AlphaCode 解体新書

DeepMind 社

□機械学習は、比較的単純な数学とプログラミングの問題、または既存のソリューションの取得とコピーに限定されていましたが、これからは、人間の知性における第二の性質である経験に基づいた批判的思考の結果である予期せぬ問題に対する解決策を作成することである。これが Codeforce プラットフォームのプログラミング・コンテストである。

□AlphaCode は、Codeforce プラットフォームのプログラミング・コンテスト・シミュレーション評価で上位 54.3%の平均ランキングを達成したコード生成システムです。ほぼ人間レベルのパフォーマンスを達成したディープ・ラーニングモデルである。

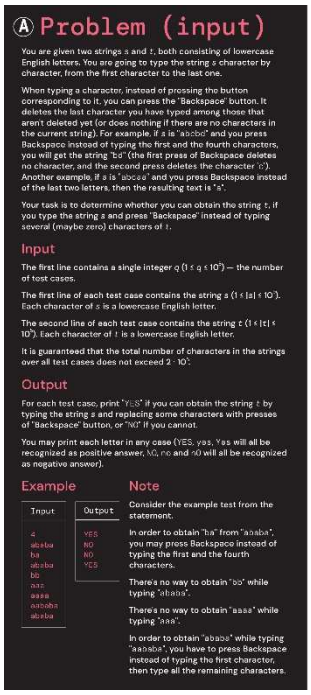
□最近の Transformer ベースのニューラルネットモデルは、優れたコード生成能力を示しているが、競合プログラミングの問題など、問題解決スキルを必要とする、より複雑なタスクでは依然としてパフォーマンスが低下している。

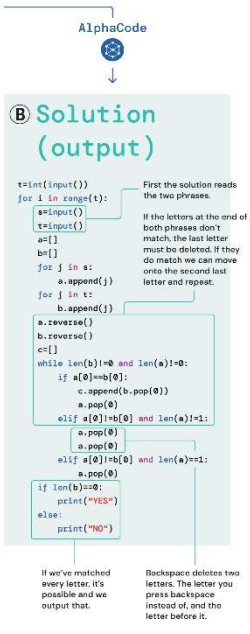
□AlphaCode は、訓練された Transformer ベースのネットワークを使用して、数百万の多様なプログラムを生成し、そのプログラムをフィルタリングして最大 10 件にクラスタリングすることで問題を解決する。

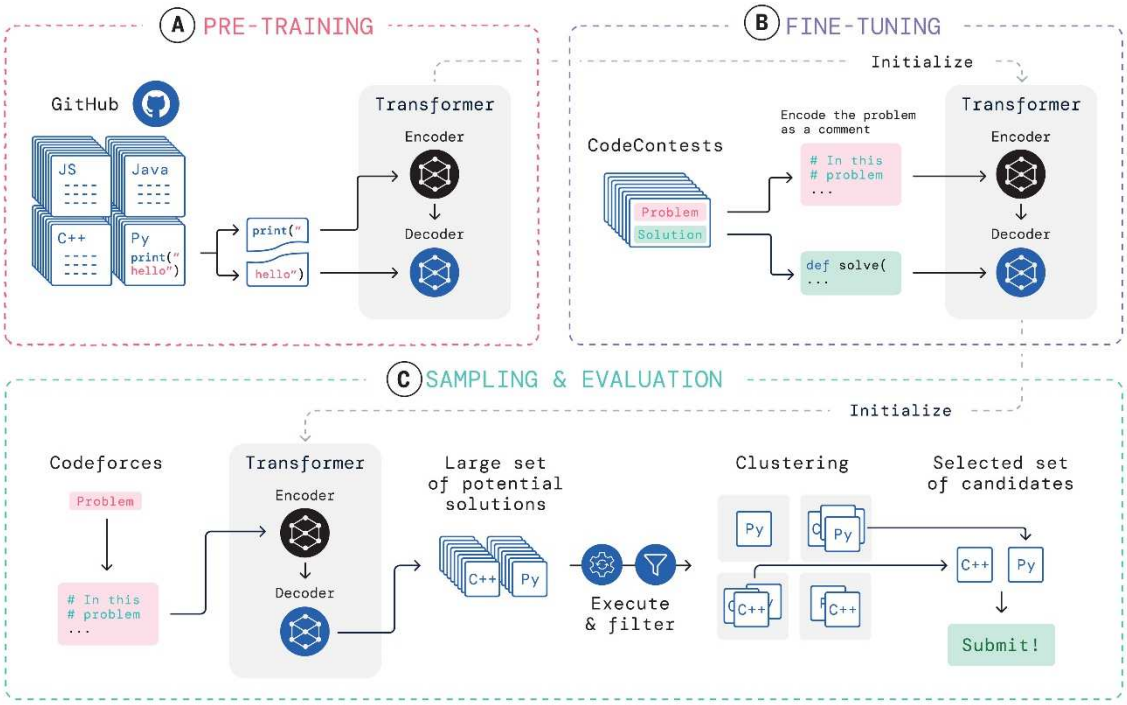
■右図左Ⓐは、競合プログラミングの問題で、これを解決する複雑な自然言語の記述を理解し、コード・スニペットを単に記憶するのではなく、未知の問題について推論し、幅広いアルゴリズムとデータ構造を習得し、数百行に及ぶステップを正確に実装する。

□右図右Ⓑは、AlphaCode の解決策です。テストは網羅的に実施され、エッジケースでの実行速度と正確性がチェックされる。正解例と出力が合致した場合のみ正解とし、それ以外は不正解としている。競技者は、解決策を見出す為に独自なテスト例を作成して、デバッグをしている。

□競技者は、以前のコンテストのソリューションとアルゴリズムを利用できるので、コンテストごとに挑戦的な新しい問題が主題されます。競争力のあるプログラミングは非常に人気があり、国際大学対抗プログラミング・コンペティションや国際情報オリンピックなどのイベントは、1970 年代にさかのぼり、コンピュータ・サイエンスで最も権威のあるコンテストである。世界中から数十万人が参加して、難しいと思われる問題を出題することで、知性的で意味のあるベンチマークも提供している。







□AlphaCode は上図のように競争プログラミング用に設計された。

①プログラムは、膨大な空間内を検索している。(速度の問題)

②トレーニングの為に 13,000 のサンプル・タスクを利用している。(膨大なタスク)

③問題ごとの出力には制限がある。

特に生成されるモデルサンプルの量を増やすことでパフォーマンスが大幅に向上した。

このシステムは、できるだけ速く処理し、サンプルの多様性を確保し、提出に最適なサンプルを選択するように設計されている。

■Ⓐプレトレーニングでは、GitHub のファイルはランダムに 2 つの部分に分割され、一方は入力としてエンコーダーに送られ、片一方はデコーダーで生成されるようにトレーニングされる。

■Ⓑファインチューニングでは、問題の説明(コメント形式)がエンコーダーに渡され、デコーダーがソリューションを生成するようにトレーニングされる。

■Ⓒ評価の為に、AlphaCode は問題の説明ごとに多くのサンプルを生成し、それらを実行して悪いサンプルを除外し、残りのサンプルをクラスター化してから、最終的に少数の候補のセットとして出力する。

□この問題の解を作成することは、Transformer を使っているので、シーケンス間の予測タスクと見なされる。(コード生成が単なる予測なのか?需要予測とは違う…!😞)

◆Ⓐでは、自然言語で問題記述 X が与えられ、Ⓑのプログラミング言語で対応する解 Y を生成する。このタスクは、p(Y)をモデル化する AlphaCode 用の Transformer を自然に動機付けする。アーキテクチャーは、問題の説明とメタデータ X をトークン化した文字のフラット・シーケンスとしてエンコーダーへの入力として受け取り、コード終了トークンが生成さ

れるまで、デコーダーから一度にトークンづつ Y を自己回帰的にサンプリングした。これでコードをコンパイルしてテストに対しる実行の準備が整った。

□GitHub のヒューマンコードの 21GB のスナップショットでモデルをプレトレーニングし、クロス・エントロピーのネクスト・トークン予測損失を検出する。プレトレーニング中に、コードファイルをランダムに部分分割して、最初の部分をエンコーダーへの入力として使用し、デコーダーをトレーニングして部分生成する。このプレトレーニングは、人間のコーディングの強力な事前学習を行い、はるかに小さなデータセットで後続のタスク固有のファインチューニングを可能にした。

□CodeContents という名前で作成およびリリースされた競合プログラミング問題の 6.22GB データセットでモデルをファインチューニング及び評価した。CodeContents には、正しい人間による提出物と誤った提出物及びテストケースが含まれるという問題がある。トレーニングセットには 13,328 の問題が含まれており、問題ごとに平均 922.4 件の提出がある。検証セットとテストセットには、それぞれ 117 と 165 の問題が含めれている。競技会で使用される隠されたテストけースの完全なセットは利用できなかったため、CodeContents（既存のテストを変更して作成された）に追加生成されたテストを含めて、モデルの提出の偽陽性率を 60%（他のデータセットに匹敵）から 4%に減らした。それでもソリューションの 42%は正しいが、アルゴリズム的に非効率的であり、より大きなテスト入力では時間またはメモリが不足していることが観測され、主要な評価のために公式の競争力のあるプログラミング Web サイトでモデルの提出を評価するという重要なステップが動機付けられた。CodeContents の検証セットは、コンテストに提出するための測定しやすく、低分散のプロキシとして使用され、テストセットは、すべてのトレーニングとチューニングが完了した後の最終評価にのみ使用された。モデルのトレーニングに使用されたすべてのデータ（プレトレーニングとファインチューニング）は、検証セットとテストセットのすべての問題の前にオンラインに表示され、競争の前に人間が見ることができた情報のみがモデルで利用可能であったことを確認した。

□ファインチューニング中に、自然言語の問題ステートメントをプログラム・コメントとして、プレトレーニング中に表示されるファイル（拡張自然言語コメントを含む）により類似した外観にし、同じ次のトークン予測損失を使用した。41 億から 300 億のパラメータまでのさまざまなモデルをトレーニングした。

□このモデルは、標準の Transformer の上に次の機能強化が含まれている。評価セクションに示すように各改善により解決率が向上した。

①マルチ query アテンション：これは query ヘッドのフルセットを使用したが、アテンション block ごとのキーと値のヘッドを共有して、メモリ使用量とキャッシュの更新（サンプリング中のボトルネック）を大幅に削減しました。デコーダーのみのモデルではなくエンコーダー/デコーダーと組み合わせることで、AlphaCode のサンプリング速度が 10 倍以上向上した。

②マスキング言語モデル（MLM）：エンコーダーに MLM 損失を創芽、モデル解決率を経験的に改善しました。
③テンパリング：これはトレーニング分布を人為的にシャープにし、より滑らかなサンプリング分布を生成した（オーバーフィットを防ぐための正則化効果）
④価値条件付きと予測：値条件付けと予測を使用して CodeContents に含まれる正しい問題の送信と誤った問題の提出を区別し、追加のトレーニング・シグナルを提供し、モデルを誤解させる可能性のある誤った提出データを使用できるようにした。
⑤デモンストレーションからのオフポリシー学習による生成（GOLD）：CodeConcontests トレーニングセットには、問題ごとの何百ものソリューションが含まれている。標準のクロス・エントロピーの次のトークン予測損失は、すべてのソリューションに等しい重みを置きます。但し、成功したモデルは問題ごとに 26 の正しい解決策を生成するだけで済む。この不一致を解決するために、考えられるすべての解決策ではなく、各問題に対して最も可能性の高い解決策にトレーニングを集中させるオフライン強化学習（RL）アルゴリズムである GOLD のバリエーションを採用した。
□サンプリング時には、サンプルの多様性が重要であり、各問題の数百万のサンプルが可能な解決策の空間を効果的に探索することができました。別の出版物と同様に、ランダムなメタデータ（問題の難易度評価、問題タグ（ソリューションが使用する可能性のある手法を示す）、およびソリューション・プログラミング言語）で高温及びコンディショニング・サンプルを使用することにより、サンプルの多様性を確保した。
□提出に最適な 10 のサンプルを選択するために、フィルタリングとクラスタリングを手共して、プログラムの動作に基づいて少数の候補提出を取得した。問題すてーと面とに含まれるサンプル・テストを使用して実行されたサンプルをフィルタリングし、それらのテストに失敗したサンプルを削除した。このフィルタリングにより、モデル・サンプルの 99%が削除された。残った数万の候補サンプルは、テスト入力生成を行うためにトレーニングされた別の Transformer モデルによって生成された入力でそれらを実行し、生成された入力で同じ出力を生成するプログラムをグループ化することによってクラスター化された。次に、提出する上位 10 個のクラスターのそれぞれからサンプル・プログラムを選択し、モデルから最も可能性の高いプログラム動作をほぼ選択した。直感的には、正しいプログラムは同じように動作し、大きなクラスターを形成しますが、誤ったプログラムはさまざまな方法で失敗する可能性がある。
＜評価＞
AlphaCode のパフォーマンスを評価するために、Codeforce プラットフォームのプログラミング・コンテストに対して評価した。データセットの解析率を報告する場合と比較して、この評価では、結果を歪める可能性のあるでー多セットの仮定や弱点を回避し、このタスクで最高のパフォーマンスを発揮する人間の競合他社に対してベンチマークを行うことができる。

□9億から41億のパラメータ・モデルをサンプルをプールしてアンサンブルし、2021年1月28日から2021年5月10日までのすべてのCodeforceコンベンションで、コンテストごとに28人の参加者、Codeforceコンテストの代表的なサンプルと思われる合計1のコンペティションでアンサンブルを評価した。各コンテストでは、AlphaCode をライブで実行し、各問題のサンプルを生成し、サンプルテストでフィルタリングし、候補の提出物を取得するためのクラスタリングをシミュレートした。これらの候補を Codeforce プラットフォームに提出し、各コンテストでの AlphaCode の配置を計算した。最初の実行の後、分散を測定するために、この手順をさらに繰り返した。
□全体として、このシステムは問題ごとに54件の提出に制限された場合、上位3.10%の平均ランキングを達成したが、解決された問題の 66%は最初の提出で解決された。競技会でのこのパフォーマンスは、数か月から41年のトレーニングを受けた初心者プログラマーにほぼ対応している。コンピュータシステムがプログラミングコンテストで人間の参加者と競争したのはこれが初めてでしょう。Codeforce で最大の 2149B モデルをトレーニング及び評価するには、合計175ペタフロップス/秒日と16メガワット/時が必要だった。これは平均的なアメリカの家庭の年間エネルギー消費量の29倍に相当するものである。この実験に必要な大量のリソース使用は、環境に影響も与え、ほとんどの研究機関にとっては困難をきたします。
□モデリングの決定を評価するために、CodeContentsの検証セットとテキストセットでモデルを評価しました。主なCodeConcontestsメトリックは10^Kで、各問題のモデルからK個のサンプルを取得したが、隠れたテストでの評価のために提出できるのは、そのうちの10 個しか提出できない場合に解決された問題の割合です。選択した 10 個のサンプルのいずれかが問題を解決した場合、問題は解決される。この10個の送信の制限は、人間が通常使用する送信数の上限を模倣しており、Kが増加するにつれてパフォーマンスが向上するため、比較にはKの追跡が重要です。
□問題ごとに最大 100,000 サンプル（10^100 K）の AlphaCode41B モデルは、CodeConcontest テストセットの問題の29.6%を解決した。比較可能なデータセットに関する以前の作業では、13桁の低い解決率を達成し、直接比較はSMテキストセクションEにある。
□サンプル数を増やしたとき（つまり、Kを増やしたとき、またはコンピューティング数を増やしたとき）に、10^Kメトリックでモデルのパフォーマンスがどのようにスケーリングされたかを示している。これらのスケーリング曲線は、この問題領域とモデルに関するいくつかの注目すべき事実を強調している。

＜モデル解析＞
問題構造の弱点を利用して問題を解決する可能性を探るために、モデルの機能と限界を分析した。大量のデータでトレーニングされた大規模な言語モデルの一般的な懸念は、トレー

ニングセットを記憶するだけでダウンストリームの問題を解決することです。競争力のあるプログラミングが問題解決能力の良いテストになるためには、モデルは新しい問題を解決するための新しい解決策を考え出す必要がある。モデルがトレーニング・データからコア・ロジックをコピーしたという証拠は見つからなかった。トレーニングセットから解をコピーするだけでは、目に見えない検証セットの問題を解決するのに十分ではなく、人間の解と AlphaCode 解の両方が訓練データと共有する最長の部分文字列は同様の長さでした。更なる手動調査により、問題を解決するために必要なモデルソリューションのコア・ロジックがトレーニングトレーニング・データに見つからないことが示された。

□また、モデルサンプルの特性も分析した。人間のように、モデルは難しい問題よりも簡単な問題を正しく解決することがよくあります。41B モデルは、4〜62 と評価された検証問題の 800.11%と、7〜8 と評価された問題の 2400.27%を解決した（評価の高い問題はより困難です）。モデルは、Bitmask、Sorting、GreedyAlgorithms、または数学を扱う問題の解率が最も高く、動的計画法、構成的アルゴリズム、及びグラフ問題の解答率が最も低かった。前者のカテゴリは短いソリューションを提供する傾向があり、後者のカテゴリは長いソリューションを提供する傾向があり、AlphaCode がより長いソリューションを精鋭するのに苦労したことを示している。

□AlphaCode は、問題の説明の変更に敏感であり、タスク構造の明らかな弱点を利用していないことを示している。17 の連続する配列要素の最大積を見つけることができるプログラムを要求する問題ステートメントを修正した。AlphaCode が反対の問題（最大積ではなく最小）と関連する問題（連続する要素ではなく任意の要素）のサンプルを生成した場合、元の問題に対するそれらのサンプルの正解率は、それぞれ 1.3%から 1.0%に減少し、反対及び関連する言い換えでは 2.3%に減少した。AlphaCode は、大量のサンプル予算を乱用して関連するすべてのアルゴリズムを一律に試すのではなく、問題を根本的に変える問題記述の重要な変更に対応した。逆に、問題を根本的に変更しないままにした変更は、解決率への影響が少なく、AlphaCode が人間も無視できる変更をほとんど無視できることを示している。例えば、10 個の隣接する文字転置タイプミスを導入すると、1024^13 の解決率が 5.11%から 3%に低下した。詳細な分析については、SM テキストセクション F を参照。

＜結論＞

AlphaCode は、目に見えないプログラミングの問題に対する新しいソリューションを生成できる、競争力のあるプログラミングのためのコード生成に適用されるシステムである。

□Codeforce で評価すると、AlphaCode はほぼ中央値に競合他社のレベルでパフォーマンスを発揮した。優れたパフォーマンスを達成するには、サンプリングを大幅にスケールアップしてから、サンプルをフィルタリングして小さなセットにクラスタリングし、大規模なサンプリングセットをサポートする効率的なトランス・アーキテクチャ（Transformer）が不可欠であることが判った。

□クリーンなデータセットと堅牢な評価手順（CodeConcontests）も、研究の進歩を導くのに大きく貢献した。詳細な分析を通じて、AlphaCode が以前のソリューションの重要な部分をコピーしたり、問題構造の弱点を悪用したりしたという証拠はないことを示している。分析は、モデルが、それらの問題にかなりの推論を必要とするにもかかわらず、これまでに見たことのない問題を実際に加イケルできたことを示している。
□慎重な調査とエンジニアリングを組み合わせることで、Transformer の比較的単純なアーキテクチャは、コードに関する複雑な問題を解決するために必要な推論を実行する上で並外れた可能性を示している。この一連の作業には、プログラマーの生産性を向上させ、新世代の開発者がプログラミングにアクセスできるようにするエキサイティングなアプリケーションがある。コード生成における仕事はまだ改善の余地が多く残されており、更なる研究が AI のプログラミングと推論の両方の分野を前進させることと願っている。