

修 士 論 文

好奇心に基づく内部報酬を用いた強化 学習によるローグライクゲームの学習

Reinforcement Learning with Curiosity-Based
Rewards in Rogue-like Games

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 48-186422 加納 由希夫

提 出 日

2020 年 1 月 29 日

概要

近年、ゲーム AI に関する研究分野では、ビデオゲームなどの複雑な環境を主な対象として、深層学習の技術を強化学習のアルゴリズムに取り入れた深層強化学習の手法を適用する研究が注目を集めており、実際に様々なゲームのタスクにおいて人間を超えるほどの性能を持つ AI を学習することができているが、現状では様々な理由で、より複雑な性質を持つゲームや実世界の環境を対象とした場合に強化学習が難しいことが指摘されている。その理由の一つに、強化学習を行う上での前提であり、学習効率にも大きな影響を与え得る報酬関数に関して、その設計が複雑な環境においては非常に難しくなることが挙げられる。そこで近年では、この問題に対処するために、環境の状態観測の新規性を内部報酬として用いる好奇心ベースの報酬生成手法の研究が注目されている。本研究では、より現実の問題に近い設定を持つゲームとして、コンピュータ RPG の一種であるログライクゲームを挙げる。ログライクゲームは非常に複雑な環境を持つだけでなく、環境のランダム性や部分観測性など様々な要因により、単に強化学習の手法を適用するだけでは学習することが難しいことが知られているが、そこで、ログライクゲームを効率的に学習するために好奇心に基づく内部報酬を用いた強化学習の手法を適用することを提案し、オリジナルのログライクゲーム環境による評価実験により、提案手法がより効率的に環境の学習を進められることを確認した。

目次

第 1 章	序論	1
1.1	背景	1
1.2	本研究の目的と貢献	3
1.3	本稿の構成	3
第 2 章	背景	5
2.1	強化学習	5
2.2	価値反復に基づくアルゴリズム	6
2.2.1	Q 学習	8
2.2.2	Deep Q-Network (DQN)	9
2.3	方策勾配に基づくアルゴリズム	11
2.3.1	方策勾配定理 (Policy Gradient Theorem) の証明	12
2.3.2	Proximal Policy Optimization (PPO)	14
2.4	強化学習と報酬設計	17
2.5	ログライクゲーム	18
第 3 章	好奇心ベースの報酬生成手法	21
3.1	Intrinsic Curiosity Module (ICM)	21
3.2	Random Network Distillation (RND)	24
第 4 章	提案手法	26
4.1	好奇心ベースの手法によるログライクゲームの学習の提案	26
4.2	評価実験用のゲーム環境	27
4.2.1	概要	27
4.2.2	エピソードの定義	28
4.3	好奇心ベースの報酬生成手法の適用	30
4.3.1	提案モデルの概要と実験設定	30
4.3.2	実験結果	34
4.3.3	考察と課題	38
第 5 章	結論	42
5.1	まとめ	42

5.2 今後の課題	42
---------------------	----

目次

2.1	強化学習における、環境とエージェントの相互作用 [1]	6
2.2	Rogue のスクリーンショット	19
3.1	Intrinsic Curiosity Module (ICM) のアーキテクチャの概要	23
3.2	Random Network Distillation (RND) のアーキテクチャの概要	24
4.1	生成されるダンジョンの形状の例	27
4.2	プレイヤーによるダンジョン探索の例	28
4.3	PPO と RND による学習モデル	30
4.4	PPO のネットワーク設定	32
4.5	RND のネットワーク設定	32
4.6	実験結果：ゲームクリア率の推移 (easy) ($N_d = 50$)	35
4.7	実験結果：エピソードの長さの推移 (easy) ($N_d = 50$)	35
4.8	実験結果：ゲームクリア率の推移 (hard) ($N_d = 50$)	36
4.9	実験結果：エピソードの長さの推移 (hard) ($N_d = 50$)	36
4.10	実験結果：ゲームクリア率の推移 ($N_d = 500$)	37
4.11	実験結果：エピソードの長さの推移 ($N_d = 500$)	37
4.12	好奇心ベースのアルゴリズムによる問題の例	40

表 目 次

4.1 実験時のハイパーパラメータ設定	34
---------------------------	----

第1章 序論

1.1 背景

近年、ニューラルネットワークを活用した深層学習技術の発展に伴って、機械学習や人工知能に関する分野の研究が非常に注目を集めるようになってきている。その背景には、ニューラルネットワークの性質として非線形な関数を扱えることや転移学習が容易であることなどの利点があるだけでなく、ニューラルネットワークの膨大な量の計算を可能とするほどの計算機性能の向上や、Tensorflow [2] や Pytorch [3] などの良質な深層学習用の公開ライブラリの存在もあるだろう。深層学習に関する研究は、特に自然言語処理や画像処理などの分野で盛んに行われているが、近年では、深層学習と強化学習を組み合わせた深層強化学習と呼ばれる枠組みにおいて、優れたコンピュータゲームプレイヤーの実現を目指すゲーム AI の研究も注目を浴びつつある。Google の DeepMind 社らが発表した囲碁 AI として、エキスパートによる棋譜を使用した教師あり学習と自己対戦による強化学習とを組み合わせた手法で訓練された AlphaGo [4] が、当時の世界最強クラスのプロ棋士に勝利するほどの偉業を成し遂げたことも記憶に新しい。強化学習、及びゲーム AI に関する研究の目的の一つには、より複雑性質を持つ現実世界における問題の解決に応用することが考えられる。ゲームは、行動の制約や報酬が得られる条件などが厳密に定められており、計算機上で環境をシミュレートすることが可能であるため非常に扱いやすく、また、学習の結果を客観的に評価することが容易であるため、人工知能の研究で扱うタスクとして非常に優れていると考えられる。このような特徴を持つ様々なゲームの環境において、汎用的に高い性能を発揮するゲーム AI を学習する手法が見つかれば、その手法を現実世界におけるより複雑な問題に応用することで、その問題解決に大きく貢献できる可能性がある。

古典的なゲーム AI には、将棋や碁、チェス [5]、オセロ [6] などのボードゲームを対象にしたものが多い。これらは二人零和有限確定完全情報ゲームであるため、ゲーム環境の状態遷移規則が予め明らかであり、ゲーム木を展開することによって最善手を探索することが可能であった。しかしながら、現実世界における問題は基本的にこの状態遷移規則が未知であることが多く、木構造を用いた探索手法を応用して用いることは難しい。また、最善手の決定には局面の良し悪しを判断する評価関数を利用することが一般的であったが、この評価関数の決定には、ゲームの熟練者の試行錯誤による関数設計や、プロの対局データを用いた機械学習など、何らかの事前知識を用いることが不可欠であった。

そこで、ゲーム AI の分野で近年注目を集めているのが強化学習の手法である。強化学習とは機械学習の枠組みの一種であり、学習の対象となる環境と自律的に行動するエージェントとの相互作用の中で、エージェント自ら試行錯誤しながら最適な行動方策を学習するような理論的枠組みである。強化学習の利点は、環境の状態遷移規則が未知で、人間による教師データやドメイン知識などの事

前知識が無いような場合においても、エージェントは試行錯誤の中で環境から受け取ることができる報酬を最大化するような行動方策を自律的に学習できるという点にある。従来の強化学習の手法では、メモリサイズや計算量などの物理的な制約から状態数が大きい環境を扱うことが難しかったが、2013年に Mnih らが発表した Deep Q-Network (DQN) [7, 8] と呼ばれる深層学習と強化学習を組み合わせた手法により、ゲームの画面の入力をそのまま環境の状態として強化学習で扱えるようになった。DQN による学習で Atari 2600 のゲームシリーズの一部において人間を上回るスコアを記録した事実も相まって、DQN 以降はビデオゲームを環境として扱う深層強化学習に関する研究が盛んに行われており、様々な手法が提案されている [9, 10, 11, 12, 13, 14, 15, 16, 17, 18]。

近年では、深層強化学習のアルゴリズム技術の発展により、多くのビデオゲームにおいてより高い性能を持つ AI を学習できるようになった。しかしながら、例えば Atari 2600 における Ms. Pac-Man や Montezuma's Revenge などの複雑な環境を持つようなビデオゲームにおいては、単純に DQN などの手法を適用しただけは学習が安定しない問題が指摘されている [19, 20]。この原因、及び解決のアプローチとして、強化学習における「報酬」の構造を見直す研究がなされている。2017年に Seijen らは、Ms. Pac-Man においてドメイン知識を活用することで、ゲームを特徴付けているアイテムごと報酬を分解し、それぞれに対してモデルを学習させるという手法を提案し、高い成果を示している [19]。同じく 2017年に Pathak らは、単純な報酬関数の実装では環境からエージェントに与えられる報酬が疎 (sparse) になりやすいことに着目し、環境側で用意されている外部報酬とは別に、エージェントの好奇心に基づいた内部報酬を計算し、それを密な報酬関数として外部報酬と組み合わせて学習に利用することでより効率的に学習ができるようになるとして、Intrinsic Curiosity Module (ICM) という手法を提案した [21]。翌年 2018年には、Burda らにより ICM を改良した手法として Random Network Distillation (RND) [22] を提案しており、RND の生成する好奇心の内部報酬を用いた強化学習により Montezuma's Revenge において当時の最高のスコアを記録した。

ゲーム AI の研究分野で発展してきた強化学習の技術を現実世界における複雑な問題解決に応用する際に問題となると予想されることの一つに、現実世界における報酬設計の難しさが挙げられる。強化学習のエージェントは、環境から受け取る報酬信号を最大化するような方策を学習するため、学習される方策は環境の持つ報酬関数に依存する。そのため、現実世界における問題を考える際には、この報酬関数の設計も課題の一つになる。その際、特に問題になると考えられるのが報酬の sparse 性である。強化学習においては、エージェントが環境から意味のある報酬を受け取れる機会が少なくなってしまうと、環境の探索が効率的に行えずに、学習も不安定になってしまう場合がある。しかし、エージェントにとって意味のある報酬を、高い頻度で与えることができるような報酬関数を設計することは、非常に困難なタスクであることが指摘されている [20]。好奇心ベースの報酬生成手法である ICM [21] や RND [22] は、この問題を解決するためのアプローチとして提案された手法である。

本稿では、より現実の問題に近い設定を持つゲームとして、コンピュータ RPG の一種であるローグライクゲームを学習する対象の環境として選択した。ローグライクゲームとは、1980年のRogueから派生したゲームの総称であり、日本国内では不思議のダンジョンシリーズなどの名で知られている。シンプルでありながら奥深いゲームシステムを持ち、ゲームクリアには非常に高い知的能力が要求される難易度の高いゲームである。ローグライクゲームを対象とした学術研究は少なく、未だに十分な成果は挙げられていないものの、多くの先行研究 [23, 24, 25, 26, 27, 28] がローグライク

ゲーム AI を研究することの意義を唱えている。本研究では、ログライクゲームの AI を学習する際に克服すべき課題として、POMDP 性（環境の状態を一部しか観測することができず、状態と観測が一致しないような環境）とランダム性（ゲーム内の様々な要素がランダムに生成・配置されること）、報酬の sparse 性（報酬がエージェントに与えられる機会が少ないこと）の 3 つの性質を挙げ、これらを解決するための手法アプローチを提案する。また、評価実験用に作成した簡単なログライクゲームの環境も同時に提案する。このログライクゲームの環境における評価実験にて、提案手法の有効性を確認した。

1.2 本研究の目的と貢献

本研究の目的は、コンピュータ RPG の一種であるログライクゲームに対して深層強化学習を適用し、より高い性能を持つコンピュータゲームプレイヤーを実現させることである。しかし、環境のランダム性や POMDP 性、報酬の sparse 性などを始めとして、ログライクゲーム特有の性質を由来とする様々な要因が、強化学習によるゲーム AI の獲得を難しくしており、既存の手法ではアイテムやモンスターなども存在する一般的なログライクゲーム環境における学習は非常に難しく、それらの要素を排除してダンジョンの探索に特化したタスクを扱う先行研究が多くを占めているものの、このダンジョン探索タスクにおいても未だに十分な結果は得られていない。だが、ログライクゲームを効率的にプレイするためには、効率的なダンジョン探索能力が必要不可欠なものとなってくる。そこで本稿では、アイテムやモンスターなどの複雑な要素を排除したログライクゲームの環境において、ダンジョンをより効率的に学習できるような手法の実現を目的とした。

Rogue を扱った Asperti らの先行研究 [28] は、ログライクゲームにおけるアイテムやモンスターなどの要素を排除するだけでなく、ログライクゲーム特有の環境を部分的にしか観測できないシステムの要素を排除しており、実験においては高い成果を示しているものの、ログライクゲーム探索 AI としは不十分な部分があると考えられる。そのため、本稿ではログライクゲームの部分観測性を保持した環境における効率的なダンジョン探索 AI の獲得を目指した。

本研究で提案する手法は、ログライクゲームを対象とした強化学習に、Random Network Distillation (RND) [22] による好奇心に基づく内部報酬を適用することである。強化学習の手法としては、Proximal Policy Optimization (PPO) [18] によるマルチエージェントの分散学習手法を用いた。既存手法の組み合わせではあるが、特に好奇心ベースの報酬生成手法をログライクゲームの学習に適用したという点で、先行研究にはない新規性を見出すことができる。評価実験にて、ログライクゲームの学習に RND が生成する好奇心の内部報酬を用いることの有効性を確認した、

1.3 本稿の構成

本稿の構成を以下に述べる。まず第 2 章では、本稿の内容を理解する上で必要となる前提知識について述べる。2.1 節では強化学習に関する基礎知識を説明し、2.2 節では価値反復に基づく強化学習のアルゴリズムについて、2.3 節では方策勾配に基づく強化学習のアルゴリズムについて、基礎的

な内容の説明と代表的な深層強化学習の手法の紹介をする。2.4 節では、強化学習における報酬をどのように設計すべきなのかということについて述べる。2.5 節では、本稿のテーマであるログライクゲームに関しての説明を述べる。続いて第 3 章では、本稿で注目している好奇心ベースの報酬生成手法について述べる。強化学習における探索をより効率的に行うために提案された、好奇心の内部報酬を生成する手法と、その発展について紹介する。第 4 章では、本稿における提案手法について述べる。4.1 節では、ログライクゲームを学習するために、本稿で提案するアプローチについて述べる。4.2 節では、評価実験用に実装したログライクなゲーム環境についての説明を述べ、4.3 節と 4.4 節では、それぞれのアプローチのモデルやアルゴリズムの説明と、評価実験の内容およびその結果、考察について述べる。最後に第 5 章では、以上の内容を踏まえた上で、本稿の結論と今後の課題について述べる。

第2章 背景

2.1 強化学習

強化学習 [29, 1] とは、マルコフ決定過程 (Markov Decision Process, MDP) [30] で規定される環境において、最適な方策を得るための機械学習の手法の一つである。強化学習で学習の対象となる環境は、実際には以下の4つの要素からなる有限マルコフ決定過程 (finite Markov decision process, finite MDP, MDP) [30] として定式化されることが多い。

- 状態の有限集合: $S = \{s_1, s_2, \dots, s_m\}$
- 行動の有限集合: $A = \{a_1, a_2, \dots, a_n\}$
- 遷移関数: 状態 $s \in S$ で行動 $a \in A$ を選択した時に状態 $s' \in S$ に遷移する確率 $T(s' | s, a) = \Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$
- 報酬関数: 状態 $s \in S$ で行動 $a \in A$ を選択し状態 $s' \in S$ に遷移した時にエージェントに与えられる報酬の期待値 $r(s, a, s') = \mathbb{E}[r_{t+1} | s_t = s, a_t = a, s_{t+1} = s']$

基本的な強化学習では、離散的な時間ステップ t における、環境と環境を学習するエージェントとの相互作用を繰り返すことで、エージェントの方策を学習していく。ある時刻 t において、エージェントはまず環境の状態 $s_t \in S$ を観測し、それに基づいて何らかの行動 $a_t \in A$ を選択する。その結果、環境は遷移関数 $T(s_{t+1} | s_t, a_t)$ に基づいて新しい状態 $s_{t+1} \in S$ に遷移すると同時に、エージェントはこの状態 s_{t+1} を観測して、環境から報酬 $r_{t+1} = r(s_t, a_t, s_{t+1})$ を受け取る。Fig. 2.1 はこの相互作用を説明した図である。

エージェントの各時刻における意思決定は、方策 (policy) $\pi_t(a | s)$ に基づいて行われる。 $\pi_t(a | s)$ とは、時刻 t における環境の観測 s_t が $s_t = s$ である時、その時に選択する行動 a_t が $a_t = a$ となる確率である。

強化学習の目標は、最終的に受け取る報酬の総量 (累計報酬 R_t) を最大化させるような方策 π を学習することにある。この累計報酬 R_t は、割引の概念を導入して定式化する場合が多い。このアプローチでは、エージェントは将来にわたり受け取る減衰報酬の合計が最大化されるような方策を学習する。ここで、時間ステップ t の後に受け取った報酬の系列を $r_{t+1}, r_{t+2}, r_{t+3}, \dots$ と表す時、割引された累計報酬 R_t は、以下のように定式化される。

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (2.1)$$

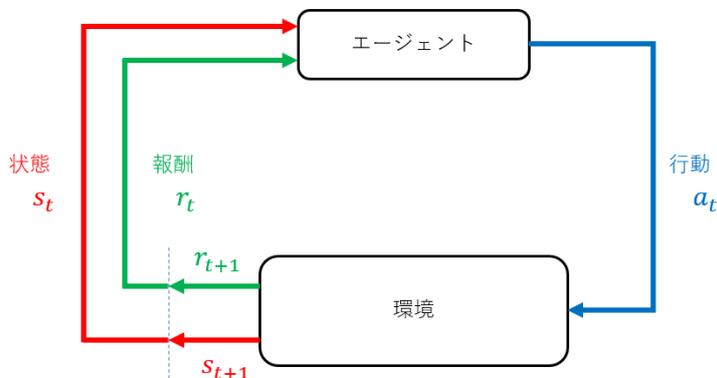


Fig. 2.1: 強化学習における、環境とエージェントの相互作用 [1]

ここで、 $\gamma \in [0, 1]$ は割引率と呼ばれるパラメータである。割引率は、将来の報酬が現在においてどれだけ価値があるのかを示している。

2.2 価値反復に基づくアルゴリズム

強化学習における方策獲得のアプローチの一つに、状態や行動の価値を推定することで方策を決定する価値反復法がある [31]。

ある方策 π のもとでの状態価値関数 $V^\pi(s)$ は、状態 s から方策 π に従って行動を選択し続けた場合の累計報酬の期待値、すなはち

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi [R_t \mid s_t = s] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \end{aligned} \quad (2.2)$$

として表される。

また、ある方策 π のもとでの行動価値関数 $Q^\pi(s, a)$ は、状態 s の下で行動 a を選択肢、その後は方策 π に従って行動を選択し続けた場合の累計報酬の期待値、すなはち

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi [R_t \mid s_t = s, a_t = a] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right] \end{aligned} \quad (2.3)$$

として表される。また、状態価値関数と行動価値関数の関係から、

$$V^\pi(s) = \sum_{a \in A} \pi(a \mid s) Q^\pi(s, a) \quad (2.4)$$

が成り立つ。

次に、式 2.2, 2.3 からベルマン方程式を導出する。まず、式 2.2 について、 $V^\pi(s)$ を分割して考えれば、

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi [R_t \mid s_t = s] \\
&= \mathbb{E}_\pi [r_{t+1} \mid s_t = s] + \mathbb{E}_\pi \left[\sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right] \\
&= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} T(s' \mid s, a) r(s, a, s') \\
&\quad + \gamma \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} T(s' \mid s, a) \mathbb{E}_\pi \left[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k+1} \mid s_{t+1} = s' \right]
\end{aligned} \tag{2.5}$$

となる。第 2 項の $\mathbb{E}_\pi \left[\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k+1} \mid s_{t+1} = s' \right]$ の部分は $V^\pi(s')$ とみなせるので、まとめると、

$$V^\pi(s) = \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} T(s' \mid s, a) (r(s, a, s') + \gamma V^\pi(s')) \tag{2.6}$$

を得る。これが状態価値関数に関するベルマン方程式である。行動価値関数の式 2.3 についても同様の変形を行うことで、

$$Q^\pi(s, a) = \sum_{s' \in S} T(s' \mid s, a) \left(r(s, a, s') + \sum_{a' \in A} \gamma \pi(a' \mid s') Q^\pi(s', a') \right) \tag{2.7}$$

を得る。これが行動価値関数に関するベルマン方程式である。

これらの価値関数は、強化学習をするエージェントにとって未知な関数である。そこで、エージェントの試行錯誤による経験で得られたデータを用いて、逐次的な更新によって価値関数を推定するのが価値反復法の基本的な考え方である。

価値反復法の代表的な手法の一つとして Sarsa [1] が知られている。Sarsa では、任意に初期化された行動価値関数 $Q(s, a)$ に対して、エージェントの経験によって得られたデータの組 $(s_t, a_t, s_{t+1}, r_{t+1})$ を用いて、ベルマン方程式を満たすように $Q(s, a)$ を

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.8}$$

によって更新する。 α は学習率である。環境が未知な場合（遷移関数が未知）でも価値関数を学習することができるが、得られる価値関数はエージェントが採用している方策に対応したものとなる。そのため、Sarsa は方策オン型（On-policy）の手法に分類される。

2.2.1 Q 学習

Q 学習 [32, 1] とは、強化学習の代表的なアルゴリズムの一つで、最適行動価値関数を推定するための価値反復法の手法である。

前項で説明したように、環境の状態が s の場合に、エージェントが行動 a を選択することの価値は、方策 π の下での行動価値関数 $Q^\pi(s, a)$ で表すことができた。ここで、得られる累積報酬を最大化させるような最適な方策 π^* を考えた時、方策 π^* の下での行動価値関数を $Q^*(s, a)$ と表記し、これを最適行動価値関数または Q 値と呼ぶ。

$$Q^*(s, a) = \mathbb{E}_{\pi^*} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right] \quad (2.9)$$

また、 Q^* はベルマン最適方程式と呼ばれる以下の関係式を満たす。これは行動価値関数に関するベルマン方程式 2.7 において、常に価値が最大となる行動を選択する方策を用いた場合を考えたものであり、ベルマン最適方程式を満たす価値関数はただ一つに定まることが証明されている。

$$Q^*(s, a) = \sum_{s' \in S} T(s' | s, a) \left(r(s, a, s') + \gamma \max_{a' \in A} Q^*(s', a') \right) \quad (2.10)$$

Q 学習では、このベルマン最適方程式を用いて、エージェントにとって未知である最適行動価値関数を学習していく。具体的な学習の流れは次のようになる。まず環境が取りうる全ての状態 s と選択できる全ての行動 a の組に対しての暫定的な行動価値 $Q(s, a)$ を保持することのできる Q-table を用意し、任意な値で初期化を行う。時間ステップ t において、環境 s_t 下のエージェントが行動 a_t を選択し、新しい状態 s_{t+1} に遷移して報酬 r_{t+1} を得た時、ここで得られた経験 $(s_t, a_t, s_{t+1}, r_{t+1})$ を用いて、 $Q(s, a)$ がベルマン最適方程式を満たすように次のような更新を行う。

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right) \quad (2.11)$$

α_t は学習率である。

Q 学習による Q 値の推定は、全ての状態行動対 (s, a) についてのエージェントの経験が十分に得られており、かつ学習率が一定の条件を満たしている場合に、全ての $Q(s, a)$ が最適な Q 値に収束することが証明されている。Sarsa の場合とは異なり、Q 学習ではエージェントが採用している方策に関わらず Q 値が同じ値に収束するため、Q 学習は方策オフ型 (Off-policy) の手法に分類される。

Q 学習では、理論的には行動をランダムに選択するようなエージェントの経験で得られるデータを用いる場合においても学習を行えるが、非常に非効率なものになることが多い。強化学習でよく用いられるゲームを学習する場合、例えばある状態からゲームを開始し、達成したい何らかの状態を目指してエージェントがゲームを学習することを考えると、行動をランダムに選択するエージェントでは、ゲーム序盤の経験データを得ることはできても、ゲームの終盤に近づくほどエージェントの到達確率が低くなり、経験データを得ることが非常に難しくなる。これはゲームの難易度が高いほどより顕著になり、ゲームの中盤以降の経験データがほとんど得られない状況になることも珍しくない。つまり、効率的な学習を行うためには、効率的な探索が必要となる。

Q 学習において効率的な探索を行うため、学習中の Q 値を行動選択に用いることで、ある程度良い方策に従った探索を可能とする手法がある。その一つに ϵ -greedy 法があり、確率 ϵ でランダムな行動を選択し、 $1 - \epsilon$ で Q 値が最大となる行動 ($\arg \max_a Q(s, a)$) を選択するような手法である。学習の途中で常に Q 値が最大となる行動を greedy に取り続けると、学習が局所解に陥る可能性があるため、一定の確率でランダムな行動を選択することでこれを回避している。学習が進むにつれて Q 値が信頼できる最適な値に近づいていくため、学習中に ϵ の値を徐々に小さくすることで、greedy な行動をより取りやすくするような工夫がなされることが多い。うまく学習できれば、単に Q 値が最大の行動を取り続けるのが最適な方策となる。

Q 学習のアルゴリズムを Algorithm 1 に示した。

Algorithm 1 Reinforcement Learning with Q-learning [32]

```

Initialize Q-Table  $Q(s, a) = 0$  for all pairs  $(s, a)$ 
for each episode do
  Reset environment, and observe initial state  $s$ 
  for each step of episode, state  $s$  is not terminal do
     $a \leftarrow$  action derived from  $Q(s)$  (e.g.  $\epsilon$ -greedy)
    Execute action  $a$ , and observe next state  $s'$  and reward  $r$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$ 
     $s \leftarrow s'$ 
  end for
end for
  
```

2.2.2 Deep Q-Network (DQN)

前項では、価値反復法の強化学習の代表的な手法として Sarsa や Q 学習を紹介した。これらの手法の共通点として、行動価値関数 $Q(s, a)$ を推定するために、MDP で規定されるすべての状態と行動の組 (s, a) に対しての暫定的な行動価値を、例えば q-table のような形で保持しておく必要があった。しかし、ビデオゲームのような状態数の非常に大きい複雑な環境を扱う強化学習は、メモリ確保の問題や学習に必要となる計算量などの観点から、これらの手法においては現実的ではないことが分かる。

Deep Q-Network (DQN) [7, 8] は、行動価値関数 $Q(s, a)$ を深層ニューラルネットワークを用いて推定し、Q 学習を行う手法である。すべての状態と行動の組 (s, a) に対応する Q 値 $Q^*(s, a)$ を求めるのではなく、入力された状態 s に対する各行動 (a_1, a_2, \dots, a_n) の価値 $Q^*(s, a_1), Q^*(s, a_2), \dots, Q^*(s, a_n)$ を予測するニューラルネットワークを学習する。つまり、ニューラルネットワーク自体が $Q(s, a)$ を表現する一つの関数になっている。

ニューラルネットワークのモデル部分には画像認識の分野で広く利用されていた Convolutional Neural Network (CNN) [33] の技術なども取り込まれている。これらの工夫により、例えばビデオ

ゲームの強化学習において、状態としてゲームの画像をそのまま入力しても効率的に学習が行えるようになった。実際に、DQN では Atari 2600 のゲームシリーズにおいて、いくつかのゲームでは人間よりも高いスコアを記録するほどの成果を上げた。DQN のような、深層学習と強化学習を組み合わせた手法を深層強化学習と呼ぶ。

Algorithm 2 Reinforcement Learning with Deep Q-Network (DQN) [8]

Initialize replay memory D with capacity M

Initialize action-value function Q with random network parameters θ

Initialize target action-value function \tilde{Q} with random network parameters θ^-

for each episode **do**

Reset environment, and observe initial state s_0

for each step t of episode, state s_t is not terminal **do**

$$a_t \leftarrow \begin{cases} \text{random } a & \text{with probability } \epsilon \\ \arg \max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$$

Execute action a_t , and observe next state s_{t+1} and reward r_{t+1}

Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in D

Sample random minibatch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from D

$$y_j \leftarrow \begin{cases} r_{j+1} & \text{if } s_{j+1} \text{ is terminal} \\ r_{j+1} + \gamma \max_{a'} \tilde{Q}(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

Perform gradient decent step on $(y_j - Q(s_j, a_j; \theta))^2$ w.r.t. θ

Every C steps, reset $\tilde{Q} \leftarrow Q$, i.e. set $\theta^- \leftarrow \theta$

end for

end for

関数近似を用いた Q 学習を行う DQN のアルゴリズムでは、次の式で定義される損失関数を最小化させるような最適化を行う。

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right] \quad (2.12)$$

ニューラルネットワークを用いて Q 関数の関数近似を行うため、Q 関数は各時点のニューラルネットワークのパラメータ θ_i を用いて $Q(s, a; \theta_i)$ と表現され、Q 学習における更新式 2.11 と同様の式で損失関数を定義している。ただし、Q 学習の場合とはことなり、DQN では Q 関数を直接近似しているために、学習する Q 関数が最適行動価値関数に収束することが保証されていない。そのため DQN では、学習の安定させるためにいくつかの工夫がなされている。一つは経験再生 (Experience Replay) の利用である。エージェントの経験によって得られたデータ $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ を、すぐに学習に用いるのではなく、一旦リプレイメモリ D に蓄積し、学習時にはリプレイメモリからランダムにサンプリングした経験を用いたミニバッチ学習を行っている。この工夫により、サンプル系

列において時間的相関があると、ニューラルネットワークにおける確率的勾配法の学習が不安定になる問題が緩和されることが経験的に分かっている。もう一つは Target Network の利用である。Q 関数 $Q(s, a; \theta_i)$ の更新時に、学習中の Q 関数だけではなくパラメータが古いもので固定されているネットワーク $Q(s, a; \theta_i^-)$ も利用することで、現在推定している Q 関数と損失関数の間の相関によって学習が発散しやすくなる問題を緩和する狙いがある。DQN による学習アルゴリズムを Algorithm 2 に示した。

DQN 以降も、価値反復法に基づく様々な手法が DQN の派生として発表されている。代表的なアルゴリズムとしては、Target Network を利用することを提案した Double DQN (DDQN) [9]、行動価値関数を状態価値関数と後述するアドバンテージ関数に分けて学習する Dueling Network [10]、これらの手法に加えて優先度付き経験再生などの既存手法を組み合わせた Rainbow [11]、DQN を分散処理によって高速化した General Reinforcement Learning Architecture (Gorila) [12]、優先度付き経験再生を分散処理で高速化した Ape-X [13]、LSTM [34] と経験再生を組み合わせる分散学習を行う Recurrent Replay Distributed DQN (R2D2) [14] などがある。

2.3 方策勾配に基づくアルゴリズム

価値反復に基づく強化学習においては、方策は行動価値関数から導かれるものであり、行動価値関数を学習することによって、最適な方策を間接的に求めていた。これとは異なるアプローチとして、方策を行動価値関数とは別のパラメータで表現し、このパラメータを学習することで最適な方策を直接的に求めることも考えられる。確率の方策として、あるパラメータベクトル θ で決定される確率モデル $\pi_\theta(a | s)$ を考え、累計報酬を最大化させるようにパラメータ θ を最適化するような強化学習のアプローチを、ここでは方策勾配に基づくアルゴリズムと呼ぶ。

目的関数 $J(\theta)$ とする。状態 s_0 から始めて方策 π_θ に従って行動し続けた時に $s_t = s$ となる確率を $Pr(s_t = s | s_0, \pi_\theta)$ とした時、割引された定常確率 $d^\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s | s_0, \pi_\theta)$ を考えれば、現在のパラメータ θ に対しての割引きされた累計報酬の期待値として、目的関数 $J(\theta)$ を次のように記述できる。

$$J(\theta) = \sum_{s \in S} d^\pi(s) V^\pi(s) = \sum_{s \in S} d^\pi(s) \sum_{a \in A} \pi_\theta(a | s) Q^\pi(s, a) \quad (2.13)$$

方策勾配に基づくアルゴリズムでは、この目的関数 $J(\theta)$ の勾配 $\nabla_\theta J(\theta)$ を計算して、確率勾配法などによってパラメータ θ の更新を行う。ここで、 $\nabla_\theta J(\theta)$ に関して、以下の方策勾配定理 (Policy Gradient Theorem) [1] が成り立つ。

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi [Q^\pi(s, a) \nabla_\theta \ln \pi_\theta(a | s)] \quad (2.14)$$

Actor-Critic と呼ばれるような方策勾配を用いた強化学習のアルゴリズムでは、方策を表現する Actor と呼ばれるモデルと、方策を評価する行動価値関数のモデルとを同時に学習するような手法が用いられており、その学習の際には、方策勾配定理における行動価値関数 $Q^\pi(s, a)$ の代わりに、アドバンテージ関数 $A^\pi(s) = Q^\pi(s, a) - V^\pi(s)$ が用いられる。これは状態価値関数 $V^\pi(s)$ が選択した行動に依存しない値であり、勾配の期待値の微分計算に影響を与えないためである。

2.3.1 方策勾配定理 (Policy Gradient Theorem) の証明

前項で述べた方策勾配定理の式 2.14 を証明する [35]。まず状態価値関数の微分を考える。

$$\begin{aligned}
& \nabla_{\theta} V^{\pi}(s) \\
&= \nabla_{\theta} \left(\sum_{a \in A} \pi_{\theta}(a | s) Q^{\pi}(s, a) \right) \\
&= \sum_{a \in A} (\nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) + \pi_{\theta}(a | s) \nabla_{\theta} Q^{\pi}(s, a)) \\
&= \sum_{a \in A} \left(\nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) + \pi_{\theta}(a | s) \nabla_{\theta} \sum_{s' \in S} T(s' | s, a) (r(s, a, s') + \gamma V^{\pi}(s')) \right) \\
&= \sum_{a \in A} \left(\nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) + \pi_{\theta}(a | s) \gamma \sum_{s' \in S} T(s' | s, a) \nabla_{\theta} V^{\pi}(s') \right)
\end{aligned} \tag{2.15}$$

と変形できるが、ここで次の関係式

$$\nabla_{\theta} V^{\pi}(s) = \sum_{a \in A} \left(\nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) + \pi_{\theta}(a | s) \gamma \sum_{s' \in S} T(s' | s, a) \nabla_{\theta} V^{\pi}(s') \right) \tag{2.16}$$

に着目すれば、 $\nabla_{\theta} V^{\pi}(s)$ に関する再帰的な構造を見出すことができる。

ここで、状態 s から状態 x へと方策 π_{θ} のもとで k ステップで到達できる確率を $\rho^{\pi}(s \rightarrow x, k)$ と表現することになると、 $k = 0$ の時は

$$\rho^{\pi}(s \rightarrow s, k = 0) = 1 \tag{2.17}$$

が、 $k = 1$ の時は

$$\rho^{\pi}(s \rightarrow s', k = 1) = \sum_{a \in A} \pi_{\theta}(a | s) T(s' | s, a) \tag{2.18}$$

が成り立つことが自明に分かる。一般の k については、 ρ^{π} について成り立つ次の関係式

$$\rho^{\pi}(s \rightarrow x, k + 1) = \sum_{s' \in S} \rho^{\pi}(s \rightarrow s', k) \rho^{\pi}(s' \rightarrow x, 1) \tag{2.19}$$

を再帰的に適用することによって求められる。

簡単のため、

$$\phi(s) = \sum_{a \in A} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) \tag{2.20}$$

とすると、 $\nabla_{\theta} V^{\pi}(s)$ は

$$\begin{aligned}
& \nabla_{\theta} V^{\pi}(s) \\
&= \phi(s) + \gamma \sum_{a \in A} \pi_{\theta}(a | s) \sum_{s' \in S} T(s' | s, a) \nabla_{\theta} V^{\pi}(s') \\
&= \phi(s) + \gamma \sum_{s' \in S} \sum_{a \in A} \pi_{\theta}(a | s) T(s' | s, a) \nabla_{\theta} V^{\pi}(s') \\
&= \phi(s) + \gamma \sum_{s' \in S} \rho^{\pi}(s \rightarrow s', 1) \nabla_{\theta} V^{\pi}(s') \\
&= \phi(s) + \gamma \sum_{s' \in S} \rho^{\pi}(s \rightarrow s', 1) \left[\phi(s') + \gamma \sum_{s'' \in S} \rho^{\pi}(s' \rightarrow s'', 1) \nabla_{\theta} V^{\pi}(s'') \right] \\
&= \phi(s) + \gamma \sum_{s' \in S} \rho^{\pi}(s \rightarrow s', 1) \phi(s') + \gamma^2 \sum_{s'' \in S} \rho^{\pi}(s \rightarrow s'', 2) \nabla_{\theta} V^{\pi}(s'') \\
&= \dots \\
&= \sum_{x \in S} \sum_{k=0}^{\infty} \gamma^k \rho^{\pi}(s \rightarrow x, k) \phi(x)
\end{aligned} \tag{2.21}$$

と変形できるから、累積報酬 $J(\theta)$ を考える際の初期状態を s_0 とすれば、(以下の変形中、簡単のため $\eta(s) = \sum_{k=0}^{\infty} \gamma^k \rho^{\pi}(s_0 \rightarrow s, k)$ と置いた。)

$$\begin{aligned}
\nabla_{\theta} J(\theta) &= \nabla_{\theta} V^{\pi}(s_0) \\
&= \sum_{s \in S} \sum_{k=0}^{\infty} \gamma^k \rho^{\pi}(s_0 \rightarrow s, k) \phi(s) \\
&= \sum_{s \in S} \eta(s) \phi(s) \\
&= \left(\sum_{s \in S} \eta(s) \right) \sum_{s \in S} \frac{\eta(s)}{\sum_{s \in S} \eta(s)} \phi(s)
\end{aligned} \tag{2.22}$$

となり、 $\sum_{s \in S} \eta(s)$ は定数であるから、

$$\nabla_{\theta} J(\theta) \propto \sum_{s \in S} \frac{\eta(s)}{\sum_{s \in S} \eta(s)} \phi(s) \tag{2.23}$$

であり、 $d^{\pi}(s)$ と $\eta(s)$ の定義から、

$$d^{\pi}(s) = \frac{\eta(s)}{\sum_{s \in S} \eta(s)} \tag{2.24}$$

が成り立つから、

$$\nabla_{\theta} J(\theta) \propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) \quad (2.25)$$

となる。よって、

$$\begin{aligned} \nabla_{\theta} J(\theta) &\propto \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) \\ &= \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(a | s) Q^{\pi}(s, a) \frac{\nabla_{\theta} \pi_{\theta}(a | s)}{\pi_{\theta}(a | s)} \\ &= \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} Q^{\pi}(s, a) \nabla_{\theta} \ln \pi_{\theta}(a | s) \\ &= \mathbb{E}_{s \sim d^{\pi}, a \sim \pi_{\theta}} [Q^{\pi}(s, a) \nabla_{\theta} \ln \pi_{\theta}(a | s)] \end{aligned} \quad (2.26)$$

と求まり、証明できた。

前述のように、方策勾配定理の式 2.14 は、簡単のため $\mathbb{E}_{\pi} = \mathbb{E}_{s \sim d^{\pi}, a \sim \pi_{\theta}}$ と表記され、また、比例式ではなく等式の形で書かれることが多い。

2.3.2 Proximal Policy Optimization (PPO)

方策勾配に基づいた深層強化学習の手法の一つに、Proximal Policy Optimization (PPO) [18] がある。

PPO では、行動を決める Actor を直接改善しながら、その方策を評価する Critic も同時に学習させる、Actor-Critic のアプローチが用いられている。具体的には、エージェントが観測した状態 s を入力とするニューラルネットワーク（パラメータ θ ）の出力として、Actor としての方策 π_{θ} と、Critic としての状態価値関数 $V^{\pi_{\theta}}(s)$ を同時に計算し、これを学習するものである。

方策勾配に基づいた Actor-Critic 系の手法には、非同期の分散学習を行う Asynchronous Advantage Actor-Critic (A3C) [15] という手法や、A3C の学習を同期的に行えるように改良した Advantage Actor-Critic (A2C) [16] という手法などがあった。これらの手法に共通して、マルチエージェントによる分散学習をする方式が採用されており、より効率的に環境を学習できることが期待される。しかしながら、学習中に行われるニューラルネットワークのパラメータ更新が起因して、1 度でも方策関数が大きく劣化してしまうと、その後は報酬が得られにくくなり方策関数の改善も困難になるなど、学習が不安定である問題が指摘されていた [17]。

PPO では、学習の安定性を向上させるために、方策関数が大きく更新されるのを防ぐようなアルゴリズムが提案されている。まず、パラメータ更新前後でのネットワークの方策関数の出力の比 $r(\theta)$ は

$$r(\theta) = \frac{\pi_{\theta}(a | s)}{\pi_{\theta_{\text{old}}}(a | s)} \quad (2.27)$$

と表せる。その時点でのアドバンテージ関数の推定値を \hat{A}_t とすると、PPO における方策の損失関数 $L^{\text{CLIP}}(\theta)$ は次の式で表される。

$$L_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r(\theta) \hat{A}_t, \text{clip} \left(r(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right] \quad (2.28)$$

更新前後の方策関数の比 $r(\theta)$ を、clip 関数を用いて $[1 - \epsilon, 1 + \epsilon]$ の範囲に抑えた値で方策勾配の計算を行うことで、方策関数の大きな更新を起こりにくくしている。具体的には、 $\hat{A}_t > 0$ の時は $r(\theta)$ が $1 + \epsilon$ を超えないように、 $\hat{A}_t < 0$ の時は $r(\theta)$ が $1 - \epsilon$ を下回らないように計算が行われる。なお、式中で用いられている clip 関数の定義は次の通りである。clip (x, a, b) ($a < b$) という関数は、 x の値を $[a, b]$ の範囲に収めるという操作を意味している。

$$\text{clip}(x, a, b) = \begin{cases} a & (x < a) \\ b & (x > b) \\ x & (\text{otherwise}) \end{cases} \quad (2.29)$$

PPO では、この方策の損失関数 $L_t^{\text{CLIP}}(\theta)$ に、アドバンテージ関数を推定するための状態価値の損失関数 $L_t^{\text{VF}}(\theta) = (V_\theta(s_t) - V_t^{\text{targ}})^2$ (ただし、 $V_t^{\text{targ}} = \hat{A}_t + V_{\theta_{\text{old}}}(s_t)$) と、学習を安定化させるための [36] エントロピー項 $L_t^{\text{S}}(\theta) = S[\pi_\theta](s_t)$ を加えたものを損失関数として、これを最大化させる学習を行う。

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right] \quad (2.30)$$

c_1, c_2 は定数である。

また、アドバンテージ関数を近似的に推定するために Generalized Advantage Estimator (GAE) [37] という手法が PPO では用いられている。通常のアドバンテージ関数は、 T ステップの経験の軌跡から次のように計算される。

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \dots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \quad (2.31)$$

この計算を一般化したものが GAE である。GAE の具体的な計算方法は次の式のようになる。

$$\hat{A}_t = \delta_t + (\gamma\lambda) \delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1} \delta_{T-1}, \quad (2.32)$$

$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

$\lambda = 1$ の時、GAE で得られるアドバンテージ関数は、 $\lambda = 0$ の時アドバンテージを考慮しない計算になり、 $\lambda = 1$ の時通常のアドバンテージ関数と同じものになる。つまり、 λ を変更することでアドバンテージの考慮具合を調整することができる。

マルチエージェントによる分散学習をする PPO のアルゴリズムを Algorithm 3 に示した。

Algorithm 3 Reinforcement Learning with PPO [18]

$N \leftarrow$ number of actors
 $K \leftarrow$ number of optimization epochs
 $D_1, \dots, D_N \leftarrow$ transition memory for each actors
Initialize policy parameters with θ , and value function parameters with ϕ
for each iteration $i = 1, 2, 3, \dots$ **do**
 for each actor $j = 1, 2, \dots, N$ **do**
 Reset the environment E_j , and observe initial state s_0
 for each step t of episode, state s_t is not terminal **do**
 Compute policy $\pi_t = \pi_\theta(s_t)$ and value $V_t = V_\phi(s_t)$
 Sample action $a_t \sim \pi_t(a_t | s_t)$
 Execute action a_t , and observe next state s_{t+1} and reward r_{t+1}
 Store transition $(\pi_t, V_t, a_t, s_t, r_{t+1})$ in D_j
 end for
 Compute advantage estimates \hat{A}_t from the set of transitions D_j of length T based on the current value function V_ϕ with GAE:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$
where $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$
 Store advantage estimates \hat{A}_t in D_j
 end for
Combine D_1, \dots, D_N as an optimization batch B
for each optimization epoch $i = 1, 2, \dots, K$ **do**
 Sample a minibatch of transitions $(\pi_t, V_t, a_t, s_t, r_{t+1}, \hat{A}_t)$ from B
 Update the policy and value function parameters θ, ϕ by minimizing the PPO objective $L_t^{\text{CLIP+VF+S}}(\theta, \phi)$ on the minibatch:

$$\begin{aligned} L_t^{\text{CLIP}}(\theta) &= \min \left(\frac{\pi_\theta(a|s)}{\pi_t(a|s)} \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_t(a|s)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \\ L_t^{\text{VF}}(\phi) &= (V_\theta(s_t) - V_t^{\text{arg}})^2 = (V_\theta(s_t) - (\hat{A}_t + V_t))^2 \\ L_t^{\text{CLIP+VF+S}}(\theta, \phi) &= L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\phi) + c_2 S[\pi_\theta](s_t) \end{aligned}$$
typically via some gradient decent algorithm like Adam [38]
 end for
Clear transition memory D_1, \dots, D_N and optimization batch B
end for

2.4 強化学習と報酬設計

強化学習における報酬とは、環境を学習するエージェントの目的や目標を信号として定式化したものである。強化学習は累計報酬を最大化させるような方策を学習するものであるから、報酬関数を設計する際には、その報酬によって学習される方策が設計者が意図していた目的や目標を達成できるものになるように考える必要がある。

例えば、迷路を脱出するようなゲームの学習を考えれば、迷路の出口に到達するまでの各ステップで -1 の報酬（ペナルティ）を与えれば、エージェントはなるべく早く迷路を脱出できるように学習することになる。または、迷路の出口に到達した時のみに $+1$ の報酬を与えるような設計でも、時間割引された累計報酬を考えていけば、本質的には同様の学習を行うことができる。チェスやオセロのような二人零和なゲームの学習ならば、勝ちを $+1$ 、負けを -1 、引き分けを 0 とする単純な報酬設計が考えられるだろう。DQN を始めとして、ゲーム AI の研究分野でよく用いられている Atari 2600 のゲームシリーズを使った学習では、ゲームの機能としてすでに搭載されているスコアの増減をそのまま、あるいはクリッピングして用いる報酬設計での学習が行われていることが多く、成功例も多い [8]。

このように、報酬関数はエージェントの目的や目標に応じて設計する必要があるため、報酬関数は多くの場合、人間が直接的に設計することになる。しかしながら、人間が報酬関数を設計することは、難易度の高いゲームや現実の問題を扱う際など、学習の対象の環境が非常に複雑な場合に、設計コストや学習効率の面でしばしば問題になる。

報酬関数の設計に関わる課題の一つに、効率的な学習を可能にするような密（報酬が得られる機会が多い）で well-shaped（学習すべきところにうまく報酬が割り当てられている）な報酬関数の設計が難しいということが挙げられる。簡単な報酬設計では、報酬が得られる機会が少ないような疎な報酬関数になりやすいが、これは学習効率の面で問題になる。例えば、上記の迷路の例において、迷路の出口に到達した時のみに $+1$ の報酬を与えるような設計を考えると、エージェントは何らかの報酬を見つけるまではランダムに行動するしかない（報酬が得られなければ方策を更新することができない）ため、迷路のサイズが小さく形状も単純で、ランダムなエージェントでもゴールしやすいような場合には問題にならないが、迷路のサイズが大きく形状も複雑であった場合には、ランダムなエージェントではほとんどゴールすることができず、結果として学習も困難になる。DQN が Atari の一部のゲームにおいて高い成果を達成したことに関して、報酬として用いていたゲームのスコアが密で well-shaped な設計であったことも大きな理由として考えられる [20]。しかし、このような良い報酬関数の設計を行うことは、環境に関しての深い理解が求められるだけでなく、その実装や検証・調整といった試行錯誤などのエンジニアリングのコストも必要となり、非常に困難なタスクであると言える。

次に、報酬関数の設計によっては、設計者の意図しない方向にエージェントが学習してしまう問題も挙げられる。例えば、チェスをプレイするエージェントに、相手の駒を取った時に報酬を与えるような設計にすると、意図していた「ゲームに勝利する方向」にはなく、予測していなかった「ゲームに敗北してでも相手の駒を獲得する方向」に学習が進んでしまう可能性がある。この問題は、基本的には報酬の設計ミスが原因となるものだが、対象となる環境が複雑になるほどこのようなミスは

起こりやすくなると考えられる。

最後に、ある環境について考えられた報酬設計は、そのまま別の環境に適用することが一般的に難しく、汎用性が低いという問題がある。

このような人間による報酬関数の設計に関わる問題を解決するためのアプローチがいくつか存在する。逆強化学習 [39] は、報酬関数から方策を学習する強化学習とは異なり、エキスパートの方策や行動軌跡から報酬関数自体を学習するような手法である。報酬関数の設計こそ難しいが、タスク自体は人間でも解けるような問題を強化学習で扱う際に、逆強化学習は非常に有用な手法であると言える。エキスパートによる行動軌跡という教師データこそ必要となるが、簡単な環境なら少量のデータでも学習が難しくないことが知られている。

逆強化学習の他にも、エージェントが現在の状況に応じて自発的に報酬を生成するような手法によるアプローチがある。これは、エージェントが環境から外発的に与えられる報酬（外部報酬）だけではなく、エージェント自らが、環境の状態やいままでの軌跡・経験などから内発的に報酬（内部報酬）を生成し、この2種類の報酬を学習に利用するというものである。これにより、環境の持つ報酬関数、つまり外部報酬が疎な設計であっても、密な内部報酬を生成するようなアルゴリズムを採用することで、密な内部報酬を用いることによる疎な外部報酬間の橋渡しを行い、学習の効率化を図る狙いがある。本稿では、この報酬生成のアプローチに注目した。

2.5 ローグライクゲーム

本稿では、強化学習の対象となるゲーム環境として、コンピュータ RPG の一種であるローグライクゲームを採用した。ローグライクゲーム (Rogue-like games) とは、1980 年の Rogue から派生したゲームの総称であり、日本では不思議のダンジョンシリーズなどがよく知られている。ローグライクゲームには様々な種類の作品があるが、どの作品も基本的には Rogue に準じた作りになっているため、その中に共通した特徴を見出すことができる。主な特徴を具体的に挙げると次のようになる。

- プレイヤーは、自動的に生成される迷路のようなダンジョンを探索しながら、ゲームクリアを目指す。
- ダンジョンは一定数の階層から成り立ち、それぞれの階層にゴールが設定されている。ゲームクリアは、最終層のゴールに到達することを指すことが多い。
- ダンジョンはある程度の広さを持った部屋と細い通路から構成されており、迷路のような形状となっている。
- ゲーム開始時のプレイヤーはダンジョンの全容を観測することができず、探索によって観測できる範囲を広げていく必要がある。
- ダンジョンには、攻略に役立つアイテムや、攻略の妨げになるモンスターや罠などが配置されることがある。

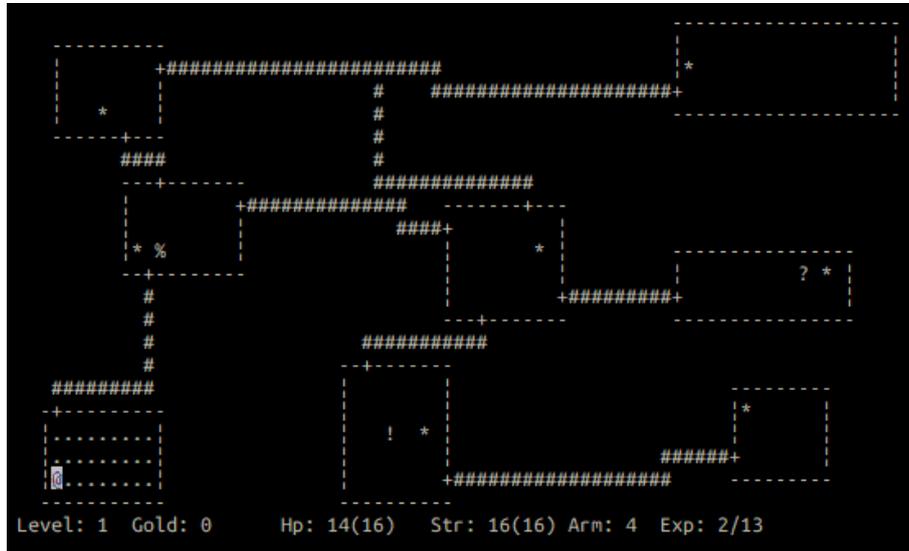


Fig. 2.2: Rogue のスクリーンショット

- ダンジョンの形状やプレイヤーの初期位置、ゴールの位置、アイテム・モンスター・罠などの配置は毎回ランダムに決定される。
- モンスターを倒すことでプレイヤーは経験値を得ることができる。
- リアルタイムの判断や操作が要求されない、ターン制のゲームである。

Fig. 2.2 は Rogue のスクリーンショットで、Level1 ステージを探索した後の状態である。長方形の部屋とそれらを繋ぐ通路が迷路のように接続されていることや、HP や経験値、アイテムなどの要素の存在が確認できる。

ローグライクゲームには非常に複雑な環境を持ち、人間がプレイしても十分な手応えを感じられるほどに難易度が高いものが多い。そのため、当然ではあるが、単純な強化学習でローグライクゲームの環境を扱うことは難しい。ローグライクゲームを対象とした強化学習に関する先行研究 [23, 24, 25, 26, 27, 28] では、強化学習を難しくする要因や、AI が克服すべき課題について、様々なことが言及されている。それらを以下にまとめた。

- **POMDP nature:** ローグライクゲームは、厳密には部分観測マルコフ決定過程 (Partially Observable Markov Decision Process, POMDP) [40] のモデルを持つゲームである。何故なら、ローグライクゲームではダンジョンの様子を部分的にしか観測できず、探索によって観測できる範囲を広げていく必要があるからである。POMDP の環境では、エージェントの観測では「同じに見える状態」であったも、実際は異なる状態であるということが起こり得るものであり、これはローグライクゲームにおいては特に顕著である。一般的に、POMDP の環境は MDP の環境と比べて学習が難しく、最適な方策を求めるにあたっては、状態と観測を区別して考える必要や、過去の経験の記憶、事前知識などが必要となる場合がある。 [28]

- **Random Environment:** ローグライクゲームでは、ダンジョンの形状や、プレイヤーやゴールの出現位置などを始めとして、様々な要素がランダムに決定される、非常にランダム性の高いゲームである。そのため、Atari などのゲームにおける学習のように、毎回同じステージから始めて学習するなどといったことができない。故に、ゲームの要素を本質的に捉えて、どのようなステージが与えられてもうまく攻略できるような学習が求められる。 [23, 28]
- **Memory and Attention:** ローグライクゲームを攻略するのに必要とされる能力として、より難しいステージに向けての稼ぎやアイテムの使用タイミングなどに関する長期的な戦略を立てる能力や、モンスターなどに対処するための短期的な戦術をその場で考える能力、どの程度まで危険を許して稼ぎを行うべきなのかを判断する能力など様々なものが挙げられる。 [23] これらの能力を AI として実現するためには、長期的な記憶を扱う能力 (Memory) や、ゲームの特徴的な部分に着目して考えられる能力 (Attention) などが前提として求められるだろう。 [28]
- **Sparse rewards:** ローグライクゲームは、ゴールに到達することが目的であるという点で、ゴールに対してのみ報酬を与えるような設計だと非常に疎な報酬設計になってしまう。 [28] また、ローグライクゲームの様々な要素 (アイテム・モンスター・罠など) は、ゲームの複雑さが故に、それらがどれほどゲームクリアの目標に貢献できるかを推定することが非常に難しく、またそれらの価値がその時の状況によっても異なってくるため、報酬設計自体が非常に難しいタスクとなってしまっているという問題がある。
- **Action space:** ローグライクゲームにおいて選択できる行動は、移動のための行動が主になるが、要所で選択すべき行動としてアイテムの使用や敵への攻撃などがあり、それにより行動の選択肢 (action space) が大きくなってしまふ。しかしながら、条件を満たしてなければ取れない行動の存在や、それぞれの行動を選択する確率に大きな差が存在すること (基本的には移動の行動が主であり、その他の行動が選択されるのは本来稀である。) などが起因し、学習が難しくなってしまう。 [23]

また、ローグライクゲームの研究を難しくしている原因としては、他にも、学習や評価に用いるためのローグライクなゲーム環境を用意することが難しいことや、その結果として研究者によってゲーム環境や評価方法が異なり提案されている手法の比較が難しくなっていることなどが挙げられる。これは、既存のゲーム環境の扱いにくさやカスタマイズ性の低さ、自前の環境を実装する難しさなども一因として考えられる。このような問題から、ローグライクゲームの研究分野では、学習用のゲーム環境を提案している研究も多い [23, 24, 25, 26]。高橋ら [23] は、AI 研究の分野でローグライクゲームをより扱いやすくするための統一されたゲーム環境プラットフォームの必要性を述べ、そのゲーム環境が持つべきローグライクな要素やルールなどを提案した。金川ら [24] は、ローグライクゲームの元となった Rogue のゲーム環境を、OpenAI-Gym [41] で提案された AI 研究用のゲーム環境上で扱えるようにしたものとして Rogue-Gym を提案している。Asperti ら [25, 26] も、Rogue のゲーム環境を元にして作られた Rogueinabox というゲーム環境を提案しており、カスタマイズ性の高さから使いやすいついものとなっている。

第3章 好奇心ベースの報酬生成手法

強化学習において、環境からエージェントが受け取る外部報酬が疎である場合、つまりエージェントが0ではない報酬を受け取る機会が少ない場合に、エージェントは効率的な探索を行うことが難しく、結果として学習もうまく進まないことがあること、そして密な外部報酬の設計が難しいことを前項では述べた。このような問題に対処するアプローチの一つに、エージェントが観測する環境の新規性に価値を見出し、それを内部報酬として利用することで、効率的な探索を行おうとする手法があった。

このアプローチに関して、近年では好奇心ベースの報酬生成手法 [21, 20, 22] が注目を集めている。共通点として、エージェントによる環境の予測と、実際の環境との違い、つまり予測誤差を「好奇心」と見なし、それを内部報酬として利用した強化学習を行っている。エージェントによる環境の予測誤差が、エージェントによる探索が十分に行われていない部分で大きくなる（探索が行われていなければ、環境のその部分についての学習も進まないため）ことを利用し、予測誤差をそのまま内部報酬（好奇心）として用いることで、探索が十分に行われていない部分をより重点的に探索できるようにする狙いがある。

ここでは、好奇心ベースの報酬生成手法として、Intrinsic Curiosity Module (ICM) [21] と Random Network Distillation (RND) [22] の代表的な2つの手法に注目し、具体的なアルゴリズムや研究背景を紹介する。

3.1 Intrinsic Curiosity Module (ICM)

Intrinsic Curiosity Module (ICM) [21] とは、環境の予測誤差によって定式化される好奇心の内部報酬を自動的に生成する手法である。ICMは2つのニューラルネットワークのモデルによって構成されている。

一つは、逆モデル (inverse dynamics model) である。まず、エージェントは環境から連続した状態 s_t と s_{t+1} を観測する。次に、これらの2つの状態の入力をニューラルネットワークなどを利用して、何らかの特徴表現 $\phi(s_t), \phi(s_{t+1})$ へ変換する。状態 s_t から s_{t+1} へは、エージェントが選択した行動 a_t によって遷移するが、この行動 a_t を2つの状態 $\phi(s_t), \phi(s_{t+1})$ から予想するのがこの逆モデルの役割である。逆モデルの出力は次のようにかける。

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I) \quad (3.1)$$

ここで、 g は逆モデルの持つニューラルネットワークで表現される関数を表しており、 θ_I はそのネットワークのパラメータである。 \hat{a}_t が、逆モデルが出力する行動の予測の確率布である。逆モデルは、

次のような最適化を以って学習を行う。

$$\min_{\theta_I} L_I(\hat{a}_t, a_t) \quad (3.2)$$

つまり、実際に取った行動 a_t と予想した行動の確率分布 \hat{a}_t との予測誤差 L_I を損失関数とし、それを最小化する方向に学習を行う。

もう一つが、順モデル (forward dynamics model) である。エージェントが時間ステップ t において観測した状態 $\phi(s_t)$ と、その時点で選択した行動 a_t から、遷移した環境の次状態 $\phi(s_{t+1})$ を予測するのが順モデルの役割である。このように、順モデルで行われる予測には、本来の 2 つの状態の入力 s_t, s_{t+1} が逆モデルの中で特徴表現に変換されたものを用いて行われる。順モデルの出力は次のようにかける。

$$\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F) \quad (3.3)$$

逆モデルと同様に、 f は順モデルの持つニューラルネットワークで表現される関数を表しており θ_F はそのネットワークのパラメータである。 $\hat{\phi}(s_{t+1})$ が、順モデルが出力する次状態の特徴表現の予測である。順モデルでは、次のような最適化を以って学習を行う。

$$\min_{\theta_F} L_F(\phi(s_t), \hat{\phi}(s_{t+1})) = \min_{\theta_F} \frac{1}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|^2 \quad (3.4)$$

つまり、実際の次状態 $\phi(s_{t+1})$ と予測した次状態 $\hat{\phi}(s_{t+1})$ との二乗誤差で計算される予測誤差 L_F を損失関数とし、それを最小化する方向に学習を行う。の時、この順モデルは内部報酬として、次式で計算される r_t^i を出力する。

$$r_{t+1}^i = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|^2 \quad (3.5)$$

ただし、 $\eta > 0$ である。

ICM は、これら 2 つのモデルを同時に学習させる。次式のように、2 つのモデルの損失関数である L_I と L_F のそれぞれに重みを付けた和を目的関数として定義し、この重み付け和を最小化させるように学習を行う。

$$\min_{\theta_I, \theta_F} [(1 - \beta) L_I + \beta L_F] \quad (3.6)$$

Fig. 3.1 に ICM のアーキテクチャの概要を示す。内部報酬を生成する上で最も重要なのは順モデルの学習である。しかし、順モデルが扱うのは次状態を予測するような計算コストの高いタスクなので、順モデル単体で学習させるのが難しい場合が多い。そこで、逆モデルの学習を順モデルの学習に組み合わせることで、選択された行動を予測するという比較的容易なタスクの学習を通して、環境の状態の特徴表現を獲得する部分の学習が効率的に行われるようになり、これによって順モデルの学習効率を向上をさせる狙いがある。

ICM の論文 [21] では、VizDoom [42] と Super Mario Bros. の 2 種類のゲーム環境を用いた実験で、ICM の生成する内部報酬によって学習がより効率的に進んだことが報告されている。翌年、この ICM をベースにして、好奇心ベースの強化学習についてより詳細な検証を行った研究 [20] が、Burda らにより発表された。Burda らは、48 種類の Atari 2600 のゲーム、Super Mario Bros.、Juggling (Roboschool)、Unity mazes、Two-player Pong などの非常に多くのゲーム環境で、外部報酬を全く

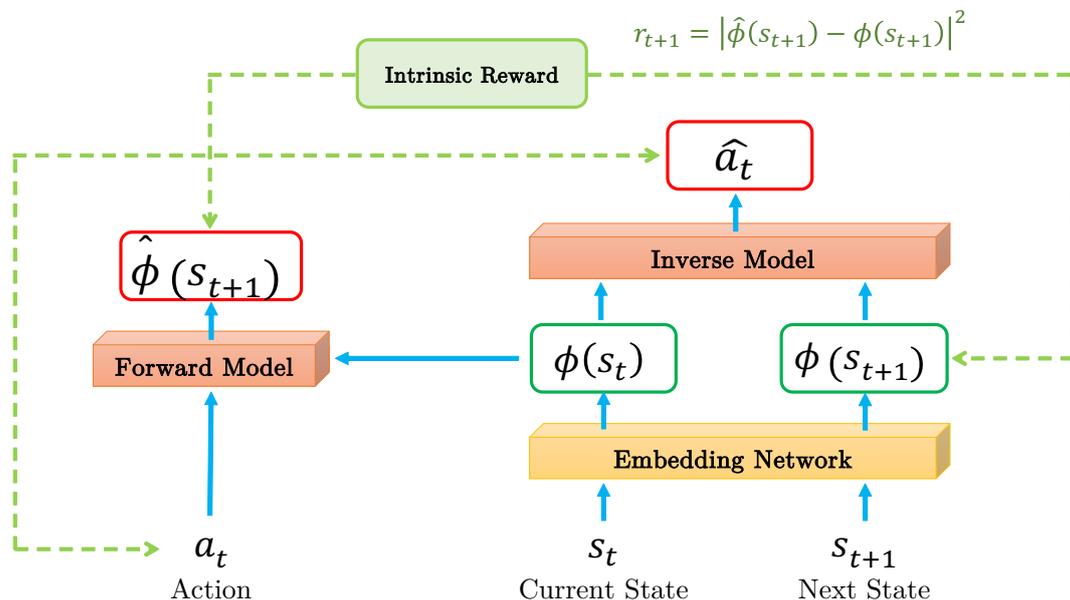


Fig. 3.1: Intrinsic Curiosity Module (ICM) のアーキテクチャの概要

利用せずに ICM の生成する好奇心の内部報酬のみを報酬として用いる学習の実験を行い、ICM の詳細な検証を行った。その結果として、ICM の抱える 2 つの問題点が明らかになった。

一つは、実質的に環境の状態の特徴表現を獲得するために用いられていた逆モデルに定常性がないという問題である。逆モデルは継続的に学習を行う必要があるが、その結果として、同じ状態の入力に対しても、学習の結果としてパラメータが更新されているために、出力される特徴表現も変わってしまうという問題がある。この非定常性が学習にどれほどの影響を与えるかは明らかではないが、実際に一部のゲーム環境において非常に不安定な結果を示していたため、問題点として指摘されている。逆モデルの改善案として、定常的な特徴表現を得る手法で、ランダムに初期化されたパラメータで固定された CNN を特徴変換用のネットワークとして用いる Random Features が提案されており、多くのゲーム環境で安定した結果を示している。

もう一つが、The Noisy-TV Problem [20] と呼ばれる問題である。これは、ICM の生成する好奇心の内部報酬のみを用いてゲームの学習を行う際に、ゲームの環境の状態遷移のランダム性が高いと学習が不安定になる可能性が高いという問題である。ある状態と行動から次状態を予測するという ICM (順モデル) の構造上、そこでの状態遷移が確率的に行われる場合、何度その遷移を経験してモデルを学習しても、どうしても一定以上の予測誤差が出てしまう。決定的な状態遷移の場合は、その遷移を十分に経験し学習すれば、その遷移で計算される予測誤差、つまり内部報酬はほぼ 0 になるため、エージェントは他の遷移を探索するように動機付けられる。一方で、確率的な状態遷移の場合は、ICM はその遷移に対して常に一定の内部報酬を生成してしまうため、エージェントはその遷移の探索を続けざるを得なくなってしまう、環境の探索に悪影響をもたらしてしまう。この問題の

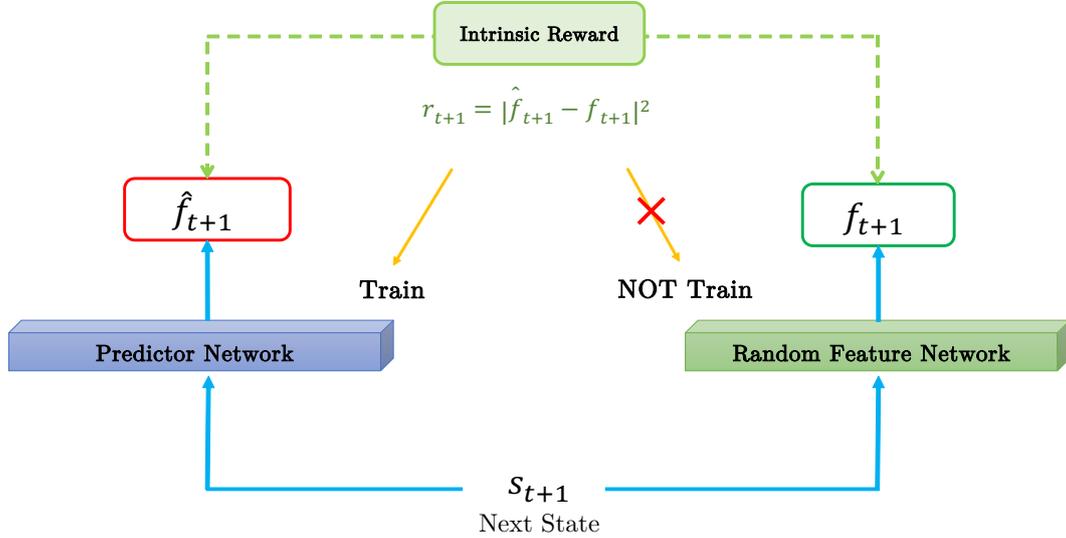


Fig. 3.2: Random Network Distillation (RND) のアーキテクチャの概要

解決には、ICM のモデル構造の見直しが必要となるだろう。

3.2 Random Network Distillation (RND)

Random Network Distillation (RND) [22] とは、Burda らによって提案された好奇心ベースの報酬生成手法である。前項で述べた ICM の抱える 2 つの問題点に着目し、ICM を改良したモデルとして提案されたものである。

The Noisy-TV Problem で指摘されたように、ICM は環境の確率的な状態遷移に弱いという問題があった。この問題を解決するために、RND では環境の次状態から直接次状態の特徴表現を推定するという手法を取る。この特徴表現の獲得には、Random Feature Network というランダムに初期化されたパラメータで固定された CNN を用いた特徴変換を用いている。RND では、環境の次状態 s_{t+1} から Random Feature Network によって得られる特徴表現 f_{t+1} を、次状態 s_{t+1} のみから Predictor Network により直接予測 (\hat{f}_{t+1}) し、その予測誤差

$$r_{t+1}^i = \left\| f_{t+1} - \hat{f}_{t+1} \right\|^2 \quad (3.7)$$

を好奇心の内部報酬として利用するという、とてもシンプルなアーキテクチャを持つ。ここで重要なのが、学習するのは Predictor Network のみであり、Random Feature Network は学習しないという点である。これにより、Random Feature Network によって獲得される特徴表現に定常性がもたらされる。Predictor Network については、単純に予測誤差 $\left\| f_{t+1} - \hat{f}_{t+1} \right\|^2$ を最小化する最適化を以って学習が行われる。Fig. 3.2 に RND のアーキテクチャの概要を示す。

次状態を直接推定するため、そこでの環境の遷移が確率的なものであったとしても、遷移の結果だけを学習すればよく、ある程度のランダム性ならば、この遷移の経験と学習を繰り返せば、いずれの結果の状態に対してもモデルがうまく予想ができるようになり、エージェントは他の遷移を探索するように動機付けられることが期待される。

RND の論文 [22] では、Atari 2600 の 6 種類のゲーム環境に対して、外部報酬と RND が生成する内部報酬を組み合わせる PPO [18] による強化学習を行った実験が紹介されており、Gravitar と Montezuma's Revenge、Venture の 3 種類のゲームについては、当時の State-of-the-art を獲得するほどの高い成果が報告されている。特に、Montezuma's Revenge については、ゲーム中に確率的な状態遷移が多く、The Noisy-TV Problem を抱える ICM ではうまく学習できなかった [20] が、RND では非常にうまく学習できていたため、The Noisy-TV Problem の観点から RND は大きな貢献をもたらしたと言える。

第4章 提案手法

4.1 好奇心ベースの手法によるローグライクゲームの学習の提案

本研究の目的は、非常に複雑な環境を持つゲームであるローグライクゲームを攻略できるようなAIを学習することである。前項で説明したように、ローグライクゲームには、環境の高いランダム性や、POMDPのような部分観測性、報酬の sparse 性など、AIの学習を難しくする様々な要素が課題として存在する。既存の手法では、ダンジョンの探索を十分に行えるほどの学習が難しく、アイテムやモンスターなどの要素を排除したシンプルな環境においてダンジョンの探索性能を向上させるような手法の研究が多い。

Rogueを扱ったAspertiらの先行研究[28]では、ランダムに生成される全体観測可能なダンジョンにおいてゴールを目指すという要素だけに単純化したRogueの環境において、Actor-Criticな強化学習アルゴリズムであるA3C (Asynchronous Advantage Actor-Critic) [15]、またはOff-policyで経験再生を利用できるようにA3Cを改変したACER (Actor Critic with experience replay) [43]の手法に、近い入力系列の情報を記憶のように用いることができるLSTM (Long-Short Term Memory) [34]を組み合わせた学習を行う手法を提案し、ダンジョン探索性能の向上に関して一定の成果を上げているものの、今後ローグライクゲームの要素をより追加したゲーム環境を学習することを目的とする場合には、全体観測可能なMDPな環境を前提としているという面で、未だに不十分であるとも考えられる。

そこで本論文では、環境が部分的にしか観測できないようなPOMDPの性質を持つローグライクゲームにおいて、ダンジョンの探索が十分に行えるように学習できるようなアーキテクチャの実現を目的とした。ローグライクゲームの持つ性質として、ゲーム内の探索が進むほど観測できる環境の範囲も広がっていくというものがあり、ダンジョンの探索がより効率的に行えるようになればPOMDPな環境においても学習は難しくならないと仮定し、効率的な探索を行うためのアプローチとして好奇心ベースの報酬生成手法を用いることを提案する。

好奇心ベースの報酬生成手法について、前述のようにICM [21]においては、環境のランダムな状態遷移に弱いというThe Noisy-TV Problem [20]と呼ばれる問題があり、ランダム性の高い環境を持つローグライクゲームとの相性は悪かったが、この問題を解決したRND [22]が提案されたことにより、ローグライクゲームに好奇心の内部報酬を適用できるようになった。

提案するアプローチは、深層強化学習の手法であるProximal Policy Optimization (PPO) [18]に、好奇心の内部報酬を生成する手法であるRandom Network Distillation (RND) [22]を組み合わせ、ローグライクゲームの学習を行うことである。既存手法の組み合わせではあるが、特に好奇心ベースの報酬生成手法をローグライクゲームの学習に適用したという点で、先行研究にはない新規性を

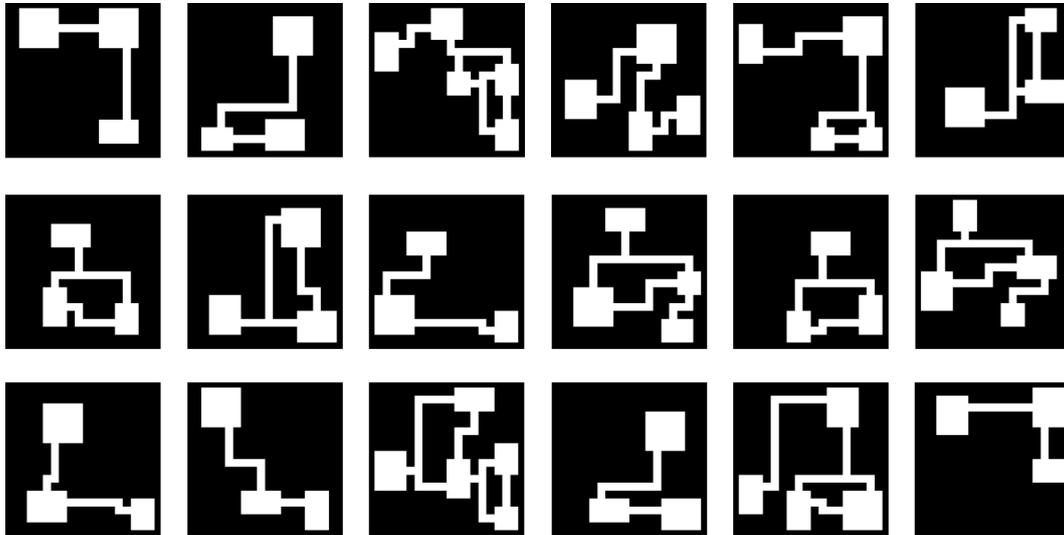


Fig. 4.1: 生成されるダンジョンの形状の例

見出すことができる。

4.2 評価実験用のゲーム環境

4.2.1 概要

前項でも述べたように、ログライクゲーム研究用のゲーム環境としては、Rogue-Gym [24] や Rogueinabox [25, 26] などが提案されているが、本稿では、環境の扱いやすさやカスタマイズ性の観点から、これらのゲーム環境ではなく自作のゲーム環境を評価実験用に用いた。ここでは、そのゲーム環境の概要を説明する。

本稿で評価実験用に実装したゲーム環境は、敵やアイテムなどの要素が存在しない、ログライクゲームとしての難易度は低いゲーム環境ではあるが、ログライクゲームのダンジョン探索におけるランダム性の高さや部分観測性を再現した環境である。Fig. 4.1 に、このゲーム環境が生成しうるダンジョンの形状の例を 18 個提示する。この例では、全体サイズが 20×20 マスのダンジョンとなっており、画面上白く表示されているマスはプレイヤーが移動することができる通路や部屋を、黒く表示されているマスはプレイヤーの移動を阻む壁を意味している。ここでは、画面上白く表示されているマスで、大きな長方形を描いている部分を部屋、これらの部屋の間を繋ぐ細さ 1 マスの部分を通路と呼んでいる。ここに例示した 18 個のダンジョンは、どれもがランダムに生成しうるダンジョンの一例であり、そのパターン数は十分に大きい。実装の観点からは、ダンジョンの全体サイズや、部屋のサイズや生成数の上限、通路の生成アルゴリズムなどもカスタマイズができるようにしてある。

ダンジョンの形状がランダムに決定された後に、プレイヤーの初期位置や階段（ゴール）の位置

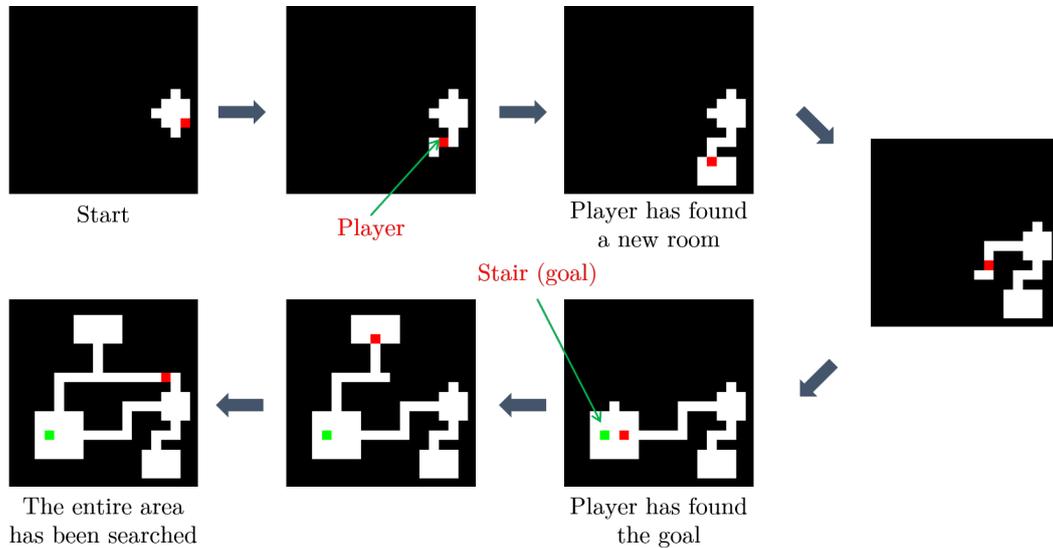


Fig. 4.2: プレイヤーによるダンジョン探索の例

が、部屋として定義されているマスの中からランダムにそれぞれ決定される。プレイヤーは上下左右のいずれかの方向に1マスだけ進むという行動を選択できるが、進む先が壁（黒いマス）だった場合は移動することができずに1回分の行動を消費する。ゲームのクリア条件は、各ダンジョンでプレイヤーを階段のところまで移動させることである。ただし、Fig. 4.1で示されているようなダンジョンの全体図をゲーム開始時の時点でプレイヤーが観測できるわけではない。ここに、ローグライクゲームらしい部分観測的なダンジョン探索要素があり、その概要をFig. 4.2に示す。ここでの赤いマスはプレイヤーの現在位置を、緑のマスはたどり着くべき階段の位置を示している。ゲーム開始時においては、いずれかの部屋に配置されたプレイヤーからは、その部屋全体とそれにつながる通路の入り口部分のマスしか移動可能なマスとして観測できず、未観測の部分は壁と同じように見えてしまう。しかし、プレイヤーが通路に入ってダンジョンを進んでいくと、プレイヤーの位置を中心に観測できる範囲が徐々に広がっていき、探索がしやすくなっていく。ダンジョンを十分に探索できれば、ダンジョン全体を観測できるようになる。もちろん、今回の目的は階段にたどり着くことなので、ダンジョン全体を観測できるようになる前に探索を終えてゴールに到達しても良い。

強化学習の観点から考えると、このゲーム環境の持つランダム性（ダンジョンの形状、プレイヤーの位置、階段の位置がランダムで決定される）と部分観測性（ダンジョンを部分的にしか観測できない POMDP な環境）が組み合わさって、学習の際の課題となることが予想される。

4.2.2 エピソードの定義

強化学習では、環境の初期状態から終端状態に至るまでの行動の軌跡をエピソードと定義し、エピソード内での行動をステップと定義することが多く、本稿でもこの定義を用いる。

本稿における評価実験では、学習を高速化するために、エピソードごとに毎回新しいダンジョンをランダムに生成して学習に用いるのではなく、何種類かのダンジョンを予め生成しておき、エピソードごとにそれらのダンジョンの中から毎回 1 個を選んで学習に用いるという形式をとっている。つまり、予め生成しておくダンジョンの数が 1 個のハイパーパラメータになる。これにより、ログライクゲームの生成するダンジョンのランダム性はある程度失われるものの、このパラメータの値が小さすぎなければ、プレイヤーの初期位置やゴールの位置もランダムに決定されることから考えても、ログライクゲームらしいランダムさは保たれていると言える。実際、一部のログライクゲームのシリーズには、ダンジョンの形状パターンが数百個程度に収まるものもある。

評価実験においては、以下のようにエピソードを定義する。エピソードが終わるごとに環境を初期化することを繰り返すという形式で学習が行われる。

- 予めランダムに生成された N_d 個のダンジョンから 1 個を選び、そのダンジョンで環境を初期化する。
- ダンジョン内における部屋として定義されるマスの中からランダムに 2 箇所を重複しないように選び、それぞれを「プレイヤーの初期位置」と「階段（ゴール）の位置」のマスとして初期化する。
- 各ステップでプレイヤーは 4 種類の行動（上下左右のいずれかの方向に 1 マス分移動する）の中から 1 種類の行動を選択し実行する。
- 各ステップでプレイヤーは、現在のゲームの状態を部分的に観測することができる。(Fig. 4.2 参照)
- プレイヤーが T ステップ以内に階段に到達することを「ゲームクリア」と呼ぶ。
- プレイヤーが階段に到達する前に T ステップが経過してしまうことを「時間切れ」と呼ぶ。
- プレイヤーがゲームクリアするか時間切れになるまでの各ステップの集合を、1 個のエピソードとして定義する。
- エージェントに与えられる外部報酬は、ゲームクリアを達成した瞬間のステップに対しては +1 で、それ以外の場合は +0 である。

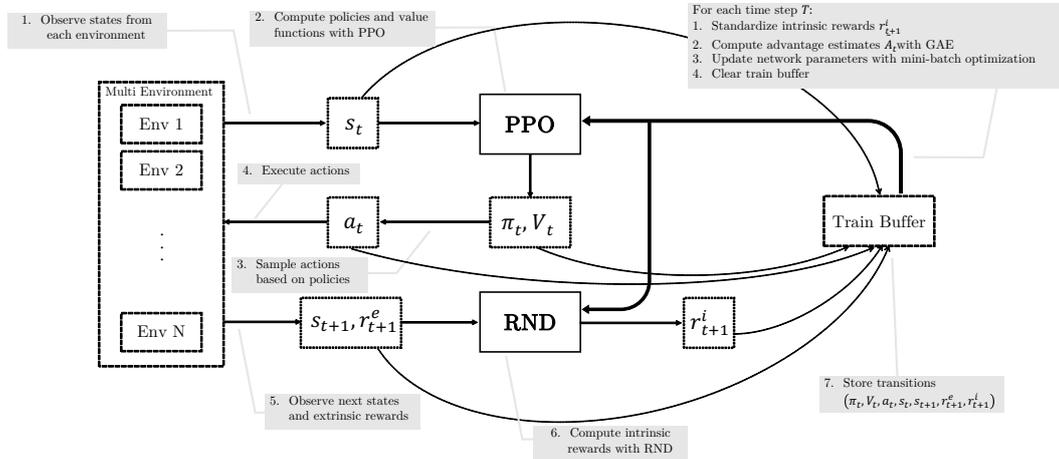


Fig. 4.3: PPO と RND による学習モデル

4.3 好奇心ベースの報酬生成手法の適用

4.3.1 提案モデルの概要と実験設定

ここでは、深層強化学習の手法である Proximal Policy Optimization (PPO) [18] に、好奇心の内部報酬を生成する手法である Random Network Distillation (RND) [22] を組み合わせた学習モデルについて説明する。本論文で実装したマルチエージェントによる学習モデルの概要を Fig. 4.3 に、アルゴリズムを Algorithm 4 に示す。マルチエージェントによる分散学習を行うことにより、複数環境からのエージェントの経験データを効率的に収集することができ、結果としてモデルの学習もより効率的に行うことができる。また、評価実験に用いたネットワークモデルの詳細な実装を Fig. 4.4 と Fig. 4.5 にそれぞれ示す。

次に、詳細な実験設定について説明する。まず、RND のモデル実装の際には、random feature を出力する部分のネットワーク (Fig. 4.5 における薄灰色で強調された部分) のパラメータをランダムに初期化した値で固定し、学習の際にパラメータが更新されないように注意して設定する必要がある。

また、今回実装した PPO のアルゴリズムでは、各エージェントに対して同期的な学習が行われる。今回の評価実験では、各ステップをそれぞれのエージェントに対して同期的に処理し、 $T = 2048$ ステップごとに Train Buffer に貯められた遷移データを用いてネットワークを学習している。

続いて、今回の実験においては、RND が生成した内部報酬に関してはエピソードごとに、GAE によって計算されたアドバンテージ関数の推定値はバッチごとに標準化して学習に用いている。標準化とは、与えられたデータの系列 $x = \{x_1, x_2, \dots, x_n\}$ を、平均が 0 で分散が 1 であるようなデー

タの系列に変換させる操作のことをいう。 x の平均を μ 、分散を σ^2 とすれば、系列内の各データ x_i は、次の演算により標準化されたデータ z_i に変換される。

$$z_i = \frac{x_i - \mu}{\sigma} \quad (4.1)$$

しかし、データの標準化は、対象となるデータの標準偏差が 0 である場合には、ゼロ除算が発生してしまう。そこで今回の実験では、次の式のように十分に小さい値 $\epsilon = 1e-8$ を用いて標準偏差 σ の値を補正している。そのため、標準化によって得られるデータの系列の標準偏差は厳密には $\sigma = 1$ ではなくってしまうものの、 ϵ の値を十分に小さく設定していれば、学習に大きな影響を与えることはないと考えられる。

$$z_i = \frac{x_i - \mu}{\sigma + \epsilon} \quad (4.2)$$

今回の実験における標準化の意図は、基本的には学習の安定性を図るためであるが、RND の内部報酬については報酬の値を安定させる目的もある。RND では、学習が進むにつれて状態の予測精度が当然向上していくため、RND の生成する内部報酬は、エピソード内における値の大小はあっても、エピソード全体の値の平均値は減少し続ける傾向にある。これは RND の性質を考えれば正しい動作ではあるが、強化学習の報酬として用いる場合には学習に悪影響を与えてしまう可能性もある。そこで、エピソードごとに内部報酬を標準化することによって、内部報酬の減衰を抑制することができ、安定した学習が可能となる。特に、RND の内部報酬を外部報酬と組み合わせて用いる際には、外部報酬の値に対する内部報酬の値が減衰していかないため、高い内部報酬を生成するような遷移をより捉えやすくなり、標準化がより効果的な操作となることが期待できる。ただし、その場合は外部報酬と内部報酬の比率が重要となる。アルゴリズムの中で、外部報酬 r_{t+1}^e と標準化された内部報酬 r_{t+1}^i を用いて、実報酬 R_{t+1} を計算する箇所があるが、その際の計算では $R_{t+1} = r_{t+1}^e + \rho r_{t+1}^i$ と、内部報酬に報酬比率 ρ (今回の実験では定数) をかけるという操作を行う。この値は学習の際に内部報酬をどれだけ考慮するのかということの意味しており、この値が大きいほど好奇心に動機づけられた学習することになる。

評価実験における、モデルのハイパーパラメータなどに関する詳細な情報は Table. 4.1 に示した。今回の実験では、16 個のエージェントと環境を並列にシミュレートさせる分散学習を行っており、各エージェントからそれぞれ 2,048 個の遷移データを Train Buffer に集める (合計 32,768 個の遷移データ) ごとに、その Train Buffer をバッチとして 5epoch 分の学習を行い、PPO と RND のパラメータを更新していく。この際、Optimizer には、PPO や RND の論文実装 [18, 22] でも用いられている Adam [38] を採用した。Adam のハイパーパラメータの具体的な意味については、Adam の論文を参照してもらいたい。GAE や PPO のハイパーパラメータの意味については、第 2.3.2 項に記載してある。ローグライクゲームの環境については、画面サイズが $20 \times 20 \times 3$ の easy な環境と、 $22 \times 80 \times 3$ (これは Rogue の画面サイズと同じ) の hard な環境を用意した。各チャンネルはそれぞれ壁の位置、プレイヤーの位置、ゴールの位置を 0,1 でマッピングしたものとなっている。これを各 50 個予めランダムに作成して学習に用いるが、同じ難易度の環境で行われた実験の比較を容易にするために乱数の seed 値は固定した。また、報酬比率 ρ は 0.005 とした。

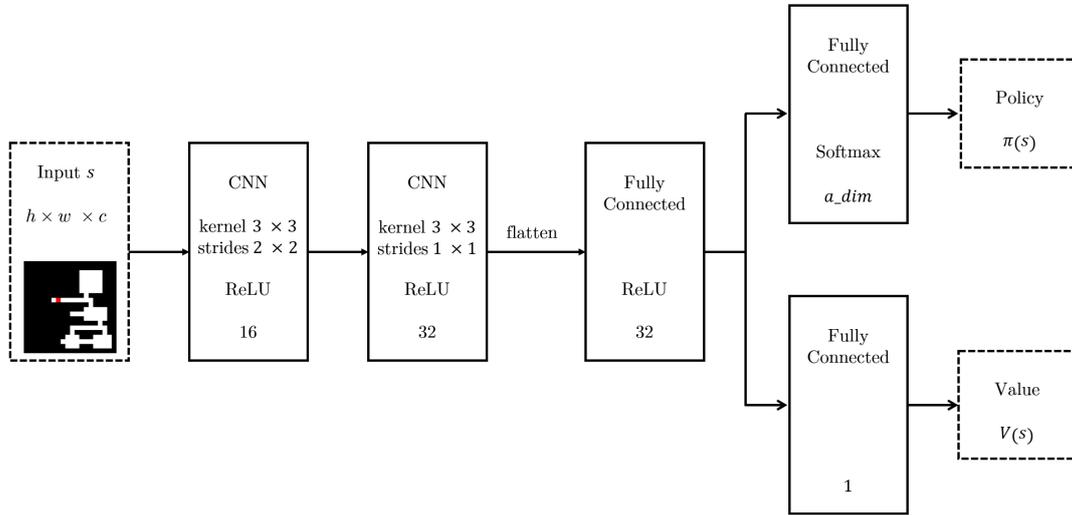


Fig. 4.4: PPO のネットワーク設定

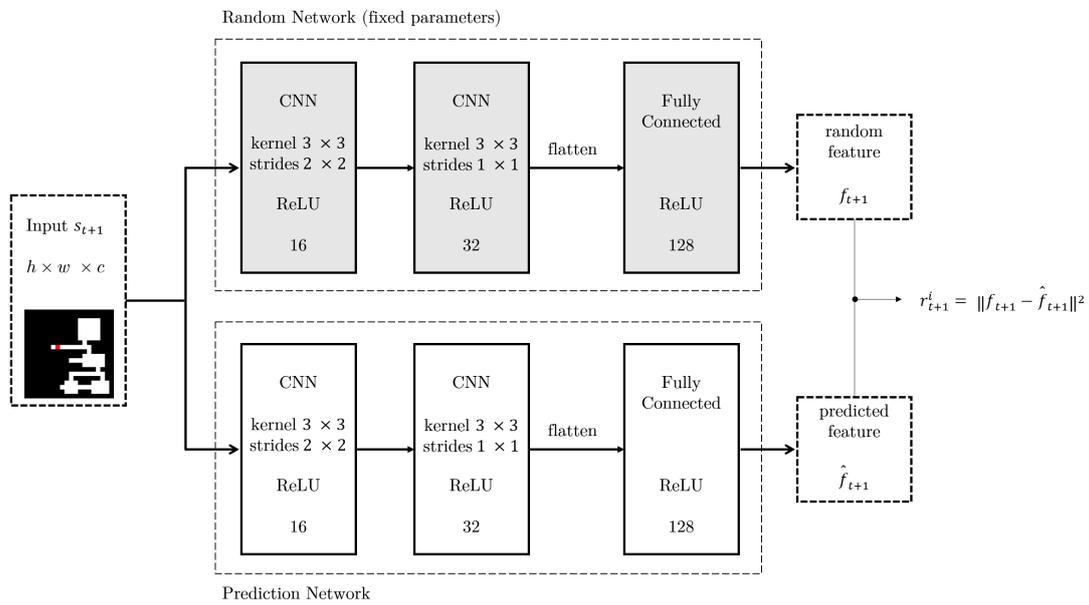


Fig. 4.5: RND のネットワーク設定

Algorithm 4 Reinforcement Learning Exploration with PPO [18] and RND [22]

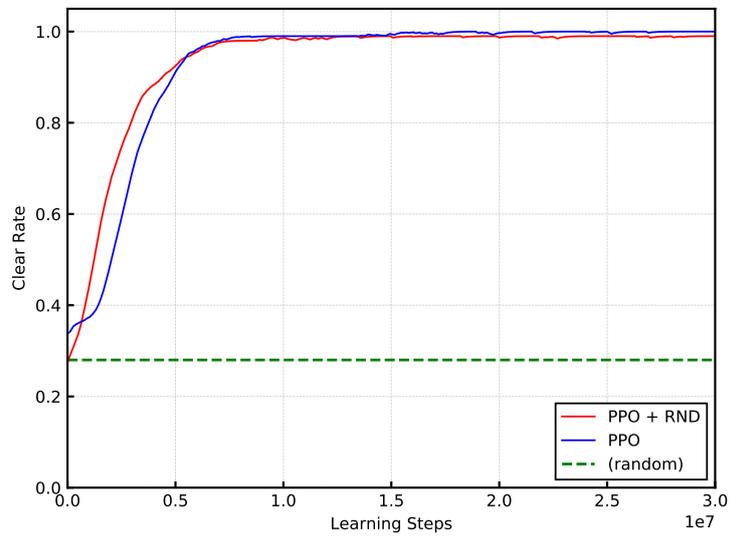
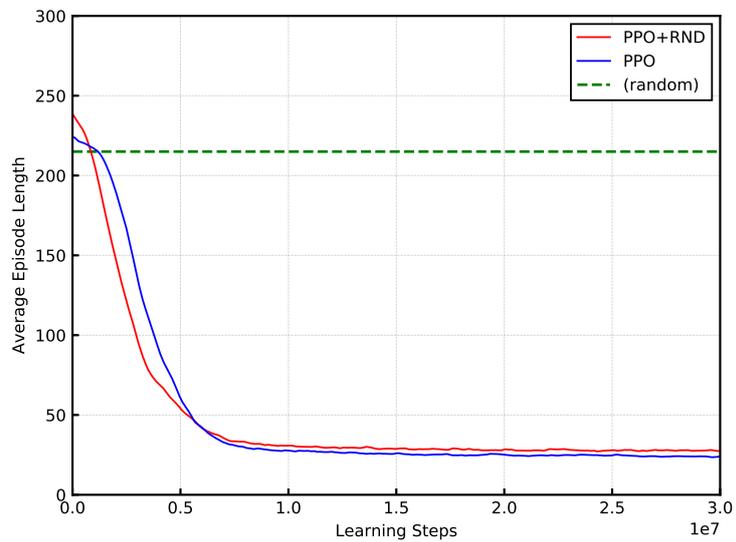
$N \leftarrow$ number of actors
 $K \leftarrow$ number of optimization epochs
 $D = \{D_1, \dots, D_N \leftarrow\}$ transition memory for each actors
Initialize PPO policy parameters with θ_{ppo} , and value function parameters with ϕ_{ppo}
Initialize RND random network parameters with θ_{rnd}
Initialize RND prediction network parameters with θ_{pred}
for each iteration $i = 1, 2, 3, \dots$ **do**
 for each actor $j = 1, 2, \dots, N$ **do**
 Reset the environment E_j , and observe initial state s_0
 for each step t of episode, state s_t is not terminal **do**
 ComputeCom policy $\pi_t = \pi_{\theta_{ppo}}(s_t)$ and value $V_t = V_{\phi_{ppo}}(s_t)$
 Sample action $a_t \sim \pi_t(a_t | s_t)$
 Execute action a_t , and observe next state s_{t+1} and extrinsic reward r_{t+1}^e
 Compute intrinsic reward $r_{t+1}^i = \left\| f_{\theta_{rnd}}(s_{t+1}) - \hat{f}_{\theta_{pred}}(s_{t+1}) \right\|^2$
 Store transition $(\pi_t, V_t, a_t, s_t, s_{t+1}, r_{t+1}^e, r_{t+1}^i)$ in D_j
 end for
 Standardize intrinsic rewards in D_j
 Compute returns R_{t+1} with extrinsic rewards and standardized intrinsic rewards
 Compute advantage estimates \hat{A}_t from the set of transitions D_j of length T based on the current value function $V_{\phi_{ppo}}$ with GAE
 Store returns and advantage estimates (R_{t+1}, \hat{A}_t) in D_j
 end for
 Standardize advantage estimates in D
 Combine D_1, \dots, D_N as an optimization batch B
 for each optimization epoch $i = 1, 2, \dots, K$ **do**
 Sample a minibatch of transitions $(\pi_t, V_t, a_t, s_t, s_{t+1}, R_{t+1}, \hat{A}_t)$ from B
 Update the PPO network parameters θ_{ppo}, ϕ_{ppo} by minimizing the PPO objective $L_t^{\text{CLIP+VF+S}}(\theta_{ppo}, \phi_{ppo})$ on the minibatch, via optimization algorithm Adam [38]
 Update the RND prediction network parameters θ_{pred} by minimizing the RND objective $L_t^{\text{RND}}(\theta_{rnd}, \theta_{pred}) = \left\| f_{\theta_{rnd}}(s_{t+1}) - \hat{f}_{\theta_{pred}}(s_{t+1}) \right\|^2$ on the minibatch without updating the RND random network parameters θ_{rnd} , via optimization algorithm Adam
 end for
 Clear transition memory D_1, \dots, D_N and optimization batch B
end for

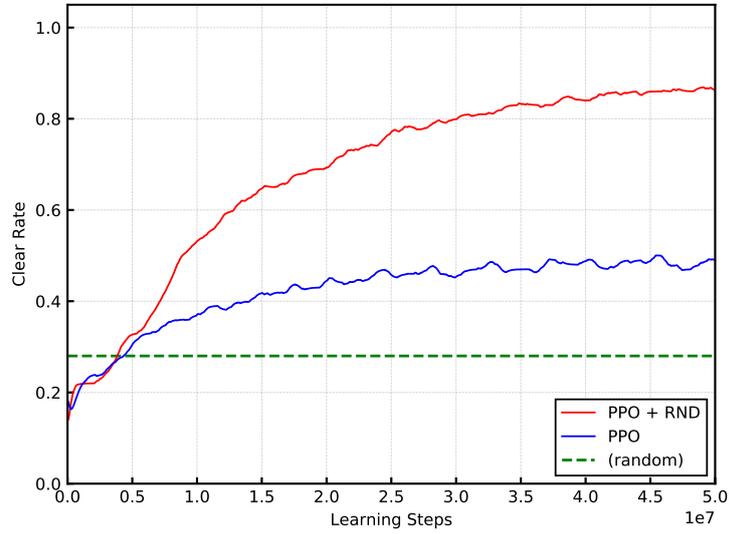
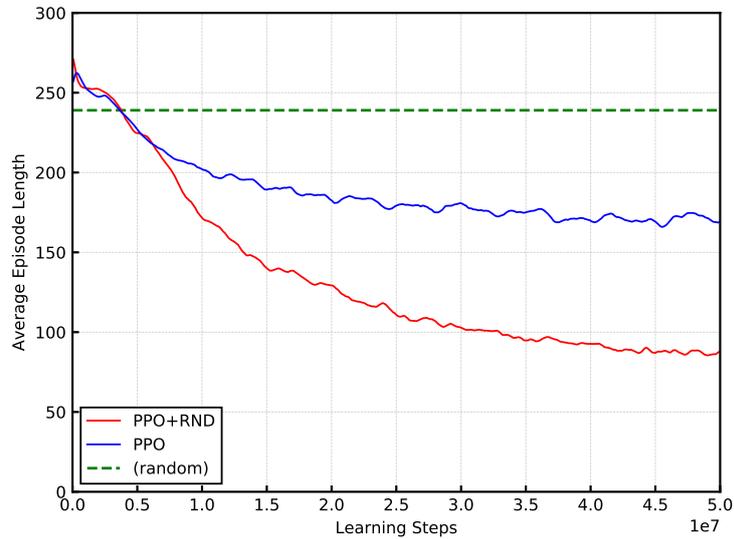
Table. 4.1: 実験時のハイパーパラメータ設定

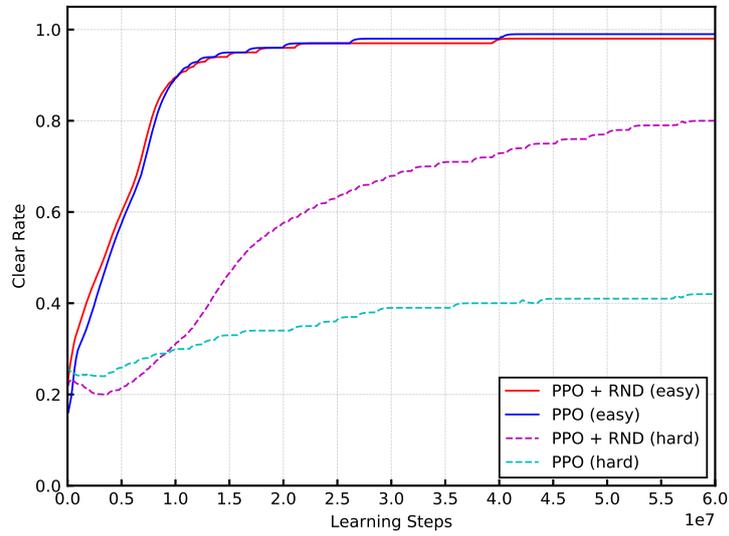
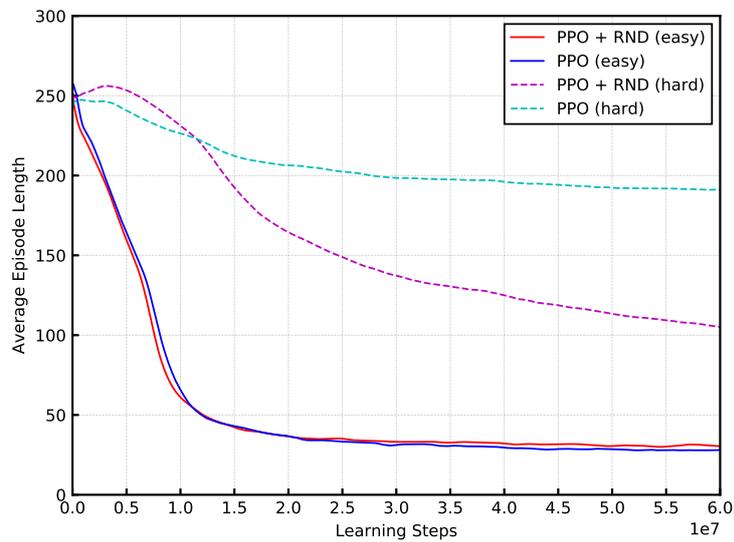
Parameter	Value
parallel actors	16
dungeon size $w \times h \times c$ (easy)	$20 \times 20 \times 3$
dungeon size $w \times h \times c$ (hard)	$22 \times 80 \times 3$
dungeon types N_d	50, (500)
episode max length T	300
iteration step length	2048
optimization epochs	5
minibatch size	64
optimizer	Adam [38]
learning rate lr	$3e-4$
Adam decay rate β_1	0.9
Adam decay rate β_2	0.999
Adam constant ϵ_1	$1e-7$
GAE discount factor γ	0.99
GAE smoothing parameter λ	0.95
PPO value function loss weight c_1	1.0
PPO entropy loss weight c_2	0.01
PPO clipping parameter ϵ_2	0.2
reward ratio ρ	0.005

4.3.2 実験結果

Fig. 4.6-4.9 に、 $N_d = 50$ とした時の実験結果を示す。Fig. 4.6, 4.8 は、各エピソードにおけるゲームのクリア率の推移を表しており、横軸をそれまでに学習した遷移（ステップ）の合計値、縦軸をそれぞれのエージェントの最新 100 エピソードにおけるゲームのクリア率の平均値としてプロットしている。Fig. 4.7, 4.9 は、各エピソードの長さの推移を表しており、横軸は同じくそれまでに学習した遷移（ステップ）の合計値、縦軸はそれぞれのエージェントの最新 100 エピソードの長さの平均値を意味している。エピソードの長さは、実質的にはエージェントがゲーム開始からゴールするまでに必要だった action 数を意味しているが、時間切れの場合はそれをエピソードの最大長 $T = 300$ として処理している点に注意してもらいたい。また、今回の評価実験では、RND による好奇心の内部報酬を環境の外部報酬と組み合わせて PPO による学習を行うという提案モデルを用いた実験（PPO+RND と表記）だけでなく、結果比較用のベースラインとして、外部報酬のみを PPO の学習に用いた実験（PPO と表記）も行った。

Fig. 4.6: 実験結果：ゲームクリア率の推移 (easy) ($N_d = 50$)Fig. 4.7: 実験結果：エピソードの長さの推移 (easy) ($N_d = 50$)

Fig. 4.8: 実験結果：ゲームクリア率の推移 (hard) ($N_d = 50$)Fig. 4.9: 実験結果：エピソードの長さの推移 (hard) ($N_d = 50$)

Fig. 4.10: 実験結果：ゲームクリア率の推移 ($N_d = 500$)Fig. 4.11: 実験結果：エピソードの長さの推移 ($N_d = 500$)

まず、easy な設定の環境における結果 (Fig. 4.6, 4.7) を見ていこう。各図における random と表記された緑色の点線は、完全にランダムな行動選択を行うエージェントによる平均スコアを示しており、その平均クリア率は 37%、平均エピソード長は 215 である。ランダムなエージェントでもある程度のクリア率を示しているのは、ランダムに決定されるプレイヤーの初期位置とゴールの位置が同じ部屋にあることが多く、ランダムな行動選択でも容易にゴールにたどり着けてしまう場合があるということが大きな理由として挙げられるだろう。さて、提案モデルとベースラインモデルの実験結果であるが、どちらも学習序盤からランダムなエージェントの結果を大きく引き離し、最終的にかなり高い確率でゲームをクリアできる AI を学習することができたが、最終的にはベースラインモデルのほうが僅かに高い性能を示していることが観察できる。

次に、hard な設定の環境における結果 (Fig. 4.8, 4.9) を見ていこう。この環境は easy な設定と比較するとおおよそ 4 倍ほどの大きさのダンジョンを持ち、完全にランダムな行動選択を行うエージェントによる平均クリア率は 28%、平均エピソード長は 239 である。こちらも easy の設定の場合と同様、ランダムなエージェントでもある程度の確率でゲームをクリアすることができるが、ゲームの難易度は確実に上がっていることが分かる。さて、提案モデルとベースラインモデルの実験結果であるが、PPO のみのモデルでは環境の学習に苦戦しているように見えるのに対し、提案モデルである PPO+RND による AI はベースラインを大きく超えるスコアを獲得している。具体的には、Learning Steps $5e+7$ 経過時点でのスコアが、PPO で平均クリア率 48% と平均エピソード長 171 なのに対し、PPO+RND で平均クリア率 86% と平均エピソード長 89 である。

続いて、予め生成しておくダンジョン数を $N_d = 50$ 個から $N_d = 500$ 個に増やした場合の実験も、同様の設定で行った。その結果を Fig. 4.10, 4.11 に示す。この場合、ゲーム環境の難易度はわずかに難しくなっていると考えられ、実際 $N_d = 50$ の時の結果と比べると最終的に学習される AI の性能はわずかに落ちているものの、学習曲線の傾向はほとんど変わらないことが確認できる。

4.3.3 考察と課題

最初に easy な設定の環境における結果 (Fig. 4.6, 4.7) を考察する。ここで注目すべき点は、提案モデルである PPO+RND 方が学習序盤においてはより高い性能を示している点と、最終的にはベースラインモデルである PPO のみの方がこのタスクにおいてはわずかに高い性能の AI を学習している点である。まず、前者の点について理由としては、RND の生成する内部報酬により、エージェントが好奇心によって動かされ、より効率的なダンジョン探索が学習序盤から行えるようになったということが予測される。強化学習では、効果的な探索が方策の学習に大きく貢献するため、好奇心の内部報酬の利点がここに発揮されていると考えることができるだろう。後者の点の理由としては、エージェントがゴールを発見したとしても、好奇心で動機づけられたエージェントがゴールに到達することよりもダンジョンの他のエリアを探索することに価値を見出してしまい、ゴールすることを後回しにしてしまう場合があることが考えられる。この性質は、今回の問題設定のようにゴールに到達することに特化したタスクを解く場合には問題となるが、今後ローグライクゲームの環境をより拡張させた場合には、ゴールを見つけるだけではなくその先のフロアに向けてアイテムの収集や経験値稼ぎなどする必要もあることを考えると、むしろ効果的な性質であるとも捉えることがで

きる。もし、ゴールに到達することに特化したタスクを解かせたい場合は、学習の過程で報酬比率 ρ を一定の割合で減衰していくなどの工夫が考えられるだろう。

一方で、hard な設定の環境における結果 (Fig. 4.8, 4.9) は、easy の設定の環境における結果と大きく傾向が異なり、提案モデルである PPO+RND が、ベースラインである PPO のみのモデルを大きく超える性能を示している。これには、easy な環境が外部報酬のみの PPO でも十分に学習ができるほど簡単な環境だったのに対し、hard な環境の学習は外部報酬のみでは非常に難しく、RND による好奇心の内部報酬の効果がより大きく得られたことが理由として考えられるだろう。しかしながら、easy な設定での実験で見られた、学習序盤における PPO-RND の優位性はここではあまり観察できなかった。その理由としては、ネットワーク初期値の影響もわずかに考えられるが、環境がより複雑になった結果、RND が意味のある報酬を生成するまでに必要な学習量が easy な設定の環境と比べて多くなってしまったために、好奇心の報酬がエージェントの探索に影響を与えるまでの時間が長くなってしまった可能性が主として考えられる。

ダンジョンの形状のランダム性の低さが学習の結果に影響を与えることを懸念して行った検証実験の結果 (Fig. 4.10, 4.11) に関しては、当然ゲームの難易度が高くなってしまいうため性能はわずかに下がっているものの、よりランダム性を高くした $N_d = 500$ での結果が $N_d = 50$ の時の結果とほぼ同様の傾向を見せており、 N_d に対する提案モデルの頑強性が確認できたと考えられる。

しかしながら、評価実験を行う過程で、一部のハイパーパラメータを変化させた際に実験に与える影響についても検証を行っていたが、特に提案モデルである PPO+RND に関しては、学習の結果が報酬比率 ρ の値に大きく依存することが分かった。

実験結果で示したのは報酬比率を $\rho = 0.005$ とした時の結果であるが、例えば $\rho = 0.01$ と少し大きな値に変更しただけで easy と hard 両方の設定において学習があまり進まなくなってしまうことを確認した。しかし、 ρ の値をこれ以上小さくしてしまうと、外部報酬に対して内部報酬の値が小さくなりすぎて好奇心が無視されてしまう可能性も考えられる。このように、ハイパーパラメータの設定が学習に大きく影響を与えてしまうのは好ましくない問題であり、これは RND の好奇心の報酬に限らず、何らかの手法で得られた内部報酬を環境の外部報酬と組み合わせて用いて強化学習を行う際の課題になると考えられる。

続いて、ログライクゲームに RND を適用することの観点から、RND の抱える課題が改善可能性について検討する。

まず、RND がこれまでに高い成果を挙げたゲーム環境とログライクゲームの環境の違いに注目し、RND とログライクゲームとの相性について論ずる。第 3 章で述べた好奇心ベースの報酬生成手法に関して、論文中で評価実験の対象として用いられ、好奇心により探索性能が向上することが確認されたゲームである Super Mario Bros. や Montezuma's Revenge などのゲーム環境やその際の実験設定を考えてみると、これらのゲームでは「見たことがない状態を観測する」ことが直接的に「ゲームをクリアする」という目的に結びついているだけではなく、エピソードを開始するごとに環境が同じ状態に初期化されるので、エージェントは常に同じ初期状態からゲームの学習を再開することができたため、これらの要因が好奇心の報酬の効果をより高めていたと言える。毎回同じ状態から学習を始めることの利点は、ゲームの初期状態に近い場面に対して生成される好奇心の報酬が学習が進むにつれて小さくなっていくために、初期状態からより遠い場面へ進んでいくことが好奇

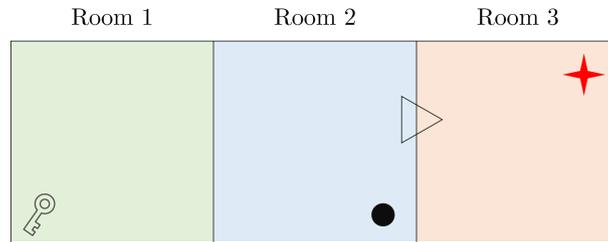


Fig. 4.12: 好奇心ベースのアルゴリズムによる問題の例 [44]。黒い円がエージェントの初期位置を示している。エージェントは、部屋 1 にある鍵を使って部屋 2 と部屋 3 を繋ぐ扉を開けることができる。エージェントの目的は、部屋 3 内の赤い星で示されたゴールに到達することである。

心によって動機づけされやすくなる点にある。また、これらのゲームにおいては、初期状態から遠い場面へ進んでいくことが、結果的にゲームをクリアできる状態に近づくことを意味してしまうために、好奇心の報酬が学習に与える効果が非常に大きくなるのだと考えることができる。一方でローグライクゲームでは、エージェントは毎回異なる初期状態からゲームの学習を再開しなければならないが、初期状態に近い状態に対しても高い好奇心の報酬が生成される可能性がある。また、ローグライクゲームにおいてより好奇心が揺さぶられる、つまりより新しい状態が観測されるような状況には、例えば見えていないエリアに存在する新しい部屋を発見することなどが挙げられるが、これらはゴールを見つけることには結びつくものの、見えていないエリアを探索することがゴールに到達してゲームをクリアすることに一致しないという問題がある。しかし、これは前述のように、ゲームの環境をよりローグライクゲームらしく拡張した時には、利点と見なすこともできる。また、これらのゲームには、ゲーム上の重要な遷移におけるゲーム画面の変化が非常に大きいという性質がある。例えば、ステージが切り替わるような場面においては、ステージの背景や配置オブジェクトが大胆に変化するため、環境の新規性が非常に捉えやすくなる。実際に、RND を使った Montezuma's Revenge における実験では、ゲーム画面の大きな変化を伴う重要な遷移に対して、標準化する前の内部報酬と比較して、通常の遷移の数倍の内部報酬が生成されているが確認できる。しかし、今回評価実験用に用いたローグライクゲームの環境では、本来重要な遷移として捉えるべき遷移に対して標準化前で通常の遷移の 1.0 – 1.5 倍程度の比較的小さい報酬を生成してしまう場合が多く、好奇心を活かしきれていない部分があると考えられる。

続いて、Hu ら [44] は、Fig. 4.12 のような状況を、RND の生成する好奇心の報酬では解決できないことを指摘している。この環境では、エージェントはまず部屋 2 から出て、部屋 1 の奥にある鍵を入手する必要がある。ここまでの行動は、RND の好奇心の報酬によって効率的に学習することができるが、問題はその後で、エージェントは鍵を見つけた後、再び部屋 1,2 を通って、部屋 3 に通ずる扉にたどり着かなければならない。しかし、観測の新規性を好奇心として扱う RND の性質上、一度通ってしまった部屋 1 や部屋 2 を再び通る際には、好奇心の報酬があまり生成されなくなってしまう。そのため、目的を到達することが難しくなってしまうのだ。同様の問題を Ecoffet らも指摘している [45]。つまり、RND は既に十分に学習したエリアに対してはほとんど報酬を生成できなくなるために、そのエリアの中、もしくは先に隠された未探索の部分を探ることが難しくなると

という問題がある。ローグライクゲームにおいても、ダンジョン探索の際に行き止まりの部屋に辿り着いてしまった場合には、辿ってきた道を引き返して探索をやり直す必要があるが、これはHuらが指摘するようにRNDにとっては難しい問題である。実際、学習済みのPPO+RNDのモデルでエージェントを動作させ、ゲームクリアに失敗したエピソードの内容をいくつか観察すると、エージェントが行き止まりの部屋に到達した後に、本来取るべき行動を見失っているような状態になる例がいくつか見受けられた。

最後に、好奇心の内部報酬をローグライクゲームに適用する上で大きな問題になりうると考えられるのは、ローグライクゲームにおいては、好奇心というものを学習全体で定義するのではなく、エピソードごとに定義しなければならないという点である。エージェントがはたらきかける環境がエピソードごとに変化するローグライクゲームでは、異なるエピソード間での環境は、同じゲームであるためある程度似てはいるものの、実際は大きく異なるものになってしまう。ローグライクゲームで好奇心の報酬を有効的に利用するためには、同エピソード内での状態観測の新規性というものをより重視すべきで、エピソードを区別せずに全ての状態観測を学習してしまっているRNDでは、あまり有効な報酬を生成することができないのではないかと問題が指摘される。しかし、単一エピソード内において、RNDのようなやり方で好奇心の報酬を生成するのは、状態予測の学習に用いることができるデータ量の観点から考えて非常に難しく、報酬の精度に大きな問題を抱えることになる。そこで、エピソード内での状態の変化を時系列で捉えて、適切な内部報酬をRNDのそれとは別に生成するような手法が必要になると考えられる。

第5章 結論

5.1 まとめ

本稿では、複雑なゲーム環境を強化学習で扱う際の課題として、適切な報酬関数を設計することの難しさを指摘した。そして、この問題に解決するためのアプローチとして、エージェントの好奇心に基づいて内部報酬を自動的に生成して学習の効率化を図るための手法を紹介した。そして、実際に非常に複雑な環境を持つゲームとしてコンピュータ RPG の一種であるログライクゲームを挙げ、このゲームを攻略できるような AI を学習するために、先行研究では見られなかったアプローチとして、好奇心に基づいて計算される内部報酬をログライクゲームの学習に適用することを考え、既存手法である PPO と RND を組み合わせた学習モデルを提案した。また、オリジナルに作成した簡単なログライクゲームの環境における評価実験を通して、提案手法がより効率的にゲームを攻略できる AI を学習できていることを確認した。

5.2 今後の課題

今後の課題としては2つ考えられる。一つは、アイテムやモンスターなどの要素、体力や空腹度、レベルなどのパラメータ、より多彩なアクションなど、ログライクゲームらしい要素がさらに追加されたより複雑な環境を対象とした学習を行えるようにすることである。その際、学習の際に考慮すべき状態が更に増加し、action space も大きくなるため、学習の難易度はより高くなることが想像される。先行研究でも未だに十分に研究されていないところではあるが、このようなログライクゲームでも効率よく攻略できる手法が見つければ、現実世界の問題へ強化学習の手法を応用するという目標にまた一歩近づくだらう。もう一つは、評価実験の考察で述べたように、ログライクゲームに RND の手法をそのまま適用することに対していくつかの潜在的な問題が考えられることである。RND を改善するためのアプローチとして、好奇心とは別の形で、ある単一のエピソード内における状態の変化を捉えて内部報酬を生成することを考察の最後で述べたが、具体的な手法として有効に機能するものを本研究では見つけることができなかつたため、今後の研究に期待したい。

参考文献

- [1] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, Vol. 135. MIT press Cambridge, 1998.
- [2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.
- [3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.
- [4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, Vol. 529, No. 7587, p. 484, 2016.
- [5] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, Vol. 134, No. 1-2, pp. 57–83, 2002.
- [6] Michael Buro. Logistello: A strong learning othello program. In *19th Annual Conference Gesellschaft für Klassifikation eV*, Vol. 2. Citeseer, 1995.
- [7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, Vol. 518, No. 7540, pp. 529–533, 2015.
- [9] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

- [10] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.
- [11] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, Vol. abs/1710.02298, , 2017.
- [12] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.
- [13] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.
- [14] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. 2018.
- [15] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, Vol. abs/1602.01783, , 2016.
- [16] 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*, pp. 1928–1937, 2016.
- [17] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.
- [18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, Vol. abs/1707.06347, , 2017.
- [19] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5392–5402, 2017.
- [20] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *arXiv:1808.04355*, 2018.

- [21] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 2778–2787, 2017.
- [22] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. 2018.
- [23] 高橋一幸, Sila Tlemsiririkkul, 池田心. ログライクゲームの研究用ルール提案とモンテカルロ法の適用. ゲームプログラミングワークショップ 2017 論文集, 第 2017 巻, pp. 19–25, nov 2017.
- [24] Yuji Kanagawa and Tomoyuki Kaneko. Rogue-gym: A new challenge for generalization in reinforcement learning. In *2019 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2019.
- [25] Andrea Asperti, Carlo De Pieri, Mattia Maldini, Gianmaria Pedrini, and Francesco Sovrano. A modular deep-learning environment for rogue. *WSEAS Transactions on Systems and Control*, Vol. 12, , 2017.
- [26] Andrea Asperti, Carlo De Pieri, and Gianmaria Pedrini. Rogueinabox: an environment for roguelike learning. *International Journal of Computers*, Vol. 2, pp. 146–154, 2017.
- [27] Andrea Asperti, Daniele Cortesi, and Francesco Sovrano. Crawling in rogue’s dungeons with (partitioned) A3C. *CoRR*, Vol. abs/1804.08685, , 2018.
- [28] ANDREA ASPERTI and DANIELE CORTESI. Reinforcement learning in rogue.
- [29] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, Vol. 4, pp. 237–285, 1996.
- [30] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, Vol. 6, pp. 679–684, 1957.
- [31] 白川真一 牧野貴樹. これからの強化学習. 森北出版, 2016.
- [32] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, Vol. 8, No. 3-4, pp. 279–292, 1992.
- [33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.
- [34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

- [35] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pp. 1057–1063, 2000.
- [36] Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*, pp. 151–160, 2019.
- [37] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- [38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [39] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, pp. 663–670, 2000.
- [40] George E Monahan. State of the art—a survey of partially observable markov decision processes: theory, models, and algorithms. *Management Science*, Vol. 28, No. 1, pp. 1–16, 1982.
- [41] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, Vol. abs/1606.01540, , 2016.
- [42] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *CoRR*, Vol. abs/1605.02097, , 2016.
- [43] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.
- [44] Hangkai Hu, Shiji Song, and Gao Huang. Self-attention-based temporary curiosity in reinforcement learning exploration. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.
- [45] 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.

発表文献

本研究に関する発表文献は以下の通りである。

- 加納由希夫, 鶴岡 慶雅. 内部報酬を自動生成する強化学習による一人用 RPG の自動攻略. ゲームプログラミングワークショップ 2017 論文集, pp. 219-225, 2017.
- 加納由希夫, 鶴岡 慶雅. 内部報酬と Hybrid Reward Architecture を用いたローグライクゲームの強化学習. ゲームプログラミングワークショップ 2018 論文集, pp. 64-71, 2018.

謝辞

本研究を進めるに当たり、様々な方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、研究内容や研究生活に関して様々な面でご指導を賜り、3年間もの間大変お世話になりました。これまでの研究生活を振り返ると、私自身の研究の進捗はかなり遅めで、なかなか結果が出せず苦しむことも多かったですが、先生はいつも真剣に考えてくださり、私の研究の内容に関するアドバイスや、問題の解決に繋がる適切な関連論文などを提示してくださいました。また、就職活動がうまくいかずに困っていた時には相談に乗っていただいたこともあり、当時は非常にご迷惑をおかけしましたが、先生のおかげで就職活動を続けることができ、結果的には就職先を見つけ、研究活動を再開することができました。ここまで研究を続けられ、この論文を出すことができたのは先生のおかげです。本当にありがとうございました。

学部4年の頃から研究室の同期である安井豪君には、様々な面で大変お世話になりました。特に、より有意義に研究生活を行う上で欠かせなかった、研究室の計算機の管理や新しいシステムの導入など、対応の難しい件に関して、いつも多大に協力して頂いたことにはとても感謝しております。ゲームAIと自然言語処理とで互いの研究分野は異なりましたが、互いの研究分野に関する知識の共有や、強化学習に関する議論などを通して、私自身も知見を広げることができました。研究に関係のない雑談にもいつも付き合ってくれました。楽しい研究生活を遅れたのは安井君のおかげです。ありがとうございました。

また、博士課程の先輩であった亀甲博貴さんと水上直紀さん、修士課程の先輩であった河村圭悟さんと水谷陽太さんには、研究対象がゲームAIという分野で共通していたこともあり、研究資源の使い方やこの分野に関する基礎知識など様々なことを教えていただいただけでなく、この研究分野やプログラミング実装に関する質問などにいつでも答えていただき、大変助かりました。先輩たちの協力があったからこそ、自分の研究の質を上げていくことができたのだと感じております。本当にありがとうございました。

最後に、今までお世話になった研究室のメンバーたちにも、ミーティングでアドバイスをいただいたり、研究や趣味に関する様々な雑談に付き合っていたり、研究室内の雑務を手伝っていただいたりなど、様々な面で非常にお世話になりました。深く感謝いたしております。