Learning to Assign Credit in Reinforcement Learning by Incorporating Abstract Relations

物体検知や位置推定などで画像の抽象的な情報を抜き出すことで、報酬が疎な環境でも学習しやすくしました。

現実に近いタスクはこの問題に悩まされがちなので、読んでみました。

適用環境はゲームに関する深層強化学習だけなので業務に完全に使うのは難しそう。

Abstract

- Credit assignment problem(貢献度配分問題)は強化学習では非常に重要な問題である。
- 貢献度配分問題：
    - 報酬を獲得した場合、報酬の獲得にどの行動が効いたのかはエージェントにはわからない。
- この問題は決定過程が幾千もの行動に関わってくる現実に近い環境ほど影響が強くなる。
- そこで、状態-行動間の貢献度を分配する計算過程を効率化することで、学習速度を向上させる。
- 具体的には、はじめに元の問題の状態と行動をより抽象化させることで、コンパクトな表現にする。これによって、元の問題を扱いやすいサイズにする。
- そのあと、抽象化した問題に対して最適な価値関数を学習し、未来の価値を予測する。
- 最終的には、抽出した価値関数から、貢献度をを元の状態-行動ペアに割り当てる。
- 実験ではDoom環境に適用し、結果として、過去のエージェントに対してスコアに大きく差をつけた。
- エージェント同士を戦わせるコンペにも出場し、2位を獲得

Introduction

- 貢献度分配問題は強化学習においてもっとも重大な挑戦である
- 一般的な環境では最終的な報酬は膨大な数の行動に依存しているため、この問題は非常に難しい。
- ほとんどの手法が、設定した期間の間の報酬を伝搬させることで無理矢理解決している。
- Reward shapingなどはエキスパート知識を組み合わせることで報酬関数に補正を加えて、この問題を解決している。
    - しかし、エキスパート知識は概念であり、関数として扱うには難しい。

    - 結局は手作業で補正を設定する羽目になる。
- この問題を解決するため、貢献度分配問題のフレームワークを提案する。
- 元の問題を抽象的な形式に圧縮→抽象的な形式での最適な価値関数を算出→元の問題の貢献度分配問題に適用していきます。
- この方法によって、何十万もの状態-行動のペアの貢献度を自動的に割り当てることができる。
- 提案手法は3つの段階に分割される。Abstraction, Planning, Feedbackの3つである。
    - Abstractionフェーズでは、一階述語論理を使って、オリジナルの問題の状態(オリジナル状態)を抽象的な状態(リレーショナル状態)する。背景など、学習に関係ない情報は除外されるので、リレーショナル状態の数はオリジナル状態の数よりはるかに小さくなる。
    - planningフェーズでは、価値反復法で、各リレーショナル状態に対して、最適な価値を算出する。
    - Feedbackフェーズでは、前のフェーズで得られた価値関数を用いて、オリジナルの報酬関数に補正を加える。
- 実験の環境としてFPSゲームのDoomを採用した。
- オリジナルの報酬関数は敵を倒すことであるが、エージェントは複雑なマップ内を索敵し敵と戦闘し勝利してようやく得られる報酬であるため、報酬は疎であり、かつ遅れて得ることになる。
- 過去に発表されたエージェントとの対戦、実際に開催されたコンペティションのランキングを評価実験として示す。
- 対戦では、過去に実装されたエージェントと戦わせた。その結果、キル数、デス数共に過去エージェントを上回った。
- 最後にDoom AI コンペティションに出場した結果、2位を獲得し、1位のエージェントのスコアとは僅差であった。

Related Work

Deep Reinforcement Learning

- ゲームを対象としたDRL研究が盛んに行われている。
- エージェントは目的の状態に到達するために適切な行動を学習する。
- 代表的な成果といえば、3Dゲームを対象としたチャレンジであり、 Deep Recurrent Q-learning やカリキュラム学習が有名。
- にも関わらず、いくつものチャレンジが計算量の都合上、残っている

Relational Reinforcement Learning

- Relational Reinforcement Learningは強化学習と Relational Learningを組み合わせたもので、状態、行動、Q関数を表現力の高い言語で示していることから内部が構造的なタスクに効果的である。

- ZambaldiらによるDRRLはエンティティ間の関係を推論するためにself-attensionを使用して、モデルに依存しないポリシーを学習した。
- 本稿では、RRLの技術を導入し、よりコンパクトな問題の表現を試みる。

Method

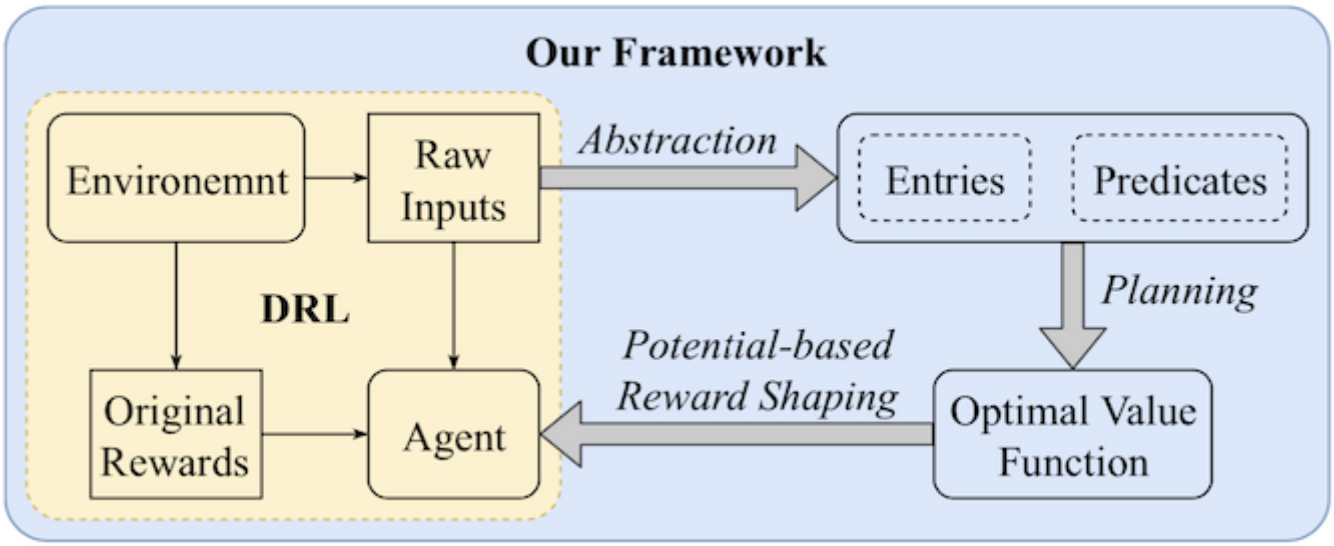
- Markov Decision Process(以下MDP)を前提におく。
- MDP: 状態遷移が確率的に生じる動的システムの環境モデル
  - S：取りうる状態集合
  - A：取りうる行動集合
  - T：状態間の遷移関数
  - R：報酬関数
  - λ：過去の報酬の割引係数
- 既存手法の代表としてQ学習を例に説明していく。
- Q学習とは各状態sと各行動aに対して行動価値関数$Q(s,a)$を定め、エージェントはQ関数を学習することで、最適な行動を学習する事になる。
- 状態価値$Q(s,a)$の更新式

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s\prime) + \lambda \max_{\alpha'} Q(s',a') - Q(s,a)]$$

- αは学習係数、$R(s,a,s')$は状態sから行動aによって次の状態s'に遷移した時の報酬、$\lambda \max_{\alpha'} Q(s',a')$は次の状態の行動価値関数$Q(s',a')$の全ての行動に対する最大値である。
- 要するに、未来に獲得できるであろう報酬を学習して行動価値関数Qに反映させている。
- 既存の現実のシナリオをQ学習で解こうとすると、報酬をもらえる頻度が少ない場合に学習がうまくいかないことがある。
  - 例えば、チェスの勝敗や迷路の脱出、FPSだと敵を倒すまでに時間がかかる。
- その場合、既存手法ではもらった報酬を長い間伝搬させる必要があるため、収束が遅れてしまうことがある。
- そこで新たにRelational Markov Decision Process (以下RMDP)を導入することで、その問題を緩和させる。
- RMDPはMDPにおける状態、行動を抽象化することで、より小さい問題に落とし込むことを目的としている。
- RMDPを通した学習により、元の報酬に補正を加えることが本手法のキモである。
- RMDP:
  - $< S_r, A_r, T_r, R_r >$
  - S_r: MDPの状態Sを抽象化させた状態、一階述語論理で表す
  - A_r: MDPではボタン操作がAであるが、RMDPでは$move(x,y)$など、述語で表す。
  - T_r: S_rの状態遷移行列
  - R_r: 状態遷移の時の報酬関数
- RMDPの主なアドバンテージはMDPでは複雑だった状態を抽象化させることで数を少なくすることである。

- これにより、エキスパート知識を使って報酬関数を設計するのが簡単になる。

提案手法のフローのイメージ↓



- Ablation, Planning, Feedbackの3つのフェーズから成る。
- はじめに、入力画像を一階述語論理のオブジェクトと述語に変換し、RDMPの状態にする。
- planningフェーズでは、RMDPでの最適価値関数を求める。
- 最適価値関数をreward shapeに適用することで学習プロセスを高速化する。

Abstraction

- 入力：オリジナルの状態と行動、出力：RMDPとして変換した状態と行動
- RMDPとして変換した状態と行動は一階述語論理として表現
- 数式としては次のように表す。
  $S_r = f_s(S), A_r = f_s(A)$
- S_r, A_rでは学習に必要な情報のみをS, Aから抽出する。
- 例えば画像に関するタスクの場合、背景情報を削除することで前景の情報のみを保存する。
- 行動がボタン操作などだったら、座標に移動するなどにマッピングする。
- この段階では複数の状態を1つの状態として表すので、状態数の削減が期待できる。
- この実装は元のタスクの入力形式や保持したい情報の種類によって異なる。
- シンプルなグリッドゲームの場合はルールベースの変換で十分だが、視覚情報を使うタスクでは、物体検知やImage segmentationなどが有効である。
- 抽象化の例
  1. 入力の画像: 箱の上にリンゴが乗っかっている。
  2. 物体検知でリンゴと箱の位置をそれぞれ推定。
  3. 座標位置から$on(apple, box)$をRMDPの状態として変換。
  4. 行動として予め定義していた述語($move(x, y)$など)を各行動にマッピングする。

Planning

- 入力：RMDPの状態と行動、出力：RMDPの最適価値関数
- もしRMDPの要素S_r, A_r, T_r, R_rが全てわかっているなら、強化学習の問題は計画問題と等価であるので簡単に解ける。
- T_r: 状態s_rから到達できる全ての状態s_r'に遷移できる確率を持つ行列
  - s_rから遷移できる行き先s_r'は論理演繹規則を事前に定義することで全て列挙する。
  - s_rからs_r'に遷移する確率は、リプレイデータから演算する。
- R_r: MDPにおけるRの部分集合となる
  - MDPでの報酬が非ゼロの状態-行動のペアはRMDPに引き継がれている。
- T_rとR_rも設計できたので、価値反復アルゴリズムによって最適価値関数を求める。
- 状態s_rの時の価値関数V_rは再帰的に計算される。

$$V_r(s_r) = \max_{a_r \in A_r}[R_r(s_r, a_r, s_r') + \lambda \sum_{s_r' \in S_r} T_r(s_r, a_r, s_r')V_r(s_r')]$$

- 全ての選択できる行動a_rによってえられる価値の最大値を求めている。
- $R_r(s_r, a_r, s_r')$：a_rを選択した時の報酬 ($R_r(s_r, a_r)$の方が正しいかもしれません)
- $\lambda \sum_{s_r' \in S_r} T_r(s_r, a_r, s_r')V_r(s_r\prime)$：$a_r$を選択した時の遷移先の状態$s_r'$の価値の期待値

Feedback

- 入力：状態S_rに対する最適価値関数V_r、出力：MDPの報酬関数Rの補正
- 前段階の出力となるV_rは貢献度割り当てのために利用される。
- MDPの状態sを入力としたポテンシャル関数ϕを定義する

$$\phi(s) = V_r(f_r(s)) = V_r(s_r)$$

- 要するに、状態sをRMDPにマッピングした時の最適価値関数である。
- ϕは sの直接的な最適価値関数ではないので、直接学習に適用することはできない。
- そこで、reward shapingの手法に適用する。
- Reward shaping:
  - 強化学習の通常の報酬値に、追加の値を加えることで、学習速度を向上させることを目指すフレームワーク。
  - $R'(s, a, s') = R(s, a, s') + F(s, s')$
- この手法ではFはこのようになる。

$$F(s, s') = \lambda\phi(s') - \phi(s)$$

- 最終的にMDPに適用される新たな報酬関数R'は以下のように導出される

$$R'(s, a, s') = R(s, a, s') + \lambda V_r(f_s(s')) - V_r(f_s(s))$$

- $R(s, a, s')$は短期的な状態価値を意味している。
- $\lambda V_r(f_s(s')), V_r(f_s(s)$は長期的な状態価値を意味している。

Application

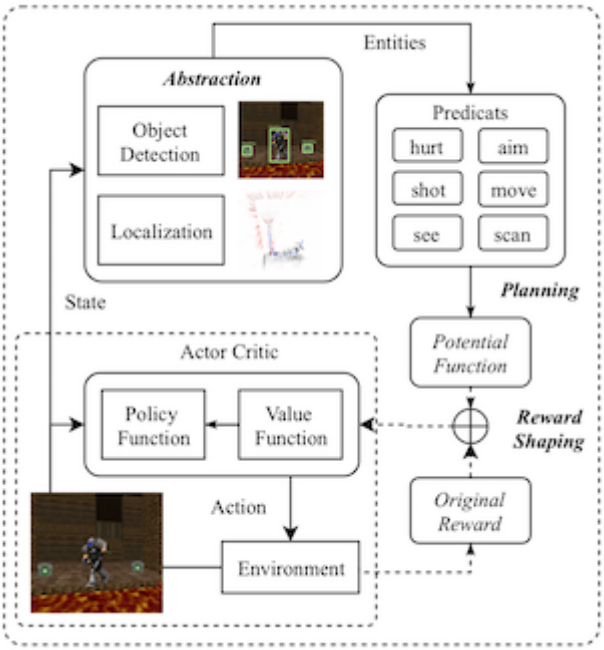
- Doomを対象にして実験を行う。
- Doomを選んだ理由：
  - 貢献度分配問題の影響を非常に受けやすいゲームであるためである。
  - オリジナルの報酬は敵を倒すことで得られるため、報酬を得られる頻度がとても少ない。
  - そのことから、どの行動が報酬に効くのかの判定がとても重要になってくる。

ゲームのルール

- プレイヤーは１人で戦い、個人でスコアを競う。
- ピストル、ショットガン、ロケットランチャーなど多種に及ぶ
- 敵に倒されたら死亡となり、一定時間後にランダムな位置に再出現する。
- 自分の体力、残弾数など、隠されていないゲーム変数は参照できる。
- エージェントの行動は左右、前後の平行移動、左右旋回、攻撃である。

Architecture

- Doomにおける提案手法とA3Cを組み合わせたフレームワークが↓



- 提案手法はDRLアルゴリズムに依存しないため今回はDoom上で実績があるA3Cを採用している。
- 画面下
  - 既存手法のA3Cを用いている。
  - 環境から得られた画像を方策関数と価値関数に与えることで最適な行動を予測している。

    - 普通なら、環境からもらった報酬も使うところだが、提案手法を通して一度加工したものを使っている。
- 画像上
    - 提案手法の抽象化と抽象化した状態の最適価値関数を求める処理の実装を示している。
    - 抽象化には、物体検知と場所検知を行い、ゲーム内のオブジェクトのbounding boxとオブジェクト間の位置情報を予測してい る。
    - 抽出されたオブジェクトの情報は、予め定義された述語に渡され、次の状態が決まる。
    - RMDPのパラメータは全て決まっているため、オフラインの計画アルゴリズムで最適価値は導き出される。
    - オリジナルの報酬と組み合わせることで、報酬を加工して、学習に利用している。

Abstraction

- 物体検知と場所検知を行って状態を一階述語論理に落とし込んでいる。
- 抽象化させた例は↓

Table 1: Entities and predicates in Doom.

Entities	agent	the agent controlled by us
	opponent	other agents
	items	pickable resources
State Predicate	hurt(x)	x has a health decrease
	see(x,y)	x is able to see y
	aimed(x,y)	x is targeted by y
	move(x)	x is moving
	stay(x)	x is standing still
	alive(x)	x is alive
	dead(x)	x is dead
Action Predicate	approach(x, y)	x walk to y
	keepmove(x)	x keep moving
	scan(x)	x checks around
	aim(x,y)	x aim at y by turning
	shoot(x)	x fires the weapon

Evaluation

- 2つの評価値からフレームワークを評価する。
    - frag: 敵を倒した数ー自滅してした数
        - (ゲームではロケットランチャーなど爆発する武器を使うこともあるので自滅することもありうる)
    - Death: 敵に倒された数
- 実験の条件:
    - バッチサイズ: 128
    - 割引報酬係数$\lambda = 0.99$
    - 学習係数 $\alpha = 10^{-4}$

Comparing with Known Opponents

- 前の大会のチャンピオンbotであるF1とIntel Actを比較対象として実験する。
- Flat Map, Map01, Map02上で弱いbot15人と個人戦を行う。
  - Flat Map: 正方形位の小さい部屋で撃ち合う。武器はピストルのみ。
  - Map01: 少し広めの部屋で戦う。武器はロケットランチャーなどのみ。
  - Map02: 広い部屋で戦う。多様な武器が落ちている。
  - 弱いBotはゲーム内情報を参照している。
- 結果が以下の表

Table 3: Frags and death number of F1, InteAct and ours

	FlatMap		Map01		Map02	
	frags	death	frags	death	frags	death
F1	451	**9**	282	**110**	35	**56**
IntelAct	864	15	-86	164	134	105
Ours	**916**	23	**333**	117	**224**	84

- F1はMap01に過学習するように学習している
- Intel Actは強い武器を拾えるように学習しているため Map02に強い
- 提案モデルは地形や武器に依存しないため、汎用的な能力を持っている。
- F1はデス数は低いが、それは戦闘を避ける傾向があるためである。
- 提案モデルは積極的に戦ってfragを獲得していた。

読んでみて

- FPSエージェントは少しでも問題のサイズが大きくなると、途端に学習が難しくなる。
- そのため、計算機パワーに頼った手法が多かったのだが、これはスマートな手法であり興味深かった。
- しかし、それでも学習に時間はかかりそうではある。
- 実際に論文内に学習時間に関する技術は皆無であった。