修　　士　　論　　文

環境の特性を活用した方策学習の効率化
―中間フレームとリセット機能の利用―

指導教員　　鶴岡 慶雅 教授

東京大学大学院情報理工学系研究科
電子情報学専攻

氏 名　　48-206444 橋本大世

提 出 日　　2022 年 1 月 27 日

概 要

強化学習は様々な意思決定問題を扱うことのできる強力なフレームワークであるが, 実応用を困難にする課題がいくつか存在する. そのうちの一つがサンプル効率であり, 強化学習は試行錯誤しながら学習する性質上学習に多くのデータを必要とする傾向がある. サンプル効率を改善する試みはこれまで多くの研究で行われてきた. それらの大部分は強化学習の一般的な形式の上で効率的に問題を解くことを目指している. 一方, 実際に解こうとするタスクは学習の効率化に有用な環境の特性をもっていることもあり, そのような特性を効果的に活用することも重要である. そこで本研究では, 環境の特性の中でも特に中間フレームとリセット機能を題材とし, それらを有効活用するアルゴリズムを提案する. 中間フレームとは, エージェントが行動を取る間の観測データのことであり, 通常は学習に使用することができない. このデータを活用するために, 擬似的な行動という概念を使った手法を提案する. リセット機能とは, 学習に用いるシミュレータにおいて任意の状態にリセットする機能のことであり, 多くの場合利用可能である. リセット機能を活用し, 学習に有用なデータを重点的に集める手法を提案する. それぞれについて複数のタスクにおいて実験を行い, 提案手法によりサンプル効率が改善されることが確認された. ただし効果が限定的な場合もあり, その要因や改善点などについて考察した. また他の有用と予想される環境の特性に関する議論も行った.

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景

近年, 計算機の演算能力の向上, インターネット上で容易に集められるビッグデータ, 深層学習技術の発展などにより, 機械学習が大きな発展を遂げている. 現時点で実用化に成功している画像認識, 機械翻訳などの技術の多くは, 機械学習の中でも教師あり機械学習に分類される. 教師あり機械学習では一般的に, 入力と出力が組になったデータが与えられ, 入力から出力を予測するモデルを学習する.

本研究のテーマである強化学習も機械学習の一種である. 強化学習では, エージェントが環境との相互作用を繰り返し試行錯誤することで, タスクを実行できるようになる. その過程において行動の正解データは必要でない. 代わりに環境から報酬を受け取り, 一連の行動を通じて得られる報酬和を最大化するように学習する. すなわちタスクを報酬という形で記述できれば, 理論的にはそのタスクを実行するエージェントが得られるということである. これは教師あり学習と大きく異なる点であり, 強化学習は教師あり学習では扱えない問題も扱うことができる.

例えば囲碁 AI を学習することを考えてみよう. 教師あり学習の枠組みで考えると, 「良い手」のデータを大量に用意し, それを真似できるように機械学習モデルを訓練することになる. このアプローチは模倣学習と呼ばれ, 盛んに研究されている [1, 2]. 囲碁のようにメジャーなゲームであれば, 学習データ (棋譜) はある程度の量用意できると思われるが, それでも昨今の深層ニューラルネットワークの学習には膨大な量のデータが必要であり, その収集は容易ではない. また, データの質が学習の結果得られる AI の性能に直結するため, 良質なデータが求められる. さらに, AI が集めたデータ以上に良い手をさせるようになることは期待できない.

一方, 強化学習では外部からデータを与える必要はない. エージェントは自らのコピーと何度も対戦を繰り返し, 試行錯誤の中で段々と強くなっていく. 学習が人間の知識に依存していないため, 人間が知らなかったような戦法を見つける可能性もある. 実際に最高ランクのプロ棋士に勝利した囲碁 AI である Alpha Go [3] は, 新しい手筋を発見したり, それまで悪手だと考えられていた手が実は好手であることを見出したりと, 人間の囲碁に関する知見を広げ始めている [4].

昨今の機械学習の研究では深層学習が欠かせない存在となっているが, 強化学習も例外ではない. 強化学習では表現力の高い関数近似器として深層ニューラルネットワークを利用する深層強化学習が活発に研究されている. 強化学習においても他の分野と同じく, 深層学習により複雑な特徴量の設計が必須ではなくなった. 例えば AlphaGo の改良版である AlphaZero [5] では, 盤面を 19x19 の画像と見て畳み込みニューラルネットワークに入力するが, 特徴量の工夫はほとんど行われていない [1]. 他にも画像入力からビデオゲームを攻略する [6] など, 深層強化学習により柔軟にタスクを解くこと

[1] 実際には現在の盤面に加え, 過去 7 手分の盤面と現在の手番を表す値を入力する

ができるようになりつつある.

しかし, 実社会での強化学習の応用は, 教師あり学習の応用に比べて進んでいないのが現状である. その主な要因として, サンプル効率, 安全性, 汎化性能の 3 つが挙げられる.

サンプル効率とは, いかに少ないデータ量でタスクを学習できるかという指標である. 強化学習では基本的にエージェントが試行錯誤しながら学習を行うため, サンプル効率は教師あり学習と比べて低いと言われている. 学習データが大量に必要な深層ニューラルネットワークを使用する場合, その傾向はさらに強くなる. サンプル効率が低いと, 学習に多大なコストがかかったり, 現実的な時間内で学習ができないということになってしまう. そのため, これまで数多くの研究がサンプル効率の向上に取り組んできた [7, 8, 9, 10]. また, シミュレータを使って学習を効率化させることも広く行われている. 例えばロボットを操作するタスクの場合, 実際のロボットを動かしてデータを収集するのは時間がかかるが, シミュレータを使うことで大幅にスピードを高められる.

強化学習ではエージェントが試行錯誤する際, 基本的には一切の制約を置かない. つまり, エージェントが致命的な失敗を犯すこともありうる. 例えば自動運転を強化学習で構築することを考えると, エージェントは学習中に何度も事故を起こす可能性がある. これまでに安全性をできるだけ高めるアルゴリズムも数多く提案されてきた [11, 12]. ただし, 詳細な事前知識がある場合を除いては, 一度も失敗せずに学習するのは原理的に不可能である. そのためエージェントが自らデータを収集する場合は, 安全性の観点においてもシミュレータを使った学習が現実的である.

強化学習では学習時とテスト時に同じ環境を用いることが多い. そのため強化学習アルゴリズムは学習環境に著しく過適合していると指摘されている [13]. 教師あり学習では大量の学習データを使うことで高い汎化性能が得られることが知られており, それが実応用を支えているとも言える. 強化学習においても, 学習環境と構造が似ている環境には汎化できる方策が実応用に向けて必要であると考えられる. この課題に取り組むため, これまでに強化学習アルゴリズムの汎化性能を測るベンチマークがいくつか開発されてきた [14, 15]. また汎化性能を高める手法も多数提案されており, 汎化しやすい状態表現を学習する方法 [16, 17] や, 多様な環境パラメータにおいて学習することでロバストな方策を取得する方法 [18, 19, 20] などがある. 最近の研究では理論的な分析も進んでおり, 有限な学習データのもとで汎化するには, 完全観測可能な環境における方策も部分観測可能性に対応できるアルゴリズムで学習することが必要であるとの指摘もある [21].

強化関連が実社会の問題解決に利用できるようにするためには, これらの問題を解決するための研究が必要である. 本研究では特にサンプル効率に注目し, 2 つのアプローチから効率改善を目指した.

1.2 本研究の目的

これまで, 強化学習の一般的な形式において方策学習の効率を改善する取り組みがなされてきた. 一方で実際のタスクにおいては, 一般的な形式には含まれない環境固有の特性を利用できることがある. 本研究では, そういった環境の特性を有効活用することにより, 方策学習を効率化することを目指す. 学習に利用できる特性には様々なものが考えられるが, ここでは特に広く利用可能と考えられるものとして, 中間フレームとリセット機能を取り上げる.

中間フレームとは, エージェントが行動を取る間の観測データのことであり, 通常は利用されていない. これには中間フレームが一般的な形式に含まれていないという理由だけでなく, 後述するように利用には技術的な問題があるという理由もある. 本研究では, この問題に対するシンプルな手法を提案し, 中間フレームの有効活用による方策学習の効率化に取り組んだ.

リセット機能とは, シミュレータにおいて任意の状態にリセットできる機能のことであり, 多くのシミュレータで利用可能である. リセット機能を活用する研究はこれまでにいくつか存在するが, エキスパートデータが必要であったり, 特定のタスクに特化しているため汎用性が低かったりと課題があった. 本研究では, リセット機能を活用する汎用性の高い手法を提案する.

1.3 本研究の構成

本論文の以降の構成は以下のようになっている:

第 2 章 中間フレームの活用

この章では, 中間フレームを活用した方策学習の効率化に取り組んだ研究について記述する.

第 3 章 リセット機能の活用

この章では, リセット機能を活用した方策学習の効率化に取り組んだ研究について記述する.

第 4 章 おわりに

最後に本研究のまとめを行い, 今後の課題について述べる.

第2章　中間フレームの活用

2.1　導入

強化学習を実社会において利用できるようにするためには, 少ないデータから安定して学習する手法が必要不可欠である. アルゴリズムの改良によりこの問題に取り組んだ研究は数多くあるが, それらの多くは収集したデータをいかにうまく活用するかという点に主眼を置いている [22, 23, 24]. 一方 Kostrikov らは, 一般的にコンピュータビジョンなどの分野で用いられているデータ拡張により, 画像から効率よく方策を学習できることを示した [9]. この研究により, 深層強化学習は他の分野と同様に過学習による悪影響を受けやすく, またデータを増やすことで過学習を軽減できることがわかった. この発見に動機づけられ, 我々はエージェントの学習データ量を増やす方法を探し, action repeat という深層強化学習で広く用いられている技法に着目した.

強化学習の一般的な設定として, エージェントが一旦行動を選択するとその行動は固定された回数繰り返される. 行動を繰り返すこの技法は action repeat と呼ばれ, 学習される方策の質に大きく影響することが知られている [25, 26]. 強化学習における離散制御タスクの代表的なベンチマークである Atari 2600 [27] では, 繰り返しの回数を 4 に設定することが多い [27, 28]. 連続制御タスクの代表的なベンチマークである DeepMind control suite [29] では環境ごとに異なる値が使われることが多いが, 2 から 4 程度が一般的である [30, 31].

action repeat は学習の安定化 [32] や探索の促進 [33] などのために広く用いられている. しかし, action repeat の問題点として, 行動を繰り返す間のデータは十分に活用されていないことが挙げられる. DeepMind control suite では通常 action repeat 間のデータは全く使用されていない. Atari 2600 では action repeat 間の最後の 2 フレームの最大値プーリングが使用されるが, これは偶数フレーム, 奇数フレームしか現れないオブジェクトを見落とさないようにするためであり, 十分活用されているとは言い難い. 環境のステップ数が一定である場合, 学習データ量は action repeat の長さに反比例するため, action repeat が長い場合に非常に多くのデータが捨てられてしまう. そこで本研究では, action repeat 間の状態遷移から得られる情報を有効活用することを目的とする.

提案手法は, 行動の平均値で表される擬似的な行動が action repeat の回数だけ繰り返されたと想定することで, 行動を繰り返す間のデータを利用可能にする. この手法は連続行動空間には直接適用できるが, 離散行動空間では意味のある行動の平均値を得るために工夫が必要である. そこで我々は, 行動の埋め込み表現を学習してその平均を用いるという方法を考案した.

本章における貢献は次の 3 点である. (1) action repeat 間の遷移データを利用することで, 連続制御タスクでデータ量を増やす手法を提案する. (2) 行動の埋め込み表現を学習することで, 提案手法を離散制御タスクに拡張する. (3) 提案手法と Deep Q-Network [6], Soft Actor-Critic [24, 34] を組

み合わせることで, いくつかの連続制御タスク, 離散制御タスクでパフォーマンスを向上させた.

2.2 背景

2.2.1 強化学習

1.1 節で述べたように, 強化学習ではエージェントが環境と相互作用しながら学習が進行する. その様子を表したのが図 2.1 である. 時刻 t においてエージェントは環境から状態 s_t を受け取り, 行動 a_t を返す. 状態から行動への写像は方策と呼ばれ, エージェントは方策を最適化する. ここで, 方策は $\pi(a|s)$ と表記する.

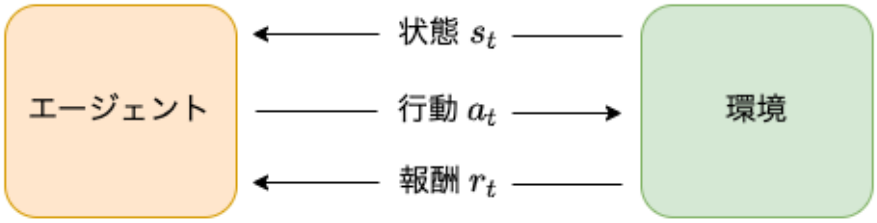


図 2.1: 強化学習におけるエージェントと環境の相互作用

正式には, 環境はマルコフ決定過程 (Markov Decision Process, MDP) [35] で表現される. MDP は以下の要素から定義される.

- 状態空間 $\mathcal{S}$
- 行動空間 $\mathcal{A}$
- 初期状態分布 $\rho_0 : \mathcal{S} \to [0, 1]$ [2]
- 状態遷移確率 $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$
- 報酬関数 $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
- 割引率 $\gamma \in [0, 1)$

これらの表記を用いると, エージェントと環境の相互作用はより具体的に記述できる.
エピソードは初期状態分布からサンプルされた初期状態から開始する.

$$s_0 \sim \rho_0(\cdot)$$

[2] ここでは離散状態空間の場合の値域を示している. 連続状態空間の場合, 値域は $[0, \infty)$ となる. 次の遷移確率についても同様.

エージェントは初期状態を受け取り, 方策に従って行動を選択する.

$$a_0 \sim \pi(\cdot|s_0)$$

状態と選択した行動に基づいて次状態に遷移し, 状態, 行動, 次状態から報酬が決定する.

$$s_1 \sim p(\cdot|s_0, a_0)$$

$$r_0 = r(s_0, a_0, s_1)$$

次の時刻に移り, エピソードが終了するまで以上を繰り返す.

以上のようにして方策 π から誘導される分布に関する期待値を $\mathbb{E}_\pi[\cdot]$ と書くこととする. 強化学習の目的は, 割引累積報酬 $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$ を最大化する方策 $\pi(a_t|s_t)$ を学習することである. ここで累積報酬を割引率 γ で割り引いているのは, 報酬をなるべく早く得る方策を学習するため, および学習を安定させるためである.

2.2.2 on-policy 学習と off-policy 学習

これまで数多くの強化学習アルゴリズムが提案されてきたが, それらは大きく on-policy 学習と off-policy 学習に分類される. それぞれの実装例の擬似コードを Algorithm 1, 2 に示す.

on-policy 学習では, 方策の更新に使うデータが現在の方策で集められるデータと同分布である必要がある. そのため, 基本的には現在の方策で学習データを収集し, それを使って方策を 1 度更新する. 過去の方策で収集したデータは分布が異なるためそのまま用いることはできず, 結果的に必要なサンプル数が多くなってしまう.[3] すなわち, on-policy 学習はサンプル効率が低い傾向にある. 一方, on-policy 学習は安定性が高いことや, 非定常な環境に対応しやすいことが知られている. 代表的なアルゴリズムには, 方策勾配法に工夫を加えた Trust Region Policy Optimization (TRPO) [36] や Proximal Policy Optimization (PPO) [23] などが挙げられる.

off-policy アルゴリズムでは学習データの分布に強い条件はなく, 過去の方策で収集したデータも利用することができる. そのためサンプル効率が高い傾向にある. しかし, off-policy 学習は on-policy 学習に比べて不安定であることが多い. 特に関数近似, ブートストラッピング, off-policy 学習の組み合わせは deadly triad と呼ばれ, これが現れるアルゴリズムは安定した学習が難しいことが知られている [37, 38]. 代表的なアルゴリズムには, Deep Q-Network [6] や Soft Actor-Critic [24, 34] などが挙げられる. これらは本稿の実験で使用しているため, それぞれ概要を後述する.

2.2.3 動的計画法

方策 π のもとでの行動価値関数は以下のように定義される.

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k} \;\middle|\; s_t = s, a_t = a\right]$$

[3] 重点サンプリングなどの方法で分布の異なるデータを利用することもできるが, 安定した尤度比の推定が難しい. そのためサンプル効率が重要な場合には off-policy 学習のアルゴリズムを使うことが一般的である.

Algorithm 1 on-policy 学習の例

Input: loss function of an algorithm L_θ, number of steps per update N
Output: parameters θ

Initialize parameters θ randomly
while not converged **do**
  Initialize a buffer $\mathcal{D} = \{\}$
  Reset to an initial state s_0
  while length($\mathcal{D}$) $< N$ **do**
    for $t = 0$ to ∞ **do**
      Take an action $a_t \sim \pi_\theta(\cdot|s_t)$
      Observe a next state s_{t+1} and a reward r_t
      Append a transition (s_t, a_t, r_t, s_{t+1}) to $\mathcal{D}$
      if episode is done **then**
        break
      end if
    end for
  end while
  Compute a loss function $L_\theta(\mathcal{D})$
  Update parameters θ using $\nabla L_\theta(\mathcal{D})$
end while
return θ

すなわち $Q^\pi(s,a)$ は, 状態 s で行動 a を取り, その後方策 π に従って行動したときの割引累積報酬の期待値である. 行動価値, 行動価値関数はそれぞれ Q 値, Q 関数とも呼ばれる.

ここで, 式変形により次の再帰的関係式が得られる.

$$
\begin{aligned}
Q^\pi(s,a) &= \mathop{\mathbb{E}}_{\pi}\left[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \;\middle|\; s_t = s, a_t = a \right] \\
&= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \;\middle|\; s_{t+1} = s', a_{t+1} = a' \right] \right) \\
&= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a') \right)
\end{aligned}
$$

この関係式は, 行動価値関数に関するベルマン方程式として知られている.

ここで, ベルマン作用素 $\mathrm{B}_\pi(f)$ を以下のように定義する.

$$
(\mathrm{B}_\pi f)(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a) \left(r(s,a,s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') f(s',a') \right)
$$

Algorithm 2 off-policy 学習の例

Input: loss function of an algorithm L_θ
Output: parameters θ

Initialize a replay buffer $\mathcal{D} = \{\}$
Initialize parameters θ randomly
while not converged **do**
    Reset to an initial state s_0
    for $t = 0$ to ∞ **do**
        Take an action $a_t \sim \pi_\theta(\cdot|s_t)$
        Observe a next state s_{t+1} and a reward r_t
        Append a transition (s_t, a_t, r_t, s_{t+1}) to $\mathcal{D}$
        Sample a min-batch $\mathcal{B}$ from $\mathcal{D}$
        Compute a loss function $L_\theta(\mathcal{B})$
        Update parameters θ using $\nabla L_\theta(\mathcal{B})$
        if episode is done **then**
            break
        end if
    end for
end while
return $\mathcal{B}$

ベルマン作用素を用いると, ベルマン方程式は

$$Q^\pi(s,a) = (\mathrm{B}_\pi Q^\pi)(s,a)$$

と書ける. すなわち Q^π はベルマン作用素の不動点である.

ベルマン作用素は縮小写像であり, かつ唯一の不動点を持つことが知られている. この性質により, 任意の推定値 $\hat{Q}_0(s,a)$ に B_π を繰り返し適用することで, 不動点 $Q^\pi(s,a)$ を得ることができる.

$$\lim_{k\to\infty} \mathrm{B}_\pi^k \hat{Q}_0(s,a) = Q^\pi(s,a)$$

以上のようにして方策 π を評価することができる. 一方, 最終的に求めたいのは割引累積報酬を最大化する最適方策 π^* である. π^* に対する行動価値関数を $Q^*(s,a)$ とおくと, ベルマン方程式に類似したベルマン最適方程式が成り立つ.

$$Q^*(s,a) = (\mathrm{B}_* Q^*)(s,a)$$

$$(\mathrm{B}_* f)(s,a) = \sum_{s'\in\mathcal{S}} p(s'|s,a)\left(r(s,a,s') + \gamma \max_{a'\in\mathcal{A}} f(s',a')\right)$$

B_* についても B_π と同様に繰り返し適用により不動点を求めることができる.

$$\lim_{k\to\infty} \mathrm{B}_*^k \hat{Q}_0(s,a) = Q^*(s,a)$$

$Q^*(s,a)$ が得られれば, 最適方策 π^* は以下のように求められる.

$$\pi(a|s) = \begin{cases} 1 & (\text{if } a = \arg\max_a \hat{Q}(s,a)) \\ 0 & (\text{otherwise}) \end{cases} \tag{2.1}$$

2.2.4 Q 学習

動的計画法を実行するには, 環境に関する完全な知識が必要である. しかし一般的に状態遷移確率 $p(s'|s,a)$, 報酬関数 $r(s,a,s')$ は未知であるため, これらを使わずに学習する必要がある.

Q 学習 [39] は代表的な強化学習アルゴリズムの一つであり, 様々なアルゴリズムの基礎となっている. まず, 各状態行動対に対する最適行動価値関数 (Q 値) を保持する Q テーブル $\hat{Q}(s,a)$ を用意する. 状態 s_t において行動 a_t を取り報酬 r_t を得て状態 s_{t+1} に遷移したとき, 次の値 y は $(\mathrm{B}_*Q)(s,a)$ の推定値となる.

$$y = r_t + \gamma \max_{a\in\mathcal{A}} \hat{Q}(s,a)$$

そこで, Q テーブルを次のように更新する.

$$\hat{Q}(s,a) \leftarrow \alpha_t y + (1-\alpha_t)\hat{Q}(s,a) = \hat{Q}(s,a) + \alpha_t(y - \hat{Q}(s,a)) \tag{2.2}$$

α_t は学習率であり, 学習率が一定の条件を満たすとき, 上の更新則により正しい Q 値が得られることが知られている.

なお, 行動を選択する際には基本的には式 2.1 のように Q 値の推定値が高い行動を選ぶが, それだけでは探索が進まない. そのため, 以下のように確率的にランダムな行動を選ぶ ε-greedy 法などが広く用いられている.

$$a = \begin{cases} \text{random action} & (\text{with probability } \varepsilon) \\ \arg\max_{a'} \hat{Q}(s,a') & (\text{with probability } 1-\varepsilon) \end{cases}$$

2.2.5 Deep Q-Network

Q 学習は Q テーブルを用意することから明らかなように, 離散かつ有限な状態空間および行動空間を扱う手法である. 一方, 現実的に解きたいタスクでは連続な状態空間を扱う場合も多い.

Deep Q-Network (DQN) は Q 値の推定に深層ニューラルネットワークを用いる手法である. Q テーブルを明示的に表現するのではなく, ニューラルネットワークのパラメータ θ によって $Q_\theta(s,a)$

と表現することで, 有限でない状態空間も扱うことができる. また, 深層学習の技術により特徴量抽出も自動で行われる.

Q 学習の更新則 (式 2.2) は, $\hat{Q}(s,a)$ を y に近づけていると解釈することができる. そこで DQN では, 次式で表される損失関数を用いてパラメータ θ を学習する.

$$L_\theta(\mathcal{B}) = \mathbb{E}_{s,a,s'\sim\mathcal{B}}\left[(y - Q_\theta(s,a))^2\right]$$

$$y = r + \gamma Q_{\theta^-}(s', a^*)$$

$$a^* = \arg\max_{a'\in\mathcal{A}} Q_\theta(s', a')$$

ここで, $\mathcal{B}$ は遷移データ (s,a,r,s') のミニバッチである.

目標値 y の計算には最新のパラメータ θ ではなく固定された古いパラメータを使ったパラメータ θ^- を用いている. これは, Q_θ が絶えず変化するため目標値として使うと最適化が難しく, その問題を回避するためである. θ^- は一定のステップ間隔で θ と同期される.

また, y の計算では行動 a^* の選択を Q_θ により行い, 価値の評価を Q_{θ^-} で行っている. これにより Q 値の過大推定を抑制できることが知られている. このように行動選択と価値評価を 2 つの推定器により行う方法は Double DQN [40] と呼ばれる.

2.2.6 Soft Actor-Critic

行動空間が連続の場合には DQN における arg max が計算できないため, 異なるアルゴリズムが必要になる. 一般的にそれらの方法では, Q 関数に加えて方策もニューラルネットワークにより表現する.

Soft Actor-Critic (SAC) は連続制御タスクで広く用いられているアルゴリズムである. SAC の特徴は, 報酬に加えて方策のエントロピーも最大化することである. これにより探索を促進することができ, さらにロバストな方策が得られることが知られている [41].

SAC では Q ネットワーク $Q_\theta(s,a)$ に加え, 方策ネットワーク $\pi_\phi(a|s)$ を学習する. 方策は正規分布 $\mathcal{N}(\mu_\phi(s), \sigma_\phi(s))$ を tanh で変形した確率分布として表される. 具体的には, 方策から行動をサンプルする際は以下のような計算を行う.

$$\varepsilon \sim \mathcal{N}(0, I)$$

$$a = \tanh(\mu_\phi(s) + \sigma_\phi(s) \odot \varepsilon)$$

ここで, $\odot$ は要素ごとの積を表す.

Q ネットワークは以下の損失関数を最小化することで学習される.

$$L_\theta(\mathcal{B}) = \mathop{\mathbb{E}}_{,a,s'\sim\mathcal{B}}\left[(y - Q_\theta(s,a))^2\right]$$

$$y = r + \gamma \mathop{\mathbb{E}}_{a'\sim\pi(\cdot|s')}\left[Q_{\theta^-}(s',a') - \alpha \log \pi_\phi(a'|s')\right]$$

ここで, $\mathcal{B}$ は遷移データのミニバッチ, α は温度パラメータ, θ^- はパラメータ θ の指数移動平均である. 目標値 y に注目すると, DQN の場合にはなかった $-\alpha\,\mathbb{E}_{a'}\left[\log \pi_\phi(a'|s')\right]$ という項が追加されている. これは方策に関するエントロピーを表している.

方策は $-Q_\theta(s,a)$ をエネルギーとするボルツマン分布に近づけるため, 以下の損失関数を最小化するように学習する.

$$L_\phi(\mathcal{B}) = \mathbb{E}_{s\sim\mathcal{B}}\left[D_{\mathrm{KL}}\left(\pi_\phi(\cdot|s)\,\middle|\middle|\,\frac{\exp(\frac{1}{\alpha}Q_\theta(s,\cdot))}{Z(s)}\right)\right]$$

ここで D_{KL} は KL ダイバージェンス, $Z(s)$ は正規化項である. $Z(s)$ は直接計算することはできないが, 上式の KL ダイバージェンスの項を整理すると

$$\int \pi_\phi(a|s)\log\frac{\pi_\phi(a|s)Z(s)}{\exp(\frac{1}{\alpha}Q_\theta(s,a))}\,da = \int \pi_\phi(a|s)\log\left(\pi_\phi(a|s) - \frac{1}{\alpha}Q_\theta(s,a)\right) + \log Z(s)$$

となり, $\nabla_\theta L_\phi$ に $Z(s)$ は現れないため無視して構わない.

最後に, 温度パラメータ α はエントロピーが目標値 $\bar{H}$ に近づくように調整される.

$$L_\alpha(\mathcal{B}) = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{B}\\ a\sim\pi_\phi(\cdot|s)}}\left[-\alpha\left(\log\pi_\phi(a|s) + \bar{H}\right)\right]$$

エントロピーの目標値 $\bar{H}$ は $-|\mathcal{A}|$ に設定されることが一般的である.

2.2.7 action repeat

環境から見たステップを環境ステップ, エージェントから見たステップをエージェントステップと呼ぶこととする. action repeat の長さを T とすると, 1 エージェントステップは T 環境ステップに対応し, エージェントは状態 $s_0, s_T, s_{2T}, \ldots$ において行動 $a_0, a_T, a_{2T}, \ldots$ を選択する. それら以外の環境ステップでの行動は, 直近に選んだ行動が繰り返される. すなわち, $a_0 = a_1 = \ldots = a_{T-1}$, $a_T = a_{T+1} = \ldots = a_{2T-1}$, $\ldots$ となる. 図 2.2 に示すように, 従来の強化学習において学習には $s_0, s_T, s_{2T}, \ldots$ のみが用いられ, その間のデータは利用されない. 以降では, 後者のデータのことを中間フレームと呼ぶ. 図 2.2 の例では, s_1 と s_3 の間では一貫した行動が取られていないため, この遷移を学習データとして用いることはできない.

action repeat にはいくつかの利点がある.

学習の安定化

同じ行動を繰り返すことにより, 状態遷移に選ばれた行動の影響がより強く現れることになる. すると, 各行動に対する Q 値の差が大きくなる. この差は action-gap [32] と呼ばれ, action-gap を増大させることは安定的な学習につながることが知られている [42]. なぜなら学習中は Q 値の推定が不正確であり, 行動ごとの違いが明確な方が行動を正しく順位付けしやすいためである. 行動の順位付けさえ正しければ, 正確な Q 値の予測ができなくても最適な行動を取ることができる.

探索の促進

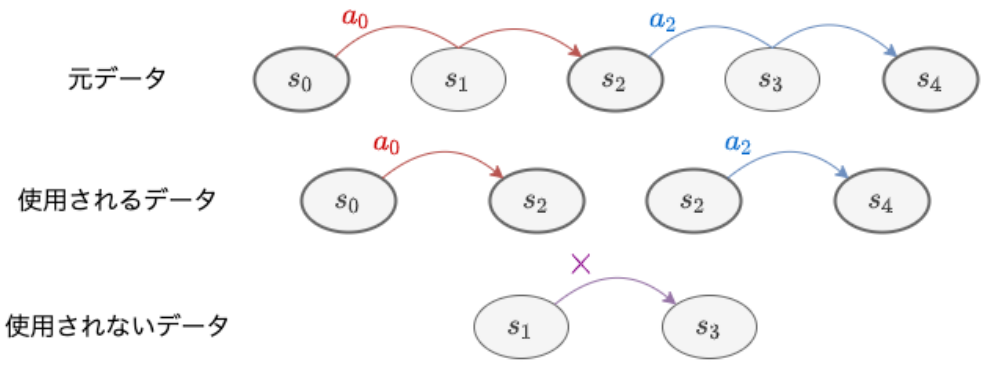


図 2.2: action repeat と学習データの関係. 図は $T = 2$ の場合.

長い action repeat は探索を促進することが期待できる [33]. 例えば迷路を探索するようなゲームの場合, 同じ行動を何度も繰り返して長く進むことで, それまで到達できていなかった状態にたどりつけることがある. もちろん同じ行動を繰り返すだけでは十分に探索できない場合もあるが, このヒューリスティクスは多くの環境で有効であると考えられる.

計算コストの削減

エージェントが行動選択する頻度を減らすことで, 計算コストを削減することができる [33]. 計算コストは, 深層強化学習を実応用するにあたって特に重要な点となりうる. 深層強化学習はニューラルネットワークによる推論にある程度時間がかかるため, リアルタイム性を確保するために action repeat を長く設定せざるを得ない場合も考えられる. また強化学習エージェントが人間に対して行動の案を提案するという利用方法の場合, 実際に操作するのは人間であり, コンピュータのように素早く行動を切り替えることはできない. そのため, エージェントが行動を選択する時間幅を長く設定する必要がある.

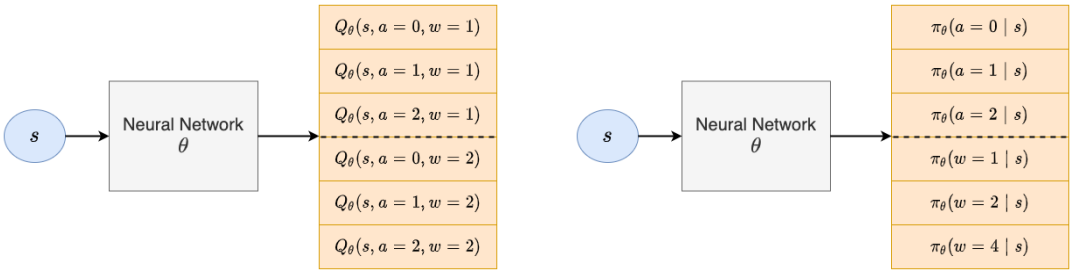
2.3 関連研究

action repeat 間のデータを利用する研究は我々の知る限りでは存在しない. そのため, ここではより広く action repeat に着目した研究について述べる.

action repeat の値は通常ハイパーパラメータとして設定され, その値はスコアに大きく影響する. Braylan らは, 強化学習のベンチマークとして広く用いられている Atari において, タスクごとに適切な action repeat を設定することが非常に重要であることを実験的に示した [26]. 例えば Seaquest というゲーム では 180 という非常に大きな値の場合に最も高い性能が得られた. 別の代表的なベンチマークである DeepMind control suite においても, タスクによって高いパフォーマンスを得やすい action repeat が異なることが経験的に知られている. そのため, タスクごとに異なる action repeat を選んでいる Planet Benchmark [30] よりも, action repeat を固定している Dreamer Benchmark [25] の方が難易度が高いとされている [9].

環境ごとに最適な action repeat を探すには, 実験を何度も繰り返す必要があり手間がかかる. また一つの環境に着目しても, どの程度行動を繰り返すべきかは状態や学習の進行度合いに応じて変化する可能性がある. Lakshminarayanan らは action repeat をハイパーパラメータとして扱うではなく動的に決定できるアルゴリズム Dynamic Frameskip DQN (DFDQN) を提案した [43]. DFDQN は DQN をベースにしており, DQN と同様に Q ネットワークを学習する. この手法の特徴は, 図 2.3a に示すように複数の action repeat に対する Q 値を予測する点である. Atari に含まれるいくつかのゲームにおいて DFDQN は DQN よりも高いスコアを獲得できることが確認されている.

しかしこの手法では, 行動が数 $|\mathcal{A}|$, 行動の繰り返し回数の候補数が $|\mathcal{W}|$ のとき, $|\mathcal{A}| \times |\mathcal{W}|$ 個の行動を扱う必要がある. そのため行動の繰り返し回数の候補を増やしたいときには適用が難しい. そこで Sharma らは DFDQN を改良したフレームワーク Fine Grained Action Repetition (FiGAR) [44] を提案した. FiGAR では方策ネットワークに行動と繰り返し回数を出力させる. これにより, 扱う行動の数を $|\mathcal{A}| + |\mathcal{W}|$ に抑えることができる. また, FiGAR は方策ネットワークを持つ任意の強化学習アルゴリズムを拡張できるフレームワークであり, 論文では A3C [45], TRPO [36], DDPG [46] の拡張について詳しく記述されている. この手法は Atari 2600, MuJoCo [47] などの環境でパフォーマンスを向上させることが実証されている.



(a) DFDQN の Q ネットワーク **(b)** FiGAR の方策ネットワーク

図 2.3: DFDQN と FiGAR のネットワーク構成の比較. 図中の w は行動の繰り返し回数を表す.

2.4 提案手法

2.2 節で述べたように, 一般的な学習では $s_0, s_T, s_{2T}, \ldots$ のみが用いられ, その間のデータは利用されない. 本研究では, 通常使われない中間フレームを活用して学習効率の改善を目指す.

2.4.1 概要

DQN や SAC など多くのオフポリシー強化学習手法では Q 関数のフィッティングが必要である. 提案手法は, Q 関数の学習に中間フレームを活用する.

Q 関数の学習に用いるデータは $(s_{kT+l}, a_{kT+l}, \sum_{t=0}^{T-1} r_{kT+l+j}, s_{kT+l+T})$ と表される. ここで, $k, l \in \mathbb{N}, 0 \leq l < T$ である. 具体的な学習の方法は 2.2.5 を参照. 上述したように, 中間フレームから始ま

る遷移データ, すなわち $l > 0$ のデータはそのままでは学習に利用できない. なぜなら, 状態 s_{kT+l} から s_{kT+l+T} まで行動 $a_{kT+l}(= a_{kT})$ が T 回繰り返されるわけではないからである. エージェントは $t = kT + T$ において新しい行動 a_{kT+T} を選択するため, $a_{kT} = a_{kT+T}$ でない限り s_{kT+l} からは行動 a_{kT} は $T - l$ 回しか繰り返されない.

そこで提案手法では, 図 2.4 に示すように状態 s_{kT+l} から擬似的な行動 $\hat{a}_{kT+l}$ が T 回繰り返されたと解釈し直す. これにより, 擬似的な遷移データ $(s_{kT+l}, \hat{a}_{kT+l}, \sum_{t=0}^{T-1} r_{kT+l+j}, s_{kT+l+T})$ を学習に用いることができるようになる. 本手法では, 擬似的な行動をどのように定義するかが重要な点である.

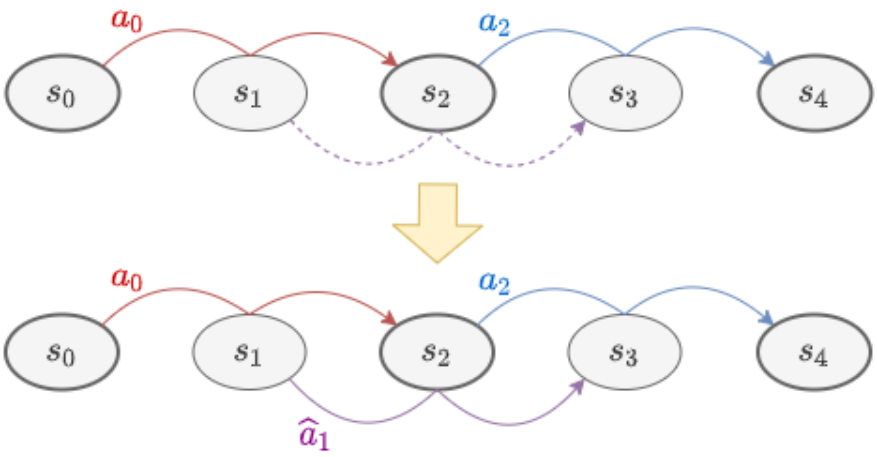


図 2.4: 提案手法の概要. 図は $T = 2, k = 0, l = 1$ の場合.

以降は, 行動空間が連続と離散の場合に分けて詳細に説明する.

2.4.2 連続行動

擬似的な行動を定める単純な方法として, 行動 $a_{kT+l}, a_{kT+l+1}, \ldots, a_{kT+l+T-1}$ の平均を用いる.

$$\hat{a}_{kT+l} = \frac{1}{T} \sum_{t=0}^{T-1} a_{kT+l+t} \tag{2.3}$$

環境のダイナミクスが局所的に線形近似できる場合, 行動の平均を用いることによって正規のデータと擬似的なデータを同様に扱うことができる. 詳細は 2.8.1 小節にて後述する.

2.4.3 離散行動

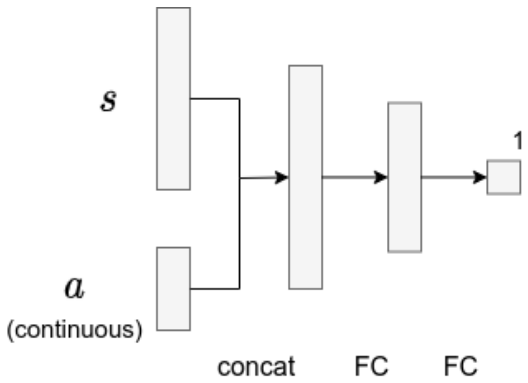
行動が離散的な場合, 単純に平均を取ることはできない. そこで, Q ネットワークの構成を変えることにより連続行動と同様に扱えるようにする.

一般的に連続行動, 離散行動それぞれに対する Q 関数は図 2.5a, 2.5b のようなネットワークで実装される. 本研究では画像入力を扱うが, 図を簡略化するため畳み込み層は省略した. 図のように, 連続行動では入力ごとに 1 つの Q 値が出力されるのに対し, 離散行動では全行動に関する Q 値が出力される.

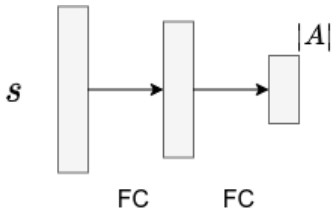
離散行動のネットワークを連続行動のネットワークに近づけるため, 図 2.5c のような構成に変更する. 行動は埋め込み表現に変換され, 1 つの Q 値が出力される. この埋め込み表現は, 目的関数を最適化する中で Q ネットワークのパラメータとともに学習される. これにより, 埋め込み表現の平均を擬似的な行動として用いることができる. つまり, 行動 a の埋め込み表現を $e_\theta(a)$ とすると, 擬似的な行動 $\hat{a}_{kT+l}$ の埋め込み表現 $e_\theta(\hat{a}_{kT+l})$ は

$$e_\theta(\hat{a}_{kT+l}) = \frac{1}{T}\sum_{t=0}^{T-1} e_\theta(a_{kT+l+t})$$

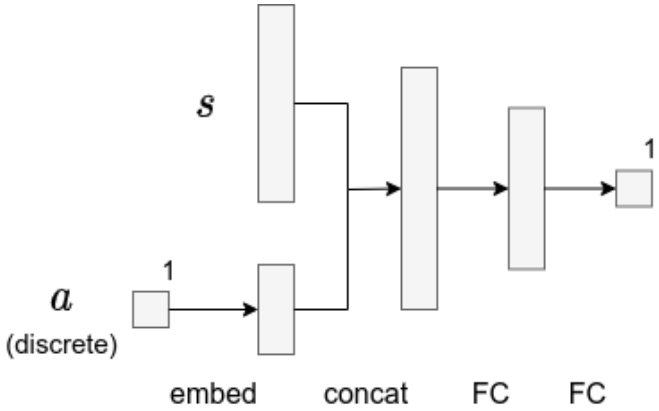
と表される.



(a) 連続行動の Q ネットワーク



(b) 離散行動の Q ネットワーク



(c) 提案手法における離散行動の Q ネットワーク

図 2.5: Q ネットワークの構成図. ”concat”は結合 (concatenate), ”FC”は全結合層 (fully connected), ”embed”は埋め込み (embedding) を表している.

Algorithm 3 Q ネットワーク学習用のミニバッチ作成

Input: replay buffer $\mathcal{D}$, mini-batch size N, action repeat parameter T
Output: mini-batch $\mathcal{B}$

Initialize $\mathcal{B} = \{\}$
for $k = 1$ to N **do**
  Sample $(s_t, a_t, r_t, \ldots, s_{t+T-1}, a_{t+T-1}, r_{t+T-1}, s_{t+T}) \sim \mathcal{D}$
  if discrete action **then**
    $e_t = e_\theta(a_t), \ldots, e_{t+T-1} = e_\theta(a_{t+T-1})$
  end if
  Current state $s = s_t$
  Next state $s' = s_{t+T}$
  Reward $r = \sum_{l=0}^{T-1} r_{t+l}$
  if continuous action **then**
    Pseudo action $\hat{a} = \frac{1}{T} \sum_{l=0}^{T-1} a_{t+l}$
    Add $(s, \hat{a}, r, s')$ to $\mathcal{B}$
  else if discrete action **then**
    Pseudo action embedding $\hat{e} = \frac{1}{T} \sum_{l=0}^{T-1} e_{t+l}$
    Add $(s, \hat{e}, r, s')$ to $\mathcal{B}$
  end if
end for
return $\mathcal{B}$

2.4.4 実装

提案手法は正規のデータに追加して擬似的なデータを用いる．それ以外は一般的な学習と同様であるため, 容易に実装できる．Q ネットワークを学習するためのミニバッチを作成する擬似コードを Algorithm 3 に示す．擬似コード中の $e_\theta(\cdot)$ は, 離散行動を受け取り対応する埋め込み表現を返す関数である．

我々は, 正規のデータと擬似的なデータを平等に扱うことにした．すなわち, サンプルされる遷移データのインデックス t は一様に選ばれるものとした．これにより, 正規データ ($l = 0$) と擬似的なデータ ($0 < l < T$) の量の比は $1 : T - 1$ となる．一方, 正規データのみを使う場合は通常の学習になる．いずれの場合もミニバッチサイズ N は同じ値にした．

擬似的なデータは近似の上で得られるため, 学習に悪影響を及ぼす可能性も考えられる．そのため, 正規のデータを重視するよう重み付けをすることが有益かもしれない．これは今後の研究課題とする．

2.5 実験

OpenAI Gym に含まれる以下の 4 つの環境において, 提案手法を通常の学習と比較する．

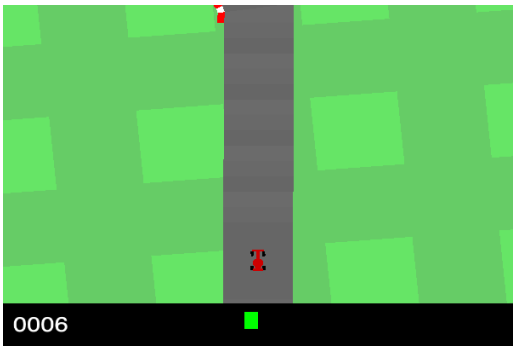
Pendulum 振り子を左右に回して直立させるタスク．振り子はランダムな位置に初期化される．行動は連続であり，次元数は 1.

CarRacing トラックに沿って車を運転するタスク．トラック上のタイルを多く踏むと高いスコアが得られる．行動は連続であり，次元数は 3.

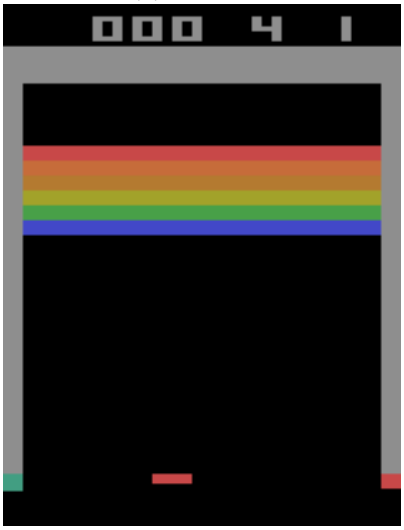
Breakout ブロック崩しのビデオゲーム．バーを左右に動かし，ボールを打ち返してブロックを崩す．行動は離散であり，取りうる行動の種類数は 4.

Freeway 車が行き交う道路にて，鳥を道路の向こう側まで渡らせるというビデオゲーム．鳥を前後に動かし，車を避けながら向こう側を目指す．行動は離散であり，取りうる行動の種類数は 3.

(a) Pendulum



(b) CarRacing



(c) Breakout

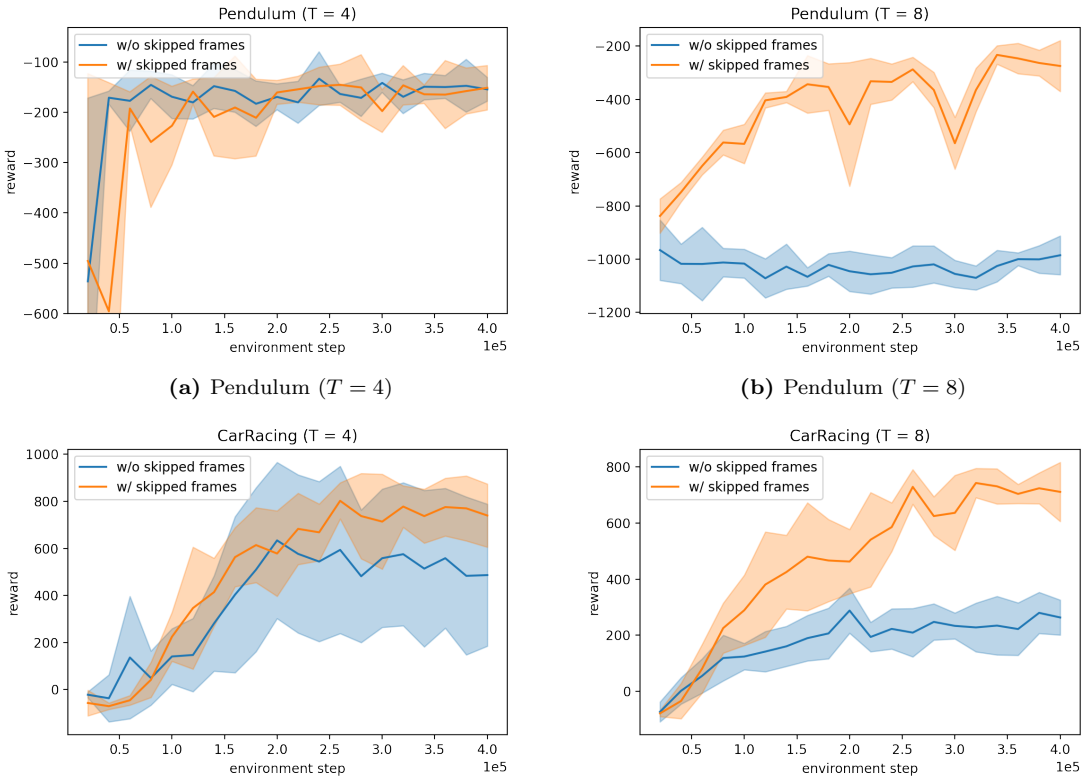
(d) Freeway

図 2.6: 各環境のサンプル画像．実際にはこれらの画像をグレースケールに変換し，サイズを縮小してから使用する．詳細は 2.8.3 小節を参照．

すべての環境で観測は画像であり, 直近 4 フレームをスタックしたものが状態として用いられた. 学習の環境ステップ数はいずれの場合も 400k に固定した. すべての実験は 5 個のランダムシードで行った. モデルの構成やハイパーパラメータなどの詳細は 2.8 節で述べる.

2.5.1 連続制御タスク

連続制御タスクにおける提案手法の有用性を検証するため, Pendulum, CarRacing で実験を行った. 強化学習手法には SAC を用いた. SAC では Q ネットワークに加えて方策ネットワークも学習するが, 比較を平等にするため, 方策ネットワークの学習に action repeat 間のデータは用いていない.



(a) Pendulum ($T = 4$) **(b)** Pendulum ($T = 8$)

(c) CarRacing ($T = 4$) **(d)** CarRacing ($T = 8$)

図 2.7: 連続制御タスクの結果. 実線は平均値, 色付き部分は標準偏差を表している.

結果を図 2.7 に示す. 中間フレーム (図中の skipped frames) を使わないベースラインと比較して, 特に $T = 8$ の場合に提案手法のパフォーマンスが高い結果となった. これは, $T = 8$ のときベースラインは全体の 1/8 のデータしか使うことができないのに対し, 提案手法ではすべてのデータを使うことができるためだと考えられる. さらに Pendulum では, $T = 8$ の場合にベースラインの学習が全く進んでいない. この原因としては, 図 2.8 に示すように, 8 枚のうち 1 フレームだけでは環境のダイ

ナミクスを捉えることが難しかったという可能性が考えられる.. 一方, $T = 4$ の場合 Pendulum ではパフォーマンスに違いはなく, CarRacing では提案手法の方が少し良い結果となった. この場合はベースラインで失われるデータ量が先程より少なく, 学習が成功していると考えられる.

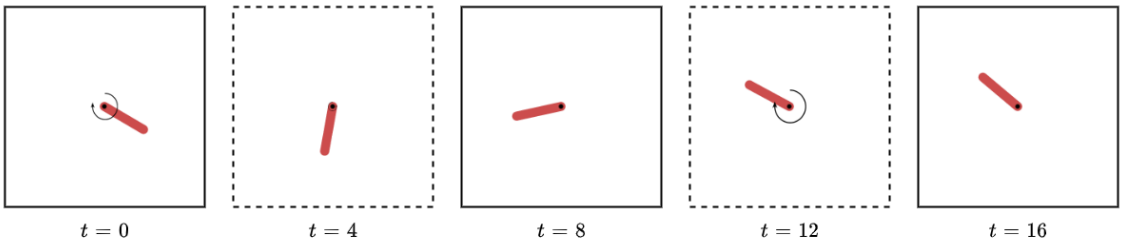
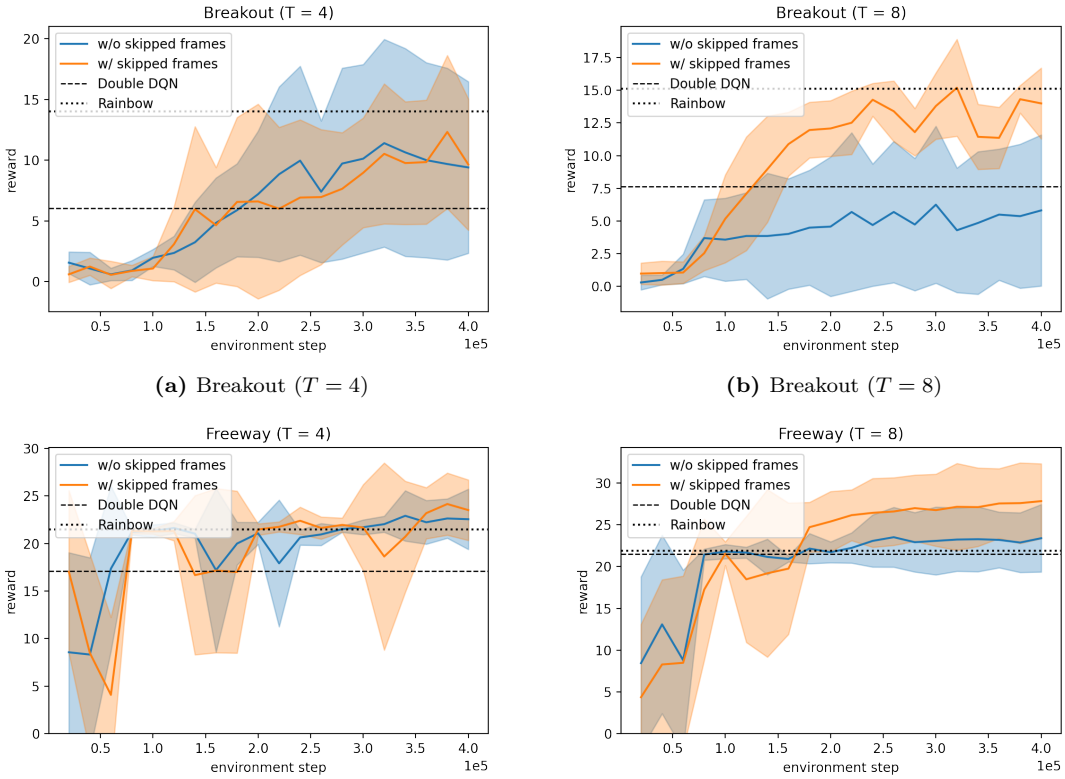


図 2.8: Pendulum における観測の例. $T = 8$ の場合には太枠で囲まれた観測のみが与えられる. その間の観測が与えられなければ, 環境のダイナミクスを捉えるのは難しいかもしれない.



(a) Breakout ($T = 4$)

(b) Breakout ($T = 8$)

(c) Freeway ($T = 4$)

(d) Freeway ($T = 8$)

図 2.9: 離散制御タスクの実験結果. 実線は平均値, 色付き部分は標準偏差を表している. 破線は Double DQN, Rainbow の最終スコアの平均値を示す.

2.5.2 離散制御タスク

離散制御タスクにおける提案手法の有用性を検証するため, Breakout, Freeway で実験を行った. 強化学習手法には Double DQN を用いた. ただし, 2.4 節で述べたように Q ネットワークの構成は通常とは異なり, 行動の埋め込み表現が使われている. また探索のための手法として, Noisy Network [48] を用いた.

結果を図 2.9 に示す. Q ネットワークの構成を変えたことによりスコアが著しく低下していないことを確かめるため, 標準的なネットワークを用いた Double DQN [40], Rainbow [22] を同じ環境ステップ数で学習した結果も表示してある. 離散制御タスクでも連続制御タスクと同様に, $T = 4$ の場合は大きな違いがなく, $T = 8$ の場合は提案手法の方が高いパフォーマンスを示した.

2.5.3 action repeat の効果

$T = 1, 4, 8$ それぞれにおける提案手法の結果を比較し, action repeat の効果を評価する. $T = 1$ の場合, 提案手法は通常の学習と完全に同等である. 結果を図 2.10 に示す. Pendulum 以外では, action repeat が学習に一定の良い影響を与えていることがわかる. これは 2.2 節で述べたように, action-gap が大きくなったことや探索が促進されたことによると考えられる. このように, action repeat は多くのタスクに必要であり, その上でスキップされたフレームを活用することは重要な意味を持つ.

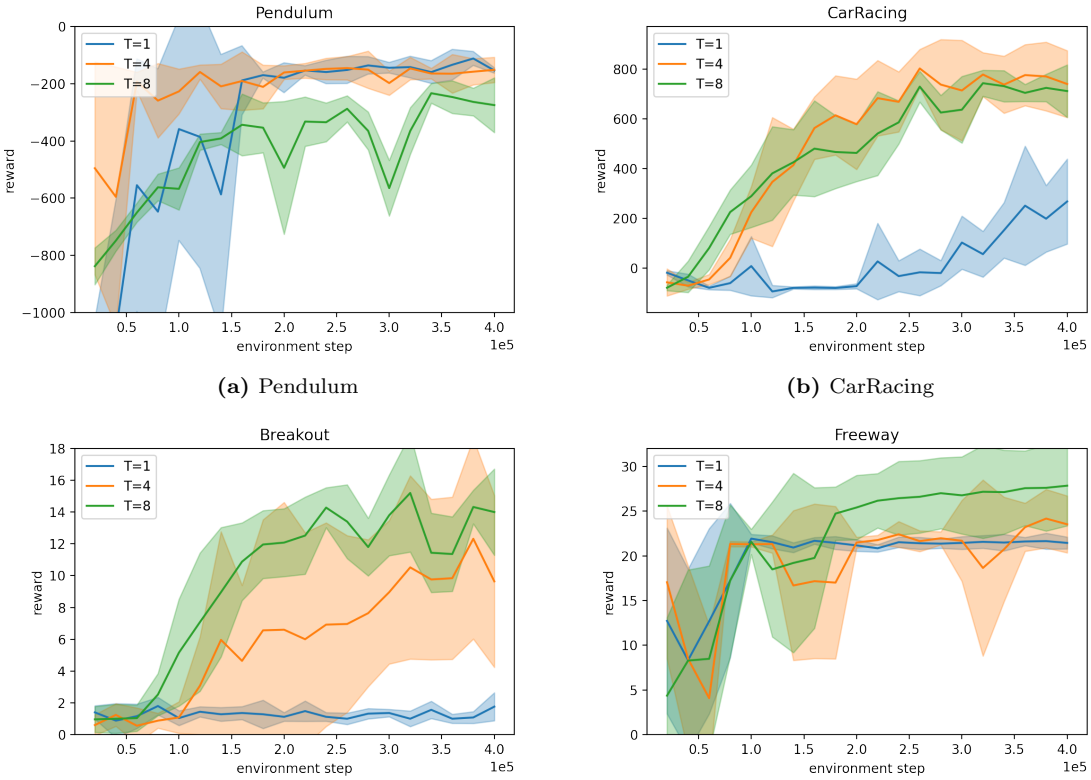
2.6 考察

今回の実験では, 行動の連続・離散に関わらず, action repeat が長い場合に提案手法の成績が高く, そうでない場合に提案手法とベースラインは同程度の成績となった. これは自然な結果ではあるが, $T = 4$ の場合も提案手法で活用できるデータ量はベースラインの 4 倍になっており, より良いパフォーマンスが期待される.

そのような結果が得られなかった原因としては, 提案手法における擬似的な行動のモデリングが最適でなく, 追加されたデータが学習に悪影響を及ぼした可能性が考えられる. 今回は簡易的な方法として連続制御タスクでは行動の重み付き平均を利用したが, 状態の変化から行動を推測する逆モデル [49, 50] などを用いることで, より良い擬似的な行動の表現が得られるかもしれない.

さらに, 離散制御タスクでは行動の埋め込み表現の平均を擬似的な行動としているが, この埋め込み表現は Q 関数の学習において得られるものである. そのため, 離散制御タスクの行動の表現は連続制御タスクと異なり, 環境のダイナミクスと直接のつながりはない. 一方, 環境モデルの学習の中で離散行動の埋め込み表現を獲得すれば, その表現はダイナミクスを反映したものになると期待される. そしてその表現を用いることで, 擬似的な行動がより適切にモデリングされ, 提案手法のパフォーマンスをさらに向上させられる可能性がある.

また今回の実験で扱った環境では, 行動が何らかの方向を表していたが, 異なる意味合いを持つ行動に関しても提案手法が有効であるかについては, さらなる検証が必要となるだろう.



(a) Pendulum **(b)** CarRacing

(c) Breakout **(d)** Freeway

図 2.10: 異なる action repeat におけるパフォーマンスの比較. 実線は平均値, 色付き部分は標準偏差を表している.

2.7 結論

本研究では, 通常利用されていない action repeat 間のデータを連続・離散両方の制御タスクで活用する手法を提案した. そして OpenAI Gym の環境で実験を行い, その有効性を示した. またいくつかのタスクにおいて, 長い action repeat がパフォーマンスの向上につながることが確認され, 行動を繰り返すことの理論的な利得が実証される結果となった.

今後の課題は, 手法を改善し, より洗練された方法で擬似的な行動を獲得することである. 2.5 節で述べたように提案手法は改善の余地があり, さらにパフォーマンスを向上させられる可能性がある.

本研究ではオンライン強化学習に取り組んだが, エージェントが環境と相互作用せずに学習するオフライン強化学習も近年盛んに研究されている [51]. オフライン強化学習では与えられたデータのみから方策を学習する必要があり, それ以上にデータを集めることはできないため, 提案手法のようにデータを拡張する手法は重要であると考えられる. データ収集時に action repeat 間のデータを保存していれば提案手法はオフライン強化学習にも適用でき, この設定における研究も興味深いと思われる.

2.8 付録

2.8.1 擬似的な行動に関する詳細

2.4 節で述べたとおり, action repeat 間のデータに関しては 状態 s_{kT+l} から行動 $a_{kT+l}(= a_{kT})$ を T 回繰り返して状態 s_{kT+l+T} に至るわけではない. そのため, そのままでは正規のデータと同様に扱うことはできない. しかし, 行動の平均として得られる擬似的な行動 $\hat{a}_{kT+l}$ に関しては, それを s_{kT+l} から T 回繰り返すと s_{kT+l+T} に近似的に等しい状態に至る.

ここでは, 連続時間の設定で考える. エージェントの状態 $x(t)$, 行動 $u(t)$ が何らかのダイナミクス f を用いて

$$x(t+T) = x(t) + \int_t^{t+T} f(x(t'), u(t'))\, dt'$$

という関係にあるとする. 状態が $t \le t' \le t+T$ において急激に変化しないと仮定すると,

$$x(t+T) \approx x(t) + \int_t^{t+T} f(x(t), u(t'))\, dt'$$

と近似できる. ここで,

$$u(t) = \begin{cases} u_1 & (t \le t + pT) \\ u_2 & (t > t + pT) \end{cases}$$

とする. ただし, p は $0 < p < 1$ を満たす実数である. すなわち, エージェントは時刻 $t + pT$ までは行動 u_1 を繰り返し, その後は新しい行動 u_2 を繰り返す.

$$x(t+T) \approx x(t) + \int_t^{t+pT} f(x(t), u_1)\, dt' + \int_{t+pT}^{t+T} f(x(t), u_2)\, dt'$$

$$= x(t) + pT\, f(x(t), u_1) + (1-p)T\, f(x(t), u_2)$$

ここで, $\hat{u} = p\, u_1 + (1-p)\, u_2$ を行動の平均とする.

$$u_1 = \hat{u} - (1-p)\,(u_1 - u_2), \quad u_2 = \hat{u} + p\,(u_1 - u_2)$$

と表されることに注意し, 行動に関して 1 次近似すると,

$$
\begin{aligned}
x(t+T) &\approx x(t) + pT\, f(x(t), \hat{u} - (1-p)\,(u_1 - u_2)) \\
&\quad + (1-p)T\, f(x(t), \hat{u} + p\,(u_1 - u_2)) \\
&\approx x(t) + pT \left\{ f(x(t), \hat{u}) - (1-p)\,(u_1 - u_2)\, \frac{\partial f}{\partial u}(x(t), \hat{u}) \right\} \\
&\quad + (1-p)T \left\{ f(x(t), \hat{u}) + p\,(u_1 - u_2)\, \frac{\partial f}{\partial u}(x(t), \hat{u}) \right\} \\
&= x(t) + T f(x(t), \hat{u}) \\
&= x(t) + \int_t^{t+T} f(x(t), \hat{u})\, dt' \approx x(t) + \int_t^{t+T} f(x(t'), \hat{u})\, dt'
\end{aligned}
$$

つまり, $u(t)$ を定数 $\hat{u}$ に置き換えても, 次の状態 $x(t+T)$ は概ね変化しない.

2.8.2 ニューラルネットワークの構成

2.4 節で述べたとおり, 提案手法では離散制御タスクのネットワークを連続制御タスクのものに近づける. そのため, 画像をエンコードするネットワークと Q ネットワークは同じ構造とした.

2.8.2.1 エンコーダネットワーク

エンコーダネットワークの構造は SAC+AE [52] を参考にした. このモデルは画像入力から SAC により方策を学習するいくつかの研究で用いられている [8, 9]. このエンコーダは 3×3 のカーネル, 32 のチャネルを持つ畳み込み層 4 層で構成されている. 活性化関数には ReLU を用いた. 第 1 層のストライドは 2, それ以降の層のストライドは 1 とした. 畳み込み層の出力は LayerNorm [53] により正規化され, 全結合層で 50 次元のベクトルに変換された後, tanh が適用される.

2.8.2.2 Q ネットワークと方策ネットワーク

Q ネットワークと方策ネットワークはいずれも 3 層の全結合層で構成されており, 隠れ層の次元は 256 とした. 活性化関数には ReLU を用いた. Q ネットワークはエンコーダネットワークの出力と行動 (または行動の埋め込み表現) を受け取り, 1 つの Q 値を出力する. 離散行動の埋め込み表現はすべてのタスクで 8 次元とした. 方策ネットワークは連続制御タスクにのみ用いられ, エンコーダネットワークの出力を受け取り, 方策を表す正規分布の平均と分散を出力する.

2.8.3 その他のハイパーパラメータ

表 2.1 にハイパーパラメータの設定を示す. 離散制御タスクでは, 学習を安定させるため Double DQN [40] を用いた. 報酬のスケールの違いに対応するため, 連続制御タスクでは PopArt [54] を用い, 離散制御タスクでは $[-1, 1]$ に報酬をクリップした.

表 2.1: 実験に用いたハイパーパラメータ

	連続行動タスク	離散行動タスク
環境ステップ数	400k	400k
画像サイズ	84×84	84×84
フレームスタック	4	4
1 アップデート当たりの環境ステップ	4	4
オプティマイザ	Adam [55]	Adam
学習率	0.001	0.0003
環境ステップ当たりの割引率 γ	$0.99^{1/4}$	$0.99^{1/4}$
ターゲットネットワークの更新率 τ	0.005	0.005
方策ネットワークの更新頻度	2	-
ミニバッチサイズ N	32	64
リプレイバッファサイズ	制限なし	制限なし
学習を開始するリプレイバッファサイズ	500	500
エピソード当たりの最大環境ステップ数	制限なし	108k
行動の埋め込み表現の次元数	-	8

第3章　リセット機能の活用

3.1　導入

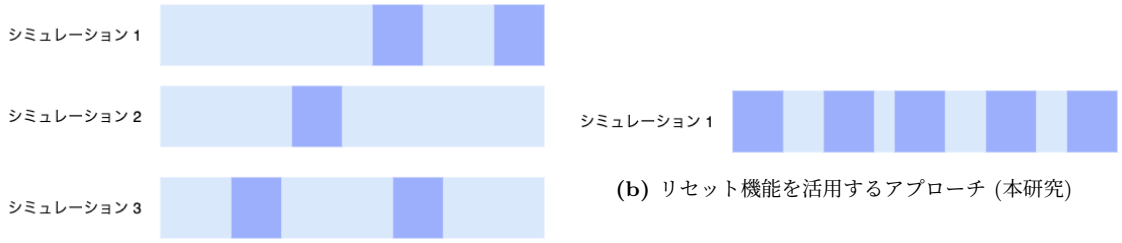
強化学習ではシミュレータが広く利用されている．その理由として, 学習に多くのデータが必要, すなわちサンプル効率が低いことが挙げられる．また, 本研究とは直接の関連はないが, 実環境での学習は安全性に問題があることも大きな理由である．ロボット制御などの分野ではシミュレータ上で学習した方策を実機で利用することが一般的である．一方ゲーム AI のように, 目的の環境そのものがシミュレータである場合もある．

シミュレータ上のデータ収集は実環境で行うよりも一般的に高速であるが, それでもサンプル効率の低さゆえ学習に長時間かかることが多い．学習時間を短縮することはそれ自体も有益であるが, 強化学習の実応用を考えると学習の高速化はさらに重要になりうる．なぜなら, 方策学習を成功させるためには多くのハイパーパラメータを適切に設定する必要があり, 学習と得られた方策の評価を繰り返すことが求められるからである．特に近年盛んに研究されている深層強化学習では, 強化学習アルゴリズム関連のハイパーパラメータだけでなく, ニューラルネットワークの構造や最適化方法など, 選択しなければならない設定が非常に多くなっている．学習を素早く行うことができると学習と評価のサイクルを多く繰り返すことができ, 結果的により良い方策を得られることにつながる．以上のことから, シミュレータにおける方策学習を効率化することは重要である．

シミュレータの特性を活かして学習を効率化するアプローチとしては, 多数のシミュレーションを同時に実行して大量のデータを素早く集めるという方法がある [45, 56]．この方法は必要な環境が準備できる場合は有効性が高いが, 多数のシミュレータやそれを動作させる計算資源を利用できない場合もある．また, 学習中に似たようなデータが大量に集められることも考えられ, 学習の時間効率は高くてもエネルギー効率は低くなってしまうという懸念がある．

本研究では別のアプローチとして, シミュレータのリセット機能を活用して効率化を図る．ここでのリセット機能とは, 学習中に訪れた状態を記録しておけば, リセット機能を使うことでいつでもその状態に戻ることができるというものである．通常の強化学習はこのような機能を前提としないが, 本研究ではリセット機能を活用し, 学習に有用なデータを重点的に集めることで方策学習を効率化する．上述した多数のシミュレーションを同時実行するアプローチと比較したイメージ図を図 3.1 に示す．図 3.1a では複数のシミュレーションを同時に実行することでデータの量を増やしている．一方, 図 3.1b では単一のシミュレーション内で適切な状態にリセットすることで, 方策学習という観点におけるデータの質を高めている．当然これらのアプローチを組み合わせてさらに効率的な学習を行うことも可能である．

リセット機能を活用する既存手法はいくつか存在する [57, 58, 59] が, ドメインに特化しているた



(a) 多数のシミュレーションを並列実行するアプローチ

(b) リセット機能を活用するアプローチ (本研究)

図 3.1: シミュレータの特性を利用する 2 つのアプローチの概念図. 各ボックスが 1 シミュレーションから得られるデータを表している. 色の濃い部分はそのデータが方策学習に特に重要であることを示している.

め汎用性に乏しかったり, リセットする状態の選び方がヒューリスティックで効果が小さかったりと改善の余地がある. 本研究ではリセット機能を用いて累積報酬の高い軌跡を素早く見つけることで, サンプル効率の高い学習を目指す. 累積報酬という一般的な指標を用いることで, 環境ごとに独自の指標を考案する必要がなく, 汎用的な手法となりうる.

3.2 背景

強化学習の一般的な知識についてはすでに 2.2 節で述べたため, ここでは省略する.

3.2.1 リセット機能

本研究では, エージェントが観測する状態 $s_t \in \mathcal{S}$ に加えてシミュレータの内部状態 $h_t \in \mathcal{H}$ を利用できると仮定する. この内部状態 h_t を記録しておき, 後にシミュレータにセットすることで, シミュレータを時刻 t と完全に同じ状態に戻すことができる. リセット機能の具体的な使用例を図 3.2 に示す.

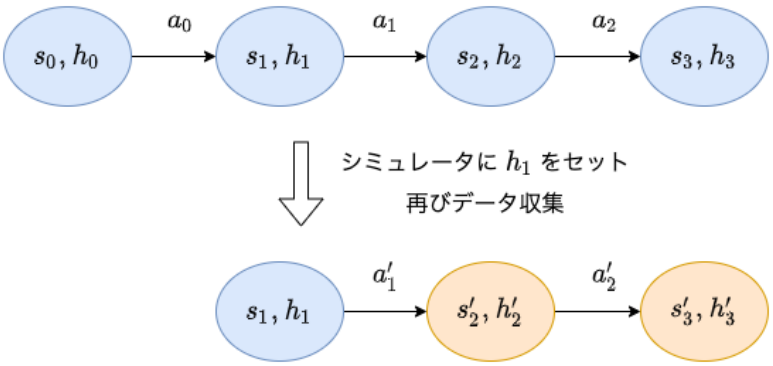


図 3.2: リセット機能の使用例

この仮定に基づくと, h_t をセットしてから行動系列 $a_t, a_{t+1}, \cdots, a_{t+k-1}$ を取ると, 状態系列 $s_{t+1}, s_{t+2}, \cdots, s_{t+k}$ が必ず得られるということになる. 環境が確率的な遷移を持つ場合, このような再現性を保証することはできない. しかし, シミュレータ内でのランダム性の実現には擬似乱数が用いられており, 乱数生成器の状態も内部状態に含めて考えると, 多くの場合においてこの仮定は成立する.

内部状態を取得する機能とセットする機能は, 多くのシミュレータにおいて実装されているか, 容易に実装可能である. 例えば強化学習の研究で広く用いられている Atari [27] や MuJoCo [47] といった環境では, 以下のような関数が用意されている.

Atari

取得する関数：clone_state
セットする関数：restore_state

MuJoCo

取得する関数：get_state
セットする関数：set_state

実験では, これらの関数を通してリセット機能を利用した.

3.3 関連研究

シミュレータによりオンラインでプランニングを行う手法では, リセット機能が活用されてきた [60, 61]. ただし, これらの手法は学習した方策を利用する際もシミュレータが必要であり, また完璧な精度のシミュレータが求められる. 本研究では学習時にのみシミュレータのリセット機能を使って効率改善を目指しているため, 以下で紹介する関連研究からプランニングによるアプローチは除いている.

3.3.1 リセット機能による多様な状態の収集

本研究と直接関連するのは, Tavakoli らの研究 [57] である. 彼らはシミュレータのリセット機能を活用して多様な状態を収集する手法を提案した. この研究は on-policy 学習を前提としており, 過去のデータを直接は利用できない on-policy 学習において, 過去に訪れた状態にリセットするという形で過去のデータを活用することを目的としている.

リセットする状態を選択する方法はヒューリスティックに決められており, タスクの性質に応じて以下の３つが使用される.

一様サンプリング

これまでに得られた状態から一様にサンプリングする. 常に初期状態からエピソードを開始するのではなく様々な状態から始めることで, より多様なデータが集められることが期待される.

状態価値推定の誤差に応じたサンプリング

状態価値 $V^\pi(s)$ は状態 s から方策 π にしたがって行動したときに得られる累積報酬の期待値であり, Q 値と以下の関係式が成り立つ.

$$V^\pi(s) = \mathbb{E}_\pi \left[Q^\pi(s, a) \right]$$

状態価値関数も Q 関数と類似する損失関数の最小化により学習できる. 損失関数の値が大きい状態は学習が進んでいないと考えられるため, そのような状態を重点的に収集する.

エピソード内で得られた報酬に応じたサンプリング

報酬を得られる状態が少ない環境は, 報酬が疎な環境と呼ばれる. 例えばゲームを完全にクリアしたときに報酬 1 が与えられ, それ以外は常に報酬 0 が与えられる環境は報酬が疎であると言える. このような場合, 報酬が得られなかったエピソードからは得られる学習信号が少なく, 成功した経験から優先的に学習することが有効である [62]. そのため, 累積報酬が高いエピソードを重点的にサンプリングし, さらにその中から 1 つの状態を一様にサンプリングする.

タスクの性質に応じてこれらの方法を使い分けることで, 実験では一定の効果が示された. しかし, on-policy 学習は off-policy アルゴリズムに比べて一般的にサンプル効率が低いことが知られている. 実際に, 彼らの実験では PPO [23] という on-policy アルゴリズムが使用されているが, MuJoCo [47] という広く使用されているタスクでの学習効率は, 後に提案された SAC [24] などの off-policy アルゴリズムに比べて大きく劣っている. 本研究はサンプル効率の向上を目的とするため, off-policy アルゴリズムを前提とする.

3.3.2 リセット機能によるカリキュラム生成

Salimans らは Montezuma's Revenge という探索の難しいゲームを, 人間プレーヤのデモンストレーションを 1 つだけ用いて解く手法を提案した [59]. 探索の難しい環境では, ランダムな行動による探索では一向にゴールまで到達できず, 全く学習が進まないということが往々にしてある. そこで彼らはエージェントをゴールに近い位置にセットし, そこからゴールまでの短い距離を学習させることで探索を簡単化した. デモンストレーションを使用すれば, ゴールに近い位置をサンプルすることは容易である. 短い距離のタスクを解けるようになったら, エージェントをセットする位置を徐々にゴールから遠ざけていく. このようにカリキュラムを生成することで, 難しいタスクも効率的な方策学習が可能になる. 学習の様子を図 3.3 に示す.

当然ながらこの手法にはデモンストレーションが必要であり, また探索の難しい環境に特化したものとなっている. 本研究では, 外部情報を極力用いないより汎用的なアルゴリズムの考案を目指す.

3.3.3 リセット機能を用いた解法の発見

Ecoffet らが提案した Go-Explore [58] は探索が困難な問題を解くための新しい枠組みである. Go-Explore では 2 つのフェーズに分けて問題を解く. 第 1 フェーズでは環境の遷移が決定的であるとして解法を見つける. 第 2 フェーズでは見つけた解法をもとに, 遷移にランダム性がある場合にも有効な方策を学習する. どちらにも強化学習は必須ではなく, 行動決定問題への新しい枠組みと言える.

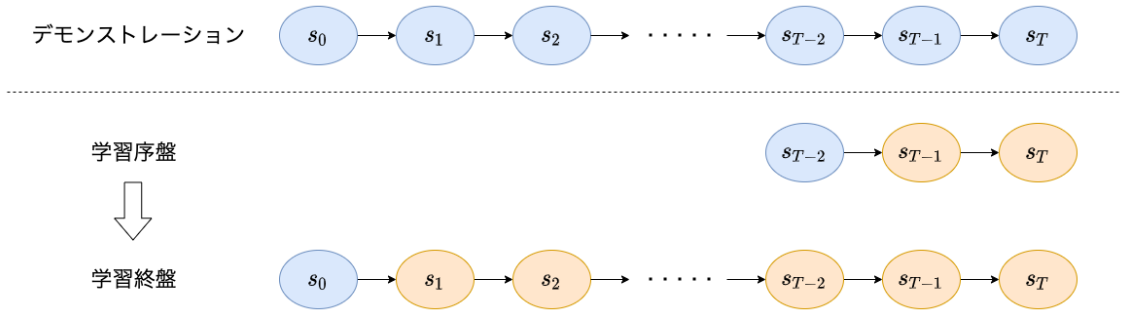


図 3.3: デモンストレーションを用いたカリキュラム学習．水色の状態はデモンストレーションに含まれるもの，橙色の状態はエージェントが収集したものを表す．

ただし，彼らが提案した具体的なアルゴリズムは Montezuma's Revenge のようなビデオゲームを解くことに特化したものであり，いくつかの点でドメイン知識が用いられている．特に重要なのは，第 1 フェーズで用いられるセル表現の抽出とリセットするセルの選択である．

Go-Explore では意味のある形で異なる状態を識別するためにセル表現を用いる．セル表現とは，ゲーム画面を白黒化，縮小などすることで粗い表現にしたものである．例えば画面のうち 1 ピクセルだけが異なっていたとしても実質的に同じものとして扱うべきであるが，状態の代わりにセル表現を使うことでそのような扱いが可能になる．さらに Montezuma's Revenge に特化した方法として，エージェントの位置や部屋番号などの外部情報によりセル表現を構築することも試している．

上記のようにして得られたセルのアーカイブから，リセットするセルを選択し，その状態からランダムな行動を取って探索を行う．セルの選択のため，各セル c に対してヒューリスティックに決められたスコア $CellScore(c)$ が計算され，スコアに応じて確率的にセルが選ばれる．具体的には $CellScore$ は以下の 3 つのサブスコアから構成される．

計数スコア

計数スコアはあるセルが様々な方法で操作された回数を表す属性から計算される．例えばそのセルがリセットに選ばれた回数や，探索中にそのセルを訪れた回数などである．効率的な探索のためには，それらの回数が少ないセルをリセットに選択することが有益である．各属性 a に対して以下のように計数スコアを定義する．

$$CntScore(c,a) = w_a \left(\frac{1}{v(c,a) + \varepsilon_1} \right)^{p_a} + \varepsilon_2$$

ここで，$v(c,a)$ はセル c の属性 a に関する回数であり，p_a, ε_1, ε_2 はハイパーパラメータである．

隣接スコア

外部情報が与えられる場合は，エージェントの位置から隣接するセルを判別することができ，そのセルにすでに訪れたことがあるかを調べることができる．隣接するセルに訪れたことがないセルは，そこから探索することで新しいセルに到達できる可能性が高く，優先的にリセットに選択すべきである．なおここでの"隣接"には，エージェントが上下左右にずれている場合や，エージェントの位置は同じで持っている鍵の個数が違う場合など複数の種類がある．そこで，隣接の各種類 n に対して以下

のように隣接スコアを定義する.

$$NeighScore(c,n) = \begin{cases} w_n & \text{(if neighbor } n \text{ of } c \text{ is already visited)} \\ 0 & \text{(otherwise)} \end{cases}$$

ここで, w_n はハイパーパラメータである. 外部情報が与えられない場合は, $NeighScore$ は 0 とする.

レベルスコア

Montezuma's Revenge において外部情報が与えられる場合は, 現在の部屋のレベルが特定できる. よりレベルの高い部屋を優先的に探索するため, レベルスコアを以下のように定義する.

$$LevelScore(c) = 0.1^{MaxLevel - Level(c)}$$

ここで, $MaxLevel$ はこれまでに到達した最高レベル, $Level(c)$ はセル c の部屋のレベルである. 外部情報が与えられない場合は, $LevelScore$ は 1 とする.

以上のサブスコアから, 最終的な $CellScore$ は以下のように計算される.

$$CellScore(c) = LevelScore(c) * \left[\sum_a CntScore(c,a) + \sum_n NeighScore(c,n) + 1 \right]$$

セル表現の設計やリセットするセルの選択方法は Montezuma's Revenge のようなゲームに特化したものであり, ユーザが決めなければならないハイパーパラメータも数多く存在する. そのためより汎用的な手法が求められる.

3.4 提案手法

リセット機能を活用し, なるべく少ない試行回数で累積報酬の高い軌跡を発見する手法を考案する. 基本的なアイディアは, これまでに見つけた最も累積報酬の高い軌跡の途中の状態にリセットし, そこから再度探索することで, さらに累積報酬の高い軌跡を見つけるというものである. エピソード途中までのデータを再利用することにより, 必要なデータを少なくすることができる.

3.4.1 全体の流れ

初期状態分布 $\rho_0(s)$ からサンプルされた初期状態にリセットすることを通常リセット, 提案手法により選ばれた状態にリセットすることを選択的リセットと呼ぶこととする. 通常リセットは通常の学習で行うリセットのことである. 選択的リセットでは以下で説明する方法でリセットする状態を選び, その状態からエピソードを開始する. エージェントは, 通常リセットで集められるデータと選択的リセットで集められるデータの比が一定になるようデータを収集する.

3.4.2 リセットする状態の選択

リセットする状態には, その状態から再開することでより高い累積報酬が得られる見込みの高いものを選ぶ必要がある. そこで, Q 値を用いた次のスコアを考える.

$$StateScore(s) = \max_a Q_\theta(s, a) - G$$

ここで, s は軌跡に含まれる状態, G は s から先で得られた割引累積報酬である. このスコアは状態 s から現在の方策に従って行動した場合に期待できる, 割引累積報酬の改善幅を表している. ただし, 状態 s における Q 値が過大に推定されている場合, このスコアも実際よりも高く見積もられてしまう. それを防ぐため, 以下のように計算方法を改良する.

$$StateScore(s) = \max_a Q_\theta(s, a) - \max\{G, \max_{a \in A(s)} Q_\theta(s, a)\} \tag{3.1}$$

ここで, $A(s)$ はこれまでに s で取られた行動の集合である. 追加した項により, Q 値が過大に推定されても不当にスコアが高くならず, 悪影響を軽減できると考えられる. スコア計算の具体例を図 3.4 に示す.

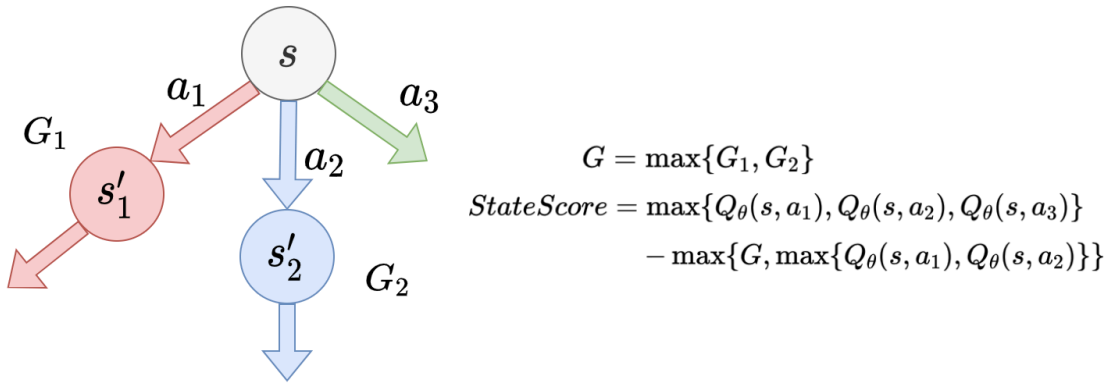


図 3.4: スコア計算の具体例. この例では, 状態 s からこれまでに行動 a_1, a_2 を取ったことがあり, a_3 は取ったことがない. すなわち $A(s) = \{a_1, a_2\}$.

$StateScore$ が高い状態を選択してリセットすることで, 累積報酬が高い軌跡を見つけることが期待できる. ただし, 単純にスコアが最大の状態を選んだ結果, エピソードの序盤の状態が選ばれると, 初期状態からエピソードを開始する通常の学習とあまり違いがなくなってしまう. そこでエピソードの終端に近い状態を優先的に選び, 積極的にデータの再利用を図る. 具体的には, 軌跡に含まれる全状態の $StateScore$ を計算し, スコアが全体の c パーセンタイル以上となる状態のうち, 最もエピソードの終端に近い状態を選ぶ. こうすることで, 大きな累積報酬の改善が期待でき, かつ多くのデータが再利用されるような状態をリセットに選ぶことができる. c はハイパーパラメータであり, $c = 100$ の場合は $StateScore$ が最大の状態を選ぶことになり, $c = 0$ の場合はエピソード終端の状態を選ぶことになる.

Algorithm 4 リセット状態の選択およびエピソード中断の閾値計算

Input: Q function Q_θ, best trajectory so far τ, hyperparameter of percentile c
Output: state to reset s_{reset}, stop threshold v_{stop}

Initialize $G_{\text{length}(\tau)} = 0$ ▷ discounted return
for $k = \text{length}(\tau) - 1$ to 0 **do**
  $(s_k, a_k, r_k) = \tau[k]$
  $G_k = r_k + \gamma G_{k+1}$
  $Q_{\text{base}} = G$
  for a in $A(s_k)$ **do**
    $Q_{\text{base}} = \max\{Q_\theta(s_k, a), Q_{\max}\}$
  end for
  $StateScore_k = \max_a Q_\theta(s_k, a) - Q_{\text{base}}$
end for
$\eta = \text{percentile}(\{StateScore_k\}_{k=0}^{\text{length}(\tau)-1}, c)$ ▷ threshold of score
for $k = \text{length}(\tau)$ to 1 **do**
  if $StateScore_k > \eta$ **then**
    $s_{\text{reset}} = s_k$
    $v_{\text{stop}} = G_k$
    break
  end if
end for
return $s_{\text{reset}}, v_{\text{stop}}$

3.4.3 その他の工夫

累積報酬の更新が難しいと判断できる場合, そのエピソードを中断して新しい状態にリセットすることもサンプル効率向上のために重要である. そのため 1 ステップごとに次式で表される N ステップリターンの上限値を計算し, これが現時点での割引累積報酬の最大値 G を P 回連続で下回った場合にエピソードを中断する.

$$\sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n (\max_a Q_\theta(s_{t+n}, a) + Q_u)$$

ここで, t はリセットに選ばれた状態のタイムステップ, n はリセットから現時点までの経過ステップ, Q_u は Q 関数の不確かさである. Q_u の真の値を得ることはできないため何らかの指標で代替する必要があるが, 今回は簡単のため TD 誤差の指数移動平均 $\overline{\delta^2}$ を使い,

$$Q_u = w\sqrt{\overline{\delta^2}}$$

とする．この値は Q 値の予測誤差を大まかに測ったものになっており, w はハイパーパラメータである．

3.4.4 実装

本手法では 1 つの状態から複数の行動を取った遷移データを扱うことになる．そこで図 3.4 のように状態をノード, 行動をエッジとする木構造としてデータを表現するように実装した．リセット状態の選択およびエピソード中断の閾値計算を行う擬似コードを Algorithm 4 に示す．擬似コード中の percentile(A, c) は, 集合 A に含まれる値の c パーセンタイルを計算する関数である．また, 擬似コードでは各状態の Q 値をその場で計算しているが, これには時間がかかる．そのため実際には, Q ネットワークの学習時の推定値を保存しておき, その値を使用して状態の選択を行った．

3.5 実験

3.5.1 実験設定

OpenAI Gym に含まれる以下の 3 つの環境において, 提案手法を通常の学習と比較する．

CartPole
: ポールが取り付けられた台車を左右に動かし, ポールが倒れないようにするタスク．状態は台車の位置, 速度などを表す 4 次元の実数値で与えられる．取りうる行動の種類数は 2.

Pong
: 卓球を模したビデオゲーム．パドルを上下に操作し, ボールを打ち返す．状態は画像で与えられる．取りうる行動の種類数は 6.

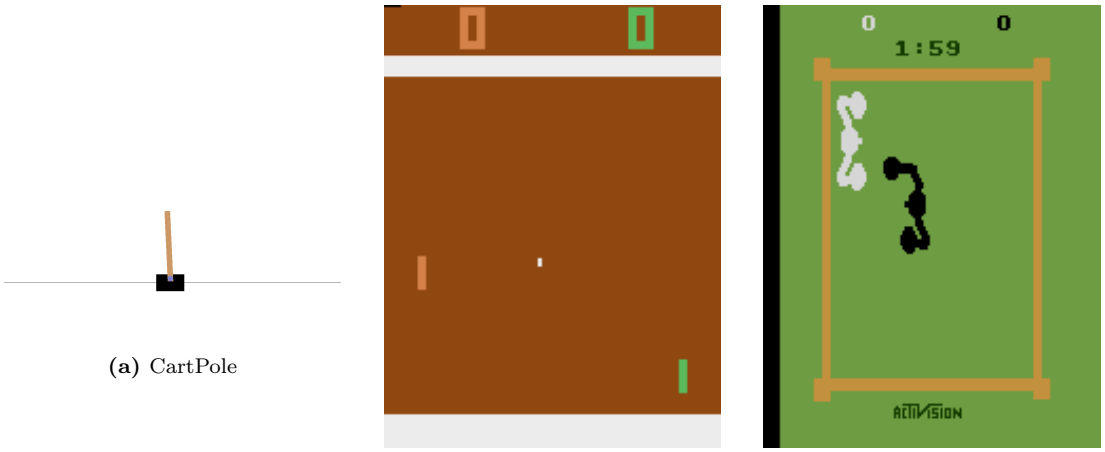
Boxing
: ボクシングを模したビデオゲーム．左右の腕を動かし, 相手にパンチを当てる．状態は画像で与えられる．取りうる行動の種類数は 18.

通常リセットで集められるデータと選択的リセットで集められるデータの比は 1:9 で固定し, 最適化していない．強化学習アルゴリズムには Double DQN を用いた．モデルの構成やその他のハイパーパラメータについては 3.8 節で述べる．すべての実験は 5 個のランダムシードで行った．

3.5.2 実験結果

結果を図 3.6 に示す．

CartPole では提案手法により大幅に学習効率が上昇している．また, ハイパーパラメータ c が結果にほとんど影響しないこともわかる．$c = 100$ の場合は単純に $StateScore$ が最大の状態を選ぶことに相当するが, その場合も高い学習効率を達成している．これは, CartPole のタスクの性質によるも



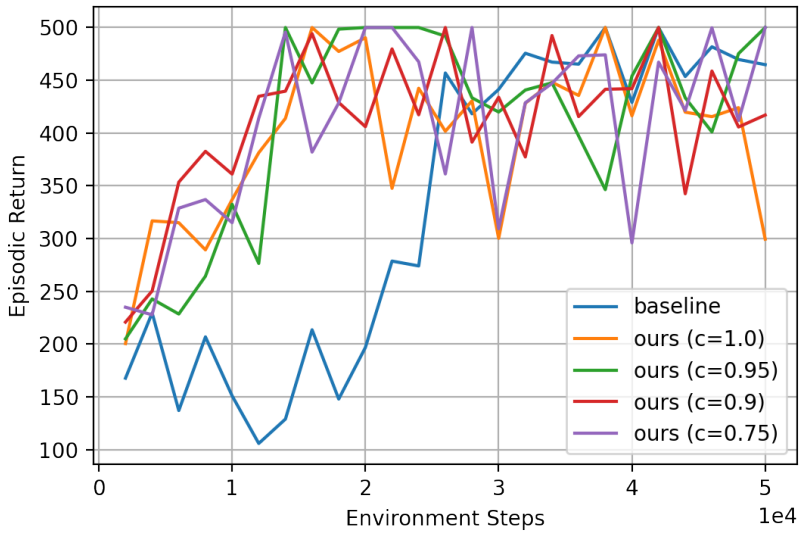
(a) CartPole **(b)** Pong **(c)** Boxing

図 3.5: 各環境のサンプル画像. 実際にはこれらの画像をグレースケールに変換し, サイズを縮小してから使用する. 詳細は 3.8.2 小節を参照.

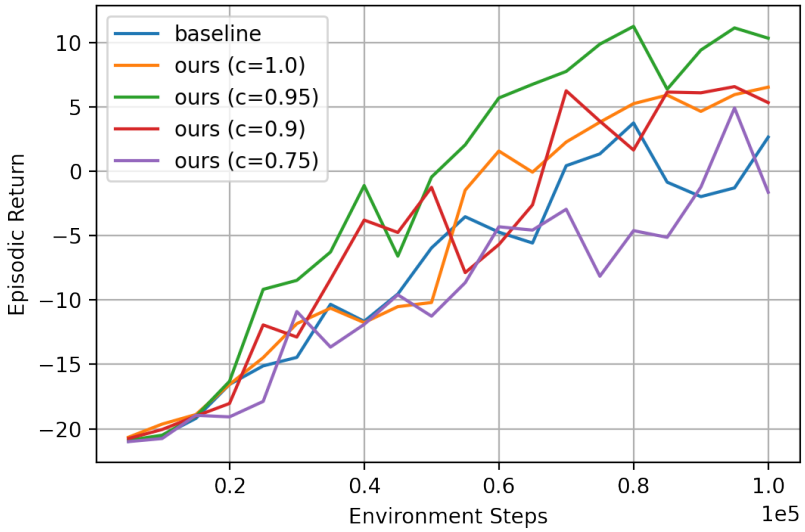
のであると考えられる. CartPole はポールが倒れるまで常に報酬 1 が与えられ, 倒れるとエピソードが終了する. そのため $StateScore$ が大きくなるのはポールが倒れる直前, すなわちエピソードの終端付近になる可能性が高い.

Pong においても提案手法による学習効率の上昇が見られる. 最もサンプル効率が良いのは $c = 95$ の場合である. $c = 100$ の場合はデータの再利用が十分でなく, また $c = 90, 75$ の場合は $StateScoer$ があまり高くない状態を選んでしまうために効率が低下すると考えられる. 実際に Pong でリセットに選択されたタイムステップの平均値を比較すると, 図 3.7 のようになる. c の値が大きいほど, タイムステップの小さい, すなわちエピソードの終端から遠い状態が選ばれていることがわかる.

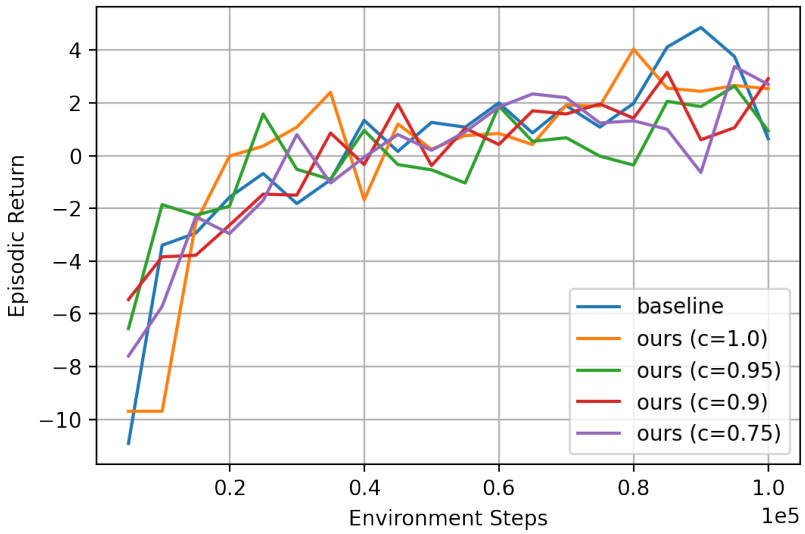
一方, Boxing では提案手法による改善は見られなかった. 提案手法で得られた累積報酬の最高値は, 図 3.8 で示すように c の値によらずテスト時の累積報酬よりも大幅に高い. つまり累積報酬の高い軌跡は見つけられているが, モデルが汎化できていないということがわかる. この原因は定かではないが, Boxing は他のタスクに比べて入力が複雑であり (高い累積報酬を得るには自分と相手の位置関係, 腕の状態などを正確に把握する必要がある), 汎化しやすい良い表現が抽出できなかった可能性が考えられる. 他にも Boxing は今回扱ったタスクの中で行動の数が最も多く, 実験したステップ数では学習が難しかったという可能性もある.



(a) CartPole



(b) Pong



(c) Boxing

図 3.6: 各タスクの学習曲線. 横軸は学習に使ったステップ数, 縦軸はエピソードあたりの累積報酬を表す. 5 回の実験の平均値を示している.

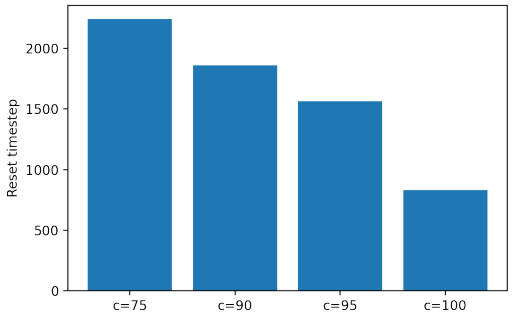


図 3.7: Pong においてリセットに選択されたタイムステップの平均値

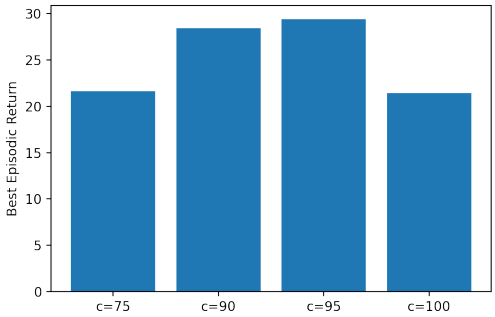
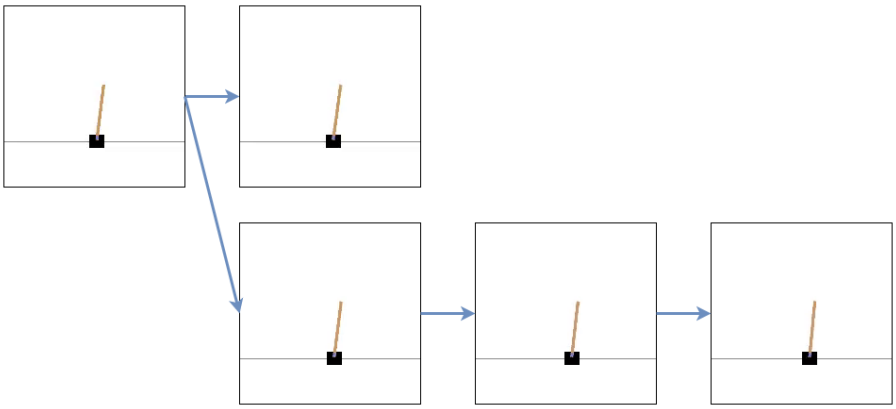
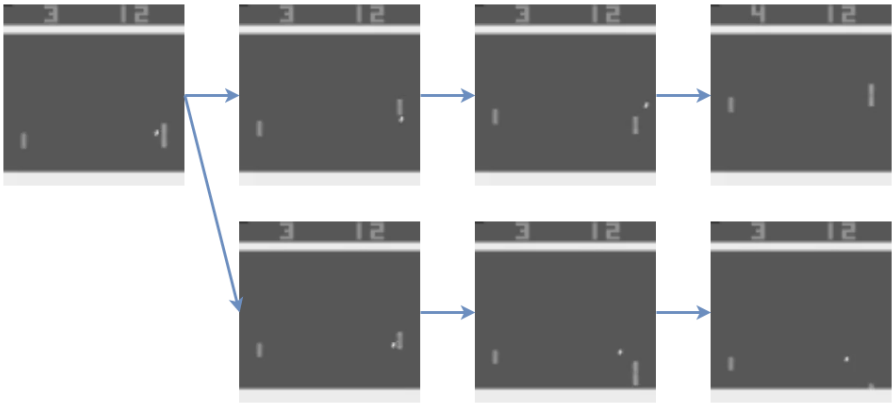


図 3.8: 各初期状態に対する累積報酬の最高値

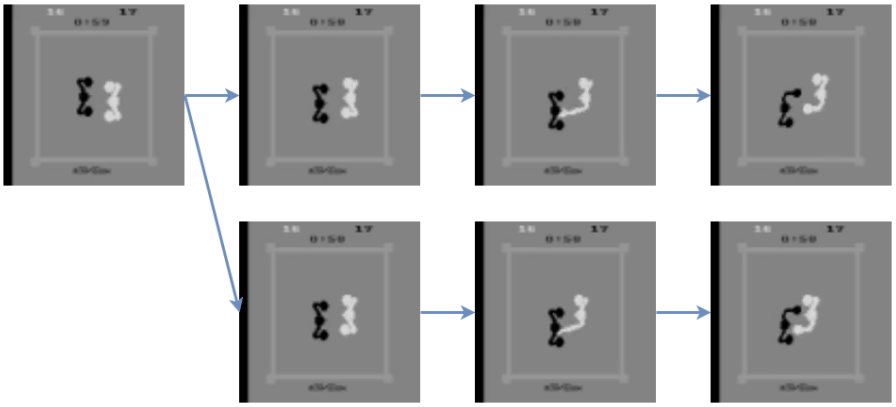
提案手法の動作に関して理解を深めるため, リセットに選択された状態の例を図 3.9 に示す. CartPole では, 上段に示すもとの軌跡ではすぐにポールが倒れてエピソードが終了しているが, 下段では操作が改善されて持ちこたえている. なお, 紙面の都合上 4 ステップ分しか表示していないが, 下段のエピソードはその先も続いている. Pong では, 上段では直後に失点しているが, 下段ではボールを打ち返して失点を防いでいる. Boxing では, 上段で得点できなかった所をリトライしているようにも見えるが, 実際には改善できていない.



(a) CartPole



(b) Pong



(c) Boxing

図 3.9: リセットに選択された状態の例. 上段が更新前の軌跡の一部, 下段が更新後の軌跡の一部を示す. ここではわかりやすさのため CartPole についても画像を表示しているが, 実際には状態は 6 次元の実数値であり, 画像ではない.

3.6 考察

今回の実験では, CartPole においては提案手法による大幅な学習効率改善が見られたものの, Pong では効果が小さく, Boxing では効果が見られなかった. 上述したようにタスク自体の複雑性により他の部分が学習のボトルネックになっている可能性がある. しかし, 提案手法にもいくつか改善の余地があると考えている.

まず, 提案手法は Q 値の予測がある程度正しくなくては意味のない動作をしてしまう恐れがある. そのため, 学習が進み始めてから加速することはできるかもしれないが, そもそも学習が難しい場面で効果を発揮できるかは定かではない.

また, 類似したデータを重複して集めてしまう可能性がある. 提案手法はスコアを早く改善することで学習の効率化を図るが, 多様なデータを集める機構は持たない. 特定の状態からのスコアを改善できると予測して実際には改善できない, という状況が長く続くと, 似たデータを大量に集めてしまい, サンプル効率を下げてしまう恐れがある.

最後に, エピソード終端の状態を選択するように仕向ける必要がある点である. 本来は何らかの統一された基準に基づきリセットする状態を選ぶのが望ましいが, 提案手法は恣意的な方法で対処している. その結果, Pong では提案手法の学習効率はハイパーパラメータ c に強く依存してしまっている.

3.7 結論

本研究ではシミュレータを用いた方策学習を効率化するため, シミュレータのリセット機能を活用する手法を考案し, その効果を検証するための実験を行った. その結果, 一部の環境ではサンプル効率を改善し, また意図した形でリセットが行われていることが確認できた. その一方で, 提案手法の効果がはっきりと現れない環境があることもわかった.

今後の課題は以下の 3 つである.

手法の改善

現状ではリセットする状態の選び方に恣意性が残っており, 最適でない可能性がある. よりシステマチックな方法を模索することで, さらにサンプル効率を高められる可能性がある.

効果が見られなかった原因の特定

Boxing では提案手法の効果が見られなかったが, この原因を特定することで手法を改善したり, 手法がどのようなタスクに効果的かを理解したりすることができる.

実験の拡充

今回は 3 つの環境で実験を行ったが, より多くの環境で実験することで提案手法の有効性を検証したい. また, 提案手法は離散行動のタスクだけでなく連続行動のタスクにも適用できるため, ロボット制御などの連続制御タスクでも実験をしていきたい.

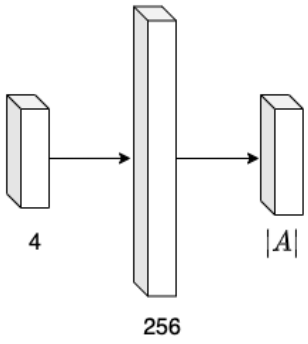
昨今は事前に集められたデータのみで学習するオフライン強化学習が盛んに研究されている [63, 64]. オフライン強化学習ではシミュレータが必須ではないこともあり, 研究コミュニティの全体的な流れとして学習のシミュレータに対する依存度を下げる方向にある [65] と感じている . しかし学習のた

めに十分な量と質のデータを事前に集めることは容易ではなく, 方策学習におけるシミュレータの利用は今後も続くと考えられる. そのため, シミュレータの特性を活用するという方向性の研究にも大きな意義があると考えている.

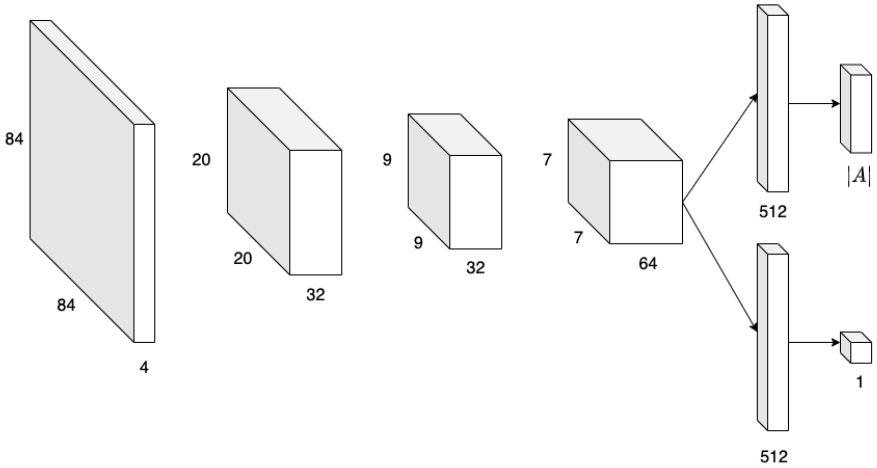
3.8 付録

3.8.1 ニューラルネットワークの構成

3.5 節で述べたとおり, 実験では入力が低次元のタスク (CartPole) と画像のタスク (Pong, Boxing) を扱った. それぞれにおける Q ネットワークの構成を図 3.10 に示す. 入力が画像のタスクでは Dueling Network [66] を用いている. 活性化関数には ReLU を使用した.



(a) CartPole



(b) Pong, Boxing

図 3.10: Q ネットワークの構成図. 図中の $|A|$ は取りうる行動の数であり, CartPole では 2, Pong では 6, Boxing では 18 である.

3.8.2 その他のハイパーパラメータ

すべての環境に共通の設定を表 3.1 に, 環境固有の設定を表 3.2, 3.3 に示す. 探索には ε-greedy を用い, ε を以下のように線形に減少させた.

$$\varepsilon_t = (1 - \alpha_t) \cdot \varepsilon\text{の初期値} + \alpha_t \cdot \varepsilon\text{の最終値}$$

$$\alpha_t = \min\left\{ \frac{t}{\text{スケジューリングのステップ数}}, 1 \right\}$$

表 3.1: 共通のハイパーパラメータ

カテゴリ	種類	値
ニューラルネットワークの学習	オプティマイザ	Adam [55]
	学習率	1.0×10^{-3}
	ステップ当たりのパラメータ更新回数	1
強化学習	割引率 γ	0.99
	リプレイバッファのサイズ	制限なし
提案手法	通常リセットと選択的リセットのステップ比	1:9
	エピソード中断までの回数 P	10
	Q 値の不確かさのスケール w	2.0

表 3.2: CartPole 環境固有のハイパーパラメータ

カテゴリ	種類	値
ニューラルネットワークの学習	バッチサイズ	128
強化学習	N ステップリターン	1
	学習ステップ	50k
	アクションリピート	1
	報酬のクリッピング	なし
	ε の初期値	1.0
	ε の最終値	0.05
	スケジューリングのステップ数	25k

表 3.3: Pong, Boxing 環境固有のハイパーパラメータ

カテゴリ	種類	値
ニューラルネットワークの学習	バッチサイズ	32
強化学習	N ステップリターン	5
	学習ステップ	100k
	アクションリピート	4
	報酬のクリッピング	$[-1, 1]$
	ε の初期値	1.0
	ε の最終値	0.05
	スケジューリングのステップ数	5000
画像処理	フレームスタック	4
	画像サイズ	84 x 84

第4章　おわりに

4.1　本稿のまとめ

本研究では, 環境の特性を活かした方策学習の効率化に取り組みんだ. 具体的には中間フレームとリセット機能を活用する手法を提案し, それらの有効性を検証する実験を行った. 2.7, 3.7 節 で指摘したように, それぞれの手法によるサンプル効率の改善が見られた一方, いくつかの課題も残っている.

4.2　今後の課題

ここでは, 環境の特性を活用するアプローチ全般に関する課題を述べる. 本研究の提案手法に関する課題については, 2.7, 3.7 節を参照.

学習の効率化に利用できる特性としては, 中間フレームやリセット機能以外にもいくつかの候補が考えられる. 以下では 3 つの候補について検討する.

学習時にのみ使用できる特権情報

特にシミュレータを使って学習する場合, 学習時はシミュレータの内部情報として様々な情報を得られるが, テスト時はそれらの情報が得られないという状況が容易に想像できる. もし特権情報が方策学習に役立つならば, まず特権情報を使って方策を学習し, 後に特権情報を使わずに同様の行動が取れるように方策を学習するという方法が有効になりうる.

Chen らはロボットハンドにより様々な物体の向きを変えるタスクにおいて, まず特権情報を用いて教師方策を学習し, それを模倣する形で生徒方策を学習するシステムを構築した [67]. このタスクでの特権情報には, ロボットハンドの指の位置や関節の角速度などが含まれている.

Li らは最高ランクの人間プレーヤと同等の強さを誇る麻雀 AI を開発した [68]. 彼らも特権情報を使って学習を効率化している. 麻雀では多くの情報がプレーヤに隠されており学習が難しいため, 学習初期には全情報が見える設定とし, 学習が進むに連れて見える情報を減らしていくという工夫を施した.

これら以外にも特権情報を使う先行研究はいくつか存在するが, その多くが特定のタスクを解くことを目的としたものであり, 特権情報の活用そのものに焦点を当てたものは多くない. そのため, このアプローチについて理論的・経験的両面でより詳細な検証が必要である. 特に麻雀の例のように特権情報が他の情報から推測しにくい場合, どのように特権情報への依存度を落としていくかは自明でなく, より良い手法が考案できる可能性もある.

報酬関数の情報

一般的に強化学習では報酬関数を未知として学習を行う. しかし, 強化学習を実用する場面を考えると報酬関数を設計するのはタスクを解こうとしているユーザであり, 報酬関数は既知であることが自然である. 報酬関数の情報が方策学習に有用であるならば, 何らかの方法で活用することが望ましい.

環境のダイナミクスを明示的に推定するモデルベース強化学習では, 報酬関数を既知として使用する既存研究もある [7]. 一方, モデルフリー強化学習で使用する研究は, 我々が知る限りでは存在しない.

報酬関数そのものの利用ではないが, ゴールに到達することが目的のタスクにおいて, ゴールの情報を使用して行動を決めることは広く行われている [69, 70]. この方法は報酬関数の情報を直接使うより方法も柔軟性に劣り, 到達したい状態が多数ある場合や, 状態のうち一部だけが報酬に関係する場合には対応しにくい. しかしこれらの問題を解決できれば, 一般的なタスクをゴールに向かうタスクに変換して解くことにより, 学習の効率化が図れる可能性はある.

ダイナミクスの微分情報

強化学習において, 環境のダイナミクスは通常ブラックボックスとして扱われる. しかし, 入出力関係に加えて微分情報も得られれば, 学習に大きく役立つ場合がある.

Mora ら [71] は入出力の微分情報が得られるシミュレータを使い, 方策学習を効率化する手法を提案した. 追加された微分情報により, 目的関数に対する方策パラメータの勾配が従来より正確に計算できるようになり, 安定した学習ができる.

微分可能なシミュレータがない場合でも, モデルベース強化学習においてダイナミクスをニューラルネットワークで近似することで微分情報が得ることができる. Hafner ら [25] は画像入力からダイナミクスモデルをニューラルネットワークにより学習し, そのモデルに関する微分情報を使って方策を学習した.

微分情報が意味を持つのは, ロボット制御などダイナミクスが連続的な環境であり, かつ報酬関数も連続的な場合であると考えられる. ただし, 状態空間を適切に変換したり, 報酬関数を等価なもので置き換えたりすることで, 対象となるタスクを広げられる可能性はある.

以上で述べた 3 つ以外にも環境の特性を利用するアプローチは数多く存在すると思われる. 様々な方向性を模索するため, 今後も研究が必要である. また, 現存する環境の特性を活かすだけでなく, 強化学習を前提とした環境の構築が重要になっていくと考えられる. 例えば本稿ではシミュレータのリセット機能を利用したが, 実装によってはリセット機能の提供が難しい場合もありうる. 様々な分野のシミュレータの開発段階で強化学習の知見を導入することで, より方策学習に適した環境を用意できるようになると期待できる.

参考文献

[1] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. In *International Joint Conference on Artificial Intelligence*, 2018.

[2] Pete Florence, Corey Lynch, Andy Zeng, Oscar Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. *arXiv preprint arXiv:2109.00137*, 2021.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 2016.

[4] Innovations of AlphaGo. `https://deepmind.com/blog/article/innovations-alphago`. Accessed: 2021-11-28.

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-Level Control through Deep Reinforcement Learning. *Nature*, 2015.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 2018.

[8] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. *arXiv preprint arXiv:2004.04136*, 2020.

[9] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[10] Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

[11] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained Policy Optimization. In *International Conference on Machine Learning*, 2017.

[12] Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. Conservative safety critics for exploration. In *International Conference on Learning Representations*, 2021.

[13] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[14] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[15] Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The NetHack Learning Environment. *Advances in Neural Information Processing Systems*, 2020.

[16] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.

[17] Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *International Conference on Learning Representations*, 2021.

[18] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international Conference on intelligent robots and systems (IROS)*, 2017.

[19] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation*, 2018.

[20] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving Rubik's Cube with a Robot Hand. *arXiv preprint arXiv:1910.07113*, 2019.

[21] Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in RL is difficult: Epistemic POMDPs and implicit partial observability. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[22] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Thirty-second AAAI Conference on artificial intelligence*, 2018.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[24] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, 2018.

[25] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*, 2020.

[26] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. Frame Skip Is a Powerful Parameter for Learning to Play Atari. *AAAI-15 Workshop on Learning for General Competency in Video Games*, 2015.

[27] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 2017.

[28] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-Based Reinforcement Learning for Atari. *arXiv preprint arXiv:1903.00374*, 2019.

[29] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*, 2018.

[30] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning Latent Dynamics for Planning from Pixels. In *International Conference on Machine Learning*, 2019.

[31] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic Latent Actor-Critic: Deep Reinforcement Learning. *arXiv preprint arXiv:1907.00953*, 2019.

[32] Amir-massoud Farahmand. Action-Gap Phenomenon in Reinforcement Learning. *Advances in Neural Information Processing Systems*, 2011.

[33] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. On Inductive Biases in Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2019.

[34] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft Actor-Critic Algorithms and Applications. *arXiv preprint arXiv:1812.05905*, 2018.

[35] Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 1957.

[36] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, 2015.

[37] Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[38] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[39] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[40] Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

[41] Benjamin Eysenbach and Sergey Levine. Maximum Entropy RL (Provably) Solves Some Robust RL Problems. *arXiv preprint arXiv:2103.06257*, 2021.

[42] Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[43] Aravind S Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic Action Repetition for Deep Reinforcement Learning. *AAAI Conference on Artificial Intelligence*.

[44] Sahil Sharma, Aravind S. Lakshminarayanan, and Balaraman Ravindran. Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning. In *International Conference on Learning Representations*, 2017.

[45] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[46] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*, 2015.

[47] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[48] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. In *International Conference on Learning Representations*, 2018.

[49] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-Driven Exploration by Self-Supervised Prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017.

[50] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for Reinforcement Learning. In *International Conference on Machine Learning*, 2019.

[51] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv preprint arXiv:2005.01643*, 2020.

[52] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving Sample Efficiency in Model-Free Reinforcement Learning from Images. *arXiv preprint arXiv:1910.01741*, 2019.

[53] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[54] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task Deep Reinforcement Learning with PopArt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[56] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning*, 2018.

[57] Arash Tavakoli, Vitaly Levdik, Riashat Islam, Christopher M Smith, and Petar Kormushev. Exploring Restart Distributions. In *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, 2019.

[58] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[59] Tim Salimans and Richard Chen. Learning Montezuma's Revenge from a Single Demonstration. In *NeurIPS 2018 Deep RL Workshop*, 2018.

[60] Nir Lipovetzky, Miquel Ramirez, and Hector Geffner. Classical planning with simulators: Results on the atari video games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[61] Wilmer Bandres, Blai Bonet, and Hector Geffner. Planning with Pixels in (Almost) Real Time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[62] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-Imitation Learning. In *International Conference on Machine Learning*, 2018.

[63] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020.

[64] Scott Fujimoto, David Meger, and Doina Precup. Off-Policy Deep Reinforcement Learning without Exploration. In *International Conference on Machine Learning*, 2019.

[65] Henry Zhu, Justin Yu, Abhishek Gupta, Dhruv Shah, Kristian Hartikainen, Avi Singh, Vikash Kumar, and Sergey Levine. The ingredients of real world robotic reinforcement learning. In *International Conference on Learning Representations*, 2020.

[66] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[67] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. *Conference on Robot Learning*, 2021.

[68] Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

[69] Dibya Ghosh, Abhishek Gupta, Justin Fu, Ashwin Reddy, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals without reinforcement learning. *arXiv preprint arXiv:1912.06088*, 2019.

[70] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*, 2020.

[71] Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. PODS: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, 2021.

対外発表

- 橋本大世, 鶴岡慶雅. 深層強化学習における擬似的な行動による中間フレームの有効活用, 第 25 回ゲームプログラミングワークショップ, 2020.（ベストポスター賞）
- Taisei Hashimoto, Yoshimasa Tsuruoka. Utilizing Skipped Frames in Deep Reinforcement Learning via Pseudo-Actions, NeurIPS Deep Reinforcement Learning Workshop, 2020.
- 橋本大世, 鶴岡慶雅. リセット機能を活用したシミュレータにおける効率的な方策学習, 第 26 回ゲームプログラミングワークショップ, 2021.（研究奨励賞）

謝辞

本研究に取り組むにあたってたくさんの方々にお世話になりました.

指導教員である鶴岡慶雅教授には, 学部の頃から 3 年間に渡り様々な方面で大変お世話になりました. 研究の進め方, 論文の書き方, 発表の仕方など, 研究に必要なことを多く教えていただきました. また, 研究のアイディアを話した際には色々な関連研究を挙げていただき, そのおかげで正しい方向に進むことができたと思います. それと同時に, 強化学習だけでなく自然言語処理の研究もされているのに, どうやってそんなに多くの研究を把握されているのだろうと不思議に感じることもありました. 数々のご指導本当にありがとうございました.

博士課程の李凌寒さんは研究室の運営を積極的にしていただきました. コロナ禍により活動が制限される中, 会議など研究室の活動がより良くなるよう様々な工夫を率先してやってくださり, 本当に助けられました. 鶴岡研究室が回っているのは凌寒さんのおかげだと思っています. ありがとうございました.

ミーティングやその後の雑談では, 多くの方から自分の考えについて色々なコメントをいただきました. 特に同期の綿引隼人くん, 後輩の海野良介くんからはよく鋭い指摘をしていただいた印象があります. 二人には直接会ったことは数えるほどしかないと思いますが, 強化学習に関する最新の研究や実装のティップスなど様々なことを話して刺激を受けていました.

論文を執筆する際はいつも, 多くの方に添削をしていただきました. 特に李凌寒さん, 博士課程の安井豪さんには, 英語論文を書いた際に私の拙い英語を根気強く修正していただきました. その結果人に読んでもらえるものが書けたかなと思っています.

最後に, 長い学生生活を通して家族には本当にお世話になりました. おかげで研究に専念し, こうして論文を書き上げることができました. これからもよろしくお願いします.

改めて, お世話になったすべての方に深く感謝いたします. ありがとうございました.