered by humans. Manually filtering a full web scrape would be exceptionally expensive so as a starting point, we scraped all outbound links from Reddit, a social media platform, which received at least 3 karma. This can be thought of as a heuristic indicator for whether other users found the link interesting, educational, or just funny. — The GPT-2 paper

GPT 3 (Released June 2020)

The paper Language Models are Few-Shot Learners introduced the world to GPT-3, which essentially said "bigger was good, so why not even bigger?" Instead of a measly 1.5 billion parameters in GPT-2, GPT-3 employs 175 billion parameters, and was trained on 45 Terabytes of text data.



A graph of model size vs performance. From the[GPT-3 paper]

From a data science perspective not much really changed. Similar model, similar core architecture, similar language modeling objective. From an engineering perspective, however, things were certainly different. GPT-2 is roughly 5 Gigabytes in size, whereas GPT-3 is around 800 Gigabytes (I've seen a lot of variation in this number online, but it's definitely big). The Nvidia H100 GPU, which retails at a cool $30,000, holds 80GB of VRAM. Keep in mind, to train a machine learning model you essentially need double the model size in VRAM to hold the parameters as well as the gradients (as I discussed in my [LoRA article]).

	Mean accuracy	95% Confidence Interval (low, hi)	t compared to control (p-value)	"I don't know" assignments
Control (deliberately bad model)	86%	83%–90%	-	3.6 %
GPT-3 Small	76%	72%–80%	3.9 (2e-4)	4.9%
GPT-3 Medium	61%	58%–65%	10.3 (7e-21)	6.0%
GPT-3 Large	68%	64%–72%	7.3 (3e-11)	8.7%
GPT-3 XL	62%	59%–65%	10.7 (1e-19)	7.5%
GPT-3 2.7B	62%	58%–65%	10.4 (5e-19)	7.1%
GPT-3 6.7B	60%	56%–63%	11.2 (3e-21)	6.2%
GPT-3 13B	55%	52%–58%	15.3 (1e-32)	7.1%
GPT-3 175B	52%	49%–54%	16.9 (1e-34)	7.8%

A table from theGPT-3 paper, stating the accuracy of humans in predicting if a news article was written by GPT-3 or not. As you can see, for the full sized 175B parameter model, only 52% of people guessed correctly. 50% would be 50/50 shot in the dark.

I've seen actual cost estimations for GPT-3 which deviate wildly, mostly as a result of how much VRAM the model actually required to train. One thing is certain; you need a cluster of super expensive computers, working in parallel, to train a model like GPT-3, and you need a similar cluster to serve that model to users.

The GPT-3 paper is titled "Language Models are Few-Shot Learners." By "Few-Shot Learners" the authors mean that GPT-3 can get pretty good at most tasks without training the model. Instead, you can just experiment with the input to the model to find the right input format for a particular task. And thus, prompt engineering was born.

GPT-4 (Released March 2023)

At this point clarity ends. OpenAI has shifted from a research non-profit to a seriously heavy hitting corporate machine, facing off against big names like Microsoft and Google. There is no peer reviewed and transparent paper on GPT-4. As you might know, OpenAI has been experiencing a bit of friction internally, this is part of the reason why.

That said, is there information? Absolutely.

GPT-4 is "Safe" To Use in Products

First of all, we can glean a lot of information from the [100 page technical report] by OpenAI, which hides a lot of the specifics in terms of modeling, but exhaustively details performance and capability of the model. This paper reads a lot more like an advertisement than anything seriously academic; the term “safety”, for instance, has a much stronger bias towards safety to use within a product rather than the traditional usage of the word in research which focuses on safety for humans.

GPT-4 Can Understand Images

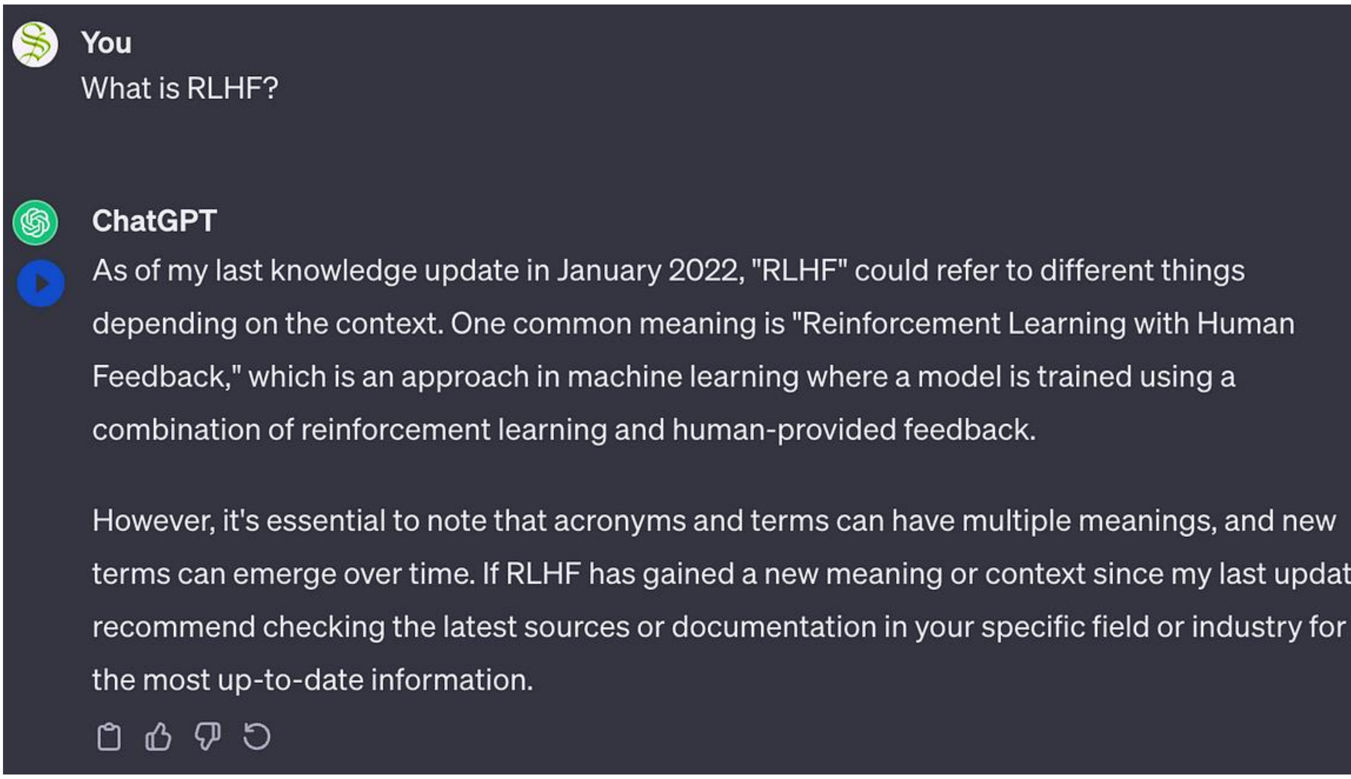
One of the big changes in GPT-4 is “multi-modality”; allowing GPT-4 to converse about both images and text.

GPT-4 conversing about an image.Source

This has all sorts of cool uses. In a pretty famous example, OpenAI demoed how this functionality could be used to turn a crude sketch into a (still fairly crude) website written in HTML, CSS, and JavaScript.

GPT-4 Is Fine Tuned on Human Feedback

Also, GPT-4 uses "Reinforcement Learning with Human Feedback", commonly abbreviated as RLHF. You know those little thumbs up and thumbs down on the ChatGPT website? Those allow OpenAI to further improve the model based on in-context human feedback.



This strategy of learning can have dubious results, and can actually degrade the models performance. However, it does help with getting GPT-4 into a better style as a sort of "AI Helper"

Our evaluations suggest RLHF does not significantly affect the base GPT-4 model's capability — GPT-4 Technical Report

GPT-4 is Really Big

There was an infamous leak about four months ago which seemed to have drop some credible evidence about some of the behind the scenes details on GPT-4. This information was released behind a paywall, which was then leaked again on Twitter, which is now… X? I guess?

Anyway, GPT-4 seems like it hits the 1.8 Trillion parameter mark, which is around ten times the size of GPT-3.

GPT-4 uses Mixture of Experts

I plan on covering Mixture of Experts (MOE) in a future article. In a nutshell MOE is the process of dividing a large model into smaller sub models, and creating a tiny model which serves to route data to individual components.

The whole idea is, the bigger your model gets, the more costly it becomes to run. If instead you can have a portion of your model dedicated to art, another dedicated to science, and another to telling jokes, you can route queries to individual sub-components. In reality these groupings are much more abstract, but the logic is the same; the model can use different sub-components on every query, allowing it to run much leaner than if it were to use all parameters for every query.

Based on the GPT-4 leak, by using MOE, the model only needs to use 280 billion parameters of the models 1.8 Trillion (a mere 15%) on any given inference.

GPT-4 uses 16 experts (the model is divided into 16 parts which are then routed to). This is much less than the current state of research, which suggests larger sets of 64 or 128 experts.

GPT-4 Training Cost

Based on the GPT-4 leak, the model cost around $63 million to train on A100 GPUs. Using more modern H100 GPUs would probably cost around $21.5 million. Such is the cost of being an early mover, I guess.

GPT-4 Uses Some Bells and Whistles

- GPT-4 uses “multi-query attention”, which is a style of attention which improves model throughput. I’ll be covering it in an upcoming article.
- GPT-4 uses “continuous batching” which I will probably cover in yet another article.
- GPT-4 uses “speculative decoding”. My goodness, the pace of AI research is keeping me busy.

GPT-4 Probably Uses Something Like Flamingo

GPT-4 uses a style of multimodal modeling similar to “Flamingo”, a multimodal model by Deepmind. I’ll cover

Flamingo specifically in the future, but I do have an article on a similar approach by the SalesForce research team, called Q-Fromer:

Visual Question Answering with Frozen Large Language Models
Talking with LLMs about images, without training LLMs on images.
towardsdatascience.com

In a nutshell, both Q-Former and Flamingo use frozen models for both language and images, and trains some glue that joins them together to create a model which can talk about images.

GPT-4 Uses High Quality Data

It seems like, based on the GPT-4 leak, OpenAI used college textbooks to improve the model's knowledge in numerous intellectual domains. The legality and ethicality of this is definitely in a gray area, which might also explain some of the current in-fighting and secrecy at OpenAI.

Conclusion

Well, that got kind of dramatic in the end, huh? First we briefly went over transformers, then described how GPT-1 adopted the decoder component of the transformer to improve text generation through self-supervised pre-training using the language modeling objective. We then discussed how GPT-2 and GPT-3 both doubled down on this approach, producing language models which were, to a large degree, Identical to GPT-1 just way bigger. We then discussed what we know of

GPT-4; the model size, some architectural changes, and even some saucy controversy.

Follow For More!

I describe papers and concepts in the ML space, with an emphasis on practical and intuitive explanations.

Get an email whenever Daniel Warfield publishes

Get an email whenever Daniel Warfield publishes By signing up, you will create a Medium account if you don't already…

medium.com

Never expected, always appreciated. By donating you allow me to allocate more time and resources towards more frequent and higher quality articles.Link

Attribution: All of the resources in this document were created by Daniel Warfield, unless a source is otherwise provided. You can use any resource in this post for your own non-commercial purposes, so long as you reference this article, https://danielwarfield.dev, or both. An explicit commercial license may be granted upon request.

Data Science

Transformers

ChatGPT

Machine Learning

Editors Pick

Written by Daniel Warfield

7.6K Followers
·Writer for

Towards Data Science

Data Scientist and Educator, teaching machine learning Intuitively and Exhaustively.

More from Daniel Warfield and Towards Data Science

Daniel Warfield

in

Level Up Coding

AGI is Not Possible

An opinion piece by a data scientist and humanist.

·8 min read·Apr 15, 2024

1.5K

42

Youness Mansar

in

Towards Data Science

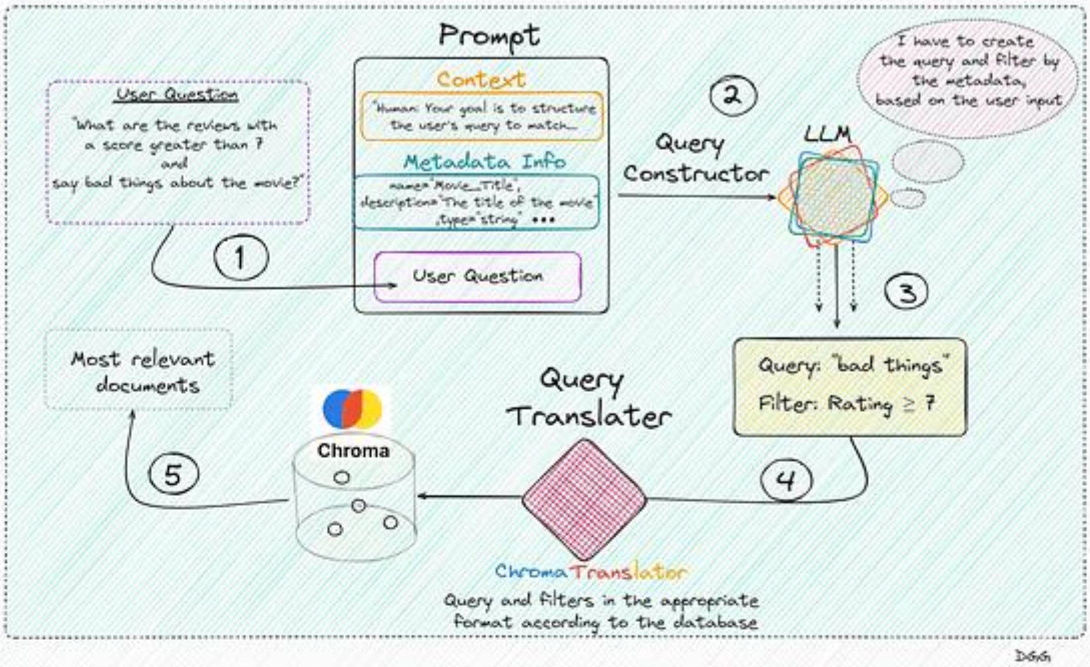
Meet the NiceGUI: Your Soon-to-be Favorite Python UI Library

Build custom web apps easily and quickly

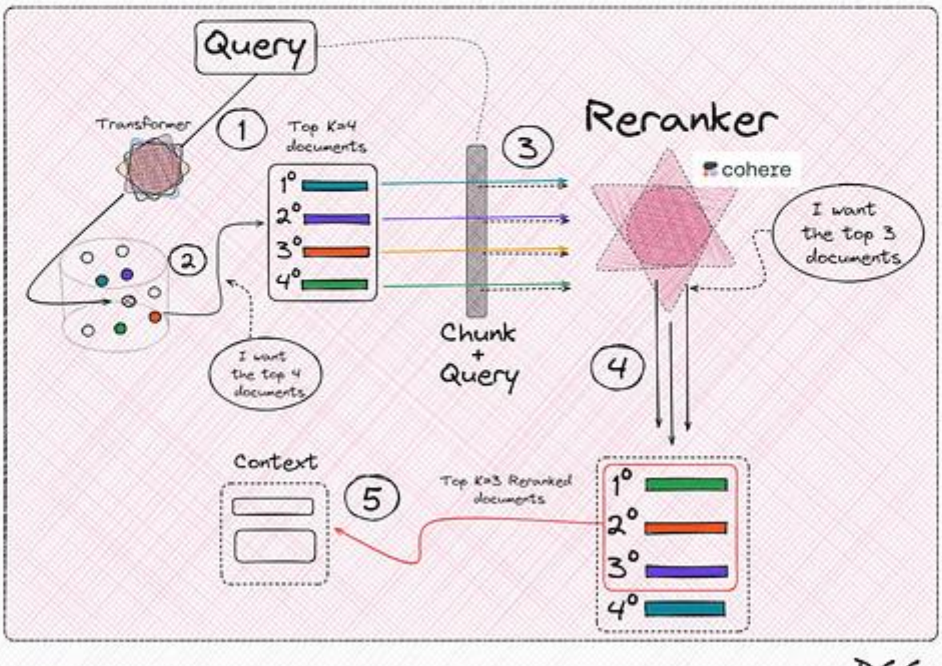
8 min read·Apr 17, 2024

Advaned Ret

Self Query Retriever



Contextaul Compression Retriever (Reranker)



Damian Gil

in

Towards Data Science

Advanced Retriever Techniques to Improve Your RAGs

Master Advanced Information Retrieval: Cutting-edge Techniques to Optimize the Selection of Relevant Documents with Langchain to Create…

18 min read·Apr 17, 2024

Daniel Warfield

in

Towards Data Science