修　　士　　論　　文

強化学習に基づく汎用ゲームプレイヤのための自動特徴生成とモデル学習

Automatic Feature Generation and Model Learning for General Game Players Based on Reinforcement Learning

指導教員　　伊庭 斉志 教授

東京大学情報理工学系研究科
電子情報学専攻

氏　名　　48-136435 藤田 康博

提 出 日　　平成 27 年 2 月 5 日

概 要

ゲームは人間に楽しみを提供するだけでなく，その知的能力を競うという側面から，人工知能のテストベッドとして研究されている．伝統的なゲーム AI 研究では特定のゲームを上手くプレイすることに重きが置かれてきたが，より汎用的なゲーム AI を目指す試みが近年注目を集めている．General Game Playing（GGP）と呼ばれる試みでは，宣言的に記述されたゲームのルールを実行時に受け取り，それのみを頼りに未知のゲームをうまくプレイすることができるような AI の実現が目標とされ，AI の性能を競う国際コンペティションが開催されている．Arcade Learning Environment（ALE）というフレームワークでは，AI はゲームのルールすら知らないまま，ゲーム画面の画像とスコアの増減のみを観測するだけでそのゲームを上手くプレイできるよう学習することが求められる．いずれの枠組みにおいても，AI はゲームに関する事前知識を利用することができないため，自らゲームを分析できる能力を持たなければならない．そのためには AI がゲームを繰り返しプレイし，試行錯誤する中で学習を進める強化学習によるアプローチが考えられる．しかし強化学習アルゴリズムの性能は特徴設計に大きく依存し，これは未知のゲームに対しては困難な課題である．特に GGP では学習に使用できる時間が厳しく制限されるため，汎用的でしかも高速な特徴設計技術が求められている．本論文では，近年提案された iFDD$^+$ という特徴拡張技術を用いることで，GGP における価値関数の学習の際に自動的に組み合わせ特徴を生成することを提案する．生成された特徴を用いた価値関数を UCT，Temporal-Difference Search 及び $\alpha - \beta$ 探索と組み合わせて複数のゲームにおいて評価した結果，iFDD$^+$ による特徴生成は一部のゲームにおいてプレイヤの性能を向上させることが示された．また，ゲームのルールすら手に入らない ALE においては先読みを行うことが難しく，Deep Q-Networks（DQN）という先読みを行わない深層ニューラルネットワークを利用した手法が成功を収めている．本論文では，訓練済みの DQN を利用して環境のモデルを学習することを提案し，それを可能にするための様々な工夫を採り上げる．実験により，学習したモデルを利用して浅い先読みを行うことによって，ゲームによっては DQN で直接ゲームをプレイした場合を上回るスコアを達成できることが示された．

目 次

図 目 次

アルゴリズム，擬似コード

第1章　序論

1.1　背景

チェスや将棋，囲碁といったゲームは楽しみを提供するだけでなく，知的活動と競争を組み合わせることによって，知的能力を測り，向上させることを可能にする．これは人間だけでなくコンピュータがゲームをプレイする場合にも当てはまり，対戦によって知能を測ることで，ゲームは人工知能のテストベッドとしての役割を果たす．

これまでのゲーム AI 研究では主に特定のゲームを人間のプロのようにうまくプレイできるプログラムの開発に重きが置かれてきた．しかし，そのようなプログラムはゲーム依存の知識に大きく依存するとともに，そこで使用される技術もそのゲームに特化されたものが使用されるため，その用途も限られたものとなる．さらに，ゲームの分析やアルゴリズムの設計は事前に人間により行われるため，プログラム自身が解かなければならない問題は多くない．これらの理由から，AI 研究において，特定のゲームのみをプレイするプログラムを開発することの意義は限定的であるという指摘がある [5].

そのような問題意識を元に，より汎用的なゲーム AI を目指すためのフレームワークがこれまでに提案されてきた．代表的なものに General Game Playing（GGP） [5] があり，これは未知のゲームを含む幅広いゲームをうまくプレイできるプログラムを実現することを目指す試みである．専用のゲーム記述言語により記述されたゲームルールがプログラムに実行時に与えられるのみで，それ以外のゲームに関する事前知識は全く手に入らないため，プログラムはそれをゲームをプレイする中で自ら獲得する必要がある．GGP のようなアイディアは 1968 年の研究 [6] にまで遡ることができるが，2005 年の国際コンペティションの開催をきっかけに近年注目を集め，盛んに研究されている．

GGP では専用のゲーム記述言語により記述可能なあらゆるゲームが対象とすることができるが，論理プログラミングによるルールの記述や処理との相性の良いパズルゲームやターン制のボードゲームがその主な対象となっている．一方，より最近提案された Arcade Learning Environment（ALE） [7] というフレームワークは，家庭用ビデオゲーム機である Atari 2600 のエミュレータを利用することにより多数のビデオゲームに対して共通のインタフェースを与え，異なるビデオゲームに対する AI の汎用性の評価を可能にしている．ALE は，GGP と同様に特定のゲームに関する事前知識が手に入らないという点だけでなく，ゲームのルールそのものも AI にとっては明らかではなく，ゲーム画面の画像とスコアの増減の情報だけからプレイングを学ぶ必要があるという点で困難な挑戦である．

本論文では，GGP や ALE が目指すところの未知のゲームを自ら分析・習得し上手くプレイできるゲーム AI プログラムを「汎用ゲームプレイヤ」と総称する．ゲームは一般にルールさえ知ってさえいればプレイすることは可能であるが，上手くプレイするためにはそのゲーム固有の性質について知ることが不可欠である．そのため，汎用ゲームプレイヤは人間の手を借りずにゲームを分析し，その性質について一人で学ぶことができなければならない．その方法としては，プログラムがゲームを繰り返しプレイし，その中で試行錯誤を繰り返しながらそのゲームについて徐々に学習していくというアプローチが考えられる．これを実現するのが機械学習の一分野である強化学習と呼ばれる技術である．強化学習によりゲーム中の状態や行動の価値を学習することを通じて，AI がゲームの性質について理解を深めることが可能になる．

しかし，未知のゲームに対して強化学習アルゴリズムを利用するための障害は多く，その 1 つとして特徴設計の困難さを挙げることができる．特徴設計は学習アルゴリズムの性能を，ひいてはプログラムの性能を大きく左右するものの，効果的な特徴設計のためには通常は問題に関する知識が不可欠であり，未知のゲームに対して効果的な特徴設計を行うことは難しい．特に，GGP では通常はゲームのルールをを教えられてから実際にプレイを開始するまでの時間は短く（多くの場合，数分程度）制限され，特徴設計や学習そのものをより困難にしている．

また，強いゲーム AI を作るためには，一般に，機械学習による状態や行動の価値の学習だけでは不十分であり，未来の状態を予測する先読みが必要であることが知られている．実際に，近年大きく性能の向上している囲碁や将棋の AI ではミニマックス探索やモンテカルロ木探索といったゲーム木探索アルゴリズムによる先読みは不可欠なものとなっており，GGP においてもゲーム木探索アルゴリズムの重要性は高い．しかし，ゲームのルールが手に入らない ALE においては，正確な先読みを行うことができず，そのような探索アルゴリズムを利用することは難しい．

1.2 本研究の目的と貢献

本研究では，GGP と ALE という汎用ゲームプレイヤの実現のための 2 つのフレームワークそれぞれについて，上に述べた対応する 2 つの課題を解決するための提案とその評価を行う．

まず，GGP において，未知のゲームに対する特徴設計という GGP における困難な課題を解決し，ゲーム固有の知識を学習し利用する強いプレイヤを実現するための手法として，iFDD$^+$ と呼ばれるアルゴリズムを用いた組み合わせ特徴の自動生成とそれを利用した価値関数の学習を提案する．学習した価値関数をゲーム木探索に用いることにより，ゲーム固有の知識を反映させた探索が可能になる．ゲーム木探索アルゴリズムとしては，一般的な $\alpha-\beta$ 探索，UCT に加え，UCT と同じくサンプリングベースの探索アルゴリズムとして位置づけられる Temporal-Difference Search を考え，実際のプレイヤの性能を評価する．組み合わせる元となる初期特徴としても状態と行動の相対的関係を用いたものを新たに提案する．iFDD$^+$ の考えうるバリエーションに関しても検討を行う．

次に，ALE において，先読みを可能にするような環境のモデルを学習により獲得するために，良い性能が知られている Deep Q-Networks（DQN）と呼ばれる深層ニューラルネットワークによる強化学習手法の成果を活かすモデル学習手法を提案する．モデルはニューラルネットワークを用いて近似され，訓練済みの DQN による行動価値の評価と共に用いやすい形で学習され，モデルによる先読みと DQN による行動評価を組み合わせたプレイングを可能にする．モデルの学習方法に関しても，複数ステップの先読みを可能にするための訓練法など様々な提案を行う．学習したモデルによる先読みと DQN の組み合わせ方のバリエーションについても検討する．

1.3 本論文の構成

本論文では，まず 2 章で強化学習とゲーム木探索という 2 つの分野にわたる関連研究を紹介する．これらは GGP，ALE いずれのタスクにおいても重要な役割を果たす．3 章ではもっぱら GGP について扱い，GGP の解説とともに先行研究を紹介し，提案手法による自動特徴生成に関して述べる．4 章では ALE を扱い，同様に解説と先行研究の紹介を行った後に提案手法によるモデル学習について述べる．

第2章　関連研究

2.1　強化学習

機械学習と呼ばれる研究分野の中で，エージェントが環境の中でどう振る舞えばよいかを学習することを可能にするようなアルゴリズムの一般的な分類を特に強化学習と呼ぶ．強化学習では，エージェントが環境との相互作用の中で報酬信号を最大化するためにどの行動を選択すべきかを学習する．教師あり学習のように学習者がどの行動を選択すべきかは教えられず，どの行動がより高い報酬に結びつくかを試行錯誤的な探索により見つける必要がある．

2.1.1　マルコフ決定過程

強化学習における環境はマルコフ決定過程（Markov Decision Process，MDP）として定式化される．マルコフ決定過程は次の 4 つの要素を持つタプル $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ により定義される．

- 可能な状態の集合 $\mathcal{S}$
- 可能な行動の集合 $\mathcal{A}$
- 遷移確率（状態 s で行動 a を選択した場合に状態 s' に遷移する確率）
  $\mathcal{P}^a_{ss'} = Prob\{s_{t+1} = s' | s_t = s, a_t = a\}$
- 期待報酬（状態 s で行動 a を選択し状態 s' に遷移する場合に獲得する報酬の期待値）
  $\mathcal{R}^a_{ss'} = \mathbb{E}\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$

環境によっては状態によって可能な行動が異なる場合があり，その場合は状態 s で可能な行動を集合を特に $\mathcal{A}(s)$ と表記する．

エージェントが現在の状態に応じてどのように行動を選択するかを表すものとして状態集合 $\mathcal{S}$ から行動集合 $\mathcal{A}$ への写像である方策 $\pi : \mathcal{S} \mapsto \mathcal{A}$ を考える．方策は決定的な関数として定義されることもあれば，行動の確率分布として定義されることもある．

方策 π が定まればそれに基づいて将来の期待収益が定まる．状態 s において行動 a を取り，その後に方策 π に従った場合の期待収益は

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\{R_t | s_t = s, a_t = a\} \tag{2.1}$$

$$= \mathbb{E}_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \tag{2.2}$$

と表すことができる．$Q^{\pi}(s,a)$ を方策 π に対する行動価値関数と呼ぶ．同様に状態 s から方策 π に従った場合の期待収益 $V^{\pi}(s)$ を π に対する状態価値関数と呼ぶ．

期待収益を最大化する方策に基づいた場合の行動価値関数は

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.3}$$

と書くことができ，これを最適行動価値関数と呼ぶ．同様に

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \tag{2.4}$$

を最適状態価値関数と呼ぶ．

2.1.2 一般化方策反復

方策 π から価値関数 V^{π}（または Q^{π}）を求めることを方策評価と呼ぶ．そうして得られた価値関数を元に，その価値関数の値を最大化するように行動を選ぶ新たな方策

$$\pi'(s) = \arg\max_{a} Q^{\pi}(s,a) \tag{2.5}$$

$$= \arg\max_{a} \mathbb{E}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \| s_t = s, a_t = a\} \tag{2.6}$$

$$= \arg\max_{a} \sum_{s'} \mathcal{P}^{a}_{ss'}[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')] \tag{2.7}$$

を考えることができる．π' を greedy 方策と呼ぶ．greedy 方策は元の方策よりも期待収益が改善される，すなわち $V^{\pi'}(s) \geq V^{\pi}(s)$ が成り立つことが方策改善定理により示される．このように価値関数を元に方策を改善する操作を方策改善と呼ぶ．

方策評価と方策改善を繰り返すと

$$\pi_0 \xrightarrow{\text{評価}} V_0^{\pi} \xrightarrow{\text{改善}} \pi_1 \xrightarrow{\text{評価}} V_1^{\pi} \xrightarrow{\text{改善}} \cdots \xrightarrow{\text{改善}} \pi^{*} \xrightarrow{\text{評価}} V^{*} \tag{2.8}$$

のような系列を得ることができ，これは有限マルコフ決定過程では最適価値関数に収束する．このように方策評価と方策改善を繰り返して最適方策を求めることを方策反復と呼ぶ．

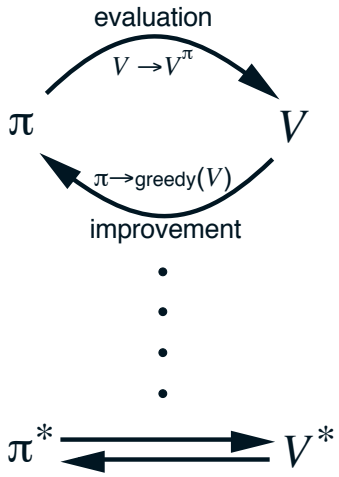


図 2.1: 一般化方策反復により価値関数と方策が最適なものに収束する過程 [1]

完全な方策評価を行うには状態集合 $\mathcal{S}$ 全体に対して計算を行う必要があるが，これは現実的でないことが多い．実際には方策評価と方策反復は必ずしも一方が終了して初めて他方を開始するというように交互に実施する必要はなく，より細かい粒度で行ったとしても問題ない．このように一般化された方策反復を一般化方策反復と呼び，その過程は図 2.1 のようになる．ほとんどすべての強化学習手法は一般化方策反復として説明することができる．

2.1.3 価値関数の近似

最も基本的な形の強化学習問題では，$V(s)$ や $Q(s,a)$ の値は各状態ごと，各状態・行動ペアごとに個別に表形式で保持・更新されるが，状態空間や行動空間の広い問題ではこの方法は取ることが難しく，また状態空間が有限でない場合には不可能である．そこで，状態や状態・行動ペアをまず特徴ベクトルという数値ベクトルにより表現した上で，その特徴ベクトルを価値へ写像する関数を関数近似によって得るという方法が採られる．

状態を D 次元の特徴ベクトルで表す場合，状態空間 $\mathcal{S}$ から特徴空間 $\Phi \subseteq \mathbb{R}^D$ への写像である特徴抽出関数 $\phi : \mathcal{S} \mapsto \Phi$ を考えると，状態 s に対応する特徴ベクトルを $\phi(s)$ と書き表すことができる．状態・行動ペアに関しても同様な特徴抽出関数 $\phi : \mathcal{S} \times \mathcal{A} \mapsto \Phi$ を考えることができる．

あるパラメータベクトル θ を考え，特徴ベクトル $\phi(s)$ と θ の内積として価値関数を

$$V(s) = \phi(s) \cdot \theta \tag{2.9}$$

$$Q(s,a) = \phi(s,a) \cdot \theta \tag{2.10}$$

のように定義するのが線形近似であり，θ の値を更新することで $V(s)$ を目標値に近づけることができる．線形近似の他にも，ニューラルネットワーク等，様々な教師あり学習の技術を価値関数の近似のために使用することができる．

2.1.4 TD(λ)

ある時間ステップにおける価値関数の値と，その後の別の時間ステップでの価値関数の値の差を利用して価値関数を学習する強化学習アルゴリズムを一般に TD（Temporal Difference）学習と呼ぶ．

TD(0) と呼ばれるアルゴリズムは TD 学習の一種であり，方策 π から状態価値関数 V^π を求める方策評価を行う．状態 s_t において π に従い行動を選び，報酬 r_t を受け取り，次の状態が s_{t+1} であった場合に，TD 誤差 δV_t を

$$\delta V_t = r_t + V(s_{t+1}) - V(s_t) \tag{2.11}$$

のように計算し，これを最小化するように $V(s_t)$ の値を

$$\Delta V(s_t) = \alpha \delta V_t \tag{2.12}$$

だけ更新する．$\alpha \in [0, 1]$ は学習率と呼ばれるパラメータで，更新量を制御する．

TD(0) では s_t の直後の状態 s_{t+1} に対する価値関数の値を利用して価値関数の更新を行っているが，より後のタイムステップの状態に対する値まで利用するアルゴリズムを TD(λ) と呼ぶ．パラメータ λ はどれくらい後の TD 誤差対してその責任を負うかという期間を決めるもので，$\lambda = 0$ の場合が TD(0) に当たる．TD(λ) では適格度トレースと呼ばれる値 $\mathbf{e}$ を保持し，これに各タイムステップで λ を掛けることにより先の状態の責任は指数関数的に減衰していくことになる．

V^π をパラメータ θ を用いて近似し，最急降下法を利用して TD 誤差を最小化する場合の TD(λ) アルゴリズムの擬似コードを Algorithm 2.1 に示す．

2.1.5 Sarsa(λ)

前節で述べた TD(0) 及び TD(λ) は方策 π から価値関数 V^π を求める方策評価を行うためのアルゴリズムである．これに一般化方策反復の考え方を適用し，方策改善を道入することによって，TD 学習をにより最適方策を求めることが可能になる．

方策改善のために状態価値関数 $V(s)$ から greedy 方策を求めるには，式（2.7）のように遷移確率 $\mathcal{P}^a_{ss'}$ 及び期待報酬 $\mathcal{R}^a_{ss'}$ が既知でなければならないが，行動価値関数 $Q(s, a)$ から greedy 方策を求める際にはそれらは必要ない．

Procedure 2.1 オンライン最急降下法 TD(λ)

Input: $\pi, \lambda, \gamma, \alpha$

Output: V^π

Initialize θ arbitrarily
for all episode **do**
  $s \leftarrow s_0$
  $\mathbf{e} \leftarrow \mathbf{0}$
  while s is not terminal **do**
    $a \leftarrow \pi(s)$
    Execute a, observe reward r and next state s'
    $\delta V \leftarrow r + \gamma V(s') - V(s)$
    $\mathbf{e} \leftarrow \lambda\gamma\mathbf{e} + \nabla_\theta V(s)$
    $\theta \leftarrow \theta + \alpha\delta V\mathbf{e}$
    $s \leftarrow s'$
  end while
end for

そのようなアルゴリズムの 1 つに Sarsa [8] がある．Sarsa は State-Action-Reward-State-Action の略で，TD(λ) と ϵ-greedy 方策による方策改善を組み合わせたものである．ϵ-greedy 方策とは

$$\pi(s) = \begin{cases} \arg\max_a Q(s,a) & (\text{確率 } 1-\epsilon) \\ \text{random action} & (\text{確率} \epsilon) \end{cases} \tag{2.13}$$

のように確率 ϵ で完全にランダムな行動を選択する greedy 方策である．

方策改善のために状態価値関数 $V(s)$ から greedy 方策あるいは ϵ-greedy 方策を求めるには，式 (2.7) のように遷移確率 $\mathcal{P}^a_{ss'}$ 及び期待報酬 $\mathcal{R}^a_{ss'}$ が既知でなければならないが，行動価値関数 $Q(s,a)$ の場合にそれらは必要ない．そのため，行動価値関数 $Q(s,a)$ の形での方策評価が行われることが多い．

Sarsa では，

- 状態 s_t において方策 π に従い行動を a_t を選ぶ
- 報酬 r_t と次の状態 s_{t+1} を観測する
- 状態 s_{t+1} で方策 π に従い行動 a_{t+1} を選ぶ

という状態遷移から得られるタプル $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ において，TD(λ) の場合と同様に TD 誤差

$$\delta Q_t = r_t + Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.14}$$

を計算し，これを最小化するように $Q(s,a)$ を更新する．$Q(s,a)$ の近似のために線形近似を使用した Sarsa の擬似コードを Algorithm 2.2 に示す．

Procedure 2.2 線形 Sarsa(λ)

Input: λ, γ, α

Output: Q^*

$\theta \leftarrow 0$
loop
  $s \leftarrow s_0$
  $\mathbf{e} \leftarrow \mathbf{0}$
  $a \leftarrow \epsilon$-greedy action from state s
  while s is not terminal **do**
    Execute a, observe reward r and next state s'
    $a' \leftarrow \epsilon$-greedy action from state s'
    $\delta Q \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $\delta\theta \leftarrow \theta + \alpha \delta Q e$
    $e \leftarrow \lambda\gamma\mathbf{e} + \phi(s, a)$
    $s \leftarrow s', a \leftarrow a'$
  end while
end loop

2.1.6 Q-learning

行動価値関数を学習するための代表的な強化学習アルゴリズムとしては，Sarsa(λ) の他に Q-learning [9] と呼ばれるものがある．Q-learning ではタプル (s_t, a_t, r_t, s_{t+1}) から行動価値関数 $Q(s, a)$ の TD 誤差を

$$\delta Q = r + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \tag{2.15}$$

と計算し，これを最小化するように $Q(s, a)$ を更新する．Sarsa の TD 誤差式（2.14）と異なり，次のステップで実際に選択した行動 a_{t+1} を用いるのではなく，s_{t+1} で greedy に行動を選択したとみなして TD 誤差を計算する．

この TD 誤差の計算方法の違いにより，Sarsa と Q-learning の間には重要な違いが生まれる．Sarsa では TD 誤差の最小化により現在の方策 π に基づく行動価値関数 Q^π を近似しようとするが，Q-learning では Q^π ではなく最適行動価値関数である Q^* を直接近似しようとする．この違いに基づき，Q-learning は方策オフ型（off-policy）であり，一方 Sarsa は方策オン型（on-policy）のアルゴリズムに分類される．

2.1.7 特徴拡張

2.1.3 項で述べたように，価値関数の近似は特徴関数 ϕ を利用して行われる．この際に，どのような特徴関数を採用するか，すなわち状態をどのような特徴ベクトルとして表現するかが問題となる．

例として，Tic-Tac-Toe（三目並べ）の盤面の状態を特徴として表現する場合について考える．Tic-Tac-Toe の盤面には $3 \times 3 = 9$ のマスがあり，それぞれのマスは

- X がマークされている
- O がマークされている
- 何もマークされていない

の 3 状態のいずれかである．従って，マス毎に 3 つの変数（x_0, x_1, x_2）を用意し，それぞれ

- x_0：X がマークされていれば 1，そうでなければ 0
- x_1：O がマークされていれば 1，そうでなければ 0
- x_2：何もマークされていなければ 1，そうでなければ 0

のようにその値を定義すれば，$9 \times 3 = 27$ の変数，すなわち 27 次元の特徴ベクトルによって盤面の状態を完全に表現することができる．このように 0 または 1 の 2 つの値のみを取る特徴ベクトルの要素をバイナリ特徴と呼ぶ．バイナリ特徴は Tic-Tac-Toe のような離散的な状態空間を持つ問題にとどまらず，連続的な状態空間を持つ問題に対しても状態の離散化を通じて使用される．

この Tic-Tac-Toe の特徴ベクトルは，線形関数近似と組み合わされた場合に問題を生じさせる．特徴ベクトル $\phi(s)$ の i 番目の要素を $\phi_i(s)$ と表記すると，線形関数近似では

$$V(s) = \phi_0(s)\theta_0 + \phi_1(s)\theta_1 + \cdots \tag{2.16}$$

のように特徴ベクトルの各要素に対し，それに対応するパラメータが掛けられ，それらの和によって価値関数が近似されるため，特徴ベクトルの各要素の寄与は独立であり，特徴間の依存関係を反映させることができない．Tic-Tac-Toe は直線上に 3 つ同じマークを書くことが勝利条件であり，「(0, 0) のマスと（1，1）のマスに X がマークされており，(3，3）のマスには何もマークされていない」という状態ならば，もし X の手番であれば X にとっては勝利の一歩手前であるし，もし O の手番であればそれを防ぐ必要がある，というように複数のマスの状態を同時にすることが形勢判断のために不可欠である．したがって，各マスの状態を独立に評価するモデルでは状態の価値を正確に近似することができない．

このように特徴ベクトルの表現力が不足している場合に，自動的に新たな特徴を追加する（特徴拡張）ことによって特徴ベクトルの表現力を増し，価値関数の学習がよりよく行えるようにするためのアルゴリズムが近年提案されている．iFDD（incremental Feature Dependency Discovery）[10] は，状態空間のうち価値関数の誤差が残る部分空間に新たな特徴を徐々に追加していくアルゴリズムであり，これまでに説明したような TD 誤差に基づく強化学習アルゴリズムと組み合わせて用いられる．

Procedure 2.3 iFDD$^+$ Discover [2]

Input: $\phi(s), \delta, \xi, F, \psi, N$

Output: F

```
for all (g, h) ∈ {(i, j)|φ_i(s)φ_j(s) = 1} do
    f ← g ∧ h
    if f ∉ F then
        ψ_f ← ψ_f + δ
        N_f ← N_f + 1
        if |ψ_f| / √N_f > ξ then
            F ← F ∪ f
        end if
    end if
end for
```

iFDD では，$\phi_f : \mathcal{S} \to \{0, 1\}, \forall f \in \mathbf{F}$ を満たすようなバイナリ特徴の初期集合 $\mathbf{F}$ を元にし，それらを論理積で組み合わせることによって新たな順次特徴を追加していく．

強化学習アルゴリズムによって TD 誤差が観測される度に，追加する候補となる組み合わせ $f : g \wedge h$ ごとに，TD 誤差との関連度 $\eta_t(f)$ という値を計算し，これがあるしきい値 ξ を超えた際に初めてその組み合わせを新たな特徴として追加する．この関連度 $\eta_t(f)$ は

$$\eta_t(f) = \sum_{i \in \{1, \cdots, t\}, \phi_f(s_i) = 1} |\delta_i| \tag{2.17}$$

と定義され，これは特徴 g 及び h が両方共にアクティブな状態，すなわち $\phi_g(s) = \phi_h(s) = 1$ の状態における TD 誤差の絶対値を過去に渡って足し合わせたものであり，これが大きいほどその状態における誤差が大きく，新たな特徴を追加する必要性が高いと言える．

後に提案された iFDD$^+$[11] ではよりよい関連度の基準として

$$\eta_t^+(f) = \frac{\sum_{i \in \{1, \cdots, t\}, \phi_f(s_i) = 1} \delta_i}{\sqrt{\sum_{i \in \{1, \cdots, t\}, \phi_f(s_i) = 1} 1}} \tag{2.18}$$

が示されている．iFDD$^+$ が関連度を計算し，追加する特徴を決定する際の擬似コードを Algorithm 2.3 に示す．過去の誤差は全て記録しておく必要はなく，特徴ごとにこれまでの和のみ保持しておけば十分である．

一旦組み合わせ特徴 $f : g \wedge h$ が追加されると，$\phi_f(s) = 1$ であるような状態 s では $\phi_g(s) = \phi_h(s) = 0$ として扱い，その結果アクティブな特徴の数が減少し，よりスパースな表現が得られることになる．新たな特徴の重みは組み合わせる特徴の重みの和 $\theta_f = \theta_g + \theta_h$ として初期化されるため，新たな特徴の追加のみによって価値関数の出力が変化することはない．

状態 s_t で TD 誤差を観測した場合の $\eta_t(f)$ の計算は s_t でアクティブな特徴の組み合わせに対してのみ行えばよく，また $\phi(s_t)$ を求める際も s_t でアクティブな初期特徴からなる組み合わせ特徴のみについて調べればよいため，時間ステップあたりの計算コストはアクティブな初期特徴の数によって抑えられる．その結果，たとえ状態空間が高次元であってもアクティブな特徴さえ少なければ高速に学習を行うことができる．さらに時間ステップあたりの収束の速さも経験的に示されている [10].

iFDD 及び iFDD$^+$ は Sarsa や LSTD，Q-Learning，Greedy-GQ などの強化学習アルゴリズムと組み合わせることによって，Inverted Pendulum や Blocks World といった古典的なものを含む複数の強化学習タスクにおいて，既存の特徴拡張アルゴリズムに比べ高い性能を示すことが確認されている [10, 11, 2].

2.2 ゲーム木探索

ゲームにおいて起こりうる状態遷移を，状態をノード，行動をエッジとした木構造で表現したものをゲーム木と呼ぶ．ゲームでは，より良い状態に到達できるような行動を選択するために，このゲーム木の上で探索を行う．以下では，ゲーム木の探索のためのアルゴリズムのうちに関して説明する．

2.2.1 モンテカルロ木探索

シミュレーションなどを乱数を用いて実行する手法を総称してモンテカルロ法と呼ぶ．ゲームにモンテカルロ法を用いることは手の価値を近似するために有効であることが実証されている [12].

その最も単純な適用方法としては，現在の状態において選択することが行動（合法手）のそれぞれに対し，その手を選んだ後の状態からゲームの終わりまでシミュレーションを繰り返すことにより勝率（または平均報酬）を計算し，それを手の価値だと考え，最も高い価値の手を選ぶという方法がある（単純モンテカルロ法）.

モンテカルロ木探索 [13] はランダムシミュレーションを用いたゲーム木探索アルゴリズムであり，シミュレーションの結果を元にノードを評価することでゲーム木の上で最良優先探索を行う．現在の状態をルートノードとし，シミュレーションを繰り返しながら少しづつメモリ上にゲーム木の部分木（これを探索木と呼ぶ）を構築することにより先読みを行う.

モンテカルロ木探索のアルゴリズムは以下の 4 つのステップの繰り返しから成る.

選択 ルートノードを出発点とし，一定の基準で子ノード（完全なゲーム木における子ノードであり，現時点で探索木に加えられているとは限らない）を選択しつつ探索木を下る.

展開 探索木に含まれていないノードに達した時点で木を下るのをやめ，そのノードを探索木に加える.

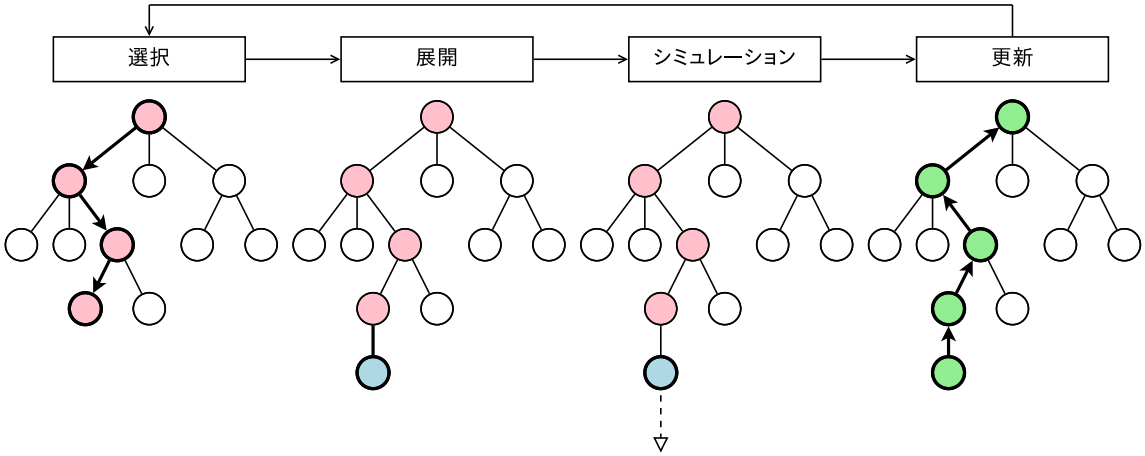


図 2.2: モンテカルロ木探索の 1 回のイテレーション．選択，展開，シミュレーション，更新の 4 ステップにより構成される．

シミュレーション 探索木に加えたノードからゲームの終わりまで 1 度だけシミュレーションを行い，報酬を得る．

更新 得られた報酬をもとに，各ノードが保持する平均報酬を更新しながら探索木を逆にたどる．

探索木のノードが保持する平均報酬は選択ステップにおける子ノードの選択に使用されるとともに，最終的に現在の状態ににおける最も良い手を選択するために使用される．

モンテカルロ木探索は 2006 年に囲碁 AI に導入されることで飛躍的な棋力向上を達成し，2008 年には 9×9 の囲碁でプロ棋士に勝利するまでに至った [14].

モンテカルロ木探索のバリエーションのうち最もよく使用されるアルゴリズムが Upper Confidence bounds applied to Trees（UCT）[15] である．UCT では選択ステップにおける子ノードの選択を多腕バンディット問題とみなし，UCB1 アルゴリズム [16] を用いて次のように定義される UCB 値が最大となるような子ノード j を選択する．

$$UCB1 = \overline{X_j} + 2C_p\sqrt{\frac{2\ln n}{n_j}} \tag{2.19}$$

ただし n は親ノードの訪問回数，n_j は子ノード j の訪問回数，$\overline{X_j}$ は子ノード j が保持する平均報酬，$C_p > 0$ は定数である．$n_j = 0$ の場合 $UCT = \infty$ となるため，全ての子ノードは少なくとも 1 度は訪問されることが保証される．

Procedure 2.4 Linear Temporal-Difference Search [3]

$\theta \leftarrow 0$
procedure SEARCH(s_0)
  while time available **do**
    $e \leftarrow 0$
    $s \leftarrow s_0$
    $a \leftarrow \epsilon$-greedy$(s; Q)$
    while s is not terminal **do**
      $s' \sim \mathcal{P}^a_{ss'}$
      $r \leftarrow \mathcal{R}^a_{ss'}$
      $a' \leftarrow \epsilon$-greedy$(s'; Q)$
      $\delta Q \leftarrow r + Q(s', a') - Q(s, a)$
      $\delta\theta \leftarrow \theta + \alpha\delta Q e$
      $e \leftarrow \lambda e + \phi(s, a)$
      $s \leftarrow s', a \leftarrow a'$
    end while
  end while
  return $\arg\max_a Q(s_0, a)$
end procedure

2.2.2 Temporal-Difference Search

Temporal-Difference Search（TD Search）[3] は TD 学習とシミュレーションによる探索を組み合わせた探索手法であり，モンテカルロ木探索のように同様に現在の状態からのシミュレーションを繰り返すことにより探索を進める．しかし，モンテカルロ木探索のように探索木は構築せず，現在の状態からのシミュレーション結果から行動価値関数 $Q(s, a)$ を持ち時間中に学習し，行動を選択する際は学習された行動価値が最も高い行動を選ぶ．この方法により，ゲーム全体ではなく現在の状態から到達可能なより小さい部分空間に対して学習が行えるとともに，異なる状態間でもその特徴が共有されていれば知識を一般化することができる．

TD Search ではプレイ中に価値関数の学習を行うため，事前に価値関数を学習しておく必要はなく，UCT のように事前知識無しで利用することができる．囲碁において RAVE[17] や事前知識を用いない UCT と比較して同じシミュレーション数でより良い性能が得られることが示されている．

第3章　General Game Playingのための組み合わせ特徴の自動生成

3.1　General Game Playing

3.1.1　International GGP Competition

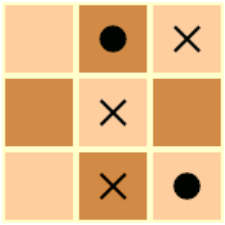
2005年から毎年International GGP Competition（IGGPC）と呼ばれるコンペティションがAAAIまたはIJCAIにおいて開催されている．2005年から使用されているコンペティションのフォーマットがそれ以降のGGP研究において踏襲されている．

2005年から2014年までのIGGPCの優勝プログラムとその使用している探索アルゴリズムを表3.1に示す．特筆すべきは，2007年のCadiaplayerの優勝以降，全ての優勝プログラムがモンテカルロ木探索のようなランダムシミュレーションを用いていることである．2011年及び2013年の優勝プログラムであるTurboTurtleの探索アルゴリズムは公開されていないが，シミュレーションを利用しているという点は明らかにされている．

表3.1: 2005年から2014年までのInternational GGP Competitionの優勝プログラムとその探索アルゴリズム

年	優勝プログラム	探索アルゴリズム
2005	Cluneplayer [18]	ミニマックス探索
2006	Fluxplayer [19]	ミニマックス探索
2007	Cadiaplayer [20]	モンテカルロ木探索
2008	Cadiaplayer [20]	モンテカルロ木探索
2009	Ary [21]	モンテカルロ木探索
2010	Ary [21]	モンテカルロ木探索
2011	TurboTurtle	情報なし（シミュレーション利用）
2012	Cadiaplayer [20]	モンテカルロ木探索
2013	TurboTurtle	情報なし（シミュレーション利用）
2014	Sancho	モンテカルロ木探索

図 3.1: Tic-Tac-Toe の盤面の事実タプルの集合による表現



→

```
(cell 1 1 b) (cell 2 1 o) (cell 3 1 x)
(cell 1 2 b) (cell 2 2 x) (cell 3 2 b)
(cell 1 3 b) (cell 2 3 x) (cell 3 3 o)
(control oplayer)
```

3.1.2 Game Description Language

GGP において個々のゲームのルールは Game Description Language（GDL）[22] によって記述される．GDL は Datalog というクエリ言語を元にした言語であり，ゲームルールは表 3.2 にある専用の関係述語を用いて宣言的に記述される．プログラムはこのゲームルールを実行時に受け取ることでルールを把握する．

ゲームの各局面は「事実」を表すタプルの集合によって表現される．Tic-Tac-Toe（○×ゲーム，三目並べ）というゲームの一局面とその表現を図 3.1 に示す．

3.1.3 GGP が扱うゲーム

GGP で扱うことができるゲームの範囲は GDL の記述力により限定される．GDL は高い表現力を持ち，8 パズルのような 1 人ゲームから Chess や Tic-Tac-Toe のような 2 人ゲーム，さらに Chinese

表 3.2: GDL の予約された関係述語

`(role P)`	P はプレイヤである
`(init X)`	初期状態で事実 X が成り立つ
`(next X)`	次状態で事実 X が成り立つ
`(true X)`	現在の状態で事実 X が成り立つ
`(legal P A)`	現在の状態においてプレイヤ P が行動 A を選択できる
`(does P A)`	現在の状態においてプレイヤ P が行動 A 選択する
`(goal P R)`	現在の状態においてプレイヤ P が得る報酬が R である
`terminal`	現在の状態が終端状態である

Checkers のような 3 人以上のゲームやじゃんけんのようにプレイヤーが同時に行動を選択するゲーム等，様々なゲームを表現することができる．しかし GDL による表現上の制約から，ゲームについて以下のような制約が満たされなければならない [22].

- 可能な状態や行動の数が有限でなければならない
- ランダム要素を含まず，決定的でなければならない
- プレイヤにとって隠れた情報が無い
- プレイヤの数がゲームの開始から終了まで変化しない

このような制約から，バックギャモン（ランダム要素を含む）やポーカー（隠れた情報がある）は GDL により記述できず，GGP で扱うことができない．これらの制約を緩和し，非決定的ゲームや不完全情報ゲームを扱えるようにするために GDL の文法を拡張した GDL-II（Game Description Language for Incomplete Information）も提案されている [23].

3.1.4 GGP における学習

これまでの GGP における学習の試みとしては，まず過去のランダムシミュレーションの結果として得られた報酬の平均を記録しておくことで行動に対する評価を学習する History Heuristic が挙げられる．Move-Average Sampling Technique（MAST）[24] では，各行動 a に対しそれを選択した場合の平均報酬 $Q_h(a)$ を記録しておき，確率分布

$$P(a) = \frac{e^{Q_h(a)/\tau}}{\sum e^{Q_h(b)/\tau}} \tag{3.1}$$

に従って行動を選ぶことで良い行動を高い確率で選ぶ．これを発展させるものとして，事実と行動の各ペアについて同様の記録を行う Predicate-Average Sampling Technique（PAST）[25] や，直近の連続した N 個の行動を用いる N-Grams[26] が提案されている．

また，強化学習により価値関数を学習する試みもなされている．[27] では事実を特徴として用い，状態価値を次のように近似する．

$$V(s) = \sigma(\phi(s) \cdot \theta) \tag{3.2}$$

ただし $\phi(s)$ は特徴ベクトル，θ は重みベクトルであり，シグモイド関数 $\sigma(t) = \frac{1}{1+e^{-t}}$ により特徴の重み付け和を $(0, 1)$ の範囲にマッピングしている．この重みベクトル θ を TD(0) によって学習している．

GGP における学習に用いられている主な特徴の例を表 3.3 に示す．タプルで表された事実や行動は GDL の処理により容易に手に入る特徴であり，それぞれ単体で特徴として用いられるだけでなく，事実と行動のペア [25] や直近の連続した行動 [26] のように組み合わせて用いられることもある．ま

た，タプル中のアトムの出現回数を数えたり [28]，特定の位置に特定のアトムが出現するタプルを数える [29] というような，バイナリではない特徴が用いられることもある．

ゲームの状態をどのようにタプルの集合で表現するかはゲームルールの記述者に委ねられるが，ボードゲームではほとんどの場合 1 つのマスを 1 つのタプルで表現している．そのため，タプルを複数組み合わせた特徴を用いることで複数の駒の間の関係を捉えることができる．しかし，機械的にあらゆる組み合わせを特徴として用いるようにすると，組み合わせる数によっては組み合わせ爆発により特徴の数が多くなりすぎ，効率的に学習を行えなくなってしまう．

3.2 提案手法

iFDD$^+$ は時間ステップあたりの計算コストが比較的小さく，また収束までに必要な時間ステップも少ないことから，学習に割くことができる時間が厳しく制限されている GGP に適した特徴拡張手法であると考えられる．そこで本論文では，GGP において価値関数を学習する際に iFDD$^+$ を用いて自動的に組み合わせ特徴を生成することを提案する．

3.2.1 価値関数の利用法

強化学習により未知のゲームに対して価値関数を学習した場合に，その価値関数をどのようにゲーム木探索に活用するかが問題になる．本論文では，以下の 3 つのゲーム木探索アルゴリズムにおいて価値関数の活用を試みる．

UCT UCT は [24] における導入から GGP において広く用いられ，特にコンペティションで成功を収めている．価値関数のようなゲーム固有の知識を組み込むことで性能が向上することが知られている．ランダムシミュレーションにおいて行動を選択する際に行動に重み付けを行うシミュレーション方策として知識を導入することは History Heuristic に見られるように GGP においても成功

表 3.3: GGP で学習に使用される特徴とその例

Feature Types	Examples
アトム [28]	`cell, x, o`
行動 [24]	`(mark 1 1)`
事実 [28, 27]	`(cell 2 2 x)`
一般化された事実 [29]	`(cell ? ? x)`
事実と行動のペア [25]	`(cell 2 2 x)(mark 1 1)`
最後の N 個の行動 [26]	`(mark 1 1)(mark 2 2)(mark 3 3)`

を収めている．そこで，行動価値関数 $Q(s,a)$ を学習し，これをシミュレーション方策として用いることがまず考えられる．

TD Search　著者の知る限りにおいて囲碁以外で用いられたという例は無いものの，TD Search は [3] で良い性能が示されているだけでなく，強化学習が探索の根幹を成しており，特徴拡張により得られる恩恵が大きいと考えられる．

$\alpha-\beta$ 探索　$\alpha-\beta$ 探索は広く知られた探索手法であり，GGP においても状態価値関数 $V(s)$ を学習あるいはゲームルールから近似した上で用いられている．状態価値関数の精度はプレイヤの性能に大きく影響する．コンペティションにおいては現在は UCT が支配的であるものの，これには GGP におけるゲーム固有の知識を得ることの困難さが影響していると考えられ，精度の高い価値関数が得られれば $\alpha-\beta$ 探索により UCT より強い性能を達成することも可能だと予想される．

3.2.2　価値関数の近似

GGP ではゲームの報酬は 0 以上 100 以下の整数を取り，二人ゼロ和ゲームの場合はほとんどの場合勝ち 100, (引き分けが存在する場合は) 引き分け 50，負け 0 として GDL により定義される．これをそのまま特徴ベクトルと重みベクトルの内積で近似することも考えられるが，本論文では報酬を $[0,1]$ の範囲に正規化した上で，$V(s)$ は (3.2) と同様とし，$Q(s,a)$ も同じくシグモイド関数を用いて

$$Q(s,a) = \sigma(\phi(s,a)\cdot\theta) \tag{3.3}$$

とする．

iFDD$^+$ により組み合わせ特徴を生成するためには，組み合わせる対象となる初期特徴が無ければならず，これは 1 または 0 の値を取るバイナリ特徴である必要がある．本論文では，まず GGP で最も容易に手に入る特徴として状態や行動を表す個々のタプルを初期特徴として利用する．Tic-Tac-Toe を例に取れば，状態 s に対するタプル特徴 $\phi(s)_{tuple}$ は

$$\phi_{tuple}(s) = \begin{array}{|c||c|}\hline \texttt{(cell 1 1 b)} & 1 \\ \hline \vdots & \vdots \\ \hline \texttt{(cell 3 3 b)} & 1 \\ \hline \text{その他} & 0 \\ \hline \end{array}$$

のような値を取る．状態 s と行動 a に対するタプル特徴は，これに行動を表すタプルを 1 つだけ加えた

$$\phi_{tuple}(s,a) = \begin{array}{|c||c|}\hline \texttt{(cell 1 1 b)} & 1 \\ \hline \texttt{(cell 1 2 b)} & 1 \\ \hline \vdots & \vdots \\ \hline \texttt{(cell 3 3 b)} & 1 \\ \hline \texttt{(mark 1 1)} & 1 \\ \hline \text{その他} & 0 \\ \hline \end{array}$$

のような値を取るものとする．

状態 s と行動 a に対しては，タプル特徴だけでなく，より表現力の高い特徴として，事実タプルと行動タプルのペアの相対的な関係を捉える特徴（RFA，Relative Fact Action）特徴についても検討する．これは，

$$\phi_{RFA}(s,a) = \begin{array}{|c||c|}\hline \texttt{(cell X Y b)(mark X Y)} & 1 \\ \hline \texttt{(cell X Y b)(mark X Y+1)} & 1 \\ \hline \vdots & \vdots \\ \hline \texttt{(cell X Y b)(mark X+3 Y+3)} & 1 \\ \hline \text{その他} & 0 \\ \hline \end{array}$$

のようなものになり，2 駒の相対的な位置関係が意味を持つようなボードゲーム等で特に有用だと考えられる．これを組み合わせることで 3 駒以上の関係も容易に表現できる．事実と行動の相対的な関係は，ゲームルールにより定義されたアトム間の順序関係と述語同士の引数の対応関係を利用することで，GDL で表された多くのゲームにおいて自動的に読み取ることができる．

3.3 評価

3.3.1 実装

アルゴリズムを実装する GGP エージェントとしては独自に C++により実装したものを用いた．GDL からの推論には Yet Another Prolog を利用した．高速な Prolog 処理系を用いた GDL 推論エンジンは，処理可能な GDL の文法やゲームの種類を制限しないものとしては現在のところ最も高速であることが示されている [30]．

3.3.2 使用するゲーム

どうぶつしょうぎ，Nine Board Tic-Tac-Toe，TTCC4，Connect Four，Breakthrough の 5 つのゲームを評価に用いた．これらは全てターン制の二人有限確定完全情報ゲームである．どうぶつしょう

ぎは独自に実装したものを，他の 4 つはスタンフォード大のゲームレポジトリ（http://gamemaster.stanford.edu/）のものを用いた．これらのゲームに関する詳細な説明は付録 A に記す．

3.3.3 共通のパラメータ

価値関数の学習のためには全て Sarsa(0) を用い，学習率は常に $\alpha = 0.1$ とした．また学習のためのシミュレーションを行う際は常に $\epsilon = 0.5$ の ϵ-Greedy 方策に従い行動を選択した．

3.3.4 UCT と TD Search の評価

UCT と TD Search に関して，まず単純なタプル特徴を初期特徴とした場合の iFDD$^+$ の性能を調べるため，次の 5 つの設定を考え，Plain UCT に対する他の設定の勝率を測定した．

- Plain UCT （価値関数は用いず，シミュレーションで一様にランダムに行動を選ぶ）
- UCT 特徴は $\phi_{tuple}(s,a)$
- UCT 特徴は $\phi_{tuple^2}(s,a)$（$\phi_{tuple}(s,a)$ の特徴の任意の 2 つの組み合わせ全て）
- UCT 特徴は iFDD$^+$，初期特徴は $\phi_{tuple}(s,a)$
- TD Search 特徴は iFDD$^+$，初期特徴は $\phi_{tuple}(s,a)$

ゲーム開始前に学習の時間はおかず，UCT でも TD Search と同様にプレイ中にシミュレーションを利用して $Q(s,a)$ を更新した．各ターンのシミュレーション数を固定し，全ての設定において同数のエピソードから学習が行われるようにした．

iFDD$^+$ のパラメータ ξ は 1 と 2 の場合をそれぞれ評価した．$\xi < 1$ とすると 1 度の TD 誤差の観測だけで $\eta_t^+ > \xi$ となってしまう場合があるため，有用でない特徴を生成しすぎてしまうおそれがある．一方，経験的に $\xi \geq 3$ では，シミュレーションの数が限られるとほとんど組み合わせ特徴を生成せずに終わってしまう場合が見られた．

ターン毎のシミュレーション回数を 100 及び 1000 に固定した場合の勝率の数値をそれぞれ表 3.4 及び表 3.5 に示す．iFDD$^+$ を用いた UCT が比較的良い性能を示しており，iFDD$^+$ により有用な組

表 3.4: Plain UCT に対する勝率（タプル特徴，100 シミュレーション毎ターン）

	UCT				TD Search	
	$\phi_{tuple}(s)$	$\phi_{tuple^2}(s,a)$	iFDD $\xi = 1$	iFDD $\xi = 2$	iFDD $\xi = 1$	iFDD $\xi = 2$
どうぶつしょうぎ	0.47 ± 0.02	0.49 ± 0.02	$\mathbf{0.52} \pm 0.03$	0.49 ± 0.02	0.41 ± 0.07	0.34 ± 0.07
Nine Board Tic-Tac-Toe	0.56 ± 0.02	0.51 ± 0.02	$\mathbf{0.57} \pm 0.03$	0.55 ± 0.02	0.45 ± 0.07	0.42 ± 0.07
TTCC4	0.62 ± 0.02	$\mathbf{0.67} \pm 0.02$	$\mathbf{0.67} \pm 0.02$	0.65 ± 0.02	0.35 ± 0.07	0.3 ± 0.06
Connect Four	0.39 ± 0.02	0.47 ± 0.02	$\mathbf{0.56} \pm 0.03$	0.46 ± 0.02	0.29 ± 0.06	0.22 ± 0.06
Breakthrough	$\mathbf{0.81} \pm 0.02$	0.8 ± 0.02	$\mathbf{0.81} \pm 0.05$	0.8 ± 0.02	0.49 ± 0.07	0.49 ± 0.07

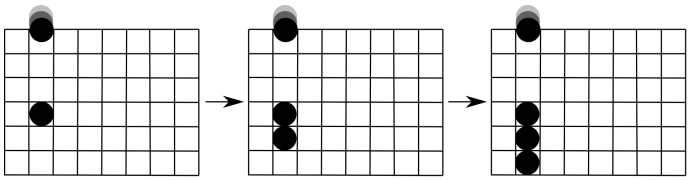


図 3.2: iFDD$^+$ により「縦に 4 つ石を並べる」特徴が生成される過程

み合わせ特徴を生成することができたと考えられる．特に，任意の 2 つのタプルを機械的に組み合わせる特徴 $\phi_{tuple^2}(s,a)$ に比べ総じて性能は高く，iFDD$^+$ により選択的にタプルを組み合わせることが有効であると言える．

ξ の値に関しては，シミュレーション回数が 100 のデータでは総じて $\xi=2$ より $\xi=1$ の方が勝率が高いが，1000 になるとこれは拮抗するようになる．ξ が小さいほうが特徴が追加されやすく，シミュレーション回数が少ないうちはその影響が現れやすいと考えられる．TD Search の勝率は同じ特徴を使っていても総じて UCT に比べ低く，特にシミュレーション回数が増えるにつれて UCT に対して相対的に低い性能を示している．ターン毎のシミュレーション回数が少ないと TD Search が相対的に強く，シミュレーション回数が多いと相対的に UCT が強いという傾向は [3] でも見られたものである．しかしシミュレーション回数が少ない場合でも UCT の方が概ね高い勝率を達成しており，TD Search の有用性を示すことはできなかった．

ここで，実際に iFDD$^+$ により生成された特徴を見てみる．Connect Four においての UCT+iFDD$^+$ ($\xi=2$) が最初の 1000 シミュレーションで生成した特徴が次の 7 つであった．

1. `<(cell 4 3 black),(drop 4)>`
2. `<(cell 2 3 black),(drop 2)>`
3. `<(cell 2 2 black),<(cell 2 3 black),(drop 2)>>`
4. `<(cell 1 3 red),(control red)>`
5. `<(cell 4 2 black),<(cell 4 3 black),(drop 4)>>`
6. `<(cell 1 3 black),(drop 1)>`

表 3.5: Plain UCT に対する勝率（タプル特徴，1000 シミュレーション毎ターン）

	UCT				TD Search	
	$\phi_{tuple}(s)$	$\phi_{tuple^2}(s,a)$	iFDD $\xi=1$	iFDD $\xi=2$	iFDD $\xi=1$	iFDD $\xi=2$
どうぶつしょうぎ	0.44 ± 0.02	0.48 ± 0.03	0.63 ± 0.03	$\mathbf{0.66} \pm 0.03$	0.23 ± 0.06	0.17 ± 0.05
Nine Board Tic-Tac-Toe	0.66 ± 0.02	0.47 ± 0.03	0.7 ± 0.03	$\mathbf{0.7} \pm 0.03$	0.34 ± 0.07	0.42 ± 0.07
TTCC4	0.87 ± 0.01	$\mathbf{0.94} \pm 0.01$	0.92 ± 0.01	0.89 ± 0.01	0.63 ± 0.07	0.43 ± 0.07
Connect Four	0.27 ± 0.02	0.39 ± 0.03	$\mathbf{0.5} \pm 0.02$	0.49 ± 0.02	0.19 ± 0.05	0.09 ± 0.04
Breakthrough	0.64 ± 0.02	$\mathbf{0.67} \pm 0.02$	0.63 ± 0.02	0.66 ± 0.02	0.07 ± 0.04	0.04 ± 0.03

7. <(cell 2 1 black),<(cell 2 2 black),<(cell 2 3 black),(drop 2)>>>

この 7 つの組み合わせ特徴のうち，赤で示した 2，3，7 を追ってみると図 3.2 のようになり，最終的に 4 つのタプルの組み合わせによりカラム 2 に石を 4 つ並べることを意味する特徴となっている．Connect Four は 4 つ石を直線上に並べると勝利というゲームなので，勝利に直結するような特徴が iFDD$^+$ により実際に生成できることが確認できた．

次に，TD Search を除外した上で，より表現力の高い RFA 特徴を用いた場合について評価を行った．Nine Board Tic-Tac-Toe はゲームルールから相対的な関係を読み取れないため，実験から除外した．タプル特徴の場合と同様に Plain UCT に対する勝率を調べた結果を表 3.6 及び表 3.7 に示す．RFA 特徴をそのまま用いたものと，それを初期特徴とした iFDD$^+$ の間では，Connect Four 以外では性能の差はほとんど見られなかった．これらのゲームについては，RFA 特徴の表現力だけで価値関数の十分な近似が得られているか，あるいは新たに追加された組み合わせ特徴の重みを学習するにはシミュレーション数が少なすぎたという可能性がある．Connect Four では例外的に iFDD$^+$ が有意に高い勝率を示したが，これは 4 つの駒を直線上に並べるゲームであり，特に組み合わせ特徴が適したゲームであることが理由と考えられる．

3.3.5 $\alpha - \beta$ 探索での評価

次に，価値関数を $\alpha - \beta$ 探索に活用する場合に関して iFDD$^+$ の評価を行った．プレイを始める前にシミュレーション回数 10000 回固定で状態価値関数 $V(s)$ の学習を行い，その後深さ固定（2-ply）の $\alpha - \beta$ 探索を行うという方式で，以下の 3 つの設定を考え，特徴 $\phi_{tuple}(s)$ を用いたものに対する他の設定の勝率を測定した．

表 3.6: Plain UCT に対する勝率（RFA 特徴，100 シミュレーション毎ターン）

	RFA	iFDD $\xi = 1$	iFDD $\xi = 2$
どうぶつしょうぎ	0.58 ± 0.03	$\mathbf{0.59} \pm 0.03$	$\mathbf{0.59} \pm 0.03$
TTCC4	$\mathbf{0.69} \pm 0.03$	0.66 ± 0.03	0.68 ± 0.03
Connect Four	0.53 ± 0.03	$\mathbf{0.59} \pm 0.03$	0.58 ± 0.03
Breakthrough	$\mathbf{0.83} \pm 0.02$	0.82 ± 0.02	0.82 ± 0.02

表 3.7: Plain UCT に対する勝率（RFA 特徴，1000 シミュレーション毎ターン）

	RFA	iFDD $\xi = 1$	iFDD $\xi = 2$
どうぶつしょうぎ	$\mathbf{0.74} \pm 0.03$	0.73 ± 0.03	$\mathbf{0.74} \pm 0.03$
TTCC4	$\mathbf{0.93} \pm 0.02$	$\mathbf{0.93} \pm 0.02$	$\mathbf{0.93} \pm 0.02$
Connect Four	0.38 ± 0.03	0.44 ± 0.03	$\mathbf{0.45} \pm 0.03$
Breakthrough	$\mathbf{0.77} \pm 0.03$	0.74 ± 0.03	0.76 ± 0.03

- $\alpha - \beta$ 探索 特徴は $\phi_{tuple}(s)$
- $\alpha - \beta$ 探索 特徴は $\phi_{tuple^2}(s)$
- $\alpha - \beta$ 探索 特徴は iFDD$^+$，初期特徴は $\phi_{tuple}(s)$ $(\xi = 1)$

さらにここで，iFDD$^+$ のバリエーションとして，特徴ベクトルを生成する際に組み合わせ元の特徴を残したまま新たな特徴を追加するもの（iFDD$^+$0）を考え，それに関しても同様に評価を行った．これは組み合わせ特徴 $f = g \wedge h$ に関して $\phi_g(s) = \phi_h(s) = \phi_h(s) = 1$ となるような特徴ベクトルを生成するものであり，元の特徴がアクティブであり続ける代わりに新たに追加される特徴の重みは 0 で初期化される．

それらの結果を表3.8 に示す．傾向としては $\phi_{tuple}(s)$ を用いたものより $\phi_{tuple^2}(s)$ や iFDD$^+$/iFDD$^+$0 を用いたものの勝率が高く，大きな例外は iFDD$^+$ の Breakthorugh における低い性能である．iFDD$^+$0 ではそのような性能低下が見らないため，特徴を組み合わせ特徴で置き換えてしまうことが性能に悪影響をおよぼす場合があると考えられる．特に学習に割けるリソースが限られる GGP においては，ある特徴の重みが収束する前にその特徴を組み合わせ特徴で置き換えてしまうしまうことで学習を阻害してしまうことがありうる．他のゲームにおいても，有意な差とは言えないものの，iFDD$^+$ より iFDD$^+$0 の方が高い勝率を達成している．$\phi_{tuple^2}(s)$ と iFDD$^+$ 及び iFDD$^+$0 を比べると，総じて大きな差は無く，比較的 Connect Four において iFDD$^+$ や iFDD$^+$0 が優位にある点は UCT における結果と同様である．

3.3.6 時間を固定した評価

ここまでの評価は全てシミュレーション数あるいは先読みの深さを固定した評価であったが，GGP の一般的なフォーマットにおいては，ゲームルールを与えられてから実際にプレイを始めるまでの時間と各ターンの思考時間の 2 つの時間的制約が対戦の度に定められ，これが探索や学習に利用可能なリソースを決める．したがってサンプル効率だけでなく，計算にかかる時間がプレイヤの性能に大きく影響する．

表 3.8: $\phi_{tuple}(s)$ 特徴を用いた $\alpha - \beta$ 探索に対する勝率（対戦前の学習に 10000 シミュレーション，各ターン 2-ply の探索）

	$\phi_{tuple^2}(s)$	iFDD$^+$ $\xi = 1$	iFDD$^+$0 $\xi = 1$
どうぶつしょうぎ	0.63 ± 0.05	0.65 ± 0.05	$\mathbf{0.67} \pm 0.04$
Nine Board Tic-Tac-Toe	$\mathbf{0.5} \pm 0.05$	0.46 ± 0.05	0.49 ± 0.04
TTCC4	0.51 ± 0.04	0.59 ± 0.04	$\mathbf{0.6} \pm 0.04$
Connect Four	0.49 ± 0.05	0.55 ± 0.05	$\mathbf{0.58} \pm 0.04$
Breakthrough	$\mathbf{0.53} \pm 0.05$	0.21 ± 0.04	0.52 ± 0.04

iFDD$^+$ を用いて特徴生成を行う場合，学習の際には組み合わせ候補について関連度を計算するコストがかかり，学習した価値関数を利用する際にもアクティブな組み合わせ特徴を探索するコストがかかる．これらのオーバーヘッドは実装の詳細に左右されるとはいえ，これがプレイヤの性能に大きな悪影響を与えないかどうかは検証が必要だといえる．

そこで，ゲームルールが与えられてから 10 秒間を学習に用い，それから各ターン 1 秒の思考時間を探索に用いるという設定で，RFA 特徴を用いた UCT と，RFA 特徴を初期特徴とした iFDD$^+$0 及び iFDD$^+$（$\xi = 1$）を用いた UCT を対戦させ，勝率を計測した．その結果が表 3.9 であり，iFDD$^+$0 は Connect Four では大幅に勝ち越している一方，その他のゲームでは同等かあるいはやや負け越している．一般的な GGP のフォーマットで iFDD$^+$ がどれほど有効かを検証するためにはより多くのデータが必要であるが，この結果から，少なくとも一部のゲームに対しては iFDD$^+$ は計算上のオーバーヘッドが存在してもなお有用であると考えられる．

3.4 まとめ

本論文では GGP において，未知のゲームに対するゲーム固有知識の効果的な学習を実現するために，特徴拡張アルゴリズム iFDD$^+$ を用いた強化学習により組み合わせ特徴を自動的に生成することを提案し，これをゲーム木探索と組み合わせる方法を検討しその有効性を評価した．

iFDD$^+$ により学習された価値関数の利用方法として，UCT，TD Search，$\alpha - \beta$ 探索の 3 つの探索手法における具体的な利用について示し，特に UCT が TD Search に比べ優れている傾向が対戦結果から確認された．

iFDD$^+$ により生成される組み合わせ特徴は，GDL のタプルから成る単純な特徴と比べた場合，プレイヤの勝率を高める上で有用であった．より表現力の高い特徴と比較した場合においても，Connect Four のような複数の駒の組み合わせが重要な役目を果たすゲームにおいては性能向上に役立ち，それは iFDD$^+$ の計算上のオーバーヘッドが問題となる限られた時間での対戦においても変わらないことが確認された．またゲームによっては，組み合わせ特徴の追加方法を変更するような iFDD$^+$ のバリエーションが有効である可能性も示唆された．

表 3.9: RFA 特徴を用いた UCT に対する，RFA 特徴を初期特徴とした iFDD$^+$ 及び iFDD$^+$0 を用いた UCT の勝率（対戦前の学習時間 10 秒，思考時間 1 秒毎ターン）

	iFDD$^+$ $\xi = 1$	iFDD$^+$0 $\xi = 1$
どうぶつしょうぎ	0.48 ± 0.04	0.48 ± 0.04
TTCC4	0.45 ± 0.04	0.42 ± 0.04
Connect Four	0.64 ± 0.04	$\mathbf{0.68} \pm 0.04$
Breakthrough	0.41 ± 0.04	0.48 ± 0.04

3.5 今後の課題

今後の課題としては，まずより多くのゲームにおいて評価実験を行うべきだと考える．本研究では評価を容易にするために同時着手を含まない 2 人ゲームのみを使用して評価を行ったが，IGGPC では 1 人ゲームや同時着手を含むゲームは珍しくない．そのようなゲームも含め評価を行うことで，GGP のエージェントとしての汎用性を評価することが可能になる．

また，より強いプレイヤを実現するためには，最適なパラメータの値や価値関数の利用法，アルゴリズムの設計に関してより広範に検討を行う必要もあると考える．中でも iFDD$^+$ とともに用いる強化学習アルゴリズムとして本論文では Sarsa(0) しか用いていないものの，$\lambda > 0$ の場合や Q-learning，バッチ強化学習など幅広い選択肢があるため，GGP における厳しい時間的制約とのバランスをとりつつ効率的に学習を行うことができる方法について検討する必要がある．

第4章　Arcade Learning Environmentのためのニューラルネットを用いたモデル学習

4.1　Arcade Learning Environment

Arcade Learning Environment (ALE) [7] は Atari 2600 というゲーム機のエミュレーション機能を元とし，さらにそれに学習・探索といった AI 技術の評価に使用しやすいようなインタフェースが加わったものである．

ALE のインタフェースを通じて，エージェントは

- 現在のスクリーン（210×160）の各ピクセルの色（128 colors）
- 整数で表される報酬信号
- 終端状態か否か

といった情報を毎フレームごとに観測し，行動を選択する．

4.1.1　General Game Playing との差異

Arcade Learning Environment は 3 章で述べた General Game Playing と同じく，ゲーム依存の知識を利用せずにうまくゲームをプレイできるエージェントの実現を目指すことを目的とし，そのための評価フレームワークとしての役割を果たす．しかしパズルやボードゲームといった種類のゲームを主に扱う GGP と，ビデオゲームを扱う ALE との間には異なる点も多い．主な差異について表 4.1 にまとめる．

4.2　関連研究

4.2.1　Deep Q-Networks

Deep Q-learning [4] は行動価値関数の近似のために深層ニューラルネットワークを利用する強化学習アルゴリズムである．そこで使用されるネットワークを特に Deep Q-Network (DQN) と呼ぶ．

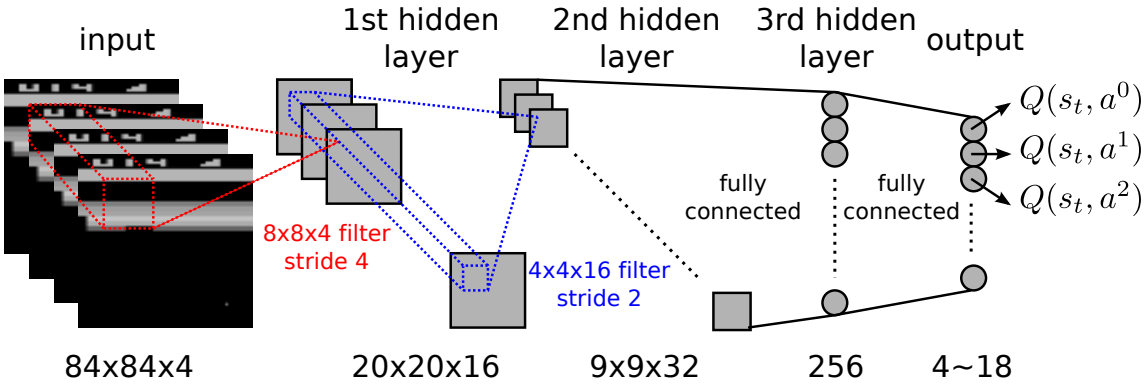


図 4.1: Deep Q-Networks の構成

DQN の構成全体像を図 4.1 に示す．DQN の特徴は，特徴抽出を必要とせず，画面のピクセルをほぼそのまま入力として受け取ることである．元のスクリーンの大きさは 210×160，色は 128 色であるが，それを

1. RGB による表現をグレースケールに変換する
2. 110×84 のサイズにダウンサンプリングする
3. ゲームの主要なエリアを含むように 84×84 の領域を切り取る

という処理により 84×84 の表現に変換する．こうして得られたスクリーン画像を直近の 4 フレーム分だけ入力として用いる．

入力の後の DQN の層の構成は

表 4.1: Arcade Learning Environment と General Game Playing の主な違い

	Arcade Learning Environment	General Game Playing
1 ターンの時間	1/60 秒	数十秒
環境モデル	与えられない	GDL として与えられる
各ターンに観測できる情報	スクリーンの画像	前のターンの他のプレイヤの行動
プレイヤの人数	1 人	1 人以上（上限なし）
ランダム性	あり	なし
学習に使用できる時間	通常は制限なし	数十〜数百秒
報酬信号を観測できるタイミング	エピソード中に随時	エピソード終了時のみ

Procedure 4.1 Deep Q-learning with Experience Replay [4]

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    $a_t \leftarrow \epsilon$-greedy action from state s
    Execute a_t, observe reward r and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

- 入力：$84 \times 84 \times 4 = 28224$
- 層：4 層
  - 8×8 のフィルター 4 個による畳み込み層（ストライド：4）
  - 4×4 のフィルター 16 個による畳み込み層（ストライド：2）
  - 256 のユニットからなる全結合層
  - 18 のユニットからなる全結合層
- 出力：各行動の行動価値

となっており，出力層以外の各層では活性化関数として Rectifier Nonlinearity（ReLU）を用いている．

Deep Q-learning により DQN のパラメータを学習する際の擬似コードを Algorithm 4.1 に示す．Deep Q-learing では Experience Replay [31] と呼ばれる仕組みを採用している．各タイムステップ t において状態遷移 (s_t, a_t, r_t, s_{t+1}) を経験した際に，Q-learning ではその遷移を元に Q 関数を更新するが，Deep Q-learning ではその状態遷移を Replay Memory と呼ばれる領域に記録しておき，そこからランダムサンプリングした遷移を元に Q 関数を更新する．これにより，現在の振る舞いだけではなく，これまでの過去の振る舞いに渡って訓練サンプルを得ることができ，パラメータが発振・発散することを防ぐことができる．

Atari 2600 の 7 つゲームに対する Deep Q-learning によるスコアを，その他の強化学習アプローチ及び人間によるスコアとともに表 4.2 に示す．

4.2.2 ALE における探索

ALE では現在の状態をプログラム中で保存し，また保存した状態を復元するというインタフェースが提供されており，これを用いて間接的に真の環境モデルにアクセスすることができる．具体的には

1. 現在の状態 s_t を保存する
2. 行動 a_t を選択する
3. 報酬信号 r_t を観測し，次の状態 s_{t+1} に遷移する
4. 必要に応じて s_{t+1} に対し処理を行う
5. s_t を復元する

とすることで，状態 s_t にいながら次の状態 s_{t+1} を予測するのと同じ効果が得られ，これを繰り返し用いることで ALE においても UCT のようなゲーム木探索手法を利用することができる．UCT を用いたプレイヤは学習なしで Deep Q-Networks より高いスコアを達成することも可能である [7].

表 4.2: 7 つのゲームに対する Deep Q-learning とその他の強化学習アプローチの平均スコアの比較．Sarsa，Contingency，DQN では $\epsilon = 0.05$ の ϵ-greedy に従って行動を選択している．Random は一様な分布に従いランダムに行動を選んだ場合の平均スコアを表す．[4]

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	-20.4	157	110	179
Sarsa [7]	996	5.2	129	-19	614	665	271
Contingency [32]	1743	6	159	-17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	-3	18900	28010	3690

表 4.3: 7 つのゲームに対する UCT を用いたプレイヤと DQN の平均スコアの比較 [33]

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
DQN	4092	168	470	20	1952	1705	581
UCT	7233	406	788	21	18850	3257	2354
UCT-I	5388	215	601	19	13189	2701	670

しかしながら，この真のモデルによる先読みの処理速度は遅く，UCT により Deep Q-Networks よりも高いスコアを達成するためには行動を選ぶ度に数秒の計算時間が必要とされるため，ゲームを実際の時間でプレイすることができなくなってしまう．そこで [33] では，UCT を DQN の訓練データの生成のためだけに用い，実際のプレイングの評価はそうして訓練された DQN を使って行うことで，プレイングのリアルタイム性を維持しながら Deep Q-learning よりも高いスコアを達成している．

4.3 提案手法

オフラインモンテカルロ木探索の実験結果からもわかるように，ALE においてもうまいプレイングのために先読みは重要な役割を果たす．そのためには環境モデルが必要となるが，人間がビデオゲームをプレイする際にはそのようなモデルは通常手に入らないため，たとえ AI の訓練の際のみであっても真のモデルを利用することは人間と比較する上で公平とは言えない．またロボットの制御等といった現実世界の問題ではそもそも真のモデルは手に入らないため，そのような問題への応用を考える上で真のモデルに依存することは好ましくない．

そこで本研究では，環境とのインタラクションを通じてモデルを学習することを目指す．未知のゲームに対してモデル学習を行うためには，価値関数を学習する際と同様に，有効な特徴の抽出が重要となる．画面中のオブジェクトの座標や位置関係のようなゲーム中のダイナミクスに深く関わる特徴を抽出することができれば，モデル学習の助けになると考えられる．

4.3.1 DQN に基づくモデル

ALE のダイナミクスは以下のモデルにより表現することができる．

- 遷移モデル $T : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$
- 報酬モデル $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{Z}$ （$\mathbb{Z}$ は整数全体の集合）
- 終端モデル $\Omega : \mathcal{S} \times \mathcal{A} \mapsto \{0, 1\}$

これらのモデルを用いて，状態 s と行動 a から

- 報酬信号 $R(s, a)$ を観測する
- もし $\Omega(s, a) = 1$ であればエピソードは終了する
- もし $\Omega(s, a) = 0$ であれば次の状態 $T(s, a)$ に遷移する

という 1 タイムステップ先の未来を予測することができる．この予測を繰り返すことで時間の許すかぎり先の未来まで予測することが可能になる．

以上の 3 種類のモデルを学習により獲得するために，本論文では，訓練済みの DQN を利用することを提案する．DQN は行動価値関数を近似するように訓練されたネットワークであり，ゲームを上手くプレイできる DQN では行動価値の予測のために必要なゲーム中の情報をネットワーク中で捕捉できていると考えられる．そうして得られた情報はゲームのダイナミクスの予測のためにも有用であると予想される．そこで，訓練済みの DQN を特徴抽出器として用い，その特徴空間においてモデル学習を行うことを考える．

DQN のユニット数 n の隠れ層の出力の空間を $\mathcal{H} \subseteq \mathbb{R}^n$ と表すこととする．その層の前後で DQN を

- $\phi_{\mathcal{H}} : \mathcal{S} \mapsto \mathcal{H}$
- $Q_{\mathcal{H}} : \mathcal{H} \times \mathcal{A} \mapsto \mathbb{R}$

の 2 つの関数に分割することができ，DQN によって近似される行動価値関数 $Q_{DQN}(s, a)$ は

$$Q_{DQN}(\phi(s), a) = Q_{\mathcal{H}}(\phi_{\mathcal{H}}(s), a) \tag{4.1}$$

と表される．このように DQN を分割することで，入力側の $\phi_{\mathcal{H}}$ をモデル学習のための特徴抽出器として用いることができる．

$\phi_{\mathcal{H}}$ によってマップされる特徴空間 $\mathcal{H}$ 上で，上で述べた 3 種類のモデル

- 遷移モデル $T_{\mathcal{H}} : \mathcal{H} \times \mathcal{A} \mapsto \mathcal{H}$
- 報酬モデル $R_{\mathcal{H}} : \mathcal{H} \times \mathcal{A} \mapsto \{-1, 0, 1\}$
- 終端モデル $\Omega_{\mathcal{H}} : \mathcal{H} \times \mathcal{A} \mapsto \{0, 1\}$

を定義することができる．次の状態の予測は状態空間 $\mathcal{S}$ 上ではなく $\mathcal{H}$ 上で行われ，DQN の後半部 $Q_{\mathcal{H}}$ を適用することで DQN による行動価値の予測も任意の時点で行うことができる．[4] と同様に報酬信号の値を正ならば 1，負ならば -1，どちらでもなければ 0 となるように正規化して用いるため，報酬モデルは任意の整数値ではなく $\{-1, 0, 1\}$ を予測するように定義する．

モデル学習のために用いる DQN の隠れ層としては，3 番目の隠れ層である全結合層の 256 次元の表現を用いる．DQN の入力 $84 \times 84 \times 4 = 28224$ と比べれば次元数は大幅に少なく，その入力をそのまま利用する場合と比べてモデルの学習及びモデルを用いた探索が効率的に行えるようになる．この層の表現としては ReLU を通した後の値を用いる．これは ReLU を通す前の表現に比べよりスパースで予測が行いやすいことが期待されるとともに，その表現をそのまま最終層に流すことで $Q_{\mathcal{H}}$ の評価がより効率的に行える．この層の表現の抽出とその前後での DQN の分割を図 4.2 に示す．

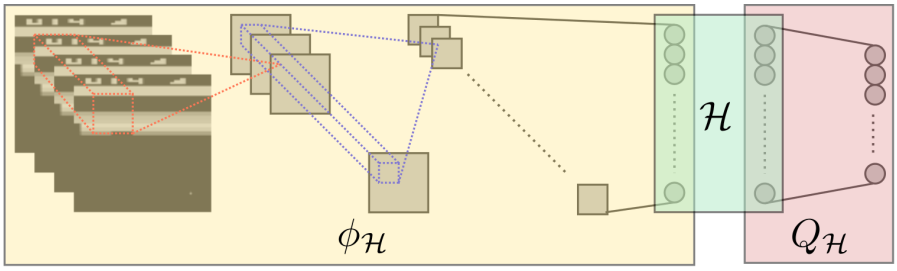


図 4.2: DQN の 3 番目の隠れ層 $\mathcal{H}$ の表現とその前後での DQN の分割

4.3.2 ニューラルネットワークによるモデルの近似

前節で述べた DQN に基づく 3 種類のモデル $T_{\mathcal{H}}, R_{\mathcal{H}}, \Omega_{\mathcal{H}}$ を学習するために，まずそれらをどのように近似するかを検討する必要がある．$\mathcal{H}$ は実数値のベクトルであるため，この問題は連続的な状態空間上でのモデル学習とみなすことができる．連続状態空間における確率的なモデルの学習はとりわけ難しい課題であり，ガウス過程を利用したアプローチ [34] 等が提案されているが，ゲームのようなおそらく多峰性のある状態確率分布をうまくモデル化できるようなアプローチは限られ，特定のゲームに特化したモデリングが行えない ALE のフレームワークにおいては極めて困難だと言える．

このような困難を避けるために，本研究では，ALE のダイナミクスを決定的なものとみなしてモデル学習を行う．ゲームによってはスクリーン画面の情報からはわからない乱数シードに依存するランダム性が存在し，またスクリーン画面そのものではなくそれに前処理を加え $\phi_{\mathcal{H}}$ により特徴空間 $\mathcal{H}$ にマッピングする過程で失われる情報に起因する不確定性が存在する可能性があるため，このアプローチは問題を生じさせる可能性があることに留意する必要がある．

ALE のダイナミクスを決定的なものとみなすことで，遷移モデル $T_{\mathcal{H}}$ は連続値ベクトル $\phi_{\mathcal{H}}(s)$ と離散変数としての行動 a から連続値ベクトル $\phi_{\mathcal{H}}(T(s,a))$ を近似する回帰問題として，報酬モデルと終端モデルはそれぞれ $\{-1,0,1\}$ 及び $\{0,1\}$ を予測する多クラス分類問題として解くことが可能になる．これらのモデルはそれぞれ別に学習することも可能であるが，これらは共に未来の予測であるという点で共通して必要とする情報が多いと予想される．モデル間でそれらの情報を隠れ層の表現を通じて共有できるように，本研究ではこれら 3 つのモデルを 1 つの多層ニューラルネットワークにより近似する．

このニューラルネットワークの構成は

- 入力：$256 + 18 = 274$ 次元
  - 状態を表す 256 次元の表現 $h \in \mathcal{H}$
  - 行動を表す 18 次元のベクトル（選択する行動に対応する次元の値が 1，それ以外は 0）
- 層：4 層

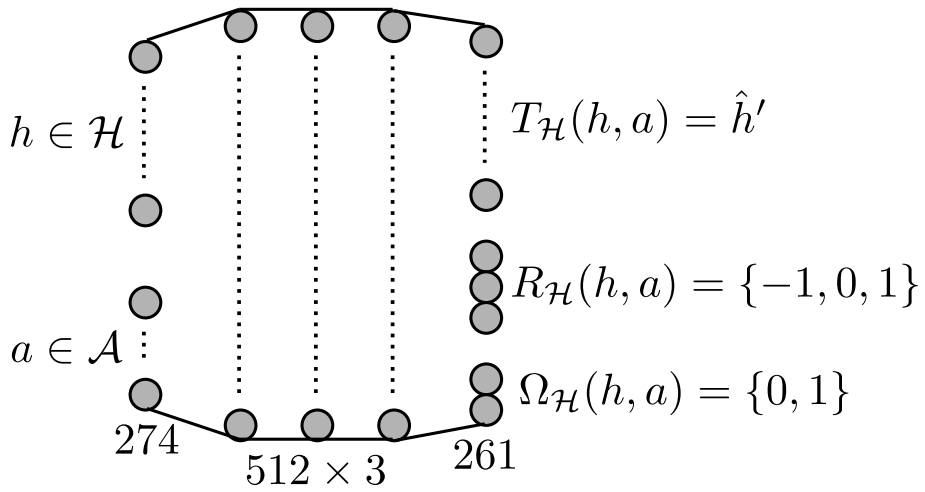


図 4.3: 遷移モデル $T_{\mathcal{H}}$，報酬モデル $R_{\mathcal{H}}$，終端モデル $\mathcal{H}$ の 1 つのニューラルネットワークによる表現

  - 512 のユニットからなる全結合層（1）
  - 512 のユニットからなる全結合層（2）
  - 512 のユニットからなる全結合層（3）
  - 261 のユニットからなる全結合層

- 出力：$256 + 3 + 2 = 261$ 次元
  - 遷移モデル：入力状態と出力状態 $h' \in \mathcal{H}$ の差を表す 256 次元の表現 $h' - h$
  - 報酬モデル：$\{-1, 0, 1\}$ の 3 クラスそれぞれのスコア
  - 終端モデル：$\{0, 1\}$ の 2 クラスそれぞれのスコア

とし，図 4.3 に示す．遷移モデルに対応する出力では $\phi_{\mathcal{H}}(s')$ そのものではなく差分 $\phi_{\mathcal{H}}(s') - \phi_{\mathcal{H}}(s)$ を出力するようにし，モデルとして用いる際には出力した値に $\phi_{\mathcal{H}}(s)$ を加えるようにする．報酬モデル及び終端モデルに関しては，スコアの最も高いクラスをモデルの出力とする．

遷移モデルに対応する出力に関しては，状態 s と行動 a から予測した次状態の表現 $T_{\mathcal{H}}(\phi_{\mathcal{H}}(s), a)$ と，実際に状態 s で行動 a を選択して観測された次状態 s' から計算された $\phi_{\mathcal{H}}(s')$ との間の二乗誤差

$$L_{\mathcal{H}}(s, a, s') = |T_{\mathcal{H}}(\phi_{\mathcal{H}}(s), a) - \phi_{\mathcal{H}}(s')|^2 \tag{4.2}$$

を最小化するというのが最も直接的な方法として考えられる．この場合のデータの流れを図 4.4 に示す．しかし，ニューラルネットワークの内部表現に現れる各次元の値は必ずしも最終的な出力を計算する上で必要ではなく，その値を正しく予測できるようにモデルを訓練することはモデルの表現力

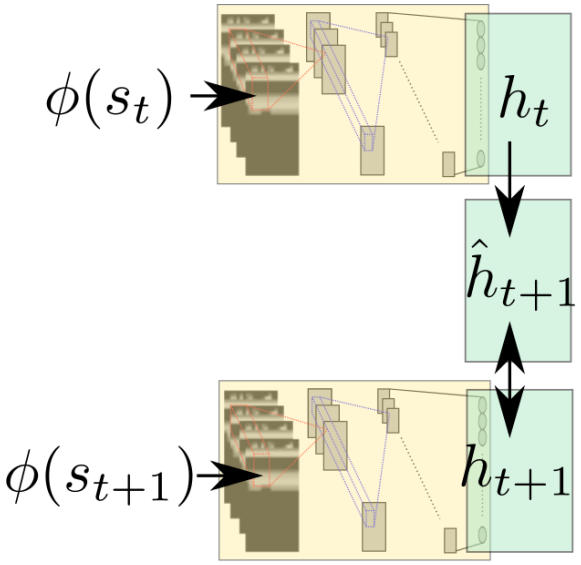


図 4.4: 内部表現の予測誤差に基づく遷移モデルの学習の際のデータの流れ

を浪費することになりかねない．また，たとえ最終的な出力を計算する上で必要な値だったとしても，その値のスケールは次元によって異なり，必ずしも最終的な出力への寄与と一致してはいない．そのため，値に大きな幅のある次元の値を予測できるようにモデルが訓練される一方で，値の幅が小さい次元の値の予測が疎かになってしまうおそれがある．

このような問題を解決するために，本研究では遷移モデルに対応する出力に関しては，表現そのものの予測誤差ではなく，表現から計算した行動価値の誤差

$$L_Q(s,a,s') = \sum_b |Q_{\mathcal{H}}(T_{\mathcal{H}}(\phi_{\mathcal{H}}(s),a),b) - Q_{\mathcal{H}}(\phi_{\mathcal{H}}(s'),b)|^2 \tag{4.3}$$

を最小化するようにモデルのパラメータを更新する．この場合のデータの流れを図 4.5 に示す．これにより，行動価値の予測誤差への寄与に応じてパラメータを更新することが可能になり，モデルにより予測した未来の状態に対して DQN による行動価値の評価を行うというような先読みを含む評価がより正確に行えるようになると予想される．

報酬モデル $R_{\mathcal{H}}$ 及び終端モデル $\Omega_{\mathcal{H}}$ の学習は，多クラス分類問題として交差エントロピー（*cross-entropy*）誤差関数を最小化することによって行うことができる．

4.3.3 部分的 Hallucinated Replay

ここまで DQN に基づくモデル学習の方法について説明してきたが，このようなモデル学習により獲得されるモデルはあくまで近似的なものであり，未来についての予測には誤りが含まれる可能性

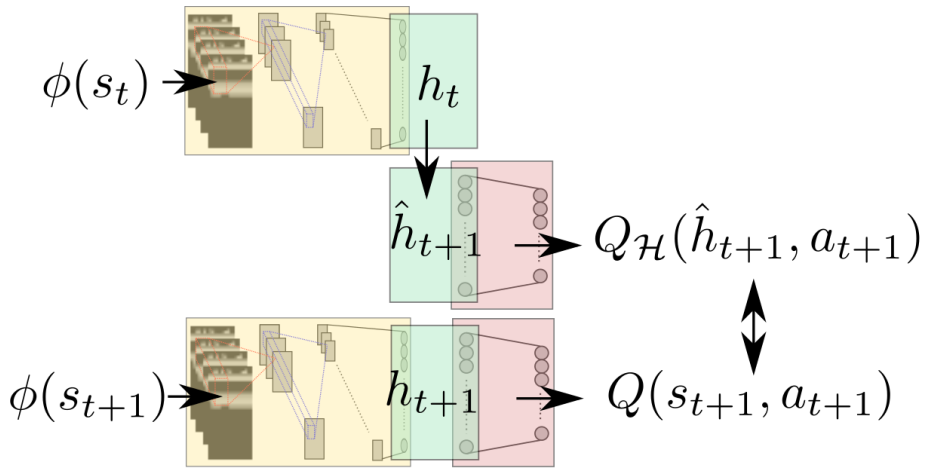


図 4.5: 内部表現から計算した行動価値の誤差に基づく遷移モデルの学習の際のデータの流れ

がある．現在の状態 s_t を元にモデルを利用して次状態 s_{t+1} を求めたところ，予測された次状態 s_{t+1} が誤りを含んでいたとする．誤りを含んだ s_{t+1} にさらにモデルを適用してその次の状態 s_{t+2} を予測しようとすれば，元の状態 s_{t+1} がそもそも誤りを含んでいるためにそこから予測された s_{t+2} はさらに大きな誤りを含んだものになりかねない．このように誤りを含むモデルを繰り返し適用するとその誤差が積み重なり，予測される未来はすぐに正しい未来から乖離してしまう．

このような問題を解決するためのモデル学習の手法として Hallucinated Replay [35] が提案されている．モデルを繰り返し適用するということはモデルの出力がそのままモデルの入力として使用されるということである．しかしモデルは通常は実際の遷移を教師として学習されているため，モデルにより生成される人工的な遷移と教師として使用されている実際の遷移の分布に不一致が生じることとなり，それゆえ後者で学習を行っているモデルは前者においては予測を行うことが難しくなる．Hallucinated Replay は，過去の遷移を元にモデル学習を行う際に，実際の遷移だけを教師として用いるのではなく，より前の状態からモデルで予測して生成した架空の遷移も教師データとして用いることによって，このような入力の分布の不一致をなくすテクニックである．

[35] では直近の 2 つの観測を元に次の観測を予測するという設定において，実際の 2 つの観測のうち片方をモデルの出力で置き換えて学習を行うという実装方法を提示している．これにより，観測の片方にモデルが生じさせた誤りが含まれていたとしても，それを訂正するように次の観測を予測することが可能になる．しかし，本研究のモデルの定式化では，予測の手がかりとして使える観測は現在のタイムステップの観測 1 つとなっている．遷移モデルの学習に用いる過去の遷移について，予測の元となる現在の観測を

$$(h_t, a_t, r_t, h_{t+1}) \to (T_{\mathcal{H}}(h_{t-1}, a_{t-1}), a_t, r_t, h_{t+1}) \tag{4.4}$$

のようにさらに 1 つ前の観測を元にモデルで予測したものに置き換えてしまうと，そもそも $T_{\mathcal{H}}(h_{t-1}, a_{t-1})$

Procedure 4.2 部分的 Hallucinated Replay 利用したモデル学習

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize $T_{\mathcal{H}}, R_{\mathcal{H}}, \Omega_{\mathcal{H}}$ with random weights
for episode $= 1, M$ **do**
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        $a_t \leftarrow \epsilon$-greedy action from state s
        Execute a_t, observe reward r and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample $M - H$ random transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        $h_j \leftarrow \phi_{\mathcal{H}}(\phi_j)$
        $h_{j+1} \leftarrow \phi_{\mathcal{H}}(\phi_{j+1})$
        Create $M - H$ real transitions (h_j, a_k, r_k, h_{j+1})
        Sample H random transitions $(\phi_{k-1}, a_{k-1}, r_{k-1}, \phi_k, a_k, r_k, \phi_{k+1})$ from $\mathcal{D}$
        $\hat{h_k} \leftarrow T_{\mathcal{H}}(\phi_{\mathcal{H}}(\phi_{k-1}), a_{k-1})$
        $h_{k+1} \leftarrow \phi_{\mathcal{H}}(\phi_{k+1})$
        Create H hallucinated transitions $(\hat{h_k}, a_k, r_k, h_{k+1})$
        minibatch of transition $\leftarrow$ real trainsitions $\cup$ hallucinated transitions
        Update $T_{\mathcal{H}}, R_{\mathcal{H}}, \Omega_{\mathcal{H}}$ on M minibatch transitions
    end for
end for

が既にある程度正確な予測だというのでない限り，それを元に h_{t+1} を予測することは困難になるはずである．したがって，全ての遷移についてこのように元の観測を置き換えてしまうわけにはいかず，置き換えるとしてもあくまで一定の割合の遷移についてだけ行うようにする必要があると考えられる．

そのための具体的なアルゴリズムとして，Deep Q-learning の学習と同様に容量 N の Replay Memory $\mathcal{D}$ からサンプリングした M 個の過去の遷移を使って一度にモデルを更新するミニバッチ学習をまず考え，その際にその M 個のうち H 個に対して式（4.4）のような置換えを行うことを考える．この方法を部分的 Hallucinated Replay と呼ぶことにし，その場合のアルゴリズムの擬似コードを Algorithm 4.2 に示す．$H = 0$ であれば Hallucinated Replay を全く行わない場合，$H = M/2$ であればミニバッチのうちちょうど半分に対して Hallucinated Replay を行う場合を意味することになる．遷移モデルに対して Hallucinated Replay を使用し訓練を行う際のデータの流れを図 4.6 に示す．

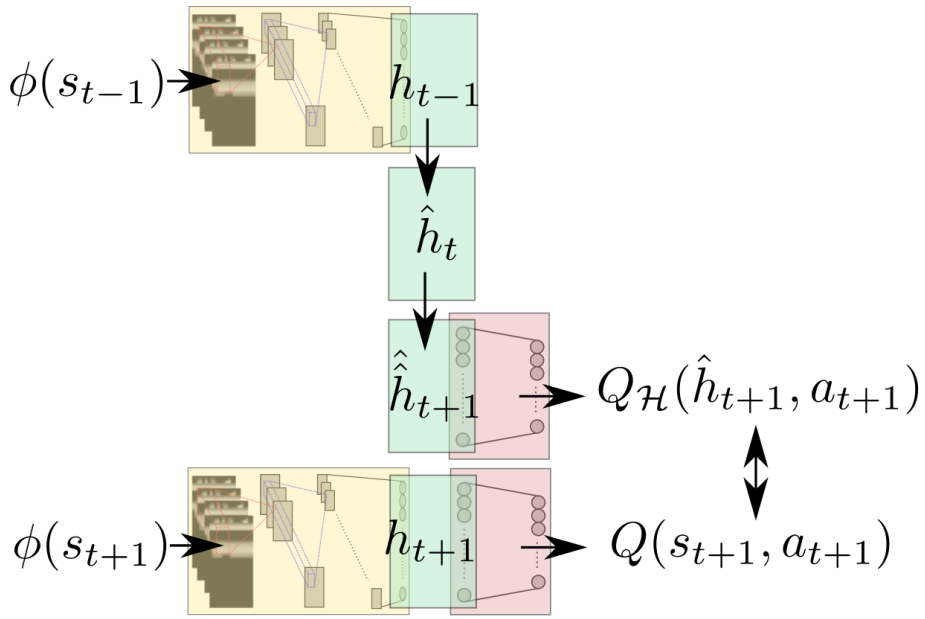


図 4.6: Hallucinated Replay による置換えを行った場合の遷移モデルの学習の際のデータの流れ

4.3.4 モデルを利用した先読み

モデルが手に入れば，それを用いて先読みを行うことができる．効果的な先読みのためには，まずモンテカルロ木探索のような成功している探索手法を適用することが考えられる．しかし，深い先読みを行うためにはモデルを繰り返し適用しなければならなず，モデルが十分に高速である必要がある．また，モデルが不完全である場合には，深い先読みが可能だとしてもその先読みが十分に正確でなければ性能向上に繋がらず，むしろ先読みを行わない場合よりも性能を悪化させてしまうおそれもある．

本論文では，最も単純な先読み方法として深さ固定のマックス探索による先読みについて検討する．深さ 1 のマックス探索による行動評価を

$$Q^1_{T,R,\Omega}(s,a) = \begin{cases} R(s,a) & (\Omega(s,a)=1) \\ R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_{DQN}(T(s,a),a') & (\Omega(s,a)=0) \end{cases} \tag{4.5}$$

のように定義する．これは状態 s における行動 a の価値を，1 タイムステップ先の状態 $T(s,a)=s'$ で $Q_{DQN}(s',a')$ を最大にするような a' を選ぶという前提で評価するものである．これを用いて深さ d のマックス探索による行動評価も同様に

$$Q^d_{T,R,\Omega}(s,a) = \begin{cases} R(s,a) & (\Omega(s,a)=1) \\ R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^{d-1}_{T,R,\Omega}(T(s,a),a') & (\Omega(s,a)=0) \end{cases} \tag{4.6}$$

と定義する．状態 s_t において深さ N のマックス探索により行動 a_t を選択することは，s_t を起点として可能な行動の順列 $(a_t, a_{t+1}, \ldots, a_{t+N})$ 全てを考え，それらのうち期待報酬が最大となるものを選ぶという操作に等しい．

ALE では 1 フレームあたりの時間が 1/60 秒と短く，[4] と同様に 4 フレームをまとめて 1 つのタイムステップとして扱ったとしても 1 秒先を予測するために深さ 15 の先読みを行わなければならない．深さ D のマックス探索の時間計算量が $O(|\mathcal{A}|^d)$ であるため，深さを 1 増やすだけで計算時間が $|\mathcal{A}|$ 倍になってしまい，長期間の先読みを行うことは難しい．一方，人間がビデオゲームをプレイする際の入力について考えてみると，特定のボタンを 1/15 秒の間だけ押すという操作は必須のものではなく，より長い時間同一のボタンを押し続けるものとしてもほとんどの場合問題無いと予想される．そこで先読みの際に，同じ行動をを n タイムステップの間繰り返すこととし，その n タイムステップが深さ 1 に当たるものとしてマックス探索を行うようにすることを考える．こうすることで同じ深さの探索でも n 倍先まで予測することができ，かつ計算コストの増加は n 倍で抑えられる．

先読みにおいて n タイムステップを最小単位として扱うために，1 ステップ単位のモデル T, R, Ω を元に以下のような n ステップモデルを構成する．

$$T^n(s,a) = \begin{cases} \text{undefined} & (\Omega^n(s,a) = 1) \\ T(s,a) & (\Omega^n(s,a) = 0 \text{ and } n = 1)) \\ T^{n-1}(T(s,a),a) & (\Omega^n(s,a) = 0 \text{ and } n > 1) \end{cases} \tag{4.7}$$

$$R^n(s,a) = \begin{cases} R(s,a) & (n = 1 \text{ or } \Omega(s,a) = 1) \\ R(s,a) + R^{n-1}(T(s,a),a) & (n > 1 \text{ and } \Omega(s,a) = 0) \end{cases} \tag{4.8}$$

$$\Omega^n(s,a) = \begin{cases} \Omega(s,a) & (n = 1) \\ 1 & (n > 1 \text{ and } \Omega(s,a) = 1) \\ \Omega^{n-1}(T(s,a),a) & (n > 1 \text{ and } \Omega(s,a) = 0) \end{cases} \tag{4.9}$$

これらを用いて，深さ d，探索単位 n タイムステップのマックス探索による評価は $Q^d_{T^n,R^n,\Omega^n}(s,a)$ と書き表すことができる．

4.4 評価

4.4.1 DQN の訓練

訓練済みの DQN をモデル学習において利用するためには，まず訓練された DQN が手に入らなければならない．しかしながら，[4] の著者らは実験に使用したプログラムの実装を公開しておらず，アルゴリズムの詳細に関しても不明な点が数多くある．Deep Q-learning を実装しその結果の再現を

試みるプロジェクトはインターネット上で複数確認することができるが，その細部については実装者によって異なっており，未だ完全な再現に成功した例は確認できない．

本研究では独自に Deep Q-learning を実装し，結果の再現を試みた．本研究で使用した実装には [4] で述べられているものとは異なるかもしれない点が多数ある．それらについて表 4.4 にまとめる．

深層ニューラルネットワークの実装に関しては，Caffe（`http://caffe.berkeleyvision.org/`）というフレームワークを用いた．Caffe は Yangqing Jia により開発され，現在も Berkeley Vision and Learning Center 及びオープンソースコミュニティにより活発に開発が進められており，高速にネットワークの訓練・評価を行うことができる．

[4] では 7 つのゲーム（Beam Rider，Breakout，Enduro，Pong，Q*bert，Seaquest，Space Invaders）について評価を行っており，それら全てのゲームにおいて DQN は既存の強化学習アプローチに比べ良い成績を残している．しかし，上に述べたような実装に基づきこの結果の再現を試みたところ，Breakout，Enduro，Pong の 3 つのゲーム以外では有効な学習を行うことができなかった．これは [4] で使用された実装との差異によるものと思われる．特にスクリーンの前処理の方法は画面上のオブジェクトを DQN が識別できるかどうかどうかに関わるため，性能への影響が大きいものと考えられる．

本研究で DQN の学習に成功した 3 つのゲーム（Breakout，Enduro，Pong）について，訓練中の DQN の性能の変化を 100 万イテレーションごとに評価した．評価の際には $\epsilon = 0.05$ の ϵ-greedy 方策に従って 100 エピソードのプレイを行い，スコアの平均，最大値，最小値を調べた．その結果をそれぞれ図 4.7，図 4.8，図 4.9 に示す．平均スコアのエラーバーは 95%信頼区間を表す．Breakout では [4] より高い平均スコアが得られている一方，Enduro 及び Pong では [4] より若干低いスコアしか得られていない．

表 4.4: 本論文で用いる Deep Q-Networks の実装と [4] で用いられたものとの差異

	本論文	[4]
スクリーンのグレースケール変換	輝度（$0.21R + 0.72G + 0.07B$）	不明
スクリーンのダウンサンプリング	平均画素法を用い直接 210×160 から 84×84 に変換	210×160 から 110×84 に変換（詳細不明）
スクリーンのクロッピング	行わない	プレイエリアを含むように 84×84 をクロップ（詳細不明）
誤差の最小化法	AdaDelta[36]（$\gamma = 0.95$）	RMSProp[37]（パラメータ不明）
Q-learning の割引率	0.95	不明
Replay Memory のサイズ	500000	1000000
$t = 1, 2, 3$ での挙動	一様にランダムに行動を選ぶ	不明

図 4.7: Breakout に対し訓練した DQN の性能の変化

訓練中に一旦上がった平均スコアが後に大きく下がるという変化がいずれのゲームに関しても見られ，特に Enduro に関して顕著であるが，このような平均スコアの著しい不安定性は [4] においても報告されている．

以降の実験では，これら 3 つのゲームそれぞれについて，上記の 100 万イテレーションごとの評価の中で最も平均スコアの高かったものを訓練済み DQN として用いることにした．それらのスコアを表 4.5 に示す．

4.4.2 真のモデルを使った探索

訓練した DQN を元にモデル学習を行う前に，まずセーブ・ロード機能を使用した真のモデルを使用したマックス探索によってプレイヤの性能を向上することが可能かどうかを検証した．もし真のモデルを使用した場合に性能向上が確認できれば，十分に高い精度のモデルを学習することにより全く同じ方法によって性能向上を達成できると考えられる．逆に，もし真のモデルを使用しても性能向上が達成できないのであれば，学習したモデルを同じ方法で使用して性能向上は達成しづらいと予想される．

表 4.5: 本論文で訓練済み DQN として用いる DQN の性能

ゲーム	学習イテレーション数	平均スコア	[4] のスコア
Breakout	1000 万	204.8 ± 16.4	168
Enduro	400 万	313.9 ± 22.9	470
Pong	800 万	18.8 ± 0.3	20

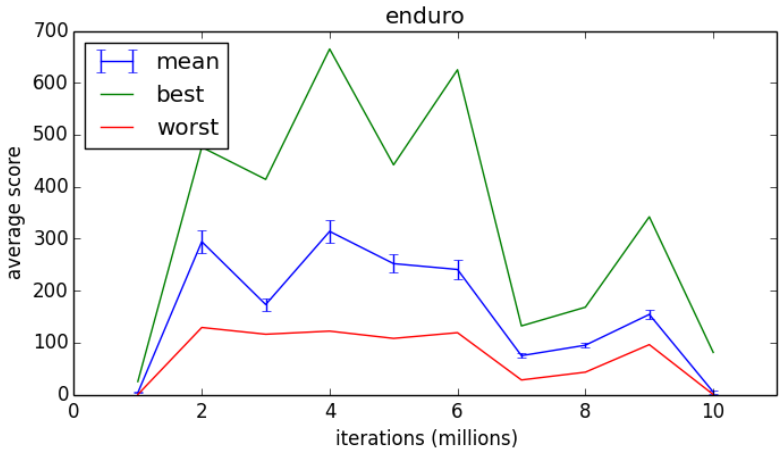


図 4.8: Enduro に対し訓練した DQN の性能の変化

真のモデル T, R, Ω を利用してマックス探索を行った場合のスコアを表 4.6 に示す．深さは 1 で固定し，探索単位は 1 及び 2 の場合についてスコアを計測した．探索単位 1 の場合では 3 つのゲームいずれにおいても性能改善は見られないばかりか，Enduro では全く有効なプレイングが行えなくなってしまった．

Enduro において探索単位 1 の場合に有効なプレイングが行えなくなってしまった理由としては，1 タイムステップ分の行動選択が直後の報酬信号や次のスクリーンの画像に全く影響を与えないという状態が Enduro に見られたということが挙げられる．深さ 1 かつ探索単位 1 の探索による行動評価は式（4.5）で与えられるが，もし $T(s,a)$，$R(s,a)$，$\Omega(s,a)$ の全てがそれぞれ a によらず同じであった場合，$Q^1_{T,R,\Omega}(s,a)$ も a によらず同じ値となる．その結果として全ての行動が同じ価値だと評価され，本研究の実装では決まった行動を選んでしまう．そうした場合に次の状態がまたこのような性質を持っていれば，常に決まった行動を採るというループに陥ってしまう．たとえ RAM の中身は行動により違う値となっていたとしても，DQN ではスクリーンの画像のみを入力として用いるため，スクリーン画像が同じであれば同じ評価値を返す．

一方，探索単位が 2 の場合では 3 つのゲーム全てで同等かそれを大きく上回るスコアを達成している．探索単位 1 の場合との先読みの差は 1/15 秒しか無いにも関わらずこのような差が生まれるの

表 4.6: 真のモデルを利用してマックス探索を行った場合のスコア

ゲーム	探索なし	$d=1, n=1$	$d=1, n=2$
Breakout	204.8 ± 16.4	195.9 ± 18.4	$\mathbf{264.9 \pm 17.1}$
Enduro	313.9 ± 22.9	0.0 ± 0.1	$\mathbf{680.8 \pm 39.0}$
Pong	$\mathbf{18.8 \pm 0.3}$	18.1 ± 0.3	18.75 ± 0.3

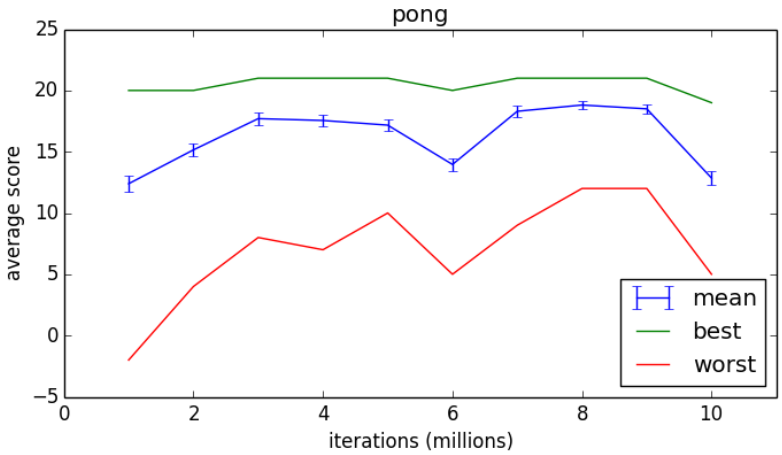


図 4.9: Pong に対し訓練した DQN の性能の変化

も，1 タイムステップの行動選択の影響が報酬や次のスクリーンの画像にすぐには現れない一方で，2 タイムステップの行動ならばその影響が現れるようになり，その結果として行動ごとに異なった評価を与えることが可能になっているためだと考えられる．

4.4.3 DQN に基づいたモデルの学習

前節の結果から，真のモデルを利用してマックス探索を行った場合，探索単位の値が適切であれば性能を向上させることができる一方で，それが適切でなければ完全なモデルを用いたとしてもむしろ性能を悪化させてしまうということが確認されたが，モデル学習により獲得されたモデルではそのモデルの不完全さによってさらに性能を悪化させてしまうおそれがある．不完全なモデルを利用した先読みを行いつつ，かつ DQN を単独で用いた場合と比較して性能が極端に悪化してしまうことを防ぐために，ここで

$$Q(s,a) = \lambda Q_{DQN}(s,a) + (1-\lambda) Q^{d}_{T^{n}_{\mathcal{H}}, R^{n}_{\mathcal{H}}, \Omega^{n}_{\mathcal{H}}}(s,a) \tag{4.10}$$

のように DQN により直接の行動価値の評価とマックス探索による評価の間のバランスをとるためのパラメータ λ を導入し，この値を変化させてどのように性能が変わるかの調査も行った．$\lambda = 0$ の場合は学習したモデルによるマックス探索による評価そのものであり，$\lambda = 1$ の場合は元の DQN による評価そのものである．

モデル学習は Algorithm 4.2 の擬似コードに従って行い，Replay Memory の容量は 50 万，総イテレーション数は 100 万という設定で，Breakout，Enduro，Pong の 3 つのゲームに対し，Hallucinated Replay の割合 $H = 0, 8, 16$ の 3 パターンそれぞれの学習を行った．そうして学習されたモデルを使

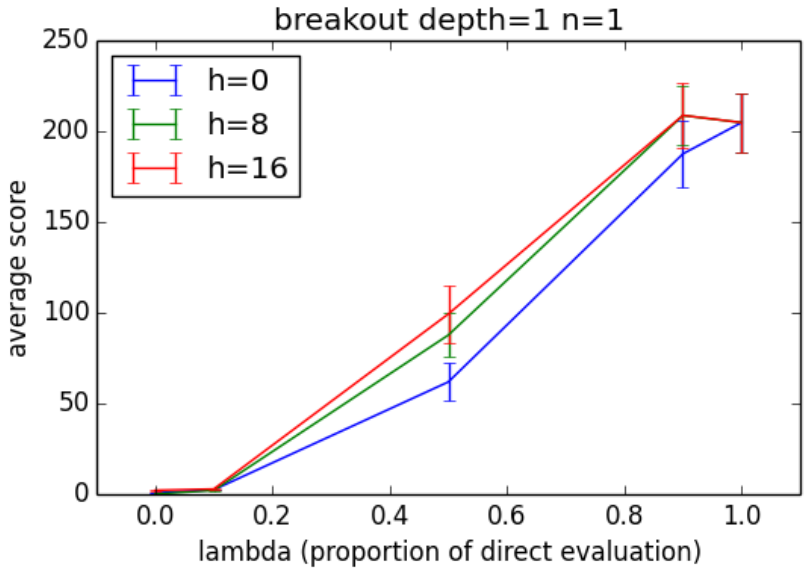


図 4.10: 学習したモデルを使用し，深さ 1，探索単位 1 の先読みを行った場合の平均スコア（Breakout）

用してマックス探索を行い，DQN による直接評価の割合 λ を変えながら各設定において 100 ゲームの平均スコアを測定した．

Breakout における結果を図 4.10，図 4.11，図 4.12，図 4.13，Enduro における結果を図 4.14，図 4.15，図 4.16，図 4.17，Pong における結果を図 4.18，図 4.19，図 4.20，図 4.21 に示す．

Breakout の結果を見ると，まず $H = 0$ のもの，すなわち Hallucinated Replay を使用しないものは，1 ステップのみ先読みを行う場合（深さ 1 かつ探索単位 1 の場合のみ）では他の設定とそれほど変わらないスコアを獲得しているが，2 ステップ以上の先読みを行う場合ではスコアが大きく落ちており有効な先読みが行えていないことが読み取れる．$H = 8$ 及び $H = 16$ の場合ではそのような大きな性能の落ち込みは確認できず，$H = 8$ と $H = 16$ では若干 $H = 16$ の方がスコアが高いが，かなり近い性能を示している．

λ の値の影響について見てみると，深さ 1 かつ探索単位 1 または 2 の結果では平均スコアとしては $\lambda = 0.9$，すなわち DQN による直接評価の割合が 0.9，先読みによる評価の割合が 0.1 の場合の評価が最も高い平均スコアを達成しているが，$\lambda = 1$，すなわち先読みをせずに直接 DQN により行動を評価する場合との差は有意ではない．深さ 1 かつ探索単位 3，深さ 2 かつ探索単位 1 の場合についても $\lambda = 0.9$ と $\lambda = 1$ の差ははっきりしない．しかし $\lambda \leq 0.5$ では 4 つの探索の設定全てにおいてはっきりと性能が低下しているのが確認でき，先読み評価の性能への悪影響の方が勝ってしまっている．

次に Enduro の結果を見てみると，Breakout の結果と異なり性能のピークが $\lambda = 0.5$ に見られる．深さ 1 かつ探索単位 1 または 2 の場合では特に先読みによる性能向上の割合が大きい．Breakout ほ

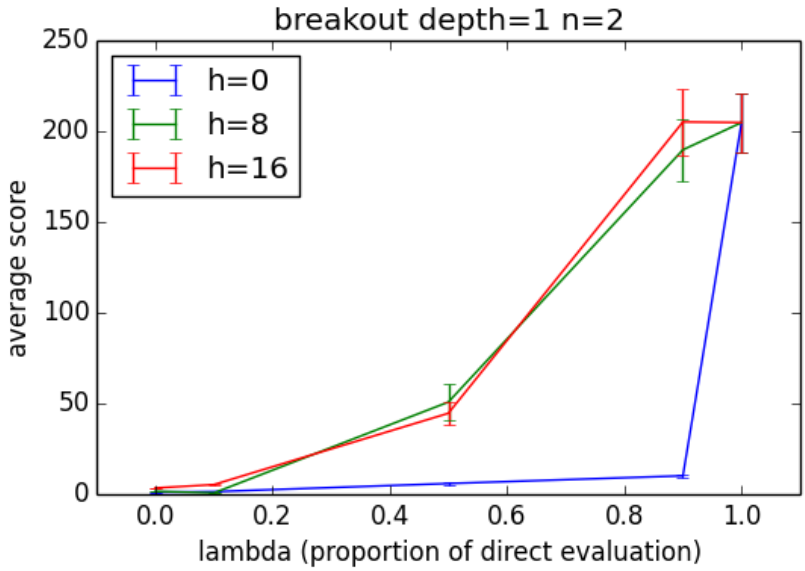


図 4.11: 学習したモデルを使用し，深さ 1，探索単位 2 の先読みを行った場合の平均スコア（Breakout）

ど顕著ではないが，Enduro でも $H = 0$ のモデルは先読みのステップ数が増えると性能が低下してしまうのが確認できる．$H = 8$ と $H = 16$ ではここでは総じて $H = 8$ の方が良く，特に深さ 1 かつ探索単位 3 の場合においては $H = 16$ は $H = 0$ と同様に性能が大きく低下してしまっているのに対し，$H = 8$ では元の DQN より高い性能を維持している．ただし $\lambda < 0.5$ ではいずれのモデルにおいても，程度の差はあれ元の DQN に比べて性能が低下してしまっている．

最後に Pong の結果を見ると，性能のピークは $\lambda = 0.9$ にあるが，$\lambda = 0.5$ でも性能がそれほど劣化していないのが確認できる．$H = 0$ のモデルに関しては，やはり先読みのステップ数が増えるのに従い性能が大きく低下してしまっていることが，深さ 1 で探索単位 1，2，3 の結果を見比べるとはっきり見て取れる．$H = 8$ と $H = 16$ の優劣に関しては，ここでも $H = 8$ 総じて良く，特に先読みのステップ数が増えると $H = 16$ との差が広がる．Pong のような単純なゲームであっても $\lambda < 0.5$ ではやはり大きな性能劣化が起きており，モデルの正確さはそれほど高くないことが予想される．

3 つのゲームの結果を総合すると，Hallucinated Replay を用いない $H = 0$ のモデルは先読みのステップ数が増えると大きく性能が劣化してしまうが，$H = 8, 16$ の場合，すなわち Hallucinated Replay を部分的に用いればそのような劣化は抑えることができるということが確かめられた．$H = 8$ と $H = 16$ のいずれが良いかに関しては，$H = 8$ の方が安定して高い性能を示しているように見えるが，Breakout では若干 $H = 16$ の方が高く，どれくらいの割合 Hallucinated Replay を持ちるのが良いかに関してはさらなる検証を要する．いずれのゲームにおいても $\lambda = 0.0, 0.1$ の場合は元の DQN に比べて性能が劣化してしまい，$\lambda = 0.0$ であっても性能向上を達成できた真のモデルの場合

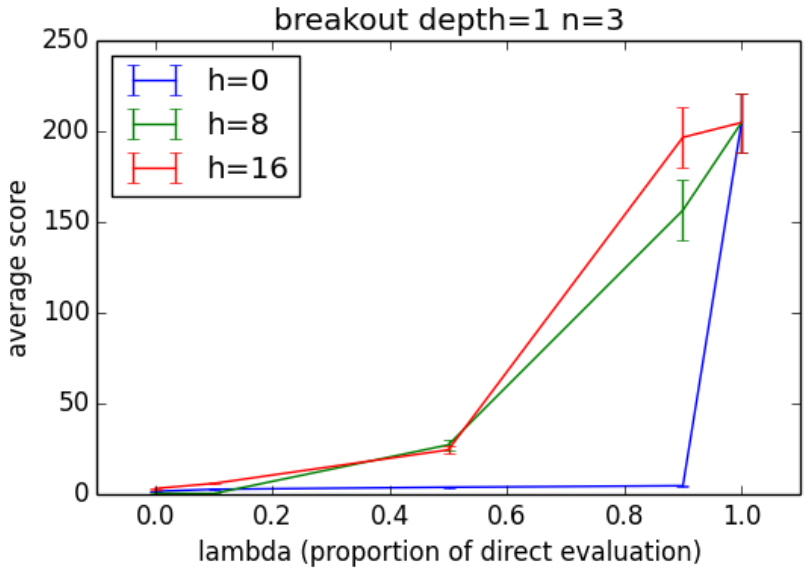


図 4.12: 学習したモデルを使用し，深さ 1，探索単位 3 の先読みを行った場合の平均スコア（Breakout）

と比べると大きく見劣りする結果だと言える．

4.5 まとめ

この章では，多様なビデオゲームを現在のスクリーンの画像とスコアの増減を表す報酬信号のみを元に上手くプレイすることを目指す Arcade Learning Environment（ALE）というフレームワークにおいて，深層ニューラルネットワークにより行動価値関数 $Q(s, a)$ を近似する Deep Q-Networks（DQN）という最先端のアルゴリズムを発展させ，訓練済みの DQN を使用してモデル学習を行う方法を提案した．深層ニューラルネットワークの内部表現を特徴空間として捉えることでその特徴抽出力をモデルの表現に活かすとともに，その特徴空間の上でモデルを定義することにより，入力・出力ともに低次元なモデルを学習することができる．

モデルのパラメータの最適化方法として，DQN の内部表現をそのまま近似するように学習を行う際に生じうる問題点を指摘し，それを回避する方法として，DQN の出力である各行動の行動価値について誤差を最小化するという方法を提案した．また複数ステップの先読みを可能にするために Hallucinated Replay という技術をミニバッチ学習に適用する方法を提案した．

以上の提案もとに，まず独自に DQN を実装・訓練した上で，実際にモデルの学習を行い，3 つのゲームについて平均スコアを測定したところ，1 つのゲームではモデル学習を行わない場合と比較して最大で約 4/3 倍という性能向上が確認できた．また複数ステップの先読みを行う際に Hallucinated

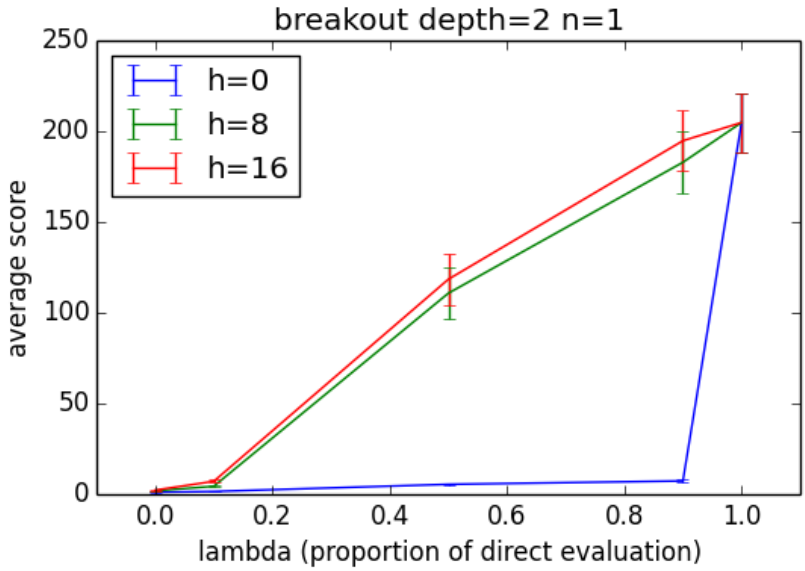


図 4.13: 学習したモデルを使用し，深さ 2，探索単位 1 の先読みを行った場合の平均スコア（Breakout）

Replay を行うか否かによってプレイヤの性能に大きな差が生じることも確認された．

4.6 今後の課題

今後の課題としては，まず DQN をよりよく訓練するための方法を探ることがあげられる．Breakout では元論文より高い平均スコアの DQN が訓練できたものの，その他のゲームでは元論文より若干低いスコアしか得られていないか，あるいはそもそも有効に学習を行うことができていない．元論文の実装に未だ不明な点が数多くあることも原因の一つではあるが，よりよい DQN の学習方法を探ることは重要だと考える．DQN がよりよく訓練できるようになれば，それは DQN に基づくモデルの性能向上にも繋がる可能性が高い．

また，学習したモデルを使用した浅い探索による評価結果では，一部のゲームでは性能向上を達成したものの DQN による直接評価にも大きく依存しており，DQN による直接評価なしでモデルによるマックス探索のみの場合では総じて低いスコアになってしまった．真のモデルを使用した場合には，は深さ 1，探索単位 2 のマックス探索で，DQN による直接評価を加えなくとも大幅な性能向上を達成できており，この違いはモデルから生じる誤差によるものだと考えられる．真のモデルと本章で学習したモデルとの差は大きく，より正確なモデルを学習する方法について検証を進める必要がある．今のところ学習できるモデルはモンテカルロ木探索のようなシミュレーションベースの探索に使用できるほど正確では無いと思われるが，十分に正確なモデルさえあればそのようなアプ

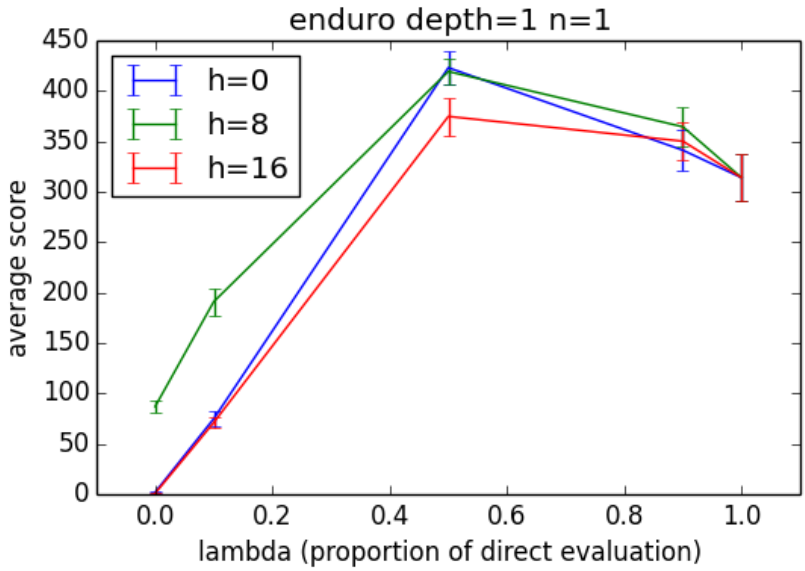


図 4.14: 学習したモデルを使用し，深さ 1，探索単位 1 の先読みを行った場合の平均スコア（Enduro）

ローチは試みる価値があると考える．

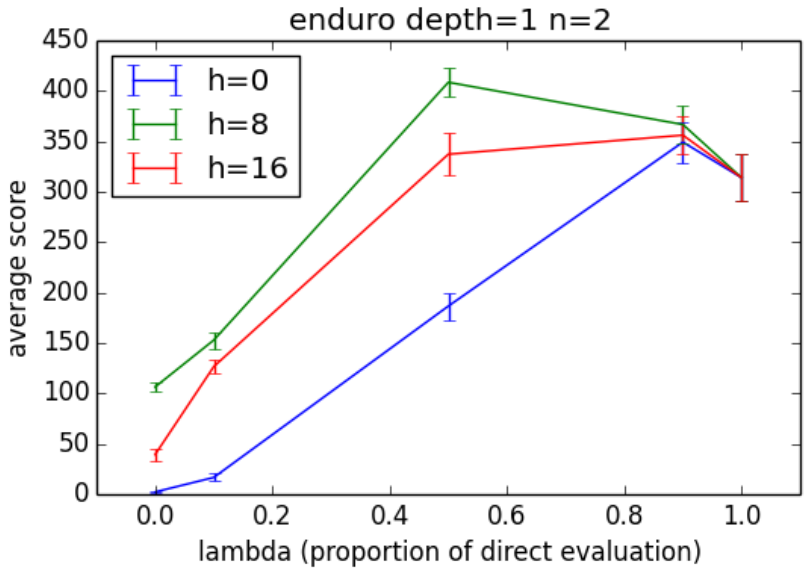


図 4.15: 学習したモデルを使用し，深さ 1，探索単位 2 の先読みを行った場合の平均スコア（Enduro）

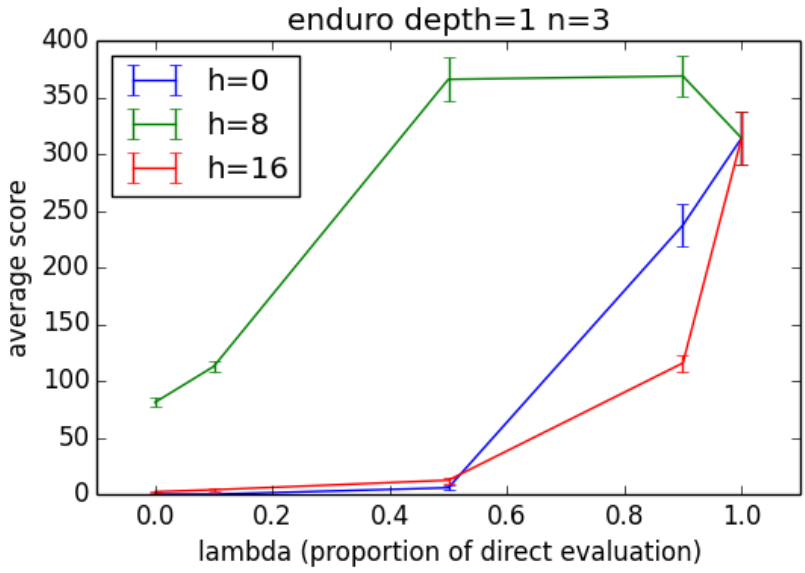


図 4.16: 学習したモデルを使用し，深さ 1，探索単位 3 の先読みを行った場合の平均スコア（Enduro）

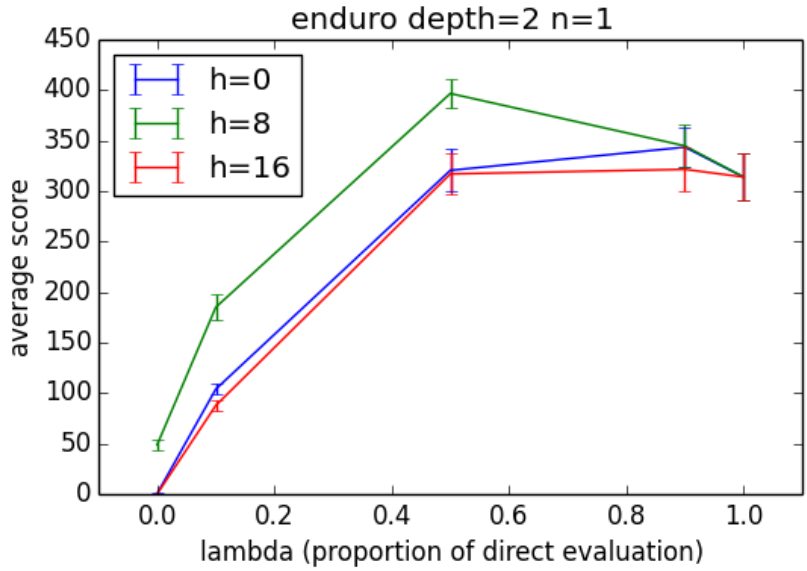


図 4.17: 学習したモデルを使用し，深さ 2，探索単位 1 の先読みを行った場合の平均スコア（Enduro）

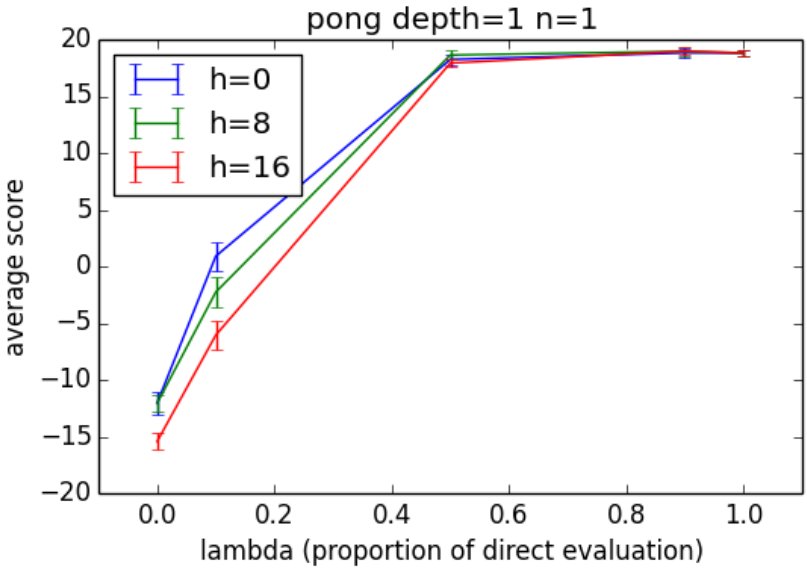


図 4.18: 学習したモデルを使用し，深さ 1，探索単位 1 の先読みを行った場合の平均スコア（Pong）

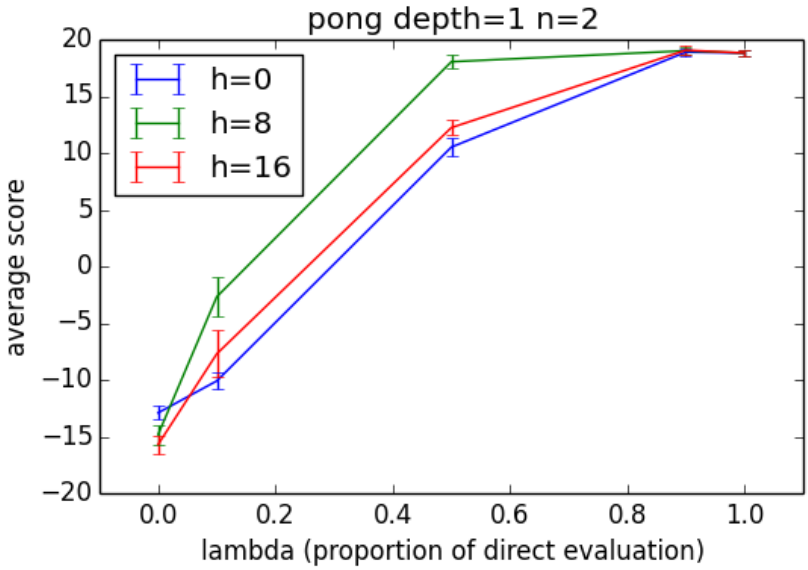


図 4.19: 学習したモデルを使用し，深さ 1，探索単位 2 の先読みを行った場合の平均スコア（Pong）

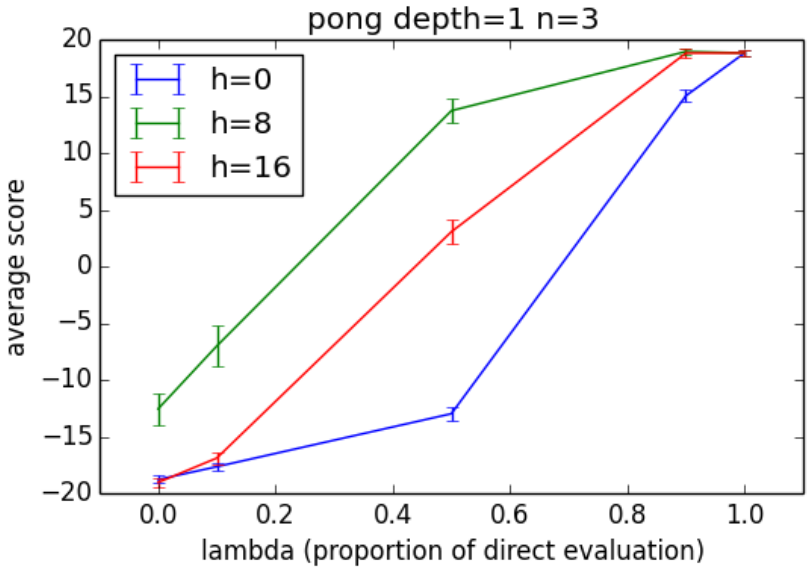


図 4.20: 学習したモデルを使用し，深さ 1，探索単位 3 の先読みを行った場合の平均スコア（Pong）

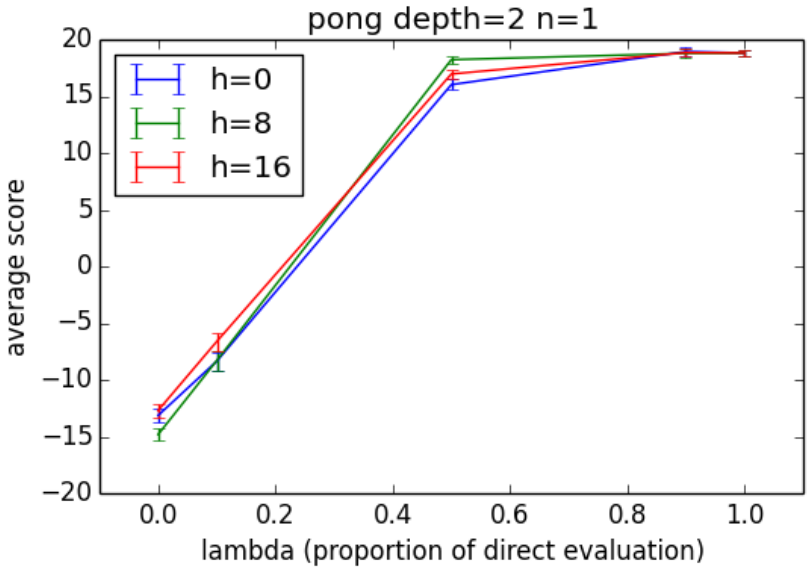


図 4.21: 学習したモデルを使用し，深さ 2，探索単位 1 の先読みを行った場合の平均スコア（Pong）

付 録A　General Game Playingのゲーム

A.1　どうぶつしょうぎ

図 A.1: どうぶつしょうぎの盤面

どうぶつしょうぎは 3 × 4 の盤面を用いる簡易版の将棋である．プレイヤは毎ターン自分の駒を 1 つ動かすか，あるいは捕まえた駒を空いている場所に置くことができる．駒の種類はライオン，ぞう，きりん，ひよこの 4 種類であり，それぞれ周囲 8 マス，斜めに接した 4 マス，上下左右の 4 マス，前方の 1 マスに動かすことができる．ひよこは相手の陣地（相手側の端の 1 行）に入るとにわとりになり，斜め後ろ以外の 6 マスに動けるようになる．相手のライオンを捕まえるか，相手の陣地に自分のライオンが入り，直後に捕まえられることがなければ勝ちとなる．

A.2　Connect Four

プレイヤは毎ターン交互に 6 × 8 の重力を持った空間に上から列を選んでディスクを 1 つ落とす．ディスクは下から積み重なっていき，先に 4 つ直線上に自分のディスクを並べたプレイヤの勝ちとなる．

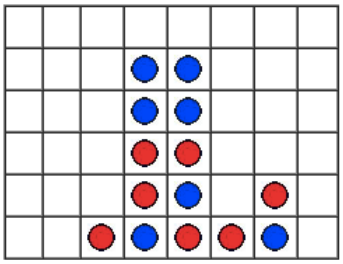


図 A.2: Connect Four の盤面

A.3 Nine Board Tic-Tac-Toe

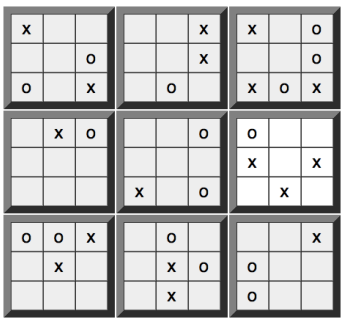


図 A.3: Nine Board Tic-Tac-Toe の盤面

Tic-Tac-Toe の 3×3 の盤面をさらに 3 の形に並べて用いる．プレイヤは毎ターンまだマークされていないマスを 1 つ選びマークする．最初にマークする盤面は 9 個のうちどれでもよいが，その次からは相手が直前にマークしたマスの盤面内の位置によって次に自分がマークできる盤面が決まる．例えば相手がある盤面内で左上に位置するマスにマークした場合，自分が次にマークするマスは左上の盤面から選ばなければならない．先にいずれかの盤面内で 3 つ直線上にマークしたプレイヤの勝ちとなる．

A.4 Breakthrough

プレイヤはそれぞれ 6×6 の盤面上の自分側の 2 行に自分の駒を有しており，毎ターン自分の駒を 1 つ，前（空きマスの場合のみ）か斜め前（相手の駒があってもよい）に 1 マスだけ動かすことができる．移動先に相手の駒がある場合はそれを捕まえることができる．先に自分のいずれかの駒を相手側の端の 1 行まで移動させたプレイヤの勝ちとなる．8×8 のものが一般的であるが，評価の際には 6×6 のものを用いた．

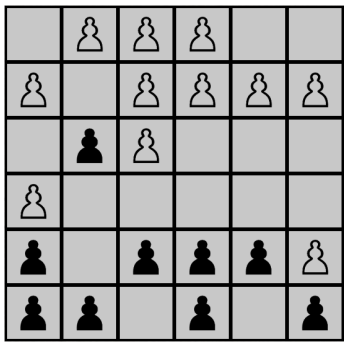


図 A.4: Breakthrough の盤面

A.5 TTCC4

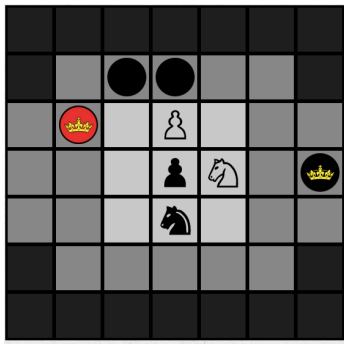


図 A.5: TTCC4 の盤面

TTCC4 は Tic Tac Chess Checkers 4 の略であり，チェス，チェッカー，Tic-Tac-Toe，Connect Four を 4 つのゲームを組み合わせたゲームである．各プレイヤはチェスのポーンとナイト及びチェッカーのキングを 1 つずつ有しており，毎ターンそれらの 1 つを動かすか，あるいは中央の 3 列で上から Connect Four のようにディスクを落とすことができる．中央の 3×3 のエリア内で先に 3 つ直線上に自分の駒を置いたプレイヤの勝ちとなる．

付 録B　Arcade Learning Environmentのゲーム

B.1　Breakout

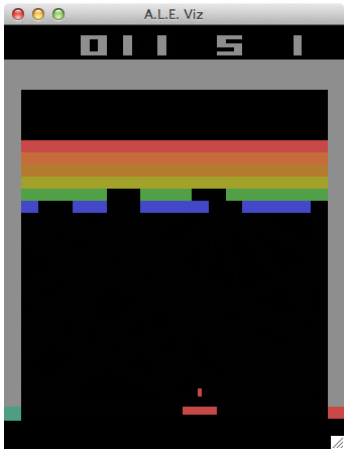


図 B.1: Breakout の画面

Breakout はいわゆるブロック崩しゲームである．画面上部にブロックの層があり，プレイヤーはパドルを動かしてボールを跳ね返し，ブロックに当てることによってブロックを崩すことができる．ブロックを崩す度にスコアが加算される．

B.2　Enduro

Enduro はレースゲームであり，各ステージにおいて予め決められた数だけ他の車だけ車をを抜き去ることで次のステージに進むことができる．最初のステージでは 200 の車を，それ以降のステージでは 300 の車を抜き去る必要がある．同じステージ内でも時間が経過するとコースの天候が変化する．図 B.2 の左はステージを開始してすぐの場面である．車を 1 台追い越すごとに 1 点のスコアを得ることができる．

図 B.2: Enduro の画面．左は開始直後，右は時間が経過し天候が変化した状態．

B.3 Pong

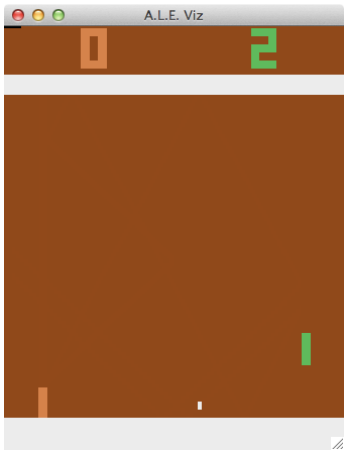


図 B.3: Pong の画面

Pong は Video Olympics というゲーム集の中に含まれるゲームである．プレイヤは右の緑のパドルを動かしてボールを跳ね返し，左端に到達させることで 1 点を得ることができる．21 点を先取した方の勝利となる．

参考文献

[1] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[2] Alborz Geramifard, Christoph Dann, and Jonathan P. How. Off-Policy Learning Combined with Automatic Feature Expansion for Solving Large MDPs. *The 1st Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2013.

[3] David Silver, Richard S. Sutton, and Martin Müller. Temporal-difference search in computer Go. *Machine Learning*, Vol. 87, No. 2, pp. 183–219, February 2012.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. In *NIPS 2014 Deep Learning Workshop*, pp. 1–9, 2013.

[5] Michael Genesereth and Nathaniel Love. General game playing: Overview of the AAAI competition. *AI magazine*, Vol. 26, No. 1, pp. 1–16, 2005.

[6] Jacques Pitrat. Realization of a general game-playing program. In *IFIP congress (2)*, pp. 1570–1574, 1968.

[7] Marc C. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, Vol. 47, pp. 253–279, 2013.

[8] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. No. September. University of Cambridge, Department of Engineering, 1994.

[9] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.

[10] Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan P. How. Online discovery of feature dependencies. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pp. 881–888, 2011.

[11] Alborz Geramifard, Thomas J. Walsh, Nicholas Roy, and Jonathan P. How. Batch-iFDD for Representation Expansion in Large MDPs. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 242–251, 2013.

[12] Bruce Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, pp. 182–193, 1990.

[13] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. *Computers and games*, Vol. 4630, pp. 72–83, 2007.

[14] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 1, No. 1, pp. 73–89, 2009.

[15] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European conference on Machine Learning*, pp. 282–293, 2006.

[16] Peter Auer, N Cesa-Bianchi, and P Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, pp. 235–256, 2002.

[17] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, pp. 273–280, 2007.

[18] James Clune. Heuristic evaluation functions for general game playing. In *Proceedings of the Twenty-Second Conference of Artificial Intelligence*, pp. 1134–1139, 2007.

[19] Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 1191–1196, 2007.

[20] Hilmar Finnsson. *Cadia-player: A general game playing agent.* PhD thesis, Reykjavik University, 2007.

[21] Mehat Jean and Tristan Cazenave. Ary , a general game playing program. In *Thirteenth Board Games Studies Colloquim*, 2010.

[22] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification, 2006.

[23] Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pp. 994–999, 2010.

[24] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pp. 259–264, 2008.

[25] Hilmar Finnsson and Yngvi Björnsson. Simulation control in general game playing agents. In *the IJCAI-09 Workshop on General Game Playing*, pp. 954–959, 2009.

[26] Mandy J.W. Tak, Mark H.M. Winands, and Yngvi Björnsson. N-grams and the last-good-reply policy applied in general game playing. *IEEE Transactions on Computational Intelligence and AI in Games, Volume 4*, pp. 73–83, 2012.

[27] Shiven Sharma, Ziad Kobti, and Scott Goodwin. Learning and knowledge generation in general games. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pp. 329–335. IEEE, 2008.

[28] Jacek Madziuk and Maciej Świechowski. Generic Heuristic Approach to General Game Playing. *Lecture Notes in Computer Science*, Vol. 7147, pp. 649–660, 2012.

[29] Karol Waledzik and Jacek Mandziuk. An Automatically-Generated Evaluation Function in General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, No. c, pp. 1–1, 2013.

[30] Stephan Schiffel and Yngvi Bjornsson. Efficiency of GDL Reasoners. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 6, No. 4, pp. 343–354, 2014.

[31] Long-Ji Lin. *Reinforcement learning for robots using neural networks.* PhD thesis, Carnegie Mellon University, 1993.

[32] Marc G. Bellemare, Joel Veness, and Michael Bowling. Investigating Contingency Awareness Using Atari 2600 Games. In *the Twenty-Sixth AAAI Conference on Artificial Intelligence*, pp. 864–871, 2012.

[33] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. *Advances in Neural Information Processing Systems (NIPS)*, Vol. 2600, pp. 1–9, 2014.

[34] Marc Peter Deisenroth and Carl Edward Rasmussen. PILCO : A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472, 2011.

[35] Erik Talvitie. Model Regularization for Stable Sample Rollouts. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

[36] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv:1212.5701*, December 2012.

[37] Tijmen. Tieleman and Geoffrey Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

発表文献

1. 藤田 康博, 鶴岡 慶雅, 伊庭 斉志. General Game Playing のための組み合わせ特徴の自動生成. ゲームプログラミングワークショップ 2014 論文集,2014,180-187. (研究奨励賞受賞)

謝辞

本研究を進めるにあたり多くの方にお世話になりました．指導教員である伊庭斉志教授には，この研究室に来てからというもの輪講の際の自分の不準備などで絶えずご迷惑・ご心配をおかけしてきたように思いますが，これまで温かく見守っていただき，また研究テーマから何まで自由にやらせていただいたことに感謝しております．TA やチューターとしての仕事を与えていただいたことも大学院生活の励みになりました．鶴岡研究室の鶴岡慶雅准教授には，同研究室で行われる AI ミーティングへの参加を許可していただきましたが，そこでの議論はこれまでゲーム AI の研究を続ける上で不可欠でした．ミーティング以外の時間でも，いつでも快く研究に関する相談にのっていただいたことに心より感謝致します．AI ミーティングの場で多数のご指摘を頂いたり，共に議論を行わせて頂いてきた，元近山・鶴岡研究室の同期である橋本和真君，亀甲博貴君，その他の鶴岡研究室の皆様にもお世話になりました．豊田工業大学知能数理研究室の三輪誠准教授には，研究の方向性が定まらずに悩んでいた修士 1 年目のうちに何度も相談に乗っていただきました．特に，本論文の後半の基礎となっている Deep Q-Networks の論文の存在について早い段階で教えていただいたことは，それからの自分の研究の方向性を決める転機となるものでした．長谷川禎彦講師には輪講の発表資料の添削等，研究を進める上での指導をしていただいただけでなく，研究室の空気をより活気のあるものとするために努力していただいたことを大変ありがたく思っております．土肥浩助教には研究室のネットワークや計算機資源の管理の面でとてもお世話になり，また頼りにさせて頂きました．研究室で同じ時間を過ごし，様々な議論を行ってきた先輩，同輩，後輩の皆さんにも感謝の意を表します．