Weights & Biases

LLMをゼロからトレーニングするためのベストプラクティス

Rebecca Li, Andrea Parker, Justin Tenuto 著
シバタ アキラ 監訳

Table of Contents

はじめに

2017年に発表されたトランスフォーマーのブレークスルーからまだ数年しか経っていませんが、LLMはすでに性能、コスト、そして将来性において大きく成長しています。W&Bでは、LLMを開発に取り組むチームをどの企業よりも多く見てきました。しかし、重要な詳細や肝心なポイントの多くは、口伝えで伝えられていることが多いのです。

このホワイトペーパーの目的は、独自のLLMをゼロからトレーニングするためのベストプラクティスを集約することです。スケーリングやハードウェアから、データセットの選択、モデルトレーニングに至るまで、すべてをカバーし、考慮すべきトレードオフを検証するとともに、その過程で起こりうる落とし穴を指摘します。LLMをスクラッチで（ゼロから）トレーニングする際のキーとなるステップや考慮すべき点について、網羅的に説明することを意図しています。

そもそも、スクラッチでトレーニングすることが、自分の組織に合っているのかどうかということを考える必要があるでしょう。まずはそこからスタートしましょう：

学習済みLLMモデルの構築と購入の比較

LLMの事前学習トレーニングを始める前に、まず気になるのは、LLMの事前学習を自分で行うか、既存のものを利用するかということです。基本的なアプローチは3つあります：

1. 商用LLMのAPIを利用する：例：GPT-3（OpenAI、2020）、Cohere APIs、AI21 J-1
2. 既存のオープンソースLLMを使用する：例：GPT-J（EleutherAI、2021）、GPT-NeoX（EleutherAI、2022）、Galactica（MetaAI）、UL2（Google、2022）、OPT（Meta-AI、2022）、BLOOM（BigScience、2022）、メガトロンLM（NVIDIA、2021）、コードジェン（Salesforce、2022）
3. 自ら、もしくは支援を受けて、LLMを事前学習する：自分でトレーニングを行うか、LLMのコンサルタントやプラットフォームを使うかです。例えば、Mosaic MLは、LLMに特化したトレーニングサービスを提供しています。

とはいえ、選択する際に考慮しなければならない細かい点はたくさんあります。ここでは、各オプションの長所、短所、適用可能なシナリオを紹介します：

Option 1 商用LLMのAPIを使う	Option 2 既存のオープンソースLLMを利用する	Option 3 自らもしくは外部支援を受けてLLMをトレーニングする
長　所		
● 必要なLLMトレーニングの技術力が最も少ない。 ● 主なコストは推論時に発生するため、トレーニングや探索のコストを最小限に抑えることができる。 ● 最もデータ要求の少ない選択肢。モデルが推論を行うために必要なのは、わずかな例だけ（あるいは例なし）でよい。 ● 市場で最もパフォーマンスの高いLLMを活用し、優れた体験を構築できる。 ● LLMモデルを使ったアプリの市場投入までの時間を短縮し、プロジェクトのリスクを軽減できる。	● ①と比較すると、LLMサービスプロバイダーの将来の方向性への依存度が低いため、ロードマップや後方互換性をコントロールしやすい。 ● ③と比較すると、LLMをゼロから構築しないため、データ、トレーニング時間、トレーニング予算が少なくて済み、Time-to-Valueが大幅に短縮される。 ● LLMが膨大なインターネットデータから学んだことを活用し、推論時に知財にお金を払うことなく、構築できる。	● ①、②に比べ、LLMのパフォーマンスと将来の方向性を最もコントロールできるため、技術革新や下流タスクへのカスタマイズの自由度が高い。 ● モデルの品質、バイアス、有害性の問題に直接影響する学習用トレーニングデータセットを完全にコントロールすることができる。これに対し、①や②では、これらの問題をコントロールすることは不可能。 ● LLMをトレーニングすることで、強固な優位性を築くことができる。水平方向のユースケースや垂直方向に合わせたLLMの優れたパフォーマンスにより、特にLLMの展開を可能にするデータ/フィードバックループを作成した場合、持続的な優位性を構築することができる。

Option 1 商用LLMのAPIを使う	Option 2 既存のオープンソースLLMを利用する	Option 3 自らもしくは外部支援を受けてLLMをトレーニングする
短　所		
● 商用LLMサービスは、微調整や推論タスクが大量に発生すると高価になることがある。これは、LLMの総所有コスト(TCO)を各推論に償却したものです。 ● 多くの業界やユースケースでは、コンプライアンス上、機密データやPIIデータの閲覧を許可できないため、商用LLMサービスの利用を禁じている(例えば、ヘルスケアユースケース)。 ● 外部アプリを構築する場合、外部のLLMサービス技術への依存度が高い場合は、他の強みを探し、事業のリスクを軽減する必要がある。 ● 下流の柔軟性が低い：エッジ推論をサポートしていない、モデルをカスタマイズする機能が限られている(微調整をするとコストがかかる)、モデルを継続的に改善する機能が限られている。	● 自分で作る時ほどではないが、オープンソースのLLMをトレーニングし、微調整し、ホストするためには、多くのドメイン専門家のスキルが必要である。LLMの再現性は依然として重要な問題であるため、必要な作業量と時間は過小評価できない。 ● より垂直な技術スタックのため、下流のアプリを構築している場合、市場投入までの時間が遅く、アジャイル性が低い。 ● オープンソースのモデルは、通常、商用モデルと比較して数ヶ月から数年単位で性能が遅れている。競合他社が商用モデルを利用している場合、彼らはLLM技術で優位に立っている可能性があるため、他の競争優位性を見つける必要がある。	● 非常に高価で、リスクも高い。NLP/ML、専門知識、ソフトウェア、ハードウェアの専門知識など、領域横断的な知識が必要。うまくやらないと、最適でないモデルで何千ドル、何百万ドルも費やしてしまったという事態になりかねない。特にトレーニングの後半でのミスは、修正・解消するのが難しい。 ● ②より効率が悪い。②は、インターネット上の全データから学習した既存のLLMを活用し、確度の高い初期モデルを提供することができる。③では、ゼロから始めることになり、モデルが汎化性能を獲得するためには、高品質で多様なデータセットが膨大に必要。
各選択肢を検討するタイミング		
● 技術チームは少ないが、LLM技術を活用してダウンストリームアプリを構築したい、あるいはパフォーマンス上の理由からクラス最高のLLMを活用したい(LLM技術をアウトソーシングする)場合に最適。 ● トレーニングデータセットが非常に限られていて、LLMのゼロ／少数回ショットの学習機能を活用したい場合。 ● アプリのプロトタイピングや、LLMで何が可能かを探索したい時。	● ②と③の間では、モデルのアーキテクチャーを変えようとしないのであれば、既存の学習済みLLMを使って微調整するか、既存の学習済みLLMのモデルを出発点として追加学習を行う方が良い場合がほとんどである。GPT-NeoXのような優れた学習済みLLMは、すでに膨大な量のデータから学習しているため、汎化性能を獲得している。特に手元のトレーニングデータセットが巨大でなく、多様でない場合、その学習を活用することができる。 ● 規制の厳しい環境にいたり、ユーザーデータや機密データがあり、市販のLLMサービスには対応できない場合。また、レイテンシーや場所の制約があり、モデルのエッジ展開が必要な場合もある。	● 既存の学習済みLLMからモデルアーキテクチャーや学習データセットを変更する必要がある場合に最適。例えば、異なるトークナイザーを使用したい場合、語彙サイズを変更したい場合、隠れ次元、アテンションヘッド、層の数を変更したい場合など。 ● 一般的に、この場合、LLMはビジネス戦略や技術的な優位性の中核をなすという前提がある。LLMトレーニングの一部または多くのイノベーションを目的とし、高価なモデルを継続的にトレーニングし、維持するための莫大な投資計画がある場合。 ● LLMサービスに関連して蓄積された多くの独自データを持ち、持続的な競争優位のための継続的なモデル改善ループを構築している、またはする予定がある場合です。

また、非常に的を絞ったユースケースしかなく、LLMによる汎用的な機能や生成能力が必要ない場合は、もっと小さなトランスフォーマーや他のもっとシンプルなディープラーニングモデルのトレーニングや微調整を検討するのも一案です。その結果、複雑さが軽減され、学習時間も短縮され、継続的なコストも削減されるでしょう。

スケーリング法則

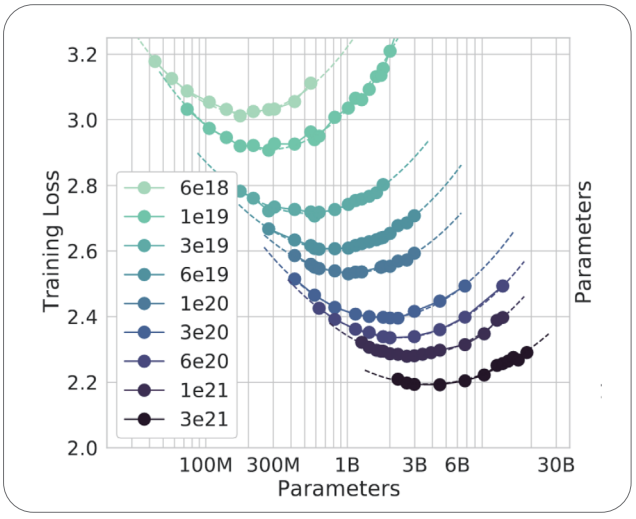
トレーニングに入る前に、LLMがどのようにスケールするかについて説明することが重要です。スケーリングを理解することで、モデルの大きさや複雑さと、トレーニングに使用するデータの大きさのバランスを効果的に計画することができます。

ここで関連する歴史をいくつか紹介します：OpenAIはもともと2020年に「LLMのスケーリング方則」を発表しています。データサイズを拡大するよりも、モデルサイズを拡大する方が重要であることを示唆したものです。これは、DeepMindがほぼ正反対のことを提案するまでの約2年間、支持されていました。DeepMindによると、それまでのモデルは著しく学習不足であり、基礎となるトレーニングデータセットを増やすことが、実際の性能向上につながるということです。

この主張が提案されたのは2022年のことです。具体的には、DeepMindがTraining Compute-Optimal Large Language Modelsという論文で、別のアプローチを提唱し、現在のLLMは実は著しく学習不足であることを発見しました。簡単に言えば、これらの大規模モデルは十分なデータでトレーニングされていなかったという主張です。

DeepMindは、上記のGopherモデルの4分の1のサイズでありながら、4.6倍のデータでトレーニングされたChinchillaというモデルでこれを示しました。このようにサイズが小さくても、はるかに多くのトレーニングデータを使用した場合、ChinchillaはGopherや他のLLMを凌駕しました。

注：NLPにおけるトークン化とは、テキストをトークンと呼ばれる小さな単位に分離する重要なステップです。トークンは単語、文字、サブワードのいずれかになります。トレーニングトークンの数は、トークン化後のトークン形式のトレーニングデータの大きさを表します。詳細なトークン化の手法については、後ほど詳しく説明します。



各曲線の最小値の左側は、モデルが小さすぎる -- より少ないデータでトレーニングされたより大きなモデルが改善されるであろう領域。各曲線の極小値の右側では、モデルが大きすぎる--より多くのデータでトレーニングされたより小さなモデルであれば改善される。最良のモデルは、最小値にある。（出典：Training Compute-Optimal Large Language Models）

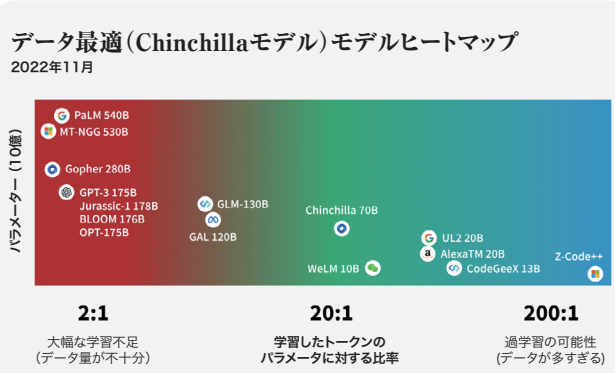
DeepMindは、最適なパフォーマンスを得るためには、モデルサイズとトレーニングトークンの数（データ量）をほぼ同じ割合で増加させるべきだという研究成果を発表しています。計算量が10倍になるなら、モデルを3.1倍大きくし、トレーニングするデータを3.1倍大きくする必要があり、計算量が100倍になるなら、モデルを10倍大きくし、データを10倍大きくする必要があります。

DeepMindは、さまざまなサイズのモデルを最適にトレーニングするために必要なトレーニングデータと計算量を示す以下の表を提供しています。

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

様々なモデルサイズにおける最適トレーニングFLOP数とトレーニングトークンの推定値

とはいえ、既存のLLMのほとんどは、まだ学習不足です：



データ/コンピュータ最適（Chinchilla）ヒートマップ、チンチラデータ最適スケーリング法則：

つまり、LLMモデルのサイズを選ぶ際の現在のベストプラクティスは、大きく分けて2つのルールに基づくといえます：

- 使用するデータセットを決め、データサイズに基づいてChinchilla最適なモデルサイズを求める（またはデータ収集制限の範囲内でチンチラ最適に近いサイズを求める）
- トレーニングの計算予算と推論のレイテンシー要件に基づいて、モデルに最適なデータとモデルサイズの組み合わせを決定する

ハードウェア

LLMの学習が、ハードウェアに負荷のかかる作業であることは驚くことではありません。ここでは、現在のモデルの以下の例が良い参考となります：

- PaLM（540B、Google）：TPUv4チップを合計6144個使用し、モデルとデータの並列性を組み合わせてデータセンターネットワーク（DCN）で接続された2つのTPUv4 Podsで構成
- OPT（175B, MetaAI）：992個の80GB A100 GPUを搭載し、Megatron-LMテンソル並列による完全共有データ並列を活用
- GPT-NeoX（20B、EleutherAI）：40GBのA100 GPUを計96枚搭載
- Megatron-Turing NLG（530B、NVIDIA&MSFT）：560 DGX A100ノード、各クラスター・ノードにNVIDIA 80-GB A100 GPUを8基搭載

LLMのトレーニングは、2つの大きな理由から、インフラ上困難なものとなっています。まず、最大級のGPU（NVIDIA 80GB-A100など）でもメモリにすべてのモデルパラメータを収めることはもはや不可能であるため、並列アーキテクチャが必要になります。もうひとつの課題は、アルゴリズム、ソフトウェア、ハードウェアスタックを同時に最適化し得ない場合、莫大な計算量が非現実的なほど長いトレーニング時間を必要とすることです（たとえば、1750億パラメータを持つGPT-3をV100 NVIDIA GPUでトレーニングすると、約288年かかります）。

メモリとコンピューティングの効率性

何千もの分散型GPUのポテンシャルを最大限に引き出すには、メモリと計算効率のバランスを取るために、アーキテクチャーに並列性を設計することが不可欠です。

メモリ効率

LLMのトレーニングには、モデルの重み、勾配、オプティマイザーの状態のために何テラバイトもの集約メモリが必要であり、1つのGPUで利用できる量をはるかに超えています。典型的な緩和策の1つは勾配累積で、トレーニングバッチ全体をマイクロバッチに分割して順番に処理し、その結果得られた勾配をモデルの重みを更新する前に累積します。つまり、トレーニングバッチのサイズは、常駐する活性化メモリのピークを増加させることなく拡張することができます。

演算効率

大規模なGPUクラスターでは、何千もの高スループットGPUを搭載することができますが、この規模で高い計算効率を達成することは困難です。バッチサイズを大きくすることはGPUカーネルの演算密度を高め、通信や同期のために滞留してしまう時間を相殺することができるため、計算効率を高める効果的な方法となり得ます。しかし、バッチサイズを大きくしすぎると、モデルの品質に悪影響を及ぼす可能性もあります。

並列化は最重要ですが、その方法はさまざまです。その中から、最も一般的なものを紹介します。

並列化のためのテクニック

並列化とは、タスクを分割して複数のプロセッサやGPUなどのデバイスに分散させ、同時並行で完了させることです。これにより、単一のプロセッサやデバイスで実行する場合と比較して、計算リソースをより効率的に使用し、完了時間を短縮することができます。複数のGPUで並列化されたトレーニングは、トレーニングプロセス全体の所要時間を短縮する有効な方法です。

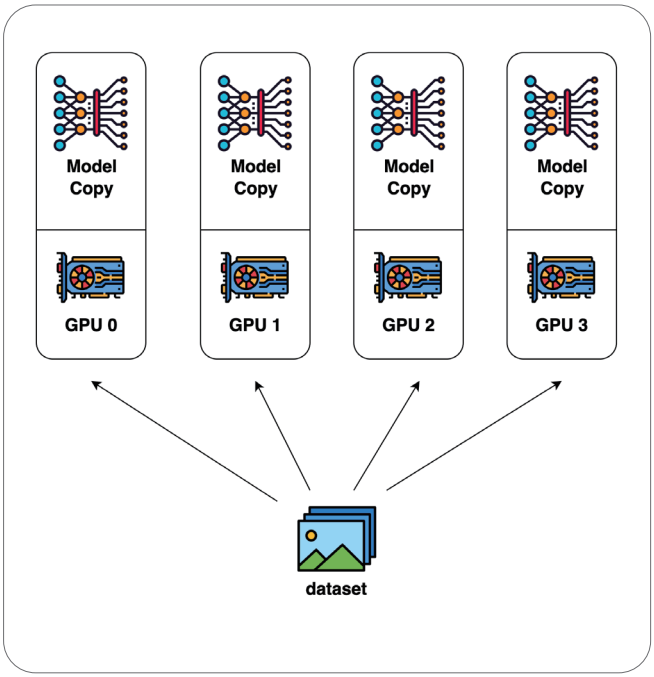
トレーニングの並列化には、勾配累積、マイクロバッチ、データ並列化、テンソル並列化、パイプライン並列化など、いくつかの異なる戦略があります。典型的なLLMの事前学習では、これらの手法を組み合わせて採用しています。下記にそれぞれを定義します：

データの並列化

ディープラーニングのワークフローにおいて、1台の機械に収まりきらない大規模なデータセットを扱うには、データ並列化が最適かつ一般的なアプローチとなります。

具体的には、トレーニングデータを複数のシャード（パーティション）に分割し、様々なノードに分散させるデータ並列化です。各ノードは、まず自分のローカルデータを使ってサブモデルをトレーニングし、次に他のノードと通信して一定時間ごとに結果を組み合わせてグローバルモデルを得ます。データ並列化のためのパラメータ更新は、非同期または同期のいずれかを選択することができます。

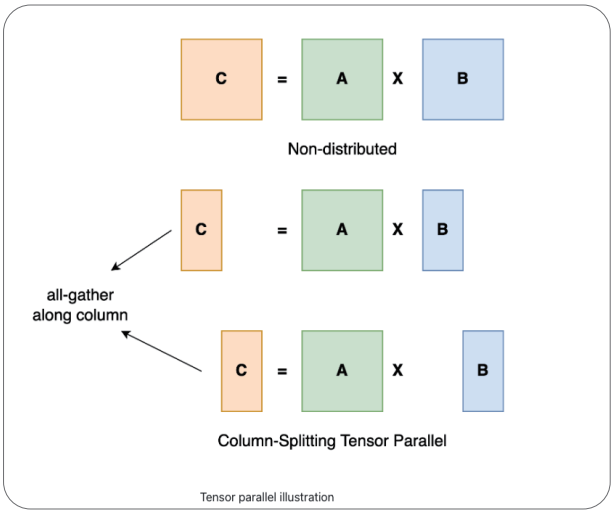
この手法の利点は、計算効率が上がることと、実装が比較的簡単であることです。最大の欠点は、逆伝播の際に、他のすべてのGPUに勾配全体を渡す必要があることです。また、モデルとオプティマイザーをすべてのワーカーに複製するため、メモリ効率が悪くなります。



テンソル並列化

テンソル並列は、大きな行列の乗算を小さなサブ行列の計算に分割し、それを複数のGPUを使って同時に実行するものです。

これにより、非同期であること、ノード間の通信オーバーヘッドを削減できることから、トレーニング時間を短縮することができます。この手法の利点は、メモリ効率が良いことです。しかし、欠点は、順伝播&逆伝播のたびに活性化の通信が追加で発生するため、効率的に行うには高い通信帯域が必要になることです。

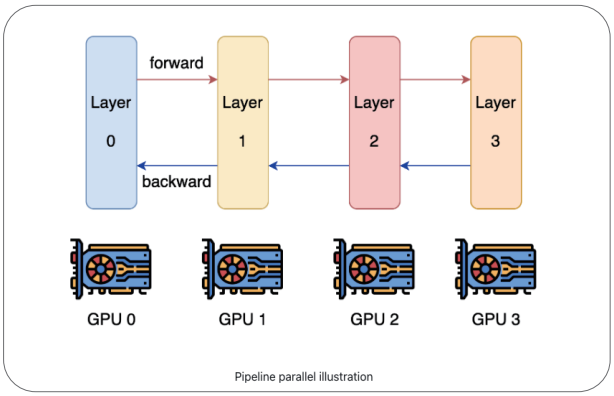


Tensor parallel illustration

パイプラインとモデルの同時並列化

パイプライン並列化は、モデルのレイヤーを並列処理できるステージに分割することで、ディープラーニングトレーニングのメモリ効率と計算効率の両方を向上させます。

これにより、通信のオーバーヘッドを最小限に抑えながら、全体のスループット速度を大幅に向上させることができます。開発フローは「層間並列」と考えることができます（テンソル並列は「層内並列」と考えることができます）。パイプライン並列と同様に、モデル並列は、モデルをGPU間で分割し、各モデルに同じデータを使用することで、各GPUはデータの一部ではなく、モデルの一部で動作します。パイプラインとモデルの同時並列の欠点は、パイプライン並列の程度がモデルの深さによって制限されるため、無限に拡張できないことです。



Pipeline parallel illustration

このセクションの冒頭で述べたように、トレーニング中に並列化技術を組み合わせて活用するチームは珍しくありません。例えば、PaLM（GoogleBrain、2022年）とOPT（MetaAI、2022年）では、テンソルモデルとデータの同時並列化を使用しました。

NVIDIAは、「Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM」という論文で、少し違ったアプローチをしています。彼らは、パイプライン、テンソル、データ並列を組み合わせたPTD-P技術を提案し、1000台のGPUで最先端の計算性能（ピークデバイススループットの52%）を達成しました。

具体的には、PTD-Pは、マルチGPUサーバー間のパイプライン並列、マルチGPUサーバー内のテンソル並列、データ並列を組み合わせて活用し、1兆個のパラメータを持つモデルを実用的にトレーニングします。また、この手法は、同一サーバー上のGPU間およびサーバー間の広帯域リンクを備えたオプティマイザー環境において、グレースフルスケーリングを採用しています。

これらの技術をLLMのトレーニングに使用するには、最高性能のGPUが効率的に動作するだけでなく、最適な通信のための広帯域ネットワーク--ノード間のデータ移動にはInfiniBandがよく使われる--が必要です。

しかし、これには当然ながらコストがかかります。LLMをトレーニングするために、何千もの高性能GPUと広帯域ネットワークを活用することは、しかるべきインフラを必要とします。たとえばPaLMモデル（540B、Google）のコストは、単純計算で$23Mにも上ると見積もられています（詳細な分析を参照してください）。

分散ディープラーニングトレーニングシステムを実装するには、Distributed TensorFlow、Torch Distributed、Horovodなどのソフトウェアツールキットや、DeepSeed、Megatronなどのライブラリが必要になることが多いです。ここには実装の複雑さがあるので、成功させるのであればシステムの専門知識が必要です。

また、並列化を実現するために、以下のような技術や戦略が一般的に採用されています：

勾配累積

勾配累積は、複数のバッチの勾配を加算した後、累積されたすべての勾配に対して1回の重み更新ステップを実行する。

このアプローチでは、1回の最適化ステップで十分な勾配を蓄積した後、再び互いに同期するまで、GPUがそれぞれのローカルバッチのデータで独立して作業することで、GPU間の通信オーバーヘッドを削減します。

非同期確率的勾配降下最適化法

また、複数のGPUでモデル最適化を行う場合、非同期の確率的勾配降下最適化手法を採用することができます。

この手法は、すべてのデータを一度に読み込むのではなく、各ノードのデータの小さなサブセット（マイクロバッチ）を使用するため、メモリ要件を削減しながら、非同期的な性質により高速な収束率を実現します。その仕組みは次の通りです：

- まず、現在のミニバッチを処理するのに必要なモデルの最新パラメータをパラメータサーバーから取得する。
- 次に、これらのパラメータに関する損失の勾配を計算する。
- 最後に、これらの勾配はパラメータサーバーに返され、パラメータサーバーはそれに応じてモデルを更新します。

マイクロバッチング

マイクロバッチングは、小さなミニバッチを大きなバッチに結合することで、より多くのバッチをより短時間で、逆伝播演算中のデバイス間の同期ポイントを少なくして処理することを可能にします。メモリ消費を抑え、スケーラビリティを向上させることができるため、多くのGPUで非常に大規模なモデルをトレーニングする際に、ますます人気が高まってきています。全体として、マイクロバッチングは、非常に大きなデータセットや、大量の処理能力を必要とするモデルを扱う際に、分散ディープラーニング技術を活用するための効果的な方法です。

さて、ここまではスケーリング、ハードウェア、トレーニングの並列化のテクニックについて説明しましたが、次はLLMが実際に何を学習するのか、データを見てみましょう。

データセットの収集

悪いデータは悪いモデルにつながる。高品質で大量かつ多様なデータセットを慎重に処理することは、モデルの収束だけでなく、下流のタスクにおけるモデルの性能にも直接貢献します。

LLMは要求されるデータ量が非常に多いのが特徴です。また、データサイズに加えて、データの多様性が重要です。なぜなら、多様性はモデルのクロスドメイン知識を向上させ、下流の汎化能力も向上させるからです。

典型的なトレーニングデータセットは、webからスクレイピングされた公共データ、オンライン出版物や書籍のリポジトリ、GithubやWikipediaのコードデータ、ニュース、ソーシャルメディアの会話など、多様なソースからのテキストデータで構成されています。

例えば、The Pileを見てみましょう。The Pileは、EleutherAIが大規模な言語モデルのために作成した人気のテキストコーパスです。22のデータソースからのデータを含み、大まかに5つの大カテゴリーに分類されています：

- アカデミックライティング：PubMed Abstracts and PubMed Central、arXiv、FreeLaw、USPTO Backgrounds、PhilPapers、NIH Exporter
- オンラインまたはスクレイピングされたリソース：CommonCrawl、OpenWebText2、StackExchange、Wikipedia
- 散文：BookCorpus2、Bibliotik、プロジェクトグーテンベルク
- ダイアログ：YouTube 字幕、Ubuntu IRC、OpenSubtitles、Hacker News、EuroParl
- その他：GitHub、DeepMind　Mathematicsのデータセット、Enronの電子メール

なお、The Pileデータセットは、一般公開されている数少ない大規模テキストデータセットの一つです。GPT-3、PaLM、Galacticaのような既存のモデルのほとんどは、トレーニングや評価のデータセットを一般に公開していません。LLMトレーニングのためにこれらのデータセットを集約し、前処理するのは大規模な労力を要するため、ほとんどの企業は競争優位性を保つために社内で保管してきました。そのため、The PileやAllenAIのようないくつかのデータセットは、一般の大規模なNLP研究目的には非常に貴重なものとなっています。

また、データセット収集の際、一般的なデータは専門家でなくても収集できますが、特定のドメインのデータは通常、医師、物理学者、弁護士などの専門家（SME）に収集または相談する必要があります。SMEは、NLPエンジニアが見逃してしまいそうなテーマやコンセプトのギャップを指摘することができます。LLMがどのように「データを表現することを学ぶか」SMEが見逃しそうなデータの異変やギャップを指摘する能力があることから、NLPエンジニアもこの段階で大きく関与する必要があります。

使用するデータセットを特定したら、そのデータをモデル用に準備する必要があります。では、その準備に入りましょう。

データセットの前処理

このセクションでは、重複排除やクリーニングなどのデータ処理と、さまざまなトークン化戦略の長所と短所の両方を取り上げます。まずは前者から：

データセットの取り扱いについて

トレーニングデータが高品質で多様であることを保証するために、学習のステップの前にいくつかの前処理を行うことができます：

データサンプリング

一部のデータの構成要素は、よりバランスのとれたデータ分布を得るためにアップサンプリングすることができます。ある研究では、フィルタリングされていないウェブデータのような低品質のデータセットをダウンサンプリングしています。また、モデルの学習目的に応じて、特定のドメインのデータをアップサンプリングする研究もあります。

また、データセットに適用されるトレーニング済み分類器モデルを使用するなど、高品質のデータをフィルタリングする高度な手法もあります。例えば、MetaAIのモデル、Galacticaは、科学研究に特化した

モデルで、特に科学知識の蓄積、関連付け、推論を目的として作られています。その目的から、学習用データセットは、主に論文、教科書、講義録、百科事典などの科学リソースからの高品質なデータで構成されています。また、このデータセットは、例えば、特定タスクごとに特化したデータセットなど、高度にキュレーションされており、新しいタスク向けに再合成することも可能です。

データクリーニング

通常、トレーニングの前にデータのクリーニングや再フォーマットが行われます。例えば、定型文の削除、HTMLコードやマークアップの削除などです。さらに、プロジェクトによっては、スペルミスの修正、ドメイン間の同形異義語の処理、バイアスや有害な内容の除去を行い、モデルの性能を向上させることもあります。他のプロジェクトでは、モデルは現実世界にあるデータをそのままを見ることで、モデル能力の一部としてスペルミスや有害性の対処を学ぶべきであるという考えから、これらのステップが使用されないこともあります。

非標準的なテキスト成分の取り扱い

場合によっては、非標準のテキスト成分をテキストに変換することが重要です。例えば、絵文字を対応するテキストに変換します：❄️ は "snowflake" というように。この変換は、もちろんプログラムで行うことができます。

データの重複排除

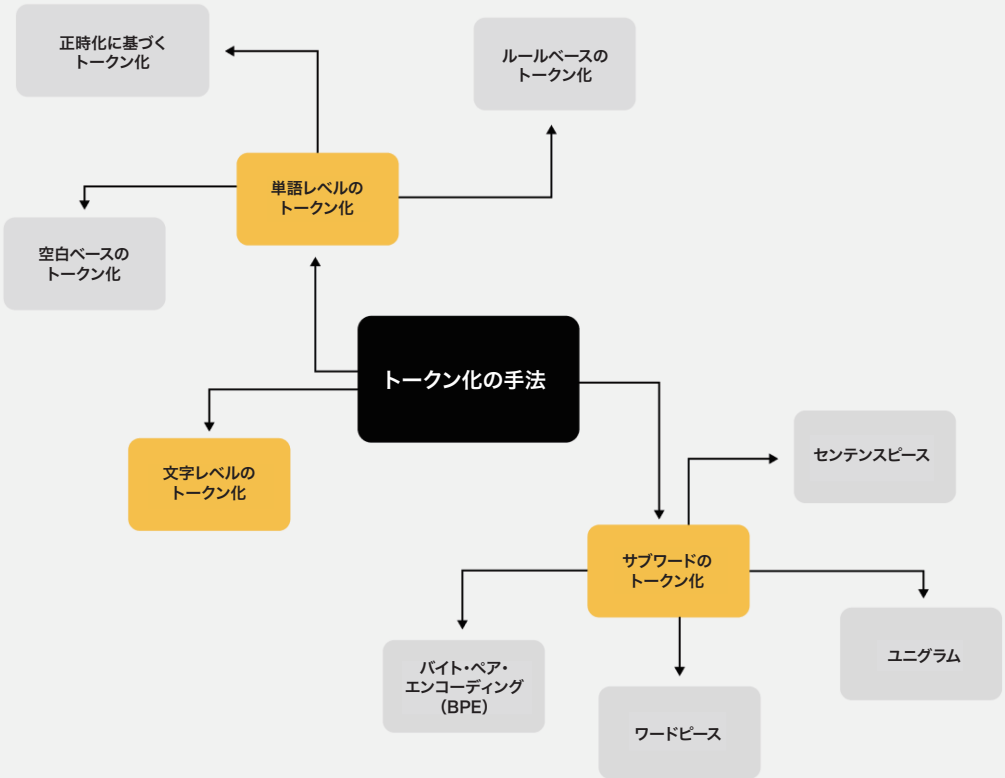
研究者の中には、トレーニングデータの重複排除をすることで大きなメリットがあると考える専門家もいます。LSH（Locality Sensitive Hashing）のようなファジーな重複排除手法がここでよく使われます。重複排除の詳細については、Deduplicating Training Data Makes Language Models Betterという論文を参照してください。

下流タスクデータ削除

データリーケージは、トレーニングに使用するデータに、予測しようとする情報が偶然含まれていた場合に起こります。評価データセットに含まれるトレーニングデータを除去するには、下流のタスクデータ除去手法（n-gramsなど）が必要です。

トークン化

トークン化とは、文字列をトランスフォーマーで読み取り可能な整数値のトークンIDにエンコードするプロセスです。最先端のLLMのほとんどは、ワードベースのアプローチとは対照的に、バイトペアエンコーディング（BPE）のようなサブワードベースのトークナイザーを使用しています。以下に、さまざまな手法の長所と短所を紹介します。特に、現在最も普及しているサブワード戦略に注目します。



最もよく使われるトークン化手法のまとめ
（出典：Two minutes NLP - A Taxonomy of Tokenization Methods）

トークン化手法	単語ベースのトークン化	文字ベースのトークン化	サブワードに基づくトークン化
トークナイザーの例	空間トークン化（スペースで文を分割）、ルールベースのトークン化（例：Moses、spaCy）	キャラクタートークナイゼーション（すべての文字に対して単純にトークン化すること）	バイトペアエンコーディング（BPE）、WordPiece、SentencePiece、Unigram（単語の一部と全体によるトークン化、上記図参照 ）
配慮事項	● デメリット：非常に大きな語彙を生成するため、入力層と出力層が巨大な埋め込み行列になる。類語に対してもそれぞれ別トークンが割り当てられてしまう。 ● トランスフォーマーモデルは、通常単一言語で学習する場足、語彙が5万語以下である。	● 個々の文字からすべての単語を組み立てることができるため、OOV（out of vocabulary）トークンとならず、少数の語彙で成立する。 ● デメリット：個々のトークンの意味が薄く、非常に長いシーケンスを生成しようとするため、モデルにとって意味のある入力表現を学習するのが非常に難しい。しかし、英語以外の言語では「1つの文字」が非常に豊富な情報を持つ場合もある。	● サブワードに基づくトークン化手法は、頻繁に使用される単語はより小さなサブワードに分割されるべきではなく、希少な単語は意味のあるサブワードに分解されるべきであるという原則に従っている。 ● メリット：単語ベースのトークン化と文字ベースのトークン化の欠点を解決し、ある程度の語彙サイズと意味のある学習済み文脈非依存表現の両方を実現する。

最終的には、トークン化技術の選択は、具体的なタスクと分析対象の言語によって決まります。

- 単語ベースのトークン化はシンプルで効率的ですが、複雑な言語への対応には限界があります。
- 文字ベースのトークン化は、単語の境界がはっきりしない言語では有効です。
- BPE、wordpiece tokenization、sentencepiece tokenization、unigram tokenizationなどのサブワードに基づくトークン化は、複雑な形態素や語彙のない単語を扱うのに特に有効です。

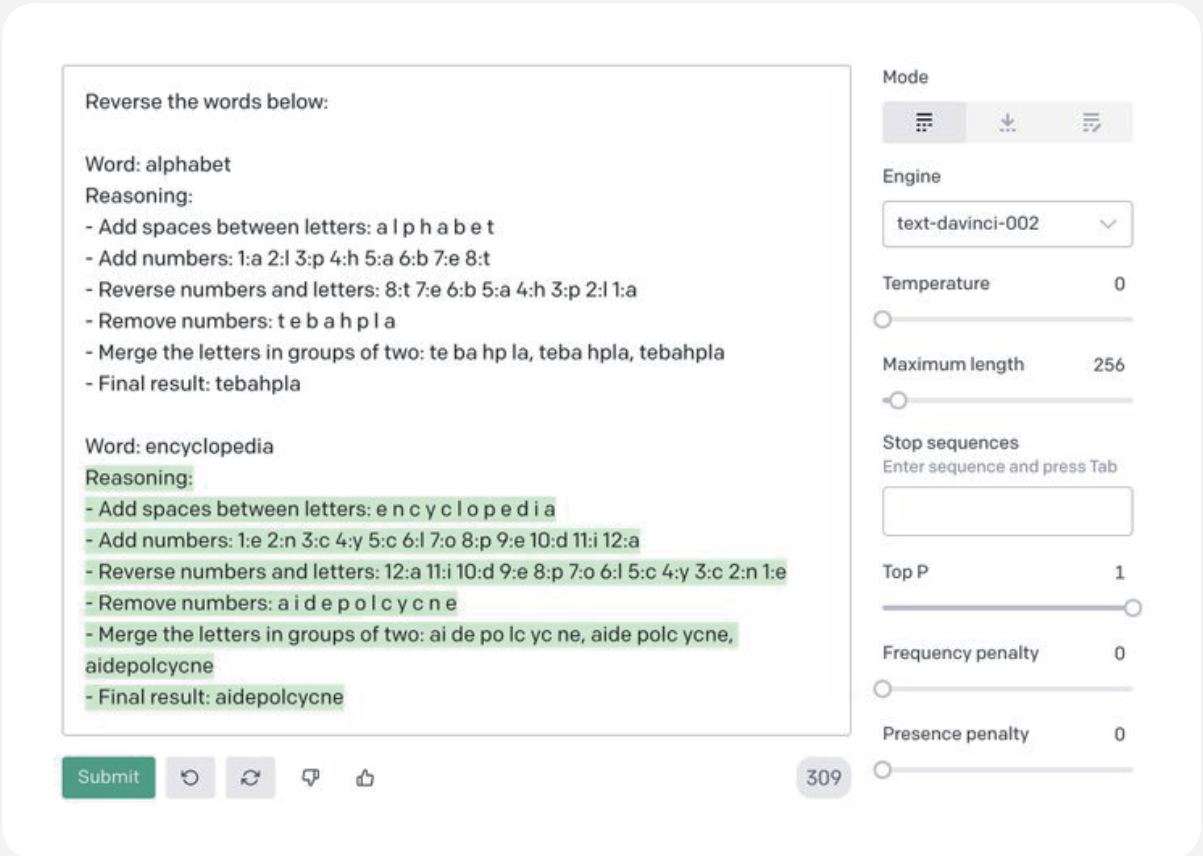
サブワードに基づく手法をもう少し詳しく見てみましょう。

サブワードに基づくトークン化手法	バイトペアエンコーディング(BPE)	WordPiece	Unigram	SentencePiece
説明	● 最も一般的なサブワードトークン化アルゴリズムの1つです。 Byte-Pair-Encodingは、文字から開始し、最も頻繁に見られる文字を結合して、新しいトークンを作成します。そして、コーパスの中で最も頻繁に見られるペアから新しいトークンを構築するために、繰り返し行います。 ● BPEは、複数のサブワード・トークンを使って見たことのない単語を作ることができるため、「unk(未知)」トークンを持つ可能性が低く、少ないボキャブラリーで対応できます。	BPEと非常に似ていますが、WordPieceは最も頻度の高いシンボルペアを選ぶのではなく、語彙に追加された後のトレーニングデータの尤度を最大化するものを選ぶことです(2つのシンボルを統合することで失うものを評価し、それに見合うかどうかを確認します)。	BPE/WordPieceとは異なり、Unigramはベースとなる語彙を多数の記号で初期化し、各記号を徐々に切り捨てていくことで語彙を少なくしていきます。 SentencePieceと一緒に使われることが多いです。	左の3つのトークナイザーは、入力テキストが単語を区切るためにスペースを使用していると仮定しているため、単語を区切るためにスペースを使用しない言語(例えば中国語)には適用できません。 SentencePieceは、入力を生の入力ストリームとして扱い、使用する文字のセットにスペースを含めます。そして、BPE/Unigramアルゴリズムを使用して、適切な語彙を構築します。
配慮事項	● BPEは、最も一般的な文字配列に基づいて新しい単語のサブワードを生成することができるため、希少語や語彙のない単語を扱うのに特に有効です。 ● デメリット:BPEは、言語的に意味のある単位に対応しないサブワードが生じる可能性があります。	WordPieceは、単語の意味が登場する文脈によって左右されるような言語では、特に有効です。	Unigramは、複雑な形態素を持つ言語に特に有効で、言語的に意味のある単位に対応するサブワードを生成することができます。しかし、Unigramは、希少な単語や語彙の少ない単語で苦労することがあります。	SentencePieceは、単語の意味が登場する文脈によって左右されるような言語では、特に有効です。
このトークン化手法を使用したトランスフォーマー	GPT-2、GPT3、ロベルタ	BERT、Distil BERT、Electra	Unigramは、どの変換モデルにも直接使用されません。その代わりに、SentencePieceと一緒に使用されます。	ALBERT、XLNet、Marian、T5

注：最近、生のバイトから学習するトークンフリーモデル（ByT5）も提案されています。これらのモデルの利点は、どの言語のテキストでもすぐに処理できること、OOV（out-of-vocabulary）単語やトークンを大量に含むコーパスに適しており、ノイズに強いこと、複雑でエラーが起こりやすいテキスト前処理フローを削除することで技術的負債を最小限に抑えることができることなどです。しかし、一般的にトークン化ベースのモデルよりも精度が低いという欠点があります。この方向性がどの程度有望であるかを判断するためには、この分野でのさらなる研究が必要です。

通常、トークン化の後には、パディングやトランケーション等のいくつかの追加ステップが追加されます。パディングとは、短い文に特別なパディングトークンを追加することで、テンソルが長方形になるようにする戦略で、すべての入力が均一なテンソル形状を持つようにします。一方、切り捨ては、モデルが扱うには長すぎる配列の長さを短くするものです。

注：サブワード・トークナイザー・ベースのLLMには、トークナイザー技術に直接関連する固有の制限があります。サブワード・トークナイザーのトークン粒度が単語と文字の間であるため、LLMは人間のように文字や単語を見ることができないのです。その代わりに、文字の塊である「トークン」を見るのです。この固有の制限に起因する例として、文字の結合が困難であることが挙げられます。



ピーター・ウェリンダー氏の例

事前学習のステップ

数十億パラメータのLLMをトレーニングするのは、通常、試行錯誤を繰り返す高度な実験プロセスです。通常は、小さなモデルサイズから始めて、それが有望であることを確認し、より多くのパラメータにスケールアップしていくことになります。規模が大きくなると、小さいデータでトレーニングしたときにはなかったような課題に対応する必要があることもあることにも留意が必要です。

一般的な事前学習ステップをまずはモデルアーキテクチャーの観点から見ていきましょう。

モデルアーキテクチャー

トレーニングが不安定になるリスクを減らすために、GPT-2やGPT-3のような人気のある先行モデルのモデルアーキテクチャーとハイパーパラメーターから始めて、トレーニング効率の向上、モデルサイズの拡大（深さと幅の両側面）、性能向上のための微調整をするアプローチが一般的です。2つの例を見ていきましょう：

GPT-NeoX-20B（20B，EleutherAI）はもともとGPT-3のアーキテクチャを受け継ぎ、このような変更を加えました：

- 性能と計算効率のバランスを考慮し、学習済みの位置エンコーディングではなく、埋め込みベクトル次元の最初の25%にロータリー埋め込みを使用。
- 主に計算効率を目的として、アテンション層とフィードフォワード層を直列に走らせるのではなく、並列に組み合わせる。
- GPT-3は密な層と疎な層を交互に使うが、GPT-NeoXは密な層のみを使い、実装の複雑さを軽減。

OPT-175B（175B，MetaAI）もGPT-3をベースに、下記の調整を行いました：

- 計算効率を高めるためのバッチサイズ。
- 学習率の計画的更新。具体的には、線形の学習率（LR）計画に従い、OPT-175Bでは最初の2000ステップ、小さいベースラインでは375Mトークンをかけて0から最大学習率まで温め、300Bトークンを超えて最大LRの10%まで減衰させる。また、学習中に何度もLRを変更する必要があった。
- OPT-175Bは、GPT-3と同じモデルサイズ（175B）であるにもかかわらず、GPT-3の300Bトークンに対して、180Bトークンという非常に小さなデータセットで学習した。

実験とハイパーパラメーター探索

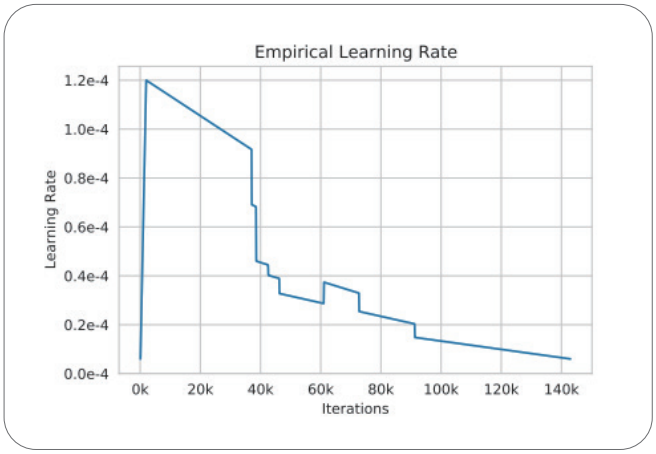
前述したように、一般的な事前学習では、モデルの性能に最適な設定を見つけるために多くの実験が行われます。実験には、重みの初期化、位置埋め込み、オプティマイザー、活性化、学習率、重み減衰、損失関数、シーケンス長、層数、アテンション層のヘッド数、パラメータ数、密層と疎層、バッチサイズ、ドロップアウトのいずれかまたはすべてを含まれます。

最適なパフォーマンスを達成するためのコンフィギュレーションを見つけるために、手動によるハイパーパラメーターの組み合わせの試行錯誤と自動ハイパーパラメーター最適化（HPO）の組み合わせが行われます。自動探索の対象となる代表的なハイパーパラメーターは、学習率、バッチサイズ、ドロップアウトなどです。

ハイパーパラメーター探索はコストのかかるプロセスであり、数十億のパラメータを持つモデルをフルスケールで処理するにはコストがかかりすぎることが多くあります。一般的には、ゼロからではなく、より小さなスケールでの実験の混合や、過去に発表された研究に基づくパラメータの補間によって、ハイパーパラメーターを選択することが多いです。

また、学習効率とトレーニングの収束性を両立させるために、学習の1エポックの間にも調整が必要なハイパーパラメーターがあります。いくつか例を見ていきましょう：

- 学習率：初期段階では直線的に増加し、学習終了に向けて減衰することがある。
- バッチサイズ：最初は小さなバッチサイズから始めて、徐々に大きなバッチサイズにすることも。



微調整したPaLM、OPTの例（出典：Open Pre-trained Transformer Language Models）

事前学習プロセスの初期に、このような実験を多く行うことをお勧めします。これは、扱うデータが少量になるため、より多くの実験を早い段階で行うことで、プロセス後半で行うよりもコストを削減できるからです。

LLMをトレーニングする場合、問題はつきものです。結局のところ、これは巨大なプロジェクトであり、あらゆる複雑なプロジェクト同様に、さまざまな課題の解決が求められます。代表的なもので見ていきましょう：

ハードウェアの不具合

トレーニングの過程では、計算クラスターにかなりの数のハードウェア障害が発生する可能性があり、その場合は手動または自動で再起動する必要があります。手動による再起動では、トレーニングの実行を一時停止し、一連の診断テストを実施して問題のあるノードを検出します。最後に保存したチェックポイントからトレーニングを再開する前に、問題が発見されたノードを封鎖する必要があります。

トレーニングの不安定さ

トレーニングの安定性も基本的な課題です。モデルのトレーニング中に、学習率や重みの初期化などのハイパーパラメーターがモデルの安定性に直接影響することが発覚することがあります。例えば、損失関数が収束せずに発散した場合、学習率を下げ、以前のチェックポイントから再開することで、ジョブが回復し、トレーニングを継続することができるかもしれません。

さらに、モデルが大きくなればなるほど、トレーニング中の損失スパイクを避けることが難しくなります。このスパイクは、非常に不規則な間隔で発生する傾向にあり、時にはトレーニングの後半に発生することもあります。

スパイクを軽減するための戦略については、これまで体系的な分析はあまり行われていません。ここでは、モデルを効果的に収束させるための、ベストプラクティスを紹介します：

- バッチサイズ：GPUが使用できる最大のバッチサイズを使用することが、基本的には最良の方針です。
- バッチの正規化：ミニバッチ内の活性度を正規化することで、収束を早め、モデル性能を向上させることができます。
- 学習率（LR）の計画的更新：学習率が高いと、損失が振動したり発散したりして、損失が急増することがあります。学習率が時間とともに低下するようにスケジューリングすることで、モデルのパラメータに対する更新の大きさを徐々に減らし、安定性を向上させることができます。

一般的な計画更新には、一定のステップ数の後に学習率を一定量ずつ減少させるステップ減衰や、学習率をステップごとに一定係数ずつ減少させる指数関数的減衰があります。どのLRを使うべきかを前もって知ることは実際には不可能ですが、異なるLR更新方法を使ってモデルがどのように反応するかを確認することはできます。

- 重みの初期化：重みを適切に初期化することで、モデルの収束を早め、性能を向上させることができます。例えば、小さなガウスノイズや、トランスフォーマーのケースではT-Fixupの初期化を使用するのが一般的です。重みの初期化に使用できる技術には、ランダムな初期化、レイヤー単位の初期化、事前学習された重みを使用した初期化などがあります。
- モデルトレーニングの出発点：関連するタスクでトレーニングされた事前学習済みモデルを出発点として使用することで、モデルの収束を早め、性能を向上させることができます。
- 正則化：ドロップアウト、荷重減衰、L1/L2正則化などの正則化技術は、過学習を減らし、汎化性能を改善することによって、モデルの収束を助けることができます。
- データオーグメンテンション：トレーニングデータを増やすことで、モデルの汎化を促進し、過学習を減らすことができます。
- トレーニング中のホットスワップ：オプティマイザーや活性化関数のホットスワップは、LLMのトレーニング中に現れた問題を処理するために使われることがあります。そのため、ほぼ24時間365日、トレーニングに携わるチームが様々なヒューリスティックを試しながら、さらなるトレーニングを行う必要があることもあります。
- その他、不安定な問題に遭遇した場合の簡単な対処法：スパイクが発生したデータバッチをスキップする（直感的には、スパイクは特定のデータバッチと特定のモデルパラメータ状態の組み合わせで発生すると考えられます）。

注：上記のモデル収束のためのベストプラクティスのほとんどは、トランスフォーマーのトレーニングに適用されるだけでなく、アーキテクチャやユースケースを超えた広範なディープラーニングの文脈でも有効です。

最後に、LLMトレーニングが終了したら、モデルトレーニング環境を保存し、その最終状態を保持することが非常に重要です。そうすれば、将来、何かをやり直したり、再現したりする必要があっても、トレーニングの状態が保存されているため、それが可能になります。

また、アブレーション研究も一考に値します。これは、「モデルのこの部分を抜いたら、性能にどのような影響が出るか」を確認するものです。アブレーション研究により、モデルの予測力をほとんど維持したまま、モデルのサイズを大幅に縮小できる場合があります。

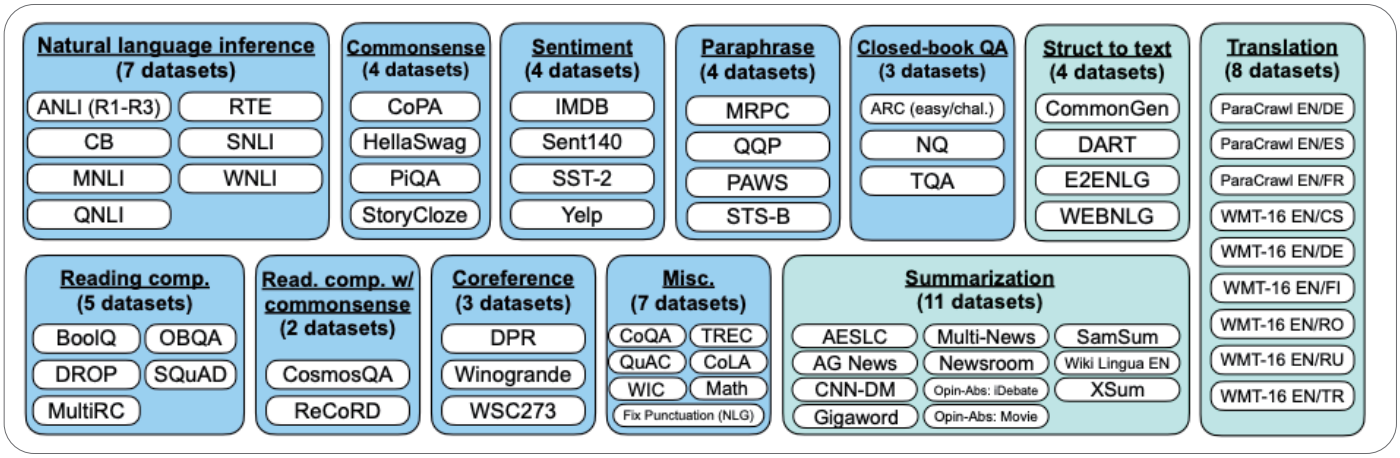
モデル評価

通常、学習済みモデルは、論理的推論、翻訳、自然言語推論、質問応答などを実行する能力を評価するために、多様な言語モデルデータセットで評価されます。

これまで、機械学習開発者は、様々な標準的な評価ベンチマークを確立してきました。代表的な例としては、以下のようなものがあります:

- オープンドメインな質問応答タスク:TriviaQA、Natural Questions、Web Questions
- 穴埋めタスクと補完タスク:LAMBADA、HellaSwag、StoryCloze
- ウィノグラード式タスク:Winograd、WinoGrande
- 常識の推論:PIQA、ARC、OpenBookQA
- コンテキストの読解:DROP、CoQA、QuAC、SQuADv2、RACE、SuperGLUE
- 自然言語推論(NLI):SNLI、QNLI
- 推論タスク: 算術的な推論タスク
- コードタスク:HumanEval、MBPP(テキストからコードへ)、TransCoder(コードからコードへ)
- 翻訳タスク:WMT言語ペアの翻訳におけるBLEUスコア
- BIG-bench:大規模言語モデルに対する挑戦的なタスクを作成することを目的とした共同ベンチマークで、テキストタスクとプログラムタスクを含む200以上のタスクが含まれています。
- LM評価ハーネス:EleutherAIがリリースした、200以上のタスクにわたる自己回帰型LLMを標準的に評価するためのライブラリです。体系的なフレームワークのアプローチと堅牢性から人気を博しています。

代表的な言語タスクののまとめ(青字がNLUタスク、ティール字がNLGタスク):



NLPのデータセットとタスクカテゴリー(出典:Finetuned Language Models are Zero-Shot Learners)

もう一つの評価ステップはNショット学習です。これはタスクに依存しない手法で、与えられたタスクの実行をモデルに求める直前に提供する教師付きサンプル(デモ)の数を意味します。Nショット学習は通常、プロンプティングと呼ばれる手法で行われます。Nショットは、以下の3つのカテゴリーに分類されます:

- Zero-shot(ゼロショット):推論時にモデルに教師付きサンプルを提供することなく、任意のタスクで評価すること。
- One-shot(ワンショット):few-shot に似ているが n=1 で、推論時にモデルに1つの教師付きサンプルが提供される。
- Few-shot(少数ショット):推論時にモデルに与える教師サンプルが複数だが少ない場合(例:5サンプル提供 → 5 shot)。

Few-shotの学習例：

- 課題：感情分析
- プロンプト ：

  ツイート ："スマホの電池が切れるのは嫌だ"

  感情：否定的

  ツイート ："私の一日は 👍 "

  感情：肯定的

  ツイート ："これが記事へのリンクです"

  感情：中立

  ツイート ："この新しいミュージックビデオは素晴らしかった"

  感情：
- 答え：_____

評価には、上記のタスクのベンチマーク・メトリクスを見る方法と、モデルにプロンプトを与え、その結果を人間が見て評価する、手動の評価の両方が含まれるのが普通です。通常、NLPエンジニアと専門家の両方が評価プロセスに参加し、モデルのパフォーマンスをさまざまな角度から検証します：

- NLPエンジニア：NLP、計算言語学、プロンプトエンジニアリングなどの専門性を持つ人たちで、モデルの意味的・構文的な欠点を調査・評価し、継続的な改善のためにモデルの失敗クラスを考え出すことができる。例えば、「LLMは整数（1、2、3など）や、そのスペルアウト形（one、two、three）を使った算術を扱えない」というような失敗クラスがあります。
- 専門家（SME）：NLPエンジニアとは対照的に、SMEはLLM出力の特定のクラスを調査し、必要に応じてエラーを修正し、そのプロセスを「声に出して表現する」ことが求められます。つまり、正しい答えと機械が生成した間違った答えの理由と論理を、段階的に説明することが要求されます。

バイアスと有害性

ウェブ上のテキストでトレーニングされた大規模な汎用言語モデルには、潜在的なリスクがあります。人間にはバイアスがあり、そのバイアスがデータに入り込み、そのデータから学習するモデルがそのバイアスを受け継いでしまう可能性があるためです。社会的なステレオタイプを永続させたり悪化させたりすることに加え、LLMが個人情報を記憶・漏洩したりしないようにする必要があります。そのような潜在的に望ましくない関連性やリスクを、透明性の高いアーティファクトで分析し、文書化することが必要です。

性能ベンチマークと同様に、LLMモデルの潜在的な有害性を評価するために、コミュニティが開発したバイアスと有害性ベンチマークのセットが存在します。代表的なベンチマークは以下の通りです：

- ヘイトスピーチの検出：ETHOSデータセットは、LLMモデルが特定の英語の文が人種差別や性差別であるかどうかを識別する能力を測定するのに役立ちます。
- 社会的バイアスの検出：CrowSPairsは、性別、宗教、人種/色、性的指向、年齢、国籍、障害、身体的外観、社会経済的地位の9つのカテゴリーにおける文内レベルのバイアスを測定することを目的としたベンチマークです。StereoSetは、職業、性別、宗教、人種の4カテゴリーにおけるステレオタイプ・バイアスを測定するベンチマークです。
- 有害な言葉による返答：RealToxicityPromptsデータセットは、モデルが有害な言語を使用しているかどうか、またどのように使用するかを評価するのに役立ちます。
- ダイアログの安全性の評価：SaferDialoguesベンチマークは、モデルの反応がどの程度有害かを、安全、現実的、安全でない、敵対的の4つのレベルで層別して測定します。

これまで、既存の学習済みモデルに関する分析のほとんどは、インターネット上のデータで学習したモデルにはインターネット特有のバイアスがあることを示しています。また、学習済みモデルは、比較的無害なプロンプトが提供された場合でも、有害な言葉を生成する傾向が高く、敵対的なプロンプトも散見されます。

バイアスと有害性の修正

では、この問題をどのように修正すればよいのでしょうか。ここでは、事前学習プロセス中や学習後のバイアスを軽減する方法をいくつか紹介します：

- トレーニングセットのフィルタリング：トレーニングデータセットのうち、バイアスが見られる要素を分析し、トレーニングデータから単純に削除します。
- トレーニングセットの修正：トレーニングデータをフィルタリングするのではなく、バイアスを軽減するために変更する手法です。例えば、特定の性別の言葉（policemanをpolicewomanやpolice　officerに変更するなど）を変更することで、バイアスを軽減することができます。

さらに、学習後のモデルでもバイアスを軽減することは可能です：

- プロンプトエンジニアリング：各クエリのモデルへの入力を、モデルをバイアスから遠ざけるために修正します（詳細は後述）。
- ファインチューニング：　トレーニングしたモデルを再トレーニングし、バイアスのかかった傾向を解消する。
- 出力ステアリング：推論手順にフィルタリングのステップを追加して、出力値を重みづけし、バイアスのかかった回答から出力を遠ざける。

インストラクション・チューニング

この時点で、学習済みの汎用的なLLMがあるとします。うまくいけばこのモデルは、few-shot学習やzero-shot学習のシナリオをチューニングすることなく、ドメイン固有のタスクに使用することができます。しかし、読解、質問応答、自然言語推論などの多くのタスクにおいて、zero-shot学習は、一般にfew-shot学習よりもはるかに悪い結果を出します。その理由の1つは、例題がないと、モデルが事前学習データの形式とは似ていないプロンプトに対して良い結果を出せなくなってしまうことです。

この問題を解決するためには、インストラクションチューニングが有効です。

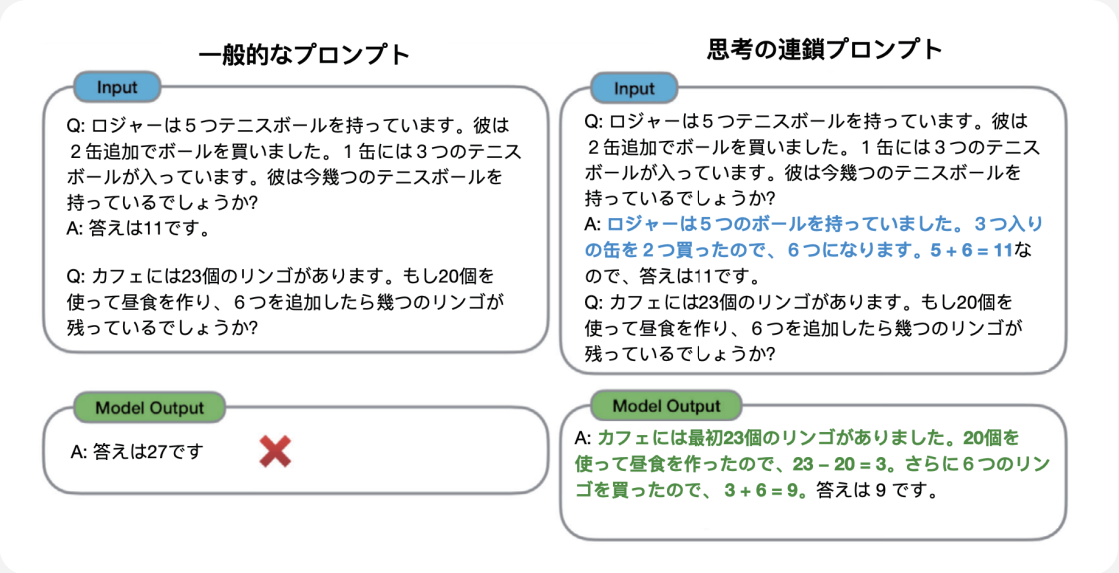
インストラクションチューニングは、学習済みのLLMをインストラクションとして表現されたタスクの集合に対して微調整する、最先端のファインチューニング手法です。学習済みLLMが指示に対してより適切に反応するようになり、プロンプトの段階で例文を必要としなくなります（つまり、zero-shot性能が飛躍的に向上します）。

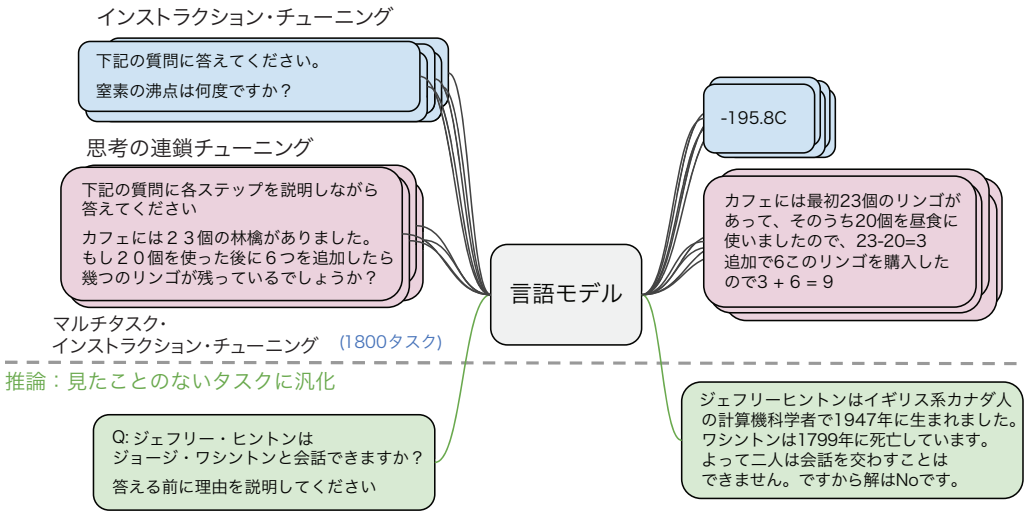
インストラクションチューニングは、汎化能力を損なうことなくモデルの性能を大幅に向上させる技術であるため、2022年に大きな人気を博しました。通常、学習済みLLMは言語タスクのセットでチューニングされ、チューニング時に未知の言語タスクのセットを実行する能力で評価され、その汎化能力とzero-shot能力が証明されます（下図を参照）。

インストラクションチューニングを事前学習＋ファインチューニングとプロンプティングで比較
（出典：Finetuned Language Models are Zero-Shot Learners）

インストラクション・チューニングを実施する上での注意点：

- インストラクションチューニングは、モデル全体のパラメーターをチューニングするもので、パラメーター効率の良いファインチューニングではパラメーターの一部を固定するのに対し、インストラクションチューニングでは、モデル全体のパラメーターをチューニングします。そのため、パラメータ効率の良い微調整のようなコストメリットはありません。しかし、パラメータ効率の良い微調整に比べ、インストラクションチューニングはより汎用性の高いモデルを生成するため、インストラクションチューニングしたモデルは、複数の下流タスクに対応する汎用モデルとして機能することができます。そのためには、インストラクションチューニングを行うためのデータセットとトレーニングの予算があるかどうかが重要です。
- インストラクションチューニングは、指示として自然に言語化されるタスク（例：NLI、QA、翻訳）には普遍的に有効ですが、推論のようなタスクには少し厄介です。このようなタスクで改善するためには、チューニングの際に思考の連鎖の例を含めるとよいでしょう。





模範解答の有無（ゼロショットと少数ショット）、思考の連鎖の有無の両方でチューニングすることで、さまざまな評価シナリオでの般化を可能にした（出典：Scaling Instruction-Finetuned Language Models）

人間のフィードバックによる強化学習（RLHF）

RLHFはインストラクションチューニングの延長線上にあり、インストラクションチューニングのステップの後に、さらに人間のフィードバックを取り入れるステップが追加されています。

学習済みLLMの項で述べたように、学習済みLLMはしばしば、事実をでっち上げたり、バイアスや有害な言葉を生成したり、単にユーザーの指示に従わなかったりといった意図しない振る舞いを示します。これは、最近の多くの大規模言語モデルの学習指標、すなわちインターネット上のウェブページ上のテキスト次のトークン（言葉）を予測するという評価指標が、"ユーザーの指示に安全に従う"という目的とは大きく異なっているためです。

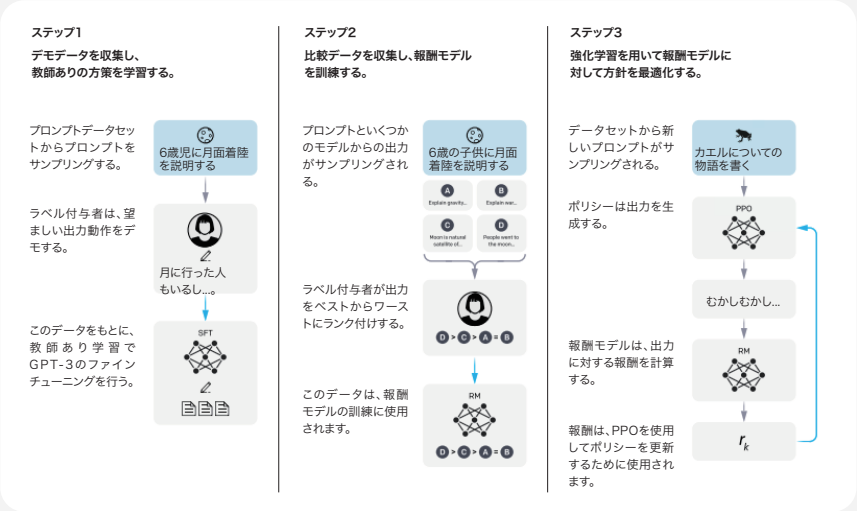
RLHFは、その名前が示すとおり、特定のプロンプトを与えられたモデルの出力に対して、人間のフィードバックを取り込みます。そして、その出力の質に関する意見を、モデル全体のパフォーマンスを向上させるための追加データとして利用します。

OpenAIは最近、InstructGPTで成功を収めています。これは基本的に学習済みモデルGPT-3をRLHFで微調整したものです。ChatGPTモデルは、より高度なGPTモデルシリーズ（GPT-3.5と呼ばれる）に対してRLHFを適用しています。

RLHFは一般的に次のような仕組みになっています：

- ステップ1：インストラクションチューニング - 望ましいモデルの振る舞いを示すラベラーのデータセットを収集し、それを使って、教師あり学習を使用して学習済みLLMを微調整する。
- ステップ2：与えられた入力に対してラベラーがどちらの出力を好むかを示す、モデルの出力間の比較のデータセットを収集。次に、人間が好む出力を予測する報酬モデルをトレーニングする。
- ステップ3：トレーニングされた報酬モデルを用いて、強化学習を通じて報酬モデルに対するポリシーを最適化する。

ステップ2と3は連続的に繰り返すことができます。より多くの比較データを収集し、現在の最良の方策で新しい報酬モデルのトレーニングに使用され、その後ポリシーが更新されます。RLHFのプロセス例は以下をご覧ください。



本手法の3つのステップを示す図：(1)教師ありファインチューニング(SFT)、(2)報酬モデル(RM)学習、(3)前述の報酬モデルに対する近接方策最適化(PPO)による強化学習。
(出典：Training language models to follow instructions with human feedback)

現在までに、RLHFはInstructGPTとChatGPTにおいて、学習済みGPTと比較して性能低下を最小限に抑えながら、真実性の向上と有害な出力生成の低減を実現し、非常に有望な結果を示しています。

RLHFは、「アラインメント税」と呼ばれるように、いくつかの下流タスクにおいてモデルの性能を若干低下させるという代償を伴うことが知られています。Scale AI、Labelbox、Surge、Label Studioなどの企業は、RLHFをサービスとして提供しているので、この手法の導入を検討している場合は、全てを自社で実施する必要はありません。RLHFの技術を使用することで、より望ましい出力をさせるためのコストを最小化することができるという有望な研究結果が示されており、絶対に検討する価値があります。

結 論

OpenAI、CohereからEleutherAIのようなオープンソースプロジェクトまで、最先端の大規模言語モデルはWeights & Biasesで構築されています。私たちのプラットフォームは、モデルのトレーニングやプロダクションへの移行に必要な複雑でコストのかかる作業を行うチーム間のコラボレーションを可能にし、主要な評価指標のログ、データセットのバージョン管理、知識の共有、ハイパーパラメーターの探索、その他多くのことを可能にします。LLMトレーニングは複雑で繊細なものです。モデルのライフサイクルを通じて、共通の情報源を持つことは、よくある間違い避け、すべてのステップでベストなモデルパフォーマンスを実現するために不可欠です。

OpenAI、Deepmind、Meta、GoogleBrainにも心からの感謝を表します。このホワイトペーパーでは、これら先端企業の研究やブレークスルーを数多く参照しました。

W&Bが御社のAI開発にどのようにお役に立てるか、ご興味のある方は、ぜひご連絡ください。また、本ホワイトペーパーに関連してご意見がありましたら、ぜひお聞かせください。

参考文献

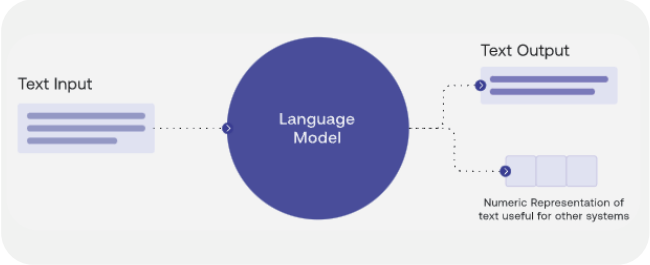
- What Language Model Architecture and pre-training Objective Work Best for Zero-Shot Generalization?
- GPT 3 Paper - Language Models are Few-Shot Learners
- GPT-NeoX-20B: An Open-Source Autoregressive Language Model
- OPT: Open Pre-trained Transformer Language Models
- PaLM: Scaling Language Modeling with Pathways
- Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM
- Scalable Deep Learning on Distributed Infrastructures: Challenges, Techniques and Tools
- New Scaling Laws for Large Language Models by DeepMind
- New Scaling Laws for Large Language Models
- Understanding the Difficulty of Training Transformers
- How To Build an Efficient NLP Model
- Emergent Abilities of Large Language Models
- Beyond the Imitation Game benchmark (BIG-bench)
- Talking About Large Language Models
- Galactica: A Large Language Model for Science
- State of AI Report 2022
- Finetuned Language Models are Zero-Shot Learners
- Scaling Instruction-Fine Tuned Language Models
- Training language models to follow instructions with human feedback

付 録

LLMの概要

学習済み大型言語モデル（Large Language Models：LLM）は、自然言語処理（NLP）の能力を飛躍的に向上させた、機械学習におけるブレークスルーです。

数千億ものパラメータで構成され、数百テラバイトのテキストデータでトレーニングされたトランスフォーマーアーキテクチャーに基づき、GPT-3（OpenAI、2020）、GPT-NeoX（EleutherAI、2022）、PaLM（GoogleBrain、2022）、OPT（MetaAI、2022）、Macaw（Allen Institute）などの最近のLLMは、幅広いNLPタスクにおいて著しい精度改善を実現しています。

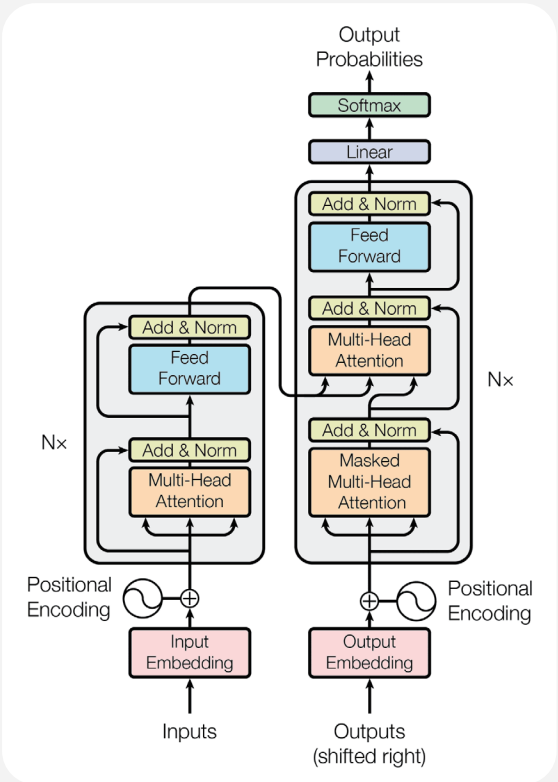


大規模言語モデルは、ソフトウェアシステムにおけるテキスト理解と生成の新たな可能性を開くコンピュータプログラム
（出典：CohereAI Large Language Models）

トランスフォーマー・モデル・アーキテクチャー

現代のLLMは、トランスフォーマーアーキテクチャをベースにしています。トランスフォーマーの主要なアーキテクチャ単位はトランスフォーマーブロックであり、ヘッドを複数もつセルフアテンション層、層の正規化、密な2層のフィードフォワードネットワーク、残差接続で構成されています。トランスフォーマースタックは、このようなブロックを並べたものです。

下図は、エンコーダー・デコーダー構造を持つ典型的なトランスフォーマーのアーキテクチャを表しています：



トランスフォーマー・モデル・アーキテクチャ
（出典：Attention Is All You Need）

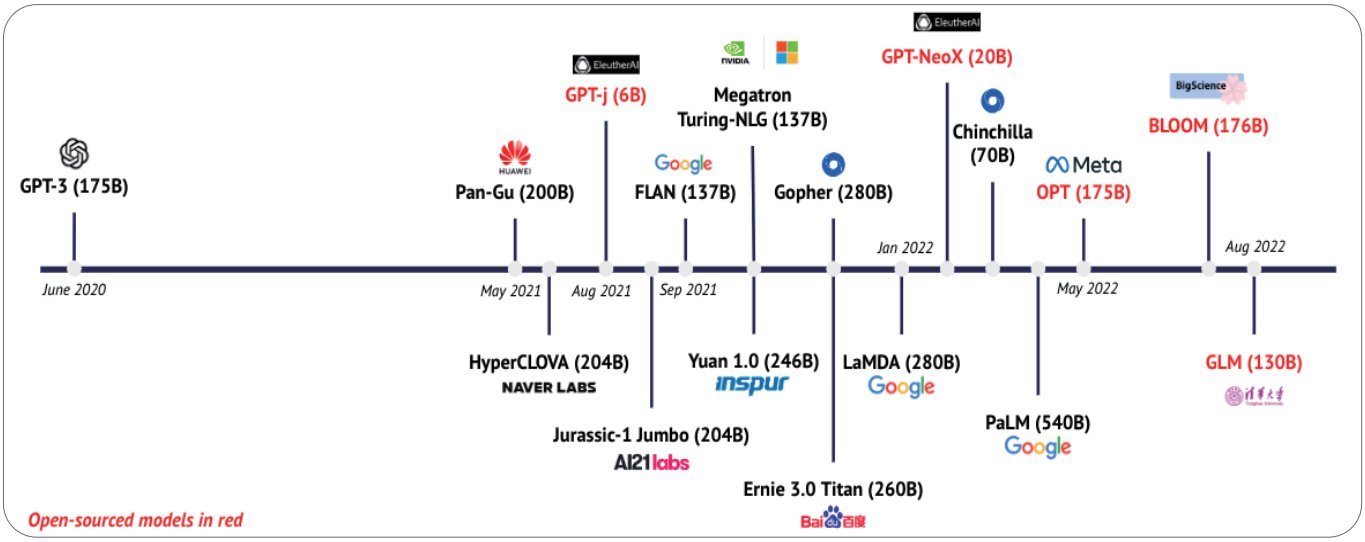
トランスフォーマーの登場以来、様々なアーキテクチャのバリエーションが提案されています。これにはアーキテクチャの違い（デコーダのみのモデル、エンコーダ・デコーダモデルなど）、学習目的の違い（完全言語モデリング、接頭辞言語モデリング、マスク付き言語モデリングなど）、その他の違いがあります。

初代のトランスフォーマーでは、入力テキストを処理するエンコーダとターゲットテキストを生成するデコーダが別々に存在していた（エンコーダー・デコーダーモデル）のに対し、GPT-3、OPT、PaLM、GPT-NeoXなどの最も一般的なLLMは、テキスト列を自己回帰的に予測するようにトレーニングした因果的デコーダのみからなるモデルです。この傾向とは対照的に、エンコーダー・デコーダーモデルは、転移学習（学習済みモデルを下流の単一タスクで微調整すること）においてデコーダのみのLLMを上回ることを示す研究も存在します。詳細なアーキテクチャと比較については、What Language Model Architecture and pre-training Objective Work Best for Zero-Shot Generalization をご覧ください。

最も一般的な学習済みアーキテクチャをいくつか紹介します：

- エンコーダー・デコーダーモデル：当初、トランスフォーマーはエンコーダーとデコーダーの2つのスタックから構成されていました。エンコーダーは、入力されたトークンの列を入力し、入力と同じ長さのベクトル列を出力します。そして、デコーダは、エンコーダの出力内容を条件として、ターゲットとなるシーケンスをトークンごとに自己回帰的に予測します。このタイプの代表的なモデルには下記が含まれます：T5、BART。
- 因果的デコーダオンリーモデル：テキスト列を自己回帰的に予測するようにトレーニングされたデコーダのみのモデルです。"因果的"とは、モデルが左の文脈（次のステップの予測）だけに依存することを意味します。このタイプの代表例として、GPT-3、GPT-J、GPT-NeoX、OPTなどがあります。
- 非因果的デコーダオンリーモデル：デコーダオンリーモデルが入力テキストのより豊かな非因果的表現を構築できるように、条件付け情報に対応する入力シーケンスの領域が非因果的マスクを持つように（すなわち、過去のトークンに制限されない）、アテンションマスクが変更されています。代表的なPLMモデルには以下のものがあります：UniLM 1-2、ERNIE-M。
- マスク付き言語モデル：通常はエンコーダのみのモデルで、マスク付き言語モデルの目的関数で事前学習されていて、周囲の文脈に基づいてマスクされたテキストを予測します。代表的なMLMモデルには以下のものがあります：BERT、ERNIE。

下図は、近年話題になった事前学習済みLLMを示したものです：



コミュニティ主導によるGPTのオープンソース化（出典：State of AI Report 2022）

ここでは特に因果的デコーダオンリーモデルを紹介します。

LLMを下流タスクに使用する最新の技術には、一般的に次の2つのフェーズがあります：(1)LLMの事前学習を行い、汎用モデルを作成します。(2)微調整やプロンプティングなどの手法を用いて汎用モデルを特定の下流タスクに適合させます。この記事では、プロセスのステップ1であるLLMの事前学習について見ていきましょう。

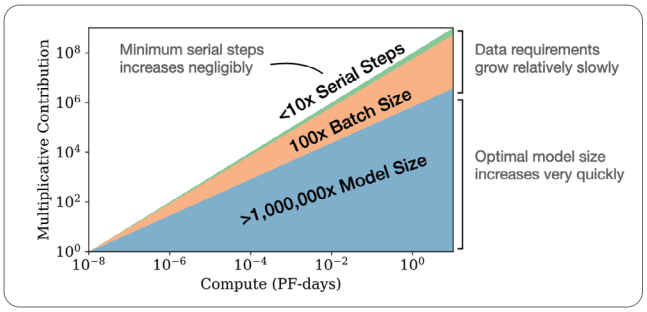
初期のLLMのスケーリング法則

LLMのスケーリング法則（OpenAIが最初に紹介したもの）は、"ある量の計算機がある場合、最高のパフォーマンスを得るためには、どの程度の大きさのモデルをトレーニングすればよいのか？"という疑問に答えようとするものです。

その答えは、基本的にモデルサイズとデータサイズのトレードオフにあります。例えば、GPT-3スケールのモデルの場合、トレードオフは次のような点です：

- (a)インターネットのアーカイブの40%で200億のパラメータモデルをトレーニングする。
- (b)インターネットのアーカイブの4%を対象に、2,000億個のパラメータモデルをトレーニングする。

2020年、OpenAIは Scaling Laws for Neural Language Models を発表しました。この論文では、計算量を最適化するトレーニングのためには、データサイズを増やすよりもモデルサイズを増やすことが重要であるとしています。計算量が10倍になったら、モデルサイズを5倍くらいにして、データサイズを2倍にするのが望ましいとしています。さらに10倍の計算量が得られれば、モデルサイズは25倍、データサイズは4倍に相当します。



モデルサイズとデータ量のバランスを前提に、より多くの計算量を割り当てる（出典:Scaling Laws for Natural Language Models）

多くの研究者がこの考え方に基づいて、より多くのデータで比較的小さなモデルをトレーニングするよりも、いかにしてより大きなモデルを設計するかに注力してきました。下図は、2020年から2022年にかけて発表されたモデルにおいてこの傾向が見られることを示しています：

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion

現在のLLMとそのサイズ（出典:Training Compute-Optimal Large Language Models）