古典論理における証明タクティクを言語モデルで学習する

塚本　慧

目的

- 数学の証明の自動化を行いたい
- 今回は古典論理の証明を言語モデルに行わせることを目標とした

モデルそのものは汎用的なので命題論理以外に対しても適用可能

裏の目的

- 言語モデルの自己改善をしたい

証明タクティクとは(簡単な説明)

- 証明支援系において証明の手続きをコマンドラインとして記述したもの

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
```

排中律の証明
証明がテクニカルなので例として選びました

証明状態

∀ B : ℙ, ~ B ∨ B

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
```

タクティク

証明状態

前提 → B : ℙ

結論 → ~ B ∨ B

証明支援系では証明状態とタクティクを
与えられると次の証明状態を返してくる

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
```

証明状態

```
B : ℙ
──────────────────
~ ~ (~ B ∨ B)
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
──────────────────
False
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
──────────────────
~ B ∨ B
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
──────────────────
B
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
  apply add_dn.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
──────────────────
~ ~ B
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
  apply add_dn.
  intro C.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
C : ~ B
————————————————
False
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
  apply add_dn.
  intro C.
  apply n.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
C : ~ B
————————————————
~ B ∨ B
```

証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
  apply add_dn.
  intro C.
  apply n.
  left.
```

証明状態

```
B : ℙ
n : ~ (~ B ∨ B)
C : ~ B
──────────────────
~ B
```

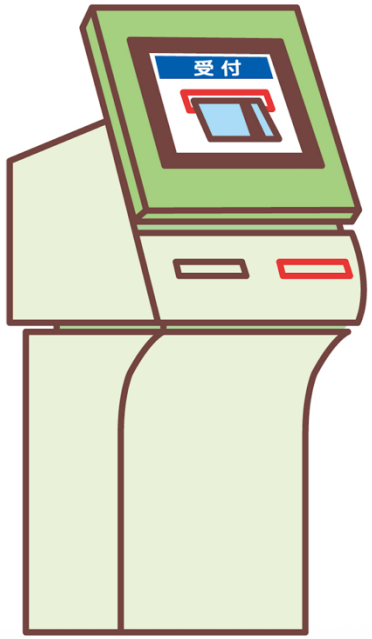
証明支援系Coqの場合

```
Theorem ss : ∀ B : ℙ, (~ B ∨ B).
Proof.
  intro B.
  apply add_dn.
  intro n.
  apply n.
  right.
  apply add_dn.
  intro C.
  apply n.
  left.
  assumption.
Qed.
```

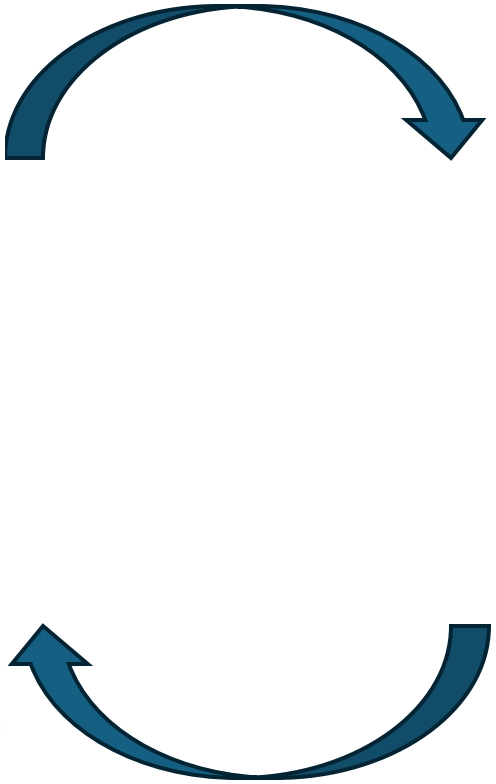
証明状態

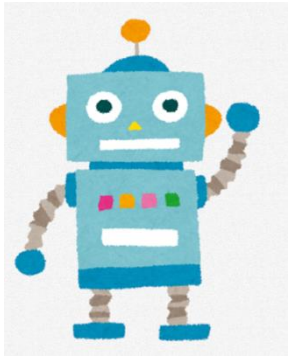
```
No more goals.
```

実験の全体像



証明支援系



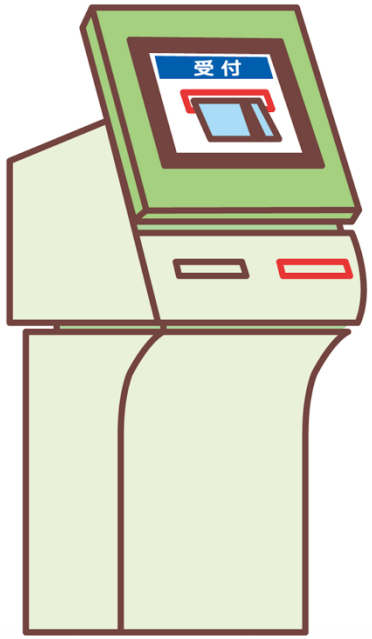


言語モデル

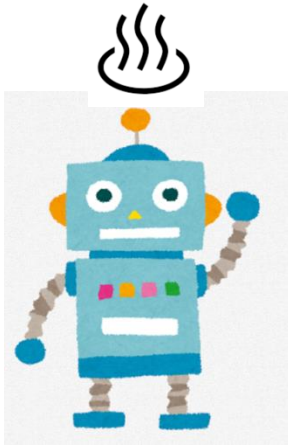
1. 証明状態を見て言語モデルが論理式を解くためのタクティクを生成
2. 言語モデルの出力したタクティクで解けるのか証明支援系で検証

証明支援系のフィードバックにより言語モデルの自己改善を目指したい

最初からフィードバックループを回そうとすると…



証明支援系



言語モデル

言語モデルでランダムにタクティクを出力するのでほとんどの場合で証明に失敗して学習データが作れない

タクティク全探索で証明を解かせるアルゴリズムで事前データを準備する

ある程度言語モデルを強くしてからフィードバックループを回す

今回の課題

- 事前データを用いて言語モデルを学習させる
- フィードバックループによって証明する能力を向上させる

ここからより詳細な話に入ります

事前学習(データ生成)

- 100万個の論理式をランダムで作り, その中でトートロジーになっているものを抽出

```
"(b → (¬c ∨ ¬((a ∧ ((c ∨ b) ∧ ¬a)))))",
"((b ∨ (c ∨ (b → (¬c ∧ ¬c)))) ∨ c)",
"((¬((¬a ∨ (b → a))) ∧ (b ∧ (b ∨ (c → c)))) → c)",
"(b → b)",
"(((¬b ∧ ((¬c ∧ b) → (c ∧ ¬c))) ∨ b) ∨ ¬b)",
"((b → (a ∨ (¬((a ∧ c)) ∨ ¬c))) ∨ a)",
"(c → (¬c → (¬c ∧ ((a ∨ a) ∨ c))))",
"(a → (a → (¬(((¬b → a) ∧ b)) ∨ ((c → a) ∨ a))))",
"((a ∨ b) ∨ ((((¬a → a) → (b ∧ c)) → a) → a))",
"(c ∨ (a ∨ ¬((a ∧ ¬(((¬a → c) ∧ (a ∧ b)))))))",
"(b → (a ∨ (c ∨ ((¬a → c) → (¬c ∧ b)))))",
"(((a ∨ b) ∨ (a ∨ (¬b → (c → ¬b)))) ∨ ¬((¬b → c)))",
```

事前学習(データ生成)

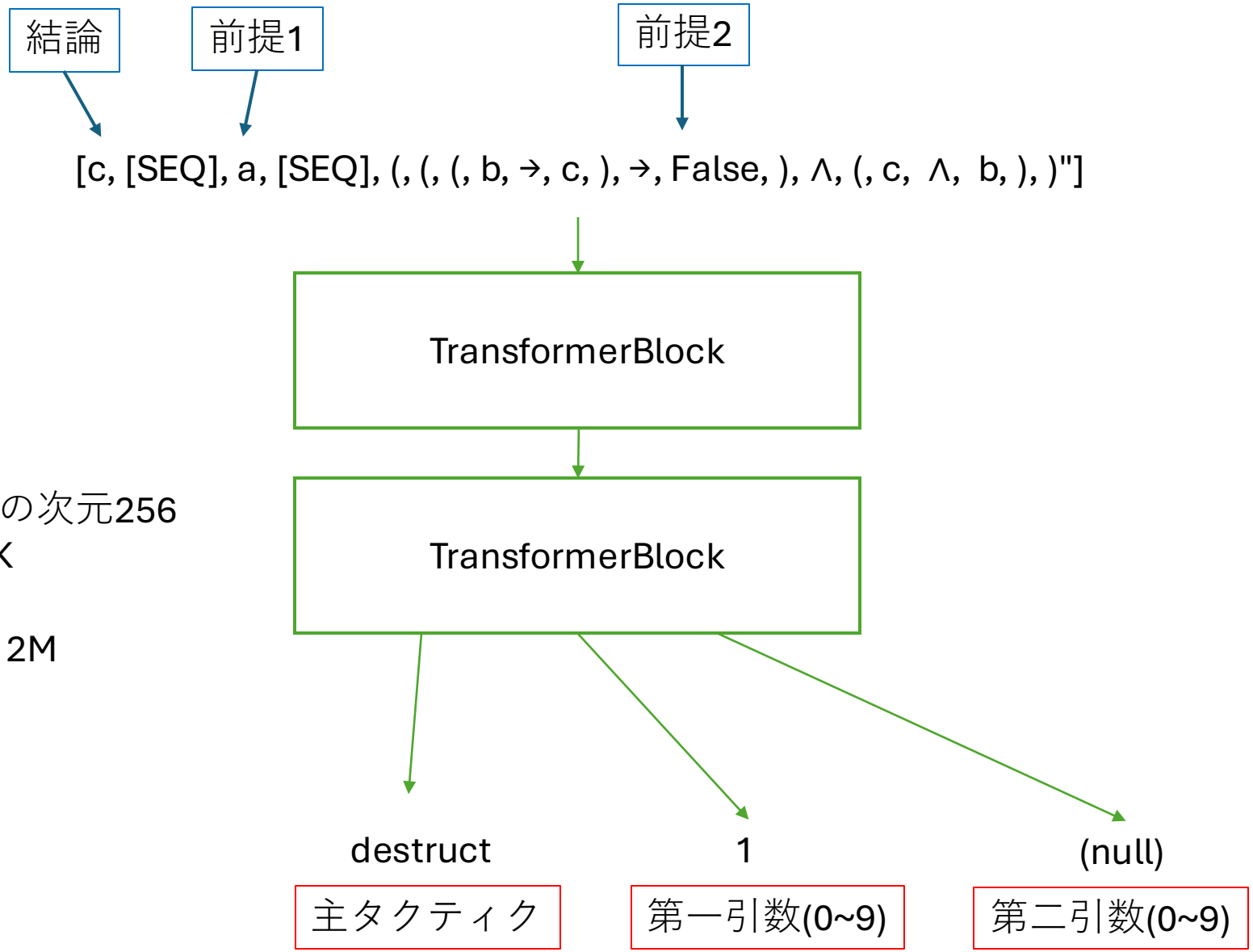
- 深さ8でタクティクの全探索を行い, 解けた論理式と解くために使われたタクティク列を学習データに加えた


```
{
  "step_index": 0,
  "premises": [],
  "goal": "(a → ((((b → c) → False) ∧ (c ∧ b)) → c))",
  "tactic": {
    "main": "intro",
    "arg1": null,
    "arg2": null
  },
  "tactic_apply": true,
  "state_hash": "c24e5ace3afe236dc333d3865a8b98d6",
  "state_tactic_hash": "c24e5ace3afe236dc333d3865a8b98d6"
},
{
  "step_index": 1,
  "premises": [
    "a"
  ],
  "goal": "((((b → c) → False) ∧ (c ∧ b)) → c)",
  "tactic": {
    "main": "intro",
    "arg1": null,
    "arg2": null
  },
  "tactic_apply": true,
  "state_hash": "120be8cc858971161112e9274a8c665e",
  "state_tactic_hash": "120be8cc858971161112e9274a8c665e"
},
```

今回用いた証明支援系でのタクティク一覧

- assumption
- intro
- split
- left
- right
- apply N
- destruct N
- specialize N M
- add_dn

事前学習



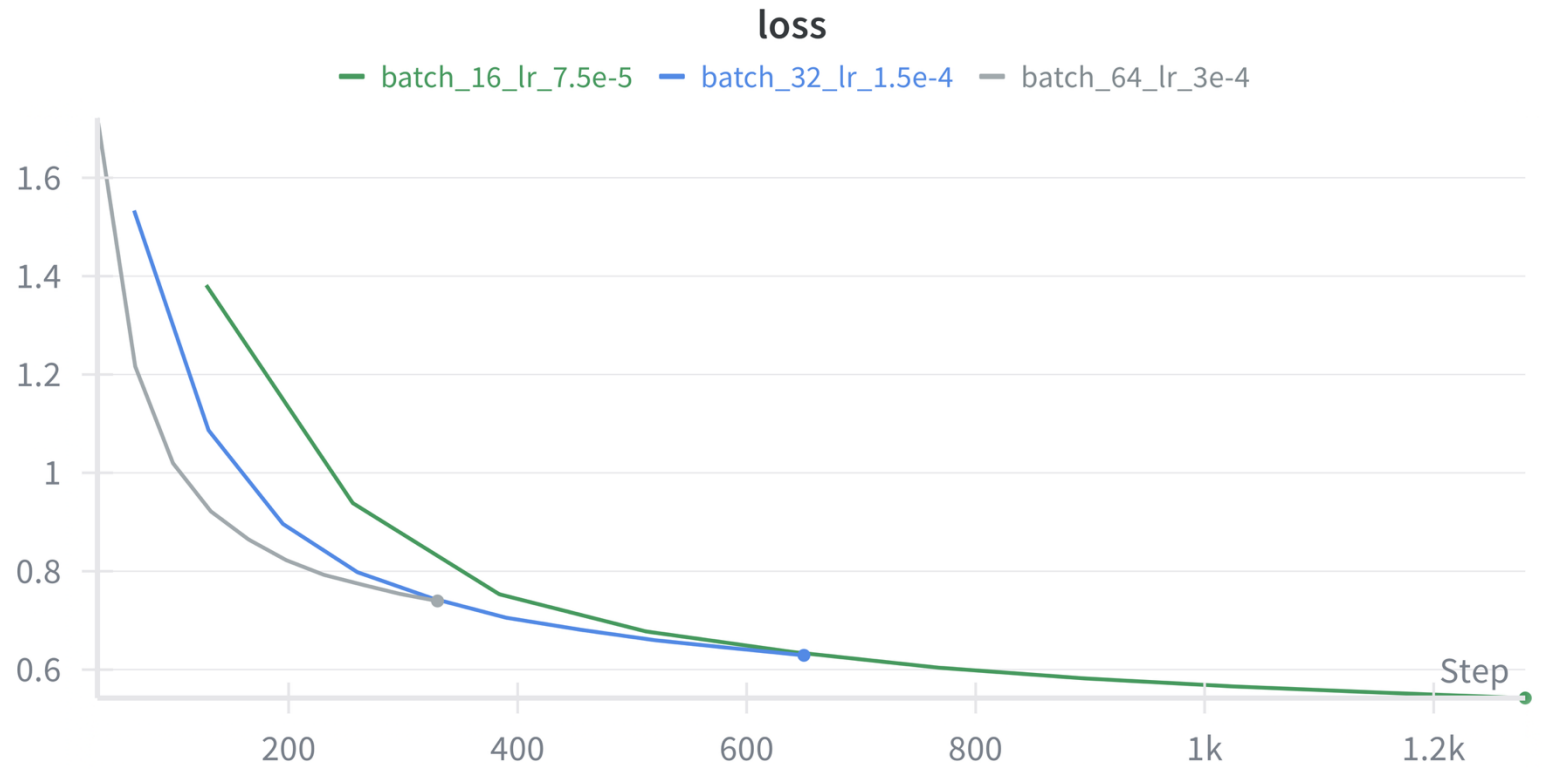
Attention headの数4
シーケンス長256
フィードフォワードの次元256
総パラメータ数 300K

データセットサイズ 2M

事前学習(バッチ学習)

- データセットサイズ 2M
- エポック数 10

事前学習(バッチ学習)

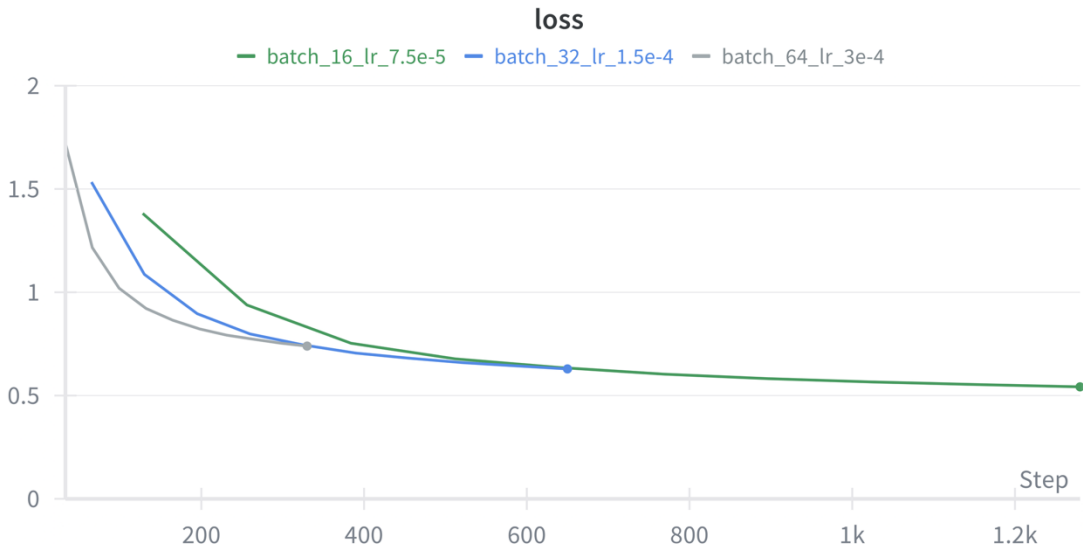


バッチサイズを減らした方がlossが下がることがわかる

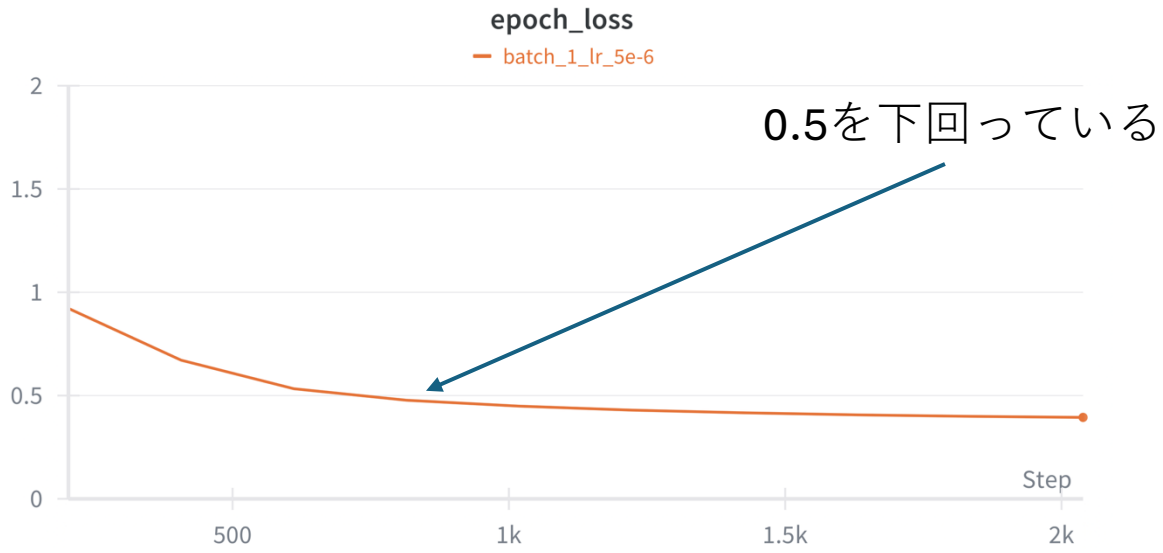
→ バッチサイズ1でやってみた

事前学習(バッチなし学習)

バッチあり



バッチなし



バッチなしで学習したモデルを事前学習済みモデルとして採用した

事前学習済みモデルの評価

- 深さ8のタクティクの全探索(auto_classical)　80.2%(588/733)
- 事前学習済みのモデル　53.6%(393/733)

- 推論は最大30ステップで, 適用できるタクティクを出力確率の順に適用していく(one-shotで評価する)

フィードバックによる自己改善

- 生成した100万論理式のなかでトートロジーな式を抽出
- 言語モデルが解いた問題をもとにデータセットを構築
- 構築したデータセットを用いて言語モデルを学習

1ループ

実験当初に立てていた仮説

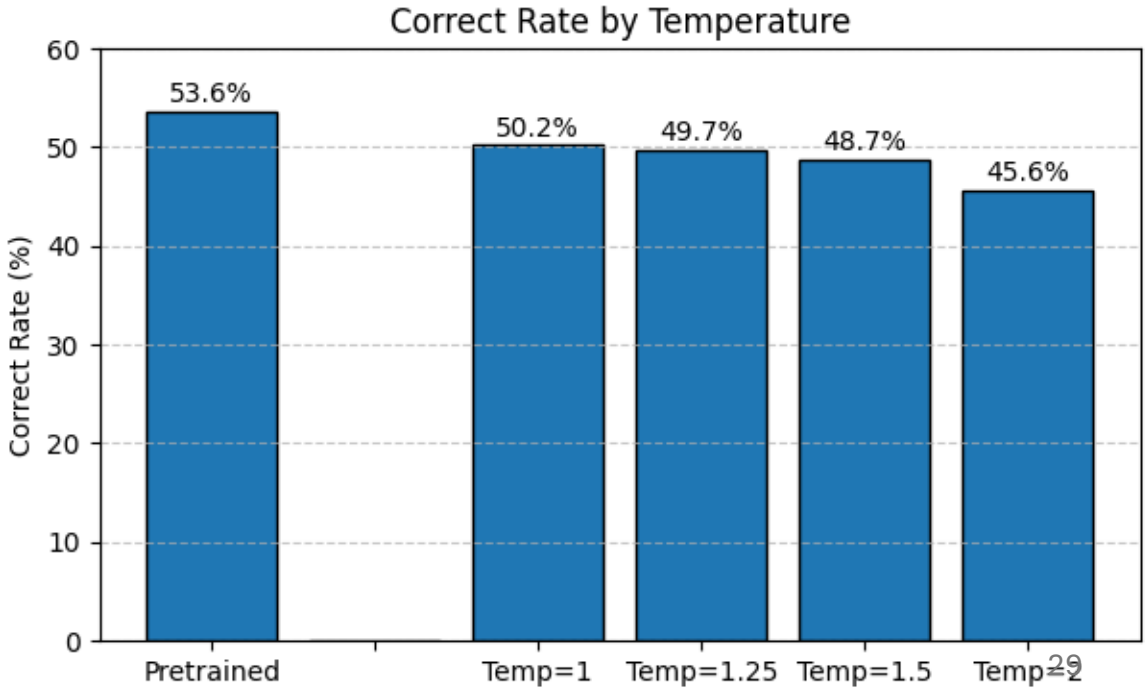
- データセットの多様性は言語モデルの生成時の温度を上げることで担保できると考えた

- 評価時と同じようにone-shotでタクティクを生成させた

結論からいうとうまくいかなかった

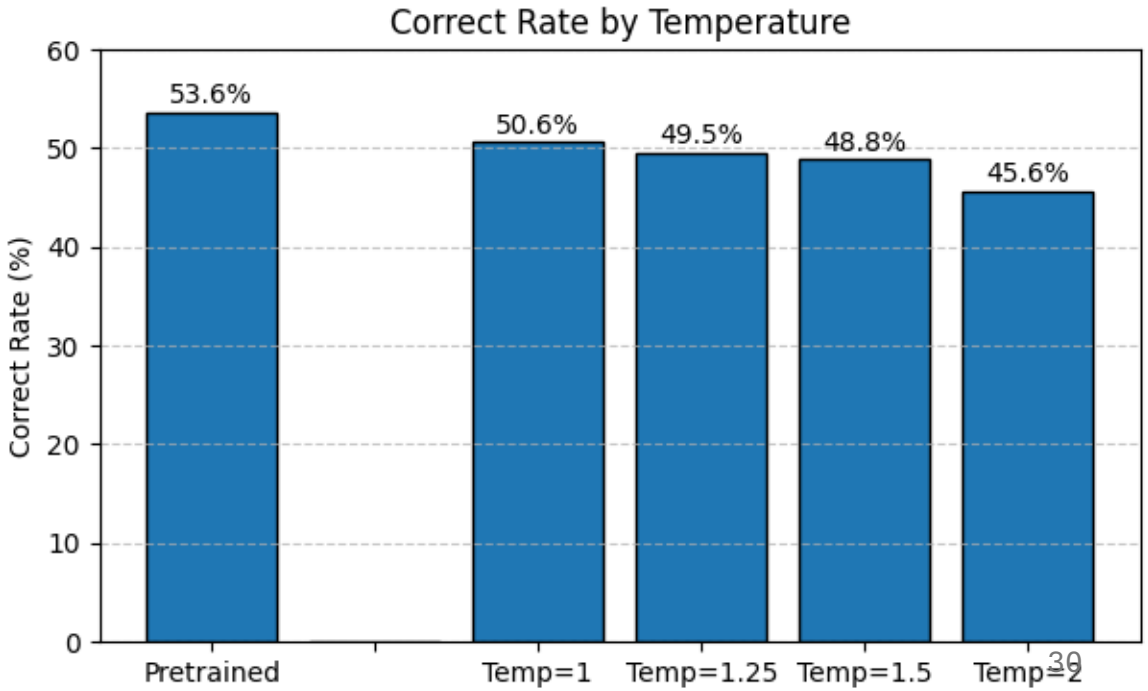
結果

- 学習率 1e-6
- バッチなし学習
- エポック数 1



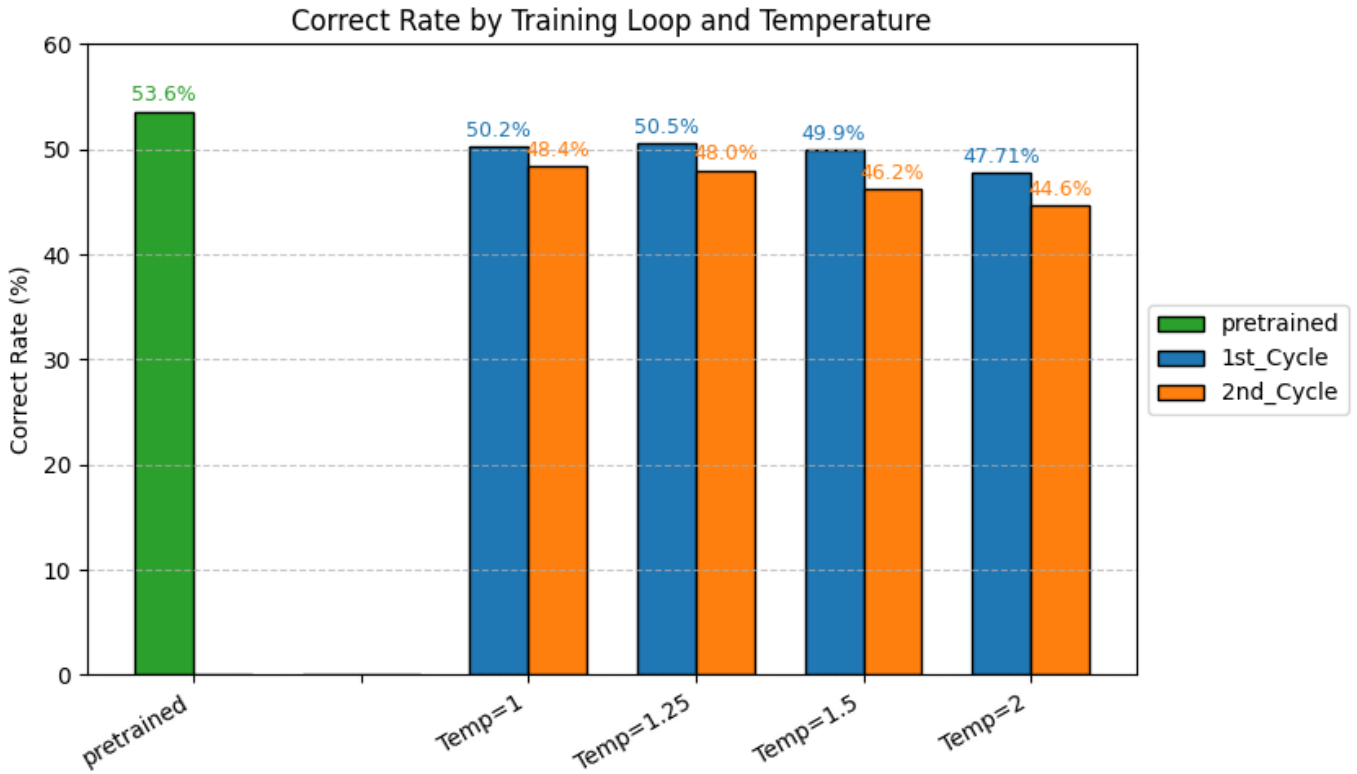
結果

- 学習率 1e-7
- バッチなし学習
- エポック数 1



結果

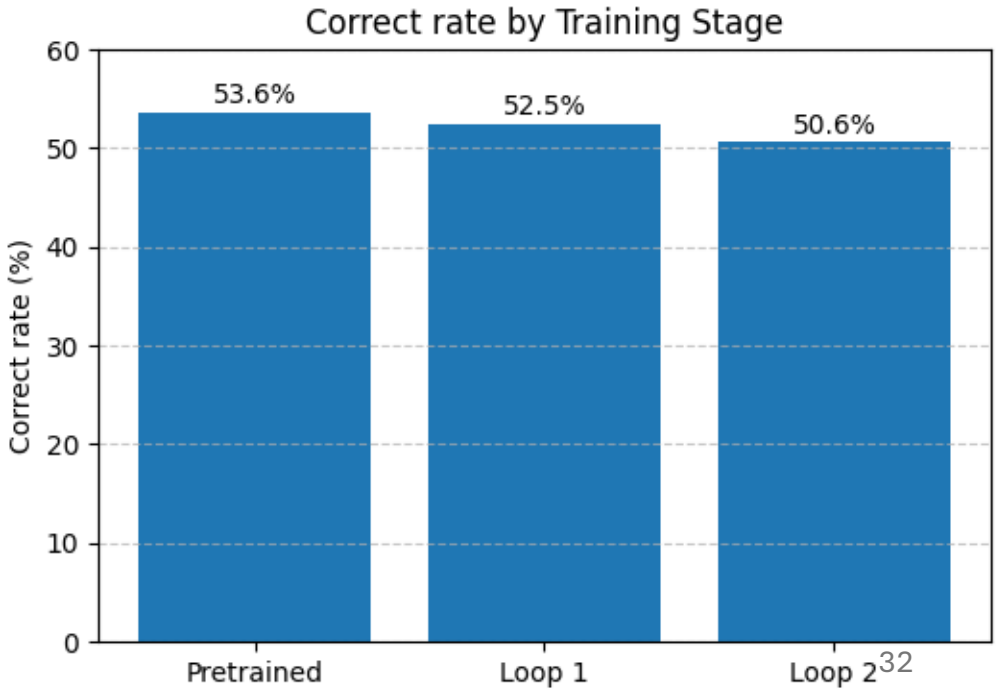
- 学習率 1e-7
- バッチなし学習
- データセットに事前学習で用いたデータを7:3で混合
- エポック数 1



結果

保守的なパラメータや方針でも正答率が下がり続けた

- 学習率 1e-7
- バッチサイズ8
- データセットに事前学習で用いたデータを7:3で混合
- 学習時のエントロピー正則化(係数0.05)
- 元のデータとのKL正則化(係数0.05)
- 温度1
- エポック数 1



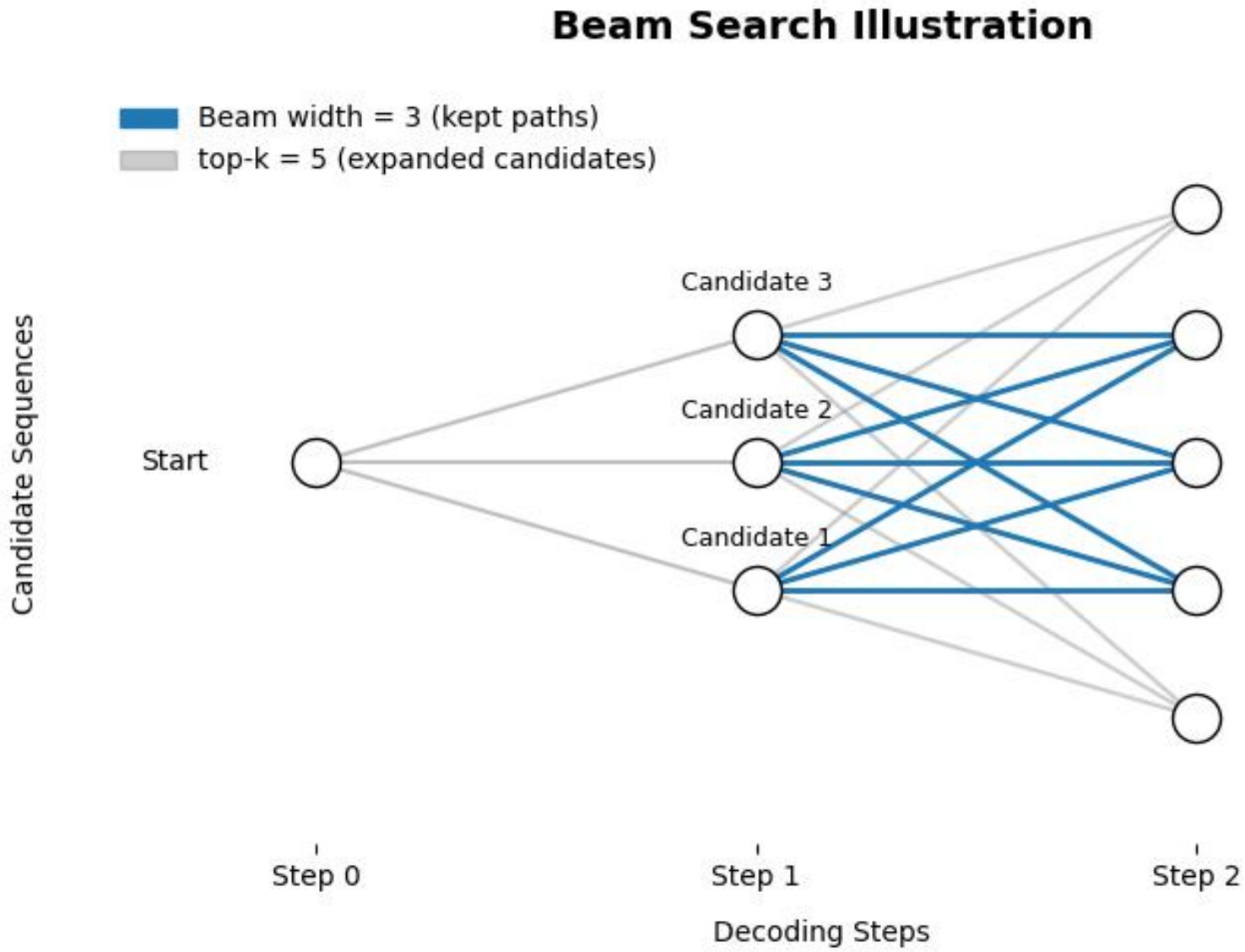
考察

- パラメータを変えながら試したが正答率はむしろ下がった
- 温度を使えば探索の多様性が担保できるという仮説は正しくないと思われる

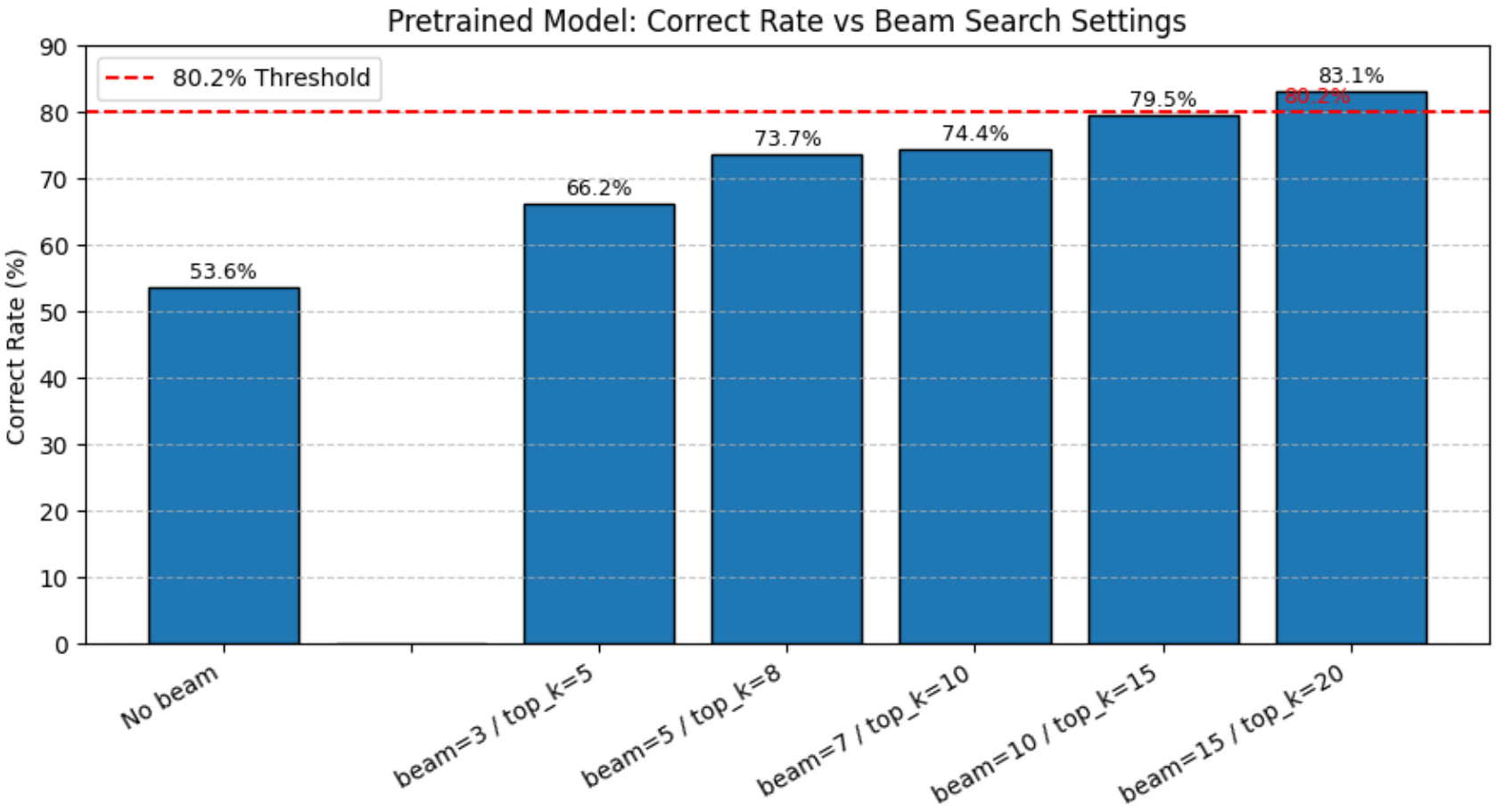
方針を変えた

- One-shotではなくビームサーチを使った推論を行うことで問題の正答率を上げた

ビームサーチとは


Beam Search Illustration
Beam width = 3 (kept paths)
top-k = 5 (expanded candidates)
Candidate Sequences
Start
Candidate 3
Candidate 2
Candidate 1
Step 0
Step 1
Step 2
Decoding Steps

事前学習済みモデルをビームサーチつきで評価する



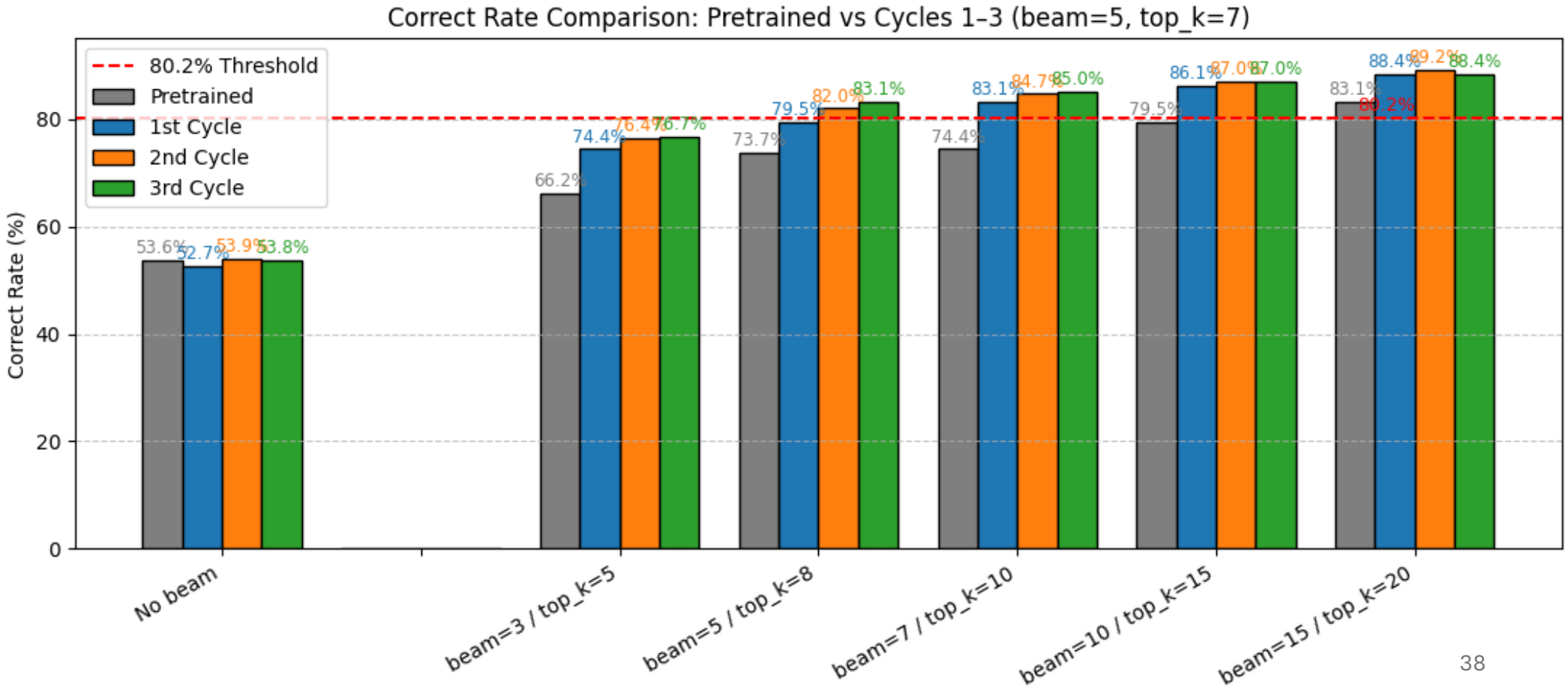
beam: ビーム幅, top_k: 各候補ごとのタクティクの数

ビームサーチ

- 学習率 1e-7
- バッチなし学習
- エポック数1

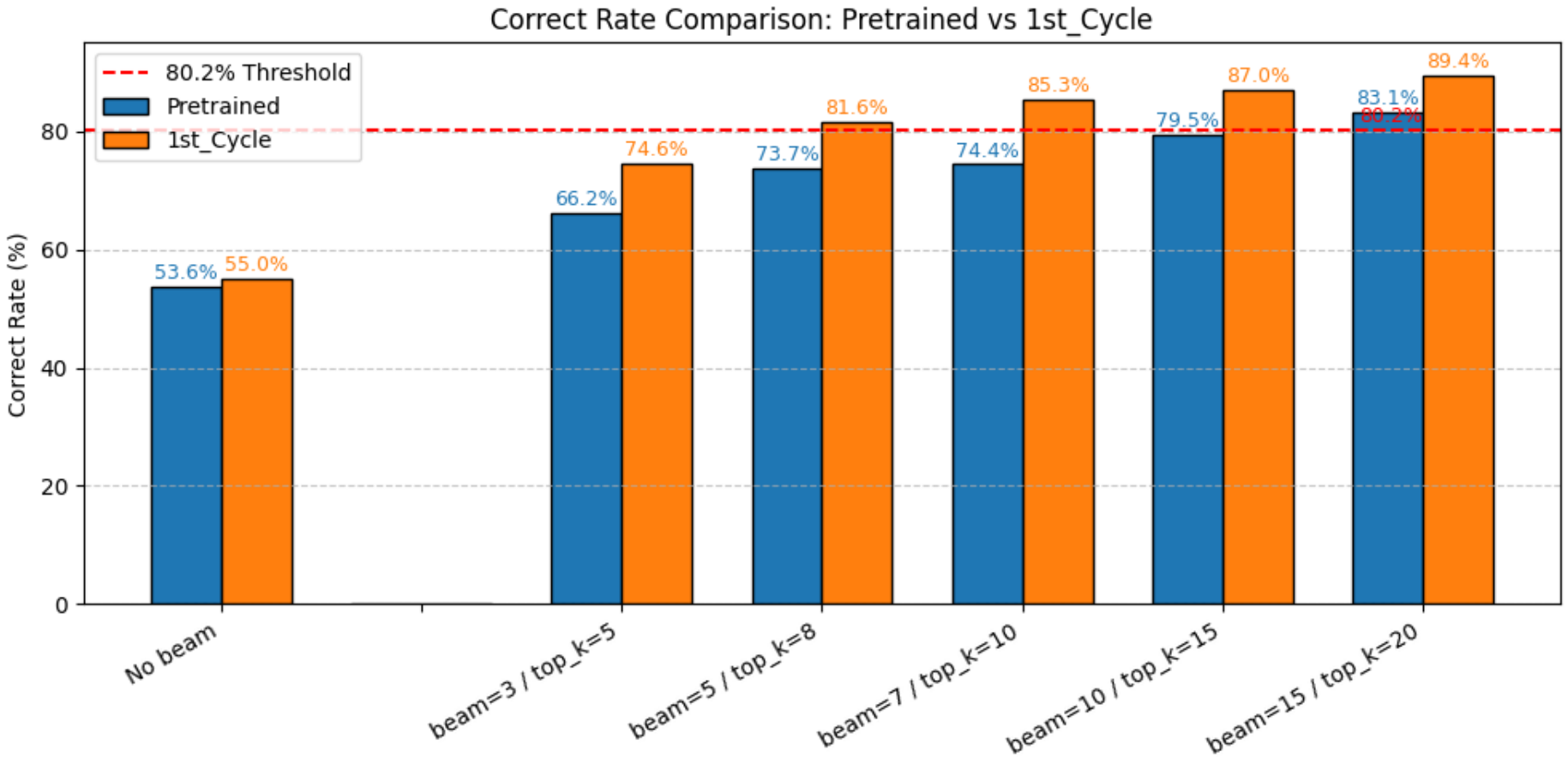
ビームサーチによるSFTの結果

- (beam, top_k) = (5,7)のとき



ビームサーチによるSFTの結果

- (beam, top_k) = (10,15)のとき



まとめ

- 古典論理における証明タクティクを言語モデルで学習した
- 事前学習後はフィードバックループによって生成した論理式で自己改善できることがわかった

謝辞

- 3年前に証明支援系のライブラリを作ってくれた同級生と, GPUを3週間貸してくれた同級生に感謝します.