Fine Tuning GPT-2 to imitate texting style

Akshaj Tyagi
at1071

Malav Doshi
md1378

Zain Mogul
Zmm21

Abstract

The objective of this project is to analyze OpenAI's paper, *Language models are unsupervised multitask learners*. In our chosen paper, the researchers trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many languages modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training. Given this is a difficult paper (and its our first-time diving into the topic of Natural language processing, especially with something as complex as masked self-attention mechanism-based transformer model), the hope of this paper is to gain a better insight about the inner working of a Generative Pre-Trained Transformer Language model. After understanding in depth exactly how this mechanism works specifically for GPT-2, we plan on fine tuning the model to take text input of a given persons text conversation and output a text snippet that mimics the way they converse.

1 Introduction

Language models are incredibly complex in general. The nature of languages is such that it is generally based upon many contexts specific information that is difficult to predict as a simple logical system. There are many high-level abstractions and ambiguities. This isn't to say that there aren't low level rules in language but to highlight that encompassing all of language is a complex problem that cannot be modeled by a simple classifier or a large database.

In this paper we focus on analyzing OpenAI's paper, *Language models are unsupervised multitask learners.* In the paper, the researchers create a language model (which in the most reductive terms is a probability distribution of a sequence of letters/words that together create a coherent/colloquially acceptable sentence). Currently, it's difficult for an algorithm to effectively model human language due to its immense complexity. GPT-2 displays a broad set of capabilities, including the ability to generate conditional synthetic text samples of unprecedented quality, where we prime the model with an input and have it generate a lengthy continuation

The nature of this algorithm is iterative. While language models have been made before, most have not been as big or as as effective. GPT stands for Generative Pre-Trained Transformer architecture that uses a deep neural network based on a transformer model which uses attention instead of recurrence-based architecture or even convolution neural networks. This is key because attention mechanisms allow the model to selectively focus on segments of input to create its best possible output. This allows for much higher parallelization and therefore outperforms the benchmarks for the older models. GPT-2 is a large transformer-based language model with 1.5 billion parameters, trained on a dataset of 8 million web pages. GPT-2 is trained with a simple objective: predict the next word, given all of the previous words within some text. The diversity of the dataset causes this simple goal to contain naturally occurring demonstrations of many tasks across diverse domains. GPT-2 is a direct scale-up of GPT, with more than 10X the parameters and trained on more than 10X the amount of data. Some limitations of this model are that it is not equipped to handle complex and long language formations and it lacks understanding of very specialized fields without sufficient training beforehand. Another notable limitation is that it needs lots of computational resource to just run the program (eg. a GPU with 12-24 gb of VRAM).

An interesting capability of GPT 2 is zero shot task transfer. Zero shot learning is a special case of zero shot task transfer where no examples are provided at all and the model understands the task based on the given instruction. Instead of rearranging the sequences, as was done for GPT-1 for fine-tuning, input to GPT-2 was given in a format which expected the model to understand the nature of task and provide answers. This was done to emulate zero-shot task transfer behaviour. E.g. for English to French translation task, the model was given an English sentence followed by the word French and a prompt (:). The model was supposed to understand that it is a translation task and give French counterpart of English sentence.

2 Related Work

The Following section focuses on understanding parts of this paper that heavily rely on previous work. Given that a large part of this paper is based on building effective language models using Transformers we will focus on Attention and Transformers before we further dive in. Since the most impressive part of Transformers is their ability to outperform traditional neural networks such as RNN, we will also discuss them first to gain an understanding of it.

2.1 Reccurent Neural Networks'

The idea behind an RNN or a recurrent neural network is based upon a neural network that seeks to produce intelligent behavior through artificial neural networks designed to simulate the behavior of neurons in biological brains. Neural networks become capable of classifying different inputs (i.e. sorting them into distinct categories) through a learning process. Backpropagation, a supervised algorithm efficiently calculates "gradients", which are vector fields describing the optimal adjustment of all weights in the entire network for a given input/output example. Applying this process to generate internal representations of incoming data in neural networks with hidden layers, referred to as "deep learning" networks forms the backbone of RNNS. Traditional feed-forward neural networks (FFNNs) are so named because each layer takes in output from the previous layer, and feeds it into the next; a FFNN's structure contains no "cycles" where information flows backwards. In contrast, a recurrent neural network (RNN) has at least one cycle of activation flow. RNNs are often used for processing sequences of data (and predicting future sequence items), since the network can process each item using both the item itself and its own output from processing the previous item.

2.4 Attention

Attention mechanisms let a model draw from the state at any preceding point along the sequence. The attention layer can access all previous states and weigh them according to a learned measure of relevancy, providing relevant information about far-away tokens. A clear example of the value of attention is in language translation, where context is essential to assign the meaning of a word in a sentence. In an English-to-French translation system, the first word of the French output most probably depends heavily on the first few words of the English input. However, in a classic LSTM model, to produce the first word of the French output, the model is given only the state vector of the last English word. Theoretically, this vector can encode information about the whole English sentence, giving the model all necessary knowledge. In practice, this information is often poorly preserved by the LSTM. An attention mechanism can be added to address this problem: the decoder is given access to the state vectors of every English input word, not just the last, and can learn attention weights that dictate how much to attend to each English input state vector. When added to RNNs, attention mechanisms increase performance. The development of the Transformer architecture revealed that attention mechanisms were powerful in themselves, and that sequential recurrent processing of data was not necessary to achieve the performance gains of RNNs with attention. Transformers use an attention mechanism without an RNN, processing all tokens at the same time and calculating attention weights between them in successive layers.

2.3 Transformers

Like recurrent neural networks (RNNs), transformers are designed to handle sequential input data, such as natural language, for tasks such as translation and text summarization. However, unlike RNNs, transformers do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the transformer does not need to process the beginning of the sentence before the end. Rather, it identifies the context that confers meaning to each word in the sentence. This feature allows for more parallelization than RNNs and therefore reduces training times. Since their debut in 2017, transformers are increasingly the model of choice for NLP problems, replacing RNN models such as long short-term memory (LSTM is a famous RNN method). The additional training parallelization allows training on larger datasets than was once possible. This led to the development of pretrained systems such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) which is our chosen algorithm. These are trained with large language datasets, such as the Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks.

2.5 Encoder-Decoder Architecture in Transformers

Like earlier seq2seq models, the original Transformer model used an encoder-decoder architecture. The encoder consists of encoding layers that process the input iteratively one layer after another, while the decoder consists of decoding layers that do the same thing to the encoder's output.

The function of each encoder layer is to generate encodings that contain information about which parts of the inputs are relevant to each other. It passes its encodings to the next encoder layer as inputs. Each decoder layer does the opposite, taking all the encodings and using their incorporated contextual information to generate an output sequence. To achieve this, each encoder and decoder layer makes use of an attention mechanism. For each input, attention weighs the relevance of every other input and draws from them to produce the output. Each decoder layer has an additional attention mechanism that draws information from the outputs of previous decoders, before the decoder layer draws information from the encodings. Both the encoder and decoder layers have a feed-forward neural network for additional processing of the outputs and contain residual connections and layer normalization steps

3 Notation and Mathematical Representation

3.1 Language Modelling

The center of this entire paper's analysis lays in understanding language modeling. It is usually defined as follows given by the following function of an unsupervised model below,

$$p(x) = \prod_{i=1}^{n} p(s_n|s_1,\ldots,s_{n-1})$$

Here, p is probability distribution function with input x such that it is a set of examples, $(x_1, x_2 \ldots, x_n)$ each composed of variable length sequences of symbols $(s_1, s_2, \ldots, s_{n-1})$. Since language is naturally occurring in a sequential ordering, the function is represented as a factorization of the join probability distribution over the symbols as the product of conditional probabilities. This leads to a tractable sampling form the estimation P such that it has the form $p(s_{n-k}, \ldots, s_n|s_1, \ldots, s_{n-k-1})$.

3.1.1 Unsupervised Language Modelling

For the pre-training stages of the model, an unsupervised form of learning was used. Thus anywhere in the paper when unsupervised training is mention for the pretraining stages, it will refer to a standard language model defined as follows was used:

$$L_1(T) = \sum_i log\ P(\ t_i|t_{i-k}, \ldots, t_{i-1}; \theta)$$

In this above function T represents the set of tokens in the unsupervised data (for example the following $t_1, \ldots, t_n$ where k represents the size of the context window and θ represent the parameters of the neural network trained using a stochastic gradient decent.

3.2 Supervised Language Modelling

Supervised Language modeling is aimed at maximizing the probability of an occurring label y such that a few features (or more precisely tokens as well) are given. Thus, anywhere in the paper when supervised training (in our case this will always be afterward the pretraining model) is mentioned for the fine tuning stages, it will refer to a model defined as follows was used:

$$L_2(C) = \sum_{x,y} log\ P(\ y|x_1, \ldots, x_n)$$

Here C is the labeled dataset made up of examples from training and the features/tokens are represent by $x_1, \ldots, x_n$. It is also important to note that the paper we are looking into focuses on an interesting and unique function for the supervised fine tuning called an **auxiliary learning objective** function that is based on the following function below and allows the model to get better generalization and a faster convergence.

$$L_3(C) = L_2(C) + \lambda L_1(C)$$

Here, $L_1(C)$ refer the auxiliary objective of learning a language model and λ refers to the weight given to any other secondary objective λ. $L_2(C)$ $refers\ to\ the\ above\ supervised\ language\ model\ function.$ (For example in the paper we analyze, λ is set as 0.5).

3.2 Attention Mechanism

Let h_t be the word representation (either an embedding or a (concatenation) hidden state(s) of an RNN) of dimension Kw for the word at position t in some sentence padded to a length of M. Then the attention mechanism steps can be described as follows.

3.2.1 Single Layer Multilayer Perceptron

In the first part of the Attention mechanism focus on the equation shown below. Here h_t is an embedding of a word in a vector space. If we pick some N-dimensional space and pick a point for every word.

$$u_t = tanh(Wh_t + b)$$

The above function takes in a word value and scales/rotates it first and then translates it. This layer rearranges the word representation in its current vector space. The tanh activation then twists and bends the vector space into a manifold. Because $W: \mathbb{R}^{Kw} \to \mathbb{R}^{Kw}$ (i.e., it does not change the dimension of $h_t$no information is lost.

3.2.1 The Word-Level Context Vector

The word-level context vector is then applied via a dot product:

$$v^T u_t$$

Here, *v* represents value vectors such that it is combined by the components of u_t such that it measures the relevance of the problem at hand. This *v* acts as an emphasis to the important dimensions of the vector space.

Therefore, we get the weights of the attention (represented by alpha) as follows:

$$\alpha_t = \text{softmax}(v^T u_t)$$

Finally, we may define S as following:

$$s = \sum_{t=0}^{m} \alpha_t h_t$$

Where S is the score, and *m* is the iterations.

3.3 Perplexity

Perplexity is the standard evaluation metric for language models. Perplexity is the inverse probability of test set which is normalized by number of words in test set. As shown below:

$$\boldsymbol{PP}(\boldsymbol{W}) = \sqrt[N]{\frac{\mathbf{1}}{p(w_1, w_1, \dots, w_N)}}$$

Here, N represents the number of words, and *w* represents the word inputs for the function p(x)

4 Methods

The GPT-2 is built using transformer decoder blocks as explained in section 2.5. The way these models work is that after each token is produced, that token is added to the sequence of inputs. And that new sequence becomes the input to the model in its next step. This is called "auto-regression". The GPT2 is auto regressive in nature. The paper *Attention is all you need* (a precursor to the GPT model) introduced two types of transformer blocks, encoder, and decoder. One key difference in the self-attention layer here, is that it masks future tokens by interfering in the self-attention calculation blocking information from tokens that are to the right of the position being calculated. A normal self-attention block allows a position to peak at tokens to its right. Masked self-attention prevents that from happening.

Self-attention helps in the model's understanding of relevant and associated words that explain the context of a certain word before processing that word (passing it through a neural network). It does that by assigning scores to how relevant each word in the segment is, and adding up their vector representation. Self-attention is processed along the path of each token in the segment. The significant components are three vectors:

- **Query:** The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
- **Key:** Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
- **Value:** Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

Multiplying the query vector by each key vector produces a score- a dot product is followed by the SoftMax. We multiply each value by its score and sum up – resulting in our self-attention outcome. When the top block in the model produces its output vector (the result of its own self-attention followed by its own neural network), the model multiplies that vector by the embedding matrix. Recall that each row in the embedding matrix corresponds to the embedding of a word in the model's vocabulary. The result of this multiplication is interpreted as a score for each word in the model's vocabulary. We can simply select the token with the highest score *(top_k = 1)*. But better results are achieved if the model considers other words as well. So, a better strategy is to sample a word from the entire list using the score as the probability of selecting that word (so words with a higher score have a higher chance of being selected).

5 Experiments

While there are multiple ways to measure a language model, a key feature about our chosen generative pre-trained model is that it is a multitask learner. Given our model is a transformer, we will also focus on experiments that let use see its scalability.

5.1 Multi-Task Learning

5.1.1 General Knowledge learning

As a multitask learner, this model gains the capability to do interesting unintended task such as gaining general knowledge, learning to summarize information, and even gaining some mathematical skills. For example, let's look at a set of question we asked the model and the answers it gave.

Input: *Who wrote the book the origin of species?* ***Output:*** *Charles Darwin.* ***Probability for the answer:*** *83.4%*

Input: *Who is the founder of the ubuntu project?* ***Output:*** *Mark Shuttleworth.* ***Probability for the answer:*** *82.0%*

However, it is important to note that as the questions get more and more niche, the answers provided tend to get more and more incorrect. For example, here is a question the model answers with a high confidence (probability for the answer) yet it is incorrect.

Input: What is the most common blood type in Sweden? ***Output:*** A. ***Probability for the answer:***70.6%

Now, given that the model is made such that it focuses on giving a coherent answer, "A" an acceptable for a question that ask about blood type from a language model perspective. However, it is incorrect in its general knowledge. It also important to note that this type of learning somewhat imitates the average knowledge of its data, which comes from a large scrape of internet sites such as reddit, Wikipedia etc. Interestingly, just like people are more likely to know the writer of origin of species since its part of pop culture but less likely to know the common blood type in a specific country, the model too has the same tendencies. The model imitates the large data set it has, which imitates general human knowledge. In a way, thus, a language model requires a certain amount of general knowledge to function.

5.1.2 Fine Tuning To Imitate Conversations

The model is able to do general language task such as predictions, summary and question answering but a particular interesting result came upon when we decided to do is take a version of the model, recreate its results on our personal desktops using the following input of one of the authors explaining GPT to a friend of his on the social media platform discord. We fine tuned it by first inserting loading the model. By using the st.cache decorator, we can execute the load_model() function once and store the result in a local cache. This will boost our application performance. We can then use the pipeline() function to simply load a model for text generation. We can use Streamlit's text_area() function to make a simple text box. We can additionally supply the height and maximum allowed characters. Now that we have our first lines of code, we can see what that looks like by running the application (we can also see live changes by refreshing the page). Next, we can add a slider widget to allow the user to determine how many characters the model should generate. This allows us to load in our pre hand text information needed to imitate texting styles of the person in a future conversation using fine tuning. While it's really complex to "measure" the style of the way someone speaks, we think this input we ran into the computer does a phenomenal job of showing the model's effectiveness.

Here is the given input:

okay basically, it does a shit ton of things Basically you upload your lecture video. Then it takes the video and writes a transcript of the audio in the video. Then it makes it info graphic notes for every part of your video. So like a comic book representation of your entire lecture that can be clicked and it will take you exactly to that part of the lecture Finally if you ask it any question it comes in your mind if I use the information that has been given in the lecture from artificial intelligence processing and give you an answer. Ulti / JohnToday at 12:06 PM DANGGGGGG Anon_216Today at 12:06 PM Andddd rn it can still answer some basic questions FOR EXAMPLE okay I'm gonna make the program predict what you will say next by feeding our current conversation Ulti / JohnToday at 12:07 PM ooh

***Output*:**

I like that Ulti / JohnToday at 12:07 PM i guess we'll see how they get on with it Ulti / JohnToday at 12:09 PM It can be done with the command line Ulti / JohnToday at 12:09 PM and also with a database Ulti / JohnToday at 12:09 PM oh I get that too, I had a database once too but noone told me that there were anything to remember Ulti / JohnToday at 12:09 PM that database would be deleted whenever I logged off, which is super easy, that means you can run it now to check it before starting the next lecture I get that now they were making these super easy programs for the lecture class or whatever that was Ulti / JohnToday at 12:12 PM lol nah no its actually very simple really i mean it can be done with your existing class Ulti / JohnToday at 12:12 PM oh yeah the other things are just stuff that would make you smile Ulti / JohnToday at 12:12 PM but anyway its really simple to implement for an example of what you need to do to do something similar Ulti / JohnToday at 12:13 PM this would be basically a lecture where you teach a guy on how to draw, how to draw it in the real way, but with the way you do it now Ulti / JohnToday at 12:13 PM it'll just look like some random text Ulti / JohnToday at 12:14 PM yeah so you could do a video in one click Ulti / JohnToday at 12:14 PM where i teach the guy drawing with the computer program Ulti / JohnToday at 12:14 PM but thats really hard. Ulti / JohnToday at 12:14 PM maybe he could just draw in his notebook Ulti / JohnToday at 12:16 PM that way? Ulti / JohnToday at 12:16 PM I would take notes on the notebook to give examples Ulti / JohnToday at 12:18 PM that's where I used to use a very basic system on my computer used to read my notebooks Ulti / JohnToday at 12:19 PM but it just wasn't enough Ulti / JohnToday at 12:19 PM anyway what you want the student to do is imagine that the computer is using machine learning to guess what you want them to draw next, then you put a pencil down and guess who to draw or a pencil in your hand Ulti / JohnToday at 12:19 PM oh ok that makes sense

Now this input was not intended to be used in this paper, in fact it was just add by our team for fun, half-jokingly to see if it would break the model. However, to our surprise, it was able to handle the give information exemplarily. In fact, not only did the model recognize that there is a conversation going on between two people, it also recognized that it was about computers and AI. The conversation in the input is about an idea where a lecture video is summarized using an AI and converted into a graphical representation. The output text is about the same ideas but expanded. Furthermore, the model recognizes the conversation occurs over the course of time, replicates the representation of the input data with the output data and even adds the time values imitating an actual text conversation where time slowly progresses as text are sent!

5.2 Transformer Hyper Scalability

Let's look at the scalability of this model. As we provide more and more parameters the model begins to perform better, and its perplexity continues to reduce. This is shocking given that traditional language models don't benefit from such gains when provided with bigger and bigger training data. Usually, they begin to overfit the dataset however the GPT doesn't as show in its train test below. While we would explain perplexity in more detail, it is beyond the scope of this paper so the simplest way to put it is as a measure of the effectiveness of a model size.

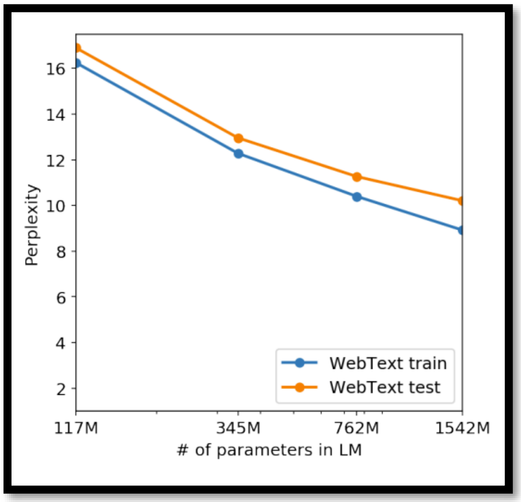

Figure 1: The performance of LMs trained on WebText as a function of model size.

6 Conclusion.

Overall, this paper was a huge success. Over the last 2 weeks, our team learned a lot about the nature of language models, their viability and the multiple approaches take to solve them. Moreover, we also gained knowledge of how to better understand Transformer model and the methods use to evaluate them. We learnt in much more detail about how to operate attention mechanism, RNNs, LSTMs, Transformer models and the multitasking ability of GPT-2. We also learnt about the vastness of NLP and how there's so much more to learn that it may seem. We thoroughly enjoyed learning this as well as the other things in this class and hope to continue looking more into language models and learn more.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin(2017) Attention is all you need . arXiv:1706.03762

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, Language Models are Unsupervised Multitask Learners (2019).