2005/10/19

Rep. 寺井 仁

Soar-RL: integrating reinforcement learning with Soar

Shelley Nason and John E. Laird

Cognitive Systems Research 6 (2005) 51-59

0. Abstract

- 目的： Soar アーキテクチャの改良
- 改良点
  - 過去の行動の成功経験についての統計的な情報の学習の機会が与える
  - オペレータ選択時にこの情報が利用される
- 評価：2 つの課題を用いて Soar-RL エージェントのパフォーマンスを示す

1. Introduction

- Soar
  - 「AI アプリケーションの開発」や「認知モデル」に利用されてきた
  - 長所：効率的な表現能力
    - 多くのメソッドを用いて広範な問題を解決するためのシンボリックな知識を用いる能力
    - サブゴール
      知識が不完全かまたは矛盾する場合どうかのサブゴールを動的に作る事が可能
    - チャンキングプロセス
      サブゴールに対する問題解決をルールにコンパイル
      ⇒ サブゴールにおける問題解決はルール駆動の意思決定として置き換わる
    - チャンキングはきわめて多目的であることが証明されている
      「説明ベースの学習」,「マクロオペレータ学習」,「方略獲得」,「インストラクションによる学習」など広範な方法を用い学習することが Soar プログラムでは可能であり
  - 短所：一般的な Soar のプロセスはシンボリック
    ⇒ 認知アーキテクチャの広範囲に対して必要（そして十分）であるとはいえ,「可能性」や「数値的報酬」のエンコーディングを行う場合には不十分である
- 強化学習
  - Soar は，知識が豊富なシンボリックな推論においては長所を持ち，知識が少なく統計的な情報を基礎にした学習においては短所を持つ
    ⇒ 強化学習は逆の特性を持つ
  - 長所
    エージェントが獲得するだろうと期待される報酬に関連した統計学的な秩序を捉える
  - 短所
    シンボリックな知識を効果的に用いたり，エンコードすることはできない

- 本論文の目的
  - Soar への強化学習の統合
    ⇒ Soar の学習能力充実
  - この統合は Soar アーキテクチャに構造的な変化を求める
    ⇒ Soar-RL
- 本論文の構成
  - Soar について
  - 強化学習の統合
  - 評価課題（2 つ）
    - Soar-RL が通常の強化学習を超えた部分
    - Soar への強化学習の寄与
  - Soar-RL と ACT-R の比較
  - まとめ

2. Soar

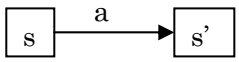
- Soar のメモリ構造 ⇒ Fig.1
  - 宣言的ワーキングメモリ
    ラベル化されたグラフ構造（状態/ゴールの階層によって構造化）を用いて現在の状況の表現を格納
  - 手続き的記憶
    プロダクションルールとしてエンコードされている
  - アクション
    - ルールの条件節がワーキングメモリの内容とマッチしたとき，ルールが発火され，アクションが実行される
    - ワーキングメモリの構造を追加，削除する
- 学習メカニズム：チャンキング
  - 問題解決をモニタし，自動的に新しいルールを作る（実行の間長期記憶に追加される）
  - サブゴールの結果を得るのに必要な処理をまとめたプロダクションルール（チャンク）
- 基本的推論サイクル ⇒ Fig.2
  - **(1)入力**：知覚の変化が処理され，ワーキングメモリの知覚バッファが更新される
  - **(2)状態評価**：新たにマッチしたルールが発火される（複数）
  - **(3)オペレータの提案**：ルールがオペレータを提案する
  - **(4)オペレータの比較と評価**：発火したルールの付加情報の値から優先度が生成される
  - **(5)オペレータの選択**：オペレータを選択するために優先度が評価される．もし，優先度が不十分または矛盾していた場合，インパスに陥る．インパス解消のため副目標が生成される．サブゴールは自身のプロセスを振り返ることに相当し，Soar はメタ推論を持つ．
  - **(6)オペレータの適用**：オペレータが適用され，状態を変化させる（モーターコマンドを含む）
  - **(7)出力**：新規のモーターコマンドが実行される

- ✧ ※ステージ 2-4 は並列に実行される．Soar の「オペレータの選択」のみが意思決定であり，シーケンシャルなボトルネックとなる．

➢ オペレータの優先度
- ✧ シンボリックな優先度（Soar で使用されている優先度）
  ⇒ オペレータを絞り込めない場合がある
- ✧ 数値的優先度（Soar-RL で導入）
  ⇒ シンボリックな優先度で絞り込めないときに，数値的優先度が考慮される

3. 強化学習の統合

➢ 強化学習の目的
- ✧ 「報酬」が最大になるように，世界においてどのように行動するかを学ぶ
- ✧ エージェントは「価値関数」を学習する
- ✧ 価値関数
  「状態-行動ペア」に対して期待される将来の報酬の合計の見積もりのマッピング（行動価値関数）



Q(s, a)：行動価値関数

- ✧ Soar-RL
  数値的な優先度に，状態－オペレータ価値関数（行動価値関数）を用いる

➢ 強化学習における問題
- ✧ 巨大な状態－オペレータ空間に対する価値関数をどのように表現するか？
- ✧ 最も単純な方法：テーブルに格納する（すべての可能な状態－オペレータのペア）
  ⇒ この表現は状態空間すべて記述することに相当し，重くて遅い

➢ 改善点
- ✧ ルール（行動価値関数）を動的に構成する
- ✧ オペレータは複数のルールによって活性化され（各々のルールは専門化した複雑な特徴（状態）検出器），複数の価値関数の線形結合として優先度を表現する

➢ Soar への強化学習の組み込み
- ✧ 数的優先度を作りながら異なる特徴（状態）セットをテストする新しいルールを作る
- ✧ 存在するルールに対する数的優先度の値を調整する

4. 優先度の修正

➢ オペレータに対する数的優先度は，ルールが数値をワーキングメモリにある特定の特徴セットと結びつけることによって生成される
- ✧ ルールは，多くの異なる状態やオペレータに渡って適用するのに一般的となっている
- ✧ 例：迷路世界のエージェントは Fig. 3 (a)のルールを持っている ⇒ Fig. 3

➢ 強化学習によってなされること
- ✧ エージェントが獲得する報酬によってルールの値を調整する（現在は，-8.2）
- ✧ 調整によって，ワーキングメモリの状態 s で与えられたオペレータ o に対する優先度の

総和が行動価値（Q(s, o)）を推定

- ✧ このような調整とソフトグリーディ方略とを組み合わせることにより，期待される報酬を最大にするような行動をエージェントが学習する

  ※ソフトグリーディ方略

  優先度を比較してオペレータを選択する場合，優先度が最大のものばかり選択するとローカルミニマムに陥るため，わずかな確率（ソフトな）で優先度が最大のもの意外を選択すること．これによって状態空間の探索が促進される．

- ➢ 例
  - ✧ 迷路世界のエージェントは「東に行く」というオペレータを選択したとする（Fig. 3 (a)と(b)における優先度の合計は-5.82 と計算される）
  - ✧ 次のステップで行動価値のゲインが-1 だった場合, 2 つのルールは(<s> ^operator <o> = -1.32)と(<s> ^operator <0> = -5.5)のアクションを持つものとして更新される
- ➢ 優先度ルールのプロトタイプ ⇒ プロトタイプフォーム
  - ✧ これらのルールはプロトタイプのフォームで表現可能
  - ✧ このルールが特定の状況において発火したとき，Fig. 3(a)と同様に，定数によって置き換えられた変数<c>, <d>, <direction>を伴った新しいルールがビルドされる
- ➢ プロトタイプフォームの意味
  - ✧ このプロトタイプによって，位置，目的，方向のすべての組み合わせと変数を自動的に結びつける
  - ✧ 実際に出会った組み合わせに対して特定のルールを構築（自動）するだけでよい

5. 結果

- ➢ 2 つのドメインでエージェントをテスト
  - ✧ 2 つの例では, ほとんどまたはまったくオペレータの選択知識を Soar エージェントに記述せず，初期の行動がランダムに決定されるように記述した
  - ✧ 強化学習のみを使用（チャンキングは未使用）
  - ✧ 予測

    エージェントが世界において経験を獲得しながら，パフォーマンスが改善するだろう

5.1. 宣教師と囚人問題

- ➢ 報酬
  - ✧ 成功状態（宣教師と囚人すべてが右側の岸にたどり着いたら）

    ⇒ +1 の報酬
  - ✧ 失敗状態（どちらかの岸で囚人の数が上回ったら）

    ⇒ -1 の報酬
  - ✧ それ以外

    ⇒ 0 の報酬
- ➢ ワーキングメモリの内容
  - ✧ どちらの岸にそれぞれ何人いるのか

- ✧ ボートはどこにあるのか
- ➢ 状態は直接的に観察可能であり，状態空間は小さい
  - ✧ 各状態－オペレータペアに対して一つの数的優先度ルールの設定（プロトタイプは使用しない？）
  - ✧ エージェントは数的優先度を持つ以外，統制された知識を持たない
- ➢ 結果 1 ⇒ Fig. 4
  - ✧ 500 回のトレーニング結果
  - ✧ エージェントはおおむね成功
  - ✧ エージェントの失敗は，現在のソフトグリーディ方略の乱数性に起因
- ➢ 結果２ ⇒ Fig. 5
  - ✧ 「前の状態に戻る移動」に対して最も低いシンボリックな優先度を設定
    ⇒ ヒューリスティックシンボリック優先度ルールの追加によってパフォーマンス改善

5.2. イーター

- ➢ パズル課題（例えば「宣教師と囚人問題」）の有効性
  - ✧ Soar のチャンキングメカニズムのスピードアップ効果を示すために良く使われる
  - ✧ 望まれる行動がわかっているため評価に対して有益
    ⇒ パズル課題は，報酬ベースの学習に対するテスト領域としては欠点がある
- ➢ パズル課題の欠点
  - ✧ より価値関数のテーブルベースの表現を要求
  - ✧ なぜなら，パズルの解決は，エージェントに組み合わせにおけるすべての特徴を考慮することを強いることによって達成されるため（各特徴のインプリケーションに独立に計算するよりもむしろ）
- ➢ イーター問題
  - ✧ パズル課題の欠点を避けるために使用された
  - ✧ パックマンライクのエージェントがボードの上を動きながらフードを探索する
  - ✧ ボードには２つのタイプのフードが配置されている（ボーナスフード，ノーマルフード）
  - ✧ 報酬
    - ボーナスフードのところに移動：＋１０
    - ノーマルフードのところに移動：＋５
    - 何もないところに移動：０
  - ✧ 行動：上下左右
  - ✧ エージェントの知覚能力（の一例）
    - Type 1：近接した４つのセルの一つの内容
    - Type 2：近接した４つのセルのすべての内容
    - Type 3：最初の移動に対して２回目の移動で見ることが可能な新たな３つのセルの内容
    - Type 4：自分を中心とした２５のセルの内容
    - ・・・

- 強化学習エージェント（2 種類）と統制用エージェント（2 種類）の優先度
  - ✧ One-step エージェント
    Type１：移動先のセルの内容と結びついた数的優先度
    ３つの数的優先度ルールを学習
    - ボーナスフードに動け
    - ノーマルフードに動け
    - 何もないセルに動け
  - ✧ Two-step エージェント
    Type１＋Type３：Type１と「移動先のその先のセルの内容と結びついた優先度」の和
  - ✧ ランダムエージェント（優先度：ランダム）
  - ✧ シンボリックエージェント（シンボリック優先度：ボーナスフード＞ノーマルフード＞なし）
- 結果　⇒　Fig. 6
  - ✧ ６００回の移動×２０回実行
  - ✧ 強化学習エージェントは両者共にランダムエージェントよりも十分に学習がなされた
  - ✧ One-step エージェントのパフォーマンスはシンボリックエージェントと同一
    獲得した優先度：50.12, 29.38, 5.66(ボーナス，ノーマル，なし)
  - ✧ Two-step エージェントでは，比較的小さいが，一貫した向上が見られた
  - ✧ この世界では，一般的に One-step エージェントよりも Two-step エージェントが有効
    - 例 ⇒ Fig. 7
    - One-step オペレータ
      東西南北それぞれ（empty）への移動：数的優先度＝14
    - Two-step オペレータ
      東への移動(empty, empty, normalfood)：数的優先度＝-6.6
      西への移動(empty, empty, wall)：数的優先度＝-13.8
  - ✧ 獲得ルール
    - Two-step エージェントは５６の Two-step の数値的優先度ルールを学習
    - しかし，３つの One-step ルールに対して，パフォーマンスの向上において不釣合い

    ⇒ この世界で必要とされるコントロールの知識のほとんどは One-step ルールによって捉えることができる
    ⇒ それでも，２つのタイプの特徴が独立には考えられないとした場合，要求されるだろう 3×56 のルールを学習することを回避

6. Comparison with ACT-R

- ACT-R は強化学習と類似の学習コンポーネントを持っている
  - ✧ ACT-R におけるルール選択
    - Soar におけるオペレータのレベルに対して，個々のルール選択のレベルで行われる（いわゆる競合解消）
    - 一度に一つのルールのみ発火し，ルールの選択は確率的に行われる（各ルールの効

用をもとに，ボルツマン選択に類似したルール選択）

- ルールの効用
  現在のゴールの価値と同様に，特定のプロダクションが選択されることによるゴールの達成の期待コストや確率の関数（ゴールの価値以外は経験から学習）

➢ Soar-RL と ACT-R のルールチューニングの違い

✧ ルールの効用

- ACT-R
  シングルゴールに関連
  異なるゴールに敏感なオペレータに対する多くの異なるルールがある
  ⇒ ACT-R で開発されるエージェントは，変化するまたは複数のゴールを伴った環境において学習が困難
- Soar-RL
  単にゴールへ向かって進むわけではない
  よりフレキシブルに報酬関数と関連している

✧ オペレータの妥当性と価値

- ACT-R
  行動の効用と選択の可能性は各ルールの条件にのみ基づいている
- Soar-RL
  行動が「妥当な時」と「価値のある時」に対する条件を区別している
  ⇒ 行動価値関数により行動に対してよりリッチな表現を与える

✧ オペレータの貢献

- ACT-R
  同じ行動に対して異なる条件を持つ複数のルールあるが，学習は選択されたルールのみに手柄を与える
- Soar-RL
  オペレータの選択の結果に対して関連したルールすべてが手柄を得る

7. まとめ

➢ 本論文では，Soar アーキテクチャへの重要な拡張（強化学習の組み込み）を示した
➢ オペレータの選択を制御する知識を学習する方法を，期待される報酬を基礎にした，新たなアーキテクチャ上のアプローチを提供した
➢ これにより，すべての Soar プログラムは，シンボリックな優先度が意思決定するのに不十分な場合に，環境からのフィードバックに自動的に適応することが可能となる
➢ 認知モデリングに対する Soar におけるインパクトは明確でない，しかし，魅力的である
➢ Soar-RL は数的およびシンボリックな評価の結合と学習において ACT-R の表現に近づいた
➢ ACT-R とは多くの共通した特徴を共有するが，十分に異なるメカニズムでもあり，人間の行動の異なる計算機モデルをもたらすだろう
➢ このモデルが現在の ACT-R 理論をしのがないとしても，ACT-R モデルのどのような側面が最も重要であるのかといった点において新たな洞察を与えてくれるだろう