論文要約

LLM関連

Evaluating the External and Parametric Knowledge Fusion of Large Language Models 大規模言語モデルの外部およびパラメトリック知識融合の評価 2024

概要

LLMは外部知識と内部知識を使い、情報を統合する能力を調査。記憶と活用に課題があるため、実験でその融合方法を評価。
LLMsはトレーニング時に多くの知識を取得しますが、それが時とともに陳腐化する問題があります。外部知識を追加してこれを補う方法がありますが、外部知識に過度に依存する傾向があり、モデル自身のパラメトリック知識の価値を過小評価することがあります。

手法

LLMsの外部知識（Ke）とパラメトリック知識（Kp）の融合を4つのシナリオに分けて調査しました：

1. シナリオ1（S1）：Keのみで質問に回答可能な場合
2. シナリオ2（S2）：Keが部分的な情報を提供しKpが補完する必要がある場合
3. シナリオ3（S3）：Keが有用な情報を提供せずKpのみに依存する場合
4. シナリオ4（S4）：KeもKpも質問に対して十分な情報を提供しない場合

これらのシナリオに対して、最新のデータと過去のデータを収集し、データを外部知識として使用する部分と、パラメトリック知識としてLLMsに注入する部分に分けました。そして、これらのデータを基にQAペアを生成し、LLMsの知識融合の能力を評価するための実験を実施しました。

結果

実験の結果、以下のことが明らかになりました：

1. 知識注入の効果
   - LLMsにパラメトリック知識を注入することで、特に外部知識が不完全な場合のパフォーマンスが向上しました。
   - ただし、モデルが注入されたすべての知識を正確に保持できるわけではなく、知識の呼び出しに課題が残りました。
2. シナリオごとのパフォーマンス
   - S1（外部知識のみで回答可能な場合）では、外部知識の品質が高ければ高いパフォーマンスを発揮しました。
   - S2（部分的な外部知識とパラメトリック知識の融合が必要な場合）では、パラメトリック知識の有効利用が鍵となりました。
   - S3（外部知識が無用でパラメトリック知識のみが必要な場合）では、注入されたパラメトリック知識の品質と量がパフォーマンスに影響しました。
   - S4（回答不能な場合）では、モデルが正しく「回答不能」と判断する能力が問われましたが、多くのモデルは外部知識に過度に依存し、誤った回答を生成する傾向が見られました。

Language Generation with Strictly Proper Scoring Rules 厳密に適正なスコアリングルールを用いた言語生成　2024

概要

新しいスコアリングルールを使い、 LLMの性能を向上させる手法を提案する。対数スコアの代わりにブライアスコアと球面スコアを使用し、より良い生成結果を得る。
言語生成における最尤推定（ MLE）をベースにした手法がテキスト生成の基本的なアプローチになっています。　MLEは通常、ログ尤度損失（統計的決定理論では対数スコアとして知られる）を最小化することによって実行されます。対数スコアは予測の正直性を促進し、観測されたサンプルの確率にのみ依存するため、自然言語の大規模なサンプル空間を処理する能力があります。
非ローカルなスコアリングルールを言語生成に適応させるための戦略を提案し、対数スコアの代替としてブライアスコアと球面スコアを使用して言語生成モデルをトレーニングします。実験結果は、損失関数を置き換えるだけでモデルの生成能力が大幅に向上することを示しています。
https://github.com/shaochenze/ScoringRulesLM

手法

1. スコアリングルールの適用:
    a. 対数スコア、ブライアスコア、球面スコアを用いた損失関数の設計
    b. スコアリングルールをトークンレベルに分配し、条件付き確率の精度を向上させる戦略
    c. 任意のスコアリングルールに対するスコアスムージング技術の導入
2. 実験:
    a. 様々なデータセット（WMT14英仏、WMT14英独、CNN/DailyMail）での性能評価
    b. 大規模言語モデル（LLaMA-7B、LLaMA-13B）への適用と評価
    c. 翻訳タスクと要約タスクでの性能比較

結果

- 対数スコアに代えてブライアスコアや球面スコアを用いることで、モデルの生成能力が向上することを確認。
- 特に大規模言語モデルにおいて、スコアリングルールの変更による性能向上が顕著。
- スコアスムージング技術により、正規化効果が強化されることを確認。

数式の説明

対数スコア、ブライアスコア、球面スコアの定義が以下のように説明されています。

1. 対数スコア:$S(p,i)=\log p_i$
   $S(p,i)=\log p_i$
2. ブライアスコア:$S(p,i)=1-j=1\Sigma m(\delta ij-pj)2=2pi-j=1\Sigma m^{**}pj2$

   $S(p,i)=1-\sum j=1m(\delta ij-pj)2=2pi-\sum j=1mpj2$
3. 球面スコア:$S(p,i)=|p|pi$

   $S(p,i)=pi|p|$

これらのスコアリングルールを用いることで、モデルが真の確率分布に従った予測を行うようになります。

Toxicity Detection for Free
Freeで行う毒性検出 2024

概要

MULIはLLMで毒性のあるプロンプトを検出する方法を使い、追加コストなしで高い精度を達成する。安全な応答のためにロジットを分析する。
一般的に安全性の要件に従うように調整され、毒性のあるプロンプトを拒否する傾向がありますが拒否しない場合もあります。また、過剰に検出し無害な例に対して拒否してしまうことがあります。
LLM自体から直接抽出した情報を使用して毒性のあるプロンプトを検出する「LLM内省を用いたモデレーション（MULI）」を使用し、特定の開始トークンのロジットに基づく簡単なモデルは、訓練や追加の計算コストを必要とせずに信頼性の高いパフォーマンスを発揮します。また、スパースロジスティック回帰モデルを使用して、より堅牢な検出器を構築しています

手法

- トイモデルの開発：開始トークンのロジットを利用することで、毒性のあるプロンプトと無害なプロンプトの間に有意なギャップが存在することを発見しました。
- スパースロジスティック回帰モデルLLMの最初の応答トークンのロジットを使用して、毒性を検出するためのモデルを構築しました。

結果

MULIは、複数の指標で最先端の検出器を大幅に上回る性能を示し、特に低い誤検知率での高い真陽性率を達成

SPECTRA: Enhancing the Code Translation Ability of Language Models by Generating Multi-Modal Specifications
SPECTRA：マルチモーダル仕様の生成による言語モデルのコード翻訳能力の向上　2024

概要

SPECTRAは高品質な仕様を生成する多段階アプローチを使いLLMのコード翻訳能力を向上させる。仕様を用いて翻訳精度を高める

手法

SPECTRAは次の2つの主要なステップで構成されています：

1. プログラムから高品質な不変条件、テストケース、自然言語記述を生成し、それらがプログラムと一貫性があるかどうかを検証します。
2. 検証された仕様をプログラムのソースコードと共に使用し、 LLMに翻訳候補を生成させます。

結果

SPECTRAはCからRustおよびCからGoへの翻訳タスクにおいて、 4つの人気のある LLMのパフォーマンスを最大 23%相対的に、10%絶対的に向上させました。これは、高品質な仕様を生成することで、 LLMの翻訳性能を効率的に向上させる可能性があることを示唆しています。

不変条件（Invariants）の生成
Here is a C program:
You are an expert Rust developer. Translate this program to Rust such that the Rust code passes the given test case.

テストケース（Test Cases）の生成

```
main(n){
    n=read(0,buf,114514);
    n--;
    puts(n+(buf[0]==buf[n-1])&1?"First":"Second");
}
```

Input:
abcde

Output:
Second

Here is a test case for the C program:

SPECTRA: Enhancing the Code Translation Ability of Language Models by Generating Multi-Modal Specifications
SPECTRA：マルチモーダル仕様の生成による言語モデルのコード翻訳能力の向上 2024

テストケース（Test Cases）の生成

Here is a C program:

You are an expert Rust developer. Translate this program to Rust such that the Rust code passes the given test case.

```
char buf[114514];
main(n){
    n=read(0,buf,114514);
    n--;
    puts(n+(buf[0]==buf[n-1])&1?"First":"Second");
}
```

Input:
abcde

Output:
Second

Here is a test case for the C program:

自然言語記述（Natural Language Descriptions）の生成
Here is a C program:

You are an expert Rust developer. Translate this program to Rust. You can take the help of the function descriptions provided as comments.

```
char buf[114514];
// Main function that reads input from standard input, determines the length of the input, and prints "First" or "Second" based on the specified condition.

main(n){
    n=read(0,buf,114514);
    n--;
    puts(n+(buf[0]==buf[n-1])&1?"First":"Second");
```

PromptWizard: Task-Aware Agent-driven Prompt Optimization Framework PromptWizard: タスク対応型エージェント駆動プロンプト最適化フレームワーク2024

概要

PromptWizardはLLMでプロンプトを自動生成し、最適化する。タスクに合わせてプロンプトと例を調整するため、精度が向上する。少数のデータでも効果的に動作する。
PromptWizardは特定のタスクに合わせたプロンプトを自動的に生成し、最適化する新しいフレームワーク。プロンプトの指示とインコンテキスト例の両方を最適化することで、モデルのパフォーマンスを最大化します。評価は9タスクで行われ、既存の手法（MedPrompt、PromptBreederなど）よりも良くなったりする

手法

PromptWizardは、タスクに特化したプロンプトを生成し、最適化するフレームワークです。LLMを活用してプロンプトの指示とインコンテキスト例を効率的に最適化します。このフレームワークは、二つの主要フェーズから構成されています
前処理フェーズと推論フェーズ。
2. 前処理フェーズ
2.1 プロンプト指示の反復的最適化

- Mutate Agent（変異エージェント）
  - 多様な思考スタイルを使ってプロンプト指示を生成します。例えば、問題を簡略化する視点や異なるアプローチを考える視点などを使用します。
  - 例：「この数学の問題をステップバイステップで解決する方法を考えよう。」
- Scoring Agent（スコアリングエージェント）
  - 生成されたプロンプト指示をトレーニングデータの一部に適用し、その効果を評価します。最も高いスコアのプロンプトが次のステップに進みます。
- Critic Agent（批評エージェント）
  - 選ばれたプロンプトに対してフィードバックを提供し、改善点を指摘します。
- Synthesize Agent（合成エージェント）
  - 批評エージェントのフィードバックを基にプロンプト指示を改良します。

これらのステップを複数回繰り返し、タスクに最適なプロンプト指示を生成します。

2.2 多様な例の選択

- Negative Examples（ネガティブ例の特定）
  - トレーニングデータからプロンプトが失敗した例を選び出します。これにより、プロンプトの多様性と性能が向上します。

2.3 プロンプト指示と例の順次最適化

- Critic AgentとSynthesize Agentの反復使用
  - 批評エージェントのフィードバックを基に、より多様でタスクに適した新しい合成例を生成します。
  - 新しい合成例を使ってプロンプト指示をさらに洗練します。

2.4 自動生成された推論と検証

Faithful Logical Reasoning via Symbolic Chain-of-Thought
シンボリックチェーンオブソートによる忠実な論理的推論 2024

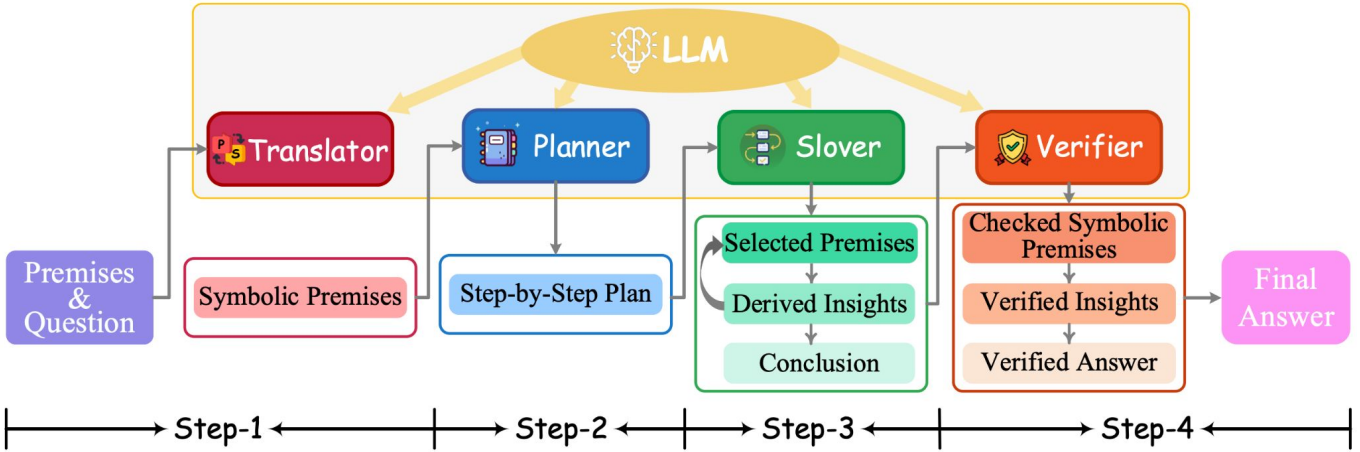
概要

SymbCoTはシンボリックな表現とルールを使い、LLMで論理的推論を強化する。問題を解決するためステップバイステップで計画し、検証する。
https://github.com/Aiden0526/SymbCoT

手法

SymbCoTの技術的な構成要素は以下の通りです:

1. 翻訳: 自然言語の文脈をシンボリック形式に翻訳する。
2. 計画: シンボリック論理ルールを用いて問題を解決するためのステップバイステップの計画を立てる。
3. 解決: 計画に従って問題を解決する。
4. 検証: 翻訳と推論のチェーンをチェックし、正確性を確認する。



Faithful Logical Reasoning via Symbolic Chain-of-Thought
シンボリックチェーンオブソートによる忠実な論理的推論 2024

1. 翻訳: 自然言語の文脈をシンボリック形式に翻訳する。

Task Description:
You are given a problem description and a question. The task is to:
1. Define all the predicates in the problem.
2. Parse the problem into logic rules based on the defined predicates.
3. Write all the facts mentioned in the problem.
4. Parse the question into the logic form.

Problem:
- If a cartoon character is yellow, it is from the Simpsons.
- If a cartoon character is from Simpsons, then it is loved by children.
(... More premises ...)

Question:
- Ben is ugly or yellow.

Output:
- Define predicates: Yellow(x), Simpsons(x), LovedByChildren(x), Ugly(x).
- Logic rules:
  1. ∀x (Yellow(x) → Simpsons(x))
  2. ∀x (Simpsons(x) → LovedByChildren(x))
  (... More rules ...)

- Facts:
  - (Yellow(ben) ∨ Ugly(ben))

Faithful Logical Reasoning via Symbolic Chain-of-Thought
シンボリックチェーンオブソートによる忠実な論理的推論 2024

2. 計画: シンボリック論理ルールを用いて問題を解決するためのステップバイステップの計画を立てる。

Task Description:
Please derive a step-by-step plan using the First-Order Logic rule for determining the conclusion based on the context.

Input:
<Premises>
- If a cartoon character is yellow, it is from the Simpsons :: ∀x (Yellow(x) → Simpsons(x))
- If a cartoon character is from Simpsons, then it is loved by children :: ∀x (Simpsons(x) → LovedByChildren(x))
(... More premises ...)
<Statement>
- Ben is ugly or yellow :: (Yellow(ben) ∨ Ugly(ben))

Output:
<Plan>
1: Identify the relevant premise of Ben.
2: Identify the relevant premise of yellow and ugly.
(... More steps ...)

Faithful Logical Reasoning via Symbolic Chain-of-Thought
シンボリックチェーンオブソートによる忠実な論理的推論 2024

3. 解決: 計画に従って問題を解決する。

Task Description:
Please solve the question based on First-Order Logic rules such as Modus Ponens...

Input:
<Premises>
- If a cartoon character is yellow, it is from the Simpsons :: ∀x (Yellow(x) → Simpsons(x))
- If a cartoon character is from Simpsons, then it is loved by children :: ∀x (Simpsons(x) → LovedByChildren(x))
(... More premises ...)
<Statement>
- Ben is ugly or yellow :: (Yellow(ben) ∨ Ugly(ben))
<Plan>
1: Identify the relevant premise of Ben.
2: Identify the relevant premise of yellow and ugly.
(... More steps ...)

Output:
<Reasoning Steps>
Step 1: Yellow(ben) → Simpsons(ben) by Modus Ponens from premise 1.
Step 2: Simpsons(ben) → LovedByChildren(ben) by Modus Ponens from premise 2.
(... More steps ...)
<Conclusion>
Thus, we can conclude that (Yellow(ben) ∨ Ugly(ben)) is false by contradiction.

Faithful Logical Reasoning via Symbolic Chain-of-Thought
シンボリックチェーンオブソートによる忠実な論理的推論 2024

4. 検証: 翻訳と推論のチェーンをチェックし、正確性を確認する。

Task Description:

Please verify: 1) the symbolic context is consistent with the natural language; 2) the solving step is logically valid.

Input:
<Premises>
- If a cartoon character is yellow, it is from the Simpsons :: ∀x (Yellow(x) → Simpsons(x))
- If a cartoon character is from Simpsons, then it is loved by children :: ∀x (Simpsons(x) → LovedByChildren(x))
(... More premises ...)
<Statement>
- Ben is ugly or yellow :: (Yellow(ben) ∨ Ugly(ben))
<Reasoning Steps>
Step 1: Yellow(ben) → Simpsons(ben) by Modus Ponens from premise 1.
Step 2: Simpsons(ben) → LovedByChildren(ben) by Modus Ponens from premise 2.
(... More steps ...)
<Conclusion>
Thus, we can conclude that (Yellow(ben) ∨ Ugly(ben)) is false by contradiction.

Output:
<Translation Verification>
1. Verify context consistency: "If a cartoon character is yellow, it is from the Simpsons" is semantically consistent with "∀x (Yellow(x) → Simpsons(x))"
(... More verification ...)
<Logic Verification>
2. Verify solving logic step by step: Step 1 correctly implements Modus Ponens using premise 1; Step 2 correctly implements Modus Ponens using premise 2;
(... More logic verification ...)
<Conclusion Verification>
Thus, the solving process is logically valid. The answer is verified to be false.

A Human-Like Reasoning Framework for Multi-Phases Planning Task with Large Language Models
大規模言語モデルによる多段階計画タスクのための人間らしい推論フレームワーク　2024

概要

LLMを人間のように計画するフレームワークを使い、旅行計画を改善する。アウトラインを生成し、情報を収集し、最適な計画を作成する。

手法

旅行計画という多段階計画問題に焦点を当て、LLMエージェントが人間のような方法で計画を立てられるフレームワークを提案します。具体的な手法は以下の通りです。

1. アウトライン生成フェーズ
    - PathFinderエージェントが旅行全体の大まかなルートを生成し、都市間の移動情報を含めたガイドラインを作成。
    - Transportation Evaluationにより、生成されたルートの合理性を評価し、必要に応じてフィードバックを提供してルートを修正。
    - Keypointsエージェントが旅行計画における重要なポイントを特定し、Commonsenseエージェントが基本的なガイドラインを生成。
2. 情報収集フェーズ
    - Thoughtエージェントが、Strategy Blockに基づいて次のステップを生成し、Toolエージェントが適切な関数表現を生成。
    - Descriptionエージェントが得られた情報をKnowledge Blockに記録し、詳細な計画作成のためにPlanエージェントに送信。
3. 計画作成フェーズ
    - Planエージェントが日毎に計画を作成し、Evaluateエージェントが各計画を評価して最良の計画を選定。
    - 複数の計画を生成し、エラーのある計画を破棄して最良の計画を採用。

1.

結果

提案したフレームワークがGPT-4-Turboと組み合わせた場合、従来のベースラインと比較して10倍のパフォーマンス向上を達成したことを示しています。特に、以下の改善点が確認されました。

- デリバリーレート: 提案フレームワークを用いることで、全モデルのデリバリーレートが向上しました。
- 常識制約パスレート: 常識的な制約を満たす計画の生成能力が向上。
- ハード制約パスレート: 明示的なハード制約を満たす計画の生成能力が向上。

Retrieval Augmented Structured Generation: Business Document Information Extraction As Tool Use
検索強化構造生成：ツールとしてのビジネス文書情報抽出 2024

概要

BDIEは非構造化文書を構造化データに変換するLLMを使い、キー情報とラインアイテムを抽出する。性能向上のためにRASGを使用しています
ビジネス文書情報抽出（BDIE）は、生のテキストやスキャンされた文書などの非構造化情報を、下流システムが解析および利用できる構造化形式に変換する問題です。これには主にキー情報抽出（KIE）とラインアイテム認識（LIR）の2つのタスクがあります。
BDIEを下流システムをツールとして使用するツール使用問題としてモデル化し、フレームワークであるRetrieval Augmented Structured Generation（RASG）を紹介

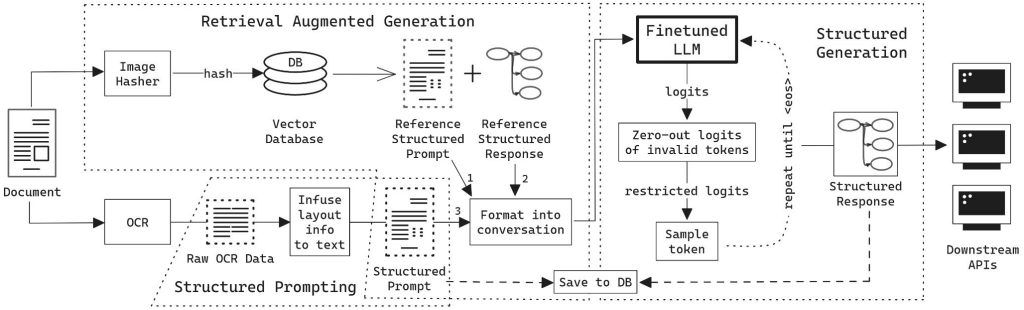
手法

Retrieval Augmented Structured Generation（RASG）は、BDIEを強化するために以下の四つのコンポーネントで構成されています。

1. **検索強化生成（Retrieval Augmented Generation）**:
   - インコンテキストラーニングを活用して、新しいデータやタスクに適応できるようにします。
2. **監督付きファインチューニング（Supervised Finetuning）**:
   - モデルが正確な情報を抽出できるよう、事前に用意されたデータセットを使用して、抽出精度を向上させます。
3. **構造生成（Structured Generation）**:
   - モデルの出力を特定のフォーマットに強制することで、下流のシステムがその出力を解析しやすくなります。例えば、抽出された情報が適切なキーと値のペアとして整理されるようにします。
4. **構造プロンプティング（Structured Prompting）**:
   - モデルに対して入力プロンプトのレイアウト情報を提供する技術です。これにより、モデルは文書の構造を理解しやすくなり、より正確な情報抽出が可能になります。

結果

正直普通



PostDoc: Generating Poster from a Long Multimodal Document Using Deep Submodular Optimization
PostDoc: 深層サブモジュラ最適化を用いた長いマルチモーダルドキュメントからのポスター生成 2024

概要

PostDocは深層サブモジュラ関数を使い、長い文書を要約してポスターを自動生成するLLMで内容をパラフレーズし、テンプレートを生成する。
文書からマルチモーダルコンテンツを抽出し、良好なカバレッジ、多様性、テキストと画像の整合性を保証する新しい深層サブモジュラ関数を提案

手法

a. **マルチモーダル要約** 文書の内容を効率的に要約するために、深層サブモジュラ関数を使用。この関数は、カバレッジ、多様性、マルチモーダル整合性を考慮しており、データから学習する。
b. **内容のパラフレーズ** 要約された内容をポスターに適した形式にパラフレーズするためにGPT-3.5-turbo（ChatGPT）を使用。
c. **テンプレート生成** 要約された内容に基づいて適切なデザイン要素を持つポスターテンプレートを生成。これには、フォント選択、色選択、レイアウト生成が含まれる。

結果

- **ROUGEスコア**: 提案手法は、テキスト要約のROUGEスコアにおいて他のベースライン手法よりも優れている。
- **カバレッジと多様性** 提案手法は、要約のカバレッジと多様性の両方において良好な結果を示した。
- **画像精度** 提案手法は、画像選択の精度においても他の手法に比べて高い精度を達成している。

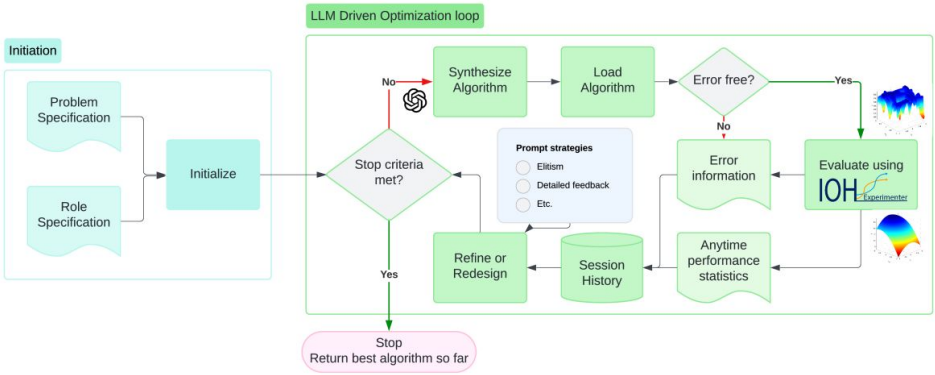
数式の説明

1. **目的関数** $f(A)=\sum u\in[d]wu\sum x\in A\sum y\in Dxuyu-\sum x\in A\sum y\in Axuyu+\sum x\in AI\sum y\in ATxuyu+|D|\sum x\in Axuf(A)=\sum u\in[d]wu\sum x\in A\sum y\in D\boldsymbol{xu}yu-\sum x\in A\sum y\in A\boldsymbol{xu}yu+\sum x\in AI\sum y\in AT\boldsymbol{xu}yu+|D|\sum x\in A^{**}xu$
    - AA: 選択された要素の集合
    - DD: 元の文書内の要素の集合
    - $wuwu$: トレーニング可能な重み
    - $xuxu$: 要素 x の埋め込みベクトルの次元u
2. **損失関数** $\min w\geq 0\sum i=1M(\max(\max A\subseteq Di,|A|\leq K\{f(A)\}-f(Ai*),0)+\lambda 2\ //\ w\ //\ 22)\min w\geq 0\sum i=1M(\max(\max A\subseteq Di,|A|\leq K\{f(A)\}-f(Ai*),0)+2\lambda\ //\ w\ //\ 22)$
    - $Ai*Ai*$: グラウンドトゥルースの要約
    - $\lambda\lambda$: 正則化パラメータ

LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics
LLaMEA: メタヒューリスティクスの自動生成のための大規模言語モデル進化アルゴリズム 2024

概要

LLaMEAフレームワークを使って、ユーザーがやりたいこと（タスク定義と基準）を入力すると、LLMがそれに基づいて最適化アルゴリズムを自動生成し、評価とフィードバックを通じて性能を向上させる。結果的に、複雑な最適化アルゴリズムを自動的に作成することが可能



手法

LLaMEAフレームワークは、次のようなプロセスで構成：

1. **アルゴリズムの初期化**：
   - LLMにタスクの説明と例を与えて最初の親アルゴリズムを生成する。
   - 生成されたアルゴリズムの性能を評価し、その結果を保存する。
2. **アルゴリズム生成のループ**：
   - 最良のアルゴリズムまたは直近のアルゴリズムを基に、新しいアルゴリズムを生成する。
   - 生成されたアルゴリズムを評価し、性能が向上した場合は最良のアルゴリズムを更新する。
3. **評価とフィードバック**：
   - IOHexperimenterというベンチマークツールを使用して、生成されたアルゴリズムの品質を評価し、LLMにフィードバックを提供する。
4. **進化アルゴリズムの適用**：
   - エリート戦略または一般的な戦略を用いて、アルゴリズムを選択し、次の世代に引き継ぐ。

結果

特に、生成された「Robust Adaptive Differential Evolution with Archive（RADEA）」アルゴリズムは、既存のDEやCMA-ESを上回る性能を示す

LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics
LLaMEA: メタヒューリスティクスの自動生成のための大規模言語モデル進化アルゴリズム 2024

1. アルゴリズムの初期化：

Your task is to design novel metaheuristic algorithms to solve black box optimization problems.
The optimization algorithm should handle a wide range of tasks, which is evaluated on a large test suite of noiseless functions.
Your task is to write the optimization algorithm in Python code.
The code should contain one function `def __call__(self, f)`, which should optimize the black box function `f` using `budget` function evaluations.
The f() can only be called as many times as the budget allows.
An example of such code is as follows:
<initial example code>
```
Give a novel heuristic algorithm to solve this task.
Give the response in the format:
# Name: <name of the algorithm>
# Code: <code>
```

3. 評価とフィードバック:

Your task is to design novel metaheuristic algorithms to solve black box optimization problems.
The optimization algorithm should handle a wide range of tasks, which is evaluated on a large test suite of noiseless functions.
Your task is to write the optimization algorithm in Python code.
<List of previously generated algorithm names with mean AOCC score>
<selected algorithm to refine (full code) and mean and std AOCC scores>
Either refine or redesign to improve the algorithm.

4. 進化アルゴリズムの適用:
多分こんなん

```
import random
class LLaMEA: def init(self, budget): self.budget = budget self.best_algorithm = None self.best_fitness = float('inf')
def initialize_algorithm(self):
    # 初期アルゴリズムの生成（例ランダム探索）
    initial_algorithm = """
def call(self, f):
# ランダム探索アルゴリズムの例
import random best = float('inf') for _ in range(self.budget): candidate = random.random() value = f(candidate) if value < best: best = value return best """ return initial_algorithm
def evaluate_algorithm(self, algorithm_code):
    # アルゴリズムの評価
    # ここでは仮の評価関数を使用
    return random.uniform(0, 1), ""
def mutate_algorithm(self, parent_code):
    # アルゴリズムの変異
    mutated_code = parent_code.replace("random.random()", "random.uniform(-1, 1)")
    return mutated_code
def run(self):
    # 初期化
    parent_code = self.initialize_algorithm()
    parent_fitness, _ = self.evaluate_algorithm(parent_code)
    self.best_algorithm = parent_code
    self.best_fitness = parent_fitness
    # 進化アルゴリズムの適用ループ
    for _ in range(self.budget):
        # 変異による新しいアルゴリズム生成
        offspring_code = self.mutate_algorithm(parent_code)
        offspring_fitness, error_info = self.evaluate_algorithm(offspring_code)
        # 評価と選択
        if offspring_fitness < parent_fitness:
            parent_code = offspring_code
            parent_fitness = offspring_fitness
            if offspring_fitness < self.best_fitness:
                self.best_algorithm = offspring_code
                self.best_fitness = offspring_fitness
    return self.best_algorithm, self.best_fitness
# 使用例
budget = 100 llamea = LLaMEA(budget) best_algorithm, best_fitness = llamea.run() print("Best Algorithm Code:", best_algorithm) print("Best Fitness:", best_fitness)
```

Knowledge Graph Tuning: Real-time Large Language Model Personalization based on Human Feedback
知識グラフ調整：人間のフィードバックに基づくリアルタイム大規模言語モデルのパーソナライズ 2024

概要

KGTは知識グラフを使い、LLMを個々のユーザーに合わせてカスタマイズするパーソナライズをする。パラメータを変更せず、ユーザーのフィードバックで知識を更新、これにより効率的で解釈可能なパーソナライズを実現する。

手法

Knowledge Graph Tuning（KGT）Tはユーザーのクエリとフィードバックから個人化された3つの要素からなる知識の単位の例えば、「犬 - 好き - 野菜」のように、主語、関係、目的語で構成される知識トリプルを抽出し、モデルパラメータを変更することなくKGを最適化されるKGTの手法は次のように構成されます

1. 知識グラフ（KG）の利用：
    - KGは構造化された形式で知識を保存するためLLMに外部知識を提供します。
    - KGを用いることで、モデルパラメータを変更することなくユーザーの個人化された知識をLMに反映させることができます。
2. KGTのプロセス：
    - ユーザーのクエリとフィードバックから個人化された知識トリプルを抽出しKGを最適化します。
    - 具体的には、LLMがユーザーのクエリに基づいてKGから知識トリプルを取得し、そのトリプルを使用してユーザーのフィードバックを生成します。
3. 最適化目標：
    - 高い個人化知識取得確率と高い知識強化推論確率を達成することを目指します。
    - 計算およびメモリコストを削減し、解釈性を確保するためにKGのトリプルの追加と削除を行います。

結果

KGTは以下の点で優れていることが示されました：

- **パーソナライズ性能の向上**：複数のデータセットと最先端のLLMを用いた実験で、KGTは既存の手法と比較して個人化のパフォーマンスが大幅に向上しました。
- **効率の向上**：計算およびメモリコストが削減され、リアルタイムのモデルパーソナライズが可能となりました。

Appendix