強化学習法における n-step return の改良

中根 勇樹[1]　片山 晋[1]　椋木 雅之[1]

概要：本研究では強化学習法における n-step return の改良方法を提案する．n-step return は，学習に用いる利得の目標値を予測する際に，実際の行動系列の情報を最大限に利用することにより，予測の精度を上げる．本研究では，選択した行動が greedy policy と一致した間の利得を使用するよう n-step return を改良した．2D 対戦格闘ゲームのキャラクター制御を題材に，その学習の効率を評価した結果，従来の DDQN の学習が進まず，提案手法の n-step return を組み込んだ DDQN の学習速度が最も速い傾向にあることが分かった．

キーワード：強化学習，ゲームにおける学習，DDQN，n-step return，格闘ゲーム，NPC 向け

Improving n-step return in reinforcement learning

Abstract: In this study, we propose an improvement method for the n-step return in reinforcement learning. The n-step return enhances the accuracy of predicting the target value of gains used in learning by maximizing the utilization of information from the actual action sequence. In this research, we refined the n-step return to utilize gains between consecutive actions that align with the greedy policy. Using 2D competitive fighting game character control as a case study, we evaluated the efficiency of the learning process. The results indicated that conventional DDQN learning stagnated, while the learning speed of DDQN incorporating the proposed n-step return tended to be the fastest.

Keywords: Reinforcement Learning, Learning in Games, DDQN, n-step return, Fighting Game, NPC-specific

1. はじめに

n-step return は DDQN などの強化学習を加速させる物である．DDQN などの強化学習では、最も価値が高いと思われる行動選択（greedy policy）以外に、一定確率でランダムな行動を取り入れることで、学習に多様性を持たせている。しかし、従来の n-step return は，行動を選択する際に greedy policy 以外のランダムな行動を選択をしている間の報酬も利用してしまうため，greedy policy に基づいた行動価値関数を学習する上で，必ずしも適切ではない．greedy policy に従っている間の実際の報酬を使用することで性能が改善することが期待できる．本研究では，選択した行動が greedy policy と一致した間の報酬の累積を使用するよう n-step return を改良し，2D 対戦格闘ゲームのキャラクター制御を題材に，その学習の効率を評価した．

[1] 宮崎大学 大学院 工学研究科

2. 関連研究・関連システム

2.1 格闘ゲーム

2D 対戦格闘格闘ゲーム [1] とは，ジャンプやしゃがみなどの基本行動，攻撃やガード，コマンド入力方式の必殺技などを駆使して，相手の体力を削りあって，最終的な体力の差を競うゲームである．勝敗は，主に以下の 2 種類がある．

- ノックアウト
  - 体力が尽きてしまう
- タイムオーバー
  - 設定されている時間が 0 になった時，残っている体力の多い方が勝ち

これらに加え，作品によっては，ステージの端より外に押し出されてしまった場合負けとなるリングアウトも採用されている．2D 対戦格闘ゲームの代表例として，1991 年に

© 2024 Information Processing Society of Japan

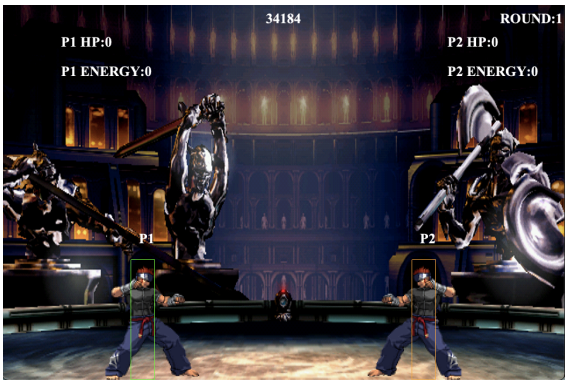

図 1 FightingICE[2]
Fig. 1 FightingICE[2]

株式会社カプコンから発売された「ストリートファイターⅡ」などがある．

2.2 FightingICE

FightingICE[2] とは，2013 年に ICE[*1] が開発した AI 研究用対戦型格闘ゲームプラットフォームである (図 1)．FightingICE はゲームを進行させる対戦マネージャと対戦するキャラクターを制御する 2 つの AI で構成される．対戦マネージャは 2 体のキャラクタからなる現在のゲーム状況を AI に送り，AI は送られてきたゲーム状況を元に各キャラクタの次の行動を決定し対戦マネージャに送る．対戦マネージャは 2 つの AI から行動を受け取った後にゲーム状況を更新し，再び AI に新たなゲーム状況を送る．このようなメッセージの通信を繰り返してゲームが進行する [3]．FightingICE は，Java プログラミングおよび，AI を開発するために Py4J による Python プログラミングをサポートする．このプラットフォームを使用すると，使用中のアルゴリズムに応じて，ユニークな攻撃，コンボ，動きなどを行うことができるため，柔軟な AI を設計し，NPC[*2]での制御に使用するための AI アルゴリズムを作成できる [2]．

2.3 Gym-FightingICE

Gym-FightingICE[4] は，OpenAI Gym のための，FightingICE の公式 API である．Gym-FightingICE には 4 つの環境が提供されている．本研究では，FightingiceDataFrameskip-v0 を使用した．

2.4 DQN(Deep Q Network)

DQN[5] は将来に得られる報酬の総和の最大化を目的とする Q-learning に，ニューラルネットワークを応用したものである．将来に得られる報酬の総和は次の式で表される割引利得 R_t によって定義される．

$$R_t \equiv \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

この式において r_t は時刻 t に環境から得られる報酬である．γ は割引率と呼ばれ，未来の報酬を定める定数である．

Q-learning[6] は，状態 s で行動 a をとり，その後最適に行動した場合の割引利得を $Q(s,a)$ で推定する．DQN は各行動の価値関数 $Q(s,a;\theta)$ の各行動毎の価値 (利得の推定値) のベクトルを出力する多層ニューラルネットワークである．θ はネットワークのパラメータを表している．

DQN アルゴリズムの 2 つの重要な要素は，ターゲットネットワークの使用と experience-replay の使用である．ターゲットネットワークのパラメータ θ^- は，エージェントが各時間ステップでの行動価値や方策を推定するのに使用されるニューラルネットワーク (オンラインネットワーク) から τ ステップごとにコピーされる．experience-replay は，学習データ間の相関関係による学習への悪影響の対処として，得られた経験をリプレイバッファ D に蓄積し，学習時に D からランダムにサンプリングした経験を用いる．エージェントが学習の際に目指すべき目標となる値 (ターゲット) を算出する際，時刻 t の報酬 $\equiv r_t$，ターゲットネットワークの時刻 t でのパラメータの推定値 θ_t^- が使われる．式は以下を使用する．

$$Y_t^{DQN} \equiv r_t + \gamma \max_a Q(s_{t+1}, a; \theta_t^-) \qquad (1)$$

Volodymyr Mnih らの実験 [5] では，DQN エージェントは，Atari2600 の 49 個のゲームのセット全体でプロの人間ゲームテスターと同等のレベルの性能を示し，ゲームの半分以上で人間のスコアの 75%以上のスコアを記録した．

2.5 Double Q-learning

Q-learning や，DQN の式 (1) の最大化演算子 max は，同じ値を使用して行動を選択し，評価するため，過大評価された値を選択する可能性が高くなり，過度に楽観的な値の推定値が発生する．Double Q-learning [7] は，この問題を防ぐために，選択肢を評価から切り離すというアイデアのもと，van Hasselt らによって提唱された方法である．Double Q-learning は，θ と θ' の 2 つのパラメータが用意されている．各パラメータは，他のパラメータからの値で次の状態について更新される．Double Q-learning には，更新ごとに，1 組のパラメータを使用して greedy policy を決定し，もう 1 組のパラメータを使用してその値を決定する．そのため，θ と θ' の 2 セットを学習する．

θ の学習は以下の式をターゲットにしている．

$$Y_t^{DoubleQ\theta} \equiv r_t + \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \theta_t); \theta_t')$$

θ' の学習は，上式の θ' と θ を入れ替えた以下の式をター

[*1] INTELLIGENT COMPUTER ENTERTAINMENT LAB., RITSUMEIKAN UNIVERSITY
[*2] コンピュータが操作するキャラクター

© 2024 Information Processing Society of Japan

ゲットにしている．

$$Y_t^{DoubleQ\theta'} \equiv r_t + \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \theta_t'); \theta_t)$$

van Hasselt らの実験では，ルーレットゲームと迷路問題で，Q-learning と比較して，Double Q-learning がはるかに学習が速く，良い性能を記録した．

2.6 DDQN(Double Deep Q Network)

DDQN[8] は，van Hasselt らにより提唱された方法で，DQN と Double Q-learning の両方を参照して，提案されたアルゴリズムである．DDQN は，Double Q-learning と比較して，現在の greedy policy の評価のための第 2 のネットワークのパラメータ θ' を，ターゲットネットワークのパラメータ θ^- に置き換えたものである．

$$Y_t^{DDQN} \equiv r_t + \gamma Q(s_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+1}, a; \theta_t); \theta_t^-)$$

ターゲットネットワークのアップデートは，オンラインネットワークの定期的なコピーによって行われる．van Hasselt らは，Atari2600 の複数個のゲームにおいて，DQN が過大評価によってスコアを低下させているのに対し，DDQN はより安定的で信頼性の高い学習をもたらすことができることを示した．

DDQN のアルゴリズムを Algorithm 1 に示す．

Algorithm 1 DDQN(Double Deep Q Network)

1: experience-replay バッファ D を容量 N で初期化
2: 行動選択のための Q ネットワークをランダムなネットワークパラメータ θ で初期化
3: 価値計算のための Q ネットワーク θ^- を θ で初期化
4: **for** $episode = 1, M$ **do**
5: 環境を初期化，初期状態を s_1 とする．
6: **for** $t = 1, T$ **do**
7: $a_t \leftarrow \begin{cases} random \quad a & \varepsilon\text{の確率} \\ \underset{a}{\operatorname{argmax}}\, Q(s_t, a; \theta) & \text{その他} \end{cases}$
8: 行動 a_t を実行
9: 報酬 r_t，次の状況 s_{t+1} を受け取る
10: D に $\{s_t, a_t, r_t, s_{t+1}\}$ を記録
11: **end for**
12: 1 試合中に出した行動の個数分 D からランダムにミニバッチ $\{s_j, a_j, r_j, s_{j+1}\}$ を取得
13: **for** ミニバッチのすべての要素において **do**
14: $target_j \leftarrow$
$\begin{cases} r_j & s_{j+1}\text{が試合終了状態} \\ r_j + \gamma Q(s_{j+1}, \underset{a}{\operatorname{argmax}}\, Q(s_{j+1}, a; \theta); \theta^-) & \text{その他} \end{cases}$
15: θに関して $(target_j - Q(s_j, a_j; \theta))^2$ を最小化する勾配法を適用
16: **end for**
17: episode=3 ごとに θ^- を θ のコピーによってアップデート
18: **end for**

2.7 *n*-step return

n-step return は，実際の報酬を用いるという考えのもと，Matteo Hessel らによって提案された rainbow アルゴリズム [9] の一部として提唱された方法である．n-step return を以下のように定義する．

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma^k r_{t+k}$$

DQN の n-step 版の損失を以下のように定義する．

$$(R_t^{(n)} + \gamma^n \max_a Q(s_{t+n}, a; \theta^-) - Q(s_t, a_t; \theta))^2$$

n を適切に調節することで，学習を高速化することができる．Matteo Hessel らの実験では，rainbow アルゴリズムにおいて，$n = 3$ とすることで，アーケード学習環境 [10] の 57 の Atari2600 ゲームのすべてのエージェントにおいて最高の性能を発揮した．

本研究で提案する手法は，より一般的なアルゴリズムである Retraceλ[11] の特定の場合と考えることができる．しかし，1 ステップあたりの計算が複雑なので，本研究ではより単純な方法で改善できないかと考え，n-step return を改良した．

3. 提案手法

提案手法では，n-step return で使用する n を greedy policy を行なっている間のステップ数，また，エピソードの終端までと設定する．rainbow アルゴリズムでは，n-step return で使用する n の値を固定にしていたが，n を可変にすることによって n を固定する場合よりも正確な報酬を学習で使用することが可能であると考えた．実際は，Algorithm 2 の通り n の値を決定する．この n-step return と，DDQN を組み合わせたターゲットの式を以下のようにする． *3

$$Y_t^{DDQN+n\text{-step return}} \equiv R_t^{(n)} + \gamma^n Q(s_{t+n}, \underset{a}{\operatorname{argmax}}\, Q(s_{t+n}, a; \theta_t); \theta_t^-)$$

全体的なアルゴリズムを Algorithm 3 に示す．Algorithm 3 では，1~3 でオンラインネットワークのパラメータ θ とターゲットネットワークのパラメータ θ^- を初期化した後，6~10 で ε-greedy 法によって行動の選択を行いながら試合を進行する． ε-greedy 法は、エージェントが行動を選択する際に，確率 $\varepsilon (0 \leq \varepsilon \leq 1)$ でランダムな行動を選択し，確率 1-ε で現在の最適な行動を選択する．これにより，エージェントは，一定の確率で未知の行動を試みることができ，その行動によって得られる報酬を考慮することができる．状態 s_t，行動 a_t，報酬 r_t，行動した次の状態 s_{t+1} は experience-replay バッファ D に記録され

*3 ここでの r，j は experience-replay バッファ D の中の要素である．

© 2024 Information Processing Society of Japan

る．試合が終了したら 12∼20 で experience-replay を行う．experience-replay はランダムにバッファ D からサンプリングを行い，そのサンプリングされた経験を使用して学習を行う．これにより，様々な行動を学習することができる．22 で 3 試合毎にオンラインネットワークのパラメータ θ のコピーによってターゲットネットワークのパラメータ θ^- をアップデートする．

Algorithm 2 n の選定方法

```
t=ランダムに選択したミニバッチ取得先 D の配列番号
n=0
while a_{t+n}が存在している間 do
    if a_{t+n+1} が存在しない or a_{t+n+1} が次の試合の要素 then
        return n
    else if a_{t+n+1} がランダムに選ばれている then
        return n
    end if
    n ← n + 1
end while
```

4. 実験

4.1 概要

Fighting ICE を用いて，n-step return の改良手法について実験により検証を行った．比較する条件は以下の通りにする．

- n-step return を使わない ($n = 1$)
- n-step return の n を固定にする ($n = 3$)
- n-step return の n を可変にする

4.2 設定

以下の設定で実験を行なった．

- 使用キャラクターを ZEN とする．
- 1 試合の最大の長さは 60 秒とする．
- Q ネットワークの学習係数を 0.1 とする．
- 割引係数 γ を 0.8 とする．
- ε-greedy で使う ε の値を 0.1 とする．
- 3 試合毎に experience-replay を実行する．
- experience-replay バッファ D の最大要素数は 10000 で，キューの構造とする．
- DDQN は，3 試合ごとに価値計算を行うニューラルネットワークを更新する．
- 報酬は，自分の攻撃が当たった場合，そのダメージ値/400 のプラスの報酬，自分が攻撃を受けた場合，そのダメージ値/400 のマイナスの報酬を受ける．相打ちの場合は，それらの差を用いる．
- ニューラルネットワークは，表 1 の構成とした．
- キャラクターの状態によって可能な行動を設定した [12].
- 学習に使うデータを定めた [12].
- 各設定で，学習 1000 試合を異なるニューラルネットワークの初期値で 5 回ずつ行う．
- 評価は，10 試合毎の学習データを使用し，それぞれ 3 試合ずつ greedy policy で試合を行わせる．
- 学習における対戦相手は Jerry MizunoAI[13] を使用した．

Algorithm 3 DDQN(Double Deep Q Network)+n-step return

```
experience-replay バッファ D を容量 N で初期化
行動選択のための Q ネットワークをランダムなネットワークパラメータ θ で初期化
価値計算のための Q ネットワーク θ^- を θ で初期化
for episode = 1, M do
    環境を初期化，初期状態を s_1 とする
    for t = 1, T do
        a_t ← { random  a                 εの確率
              { argmax_a Q(s_t, a; θ)     その他
        行動 a_t を実行
        報酬 r_{t+1}，次の状態 s_{t+1} を受け取る
        D に (s_t, a_t, r_t, s_{t+1}) を記録
    end for
    1 試合中に出した行動の個数分 D からランダムにミニバッチ {s_j, a_j, r_j, s_{j+1}} を取得
    for ミニバッチのすべての要素において do
        if a が期待報酬値の予測によって選ばれていた場合 then
            Algorithm 2 の通り n を決める
            target_j ←
                { r_j + γr_{j+1} + γ^2 r_{j+2} + ∼ +
                        γ^n r_{j+n}                    s_{j+n+1}が試合終了状態
                { r_j + γr_{j+1} + γ^2 r_{j+2} + ∼ + γ^n Q(s_{j+n},
                        argmax_a Q(s_{j+n}, a; θ); θ^-)   その他
        else
            target_j ←
                { r_j                                          s_{j+1}が試合終了状態
                { r_j + γQ(s_{j+1}, argmax_a Q(s_{j+1}, a; θ); θ^-)   その他
        end if
        θに関して(target_j − Q(s_j, a_j; θ))^2 を最小化する勾配法を適用
    end for
    episode=3 ごとに θ^- を θ のコピーによってアップデート
end for
```

表 1 ニューラルネットワークの構成

	入力数	出力数	活性化関数
入力層	-	88	-
中間層 1	88	42	sigmoid
中間層 2	42	42	sigmoid
出力層	42	42	-

© 2024 Information Processing Society of Japan

4.3 使用した環境

以下の環境で実験・評価を行った.

- Fighting ICE の環境
  - Fighting ICE 4.50
  - FightingiceDataFrameskip-v0
- コンピュータの環境
  - Ubuntu 20.04.5 LTS
  - Intel Core i5-10500 3.10GHz(6 cores)
  - NVIDIA Quadro P1000

4.4 実験結果

各条件の greedy policy の評価の結果を図 2∼4 にまとめ，それら 3 つのグラフをを図 5 にまとめた．図 2 は n-step return を組み込んでいない DDQN，図 3 は $n = 3$ で固定した n-step return を組み込んだ DDQN，図 4 は提案手法であり，n を可変にした n-step return を組み込んだ DDQN である．グラフの縦軸は，10 試合毎のモデルを使って greedy policy で行わせた計 15 試合で得られた報酬の平均を表しており，横軸は試合数になっている．薄い色は，各線に対応した標準偏差の範囲である． 3 つの条件の greedy policy での評価を比較すると，n-step return を組み込んでいない DDQN は，試合回数を重ねてもグラフは横ばいになっており，あまり学習が進んでいない．n-step return の n を固定にしたものと可変にした 2 つは，グラフが右肩上がりになっており，学習が進んでいると考えられる．提案手法の n を可変にした n-step return と，rainbow の一部として組み込まれていた $n = 3$ で固定した n-step return を比較すると，$n = 3$ で固定した n-step return は試合数 50 以降で，0.5 前後の報酬値となるが，グラフの振れ幅が大きい．それに対し，提案手法の n を可変にした n-step return は，試合数 50 前後で，0.6 前後の報酬値となり $n = 3$ 固定の場合より多くの報酬を得ている．また，グラフの振れ幅が少ない．この結果から，n-step return の n を可変とすることで，学習の速度を改善できると考えられる．

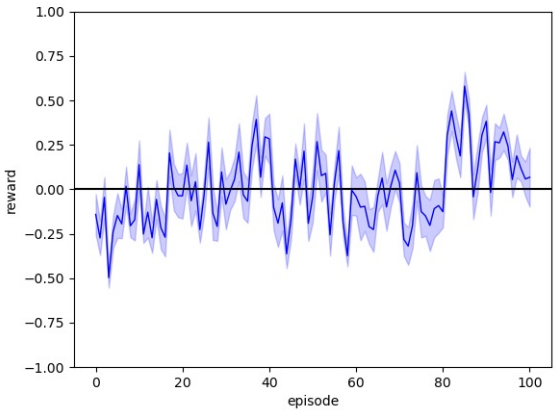

図 2 n-step return を組み込んでいない DDQN の性能評価
Fig. 2 Performance evaluation of DDQN without incorporating n-step return.

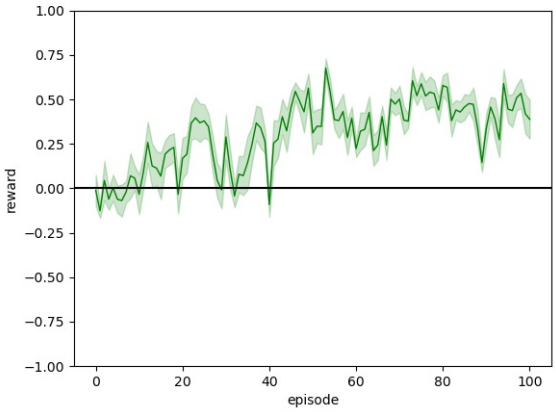

図 3 $n = 3$ とした n-step return を組み込んだ DDQN の性能評価
Fig. 3 Performance evaluation of DDQN incorporating n-step return with n=3.

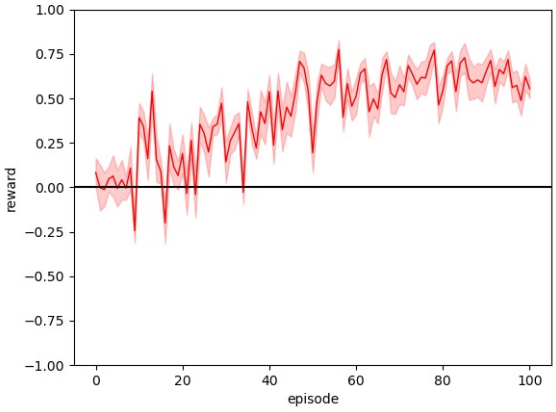

図 4 n を可変にした n-step return を組み込んだ DDQN の性能評価 (提案手法)
Fig. 4 Performance Evaluation of DDQN with Variable n-step Return (Proposed Method)

© 2024 Information Processing Society of Japan

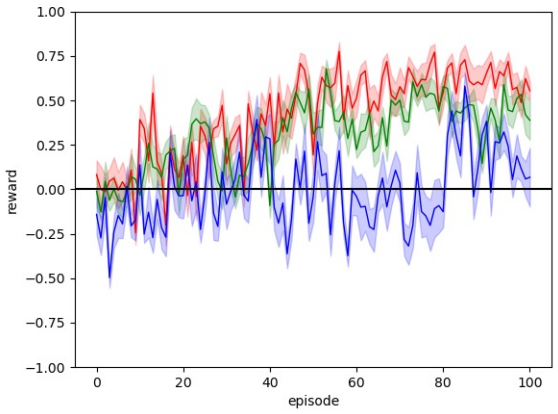

図 5 全条件のグラフの比較
Fig. 5 Comparison of graphs under all conditions.

5. まとめ

本研究では，学習で greedy policy に従っている間に実際の報酬を使用することことが有効であるというアイデアに基づき，格闘ゲームの環境下で，n-step return の n を可変にする手法を提案した．n-step return を組み込んだ DDQN は，格闘ゲームの環境下で，従来の DDQN と比較して，学習の改善が見られた．n-step return については，n を可変にすることで，n を固定するよりも，速く学習できる傾向があることが分かった．対照的に，工夫を施さなかった DDQN は，本研究の条件下での学習が困難であることが分かった．

本研究では，対戦相手を 1 つの AI に固定しており，試行回数は，各条件で 1000 試合を 5 回ずつ行なったため，別の消極的な AI など，対戦相手の違いによる学習の変化を，試行回数を増やして調査する必要がある．他にも，DDQN に n-step return を組み込む工夫だけをしているため，行動選択や，学習部分に他の工夫を施し，工夫ごとの学習の変化を比較する必要がある．

参考文献

[1] Wikipedia. 対戦型格闘ゲーム — Wikipedia, the free encyclopedia. https://ja. wikipedia.org/wiki/対戦型格闘ゲーム , 2023.
[2] DareFightingICE Competition. `https://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-2.html`.
[3] 山岡 勇太保木 邦仁. CFR 法と Q 学習法の格闘ゲーム人工知能への適用. 研究報告ゲーム情報学 (GI), 第 6 巻, pp. 1–8, 2020.
[4] Official gym API for game FightingICE. `https://github.com/TeamFightingICE/Gym-FightingICE`.
[5] Volodymyr Mnih et.al. Human-level control through deep reinforcement learning. Vol. 518 of *Nature*, pp. 529–533, 2015.
[6] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. Vol. 8 of *Machine Learning*, pp. 279–292, 1992.
[7] Hado Hasselt. Double Q-learning. Vol. 23 of *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.
[8] Hado van Hasselt et.al. Deep Reinforcement Learning with Double Q-Learning. AAAI, pp. 2094–2100, 2016.
[9] Matteo Hessel et.al. Rainbow: Combining Improvements in Deep Reinforcement Learning. AAAI, pp. 3215–3222, 2018.
[10] M. G. Bellemare et.al. The Arcade learning Environment: An Evaluation Platform for General Agents. Vol. 47 of *Journal of Artificial Intelligence Research*, pp. 253–279, 2013.
[11] Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and Efficient Off-Policy Reinforcement Learning. Vol. 29 of *Advances in Neural Information Processing Systems*, 2016.
[12] 中根勇樹. 格闘ゲームにおける DDQN の改良. 宮崎大学工学部 情報システム工学科 卒業論文, 2023.
[13] Sample AIs for Version 4.00 or later (available on March 6, 2018). `http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/2018_Sample_AIs.zip`.

© 2024 Information Processing Society of Japan